

MPLAB Harmony Applications Help

MPLAB Harmony Integrated Software Framework

© 2013-2018 Microchip Technology Inc. All rights reserved.

Volume I: Getting Started With MPLAB Harmony Libraries and Applications

This volume introduces the MPLAB® Harmony Integrated Software Framework.

Description

MPLAB Harmony is a layered framework of modular libraries that provide flexible and interoperable software "building blocks" for developing embedded PIC32 applications. MPLAB Harmony is also part of a broad and expandable ecosystem, providing demonstration applications, third-party offerings, and convenient development tools, such as the MPLAB Harmony Configurator (MHC), which integrate with the MPLAB X IDE and MPLAB XC32 language tools.



Legal Notices

Please review the *Software License Agreement* prior to using MPLAB Harmony. It is the responsibility of the end-user to know and understand the software license agreement terms regarding the Microchip and third-party software that is provided in this installation. A copy of the agreement is available in the `<install-dir>/doc` folder of your MPLAB Harmony installation.

The OPENRTOS® demonstrations provided in MPLAB Harmony use the OPENRTOS evaluation license, which is meant for demonstration purposes only. Customers desiring development and production on OPENRTOS must procure a suitable license. Please refer to one of the following documents, which are located in the `<install-dir>/third_party/rtos/OPENRTOS/Documents` folder of your MPLAB Harmony installation, for information on obtaining an evaluation license for your device:

- `OpenRTOS Click Thru Eval License PIC32MXxx.pdf`
- `OpenRTOS Click Thru Eval License PIC32MZxx.pdf`

TIP!

Throughout this documentation, occurrences of `<install-dir>` refer to the default MPLAB Harmony installation path:

- Windows: `C:/microchip/harmony/<version>`
- Mac OS/Linux: `~/microchip/harmony/<version>`

Applications Help

This section provides information on the various application demonstrations that are included in MPLAB Harmony.

Description

Applications determine how MPLAB Harmony libraries (device drivers, middleware, and system services) are used to do something useful. In a MPLAB Harmony system, there may be one main application, there may be multiple independent applications or there may be one or more Operating System (OS) specific applications. Applications interact with MPLAB Harmony libraries through well defined interfaces. Applications may operate in a strictly polling environment, they may be interrupt driven, they may be executed in OS-specific threads, or they may be written so as to be flexible and easily configured for any of these environments. Applications generally fit into one of the following categories.

Demonstration Applications

Demonstration applications are provided (with MPLAB Harmony or in separate installations) to demonstrate typical or interesting usage models of one or more MPLAB Harmony libraries. Demonstration applications can demonstrate realistic solutions to real-life problems.

Sample Applications

Sample applications are extremely simple applications provided with MPLAB Harmony as examples of how to use individual features of a library. They will not normally accomplish anything useful on their own. They are provided primarily as documentation to show how to use a library.

Audio Demonstrations

This section provides information on the Audio Demonstrations provided in your installation of MPLAB Harmony.

MPLAB Harmony is available for download from the Microchip website by visiting: http://www.microchip.com/mplabharmony. Once you are on the site, click the Downloads tab to access the appropriate download for your operating system. For additional information on this demonstration, refer to the “Applications Help” section in the MPLAB Harmony Help.

Introduction

MPLAB Harmony Audio Demonstrations Help.

Description

This help file contains instructions and associated information about MPLAB Harmony Audio demonstration applications, which are contained in the MPLAB Harmony Library distribution.

Demonstrations

This topic provides instructions about how to run the demonstration applications.

audio_microphone_loopback

This topic provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

In this demonstration application, the AK4642 or AK4954 Codec Driver sets up the codec, which is the on-board device on the Audio Codec Daughter Board AK4642EN, or AK4954A and can receive audio data through the microphone onboard the daughter board and sends this audio data out through the on-board headphones. Either the Audio Codec Daughter Board AK4642EN, or the Audio Codec Board AC324954, which are available for purchase from Microchip, can be connected to the PIC32 Bluetooth Audio Development Kit.

The demonstration application also shows how to configure the I2S client and DMA channels for applications that need two-way data communication between the Codec and the PIC32 microcontroller.

Refer to the *Framework Help* > *Driver Libraries Help* > *Decoder Libraries Help* > AK4642 (or AK4954) Codec Driver Library section for more information on the driver, including configuration details as well as the available APIs.

The DRV_CODEC_IO_INTENT_READWRITE mode of the codec driver is useful when the application can guarantee that the I2S playback buffers (even if it is zeros) are the same size and same sampling rate as the microphone (record) buffers that are received over I2S.

Architecture

This application runs on the PIC32 Bluetooth Audio Development Kit, which contains a PIC32MX470F512L microcontroller with 512 KB of Flash memory and 128 KB of RAM running at 96 MHz and uses the following additional feature of the PIC32 Bluetooth Audio Development Kit:

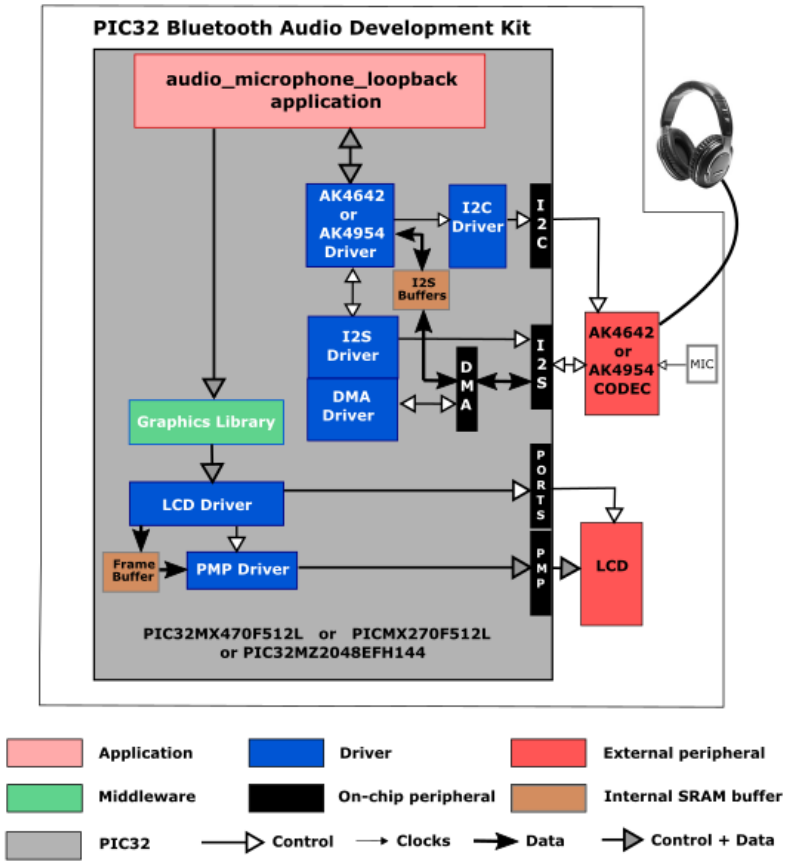
- 220x176 color LCD

Note: The PIC32 Bluetooth Audio Development Kit comes with a BTM805 Bluetooth Daughter Board and an AK4384 Audio DAC Daughter Board ,mounted on headers (J8/J11 and J9/J10); however, you must replace the AK4384 Audio DAC Daughter Board with either the PIC32 Audio Codec Daughter Board AK4642EN, which is sold separately on microchipDIRECT as part number AC320100, or the PIC32 Audio Codec Daughter Board AK4954A, which is sold separately on microchipDIRECT as part number AC324954..

Surrounding the PIC32MX470F512L microcontroller are a set of headers, which accept various plug-in modules (PIM), including one for a PIC32MX270F512L PIM (with 512 KB of Flash and 64 KB of RAM), and also a PIC32MZ2048EFH144 PIM (with 2 MB of Flash and 512 KB of RAM) as alternate configurations for this project.

The program takes up only about 27% (142K) of the PIC32MX470F512 microcontroller's program space, and 15% (20K) of its RAM. The two audio buffers take up 3200 bytes, with the rest largely used by graphics. The heap uses an additional 8K, with its size based on the graphics taking up about half of that.

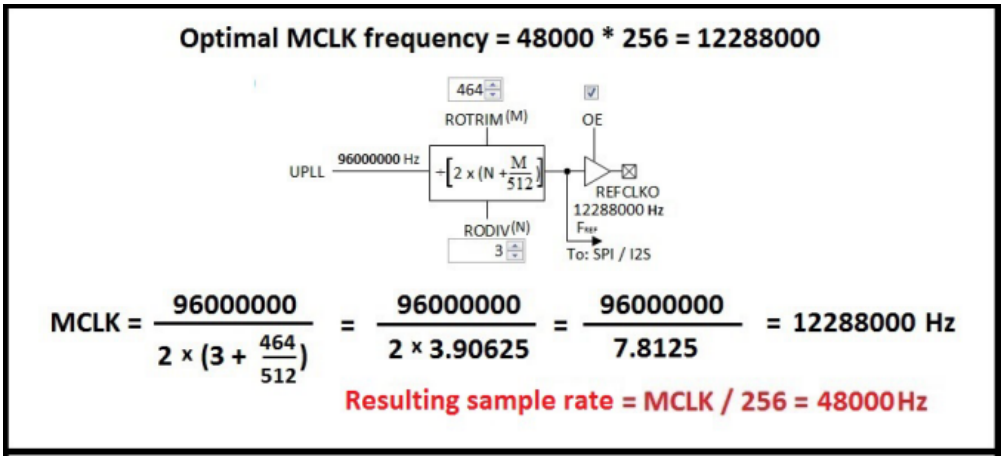
The following figure illustrates the application architecture.



The PIC32 microcontroller (MCU) runs the application code, and communicates with the codec using an I2C interface.

The interface between the PIC32 MCU and the LCD is an 8-bit parallel master port (PMP). The program uses the MPLAB Harmony v2.x Graphics Library to draw on the screen.

As with any MPLAB Harmony application, the SYS_Initialize function, which is located in the `system_init.c` source file, makes calls to initialize various subsystems, such as the clock, ports, board support package (BSP), PMP, codec, graphics, and interrupts.



The codec driver, graphics, and the application state machines are all updated through calls located in the SYS_Tasks function in the `system_tasks.c` file. Interrupt handlers in the `system_interrupt.c` file are used for the two DMA channels, and the I2C driver which controls the codec.

The application code is contained in the source file `app.c`, which contains a state machine (APP_Tasks). It first initializes the application, then receives a handle to the codec driver by calling the codec's DRV_CODEC_Open function with a mode of DRV_IO_INTENT_READWRITE. Then it registers an event handler, APP_CodecBufferEventHandler as a callback with the codec driver. Finally it makes an initial call to the codec's DRV_CODEC_BufferAddWriteRead is done to get things started (writing zeros to the output for the first buffer only).

The event handler callback is given control whenever a buffer has been processed. Two 400 word (uint16_t) buffers, micbuf1 and micbuf2, are toggled back and forth in a ping-pong fashion (one used for reading, one used for writing) so that when a buffer is filled on the receive side (from the microphone on the codec daughter board), it is turned right around and sent back as output to the transmit side and played through the headphones.

Although the codec driver interfaces with the codec via I2S and DMA, which is largely transparent to the application, it is all taken care of by MPLAB Harmony.

As a minimum, the buffers must be large enough so that there is enough time when the event handler callback is handled to process the swapping of the buffer pointers, plus a call to BufferAddWriteRead. At 48000 samples per second, a buffer with 400 entries will result in a callback every 8.3 ms. This could probably be reduced down to as low as 50 entries, or about 1 ms and still work. However, there would not be much time in the application to do anything else. If a lot of graphics processing was added to this application, or other work not associated with the audio processing, the buffer size might even need to be increased to 800 (16.7 ms) or even more.

Demonstration Features

- Uses the AK4642 or AK4954 Codec Driver Library to read audio samples from the built-in microphone on the codec daughter board, and send audio back to the headphone output
- I2S audio output/speaker driver
- I2S microphone driver
- At a lower level, using the I2S Driver Library between the codec library and the codec for audio, and the I2C Driver Library for control of the codec
- Use of "ping-pong" audio buffers
- Simultaneous I2S data
- Displays graphics on the 220x176 LCD of the PIC32 Bluetooth Audio Development Kit using the MPLAB Harmony Graphics Library (see MPLAB Harmony Graphics Composer Suite) and the Parallel Master Port (see PMP Driver Library)

Tools Setup Differences

When building a new application, start by creating a 32-bit MPLAB Harmony project in MPLAB X IDE by selecting *File > New Project*. Select the desired processor (PICMX470F512L or PIC32MX270F512L), and the PIC32MX Bluetooth Audio Development Kit. Click the MPLAB Harmony icon to start the MPLAB Harmony Configurator (MHC). Under BSP Configuration, select PIC32 Bluetooth Audio Development Kit (AK4384) or (AK4954), or PIC32MX270F512L with Bluetooth Audio Development Kit (AK4384), as appropriate for your hardware.

Under *Drivers >CODEC*, select **Use Codec AK4642? or Use Codec AK4954?**. Under *I2S*, Use I2S Driver should already be checked. Expand that option and select **DMA Mode**, and then also select **Transmit DMA Support** and **Receive DMA Support**. Under *Audio Communication Width*, select SPI_AUDIO_COMMUNICATION_16DATA_16FIFO_32CHANNEL. Under Audio Protocol Mode, select DRV_I2S_LEFT_JUSTIFIED if using the AK4642, or DRV_I2S_AUDIO_I2S if using the AK4954. Change the Receive DMA Channel Instance to 1.

Under *System Services >DMA*, select **Use DMA System Services**. Change the Number of DMA Channel Instances to 2 and press **Enter**. Under *DMA Channel Instance 1*, change the DMA Channel to DMA_CHANNEL_2.

Note that the I2C Driver has been automatically selected and properly setup so you do not need to do anything further with it if using the AK4642. If using the Ak4954, check the box Include Force Write I2C Function.

If your application is going to be using graphics, as this one does, within *Graphics Stack*, select **Use Graphics Stack?**. Further down, within *Use Harmony Graphics Composer Suite? > Middleware > Use Aria User Interface Library?*, clear **Enable Touch** since the LCD on the PIC32 Bluetooth Audio Development Kit does not support a touchscreen. Since you are using the LCD, you will also need to go back to the Drivers section, and

within PMP, select **Use PMP Driver?**. Expand that section, and change the value of Strobe Wait Sates to PMP_STROBE_WAIT_4.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the demonstration.

Description

To build this project, you must open the `audio_microphone_loopback.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/audio/audio_microphone_loopback`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`audio_microphone_loopback.X`	`<install-dir>/apps/audio/audio_microphone_loopback/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
bt_audio_dk_16bit	bt_audio_dk	This demonstration runs on the PIC32MX470F512L mounted on the PIC32 Bluetooth Audio Development Kit and the Audio Codec Daughter Board AK4642EN.
pic32mx270f512l_pim_bt_audio_dk	pic32mx270f512l_pim+bt_audio_dk	This demonstration runs on the PIC32MX270F512L PIM mounted on the PIC32 Bluetooth Audio Development Kit and the Audio Codec Daughter Board AK4642EN.
pic32mz_ef_pim_bt_audio_dk	pic32mz_ef_pim+bt_audio_dk	This demonstration runs on the PIC32MZ2048EFH144 PIM mounted on the PIC32 Bluetooth Audio Development Kit and the Audio Codec Daughter Board AK4642EN.
bt_audio_dk_16bit_ak4954	bt_audio_dk+ak4954	This demonstration runs on the PIC32MX470F512L mounted on the PIC32 Bluetooth Audio Development Kit and the Audio Codec Daughter Board AK4954A.

Configuring the Hardware

This section describes how to configure the supported hardware.

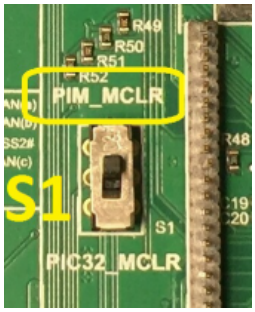
Description

PIC32 Bluetooth Audio Development Kit with PIC32MX270F512L PIM for Bluetooth Audio Development Kit and the Audio Codec Daughter Board AK4642EN (see **Note**)

Switch S1 on the PIC32 Bluetooth Audio Development Board (between the audio DAC daughter board and the microcontroller PIM) should be set to PIM_MCLR.

The PIC32 Bluetooth Audio Development Kit with PIC32MZ2048EFH144 Plug-in Module (PIM) and the Audio Codec Daughter Board AK4642EN (see **Note**)

Switch S1 on the PIC32 Bluetooth Audio Development Board should be set to PIM_MCLR.

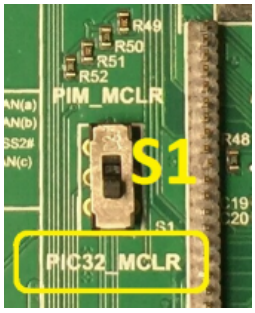


PIC32 Bluetooth Audio Development Kit with the PIC32MX470F512L and the Audio Codec Daughter Board AK4642EN (see **Note**).

Switch S1 on the PIC32 Bluetooth Audio Development Board (between the audio DAC daughter board and the microcontroller PIM) should be set to PIC32_MCLR.

PIC32 Bluetooth Audio Development Kit with the PIC32MX470F512L and the Audio Codec Daughter Board AK4954A (see **Note**).

Switch S1 on the PIC32 Bluetooth Audio Development Board (between the audio DAC daughter board and the microcontroller PIM) should be set to PIC32_MCLR.



Note: The PIC32 Bluetooth Audio Development Kit includes an Audio DAC Daughter Board; however, the Audio DAC Daughter Board must be replaced by the PIC32 Audio Codec Daughter Board AK4642 which is sold separately on micochipDIRECT as part number AC320100 , or the PIC32 Audio Codec Daughter Board AK4954A, which is sold separately on microchipDIRECT as part number AC324954.

Running the Demonstration

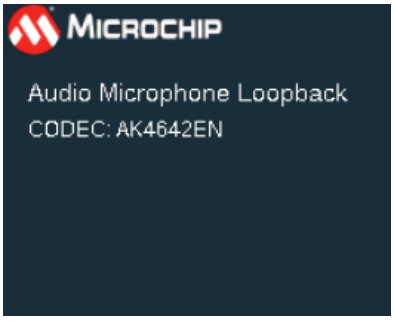
This section demonstrates how to run the demonstration.

Description

Important! Prior to using this demonstration, it is recommended to review the MPLAB Harmony Release Notes for any known issues. A PDF copy of the release notes is provided in the `<install-dir>/doc` folder of your installation.

Do the following to run the demonstration:

1. Compile and program the target device. While compiling, select the appropriate MPLAB X IDE project configuration based on the demonstration board. Refer to Building the Application for details.
2. Connect headphones to the HP OUT connector on the Audio Codec Daughter Board AK4642EN or AK4954A.
3. The on-board microphone (MIC3) will begin capturing surrounding audio and start looping it through the Codec to the microprocessor and back to the Codec headphones where you should be able to audibly observe the microphone input. An easy way to test this is to gently rub the microphone with your fingertip and listen for the resulting sound in your speaker or headphones.



Note: The screen is static; therefore, there are no user controls.

audio_tone

This topic provides instructions and information about the MPLAB Harmony Codec Driver demonstration application, which is included in the MPLAB Harmony Library distribution.

Description

In this demonstration application, depending on the configuration, the Codec Driver sets up the AK4384 DAC, AK4953 Codec, or AK4954 Codec. The demonstration sends out generated audio waveforms (sine tone and chirp) with parameters modifiable through on-board push buttons. Success is indicated by an audible output corresponding to displayed parameters.

The sine tone is of any frequency that is four times less than the sampling rate using a 32-bit fixed point algorithm. The tone can be continuously

modified in frequency so as to also generate a chirp waveform. A timer is used control the duration of the sine tone or chirp, based on displayed settings modified by the buttons.

To know more about the MPLAB Harmony Codec Drivers, configuring the Codec Drivers, and the APIs provided by the Codec Drivers, refer to Codec Driver Libraries.

Architecture

PIC32 Bluetooth Audio Development Kit Configurations:

Three of the configurations run on the PIC32 Bluetooth Audio Development Kit, which contains a PIC32MX470F512L microcontroller with 512 KB of Flash memory and 128 KB of RAM running at 96 MHz using the following features:

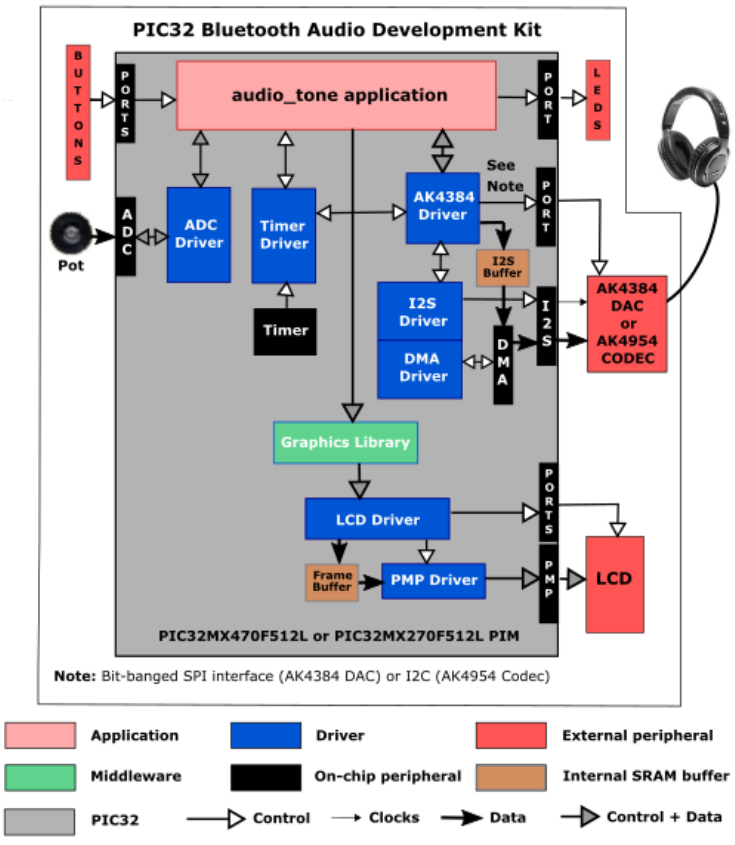
- 220x176 color LCD
- Five push buttons (SW1-SW5)
- Five LEDs
- PIC32 Audio AK 4384 DAC Daughter Board, or AK4954A Codec Daughter Board mounted on a X32 socket
- Potentiometer

The program takes up to approximately 32% (166K) of the PIC32MX470F512L microcontroller's program space. The 16-bit configuration uses 20% (25K) of the RAM (with 15K of that used by the three audio buffers), and the 24-bit configuration uses 32% (40K) of the RAM (with 30K used by the audio buffers). The rest is largely used by graphics. The heap uses an additional 16K, with its size based on the graphics taking up about 12K of that.

A fourth configuration also makes use of the PIC32 Bluetooth Audio Development Kit, but runs on a PIC32MX270F512L PIM attached to the board. It has 512 KB of Flash memory and 64 KB of RAM running at 48 MHz.

The program takes up to approximately 34% (173K) of the PIC32MX270F512L microcontroller's program space, and uses 39% (25K) of the RAM (with 15K of that used by the three audio buffers). The rest is largely used by graphics. The heap uses an additional 16K, with its size based on the graphics taking up about 12K.

The following figure illustrates the application architecture, for the four PIC32 Bluetooth Audio Development Kit configurations:



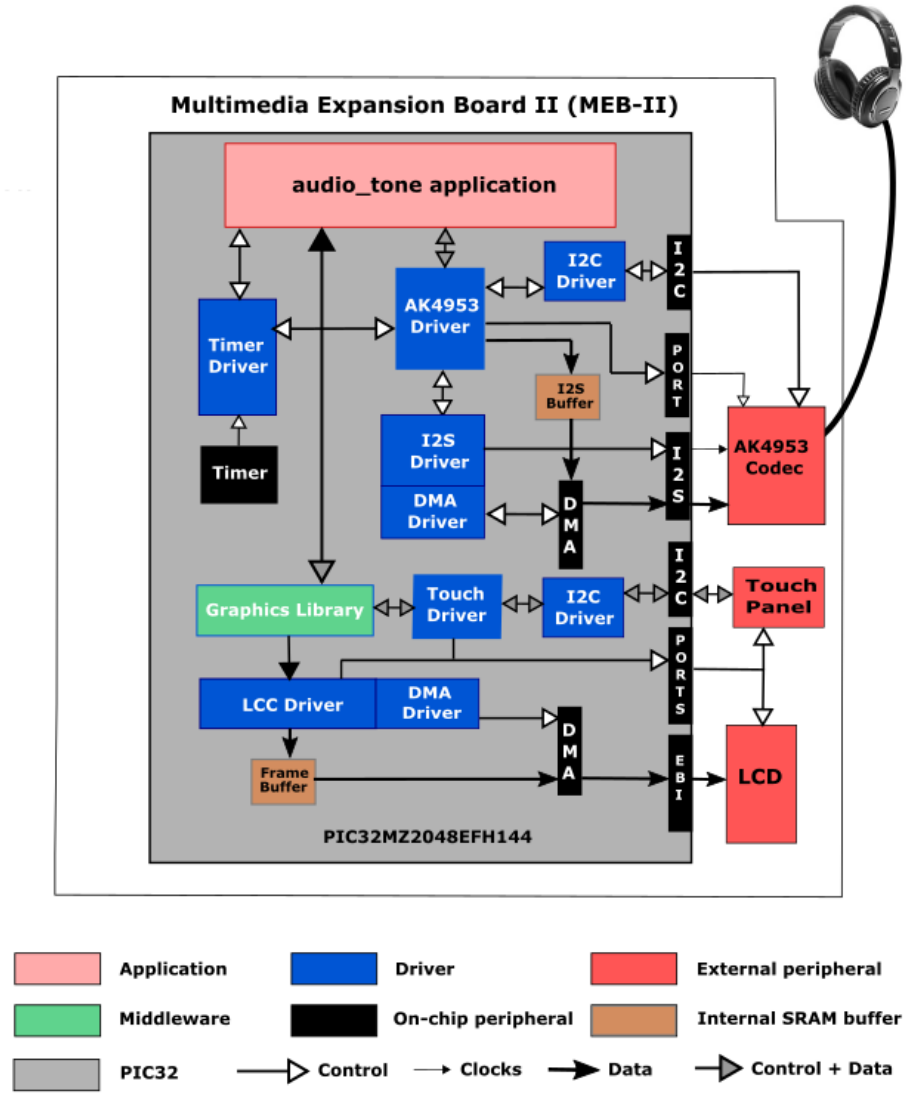
Multimedia Expansion Board II configuration:

This configuration runs on a PIC32MZ EF Starter Kit, mounted on a Multimedia Expansion Board II (MEB-II). The PIC32MZ2048EF microcontroller has 2 MB of Flash memory and 512 KB of RAM, running at 198 MHz. The program makes use of the following features on the MEB-II board:

- PDA 480x272 (WQVGA) MaxTouch LCD display daughter board
- PIC32 AK4953 Audio Codec

The program takes up to approximately 11% (220K) of the PIC32MZ2048EF microcontroller's program space. The 16-bit configuration uses 55%

(284K) of the RAM (only 15K of that used by the three audio buffers). The rest is largely used by graphics. The heap uses an additional 20K, with its size based on the graphics taking up about 15K of that. The following figure illustrates the application architecture:

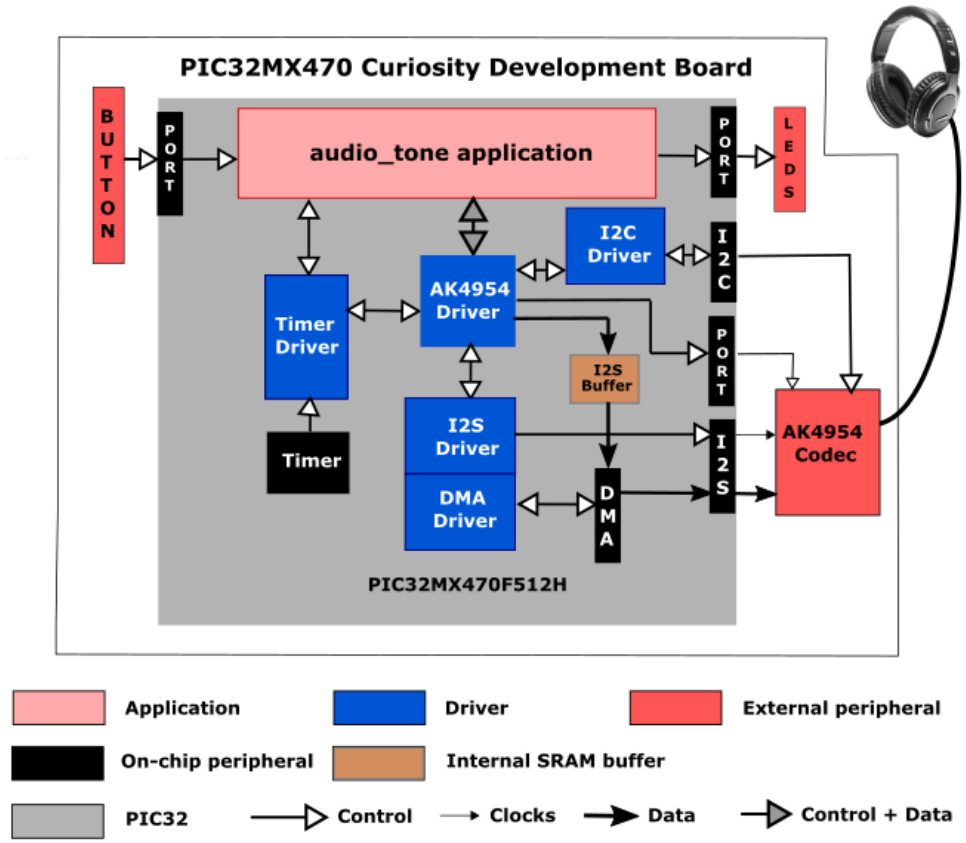


Curiosity PIC32MX470 Development Board configuration:

This configuration runs on the Curiosity PIC32MX470 Development Board, which contains a PIC32MX470F512H microcontroller with 512 KB of Flash memory and 128 KB of RAM running at 120 MHz making use of the following features:

- One push button (S1)
- Two of the four LEDs
- PIC32 AK4954 Audio Codec Daughter Card mounted on an X32 socket

The program takes up to approximately 7% (34K) of the PIC32MX470F512H microcontroller's program space. The 16-bit configuration uses 13% (17K) of the RAM (with 15K of that used by the three audio buffers). The heap is not used. The following figure illustrates the application architecture.

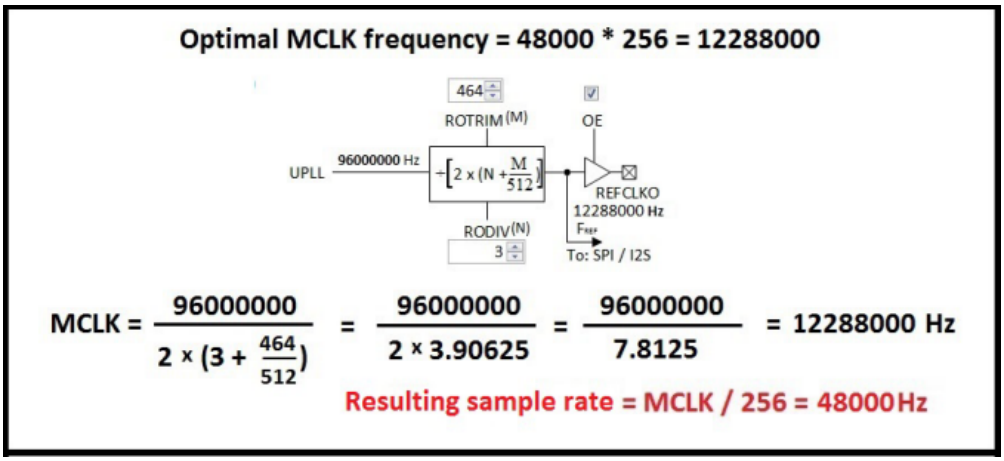


The PIC32 microcontroller (MCU) runs the application code, and communicates with the AK4384 DAC via I2S and a bit-banged SPI interface.

The audio interface between the PIC32 and the AK4384 DAC, AK4953, or AK4954 codecs use the I2S interface. There are two configurations as set up in MPLAB Harmony Configurator (MHC):

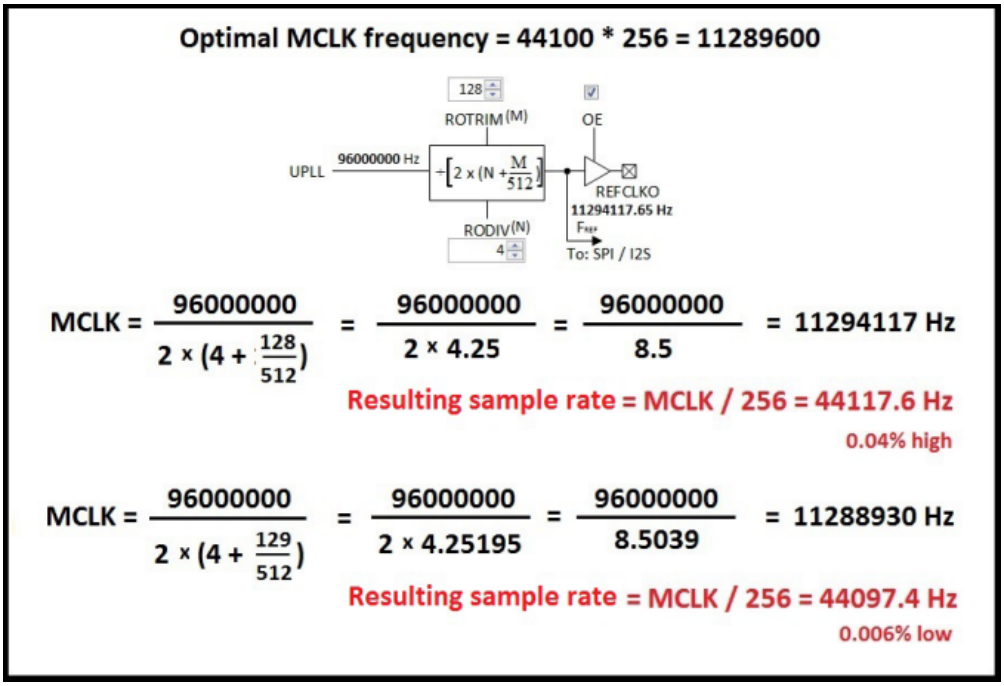
- 16-bit, 48,000 samples/second, right-justified. 16-bit, 48 kHz is the standard rate used for DVD audio. This rate reproduces music faithfully.
- 24-bit, 44,100 samples/second, I2S format. This is the same sample rate as used for CD's, but at a higher bit depth (CD uses 16-bit at 44.1 kHz). This rate also reproduces music very well. The higher bit depth (24) provides a significantly better (49 dB) signal to quantization noise ratio than 16-bits.

The Master Clock (MCLK) signal used by the DAC is generated by the Reference Clock section of the PIC32. For a sample rate of 48 kHz it is 48,000 times 256, or 12,288,000 Hz. (The 256 value is set up in MHC as the Sampling Rate Multiplier.) MCLK, or REFCLKO, is generated from the 96 MHz PLL clock of the PIC32MX470512L using the formula shown in the following figure:



The calculations for the other PIC32 boards is similar, just substituting different clock inputs for the 96000000 Hz in the formula.

For a sample rate of 44.1 kHz it is 44,100 times 256, or 11,289,600 Hz. MCLK, or REFCLKO, is generated from the 96 MHz PLL clock using the formula as shown below. Note that it is not possible to generate the desired frequency exactly. The first value is the one used by the application, as it is the one generated using the Auto_Calulate button in the Clock Diagram:



The calculations for the other PIC32 boards is similar, just substituting different clock inputs for the 96000000 Hz in the formula.

For configurations running on the Bluetooth Audio Development Kit, the interface between the PIC32 MCU and the LCD is an 8-bit parallel master port (PMP). The program uses the MPLAB Harmony v2.x Graphics Library to draw on the screen. The buttons are also interfaced using GPIO pins. Volume is controlled by a potentiometer, which is read via an ADC interface.

For the configuration running on the Multimedia Expansion Board II, the interface between the PIC32 MCU and the LCD uses the External Bus Interface (EBI).

For the configuration running on the Curiosity PIC32MX470 Development Board, the button and LEDs are interfaced using GPIO pins. There is no screen.

As with any MPLAB Harmony application, the SYS_Initialize function, which is located in the `system_init.c` source file, makes calls to initialize various subsystems as needed, such as the clock, ports, board support package (BSP), ADC, AK4384 DAC, AK4953 codec, AK4954 codec, I2S, PMP, EBI, DMA, timers, graphics, and interrupts.

The DAC or codec driver, graphics (if used), and the application state machines are all updated through calls located in the SYS_Tasks function in the `system_tasks.c` file. Interrupt handlers in the `system_interrupt.c` file are used for the single DMA channel, port change notices (if physical buttons are used), and three timers.

The application code is contained in the several source files. The application's state machine (APP_Tasks) is contained in `app.c`. It first initializes the application, which includes getting a handle to a timer driver instance and sets up a periodic (alarm) callback. For the configurations running on the Bluetooth Audio Development Kit, APP_Tasks then periodically calls APP_ButtonTask in `btad_buttons.c` to process any pending button presses and APP_VolumeTasks in `volume_mx.c` to process the potentiometer input (ADC). Once the DAC or codec has been initialized, it also calls APP_DisplayTask in `display_task.c` to process any pending screen updates.

For the configuration running on the MEB-II, APP_Tasks then periodically calls APP_ButtonTask in `meb2_buttons.c` to process any pending button presses coming from the touch screen. Once the AK4954 has been initialized, it also calls APP_DisplayTask in `display_task.c` to process ant pending screen updates.

For the configuration running on the Curiosity Board, APP_Tasks then periodically calls APP_ButtonTask in `curiosity_button.c` to process any pending button press. It then initializes the AK4954 codec.

Then the application state machine inside APP_Tasks is given control, which first gets a handle to the DAC or codec driver by calling the DRV_CODEC_Open function (in `audio_codec.c`) with a mode of DRV_IO_INTENT_WRITE and sets up the sampling rate. It then calls the function AudioGenerateFx (in either `AudioGenerate24BitFx.c` or `AudioGenerateFx.c` to generate audio samples for the next cycle. These are then added to a queue of buffers via a call to PlayBufferQueueAdd in `AudioPlayBufferQueue.c`.

The application state machine then registers an event handler APP_CODEC_BufferEventHandler as a callback with the DAC or codec driver. Finally it makes an initial call to the codec's DRV_CODEC_BufferAddWrite (in `audio_codec.c`) to start playing the generated tone.

As buffers are freed up, additional cycles are generated. However in the current scheme, only one cycle is generated per buffer, so at 1000 KHz, this means a the audio generator must be called every millisecond to keep up, and at 5 KHz, every 200 µs. Since a screen update can take over 10 ms, this means an audio artifact will be heard whenever the screen is updated while audio is being played, for example when turning the volume control on the Bluetooth Audio Development Kit or adjusting the volume using touch buttons on the MEB II screen. To fix this, more cycles should be generated at a time.

Currently, the play buffer is sized as 1280 * 2 samples, where 2 refers to the number of channels. Since each buffer is only used to generate one cycle of a tone, this means the lowest frequency that can be generated at 44100 samples per second is 1 / (1280 / 44100) = 34.5 Hz.

Although the driver interfaces with the DAC or codec via I2S and DMA, which is largely transparent to the application, it is all taken care of by MPLAB Harmony.

Demonstration Features

- Uses the appropriate Codec Driver Library to write audio samples to the AK4384 DAC, AK4953, or AK4954
- At a lower level, uses the I2S Driver Library between the codec library and the DAC or codec for audio
- Use of a circular buffer queue
- Displays graphics on the 220x176 LCD of the PIC32 Bluetooth Audio Development Kit using the MPLAB Harmony Graphics Library (see MPLAB Harmony Graphics Composer Suite) and the Parallel Master Port (see PMP Driver Library)
- Displays graphics on the 480x272 LCD of the Multimedia Expansion Board II using the MPLAB Harmony Graphics Library (see MPLAB Harmony Graphics Composer Suite) and the External Bus Interface
- Processing of button pushes on the PIC32 Bluetooth Audio Development Kit and Curiosity PIC32MX470 Development Board (see Ports Peripheral Library)
- Processing of touch events on the MaxTouch display mounted on the MEB II using the touch API calls of the MPLAB Harmony Graphics Library
- Use of three timers: one as a periodic 1 ms timer for the application, and a second used by the Codec Driver (see Timer Driver Library)

Tools Setup Differences

When building a new application, start by creating a 32-bit MPLAB Harmony project in MPLAB X IDE by selecting *File* > *New Project*. Select the appropriate processor (PIC32MX470F512L, PIC32MX270F512L, PIC32MX470F512H or PIC32MZ2048EFH144) and either the PIC32 Bluetooth Audio Development Kit, PIC32MZ (EF) Starter Kit + MEB II, or PIC32MX470 Curiosity Development Board. Click the MPLAB Harmony icon to start the MPLAB Harmony Configurator (MHC).

Under BSP Configuration, select either PIC32 Bluetooth Audio Development Kit (AK4384) or (AK4954); PIC32MX270F512L w/ Bluetooth Audio Development Kit (AK4384); PIC32MZ EF Starter Kit w/ Multimedia Expansion Board (MEB) II; or PIC32MX470 Curiosity Development Board as appropriate for your hardware.

If running on the PIC32 Bluetooth Audio Development Kit, navigate to *Harmony Framework Configuration* > *Drivers* > *ADC*, and select **Use ADC Driver?**. Expand *Clock Options*, and for TAD Clock (Tad), enter **32**. Expand *ADC Analog Channel Instance 0*, and for Select Dedicated Analog Channel select **ADC_INPUT_POSITIVE_AN1**.

Under *Drivers* > *CODEC*, select the appropriate device: Use **Codec AK4384?** or Use Codec AK4954?, for the Bluetooth Audio Development Kit, Use **Codec AK4953?** for the MEB II, or Use **Codec AK4954?** for the Curiosity Board. The remaining default values do not require any changes. If using the AK4954, then under Drivers > I2C, check the box Include Force Write I2C Function.

Under *Drivers* > *I2S*, select **Use I2S Driver?**. Also, select **DMA Mode** along with **Transmit DMA Support**. **Enable DMA Channel Interrupts** should also be selected. Under *I2S Driver Instance 0*, for Audio Communication Width, select **SPI_AUDIO_COMMUNICATION_16DATA_16FIFO_32CHANNEL**. Under Audio Protocol Mode, select **DRV_I2S_RIGHT_JUSTIFIED**.

If running on a Bluetooth Audio Development Kit, under *Drivers* > *Timer*, select **Use Timer Driver?**, and change the Number of Timer Instances to 3. Under *TMR Driver Instance 0*, change the Timer Module ID to TMR_ID_4, and the Interrupt Priority to INT_PRIORITY_LEVEL_4. Under *TMR Driver Instance 1*, change the Prescale value to TMR_PRESCALE_VALUE_1. Under *TMR Driver Instance 2*, change the Timer Module ID to TMR_ID_2, the Interrupt Priority to INT_PRIORITY_LEVEL_4, and the Prescale value to TMR_PRESCALE_VALUE_1, and the Operation Mode to DVR_TMR_OPERATION_MODE_32_BIT.

Under *Math Library*, expand *LibQ Fixed-Point Math Library Configuration* and select **Use LibQ C Fixed Point Math Library?**.

Under *System Services* >*DMA*, select **Use DMA System Services**. Under *DMA Channel Instance 0*, change the channel number to DMA_CHANNEL_2 AND THE Interrupt Priority to to INT_PRIORITY_LEVEL_2.

For the Bluetooth Audio Development Kit or MEBII, within *Graphics Sta*ck, select **Use Graphics Stack?**. For the Bluetooth Audio Development Kit configurations only, further down, within *Use Harmony Graphics Composer Suite?* > *Middleware* > *Use Aria User Interface Library?*, clear **Enable Touch** since the LCD on the PIC32 Bluetooth Audio Development Kit does not support a touchscreen. Since you are using the LCD, you will also need to go back to the *Drivers* section, and within *PMP*, **select Use PMP Driver**?. Expand that section, and change the value of Strobe Wait Sates to PMP_STROBE_WAIT_4.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the demonstration.

Description

To build this project, you must open the `audio_tone.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/audio/audio_tone`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`audio_tone.X`	`<install-dir>/apps/audio/audio_tone/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
bt_audio_dk_16bit_48000_RJ	bt_audio_dk	This demonstration runs on the PIC32MX470F512L device on the PIC32 Bluetooth Audio Development Kit and the AK4384 Audio DAC Daughter Board. The configuration is for a sine tone or chirp signal with 16-bit data width, 48000 Hz sampling frequency, and right-justified audio protocol.
bt_audio_dk_24bit_44100_I2S	bt_audio_dk	This demonstration runs on the PIC32MX470F512L device on the PIC32 Bluetooth Audio Development Kit and the AK4384 Audio DAC Daughter Board. The configuration is for a sine tone or chirp signal with 24-bit data width in a 32-bit channel, 44100 Hz sampling frequency, and I2S audio protocol.
pic32mz_ef_sk_meb2_16bit_48000_RJ	pic32mz_ef_sk+meb2	This demonstration runs on the PIC32MZ EF Starter Kit mounted on the Multimedia Expansion Board II (MEB-II) with a PDA 4.3” WQVGA MaxTouch PCAP touch display module. The configuration is for a sine tone or chirp signal with 16-bit data width, 48000 Hz sampling frequency, and right-justified audio protocol.
pic32mx470_curiosity_16bit_48000_RJ_ak4954	pic32mx470_curiosity	This demonstration runs on the Curiosity PIC32MX470 Development Board with the AK4954 Audio Codec Daughter Board. The configuration is for a sine tone or chirp signal with 16-bit data width, 48000 Hz sampling frequency, and right-justified audio protocol.
pic32mx270f512l_pim_bt_audio_dk_16bit_48000_RJ	pic32mx270f512l_pim_bt_audio_dk	This demonstration runs on the PIC32MX270F512L PIM mounted on the PIC32 Bluetooth Audio Development Kit along with the AK4384 Audio DAC Daughter Board. The configuration is for a sine tone or chirp signal with 16-bit data width, 48000 Hz sampling frequency, and right-justified audio protocol.
bt_audio_dk_16bit_48000_RJ_ak4954	bt_audio_dk+ak4954	This demonstration runs on the PIC32MX470F512L device on the PIC32 Bluetooth Audio Development Kit and the AK4954A Audio DAC Daughter Board. The configuration is for a sine tone or chirp signal with 16-bit data width, 48000 Hz sampling frequency and right-justified audio protocol.

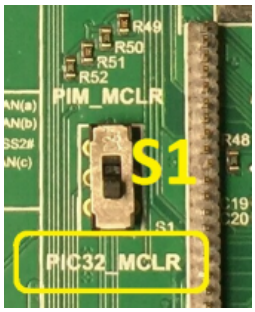
Configuring the Hardware

This section describes how to configure the supported hardware.

Description

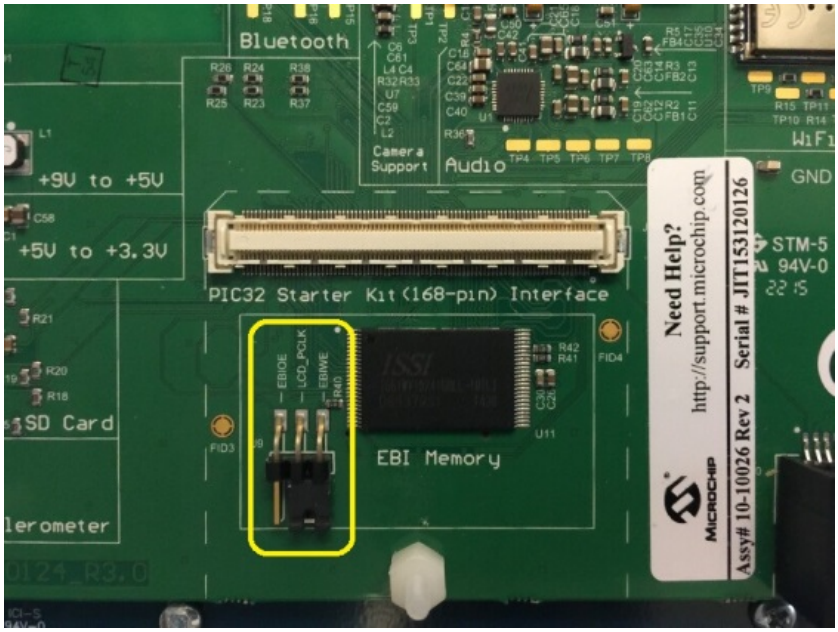
PIC32 Bluetooth Audio Development Kit and AK4384 Audio DAC Daughter Board, or AK4954A Codec Daughter Board

Switch S1 on the PIC32 Bluetooth Audio Development Board (between the audio DAC daughter board and the microcontroller PIM) should be set to PIC32_MCLR.



PIC32MZ EF Starter Kit mounted on the Multimedia Expansion Board II (MEB-II):

The jumper for J9, located on the backside of the MEB-II underneath the PIC32MZ EF Starter Kit daughter board, should be placed on pins LCD_PCLK and EBIWE as shown below in yellow, so as to enable the internal frame buffer.

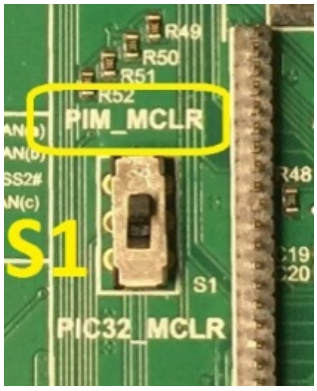


Curiosity PIC32MX470 Development Board:

(No special configuration.)

PIC32 Bluetooth Audio Development Kit with PIC32MX270F512L PIM and Audio AK4384 DAC Daughter Board:

Switch S1 on the PIC32 Bluetooth Audio Development Board (between the audio DAC daughter board and the microcontroller PIM) should be set to PIM_MCLR.



Running the Demonstration

This section demonstrates how to run the demonstration.

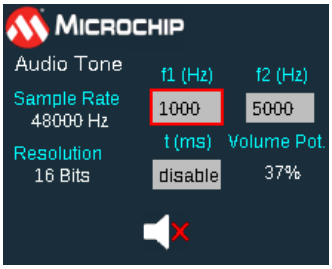
Description

PIC32 Bluetooth Audio Development Kit configurations:

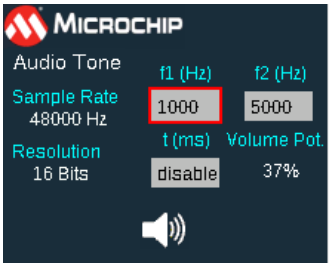
Both continuous sine tones and finite length sine tones and chirps can be generated. **Table 1** provides a summary of the button actions that can used to control the audio output waveform characteristics.

Compile and program the target device. While compiling, select the appropriate MPLAB X IDE project configuration based on the demonstration board. Refer to Building the Application for details.

1. Connect headphones to the line-out connector of the Audio DAC/Codec Daughter Board (see **Figure 1**).
2. The tone can be quite loud, especially when using a pair of headphones. Before running the demonstration, turn the volume control, P2, which is located on the development board underneath the codec daughter board, all the way clockwise, and then turn it counterclockwise until the tone can be heard clearly.
3. Initially, the tone is disabled and the screen will appear, as shown in the following figure.



4. SW4 will turn the audio on or off, as indicated by the audible output and the two illuminated green LEDs on the Audio DAC Daughter Board (if being used) along with the speaker icon on the screen, which changes, as shown in the following figure.



5. Modifiable parameters are selected by pressing SW3, which cycles through the parameters: "f1 (Hz)", f2 (Hz), and "t (ms)". Initially, the "f1 (Hz)" parameter is selected (initially, it is 1000 Hz). The value of any selected parameter can be changed by using SW1/SW2 to raise or lower the value, respectively. The "f2 (Hz)" parameter, is the chirp final frequency. The "t (ms)" parameter is the duration of the signal in milliseconds. Note that as SW3 is pressed, the LEDs D5, D6, and D7 illuminate to indicate the mode f1, f2, or t;, the demonstration is advancing to the next parameter, and then wraps around.

Note: A long press (5 seconds) of SW3 will set the selected parameter value to its maximum allowable value. A long press of SW5 will set the minimum value.

6. The Sine Tone or Chirp modes are selected by pressing SW5. Chirp mode is indicated by LED D8 being illuminated and the display changing from "Sine" to "Chirp". The output automatically turns off when a new mode is selected. Pressing SW4 initiates the audible output based on the current settings of the parameters.

Notes:

1. When the "t (ms)" parameter is disabled, a continuous sine tone is generated at the "f1 (Hz)" frequency for either mode. Incrementing the value by pressing SW1 will initiate finite length chirps or sine tones starting from 0 and increasing by 10 ms steps for each increment/decrement. A non-zero t value is indicated by LED D9 being illuminated.
2. Long presses of SW1/SW2 will accelerate the incrementing/decrementing of the selected value.

7. As an optional step, if there is interest and the necessary equipment, the displayed sampling frequency can be verified by probing the point "LRCK Pin Point" of the Audio DAC/Codec Daughter Board (PIC32 Bluetooth Audio Development Kit configurations using the AK4384), as shown in **Figure 1**.
8. As an optional step, if there is interest and the necessary equipment, the signal frequency of the continuous Sine Tone output can be verified by probing the "Line Out Point" of the Audio DAC/Codec Daughter Board, as shown in **Figure 1**. Finite length sine tones and chirp parameters can also be verified by probing this point with a storage oscilloscope.

Figure 1: Audio DAC Daughter Board on PIC32 Bluetooth Audio Development Kit

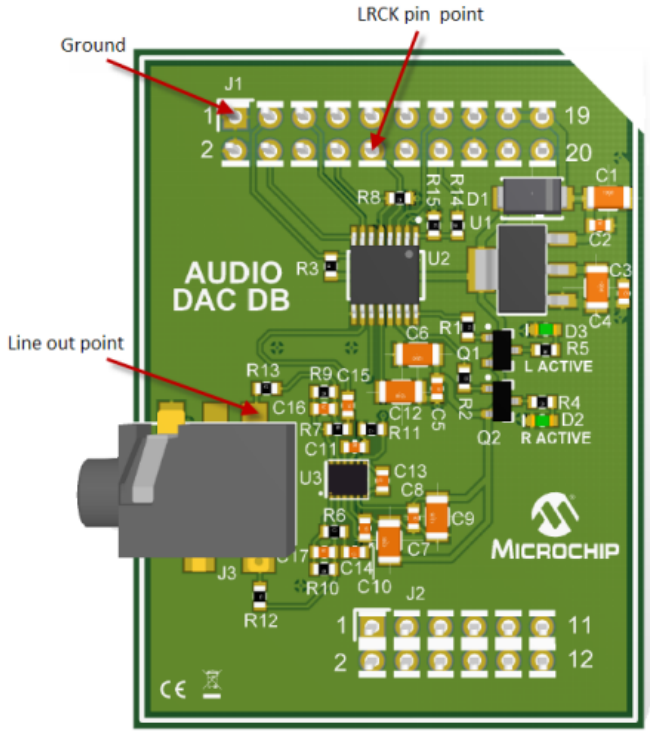
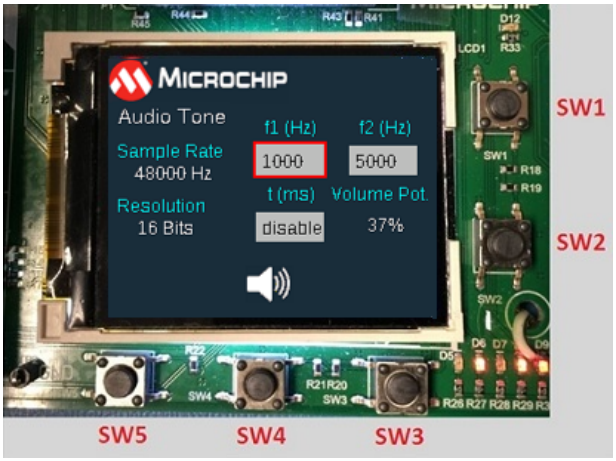


Figure 2: Audio Tone Graphics Display



Control Descriptions

Table 1: Button Controls for Bluetooth Audio Development Kit

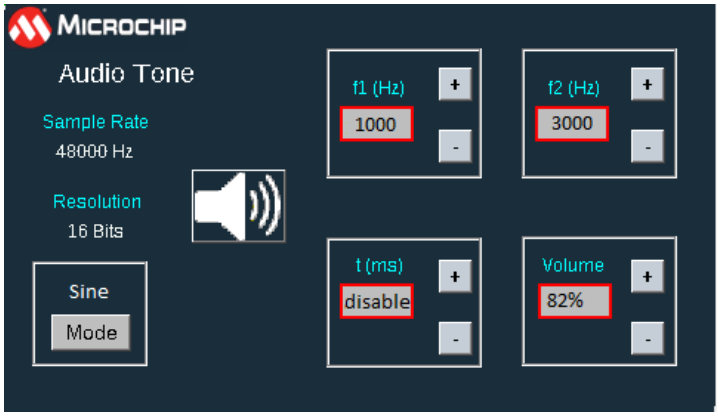
Control	Description
SW1/SW2	Parameter increment/decrement (long press - accelerates increment/decrement).
SW3	f1/f2/t parameter selection (long press - parameter maximum value).
SW4	Audio ON/OFF **Note:** Finite length waveforms will play, and then turn OFF.
SW5	Chirp/Sine mode selection (long press - parameter minimum value).

Multimedia Expansion Board II configuration:

Both continuous sine tones and finite length sine tones and chirps can be generated. Table 3 provides a summary of the button actions that can used to control the audio output waveform characteristics.

Compile and program the target device. While compiling, select the appropriate MPLAB X IDE project configuration based on the demonstration board. Refer toBuilding the Application for details.

1. Connect headphones to the line-out connector of the MEB-II (see Figure 3).
2. Initially, the tone is disabled and the screen will appear as shown in the following figure:



3. Pressing the speaker icon will turn the audio on, as indicated by the audible output along with the speaker icon on the screen changing to a muted speaker (see below). Pressing the button again will turn off the audio.



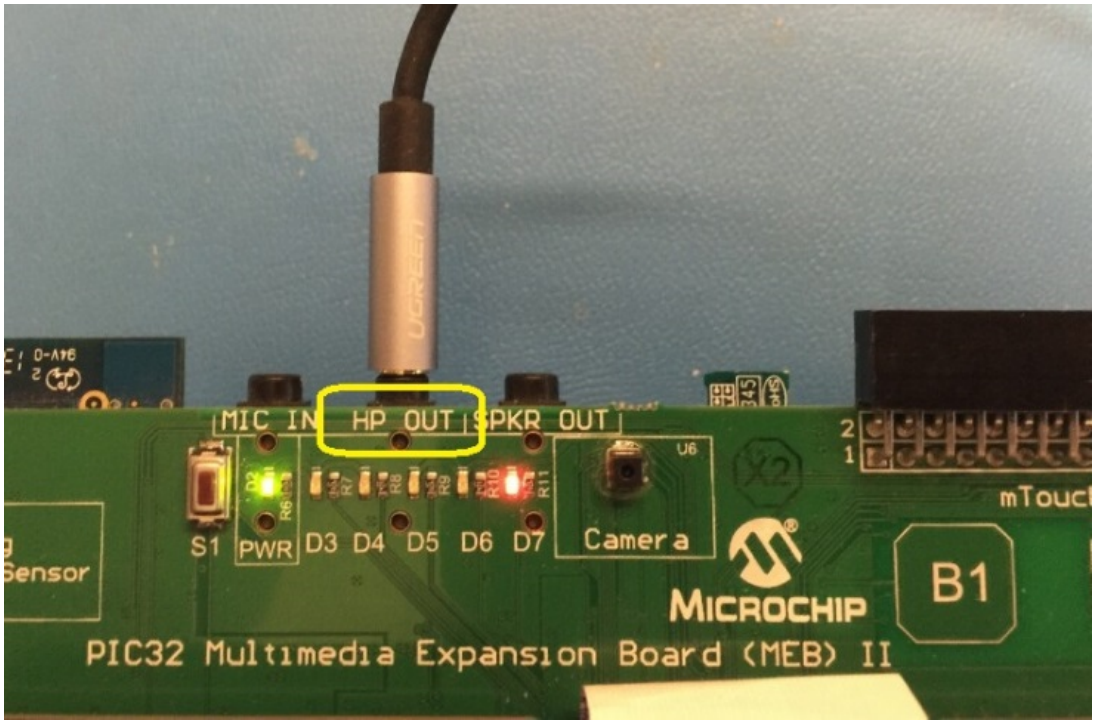
4. Modifiable parameters are selected by pressing the touch buttons + or - associated with each of the parameters: "f1 (Hz)", f2 (Hz), "t (ms)", and “Volume”.

5. Initially, the "f1 (Hz)" parameter is selected (default 1000 Hz). The "f2 (Hz)" parameter is the chirp final frequency. The "t (ms)" parameter is the duration of the audio signal (sine or chirp) in milliseconds.

6. The Sine Tone or Chirp modes are selected by pressing the Mode button on the lower-left corner of the screen. The text above it indicates the current mode.

Notes:

1. When the "t (ms)" parameter is disabled, a continuous sine tone is generated at the "f1 (Hz)" frequency for either mode. Incrementing the value by the + button will initiate finite length chirps or sine tones starting from 10 ms and increasing by 10 ms steps for each increment/decrement.
2. Long presses of any of the + or - buttons will accelerate the incrementing/decrementing of the associated value.



Curiosity PIC32MX470 Development Board configuration:

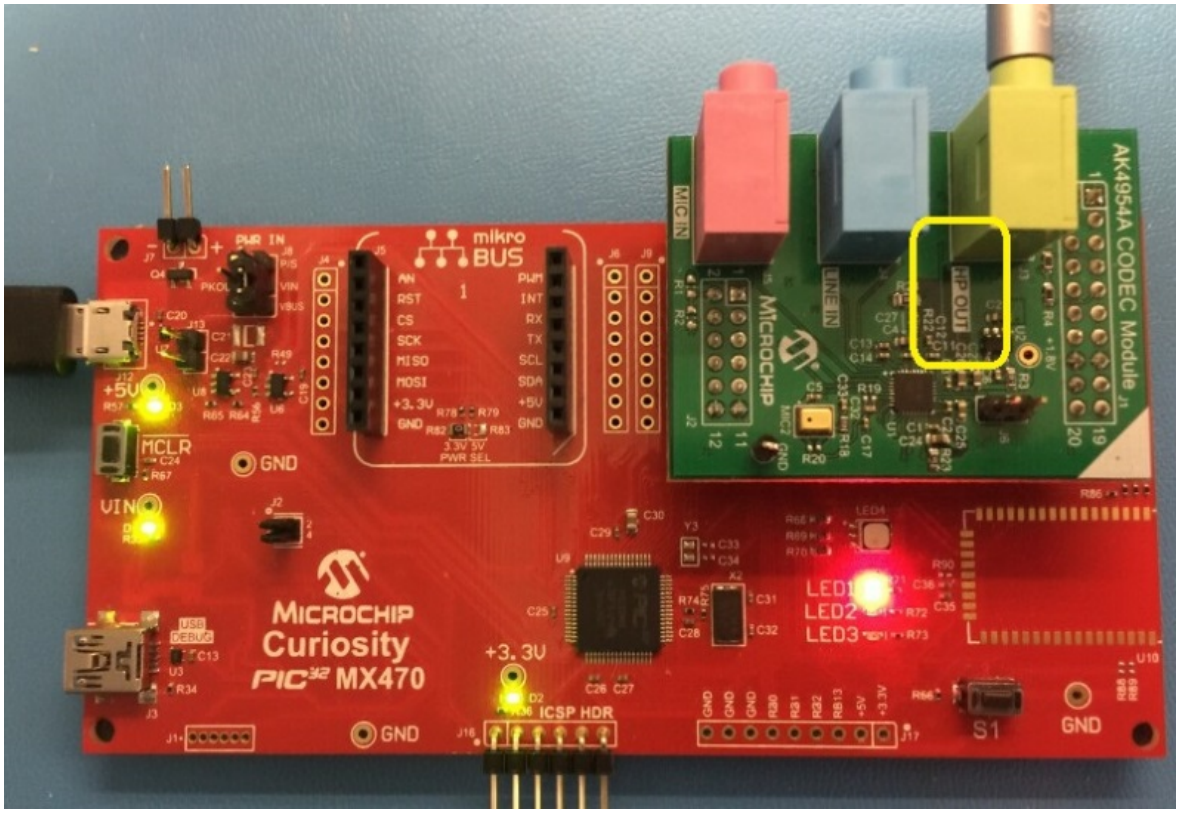
The following can be generated; continuous sine tones, finite length sine tones, and chirps. The table below provides a summary of the button actions that can used to control the audio output waveform characteristics.

Compile and program the target device. While compiling, select the appropriate MPLAB X IDE project configuration based on the demonstration board.

Refer to Building the Application for details.

1. Connect headphones to the headphone connector (HP OUT) of the Audio DAC/Codec Daughter Board (refer to the following figure).
2. Initially, the tone is disabled and all LEDs are off.
3. Pressing and releasing S1 button will turn the audio on or off, as indicated by the audible output and LED 1 (red) being illuminated.
4. A long press (four seconds) and release of S1 will toggle the mode from a continuous sine tone to a two second chirp tone and back. Chirp mode is indicated by LED 2 (green) being illuminated.
5. In chirp mode, pressing and releasing S1 will enable the chirp tone, which will go off automatically after the two second period.

Note: Due to the lack of sufficient user buttons, the frequencies, duration and volume cannot be changed by the user interface as with the other configurations. The starting and ending frequencies can be modified by changing the fHz1 and fHz2 members of the agParamInit and guiDataInit structures in `app.c`, and the duration can be modified by changing the FIXED_CHIRP_DURATION macro in `app_config.h`. The volume can be configured using MHC.



Control Descriptions

Button Controls for Curiosity Board

Control	Description
S1	Audio ON/OFF (long press – toggles between sine and chirp modes)

emwin_media_player

This topic demonstrates an audio player application based on Segger emWin graphics

Description

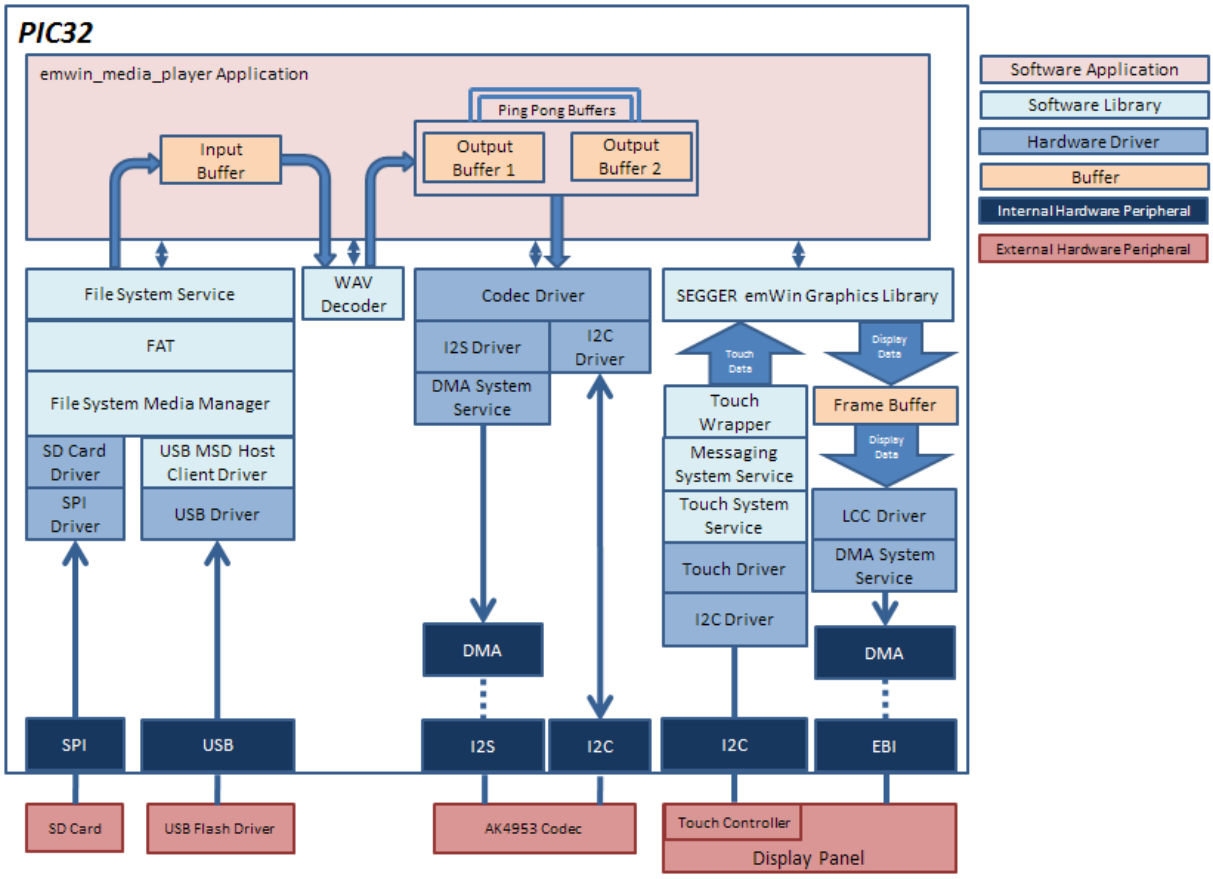
This application demonstrates the creation of an audio player that plays WAV files from an SD Card and from a USB flash-drive. The graphical user interface (GUI) with touch screen support is designed using the SEGGER emWin Graphics Library. The GUI provides options to select media type (SD Card / Flash-drive), volume controls, random selection and shuffling of tracks, and play list view with progress bar and seek bar.

Architecture

The emwin_media_player application uses the SEGGER emWin Graphics Library to render graphics to the display. The graphics library draws the widgets and images into the internal frame buffer. This frame buffer is transferred to the LCD display panel by the Low Cost Controllerless (LCC) driver using the DMA. Using the I2C Driver, the touch inputs from the touch controller are read by the touch driver. The Touch System Service passes the touch-related information to the graphics library using the messaging system service and the touch wrapper.

The application can play `.wav` files either from the SD Card or the USB Flash drive. The SD Card driver uses the SPI driver to interact with the SD Card. The application uses the File System Service to read/write data to the selected media (SD Card/USB Flash drive). The File System Media Manager directs the calls from the native file system to the selected media. The audio data read from the SD card is decoded by passing it to the WAV decoder. The decoded output is saved in the output buffers 1 and 2, which are used in ping pong manner. The output buffers 1 and 2 are submitted to the Codec driver for playing. Similarly, the audio data read from the USB flash drive is saved in the ping pong buffers which are then submitted to the Codec driver for playing.

The Codec driver sends the audio data to the AK4953 Codec using the I2S Driver, which in turn uses DMA to transfer the audio data. The Codec driver uses the I2C Driver to send commands to the AK4953 Codec.



Demonstration Features

- SEGGER emWin Graphics Library
- SD Card Driver
- USB MSD Host Client Driver Library
- AK4953 Codec Driver
- MTCH6301 Touch Driver
- File System Service
- WAV Decoder
- Low-Cost Controllerless (LCC) Graphics Driver
- SPI Driver used by SD Card Driver
- I2S Driver used by Codec Driver for audio data transfer
- I2C Driver used by Codec (for command transfer) and Touch Drivers
- DMA System Service
- Random Number Generator System Service

Tools Setup Differences

- The Master Clock Input (MCKI) to the Codec is provided by Reference Clock Generator Output1 (REFCLKO1) of PIC32. The value for REFCLKO1 is set by navigating to *MPLAB Harmony Configurator* > *Clock Diagram* and selecting **Auto Calculate** under REFCLKO1. Here, select **Target I2S Input Frequency** and click **Apply**. The REFCLKO1 will be set to 256 times the sampling frequency or 48000 x 256 = 12288000 Hz.
- The output of the System PLL is set to 198 MHz to accurately generate the REFCLKO1, which is set to 48000 x 256 = 12288000 Hz. This is done by setting the System PLL in the *MPLAB Harmony Configurator* > *Clock Diagram*. Here, select **Auto Calculate** and set the “Desired System Frequency” to 198 MHz.
- The REFCLKO1 signal is mapped to PIN 70 (RD15). This is done in the *MPLAB Harmony Configurator* > *Pin Table*.
- The touch driver uses the PIC32's "External Interrupt 1" (INT1) signal for touch events. The "External Interrupt 1" signal is mapped to pin RE8 (Pin 23). This is done through the *MPLAB Harmony Configurator* > *Pin Table*.
- “Include Force Write I2C Function” is checked under *Harmony Framework Configuration* > *Drivers* > *I2C*. This will include an API that sends data to the slave even if the slave NACKs data. This is needed by the AK4953 CODEC since it NACKs I2C data during the initialization sequence.
- “Use SEGGER emWin Touch Wrapper?” is enabled under *Third Party Libraries* > *Graphics* > *Use SEGGER emWin Graphics Library?* >

SEGGER emWin Graphics Library, which integrates the touch-related SEGGER emWin application code with MPLAB Harmony.

- "Use SEGGER emWin GUI Wrapper?" is enabled, which is used for integrating the GUI related SEGGER emWin code with MPLAB Harmony. This is done through *Third Party Libraries > Graphics > Use SEGGER emWin Graphics Library? > SEGGER emWin Graphics Library*. Here, the memory block used by the SEGGER emWin graphics library is set to 10000 to allow sufficient memory to the internal memory management system of the graphics library.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the emWin Media Player Demonstration.

Description

To build this project, you must open the `emwin_media_player.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/audio/emwin_media_player`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`emwin_media_player.X`	`<install-dir>/apps/audio/emwin_media_player/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mz_ef_sk_meb2_legacy	pic32mz_ef_sk+meb2_legacy	This configuration runs on the PIC32MZ EF Starter Kit connected to the legacy MEB II. The micro SD card is used in SPI mode and is configured for Interrupt mode and dynamic operation. The USB library is configured for High-Speed operation with the MSD Host client driver. The Codec is interfaced over I2C for command and I2S for data and uses DMA for data transfers. The graphical display is driven by the LCC driver and uses DMA for data transfers. The touch screen driver is interfaced using I2C configured for Interrupt mode and dynamic operation. The Graphical User Interface is designed using the SEGGER emWin Graphics Library.

Configuring the Hardware

This section describes how to configure the supported hardware.

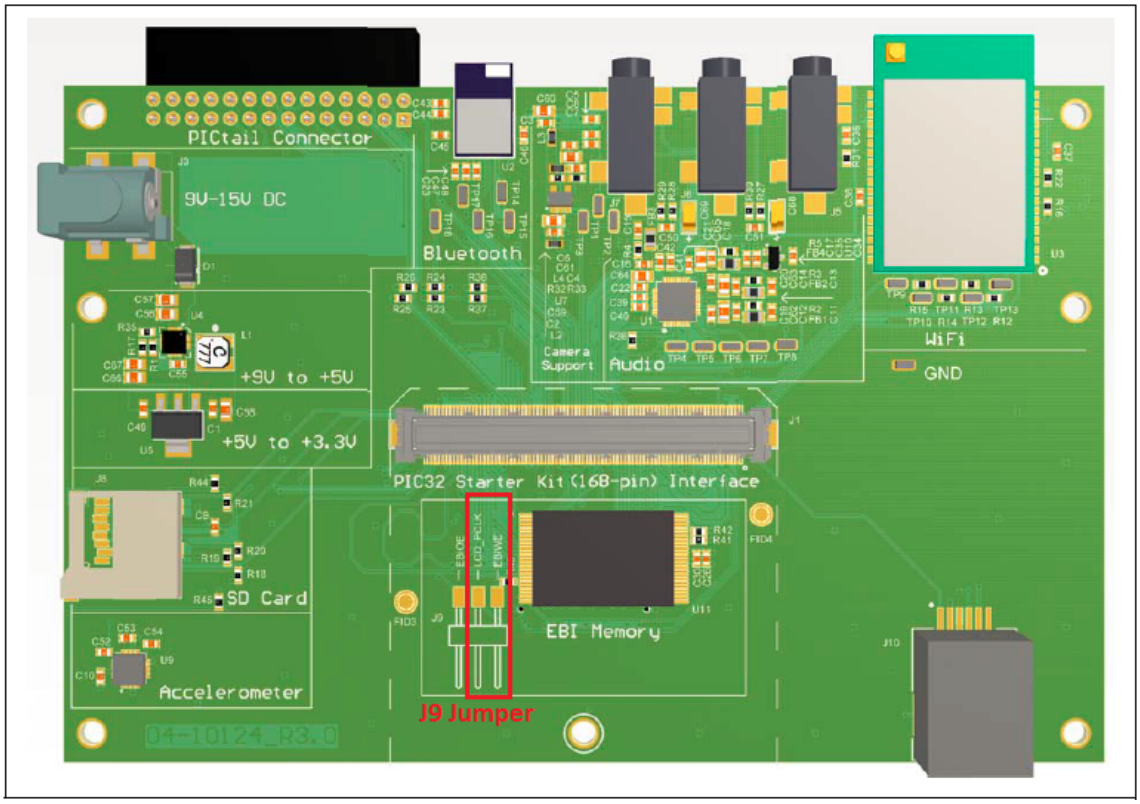
Description

PIC32MZ EF Starter Kit and First Generation Multimedia Expansion Board II (MEB II)

Configuration: pic32mz_ef_sk_meb2_legacy

On the MEB II, the EBIWE and LCD_PCLK (J9) must be closed. The jumper (J9) is available on the bottom side of the MEB II board. See the following figure for the jumper location.

- Connect the PIC32MZ EF Starter Kit board to the MEB II board.
- Power up the board by connecting the power adapter to J3 power connector on the MEB II board or through a USB cable to the USB DEBUG port J3 on the Starter Kit board.
- Insert your micro SD card into the SD card slot (J8) on the MEB II board. Ensure that the SD card contains `.wav` audio files
- Connect speakers or headphones to the headphone out (HP OUT) on the MEB II



Running the Demonstration

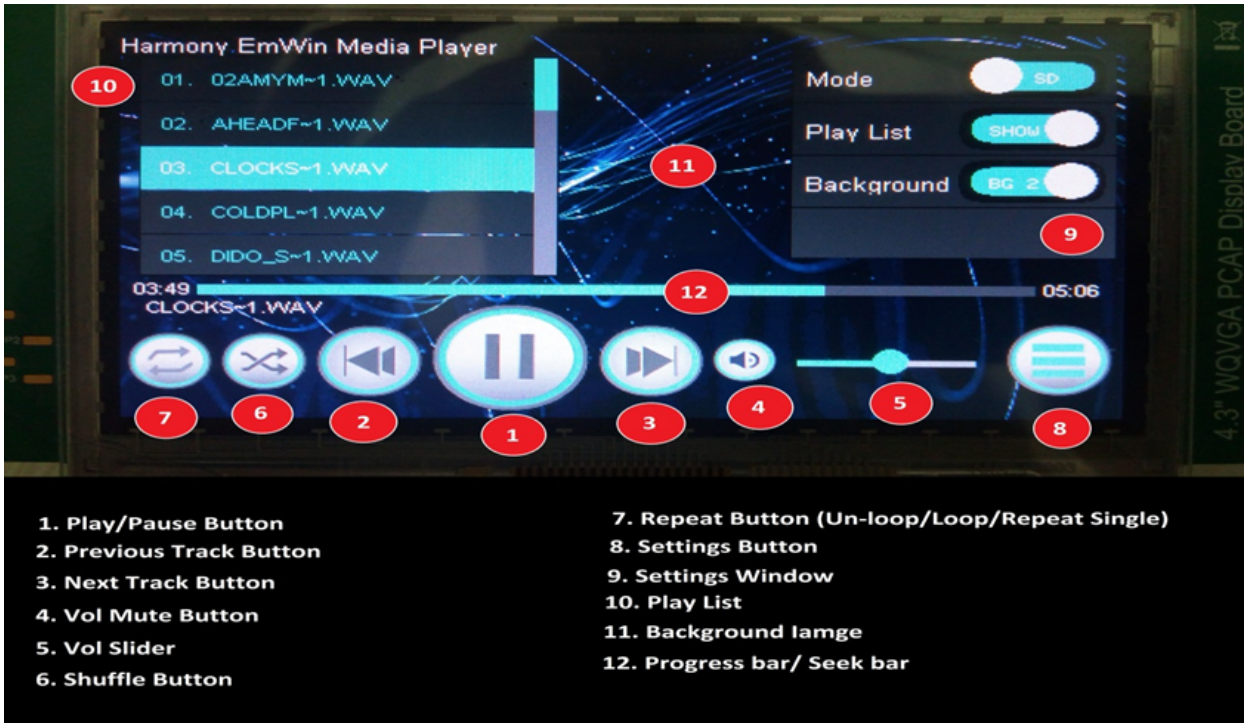
This section provides instructions about how to build and run the emWin Media Player Audio Demonstration.

Description

By default, the application has the SD card selected as the media source. After the board powers up, the GUI should appear as shown in the following figure.



Refer to the following figure for the locations of the GUI options.



1. Touch Play/Pause (1) to listen to the audio. The volume slider (5) will allow you to increase/decrease the volume level and the mute button (4) to mute/un-mute the audio. The track can be changed by pressing the next (3) and previous (2) buttons.
2. The progress bar (12) will indicate the progress (elapsed time) of the track being played. The progress bar also acts as a seek-bar allowing you to seek the track in forward or reverse direction by touching the progress bar.
3. To open the Settings window (9), click on the settings button (8). Here, there are three options.
   - Mode: To select between the SD Card and Flash-drive (USB) as the media source. By default, the SD Card is selected as the media source. Insert the Flash-drive after changing the mode to the Flash-drive (USB) mode.
   - Play List: To show/hide the play list (10). By default, the play list is hidden.
   - Background: To change the background image (11). By default, the BG1 is selected as the background image.
4. In the Settings window (9), click on the Play List button to un-hide (show) the play list. Any random track can be selected through the play list.
5. Pressing the Repeat Button (7) will allow you to either:
   - Unloop the track list (default). Track list will end when the last song in the list is played out.
   - Loop the track list. Track list will loop continuously.
   - Repeat single track. The selected track will loop continuously.
6. The Shuffle button (6) will allow you to shuffle the track list. This is done through a random number generator which selects a random track in the play list for playing.

Notes:

1. Shuffling is turned off when the Repeat Button (7) is in “Repeat Single Track” mode.
2. When the player is in flash-drive (USB) mode, removing the flash drive causes the player to automatically switch back to the default (SD Card) mode.

Important:

1. The plug-and-play feature of the SD card is not supported. The demonstration does not respond if you remove the SD card and then insert it while the audio is being played. If you want to remove/connect or replace an SD card, power down the device, remove/connect or replace the SD card, and then power up the device.
2. The USB Flash drive must be inserted when the player is in the Flash drive (USB) mode. Trying to insert a Flash drive while in the SD Card mode may result in the Flash drive not being detected and may require a power reset to detect the Flash drive.

mac_audio_hi_res

This topic demonstrates a USB Audio 2.0 Device that emulates a USB speaker.

Description

This demonstration application uses the USB Audio 2.0 Device class to implement a speaker.

Note: This demonstration can be used only with an Apple® Mac® personal computer such as the Apple MacBook Air® OS X 10.9.4 with iTunes® 11.2.1 or VLC media player version 2.2.1. Any versions prior to those listed may not work with this demonstration.

Architecture

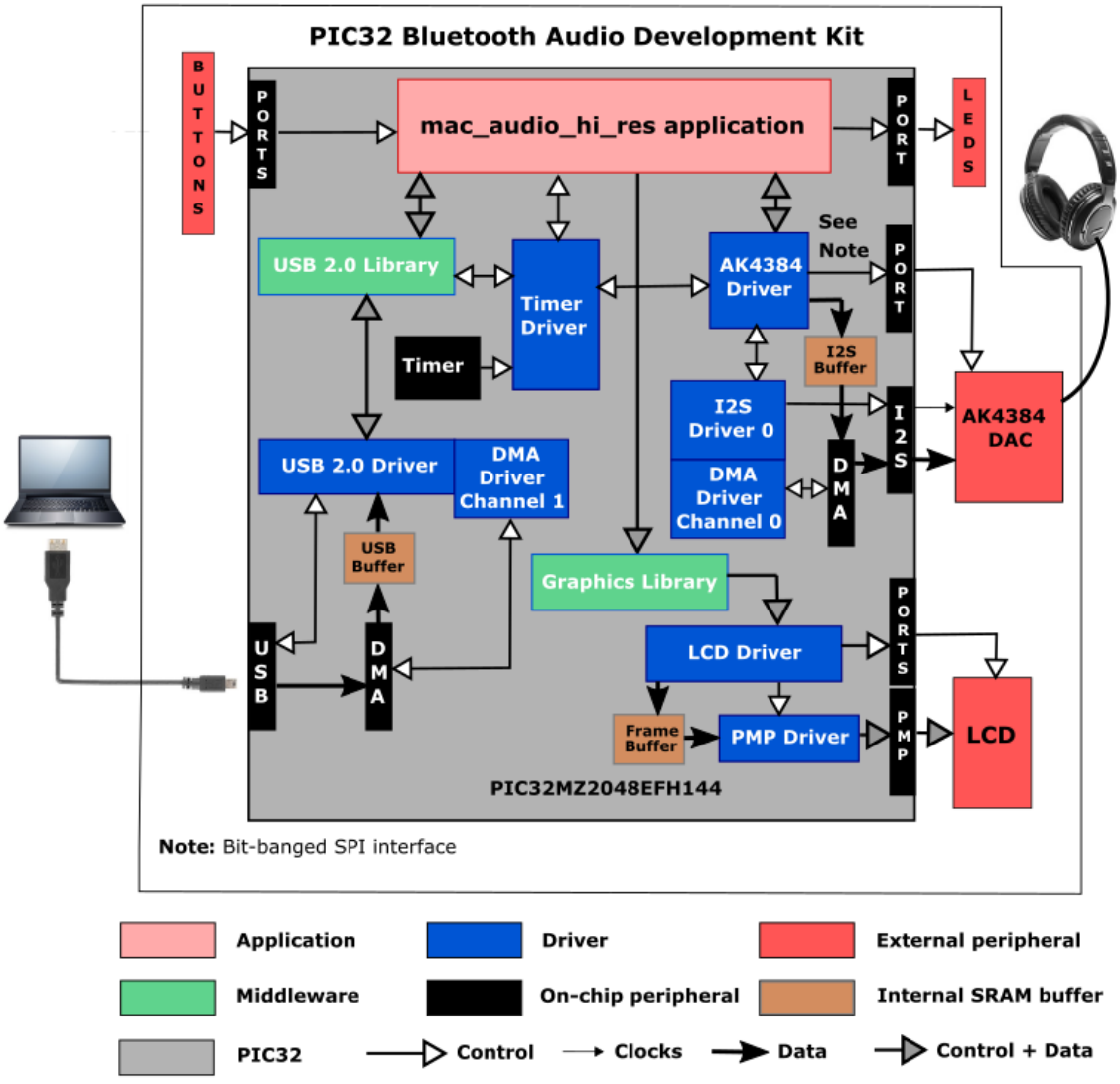
This application runs on the PIC32 Bluetooth Audio Development Kit and uses the following features:

- 220x176 color LCD
- Three push buttons (SW1-2 and SW4)
- Five LEDs
- USB Device interface
- AK4384 Audio DAC Daughter Board mounted on X32 socket

Surrounding the PIC32MX470F512L microcontroller are a set of headers, which accept various plug-in modules (PIM), including one for a PIC32MZ2048EFH144 PIM (which has 2 MB of flash and 512 KB of RAM) which Is used instead of the on-board X32 PIC32MX470F512L.

The program takes up only about 10% (202K) of the PIC32MZ2048EFH144 microcontroller's program space, and 6% (33K) of its RAM. The 64 audio buffers take up 12K of that, and another 7K is used by graphics. The heap uses an additional 16K, with its size based on the graphics taking up about 6K of that.

The following figure illustrates the application architecture.

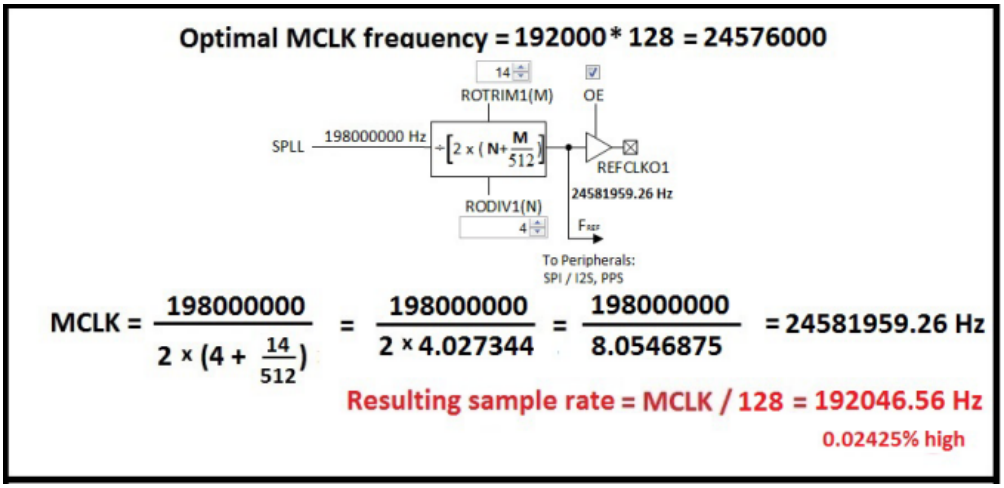


The PIC32 microcontroller (MCU) runs the application code.

The interface between the PIC32 MCU and the LCD is an 8-bit parallel master port (PMP). The program uses the MPLAB Harmony v2.0 Graphics Library to draw on the screen. The buttons are also interfaced using GPIO pins.

The audio interface between the PIC32 and the AK4384 DAC uses a I2S interface. The default configuration is set up for 24-bit and 192,000 samples/second. 192K, or high-definition audio, is four times the 48K standard normally used for professional audio recording. The higher bit depth (24) provides a significantly better (49 dB) signal to quantization noise ratio than 16-bits.

The Master Clock (MCLK) signal used by the DAC is generated by the Reference Clock section of the PIC32. For a sample rate of 192 kHz it is 48,000 times 128, or 24,576,000 Hz. (The 128 value is set up in MHC as the Sampling Rate Multiplier.) MCLK, or REFCLKO, is generated from the 198 MHz SPLL clock using the formula as shown below. The desired clock can not be obtained exactly, but instead the resulting rate (192046 Hz) is 0.024% high.



As with any MPLAB Harmony application, the SYS_Initialize function, which is located in the `system_init.c` source file, makes calls to initialize various subsystems, such as the clock, ports, board support package (BSP), USB, AK4384 DAC, I2S, PMP, DMA, timers, graphics, and interrupts.

The USB 2.0 driver, AK4384 driver, graphics, and the application state machines are all updated through calls located in the SYS_Tasks function in the `system_tasks.c` file. Interrupt handlers in the `system_interrupt.c` file are used for the two DMA channels, port change notices, and two timers.

The application code is contained in the several source files. The application's state machine (APP_Tasks) is contained in `app.c`. It first initializes the application, which includes getting a handle to the USB Device driver via a call to USB_DEVICE_Open. It then sets up an event handler APP_UsbDeviceEventCallBack for endpoint 0 (control events).

Once the device has been configured, the application gets a handle to the AK4384 driver by calling DRV_CODEC_Open, and sets up an event handler APP_CodecBufferEventHandler to get buffer-related events. It then submits a request to get USB 2.0 audio from the host PC using the call USB_DEVICE_AUDIO_V2_Read.

When data is available from the USB host, it is sent to the AK4384 DAC via a call to DRV_CODEC_BufferAddWrite. A set of circular buffers are used, and the number of available buffers is continually compared with upper and lower limits, with feedback given back to the PC whether to speed up, slow down or remain the same via a call to USB_DEVICE_AUDIO_V2_Write.

The app state machine also calls APP_ButtonTask in `btad_buttons.c` to process any pending button requests, which will be either for mute on/off (SW4), or volume level changes (SW1/SW2). Any status change will be reflected in a screen update processed in display.c.

This application uses USB Audio Class 2.0, which uses sends a frame every 125 µs, eight times faster than USB Audio Class 1.0, which is limited to a frame very 1 ms. So USB Audio Class 2.0 allows audio formats as high as 384K samples per second, compared to a maximum of 96K samples per second for the 1.0 standard.

In this application, we are using up to a maximum of 24-bits and 192K samples per second, limited by the AK4383 DAC. The nominal number of bytes transferred per frame is set at 192, so in 1 ms you have 192*4(bytes/sample)*2(channels) or 192000 32-bit samples per second per channel. (The 24-bit audio is carried inside a 32-bit frame.) This represents a bandwidth of 12MB/s (plus overhead).

A total of 64 200-byte buffers, using 12K of RAM, are made available in the receive queue. (The eight extra bytes are allowed for tuning.) So they will be rotated every 8 ms.

Although the AK4642 Codec Driver interfaces with the codec via I2S and DMA, and the USB also uses the DMA, which is largely transparent to the application, it is all taken care of by MPLAB Harmony.

Demonstration Features

- Uses the USB Audio 2.0 Device Library to receive audio samples from the host PC over the USB 2.0 audio interface
- Uses the AK4384 Codec Driver Library to write audio samples to the AK4384 DAC
- At a lower level, uses the I2S Driver Library between the codec library and the DAC for audio
- Use of a circular buffer queue with a throttling mechanism
- Displays graphics on the 220x176 LCD of the PIC32 Bluetooth Audio Development Kit using the MPLAB Harmony Graphics Library (see MPLAB Harmony Graphics Composer Suite) and the Parallel Master Port (see PMP Driver Library)
- Processing of button pushes on the PIC32 Bluetooth Audio Development Kit (see Ports Peripheral Library)
- Use of two timers: one as a periodic 1 ms timer for the application, and a second used by the AK4384 driver (see Timer Driver Library)

Tools Setup Differences

When building a new application, start by creating a 32-bit MPLAB Harmony project in MPLAB X IDE by selecting *File > New Project*. Select the PIC32MX Bluetooth Audio Development Kit and the configuration audio_bt_dk. Click the MPLAB Harmony icon to start the MPLAB Harmony Configurator (MHC).

Expand *Drivers >CODEC* and select **Use Codec AK4384?**. Select **Specify MCLK value**, and enter **128**. Expand *Use Bit Banged SPI Control*

Interface and also *Code AK4384 Driver Instance 0*. Change the Timer driver (used for bit banging) instance to 1.

Under *I2S*, select **Use I2S Driver?**. Select **DMA Mode**, along with Transmit DMA Support, Use DMA Channel Chaining, and Enable DMA Channel Interrupts. Under *Sampling Rate*, enter **44100**. For Master Clock\Bit Clock Ratio, enter 2. Expand *I2S Driver Instance 1*, and for Audio Communication Width, choose SPI_AUDIO_COMMUNICATIONS_32DATA_32FIFO_32CHANNEL. Change Queue Size Transmit to 64.

In the Timer section, select a second timer (change Number of Timer Instances from 1 to 2), and then change the Timer Module ID for TMR Driver Instance 0 to TMR_ID_2, and the Timer Module ID for TMR Driver Instance 1 to TMR_ID_4.

Under *Harmony Framework Configuration >USB Library*, select **Use USB Stack?**. Expand *Select Host* or *Device Stack*. USB Device should already be selected. Change the Number of Endpoints Used to 23. Under *USB Device Instance 0*, expand *Function 1*. Change the Device Class to AUDIO_2_0. Change the Number of Interfaces to 2, Speed to USB_SPEED_HIGH, Audio Read Queue Size to 64, and Audio Write Queue Size to 2. Change the Product ID Selection to “mac_audio_hi_res_demo”, which will fill in the Vendor and Product ID fields.

Under *System Services >DMA*, select **Use DMA System Services**. Change the Number of DMA Channel Instances to 2 and press **Enter**. Under both DMA Channel Instances 0 and 1, change the Interrupt Priority to INT_PRIORITY_LEVEL_5.

If your application is going to be using graphics, as this one does, within *Graphics Stack*, select **Use Graphics Stack?**. Further down, within *Use Harmony Graphics Composer Suite? > Middleware > Use Aria User Interface Library?*, clear **Enable Touch** since the LCD on the PIC32 Bluetooth Audio Development Kit does not support a touchscreen. Since you are using the LCD, you will also need to go back to the Drivers section, and within *PMP*, select **Use PMP Driver?**. Expand that section, and change the value of Strobe Wait Sates to PMP_STROBE_WAIT_4.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the High-resolution Audio Demonstration.

Description

To build this project, you must open the `mac_audio_hi_res.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/audio/`mac_audio_hi_res.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`mac_audio_hi_res.X`	`<install-dir>/apps/audio/mac_audio_hi_res/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mz_ef_pim_bt_audio_dk	pic32mz_ef_pim+bt_audio_dk	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32 Bluetooth Audio Development Kit configured for Interrupt mode and dynamic operation. This configuration also requires the PIC32MZ2048EFH144 Audio PIM.

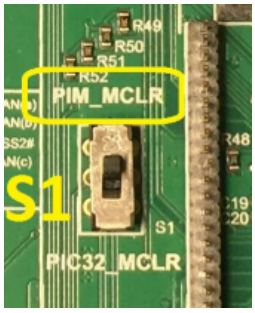
Configuring the Hardware

This section describes how to configure the supported hardware.

Description

PIC32 Bluetooth Audio Development Kit and PIC32MZ2048EFH144 Audio Plug-in Module (PIM)

1. Insert the PIC32MZ Audio PIM onto the PIC32 Bluetooth Audio Development Kit.
2. Switch S1 on the PIC32 Bluetooth Audio Development Kit (between the audio DAC daughter board and the microcontroller PIM) should set to PIM_MCLR.



3. Connect headphones to the jack on the Audio DAC Daughter Board, which is included with the PIC32 Bluetooth Audio Development Kit.

Running the Demonstration

This section provides instructions about how to build and run the High-resolution Audio Demonstration.

Description

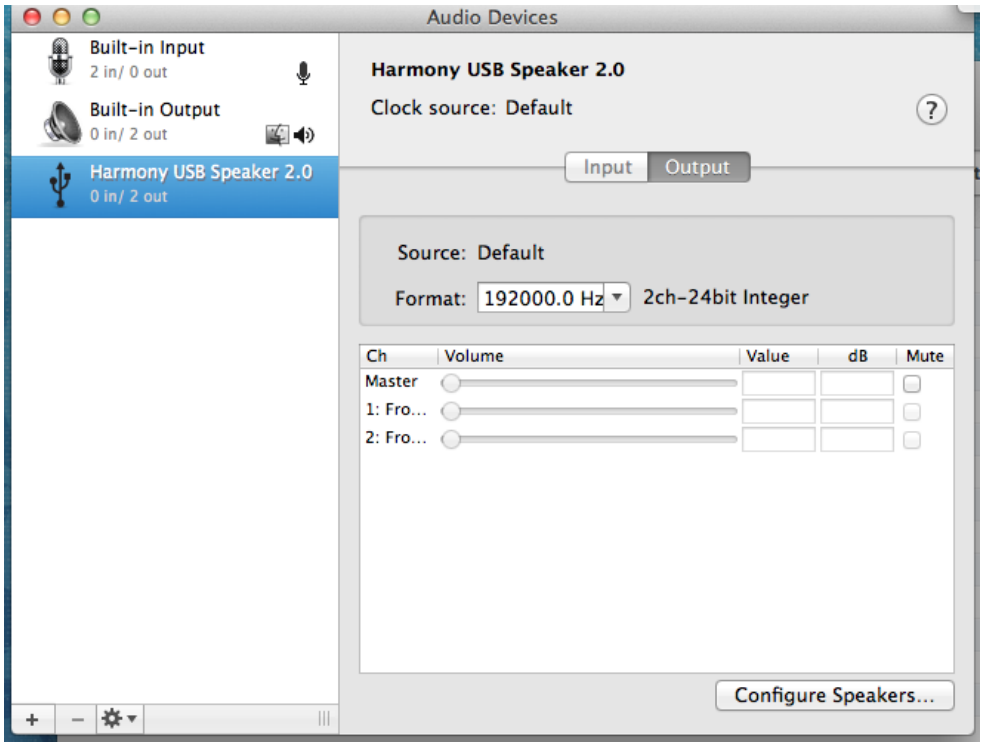
This demonstration functions as a speaker when plugged into a computer that supports USB Audio 2.0 devices.

Notes:
1. At the time of release, only Apple Mac personal computers natively support Audio 2.0 USB devices.
2. The demonstration has been tested with a third-party Audio 2.0 USB device driver on Windows® 7.

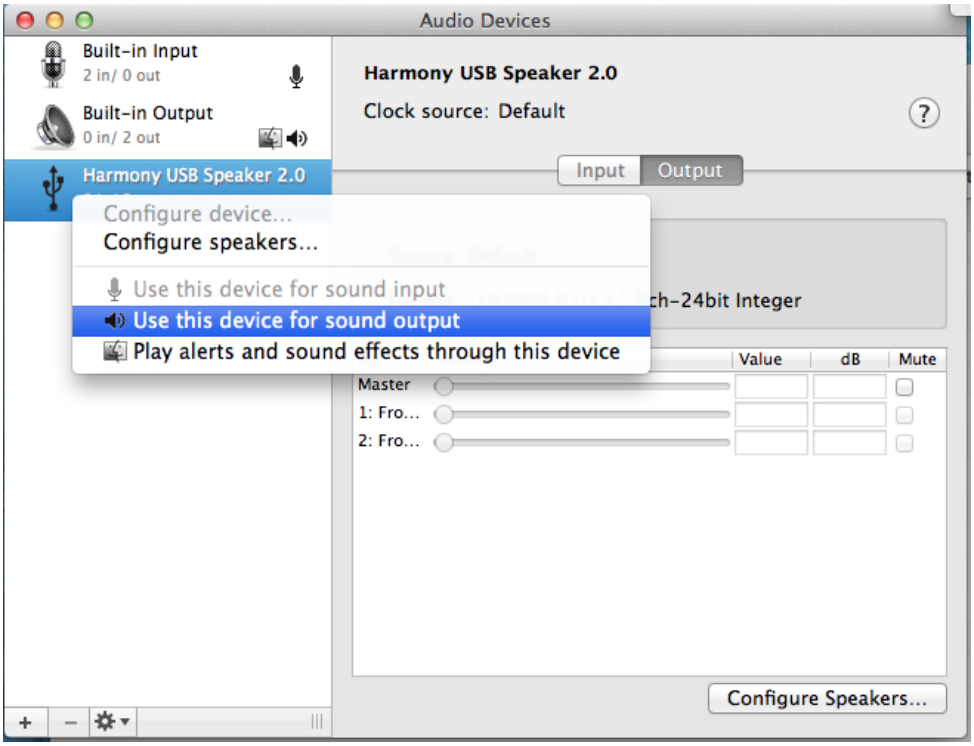
Do the following to run the demonstration:

1. Build the demonstration application and program the device.
2. Connect the device to a computer. For example, an Apple Mac book.
3. Use a feature of the computer that outputs sound to a speaker. On the Apple Mac book with OS X 10.9.4, the Audio MIDI Setup application could be used, as follows:

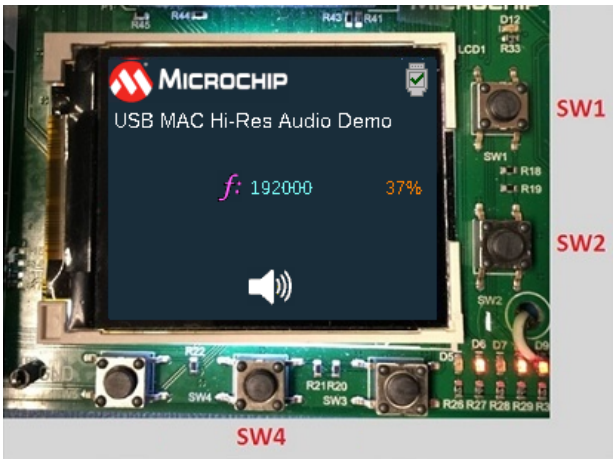
- Open Audio MIDI Setup



- Right click **Harmony USB Speaker 2.0**, which is listed in the left column
- Select **Use this device for sound output**



4. Open the Audio player (iTunes or VLC) and play the audio of your choice.
5. The feature unit only supports Mute control. Audio being played can also be muted using switch SW4 on the development board.
6. The audio volume can be controlled through the computer media player (iTunes or VLC on Apple Mac book) and also through the switches SW1 and SW2 on the development board.

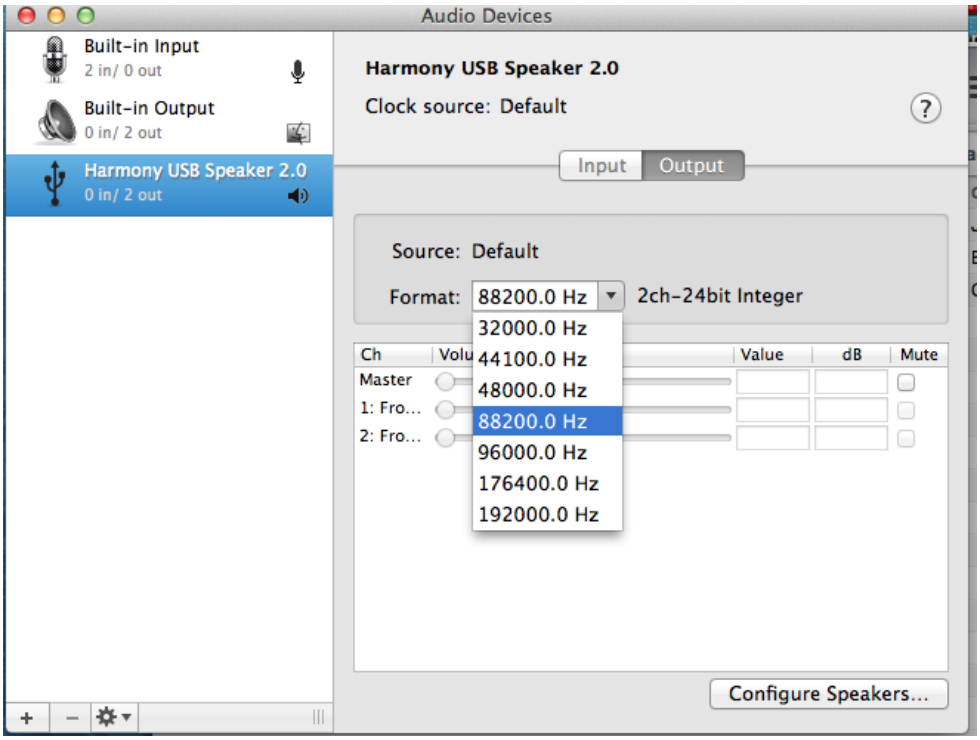


7. Note that some applications lock into a sound source when they open or close (such as some Web browsers or plug-ins). Therefore, if the speaker is plugged in with a Web page or an already playing video, the sound may not be redirected to the USB-based speakers until the browser is closed and reopened.
8. The audio device created in this demonstration has the following characteristics:
   - Supported sampling rates:
     - 32000 Hz
     - 44100 Hz
     - 48000 Hz
     - 88200 Hz
     - 96000 Hz
     - 176400 Hz
     - 192000 Hz
   - 2-Channel (Stereo)
   - PCM format (24 bits per sample)

Sampling Rate

The demonstration application allows the default sampling rate (set to 192000 Hz) to be changed. This can be done using the following procedure on an Apple Mac book.

1. If audio is being played, Stop it (PAUSE on iTunes).
2. Open the Audio MIDI Setup application.
3. Select Harmony USB Speaker 2.0 listed in the left column.
4. In the right pane, select the desired sampling rate from the *Format* drop-down menu, as shown in the following figure.



5. Verify that the sampling rate has changed on the display on the board.
6. Select 'PLAY' on the Audio player to play the audio with the changed sampling rate.

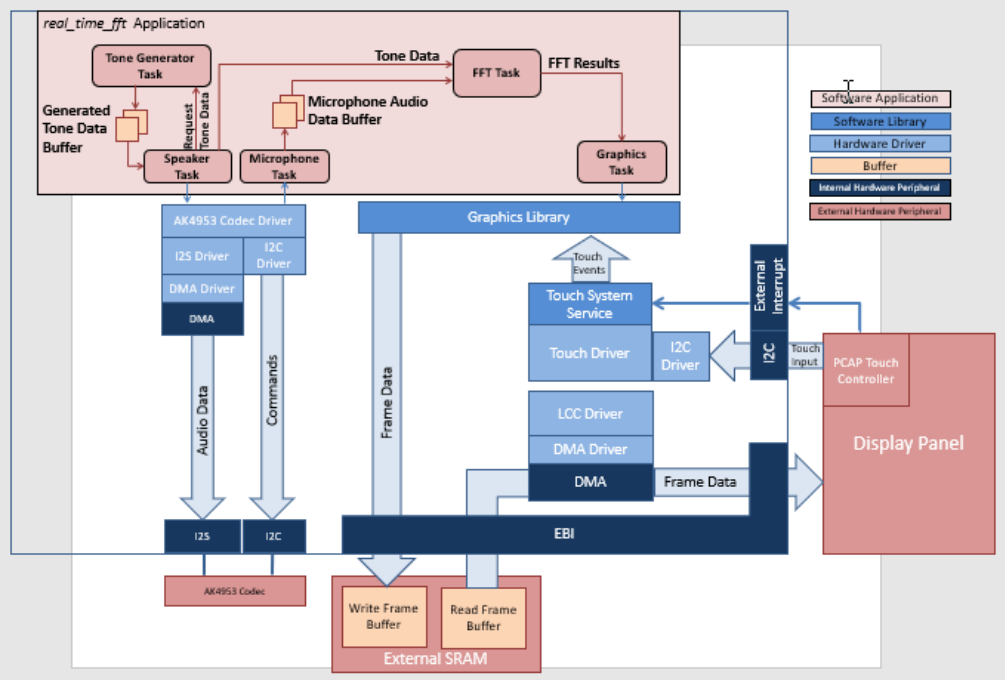
real_time_fft

This application demonstrates the usage of DSP Fixed-Point Math Library.

Description

This application analyzes the audio data received from the microphone and displays the spectrum of frequencies present in the audio data. The application also allows users to generate complex signals by mixing up to three pure sine tones of different frequencies and amplitude. The generated complex signal is then fed as an input to the FFT algorithm, and the frequency components present in the signal are displayed on the GUI. The application demonstrates the usage of windowing functions before the time domain signal is passed to the FFT algorithm.

Architecture



The real_time_fft application provides two modes of operation, MIC and Tone, selected through the GUI.

1. **MIC mode:** By default, the application is in the microphone mode. In this mode, the AK4953A codec is configured to sample the microphone data at 48000 Hz. (The codec driver internally uses the I2S driver with DMA for data interface and the I2C driver for the command interface). Using the AK4953A codec driver, the application reads chunks of 4096 samples (or 4096/48000 = 85.33 ms) of single channel microphone data and saves it in a buffer. This data is then passed to the FFT task for frequency analysis. The FFT transform assumes the signal to be continuous, as the input signal to the FFT algorithm contains nonintegral cycles of the signal, the sharp discontinuities spread out in the frequency domain resulting in spectral leakage. To reduce the spectral leakage, the data is first passed through a Hanning window function, using the windowing functions provided by the math library. The windowed output is then passed through the FFT function provided by the math library, which calculates the frequency components present in the time-domain signal. In this case, the FFT output will have a frequency resolution of 11.71875 Hz (?f = Fs/N = 48000/4096 = 11.71875 Hz). The frequency output is then represented in one-third octave frequency bands, with the entire frequency spectrum divided into 24 bands (100Hz, 125Hz … 16kHz, 20kHz). The center frequency of each band is calculated by multiplying the center frequency of previous band by $2^{1/3}$. The amplitude is displayed in the dBFS unit.
2. **Tone mode:** The Tone mode allows the user to generate complex signals by mixing up to three pure sine tones of different frequencies and amplitudes. This complex signal is passed through a window function (Hanning or Blackman window, selected by user through the GUI). The windowed signal is then passed to the FFT algorithm for frequency analysis. The FFT results on the display can be used to verify the frequency components present in the complex signal. The signal data is also sent to the codec and can be heard by connecting a headphone to the HP OUT on the MEB II board. The time domain view of the generated signal can also be viewed on the GUI.

Demonstration Features

- DSP Fixed-Point Math Library (see *Volume V: MPLAB Harmony Framework Reference > Math Libraries Help > DSP Fixed-Point Math Library*).
- AK4953 Codec Driver (see *Volume V: MPLAB Harmony Framework Reference > Driver Libraries Help > Codec Driver Libraries > AK4953 Codec Driver Library*).
- I2S Driver used by the Codec Driver for audio data transfer (see *Volume V: MPLAB Harmony Framework Reference > Driver Libraries Help > I2S Driver Library Help*).
- I2C Driver used by the Codec (for command transfer) and Touch Drivers (see *Volume V: MPLAB Harmony Framework Reference > Driver Libraries Help > I2C Driver Library Help*).
- DMA System Service (see *Volume V: MPLAB Harmony Framework Reference > System Service Libraries Help > Direct Memory Access (DMA) System Service Library*).
- Aria User Interface Library (see *Volume V: MPLAB Harmony Framework Reference > Graphics Libraries Help > MPLAB Harmony Graphics Composer (MHGC) Suite > Aria User Interface Library)*.
- MXT336T Touch Driver (*see Volume V: MPLAB Harmony Framework Reference > Driver Libraries Help > Touch Driver Libraries Help > mXT336T Touch Driver Library*).
- Low Cost Controllerless (LCC) Graphics Driver.

Tools Setup Differences

pic32mz_ef_sk_meb2 Configuration

MHC changes:

- The output of the System PLL is set to 198 MHz to accurately generate the REFCLKO1 which is set to 48000 x 256 = 12288000 Hz. This is done by setting the System PLL in the *MPLAB Harmony Configurator > Clock Diagram*. Here, click on the **Auto Calculate button** and set the desired system frequency to 198 MHz
- The MCKI (Master Clock Input) to the Codec is provided by the Reference Clock Generator Output1 (REFCLKO1) of the PIC32. The value for REFCLKO1 is set by navigating to the *MPLAB Harmony Configurator > Clock Diagram* and clicking on the **Auto Calculate button** under the REFCLKO1. Here, select the **Target I2S Input Frequency Radio button** and click on Apply. The REFCLKO1 will be set to 256 times the sampling frequency or 48000 x 256 = 12288000 Hz.

- The REFCLKO1 signal is mapped to PIN 70 (RD15). This is done in the *MPLAB Harmony Configurator > Pin Table*.
- The touch driver uses the PIC32's "External Interrupt 1" (INT1) signal for touch events. The "External Interrupt 1" signal is mapped to pin RE8 (Pin 23). This is done in the *MPLAB Harmony Configurator > Pin Table*.
- Under Harmony Framework Configuration select **Use Graphics stack?** Under *Graphics Controller>Low Cost Controllerless* set **Frame Buffer Mode** to **Double Buffer**. Expand **Draw Settings>Draw Count Limited** and set the **Max Draw Count Per DMA Transmit** to 100. Expand Memory Settings and set the **Memory Interface Mode** to External Memory.
- Under *Harmony Framework Configuration>Math Library>DSP Fixed-Point Math Library Configuration*, select **Use DSP Fixed-Point Math Library**?
- Under **Harmony Framework Configuration>Drivers>CODEC select Use Codec AK4953?** And set the number of AK4953 Driver Clients to 2. One client will transmit data to the codec (speaker output), and the other client will receive data from the codec (microphone input). The codec will use I2S driver instance 0 and I2C driver instance 0.
- Under *Harmony Framework Configuration>Drivers>Input Drivers>Touch Drivers>Use MXT336T Driver?* set the I2C driver module index to **DRV_I2C_INDEX_1** to use I2C driver instance 1.
- Under *Harmony Framework Configuration>Drivers>I2C>Use I2C Driver?* set the number of I2C Driver Instances to 2 and select **Include Force Write I2C Function**. This is needed by the AK4953 CODEC since it NACKs I2C data during the initialization sequence. Expand I2C Driver Instance 0 and set the I2C Module ID to `I2C_ID_2`. Set the Bit Bang Timer Source to `TIMER_ID_8`. Expand I2C Driver Instance 1 and set `I2C CLOCK FREQUENCY` (Hz) to 10000.
- Under **Harmony Framework Configuration>Drivers>I2S>Use I2S Driver,** select DMA Mode and select **Transmit DMA Support** and **Receive DMA Support**. Set the number of I2S Driver Clients to 2. Expand I2S Driver Instance 0 and set Audio Communication Width to `SPI_AUDIO_COMMUNICATION_16DATA_16FIFO_32CHANNEL` and Audio Mode to `SPI_AUDIO_TRANSMIT_MONO`. Set Queue Size Transmit to 5 and Queue Size Receive to 3. Set Receive DMA Channel Instance to 1.
- Under *Harmony Framework Configuration>System Services>DMA* set the number of DMA Channel Instances to 2. DMA Instance 0 will be used for the I2S data transmission, and DMA Instance 1 will be used for I2S data reception. Expand DMA System Channel Instance 0 and set the DMA Channel to use `DMA_CHANNEL_2`. Expand DMA System Channel Instance 1 and set the DMA Channel to use `DMA_CHANNEL_3`.
- The heap size is set to 60,000 to ensure enough memory for dynamic memory allocations. This is done in the MPLAB Harmony Configurator (MHC) by setting the size in *Device & Project Configuration >Project Configuration >XC32 (Global Options) >xc32-ld >General*.

Building the Application

This section identifies the MPLAB X IDE project name and location, and lists and describes the available configurations for the real time fft demo.

Description

To build this project, you must open the `real_time_fft.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is *<install-dir>/apps/audio/real_time_fft*.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`real_time_fft.x`	*<install-dir>/apps/audio/real_time_fft/firmware*

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within *./firmware/src/system_config*.

Project Configuration Name	BSP Used	Description
pic32mz_ef_sk_meb2_ext	pic32mz_ef_sk+meb2	This configuration runs on the PIC32MZ EF Starter Kit connected to the MEB II. The Codec is interfaced over I2C for command and I2S for data and uses DMA for data transfers. The Codec is configured for 16-bit data width and 48kHz sampling frequency. The graphical display is driven by the LCC driver and uses DMA for data transfers. The touch screen driver is interfaced using I2C configured for Interrupt mode and dynamic operation.

Configuring the Hardware

This section describes how to configure the supported hardware.

Description

Configuration: PIC32MZ EF Starter Kit and MEB II

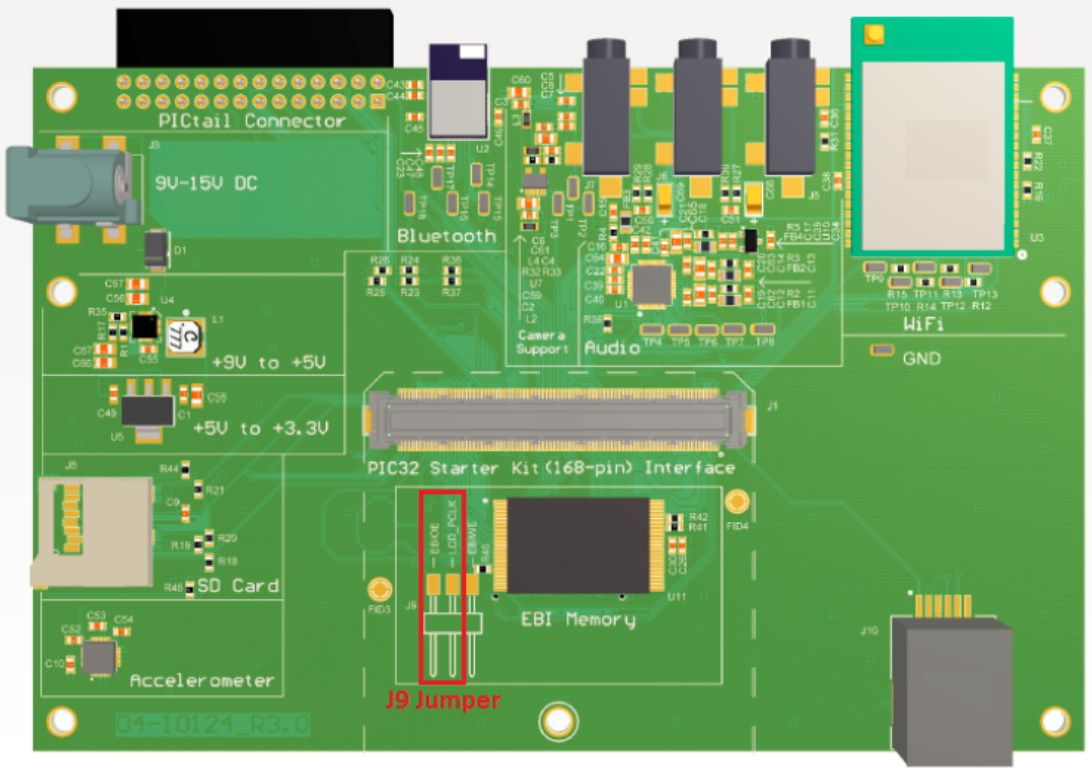
• On the MEB II, the EBIOE and LCD_PCLK (J9) must be closed. The jumper (J9) is available on the bottom side of the MEB II board. See the following figure for the jumper location.

• Connect the PIC32MZ EF Starter Kit board to the MEB II board

• Power up the board by connecting the power adapter to the J3 power connector on the MEB II board, or through a USB cable to the USB DEBUG port J3 on the Starter Kit board

• If the demo is in MIC mode, connect a microphone to the microphone input (MIC IN) on the MEB II

• If the demo is in TONE mode, a headphone may optionally be connected to the headphone output (HP OUT) on the MEB II board

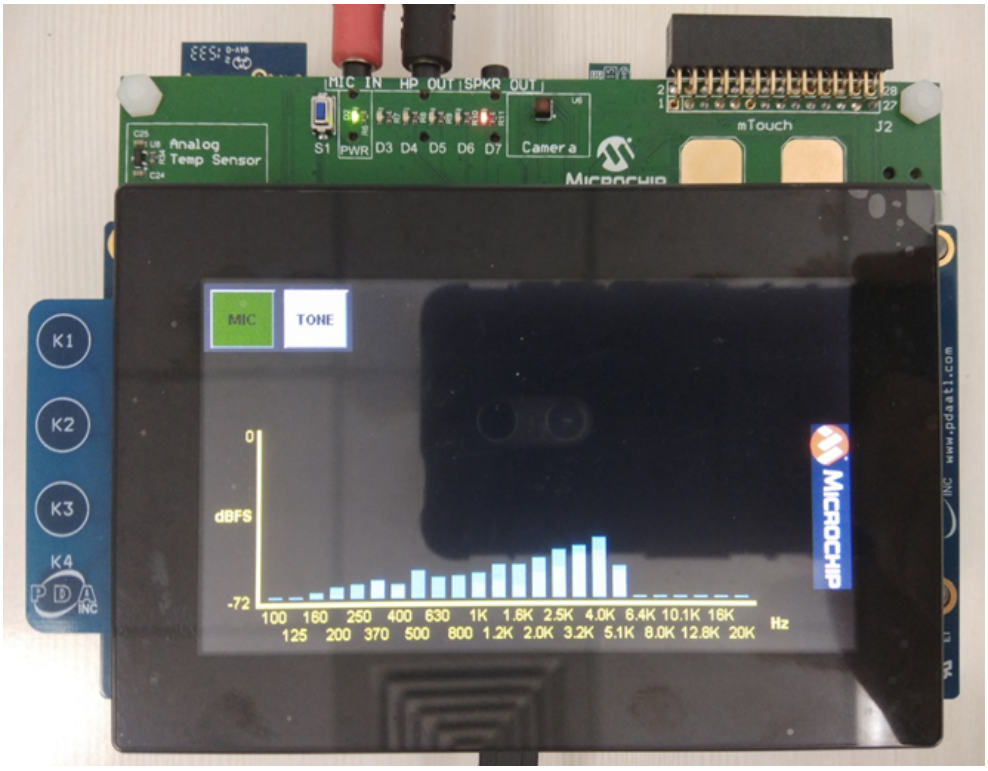


Running the Demonstration

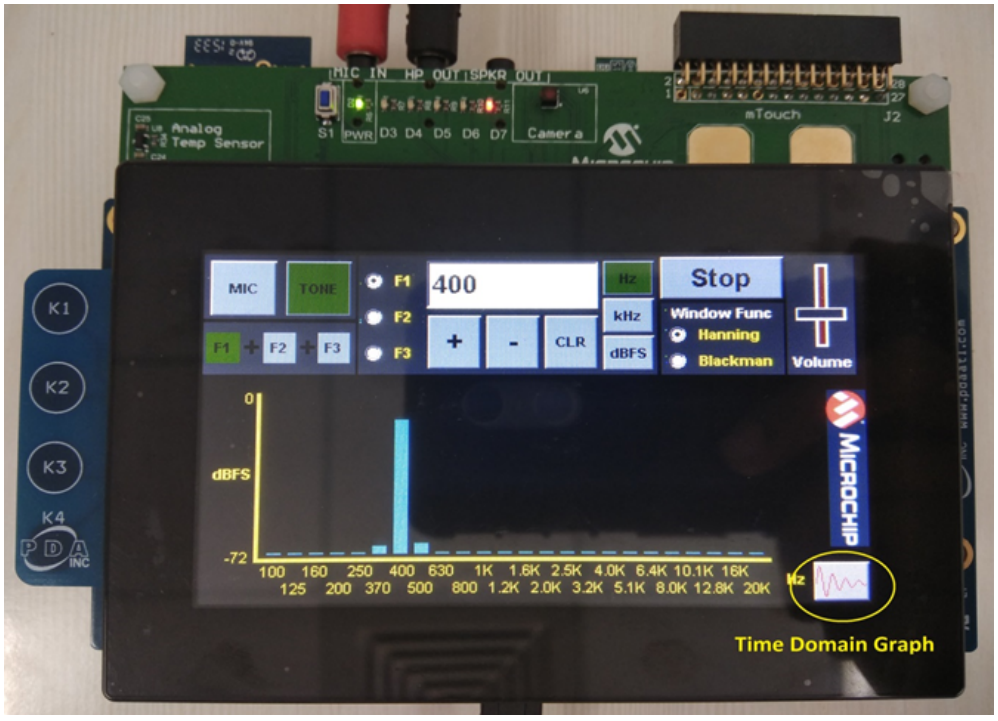
This section provides instructions on how to use the real time fft application.

Description

After the board powers up, the GUI should appear as shown in the following figure. By default, the application is in the MIC mode.

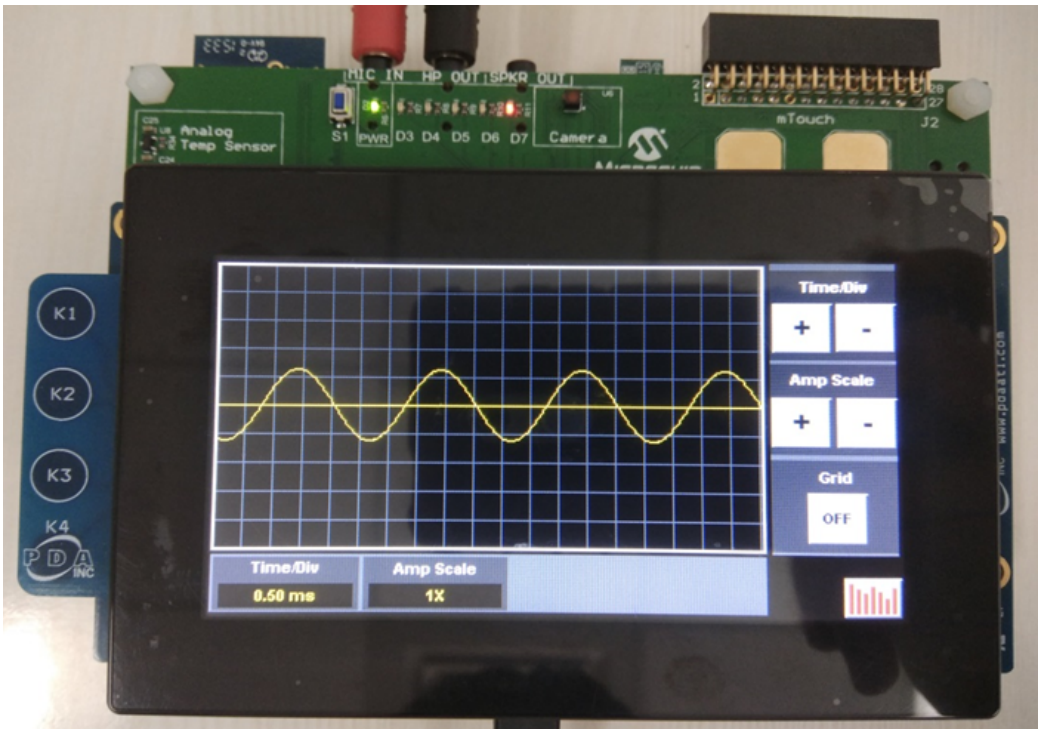


1. In MIC mode, speak into the microphone connected to the microphone input (MIC IN) on the MEB II board, and observe the amplitude of the frequency components present in the audio input on the GUI.
2. Enter Tone mode, by pressing the **TONE** button. In Tone mode, the GUI should appear as shown in the following figure.



3. Generate a complex signal by mixing sine tones of different frequency and amplitude. Three sine tones can be added by selecting F1, F2 and F3. The frequency for each sine tone can be set by pressing the Hz or kHz button and then pressing the **plus** or **minus** buttons to increase or decrease the frequency. Similarly, the amplitude can be set by pressing the dBFS button. By default, the amplitude of the tone is set to -12 dBFS.
4. Hanning or Blackman window function may be applied by selecting the **Hanning** or **Blackman** radio buttons.
5. Once done, press the **Play** button to start the generation of the complex signal. The generated tone can be heard by connecting a headphone to the headphone output (HP OUT) on the MEB II board.
6. Verify that the GUI shows the different frequency components present in the generated tone.
7. The time domain view of the generated signal can be seen by pressing the **Graph** button located on the bottom right of the screen. Navigate

back to the Tone mode screen by pressing the Graph button again.



Notes:

1. In **Tone** mode, if the generated signal is saturated, a message is displayed saying “Signal saturated. Lower the signal amplitude”. When the signal is saturated the FFT output may contain several high frequency components. Lowering the signal amplitude will remove the message on the GUI.
2. In **Tone** mode, the signal generation will stop after 300 seconds.

sdcard_player

This section demonstrates an SD card Audio Player for .WAV audio files.

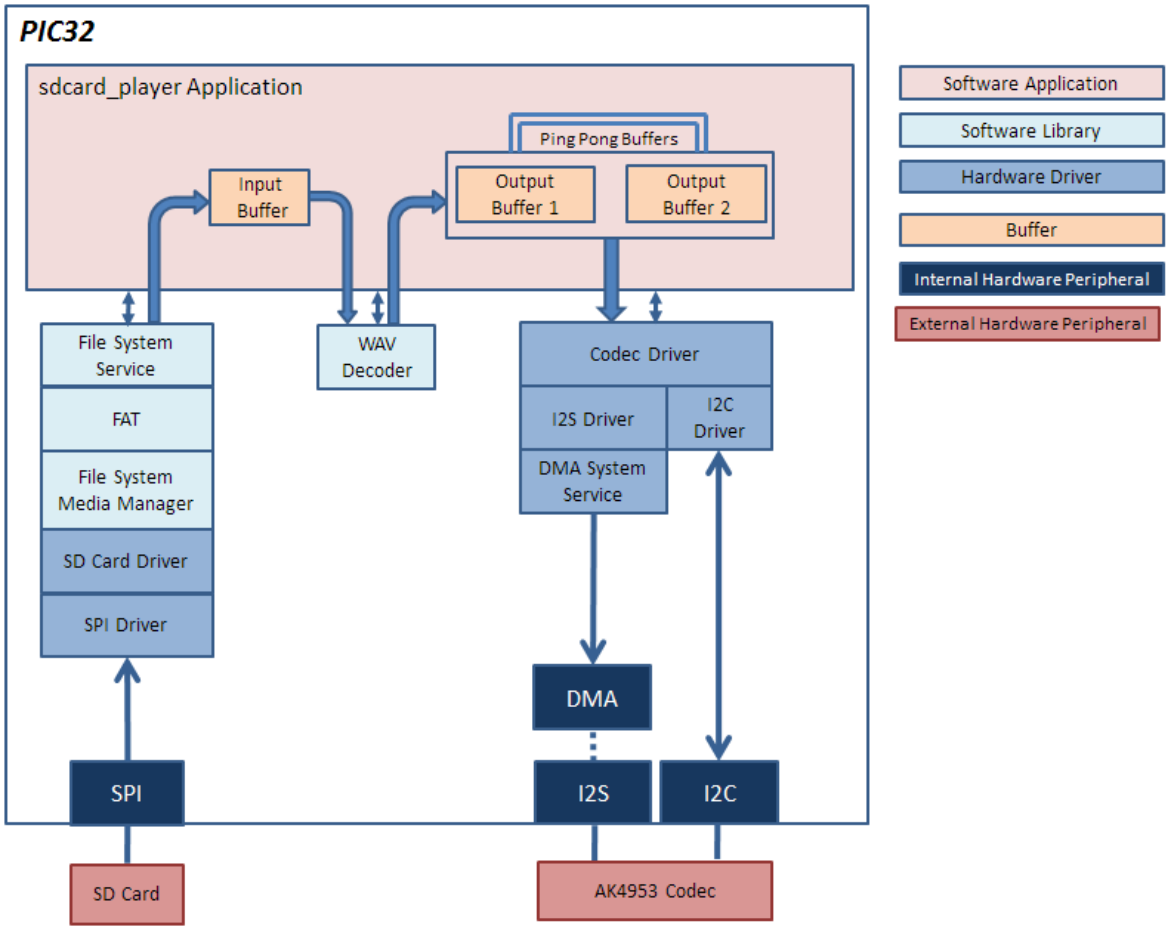
Description

The demonstration application creates an audio player that reads audio files (.WAV format only) from an SD card mounted on the click interface. The audio is played through the CODEC placed on the X32 header interface. It also provides feature to switch to the next track on the media.

Architecture

The sdcard_player application plays `.wav` files from the SD Card. The SD Card driver uses the SPI driver to interact with the SD Card. The application uses the File System Service to read/write data on the SD Card. The audio data read from the SD card is decoded by passing it to the WAV decoder. The decoded output is saved in the output buffers 1 and 2 which are used in ping pong manner. The output buffers 1 and 2 are submitted to the Codec driver for playing.

The Codec is initially configured for 16-bit data and 48 kHz sampling frequency. The sampling frequency is changed on the fly based on the media sampling frequency. The Codec driver sends the audio data to the AK4642 Codec using the I2S driver, which in turn uses DMA to transfer the audio data. The Codec driver uses the I2C driver to send commands to the AK4642 Codec.



Tools Setup Differences

- The Master Clock Input (MCKI) to the Codec is provided by Reference Clock Generator Output1 (REFCLKO1) of PIC32. The value for REFCLKO1 is set by navigating to the *MPLAB Harmony Configurator* > *Clock Diagram* and selecting **Auto Calculate** under REFCLKO1. Here, select **Target I2S Input Frequency** and click **Apply**. The REFCLKO1 will be set to 256 times the sampling frequency or 48000 x 256 = 12288000 Hz.
- The output of the System PLL is set to 198 MHz for the MZ Curiosity configuration and 80 MHz for the MX Curiosity Configuration to accurately generate the REFCLKO1, which is set to 48000 x 256 = 12288000 Hz. This is done by setting the System PLL in the *MPLAB Harmony Configurator* > *Clock Diagram*. Here, select **Auto Calculate** and set the “Desired System Frequency” to 198 MHz for the MZ Curiosity configuration and 80 MHz for the MX Curiosity configuration.
- The REFCLKO1 signal is mapped to Pin 21 (RB8) on MX Curiosity Board and Pin3 (RE5) on MZ Curiosity Board. This is done through the *MPLAB Harmony Configurator* > *Pin Table*.
- The heap size is set to 4096 bytes to ensure enough memory for dynamic allocations. This is done by setting the size in MHC through the following menu selection: *Device & Project Configuration* > *Project Configuration* > *XC32 (Global Options)* > *xc32-ld* >*General*.

Demonstration Features

- AK4642 Codec Driver
- SD Card Driver
- File System Service
- WAV Decoder
- SPI Driver used by SD Card Driver
- I2S Driver used by Codec Driver for audio data transfer
- I2C Driver used by Codec (for command transfer)
- DMA System Service

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the SD card audio player demonstration.

Description

To build this project, you must open the sdcard_player.X project in MPLAB X IDE, and then select the desired configuration. The following tables list and describe the project and supported configurations. The parent folder for these files is

`<install-dir>/apps/audio/sdcard_player.`

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`sdcard_player.X`	`<install-dir>/apps/audio/sdcard_player/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx470_curiosity	pic32mx470_curiosity	Select this MPLAB X IDE project configuration to run the demonstration application to run on the PIC32MX470 Curiosity Development Board, with the PIC32MX470F512H microcontroller. This configuration uses "microSD click" from MikroElektronika mounted on the mikroBUS header interface and PIC32 Audio Codec Daughter Card - AK4642EN mounted on the X32 header interface.
pic32mz_ef_curiosity	pic32mz_ef_curiosity	Select this MPLAB X IDE project configuration to run the demonstration application to run on the PIC32MZ EF Curiosity Development Board, with the PIC32MZ2048EFM100 microcontroller. This configuration uses "microSD click" from MikroElektronika mounted on the mikroBUS header interface and PIC32 Audio Codec Daughter Card - AK4642EN mounted on the X32 header interface.

Configuring the Hardware

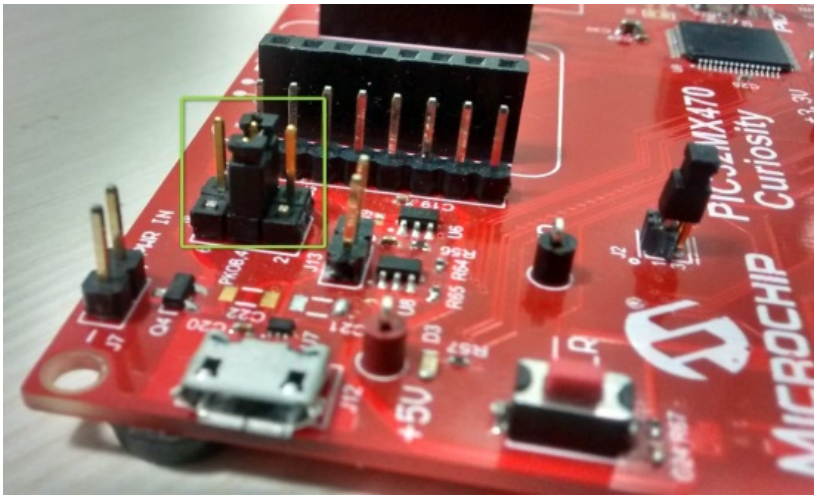
This section describes how to configure the supported hardware.

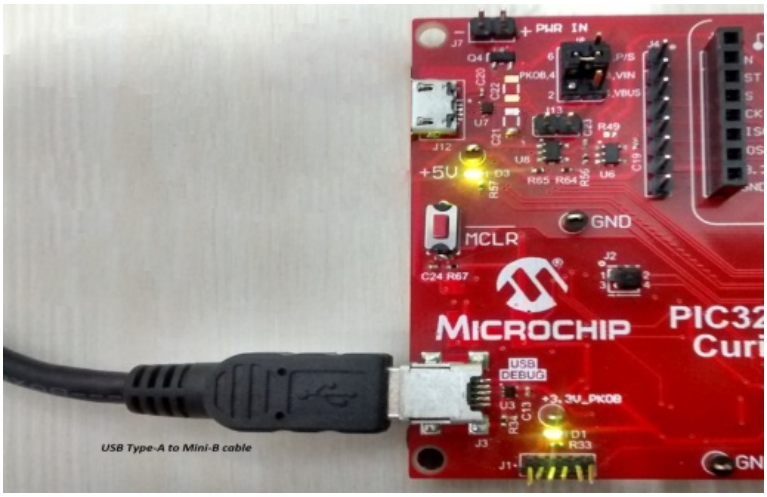
Description

PIC32MX470 Curiosity Development Board

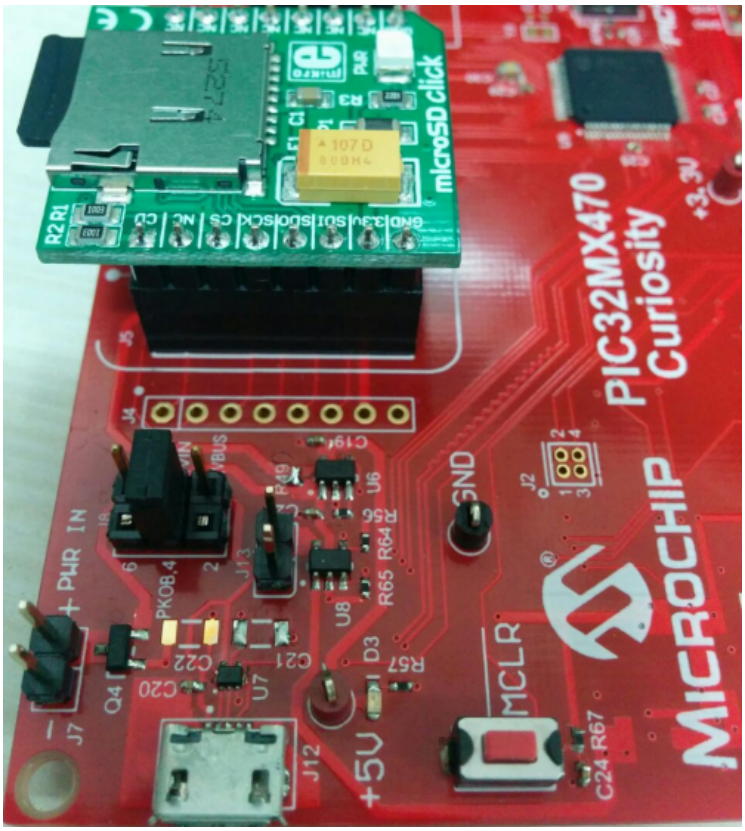
Configuration: pic32mx470_curiosity

1. Ensure that a jumper is placed at 4-3 on J8, to select supply from debug USB connector.
2. Power the PIC32MX470 Curiosity Development Board from a Host PC through a Type-A male to mini-B USB cable connected to Mini-B port (J3).

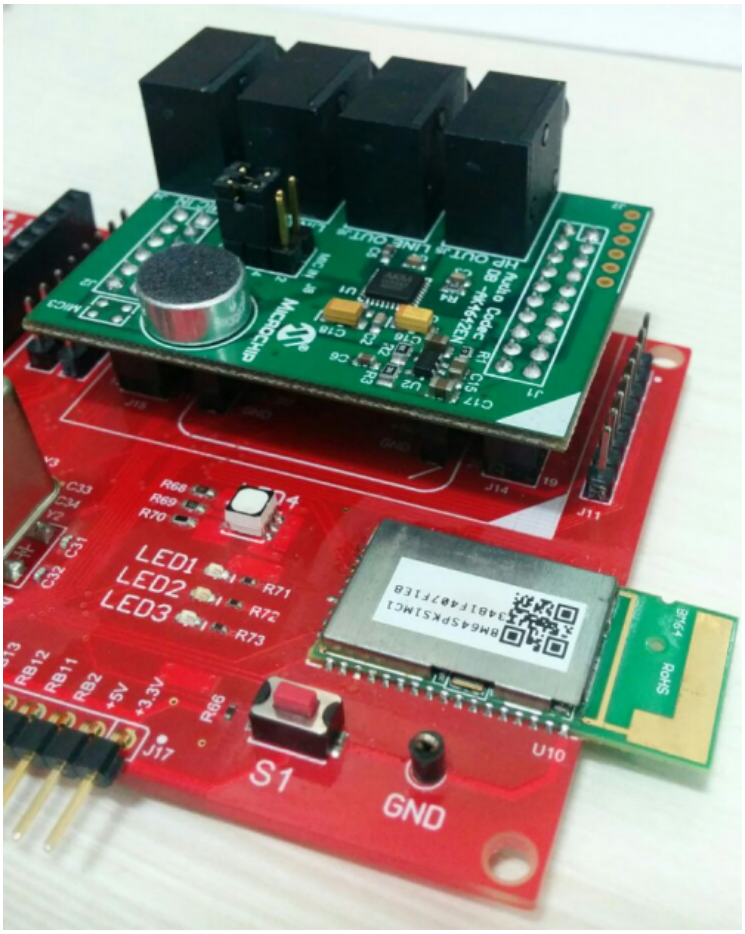




3. Mount the SD Click board, "microSD click" from MikroElektronika (http://www.mikroe.com/click/microsd/) on the mikro bus interface J5.
4. Plug a micro SD card into the microSD click board card slot.



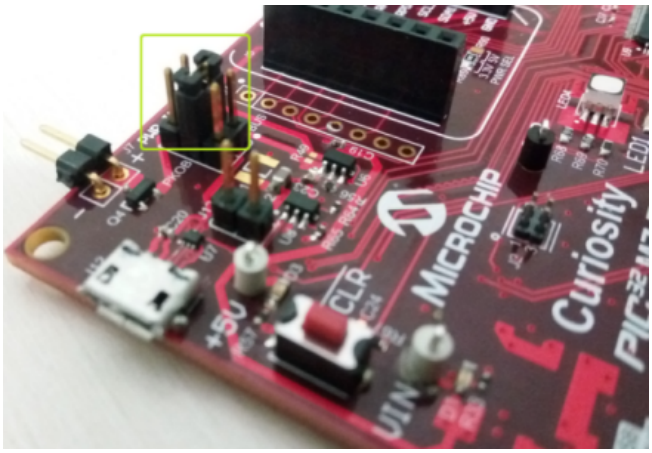
5. Mount the is PIC32 Audio Codec Daughter Card - AK4642EN (http://www.microchip.com/DevelopmentTools/ProductDetails.aspx?PartNO=ac320100) on the X32 interface header (J14, J15)

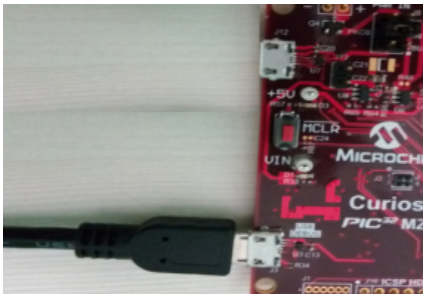


PIC32MZ EF Curiosity Development Board

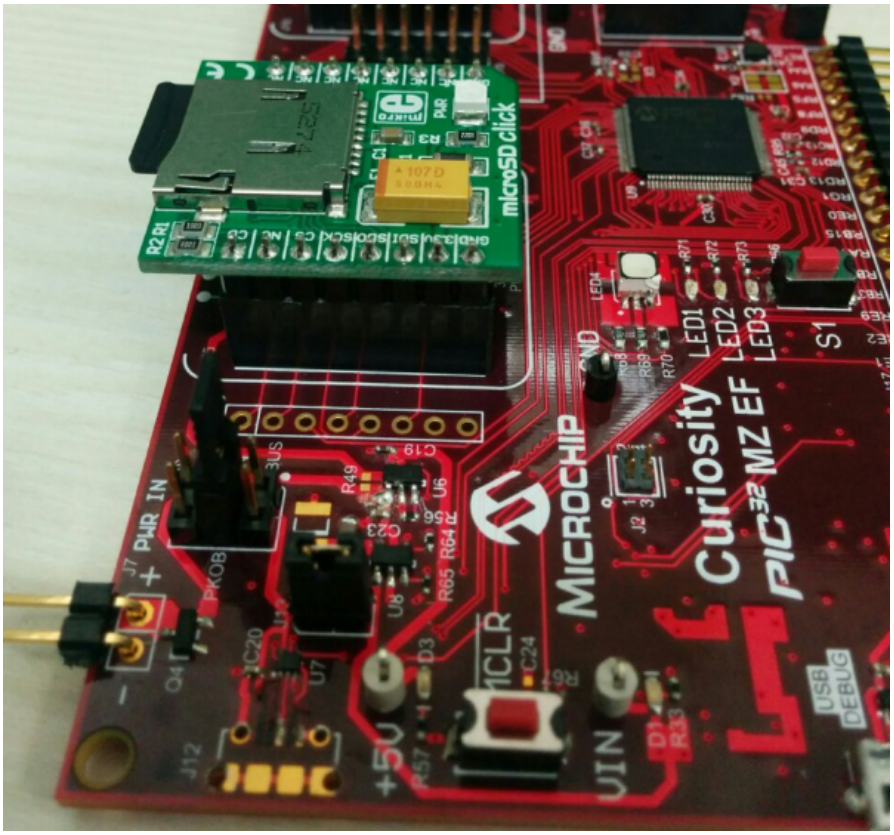
Configuration: pic32mz_ef_curiosity

1. Ensure that a jumper is placed at 4-3 on J8, to select supply from debug USB connector.
2. Power the PIC32MZ EF Curiosity Development Board from a Host PC through a Type-A male to micro-B USB cable connected to Micro-B port (J3).

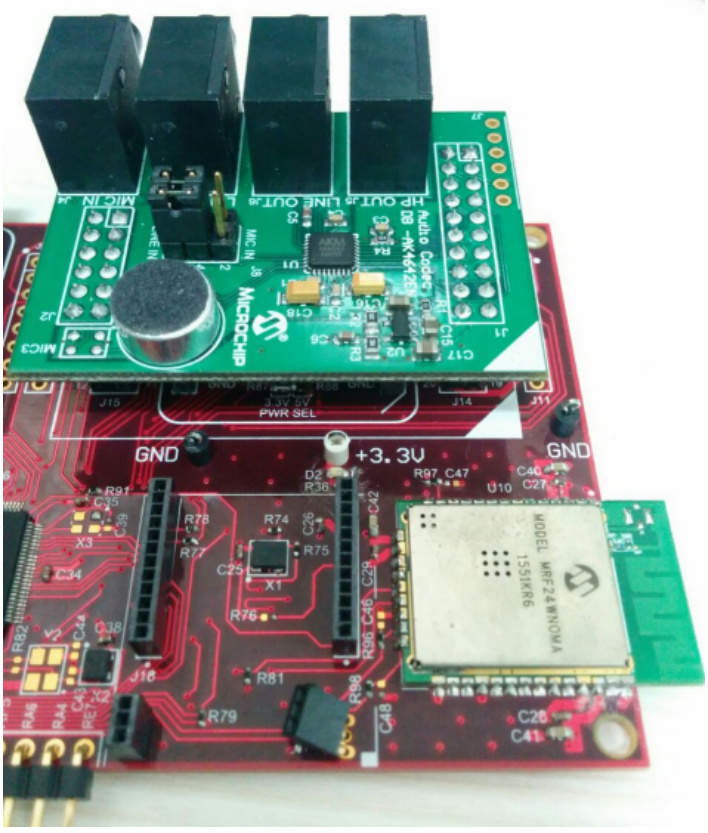




3. Mount the SD Click board, ‘microSD click’ from MikroElektronika (http://www.mikroe.com/click/microsd/) on the mikro bus interface J5.
4. Insert a micro SD card into the microSD click board card slot.



5. Mount the is PIC32 Audio Codec Daughter Card - AK4642EN (http://www.microchip.com/DevelopmentTools/ProductDetails.aspx?PartNO=ac320100) on the X32 interface header (J14, J15).



Running the Demonstration

This section provides instructions about how to build and run the SD card Audio Player demonstration.

Description

Important! Prior to using this demonstration, it is recommended to review the MPLAB Harmony Release Notes for any known issues. A PDF copy of the release notes is provided in the `<install-dir>/doc` folder of your installation.

In MPLAB X IDE, compile and program the target device. While the target device is compiling, select the appropriate MPLAB X IDE project configuration for the demonstration board. These configurations set the target processor and the board to be used in the output interface.

This demonstration plays WAV audio files stored on the SD card.

Refer to Building the Application for details.

Run the demonstration as follows:

1. Insert your micro SD card into the SD card slot on the micro SD click board. Ensure that the SD card contains WAV audio files.
2. Connect speaker or headphone to the headphone out (HP OUT) connector on top of the PIC32 Audio Codec Daughter Card.
3. Connect power to the board. After the board powers up, the first WAV audio track on the media is played, indicated by the glowing of LED1 on the board.
4. Plug in the headphone and you should be able to hear the audio track being played from the SD card.
5. Switch to the next track on the media by pressing button S1. The changing of the track is indicated by the toggling of LED3 on the board.

sdcard_usb_audio

This topic demonstrates playback of audio files stored on a SD card and audio data streamed over a USB interface.

Description

This application demonstrates an audio player application by playing audio files stored on a SD card. This demonstration also acts as a USB speaker with audio data streaming from a personal computer to PIC32 device.

Full-Speed USB is used for communication between the host computer and PIC32 device. The application also provides a Graphical User Interface with touch screen support to access and randomly select media tracks, and also provides controls to increase or decrease volume and mute or unmute the audio output. Additionally, the demonstration provides an option to select the media source, either a SD card or USB. The application supports playing 48 kHz, 16-bit audio.

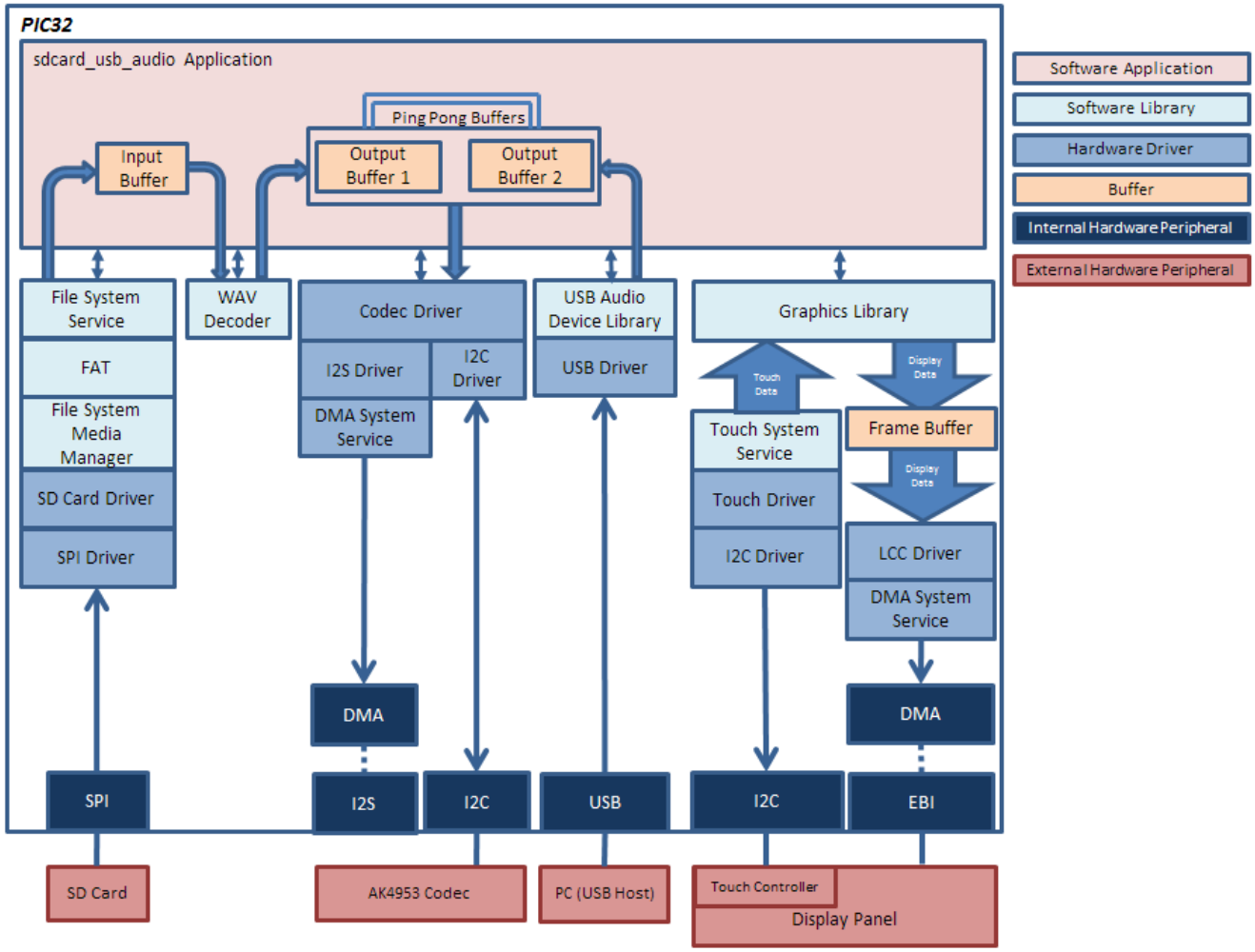
Note: The Audio Player application only support playback of WAVE (`.wav`) files.

Architecture

The sdcard_usb_audio application uses Graphics Library to render graphics to the display. The graphics library draws the widgets and images into the internal frame buffer. This frame buffer is transferred to the LCD display panel by the Low Cost Controllerless (LCC) driver using the DMA. Touch input from the touch controller goes through the I2C port, and the Touch System Service acquires the touch input information from the Touch and I2C drivers. The Touch System Service sends touch events to the Graphics Library, which processes these events and updates the frame data accordingly.

The application plays WAV files from the SD Card. The SD Card driver uses the SPI driver to interact with the SD Card. The application uses the File System Service to read/write data on the SD Card. The audio data read from the SD card is decoded by passing it to the WAV decoder. The decoded output is saved in the output buffers 1 and 2 which are used in ping pong manner. The output buffers 1 and 2 are submitted to the Codec driver for playing. Similarly, the audio data read from the USB host is saved in ping pong buffers which are then submitted to the Codec driver for playing.

The Codec is configured for 16-bit data and 48 kHz sampling frequency. The Codec driver sends the audio data to the AK4953 Codec using the I2S driver which in turn uses DMA to transfer the audio data. The Codec driver uses the I2C driver to send commands to the AK4953 Codec.



Demonstration Features

- AK4953 Codec Driver
- SD Card Driver
- USB Device (Audio Class) Library
- MTCH6301 Touch Driver Library
- File System Service
- WAV Decoder
- SPI Driver used by SD Card Driver
- I2S Driver used by Codec Driver for audio data transfer
- I2C Driver used by Codec (for command transfer)
- DMA System Service

Tools Setup Differences

- The Master Clock Input (MCKI) to the Codec is provided by Reference Clock Generator Output1 (REFCLKO1) of PIC32. The value for REFCLKO1 is set by navigating to *MPLAB Harmony Configurator* > *Clock Diagram* and selecting **Auto Calculate** under REFCLKO1. Here, select **Target I2S Input Frequency** and click **Apply**. The REFCLKO1 will be set to 256 times the sampling frequency or 48000 x 256 = 12288000 Hz.
- The output of the System PLL is set to 198 MHz to accurately generate the REFCLKO1, which is set to 48000 x 256 = 12288000 Hz. This is done by setting the System PLL in *MPLAB Harmony Configurator* > *Clock Diagram*. Here, select **Auto Calculate** and set the “Desired System Frequency” to 198 MHz
- The REFCLKO1 signal is mapped to PIN 70 (RD15). This is done through the *MPLAB Harmony Configurator* > *Pin Table*.
- The touch driver uses the PIC32's "External Interrupt 1" (INT1) signal for touch events. The "External Interrupt 1" signal is mapped to pin RE8 (Pin 23). This is done through the *MPLAB Harmony Configurator* > *Pin Table*.
- The “Include Force Write I2C Function” is selected in *Harmony Framework Configuration* > *Drivers* > *I2C*. This will include an API that sends data to the slave even if the slave NACKs data. This is needed by the AK4953 CODEC since it NACKs I2C data during the initialization

sequence.

- The heap size is set to 100000 bytes to ensure enough memory for dynamic allocations. This is done by setting the size in MHC through the following menu selection: *Device & Project Configuration > Project Configuration > XC32 (Global Options) > xc32-ld > General*.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the demonstration.

Description

To build this project, you must open the `sdcard_usb_audio.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/audio/sdcard_usb_audio`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`sdcard_usb_audio.X`	`<install-dir>/apps/audio/sdcard_usb_audio/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mz_ef_sk_meb2_legacy	pic32mz_ef_sk+meb2_legacy	This configuration runs on the PIC32MZ EF Starter Kit connected to the legacy MEB II. The micro SD card is used in SPI mode and is configured for Interrupt mode and dynamic operation. The USB library is configured for Full-Speed operation in Audio Device mode. The Codec is interfaced over I2C for command and I2S for data and uses DMA for data transfers. The Codec is also configured for 16-bit data width and 4 8kHz sampling frequency. The graphical display is driven by the LCC driver and uses DMA for data transfers. The touch screen driver is interfaced using I2C configured for Interrupt mode and dynamic operation.

Configuring the Hardware

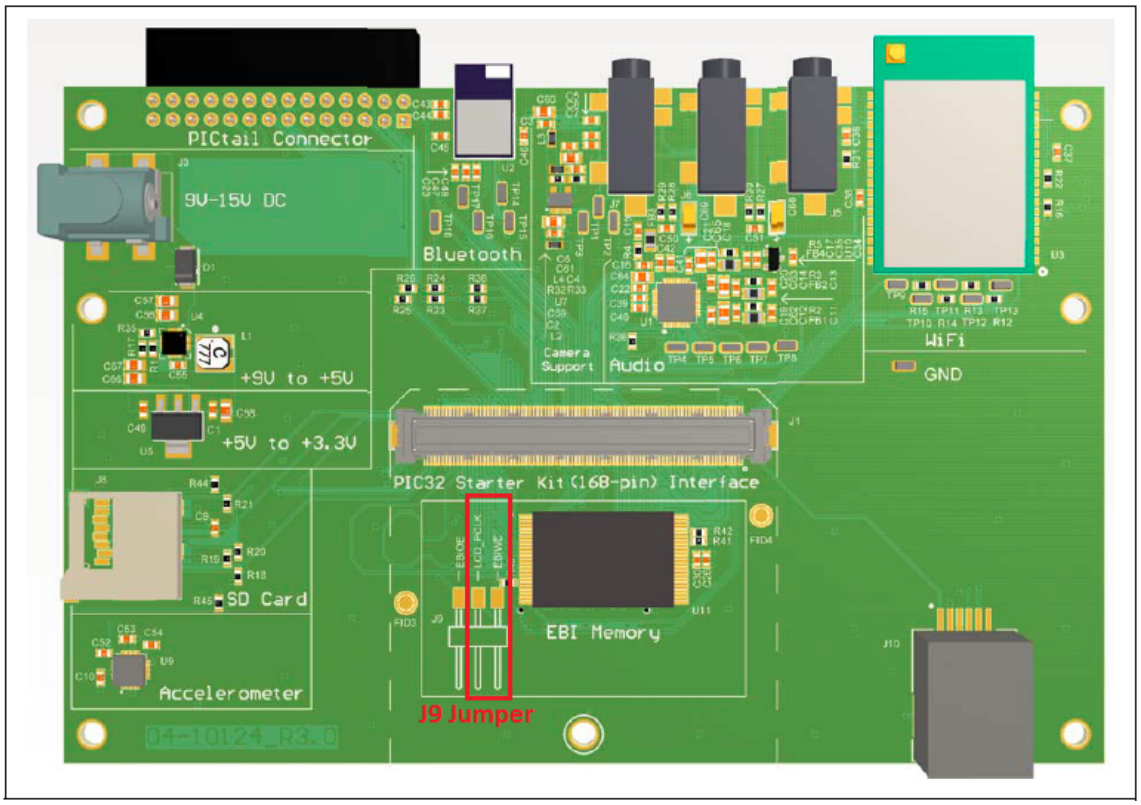
This section describes how to configure the supported hardware.

Description

PIC32MZ EF Starter Kit and Legacy MEB II

Configuration: pic32mz_ef_sk_meb2_legacy

- On the MEB II, the EBIWE and LCD_PCLK (J9) must be closed. The jumper (J9) is available on the bottom side of the MEB II. See the following figure for jumper location.
- Connect the PIC32MZ EF Starter Kit board to the MEB II board
- Insert your micro SD card into the SD card slot (J8) on the MEB II board. Ensure that the SD card contains `.wav` audio files.
- Connect speakers or headphones to the headphone out (HP OUT) on the MEB II.
- Connect a micro-B USB cable between the port J4 on the PIC32MZ EF Starter Kit and the Host computer.
- Power up the board by connecting the power adapter to J3 power connector on the MEB II board or through a USB cable to the USB DEBUG port J3 on the Starter Kit board.



Running the Demonstration

This section demonstrates how to run the demonstration.

Description

Note: Prior to using this demonstration, it is recommended to review the MPLAB Harmony Release Notes for any known issues. A PDF copy of the release notes is provided in the `<install-dir>/doc` folder of your installation.

In MPLAB X IDE, compile and program the target device. While compiling, select the appropriate MPLAB X IDE project configuration for the demonstration board. These configurations set the target processor and the board, to be used in the output interface.

This demonstration plays .wav audio files stored on SD card as the storage media, when the source of audio is selected as SD card (default audio source).

When the source of audio is selected as USB, the demonstration plays audio data streamed by PC over a USB interface. The device can then be used as USB speaker.

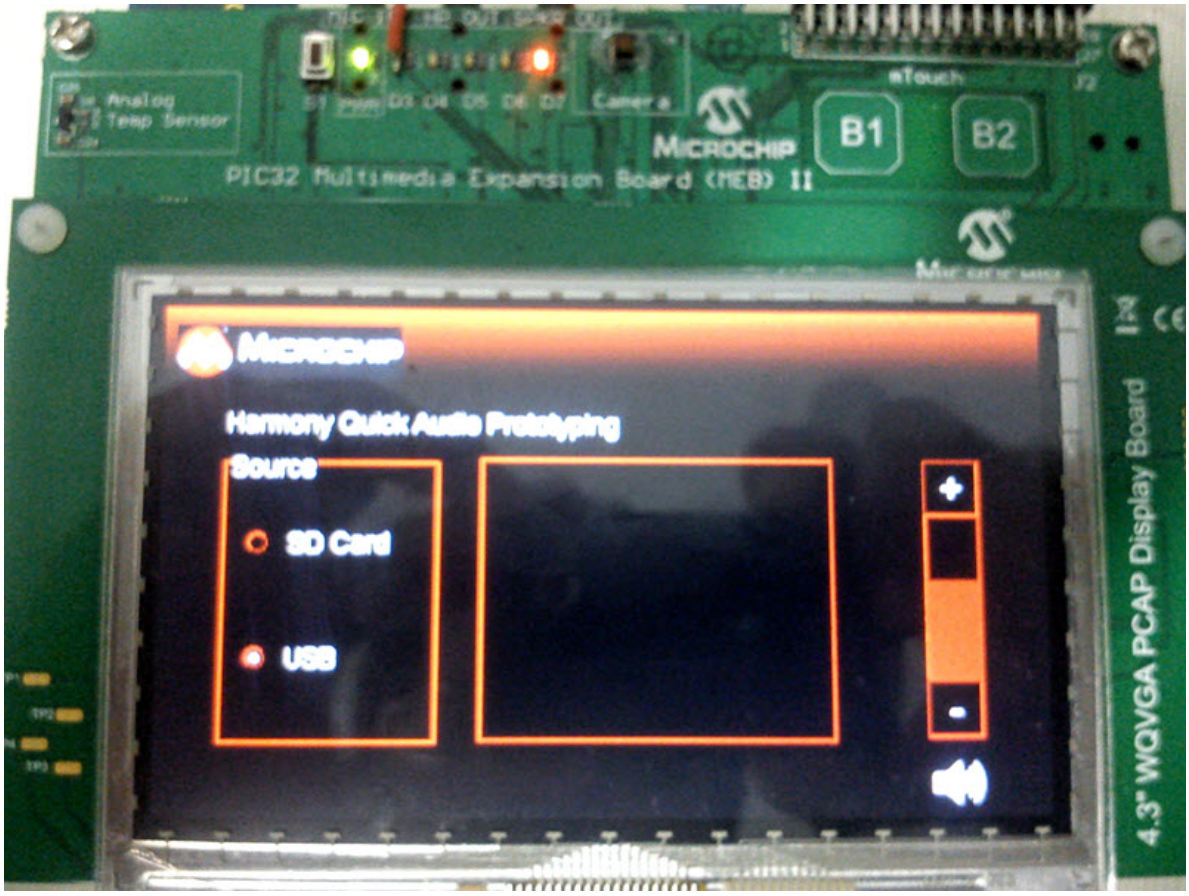
Refer to Building the Application for details.

By default, the application has the SD card selected as the audio source. After the board powers up, the GUI should appear like the following figure.

As shown in the figure, the default audio source selected is SD card, with tracks displayed in the tracks list box. You can scroll through the tracks list by using the scroll bar, which allows you to select and play random tracks. The volume button allows you to increase/decrease the volume and the mute button to mute/unmute audio.



1. Plug in the headphones and you should be able to hear the audio track being played from the SD card.
2. Connect a micro-B USB cable between the port J4 on the PIC32MZ EF Starter Kit and the Host computer.
3. Change the audio source to USB, by selecting the ‘USB’ radio button on the GUI. The GUI should now appear as shown in the following figure. The track list will be blank, as the audio is now being streamed by the host computer.
4. Allow the Host computer to acknowledge, install drivers (if needed), and enumerate the device. No special software is necessary on the host side.
5. If needed, configure the Host computer to use the MPLAB Harmony USB speaker as the selected audio device. This may be done in the system configuration or Control Panel depending on the operating system.
6. Play audio on the Host computer. This may be done with a standard media player or through a variety of sources including operating system generated sounds or video.
7. Listen to the audio output on the speakers or headphones connected to the board. You can adjust the volume and mute/unmute either by the application running on the host, or from the on board GUI.
8. You can easily switch between the two sources of audio, SD card and USB, through the radio button selection on the GUI.



universal_audio_decoders

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

The Universal Audio Decoder application configures the development board to be in USB Host mode. The application supports the FAT32 file system. When a mass storage device is connected to the development board, the application begins to scan the root directory. Once the scan is complete, the first track in the list will be opened and played. The fill buffer routine will read a chunk of data as an input to the decoder of the supported file format. The decoder will decode the packet and send the first frame of audio data to the codec driver through I2S Driver and the output will be audible through the speakers. The following block diagram depicts a representation of the application.

Button controls provide support to traverse the directory tree and play audio files from other directories or sub-directories. By default, the application only supports WAVE (`.wav`) format files.

In addition to WAVE formats, the application also supports MP3, AAC, WMA, ADPCM, and Speex provided the decoder libraries and the supported source files are added as plug-ins. The MP3, AAC, and WMA are premium packages and must be purchased, whereas the PCM (i.e., WAVE format), ADPCM, Speex, Opus, and FLAC are included free of charge. Refer to the Microchip Premium MPLAB Harmony Audio web page (http://www.microchip.com/design-centers/audio-and-speech) for information.

Once purchased, the MP3, AAC, and WMA decoder modules can be added to the application as described in Selecting the Decoders Using MHC. Separate from purchasing the MP3, AAC and WMA libraries from Microchip, the use of MP3, AAC, and WMA in product design must be licensed from the third-party under their patents.

If support for any decoder is not available or was removed using MHC, the particular file format will not be scanned.

Audio Format	Package	Sampling Rates (kHz)	Description
ADPCM	Free of charge	8, 16	Adaptive Delta Pulse Code Modulation (ADPCM) is a sub-class of the Microsoft waveform (`.wav`) file format. In this demonstration, it only decodes ADPCM audio, which has a WAV header. The extension name for this format is pcm or PCM.
Speex	Free of charge	8, 16	Speex is an Open Source/Free Software patent-free audio compression format designed for speech. In this demonstration, only Speex bit-streams within an Ogg container can be decoded. The extension name for this format is spx or SPX. The library is built on Speex v1.0.5 source code from www.speex.org.

WAVE	Free of charge	8 through 96	The WAVE file format is the native file format used by Microsoft Windows for storing digital audio data.
AAC	Premium (must be purchased)	8, 11.025, 12, 16, 22.05, 24, 32, 44.1, 48, 64, 88.2, and 96	The AAC format is a lossy digital compression format of audio data with an ADTS header. The AAC decoder supports MPEG-2 and MPEG-4 AAC. To make sure the AAC audio files work with the AAC decoder, you can always convert any audio files to MPEG-2, 4 AAC files by a MPEG-2, 4 AAC encoder, one known working encoder is FAAC (Freeware Advanced Audio Coder). The current version of the AAC Library in MPLAB Harmony is v1.0.
MP3	Premium (must be purchased)	8 through 48	The MPEG1, MPEG2, and MPEG2.5 Layer 3 are lossy digital compression formats for audio data. The current version of the MP3 Library version in MPLAB Harmony is v3.0.
WMA	Premium (must be purchased)	8, 11.025, 16, 22.05, 32, 44.1, and 48	The Windows Media Application (WMA) format allows storing digital audio using lossy compression algorithm. The WMA decoder supports the ASF container format. The Windows Media Encoder 9 Series can be downloaded from the Microsoft website to convert any audio files to WMA v9.2 files to work with this WMA decoder. The current version of the WMA Library in MPLAB Harmony is v1.01.
FLAC	Free of charge	44.1, 88.2, and 96	The Free Lossless Audio Codec (FLAC) is an audio format similar to MP3, but lossless, meaning that audio is compressed without any loss in quality. The FLAC Library was built using source code v1.3.2 from http://downloads.xiph.org/releases/flac/.
OPUS	Free of charge	8 through 48	Opus is an open source audio codec, which is specifically designed for interactive speech and music transmission over the Internet. The Opus Library was built using source code v1.1.2 from http://opus-codec.org/.

Note: The AAC and MP3 Decoder Libraries have two versions: PIC32MX (for use with PIC32MX devices) and microAptiv (for use with PIC32MZ devices with the microAptiv core). When selecting either the AAC or MP3 library in MHC, for PIC32MX devices, the PIC32MX version of the library will be automatically added in the project. For PIC32MZ devices that have the microAptiv core, the microAptiv version of the library will be added in the project. Theoretically, MHC will automatically add the correct library; however, be sure to use the correct libraries on different devices.

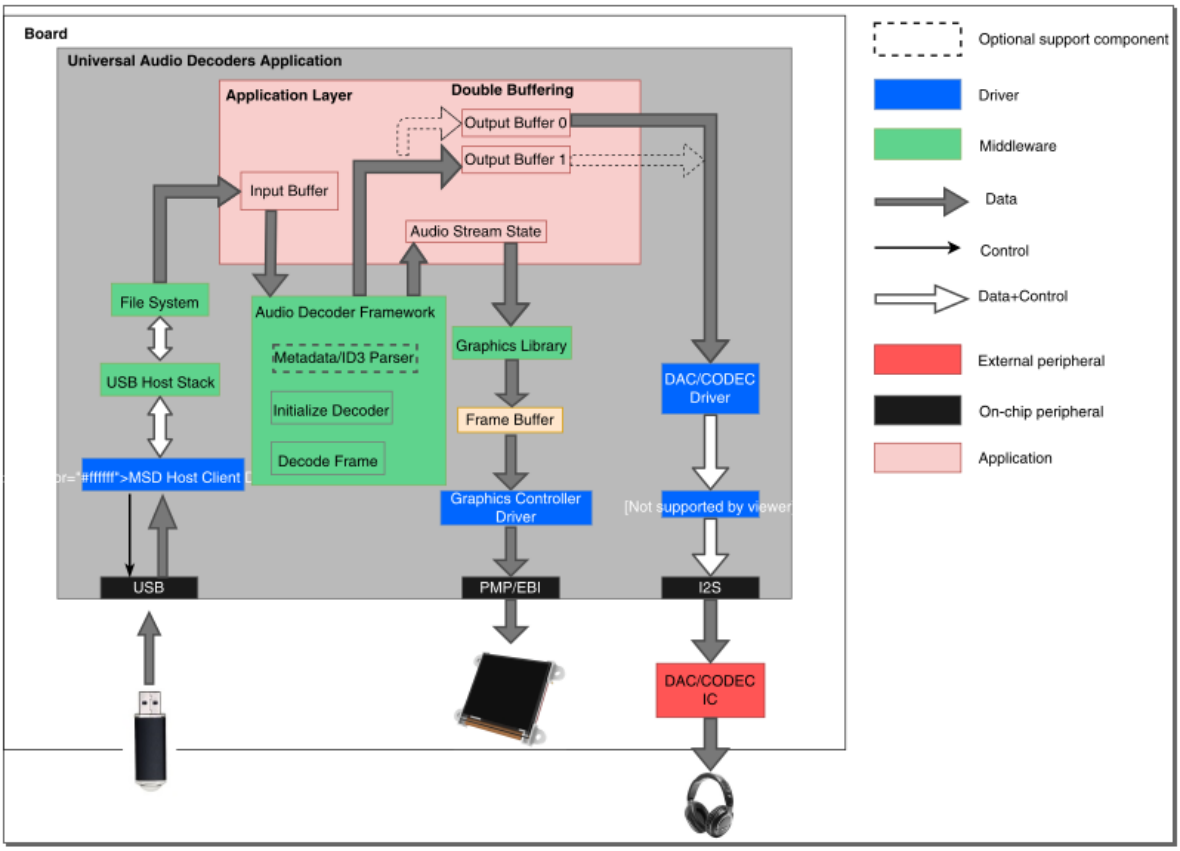
The macro DISK_MAX_DIRS and DISK_MAX_FILES in `system_config.h` under each configuration, determines the maximum number of directories that should be scanned at each level (to prevent stack overflow, the traversing level is limited to 5), and the maximum number of songs in total the demonstration should scan. For each configuration, the value could be different. For example, for the PIC32MX270F512L PIM, DISK_MAX_FILES can be approximately 600, while for the PIC32MZ EF Starter Kit, DISK_MAX_FILES may be as large as 800.

Refer to AAC Decoder Library, MP3 Decoder Library, and WMA Decoder Library in the Framework Help, as well as the Microchip Premium MPLAB Harmony Audio web page (www.microchip.com/pic32harmonypremiumaudio) for additional information on the Decoder Libraries.

Architecture

The universal_audio_decoders application uses the MPLAB Harmony Decoder Library to decode music files on a USB thumb drive. By default, it scans WAV (PCM) format files on mounted FAT32 USB thumb drive and streaming audio through a AKM DAC/Codec to a speaker; however, more decoders can be enabled by selecting decoders in MHC and regenerating the project. In the application, the number of audio output buffers can always set to be more than two to enhance the audio quality. And, the size of input buffer in this application is be chosen to be able to handle all decoders supported in MPLAB Harmony. Generally for non-Hi-Fi quality audio files, the size gives the application 23 ms to 26 ms buffering time on the AAC and MP3 decoders, 130 ms for Opus and Speex, 9 2ms for WMA, and 42 ms for FLAC. The following figure shows the architecture for the demonstration.

Architecture Block Diagram



Demonstration Features

- USB MSD Host Client Driver (see USB MSD Host Client Driver Library)
- FAT32 File System (see File System Service Library)
- Audio real-time buffer handling
- Decoding Multiple Audio decoders in real-time
- AKM Codec Drivers (see Codec Driver Libraries)
- I2S usage in audio system (see I2S Driver Library Help)

Tools Setup Differences

The application supports the WAVE, Compact MP3, AAC (ADTS header), WMA (v9.2), ADPCM (WAV header), and Speex (Ogg container) audio formats.

Note: The MP3, AAC, and WMA Decoder Libraries are premium packages and must be purchased. Refer to the Microchip Premium MPLAB Harmony Audio web page (www.microchip.com/pic32harmonypremiumaudio) for information.

The audio format can be chosen as required using MHC as follows:

1. Open the project on MPLAB X IDE and select the project configuration depending upon the hardware.
2. Open the MHC and select the Decoder.
3. Select **Use audio decoders libraries?** and the list of available decoders will be listed in the drop-down menu.
4. Select the decoders that are required and click **Generate** to add the supporting decoder files and libraries to the project.

Audio Codec Driver

For Configurations using the Audio DAC Daughter Board (AK4384). In the Timer driver instance field, the index number is changed to 1.

This is because Timer instance 0 is occupied by other drivers or services.

Timer Driver

The Timer driver configuration, Timer driver instance 0, is used by a system driver (which is also a MHC setting in System Service section), and Timer instance 1 is for the Audio DAC driver, timer instance 2 is set up for ticking function in application.

I2S Driver

Initial sampling rate is set to 48000; however, this demonstration is supported for different sampling rates during run-time, with an API available in the Audio Codec Driver.

SPI/I2S Module

SPI/I2S module pin setting is needed for to ensure that the I2S running correctly.

For the Bluetooth Audio Development Kit configuration, configure the following settings:

- SCK of I2S1 is set to port 70
- SDI1 is set to port 73
- SDO1 is set to 72
- SS1(out) is set to 69

Heap Size Setting

This application intends to set the heap size to the MCU at maximum SRAM capacity. This is done to support as many audio decoders as possible.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the demonstration.

Description

To build this project, you must open the `universal_audio_decoders.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/audio/universal_audio_decoders`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`universal_audio_decoders.X`	`<install-dir>/apps/audio/universal_audio_decoders/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
ak7755_bt_audio_dk	bt_audio_dk+ak7755	This configuration runs on the PIC32 Bluetooth Audio Development Kit with the AK7755 Codec Daughter Board.
bt_audio_dk	bt_audio_dk	This configuration runs on the PIC32 Bluetooth Audio Development Kit with the Audio DAC Daughter Board included with the kit.
270f512lpim_bt_audio_dk	pic32mx270f512l_pim+bt_audio_dk	This configuration runs on the PIC32MX270F512L PIM and the PIC32 Bluetooth Audio Development Kit with the Audio DAC Daughter Board included with the kit.
pic32mz_ef_sk_meb2	pic32_mz_ef_sk+meb2	This configuration runs on the PIC32MZ EF Starter Kit and the Multimedia Expansion Board II with a high-performance 4.3" WQVGA maXTouch Display Module
pic32mz_ef_sk_meb2_legacy	pic32_mz_ef_sk+meb2_legacy	This configuration runs on the PIC32MZ EF Starter Kit and the Multimedia Expansion Board II with a 4.3" WQVGA Display Board.

Configuring the Hardware

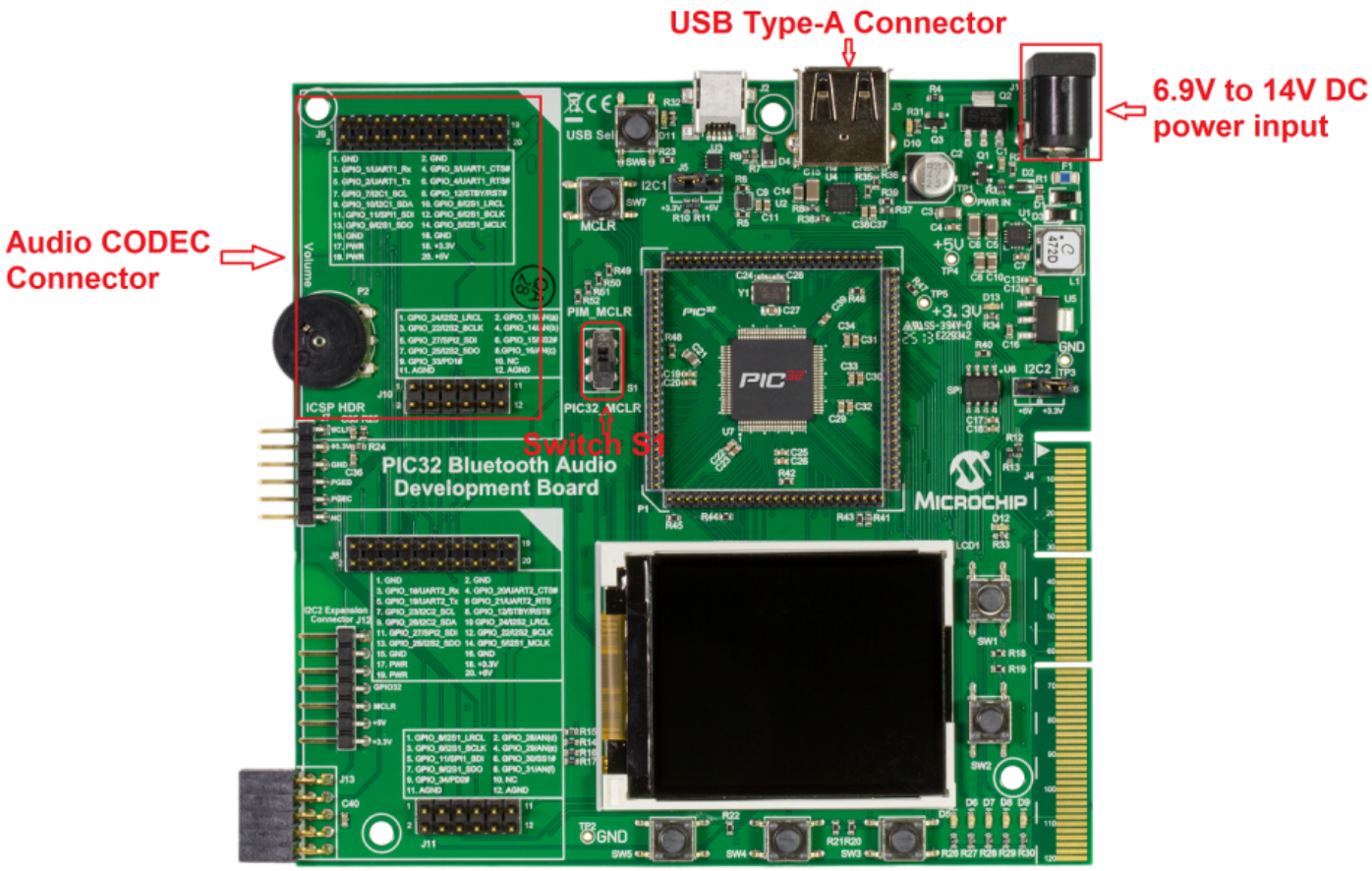
This section describes how to configure the supported hardware.

Description

For all configurations, a speaker or headset should be connected to the HP Out jack of the Audio Codec. A FAT32 file system USB Flash drive will be needed to inserted in the USB Type-A female connector on the board.

PIC32 Bluetooth Audio Development Kit with the PIC32 Audio DAC Daughter Board

Attach the Audio DAC (i.e., AK4383) Codec to the PIC32 Bluetooth Development Board J8 Connector. Switch S1 should be set to PIC32_MCLR.



PIC32 Bluetooth Audio Development Kit with the PIC32 Audio Codec Daughter Board - AK7755

Attach the AK7755 Codec to the PIC32 Bluetooth Development Board J8 Connector. Switch S1 should be set to PIC32_MCLR.

PIC32 Bluetooth Audio Development Kit with the PIC32MX270F512L Plug-in Module (PIM) and the PIC32 Audio DAC Daughter Board

Attach the Audio DAC (i.e., AK4384) Codec to the PIC32 Bluetooth Development Board J8 Connector. Attach the PIC32MX270F512L PIM to the PIM socket connector on the board. Switch S1 should be set to PIM_MCLR.

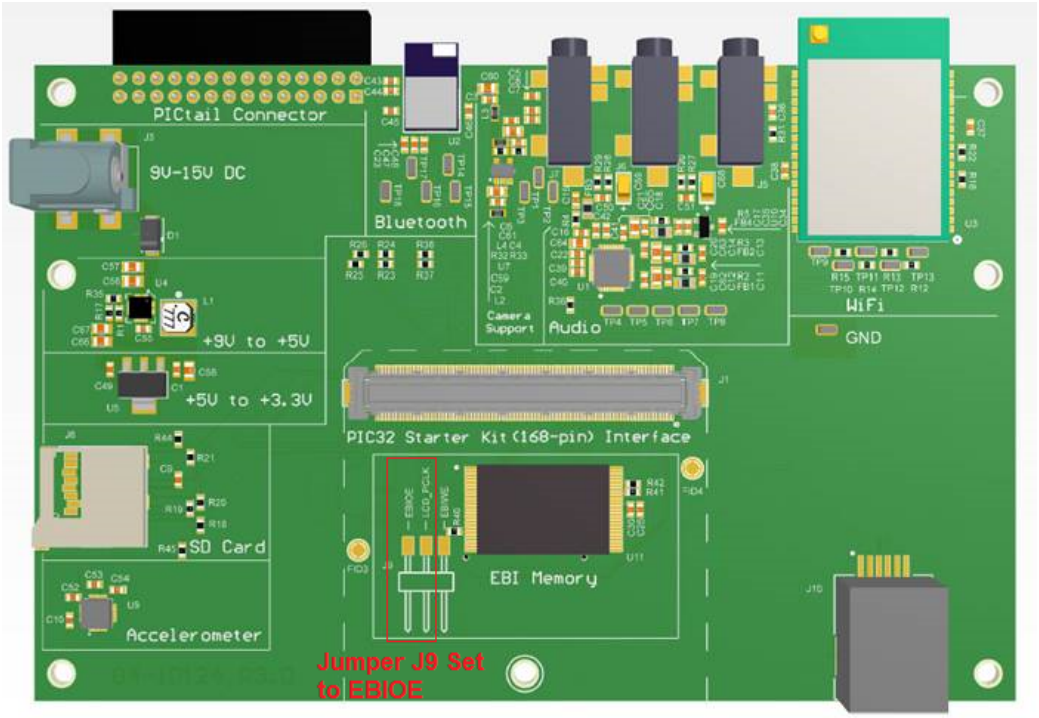
Multimedia Expansion Board II (MEB II) with 4.3" WQVGA Display Board and the PIC32MZ Embedded Connectivity with FPU (EF) Starter Kit

In this configuration, Jumper J9 must be set to EBIOE, otherwise the display will be blank white. Jumper J9 is located at back side of MEB II. See the following figure.

- Set the Jumper 9 to EBIOE
- Connect PIC32EF Starter Kit
- Plug in Power cord

Note that setting jumper 9 to EBIOE is meant to enable external SRAM, which is used by graphics double buffering in this configuration. In fact, external SRAM is not mandatory for graphics double buffering. In this application, enabling external SRAM is to save more internal SRAM space for memory-consuming audio decoders.

Note: This configuration is for use with the original MEB II, which is no longer available for purchase.



Multimedia Expansion Board II (MEB II) with the 4.3" WQVGA Display Module with maXTouch and the PIC32MZ Embedded Connectivity with FPU (EF) Starter Kit

In this configuration, Jumper J9 must be set to EBIOE, otherwise the display will be blank white. Jumper J9 is located at back side of MEB II. See the above figure.

- Set the Jumper 9 to EBIOE
- Connect PIC32EF Starter Kit
- Plug in Power cord.

Note that setting jumper 9 to EBIOE is to enable external SRAM, which is used by graphics double buffering in this configuration. In fact, external SRAM is not mandatory for graphics double buffering. In this application, enabling external SRAM is to save more internal SRAM space for memory-consuming audio decoders.

Note: This configuration is for use with the latest version of the MEB II.

Selecting the Decoders Using MHC

This topic describes how to select the decoders using the MPLAB Harmony Configurator (MHC).

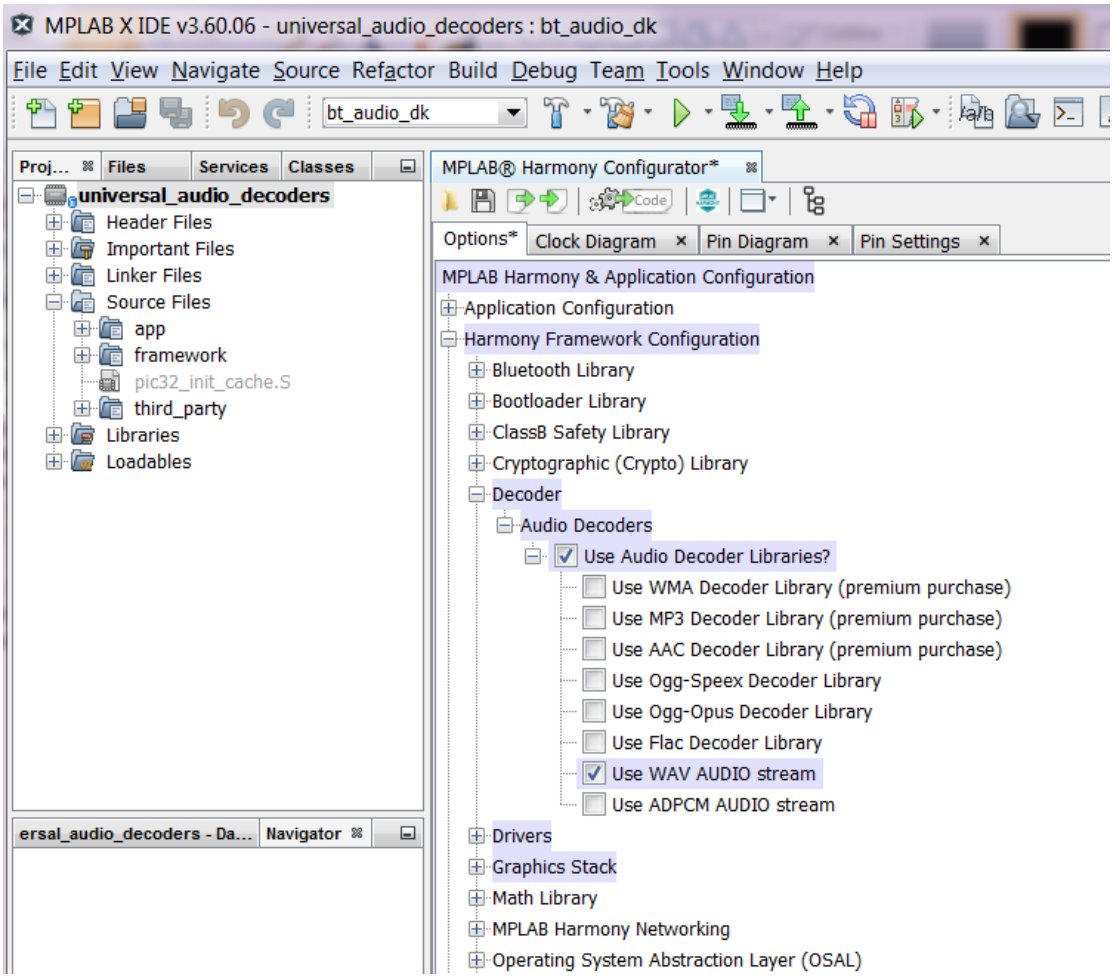
Description

The application supports the WAVE, Compact MP3, AAC (ADTS header), WMA (v9.2), ADPCM (WAV header), and Speex (Ogg container) audio formats.

Note: The MP3, AAC, and WMA Decoder Libraries are premium packages and must be purchased. Refer to the Microchip Premium MPLAB Harmony Audio web page (www.microchip.com/pic32harmonypremiumaudio) for information.

The audio format can be chosen as required using MHC as follows:

1. Open the project on MPLAB X IDE and select the project configuration depending upon the hardware.
2. Open the MHC and select the Decoder, as shown in the following figure.



3. Select **Use audio decoders libraries?** and the list of available decoders will be listed in the drop-down menu.
4. Select the decoders that are required and click **Generate** to add the supporting decoder files and libraries to the project.

Running the Demonstration

This section demonstrates how to run the demonstration.

Description

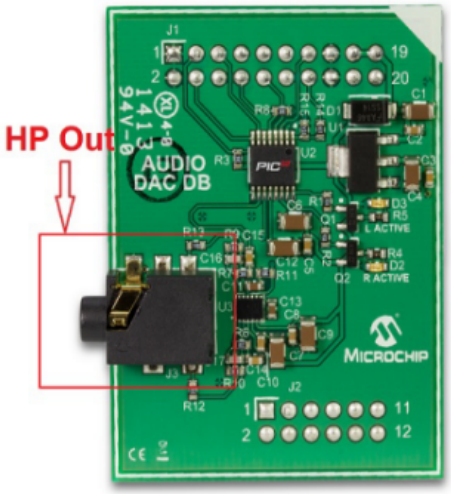
Note: Before running this demonstration, it is recommended to review the MPLAB Harmony Release Notes for any known issues. A PDF copy of the release notes is provided in the `<install-dir>/doc` folder of your installation.

Compile and program the target device. While compiling, select the appropriate MPLAB X IDE project configuration based on the demonstration board. These configurations set the target processor, the board and the Codec to be used in the output interface.

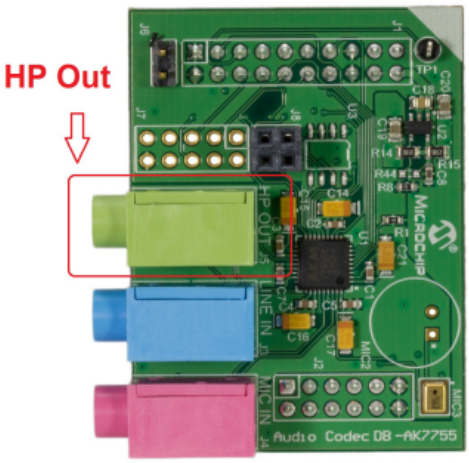
Refer to Building the Application for details.

Do the following to run the demonstration:

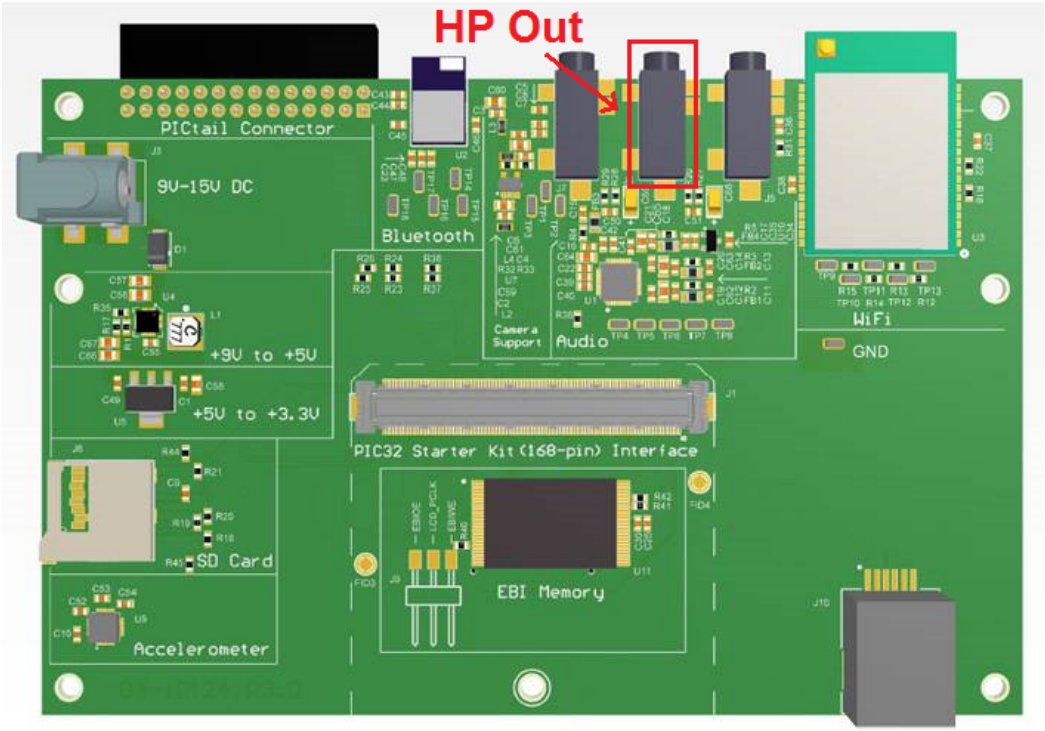
1. Connect speakers or headphones with a 3.5 mm jack to the output line connector on the Audio DAC Daughter Board (included with the PIC32 Bluetooth Audio Development Kit), or the headphone (HP Out) connector on the Audio Codec Daughter Board AK7755, or the Speaker Out/Line Out connectors on the Multimedia Expansion Board II (MEB II), as appropriate.



PIC32 Audio DAC Daughter Board
(Part # AC320032-2)

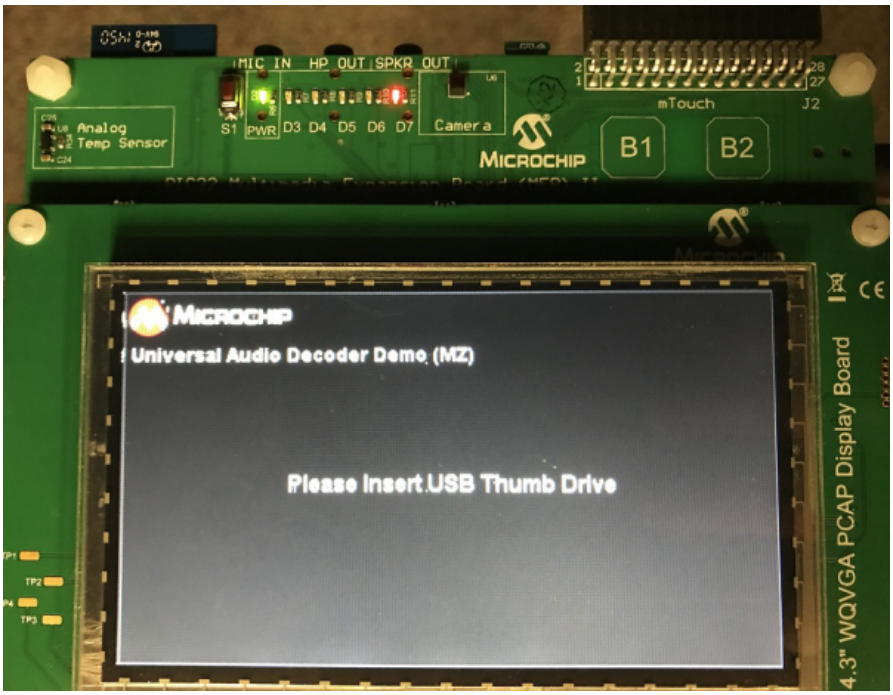


AK7755 Audio CODEC Daughter Board



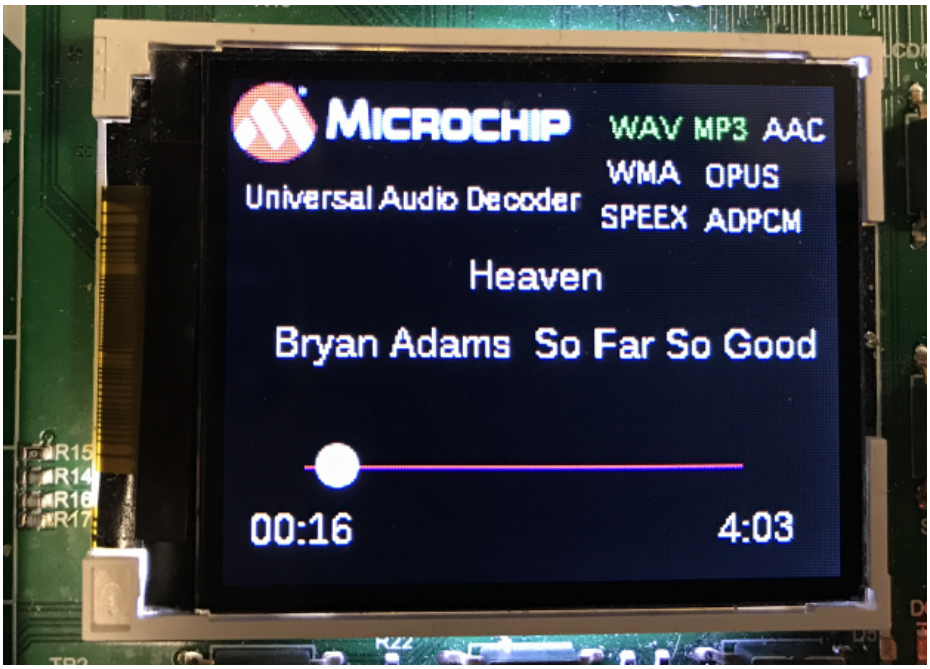
2. Connect power to the board. The system will be in a wait state for USB to be connected. The LEDs, D5, D6, and D7 (PIC32 Bluetooth Audio Development Board only), will be OFF during the wait state. Screen displays are for the PIC32MZ EF Starter Kit with MEB II board configurations.

Multimedia Expansion Board II (MEB II) Legacy Version Welcome Screen

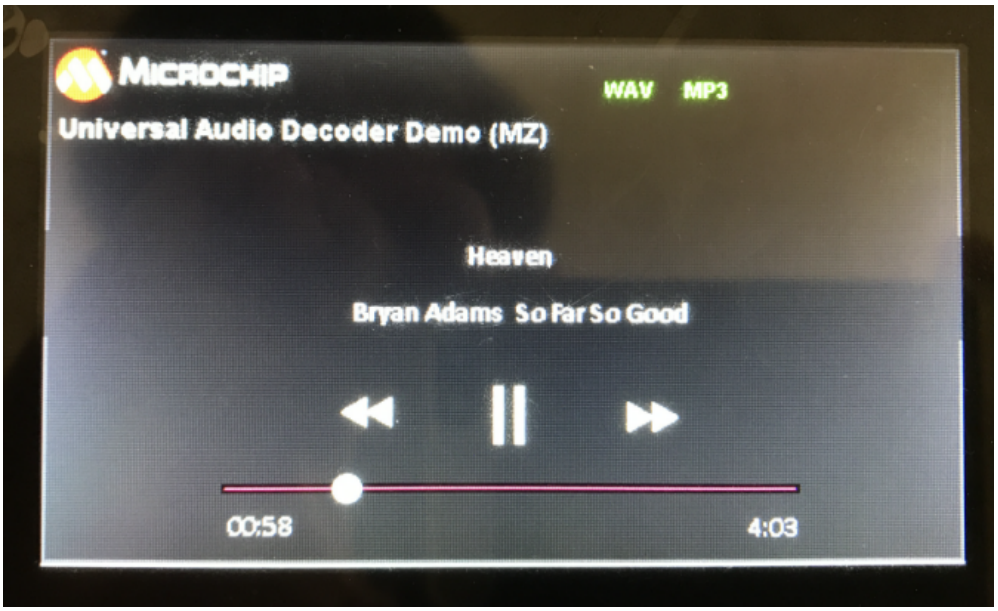


3. Connect a USB mass storage device that contains songs of supported audio format. The application by default can stream WAVE (`.wav`) format audio files.
4. When the USB device is connected the system will scan for audio files. The LEDs, D5 and D7 (PIC32 Bluetooth Audio Development Board only), will be ON during scanning.
5. Once the scanning is complete, listen to the audio output on the speakers or headset connected to the board. The LED, D5, will be ON for WAVE audio, the LED, D6, will be ON for AAC, and the LED, D7, will be ON for MP3 (PIC32 Bluetooth Audio Development Board only).

Bluetooth Audio Development Kit Playback Screen



Multimedia Expansion Board II (MEB II) Playback Screen



LED States (PIC32 Bluetooth Audio Development Board only)

State	D5	D6	D7
Wait for USB	OFF	OFF	OFF
Scan for files	ON	OFF	ON
WAVE audio stream	ON	OFF	OFF
AAC audio stream	OFF	ON	OFF
MP3 audio stream	OFF	OFF	ON
WMA audio stream	OFF	ON	ON

Demonstration Controls

Component	Label	PIC32 Bluetooth Audio Development Kit	PIC32MZ EF Starter Kit
Switch	SW1	N/A	Next Track (Toggle)/Fast Forward (Long Hold)
Switch	SW2	N/A	Play/Pause (Toggle)
Switch	SW3	Next Track (Toggle)/Fast Forward (Long Hold)	Previous Track (Toggle)/Fast Rewind (Long Hold)
Switch	SW4	Play/Pause (Toggle)	N/A
Switch	SW5	Previous Track (Toggle)/Fast Rewind (Long Hold)	N/A

universal_audio_encoders

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

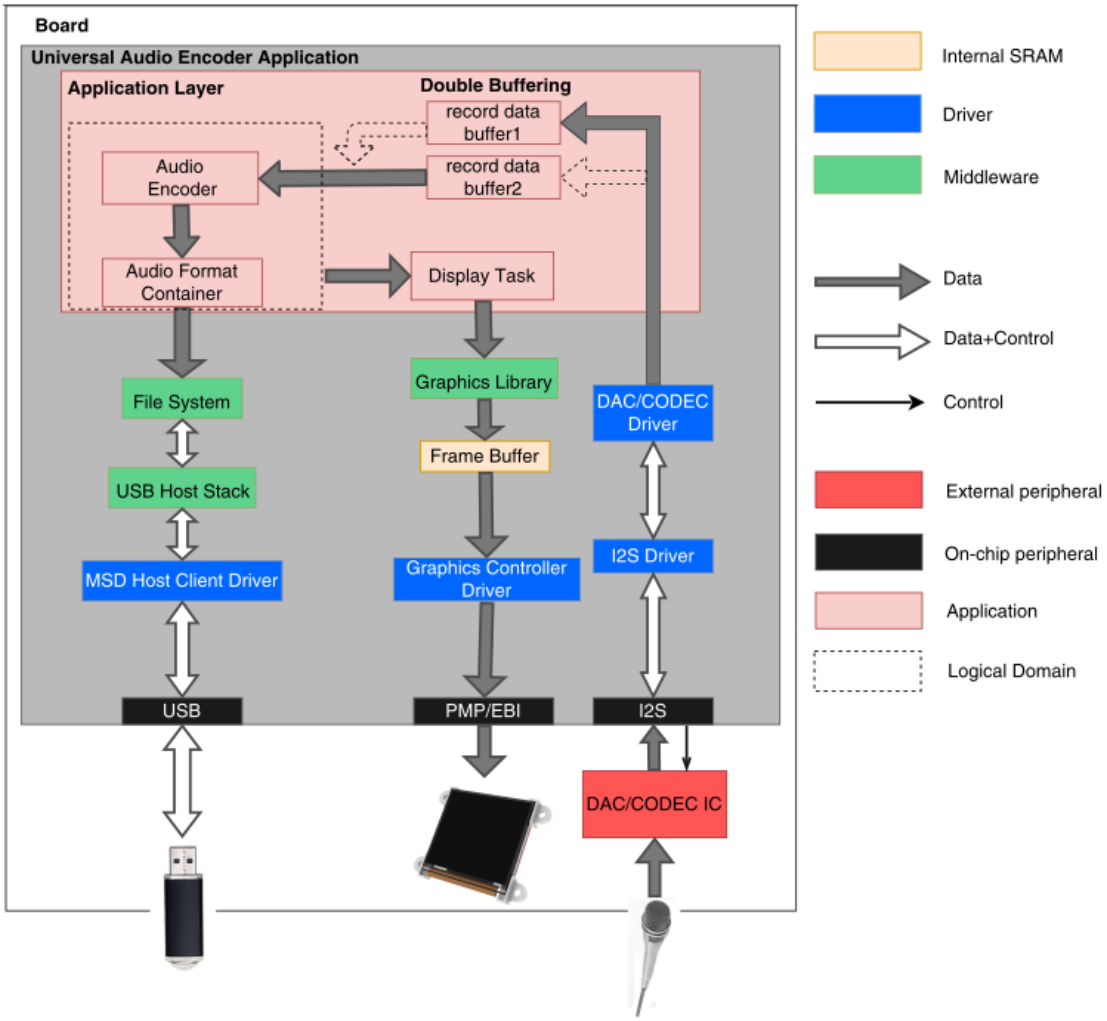
Description

This application demonstrates an audio encoder by taking a voice recording from a microphone, and storing encoded voice data to a mass storage device (USB Flash drive). This application first waits for a USB Flash drive to connect, and after connection is successful, voice data from a microphone is ready to be captured. During capture, data is encoded at the same time, the encoded data is written to a new create file on the root level of USB thumb drive file system, and when the voice capture ends, a `.wav` format will be saved on the USB Flash drive.

This application provides examples on encoding PCM and Opus data. In the configuration ak4642_bt_audio_dk, 16 kHz, Stereo 16-bit audio PCM data is encoded and encapsulated in WAV format, while in the configuration pic32mz_ef_sk_meb2, 16 kHz, Stereo 16-bit audio Opus data is encoded and encapsulated in Ogg format. The Opus Encoder is not supported on the ak4642bt_audio_dk configuration, since the Opus Codec Library requires 292 KB program memory.

Architecture

This universal audio encoders application demonstrates a framework to encode data to an audio file. the ping-pong buffer technique is used in the application to record and encode voice data simultaneously. This application also provides a graphics screen displaying prompting help messages at each step in the demonstration process.



Demonstration Features

- USB MSD Host Client Driver
- FAT32 File System
- AKM Codec Drivers
- I2S Driver usage in audio system
- I2C Driver for AKM Codec control
- Graphics Library usage
- Audio Encoders and Audio Containers System

Tools Setup Differences

The following descriptions explain the different MHC settings for this demonstration.

I2S Driver

The Sampling rate field is set to 16000 for voice recording. In DMA mode, only receive DMA support and Enable DMA channel Interrupts are selected. Under I2S Driver Instance, choose the Audio Communication Width to be SPI_AUDIO_COMMUNICATION_16DATA_16FIFO_32CHANNEL and choose Queue Size Receive to be 2.

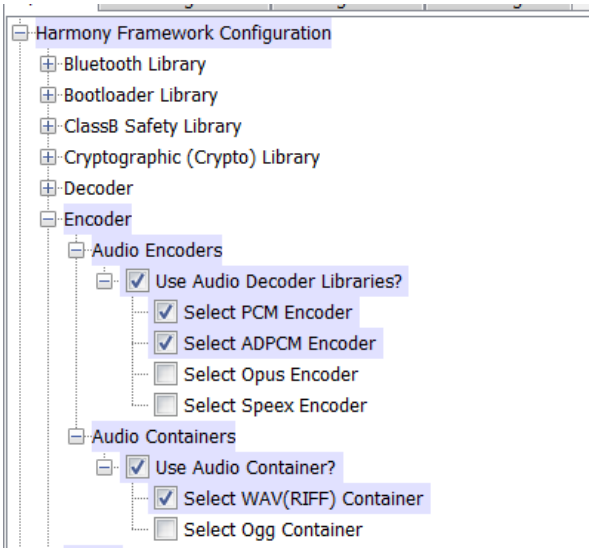
SPI/I2S Module Pin

The SPI/I2S module pin setting is needed to ensure that I2S is running correctly.

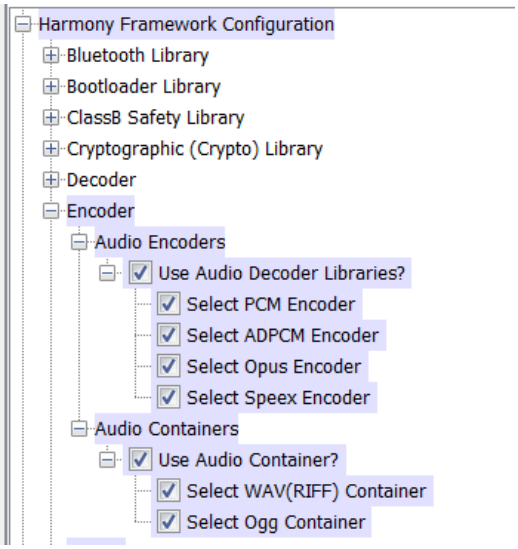
For example, on the PIC32 Bluetooth Audio Development Kit, SCK of I2S1 is set to port 70, SDI1 is to port 73, SDO1 is to 72, and SS1(out) is set to 69.

Audio Encoders and Containers

In the ak4642_bt_audio_dk, and ak4954_bt_audio_dk configurations, the PCM and ADPCM encoders are enabled, and the WAV container is selected in MHC, as shown in the following figure.



In the pic32mz_ef_sk_meb2 configuration, all encoders are selected in MHC, as shown in the following figure.



.

Note: PCM and ADPCM encoders can only be composited with WAV container, and Opus encoder can only be composited with Ogg containers. Any combinations other than these are not guaranteed to work properly.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the demonstration.

Description

To build this project, you must open the `universal_audio_encoders.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/audio/universal_audio_encoders`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`universal_audio_encoders.X`	`<install-dir>/apps/audio/universal_audio_encoders/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
ak4642_bt_audio_dk	bt_audio_dk+ak4642	This configuration runs on the PIC32 Bluetooth Audio Development Kit with the AK4642 Codec Daughter Board.
pic32mz_ef_sk_meb2	pic32mz_ef_sk+meb2	This configuration runs on the PIC32MZ EF Starter Kit and the Multimedia Expansion Board II (MEB II) with a high-performance 4.3" WQVGA maXTouch Display Module.
ak4954_bt_audio_dk	bt_audio_dk+ak4954	This configuration runs on the PIC32 Bluetooth Audio Development Kit with the AK4954 Codec Daughter Board.

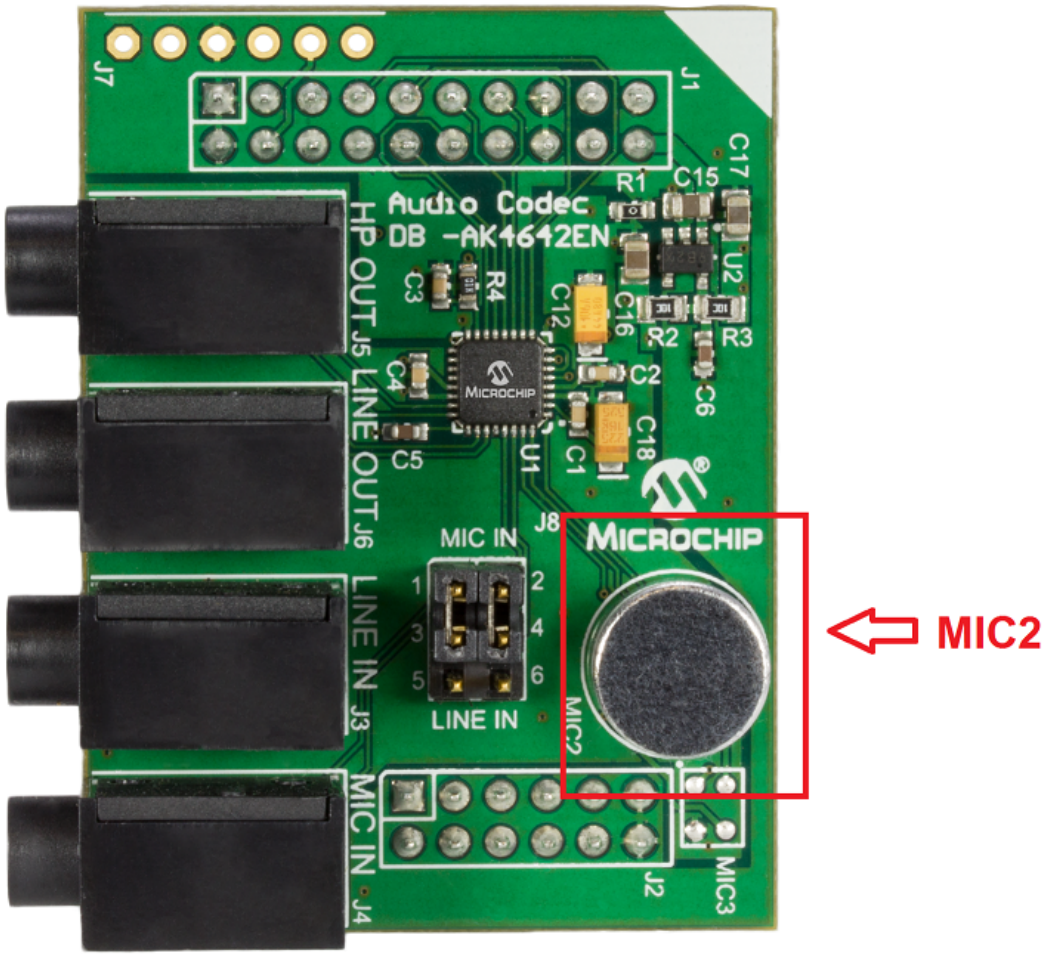
Configuring the Hardware

This section describes how to configure the supported hardware.

Description

PIC32 Bluetooth Audio Development Kit with the PIC32 Audio Codec Daughter Board – AK4642EN

The microphone (labeled MIC2) on the AK4642 daughter board will be used as the input microphone.

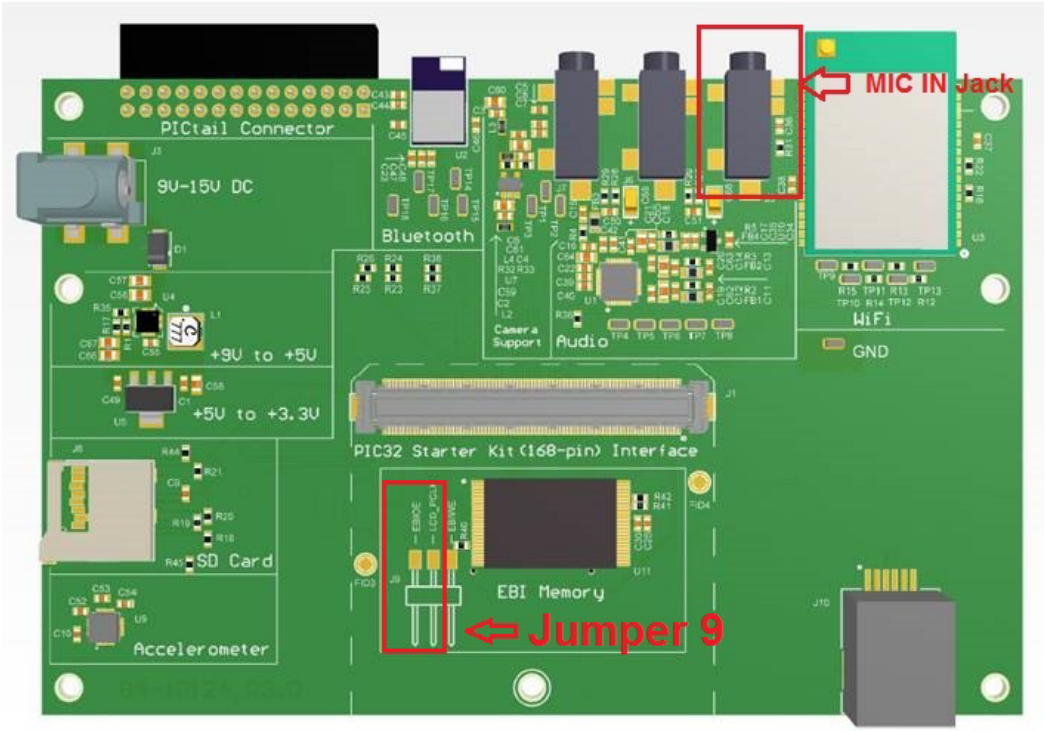


Multimedia Expansion Board II (MEB II) with the 4.3" WQVGA Display Module with maXTouch and the PIC32MZ Embedded Connectivity with FPU (EF) Starter Kit

In this configuration, Jumper J9 must be set to EBIOE, otherwise the display will be blank white. Jumper J9 is located at back side of MEB II.

- Set the Jumper 9 to EBIOE
- Connect PIC32EF Starter Kit
- Connect a microphone device to MIC IN Jack of the AK4953 Codec on MEB II board
- Plug in Power cord.

Note that setting jumper 9 to EBIOE is to enable external SRAM, which is used by graphics double buffering in this configuration. In fact, external SRAM is not mandatory for graphics double buffering. In this application, enabling external SRAM is to save more internal SRAM space.



Running the Demonstration

This section demonstrates how to run the demonstration.

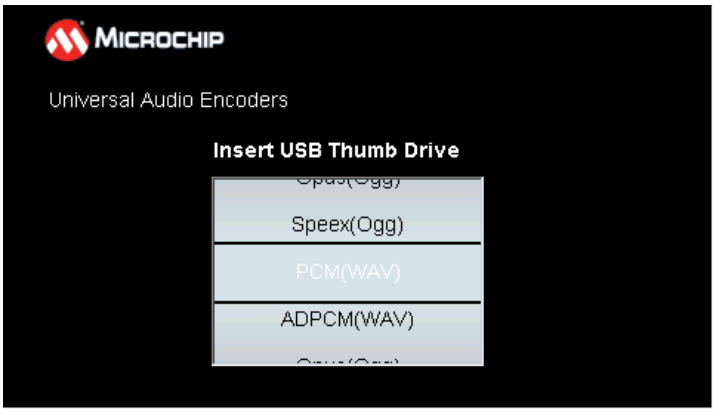
Description

Compile and program the target device. While compiling, select the appropriate MPLAB X IDE project configuration based on the demonstration board. These configurations set the target processor, the board and the Codec to be used in the output interface.

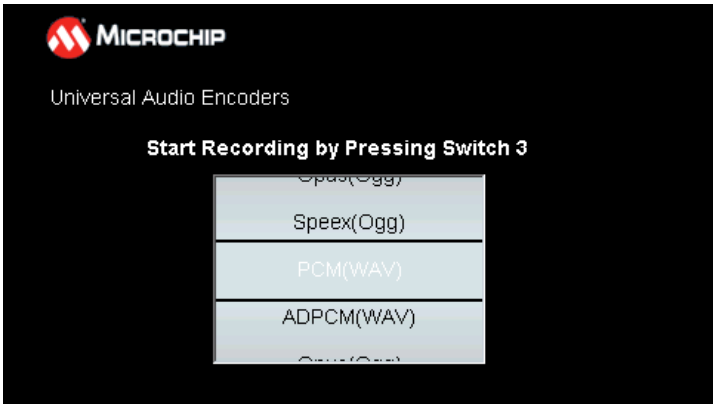
Refer to Building the Application for details.

The following steps show the demonstration running using the pic32mz_ef_sk_meb2 configuration.

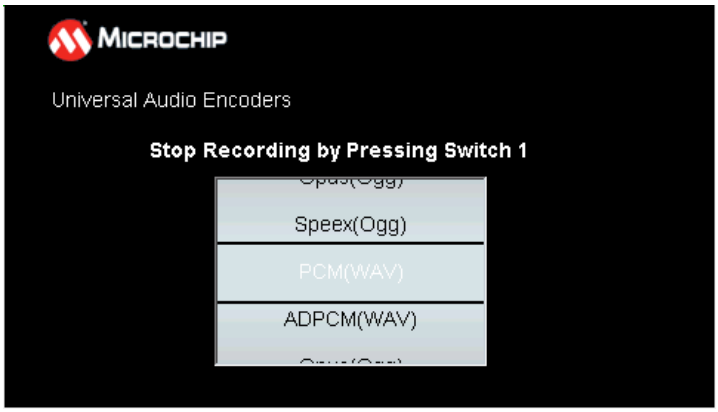
1. For the pic32mz_ef_sk_meb2 configuration, connect a microphone with a 3.5 mm jack to the MIC-IN line connector on the MEB II.
2. Connect power to the board. The system will be in a wait state for the USB Flash drive to connect. Select desired encoder in list wheel, use finger to scroll list down or up.



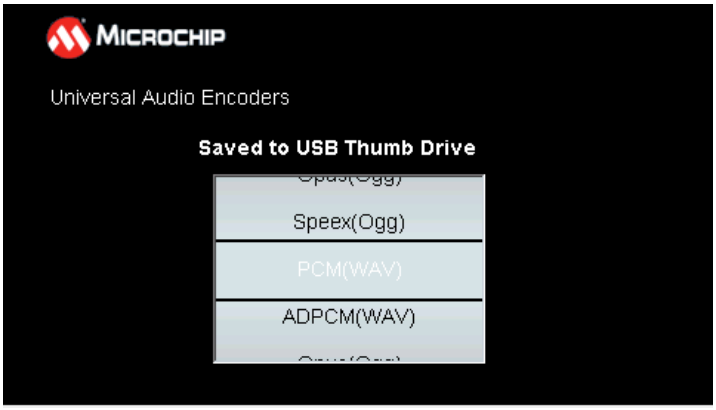
3. Connect a USB mass storage device that is formatted as FAT32 file system. After the USB Flash drive is connected and the file system has mounted successfully, the display prompts you to begin recording.



4. Press switch 5 to start recording for the ak4642_bt_audio_dk configuration. Press switch 3 to start recording for the pic32mz_ef_sk_meb2 configuration.
5. Once recording has started, the microphone will immediately begin to capture audio data, and at the same time, the application begins encoding and saving data to a created file on the USB Flash drive. Do not remove the USB Flash drive while recording.



6. Press switch 3 to stop recording for the ak4642_bt_audio_dk configuration. Press switch 1 to stop recording for the pic32mz_ef_sk_meb2 configuration.
7. After recording has stopped, the application will encapsulate the encoded data in a corresponding format depends on which encoder is selected before encoding. The name for this file will be called encoder.{format name} on the USB Flash drive.



usb_headset

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This demonstration application configures the a USB headset system configurable to 48/32/16 kHz sampling rate at 16 bit per sample.

The system will interface to a USB Host (such as a personal computer), which can accommodate a USB device class headset. The embedded system will enumerate with a USB audio device endpoint and enable the host system to input audio from the USB port using a standard USB Full-Speed implementation. The embedded system will take the data from a microphone the Codec and send it to the audio USB interface, while

simultaneously streaming playback audio to the Codec. The Codec Driver sets up the audio interface, timing, DMA channels and buffer queue to enable a continuous audio stream. The digital audio is transmitted/received to/from the Codec through a bidirectional I2S data module.

The application runs on the PIC32 Bluetooth Audio Development Kit with the AK4642 Codec interfaced to the PIC33MX470F512L processor. microcontroller. It also runs on the PIC32MZ EF Starter Kit on the Multimedia Entertainment Board II (MEB II) with the AK4953 Codec interfaced to the PIC32MZ2048EFH144 processor.

Architecture

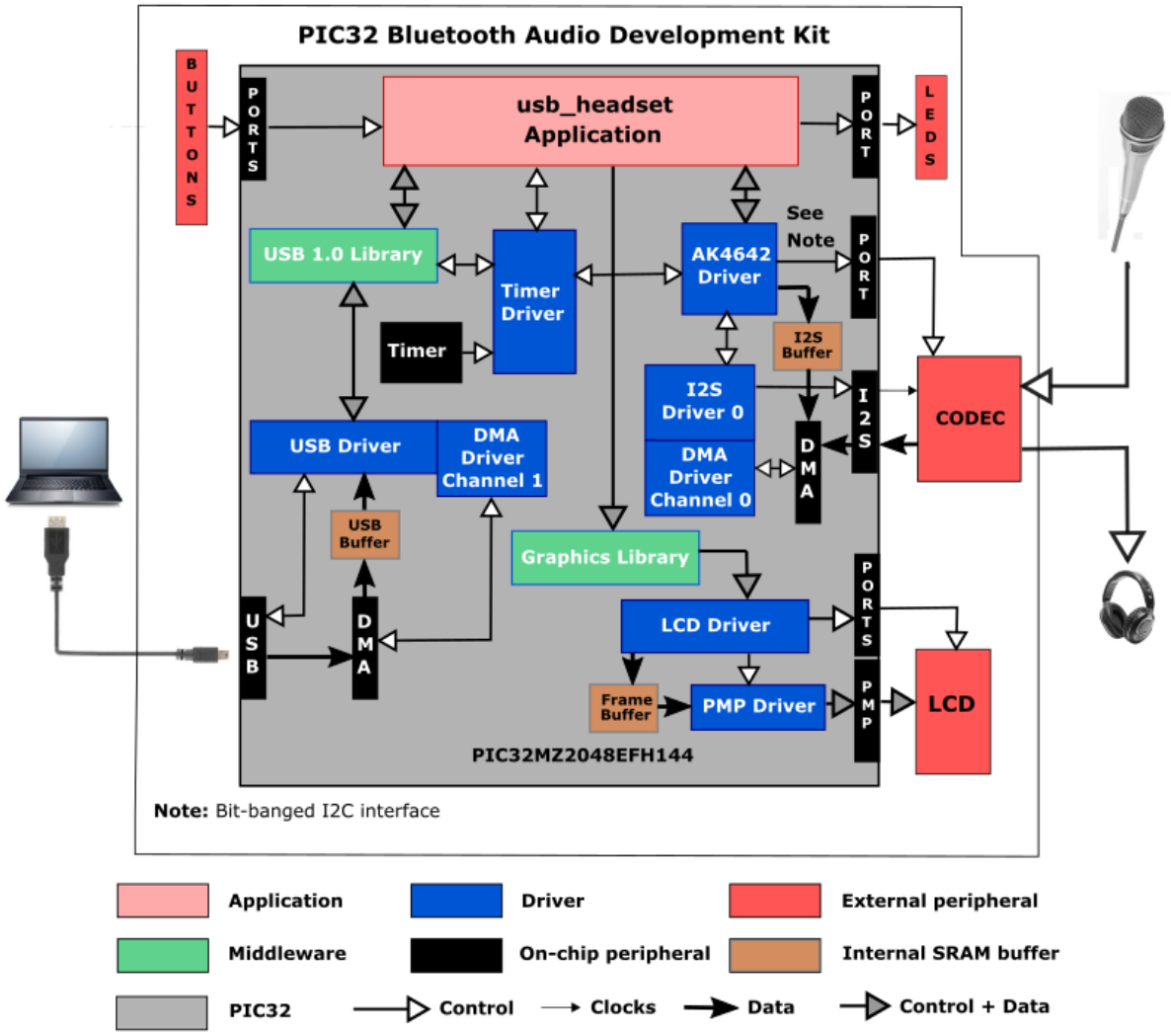
This application runs on the PIC32 Bluetooth Audio Development Kit, which contains a PIC32MX470F512L microcontroller with 512 KB of Flash memory and 128 KB of RAM running at 96 MHz, using the following hardware:

- 220x176 color LCD
- Single push button
- USB Device and Host interfaces
- Five LEDs
- Two X32 sockets, one of which will be used to host a AK4642 Codec module daughter board having both an on-board internal microphone that will be used for the demonstration, and an external microphone input through a 3.5 mm audio jack

The PIC32 Bluetooth Audio Development Kit comes with an AK4384 Audio DAC daughter board; however, you must replace this board with the AK4642 daughter board.

This application also runs on the PIC32MZ EF Starter Kit on the Multimedia Expansion Board II (MEB II). The SK contains a PIC32MZ2048EFH144 microcontroller with 2M bytes of flash memory and 144K bytes of RAM running at 198 MHz. The SK interfaces to the MEB2, which supports a 480x272 pixel color LCD touch screen; USB device and USB host interfaces; and an AK4953 Codec that connects to an external microphone input through a 3.5 mm audio jack.

The following figure shows the application structure and hardware used. The PIC32 runs all of the application code. The buttons are interfaced using GPIO pins.



The interface between the PIC32MX480F512L and the LCD display on the PIC32 Bluetooth Development Kit is an 8-bit parallel master port (PMP) used for data transfer of graphics commands implemented by a graphics processor. The program uses the MPLAB Harmony v2.0 Graphics Library to draw on the screen.

The PIC32MZ2048EFH144 writes pixels directly to the graphics frame buffer memory, not using a off-chip graphics process, thus the PMP is not needed for the PIC32MZ EF Starter Kit/MEB II configuration.

The Codec utilizes 1 instance of I2S , with the I2S module and Codec configured, as shown in the following tables. The I2S is configured for bidirectional data flow, using the SDI1 pin for input from the microphone and SDO1 for output to the headphone jack. The I2S Parameters value table shows the parameters being used to generate these clocks and configure the Codec device (either AK4642 or AK4953) over I2C. Initially, the Codec is set for 16 kHz; however the USB can change this to also be 32 or 48 kHz. This is performed automatically by changing the value of the MCLK1/REFCLK1, BCLK Multiplier value and/or the MCLK Multiplier value using a call to the DRV_CODEC_SampleRateSet function.

The I2S is set up for 16 bit data per stereo channel data transferred serially using 32 bit framing per channel (the 2 least significant bytes are 0 and ignored by the module). The data is configured as stereo, despite the microphone data being mono with the data duplicated across both 16 kHz channels (left and right). The 16bit stereo samples are transferred to the memory buffer via DMA Channel 3 (PIC32MZ EF Starter Kit/AK4953) or DMA Channel 2 (PIC32 Bluetooth Development Kit/AK4642).

The I2S clock values generated by the master are derived from the following formulas, starting with the REFCLK0 value (4.096 Khz) used for the I2S MCLK and the sample rate, Fs (16000, 32000, and 48000 Hz):

MCLK = 4096000 Hz

NUM_CHANNELS = 2

FRAME_SIZE = 32 bits/channel*NUM_CHANNELS = 64

BCLK = FRAME_SIZE * Fs

LRCK = Fs

The codec slave uses the following parameters calculated from the clock values:

MCLK_DIV = MCLK/LRCK

BCLK_DIV = MCLK/BCLK

The folowing table shows the values used for Fs=16000

16 kHz Sampling I2S Clock Values

I2S Function	Value	Description/PIC32 Pin (Port)
LRCK	16.000000 kHz	Sample rate clock /SS1 on pin 69 (RD1)
BCLK	1024000 Hz	Bit rate clock /SCK1 on pin 70 (RD10)
MCLK	4.096000 MHz	Master clock /REFCLK0 on pin 53 (RF8)

32 kHz Sampling I2S Clock Values

I2S Function	Value	Description/PIC32 Pin (Port)
LRCK	32000000 kHz	Sample rate clock /SS1
BCLK	2048000 Hz	Bit rate clock /SCK1
MCLK	N/A	Master clock /SCK1

48 kHz Sampling I2S Clock Values

I2S Function	Value	Description/PIC32 Pin (Port)
LRCK	48.000000 kHz	Sample rate clock /SS1
BLCK	3072000 Hz	Bit rate clock /SCK1
MCLK	N/A	Master clock /SCK1

PIC32MX470F512L I2S Data Pins

PIC32 Name	Description	I2S	PIC32 Bluetooth Audio Development Kit Pin	PIC32MZ EF Starter Kit/MEB II Pin
SDI1	Serial Data In	SDI1	73 (RC13)	69 (RD14)
SDO1	Serial Data Out	SDI1	72 (RD0)	49 (RB10)
SS1	Bit Clock	BCLK	69 (RD0)	58 (RF12)
SCK1	Master Clock	MCLK	70 (RD10)	109 (RD1)
REFCLKO1	Reference Clock	N/A	53 (RF8)	70 (RD15)

I2S Parameter Values

I2S	Value	Description/PIC32 Pin (Port)
PIC32 Mode	HOST	

AK4642 Mode	SLAVE	Controlled by the PIC32
MCLK_MULT (mhc)	256	16 kHz sample rate multiplier for MCLK
MCLK_BCLK_RATIO	4	16 kHz sample rate multiplier for BCLK
Data Sizes	16/16/32	16-bit data/16-deep queue/32-bit frame
Channels	Stereo	2-channel
Timing	LJ	Left-justified

The I2S interface used 4 device pins as shown in the table. The REFCLOCK01 is available but unused by the Codec interface.

The application uses USB Library as a "Device" stack, which will connect to a "Host". It is configured for 5 USB endpoints with a single function having three interfaces. All of these are defined for a USB Microphone device in the fullSpeedConfigurationDescriptor array variable structure (located in `system_init.c`). This structure defines the connection to the host, variable sample rates of 48, 32, or 16 kHz; with a 16-bit stereo channel data. A 64 packet queue is used for playback data, and a ping-pong buffer scheme is used for microphone record data. The maximum USB packets size is set to 482 channels/sample 2 bytes/channel= 192 bytes, which gives a 1 ms stereo sample packet size at 48 kHz; therefore, the buffer size needs to be of the same size.

All Harmony applications use the function SYS_Initialize located in the source file system_init.c and executed from main to initialize various subsystems such as the clock, ports, BSP (board support package), PMP, Codec, timers, UART, graphics, and interrupts. The USB, AK4642 driver, graphics, and the application state machines are all updated via calls located in the function SYS_Tasks, executed from the main polling loop, located in `system_tasks.c`.

The application code is contained in the standard source file app.c. The application code is initialized in SYS_Initialize using the application APP_Initialize function to instantiate driver objects and start the USB Device interface. The application utilizes a simple state machine (APP_Tasks executed from SYS_Tasks) with the following functions:

- Sets up the drivers and USB Library interface
- Responds to USB Host commands ("USB Record at 48Khz/32Khz/16Khz", "USB Playback at 48Khz/32Khz/16Khz")
- Initiate and maintains the bidirectional data audio streaming for "USB Record" and "USB Playback" functions.

Demonstration Features

- Record and Playback using an AK4642 codec daughter board using multiple sample rates
- USB connection to a host system using the USB Library Device Stack for a USB Headset device for multiple selectable sample rates (16/32/48 kHz)
- Displaying graphics on the PIC32 Bluetooth Audio Development Kit 220x176 LCD using the Graphics Library and the Parallel Master Port (PMP) Driver Library
- Displaying graphics on the PIC32 MZ EF Starter Kit/MEB II 480X272 pixel touchscreen using MPLAB Harmony Graphics Library Library and the Parallel Master Port (PMP) Driver Library

Tools Setup Differences

bt_audio_dk_ak4642 Configuration

If building this application from scratch, start by creating a 32-bit MPLAB Harmony project in MPLAB X IDE by selecting *File > New Project*. Select the PIC32MX Bluetooth Audio Development Kit and the configuration bt_audio_dk_AK4642. Click the green MPLAB Harmony icon to open the MPLAB Harmony Configurator (MHC).

Expand *Harmony Configurator Framework > Drivers*, and then expand *Codec > AK4642*. Select **Use AK4642 Driver?** and a sub-menu will appear. For this application the default values will suffice, although the volume setting can be adjusted between a -75 dB and +12 dB gain range. The I2S and I2C driver (default to DRV_I2S_INSTANCE_0 and DRV_I2C_INSTANCE_0, respectively) associated to the Codec driver can be configured as described previously.

To use the Graphics Stack Library, under Graphics Stack, select **Use Graphics Stack?**. Further down, under *Use Harmony Graphics Composer Suite? > Middleware > Use Aria User Interface Library?*, clear **Enable Touch**. Since the board uses the Crystal Fontz 176x220 pixel LCD with the OTM2201A controller the PIC32 Bluetooth Audio Development Kit does not support a touchscreen. Under PMP, select **Use PMP Driver?**, which is used for graphics data transfer. Expand that section, and change the value of Strobe Wait Sates to PMP_STROBE_WAIT_4.

To use the USB Stack Library, under USB Library, select **Use USB Stack Under Select Host** or for the Device Stack, select **Device**. Change the Number of Endpoints Used to 5. Below that, select **USB Device Instance 0** and under that make Function 1 (matching the fullSpeedConfigurationDescriptor struct in system_init.c: AUDIO Device Class, 3 Interfaces, 2 Audio Streaming interfaces; with a Write Queue Size of 2 (for the ping-pong buffer arrangement) and a Read Queue Size of 2 (not used). The Product ID Selection gives the name shown at the Host, which in this case is "usb_microphone_demo". A USB Interrupt Priority of 1 is also used.

Note: Do not change the current fullSpeedConfigurationDescriptor values in `system_init.c` when using MHC to generate the project files.

pic32mz_ef_sk_meb2 Configuration

If building this application from scratch, start by creating a 32-bit MPLAB Harmony project in MPLAB X IDE by selecting *File > New Project*. Select the PIC32MZ(EF) Starter Kit plus MEB II and the configuration pic32mz_ef_sk_meb2. Click the green MPLAB Harmony icon to open the MPLAB Harmony Configurator (MHC).

Expand the *Harmony Configurator Framework* > *Drivers* section, and then expand *Codec* > *AK4953*. Select **Use AK4953 Driver?** and a sub-menu will appear. For this application the default values will suffice, except two clients are used (read and write), and volume setting can be adjusted between a -75 dB and +12 dB gain range. The I2S and I2C driver (default to DRV_I2S_INSTANCE_0 and DRV_I2C_INSTANCE_0, respectively) associated to the Codec driver. I2S can be configured as described previously, using the 16 kHz values, although the application can modify these during runtime.

The bit-banged implementation and "Include Force Write ...)" is required for the PIC32MZ I2C module (I2C_ID_2). TMR1 is used for the bit-banged I2C signal timing over GPIO pins.

To use the Graphics Stack Library, under *Graphics Stack*, select **Use Graphics Stack?**. The default values given by the BSP can be used, which select the LCC graphics interface to a WQVGA screen and Touch Screen Controller with event processing. The only change is to use a V-Sync Refresh Strategy of "Conventional" rather than "Aggressive" to allow more processing time away from graphics processing and data bus transfer.

To use the USB Stack Library, under *USB Library*, select **Use USB Stack Under Select Host** or **Device Stack Select Device**. Change the Number of Endpoints Used to 2. Below that select **USB Device Instance 0** and under that select a Device Speed of USB_SPEED_FULL Select Function 1 and the AUDIO Device Class 3 Interfaces, 2 Audio Streaming interfaces; with a Write Queue Size of 8 (but using a ping-pong buffer arrangement with the microphone buffers) and a Read Queue Size of 64 (to minimize "underflow/overflow" due to mismatched clocks between USB and the CODEC. The Product ID Selection gives the name shown at the Host, which in this case is "usb_headset_demo".

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the demonstration.

Description

To build this project, you must open the `usb_headset.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/audio/usb_headset`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`usb_headset.X`	`<install-dir>/apps/audio/usb_headset/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
bt_audio_dk_ak4642	bt_audio_dk	This configuration runs on the PIC32 Bluetooth Audio Development Kit with the Audio Codec Daughter Board AK4642EN.
pic32mz_ef_sk_meb2	pic32mz_ef_sk+meb2	This configuration runs on the PIC32MZ EF Starter Kit with the MEB II.

Configuring the Hardware

This section describes how to configure the supported hardware.

Description

PIC32 Bluetooth Audio Development Kit and Audio Codec Daughter Board AK4642EN

Switch S1 on the PIC32 Bluetooth Audio Development Board should be set to PIC32_MCLR.

PIC32MZ EF Starter Kit and MEB II

In this configuration, Jumper J9 must be set to EBIOE, otherwise the display will be blank white. Jumper J9 is located at back side of MEB II.

- Set the Jumper 9 to EBIOE
- Connect PIC32EF Starter Kit
- Connect a microphone device to MIC IN Jack of the AK4953 Codec on MEB II board
- Plug in Power cord.

Note that setting jumper 9 to EBIOE is to enable external SRAM, which is used by graphics double buffering in this configuration. In fact, external SRAM is not mandatory for graphics double buffering. In this application, enabling external SRAM is to save more internal SRAM space.

Running the Demonstration

This section demonstrates how to run the demonstration.

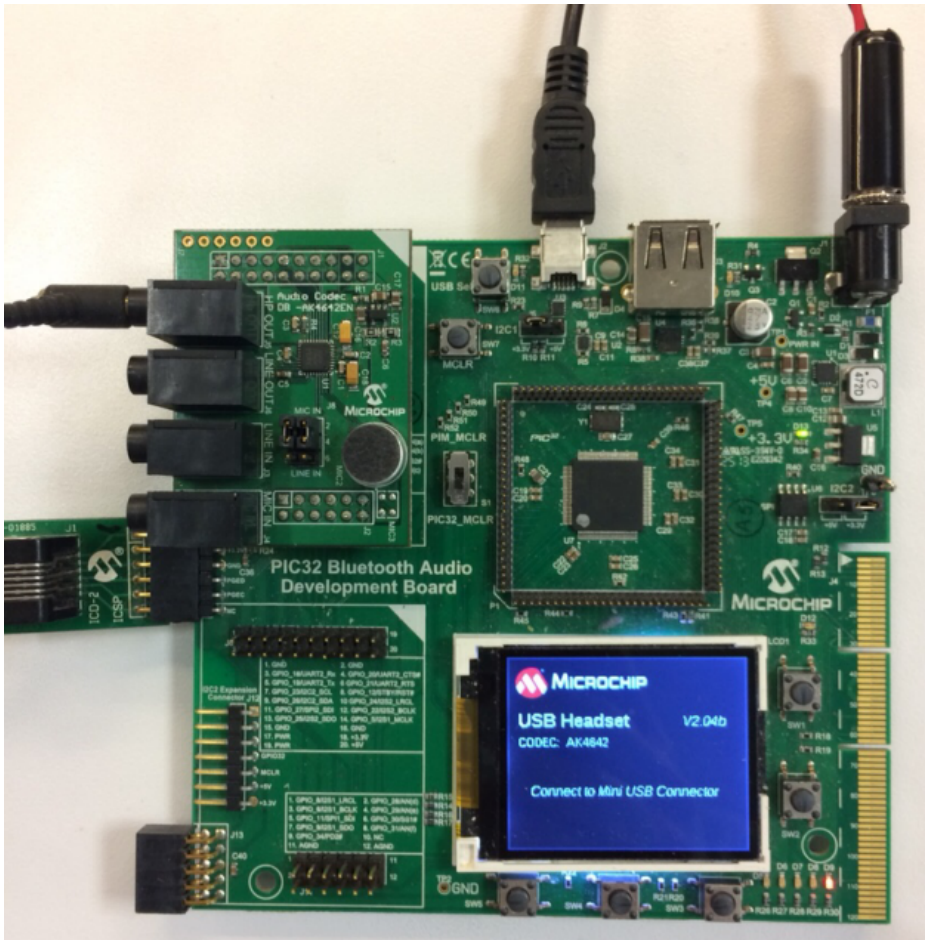
Description

Important! Prior to using this demonstration, it is recommended to review the MPLAB Harmony Release Notes for any known issues. A PDF copy of the release notes is provided in the `<install-dir>/doc` folder of your installation.

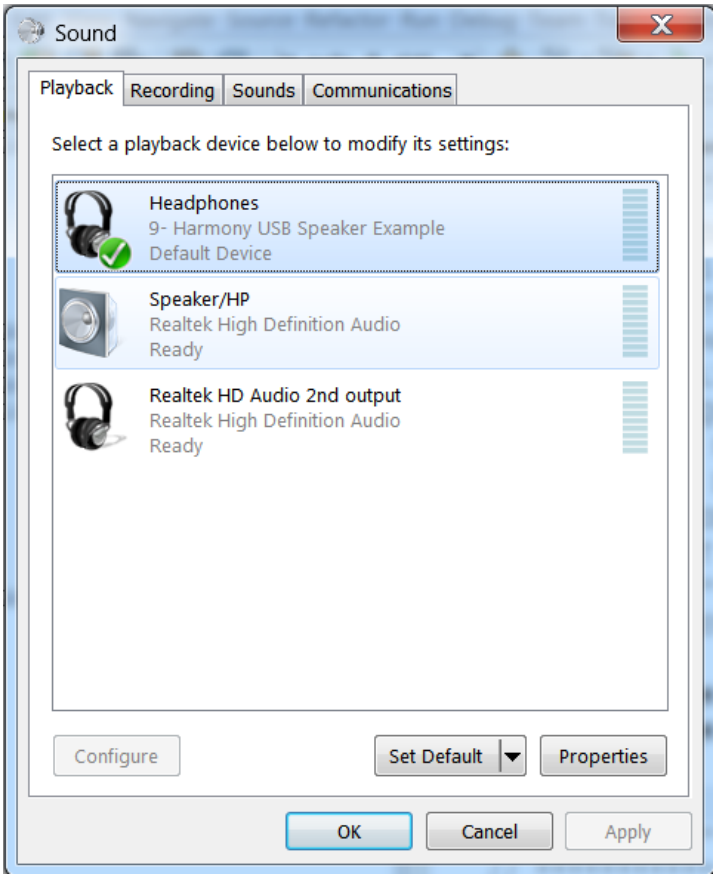
Compile and program the target device. While compiling, select the appropriate MPLAB X IDE project configuration based on the demonstration board. These configurations set the target processor, the board and the Codec to be used in the output interface. Refer to Building the Application for details.

Do the following to run the demonstration:

1. Power up the board and connect the programmer. Compile the application, program the target device, and run it. After the program starts, the name of the application should appear on the display, as shown in the following figure.

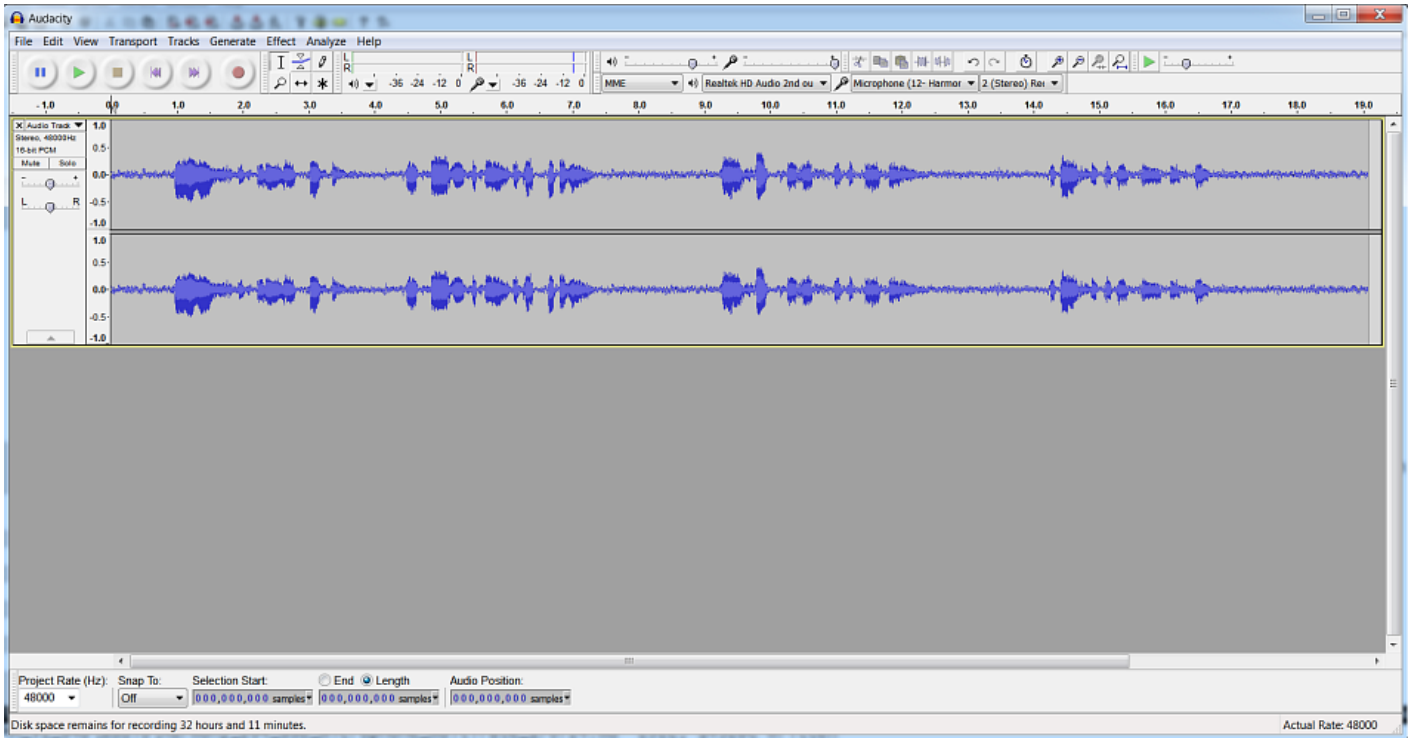


2. The PIC32 Audio DAC Daughter Board AK4642EN provides an on-board microphone. Place it near the source of audio to be recorded. The configuration using the MEB II requires an external microphone connected to the MIC IN jack.
3. Connect the board using the USB mini-B connector (for the PIC32 Bluetooth Audio Development Kit or to the mini-micro connector on the starter kit) to the Host computer with a standard USB cable.
4. Allow the Host computer to acknowledge, install drivers (if needed), and enumerate the device. No special software is necessary on the Host side.
5. If needed, configure the Host computer to use the usb_microphone as the selected audio recording device and playback device. For Windows, this is done in the "Recording" and "Playback" tabs in the "Sound" dialog (as shown in the following figure) accessed by right clicking the loudspeaker icon in the taskbar.



Note: The device "Harmony USB Microphone Example" should be available along with a sound level meter indication audio input when touching the microphone. If no sound level is registering, uninstall the driver and reboot the host computer, since it may have incorrect configuration set by a similar connection to one of the other MPLAB-X Harmony Audio Demos.

6. Open a recording application (such as Audacity, as shown in the following figure) and initiate a recording from the USB microphone source. You can also play back the recording using the usb_headset as the playback device at the same time. This can be demonstrated by overdubbing the first recording as it is playing back through the headphones by making a new recording at the same time on another track.
7. Playback of the recording should demonstrate that the audio is being received from the microphone and saved on the Host Computer.



usb_host_headset

This specifies a host interface for a USB headset connected to a smartphone.

Description

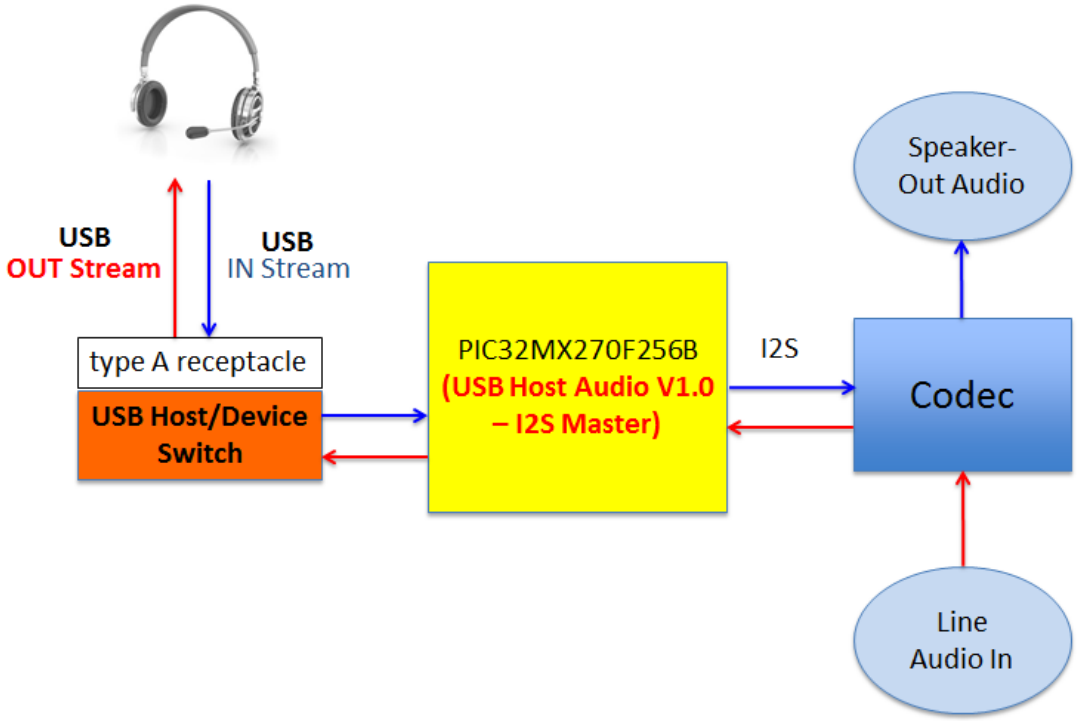
The demonstration implements a USB Audio 1.0 Host interface to connect the USB headset to a Type A USB connector at 48 kHz sampling rate, and 16-bit stereo data. The source of the audio will be an analog input (LINE-IN) to the codec. If the device is a headset, the USB microphone can also be monitored through the Codec output (HP-OUT).

Once attached, the embedded system will detect the USB headset, enumerate the available interfaces, feature units, terminal links, and endpoints, and select an OUT Stream (to the headset) and optionally an IN steam from the microphone. Both streams are sources and synced from/to the codec.

Architecture

This application runs on the PIC32 Bluetooth Audio Development Kit, which contains a PIC32MX270F512L microcontroller with 512 KB of Flash memory and 64 KB of RAM running at 48 MHz, or a PIC32MX470F512L microcontroller with 512 KB of Flash memory and 128 KB of RAM running at 96 MHz using the following hardware:

- 220x176 color LCD using the OTM2201B Controller
- USB Host interface using a Type-A connector, which must be selected in place of the USB device interface
- Five LEDs
- Two X32 sockets, one of which will be used to host a Codec module daughter board. The Codec will be used for the analog audio source to the USB headset through the 3.5 mm audio jack(LINE-IN). The HP out of the Codec DB will be used to monitor the USB Microphone, if available.



The interface between the processor and the LCD display on the PIC32 Bluetooth Audio Development Kit is an 8-bit Parallel Master Port (PMP) used for data transfer of graphics commands implemented by a graphics processor. The program uses the MPLAB Harmony 2.0 Graphics Library to draw on the screen.

The Codec utilizes one instance of I2S, with the I2S module and Codec configured, as shown in the following tables. The I2S is configured for bidirectional data flow, using the SDI1 pin for input (unused) and SDO1 for output to the headset jack. **Table 3** provides the parameters being used to generate these clocks, and configure the Codec device (either AK4642 or AK4953) over I2C. Initially, the Codec is set for 16 kHz

The I2S is set up for 16-bit data per stereo channel data transferred serially using 32-bit framing per channel (the two least significant bytes are zero and ignored by the module). The data is configured as stereo. The 16-bit stereo samples are transferred to the Codec memory buffer via DMA.

Table 1. 48 kHz Sampling I2S Clock Values

I2S Function	Value	Description/PIC32 Pin (Port)
LRCK	48.000000 kHz	Sample Rate Clock
BCLK	3072000 Hz	Bit Rate Clock
MCLK	12.288000	Master Clock

Table 2. I2S Data Pins PIC32MX480F512L (PIC32 Bluetooth Audio Development Kit)

PIC32 Name	Description	I2S	PIC32 Pin
SDI1	Serial Data In	SDI1	73 (RC13)
SD01	Serial Data Out	SDI1	72 (RD0)
SS1	Bit Clock	BCLK	69 (RD9)
SCK1	Master Clock	MCLK	70 (RD10)
REFCLK01	Reference Clock	-	53 (RF8)

Table 3. I2S Parameter Values (48 kHz Sampling)

I2S	Value	Description/PIC32 Pin (Port)
PIC32 Mode	HOST	Description
AK4642 Mode	SLAVE	Controlled by the PIC32
MCLK_MULT (mhc)	512	Sample rate multiplier for MCLK
MCLK_BCLK_RATIO	4	Sample rate multiplier for BCLK

Data Sizes	16/16/32	16 bit data/16 deep queue/32 bit frame
Channels	Stereo	2-channels
Timing	LJ	Left Justified

The application uses the USB Library as a host stack, which accepts the device connection. It is configured for six USB endpoints to account for the IN and OUT audio stream interfaces possibly requiring three endpoints (0-bandwidth, audio stream, and control).

When the USB headset is attached to the Type-A connector, the device enumeration process is executed by the application, where the device will transmit descriptors describing its function and the host application identifies the available functions. The USB headset will send a v1.0 USB device descriptor to the USB host that must contain an OUT audio stream. It will also send a descriptor for an IN Audio stream if it is a full headset and not just a headset.

The OUT stream format can be identified in the device enumeration process, as shown in the following:

```
const APP_USB_HOST_AUDIO_STREAM_FORMAT audioSpeakerStreamFormat =
{
 .streamDirection = USB_HOST_AUDIO_V1_DIRECTION_OUT,
 .format = USB_AUDIO_FORMAT_PCM,
 .nChannels = 2,
 .bitResolution = 16,
 .subFrameSize = 2,
 .samplingRate = AUDIO_SAMPLING_RATE
};
```

This indicates a stereo (2-channel) stream (nChannels) in the out direction from the host (streamDirection), with 16 bits per channel (bitResolution in bits), with a framing of 16 bits (subFrameSize * 2 bytes) at the AUDIO_SAMPLING_RATE, which is fixed for this application at 48000 Hz. The IN stream used for the USB microphone data uses the same format except for the streamDirection.

The enumerated device descriptor will contain a descriptor that can accept the required 48000 Hz sampling rate. Two forms of the descriptor are possible. The discrete frequency format is shown in the following table.

Table 4: Audio Streaming Format Type Descriptor

Descriptor Name	Value
bLength	0x0E
bDescriptorType	0x24
bDescriptorSubtype	0x02
bFormatType	0x01
bNrChannels	0x02
bSubframeSize	0x02
bBitResolution	0x10
bSamFreqType	0x01
tSamFreq	0x00BB80 (48000 Hz)

Where bSamFreqType gives the number of discrete values in the tSamFreq array, of which there is only one value given here (48000 Hz). Note that the bNrChannels, bBitResolution, and the bSubframeSize also match the usable stream values.

The second form of the descriptor gives a continuous range of frequencies, as shown in the following table.

Table 5: Audio Streaming Format Type Descriptor

Descriptor Name	Value
bLength	0x0E
bDescriptorType	0x24
bDescriptorSubtype	0x02
bFormatType	0x01
bNrChannels	0x02
bSubframeSize	0x02
bBitResolution	0x10
bSamFreqType	0x00
tLowerSamFreq	0x001F40 (8000 Hz)
tUpperSamFreq	0x00BB80 (48000 Hz)

The bSamFreqType is '0' which indicates the next two 24 bit values give a frequency range of 8000 Hz to 48000 Hz. Again this audio interface descriptor describes a usable stream interface.

The USB full speed interface uses a 1 ms packet size for audio streaming. This requires that the USB packet size is set to 48000 Hz/1000 ms/sec * 2 channels/sample * 2 bytes/channel= 192 bytes. This should match the wMaxPacketSize parameter in the Endpoint descriptor for the interface.

Software Structure

All MPLAB Harmony applications use the function SYS_Initialize located in the source file `system_init.c` and executed from the main function to initialize various subsystems such as the clock, ports, Board Support Package (BSP), PMP, Codec, timers, graphics, and interrupts. The USB, AK4642 driver, graphics, and the application state machines are all updated via calls located in the function SYS_Tasks, executed from the main polling loop, located in `system_tasks.c`.

The application code is contained in the standard source file `app.c`. The application code is initialized in SYS_Initialize using the application APP_Initialize function to instantiate driver objects and start the USB Device interface. The application utilizes a simple state machine (APP_Tasks executed from SYS_Tasks) with the following functions:

- Setup the device peripheral drivers and USB Library interface
- Responses to USB host stream requests ("Initiate stream interface completed," “Change stream sample rate completed," “Read packet received," and “Write packet sent”
- Responses to USB host control requests
- The Data buffer completes from the codec read and write buffer DMA complete interrupts, used to initiate USB buffer read and write requests

The USB interface uses a separate host and device data timing. Buffering between the Codec reads from LINE-OUT to USB OUT stream writes uses a ping-pong buffer arrangement, which is allowable since clock domain mismatches generating buffer underflow and overflow will be mitigated as the USB device. The USB host receiving the USB IN stream must mitigate underflow and overflow problems to the Codec, which is done by using a packet buffer queue. A queue length of 64 eliminates most of the dropouts due to packet underflow.

Demonstration Features

- USB headset playback using the AK4642 Codec Daughter Board LINE-IN analog audio source
- USB headset microphone monitoring using the AK4642 Codec Daughter Board HP-OUT
- USB connection to a USB audio device using the MPLAB Harmony USB Library Host Stack
- Displaying graphics on the BTADK 220x176 LCD using MPLAB Harmony Graphics Library and PMP interface to OTM0221B Graphics Controller

Tools Setup Differences

PIC32MX470F512L on PIC32 Bluetooth Audio Development Kit with AK4642 Configuration

If building this application from scratch, start by creating a 32-bit MPLAB Harmony project in MPLAB X IDE by selecting *File > New Project dialog*. Select PIC32MX Bluetooth Audio Development Kit and a configuration name, such as pic32mx270f512l_pim_btadk_ak4642, indicating the processor, the board and the Codec Daughter Board. Click the MPLAB Harmony icon to open the MPLAB Harmony Configurator (MHC). When using a PIM, select the board with the associated PIM, after selecting the device: *PIC32MX270F512L*.

Open the *Harmony Configurator Framework > Drivers* section, and then expand *Codec >*. Then, select the required *Codec/DAC Driver?*** and a sub-menu will appear. For this application, the default values will suffice, except for the following:

- The volume setting can be adjusted between a -75 dB and +12 dB gain range
- The number of clients should be 2, to accommodate a read and a write client

Expand the I2S, the *Use I2S Driver?* option should be selected. The driver callbacks require *Transmit DMA Support* (for the write client), *Receive DMA Support* (for the read client). This automatically selects Enable DMA Channel Interrupts for the callback when the buffer transfer has completed.

Note: The AK4953 driver is also used for the AK4954 Codec.

To use the Graphics Stack Library, within *Graphics Stack*, select *Use Graphics Stack?*. Further down, within *Use Harmony Graphics Composer Suite? > Middleware > Use Aria User Interface Library?*, clear *Enable Touch*. Since the board uses the Crystal Fontz 176x220 pixel LCD with the OTM2201A Graphics Controller the PIC32 Bluetooth Audio Development Kit does not support a touchscreen. Within *PMP*, select *Use PMP Driver?* This is used for graphics data transfer. Expand that section, and change the value of *Strobe Wait Sates* to *PMP_STROBE_WAIT_4*.

To use the USB Stack Library, within *USB Library*, select *Use USB Stack Under Select Host* or *Device Stack Select Host*. Change the *Number of Endpoints Used* to 6" and also select *Use Audio v1.0 Host Client Driver*. The number of *AUDIO v1.0 Streaming Interfaces* is 2 (IN and OUT streaming). The Number of *Audio V1.0 Sampling Frequencies* is 0 since the application overrides the driver and accepts all the frequencies given by the device audio stream descriptor and ignores this value.

Building the Application

This section identifies the MPLAB X IDE project name, location,lists and descriptions for the available configurations of the demonstration.

Description

To build this project, you must open the `usb_host_headset.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/audio`

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`usb_host_headset.X`	`<install-dir>/apps/audio/usb_host_headset/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
bt_audio_dk_ak4642	bt_audio_dk+4642	This demonstration runs on the PIC32MX470F512L device mounted on the PIC32 Bluetooth Audio Development Kit with the AK4642 Codec Daughter Board.
pic32mx270f512l_pim_bt_audio_dk_ak4642	pic32mx270f512l_pim+bt_audio_dk	This demonstration runs on the PIC32 Bluetooth Audio Development Kit with the PIC32MX270F512L Plug-in Module (PIM) and AK4642 Codec Daughter Board.

Configuring the Hardware

This section describes how to configure the supported hardware.

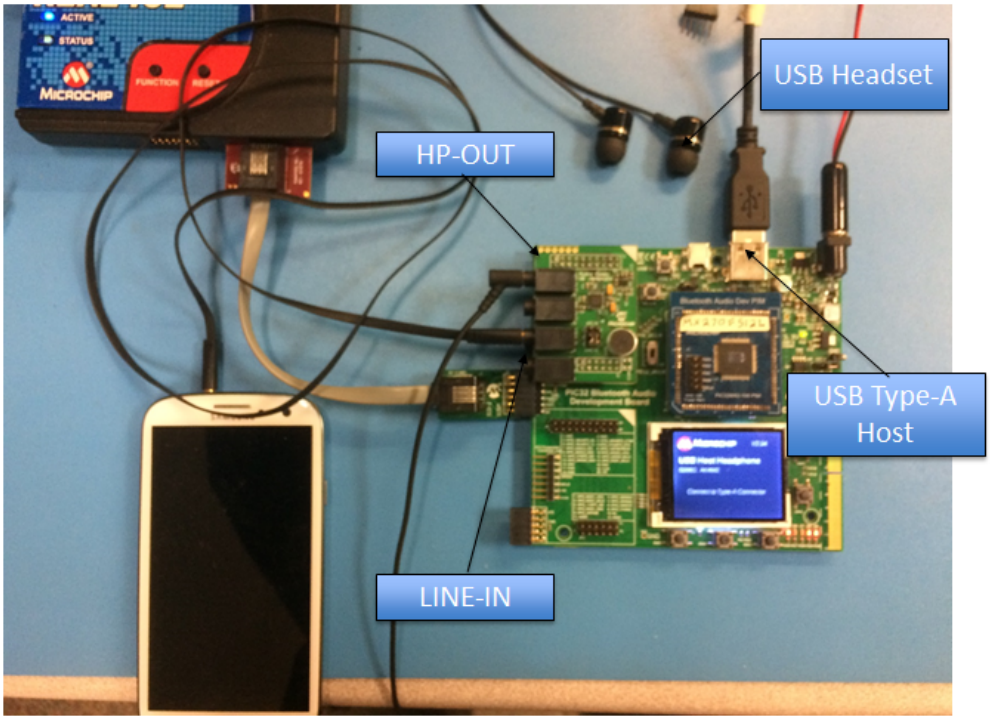
Description

PIC32 Bluetooth Audio Development Kit with the PIC32MX470F512L and the AK4642 Codec Module daughter board.

Connect switch S1 located in the middle of the PIC32 Bluetooth Audio Development Board to PIC32_MCLR. Connect the AK4642 Codec Daughter Board to the X32 Connector, as shown in the following figure.

PIC32 Bluetooth Audio Development Kit with the PIC32MX270F512L PIM, and the AK4642 Codec Module daughter board.

Attach the PIM and switch S1 located in the middle of the PIC32 Bluetooth Audio Development Board to PIM_MCLR. Attach the AK4642 Codec Daughter Board to the X32 Connector, as shown in the following figure.



Running the Demonstration

This section demonstrates how to run the demonstration.

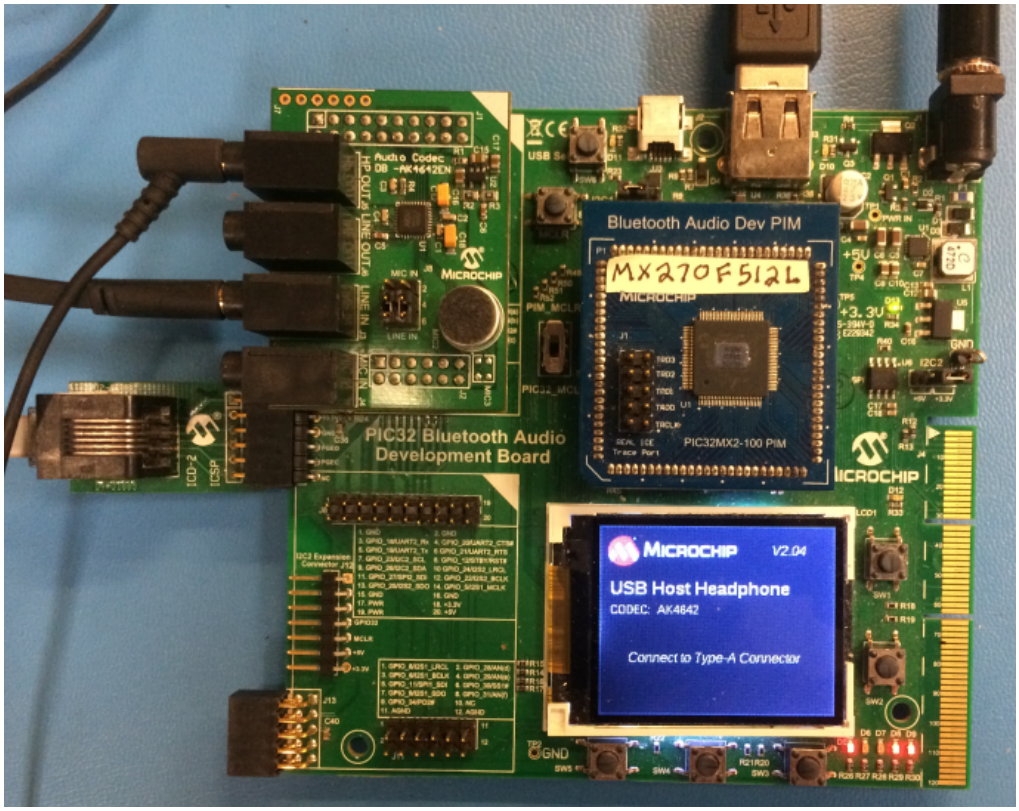
Description

Important!

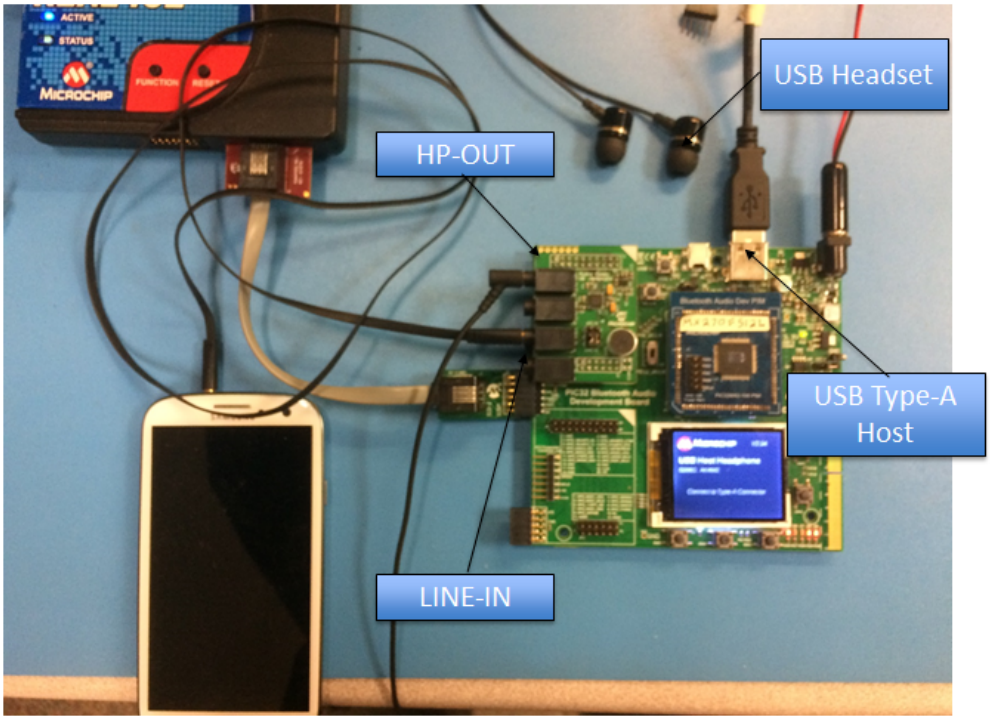
Prior to using this demonstration, it is recommended to review the MPLAB Harmony Release Notes for any known issues. A PDF copy of the release notes is provided in the `<install-dir>/doc` folder of your installation.

Do the following to run the demonstration:

1. Power up the board and connect the programmer. Compile the application, program the target device, and run it. After the program starts, the name of the application should appear on the display (for those configurations with a display), as shown in the following figure.



2. Connect a line audio source to the 3.5 mm LINE-IN jack for the AK4642 Codec Daughter Board (see the following figure).
3. Connect the board to the USB Headset to the Type-A connector (see the following figure).
4. LED 1 indicates the device attachment. LED 4 indicates that the OUT Stream has been enumerated and that the LINE-IN audio can be heard in the headset. LED 5 indicates that the IN stream has been enumerated and that the headset microphone can be monitored on the HP-OUT jack on the AK4642 Codec Daughter Board.



usb_microphone

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This demonstration application configures a USB microphone system operating at a 16 kHz sampling rate with 16-bit data.

The system will interface to a USB Host (such as a personal computer), which can accommodate a USB device class microphone. The embedded system will enumerate with a USB audio device endpoint and enable the host system to input audio from the USB port at 16 kHz/16-bit stereo using a standard USB Full-Speed implementation.

The embedded system will take the data from a microphone via the Codec Driver and send it to the audio USB interface. The Codec Driver sets up the audio output interface, timing, DMA channels and buffer queue to enable a continuous audio stream. The digital audio is received from the Codec through an I2S data channel at 16 kHz via USB to the host computer. The AK4642 Codec is utilized with the PIC33MX470F512L microcontroller on the PIC32 Bluetooth Audio Development Kit.

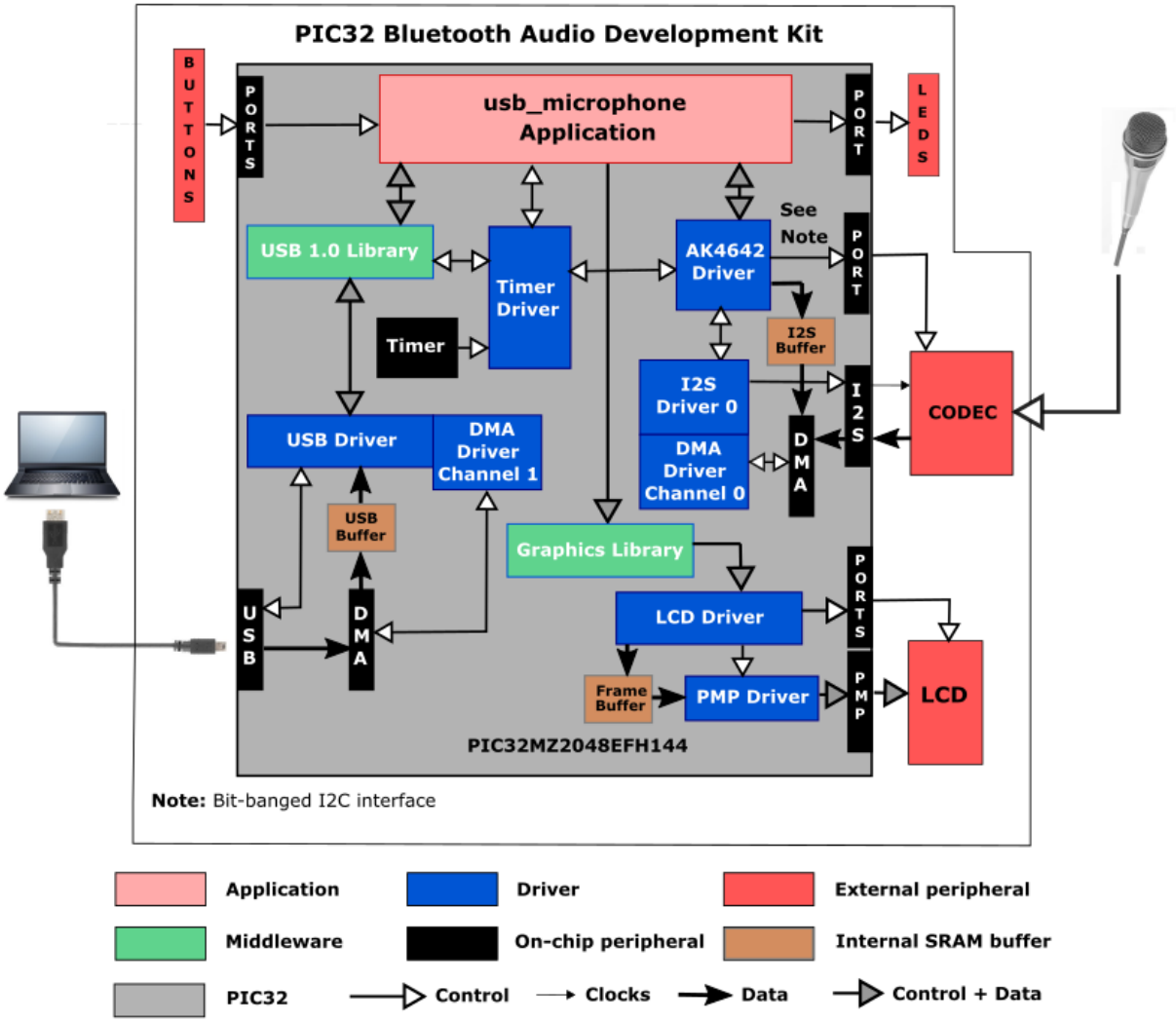
Architecture

This application runs on the PIC32 Bluetooth Audio Development Kit, which contains a PIC32MX470F512L microcontroller with 512 KB of Flash memory and 128 KB of RAM running at 96 MHz, using the following hardware:

- 220x176 color LCD
- Five LEDs
- USB Device interfaces
- Two X32 sockets, one of which will be used to host a AK4642 Codec module daughter board having both an on-board internal microphone that will be used for the demonstration, and an external microphone input through a 3.5 mm audio jack

The PIC32 Bluetooth Audio Development Kit comes with a BTM805 Bluetooth daughter board and an AK4384 Audio DAC daughter board; however, you must replace the AK4384 daughter board with the AK4642 daughter board.

The following figure shows the application structure and hardware used. The PIC32 device runs all of the application code.



The interface between the PIC32 and the LCD display is an 8-bit parallel master port (PMP) used for data transfer. The program uses the MPLAB Harmony 2.0 Graphics Library to draw on the screen. The buttons are interfaced using GPIO pins.

The AK4642 utilizes one instance of I2S at Port 1, with the I2S module and Codec configured as shown in the following tables. The I2S is

configured for bidirectional data flow, however only the SDI1 pin is utilized in order to receive the microphone data. The I2S Parameters Values table shows the parameters being used to generate these clocks and configure the Codec device over I2C. The I2S is set up for 16-bit data per stereo channel data transferred serially using 32 bit framing per channel (the 2 least significant bytes are 0 and ignored by the module). The data is configured as stereo, despite the microphone data being mono with the data duplicated across both 16 kHz channels (left and right). The 16-bit stereo samples are transferred to the memory buffer via DMA Channel 1.

The I2S clock values generated by the PIC32 master are derived from the following formulas, starting with the REFCLK0 value (4.096 Mhz) used for the I2S MCLK and the sample rate, Fs (48000 Hz):

MCLK = 12,288,000 Hz

Fs = 16000 Hz

NUM_CHANNELS = 2 channels

FRAME_SIZE = 32 bits/channel * NUM_CHANNELS = 64 bits

BCLK = FRAME_SIZE * Fs

LRCK = Fs

The codec slave uses the following parameters calculated from the clock values:

MCLK_DIV = MCLK/LRCK

BCLK_DIV = MCLK/BCLK

The table shows the calculated parameters and generated clock rates.

16 kHz Sampling I2S Clock Values

I2S Function	Value	Description/PIC32 Pin (Port)
LRCK	16.000000 kHz	Sample rate clock /SS1 on pin 69 (RD1)
BLCK	1024000 Hz	Bit rate clock /SCK1 on pin 70 (RD10)
MCLK	4.096000 MHz	Master clock /REFCLK0 on pin 53 (RF8)

I2S Data Pins

PIC32 Name	Description	I2S	PIC32 Bluetooth Audio Development Kit Pin
SDI1	Serial Data In	SDI1	73 (RC13)
SDO1	Serial Data Out	SDI1	72 (RD0)

I2S Parameter Values

I2S	Value	Description/PIC32 Pin (Port)
PIC32 Mode	HOST	
AK4642 Mode	SLAVE	Controlled by the PIC32
MCLK_MULT (mhc)	256	16 kHz sample rate multiplier for MCLK
MCLK_BCLK_RATIO	4	16 kHz sample rate multiplier for BCLK
Data Sizes	16/16/32	16-bit data/16-deep queue/32-bit frame
Channels	Stereo	2-channel
Timing	LJ	Left-justified

The application uses the USB Library as a "Device" stack, which will connect to a "Host". It is configured for five USB endpoints with a single function having three interfaces. All of these are defined for a USB Microphone device in the fullSpeedConfigurationDescriptor array variable structure (located in `system_init.c`). This structure defines the connection to the host, with a sample rate of 48 kHz, 16-bit mono channel data. buffer queue of size 2 is used for USB packets of size 482 samples 2 bytes/sample = 192 bytes. This gives a pin-pong buffer arrangement using 2 ms buffers.

All MPLAB Harmony applications use the function SYS_Initialize located in the source file `system_init.c` and executed from main to initialize various sub-systems such as the clock, ports, board support package (BSP), PMP, Codec, timers, UART, graphics, and interrupts. The USB, AK4642 driver, graphics, and the application state machines are all updated via calls located in the function SYS_Tasks, executed from the main polling loop, located in `system_tasks.c`.

The application code is contained in the standard source file `app.c`. The application code is initialized in SYS_Initialize using the application APP_Initialize function to instantiate driver objects and start the USB Device interface. The application utilizes a simple state machine (APP_Tasks executed from SYS_Tasks) with the following functions:

- Sets up the drivers and USB Library interface
- Responds to USB Host commands ("USB Record at 48Khz")
- Initiates and maintains the microphone data audio streaming for "USB Record" function.

Demonstration Features

- Microphone data input using a AK4642 codec daughter board
- USB connection to a host system using the USB Library Device Stack
- Displaying graphics on the PIC32 Bluetooth Audio Development Kit 220x176 LCD using MPLAB Harmony Graphics Library and the Parallel Master Port (PMP) Driver Library

Tools Setup Differences

If building this application from scratch, start by creating a 32-bit MPLAB Harmony project in MPLAB X IDE by selecting *File* > *New Project*. Select the PIC32MX Bluetooth Audio Development Kit and the configuration name of bt_audio_dk_AK4642. Click the green MPLAB Harmony icon to open the MPLAB Harmony Configurator (MHC).

Expand *Harmony Configurator Framework* > *Drivers*, and then expand *Codec* > *AK4642*. Select **Use AK4642 Driver?** and a sub-menu will appear. For this application the default values will suffice, although volume setting can be adjusted between a -75 dB and +12 dB gain range. The I2S and I2C driver (default to DRV_I2S_INSTANCE_0 and DRV_I2C_INSTANCE_0, respectively) associated to the Codec driver can be configured, as described previously.

To use the Graphics Stack Library, under *Graphics Stack*, select **Use Graphics Stack?**. Further down, under *Use Harmony Graphics Composer Suite?* > *Middleware* > *Use Aria User Interface Library?*, clear **Enable Touch**. Since the board uses the Crystal Fontz 176x220 pixel LCD with the OTM2201A controller the PIC32 Bluetooth Audio Development Kit does not support a touchscreen. Under PMP, select **Use PMP Driver?**, which is used for graphics data transfer. Expand that section, and change the value of Strobe Wait Sates to PMP_STROBE_WAIT_4.

To use the USB Stack Library, under *USB Library*, select either **Use USB Stack Under Select Host** or **Device Stack Select Device**. Change the Number of Endpoints Used to 5. Below that select USB Device Instance 0 and under that make Function 1 (matching the fullSpeedConfigurationDescriptor struct in system_init.c: AUDIO Device Class, 3 Interfaces, 2 Audio Streaming interfaces; with a Write Queue Size of 2 (for the ping-pong buffer arrangement) and a Read Queue Size of 2 (not used). The Product ID Selection gives the name shown at the Host, which in this case is "usb_microphone_demo". A USB Interrupt Priority of 1 is also used.

Note: Do not change the current fullSpeedConfigurationDescriptor values in `system_init.c` when using MHC to generate the project files.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the demonstration.

Description

To build this project, you must open the `usb_microphone.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/audio/usb_microphone`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`usb_microphone.X`	`<install-dir>/apps/audio/usb_microphone/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
bt_audio_dk_ak4642	bt_audio_dk	This configuration runs on the PIC32 Bluetooth Audio Development Kit with the Audio Codec Daughter Board AK4642EN.

Configuring the Hardware

This section describes how to configure the supported hardware.

Description

PIC32 Bluetooth Audio Development Kit with the Audio Codec Daughter Board AK4642EN

Switch S1 on the PIC32 Bluetooth Audio Development Board should be set to PIC32_MCLR.

Running the Demonstration

This section demonstrates how to run the demonstration.

Description

Important! Prior to using this demonstration, it is recommended to review the MPLAB Harmony Release Notes for any known issues. A PDF copy of the release notes is provided in the `<install-dir>/doc` folder of your installation.

Compile and program the target device. While compiling, select the appropriate MPLAB X IDE project configuration based on the demonstration board. These configurations set the target processor, the board and the Codec to be used in the output interface. Refer to Building the Application for details.

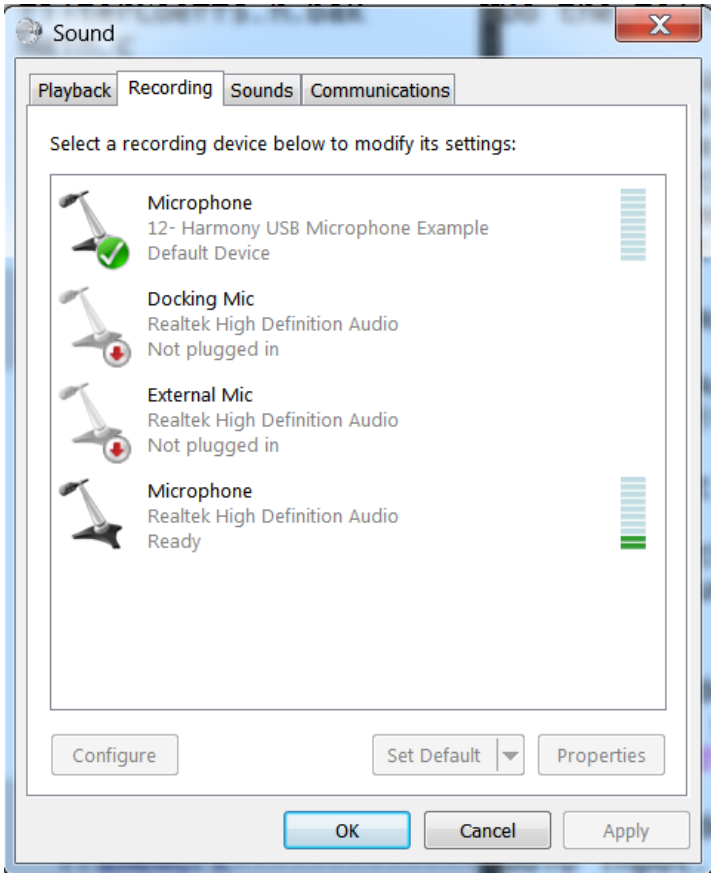
Do the following to run the demonstration:

1. Power up the board and connect the programmer. Compile the application, program the target device, and run it. After the program starts, the name of the application should appear on the display.

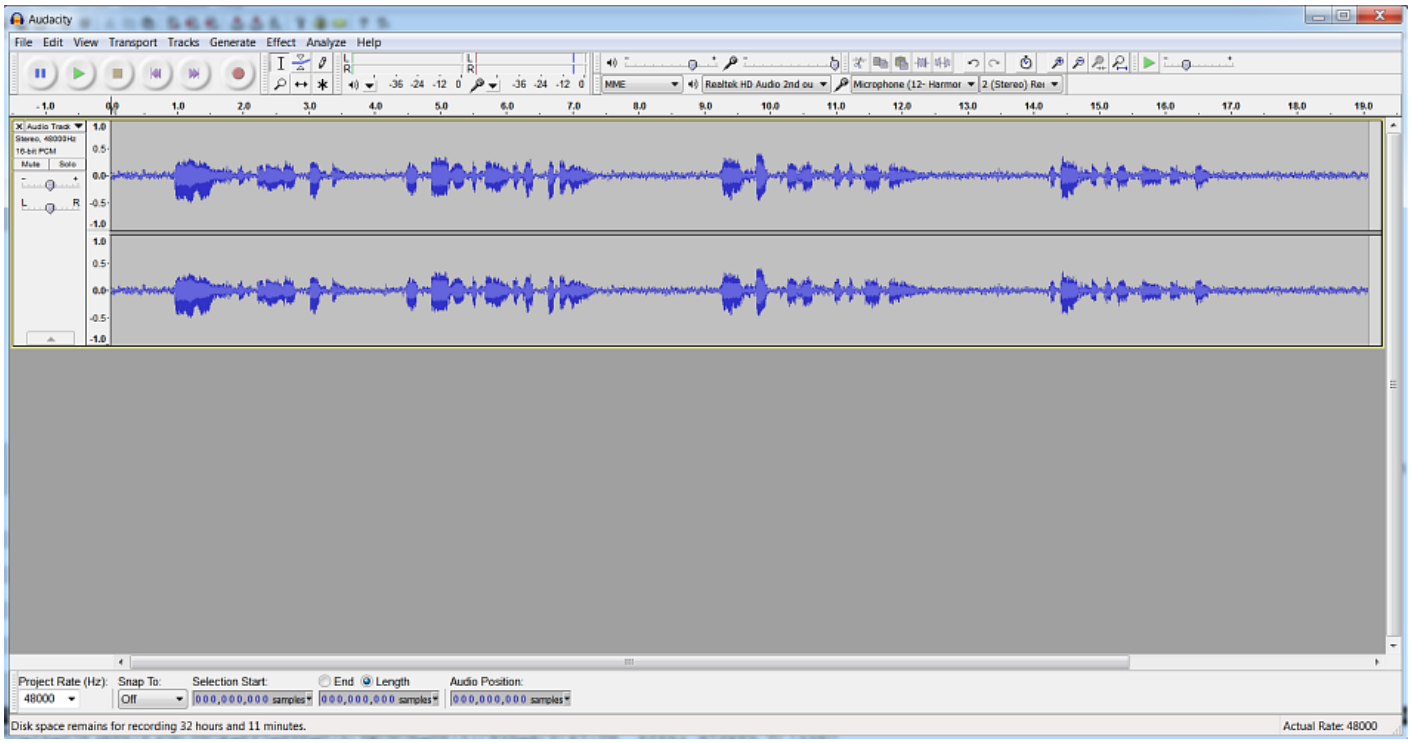


2. The PIC32 Audio DAC Daughter Board AK4642EN provides an on-board microphone. Place it near the source of audio to be recorded.
3. Connect the board using the USB mini-B connector (Device) to the Host computer with a standard USB cable.
4. Allow the Host computer to acknowledge, install drivers (if needed), and enumerate the device. No special software is necessary on the Host side.
5. Audio recording device. For Windows, this is done in the "Recording" tab in the "Sound" dialog (as shown in the following figure) accessed by right clicking the loudspeaker icon in the taskbar.

Note: The device "Harmony USB Microphone Example" should be available along with a sound level meter indication audio input when touching the microphone. If no sound level is registering, uninstall the driver and reboot the host computer, since it may have incorrect configuration set by a similar connection to one of the other MPLAB Harmony audio Demonstrations.



6. Open a recording application (such as Audacity, as shown in the following figure) and initiate a recording from the USB microphone source.



7. Playback of the recording should demonstrate that the audio is being received from the microphone and saved on the Host Computer.
8. Listen to the audio output on the speakers or headphones connected to the board. The volume will typically be adjusted by the host.

usb_microphone_multirate

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This demonstration application configures a USB microphone system operating at a 16 kHz sampling rate with 16-bit data.

The system will interface to a USB Host (such as a personal computer), which can accommodate a USB device class microphone. The embedded system will enumerate with a USB audio device endpoint and enable the host system to input audio from the USB port at 16 kHz/16-bit stereo using a standard USB Full-Speed implementation.

The embedded system will take the data from a microphone via the Codec Driver and send it to the audio USB interface. The Codec Driver sets up the audio output interface, timing, DMA channels and buffer queue to enable a continuous audio stream. The digital audio is received from the Codec through an I2S data channel at 16 kHz via USB to the host computer. The AK4642 Codec is utilized with the PIC33MX470F512L microcontroller on the PIC32 Bluetooth Audio Development Kit.

Architecture

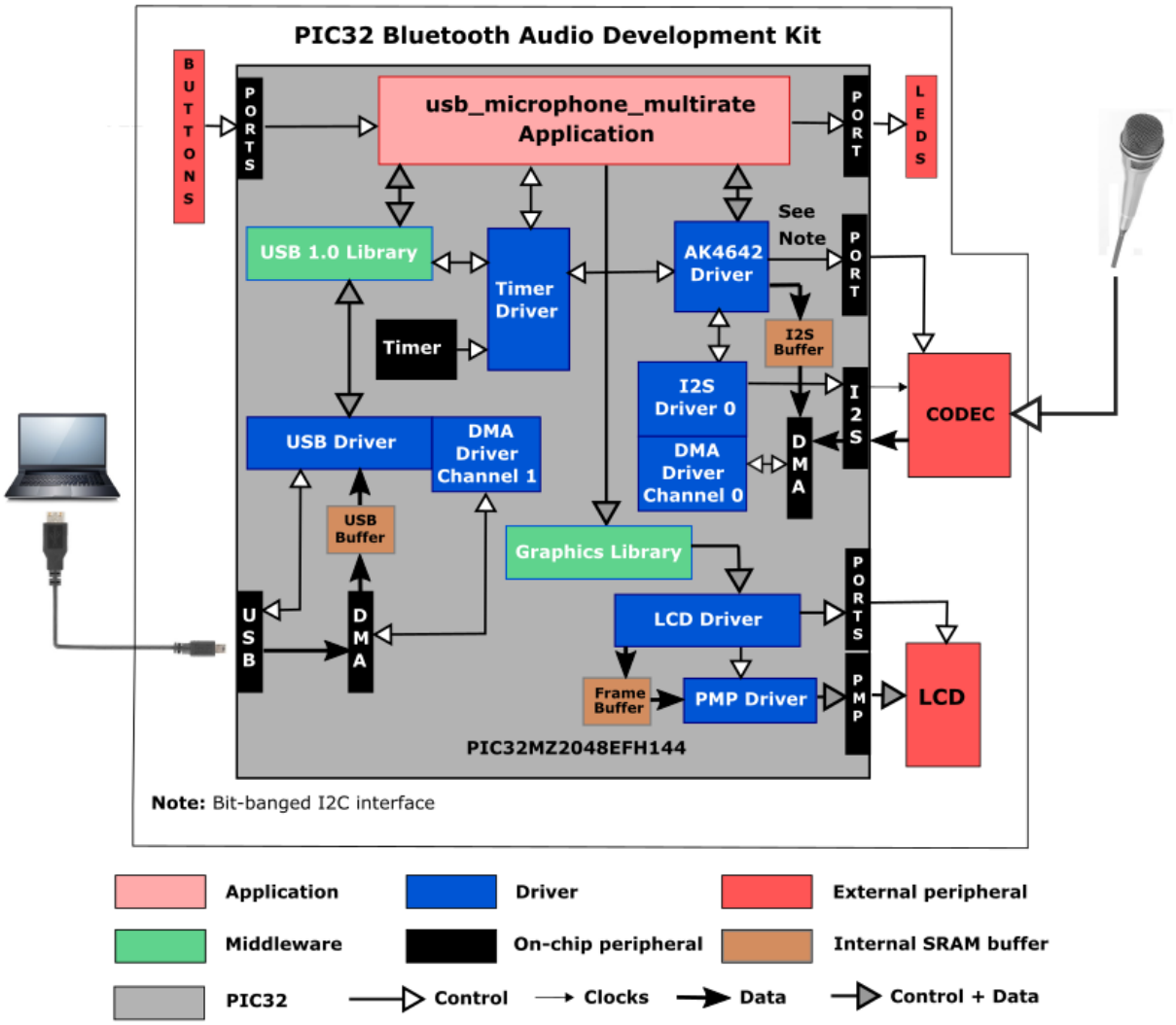
The board can also be configured for an external microphone through the 3.5 mm audio jack.

This application runs on the PIC32 Bluetooth Audio Development Kit, which contains a PIC32MX470F512L microcontroller with 512 KB of Flash memory and 128 KB of RAM running at 96 MHz with the following features:

- 220x176 color LCD
- Five LEDs
- USB Device interfaces
- Two X32 sockets, one of which will be used to host a AK4642 Codec module daughter board having both an on-board internal microphone that will be used for the demonstration, and an external microphone input through a 3.5 mm audio jack

The PIC32 Bluetooth Audio Development Kit comes with a BTM805 Bluetooth daughter board and an AK4384 Audio DAC daughter board; however, you must replace the AK4384 daughter board with the AK4642 daughter board. The BTM805 daughter board is unused.

The following figure shows the application structure and hardware used. The PIC32 device runs all of the application code.



The interface between the PIC32 and the LCD display is an 8-bit parallel master port (PMP) used for data transfer. The program uses the MPLAB Harmony 2.0 Graphics Library to draw on the screen. The buttons are interfaced using GPIO pins.

As with any MPLAB Harmony application, the function SYS_Initialize located in the source file `system_init.c` makes calls to initialize various sub-systems, such as the clock, ports, board support package (BSP), PMP, Codec, timers, UART, graphics, and interrupts. The USB, Codec driver, graphics, and the application state machines are all updated via calls located in the function SYS_Tasks in `system_tasks.c`. Interrupt handlers in the file `system_interrupt.c` are only used for receiving characters from the UART, and for the two timers

The AK4642 utilizes one instance of I2S at Port 1, with the I2S module and Codec configured as shown in the following tables. The I2S is configured for bidirectional data flow, however only the SDI1 pin is utilized in order to receive the microphone data. The I2S Parameters Values table shows the parameters being used to generate these clocks and configure the Codec device over I2C. The I2S is set up for 16-bit data per stereo channel data transferred serially using 32 bit framing per channel (the 2 least significant bytes are 0 and ignored by the module). The data is configured as stereo, despite the microphone data being mono with the data duplicated across both 16 kHz channels (left and right). The 16-bit stereo samples are transferred to the memory buffer via DMA Channel 1.

The I2S clock values generated by the PIC32 master are derived from the following formulas, starting with the REFCLK0 value (4.096 Mhz) used for the I2S MCLK and the sample rate, Fs (48000 Hz):

MCLK = 12288000Hz

Fs = 48000

NUM_CHANNELS = 2 channels

FRAME_SIZE = 32 bits/channel * NUM_CHANNELS

BCLK = FRAME_SIZE * Fs

LRCK = Fs

The codec slave uses the following parameters calculated from the clock values:

MCLK_DIV = MCLK/LRCK

BCLK_DIV = MCLK/BCLK

The table shows the calculated parameters and generated clock rates.

16 kHz Sampling I2S Clock Values

I2S Function	Value	Description/PIC32 Pin (Port)
LRCK	48000 Hz	Sample rate clock /SS1 on pin 69 (RD1)
BCLK	1,536,000 Hz	Bit rate clock /SCK1 on pin 70 (RD10)
MCLK	12,288,000 Hz	Master clock /REFCLK0 on pin 53 (RF8)

I2S Data Pins

PIC32 Name	Description	I2S	PIC32 Bluetooth Audio Development Kit Pin
SDI1	Serial Data In	SDI1	73 (RC13)
SDO1	Serial Data Out	SDI1	72 (RD0)

I2S Parameter Values

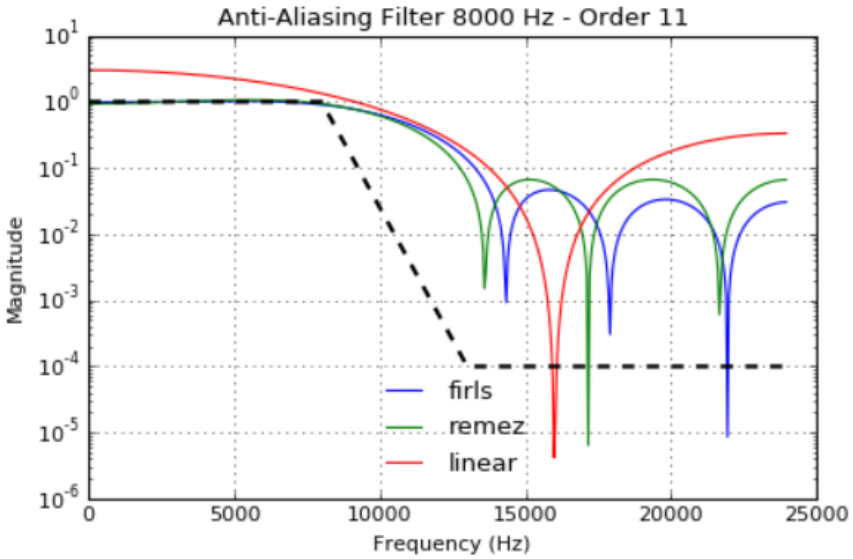
I2S	Value	Description/PIC32 Pin (Port)
PIC32 Mode	HOST	
AK4642 Mode	SLAVE	Controlled by the PIC32
MCLK_MULT (mhc)	256	48 kHz sample rate multiplier for MCLK
MCLK_BCLK_RATIO	4	48 kHz sample rate multiplier for BCLK
Data Sizes	16/16/32	16-bit data/16-deep queue/32-bit frame
Channels	Stereo	2-channel
Timing	LJ	Left-justified

The application uses the USB Library as a "Device" stack, which will connect to a "Host". It is configured for five USB endpoints with a single function having three interfaces. All of these are defined for a USB Microphone device in the fullSpeedConfigurationDescriptor array variable structure (located in `system_init.c`). This structure defines the connection to the host, with a sample rate of 48 kHz, 16-bit mono channel data. buffer queue of size 2 is used for USB packets of size 482 samples 2 bytes/sample = 192 bytes. This gives a pin-pong buffer arrangement using 2 ms buffers.

All MPLAB Harmony applications use the function SYS_Initialize located in the source file `system_init.c` and executed from main to initialize various sub-systems such as the clock, ports, board support package (BSP), PMP, Codec, timers, UART, graphics, and interrupts. The USB, AK4642 driver, graphics, and the application state machines are all updated via calls located in the function SYS_Tasks, executed from the main polling loop, located in `system_tasks.c`.

The conversion of the 16 kHz/16-bit stereo buffers from the Codec/I2S to the 48 kHz/16-bit mono buffers requires that one input channel be removed and that the SRC be applied as a 3x upsample on the remaining channel. A single stage upsample filter using the efficient multi-rate linear interpolation FIR structure is used with a 6-tap filter and three phases of 2-taps/phase each (see **Note**). The advantage of this structure is that a single 5th order filter can be used to calculate all three output samples at the 16 kHz input rate. The response of the linear interpolation filter showing the rejection of aliasing artifacts (in red), as shown in the following figure. Other filters (11th order with 3-taps/phase, as shown in blue and green) can be designed the give even greater rejection.

Note: Refer to *Multirate Systems and Filter Banks* by P. P. Vaidyanathan, Prentice-Hall, 1993. ISBN-13: 978-0136057185; ISBN-10: 0136057187.



The application code is contained in the standard source file `app.c`. The application code is initialized in SYS_Initialize using the application APP_Initialize function to instantiate driver objects and start the USB Device interface. The application utilizes a simple state machine (APP_Tasks executed from SYS_Tasks) with the following functions:

- Sets up the drivers and USB Library interface
- Responds to USB Host commands ("USB Record at 48Khz")
- Initiates and maintains the microphone data audio streaming for "USB Record" function.

Demonstration Features

- Microphone data input using a Audio Codec Daughter Board AK4642EN
- Sample Rate Conversion using the LibQ Fixed-Point 'C' Math Library to implement an efficient 3x upsampler
- USB connection to a host system using the USB Library Device Stack
- Displaying graphics on the PIC32 Bluetooth Audio Development Kit 220x176 LCD using the MPLAB Harmony Graphics Library and the Parallel Master Port (PMP) Driver Library

Tools Setup Differences

If building this application from scratch, start by creating a 32-bit MPLAB Harmony project in MPLAB X IDE by selecting *File > New Project*. Select the PIC32MX Bluetooth Audio Development Kit and the configuration name of bt_audio_dk_AK4642. Click the green MPLAB Harmony icon to open the MPLAB Harmony Configurator (MHC).

Expand *Harmony Configurator Framework > Drivers*, and then expand *Codec > AK4642*. Select **Use AK4642 Driver?** and a sub-menu will appear. For this application the default values will suffice, although volume setting can be adjusted between a -75 dB and +12 dB gain range. The I2S and I2C driver (default to DRV_I2S_INSTANCE_0 and DRV_I2C_INSTANCE_0, respectively) associated to the Codec driver can be configured, as described previously.

To use the Graphics Stack Library, under *Graphics Stack*, select **Use Graphics Stack?**. Further down, under *Use Harmony Graphics Composer Suite? > Middleware > Use Aria User Interface Library?*, clear **Enable Touch**. Since the board uses the Crystal Fontz 176x220 pixel LCD with the OTM2201A controller the PIC32 Bluetooth Audio Development Kit does not support a touchscreen. Under PMP, select **Use PMP Driver?**, which is used for graphics data transfer. Expand that section, and change the value of Strobe Wait Sates to PMP_STROBE_WAIT_4.

To use the USB Stack Library, under *USB Library*, select either **Use USB Stack Under Select Host** or **Device Stack Select Device**. Change the Number of Endpoints Used to 5. Below that select USB Device Instance 0 and under that make Function 1 (matching the fullSpeedConfigurationDescriptor struct in system_init.c: AUDIO Device Class, 3 Interfaces, 2 Audio Streaming interfaces; with a Write Queue Size of 2 (for the ping-pong buffer arrangement) and a Read Queue Size of 2 (not used). The Product ID Selection gives the name shown at the Host, which in this case is "usb_microphone_demo". A USB Interrupt Priority of 1 is also used.

Note: Do not change the current fullSpeedConfigurationDescriptor values in `system_init.c` when using MHC to generate the project files.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the demonstration.

Description

To build this project, you must open the `usb_microphone_multirate.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/audio/usb_microphone_multirate`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`usb_microphone_multirate.X`	`<install-dir>/apps/audio/usb_microphone_multirate/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
bt_audio_dk_ak4642	bt_audio_dk	This configuration runs on the PIC32 Bluetooth Audio Development Kit with the Audio Codec Daughter Board AK4642EN.

Configuring the Hardware

This section describes how to configure the supported hardware.

Description

PIC32 Bluetooth Audio Development Kit with the Audio Codec Daughter Board AK4642EN

Switch S1 on the PIC32 Bluetooth Audio Development Board should be set to PIC32_MCLR.

Running the Demonstration

This section demonstrates how to run the demonstration.

Description

Important! Prior to using this demonstration, it is recommended to review the MPLAB Harmony Release Notes for any known issues. A PDF copy of the release notes is provided in the `<install-dir>/doc` folder of your installation.

Compile and program the target device. While compiling, select the appropriate MPLAB X IDE project configuration based on the demonstration board. These configurations set the target processor, the board and the Codec to be used in the output interface. Refer to Building the Application for details.

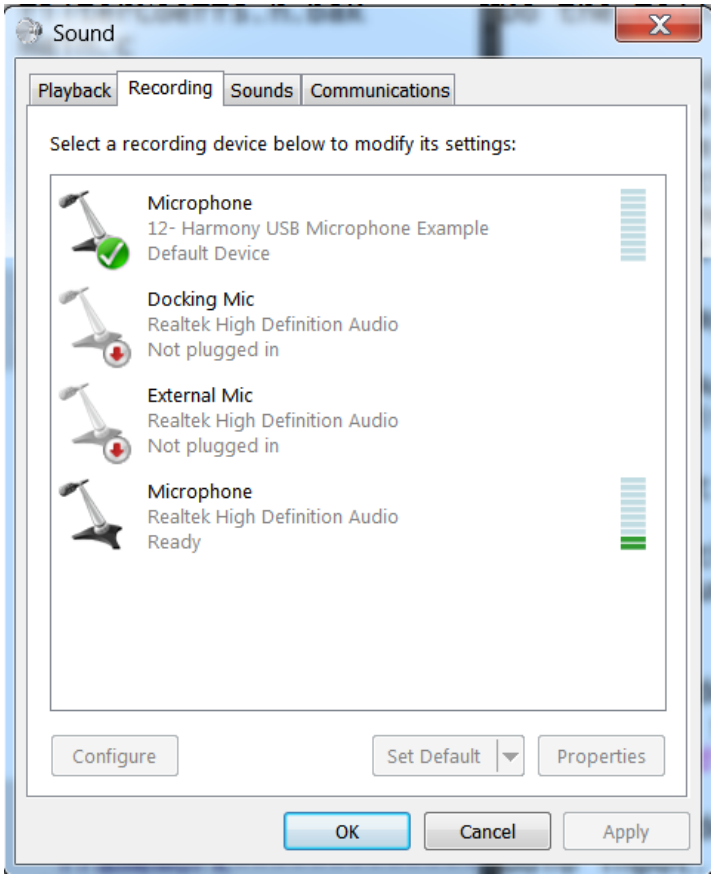
Do the following to run the demonstration:

1. Power up the board and connect the programmer. Compile the application, program the target device, and run it. After the program starts, the name of the application should appear on the display.

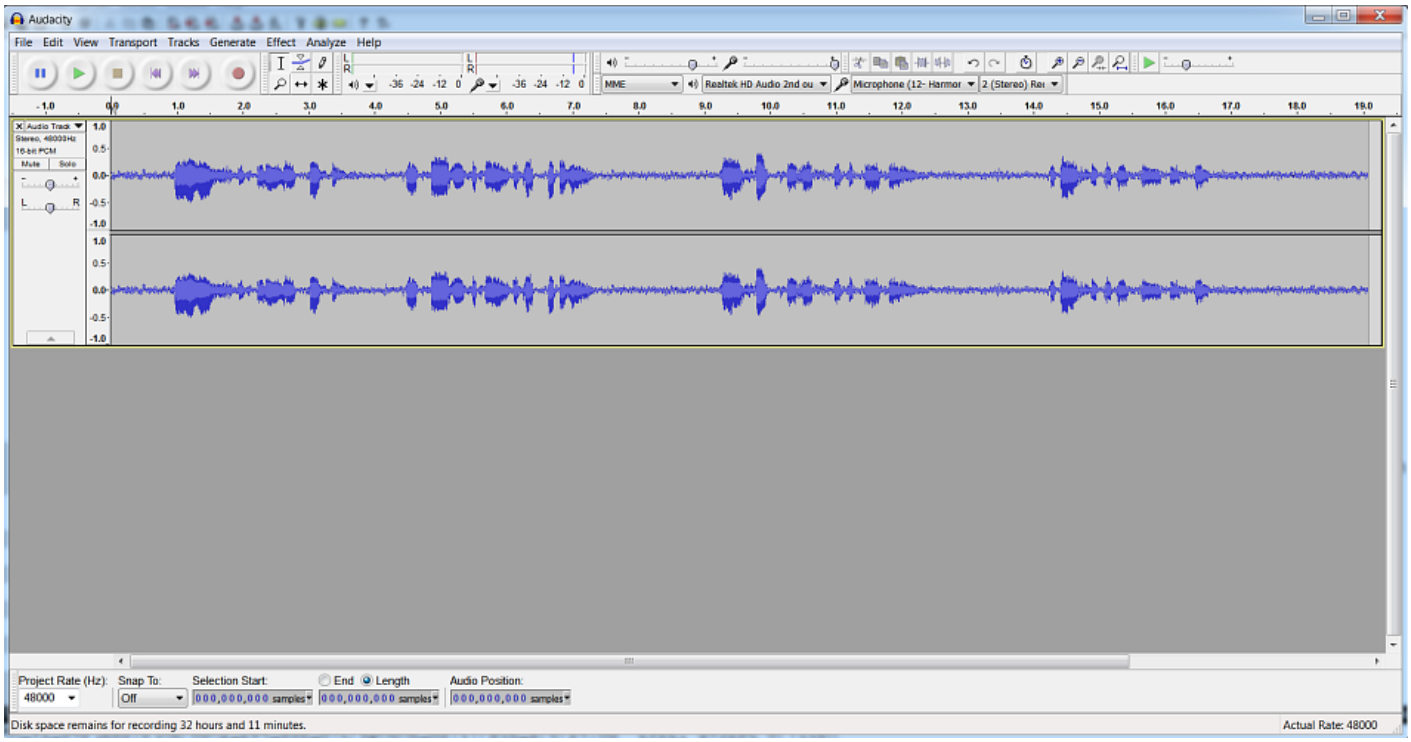


2. The PIC32 Audio DAC Daughter Board AK4642EN provides an on-board microphone. Place it near the source of audio to be recorded.
3. Connect the board using the USB mini-B connector (Device) to the Host computer with a standard USB cable.
4. Allow the Host computer to acknowledge, install drivers (if needed), and enumerate the device. No special software is necessary on the Host side.
5. audio recording device. For Windows, this is done in the "Recording" tab in the "Sound" dialog (as shown in the following figure) accessed by right clicking the loudspeaker icon in the taskbar.

Note: The device "Harmony USB Microphone Example" should be available along with a sound level meter indication audio input when touching the microphone. If no sound level is registering, uninstall the driver and reboot the host computer, since it may have incorrect configuration set by a similar connection to one of the other MPLAB Harmony audio Demonstrations.



6. Open a recording application (such as Audacity, as shown in the following figure) and initiate a recording from the USB microphone source.



7. Playback of the recording should demonstrate that the audio is being received from the microphone and saved on the Host Computer.

usb_smart_speaker

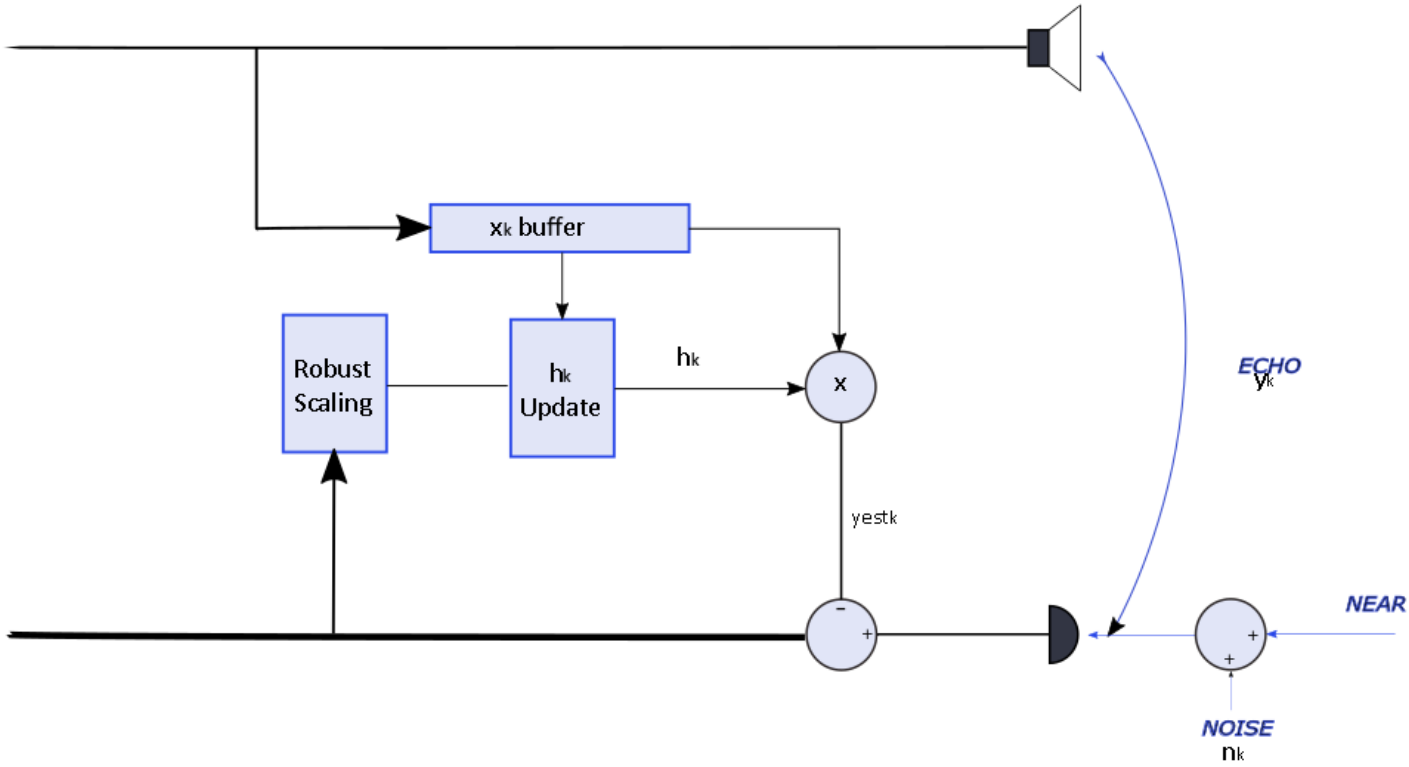
This demonstrates full-duplex USB audio playback and recording through a loudspeaker and an input microphone, respectively. The echo of the playback audio to the microphone is removed using Acoustic Echo Cancellation (AEC), which allows near speech to be recorded without interference. It is possible to use the recorded near speech to be used for the local *trigger word or phrase recognition* that initiates more involved speech recognition tasks, such as required for Google Voice Assistant and Amazon Alexa server API's.

Description

In this demonstration application:

1. Record and Playback audio streams are setup to a PC Through a PIC32 USB V1.1 peripheral audio device interface.
2. The users voice is received using the Codec (AK4953) microphone interface, while the playback occurs through the Codec line out interface to an amplified loudspeaker.
3. The echo of the playback is used to train a robust normalized Least Mean Squares (rnLMS) adaptive filter during playback, which is used to cancel that same acoustic echo when someone speaks into the microphone.

The adaptive filter block diagram is shown below:



The rnLMS algorithm is used to adapt the coefficients of a 512 tap FIR filter running at an 8Khz sampling rate. This filter structure incorporates features of robust error scaling to mitigate divergence of the FIR filter coefficients, h_k, once the filter has converged to a high quality solution of the echo path.

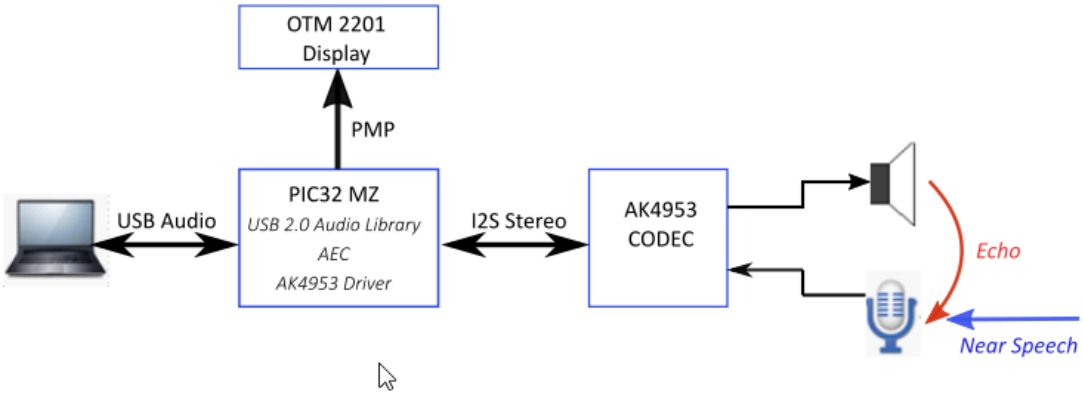
Audacity can be used for playback of one track, and concurrent recording to another track, to demonstrate echo cancellation and to measure performance of the echo canceller. During Single Talk (ST), when only playback is occurring, the residual echo can be measured along with low level approximately stationary background noise. The following measurements provide performance information of the AEC:

1. ERL (Echo Return Loss): the ratio of the loudspeaker input to the power of the echo received.
2. ERLE (Echo Return Loss Enhancement) measurement: the ratio of the echo residual to input echo.

During Double Talk (DT), when near speech is occurring during playback, the ENR (Echo to Near Ratio) can be measured, i.e. the ratio of the power of the echo at the microphone relative to the power of the near speech. The ERLE will determine the ENR requirement for the residual echo to be at a level where the near speech is recognizable by a human or a computer.

Architecture

The block diagram is given below:



Demonstration features

- AK453 Codec driver
- Acoustic Echo Cancellation (AEC)
- Double Talk Detection (DTD)
- USB V1.1 Audio Device Library

Building the Application

This section details where to find the project configurations for building the application.

Description

To build this project, you must open the `usb_smart_speakerX` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is:

<install-dir>/apps/audio/usb_smart_speaker

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`usb_smart_speaker.X`	*<install-dir>/apps/audio/usb_smart_speaker/firmware*

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within *./firmware/src/system_config*.

Project Configuration Name	BSP Used	Description
pic32mz_ef_sk_meb2	pic32mz_ef_sk+meb2	This configuration runs on the PIC32MZ EF Starter Kit with the MEB II.

Configuring the Hardware

This section describes how to configure the supported hardware.

Description

PIC32MZ EF Starter Kit and MEB II

In this configuration, Jumper J9 must be set to EBIOE, otherwise the display will be blank white. Jumper J9 is located at back side of MEB II.

- Set the Jumper 9 to EBIOE (for internal SRAM)
- Connect PIC32EF Starter Kit
- Connect a microphone device to MIC IN jack of the AK4953 Codec on MEB II board
- Connect a amplified loudspeaker device to the HP OUT jack
- Plug in Power cord

Note that setting jumper 9 to EBIOE is to enable the external SRAM, which is used by graphics double buffering in this configuration. External SRAM is not mandatory for graphics double buffering. In this application, enabling external SRAM is used to save internal SRAM space.



Running the Demonstration

This section provides instructions on how to build and run the usb_smart_speaker demonstration.

Description

Important! Prior to using this demonstration, it is recommended to review the MPLAB Harmony Release Notes for any known issues. A PDF copy of the release notes is provided in the *<install-dir>/doc* folder of your installation.

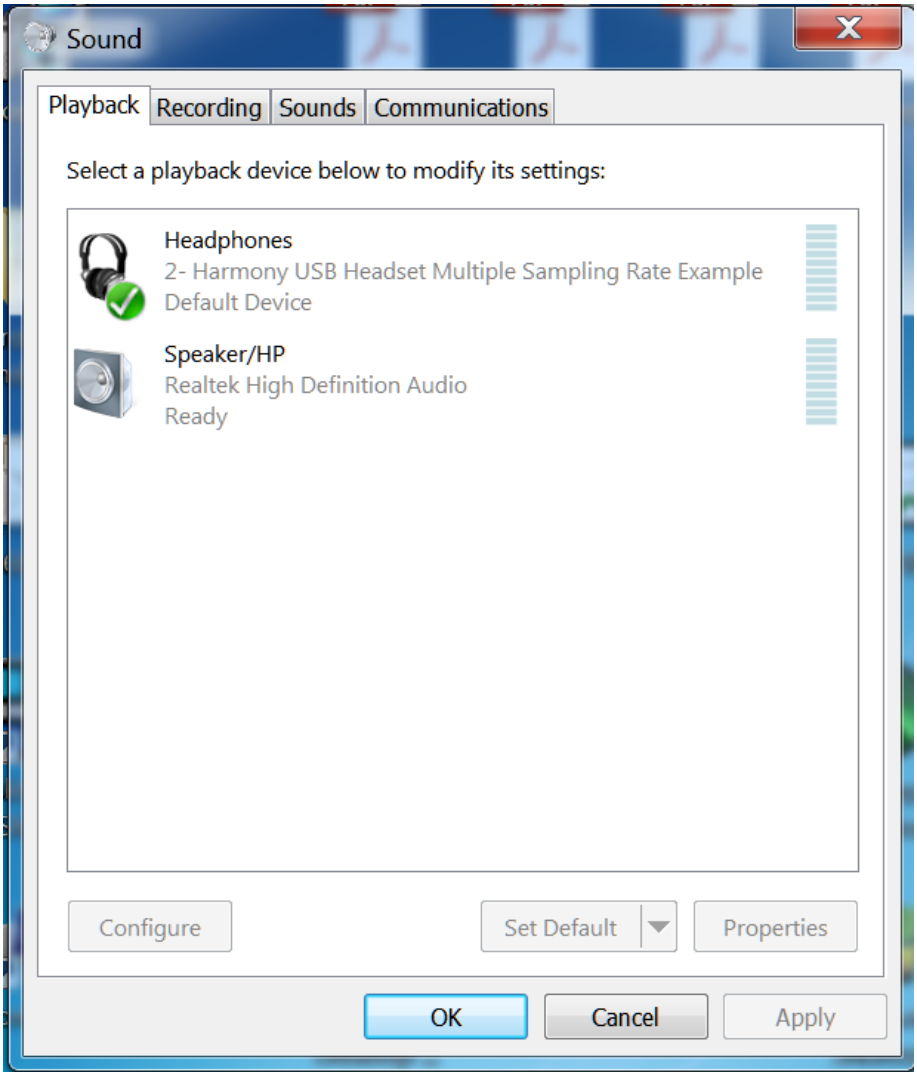
Compile and program the target device. While compiling, select the appropriate MPLAB X IDE project configuration based on the demonstration board. These configurations set the target processor, and the board and the Codec to be used in the output interface. Refer to Building the Application for details.

Do the following to run the demonstration:

1. Power up the board and connect the programmer. Compile the application, program the target device, and run it. After the program starts, the name of the application should appear on the display, as shown in the following figure.



2. The MEB II has an on-board AK4953 Codec. An external microphone connected to the "MIC IN" jack and loudspeaker, which is connected to the "HP OUT" jack is required, as shown. For demonstration purposes align the loudspeaker to the microphone so it can receive the echo to train the adaptive filter.
3. Connect the board using the USB mini-B connector (for the PIC32 Bluetooth Audio Development Kit or to the mini-micro connector on the starter kit) to the Host computer with a standard USB cable.
4. Allow the Host computer to acknowledge, install drivers (if needed), and enumerate the device. No special software is necessary on the Host side.
5. If needed, configure the Host computer to use the usb_microphone as the selected audio recording device and playback device. For Windows, this is done in the "Recording" and "Playback" tabs in the "Sound" dialog accessed by right clicking the loudspeaker icon in the taskbar, as shown in the following figure.

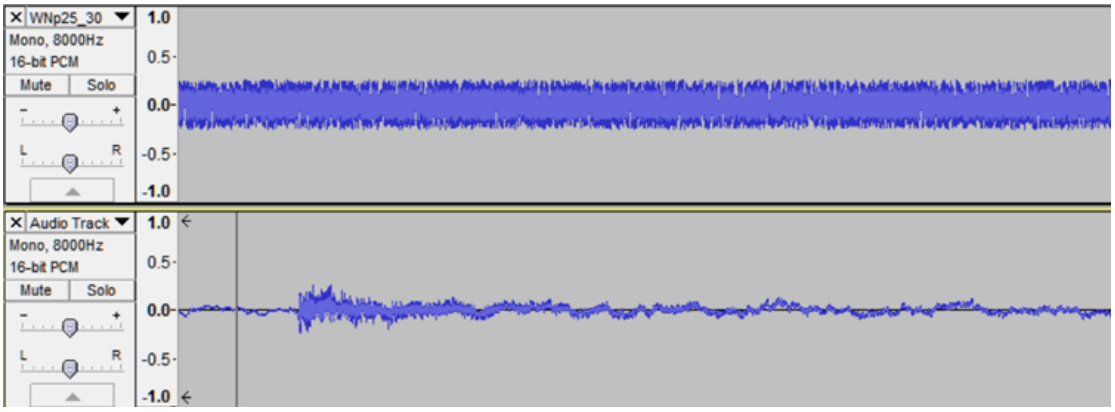


Note: The device *Harmony USB Smart Speaker Example* should be available along with a sound level meter indicating audio input when touching the microphone. If no sound level is registering, uninstall the driver and reboot the Host computer, since it may have incorrect configuration set by a similar connection to one of the other MPLAB-X Harmony Audio Demostrations.

Initial AEC Training (ST Training Test)

1. Audacity is used as the playback and record application on the PC. This is freely available and works well with this demonstration, although other playback and record applications will work also. Typically a smart speaker will initiate a training signal before general playback, such as the `WN_p25_30Sec.wav` white noise file available in the *usb_smart_speaker/Audio* folder. This can be played initially to pretrain the AEC. After that any audio can be played if the speaker is not moved, and nothing has changed the echo path. Using Audacity to train the AEC requires the following:
   - Audacity must be set up in overdub mode. Therefore, playback occurs at the same time as record.
   - The record button must be used (Playback can be used to listen to the recording after muting the training signal track in audacity). This will initiate playback at the same time as record when in overdub mode.

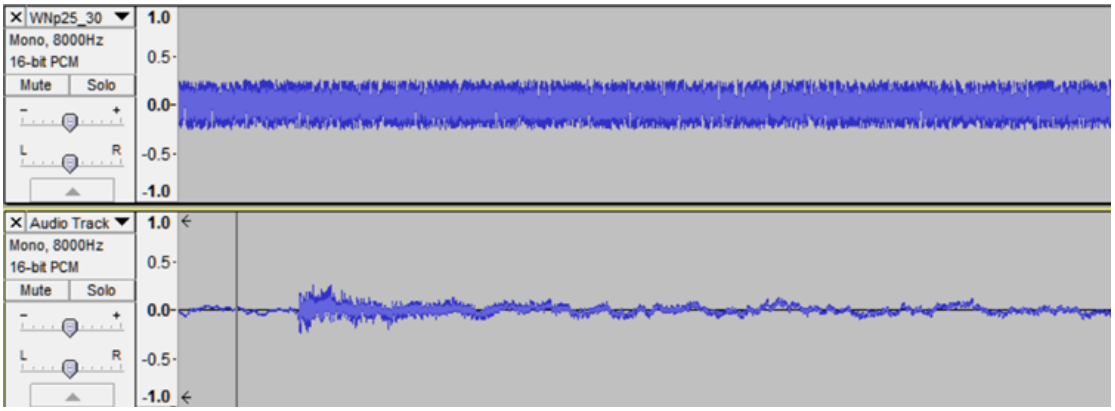
The initial training recording will show the echo gradually reduced to the background noise level, as shown below:



2. The Echo Return Loss (ERL) and Echo Return Loss Enhancement (ERLE) can be measured by making a recording without echo cancellation using the usb_headset demo with the same setup, and comparing the recording to the recording in step 1. ERL gives the power lost from the echo reference signal through the loudspeaker, to the signal received by the AEC from the microphone. The ERLE provides further echo power reduction through the AEC.

Echo Cancellation while Playing Music (DT Echo Cancellation Test)

1. The recording of near speech without echo is demonstrated by repeating the training setup with the following differences:
   - The playback file can be any audio file (preferably continuous audio)
   - After hitting the record button, speak into the microphone a test sequence. Typically this is a number sequence, or the ABC's, which will allow you to know what is supposed to be heard. However, any speech is allowed.
2. The recording should only have echo residual plus the near microphone speech, as shown in the following figure.



usb_speaker

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This demonstration application configures a USB Speaker device to run at a 48 kHz sampling rate at 16-bits per sample.

The system will interface to a USB Host (such as a personal computer), which can accommodate a USB device class speaker. The embedded system will enumerate with a USB audio device endpoint and enable the host system to input audio from the USB port using a standard USB Full-Speed implementation. The embedded system will stream USB playback audio to the Codec. The Codec Driver sets up the audio interface, timing, DMA channels and buffer queue to enable a continuous audio stream. The digital audio is transmitted to the Codec through a bidirectional I2S data module.

Architecture

This application runs on the PIC32 Bluetooth Audio Development Kit, which contains a PIC32MX470F512L microcontroller with 512 KB of Flash memory and 128 KB of RAM running at 96 MHz with the following features:

- 220x176 color LCD
- Five LEDs
- USB Device and Host interfaces
- Two X32 sockets, one of which will be used to host a Codec module daughter board that will be used for the USB playback demonstration through a 3.5 mm audio jack (labeled HP Out)

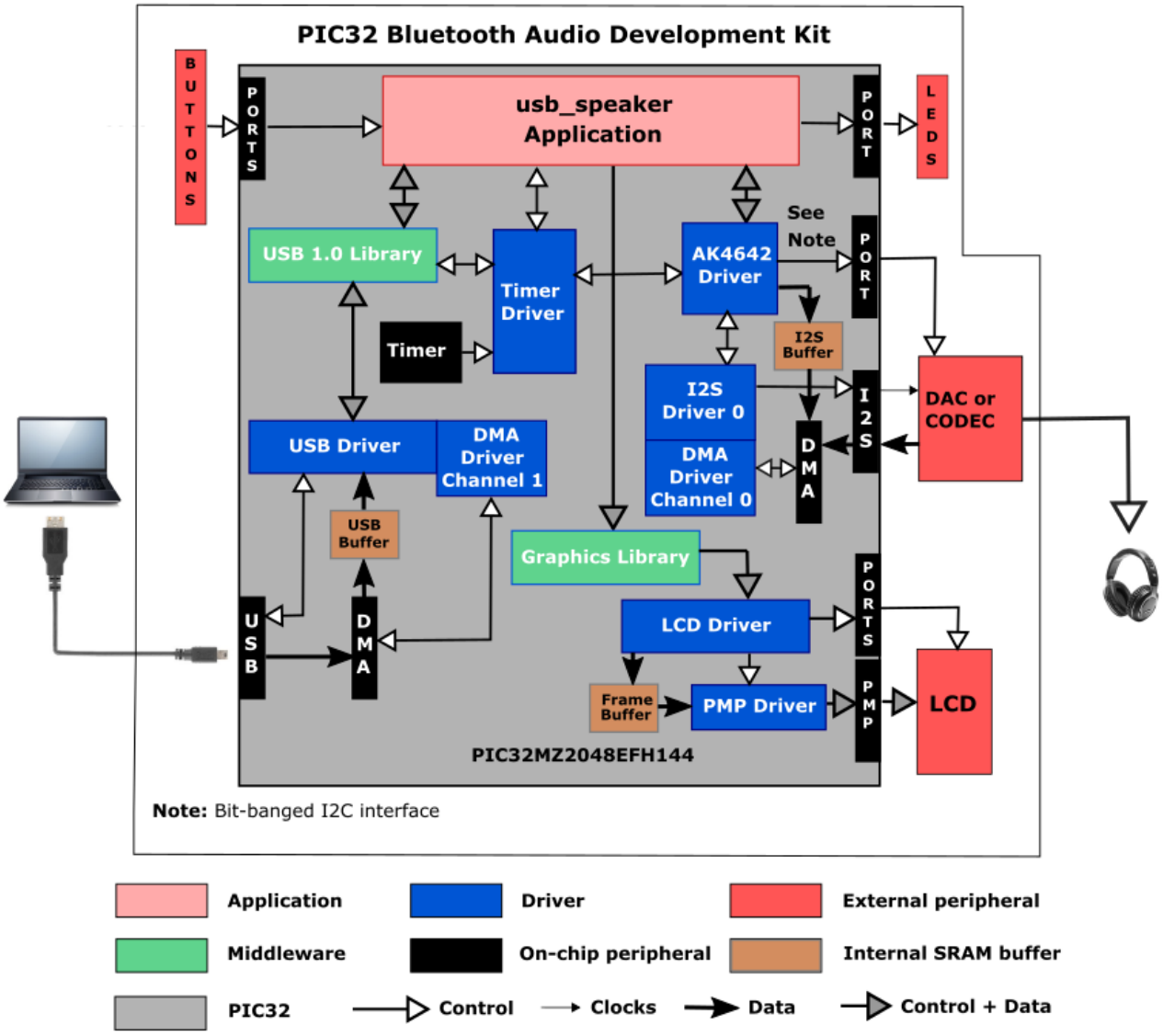
The PIC32 Bluetooth Audio Development Kit comes with a BTM805 Bluetooth Daughter Board and an AK4384 Audio DAC Daughter Board; you

must replace the AK4384 Daughter Board with the AK4642, AK4394, or AK7755 daughter board. The BTM805 daughter board is unused.

The board can also be configured for an external microphone through the 3.5 mm audio jack.

This application also runs on the PIC32MZ EF Starter Kit with the Multimedia Expansion Board II (MEB II). The starter kit contains a PIC32MZ2048EFH144 microcontroller with 2 MB of Flash memory and 144 KB of RAM running at 198 MHz. The starter kit interfaces to the MEB II, which supports a 480x272 pixel color LCD touch screen, USB device and USB host interfaces, and a AK4953 Codec that connects to headphones through a 3.5 mm audio jack (HP Out)

The following figure shows the application structure and hardware used. The PIC32 runs all the application code. The buttons are interfaced using GPIO pins.



This demonstration application configures the a USB Speaker device configured to run at a 48 kHz sampling rate at 16-bits per sample.

The system will interface to a USB Host (such as a personal computer), which can accommodate a USB device class speaker. The embedded system will enumerate with a USB audio device endpoint and enable the host system to input audio from the USB port using a standard USB Full-Speed implementation.

The embedded system will stream USB playback audio to the Codec.

The Codec Driver sets up the audio interface, timing, DMA channels and buffer queue to enable a continuous audio stream.

The digital audio is transmitted to the Codec through a bidirectional I2S data module.

48 kHz Sampling I2S Clock Values

I2S Function	Value	Description/PIC32 Pin (Port)
LRCK	48,000 Hz	Sample rate clock
BLCK	3,072,000 Hz	Bit rate clock

MCLK	12.288000	Master clock

PIC32MX470F512L I2S Data Pins

PIC32 Name	Description	I2S	PIC32 Bluetooth Audio Development Kit Pin	PIC32MZ EF Starter Kit/MEB II Pin
SDI1	Serial Data In	SDI1	73 (RC13)	69 (RD14)
SDO1	Serial Data Out	SDI1	72 (RD0)	49 (RB10)
SS1	Bit Clock	BCLK	69 (RD0)	58 (RF12)
SCK1	Master Clock	MCLK	70 (RD10)	109 (RD1)
REFCLKO1	Reference Clock	N/A	53 (RF8)	70 (RD15)

I2S Parameter Values (48 kHz Sampling)

I2S	Value	Description/PIC32 Pin (Port)
PIC32 Mode	HOST	
AK4642 Mode	SLAVE	Controlled by the PIC32
MCLK_MULT (mhc)	256	48 kHz sample rate multiplier for MCLK
MCLK_BCLK_RATIO	4	48 kHz sample rate multiplier for BCLK
Data Sizes	16/16/32	16-bit data/16-deep queue/32-bit frame
Channels	Stereo	2-channel
Timing	LJ	Left-justified

The I2S clock values generated by the PIC32 master are derived from the following formulas, starting with the REFCLK0 value (4.096 Mhz) used for the I2S MCLK and the sample rate, Fs (48000 Hz):

MCLK = 12,288,000 Hz

Fs = 48,000 Hz

NUM_CHANNELS = 2 channels

FRAME_SIZE = 32 bits/channel * NUM_CHANNELS = 64 bits/frame

LRCK = Fs

BCLK = FRAME_SIZE * Fs

The codec slave uses the following parameters calculated from the clock values:

MCLK_DIV = MCLK/LRCK

BCLK_DIV = MCLK/BCLK

The table shows the calculated parameters and generated clock rates.

The application uses the USB Library as a "Device" stack, which will connect to a "Host". It is configured for five USB endpoints with a single function having three interfaces. All of these are defined for a USB microphone device in the fullSpeedConfigurationDescriptor array variable structure (located in `system_init.c`). This structure defines the connection to the host at 48 kHz with 16-bit stereo channel data. A 64 packet queue is used for playback data, and a ping-pong buffer scheme is used for microphone record data. The maximum USB packets size is set to 48 * 2 channels/sample * 2 bytes/channel= 192 bytes, which gives a 1 ms stereo sample packet size at 48 kHz; therefore, the buffer size needs to be of the same size.

All MPLAB Harmony applications use the function SYS_Initialize located in the source file `system_init.c` and are executed from main to initialize various sub-systems such as the clock, ports, board support package (BSP), PMP, Codec, timers, UART, graphics, and interrupts. The USB, AK4642 driver, graphics, and the application state machines are all updated via calls located in the function SYS_Tasks, executed from the main polling loop, located in `system_tasks.c`.

The application code is contained in the standard source file `app.c`. The application code is initialized in SYS_Initialize using the application APP_Initialize function to instantiate driver objects and start the USB Device interface. The application utilizes a simple state machine (APP_Tasks executed from SYS_Tasks) with the following functions

- Sets up the drivers and the USB Library interface
- Responds to USB Host commands ("USB Record at 48 kHz/32 kHz/16 kHz", "USB Playback at 48 kHz/32 kHz/16 kHz")
- Initiates and maintains the bidirectional data audio streaming for "USB Record" and "USB Playback" functions.

Demonstration Features

- Playback using an AK4384 DAC daughter board on the PIC32 Bluetooth Audio Development Kit
- Playback using an AK4642 codec daughter board on the PIC32 Bluetooth Audio Development Kit
- Playback using an AK4954 codec daughter board on the PIC32 Bluetooth Audio Development Kit

- Playback using an AK7755 codec daughter board on the PIC32 Bluetooth Audio Development Kit
- USB connection to a host system using the USB Library Device Stack for a USB Speaker device using PIC32 Bluetooth Audio Development Kit, PIC32MZ EF STARTER KIT with MEB2, and BTSK
- Displaying graphics on the PIC32 Bluetooth Audio Development Kit 220x176 LCD using MPLAB Harmony Graphics Library Library (add link to MPLAB Harmony Graphics Composer Suite) and PMP (add link to Parallel Master Port (PMP) Driver Library) interface to OTM0221B graphics controller.
- Displaying graphics on the PIC32 MZ EF SK/MEB2 480X272 pixel touchscreen using MPLAB Harmony Graphics Library Library * with the LCC Interface to memory mapped frame buffer.

Tools Setup Differences

bt_audio_dk_ak4642 Configuration

If building this application from scratch, one would start by creating a 32-bit MPLAB Harmony project in MPLAB X (File -> New Project dialog). Select PIC32MX Bluetooth Audio Development Kit and a configuration name of bt_audio_dk_. Click on the green MPLAB Harmony icon to bring up the MPLAB Harmony Configurator (MHC). When using a PIM, select the board with the associated PIM, after selecting the device: PIC32MZ2048EFH144

Open the Harmony Configurator Framework -> Drivers section, then expand Codec -> Click on the box for the required Codec/DAC Driver?** and a submenu will appear. For this application the default values will suffice, although volume setting can be adjusted between a -75dB and +12dB gain range. The I2S and I2C driver (default to DRV_I2S_INSTANCE_0 and DRV_I2C_INSTANCE_0, respectively) associated to the Codec driver can be configured as described previously.

Note: The AK4953 driver is also used for the AK4954 Codec.

To use the Graphics Stack Library, under Graphics Stack, select **Use Graphics Stack?**. Further down, under *Use Harmony Graphics Composer Suite?* > *Middleware* > *Use Aria User Interface Library?*, clear **Enable Touch**. Since the board uses the Crystal Fontz 176x220 pixel LCD with the OTM2201A controller the PIC32 Bluetooth Audio Development Kit does not support a touchscreen. Under PMP, select **Use PMP Driver?**, which is used for graphics data transfer. Expand that section, and change the value of Strobe Wait Sates to PMP_STROBE_WAIT_4.

To use the USB Stack Library, under USB Library, select either **Use USB Stack Under Select Host** or **Device Stack Select Device**. Change the Number of Endpoints Used to 5. Below that select **USB Device Instance 0** and under that make Function 1 (matching the fullSpeedConfigurationDescriptor struct in system_init.c: AUDIO Device Class, 3 Interfaces, 2 Audio Streaming interface; with a Write Queue Size of 2 (not used) and a Read Queue Size of 64 (to mitigate any clock synchronization underflow/overflow audio artifacts). The Product ID Selection gives the name shown at the Host, which in this case is "usb_speaker_demo". A USB Interrupt Priority of 1 is also used.

Note: Do not change the current fullSpeedConfigurationDescriptor values in system_init.c when using MHC to generate the project files.

pic32mz_ef_sk_meb2 Configuration

If building this application from scratch, start by creating a 32-bit MPLAB Harmony project in MPLAB X IDE by selecting in *File* > *New Project*. Select the PIC32MZ EF Starter Kit plus MEB II and a configuration name of pic32mz_ef_sk_meb2. Click the green MPLAB Harmony icon to open the MPLAB Harmony Configurator (MHC).

Expand the *Harmony Configurator Framework* > *Drivers* section, and then expand *Codec* > *AK4953*. Select Use **AK4953 Driver?** and a sub-menu will appear. For this application the default values will suffice, except two clients are used (read and write), and the volume setting can be adjusted between a -75 dB and +12 dB gain range. The I2S and I2C driver (default to DRV_I2S_INSTANCE_0 and DRV_I2C_INSTANCE_0, respectively) associated to the Codec driver. I2S can be configured as described previously, using the 16 kHz values, although the application can modify these during run-time.

The bit-banged implementation and "Include Force Write ...)" is required for the PIC32MZ I2C module (I2C_ID_2). TMR1 is used for the bit-banged I2C signal timing over GPIO pins.

To use the Graphics Stack Library, under *Graphics Stack*, select **Use Graphics Stack?**. The default values given by the BSP can be used, which select the LCC graphics interface to a WQVGA screen and Touch Screen Controller with event processing. The only change is to use a V-Sync Refresh Strategy of "Conventional" rather than "Aggressive" to allow more processing time away from graphics processing and data bus transfer.

To use the USB Stack Library, under *USB Library*, select **Use USB Stack Under Select Host** or **Device Stack Select Device**. Change the Number of Endpoints Used to 2. Below that select **USB Device Instance 0** and under that select a Device Speed of USB_SPEED_FULL Select Function 1 and the AUDIO Device Class 3 Interfaces, 2 Audio Streaming interfaces; with a Write Queue Size of 8 (not used) buffers) and a Read Queue Size of 64 (to minimize "underflow/overflow" audio artifacts due to mismatched clocks between USB and the Codec). The Product ID Selection gives the name shown at the Host, which in this case is "usb_speaker_demo".

pic32mz_ef_pim_bt_audio_dk Configuration

PIC32MX270F512L on BTSK with AK4642/AK4755/AK4954 Configurations

These configurations are similar to that used for PIC32MX270F512L PIM on the PIC32 Bluetooth Audio Development Kit.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the demonstration.

Description

To build this project, you must open the `usb_speaker.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/audio/usb_speaker`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`usb_speaker.X`	`<install-dir>/apps/audio/usb_speaker/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
bt_audio_dk	bt_audio_dk	This configuration runs on the PIC32 Bluetooth Audio Development Kit using the Audio DAC Daughter Board included with the kit.
bt_audio_dk_ak4642	bt_audio_dk	This configuration runs on the PIC32 Bluetooth Audio Development Kit with the Audio Codec Daughter Board AK4642EN.
pic32mx270f512l_pim_bt_audio_dk	pic32mx270f512l_pim+bt_audio_dk	This configuration runs on the PIC32 Bluetooth Audio Development Kit configured with the PIC32MX270F512L PIM and the Audio DAC Daughter Board included with the kit.
pic32mz_ef_pim_bt_audio_dk	pic32mz_ef_pim+bt_audio_dk	This configuration runs on the PIC32 Bluetooth Audio Development Kit configured with the PIC32MZ2048EFH144 PIM and the Audio DAC Daughter Board included with the kit.
pic32mz_ef_sk_meb2	pic32mz_ef_sk+meb2	This configuration runs on the PIC32MZ EF Starter Kit with the MEB II.
bt_audio_dk_ak4954	bt_audio_dk+ak4954	This configuration runs on thePIC32 Bluetooth Audio Development Kit with the Audio Codec Daughter Board AK4954EN.
bt_audio_dk_ak7755	bt_audio_dk+ak7755	This configuration runs on the PIC32 Bluetooth Audio Development Kit with the Audio Codec Daughter Board AK7755EN.
pic32mx_bt_sk_ak4642	bt_sk+ak4642	This configuration runs on the PIC32 Bluetooth Starter Kit with the Audio Codec Daughter Board AK4642EN.
pic32mx_bt_sk_ak4954	bt_sk+ak4954	This configuration runs on the PIC32 Bluetooth Starter Kit with the Audio Codec Daughter Board AK4954EN.
pic32mx_bt_sk_ak7755	bt_sk_+ak7755	This configuration runs on the PIC32 Bluetooth Starter Kit with the Audio Codec Daughter Board AK7755EN.

Configuring the Hardware

This section describes how to configure the supported hardware.

Description

PIC32 Bluetooth Audio Development Kit and the Audio DAC Daughter Board (included in the kit)

Switch S1 on the PIC32 Bluetooth Audio Development Board should be set to PIC32_MCLR.

PIC32 Bluetooth Audio Development Kit and the Audio Codec Daughter Board AK4642EN (see **Note**)

Switch S1 on the PIC32 Bluetooth Audio Development Board should be set to PIC32_MCLR.

Note: The PIC32 Bluetooth Audio Development Kit includes an Audio DAC Daughter Board; however, the Audio DAC Daughter Board must be replaced by the Audio Codec Daughter Board AK4642.

PIC32 Bluetooth Audio Development Kit with PIC32MX270F512L PIM and the Audio Codec Daughter Board AK4384EN

Switch S1 on the PIC32 Bluetooth Audio Development Board should be set to PIC32_MCLR.

PIC32MZ EF Starter Kit and the Multimedia Expansion Board II (MEB II)

In this configuration, Jumper J9 must be set to EBIOE, otherwise the display will be blank white. Jumper J9 is located at back side of MEB II.

- Set the Jumper 9 to EBIOE
- Connect PIC32EF Starter Kit
- Connect a microphone device to MIC IN Jack of the AK4953 Codec on MEB II board
- Plug in Power cord.

Note that setting jumper 9 to EBIOE is to enable external SRAM, which is used by graphics double buffering in this configuration. In fact, external SRAM is not mandatory for graphics double buffering. In this application, enabling external SRAM is to save more internal SRAM space.

Running the Demonstration

This section demonstrates how to run the demonstration.

Description

Important! Prior to using this demonstration, it is recommended to review the MPLAB Harmony Release Notes for any known issues. A PDF copy of the release notes is provided in the `<install-dir>/doc` folder of your installation.

Compile and program the target device. While compiling, select the appropriate MPLAB X IDE project configuration based on the demonstration board. These configurations set the target processor, the board and the Codec to be used in the output interface. Refer to Building the Application for details.

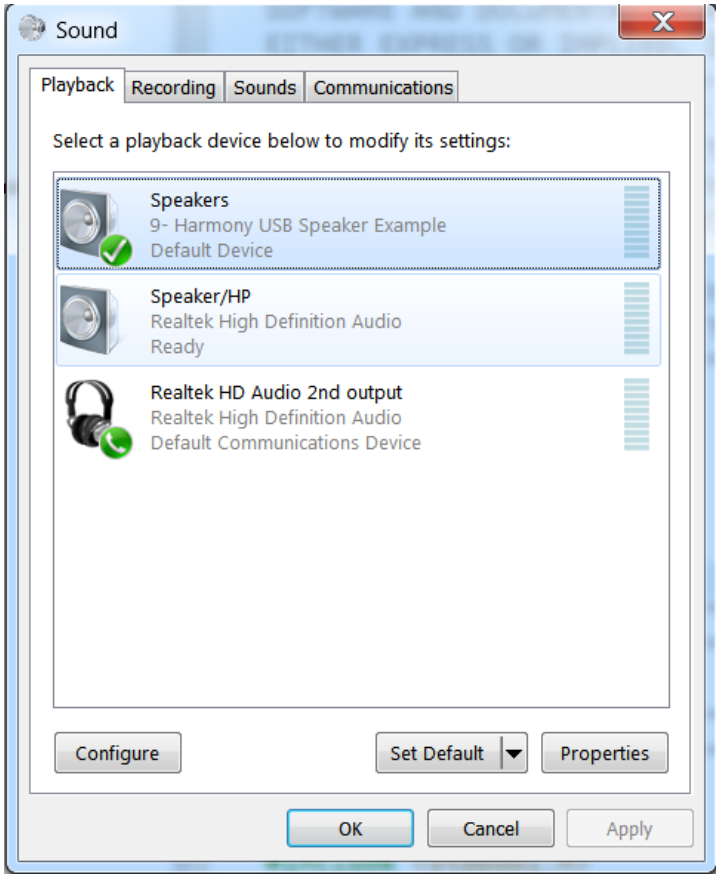
Do the following to run the demonstration:

1. Connect speakers or headphones to the headphone (HP) or line-out connector on the Audio DAC Daughter Board or the PIC32 Audio Codec Daughter Board AK4642EN, or the MEB II, as appropriate.





2. Connect power to the board.
3. Connect the board using the USB mini-B connector (Device) for the PIC32 Bluetooth Audio Development Board, or the USB micro-B connector for the MEB II to the Host computer with a standard USB cable.
4. Allow the Host computer to acknowledge, install drivers (if needed) and enumerate the device. No special software is necessary on the host side.
5. If needed, configure the Host computer to use the usb_speaker outputs as the selected audio device. This may be done in the system configuration or "Control Panel" depending on the operating system. For Windows, this is done in the Playback Devices dialog, which is accessed by right clicking the loudspeaker icon in the taskbar.
6. Play audio on the Host computer. This may be done with a standard media player or through a variety of sources including operating system generated sounds or video. You can observe that the USB connection through the Playback tab in the Sound Dialog.



7. Listen to the audio output on the speakers or headphones connected to the board. The volume will typically be adjusted by the host.

usb_speaker_hi_res

This section describes a USB Speaker that operates at 96 kHz sampling rate and 24-bit data. Information is provided on the supported demonstration boards, how to configure the hardware, and how to run the demonstration.

Description

This demonstration application configures a USB Speaker device to run at a 96 kHz sampling rate and 24-bits per stereo sample.

The system will attach to a USB host (such as a personal computer) using a USB Audio 1.0 Device interface, which can accommodate a USB Device class speaker. The embedded system will enumerate with a USB audio device endpoint, and enable the host system to input audio from the USB port using a standard USB full-speed implementation. The embedded system will stream USB playback audio to the codec over the I2S interface. The codec driver sets up this audio interface, timing, DMA channels and buffer queue, to enable a continuous audio stream. The digital audio is transmitted to the codec through a bidirectional I2S data module. The codec is configured through and I2C command interface.

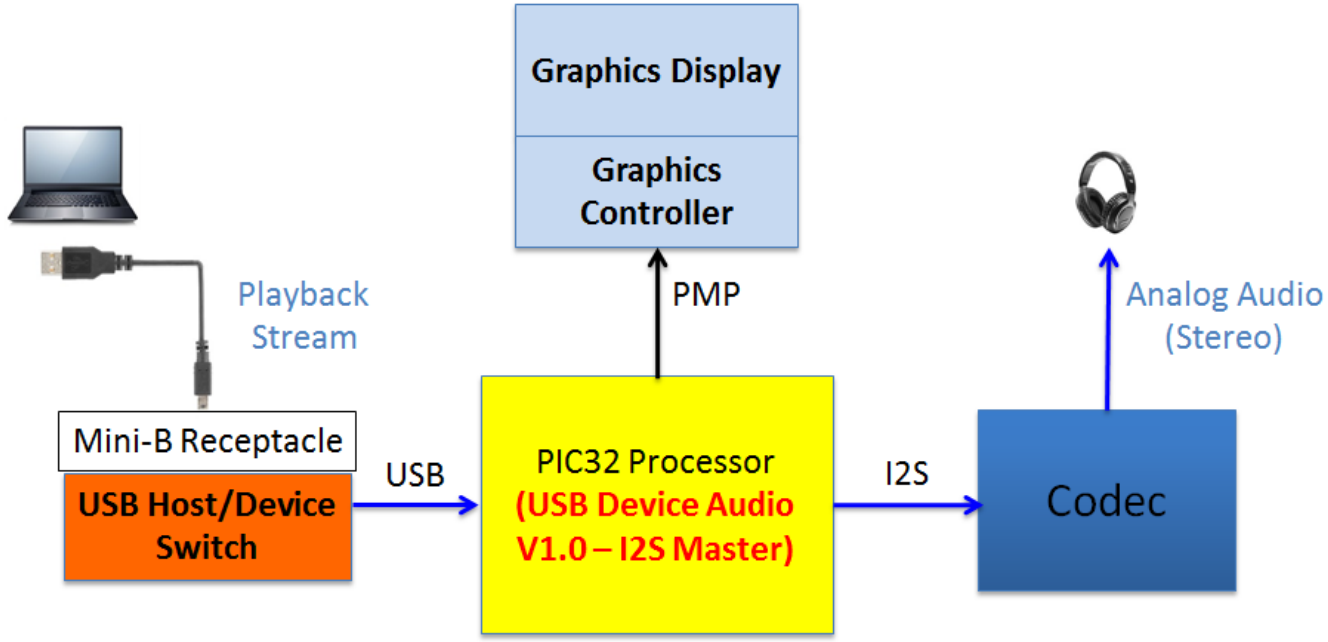
System Application Architecture

This application runs on the PIC32 Bluetooth Audio Development Kit, which contains a PIC32MX470F512L microcontroller with 512 KB of Flash memory and 128 KB of RAM running at 96 MHz. The demonstration uses the following features:

- 220x176 color LCD
- Five LEDs
- USB Device interfaces
- Two X32 sockets, one of which will be used to host a codec module daughter board that will be used for the USB playback demonstration through a 3.5 mm audio jack (labeled HP Out)

Note: The PIC32 Bluetooth Audio Development Kit comes with a BTM805 Bluetooth Daughter Board and an PIC32 Audio DAC Daughter Board - AK4384VT; however, the BTM805 Daughter Board is unused.

The following figure shows the applications system architecture.



The interface between the processor and the LCD display on the Bluetooth Audio Development Kit is an 8-bit Parallel Master Port (PMP) used for data transfer of graphics commands implemented by a graphics processor. The program uses the MPLAB Harmony Graphics Composer User's Guide to draw on the screen.

The codec utilizes one instance of I2S , with the I2S module and codec configured, as shown in the following tables.

The following table displays the I2S clock values generated by the PIC32 master. These values are derived from the following formulas, starting with the REFCLK0 value (12.288 MHz) used for the I2S MCLK, and the sample rate, Fs (48000 Hz), as given by:

```
MCLK = 12,288,000 Hz
Fs = 96,000 Hz
NUM_CHANNELS = 2 channels
FRAME_SIZE = 32 bits/channel * NUM_CHANNELS = 64 bits/frame
LRCK = Fs
BCLK = FRAME_SIZE * Fs
```

Table 1. 96 kHz Sampling I2S Clock Values

I2S Function	Value	Description/PIC32 Pin (Port)
LRCK	96.000000 kHz	Sample rate clock
BCLK	6.144000 kHz	Bit Rate Clock
MCLK	12.288000 kHz	Master Clock (REFCLK01)

The I2S is configured for bidirectional data flow, using the SDI1 pin for input (unused) and SDO1 for output to the headphone jack. The following table displays the pins used on the PIC32 Bluetooth Audio Development Kit for the I2S interface to the codec.

Table 2. I2S Data pins PIC32MX480F512L (Bluetooth Audio Development Kit)

PIC32 Name	Description	I2S	PIC32 Pin
SDI1	Serial Data In	SDI1	73(RC13)
SD01	Serial Data Out	SDI1	72(RD0)
SS1	Bit Clock	BCLK	69(RD9)
SCK1	Master Clock	MCLK	70(RD10)
REFCLK01	Reference Clock	-	53(RF8)

The following table displays the I2S set up for 24-bit data per stereo channel data transferred serially using 32-bit framing per channel (the least significant byte is 0). The data is configured as stereo (two channel). The 24-bit stereo samples are transferred to the codec memory buffer via DMA using 32-bit words. A 16-deep buffer queue is used to mitigate USB host, and USB device clock synchronization issues. The codec I2S slave is configured using the following parameters calculated from the clock values:

```
MCLK_DIV = MCLK/LRCK
```

```
BCLK_DIV = MCLK/BCLK
```

Table 3. I2S Parameter Values (48 kHz Sampling)

I2S	Value	Description/PIC32 Pin (Port)
PIC32 Mode	HOST	Description
AK4642 Mode	SLAVE	Controlled by the PIC32
MCLK_MULT (mhc)	128	Sample rate multiplier for MCLK
MCLK_BCLK_RATIO	2	Sample rate multiplier for BCLK
Data Sizes	32/32/32	32-bit data/32 FIFO/32 bits per channel
Channels	Stereo	2-channels
Timing	LJ	Left Justified

Software Architecture

All MPLAB Harmony applications use the function SYS_Initialize located in the source file system_init.c, which is part of the main function used to initialize various subsystems such as: the clock, ports, Board Support Package (BSP), PMP, codec, timers, graphics, and interrupts. The USB, AK4642 driver, graphics, and the application state machines are all updated via calls located in the function SYS_Tasks, executed from the main polling loop, located in system_tasks.c.

The application code is contained in the standard source file app.c. The application code is initialized within SYS_Initialize using the application APP_Initialize function to instantiate driver objects and start the USB device interface. The application utilizes a simple state machine (APP_Tasks executed from SYS_Tasks) with the following functions:

1. Set up the device peripheral drivers and USB Library interface.
2. Initiating read requests to the USB host over the USB device interface to start the playback audio stream.
3. Initiating the write data stream to the codec when the USB read data packet buffer has filled to a set level.
4. Execute responses to the USB device stream requests “read packet received” to maintain the stream and to playback the data through writes to the codec.
5. Respond to data buffer completes from the codec read and write buffer DMA complete interrupts, used to initiate USB device data reads to maintain the audio stream from the USB host.
6. To control the muting function of the USB device interface when the stream data is paused, or playback resumes of the USB interface, (i.e., change of the USB audio interface alternate settings).

The USB interface uses separate host and device data timing. The USB Device receiving the playback stream must mitigate buffer underflow and overflow problems to the codec, which is done by using a 64-deep packet buffer queue. Dropouts can occur if the receive rate is lower than the transmit rate to the codec, however this occurs rarely using a buffer of this length.

Demonstration Features

- High-speed (96 kHz) and high-definition (24-bit) USB audio connection to a host system using the USB V1.0 Library Device Stack for a USB speaker device using PIC32 Bluetooth Audio Development Kit.
- Playback of 96 kHz/24 bit audio using an AK4384 DAC daughter board on the PIC32 Bluetooth Audio Development Kit.
- Displaying graphics on the PIC32 Bluetooth Audio Development Kit 220x176 LCD using MPLAB Harmony Graphics Composer Suite and the Parallel Master Port Driver Library interface to OTM0221B graphics controller.

Tools Setup Differences

PIC32MX470F512L on the PIC32 Bluetooth Audio Development Kit with AK4384 Configuration

When building this application, start by creating a 32-bit MPLAB Harmony project in MPLAB X IDE by selecting *File > New Project*. Next, select **PIC32 Bluetooth Audio Development Kit** and a configuration name, such as bt_audio_dk indicating the default processor and codec daughter board for the PIC32 Bluetooth Audio Development Kit. Click the MPLAB Harmony icon to open the MPLAB Harmony Configurator (MHC).

Expand the *MPLAB Harmony Configurator Framework > Drivers* section, and then expand *Codec >.* Select the required Codec (AK4384) and a sub-menu will appear. For this application the default values will suffice, except for the following:

- The volume setting can be adjusted between a -75 dB and +12 dB gain range using the *range 0 to 255 setting*.

Expand I2S, the *Use I2S Driver?* option should already be selected. The driver callbacks require *Receive DMA Support* (for the read client). This automatically selects *Enable DMA Channel Interrupts* for the callback when buffer transfer has completed. Expand *I2S Driver Instance* and select *Audio Communication Width* corresponding to 32 Data/32 FIFO/32 per Channel. The *Audio Protocol Mode* is left justified. The *Queue Size Transmit* is set to 64. Note that the transmit queue is used for the codec read client of the AK4384. *Queue Size Receive* is used for the read client for other codecs.

The codec I2S driver is implemented using the PIC32 SPI peripheral block. The AK4384 Codec Driver also uses a bit-banged SPI control interface by default, which defaults to *Timer instance 0* when used with the *Prescale value of one* to control the SPI timing.

To use the *Graphics Stack Library*, select *Use Graphics Stack?.* Further down, within *Use Harmony Graphics Composer Suite? > Middleware > Use Aria User Interface Library?*, clear *Enable Touch*. Since the board uses the Crystal Fontz 176x220 pixel LCD with the OTM2201A controller, the PIC32 Bluetooth Audio Development Kit does not support a touchscreen. Within *PMP*, select *Use PMP Driver?,* which is used for graphics

data transfer. Expand that section, and change the value of *Strobe Wait States* to *PMP_STROBE_WAIT_4*.

Under the *USB Stack Library* select *Use USB Stack*, and then use the default selection, which is *Use Device*. Change the *Number of Endpoints Used* to "2". Below that select *USB Device Instance 0,* and then select *usb_speaker_demo* from *Product ID Selection*. This selection generates the fullSpeedConfigurationDescriptor struct in `system_init.c`: AUDIO Device Class, 3 Interfaces, 2 Audio Streaming interface; with a Write Queue Size of 2 (not used) and a Read Queue Size of 64 (the same as the codec write queue size). The *Product ID Selection* also gives the name shown at the host, which in this case is *usb_speaker_demo*. A *USB Interrupt Priority* of "1" is also used.

Note: Do not change the current fullSpeedConfigurationDescriptor values in `system_init.c` when using MHC to generate the project files, since it has been modified from 16-bit data and 48 kHz sampling rate to 96 kHz and 24-bit data, which also changes the packet size (1 ms audio packets are used for the full speed USB interface).

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the demonstration.

Description

To build this project, you must open the `usb_speaker_hi_res.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and the supported configurations. The parent folder for these files is `<install-dir>/apps/audio.`

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`usb_speaker_hi_res.X`	`<install-dir>/apps/audio/usb_speaker_hi_res/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
bt_audio_dk	bt_audio_dk	This demonstration runs on the PIC32MX470F512L mounted on the PIC32 Bluetooth Audio Development Kit with the PIC32 Audio DAC Daughter Board - AK4384VT.

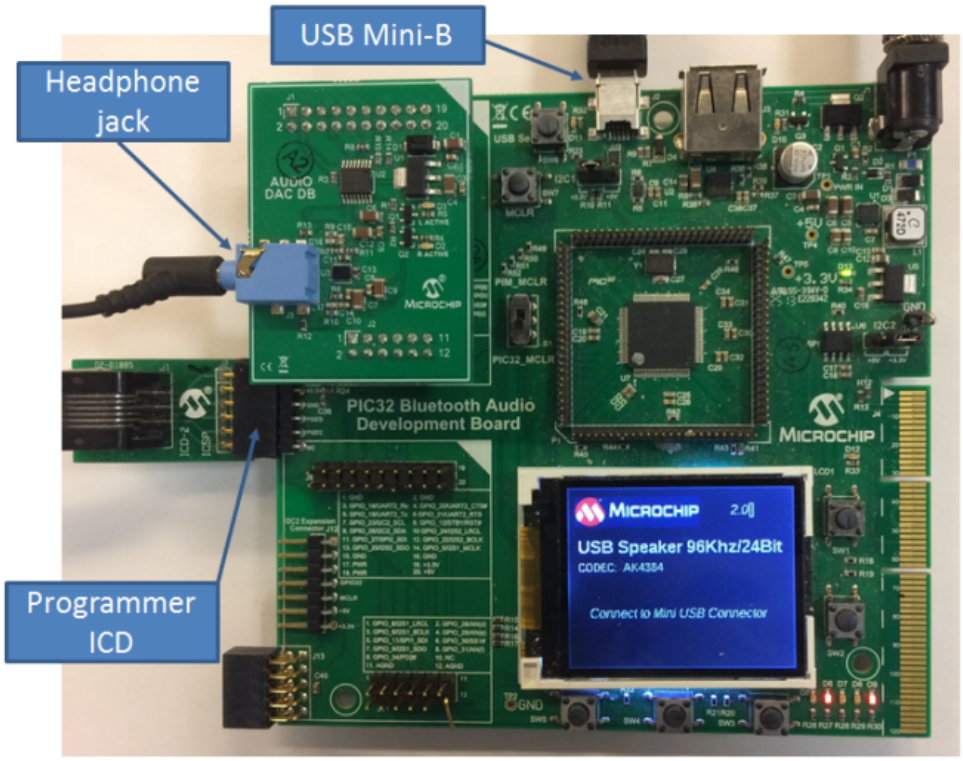
Configuring the Hardware

This section describes how to configure the supported hardware.

Description

PIC32 Bluetooth Audio Development Kit with the PIC32MX470F512L and the PIC32 Audio DAC Daughter Board - AK4384VT.

Connect switch S1 in the middle of the PIC32 Bluetooth Audio Development Board to PIC32_MCLR. Connect the AK4384 Daughter Board to the X32 Connector, as shown in the following figure.



Running the Application

This section demonstrates how to run the demonstration.

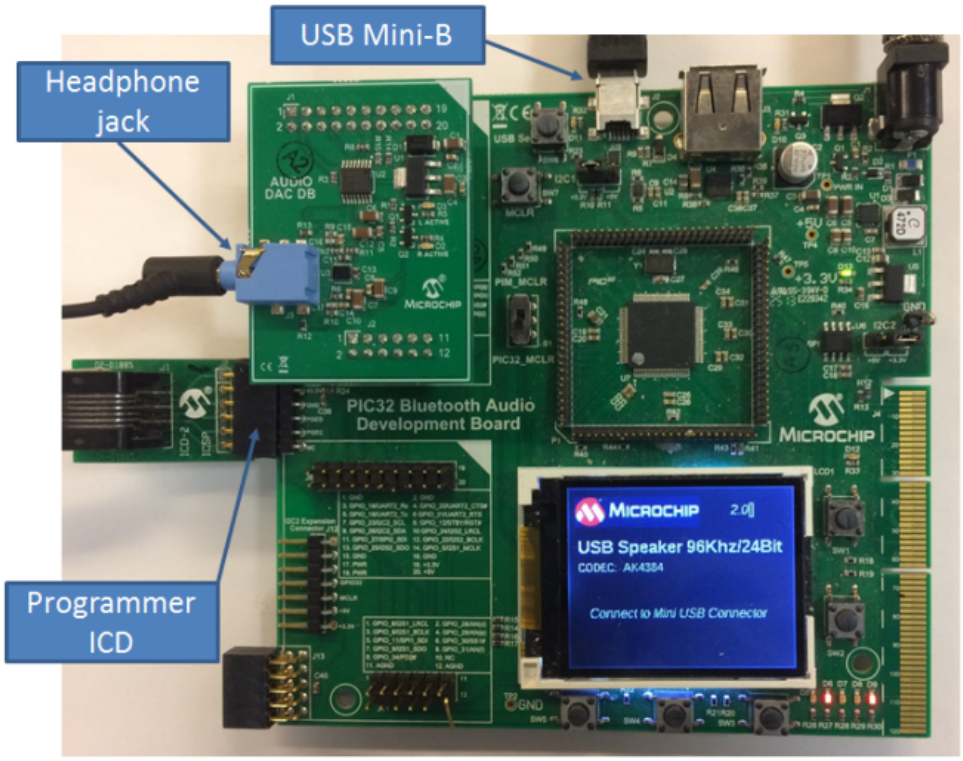
Description

Important!

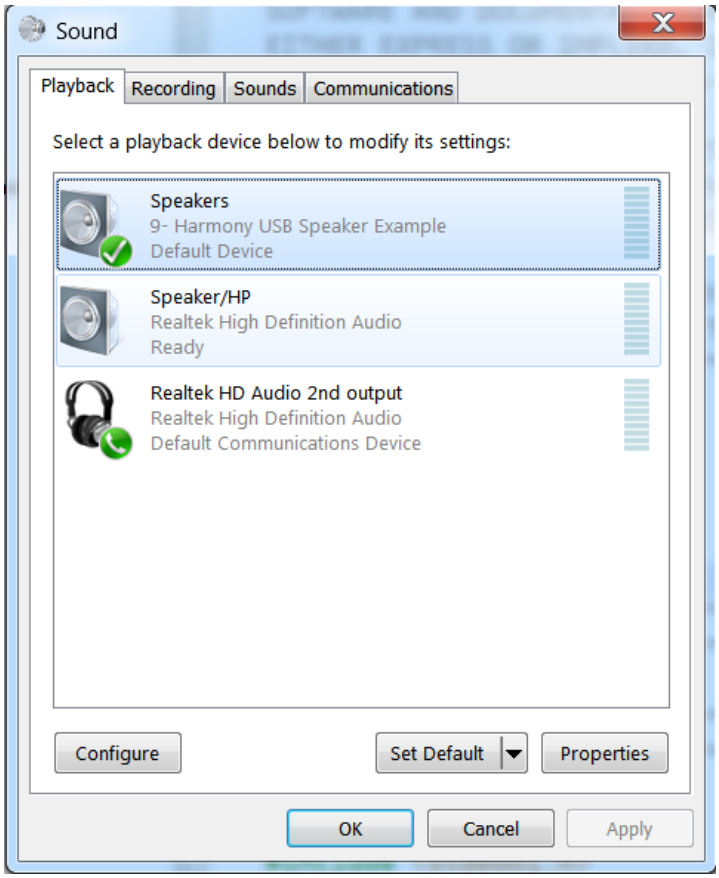
Prior to using this demonstration, it is recommended to review the MPLAB Harmony Release Notes for any known issues. A PDF copy of the release notes is provided in the `/doc` folder of your installation.

Do the following to run the demonstration:

1. Power up the board and connect the programmer. Compile the application, program the target device, and run it. After the program starts, the name of the application should appear on the display.
2. Connect speakers or headphones to the headphone (HP) connector of the PIC32 Audio DAC Daughter Board - AK4384VT, as shown in the following figure.
3. Connect the board using the USB mini-B connector (Device) for the PIC32 Bluetooth Audio Development Kit, as shown in the following figure.



4. Allow the host computer to acknowledge, install drivers (if needed), and enumerate the device. No special software is necessary on the host side.
5. If needed, configure the host computer to use the usb_speaker_hi_res outputs as the selected audio device. This may be done in the system configuration or Control Panel depending on the operating system. For Windows, this is done in the Playback Devices tab of the Sound dialog, which is accessed by right clicking the loudspeaker icon in the taskbar. Make the USB Speaker Example the default device, as shown in the following figure.



6. Play audio on the host computer. This may be done with a standard media player, or through a variety of sources including operating system generated sounds or video. You can observe the USB connection through the Playback tab in the Sound dialog.

7. Listen to the audio output on the speakers or headphones connected to the board. The volume will typically be adjusted by the host.

Bluetooth Demonstrations

This section provides descriptions of the PIC32 Bluetooth demonstrations.

MPLAB Harmony is available for download from the Microchip website by visiting: http://www.microchip.com/mplabharmony. Once you are on the site, click the Downloads tab to access the appropriate download for your operating system. For additional information on this demonstration, refer to the “Applications Help” section in the MPLAB Harmony Help.

Introduction

PIC32 Bluetooth Demonstration Applications Help.

Description

This help file contains instructions and associated information about MPLAB Harmony Bluetooth demonstration applications, which are contained in the MPLAB Harmony Library distribution.

There are two types of applications: those built using a Bluetooth Driver (currently there are only two, for the BM64), and those using the MPLAB Harmony PIC32 Bluetooth Stack Library (which is not written as a driver).

The Bluetooth Stack Library, which is considered the "basic" Bluetooth Stack , includes basic demonstrations that are referred to as Data Demonstrations. This section describes the hardware requirement and procedures to run these Basic Bluetooth Stack firmware projects on Microchip demonstration and development boards.

In addition to the Data Demonstrations provided with the PIC32 Bluetooth Basic Stack Library, additional Premium Demonstrations, which are available for purchase, demonstrate the capabilities of the Bluetooth Audio Stack Library. Information is provided in the Premium Demonstrations section on how to obtain, configure, and run these demonstrations.

To learn more about MPLAB Harmony stacks and libraries refer to the related documentation in Volume V: MPLAB Harmony Framework Reference.

Demonstrations

This section provides information on the standard Data Demonstrations and demonstration applications built with the BM64 Driver, as well as the purchased Premium Demonstrations.

Demonstration Functionality

Describes the functionality of the demonstrations.

Description

Basic Functionality

Bluetooth Module

The PIC32 Bluetooth Starter Kit and the PIC32 Bluetooth Audio Development Kit provide hardware support for the BlueCore® CSR8811™.

CSR8811

The CSR8811 is a single-chip radio and baseband IC for Bluetooth 2.4 GHz systems including Enhanced Data Rate (EDR) to 3 Mbps and Bluetooth low energy. The CSR8811 supports Bluetooth Class 1 transmission, and supports multiple device connection. The PIC32 Bluetooth Starter Kit and the PIC32 Bluetooth Audio Development Kit use a module based on the CSR8811 radio in its default configuration (see **Note**).

Note: The Flairmicro BTM805 module using the CSR8811 device is integrated in the PIC32 Bluetooth Starter Kit and is integrated in the BTM805 Bluetooth Daughter Board that is mounted on the PIC32 Bluetooth Audio Development Kit.

Bluetooth Device IDs

The Bluetooth software remembers and stores in Flash memory the last 10 unique Bluetooth device IDs to which it successfully paired to facilitate faster automatic reconnection when there is no currently active Bluetooth connection. If Bluetooth is turned OFF on a user smartphone that is currently connected and re-enabled later, it will automatically reconnect if in range or when it comes back into range.

Note: Currently, the demonstration does not have support for SPI Flash memory due to limitations in MPLAB Harmony, and therefore, the pairing information will not be stored or recovered on a power or hardware reset.

Bluetooth Device Address

When the development kit is powered on, it generates a random unique Bluetooth Device Address for any given development kit hardware. Optionally, at design time, the user can specify a Bluetooth Device Address in the application code of the development kit.

The device address is a six byte hexadecimal value. The macro, BT_DEVICE_ID, defines the first four bytes of the hexadecimal value and BT_DEVICE_ID_2LSB defines the last two bytes of the hexadecimal value. The last two bytes of the device address can be randomized by enabling BT_DEVICE_ID_2LSB_RANDOMIZE. These macros are defined in `btconfig.h`.

Setting a specific hard-coded device address is not recommended during the design and development state, as Bluetooth connection problems may be experienced if another development board with the same Bluetooth Device Address is within range.

Additional Bluetooth Resources

Provides information on additional Bluetooth demonstration resources.

Description

In addition to the demonstration capabilities described in the Demonstrations section, additional Bluetooth demonstration resources are available.

Other Android Applications

The Bluetooth data smartphone demonstration can also be executed using the Bluetooth SPP-pro application. This app can be downloaded by visiting:

https://play.google.com/store/apps/details?id=mobi.dzs.android.BLE_SPP_PRO&hl=en

Once installed, the commands can be sent and received in CMD line mode/Keyboard mode of the terminal emulator of the application. The commands will be sent and received in the format shown in the following table.

Windows Handset Applications

For a Windows mobile handset, the commands can be sent and received via data terminal. The application can be downloaded by visiting:

http://www.windowsphone.com/en-us/store/app/bt-terminal/09d679af-bdd8-40b2-b54e-56d68aeb03e0

Once installed, the commands can be sent and received from the terminal emulator of the application. The commands can be sent and received in the format shown in the following table.

Bluetooth Data Windows Personal Computer Demonstration Setup

Program the device with the hex file, `data_temp_sens_rgb.X.production.hex`.

1. On the Windows personal computer, select *Start> Control Panel > Hardware and Sounds > Add a device*. A list of available Bluetooth devices appears.
2. From the list, select **BTSK**.
3. Open a terminal emulator. For this example PuTTY was used. This Windows application can be obtained by visiting:

http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html

4. A connection is required to be established between PuTTY and the PIC32 Bluetooth Starter Kit development board. This can be done by selecting the "Serial" Connection type and the COM port assigned to the board in the PuTTY configuration window. The assigned COM ports can be identified from *Device Manager > Ports*.
5. Once connection is established, the following commands can be sent.

Command	Description
R	Programs LED for 100% of Red
G	Programs LED for 100% of Green
B	Programs LED for 100% of Blue
r	Programs LED for 50% of Red
g	Programs LED for 50% of Green
b	Programs LED for 50% of Blue

BM64 Driver Demonstrations

This section provides instructions about how to run the demonstration applications built using the BM64 Driver.

BM64_a2dp_hfp

This topic provides information on the supported demonstration boards, how to configure the hardware, and how to run the demonstration.

Description

In this demonstration application, the BM64 Module Daughter Board, installed on the PIC32 Bluetooth Audio Development Kit (BTADK), is used to stream A2DP audio from a Bluetooth host, such as a smartphone, to a pair of headphones connected to the Audio DAC Daughter Board which comes with the BTADK.

The demonstration can also automatically answer a voice call coming in via Hands-Free Protocol (HFP), interrupting (and pausing) any A2DP streaming in progress. Local audio is input using a microphone connected to the MIC IN input of the BM64 Module Daughter Board. When the call is terminated, streaming resumes.

Architecture

This application runs on the PIC32 Bluetooth Audio Development Kit, which contains a PIC32MX470F512L microcontroller with 512 KB of Flash memory and 128 KB of RAM running at 96 MHz with the following features:

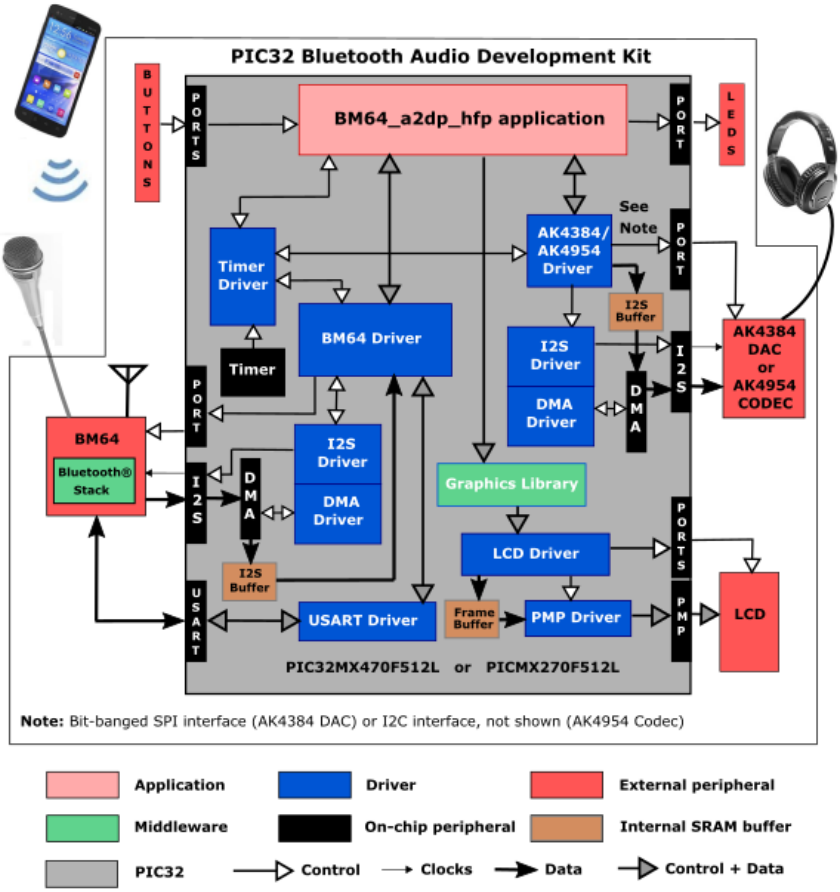
- 220x176 color LCD
- Six push buttons (SW1-SW6)
- Five LEDs
- USB Device and Host interfaces
- Two X32 sockets, one for Bluetooth module and a second for a codec or DAC

Notes:

1. The PIC32 Bluetooth Audio Development Kit comes with a BTM805 Bluetooth Daughter Board and an AK4384 Audio DAC daughter Board; however, you must replace the BTM805 daughter board with the daughter board for the BM64, and the AK4384 DAC with the AK4954A Codec Daughter Board, if applicable.
2. The USB Device and Host interfaces are not used in this application.

Surrounding the PIC32MX470F512L microcontroller are a set of headers, which accept various plug-in modules (PIM), including one for a PIC32MX270F512L PIM as a second configuration for this project.

The following figure illustrates the application architecture.



The PIC32 microcontroller (MCU) runs the application code, and communicates with the BM64 over a USART interface operating at 115,200 baud. The BM64 module contains its own Bluetooth stack, so it is unnecessary to include a software Bluetooth stack in the PIC32 MCU. The program takes up about 40% of the PIC32MX470F512 microcontroller's program space, and 27% of its RAM.

The interface between the PIC32 MCU and the LCD is an 8-bit parallel master port (PMP). The program uses the MPLAB Harmony v2.0 Graphics Library to draw on the screen. The buttons are also interfaced using GPIO pins.

As with any MPLAB Harmony application, the SYS_Initialize function, which is located in the `system_init.c` source file, makes calls to initialize various subsystems, such as the clock, ports, board support package (BSP), BM64, codec, PMP, timers, graphics, and interrupts.

The BM64 and codec drivers, graphics, timers, and the application state machines are all updated through calls located in the SYS_Tasks function in system_tasks.c file. Interrupt handlers in the `system_interrupt.c` file are used for the four DMA channels (only channels 0 and 1 are used), the USART, and the two timers.

The application code is contained in two source files, `app.c` and `audio.c`. The former contains the application state machine (APP_Tasks). It

first initializes the application, sets up a periodic timing callback, and then waits for the BM64 driver's internal state machine to complete initialization (indicated by the function DRV_BT_GetPowerStatus returning RV_BT_STATUS_READY). At that point the application calls a function called audioStart which initiates audio processing.

Meanwhile, the audio state machine in audio.c has received a handle to the BM64 driver by calling DRV_BT_Open, and then setting callbacks for two event handlers (AUDIO_BT_RxBufferEventHandler for buffer event handling, and _audioEventHandler for general event handling). After that it gets a handle to the codec driver by calling its DRV_CODEC_Open function with a mode of DRV_IO_INTENT_WRITE. Then it registers an event handler, AUDIO_CODEC_TxBufferEventHandler as a callback with the codec driver. Then, it waits for the application to call audioStart before it makes its first call to DRV_BT_BufferAddRead to request audio samples from the BM64.

The event handler callback is given control whenever a buffer has been processed. By default two sets of buffers are used (AUDIO_QUEUE_SIZE = 2), each with 1250 32-bit samples; however, simply by changing the queue size a circular buffer scheme can be used instead. As each buffer is handed off to thebe sent to the codec, the other one becomes available to be filled by the BM64.

The BM64 and codec drivers interface with their respective I2S interfaces via DMA, but this is largely transparent to the application, as it is all taken care of by MPLAB Harmony.

Demonstration Features

- Uses the BM64 Bluetooth Driver Library to read audio samples from the BM64
- Use of either a "ping-pong" or circular buffer scheme
- At a lower level, the BM64 driver uses the USART Driver Library to talk to the BM64 module
- Uses the AK4954 or AK4384 Codec Driver Library to write samples to the AK4384 DAC or AK4954 Codec
- At a lower level, using the I2S Driver Library between the BM64 and BM64 driver, and the AK4384 or AK4954 Codec Library and the AK4384 DAC or AK4954 Codec
- Use of "ping-pong" audio buffers
- Sending and receiving characters between the USART of the PIC32 MCU and the BM64 (see USART Driver Library)
- Displays graphics on the 220x176 LCD of the PIC32 Bluetooth Audio Development Kit using the MPLAB Harmony Graphics Library (see MPLAB Harmony Graphics Composer Suite) and the Parallel Master Port (see PMP Driver Library)
- Processing of button pushes on the PIC32 Bluetooth Audio Development Kit using a state machine for contact debouncing (see Ports Peripheral Library)
- Use of two timers: one as a periodic 1 ms timer for the application, and a second used by the BM64 driver (see Timer Driver Library)

Tools Setup Differences

When building a new application, start by creating a 32-bit MPLAB Harmony project in MPLAB X IDE by selecting *File* > *New Project*. Select the desired processor (PICMX470F512L or PIC32MX270F512L), and the PIC32MX Bluetooth Audio Development Kit. Click the MPLAB Harmony icon to start the MPLAB Harmony Configurator (MHC). Under BSP Configuration, select the PIC32 Bluetooth Audio Development Kit (BM64+AK4384), (BM64+AK4954), or PIC32MX270F512L w/ Bluetooth Audio Development Kit (BM64+AK4384) as appropriate for the hardware.

Open the *Harmony Configurator Framework* > *Drivers* section, and then expand *Bluetooth* > *BM64* and select *Use BM64 Driver?*. A sub-menu will appear with all options selected. Since you will only be using HFP, A2DP, and AVRCP protocols, clear the BLE Features? check box. Leaving the HFP, A2DP, AVRCP Protocols selected will by default bring in the drivers for both the codec and the I2S (and I2C driver, if using the AK4954).

If using the AK4384 DAC, expand *Drivers >CODEC* > *Use Codec AK4384?*, which should already be selected. Select **Specify MCLK value**, and enter 256. Expand *Use Bit Banged SPI Control Interface* and also *Code AK4384 Driver Instance 0*. Change the Timer driver (used for bit banging) instance to 1. If using the AK4954 Codec instead, expand *Drivers >CODEC* > *Use Codec AK4954?*, which should already be selected. Select **Specify MCLK value**, and enter 256. Then expand *Drivers* > *Use I2C*, which should also already be selected, and check the box Include Force Write I2C Function.

Under I2S, Use I2S Driver should already be selected. Under *Sampling Rate*, enter 44100. Expand *I2S Driver Instance 1*, and change Transmit DMA Channel to 1, and Receive DMA Channel to 3.

You will also want to go into the Timer section, select a second timer (change Number of Timer Instances from 1 to 2), and then within TMR Driver Instance 1, change the Timer Module ID to TMR_ID_2, and the Prescale value to TMR_PRESCALE_VALUE_1.

In the USART section, Use USART Driver? should already be selected. The default parameters are okay as is.

Under *System Services >DMA*, Use DMA System Service should already be selected. Change the Number of DMA Channel Instances to 4 and press **Enter**. Under DMA Channel Instance 0, change the Interrupt Priority to INT_PRIORITY_LEVEL2. Under DMA Channel Instance 1, change the Interrupt Priority to INT_PRIORITY_LEVEL4. Under DMA Channel Instances 2 and 3, change the Interrupt Priority to INT_DISABLE_INTERRUPT.

If your application is going to be using graphics, as this one does, within Graphics Stack, select **Use Graphics Stack?**. Further down, within *Use Harmony Graphics Composer Suite?* > *Middleware* > *Use Aria User Interface Library?*, clear **Enable Touch** since the LCD on the PIC32 Bluetooth Audio Development Kit does not support a touchscreen. Since you are using the LCD, you will also need to go back to the Drivers section, and within PMP, select **Use PMP Driver?**. Expand that section, and change the value of Strobe Wait Sates to PMP_STROBE_WAIT_4.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Bluetooth Demonstration.

Description

To build this project, you must open the `a2dp_avrcp_hfp.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/bluetooth/audio/BM64_a2dp_hfp`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`a2dp_avrcp_hfp.X`	`<install-dir>/apps/bluetooth/audio/BM64_a2dp_hfp/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
bt_audio_dk	bt_audio_dk+bm64	This demonstration runs on the PIC32MX470F512L mounted on the PIC32 Bluetooth Audio Development Kit and the BM64 Bluetooth Module Daughter Board.
pic32mx270f512l_pim_bt_audio_dk	pic32mx270f512l_pim+bt_audio_dk+bm64	This demonstration runs on the PIC32MX270F512L PIM mounted on the PIC32 Bluetooth Audio Development Kit and the BM64 Bluetooth Module Daughter Board.
bt_audio_dk_ak4954	bt_audio_dk+bm64+ak4954	This demonstration runs on the PIC32MX470F512L mounted on the PIC32 Bluetooth Audio Development Kit and the BM64 Bluetooth Module Daughter Board plus AK4954 Codec Daughter Board.

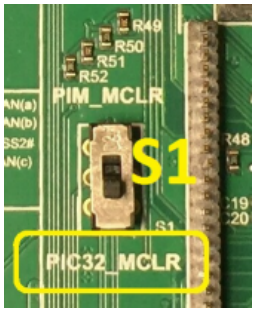
Configuring the Hardware

This section describes how to configure the supported hardware.

Description

PIC32 Bluetooth Audio Development Kit with the on-board PIC32MX470F512L device and the BM64 Bluetooth Module Daughter Board (see **Note**), using either the AK4384 Audio DAC Daughter Board, or the AK4954A Codec Daughter Board.

Switch S1 on the PIC32 Bluetooth Audio Development Board should be set to PIC32_MCLR.

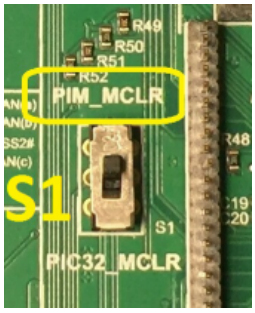


PIC32 Bluetooth Audio Development Kit with the PIC32MX270F512L Plug-in Module (PIM) and the BM64 Bluetooth Module Daughter Board (see **Note**).

Switch S1 on the PIC32 Bluetooth Audio Development Board should be set to PIM_MCLR.

PIC32 Bluetooth Audio Development Kit with the on-board PIC32MX470F512L device and the BM64 Bluetooth Module Daughter Board (see **Note**), plus AK4954 Codec Daughter Board.

Switch S1 on the PIC32 Bluetooth Audio Development Board should be set to PIC32_MCLR.



Note: The PIC32 Bluetooth Audio Development Kit includes a BTM805 Bluetooth Daughter Board; however, it must be replaced by the BM64 Bluetooth Module Daughter Board (AC320032-3). If using the AK4954 codec, the Audio DAC Daughter board must be replaced by the PIC32 Audio Codec Daughter Board AK4954A, which is sold separately on microchipDIRECT as part number AC324954.

Running the Demonstration

This section describes how to run the demonstration.

Description

Do the following to run the demonstration:

1. Using the BM64_bootloader application, if you have not done so, configure the BM64 as an I2S slave device and give it a unique device name.
2. Connect a pair of headphones to the headphone out connector on the Audio DAC Daughter Board which comes by default with the PIC32 Bluetooth Audio Development Kit.
3. Compile the application, program the target device, and run it. While compiling, select the appropriate MPLAB X IDE project configuration based on whether you are using the PIM, and AK4954 Codec instead of the AK4384 DAC. Refer to Building the Application for details. After the program starts, the device name and address should appear on the display and the connection icon in the top-right corner will be gray.



4. Press and hold the SW1 button on the Bluetooth Audio Development Kit to put the BM64 Module into pairing mode, indicated by the LEDs on the BM64 Module flashing alternately.
5. In the Bluetooth Settings of your smartphone, find the name of the BM64 which you set up in Step 1, then select and pair with it. The two LEDs on the BM64 Module should stop flashing, and instead just one should flash twice every few seconds. Meanwhile the connection icon in the top right of the LCD should have changed from gray to bright blue.
6. Go into an application on the smartphone that outputs audio, such as Music. Select a song and press play. A Play icon should appear on the LCD screen of the Bluetooth Audio Development Kit as shown below and you should hear audio from the headphones.



7. Use SW1 and SW2 on the Bluetooth Audio Development Kit to increase and decrease the volume, SW4 to alternately pause and resume the playback, SW3 and SW5 to go to the next or previous song (or use a long press of SW3/SW5 for fast forward or rewind). Set Table 1 below.
8. Use a long press of SW2 to disconnect from the smartphone, and a another long press of SW2 to reconnect. Use a long press of Sw4 to forget all previous pairings.
9. Call the smartphone from another phone such as a wired desk phone. When the smartphone is answered, either automatically or manually, the

audio should switch over to the BM64. Talking into the handset of the desk phone should come out of the headphones, and talking into a microphone connected to the BM64 Module should come out of the desk phone handset. When the call is terminated, any A2DP streaming that was in progress should resume.

Control Descriptions

Control	Description
SW1	Volume Up (hold > 1 second for Begin Pairing function).
SW2	Volume Down (hold > 1 second for Disconnect or Link Last function).
SW3	Play Next Song (hold > 1 second for Fast Forward).
SW4	Play Next Song (hold > 1 second for Fast Forward).
SW5	Play Previous Song (hold > 1 second for Rewind).

BM64_ble_comm

This topic provides information on the supported demonstration boards, how to configure the hardware, and how to run the demonstration.

Description

In this demonstration application, the Bluetooth® Low Energy (BLE) capabilities of the BM64 Module Daughter Board, installed on the PIC32 Bluetooth Audio Development Kit, are used to send a string of characters to a smartphone when one of the push buttons is pressed on the BTADK, and to receive a string of characters from the smartphone and display them on the LCD of the PIC32 Bluetooth Audio Development Kit.

Although the current BM64 module does not support standard BLE GATT predefined profiles, it does provide what is called a Transparent Service, which allows generic text strings up to 20 bytes in length to be sent back and forth between a server (e.g., BM64) and a client (e.g., smartphone). Therefore, users can add their own structure on top of this capability and implement their own protocol such as a command/data format.

The Transparent Service provides a BLE substitute for the Serial Port Protocol (SPP) of classic Bluetooth for use with an Apple® iPhone®, which *does not* support SPP.

To successfully run this application, you will need an iPhone 4s or later, which supports Bluetooth 4.0.

Note: This MPLAB Harmony application has been designed to also work with a smartphones running Android™, as well as an iPhone; however, the firmware on the BM64 module currently does not support connections to Android smartphones. If you need this application to work with an Android™ phone, please contact your local Microchip Marketing Representative as to the possibility of getting an update to the BM64 firmware.

Architecture

This application runs on the PIC32 Bluetooth Audio Development Kit, which contains a PIC32MX470F512L microcontroller with 512 KB of Flash memory and 128 KB of RAM running at 96 MHz with the following features:

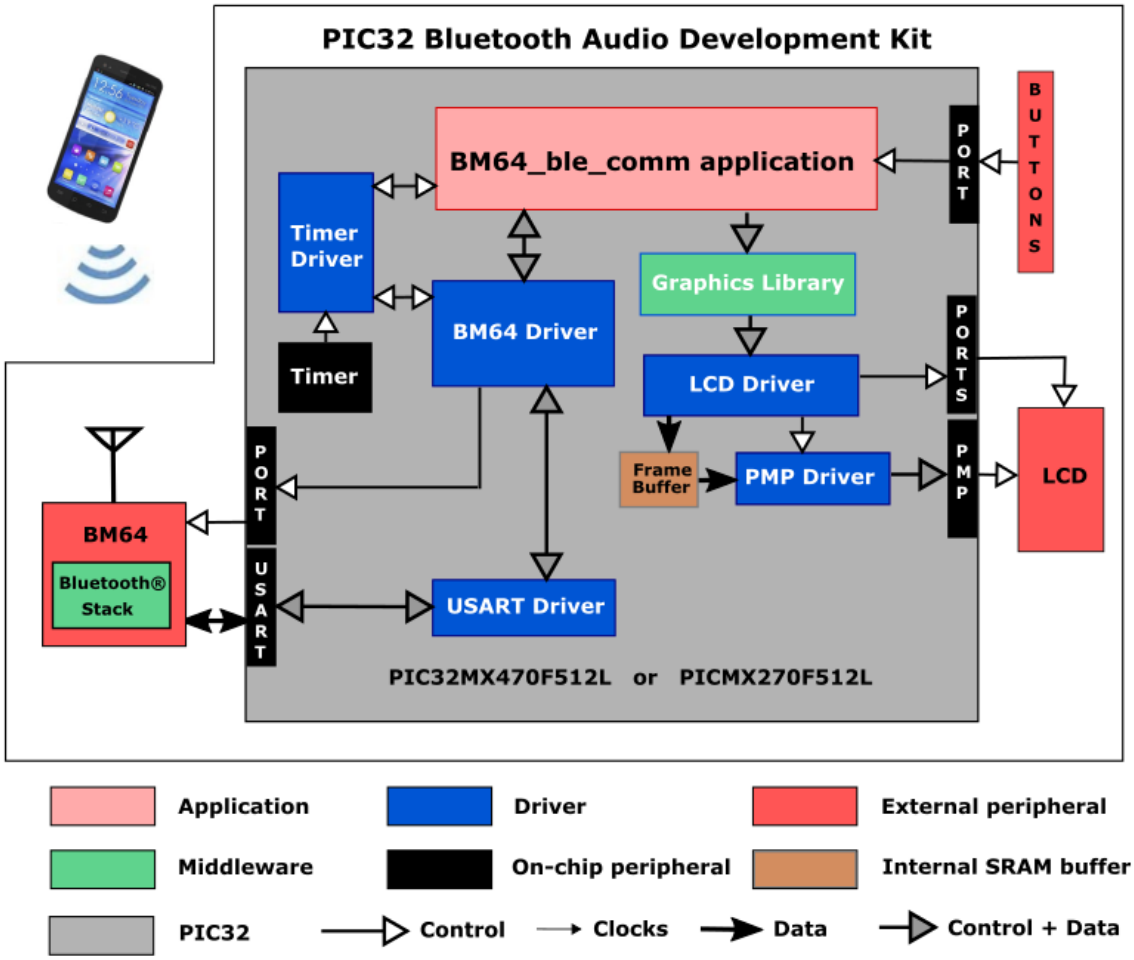
- 220x176 color LCD
- Six push buttons (SW1-SW6)
- Five LEDs
- USB Device and Host interfaces
- Two X32 sockets, one one for a Bluetooth module and a second for a Codec or DAC

Notes:

1. The PIC32 Bluetooth Audio Development Kit comes with a BTM805 Bluetooth Daughter Board and an AK4384 Audio DAC daughter Board; however, you must replace the BTM805 daughter board with the daughter board for the BM64.
2. The USB Device and Host interfaces, as well as the LEDs, and the AK4384 Audio DAC Daughter Board *are not used* in this application.

Surrounding the PIC32MX470F512L microcontroller are a set of headers, which accept various plug-in modules (PIM), including one for the PIC32MX270F512L as a second configuration for this project.

The following figure illustrates the application architecture.



The PIC32 microcontroller (MCU) runs the application code, and communicates with the BM64 over a USART interface operating at 115,200 baud. The BM64 module contains its own Bluetooth stack, so it is unnecessary to include a software Bluetooth stack in the PIC32 MCU. The program takes up only about 35% of the PIC32MX470F512 microcontroller's program space, and 25% of its RAM.

The interface between the PIC32 MCU and the LCD is an 8-bit parallel master port (PMP). The program uses the MPLAB Harmony v2.0 Graphics Library to draw on the screen. The buttons are also interfaced using GPIO pins.

As with any MPLAB Harmony application, the SYS_Initialize function, which is located in the `system_init.c` source file, makes calls to initialize various subsystems, such as the clock, ports, board support package (BSP), PMP, BM64, timers, UART, graphics, and interrupts.

The BM64 driver, graphics, and the application state machines are all updated through calls located in the SYS_Tasks function in `system_tasks.c` file. Interrupt handlers in the `system_interrupt.c` file are only used for receiving characters from the UART and for the two timers.

The application code is contained in two principal source files. The first, `app.c`, contains a simple state machine (APP_Tasks), which initializes the application, and then while idle, calls a secondary state machine, buttonTasks, to process any button pushes (resulting in a call to the BM64 driver to send a string from the BM64 to a connected smartphone).

APP_Tasks also calls another secondary state machine (bleTasks), which is located in the source file `ble.c` to process BLE-related tasks. bleTasks receives a handle to the BM64 driver by calling DRV_BT_Open, and then registers an event handler, _BLEEventHandler, as a callback with the BM64 driver. The callback is called when either a message is received from the BM64 (which is then put on the display) or when the BLE status changes (i.e., when it is connected or disconnected). Both of these actions result in updates to the LCD display through calls to the graphics sub-system.

Demonstration Features

- BM64 Bluetooth Driver Library
- At a lower level, the BM64 Driver uses the USART Driver Library to communicate with the BM64 module
- BLE advertising
- Connection of BM64 to the smartphone
- Display of service and characteristic UUIDs on the smartphone
- Sending of a text string from the BTADK/BM64 to the smartphone when a button is pressed
- Sending of a text string from the smartphone to the BM64/BTADK, which displays it on the LCD
- Sending and receiving characters between the USART of the PIC32 MCU and the BM64. (see USART Driver Library)
- Displaying graphics on the 220x176 LCD of the PIC32 Bluetooth Audio Development Kit using the MPLAB Harmony Graphics Library (see

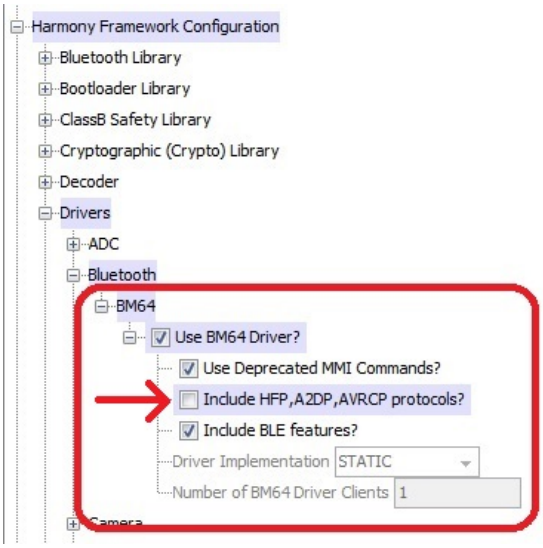
MPLAB Harmony Graphics Composer Suite) and the Parallel Master Port (PMP) Driver Library.)

- Processing of button pushes on the PIC32 Bluetooth Audio Development Kit using a state machine for contact debouncing. (see Ports Peripheral Library)
- Use of two timers: one as a periodic 1 ms timer for the application, and a second used by the BM64 driver (see Timer Driver Library)

Tools Setup Differences

When building a new application, start by creating a 32-bit MPLAB Harmony project in MPLAB X IDE by selecting *File* > *New Project*. Select the desired processor (PICMX470F512L or PIC32MX270F512L) and the PIC32MX Bluetooth Audio Development Kit. Click the green MPLAB Harmony icon to start the MPLAB Harmony Configurator (MHC). Under BSP Configuration, select PIC32 Bluetooth Audio Development Kit (BM64+AK4384) or PICMX270F512L w/ Bluetooth Audio Development Kit (BM64+AK4384).

Open the *Harmony Configurator Framework* > *Drivers* section, and then expand *Bluetooth* > *BM64* and select **Use BM64 Driver?**. A submenu will appear with all options selected. Since you will only be using BLE functions, clear **HFP,A2DP,AVRCP protocols?**. If this option is selected, it would also by default, bring in the drivers for the AK4384 codec and I2S.



You will also want to go into the Timer section, select a second timer (change Number of Timer Instances from 1 to 2), and then within TMR Driver Instance 1, change the Prescale value to TMR_PRESCALE_VALUE_1.

In addition, go into the USART section and select **Use USART Driver?** and use the default parameters.

If your application is going to be using graphics, as this one does, within Graphics Stack, select **Use Graphics Stack?**. Further down, within *Use Harmony Graphics Composer Suite?* > *Middleware* > *Use Aria User Interface Library?*, clear **Enable Touch** since the LCD on the PIC32 Bluetooth Audio Development Kit does not support a touchscreen. Since you are using the LCD, you will also need to go back to the Drivers section, and within PMP, select **Use PMP Driver?**. Expand that section, and change the value of Strobe Wait Sates to PMP_STROBE_WAIT_4.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the demonstration.

Description

To build this project, you must open the `BM64_ble_comm.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/bluetooth/audio/BM64_ble_comm`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`BM64_ble_comm.X`	`<install-dir>/apps/bluetooth/utilities/BM64_ble_comm/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

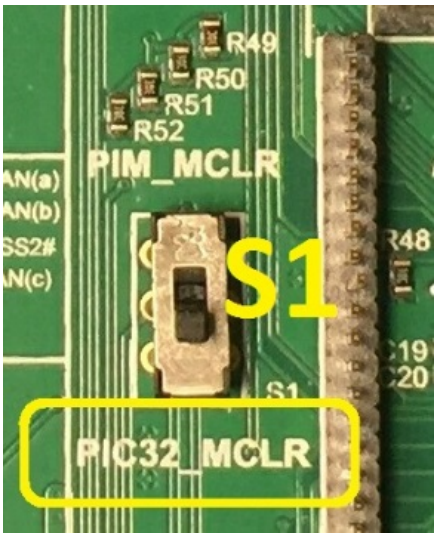
Project Configuration Name	BSP Used	Description
bt_audio_dk	bt_audio_dk+bm64	This demonstration runs on the PIC32MX470F512L mounted on the PIC32 Bluetooth Audio Development Kit and the BM64 Bluetooth Module Daughter Board.
pic32mx270f512l_pim_bt_audio_dk	pic32mx270f512l_pim+bt_audio_dk+bm64	This demonstration runs on the PIC32MX270F512L PIM mounted on the PIC32 Bluetooth Audio Development Kit and the BM64 Bluetooth Module Daughter Board.

Configuring the Hardware

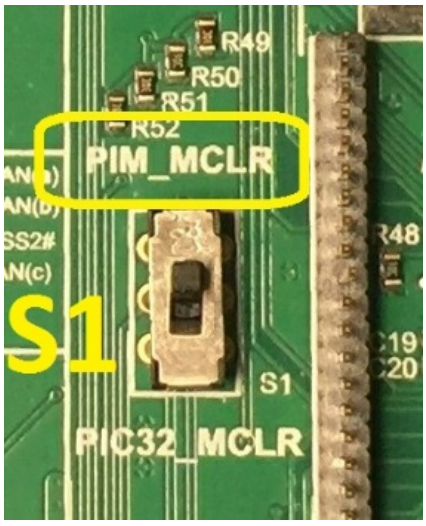
Description

PIC32 Bluetooth Audio Development Kit with the on-board PIC32MX470F512L device and the BM64 Bluetooth Module Daughter Board (see **Note**)

When using the PIC32MX470F512L and the BM64 Bluetooth Module Daughter Board, switch S1 on the PIC32 Bluetooth Audio Development Board should be set to *PIC32*_MCLR.



When using the PIC32MX270F512L PIM and the BM64 Bluetooth Module Daughter Board, switch S1 on the PIC32 Bluetooth Audio Development Board must be set to *PIM*_MCLR (see **Note**).



Note: The PIC32 Bluetooth Audio Development Kit includes a BTM805 Bluetooth Daughter Board; however, it must be replaced by the BM64 Bluetooth Module Daughter Board (AC320032-3) on headers J8 and J11.



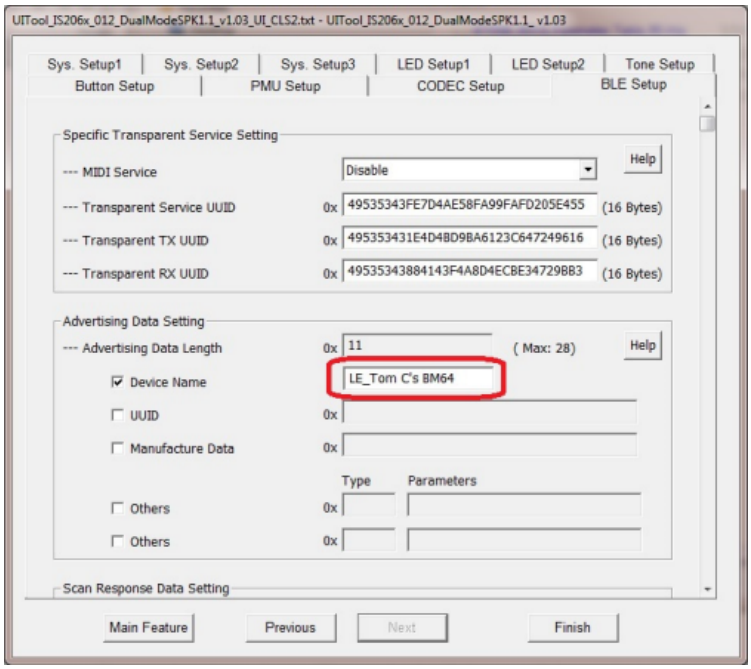
Running the Demonstration

This section describes how to run the demonstration.

Description

Do the following to run the demonstration:

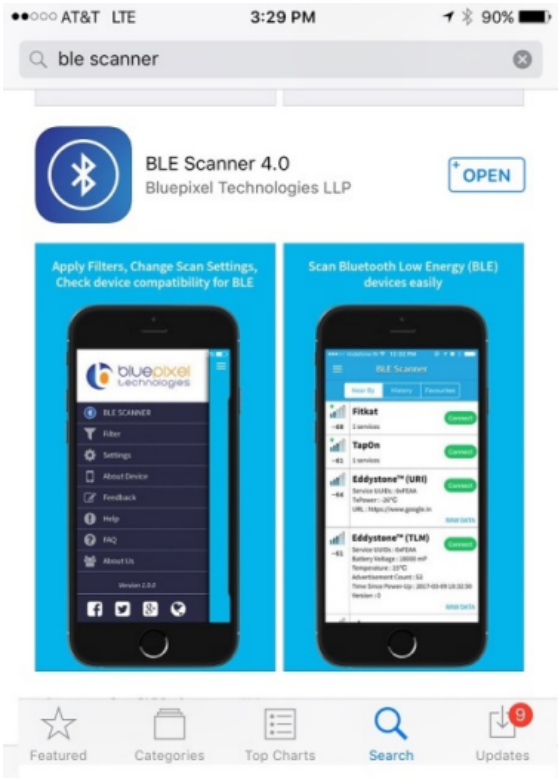
1. If you have not already done so, use the BM64_bootloader application to give the BM64 module a unique device name.



2. Compile the application selecting the appropriate MPLAB X IDE project configuration based on whether or not you are using the PIM (refer to Building the Application for details).
3. Program the target device, and run the application. After the program starts, the name of the application should appear on the display and the connection icon in the top-right corner will be white (advertising).

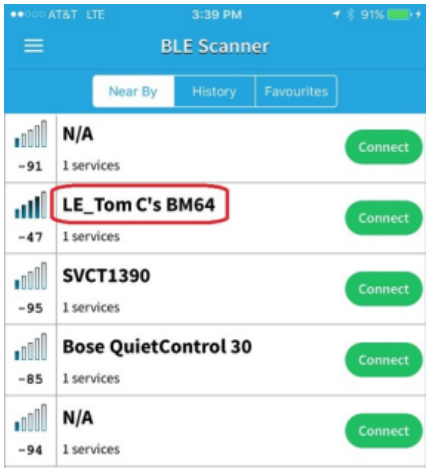


4. The mobile application, BLE Scanner, by Bluepixel Technologies LLP, is used in this demonstration. This application is available as a free download from iTunes. Locate and download this application to your smartphone.

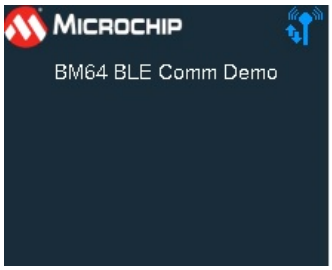


Note: If you are unable find this application for your smartphone, there are several other BLE scanner applications available for the iPhone, which can be used with this MPLAB Harmony application. The functionality will be the same, but the screens may appear somewhat different from those shown here.

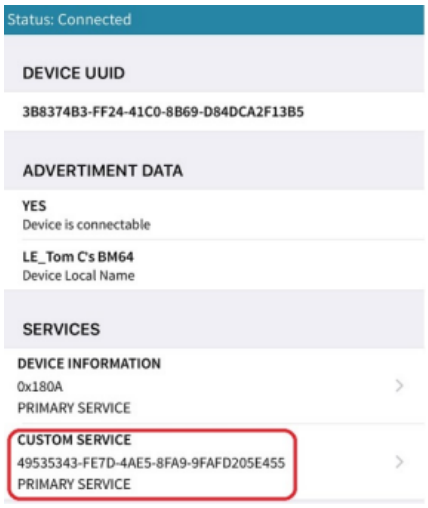
6. Start the BLE Scanner app. Once opened, it will begin scanning for BLE devices.



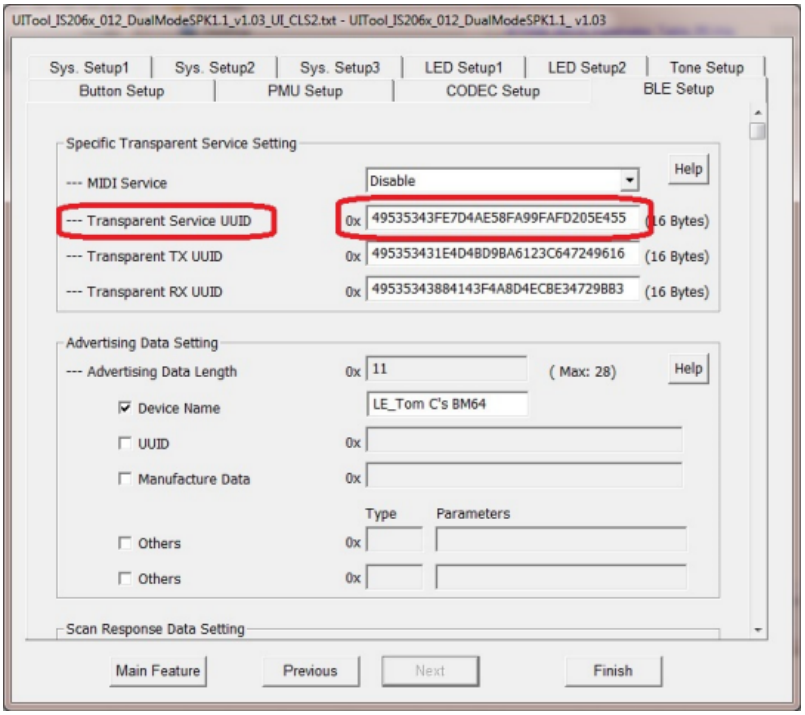
7. Select the device name that you set up for your BM64 in Step 1 (running the BM64_bootloader program) and tap **Connect**. The connection icon in the top right corner of the LCD should change from gray to bright blue.



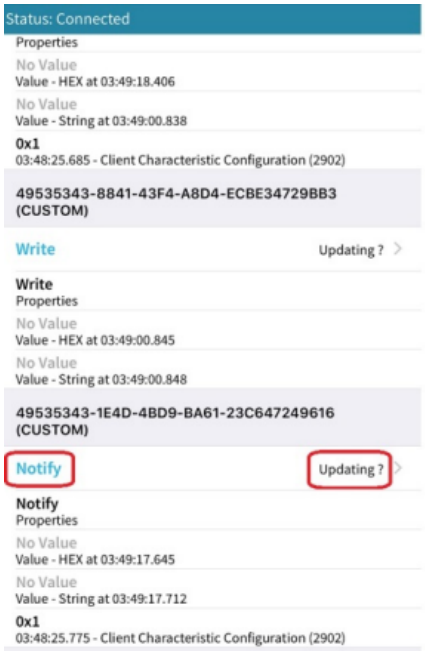
8. A list of services is displayed.



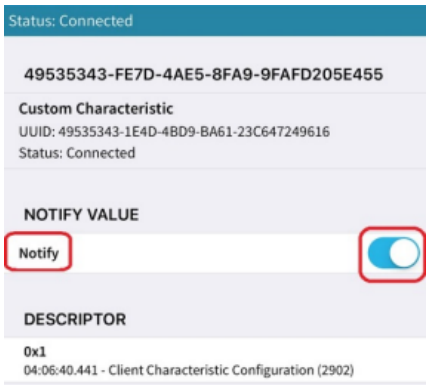
9. Select the service named CUSTOM SERVICE, with the UUID starting with 4953 and ending in E455. This service matches the Transparent Service UUID set up in the BLE section of the UI tool.



10. A list of characteristics are displayed.



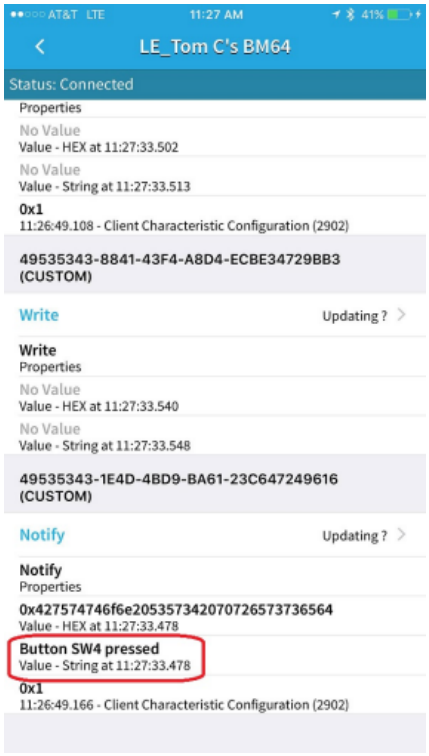
11. If necessary, scroll down and tap the characteristic labeled **Notify**. Tap the Updating ? icon to the right, which will display the following screen (ensure that the slider switch to the right of Notify is enabled (set to the right position). Tap the white less than (<) symbol in the upper left corner of the screen to return to the previous screen).



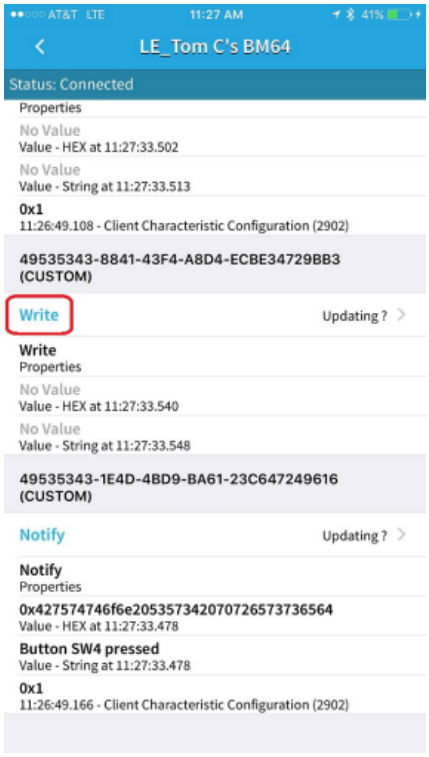
12. On the PIC32 Bluetooth Audio Development Kit, press one of the push buttons, SW1 through SW5.



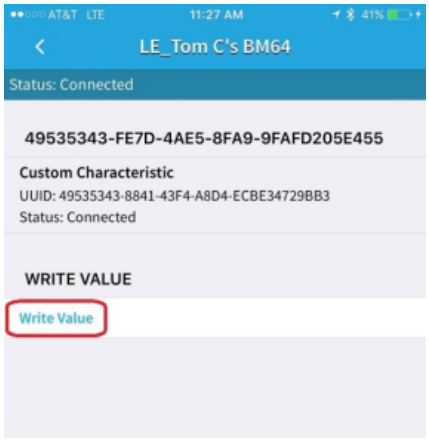
13. The text string 'Button SWx pressed' (where 'x' is the number of the button you pressed) should appear on your smartphone (in this example, SW4 was pressed):



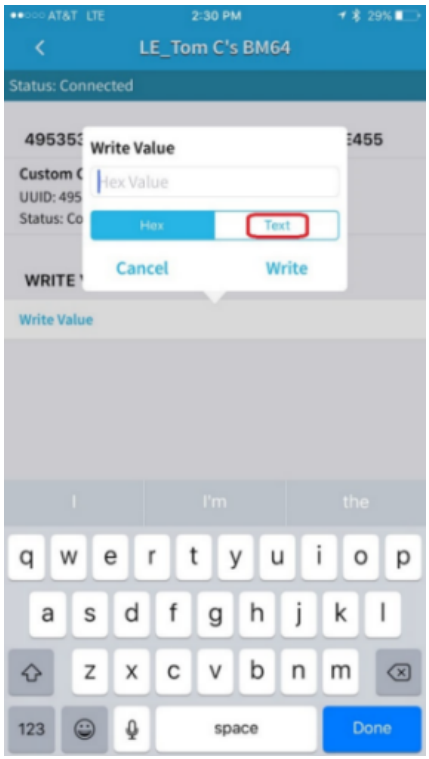
14. To send a text string from your smartphone back to the BM64, tap **Write** in the BLE Scanner screen:



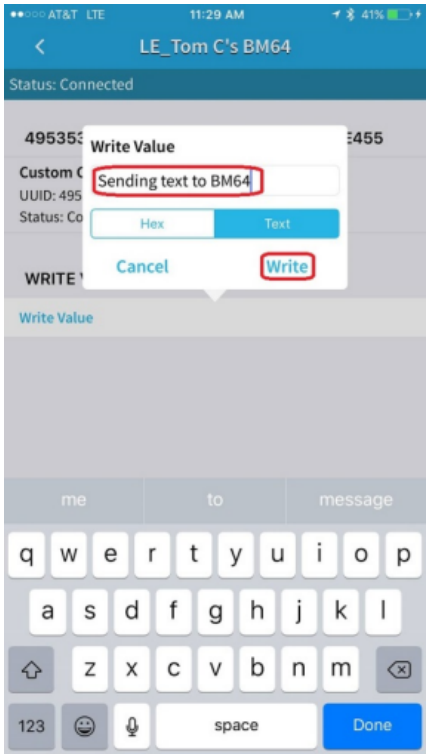
15. On the next screen, tap **Write Value**:



16. When the popup appears, tap **Text**.



17. Enter a message in the text box up to 20 characters in length, such as 'Sending text to BM64, and then tap **Write**.



18. The message you entered should appear in the LCD screen of the PIC32 Bluetooth Audio Development Kit.



19. If you send additional messages to the BM64 (always less than or equal to 20 characters in length), the previous messages will scroll up, and the newest message will appear at the bottom of the screen.



BM64_bootloader

This topic provides information on the supported demonstration boards, how to configure the hardware and how to run the demonstration.

Description

In this demonstration application, the processor on the PIC32 Bluetooth Audio Development Kit (BTADK) is used as a bootloader to configure the EEPROM of the BM64 module on the BM64 Module Daughter Board. This must be done prior to using the BM64 module with the BM64_a2dp_hfp application, since the I2S interface of the BM64 must be changed from being a I2S master (default) to an I2S slave.

When the application is running, the PIC32 Bluetooth Audio Development Kit processor appears as a virtual COM port on the PC.

Architecture

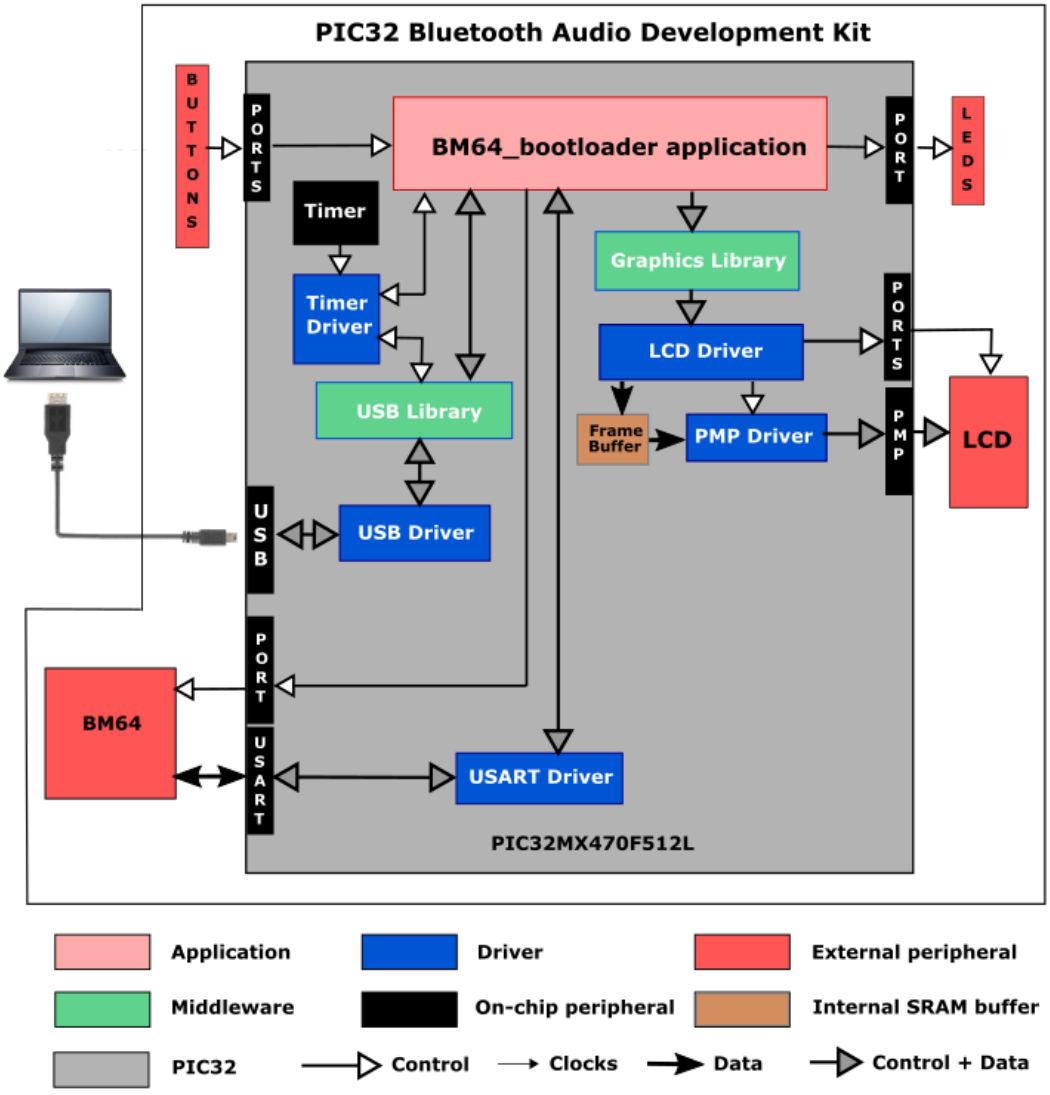
This application runs on the PIC32 Bluetooth Audio Development Kit, which contains a PIC32MX470F512L microcontroller with 512 KB of Flash memory and 128 KB of RAM running at 96 MHz with the following features:

- 220x176 color LCD
- Six push buttons (SW1-SW6)
- Five LEDs
- USB Device and Host interfaces
- Two X32 sockets, one for a Bluetooth module and a second for a codec or DAC

Notes:

1. The PIC32 Bluetooth Audio Development Kit comes with a BTM805 Bluetooth Daughter Board and an AK4384 Audio DAC daughter Board; however, you must replace the BTM805 daughter board with the daughter board for the BM64.
2. The USB Host interface as well as the AK4384 Audio DAC Daughter Board are not used in this application.

The following figure illustrates the application architecture.



The PIC32 microcontroller (MCU) runs the application code, and communicates with the BM64 over a USART interface operating at 115,200 baud. The BM64 module contains its own Bluetooth stack, but it is not used in this application since it is only used to update the BM64's configuration. The program takes up only about 32% of the PIC32MX470F512 microcontroller's program space, and 65% of its RAM.

The interface between the PIC32 MCU and the LCD is an 8-bit parallel master port (PMP). The program uses the MPLAB Harmony v2.x Graphics Library to draw on the screen. The buttons are also interfaced using GPIO pins.

As with any MPLAB Harmony application, the SYS_Initialize function, which is located in the `system_init.c` source file, makes calls to initialize various sub-systems, such as the clock, ports, board support package (BSP), USB, BM64, timers, UART, PMP, graphics, and interrupts.

The USB driver, timer, graphics, and the application state machines are all updated through calls located in the SYS_Tasks function in the `system_tasks.c` file. An interrupt handler in the `system_interrupt.c` file is used for receiving characters from the BM64. To avoid missing any characters, the UART driver's DRV_USART_TasksReceive is not used, and instead bytes are read directly in the interrupt routine using

DRV_USART_ReadByte and placed into the uartReceivedData buffer.

The program is derived from the cdc_serial_emulator demonstration program. The USB interface looks like a virtual communications port on the PC. It reads characters from the host PC over the USB interface and sends them straight through to the BM64 over the UART; likewise, it reads characters from the BM64 via the UART and sends them back to the host PC.

The application code is contained in the source file `app.c`, which contains a state machine (APP_Tasks) for the application, as well as two state machines for handling the USB interface. It first initializes the application, then receives a handle to the USB Device driver by calling USB_DEVICE_Open function with a mode of DRV_IO_INTENT_READWRITE. It also gets a handle to the UART driver by calling DRV_USART_Open also with a mode of DRV_IO_INTENT_READWRITE. Then it sets up a fixed baud rate of 115,200 baud, and registers an event handler, APP_USBDeviceEventHandler as a callback with the USB driver. It then does an initial call to USB_DEVICE_CDC_Read to receive any characters from the PC over the USB interface.

After that, it waits for characters from either side of the interface and transfers them to the other. Once the first character has come in from the UART via the interrupt, it disables interrupts temporarily and gets any further characters in the UART's FIFO in a tight loop, transferring them to the outgoing CDC buffer. If characters were received, it calls USB_DEVICE_CDC_Write to send them on to the PC.

If characters have come in through from the USB interface from the PC, it writes them to the transmit side of the UART interface to the BM64. Once all pending characters have been written, it issues a new USB_DEVICE_CDC_Read call to get more.

The APP_Tasks function also takes care of polling the buttons, and if pressed, either asserts the BM64's reset line for SW5 or asserts the BM64's MFB (multi-function button line) for SW4.

Demonstration Features

- Sending and receiving characters between the USB device interface of the PIC32 MCU and a host PC (see USB Device Driver Library)
- Sending and receiving characters between the USART of the PIC32 MCU and the BM64. (see USART Driver Library)
- Displays graphics on the 220x176 LCD of the PIC32 Bluetooth Audio Development Kit using the MPLAB Harmony Graphics Library (see MPLAB Harmony Graphics Composer Suite) and the Parallel Master Port (see PMP Driver Library)
- Processing of button pushes on the PIC32 Bluetooth Audio Development Kit using a state machine for contact debouncing (see Ports Peripheral Library)
- Use of two timers: one as a periodic 1 ms timer for the application, and a second used by the BM64 driver (see Timer Driver Library)

Tools Setup Differences

When building a new application, start by creating a 32-bit MPLAB Harmony project in MPLAB X IDE by selecting *File > New Project*. Select the PIC32MX Bluetooth Audio Development Kit. Click the MPLAB Harmony icon to start the MPLAB Harmony Configurator (MHC). Under BSP Configuration, select PIC32 Bluetooth Audio Development Kit (BM64+AK4384).

Under *Harmony Framework Configuration >Drivers >Timers*, select **Use Timer Driver**. The remaining options in this section do not require any changes.

In the *USART* section, select **Use USART Driver?**. Expand *USART Driver Instance 0*. Change the USART Module ID to USART_ID_2, and the Baud Rate to 115200.

Under *Harmony Framework Configuration >USB Library*, select **Use USB Stack?**. Expand *Select Host* or *Device Stack*. USB Device should already be selected. Change the Number of Endpoints Used to 3. Under USB Device Instance 0, expand Function 1. Change the Device Class to CDC. Change the Product ID Selection to "cdc_serial_emulator_demo", which will fill in the Vendor and Product ID fields.

If your application is going to be using graphics, as this one does, within *Graphics Stack*, select **Use Graphics Stack**?. Further down, within *Use Harmony Graphics Composer Suite? > Middleware > Use Aria User Interface Library?*, clear **Enable Touch** since the LCD on the PIC32 Bluetooth Audio Development Kit does not support a touchscreen. Since you are using the LCD, you will also need to go back to the Drivers section, and within PMP, select **Use PMP Driver?**. Expand that section, and change the value of Strobe Wait Sates to PMP_STROBE_WAIT_4.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the demonstration.

Description

To build this project, you must open the `BM64_bootloader.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/bluetooth/audio/BM64_bootloader`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`BM64_bootloader.X`	`<install-dir>/apps/bluetooth/utilities/BM64_bootloader/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
bt_audio_dk	bt_audio_dk+bm64	This demonstration runs on the PIC32MX470F512L mounted on the PIC32 Bluetooth Audio Development Kit and the BM64 Bluetooth Module Daughter Board.

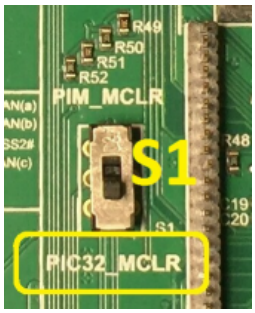
Configuring the Hardware

This section describes how to configure the supported hardware.

Description

PIC32 Bluetooth Audio Development Kit with the on-board PIC32MX470F512L device and the BM64 Bluetooth Module Daughter Board (see **Note**).

Switch S1 on the PIC32 Bluetooth Audio Development Board should be set to PIC32_MCLR.



Note: The PIC32 Bluetooth Audio Development Kit includes a BTM805 Bluetooth Daughter Board; however, it must be replaced by the BM64 Bluetooth Module Daughter Board (AC320032-3).

Running the Demonstration

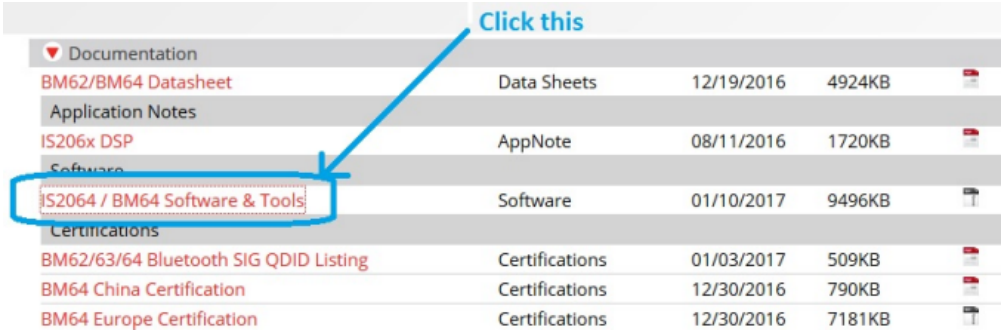
This section describes how to run the demonstration.

Description

Do the following to run the demonstration:

Program the Device and Obtain Tools

1. Compile, download, and run the program on the target device. There is only one MPLAB X IDE project configuration for the Bootloader.
2. Download the BM64 Software and Tools from here: http://www.microchip.com/wwwproducts/en/BM64, under *Documentation* > *Software* > *(IS2064 / BM64 Software and Tools)*.

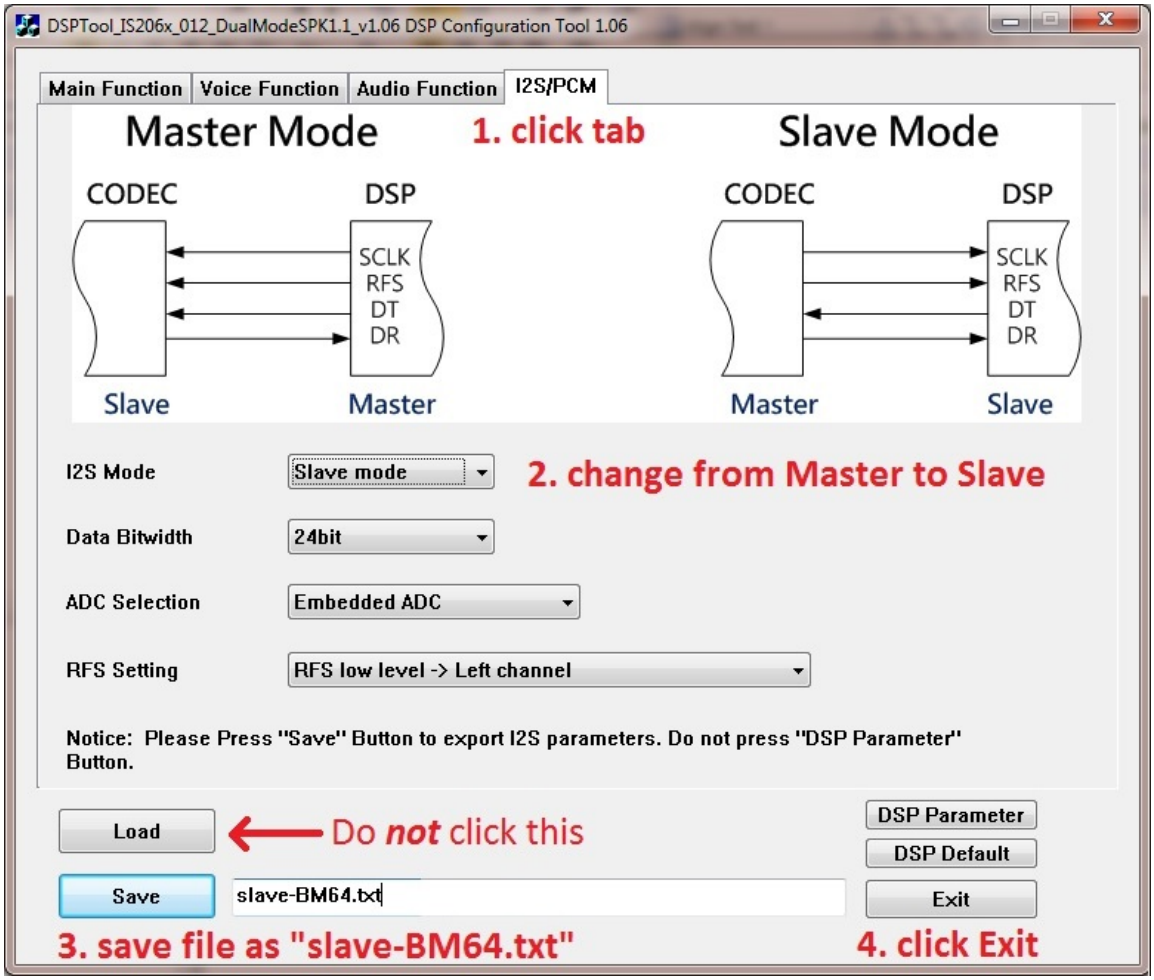


Unzip wherever it is convenient for you (e.g., `C:\ IS2064_BM64 Software & Tools`), and unzip any zip files contained within the folders.

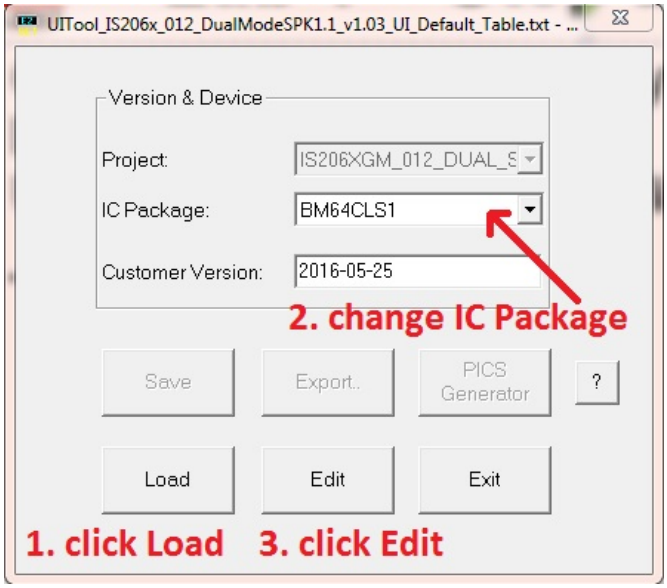
Program the BM64 Module

The sequence of steps to program the BM64 Module are as follows:

1. Run the tool `C:\IS2064_BM64 Software & Tools\ DSPTool_IS206x_012_DualModeSPK1.1_v1.06.exe`.
2. Do not click on the Load button. Under the I2S/PCM tab, set the I2S Mode to “Slave mode”. Then, save the file as: `slave-BM64.txt`, and exit the DSP Tool.

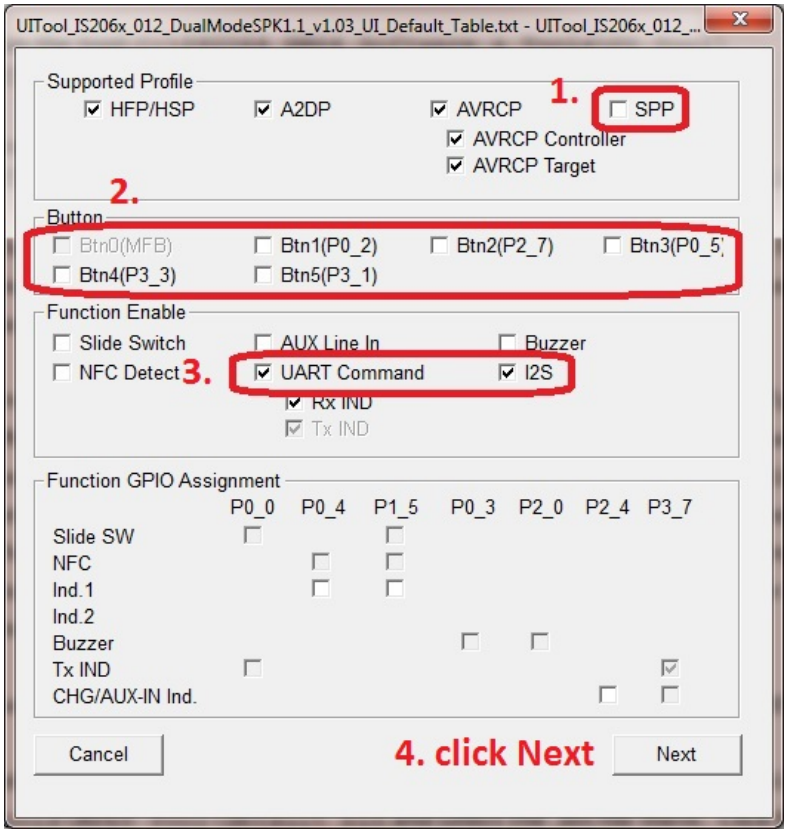


3. Run the tool: `C:\IS2064_BM64 Software & Tools\UI Tool\UITool_IS206x_012_DualModeSPK1.1_v1.03.exe`.
4. Click **Load**, and load the file: `UITool_IS206x_012_DualModeSPK1.1_v1.03_UI_Default_Table.txt`. Then, change the IC Package type to BM64CLS1, and then click **Edit**.

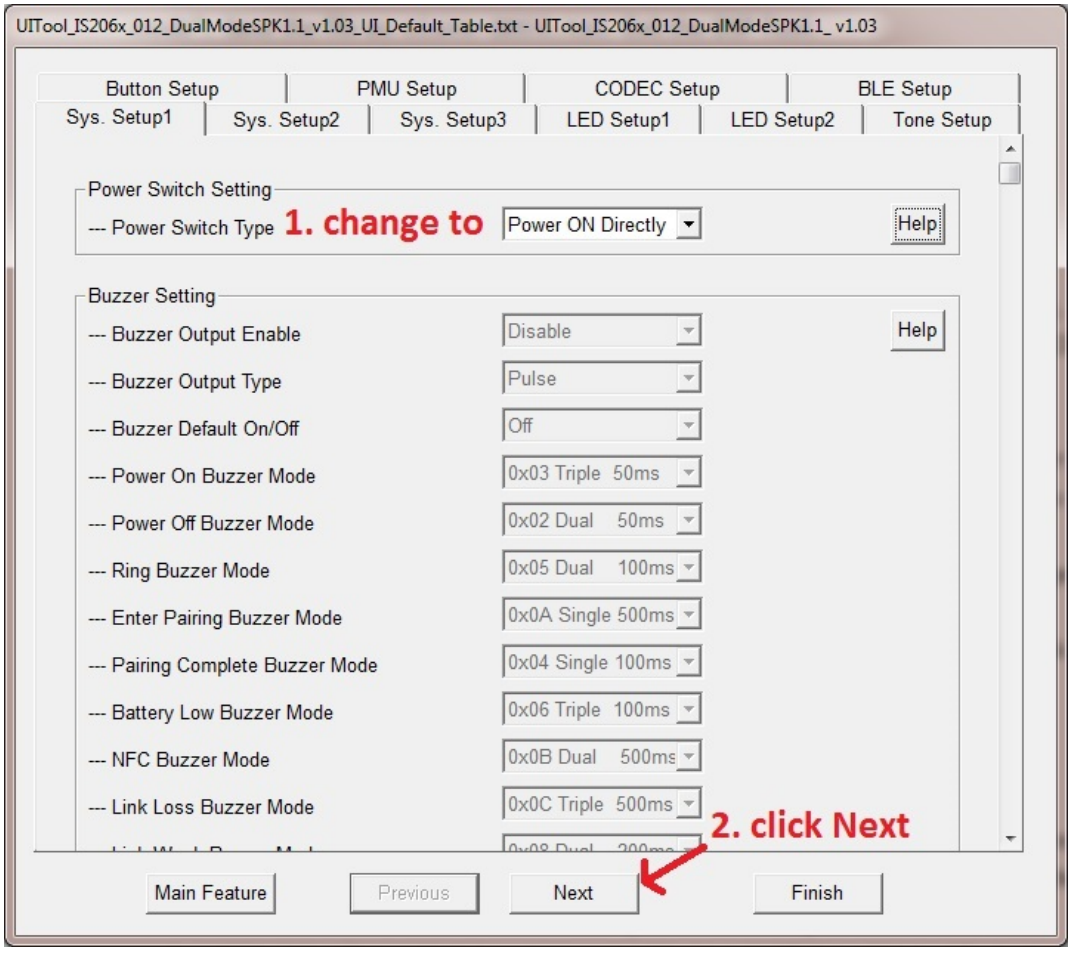


5. Do the following In the Main dialog:
   - Clear SPP

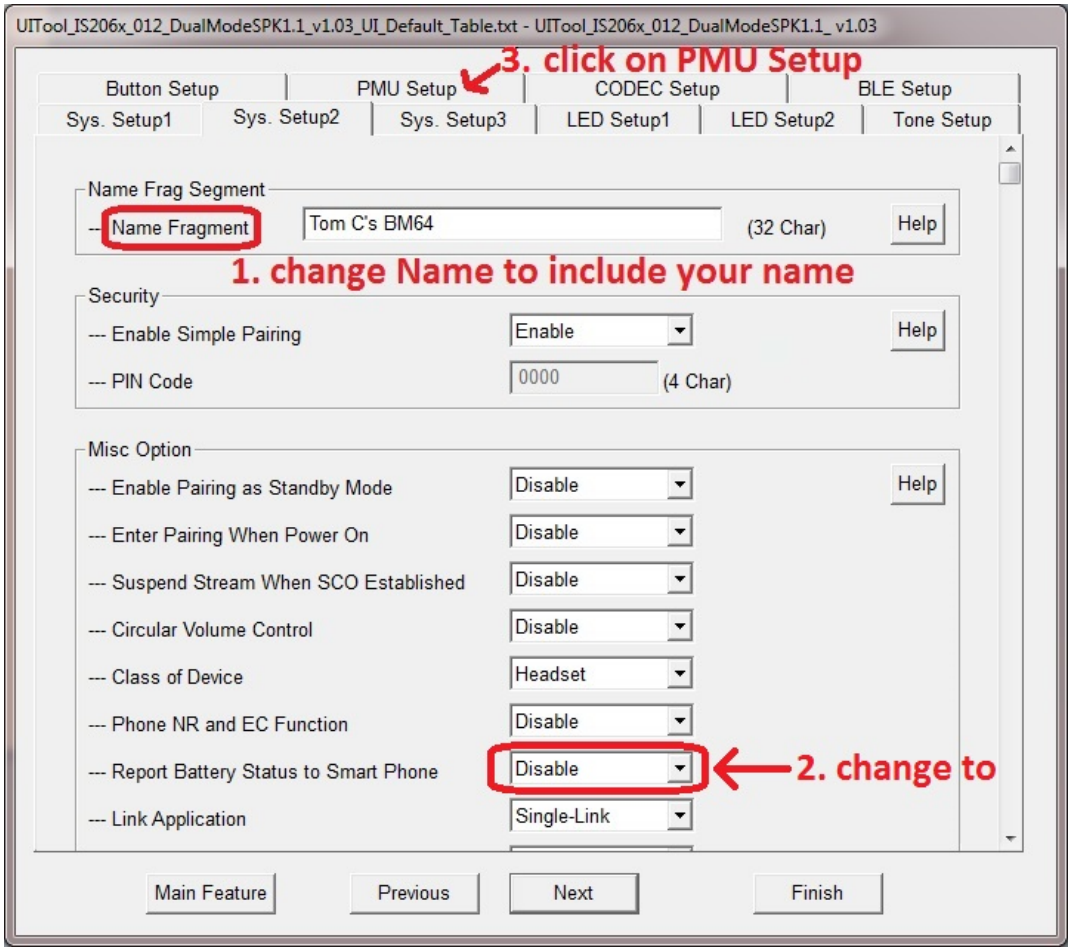
- Clear all of the Button check boxes
- Select UART Command and I2S, and then click **Next**



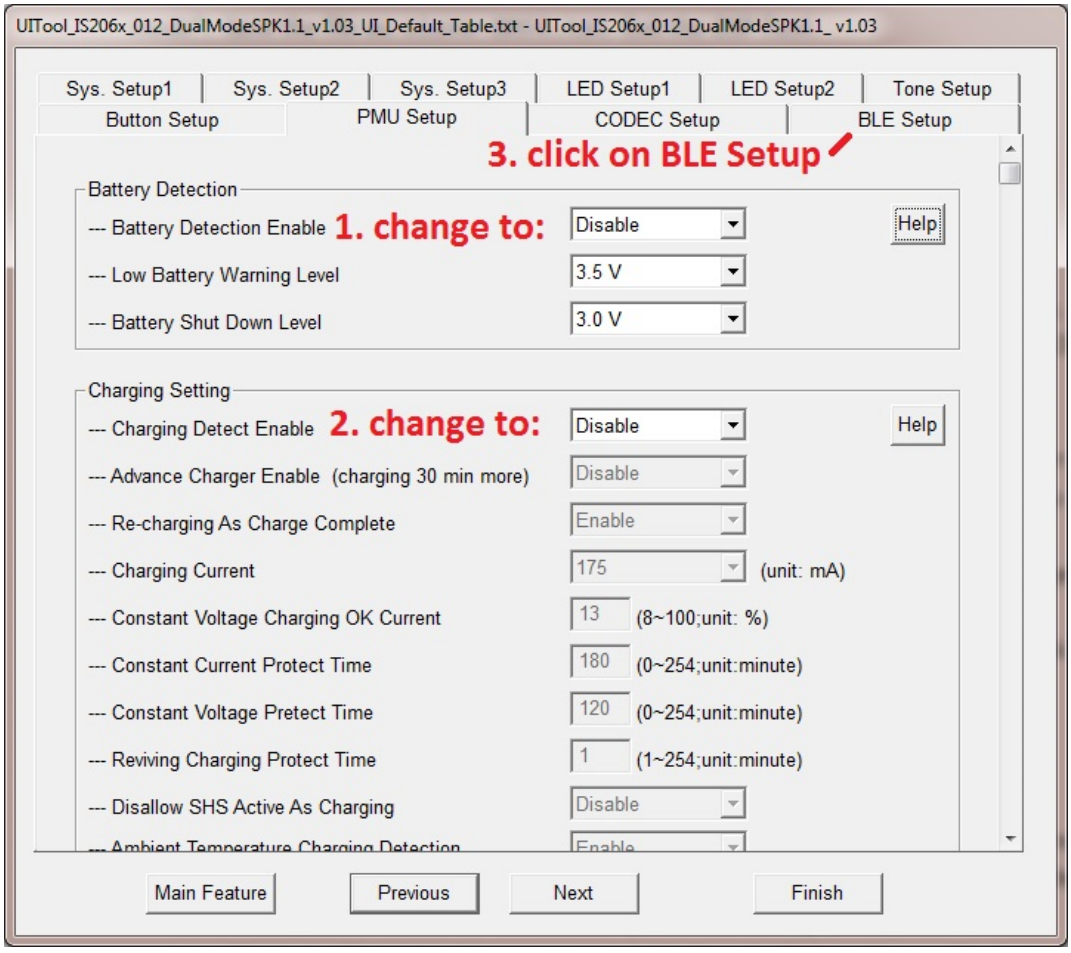
6. Under the Sys Setup1 tab, change the Power Switch Setting/Power Switch Type to “Power ON Directly”, and then click **Next**.



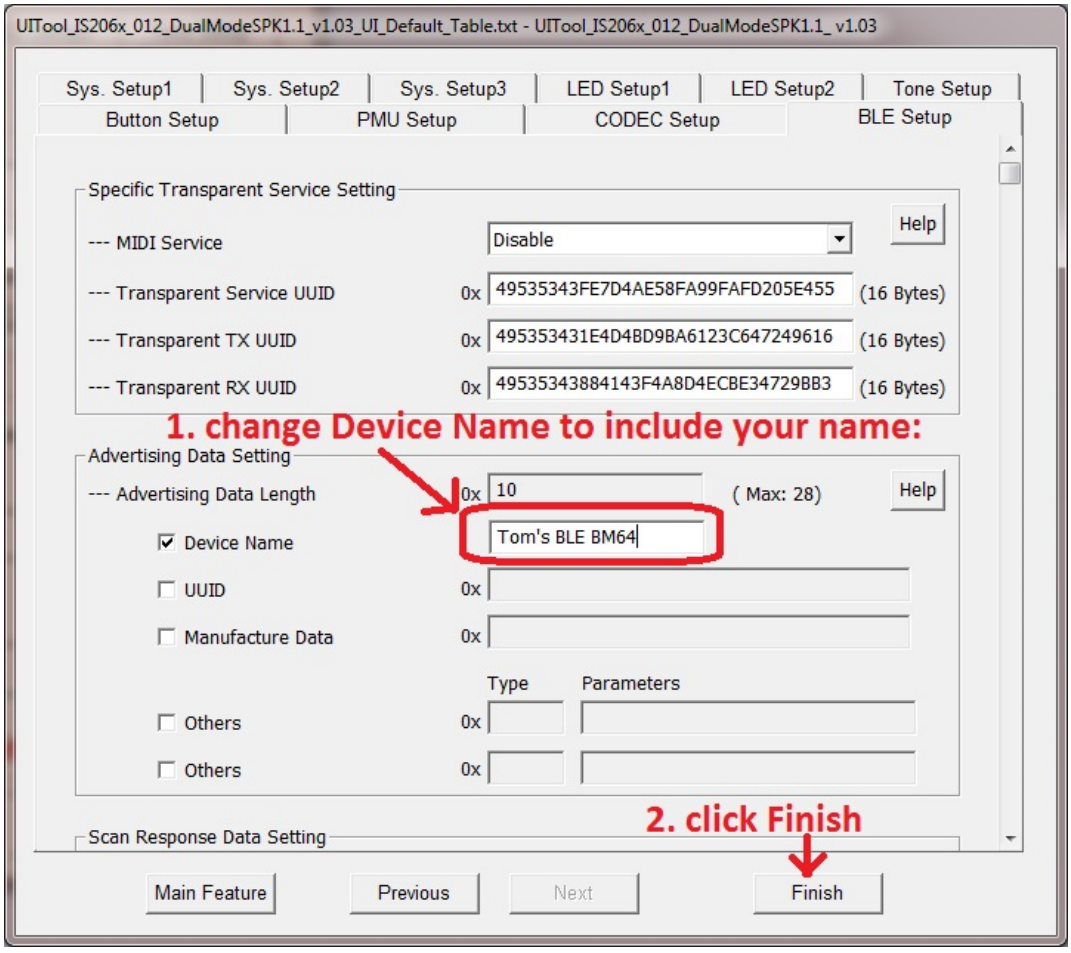
7. Under the Sys Setup2 tab, change the Name Fragment to whatever you want (this name will show up in your Smart Phone's Bluetooth setup). Change the Report Battery Status to Smart Phone setting to Disable, and then select the PMU Setup tab.



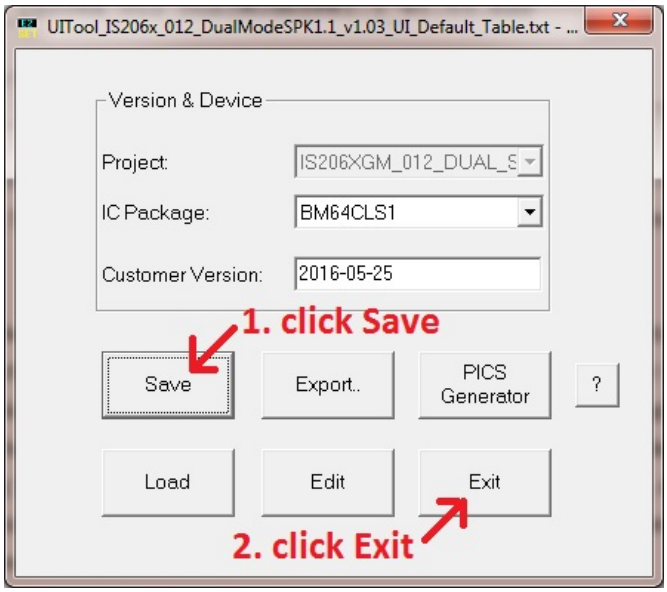
8. Under PMU Setup, change Battery Detection/Battery Detection Enable to Disable and also change Charging Setting/ Charging Detect Enable to Disable, and then select the BLE Setup tab.



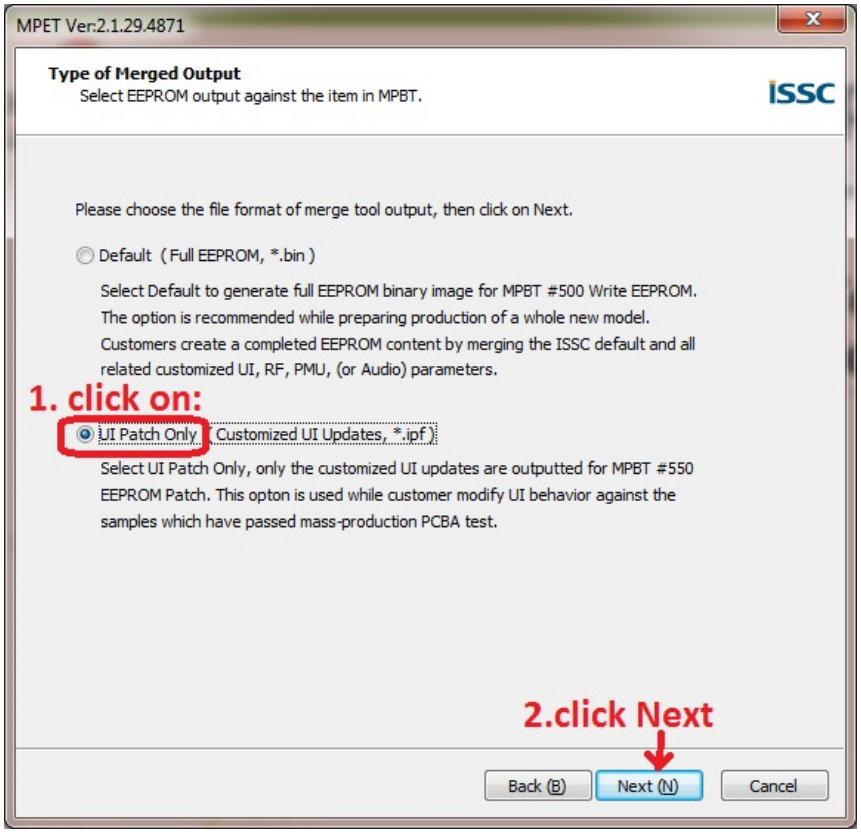
9. Under BLE Setup tab, change the Device Name to whatever you want (this name will show up in your Smart Phone's BLE setup), and then click **Finish**.



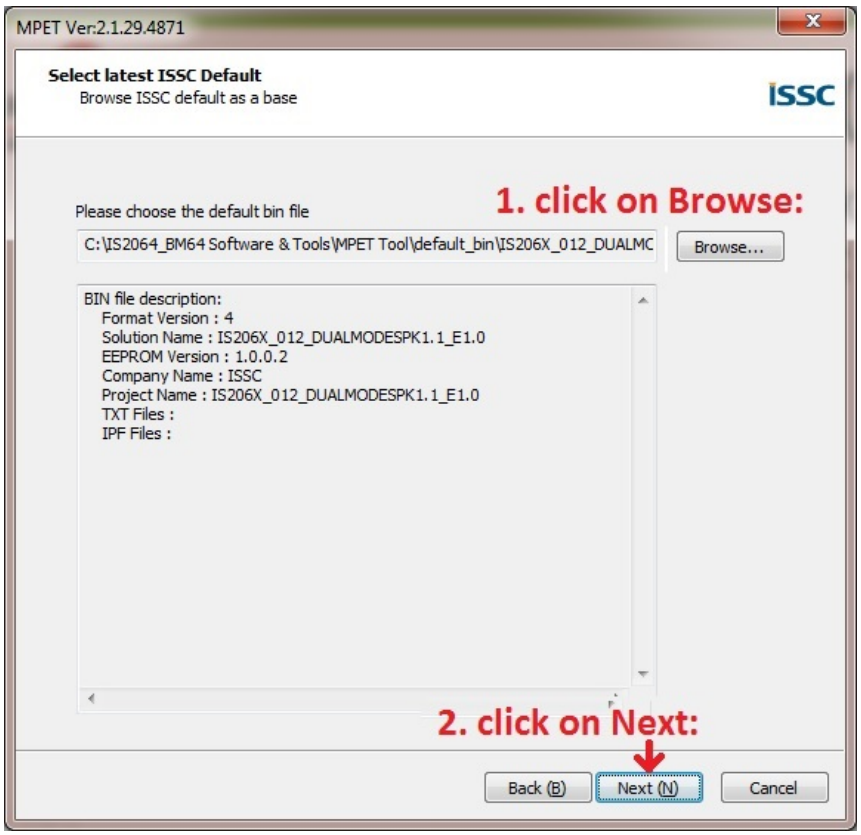
10. Save the file as: BM64 UI, and then click **Exit**.



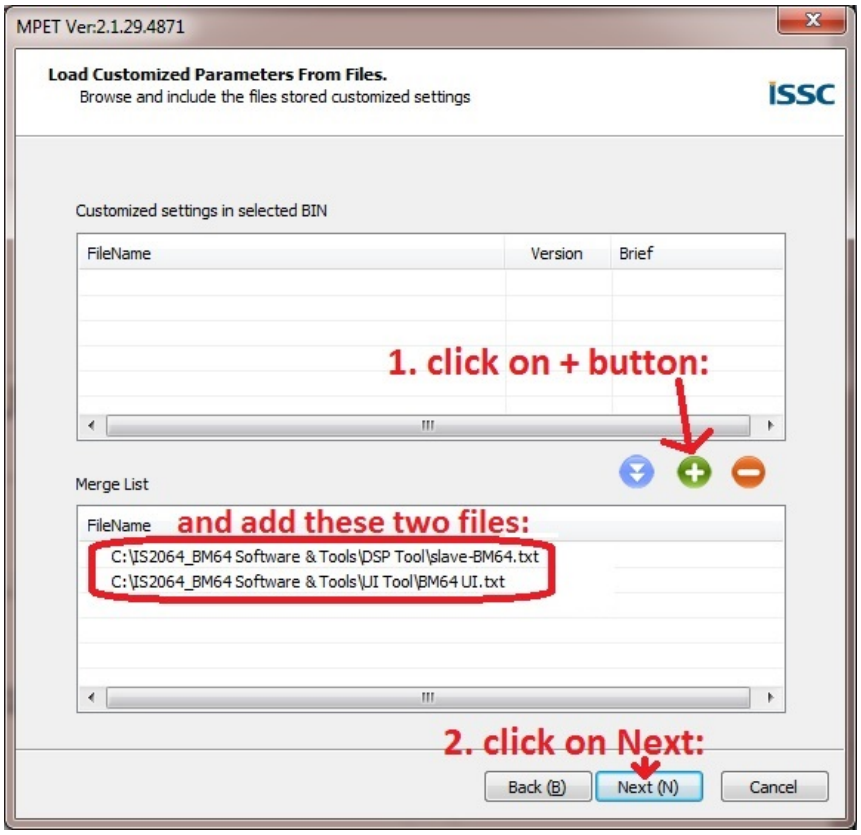
11. Run the tool: `C:\IS2064_BM64 Software & Tools\MPET Tool\MPET.exe`.
12. Select UI Patch Only, and then click **Next**.



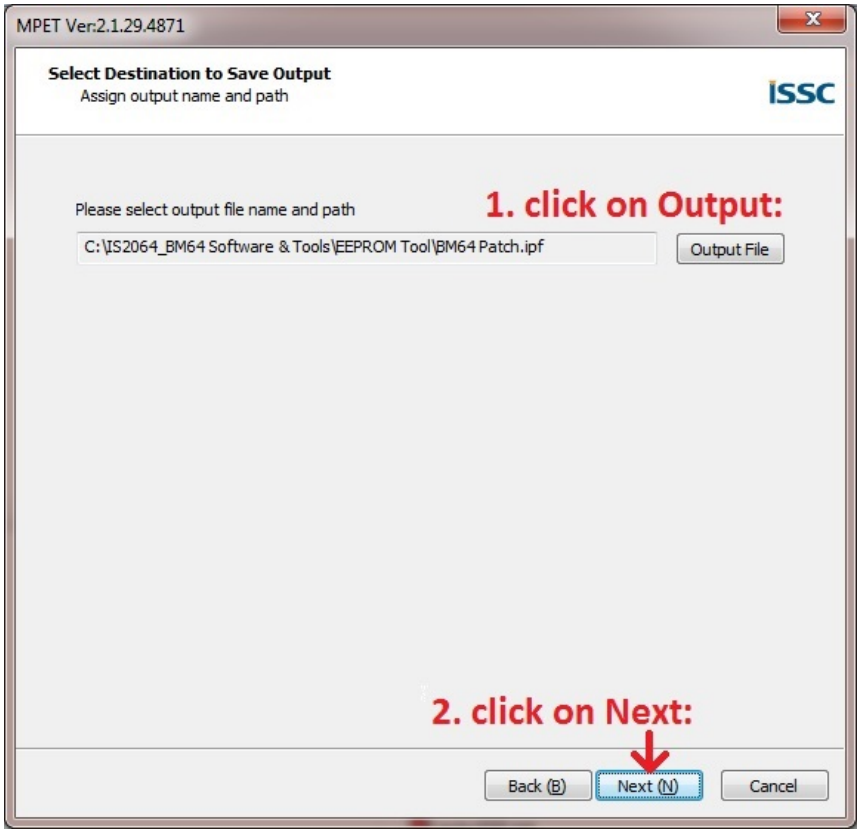
13. Click **Browse**, and browse to the folder `C:\IS2064_BM64 Software & Tools\MPET Tool\default_bin` and select the `.bin` file there, and then click **Next**.



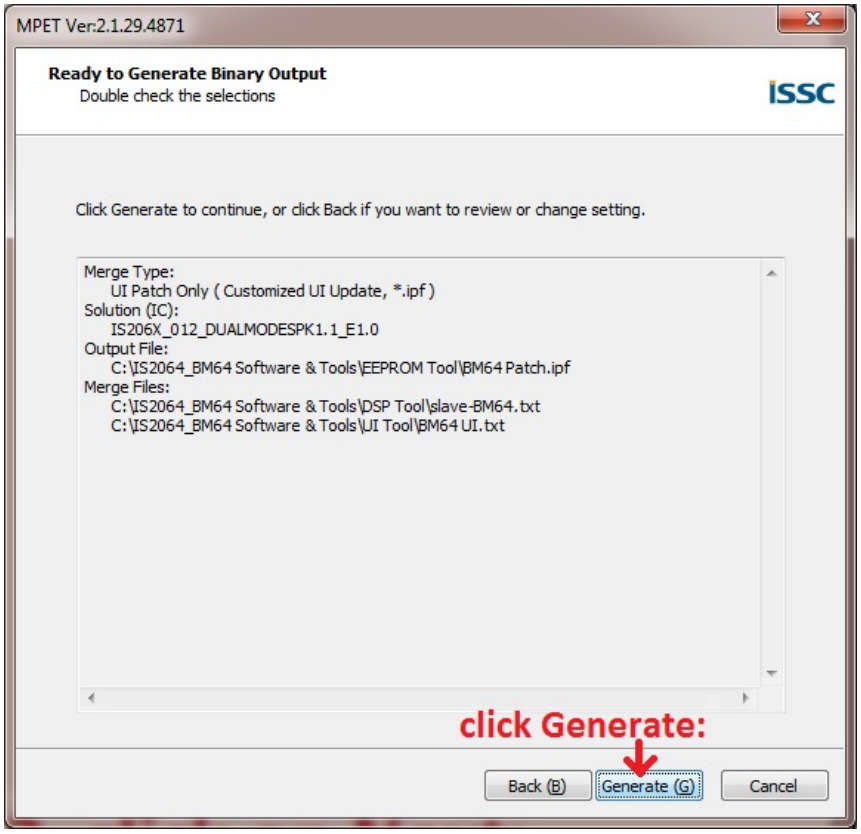
14. Click the icon and load the file `slave-BM64.txt` that you saved earlier in Step 2. Click the icon again and load the file `BM64 UI.txt` you saved earlier in Step 10, and then click **Next**.



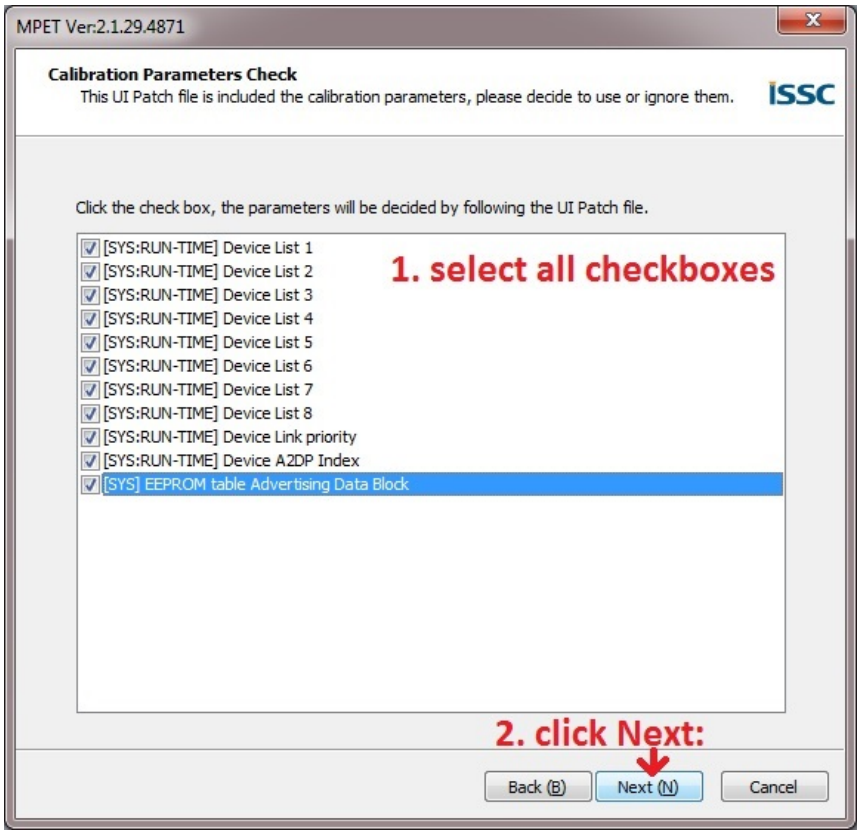
15. Click the Output File button and save the file as `BM64 Patch`, and then click **Next**.



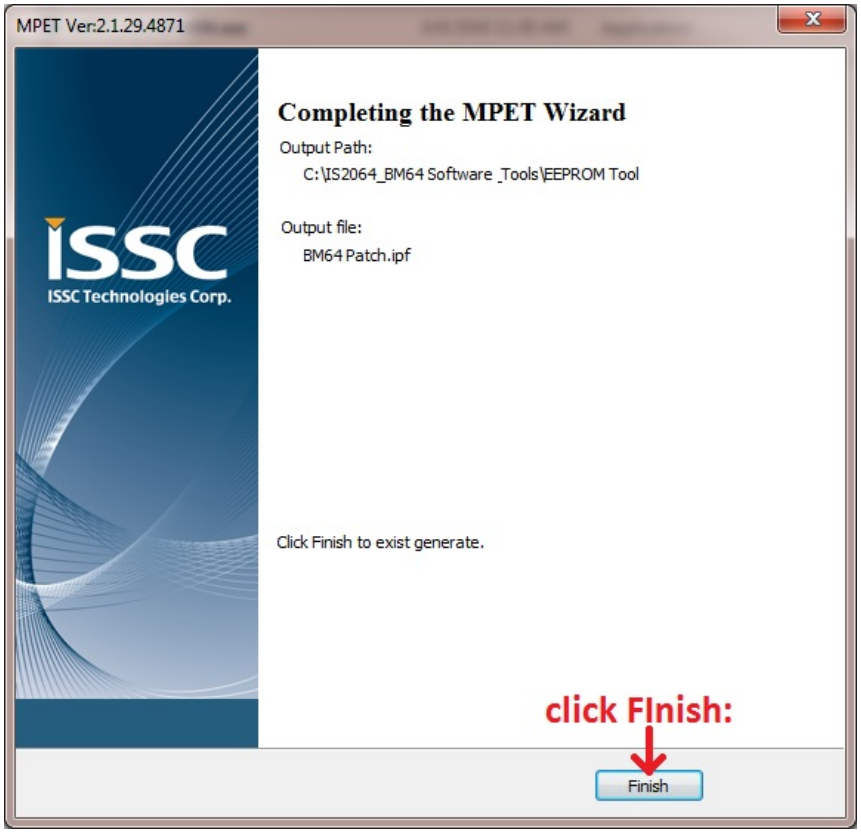
16. On this screen, click **Generate**.



17. In the next dialog, select all 11 check boxes, and then click **Next**.

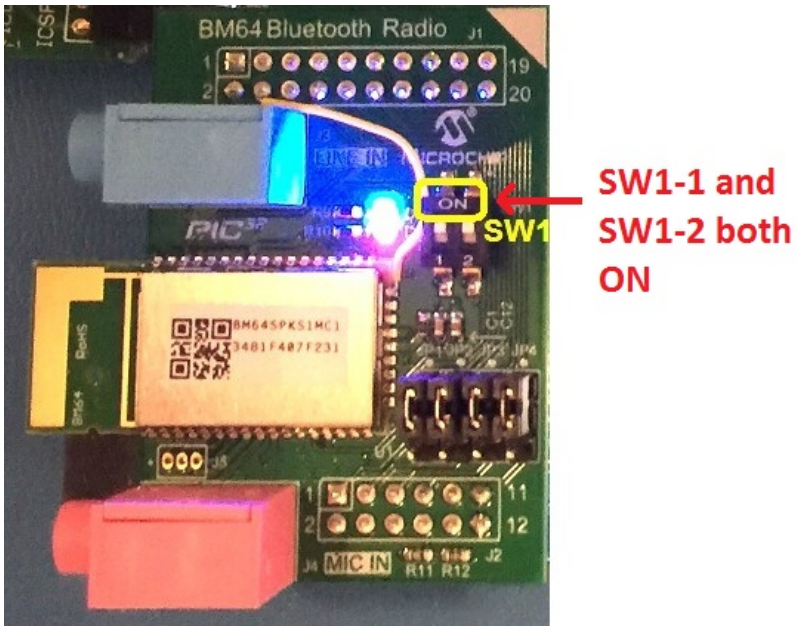


18. Finally, click **Finish**.

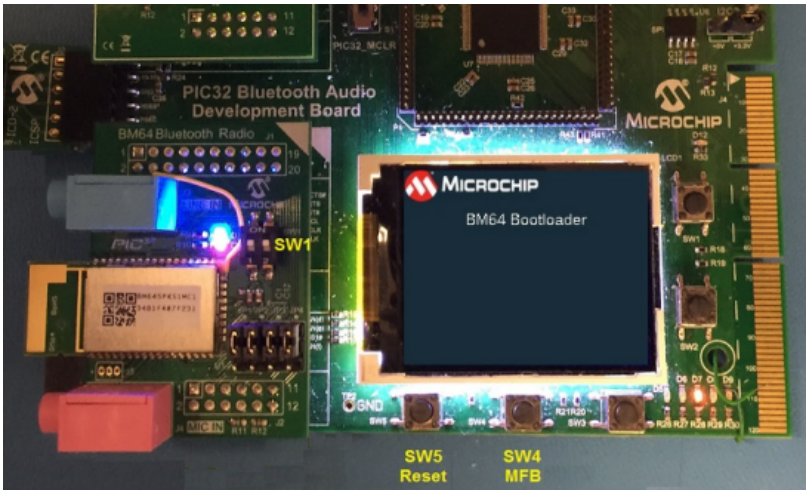


Running the Demonstration on the Development Hardware

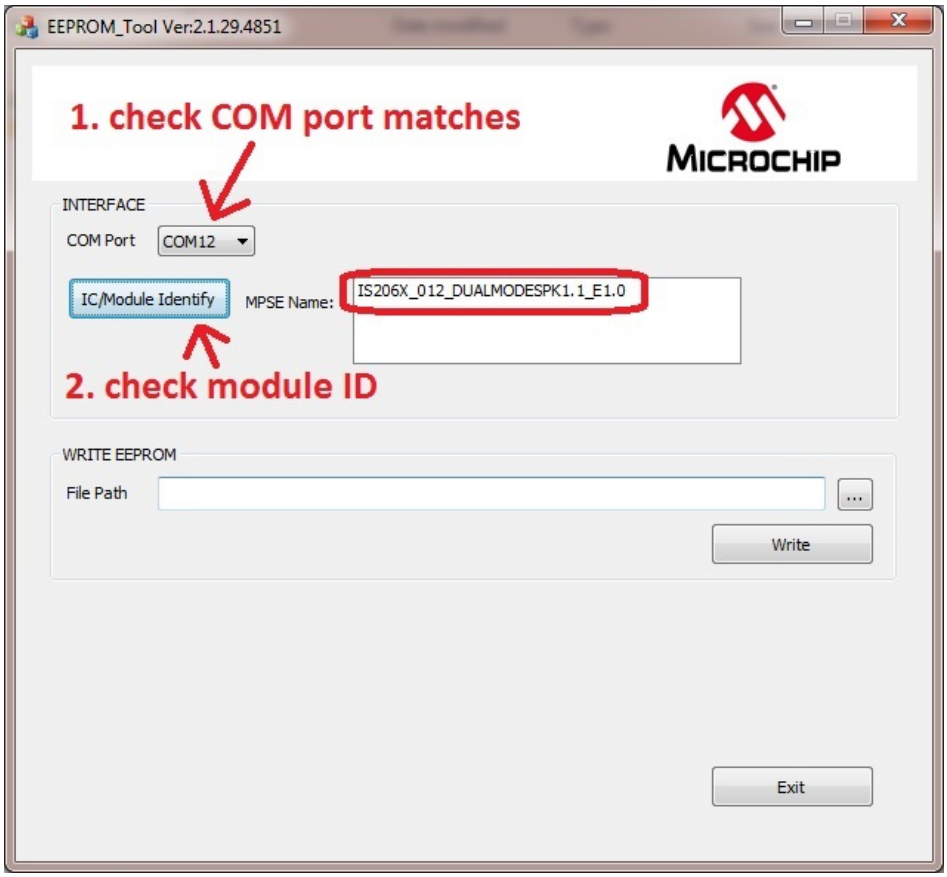
1. Plug one end of a USB cable into the mini-B USB jack of the Bluetooth Audio Development Kit (next to SW6 at the top of the board), and the other into your PC. Set both switches SW1-1 and SW1-2 on the BM64 Module daughter board to the ON position.



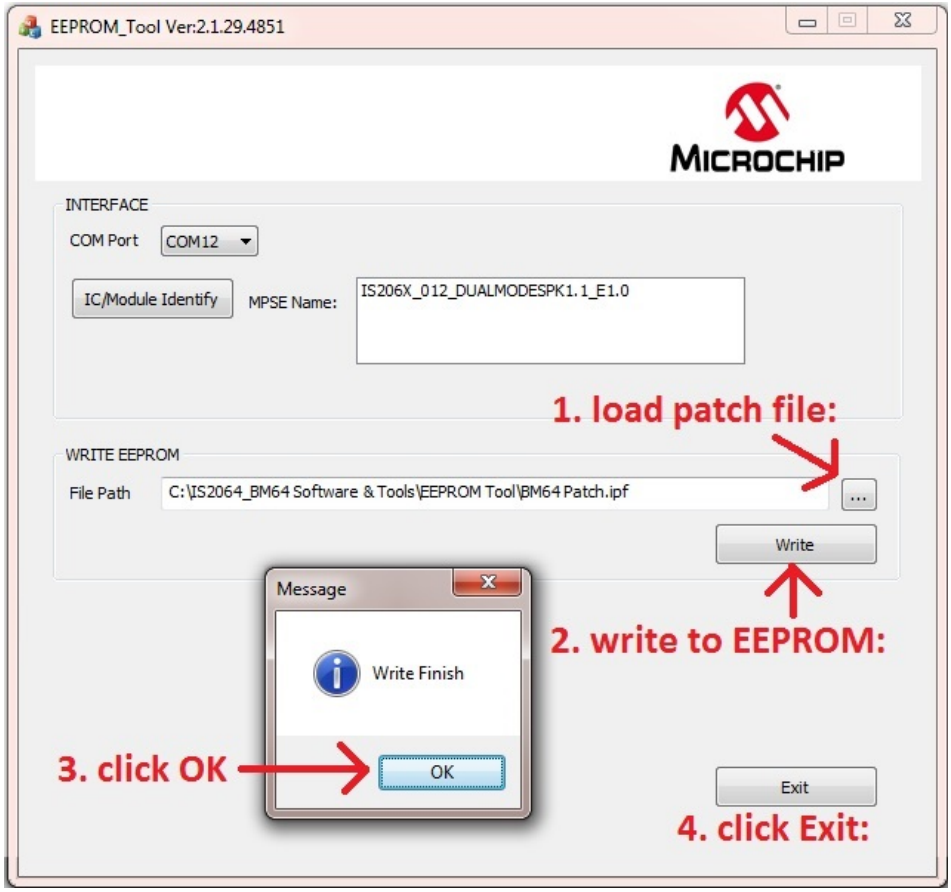
2. Press and release SW5 (Reset button), and then press and release SW4 (MFB button). The two LEDs on the BM64 module should light up steady as shown in the following figure.



3. Run the tool: `C:\IS2064_BM64 Software & Tools\EEPROM Tool\ EEPROM_Tool_V4851.exe`.
4. Ensure that the correct COM Port is selected (you can use Window's Device Manager to verify this), and then click **IC/Module Identify**. It should display one or two strings starting with IS206X…



5. To the right of WRITE EEPROM, click on the ... button and browse to the `BM64 Patch.ipf` file you specified earlier, and then click **Write**. When done and the dialog appears, click **OK**, and then click **Exit**.



6. On the BM64 Module Daughter Board, set both switches SW1-1 and SW1-2 to the OFF position.



7. On the Bluetooth Audio Development Kit, press and release the SW5 (Reset) button. The two LEDs on the BM64 Module should turn off. The BM64 Module is now ready to run with the BM64_a2dp_hfp application.

Control Descriptions

Control	
SW4	Multi-Function Button (MFB)
SW5	Reset

Data Demonstrations

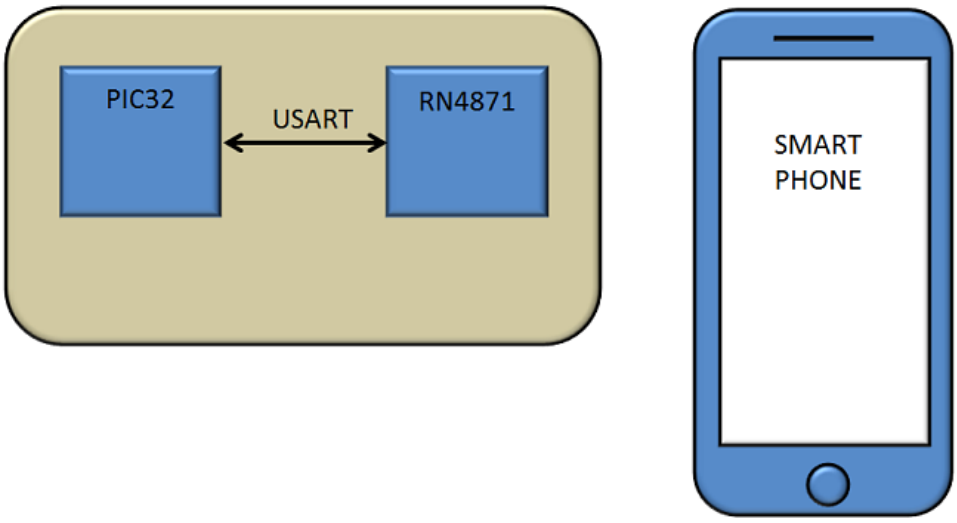
This topic provides information on how to run the PIC32 Bluetooth Basic Stack Library "Data Demonstration" applications that are included free-of-charge in this release of MPLAB Harmony.

ble_rn4871_comm

This topic provides information on how to run the PIC32 Bluetooth Low-Energy Data Demonstration that in this release of MPLAB Harmony.

Description

This demonstration application shows how to connect to a RN4871 BLE radio that is a Stack on Radio, which communicates through a USART interface. These USART commands are ASCII-level commands that set up the radio, which includes broadcast name, setting of public and private services, and data to these services. This demonstration also has a USB CDC COM port for `printf` style message debug.



Demonstration Features

USART Driver

Timer Driver

USB CDC (Print message debug)

Tools Setup Differences

Not applicable.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Bluetooth Demonstration.

Description

To build this project, you must open the `ble_rn4871_comm.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/bluetooth/data/ble_rn4871_comm`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`ble_rn4871_comm.X`	`<install-dir>/apps/bluetooth/data/ble_rn4871_comm/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx_xlp_sk	pic32mx_xlp_sk	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32 XLP Starter Kit.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

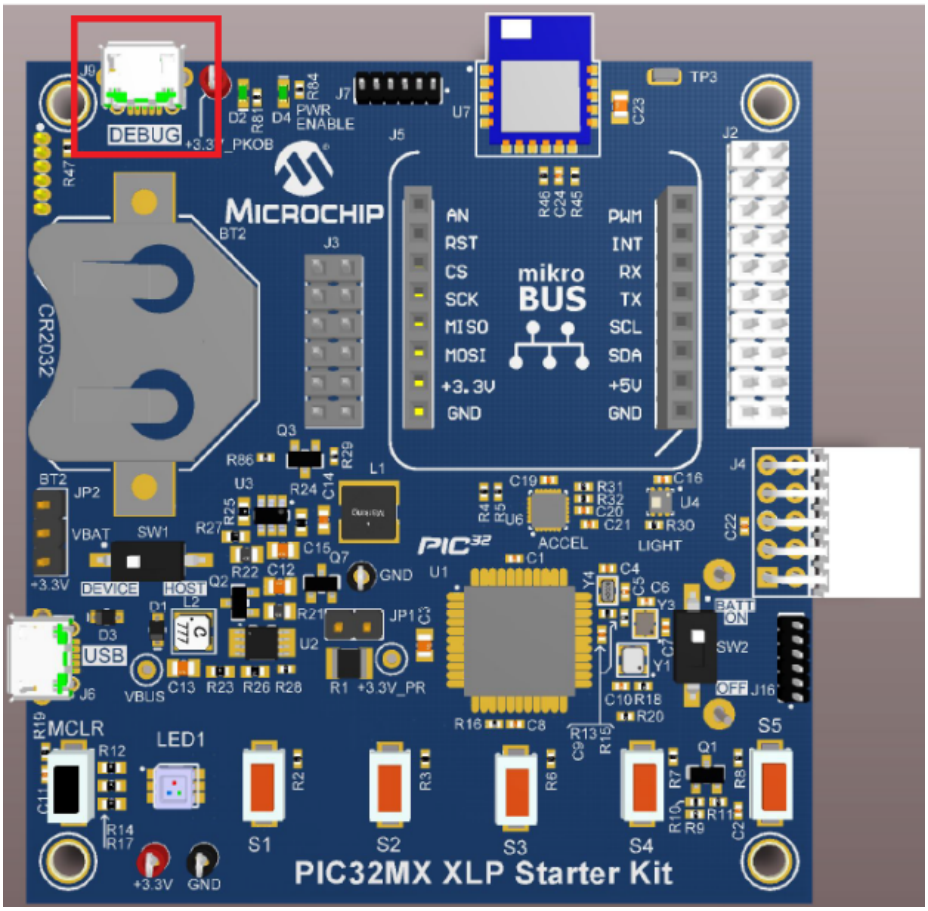
PIC32MX XLP Starter Kit

No hardware related configuration or jumper setting changes are necessary.

Configuration: pic32mx_xlp_sk

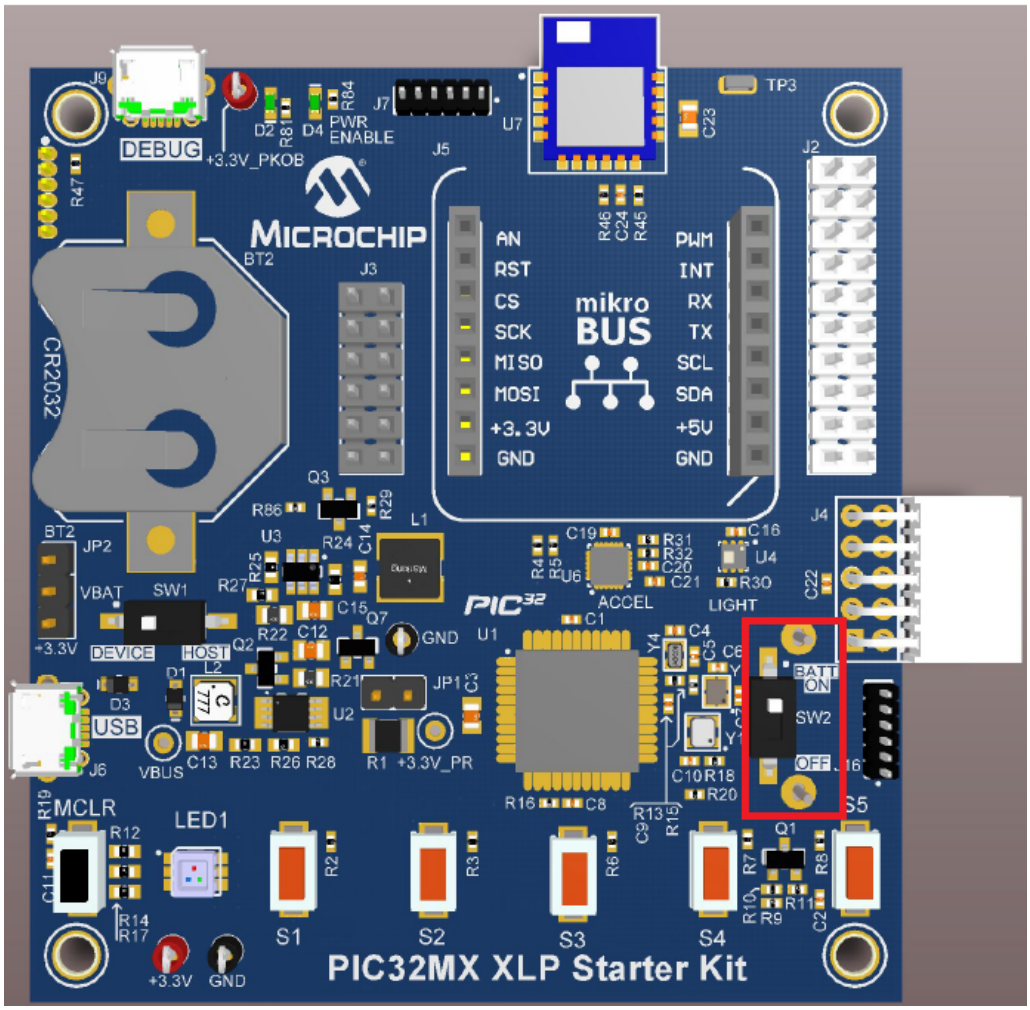
Wall power option:

Connect power to the debugger “J9”.



Battery Operation:

AA battery holder located on the back side of the board (batteries not included with starter kit). Insert batteries per the holder markings. Put the switch, SW2, to “ON” to power the board from the batteries. Put the switch, SW2, to “OFF” to stop powering the board from the batteries.



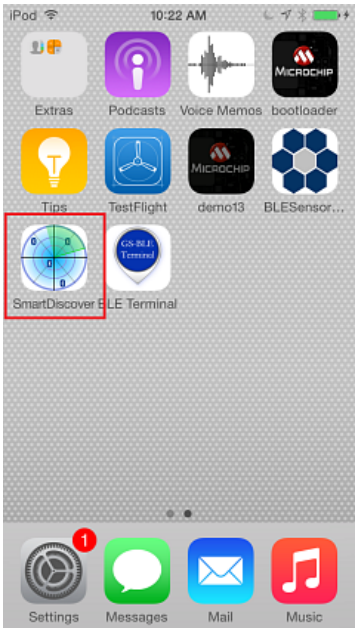
Running the Demonstration

This demonstration performs basic BLE data transmission.

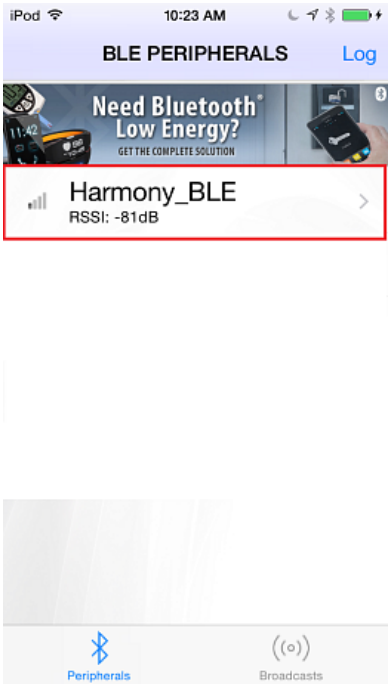
Description

Note: Before running the demonstration, it is necessary to install the Smart Discover Android™ application or Smart Discover app from the iTunes® Play Store.

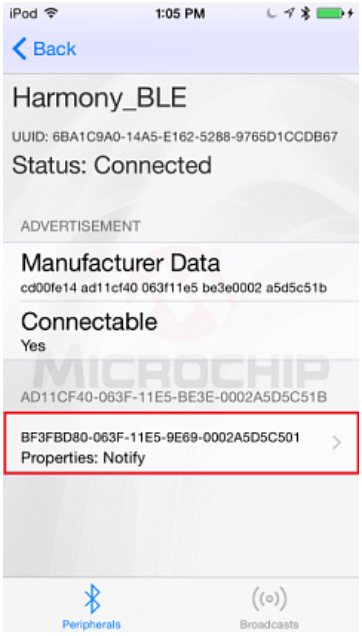
1. Compile and program the target device. While compiling, select the appropriate MPLAB X IDE project configuration based on the demonstration board. Refer to Building the Application for details.
2. If you have not already done so, install the Smart Discover App on either your Android or Apple® iOS device, found on the Play Store/App Store; otherwise, skip this step.
3. Open the Smart Discover App.



4. Scan for the available Bluetooth devices. The target Bluetooth device, Harmony ble, should also be displayed in the list of available devices on your smartphone.



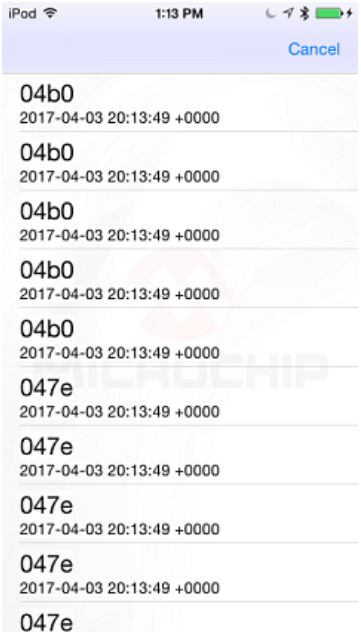
5. Select the device to pair and connect.
6. Select the box at the bottom (highlighted in Red).



7. Enable Notify.



8. Select **Read Log** to view the data coming across.



BLE Status	LED Color
Ready to be connected	Blue
Connected	Green

Additional Debug messages and output are available via the USB CDC com port. To view this information, open a terminal window and connect to the new com port that will appear once the USB has been connected. Set the baud rate to 115200 and send 1 command/character/line feed to the device in the terminal window. Debug information will begin to stream in the terminal.

data_basic

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This demonstration performs full duplex data transfer over a Bluetooth connection using the SPP profile.

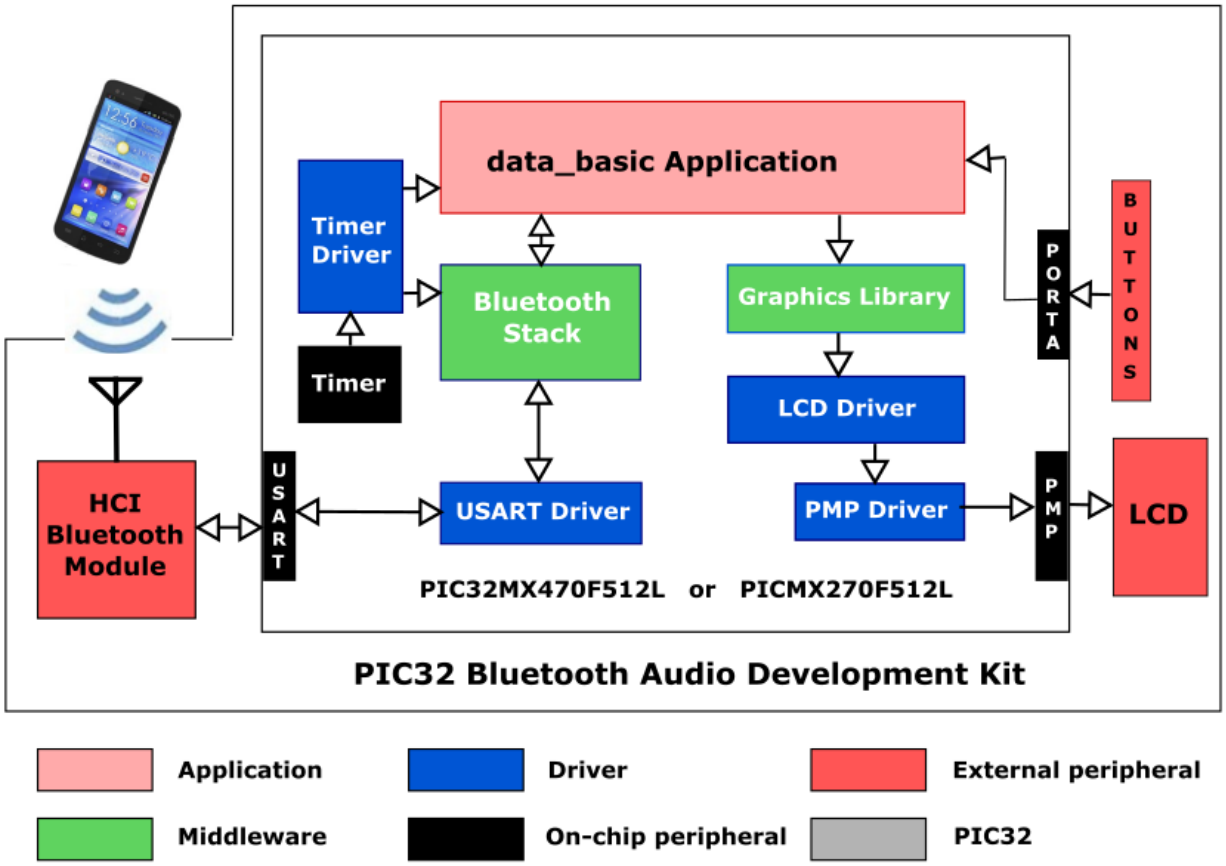
The data transfer from the smartphone to the development board is demonstrated by the user sending and receiving data from an Android

Smartphone application that performs the following actions:

- Executes a command entered on the smartphone to turn a "Virtual" LED (simulated on the display) ON or OFF
- Executes a command entered on the smartphone to change the color of a "Virtual" RGB LED (simulated on the color display) as selected from a palette
- Button presses are signalled to the smartphone application over the connection. The data received by the smartphone is displayed on its screen as ‘Button 1’, ‘Button 2’, etc., by pressing buttons on top of the board.

Note: Only an Android™ smartphone can be used with the SPP profile. For the SPP profile to be used with an Apple® iPhone®, iPad® or iPod®; the iPod, the Accessory Protocol (iAP) must be used as part of the Apple MFi Program (refer to the Apple MFi Frequently Asked Questions page for information on the MFi program by visiting: https://mfi.apple.com/faqs). Data demonstrations utilizing iAP can be obtained by contacting your local Microchip distributor as part of the MFi program. An alternative to using iAP for iPhone data is to use a BLE (Bluetooth Low Energy) profile, as is demonstrated in the BM64_ble_comm) application.

Architecture



Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Bluetooth Demonstration.

Description

To build this project, you must open the `data_basic.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/bluetooth/data/data_basic`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`data_basic.X`	`<install-dir>/apps/bluetooth/data/data_basic/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
bt_audio_dk	bt_audio_dk	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32 Bluetooth Audio Development Kit.
pic32mx270f512l_pim_bt_audio_dk	pic32mx270f512l_pim+bt_audio_dk	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32 Bluetooth Audio Development Kit using the PIC32MX270F512L PIM.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 Bluetooth Audio Development Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32 Bluetooth Audio Development Kit and PIC32MX270F512L Plug-in Module (PIM)

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the Bluetooth basic data demonstration.

Description

This demonstration performs basic SPP full-duplex data transmission.

Note: Before running the demonstration, it is necessary to install the Bluetooth SPP-pro Android application. See *Additional Bluetooth Resources > Other Android Applications* in the Demonstrations section for details.

Running the Demonstration

1. Compile and program the target device. While compiling, select the appropriate MPLAB X IDE project configuration based on the demonstration board. Refer to Building the Application for details.
2. The running of Bluetooth device demonstration is indicated when LED1 and LED2 turn ON.
3. Enable Bluetooth on your smartphone.
4. Start the Android application on your smartphone.
5. Scan for the available Bluetooth devices. The target Bluetooth device should also be displayed in the list of available devices on your smartphone.
6. The name of the target Bluetooth device will be one of the following:

Configuration	Device Name
bt_audio_dk	BTAD
pic32mx270f512l_pim_bt_audio_dk	BTAD+MX270

Note: Occasionally, the name of the Bluetooth device is not resolved and will appear as "null". After some time the name will change from "null" to the configuration specific name mentioned previously. The visible MAC Address will be fixed for the first eight digits and the last four will vary (12:34:56:78:XX: XX). If the name of the Bluetooth device is not found during the application scan, try pairing it using the *Settings > Bluetooth* menu.

7. Select the device to pair and connect.

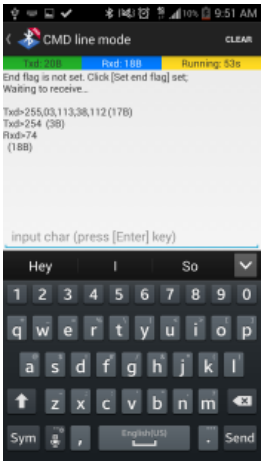


8. If the connection is successful, the message "connected to <Device Name>" appears on top of the screen.
9. Select the "CMD Line Mode" tab, enter characters and press the "Send" button. The reception of characters by the Bluetooth device is

indicated by the LEDs, ‘LED4’ and ‘LED5’, switching from OFF to ON and back to OFF. Every time data is received the Bluetooth device repeats this sequence.



9. Data can also be sent from the Bluetooth device to the connected smartphone. This can be done by pressing the switches SW1 to SW5 placed on the development board. When the switch is pressed the two LEDs, 'LED4' and 'LED5', should toggle from OFF to ON and back to OFF. The smartphone receives the data and displays "Button 1" for the SW1 switch, "Button 2" for the SW2 switch, "Button 3" for the SW3 switch, "Button 4" for the SW4 switch, and "Button 5" for the SW5 switch.

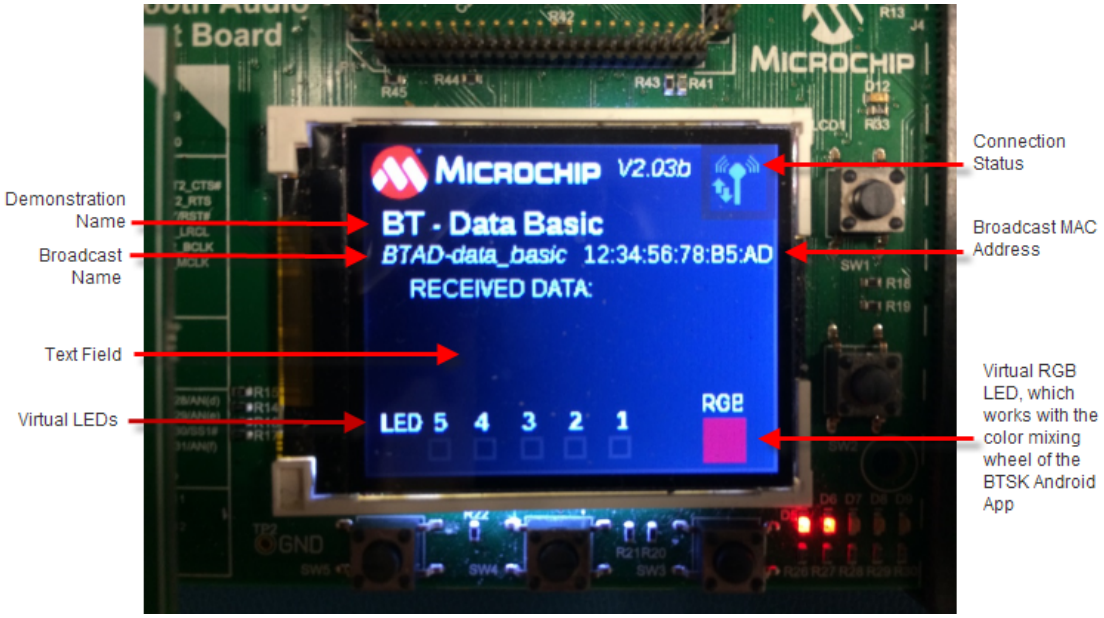


The following table describes the controls used in the supported configuration of the demonstration.

Control	bt_audio_dk
LED1	Red Color LED 'D5' on the PIC32 Bluetooth Audio Development Kit development board.
LED2	Red Color LED 'D6' on the PIC32 Bluetooth Audio Development Kit development board.
LED3	Red Color LED 'D7' on the PIC32 Bluetooth Audio Development Kit development board.
LED4	Red Color LED 'D8' on the PIC32 Bluetooth Audio Development Kit development board.
LED5	Red Color LED 'D9' on the PIC32 Bluetooth Audio Development Kit development board.
SW1	SW1 switch located on the PIC32 Bluetooth Audio Development Kit development board.
SW2	SW2 switch located on the PIC32 Bluetooth Audio Development Kit development board.
SW3	SW3 switch located on the PIC32 Bluetooth Audio Development Kit development board.
SW4	SW4 switch located on the PIC32 Bluetooth Audio Development Kit development board.
SW5	SW5 switch located on the PIC32 Bluetooth Audio Development Kit development board.

Demonstration Display

The following figure describes the display format when running the demonstration.



Connection Status

The color of the Connection Status icon on the display indicates the Bluetooth status of the demonstration, as described in the following table.

Icon Color	Display Example	Description
Gray		Bluetooth *is not* paired and connected.
White		Bluetooth *is* paired, or *is* paired but *is not* connected.
Blue		Bluetooth *is* paired *and* connected.

Demonstration Commands

Commands to control different aspects of the demonstration are listed in the following table. Note that all commands are case-sensitive and commands that are recognized by the demonstration will not display in the Text Field by default. To see all text that is transmitted, a "DISPLAY_ALL" or "DAI" command must be sent first.

Commands Sent Over Bluetooth

Command	Shortcut	Action	Example
DISPLAY_ALL	DAI	Displays all text.	DISPLAY_ALL
DISPLAY_ALL_STOP	DAS	Stop displaying all messages and will only display non-recognized commands (default).	DISPLAY_ALL_STOP
ledxon ('x' = 1-5)		Turns on the specified virtual LED (LED1 = left, LED5 = right).	led3on
ledxoff ('x' = 1-5)		Turns off the specified virtual LED (LED1 = left, LED5 = right).	led3off
ledxtoggle ('x' = 1-5)		Toggles the state of the specified virtual LED (LED1 = left, LED5 = right).	led3toggle
Lx ('x' = 1-31)		Displays a binary pattern using the virtual LEDs (MSB = left, LSB = right).	L21
255,03,R,G,B		The color of the virtual RGB LED is modified to the specified RGB value, varying from 0-254, respectively.	255,03,127,127,127

data_temp_sens_rgb

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This demonstration performs full duplex data transfer over a Bluetooth connection using the SPP profile.

The data transfer from the smartphone to the development board is demonstrated by the user sending and receiving data from an Android Smartphone application that performs the following actions:

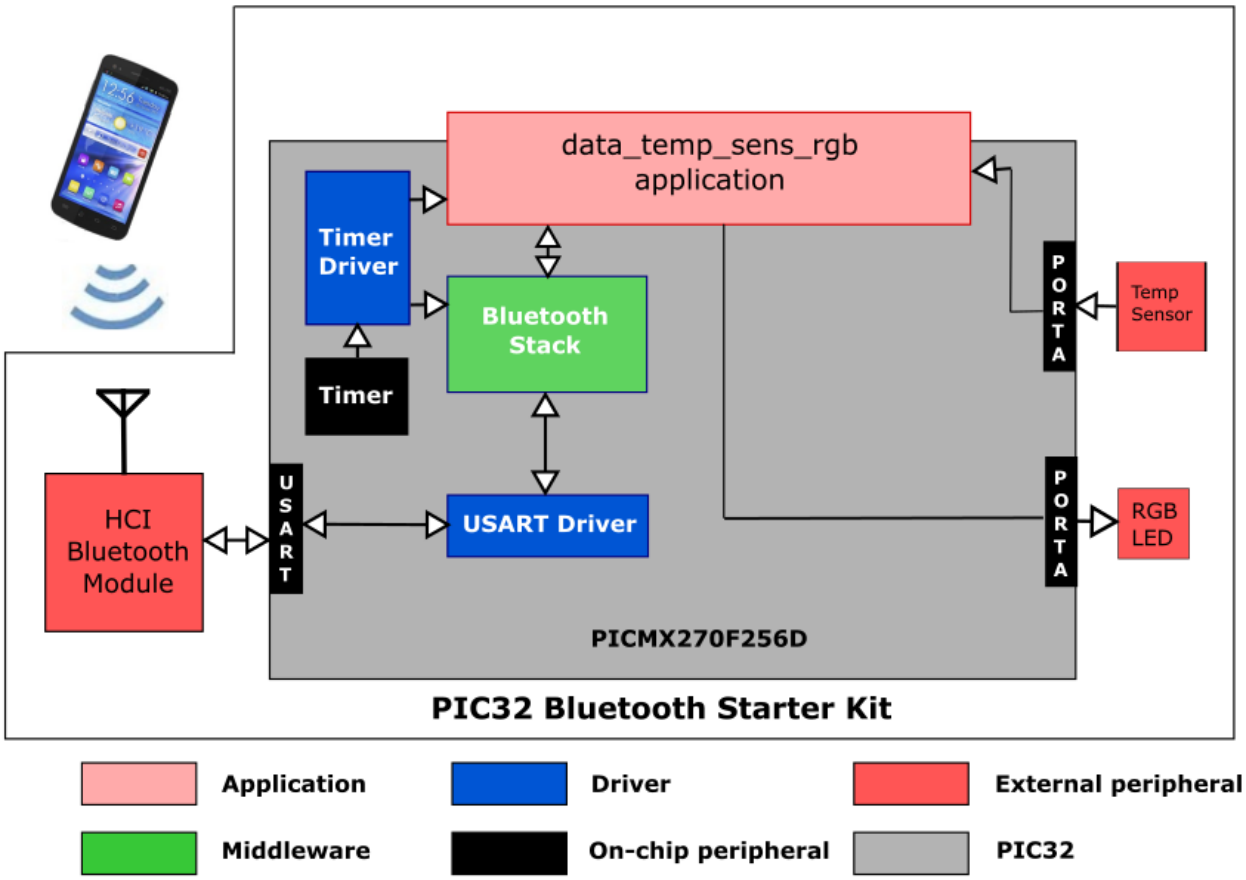
- Executes a command enter on the smartphone to change the color of a RGB LED (simulated on the color display) as selected from a palette
- Button pushes are signaled to the smartphone application over the connection. The data received by the smartphone is displayed on its screen as 'Button 1', 'Button 2', etc., by pressing buttons on top of the board.

Note: Only an Android™ smartphone can be used with the SPP profile. For the SPP profile to be used with an Apple® iPhone®, iPad® or iPod®; the iPod, the Accessory Protocol (iAP) must be used as part of the Apple MFi Program (refer to the Apple MFi Frequently Asked Questions page for information on the MFi program by visiting: https://mfi.apple.com/faqs). Data demonstrations utilizing iAP can be obtained by contacting your local Microchip distributor as part of the MFi program. An alternative to using iAP for iPhone data is to use a BLE (Bluetooth Low Energy) profile, as is demonstrated in the BM64_ble_comm) application.

Architecture

This application runs on the PIC32 Bluetooth Starter Kit, which contains a PIC32MX470F256D microcontroller with 512 KB of Flash memory and 128 KB of RAM running at 48 MHz with the following features:

- Six push buttons (SW1-SW6)
- Cree RGB LED
- Temperature Sensor/Accelerometer
- USB Device and Host interfaces
- One X32 sockets
- CSR881x HCI Bluetooth Module



Demonstration Features

- Uses the Bluetooth Stack Library SPP Profile on the PIC32 Bluetooth Starter Kit to send and receive data from an Android smartphone
- Shows how to interface and use the temperature measurement module and Cree RGB LED of the PIC32 Bluetooth Starter Kit

Tool Setup Differences

The Bluetooth Stack is configured to use the SPP profile to communication with the Android smartphone running the Microchip SPP Applications.

Timer1 is used to time the reporting of the temperature when this is enabled. A prescale value of 8 is used on the Secondary Oscillator clock.

USART2 is used for HCI protocol communication with the Bluetooth modules at 115200 baud.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Bluetooth Demonstration.

Description

To build this project, you must open the `data_temp_sens_rgb.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/bluetooth/data/data_temp_sens_rgb`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`data_temp_sens_rgb.X`	`<install-dir>/apps/bluetooth/data/data_temp_sens_rgb/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx_bt_sk	pic32mx_bt_sk	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32 Bluetooth Starter Kit.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 Bluetooth Starter Kit

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the Bluetooth temperature sensor and RGB data demonstration.

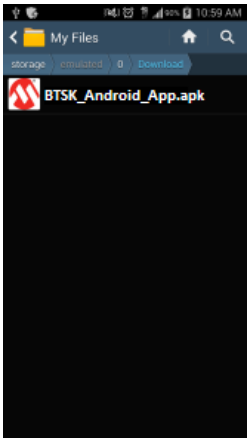
Description

This demonstration allows the SPP data transfer/receive of temperature sensor and RGB data.

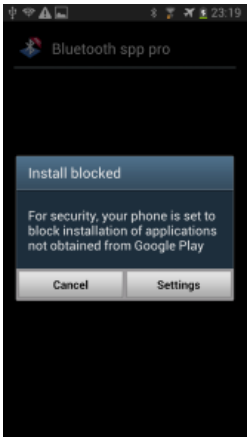
Note: Before running the demonstration, it is necessary to install the PIC32 Bluetooth Starter Kit Android application to your Android v4.0 or later smartphone.

Installing the PIC32 Bluetooth Starter Kit Android Application

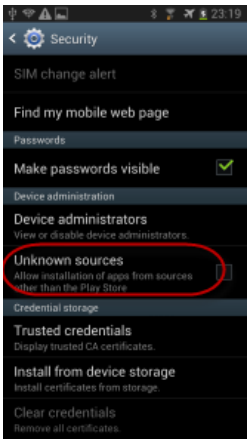
1. Install the Android application, `BTSK_Android_App.apk`, to your Android 4.0 or later smartphone. This file is available in the following MPLAB Harmony installation folder: `<install-dir>/apps/bluetooth/data_temp_sens_rgb/android_app`.
2. Connect the Android device to a computer using a mini-B USB connector.
3. It is suggested to copy the Android application into the `Download` folder of the Android device.
4. On the Android device, select *My Files>All Files>Download>BTSK_Android_App.apk*.



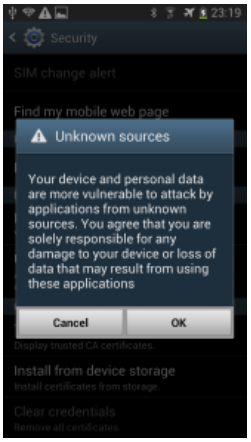
5. After selecting the `.apk` file, the warning message, "blocking installation", will appear.



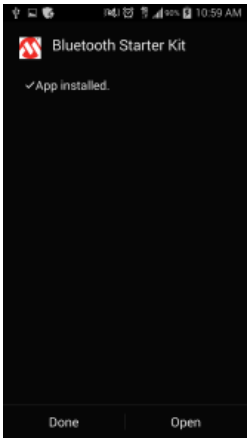
6. Select **Settings** and the security window will appear.
7. Choose the option to install applications from unknown sources.



8. Once the option is selected, a warning will appear, as shown in the following figure.

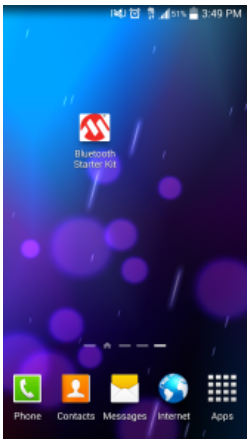


9. After selecting **OK** a window will appear requesting confirmation for installing the application.
10. Once the installation is complete, select **Open** to run the application, as shown in the following figure.

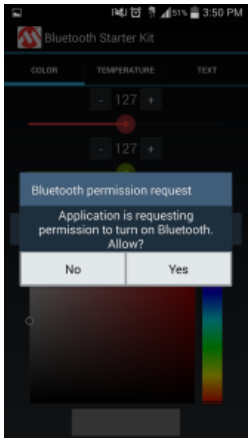


Running the Temperature Sensor and RGB Demonstration

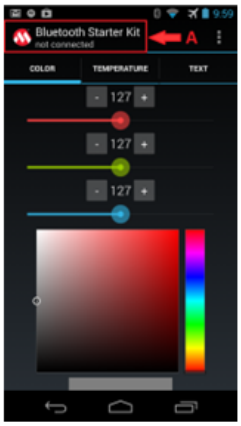
1. Compile and program the target device with the hex file, `data_temp_sens_rgb.X.production.hex`.
2. Select the PIC32 Bluetooth Starter Kit on the Android device and a window will appear, as shown in the following figure.



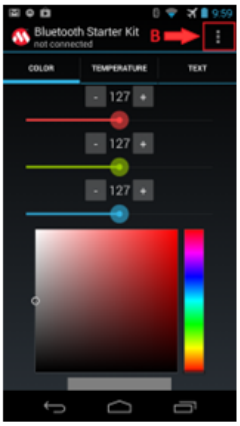
3. Select the Bluetooth Starter Kit icon in the application window.
4. If prompted, turn on Bluetooth by selecting **Yes**.



5. There are three methods for performing the next steps depending on the phone and Android version you are using. **[A]** Pressing the Microchip logo and the words "Bluetooth Starter Kit", [B] pressing the on-screen Menu button **[B]** (if supported), or **[C]** Pressing the Menu button (hardware). In general, if you have a hardware button you will not have an on-screen button and vice versa. After opening the menu, select **Connect a device - Insecure**.



Method [A]

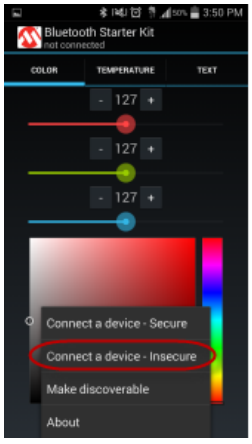


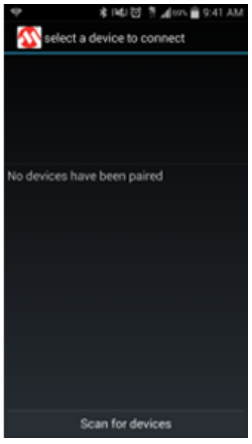
Method [B]



Method [C]

6. Select the option "Connect a device – Insecure" and a window will appear showing a list of paired device and option to scan for devices, as shown in the following figure.



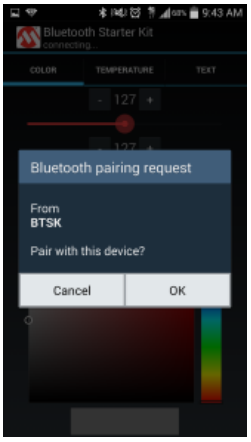


7. A list of paired devices will appear. If this is the first time connecting with the PIC32 Bluetooth Starter Kit, select **Scan for Devices**. It should be noted that sometimes the name is not resolved and will appear as "null". After some time the name will change from "null" to "BTSK". The MAC Address will be fixed for the first eight digits and the last four will vary (12:34:45:78: XX: XX) when the device is programmed with code. If the name of the Bluetooth device is not found during the application scan, try pairing it using the *Settings* > *Bluetooth* menu.

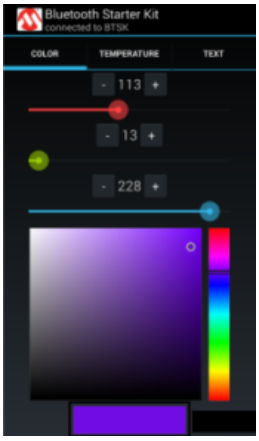




8. When selected, the Android application will pair and connect with the PIC32 Bluetooth Starter Kit. Accept any pair requests, as follows.



9. Status indicators will confirm that the connection was successful, as shown in the following figure.



Note: If the application was unable to connect, verify that the PIC32 Bluetooth Starter Kit is powered on, within range, and is not connected to another Bluetooth device.

LED Color Control

The application consists of three color sliders for Red (R), Green (G), and Blue (B), respectively. The color of the LED is programmed as the slider **(1)** is modified. The color of the LED can also be modified using the increment button or the decrement button **(2)**. Similarly, the color can be modified by selecting the color based on hue and saturation on the color palette **(3)**. The slider and the increment/decrement buttons send commands to the PIC32 Bluetooth Starter Kit via SPP to program the LED color. The colors of R, G, and B can vary from 0 to 254, respectively. The color bar **(4)** indicates the resulting modified color of the RGB combination. The LED changes in real-time to approximate the color in the color bar. The LED color will be of an uncalibrated nature of the integrated three color diodes.

This is a demonstration of full-duplex data transmission from and to the Android device to the development board. Each time the color is modified, a command is sent in a string format via SPP to the development board. The command sent can be viewed in the Text menu.



Temperature Sensor

Select the Temperature menu on the Android application and the window will appear as shown in the next figure. Fahrenheit temperature readings are shown in a graph, which updates automatically as new readings are received. The graph is zoomable and scrollable in the X direction. The last received reading is also displayed as large text.

The PIC32 Bluetooth Starter Kit has a temperature sensor and once Start is selected, the temperature will be updated for every second by default in a periodic mode. The update rate can be modified from 250 milliseconds to 8 seconds by using the increment or the decrement button **(1)**. The time duration can also be modified by the slider provided in the application **(2)**. Once the time duration is set, select **Set Timer** to initiate periodic update for the modified duration. The temperature update can be halted by selecting **Stop**. Start/Stop and Set Timer sends/receives commands to/from the PIC32 Bluetooth Starter Kit through SPP full-duplex transmission. The commands sent and received can be viewed in the Text menu.



Text Control

LED Control

The LED color can be modified by sending command via Terminal emulator of the app. Select the Text menu to view the terminal window of the application. To modify the LED color the following command is sent in the format, `255, 03, R, G, B`, where:

- `255` is the command to modify the LED color.
- `03` determines the number of Bytes to be sent, currently it is 3 (R, G, B)

- R specifies the value for Red color from 0-254
- G specifies the value for Green color from 0-254
- B specifies the value for Blue from 0-254

For example, `255, 03, 127, 60, 128` (R = 127, G = 60, B = 128)

The commands ‘r' or ‘g' or ‘b' set the LED to 50% of Red, Green, or Blue, respectively. Similarly, the commands ‘R' or ‘G' or ‘B' set the LED to 100% of Red, Green, or Blue, respectively. Refer to **Table 1: Text Commands** for details.

Temperature Control

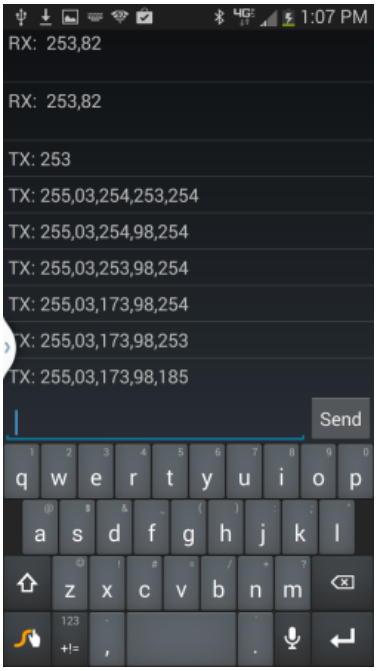
The command 254 is sent to the PIC32 Bluetooth Starter Kit via SPP data transmission. This command transmits the current temperature to the Android device terminal emulator once on request. Similarly, command 253 is sent to the PIC32 Bluetooth Starter Kit to receive the current temperature on a periodic time period for every one second by default. The rate of the update can be modified by the following command 252, with the time in milliseconds.

252 - Command the update timer rate change. The periodic temperature update can be stopped by sending command 253.

Refer to **Table 1: Text Commands** for details.

Table 1: Text Commands

Feature	Command	TX Format	RX Format	Description	Example
LED	255	255,03,R,G,B	N/A	The led color is modified depending on the value of R, G and B varying from 0-254 respectively.	255,03,127,127,127
LED	R	R	N/A	Programs LED for 100% of Red	R
LED	G	G	N/A	Programs LED for 100% of Green	G
LED	B	B	N/A	Programs LED for 100% of Blue	B
LED	r	r	N/A	Programs LED for 50% of Red	r
LED	g	g	N/A	Programs LED for 50% of Green	g
LED	b	b	N/A	Programs LED for 50% of Blue	b
Temperature	254	254	Temperature in Fahrenheit	On transmitting 254 command, the PIC32 Bluetooth Starter Kit sends the current temperature once per request	254
Temperature	253	253	253, temperature in Fahrenheit	On transmitting 253, the PIC32 Bluetooth Starter Kit updates the temperature for every one second	TX: 253 RX:253,80
Temperature	252	252, rate in milliseconds	253, temperature	On transmitting 252, the PIC32 Bluetooth Starter Kit updates the temperature every 'n' milliseconds	TX: 253,399 RX: 253,80
Temperature	253	253	N/A	The periodic display can be halted by resending 253	N/A



bt_data_voice_control

This demonstration uses Google Voice to control LEDs. There are two applications:

1. A PIC32 firmware application on the Bluetooth Audio Development Kit to receive and encode voice data.
2. An Android application for decoding voice speech data, and getting speech recognized as text from the Google Cloud engine.

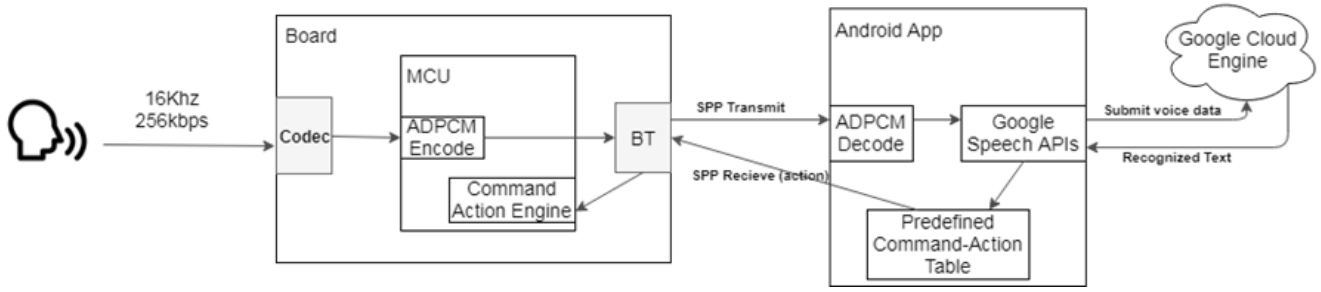
Description

In this demonstration application:

1. The users voice is received using the Codec (AK4642) microphone interface.
2. The PIC32 microcontroller compression codes the voice data, and sends the encoded data to an Android phone app via the Bluetooth SPP protocol.
3. The Android phone app receives the encoded data, and decodes the voice back to linear PCM data, which is then sent to the Google Cloud.
4. The Google Cloud Engine Voice Recognition sends recognized speech text to the Android app.
5. The Android app sends a corresponding Bluetooth SPP command to the PIC32 application over Bluetooth.

In this release, there are two voice commands supported: *light on,* and *light off*.

Architecture



Demonstration Features

- AK4642 Codec driver
- ADPCM Encoder and Decoder
- Bluetooth Classic Stack and Bluetooth SPP transmit
- Google Cloud Speech API

Building the Application

Description

To build this project, first open the `bt_data_voice_control.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is:

<install-dir>/apps/bluetooth/data/bt_data_voice_control.

To build Android application, first follow the Google Cloud Speech API Android Sample readme page to setup your credential file for android application.

Open *<install-dir>/apps/bluetooth/data/bt_data_voice_control/android_app/Bluetooth_Voice_Control_App* in Android Studio, upload your own credentials `json` file to the *app/src/main/res/raw* folder, then build the application to an android phone with internet connection.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`bt_data_voice_control.X`	*<install-dir>/apps/bluetooth/data/ bt_data_voice_control/firmware*

MPLAB X IDE Project Configurations

The following table lists and describes the supported configurations of the demonstration, which are located within *./firmware/src/system_config.*

Project Configuration Name	BSP Used	Description
bt_audio_dk	bt_audio_dk	This demonstration runs on the PIC32MX470F512L mounted on the PIC32 Bluetooth Audio Development Kit, the CSR8811 Bluetooth Module Daughter Board, and AK4642EN Codec Daughter Board.
pic32mx270f512l_pim_bt_audio_dk	pic32mx270f512l_pim+bt_audio_dk	This demonstration runs on the PIC32MX270F512L PIM mounted on the PIC32 Bluetooth Audio Development Kit, the CSR8811 Bluetooth Module Daughter Board, and AK4642EN Codec Daughter Board.

Configuring the Hardware

This section describes how to configure the supported hardware.

Description

PIC32 Bluetooth Audio Development Kit

Replace the AK4384VT DAC with the AK4642EN Codec.

PIC32 Bluetooth Audio Development Kit and PIC32MX270F512L Plug-in Module (PIM)

Replace the AK4384VT DAC with the AK4642EN Codec and Set the S1 to PIM_MCLR position.

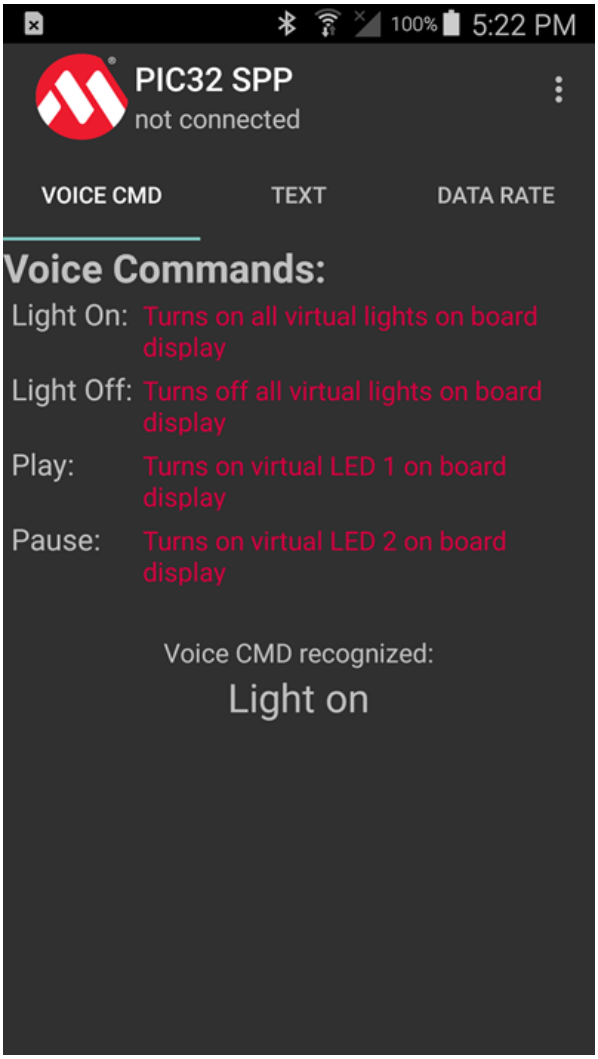
Running the Demonstration

This section provides instructions on how to build and run the Bluetooth basic data demonstration.

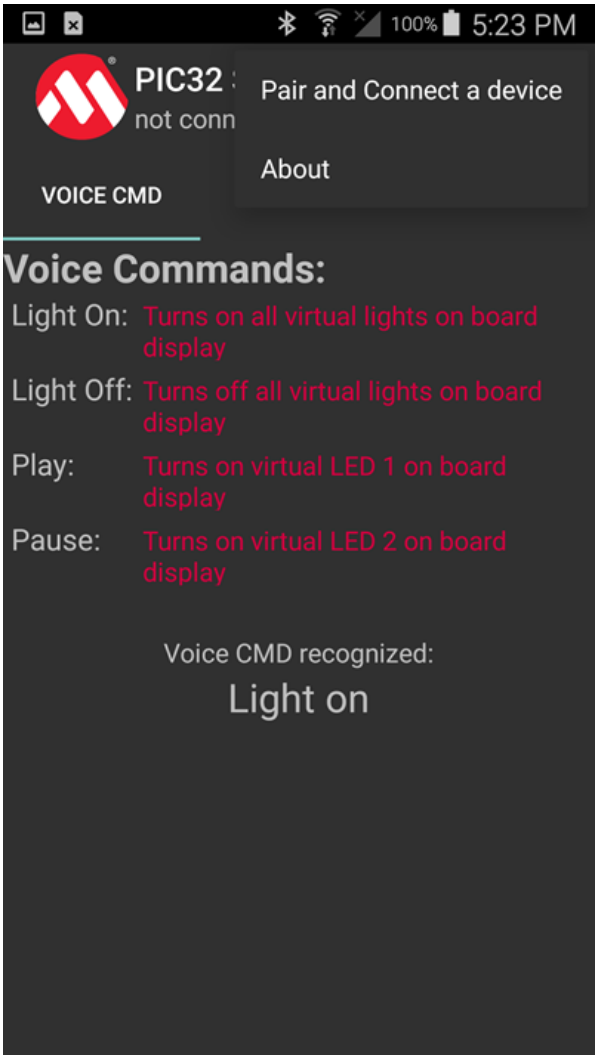
Description

This demonstration performs a voice control to the Bluetooth Audio Development Kit.

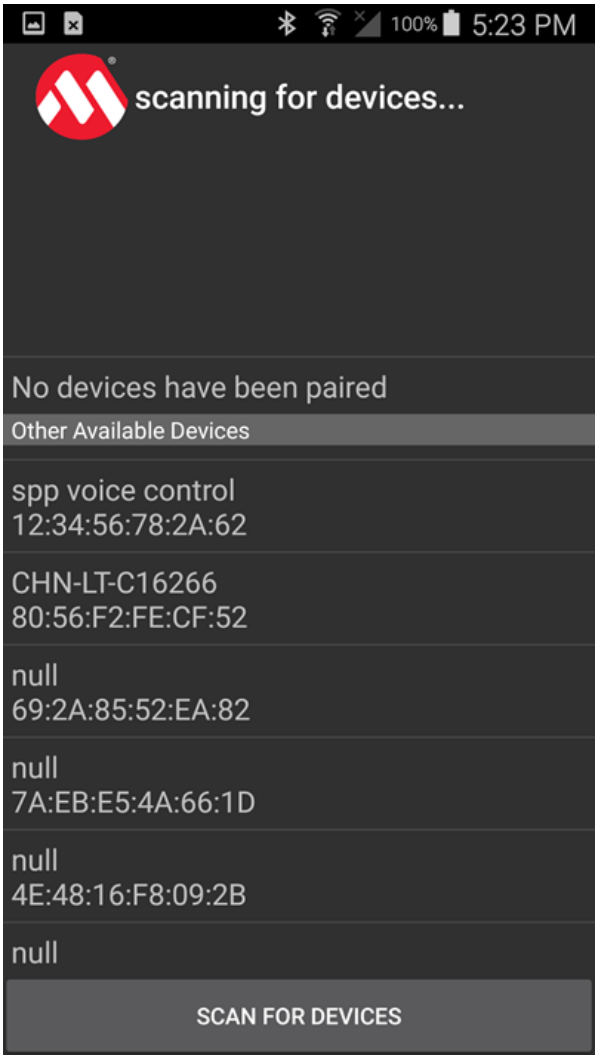
1. Compile and program the target device. While compiling, select the appropriate MPLAB X IDE project configuration based on the demonstration board. Refer to Building the Application for details.
2. The running of the Bluetooth device demonstration is indicated when D5 and D6 turn ON. Make sure the Bluetooth address shows on the display.
3. Enable Bluetooth and internet access on the device.
4. Start the Android application on the device and press the action bar at the top right.



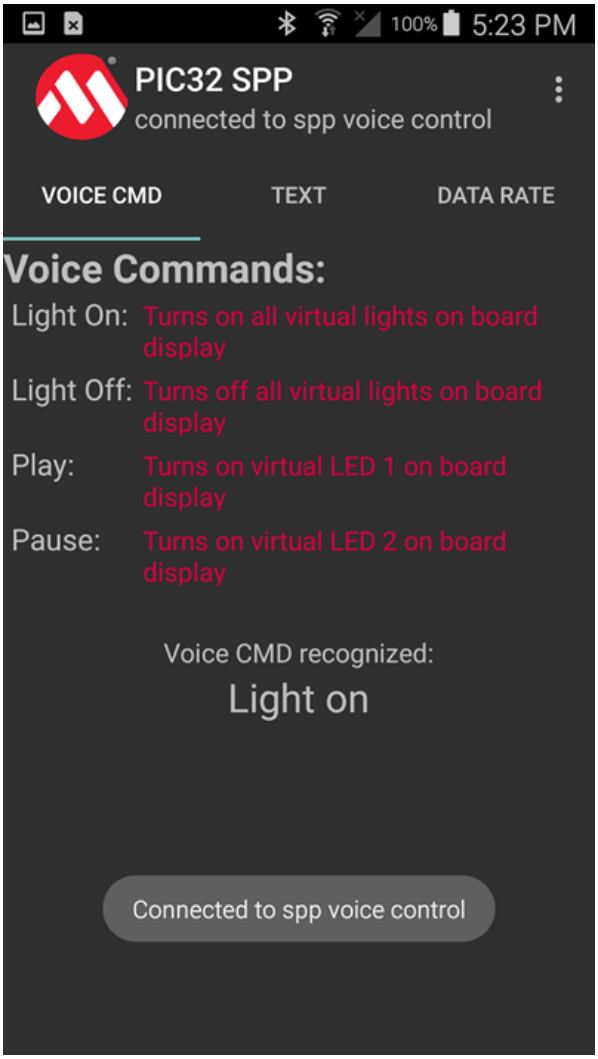
5. Select *Pair and Connect a Device* from the dropdown list when pressing the action bar.



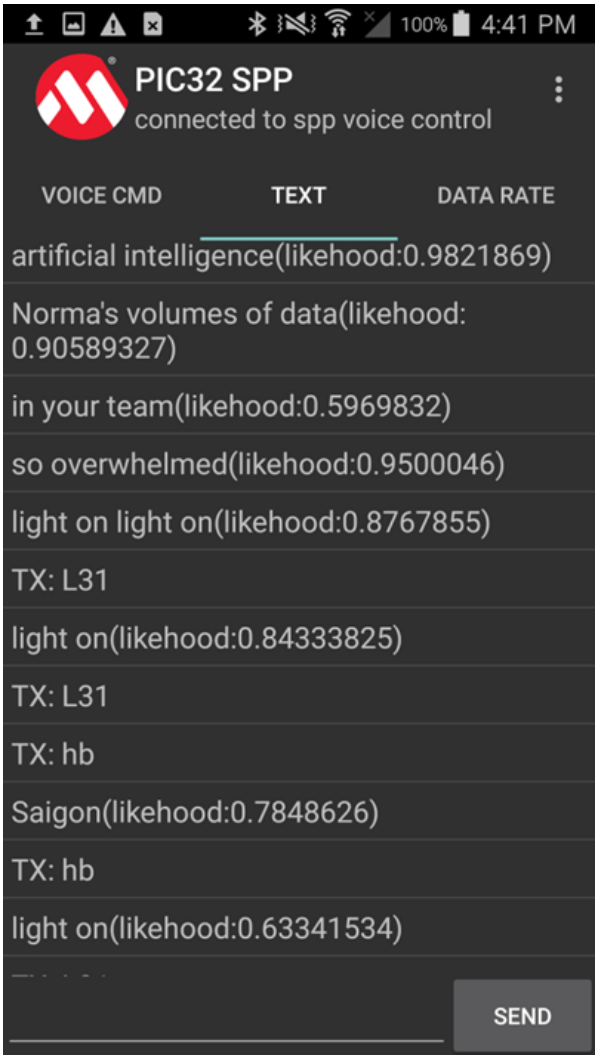
6. The name of the target Bluetooth device will be *spp voice control*.



7. Select the device to pair and connect.
8. If the connection is successful, the message, *connected to spp voice control*, will appear on the screen.



9. Navigate to the *TEXT* tab, speak any English words or sentences to the AK4642EN Microphone, and Google Cloud speech recognition starts to work if the corresponding text shows on screen.
10. Speak the *Light on* command to the microphone, and the *light on* text will show on the screen, and all virtual LEDs will be on.





11. Speak the *Light off* command to the microphone, and the *light off* text will show on the screen, and all virtual LEDs will turn off on the board display.



Premium Demonstrations

This topic provides information on how to obtain, build, configure, and run the purchased "Premium" demonstration.

Description

For information on purchasing the premium demonstration, please refer to the Microchip Premium MPLAB Harmony Audio web page (www.microchip.com/pic32harmonypremiumaudio).

a2dp_avrcp

Note: The Premium Demonstrations are not included in the standard release of MPLAB Harmony and must be purchased. Refer to the Microchip Premium MPLAB Harmony Audio web page (www.microchip.com/pic32harmonypremiumaudio) for more information.

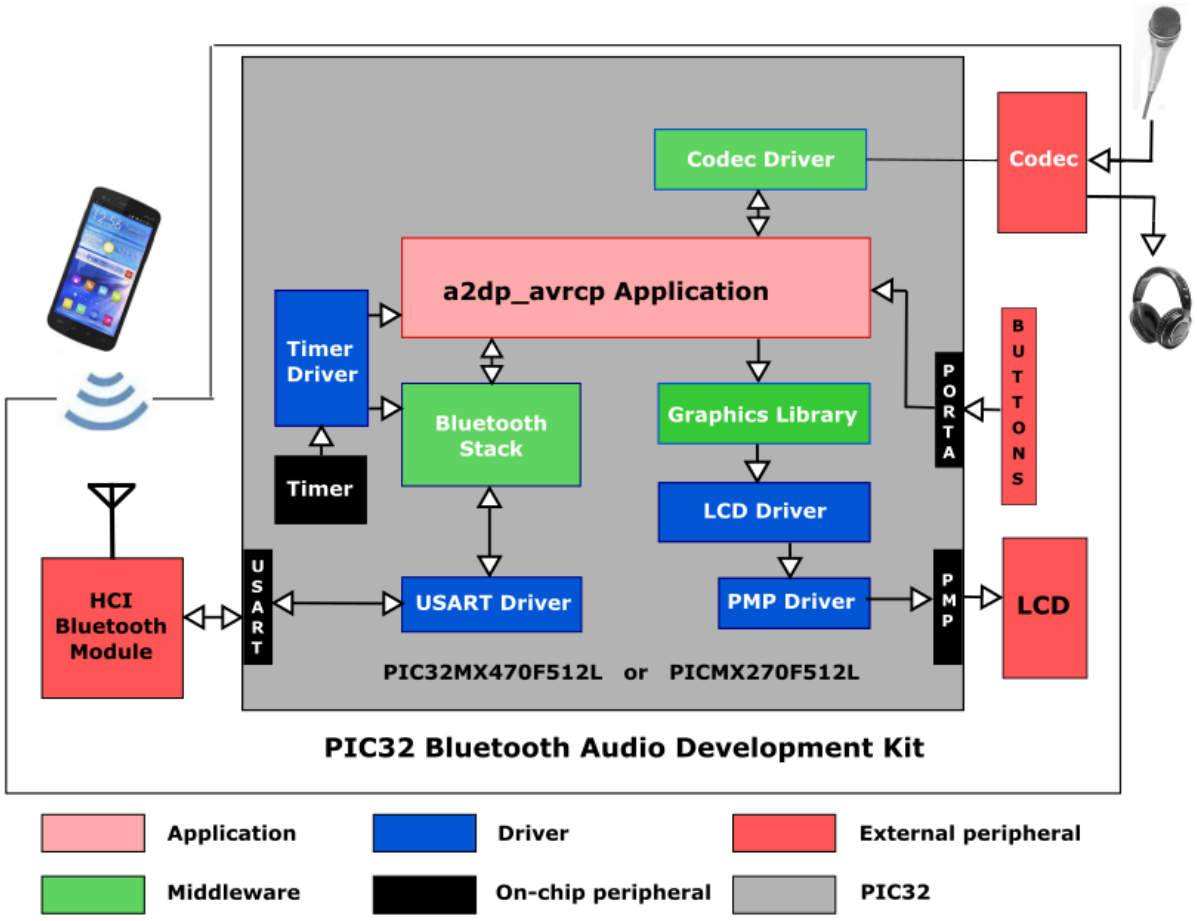
This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This demonstration performs streaming of wireless Bluetooth audio from any smartphone (i.e., Apple, Samsung, Google, etc.), personal computer, or Bluetooth-enabled device. The demonstration supports the following features:

- A2DP
- AVRCP
- SSP
- SBC Decoder

Architecture



Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Premium Demonstration.

Description

To build this project, you must open the `a2dp_avrcp.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/bluetooth/premium/audio/a2dp_avrcp`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
a2dp_avrcp.X	<install-dir>/apps/bluetooth/premium/audio/a2dp_avrcp/firmware

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
ak7755_bt_audio_dk	bt_audio_dk	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32 Bluetooth Audio Development Kit with the AKM AK7755 Codec.
bt_audio_dk	bt_audio_dk	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32 Bluetooth Audio Development Kit.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 Bluetooth Audio Development Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32 Bluetooth Audio Development Kit and AK7755 Codec

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the Premium audio demonstration

Description

1. Build and program the target device. While building, select the appropriate MPLAB X IDE project configuration based on the demonstration board. Refer to Building the Application for details.
2. When LED2 and LED3 turn ON, this indicates that the demonstration is running on the PIC32 Bluetooth Audio Development Kit.
3. Enable Bluetooth on the Bluetooth audio device (for example, Smartphone).
4. Scan for the available Bluetooth devices. The target Bluetooth device should also be displayed in the list of available devices on your Bluetooth audio device.
5. The name of the target Bluetooth device will be as follows:

Configuration	Device Name
bt_audio_dk	Microchip A2DP
ak7755_bt_audio_dk	Microchip A2DP

Note: Occasionally, the name of the Bluetooth device is not resolved and will appear as "null". After some time the name will change from "null" to the configuration-specific name previously mentioned. The visible MAC Address will be fixed for the first eight digits and the last four will vary (12:34:45:78:XX: XX).

6. Select the device to pair and connect.
7. If prompted by your device for a PIN, enter 0000.
8. If the connection is successful, the message "connected to <Device Name>" appears at the top of the screen of your Bluetooth audio device.
9. Connect a speaker or headphones to the line-out/headphone jack of the development board.
10. Select the music track and tap **Play**.

Demonstration Controls

Bluetooth Mode	Control	bt_audio_dk and ak7755_bt_audio_dk	pic32mz_da_sk_meb2
Shuffle (Toggle) / Force Bluetooth Device to Unpair	SW1	Switch, SW1, located on top of the board.	N/A

Repeat Track (Toggle) / Bluetooth Device Disconnect	SW2	Switch, SW2, located on top of the board.	N/A
Next Track/Fast Forward	SW3	Switch, SW3, located on top of the board.	N/A
Play/Pause (Toggle) / Soft Mute (toggle)	SW4	Switch, SW4, located on top of the board.	N/A
Previous Track / Rewind	SW5	Switch, SW5, located on top of the board.	N/A
N/A	LED1	Red LED, D5, located on top of the board.	Red LED, D3, located on top of the board.
Bluetooth Device Connection Ready	LED2	Red LED, D6, located on top of the board.	Red LED, D4, located on top of the board.
Bluetooth Device Connection Ready	LED3	Red LED, D7, located on top of the board.	Red LED, D5, located on top of the board.
Audio Stream Indication	LED4	Red LED, D8, located on top of the board.	Red LED, D6, located on top of the board.
N/A	LED5	Red LED, D9, located on top of the board.	Red LED, D7, located on top of the board.
CPU Exception Error	LED1-LED5	Red LEDs, D5-D9, located on top of the board.	Red LEDs, D3-D7, located on top of the board.

Bootloader Demonstrations

This section provides descriptions of the Bootloader demonstrations.

MPLAB Harmony is available for download from the Microchip website by visiting: http://www.microchip.com/mplabharmony. Once you are on the site, click the Downloads tab to access the appropriate download for your operating system. For additional information on this demonstration, refer to the “Applications Help” section in the MPLAB Harmony Help.

Introduction

Bootloader Demonstration Applications Help.

Description

This distribution package contains firmware projects that demonstrate the capabilities of the MPLAB Harmony Bootloader. This section describes the hardware requirement and procedures to run these firmware projects on Microchip demonstration and development boards.

To learn more about MPLAB Harmony stacks and libraries refer to the related documentation in Volume V: MPLAB Harmony Framework Reference.

Demonstrations

This topic provides information on how to run the Bootloader demonstration applications included in this release.

basic

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This demonstration implements a Bootloader that resides in boot Flash. With the Bootloader operating on the target device, the device can then be programmed with application code without the need for an external programmer or debugger.

The Bootloader is, operationally, similar to the bootloader described in AN1388 *"PIC32 Bootloader"*, and will work with the personal computer application provided with the related source archive file. The application note and archive file are available for download from the Microchip web site (www.microchip.com).

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Bootloader Demonstration.

Description

To build this project, you must open the `basic.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/bootloader/basic`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`basic.X`	`<install-dir>/apps/bootloader/basic/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are within `./firmware/src/system_config`.

Project Configuration Name	BSP(s) Used	Description
udp_pic32mx_eth_sk	pic32mx_eth_sk	Demonstrates the UDP Ethernet mode on the PIC32 Ethernet Starter Kit.
udp_pic32mz_ef_sk	pic32mz_ef_sk	Demonstrates the UDP Ethernet mode on the PIC32MZ EF Starter Kit
udp_pic32mz_da_sk_intddr	pic32mz_da_sk_intddr	Demonstrates the UDP Ethernet mode on the PIC32MZ DA Starter Kit.
usart_pic32mx_eth_sk	pic32mx_eth_sk	Demonstrates the UART Bootloader on the PIC32 Ethernet Starter Kit.
usart_pic32mz_ef_sk	pic32mz_ef_sk	Demonstrates the UART Bootloader on the PIC32MZ EF Starter Kit. **Note:** This demonstration does not rely on the hardware encryption module.
usart_pic32mz_da_sk_intddr	pic32mz_da_sk_intddr	Demonstrates the UART Bootloader on the PIC32MZ DA Starter Kit. **Note:** This demonstration does not rely on the hardware encryption module.
usbdevice_pic32mx_usb_sk2	pic32mx_usb_sk2	Demonstrates the USB Device mode on the PIC32 USB Starter Kit II.
usbdevice_pic32mz_ef_sk	pic32mz_ef_sk	Demonstrates the USB Device mode on the PIC32MZ EF Starter Kit.
usbhost_pic32mx_usb_sk2	pic32mx_usb_sk2	Demonstrates the USB Host Bootloader on the PIC32 USB Starter Kit II.
usbhost_pic32mz_ef_sk	pic32mz_ef_sk	Demonstrates the USB Host Bootloader on the PIC32MZ EF Starter Kit. **Note:** This demonstration does not rely on the hardware encryption module.
sdcard_pic32mz_ef_sk_meb2	pic32mz_ef_sk+meb2	Demonstrates the SD Card Bootloader on the PIC32MZ EF Starter Kit and MEB II. **Note:** This demonstration does not rely on the hardware encryption module..

Configuring the Hardware

Describes how to configure the supported hardware.

Description

The following configuration information is for UART mode.

PIC32 Ethernet Starter Kit, PIC32 Ethernet Starter Kit II, or PIC332 USB Starter Kit II

UART communication is done through the UART2 module. The U2RX and U2TX pins can be accessed through the Starter Kit I/O Expansion Board. One way to do this is with the MCP2200 Breakout Module. Connect the TX pin of the module to pin 46 on connector J11. Connect the RX pin of the module to pin 48 on connector J11. Connect GND pins together to ensure shared grounding.

Figure 1 and Figure 2 show the hardware configuration and close-ups of the jumper wire connections.

Figure 1 - Hardware Configuration

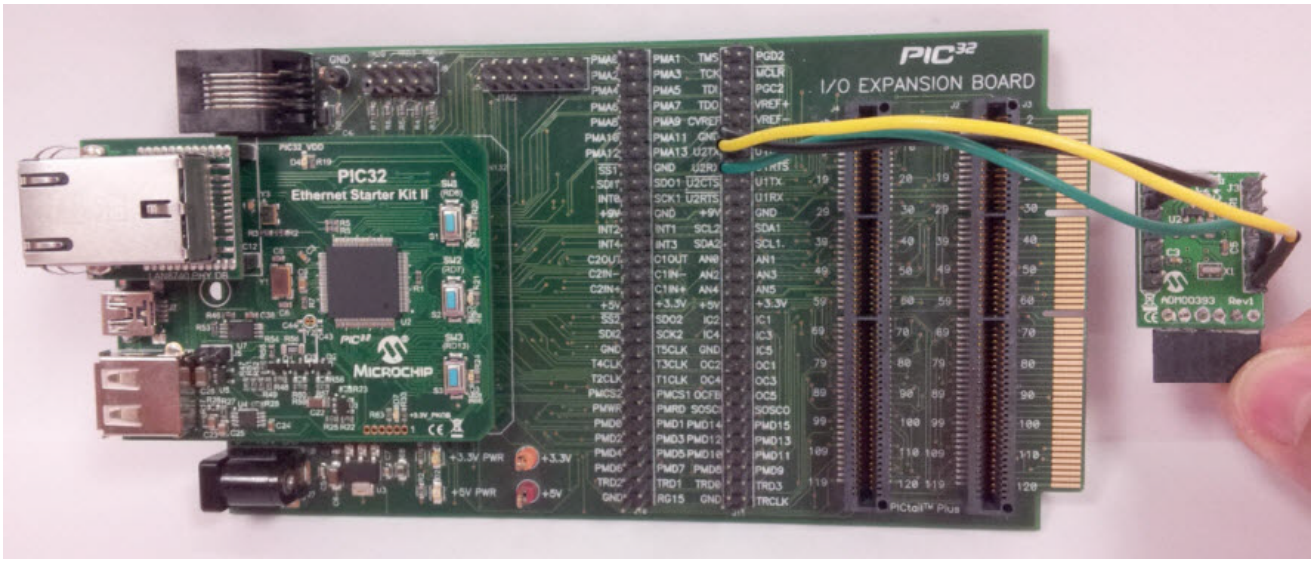
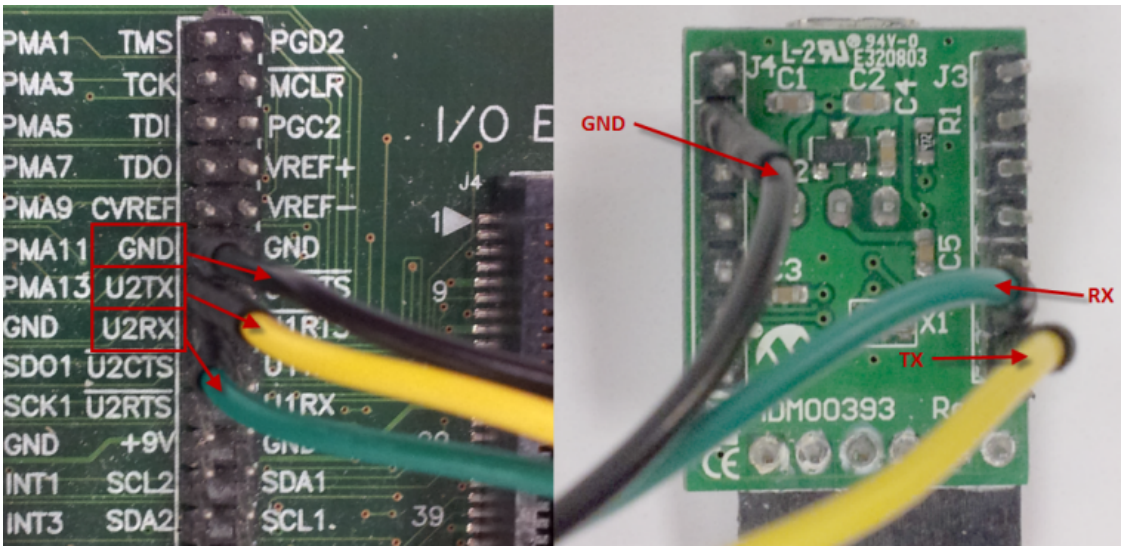


Figure 2 - Jumper Wire Connections



PIC32MZ EF Starter Kit

Connect the host computer to J11 of the PIC32MZ EF Starter Kit.

PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit

Connect the host computer to J5 of the PIC32MZ DA Starter Kit.

Running the Demonstration

Provides instructions on how to build and run the Bootloader demonstration.

Description

This demonstration requires a host application running on another computer, which will allow the UART, USB Device, and Ethernet UDP Bootloaders to communicate with it and to send commands and the new program to it. A host application is available at http://ww1.microchip.com/downloads/en/DeviceDoc/UnifiedHost-0.1.8-dist.zip, which can be used on computers running Windows®, Linux, or macOS®. These instructions will use that application to run them.

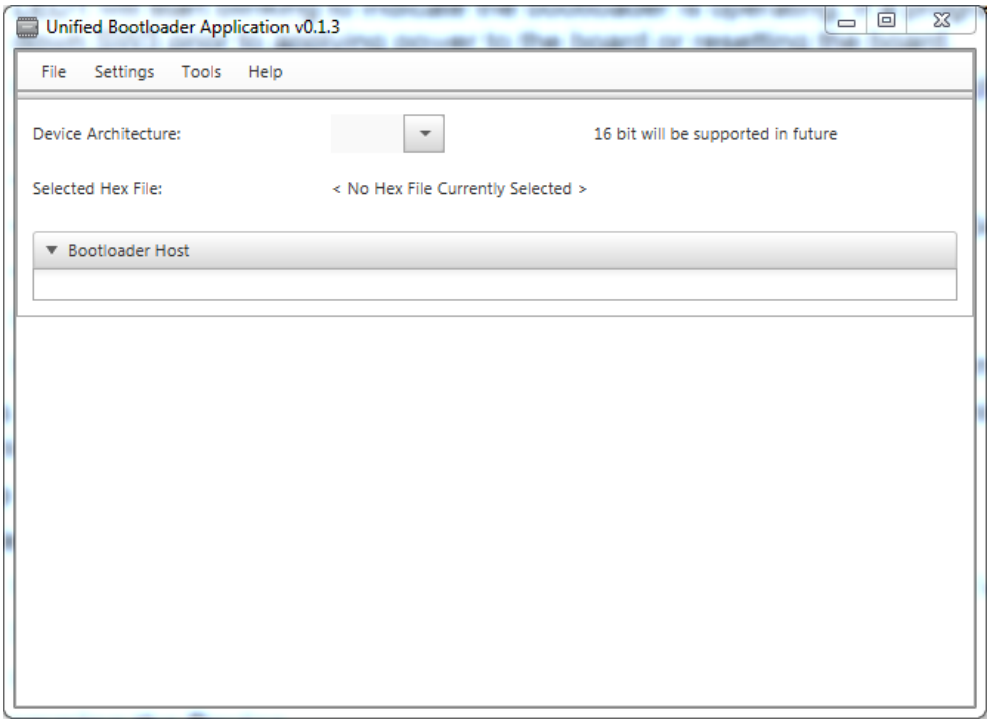
Personal Computer-based Host Demonstration

Do the following when using the configurations:

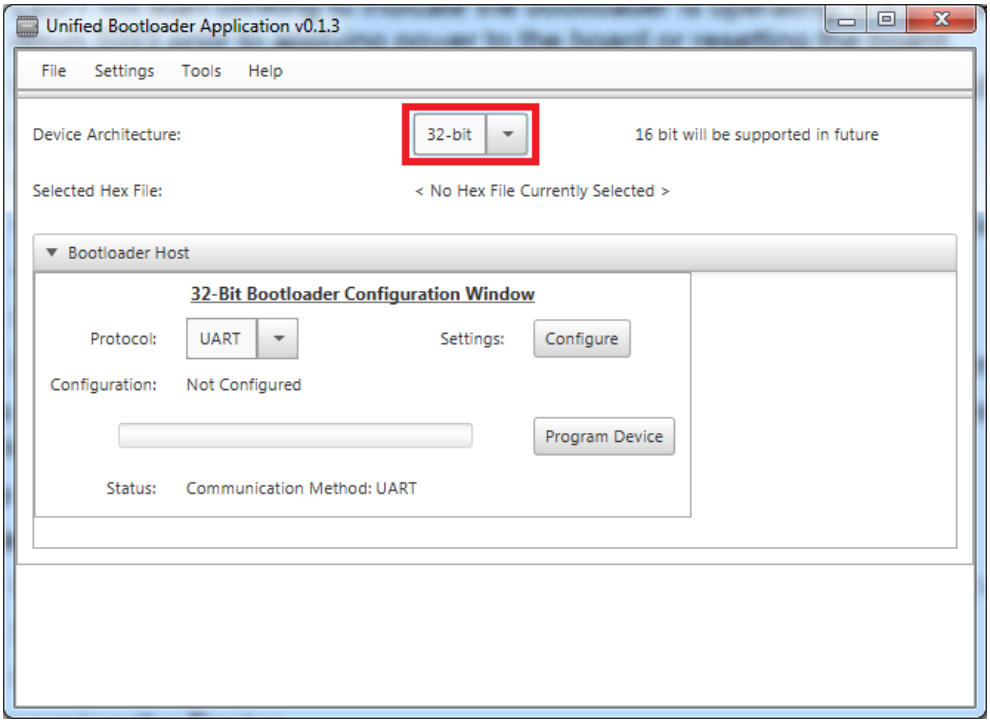
Operation

The Bootloader is operated as follows:

1. Select the configuration suitable for the target hardware.
2. Compile and program the device.
3. Press and hold SW1 on the starter kit to force Bootloader operation.
4. LED1 will start blinking to indicate the Bootloader is operating. If a program had previously been programmed, it may be necessary to press and hold SW1 prior to applying power to the board or resetting the board.
5. Open the Unified Bootloader Host application by starting the `Unified Bootloader-x.y.z.jar` file.

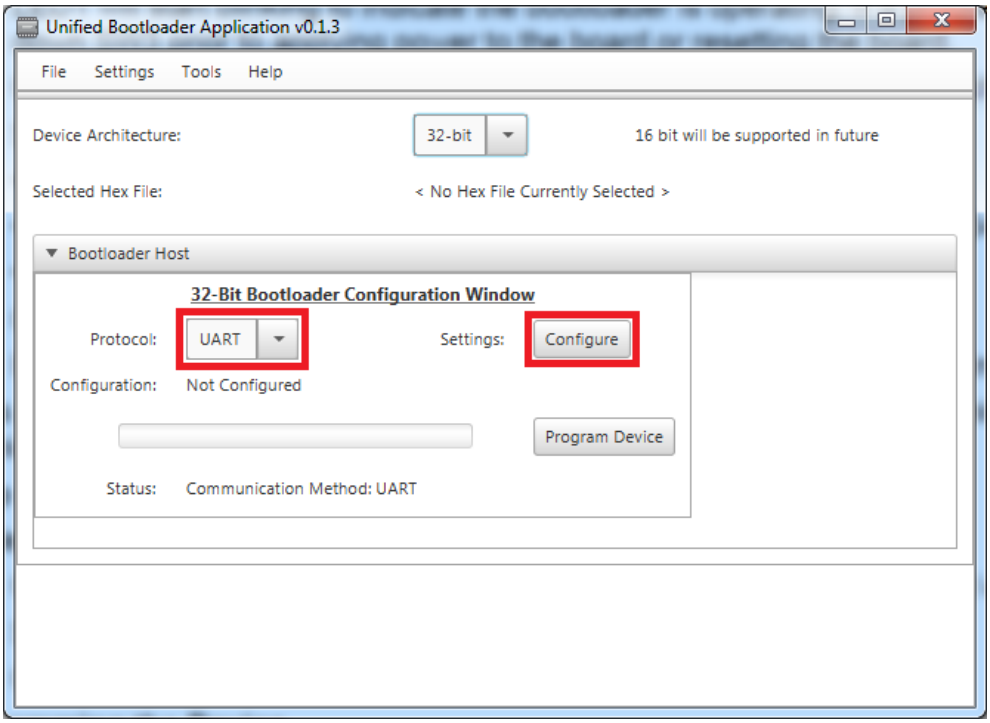


6. Select **32-bit** for the Device Architecture.

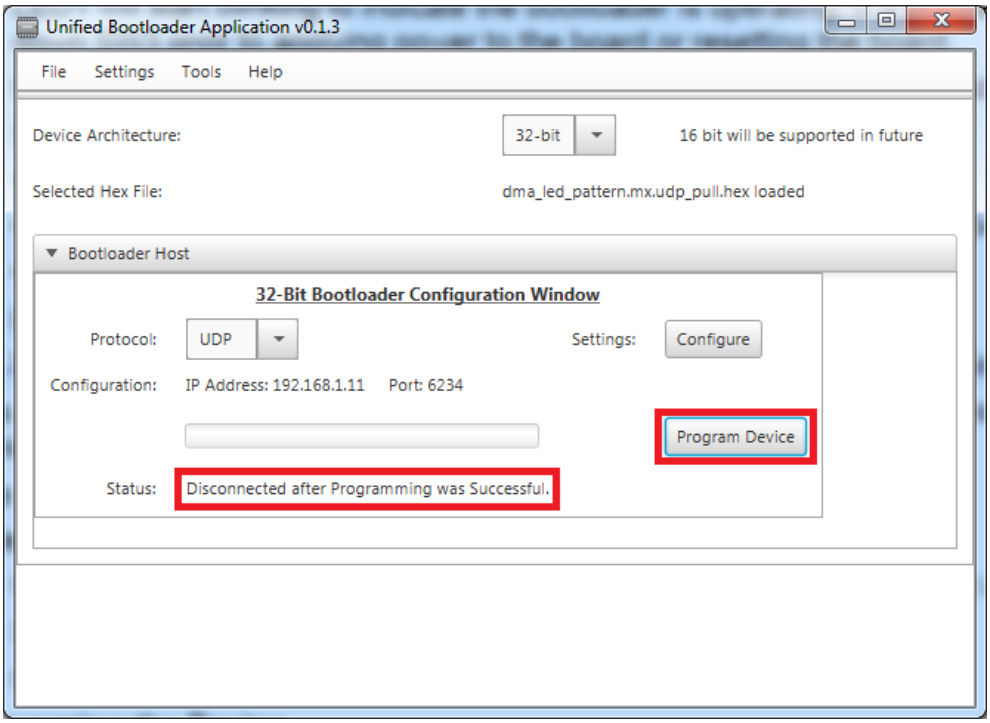


7. Select the appropriate communication path using the Protocol drop-down. Then click **Configure** to change the options. For most Bootloaders, these are the options to use:

- UART - Set the baud rate to 115,200
- UDP - Set the IP address at 192.168.1.11 and the UDP port at 6234
- Set the IP address for your host PC to 192.168.1.12 with default subnet mask
- USB Device - Set the VID at 0x4D8 and the PID at 0x03C

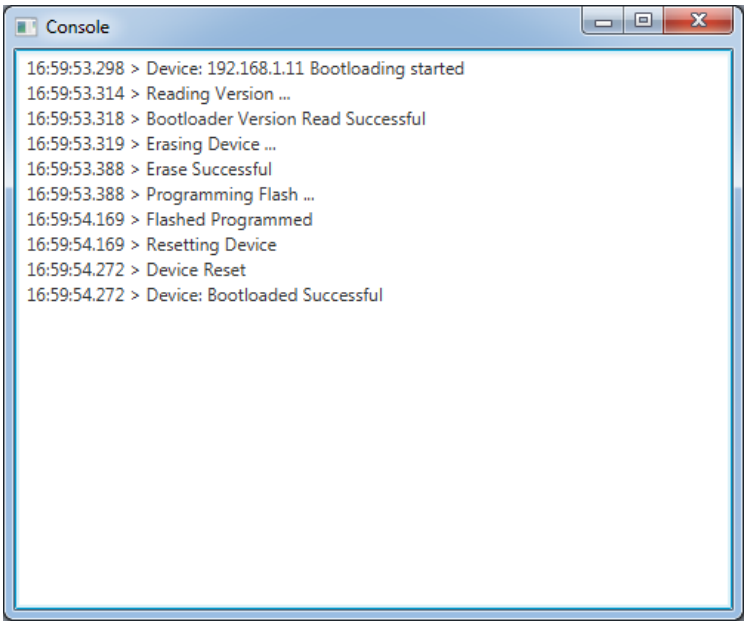


8. Select *File >Open/Load File* (`*.hex`), and navigate to the desired Hex file for the application to be bootloaded. Select the file, and click **Open**.
9. Click **Program Device**. The program combines the steps of identifying the Bootloader, erasing the device, sending down the new program, verifying the application, and starting the new program. The application will display the message "Disconnected after Programming was Successful" to indicate the programming was complete.



10. If desired, a Console window is available by selecting *Tools >Console*. The console will display messages showing the progress of the

bootloading process.



11. If the application has been programmed correctly, the green, yellow, and red LEDs will blink in an alternating pattern..

USB Host-based Demonstration

Do the following when using the configurations:

Operation

The Bootloader is operated as follows:

1. Select the configuration suitable for the target hardware.
2. Compile and program the device.
3. If a program had previously been programmed in the application space, press and hold the applicable switch for the hardware in use prior to applying power to the board or resetting the board or resetting the board to force into Booloader operation:
   - For PIC32M-based hardware, press and hold SWITCH_1
4. Depending on the hardware in use, an LED will blink to indicate the Bootloader mode:
   - For PIC32M-based hardware, LED_1

Setup

The demonstration application is prepared as follows:

1. Open the demonstration application to be Bootloaded for the hardware in use:
   - For PIC32M-based hardware, `dma_led_pattern.X` project from `<install-dir>/apps/examples/peripheral/dma/dma_led_pattern/firmware/`
2. Select the configuration suitable for the target hardware.
3. Compile the program, but do not program the device.
4. Copy the resultant `.hex` file to the Flash drive that will be inserted into the target USB port. The files are located in the paths mentioned in Step 1.
5. Rename the hex file on the Flash drive to `image.hex`.

Programming the Device

With the demonstration application compiled, the generated hex file can now be programmed into the device using the Bootloader.

To program the application into the device:

1. Insert the Flash drive into the type-A USB port on the starter kit.
2. The program will be loaded from the Flash drive and programmed into the device.
3. Remove the Flash drive when the program starts running.

SD Card-based Demonstration

Do the following when using this configuration:

Operation

The Bootloader is operated as follows:

1. Select the configuration suitable for the target hardware.

2. Compile and program the device.
3. If a program had previously been programmed in the application space, press and hold the applicable switch for the hardware in use prior to applying power to the board or resetting the board or resetting the board to force into Bootloader operation:
   - For PIC32M-based hardware, press and hold SWITCH_1
4. Depending on the hardware in use, an LED will blink to indicate the Bootloader mode:
   - For PIC32M-based hardware, LED_1

Setup

The demonstration application is prepared as follows:

1. Open the demonstration application to be Bootloaded for the hardware in use:
   - For PIC32M-based hardware, `dma_led_pattern.X project from <install-dir>/apps/examples/peripheral/dma/dma_led_pattern/firmware/`
2. Select the configuration suitable for the target hardware.
3. Compile the program, but do not program the device.
4. Copy the resultant .hex file to the Flash drive that will be inserted into the target USB port. The files are located in the paths mentioned in Step 1.
5. Rename the hex file on the SD Card drive to `image.hex`.

Programming the Device

1. To program the application into the device:
   - For PIC32M-based hardware, insert the SD Card into the slot on the MEB II. Connect the starter kit to the MEB II.
2. When power is applied, or the device is Reset, the `image.hex` file is read from the SD Card and programmed into the device.

LiveUpdate_App

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This demonstration implements a LiveUpdate application, which can be bootloaded by the LiveUpdate_Switcher application already programmed and running on the PIC32MZ EF Starter Kit.

This application can also program another LiveUpdate application into the opposite program Flash panel without affecting its application task.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Bootloader demonstration.

Description

To build this project, you must open the `LiveUpdate_App.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/bootloader/LiveUpdate_App`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`LiveUpdate_App.X`	`<install-dir>/apps/bootloader/LiveUpdate_App/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are within `./firmware/src/system_config`.

Project Configuration Name	BSP(s) Used	Description
pic32mz_ef_sk_Inst_1	pic32mz_ef_sk	Demonstrates the LiveUpate application on the PIC32MZ EF Starter Kit. **Note:** This demonstration does not rely on the hardware encryption module.
pic32mz_ef_sk_Inst_2	pic32mz_ef_sk	Demonstrates the LiveUpate application on the PIC32MZ EF Starter Kit. **Note:** This demonstration does not rely on the hardware encryption module.

pic32mz_ef_sk_Inst_3	pic32mz_ef_sk	Demonstrates the LiveUpate application on the PIC32MZ EF Starter Kit. **Note:** This demonstration does not rely on the hardware encryption module.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MZ EF Starter Kit

Connect the host computer to J11 of the PIC32MZ EF Starter Kit.

Running the Demonstration

Provides instructions on how to build and run the LiveUpdate_App demonstration.

Description

Operation

The LiveUpdate_Switcher needs to be running on the target device for the first instance to be bootloaded.

Setup

1. In MPLAB X IDE, open the `LiveUpdate_App.X` project from <install-dir>/apps/bootloader/LiveUpdate/LiveUpdate_App/firmware
2. Select the configuration suitable for the target hardware.
3. Compile the program, but do not program the device.
4. Repeat step 2 and step 3 for all three of the configurations (pic32mz_ef_sk_Inst_[1-3]).
5. Refer to step 5 through step 10 of the Running the Demonstration topic of the basic demonstration for instructions on setting up the Unified Bootloader Host application.

Programming the Device

With the `LiveUpdate_App.X` demonstration application has been compiled for all three instances, the generated hex files can now be programmed into the device using the Bootloader or the application itself (LiveUpdate feature).

1. BootLoad (first time) the `LiveUpdate_App.X\dist\pic32mz_ef_sk_Inst_1\production\LiveUpdate_App.X.production.hex` file and program the device.
2. LiveUpdate_App Instance 1 is Bootloaded into Panel 1 and LED_1 is illuminated.
3. Load the `LiveUpdate_App.X\dist\pic32mz_ef_sk_Inst_2\production\LiveUpdate_App.X.production.hex` file and program device. LED_1 should continue to be illuminated.
4. Reset the device to jump to LiveUpdate_App Instance 2 loaded in panel 2.
5. LiveUpdate_App Instance 2 is loaded in panel 2 and LED_2 is illuminated.
6. Load the `LiveUpdate_App.X\dist\pic32mz_ef_sk_Inst_3\production\LiveUpdate_App.X.production.hex` file and program device. LED_2 should continue to be illuminated.
7. Reset the device to jump to LiveUpdate_App Instance 3 loaded in panel 1.
8. LiveUpdate_App Instance 3 is loaded in panel 1 and LED_2 is illuminated.
9. Steps 1 through 8 can be repeated for the same application or different Applications, which are configured to LiveUpdate mode through the MHC (Refer to Generating the Application Linker Script in the Bootloader Library Help).

Note: If the device is reset without programming a new application, LiveUpdate_Switcher will always jump to the latest application programmed in either of the panels.

LiveUpdate_Switcher

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This demonstration implements a live update Bootloader that looks for a change during start-up that swaps the Flash panels on the PIC32MZ EF Starter Kit. The same LiveUpdate_App demonstration application as the basic Bootloader demonstration can be used as the application source for a programming example with this Bootloader demonstration.

The Bootloader is, operationally, similar to the Bootloader described in AN1388 *"PIC32 Bootloader"*, and will work with the personal computer

application provided with the related source archive file. The application note and archive file are available for download from the Microchip web site (www.microchip.com).

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Bootloader Demonstration.

Description

To build this project, you must open the `LiveUpdate_Switcher.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/bootloader/LiveUpdate_Switcher`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`LiveUpdate_Switcher.X`	`<install-dir>/apps/bootloader/LiveUpdate_Switcher/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are within `./firmware/src/system_config`.

Project Configuration Name	BSP(s) Used	Description
usart_pic32mz_ef_sk	pic32mz_ef_sk	Demonstrates the UART LiveUpate Bootloader on the PIC32MZ EF Starter Kit. **Note:** This demonstration does not rely on the hardware encryption module.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MZ EF Starter Kit

Connect the host computer to J11 of the PIC32MZ EF Starter Kit.

Running the Demonstration

Provides instructions on how to build and run the Bootloader demonstration.

Description

The Bootloader is operated as follows:

1. Select the configuration suitable for the target hardware.
2. Compile and program the device.
3. Press and hold SW1 on the starter kit to force Bootloader operation.
4. LED1 will start blinking to indicate the Bootloader is operating. If a program had previously been programmed, it may be necessary to press and hold SW1 prior to applying power to the board or resetting the board.

Class B Library Demonstrations

This section provides descriptions of the Class B Library demonstrations.

MPLAB Harmony is available for download from the Microchip website by visiting: http://www.microchip.com/mplabharmony. Once you are on the site, click the Downloads tab to access the appropriate download for your operating system. For additional information on this demonstration, refer to the “Applications Help” section in the MPLAB Harmony Help.

Introduction

Class B Library Demonstration Applications Help.

Description

This distribution package contains one Class B-related firmware project that demonstrates the capabilities of the MPLAB Harmony Class B Library. This section describes the hardware requirement and procedures to run these firmware projects on Microchip demonstration and development boards.

To learn more about MPLAB Harmony stacks and libraries refer to the related documentation in Volume V: MPLAB Harmony Framework Reference.

Demonstrations

This topic provides information on how to run the Class B Library demonstration applications included in this release.

ClassBDemo

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This application invokes each of the Class B Safety software Library interfaces one at a time and collects the responses into a single structure. This demonstrates the use of the library, as well as some of the prerequisites that must be met to use the library.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Class B Library demonstration.

Description

To build this project, you must open the `ClassBDemo.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/crypto/ClassBDemo`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`ClassBDemo.X`	`<install-dir>/apps/crypto/ClassBDemo/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are within `./firmware/src/system_config`.

Project Configuration Name	BSP(s) Used	Description
pic32mx_eth_sk2	pic32mx_eth_sk2	Demonstrates each of the Class B Safety Library functions on the PIC32 Ethernet Starter Kit II.
pic32mz_ef_sk	pic32mz_ef_sk	Demonstrates each of the Class B Safety Library functions on the PIC32MZ EF Starter Kit.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 Ethernet Starter Kit II

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the Class B Library demonstration.

Description

This demonstration tests device core components and demonstrates the use of the Class B Software Safety Library API.

1. Select the desired MPLAB X IDE project configuration:
   - pic32mz_ef_sk (for PIC32MZ EF devices)
   - pic32mx_eth_sk (for PIC32MX devices)
2. Build the selected configuration in the MPLAB X IDE project and program the demonstration board by selecting Debug Main Project from the Debug Menu. The program should build, download, and run.
3. Either single step into, or step over each test in turn. As each test completes, look at the appropriate bit of the ClassB_Test_Flags structure. They will be set with either a '1' indicating failure or a '0' indicating success.

Crypto Demonstrations

This section provides descriptions of the Crypto demonstrations.

MPLAB Harmony is available for download from the Microchip website by visiting: http://www.microchip.com/mplabharmony. Once you are on the site, click the Downloads tab to access the appropriate download for your operating system. For additional information on this demonstration, refer to the “Applications Help” section in the MPLAB Harmony Help.

Introduction

Crypto Library Demonstration Applications Help.

Description

This distribution package contains three Crypto-related firmware projects that demonstrate the capabilities of the MPLAB Harmony Crypto Library. This section describes the hardware requirement and procedures to run these firmware projects on Microchip demonstration and development boards.

To learn more about MPLAB Harmony stacks and libraries refer to the related documentation in Volume V: MPLAB Harmony Framework Reference.

Demonstrations

This topic provides information on how to run the Crypto Library demonstration applications included in this release.

encrypt_decrypt

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This demonstration exercises several cryptographic functions, including MD5, TDES, DES, AES, RSA, ECC, and Random Number Generation, to verify that the software or hardware is performing correctly. While the demonstration is running, the yellow LED on the starter kit will light to indicate processing. If all functions execute successfully, the green LED on the starter kit will illuminate.

When testing hardware encryption, the Starter Kit with Crypto Engine (DM320006-C) must be used. Software encryption can be performed on either PIC32MX795F512L or any version of PIC32MZ device.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Crypto Demonstration.

Description

To build this project, you must open the `encrypt_decrypt.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/crypto/encrypt_decrypt`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
encrypt_decrypt.X	<install-dir>/apps/crypto/encrypt_decrypt/firmware

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are within ./firmware/src/system_config.

Project Configuration Name	BSP(s) Used	Description
pic32mx_eth_sk	pic32mx_eth_sk	Demonstrates encryption, decryption, hashing, and random number generation using Software Libraries on the PIC32 Ethernet Starter Kit.
pic32mz_ec_sk_hw	pic32mz_ec_sk	Demonstrates encryption, decryption, hashing, and random number generation using the Hardware Encryption module on the PIC32MZ Embedded Connectivity (EC) Starter Kit with Crypto.
pic32mz_ef_sk_hw	pic32mz_ef_sk	Demonstrates encryption, decryption, hashing, and random number generation using the Hardware Encryption module on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit with Crypto.
pic32mz_ec_sk_sw	pic32mz_ec_sk	Demonstrates encryption, decryption, hashing, and random number generation using Software Libraries on the PIC32MZ Embedded Connectivity (EC) Starter Kit.
pic32mz_ef_sk_sw	pic32mz_ef_sk	Demonstrates encryption, decryption, hashing, and random number generation using Software Libraries on the PIC32MZ Embedded Connectivity with Floating Point Unit (EC) Starter Kit.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 Ethernet Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ EC Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the Crypto demonstration.

Description

This demonstration exercises various encryption, decryption, hashing, and random number functions.

1. First compile and program the target device. While compiling, select the configuration for the hardware in use.
2. Observe the status of LEDs on the starter kit. The yellow LED will be illuminated while the demonstration executes. If all function passes succeed, the green LED will illuminate. If an error occurred, the red LED is illuminated.

large_hash

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This application demonstrates how to execute hashes on large blocks of data. In this case, the demonstration performs MD5, SHA-1, SHA-256, SHA-384, and SHA-512 hashing on a 512 * 1024 block of the letter 'a'.

On PIC32MZ devices, which have adequate Flash memory, the linker script is configured to create the 512 * 1024 block starting at physical

address 0x9D08_0000.

The application runs the hashes in two ways:

- On PIC32MZ devices, the first way it runs it is by passing the entire 512 * 1024 block in one function call.
- With the second way, which is the only one that runs on PIC32MX devices, it passes a 1024 block of the letter 'a' that is allocated on the stack to the engine, doing it 512 times.

After the hashing has been performed, the application outputs via the system console the results of the hashing, and the time it took to perform each form. It then compares the generated hashes with known values for each algorithm. If all tests pass, the green LED is lit, and a message is presented through the system console. If any tests fail, the red LED is lit, and a corresponding message is presented through the system console.

When testing hardware encryption, the PIC32MZ EC Starter Kit configured with the Crypto Engine (DM320006-C) must be used. Software encryption can be performed on any version of a PIC32MX or PIC32MZ device.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Crypto Demonstration.

Description

To build this project, you must open the `large_hash.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/crypto/large_hash`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`large_hash.X`	`<install-dir>/apps/crypto/large_hash/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are within `./firmware/src/system_config`.

Project Configuration Name	BSP(s) Used	Description
pic32mx_eth_sk	pic32mx_eth_sk	Demonstrates hashing large blocks of data using Software Libraries on the PIC32 Ethernet Starter Kit.
pic32mz_ec_sk_hw	pic32mz_ec_sk	Demonstrates hashing large blocks of data using the Hardware Encryption module on the PIC32MZ Embedded Connectivity (EC) Starter Kit with Crypto.
pic32mz_ef_sk_hw	pic32mz_ef_sk	Demonstrates hashing large blocks of data using the Hardware Encryption module on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit with Crypto.
pic32mz_ec_sk_sw	pic32mz_ec_sk	Demonstrates hashing large blocks of data using Software Libraries on the PIC32MZ Embedded Connectivity (EC) Starter Kit.
pic32mz_ef_sk_sw	pic32mz_ef_sk	Demonstrates hashing large blocks of data using Software Libraries on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 Ethernet Starter Kit

UART communication is done through the UART2 module. The U2RX and U2TX pins can be accessed through the Starter Kit I/O Expansion Board. One way to do this is with the MCP2200 Breakout Module. Connect the TX pin of the module to pin 46 on connector J11. Connect the RX pin of the module to pin 48 on connector J11. Connect GND pins together to ensure shared grounding.

Figure 1 and Figure 2 show the hardware configuration and close-ups of the jumper wire connections.

Figure 1 - Hardware Configuration

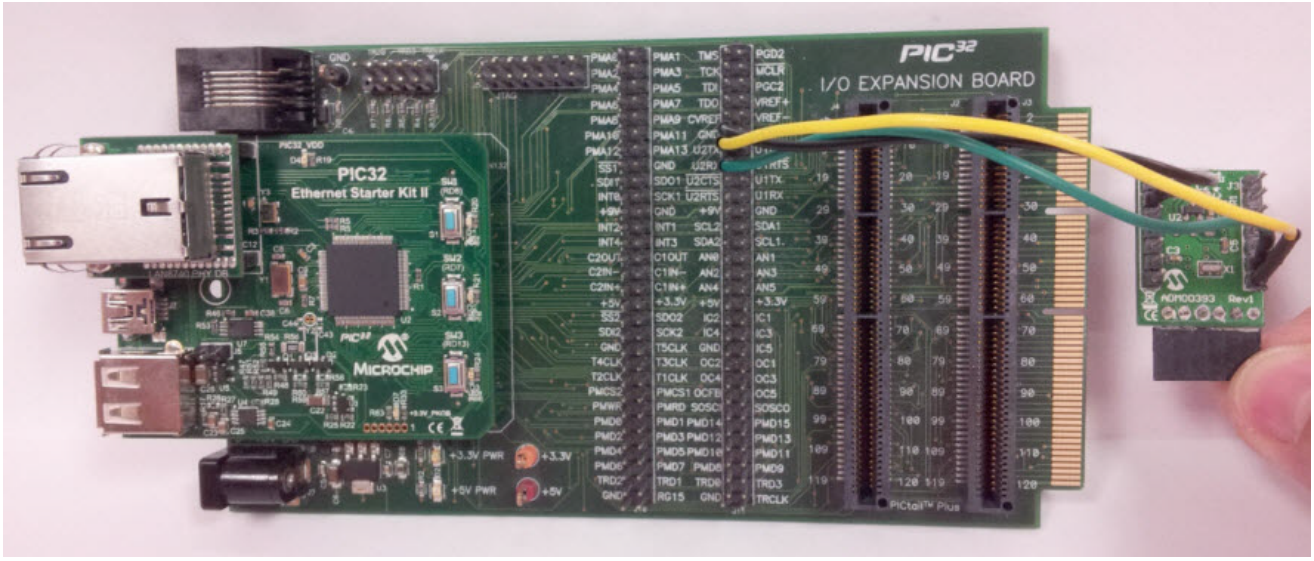
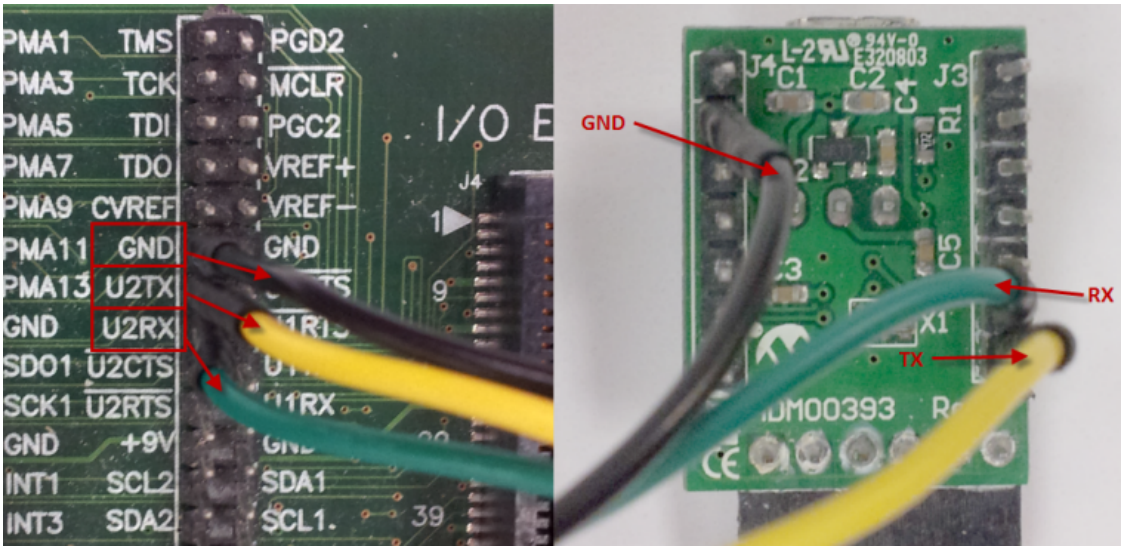


Figure 2 - Jumper Wire Connections



PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ Embedded Connectivity (EC) Starter Kit

UART communication is done through the UART1 module, routed through PPS to RPF0 and RPF1. The U1RX and U1TX pins can be accessed through the PIC32MZ Starter Kit Adapter Board. One way to do this is with the MCP2200 Breakout Module. Connect the TX pin of the module to the EBID10 pin of JP3 on the underside of the adapter. Connect the RX pin of the module to the EBID11 pin of JP1 on the underside of the adapter. Connect grounds between the module and the starter kit to ensure shared grounding.

Figure 3 and Figure 4 show the hardware configuration.

Figure 3 - Hardware Configuration (Front)

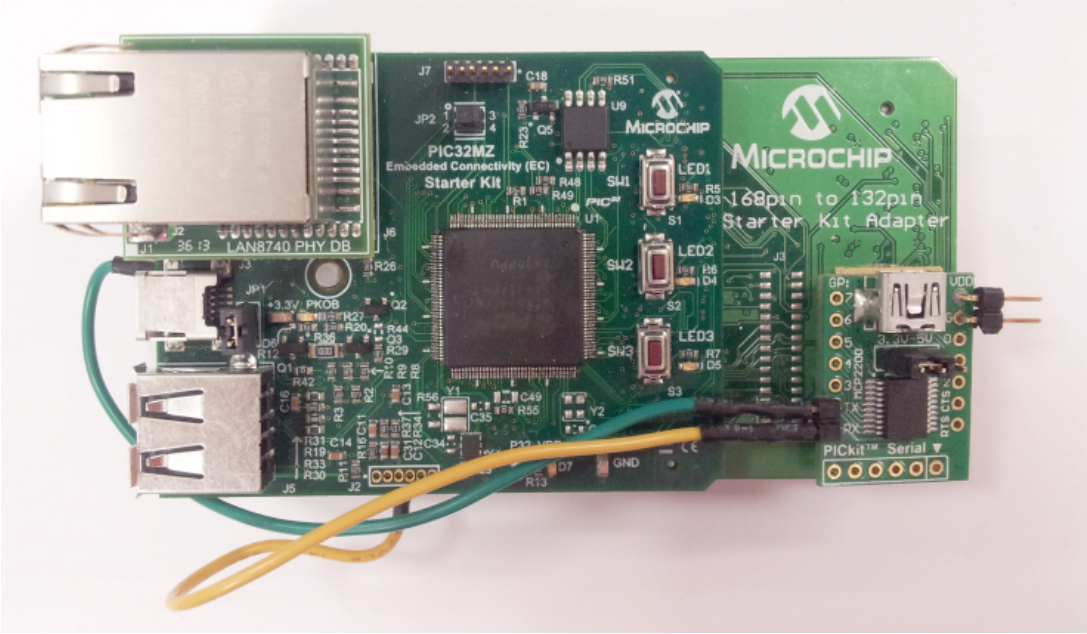
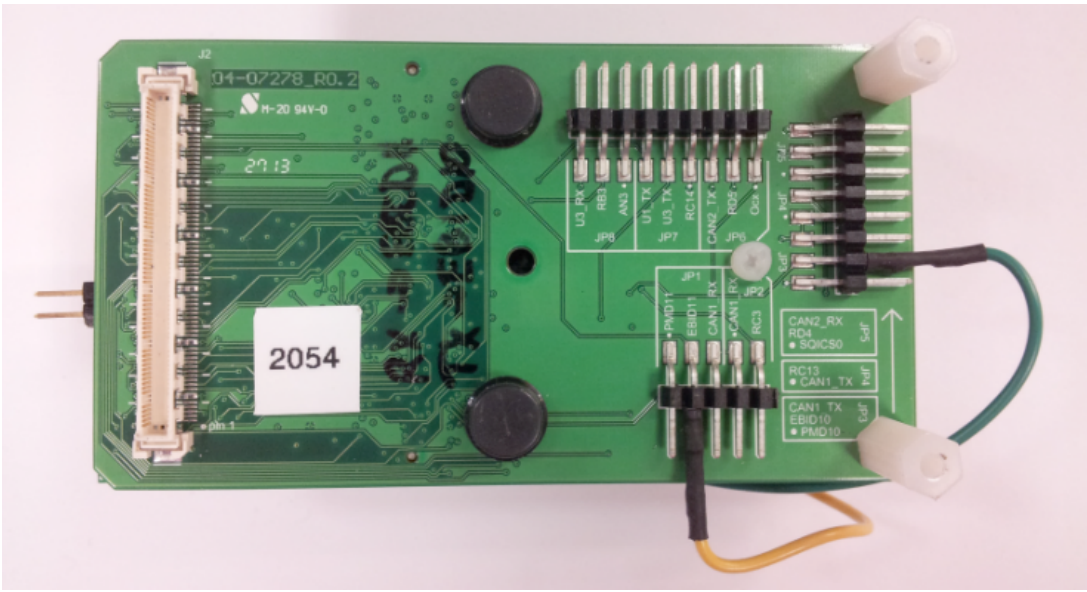


Figure 4 - Hardware Configuration (Back)



Running the Demonstration

Provides instructions on how to build and run the Crypto demonstration.

Description

This demonstration exercises hashing functions on large blocks of data.

1. First, compile and program the target device. While compiling, select the configuration suitable for hardware.
2. Open a serial terminal program, such as PuTTY or TeraTerm, and connect it to the serial port for the MCP2200 Breakout Module. The serial configuration is 115200 baud, 8 data bits, no parity bit, 1 stop bit.
3. Observe the status of LEDs on the starter kit. The yellow LED will be illuminated while the demonstration executes. If all function passes succeed, the green LED will illuminate. If an error occurred, the red LED is illuminated.
4. Observe the output of the program in the serial terminal program. It will report the results of the hashes, and the cycles taken to execute. The actual cycles taken will depend on the hardware used, and the size of the buffers available to the hardware engine. The following example shows the output using the PIC32MZ EC Starter Kit configured with the Crypto Engine:

```
Starting the test.

MD5 from Flash: 30C2557E8302A5BEB290C71520D87F42 took 481405 clock cycles
MD5 from feed: 30C2557E8302A5BEB290C71520D87F42 took 804934 clock cycles
SHA from Flash: F7FEC128D7FCD59222BA37368D3B7210D4C7B6EF took 481151 clock cycles
```

```
SHA from feed: F7FEC128D7FCD59222BA37368D3B7210D4C7B6EF took 804901 clock cycles
SHA256 from Flash: 85A84A75886E8A526DBEC4E16E3375FAA307B4AEAD79C9ED3264C0477A6F6EBA took 702033 clock cycles
SHA256 from feed: 85A84A75886E8A526DBEC4E16E3375FAA307B4AEAD79C9ED3264C0477A6F6EBA took 806480 clock cycles
SHA384 from Flash:
A550561A6330048EFE826A97E5FED843FA1CE646A9BF546CCB433C2FCB0E54821C4C945EED9A592B5BF43157E212F277 took
45452328 clock cycles
SHA384 from feed:
A550561A6330048EFE826A97E5FED843FA1CE646A9BF546CCB433C2FCB0E54821C4C945EED9A592B5BF43157E212F277 took
45391039 clock cycles
SHA512 from Flash:
7F49157FB359B39EA6DA934DC9A10709FEDF8846D139D0E637A3C0FC833B6F42703858DBACEE28F4489B5E95FAB5E5655A25F838B0DC
7BF3C84C7CC0264F6A4F
took 45807606 clock cycles
SHA512 from feed:
7F49157FB359B39EA6DA934DC9A10709FEDF8846D139D0E637A3C0FC833B6F42703858DBACEE28F4489B5E95FAB5E5655A25F838B0DC
7BF3C84C7CC0264F6A4F
took 45775518 clock cycles
All tests passed.
```

ecc_asymmetric

This application demonstrates the authentication of a remote device with a host (The Curiosity PIC32MZ EF Development Board and the secure 4 click board using the cryptography module ATECC608A) by using an asymmetric authentication method, where the host verifies the signature from the remote through the public key of the remote.

The application allows adding information to the configuration, using the configuration data, and key data to configure a secure 4 click board. The application flow is realized through an interactive user interface through the serial terminal program (Tera term) interfaced through the USB of a computer.

For more information on the features and layout of the Curiosity PIC32MZ EF Development Board, refer to the PIC32MZ EF Curiosity Development Board User's Guide.

For more information on the features of the ATECC608A module, refer to the Product Data Sheet.

Description

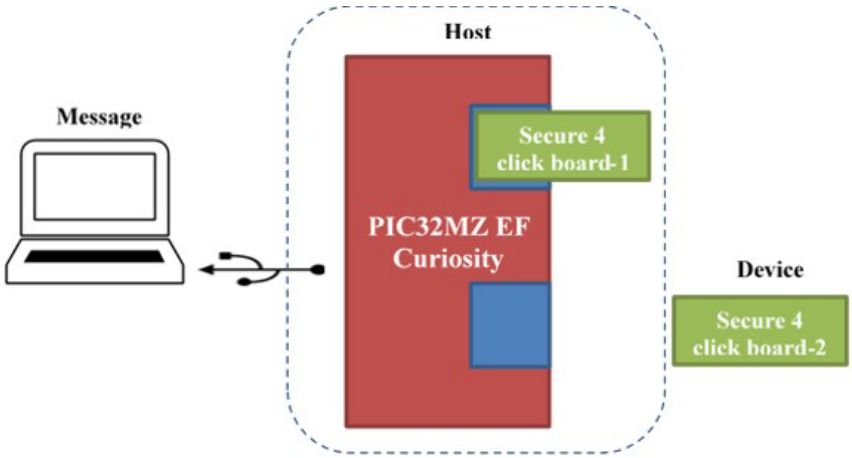
This application demonstrates the use of the ATECC608A module to authenticate and verify if the device is secure or not. The authentication method used is Asymmetric.

In asymmetric authentication, a verifier checks the authenticity of remote by validating the signature.

Asymmetric authentication is based on the use of two keys. One of the keys needs to be kept secret. This key is called the Private Key. The second key is mathematically related to the Private Key and is called the Public Key. The public key is openly shared. The key owner will use the Public Key to authenticate the signature.

In this application, a secure hardware key storage device (ATECC608A on a Secure 4 click board) is used to generate a signature by the remote, and the host uses the public key of the remote, and verifies the signature.

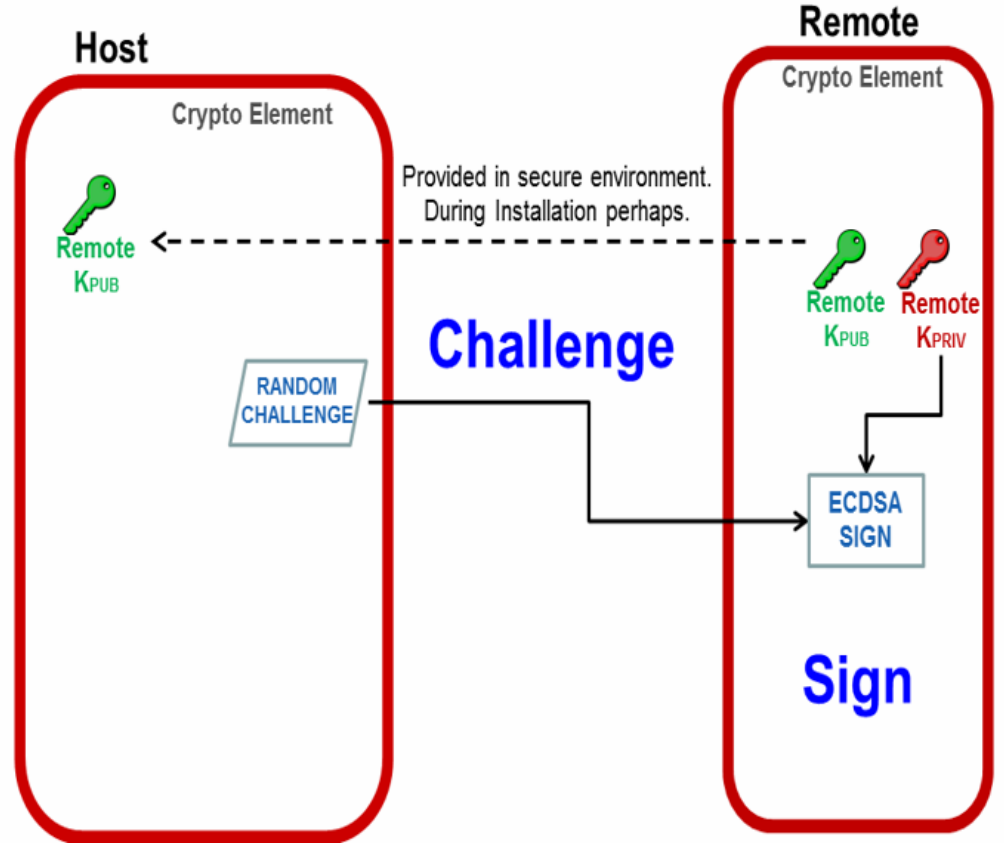
The following figure represents the functional block diagram of the application.

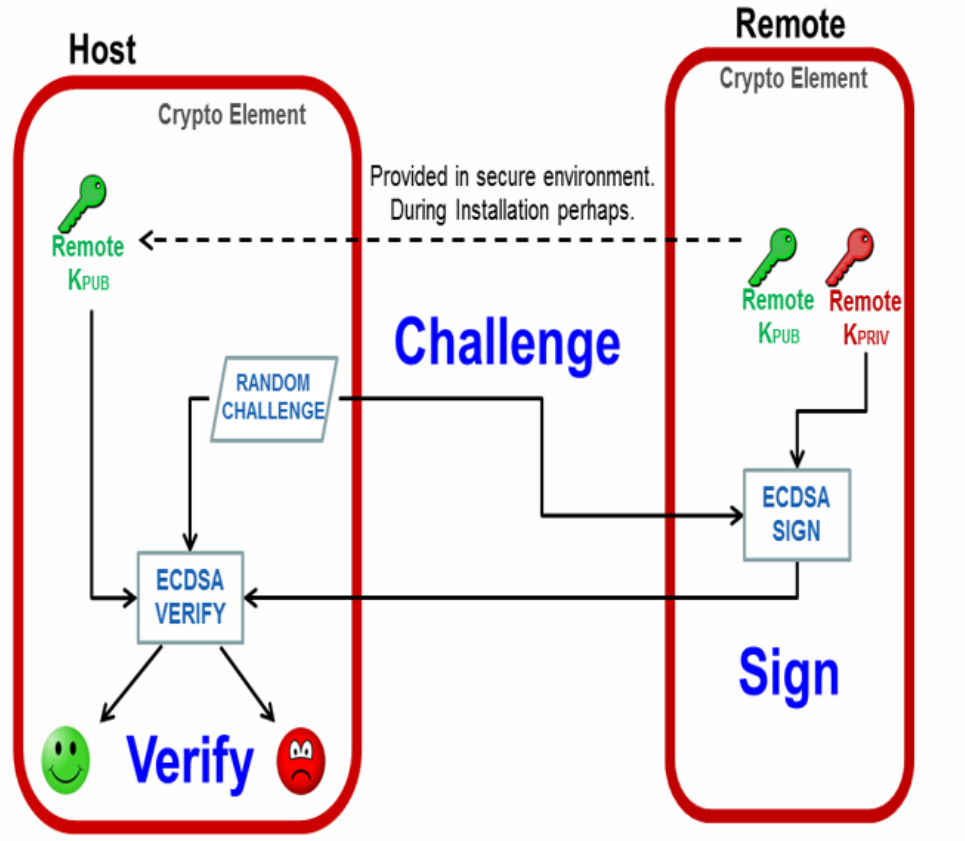


Note:	The Secure 4 click board 1 and Secure 4 click board 2 house the ATECC608A module and interface with the Curiosity PIC32MZ EF Development Board microcontroller over I^2C interface.

Authentication Process

The host sends a random challenge to the remote device. The remote device responds with a signature. However, the host only needs the public key from the remote (not a secret key) to verify the signature on the challenge.





If the signature verification matches, then the remote device has successfully responded to the challenge and the host can trust the remote device.

Building the Application

Description

To build this project, you must open the `ecc_asymmetric.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is *<install-dir>/apps/crypto/ecc_asymmetric*.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`ecc_asymmetric.X`	*<install-dir>/apps/crypto/ecc_asymmetric/firmware*

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within ./firmware/src/system_config.

Project Configuration Name	BSP Used	Description
pic32mz_ef_curiosity	pic32mz_ef_curiosity	Select this MPLAB X IDE project configuration to run the demonstration application to run on the PIC32MZ EF Curiosity Development Board, with the PIC32MZ2048EFM100 microcontroller. This configuration uses "Secure 4 click" board from MikroElektronika mounted on the mikroBUS header interfaces.

Configuring the Hardware

The following section describes how to configure the hardware for the demonstration.

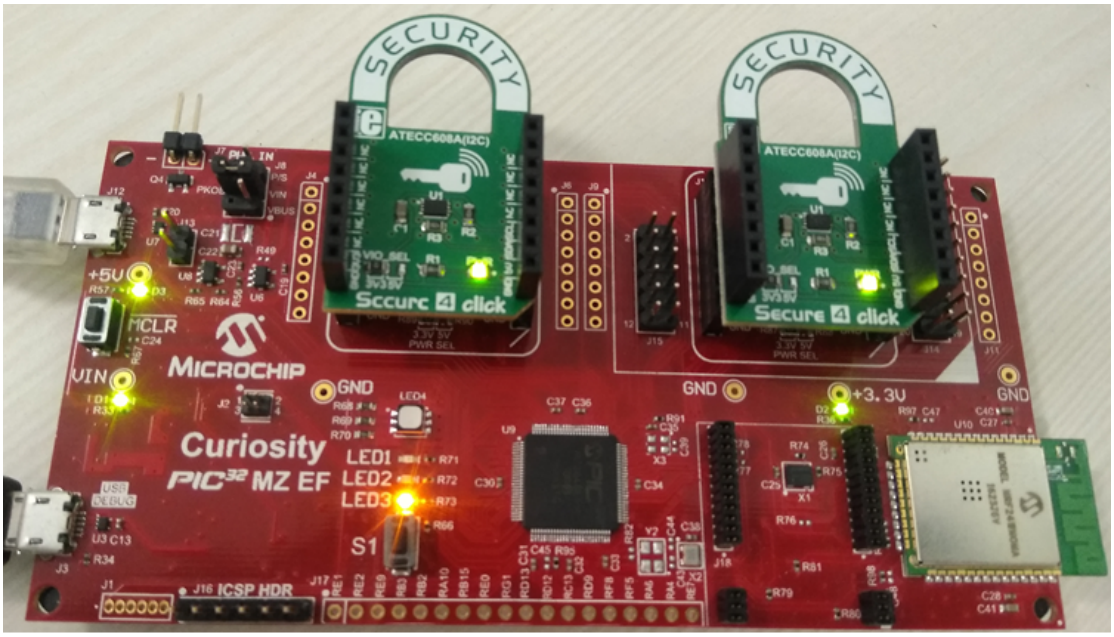
Description

PIC32MZ EF Curiosity Development Board

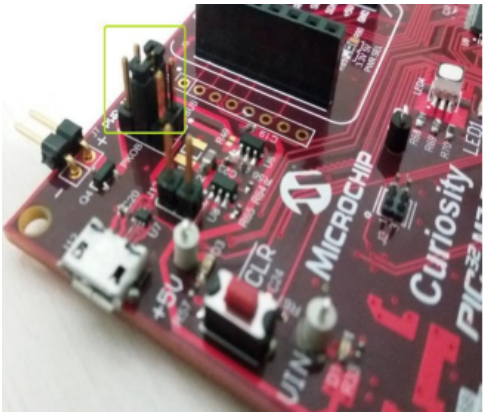
Configuration: pic32mz_ef_curiosity

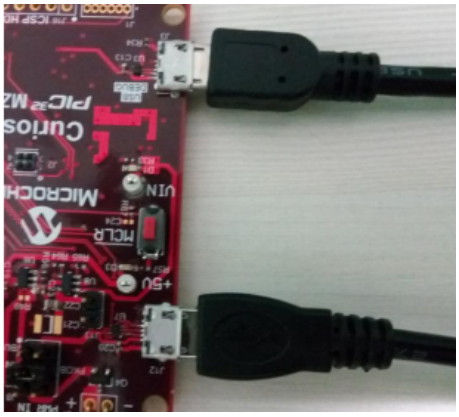
To configure the hardware, use the following steps:

1. For the host operation, mount a Secure 4 Click board on the mikroBUS socket J5.
2. For the remote device operation, mount a Secure 4 Click board on the mikroBUS socket J10.



3. Power the Curiosity PIC32MZ EF Development Board from the host computer through a Type-A male to Micro-B USB cable connected to the Micro-B port (J3). The cable is not included with the kit. Ensure that a jumper is placed in the J8 header (between 4 and 3) to select the supply from the debug USB connector.
4. Ensure that the jumper is not present in the J13 header to use the Curiosity board in Device mode. In Device mode, the board acts as a USB device to the computer. Plug in a USB cable with a Micro-B type connector to Micro-B port (J12), and plug the other end into the computer.





Running the Demonstration

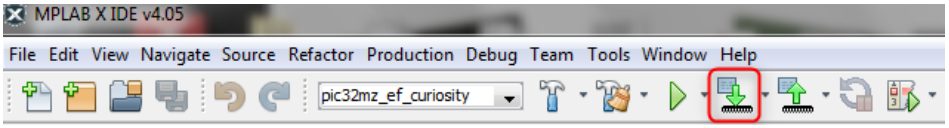
The following section describes how to run the demonstration.

Description

Important!	Prior to using this demonstration, it is recommended to review the MPLAB Harmony Release Notes for any known issues. A PDF copy of the release notes is provided in the *<install-dir>/doc* folder of your installation.

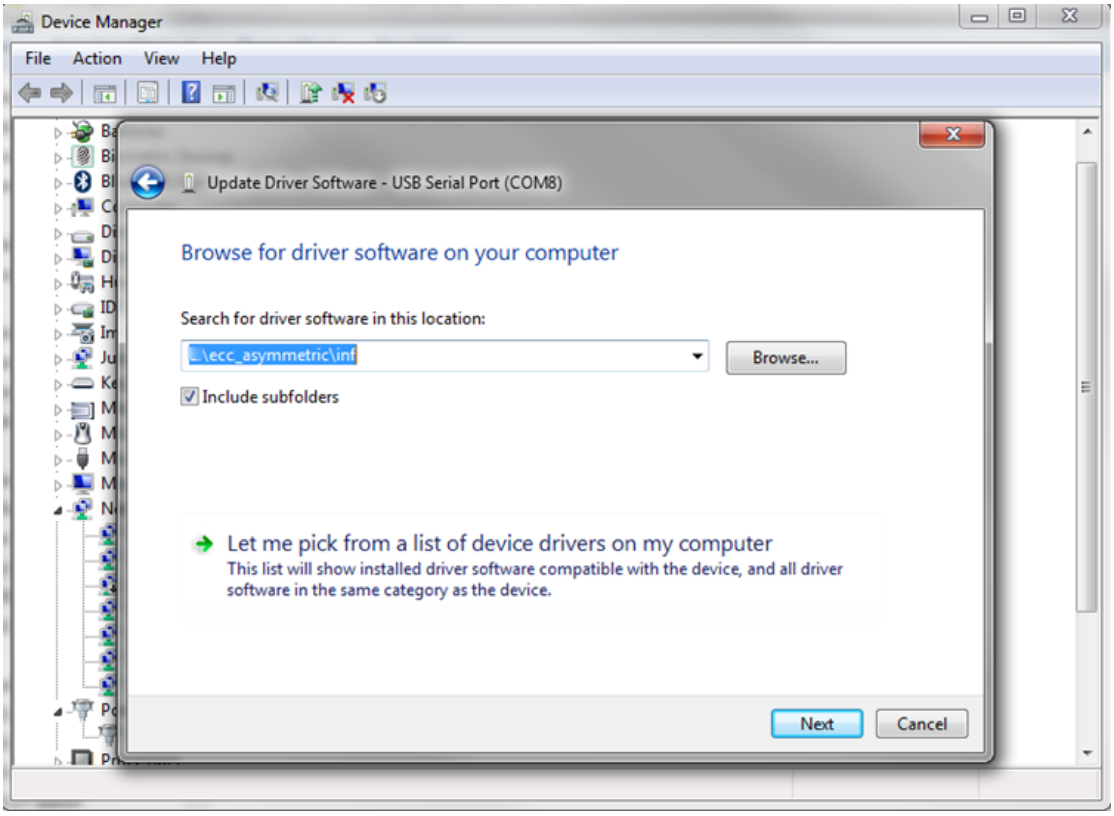
The following steps are used to run the demonstration:

1. Open the project in MPLAB X IDE and select the pic32mz_ef_curiosity project configuration.
2. Build the code and program the device by clicking on the program button as shown below.

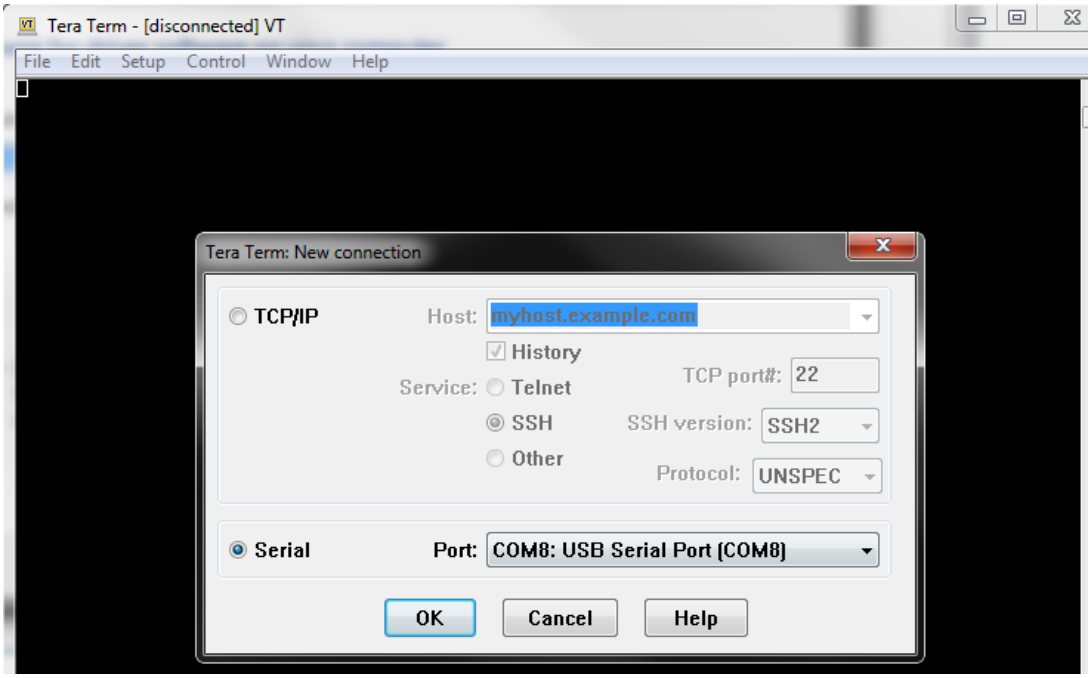


3. After power up, the demonstration is active. This is indicated by a yellow LED (LED3) on the board.
4. Plug in a USB cable with a Micro-B type connector to the Micro-B port (J12) of the Curiosity board, and plug the other end into the computer.
5. If this is the first time using this device with a personal computer, there may be a prompt for a `.inf` file.
6. Select the Install from a list or specific location (Advanced) option. Specify the path from *<install-dir>/apps/crypto/ecc_asymmetric/inf* directory.

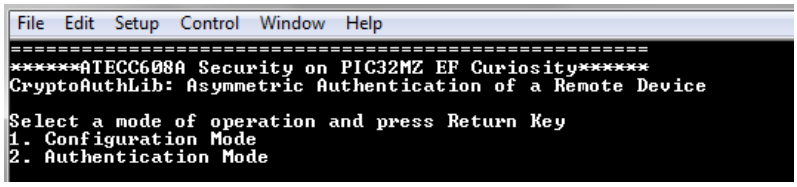
Note:	Optionally, to specify the driver, open the device manager and expand the Ports (COM & LPT) tab, then right click on Update Driver Software.



7. Once the device is installed, open a terminal program, such as Tera Term or HyperTerminal. Select the appropriate COM port for the terminal. The following figure shows the COM port selection for the Tera Term terminal program.



8. Once the Tera Term screen is displayed, Press the Enter key. The following modes of operation will be displayed.



9. The application demonstration offers the following two modes of operation.
   - Configuration Mode: This mode is used to configure a blank ATECC608A module with the configuration data and keys to be stored. A blank ATECC608A device is in an unlocked state. This operation performs a lock on the Configuration Zone and Data Zones on the ATECC608A device. The locking operation is a one time operation and is irreversible.
   - Authentication Mode: This mode performs the secured authentication of the Remote ATECC608A device.

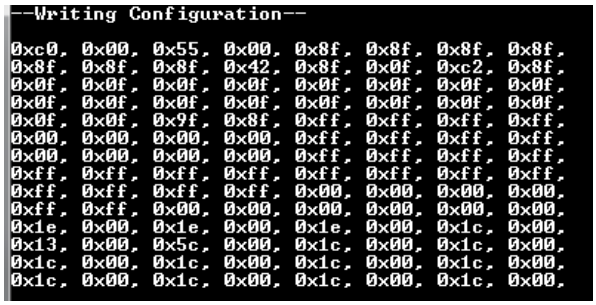
10. Selecting Option 1 - Configuration Mode

The display prompts the user to perform an action:



When the Secure 4 click board is plugged-in, the application enters into Configuration Write mode, and prints the existing or default configuration.

Note:	The existing or default configuration may be different from what is shown in the following figure.



If the plugged-in board is a brand new Secure 4 click board, the application will write the new configuration to the configuration zone and lock it. The application will display the following messages:

- Configuration Write Complete
- Locking Configuration Zone
- Configuration Zone Lock Complete

This would be followed by the writing of the new data (data slot contents and new keys) to the data zone and locking it. It would display the following messages:

- Writing Data Zone

- Data Zone Write Complete
- Locking Data Zone
- Data Zone Lock Complete
- **Host board Configuration Done**

If the plugged-in board is already configured, the application displays the following message:

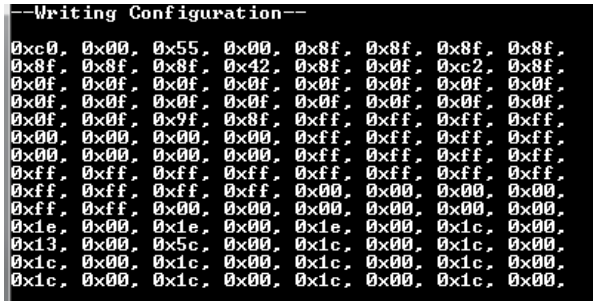
- ATCA already configured

Once the host configuration is completed, the display prompts the user to choose an action:

- Plug in remote Secure 4 Click board in Mikro Bus Interface 2 and press S1.

When the Secure 4 click board is plugged-in, the application enters the configuration write mode and prints the existing or default configuration.

Note:	The existing or default configuration may be different from what is shown in the following figure.



If the plugged-in board is a brand new Secure 4 click board, the application will write the new configuration to the configuration zone and lock it. It will display the following messages:

- Configuration Write Complete
- Locking Configuration Zone..
- Configuration Zone Lock Complete

This would be followed by writing of the new data (data slot contents and new keys) to the data zone and locking it. It would display the following messages:

- Writing Data Zone
- Data Zone Write Complete
- Locking Data Zone..
- Data Zone Lock Complete
- **Remote board Configuration Done**

If the plugged-in board is already configured, the application displays the following message:

- ATCA already configured.

The application provides a user the option to return to main menu:

- Press a key followed by 'enter' key to return to Options Menu.

11. Selecting option 2 - Authentication Mode

Fail Case:

- By default, if the application fails to authenticate the Remote Secure 4 click (plugged in J10), it will be indicated by a red LED (LED1) on the board

```
------In Authentication Mode------
Random from host
0x33, 0x59, 0x7b, 0x71, 0xba, 0x75, 0xa9, 0x64,
0x11, 0x75, 0x40, 0xbc, 0x15, 0xce, 0xfb, 0x2d,
0xb1, 0x6d, 0x18, 0xd2, 0x27, 0x98, 0x91, 0xa5,
0x8f, 0x8b, 0xef, 0xfb, 0xd1, 0x8f, 0x63, 0x53,

Signature from remote
0xc0, 0x36, 0x3a, 0x8b, 0x0c, 0x87, 0x8f, 0x52,
0x90, 0x42, 0x4a, 0xc1, 0xe8, 0xd2, 0x71, 0x28,
0xca, 0xb3, 0x0f, 0x04, 0xf6, 0x9f, 0x30, 0x31,
0xff, 0xec, 0xf5, 0x5c, 0xc5, 0xe3, 0xed, 0x0e,
0x69, 0x53, 0x87, 0xc3, 0xb0, 0x4c, 0x03, 0xa2,
0x9c, 0x9a, 0x57, 0x44, 0xfc, 0x43, 0xc2, 0x72,
0x7a, 0x96, 0xd6, 0x85, 0x8e, 0xd5, 0xd7, 0x24,
0x8e, 0x36, 0xea, 0x63, 0xc0, 0xde, 0x01, 0x9c,

Remote disposable public key
0x07, 0x8c, 0xe2, 0x2c, 0x19, 0xbc, 0xf4, 0x09,
0x4c, 0x56, 0xcb, 0x9a, 0x3c, 0xa0, 0x93, 0x74,
0xc7, 0xc2, 0xd2, 0x2d, 0x15, 0x76, 0x74, 0x9d,
0xdd, 0x8a, 0x6a, 0x22, 0xae, 0x67, 0xee, 0xc3,
0x97, 0x90, 0x73, 0xa2, 0x39, 0x72, 0x68, 0x30,
0x7e, 0xdf, 0x21, 0xa0, 0x37, 0xad, 0xcc, 0x2d,
0x7a, 0x0b, 0xa5, 0xd6, 0xac, 0x64, 0xab, 0x01,
0x00, 0xbc, 0x8b, 0xe5, 0x8a, 0xa1, 0x1e, 0xba,

Remote's Public Key not found in Host's Database

Signature not verified - Authentication Failure!
```

The Remote Secure 4 click fails to authenticate when the host does not have the public key for the remote in its database. The host will verify only those remote devices whose public key it possesses.

```
if(key_found)
{
    status = atcab_verify_extern((const uint8_t*)&nonce,
                                 (const uint8_t*)&signature,
                                 (const uint8_t*)key_store[i].pub_key,
                                 &verify);
}
else
{
    SYS_PRINT("Remote's Public Key not found in Host's Database \r\n\r\n");
}
```

Pass Case:

To verify the signature of the remote, place the Disposable Public Key of the remote in the database of the host. Copy the Remote Disposable Public Key from the previous figure and paste the key into the array key_store[] in ecc_asymmetric_app.c file as shown in the following figure.

```
asymm_public_key_t key_store[4] =
{
    0x07, 0x8c, 0xe2, 0x2c, 0x19, 0xbc, 0xf4, 0x09,
    0x4c, 0x56, 0xcb, 0x9a, 0x3c, 0xa0, 0x93, 0x74,
    0xc7, 0xc2, 0xd2, 0x2d, 0x15, 0x76, 0x74, 0x9d,
    0xdd, 0x8a, 0x6a, 0x22, 0xae, 0x67, 0xee, 0xc3,
    0x97, 0x90, 0x73, 0xa2, 0x39, 0x72, 0x68, 0x30,
    0x7e, 0xdf, 0x21, 0xa0, 0x37, 0xad, 0xcc, 0x2d,
    0x7a, 0x0b, 0xa5, 0xd6, 0xac, 0x64, 0xab, 0x01,
    0x00, 0xbc, 0x8b, 0xe5, 0x8a, 0xa1, 0x1e, 0xba,
}; //cut and paste in remote public key
```

Build and program the code. Repeat the user actions to select Authentication mode. The application now passes to verify the signature from the Remote Secure 4 click (plugged in J10). This is indicated by a green LED (LED2) on the board.

```
------In Authentication Mode------
Random from host
0xe4, 0x46, 0x63, 0x22, 0xd3, 0x2a, 0x88, 0x40,
0x69, 0xef, 0xdb, 0x0d, 0x42, 0xd0, 0x01, 0x53,
0xfb, 0x1a, 0x61, 0xd0, 0x1a, 0x08, 0xcc, 0x72,
0x81, 0xdd, 0x4d, 0x64, 0x36, 0x05, 0xa8, 0x86,

Signature from remote
0x3e, 0x73, 0x77, 0x25, 0xb3, 0xc8, 0x0d, 0xe7,
0xb1, 0xe0, 0xb4, 0xb5, 0xca, 0x11, 0x53, 0xe9,
0x99, 0xf9, 0xc5, 0x33, 0x64, 0x8c, 0xb2, 0x3a,
0x12, 0xc2, 0xc7, 0x67, 0xa8, 0xb7, 0x13, 0x50,
0xf9, 0x2c, 0x7f, 0xaf, 0x8d, 0x95, 0x9c, 0x42,
0xca, 0xde, 0x1f, 0xb4, 0x92, 0x9b, 0x86, 0x61,
0x8f, 0xb3, 0x6c, 0x8b, 0x93, 0xc0, 0x6a, 0x9a,
0xb9, 0x0e, 0x96, 0x8f, 0x81, 0xf2, 0x52, 0xca,

Remote disposable public key
0x07, 0x8c, 0xe2, 0x2c, 0x19, 0xbc, 0xf4, 0x09,
0x4c, 0x56, 0xcb, 0x9a, 0x3c, 0xa0, 0x93, 0x74,
0xc7, 0xc2, 0xd2, 0x2d, 0x15, 0x76, 0x74, 0x9d,
0xdd, 0x8a, 0x6a, 0x22, 0xae, 0x67, 0xee, 0xc3,
0x97, 0x90, 0x73, 0xa2, 0x39, 0x72, 0x68, 0x30,
0x7e, 0xdf, 0x21, 0xa0, 0x37, 0xad, 0xcc, 0x2d,
0x7a, 0x0b, 0xa5, 0xd6, 0xac, 0x64, 0xab, 0x01,
0x00, 0xbc, 0x8b, 0xe5, 0x8a, 0xa1, 0x1e, 0xba,

Verified Signature - Authentication Successful!
```

The signature of the Remote Secure 4 click is successfully verified as the remote had generated the signature using its private key and the random challenge from the host, while the host verified the signature by authenticating it against the public key of the remote.

The application provides the following option to return to the main menu:

- Press a key followed by 'enter' key to return to Options Menu.

ecc_symmetric

This application demonstrates the authentication of a remote device with a host (The Curiosity PIC32MZ EF Development Board and the secure 4 click board using the cryptography module ATECC608A) by using symmetric authentication method, where the host and the remote devices share the same key. The application allows adding information to the configuration, using the configuration data, and key data to configure a secure 4 click board. The application flow is realized through an interactive user interface through the serial terminal program (Tera term) interfaced through the USB of a computer.

For more information on the features and layout of the Curiosity PIC32MZ EF Development Board, refer to the PIC32MZ EF Curiosity Development Board User's Guide.

For more information on the features of the ATECC608A module, refer to the Product Data Sheet.

Description

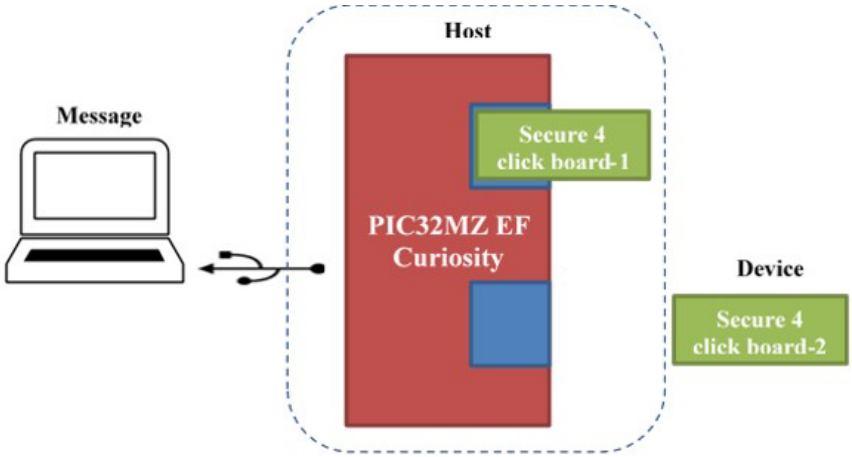
This application demonstrates the use of the ATECC608A module to authenticate the security of the

connected device. The authentication method used is Symmetric.

Symmetric authentication uses a challenge and response process. The host challenges a remote device to assure that it is authentic and can be trusted. The challenged device responds with the expected results. This method requires that both the host and the remote devices share the same key. Additionally, the remote device can send a unique serial number so the responses are unique from other remote devices.

In this application, a secure hardware key storage device (ATECC608A on a Secure 4 click board) is used to contain the shared key and unique serial number, in the host and remote device.

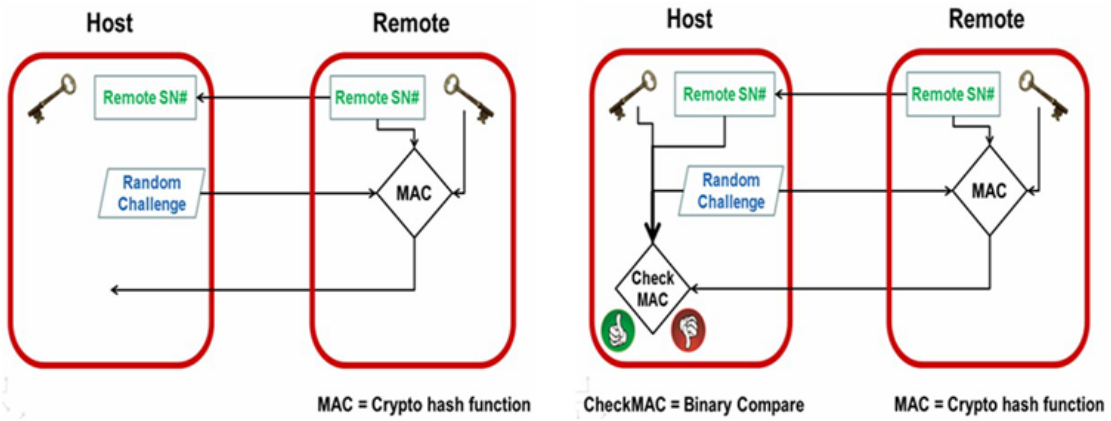
The following figure represents a functional block diagram of the application.



Note:	The Secure 4 click board 1 and Secure 4 click board 2 house the ATECC608A module and interface with the Curiosity PIC32MZ EF Development Board microcontroller over I2C interface.

Authentication Process

The authentication begins with the host asking for the serial number of the remote device. The host sends a random number, which it expects the remote device to hash with the shared secret key. This is called a challenge because it challenges the remote device to provide a correct answer. This challenge process is shown in the following figure.



The remote device hashes the random number with the shared key and the unique serial number, then sends back the resulting output of the hash, which is referred to as a Message Authentication Code (MAC).

The host checks the returned MAC by repeating the same operation. It hashes the shared key with the random number and the unique serial number of the remote device. The host compares the two results.

A matching result indicates that the challenge has been successfully responded to by the remote device and the host can trust the external device.

Building the Application

The following section describes how to build the application.

Description

To build this project, you must open the `ecc_symmetric.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is *<install-dir>/apps/crypto/ecc_symmetric.*

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`ecc_symmetric.X`	*<install-dir>/apps/crypto/ecc_symmetric/firmware*

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within ./firmware/src/system_config.

Project Configuration Name	BSP Used	Description
pic32mz_ef_curiosity	pic32mz_ef_curiosity	Select this MPLAB X IDE project configuration to run the demonstration application to run on the PIC32MZ EF Curiosity Development Board, with the PIC32MZ2048EFM100 microcontroller. This configuration uses "Secure 4 click" board from MikroElektronika mounted on the mikroBUS header interfaces.

Configuring the Hardware

This section describes how to configure the hardware for the demonstration.

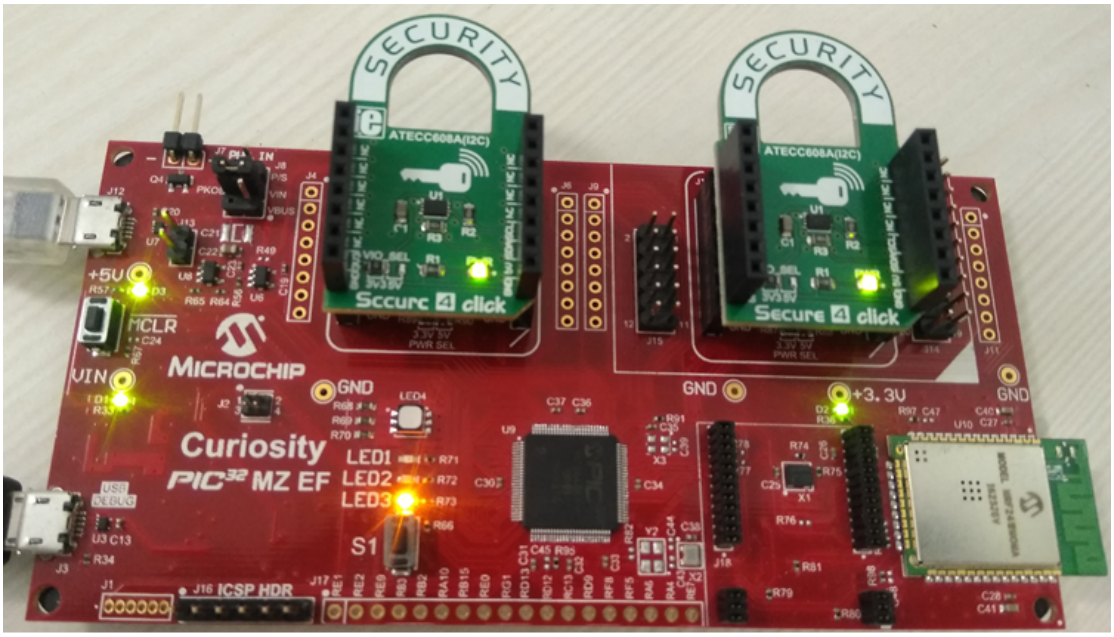
Description

PIC32MZ EF Curiosity Development Board

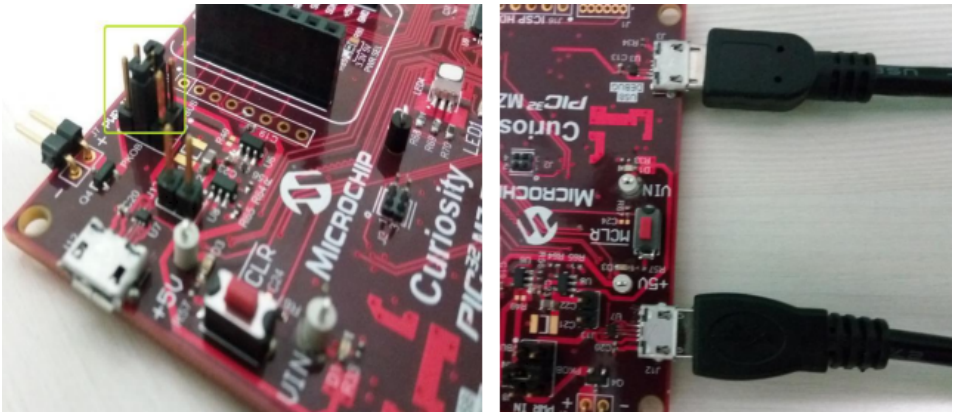
Configuration: pic32mz_ef_curiosity

To configure the hardware, use the following steps:

1. For the host operation, mount a Secure 4 click board on the mikroBUS socket J5.
2. For the remote device operation, mount a Secure 4 click board on the mikroBUS socket J10.



3. Power the Curiosity PIC32MZ EF Development Board from the host computer through a Type-A male to Micro-B USB cable connected to the Micro-B port (J3). The cable is not included with the kit. Ensure that a jumper is placed in the J8 header (between 4 and 3) to select the supply from the debug USB connector.
4. Ensure that the jumper is not present in the J13 header to use the Curiosity board in Device mode. In Device mode, the board acts as a USB device to the computer. Plug in a USB cable with a Micro-B type connector to Micro-B port (J12), and plug the other end into the computer.



Running the Demonstration

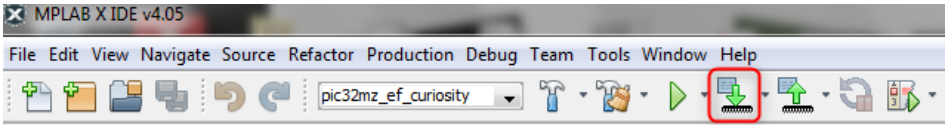
The following section describes how to run the demonstration.

Description

Important!	Prior to using this demonstration, it is recommended to review the MPLAB Harmony Release Notes for any known issues. A PDF copy of the release notes is provided in the *<install-dir>/doc* folder of your installation.

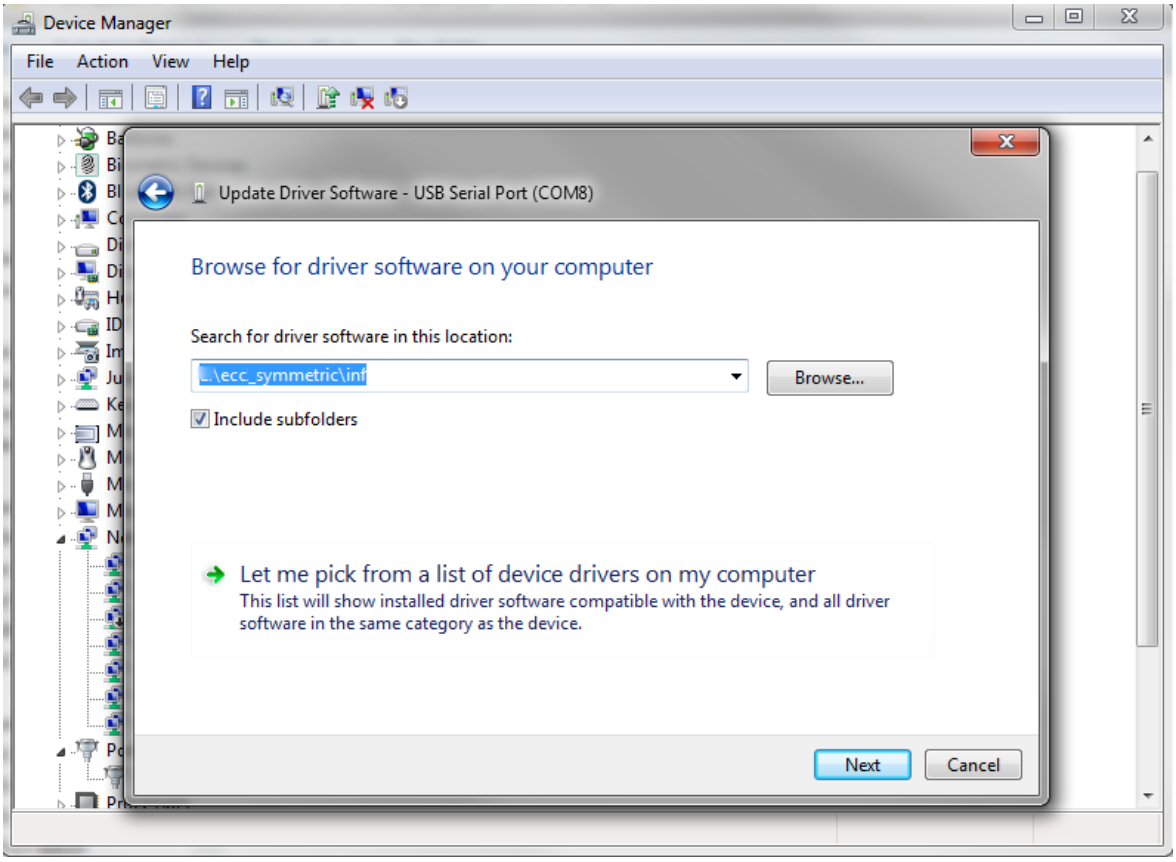
The following steps are used to run the demonstration:

1. Open the project in MPLAB X IDE and select the pic32mz_ef_curiosity project configuration.
2. Build the code and program the device by clicking on the program button as shown below.

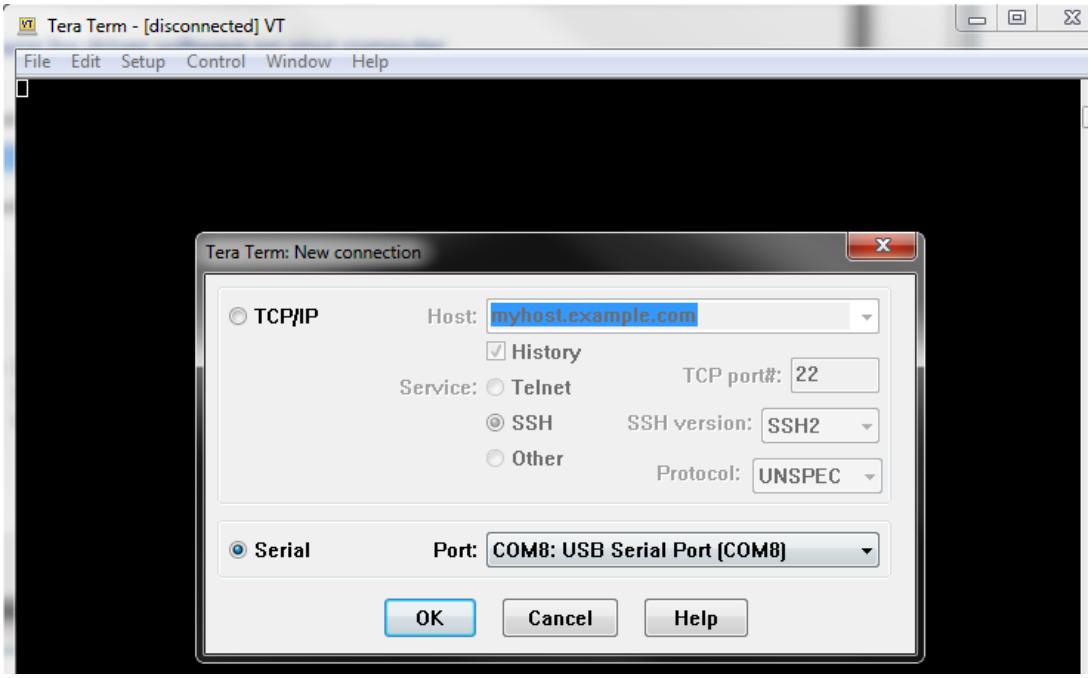


3. After power up, the demonstration is active. This is indicated by a yellow LED (LED3) on the board.
4. Plug in a USB cable with a Micro-B type connector to the Micro-B port (J12) of the Curiosity board, and plug the other end into the computer.
5. If this is the first time using this device with a personal computer, there may be a prompt for a .inf file.
6. Select the Install from a list or specific location (Advanced) option. Specify the path from <install-dir>/apps/crypto/ecc_symmetric/inf directory.

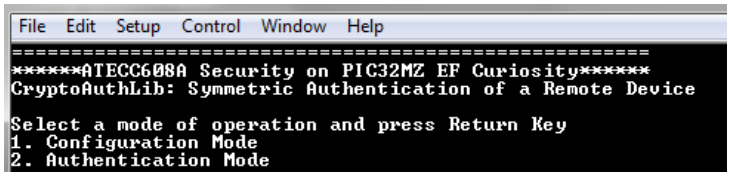
Note:	Optionally, to specify the driver, open the device manager and expand the Ports (COM & LPT) tab, then right click on Update Driver Software.



7. Once the device is installed, open a terminal program, such as Tera Term or HyperTerminal. Select the appropriate COM port for the terminal. The following figure shows the COM port selection for the Tera Term terminal program.



8. Once the Tera Term screen is displayed, Press the Enter key. The following modes of operation will be displayed.



9. The application demonstration offers the following two modes of operation.

- Configuration Mode: This mode is used to configure a blank ATECC608A module with the configuration data and keys to be stored. A blank ATECC608A device is in an unlocked state. This operation performs a lock on the Configuration Zone and Data Zones on the ATECC608A device. The locking operation is a one time operation and is irreversible.
- Authentication Mode: This mode performs the secured authentication of the Remote ATECC608A device.

10. Selecting Option 1 - Configuration Mode

- The display prompts the user to perform an action



When the Secure 4 click board is plugged-in, the application enters into Configuration Write mode, and prints the existing or default configuration.

Note:	The existing or default configuration may be different from what is shown in the following figure.



If the plugged-in board is a brand new Secure 4 click board, the application will write the new configuration to the configuration zone and lock it. The application will display the following messages:

- Configuration Write Complete
- Locking Configuration Zone..
- Configuration Zone Lock Complete

This would be followed by the writing of the new data (data slot contents and new keys) to the data zone and locking it. It would display the following messages:

- Writing Data Zone

- Data Zone Write Complete
- Locking Data Zone..
- Data Zone Lock Complete
- **Host board Configuration Done**

If the plugged-in board is already configured, the application displays the following message:

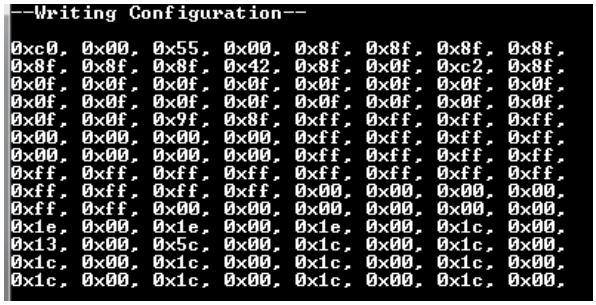
- ATCA already configured.

Once the host configuration is completed, the display prompts the user to choose an action:

- Plug in remote Secure 4 Click board in Mikro Bus Interface 2 and press S1.

When the Secure 4 click board is plugged-in, the application enters the configuration write mode and prints the existing or default configuration.

Note:	The existing or default configuration may be different from what is shown in the following figure.



If the plugged-in board is a brand new Secure 4 click board, the application will write the new configuration to the configuration zone and lock it. It will display the following messages:

- Configuration Write Complete
- Locking Configuration Zone
- Configuration Zone Lock Complete

This would be followed by writing of the new data (data slot contents and new keys) to the data zone and locking it. It would display the following messages:

- Writing Data Zone
- Data Zone Write Complete
- Locking Data Zone
- Data Zone Lock Complete
- **Remote board Configuration Done**

If the plugged-in board is already configured, the application displays the following message:

- ATCA already configured

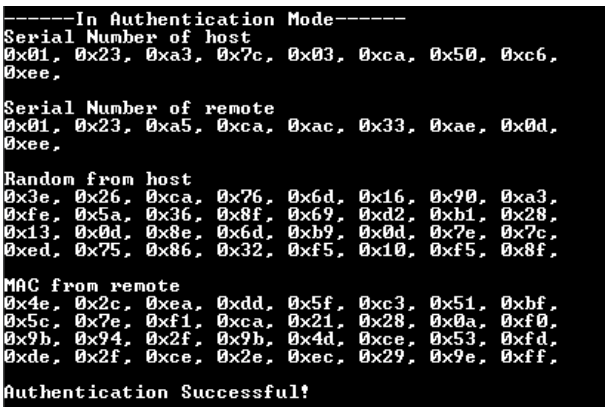
The application provides a user the option to return to main menu:

- Press a key followed by 'enter' key to return to Options Menu.

11. Selecting Option 2 - Authentication Mode

Pass Case:

- By default, the application performs successful authentication of the Remote Secure 4 click (plugged in J10). The successful authentication will be indicated by a green LED (LED2) on the board



- The Remote Secure 4 click successfully authenticates as it shares the secret key with the host.

Note:	In configuration mode, both the host and device (ATECC608A) are programmed with four identical keys. Refer to the function, `ECC_ATCA_CONFIGURE_WriteData` in `ecc_atca_configure.c`.

- In the file ecc_symmetric_app.c the implementation of the function _ECC_SYMMETRIC_APP_Handle_Authentication, the MAC is computed and verified for Slot 0, which corresponds to Key 0 being used to compute the MAC on the remote device. The same Key 0 is used to verify the MAC on the host device. Since the key is same, the MAC verification on the host is successful.

```
status = atcab_checkmac(mode, slot, (const uint8_t*)&nonce,
                        (const uint8_t*)&mac, (const uint8_t*)&otherdata);
```

Fail Case:

- To test the Authentication Failure case, uncomment and comment the below lines of code in the function `_ECC_SYMMETRIC_APP_Handle_Authentication` in the file `ecc_symmetric_app.c`.

```
/* CheckMac API Call for Testing Authentication Fail Case. Uncomment and build
 * the below code to verify the Authentication Fail case. */
slot = 1; // Using a different slot
status = atcab_checkmac(mode, slot, (const uint8_t*)&nonce,
                        (const uint8_t*)&mac, (const uint8_t*)&otherdata);


/* CheckMac API Call for Testing Authentication Pass Case. To test fail case
 * comment the below atcab_checkmac API call and uncomment the above one */
//status = atcab_checkmac(mode, slot, (const uint8_t*)&nonce,
//                        (const uint8_t*)&mac, (const uint8_t*)&otherdata);
```

- Build and program the code. Repeat the user actions to select Authentication Mode. The application will fail to authenticate the Remote Secure 4 click (plugged in J10). The failure is indicated by a red LED (LED1) on the board.

```
------In Authentication Mode------
Serial Number of host
0x01, 0x23, 0xa3, 0x7c, 0x03, 0xca, 0x50, 0xc6,
0xee,

Serial Number of remote
0x01, 0x23, 0xa5, 0xca, 0xac, 0x33, 0xae, 0x0d,
0xee,

Random from host
0x8e, 0xd5, 0xe8, 0x9a, 0x80, 0xf6, 0xe9, 0x80,
0x8d, 0xbb, 0x24, 0xca, 0x99, 0xd5, 0x89, 0x34,
0xa2, 0x23, 0xdc, 0x45, 0x02, 0xcb, 0x57, 0x0d,
0xd1, 0x7e, 0x77, 0x65, 0xc8, 0x7c, 0xc1, 0x24,

MAC from remote
0xfd, 0x05, 0x30, 0x3e, 0xe2, 0xdf, 0x5a, 0xbb,
0x5f, 0xab, 0x2b, 0x35, 0xfe, 0x88, 0x27, 0x97,
0x21, 0x73, 0x1b, 0xa0, 0xdd, 0x50, 0x9a, 0xc0,
0xab, 0xdd, 0x3d, 0xac, 0x0f, 0x92, 0xa1, 0x86,

Authentication Failure!
```

- The remote Secure 4 click failed to authenticate because the secret key used to compute the MAC by the remote, and the secret key used to verify the MAC from the remote by the host are different. The remote used the Key 0 to compute the MAC, while the host used Key 1 to verify the computed MAC. Since the Keys are not identical, the MAC verification fails, indicating that the host and the remote do not share a key.
- The application then provides the following message to return to the main menu:
  - Press a key followed by 'enter' key to return to Option Menu.

Driver Demonstrations

This section provides descriptions of the Driver demonstrations.

Data EEPROM Driver Demonstration

This topic provides descriptions of the Data EEPROM Driver demonstration.

Introduction

This help file contains instructions and associated information about MPLAB Harmony Data EEPROM Driver Library application demonstrations, which are included in the MPLAB Harmony Library distribution.

Description

This application demonstrates the capabilities of the MPLAB Harmony Data EEPROM Driver Library. This section describes the hardware requirement and procedures to build and run the demonstration project on Microchip development tools. In this demonstration application, the Data

EEPROM Driver is used to access the Data EEPROM of an PIC32MK family device to perform read, write operations.

To know more about MPLAB Harmony EEPROM Driver, configuring the Data EEPROM Driver and the APIs provided by the Data EEPROM Driver, refer to the MPLAB Harmony Data EEPROM Driver Library documentation.

Demonstrations

This topic provides information on how to run the Data EEPROM Driver demonstration applications included in this release.

eeprom_read_write

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Data EEPROM Read/Write Demonstration.

Description

To build this project, you must open the `eeprom_read_write.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/driver/eeprom/eeprom_read_write`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`eeprom_read_write.X`	`<install-dir>/apps/driver/eeprom/eeprom_read_write/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mk_gp_db_int_dyn	pic32mk_gp_db	This configuration runs on the PIC32MK GP Development Board.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MK GP Development Board

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

This section describes how to run the Data EEPROM Driver demonstration.

Description

This is a simple demonstration to show how to configure and make use of EEPROM Driver APIs to implement and access the on-board Flash memory of PIC32MX and PIC32MZ devices.

How to Run This Demonstration Application

Once the demonstration application is compiled successfully, you are ready to program the firmware in the target device.

To run the demonstration in Debug mode, perform the following steps:

1. Select the appropriate configuration from the MPLAB X IDE Project Properties based on the target hardware.
2. Select your device programmer from the Hardware Tool menu in MPLAB X IDE.
3. Select either *Debug > Debug Main Project* or click **Debug Main Project** in the toolbar.
4. Build the selected configuration in the MPLAB X IDE project and program the demonstration board.

The execution status (pass/fail) of the demonstration is indicated by LEDs on the demonstration board, as shown in the following table.

Demonstration Board	Success Indication	Failure Indication
PIC32MK GP Development Board	LED3	LED1

The application does the following:

- Opens the Data EEPROM Driver with read and write intent
- Fetches the geometry of the media and determines the block size and number of blocks of read, write and erase regions
- Writes 16 bytes of known data starting at block address 0
- Waits for the Write operation to complete
- Reads back and verifies the 16 bytes of data
- If the verification is successful, LED3 is turned ON; otherwise, LED1 is turned ON

I2C Driver Demonstrations

This topic provides descriptions of the I2C Driver demonstrations.

Introduction

This help file contains instructions and associated information about MPLAB Harmony I2C Driver application demonstrations, which are included in the MPLAB Harmony Library distribution.

Description

This application demonstrates the capabilities of the MPLAB Harmony I2C Driver. This section describes the hardware requirements and procedures to build and run the demonstration project using Microchip development tools.

One demonstration application is provided:

- i2c_rtcc - In this demonstration, one instance of the I2C peripherals acts as a Master and sends and receives data from two an external slave device. The slave device is the external MCP7049N Real-Time Clock Calendar (RTCC) device.

To know more about the MPLAB Harmony I2C Driver, configuring the driver and APIs provided by the I2C Driver, refer to the I2C Driver Library Help documentation.

Demonstrations

This topic provides information on how to run the I2C Driver demonstration applications included in this release.

i2c_rtcc

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the I2C RTCC demonstration.

Description

To build this project, you must open the `i2c_rtcc.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/driver/i2c/i2c_rtcc`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`i2c_rtcc.X`	`<install-dir>/apps/driver/i2c/i2c_rtcc/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx795_pim_e16	pic32mx795_pim+e16	The purpose of this configuration is to demonstrate I^2C Master mode transfer setup in Interrupt mode and dynamic operation. The hardware used is the PIC32MX795F512L PIM connected to the Explorer 16 Development Board.
pic32mx795_pim_e16_freertos	pic32mx795_pim+e16	This configuration is the freertos version of the demo. The purpose of this configuration is to demonstrate the I^2C Master mode transfer setup in Interrupt mode and dynamic operation. The hardware used is the PIC32MX795F512L PIM connected to the Explorer 16 Development Board.
pic32mz_ef_sk_meb2	pic32mz_ef_sk+meb2	The purpose of this configuration is to demonstrate I^2C Master mode transfer setup in Interrupt mode and dynamic operation. The hardware used is the PIC32MZ EF Starter Kit connected to the MEB II.
pic32mz_ef_sk_meb2_16b	pic32mz_ef_sk+meb2	This configuration is microMIPS version of the demo. The purpose of this configuration is to demonstrate I^2C Master mode transfer setup in Interrupt mode and dynamic operation. The hardware used is the PIC32MZ EF Starter Kit connected to the MEB II.
pic32mz_ef_sk_meb2_freertos	pic32mz_ef_sk+meb2	This configuration is FreeRTOS version of the demo. The purpose of this configuration is to demonstrate I^2C Master mode transfer setup in Interrupt mode and dynamic operation. The hardware used is the PIC32MZ EF Starter Kit connected to the MEB II.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

Note: The i2c_rtcc demonstration was tested on the Microchip MCP7949N RTCC device. The address of this device is 0xDE.

Explorer 16 Development Board with the PIC32MX795F512L CAN-USB PIM

- Before attaching the PIC32MX795F512L PIM to the Explorer 16 Development Board, ensure that the processor select switch (S2) is in the PIM Position
- Short JP2 on the Explorer 16 Development Board to enable the LEDs

If a Starter Kit I/O Expansion Board is used, make the following connections:

- Jumper I/O Expansion board J11 pin 36 (SDA2) and J11 pin 38 (SCL2) should be pulled up to 3.3V through a 2.2k ohm resistor

If an external RTCC is used, make the following connections:

- Jumper I/O Expansion board J11 pin 36 (SDA2) and J11 pin 38 (SCL2) to the corresponding lines of an external I2C device
- If an external I2C device is connected, ensure that the Master and Slave device share a common ground.

If a PICtail Plus Daughter Board is used, make the following connections:

- PICtail Plus Daughter Board pins RA2 (SCL2) and RA3 (SDA2) should be pulled up to 3.3V through a 2.2k ohm resistor

If an external RTCC is used, make the following connections:

- Jumper PICtail Plus Daughter Board pins RA2 (SCL2) and pin RA3 (SDA2) to the corresponding SCL and SDA lines of an external I2C device
- If an external I2C device is connected, ensure that the Master and Slave device share a common ground

PIC32MZ EC Starter Kit or PIC32MZ EF Starter Kit connected to the MEB II

The jumper JP2 on PIC32MZ EC/EF Starter Kit should connected according to the debugger/programmer used, as follows:

- If PKOB is used, pins 1 and 3 and pins 2 and 4 should be shorted
- If MPLAB REAL ICE or MPLAB ICD 3 is being used, pins 1 and 3 and pins 2 and 4 should be left open

The connections pertaining to I2C are as follows:

- Connect MEB II J2 pin 3 (SCL2) to the corresponding SCL line of the external I2C device
- Connect MEB II J2 pin 5 (SDA2) to the corresponding SDA line of the external I2C device
- If an external I2C device is connected, ensure that the Master and Slave device share a common ground

Running the Demonstration

Provides instructions on how to build and run the I2C RTCC demonstration.

Description

1. This demonstration shows how to configure and make use of the I2C Driver APIs to support buffered operation of I2C in Interrupt mode. In this demonstration, the I2C is configured as single instance and single client.

Once the demonstration application is compiled successfully for the selected configuration, the firmware can be programmed into the target device.

To run the demonstration in Debug mode, perform the following steps:

1. Select the appropriate configuration from the MPLAB X IDE Project Properties based on the target hardware.
2. Select your device programmer from the Hardware Tool menu in MPLAB X IDE.
3. Select either *Debug > Debug Main Project* or click **Debug Main Project** in the toolbar.
4. Build the selected configuration in the MPLAB X IDE project and program the demonstration board.

The I2C Driver configures the I2C2 instance of the I2C peripheral in Master mode. The SDA and SCL lines are connected to the Microchip MCP7940N RTCC device as described in Configuring the Hardware. The Master writes to sequential memory locations in SRAM memory of the RTCC device. The Master then reads back the content from the same page.

The contents of the buffer variable can be checked to determine the result of the operation.

The expected results are shown in the following table.

Test Case	Contents of Buffer
I2C2 (Master)	`RXbuffer_4[] = "3RTCCSLAVE"` (data received from RTCC device)

NVM Driver Demonstration

This topic provides descriptions of the NVM Driver demonstration.

Introduction

This help file contains instructions and associated information about MPLAB Harmony NVM Driver Library application demonstrations, which are included in the MPLAB Harmony Library distribution.

Description

This application demonstrates the capabilities of the MPLAB Harmony NVM Driver Library. This section describes the hardware requirement and procedures to build and run the demonstration project on Microchip development tools. In this demonstration application, the NVM driver is used to access the internal Flash memory of PIC32MX and PIC32MZ devices to perform Write and Read operations and indicates the result by LED.

To know more about MPLAB Harmony NVM driver, configuring the NVM driver and the APIs provided by the NVM driver, refer to the MPLAB Harmony NVM Driver Library documentation.

Demonstrations

This topic provides information on how to run the NVM Driver demonstration applications included in this release.

nvm_read_write

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the NVM Read/Write Demonstration.

Description

To build this project, you must open the `nvm_read_write.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/driver/nvm/nvm_read_write`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`nvm_read_write.X`	`<install-dir>/apps/driver/nvm/nvm_read_write/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx_usb_sk2	pic32mx_usb_sk2	The purpose of this configuration is to execute the demonstration on a PIC32MX family device using the PIC32 USB Starter Kit II with the dynamic NVM Driver implementation.
pic32mx_usb_sk2_sta	pic32mx_usb_sk2	The purpose of this configuration is to execute the demonstration on a PIC32MX family device using the PIC32 USB Starter Kit II with the static NVM Driver implementation.
pic32mz_ef_sk	pic32mz_ef_sk	The purpose of this configuration is to execute the demonstration on a PIC32MZ EF family device using the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit with the dynamic NVM Driver implementation.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 USB Starter Kit II

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

This section describes how to run the NVM Driver demonstration.

Description

This is a simple demonstration to show how to configure and make use of NVM Driver APIs to implement and access the on-board Flash memory of PIC32MX and PIC32MZ devices.

How to Run This Demonstration Application

Once the demonstration application is compiled successfully, you are ready to program the firmware in the target device.

To run the demonstration in Debug mode, perform the following steps:

1. Select the appropriate configuration from the MPLAB X IDE Project Properties based on the target hardware.
2. Select your device programmer from the Hardware Tool menu in MPLAB X IDE.
3. Select either *Debug > Debug Main Project* or click **Debug Main Project** in the toolbar.
4. Build the selected configuration in the MPLAB X IDE project and program the demonstration board.

The execution status (pass/fail) of the demonstration is indicated by LEDs on the demonstration board, as shown in the following table.

Demonstration Board	Success Indication	Failure Indication
PIC32 USB Starter Kit II PIC32MZ EF Starter Kit	LED3 (Green LED)	LED1 (Red LED)

The demonstration application makes use of 32 KB of NVM memory area starting at address DRV_NVM_MEDIA_START_ADDRESS.

The application does the following:

- Erases the entire 32 KB of memory and verifies the erase operation by reading back the data
- Performs sequential writes within a page by queuing the write operations. Reads back and verifies the data.
- Repeats step 1 to erase all data of the previous operation
- Performs random writes to addresses spread across the available memory area. This operation demonstrates the queuing of the write operations at the driver layer. It also demonstrates the usage of the driver event handler to track the completion of the queued operations. Reads back and verifies the data.
- Repeats step 1 to erase all data from the previous operation
- Performs an EraseWrite operation. This operation demonstrates the usage of the EraseWrite feature

SPI Driver Demonstrations

This topic provides descriptions of the SPI Driver demonstrations.

Introduction

This help file contains instructions and associated information about MPLAB Harmony SPI driver application demonstrations, which are included in the MPLAB Harmony Library distribution.

Description

This application demonstrates the capabilities of the MPLAB Harmony SPI Driver. This section describes the hardware requirements and procedures to build and run the demonstration project using Microchip development tools.

Three demonstration applications are provided:

- serial_eeprom - In this demonstration application, the SPI Driver is used to access the external EEPROM in the Explorer 16 Development Board to perform Write and Read operations and indicates the result by LED
- spi_loopback - In this demonstration application, the SPI driver is used to transfer data between the SPI Master and slave on the same device and indicates the result by LED
- spi_multislave - In this demonstration application, the SPI Driver is used to transfer data between a single Master and two Slaves on the same device using the Slave Select (SS) feature
- spi_self_loopback - In this demonstration application, the SPI driver is used to transfer data between the SPI output pin and SPI input pin of the same SPI module/instance and the result of the transfer is indicated using LEDs

To know more about the MPLAB Harmony SPI driver, configuring the SPI driver and APIs provided by the SPI driver, refer to the SPI Driver Library documentation.

Demonstrations

This topic provides information on how to run the SPI Driver demonstration applications included in this release.

serial_eeprom

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

In this demonstration application, the SPI Driver is used to access the external EEPROM in the Explorer 16 Development Board to perform Write and Read operations and indicates the result by LED.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Serial EEPROM Demonstration.

Description

To build this project, you must open the `serial_eeprom.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/driver/spi/serial_eeprom`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`serial_eeprom.X`	`<install-dir>/apps/driver/spi/serial_eeprom/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx360_pim_e16	pic32mx795_pim+e16	The purpose of this configuration is to execute the demonstration using the PIC32MX360F512L PIM connected to the Explorer 16 Development Board configured for Interrupt mode and dynamic operation.
pic32mx795_pim_e16	pic32mx795_pim+e16	The purpose of this configuration is to execute the demonstration using the PIC32MX795F512L PIM connected to the Explorer 16 Development Board configured for Interrupt mode and dynamic operation.

pic32mx795_pim_e16_sta	pic32mx795_pim+e16	The purpose of this configuration is to execute the demonstration using the PIC32MX795F512L PIM connected to the Explorer 16 Development Board configured for Interrupt mode and static operation.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

Explorer 16 Development Board with the PIC32MX795F512L CAN-USB PIM

- Before attaching the PIC32MX795F512L PIM to the Explorer 16 Development Board, ensure that the processor select switch (S2) is in the PIM Position
- Short JP2 on the Explorer 16 Development Board to enable the LEDs

Explorer 16 Development Board with the PIC32MX360F512L PIM

- Before attaching the PIC32MX360F512L PIM to the Explorer 16 Development Board, ensure that the processor select switch (S2) is in the PIM Position
- Short JP2 on the Explorer 16 Development Board to enable the LEDs

Running the Demonstration

This section demonstrates how to run the SPI Driver Serial EEPROM Demonstration.

Description

This demonstration shows how to configure and make use of the SPI Driver APIs to implement and access the on-board EEPROM of the Explorer 16 Development Board.

How to run this demonstration application:

Once the demonstration application is successfully compiled, you are ready to program the firmware in the target device.

To run the demonstration in debug mode, perform the following steps:

1. Select your device programmer from *Project Properties > Hardware Tools* in MPLAB X IDE.
2. Select either *Debug > Debug Main Project* or click **Debug Main Project** in the toolbar.

Once the device is successfully programmed, you can observe that the LED in the Explorer 16 Development Board has been turned ON. This shows the demonstration project ran successfully. The result of the programming can be read through the other LEDs. If LED "D9" is turned ON, it indicates the EEPROM W/R functionality has failed. If LED "D10" is turned ON, it indicates the EEPROM W/R functionality has passed.

spi_loopback

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

In this demonstration application, the SPI driver is used to transfer data between the SPI Master and Slave on the same device and indicates the result by LED.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the SPI Loopback Demonstration.

Description

To build this project, you must open the `spi_loopback.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/driver/spi/spi_loopback`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`spi_loopback.X`	`<install-dir>/apps/driver/spi/spi_loopback/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx_usb_sk2_int_dyn	pic32mx_usb_sk2	Demonstrates SPI loopback on the PIC32 USB Starter Kit II configured for Interrupt mode and dynamic operation.
pic32mx_usb_sk2_int_sta	pic32mx_usb_sk2	Demonstrates SPI loopback on the PIC32 USB Starter Kit II configured for Interrupt mode and static operation.
pic32mx_usb_sk2_poll_dyn	pic32mx_usb_sk2	Demonstrates SPI loopback on the PIC32 USB Starter Kit II configured for Polled mode and dynamic operation.
pic32mz_ef_sk_int_dyn	pic32mz_ef_sk	Demonstrates SPI loopback on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit configured for Interrupt mode and dynamic operation.
pic32mz_ef_sk_int_dyn_dma	pic32mz_ef_sk	Demonstrates SPI loopback on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit configured for Interrupt mode and dynamic operation using DMA.
pic32mz_ef_sk_poll_dyn	pic32mz_ef_sk	Demonstrates SPI loopback on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit configured for Polled mode and dynamic operation.
pic32mz_ef_sk_int_dyn_16b	pic32mz_ef_sk	Demonstrates SPI multi-slave on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit configured for Interrupt mode and dynamic operation in microMIPS mode.
pic32mz_ef_sk_int_dyn_freertos	pic32mz_ef_sk	Demonstrates SPI multi-slave on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit configured for Interrupt mode and dynamic operation with FreeRTOS.
pic32mx_usb_sk2_int_dyn_freertos	pic32mx_usb_sk2	Demonstrates SPI loopback on the PIC32 USB Starter Kit II configured for Interrupt mode and dynamic operation with FreeRTOS.
pic32wk_sk_int_dyn	pic32wk_gpb_gpd_sk+module	Demonstrates SPI loopback on the PIC32WK Wi-Fi Starter Kit configured for Interrupt mode and dynamic operation.
pic32wk_sk_int_sta	pic32wk_gpb_gpd_sk+module	Demonstrates SPI loopback on the PIC32WK Wi-Fi Starter Kit configured for Interrupt mode and static operation.
pic32wk_sk_poll_dyn	pic32wk_gpb_gpd_sk+module	Demonstrates SPI loopback on the PIC32WK Wi-Fi Starter Kit configured for Polled mode and dynamic operation.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

Required Hardware

This demonstration requires the following hardware:

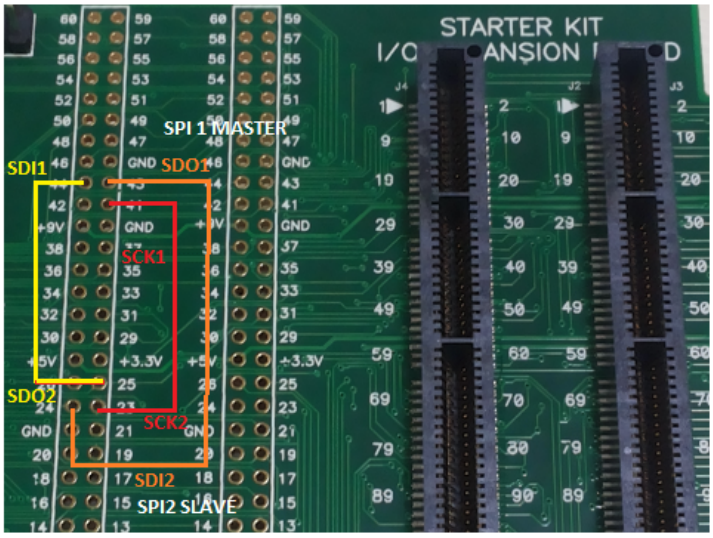
- PIC32MZ EF Starter Kit or PIC32WK Wi-Fi Starter Kit
- PIC32MZ Starter Kit Adapter Board
- Starter Kit I/O Expansion Board

Depending on the starter kit in use, the SPI1 and SPI2 modules or SPI1 and SPI3 modules are used in this demonstration. PIC32MZ and PIC32WK devices support the Peripheral Pin Select (PPS) feature. The SPI Pins of the SPI modules on this device are required to be configured using the PPS. Any related pin mapping (PPS) configuration code along with other port initialization can be found in the `sys_port_static.c` file.

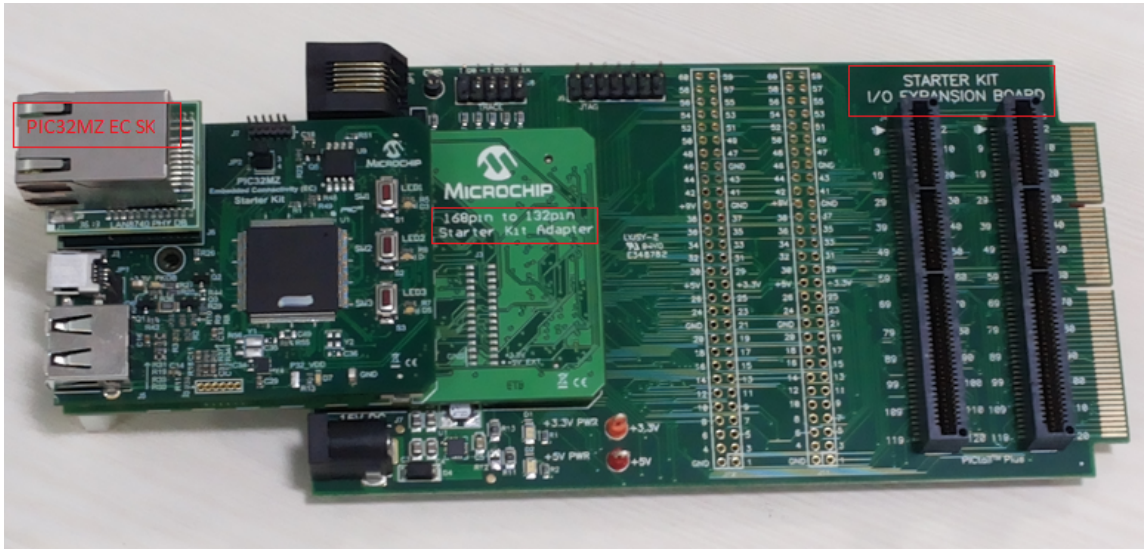
Note: To know more about the PIC32MZ and PIC32WK PPS feature and pin configuration, please refer to **Section 12.3 "Peripheral Pin Select (PPS)"** in the **"I/O Ports"** chapter of the specific device data.

PIC32MZ EF Starter Kit Configuration

1. Connect (using wires) the pins on the Starter Kit I/O Expansion Board, as shown in the following figure. This hardwired connection would connect the SPI1 lines to the SPI2 lines of the PIC32MZ2048ECH144 device on the PIC32MZ EC Starter Kit or the PIC32MZ2048EFM144 device on the PIC32MZ EF Starter Kit.



2. Next, the starter kit, the PIC32MZ Starter Kit Adapter Board, and the Starter Kit I/O Expansion Board, as shown in the following figure.



PIC32MZ EF Pin Mapping

The following table illustrates how the SPI1 and SPI2 pins for the PIC32MZ EF devices are mapped and connected to each other through the Starter Kit I/O Expansion Board.

SPI Module	SPI Lines	PIC32MZ2048EFM144 Device Pin #	Analog Pin	PIC32MZ2048EFM144 Port Pin Name/Function	Pin # on I/O Expansion Board J10 Connector	For this demonstration, attach this pin to:
SPI1	SCK1	109	No	SCK1	41	SCK2
SPI1	SDI1	69	AN32	RPD14	44	SDO2
SPI1	SDO1	98	No	RPD10	43	SDI2
SPI1	/SS	Not Used	Not Used	Not Used	Not Used	Not Used
SPI2	SCK2	14	AN14	SCK2	23	SCK1
SPI2	SDI2	121	No	RPD7	24	SDO1
SPI2	SDO2	25	AN45	RPB5	25	SDI1
SPI2	/SS	Not Used	Not Used	Not Used	Not Used	Not Used

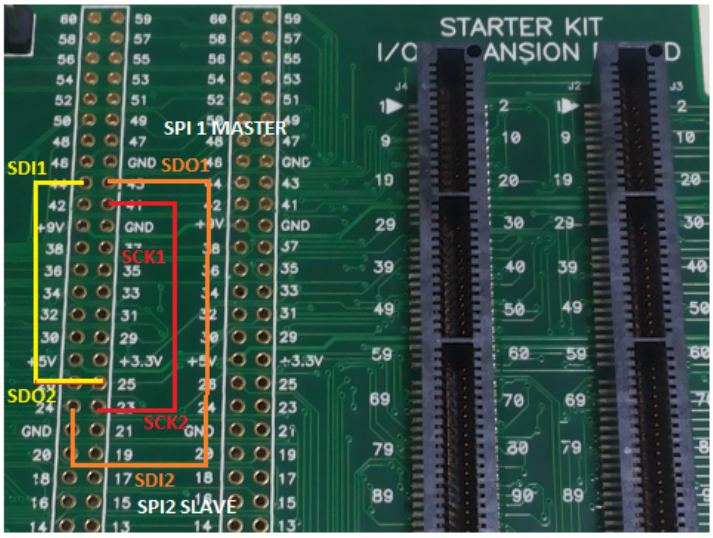
PIC32 USB Starter Kit II

This demonstration requires the following hardware:

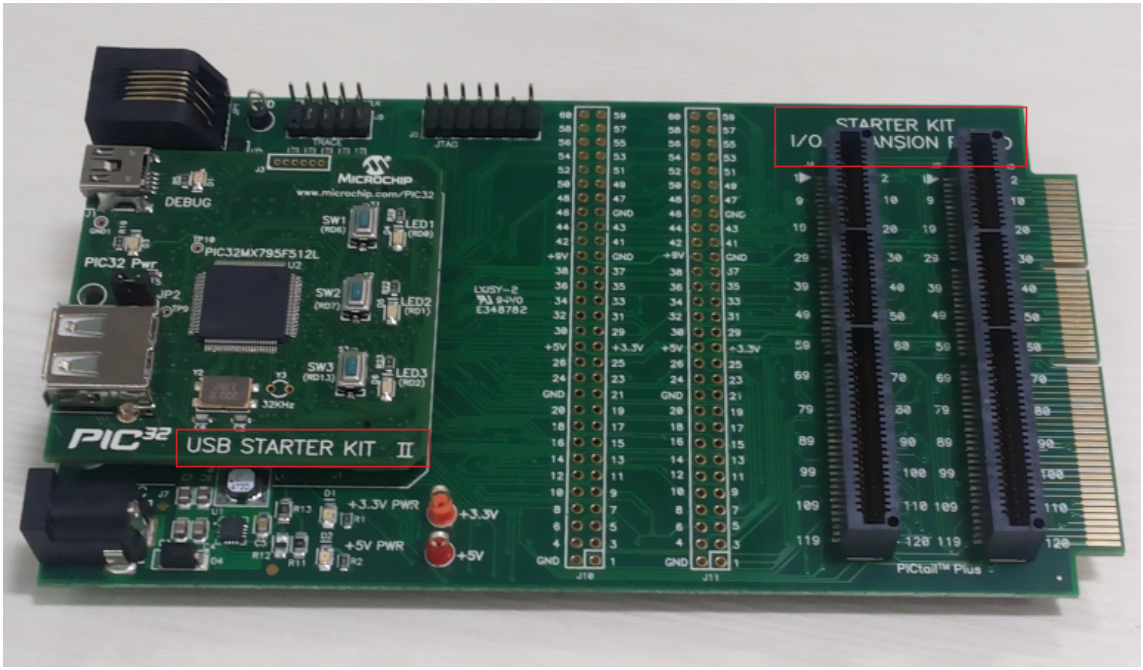
- PIC32 USB Starter Kit II
- Starter Kit I/O Expansion Board

In this demonstration application, the SPI1 and SPI2 modules are used.

1. Connect (using wires) the pins on the Starter Kit I/O Expansion Board, as shown in the following figure. This hardwired connection would attach the SPI1 lines to the SPI2 lines of the PIC32MX795F512L device on the PIC32 USB Starter Kit II through the Starter Kit I/O Expansion Board.



2. Next, connect the PIC32 USB Starter Kit II and the Starter Kit I/O Expansion Board, as shown in the following figure.



For the PIC32MX795F512L device, the SPI modules has dedicated I/O pins.

PIC32 USB Starter Kit II Pin Mapping

The following table illustrates how the SPI1 and SPI2 pins are mapped and connected to each other through the Starter Kit I/O Expansion Board.

SPI Module	SPI Lines	PIC32MX795F512L Device Pin #	PIC32MX795F512L Port Pin Name/Function	Pin # on I/O Expansion Board J10 Connector	For this demonstration, attach this pin to:
SPI1	SCK1	70	SCK1	41	SCK2
SPI1	SDI1	9	SDI1	44	SDO2
SPI1	SDO1	72	SDO1	43	SDI2
SPI1	/SS	Not Used	Not Used	Not Used	Not Used
SPI2	SCK2	10	SCK2	23	SCK1

SPI2	SDI2	11	SDI2	24	SDO1
SPI2	SDO2	12	SDO2	25	SDI1
SPI2	/SS	Not Used	Not Used	Not Used	Not Used

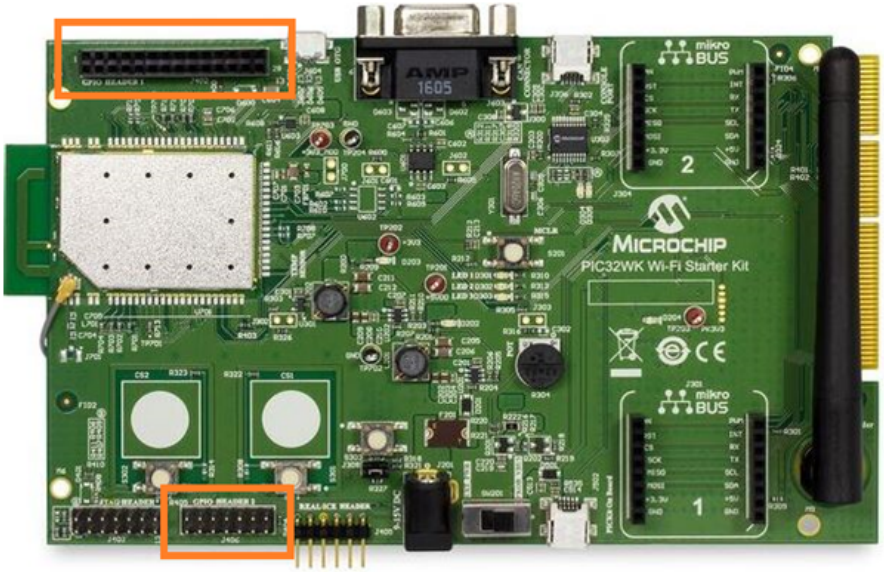
PIC32WK Wi-Fi Starter Kit II

This demonstration requires the following hardware:

- PIC32WK Wi-Fi Starter Kit

In this this demonstration application, the SPI1 and SPI2 modules are used. Connect (using wires) the pins on the Starter Kit I/O Expansion Board, as shown in the table and image below. This hardwired connection would attach the SPI1 lines to the SPI2 lines of the PIC32WK2057GPD132 device on the PIC32WK WiFI starter kit through the GPIO headers.

SPI Module	SPI Lines	PIC32WK2057GPD132 Device Pin #	PIC32WK2057GPD132 Port Pin Name/Function	Pin # on starter kit	For this demonstration, attach this pin to:
SP1	SCK1	Pin 2 (GPIO header 2)	SCK1		SCK2
SP1	SDI1	Pin 5 (GPIO header 2)	SDI1		SDO2
SP1	SDO1	Pin 3 (GPIO header 2)	SDO1		SDI2
SP1	/SS	Not Used	Not Used		Not Used
SP2	SCK2	Pin 6 (GPIO header 1)	SCK2		SCK1
SP2	SDI2	Pin1 6 (GPIO header 1)	SDI2		SDO1
SP2	SDO2	Pin 12 (GPIO header 1)	SDO2		SDI1
SP2	/SS	Not Used	Not Used		Not Used



Running the Demonstration

This section demonstrates how to run the SPI Driver SPI Loopback Demonstration.

Description

This demonstration shows how to configure and make use of the SPI Driver APIs to support multiple SPI hardware instances to multiple clients of the driver in both interrupt mode and polled mode. This demonstration shows the SPI driver's "multi-instance multi-client" feature.

Once the demonstration application is compiled successfully, you are ready to program the firmware in the target device.

To run the demonstration in Debug mode, perform the following steps:

1. Select the appropriate configuration from the MPLAB X IDE Project Properties based on the target hardware.
2. Select your device programmer from the Hardware Tool menu in MPLAB X IDE.
3. Select either *Debug > Debug Main Project* or click **Debug Main Project** in the toolbar.

4. Build the selected configuration in the MPLAB X IDE project and program the demonstration board.

The SPI Driver configures the SPI1 as Master and SPI2 as Slave on the same PIC32 device. The SPI1 and SPI2 lines are connected to each other as described in Configuring the Hardware. The Master sends a chunk of data (64 bytes) to the Slave. The data is sent and received back on the same PIC32 device through the SPI. The data is looped back to the sender.

Upon execution of the program, the SPI Master (SPI1) will transmit a sequence of character string/data through the SPI channel using the settings defined in the application to SPI1.

After transmitting the data from SPI1, the driver will read SPI2 for any data received. if the data is received, the program will verify the validity of the received data. Based on the verification result, the program will go either to a success or error state.

If an error occurs, or if the transmitted data is not the same as the received data, LED2 (Yellow) of the starter kit will illuminate, which indicates the demonstration has failed.

If the transmitted data is exactly the same as the received data, LED3 (Green) of the starter kit will illuminate, which indicates the demonstration was successful.

spi_multislave

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

In this demonstration application, the SPI driver is used to demonstrate the single Master and multiple Slaves capability of the SPI protocol using the DRV_SPI_ClientConfigure function of the SPI Driver to switch between slaves.

There are two data transfers in this demonstration, where the Master transfers a chunk of data to each Slave.

The Slaves are selected using the Slave Select (/SSx) pins, which means that when the Slave /SSx pin is active-low, only the Slave can receive the data. Therefore, while one Slave is receiving the data, the other Slave is kept idle by making the /SSx pin of that Slave high.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the SPI Multi-slave Demonstration.

Description

To build this project, you must open the `spi_multislave.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/driver/spi/spi_multislave`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`spi_multislave.X`	`<install-dir>/apps/driver/spi/spi_multislave/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mz_ef_sk_int_dyn	pic32mz_ef_sk	Demonstrates SPI multi-slave on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit configured for Interrupt mode and dynamic operation.
pic32mz_ef_sk_int_dyn_16b	pic32mz_ef_sk	Demonstrates SPI multi-slave on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit configured for Interrupt mode and dynamic operation in microMIPS mode.
pic32mz_ef_sk_int_dyn_freertos	pic32mz_ef_sk	Demonstrates SPI multi-slave on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit configured for Interrupt mode and dynamic operation with FreeRTOS.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

Required Hardware

This demonstration requires the following hardware:

- PIC32MZ EF Starter Kit
- PIC32MZ Starter Kit Adapter Board
- Starter Kit I/O Expansion Board

SPI1, SPI2 and SPI3 modules are used in this demonstration as Master, Slave 1 and Slave 2 respectively. In addition, two GPIO pins are used to control the Slave Select (/SSx) pins.

PIC32MZ devices support the Peripheral Pin Select (PPS) feature. The SPI Pins of the SPI modules on this device must be configured using the PPS.

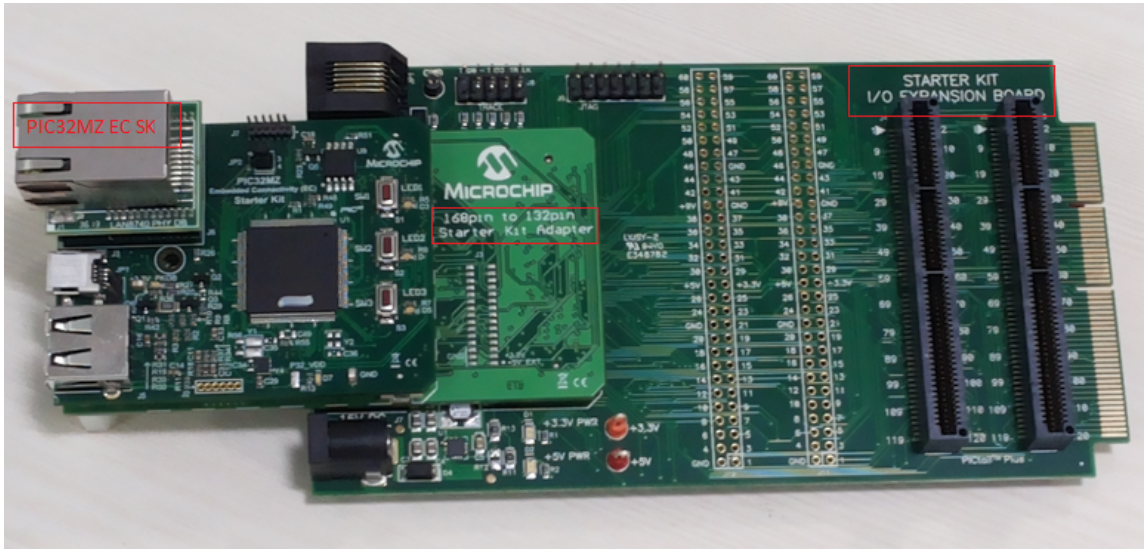
Note: To know more about the PIC32MZ PPS feature and pin configuration, please refer to **Section 12.3 "Peripheral Pin Select (PPS)"** in the **"I/O Ports"** chapter of the specific device data.

PIC32MZ EF Starter Kit Configuration

1. Short the pins on the J10 and J11 headers of the Starter Kit I/O Expansion Board, as follows:

- Pin 41 (J10), pin 23 (J10) and pin 48 (J11): SCK1, SCK2, and SCK3
- Pin 43 (J10), pin 24 (J10) and pin 52 (J11): SDO1, SDI2, and SDI3
- Pin 44 (J10), pin 25 (J10) and pin 37 (J11): SDI1, SDO2, and SDO3
- Pin 34 (J10) and pin 26 (J10): RH10 and SS2
- Pin 33 (J10) and pin 46 (J11): RH15 and SS3

2. Next, attach the starter kit and the Starter Kit I/O Expansion Board, as shown in the following figure.



PIC32MZ EF Pin Mapping

The following table illustrates how the SPI1, SPI2, and SPI3 pins for the PIC32MZ EF devices are mapped and connected to each other through the Starter Kit I/O Expansion Board.

Module	Function	Device Pin #	Port Pin Name	Starter Kit I/O Expansion Board Pin #	Demonstration Setup Connection
SPI1	SCK1	109	SCK1	41 (J10)	SCK2, SCK3
	SDI1	69	RPD14	44 (J10)	SDO2, SDO3
	SDO1	98	RPD10	43 (J10)	SDI2, SDI3
	/SS1	Not Used	Not Used	Not Used	Not Used
SPI2	SCK2	14	SCK2	23 (J10)	SCK1
	SDI2	121	RPD7	24 (J10)	SDO1
	SDO2	25	RPB5	25 (J10)	SDI1
	/SS2	104	RPD0	26 (J10)	RH10
SPI3	SCK3	61	SCK3	48 (J11)	SCK1
	SDI3	49	RPB10	52 (J11)	SDO1
	SDO3	96	RPA15	37 (J11)	SDI1
	/SS3	62	RPB15	46 (J11)	RH15
GPIO (for Slave Select)	Output	83	RH10	34 (J10)	SS2
	Output	103	RH15	33 (J10)	SS3

Running the Demonstration

This section demonstrates how to run the SPI Driver SPI Multi-slave Demonstration.

Description

Once the demonstration application is compiled successfully, you are ready to program the firmware in the target device.

To run the demonstration in Debug mode, perform the following steps:

1. Select the appropriate configuration from the MPLAB X IDE Project Properties based on the target hardware.
2. Select your device programmer from the Hardware Tool menu in MPLAB X IDE.
3. Select either *Debug > Debug Main Project* or click **Debug Main Project** in the toolbar.

Once the device is successfully programmed, you can observe that the LEDs in the starter kit have turned ON. This indicates that the demonstration project ran successfully. The result of the programming can be read through the LEDs.

The SPI Driver configures SPI1 as the Master and SPI2 and SPI3 as Slaves on the same PIC32 device. The SPI1, SPI2 and SPI3 lines are connected to each other as described in Configuring the Hardware.

The Master sends a chunk of data (numbers 0 to 63, total 64 bytes) to the Slave 1 in the first transfer and sends one more chunk of data (numbers 64 to 1, total 64 bytes) to the Slave 2 in the second transfer.

The respective Slaves are selected in the each transfer using the /SS2 and /SS3 pins, which are driven by the GPIO pins RH10 and RH15, respectively.

The following table explains the states of SPI Slaves in different stages of the application:

Application State	RH10 State (Connected to /SS2)	RH15 State (Connected to /SS3)	Slave 1	Slave 2
Initialization	High	High	Inactive	Inactive
Transfer 1	Low	High	Active	Inactive
Transfer 2	High	Low	Inactive	Active
Idle	High	High	Inactive	Inactive

After the completion of both transfers, the received data for both of the slaves is verified with the transmitted data:

- If the data received by SPI2 matches with the data transmitted in the first transfer, LED2 (YELLOW) will illuminate
- If the data received by SPI3 matches with the data transmitted in the second transfer, LED3 (GREEN) will illuminate
- If both the transfers were not completed or the data of any Slave does not match, LED1 (RED) will illuminate

spi_self_loopback

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This application demonstrates how to use the SPI Driver for a simple self loopback operation. Although this application may not be a practical use

case scenario, in general, it can be used to understand how to use MPLAB Harmony drivers, and in particular the SPI driver for a write-read operation. It also demonstrates how callbacks work for the SPI Driver.

Demonstration Features

SPI Driver Library

Tools Setup Differences

This demonstration has multiple configurations for different devices. Configurations that are for devices with the pin remapping feature have their SPI pins mapped in the Pin Setting window of MHC as per the device and board specifications.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the SPI Self-Loopback demonstration.

Description

To build this project, you must open the `spi_self_loopback.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/driver/spi/spi_self_loopback`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`spi_self_loopback.X`	`<install-dir>/apps/driver/spi/spi_self_loopback/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mz_ef_sk_int_dyn	pic32mz_ef_sk	Demonstrates the SPI loopback on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit and the Starter Kit I/O Expansion Board combination using the SPI Driver in Dynamic mode.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MZ EF Starter Kit and Starter Kit I/O Expansion Board

- Using wires, connect the SDI2 pin (J11 pin 32) and SDO2 pin (J10 pin 35) on the Starter Kit I/O Expansion Board
- Connect the PIC32MZ EF Starter Kit with the adapter board to the Starter Kit I/O Expansion Board

Running the Demonstration

This section demonstrates how to run the SPI Driver SPI Self-Loopback demonstration.

Description

This demonstration loops back data between the SDI and SDO pins of an SPI module.

1. First, compile and program the target device. While compiling, select the configuration suitable for the hardware in use.
2. Check the state of the LEDs on the board:
   - On the PIC32MZ EF Starter Kit, if LED3 illuminates, the demonstration was successful; otherwise, if LED1 is illuminated, this indicates the demonstration has failed.

SPI Flash Driver Demonstrations

This topic provides descriptions of the SST25VF020B SPI Flash Driver demonstrations.

Introduction

This help file contains instructions and associated information about the MPLAB Harmony SPI Flash Driver application demonstration, which is included in the MPLAB Harmony Library distribution.

Description

This application demonstrates the capabilities of the MPLAB Harmony SPI Flash Driver. This section describes the hardware requirements and procedures to build and run the demonstration project using Microchip development tools.

Demonstrations

This topic provides information on how to run the SPI Flash Driver demonstration application included in this release.

sst25vf020b

This demonstration uses the SST25VF020B SPI Flash Driver to erase, write, and read from the on-board SST25VF020B Flash through SPI and verifies whether or not operation occurred correctly.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the SPI Flash Driver Demonstration.

Description

To build this project, you must open the `sst25vf020b.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/driver/spi_flash/sst25vf020b`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`sst25vf020b.X`	`<install-dir>/apps/driver/spi_flash/sst25vf020b/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
bt_audio_dk_int_dyn	bt_audio_dk	This configuration runs on the PIC32 Bluetooth Audio Development Kit. The SPI Driver and SST25VF20B SPI Flash Driver are configured for Interrupt mode and dynamic operation.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 Bluetooth Audio Development Kit

Ensure that Switch S1 is set to PIC32_MCLR.

Running the Demonstration

This section describes how to run the SPI Flash Driver demonstration.

Description

To run this demonstration:

1. First compile and program the target device. While compiling, select the configuration suitable for hardware.
2. Observe the status of LEDs on the development kit. If LED 8 and LED 9 are illuminated, this indicates the demonstration is working correctly. If

either of LED 5 or LED 6 are illuminated, this indicates the demonstration is not working correctly.

USART Driver Demonstrations

This topic provides descriptions of the USART Driver demonstrations.

Introduction

This section provides instructions and information about the MPLAB Harmony USART Driver demonstration applications, which are included in the MPLAB Harmony Library distribution.

Description

This application demonstrates how to use the MPLAB Harmony USART Driver. This section describes the hardware requirement and procedures to build and execute the demonstration project on Microchip development tools. Two demonstration are provided:

- usart_echo - In this demonstration application, the USART Driver will initially transmit strings of data, and then accept any characters received and transmit the data back. Error or success status is indicated by LED.
- usart_loopback - In this demonstration application, the USART driver "multi-instance multi-client" feature is used. The application uses two USART hardware instances of the same PIC32 device. Application transmits chunk of data (64 bytes) on USART1, receives it at USART2, transmits back from USART2 to USART1. USART1 receives back all the data that was transmitted. The data is looped back to USART1.

To know more about the MPLAB Harmony USART driver, configuring the USART Driver and the APIs provided by the USART Driver, refer to USART Driver Library documentation.

Demonstrations

This topic provides information on how to run the USART Driver demonstration applications included in this release.

usart_echo

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the USART Echo Demonstration.

Description

To build this project, you must open the `usart_echo.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/driver/usart/usart_echo`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`usart_echo.X`	`<install-dir>/apps/driver/usart/usart_echo/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx795_pim_e16_int_dyn	pic32mx795_pim+e16	This demonstration runs on the PIC32MX795F512L PIM and the Explorer 16 Development Board configured for Interrupt mode and dynamic operation.
pic32mx795_pim_e16_int_sta	pic32mx795_pim+e16	This demonstration runs on the PIC32MX795F512L PIM and the Explorer 16 Development Board configured for Interrupt mode and static operation.
pic32mx795_pim_e16_poll_dyn	pic32mx795_pim+e16	This demonstration runs on the PIC32MX795F512L PIM and the Explorer 16 Development Board configured for Polled mode and dynamic operation.
pic32mx795_pim_e16_poll_sta	pic32mx795_pim+e16	This demonstration runs on the PIC32MX795F512L PIM and the Explorer 16 Development Board configured for Interrupt mode and static operation.

pic32mx795_pim_int_dyn_freertos	pic32mx795_pim+e16	FreeRTOS version of this demonstration, which runs on the PIC32MX795F512L PIM and the Explorer 16 Development Board configured for Interrupt mode and dynamic operation.

Configuring the Hardware

This section describes how to configure the supported hardware.

Description

Explorer 16 Development Board

Following are the hardware configuration settings required to execute this demonstration:

1. Before attaching the PIC32MX795F512L PIM to the Explorer 16 Development Board, ensure that the processor select switch (S2) is in the PIM Position.
2. Short JP2 on the Explorer 16 Development Board to enable the LEDs.
3. Use a RS-232 to USB Converter to connect to the computer.

PIC32MX795F512L CAN-USB Plug-in Module (PIM)

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

This section demonstrates how to run the USART Driver Demonstration.

Description

Once the demonstration application has been compiled successfully for the intended configuration, program the firmware into the target device.

Upon execution, the application will transmit the following strings through the USART using the settings defined in the application.

```
**************************************************
Welcome to Microchip USART Driver Demo Application.
Press any character, the character will be echoed back.
Press 'ESC' key to exit the Demo Application.
**************************************************
```

After transmitting the text, the firmware will read any characters received through the USART and it will transmit back the same data. If the received character is "ESC" (0x1B), the program will enter into Idle mode. Once in Idle mode, the firmware will neither transmit nor receive any data. If the application enters Idle mode, LED 5 of the Explorer 16 Development Board will illuminate. If any error occurs, LED 9 of the Explorer 16 Development Board will illuminate.

usart_loopback

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the USART Loopback Demonstration.

Description

To build this project, you must open the `usart_loopback.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/driver/usart/usart_loopback`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`usart_loopback.X`	`<install-dir>/apps/driver/usart/usart_loopback/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx_usb_sk2_int_dyn	pic32mx_usb_sk2	Demonstrates USART loopback on the PIC32 USB Starter Kit II configured for Interrupt mode and dynamic operation.
pic32mx_usb_sk2_int_sta	pic32mx_usb_sk2	Demonstrates USART loopback on the PIC32 USB Starter Kit II configured for Interrupt mode and static operation.
pic32mx_usb_sk2_poll_dyn	pic32mx_usb_sk2	Demonstrates USART loopback on the PIC32 USB Starter Kit II configured for Polled mode and dynamic operation.
pic32mz_ef_sk_int_dyn	pic32mz_ef_sk	Demonstrates USART loopback on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit configured for Interrupt mode and dynamic operation.
pic32mz_ef_sk_poll_dyn	pic32mz_ef_sk	Demonstrates USART loopback on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit configured for Polled mode and dynamic operation.
pic32mz_ef_sk_int_dyn_16b	pic32mz_ef_sk	microMIPS version of the demonstration, which demonstrates USART loopback on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit configured for Interrupt mode and dynamic operation.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MZ EF Starter Kit

Note: For space consideration, only the PIC32MZ EC Starter Kit is shown. However, the following configuration information applies to either starter kit.

This demonstration requires the following hardware:

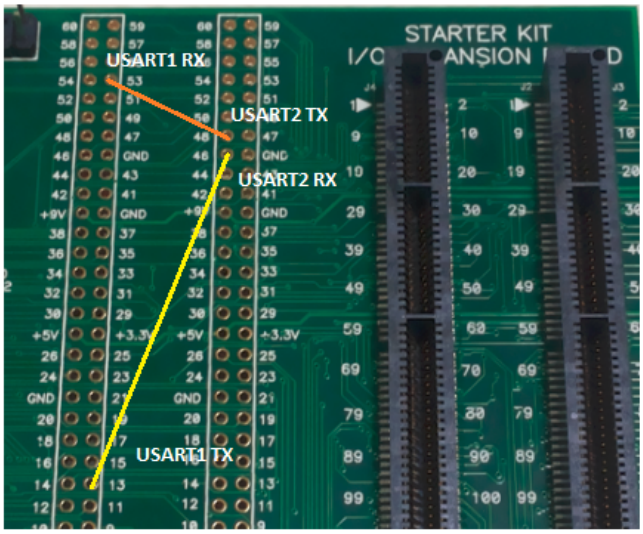
- PIC32MZ EF Starter Kit
- PIC32MZ Starter Kit Adapter Board
- Starter Kit I/O Expansion Board

PIC32MZ EF Starter Kit

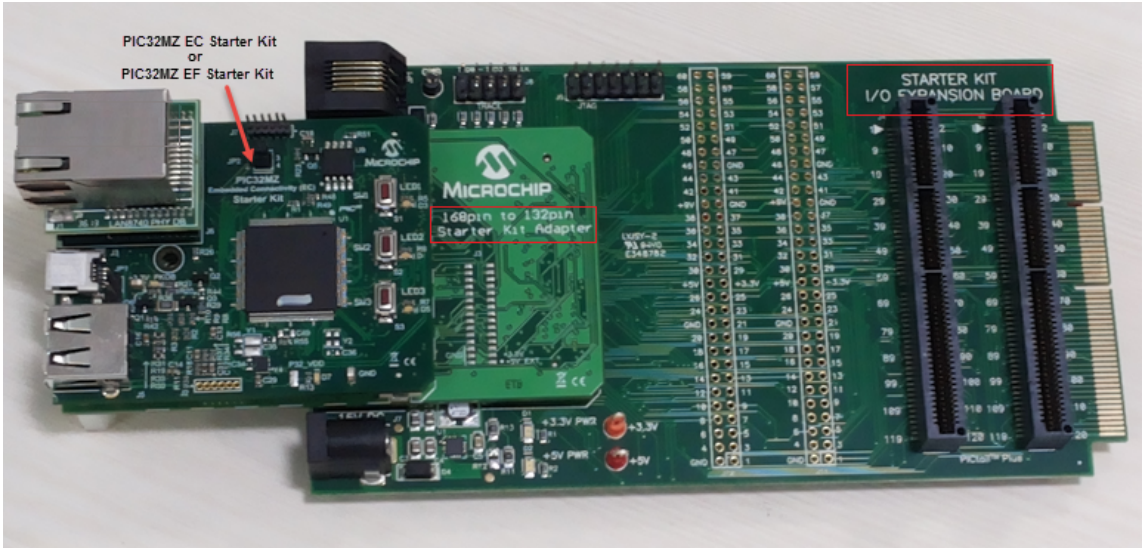
In this demonstration application, the USART1 and USART2 modules are used. PIC32MZ devices support the Peripheral Pin Select (PPS) feature. The USART pins of the USART modules on this device are required to be configured using the PPS.

Note: To learn more about the PIC32MZ PPS feature and pin configuration, please refer to **Section 12.3 "Peripheral Pin Select (PPS)"** in the **"I/O Ports"** chapter of the specific device data sheet. These documents are available for download from the Microchip web site (www.microchip.com).

1. Connect (using wires) the pins on the Starter Kit I/O Expansion Board, as shown in the following figure. This hardwired connection would connect the USART1 lines to the USART2 lines of the PIC32MZ2048EFM144 device on the PIC32MZ EF Starter Kit.



2. Next, connect the PIC32MZ EF Starter Kit, the PIC32MZ Starter Kit Adapter Board, and the Starter Kit I/O Expansion Board, as shown in the following figure.



PIC32MZ EF Starter Kit

The following table illustrates how the USART1 and USART2 pins are mapped and connected to each other through the Starter Kit I/O Expansion Board.

USART Module	USART Lines	PIC32MZ2048EFM Device Pin #	Analog Pin	PIC32MZ2048EFM Port Pin Name/Function	Pin # on I/O Expansion Board J10 Connector	Pin # on I/O Expansion Board Pin J11 Connector	For this Demonstration, attach this pin to:
USART1	TX	13	AN19	PMRD	13	N/A	USART2 RX
USART1	RX	48	AN49	PMA7	53	N/A	USART2 TX
USART2	TX	61	AN9	RPB14	N/A	48	USART1 RX
USART2	RX	62	AN10	RPB15	N/A	46	USART1 TX

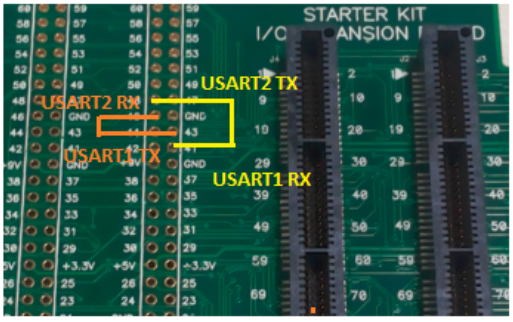
PIC32 USB Starter Kit II

This demonstration requires the following hardware:

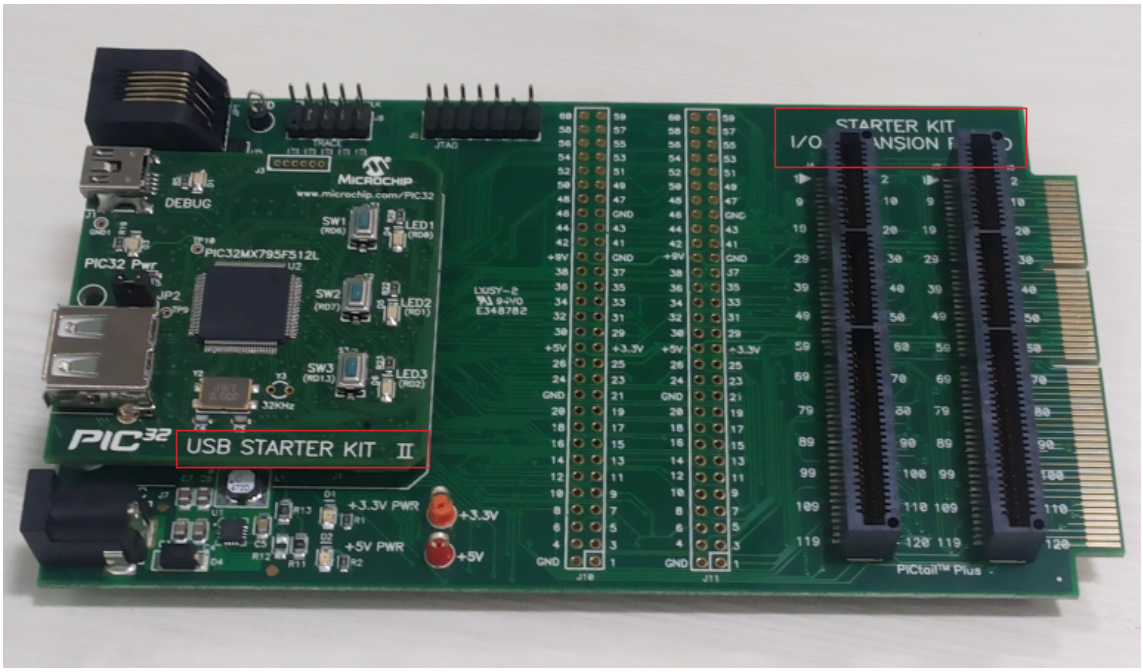
- PIC32 USB Starter Kit II
- Starter Kit I/O Expansion Board

In this demonstration application, the USART and USART2 modules are used.

1. Connect (using wires) the pins on the Starter Kit I/O Expansion Board, as shown in the following figure. This hardwired connection would connect the USART1 lines to the USART2 lines of the PIC32MX795F512L device on the PIC32 USB Starter Kit II through the Starter Kit I/O Expansion Board.



2. Next, connect the PIC32 USB Starter Kit II and the Starter Kit I/O Expansion Board, as shown in the following figure.



For the PIC32MX795F512L device, the USART modules has dedicated I/O pins.

The following table illustrates how the USART1 and USART2 pins are mapped and connected to each other through the Starter Kit I/O Expansion Board.

USART Module	USART Lines	PIC32MX795F512L Device Pin #	PIC32MX795F512L Port Pin Name/Function	Pin # on I/O Expansion Board J11 Connector	For this demonstration, attach this pin to:
USART1	TX	53	U1TX	43	U2RX
USART1	RX	52	U1RX	41	U2TX
USART2	TX	50	U2TX	48	U1RX
USART2	RX	49	U2RX	46	U1TX

Running the Demonstration

This section demonstrates how to run the USART Driver USART Loopback Demonstration.

Description

This demonstration shows how to configure and make use of the USART Driver APIs to support multiple USART hardware instances to multiple clients of the driver in both interrupt mode and polled mode. This demonstration shows the USART driver's "multi-instance multi-client" feature.

Once the demonstration application is compiled successfully, you are ready to program the firmware in the target device.

To run the demonstration in Debug mode, perform the following steps:

1. Select the appropriate configuration from the MPLAB X IDE Project Properties based on the target hardware.
2. Select your device programmer from the Hardware Tool menu in MPLAB X IDE.
3. Select either *Debug > Debug Main Project* or click **Debug Main Project** in the toolbar.
4. Build the selected configuration in the MPLAB X IDE project and program the demonstration board.

The USART Driver configures the USART1 and USART2 modules on the same PIC32 device as defined in the `system_config.h` file. The USART1 and USART2 lines are connected to each other as described in Configuring the Hardware. USART1 sends a chunk of data (64 bytes) to USART2. USART2 sends back the same data to USART1. The data is sent and received back on the same USART module through another USART. The data is looped back to the sender.

Upon execution of the program, the USART1 will transmit a sequence of character string/data through the USART1 TX channel using the settings defined in the application for USART1.

After transmitting the data from USART1, the driver will read USART2 for any data received. If the data is received, the same data is sent back to USART1. The program will verify the received data to the transmitted data and enter into Idle mode.

Once in Idle mode, the firmware will neither transmit nor receive any data. If the application enters Idle mode, LED2 (Yellow) of the starter kit will illuminate.

If any error occurs, or if the transmitted data is not same to received data, LED1 (Red) of the starter kit will illuminate, which indicates the demonstration has failed.

If the transmitted data and received data on UART1 is same, LED3 (Green) of the starter kit will illuminate, which indicates the demonstration was

successful.

Examples

Provides information on MPLAB Harmony example applications. The MPLAB Harmony example applications provide simple single-purpose application projects that illustrate how to use a selected MPLAB Harmony library. These are the working examples of source code that are provided throughout the MPLAB Harmony documentation.

my_first_app

Describes the my_first_app example.

Peripheral Library Examples

This topic describes the peripheral library examples.

Introduction

The example applications for MPLAB peripheral libraries (PLIBs) provide very simple single-purpose examples of how to use MPLAB Harmony peripheral libraries.

Description

Peripheral libraries (PLIBs) are the lowest level libraries provided with MPLAB Harmony. They provide a functional abstraction of the peripherals available on Microchip microcontrollers to hide differences in register details that can exist from device to device. However, they do not maintain any state data (at least not any that isn't stored in the hardware registers) from call to call and they do not provide any protection of the peripheral's resources. As such, any code that calls the PLIB for a peripheral must take responsibility for ownership of that peripheral and is responsible for managing the behavior of that peripheral.

As such, PLIBs are normally only used by MPLAB Harmony device drivers or system services. However, there are some times when it is necessary and appropriate to interact with a PLIB directly from an application. Therefore, simple examples are provided to show how to use the interfaces to the peripheral libraries. These examples are available in the MPLAB Harmony installation in the following location: `<install-dir>/apps/examples/peripheral`.

PIC32MX device examples are written for the Explorer16 Development Board with a PIC32MX795F512L PIM. PIC32MZ device examples are written for the PIC32MZ Embedded Connectivity (EC) Starter Kit. The examples are tested and working on the Explorer 16 Development Board, the PIC32 USB Starter Kit II, and the PIC32MZ Embedded Connectivity (EC) Starter Kit, and the appropriate board configuration for each project can be selected in MPLAB X IDE.

ADC Peripheral Library Examples

This topic provides descriptions of the ADC Peripheral Library examples.

Introduction

ADC Peripheral Library Demonstration Applications Help

Description

This distribution package contains one ADC related firmware project that demonstrates the capabilities of the MPLAB Harmony ADC Peripheral Library.

This section describes the hardware requirement and procedures to run this firmware project on Microchip demonstration and development boards.

To learn more about MPLAB Harmony stacks and libraries refer to the related documentation in Volume V: MPLAB Harmony Framework Reference.

Demonstrations

This topic provides information on how to run the ADC Peripheral Library demonstration applications included in this release.

adc_pot

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This example reads the potentiometer, R6 on the Explorer 16 Development Board using a PIC32MX795F512L PIM. The results of the read are displayed on the LEDs. As the potentiometer is adjusted, the LEDs displayed will "slide" up or down.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the ADC Peripheral Library demonstration.

Description

To build this project, you must open the `adc_pot.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/examples/peripheral/adc/adc_pot`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`adc_pot.X`	`<install-dir>/apps/examples/peripheral/adc/adc_pot/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx795_pim_e16	pic32mx795_pim+e16	Demonstrates the ADC reading potentiometer R6 on the PIC32MX795F512L PIM and the Explorer 16 Development Board.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MX795F512L CAN-USB PIM, Explorer 16 Development Board, and Starter Kit I/O Expansion Board

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the ADC demonstration.

Description

This demonstration reads the setting of the potentiometer through the ADC.

1. First compile and program the target device. While compiling, select the available configuration for the hardware in use.
2. Turn the potentiometer R6 in either direction. Check the state of the LEDs on the Explorer 16 Development Board. As the potentiometer is turned clockwise, more LEDs will be lit. As the potentiometer is turned counter-clockwise, fewer LEDs are lit.

adc_pot_dma

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This example reads the potentiometer, R6 on the Explorer 16 Development Board (PIC32MX795F512L PIM) using the ADC while storing data on RAM using DMA.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the ADC Peripheral Library demonstration.

Description

To build this project, you must open the `adc_pot_dma.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/examples/peripheral/adc/adc_pot_dma`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
adc_pot_dma.X	<install-dir>/apps/examples/peripheral/adc/adc_pot_dma/firmware

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within ./firmware/src/system_config.

Project Configuration Name	BSP Used	Description
pic32mx795_pim_e16	pic32mx795_pim+e16	Demonstrates the ADC using a DMA channel to read the potentiometer R6 on the PIC32MX795F512L PIM and the Explorer 16 Development Board.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MX795F512L CAN-USB PIM, Explorer 16 Development Board, and Starter Kit I/O Expansion Board

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the ADC demonstration.

Description

This demonstration reads the setting of the potentiometer through the ADC.

1. First compile and program the target device. While compiling, select the available configuration for the hardware in use.
2. Turn the potentiometer R6 in either direction. Check the state of the LEDs on the Explorer 16 Development Board. As the potentiometer is turned clockwise, more LEDs will be lit. As the potentiometer is turned counter-clockwise, fewer LEDs are lit.

12-bit High-Speed SAR ADC (ADCHS) Peripheral Library Examples

This topic provides descriptions of the 12-bit High-Speed SAR ADC (ADCHS) Peripheral Library examples.

Introduction

12-bit High-Speed SAR ADC (ADCHS) Peripheral Library Demonstration Applications Help

Description

This distribution package contains one ADC-related firmware project that demonstrates the capabilities of the MPLAB Harmony ADCHS Peripheral Library.

This section describes the hardware requirement and procedures to run this firmware project on Microchip demonstration and development boards.

To learn more about MPLAB Harmony stacks and libraries refer to the related documentation in Volume V: MPLAB Harmony Framework Reference.

Demonstrations

This topic provides information on how to run the ADCHS Peripheral Library demonstration applications included in this release.

adchs_3ch_dma

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This example demonstrates the use of system DMA to automatically store conversion data of three ADC channels into three separate buffers. Each channel is set to sample at a conversion rate of 2 Msps using a single timer trigger to synchronize the conversion of all three channels. An average converted data displays on a terminal emulator, such as RealTerm.

This application is to be used with the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit using the Starter Kit I/O Expansion Board with the Adapter Board.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the ADCHS Peripheral Library demonstration.

Description

To build this project, you must open the `adchs_3ch_dma.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/examples/peripheral/adchs/adchs_3ch_dma`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`adchs_3ch_dma.X`	`<install-dir>/apps/examples/peripheral/adchs/adchs_3ch_dma/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mz_ef_sk	pic32mz_ef_sk	Demonstrates operation of three system DMA channels to store conversion of a Class 1 analog input (potentiometer) when connected to the PIC32MZ EF Starter Kit using the Starter Kit I/O Expansion Board and the PIC32MZ EC Starter Kit Adapter Board.
pic32mz_da_sk_intddr	pic32mz_da_sk_intddr	Demonstrates operation of three system DMA channels to store conversion of a Class 1 analog input when connected to the PIC32MZ DA Starter Kit.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit

1. Connect the PIC32MZ EF Starter Kit to the PIC32MZ EC Starter Kit Adapter Board.
2. Connect the combination of two boards (as described in Step 1) on the Starter Kit I/O Expansion Board.
3. A 3 analog signal or 3 potentiometer of suitable value (4.7K or 10K) should be connected with two ends to 3.3V and the GND pin of J11; and wiper of potentiometer to J11 pin 34 and 33 (AN0 and AN1) and J10 pin 32 (AN2) on the Starter Kit I/O Expansion Board.
4. Connect a mini-USB cable from a PC to the mini-USB connector J11 on the PIC32MZ EF Starter Kit.
5. Power the Starter Kit I/O Expansion Board with a 9V power supply.

PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit

1. A 3 analog signal or 3 potentiometer of suitable value (4.7K or 10K) should be connected with two ends to 3.3V and the GND pin of J11; and wiper of potentiometer to J15 pin 35, 27, and 16 (AN0, AN1, and AN2) on the PIC32MZ EF Starter Kit.
2. Power and download firmware to the Starter Kit by connecting a mini-USB cable from a PC to the mini-USB connector J19 (DEGUG) on the PIC32MZ EF Starter Kit.
3. Connect a mini-USB cable from a PC to the mini-USB connector J11 on the PIC32MZ EF Starter Kit.

Running the Demonstration

Provides instructions on how to build and run the ADCHS Peripheral Library demonstration.

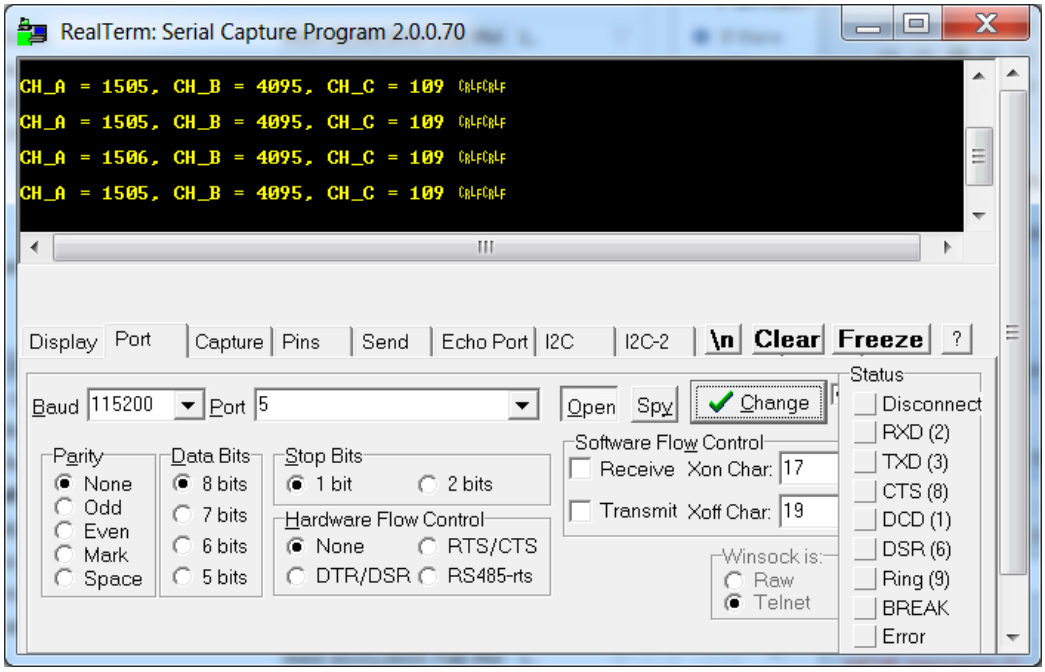
Description

To build this project, you must open the `adchs_3ch_dma.X` project in MPLAB X IDE, and then select the desired configuration.

Do the following to run the demonstration:

1. First compile and program the target device. While compiling, select the available configuration for the hardware in use.
2. Launch a terminal emulator (RealTerm/Tera Term, etc.,) and select the appropriate COM port and set the serial port settings to 115200-N-1.
3. Launch the demonstration. The following message will appear in the console window: Result: 3171.
4. This example demonstrates the use of system DMA to automatically store conversion data of three ADC channels into three separate buffers. Each channel is set to sample at a conversion rate of 2 Msps using a single timer trigger to synchronize the conversion of all three channels. An average converted data displays on a terminal emulator, such as RealTerm.

The following figure shows a working terminal emulator window.



adchs_oversample

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This example demonstration reads the potentiometer that is connected to the PIC32MZ DA Starter Kit or the PIC32MZ EF Starter Kit using the Starter Kit I/O Expansion Board with the PIC32MZ EC Starter Kit Adapter Board, and performs oversampling of the converted data and displays the higher resolution ADC converted readings on a terminal emulator (such as RealTerm).

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the ADCHS Peripheral Library demonstration.

Description

To build this project, you must open the `adchs_oversample.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/examples/peripheral/adchs/adchs_oversample`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`adchs_oversample.X`	`<install-dir>/apps/examples/peripheral/adchs/adchs_oversample/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mz_ef_sk	pic32mz_ef_sk	Demonstrates operation of a digital filter in Oversampling mode, while converting a Class 1 analog input (potentiometer) when connected to the PIC32MZ EF Starter Kit using the Starter Kit I/O Expansion Board and the PIC32MZ EC Starter Kit Adapter Board.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit

1. Connect the PIC32MZ EF Starter Kit to the PIC32MZ EC Starter Kit Adapter Board.
2. Connect the combination of two boards (as described in Step 1) on the Starter Kit I/O Expansion Board.
3. A potentiometer of suitable value (4.7K or 10K) should be connected with two ends to 3.3V and the GND pin of J11; and wiper of potentiometer to pin 34 of J11 on the Starter Kit I/O Expansion Board.
4. Connect a mini-USB cable from a PC to the mini-USB connector J11 on the PIC32MZ EF Starter Kit.
5. Power the Starter Kit I/O Expansion Board with a 9V power supply.

Running the Demonstration

Provides instructions on how to build and run the ADCHS Peripheral Library demonstration.

Description

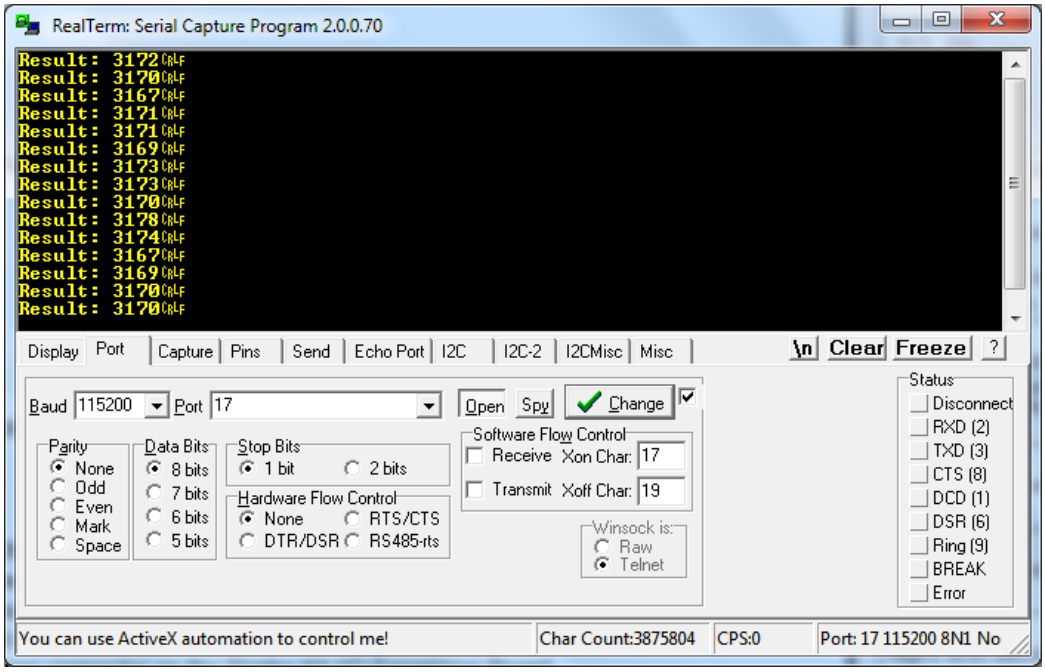
This demonstration oversamples the converted data of the potentiometer using digital filters. Do the following to run the demonstration:

1. First compile and program the target device. While compiling, select the available configuration for the hardware in use.
2. Launch a terminal emulator (RealTerm/Tera Term, etc.,) and select the appropriate COM port and set the serial port settings to 115200-N-1.
3. Launch the demonstration. The following message will appear in the console window: Result: 3171.
4. As the potentiometer is turned, the "Result:" on the console shows the high resolution ADC converted value.
5. The higher resolution data can be verified by first running the adchs_pot demonstration and observing the 12-bit converted data. Later, keeping the potentiometer setting same, the adchs_oversample demonstration is run and the higher resolution 15-bit converted data can be observed and verified. Theoretical calculation is as follows:
   - Max count of 12 bit resolution: 4096
   - Max count of 15 bit resolution: 32768
   - ADC Ref input voltage: 3.3 Volts

Consider an analog input of 0.319 Volts:

- 12-bit resolution ADC reading (obtained by adchs_pot): ((4096/3.3) * 0.319) = 396
- 15-bit resolution ADC reading (oversampled by adchs_oversample): ((32768/3.3) * 0.319) = 3167

The following figure shows a working terminal emulator window.



adchs_pot

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This example demonstration reads the potentiometer that is connected to the PIC32MZ DA Starter Kit or the PIC32MZ EF Starter Kit using the Starter Kit I/O Expansion Board with the PIC32MZ EC Starter Kit Adapter Board or the PIC32WK WiFi Starter Kit.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the ADCHS Peripheral Library demonstration.

Description

To build this project, you must open the `adchs_pot.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/examples/peripheral/adchs/adchs_pot`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`adchs_pot.X`	`<install-dir>/apps/examples/peripheral/adchs/adchs_pot/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mz_ef_sk	pic32mz_ef_sk	Demonstrates a read of the potentiometer by the ADC when connected to the PIC32MZ EF Starter Kit using the Starter Kit I/O Expansion Board and the PIC32MZ EC Starter Kit Adapter Board.
pic32wk_sk	pic32wk_gpb_gpd_sk+module	Demonstrates a read of the potentiometer by the ADC when connected to the PIC32WK WiFi Starter Kit.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit

1. Connect the PIC32MZ EF Starter Kit to the PIC32MZ EC Starter Kit Adapter Board.
2. Connect the combination of two boards (as described in Step 1) on the Starter Kit I/O Expansion Board.
3. A potentiometer of suitable value (4.7K or 10K) should be connected with two ends to 3.3V and the GND pin of J11; and wiper of potentiometer to pin 34 of J11 on the Starter Kit I/O Expansion Board.
4. Connect a mini-USB cable from a PC to the mini-USB connector J11 on the PIC32MZ EF Starter Kit.
5. Power the Starter Kit I/O Expansion Board with a 9V power supply.

PIC32WK WiFi Starter Kit

No hardware related configuration or jumper setting changes are necessary.

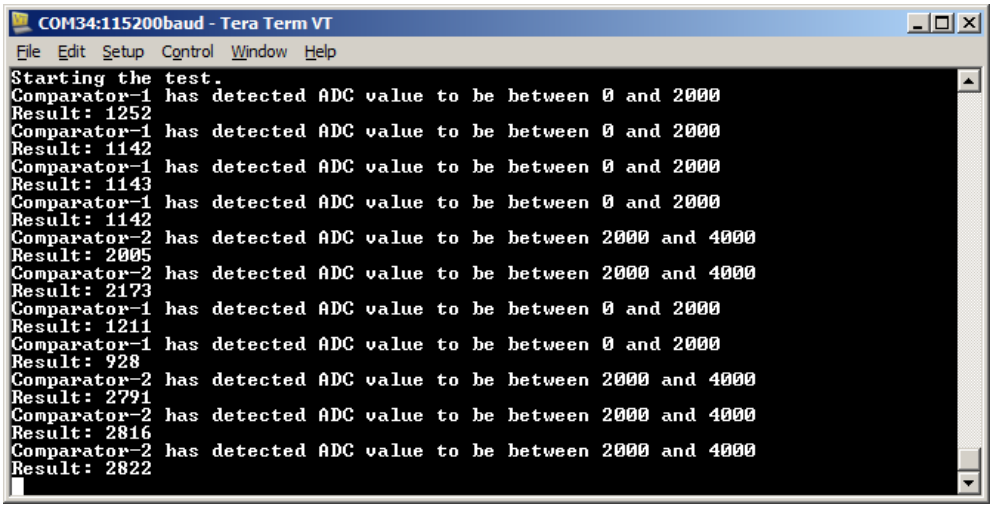
Running the Demonstration

Provides instructions on how to build and run the ADCHS Peripheral Library demonstration.

Description

This demonstration reads the setting of the potentiometer through the ADC.

1. First compile and program the target device. While compiling, select the available configuration for the hardware in use.
2. Launch a terminal emulator (RealTerm/Tera Term, etc.,) and select the appropriate COM port and set the serial port settings to 115200-N-1.
3. Launch the demonstration. The following messages will appear in the console window:
   - Comparator 1 has detected the ADC value to be between 0 and 2000
   - Result: 960
4. As the potentiometer is turned, the "Result:" on the console shows the ADC converted value. Two digital comparators process the ADC converted value and generate an event. The conditions for digital comparator events are as follows:
   - Digital Comparator 1 generates an event when the ADC converted value is between 0 to 2000. Also, LED D1 on the starter kit turns ON.
   - Digital Comparator 2 generates an event when the ADC converted value is between 2000 to 4000. Also, LED D2 on the starter kit turns ON.
5. The following figure shows a working terminal emulator window.



adchs_sensor

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This peripheral library example demonstrates the conversion of three Class 2 and one Class 3 analog inputs in Channel Scan mode, triggered by a Timer3 match. On the MEB II board, the chosen three Class 2 analog inputs are connected to the three axes of an accelerometer and the Class 3 analog input is connected to a temperature sensor. The demonstration application code uses the converted value of the three axes of the accelerometer, converts them into the tilt of the three axes, and displays the results on the serial port terminal (in radians). Also, the converted data from the temperature sensor is scaled to degrees Celsius and displayed on the serial port terminal.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the ADCHS Peripheral Library demonstration.

Description

To build this project, you must open the `adchs_sensor.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/examples/peripheral/adchs/adchs_sensor`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`adchs_sensor.X`	`<install-dir>/apps/examples/peripheral/adchs/adchs_sensor/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mz_ef_sk_meb2	pic32mz_ef_sk+meb2	Demonstrates the scan conversion feature with a timer trigger, while converting three Class 2 analog inputs (accelerometer) and one Class 3 analog input (temperature sensor) when connected to the PIC32MZ EF Starter Kit on the MEB II.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MZ EF Starter Kit connected to the MEB II

1. Mount the PIC32MZ EF Starter Kit on the MEB II.
2. Connect a mini-USB cable from a PC to the mini-USB connector J11 on the PIC32MZ EF Starter Kit.
3. Power the Starter Kit I/O Expansion Board with a 9V power supply on the MEB II or through the Debug connector (J3) on SK.

Note: The accelerometer y and z axes are connected to AN7 (RB12) and AN8 (RB13) on the PIC32MZ EF Starter kit. The same pins are also used for switch inputs, SW1 and SW2. During the application initialization, the ports are configured as analog. While running this demonstration care should be taken not to press SW1 or SW2. Otherwise, the converted ADC values will be incorrect.

Running the Demonstration

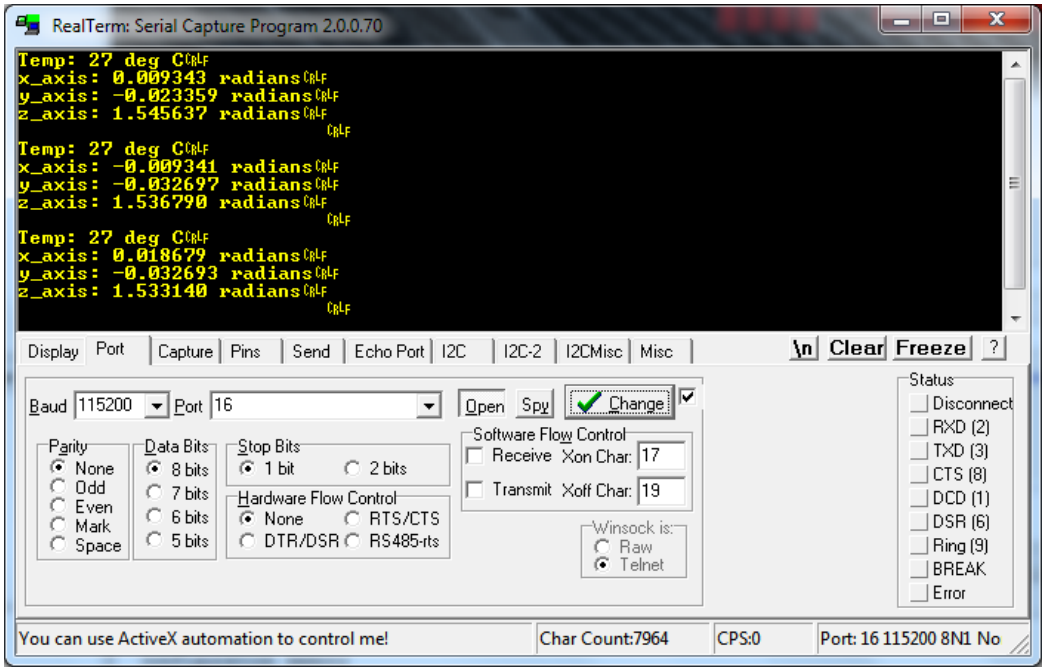
Provides instructions on how to build and run the ADCHS Peripheral Library demonstration.

Description

This demonstration reads the accelerometer and temperature sensor. Do the following to run the demonstration:

1. First compile and program the target device. While compiling, select the available configuration for the hardware in use.
2. Launch a terminal emulator (RealTerm/Tera Term, etc.,) and select the appropriate COM port and set the serial port settings to 115200-N-1.
3. Launch the demonstration. The following messages will appear in the console window:
   - Temp: 27 deg C
   - x-axis: 0.001 radians
   - y-axis: 0.01 radians
   - z-axis: 1.52 radians
4. As the MEB II board is tilted, the angles for each axis displayed on the console windows changes.
5. If the temperature sensor "U8" is heated (by touching), the displayed temperature on the console window changes.

The following figure shows a working terminal emulator window.



adchs_touchsense

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This peripheral library example demonstrates the Capacitive Voltage Divider (CVD) feature of the ADCHS Peripheral Library and analog channel scan features using the PIC32MZ EF Starter Kit and the touch pad (B2) on the Multimedia Expansion Board II (MEB II).

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the ADCHS Peripheral Library demonstration.

Description

To build this project, you must open the `adchs_touchsense.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/examples/peripheral/adchs/adchs_touchsense`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
adchs_touchsense.X	<install-dir>/apps/examples/peripheral/adchs/adchs_touchsense/firmware

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mz_ef_sk_meb2	pic32mz_ef_sk+meb2	Demonstrates the CVD feature of ADCHS while converting a Class 3 analog input (AN28), in Scan mode when connected to the PIC32MZ EF Starter Kit on the MEB II.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MZ EF Starter Kit and MEB II

1. Mount the PIC32MZ EF Starter Kit on the MEB II.
2. Connect a mini-USB cable from a PC to the mini-USB connector J11 on the PIC32MZ EF Starter Kit.
3. Power the Starter Kit I/O Expansion Board with a 9V power supply on the MEB II or through the debug connector (J3) on the PIC32MZ EF Starter Kit.

Running the Demonstration

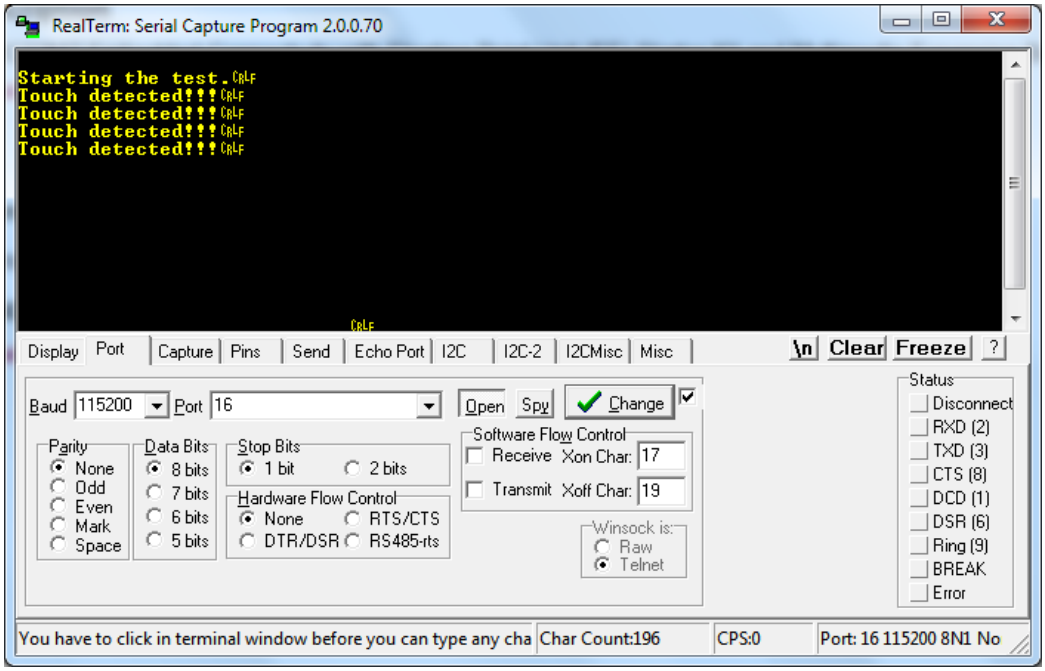
Provides instructions on how to build and run the ADCHS Peripheral Library demonstration.

Description

This example demonstrates the CVD feature of ADCHS and senses the touch event on the B2 touchpad of the MEB II.

1. First compile and program the target device. While compiling, select the available configuration for the hardware in use.
2. Launch a terminal emulator (RealTerm/Tera Term, etc.,) and select the appropriate COM port and set the serial port settings to 115200-N-1.
3. Launch the demonstration.
4. Touch the B2 touch pad on the MEB II. The B2 touch pad is located on the LCD side of the MEB II and not on the starter kit side of MEB II. Once touched, the LED D4 on the MEB II illuminates and "Touch Detected!!!" is displayed on the serial terminal program. This display repeats every 1 second, until B2 is no longer being touched.
5. Once B2 is not longer being touched, the LED D4 turns off and after a brief delay, LED D3 illuminates.

The following figure shows a working terminal emulator window.



Pipelined ADC (ADCP) Peripheral Library Examples

This topic provides descriptions of the Pipelined Analog-to-Digital Converter (ADCP) Peripheral Library examples.

Introduction

Pipelined ADC Peripheral Library Demonstration Applications Help

Description

This distribution package contains one ADC related firmware project that demonstrates the capabilities of the MPLAB Harmony Pipelined ADC Peripheral Library.

This section describes the hardware requirement and procedures to run this firmware project on Microchip demonstration and development boards.

To learn more about MPLAB Harmony stacks and libraries refer to the related documentation in Volume V: MPLAB Harmony Framework Reference.

Demonstrations

This topic provides information on how to run the Pipelined ADC Peripheral Library demonstration applications included in this release.

adcp_cal

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This example illustrates how to configure the Pipelined ADC prior to using it for normal operations. It also illustrates the required errata setup of channel, oversampling, and DMA transfer of data to achieve the best results.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Pipelined ADC calibration demonstration.

Description

To build this project, you must open the `adcp_cal.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/examples/peripheral/adcp/adcp_cal`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`adcp_cal.X`	`<install-dir>/apps/examples/peripheral/adcp/adcp_cal/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mz_ec_pim_e16	pic32mz_ec_pim+e16	Using the PIC32MZ2048ECH100 Plug-in Module (PIM) connected to the Explorer 16 Development Board, this demonstration illustrates a software calibration step that is needed before using the Pipelined ADC. This calibration calculates the DC offset by sampling the internal voltage reference (IVREF) attached to the ADC.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MZ2048ECH100 PIM, Explorer 16 Development Board, and Starter Kit I/O Expansion Board

The software is configured to use an external VREF+ and VREF- attached to pins 33 and 32, respectively, of the PIM. This can be accomplished through the Starter Kit I/O Expansion board or a PICtail™ Prototyping Daughter Board.

The VREF+ pin voltage needs to be applied either after, or at the same time as the main VDD power is applied. The software is set to accept 3.3V as the VREF+ voltage. The software is set to accept 0.0V as the VREF- voltage, so that pin may be connected to GND or AVSS.

If another voltage is used for VREF+ or VREF-, the corresponding setting in the software, in `adcp.h`, needs to be updated to make the calculations correct.

The potentiometer R6 on the Explorer 16 Development Board can be used to drive one of the inputs in the demonstration. Run a jumper between RB5 and RG8 on the PICtail prototyping daughter board, inserted into the PICtail slot. This board can also be used to set up the VREF+ and VREF- connections.

VREF+ can be set up by connecting RB9 on the PICtail prototyping daughter board to +3.3V. VREF- can be set up by connecting RB8 on the PICtail Prototyping Daughter Board.

Running the Demonstration

Provides instructions on how to build and run the Pipelined ADC demonstration.

Description

This demonstration shows how the Pipelined ADC module operates on PIC32MZ devices in accordance with the Errata and Data Sheet clarifications.

1. Connect and program the device using your debugger.
2. Set a breakpoint in `app.c`, near line 234, which reads:
   ```
   appData.state = APP_STATE_NORMALIZE_DATA;
   ```
3. Set a breakpoint in `app.c`, near line 241, which reads:
   ```
   appData.state = APP_STATE_DISPLAY_DATA;
   ```
4. Run the program.
5. When the program reaches the first breakpoint, you can use the debugger variables window to observe the data in `appData.ADC_Data1`.
6. Continue running the program from the breakpoint.
7. When the program reaches the second breakpoint, you can use the debugger variables window to observe the data in `appData.ADC_Data1`. This data has now been normalized, meaning that the values have been converted to 8-bit results. This would be the data an application would use.
8. Continue running the program to collect a new set of data, repeating Steps 5 through 7, as desired. If you have connected the jumper to allow potentiometer R6 operation, turn the potentiometer and observe the change in results.
9. The scale of the readings on the channel connected to the potentiometer is also reflected on the LEDs of the Explorer 16 Development Board.

Note: Timer3 is used to trigger the ADC scan automatically. If the number of channels to be used is changed, the Timer3 period needs to be adjusted accordingly.

Do the following to make the adjustment:

1. Open the Microchip Harmony Configurator.
2. Load the configuration for the adcp_cal project.
3. Navigate to *Harmony Framework Configuration* > *Drivers* > *Timer* > *Use Timer Driver?* and select *TMR Driver Instance 0*.
4. Change the Timer Period value according to this formula:
   - `Timer Period = 721 * APP_NUM_ANX_PINS` (where, `APP_NUM_ANX_PINS` is defined in `adcp_config.h` and represents the number of pins to sample)
5. Generate and run the demonstration again

BMX Peripheral Library Examples

This topic provides descriptions of the BMX Peripheral Library examples.

Introduction

BMX Peripheral Library Demonstration Applications Help

Description

This distribution package contains one BMX related firmware project that demonstrates the capabilities of the MPLAB Harmony BMX Peripheral Library.

This section describes the hardware requirement and procedures to run this firmware project on Microchip demonstration and development boards.

To learn more about MPLAB Harmony stacks and libraries refer to the related documentation in Volume V: MPLAB Harmony Framework Reference.

Demonstrations

This topic provides information on how to run the BMX Peripheral Library demonstration applications included in this release.

mem_partition

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This example sets up a memory partition in the PIC32MX device for Kernel and User segments. If the memory is set correctly, the LEDs on the Explorer 16 Development Board are lit.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the BMX Peripheral Library BMX Handler demonstration.

Description

To build this project, you must open the `mem_partition.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/examples/peripheral/bmx/mem_partition`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`mem_partition.X`	`<install-dir>/apps/examples/peripheral/bmx/mem_partition/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx795_pim_e16	pic32mx795_pim+e16	Demonstrates the BMX memory partition demonstration on the PIC32MX795F512L PIM and Explorer 16 Development Board.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MX795F512L CAN-USB PIM and Explorer 16 Development Board

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the BMX demonstration.

Description

This demonstration sets up Kernel and User memory partitions through the BMX Peripheral Library.

1. First compile and program the target device. While compiling, select the configuration suitable for the hardware in use.
2. Observe the status of the LEDs on the Explorer 16 Development Board. If the memory was correctly partitioned, the LEDs on the Explorer 16 Development Board are lit.

CAN Peripheral Library Examples

This topic provides descriptions of the CAN demonstrations.

Introduction

CAN Library Demonstration Applications Help.

Description

This distribution package contains one CAN-related firmware project that demonstrates the capabilities of the MPLAB Harmony CAN Library. This section describes the hardware requirement and procedures to run this firmware project on Microchip demonstration and development boards.

To learn more about MPLAB Harmony stacks and libraries, refer to the related documentation in Volume V: MPLAB Harmony Framework Reference.

Demonstrations

This topic provides information on how to run the CAN Library demonstration applications included in this release.

echo_send

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This demonstration shows how to use the CAN Peripheral Library and CAN peripheral on a device to send out a message over the CAN bus. This demonstration uses message filtering to echo back user data sent to its specific address. This demonstration does not use a CAN Driver or the CAN Stack.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the CAN Echo Send demonstration.

Description

To build this project, you must open the `echo_send.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/examples/peripheral/can/echo_send`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`echo_send.X`	`<install-dir>/apps/examples/peripheral/can/echo_send/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are within `./firmware/src/system_config`.

Project Configuration Name	BSP(s) Used	Description
pic32mx_125_sk	pic32mx_125_sk	Demonstrates sending a message over the CAN bus on the PIC32MX1/2/5 Starter Kit.
pic32mz_ef_sk_io_ebrd_can_pictail	pic32mz_ef_sk	Demonstrates sending a message over the CAN bus using the CAN/LIN PICtail Plus Daughter Board and the PIC32MZ EF Starter Kit connected to the PIC32MZ Starter Kit Adapter Board and Starter Kit I/O Expansion Board.
pic32mx_usb_sk2	pic32mx_usb_sk2	Demonstrates sending a message over the CAN bus using the combined CAN/LIN PICtail Plus Daughter Board and the PIC32 USB Starter Kit II.
pic32mz_ef_sk_io_ebrd_can_pictail_16b	pic32mz_ef_sk	Demonstrates sending a message over the CAN bus with 16-bit configuration using the CAN/LIN PICtail Plus Daughter Board and the PIC32MZ EF Starter Kit connected to the PIC32MZ Starter Kit Adapter Board and the Starter Kit I/O Expansion Board.
pic32mk_gp_db	pic32mk_gp_db	Demonstrates sending a message over the CAN bus using the CAN socket available on the PIC32MK GP Development Board.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MX1/2/5 Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32 USB Starter Kit II

No hardware related configuration or jumper setting changes are necessary.

Connect DB9 on CAN1 socket using CAN/LIN PICtail Plus Daughter Board. Use I/O expansion board as interface between Starter kit and PICtail board.

PIC32MZ EF Starter Kit connected to the PIC32MZ Starter Kit Adapter Board

On the PIC32MZ Starter Kit Adapter Board, short pins 1 and 2 of JP1 and short pins 1 and 2 of JP3.

Connect DB9 on CAN1 socket using CAN/LIN PICtail Plus Daughter Board. Use the I/O expansion board as the interface between the starter kit and the PICtail daughter board.

CAN/LIN PICtail Plus Daughter Board

For CAN1, set J4 to 1 and J2 to 2-3.

PIC32MK GP Development Board

No hardware related configuration or jumper setting changes are necessary.

Connect DB9 on the CAN1 socket.

Running the Demonstration

Provides instructions on how to build and run the CAN Echo Send demonstration.

Description

The following tools are required to run this demonstration:

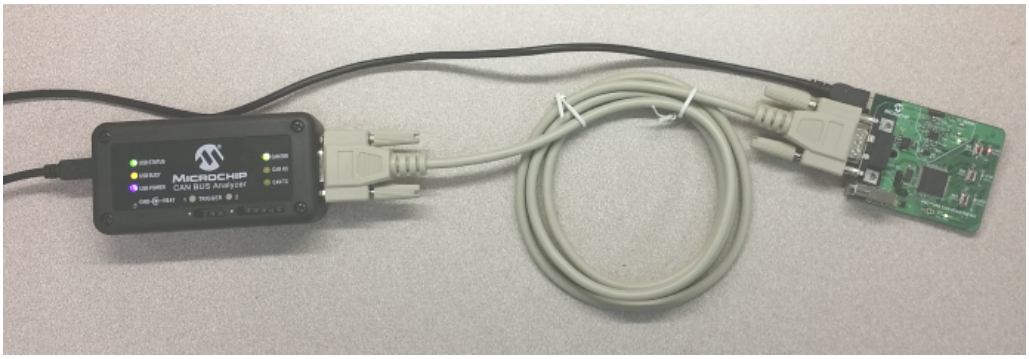
- MPLAB X IDE
- MPLAB XC32 C/C++ Compiler

Note: Prior to using this demonstration, it is recommended to review the MPLAB Harmony Release Notes for the specific versions. A PDF copy of the release notes is provided in the `<install-dir>/doc` folder of your installation.

- One of the following:
  - PIC32MX1/2/5 Starter Kit
  - CAN/LIN PICtail Plus Daughter Board and PIC32 USB Starter Kit II combination
  - PIC32MZ EF Starter Kit and PIC32MZ Starter Kit Adapter Board combination
  - PIC32MK GP Development Board
- One Microchip CAN BUS Analyzer (APGDT002) or equivalent
- One DB9 cable
- Two USB A-to-B mini cables
- A personal computer

Hardware Example

The following figure illustrates the hardware setup for the PIC32MX1/2/5 Starter Kit.



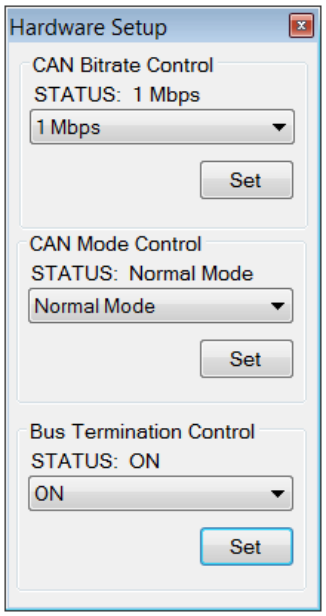
Refer to *"CAN BUS Analyzer User's Guide"* (DS50001848) for further operating instructions regarding the CAN BUS Analyzer.

For more information on the CAN Bus, visit: http://www.can-cia.org/

Do the following to run the demonstration:

1. Connect one end of the DB9 cable to the starter kit and the other end to the CAN BUS Analyzer.
2. Connect the USB cables to J3 on the starter kit and to the CAN BUS Analyzer.

3. Open the CAN BUS Analyzer software.
4. Configure the Analyzer for the Baud Rate Selected inside MHC (1 Mbps Typical), Normal Operation, Termination Resistor: ON.



5. Start MPLAB X IDE and open the project, `echo_send.X`.
6. Program the starter kit.
7. Open the Rolling Trace window in the CAN BUS Analyzer software.
8. LED1 must be ON by default. Press switch 1 to see the output in the trace window. Note: While switch is pressed LED should turn OFF.

Note: When the switch is pressed, the LED should turn OFF.

9. Press the button(s) on the starter kit to see the output in the trace window.
10. Open the Transmit window.
11. To get a response from the starter kit, the following address must be used: ID = 0x201. Following the address, enter the DLC, which is the Data Length Code (the number of bytes to follow). For example, if 0x201, DLC, Byte1, Byte2, Byte3, Byte4 was entered, the starter kit will respond with: 0x102, DLC, Byte1, Byte2, Byte3, Byte4.

Demonstration Output

The following figure shows the Rolling Trace send/receive output from the CAN Bus Analyzer.

Rolling Trace

TRACE	ID	DLC	DATA 0	DATA 1	DATA 2	DATA 3	DATA 4	DATA 5	DATA 6	DATA 7	TIME STAMP (sec)	TIME DELTA (sec)
RX	0x102	4	0x01	0x02	0x03	0x04					461.4672	459.717
TX	0x201	4	0x01	0x02	0x03	0x04					1.7501	3848.645

Button Press Mapping

SW1 Press **-** 0x12C, 2, 0x11, 0x222.

The following figure shows the Rolling Trace output message.

Rolling Trace

RX	0x102	4	0x01	0x02	0x03	0x04				461.4672	459.717
TX	0x201	4	0x01	0x02	0x03	0x04				1.7501	3848.645

can_display

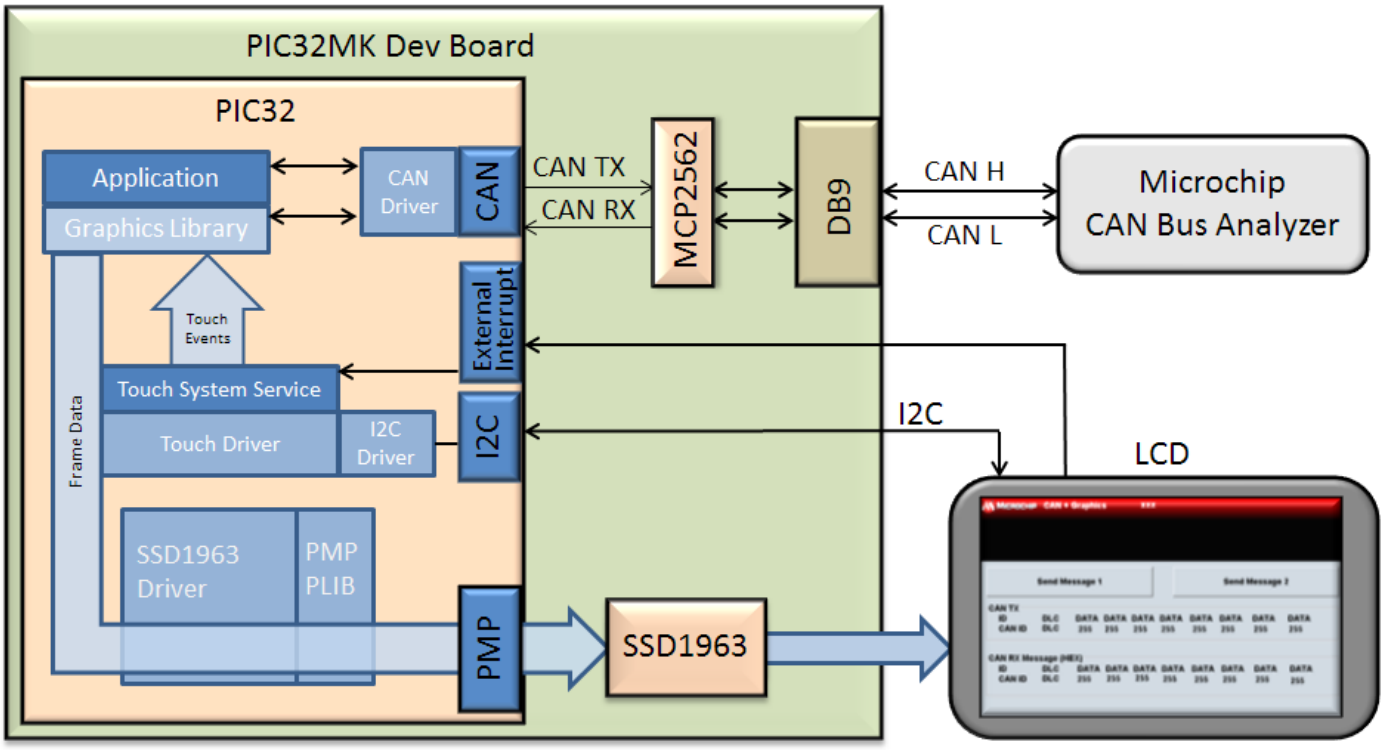
This topic provides information on how to run the can_display demonstration application included in this release.

Description

The purpose of this demonstration is to show how to use the CAN Peripheral Library and Graphics Library together. This demonstration should show you all incoming messages it sees on the bus. In addition to reading all incoming messages, it is capable of transmitting messages using the two on-screen buttons.

Note: The messages transmitted have no meaning other than a placeholder and to show that messages can be sent and is random made up data that is within the allowable ranges (0x0-0xFF).

Architecture



SSD1963

The SSD1963 is and on-board LCD controller. This dedicated device contains the logic that generates the LCD timing, as well as the frame buffers used for storing graphics.

MCP2562

The MCP2562 is a CAN transceiver, it takes CAN Receive (RX) and CAN Transmit (TX) from the microcontroller and converts the signals into a split differential pair for communicating on the bus.

DB9 connector

This is connector interface that is a standard CAN to DP9 pin-out interface.

CAN Bus Analyzer

This is a tool that allows the user to view, send, and receive messages on the CAN bus.

LCD Display

There are currently two display options available for this demonstration:

- 4.3” WQVGA TFT display that is 480x272 and has a maXTouch touch controller
- 5.0” WVGA TFT display that is 800x480 and has a maXTouch touch controller

Demonstration Features

- CAN Driver
- I2C Driver
- PMP Driver
- SSD1963 Driver
- CAN Peripheral Library
- Aria User Interface Library

Tools Setup Differences

- The following was changed in MHC: operation mode. This was set to “CAN_LISTEN_ALL_MESSAGE_MODE” because we want to see all traffic on the CAN bus and display it on the LCD under the CAN RX Messages (HEX) group box.
- Setting the baud rate for the CAN module can be very complicated, a calculator in included in MHC to make this easier. Enter the desired baud rate and click **Execute**

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the CAN Display demonstration.

Description

To build this project, you must open the `can_display.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/examples/peripheral/can/can_display`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`can_display.X`	`<install-dir>/apps/examples/peripheral/can/can_display/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are within `./firmware/src/system_config`.

Project Configuration Name	BSP(s) Used	Description
pic32mk_gp_db_wqvga_mxt	pic32mk_gp_db_wqvga_mxt	Demonstrates the use of the CAN Peripheral Library and graphics library in the form of an application. GUI information is displayed on a 4.3” touch screen.
pic32mk_gp_db_wvga_mxt	pic32mk_gp_db_wvga_mxt	Demonstrates the use of the CAN Peripheral Library and graphics library in the form of an application. GUI information is displayed on a 5.0” touch screen

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MK GP Development Board

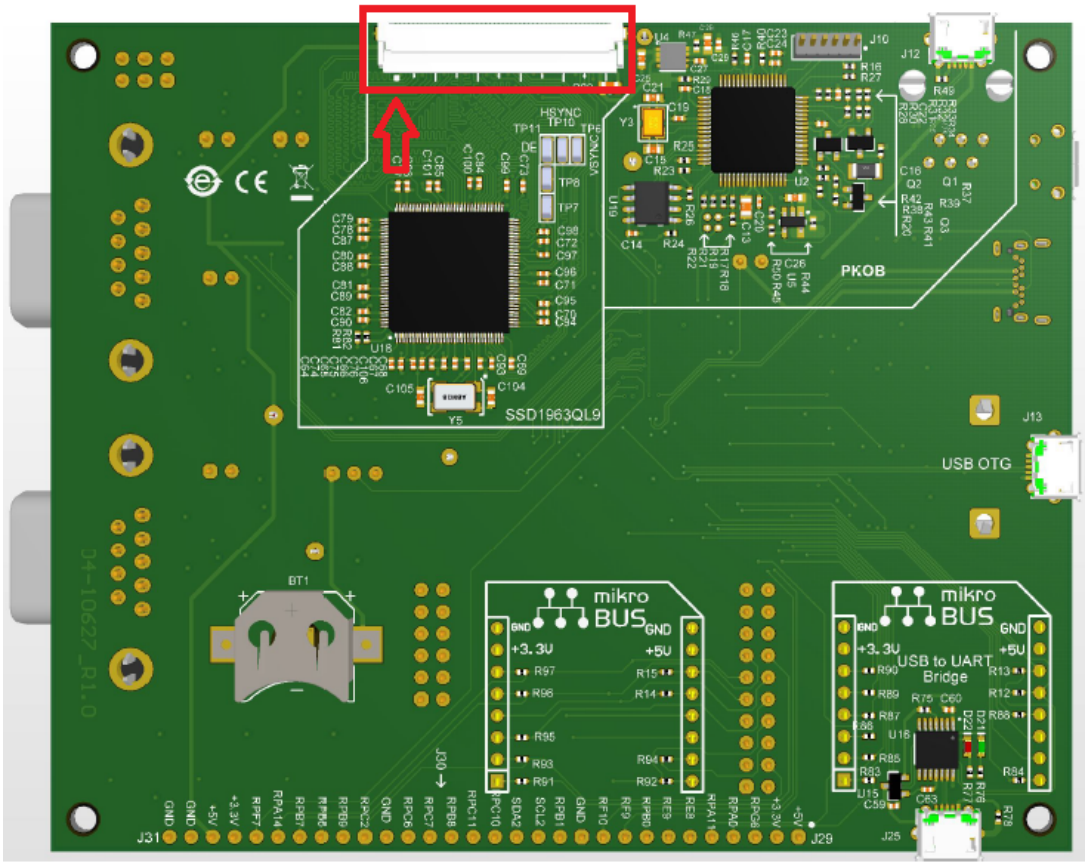
Configurations: pic32mk_gp_db_wqvga and pic32mk_gp_db_wvga

- pic32mk_gp_db_wqvga_mxt requires the 4.3” High-Performance WQVGA Display with maXTouch (Microchip P/N: AC320005-4)
- pic32mk_gp_db_wvga_mxt requires the 5.0” High-Performance WQVGA Display with maXTouch (Microchip P/N: AC320005-5)

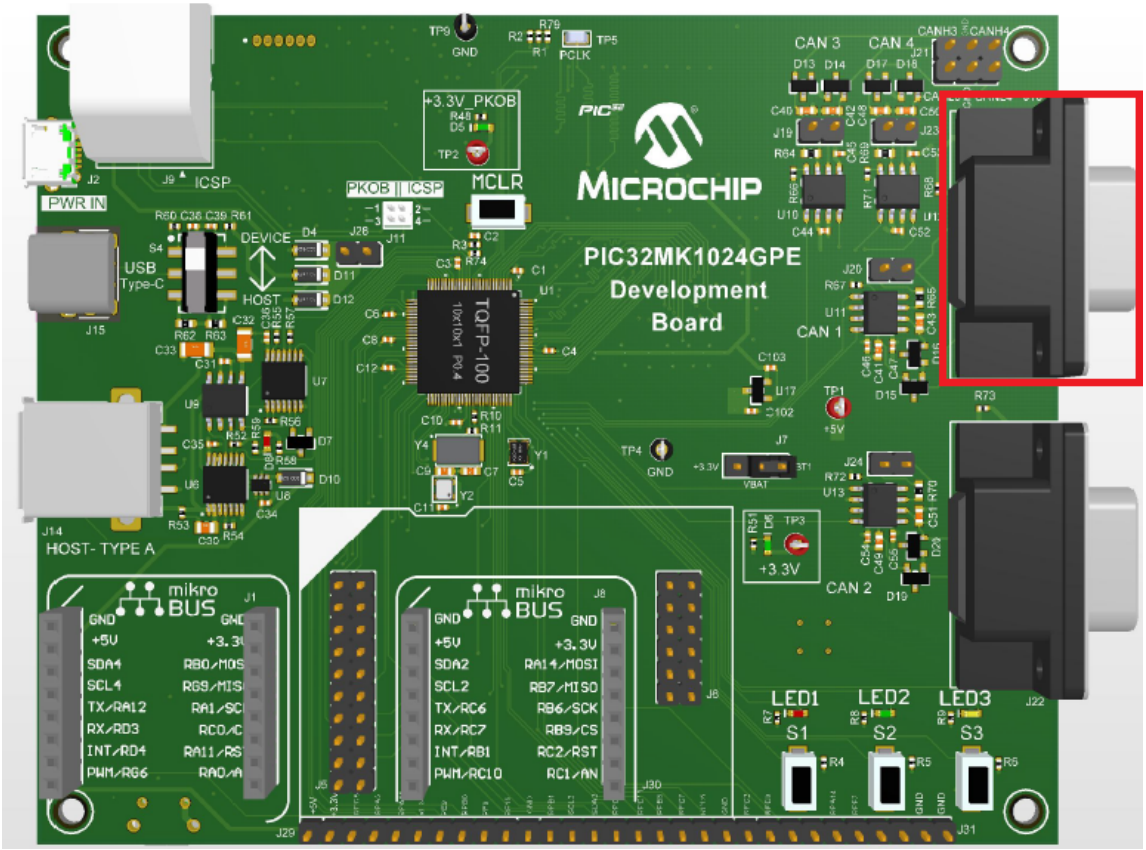
No hardware related configuration or jumper setting changes are necessary.

Connect the hardware as follows:

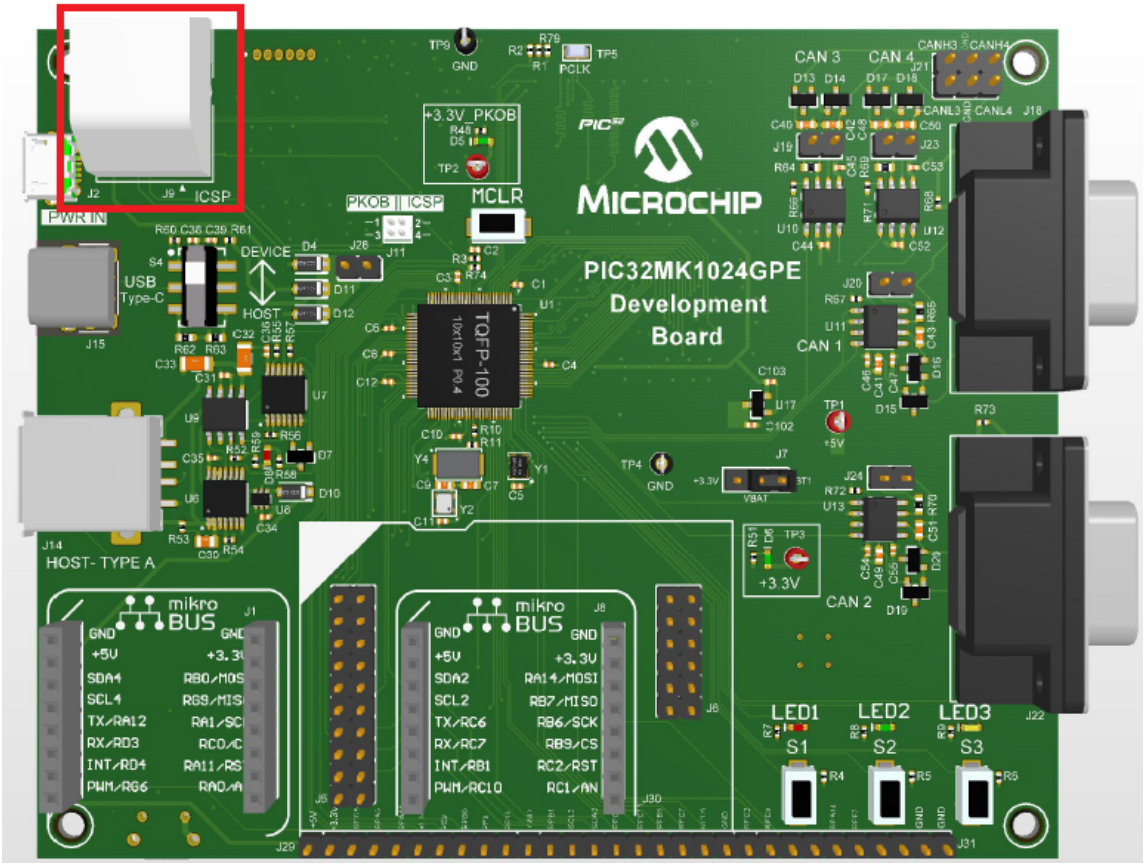
1. Connect the desired display to the 50-pin flex cable located at the bottom of the board. The arrow points to PIN 1.



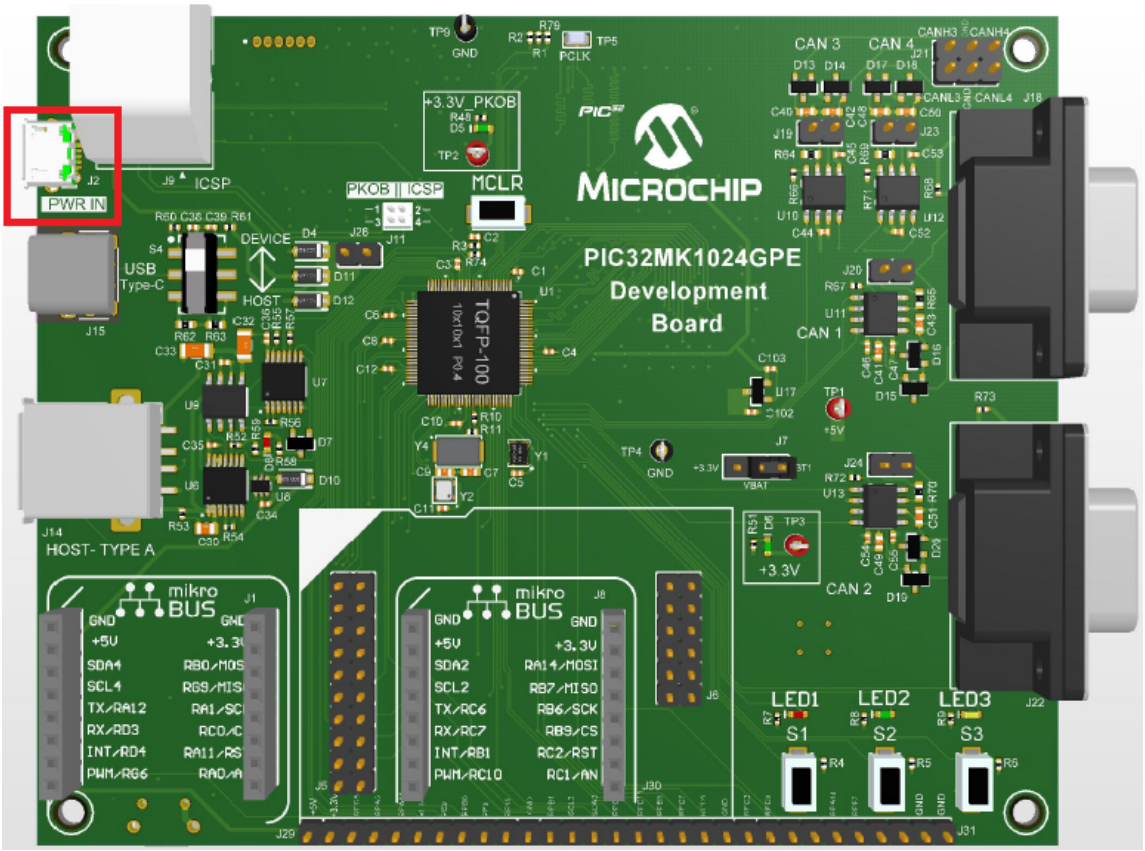
2. Connect the CAN bus analyzer to J18 located on the top of the board.



3. Connect a MPLAB REAL ICE In-Circuit Emulator or MPLAB ICD3 In-Circuit Emulator to J9 (RJ-11 socket).



4. Connect power by using a USB A to B micro into J2.



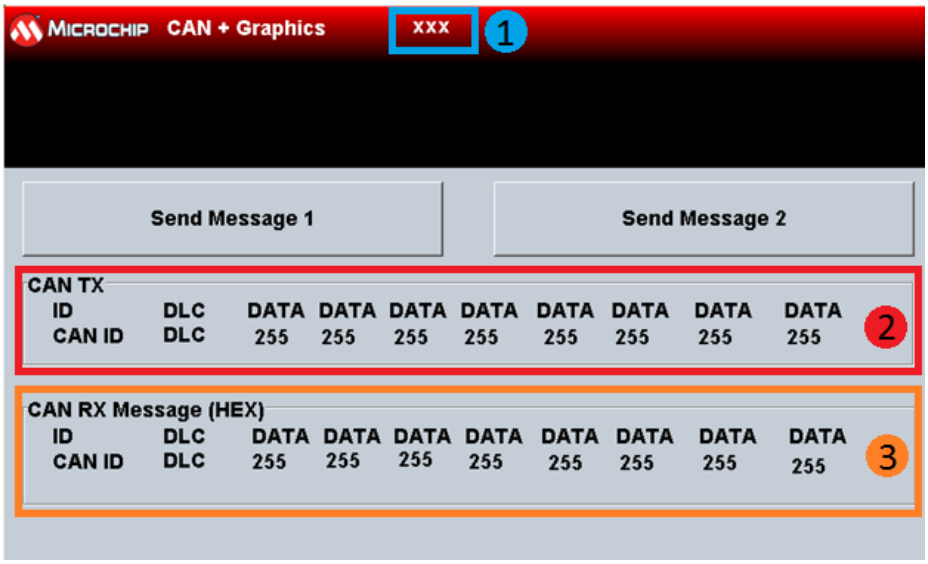
Running the Demonstration

Provides instructions on how to build and run the CAN Display demonstration.

Description

This demonstration exercises the MPLAB Harmony CAN Peripheral Library and Graphics Library.

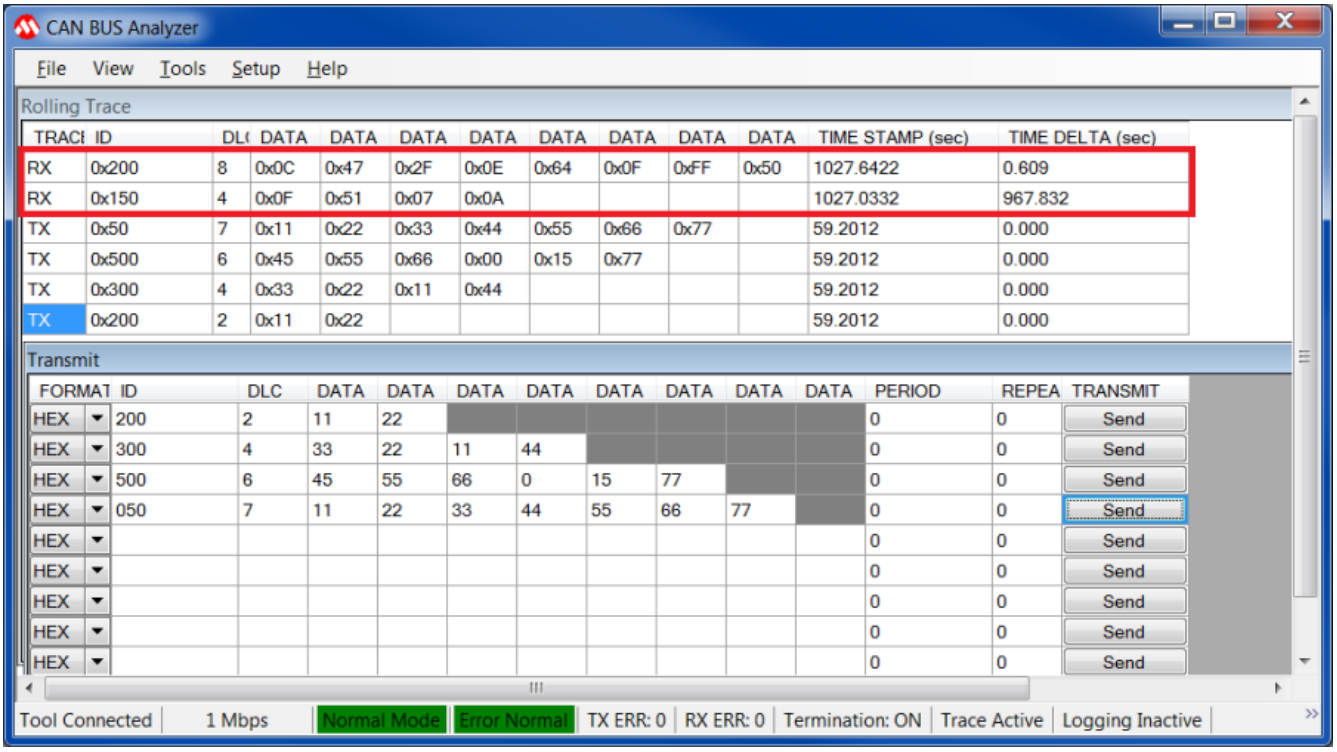
1. Start MPLAB X IDE and open the project, `can_display.X`.
2. Program the starter kit.
3. You should see a screen very similar to that of the following figure. Due to the difference in screen sizes (WVGA versus WQVGA) the layout had to be changed slightly to fit everything on the screen.



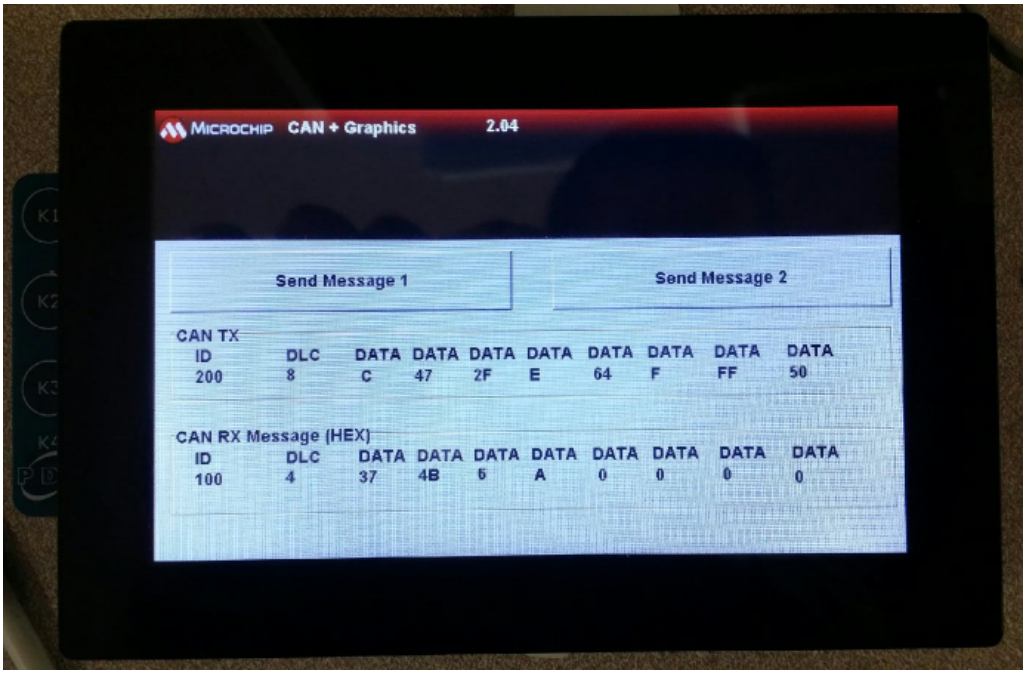
- 1) Shows the Harmony Version number
- 2) Shows all data being transmitted from the demonstration board
- 3) Shows all the data being transmitted on the bus

5. Open your CAN tool of choice and set it to 1 Mbps, press either message button to the transmitting message (below in red). From your CAN tool, send a message to see it on the screen under “CAN RX Messages (HEX)”.

The following figure shows the Microchip CAN Bus Analyzer tool interface (APGDT002). This tool can send and receive messages. The values shown above are just examples of data of what could be sent. For more information on using this tool refer to the *CAN Bus Analyzer User's Guide* (DS51848), which is available for download from the Microchip website (www.microchip.com).



The next figure shows the demonstration running in real-time. The values under “CAN TX” correspond to “Send Message 2” and can also been seen in the red box in the Microchip CAN Bus Analyzer tool.



	ID	DLC	DATA 1	DATA 2	DATA 3	DATA 4	DATA 5	DATA 6	DATA 7	DATA 8
Send Message 1	150	4	F	51	7	A	-	-	-	-
Send Message 2	200	8	C	47	2F	E	64	F	FF	50

Note: All values for this are in HEX, 0x is not show as the leading values. The values being transmitted is just example data.

Comparator Peripheral Library Examples

This topic provides descriptions of the Comparator Peripheral Library examples.

Introduction

Comparator Peripheral Library Demonstration Applications Help

Description

This distribution package contains one Comparator related firmware project that demonstrates the capabilities of the MPLAB Harmony Comparator Peripheral Library.

This section describes the hardware requirement and procedures to run this firmware project on Microchip demonstration and development boards.

To learn more about MPLAB Harmony stacks and libraries refer to the related documentation in Volume V: MPLAB Harmony Framework Reference.

Demonstrations

This topic provides information on how to run the Comparator Peripheral Library demonstration applications included in this release.

simple_comparator

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This example sets up the Comparator in the PIC32MX device to compare the potentiometer input on inverting input to the reference voltage on non-inverting input. Depending on the comparison, one half of the LEDs on the Explorer 16 Development Board will be lit.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Comparator Peripheral Library demonstration.

Description

To build this project, you must open the `simple_comparator.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/examples/peripheral/cmp/simple_comparator`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`simple_comparator.X`	`<install-dir>/apps/examples/peripheral/cmp/simple_comparator/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx795_pim_e16	pic32mx795_pim+e16	Simple comparator demonstration on the PIC32MX795F512L PIM and Explorer 16 Development Board.
pic32mx795_pim_e16_freertos	pic32mx795_pim+e16	FreeRTOS version of the simple comparator demonstration on the PIC32MX795F512L PIM and Explorer 16 Development Board.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MX795F512L CAN-USB PIM and Explorer 16 Development Board

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the Comparator demonstration.

Description

This demonstration sets up the analog comparator to trigger an interrupt when the inverting and non-inverting inputs change relative value.

1. First compile and program the target device. While compiling, select the configuration suitable for the hardware in use.
2. Observe the status of the LEDs on the Explorer 16 Development Board. Depending on the comparison between the inverting and non-inverting inputs, one half of the LEDs (D3-D6 or D7-D10) will be lit.

CVREF Peripheral Library Examples

This topic provides descriptions of the CVREF Peripheral Library examples.

Introduction

CVREF Peripheral Library Demonstration Applications Help.

Description

This distribution package contains one CVREF related firmware project that demonstrates the capabilities of the MPLAB Harmony CVREF Peripheral Library.

This section describes the hardware requirement and procedures to run this firmware project on Microchip demonstration and development boards.

To learn more about MPLAB Harmony stacks and libraries refer to the related documentation in Volume V: MPLAB Harmony Framework Reference.

Demonstrations

This topic provides information on how to run the CVREF Peripheral Library demonstration applications included in this release.

triangle_wave

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This example sets up the CVREF voltage divider in the PIC32 devices to create a triangle waveform that can be observed on an oscilloscope.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the CVREF Peripheral Library demonstration.

Description

To build this project, you must open the `triangle_wave.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/examples/peripheral/cvref/triangle_wave`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`triangle_wave.X`	`<install-dir>/apps/examples/peripheral/cvref/triangle_wave/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Configuration Name	BSP Used	Description
pic32mx795_pim_e16	pic32mx795_pim+e16	CVREF triangle wave demonstration on the PIC32MX795F512L PIM and Explorer 16 Development Board.
pic32mz_ef_pim_e16	pic32mz_ef_pim+e16	CVREF triangle wave demonstration on the PIC32MZ2048EFH100 PIM and Explorer 16 Development Board.
pic32mx795_pim_e16_freertos	pic32mx795_pim+e16	FreeRTOS version of the CVREF triangle wave demonstration on the PIC32MX795F512L PIM and Explorer 16 Development Board.

pic32mz_ef_pim_e16_freertos	pic32mz_ef_pim+e16	FreeRTOS version of the CVREF triangle wave demonstration on the PIC32MZ2048EFH100 PIM and Explorer 16 Development Board.
pic32mz_ef_sk_16b	pic32mz_ef_sk	microMIPS version of the CVREF triangle wave demonstration on the PIC32MZ EF Starter Kit.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MX795F512L CAN-USB PIM and Explorer 16 Development Board

Attach an oscilloscope probe to RB10, which is the CVREFOUT pin.

PIC32MZ2048EFH100 PIM and Explorer 16 Development Board

Attach an oscilloscope probe to RB10, which is the CVREFOUT pin.

PIC32MZ EF Starter Kit, PIC32MZ Starter Kit Adapter Board and Starter Kit I/O Expansion Board

Attach an oscilloscope probe to CVREF pin on J11 in Starter Kit I/O Expansion Board, which is the CVREFOUT pin.

Running the Demonstration

Provides instructions on how to build and run the CVREF demonstration.

Description

This demonstration changes the voltage output on the CVREF pin to create a triangle waveform, which can be observed using an oscilloscope.

1. First compile and program the target device. While compiling, select the configuration suitable for the hardware in use.
2. Observe the output of the CVREFOUT pin using the oscilloscope. A triangle waveform of different voltage levels should be observed.

DDR Peripheral Library Examples

This topic provides descriptions of the DDR Peripheral Library examples.

Introduction

DDR Peripheral Library Demonstration Applications Help

Description

This distribution package contains one DDR related firmware project that demonstrates the capabilities of the MPLAB Harmony DDR Peripheral Library.

This section describes the hardware requirement and procedures to run this firmware project on Microchip demonstration and development boards.

To learn more about MPLAB Harmony stacks and libraries refer to the related documentation in Volume V: MPLAB Harmony Framework Reference.

Demonstrations

This topic provides information on how to run the DDR Peripheral Library demonstration applications included in this release.

Description

This release contains one demonstration:

- write_read_ddr2

write_read_ddr2

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This example demonstration tests the DDR functionality that is included on the PIC32MZ DA Family Starter Kits.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the DDR Peripheral Library demonstration.

Description

To build this project, you must open the `write_read_ddr2.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/examples/peripheral/ddr/write_read_ddr2`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`write_read_ddr2.X`	`<install-dir>/apps/examples/peripheral/ddr/write_read_ddr2/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mz_da_sk_extddr	pic32mz_da_sk_extddr	Demonstrates DDR functionality on the PIC32MZ Embedded Graphics with External DRAM (DA) Starter Kit.
pic32mz_da_sk_intddr	pic32mz_da_sk_intddr	Demonstrates DDR functionality on the PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MZ Embedded Graphics with External DRAM (DA) Starter Kit and PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the DDR demonstration.

Description

This demonstration tests the functionality of the DDR SDRAM included on the PIC32MZ Graphics (DA) Family Starter Kit.

1. First compile and program the target device. While compiling, select the available configuration for the hardware in use.
2. Run the application. LED1 and LED2 (Red and Yellow) will flash as the DDR is being filled. LED2 and LED3 (Yellow and Green) will flash as the DDR is read and the pattern verified. LED3 (Green) will flash if all patterns match those written. LED1 (RED) will flash if any pattern does not match the one written.
3. Pressing Switch 1 at any time will restart the test.

DMA Peripheral Library Examples

This topic provides descriptions of the DMA Peripheral Library examples.

Introduction

DMA Peripheral Library Demonstration Applications Help

Description

This distribution package contains one DMA related firmware project that demonstrates the capabilities of the MPLAB Harmony DMA Peripheral Library.

This section describes the hardware requirement and procedures to run this firmware project on Microchip demonstration and development boards.

To learn more about MPLAB Harmony stacks and libraries refer to the related documentation in Volume V: MPLAB Harmony Framework Reference.

Demonstrations

This topic provides information on how to run the DMA Peripheral Library demonstration applications included in this release.

dma_led_pattern

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This demonstration displays a pattern stored in Flash on LEDs using DMA transfers (by exercising the DMA Peripheral Library), which are triggered by a Timer1 interrupt.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the DMA Peripheral Library DMA Handler demonstration.

Description

To build this project, you must open the `dma_led_pattern.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/examples/peripheral/dma/dma_led_pattern`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`dma_led_pattern.X`	`<install-dir>/apps/examples/peripheral/dma/dma_led_pattern/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
bootloader_pic32mx_eth_sk	pic32mx_eth_sk	This configuration is used and programmed by the Bootloader Demonstrations. Refer to those demonstrations for more information.
bootloader_pic32mz_ef_sk	pic32mz_ef_sk	This configuration is used and programmed by the Bootloader Demonstrations. Refer to those demonstrations for more information.
bootloader_pic32mz_da_sk_intddr	pic32mz_da_sk_intddr	This configuration is used and programmed by the Bootloader Demonstrations. Refer to those demonstrations for more information.
pic32mx795_pim_e16	pic32mx795_pim+e16	Demonstrates the DMA led pattern on PIC32MX795F512L CAN-USB PIM and Explorer 16 Development Board combination.
pic32mz_ef_sk	pic32mz_ef_sk	Demonstrates the DMA LED pattern on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit.
pic32mz_da_sk_intddr	pic32mz_da_sk_intddr	Demonstrates the DMA LED pattern on the PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 Ethernet Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32MX795F512L CAN-USB PIM and Explorer 16 Development Board

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the DMA demonstration.

Description

This demonstration triggers an interrupt based on the DMA alarm.

1. First compile and program the target device. While compiling, select the configuration for the hardware in use.
2. Observe the pattern on the LEDs upon successful execution. The pattern will repeat and continuously blink the LEDs in succession. If there is a failure, the LEDs will stop toggling.

EBI Peripheral Library Examples

This topic provides descriptions of the EBI Peripheral Library examples.

Introduction

External Bus Interface (EBI) Peripheral Library Demonstration Applications Help

Description

This distribution package contains an EBI SRAM read/write demonstration project that demonstrates the capabilities of the EBI and its ability to store data and access data that is attached to it.

This section describes the hardware requirement and procedures to run these firmware projects on Microchip demonstration and development boards.

To learn more about MPLAB Harmony stacks and libraries refer to the related documentation in Volume V: MPLAB Harmony Framework Reference.

Demonstrations

This topic provides information on how to run the EBI Peripheral Library demonstration applications included in this release.

sram_read_write

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Building the Application

for the EBI Peripheral Library SRAM Read/Write Demonstration.

Description

To build this project, you must open the `sram_read_write.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/examples/peripheral/ebi/sram_read_write`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`sram_read_write.X`	`<install-dir>/apps/examples/peripheral/ebi/sram_read_write`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mz_ef_sk_meb2	pic32mz_ef_sk+meb2	Demonstrates SRAM read/write on the PIC32MZ EF Starter Kit with the MEB II.

pic32mz_ef_sk_meb2_16b	pic32mz_ef_sk+meb2	This configuration is the microMIPS version of the demonstration. Demonstrates SRAM read/write on the PIC32MZ EF Starter Kit with the MEB II.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MZ EF Starter Kit and MEB II

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the EBI basic demonstration.

Description

Please use the following procedure to run the demonstration:

1. Load the demonstration project into MPLAB X IDE.
2. Connect the PIC32MZ EC Starter Kit to the MEB II.
3. Connect the USB cable to the mini-B debugger port on the PIC32MZ EC Starter Kit and the other end to the computer.
4. Build, download, and run the demonstration project on the target board.

There are four states to the state machine in this demonstration, APP_STATE_WRITE, APP_STATE_READBACK, APP_STATE_DONE, and APP_STATE_FAIL.

The demonstration will initialize the EBI hardware module. After the hardware had been configured, the demonstration will write in a walking address to the entire memory array. After the demonstration has finished writing in data, it reads back the entire memory array and checks it against the expected data.

If the check PASSES, the demonstration will go to APP_STATE_DONE state where it will remain and indicate the demonstration success by turning ON LED3 (GREEN LED) of the PIC32MZ EC Starter Kit.

If the check FAILS, the demonstration will go to APP_STATE_FAIL state where it will remain and indicate the failure by turning ON LED1 (RED LED) of the PIC32MZ EC Starter Kit.

I2C Peripheral Library Examples

This topic provides descriptions of the I2C Peripheral Library examples.

Introduction

I2C Peripheral Library Demonstration Applications Help

Description

This distribution package contains I2C-related firmware projects that demonstrate the capabilities of the MPLAB Harmony I2C Peripheral Library.

This section describes the hardware requirement and procedures to run these firmware projects on Microchip demonstration and development boards.

To learn more about MPLAB Harmony stacks and libraries refer to the related documentation in Volume V: MPLAB Harmony Framework Reference.

Demonstrations

This topic provides information on how to run the I2C Peripheral Library demonstration applications included in this release.

i2c_interrupt

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the I^2C Peripheral Library example demonstration.

Description

To build this project, you must open the `i2c_interrupt.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/examples/peripheral/i2c/i2c_interrupt`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`i2c_interrupt.X`	`<install-dir>/apps/examples/peripheral/i2c/i2c_interrupt/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx795_pim_e16	pic32mx795_pim+e16	The purpose of this configuration is to demonstrate I^2C Master mode transfer setup in Interrupt mode and static operation. The hardware used is the PIC32MX795F512L PIM connected to the Explorer 16 Development Board.
pic32mx795_pim_e16_freertos	pic32mx795_pim+e16	This configuration is FreeRTOS version of the demo. The purpose of this configuration is to demonstrate I^2C Master mode transfer setup in Interrupt mode and static operation. The hardware used is the PIC32MX795F512L PIM connected to the Explorer 16 Development Board.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

Note: The i2c_interrupt demonstration was tested on the Microchip MCP7949N RTCC device. The address of this device is 0xDE.

Explorer 16 Development Board with the PIC32MX795F512L CAN-USB PIM

- Before attaching the PIC32MX795F512L PIM to the Explorer 16 Development Board, ensure that the processor select switch (S2) is in the PIM Position
- Short JP2 on the Explorer 16 Development Board to enable the LEDs

If a Starter Kit I/O Expansion Board is used, make the following connections:

- Jumper I/O Expansion board J11 pin 36 (SDA2) and J11 pin 38 (SCL2) should be pulled up to 3.3V through a 2.2k ohm resistor

If an external RTCC is used, make the following connections:

- Jumper I/O Expansion board J11 pin 36 (SDA2) and J11 pin 38 (SCL2) to the corresponding lines of an external I2C device
- If an external I2C device is connected, ensure that the Master and Slave device share a common ground.

If a PICtail Plus Daughter Board is used, make the following connections:

- PICtail Plus Daughter Board pins RA2 (SCL2) and RA3 (SDA2) should be pulled up to 3.3V through a 2.2k ohm resistor

If an external RTCC is used, make the following connections:

- Jumper PICtail Plus Daughter Board pins RA2 (SCL2) and pin RA3 (SDA2) to the corresponding SCL and SDA lines of an external I2C device
- If an external I2C device is connected, ensure that the Master and Slave device share a common ground

PIC32MZ EC Starter Kit or PIC32MZ EF Starter Kit connected to MEB II

The jumper JP2 on PIC32MZ EC/EF Starter Kit should be connected according to the debugger/programmer used, as follows:

- If PKOB is used, pins 1 and 3 and pins 2 and 4 should be shorted
- If MPLAB REAL ICE or MPLAB ICD 3 is being used, pins 1 and 3 and pins 2 and 4 should be left open

The connections pertaining to I2C are as follows:

- Connect MEB II J2 pin 3 (SCL2) to the corresponding SCL line of the external I2C device
- Connect MEB II J2 pin 5 (SDA2) to the corresponding SDA line of the external I2C device
- If an external I2C device is connected, ensure that the Master and Slave device share a common ground

Running the Demonstration

Provides instructions on how to build and run the I2C demonstration.

Description

This demonstration shows how to configure and make use of the I2C Driver APIs to support buffered operation of I2C in Interrupt mode. In this demonstration, the I2C is configured as single instance and single client.

Once the demonstration application is compiled successfully for the selected configuration, the firmware can be programmed into the target device.

To run the demonstration in Debug mode, perform the following steps:

1. Select the appropriate configuration from the MPLAB X IDE Project Properties based on the target hardware.
2. Select your device programmer from the Hardware Tool menu in MPLAB X IDE.
3. Select either *Debug* > *Debug Main Project* or click **Debug Main Project** in the toolbar.
4. Build the selected configuration in the MPLAB X IDE project and program the demonstration board.

The I2C Driver configures the I2C2 instance of the I2C peripheral in Master mode. The SDA and SCL lines are connected to the Microchip MCP7940N RTCC device as described in Configuring the Hardware. The Master writes to sequential memory locations in SRAM memory of the RTCC device. The Master then reads back the content from the same page.

The contents of the buffer variable can be checked to determine the result of the operation.

The expected results are shown in the following table.

Test Case	Contents of Buffer
I2C2 (Master)	`RXbuffer_6[] = "3RTCCSLAVE"` (data received from RTCC device)

Input Capture Peripheral Library Examples

This topic provides descriptions of the Input Capture Peripheral Library examples.

Introduction

Input Capture Peripheral Library Demonstration Applications Help

Description

This distribution package contains one Input Capture related firmware project that demonstrates the capabilities of the MPLAB Harmony Input Capture Peripheral Library.

This section describes the hardware requirement and procedures to run this firmware project on Microchip demonstration and development boards.

To learn more about MPLAB Harmony stacks and libraries refer to the related documentation in Volume V: MPLAB Harmony Framework Reference.

Demonstrations

This topic provides information on how to run the Input Capture Peripheral Library demonstration applications included in this release.

ic_basic

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This demonstration exercise the timer capture based on the rising edge of the Input Capture 1 pin (RD8). The timer capture value is stored in 'CaptureTime' variable.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Input Capture Peripheral Library Input Capture Handler demonstration.

Description

To build this project, you must open the `ic_basic.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/examples/peripheral/ic/ic_basic`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`ic_basic.X`	`<install-dir>/apps/examples/peripheral/ic/ic_basic/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx_usb_sk2	pic32mx_usb_sk2	Demonstrates the Input Capture Peripheral Library on the PIC32 USB Starter Kit II and the I/O Expansion Board combination.
pic32mx_usb_sk2_freertos	pic32mx_usb_sk2	Demonstrates the Input Capture Peripheral Library on the PIC32 USB Starter Kit II and the I/O Expansion Board combination.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 USB Starter Kit II and Starter Kit I/O Expansion Board

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the IC demonstration.

Description

This demonstration triggers an interrupt.

1. First compile and program the target device. While compiling, select the configuration suitable for the hardware in use.
2. LED3 will turn ON once a successful input capture has been made.

Flash/NVM Peripheral Library Examples

This topic provides descriptions of the Flash/NVM Peripheral Library examples.

Introduction

NVM Peripheral Library Demonstration Applications Help

Description

This distribution package contains one Flash-related firmware project that demonstrates the capabilities of the MPLAB Harmony NVM Peripheral Library.

This section describes the hardware requirement and procedures to run this firmware project on Microchip demonstration and development boards.

To learn more about MPLAB Harmony stacks and libraries refer to the related documentation in Volume V: MPLAB Harmony Framework Reference.

Demonstrations

This topic provides information on how to run the NVM Peripheral Library demonstration applications included in this release.

flash_modify

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This demonstration does the following:

- Erases a page in the program Flash
- Writes a row of data into the program Flash
- Reads and verifies the written data
- Indicates the success or failure of the operation through on-board LEDs

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the NVM Peripheral Library demonstration.

Description

To build this project, you must open the `flash_modify.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/examples/peripheral/flash/flash_modify`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`flash_modify.X`	`<install-dir>/apps/examples/peripheral/flash/flash_modify/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx795_pim_e16	pic32mx795_pim+e16	Demonstrates the NVM Peripheral Library on the PIC32MX795F512L PIM and Explorer 16 Development Board combination.
pic32mz_ec_sk	pic32mz_ec_sk	Demonstrates the NVM Peripheral Library on the PIC32MZ Embedded Connectivity (EC) Starter Kit.
pic32mz_ef_sk	pic32mz_ef_sk	Demonstrates the NVM Peripheral Library on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit.
pic32mz_ef_sk_16b	pic32mz_ef_sk	Demonstrates the NVM Peripheral Library with 16-bit configuration on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit.
pic32mz795_pim_e16_freertos	pic32mx795_pim+e16	Demonstrates the NVM Peripheral Library on the PIC32MX795F512L PIM and Explorer 16 Development Board combination with FreeRTOS.
pic32mz_ef_sk_freertos	pic32mz_ef_sk	Demonstrates the NVM Peripheral Library on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit with FreeRTOS.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MX795F512L CAN-USB PIM and Explorer 16 Development Board

- Switch S2 should be set to PIM
- Jumper J2 should be in place

PIC32MZ EC Starter Kit

Remove Jumper JP1.

PIC32MZ EF Starter Kit

Remove Jumper JP1.

Running the Demonstration

Provides instructions on how to build and run the NVM demonstration.

Description

This demonstration exercises internal Flash erase, write, and read operations through the NVM Peripheral Library.

1. First compile and program the target device. While compiling, select the configuration suitable for the hardware in use.
2. Upon successful execution of the demonstration, the LEDs on the respective hardware turn ON to indicate success or failure. Refer to the following table for the LED status for the different configurations.

Project Configuration	Success State	Failure State
pic32mx795_pim_e16	LED4 ON	LED3 ON
pic32mz_ec_sk pic32mz_ef_sk pic32mz_da_sk	LED3 ON	LED1 ON

Output Compare Peripheral Library Examples

This topic provides descriptions of the Output Compare Peripheral Library examples.

Introduction

Output Compare Peripheral Library Demonstration Applications Help.

Description

This distribution package contains one Output Compare related firmware project that demonstrates the capabilities of the MPLAB Harmony Output Compare Peripheral Library.

This section describes the hardware requirement and procedures to run this firmware project on Microchip demonstration and development boards.

To learn more about MPLAB Harmony stacks and libraries refer to the related documentation in Volume V: MPLAB Harmony Framework Reference.

Demonstrations

This topic provides information on how to run the Output Compare Peripheral Library demonstration applications included in this release.

oc_pwm

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This demonstration generates a 40 kHz PWM with a 25% duty cycle on the Output Compare 1 output pin.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Output Compare Peripheral Library Output Compare Handler demonstration.

Description

To build this project, you must open the `oc_pwm.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/examples/peripheral/oc/oc_pwm`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`oc_pwm.X`	`<install-dir>/apps/examples/peripheral/oc/oc_pwm`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx_usb_sk2	pic32mx_usb_sk2	Demonstrates the Output Compare PWM on PIC32 USB Starter Kit II.
pic32mz_ef_sk	pic32mz_ef_sk+meb2	Demonstrates the Output Compare PWM on the PIC32MZ EF Starter Kit and MEB II combination.
pic32mx_usb_sk2_freertos	pic32mx_usb_sk2	Demonstrates the Output Compare PWM on PIC32 USB Starter Kit II.
pic32mz_ef_sk_freertos	pic32mz_ef_sk+meb2	Demonstrates the Output Compare PWM on the PIC32MZ EF Starter Kit and MEB II combination.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 USB Starter Kit II and Starter Kit I/O Expansion Board

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ EC Starter Kit and MEB II

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ EF Starter Kit and MEB II

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the OC demonstration.

Description

This demonstration exhibits the generated PWM on the Output Compare pin.

1. First compile and program the target device. While compiling, select the configuration suitable for the hardware in use.
2. The user should see a 40 kHz signal with 25% duty cycle on the oscilloscope on the following pins for the respective configurations:
   - pic32mx_usb_sk2: Output Compare 1 output pin on the I/O Expansion Board
   - pic32mz_ec_sk or pic32mz_ef_sk: Output Compare 5 output pin (pin 22) on the PICtail connector of the MEB II

Oscillator Peripheral Library Examples

This topic provides descriptions of the Oscillator Peripheral Library examples.

Introduction

Oscillator Peripheral Library Demonstration Applications Help

Description

This distribution package contains one Oscillator related firmware project that demonstrates the capabilities of the MPLAB Harmony Oscillator Peripheral Library.

This section describes the hardware requirement and procedures to run this firmware project on Microchip demonstration and development boards.

To learn more about MPLAB Harmony stacks and libraries refer to the related documentation in Volume V: MPLAB Harmony Framework Reference.

Demonstrations

This topic provides information on how to run the Oscillator Peripheral Library demonstration applications included in this release.

osc_config

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This example demonstrates how to change the system clock source and PLL values during run-time.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Oscillator Peripheral Library.

Description

To build this project, you must open the `osc_config.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/examples/peripheral/osc/osc_config`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
osc_config.X	<install-dir>/apps/examples/peripheral/osc/osc_config/firmware

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within ./firmware/src/system_config.

Project Configuration Name	BSP Used	Description
pic32mx_usb_sk2	pic32mx_usb_sk2	Demonstrates the Timer3 interrupt on the PIC32 USB Starter Kit II.
pic32mz_ef_sk	pic32mz_ef_sk	Demonstrates the Timer3 interrupt on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit.
pic32mz_ef_sk_meb2_16b	pic32mz_ef_sk+meb2	This configuration is the microMIPS version of the demonstration. Demonstrates the Timer3 interrupt on the PIC32MZ Embedded Connectivity with the Floating Point Unit (EF) Starter Kit.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32 USB Starter Kit II

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the Oscillator demonstration.

Description

This demonstration configures the Oscillator for different setups at run-time.

1. First compile and program the target device. While compiling, select the appropriate configuration for the hardware in use.
2. LED3 will turn ON if the clock settings have changed during run-time.

PMP Peripheral Library Examples

This topic provides descriptions of the PMP Peripheral Library examples.

Introduction

PMP Peripheral Library Demonstration Applications Help

Description

This distribution package contains one PMP related firmware project that demonstrates the capabilities of the MPLAB Harmony PMP Peripheral Library.

This section describes the hardware requirement and procedures to run this firmware project on Microchip demonstration and development boards.

To learn more about MPLAB Harmony stacks and libraries refer to the related documentation in Volume V: MPLAB Harmony Framework Reference.

Demonstrations

This topic provides information on how to run the PMP Peripheral Library demonstration applications included in this release.

pmp_lcd

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This example initializes the LCD on the Explorer 16 Development Board and a sends string of information to it using the PMP Peripheral Library.

Building the Application

for the PMP Peripheral Library PMP Alarm Demonstration.

Description

To build this project, you must open the `pmp_lcd.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/examples/peripheral/pmp/pmp_lcd`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`pmp_lcd.X`	`<install-dir>/apps/examples/peripheral/pmp/pmp_lcd/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx795_pim_e16	pic32mx795_pim+e16	Demonstrates the PMP LCD configuration on the Explorer 16 Development Board with the PIC32MX795F512L PIM.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MX795F512L CAN-USB PIM and Explorer 16 Development Board

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the PMP demonstration.

Description

This demonstration shows the PMP/counter capabilities using PMP peripheral library functions.

1. First compile and program the target device. While compiling, select the configuration suitable for the hardware in use.
2. Check the LCD of the Explorer 16 Development Board. The message, "Hello! Testing...", should appear on the display at run-time.

Ports Peripheral Library Examples

This topic provides descriptions of the PORTS Peripheral Library examples.

Introduction

Ports Peripheral Library Demonstration Applications Help.

Description

This distribution package contains two Ports related firmware projects that demonstrate the capabilities of the MPLAB Harmony Ports Peripheral Library.

This section describes the hardware requirement and procedures to run these firmware projects on Microchip demonstration and development boards.

To learn more about MPLAB Harmony stacks and libraries refer to the related documentation in Volume V: MPLAB Harmony Framework Reference.

Demonstrations

This topic provides information on how to run the Ports Peripheral Library demonstration applications included in this release.

blinky_leds

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This example blinks and LED with the selected frequency.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Ports Peripheral Library.

Description

To build this project, you must open the `blinky_leds.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/examples/peripheral/ports/blinky_leds`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`blinky_leds.X`	`<install-dir>/apps/examples/peripheral/ports/blinky_leds/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx_eth_sk	pic32mx_eth_sk	Demonstrates the Ports Peripheral Library on the PIC32 Ethernet Starter Kit.
pic32mx_xlp_sk	pic32mx_xlp_sk	Demonstrates the Ports Peripheral Library on the PIC32MX XLP Starter Kit.
pic32mz_ef_sk	pic32mz_ef_sk	Demonstrates the Ports Peripheral Library on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit.
pic32mk_gp_db	pic32mk_gp_db	Demonstrates the Ports Peripheral Library on the PIC32MK General Purpose (GP) Development Board.
pic32mz_ef_sk_16b	pic32mz_ef_sk	Demonstrates the Ports Peripheral Library with 16-bit configuration on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit.
pic32wk_sk	pic32wk_gpb_gpd_sk+module	Demonstrates the Ports Peripheral Library on the PIC32WK Wi-Fi Starter Kit.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 Ethernet Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32MX XLP Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32MK GP Development Board

No hardware related configuration or jumper setting changes are necessary.

PIC32WK Wi-Fi Starter Kit

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the PORTS demonstration.

Description

This demonstration exercises the Ports Peripheral Library functionality.

1. First compile and program the target device. While compiling, select the configuration suitable for the hardware in use.
2. Observe that LED3 (see **Note**) is blinking.

Note: On the PIC32MX XLP Starter Kit, the red LED (LED1) should be blinking.

3. Change the value of APP_LED_BLINK_DELAY in `app.h`, and then compile and run the code. Observe the change in frequency of the blinking of the LED.

cn_interrupt

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This example controls LED toggling using change notice interrupts associated with the peripheral libraries and specific hardware.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Ports Peripheral Library.

Description

To build this project, you must open the `cn_interrupt.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/examples/peripheral/ports/cn_interrupt`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`cn_interrupt.X`	`<install-dir>/apps/examples/peripheral/ports/cn_interrupt/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx_eth_sk	pic32mx_eth_sk	Demonstrates the Ports Peripheral Library on the PIC32 Ethernet Starter Kit.
pic32mx_eth_sk_freertos	pic32mx_eth_sk	Demonstrates the Ports Peripheral Library on the PIC32 Ethernet Starter Kit with FreeRTOS enabled.
pic32mz_ef_sk	pic32mz_ef_sk	Demonstrates the Ports Peripheral Library on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit.
pic32mz_ef_sk_freertos	pic32mz_ef_sk	Demonstrates the Ports Peripheral Library on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit with FreeRTOS enabled.
pic32mz_ef_sk_16b	pic32mz_ef_sk	Demonstrates the Ports Peripheral Library with 16-bit configuration on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit.
pic32wk_sk	pic32wk_gpb_gpd_sk+module	Demonstrates the Ports Peripheral Library on the PIC32WK WiFi Starter Kit.
pic32mz_da_sk	pic32mz_da_sk_extddr	Demonstrates the Ports Peripheral Library on the PIC32MZ DA Starter Kit.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32 Ethernet Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32WK Wi-Fi Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ DA Starter Kit

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides information on how to build and run the Change Notice Interrupt demonstration.

Description

This demonstration exercises the control of on board LED toggling on the basis of change in the switch state using Port Peripheral Library.

1. First compile and program the target device. While compiling, select the configuration suitable for the hardware in use.
2. Observe the status of the LEDs on the demonstration board. Once the demonstration starts running on the demonstration board, the LED3 will begin toggling. Pressing switch SW1, LED3 from toggling and it will be in an OFF state. On another press of switch SW1, LED3 will begin toggling again.

Power Peripheral Library Examples

This topic provides descriptions of the Power Peripheral Library examples.

Introduction

Power Peripheral Library Demonstration Applications Help.

Description

This distribution package contains two Power related firmware project that demonstrates the capabilities of the MPLAB Harmony Power Peripheral Library.

This section describes the hardware requirement and procedures to run this firmware project on Microchip demonstration and development boards.

To learn more about MPLAB Harmony stacks and libraries refer to the related documentation in Volume V: MPLAB Harmony Framework Reference.

Demonstrations

This topic provides information on how to run the Power Peripheral Library demonstration applications included in this release.

sleep_mode

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This demonstration uses the Watchdog Timer to wake the device from Sleep mode and then toggle LEDs based on the status.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Power Peripheral Library Sleep Mode demonstration.

Description

To build this project, you must open the `sleep_mode.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/examples/peripheral/power/sleep_mode`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
sleep_mode.X	<install-dir>/apps/examples/peripheral/power/sleep_mode/firmware

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within ./firmware/src/system_config.

Project Configuration Name	BSP Used	Description
pic32mx_eth_sk	pic32mx_eth_sk	Demonstrates Sleep mode on the PIC32 Ethernet Starter Kit.
pic32mz_ef_sk	pic32mz_ef_sk	Demonstrates Sleep mode on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit.
pic32mz_ef_sk_16b	pic32mz_ef_sk	Demonstrates Sleep mode with 16-bit configuration on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32 Ethernet Starter Kit

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the Power demonstration.

Description

This demonstration puts the device in Sleep mode and than wakes it using the Watchdog Timer.

1. First compile and program the target device. While compiling, select the configuration suitable for the hardware in use.
2. Check the state of the LEDs. LED1 should glow for a few seconds and then turn off and then LED3 will begin toggling.

Reset Peripheral Library Examples

This topic provides descriptions of the Reset Peripheral Library examples.

Introduction

Reset Peripheral Library Demonstration Applications Help

Description

This distribution package contains one Reset related firmware project that demonstrates the capabilities of the MPLAB Harmony Reset Peripheral Library.

This section describes the hardware requirement and procedures to run this firmware project on Microchip demonstration and development boards.

To learn more about MPLAB Harmony stacks and libraries refer to the related documentation in Volume V: MPLAB Harmony Framework Reference.

Demonstrations

This topic provides information on how to run the Reset Peripheral Library demonstration applications included in this release.

reset_handler

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This example performs checks on various reset flags, assigning each one to an I/O port pin. If the flag is set, the corresponding LED is turned ON. All of the flags can then be cleared by pressing a switch on the board.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Reset Peripheral Library Reset Handler demonstration.

Description

To build this project, you must open the `reset_handler.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/examples/peripheral/reset/reset_handler`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`reset_handler.X`	`<install-dir>/apps/examples/peripheral/reset/reset_handler/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx795_pim_e16	pic32mx795_pim+e16	Demonstrates the Reset Peripheral Library on the PIC32MX795F512L PIM and the Explorer 16 Development Board.
pic32mz_ef_sk	pic32mz_ef_sk	Demonstrates the Reset Peripheral Library on the PIC32MZ EF Starter Kit.
pic32mz_ef_sk_16b	pic32mz_ef_sk	microMIPS version of the demonstration, which demonstrates the Reset Peripheral Library on the PIC32MZ EF Starter Kit.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MX795F512L CAN-USB PIM and Explorer 16 Development Board

Jumper JP2 should be connected (shorted) for the LEDs to function.

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the RESET demonstration.

Description

This demonstration finds the reason for a Reset and illuminates the LEDs accordingly.

First compile and program the target device. While compiling, select the configuration suitable for the hardware in use.

PIC32MX795F512L PIM with Explorer 16 Development Board:

Check the status of the LEDs to observe the different reasons for a Reset, as follows:

1. Press Switch S6 to clear all Reset flags, which turns all LEDs OFF.
2. Press the MCLR switch on the board to cause a Reset. LED D9 should turn ON.
3. Unplug the power and plug it again to cause a Power-on Reset (POR). LED D5 should turn ON.

4. Press the MCLR switch again. LED D5 and D9 should both turn ON.
5. Press Switch S6 again to clear all Resets, which turns all LEDs OFF.
6. Press Switch S4 to enable the Watchdog Timer. After a few seconds, LED 10 should turn ON indicating a WDT reset has occurred.

These steps can be repeated from the beginning.

PIC32MZ EF Starter Kit:

1. Press Switch S3 to clear all Reset flags, which turns all LEDs OFF.
2. Press Switch S1 to enable the Watchdog Timer. After a few seconds, LED 1 should turn ON indicating a WDT reset has occurred.

Note: A POR clears all the other Reset flags, with the exception of a BOR and a POR flags.

SPI Peripheral Library Examples

This topic provides descriptions of the SPI Peripheral Library examples.

Introduction

SPI Peripheral Library Demonstration Applications Help

Description

This distribution package contains two SPI related firmware project that demonstrates the capabilities of the MPLAB Harmony SPI Peripheral Library.

This section describes the hardware requirement and procedures to run this firmware project on Microchip demonstration and development boards.

To learn more about MPLAB Harmony stacks and libraries refer to the related documentation in Volume V: MPLAB Harmony Framework Reference.

Demonstrations

This topic provides information on how to run the SPI Peripheral Library demonstration applications included in this release.

spi_loopback

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This demonstration loops back data transfer. This example assumes that the SPI SDO (output) is connected to the SDI (input).

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the SPI Peripheral Library Flash Read in PIO Mode Demonstration.

Description

To build this project, you must open the `spi_loopback.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/examples/peripheral/spi/spi_loopback`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`spi_loopback.X`	`<install-dir>/apps/examples/peripheral/spi/spi_loopback/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx_usb_sk2	pic32mx_usb_sk2	Demonstrates the SPI loopback on the PIC32 USB Starter Kit II and the Starter Kit I/O Expansion Board combination.
pic32mx795_pim_e16	pic32mx795_pim+e16	Demonstrates the SPI loopback on the PIC32MX795F512L PIM, Explorer 16 Development Board, and the Starter Kit I/O Expansion Board combination.
pic32mz_ef_sk	pic32mz_ef_sk	Demonstrates the SPI loopback on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit and the Starter Kit I/O Expansion Board combination.
pic32mz_ef_sk_16b	pic32mz_ef_sk	Demonstrates the SPI loopback on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit and the Starter Kit I/O Expansion Board combination in microMIPS mode.
pic32mz_ef_sk_freertos	pic32mz_ef_sk	Demonstrates the SPI loopback on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit and the Starter Kit I/O Expansion Board combination with FreeRTOS.
pic32mx795_pim_e16_freertos	pic32mx795_pim+e16	Demonstrates the SPI loopback on the PIC32MX795F512L PIM, Explorer 16 Development Board, and the Starter Kit I/O Expansion Board combination with FreeRTOS.
pic32wk_sk	pic32wk_gpb_gpd_sk+module	Demonstrates the SPI loopback on the PIC32WK WiFi Starter Kit.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MX795F512L CAN-USB PIM, Explorer 16 Development Board, and Starter Kit I/O Expansion Board

- Connect (using wires) the SDI1 and SDO1 pins on the Starter Kit I/O expansion board
- Connect the Starter Kit I/O Expansion Board to the Explorer 16 Development Board and mount the PIM on the development board

PIC32MZ EF Starter Kit and Starter Kit I/O Expansion Board

- Connect (using wires) the SDI2 pin (J11 pin 32) and SDO2 pin (J10 pin 35) on the Starter Kit I/O Expansion Board
- Connect the PIC32MZ EF Starter Kit with the adapter board to the Starter Kit I/O Expansion Board

PIC32 USB Starter Kit II and Starter Kit I/O Expansion Board

- Connect (using wires) the SDI1 and SDO1 pins on the Starter Kit I/O Expansion Board
- Connect the Starter Kit I/O Expansion Board to the PIC32 USB Starter Kit II

PIC32WK Wi-Fi Starter Kit

- Connect (by using wires) the SDI1 and SDO1 pins on the Starter Kit

Running the Demonstration

Provides instructions on how to build and run the SPI PIO mode transfer demonstration.

Description

This demonstration loops back data on the SPI module (SPI1 and SPI2 when using the PIC32MZ EC Starter Kit or the PIC32MZ EF Starter Kit, and SPI1 and SPI3 when using the PIC32MZ DA Starter Kit.

1. First compile and program the target device. While compiling, select the configuration suitable for the hardware in use.
2. Check the state of LEDs:

- LED D4 and D5 on the Explorer 16 Development Board
- LED3 and LED2 on the PIC32 USB Starter Kit II, PIC32MZ EC Starter Kit, PIC32MZ EF Starter Kit, PIC32MZ DA Starter Kit, or PIC32WK WiFi Starter Kit.

3. If the demonstration was successful, the LED D5 or LED3 illuminates. If the demonstration fails, LED D4 or LED2 illuminates.

SQI Peripheral Library Examples

This topic provides descriptions of the SQI Peripheral Library examples.

Introduction

SQI Peripheral Library Demonstration Applications Help

Description

This distribution package contains two SQI related firmware projects that demonstrate the capabilities of the MPLAB Harmony SQI peripheral libraries. This section describes the hardware requirement and procedures to run these firmware projects on Microchip demonstration and development boards.

To learn more about MPLAB Harmony stacks and libraries refer to the related documentation in Volume V: MPLAB Harmony Framework Reference.

Demonstrations

This topic provides information on how to run the SQI Peripheral Library demonstration applications included in this release.

flash_read_dma_mode

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This demonstration application exercises SQI module in DMA mode, by reading a page in the SQI Flash (SST26VF032/SST26VF032B) device on the PIC32MZ EC Starter Kit or the PIC32MZ EF Starter Kit. This demonstration uses PIO mode to write the Flash. Following steps describe the functionality per the calls during run time:

1. Sets up SQI to run at 25 MHz. (`system_init.c`:: SYS_Initialize, `app.c`:: APP_Initialize)
2. Once the clock frequency is programmed, the demonstration prepares the connected Serial Flash device for a write read transaction (`app.c`:: APP_Tasks:: APP_STATE_INIT_FLASH:: SQI_Flash_Setup).
3. Reads the device ID of the attached Flash device (`app.c`:: APP_Tasks:: APP_STATE_FLASH_ID_READ:: SQI_FlashID_Read).
4. Writes a page in Flash in PIO mode (`app.c`:: APP_Tasks:: APP_STATE_WRITE_FLASH:: SQI_PIO_PageWrite(FLASH_PAGE_ADDR)).
5. Reads the contents of the page in Flash using DMA mode and compares it to the data written to make sure the operation is successful (`app.c`:: APP_Tasks:: APP_STATE_READ_FLASH_DMA_MODE:: SQI_DMA_Read(FLASH_PAGE_ADDR)).
6. Indicates demonstration success using LED3 (Green LED) on the starter kit (`app.c`:: APP_Tasks:: APP_STATE_DONE:: BSP_SwitchONLED(LED_GRN)).
7. LED3 OFF, indicates demonstration FAILURE (stuck in one of the states).

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the SQI Peripheral Library Flash Read in DMA Mode Demonstration.

Description

To build this project, you must open the `flash_read_dma_mode.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/examples/peripheral/sqi/flash_read_dma_mode`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`flash_read_dma_mode.X`	`<install-dir>/apps/examples/peripheral/sqi/flash_read_dma_mode/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mz_ef_sk	pic32mz_ef_sk	Demonstrates the SQI DMA mode transfer on the PIC32MZ EF Starter Kit.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the SQI DMA mode transfer demonstration.

Description

This demonstration allows page writes and reads to/from an SQI Flash device.

1. First compile and program the target device. While compiling, select the appropriate configuration for the starter kit in use.
2. Check the state of LED3 on the starter kit in use to determine the status of the demonstration (LED ON indicates success, LED OFF indicates failure).

flash_read_pio_mode

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This demonstration application exercises the SQI module in PIO mode, by writing and reading a page in the SQI Flash (SST26VF032/SST26F032B) device on the PIC32MZ EC Starter Kit or PIC32MZ EF Starter Kit using the SQI Peripheral Library.

The following steps describe the functionality per the calls during run-time:

1. Sets up SQI to run at 25 MHz. (`system_init.c`:: SYS_Initialize, `app.c`:: APP_Initialize).
2. Once the clock frequency is programmed, the demonstration prepares the connected Serial Flash device for a write read transaction (`app.c`:: APP_Tasks:: APP_STATE_INIT_FLASH:: SQI_Flash_Setup).
3. Reads the device ID of the attached Flash device (`app.c`:: APP_Tasks:: APP_STATE_FLASH_ID_READ:: SQI_FlashID_Read)
4. Writes a page in Flash in PIO mode (`app.c`:: APP_Tasks:: APP_STATE_WRITE_FLASH:: SQI_PIO_PageWrite(FLASH_PAGE_ADDR)).
5. Reads the contents of the page in Flash and compares it to the data written to make sure the operation is successful (`app.c`:: APP_Tasks:: APP_STATE_READ_FLASH_PIO_MODE:: SQI_PIO_Read(FLASH_PAGE_ADDR)).
6. Indicates demonstration success using LED3 (Green LED) on the starter kit (`app.c`:: APP_Tasks:: APP_STATE_DONE:: BSP_SwitchONLED(LED_GRN)).
7. LED3 OFF, indicates demonstration FAILURE (stuck in one of the states).

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the SQI Peripheral Library Flash Read in PIO Mode Demonstration.

Description

To build this project, you must open the `flash_read_pio_mode.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/examples/peripheral/sqi/flash_read_pio_mode`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`flash_read_pio_mode.X`	`<install-dir>/apps/examples/peripheral/sqi/flash_read_pio_mode/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mz_ef_sk	pic32mz_ef_sk	Demonstrates the SQI PIO mode transfer on the PIC32MZ EF Starter Kit.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the SQI PIO mode transfer demonstration.

Description

This demonstration allows page writes and reads to/from an SQI Flash device.

1. First compile and program the target device. While compiling, select the appropriate configuration for the starter kit in use.
2. Check the state of LED3 on the starter kit in use to determine the status of the demonstration (LED3 ON indicates success, LED3 OFF indicates failure).

flash_read_xip_mode

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This demonstration application exercises the SQI module in XIP mode, by reading a page in the SQI Flash (SST26VF032/SST26VF032B) device on the PIC32MZ EC Starter Kit or the PIC32MZ EF Starter Kit. This demonstration uses PIO mode to write the Flash.

The following steps describe the functionality per the calls during run-time:

1. Sets up SQI to run at 25 MHz. (system_init.c:: SYS_Initialize, app.c:: APP_Initialize)
2. Once the clock frequency is programmed, the demonstration prepares the connected Serial Flash device for a write read transaction (app.c:: APP_Tasks:: APP_STATE_INIT_FLASH:: SQI_Flash_Setup).
3. Reads the device ID of the attached Flash device (app.c:: APP_Tasks:: APP_STATE_FLASH_ID_READ:: SQI_FlashID_Read).
4. Writes a page in Flash in PIO mode (app.c:: APP_Tasks:: APP_STATE_WRITE_FLASH:: SQI_PIO_PageWrite(FLASH_PAGE_ADDR)).
5. Reads the contents of the page in Flash using DMA mode and compares it to the data written to make sure the operation is successful (app.c:: APP_Tasks:: APP_STATE_READ_FLASH_DMA_MODE:: SQI_XIP_Read(FLASH_PAGE_ADDR)).
6. Indicates demonstration success using LED3 (Green LED) on the starter kit (app.c:: APP_Tasks:: APP_STATE_DONE:: BSP_SwitchONLED(LED_GRN)).
7. LED3 OFF, indicates demonstration FAILURE (stuck in one of the states).

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the SQI Peripheral Library Flash Read in XIP Mode Demonstration.

Description

To build this project, you must open the `flash_read_xip_mode.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/examples/peripheral/sqi/flash_read_pio_mode`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`flash_read_xip_mode.X`	`<install-dir>/apps/examples/peripheral/sqi/flash_read_xip_mode/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within ./firmware/src/system_config.

Project Configuration Name	BSP Used	Description
pic32mz_ef_sk	pic32mz_ef_sk	Demonstrates the SQI XIP mode transfer on the PIC32MZ EF Starter Kit.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the SQI XIP mode transfer demonstration.

Description

This demonstration allows page writes and reads to/from an SQI Flash device.

1. First compile and program the target device. While compiling, select the appropriate configuration for the starter kit in use.
2. Check the state of LED3 on the starter kit to determine the status of the demonstration (LED3 ON indicates success, LED3 OFF indicates failure).

Timer Peripheral Library Examples

This topic provides descriptions of the TMR Peripheral Library examples.

Introduction

Timer Peripheral Library Demonstration Applications Help

Description

This distribution package contains one Timer related firmware project that demonstrates the capabilities of the MPLAB Harmony Timer Peripheral Library.

This section describes the hardware requirement and procedures to run this firmware project on Microchip demonstration and development boards.

To learn more about MPLAB Harmony stacks and libraries refer to the related documentation in Volume V: MPLAB Harmony Framework Reference.

Demonstrations

This topic provides information on how to run the Timer Peripheral Library demonstration applications included in this release.

timer3_interrupt

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This demonstration uses Timer3 to generate an interrupt based on the time-out (1 second).

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the TMR Peripheral Library TMR Alarm Demonstration.

Description

To build this project, you must open the `timer3_interrupt.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/examples/peripheral/tmr/timer3_interrupt`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`timer3_interrupt.X`	`<install-dir>/apps/examples/peripheral/tmr/timer3_interrupt/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx795_pim_e16	pic32mx795_pim+e16	Demonstrates the Timer3 interrupt on the PIC32MX795F512L CAN-USB PIM and Explorer 16 Development Board combination.
pic32mx_usb_sk2	pic32mx_usb_sk2	Demonstrates the Timer3 interrupt on the PIC32 USB Starter Kit II.
pic32mz_ec_sk	pic32mz_ec_sk	Demonstrates the Timer3 interrupt on the PIC32MZ Embedded Connectivity (EC) Starter Kit.

pic32mz_ef_sk	pic32mz_ef_sk	Demonstrates the Timer3 interrupt on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit.
pic32mz_da_sk_intddr	pic32mz_da_sk_intddr	Demonstrates the Timer3 interrupt on the PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit.
pic32mz_ef_sk_16b	pic32mz_ef_sk	Demonstrates the Timer3 interrupt with 16-bit configuration on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit.
pic32mx_usb_sk2_freertos	pic32mx_usb_sk2	Demonstrates the Timer3 interrupt on the PIC32 USB Starter Kit II with FreeRTOS running on the configuration.
pic32mz_ef_sk_freertos	pic32mz_ef_sk	Demonstrates the Timer3 interrupt on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit with FreeRTOS running on the configuration.
pic32wk_sk	pic32wk_gpb_gpd_sk+module	Demonstrates the Timer3 interrupt on the PIC32WK Wi-Fi Starter Kit.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MX795F512L CAN-USB PIM and Explorer 16 Development Board

Jumper JP2 should be connected (shorted) for the LEDs to function.

PIC32MZ EC Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32 USB Starter Kit II

No hardware related configuration or jumper setting changes are necessary.

PIC32WK Wi-Fi Starter Kit

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the Timer demonstration.

Description

This demonstration shows the timer/counter capabilities using TMR peripheral library functions.

PIC32MZ, PIC32MX, and PIC32WK

1. First compile and program the target device. While compiling, select the configuration suitable for the hardware in use.
2. Check the state of the LEDs. LED3 ON status indicates success and LED3 OFF status indicates failure.

USART Peripheral Library Examples

This topic provides descriptions of the USART Peripheral Library examples.

Introduction

USART (USART) Peripheral Library Demonstration Applications Help

Description

This distribution package contains one USART related firmware project that demonstrates the capabilities of the MPLAB Harmony USART Peripheral Library.

This section describes the hardware requirement and procedures to run this firmware project on Microchip demonstration and development boards.

To learn more about MPLAB Harmony stacks and libraries refer to the related documentation in Volume V: MPLAB Harmony Framework Reference.

Demonstrations

This topic provides information on how to run the USART Peripheral Library demonstration applications included in this release.

uart_basic

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This demonstration uses USART2 in UART mode to transmit characters to the console and echo received characters on the console while turning on a LED.

Building the Application

for the USART Peripheral Library USART Alarm Demonstration.

Description

To build this project, you must open the `uart_basic.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/examples/peripheral/usart/uart_basic`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`uart_basic.X`	`<install-dir>/apps/examples/peripheral/usart/uart_basic/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Configuration Name	BSP Used	Description
pic32mx795_pim_e16	pic32mx795_pim+e16	Demonstrates the USART time-out function on the PIC32MX795F512L PIM and Explorer 16 Development Board combination.
pic32mx_usb_sk2	pic32mx_usb_sk2	Demonstrates the USART peripheral library on PIC32 USB Starter Kit II.
pic32mz_ef_sk	pic32mz_ef_sk	Demonstrates the USART peripheral library on PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit.
pic32mz_da_sk_intddr	pic32mz_da_sk_intddr	Demonstrates the USART peripheral library on PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit.
pic32mx795_pim_e16_freertos	pic32mx795_pim+e16	FreeRTOS version of the demonstration, which demonstrates the USART time-out function on the PIC32MX795F512L PIM and Explorer 16 Development Board combination.
pic32mz_ef_sk_freertos	pic32mz_ef_sk	FreeRTOS version of the demonstration, which demonstrates the USART peripheral library on PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit.
pic32mz_ef_sk_16b	pic32mz_ef_sk	microMIPS version of the demonstration, which demonstrates the USART peripheral library on PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit.
pic32mk_gp_db	pic32mk_gp_db	Demonstrates USART functionality on the PIC32MK General Purpose (GP) Development Board.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MX795F512L CAN-USB PIM and Explorer 16 Development Board

Connect a standard serial cable or USB-to-RS-232 adapter cable between the personal computer and the Explorer 16 Development Board P1 (UART) port.

PIC32MZ EF Starter Kit

Connect a USB cable to the J11 Mini Type B connector on the PIC32MZ EF Starter Kit. Connect this USB cable to the computer running the terminal emulation program.

PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit

Connect a USB cable to the J5 Mini Type B connector on the PIC32MZ DA Starter Kit. Connect this USB cable to the computer running the terminal emulation program.

PIC32 USB Starter Kit II

The optional MCP2200 Breakout Module, which is a USB-to-UART serial converter adapter board, may be used to run this demonstration. If the Breakout Module will be used, an additional interface is required. The acceptable interface is the Starter Kit I/O Expansion Board. Jumper from pins 48 and 46 of J11 header to RX and TX of the Breakout Module, respectively. In addition, GND must be shorted.

PIC32MK GP Development Board

Connect a USB cable to the Mini USB-to-UART connector on the bottom side of PIC32MK GP Development Board. Connect this USB cable to the computer running the terminal emulation program.

Running the Demonstration

Provides instructions on how to build and run the USART demonstration.

Description

This demonstration shows the USART capabilities using the USART Peripheral Library functions.

1. First compile and program the target device. While compiling, select the configuration suitable for the hardware in use.
2. Launch a console client (OS native/Tera Term, etc.,) and set the serial port settings to 9600-N-1.
3. Launch the demonstration. The following messages will appear in the console window:
```
*** UART Interrupt-driven Application Example ***
*** Type some characters and observe the LED turn ON ***
```
4. As indicated in the message, notice the typed characters echoed on the console window and LED.
5. Turn on LED (D3 on Explorer 16 Development Board or LED3 on the starter kit in use) indicating interrupt processing.

WDT Peripheral Library examples

This topic provides descriptions of the WDT Peripheral Library examples.

Introduction

Watchdog Timer (WDT) Peripheral Library Demonstration Applications Help

Description

This distribution package contains one WDT related firmware project that demonstrates the capabilities of the MPLAB Harmony Watchdog Timer Peripheral Library.

This section describes the hardware requirement and procedures to run this firmware project on Microchip demonstration and development boards.

To learn more about MPLAB Harmony stacks and libraries refer to the related documentation in Volume V: MPLAB Harmony Framework Reference.

Demonstrations

This topic provides information on how to run the WDT Peripheral Library demonstration applications included in this release.

wdt_timeout

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This demonstration exercises the watchdog time-out function using the WDT Peripheral Library.

Building the Application

for the WDT Peripheral Library WDT Alarm Demonstration.

Description

To build this project, you must open the `wdt_timeout.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/examples/peripheral/wdt/wdt_timeout`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`wdt_timeout.X`	`<install-dir>/apps/examples/peripheral/wdt/wdt_timeout/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Configuration Name	BSP Used	Description
pic32mx795_pim_e16	pic32mx795_pim+e16	Demonstrates the WDT time-out function on the PIC32MX795F512L PIM and Explorer 16 Development Board combination.
pic32mx_usb_sk2	pic32mx_usb_sk2	Demonstrates the WDT time-out function on the PIC32 USB Starter Kit II.
pic32mz_ec_sk	pic32mz_ec_sk	Demonstrates the WDT time-out function on the PIC32MZ Embedded Connectivity (EC) Starter Kit.
pic32mz_ef_sk	pic32mz_ef_sk	Demonstrates the WDT time-out function on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit.
pic32mx795_pim_e16_freertos	pic32mx795_pim+e16	FreeRTOS version of the demonstration, which demonstrates the WDT time-out function on the PIC32MX795F512L PIM and Explorer 16 Development Board combination.
pic32mz_ef_sk_freertos	pic32mz_ef_sk	FreeRTOS version of the demonstration, which demonstrates the WDT time-out function on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit.
pic32mz_ef_sk_16b	pic32mz_ef_sk	microMIPS version of the demonstration, which demonstrates the WDT time-out function on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 USB Starter Kit II

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ EC Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32MX795F512L CAN-USB PIM and Explorer 16 Development Board

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the WDT time-out demonstration.

Description

This demonstration exercises the WDT time-out function.

1. First compile and program the target device. While compiling, select the configuration suitable for the hardware in use.
2. The LEDs (LED1 on the PIC32 USB Starter Kit II, PIC32MZ EC Starter Kit, PIC32MZ EF Starter Kit and PIC32MZ DA Starter Kit, or LED D3 on the Explorer 16 Development Board) will blink during normal operation.

3. Press Switch 1 (PIC32 USB Starter Kit II, PIC32MZ EC Starter Kit, PIC32MZ EF Starter Kit and PIC32MZ DA Starter Kit) or Switch S3 (Explorer 16 Development Board) for the WDT time-out reset. If the demonstration is successful, LED3 (PIC32 USB Starter Kit II, PIC32MZ EC Starter Kit, PIC32MZ EF Starter Kit, and PIC32MZ DA Starter Kit) or LED D5 (Explorer 16 Development Board) will illuminate. If they do not, this indicates demonstration failure.

System Service Library Examples

Introduction

The example applications provide very simple single-purpose examples of how to use MPLAB Harmony system service libraries.

Description

System services have two primary types of implementations:

- System Service Libraries
- Low-Level Support

Most system service libraries follow the same basic model as a device driver (directly using a peripheral library to access hardware) or a middleware library (using a device driver to access hardware) as the rest of the system.

Command Processor System Service Examples

This topic provides descriptions of the Command Processor System Service examples.

Introduction

Command Processor System Service Library Demonstration Applications Help.

Description

This distribution package contains a firmware project that demonstrates the capabilities of the MPLAB Harmony Command Processor System Service. This section describes the hardware requirement and procedures to run this firmware project on Microchip demonstration and development boards.

To learn more about MPLAB Harmony stacks and libraries refer to the related documentation in Volume V: MPLAB Harmony Framework Reference.

Demonstrations

This topic provides information on how to run the Command Processor System Service Library demonstration applications included in this release.

command_appio

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This demonstration application exercises the Command Processor System Service by receiving user keyboard input and displaying the output string to the PIC AppIO window in MPLAB X IDE.

The demonstration application does the following:

1. Launches the application via MPLAB REAL ICE to a target device.
2. Displays "Ready to accept command input" in the PIC AppIO window in MPLAB X IDE at initialization.
3. Listens for user command input.
4. Supports two simple commands native to this application.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Command Processor System Service Demonstration.

Description

To build this project, you must open the `command_appio.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/examples/system/command_appio`

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
command_appio.X	<install-dir>/apps/examples/system/command_appio/firmware

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within ./firmware/src/system_config.

Project Configuration Name	BSP Used	Description
pic32mx_usb_sk2	pic32mx_usb_sk2	Demonstrates application I/O using the PIC32 USB Starter Kit II, the Starter Kit I/O Expansion Board and the MPLAB REAL ICE in-circuit debugger.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 USB Starter Kit II

This demonstration requires the MPLAB REAL ICE in-circuit emulator and the Starter Kit I/O Expansion Board.

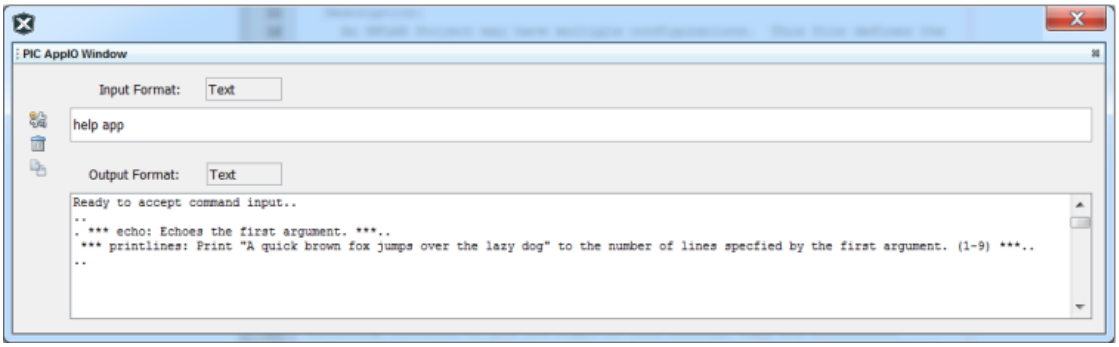
Running the Demonstration

Provides instructions on how to build and run the Command Processor System Service demonstration.

Description

Do the following to run the demonstration:

1. Ensure that the PIC32 USB Starter Kit II, the Starter Kit I/O Expansion Board, and the MPLAB REAL ICE in-circuit emulator are connected and ready.
2. Open the PIC AppIO window in MPLAB X IDE by selecting *Window* > *Debugging* > *PIC AppIO*.
3. Make sure the Output Format in PIC AppIO is set to "Text" (the default is 8-bit hex).
4. Debug the project.
5. Wait for "Ready to accept command input" to display in the Output Format section.
6. Type keyboard input into the Input Format bar and press <ENTER>.
7. Typing "help" will display the general help sections. Typing "help app" will display the commands unique to the application including a brief description.
8. Try the commands listed by "help app".



Console System Service Examples

This topic provides descriptions of the Console System Service examples.

Introduction

Console System Service Library Demonstration Applications Help.

Description

This distribution package contains a firmware project that demonstrates the capabilities of the MPLAB Harmony Console System Service. This section describes the hardware requirement and procedures to run this firmware project on Microchip demonstration and development boards.

To learn more about MPLAB Harmony stacks and libraries refer to the related documentation in Volume V: MPLAB Harmony Framework

Reference.

Demonstrations

This topic provides information on how to run the Console System Service Library demonstration applications included in this release.

multi_instance_console

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This demonstration illustrates the read and write operation on the Console when multiple Console instances are running simultaneously. The project has three configurations, the first configuration demonstrates simultaneous operation of the UART Console and the USB CDC Console, the second configuration shows the UART Console and the AppIO Console and the third configuration shows the USB CDC Console and the AppIO Console.

The application perform the following tasks:

- Each instance of Console prints a welcome message and prompts the user to enter a string
- The string that is entered by the user is echoed back to the Console
- The read operation completion can be checked through both polling and by indication of a callback
- The application is designed so that the two Console instances can run independently of each other

Libraries Used

The Console System Service resides as the top layer. The UART and USB Drivers are used depending on the choice of Console selected. The UART and USB Drivers call their respective Peripheral Libraries (PLIBs) to interact with the hardware. If AppIO is selected as the Console choice, the AppIO service is used, which is provided with the MPLAB X IDE C/C++ XC32 Compiler.

The application interacts directly with the Console System Service Library. The Console System Service Library, depending upon the choice of console driver, makes calls to the UART, USB, or AppIO.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Command Processor System Service Demonstration.

Description

To build this project, you must open the `multi_instance_console.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/examples/system/multi_instance_console`

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`multi_instance_console.X`	`<install-dir>/apps/examples/system/multi_instance_console/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx_usb_sk2	pic32mx_usb_sk2	Demonstrates USB CDC and AppIO Console instances running simultaneously on the PIC32 USB Starter Kit II.
pic32mx795_pim_e16	pic32mx795_pim+e16	Demonstrates UART and AppIO Console instances running simultaneously on the the PIC32MX795F512L PIM with the Explorer 16 Development Board.
pic32mz_ef_sk_meb2	pic32mz_ef_sk	Demonstrates UART and USB CDC Console instances running simultaneously on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 USB Starter Kit II

This demonstration requires the MPLAB REAL ICE in-circuit emulator for communication with AppIO and the Starter Kit I/O Expansion Board.

PIC32MX795F512L CAN-USB PIM with the Explorer 16 Development Board

This demonstration requires the MPLAB REAL ICE in-circuit emulator for communication with AppIO.

PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the Console System Service demonstration.

Description

Depending on the hardware in use, use on the of the following three procedures to run the demonstration.

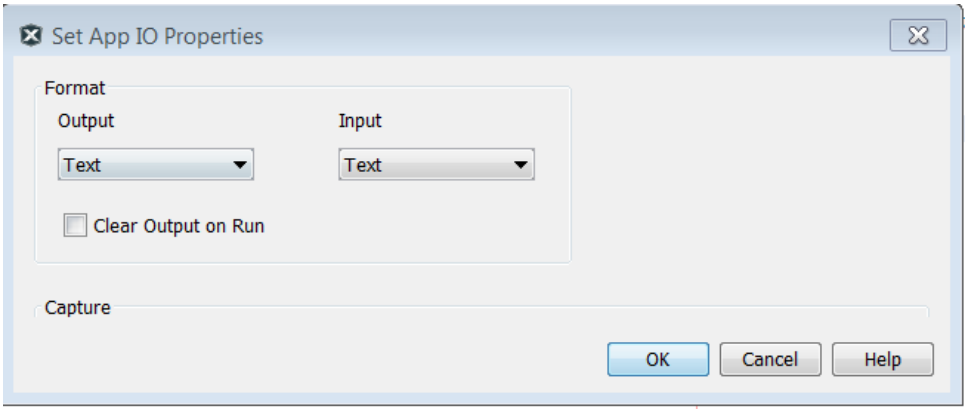
PIC32 USB Starter Kit II

This demonstration writes and reads to/from a terminal program running on a personal computer host and also from the AppIO interface in MPLAB X IDE.

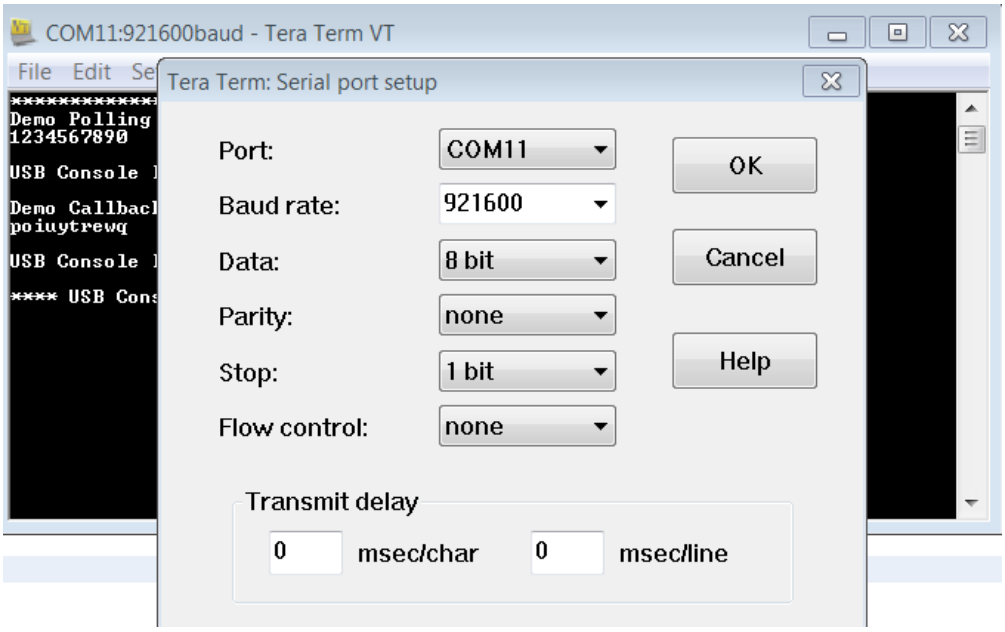
This demonstration utilizes the PIC32 USB Starter Kit II and requires the starter kit to be paired with the Starter Kit I/O Expansion Board.

Since AppIO uses the Debug interface, it also requires the MPLAB REAL ICE in-circuit emulator.

1. In MPLAB X IDE, select the hardware configuration to be used. Compile and program the target device in Debug mode with the selected configuration.
2. Open the AppIO window in MPLAB X IDE by selecting *Window* > *Debugging* > *PIC AppIO*.
3. Ensure that the Output Format in PIC AppIO is set to "Text" (the default is 8-bit hex), as shown in the following figure.

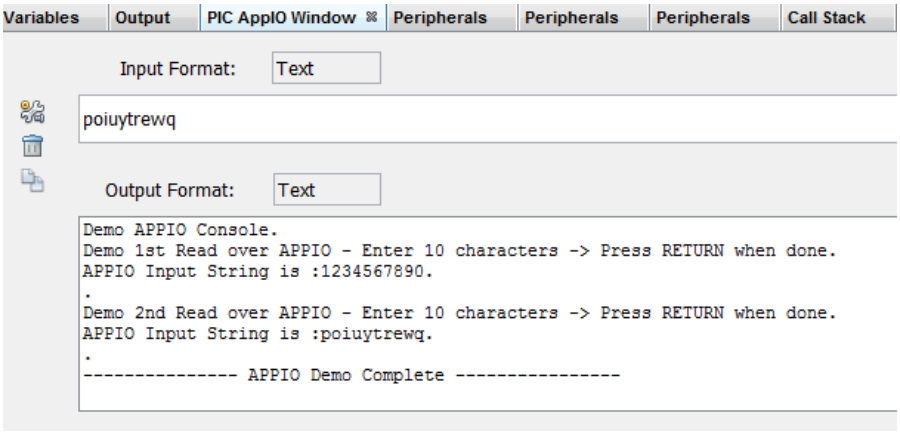


4. Before connecting the micro-USB cable to the host computer, the demonstration must be running for the terminal program to recognize the USB COM device. Once the demonstration is running, start a terminal emulator program (Tera Term shown) with serial port settings (921600 baud, 8 bit data, no parity, 1 bit stop, no flow control), as shown in the following figure.



Demonstration Output

When running the program in debug mode, the AppIO interface will display messages, as shown in the following figure.

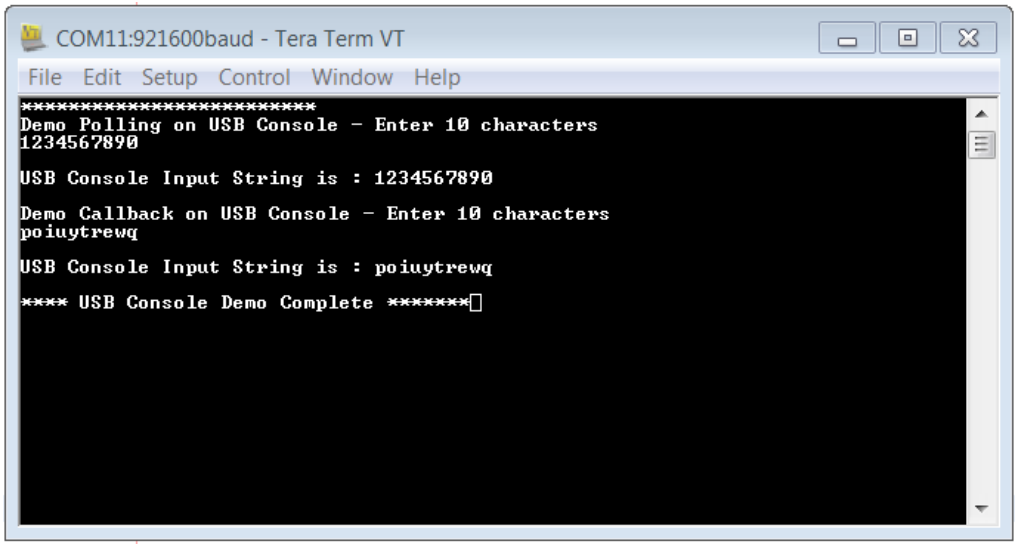


USB can be executed only after the AppIO demonstration is complete (see **Notes 1** and **2**).

Notes:

1. The AppIO interface implements a blocking read call expecting data from the debug interface until the user enters a RETURN. Therefore, all of the necessary AppIO read operations should be executed *first* before proceeding with the USB Console.
2. The USB interface implements a blocking write out to the console until the user presses any key to start the communication.

A display similar to the following figure can be expected after execution of the USB Console demonstration.

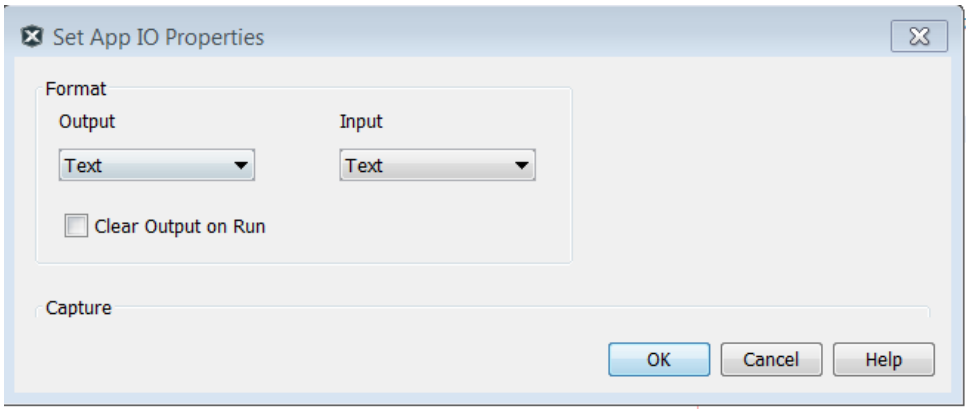


PIC32MX795F512 PIM and Explorer 16 Development Board

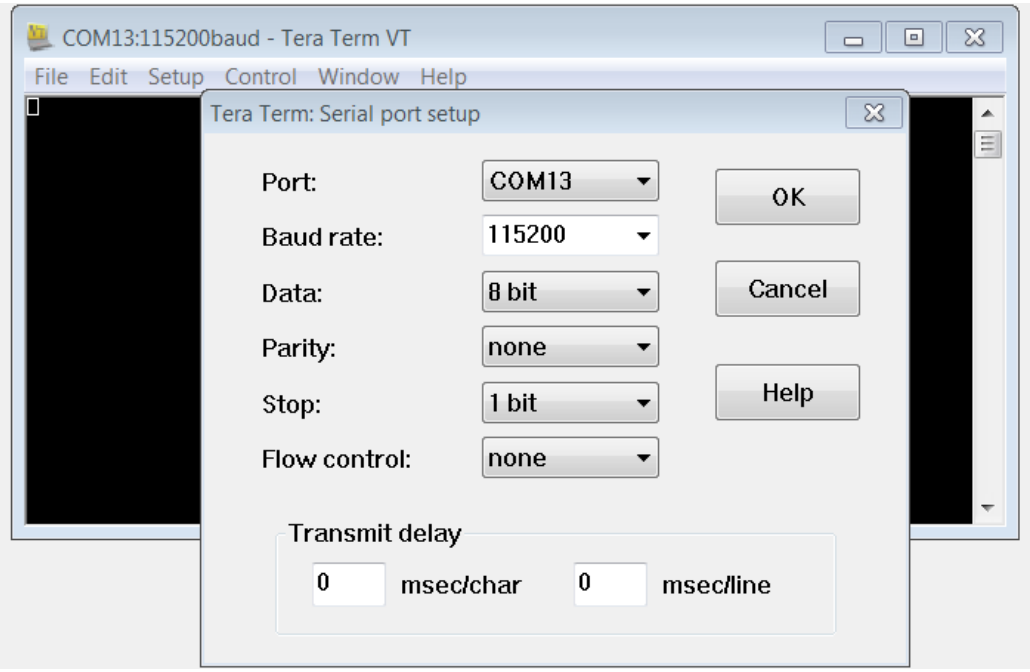
This demonstration reads and writes to a terminal program running on a personal computer host and also from the AppIO interface in MPLAB X IDE.

This demonstration utilizes the PIC32795F512L PIM paired with the Explorer 16 Development Board.

1. In MPLAB X IDE, select the hardware configuration to be used. Compile and program the target device in Debug mode with the selected configuration.
2. Open the AppIO window in MPLAB X IDE by selecting *Window > Debugging > PIC AppIO*.
3. Ensure that the Output Format in PIC AppIO is set to "Text" (the default is 8-bit hex), as shown in the following figure.

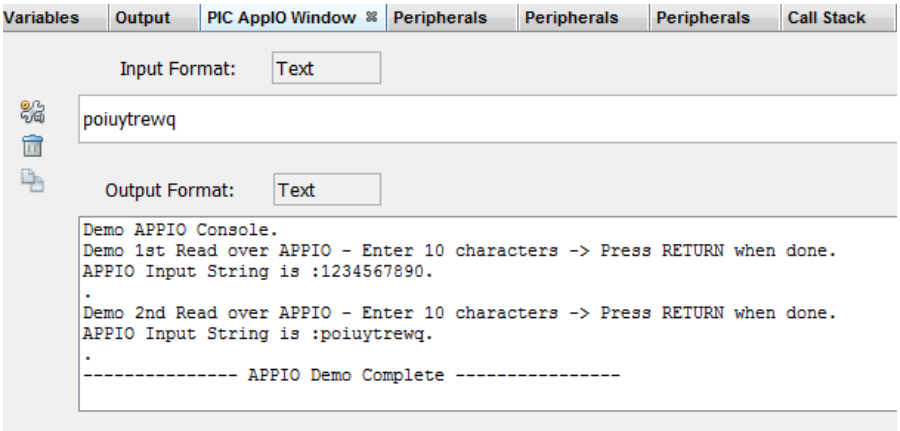


4. Connect the Explorer 16 Development Board to the host personal computer using a RS-232 UART connection.
5. A terminal program like Tera Term can be used with the settings shown in the following figure.



Demonstration Output

When running the program in debug mode, the AppIO interface will display messages, as shown in the following figure.

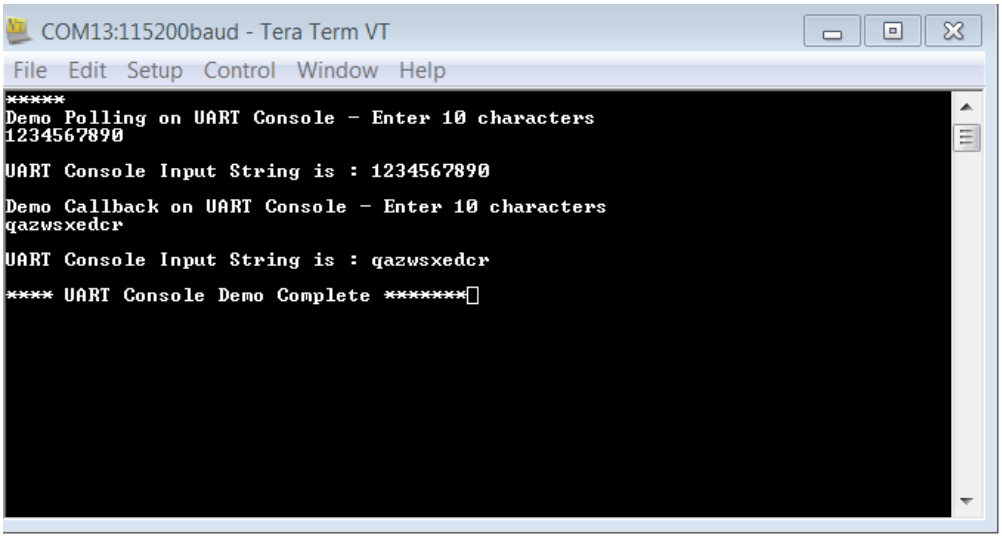


Once AppIO operation is complete, the UART Console can be executed (see **Notes 1** and **2**.

Notes:

1. The AppIO interface implements a blocking read call expecting data from the debug interface until the user enters a RETURN. Therefore, all of the necessary AppIO read operations should be executed *first* before proceeding with the UART Console.
2. The USB interface implements a blocking write out to the console until the user presses any key to start the communication.

A display similar to the following figure can be expected.

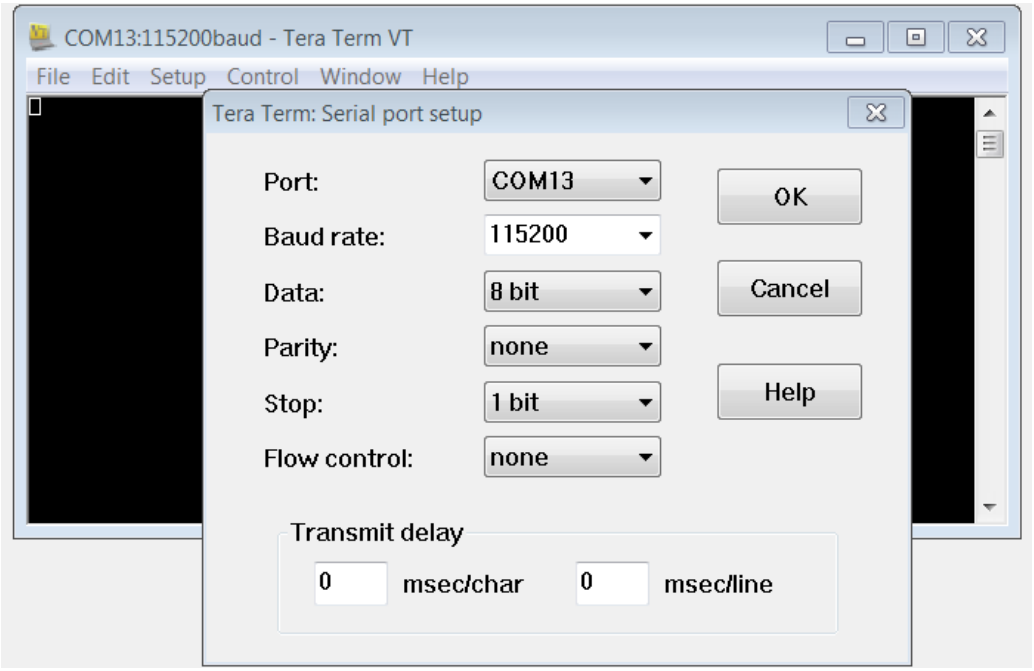


PIC32MZ EF Starter Kit

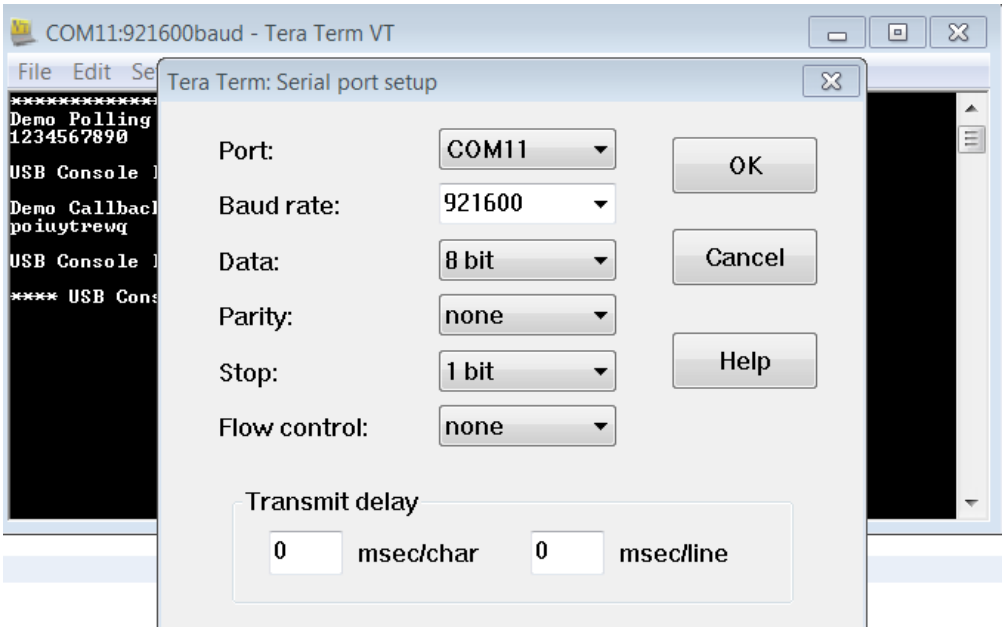
This demonstration reads and writes and to a terminal program running on a personal computer host that uses the UART and USB CDC 2 Console.

This demonstration utilizes the PIC32MZ EF Starter Kit.

1. In MPLAB X IDE, select the hardware configuration to be used. Compile and program the target device in Debug mode with the selected configuration.
2. Connect the PIC32MZ EF Starter Kit to the host personal computer using a mini-USB cable for the UART connection and a micro-USB cable for the USB connection.
3. A terminal program such as Tera Term can be used with the following UART settings.



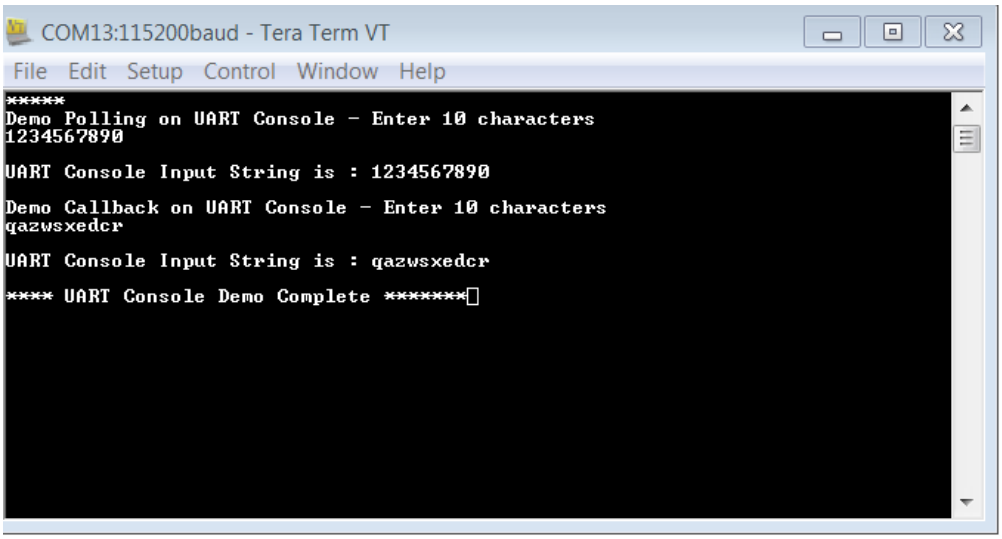
4. Before connecting the micro-USB cable to the host computer, the demonstration must be running for the terminal program to recognize the USB COM device. Once the demonstration is running, start a terminal emulator program (e.g., Tera Term) with serial port with the following USB settings: (921600 baud, 8 bit data, no parity, 1 bit stop, no flow control), as shown in the following figure.



Demonstration Output

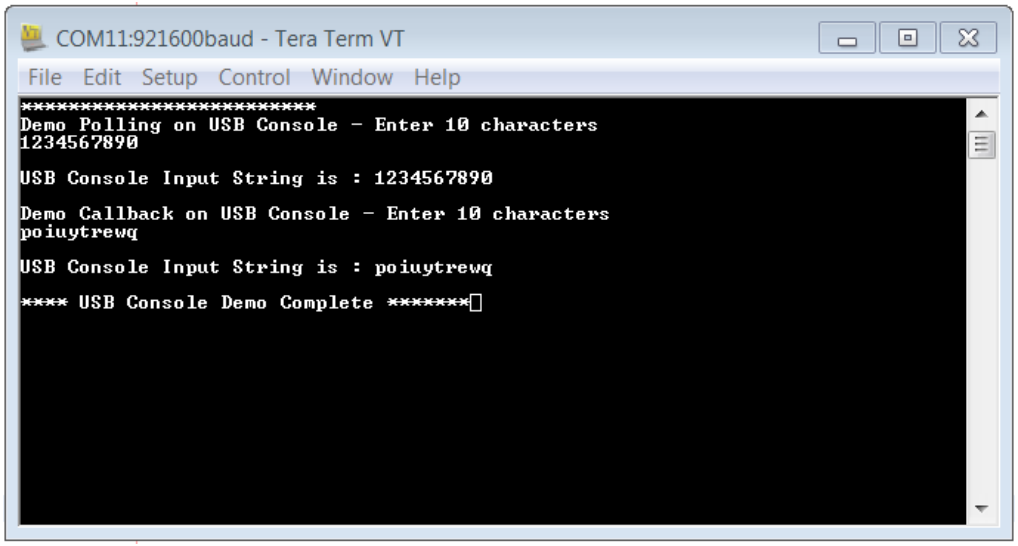
Since the UART and USB Console are running independently, the order in which the user interacts with each Console is irrelevant.

The UART Console should show an output similar to the following figure.



The USB Console should show an output similar to the following figure.

Note: The USB interface implements a blocking write out to the console until the user presses any key to start the communication.



Debug System Service Examples

This topic provides descriptions of the Debug System Service examples.

Introduction

Debug System Service Library Demonstration Applications Help.

Description

This distribution package contains a firmware project that demonstrates the capabilities of the MPLAB Harmony Debug System Service. This section describes the hardware requirement and procedures to run this firmware project on Microchip demonstration and development boards.

To learn more about MPLAB Harmony stacks and libraries refer to the related documentation in Volume V: MPLAB Harmony Framework Reference.

Demonstrations

This topic provides information on how to run the Debug System Service Library demonstration applications included in this release.

debug_uart

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This demonstration application exercises the Debug and Console System Services by routing messages from the Debug System Service through the Console System Service to a terminal program running on a personal computer via the UART communications protocol.

The demonstration application does the following:

- Demonstrates a direct console write by outputting a string to the terminal
- Demonstrates formatted and unformatted message writes to the terminal
- Demonstrates the use of the global error level
- Demonstrates debug output messaging using both polling and callback notification of completion
- Demonstrates what happens when the write queue overflows
- Demonstrates console flush in reaction to an error condition
- Demonstrates console read using both polling and callback notification
- Demonstrates an echo function in which the characters are written back to the terminal as they are read

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Debug System Service Demonstration.

Description

To build this project, you must open the `debug_uart.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/examples/system/debug_uart`

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`debug_uart.X`	`<install-dir>/apps/examples/system/debug_uart/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx795_pim_e16	pic32mx795_pim+e16	Demonstration console communication via UART RS-232 using the Explorer 16 Development Board and the PIC32MX795F512L PIM.
pic32mz_ef_sk	pic32mz_ef_sk	Demonstration console communication via UART RS-232 using using the PIC32MZ EF Starter Kit.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MX795F512L CAN-USB PIM combined with the Explorer 16 Development Board

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ EF Starter Kit

Ground pin 1 on the ICSP header (master clear bar) to ground, to reset the device.

Running the Demonstration

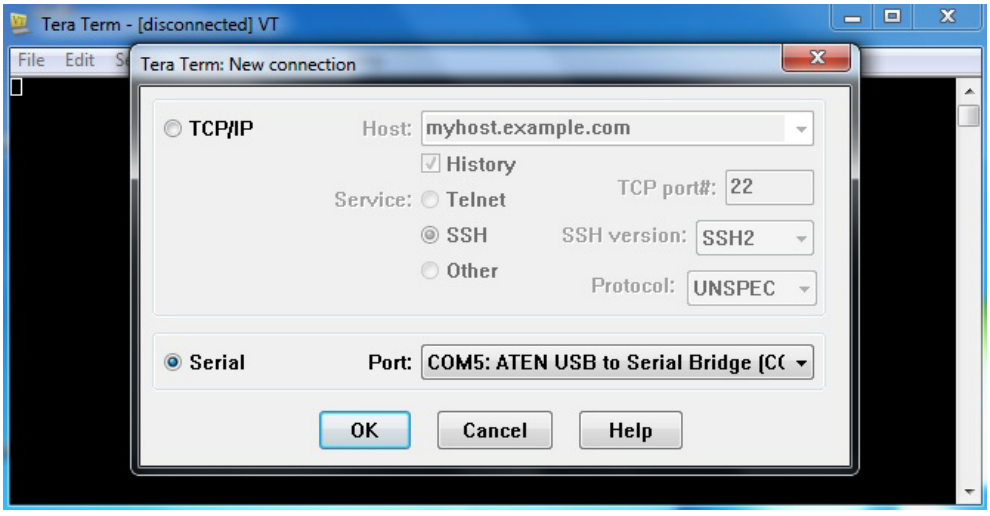
Provides instructions on how to build and run the Debug System Service demonstration.

Description

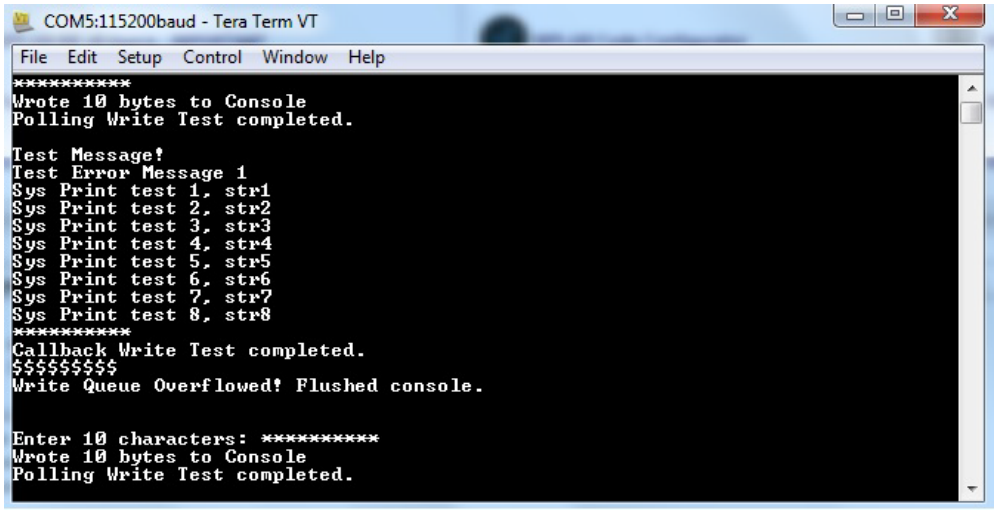
This demonstration writes and reads to/from a terminal program running on a personal computer host.

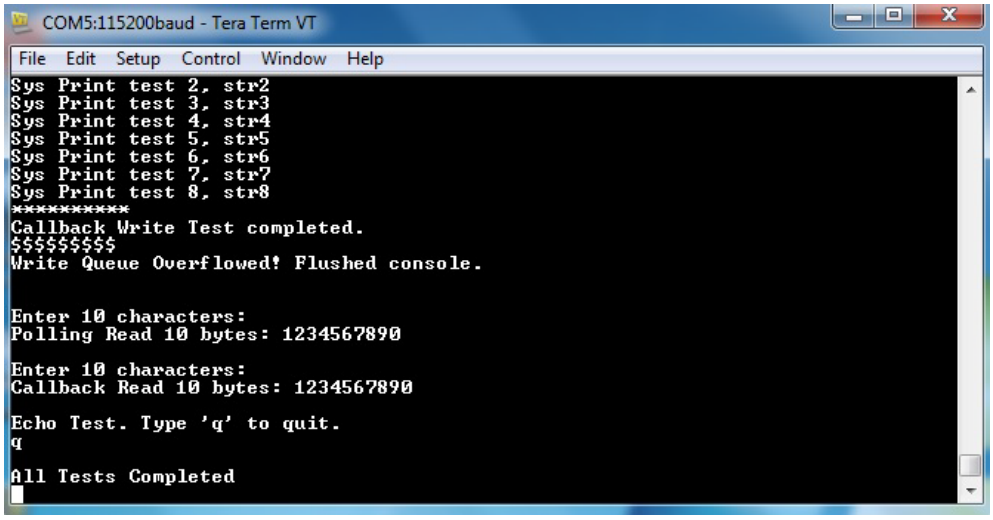
PIC32MX

1. First compile and program the target device. While compiling, select the configuration for the hardware in use.
2. Connect the Explorer 16 Development Board to the host personal computer using a RS-232 UART connection. Connect on board USB to CDC converter to the PC via USB cable.
3. The demonstration must be running for the terminal program to recognize the COM device. Start a terminal emulator program (Tera Term shown).



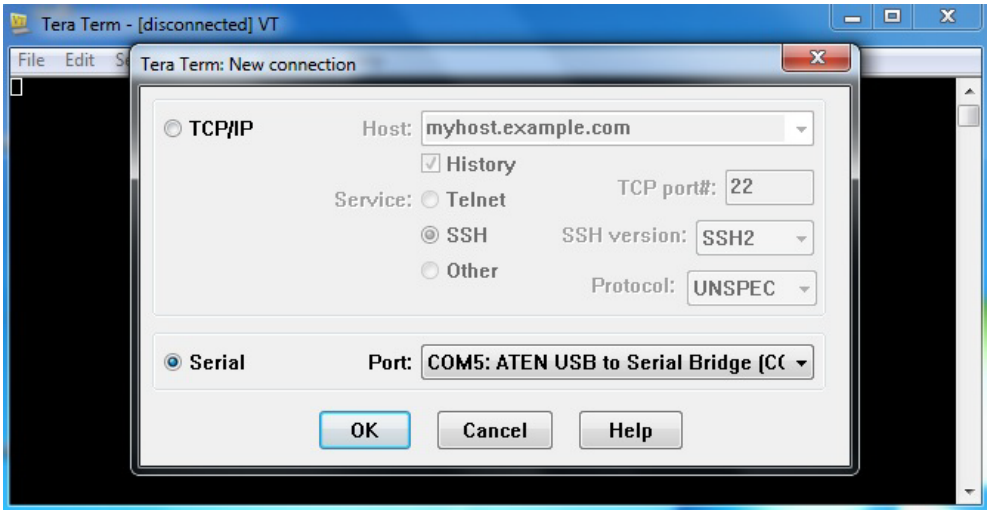
4. Configure the terminal for 115200 baud, 8 bits, 1 stop bit, no parity.
5. Press SW1 to start the demonstration.
6. Follow the instructions on the terminal.



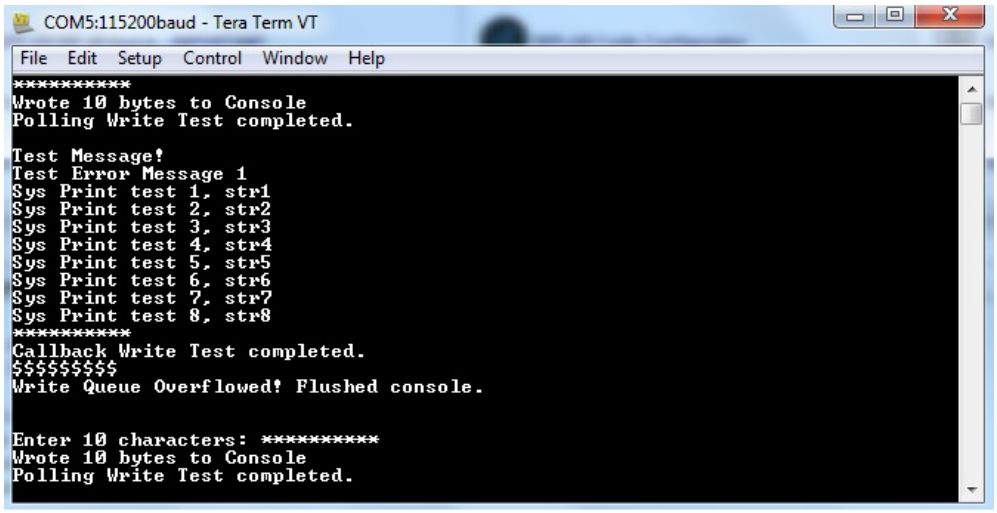


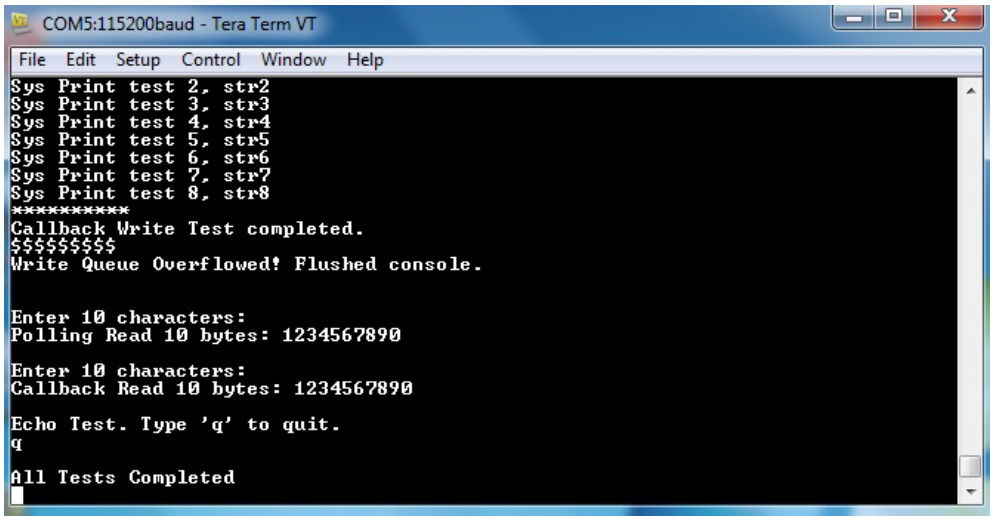
PIC32MZ

1. First compile and program the target device. While compiling, select the configuration for the hardware in use.
2. Connect the on board USB to the CDC converter, then connect to the personal computer using a USB cable.
3. The demonstration must be running for the terminal program to recognize the COM device. Start a terminal emulator program (Tera Term shown).



4. Configure the terminal for 115200 baud, 8 bits, 1 stop bit, no parity.
5. Follow the instructions on the terminal.





debug_usb_cdc_2

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This demonstration application exercises the Debug and Console System Services by routing messages from the Debug System Service through the Console System Service to a terminal program running on a personal computer via the USB-CDC communications protocol.

The demonstration application does the following:

- Demonstrates a direct console write by outputting a string to the terminal
- Demonstrates formatted and unformatted message writes to the terminal
- Demonstrates the use of the global error level
- Demonstrates debug output messaging using both polling and callback notification of completion
- Demonstrates what happens when the write queue overflows
- Demonstrates console flush in reaction to an error condition
- Demonstrates console read using both polling and callback notification
- Demonstrates an echo function in which the characters are written back to the terminal as they are read

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Debug System Service Demonstration.

Description

To build this project, you must open the `debug_usb_cdc_2.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/examples/system/debug_usb_cdc_2`

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`debug_usb_cdc_2.X`	`<install-dir>/apps/examples/system/debug_usb_cdc_2/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx_usb_sk2	pic32mx_usb_sk2	Demonstrates console communication through USB with the PIC32 USB Starter Kit II.
pic32mz_ef_sk	pic32mz_ef_sk	Demonstrates console communication through USB with the PIC32MZ EF Starter Kit.
pic32mx_usb_sk2_freertos	pic32mx_usb_sk2	FreeRTOS version of the console communication through USB using the PIC32 USB Starter Kit II.
pic32mz_ef_sk_freertos	pic32mz_ef_sk	FreeRTOS version of the console communication through USB using the PIC32MZ EF Starter Kit.
pic32mz_ef_sk_16b	pic32mz_ef_sk	Demonstrates console communication through USB using Micro-MIPS and PIC32MZ EF Starter Kit.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32 USB Starter Kit II

No hardware related configuration or jumper setting changes are necessary.

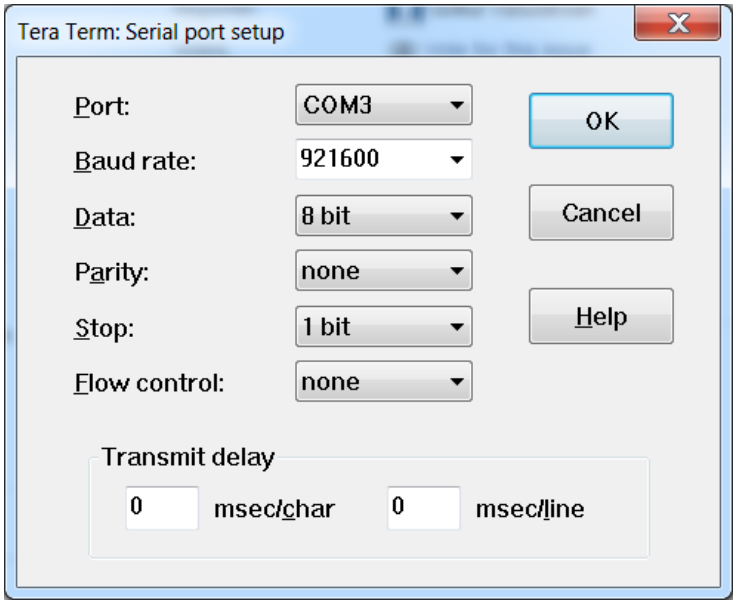
Running the Demonstration

Provides instructions on how to build and run the Debug System Service demonstration.

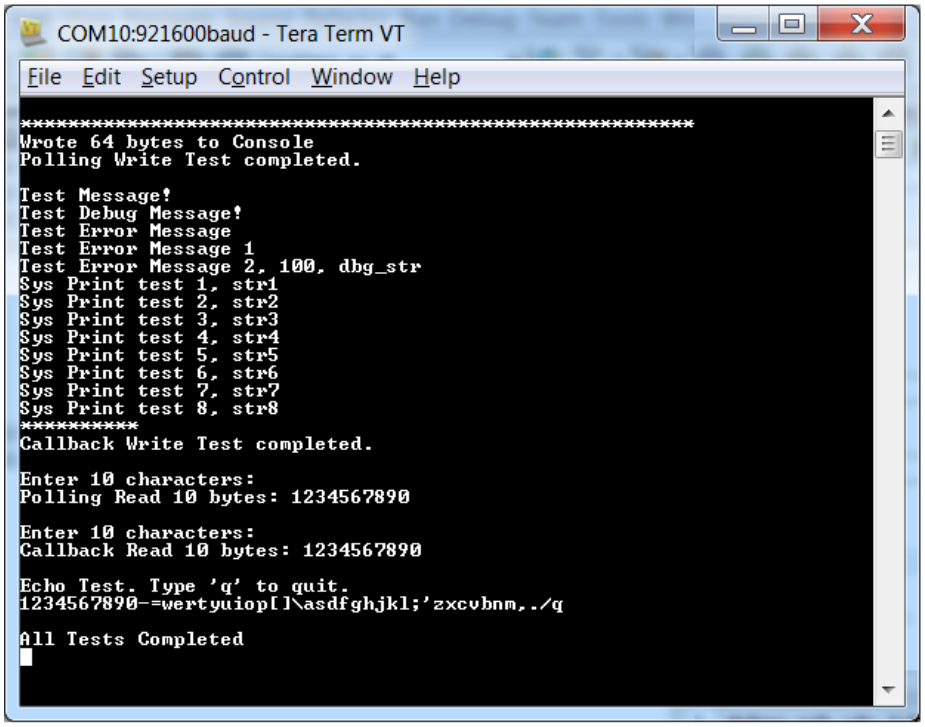
Description

This demonstration writes and reads to/from a terminal program running on a personal computer host.

1. First compile and program the target device. While compiling, select the appropriate configuration based on the hardware in use:
   - pic32mz_ef_sk, pic32mz_ef_sk_16b and pic32mz_ef_sk_freertos configuration for the PIC32MZ EF Starter Kit
   - pic32mx_usb_sk2 and pic32mx_usb_sk2_freertos configuration for the PIC32MX USB Starter Kit II
2. Connect the desired hardware to the host personal computer using an A to micro-A/B USB cable.
3. The demonstration must be running for the terminal program to recognize the COM device. Start a terminal emulator program (Tera Term shown) with serial port with the following settings: (921600 baud, 8 bit data, no parity, 1 bit stop, no flow control).



4. The terminal will indicate a successful connection. Click on the terminal window and press any key on the computer to start the demonstration.
5. Follow the instructions on the terminal.



Device Control System Service Examples

This topic provides descriptions of the Device Control System Service examples.

Introduction

Device Control System Service Library Demonstration Applications Help.

Description

This distribution package contains a firmware project that demonstrates the capabilities of the MPLAB Harmony Device Control System Service. This section describes the hardware requirement and procedures to run this firmware project on Microchip demonstration and development boards.

To learn more about MPLAB Harmony stacks and libraries refer to the related documentation in Volume V: MPLAB Harmony Framework Reference.

Demonstrations

This topic provides information on how to run the Device Control System Service Library demonstration applications included in this release.

devcon_cache_clean

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This example application demonstrates how cache coherency issues arise when transferring data out of memory using DMA. The MPLAB Harmony Device Control System Service contains various cache functions, which are used in this example to resolve the problem.

The application first allocates two buffers, with the intention of copying the data in one buffer to the other using DMA. The source buffer is first loaded with data and then the DMA transfer is initiated. After the transfer is complete, it is clear that the two buffers contain different values. This is an issue relating to cache coherency. When data is written by the CPU, it is stored in the cache but it is not written back to RAM unless the line is evicted or an explicit cache write-back instruction is executed. The cache instructions are accessed through the Device Control System Service and serve as a wrapper to the underlying MIPS® cache operations.

In this specific example, the data in the source buffer was never written back to RAM before being transferred to the destination buffer using DMA. Data that is stored in cache but has not yet been written back to RAM is termed "dirty". In the second part of the application, the correct method of maintaining cache coherency is demonstrated. After writing data to the source buffer, the function SYS_DEVCON_DataCacheClean is executed, forcing the data to be written back to main memory. When the DMA transfer is initiated, it then pulls the data out of RAM and transfers it to the destination buffer – this time, the data in both buffers will match.

If the application runs as intended, the red and green LEDs will be lit. The yellow LED will only be lit if an error has occurred.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Device Control System Service Cache Clean Demonstration.

Description

To build this project, you must open the `devcon_cache_clean.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/examples/system/devcon`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`devcon_cache_clean.X`	`<install-dir>/apps/examples/system/devcon/devcon_cache_clean/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mz_ef_sk	pic32mz_ef_sk	Demonstrates Device Control System Service cache functions using the PIC32MZ EF Starter Kit.
pic32mz_ef_sk_16b	pic32mz_ef_sk	Demonstrates Device Control System Service cache functions in microMIPS mode using the PIC32MZ EF Starter Kit.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the demonstration.

Description

This application demonstrates the necessity and proper use of the SYS_DEVCON_DataCacheClean function.

1. First compile and program the target device. When compiling, select the correct configuration for the device.
2. Observe the LEDs on the starter kit development board. If the red and green LEDs are both lit, the application executed successfully. If the yellow LED is lit, an error has occurred.

devcon_cache_invalidate

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This example application demonstrates how cache coherency issues arise when transferring data into memory using DMA. The MPLAB Harmony Device Control System Service contains various cache functions, which are used in this example to resolve the problem.

The application first allocates three buffers – two source buffers and one destination buffer. The two source buffers are filled with two different sets of data. The first buffer is copied to the destination buffer using DMA and the application checks to ensure they both contain the same data (which they do). The second source buffer is then copied to the destination buffer using DMA and once again the data is compared. This time the data does not match and once again, it is due to a cache coherency issue.

After the first DMA transfer and read of the destination buffer, the data is pulled into the cache. The second DMA transfer then updates the data contained in the destination buffer, but only the data contained in RAM. As far as the cache is aware, the stale data in the cache still matches the fresh data that is now in RAM. The cache simply sees that it is already storing a copy of the destination buffer, so it need not bother to pull the fresh data into itself when the CPU does the second set of reads. What we need is a way to tell the CPU to get rid of the stale cached data and pull in fresh data from main memory – this is known as an 'invalidate'. The example application demonstrates the proper technique for taking care of this issue. After the second DMA transfer, the destination buffer must be invalidated with the use of the function SYS_DEVCON_DataCacheInvalidate. This marks the data of interest in the cache as invalid. Now when the CPU performs a read on the destination buffer, fresh data is pulled out of main memory and into the cache, before being presented to the CPU.

If the application runs as intended, the red and green LEDs will be lit. The yellow LED will only be lit if an error has occurred.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Device Control System Service Cache Invalidate Demonstration.

Description

To build this project, you must open the `devcon_cache_invalidate.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/examples/system/devcon`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`devcon_cache_invalidate.X`	`<install-dir>/apps/examples/system/devcon/devcon_cache_invalidate/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mz_ef_sk	pic32mz_ef_sk	Demonstrates Device Control System Service cache functions using the PIC32MZ EF Starter Kit.

pic32mz_ef_sk_16b	pic32mz_ef_sk	Demonstrates Device Control System Service cache functions in microMIPS mode using the PIC32MZ EF Starter Kit.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the demonstration.

Description

This application demonstrates the necessity and proper use of the SYS_DEVCON_DataCacheInvalidate API.

1. First compile and program the target device. When compiling, select the correct configuration for the device.
2. Observe the LEDs on the starter kit development board. If the red and green LEDs are both lit, the application executed successfully. If the yellow LED is lit, an error has occurred.

devcon_sys_config_perf

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This demonstration provides an example of how to initialize the service and configure optimum system performance using the SYS_DEVCON_PerformanceConfig API.

The demonstration application does the following:

- Initializes the Device Control System Service
- Calls the SYS_DEVCON_PerformanceConfig API
- Waits in a busy loop

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Device Control System Service Demonstration.

Description

To build this project, you must open the `devcon_sys_config_perf.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/examples/system/devcon`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`devcon_sys_config_perf.X`	`<install-dir>/apps/examples/system/devcon/devcon_sys_config_perf/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx_eth_sk	pic32mx_eth_sk	Demonstrates various Device Control System Service cache functions using the PIC32 Ethernet Starter Kit.
pic32mz_ef_sk	pic32mz_ef_sk	Demonstrates Device Control System Service cache functions using the PIC32MZ EF Starter Kit.

pic32mz_ef_sk_16b	pic32mz_ef_sk	Demonstrates Device Control System Service cache functions in microMIPS mode using the PIC32MZ EF Starter Kit.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32 Ethernet Starter Kit

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the Device Control System Service demonstration.

Description

This demonstration initializes the Device Control System Service and demonstrates the use of the SYS_DEVCON_PerformanceConfig API.

1. Select the desired MPLAB X IDE project configuration:
   - pic32mz_ef_sk (for PIC32MZ EF devices)
   - pic32mx_eth_sk (for PIC32MX devices)
2. Build the selected configuration in the MPLAB X IDE project and program the demonstration board by selecting **Debug Main Project** from the Debug Menu. The program should build, download, and run.
3. Select **Pause** from the Debug menu. The program should pause in one of Tasks routines.
4. To verify that the device was initialized correctly, select *Window* > *PIC Memory Views* > *Peripherals* and check for the following:
   - for PIC32MZ, verify in the PRECON register
   - PFMWS is set to 2
   - PREFEN is set to 3
   - PIC32MX, verify in the CHECON register
   - PFMWS is set to 2
   - PREFEN is set to 3

DMA System Service Examples

This topic provides descriptions of the DMA System Service examples.

Introduction

DMA System Service Library Demonstration Applications Help.

Description

This distribution package contains a firmware project that demonstrates the capabilities of the MPLAB Harmony DMA System Service. This section describes the hardware requirement and procedures to run this firmware project on Microchip demonstration and development boards.

To learn more about MPLAB Harmony stacks and libraries refer to the related documentation in Volume V: MPLAB Harmony Framework Reference.

Demonstrations

This topic provides information on how to run the DMA System Service Library demonstration applications included in this release.

dma_crc

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This application demonstrates how to use the special function of CRC computation of the PIC32 DMA module by using the MPLAB Harmony DMA System Service Library. This section describes the hardware requirement and procedures to build and execute the demonstration project on Microchip development tools.

In this demonstration application, the DMA System Service sets up a memory to memory data transfer. It also enables the CRC engine to compute the CRC of the data being transferred from source location to destination location.

To know more about the MPLAB Harmony DMA System Service, configuring the DMA System Service and the APIs provided by the DMA System service, refer to the DMA System Service Library section of the help.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the DMA System Service Demonstration.

Description

To build this project, you must open the `dma_crc.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/examples/system/dma`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`dma_crc.X`	`<install-dir>/apps/examples/system/dma/dma_crc/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mz_ef_sk	pic32mz_ef_sk	This demonstration runs on the PIC32MZ2048EFM144 device on-board the PIC32MZ EF starter kit. The configuration can be used for generating CRC using a 16-bit polynomial in background mode.
pic32mz_ef_sk_16b	pic32mz_ef_sk	This demonstration runs on the PIC32MZ2048EFM144 device on-board the PIC32MZ EF starter kit. The configuration can be used for generating CRC using a 16-bit polynomial in background mode.
pic32mz_ef_sk_freertos	pic32mz_ef_sk	This demonstration runs on the PIC32MZ2048EFM144 device on-board the PIC32MZ EF starter kit. The configuration can be used for generating CRC using a 16-bit polynomial in background mode.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the demonstration.

Description

1. Compile the demonstration application. While compiling, select the appropriate MPLAB X IDE project configuration based on the demonstration board. Refer to Building the Application for details.
2. Run the demonstration application in Debug mode by clicking **Debug** in MPLAB X IDE
3. The LED2 should illuminate. The illumination of LED2 indicates that the data is being transferred from the source to the destination and the CRC is computed as the data was transferred.
4. Click **Pause** in the MPLAB X IDE.
5. In the Variables window, observe the debug variable `blockCrc`. The `blockCrc` variable should have a value 0x31C3, which is the CRC value for 16-bit polynomial with an initial seed value of '`0`'.

Note on CRC Calculation: The CRC computation on the various websites usually uses the table approach method for CRC calculation. The table approach CRC calculation method generally pads the incoming message with extra 0 bits (16 bits in the case of a 16-bit polynomial, and 32 bits in the case of 32-bit polynomial). The PIC32 DMA CRC generation does not pad with any extra bits. The PIC32 DMA CRC calculation engine strictly calculates CRC of the input data buffer.

To obtain comparable results with the websites we have to append extra bits; 16 bits, or two zeros (bytes) for 16-bit polynomial computation. 32 bits or four zeros (bytes) for 32-bit polynomial computation. Therefore, the data transfer call in our DMA System Service CRC computation application has the size appended with polynomial length as shown in the following example:

```
SYS_DMA_ChannelTransferAdd(channelHandle, flashBuff,
        (APP_DMA_CRC_BUFFER_SIZE + (crc.polyLength/8)) ,
        ramBuff,(APP_DMA_CRC_BUFFER_SIZE + (crc.polyLength/8)),
        (APP_DMA_CRC_BUFFER_SIZE + (crc.polyLength/8)));
```

dma_mem2mem

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This application demonstrates how to transfer a block of data from one memory location to another memory location using the MPLAB Harmony DMA System Service Library.

This section describes the hardware requirement and procedure to build and execute the demonstration project on Microchip development tools.

To know more about the MPLAB Harmony DMA System Service, configuring the DMA System Service and the APIs provided by the DMA System service, refer to the DMA System Service Library section of the help.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the DMA System Service Demonstration.

Description

To build this project, you must open the `dma_mem2mem.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/examples/system/dma`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`dma_mem2mem.X`	`<install-dir>/apps/examples/system/dma/dma_mem2mem/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx_usb_sk2	pic32mx_usb_sk2	This demonstration runs on the PIC32MX795F512L device of the PIC32 USB Starter Kit II.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 USB Starter Kit II

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the demonstration.

Description

To run the demonstration, perform the following steps:

1. Compile the demonstration application. While compiling, select the appropriate MPLAB X IDE project configuration based on the demonstration board. Refer to Building the Application for details.
2. Program the firmware into the target board using the Program button in MPLAB X IDE.
3. Demonstration success or failure is indicated by an LED:

Demonstration Board	Success Indication	Failure Indication
PIC32 USB Starter Kit II	LED3	LED1

RTCC System Service Examples

This topic provides descriptions of the RTCC System Service examples.

Introduction

RTCC System Service Library Demonstration Applications Help.

Description

This distribution package contains a firmware project that demonstrates the capabilities of the MPLAB Harmony RTCC System Service. This section describes the hardware requirement and procedures to run this firmware project on Microchip demonstration and development boards.

To learn more about MPLAB Harmony stacks and libraries refer to the related documentation in Volume V: MPLAB Harmony Framework Reference.

Demonstrations

This topic provides information on how to run the RTCC System Service Library demonstration applications included in this release.

rtcc_timestamps

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This demonstration application shows how to use the RTCC System Service. A simple callback is registered with the RTCC System Service and the alarm causes the current time to be stored in an array. The RTCC System Service can be set to be interrupt-driven to increase efficiency in that it will only be called when the alarm interrupt occurs.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Command Processor System Service Demonstration.

Description

To build this project, you must open the `rtcc_timestamps.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/examples/system/rtcc/rtcc_timestamps`

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`rtcc_timestamps.X`	`<install-dir>/apps/examples/system/rtcc/rtcc_timestamps/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mz_ef_sk	pic32mz_ef_sk	RTCC demonstration using the PIC32MZ EF Starter Kit.
pic32mz_ef_sk_freertos	pic32mz_ef_sk	FreeRTOS version of the RTCC demonstration using the PIC32MZ EF Starter Kit.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the RTCC System Service demonstration.

Description

Do the following to run the demonstration:

1. Compile the demonstration application. While compiling, select the appropriate MPLAB X IDE project configuration based on the hardware in use. Refer to Building the Application for details.
2. As shown in the following figure, create a breakpoint near line 155 with the following text:
   - `appData.rtccTimeStateMachine = APP_RTCC_TIMESTAMP_DONE;`

```
150
151             case APP_RTCC_TIMESTAMP_CHECK:
152                 /* real work is being done in the call back. Are we done? */
153                 if (timestamps.index == timestamps.limit)
154                 {
                        appData.rtccTimeStateMachine =  APP_RTCC_TIMESTAMP_DONE;
156                 }
157                 break;
158
159             case APP_RTCC_TIMESTAMP_DONE:
160                 break;
161         }
162     }
163
```

3. Run the demonstration application in Debug mode by clicking **Debug** in MPLAB X IDE.
4. When the debug session reaches the breakpoint (in approximately 10 seconds), the 'timestamps' array can be observed with the timestamps recorded at each second, as shown in the following figure.

```
139                 timestamps.index = 0;
140                 appHandle = SYS_RTCC_AlarmRegister(APP_RTCC_TIMESTAMP_Callback,
141                     (u
142
143                 /* con
144                 if (ap
145                 {
146                     SY
147                     ap                                          MESTAMP_CHECK;
148                 }
149                 break;
150
151             case APP_R
152                 /* rea                                         e we done? */
153                 if (timestamps.index == timestamps.limit)
154                 {
                        appData.rtccTimeStateMachine =  APP_RTCC_TIMESTAMP_DONE;
156                 }
157                 break;
158
```

```
Address = 0x800003CC,  timestamps.stamp[0] = 0x23595000
Address = 0x800003D0,  timestamps.stamp[1] = 0x23595100
Address = 0x800003D4,  timestamps.stamp[2] = 0x23595200
Address = 0x800003D8,  timestamps.stamp[3] = 0x23595300
Address = 0x800003DC,  timestamps.stamp[4] = 0x23595400
Address = 0x800003E0,  timestamps.stamp[5] = 0x23595500
Address = 0x800003E4,  timestamps.stamp[6] = 0x23595600
Address = 0x800003E8,  timestamps.stamp[7] = 0x23595700
Address = 0x800003EC,  timestamps.stamp[8] = 0x23595800
Address = 0x800003F0,  timestamps.stamp[9] = 0x23595900
Address = 0x800003F4,  timestamps.index = \n; 0x0a
Address = 0x800003F5,  timestamps.limit = \n; 0x0a
```

File System Demonstrations

This section provides descriptions of the File System demonstrations.

MPLAB Harmony is available for download from the Microchip website by visiting: http://www.microchip.com/mplabharmony. Once you are on the site, click the Downloads tab to access the appropriate download for your operating system. For additional information on this demonstration, refer to the “Applications Help” section in the MPLAB Harmony Help.

Description

Introduction

MPLAB Harmony File System Demonstration Help.

Description

This help file contains instructions and associated information about MPLAB Harmony File System demonstration applications, which are contained in the MPLAB Harmony Library distribution.

Demonstrations

Provides instructions on how to run the demonstration applications.

nvm_fat_single_disk

This demonstration uses a FAT12 image of a file on NVM Flash memory and demonstrates the working of all file system functions.

Description

This demonstration shows an example of implementing a FAT12 disk in device Flash memory. The demonstration contains a FAT12 disk image consisting of a Master Boot Record (MBR) sector, Logical Boot Sector, File Allocation Table, and Root Directory Area.

The demonstration opens an existing file named `FILE.TXT` and performs all file system related function calls on the file: SYS_FS_FileStat, SYS_FS_FileSize, SYS_FS_FileSeek, and SYS_FS_FileEOF. Finally, the string "Hello World" is written to this file. The string is then read and compared with the string that was written to the file. If the string compare is successful, LED indication is provided.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the NVM FAT Single Disk Demonstration.

Description

To build this project, you must open the `nvm_fat_single_disk.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/fs/nvm_fat_single_disk`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`nvm_fat_single_disk.X`	`<install-dir>/apps/fs/nvm_fat_single_disk/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx_bt_sk_int_dyn	pic32mx_bt_sk	This configuration runs on PIC32 Bluetooth Starter Kit. The media drivers are configured for Interrupt mode and dynamic operation.
pic32mx_usb_sk_int_dyn	pic32mx_usb_sk2	This configuration runs on the PIC32 USB Starter Kit II. The media drivers are configured for Interrupt mode and dynamic operation.
pic32mx_usb_sk_int_dyn_freertos	pic32mx_usb_sk2	This configuration runs on the PIC32 USB Starter Kit II and makes use of FreeRTOS as the underlying RTOS. The media drivers are configured for Interrupt mode and dynamic operation.
pic32mx_usb3_sk_int_dyn	pic32mx_usb_sk3	This configuration runs on PIC32 USB Starter Kit III. The media drivers are configured for Interrupt mode and dynamic operation.
pic32mz_ec_sk_int_dyn	pic32mz_ec_sk	This configuration runs on the PIC32MZ EC Starter Kit. The media drivers are configured for Interrupt mode and dynamic operation.
pic32mz_ef_sk_int_dyn	pic32mz_ef_sk	This configuration runs on the PIC32MZ EF Starter Kit. The media drivers are configured for Interrupt mode and dynamic operation.
pic32mz_ef_sk_int_dyn_freertos	pic32mz_ef_sk	This configuration runs on the PIC32MZ EF Starter Kit and makes use of FreeRTOS as the underlying RTOS. The media drivers are configured for Interrupt mode and dynamic operation.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 USB Starter Kit II

No hardware related configuration or jumper setting changes are necessary.

PIC32 USB Starter Kit III

No hardware related configuration or jumper setting changes are necessary.

PIC32 Bluetooth Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ EC Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the NVM FAT single disk demonstration.

Description

Select the MPLAB X IDE project configuration:

- pic32mx_usb_sk2_int_dyn (for PIC32MX devices)
- pic32mx_usb_sk_int_dyn_freertos (for PIC32MX devices)
- pic32mx_bt_sk_int_dyn (for PIC32MX devices)
- pic32mx_usb_sk3_int_dyn (for PIC32MX devices)
- pic32mz_ec_sk_int_dyn (for PIC32MZ EC devices)
- pic32mz_ef_sk_int_dyn (for PIC32MZ EF devices)
- pic32mz_ef_sk_int_dyn_freertos (for PIC32MZ EF devices)

Build the selected configuration in the MPLAB X IDE project and program the demonstration board. The execution status (pass/fail) of the demonstration is indicated by LEDs on the demonstration board.

Demonstration Board	Demonstration Success	Demonstration Failure
PIC32 USB Starter Kit II	LED3	LED1
PIC32 Bluetooth Starter Kit	Green LED	Red LED
PIC32 USB Starter Kit III	LED3	LED1
PIC32MZ EC Starter Kit PIC32MZ EF Starter Kit	Green LED	Red LED

About the Demonstration:

This demonstration shows an example of:

- Implementing a FAT12 disk in device Flash memory
- Opening a file for read or write
- Implements file system functions
- Closing a file

The demonstration contains a FAT12 disk image consisting of a Master Boot Record (MBR) sector, Logical Boot Sector, File Allocation Table, and Root Directory Area. This image is implemented in the file `nvm_disk_images.c` (this is project configuration specific file and is contained in the `nvm_disk_images` logical folder in the MPLAB X IDE project). The image contains a single file named `FILE.TXT` which contains the string "Data".

The demonstration opens an existing file named `FILE.TXT` and performs all file system related function calls on the file: SYS_FS_FileStat, SYS_FS_FileSize, SYS_FS_FileSeek, and SYS_FS_FileEOF. Finally, the string "Hello World" is written to this file. The string is then read and compared with the string that was written to the file. If the string compare is successful, LED indication is provided.

The demonstration application logic is implemented as a state machine in the APP_Tasks function in the file `main.c`.

1. The disk is first mounted using the SYS_FS_Mount function. The `/dev/nvma1` path instructs the mount command to mount an internal Flash volume. The volume is mounted against a FAT type file system and mounted at `/mnt/myDrive/`.
2. If the mount is successful, the application opens a file `FILE.TXT` for reading and writing (SYS_FS_FILE_OPEN_READ_PLUS) with a SYS_FS_FileOpen function. The valid file handle is received once a successful opening of the file is performed.
3. If file open is successful, the status of file `FILE.TXT` is stored in the appData.fileStatus structure, using SYS_FS_FileStat function.
4. If the file status check is successful, the size of the `FILE.TXT` is checked by passing the file handle to the SYS_FS_FileSize function.
5. If the file size check is successful, the size of file is compared with the size element received earlier as a part of appData.fileStatus structure. If both values match, the code moves to the next step of file seek.
6. The file pointer is then moved by 4 bytes from the start of the file by calling the function SYS_FS_FileSeek and passing parameter as SYS_FS_SEEK_SET (seek from the beginning of the file).
7. If the file seek operation is successful, the file pointer should have reached the end of the file. This is because, the content of the `FILE.TXT` had only 4 byte string = "Data". The end of file is verified by calling the function SYS_FS_FileEOF. If the function returns "true" (end of file reached), the next step is performed.
8. In the next step, the file pointer is moved again by [(-1)*size of file], from the end of the file by calling the function SYS_FS_FileSeek and passing parameter SYS_FS_SEEK_END (seek from the end of the file).
9. If the file seek operation is successful, the file pointer should have reached the beginning of the file. Then, 4 Bytes are read from the file using SYS_FS_FileRead function (into a buffer).
10. If the read is successful, the content of file (present in the buffer) is compared with the "Data" string. If the comparison passed, the code moves to the next step.
11. In the next step, the file pointer is moved again by [(-1)*size of file], from the end of the file by calling the function SYS_FS_FileSeek and passing parameter SYS_FS_SEEK_END (seek from the end of the file).
12. If the file seek operation is successful, the string "Hello World" is written to the file using SYS_FS_FileWrite function.
13. If the write operation is successful, the file pointer is moved to the beginning of the file.
14. If the file seek is successful, the contents of the file is read using SYS_FS_FileRead function (into a buffer).
15. If the read operation is successful, the content of the file (present in the buffer) is compared with the "Hello World" string using the strcmp function. A LED indicates the success of the demonstration.

nvm_mpfs_single_disk

This demonstration uses a MPFS image of two files on NVM Flash memory and demonstrates the working of all file system functions.

Description

This demonstration shows an example of implementing a MPFS disk in device Flash memory. The demonstration contains a MPFS disk image in the internal Flash memory. The disk image contains two files named:

- `FILE.txt`, Size = 11 Bytes. The content of the file is: "Hello World".
- `TEST.txt`, Size = 72 Bytes. The content of the file is: "This file contains a test string and it is meant for testing. 1234567890".

The demonstration performs all file system related function calls on the file: SYS_FS_FileRead, SYS_FS_FileStat, SYS_FS_FileSize, SYS_FS_FileSeek, SYS_FS_FileEOF. If all tests are successful, LED indication is provided.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the NVM MPFS Single Disk Demonstration.

Description

To build this project, you must open the `nvm_mpfs_single_disk.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/fs/nvm_mpfs_single_disk`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`nvm_mpfs_single_disk.X`	`<install-dir>/apps/fs/nvm_mpfs_single_disk/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx_bt_sk_int_dyn	pic32mx_bt_sk	This configuration runs on PIC32 Bluetooth Starter Kit. The media drivers are configured for Interrupt mode and dynamic operation.
pic32mx_usb_sk_int_dyn	pic32mx_usb_sk2	This configuration runs on the PIC32 USB Starter Kit II. The media drivers are configured for Interrupt mode and dynamic operation.
pic32mx_usb_sk_int_dyn_freertos	pic32mx_usb_sk2	This configuration runs on the PIC32 USB Starter Kit II and makes use of FreeRTOS as the underlying RTOS. The media drivers are configured for Interrupt mode and dynamic operation.
pic32mx_usb3_sk_int_dyn	pic32mx_usb_sk3	This configuration runs on PIC32 USB Starter Kit III. The media drivers are configured for Interrupt mode and dynamic operation.
pic32mz_ec_sk_int_dyn	pic32mz_ec_sk	This configuration runs on the PIC32MZ EC Starter Kit. The media drivers are configured for Interrupt mode and dynamic operation.
pic32mz_ef_sk_int_dyn	pic32mz_ef_sk	This configuration runs on the PIC32MZ EF Starter Kit. The media drivers are configured for Interrupt mode and dynamic operation.
pic32mz_ef_sk_int_dyn_freertos	pic32mz_ef_sk	This configuration runs on the PIC32MZ EF Starter Kit and makes use of FreeRTOS as the underlying RTOS. The media drivers are configured for Interrupt mode and dynamic operation.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 USB Starter Kit II

No hardware related configuration or jumper setting changes are necessary.

PIC32 USB Starter Kit III

No hardware related configuration or jumper setting changes are necessary.

PIC32 Bluetooth Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ EC Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the NVM MPFS single disk demonstration.

Description

Select the MPLAB X IDE project configuration:

- pic32mx_usb_sk2_int_dyn (for PIC32MX devices)
- pic32mx_usb_sk_int_dyn_freertos (for PIC32MX devices)
- pic32mx_usb_sk3_int_dyn (for PIC32MX devices)
- pic32mx_bt_sk_int_dyn (for PIC32MX devices)
- pic32mz_ec_sk_int_dyn (for PIC32MZ EC devices)
- pic32mz_ef_sk_int_dyn (for PIC32MZ EF devices)
- pic32mz_ef_sk_int_dyn_freertos (for PIC32MZ EF devices)

Build the selected configuration in the MPLAB X IDE project and program the demonstration board. The execution status (pass/fail) of the demonstration is indicated by LEDs on the demonstration board.

Demonstration Board	Demonstration Success	Demonstration Failure
PIC32 USB Starter Kit II PIC32 USB Starter Kit III	LED3	LED1
PIC32 Bluetooth Starter Kit	Green LED	Red LED
PIC32MZ EC Starter Kit PIC32MZ EF Starter Kit	Green LED	Red LED

About the demonstration:

This demonstration shows an example of:

- Implementing a MPFS disk in device Flash memory
- Opening a file for read
- Implements all the file system functions
- Closing a file

This demonstration shows an example of implementing a MPFS disk in device Flash memory. The demonstration contains a MPFS disk image in the internal Flash memory. The disk image contains two files named:

- `FILE.txt`, Size = 11 bytes. The content of the file is: "Hello World".
- `TEST.txt`, Size = 72 bytes. The content of the file is: "This file contains a test string and it is meant for testing. 1234567890".

The demonstration application logic is implemented as a state machine in the APP_Tasks function in the file `main.c`.

1. The disk is first mounted using the SYS_FS_Mount function. The `/dev/nvma1` path instructs the mount command to mount an internal Flash volume. The volume is mounted against a MPFS2 type file system and mounted at `/mnt/myDrive/`.
2. If the mount is successful, the application opens a file `FILE.txt` for reading with a SYS_FS_FileOpen function.
3. If the open is successful, the application opens another file `TEST.txt` for reading with SYS_FS_FileOpen function.
4. If the open is successful, the application checks for size of file `FILE.txt`, by passing the handle obtained during file open, to the function SYS_FS_FileSize.
5. If the file size matches the known value of 11 bytes, the application moves the file pointer for the file `TEST.txt` 10 bytes from the end of file, using the function SYS_FS_FileSeek.
6. If file seek is successful, 10 bytes of content of the file `TEST.txt` is read into the application buffer, using the function SYS_FS_FileRead.
7. If read is successful, the application buffer content is compared with the known string of `1234567890` using the strncmp function.
8. If the comparison is successful, the application checks if the file pointer for file "`TEST.txt`" has reached the end of file using the SYS_FS_FileEOF function.
9. If end of file is reached, a LED indicates the success of the demonstration.

nvm_sdcard_fat_mpfs_multi_disk

This demonstration uses NVM and a Secure Digital (SD) Card with MPFS and FAT image of file and performs a read/write/verify operation from file of one media to another media.

Description

This demonstration shows an example of using the MPLAB Harmony File System to access files across multiple disks and multiple file system. The demonstration contains a MPFS disk image in the internal Flash memory. The disk image contains a file named `abc.txt` with content "Hello World". Another disk is a SD card, which is formatted to FAT (FAT16 or FAT32). The demonstration reads the contents of `abc.txt` from the disk implemented on internal Flash memory and writes the contents to `FILE.TXT` on the SD card. A successful write is indicated by an illuminated LED.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the NVM SD Card FAT MPFS Multi-disk Demonstration

Description

To build this project, you must open the `nvm_sdcard_fat_mpfs_multi_disk.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/fs/nvm_sdcard_fat_mpfs_multi_disk`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
nvm_sdcard_fat_mpfs_multi_disk.X	<install-dir>/apps/fs/nvm_sdcard_fat_mpfs_multi_disk/firmware

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx795_pim_e16_int_dyn	pic32mx795_pim+e16	This configuration runs on the Explorer 16 Development Board using the PIC32MX795F512L PIM and PICtail Daughter Board for SD and MMC. The media drivers are configured for Interrupt mode and dynamic operation.
pic32mx795_pim_e16_int_dyn_freertos	pic32mx795_pim+e16	This configuration runs on the Explorer 16 Development Board using the PIC32MX795F512L PIM and PICtail Daughter Board for SD and MMC and makes use of FreeRTOS as the underlying RTOS. The media drivers are configured for Interrupt mode and dynamic operation.
pic32mx470_pim_e16_int_dyn	pic32mx470_pim+e16	This configuration runs on the Explorer 16 Development Board using the PIC32MX470F512L PIM and PICtail Daughter Board for SD and MMC. The media drivers are configured for Interrupt mode and dynamic operation.
pic32mz_ec_sk_int_dyn	pic32mz_ec_sk+meb2	This configuration runs on the PIC32MZ EC Starter Kit connected to the MEB II. The media drivers are configured for Interrupt mode and dynamic operation.
pic32mz_ef_sk_int_dyn	pic32mz_ef_sk+meb2	This configuration runs on the PIC32MZ EF Starter Kit connected to the MEB II. The media drivers are configured for Interrupt mode and dynamic operation.
pic32mz_ef_sk_int_dyn_freertos	pic32mz_ef_sk+meb2	This configuration runs on the PIC32MZ EF Starter Kit connected to the MEB II and makes use of FreeRTOS as the underlying RTOS. The media drivers are configured for Interrupt mode and dynamic operation.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

Explorer 16 Development Board, PIC32MX450/470F512L Plug-in Module (PIM) or PIC32MX795F512L CAN-USB Plug-in Module (PIM), and PICtail Daughter Board for SD and MMC

Use the following instructions for all Explorer 16 Development Board-based demonstration boards.

Since some peripheral functions are multiplexed through the Peripheral Pin Select (PPS) feature, the hardware on the following PIM must be modified by connecting the PIM pins, as follows:

PIC32MX470F512L PIM (MA320002-2) - PIM pin 99 to PIM pin 24 for CS

Refer to PIC32MX450/470F512L Plug-in Module (PIM) or PIC32MX795F512L CAN-USB Plug-in Module (PIM) for PIM pin locations.

Use the following general test setup for either PIM:

1. Insert the PIM into the Explorer 16 Development Board PIM connector.
2. Jumpers JP1, JP2 and JP3 on the PICtail Daughter Board for SD and MMC should connect to points 1 and 2 on their respective connectors (see PICtail Daughter Board for SD and MMC).
3. Insert the Daughter Board into the PICtail Plus connector on the Explorer 16 Development Board.
4. Insert a SD card into the SD card connector on the Daughter Board.
5. Power up the board.

PIC32MZ EC Starter Kit and MEB II

No hardware setting change is required. Insert the microSD card into the connector and power up the board.

PIC32MZ EF Starter Kit and MEB II

No hardware setting change is required. Insert the microSD card into the connector and power up the board.

Running the Demonstration

Provides instructions on how to build and run the NVM SD card FAT MPFS multi disk demonstration.

Description

Select the MPLAB X IDE project configuration:

- pic32mx795_pim_e16_int_dyn (for PIC32MX devices)
- pic32mx795_pim_e16_int_dyn_freertos (for PIC32MX devices)
- pic32mx470_pim_e16_int_dyn (for PIC32MX devices)
- pic32mz_ec_sk_int_dyn (for PIC32MZ EC devices)
- pic32mz_ef_sk_int_dyn (for PIC32MZ EF devices)
- pic32mz_ef_sk_int_dyn_freertos (for PIC32MZ EF devices)

Insert the SD card, which contains the file `FILE.TXT` (the file contains some arbitrary existing content).

Compile the selected configuration in the MPLAB X IDE project and run the code. After a few seconds, once the LED illuminates, remove the SD card from the SD Card PICtail Daughter Board and insert it into the SD card reader on a personal computer. Examine the contents of the file named `FILE.TXT`. The file should contain the string "Hello World".

Demonstration Board	Demonstration Success	Demonstration Failure
Explorer 16 Development Board	LED5 (D5)	LED6 (D6)
PIC32MZ EC Starter Kit PIC32MZ EF Starter Kit	Green LED	Red LED

About the Demonstration:

This demonstration shows an example of:

- Implementing a multi-disk demonstration with MPFS on internal Flash memory (NVM) and FAT File system (FAT16/ FAT32) on SD card
- Opening two files from two different disks and read the contents of the file from the first disk and write into the file of the second disk
- Closing the files

The demonstration contains a MPFS disk image in the internal Flash memory. The disk image contains a file named `abc.txt` with content "Hello World". Another disk is a SD card, which is formatted to FAT (FAT16 or FAT32). The demonstration reads the contents of `abc.txt` from the disk implemented on internal Flash memory and writes the contents to `FILE.TXT` on the SD card. A successful write is indicated by illumination of an LED.

The demonstration application logic is implemented as a state machine in the APP_Tasks function in the file `main.c`.

1. The first disk is mounted using the SYS_FS_Mount function. The `/dev/nvma1` path instructs the mount command to mount a internal Flash volume. The volume is mounted against a MPFS2 type file system and mounted at `/mnt/myDrive1/`.
2. If the mount is successful, the second disks is mounted using the SYS_FS_Mount function. The `/dev/mmcblka1` path instructs the mount command to mount a SD card volume. The volume is mounted against a FAT type file system and mounted at `/mnt/myDrive2/`.
3. If the mount is successful, the application opens the file `abc.txt` from `/mnt/myDrive1/` for reading and `FILE.TXT` from `/mnt/myDrive2/` for writing with the SYS_FS_FileOpen function.
4. If the file open is successful, the application reads 13 bytes from `abc.txt` of the internal Flash volume using SYS_FS_FileRead.
5. If the read is successful, the application closes `abc.txt` from the internal Flash volume, and then writes the 13 bytes into `FILE.TXT` of the SD card volume using the SYS_FS_FileWrite function. If the file write is successful, the application closes `FILE.TXT` from the SD card volume.
6. If file close is successful, a LED indicates the success of the operation.

nvm_sdcard_fat_multi_disk

This demonstration uses NVM and a Secure Digital (SD) card as media, searches a file from the NVM media, opens and reads the file, and then writes the data into another file in the SD card media.

Description

This demonstration shows an example of using the MPLAB Harmony File System to access files across multiple media. The demonstration contains a FAT12 disk image consisting of a Master Boot Record (MBR) sector, Logical Boot Sector, File Allocation Table, and Root Directory Area, placed in the internal Flash memory (NVM). Also, a SD card is used as another disk, which might have FAT16 or FAT32 implemented on it (dependent on the formatting of SD card). The demonstration searches the NVM media for a named `FILE.TXT`, opens and reads the contents of the file in NVM and copies the contents to the file, `FILE.TXT`, in the SD card. Once the copy is successful, an addition string "Test is successful" is added to the file. If the write operation is successful, LED indication is provided.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the NVM SD Card FAT Multi-disk Demonstration.

Description

To build this project, you must open the `nvm_sdcard_fat_multi_disk.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/fs/nvm_sdcard_fat_multi_disk`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
nvm_sdcard_fat_multi_disk.X	<install-dir>/apps/fs/nvm_sdcard_fat_multi_disk/firmware

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within ./firmware/src/system_config.

Project Configuration Name	BSP Used	Description
pic32mx795_pim_e16_int_dyn	pic32mx795_pim+e16	This configuration runs on the Explorer 16 Development Board using the PIC32MX795F512L PIM and PICtail Daughter Board for SD and MMC. The media drivers are configured for Interrupt mode and dynamic operation.
pic32mx795_pim_e16_int_dyn_freertos	pic32mx795_pim+e16	This configuration runs on the Explorer 16 Development Board using the PIC32MX795F512L PIM and PICtail Daughter Board for SD and MMC and makes use of FreeRTOS as the underlying RTOS. The media drivers are configured for Interrupt mode and dynamic operation.
pic32mx470_pim_e16_int_dyn	pic32mx470_pim+e16	This configuration runs on the Explorer 16 Development Board using the PIC32MX470F512L PIM and PICtail Daughter Board for SD and MMC. The media drivers are configured for Interrupt mode and dynamic operation.
pic32mz_ec_sk_int_dyn	pic32mz_ec_sk+meb2	This configuration runs on the PIC32MZ EC Starter Kit connected to the MEB II. The media drivers are configured for Interrupt mode and dynamic operation.
pic32mz_ef_sk_int_dyn	pic32mz_ef_sk+meb2	This configuration runs on the PIC32MZ EF Starter Kit connected to the MEB II. The media drivers are configured for Interrupt mode and dynamic operation.
pic32mz_ef_sk_int_dyn_freertos	pic32mz_ef_sk+meb2	This configuration runs on the PIC32MZ EF Starter Kit connected to the MEB II and makes use of FreeRTOS as the underlying RTOS. The media drivers are configured for Interrupt mode and dynamic operation.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

Explorer 16 Development Board, PIC32MX450/470F512L Plug-in Module (PIM) or PIC32MX795F512L CAN-USB Plug-in Module (PIM), and PICtail Daughter Board for SD and MMC

Use the following instructions for all Explorer 16 Development Board-based demonstration boards.

Since some peripheral functions are multiplexed through the Peripheral Pin Select (PPS) feature, the hardware on the following PIM must be modified by connecting the PIM pins, as follows:

PIC32MX470F512L PIM (MA320002-2) - PIM pin 99 to PIM pin 24 for CS

Refer to PIC32MX450/470F512L Plug-in Module (PIM) or PIC32MX795F512L CAN-USB Plug-in Module (PIM) for PIM pin locations.

Use the following general test setup for either PIM:

1. Insert the PIM into the Explorer 16 Development Board PIM connector.
2. Jumpers JP1, JP2 and JP3 on the PICtail Daughter Board for SD and MMC should connect to points 1 and 2 on their respective connectors (see PICtail Daughter Board for SD and MMC).
3. Insert the Daughter Board into the PICtail Plus connector on the Explorer 16 Development Board.
4. Insert a SD card into the SD card connector on the Daughter Board.
5. Power up the board.

PIC32MZ EC Starter Kit and MEB II

No hardware setting change is required. Insert the microSD card into the connector and power up the board.

PIC32MZ EF Starter Kit and MEB II

No hardware setting change is required. Insert the microSD card into the connector and power up the board.

Running the Demonstration

Provides instructions on how to build and run the NVM SD card FAT multi-disk demonstration.

Description

Select the MPLAB X IDE project configuration:

- pic32mx795_pim_e16_int_dyn (for PIC32MX devices)
- pic32mx795_pim_e16_int_dyn_freertos (for PIC32MX devices)
- pic32mx470_pim_e16_int_dyn (for PIC32MX devices)
- pic32mz_ec_sk_int_dyn (for PIC32MZ EC devices)
- pic32mz_ef_sk_int_dyn (for PIC32MZ EF devices)
- pic32mz_ef_sk_int_dyn_freertos (for PIC32MZ EF devices)

Insert the SD card, which contains the file `FILE.TXT` and the file contains "abc" (some arbitrary existing content).

Compile the selected configuration in the MPLAB X IDE project and run the code. After a few seconds, once the LED is illuminated, remove the SD card from the Board and insert it into the SD card reader on a personal computer. Examine the contents of the file named `FILE.TXT`. The file should contain the string "This data from NVM Disk Test is successful".

Demonstration Board	Demonstration Success	Demonstration Failure
Explorer 16 Development Board	LED5 (D5)	LED6 (D6)
PIC32MZ EC Starter Kit PIC32MZ EF Starter Kit	Green LED	Red LED

About the demonstration:

This demonstration shows an example of:

- Implementing a multi-disk demonstration with FAT12 on internal Flash memory (NVM) and FAT File system (FAT16/FAT32) on a SD card
- Opening two files from two different disks, read content of file from first disk and write into the file of second disk
- Closing the files

The demonstration contains a FAT12 disk image consisting of a Master Boot Record (MBR) sector, Logical Boot Sector, File Allocation Table, Root Directory Area, placed in the internal Flash memory (NVM). Also, a SD card is used as another disk, which might have FAT16 or FAT32 implemented on it (depends on the formatting of SD card).The demonstration searches the NVM media for a file named `FILE.TXT`, opens and reads the contents of the file in NVM and copies the contents to `FILE.TXT` to the SD Card. Once the copy is successful, an additional string "Test is successful" is added to the file. If the write operation is successful, LED indication is provided.

The demonstration application logic is implemented as a state machine in the APP_Tasks function in the file `main.c`.

1. The first disk is mounted using the SYS_FS_Mount function. The `/dev/nvma1` path instructs the mount command to mount a internal Flash volume. The volume is mounted against a FAT type file system and mounted at `/mnt/myDrive1/`.
2. If the mount is successful, the second disk is mounted using the SYS_FS_Mount function. The `/dev/mmcblka1` path instructs the mount command to mount a SD card volume. The volume is mounted against a FAT type file system and mounted at `/mnt/myDrive2/`.
3. If the mount is successful, the application opens the root directory of `/mnt/myDrive1/` with the function SYS_FS_DirOpen.
4. If the directory open is successful, the application searches for the file, `FILE.TXT`, using the wildcard character `FIL*.*`. The function used for directory search is SYS_FS_DirSearch.
5. If the directory search returns a file found, the directory is closed with the function SYS_FS_DirClose. Also, the searched file name is compared with the `FILE.TXT` name.
6. If the file name matches successfully, the application opens the file, `FILE.TXT`, from `/mnt/myDrive1/` for reading and `FILE.TXT` from `/mnt/myDrive2/` for writing with a SYS_FS_FileOpen function.
7. If the file open is successful, the application reads 27 Bytes from `FILE.TXT` of internal Flash volume using SYS_FS_FileRead.
8. If the read is successful, the application closes `FILE.TXT` from internal Flash volume and then writes the 27 bytes into `FILE.TXT` of SD card volume using SYS_FS_FileWrite.
9. If the write is successful, a character and string are written to the file using SYS_FS_FileCharacterPut and SYS_FS_FileStringPut.
10. If the write is successful, the application closes `FILE.TXT` from the SD card volume.
11. If file close is successful, a LED indicates the success of the operation.

sdcard_fat_single_disk

This demonstration uses a Secure Digital (SD) card with a FAT file system as media, performs a read/write/verify operation on the files using long file names (LFN), and performs directory creation.

Description

This demonstration shows an example of using the MPLAB Harmony File System to access and modify the contents of a SD card. The demonstration opens a file named `FILE_TOO_LONG_NAME_EXAMPLE_123.JPG` on the SD card, reads the content of the file, creates a directory named `Dir1` and inside the directory, writes the content into another file `FILE_TOO_LONG_NAME_EXAMPLE_123_1.JPG` (creates a copy of one file into another file, inside a directory).

The input file `FILE_TOO_LONG_NAME_EXAMPLE_123.JPG` is not provided along with the release package. It could be any arbitrary JPEG (image) file chosen by the user and then renamed to `FILE_TOO_LONG_NAME_EXAMPLE_123.JPG`. The reason for choosing a JPEG file for test purposes is that the duplicate file, `FILE_TOO_LONG_NAME_EXAMPLE_123_1.JPG`, created by the FS demonstration could be easily verified for correctness

by inserting the SD card into a computer and opening the `FILE_TOO_LONG_NAME_EXAMPLE_123_1.JPG` file. If the file opens for viewing on the computer, the test is deemed to have passed. Otherwise, if the file does not open (i.e., is corrupted), the test will be considered to have failed. Since the demonstration creates a directory named `Dir1`, it is important that the a folder with the same name does not exist on the SD card. If a directory named `Dir1` is already present on the SD card, the demonstration will fail.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the SD Card FAT Single Disk Demonstration.

Description

To build this project, you must open the `sdcard_fat_single_disk.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/fs/sdcard_fat_single_disk`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`sdcard_fat_single_disk.X`	`<install-dir>/apps/fs/sdcard_fat_single_disk/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx795_pim_e16_int_dyn	pic32mx795_pim+e16	This configuration runs on the Explorer 16 Development Board using the PIC32MX795F512L PIM and PICtail Daughter Board for SD and MMC. The media drivers are configured for Interrupt mode and dynamic operation.
pic32mx795_pim_e16_int_dyn_freertos	pic32mx795_pim+e16	This configuration runs on the Explorer 16 Development Board using the PIC32MX795F512L PIM and PICtail Daughter Board for SD and MMC and makes use of FreeRTOS as the underlying RTOS. The media drivers are configured for Interrupt mode and dynamic operation.
pic32mx470_pim_e16_int_dyn	pic32mx470_pim+e16	This configuration runs on the Explorer 16 Development Board using the PIC32MX470F512L PIM and PICtail Daughter Board for SD and MMC. The media drivers are configured for Interrupt mode and dynamic operation.
pic32mz_ec_sk_int_dyn	pic32mz_ec_sk+meb2	This configuration runs on the PIC32MZ EC Starter Kit connected to the MEB II. The media drivers are configured for Interrupt mode and dynamic operation.
pic32mz_ef_sk_int_dyn	pic32mz_ef_sk+meb2	This configuration runs on the PIC32MZ EF Starter Kit connected to the MEB II. The media drivers are configured for Interrupt mode and dynamic operation.
pic32mz_ef_sk_int_dyn_freertos	pic32mz_ef_sk+meb2	This configuration runs on the PIC32MZ EF Starter Kit connected to the MEB II and makes use of FreeRTOS as the underlying RTOS. The media drivers are configured for Interrupt mode and dynamic operation.
pic32mz_da_sk_adma	pic32mz_da_sk_intddr	This configuration runs on the PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit. The media driver is configured to use SD Host Controller ADMA2 Transfer mode operation.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

Explorer 16 Development Board, PIC32MX450/470F512L Plug-in Module (PIM) or PIC32MX795F512L CAN-USB Plug-in Module (PIM), and PICtail Daughter Board for SD and MMC

Use the following instructions for all Explorer 16 Development Board-based demonstration boards.

Since some peripheral functions are multiplexed through the Peripheral Pin Select (PPS) feature, the hardware on the following PIM must be modified by connecting the PIM pins, as follows:

PIC32MX470F512L PIM (MA320002-2) - PIM pin 99 to PIM pin 24 for CS

Refer to PIC32MX450/470F512L Plug-in Module (PIM) or PIC32MX795F512L CAN-USB Plug-in Module (PIM) for PIM pin locations.

Use the following general test setup for either PIM:

1. Insert the PIM into the Explorer 16 Development Board PIM connector.
2. Jumpers JP1, JP2 and JP3 on the PICtail Daughter Board for SD and MMC should connect to points 1 and 2 on their respective connectors (see PICtail Daughter Board for SD and MMC).
3. Insert the Daughter Board into the PICtail Plus connector on the Explorer 16 Development Board.
4. Insert a SD card into the SD card connector on the Daughter Board.
5. Power up the board.

PIC32MZ EC Starter Kit and MEB II

No hardware setting change is required. Insert the microSD card into the connector and power up the board.

PIC32MZ EF Starter Kit and MEB II

No hardware setting change is required. Insert the microSD card into the connector and power up the board.

PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit and MEB II

No hardware setting change is required.

Running the Demonstration

Provides instructions on how to build and run the SD card FAT single disk demonstration.

Description

Select the MPLAB X IDE project configuration:

- pic32mx795_pim_e16_int_dyn (for PIC32MX devices)
- pic32mx795_pim_e16_int_dyn_freertos (for PIC32MX devices)
- pic32mx470_pim_e16_int_dyn (for PIC32MX devices)
- pic32mz_ec_sk_int_dyn (for PIC32MZ EC devices)
- pic32mz_ef_sk_int_dyn (for PIC32MZ EF devices)
- pic32mz_ef_sk_int_dyn_freertos (for PIC32MZ EF devices)
- pic32mz_da_sk_adma (for PIC32MZ DA devices)

Insert the SD card, which contains the file named `FILE_TOO_LONG_NAME_EXAMPLE_123.JPG`. The file can be of any size.

Compile the selected configuration in the MPLAB X IDE project and run the code. After a few seconds, once the LED illuminates, remove the SD card from the SD Card PICtail Daughter Board (for PIC32MX) or MEB II (for PIC32MZ) and insert it into the SD card reader on a personal computer. Upon examining the contents of the SD card, a directory `Dir1` will have been created. Inside the `Dir1` folder, a file named `FILE_TOO_LONG_NAME_EXAMPLE_123_1.JPG` will be present. The file, `FILE_TOO_LONG_NAME_EXAMPLE_123_1.JPG`, will be a copy of `FILE_TOO_LONG_NAME_EXAMPLE_123.JPG`. Verify that both files open for viewing on a personal computer.

Demonstration Board	Demonstration Success	Demonstration Failure
Explorer 16 Development Board	LED5 (D5)	LED6 (D6)
PIC32MZ Embedded Connectivity (EC) Starter Kit	Green LED	Red LED

About the Demonstration:

This demonstration shows an example of:

- Implementing a FAT File system (FAT16/ FAT32) on a SD card
- Implementing long file name (LFN)
- Opening a file for read or write
- Closing a file

The demonstration opens a file named `FILE_TOO_LONG_NAME_EXAMPLE_123.JPG` on the SD Card, reads the content of the file, creates a directory named `Dir1` and inside the directory, writes the content into another file `FILE_TOO_LONG_NAME_EXAMPLE_123_1.JPG` (creates a copy of one file into another file, inside a directory).

The demonstration application logic is implemented as a state machine in the APP_Tasks function in the file `main.c`.

1. The disk is first mounted using the SYS_FS_Mount function. The `/dev/mmcblka1` path instructs the mount command to mount a SD card volume. The volume is mounted against a FAT type file system and mounted at `/mnt/myDrive/`.
2. If the mount is successful, the volume is unmounted by passing the mount name `/mnt/myDrive` to SYS_FS_Unmount function. This unmounting is done for demonstration purposes only. Real applications do not need to unmount unless it is required for the application.
3. If the unmount is successful, the mounting process is repeated.
4. If the mount is successful, the application opens the file `FILE_TOO_LONG_NAME_EXAMPLE_123.JPG` for reading with the SYS_FS_FileOpen function.
5. If the file open is successful,the application creates a directory named `Dir1`, with the SYS_FS_DirectoryMake function.
6. If the directory creation is successful, the application opens the file, `FILE_TOO_LONG_NAME_EXAMPLE_123_1.JPG`, inside the directory `Dir1`

for writing with the SYS_FS_FileOpen function. The attributes for file write is selected as "SYS_FS_FILE_OPEN_WRITE". Therefore, if the file does not exist, the file is created.

7. If the file open is successful, 512 bytes from the file `FILE_TOO_LONG_NAME_EXAMPLE_123.JPG` is read into application buffer using the SYS_FS_FileRead function.
8. If the file read successful, the 512 bytes is written from the application buffer to the `FILE_TOO_LONG_NAME_EXAMPLE_123_1.JPG` file using the SYS_FS_FileWrite function.
9. If the write operation is successful, the end of file for `FILE_TOO_LONG_NAME_EXAMPLE_123.JPG` is checked. If the end of file is not reached, the process of reading and writing continues (step 7 and step 8).
10. Once the end of the file is reached, both files are closed with the SYS_FS_FileClose function.
11. Finally, a LED indicates the success of the demonstration.

sdcard_msd_fat_multi_disk

This demonstration uses a USB Flash drive and a Secure Digital (SD) card as media. The application searches for a file on the USB Flash drive, opens and reads the file, and then writes the data into another file in the SD card media.

Description

This demonstration searches for a file using wildcard characters "`mch*.*`", reads the content of the file, and then writes the contents of the file to the SD card.

The demonstration application logic is implemented as a state machine in the APP_USB_MSDTasks and APP_SDCardTasks functions in the file `app.c`.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the SD Card MSD FAT Multi-disk Demonstration.

Description

To build this project, you must open the `sdcard_msd_fat_multi_disk`.X project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/fs/sdcard_msd_fat_multi_disk`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`sdcard_msd_fat_multi_disk.X`	`<install-dir>/apps/fs/sdcard_msd_fat_multi_disk/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mz_ec_sk_meb2	pic32mz_ec_sk+meb2	This configuration runs on PIC32MZ EC Starter Kit with the MEB II. The media drivers are configured for Interrupt mode and dynamic operation.
pic32mz_ef_sk_meb2	pic32mz_ef_sk+meb2	This configuration runs on PIC32MZ EF Starter Kit with the MEB II. The media drivers are configured for Interrupt mode and dynamic operation.
pic32mz_ef_sk_meb2_freertos	pic32mz_ef_sk+meb2	This configuration runs on the PIC32MZ EF Starter Kit connected to the MEB II and makes use of FreeRTOS as the underlying RTOS. The media drivers are configured for Interrupt mode and dynamic operation.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MZ EC Starter Kit and MEB II

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ EF Starter Kit and MEB II

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the SD card MSD FAT multi-disk demonstration.

Description

This demonstration shows an example of:

- Implementing a FAT File System on a SD card and a USB Flash drive
- Opening a file for read or write
- Searching for a file
- Checking for end of the file
- Closing a file

Do the following to run the demonstration:

1. Select the desired MPLAB X IDE project configuration for the hardware in use.
2. Copy the `mchpLogo.bmp` file from `<install-dir>/apps/fs/sdcard_msd_fat_multi_disk/firmware/src` to the USB Flash drive.
3. Insert the SD card into the MEB II.
4. Inset the USB Flash drive card into the starter kit.
5. Compile the selected configuration in the MPLAB X IDE project and run the code.
6. The demonstration searches for the `mchpLogo.bmp` on the USB Flash drive and copies it from the USB Flash drive to the SD card. The Green LED indicates the file has been successfully copied from the USB Flash drive to SD card.
7. Remove the SD card from the MEB II board and connect it to a personal computer to verify that the `mchpLogo.bmp` file is available on SD card.

Demonstration Board	Demonstration Success	Demonstration Failure
PIC32MZ EC Starter Kit PIC32MZ EF Starter Kit PIC32MZ DA Starter Kit	Green LED ON	Red LED ON

This demonstration searches for a file using wildcard characters "`mch*.*`", reads the content of the file, and then writes the contents of the file to the SD card.

The demonstration application logic is implemented as a state machine in the APP_USB_MSDTasks and APP_SDCardTasks functions in the file `app.c`.

The demonstration process is as follows:

1. The SD card is mounted using the SYS_FS_Mount function. The `/dev/mmcblka1` path instructs the mount function to mount a SD card volume. The volume is mounted with a FAT file system and mounted as `/mnt/sdDrive/`.
2. If the mount is successful, the SD card state machine waits until the USB Flash drive is connected and mounted.
3. The USB MSD task function opens an instance of the USB Host stack and enables the USB Host operations before mounting the USB Flash drive.
4. The USB Flash drive is mounted once it is connected using the SYS_FS_Mount function. The `/dev/sda1` path instructs the mount function to mount a SD USB Flash drive volume. The volume is mounted with the FAT file system as `/mnt/msdDrive/`.
5. If the USB Flash drive is mounted successfully, the application opens the root directory on the USB Flash drive and searches for the file using the wildcard characters "`mch*.*`" using SYS_FS_DirSearch.
6. If the search operation is successful, the application opens the file in read mode using SYS_FS_FileOpen.
7. If the file open is successful, the application sets the current working directory as `sdDrive` using SYS_FS_CurrentDriveSet and creates a new file on the SD card in write mode. The name of the new file is the same as was returned from SYS_FS_DirSearch.
8. If the file is created successfully, the application reads the data from the file on the USB Flash drive using SYS_FS_FileRead and writes it to the file on the SD card using SYS_FS_FileWrite until the end of file is reached.
9. Once the end of the file is reached, both files are closed with the SYS_FS_FileClose function.
10. Green LED indicates successful operation.

sqi_fat

This application demonstrates the use of the MPLAB Harmony File System with SQI Flash media.

Description

This application demonstrates the use of the MPLAB Harmony File System with SQI Flash media. The application formats the SQI Flash media, opens a file named “newFile.txt”, and writes the string “Hello World” to the file. The string is then read and compared with the string that was written to the file. If the string comparison is successful, LED indication is provided.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the SQI Flash Demonstration.

Description

To build this project, you must open `sqi_fat.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/fs/sqi_fat`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`sqi_fat.X`	`<install-dir>/apps/fs/sqi_fat/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mz_ef_sk	pic32mz_ef_sk	This configuration runs on PIC32MZ EF Starter Kit.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the SQI Flash demonstration.

Description

This demonstration illustrates how the MPLAB Harmony File System works with SST26VF SQI Flash media. The application does the following:

- Formats the SQI Flash media
- Opens a file named “newFile.txt”
- Writes the string “Hello World” to the file and then flushes the data onto the disk using the Sync operation
- Performs a file size check using the Stat operation, does a seek to the end of the file. The end of file is verified by using the EOF check operation.
- Another seek to the beginning of the file is done. The file data is read and compared. If the comparison is successful, the file is closed and the disk is unmounted.
- The application then enters an Idle state and LED3 is turned ON to indicate the the demonstration was successful
- If an error occurs at any stage of the demonstration, LED1 is turned ON to indicate the demonstration failed

sst25_fat

This application demonstrates the use of the MPLAB Harmony File System with SST25 Flash media.

Description

This application demonstrates the use of the MPLAB Harmony File System with SST25 Flash media. The application formats the SST25 Flash media, opens a file named “newFile.txt”, and writes the string “Hello World” to the file. The string is then read and compared with the string that was written to the file. If the string comparison is successful, LED indication is provided.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the SST25 Flash

Demonstration.

Description

To build this project, you must open `sst25_fat.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/fs/sst25_fat`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`sst25_fat.X`	`<install-dir>/apps/fs/sst25_fat/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
bt_audio_dk_int_dyn	bt_audio_dk	This configuration runs on PIC32 Bluetooth Audio Development Kit.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 Bluetooth Audio Development Kit

Ensure that switch S1 is set to PIC32_MCLR.

Running the Demonstration

Provides instructions on how to build and run the SST25 Flash demonstration.

Description

This demonstration illustrates the how the MPLAB Harmony File System works with the SST25VF SPI Flash media. The application does the following:

- Formats the SPI Flash media
- Opens a file named “newFile.txt”
- Writes the string “Hello World” to the file and then flushes the data onto the disk using the Sync operation
- Performs a file size check using the Stat operation, does a seek to the end of the file. The end of file is verified by using the EOF check operation.
- Another seek to the beginning of the file is done. The file data is read and compared. If the comparison is successful, the file is closed.
- The application then enters an Idle state and LED D9 is turned ON to indicate the demonstration was successful
- If an error occurs at any stage of the demonstration, LED D8 is turned ON to indicate the demonstration failed

Graphics Demonstrations

This section provides descriptions of the Graphics demonstrations.

MPLAB Harmony is available for download from the Microchip website by visiting: http://www.microchip.com/mplabharmony. Once you are on the site, click the Downloads tab to access the appropriate download for your operating system.

Description

Introduction

Graphics Library Demonstrations Applications Help

Description

This distribution package contains a variety of Graphics-related firmware projects that demonstrate the capabilities of the MPLAB Harmony Graphics library. This help file describes the hardware requirements, hardware setups, and procedures to run these demonstration projects on Microchip graphics boards.

Prior to using this demonstration, it is recommended to review the MPLAB Harmony Release Notes for any known issues. A PDF copy of the release notes is provided in the `<install-dir>/doc` folder of your installation.

To learn how to create graphics applications within MPLAB Harmony, see Volume III,MPLAB Harmony Graphics Composer User's Guide. In Volume III, the Advanced Topics section of the MPLAB Harmony Graphics Composer User's Guide also discusses how to add a third-party display to MPLAB Harmony graphics. In Volume I there are three Quick Start Guides related to graphics, including "Creating New Graphics Applications". To know more about MPLAB Harmony Graphics, configuring the library and the APIs provided; refer to the Graphics Library documentation found in Volume V.

Demonstrations

This topic provides information on how to run the Graphics Library demonstration applications included in this release.

aria_adventure

The application demonstrates advanced graphics techniques such as parallax and sprite animation.

Description

This application showcases how parallax and sprite animation can be achieved using the Aria Graphics Library.

The demonstration launches with a splash screen highlighting basic motion capability supported by the Aria Graphics Library. When running the application, the user can interface with it via capacitive single-fingered touch and swiping gestures.

The primary mode is presented in a mobile-game like style.

In addition, two other features demonstrated by this application includes the circular gauge widget and the ability to display updating text at a high update rate (showing the score count).

The lamb sprite character is animated using more than one hundred frames of animation and is blitted quickly using the GPU and the image preprocessing technique which stores lamb frames as raw pixels in DDR SDRAM.

Architecture

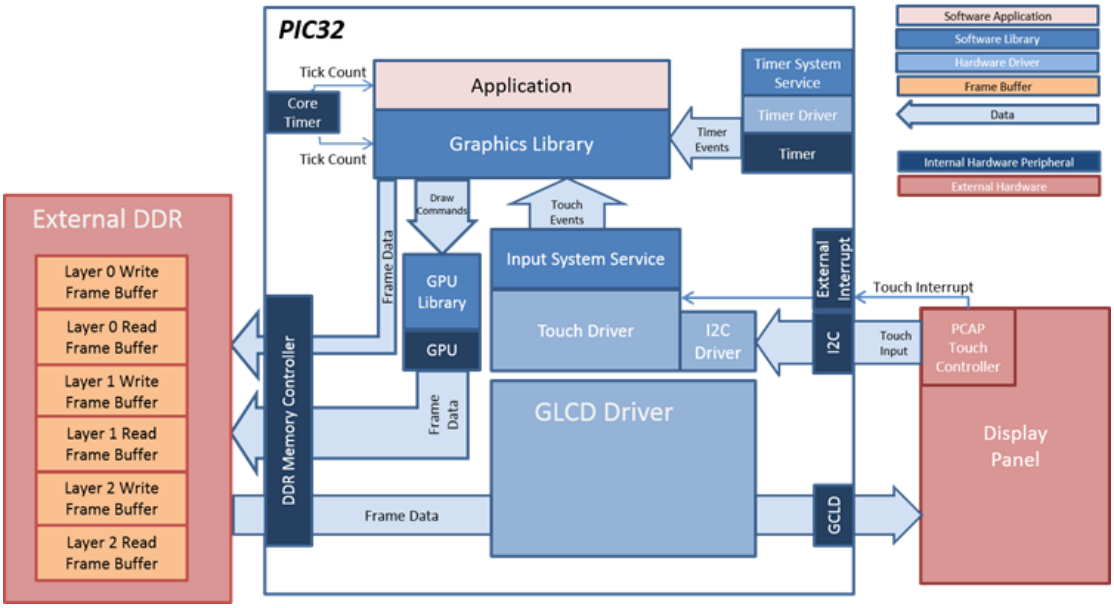
The diagrams below show the various software and hardware components for each configuration.

pic32mz_da_sk_extddr_meb2, pic32mz_da_sk_extddr_meb2_wvga

For these configurations, the application uses the Graphics library to render graphics to the display. The Graphics library passes draw commands into the GPU Library, which then draws the widgets and images to the three individual write frame buffers (one for each layer) that are stored on the external DDR2 Memory. The GLCD display controller peripheral continuously transfers frame data from the all three read buffers onto to the LCD display using the DDR2 Memory Controller. The write and read frame buffer pairs are swapped independently as required when the Graphics Library finishes rendering to the write frame buffer and GLCD driver signals the GLCD peripheral to change its read location.

The application also features user touch input through the integrated touch screen on the display panel. Touch input from the touch controller goes thru the I2C port, and the Input System Service acquires the touch input information from the touch and I2C drivers. The Input System Service sends touch events to the Graphics library which processes these events and updates the frame data accordingly.

The core timer is used by the application and the Graphics library to manage the movement of the splash screen.



Demonstration Features

- Integrated PCAP Touch Input (Input System Service Library Help) (High-Performance 4.3 Inch WQVGA Display Module with maXTouch)
- Three graphics layer supported via the GLCD peripheral on the PIC32MZ DA device (GLCD Controller Peripheral Library)
- GPU peripheral supported, can be enabled/disabled at run-time (GFXLIB Nano2D Driver Library)
- 32-bit RGBA8888 color depth support (16.7 million unique colors)
- Per-layer frame double-buffering
- Image compression techniques using Run-Length Encoding, PNG, and JPEG
- Simultaneous MP3 decoding from internal flash

Tools Setup Differences

For all configurations:

- *Use Graphics Stack* is selected in MHC. This enables the graphics and touch libraries and drivers for the pic32mz_da_sk_extddr_meb2, and pic32mz_da_sk_extddr_meb2_wvga.
- Pin Settings: External Interrupt 4 mapped to pin A14 (RB1)

Building the Application

This sections describes how to build the application for the demonstration.

Description

To build this project, you must open the aria_adventuret.X project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is <install-dir>/apps/gfx/aria_adventure.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
aria_adventure.X	<install-dir>/apps/gfx/aria_adventure/firmware

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within ./firmware/src/system_config.

Project Configuration Name	BSP Used	Description
pic32mz_da_sk_extddr_meb2	pic32mz_da_sk_extddr+meb2	Demonstration for the PIC32MZ Embedded Graphics with External DRAM (DA) Starter Kit plus Multimedia Expansion Board II (MEB II).
pic32mz_da_sk_extddr_meb2_wvga	pic32mz_da_sk_extddr+meb2+wvga	Demonstration for the PIC32MZ Embedded Graphics with External DRAM (DA) Starter Kit Memory plus the Multimedia Expansion Board II (MEB II) with a High-Performance (5") WVGA Display Module with maXTouch.

Configuring the Hardware

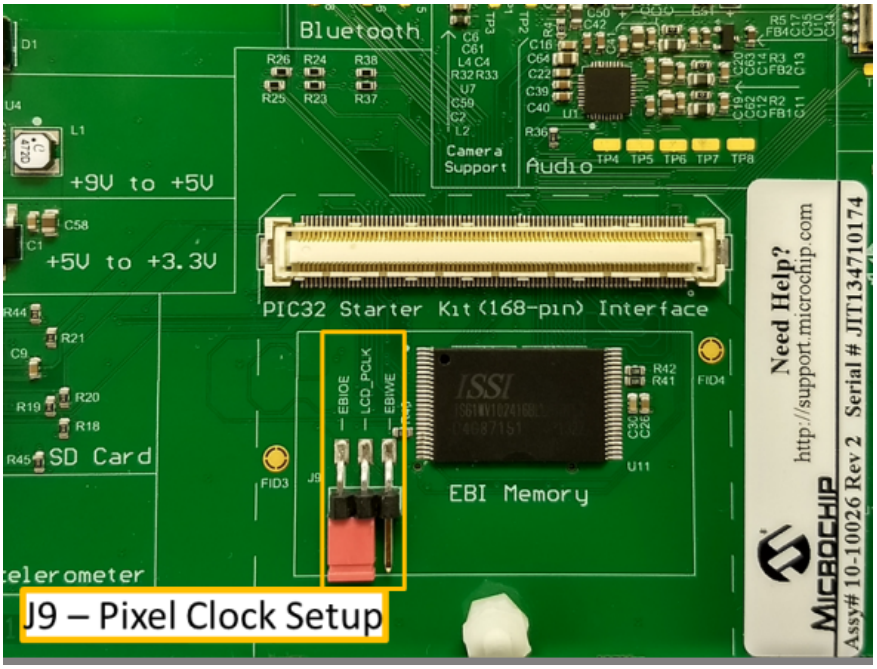
This section details how to configure the hardware for the demonstration.

Description

PIC32MZ Embedded Graphics with DA Starter Kit with the Multimedia Expansion Board II (MEB II) (First and Second Generation)

Configurations: pic32mz_da_sk_extddr_meb2,, and pic32mz_da_sk_extddr_meb2_wvga

- These configurations require that the J9 jumper be set to provide the GLCD's pixel clock. Set the J9 jumper to connect EPIOE to LCD_PCLK. The J9 jumper is located on the bottom of the MEB II board, beneath where the starter kit is plugged into the board. Refer to the followin figure for the exact location.
- Connect the PIC32MZ DA Starter Kit board to the MEB II board
- Power up the board by connecting the power adapter to J3 power connector on the MEB II board or a powered USB cable to the USB DEBUG port on the Starter Kit board



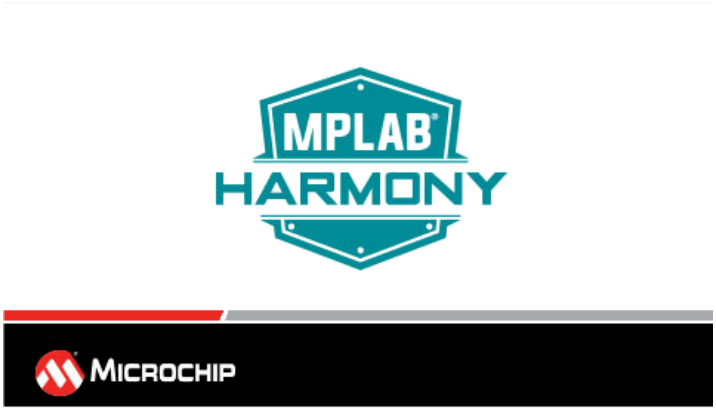
Running the Demonstration

This section details how to run the demonstration.

Description

Splash Screen

On start-up, the application will display a splash screen:



Main Screen

Subsequently, the demonstrations main mode will appear:



Without any touch interaction, the lamb sprite character remains stationary and loops in an idle animation. Pressing a finger on left or right half of the screen near one of the semi-transparent chevrons will trigger the lamb sprite to start moving towards that respective side of the screen. The longer the finger remain pressed, the “faster” the lamb will travel. Note the speed of the background increases as the lamb seemingly speeds up. The needle on the circular gauge will increase as the lamb travels. When the needle is in the yellow area, the lamb is in “super speed” mode. But if the needle touches the red section of the meter, the lamb becomes “exhausted”. It would fall and become dizzy before recovering. Touch interaction is disabled while the lamb takes a fall and is dizzy. A score is tallied based on how “far” the lamb has travelled before becoming exhausted. A high score is kept for as long as the application has power.

Parallax

The parallax feature is in full demonstration while the lamb is moving, as the speed of the clouds, the rolling hills and the pathway that the lamb is traveling on are all moving at different speed to create an illusion of movement and distance. The art of the background was chosen to be “loop-able”. The application keeps track of which part of the background scenery that is being shown and loops back to the start position seamlessly when the end is reached.

Animations

The animations used to support the lamb include:

- Idle Facing Right
- Idle Facing Left
- Walk/Run Facing Right
- Walk/Run Facing Left
- Super Speed Facing Right
- Super Speed Facing Left

- Fall Facing Right
- Fall Facing Left
- Dizzy Facing Right
- Dizzy Facing Left

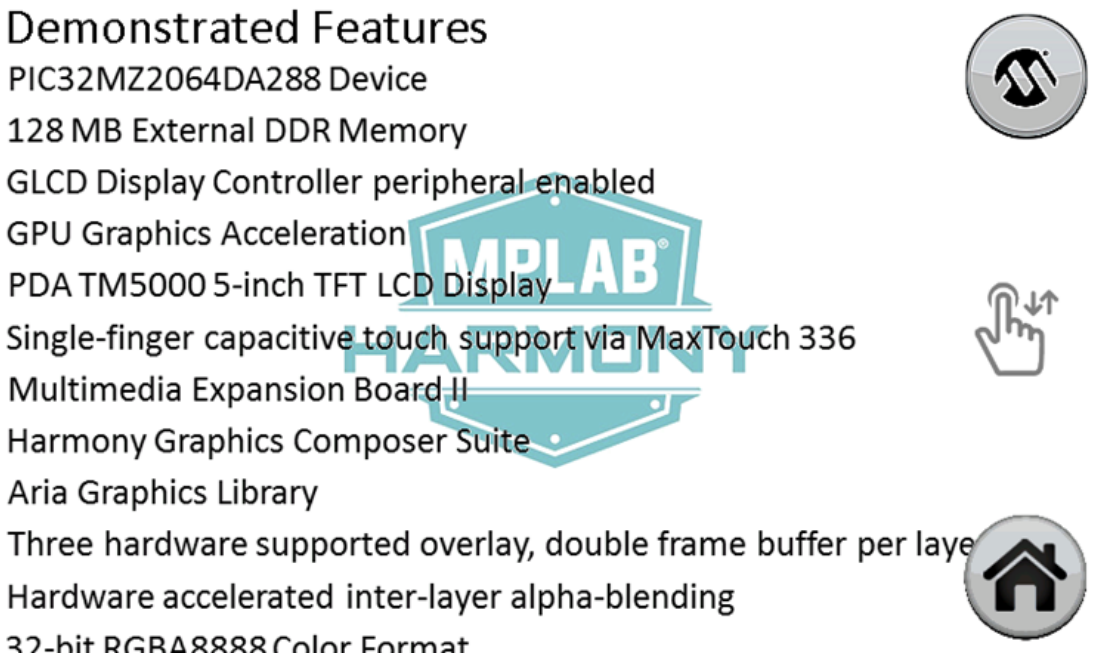
The application contains a decision-tree that picks the right set of animation to play out based on the facing direction and the “speed” of the lamb movement on-screen.

Information Screen

Tapping the Microchip logo on the upper corner of the main screen will switch the application to the information screen.

The information screen serves as a screen to list out the demonstrated features of the demo. It also provides two features.

- The features are located on layer1. Sliding your finger up or down will cause the feature list to scroll up and down as if it’s on an invisible glass pane. The MPLAB Harmony logo that is rendered on layer0 is another demonstration of the real-time alpha-blending capability of the GLCD.
- Touching the Microchip logo will trigger no visible change on-screen, but will toggle on/off a hidden “Easter Egg” mode. When the hidden mode is enabled, after going back to the main screen, the lamb character can go into “super speed” mode in-definitely without succumbing to exhaustion.



aria_basic_motion

This application demonstrates the use of double-buffered frame buffers in internal SRAM through the use of a global palette look-up table that reduces frame buffers to just one byte per pixel for the LCC driver. In addition, basic motion through the use of the Graphic Suite's ImageSequenceWidget is demonstrated.

Description

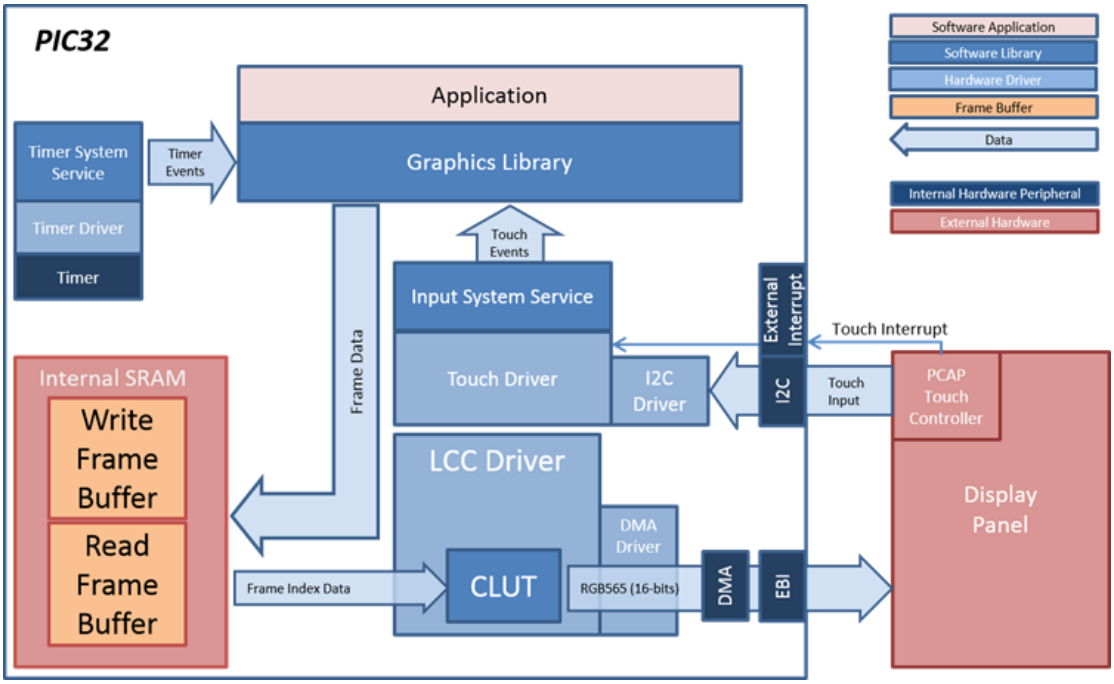
This application demonstrates basic motion through the use of Graphics Suite’s ImageSequenceWidget. The demonstration spins a set of 3D images and cycles through a set of battery level image presentations. The demonstration also shows the use of a 480x272x1 (8-bit) frame buffer, in a double buffered configuration, using a 256 custom generated color palette created from colors obtained from the User Interface (UI) designer-based graphic image assets. The demonstration illustrates that two buffers (261 KB total) of internal SRAM is sufficient to produce appealing, low-memory use applications.

The following key features of this demonstration application include:

- Using motion to show double buffering
- Double buffering in internal SRAM
- New tool that generates a global palette, so that this can be an easy process without apparent color loss in many graphic schemes

Architecture

The following figure shows the various software and hardware components used by this application for the pic32mz_ef_sk_meb2 target configuration.



The aria_basic_motion application uses the MPLAB Harmony Graphics Library to render graphics to the display. The Graphics Library draws the widgets and images as 8-bit indices to an 8-bit frame buffer, which is stored in the internal SRAM of the PIC32 device. Using the DMA, the Low-Cost Controllerless (LCC) Display Driver continuously transfers the frame data to the LCD display. During display refresh, the LCC driver performs a 16-bit color lookup using the indices stored in the refreshable framebuffer.

The final RGB565 color is derived from a custom palette Color Look-up Table (CLUT). The global palette is created for the entire application. It is created using the new palette generation tool which is capable of extracting colors from existing User Interface (UI) assets to establish a color array specific to the application with no wasted color positions.

The application is touch enabled. It features user touch input through the integrated touch screen on the display panel. Touch input from the touch controller goes through the I2C port, and the Input System Service acquires the touch input information from the Touch and I2C Drivers. The Input System Service sends touch events to the Graphics Library, which processes these events and updates the frame data accordingly.

In addition, the application uses the Timer System Service and core timer manage clock ticks for image motion control.

Demonstration Features

- 8-bit Custom Palette created using the new palette tool
- 2 Framebuffers (double-buffered UI)
- Graphic designer image assets
- Internal memory – low memory framebuffer footprint 261 KB
- LCC CLUT lookup driver – 8-bit index to 16-bit RGB565 color
- Graphics Image Sequence Widget

Tools Setup

- Enable Graphics Stack
- Enable Graphics Controller
  - Low-Cost Controllerless
  - Framebuffer Mode – Double-Buffer
  - Select Palette Mode
- Enable Global Palette
  - Use Harmony Graphics Composer tool
  - Auto-Calculate Global Palette after creating UI elements

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Aria Basic Motion demonstration.

Description

To build this project, you must open the `aria_basic_motion.X` project in MPLAB X IDE, and then select the desired configuration. The following tables list and describe the project and supported configurations. The parent folder for these files is

`<install-dir>/apps/gfx/aria_basic_motion`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`aria_basic_motion.X`	`<install-dir>/apps/gfx/aria_basic_motion/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mz_ef_sk_meb2	pic32mz_ef_sk+meb2	Demonstration for the PIC32MZ EF Starter Kit with the Multimedia Expansion Board II (MEB II).

Important! This application may contain custom code that is marked by the comments `// START OF CUSTOM CODE ...` and `// END OF CUSTOM CODE`. If you use the MPLAB Harmony Configurator to regenerate the application code, use the "Prompt Merge For All Differences" merging strategy and *do not* remove or replace the custom code.

Configuring the Hardware

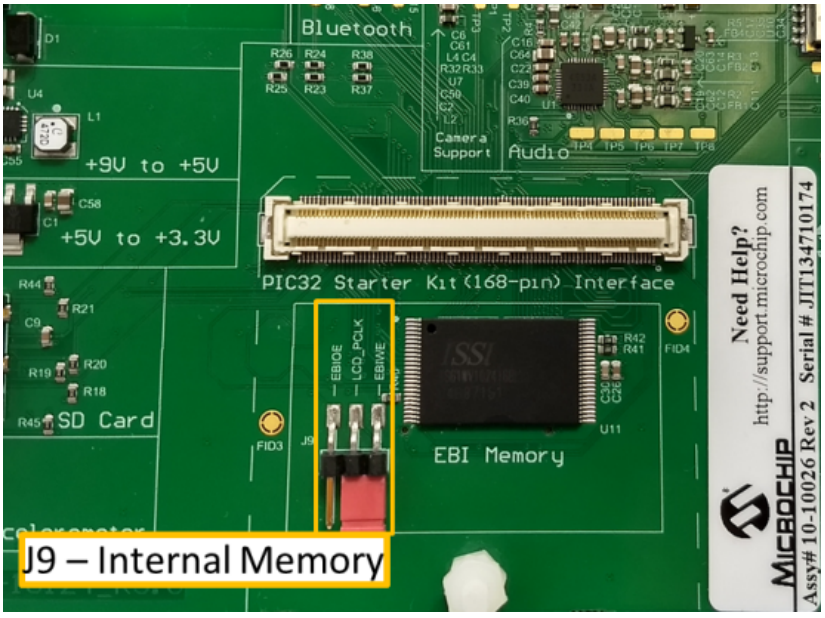
This section describes how to configure the supported hardware.

Description

PIC32MZ EF Starter Kit and MEB II

Configuration: pic32mz_ef_sk_meb2

- This configuration requires that the J9 jumper be set to enable internal SRAM for the frame buffer. Set the J9 jumper to connect the EBIOE and LCD_PCLK pins. The J9 jumper is located on the bottom of the MEB II board, beneath where the starter kit is plugged into the board. Refer to the following figure for the exact location.
- Connect the PIC32MZ EF Starter Kit board to the MEB II board
- Power up the board by connecting the power adapter to J3 power connector on the MEB II board or a powered USB cable to the USB DEBUG port on the Starter Kit board



Running the Demonstration

This section provides instructions on how to use the Aria Basic Motion demonstration.

Description

The aria_basic_motion demonstration has two UI screens. The main (default) screen is displayed upon boot-up. This screen has two user controls,

a battery indicator (simulated), application name, information button, and a 3-D Microchip logo. The second screen is the info_screen, which contains information text about the demonstration and an Exit button.

- Note that the battery indicator is “simulated” as an example of a multi-color image
- The motion is intended to show double buffering, so artifacts are minimized
- The battery will be discharged after approximately 4-5 revolutions of the image.
- The “recharge” is also simulated. The battery is just a user experience example (no recharging is actually done). The battery recharges in three seconds



Demonstration Symbols

Name	Icon
Power Button	Activates image spinning Turns green on toggle Indicates battery is empty
Battery	Battery is fully charged Battery is discharging Battery is not charged Battery is recharging
Information	Transitions to the info screen
Exit	Exit and return to the main screen

Touching the Power Button activates image spinning. When toggling, the power button turns green.

The battery image will transition from green to white if the demonstration continues to cycle for at least six seconds. If empty, the power button turns gray.

When the battery is empty, the recharge button is displayed, allowing the user to recharge the battery. When recharging the battery the recharge symbol is displayed,

Touching the info button allows the user to transition to the info screen. The info screen contains the Exit button, located at the top right of the screen, which exits the info screen and returns to the default (main) screen.

Standalone Demonstration Mode

As built, Demonstration mode is always active, which consists of two phases: Record or Playback.

Record Phase

The application starts in the Record Phase once it reaches the main screen after startup. Touch events are recorded until the event buffer is full or no touch events are seen and an idle period times out (Idle is typically 20-60 seconds). Having recorded a buffer of touch events the Playback Phase starts.

Playback Phase

Playback starts after the idle period times out. The application returns to the opening splash screen and the playback of recorded touch events starts when the application reaches the main screen. After executing the touch events recorded, the application will wait until a replay delay (typically 5-15 seconds) before restarting the replay.

If the user initiates a touch event during playback it will terminate and not start up again until the idle period times out again. After the idle time out, the application will return to the splash screen and event replay will start once the main screen is active.

Once the initial idle time out occurs it is not possible to record a new set of touch events except by restarting the application via a power up or master clear.

Disabling Standalone Demonstration Mode

Demonstration mode can be disabled through MHC by selecting *Harmony Framework Configuration > Graphics Stack > Use Graphics Stack? > Use Harmony Graphics Composer Suite? > Middleware > Use Aria User Interface Library> > Enable Demo Mode?*. Then, reconfigure the application using MHC, rebuild, and then reload.

aria_benchmark

This application presents frame update rate metrics on the various rendering operations in the Harmony graphics library.

Description

This application shows the frame update rates for various operations in the MPLAB Harmony Graphics Library, including string rendering, area fills and image decode and rendering. The benchmarks can be configured for different text sizes, number and size of discrete area fills and different image formats. The instantaneous and averaged frame update rates are shown at real-time on the demo. For microcontrollers with a 2D Graphics Processing Unit (GPU) (e.g., PIC32MZ DA), the GPU can be turned on and off at run-time.

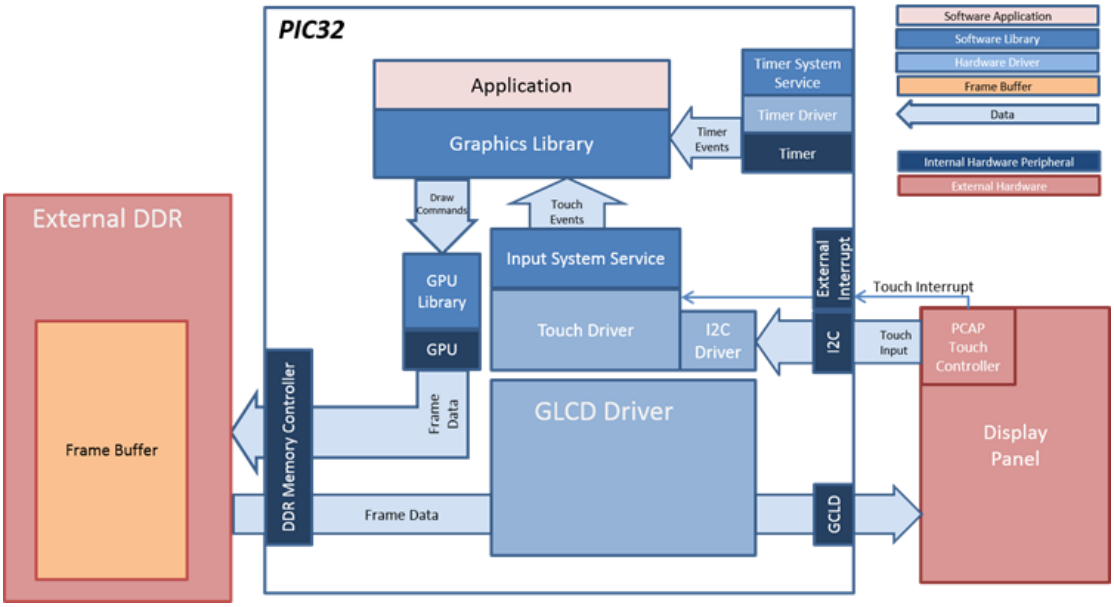
Architecture

The application continuously uses the graphics library to render text, fill areas, and draw images to the screen. Once a layer is completely rendered to, the graphics library increments a layer swap counter. The application periodically (at one second intervals) samples the layer swap counter, and calculates the difference from the previous sample. This difference is shown as the Frame Update Rate (Hz).

The following diagrams show the various software and hardware blocks used in this application:

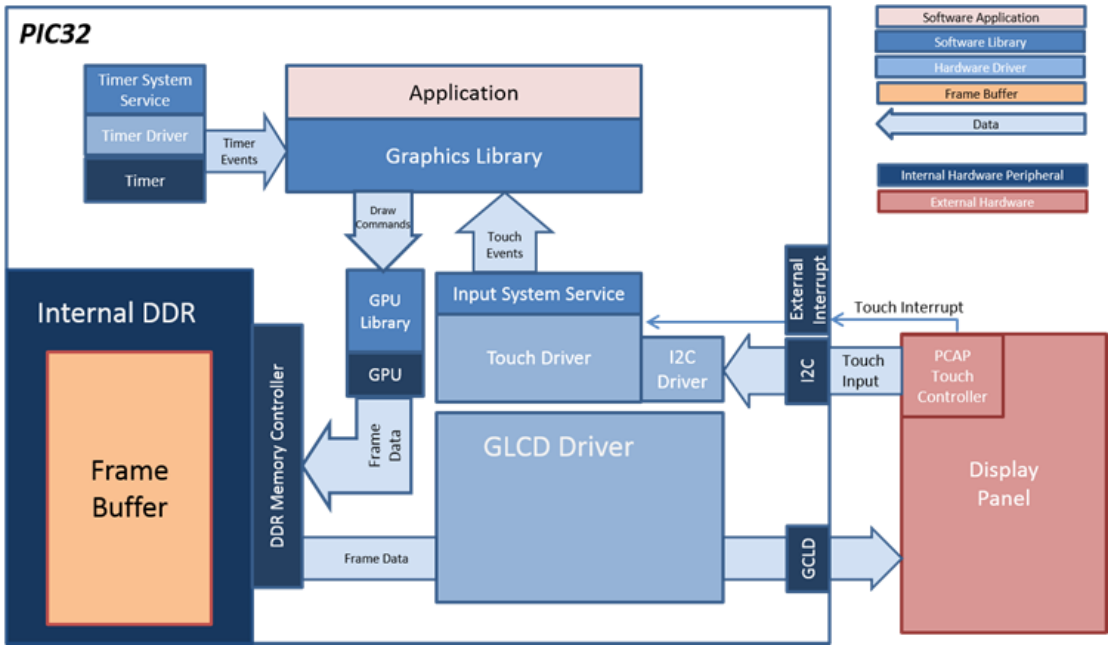
pic32mz_da_sk_extddr_meb2

In this configuration, the frame buffer is located in the external DDR and the GLCD display controller continuously fetches the frame buffer data and writes them to the display panel.



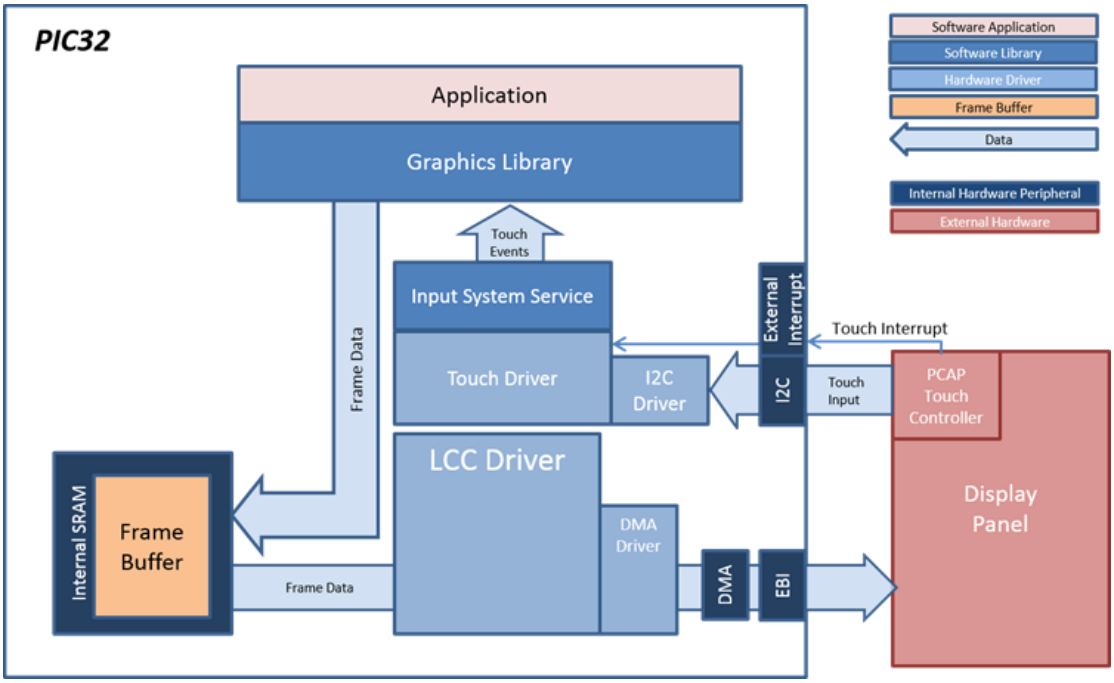
pic32mz_da_sk_intddr_meb2

In this configuration, the frame buffer is located in the internal DDR and the GLCD display controller continuously fetches the frame buffer data and writes them to the display panel.



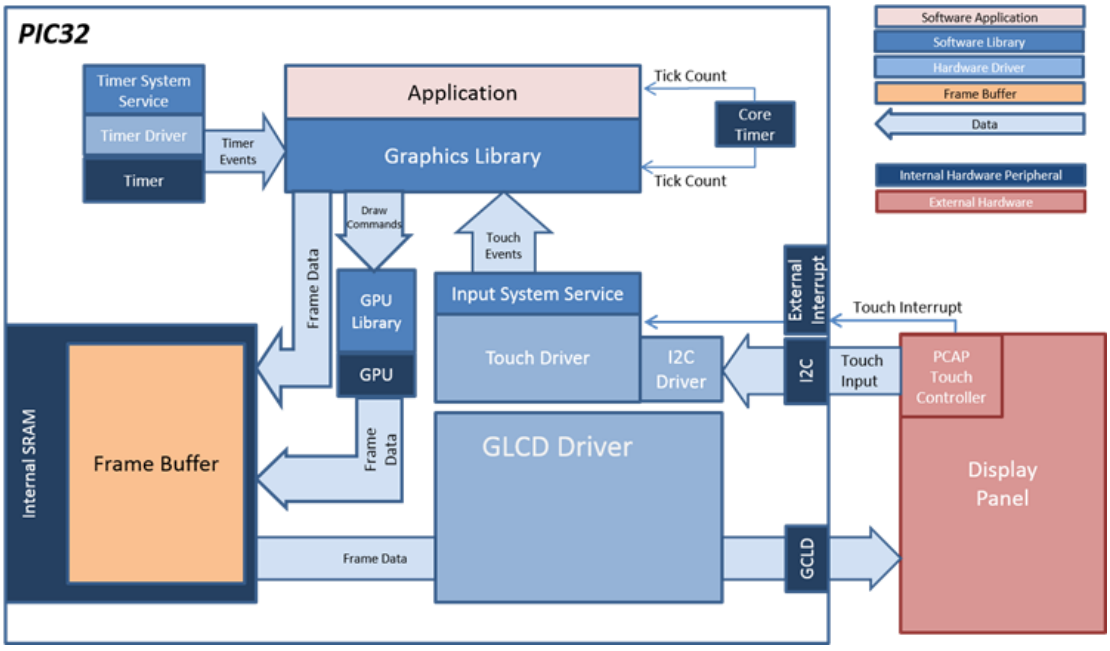
pic32mz_ef_sk_meb2

In this configuration, the frame buffer is located in the internal SRAM and the Low-Cost Controllerless (LCC) display driver uses the DMA to continuously fetch the frame buffer data and write them to the display panel.



pic32mz_da_sk_noddr_meb2

In this configuration, the frame buffer is located in the internal SRAM and the GLCD display controller continuously fetches the frame buffer data and writes them to the display panel.



Demonstration Features

- Timer hardware and Timer System Service
- Text rendering
- Rectangle widget motion and fills
- Image rendering of various formats (PNG, RAW RLE 16-bit, RAW 16-bit, JPEG 24-bit, RAW 32-bit predecoded in DDR)

Tools Setup Differences

- Enable “Timer” in “System Services” in MHC
- Set the Heap Size by selecting *Device & Project Configuration* > *Project Configuration* > *XC32(Global Options)* > *xc32-ld* > *General* and specifying 204800. This is needed to support decoding of large PNG files.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Aria Benchmark demonstration.

Description

To build this project, you must open the `aria_benchmark.X` project in MPLAB X IDE, and then select the desired configuration. The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/gfx/aria_benchmark`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`aria_benchmark.X`	`<install-dir>/apps/gfx/aria_benchmark/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32_mz_da_sk_extddr_meb2	pic32mz_da_sk_extddr+meb2	PIC32MZ Embedded Graphics with External DRAM (DA) Starter Kit, Multimedia Expansion Board II (MEB II) and Graphics Display Powertip 4.3" 480x272 Board.
pic32_mz_da_sk_intddr_meb2	pic32mz_da_sk_intddr+meb2	PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit, Multimedia Expansion Board II (MEB II) and Graphics Display Powertip 4.3" 480x272 Board.

pic32mz_ef_sk_meb2	pic32mz_ef_sk+meb2	Demonstration for the PIC32MZ EF Starter Kit with the Multimedia Expansion Board II (MEB II).
pic32mz_da_sk_noddr_meb2	pic32mz_da_sk_noddr+meb2	Demonstration for PIC32MZ Embedded Graphics with Disabled DRAM (DA) Starter Kit, Multimedia Expansion Board II (MEB II) and 4.3” WQVGA (480x272) Display.

Important! This application may contain custom code that is marked by the comments `// START OF CUSTOM CODE ...` and `// END OF CUSTOM CODE`. If you use the MPLAB Harmony Configurator to regenerate the application code, use the "Prompt Merge For All Differences" merging strategy and *do not* remove or replace the custom code.

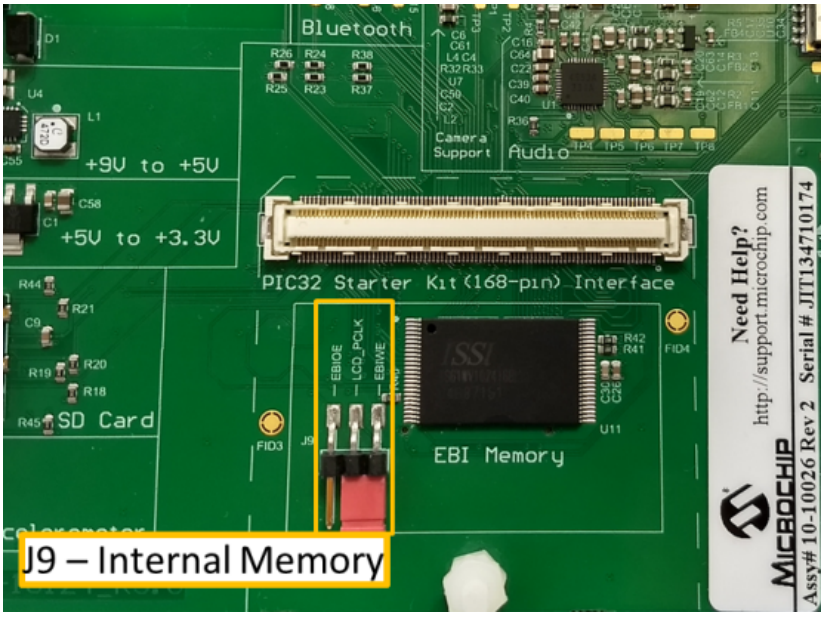
Configuring the Hardware

Description

PIC32MZ EF Starter Kit and MEB II

Configuration: pic32mz_ef_sk_meb2

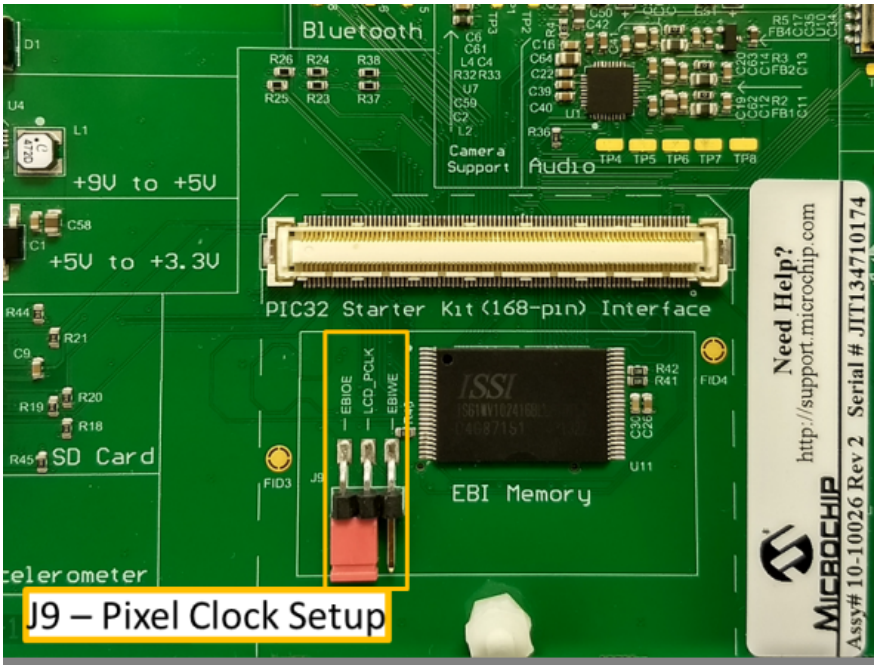
- This configuration requires that the J9 jumper be set to enable internal SRAM for the frame buffer. Set the J9 jumper to connect the EBIOE and LCD_PCLK pins. The J9 jumper is located on the bottom of the MEB II board, beneath where the starter kit is plugged into the board. Refer to the following figure for the exact location.
- Connect the PIC32MZ EF Starter Kit board to the MEB II board
- Power up the board by connecting the power adapter to J3 power connector on the MEB II board or a powered USB cable to the USB DEBUG port on the Starter Kit board



PIC32MZ Embedded Graphics with External DRAM (DA) Starter Kit , PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit, PIC32MZ Embedded Graphics with Disabled DRAM (DA) Starter Kit, and MEB II

Configurations: pic32mz_da_sk_extddr_meb2, pic32mz_da_sk_intddr_meb2, pic32mz_da_sk_noddr_meb2

- On the MEB II, the EBIOE and LCD_PCLK (J9) must be jumpered. A connection establishes the GLCD's pixel clock output timing. The external SRAM memory on the board is disabled. The jumper (J9) is available on the bottom side of the MEB II board under the starter kit. The J9 jumper is located on the bottom of the MEB II board, beneath where the starter kit is plugged into the board. Refer to the following image for the exact location.
- Connect the PIC32MZ DA Starter Kit board to the MEB II board
- Power up the board by connecting the power adapter to J3 power connector on the MEB II board or a powered USB cable to the USB DEBUG port on the Starter Kit board



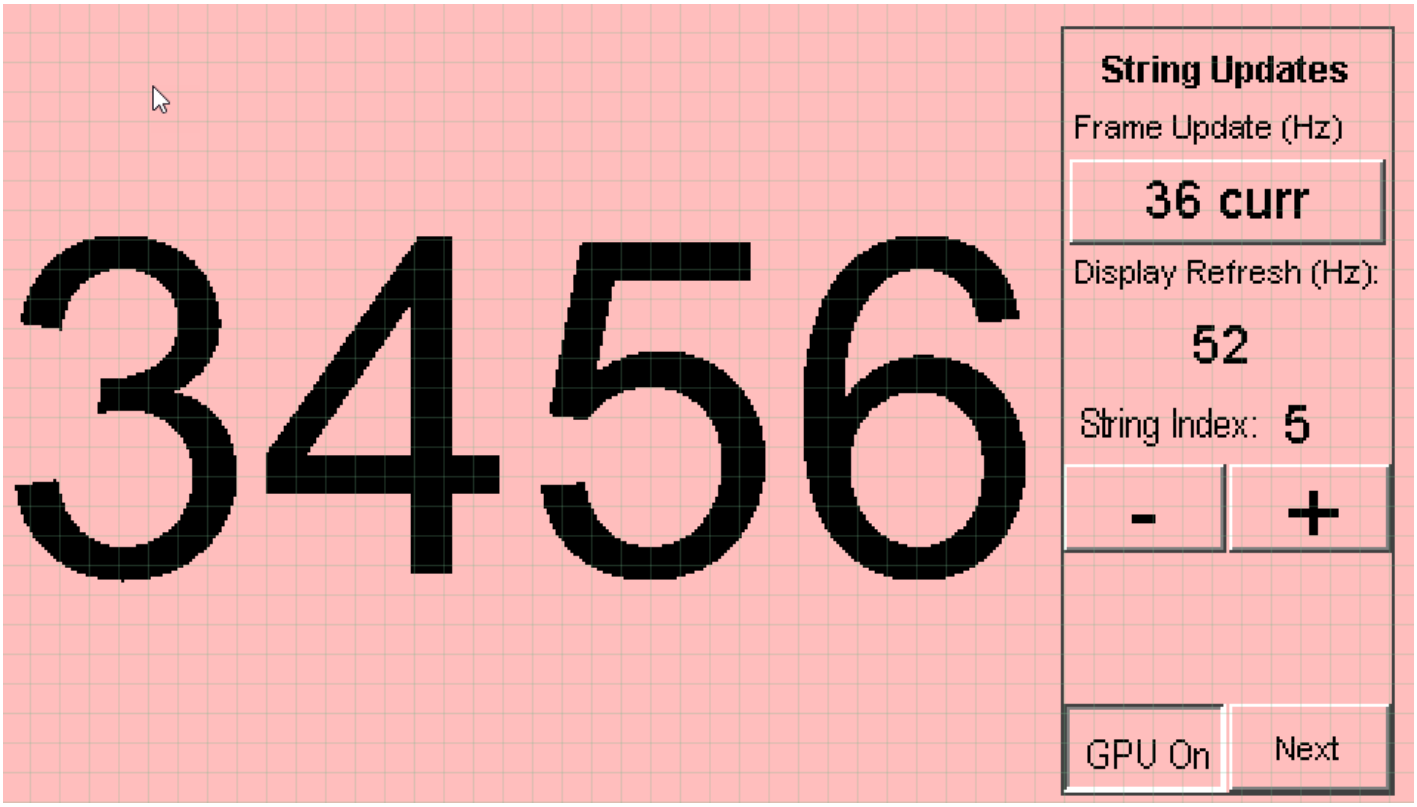
Running the Demonstration

This section provides information on how to run and use the application.

Description

On start-up, the application will display a splash screen.

After the splash-screen completes, the String Update benchmark screen is shown. In this screen, a counter is incremented at every application cycle. The screen demonstrates the rate at which the graphics library renders a string on the screen. This involves a fill operation that clears the background, lookup of the glyphs from the string library, and the drawing of the glyphs on the frame buffer.

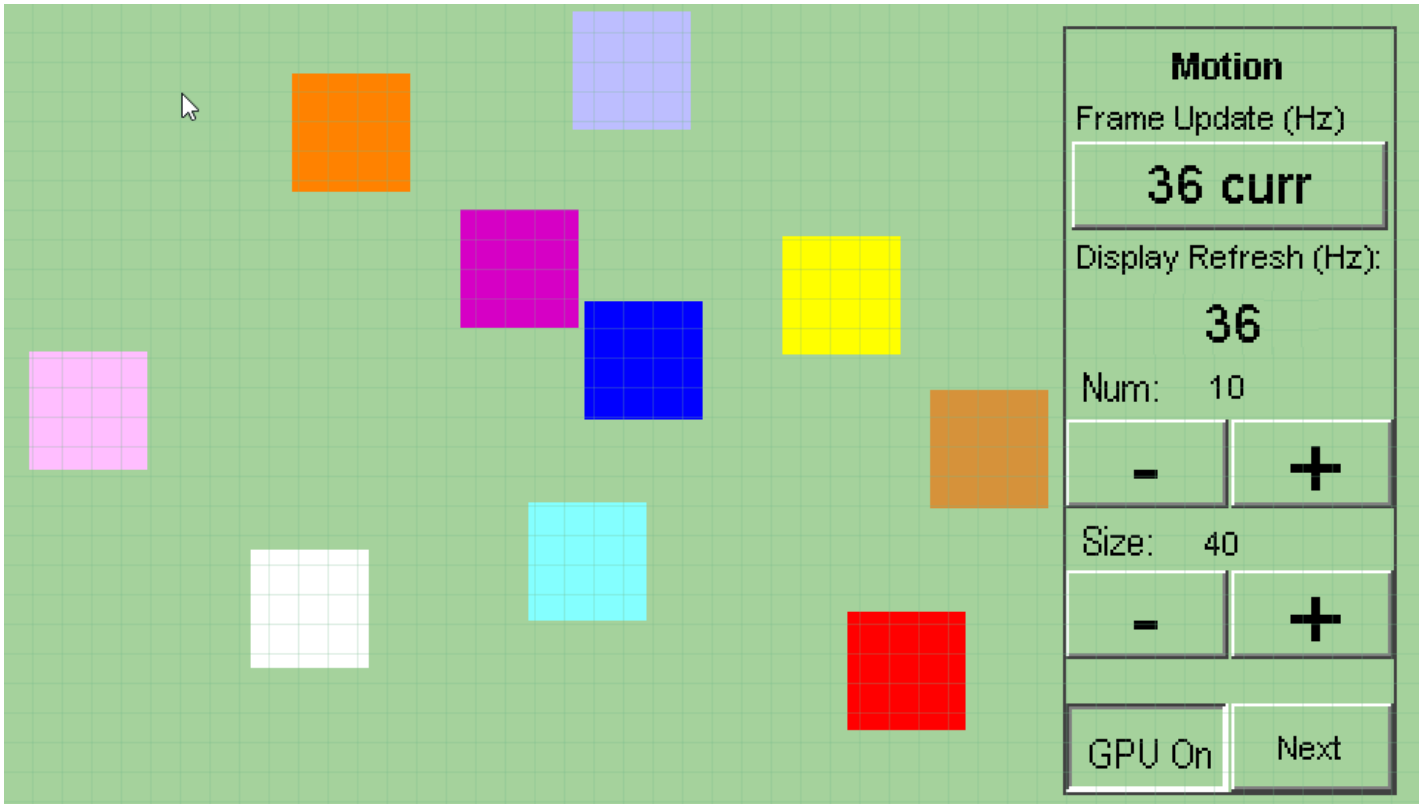


The “Frame Update (Hz)” field shows the current or instantaneous rate at which the graphics library updates the label widget that shows the counter value. Tapping the Frame Update value switches between the current value (curr) and the average (avg) value across 10 samples.

Tapping the “+” and “-“ buttons increases and decreases the size of the string, respectively.

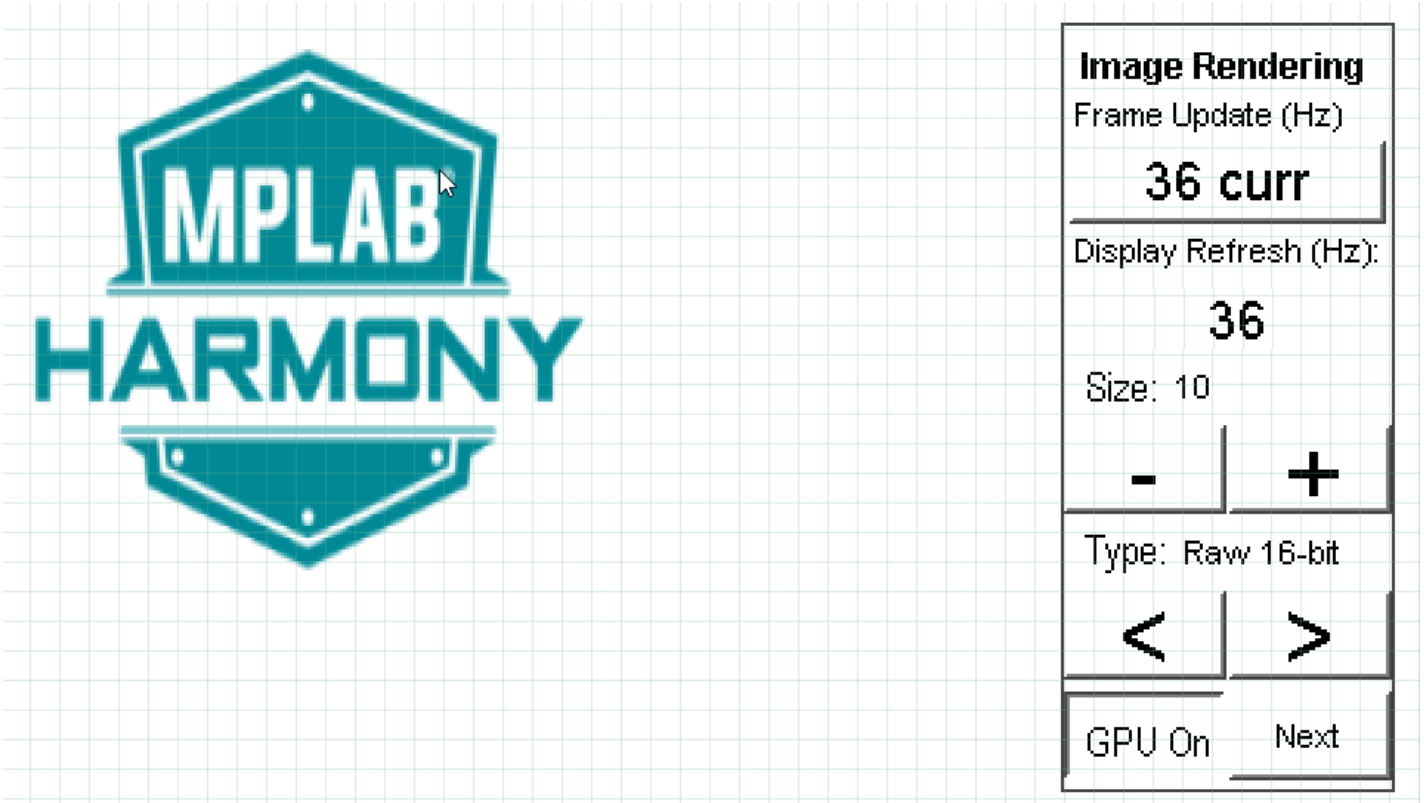
On the PIC32MZ DA configurations, toggling the “GPU On” / “GPUOff” button switches the GPU on and off.

Tapping the “Next” button switches to the Motion and Fill benchmark screen. In this screen, squares are shown moving across the screen. The Frame Update value is the rate at which the graphics library is able to render all the squares on the screen at their new positions. This involves a fill operation of the background color at the old location of the squares and a fill of the squares’ colors at the new position.



The number and size of the squares can be increased and decreased using the “+“ and “-“ buttons. If the maximum or minimum size is reached, touching “+” or “-“, respectively, will switch to a full screen fill of alternating colors.

Tapping the “Next” button transitions to the Image Decode and Rendering screen. In this screen, two images of the same size are alternately rendered between application cycles. This involves a fill of the background color, decode and conversion of the image to the frame buffer format, and the drawing of the image to the frame buffer. The Frame Update value is the rate at which the graphics library is able to render an image on the screen.



The size of the images can be increased and decreased using the “+“ and “-“ buttons.

Tapping the “<” and “>” buttons switches between the various image formats. The formats that are supported are PNG, RAW RLE 16-bit, RAW 16-bit and JPEG 24-bit. RAW 32-bit predecoded in DDR is also supported in the PIC32MZ DA configurations with DDR memory. Due to heap limitations, the maximum size of PNG images is 100x100 pixels.

aria_coffee_maker

This demonstration provides a practical multi-layered application using the Aria User Interface Library.

Description

This application showcases an example of how a graphics controller with multiple graphical layer support can be achieved using the Aria User Interface Library. The demonstration also highlights how tightly integrated the MPLAB Harmony Graphics Composer Suite can be in supporting a rich feature set in an application.

Various Aria Graphics Library features such as turning on/off the GPU at run-time, adjusting the GLCD’s per-layer alpha-blending, cycling through multi-language localization support is embedded within the demo disguised as a user-interface for a coffee maker appliance.

The demonstration launches with a splash screen highlighting basic motion capability supported by Aria Graphics Library. When running the application, the user can interface with it via capacitive single-fingered touch and swiping gestures.

Images used with the demonstration are preprocessed in the DDR at start. The purpose of the preprocessing is to highlight the GPU's block transfer capability. The text in the information screen is also handled as a preprocessed image to leverage this same capability.

As a showcase application, it also features ‘demo mode’, where it autonomously runs the demonstrations after a specified period of idle time.

Architecture

The following diagrams show the various software and hardware components for each MPLAB X IDE project configuration.

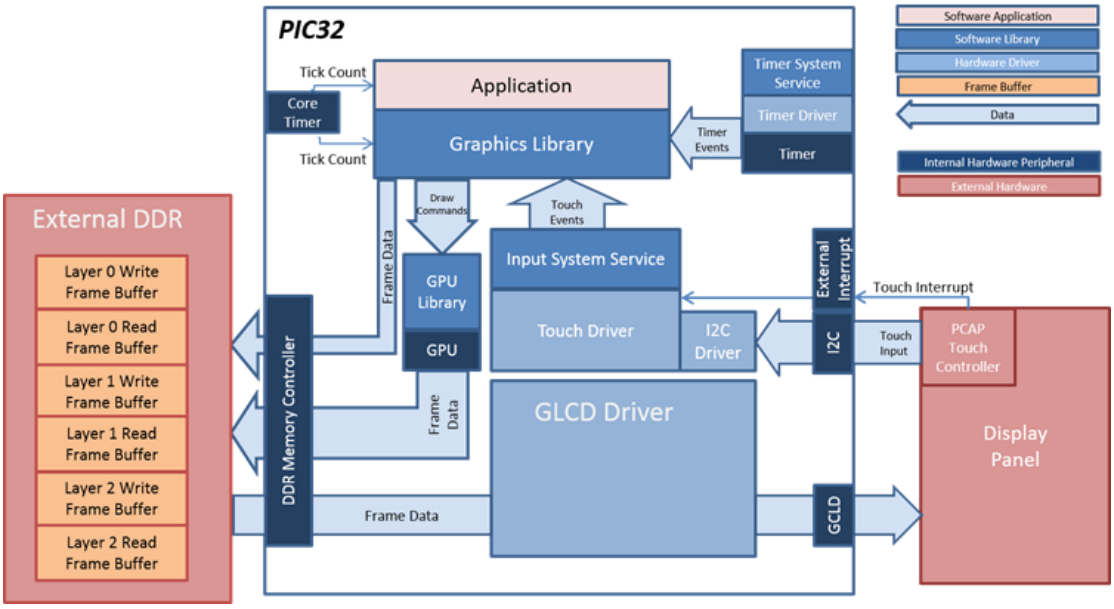
pic32mz_da_sk_extddr_meb2, and pic32mz_da_sk_extddr_meb2_wvga

For these configurations, the application uses the Graphics library to render graphics to the display. The Graphics library passes draw commands into the GPU Library, which then draws the widgets and images to the three individual write frame buffers (one for each layer) that are stored on the external DDR2 Memory. The Graphics library Draw commands going into the GPU Library can be dynamically enabled/disabled at run-time and redirected to the write frame buffer directly. The intent is to contrast draw performance of one versus the other. Via the DDR2 Memory Controller, the GLCD display controller peripheral continuously transfers frame data from the all three read buffers onto to the LCD display. The write and read frame buffer pairs are swapped independently as required when the Graphics Library is done rendering to the write frame buffer and GLCD driver signals the GLCD peripheral to change its read location.

The application also features user touch input through the integrated touch screen on the display panel. Touch input from the touch controller goes thru the I2C port, and the Input System Service acquires the touch input information from the touch and I2C drivers. The Input System Service sends touch events to the Graphics library which processes these events and updates the frame data accordingly.

The application also uses the Timer System Service and driver to send timer events for Demo Mode.

The core timer is used by the application and the Graphics library to manage the movement of the splash screen and the momentum decay of the list wheel tumbler.



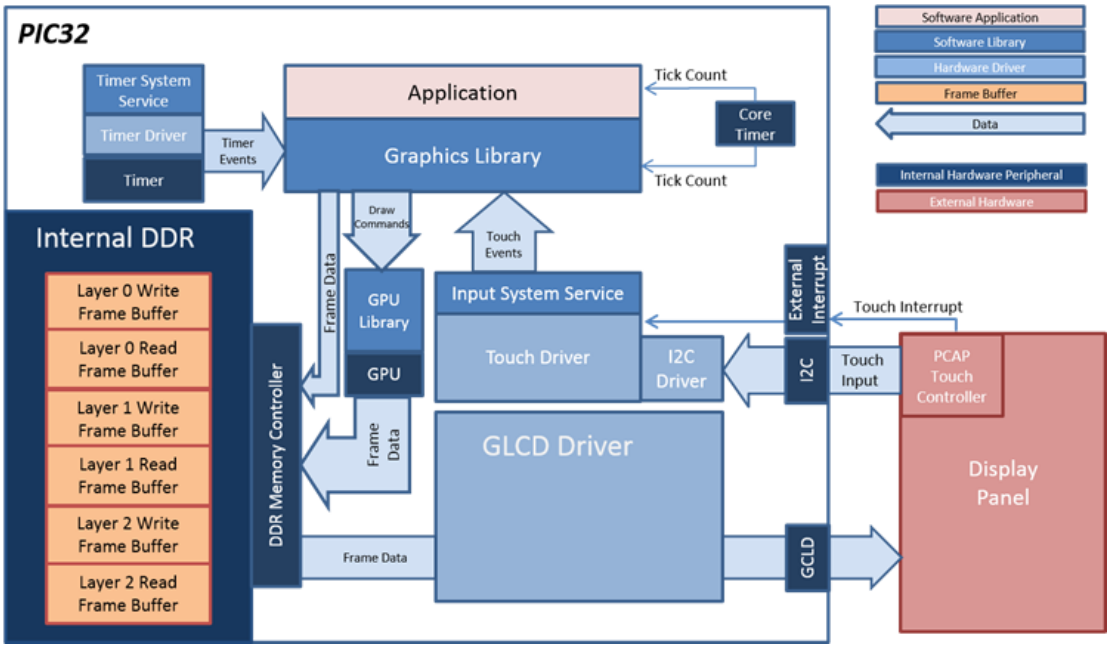
pic32mz_da_sk_intddr_meb2, and pic32mz_da_sk_intddr_meb2_wvga

For these configurations, the application uses the Graphics library to render graphics to the display. The Graphics library passes draw commands into the GPU Library, which in-turn draws the widgets and images to the three individual write frame buffers (one for each layer) that are stored on the internal DDR2 memory. The Graphics Library Draw commands going into the GPU Library can be dynamically enabled or disabled at run-time and redirected to the write frame buffer directly. The intent is to contrast draw performance of one or the other. Via the DDR2 Memory Controller, the GLCD display controller peripheral continuously transfers frame data from the all three read buffers onto to the LCD display. The write and read frame buffer pairs are swapped independently as required when the Graphics Library is done rendering to the write frame buffer and GLCD driver signals the GLCD peripheral to change its read location.

The application also features user touch input through the integrated touch screen on the display panel. Touch input from the touch controller goes thru the I2C port, and the Input System Service acquires the touch input information from the touch and I2C drivers. The Input System Service sends touch events to the Graphics library which processes these events and updates the frame data accordingly.

The application also uses the Timer System Service and driver to send timer events for Demo Mode.

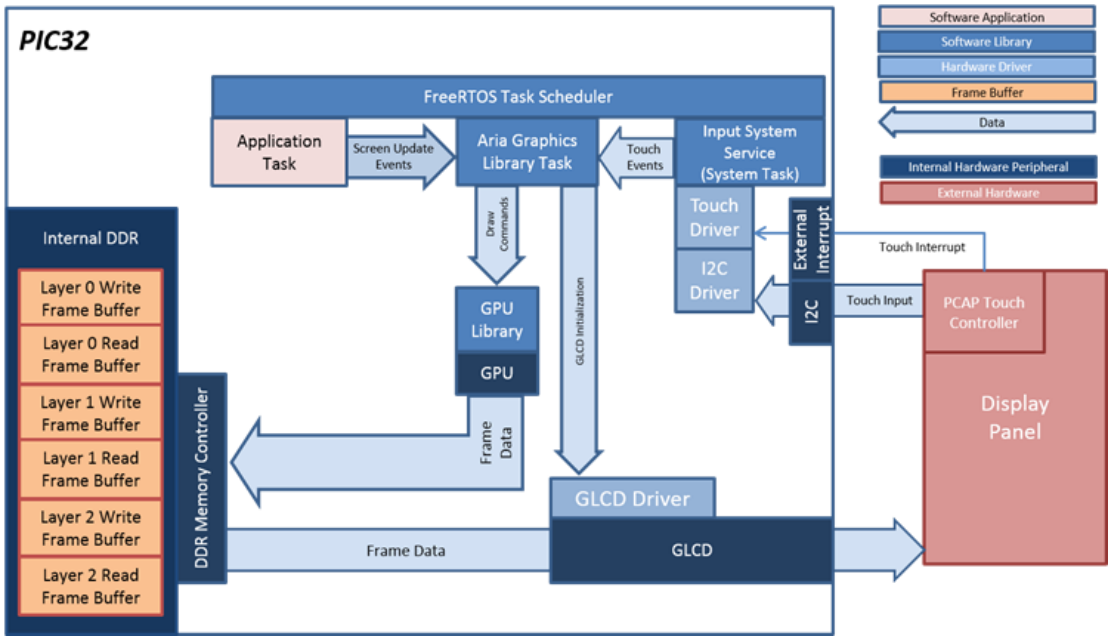
The core timer is used by the application and the Graphics library to manage the movement of the splash screen and the momentum decay of the list wheel tumbler.



pic32mz_da_sk_intddr_meb2_freertos

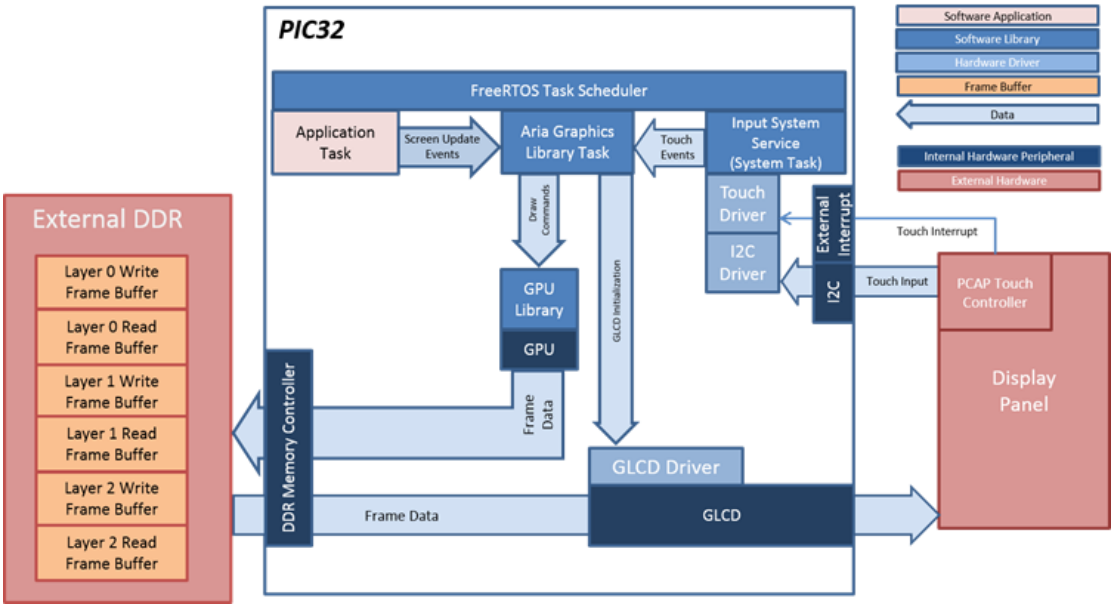
This configuration runs in a run-time operating system (FreeRTOS) environment. The tasks are managed and scheduled for execution by the FreeRTOS Task Scheduler. The Aria User Interface Library runs as a high priority, blocking task that waits for events from other tasks like the

Application or the Input System Service. When an event is received, say a touch event from the Input System Service, the Aria User Interface Library unblocks and processes this event. If an event requires that the screen be updated, the graphics library uses the GPU to draw to the layer frame buffers in internal DDR. The contents of the frame buffers are continuously delivered to the display panel using the Graphics LCD (GLCD) controller.



pic32mz_da_sk_extddr_meb2_freertos

This configuration runs in a run-time operating (FreeRTOS) environment, the pic32mz_da_sk_intddr_meb2_freertos configuration. The main difference in this configuration is the frame buffers are stored in external DDR.



Demonstration Features

- Integrated PCAP Touch Input
- Three graphics layer supported via the GLCD peripheral on the PIC32MZ DA device
- GPU peripheral supported, can be enabled/disabled at run-time
- Support of RTOS by the entire Graphics Stack
- 32-bit RGBA8888 color depth support (16.7 million unique colors) (see Options)
- Per-layer runtime adjustable global alpha-blending
- Per-layer frame double-buffering

- Language localization in English, French, German, and Italian (see Widget Tool Box Panel)
- Image compression techniques using Run-Length Encoding and JPEG (see Enabling Demo Mode in a MPLAB Harmony Graphics Application)
- Run-time JPEG decoding
- UTF-16 character font support (see Font Assets)
- Software-based single-touched gestured movement detection
- Run-time graphic widget motion
- Button and List Wheel Widgets
- Trigger external event via button press (see Event Manager)
- Time-delayed self-demonstration mode (see Enabling Demo Mode in a MPLAB Harmony Graphics Application)

Tools Setup Differences

- “Use Graphics Stack” is selected in MHC. This enables the graphics and touch libraries and drivers for the BSP.
- In the Clock Diagram, the Memory PLL is configured to output at 220 MHz (MPLLIDV = 1, MPLLMULT = 55, MPLLODIV1 = 6, MPLLODIV2 = 1)
- Graphics Processor NANO 2D has been enabled
- Demo Mode has been enabled, records up to 1000 input events, the idle timeout is set to 60 seconds

There are custom code blocks identified by `// CUSTOM CODE – DO NOT DELETE … // END OF CUSTOM CODE` comments that represent modifications to the code outside of MHC. These code blocks are necessary for correct operation of the demonstration under these configurations.

Important! When regenerating the project from within MHC use the "Prompt Merge For All Differences" merging strategy and do not change these custom code blocks.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Aria Coffee Maker demonstration.

Description

To build this project, you must open the `aria_coffee_maker.X` project in MPLAB X IDE, and then select the desired configuration. The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/gfx/aria_coffee_maker`.

There are custom code blocks identified by `// CUSTOM CODE – DO NOT DELETE … // END OF CUSTOM CODE` comments that represent modifications to the code outside of MHC. These code blocks are necessary for correct operation of the demonstration under the following configurations.

Important! When regenerating the project from within MHC do not change these custom code blocks.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`aria_coffee_maker.X`	`<install-dir>/apps/gfx/aria_coffee_maker/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mz_da_sk_extddr_meb2	pic32mz_da_sk_extddr+meb2	Demonstration for the Multimedia Expansion Board II (MEB II) connected to the PIC32MZ Embedded Graphics with External DRAM (DA) Starter Kit.
pic32mz_da_sk_extddr_meb2_wvga	pic32mz_da_sk_extddr+meb2+wvga	Demonstration for the PIC32MZ Embedded Graphics with External DRAM (DA) Starter Kit Memory plus the Multimedia Expansion Board II (MEB II) with a High-Performance (5") WVGA Display Module with maXTouch.

pic32mz_da_sk_intddr_meb2	pic32mz_da_sk_intddr+meb2	Demonstration for the Multimedia Expansion Board II (MEB II) connected to the PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit.
pic32mz_da_sk_intddr_meb2_wvga	pic32mz_da_sk_intddr+meb2+wvga	Demonstration for the Multimedia Expansion Board II (MEB II) connected to the PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit and High-Performance 5" WVGA Display Module with maXTouch.
pic32mz_da_sk_extddr_meb2_freertos	pic32mz_da_sk_extddr+meb2	Demonstration for the Multimedia Expansion Board II (MEB II) connected to the PIC32MZ Embedded Graphics with External DRAM (DA) Starter Kit using FreeRTOS.
pic32mz_da_sk_intddr_meb2_freertos	pic32mz_da_sk_intddr+meb2	Demonstration for the Multimedia Expansion Board II (MEB II) connected to the PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit using FreeRTOS.

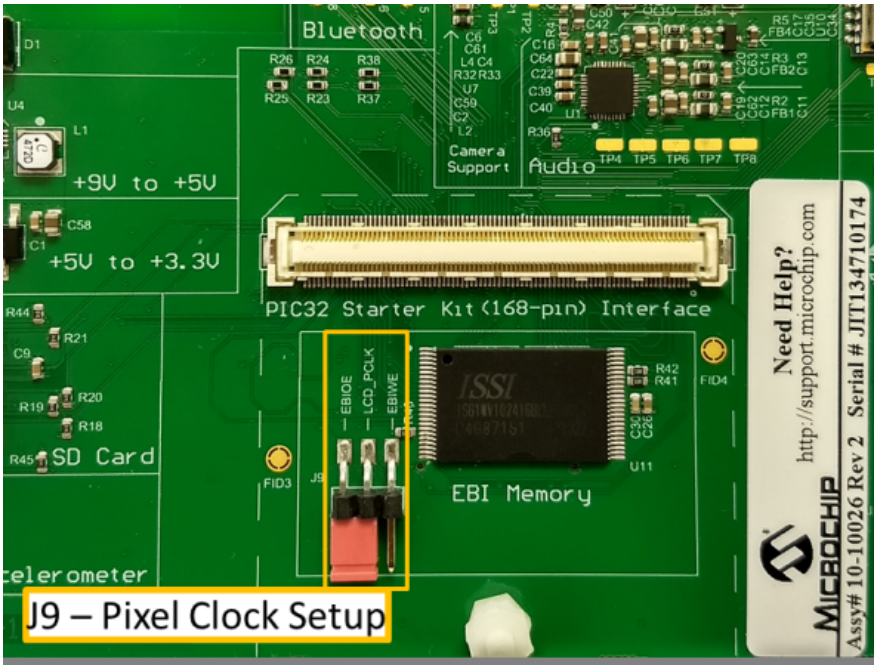
Configuring the Hardware

This section describes how to configure the supported hardware.

Description

Configurations: pic32mz_da_sk_extddr_meb2, pic32mz_da_sk_extddr_meb2_freertos, pic32mz_da_sk_extddr_meb2_wvga, pic32mz_da_sk_intddr_meb2,pic32mz_da_sk_intddr_meb2_freertos, and pic32mz_da_sk_intddr_meb2_wvga

- On the MEB II, the EBIOE and LCD_PCLK (J9) must be jumpered. A connection establishes the GLCD's pixel clock output timing. The external SRAM memory on the board is disabled. The jumper (J9) is available on the bottom side of the MEB II board under the starter kit. The J9 jumper is located on the bottom of the MEB II board, beneath where the starter kit is plugged into the board. Refer to the following image for the exact location.
- Connect the PIC32MZ DA Starter Kit board to the MEB II board
- Power up the board by connecting the power adapter to J3 power connector on the MEB II board or a powered USB cable to the USB DEBUG port on the Starter Kit board



Running the Demonstration

This section provides a brief description of what to expect once the demonstration is running.

Description

On start-up, the application will display a splash screen:

Subsequently, the demonstration's main screen will appear. The following image shows the main screen for the pic32mz_da_sk_extddr_meb2,and pic32mz_da_sk_intddr_meb2, configurations.



As indicated by the hand icon (), the gray areas on each side are trays that can slide horizontally.

To reveal or hide the trays from the screen area, the application can recognize two different behaviors when pressing your finger near the hand icon:

1. Moving your finger slowly in a horizontal direction while maintaining contact with the touch screen and the tray you are in contact with will perform a pixel-by-pixel movement until the tray reaches its minimum/maximum constraint of movement.
2. Swiping your finger in a quicker horizontal motion will trigger the application to automatically move the tray in the same direction as your finger swipe.

Left Tray

The left tray has six buttons. Touching each demonstrates a different feature.

The flag button triggers the Aria User Interface Library multi-lingual localization feature to cycle through English, German, Italian and French. The image on the button changes to American, German, Italian and French flags to indicate the current language being displayed. Note that it is not just the labels in the current screen that has their text changed. The label widgets within the Info Screen also have their text changed to reflect the current language selected.

The coffee bean button simulates changing the coffee maker appliance to use coffee beans from different roast level (Dark, Medium and Light). Each time the button is touched and released, the coffee bean image in the background changes along with the label widget indicating which roast level the application is at currently. There is one image for dark, two for medium and one for light. The image and the label widget are actually rendered on layer 0 in the application. Changing them independently while trays and list wheels are moving in the foreground serves as a way to demonstrate the hardware accelerated per-layer blending capability supported by the GLCD.

The tea cup button cycles the two trays (each on its own layer) through three different alpha-blend settings. This button demonstrates the capability for the GLCD peripheral to manage the layer-wide "global" alpha-blend value.

The coffee cup button simulates adjusting the temperature of the water for the coffee maker appliance. There are three temperature levels as indicated by the icon on the button. As the temperature level changes, the images on the list wheel on the right tray change accordingly. Touching

this button also triggers the right tray to automatically slide out if it is not completely exposed.

The GPU button turns the GPU support to the Graphics Library ON or OFF at real-time. When the button shows “GPU ON”, the Graphics Library is using the Nano2D library to drive the GPU to draw to the frame buffer. When it is showing “GPU OFF”, the Graphics Library is relying solely on its own software-based drawing algorithms to compose the frame buffer.

The “?” button leads the application to switch to the Information Screen, where the features demonstrated by this application is listed in a vertically touch draggable text plane.

Right Tray

The right tray has a list wheel and a button.

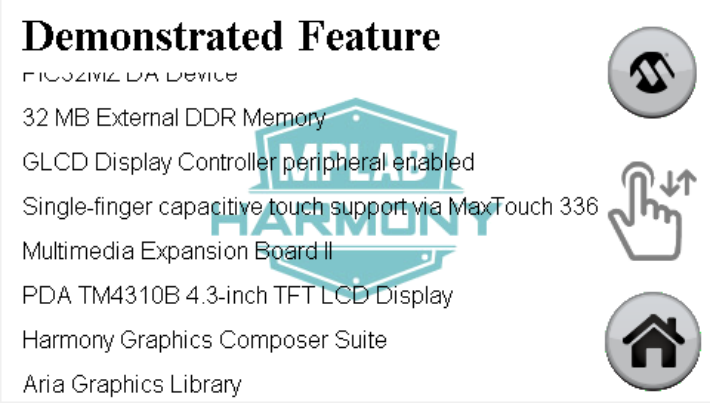
The list wheel simulates the brew size selection for a coffee maker. There are four different sizes. Depending on which one is selected, the button below the list wheel updates to the current brew size selected. As mentioned before, the images on the list wheel will change depending on the temperature selected by the button on the left tray.

Touching the brew button below the list wheel activates an LED on the MEB II to simulate triggering a motor in the appliance to begin the brewing process. Depending on which brew size is selected, a different LED is enabled.

Information Screen

The information screen serves as a screen to list out the demonstrated features of the demo. It also provides two features.

- The features are located on layer1. Sliding your finger up or down will cause the feature list to scroll up and down as if it’s on an invisible glass pane. The MPLAB Harmony logo that is rendered on layer0 is another demonstration of the real-time alpha-blending capability of the GLCD.
- Touching the Microchip logo will trigger the application to the splash screen. Once the splash screen animation has complete, the application will return to the information screen.



Autonomous Demonstration Mode

As built, Demonstration Mode is always active. It consists of two phases: Record or Playback.

Record Phase:

The application starts in the Record Phase once it reaches the main screen after startup. Touch events are recorded until the event buffer is full or no touch events are seen and an idle period times out. (Idle is typically 20-60 seconds). Having recorded a buffer of touch events the Playback Phase starts.

Playback Phase:

Playback starts after the idle period times out. The application returns to the opening splash screen and the playback of recorded touch events starts when the application reaches the main screen. After executing the touch events recorded, the application will wait until a replay delay (typically 5-15 seconds) before starting the replay over.

If the user initiates a touch event during playback it will terminate and not start up again until the idle period times out again. After the idle time out, the application will return to the splash screen and event replay will start once the main screen is active.

Once the initial idle time out occurs it is not possible to record a new set of touch events except by restarting the application via a power up or master clear.

Disabling Demo Mode:

Demo mode can be disabled through MHC by selecting *Harmony Framework Configuration > Graphics Stack > Use Graphics Stack? > Use Harmony Graphics Composer Suite? > Middleware > Use Aria User Interface Library>* ***Enable Demo Mode?***. Then, reconfigure the application using MHC, rebuild, and then reload.

aria_counter

This application demonstrates the use of double buffering to show multiple counters running at high speed on a fixed (high-spatial frequency) image background.

Description

This application demonstrates the use of double-buffering to show multiple counters running at high speed on a fixed, high-spatial frequency, image background. Double-buffering helps eliminate tearing and artifacts in graphics applications. The application can also switch between double- and single-buffering at run-time to show the advantages of double-buffering.

This application also provides a reference on how to update the text in a label widget at run-time.

Architecture

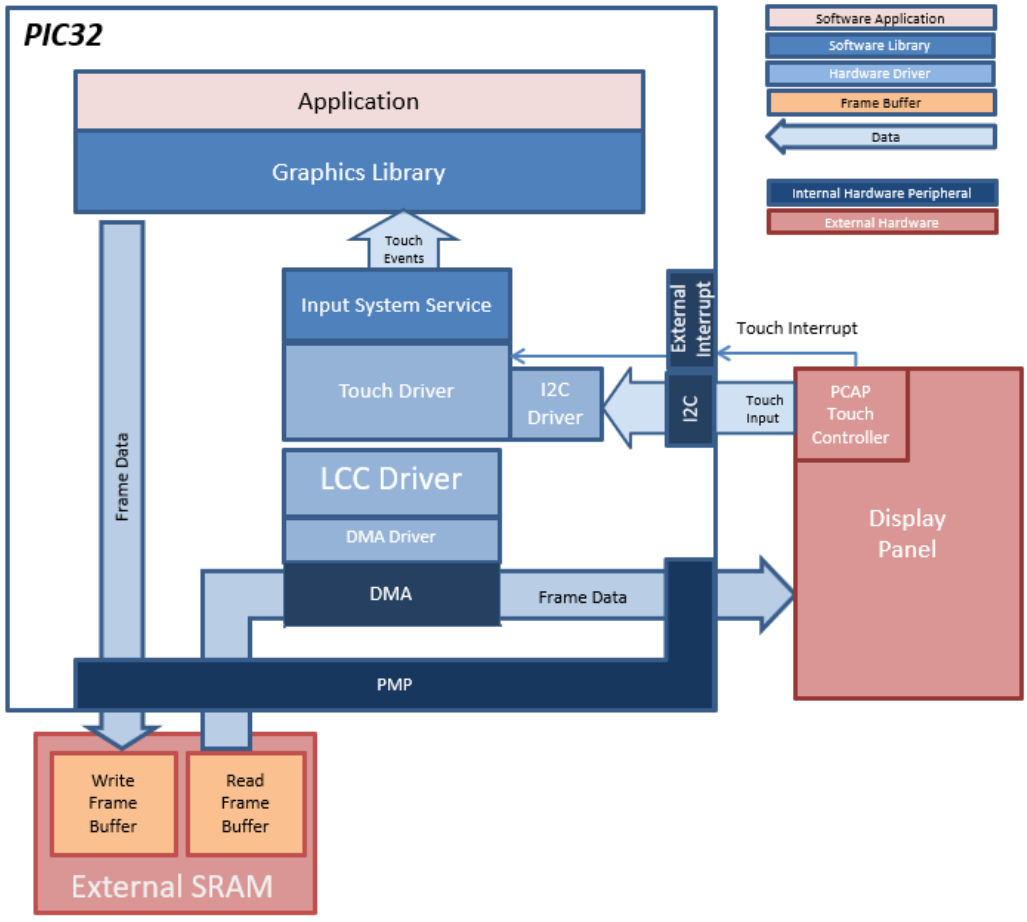
The aria_counter application uses the MPLAB Harmony Graphics Library to render graphics to the display. The application uses double buffering to eliminate tearing, and this requires two frame buffers; a read buffer and a write buffer. The Graphics library draws the widgets and images to the write frame buffer, while the contents of the read frame buffer are being delivered to the display panel. The write and read frame buffers are swapped when the Graphics library is done rendering to the write frame buffer, and the contents of the read frame buffer have been sent to the display panel.

The application also features user touch input through the integrated touch screen on the display panel. Touch input from the touch controller goes through the I2C port, and the Input System Service acquires the touch input information from the Touch and I2C Drivers. The Input System Service sends touch events to the Graphics library, which processes these events and updates the frame data accordingly.

The following figures display the various software and hardware components used by the different configurations in this application.

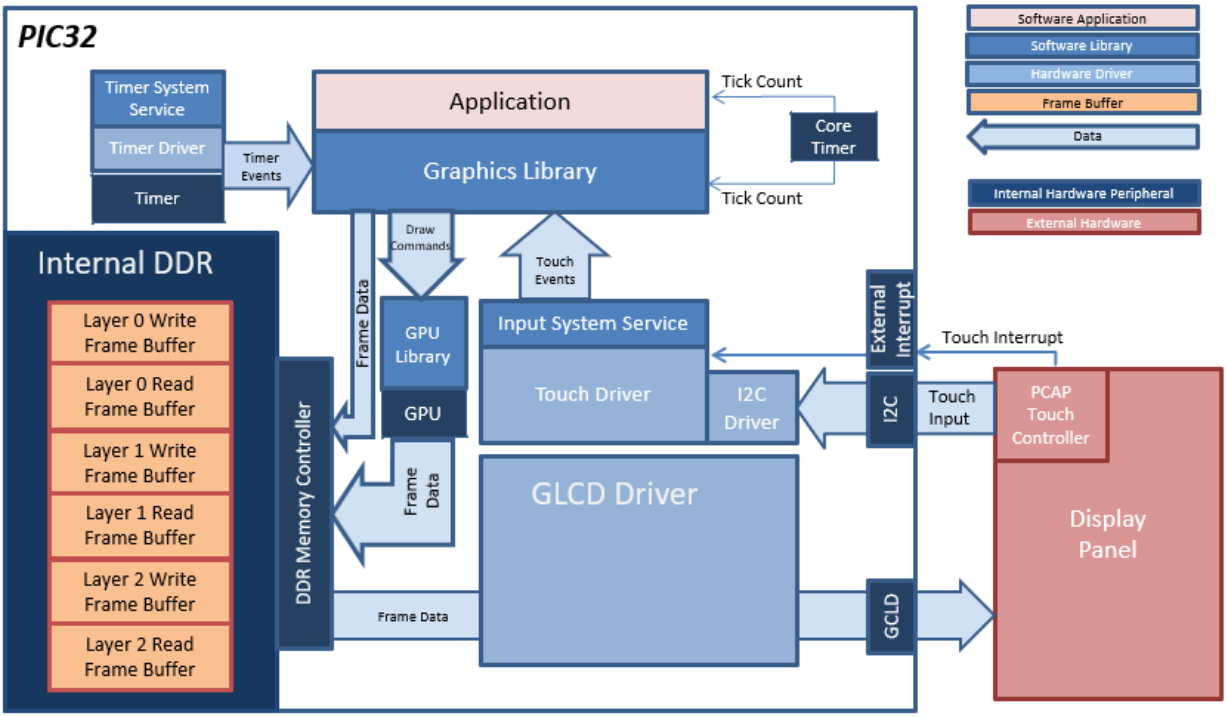
pic32mz_ef_sk_meb2_ext

Two 16-bit WQVGA frame buffers (~522kB) will not fit into the internal SRAM, thus the external SRAM is used for the frame buffers. These configurations use the Low Cost Controller-less (LCC) display driver to manage the DMA that transfers the frame buffer contents to the display.



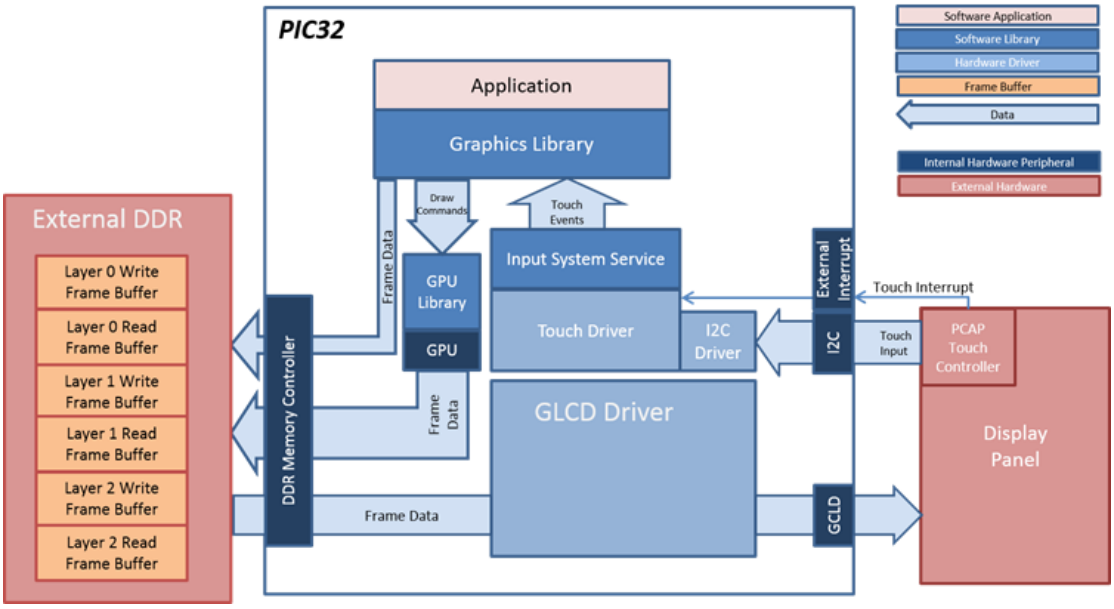
pic32mz_da_sk_intddr_meb2

In this configuration, the frame buffers are stored in the internal DDR memory. The DDR memory is large enough to allow 3 layers, each with double frame buffers. The contents of the read frame buffer are transferred continuously by the Graphics LCD (GLCD) controller to the display panel.



pic32mz_da_sk_extddr_meb2

This configurations works similar to the pic32mz_da_sk_intddr_meb2 except that the frame buffers are stored in external DDR.



Demonstration Features

- Input System Service and Touch driver
- DMA System Service
- Low-Cost Controllerless (LCC) Graphics Driver (PIC32MZ EF)
- I2C Driver
- Graphics LCD (GLCD) controller (PIC32MZ DA)
- 16-bit RGB565 color depth support (65535 unique colors)
- JPEG and RAW images stored in internal Flash (see *Volume III: MPLAB Harmony Configurator (MHC) > MPLAB Harmony Graphics Composer User's Guide > Graphics Composer Window User Interface > Graphics Composer Asset Management > Image Assets*)
- GFX widgets: Label and Buttons (see *Volume III: MPLAB Harmony Configurator (MHC) > MPLAB Harmony Graphics Composer User's Guide > Graphics Composer Window User Interface >Graphics Composer Asset Management > String Assets*)
- Double-buffering

Tools Setup Differences

- “Use Graphics Stack” is selected in the MPLAB Harmony Configurator (MHC), which enables the Graphics and Touch libraries and drivers for the Board Support Package (BSP).
- The heap size is set to 102400 bytes to ensure enough memory for dynamic allocations. This is done by setting the size in MHC through the following menu selection: *Device & Project Configuration >Project Configuration > XC32 (Global Options) >xc32-ld >General*.
- Double-buffering is enabled in the LCC Display Driver. This is done by selecting Double Buffer in MHC through the following menu selection: *Graphics Stack >Graphics Controller >Low Cost Controllerless >Frame Buffer Mode*.
- Use external memory to fit two frame buffers. This is done by selecting “External Memory” in MHC through the following menu selection: *Graphics Stack >Graphics Controller >Low Cost Controllerless >Memory Settings >Memory Interface Mode*.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Aria Counter demonstration.

Description

To build this project, you must open the `aria_counter.X` project in MPLAB X IDE, and then select the desired configuration. The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/gfx/aria_counter`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`aria_counter.X`	`<install-dir>/apps/gfx/aria_counter/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mz_ef_sk_meb2_ext	pic32mz_ef_sk+meb2	Demonstration for the PIC32MZ EF Starter Kit with the Multimedia Expansion Board II (MEB II) and external memory enabled.
pic32mz_da_sk_intddr_meb2	pic32mz_da_sk_intddr+meb2	Demonstration for PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit, Multimedia Expansion Board II (MEB II) and 4.3” WQVGA (480x272) Display.
pic32mz_da_sk_extddr_meb2	pic32mz_da_sk_extddr+meb2	Demonstration for PIC32MZ Embedded Graphics with External DRAM (DA) Starter Kit, Multimedia Expansion Board II (MEB II) and 4.3” WQVGA (480x272) Display.

Configuring the Hardware

This section describes how to configure the supported hardware.

Description

PIC32MZ EF Starter Kit and MEB II

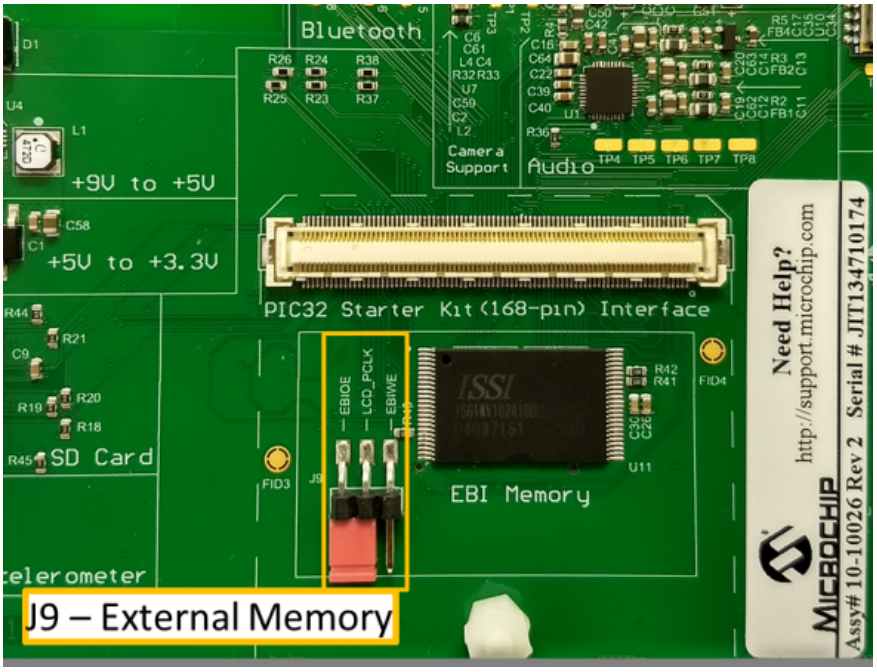
Configurations: pic32mz_ef_sk_meb2_ext

- This configuration requires that the J9 jumper be set to enable external SRAM for the frame buffer. Set the J9 jumper to connect the EBIOE and LCD_PCLK pins. The J9 jumper is located on the bottom of the MEB II board, beneath where the starter kit is plugged into the board. Refer to the following figure for the exact location.
- Connect the PIC32MZ EF Starter Kit board to the MEB II board
- Power up the board by connecting the power adapter to J3 power connector on the MEB II board or a powered USB cable to the USB DEBUG port on the Starter Kit board



Important!

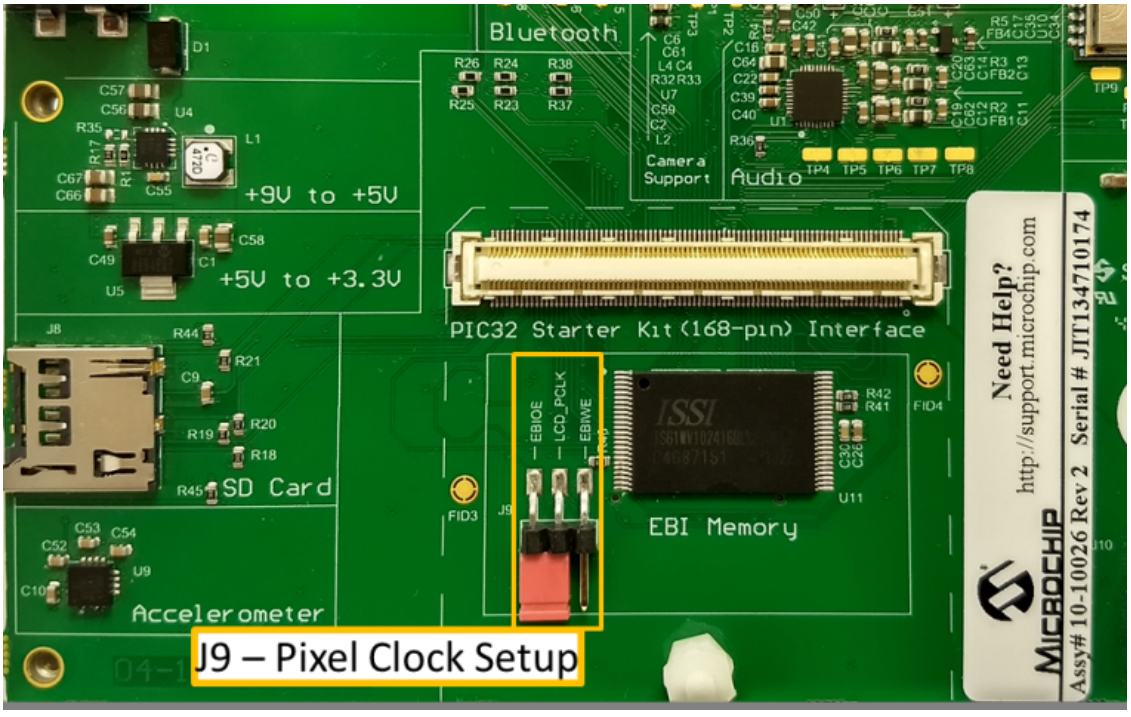
There are custom code blocks identified by // CUSTOM CODE – DO NOT DELETE … // END OF CUSTOM CODE comments that represent modifications to the code outside of MHC. These code blocks are necessary for correct operation of the demonstration under these configurations. When regenerating the project from within MHC use the "Prompt Merge For All Differences" merging strategy and do not change these custom code blocks.



BSP PIC32MZ Embedded Graphics with External DRAM (DA) Starter Kit, PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit and MEB II

Configurations: pic32_mz_da_sk_intddr_meb2, and pic32_mz_da_sk_extddr_meb2

- On the MEB II, the EBIOE and LCD_PCLK (J9) must be jumpered. A connection establishes the GLCD's pixel clock output timing. The external SRAM memory on the board is disabled. The J9 jumper is located on the bottom of the MEB II board, beneath where the starter kit is plugged into the board. Refer to the following figure for the exact location
- Connect the desired PIC32MZ DA Starter Kit to the MEB II board
- Power up the board by connecting the power adapter to J3 power connector on the MEB II board or a powered USB cable to the USB DEBUG port on the Starter Kit board



Running the Demonstration

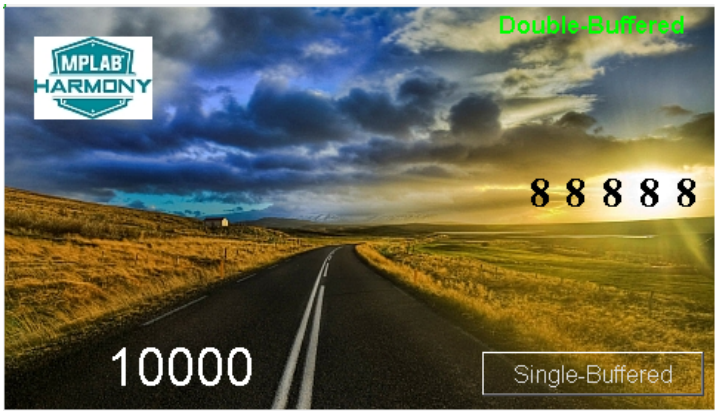
This section provides instructions on how to use the Aria Counter application.

Description

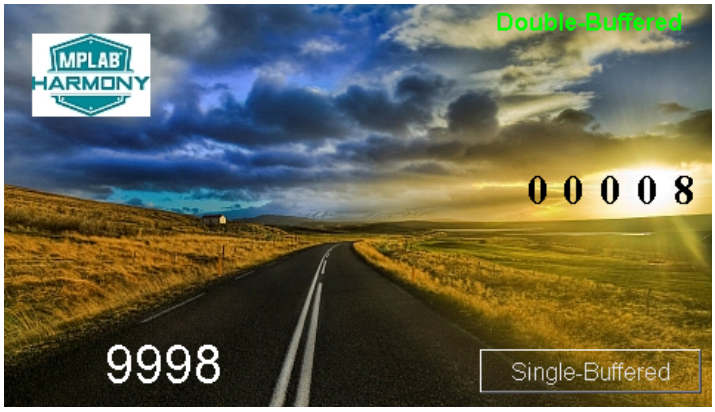
Upon boot-up, the application displays the animation Splash Screen, which shows an animated splash bar while blending in the MPLAB Harmony and Microchip PIC32 logos, as shown in the following figure.



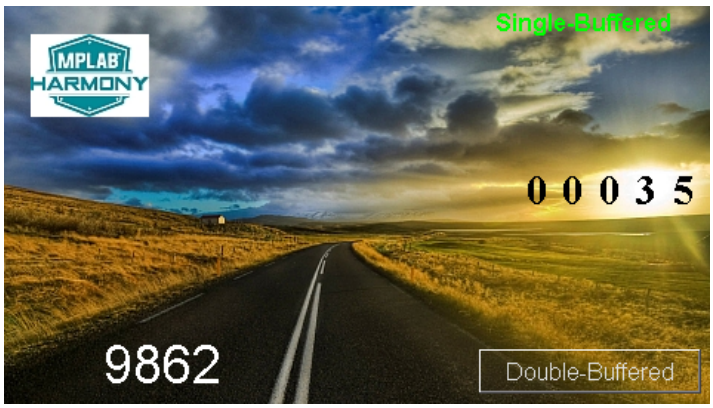
When the Splash Screen animation is complete, the Double-Buffered Screen will be shown:



The counters will automatically start with the value of upper counter incrementing, and the lower counter in white text decrementing. Since the application uses double-buffering, the counters update without tearing or flickering.



The Single-Buffered button in the lower-right corner can be touched to switch to a single-buffered screen. In the single-buffered screen, the counters will continue running, but they flicker as they are updated.



aria_external_resources

This demonstration illustrates the capability of storing media to internal, as well as external memory, and outlines how this can be done. The demonstration shows a graphic design that distributes resources between internal memory, external on board memory (i.e., SQI Flash), as well as external Memory Storage Device (MSD) (i.e, USB Flash drive). When running, the demonstration then fetches these resources from the external, as well internal memory at run-time and displays the resources on a screen.

The aria_flash demonstration serves as an external memory programmer to flash the off-chip non-volatile memory with the resources held on an MSD, such as a USB or a SD card.

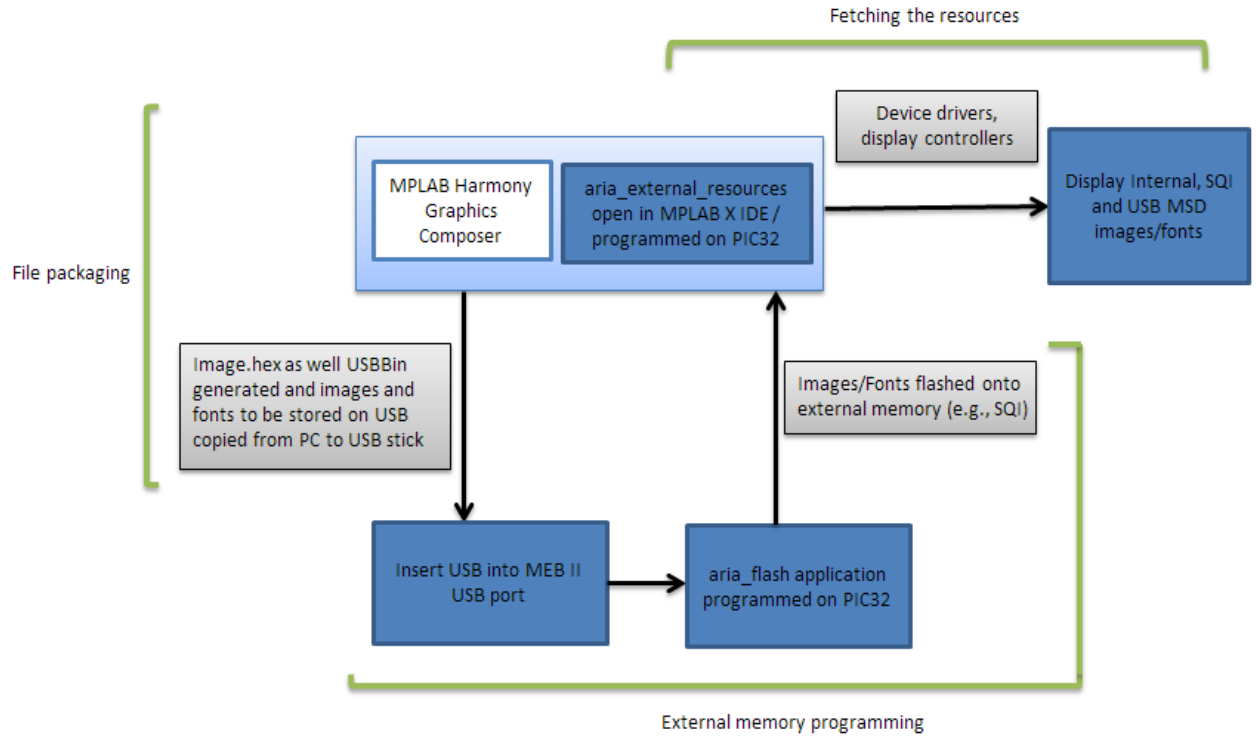
Description

The capability of storing resources on external memory is useful for applications that have large graphics resource requirements, such as storing large bitmap files or multiple page menu screens that require several images or large or multiple font packages, etc. In these instances, storing these graphics resources on-chip may result in insufficient memory or may be prohibitive to a possible cost benefit. A solution is to store the graphics resources to off-chip memory, such as non-volatile memory or MSD, thereby preserving the on-chip memory for program memory and allowing for more complex functional features.

To demonstrate how to access graphics resources stored on an external memory device, three components are needed:

- File Packaging
- Bootloader Application
- Fetch Application

The following figure shows the external resources process diagram.

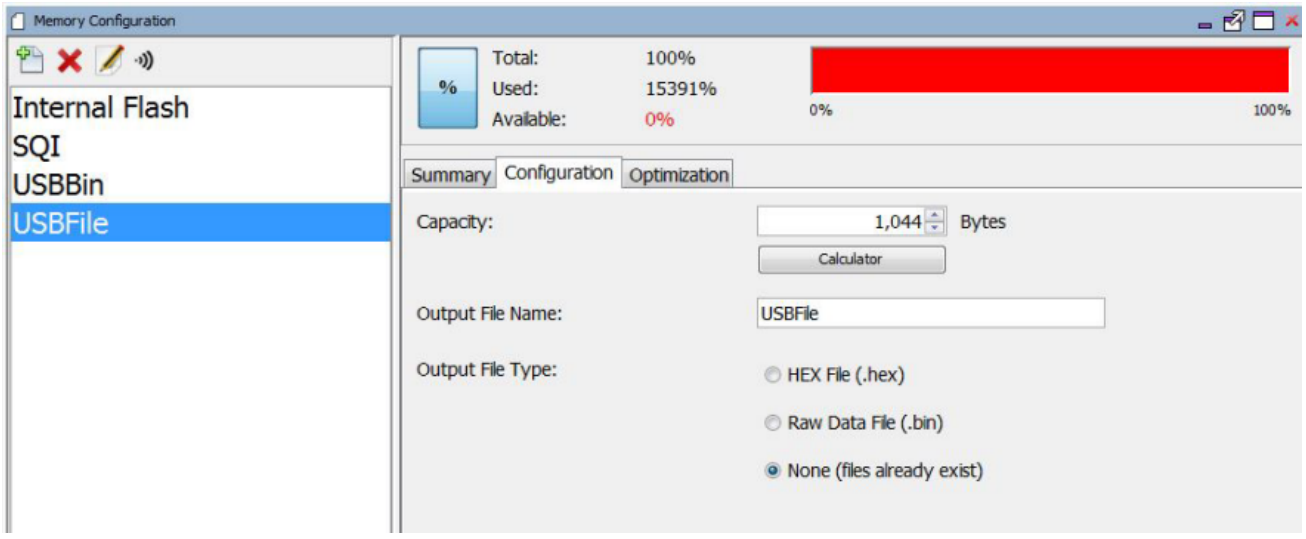


File Packaging

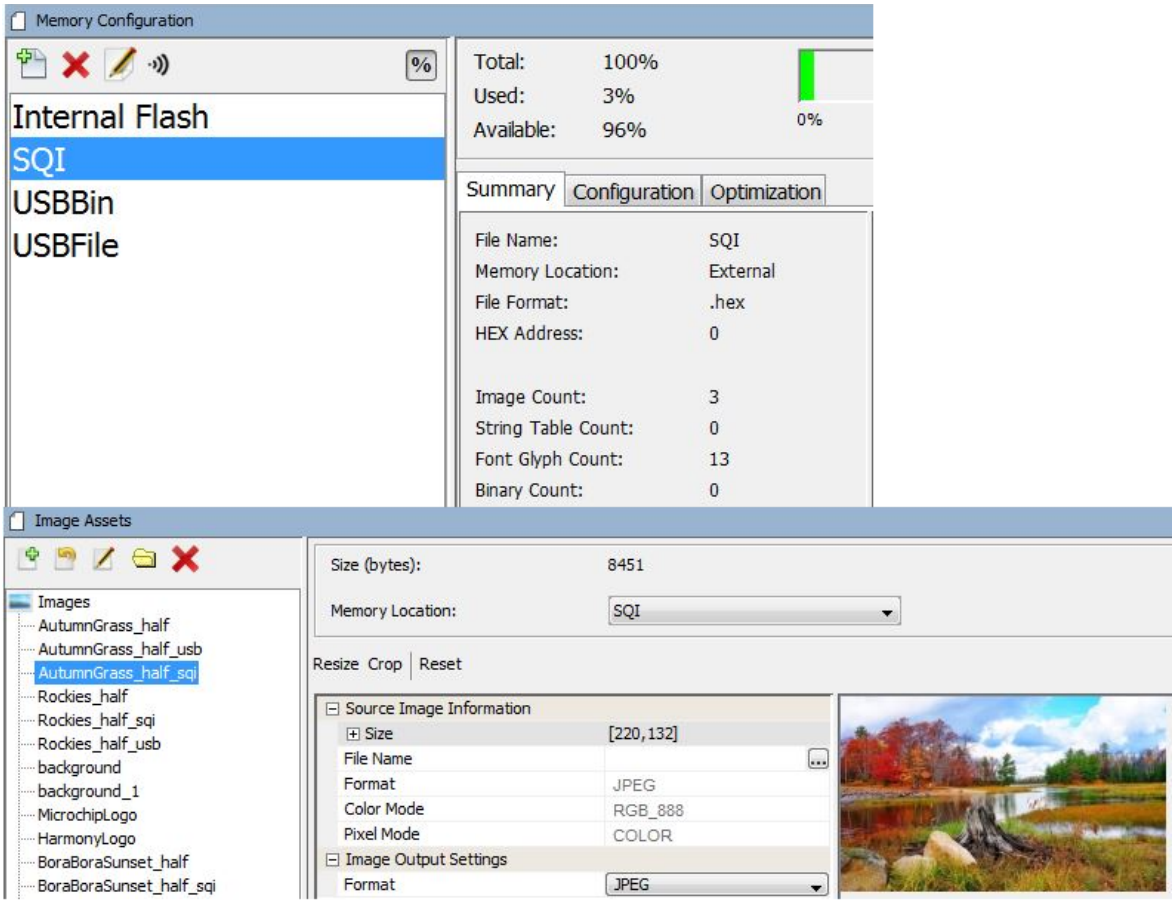
The resources to be stored externally need to be packaged into a resource file that would be flashed into the external memory. The MPLAB Harmony Graphics Composer, which is the primary tool used by the MPLAB Harmony application for configuring the graphic design for this application, has the ability to generate the required resource file containing the binary data for all of the graphics images to be stored on external memory, such as SQI, as well as a binary file to be stored on USB.

The graphics composer has the capability to manage the application assets by importing different resources including images, fonts, etc., into an application and choosing the memory location to store it (see the following figure).

Generating the graphics composer configuration creates the `SQI.hex` file as well as `USBBin.bin` file if resources were to be placed in USB by the composer design. The images to be stored as files on the USB can be loaded on the USB and do not need any HEX or BIN file information. However the user must make sure, when adding images to the MHGC design in the Graphics Composer, that the USB File option is selected, and in the Configuration tab, the *None* setting is selected for the memory location. Please refer to the following figure for further information.



MHGC Asset Manager for External Memory Resource Management



Memory Writer Application

A memory writer application is required to flash the .hex file containing the binary data of the external graphics resources, in this case SQI.hex into the external non-volatile memory. The memory writer application, aria_flash, uses a USB Flash drive or SD card, which has a FAT32 file system installed, in the Host mode as the source for the SQI.hex file. The USB Flash drive in the Host mode is scanned by the memory writer for this file and it is flashed onto the SQI Flash external memory. Refer to aria_flash for usage information.

Fetching the Resources

The third component is the fetching of the stored external graphics resources. The aria_external_resources application will demonstrate the fetching of the externally stored graphics images and using the MPLAB Harmony framework drivers, display controllers, etc., display those resources on the screen. The demos aim is to showcase the capability of storing media to internal as well as external memory and outline the procedure for the users of how this can be achieved.

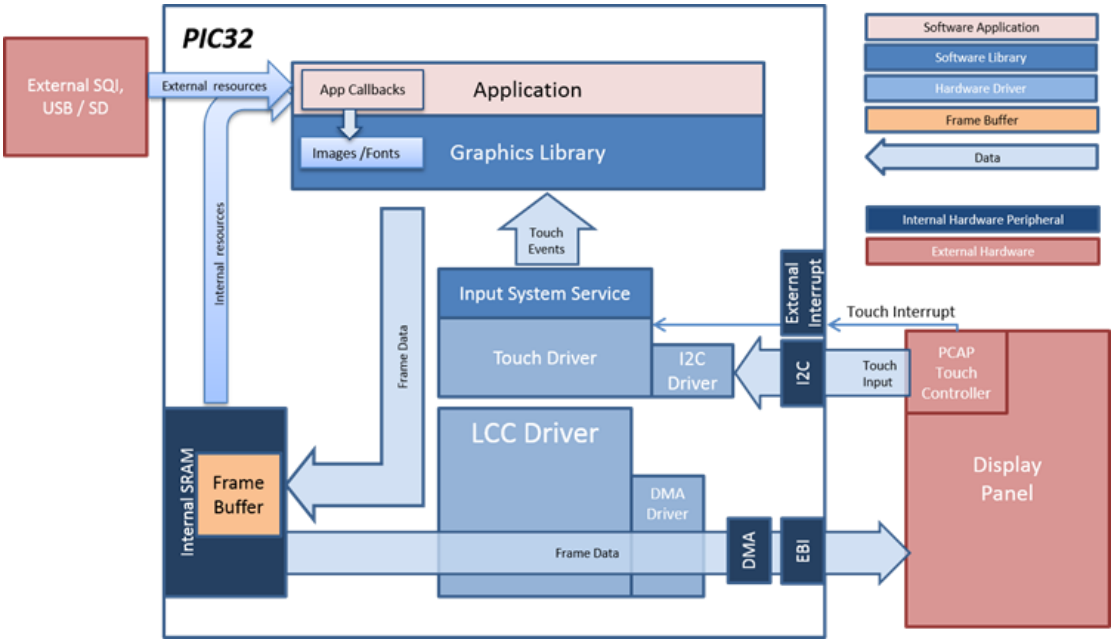
The application currently supports jpeg images and fonts. The same image/ text are stored and fetched from Internal Flash, SQI memory and USB. Selecting a listwheel setting, the user can choose which two media locations to load the resources from and compare the quality as well as efficiency loading from the different storage locations.

Architecture

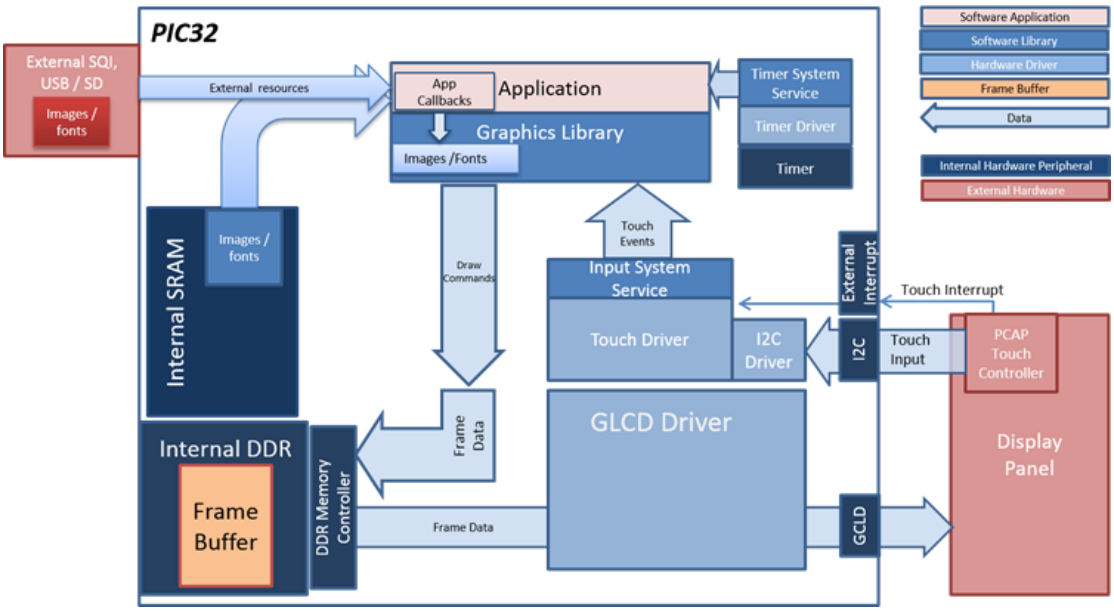
The aria_external_resources application uses the Graphics Library to render graphics to the display. The Graphics Library draws the widgets and images to the frame buffer that is stored in internal SRAM. Using the DMA, the Low-Cost Controllerless (LCC) Display Driver continuously transfers frame data from the frame buffer out to the LCD display. The resources (images and fonts) are fetched from the Internal SRAM, external non-volatile SQI memory and from the USB MSD attached to the Multimedia Expansion Board II (MEB II) using the USB file system.

The application also features user touch input through the integrated touch screen on the display panel. Touch input from the touch controller goes through the I2C port, and the Input System Service acquires the touch input information from the Touch and I2C drivers. The Input System Service sends touch events to the Graphics Library, which processes these events and updates the frame data accordingly.

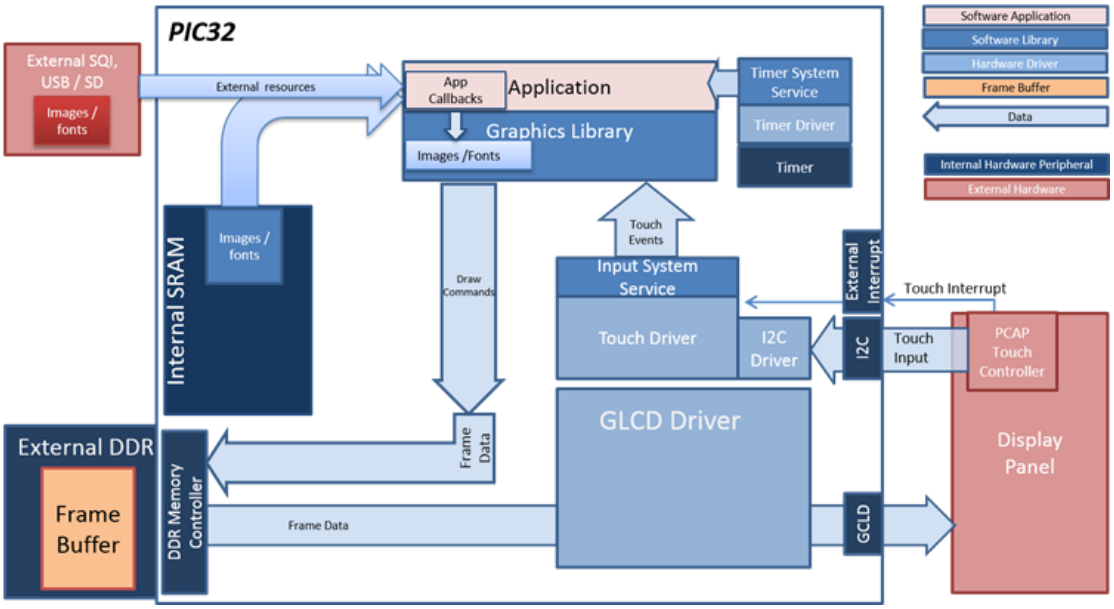
Demonstration Architecture Diagram: pic32mz_ef_sk_meb2



pic32mz_da_sk_intddr_meb2



pic32mz_da_sk_extddr_meb2



Demonstration Features

- Aria User Interface Library
- Input System Service
- Touch Driver
- DMA System Service
- I2C Driver
- External SQI memory
- USB system services and driver
- pic32mz_ef_sk_meb2 specific features:
  - 6 bit RGB565 color depth
  - LCC graphics
- pic32mz_da_sk_intddr_meb2 and pic32mz_da_sk_extddr_meb2 specific features:
  - 24 bit RGBA8888 color
  - GLCD internal display controller
  - Multiple layer graphics facilitated by DA hardware layers
  - Single frame buffer stored in the DDR

Tools Setup Differences

- “Use Graphics Stack” is selected in the MPLAB Harmony Configurator (MHC), which enables the Graphics and Touch libraries and drivers for the Board Support Package (BSP).
- The heap size is set to 204800 bytes to ensure enough memory for dynamic allocations. This is done by setting the size in MHC through the following menu selection: *Device & Project Configuration* > *Project Configuration* > *XC32 (Global Options)* > *xc32-ld* > General.
- USB Stack and USB Host Mode Driver

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Aria External Resources demonstration.

Description

To build this project, you must open the `aria_external_resources.X` project in MPLAB X IDE, and then select the desired configuration. The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/gfx/aria_external_resources`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`aria_external_resources.X`	`<install-dir>/apps/gfx/aria_external_resources/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mz_ef_sk_meb2	pic32mz_ef_sk+meb2	Demonstration for the PIC32MZ EF Starter Kit with the Multimedia Expansion Board II (MEB II) and external memory enabled.
pic32mz_da_sk_intddr_meb2	pic32mz_da_sk_intddr+meb2	Demonstration for the PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit with the Multimedia Expansion Board II (MEB II).
pic32mz_da_sk_extddr_meb2	pic32mz_da_sk_extddr+meb2	Demonstration for the PIC32MZ Embedded Graphics with External DRAM (DA) Starter with the Multimedia Expansion Board II (MEB II).

Important! This application contains custom code that is marked by the comments `// CUSTOM CODE ...` and `// END OF CUSTOM CODE`. If you use the MPLAB Harmony Configurator to regenerate the application code, use the "Prompt Merge For All Differences" merging strategy and *do not* remove or replace the custom code.

Configuring the Hardware

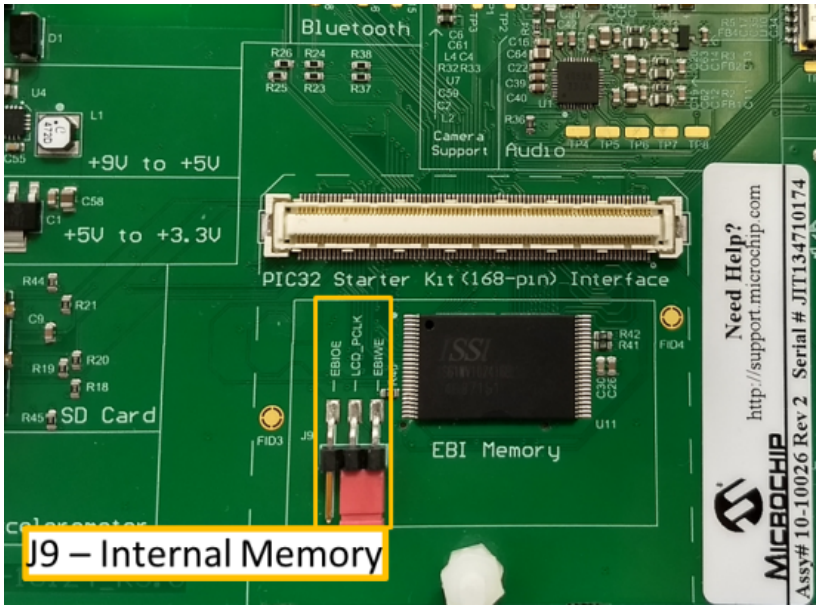
This section describes how to configure the supported hardware.

Description

PIC32MZ EF Starter Kit and MEB II

Configuration: pic32mz_ef_sk_meb2

- This configuration requires that the J9 jumper be set to enable internal SRAM for the frame buffer. Set the J9 jumper to connect the EBIOE and LCD_PCLK pins. The J9 jumper is located on the bottom of the MEB II board, beneath where the starter kit is plugged into the board. Refer to the following figure for the exact location.
- Connect the PIC32MZ EF Starter Kit board to the MEB II board
- Power up the board by connecting the power adapter to J3 power connector on the MEB II board or a powered USB cable to the USB DEBUG port on the Starter Kit board



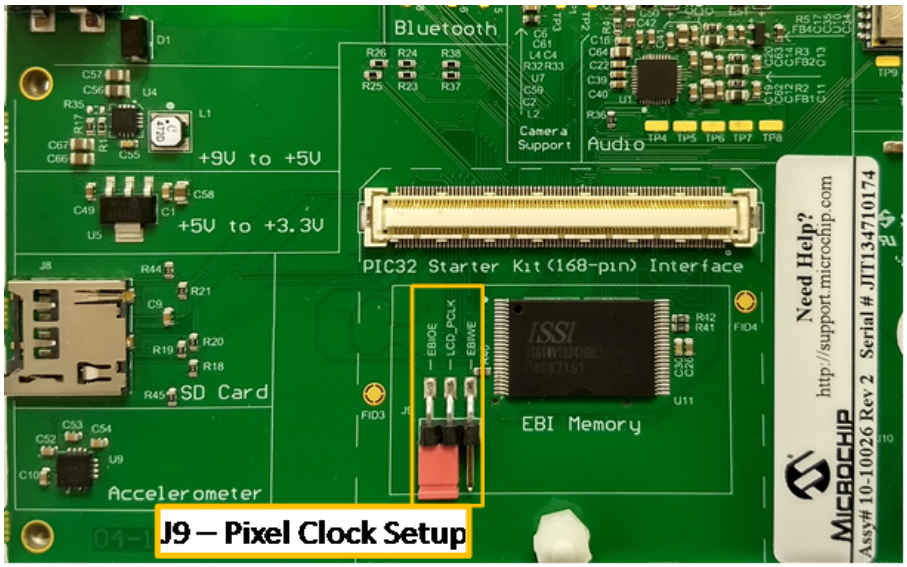
PIC32MZ Embedded Graphics with External DRAM (DA) Starter Kit and Multimedia Expansion Board II (MEBII), PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit and Multimedia Expansion Board II (MEB II)

Configuration: pic32mz_da_sk_intddr_meb2, and pic32mz_da_sk_extddr_meb2

On the MEB II, the EBIOE and LCD_PCLK (J9) must be jumpered. A connection establishes the GLCD's pixel clock output timing. The external SRAM memory on the board is disabled. The J9 jumper is located on the bottom of the MEB II board, beneath where the starter kit is plugged into the board. Refer to the following figure for the exact location.

- Connect the PIC32MZ DA Starter Kit board to the MEB II board
- Power up the board by connecting the power adapter to J3 power connector on the MEB II board or a powered USB cable to the USB DEBUG port on the Starter Kit board

- Use USB host port on Starter Kit for the media



Running the Demonstration

This section provides instructions on how to use the Aria External Resources demonstration.

Description

Demonstration Setup

1. In this step we are using the MPLAB Harmony Graphics Composer portion of the aria_external_resources application to create the `SQI.hex` file.
   - Open the `<install-dir>\apps\gfx\aria_external_resources` folder on your computer. The external resources to be stored on the external non-volatile memory are available in the `SQI.hex` file. In this demonstration, the file has already been generated and is available in the `<install-dir>\apps\gfx\aria_external_resources\firmware\src\system_config\<target configuration>\` folder. This file will be required by the external memory programmer application to flash the SQI memory.
   - If the user chooses to change the graphics design or external resources, the project aria_external_resources should be opened using MPLAB X IDE and the graphics should be changed using the graphics composer. Any change to the graphics would require regenerating the configuration and regenerating the `SQI.hex` file before being used by the external memory programmer application. This process requires both MPLAB X IDE and MPLAB Harmony. Refer to the MPLAB X IDE online help and *Volume III: MPLAB Harmony Configurator (MHC) > MPLAB Harmony Graphics Composer User's Guide* for usage information.
   - There are USB files that are images (JPEG and BMP) intended to be stored on the USB drive. These image files are located in the `\apps\gfx\aria_external_resources\firmware\src\assets.7z archive`. For this application, they are named `Image_usb_1.jpg`, `Image_usb_2.jpg`, `Image_usb_3.jpg`, and `Image_usb_4.bmp`.
   - See the aria_flash demonstration for instructions covering how to program the SQI external flash memory on the MEB2 board with the contents of the SQI.hex file.
2. The aria_external_resources application fetches and displays the external resources stored in external SQI memory. Program the aria_external_resources application onto the device.
   - The application demonstrates both internal and external resources being used together by distributing the images between internal and external memory
   - The application currently supports jpeg images and fonts. The identical image and font resources are stored and fetched from Internal Flash, SQI memory and USB.
   - The purpose of the demonstration is to showcase the capability of storing media to internal as well as external memory and outline the procedure for the users of how this can be achieved.
3. Please note that once the aria_external_resources have been flashed into the SQI memory, it is not required to reflash the resources every time while using the aria_external_resources application. However, if the user changes the graphics design affecting the external resources in the demonstration, the configuration needs to be regenerated and the new `SQI.hex` file needs to be reflashed to external memory using Step 2. This is to ensure that the external graphics resources being flashed on external memory are synchronized with the new graphics composer design utilizing the resources. Failure to do so will result in externally stored graphics images not being displayed correctly on screen.

The application boots up with the aria splash screen and the home screen, as shown in the following figure.



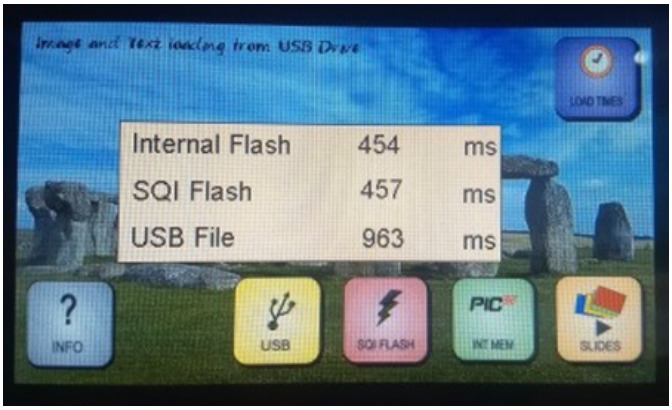
The home screen has an image from the internal memory loaded by default. It also loads text from the internal memory, and the load time for the image in milliseconds. The home screen also has buttons for the other available sources with resources preloaded on them, such as the SQI Flash memory, and the USB. If the USB is not plugged in, the USB button will not pop up on the screen. Pressing any of these buttons will load the same image and text from the respective storage sources. Each source has a different font, which shows the user where the font is coming from, and also displays the load times for the image for comparison.

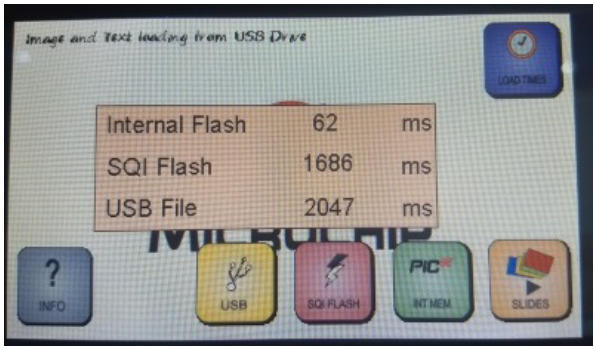
Another button available on the home screen, is the *Slides* button, which starts a slide show of different images loaded from a selected storage source. At any time during the running of the slide show, the user can press an alternate source to load images from, and the images and text start loading from the selected storage source. The slide show can be paused by pressing the same button again.

The home screen also has a button named *Load Times*. This button allows the comparison of load times of the same image from the three different storage sources - Internal Memory, SQI flash, and USB driver (when plugged in). It loads the image from each storage sequentially, and then displays the load times for all three sources together.

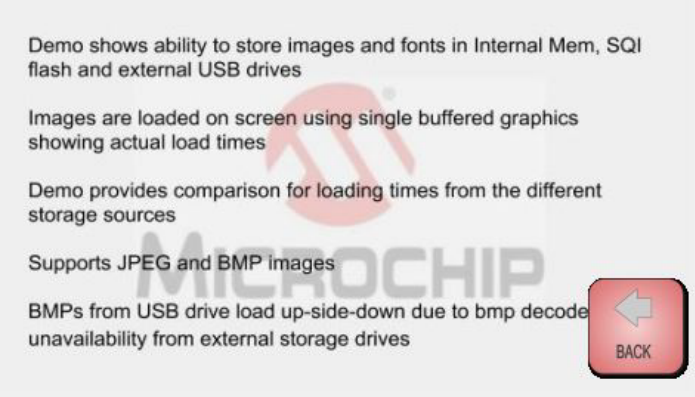
Finally, the INFO button on the screen takes the user to the help screen, which outlines briefly the objective of the demonstration and its highlights.

The application currently supports JPEG and BMP images. The BMP images show a large difference in the loading times from the Internal memory, as opposed to the external media from SQI Flash, and the USB. With JPEG images, the difference between loading a relatively smaller sized image due to data bus bandwidth, is outweighed by the decoding overhead. The load times are considerably closer for the SQI Flash and internal memory.





The Question mark button on the home screen takes the user to the help or Info screen, as shown in the following figure.



aria_flash

The aria_flash demonstration application serves as an external memory programmer to flash the off-chip non-volatile memory with the resources held on an Memory Storage Device (MSD), such as a USB or a SD card, which can then be accessed by other applications saving on-chip memory for other programs and resources.

The application aria_external_resources in MPLAB Harmony needs to use preloaded images/fonts from SQI flash external non-volatile memory. This would require aria_flash to flash the required image and font resources onto the SQI flash. Refer to aria_external_resources for usage model information.

Description

Applications that use large bitmap images or multiple page menu screens, which require several images or large or multiple font packages, etc., have a very large memory requirement for their graphics resources. In such applications, storing these graphics resources on-chip may result in insufficient memory or may be prohibitive to a possible cost benefit. A solution is to store the graphics resources to off-chip memory, such as non-volatile memory, thereby preserving the on-chip memory for program memory and allowing for more complex functional features.

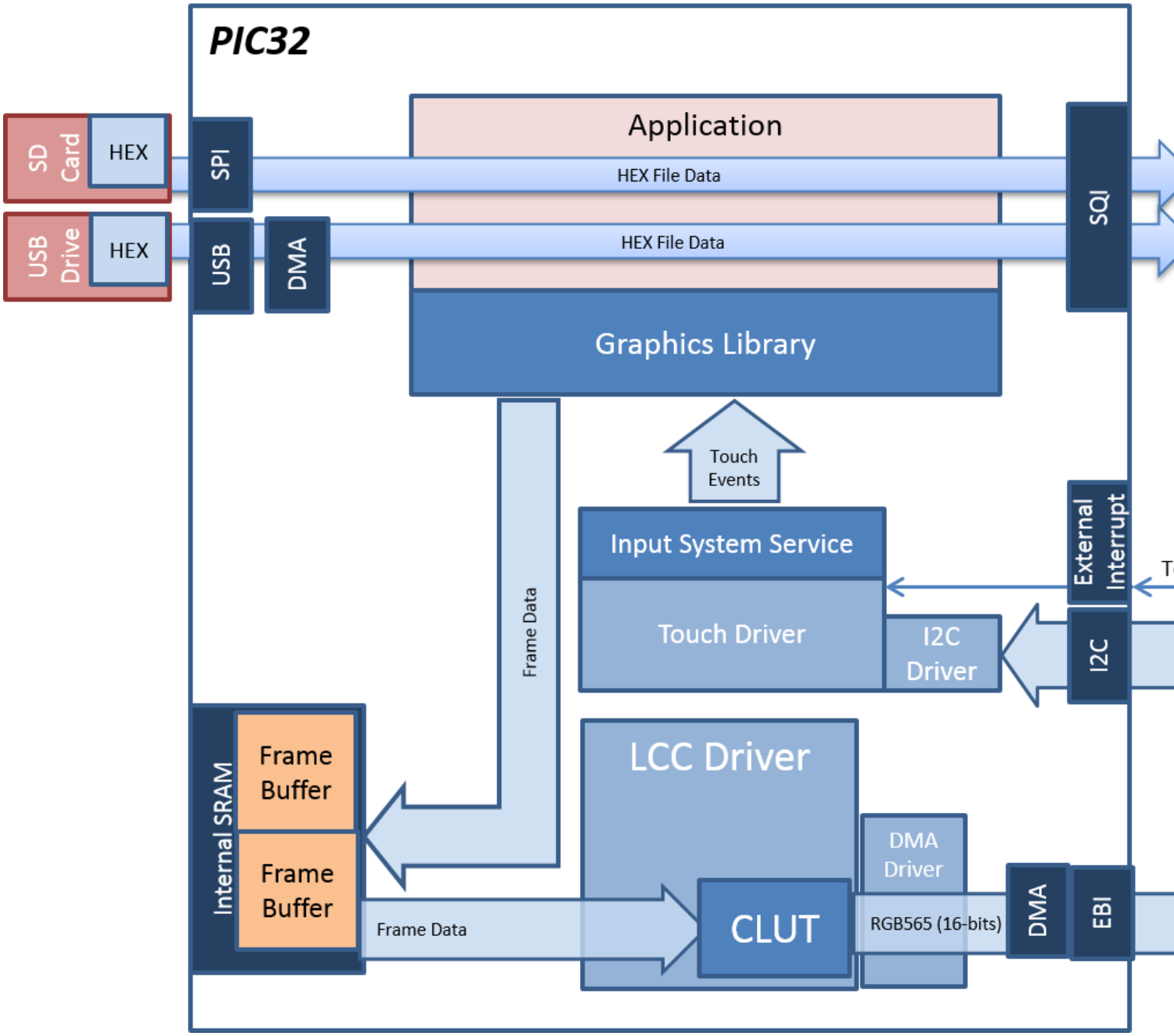
The aria_flash application flashes the `.hex` file containing the binary data of the external graphics resources, in this case `SQI.hex`, into the external non-volatile memory. The resource file is copied from the computer to a USB Flash drive or SD card, which has a FAT32 file system installed. The USB Flash drive/SD card is then plugged into the MEB II development board and the application scans the MSD for the resource file. The aria_flash application copies the resource file sector by sector and flashes the binary resource content onto the SQI Flash external memory.

Architecture

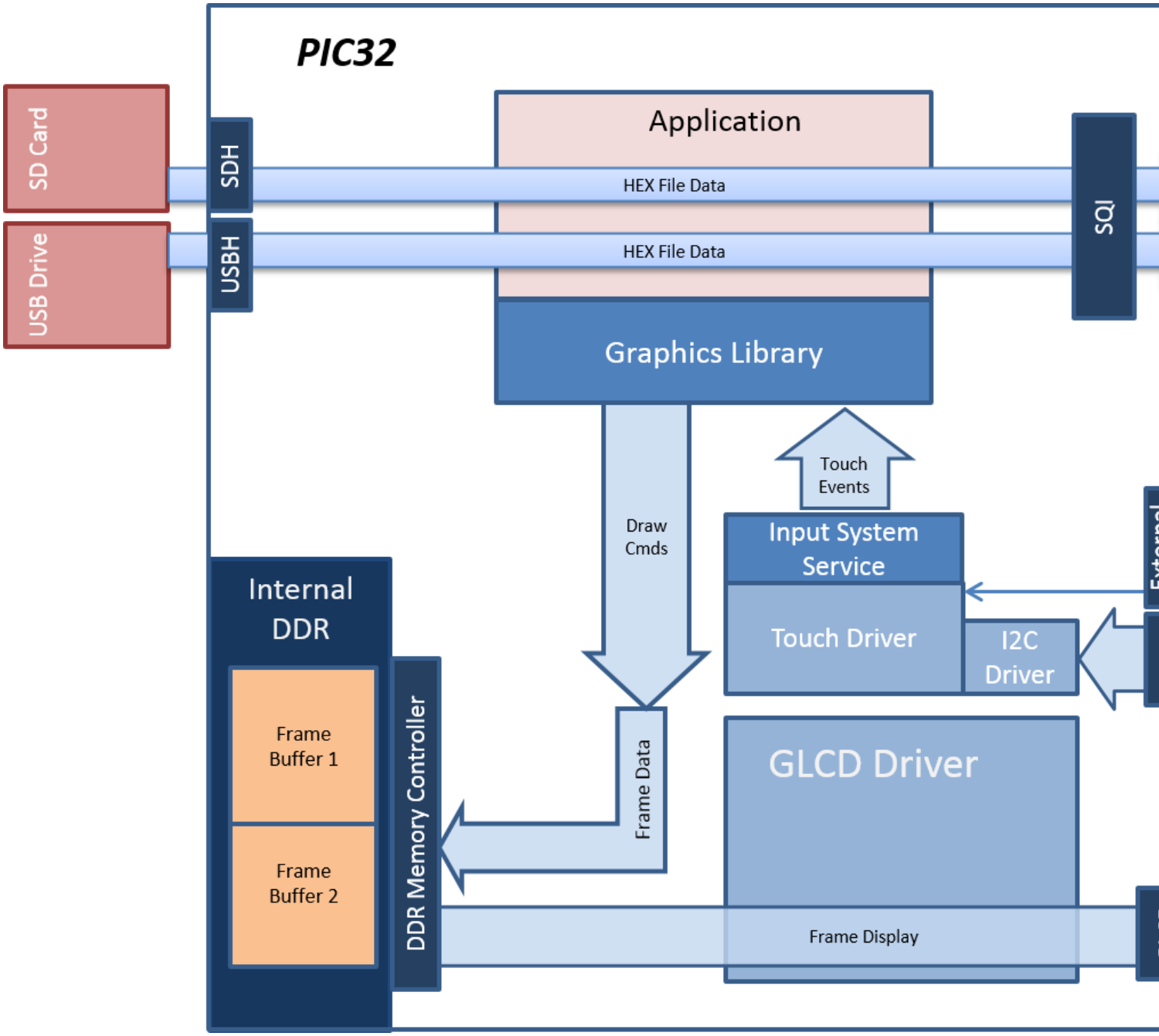
The aria_flash application uses the USB and SD card file systems in MPLAB Harmony and the SPI and USB drivers to scan the MSD for a `.hex` file with resources and reads them sector by sector and programs the external non-volatile SQI memory. The Graphics Library is used to render graphics to the display. Using the DMA, the Low-Cost Controllerless (LCC) Display Driver continuously transfers frame data from the frame buffer out to the LCD display.

The application also features user touch input through the integrated touch screen on the display panel. Touch input from the touch controller goes through the I2C port, and the Input System Service acquires the touch input information from the Touch and I2C drivers. The Input System Service sends touch events to the Graphics Library, which processes these events and updates the frame data accordingly.

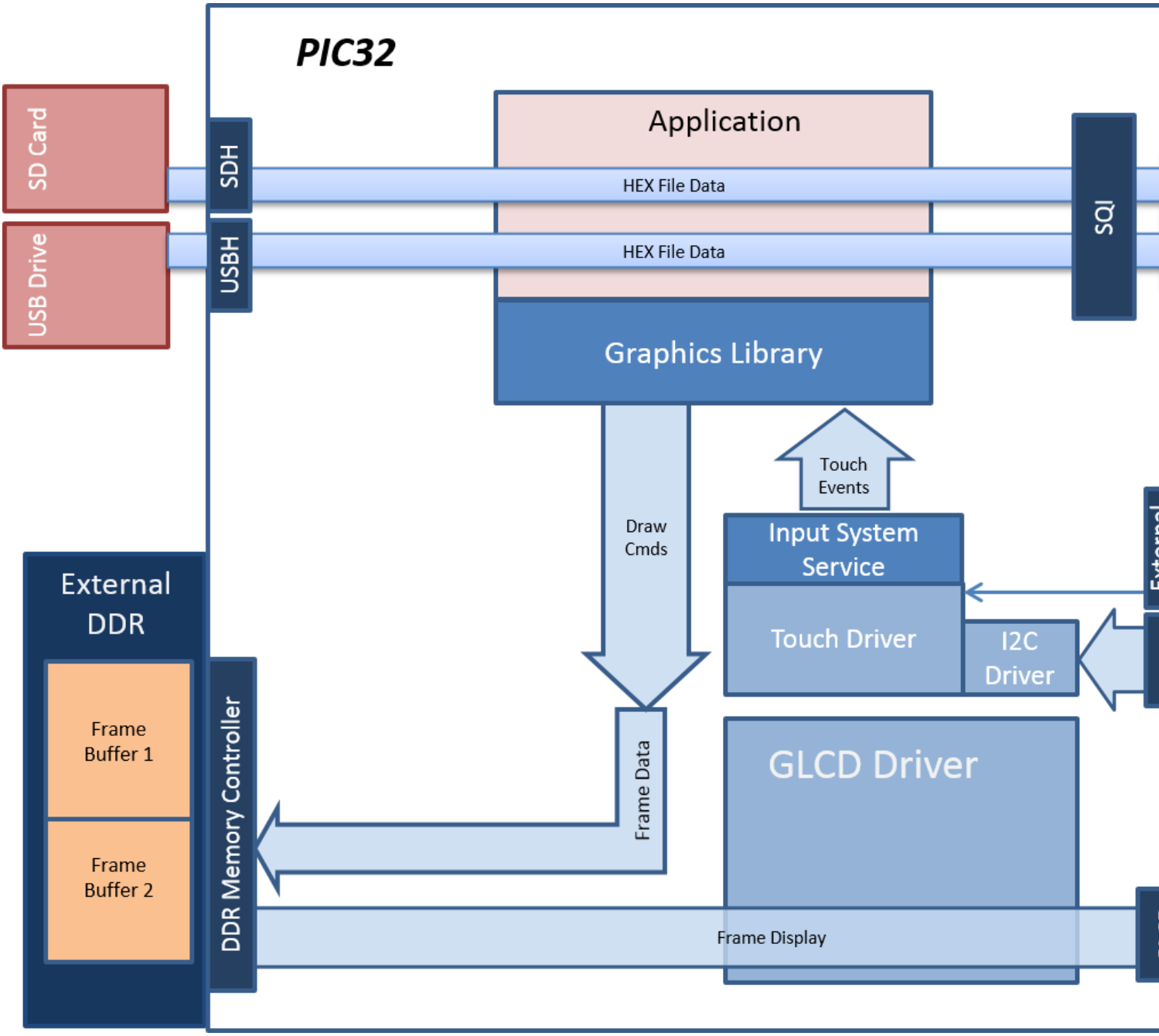
pic23mz_ef_sk_meb2_sqi



pic32mz_da_sk_intddr_meb2_sqi



pic32mz_da_sk_extddr_meb2_sqi



Demonstration Features

- USB file system and peripheral
- SD card file system and SPI Driver
- Aria User Interface Library
- Input System Service
- Touch Driver
- DMA System Service
- I2C Driver
- External SQI memory

Tools Setup Differences

- “Use Graphics Stack” is selected in the MPLAB Harmony Configurator (MHC), which enables the Graphics and Touch libraries and drivers for the Board Support Package (BSP).
- The heap size is set to 204800 bytes to ensure enough memory for dynamic allocations. This is done by setting the size in MHC through the following menu selection: *Device & Project Configuration > Project Configuration > XC32 (Global Options) > xc32-ld > General*.
- USB Stack and USB Host Mode Driver

- SD card and SPI Driver selection
- SQI external memory support: Since a formal SQI driver is not available in MPLAB Harmony, the external memory programmer uses the SQI Peripheral Library. Please note that the source and header file for SQI enabling have been manually included in the project within `SourceFiles/drv_nvm_flash_sqi_sst26.c` and `HeaderFiles/drv_nvm_flash_sqi_sst26.h`.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Aria Flash demonstration.

Description

To build this project, you must open the `aria_flash.X` project in MPLAB X IDE, and then select the desired configuration. The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/gfx/aria_flash`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`aria_flash.X`	`<install-dir>/apps/gfx/aria_flash/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mz_ef_sk_meb2_sqi	pic32mz_ef_sk+meb2	Demonstration for the PIC32MZ EF Starter Kit with the Multimedia Expansion Board II (MEB II) and external memory enabled.
pic32mz_da_sk_intddr_meb2_sqi	pic32mz_da_sk_intddr+meb2	Demonstration for the PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit with the Multimedia Expansion Board II (MEB II).
pic32mz_da_sk_extddr_meb2_sqi	pic32mz_da_sk_extddr+meb2	Demonstration for the PIC32MZ Embedded Graphics with External DRAM (DA) Starter Kit with the Multimedia Expansion Board II (MEB II).

Important! This application may contain custom code that is marked by the comments `// CUSTOM CODE ...` and `// END OF CUSTOM CODE`. If you use the MPLAB Harmony Configurator to regenerate the application code, use the "Prompt Merge For All Differences" merging strategy and *do not* remove or replace the custom code.

Configuring the Hardware

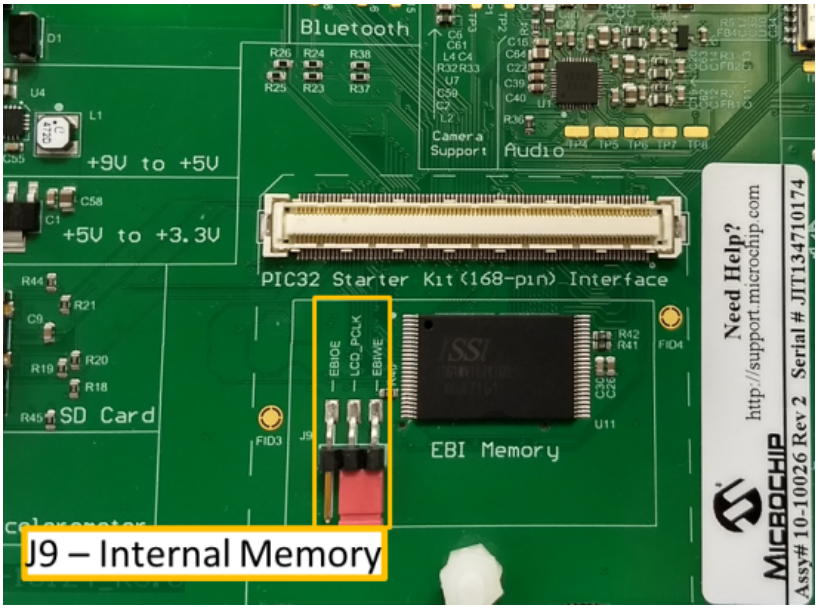
This section describes how to configure the supported hardware.

Description

PIC32MZ EF Starter Kit and MEB II

Configuration: pic32mz_ef_sk_meb2_sqi

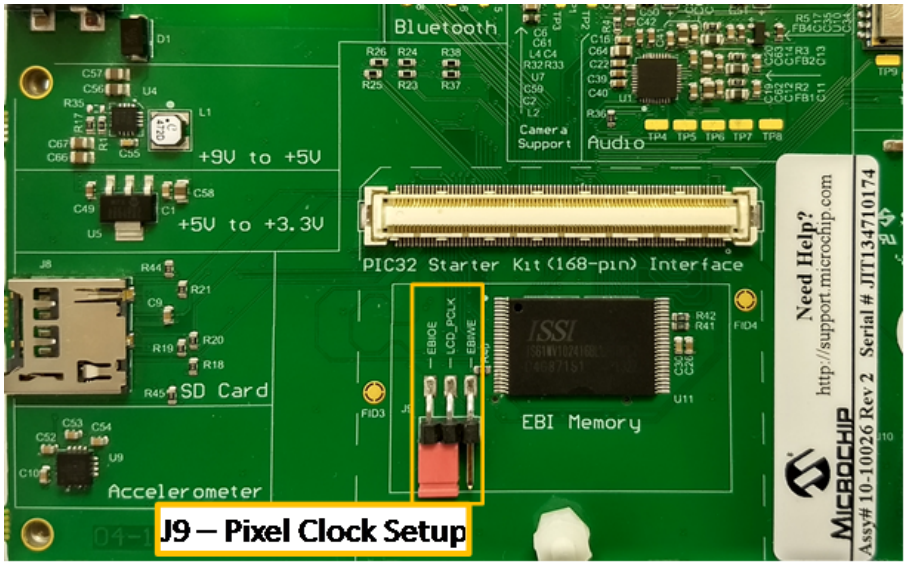
- This configuration requires that the J9 jumper be set to enable internal SRAM for the frame buffer. Set the J9 jumper to connect the EBIOE and LCD_PCLK pins. The J9 jumper is located on the bottom of the MEB II board, beneath where the starter kit is plugged into the board. Refer to the following figure for the exact location.
- Connect the PIC32MZ EF Starter Kit board to the MEB II board
- Power up the board by connecting the power adapter to J3 power connector on the MEB II board or a powered USB cable to the USB DEBUG port on the Starter Kit board



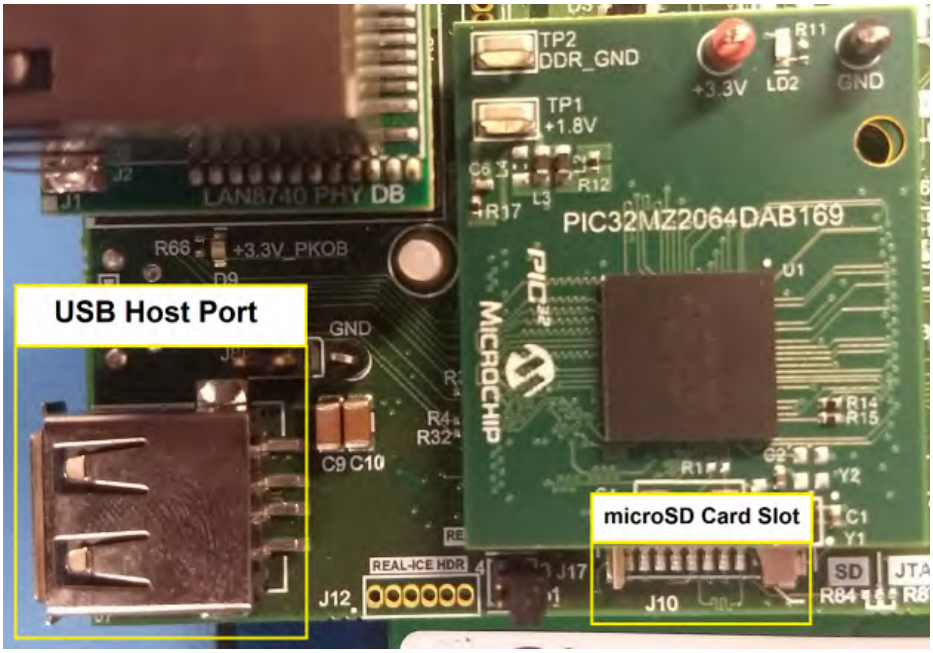
PIC32MZ Embedded Graphics with External DRAM (DA) Starter Kit with Multimedia Expansion Board II (MEBII), PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit and Multimedia Expansion Board II (MEB II)

Configurations: pic32mz_da_sk_extddr_meb2_sqi, pic32mz_da_sk_intddr_meb2_sqi

On the MEB II, the EBIOE and LCD_PCLK (J9) must be jumpered. A connection establishes the GLCD's pixel clock output timing. The external SRAM memory on the board is disabled. The J9 jumper is located on the bottom of the MEB II board, beneath where the starter kit is plugged into the board. Refer to the following figure for the exact location.



- Connect the PIC32MZ DA Starter Kit board to the MEB II board
- Power up the board by connecting the power adapter to J3 power connector on the MEB II board or a powered USB cable to the USB DEBUG port on the Starter Kit board
- Use the microSD slot and USB host port on Starter Kit for the media. Refer to the following figure for the location of the ports



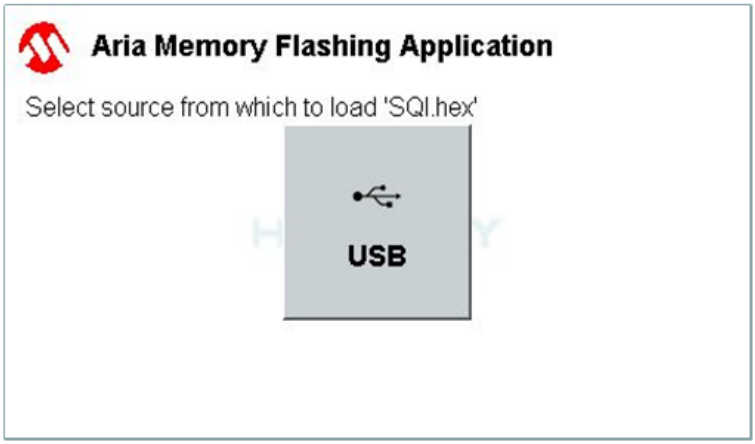
Running the Demonstration

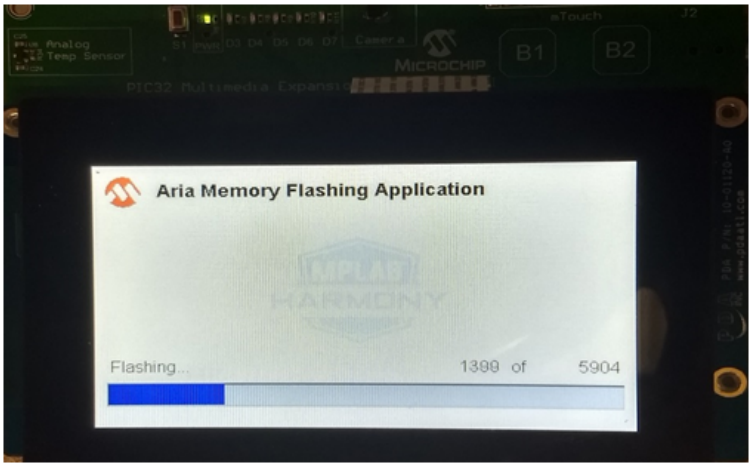
This section provides instructions on how to use the Aria Flash demonstration.

Description

The aria_flash application loads the required `SQI.hex` file to an off-chip external memory to facilitate large resource requirements, which would not be easily accommodated on on-chip internal memory. The external memory programmer is operated as follows.

1. The external resources to be flashed onto the external memory are required by the aria_flash application in the form of a HEX text file with the default name SQI.hex. If you do not have the resource binary file, you can obtain it from the following MPLAB Harmony installation folder: `<install-dir>\apps\gfx\aria_external_resources\firmware\src\system_config\<target configuration>\SQI.hex`, which should copied and renamed to `SQI.hex`.
2. Connect a USB Flash drive or SD card to the computer port and copy the `SQI.hex` file from `<install-dir>\apps\gfx\aria_external_resources\firmware\src\system_config\<target configuration>\` to the USB Flash drive or SD card.
3. After the file copy has finished, remove the USB Flash drive or SD card from the computer and insert it into the USB host connector on your starter kit board or the SD card slot.
4. Program the PIC32MZ device with the external memory programmer application, aria_flash. This application flashes the `SQI.hex` file available on the USB Flash drive or SD card to the external SQI memory.
5. Please make sure that the USB thumb drive or SD card is not unplugged while the program is actively flashing the memory.
6. When the USB or SD card is plugged in, the graphics display shows the USB or SD card pop-up for selection to start the flashing process. Once selected, a progress bar shows the user the flashing process starting and progressing to completion.





aria_image_viewer

This demonstration discusses how to use the aria_image_viewer application.

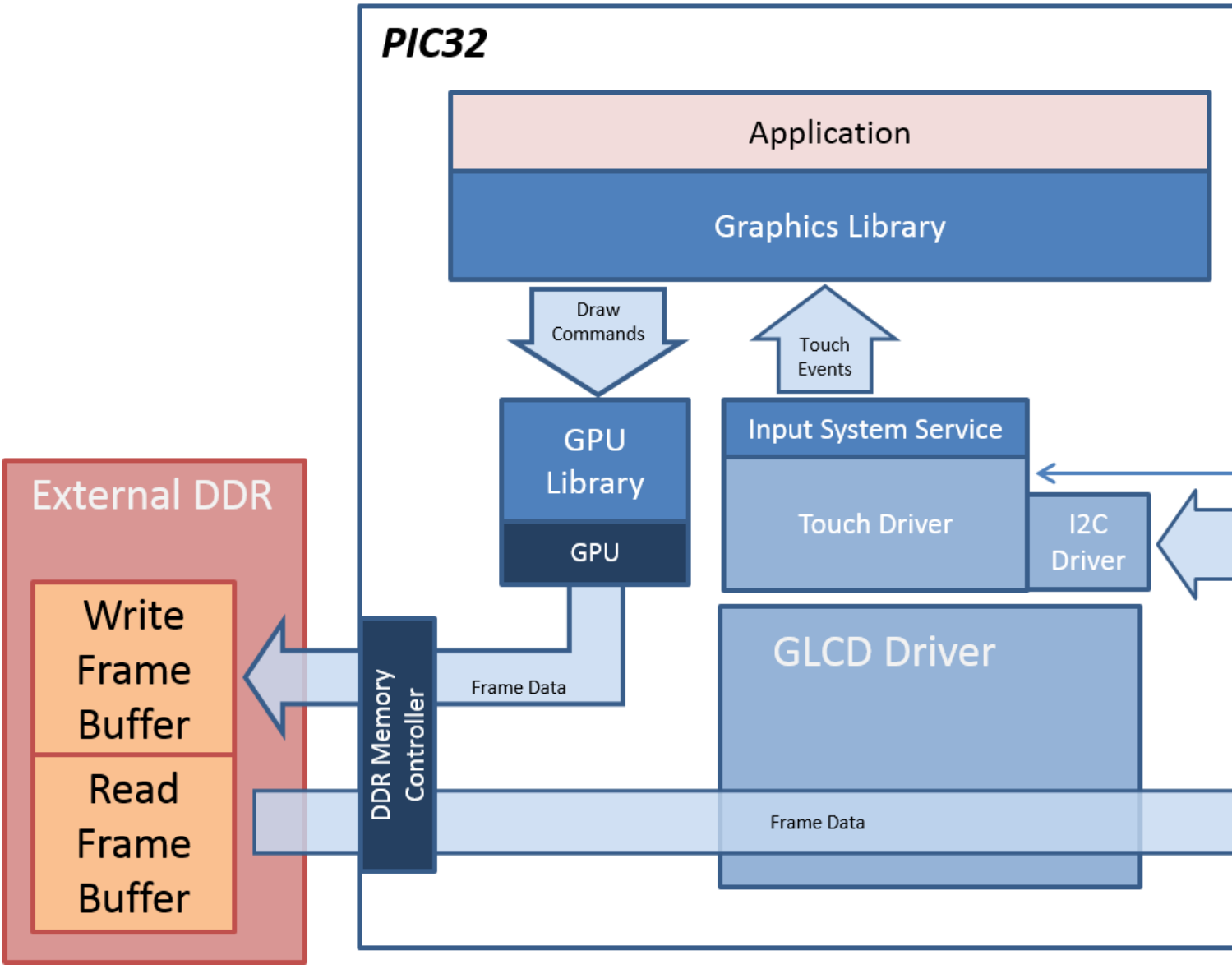
Description

This application demonstrates high-resolution image viewing using single and multi-touch gestures using the capabilities of the Graphics Suite's GPU Bit Block (Blit) rendering engine. It displays a 2716x1810 high resolution image onto the native MEBII display size using Blit stretch, shrink, and copy features. It also demonstrates image resizing and panning through Input System Services, user gesture touch controls, and image orientation using the User Interface (UI) button event. The application uses three layers of the GLCD, Aria Graphics Library, and low-level GPU APIs for the Nano2D Library to realize image viewing features. The demonstration communicates fast block copying of 4.9 million 32bpp pixels, smooth real-time motion, layer composition and blending. The key points are:

- Using GPU Blit copy to show motion
- Using DDR memory to stage a 2716x1810 high-resolution image
- Using Aria Graphics Library to for all image decoding and staging
- Using GLCD layer composition
- Input System Services (multi-touch)

Architecture

The following figure shows the various software and hardware components used by this application.



The aria_image_viewer application uses the MPLAB Harmony Graphics Library to render graphics to the display. It uses the 2-D GPU and the GLCD internal hardware peripherals. The Graphics Suite's Graphics Composer Tool is used to establish the UI design. All images are staged to the external DDR during initialization.

The first layer (0) is the image destination layer. At initialization, the Graphics Library decodes and stages four full 2716x1810 high-res images to the external DDR. The staged images are used as a source for subsequent user control requests, such as shrink, stretch, and pan. A version of the image is displayed to the destination layer 0 based on the user control request.

The second layer (1) is the UI layer. The Graphics Composer Tool is used to establish all user controls. An Orientation Button Widget is used to set image orientation at 0, 90, 180, and 270. The UI layer also contains non-visible touch areas for multi-touch gestures. User supported gestures are swipe to next image, two-finger pinch-zoom, and single finger pan. Lastly, the UI layer contains a touch area to display a help guide.

The third layer (2) is the help layer. The Graphics Composter Tool is used to establish helpful UI Text Labels. This layer maintains an alpha-channel which is set to blend over the UI and image layers. It is only available at initialization and upon user request. Its alpha value gradually transitions from opaque to fully transparent, so the panel becomes invisible after a few seconds.

The application is touch enabled. It features user touch input through the integrated touch screen on the display panel. Touch input from the touch controller goes through the I2C port, and the Input System Service acquires the touch input information from the Touch and the I2C Drivers. The Input System Service sends touch events to the Graphics Library, which processes these events and updates the frame data accordingly.

In addition, the application uses the core timer clock ticks for some automatic image motion control.

Demonstration Features

- 32bit true-color with alpha-channel
- GPU Blit copy, shrink, stretch
- Three GLCD layers
- 2716x1810 high-resolution image

- External memory DDR2
- Graphics User controls
- Simple real-time motion

Tools Setup

- Enable Graphics Stack
- Enable Graphics Controller
  - GLCD
  - Enable all 2 layers
- Enable Graphics Processor
  - Use Nano2D

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Aria Image Viewer demonstration.

Description

To build this project, you must open the `aria_image_viewer.X` project in MPLAB X IDE, and then select the desired configuration. The following tables list and describe the project and supported configurations. The parent folder for these files is *<install-dir>/apps/gfx/aria_image_viewer*.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Project Location
aria_image_viewer	*<install-dir>/apps/gfx/ aria_image_viewer /firmware*

MPLAB X IDE Project Configuration

This table lists and describes the supported configurations of the demonstration, which are located within *./firmware/src/system_config*.

Project Configuration Name	BSP Used	Description
pic32mz_da_sk_extddr_meb2	pic32mz_da_sk_extddr+meb2	Demonstration for the PIC32MZ DA Starter Kit with the Multimedia Expansion Board II (MEB II) (blue board).

Note: This application may contain custom code that is marked by the comments "// START OF CUSTOM CODE ..." and "// END OF CUSTOM CODE". If you use the MPLAB Harmony Configurator to regenerate the application code, use the "Prompt Merge For All Differences" merging strategy and do not remove or replace the custom code.

Configuring the Hardware

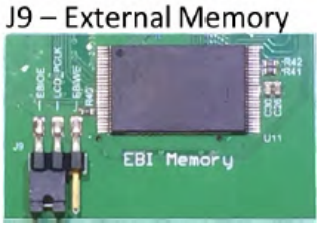
This section describes how to configure the supported hardware.

Description

PIC32MZ DA Starter Kit and MEB II

Configurations: pic32mz_da_sk_extddr_meb2

- On the MEB II, the EBIWE and LCD_PCLK (J9) must be open. A closed setting turns on internal SRC and open setting turns on I external memory. The jumper (J9) is available on the bottom side of the MEB II board. See the following figure below for jumper location.
- Connect the PIC32MZ DA Starter Kit board to the MEB II board
- Power up the board by connecting the power adapter to J3 power connector on the MEB II board or a powered USB cable to the USB DEBUG



Running the Demonstration

This section provides instructions on how to use the Aria Image Viewer demonstration.

Description

The aria_image_viewer demonstration has two UI screens. The first, SplashScreen, is displayed upon boot-up. It has no user controls, but displays a generic method that can be used to display logo content during initialization. The second UI screen is the ImageScreen. After the SplashScreen is displayed, the application automatically transitions to the ImageScreen and displays the touch gesture areas. During the touch gesture display, the image is loaded, then the screen will fade away showing a 2716x1810 high-resolution bridge image, that is reduced to the native display screen size of the MEBII. At this point, the pinch-zoom and swipe gestures are available for use.

Initial Image



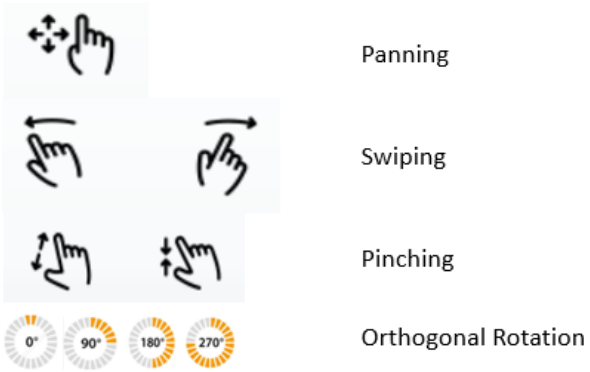
Initial Image at 90 Degree Orientation



Initial Image at Zoom-In Using Two-Finger Zoom and Pan

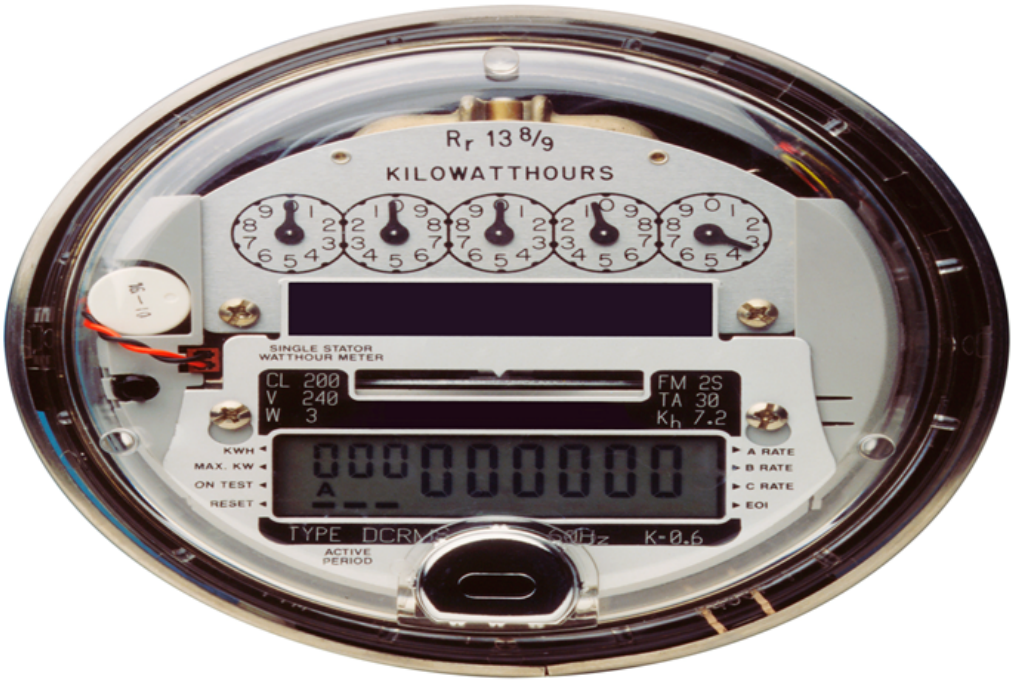


Help Overlay

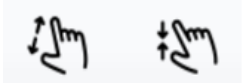


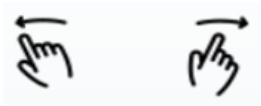
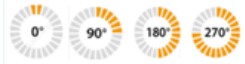
The following images are the remaining three of the four images that can be used for gesture actions.







Name of Motion	Description of Function
	Enables the user to perform zoom-in on the center aspect. User can then pan the image and perform additional zoom or pinch gestures. Each movement exercises 2-D GPU Blit using smaller or larger rectangles clipping of the image source placed on to the native LCD display.

	Enables the user to swipe to the next image. The current image will dim using a A8 alpha buffer. The next image will be displayed at zoom level 0. Each movement is tested on speed of swipe over a distance.
	Enables the user to move the image in any direction limited by the image extents. Pan requires the zoom-in operation to be greater than 0%. Each movement exercises 2-D GPU Blit operation.
	Enables the user to set the orientation to an orthogonal value of 0, 90, 180, or 270 degrees. Each movement exercises 2-D GPU buffer orientation operation.
Help Area (top of display)	Enables the user to activate the help overlay. Displays the supported touch gestures.
Image Stretch Text (lower left)	Name of demonstration

aria_oven_controller

The following demonstration details how to use the aria_oven_controller.

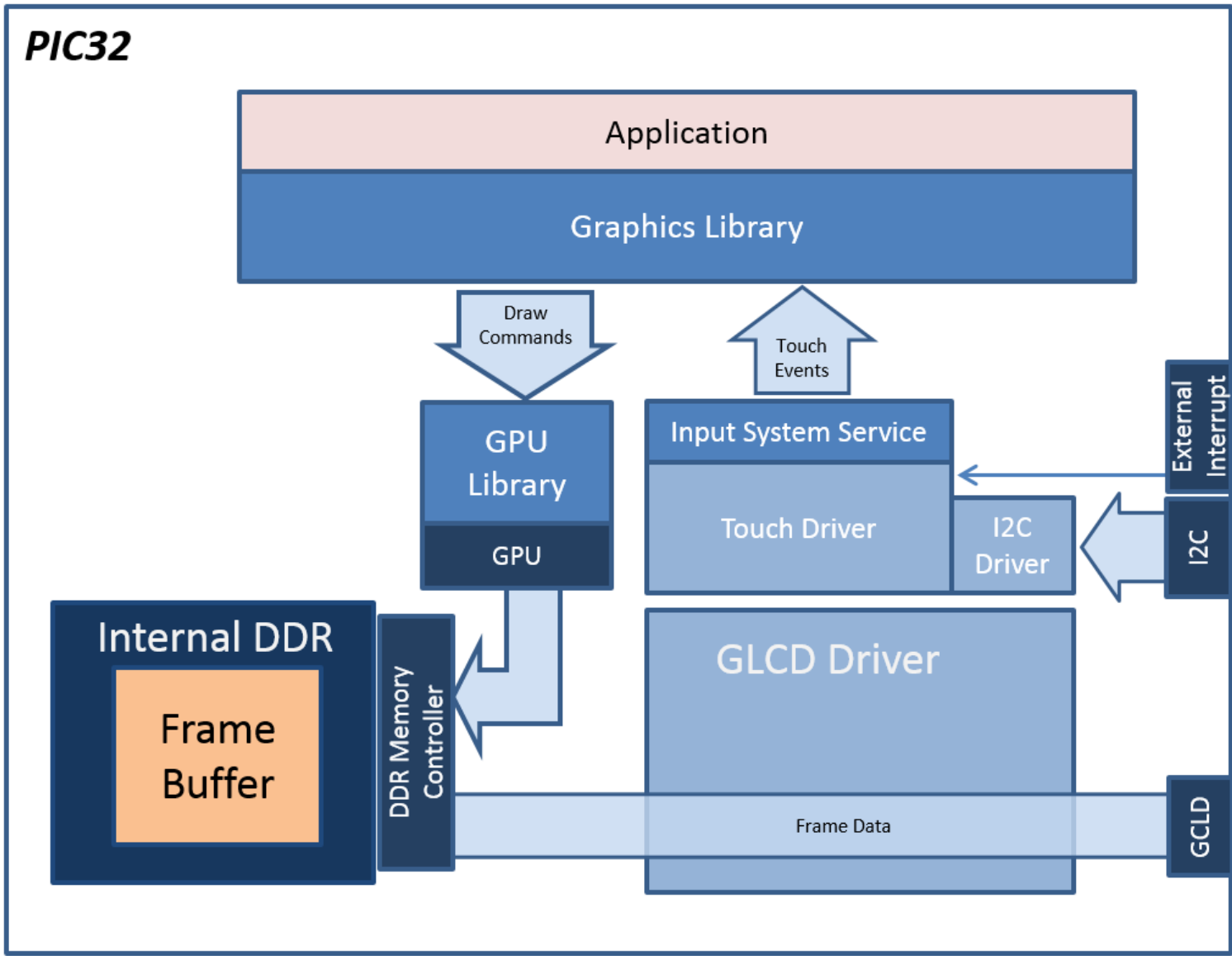
Description

This application demonstrates the Graphics Processing Unit (GPU) features on the PIC32MZ Embedded Graphics with Disabled DRAM (DA) Starter Kit hardware package. This SRAM version has 640K of accessible graphics memory. The application shows dynamic movement of image assets and basic motion control. The application performs time-based motion menu controls and User-Interface (UI) indicators. This demonstration displays the use of 640K SRAM by allocating 480x272x2 (16-bit) frame buffer, and a 480x272x2 asset staging area for fast run-time access to pre-rendered images. The demonstration shows that the GPU can be used in a low memory environment using 522k of internal frame buffer memory, and can produce appealing, low memory use GPU based applications. These are the key points:

- Internal SRAM package: a DDR is not required for graphics
- Multiple image staging with the use of sprites
- High color contrast images with drop shadow shading
- Modern UI
- Use of motion to show double buffering
- 13% of SRAM remains for non-graphics use, this is about 83K
- Use of MHGC screens for image asset. Managing image locations efficiently in memory can be an easy process

Architecture

The following figure shows the various software and hardware components used by this application.

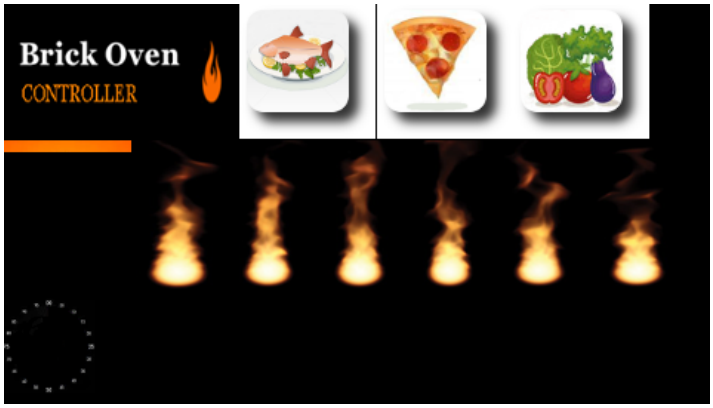


The aria_oven_controller application uses the MPLAB Harmony Graphics Library to render graphics to the display in a RGB565 color mode. The Graphics library draws the widgets and images to a 16-bit frame buffer which is stored in the internal SRAM of the PIC32MZ DA device. Using the Graphics Display Controller (GLCD), the PIC32MZ DA Display Driver continuously transfers frame data to the LCD display. During display refresh, the application will use the GPU to update the frame buffer using the nano2D software driver, which retrieves data from the staging area (source) for all image assets.

The demonstration contains five UI screens:

- Splash Screen: initial boot display
- Home Screen: controller demo idle state display
- Controller Screen: brick oven roasting control UI
- Info Screen: information and help
- Asset Layout Screen: image asset memory location and usage

Image assets are pre-rendered from internal Flash memory to SRAM on application start. Each image is accessed during runtime to create dynamic sprite affects and motion. The layout and location of images were positioned using the GFXLIB Graphics Composer Suite (MHGC) to produce accurate coordinate location and efficient memory use. Note that there exists space for additional images. The layout is shown in the following figure:



During run-time, the GPU is used to clear, colorize, blit and draw. The application uses `n2d_blit()` to block copy images from the SRAM staging area onto the refreshing frame buffer. The application uses `n2d_line()` to create the timer affect, and uses `n2d_fill` to clear the orange active menu button bar.

All motion effects make use of the `n2d_blit` (stretch/shrink) feature by setting the destination rectangle smaller or larger that the source rectangle (image). Image transparency is established by setting the background of the image to the display background. In addition, the application uses the Timer System Service and core timer clock ticks for image motion control.

The application is touch enabled. It features user touch input through the integrated touch screen on the display panel. Touch input from the touch controller goes through the I2C port, and the Input System Service acquires the touch input information from the Touch and I2C Drivers. The Input System Service sends touch events to the Graphics library, which processes these events and updates the frame data accordingly.

Demonstration Features:

- 16-Bit RGB 565 Frame buffer
- Image manipulation
- 480x272x2 Image staging memory space
- Modern drop-shadow, gradient image assets
- Internal memory: low memory frame buffer footprint 262K

Tools Setup:

- Enable Graphics Stack
- Enable Graphics Controller
  - GLCD
- Enable Graphics Processor
  - Nano2D

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the aria_oven_controller demonstration.

Description

To build this project, you must open the `aria_oven_controller.X` project in MPLAB X IDE, and then select the desired configuration. The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/gfx/aria_oven_controller`.

MPLAB X IDE Project

The following table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project	Location
aria_oven_controller	`<install-dir>/apps/gfx/aria_oven_controller/firmware`

MPLAB X IDE Project Configuration

The following table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config.`

Project Configuration Name	BSP Used	Description
pic32mz_da_sk_noddr_meb2_rgb_565	pic32mz_da_sk_noddr+meb2	Demonstration for the PIC32MZ Embedded Graphics with Disabled DRAM (DA) Starter Kit and the Multimedia Expansion Board II (MEB II).

Note: This application may contain custom code that is marked by the comments "`// START OF CUSTOM CODE ...`" and "`// END OF CUSTOM CODE`". If you use the MPLAB Harmony Configurator to regenerate the application code, use the "Prompt Merge For All Differences" merging strategy and do not remove or replace the custom code.

Configuring the Hardware

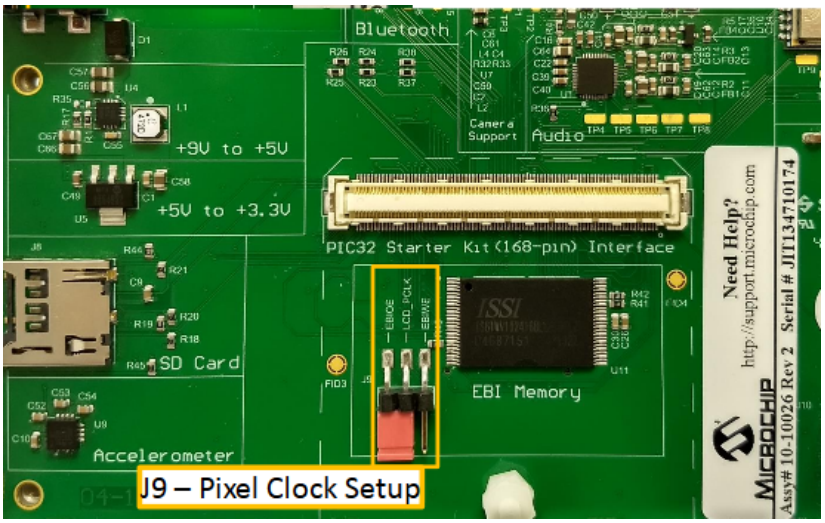
This section describes how to configure the supported hardware.

Description

PIC32MZ DA Starter Kit and MEB II

Configurations: pic32mz_ef_sk_intddr_meb2

- On the MEB II, the EBIWE and LCD_PCLK (J9) must be jumpered. A connection establishes the GLCD's pixel clock timing. The external SRAM memory on the board is disabled. The jumper (J9) is available on the bottom side of the board under the starter kit. The J9 jumper is located on the bottom of the MEB II board, beneath where the starter kit attaches to the board. Refer to the following image for the exact location
- Connect the PIC32MZ EF Starter Kit board to the MEB II board
- Power up the board by connecting the power adapter to J3 power connector on the MEB II board or a powered USB DEBUG port on the Starter Kit board



Running the Demonstration

This section provides information on how to run the demonstration.

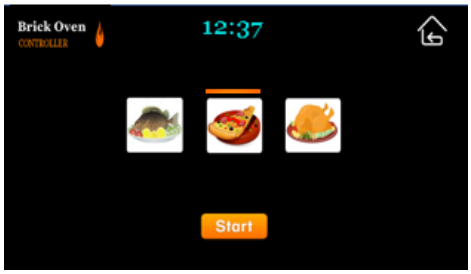
Description

On startup, the application will display a Splash Screen. After the splash screen completes, the Controller Screen is shown. In this screen, the Brick Oven Logo, Information button, and current time is displayed.

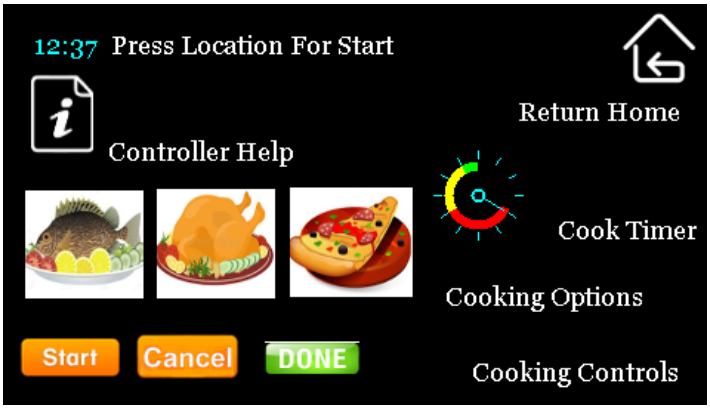


When the current time area is pressed, the Controller Screen is shown. On the display, the Brick Oven logo is reduced by 50% and positioned to the upper left corner of the screen. Three fire roasting bake options (fish, pizza, turkey) are displayed. When the user presses a button, an orange active bar is displayed over the button, and the start button is made visible as well as the pre-set cook time. When the Start button is pressed, the

flame animation will start and the timer will tick down to zero. When the time has finished, the Done button will be made visible, or the user is allowed to cancel before completion.



Each screen has a button that provides a transition to another screen.



aria_quickstart

This demonstration provides a touch-enabled starting point for the Aria Graphics Library using Low-Cost Controllerless technology.

Description

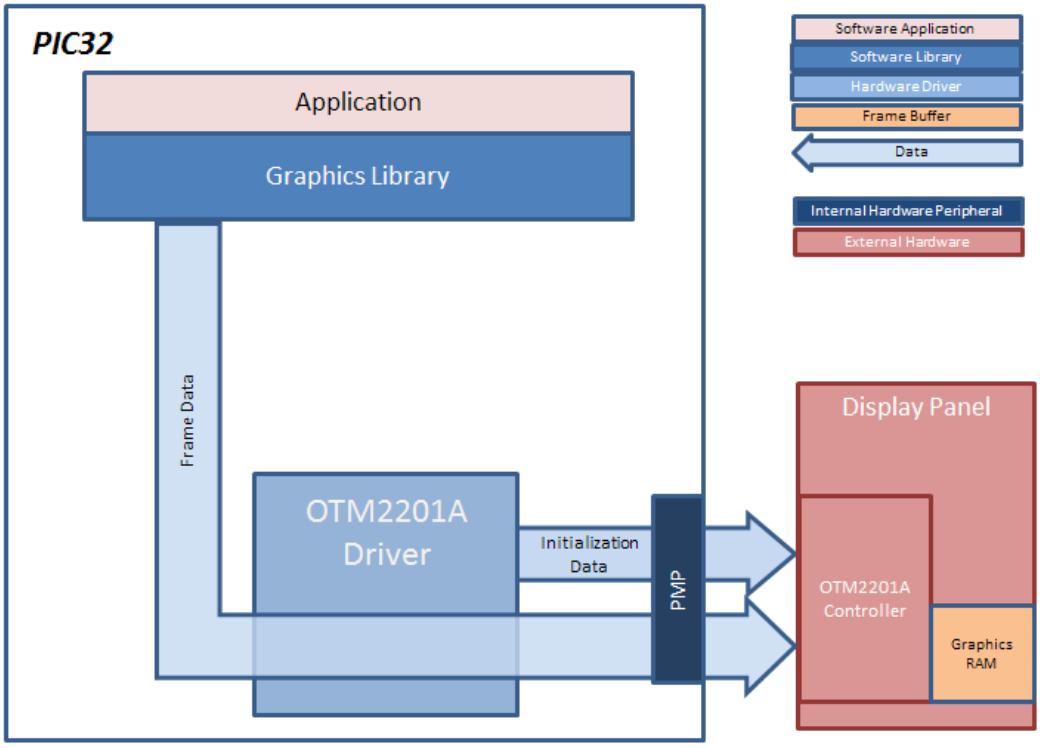
This demonstration serves as a preconfigured starting point for a touch-enabled application powered by the Aria User Interface Library. The demonstration has multiple configurations to demonstrate the graphics library running on different MCUs, displays, and display drivers.

Architecture

bt_audio_dk

For this configuration, the display has an on-board OTM2202a display controller. The PIC32MX uses a 16-bit parallel interface to initialize and communicate to the display peripheral. The aria_quickstart application uses the Graphics Library to render graphics to the display. As the Graphics Library draws the widgets and images, it uses the OTM2202a display controller driver APIs to write to the OTM2202a internal graphics RAM.

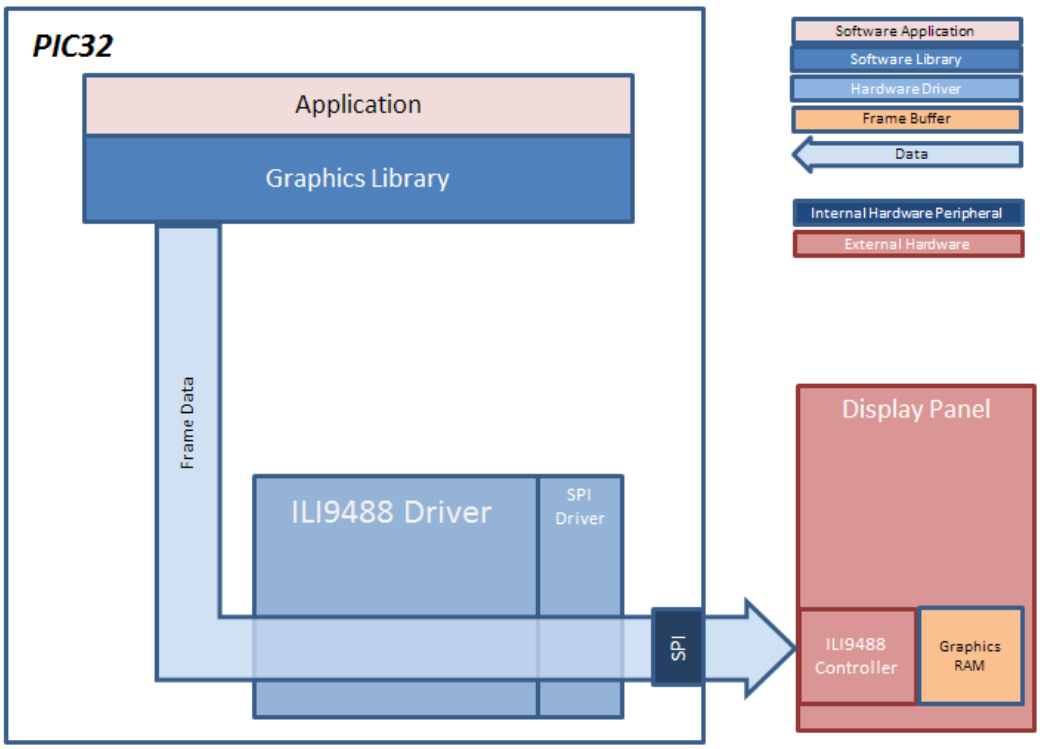
Note: Touch is not supported for this configuration.



pic32mz_ef_sk_xpro

For this configuration, the display has an on-board ILI9488 display controller that interfaces to the PIC32MZ MCU through the SPI. The aria_quickstart application uses the Graphics Library to render graphics to the display. As the Graphics Library draws the widgets and images, it uses the ILI9488 display controller driver APIs to write to the ILI9488 internal graphics RAM. The ILI9488 display driver uses the SPI peripheral library to talk to the ILI9488 device through the SPI peripheral port.

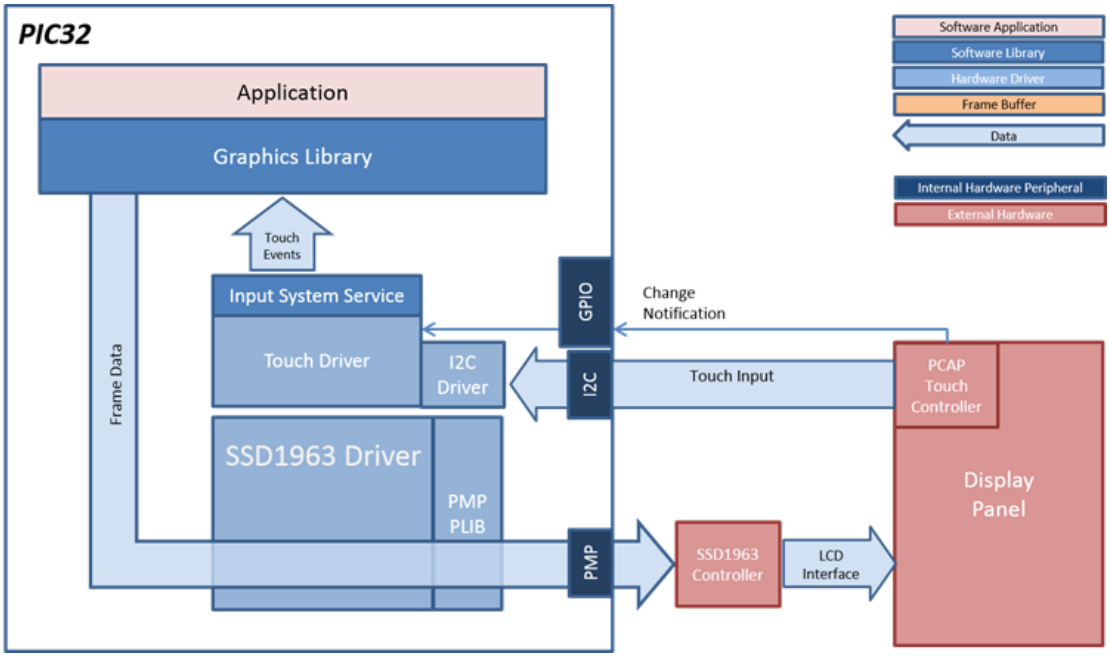
Note: Touch is not supported for these configurations.



pic32mk_gp_db_wqvga_mxt, and pic32mk_gp_db_wvga_mxt

For these configurations, the display is driven by a SSD1963 display controller (located on the underside of the PIC32MK GP Development Kit board) that interfaces to the PIC32MK MCU through the PMP. The MCU-SSD1963 interface consists of 16 PMP data pins (PMP0-15), Read Strobe pin, Write Strobe pin, Data/Command_Bar pin, and Chip_Select_Bar pin. The PMP peripheral is driven by PMP peripheral library calls. As the Graphics Library draws the widgets and images, the Graphics Library uses the SSD1963 display controller driver APIs to command pixel and rectangle draws. The SSD1963 interfaces to the display through a 50-pin ribbon cable. This interface has 24 lines for RGB color, pixel clock line, vertical and horizontal sync lines, and a data enable line.

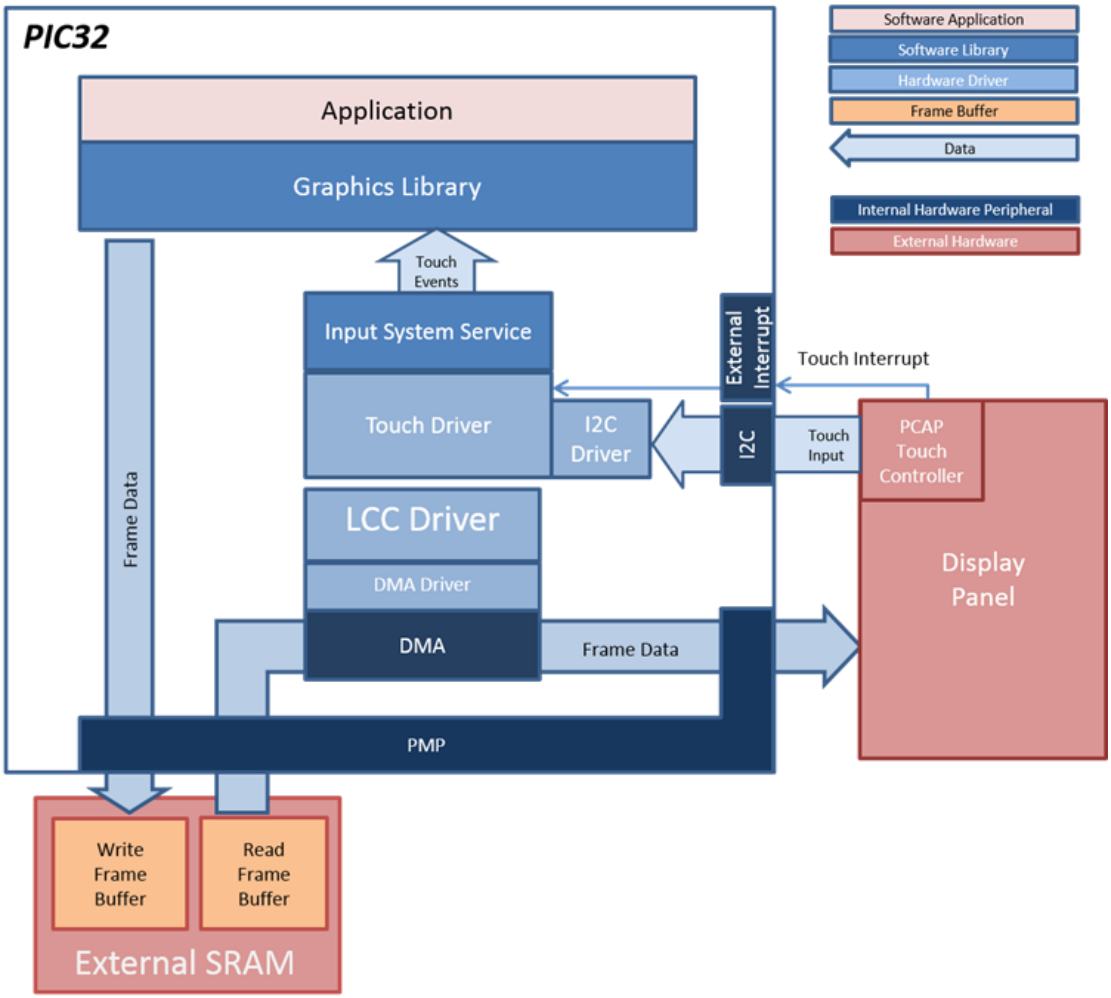
The application also features user touch input through the integrated touch screen on the display panel. Touch input from the touch controller goes through the I2C port, and the Input System Service acquires the touch input information from the touch and I2C drivers. The Input System Service sends touch events to the Graphics Library, which processes these events and updates the frame data accordingly. The MCU is alerted to touch events by the low assertion of the MAXTouch_Change event pin. This triggers a change notification ISR on the MCU, which then queues a read request to the touch driver.



pic32mx_pcap_db

For this configuration, the aria_quickstart application uses the Graphics Library to render graphics to the display. The Graphics Library draws the widgets and images to the write frame buffer that is stored in an external SRAM. Using the DMA, the Low-Cost Controllerless (LCC) display driver continuously transfers frame data from the read frame buffer out to the LCD display. The write and read frame buffers are swapped when the Graphics Library is done rendering to the write frame buffer and the LCC driver is done transferring the frame data from the read frame buffer.

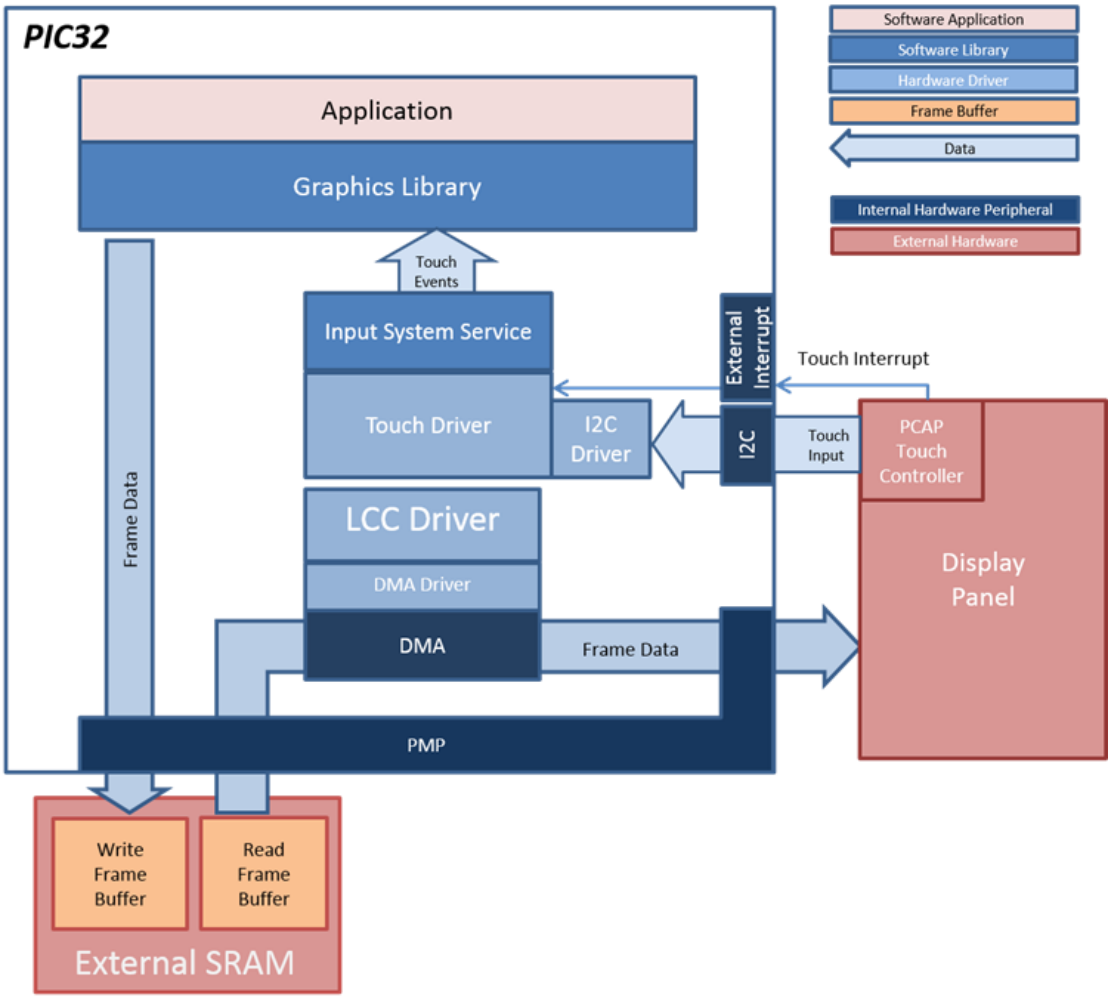
The application also features user touch input through the integrated touch screen on the display panel. Touch input from the touch controller goes through the I2C port, and the Input System Service acquires the touch input information from the touch and I2C drivers. The Input System Service sends touch events to the Graphics Library, which processes these events and updates the frame data accordingly.



pic32mx_usb_sk2_lcc_pictail_qvga

For this configuration, the aria_quickstart application uses the Graphics Library to render graphics to the display. The Graphics Library draws the widgets and images to the write frame buffer that is stored in an external SRAM. Using the DMA, the Low-Cost Controllerless (LCC) display driver continuously transfers frame data from the read frame buffer out to the LCD display. The write and read frame buffers are swapped when the Graphics Library is done rendering to the write frame buffer and the LCC driver is done transferring the frame data from the read frame buffer. The display has a built-in SSD1289 timing controller, which is managed by the SSD1289 Timing Controller (TCON) Driver through GPIO pins. The SSD1289 TCON Driver is initiated by the LCC Driver.

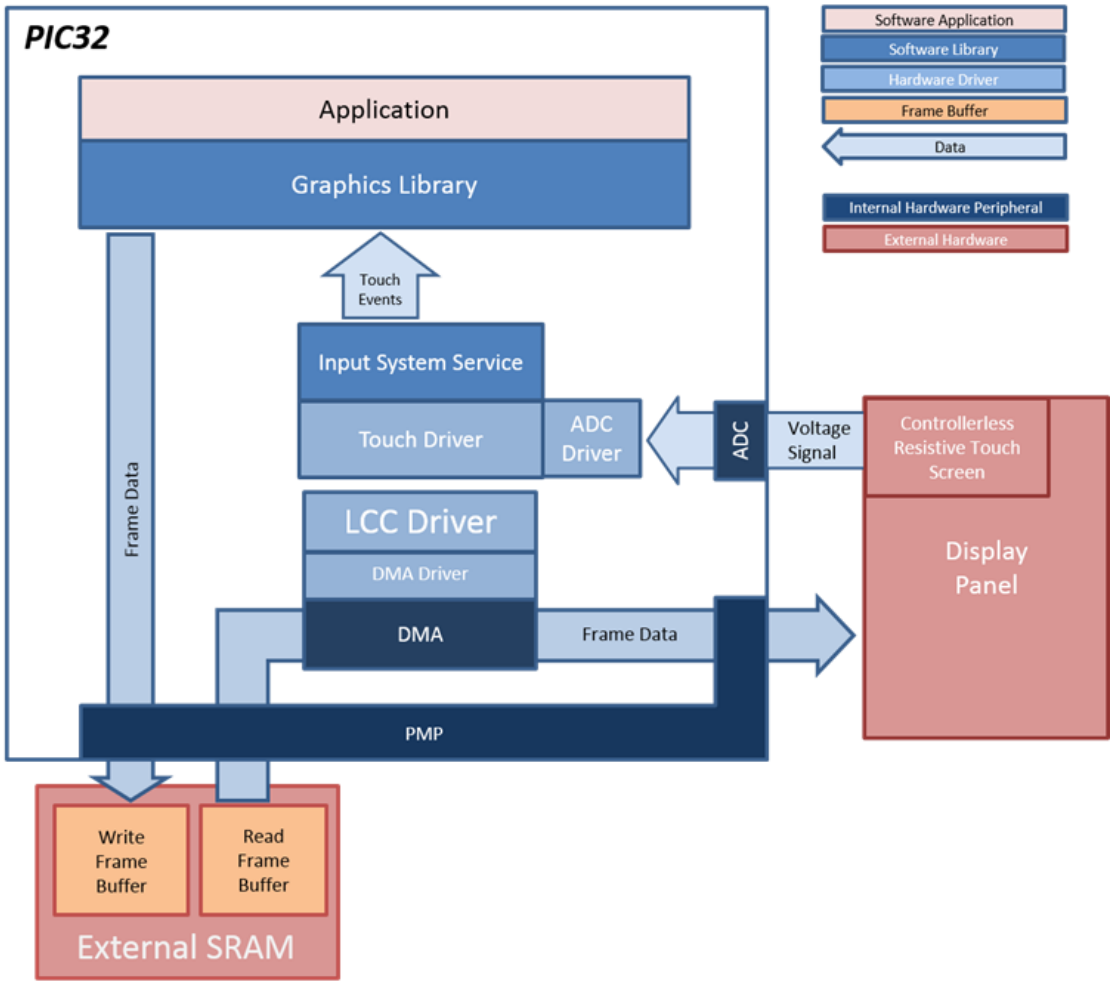
The application also features user touch input by measuring the voltages from the display panel. The voltage signal are measured by the ADC and interpreted by the touch driver, and the Input System Service acquires the touch input information from the touch driver. The Input System Service sends touch events to the Graphics Library, which processes these events and updates the frame data accordingly.



pic32mx_usb_sk2_lcc_pictail_wqvga

For these configurations, the aria_quickstart application uses the Graphics Library to render graphics to the display. The Graphics Library draws the widgets and images to the write frame buffer that is stored in an external SRAM. Using the DMA, the Low-Cost Controllerless (LCC) display driver continuously transfers frame data from the read frame buffer out to the LCD display. The write and read frame buffers are swapped when the Graphics Library is done rendering to the write frame buffer and the LCC driver is done transferring the frame data from the read frame buffer.

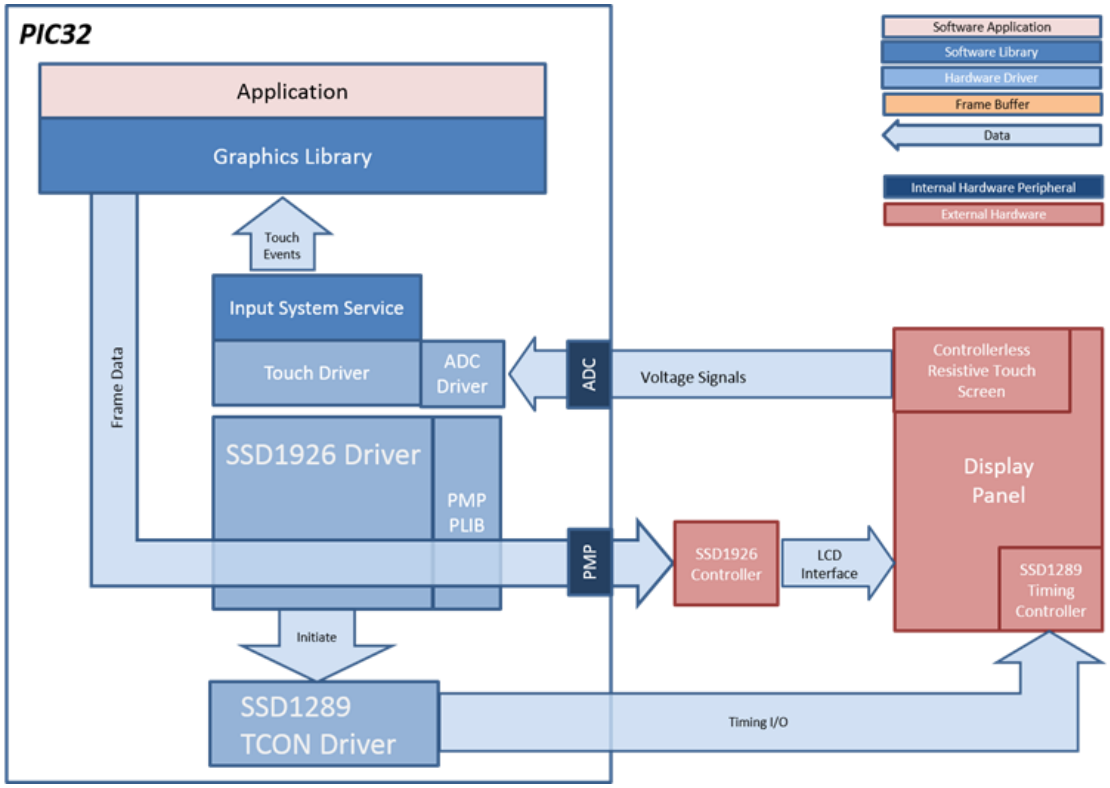
The application also features user touch input by measuring the voltages from the display panel. The voltage signal is measured by the ADC and interpreted by the touch driver, and the Input System Service acquires the touch input information from the touch driver. The Input System Service sends touch events to the Graphics Library, which processes these events and updates the frame data accordingly.



pic32mx_usb_sk2_meb

For this configuration, the display is driven by a SSD1926 display controller (located on the underside of the Multimedia Expansion Board II (MEB II)) that interfaces to the PIC32MX MCU through the PMP. The MCU-SSD1926 interface consists of 16 PMP data pins (PMP0-15), Read Strobe pin, Write Strobe pin, Data/Command_Bar pin, and Chip_Select_Bar pin. The PMP peripheral is driven by PMP PLIB calls. As the Graphics Library draws the widgets and images, the graphics library uses the SSD1926 display controller driver APIs to command pixel and rectangle draws. The display has a built-in SSD1289 timing controller, which is managed by the SSD1289 Timing Controller (TCON) Driver through GPIO pins. The SSD1289 TCON Driver is initiated by the SSD1926 display graphics controller driver.

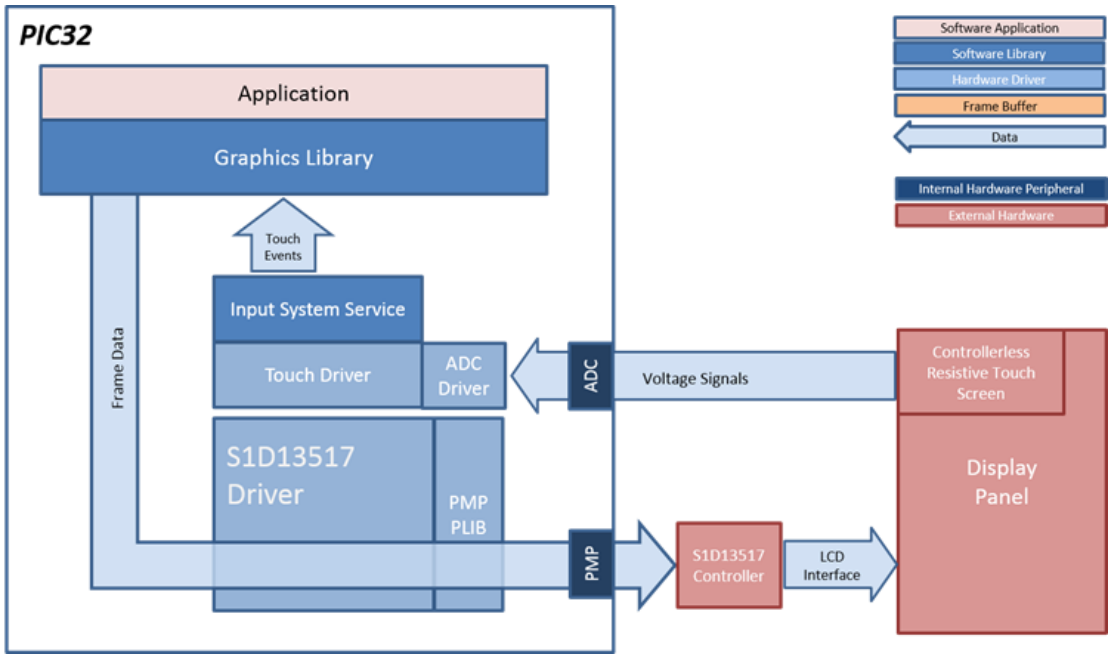
The application also features user touch input by measuring the voltages from the display panel. The voltage signal is measured by the ADC and interpreted by the touch driver, and the Input System Service acquires the touch input information from the touch driver. The Input System Service sends touch events to the Graphics Library, which processes these events and updates the frame data accordingly.



pic32mx_usb_sk2_s1d_pictail_wqvga, pic32mx_usb_sk2_s1d_pictail_wvga

For this configuration, the display is driven by a EPSON S1D13517 display controller (located on the Graphics Controller PICtail Plus Epson S1D13517 Daughter Board) that interfaces to the PIC32MX MCU through the PMP. The MCU-S1D13517 interface consists of 16 PMP data pins (PMP0-15), Read Strobe pin, Write Strobe pin, Data/Command_Bar pin, and Chip_Select_Bar pin. The PMP peripheral is driven by PMP PLIB calls. As the Graphics Library draws the widgets and images, the graphics library uses the S1D13517 display controller driver APIs to command pixel and rectangle draws.

The application also features user touch input by measuring the voltages from the display panel. The voltage signal is measured by the ADC and interpreted by the touch driver, and the Input System Service acquires the touch input information from the touch driver. The Input System Service sends touch events to the Graphics Library, which processes these events and updates the frame data accordingly.

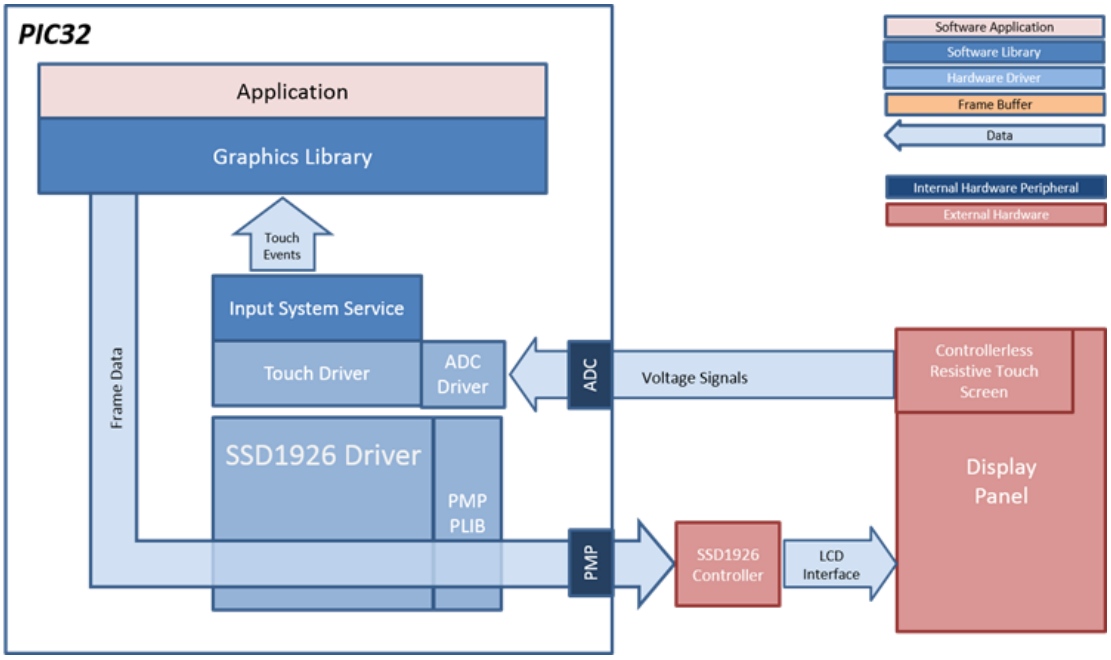


pic32mx_usb_sk2_ssd_pictail_qvga

For this configuration, the display is driven by a SSD1926 display controller (located on the Graphics LCD Controller PICtail Plus SSD1926 Daughter Board) that interfaces to the PIC32MX through the PMP. The MCU-SSD1926 interface consists of 16 PMP data pins (PMP0-15), Read Strobe pin, Write Strobe pin, Data/Command_Bar pin, and Chip_Select_Bar pin. The PMP peripheral is driven by PMP peripheral library calls. As the Graphics Library draws the widgets and images, the graphics library uses the SSD1926 display controller driver APIs to command pixel and

rectangle draws. The display has a built-in SSD1289 timing controller, which is managed by the SSD1289 timing controller (TCON) Driver through GPIO pins. The SSD1289 TCON driver is initiated by the SSD1926 display graphics controller driver.

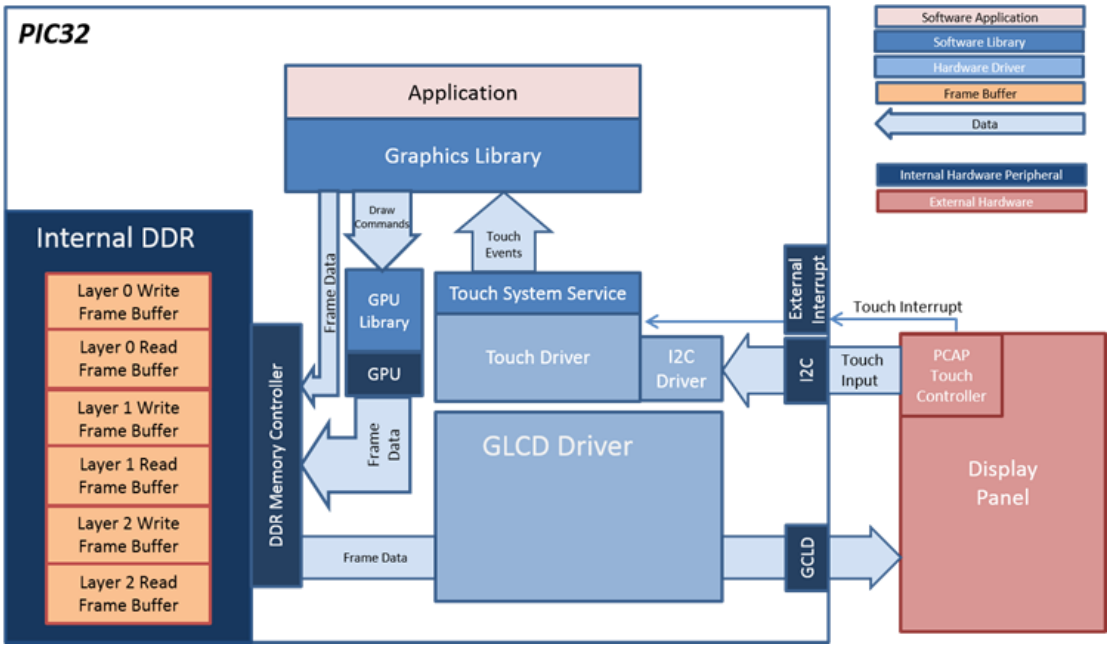
The application also features user touch input by measuring the voltages from the display panel. The voltage signal is measured by the ADC and interpreted by the touch driver, and the Input System Service acquires the touch input information from the touch driver. The Input System Service sends touch events to the Graphics Library which, processes these events and updates the frame data accordingly.



pic32mz_da_sk_extddr_meb2, and pic32mz_da_sk_extddr_meb2_wvga

For these configurations, the aria_quickstart application uses the Graphics Library to render graphics to the display. The Graphics Library draws the widgets and images to the write frame buffer that is stored in an external DDR2. Through the DDR2 Memory Controller, the GLCD display controller peripheral continuously transfers frame data from the frame buffer onto to the LCD display. The write and read frame buffers are swapped when the Graphics Library is done rendering to the write frame buffer and GLCD driver signals the GLCD peripheral to change its read location.

The application also features user touch input through the integrated touch screen on the display panel. Touch input from the touch controller travels through the I2C port, and the Input System Service acquires the touch input information from the touch and I2C drivers. The Input System Service sends touch events to the Graphics Library, which processes these events and updates the frame data accordingly.

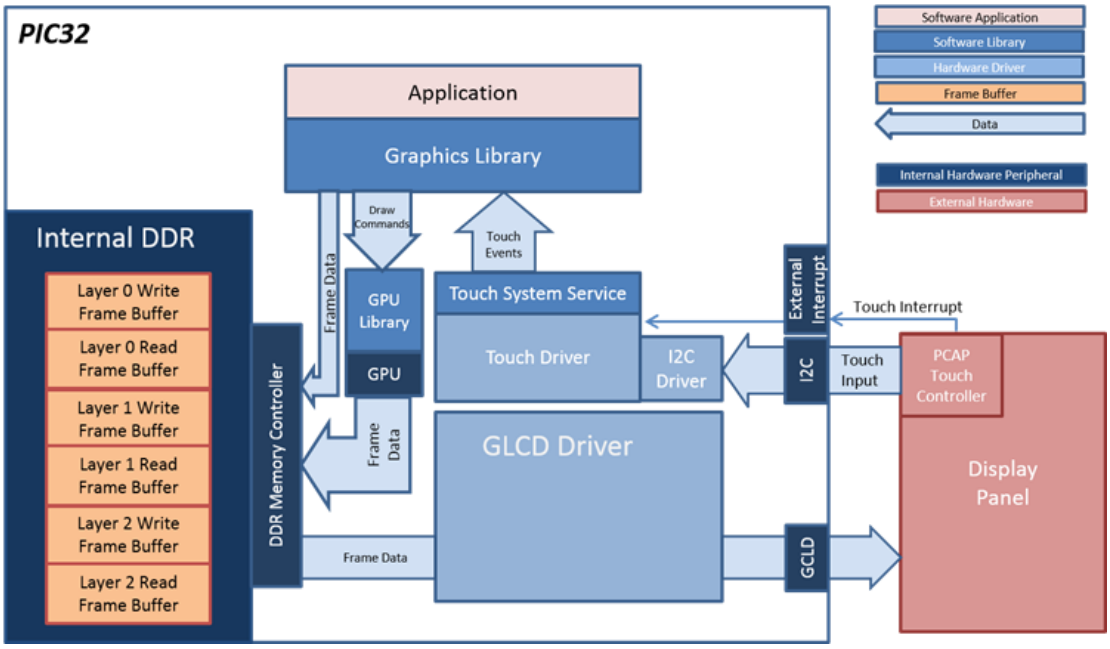


pic32mx_da_sk_intddr_meb2, and pic32mz_da_sk_intddr_meb2_wvga

For these configurations, the application uses the Graphics Library to render graphics to the display. The Graphics library passes draw commands into the GPU Library, which in turn draws the widgets and images to the three individual write frame buffers (one for each layer) that are stored in an internal DDR2. Via the DDR2 Memory Controller, the GLCD display controller peripheral continuously transfers frame data from all three read buffers onto the LCD display. The write and read frame buffer pairs are swapped independently as required when the Graphics Library is done

rendering to the write frame buffer and GLCD driver signals the GLCD peripheral to change its read location.

The application also features user touch input through the integrated touch screen on the display panel. Touch input from the touch controller travels through the I2C port, and the Input System Service acquires the touch input information from the touch and I2C drivers. The Input System Service sends touch events to the Graphics library which processes these events and updates the frame data accordingly.



pic32mz_da_sk_noddr_meb2_rgb565

For this configuration, the aria_quickstart application uses the Graphics Library to render graphics to the display. The Graphics Library draws the widgets and images to the write frame buffer that is stored on the internal SRAM.

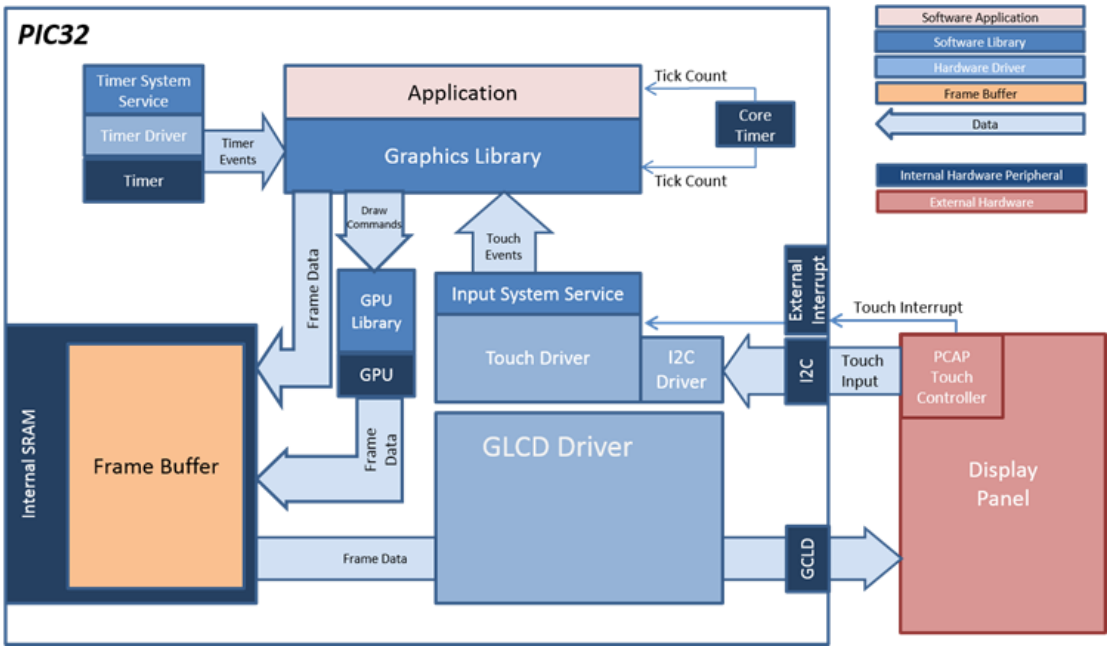
Since the color format is set to RGB565 (16-bit), a frame buffer requires 261120 bytes of memory (480 wide x 272 height x 2 bytes per pixel). A double-buffered configuration is possible with 640 kilobytes of internal SRAM in the MZ DA device.

Some of the frame buffer updates come directly from the Graphics Library while others are accelerated by the GPU.

Using internal memory directly from the frame buffer, the GLCD display controller peripheral continuously transfers frame data from the read frame buffer onto to the LCD display.

The write and read frame buffers are swapped when the Graphics Library is done rendering to the write frame buffer and GLCD driver signals the GLCD peripheral to change its read location.

The application also features user touch input through the integrated touch screen on the display panel. Touch input from the touch controller travels through the I2C port, and the Input System Service acquires the touch input information from the touch and I2C drivers. The Input System Service sends touch events to the Graphics Library, which processes these events and updates the frame data accordingly.



pic32mz_da_sk_noddr_meb2_rgba8888

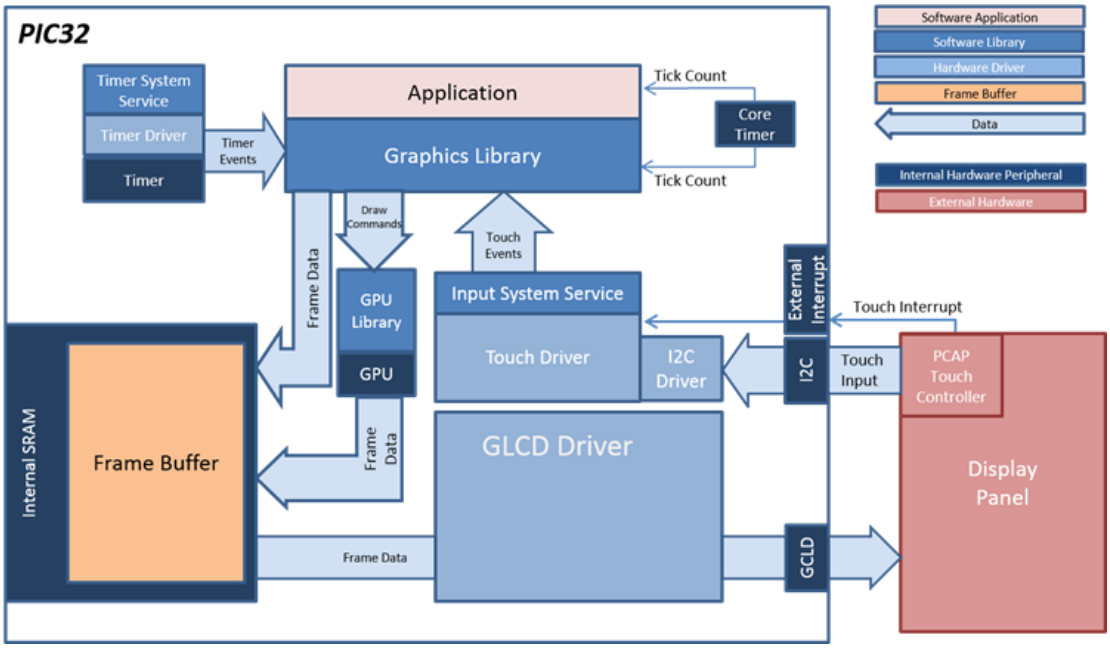
For this configuration, the aria_quickstart application uses the Graphics Library to render graphics to the display. The Graphics Library draws the widgets and images to the frame buffer that is stored on the internal SRAM.

The color format is set to RGBA8888 (24-bit color with an 8 bit alpha channel), a frame buffer requires 522240 bytes of memory (480 wide x 272 height x 4 bytes per pixel). A single frame buffer configuration is possible within 640 kilobytes of internal SRAM on the MZ DA device.

The Graphics Library draws the widgets and images to the frame buffer that is stored on the internal SRAM. Some of the frame buffer updates comes directly from the Graphics Library while others are accelerated by the GPU.

Using internal memory directly from the frame buffer, the GLCD display controller peripheral continuously transfers frame data from the read frame buffer onto to the LCD display.

The application also features user touch input through the integrated touch screen on the display panel. Touch input from the touch controller goes through the I2C port, and the Input System Service acquires the touch input information from the touch and I2C drivers. The Input System Service sends touch events to the Graphics Library, which processes these events and updates the frame data accordingly.



pic32mz_da_sk_noddr_meb2_wvga_lut8

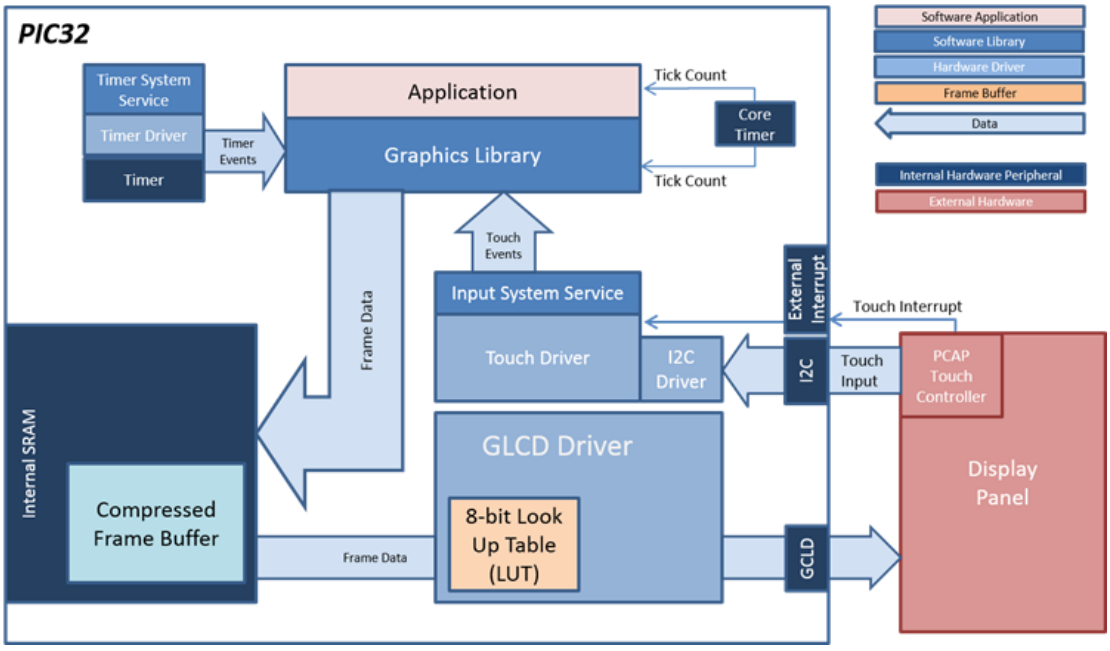
For this configuration, the aria_quickstart application uses the Graphics Library to render graphics to the display. The Graphics Library draws the widgets and images to a compressed frame buffer that is stored on the internal SRAM.

The application is condensed to 256 colors. Each color is presented as an 8-bit index value in the Look-Up Table (LUT). The LUT is created using the Global Palette feature within Harmony Graphics Composer. Each of these 256 colors is RGB888 (24-bit color). Because of this arrangement, a frame buffer requires 384000 bytes of memory (800 wide x 480 height x 1 byte per pixel). A single frame buffer configuration is possible within the 640 kilobytes of internal SRAM on the MZ DA device.

The Graphics Library draws the widgets and images and remaps as an index value into the compressed frame buffer.

At launch, the 256-color LUT is loaded into the GLCD's registers. Using the LUT in its register to translate index value into RGB888 color, the GLCD display controller peripheral continuously transfers frame data from the frame buffer onto to the LCD display.

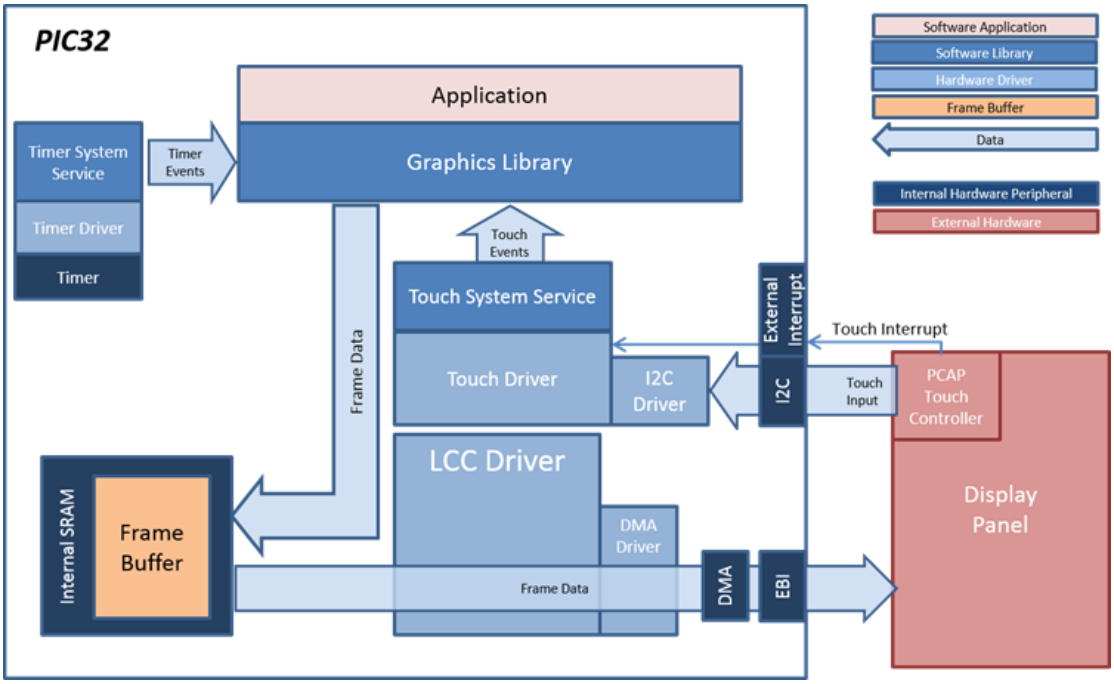
The application also features user touch input through the integrated touch screen on the display panel. Touch input from the touch controller goes through the I2C port, and the Input System Service acquires the touch input information from the touch and I2C drivers. The Input System Service sends touch events to the Graphics Library, which processes these events and updates the frame data accordingly.



pic32mz_ef_sk_meb2

For this configuration, the aria_quickstart application uses the Graphics Library to render graphics to the display. The Graphics Library draws the widgets and images to the write frame buffer that is stored in an internal SRAM. Using the DMA, the Low-Cost Controllerless (LCC) display driver continuously transfers frame data from the frame buffer out to the LCD display.

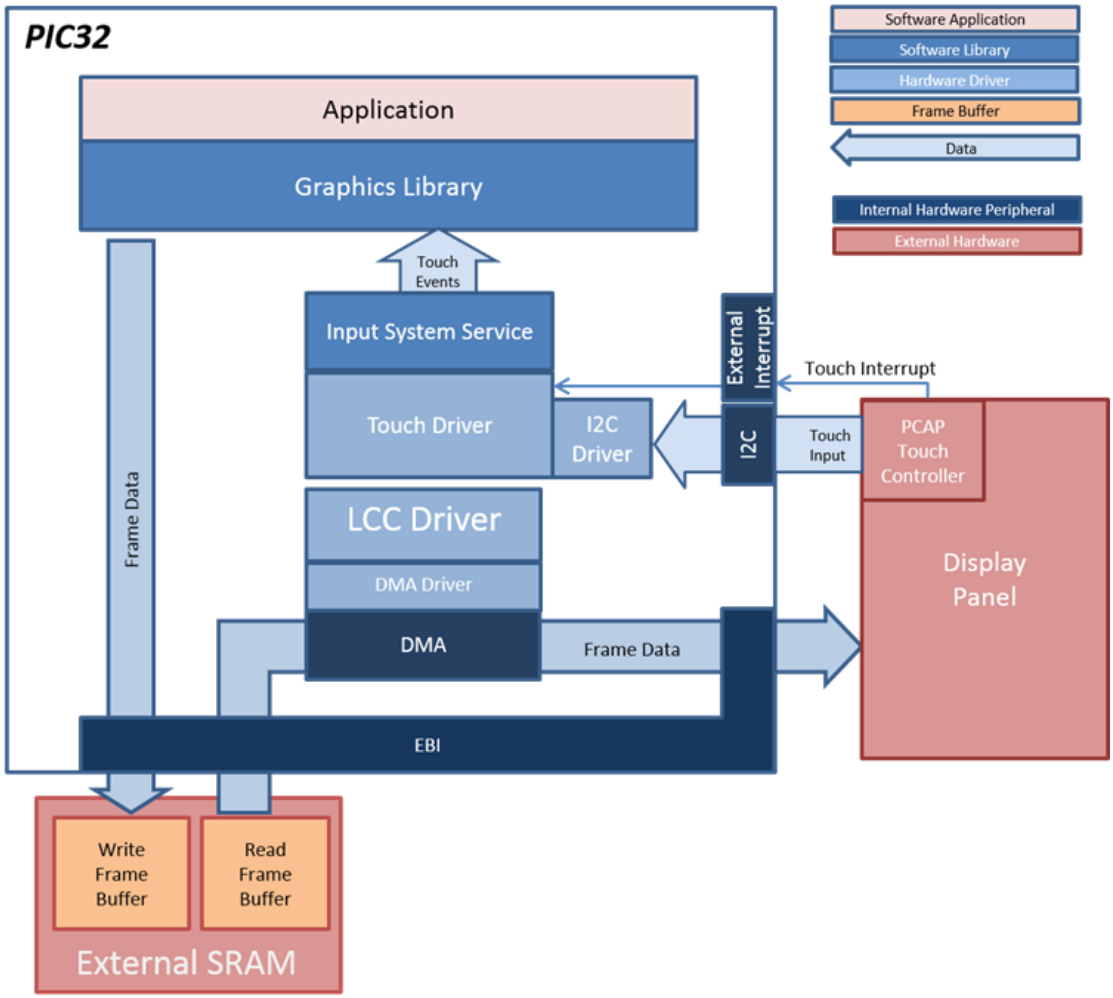
The application also features user touch input through the integrated touch screen on the display panel. Touch input from the touch controller travels through the I2C port, and the Input System Service acquires the touch input information from the touch and I2C drivers. The Input System Service sends touch events to the Graphics Library, which processes these events and updates the frame data accordingly.



pic32mz_ef_sk_meb2_ext

For this configuration, the aria_quickstart application uses the Graphics Library to render graphics to the display. The Graphics Library draws the widgets and images to the write frame buffer that is stored in an external SRAM. Using the DMA, the Low-Cost Controllerless (LCC) display driver continuously transfers frame data from the read frame buffer out to the LCD display. The write and read frame buffers are swapped when the Graphics Library is done rendering to the write frame buffer and the LCC driver is done transferring the frame data from the read frame buffer.

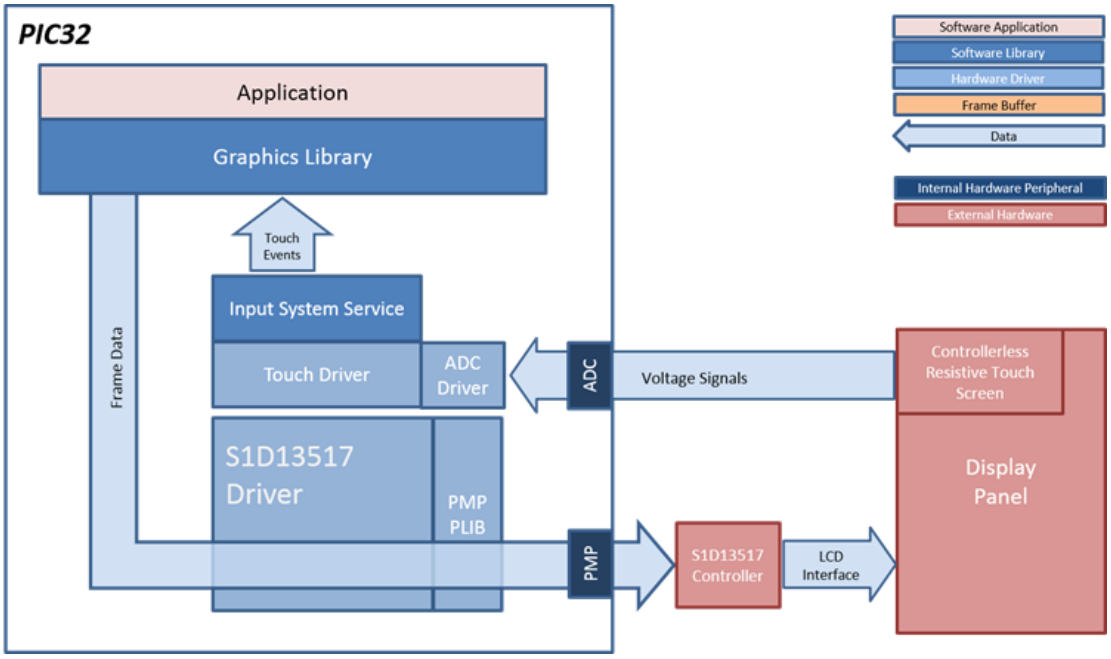
The application also features user touch input through the integrated touch screen on the display panel. Touch input from the touch controller travels through the I2C port, and the Input System Service acquires the touch input information from the touch and I2C drivers. The Input System Service sends touch events to the Graphics Library, which processes these events and updates the frame data accordingly.



pic32mz_ef_sk_s1d_pictail_wqvga

For this configuration, the display is driven by an EPSON S1D13517 display controller (located on the Graphics LCD Controller PICtail Plus SSD1926 Daughter Board) that interfaces to the PIC32MZ through the PMP. The MCU-S1D13517 interface consists of 16 PMP data pins (PMP0-15), Read Strobe pin, Write Strobe pin, Data/Command_Bar pin, and Chip_Select_Bar pin. The PMP peripheral is driven by PMP peripheral library calls. As the Graphics Library draws the widgets and images, the graphics library uses the S1D13517 display controller driver APIs to command pixel and rectangle draws.

The application also features user touch input by measuring the voltages from the display panel. The voltage signal are measured by the ADC and interpreted by the touch driver, and the Input System Service acquires the touch input information from the touch driver. The Input System Service sends touch events to the Graphics Library, which processes these events and updates the frame data accordingly.



Demonstration Features

- RGBA8888 Color Mode (PIC32MZ DA)
- RGB565 Color Mode (PIC32MZ EF)
- Internal Flash Asset Storage
- 8-bit Palette Compressed Images
- Run-length Encoded Images
- 1bpp Font Glyph Decoding
- ASCII Character Encoding
- Single Display Layer
- Display Edge Clipping
- User Interface Tree Hierarchy
- Parent/Child Area Clipping
- UI Dragging and Touch (with the following configuration exceptions):
  - pic32mz_ef_sk_xpro
  - bt_audio_dk
- Button Widget
- Label Widget
- UI Widget Bounds that Exceed Display Area
- Double Buffering (PIC32MZ DA)
- Vertical Sync Swapping (PIC32MZ DA)

Tools Setup Differences

All configurations

- “Use Graphics Stack” is selected in MHC. This enables the graphics and touch libraries and drivers for the BSP.
- The heap size is set to 102400 bytes to ensure enough memory for dynamic allocations. This is done in the MPLAB Harmony Configurator (MHC) by setting the size in *Device & Project Configuration >Project Configuration >XC32 (Global Options) >xc32-ld >General*.

pic32mk_gp_db_wqvga_mxt

pic32mk_gp_db_wvga_mxt

There are custom code blocks identified by `// CUSTOM CODE – DO NOT DELETE …` `// END OF CUSTOM CODE` comments that represent modifications to the code outside of MHC. These code blocks are necessary for correct operation of the demonstration under these configurations.

Important! When regenerating the project from within MHC use the "Prompt Merge For All Differences" merging strategy and do not change these custom code blocks.

File: `system_config.h`

Custom Code: SCL/SDA port and pin assignments for the I2C bit-banged driver.

Custom Code: //#define GFX_ASSERT_ENABLE // Enable asserts in GFX

File: `sytem_init.c`

Custom Code: Release of RESET to the maXTouch Driver

Custom Code: Setup of interrupt priority and subpriority for change notification ISR

Custom Code: SYS_CONSOLE_Write announcing that application has been initialized

Custom Code: Commented out tests for assert and exception handling

File: `system_interrupt.c`

Custom Code: Customization of ISRs to support monitoring touch events

Custom Code: Setup of change notification ISR to support touch events

Important! There are custom code blocks identified by // CUSTOM CODE – DO NOT DELETE … // END OF CUSTOM CODE comments that represent modifications to the code outside of MHC. These code blocks are necessary for correct operation of the demonstration under these configurations. When regenerating the project from within MHC use the "Prompt Merge For All Differences" merging strategy and do not change these custom code blocks.

pic32mx_pcap_db

Due to the pin-out, the I2C2 module is selected to support the touch operation.

pic32mx_usb_sk2_lcc_pictial_qvga

The touch driver used for this configuration is the Touch ADC Driver. The ADC Driver is configured by default when the ADC Touch Driver is enabled. For details on the ADC Touch Driver, please refer to ADC Touch Driver Library and ADC Driver Library.

The voltage measurement pins are configured by default by the BSP and can be inspected in the Pin Manager. The ADC Touch Driver is programmed to use exact pin names.

The QVGA display is fitted with a SSD1289 Timing Controller module. The SSD1289 driver is enabled in the LCC mode to support this project.

pic32mx_usb_sk2_lcc_pictial_wqvga

The touch driver used for this configuration is the Touch ADC Driver. The ADC Driver is configured by default when the ADC Touch Driver is enabled. For details on the ADC Touch Driver, please refer to ADC Touch Driver Library and ADC Driver Library.

By default, the voltage measurement pins are configured by the BSP and can be inspected in the Pin Manager. The ADC Touch Driver is programmed to use exact pin names as shown.

pic32mx_usb_sk2_meb

The touch driver used for this configuration is the ADC Touch Driver. The ADC Driver is configured by default when the ADC Touch Driver is enabled. For details on the ADC Touch Driver, please refer to ADC Touch Driver Library and ADC Driver Library.

By default, the voltage measurement pins are configured by the BSP and can be inspected in the Pin Manager. The ADC Touch Driver is programmed to use exact pin names.

The QVGA display is fitted with a SSD1289 Timing Controller module. The SSD128 Driver is enabled in the SSD1926 mode to support this project.

The PMP driver is enabled to support the SSD1926 Graphics Display Controller Driver.

pic32mx_usb_sk2_s1d_pictail_wqvga, and pic32mx_usb_sk2_s1d_pictail_wvga

The touch driver used for this configuration is the ADC Touch Driver. The ADC Driver is configured by default when the ADC Touch Driver is enabled. For details on the ADC Touch Driver, please refer to ADC Touch Driver Library and ADC Driver Library.

By default, the voltage measurement pins are configured by the BSP and can be inspected in the Pin Manager. The ADC Touch Driver is programmed to use the exact pin names as shown.

The PMP driver is enabled to support the Epson S1D13517 Graphics Display Controller driver.

pic32mx_usb_sk2_ssd_pictail_qvga

The touch driver used for this configuration is the ADC Touch Driver. The ADC Driver is configured by default when the ADC Touch Driver is enabled. For details on the ADC Touch Driver, please refer to ADC Touch Driver Library and ADC Driver Library.

By default, the voltage measurement pins are configured by the BSP and can be inspected in the Pin Manager. The ADC Touch Driver is programmed to use exact pin names.

The QVGA display is fitted with a SSD1289 Timing Controller module. The SSD1289 driver is enabled in SSD1926 Mode to support this project.

The PMP driver is enabled to support the SSD1926 Graphics Display Controller driver.

pic32mz_da_sk_extddr_meb2

pic32mz_da_sk_extddr_meb2_wvga

For these two configurations, the Memory System Service is enabled to support the external DDR.

On-Die Termination for Write is enabled.

Reference Clock 5 is enabled to provide a master clock signal to the GLCD peripheral.

Finally, the MPLL is set up with an output clock of 200 MHz to drive the DDR2 memory.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Aria Quick Start demonstration.

Description

To build this project, you must open the `aria_quickstart.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/gfx/aria_quickstart`.



Important!

This application contains custom code that is marked by the comments `// START OF CUSTOM CODE ...` and `// END OF CUSTOM CODE`. If you use the MPLAB Harmony Configurator (MHC) to regenerate the application code, do not remove or replace the custom code.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`aria_quickstart.X`	`<install-dir>/apps/gfx/aria_quickstart/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
bt_audio_dk	bt_audio_dk	Demonstration for the Bluetooth Audio Development Board.
pic32mk_gp_db_wqvga_mxt	pic32mk_gp_db+wqvga_mxt	Demonstration for the PIC32MK GP Development Kit with a High-Performance 4.3” 480x272 WQVGA Display Module with maXTouch.
pic32mk_gp_db_wvga_mxt	pic32mk_gp_db+wvga_mxt	Demonstration for the PIC32MK GP Development Kit with a High-Performance (5”) WVGA Display Module with maXTouch.
pic32mx_pcap_db	pic32mx_pcap_db	Demonstration for the PIC32 GUI Development Board with Projected Capacitive Touch Board.
pic32mx_usb_sk2_lcc_pictail_qvga	pic32mx_usb_sk2+lcc_pictail+qvga	Demonstration for the PIC32MX USB Starter Kit II plus the LCC Graphics PICtail Plus Daughter Board with the Graphics Display Truly 3.2" 320x240 Board.
pic32mx_usb_sk2_lcc_pictail_wqvga	pic32mx_usb_sk2+lcc_pictail+wqvga	Demonstration for the PIC32MX USB Starter Kit II plus the LCC Graphics PICtail Plus Daughter Board with the Graphics Display Powertip 4.3” 480x272 display.
pic32mx_usb_sk2_meb	pic32mx_usb_sk2+meb	Demonstration for the PIC32 MX USB Starter Kit II plus Multimedia Expansion Board (MEB).
pic32mx_usb_sk2_s1d_pictail_wqvga	pic32mx_usb_sk2+s1d_pictail+wqvga	Demonstration for the PIC32MX USB Starter Kit II plus the Graphics PICtail Plus Epson S1D13517 Daughter Board with a Graphics Display Powertip 4.3" 480x272 Board.
pic32mx_usb_sk2+s1d_pictail+wvga	pic32mx_usb_sk2+s1d_pictail+wvga	PIC32 USB Starter Kit II plus the Graphics Controller PICtail Plus Epson S1D13517 Daughter Board with Graphics Display Truly 7" 800x400 Board.
pic32mx_usb_sk2_ssd_pictail_qvga	pic32mx_usb_sk2+ssd_pictail+qvga	Demonstration for the PIC32MX USB Starter Kit II plus the Graphics LCD Controller PICtail Plus SSD1926 Board with a Graphics Display Truly 3.2" 320x240 Board.

pic32mz_da_sk_extddr_meb2	pic32mz_da_sk_extddr+meb2	Demonstration for the PIC32MZ Embedded Graphics with External DRAM (DA) Starter Kit plus Multimedia Expansion Board II (MEB II).
pic32mz_da_sk_extddr_meb2_wvga	pic32mz_da_sk_extddr+meb2+wvga	Demonstration for the PIC32MZ Embedded Graphics with External DRAM (DA) Starter Kit Memory plus the Multimedia Expansion Board II (MEB II) with a High-Performance (5") WVGA Display Module with maXTouch.
pic32mz_ef_sk_meb2	pic32mz_ef_sk+meb2	Demonstration for the PIC32MZ EF Starter Kit plus the Multimedia Expansion Board II (MEB II) (Internal SRAM Frame Buffer).
pic32mz_ef_sk_meb2_ext	pic32mz_ef_sk+meb2	Demonstration for the PIC32MZ EF Starter Kit plus the Multimedia Expansion Board II (MEB II) (External SRAM Frame Buffer).
pic32mz_ef_sk_s1d_pictail_wqvga	pic32mz_ef_sk+s1d_pictail+wqvga	Demonstration for the PIC32MZ EF Starter Kit plus the Graphics PICtail Plus Epson S1D13517 Daughter Board with a Graphics Display Powertip 4.3" 480x272 Board.
pic32mz_ef_sk_xpro	pic32mz_ef_sk+maxtouch_xplained_pro_3_5	Demonstration for the PIC32MZ EF Starter Kit with 3.5” Xplained Pro board connected through SPI.
pic32mz_da_sk_intddr_meb2	pic32mz_da_sk_intddr+meb2	Demonstration for the PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit and the Multimedia Expansion Board II (MEB II).
pic32mz_da_sk_intddr_meb2_wvga	pic32mz_da_sk_intddr+meb2+wvga	Demonstration for the PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit plus Multimedia Expansion Board II (MEB II) with a High-Performance 5” WVGA Display Module with maXTouch.
pic32mz_da_sk_noddr_meb2_rgb565	pic32mz_da_sk_noddr+meb	Demonstration for the PIC32MZ Embedded Graphics with Disabled DRAM (DA) Starter Kit plus Multimedia Expansion Board II (MEB II) in RGB_565 Color Mode.
pic32mz_da_sk_noddr_meb2_rgba8888	pic32mz_da_sk_noddr+meb	Demonstration for the PIC32MZ Embedded Graphics with Disabled DRAM (DA) Starter Kit plus Multimedia Expansion Board II (MEB II) in RGBA_8888 Color Mode.
pic32mz_da_sk_noddr_meb2_wvga_lut8	pic32mz_da_sk_noddr_meb2_wvga	Demonstration for the PIC32MZ Embedded Graphics with Disabled DRAM (DA) Starter Kit plus Multimedia Expansion Board II (MEB II) with a High-Performance 5” WVGA Display Module with maXTouch using Global 8-Bit Palette.

Configuring the Hardware

This section describes how to configure the supported hardware.

Description

PIC32 Bluetooth Audio Development Kit

Configuration: bt_audio_dk

No hardware related configuration or jumper setting changes are necessary.

PIC32 USB Starter Kit II with the Low-Cost Controllerless (LCC) Graphics PICtail Plus Daughter Board and Graphics Display Truly 3.2" 320x240 Board

Configuration: pic32mx_usb_sk2_lcc_pictail_qvga

No hardware related configuration or jumper setting changes are necessary.

PIC32 USB Starter Kit II with the Low-Cost Controllerless (LCC) Graphics PICtail Plus Daughter Board and Graphics Display Powertip 4.3" 480x272 Board

Configuration: pic32mx_usb_sk2_lcc_pictail_wqvga

No hardware related configuration or jumper setting changes are necessary.

PIC32 USB Starter Kit II with the Multimedia Expansion Board II (MEB II)

Configuration: pic32mx_usb_sk2_meb

No hardware related configuration or jumper setting changes are necessary.

PIC32 USB Starter Kit II with the Graphics Controller PICtail Plus Epson S1D13517 Daughter Board

Configuration: pic32mx_usb_sk2_s1d_pictail_wqvga, pic32mx_usb_sk2_s1d_pictail_wvga

P2 on the S1D13517 Daughter Board should be closed (16-bit PMP) for the WQVGA display, and open (8-bit PMP) for the WVGA display.

PIC32 USB Starter Kit II with the Graphics LCD Controller PICtail Plus SSD1926 Daughter Board

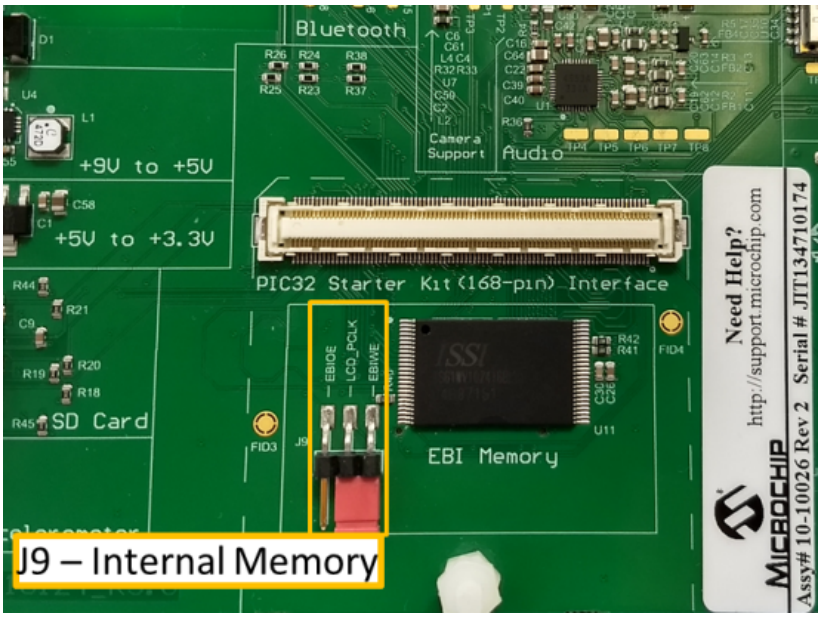
Configuration: pic32mx_usb_sk2_ssd_pictail_qvga

JP2 on the SSD1926 Daughter Board should be set to select the 8-bit PMP interface.

PIC32MZ EF Starter Kit connected to the Multimedia Expansion Board II (MEB II) (Internal SRAM Frame Buffer)

Configuration: pic32mz_ef_sk_meb2

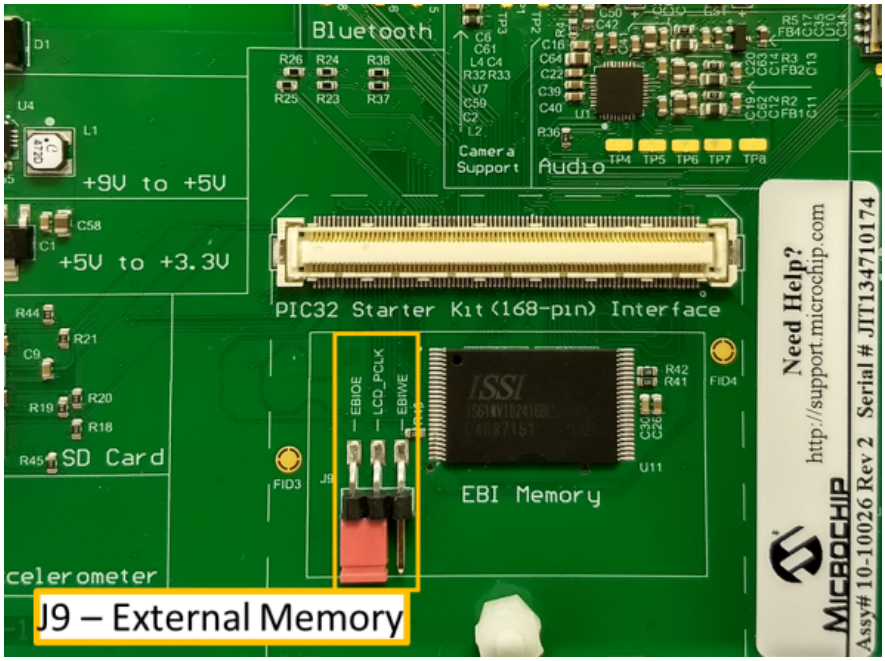
- The J9 Jumper on the MEB II board, located beneath the PIC32MZ EF Starter Kit, should be configured to support using internal SRAM for the graphics frame buffer, as shown in the following figure.
- Connect the PIC32MZ EF Starter Kit board to the MEB II board
- Power up the board by connecting the power adapter to J3 power connector on the MEB II board or a powered USB cable to the USB DEBUG port on the Starter Kit board



PIC32MZ EF Starter Kit connected to the Multimedia Expansion Board II (MEB II) (External SRAM Frame Buffer)

Configuration: pic32mz_ef_sk_meb2_ext

- The J9 Jumper on the MEB II board, located beneath the PIC32MZ EF Starter Kit, should be configured to support using external SRAM for the graphics frame buffer, as shown in the following figure.
- Connect the PIC32MZ EF Starter Kit board to the MEB II board
- Power up the board by connecting the power adapter to J3 power connector on the MEB II board or a powered USB cable to the USB DEBUG port on the Starter Kit board



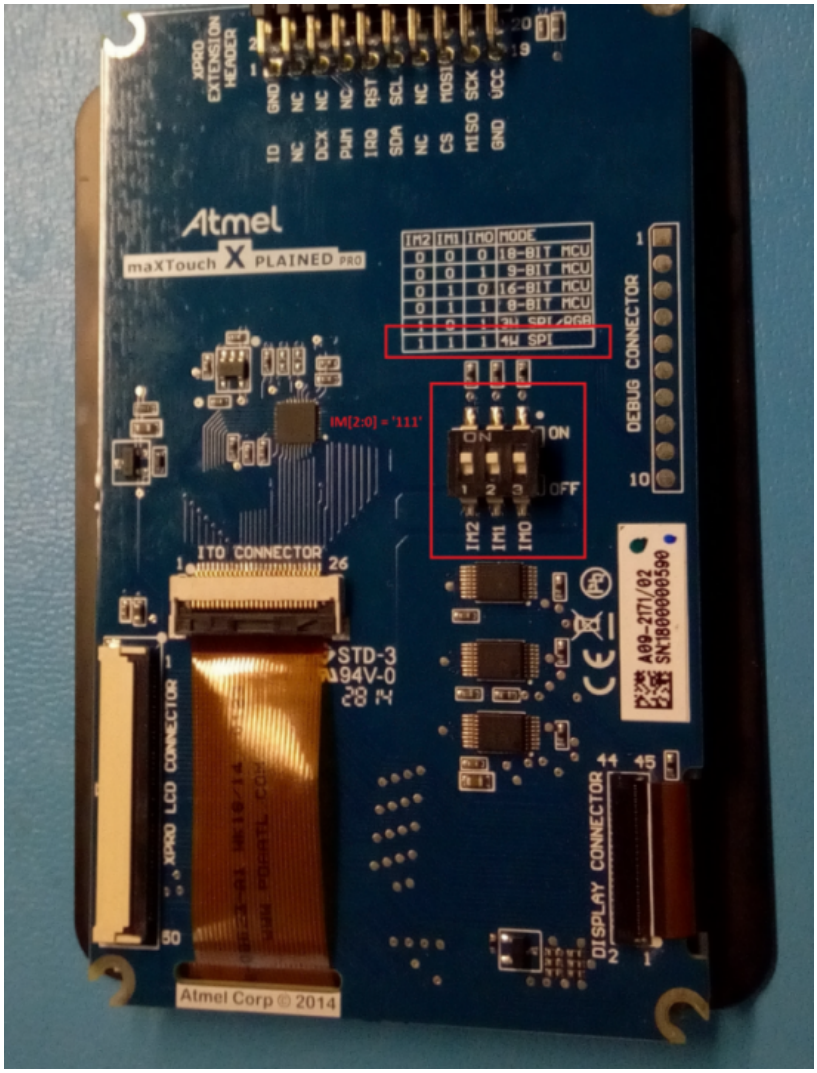
PIC32MZ EF Starter Kit with the maXTouch Xplained Pro 3.5" display.

pic32mz_ef_sk_xpro

1. Connect the pins of the maXTouch Xplained Pro board to the PIC32MZ EF Starter Kit, as described in the following table.

	PIC32MZ EF Starter Kit		3.5" Xplained Pro maXTouch	
Signal name	MZ EF SK Name	PIN # on J12	Name	XPRO Expansion Header
Touch Interface				
I2C Clock	SCL2/RA2	5	SCL	12
I2C Data	SDA3/RA3	3	SDA	11
INT	SDA3/RPF2/RF2	12	IRQ	9
Other Controls				
Reset	RPA14/SCL1/RA14	7	RST	10
Backlight Control	AN9/RPB14/SCK3/RB14	10	PWM	7
SPI Interface				
Clock (SCK)	RPD1/SCK1/RD1	23	SCK	18
MOSI (SDO)	AN2/C2INA/RPB3/RB3	15	MOSI	16
MISO (SDI)	EBIA9/PMA9/RPF4	21	MISO	17
Slave Select (/SS)	AN14/C1IND/RPG6/SCK2/RG6	8	CS	15
Data Command Select (D/C)	INT0/RD0/RPD0	11	DCX	5

2. Set the interface mode switches on the maXTouch Xplained Pro to “4W SPI (111)."



3. Finally, power up the PIC32MZ EF Starter Kit by connecting a powered USB cable to the USB DEBUG mini-USB port on the board.

PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit with the PIC32MZ Starter Kit Adapter Board (168-to-132 pin Starter Kit Adaptor), Graphics Controller PICtail Plus Epson S1D13517 Daughter Board, and the Graphics Display Powertip 4.3" 480x272 Board

Configuration: pic32mz_ef_sk_s1d_pictail_wqvga

1. On the Graphics Controller PICtail Plus Epson S1D13517 Daughter Board, jumper JP2 should be open to select the 8-bit PMP interface to the display. JP2 is found on the lower half of the board between the SDRAM chip and the Display_Controller chip.
2. On the bottom of the 168-to-132 Pin Starter Kit Adapter board, JP2 and JP3 should be jumpered closed. JP1 should be jumpered to connect PMD11 to EBID11.
3. On the EF Starter Kit: Remove R67 from U8 (MCP2221) circuitry. Jumper Pin 8 to Pin 13 on J12. J12 is located on the far right side of the board.
4. P2 on the S1D13517 Daughter Board should be closed (16-bit PMP) for the WQVGA display.

PIC32MK General Purpose (GP) Development Board

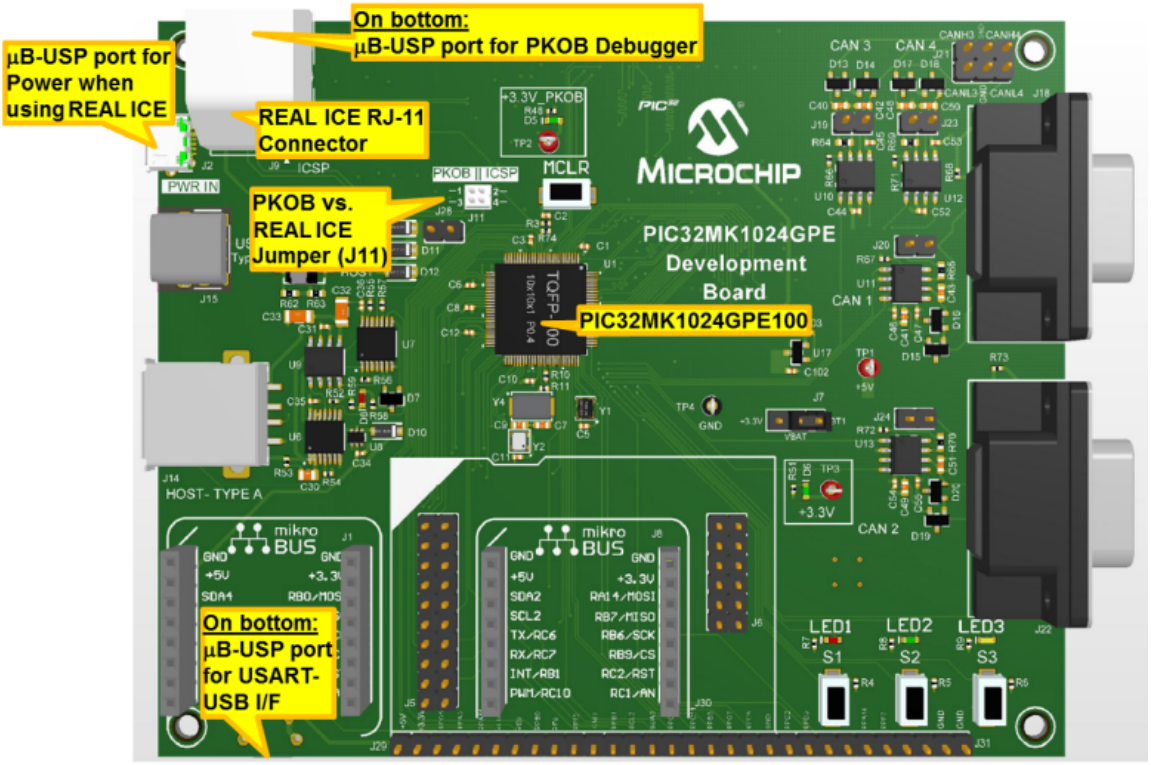
Configurations: pic32mk_gp_db_wqvga_mxt *or* pic32mk_gp_db_wvga_mxt

These configurations use the PIC32MK GP Development Board (DM320106) and one of two displays plus maXTouch module setups. Either a High-Performance 4.3" WQVGA Display Module with maXTouch (AC320005-4) board, or a High-Performance (5") WVGA Display Module with maXTouch. (AC320005-5) board is connected to the development board using a 50-line ribbon cable from the display to the development kit board.

Both the 480x272 WQVGA and the 800x480 WVGA modules consist of two boards sandwiched together and connected via a 50-line ribbon cable. The second board is an interposer that adapts the display to the Multimedia Expansion Board II (MEB II) (DM320005-5). By releasing the ribbon cable from the interposer board, it is possible to connect the display to the development kit board via connector J26 on the bottom of the development kit board.

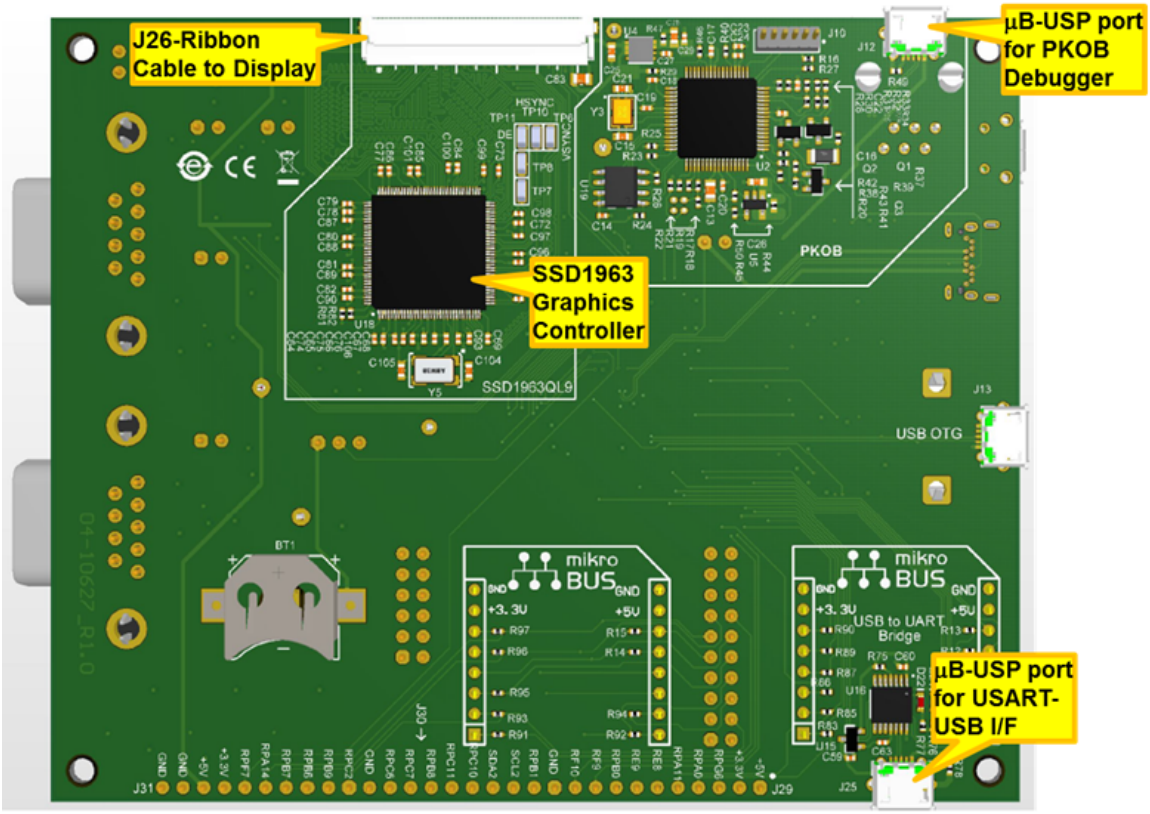
Details for the PIC32MK GP Development Board

The top of the board is shown in the following figure.



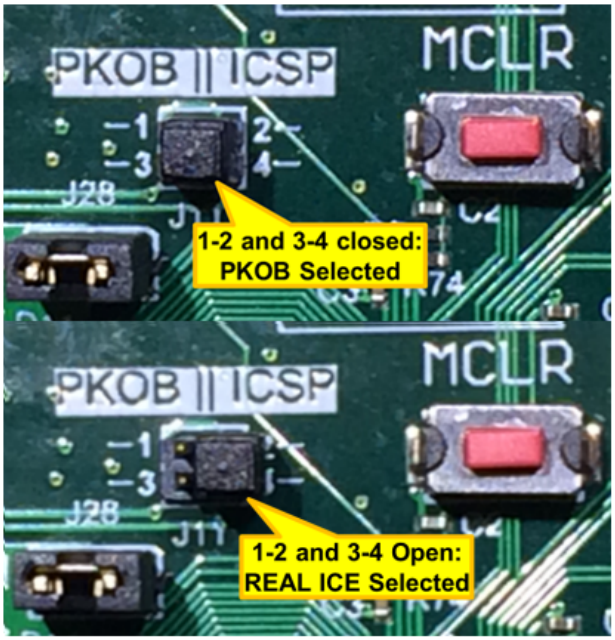
The board can be programmed using the PICkit On-board (PKOB) interface using a microB-A USB cable or using a MPLAB REAL ICE™ In-Circuit Emulator using an RJ11 cable. When using the REAL ICE it is required that a microB-A USB cable be connected to the “PWR IN” port on the board to provide 5 Volts to the board from your PC.

The bottom of the board is shown in the following figure:



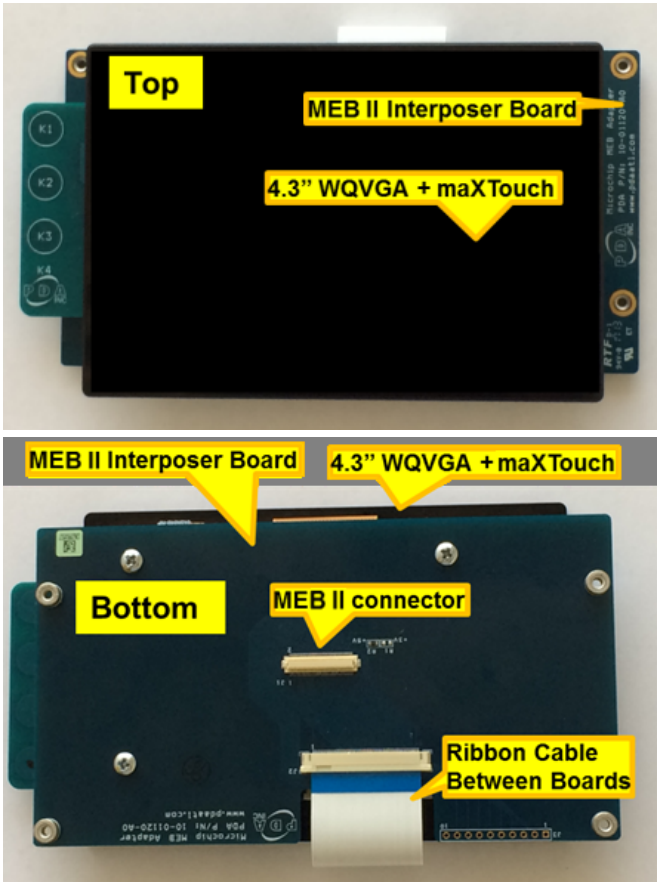
The microB USB port on the bottom of the board provides a USART-USB gateway connection to a HyperTerminal equivalent application (e.g., RealTerm) via a COM port on your PC. (Set up the PC’s COM port to be 115200 Baud, 8 bits, 1 stop bit, no parity.) Both projects are set up to support asserts, exception handling, and System Console writes to this USART port.

To switch between the PKOB and REAL ICE debuggers, you must change the J11 Jumper, as follows:



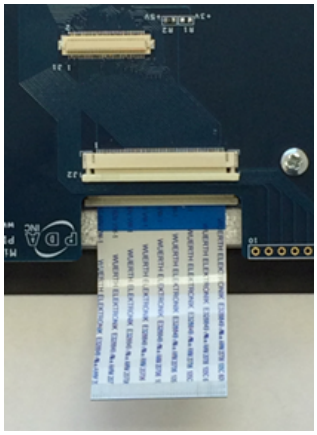
Details of the 4.3” WQVGA plus maXTouch

(The 5” WVGA plus maXTouch board is similar in appearance.)

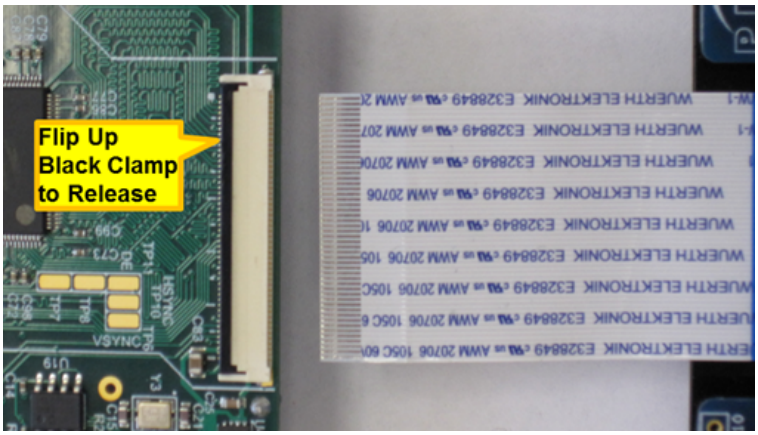


How to connect the boards:

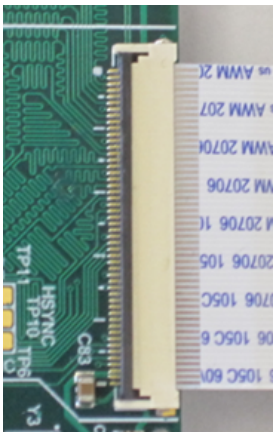
1. Turn over the display/interposer and release the ribbon cable from the interposer board:



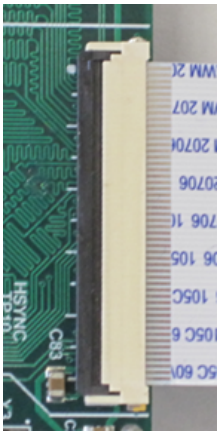
2. Turn over the development kit board and release the clamp on the J26 ribbon socket:



3. Insert the ribbon cable all the way into J26:



4. Close the clamp on J26:

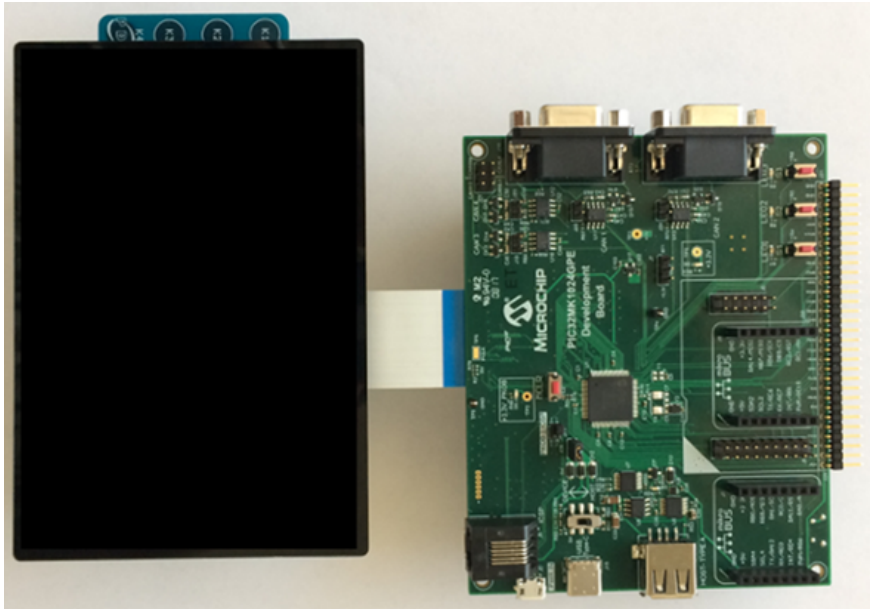


Completed Board Configurations

pic32mk_gp_db_wqvga_mxt



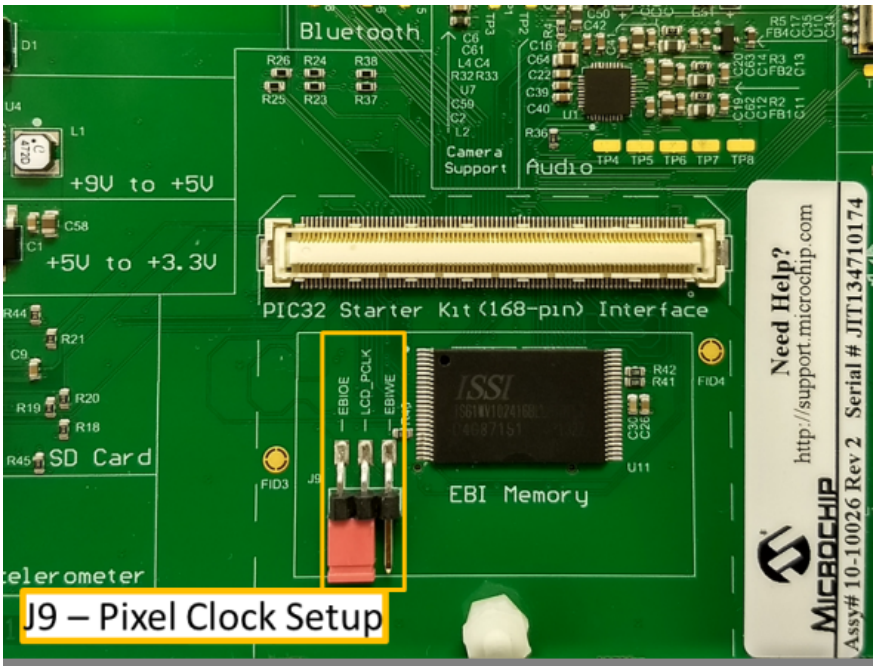
pic32mk_gp_db_wvga_mxt



PIC32MZ Embedded Graphics with DA Starter Kit with the Multimedia Expansion Board II (MEB II) (First and Second Generation)

Configurations: pic32mz_da_sk_extddr_meb2, pic32mz_da_sk_intddr_meb2, pic32mz_da_sk_intddr_meb2, pic32mz_da_sk_intddr_meb2_wvga, pic32mz_da_sk_noddr_meb2_rgb565, pic32mz_da_sk_noddr_meb2_rgba8888, and pic32mz_da_sk_noddr_meb2_wvga_lut8

- These configurations require that the J9 jumper be set to provide the GLCD's pixel clock. Set the J9 jumper to connect EPIOE to LCD_PCLK. The J9 jumper is located on the bottom of the MEB II board, beneath where the starter kit is plugged into the board. Refer to the followin figure for the exact location.
- Connect the PIC32MZ DA Starter Kit board to the MEB II board
- Power up the board by connecting the power adapter to J3 power connector on the MEB II board or a powered USB cable to the USB DEBUG port on the Starter Kit board



Running the Demonstration

This section provides instructions about how to build and run the Aria Quick Start demonstration.

Description

Note: For this demonstration, the following configurations do not support touch:

- pic32mz_ef_sk_xpro
- bt_audio_dk,

When power-on is successful, the demonstration will display a similar menu to that shown in the following figure (different configurations may have slight variation in the screen aspect ratio):



When **Make changes. Generate. Run.** is touched, the button will toggle with each individual touch.



aria_radial_menu

The aria_radial_menu demonstrates four different applications of the radial menu widget.

Description

This application showcases four ways of using the radial menu widget.

It simulates 3D rotation by using an elliptical track as a 3D-to-2D projection. Images used in the radial menus have the option to grow or shrink to further enhance the 3D rotation effect. The application supports touch interaction.

The demonstration launches with a splash screen highlighting basic motion capability supported by Aria Graphics Library.

When running the application, the user can interface with it via capacitive single-fingered touch and swiping gestures.

The main menu is a radial menu that features images stored as PNGs having been preprocessed at boot-time. It is also used as a navigation menu to the other three radial menus.

The other three radial menus are used as image carousel. One shows a series of music albums arranged in an orbital track. Another is used to show representations of famous portraits, and demonstrates how the radial menu widget handles items with varying dimensions. The last radial menu is showing vertical rotation to simulate a business card rolodex.

An information screen is used to explain the features demonstrated in this application.

Architecture

The diagrams below show the various software and hardware components for each configuration.

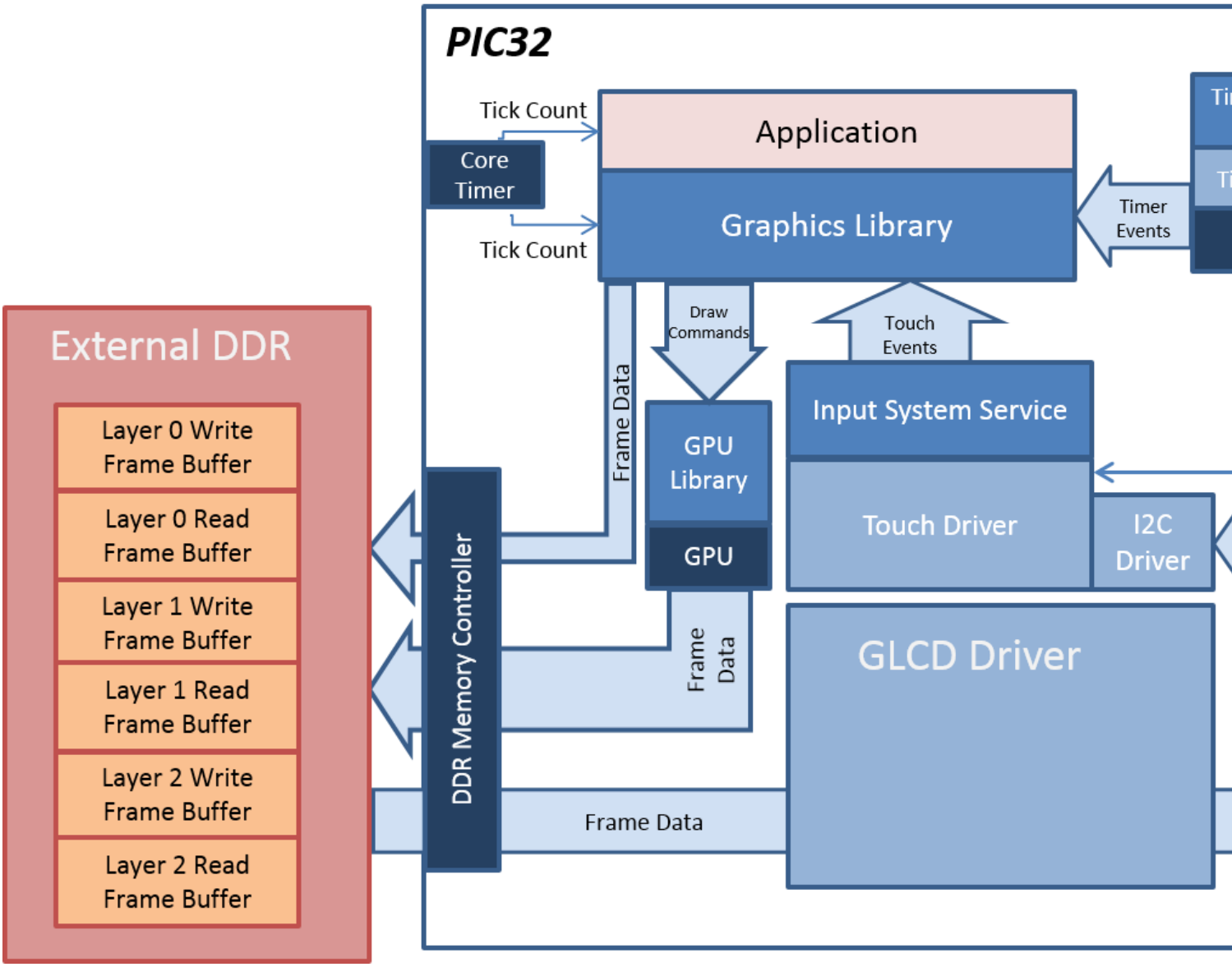
pic32mz_da_sk_extddr_meb2_wvga and pic32mz_da_sk_extddr_meb2

For this configuration, the application uses the Graphics library to render graphics to the display. The Graphics library passes draw commands into the GPU Library, which in-turn draws the widgets and images to the frame buffer that is stored in an external DDR2. Using the DDR2 Memory

Controller, the GLCD display controller peripheral continuously transfers frame data from the frame buffer onto to the LCD display.

The application also features user touch input through the integrated touch screen on the display panel. Touch input from the touch controller goes thru the I2C port, and the Input System Service acquires the touch input information from the touch and I2C drivers. The Input System Service sends touch events to the Graphics library which processes these events and updates the frame data accordingly.

The core timer is used by the application and the Graphics library to manage the movement of the splash screen.

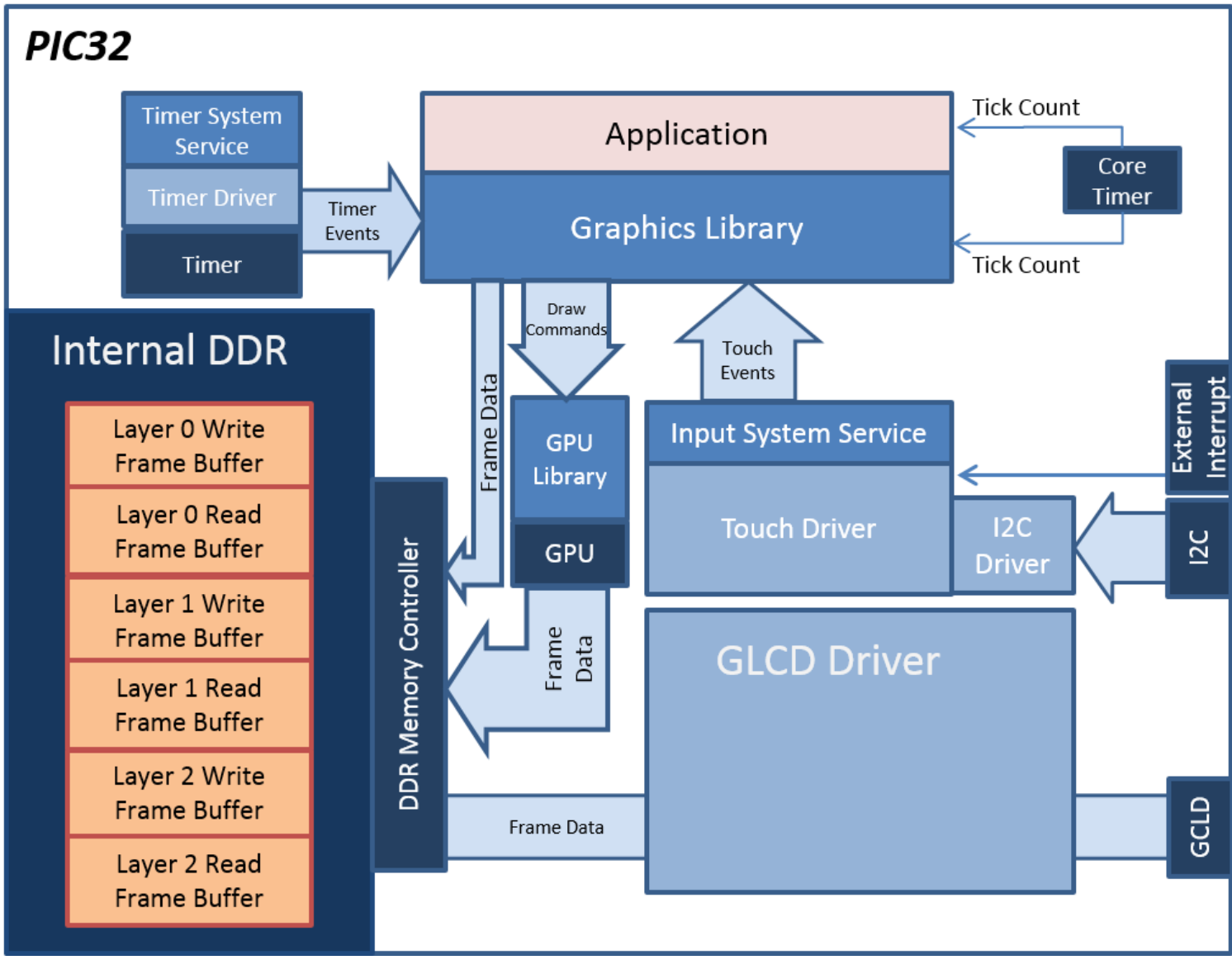


pic32mz_da_sk_intddr_meb2_wvga and pic32mz_da_sk_intddr_meb2

For this configuration, the application uses the Graphics library to render graphics to the display. The Graphics library passes draw commands into the Nano2D GPU Library, which in-turn draws the widgets and images to the three individual write frame buffers (one for each layer) that are stored in an external DDR2. Using the DDR2 Memory Controller, the Graphics LCD (GLCD) display controller peripheral continuously transfers frame data from the all three read buffers onto to the LCD display. The write and read frame buffer pairs are swapped independently as required when the Graphics Library is done rendering to the write frame buffer and GLCD driver signals the GLCD peripheral to change its read location.

The application also features user touch input through the integrated touch screen on the display panel. Touch input from the touch controller goes thru the I2C port, and the Input System Service acquires the touch input information from the touch and I2C drivers. The Input System Service sends touch events to the Graphics library which processes these events and updates the frame data accordingly.

The core timer is used by the application and the Graphics library to manage the movement of the splash screen.



Demonstration Features

- Radial Menu Widget
- Image Blit and Image Stretch supported via the Nano2D library (Nano2D Driver Library)
- Integrated PCAP Touch Input
- Three graphics layer supported via the GLCD peripheral on the PIC32MZ DA device (GLCD Controller Peripheral Library)
- GPU peripheral supported, can be enabled/disabled at run-time (GPU Hardware Accelerated Features)
- 32-bit RGBA8888 color depth support (16.7 million unique colors)
- Per-layer frame double-buffering
- Image compression techniques using Run-Length Encoding, PNG, and JPEG (Enabling Demo Mode in a MPLAB Harmony Graphics Application)

Tools Setup Differences

For all configurations:

- Use Graphics Stack is selected in MHC. This enables the graphics and touch libraries and drivers for the BSP

pic32mz_da_sk_intddr_meb2_wvga, pic32mz_da_sk_extddr_meb2_wvga

- Pin Settings: External Interrupt 4 mapped to pin A14 (RB1)

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Aria Radial Menu

demonstration.

Description

To build this project, you must open the aria_radial_menu.X project in MPLAB X IDE, and then select the desired configuration. The following tables list and describe the project and supported configurations. The parent folder for these files is <install-dir>/apps/gfx/aria_radial_menu.

MPLAB X IDE Project

The following table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`aria_radial_menu.X`	*<install-dir>/apps/gfx/aria_radial_menu/firmware*

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within ./firmware/src/system_config.

Project Configuration Name	BSP Used	Description
pic32mz_da_sk_intddr_meb2_wvga	pic32mz_da_sk_intddr+meb2+wvga	Demonstration for the Multimedia Expansion Board II (MEB II) connected to the PIC32MZ Embedded Graphics with InternalDRAM (DA) Starter Kit and 5” WVGA PCAP Display.
pic32mz_da_sk_extddr_meb2_wvga	pic32mz_da_sk_extddr+meb2+wvga	Demonstration for the Multimedia Expansion Board II (MEB II) connected to the PIC32MZ Embedded Graphics with ExternalDRAM (DA) Starter Kit and 5” WVGA PCAP Display.
pic32mz_da_sk_extddr_meb2	pic32mz_da_sk_extddr+meb2	Demonstration for the Multimedia Expansion Board II (MEB II) connected to the PIC32MZ Embedded Graphics with External DDR (DA) Starter Kit.
pic32mz_da_sk_intddr_meb2	pic32mz_da_sk_intddr+meb2	Demonstration for the Multimedia Expansion Board II (MEB II) connected to the PIC32MZ Embedded Graphics with Internal DDR (DA) Starter Kit.

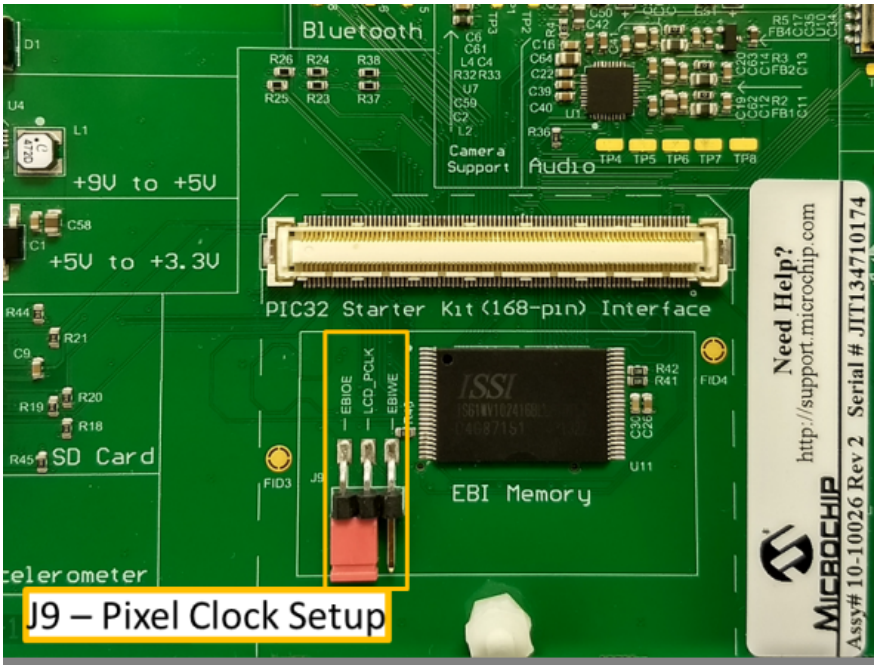
Configuring the Hardware

This section describes how to configure the supported hardware.

Description

Configurations: pic32mz_da_sk_extddr_meb2_wvga, pic32mz_da_sk_intddr_meb2_wvga

- On the MEB II, the EBIOE and LCD_PCLK (J9) must be jumpered. A connection establishes the GLCD's pixel clock output timing. The external SRAM memory on the board is disabled. The jumper (J9) is available on the bottom side of the MEB II board under the starter kit. The J9 jumper is located on the bottom of the MEB II board, beneath where the starter kit is plugged into the board. Refer to the following image for the exact location.
- Connect the PIC32MZ DA Starter Kit board to the MEB II board
- Power up the board by connecting the power adapter to J3 power connector on the MEB II board or a powered USB cable to the USB DEBUG port on the Starter Kit board

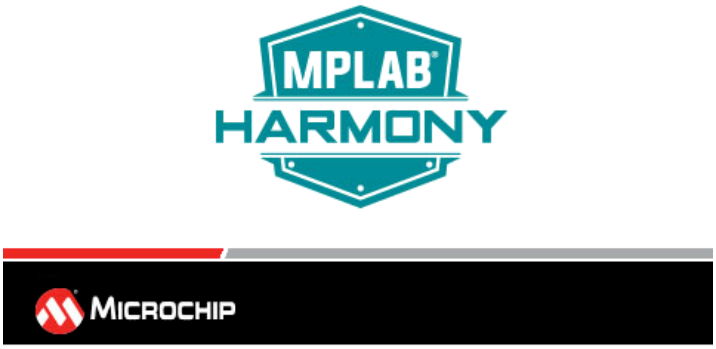


Running the Demonstration

This section provides a brief description of what to expect once the demonstration is running.

Description

On start-up, the application will display a splash screen:



Main Screen

Subsequently, the demonstration's main screen will appear:



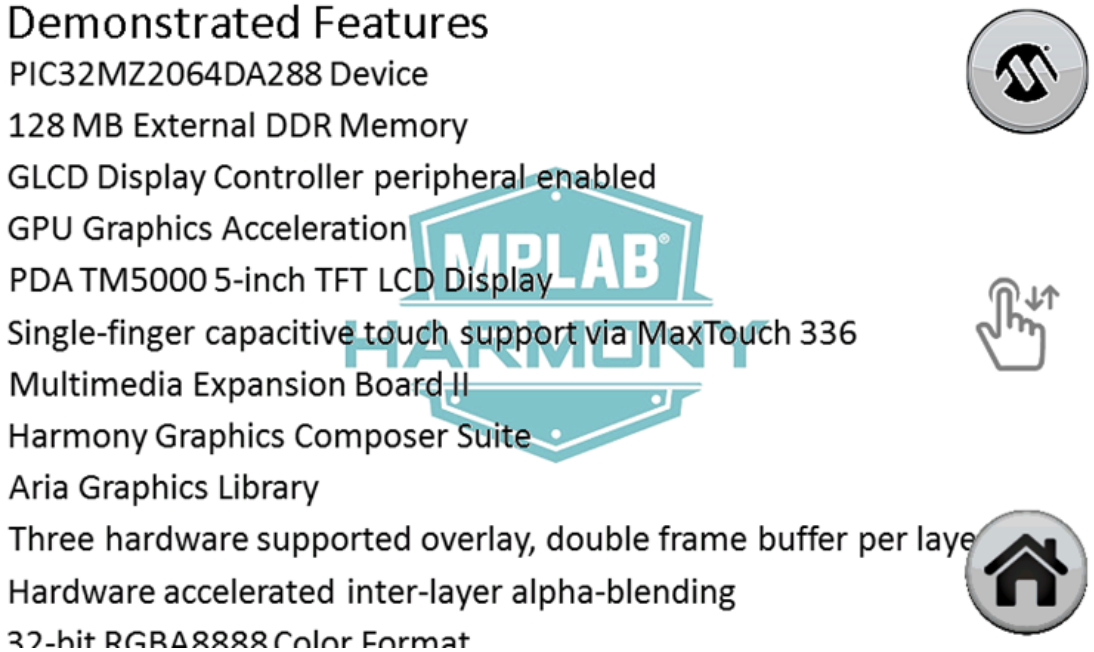
Using a single-finger horizontal swiping motion, each of the four available selections in the main menu can be accessed. When finger touch input is released, the radial menu will *reset* such that the image closest to the front is placed front and center. Once the radial menu motion stops and one of the images is front and center, the corresponding mode is shown in the text label. These are: *Demonstrated Features*, *Albums*, *Portraits*, and *Rolodex*. Only the icon in this *prominent* position is the selectable option. Press and release the image in the *prominent* position to navigate to the screen supported by the stated mode.

Information Screen

Tapping the icon corresponding to the *Demonstrated Features* label in the main screen will switch the application to the information screen.

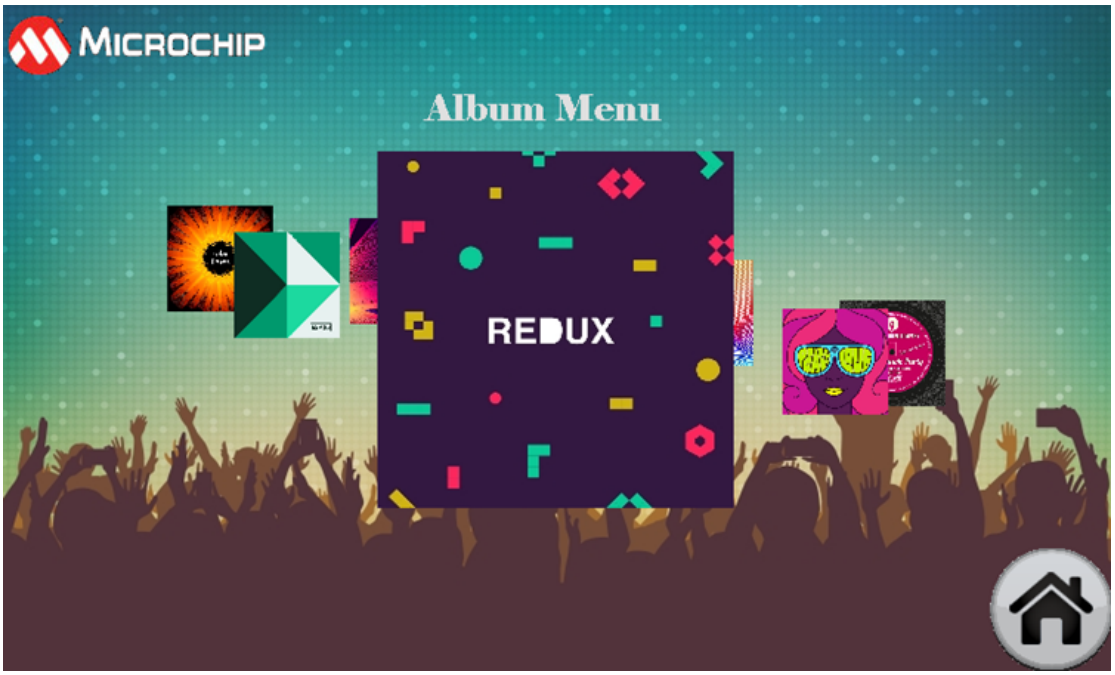
The information screen serves as a screen to list out the features of the demonstration.

- The features are located on layer1. Sliding your finger up or down will cause the feature list to scroll up and down. The MPLAB Harmony logo that is rendered on layer0 is another demonstration of the real-time alpha-blending capability of the GLCD.
- Touching the Microchip logo will trigger the application to the splash screen. Once the splash screen animation has completed, the application will return to the information screen
- Pressing the Home icon will switch the application back to the main menu



Album Menu Screen

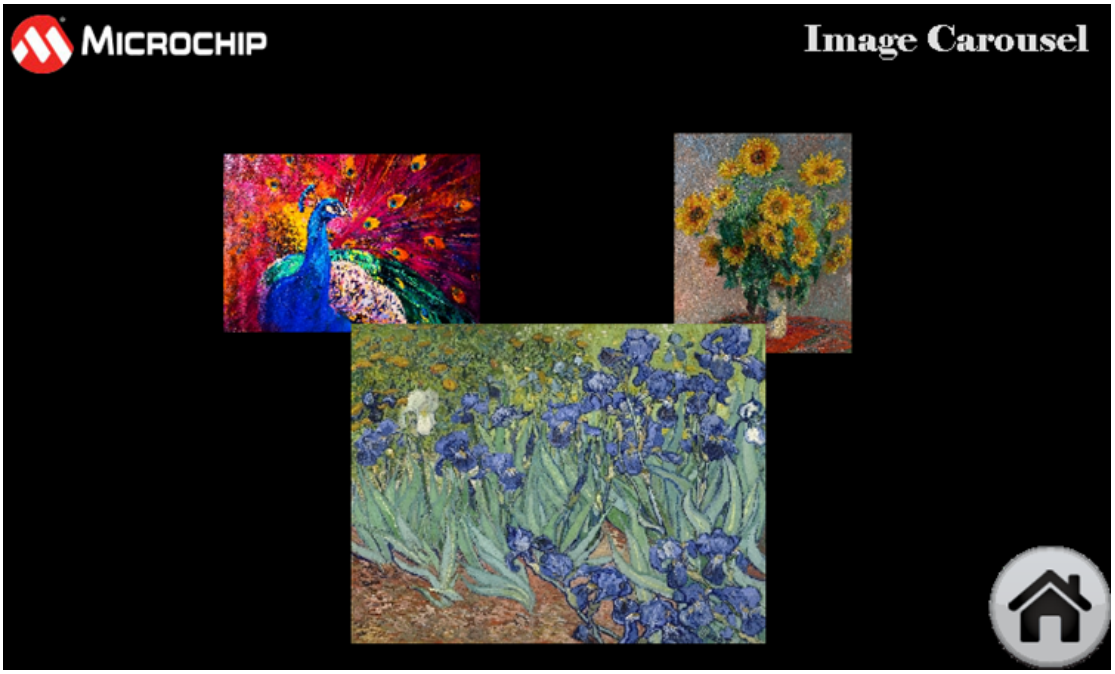
Tapping the icon corresponding to *Albums* label in the main screen will switch the application to the Album Menu screen.



- Using a single-finger horizontal swiping motion to cycle through the images. The radial menu widget in this screen is configured to follow an 8-degree tilt orbital elliptical track. Image scaling is set to *Prominent Only*. That means only the image that is in the Prominent position gets an increase in size. There are 10 unique fictitious album covers, but only 7 are shown at any moment. The remaining three are hidden.
- Pressing the Home icon will switch the application back to the main menu.

Image Carousel Screen

Tapping the icon corresponding to *Portraits* label in the main screen will switch the application to the Image Carousel screen.



- Using a single-finger horizontal swiping motion to cycle through the images. The radial menu widget in this screen is configured to follow the default elliptical track, which maximizes the track-travel to the rectangular area as defined by the original image. Image scaling is set to *Gradual*. That means each image gradually increases in scale from 30% (of the original image size) in the back, to 100% in the front. There are

seven total unique portraits, with the widget showing three at any one time.

- Pressing the Home icon will switch the application back to the main menu.

Rolodex Menu Screen

Tapping the icon corresponding to *Rolodex* label in the main screen will switch the application to the Rolodex Menu screen.



- Using a single-finger vertical swiping motion to cycle through the placeholder business card images. The radial menu widget in this screen is configured to follow a vertical track. Image scaling is set to *Gradual*. This means each image gradually increases in scale from 30% (of the original image size) in the back, to 100% in the front. There are six unique business card designs, but only five are shown at any moment.
- Pressing the Home icon will switch the application back to the main menu.

aria_scrolling

This demonstration provides a preconfigured touch-enabled example of the off-screen widget management drag-based widget motion capabilities of the Aria User Interface Library.

Description

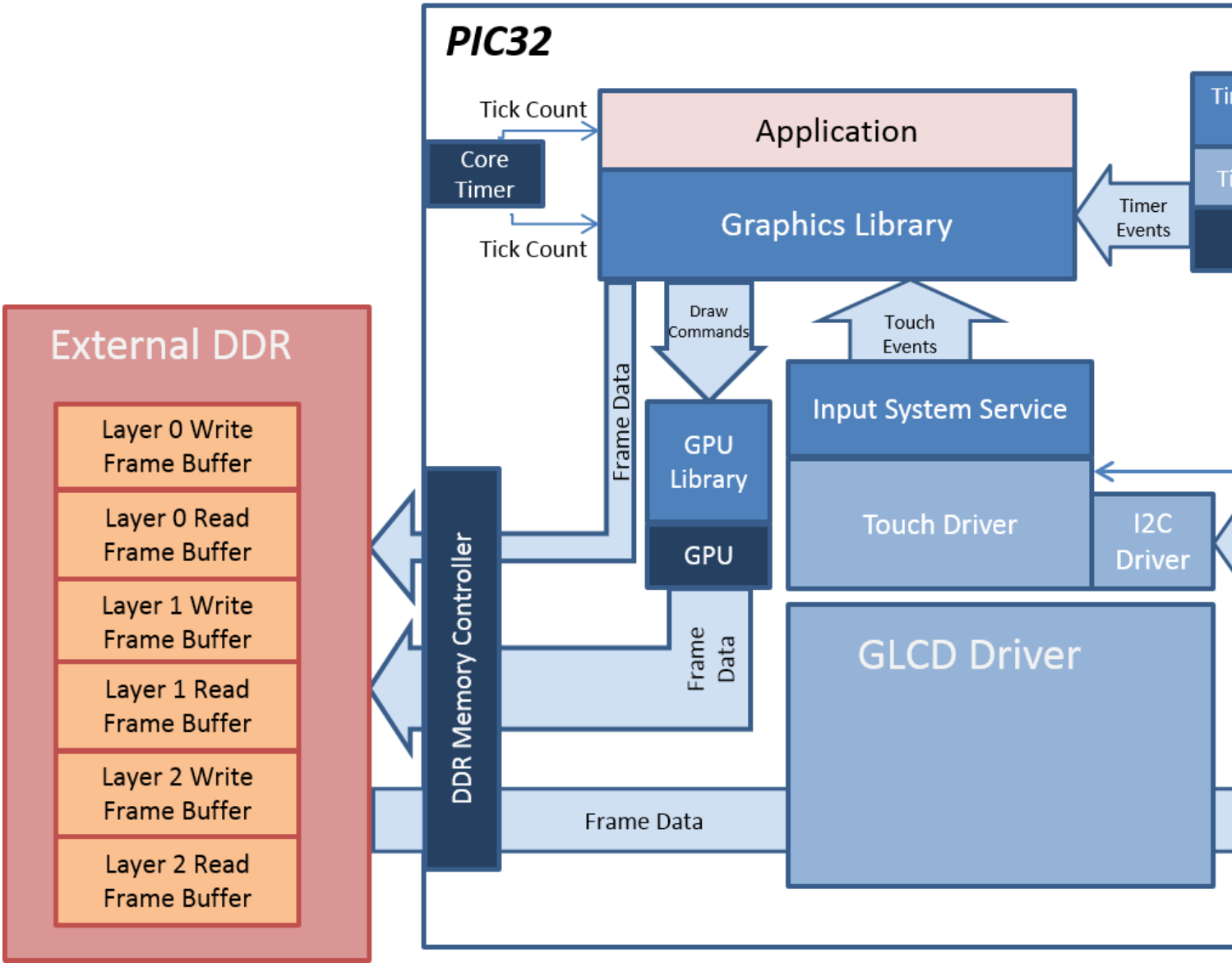
This demonstration provides a preconfigured touch-enabled example of the off-screen widget management drag-based widget motion capabilities of the Aria User Interface Library. The demonstration has multiple configurations to demonstrate the graphics library running on different MCUs, displays and display drivers.

Architecture

pic32mz_da_sk_extddr_meb2, pic32mz_da_sk_extddr_meb2_wvga

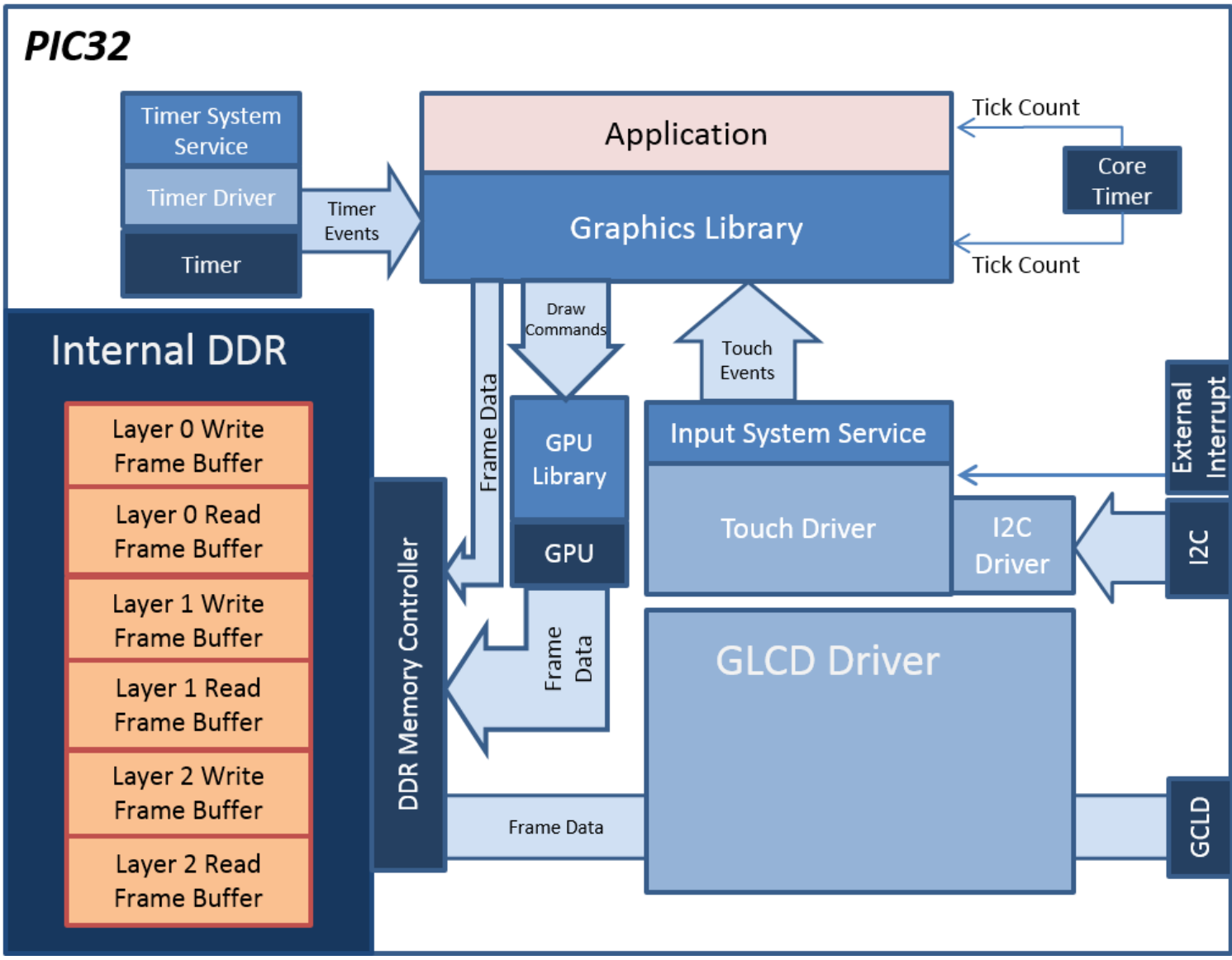
For these configurations, the application uses the Graphics Library to render graphics to the display. The Graphics Library draws the widgets and images to the write frame buffer that is stored in an external DDR2. Through the DDR2 Memory Controller, the GLCD display controller peripheral continuously transfers frame data from the frame buffer onto to the LCD display. The write and read frame buffers are swapped when the Graphics Library is done rendering to the write frame buffer and GLCD driver signals the GLCD peripheral to change its read location.

The application also features user touch input through the integrated touch screen on the display panel. Touch input from the touch controller goes through the I2C port, and the Input System Service acquires the touch input information from the touch and I2C drivers. The Input System Service sends touch events to the Graphics Library, which processes these events and updates the frame data accordingly.



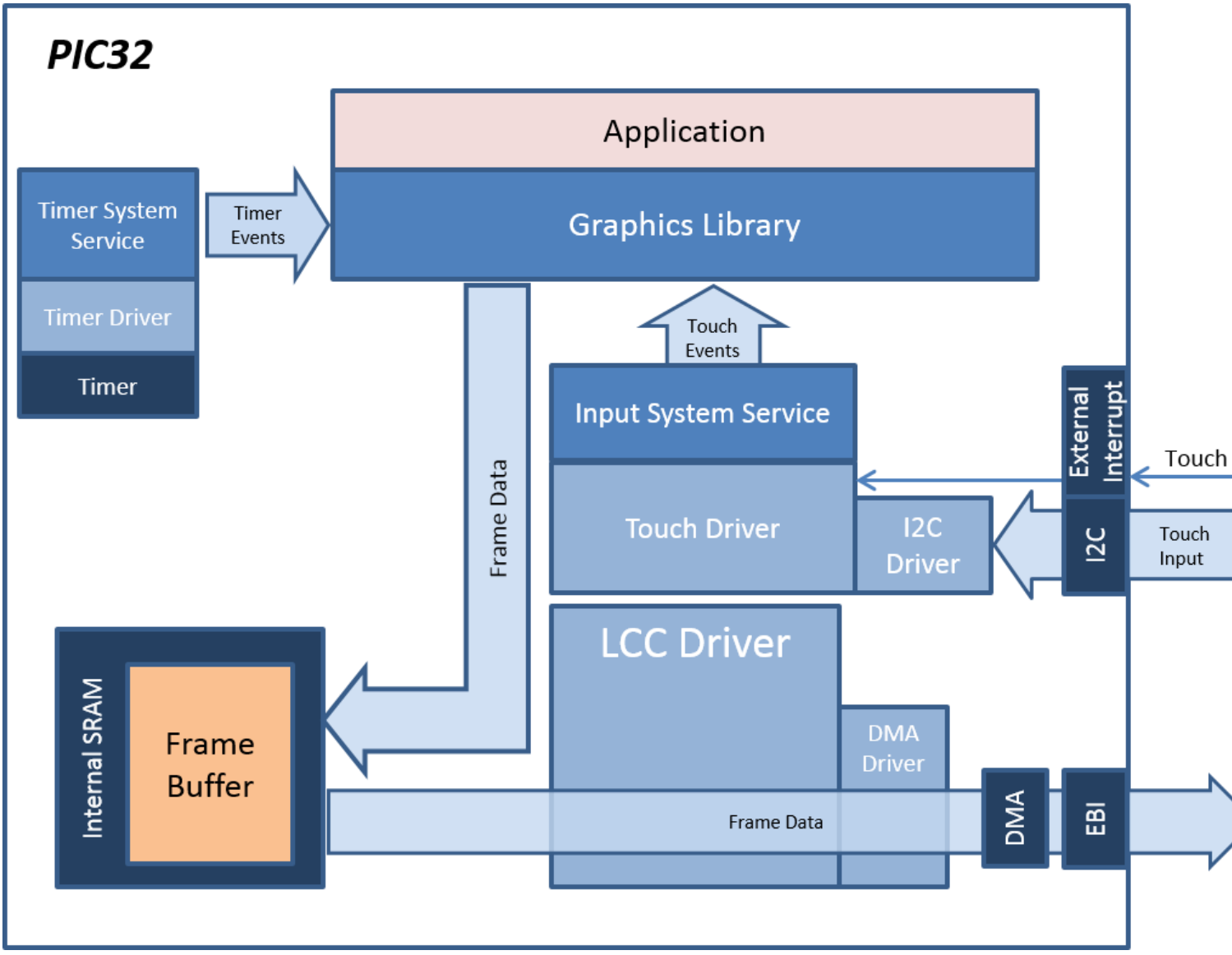
pic32mz_da_sk_intddr_meb2, pic32mz_da_sk_intddr_meb2_wvga

For these configurations, the application uses the Graphics Library to render graphics to the display. The Graphics Library draws the widgets and images to the write frame buffer that is stored in an internal SRAM. Using the DMA, the Low-Cost Controllerless (LCC) display driver continuously transfers frame data from the frame buffer out to the LCD display.



pic32mz_ef_sk_meb2

The application also features user touch input through the integrated touch screen on the display panel. Touch input from the touch controller goes through the I2C port, and the Input System Service acquires the touch input information from the touch and I2C drivers. The Input System Service sends touch events to the Graphics Library, which processes these events and updates the frame data accordingly.



Demonstration Features

This application demonstrates the following MPLAB Harmony Graphics features:

- RGBA8888 Color Mode (MZDA)
- RGB565 Color Mode (MZEF)
- Internal Flash Asset Storage
- 8-bit Palette Compressed Images
- Run-length Encoded Images
- 1bpp Font Glyph Decoding
- ASCII Character Encoding
- Single Display Layer
- Display Edge Clipping
- User Interface Tree Hierarchy
- Parent/Child Area Clipping
- UI Dragging and Touch
- Button Widget
- Label Widget
- UI Widget Bounds that Exceed Display Area
- Double Buffering (MZDA)

- Vertical Sync Swapping (MZDA)

Tools Setup Differences

- “Use Graphics Stack” is selected in the MPLAB Harmony Configurator (MHC), which enables the Graphics and Touch libraries and drivers for the Board Support Package (BSP).
- The heap size for all configurations except the pic32mz_da_sk_extddr_meb2_wvga is set using the MHGC Heap Estimator Tool.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Aria Scrolling demonstration.

Description

To build this project, you must open the `aria_scrolling.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/gfx/aria_scrolling`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`aria_scrolling.X`	`<install-dir>/apps/gfx/aria_scrolling/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mz_da_sk_extddr_meb2	pic32mz_da_sk_extddr+meb2	Demonstration for the Multimedia Expansion Board II (MEB II) connected to the PIC32MZ Embedded Graphics with External DRAM (DA) Starter Kit.
pic32mz_da_sk_extddr_meb2_wvga	pic32mz_da_sk_extddr+meb2+wvga	Demonstration for the Multimedia Expansion Board II (MEB II) connected to the PIC32MZ Embedded Graphics with External DRAM (DA) Starter Kit and High-Performance 5" WVGA Display Module with maXTouch.
pic32mz_ef_sk_meb2	pic32mz_ef_sk+meb2	Demonstration for the Multimedia Expansion Board II (MEB II) connected to the PIC32MZ EF Starter Kit.
pic32mz_da_sk_intddr_meb2	pic32mz_da_sk_intddr+meb2	Demonstration for the Multimedia Expansion Board II (MEB II) connected to the PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit.
pic32mz_da_sk_intddr_meb2_wvga	pic32mz_da_sk_intddr+meb2+wvga	Demonstration for the Multimedia Expansion Board II (MEB II) connected to the PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit and a High-Performance 5" WVGA Display Module with maXTouch.

Configuring the Hardware

This section describes how to configure the supported hardware.

Description

PIC32MZ Embedded Graphics with External DRAM (DA) Starter Kit and MEB II, PIC32MZ Embedded Graphics with External DRAM (DA) Starter Kit with MEB II and High-Performance WVGA Display Module with maXTouch, PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit and MEB II, PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit with MEB II and High Performance WVGA Display Module with maXTouch

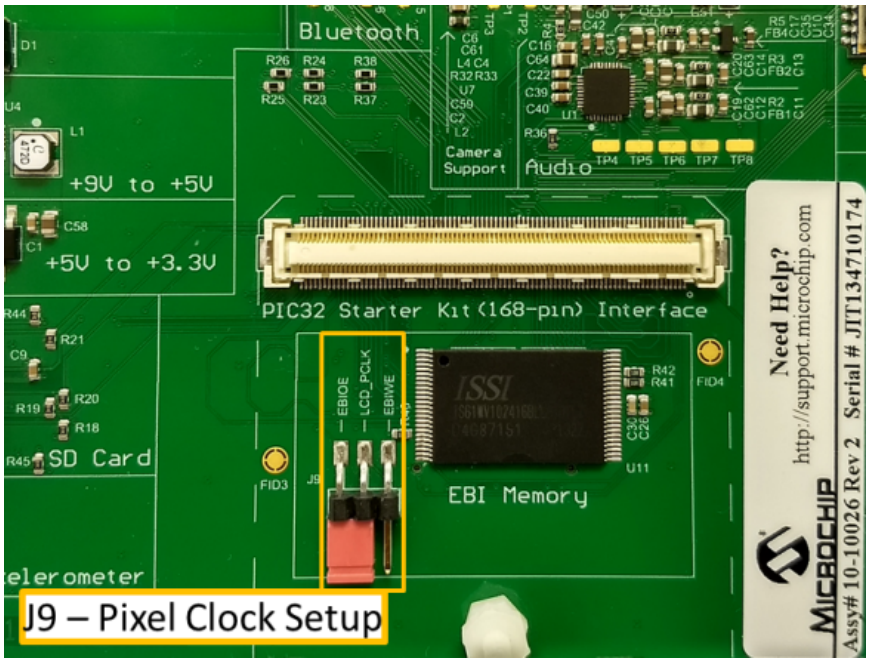
Configurations: pic32mz_da_sk_extddr_sk+meb2, pic32mz_da_sk_extddr_meb2, pic32mz_da_sk_extddr_meb2_wvga, pic32mz_da_sk_intddr_meb2,

and pic32mz_da_sk_intddr_meb2_wvga

- On the MEB II, the EBIOE and LCD_PCLK (J9) must be jumpered. A connection establishes the GLCD's pixel clock output timing. The external SRAM memory on the board is disabled. The jumper (J9) is available on the bottom side of the MEB II board under the starter kit. The J9 jumper is located on the bottom of the MEB II board, beneath where the starter kit is plugged into the board. Refer to the following image for the

exact location.

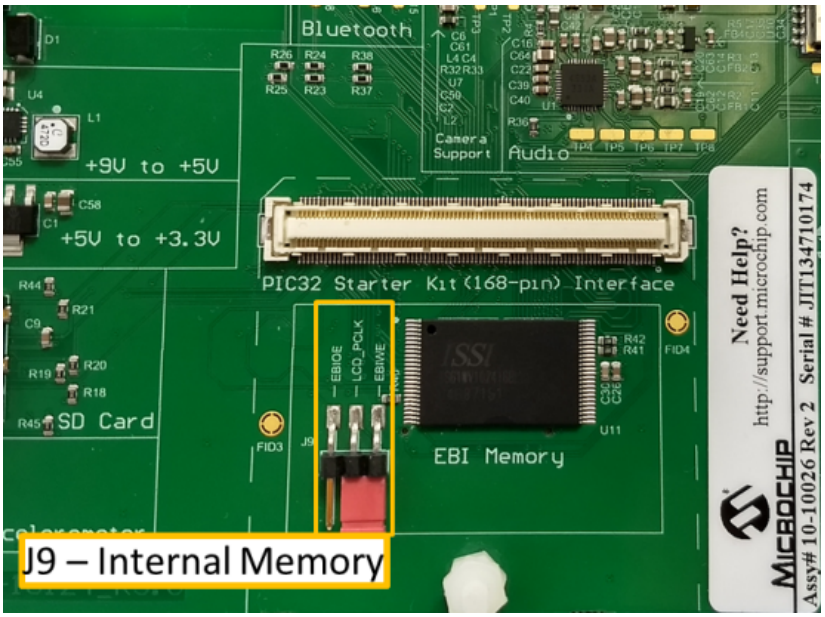
- Connect the PIC32MZ DA Starter Kit board to the MEB II board
- Power up the board by connecting the power adapter to J3 power connector on the MEB II board or a powered USB cable to the USB DEBUG port on the Starter Kit board



PIC32MZ EF Starter Kit and MEB II

Configuration: pic32mz_ef_sk_meb2

- This configuration requires that the J9 jumper be set to enable internal SRAM for the frame buffer. Set the J9 jumper to connect the EBIOE and LCD_PCLK pins. The J9 jumper is located on the bottom of the MEB II board, beneath where the starter kit is plugged into the board. Refer to the following figure for the exact location.
- Connect the PIC32MZ EF Starter Kit board to the MEB II board
- Power up the board by connecting the power adapter to J3 power connector on the MEB II board or a powered USB cable to the USB DEBUG port on the Starter Kit board

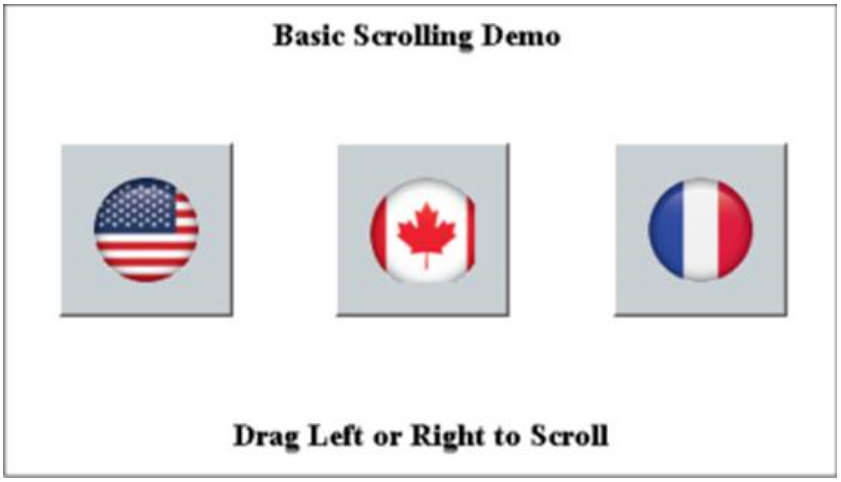


Running the Demonstration

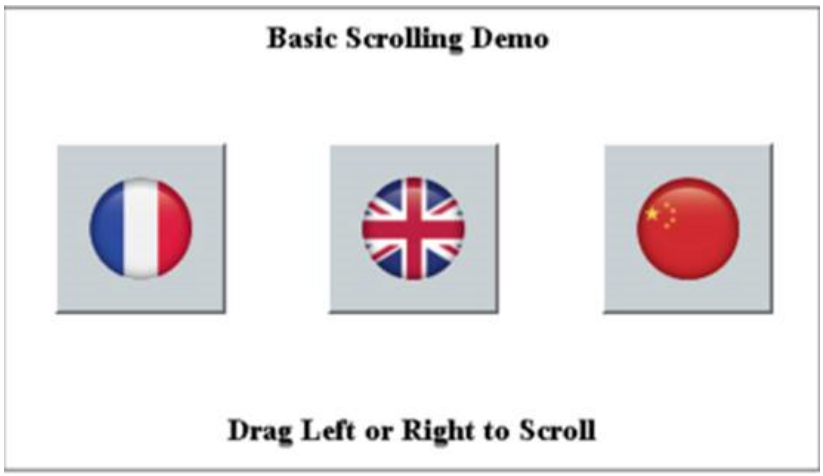
This section provides instructions about how to build and run the Aria Scrolling demonstration.

Description

When power-on is successful, the demonstration will display a similar menu to that shown in the following figure:



The user can now touch the screen and drag their finger horizontally to scroll the list of buttons. This will cause the buttons that are off-screen to become visible and allow interaction.



aria_showcase

This demonstration provides a subset of capabilities offered by the Aria Graphics Library using Low-Cost Controllerless features with touch screen capabilities.

Description

This application showcases the various widgets and advanced capabilities offered by the Aria User Interface Library. The application features an interactive menu screen where the user can access the different widget and graphics demonstrations. As a showcase application, it also features 'Demo mode', where it autonomously runs the demonstrations after a specified period of idle time (typically 5-20 seconds). After idle time-out, the application will replay a series of prerecorded touch events, repeating the replay after a delay (typically 5 seconds). Event replay can be terminated by user initiation of a touch event and will not restart until another idle time out.

Demo mode can be disabled through MHC by selecting *Harmony Framework Configuration > Graphics Stack > Use Graphics Stack? > Use Harmony Graphics Composer Suite? > Middleware > Use Aria User Interface Library> >* ***Enable Demo Mode?***. Then, reconfigure the application using MHC, rebuild, and then reload..

Architecture

The aria_showcase application uses the MPLAB Harmony Graphics Library to render graphics to the display. The graphics library draws the widgets and images to a frame buffer. For configurations that have a 2D Graphics Processing Unit (GPU), the graphics library uses the GPU to draw to the frame buffer. The contents of the frame buffer are continuously transferred to refresh the contents of the LCD display.

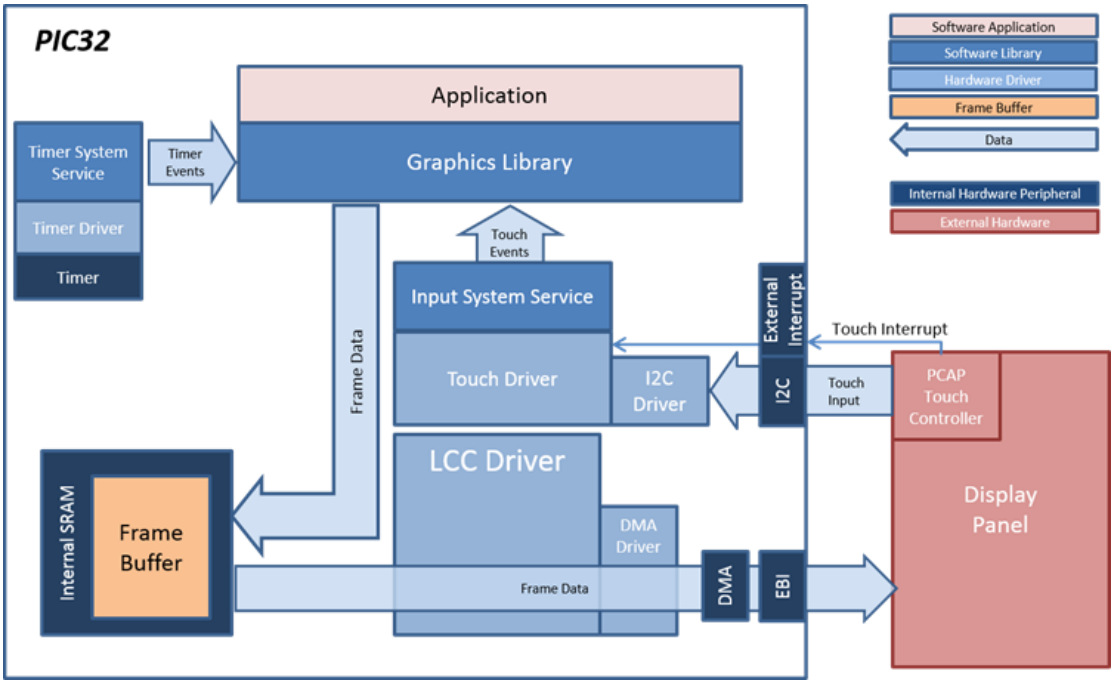
The application also features user touch input through the integrated touch screen on the display panel. Touch input from the touch controller goes through the I2C port, and the Input System Service acquires the touch input information from the Touch and I2C Drivers. The Input System Service sends touch events to the Graphics Library, which processes these events and updates the frame data accordingly.

Finally, the application uses the Timer System Service and driver to send timer events for Demo mode and for transitioning images in the Slideshow Demo at specified intervals.

The following diagrams show the various software and hardware components used by the different configurations in this application.

pic32mz_ef_sk_meb2

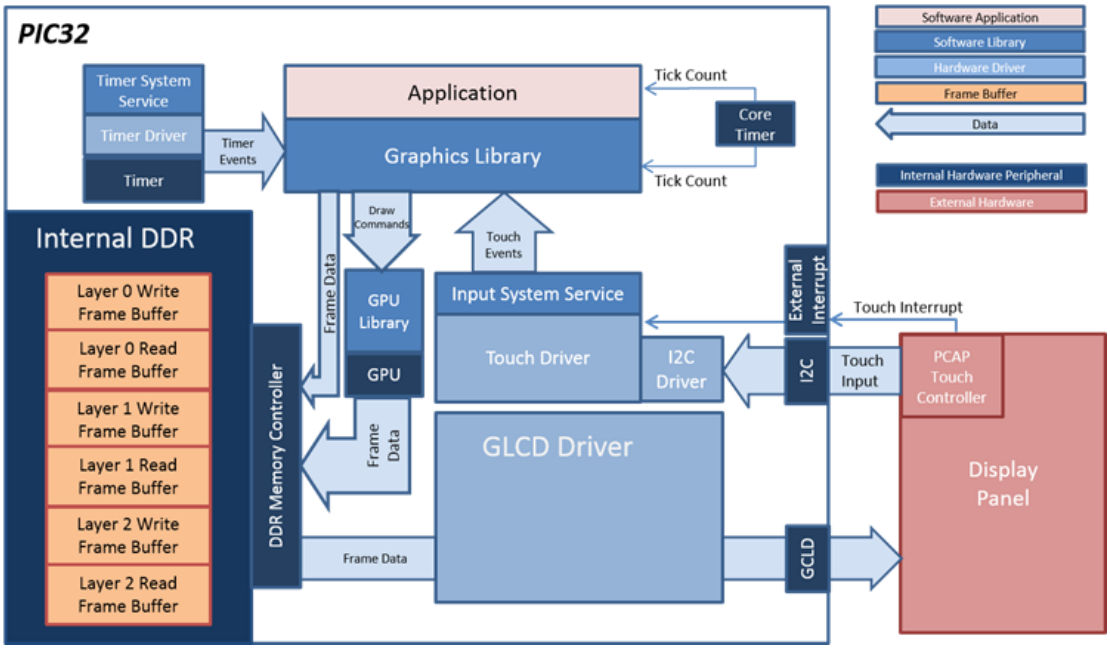
In this configuration, the frame buffer is stored in the internal SRAM. Due to the limited size of the internal SRAM, only a single frame buffer can be used which can cause tearing to be visible as the GFX library draws on the screen. These configurations use the Low Cost Controller-less (LCC) display driver to manage the DMA that transfers the framebuffer contents to the display.



pic32mz_da_sk_intddr_meb2

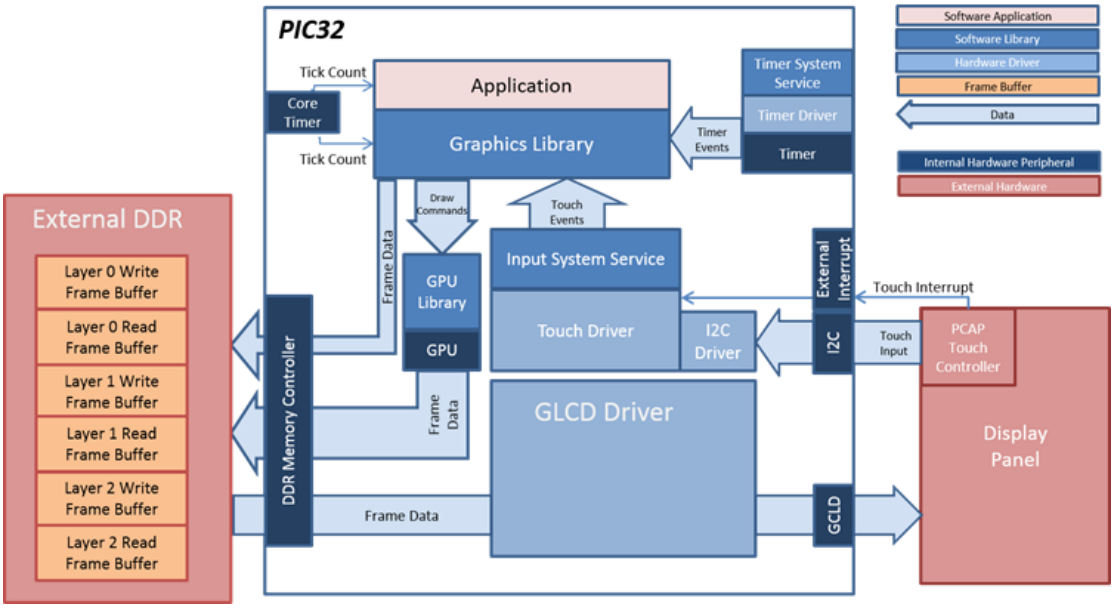
In this configuration, the frame buffers are stored in the internal DDR memory. The DDR memory is large enough to allow 3 layers with double frame buffers. Double buffering eliminates the tearing effect since the contents of the write frame buffer are not shown in the display until the GFX library is done drawing on it. When the GFX library is done drawing on the write frame buffer, it is swapped with the read frame buffer in order to be shown on the display. The contents of the read frame buffer are transferred continuously by the Graphics LCD (GLCD) controller to the display panel.

This configuration also takes advantage of the DDR by predecoding some of the images to DDR as 32-bit RAW image assets. This allows faster image rendering since the images do not have to be decoded while rendering, and it uses the GPU to directly render or ‘blit’ the images to the frame buffer at a faster rate.



pic32mz_da_sk_extddr_meb2

This configurations works similar to the pic32mz_da_sk_intddr_meb2, except that the frame buffers and pre-decoded RAW images are now stored in external DDR.



Demonstration Features

- Input System Service and Touch driver
- Timer System Service and driver
- DMA System Service
- Low-Cost Controllerless (LCC) Graphics Driver
- I2C Driver
- 16-bit RGB565 color depth support (65535 unique colors for PIC32MZ EF configurations)
- 32-bit RGBA8888 color depth support for PIC32MZ DA configurations
- JPEG and PNG images stored in internal Flash (see *Volume III: MPLAB Harmony Configurator (MHC) > MPLAB Harmony Graphics Composer User's Guide > Graphics Composer Window User Interface > Graphics Composer Asset Management >Image Assets*))
- UTF-8 and UTF-16 character font support (see *Volume III: MPLAB Harmony Configurator (MHC) > MPLAB Harmony Graphics Composer*

User's Guide > Graphics Composer Window User Interface >Graphics Composer Asset Management > Font Assets)

- Full multi-lingual font and localization (Chinese and English) (see *Volume III: MPLAB Harmony Configurator (MHC) >MPLAB Harmony Graphics Composer User's Guide > Graphics Composer Window User Interface > Graphics Composer Asset Management > Widget Tool Box Panel*)
- Graphics Demo mode (see Enabling Demo Mode in MPLAB Harmony Graphics Application)
- GFX widgets: List wheel, Touch Test, Keypad, Slideshow (see *Volume III: MPLAB Harmony Configurator (MHC) > MPLAB Harmony Graphics Composer User's Guide > Graphics Composer Window User Interface > Graphics Composer Asset Management >String Assets*)
- Alpha-blending
- Pre-decoded images in DDR (for PIC32MZ DA configurations with DDR)

Tools Setup Differences

- "Use Graphics Stack" is selected in the MPLAB Harmony Configurator (MHC), which enables the Graphics and Touch libraries and drivers for the Board Support Package (BSP).
- The heap size is set to 204800 bytes to ensure enough memory for dynamic allocations. This is done by setting the size in MHC through the following menu selection: *Device & Project Configuration >Project Configuration > XC32 (Global Options) >xc32-ld >General*.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Aria Showcase demonstration.

Description

To build this project, you must open the `aria_showcase.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/gfx/aria_showcase`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`aria_showcase.X`	`<install-dir>/apps/gfx/aria_showcase/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mz_ef_sk_meb2	pic32mz_ef_sk+meb2	Demonstration for the PIC32MZ EF Starter Kit with the Multimedia Expansion Board II (MEB II).
pic32mz_da_sk_intddr_meb2	pic32mz_da_sk_intddr+meb2	Demonstration for the PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit and the Multimedia Expansion Board II (MEB II).
pic32mz_da_sk_extddr_meb2	pic32mz_da_sk_extddr+meb2	Demonstration for the PIC32MZ Embedded Graphics with External DRAM (DA) and the Multimedia Expansion Board II (MEB II).

Note: This application contains custom code that is marked by the comments "`// START OF CUSTOM CODE ...`" and "`// END OF CUSTOM CODE`". If you use the MPLAB Harmony Configurator to regenerate the application code, use the "Prompt Merge For All Differences" merging strategy and do not remove or replace the custom code.

Configuring the Hardware

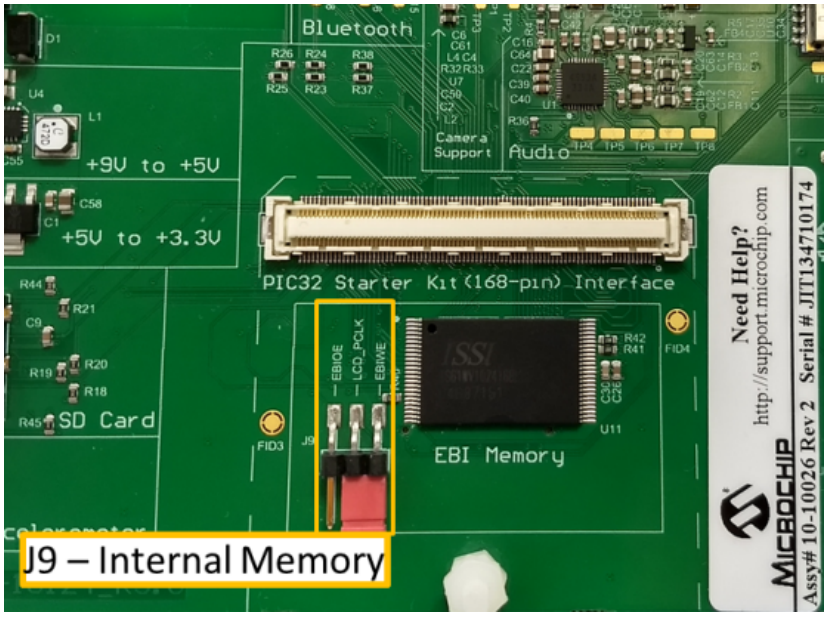
This section describes how to configure the supported hardware.

Description

PIC32MZ EF Starter Kit and MEB II

Configuration: pic32mz_ef_sk_meb2

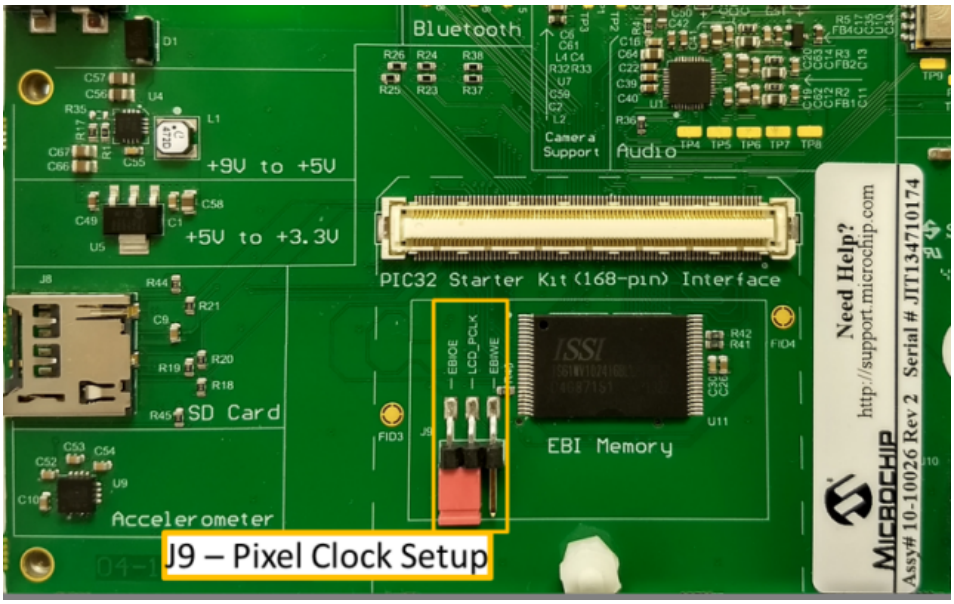
- The J9 Jumper on the MEB II board, located beneath the PIC32MZ EF Starter Kit, should be configured to support using internal SRAM for the graphics frame buffer, as shown in the following figure.
- Connect the PIC32MZ EF Starter Kit board to the MEB II board
- Power up the board by connecting the power adapter to J3 power connector on the MEB II board or a powered USB cable to the USB DEBUG port on the Starter Kit board



PIC32MZ Embedded Graphics with External DRAM (DA) Starter Kit or PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit and MEB II

Configurations: pic32_mz_da_sk_intddr_meb2, pic32_mz_da_sk_extddr_meb2

- On the MEB II, the EBIOE and LCD_PCLK (J9) must be jumpered. A connection establishes the GLCD's pixel clock output timing. The external SRAM memory on the board is disabled. The jumper (J9) is available on the bottom side of the MEB II board under the starter kit. The J9 jumper is located on the bottom of the MEB II board, beneath where the starter kit is plugged into the board. Refer to the following image for the exact location.
- Connect the PIC32MZ DA Starter Kit board to the MEB II board.
- Power up the board by connecting the power adapter to J3 power connector on the MEB II board or a powered USB cable to the USB DEBUG port on the Starter Kit board



Running the Demonstration

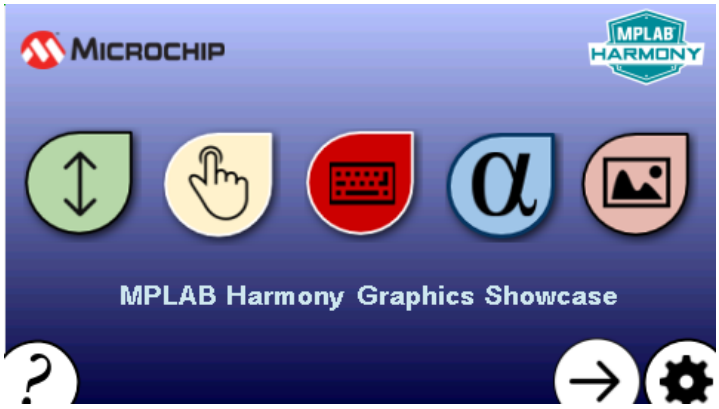
This section provides instructions on how to use the Aria Showcase demonstration.

Description

Upon boot-up, the application displays the animation Splash Screen, which shows an animated splash bar while blending in the MPLAB Harmony and Microchip PIC32 logos, as shown in the following figure.

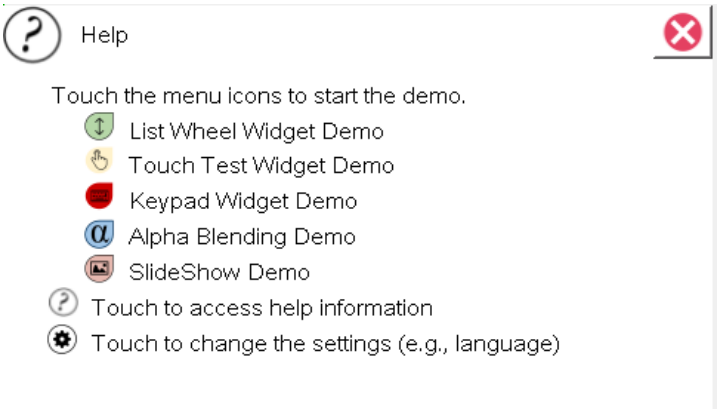


When power-on is successful, the application Home Screen will be displayed.



On the Home Screen, touching the large icons will open various screens that demonstrate the functionality of the widgets in the Aria graphics library.

Touching the Help (?) button on the lower left corner of each screen will show help information similar to the following figure.



The Settings Screen shows an option to switch the language between English and Simplified Chinese and demonstrates the Aria string library's capability to support multi-lingual fonts and symbols.

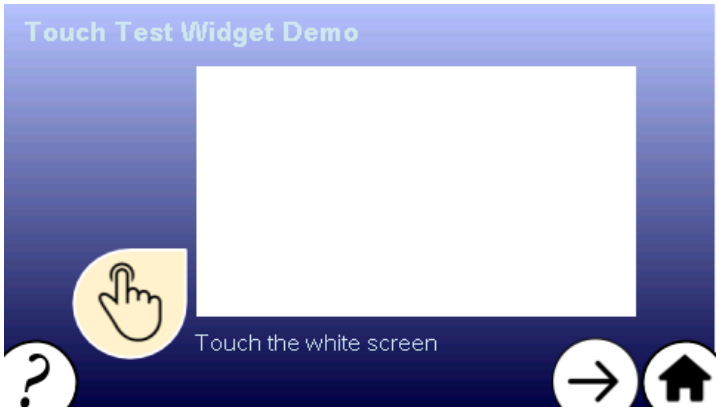


The List Wheel Widget Demo screen shows an example on how the list wheel widget can be used to provide UI controls for changing the time. The following figures show the List Wheel Widget Demo screen on PIC32MZ DA and PIC32MZ EF, respectively. The List Wheel widget on the PIC32MZ DA configurations makes use of the GPU to feature effects like zoom and auto-hide.

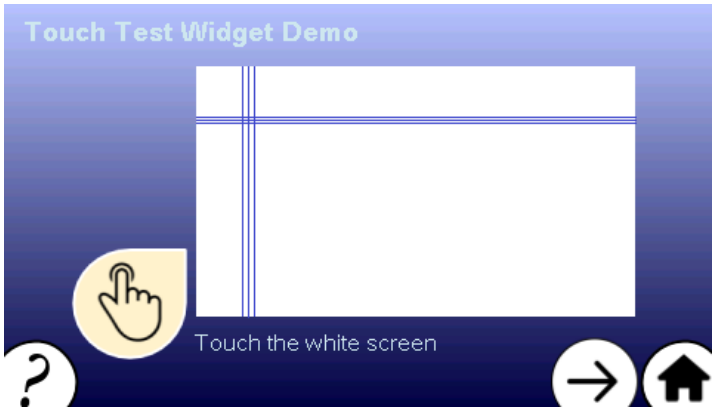




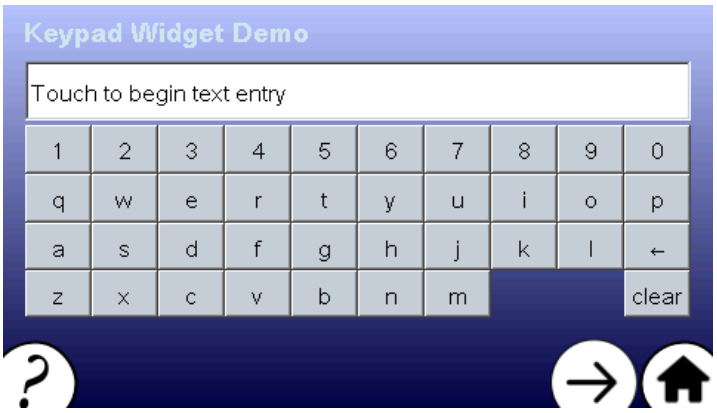
The Touch Test Widget Demo screen shows the touch screen functionality.



When the white active area of the Touch Test Widget is touched, intersecting horizontal and vertical lines indicate the touch points.



The Keypad Widget Demo Screen shows how the keypad and text entry widgets can be used to provide an interface for obtaining user text input.



The Alpha Blending Demo screen shows the alpha blending capabilities of the Aria User Interface Library. The demonstration features two JPEG images that are alpha blended on top of each other. The (+) and (-) buttons and the slider widget on the right side of the images provides a way to change the alpha amounts.

Alpha blending at 50%



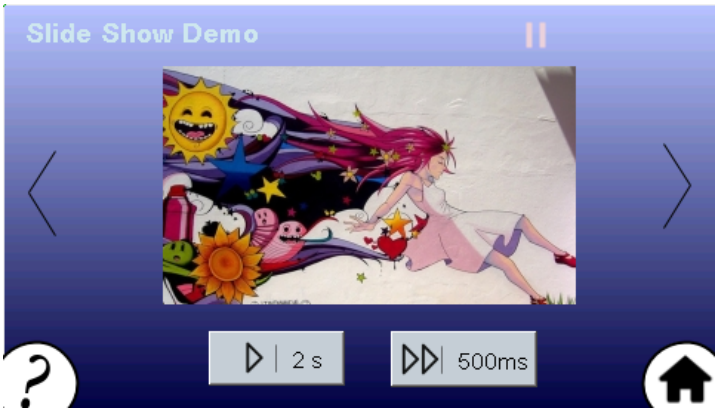
Alpha blending at 100%



Alpha blending at 0%



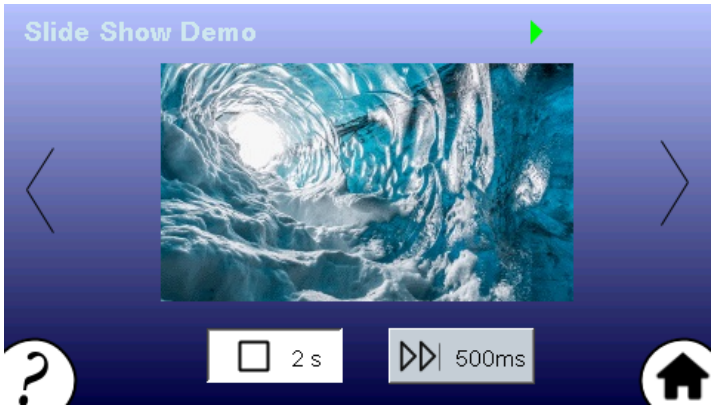
The Slide Show Demo Screen features the slide show widget being used to transition between several JPEG images using the available controls.



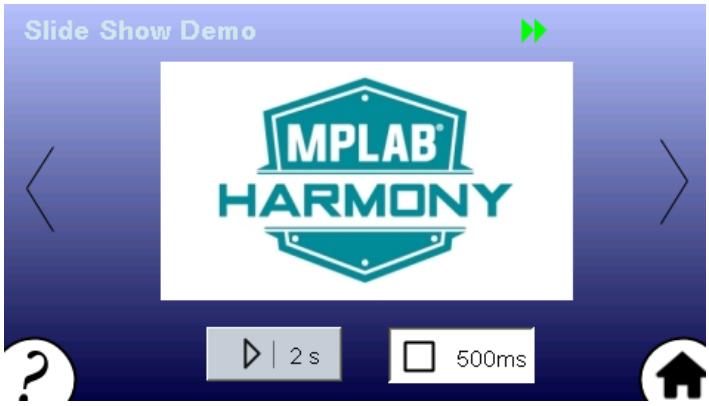
Touching the LEFT (<) and RIGHT (>) buttons manually transitions through the list of images. Touching the PLAY (|>) and FAST-FORWARD (|>>)

buttons will automatically transition the images at 2s and 500ms intervals, respectively. The transitions are triggered using events from the Timer System Service.

The following figure shows Slide Show widget in Play mode.



The following figure shows the Slide Show widget in FAST-FORWARD mode.



Help information for the demonstration screens can be accessed by touching the (?) button on the lower left corner of each screen. Touching the HOME button on the lower right corner takes the application back to the Home Screen.

aria_showcase_reloaded

This application showcases the circular (arc, circular slider, circular gauge), graphing (pie chart, bar graph, line graph) and radial menu widgets. These widgets are ready-to-use within the Harmony Graphics Composer suite.

Description

This application showcases the circular (arc, circular slider, circular gauge), graphing (pie chart, bar graph, line graph) and radial menu widgets.

- Arc – a primitive drawing widget that can be used to draw filled arcs or circles
- Circular Slider – a circular widget that takes user input to set or show a value within a specified range
- Circular Gauge – a circular widget that shows a value within a range using a needle and tick marks
- Pie Chart – a graphing widget that shows data values as sectors in a circle
- Bar Graph – a graphing widget that shows data values in categories using rectangular bars
- Line Graph – a graphing widget that shows data values in categories using points and lines
- Radial Menu - a widget that shows revolving menu items. For optimum performance, this widget uses the 2D GPU and is only available on PIC32MZ DA configurations.

Each widget is demonstrated on a separate application screen. Touching the screen or an applicable widget button will show each widget change in form or value.

The application has Demo Mode enabled, and will run autonomously if no touch input is detected after 20 seconds.

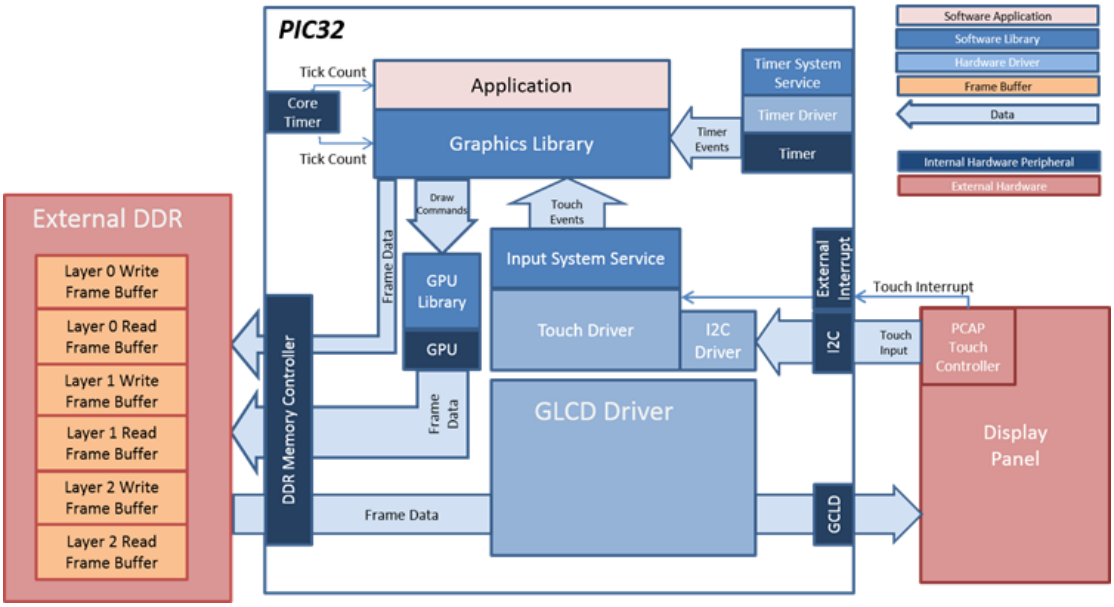
Architecture

The following block diagrams show the various software and hardware blocks used in this application.

In all configurations, the graphics library renders the widgets to the frame buffer(s), which is periodically refreshed to the display. The graphics library also processes user input via the touch screen and sends events to the application code which updates the widgets appropriately.

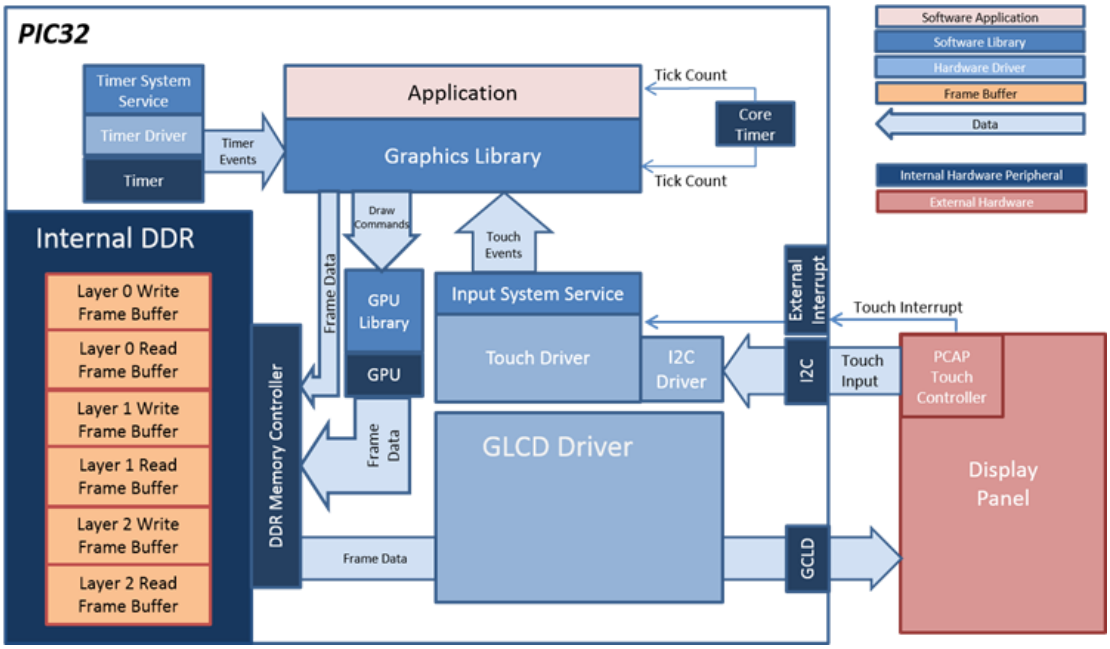
pic32mz_da_sk_extddr_meb2

In this configuration, three pairs of 32-bit RGBA8888 frame buffers are stored in external DDR, one pair for each layer. The graphics library uses double-buffering to eliminate tearing artifacts when rendering. The graphics library renders the widgets to the write frame buffers and uses the GPU to speed up the rendering. Once the rendering is done, the write frame buffer is swapped with the read frame buffer, and the GLCD sends the new frame data to the display.



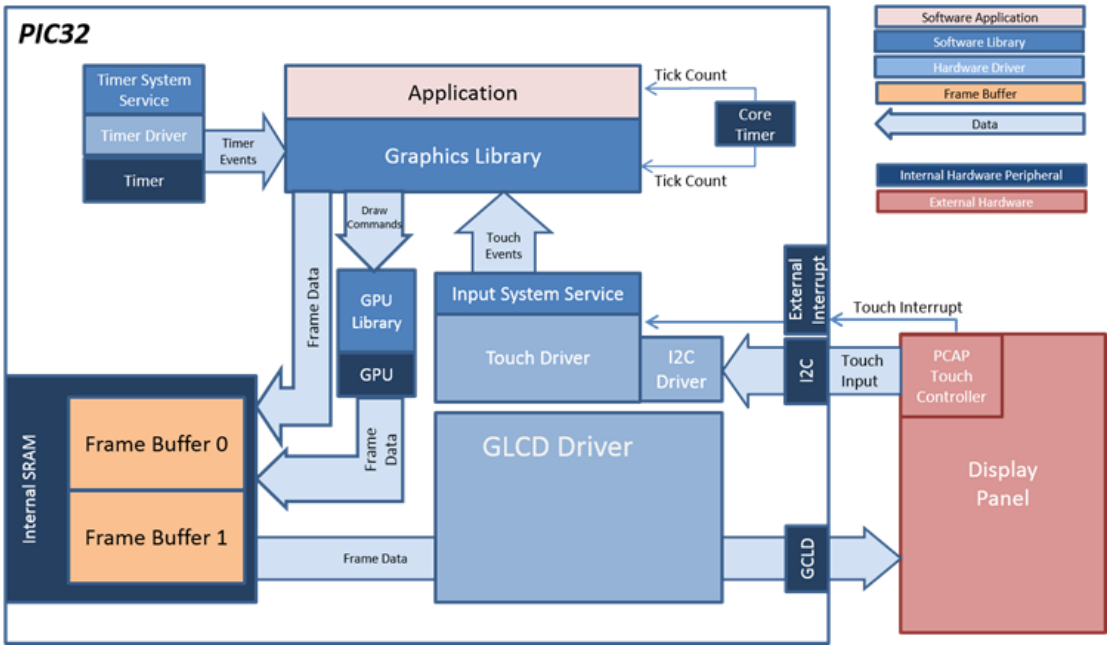
pic32mz_da_sk_intddr_meb2

In this configuration, three pairs of 32-bit RGBA8888 frame buffers are stored in internal DDR, one pair for each layer. The graphics library uses double-buffering to eliminate tearing of artifacts when rendering. The graphics library renders the widgets to the write frame buffers and uses the GPU to speed up the rendering. Once the rendering is done, the write frame buffer is swapped with the read frame buffer and the GLCD sends the new frame data to the display.



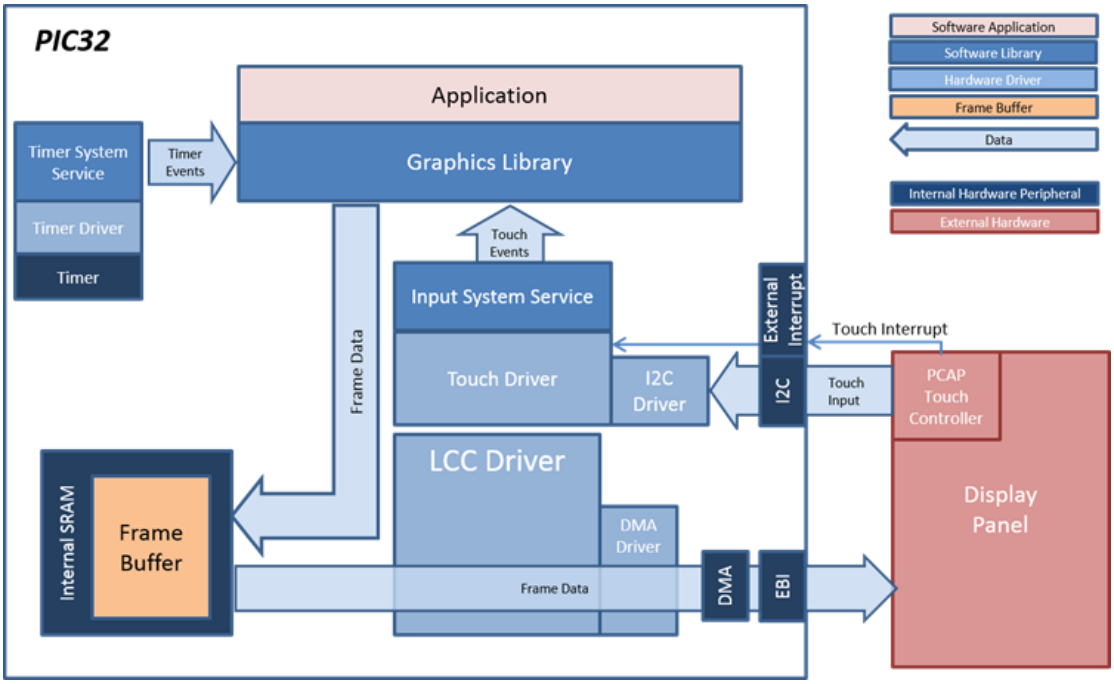
pic32mz_da_sk_noddr_meb2

This PIC32MZ DA configuration does not have DDR memory to store multiple 32-bit frame buffers. To support double buffering and eliminate tearing effects, two 16-bit RGB565 frame buffers are stored in the internal SRAM, and only one layer is used. The graphics library renders the widgets to the write frame buffer, and the GPU is used to speed up the rendering. Once the rendering is done, the write frame buffer is swapped with the read frame buffer. and the GLCD sends the new frame data to the display.



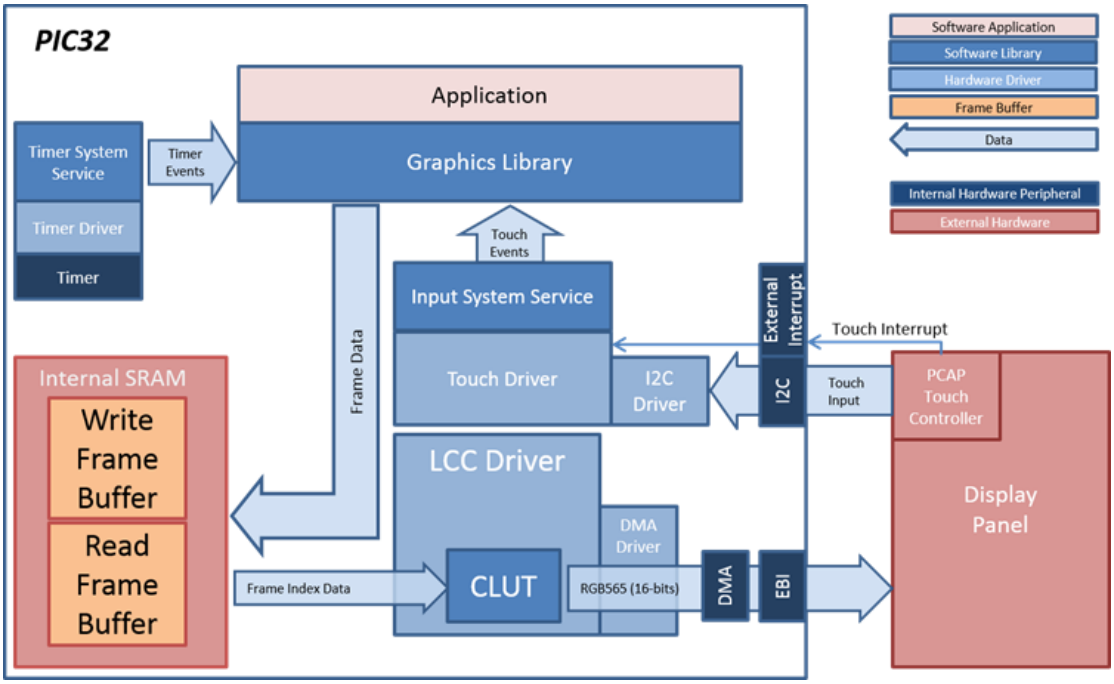
pic32mz_ef_sk_meb2

In this configuration, the frame buffer is in the internal SRAM. Due to the limited size of the internal SRAM, only a single 16-bit RGB565 frame buffer can be used. With single-buffering, tearing will be visible when the graphics library updates the widgets, and the LCC (low-cost controllerless) driver pushes the frame buffer data to the display.



pic32mz_ef_sk_meb2_lut8

Since two 16-bit RGB565 buffers cannot fit into the internal device SRAM, this configuration supports uses two 8-bit buffers to support double-buffering. The 8-bit buffers contain the indices of 16-bit colors in a palette lookup table (LUT). The graphics library renders the widgets by writing the index of a pixel color into the buffer. During a display line refresh, the LCC (low-cost controllerless) driver performs a palette lookup to convert the 8-bit indices to their 16-bit color equivalent for each line, and then writes the line data to the display via the DMA. Using the 8-bit palletized buffers allows for double-buffering and eliminates tearing during rendering. However, the lookup table conversion requires extra processing time and reduces performance.



Demonstration Features

- Circular widgets – arc, circular slider, circular gauge
- Graphing widgets – pie chart, bar graph, line graph
- Radial Menu widget (PIC32MZ DA)
- Timer HW and Timer System Service
- Image compression techniques using Run-Length Encoding
- Double-buffering using LUT (lookup-table) based frame buffers

Tools Setup Differences

- Due to the small available memory in SRAM for the pic32mz_da_sk_noddr configuration, reduce the heap size to 48000. This is done by setting the size in MHC through the following menu selection: *Device & Project Configuration >Project Configuration > XC32 (Global Options) >xc32-ld >General.*
- For the pic32mz_ef_sk_meb2_lut8 configuration, the graphics library and the LCC driver need be configured to use the global palette LUT. In the Harmony Graphics Composer, check *Tools -> Global Palette -> Enable Global Palette*. In MHC, set *Harmony Framework Configuration > Graphics Stack > Graphics Controller > Low Cost Controllerless > Frame Buffer Mode > Double Buffer*, and check Palette Mode.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Aria Showcase Reloaded demonstration.

Description

The parent directory for this application is *<install-dir>/apps/gfx/aria_showcase_reloaded*. To build this application, open the *<install-dir>/apps/gfx/aria_showcase_reloaded /firmware/aria_showcase_reloaded.X* project file in MPLABX IDE and then select the desired configuration for your board.

The following table lists and describes the supported configurations.

Project Configuration Name	BSP Used	Description
pic32_mz_da_sk_extddr_meb2	pic32mz_da_sk_extddr+meb2	Pic32MZ DA Starter Kit with external DDR, Expansion Board II (MEB II) and 4.3” WQVGA (480x272) Display.
pic32_mz_da_sk_intddr_meb2	pic32mz_da_sk_intddr+meb2	Pic32MZ DA Starter Kit with internal DDR, Expansion Board II (MEB II) and 4.3” WQVGA (480x272) Display.
pic32_mz_da_sk_noddr_meb2	pic32mz_da_sk_noddr+meb2	PIC32MZ with Disabled DRAM (DA) Starter Kit, Multimedia Expansion Board II (MEB II) and 4.3” WQVGA (480x272) Display.

pic32mz_ef_sk_meb2	pic32mz_ef_sk+meb2	Pic32MZ EF Starter Kit with Expansion Board II (MEB II) and 4.3" WQVGA (480x272) Display.
pic32mz_ef_sk_meb2_lut8	pic32mz_ef_sk+meb2	Pic32MZ EF Starter Kit with Expansion Board II (MEB II) and 4.3" WQVGA (480x272) Display.

Note: This application may contain custom code that is marked by the comments "`// START OF CUSTOM CODE ...`" and "`// END OF CUSTOM CODE`". If you use the MPLAB Harmony Configurator to regenerate the application code, use the "Prompt Merge For All Differences" merging strategy and do not remove or replace the custom code.

Configuring the Hardware

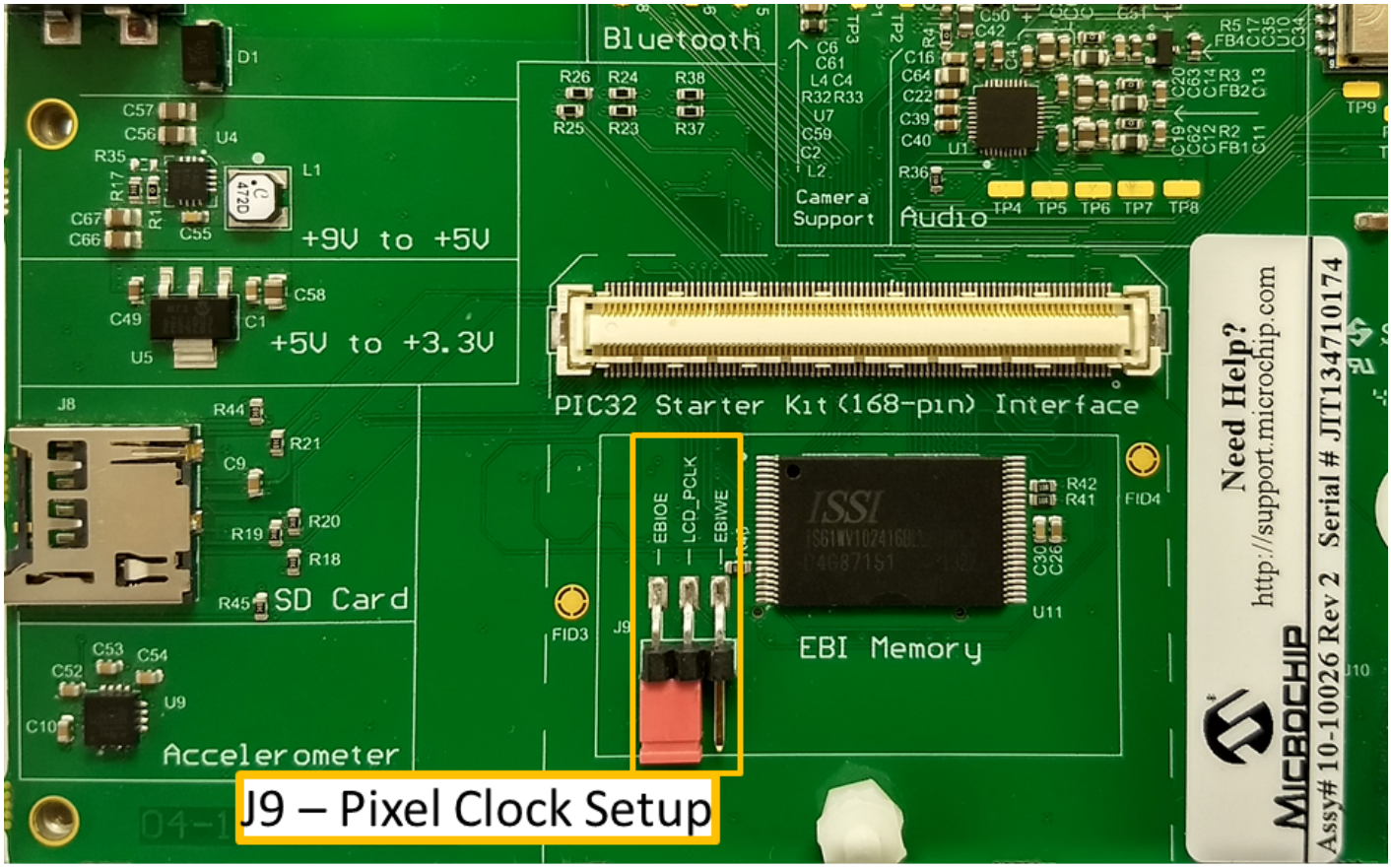
This section describes how to configure the supported hardware.

Description

PIC32MZ DA Starter Kit and MEB II

Configurations: pic32_mz_da_sk_intddr_meb2, pic32_mz_da_sk_intddr_meb2, pic32_mz_da_sk_noddr_meb2

- On the MEB II, the EBIOE and LCD_PCLK (J9) must be jumpered. A connection establishes the GLCD's pixel clock output timing. The external SRAM memory on the board is disabled. The jumper (J9) is available on the bottom side of the MEB II board under the starter kit. The J9 jumper is located on the bottom of the MEB II board, beneath where the starter kit is plugged into the board. Refer to the following image for the exact location.

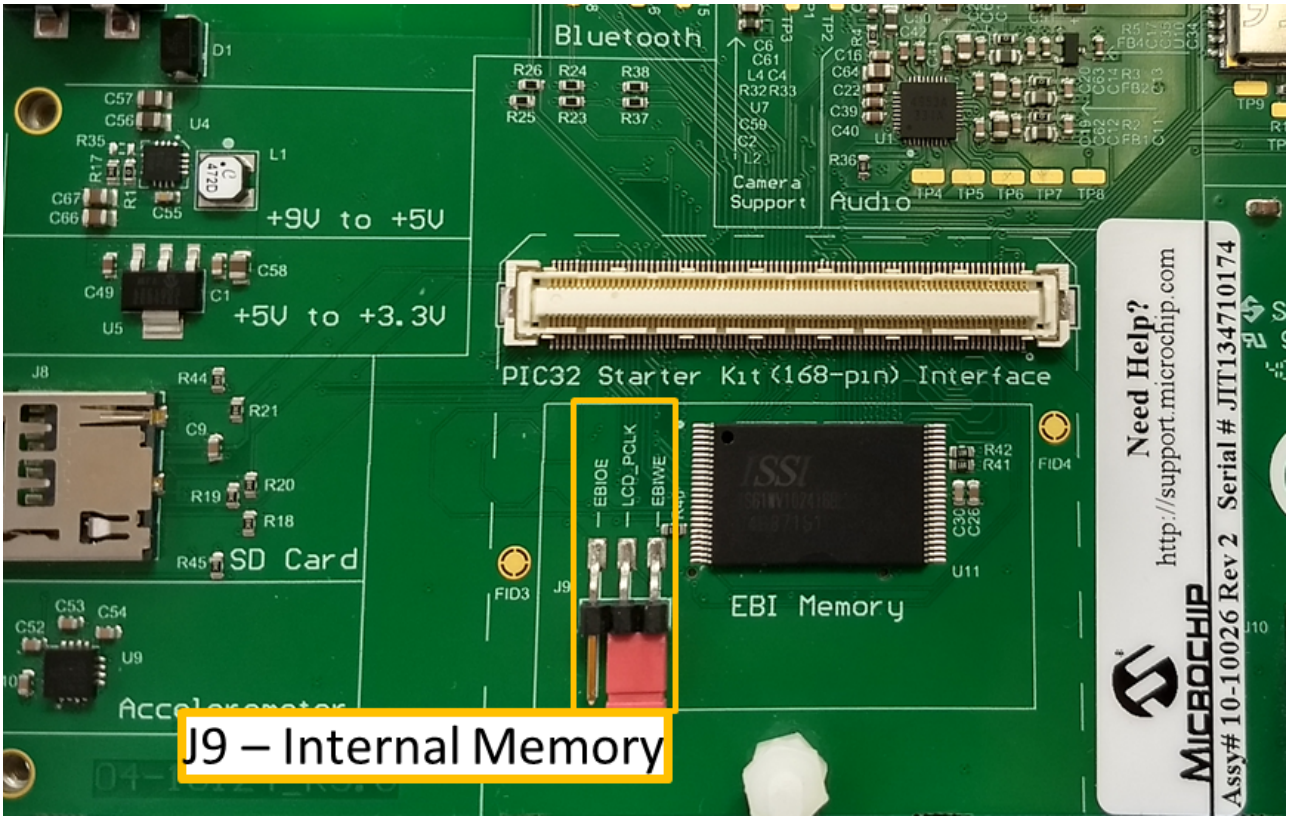


- Connect the PIC32MZ DA Starter Kit board to the MEB II board.
- Power up the board by connecting the power adapter to J3 power connector on the MEB II board or a powered USB cable to the USB DEBUG port on the Starter Kit board

PIC32MZ EF Starter Kit and MEB II

Configuration: pic32mz_ef_sk_meb2, pic32mz_ef_sk_meb2_lut8

- The J9 Jumper on the MEB II board, located beneath the PIC32MZ EF Starter Kit, should be configured to support using internal SRAM for the graphics frame buffer, as shown in the following figure.



- Connect the PIC32MZ EF Starter Kit board to the MEB II board
- Power up the board by connecting the power adapter to the J3 power connector on the MEB II board or a powered USB cable to the USB DEBUG port on the Starter Kit board

Running the Demonstration

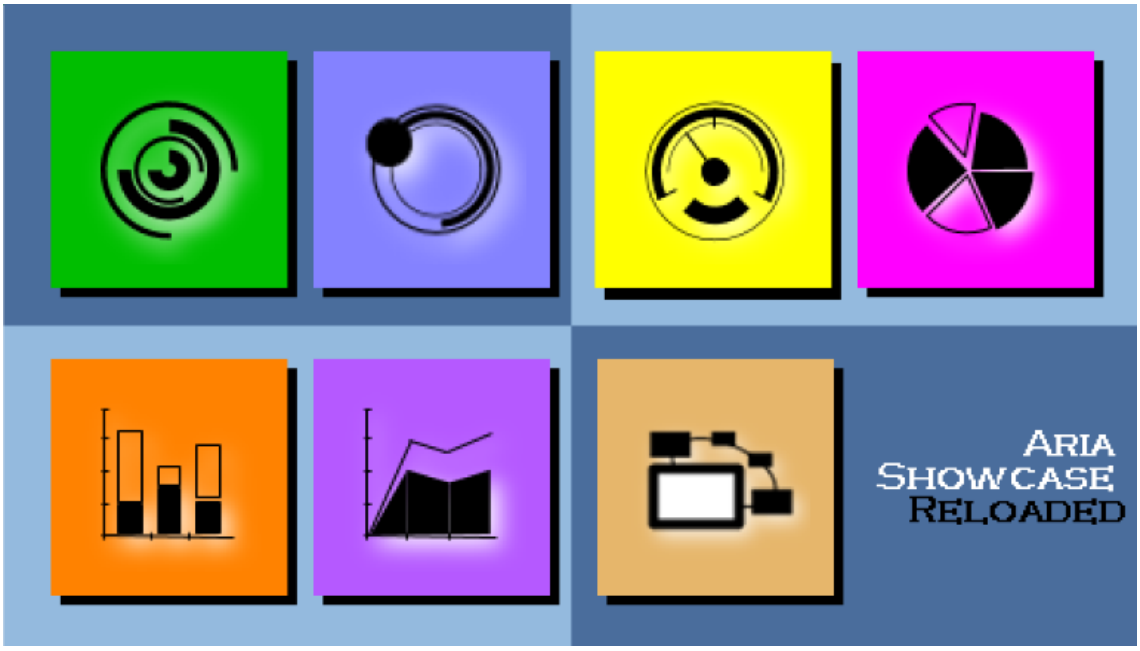
This section provides information on how to run and use the application.

Description

On start-up, the application will display a splash screen.



After the splash screen completes, the Main Menu will display.



The Main Menu buttons operate as follows:

Button	Description
	Arc Demo Button – Opens the Arc Widget Demo Screen.
	Circular Slider Demo Button – Opens the Circular Slider Demo Screen.
	Circular Gauge Demo Button – Opens the Circular Gauge Demo Screen.
	Pie Chart Demo Button – Opens the Pie Chart Demo Screen.
	Bar Graph Demo Button – Opens the Bar Graph Demo Screen
	Line Graph Demo Button – Opens the Line Graph Demo Screen
	Radial Menu Demo Button - Opens the Radial Menu Demo Screen (PIC32MZ DA only)

Touching one of the buttons above opens the corresponding widget demonstration screen. Each demonstration screen will have the following buttons:

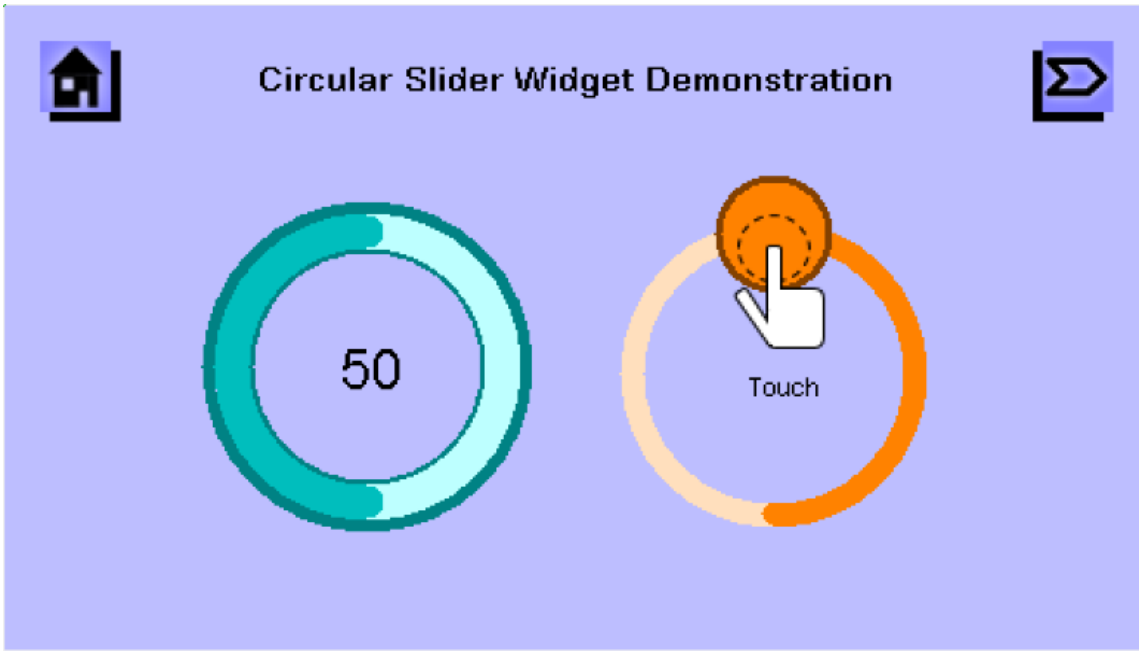
Button	Description
	Home Button – Opens the Main Menu Screen
	Next Button – Opens the Next Demo Screen

Touching the Arc Demo Button on the Main Menu Screen brings up the Arc Widget Demo Screen, as shown in the following figure. Touching the screen will show animated, rotating concentric arcs and circles. This demonstrates the graphics library's ability to draw arcs of varying thickness, angles and colors.

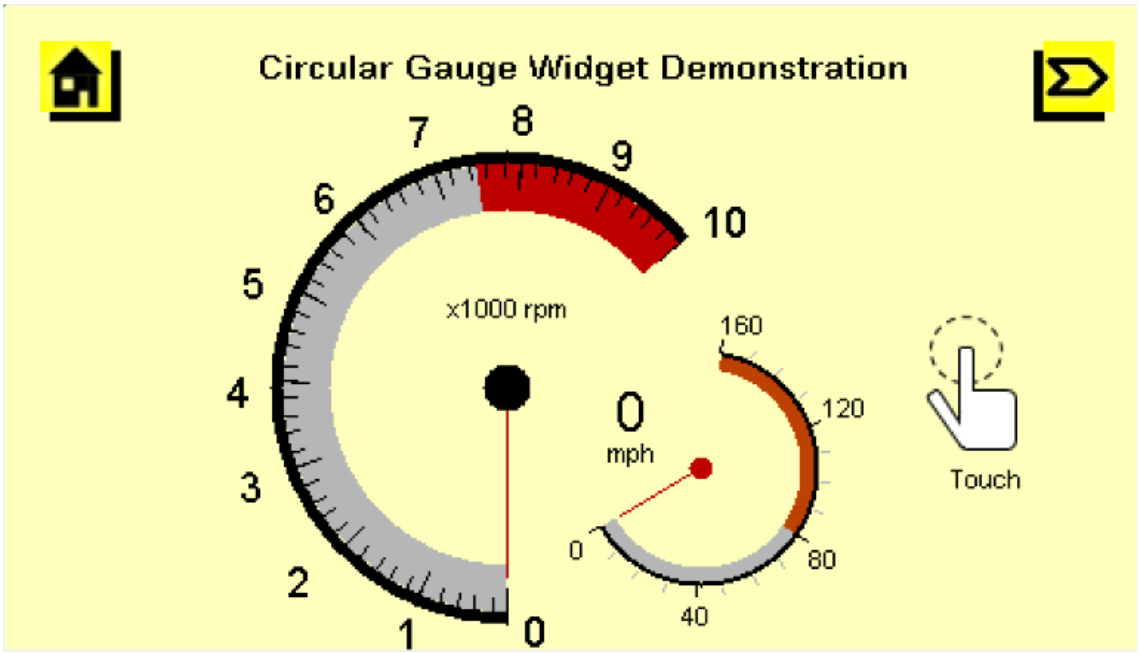


Touching the Circular Slider Button on the Main Menu or the Next Button on the Arc Widget Demo Screen will open the Circular Slider Demo Screen, as shown below. The screen contains a Circular Slider widget on the right, and Circular Progress Bar widget on the left. The Circular Progress Bar widget is a form of the Circular Slider widget with no button and does not respond to touch input.

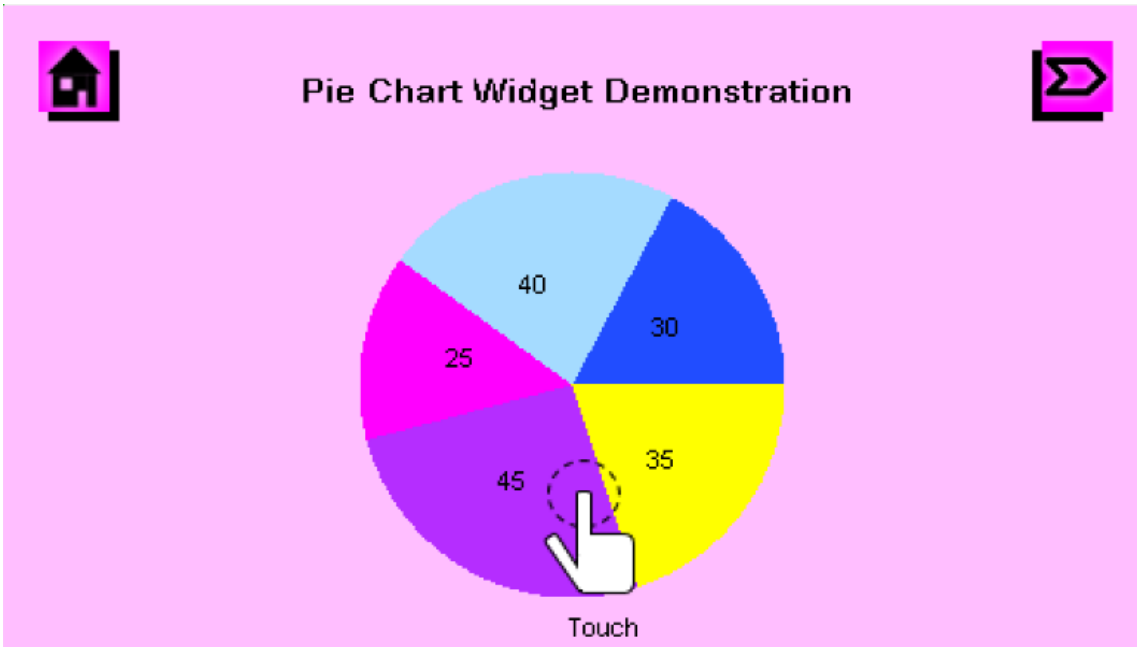
Touching and moving the button on the Circular Slider widget clockwise or counter-clockwise, will decrease or increase its value. This value is assigned to the Circular Progress Bar widget, which updates based on the set value, and to a label widget inside it to show the numeric value.



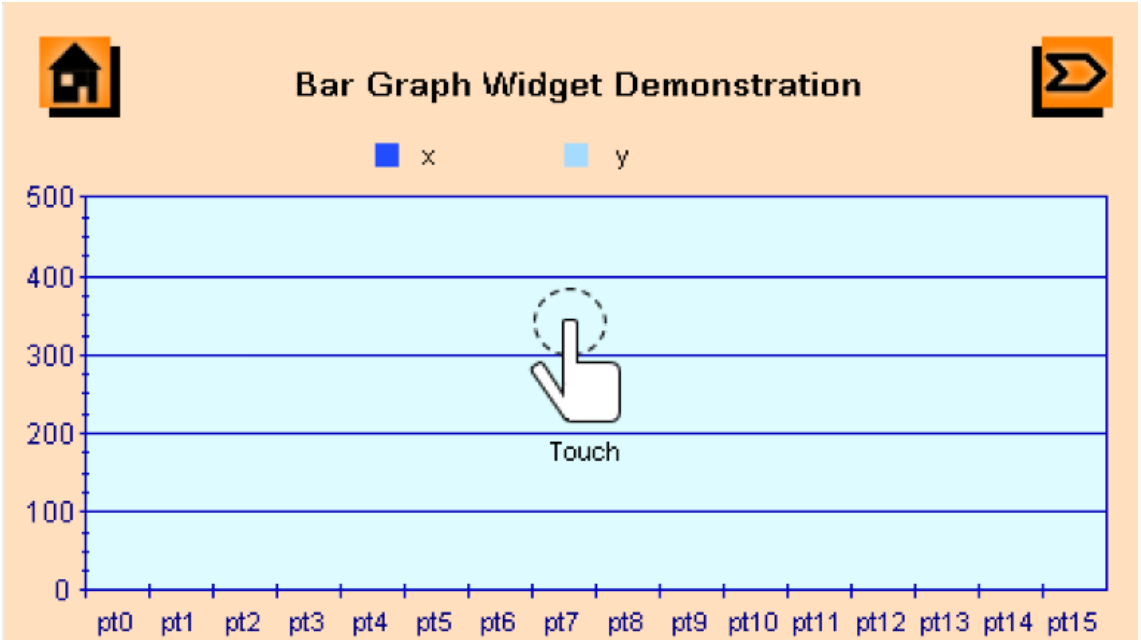
Touching the Circular Gauge button on the Main Menu screen or the Next button on the Circular Slider Demo screen opens the Circular Gauge Demo Screen, as shown in the following figure. This simulates a vehicle dashboard using two Circular Gauge widgets as tachometer and speedometer. Touching the screen increases the value of the two gauges.



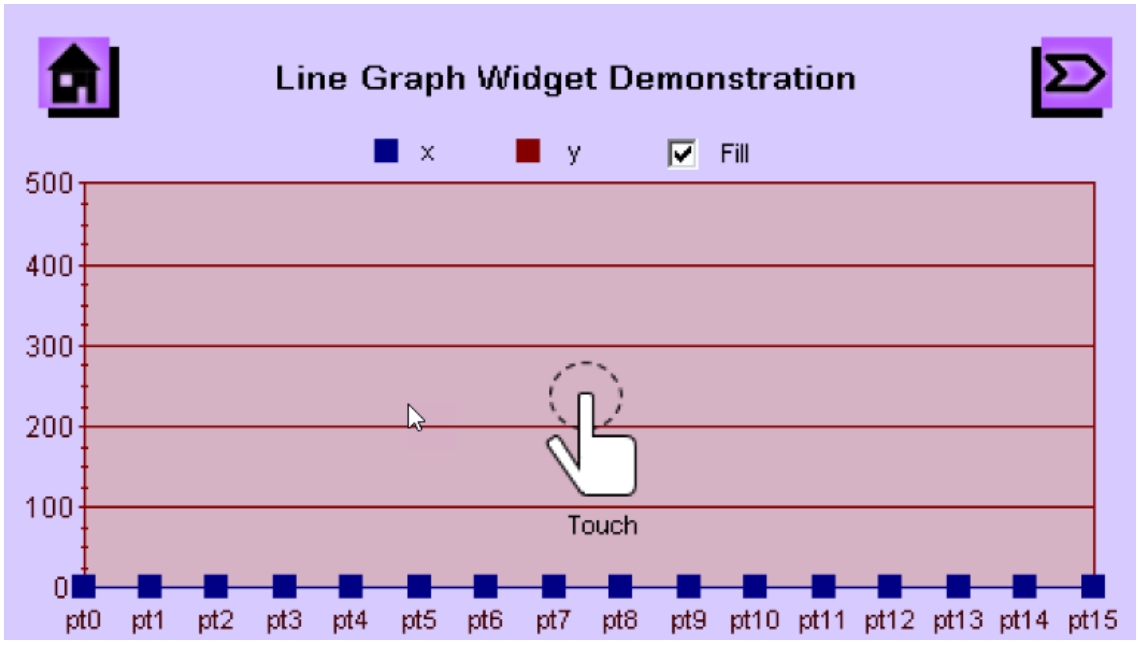
Touching the Pie Chart button on the Main Menu or the Next button on the Circular Gauge Widget Demo screen opens the Pie Chart Demo Screen, as shown in the following figure. Touching a slice on the Pie Chart widget will change the slice's radius and offset from the center.



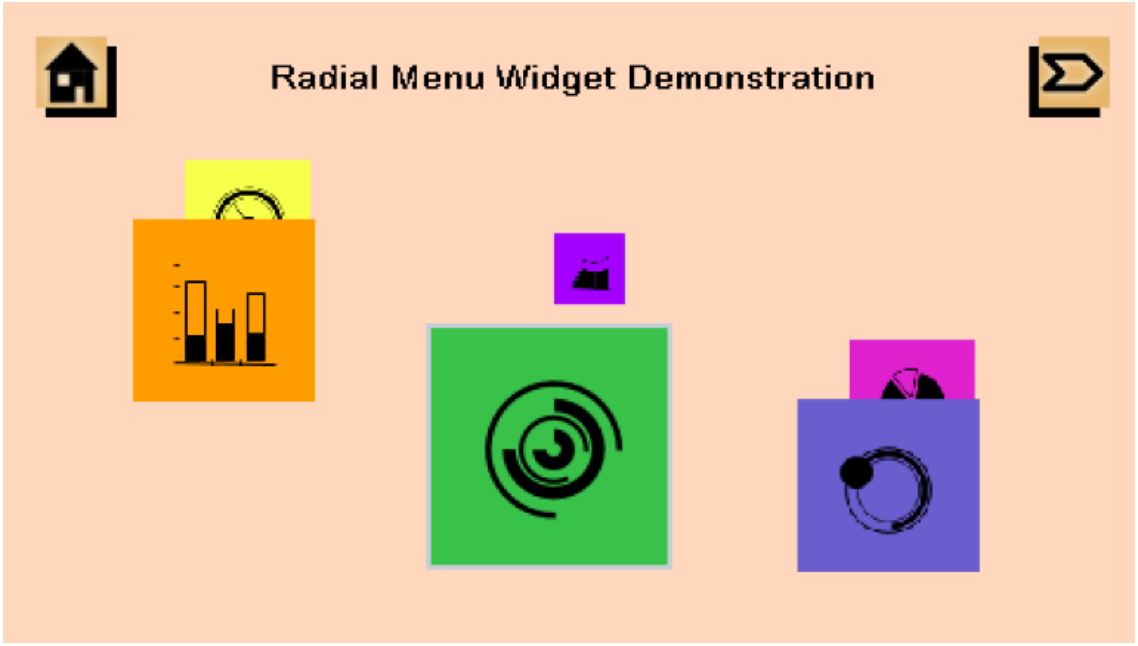
Touching the Bar Graph Button on the Main Menu, or the Next Button on the Pie Chart Demo Screen will open the Bar Graph Demo Screen, as shown in the following figure. Touching the screen will plot the x and y coordinates of the touch point in the bar graph widget.



Touching the Line Graph Button on the Main Menu or the Next Button on the Bar Graph Demo Screen will open the Line Graph Demo Screen, as shown in the following figure. Touching the screen will plot the x and y coordinates of the touch points in the line graph widget.



On PIC32MZ DA configurations, touching the Radial Menu Button on the Main Menu or the Next Button on the Line Graph Demo Screen will open the Radial Menu Demo Screen, as shown in the following figure. Touching and moving your finger on the screen will rotate the items on the menu. Touching the focused item in the radial menu will open the corresponding demo screen.



aria_splash_screen

The demonstration provides a splash screen. It is preconfigured with the graphics stack and touch and is intended to serve as a development starting point.

Description

This application showcases how a splash screen involving cross-fades and simple motion can be achieved using the Aria User Interface Library.

A single-layered version is supported by the pic32mz_ef_sk_meb2 and the pic32mz_ef_sk_meb2_ext configurations.

A multi-layered version is supported by the pic32mz_da_sk_extddr_meb2, pic32mz_da_sk_extddr_meb2_wvga, pic32mz_da_sk_intddr_meb2, and pic32mz_da_sk_intddr_meb2_wvga versions.

Although not demonstrated explicitly, all configurations are preconfigured to support touch.

Architecture

The following diagrams show the various software and hardware components for each configuration.

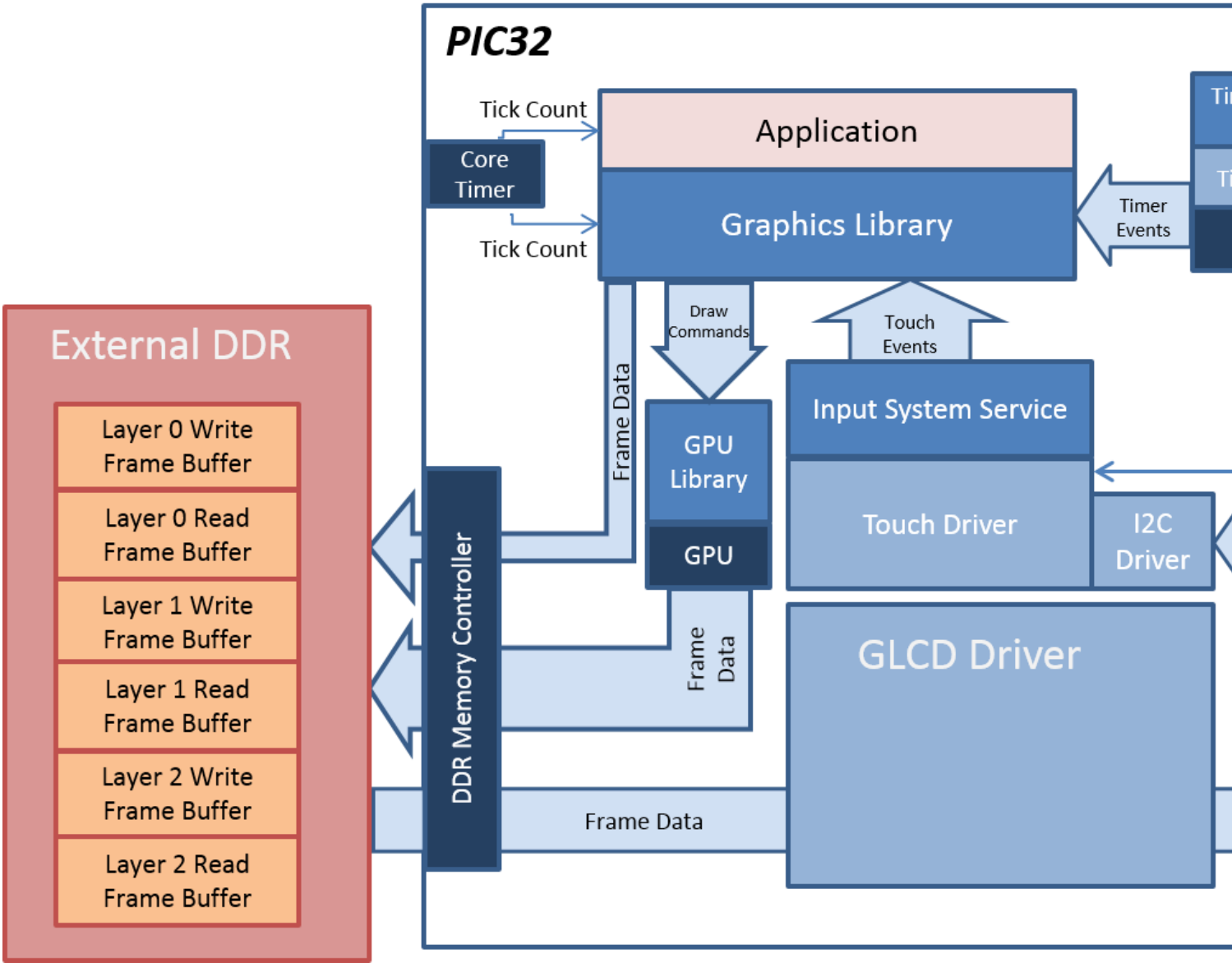
pic32mz_da_sk_extddr_meb2

pic32mz_da_sk_extddr_meb2_wvga

For these configurations, the application uses the Graphics Library to render graphics to the display. The Graphics Library passes draw commands into the GPU Library, which in turn draws the widgets and images to the frame buffer that is stored in an external DDR2. Through the DDR2 Memory Controller, the GLCD display controller peripheral continuously transfers frame data from the frame buffer onto to the LCD display.

The application also features user touch input through the integrated touch screen on the display panel. Touch input from the touch controller goes through the I2C port, and the Input System Service acquires the touch input information from the touch and I2C drivers. The Input System Service sends touch events to the Graphics Library, which processes these events and updates the frame data accordingly.

The core timer is used by the application and the Graphics Library to manage the movement of the splash screen.



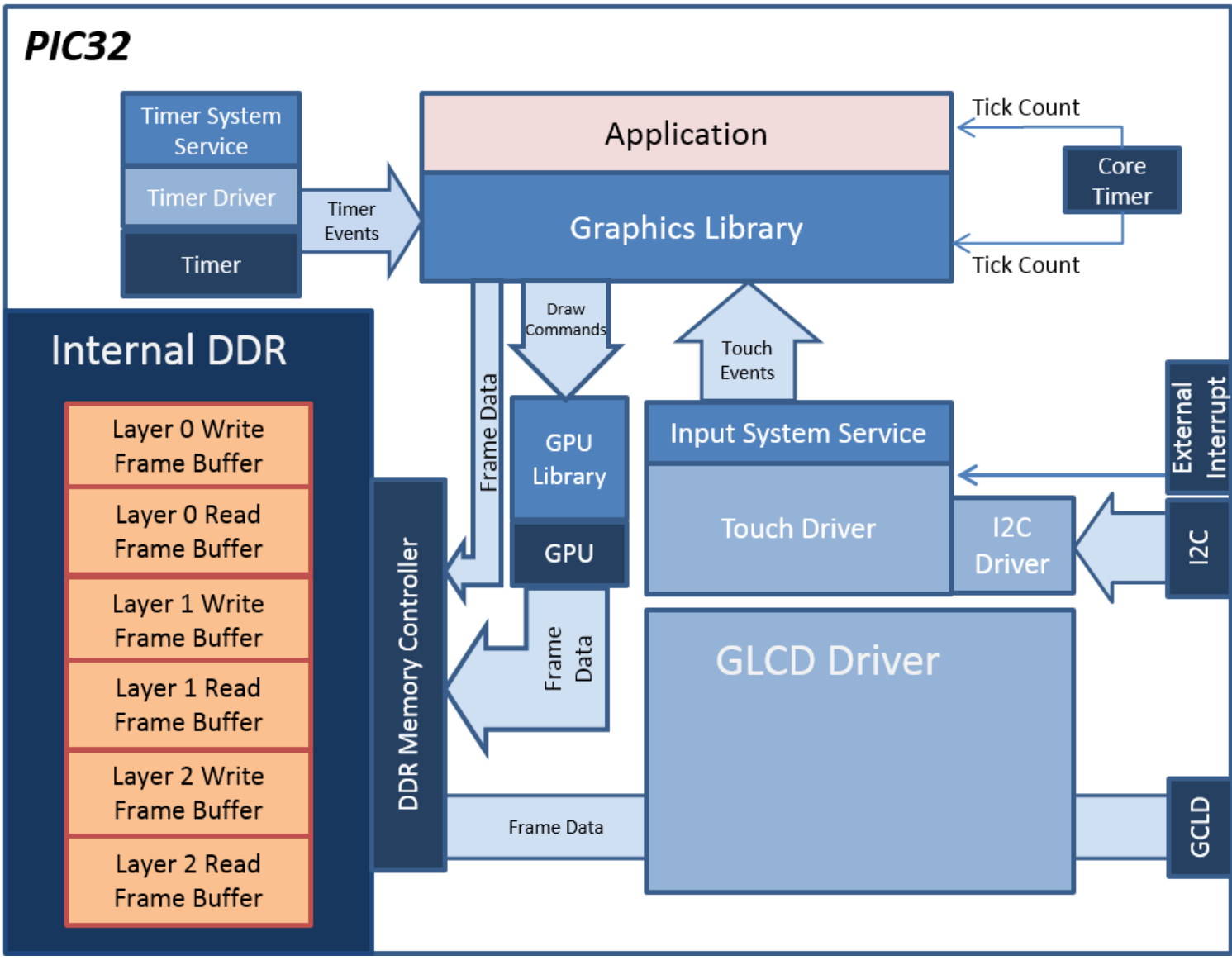
pic32mz_da_sk_intddr_meb2

pic32mz_da_sk_intddr_meb2_wvga

For these configurations, the application uses the Graphics Library to render graphics to the display. The Graphics library passes draw commands into the GPU Library, which in turn draws the widgets and images to the three individual write frame buffers (one for each layer) that are stored in an external DDR2. Via the DDR2 Memory Controller, the GLCD display controller peripheral continuously transfers frame data from all three read buffers onto the LCD display. The write and read frame buffer pairs are swapped independently as required when the Graphics Library is done rendering to the write frame buffer and GLCD driver signals the GLCD peripheral to change its read location.

The application also features user touch input through the integrated touch screen on the display panel. Touch input from the touch controller goes thru the I2C port, and the Touch System Service acquires the touch input information from the touch and I2C drivers. The Touch System Service sends touch events to the Graphics library which processes these events and updates the frame data accordingly.

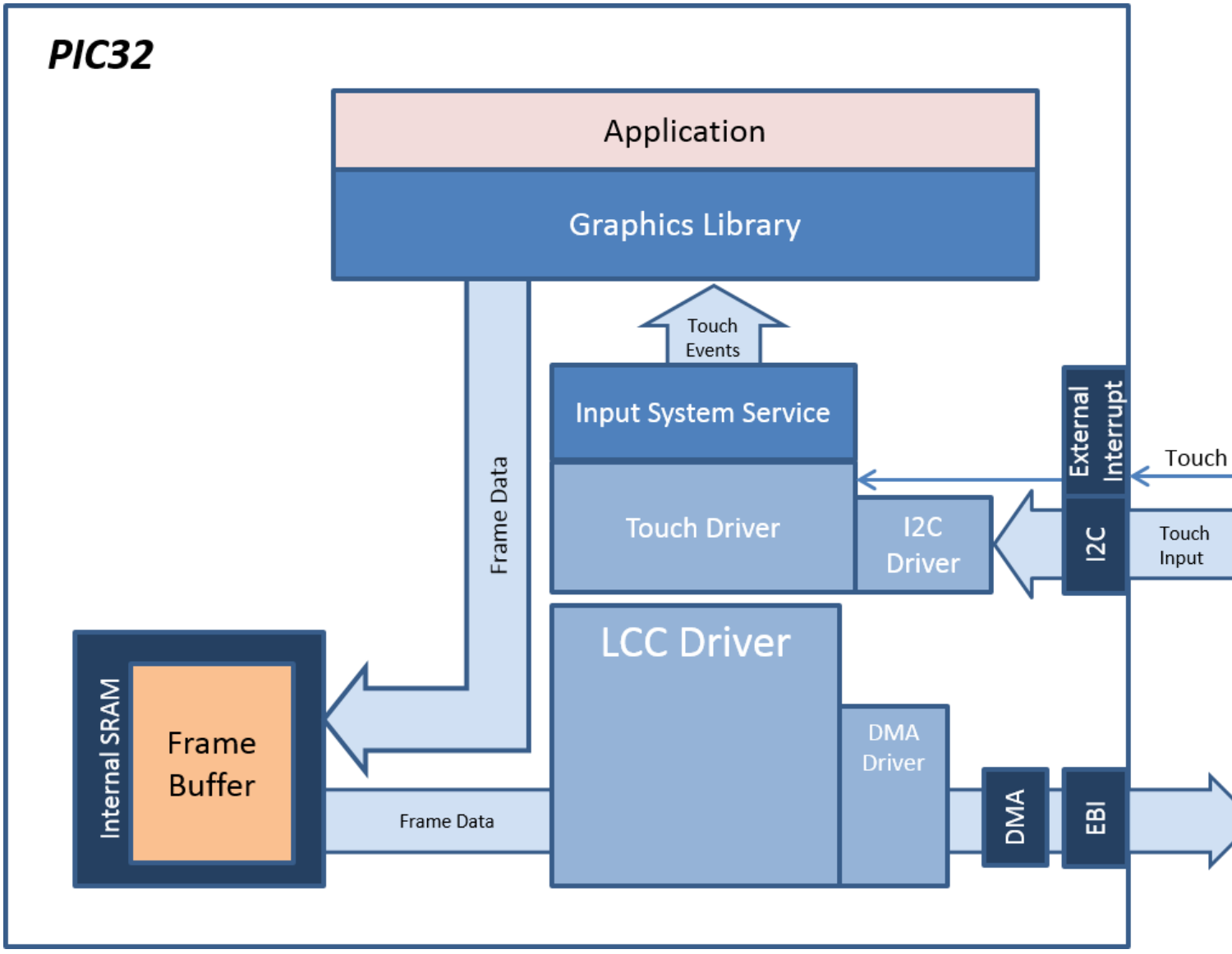
The core timer is used by the application and the Graphics library to manage the movement of the splash screen and the momentum decay of the list wheel tumbler.



pic32mz_ef_sk_meb2

For this configuration, the application uses the Graphics Library to render graphics to the display. The Graphics Library draws the widgets and images to the frame buffer that is stored on the internal SRAM. Using the DMA, the Low-Cost Controllerless (LCC) display driver continuously transfers frame data from the frame buffer out to the LCD display.

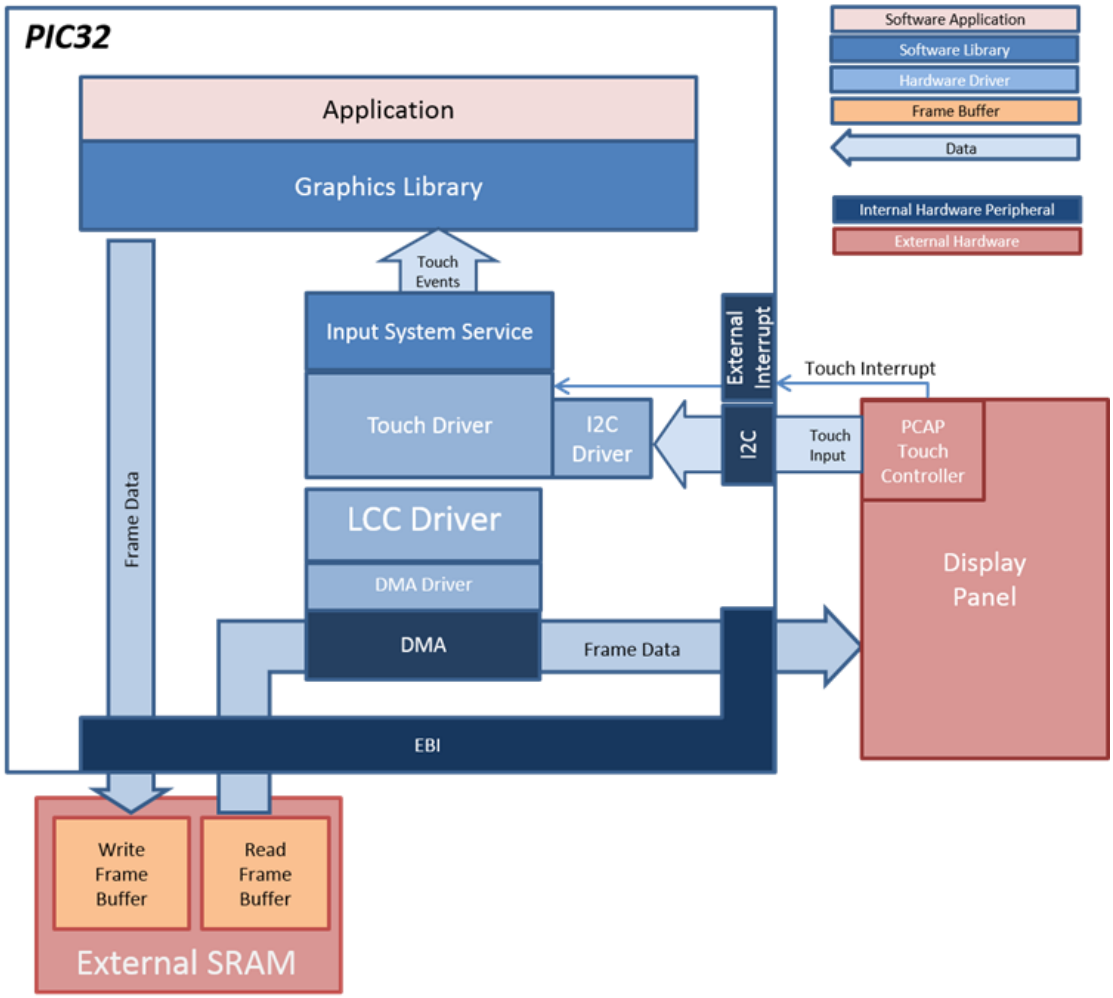
The application also features user touch input through the integrated touch screen on the display panel. Touch input from the touch controller goes thru the I2C port, and the Input System Service acquires the touch input information from the touch and I2C drivers. The Input System Service sends touch events to the Graphics Library which processes these events and updates the frame data accordingly.



pic32mz_ef_sk_meb2_ext

For this configuration, the aria_splash_screen application uses the Graphics Library to render graphics to the display. The Graphics Library draws the widgets and images to the write frame buffer that is stored in an external SRAM. Using the DMA, the Low-Cost Controllerless (LCC) display driver continuously transfers frame data from the read frame buffer out to the LCD display. The write and read frame buffers are swapped when the Graphics Library is done rendering to the write frame buffer and the LCC driver is done transferring the frame data from the read frame buffer.

The application also features user touch input through the integrated touch screen on the display panel. Touch input from the touch controller goes through the I2C port, and the Input System Service acquires the touch input information from the touch and I2C drivers. The Input System Service sends touch events to the Graphics Library, which processes these events and updates the frame data accordingly.



Important! There are custom code blocks identified by // CUSTOM CODE – DO NOT DELETE … // END OF CUSTOM CODE comments that represent modifications to the code outside of MHC. These code blocks are necessary for correct operation of the demonstration under these configurations. When regenerating the project from within MHC, use the "Prompt Merge For All Differences" merging strategy and do not change these custom code blocks.

Demonstration Features

- Integrated PCAP Touch Input (see mXT336T Touch Driver Library)
- Low-Cost Controllerless (LCC) Graphics driver on the PIC32MZ EF device
- 16-bit RGB565 color depth support (65536 unique colors) for PIC32MZ EF device (see Volume III: MPLAB Harmony Configurator (MHC)
- Three graphics layer supported via the GLCD peripheral on the PIC32MZ DA device
- GPU peripheral supported, can be enabled/disabled at run-time
- 32-bit RGBA8888 color depth support (16.7 million unique colors) (see *Volume III: MPLAB Harmony Graphics Configurator (MHC)* > *MPLAB Harmony Graphics Composer User's Guide* > *Graphics Composer Window User Interface* > Options
- Per-layer frame double-buffering
- Image compression techniques using Run-Length Encoding and JPEG (see Enabling Demo Mode in a MPLAB Harmony Graphics Application)

Tools Setup Differences

For all configurations:

"Use Graphics Stack" is selected in MHC. This enables the graphics and touch libraries and drivers for the BSP

pic32mz_da_sk_intddr_meb2, and pic32mz_da_sk_intddr_meb2_wvga configurations

Pin Settings: External Interrupt 4 mapped to pin A14 (RB1).

pic32mz_ef_sk_meb2 Configuration

MHC changes:

- *Driver > I2C > I2C Clock Frequency* is set to 10 kHz.
- *Graphics Stack > Graphics Options > Enable Draw Pipeline* > Enable Alpha Blending (Off)
- *Graphics Stack > Graphics Options > Enable Draw Pipeline* > Enable Color Conversion (Off)
- *Graphics Stack > Graphics Options > Enable Draw Pipeline* > Enable Image Clipping (Off)
- *Graphics Stack > Graphics Options > Enable Draw Pipeline* > Enable Orientation (Off)
- *Device & Project Configuration > Project Configuration > XC32 (Global Options) > xc32-ld > General > Heap Size (bytes)* > 102400

Pin Settings: External Interrupt 1 mapped to pin 23 (RE8).

pic32mz_ef_sk_meb2_ext Configuration

MHC changes:

- Driver > I2C > I2C Clock Frequency is set to 10 kHz.
- Graphics Stack > Graphics Options > Enable Draw Pipeline > Enable Alpha Blending (Off)
- Graphics Stack > Graphics Options > Enable Draw Pipeline > Enable Color Conversion (Off)
- Graphics Stack > Graphics Options > Enable Draw Pipeline > Enable Image Clipping (Off)
- Graphics Stack > Graphics Options > Enable Draw Pipeline > Enable Orientation (Off)
- Graphics Stack > Graphics Controller > Low Cost Controllerless > Frame Buffer Mode > Double Buffer
- Graphics Stack > Graphics Controller > Low Cost Controllerless > Memory Interface > External Memory
- Device & Project Configuration > Project Configuration > XC32 (Global Options) > xc32-ld > General > Heap Size (bytes) > 102400

Pin Settings: External Interrupt 1 mapped to pin 23 (RE8).

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Aria Splash Screen demonstration.

Description

To build this project, you must open the `aria_splash_screen.X` project in MPLAB X IDE, and then select the desired configuration. The following tables list and describe the project and supported configurations. The parent folder for these files is

`<install-dir>/apps/gfx/aria_splash_screen`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`aria_splash_screen.X`	`<install-dir>/apps/gfx/aria_splash_screen/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mz_da_sk_extddr_meb2	pic32mz_da_sk_extddr+meb2	Demonstration for the Multimedia Expansion Board II (MEB II) connected to the PIC32MZ Embedded Graphics with External DRAM (DA) Starter Kit.
pic32mz_da_sk_extddr_meb2_wvga	pic32mz_da_sk_extddr+meb2+wvga	Demonstration for the Multimedia Expansion Board II (MEB II) connected to the PIC32MZ Embedded Graphics with External DRAM (DA) Starter Kit and High-Performance 5" WVGA Display Module with maXTouch.
pic32mz_da_sk_intddr_meb2	pic32mz_da_sk_intddr+meb2	Demonstration for the Multimedia Expansion Board II (MEB II) connected to the PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit.
pic32mz_da_sk_intddr_meb2_wvga	pic32mz_da_sk_intddr+meb2_wvga	Demonstration for the Multimedia Expansion Board II (MEB II) connected to the PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit and High-Performance (5") WVGA Display Module with maXTouch.
pic32mz_ef_sk_meb2	pic32mz_ef_sk+meb2	Demonstration for the Multimedia Expansion Board II (MEB II) connected to the PIC32MZ EF Starter Kit.
pic32mz_ef_sk_meb2_ext	pic32mz_ef_sk+meb2	Demonstration for the Multimedia Expansion Board II (MEB II) connected to the PIC32MZ EF Starter Kit (utilizing External Memory).

Configuring the Hardware

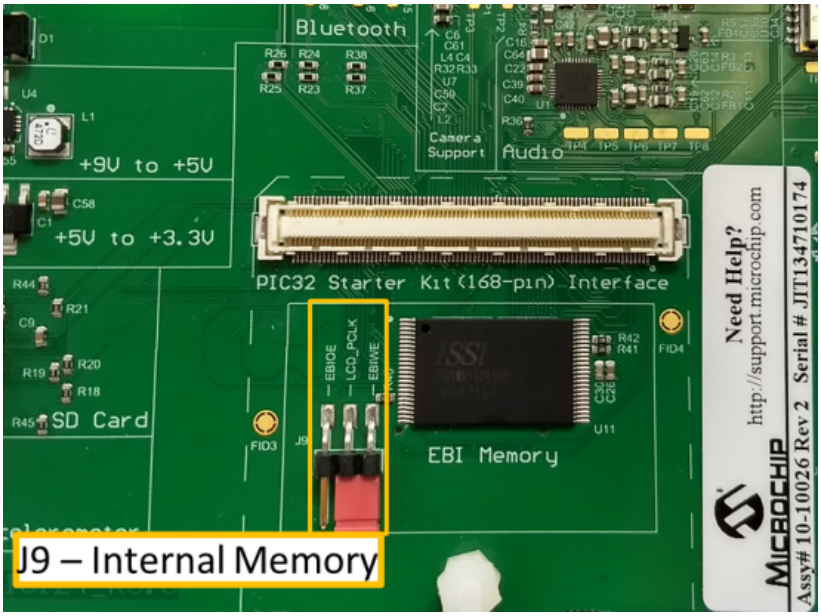
This section describes how to configure the supported hardware.

Description

PIC32MZ EF Starter Kit connected to the Multimedia Expansion Board II (MEB II) (Internal SRAM Frame Buffer)

Configuration: pic32mz_ef_sk_meb2

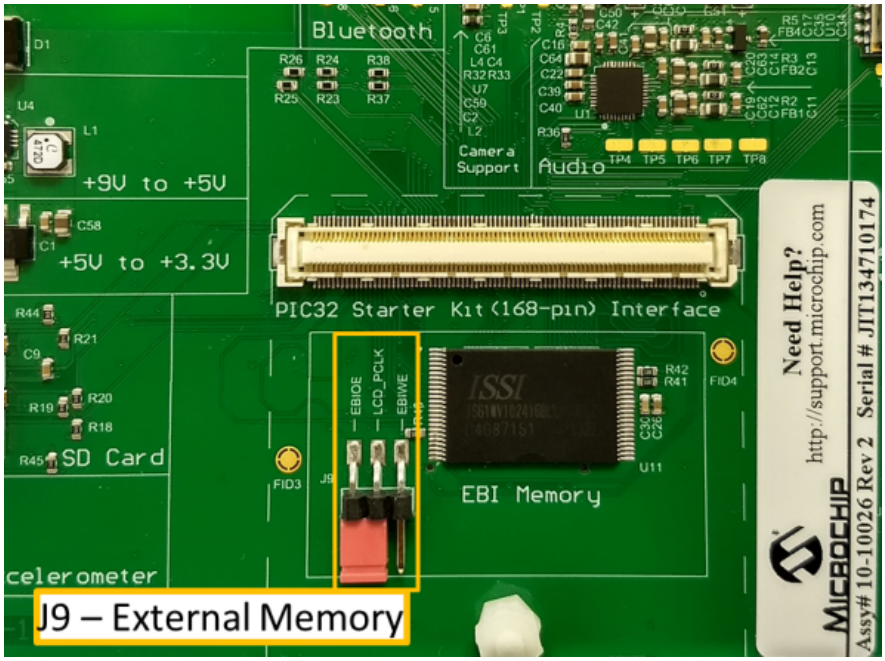
- The J9 Jumper on the MEB II board, located beneath the PIC32MZ EF Starter Kit, should be configured to support using internal SRAM for the graphics frame buffer, as shown in the following figure.
- Connect the PIC32MZ EF Starter Kit board to the MEB II board
- Power up the board by connecting the power adapter to J3 power connector on the MEB II board or a powered USB cable to the USB DEBUG port on the Starter Kit board



PIC32MZ EF Starter Kit connected to the Multimedia Expansion Board II (MEB II) (External SRAM Frame Buffer)

Configuration: pic32mz_ef_sk_meb2_ext

- The J9 Jumper on the MEB II board, located beneath the PIC32MZ EF Starter Kit, should be configured to support using external SRAM for the graphics frame buffer, as shown in the following figure.
- Connect the PIC32MZ EF Starter Kit board to the MEB II board
- Power up the board by connecting the power adapter to J3 power connector on the MEB II board or a powered USB cable to the USB DEBUG port on the Starter Kit board

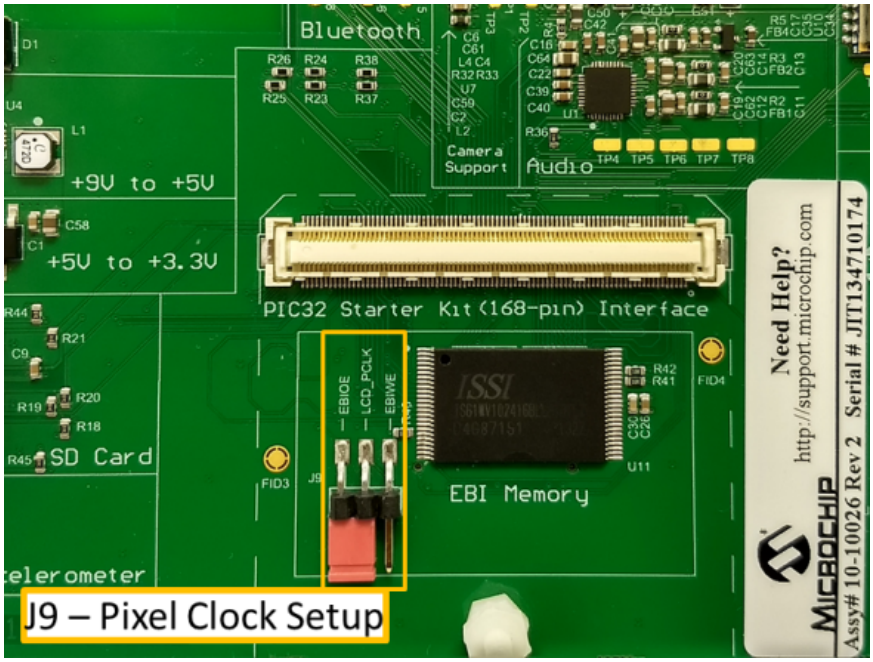


PIC32MZ Embedded Graphics with External DRAM (DA) Starter Kit and the Multimedia Expansion Board II (MEB II) with one of the following displays:

- Standard 4.3" WQVGA Display Module with maxTouch
- High-Performance (5") WVGA Display Module with maxTouch

Configurations: pic32mz_da_sk_extddr_meb2, and pic32mz_da_sk_extddr_meb2_wvga

- On the MEB II, the EBIOE and LCD_PCLK (J9) must be closed. A closed setting establishes the GLCD's pixel clock output timing. The jumper (J9) is available on the bottom side of the MEB II board. See the following figure below for the jumper location.
- Connect the PIC32MZ DA Starter Kit board to the MEB II board
- Power up the board by connecting the power adapter to J3 power connector on the MEB II board or a powered USB cable to the USB DEBUG port on the Starter Kit board



PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit and the Multimedia Expansion Board II (MEB II) with one of the following displays:

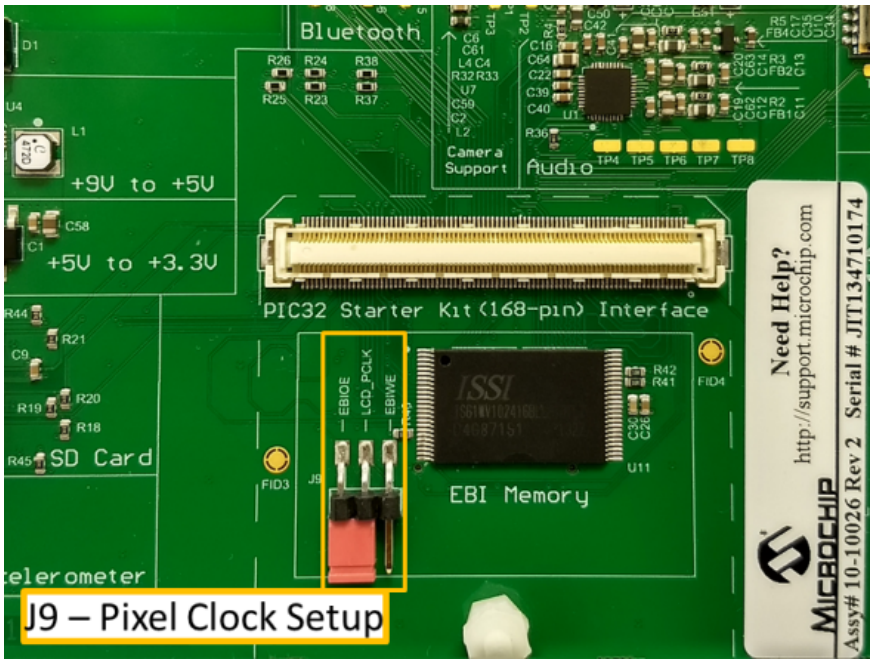
- Standard 4.3" WQVGA Display Module with maxTouch
- High-Performance (5") WVGA Display Module with maxTouch

Configurations: pic32mz_da_sk_intddr_meb2, and pic32mz_da_sk_intddr_meb2_wvga

- These configuration require that the J9 jumper be set to provide the GLCD's pixel clock. Set the J9 jumper to connect EPIOE to LCD_PCLK.

The J9 jumper is located on the bottom of the MEB II board, beneath where the starter kit is plugged into the board. Refer to the following image for the exact location.

- Connect the PIC32MZ DA Starter Kit board to the MEB II board
- Power up the board by connecting the power adapter to J3 power connector on the MEB II board or a powered USB cable to the USB DEBUG port on the Starter Kit board



Running the Demonstration

This section provides a brief description of what to expect once the demonstration is running.

Description

On power-up, the application will display an animated splash screen. There are two different versions of the splash screen.

With the multi-layer version, supported by the PIC32MZ DA configurations, the demonstration will fade to white.

Single-Layer Version

For the pic32mz_ef_sk_meb2 and pic32_ef_sk_meb2_ext configurations, the single-layer version is the displayed. The following images provide an approximation of what to expect on-screen.

First, the PIC32 logo appears.



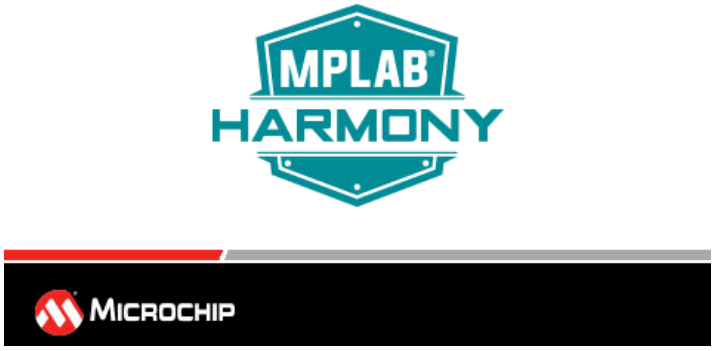
Then, at the bottom of the screen, a black bar will slide in from the right, stopping when it touches the left.



Next, the PIC32 logo fades out as the MPLAB Harmony logo fades in.



Finally, the MPLAB Harmony logo appears at full opacity and the Microchip logo appears at the same time.



With the single-layer version, the demonstration holds steady at this image.

Multi-Layer Version

For the pic32mz_da_sk_extddr_meb2, pic32mz_da_sk_extddr_meb2_wvga, pic32mz_da_sk_intddr_meb2, and pic32mz_da_sk_intddr_meb2_wvga configurations, the multi-layer version is the displayed.

The following image provide an approximation of what to expect.

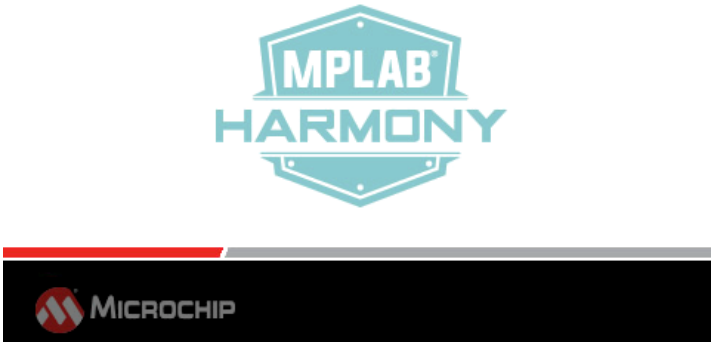
First, the PIC32 logo will appear.



At the bottom of the screen, a black bar will slide in from the right, stopping when it touches the left. At the same time, the PIC32 logo fades out.

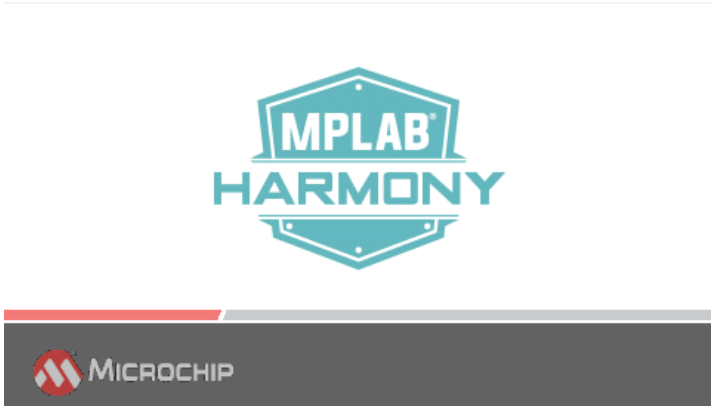


Next, the MPLAB Harmony logo and the Microchip logo fade in.



Once both the Harmony logo and the Microchip logo reaches full opacity, the entire screen will begin fade to white.





With the multi-layer version, the demonstration fades to white and remains so.

aria_touchadc_calibrate

This application demonstrates a 4-point resistive touch calibration using the Aria Graphics Library. It also provides an example and method for 4-point calibration with initialization and persistent storage.

Description

This application is an example on how to calibrate a resistive touch overlay using the Aria Graphics Library and the Harmony `drv_touchadc.c` driver on PIC32MZ EF and PIC32MX parts. Calibration flash storage is not available for pic32mx_usb_sk2_lcc_pictail_qvga, pic32mx_usb_sk2_lcc_pictail_wqvga, and pic32mz_ef_sk_s1d_pictail_wqvga configurations.

Architecture

In all configurations, the application consists of the Aria Graphics Library, Input System Services, and application calibration logic. The Aria Graphics Library is used to establish the UI presentation. Input System Services is used to interface with the resistive touch overlay and implement a 4-point calibration method.

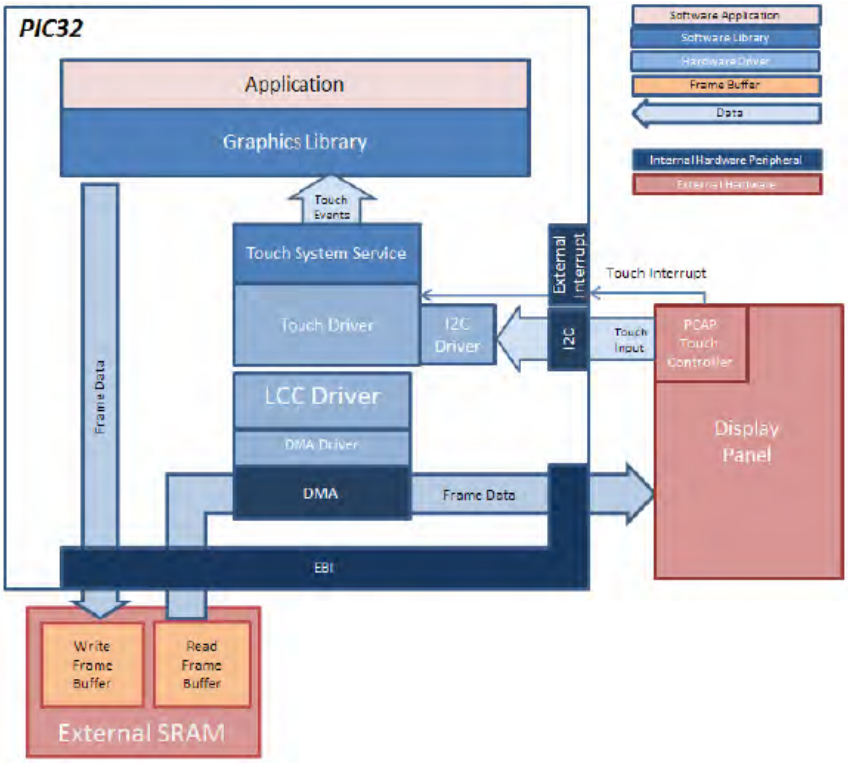
The graphic design layout consists of three screens. The TouchWidget screen has two buttons which provide a transition to the Calibration and KeyPad. The KeyPad screen as an array of buttons used to verify touch response, and a TouchWidget button which transitions to the TouchWidget screen. The Calibration screen renders four sequential touch points (crosshairs) to the display 10% from each corner of the display. The user is guided to press and release a stylus at the center of each cross hair. On release, the cross-hair turns green and the coordinate location is displayed for each image point touched. Once calibrated, the calibration data is used immediately by the application and saved to the external SRAM.

This Demonstration uses the Harmony `drv_touchadc.c` driver coupled with a method to obtain, use, and store calibration data.

These are the key points:

- Two-touch Interrupt versus polling driver for PIC32MZ EF and PIC32MX
- MHC calibration settings for program flash storage persistence
- External SRAM storage example
- Calibration verification
- Input System Services

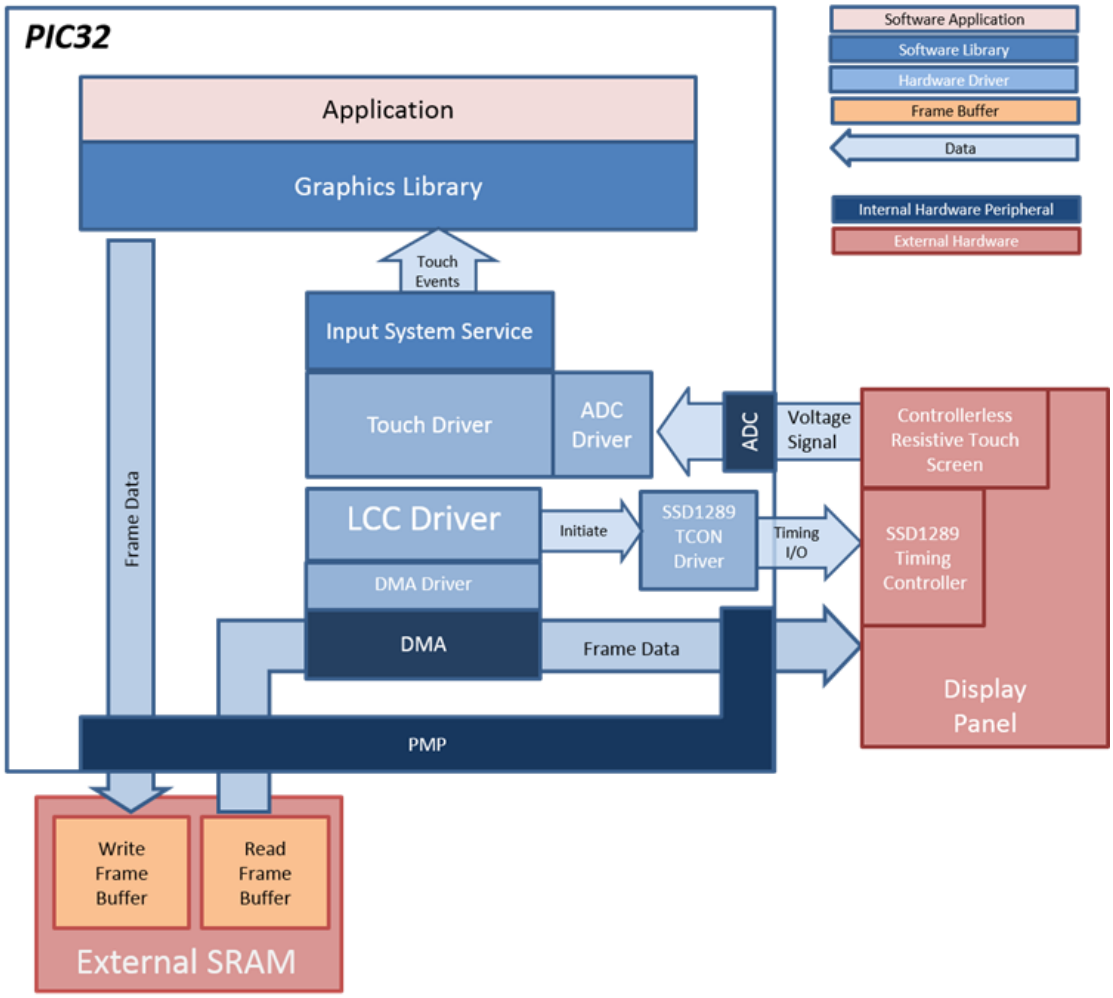
The following figure shows the various software and hardware components used by this application.



pic32mx_usb_sk2_lcc_pictail_qvga, pic32mx_usb_sk2_lcc_pictail_wqvga

For this configuration, the aria_touchadc_calibrate application uses the Graphics Library to render graphics to the display. The Graphics Library draws the widgets and images to the write frame buffer that is stored in the external SRAM. Using the DMA, the Low-Cost Controllerless (LCC)display driver continuously transfers frame data from the read frame buffer out to the LCD display. The write and read frame buffers are swapped when the Graphics Library is done rendering to the write frame buffer, and the LC driver is done transferring the frame data from the read frame buffer. The display has a built-in SSD1289 TCON Driver, which is initiated by the LCC Driver.

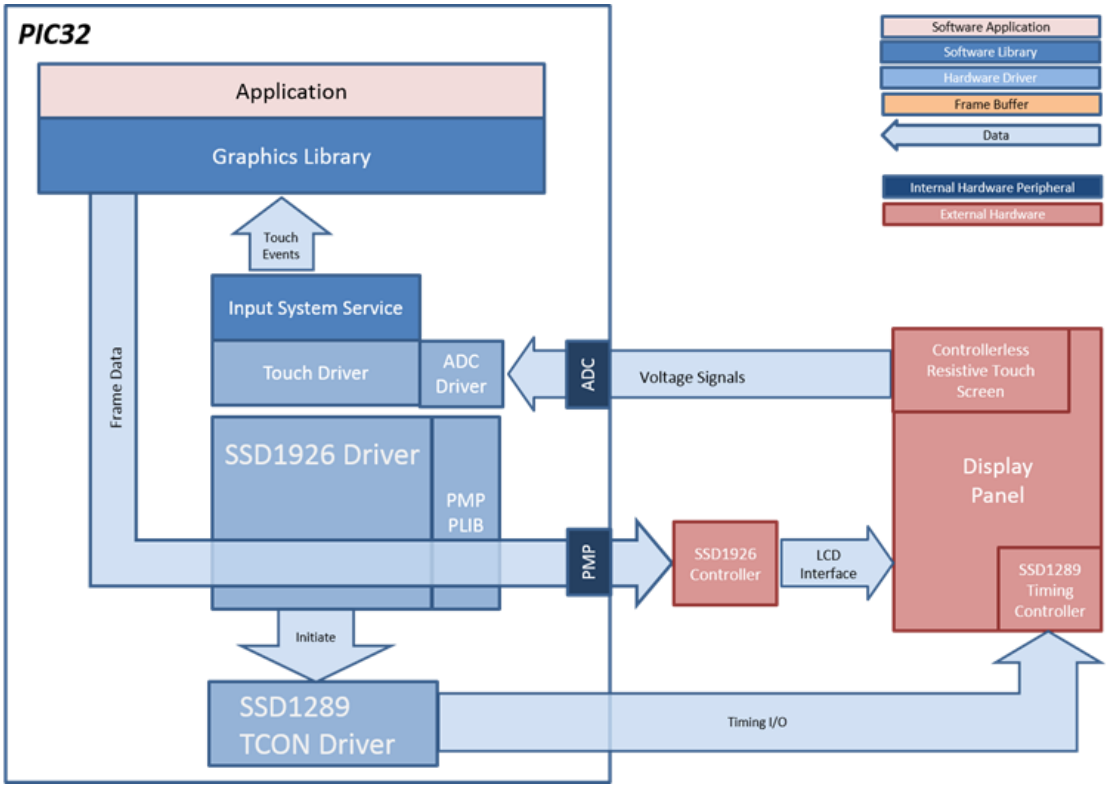
The application also features user touch input by measuring the voltages from the display panel. The voltage signal is measured by the ADC and interpreted by the touch driver, and the Input System Service acquires the touch input information from the touch driver. The Input System Service sends touch events to the Graphics Library, which processes these events and updates the frame data accordingly.



pic32mx_usb_sk2_meb

For this configuration, the display is driven by a SSD1926 display controller (located on the underside of the Multimedia Expansion Board II (MEB II)) that interfaces to the PIC32MX MCU through the PMP. The MCU-SSD1926 interface consists of 16 PMP data pins (PMP0-15), these are the Read Strobe pin, Write Strobe pin, Data/Command_Bar pin, and the Chip_Select_Bar pin. The PMP peripheral is driven by PMP PLIB calls. As the Graphics Library draws the widgets and images, the graphics library uses the SSD19126 display controller driver APIs to command the pixel and rectangle draws. The display has a built-in SSD1289 timing controller, which is managed by the SSD1289 Timing Controller (TCON) Driver through GPIO pins. The SSD1289 TCON Driver is initiated by the SSD1926 display graphics controller driver.

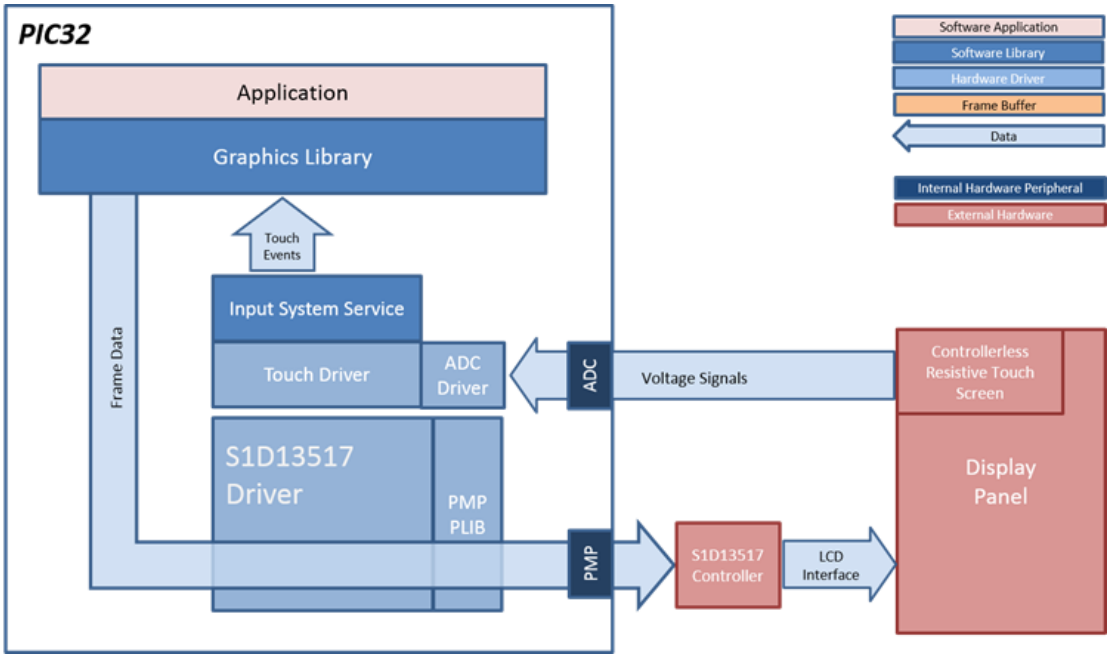
The application also features user touch input by measuring the voltages from the display panel. The voltage signal is measured by the ADC and interpreted by the touch driver, and the Input System Service acquires the touch input information from the touch driver. The Input System Service sends touch events to the Graphics Library, which processes these events and updates the frame data accordingly.



pic32mx_usb_sk2_s1d_pictail_wqvga, pic32mx_usbsk2_s1d_pictail_wvga

For this configuration, the display is driven by the EPSON SID13517 display controller (located on the Graphics Controller PICtail Plus Epson S1D13517 Daughter Board) that interfaces to the PIC32MX MCU through the PMP. The MCU-S1D13517 interface consists of 16t PMP data pins (PMP0-15), these are the Read Strobe pin, Write Strobe pin, Data/Command_Bar pin, and Chip_Select_Bar pin. The PMP peripheral is driven by PMP PLIB calls. As the Graphics Library draws the widgets and images, the graphics library uses the S1D13517display controller driver APIs to command the pixel and rectangle draws.

The application also features user touch input by measuring the voltages from the display panel. The voltage signal is measured by the ADC and interpreted by the touch driver, and the Input System Service acquires the touch input information from the touch driver. The Input System Service sends touch events to the Graphics Library, which processes these events and updates the frame data accordingly.

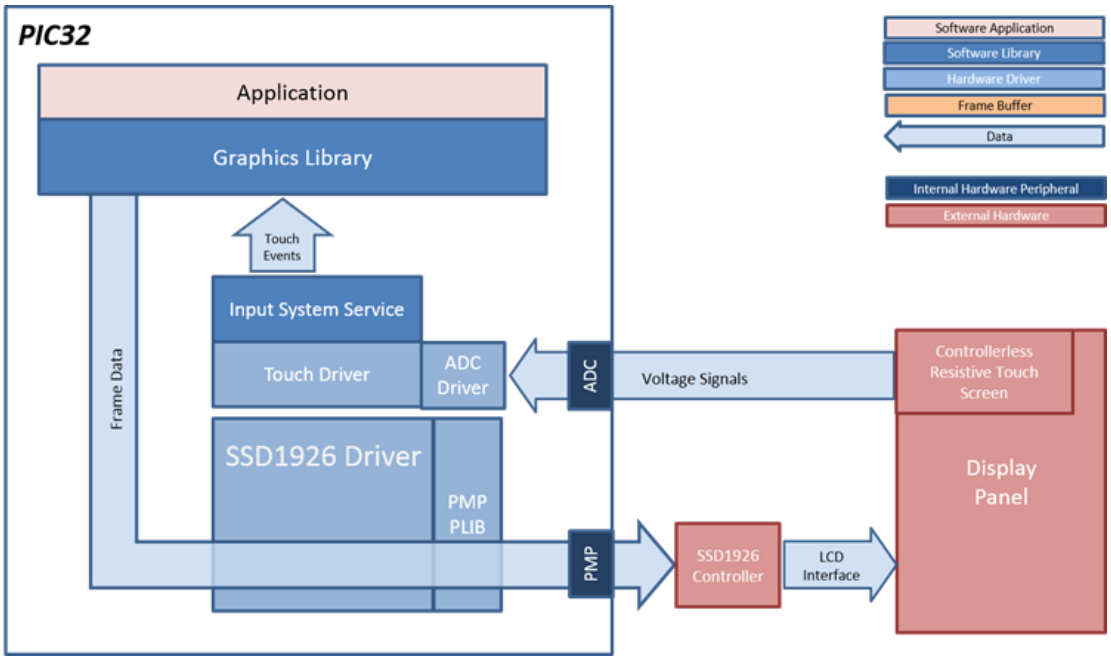


pic32mx_usb_sk2_ssd_pictail_qvga

For this configuration, the display is driven by a SSD1926 display controller (located on the underside of the Multimedia Expansion Board II (MEB II)) that interfaces to the PIC32MX MCU through the PMP. The MCU-SSD1926 interface consists of 16 PMP data pins (PMP0-15), these are the

Read Strobe pin, Write Strobe pin, Data/Command_Bar pin, and Chip_Select_Bar pin. The PMP peripheral is driven by PMP PLIB calls. As the Graphics Library draws the widgets and images, the graphics library uses the SSD19126 display controller driver APIs to command the pixel and rectangle draws. The display has a built-in SSD1289 timing controller, which is managed by the SSD1289 Timing Controller (TCON) Driver through the GPIO pins. The SSD1289 TCON Driver is initiated by the SSD1926 display graphics controller driver.

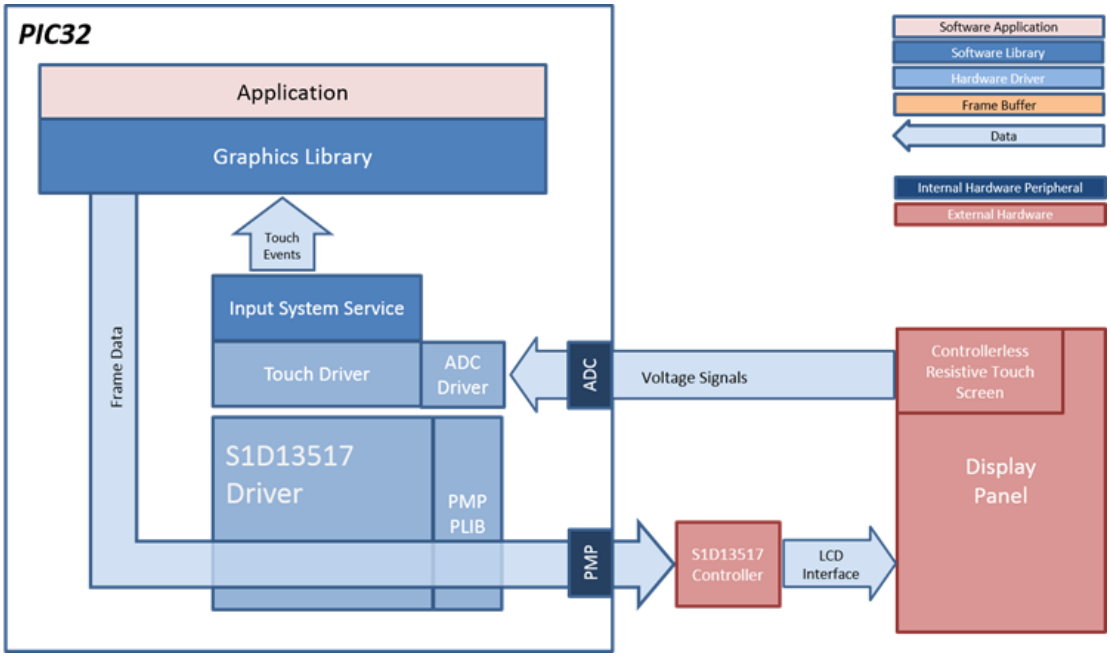
The application also features user touch input by measuring the voltages from the display panel. The voltage signal is measured by the ADC and interpreted by the touch driver, and the Input System Service acquires the touch input information from the touch driver. The Input System Service sends touch events to the Graphics Library, which processes these events and updates the frame data accordingly.



pic32mz_ef_sk_s1d_pictail_wqvga

For this configuration, the display is driven by the APSON SID13517 display controller (located on the Graphics Controller PICtail Plus Epson S1D13517 Daughter Board) that interfaces to the PIC32MX MCU through the PMP. The MCU-S1D13517 interface cosists of 16t PMP data pins (PMP0-15), these are the Read Strobe pin, Write Strobe pin, Data/Command_Bar pin, and Chip_Select_Bar pin. The PMP peripheral is driven by PMP PLIB calls. As the Graphics Library draws the widgets and images, the graphics library uses the S1D13517display controller driver APIs to command the pixel and rectangle draws.

The application also features user touch input by measuring the voltages from the display panel. The voltage signal is measured by the ADC and interpreted by the touch driver, and the Input System Service acquires the touch input information from the touch driver. The Input System Service sends touch events to the Graphics Library, which processes these events and updates the frame data accordingly.



Demonstration Features

- RGB565 Color Mode (PIC32MZ EF)
- Internal Flash Asset Storage
- Single Display Layer
- User Interface Tree Hierarchy
- TouchTest Widget
- Keypad Widget
- Resistive Touch

Tools Differences

All configurations

- *Use Graphics Stack* is selected in MHC. This enables the graphics and touch libraries and drivers for the BSP.
- The Input System Service and Input Driver is selected. The touch driver used is the Touch ADC Driver, please refer to the ADC Touch Driver Library and ADC Driver Library.
- The heap size is set to 100268 bytes to ensure enough memory for dynamic allocations. This is done in the MPLAB Harmony Configurator (MHC) by setting the size in Device & Project Configuration > Project Configuration>XC32 (Global Options)>xc32-id>General.

pic32mx _usb_sk2_lcc_pictail_qvga

pic32mx_usb_sk2_lcc_pictail_wqvga

pic32mx_usb_sk2_meb

pic32mx_usb_sk2_s1d_pictal_wqvga

pic32mx_usb_sk2_s1d_pictal_wvga

pic32mx_usb_sk2_ssd_pictail_qvga

pic32mz_ef_sk_s1d_pictail_wqvga

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Aria TouchADC Calibrate demonstration.

Description

To build this project, you must open the `aria_touchadc_calibrate.X` project in MPLAB X IDE, and then select the desired configuration. The following tables list and describe the project and supported configurations. The parent folder for these files is <install-dir>/apps/gfx/aria_touchadc_calibrate.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
aria_touchadc_calibrate	*<install-dir>/apps/gfx/ aria_touchadc_calibrate /firmware*

MPLAB X IDE Project Configuration

This table lists and describes the supported configurations of the demonstration, which are located within *./firmware/src/system_config.*

Project Configuration Name	BSP Used	Description
pic32mx_usb_sk2_lcc_pictal_qvga	pic32mx_usb_sk2+lcc_pictail+qvga	Demonstration for the PIC32MX USB Starter Kit II plus the LCC Graphics PICtail Plus Daughter Board with the Graphics Display Truly 3.2” 320x240 Board.
pic32mx_usb_sk2_lcc_pictal_wqvga	pic32mx_usb_sk2+lcc_pictail+wqvga	Demonstration for the PIC32MX USB Starter Kit II plus the LCC Graphics PICtail Plus Daughter Board with the Graphics Display Truly 4.3” 480x272 Board.
pic32mx_usb_sk2_meb	pic32mx_usb_sk2+meb	Demonstration for the PIC32 MX USB Starter Kit II plus Multimedia Expansion Board (MEB).

pic32mx_usb_sk2_s1d_pictail_wqvga	pic32mx_usb_sk2+s1d_pictail+wqvga	Demonstration for the PIC32MX USB Starter Kit II plus the Graphics PICtail Plus Epson S1D13517 Daughter board with a Graphics Display Powertip 4.3” 480x272 Board.
pic32mx_usb_sk2_s1d_pictail_wvga	pic32mx_usb_sk2+s1d_pictail+wvga	Demonstration for the PIC32MX USB Starter Kit II plus the Graphics PICtail Plus Epson S1D13517 Daughter board with a Graphics Display Truly 7” 800x480 Board.
pic32mx_usb_sk2_ssd_pictail_qvga	pic32mx_usb_sk2+ssd_pictail+qvga	Demonstration for the PIC32MX USB Starter Kit II plus the Graphics PICtail Plus SSD1926 board with a Graphics Display Truly 3.2” 320x240 Board.
pic32mz_ef_sk_s1d_pictail_wqvga	pic32mz_ef_sk+s1d_pictail+wqvga	Demonstration for the PIC32MX USB Starter Kit II plus the Graphics PICtail Plus Epson S1D13517 Daughter board with a Graphics Display Powertip 4.3” 480x272 Board.

Note: This application contains custom code that is marked by the comments // START OF CUSTOM CODE … and // END OF CUSTOM CODE. If you use the MPLAB Harmony Configurator (MHC) to regenerate the application code, do not remove or replace the custom code.

Configuring the Hardware

This section describes how to configure the supported hardware.

Description

PIC32 USB Starter Kit II with the Low-Cost Controllerless (LCC) Graphics PICtail Plus Daughter Board and Graphics Display Truly 3.2” 320x240 Board

Configurations: pic32mx_usb_sk2_lcc_pictail_qvga

No hardware related configuration or jumper setting changes are necessary.

PIC32 USB Starter Kit II with the Low-Cost Controllerless (LCC) Graphics PICtail Plus Daughter Board and Graphics Display Powertip 4.3” 480x272 Board

Configurations: pic32mx_usb_sk2_lcc_pictail_wqvga

No hardware related configuration or jumper setting changes are necessary.

PIC32 USB Starter Kit II with the Multimedia Expansion Board (MEB)

Configurations: pic32mx_usb_sk2_meb

No hardware related configuration or jumper setting changes are necessary.

PIC32 USB Starter Kit II with the Graphics Controller PICtail Plus Epson S1D13517 Daughter board

Configurations: pic32mx_usb_sk2_s1d_pictail_wqvga, pix32mx_usb_sk2_s1d_pictail_wvga

P2 on the S1D13517 Daughter board should be closed (16-bit PMP) for the WQVGA display, and open (8-bit PMP) for the WVGA display.

PIC32 USB Starter Kit II with the Graphics LCD Controller PICtail Plus SSD1926 Daughter Board

Configurations: pic32mx_usb_sk2_ssd_pictail_qvga

JP2 on the SSD1926 Daughter board should be set to select the 8-bit PMP interface.

PIC32MZ EF Starter Kit connected to the Graphics Controller PICtail Plus Epson S1D13517 Daughter Board

Configurations: pic32mz_ef_sk_s1d_pictail_wqvga

JP2 on the SSD1926 Daughter board should be set to select the 8-bit PMP interface.

Running the Demonstration

This section provides instructions on how to use the Aria TouchADC Calibrate demonstration.

Description

For this demonstration, the following configurations do not support calibration SRAM persistence:

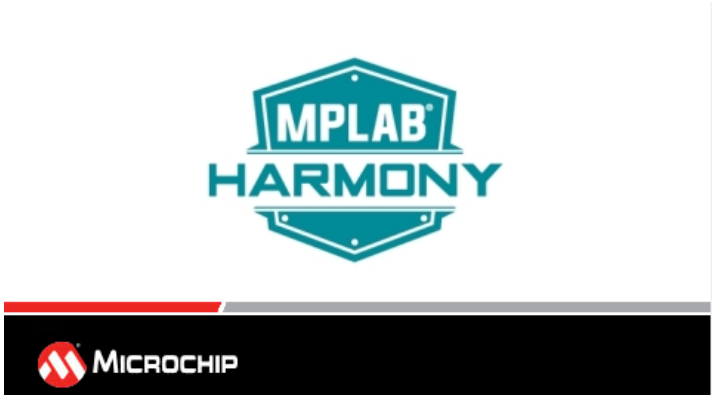
- pic32mx_usb_sk2_lcc_pictal_qvga
- pic32mx_usb_sk2_lcc_pictal_wqvga

- pic32mz_ef_sk_s1d_pictail_wqvga

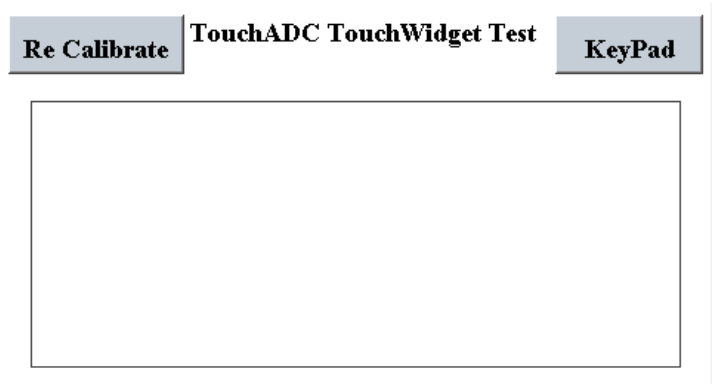
When power-on is successful, the demonstration will display a splash screen and then transition to the TouchADC Touch Widget Test screen if the user has not held a touch point. If the user has held down a touch point, then the Calibration screen is shown.

Different configurations may have slight variation in the screen aspect ratio.

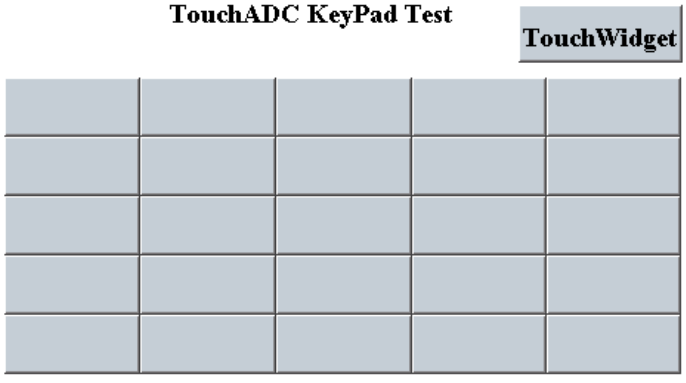
Splash Screen



Touch Widget Test



Keypad Test



Calibration Screen



Name	Description
TouchWidget Test	Renders the TouchWidget to allow the user to test the x and y crossing of touch coordination as a point is drag within the widget.
KeyPad Test	Renders the KeyPad Widget to allow the user to test button press at specific locations.
Calibration Screen	Renders touch points at four cross-hair locations. Each location sequentially turns green when user releases the red cross-hair.

aria_video_player

This application demonstrates video playback with full UI controls using the Aria Graphics Library. It also showcases the capabilities of MPLAB Harmony for reading data from external media like a USB drive or SD card.

Description

This application is an example on how to play video with user-interface controls using the Aria User Interface Library. The video is in RAW format that is pre-generated using publicly available software like FFMPEG. The video file is then read by the application from external media like a USB drive or SD card.

User interface controls are also shown simultaneously with the video playback. This allows the user to control the video playback properties, and also shows useful information such as frame rates, bandwidth, etc.

Architecture

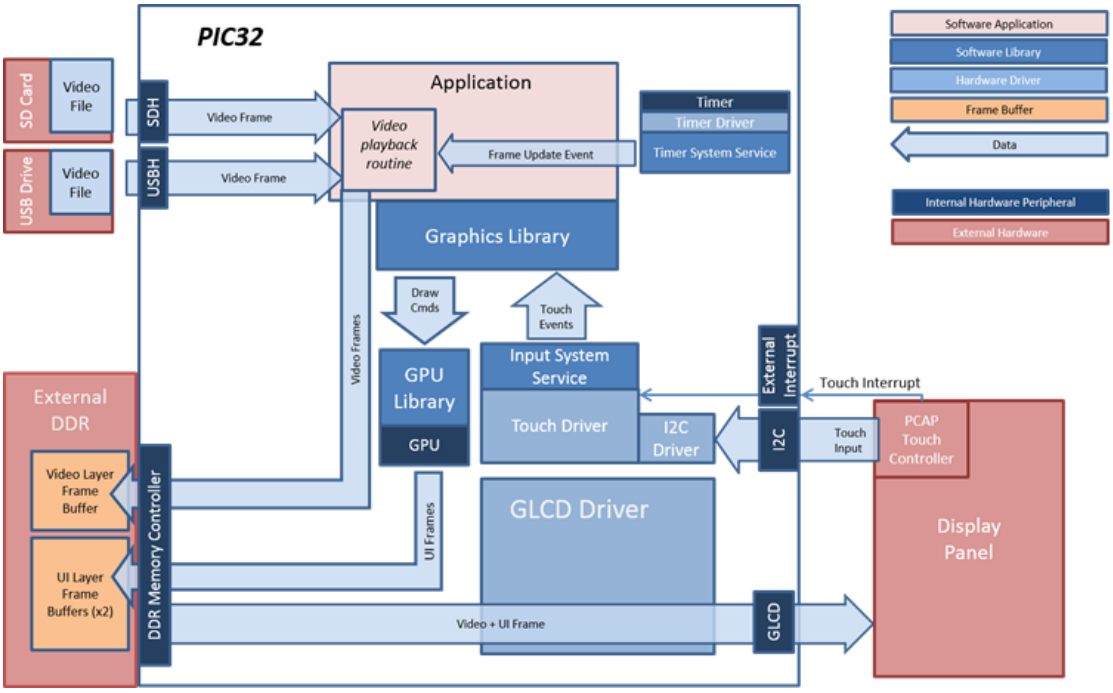
The following block diagrams show the various software and hardware blocks used in this application.

In all configurations, the application reads the video frame data from raw video files in a USB thumb drive or a microSD card. To optimize the video frame rendering process, the application writes the frame data directly to the frame buffer. Thus the video frame format should match the format of the frame buffer; in this case it is RGB565. A hardware timer and the Timer System Service send events to the application to update the frames based on the user-selected frame update rate.

The user interface controls and widgets are rendered to the frame buffer by the Graphics Library. The Graphics library also processes the input touch events and directs them to the application as needed.

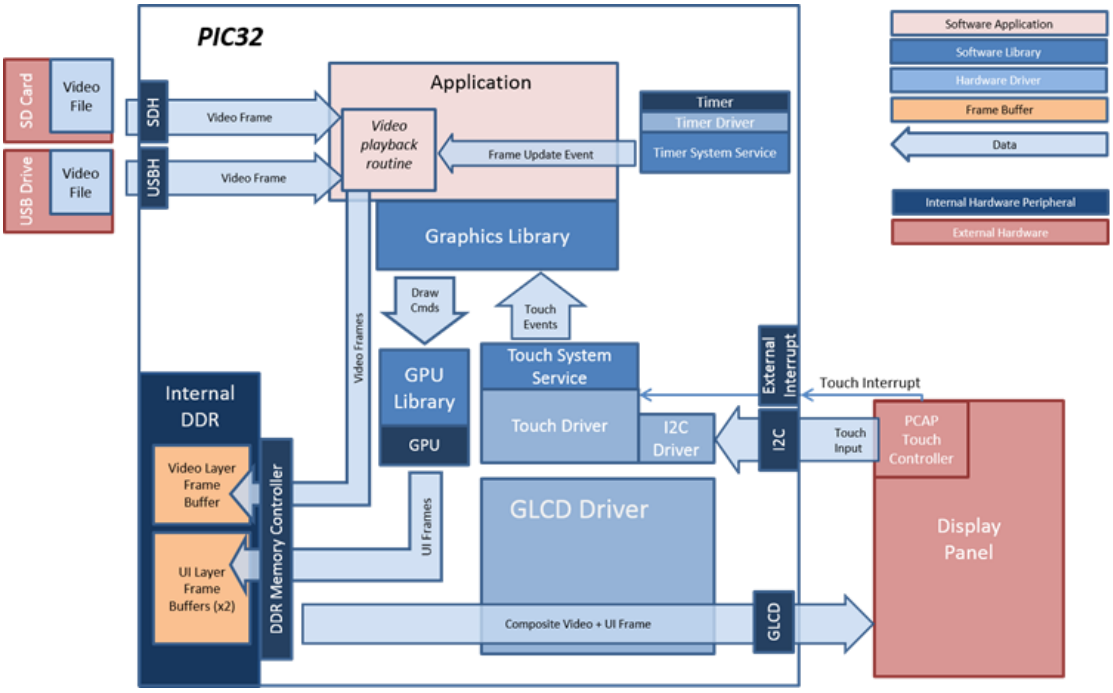
pic32mz_da_sk_extddr_meb2, pic32mz_da_sk_extddr_meb2_wvga

In these configurations, the frame buffers used for the video frames and the UI controls are located in the external DDR. The application takes advantage of the multiple (3) layers supported by the Graphics LCD (GLCD) Controller. The application writes out the video frames to a dedicated video layer and the Graphics Library renders the widgets and controls the other two layers. The GLCD continuously composites the video and UI layers into a single frame and sends the final frame to the display.



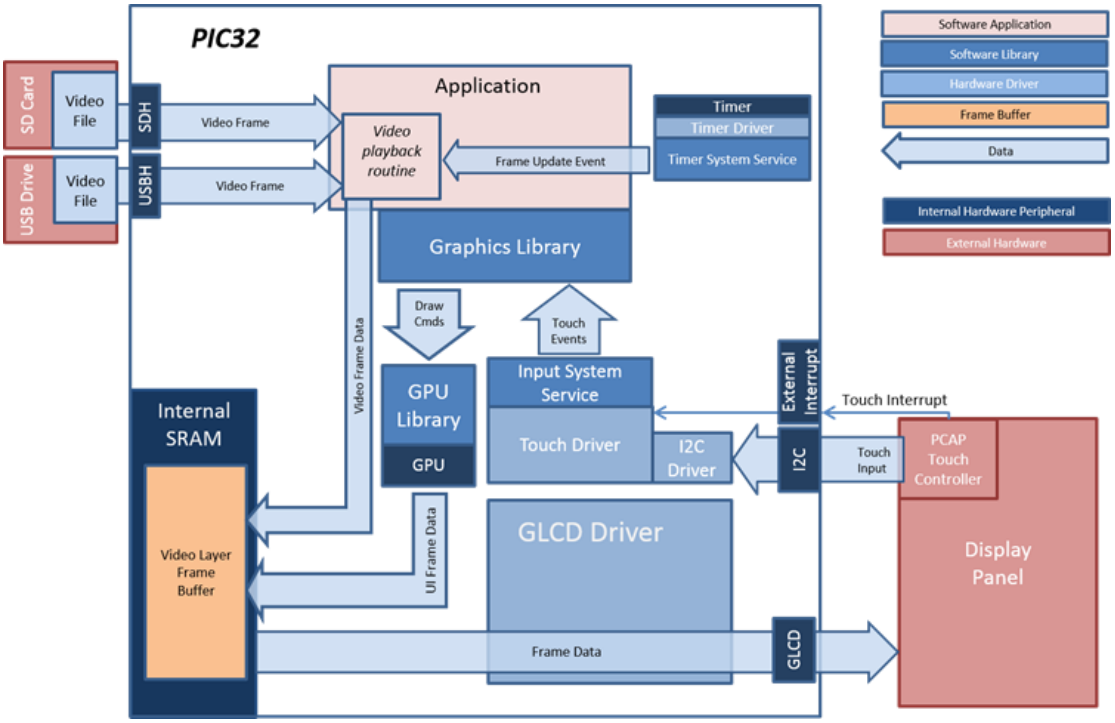
pic32mz_da_sk_intddr_meb2, pic32mz_da_sk_intddr_meb2_wvga

In these configurations, the frame buffers used for the video frames and the UI controls are located in the internal DDR. The application takes advantage of the multiple (3) layers supported by the GLCD controller. The application writes out the video frames to a dedicated video layer and the Graphics Library renders the widgets and controls the other two layers. The GLCD continuously composites the video and UI layers into a single frame and sends the final frame to the display.



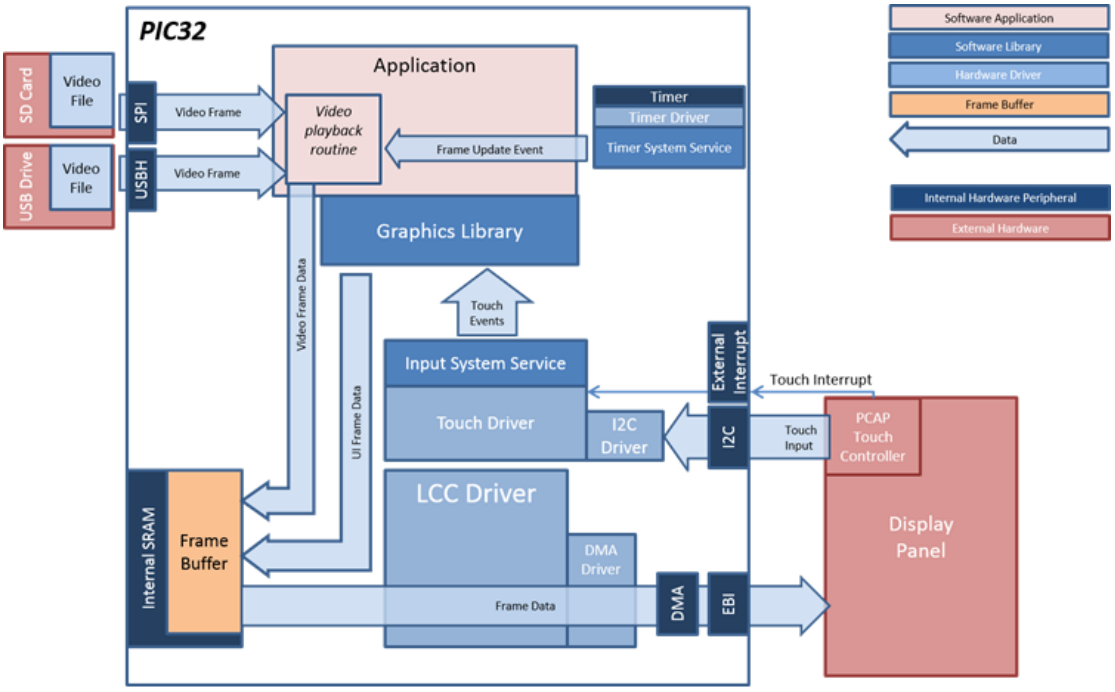
pic32mz_da_sk_noddr_meb2

In this configuration, the video frame and UI widgets are drawn in a single frame buffer in the internal SRAM. Since only one layer is used, the application must ensure that the video frames do not overwrite the controls on the screen. When showing the playback controls in full screen playback mode, the controls are in the bottom area of the screen. Therefore, the bottom part of the video frame that would have overlapped with the controls is not rendered. In non-full screen mode, the controls and video frames are positioned so that they do not overlap with each other.



pic32mz_ef_sk_meb2

In this configuration, the video frame and UI widgets are drawn on a single frame buffer in the internal SRAM. Since only one layer is used, the application must ensure that the video frames do not overwrite the controls on the screen. When showing the playback controls in full screen playback mode, the controls are located at the bottom area of the screen, thus the bottom part of the video frame that would have overlapped with the controls are not rendered. In non-full screen mode, the controls and video frames are positioned so that they do not overlap with each other.



Demonstration Features

- Video Playback (RAW video format)
- External Media Support (USB Drive, microSD Card)
- Three graphics layer supported via GLCD (PIC32MZ DA)
- Per layer run-time adjustable alpha blending (PIC32MZ DA)
- Hardware timer and Timer System Service
- Image compression techniques using Run-Length Encoding

Tools Setup Differences

- For the PIC32MZ EF configuration, *SD Card Driver* is enabled in *Drivers* in MHC
- For the PIC32MZ DA configurations, *SD Host Controller Driver* is enabled in *Drivers* in MHC
- *File System Service* is enabled in *System Services* in MHC
  - *Use File System Auto Mount Feature* is enabled
  - The *Total Number Of Media* is set to 2. For *Media 0*, set *Media Type* to `SYS_FS_MEDIA_TYPE_MSD`, *Media Mount Name* to `/mnt/usb`. For *Media 1*, set *Media Type* to `SYS_FS_MEDIA_TYPE_SD_CARD`, *Media Mount Name* to `/mnt/sdcard`
- *Use USB Stack* is enabled in *USB Library* in MHC. *USB Host* and *Use MSD Host Client Driver* are enabled
- Select the ON and OE for REFCLK4 in the Clock Diagram tab in MHC
- The heap size is set to 102400 bytes to ensure enough memory for dynamic allocations. This is done by setting the size in MHC through the following menu selection: *Device & Project Configuration >Project Configuration > XC32 (Global Options)>xc32-ld >General*

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Aria Video Player demonstration.

Description

To build this project, you must open the `aria_video_player.X` project in MPLAB X IDE, and then select the desired configuration. The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/gfx/aria_video_player`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`aria_video_player.X`	`<install-dir>/apps/gfx/aria_video_player/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32_mz_da_sk_extddr_meb2	pic32mz_da_sk_extddr+meb2	Demonstration for the PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit, Multimedia Expansion Board II (MEB II) and 4.3” WQVGA (480x272) Display.
pic32_mz_da_sk_extddr_meb2_wvga	pic32mz_da_sk_extddr+meb2+wvga	Demonstration for the PIC32MZ DA Starter Kit with External DDR2 Memory plus the Multimedia Expansion Board II (MEB II) with a 5” 800x480 WVGA PCAP TFT display.
pic32_mz_da_sk_intddr_meb2	pic32mz_da_sk_intddr+meb2	Demonstration for the PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit, Multimedia Expansion Board II (MEB II) and 4.3” WQVGA (480x272) Display.
pic32_mz_da_sk_intddr_meb2_wvga	pic32mz_da_sk_intddr+meb2+wvga	Demonstration for the PIC32MZ Embedded Graphics with External DRAM (DA) Starter Kit plus the Multimedia Expansion Board II (MEB II) with a 5” 800x480 WVGA PCAP TFT display.
pic32mz_ef_sk_meb2	pic32mz_ef_sk+meb2	Demonstration for the PIC32MZ EF Starter Kit plus the Multimedia Expansion Board II (MEB II) (Internal SRAM Frame Buffer).
pic32mz_da_sk_noddr_meb2	pic32mz_da_sk_noddr+meb2	Demonstration for PIC32MZ Embedded Graphics with Disabled DRAM (DA) Starter Kit, Multimedia Expansion Board II (MEB II) and 4.3” WQVGA (480x272) Display.

Configuring the Hardware

This section describes how to configure the supported hardware.

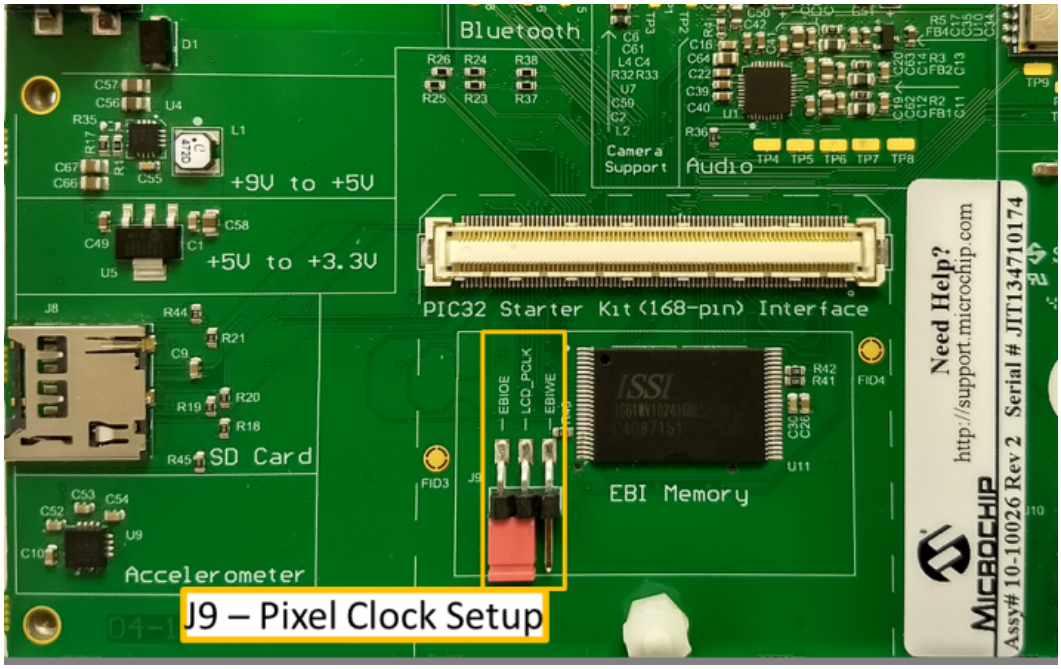
Description

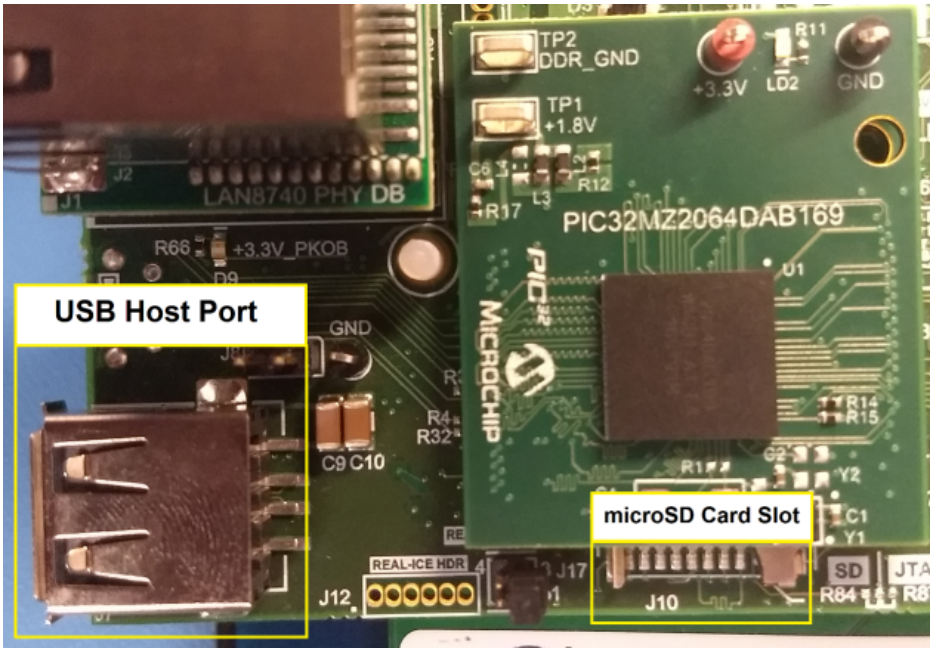
PIC32MZ Embedded Graphics with External DRAM (DA) Starter Kit or PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit, or PIC32MZ Embedded Graphics with Disabled DRAM (DA) Starter Kit, and MEB II

Configurations: pic32_mz_da_sk_intddr_meb2, pic32_mz_da_sk_intddr_meb2_wvga, pic32_mz_da_sk_extddr_meb2,

pic32_mz_da_sk_extddr_meb2_wvga, pic32mz_da_sk_noddr_meb2

- On the MEB II, the EBIOE and LCD_PCLK (J9) must be jumpered. A connection establishes the GLCD's pixel clock output timing. The external SRAM memory on the board is disabled. The jumper (J9) is available on the bottom side of the MEB II board under the starter kit. The J9 jumper is located on the bottom of the MEB II board, beneath where the starter kit is plugged into the board. Refer to the following image for the exact location.
- Connect the PIC32MZ DA Starter Kit board to the MEB II board.
- Power up the board by connecting the power adapter to J3 power connector on the MEB II board or a powered USB cable to the USB DEBUG port on the Starter Kit board
- Use the microSD slot and USB host port on Starter Kit for the media. Refer to the image below for the location of the ports.

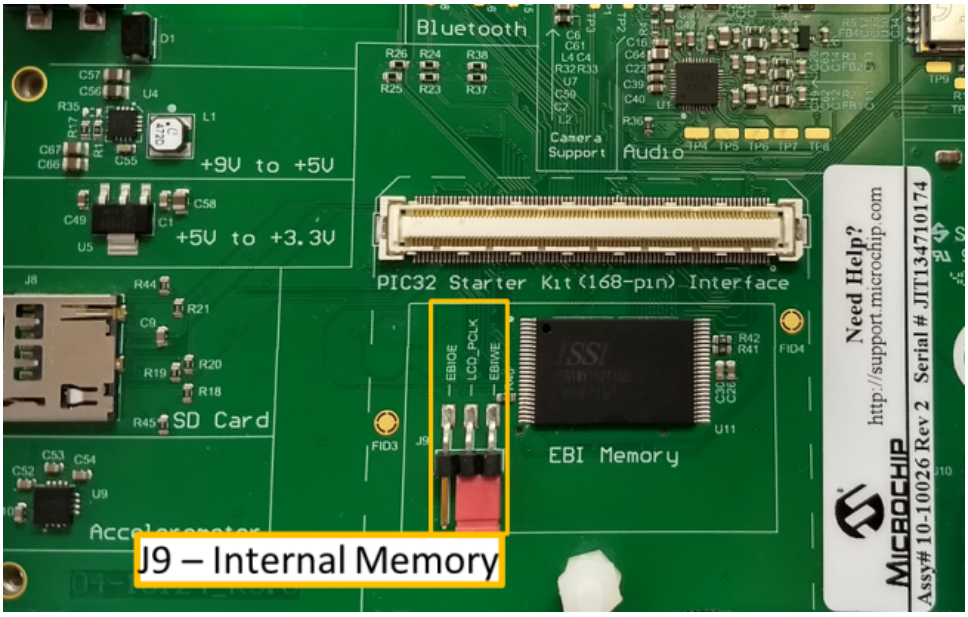




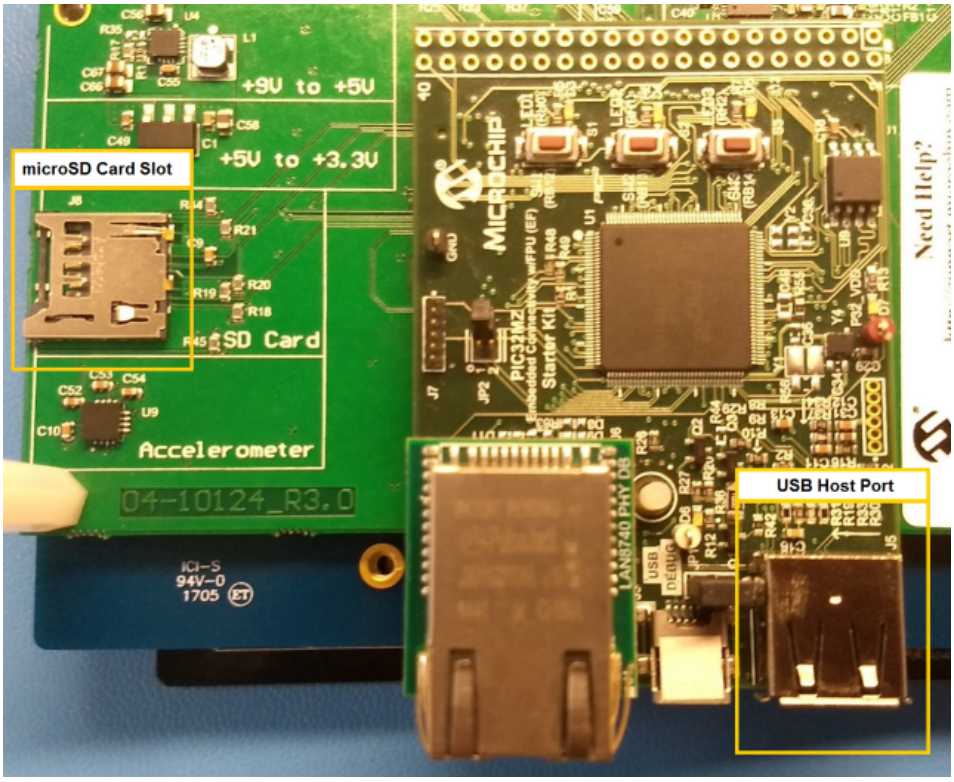
PIC32MZ EF Starter Kit and MEB II

Configuration: pic32mz_ef_sk_meb2

- The J9 Jumper on the MEB II board, located beneath the PIC32MZ EF Starter Kit, should be configured to support using internal SRAM for the graphics frame buffer, as shown in the following figure.



- Connect the PIC32MZ EF Starter Kit board to the MEB II board
- Use the microSD slot on the back of the MEB II and USB host port on the Starter Kit for the media. Refer to the image below for the location of the ports.



- Power up the board by connecting the power adapter to J3 power connector on the MEB II board or a powered USB cable to the USB DEBUG port on the Starter Kit board

Generating Video Files

The application plays RAW video files only. To generate RAW video files from other video formats, use FFMPEG. FFMPEG can be downloaded from https://ffmpeg.org.

To generate the video files, use the following command:

```
ffmpeg.exe –i <source video> -s <resolution> -r 15 –c:v rawvideo –vf “format=rgb565le” <video filename>
```

The application supports specific video resolutions. Use one of the following resolutions with the corresponding output video filename in FFMPEG.

Resolution	Output Video Filename
320x180	`video0.rgb`
320x240	`video1.rgb`
480x272	`video2.rgb`
800x480 (only in WVGA configurations)	`video3.rgb`

Copy the video file to the parent root directory of a USB flash drive or microSD card.

For reference, a sample RAW video clip (`video2.rgb.zip`) is included in the application resources directory in the MPLAB Harmony library. The clip is converted from the “Big Buck Bunny” video published by Blender Foundation, (c) copyright 2008, Blender Foundation, www.bigbuckbunny.org. To play the video, unzip the file and copy `video2.rgb` to a supported media. Make sure to select 480x272 as the video resolution on the Settings screen.

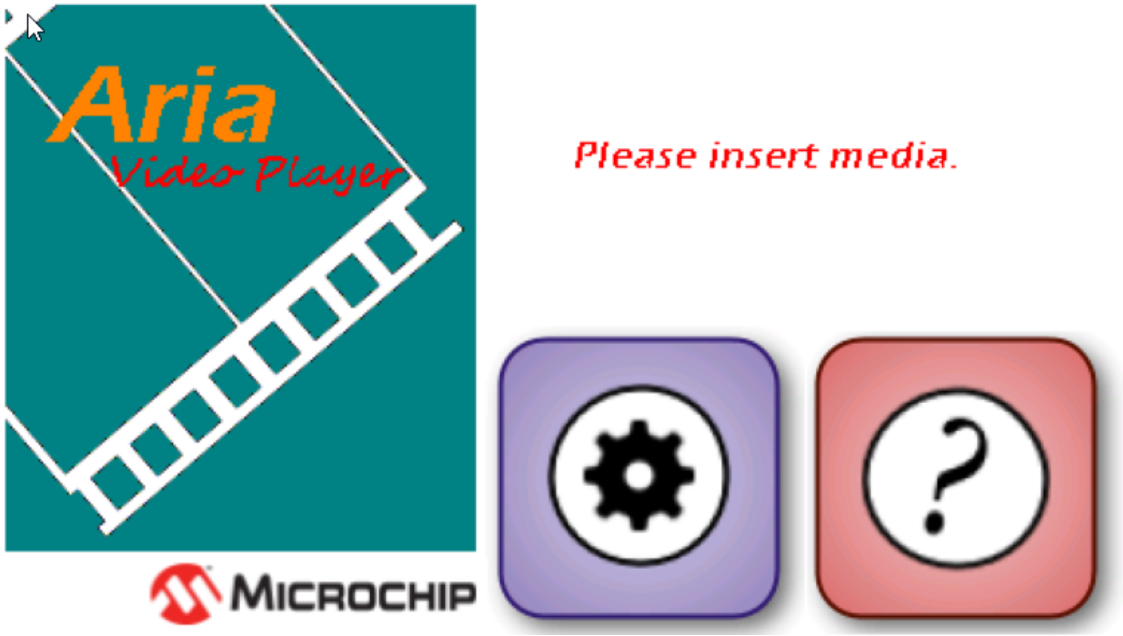
Running the Demonstration

This section provides information on how to run and use the application.

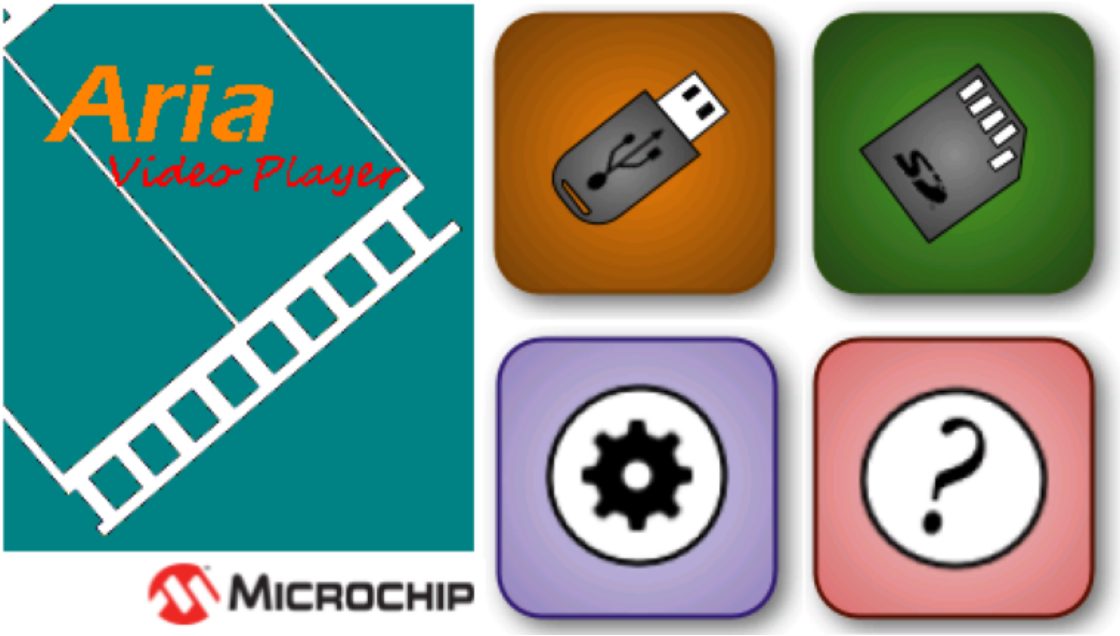
Description

On start-up, the application will display a splash screen.

After the splash-screen completes, the Main Menu screen shows. If there are no external media detected, the Main Menu will show a note to Insert Media.



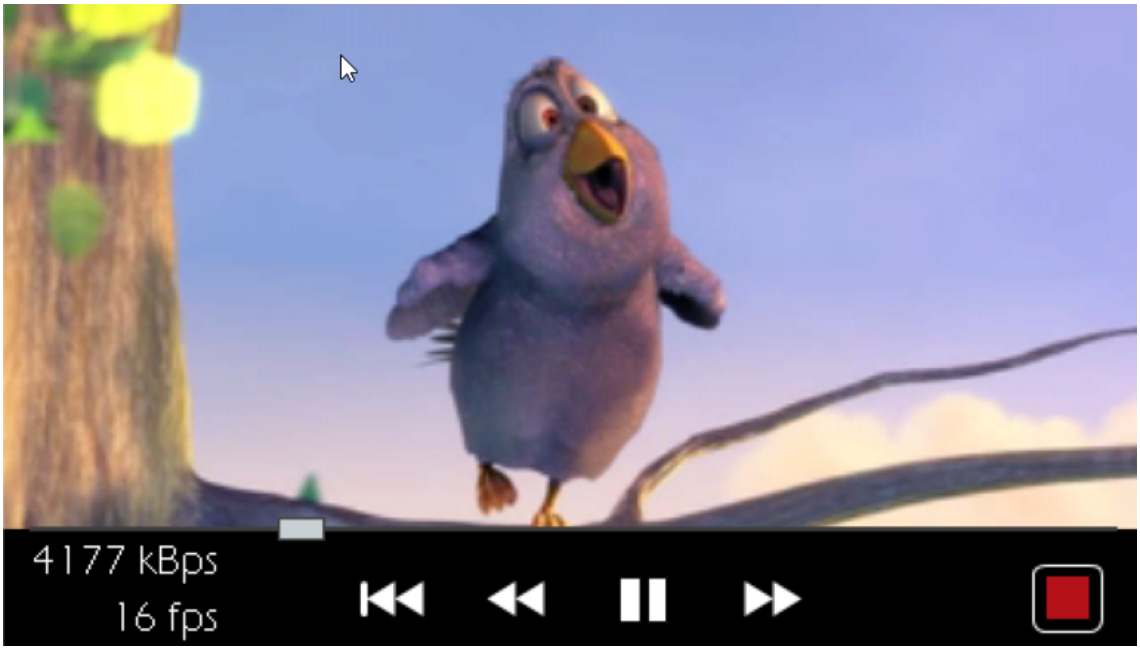
Insert media with video files to proceed. When a media is detected, it will show the USB Drive and/or SD Card buttons.



The menu buttons operate as follows:

Button	Description
	Play video from USB Drive
	Play video from microSD card
	Go to the Settings Screen
	Go to the Help Screen

Touching the USB or SD Card playback buttons will play videos from the selected media source. By default, it will play in full screen mode. Below is a sample frame from the “Big Buck Bunny” video clip sample. The clip is converted from the “Big Buck Bunny” video published by Blender Foundation, (c) copyright 2008, Blender Foundation, www.bigbuckbunny.org.



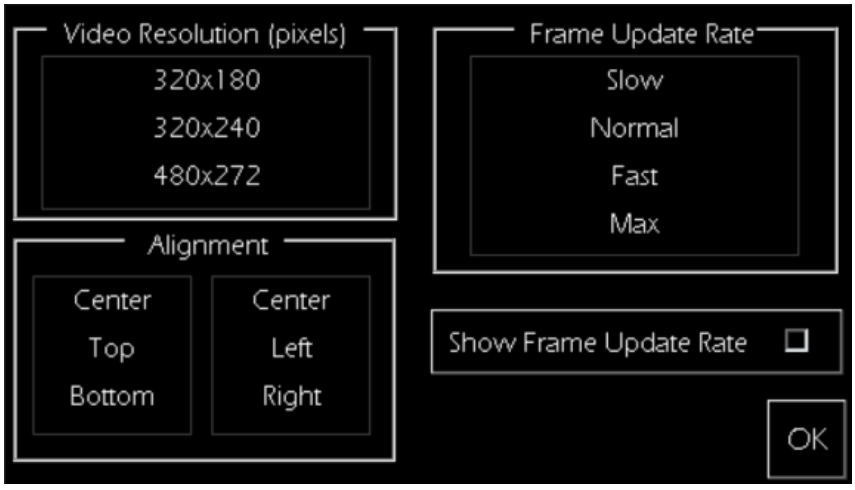
Touching the screen will hide or show the control panel. The following buttons are available in the control panel to control the video playback:

Button	Description
	Restarts playback to first frame.
	Skips frames to rewind. Pressing this button multiple times will increase the number of frames being skipped.
	Pauses the video.
	Starts/Resumes playback.
	Skips frames to fast-forward. Pressing this button multiple times will increase the number of frames being skipped.
	Stops playback and returns to Main Menu Screen.

A slider widget shows playback progress and provides a way to seek to a frame in the video.

If *Show Frame Update Rate* is enabled in the settings page, the control panel will also show the read throughput and frame update rate. These metrics are captured at real-time and updates every second.

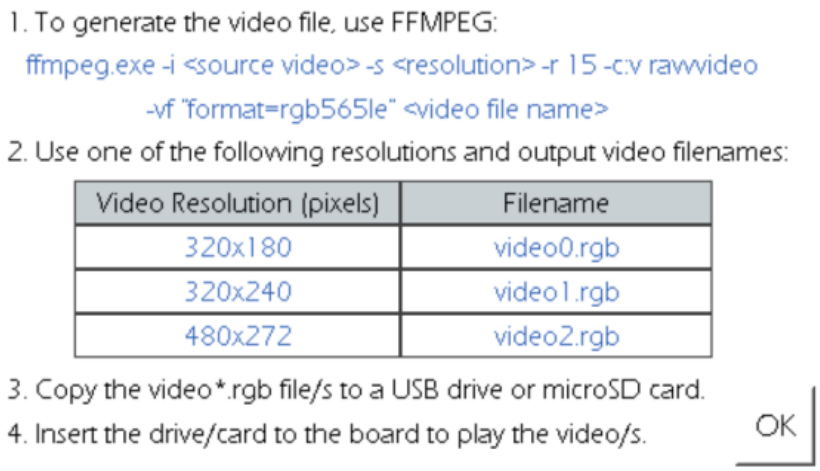
Touching the Settings button from the Main Menu screen opens the Settings Screen.



On the Settings Screen, the following settings can be configured:

Setting	Description
Video Resolution	Select the resolution of the video to play.
Frame Update Rate	Select the max rate at which the video frames are rendered. (Normal = up to 15 fps, Slow = up to 8 fps, Fast = up to 25 fps, Max = up to 80 fps)
Alignment	Select the horizontal and vertical position of the video. If the video size is smaller than the display size, the video control pane will be positioned opposite the horizontal position of the video.
Show Frame Update Rate	Touching this button down will show the frame rate and read throughput on the control panel during video playback.

For help on how to generate the video files, touch the HELP (?) button to show the instructions.



aria_weather_forecast

This demonstration provides a practical single-layered single-buffered application using the Aria User Interface Library.

Description

This application showcases an example of how a graphics controller with single graphical layer support can be achieved using the Aria Graphics Library. It also highlights how tightly integrated the MPLAB Harmony Graphics Composer Suite can be in supporting a rich feature set in an application.

Various Aria Graphics Library features such as multi-language localization support and tree-based architecture are highlighted in this demo, which is disguised as a user-interface for weather forecast application.

The demonstration launches with a splash screen highlighting basic motion capability supported by Aria Graphics Library. When running the application, the user can interface with it via capacitive single-fingered touch.

Architecture

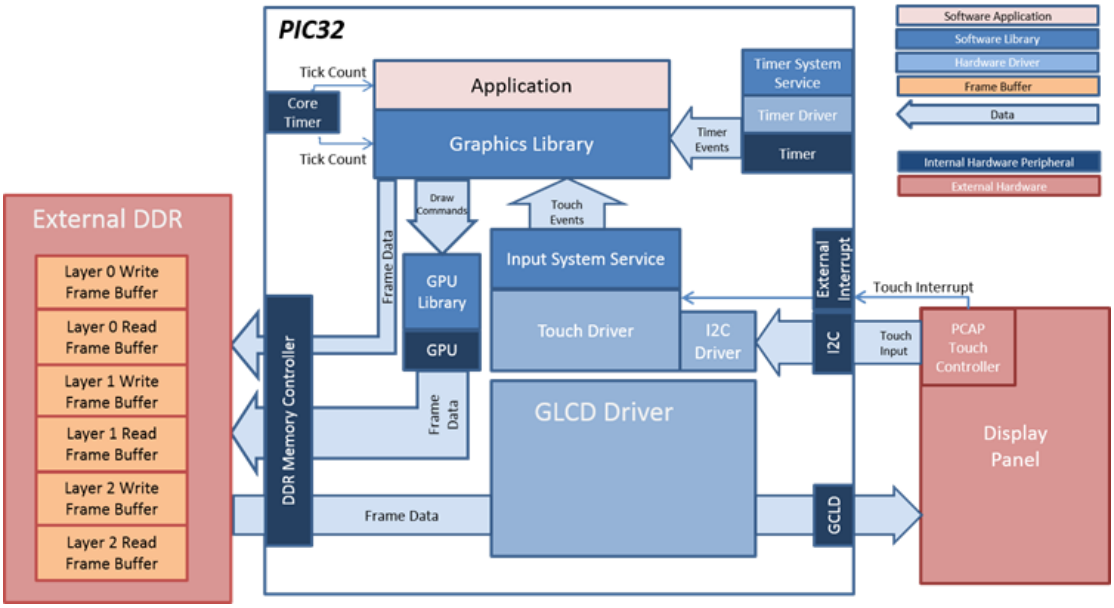
The following diagrams show the various software and hardware components for each configuration.

pic32mz_da_sk_extddr_meb2, and pic32mz_da_sk_extddr_meb2_wvga

For these configurations, the application uses the Graphics library to render graphics to the display. The Graphics library passes draw commands into the GPU Library, which in-turn draws the widgets and images to the frame buffer that is stored in an external DDR2. Via the DDR2 Memory Controller, the GLCD display controller peripheral continuously transfers frame data from the frame buffer onto to the LCD display.

The application also features user touch input through the integrated touch screen on the display panel. Touch input from the touch controller goes thru the I2C port, and the Input System Service acquires the touch input information from the touch and I2C drivers. The Input System Service sends touch events to the Graphics library which processes these events and updates the frame data accordingly.

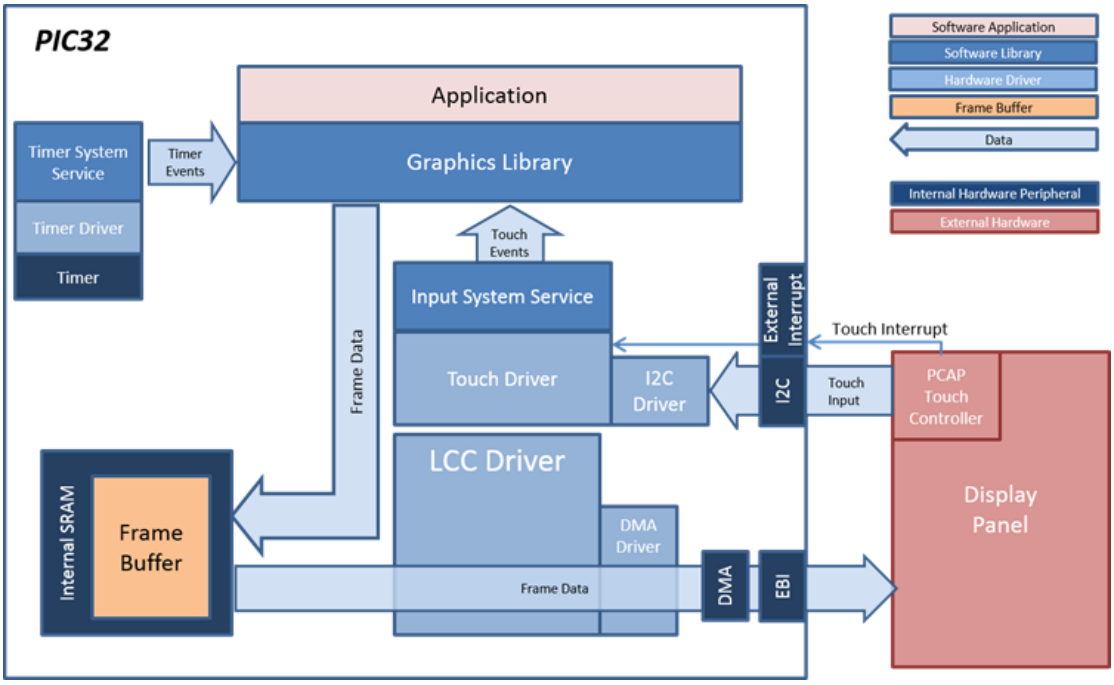
The core timer is used by the application and the Graphics library to manage the movement of the splash screen.



pic32mz_ef_sk_meb2

For this configuration, the application uses the Graphics library to render graphics to the display. The Graphics library draws the widgets and images to the frame buffer that is stored on the internal SRAM. Using the DMA, the Low-Cost Controller-less (LCC) display driver continuously transfers frame data from the frame buffer out to the LCD display.

The application also features user touch input through the integrated touch screen on the display panel. Touch input from the touch controller goes thru the I2C port, and the Input System Service acquires the touch input information from the touch and I2C drivers. The Input System Service sends touch events to the GFX library which processes these events and updates the frame data accordingly.



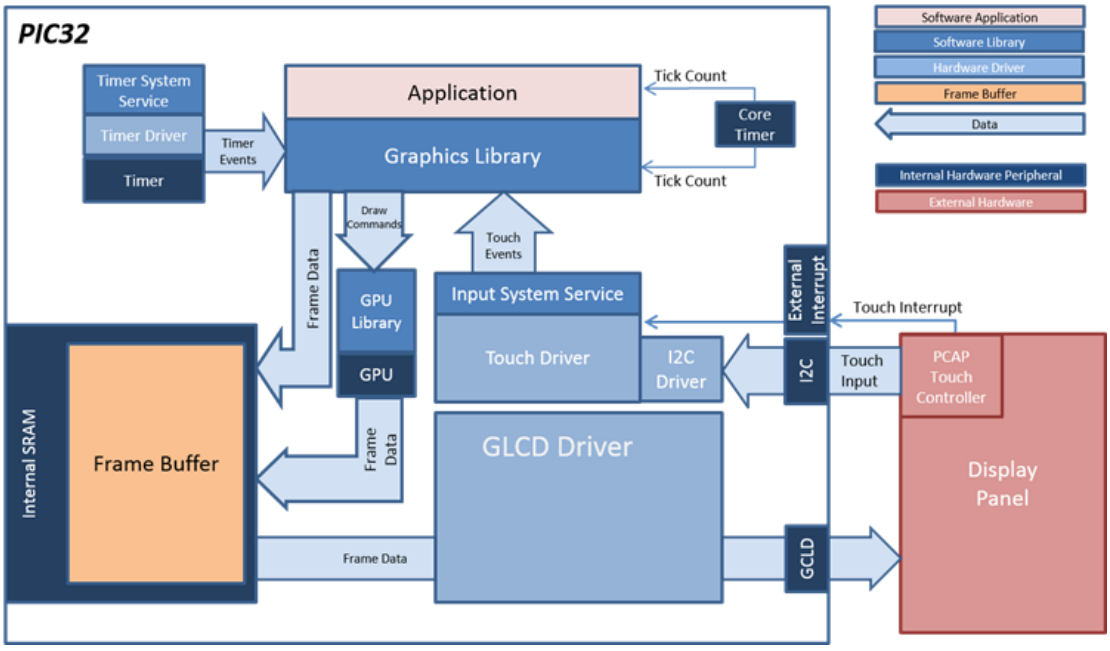
pic32mz_da_sk_noddr_meb2

For this configuration, the aria_quickstart application uses the Graphics Library to render graphics to the display. The Graphics Library draws the widgets and images to the write frame buffer that is stored on the internal SRAM.

Because the color format is set to RGB565 (16-bit), a frame buffer requires 261120 bytes of memory (480 wide x 272 height x 2 bytes per pixel). A double-buffered configuration is possible with 640 kilobytes of internal SRAM in the MZ DA device, but in the case, the design is set to single buffer.

Directly from the frame buffer on internal memory, the GLCD display controller peripheral continuously transfers frame data onto to the LCD display.

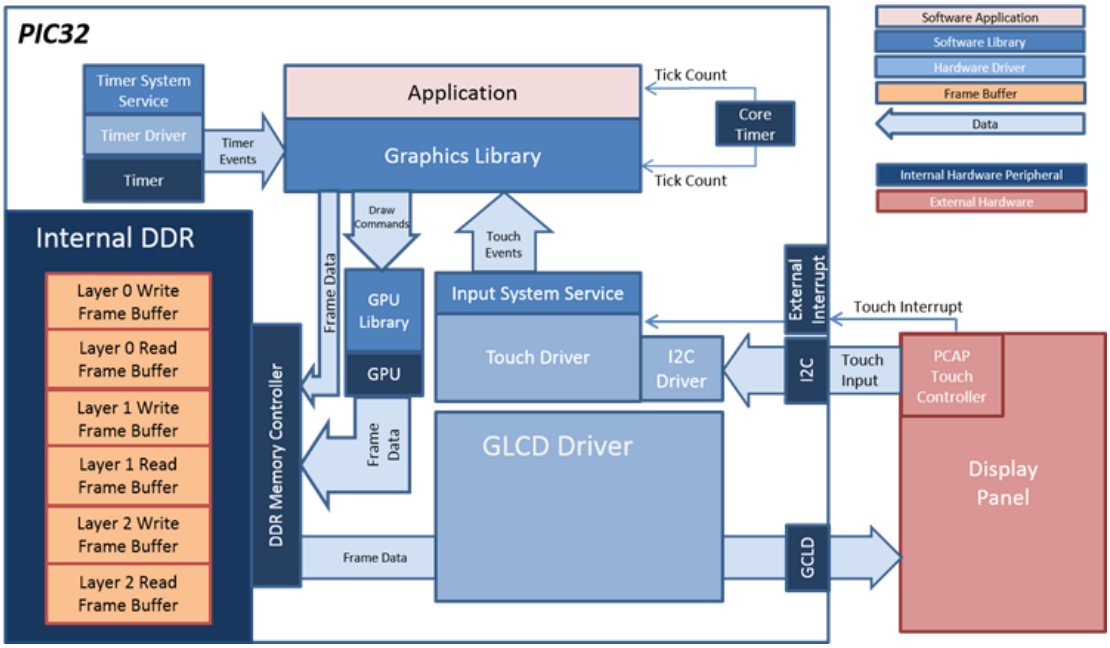
The application also features user touch input through the integrated touch screen on the display panel. Touch input from the touch controller goes through the I2C port, and the Input System Service acquires the touch input information from the touch and I2C drivers. The Input System Service sends touch events to the Graphics Library, which processes these events and updates the frame data accordingly.



pic32mx_da_sk_intddr_meb2, and pic32mz_da_sk_intddr_meb2_wvga

For these configurations, the application uses the Graphics Library to render graphics to the display. The Graphics library passes draw commands into the GPU Library, which in turn draws the widgets and images to the three individual write frame buffers (one for each layer) that are stored in an internal DDR2. Via the DDR2 Memory Controller, the GLCD display controller peripheral continuously transfers frame data from all three read buffers onto the LCD display. The write and read frame buffer pairs are swapped independently as required when the Graphics Library is done rendering to the write frame buffer and GLCD driver signals the GLCD peripheral to change its read location.

The application also features user touch input through the integrated touch screen on the display panel. Touch input from the touch controller travels through the I2C port, and the Input System Service acquires the touch input information from the touch and I2C drivers. The Input System Service sends touch events to the Graphics library which processes these events and updates the frame data accordingly.



Demonstration Features

- Integrated PCAP Touch Input (see mXT336T Touch Driver Library
- Low-Cost Controllerless (LCC) Graphics driver on the PIC32MZ EF device
- Single graphic layer supported via the GLCD peripheral on the PIC32MZ DA device
- 16-bit RGB565 color depth support (65536 unique colors) for PIC32MZ EF device (see Volume III: MPLAB Harmony Configurator (MHC))
- 32-bit RGBA8888 color depth support (16.7 million unique colors) for PIC32MZ DA device (see *Volume III: MPLAB Harmony Configurator (MHC) > MPLAB Harmony Graphics Composer User's Guide > Graphics Composer Window User Interface > Options*)
- Language localization in English, Chinese Simplified, and Spanish[add link to (see *Volume III: MPLAB Harmony Configurator (MHC) > MPLAB Harmony Graphics Composer User's Guide > Graphics Composer Window User Interface > Widget Tool Box Panel*)

- Image compression techniques using Run-Length Encoding and JPEG (see Enabling Demo Mode in a MPLAB Harmony Graphics Application
- Run-time JPEG decoding
- UTF-8 character font support (see *Volume III: MPLAB Harmony Configurator (MHC) > MPLAB Harmony Graphics Composer User's Guide > Graphics Composer Window User Interface > Font Assets*)
- Button and Image Widgets

Tools Setup Differences

"Use Graphics Stack" is selected in MHC. This enables the graphics and touch libraries and drivers for the BSP.

pic32mz_ef_sk_meb2

- MHC changes: *Driver > I2C > I2C Clock Frequency* is set to 10 KHz.
- *Graphics Stack > Graphics Display > Select Display Type* > Custom Display
- *Graphics Stack > Graphics Display > Horizontal Pulse Width* > 41
- *Graphics Stack > Graphics Display > Vertical Pulse Width* > 9
- *Graphics Stack > Graphics Controller > Low Cost Controllerless > V-Sync Refresh Strategy* > Aggressive
- *Graphics Stack > Graphics Options > Enable Draw Pipeline* > Enable Alpha Blending (Off)
- *Device & Project Configuration > Project Configuration > XC32 (Global Options) > xc32-ld > General > Heap Size (bytes)* > 152400
- Pin Settings: External Interrupt 1 mapped to pin 23 (RE8)

pic32mz_da_sk_extddr_meb2, and pic32mz_da_sk_intddr_meb2

- *Device & Project Configuration > Project Configuration > XC32 (Global Options) > xc32-ld > General > Heap Size (bytes)* > 204800
- Pin Settings: External Interrupt 4 mapped to pin A14 (RB1)

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Aria Weather Forecast demonstration.

Description

To build this project, you must open the `aria_weather_forecast.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/gfx/aria_weather_forecast`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`aria_weather_forecast.X`	`<install-dir>/apps/gfx/aria_weather_forecast/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mz_da_sk_extddr_meb2	pic32mz_da_sk_extddr+meb2	Demonstration for the Multimedia Expansion Board II (MEB II) connected to the PIC32MZ Embedded Graphics with External DRAM (DA) Starter Kit.
pic32mz_da_sk_extddr_meb2_wvga	pic32mz_da_sk_extddr+meb2+wvga	Demonstration for the Multimedia Expansion Board II (MEB II) connected to the PIC32MZ Embedded Graphics with External DRAM (DA) Starter Kit with High-Performance (5") WVGA Display Module with maXTouch.
pic32mz_ef_sk_meb2	pic32mz_ef_sk+meb2	Demonstration for the Multimedia Expansion Board II (MEB II) connected to the PIC32MZ EF Starter Kit.
pic32mz_da_sk_intddr_meb2	pic32mz_da_sk_intddr+meb2	Demonstration for the Multimedia Expansion Board II (MEB II) connected to the PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit.
pic32mz_da_sk_intddr_meb2_wvga	pic32mz_da_sk_intddr+meb2+wvga	Demonstration for the Multimedia Expansion Board II (MEB II) connected to the PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit with High-Performance (5") WVGA Display Module with maXTouch.

pic32mz_da_sk_noddr_meb2	pic32mz_da_sk_noddr+meb2	Demonstration for the Multimedia Expansion Board II (MEB II) connected to the PIC32MZ Embedded Graphics with Disabled DRAM (DA) Starter Kit.

Configuring the Hardware

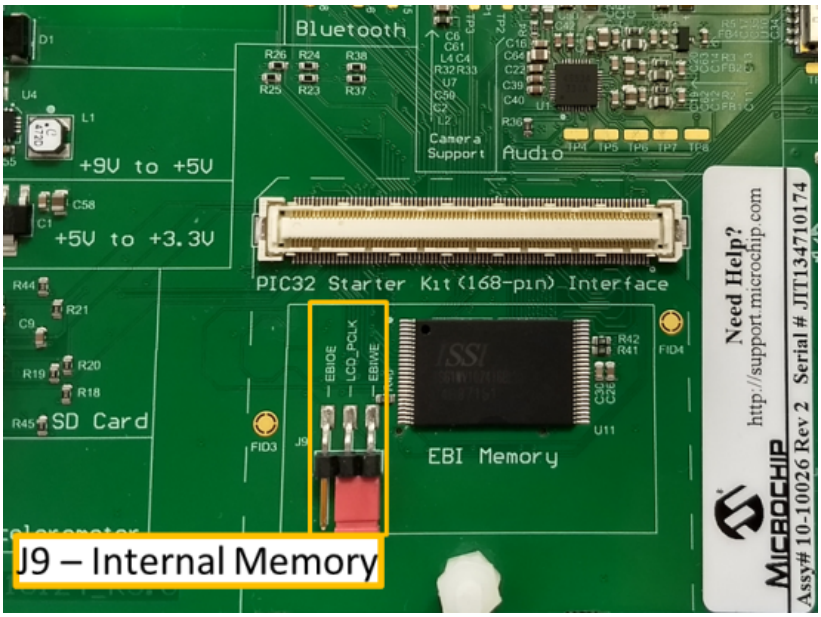
This section describes how to configure the supported hardware.

Description

PIC32MZ Embedded Graphics with Disabled DRAM (DA) Starter Kit, PIC32MZ Embedded Graphics with External DRAM (DA) Starter Kit, PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit, and MEB II (First and Second Generation)

Configurations: pic32mz_ef_sk_meb2

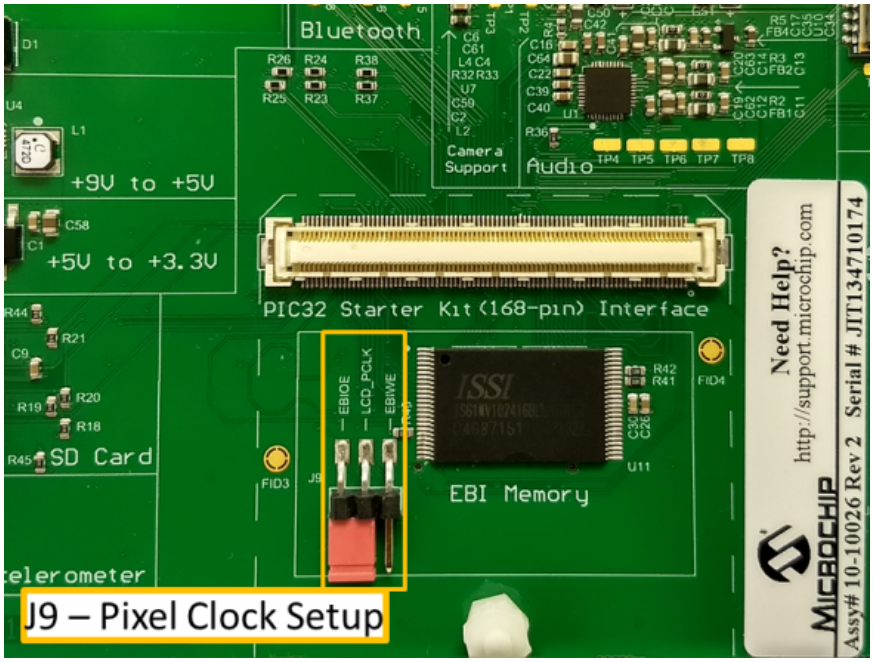
- This configuration requires that the J9 jumper be set to enable internal SRAM for the frame buffer. Set the J9 jumper to connect the EBIOE and LCD_PCLK pins. The J9 jumper is located on the bottom of the MEB II board, beneath where the starter kit is plugged into the board. Refer to the following image for the exact location.
- Connect the PIC32MZ EF Starter Kit board to the MEB II board
- Power up the board by connecting the power adapter to J3 power connector on the MEB II board or a powered USB cable to the USB DEBUG port on the Starter Kit board



PIC32MZ Embedded Graphics with External DRAM (DA) Starter Kit and First and Second Generation MEB II, plus displays.

Configurations: pic32mz_da_sk_extddr_meb2, pic32mz_da_sk_extddr_meb2_wvga, pic32mz_da_sk_intddr_meb2, pic32mz_da_sk_intddr_meb2_wvga, and pic32mz_da_sk_noddr_meb2

- These configurations require that the J9 jumper be set to provide the GLCD's pixel clock. Set the J9 jumper to connect EPIOE to LCD_PCLK. The J9 jumper is located on the bottom of the MEB II board, beneath where the starter kit is plugged into the board. Refer to the following figure for the exact location.
- Connect the PIC32MZ DA Starter Kit board to the MEB II board
- Power up the board by connecting the power adapter to J3 power connector on the MEB II board or a powered USB cable to the USB DEBUG port on the Starter Kit board



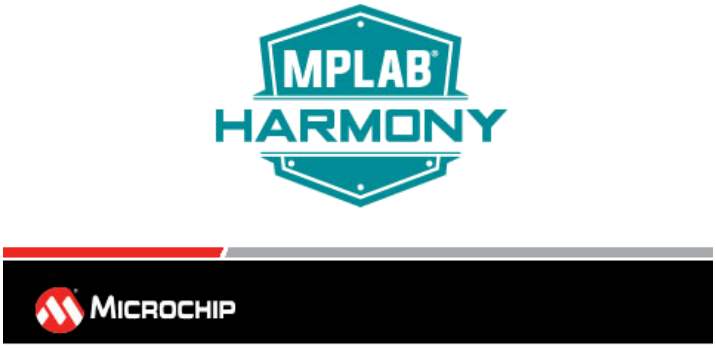
Running the Demonstration

This section provides instructions about how to build and run the Aria Weather Forecast demonstration.

Description

Splash Screen

When power-on is successful, the demonstration will display an animated splash screen:



Main Screen



Touch the Cloud to change the weather between Cloudy, Rainy, and Sunny.



blank_quickstart

This application demonstrates how to run custom graphics applications or integrate third-party graphics libraries with the MPLAB Harmony Hardware Abstraction Layer (HAL).

Description

This application demonstrates a simple way to create and run a custom graphics application that directly uses the MPLAB Harmony Hardware Abstraction Layer (HAL) to show images on the screen. The same model can also be used to integrate third-party graphics libraries into MPLAB Harmony. The Aria graphics library is disabled and not used in this application.

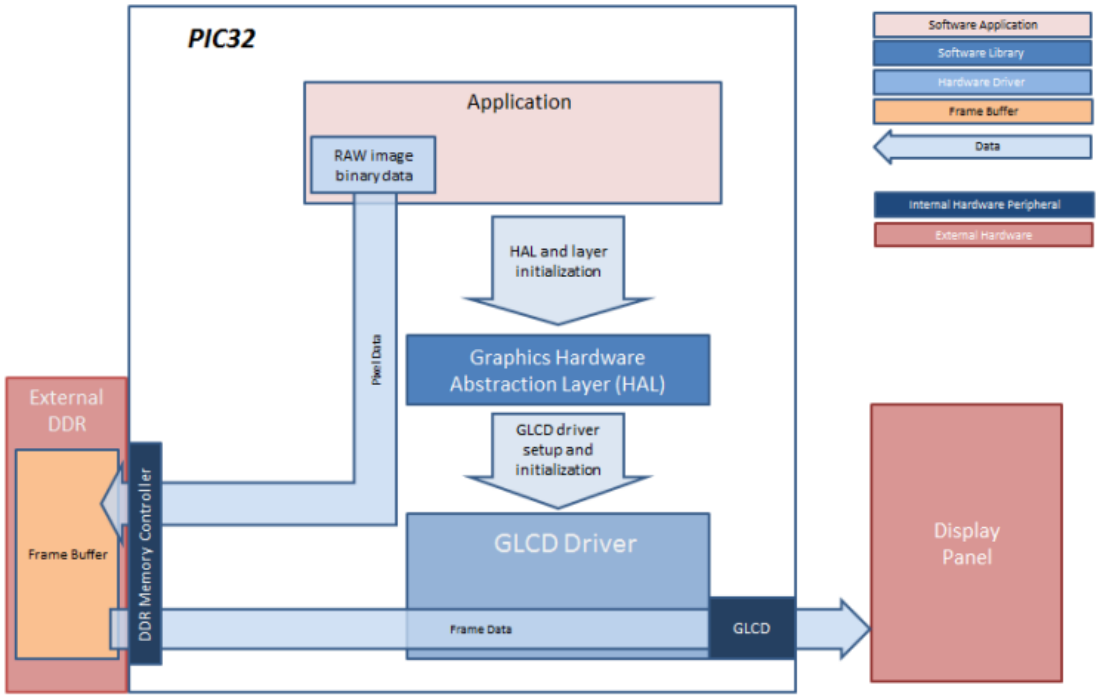
Architecture

The diagrams below show the various software and hardware blocks used in this application:

pic32mz_da_sk_extddr_meb2

In this configuration, the application calls HAL APIs to initialize the HAL data structures and set up the HAL graphics layers. These API calls translate into Graphics LCD (GLCD) driver set up and initialization calls that configure the GLCD. After the initialization phase, the application transitions into the paint phase where it draws an image to the frame buffer in external DDR. The application uses a HAL API to get the start address of the frame buffer in the DDR and writes the pixel data directly to the frame buffer memory address.

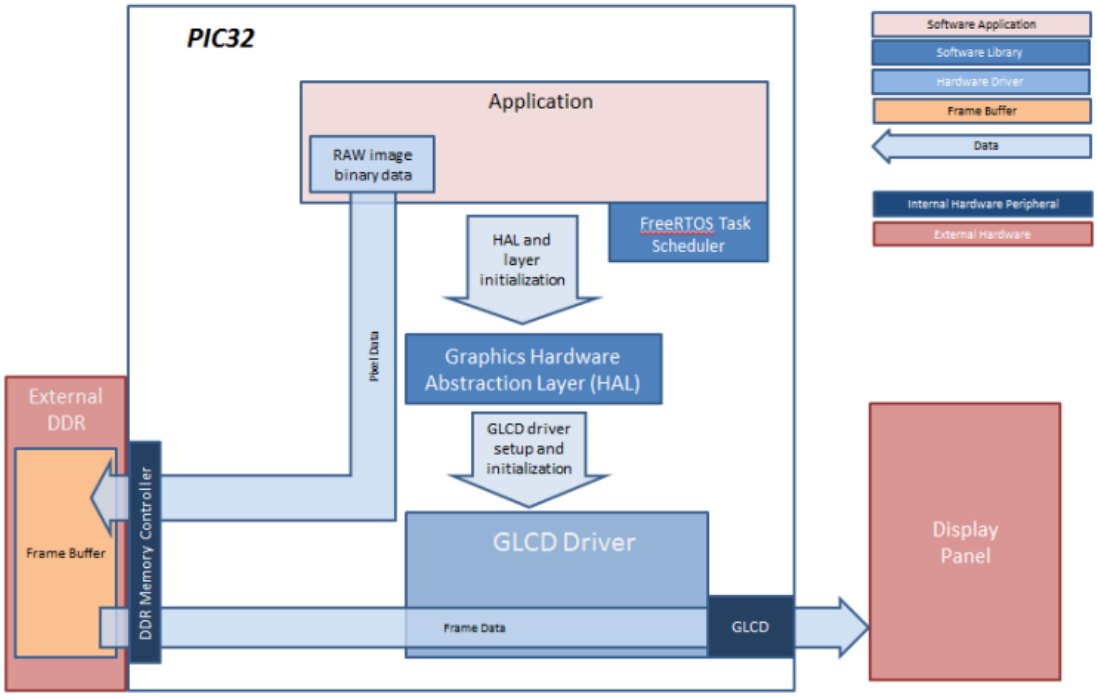
The GLCD hardware peripheral continuously refreshes the display panel with data from the frame buffer and the images are shown on the display.



pic32mz_da_sk_extddr_meb2_freertos

In this configuration, the application is executed within an OS task context that is scheduled by the FreeRTOS scheduler. When the application task is executed, it calls the HAL APIs to initialize the HAL data structures and setup the HAL graphics layers. These API calls translate into GLCD driver setup and initialization calls that configure the GLCD. After the initialization phase, the application transitions into the paint phase where it draws an image to the frame buffer in external DDR. The application uses a HAL API to get the start address of the frame buffer in the DDR and writes the pixel data directly to the frame buffer memory address.

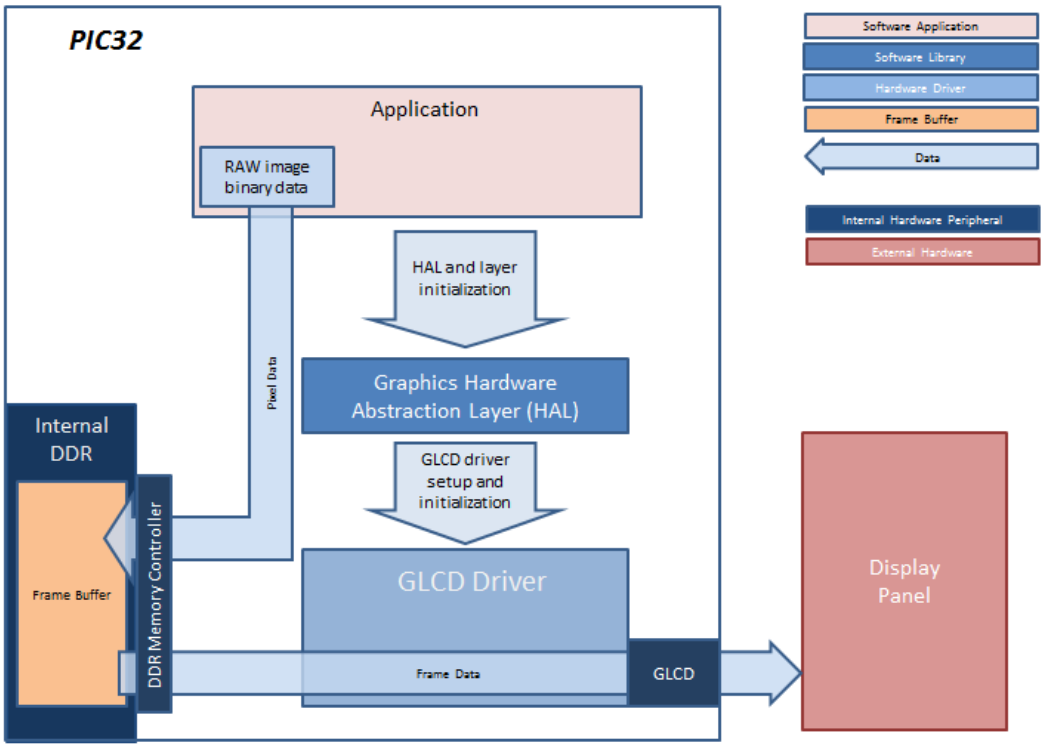
The GLCD hardware peripheral continuously refreshes the display panel with data from the frame buffer and the images are shown on the display.



pic32mz_da_sk_intddr_meb2

In this configuration, the application calls HAL APIs to initialize the HAL data structures and set up the HAL graphics layers. These API calls translate into Graphics LCD (GLCD) driver set up and initialization calls that configure the GLCD. After the initialization phase, the application transitions into the paint phase, where it draws an image to the frame buffer. The application uses a HAL API to get the start address of the frame buffer in internal DDR and writes pixel data directly to the frame buffer memory address.

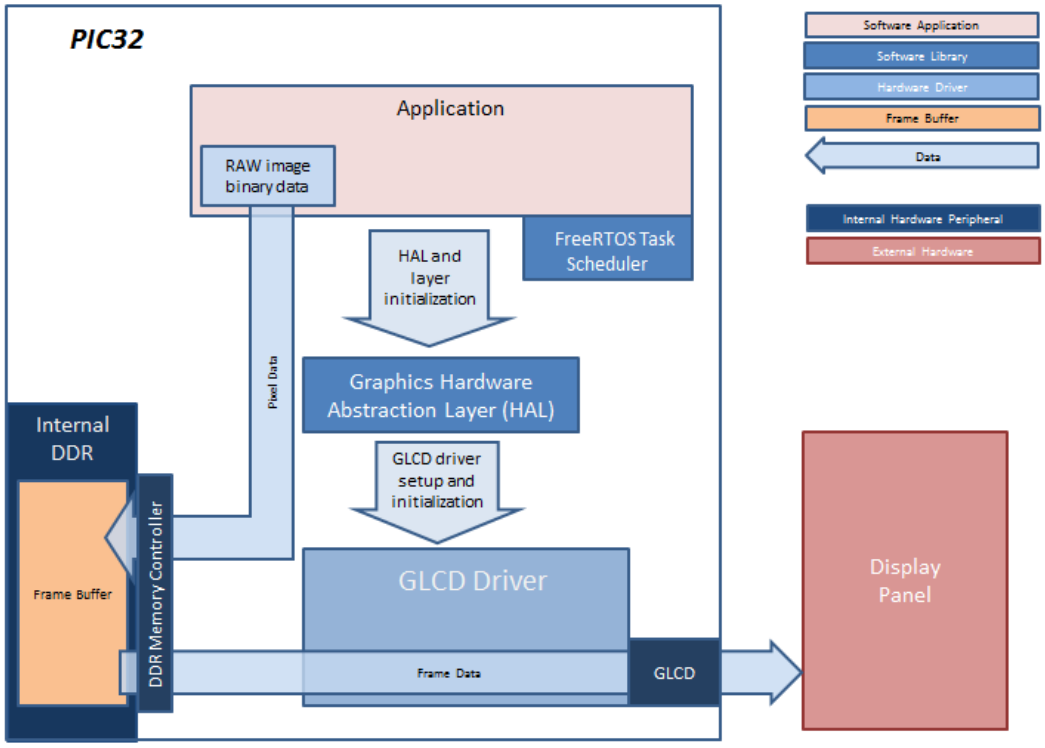
The GLCD hardware peripheral continuously refreshes the display panel with data from the frame buffer and the images are shown on the display.



pic32mz_da_sk_intddr_meb2_freertos

In this configuration, the application is executed within an operating system task context that is scheduled by the FreeRTOS scheduler. When the application task is executed, it calls the HAL APIs to initialize the HAL data structures and setup the HAL graphics layers. These API calls translate into GLCD driver setup and initialization calls that configure the GLCD. After the initialization phase, the application transitions into the paint phase, where it draws an image to the frame buffer. The application uses a HAL API to get the start address of the frame buffer in internal DDR and writes pixel data directly to the frame buffer memory address.

The GLCD hardware peripheral continuously refreshes the display panel with data from the frame buffer and the images are shown on the display.



Demonstration Features

- Graphics Hardware Abstraction Layer (HAL)

- GLCD Driver
- RTOS support (FreeRTOS)

Tools Setup Differences

- Enable *Use Graphics Stack* in MPLAB Harmony Configurator (MHC)
- In MHC, under the *Graphics Stack* options
  - Set *Graphics Processor > Select Processor Type* to *None*
  - Uncheck *Use Harmony Graphics Composer Suite* to disable the Aria User Interface Library
  - Uncheck *Graphics Options > Enable Draw Pipeline*
- For the FreeRTOS configuration (e.g., pic32mz_da_sk_intddr_meb2_freertos), set the following in MHC:
  - *Third Party Libraries > RTOS*, enable *Use RTOS*. Set *Select RTOS* to *FreeRTOS* and set *RTOS Configuration > Tick Rate (Hz)* to “100”
  - In *Application Configuration > Application 0 Configuration > RTOS Configuration*, set *Task Priority* to “2”, check *Use Task Delay* and set *Task Delay* to “10”.
  - In *Harmony Framework Configuration > RTOS Configuration*, set *System Task Priority* to “2”, check *Use System Task Delay* and set *Task Delay* to “10”.
  - In *Harmony Framework Configuration > Graphics Stack > Use Graphics Stack > Graphics RTOS Configuration*, set *Run Library Tasks As* to *Combine with System Tasks*.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Blank Quick Start demonstration.

Description

To build this project, you must open the `blank_quickstart.X` project in MPLAB X IDE, and then select the desired configuration. The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/gfx/blank_quickstart`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`blank_quickstart.X`	`<install-dir>/apps/gfx/blank_quickstart/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mz_da_sk_extddr_meb2	pic32mz_da_sk_extddr+meb2	Demonstration for PIC32MZ Embedded Graphics with External DRAM (DA) Starter Kit, Multimedia Expansion Board II (MEB II) and 4.3” WQVGA (480x272) Display.
pic32mz_da_sk_extddr_meb2_freertos	pic32mz_da_sk_extddr+meb2	Demonstration for PIC32MZ Embedded Graphics with External DRAM (DA) Starter Kit, Multimedia Expansion Board II (MEB II) and 4.3” WQVGA (480x272) Display running FreeRTOS.
pic32_mz_da_sk_intddr_meb2	pic32mz_da_sk_intddr+meb2	Demonstration for PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit, Multimedia Expansion Board II (MEB II) and 4.3” WQVGA (480x272) Display.
pic32_mz_da_sk_intddr_meb2_freertos	pic32mz_da_sk_intddr+meb2	Demonstration for PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit, Multimedia Expansion Board II (MEB II) and 4.3” WQVGA (480x272) Display running FreeRTOS.

Configuring the Hardware

This section describes how to configure the supported hardware.

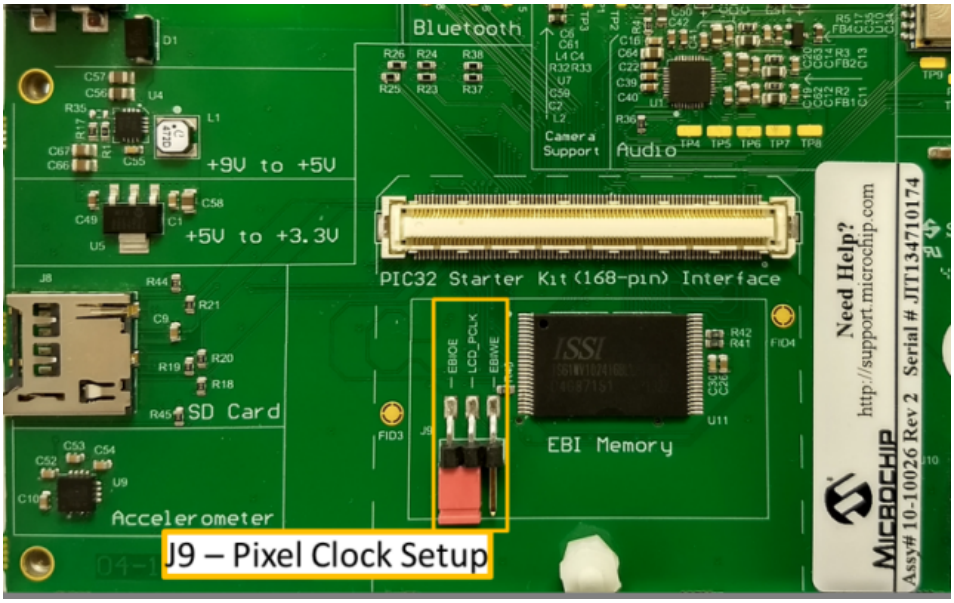
Description

PIC32MZ Embedded Graphics with External DRAM (DA) Starter Kit, PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit, and MEB II

Configurations: pic32_mz_da_sk_intddr_meb2, pic32_mz_da_sk_intddr_meb2_freertos, pic32mz_da_sk_extddr_meb2,

pic32mz_da_sk_extddr_meb2_freertos

- On the MEB II, the EBIOE and LCD_PCLK (J9) must be jumpered. A connection establishes the GLCD's pixel clock output timing. The external SRAM memory on the board is disabled. The jumper (J9) is available on the bottom side of the MEB II board under the starter kit. The J9 jumper is located on the bottom of the MEB II board, beneath where the starter kit is plugged into the board. Refer to the following image for the exact location.
- Connect the PIC32MZ DA Starter Kit board to the MEB II board.
- Power up the board by connecting the power adapter to J3 power connector on the MEB II board or a powered USB cable to the USB DEBUG port on the Starter Kit board

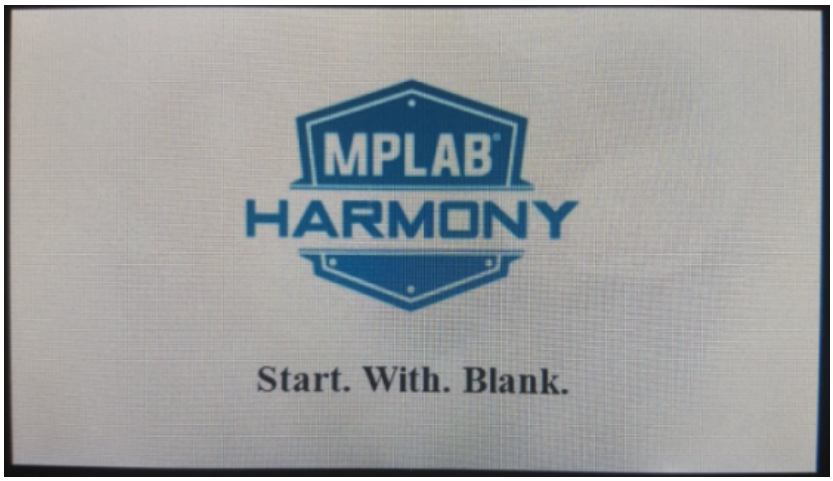


Running the Demonstration

This section provides information on how to run and use the application.

Description

Once the board is powered on, the application will run and show the following image on the display panel.



emwin_multilanguage

This demonstration is a Graphical User Interface (GUI) application that uses the SEGGER emWin Graphics Library and demonstrates the capability of emWin to render multiple languages (English and Chinese) on-screen by using the Font Generator tool provided by SEGGER.

Description

The application uses the third-party SEGGER emWin Graphics Library to render graphics to the display. emWin provides user interface tools like GUIBuilder to create the graphic UI design, which is then imported into the project as C files. This UI design includes widget creation, placement and look and feel. It does not include incorporated events such as touch, etc., supported by the MPLAB Harmony Aria User Interface Library.

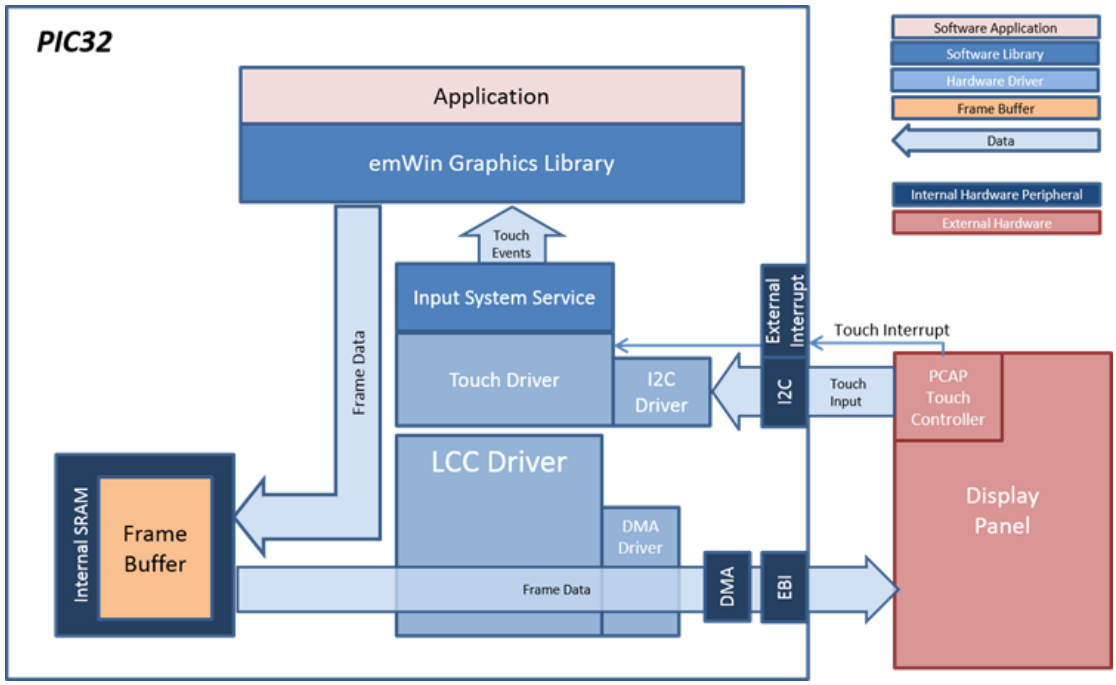
The Graphics Library draws the widgets and images to the frame buffer that is stored in an internal SRAM. This demonstration we uses another

tool called the Font Converter tool that is provided by SEGGER but is not a part of the default emWin Pro package.

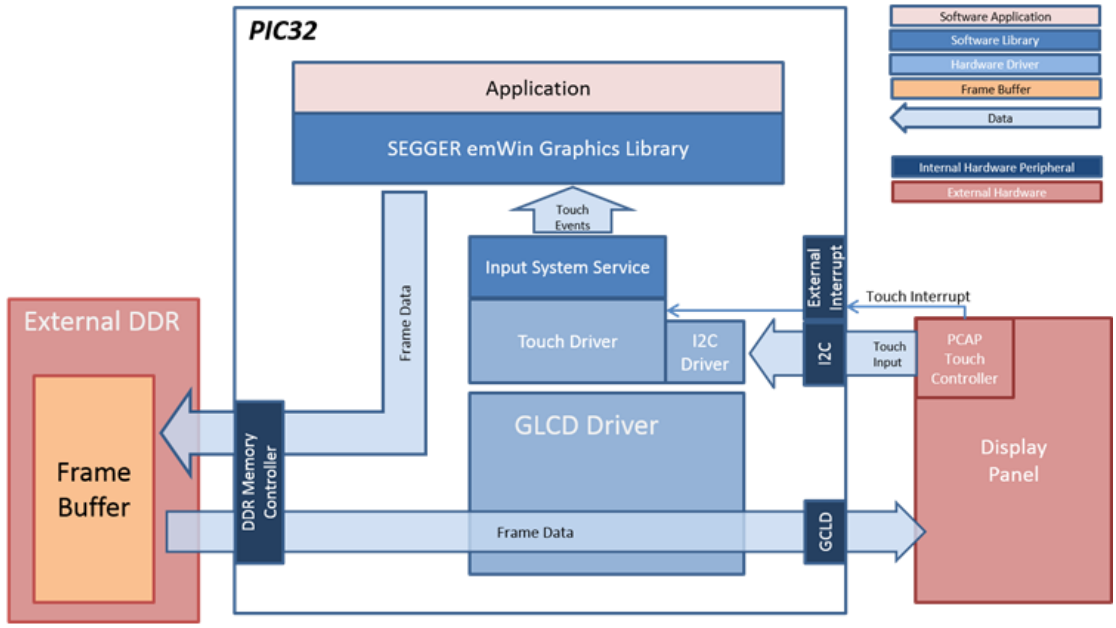
Using the DMA, the Display Driver continuously transfers frame data from the read frame buffer out to the LCD display. The application also features user touch input through the integrated touch screen on the display panel. Touch input from the touch controller goes through the I2C port, and the Input System Service acquires the touch input information from the Touch and I2C drivers. The Input System Service sends touch events to the Graphics Library, which processes these events and updates the frame data accordingly.

Another important part is the integration of the display controller. This demonstration has multiple configurations and each requires a suitable display controller/driver strategy for driving the pixel information from the emWin Graphics Library to the display. These configurations require appropriate APIs to be available to the SEGGER emWin Graphics Library to drive the pixel information to the display. Currently, this is handled by the wrapper layer provided in MPLAB Harmony, which generates the suitable APIs in the `LCDConf.c` file. This file will be generated by the emWin GUI Wrapper and will be added within the project logical folder `system_config\third_party\gfx\emwin\config`.

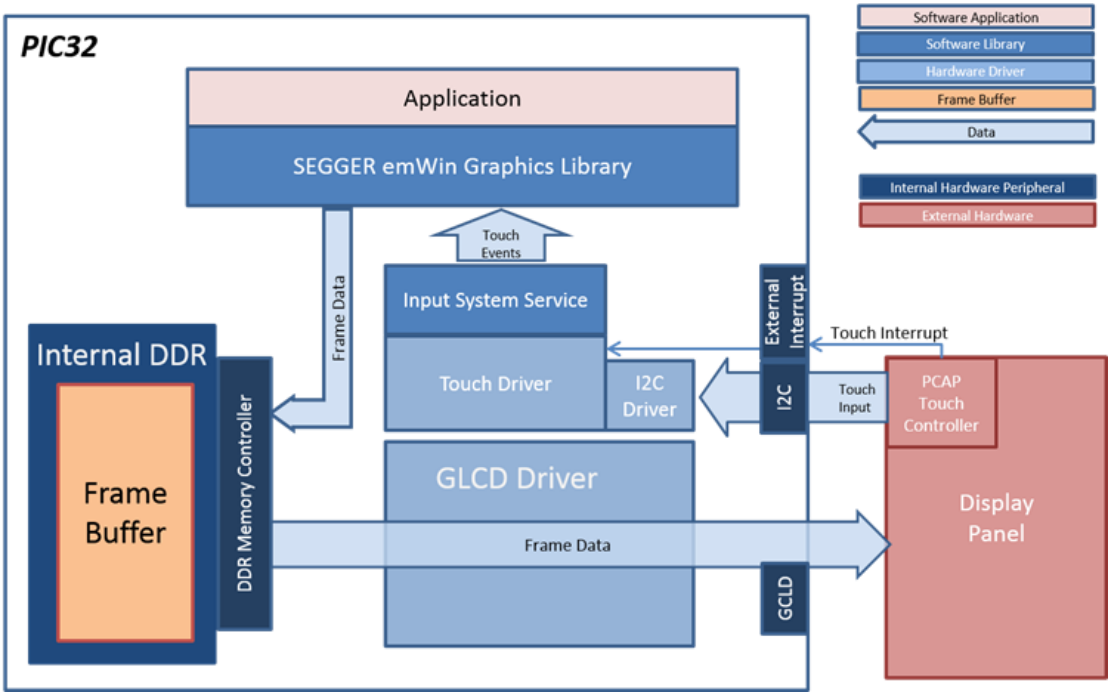
pic32mz_ef_sk_meb2



pic32mz_da_sk_extddr_meb2



pic32mz_da_sk_intddr_meb2



Demonstration Features

- SEGGER emWin Graphics Library
- SEGGER emWin tools usage
- Font Converter-generated font file added to the application for Chinese fonts
- Input System Service
- Touch Driver
- DMA System Service
- Low-Cost Controllerless (LCC) Graphics Driver for the pic32mz_ef_sk_meb2 configuration
- GLCD Display driver for pic32mz_da_sk_extddr_meb2
- S1D driver wrappers for SEGGER emWin
- I2C Driver

Tools Setup Differences

- SEGGER emWin graphic development GUI tool: GUIBuilder.exe
- SEGGER emWin font generator: FontCvt.exe
- The MPLAB Harmony Graphics Stack still provides the display configuration and the display driver for the project

pic32mz_ef_sk_meb2 Configuration

- Use *Graphics Stack > Use Graphics Stack?* with a Graphics Controller type of Low Cost Controllerless
- Use *Third Party Libraries > Graphics > Use SEGGER emWin Graphics Library?* with USE SEGGER emWin Touch Wrapper and Use SEGGER emWin GUI Wrapper selected, and the *emWin GUI Wrapper > Color Mode* type of EMWIN_COLOR_MODE_RGB_565

pic32mz_da_sk_extddr_meb2 Configuration

- Use *Graphics Stack > Use Graphics Stack?* with a Graphics Controller type of GLCD
- Use *Third Party Libraries > Graphics > Use SEGGER emWin Graphics Library?* with USE SEGGER emWin Touch Wrapper and Use SEGGER emWin GUI Wrapper selected, and the *emWin GUI Wrapper > Color Mode* type of EMWIN_COLOR_MODE_ARGB_888

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the emWin Multi-language demonstration.

Description

To build this project, you must open the `emwin_multilanguage.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/gfx/emwin_multilanguage`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`emwin_multilanguage.X`	`<install-dir>/apps/gfx/emwin_multilanguage/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mz_ef_sk_meb2	pic32mz_ef_sk+meb2	SEGGER emWin GUI demonstration for the Multimedia Expansion Board II (MEB II) connected to the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit.
pic32mz_da_extddr_sk_meb2	pic32mz_da_sk_extddr+meb2	SEGGER emWin GUI demonstration for the Multimedia Expansion Board II (MEB II) connected to the PIC32MZ Embedded Graphics with External DRAM (DA) Starter Kit.
pic32mz_da_intddr_sk_meb2	pic32mz_da_sk_intddr+meb2	SEGGER emWin GUI demonstration for the Multimedia Expansion Board II (MEB II) connected to the PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MZ EF Starter Kit connected to the Multimedia Expansion Board II (MEB II)

Configuration: pic32mz_ef_sk_meb2

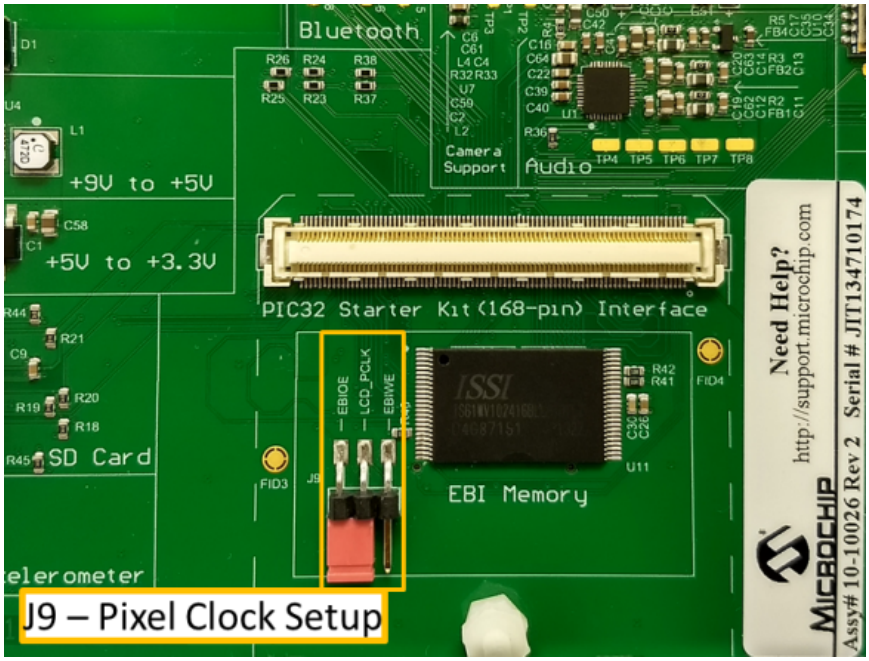
The MEB II uses LCC as the display controller driver. LCC can be configured to use internal or external RAM for the frame buffer. This demonstration uses the LCC driver with internal RAM as frame buffer. To configure the MEB II to use internal RAM, EBIWE and LCD_PCLK (J9) must be closed.

PIC32MZ Embedded Graphics with External DRAM (DA) Starter Kit or

PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kitwith the Multimedia Expansion Board II (MEB II)

Configuration: pic32mz_da_sk_extddr_meb2, pic32mz_da_sk_intddr_meb2

- This configuration requires that the J9 jumper be set to provide the GLCD's pixel clock. Set the J9 jumpber to connect EPIOE to LCD_PCLK. The J9 jumper is located on the bottom of the MEB II board, beneath where the starter kit is plugged into the board. Refer to the following figure for the exact location.
- Connect the PIC32MZ DA Starter Kit board to the MEB II board
- Power up the board by connecting the power adapter to J3 power connector on the MEB II board or a powered USB cable to the USB DEBUG port on the Starter Kit board

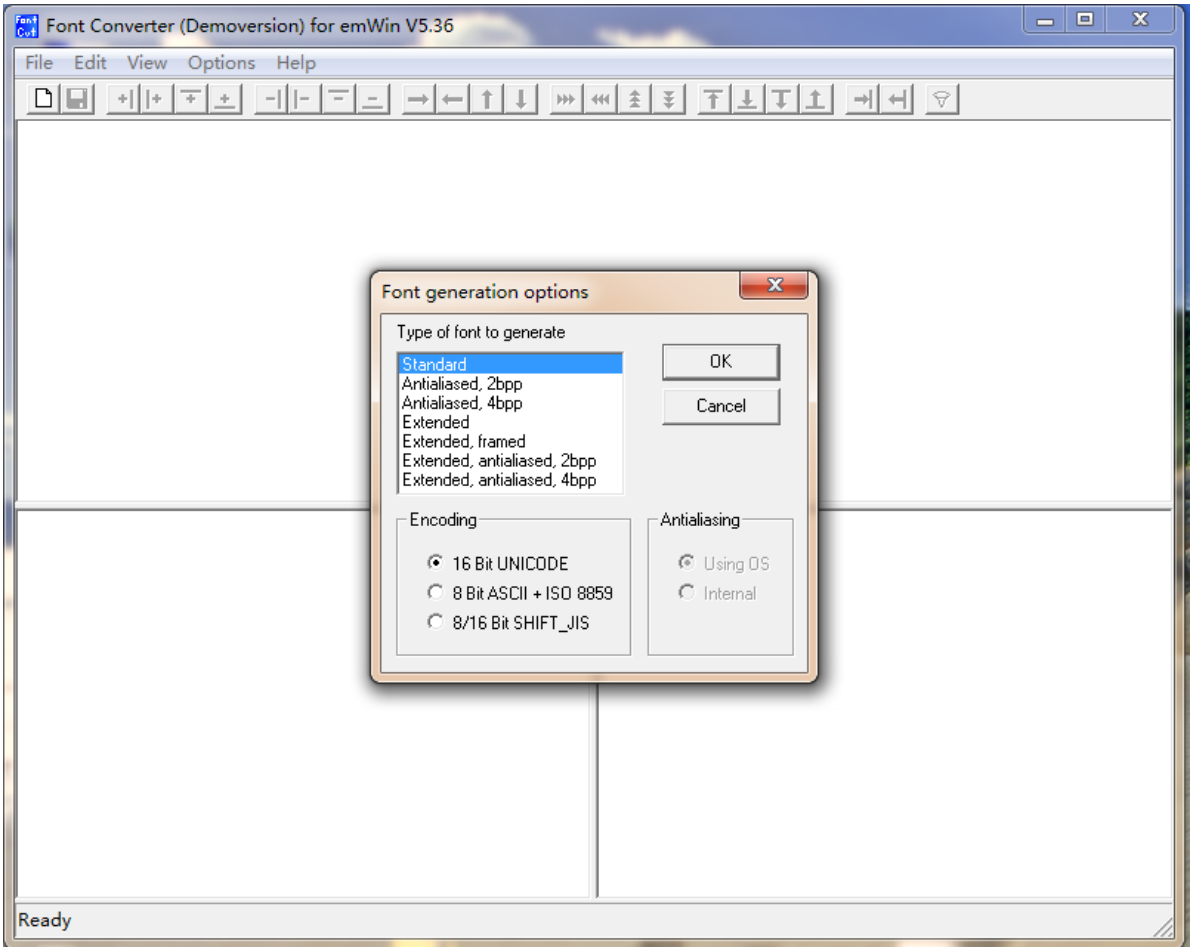


Adding a New Font File to the Application

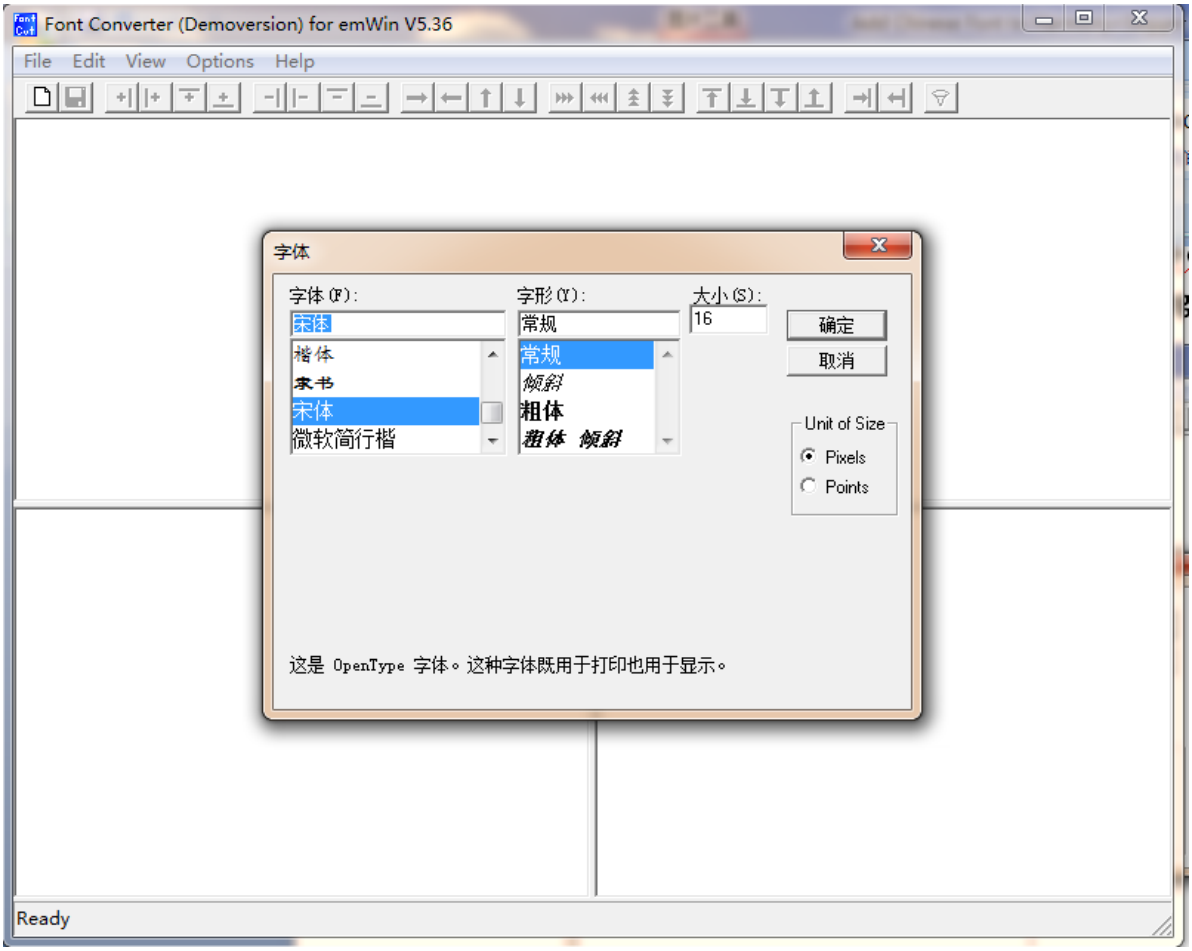
Description

Before checking how the demonstration functions, the following instructions privde a quick walk-through of how the font file is incorporated into the project. This font file is generated outside the project by a specific tool called Font Converter, which can be purchased from SEGGER (it is not a part of the emWin Pro package that is included with your installation of MPLAB Harmony.) A trial version of the Font Converter tool is provided with MPLAB Harmony.

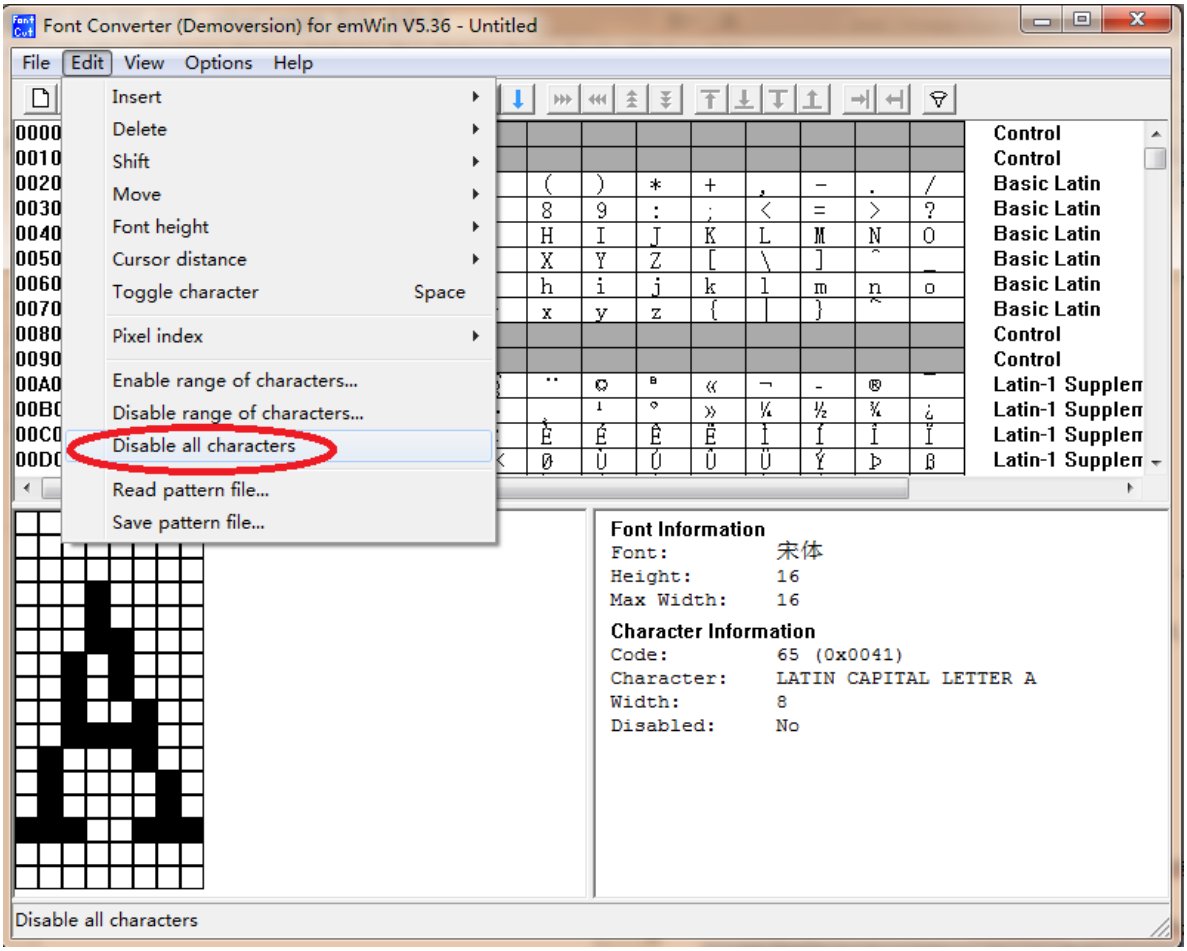
1. To create a font file using FontCvtDemo, ensure that it is installed. The installer is located in: `<install-dir>\utilities\segger\emwin`.
2. Start FontCvtDemo and select *File* > *New* and choose **Standard**.



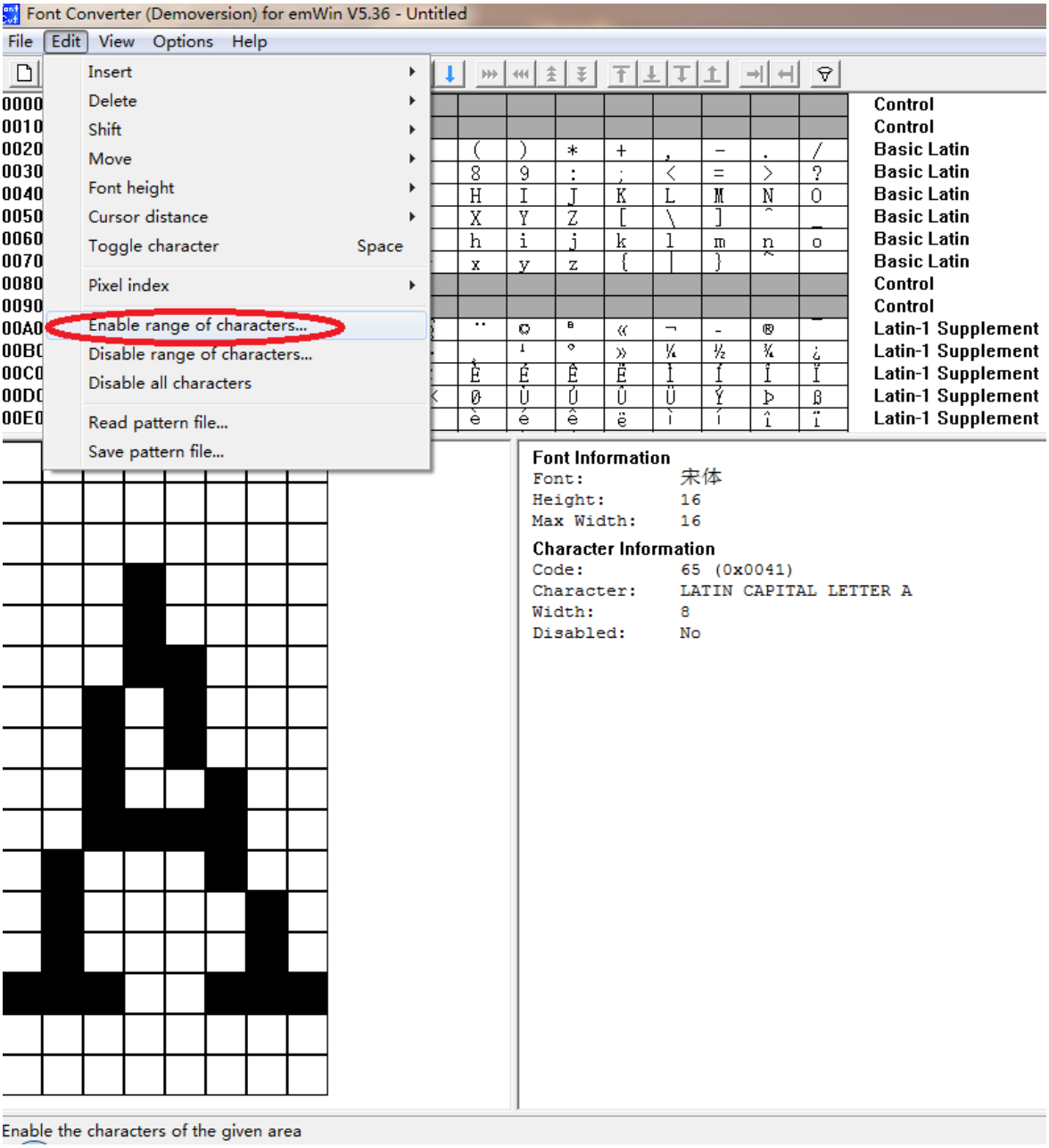
3. Select a Chinese Font.



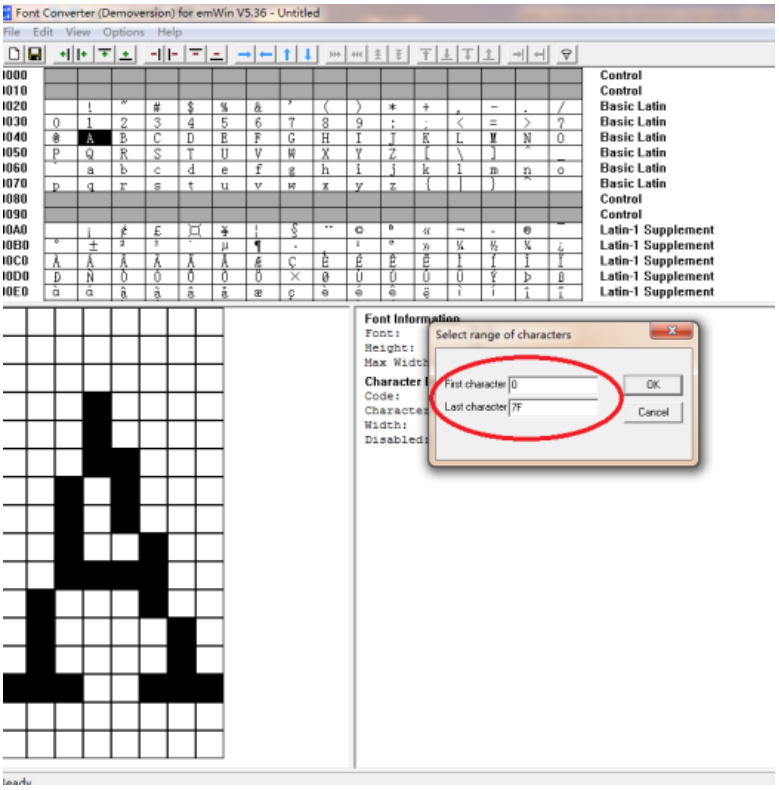
4. Disable all characters.



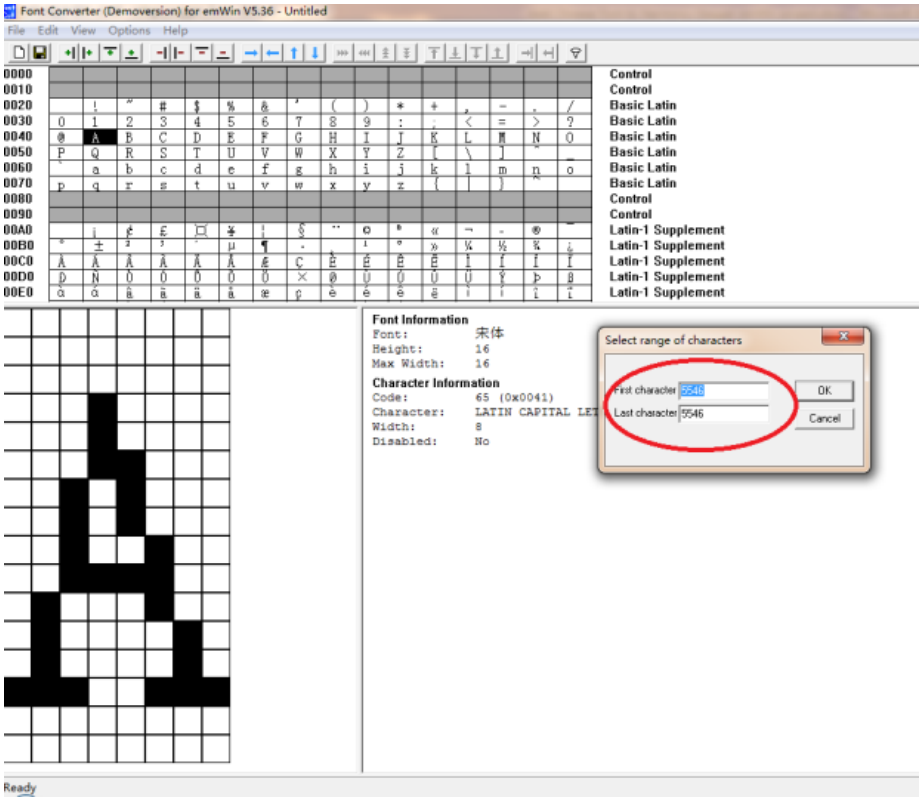
5. Enable all the characters needed by the Unicode.



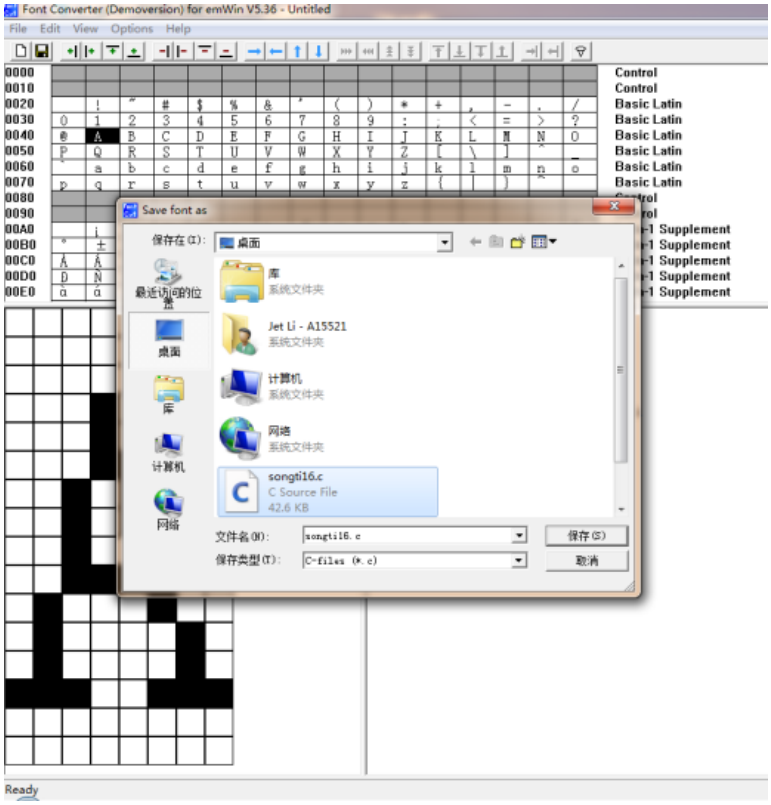
6. Add all ASCII characters.



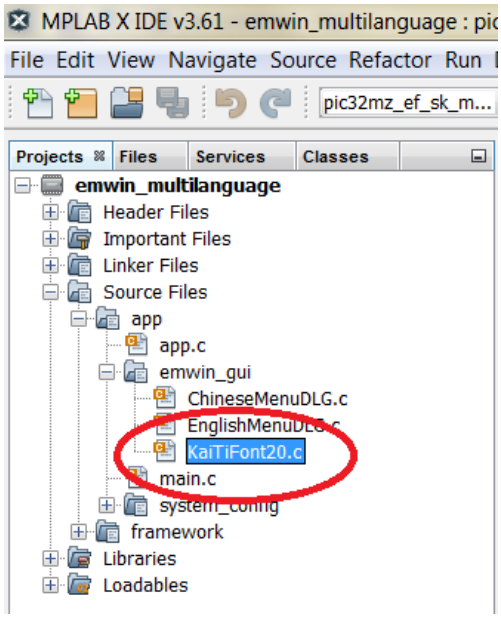
7. Add the Chinese characters one by one, for example, the Unicode for “???????” are “0x5fae,0x82af,0x5546,0x8d38,0x6b22,0x8fce,0x60a8”.



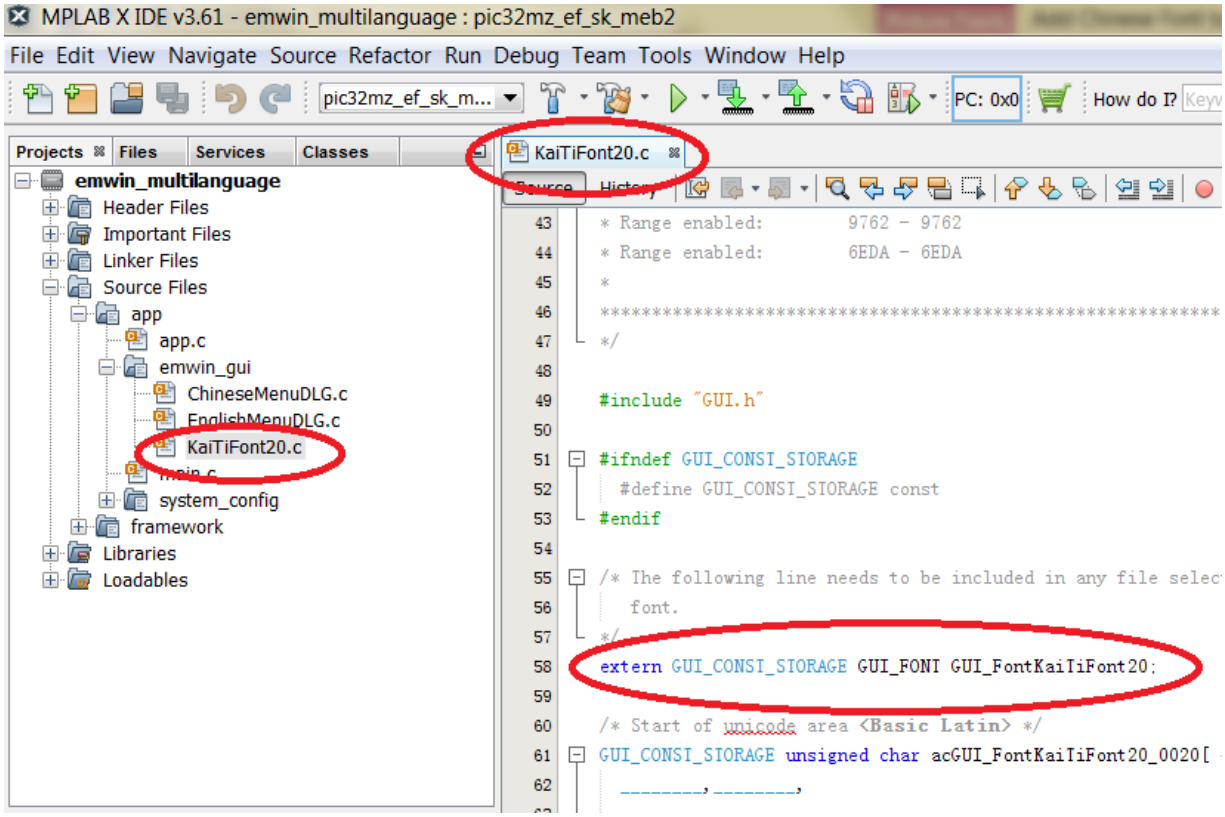
8. Save the font file.

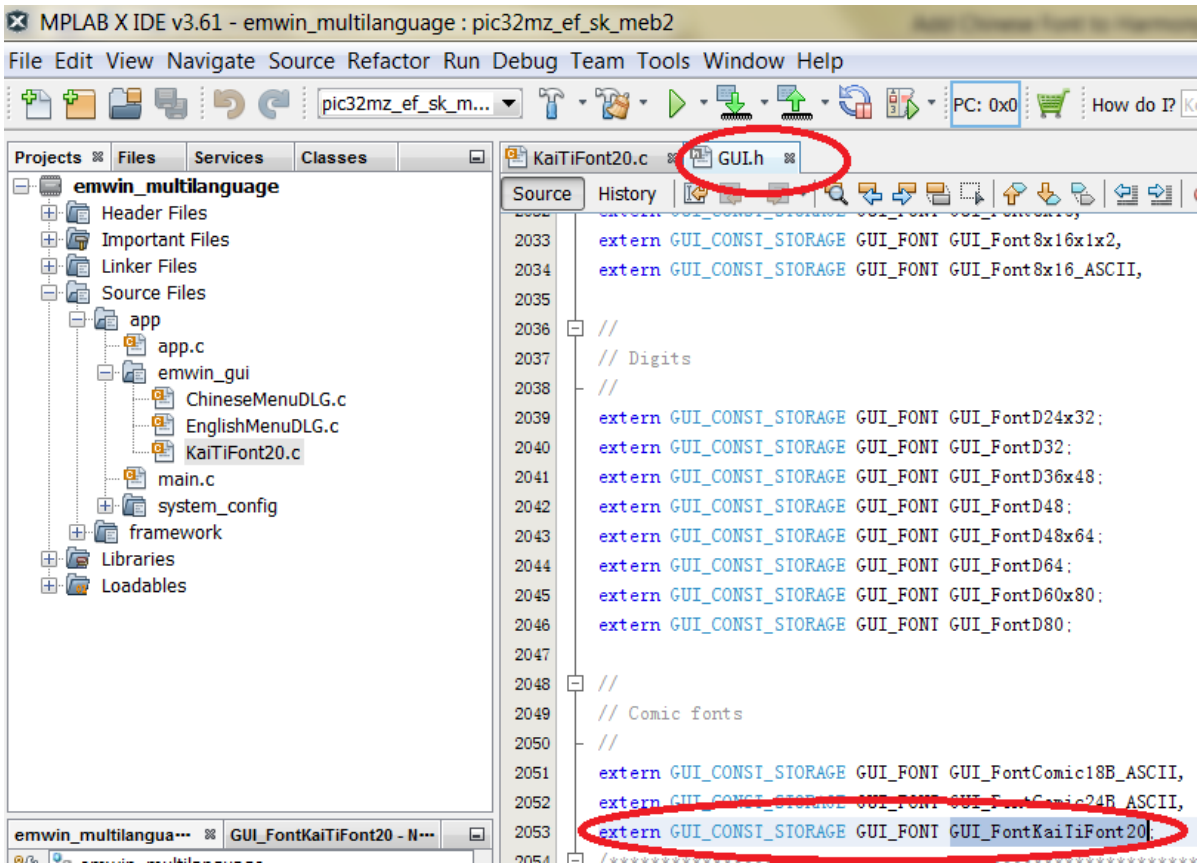


9. Add the font file to the MPLAB Harmony SEGGER project.

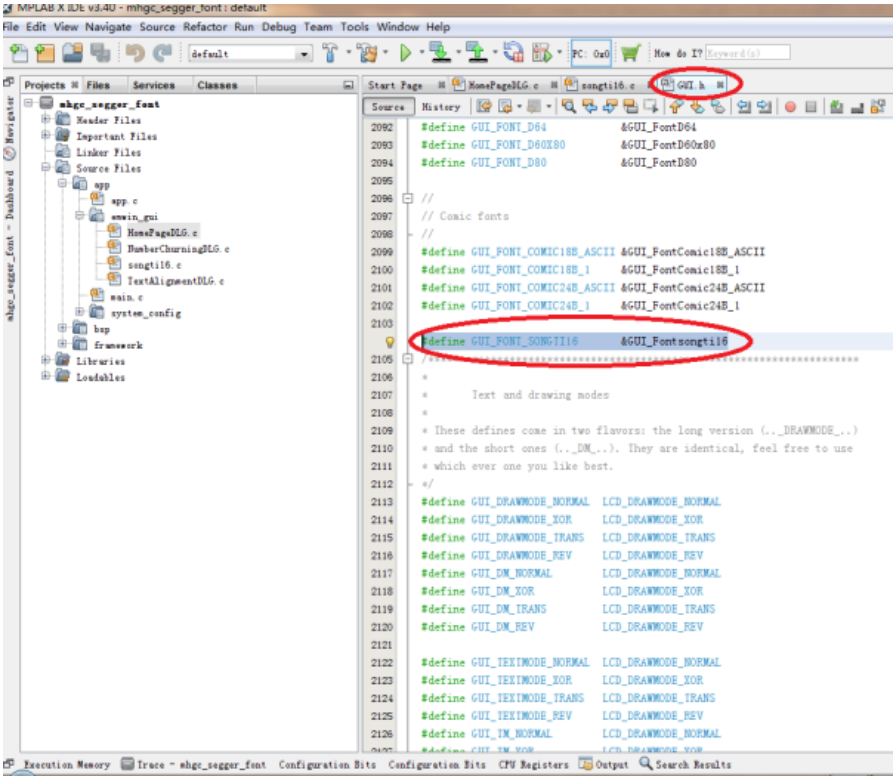


10. Copy the font extern definition from the font file (`songti16.c`) to `GUI.h`.

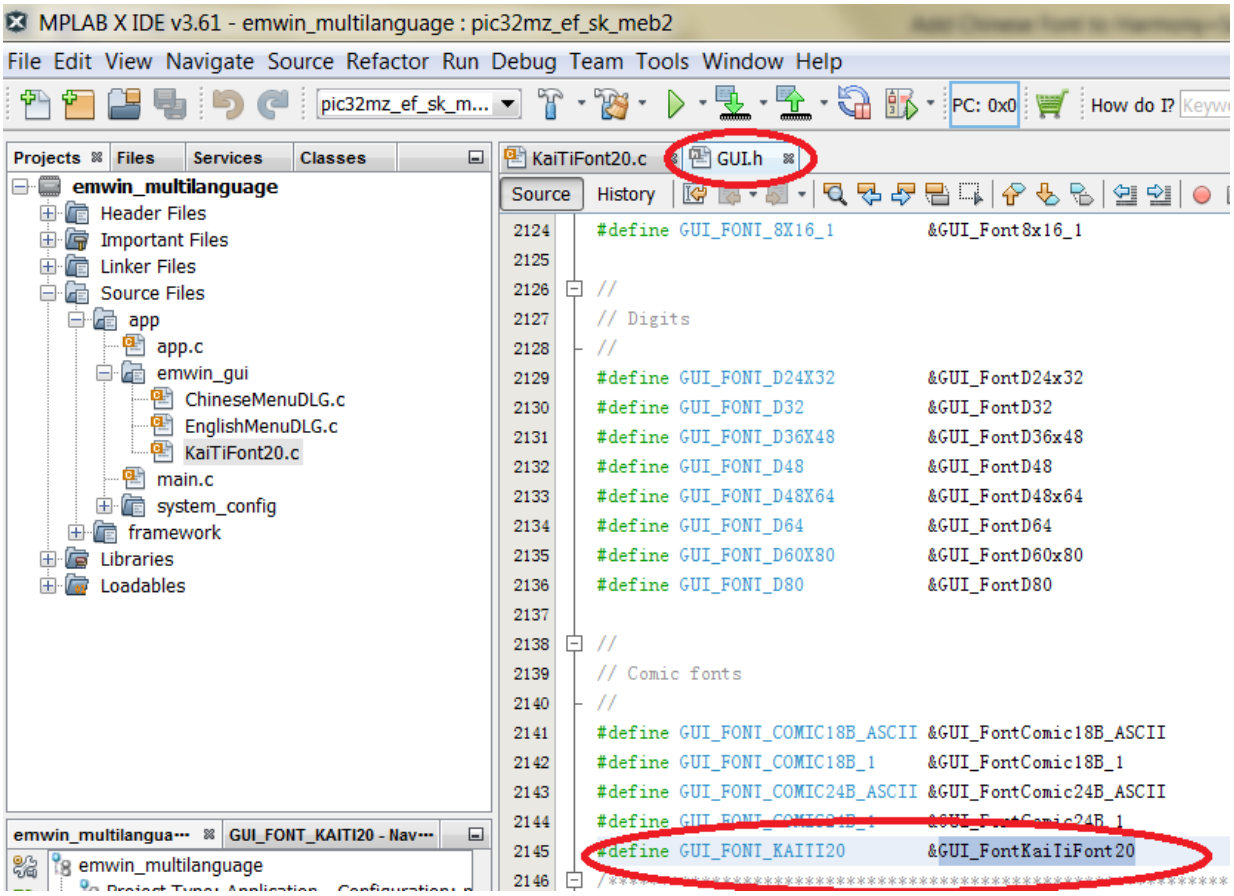




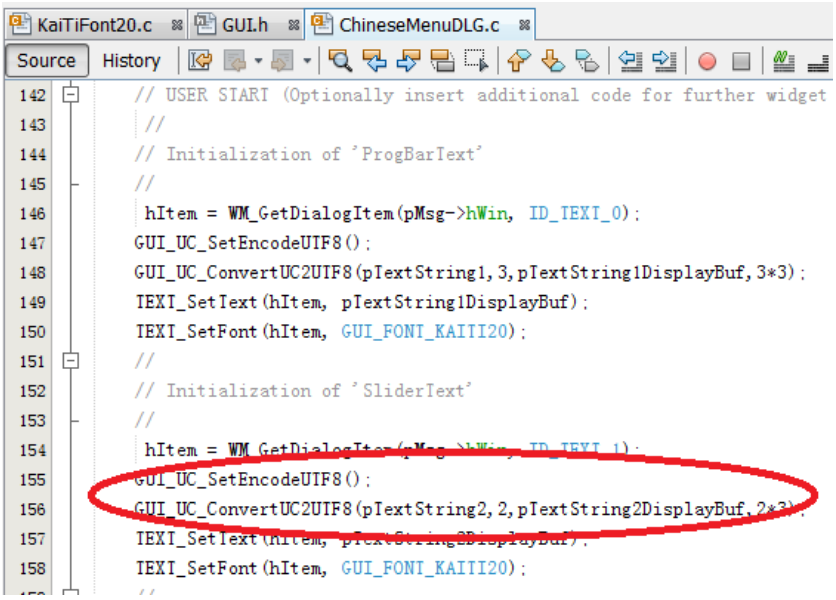
11. Define a new font in `GUI.h`.



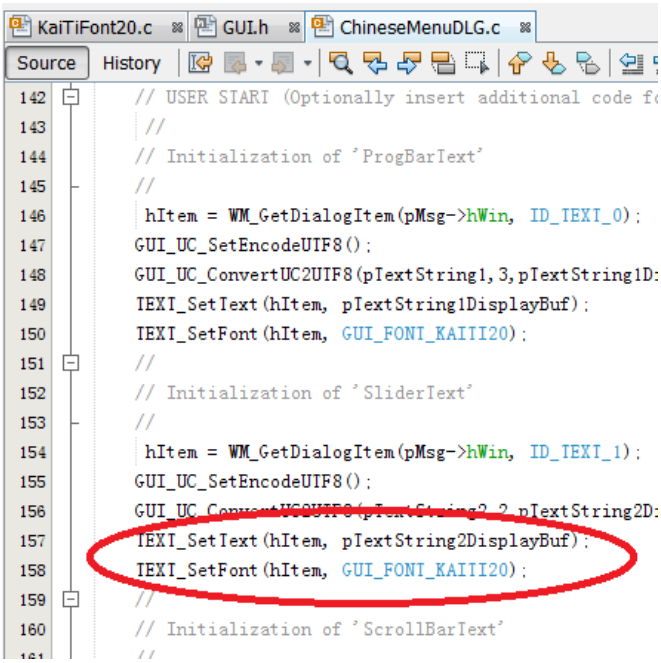
12. Use the new font where a text string is displayed. Define a string buffer to which the Unicode of the text is initialed.



13. Set UTF8 encode and convert the string Unicode to UTF8, one Unicode will be covert to 3 Byte in UTF8.

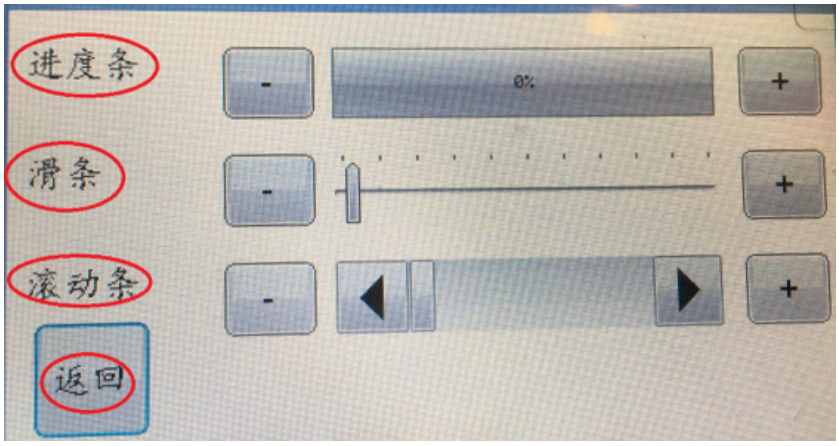


14. Use the new string and new font to display a Chinese string.



Result

Now you should be able to display a Chinese String, as shown in the following figure.



Running the Demonstration

This section provides instructions about how to build and run the emWin Multi-language demonstration.

Description

The demonstration consists of two language screens created using the SEGGER emWin GUIBuilder utility:

- English
- Chinese

Each screen uses different widgets from the SEGGER emWin Graphics Library.

The English language screen uses the following widgets:

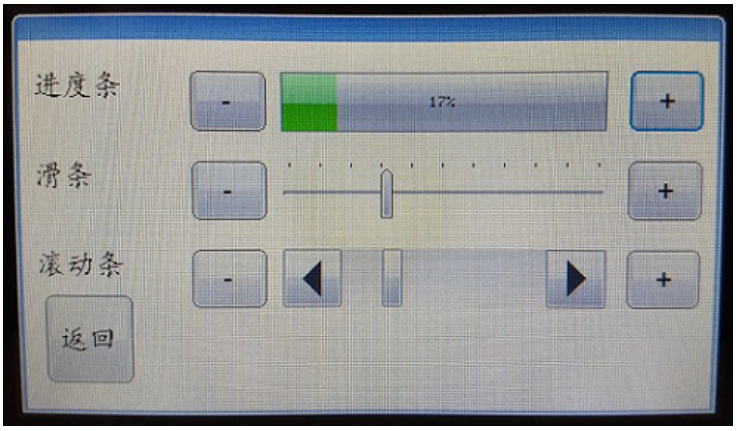
- Framewin
- Image: MPLAB Harmony Logo and SEGGER logo
- Text: emWin Multi-language
- Button: Next

Touch **Next** to navigate from the English language screen to the Chinese language screen.



The Chinese language screen use the following widgets:

- Framewin
- Slider: Single Slider with 10 divisions. The Slider position is updated by touch input.
- Progress Bar : Values updated by the '+' and '-' buttons along the progress bar
- Button: Previous
- Text: Labels for the widgets on screen, as well as the button text
- Scroll Bar: Values updated by the '+' and '-' buttons along the scroll bar



emwin_quickstart

This demonstration is a Graphical User Interface (GUI) application that uses SEGGER emWin and integrates it with MPLAB Harmony on PIC32 development hardware.

Description

This is a simple graphics application with the basic building blocks – a widget, an image and some text, which demonstrates the use of the third-party SEGGER emWin Graphics Library being used inside the MPLAB Harmony framework. The SEGGER emWin Graphics Library is offered free for use on PIC32 devices with the MPLAB Harmony installation. The application has three screens that transition between each other using buttons. Each screen demonstrates some basic features, different widgets, and image rendering available in the SEGGER emWin Graphics Library

A step-by-step guide to how this application was developed is provided in Start-to-End Example of SEGGER emWin Graphics with MPLAB Harmony under SEGGER emWin Graphics Library Help.

Architecture

The application uses the SEGGER emWin Graphics Library to render graphics to the display. SEGGER emWin provides user interface tools like GUIBuilder to create the graphic UI design, which is then imported into the project as C files. This UI design includes widget creation, placement and look and feel; however, it does not include incorporated events, such as touch, etc. supported by the MPLAB Harmony Aria User Interface Library.

The MPLAB Harmony Graphics Library draws the widgets and images to the write frame buffer that is stored in an internal SRAM. Using the DMA, the Display Driver continuously transfers frame data from the read frame buffer out to the LCD display. The write and read frame buffers are swapped when the Graphics Library is done rendering to the write frame buffer and the display driver is done transferring the frame data from the read frame buffer.

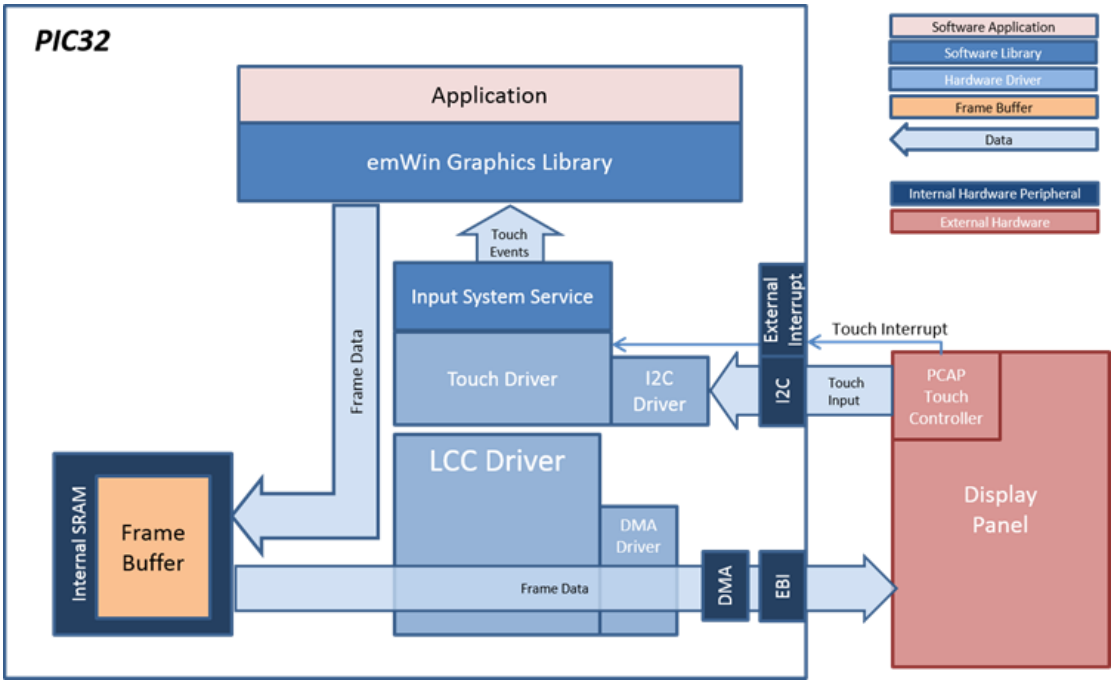
The application also features user touch input through the integrated touch screen on the display panel. Touch input from the touch controller goes through the I2C port, and the Input System Service acquires the touch input information from the Touch and I2C drivers. The Input System Service sends touch events to the Graphics Library, which processes these events and updates the frame data accordingly.

Another important part is the integration of the display controller.

This demonstration has multiple configurations and each requires a suitable display controller/driver strategy for driving the pixel information from the emWin Graphics Library to the display. These configurations require appropriate APIs to be available to the emWin Graphics Library to drive the pixel information to the display. Currently, this is handled by the wrapper layer provided in MPLAB Harmony, which generates the suitable APIs in the `LCDConf.c` file. This file will be generated by the emWin GUI Wrapper and will be added within the project logical folder `system_config\third_party\gfx\emwin\config`. We will discuss this in more detail within the individual configuration.

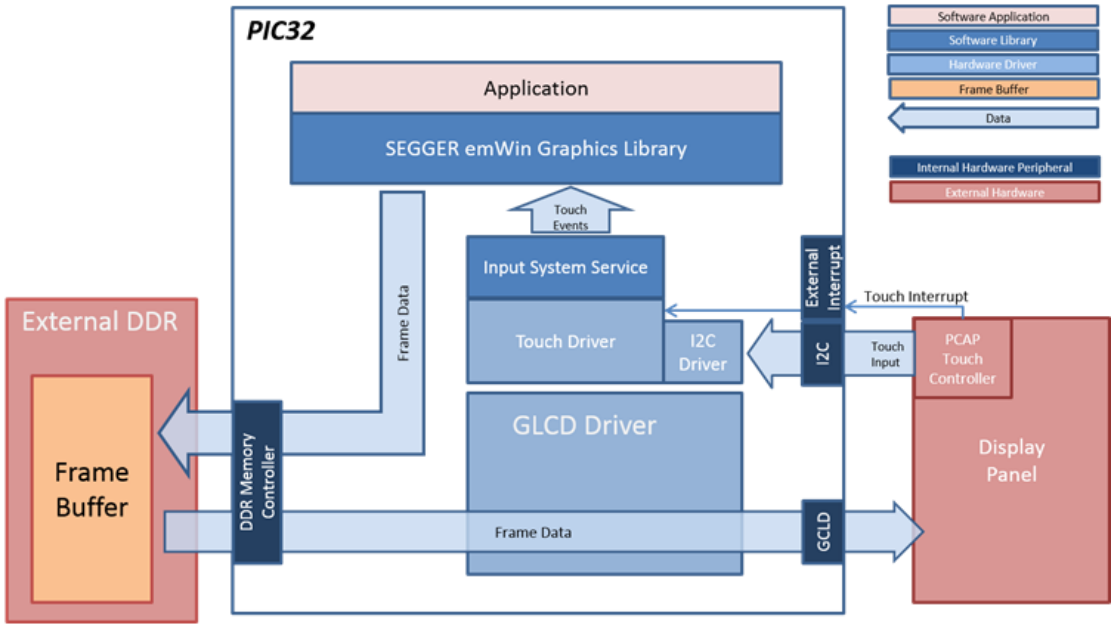
pic32mz_ef_sk_meb2

This configuration uses the Low-Cost Controllerless (LCC) graphics as its display driver. The LCC driver in MPLAB Harmony fetches the framebuffer from internal or external memory and outputs it to the display. In our current configuration, we have the SEGGER emWin Graphics Library configured to only have access to the internal on-chip memory. We need to pass the address of the framebuffer to the emWin Graphics Library. This is handled by the APIs generated by the GUI Wrapper in the `LCDConf.c` file.



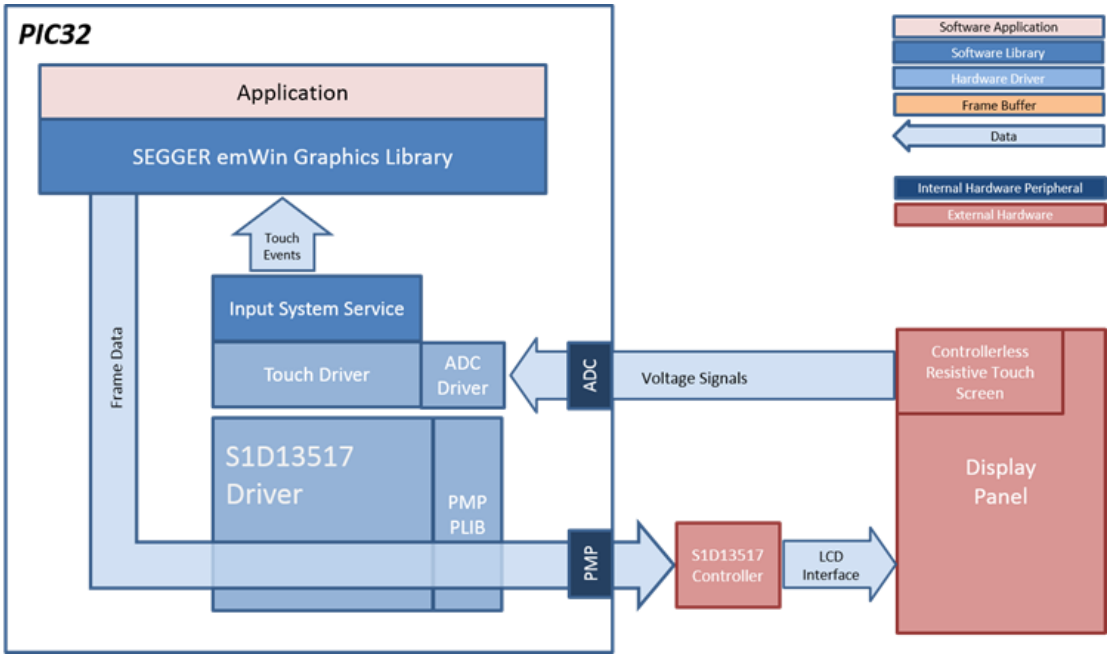
pic32mz_da_sk_extddr_sk_meb2

This configuration uses the GLCD display controller. The GLCD also fetches the framebuffer from the DDR memory and outputs to the display. The SEGGER emWin Graphics Library needs this framebuffer address to write the graphics libraries output to. The appropriate API to enable this is generated by the GUI Wrapper in the `LCDConf.c`, which will be included in the project.



pic32mx_usb_sk2_s1d_pictail_wqvga

This configuration requires an external display controller such as the S1D13517 to drive the display. The memory in which the pixel information is stored for the S1D13517 controller is on the controller itself. This memory is not directly accessible to the emWin Graphics Library for filling out graphics output. Therefore, the S1D13517 Driver provided by the SEGGER emWin Graphics Library must be used and appropriately hook it into the S1D13517 driver that comes with the MPLAB Harmony framework, which actually drives the display. The emWin GUI Wrapper generates the appropriate APIs in the `LCDConf.c` file.



SEGGER emWin and MPLAB Harmony Integration

C File Integration

The SEGGER emWin GUIBuilder utility has generated three C files with one file per screen. All files are added to the project within the logical folder `app\emwin_gui`. To integrate the generated files with MPLAB Harmony, the application uses the emWin GUI Wrapper. On selection of the emWin GUI Wrapper within MHC, the MHC code generation tool will generate the required wrapper library. Similarly, for Touch integration, an emWin Touch Wrapper Library is also available. The appropriate emWin GUI Wrapper and emWin Touch Wrapper APIs need to be called under the application code. Refer to GUI and Touch Wrapper Library for SEGGER emWIN for more information.

Demonstration Features

- Third Party SEGGER emWin Graphics Library
- Input System Service
- Touch Driver
- DMA System Service
- I2C Driver
- 16-bit RGB565 color depth support (65535 unique colors)
- Graphics UI generated using the GUIBuilder tools provided by the SEGGER emWin tool set, which is included with your installation of MPLAB Harmony and is available for use under the license agreement
- Display Controller Integration

Tools Setup Differences

The MPLAB Harmony Graphics Stack still provides the display configuration and the display driver for the project.

The graphics library selected is the third-party SEGGER emWin Graphics Library. Also note the Color format needs to be specified depending on the microcontroller used in the project and the display driver supported.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the emWin Quick Start demonstration.

Description

To build this project, you must open the `emwin_quickstart.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/gfx/emwin_quickstart`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
emwin_quickstart.X	<install-dir>/apps/gfx/emwin_quickstart/firmware

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within ./firmware/src/system_config.

Project Configuration Name	BSP Used	Description
pic32mz_ef_sk_meb2	pic32mz_ef_sk+meb2	SEGGER emWin GUI demonstration for the Multimedia Expansion Board II (MEB II) connected to the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit.
pic32mz_da_extddr_sk_meb2	pic32mz_da_sk_extddr_sk+meb2	SEGGER emWin GUI demonstration for the Multimedia Expansion Board II (MEB II) connected to the PIC32MZ Embedded Graphics with External DRAM (DA) Starter Kit.
pic32mx_usb_sk2_s1d_pictail_wqvga	pic32mx_usb_sk2+s1d_pictail+wqvga	SEGGER emWin GUI demonstration for the Graphics Controller PICtail Plus Epson S1D13517 Daughter Board connected to the PIC32 USB Starter Kit II (with on-board PIC32MX795F512L MCU).

Configuring the Hardware

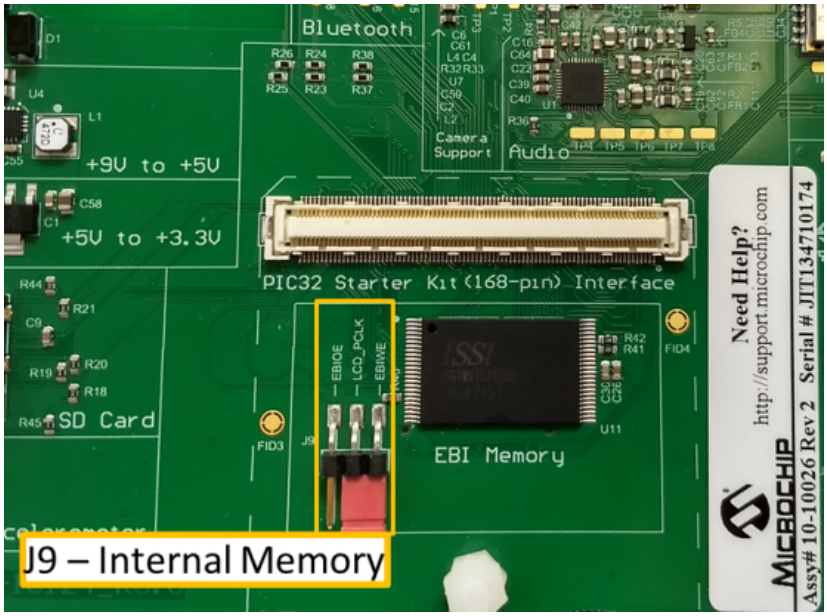
Describes how to configure the supported hardware.

Description

PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit and MEB II

Configuration: pic32mz_ef_sk_meb2

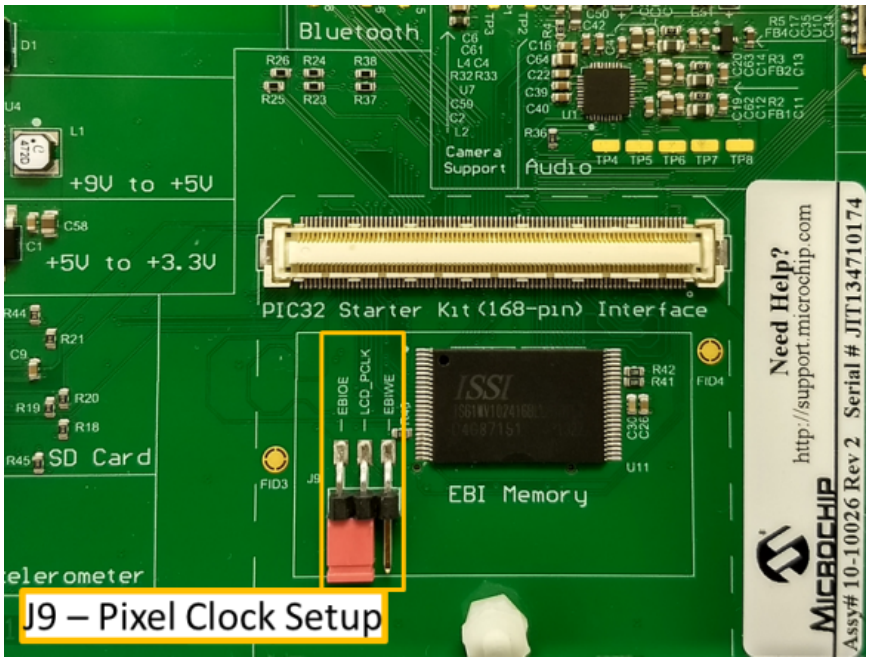
- This configuration requires that the J9 jumper be set to enable internal SRAM for the frame buffer. Set the J9 jumper to connect the EBIOE and LCD_PCLK pins. The J9 jumper is located on the bottom of the MEB II board, beneath where the starter kit is plugged into the board. Refer to the following figure for the exact location.
- Connect the PIC32MZ EF Starter Kit board to the MEB II board
- Power up the board by connecting the power adapter to J3 power connector on the MEB II board or a powered USB cable to the USB DEBUG port on the Starter Kit board



PIC32MZ Embedded Graphics with External DRAM (DA) Starter Kit and MEB II

Configuration: pic32mz_da_sk_extddr_sk+meb2

This configuration requires that the J9 jumper be set to provide the GLCD's pixel clock. Set the J9 jumpber to connect EPIOE to LCD_PCLK. The J9 jumper is located on the bottom of the MEB II board, beneath where the starter kit is plugged into the board. Refer to the following figure for the exact location.



PIC32 USB Starter Kit II with On-board PIC32MX795F512L MCU, and Graphics Controller PICtail Plus Epson S1D13517 Daughter Board

Configuration: pic32mx_usb_sk2+s1d_pictail+wqvga

P2 on the S1D13517 Daughter Board should be closed (16-bit PMP) for the WQVGA Display.

Running the Demonstration

This section provides instructions about how to build and run the emWin Quick Start demonstration.

Description

Demonstration Screens

The demonstration consists of three screens created using the SEGGER emWin GUIBuilder utility:

- Home
- Number Churning
- Text Alignment

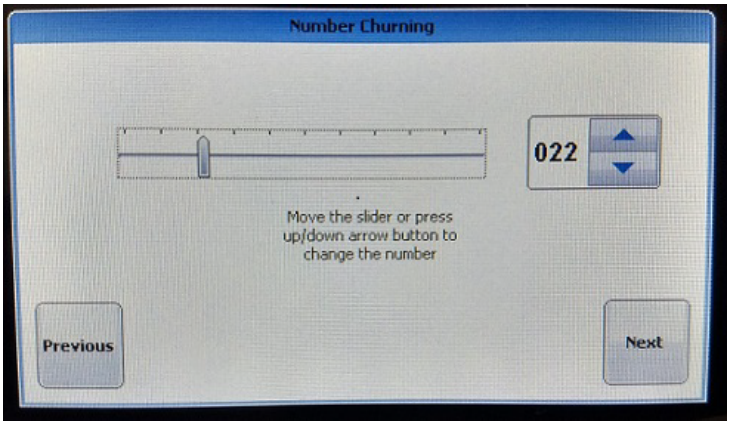
Each screen uses different widgets from the SEGGER emWin Graphics Library. The Home screen uses the following widgets:

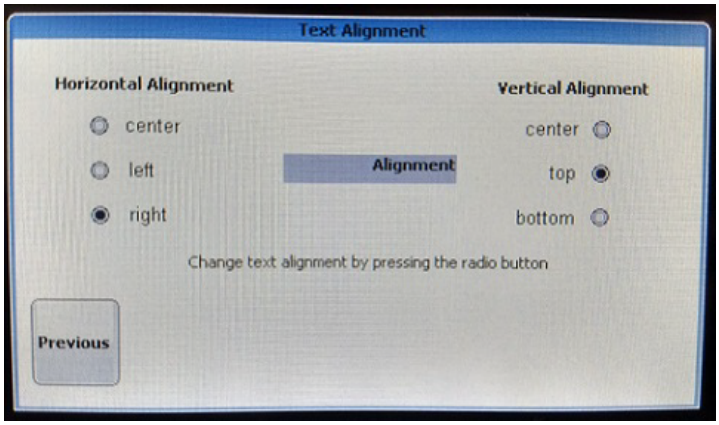
- Framewin: Home Screen
- Image: MPLAB Harmony Logo and SEGGER logo
- Text: Powered by and emWin
- Button: Next

Demonstration Process

1. Touch the **Next** button to navigate from the Home screen to the Number Churning screen. The Number Churning screen uses the following widgets:
   - Framewin: Number Churning
   - Slider: Single Slider with 10 divisions. The Slider position is updated by both touch input and spinbox.
   - Spinbox : Three-digit spinbox with value updated by both arrow buttons and slider
   - Button: Previous and Next
   - Text: Move the slider or press up/down arrow button to change the number
2. Touch the **Next** button to navigate from Number Churning Screen to Text Alignment screen. Similarly, press the **Previous** button to navigate from Number Churning Screen back to the Home screen. The Text Alignment screen consists of the following widgets:
   - Framewin: Text Alignment
   - Radio: Six radio buttons with each radio button affecting the alignment of the Alignment text
   - Text: Horizontal Alignment, Vertical Alignment, center, left, top, right, bottom, and Change text alignment by pressing the radio button
3. Touch the **Previous** button to navigate from Text Alignment screen back to the Number Churning screen.







emwin_showcase

This demonstration shows basic and advanced capabilities of the SEGGER emWin Graphics Library utilizing the software graphics controller on an LCD display.

Description

The emwin_showcase demonstration provides the ability to display some of the many features supported by the SEGGER emWin Graphics Library as a simple Graphical User Interface (GUI) demonstration.

SEGGER emWin is designed to provide an efficient, processor, and LCD controller-independent GUI for any application that operates with a graphical LCD. Some of the features demonstrated in the application include GUIs that shows alpha blending, sprites, radial menu operations, listing and tree widget features, multiple layer and drawing on the layers, use of different image formats, coloring features provided by the library, and so on.

Currently, the demonstration includes the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit on a Multimedia Expansion Board II (MEB II), which has controllerless graphics on a WQVGA LCD display.

This demonstration requires the use of internal memory (SRAM). The memory setting must be configured with a hardware jumper and corresponding MHC setting must be selected.

Important: To set up the internal memory, a jumper setting on the board is required. Failure to configure this jumper setting will prevent the display from working, though the software may still run. See Configuring the Hardware for details about the appropriate jumper settings.

Architecture

The application uses the third-party SEGGER emWin Graphics Library to render graphics to the display. It essentially showcases different emWin capabilities. The Graphics Library draws the widgets and images to the frame buffer that is stored in an internal SRAM. Using the DMA, the Display Driver continuously transfers frame data from the frame buffer out to the LCD display.

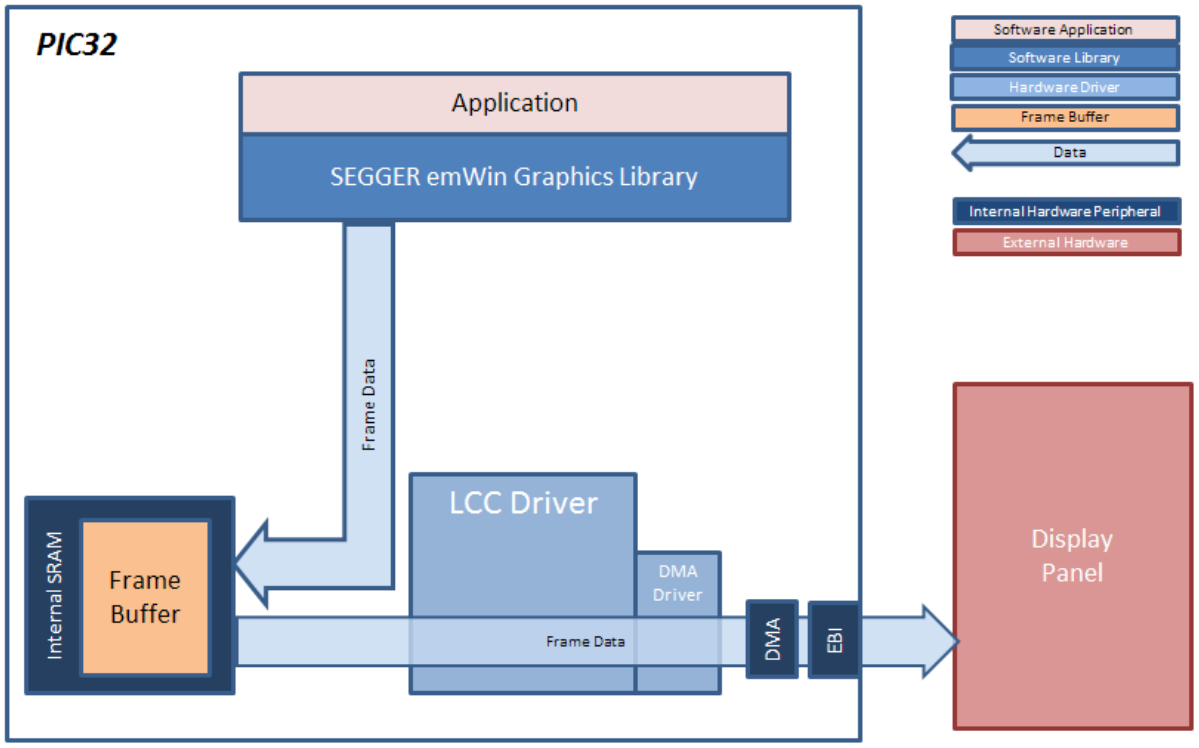
The application does not have touch support.

The Display controller and the LCD display driver are selected within the Graphics Stack provided by MPLAB Harmony.

pic32mz_ef_sk_meb2

This configuration uses the Low Cost Controllerless (LCC) graphics as its display driver.

The LCC driver in MPLAB Harmony fetches the framebuffer from internal or external memory and outputs it to the display. In our current configuration, we have the SEGGER emWin Graphics Library configured to only have access to the internal on-chip memory. We need to pass the address of the framebuffer to the SEGGGER emWin Graphics Library. This is handled by the APIs generated by the GUI Wrapper in the `LCDConf.c` file.



Demonstration Features

- Third-Party SEGGER emWin Graphics Library
- DMA System Service
- 16-bit RGB565 color depth support (65535 unique colors)
- Graphics UI generated using the GUIBuilder tools provided by the SEGGER emWin tool set, which are provided with MPLAB Harmony and are available for use under the license agreement.
- Low-Cost Controllerless Graphics

Tools Setup Differences

- “Use Graphics Stack” is selected in MHC. This enables the graphics Low-Cost Controllerless Graphics and Select the NewHaven 4.3” display.
- The graphics library selected is the third-party SEGGER emWin Graphics Library

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the SEGGER emWin MEB II Demonstration.

Description

To build this project, you must open the `emwin_showcase.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/gfx/emwin_showcase`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`emwin_showcase.X`	`<install-dir>/apps/gfx/emwin_showcase/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mz_ef_sk_meb2	pic32mz_ef_sk+meb2	Select this MPLAB X IDE project configuration to run the demonstration on the Multimedia Expansion Board II (MEB II) connected to the PIC32MZ EF Starter Kit.

Configuring the Hardware

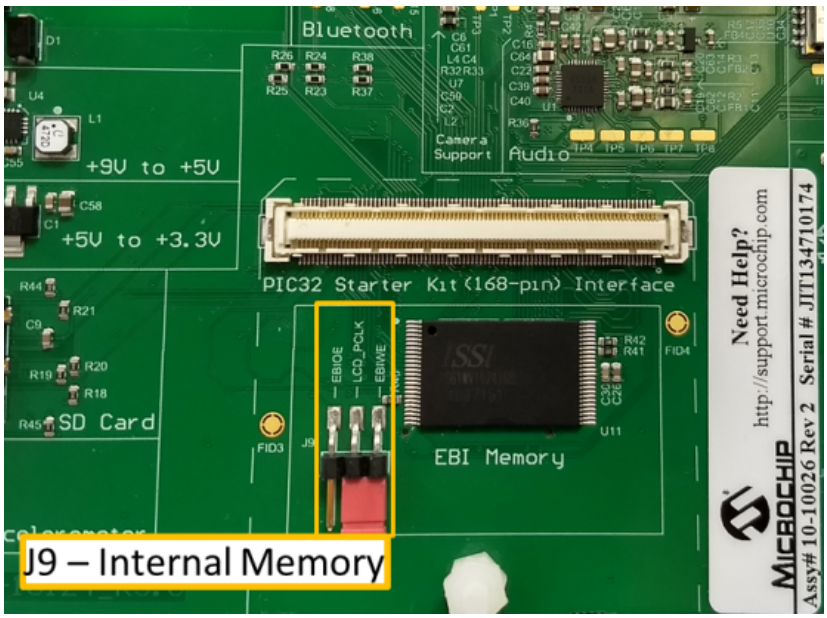
This section describes how to configure the supported hardware.

Description

PIC32MZ EF Starter Kit and MEB II

Configuration: pic32mz_ef_sk_meb2

- This configuration requires that the J9 jumper be set to enable internal SRAM for the frame buffer. Set the J9 jumper to connect the EBIOE and LCD_PCLK pins. The J9 jumper is located on the bottom of the MEB II board, beneath where the starter kit is plugged into the board. Refer to the following figure for the exact location.
- Connect the PIC32MZ EF Starter Kit board to the MEB II board
- Power up the board by connecting the power adapter to J3 power connector on the MEB II board or a powered USB cable to the USB DEBUG port on the Starter Kit board



Running the Demonstration

This section provides instructions on how to build and run the emWin Showcase demonstration.

Description

This demonstration shows the Graphics Library interfacing with the Low-Cost Controllerless (LCC) software display controller. The demonstration displays some of the many features supported by the SEGGER emWin graphics library as a simple GUI demonstration.

Use the following procedure to run the demonstration:

1. Load the demonstration project into MPLAB X IDE.
2. Build, Download, and Run the demonstration project on the target board.

Note: This is a GUI only demonstration; there is no touch input being processed.



Demonstration Screens

The following are the different screens that demonstrate the various graphics features:

- **Radial menu** - Select an icon from a radial menu. Changing the selection is done using emWin motion support.
- **Bargraph demo** - Shows a bar graph using alpha blending.
- **Antialiased text** - Shows anti-aliased text with different anti-aliasing qualities. Outputs anti-aliased text on different backgrounds (2 bpp, 4 bpp).
- **Transparent dialog** – Uses alpha blending for transparency effect on moving background.
- **Washing machine** - Shows a washing machine demonstration with blue dolphin sprites moving on top of the application.
- **Iconview demo** - Uses the ICONVIEW widget for showing an icon-based menu, which is often required in hand held devices. Shows the change of selection and change of icon text alignment.
- **Treeview widget** - Shows a customized TREEVIEW widget. Demonstrates hierarchical view of items in a directory and some sprites, show/hide lines, moving cursor, open/close nodes, change selection modes, setting images, show/hide lines.
- **Listview widget** - Shows the use of a LISTVIEW widget. Demonstrates changing order, enable sorting, using reverse/normal sorting order, moving rows, coloring row/column individual elements in the list.
- **Drawing a graph** - Uses the GRAPH widget to visualize a function graph (e.g., heartbeat, sine waves).
- **High speed** – Demonstrates multi-layer clipping and highly optimized drivers.
- **Pixels speed** – Demonstrates pixel speed as pixels per seconds.
- **Bitmaps** - Demonstrates `*.BMP` files by displaying all bitmaps of the Windows directory. Palette-based bitmaps, changing the pallete, bmp, gif, jpeg, 12, 16, 24 bpp formats, alpha bitmaps, changing color, grayscale bitmaps.
- **Color bar** - Shows a color bar with gradient bars (Black > Color, White > Color). Integrated color management, which finds the optimized color for any logical color.

Motor Control Demonstrations

This section provides descriptions of the Motor Control demonstrations.

MPLAB Harmony is available for download from the Microchip website by visiting: http://www.microchip.com/mplabharmony. Once you are on the site, click the Downloads tab to access the appropriate download for your operating system. For additional information on this demonstration, refer to the “Applications Help” section in the MPLAB Harmony Help.

Introduction

Motor Control Demonstration Applications Help.

Description

This installation of MPLAB Harmony consists of two broad categories of motor control demonstrations.

Stand-alone Sensorless Field Oriented Control of PMSM Motor Using a PLL-based Estimator

This demonstration is designed to work with PIC32MK 100-pin Motor Control PIM mounted on the dsPICDEM™ MCLV-2 Development Board for Small Hurst Motor (DMB0224C10002), and has two variants:

- Dual Shunt PLL Estimator-based FOC demonstration on PIC32MK using an external Op amp configuration
- Dual Shunt PLL Estimator-based FOC demonstration on PIC32MK using an internal Op amp configuration

Please refer to the Microchip application note, AN2520 *Sensorless Field Oriented Control (FOC) for a Permanent Magnet Synchronous Motor (PMSM) Using a PLL Estimator and Flux Weakening (FW)* for more details

Integrated Application of Average Current Mode Power Factor Correction and Sensorless Field Oriented Control of a PMSM Motor

This demonstration is designed to work with PIC32MK 100-pin Motor Control PIM mounted on the dsPICDEM™ MCHV-3 Development Board (High Voltage), and has two variants:

- Integrated PFC and FOC demonstration on PIC32MK using an external Op amp configuration
- Integrated PFC and FOC demonstration on PIC32MK using an internal Op amp configuration

Please refer to the Microchip application note, *Integrated Power Factor Correction (PFC) and Sensorless Field Oriented Control (FOC) for Microchip 32-bit Microcontrollers* for more details.

To learn more about MPLAB Harmony stacks and libraries refer to the related documentation in Volume V: MPLAB Harmony Framework Reference.

Demonstrations

This topic provides information on how to run the Motor Control demonstration applications included in this release.

dualshunt_pll_foc_mclv2_ext_opamp

This section provides information on the supported demonstration boards, how to configure the hardware, and how to run the demonstration.

Description

This demonstration implements Sensorless Field Oriented Control (FOC) of a PMSM motor using a dual shunt configuration, and utilizes external (off-chip) Op amps for current sensing.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the demonstration.

Description

To build this project, you must open the `dualshunt_pll_foc_mclv2_ext_opamp.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/motor_control/dualshunt_pll_foc_mclv2_ext_opamp`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`dualshunt_pll_foc_mclv2_ext_opamp.X`	`<install-dir>/apps/motor_control/dualshunt_pll_foc_mclv2_ext_opamp/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are within `./firmware/src/system_config`.

Project Configuration Name	BSP(s) Used	Description
small_hurst_motor_mclv2_ext_opamp	Not applicable.	Demonstrates dual shunt PLL estimator-based FOC of a Small Hurst Motor (DMB0224C10002) using external op-amps.

Configuring the Hardware

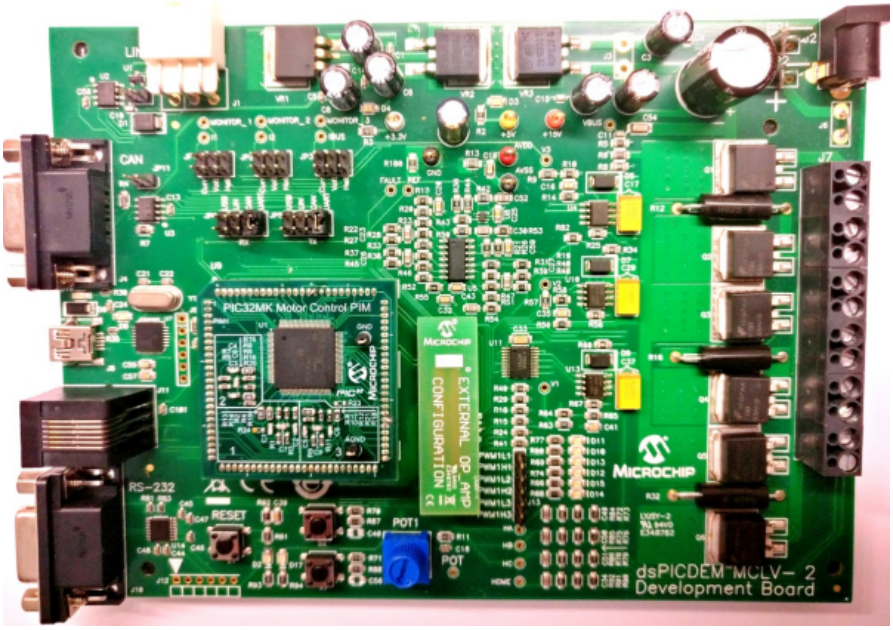
Describes how to configure the supported hardware.

Description

PIC32MK 100-pin Motor Control Plug-in Module (PIM) with the dsPICDEM MCLV-2 Development Board

1. Mount the PIC32MK 100-pin Motor Control PIM on the dsPICDEM MCLV-2 Development Board, as shown in the following figure.
2. Connect the External Op amp Configuration Matrix Board to header J14.
3. Connect the three-phase wires coming from P1 header of the small Hurst motor (i.e., Red, White and Black to M1, M2 and M3 ports, respectively). Leave the green wire from the motor unconnected.
4. Connect the programmer/debugger using the J11 connector.
5. Apply 24V DC at BP1-BP2/J2.
6. This application supports communication with the X2C-Scope plug-in to monitor or plot any variables used in the application. The communication between the target and the X2C-Scope plug-in can be established using the RS-232 serial port (J10) or mini-USB port (J8). To use the RS-232 serial port, connect JP4 and JP5 to the UART position. To use the mini USB port, connect JP4 and JP5 to the USB position. Refer to Using the X2C Scope for details on how to use X2C Scope in this demonstration.

PIC32MK MC PIM with dsPICDEM™ MCLV-2 using the External OPAMP Configuration



Running the Demonstration

Provides instructions on how to build and run the demonstration.

Description

Do the following to run the demonstration:

1. Compile and build the project.
2. Program the target device.
3. Press the ‘S2’ switch to start spinning the motor.
4. Vary the potentiometer, P1, to change the motor speed.
5. Press the 'S1’ switch to stop the motor.

Modifying the System Parameters

The system parameters, such as motor coefficients, maximum current, etc., can be modified in the header file `mc_app.h`, as shown in the

following code example.

```
//============================================================
// Following parameters for MCLV-2 board
// Gain of op amp = 15
// Shunt resistor = 0.025 ohms
// DC offset = 1.65V
// Max current = x
// (x * 0.025 * 15) + 1.65V = 3.3V
// x = 4.4Amps
#define     MAX_BOARD_CURRENT                (float)(4.4)
#define     MAX_MOTOR_CURRENT                (float)(4.2)
#define     MAX_MOTOR_CURRENT_SQUARED        (float)((float)MAX_MOTOR_CURRENT*
                                             (float)MAX_MOTOR_CURRENT)
#define     VREF_DAC_VALUE                   (int) 2048
#define     ADC_CURRENT_SCALE                (float)(MAX_BOARD_CURRENT/(float)2048)
#define     CURRENT_LIMIT_CMP_REF            (int)(((float)2048*
                                             (MAX_MOTOR_CURRENT/MAX_BOARD_CURRENT))
                                             +VREF_DAC_VALUE)
#define     MOTOR_PER_PHASE_RESISTANCE       ((float)2.10)     // Resistance in Ohms
#define     MOTOR_PER_PHASE_INDUCTANCE       ((float)0.00192)  // Inductance in Henrys
#define     MOTOR_BACK_EMF_CONSTANT_Vpeak_Line_Line_KRPM_MECH   (float)7.24
                                             // Back EMF Constant in Vpeak/KRPM
#define     NOPOLESPAIRS                     5
#define     MAX_ADC_COUNT                    (float)4095     // for 12-bit ADC
#define     MAX_ADC_INPUT_VOLTAGE            (float)3.3      // volts
```

Application Parameters

Parameter Name	Description	Units
PWM_FREQ	PWM frequency	Hz
DEADTIME_SEC	Dead time	Seconds
MAX_BOARD_CURRENT	Maximum inverter DC bus current through the board that can be measured without saturating the ADC	A
MAX_MOTOR_CURRENT	Maximum motor phase current	A
MOTOR_PER_PHASE_RESISTANCE	Motor per phase resistance	O
MOTOR_PER_PHASE_INDUCTANCE	Motor per phase inductance	H
MOTOR_BACK_EMF_CONSTANT_Vpeak_Line_Line_KRPM_MECH	Motor Back EMF constant	Vpeak(line-line)/KRPM
NOPOLESPAIRS	Number of motor pole pairs	-
MAX_ADC_COUNT	Full scale ADC count, 2n bit ADC --1 (For 12 bit ADC, 212-1= 4095	-
MAX_ADC_INPUT_VOLTAGE	ADC reference voltage	V
DCBUS_SENSE_TOP_RESISTOR	High-side resistance of DC BUS Sense voltage divider	O
DCBUS_SENSE_BOTTOM_RESISTOR	Low-side resistance of DC BUS Sense voltage divider	O
LOCK_TIME_IN_SEC	Rotor lock time to a forced rotor angle before spinning the motor	Seconds
END_SPEED_RPM	Motor speed in Open Loop mode at which the algorithm switches to closed loop	RPM
RAMP_TIME_IN_SEC	Time to reach open loop end speed (END_SPEED_RPM) during open loop operation	Seconds
Q_CURRENT_REF_OPENLOOP	Q-axis current during open loop operation	A
NOMINAL_SPEED_RPM	Maximum rated speed in constant torque mode (without field weakening)	RPM
MOVING_AVG_WINDOW_SIZE	Total number of current samples used to calculate moving average of the current to obtain ADC current offset = 2MOVING_AVG_WINDOW_SIZE	-
CURRENT_OFFSET_MAX	Maximum limit of the ADC current offset	-

CURRENT_OFFET_MIN	Minimum limit of the ADC current offset	-
CURRENT_OFFSET_INIT	Initial value of the ADC current offset	-
KFILTER_ESDQ	First order low-pass filter coefficient for Ed, Eq estimator parameters	-
KFILTER_VELESTIM	First order low-pass filter coefficient for speed estimation	-
FW_SPEED_RPM	Maximum rated speed in Field Weakening mode	RPM
MAX_FW_NEGATIVE_ID_REF	Maximum applicable negative D-axis current	A
D_CURRCNTR_PTERM	D-axis current controller proportional gain	-
D_CURRCNTR_ITERM	D-axis current controller integral gain	-
D_CURRCNTR_CTERM	D-axis current controller anti-windup gain	-
D_CURRCNTR_OUTMAX	D-axis current controller saturation limit	-
Q_CURRCNTR_PTERM	Q-axis current controller proportional gain	-
Q_CURRCNTR_ITERM	Q-axis current controller integral gain	-
Q_CURRCNTR_CTERM	Q-axis current controller anti-windup gain	-
Q_CURRCNTR_OUTMAX	Q-axis current controller saturation limit	-
SPEEDCNTR_PTERM	Speed controller proportional gain	-
SPEEDCNTR_ITERM	Speed controller integral gain	-
SPEEDCNTR_CTERM	Speed current controller anti-windup gain	-
SPEEDCNTR_OUTMAX	Speed current controller saturation limit	-

Class B Tests

This demonstration runs the following Class B tests during startup:

- Clock Test
- Program Counter Test
- CPU Registers Test
- Flash Test.
- Checkerboard RAM Test
- March B Test
- March C Test

All Class B memory tests in this demonstration are non-destructive.

Faults

In this demonstration, the ON state of LED D2 indicates a Fault detection. This demonstration is equipped to detect the Class B test fail and overcurrent faults. Class B tests are run immediately after the device power-up. If any of the Class B tests fail, it will be indicated by LED D2 turning ON. Also, failure of Class B tests would inhibit the start of the motor.

Once the Class B tests have passed successfully, the motor can be started by pressing the switch ‘S2’. While the motor is spinning, if there is an overcurrent condition detected, this will cause the MCPWM to shutdown and LED D2 will turn ON indicating a Fault.

dualshunt_pll_foc_mclv2_int_opamp

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

This demonstration implements Sensorless Field Oriented Control (FOC) of a PMSM motor using a dual shunt configuration, and utilizes internal (on-chip) Op amps for current sensing.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the demonstration.

Description

To build this project, you must open the `dualshunt_pll_foc_mclv2_int_opamp.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/motor_control/dualshunt_pll_foc_mclv2_int_opamp`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`dualshunt_pll_foc_mclv2_int_opamp.X`	`<install-dir>/apps/motor_control/dualshunt_pll_foc_mclv2_int_opamp/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are within `./firmware/src/system_config`.

Project Configuration Name	BSP(s) Used	Description
small_hurst_motor_mclv2_int_opamp	Not applicable.	Demonstrates dual shunt PLL estimator-based FOC of a Small Hurst Motor (DMB0224C10002) using internal op-amps.

Configuring the Hardware

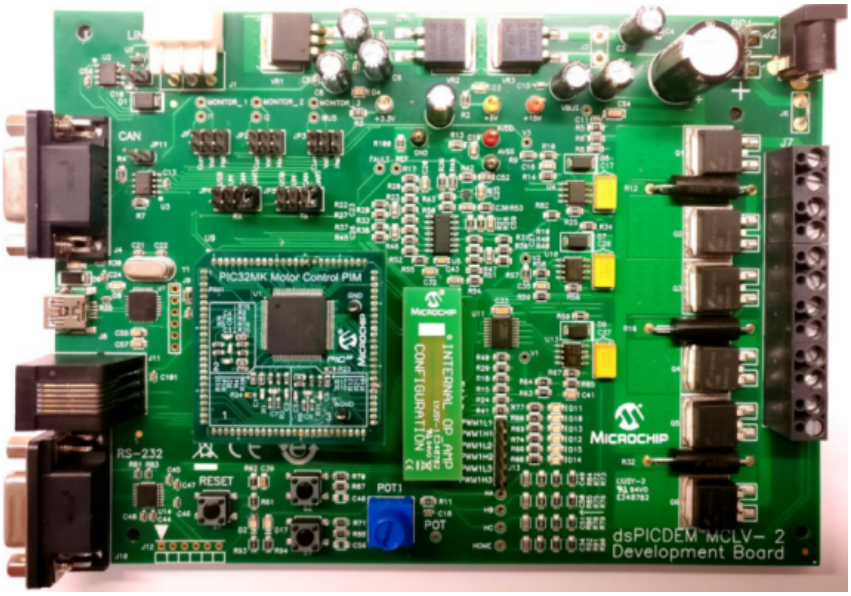
Describes how to configure the supported hardware.

Description

PIC32MK 100-pin Motor Control Plug-in Module (PIM) with the dsPICDEM MCLV-2 Development Board

1. Mount the PIC32MK 100-pin Motor Control PIM on the dsPICDEM MCLV-2 Development Board, as shown in the following figure.
2. Connect the Internal Op amp Configuration Matrix Board to header J14.
3. Connect the three-phase wires coming from P1 header of the small Hurst motor (i.e., Red, White, and Black to M1, M2, and M3 ports, respectively). Leave the green wire from the motor unconnected.
4. Connect the programmer/debugger using the J11 connector.
5. Apply 24V DC at BP1-BP2/J2.
6. This application supports communication with DMCI to monitor/plot any variables used in the application. The communication between the target and the DMCI plug-in can be established using the RS-232 serial port (J10) or mini-USB port (J8). If using the RS-232 serial port, connect JP4 and JP5 to the UART position. If using the mini-USB port, connect JP4 and JP5 to the USB position.

PIC32MK MC PIM with dsPICDEM™ MCLV-2 the Internal Op amp Configuration



Running the Demonstration

Provides instructions on how to build and run the demonstration.

Description

For instructions, refer to the Running the Demonstration section for the dualshunt_pll_foc_mclv2_ext_opamp demonstration.

integrated_pfc_foc_mhcv3_ext_opamp

This section provides information on the supported demonstration boards, how to configure the hardware, and how to run the demonstration.

Description

This demonstration implements integrated application of average current mode PFC and Sensorless Field Oriented Control (FOC) of a PMSM motor using a dual shunt configuration, and utilizes external (off-chip) Op amps for current sensing.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the demonstration.

Description

To build this project, you must open the `integrated_pfc_foc_mchv3_ext_opamp.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is:

`<install-dir>/apps/motor_control/integrated_pfc_foc_mchv3_ext_opamp`.

MPLAB X IDE Project

The following table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`integrated_pfc_foc_mchv3_ext_opamp.X`	`<install-dir>/apps/motor_control/integrated_pfc_foc_mchv3_ext_opamp/firmware`

MPLAB X IDE Project Configurations

The following table lists and describes the supported configurations of the demonstration, which are within `./firmware/src/system_config`.

Project Configuration Name	BSP(s) Used	Description
motor_80_mchv3_ext_opamp	N/A	Demonstrates integrated application of average current mode PFC and sensorless FOC of PMSM motor using external op-amps.

Configuring the Hardware

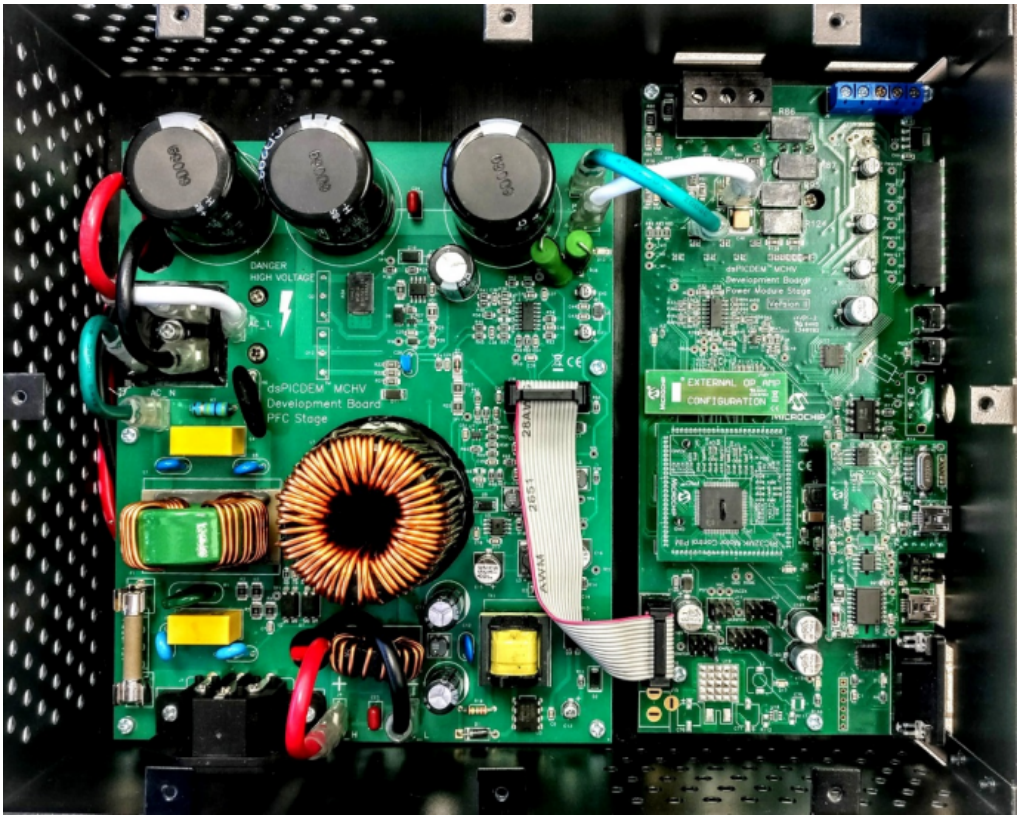
This section describes how to configure the supported hardware.

Description

PIC32MK 100-pin Motor Control PIM with the dsPICDEM MCHV-3 Development Board

1. Mount the PIC32MK 100-pin Motor Control PIM on the dsPICDEM MCHV-3 Development Board, as shown in the following figure.
2. Connect the External Op amp Configuration Matrix Board to header J4.
3. Connect the three-phase wires of the PMSM motor to M1, M2 and M3 ports.
4. Connect the on board programmer and debugger to J20 using USB Mini B cable.
5. Connect the AC mains using J1.
6. This application supports communication with the X2C-Scope plug-in to monitor and plot any variables used in the application. The communication between the target and the X2C-Scope plug-in can be established using the RS-232 serial port (J8) or mini-USB port (J6). Refer to the jumper positions marked on the development board chassis to select between the RS-232 serial port (J8) and the mini-USB port (J6). Refer to Using the X2C-Scope Plug-in for details on how to use the X2C-Scope in this demonstration.

Note:	Please refer to dsPICDEM MCHV-3 Development Board User's Guide for safety and operating details of the development board.



Running the Demonstration

This section provides instructions on how to build and run the demonstration.

Description

Do the following to run the demonstration:

1. Compile and build the project.
2. Program the target device.
3. Press the ‘S1’ switch to start the PFC. Once the bus voltage has reached the set point, the motor will begin spinning.
4. Vary the potentiometer, POT1, to change the motor speed.
5. Press the ‘Reset’ switch to stop the motor.

Modifying the System Parameters

The system parameters, such as motor coefficients, maximum current, etc., can be modified in the header file `mc_app.h`, as shown in the following code example

```
//=============================================================
// Following parameters for MCHV-3 board (External Opamp Configuration)
// Gain of opamp = 10.06
// shunt resistor = 0.01 ohms
// DC offset = 1.65V
// max current = x
// (x * 0.01 * 10.06) + 1.65V = 3.3V
// x = 16.4 Amps
#define     MAX_BOARD_CURRENT                                   (float)(16.4)
#define     MAX_MOTOR_CURRENT                                   (float)(8)
#define     MAX_MOTOR_CURRENT_SQUARED
(float)((float)MAX_MOTOR_CURRENT*(float)MAX_MOTOR_CURRENT)

#define     ADC_CURRENT_SCALE                                   (float)(MAX_BOARD_CURRENT/(float)2048)

#define     CURRENT_LIMIT_CMP_REF
(int)(((float)2048*(MAX_MOTOR_CURRENT/MAX_BOARD_CURRENT))+VREF_DAC_VALUE)

// Resistance in Ohms
#define     MOTOR_PER_PHASE_RESISTANCE                          ((float)16)
```

```
// Inductance in Henrys
#define     MOTOR_PER_PHASE_INDUCTANCE                          ((float)0.022)

#define     MOTOR_PER_PHASE_INDUCTANCE_DIV_2_PI
((float)(MOTOR_PER_PHASE_INDUCTANCE/(2*M_PI)))

// Back EMF Constant in Vpeak/KRPM
#define     MOTOR_BACK_EMF_CONSTANT_Vpeak_Line_Line_KRPM_MECH   (float)75

#define     NOPOLESPAIRS                                        2
#define     MAX_ADC_COUNT                                       (float)4095     // for 12-bit ADC
#define     MAX_ADC_INPUT_VOLTAGE                       (float)3.3      // volts
```

Application Parameters

Parameter Name	Description	Units
PWM_FREQ	PWM Frequency	Hz
DEADTIME_SEC	Dead Time	Seconds
MAX_BOARD_CURRENT	Maximum Inverter DC bus current through the board that can be measured without saturating the ADC	A
MAX_MOTOR_CURRENT	Maximum Motor Phase Current	A
MOTOR_PER_PHASE_RESISTANCE	Motor per phase resistance	O
MOTOR_PER_PHASE_INDUCTANCE	Motor per phase inductance	H
MOTOR_BACK_EMF_CONSTANT_Vpeak_Line_Line_KRPM_MECH	Motor Back EMF constant	Vpeak(line-line)/KRPM
NOPOLESPAIRS	Number of Motor Pole Pairs	-
MAX_ADC_COUNT	Full Scale ADC Count, 2n bit ADC -1 (For 12 bit ADC, 212-1= 4095	-
MAX_ADC_INPUT_VOLTAGE	ADC Reference voltage	V
DCBUS_SENSE_TOP_RESISTOR	High side resistance of DC BUS Sense voltage divider	O
DCBUS_SENSE_BOTTOM_RESISTOR	Low side resistance of DC BUS Sense voltage divider	O
LOCK_TIME_IN_SEC	Rotor lock time to a forced rotor angle before spinning the motor	Seconds
END_SPEED_RPM	Motor speed in Open Loop mode at which the algorithm switches to closed loop	RPM
RAMP_TIME_IN_SEC	Time to reach Open Loop End Speed (END_SPEED_RPM) during open loop operation	Seconds
Q_CURRENT_REF_OPENLOOP	Q axis current during Open Loop operation	A
NOMINAL_SPEED_RPM	Maximum rated speed in Constant Torque mode (without field weakening)	RPM
MOVING_AVG_WINDOW_SIZE	Total number of current samples used to calculate moving average of the current to obtain ADC Current Offset = 2MOVING_AVG_WINDOW_SIZE	-
CURRENT_OFFSET_MAX	Maximum limit of the ADC Current Offset	-
CURRENT_OFFSET_MIN	Minimum limit of the ADC Current Offset	-
CURRENT_OFFSET_INIT	Initial value of the ADC Current Offset	-
KFILTER_ESDQ	First order low pass filter coefficient for Ed, Eq estimator parameters	-
KFILTER_VELESTIM	First order low pass filter coefficient for speed estimation	-
FW_SPEED_RPM	Maximum rated speed in Field Weakening mode	RPM
MAX_FW_NEGATIVE_ID_REF	Maximum applicable negative D-axis current	A

D_CURRCNTR_PTERM	D-axis Current Controller Proportional Gain	-
D_CURRCNTR_ITERM	D-axis Current Controller Integral Gain	-
D_CURRCNTR_CTERM	D-axis Current Controller Anti-Windup Gain	-
D_CURRCNTR_OUTMAX	D-axis Current Controller Saturation Limit	-
Q_CURRCNTR_PTERM	Q-axis Current Controller Proportional Gain	-
Q_CURRCNTR_ITERM	Q-axis Current Controller Integral Gain	-
Q_CURRCNTR_CTERM	Q-axis Current Controller Anti-Windup Gain	-
Q_CURRCNTR_OUTMAX	Q-axis Current Controller Saturation Limit	-
SPEEDCNTR_PTERM	Speed Controller Proportional Gain	-
SPEEDCNTR_ITERM	Speed Controller Integral Gain	-
SPEEDCNTR_CTERM	Speed Current Controller Anti-Windup Gain	-
SPEEDCNTR_OUTMAX	Speed Current Controller Saturation Limit	-

Class B Tests

This demonstration runs the following Class B tests during startup:

- Clock Test
- Program Counter Test
- CPU Registers Test
- Flash Test
- Checkerboard RAM Test
- March B Test
- March C Test

All Class B memory tests in this demonstration are non-destructive.

Faults

In this demonstration, the ON state of LED D2 indicates a Fault detection. This demonstration is equipped to detect a Class B test fail, and PFC or Motor Control faults. Class B tests are run immediately after the device power-up. If any of the Class B tests fail, it will be indicated by LED D2 turning ON. Also, the failure of Class B tests would inhibit the start of the PFC or motor. Once the Class B tests have passed successfully, the motor can be started by pressing the switch ‘S1’. While the PFC and motor are operational, if there is an Overcurrent or over-voltage condition detected on the PFC or Motor, this will cause the MCPWM to shutdown and the LED D2 will turn ON indicating a Fault.

integrated_pfc_foc_mchv3_int_opamp

This section provides information on the supported demonstration boards, how to configure the hardware, and how to run the demonstration.

Description

This demonstration implements integrated application of average current mode PFC and Sensorless Field Oriented Control (FOC) of a PMSM using a dual shunt configuration, and utilizes internal (on-chip) Op amps for current sensing.

Building the Application

This section identifies the MPLAB X IDE project name and location. It also lists and describes the available configurations for the demonstration.

Description

To build this project, you must open the `integrated_pfc_foc_mchv3_int_opamp.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is:

`<install-dir>/apps/motor_control/integrated_pfc_foc_mchv3_int_opamp`

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`integrated_pfc_foc_mchv3_int_opamp.X`	`<install-dir>/apps/motor_control/integrated_pfc_foc_mchv3_int_opamp/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are within `./firmware/src/system_config`

Project Configuration Name	BSP (s) Used	Description
motor_80_mchv3_int_opamp	N/A	Demonstrates integrated application of average current mode PFC and sensorless FOC of PMSM motor using internal op-amps.

Configuring the Hardware

This section describes how to configure the supported hardware.

Description

PIC32MK 100-pin Motor Control PIM with the dsPICDEM MCHV-3 Development Board

1. Mount the PIC32MK 100-pin Motor Control PIM on the dsPICDEM MCHV-3 Development Board, as shown in the following Figure.
2. Connect the Internal Op amp Configuration Matrix Board to header J4.
3. Connect the three-phase wires of the PMSM motor to M1, M2 and M3 ports.
4. Connect to the on board programmer and debugger to J20 using USB Mini B cable.
5. Connect the AC mains using J1.
6. This application supports communication with the X2C-Scope plug-in to monitor and plot any variables used in the application. The communication between the target and the X2C-Scope plug-in can be established using the RS-232 serial port (J8) or mini-USB port (J6). Refer to the jumper positions marked on the development board chassis to select between the RS-232 serial port (J8) and the mini-USB port (J6). Refer to Using the X2C-Scope Plug-in for details on how to use the X2C-Scope in this demonstration.

Note:	Please refer to dsPICDEM MCHV-3 Development Board User's Guide for safety and operating details of the development board.



Running the Demonstration

This section provides instructions on how to build and run the demonstration.

Description

For instructions, refer to the Running the Demonstration section for the integrated_pfc_foc_mchv3_int_opamp demonstration.

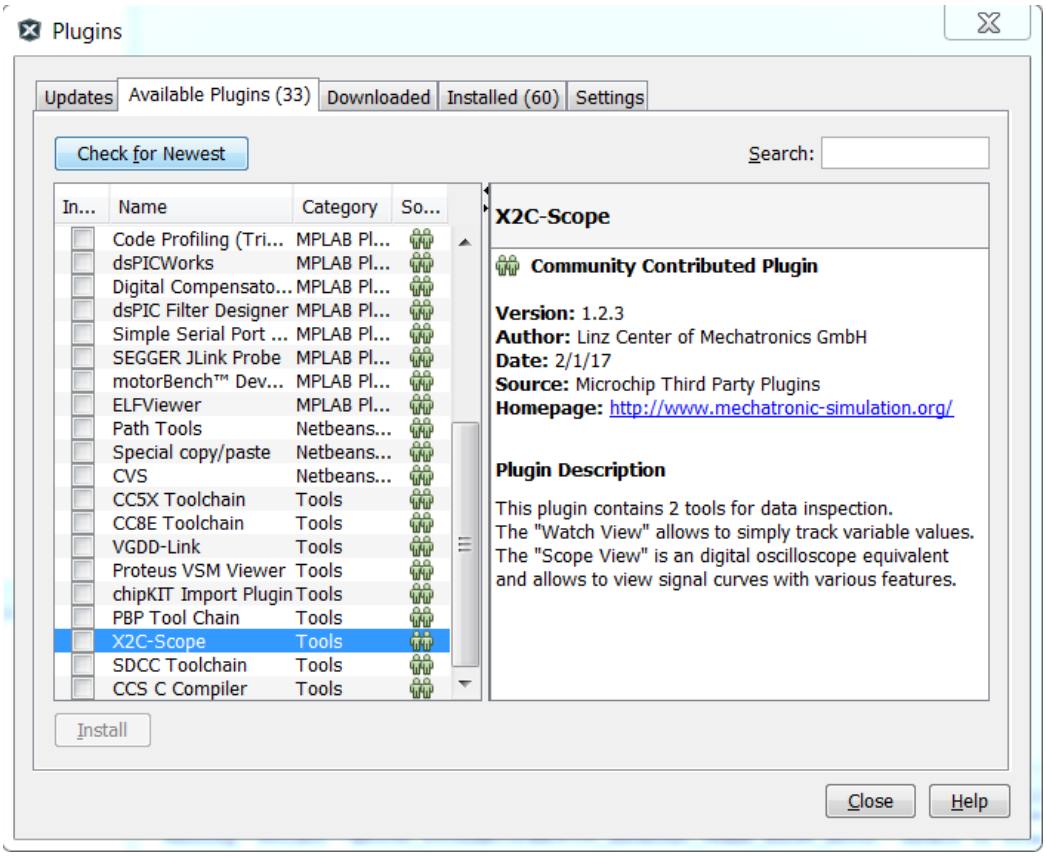
X2C-Scope Plug-in

This topic describe how to use the X2C-Scope MPLAB X IDE plug-in.

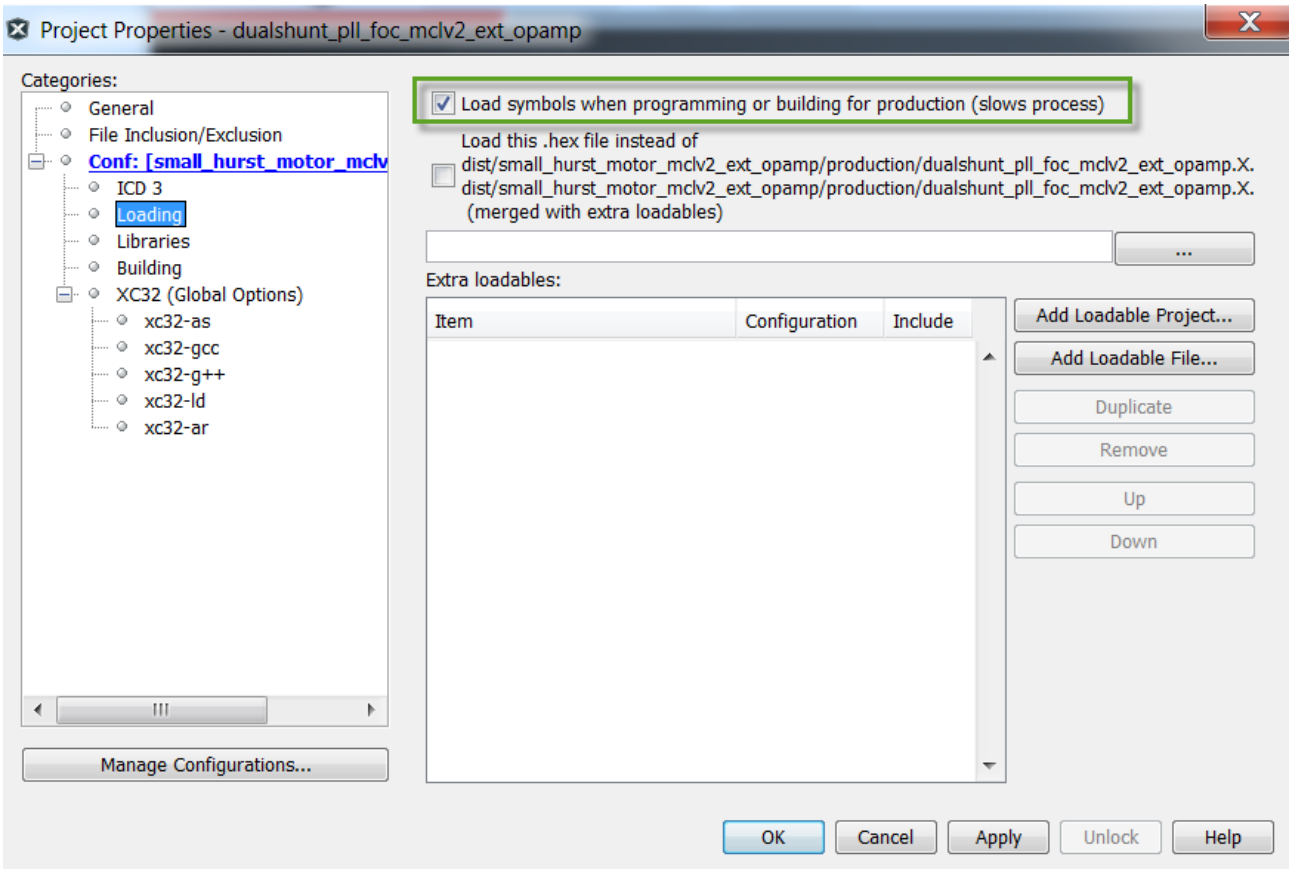
Description

The MPLAB X IDE enables use of the X2C-Scope plug-in to read, write, and plot global variables in real time. In this demonstration, the X2C-Scope communicates with the target using UART Channel 2. The baud rate of the UART communication can be modified using the MPLAB Harmony Configurator (MHC) (the default value is 38400 bps).

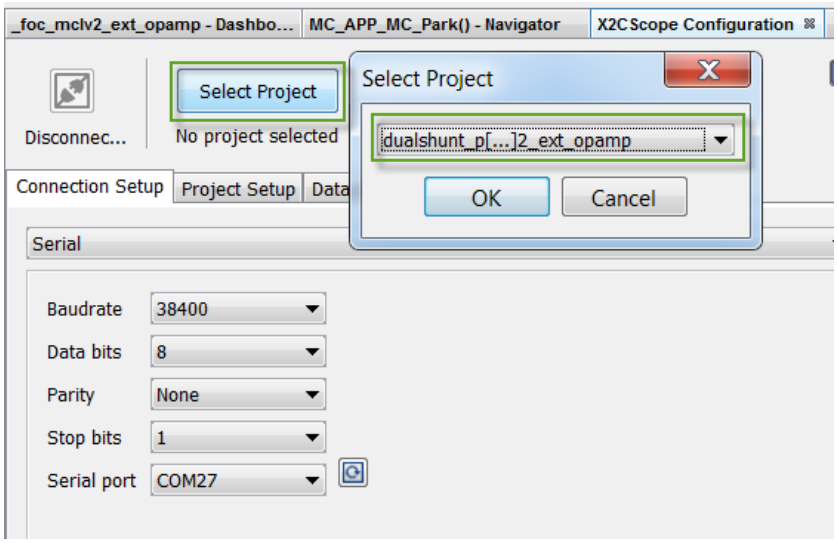
1. If not already installed, in MPLAB X IDE, select *Tools* > *Plugins* > *Available Plugins* > *Select X2C-Scope*> *Install*.
2. Restart MPLAB X IDE to complete the plug-in installation.



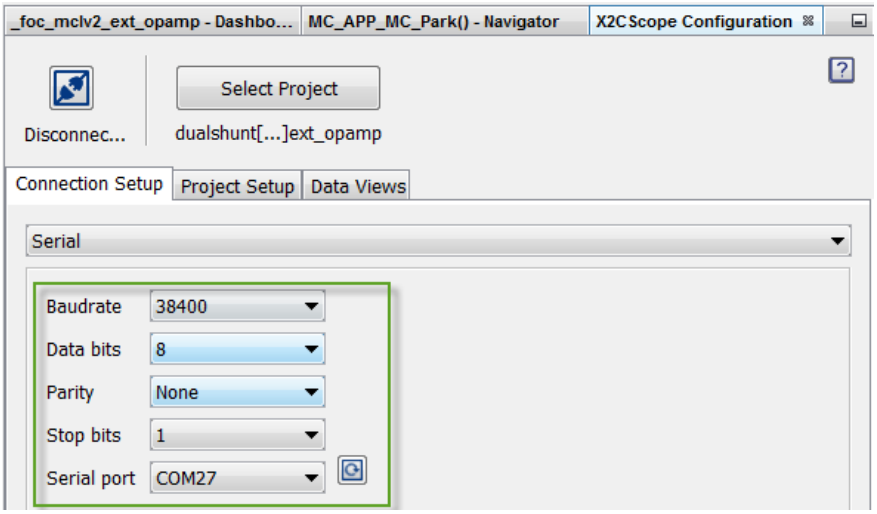
3. Open the X2C-Scope by selecting *Tools* > *Embedded* > *X2C-Scope*.
4. Ensure the symbols are loaded during the project build by selecting *Project Properties* > *Selected Configuration* > *Loading*, and ensure that the check box for *Load Symbols when programming or building for production (slows process)* is selected. If this check box was not selected, rebuild the project and reprogram the device.



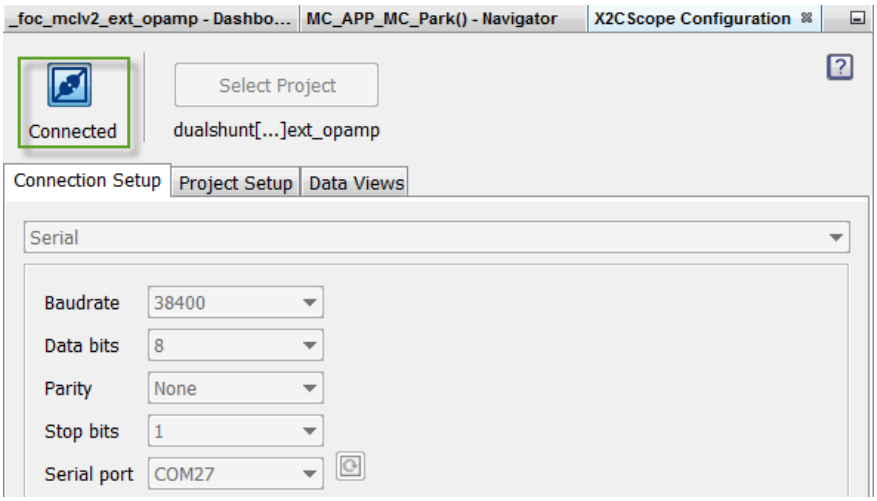
5. In the *X2C-Scope Configuration* tab, select the project.



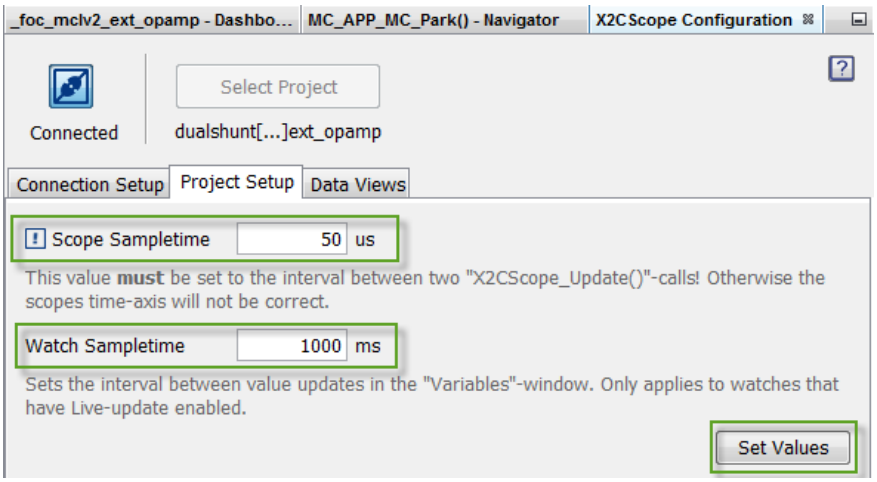
6. In Connection Setup, set the *Baud Rate* to '38400', *Data bits* to '8', *Parity* to 'None', *Stop Bits* to '1', and then select the associated COM port.



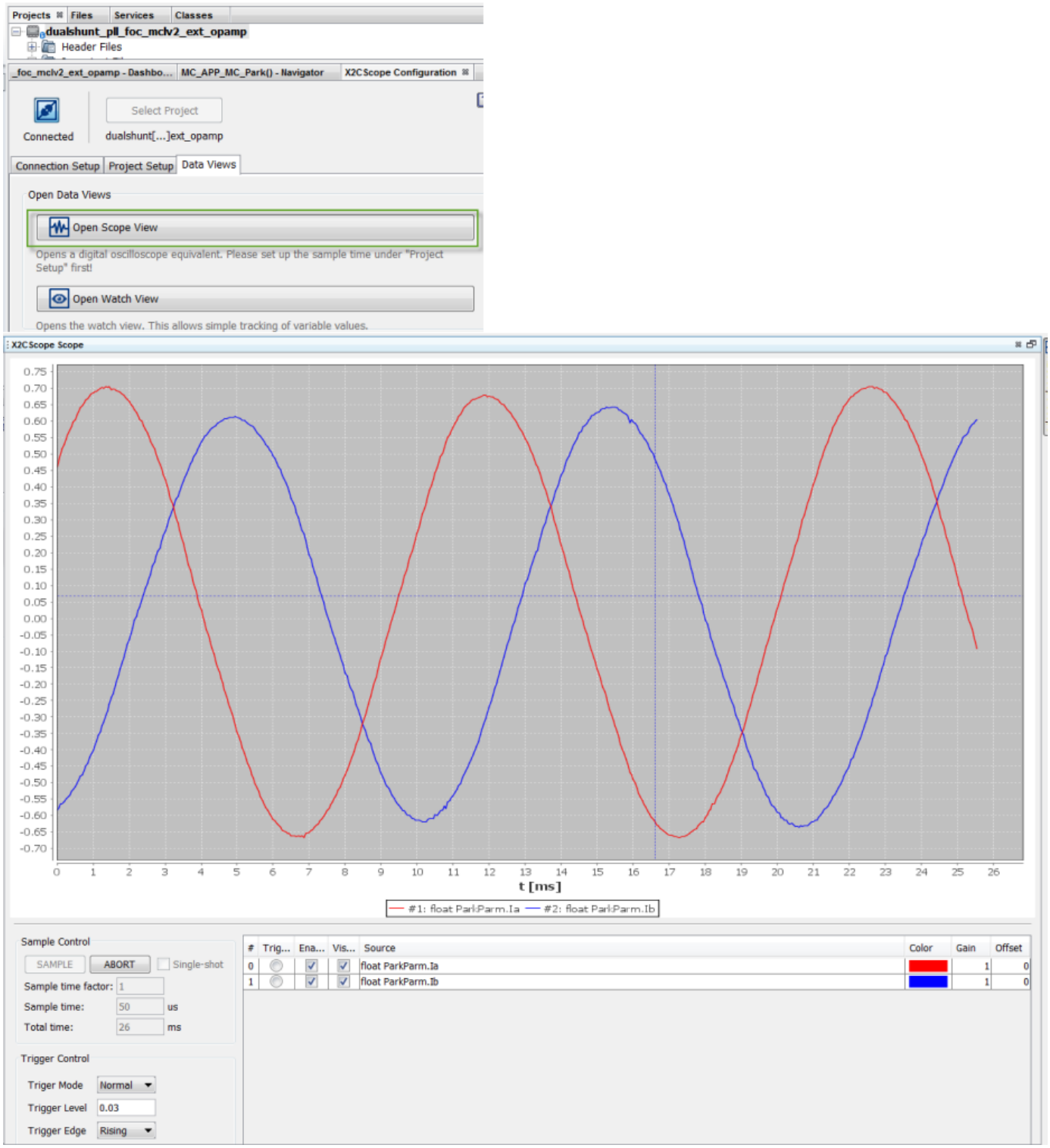
7. Connect to the target by clicking **Connect** (or **Disconnect**).



8. In *Project Setup*, Set the *Scope Sampletime* to '50 us.' In this demonstration, the `X2CScope_Update` function is called from the motor control ISR, which executes every 50 µs. Set the *Watch Sample time* to a value at which you want to update the watch window variables (the default is 1000 ms). Click **Set Values** after setting the *Scope Sampletime* and *Watch Sampletime*.

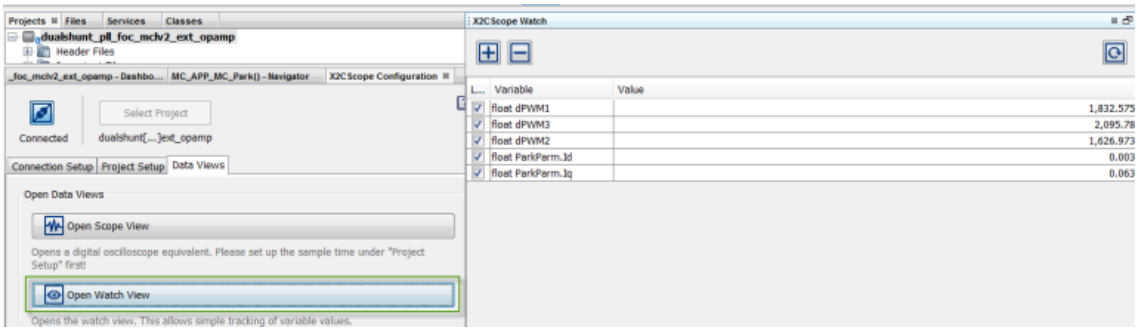


9. In *Data Views*, select *Open Scope View* to plot any two global variables in run-time.



Note:	The free version of X2C-Scope allows up to two global variables to be plotted simultaneously. The professional Version of X2C-Scope allows up to seven global variables to be plotted simultaneously.

11. Under *Data Views*, select *Open Watch View* to read or write to any global variables in run time.



RTOS Demonstrations

This section provides descriptions of the RTOS demonstrations.

MPLAB Harmony is available for download from the Microchip website by visiting: http://www.microchip.com/mplabharmony. Once you are on the site, click the Downloads tab to access the appropriate download for your operating system. For additional information on this demonstration, refer to the “Applications Help” section in the MPLAB Harmony Help.

Introduction

RTOS Demonstration Applications Help

Description

This distribution package contains a variety of RTOS-based firmware projects that demonstrate the capabilities of the MPLAB Harmony services and stacks integrated with RTOS running on PIC32 devices. This section describes the hardware requirement and procedures to run these firmware projects on Microchip demonstration and development boards.

To learn more about MPLAB Harmony stacks and libraries refer to the related documentation in Volume V: MPLAB Harmony Framework Reference.

Source Code Disclaimers

OPENRTOS

The OPENRTOS demonstrations provided in MPLAB Harmony use the OPENRTOS evaluation license, which is meant for demonstration purposes only. Customers desiring development and production on OPENRTOS must procure a suitable license. Please refer to one of the following documents, which are located in the `third-party` folder of the MPLAB Harmony installation, for information on obtaining an evaluation license for your device:

- `OpenRTOS Click Thru Eval License PIC32MXxx.pdf`
- `OpenRTOS Click Thru Eval License PIC32MZxx.pdf`

Micriμm

All μC/OS-III demonstrations have added the `crt0.S` "C" run-time library start-up file to the project. The demonstration sets the linker option "do not link startup code". This is necessary for μC/OS-III to work correctly with PIC32 devices as the general exception vector is located in `crt0.S`. μC/OS-III overrides this interrupt source (general exception handler) to perform OS-specific functionality.

If the user wants to implement their own application using μC/OS-III and a PIC32 device, they must add the `crt0.S` file to their project and override the general exception interrupt vector. See the current RTOS examples for this implementation.

A `crt0.S` template file can be found in the MPLAB XC32 C/C++ Compiler installation directory: `..\Microchip\xc32\<version>\pic32-libs\libpic32`.

Important! The Micriμm μC/OS-II and μC/OS-III source code that is distributed with MPLAB Harmony is for FREE short-term evaluation, for educational use, or peaceful research. If you plan or intend to use μC/OS-II and μC/OS-III in a commercial application/product, you need to contact Micriμm to properly license μC/OS-II and μC/OS-III for its use in your application/product. The source code is provided for your convenience and to help you experience μC/OS-II and μC/OS-III. The fact the source is provided does NOT mean that you can use it commercially without paying a licensing fee. Knowledge of the source code may NOT be used to develop a similar product. If you are unsure about whether you need to obtain a license for your application, please contact Micriμm and discuss the intended use with a sales representative (www.micrium.com).

Express Logic ThreadX

The source code for the ThreadX RTOS is not freely distributed. To obtain source code and the proper licensing agreement go to the Express Logic ThreadX website: http://rtos.com/products/threadx/.

SEGGER embOS

The SEGGER embOS libraries provided with MPLAB Harmony use the SEGGER evaluation license, which is meant for demonstration purposes

only. Customers desiring development must procure a suitable license from SEGGER. To obtain source code and the proper licensing agreement visit the SEGGER embOS website: https://www.segger.com/license-models.html.

Express Logic ThreadX Demonstrations

This section provides descriptions of the Express Logic ThreadX RTOS demonstrations.

basic

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

Source Code Disclaimer

The source code for this ThreadX RTOS demonstration is not freely distributed. To obtain source code and the proper licensing agreement go to the Express Logic ThreadX website: http://rtos.com/products/threadx/. So that ThreadX can work with the applicable MPLAB Harmony demonstrations, install the source in the following location: `<install-dir>/third_party/rtos/ThreadX/`.

The demonstrations will not compile unless the source code is provided and installed in the correct location.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the ThreadX Basic Demonstration.

Description

To build this project, you must open the `basic_threadx.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/rtos/threadx/basic`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`basic_threadx.X`	`<install-dir>/apps/rtos/threadx/basic`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mz_ef_sk	pic32mz_ef_sk	This configuration runs on the PIC32MZ EF Starter Kit.
pic32mz_ef_sk_microMIPS	pic32mz_ef_sk	This configuration runs on the PIC32MZ EF Starter Kit in microMIPS mode.
pic32mx_sk_mips16	pic32mx_usb_sk2	This configuration runs on the PIC32 USB Starter Kit II.
pic32mx_sk	pic32mx_usb_sk2, pic32mx_usb_sk3, pic32mx_eth_sk, and pic32mx_eth_sk2	This configuration runs on PIC32MX-based starter kits: PIC32 Ethernet Starter Kit, PIC32 Ethernet Starter Kit II, PIC32 USB Starter Kit II, PIC32 USB Starter Kit III.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 Ethernet Starter Kit, PIC32 Ethernet Starter Kit II, PIC32 USB Starter Kit II, and PIC32 USB Starter Kit III

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the Express Logic ThreadX basic demonstration.

Description

Once the demonstration is up and running an LED will toggle every 500 ms. This demonstration will show the RTOS running with the selected hardware.

usb

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

Source Code Disclaimer

The source code for this ThreadX RTOS demonstration is not freely distributed. To obtain source code and the proper licensing agreement go to the Express Logic ThreadX website: http://rtos.com/products/threadx/. So that ThreadX can work with the applicable MPLAB Harmony demonstrations, install the source in the following location: `<install-dir>/third_party/rtos/ThreadX/`.

The demonstrations will not compile unless the source code is provided and installed in the correct location.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Express Logic ThreadX and MPLAB Harmony Graphics plus USB Library Demonstration.

Description

To build this project, you must open the `usb_threadx.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/rtos/threadx/usb`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`usb_threadx.X`	`<install-dir>/apps/rtos/threadx/usb`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mz_ef_sk	pic32mz_ef_sk	This configuration runs on the PIC32MZ EF Starter Kit.
pic32mx_usb_sk2	pic32mx_usb_sk2	This configuration runs on the PIC32 USB Starter Kit II.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32 USB Starter Kit II

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the ThreadX and MPLAB Harmony USB Library demonstration.

Description

Power the board, open the demonstration with MPLAB X IDE, and then build and load the demonstration. A USB keyboard will be emulated when a micro USB cable is plugged into J4 on the PIC32MZ EC Starter Kit. Keystrokes, a-z and 1-9, will be sent when the push button SW3 is pressed. The user LED, D3, will toggle every 500 ms.

There are three tasks and four interrupts used in this application/system.

SystemUSBDeviceTask is the highest priority task and is responsible for getting all USB data queued up by the application ready for sending over the USB by calling the appropriate MPLAB Harmony USB stack function. ApplicationUSBDeviceTask is the next highest priority task. ApplicationUSBDeviceTask emulates the keyboard key presses when it detects that the push button SW3 is being pressed. ApplicationLEDblinkTask is the lowest priority task and toggles the user LED, D3, every 500 ms.

CPU core timer hardware interrupt is used by the RTOS to source the RTOS Tick. The Timer2 hardware interrupt is used to call the MPLAB Harmony Timer Driver function. USB general event interrupt is used to call the appropriate MPLAB Harmony USB driver function, which takes all USB data queued up by the application and physically writes it out to the USB hardware. The SYSCALL general exception is invoked by the RTOS and is used to process the task context switch routine.

FreeRTOS Demonstrations

This section provides descriptions of the FreeRTOS RTOS demonstrations.

basic

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

The demonstration blinks the three user LEDs on a starter kit to show the RTOS threads that are running and to indicate status.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the FreeRTOS Basic Demonstration.

Description

To build this project, you must open the `basic_freertos.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/rtos/freertos`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`basic_freertos.X`	`<install-dir>/apps/rtos/freertos/basic`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./src/system_config`.

Project Configuration Name	BSP(s) Used	Description
pic32mx_sk	pic32mx_usb_sk2, pic32mx_usb_sk3, pic32mx_eth_sk, and pic32mx_eth_sk2	This configuration runs on PIC32MX-based starter kits: PIC32 Ethernet Starter Kit, PIC32 Ethernet Starter Kit II, PIC32 USB Starter Kit II, PIC32 USB Starter Kit III.
pic32mx_sk_mips16	pic32mx_usb_sk2, pic32mx_usb_sk3, pic32mx_eth_sk, and pic32mx_eth_sk2	This configuration runs on PIC32MX-based starter kits in MIPS16 mode: PIC32 Ethernet Starter Kit, PIC32 Ethernet Starter Kit II, PIC32 USB Starter Kit II, PIC32 USB Starter Kit III.
pic32mz_sk	pic32mz_ec_sk	This configuration runs on the PIC32MZ EC Starter Kit.
pic32mz_ef_sk	pic32mz_ef_sk	This configuration runs on the PIC32MZ EF Starter Kit.
pic32mz_ef_sk_microMIPS	pic32mz_ef_sk	This configuration runs on the PIC32MZ EF Starter Kit in microMIPS mode.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 Ethernet Starter Kit, PIC32 Ethernet Starter Kit II, PIC32 USB Starter Kit II, and PIC32 USB Starter Kit III

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the FreeRTOS basic demonstration.

Description

Please use the following procedure to run the demonstration:

1. Load the demonstration project into MPLAB X IDE.
2. Connect the debug USB port of the target board to a USB port on the development computer using the USB cable provided in the kit.
3. Build, download, and run the demonstration project on the target board.

The demonstration application features the following:

- Application creates one queue and four tasks. One task that sends the data using the FreeRTOS queue to the two tasks that wait for the data in the queue. (QueueReceiveTask2 priority is higher than the QueueReceiveTask1 priority.)
- QueueReceiveTask2 receives the data first, toggles the LED, and then sleeps for the specified time
- QueueReceiveTask1 receives the next data since QueueReceiveTask2 is not in running state
- QueueReceiveTask1 receives the data, toggles the LED and waits for the data arrival

cdc_com_port_dual

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

This RTOS based demonstration application creates a USB CDC Device that enumerates as two serial ports on the USB Host personal computer. This application demonstrates the ability of the MPLAB Harmony USB Stack to operate in an Real-Time Operating System (this example uses FreeRTOS) and to support multiple instances of the same device class.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for this demonstration application.

Description

To build this project, you must open the `cdc_com_port_dual.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/rtos/freertos/cdc_com_port_dual`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`cdc_com_port_dual.X`	`<install-dir>/apps/rtos/freertos/cdc_com_port_dual/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx_usb_sk2_int_dyn	pic32mx_usb_sk2	Select this configuration to run the demonstration application on the PIC32 USB Starter Kit II in Interrupt mode and dynamic operation.

pic32mz_ef_sk_int_dyn	pic32mz_ef_sk	Select this configuration to run the demonstration application on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit in Interrupt mode and dynamic operation.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 USB Starter Kit II

Remove jumper JP2.

PIC32MZ EF Starter Kit

Remove jumper JP1.

Running the Demonstration

Provides instructions on how to build and run the cdc_com_port_dual demonstration.

Description

Refer to the Running the Demonstration topic for the bare-metal (non RTOS) version of the cdc_com_port_dual demonstration applications for running the demonstration.

The demonstration application contains six tasks. A description of these tasks is as follows:

- The FrameworkTasks task is created in the SYS_Tasks function. This task calls the USB Device Layer Tasks function (USB_DEVICE_Tasks). The priority of this task is designed to be the lowest when compared to the priorities of other tasks. Hence this tasks runs when all other tasks are either in blocked state or not ready to run.
- The APP_USB_DEVICE_Open task is created in the APP_Tasks function. This task creates all the semaphores and message queues needed for the application. It creates 4 tasks which implement the application logic. It attempts to open the Device Layer and then blocks on the xSemaphoreBlockUsbConfigure semaphore. The xSemaphoreBlockUsbConfigure is given in the USB Device Layer Event Handler when the device is configured by the Host. The tasks then resumes the 4 application logic tasks and suspends itself.
- The APP_CDC1Read Task is created in the APP_USB_DEVICE_Open task. It schedules a read on the CDC1 instance and then blocks on the CDC1 instance xSemaphoreCDCReadComplete semaphore. This semaphore is given in the Read Complete event in the CDC Application Event Handler. The tasks will then post the data that it has received from the Host to the CDC 1 instance Message Queue.
- The APP_CDC2Read Task is created in the APP_USB_DEVICE_Open task. It schedules a read on the CDC2 instance and then blocks on the CDC2 instance xSemaphoreCDCReadComplete semaphore. This semaphore is given in the Read Complete event in the CDC Application Event Handler. The tasks will then post the data that it has received from the Host to the CDC 1 instance Message Queue.
- The APP_CDC1Write Task is created in the APP_USB_DEVICE_Open task. It blocks on the CDC 2 message queue. When APP_CDC2Read Task posts a message to this queue, the APP_CDC1Write gets ready to run and the writes the data (received on the queue) to the CDC 1. This data is then transferred to the Host.
- The APP_CDC2Write Task is created in the APP_USB_DEVICE_Open task. It blocks on the CDC 1 message queue. When APP_CDC1Read Task posts a message to this queue, the APP_CDC2Write gets ready to run and the writes the data (received on the queue) to the CDC 2. This data is then transferred to the Host.

cdc_msd_basic

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for this demonstration application.

Description

To build this project, you must open the `cdc_msd_basic.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/rtos/freertos/cdc_msd_basic`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`cdc_msd_basic.X`	`<install-dir>/apps/rtos/freertos/cdc_msd_basic/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx_usb_sk2_int_dyn	pic32mx_usb_sk2	Select this configuration to run the demonstration application on the PIC32 USB Starter Kit II in Interrupt mode and dynamic operation.
pic32mz_ec_sk_int_dyn	pic32mz_ec_sk	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32MZ Embedded Connectivity (EC) Starter Kit configured for Interrupt mode and dynamic operation.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 USB Starter Kit II

JP2 should be in place if the attached USB device is bus-powered. It should be removed if the attached USB device is self-powered.

PIC32MZ Embedded Connectivity (EC) Starter Kit

Remove jumper JP1.

Running the Demonstration

Provides instructions on how to build and run the cdc_msd_basic demonstration.

Description

This USB Host demonstration application exercises the CDC and MSD interfaces on the attached composite USB device.

1. Open the project in MPLAB X IDE and select the desired project configuration.
2. Build the code and program the device.
3. Follow the directions for setting up and running the cdc_serial_emulator_msd USB device demonstration.
4. Connect the UART (P1) port on the Explorer 16 Development Board (running the cdc_serial_emulator_msd demonstration) to a USB Host personal computer via a commercially available Serial-to-USB Dongle.
5. Start a terminal program on the USB Host personal computer and select the Serial-to-USB Dongle as the communication port. Select the baud rate as 9600, no parity, 1 Stop bit and no flow control.
6. Connect the mini – B connector on the USB PICtail Plus Daughter Board, of the cdc_serial_emulator_msd demonstration setup, to the Type-A USB host connector on the starter kit.
7. A prompt (DATA :) will be displayed immediately on the terminal emulation program.
8. Type a string less than 12 characters and press the <Enter> key. The string entered here will be stored in the MSD device in a file named `file.txt`.
9. Step 8 can be repeated. Data entered in the prompt will be appended to the file.
10. Unplug the USB Device from the Host and connect it a personal computer host to examine the contents of the Mass Storage Device. This should contain a file named `file.txt` and this should contain the data that was entered at the terminal program prompt.

Tasks

This USB Host demonstration application contains four tasks. Descriptions of these tasks is as follows:

- The FrameworkTasks task is created in the SYS_Tasks function. This task calls the USB Host Layer Tasks function (USB_HOST_Tasks). The priority of this task is designed to be the lowest when compared to the priorities of other tasks. Hence this tasks runs when all other tasks are either in blocked state or not ready to run.
- The APP_USB_HOST_Open task is created in the APP_Tasks function. This task creates all the semaphores and message queues needed for the application. It creates two tasks that implement the application logic. It attempts to open the Host Layer and then enable USB Host Operation. The task then resumes the two application logic tasks and suspends itself.
- The APP_USBHostCDCTask Task is created in the APP_USB_HOST_Open task. It blocks on xSemaphoreUSBCDCAttach semaphore. This semaphore is given in the Application CDC Class Driver event handler when a CDC device is enumerated. The task then prints a prompt and schedules a read from the CDC interface of the attached USB device. When data is available, it posts the xSemaphoreCDCReadComplete semaphore.
- The APP_USBHostMSDTask Task is created in the APP_USB_HOST_Open task. It blocks on the xSemaphoreUSBMSDAttach semaphore. This semaphore is given in the Application MSD event Handler when a MSD device is enumerated. The task then mounts the attached storage media and then blocks on the xSemaphoreCDCReadComplete semaphore. The xSemaphoreCDCReadComplete semaphore is given by the APP_USBHostCDCTask task when the CDC Host has received data. The APP_USBHostMSDTask Task will then open a file on the mounted drive and will append the data (received from CDC) in the file. It closes the file, and then appends on xSemaphoreCDCReadComplete semaphore again.

tcpip_client_server

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

The TCP/IP Client Server application, tcpip_client_server, demonstrates how to run multiple TCP and UDP servers and clients using the TCP/IP Stack in an RTOS environment. The demonstration also has the HTTP Web server running using the Non-Volatile Memory (NVM) Microchip Proprietary File System (MPFS) to store the web pages in the internal PIC32 Flash.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the FreeRTOS and MPLAB Harmony TCP/IP Demonstration.

Description

To build this project, you must open the `tcpip_client_server.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/rtos/freertos`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`tcpip_client_server.X`	`<install-dir>/apps/rtos/freertos/tcpip_client_server/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within ./firmware/src/system_config.

Project Configuration Name	BSP Used	Description
pic32mz_ef_sk	pic32mz_ef_sk	This configuration runs on the PIC32MZ EF Starter Kit and the Starter Kit I/O Expansion Board.
pic32mx_eth_sk	pic32mx_eth_sk	This configuration runs on the PIC32 Ethernet Starter Kit and the Starter Kit I/O Expansion Board.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MZ EF Starter Kit connected to the Starter Kit I/O Expansion Board

No jumper settings are required for this configuration.

The demonstration makes extensive use of the UART. To use the UART output, you will need to connect an RS-232 level-shifter to UART2 on the Starter Kit I/O Expansion Board. From J11, connect U2TX (48) to the level-shifter TX and connect U2RX (46) to the level-shifter RX. Use any +5V and GND to complete the wiring of the RS-232 level-shifter. The baud rate is 115200 Baud, N, 8, 1.

PIC32 Ethernet Starter Kit connected to the Starter Kit I/O Expansion Board

No jumper settings are required for this configuration.

The demonstration makes extensive use of the UART. To use the UART output, you will need to connect an RS-232 level-shifter to UART2 on the Starter Kit I/O Expansion Board. From J11, connect U2TX (48) to the level-shifter TX and connect U2RX (46) to the level-shifter RX. Use any +5V and GND to complete the wiring of the RS-232 level-shifter. The baud rate is 115200 Baud, N, 8, 1.

Running the Demonstration

Provides instructions on how to build and run the FreeRTOS RTOS with TCP/IP demonstration.

Description

The run the demonstration application, apply power to the board, open the demonstration with MPLAB X IDE, and then build and load the

demonstration.

This demonstration runs IPv4 only on the Ethernet interface. To view the Web page hosted by the demonstration application, open a Web browser and direct it to the board running the HTTP server by typing the URL in the address bar (for example, http://mchpboard_e or http://mchpboard_c), and then pressing **Enter**.

Notes:

1. The NetBIOS name of the TCP/IP application is specified at the time the TCP/IP stack is initialized, which is usually in the hostName member of the tcpip_stack_init.c:: TCPIP_HOSTS_CONFIGURATION structure. The NetBIOS service must be enabled for the PIC32 demonstration to respond to NetBIOS queries (the NetBIOS service is enabled by default for this demo). Alternatively, you can use the IPv4 of the board directly, for example, http://192.168.1.131.
2. The IPv4 address can be obtained from running the TCP/IP Discovery application on the PC side. It requires that the TCP/IP Announce module is enabled when building the stack (the Announce module is enabled by default in this application).

Advanced Features

To use the advanced features of this demonstration, a system console must be enabled for the application. A serial console is preferred and is enabled by default for the demonstration. Alternatively, the application can be reconfigured using MHC to use the USB console.

The PIC32MZ configurations have Telnet enabled by default. A Telnet connection could be also used for delivering the commands needed by the application. On the PIC32MX configuration, the Telnet module is *not* enabled due to limited memory resources.

The PIC32MZ configuration is preferred for running this demonstration as the PIC32MX version may run out of memory at run-time.

TCP/IP Tasks

There are four TCP/IP tasks in the application that demonstrate the use of IPv4 TCP and UDP sockets in a multi-threaded system. Each of these tasks implements (and registers with the system command processor) specific commands. The commands allow the corresponding sockets to be opened and to start the communication with the remote hosts. On all hosts, the server sockets must first be opened, and then have the client sockets connect to them. There are also commands for configuring each socket/communication channel.

Following the model in this application, extra commands could be added for any of the tasks to achieve desired operation.

Each of these communication channels can be exercised simultaneously or in turn. For the purposes of the demonstration, at least two different communication channels should be opened simultaneously to put in evidence the multi-threaded behavior.

A common scenario listing the console commands and steps needed for running this demonstration would be:

1. Start app1 – TCP server:
   - Start the PIC32 TCP server socket that listens on port 9760 by issuing the console command: `topen_s1<CR>`
   - On the client side (PC, etc.) open a client that connects over TCP to the PIC32 server using port 9760. Use any network tools (netcat, etc.) or special applications, scripts (TCL, Python, etc.) to transmit and receive data files.
2. Start app2 – TCP client:
   - On the remote host side (PC, etc.) open a TCP server that listens for incoming connections on port 9761. Use any network tools (netcat, etc.) or special applications, scripts (TCL, Python, etc.) to receive (and transmit) data files.
   - Set the PIC32 client side address and port for the server to connect to, for example: `tsrv4_c1 192.168.100.101 9761<CR>`
   - Start the PIC32 TCP client socket by issuing the console command: `topen_c1<CR>`
3. Start app3 – UDP server:
   - Start the PIC32 UDP server socket that listens on port 32323 by issuing the console command: uopen_s1<CR>
   - On the client side (PC, etc.) open a client that connects over UDP to the PIC32 server using port 32323. Use any network tools (netcat, etc.) or special applications, scripts (TCL, Python, etc.) to transmit and receive data files.
4. Start app4 – UDP client:
   - On the remote host side (PC, etc.) open a UDP server that listens for incoming connections on port 32324. Use any network tools (netcat, etc.) or special applications, scripts (TCL, Python, etc.) to receive (and transmit) data files.
   - Set the PIC32 client side address and port for the server to connect to, for example: `usrv4_c1 192.168.100.101 32324<CR>`
   - Start the PIC32 UDP client socket by issuing the console command: `uopen_c1<CR>`
5. Now that you have the TCP/IP tasks running you can check the progress at run time. These commands give the RX and TX statistics showing the amount of data transferred by each task:
   - `tstat_s1<CR>`
   - `tstat_c1<CR>`
   - `ustat_s1<CR>`
   - `ustat_c1<CR>`
6. Once the data transfer is completed, close the TCP/IP sockets (if not already closed by the remote party):
   - `tclose_s1<CR>`
   - `tclose_c1<CR>`
   - `uclose_s1<CR>`
   - `uclose_c1<CR>`

TCP/IP Task Descriptions and Commands

app1.c::TCP server

This task uses a TCP server socket that listens by default on port 9760 to implement a simple relay server. Any message that is received on that TCP port will be relayed back to the originating socket. The following table lists and describes the available commands

Command	Description
topen_s1	Opens the listening TCP server socket.
tmsg_s1	Sends a short message using this socket to the remote client.
tabort_s1	Sends an abort/RST to the remote client and stops the communication.
tclose_s1	Closes the socket and stops the communication.
ttxsize_s1	Sets the TX buffer size of the server socket. The larger the buffer the more memory is used by the socket and the more efficient is the transfer. Default value is 2048 bytes.
trxsize_s1	Sets the RX buffer size of the server socket. The larger the buffer the more memory is used by the socket and the more efficient is the transfer. Default value is 2048 bytes.
tdisplay_s1	Enables displaying of the received messages locally to the system console.
tstat_s1	Displays/clears the current TX and RX statistics for the current connection.

app2.c::TCP client

This task uses a TCP client socket to connect to a remote server that listens by default on port 9761. Any message that is received by the socket can be optionally displayed locally. The client has the possibility of sending messages to the server. The following table lists and describes the available commands.

Command	Description
topen_c1	Opens the TCP client socket.
tmsg_c1	Sends a message to the remote server.
tabort_c1	Sends an abort/RST to the server and stops the communication.
tclose_c1	Closes the socket and stops the communication.
tsrv4_c1	Sets the server IPv4 address and port. The default values are 192.168.100.101 and 9761.
tasync_c1	Enables the continuous transmission of messages to the server.
tdisplay_c1	Enables displaying of the received messages locally to the system console.
tstat_c1	Displays/clears the current TX and RX statistics for the current connection.

app3.c::UDP server

This task uses a UDP server socket that listens by default on port 32323 to implement a simple relay server. Any message that is received on that UDP port will be relayed back to the originating socket. The following table lists and describes the available commands.

Command	Description
uopen_s1	Opens the listening UDP server socket.
uclose_s1	Closes the socket and stops the communication.
ustnet_s1	Selects the strict network option for the UDP socket (incoming connections from any network are allowed or only from the network that initiated the first connection).
ustport_s1	Selects the strict port option for the UDP socket (incoming connections from any host port are allowed or only from the port that was used when the connection was first initiated).
ustadd_s1	Selects the strict address option for the UDP socket (incoming connections from any host address are allowed or only from the address that was used when the connection was first initiated).
udisplay_s1	Enables displaying of the received messages locally to the system console.
ustat_s1	Displays/clears the current TX and RX statistics for the current connection.
utxsize_s1	Sets the TX buffer size of the server socket. The larger the buffer the more memory is used by the socket and the more efficient is the transfer. Default value is 1024 bytes. Note that this value should not be made larger than 1460 for an Ethernet network (to avoid packets larger than the link MTU).

app4.c::UDP client

This task uses a UDP client socket to connect to a remote server that listens by default on port 32324. Any message that is received by the socket can be optionally displayed locally. The client has the possibility of sending messages to the server. The following table lists and describes the available commands.

Command	Description
uopen_c1	Opens the UDP client socket.
umsg_c1	Sends a message to the remote server.

uclose_c	Closes the socket and stops the communication.
usrv4_c1	Sets the server IPv4 address and port. The default values are 192.168.100.101 and 32324.
uasync_c1	Enables the continuous transmission of messages to the server.
udisplay_c1	Enables displaying of the received messages locally to the system console.
ustat_c1	Displays/clears the current TX and RX statistics for the current connection.
utxsize_c1	Sets the TX buffer size of the server socket. The larger the buffer the more memory is used by the socket and the more efficient is the transfer. Default value is 1024 bytes. Note that this value should not be made larger than 1460 for an Ethernet network (to avoid packets larger than the link MTU).

Micrium uC_OS_II Demonstrations

This section provides descriptions of the Micriμm μC/OS-II RTOS demonstrations.

basic

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

The demonstration blinks a user LED on a starter kit to show the RTOS threads that are running and to indicate status.

Legal Disclaimer

The source code for this demonstration is not freely distributed. To obtain the proper licensing agreement go to the Micriμm website: http://www.micrium.com. The Micriμm μC/OS-II source has been installed in the following location, `<install_dir>/third_party/rtos/MicriumOSII/Software`, so that the applicable MPLAB Harmony demonstrations can work.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Micriμm μC/OS-II Basic Demonstration.

Description

To build this project, you must open the `basic_ucos_II.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/rtos/uC_OS_II`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
basic_ucos_II.X	<install-dir>/apps/rtos/uC_OS_II/basic

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx_sk	pic32mx_usb_sk2	This configuration runs on the PIC32 USB Starter Kit II.
pic32mz_sk	pic32mz_ec_sk	This configuration runs on the PIC32MZ EC Starter Kit.
pic32mz_ef_sk	pic32mz_ef_sk	This configuration runs on the PIC32MZ EF Starter Kit.
pic32mz_ef_sk_microMIPS	pic32mz_ef_sk	This configuration runs on the PIC32MZ EF Starter Kit in microMIPS mode.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MZ EC Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32 USB Starter Kit II

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the Micriμm μC/OS-II basic demonstration.

Description

In this demonstration, there is one user task and one hardware interrupt. The one user task, LEDBlinkTask, is responsible for toggling the user LED to show that the RTOS and the application are up and running. The hardware interrupt is used by the RTOS to run the RTOS Tick. The internal core timer is used as the source for the hardware interrupt.

Once the demonstration is up and running an LED will toggle every 500 ms. This demonstration will show the RTOS running with the selected hardware.

Micrium uC/OS-III Demonstrations

This section provides descriptions of the Micriμm μC/OS-III RTOS demonstrations.

basic

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

The demonstration blinks a user LED on a starter kit to show the RTOS threads that are running and to indicate status.

Legal Disclaimer

The source code for this demonstration is not freely distributed. To obtain the proper licensing agreement go to the Micriμm website: http://www.micrium.com. The Micriμm μC/OS-III source has been installed in the following location, `<install_dir>/third_party/rtos/MicriumOSIII/Software`, so that the applicable MPLAB Harmony demonstrations can work.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Micriμm μC/OS-III Basic Demonstration.

Description

To build this project, you must open the `basic_ucos_III.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/rtos/uC_OS_III`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`basic_ucos_III.X`	`<install-dir>/apps/rtos/uC_OS_III/basic`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx_sk	pic32mx_usb_sk2	This configuration runs on the PIC32 USB Starter Kit II.
pic32mz_sk	pic32mz_ec_sk	This configuration runs on the PIC32MZ EC Starter Kit.
pic32mz_ef_sk	pic32mz_ef_sk	This configuration runs on the PIC32MZ EF Starter Kit.
pic32mx_sk_mips16	pic32mx_usb_sk2	This configuration runs on the PIC32 USB Starter Kit II in MIPS16 mode.
pic32mz_ef_sk_microMIPS	pic32mz_ef_sk	This configuration runs on the PIC32MZ EF Starter Kit in microMIPS mode

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MZ EC Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32 USB Starter Kit II

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the Micriμm μC/OS-III basic demonstration.

Description

In this demonstration, there is one user task and one hardware interrupt. The one user task, LEDBlinkTask, is responsible for toggling the user LED to show that the RTOS and the application are up and running. The hardware interrupt is used by the RTOS to run the RTOS Tick. The internal core timer is used as the source for the hardware interrupt.

Once the demonstration is up and running an LED will toggle every 500 ms. This demonstration will show the RTOS running with the selected hardware.

usb

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

Legal Disclaimer

The source code for this demonstration is not freely distributed. To obtain the proper licensing agreement go to the Micriμm website: http://www.micrium.com. The Micriμm μC/OS-III source has been installed in the following location, `<install_dir>/third_party/rtos/MicriumOSIII/Software`, so that the applicable MPLAB Harmony demonstrations can work.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Micriμm μC/OS-III and MPLAB Harmony Graphics plus USB Library Demonstration.

Description

To build this project, you must open the `usb_ucos_III.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/rtos/uC_OS_III/usb`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`usb_ucos_III.X`	`<install-dir>/apps/rtos/uC_OS_III/usb`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mz_ef_sk	pic32mz_ef_sk	This configuration runs on the PIC32MZ EF Starter Kit.
pic32mx_usb_sk2	pic32mx_usb_sk2	This configuration runs on the PIC32 USB Starter Kit II.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32 USB Starter Kit II

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the Micriμm μC/OS-III and MPLAB Harmony USB Library demonstration.

Description

Power the board, open the demonstration with MPLAB X IDE, and then build and load the demonstration. A USB keyboard will be emulated when a micro USB cable is plugged in to the starter kit. Keystrokes, a-z and 1-9, will be sent when the push button SW3 is pressed. The user LED, D3, will toggle every 500 ms.

There are three tasks and four interrupts used in this application/system.

SystemUSBDeviceTask is the highest priority task and is responsible for getting all USB data queued up by the application ready for sending over the USB by calling the appropriate MPLAB Harmony USB stack function. ApplicationUSBDeviceTask is the next highest priority task. ApplicationUSBDeviceTask emulates the keyboard key presses when it detects that the push button SW3 is being pressed. ApplicationLEDblinkTask is the lowest priority task and toggles the user LED, D3, every 500 ms.

CPU core timer hardware interrupt is used by the RTOS to source the RTOS Tick. The Timer2 hardware interrupt is used to call the MPLAB Harmony Timer Driver function. USB general event interrupt is used to call the appropriate MPLAB Harmony USB driver function, which takes all USB data queued up by the application and physically writes it out to the USB hardware. The SYSCALL general exception is invoked by the RTOS and is used to process the task context switch routine.

OPENRTOS Demonstrations

This section provides descriptions of the OPENRTOS RTOS demonstrations.

basic

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

The demonstration blinks the three user LEDs on a starter kit to show the RTOS threads that are running and to indicate status.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the OPENRTOS Basic Demonstration.

Description

To build this project, you must open the `basic_openrtos.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/rtos/openrtos/basic`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`basic_openrtos.X`	`<install-dir>/apps/rtos/openrtos/basic`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mz_sk	pic32mz_ec_sk	This configuration runs on the PIC32MZ EC Starter Kit.
pic32mx_sk	pic32mx_usb_sk2, pic32mx_usb_sk3, pic32mx_eth_sk, and pic32mx_eth_sk2	This configuration runs on PIC32MX-based starter kits: PIC32 Ethernet Starter Kit, PIC32 Ethernet Starter Kit II, PIC32 USB Starter Kit II, and PIC32 USB Starter Kit III.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 Ethernet Starter Kit, PIC32 Ethernet Starter Kit II, PIC32 USB Starter Kit II, and PIC32 USB Starter Kit III

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ EC Starter Kit

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the OPENRTOS basic demonstration.

Description

Once the demonstration is up and running an LED will toggle every 500 ms. This demonstration will show the RTOS running with the selected hardware. The demonstration blinks the three user LEDs on a starter kit to show the RTOS threads that are running and to indicate status.

cdc_com_port_dual

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

This RTOS based demonstration application creates a USB CDC Device that enumerates as two serial ports on the USB Host personal computer. This application demonstrates the ability of the MPLAB Harmony USB Stack to operate in an Real-Time Operating System (this example uses OPENRTOS) and to support multiple instances of the same device class.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for this demonstration application.

Description

To build this project, you must open the `cdc_com_port_dual.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/rtos/openrtos/cdc_com_port_dual`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`cdc_com_port_dual.X`	`<install-dir>/apps/rtos/openrtos/cdc_com_port_dual/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx_usb_sk2_int_dyn	pic32mx_usb_sk2	Select this configuration to run the demonstration application on the PIC32 USB Starter Kit II in Interrupt mode and dynamic operation.
pic32mz_ef_sk_int_dyn	pic32mz_ef_sk	Select this configuration to run the demonstration application on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit in Interrupt mode and dynamic operation.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 USB Starter Kit II

Remove jumper JP2.

PIC32MZ EF Starter Kit

Remove jumper JP1.

Running the Demonstration

Provides instructions on how to build and run the cdc_com_port_dual demonstration.

Description

Refer to the Running the Demonstration topic for the bare-metal (non RTOS) version of the cdc_com_port_dual demonstration applications for running the demonstration.

The demonstration application contains six tasks. A description of these tasks is as follows:

- The FrameworkTasks task is created in the SYS_Tasks function. This task calls the USB Device Layer Tasks function (USB_DEVICE_Tasks). The priority of this task is designed to be the lowest when compared to the priorities of other tasks. Hence this tasks runs when all other tasks are either in blocked state or not ready to run.
- The APP_USB_DEVICE_Open task is created in the APP_Tasks function. This task creates all the semaphores and message queues needed for the application. It creates 4 tasks which implement the application logic. It attempts to open the Device Layer and then blocks on the xSemaphoreBlockUsbConfigure semaphore. The xSemaphoreBlockUsbConfigure is given in the USB Device Layer Event Handler when the device is configured by the Host. The tasks then resumes the 4 application logic tasks and suspends itself.
- The APP_CDC1Read Task is created in the APP_USB_DEVICE_Open task. It schedules a read on the CDC1 instance and then blocks on the CDC1 instance xSemaphoreCDCReadComplete semaphore. This semaphore is given in the Read Complete event in the CDC Application Event Handler. The tasks will then post the data that it has received from the Host to the CDC 1 instance Message Queue.
- The APP_CDC2Read Task is created in the APP_USB_DEVICE_Open task. It schedules a read on the CDC2 instance and then blocks on the CDC2 instance xSemaphoreCDCReadComplete semaphore. This semaphore is given in the Read Complete event in the CDC Application Event Handler. The tasks will then post the data that it has received from the Host to the CDC 1 instance Message Queue.
- The APP_CDC1Write Task is created in the APP_USB_DEVICE_Open task. It blocks on the CDC 2 message queue. When APP_CDC2Read Task posts a message to this queue, the APP_CDC1Write gets ready to run and the writes the data (received on the queue) to the CDC 1. This data is then transferred to the Host.
- The APP_CDC2Write Task is created in the APP_USB_DEVICE_Open task. It blocks on the CDC 1 message queue. When APP_CDC1Read Task posts a message to this queue, the APP_CDC2Write gets ready to run and the writes the data (received on the queue) to the CDC 2. This data is then transferred to the Host.

cdc_msd_basic

Demonstrates host support for a composite USB Device in a RTOS application.

Description

This demonstration application creates a USB Host application that demonstrates operation of composite USB Device. The Host application enumerates the CDC and MSD interfaces on the attached composite devices and then operates these in one application. The demonstration application uses a RTOS to create thread that manage the CDC and MSD aspects of the Host application.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for this demonstration application.

Description

To build this project, you must open the `cdc_msd_basic.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/rtos/openrtos/cdc_msd_basic`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`cdc_msd_basic.X`	`<install-dir>/apps/rtos/openrtos/cdc_msd_basic/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx_usb_sk2_int_dyn	pic32mx_usb_sk2	Select this configuration to run the demonstration application on the PIC32 USB Starter Kit II in Interrupt mode and dynamic operation.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 USB Starter Kit II

JP2 should be in place if the attached USB device is bus-powered. It should be removed if the attached USB device is self-powered.

Running the Demonstration

Provides instructions on how to build and run the cdc_msd_basic demonstration.

Description

This USB Host demonstration application exercises the CDC and MSD interfaces on the attached composite USB device.

1. Open the project in MPLAB X IDE and select the desired project configuration.
2. Build the code and program the device.
3. Follow the directions for setting up and running the cdc_serial_emulator_msd USB device demonstration.
4. Connect the UART (P1) port on the Explorer 16 Development Board (running the cdc_serial_emulator_msd demonstration) to a USB Host personal computer via a commercially available Serial-to-USB Dongle.
5. Start a terminal program on the USB Host personal computer and select the Serial-to-USB Dongle as the communication port. Select the baud rate as 9600, no parity, 1 Stop bit and no flow control.
6. Connect the mini – B connector on the USB PICtail Plus Daughter Board, of the cdc_serial_emulator_msd demonstration setup, to the Type-A USB host connector on the starter kit.
7. A prompt (DATA :) will be displayed immediately on the terminal emulation program.
8. Type a string less than 12 characters and press the <Enter> key. The string entered here will be stored in the MSD device in a file named `file.txt`.
9. Step 8 can be repeated. Data entered in the prompt will be appended to the file.

10. Unplug the USB Device from the Host and connect it a personal computer host to examine the contents of the Mass Storage Device. This should contain a file named `file.txt` and this should contain the data that was entered at the terminal program prompt.

Tasks

This USB Host demonstration application contains four tasks. Descriptions of these tasks is as follows:

- The FrameworkTasks task is created in the SYS_Tasks function. This task calls the USB Host Layer Tasks function (USB_HOST_Tasks). The priority of this task is designed to be the lowest when compared to the priorities of other tasks. Hence this tasks runs when all other tasks are either in blocked state or not ready to run.
- The APP_USB_HOST_Open task is created in the APP_Tasks function. This task creates all the semaphores and message queues needed for the application. It creates two tasks that implement the application logic. It attempts to open the Host Layer and then enable USB Host Operation. The task then resumes the two application logic tasks and suspends itself.
- The APP_USBHostCDCTask Task is created in the APP_USB_HOST_Open task. It blocks on xSemaphoreUSBCDCAttach semaphore. This semaphore is given in the Application CDC Class Driver event handler when a CDC device is enumerated. The task then prints a prompt and schedules a read from the CDC interface of the attached USB device. When data is available, it posts the xSemaphoreCDCReadComplete semaphore.
- The APP_USBHostMSDTask Task is created in the APP_USB_HOST_Open task. It blocks on the xSemaphoreUSBMSDAttach semaphore. This semaphore is given in the Application MSD event Handler when a MSD device is enumerated. The task then mounts the attached storage media and then blocks on the xSemaphoreCDCReadComplete semaphore. The xSemaphoreCDCReadComplete semaphore is given by the APP_USBHostCDCTask task when the CDC Host has received data. The APP_USBHostMSDTask Task will then open a file on the mounted drive and will append the data (received from CDC) in the file. It closes the file, and then appends on xSemaphoreCDCReadComplete semaphore again.

SEGGER embOS Demonstrations

This section provides descriptions of the SEGGER embOS RTOS demonstrations.

basic

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

The demonstration blinks the LED1 on a starter kit to show the RTOS threads are running and to indicate status.

Legal Disclaimer

The source code for this SEGGER embOS RTOS demonstration is not freely distributed. To obtain source code and the proper licensing agreement visit the SEGGER embOS website: https://www.segger.com/license-models.html.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the SEGGER embOS Basic Demonstration.

Description

To build this project, you must open the `basic_embos.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/rtos/embos/basic`.

Note: The floating point options used may depend on the libraries shared by SEGGER. The default libraries shared with MPLAB Harmony uses the soft floating point (mfloat-abi=softfp) option.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`basic_embos.X`	`<install-dir>/apps/rtos/embos/basic/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx_usb_sk2	pic32mx_usb_sk2	This configuration runs on the PIC32 USB Starter Kit II.
pic32mx_usb_sk2_mips16	pic32mx_usb_sk2	This configuration runs on the PIC32 USB Starter Kit II in MIPS16 mode.
pic32mz_ef_sk	pic32mz_ef_sk	This configuration runs on the PIC32MZ EF Starter Kit.
pic32mz_ef_sk_microMIPS	pic32mz_ef_sk	This configuration runs on the PIC32MZ EF Starter Kit in microMIPS mode.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32 USB Starter Kit II

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the SEGGER embOS basic demonstration.

Description

In this demonstration, there is one user task and one hardware interrupt. A user task is responsible for toggling the user LEDs to show that the RTOS and the application are up and running. The hardware interrupt is used by the RTOS to run the RTOS Tick. The internal core timer is used as the source for the hardware interrupt.

Once the demonstration is up and running a LED will toggle every 500 ms. This demonstration will show the RTOS running with the selected hardware.

usb

This section provides information on the supported demonstration boards, how to configure the hardware (if needed), and how to run the demonstration.

Description

Legal Disclaimer

The source code for this SEGGER embOS RTOS demonstration is not freely distributed. To obtain source code and the proper licensing agreement visit the SEGGER embOS website: https://www.segger.com/license-models.html.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the SEGGER embOS and MPLAB Harmony Graphics plus USB Library Demonstration.

Description

To build this project, you must open the `usb_embos.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/rtos/embos/usb`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`usb_embos.X`	`<install-dir>/apps/rtos/embos/usb/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mz_ef_sk	pic32mz_ef_sk	This configuration runs on the PIC32MZ EF Starter Kit.
pic32mx_usb_sk2	pic32mx_usb_sk2	This configuration runs on the PIC32 USB Starter Kit II.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32 USB Starter Kit II

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the SEGGER embOS and MPLAB Harmony USB Library demonstration.

Description

Power the board, open the demonstration with MPLAB X IDE, and then build and load the demonstration. A USB keyboard will be emulated when a micro USB cable is plugged in to the starter kit. Keystrokes, a-z and 1-9, will be sent when the push button SW3 is pressed. A user LED will toggle every 500 ms.

There are three tasks and four interrupts used in this application/system.

SystemUSBDeviceTask is the highest priority task and is responsible for getting all USB data queued up by the application ready for sending over the USB by calling the appropriate MPLAB Harmony USB stack function. ApplicationUSBDeviceTask is the next highest priority task. ApplicationUSBDeviceTask emulates the keyboard key presses when it detects that the push button SW3 is being pressed. ApplicationLEDblinkTask is the lowest priority task and toggles the user LED every 500 ms.

CPU core timer hardware interrupt is used by the RTOS to source the RTOS Tick. The Timer2 hardware interrupt is used to call the MPLAB Harmony Timer Driver function. USB general event interrupt is used to call the appropriate MPLAB Harmony USB driver function, which takes all USB data queued up by the application and physically writes it out to the USB hardware. The SYSCALL general exception is invoked by the RTOS and is used to process the task context switch routine.

TCP/IP Demonstrations

This section provides descriptions of the TCP/IP demonstrations.

MPLAB Harmony is available for download from the Microchip website by visiting: http://www.microchip.com/mplabharmony. Once you are on the site, click the Downloads tab to access the appropriate download for your operating system. For additional information on this demonstration, refer to the “Applications Help” section in the MPLAB Harmony Help.

Introduction

TCP/IP and Wi-Fi® Demonstration Applications Help

Description

Important!

1. The TCP/IP demonstration applications assume that IPv4 is enabled. If IPv4 is disabled in MHC, the `app.c` code will no longer build and needs to be updated to remove the IPv4 dependencies.
2. The Ethernet Controller requires a minimum clock frequency to be able to keep up with 100 Mbps traffic. Currently, this frequency must be at least 40 MHz for PIC32MX/PIC3MZ platforms. This is a minimum value, and depending on the system bus load, the actual running frequency may need to be higher than this.

TCP/IP Demonstrations

This section describes Microchip's TCP/IP Demonstration projects, including information about demonstration-hardware compatibility and also provides the information about how to configure and run the demonstrations.

Wi-Fi Demonstrations

This distribution package contains a variety of Wi-Fi-based firmware projects that demonstrate the capabilities of the MPLAB Harmony Wi-Fi services and TCP/IP Stack running on PIC32 devices. This section describes the hardware requirements and procedures to run these firmware projects on Microchip demonstration and development boards.

Wi-Fi Console Commands

This section describes the demonstration support commands available for the Wi-Fi Web Server and EasyConfig demonstrations.

Description

Both the Web Server and the EasyConfig demonstrations support Wi-FI Console commands, which enable control over the Wi-Fi settings.

Command: eraseconf

Parameters	Description
None.	Wi-Fi console command to erase saved Wi-Fi configuration in memory.

Command: iwconfig

Parameters	Description
[ssid <name>]	name: Specifies the name of the SSID (1-32 ASCII characters).
[mode <idle | managed>]	idle: Disconnected from the current configuration. managed: Connects in infrastructure mode to the currently set SSID.
[power <enable | disable>]	enable: Enables all Power-Saving features (PS_POLL). Will wake up to check for all types of traffic (unicast, multicast, and broadcast). disable: Disables any Power-Saving features. Will always be in an active power state.

[security <mode>]	mode: open/wep40/wep104/wpa/wpa2/pin/pbc. For example:
	`iwconfig security open`
	`iwconfig security wep40 <key>`
	`iwconfig security wep104 <key>`
	`iwconfig security wpa <key>`
	`iwconfig security wpa2 <key>`
	`iwconfig security pin <pin>`
	`iwconfig security pbc`

[scan]	Starts a Wi-Fi scan.
[scanget <scan_index>]	scan_index: Retrieves the scan result after the scan completes (1 - n).

Command: mac

Parameters	Description
None.	Wi-Fi console command to retrieve the MAC address of the Wi-Fi module.

Command: readconf

Parameters	Description
None.	Wi-Fi console command to read saved Wi-Fi configuration in memory.

Command: saveconf

Parameters	Description
None.	Wi-Fi console command to save Wi-Fi configuration to memory.

Demonstrations

Description of TCP/IP Stack Library Demonstration Application.

Description

PHY Driver Support

All of the PIC32MX and PIC32MZ projects that are part of the distribution and use the Microchip reference development boards are preconfigured with specific PHY Drivers. Where the board supports different PHY daughter boards, the default PHY could be changed. To use a different PHY for a specific board the following must be done:

1. Use the MHC to configure your project to use the correct PHY and make sure that both the correct PHY address and configuration flags are used for the particular PHY daughter board. The MII/RMII and I/O configuration flags for the PHY board should match the project configuration fuses.
2. Regenerate the project and make sure that the new PHY driver is selected for the configuration that you're using.

Alternatively, you can manually set up your project, as follows:

- The project should select the PHY driver that corresponds to the PHY Daughter Board (i.e, LAN8720, LAN8740, LAN9303, etc.) in use for the selected configuration
- Modify for the TCPIP_EMAC_PHY_ADDRESS and TCPIP_EMAC_PHY_CONFIG_FLAGS to have the correct PHY address (the PHY address for both the SMSC PHY Daughter Boards is usually zero, for example) and the configuration flags (MII/RMII, I/O pin configuration, etc.)
- Or update directly the tcpip_stack_init.c:: tcpipMACPIC32INTInitData structure to have the correct PHY address and the configuration flags
- Make sure that the configuration fuses are properly selected to match your hardware and PHY board
- Rebuild the project

berkeley_tcp_client

Before using this demonstration, please see the important notes in the *TCP/IP Demonstrations* > *Introduction*.

This configuration demonstrates creating an Internet client that uses the Berkeley API to create a TCP/IP connection to a web server.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Berkeley TCP Client Demonstration.

Description

To build this project, you must open the `berkeley_tcp_client.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/tcpip/berkeley_tcp_client`.



Warning

When using the Microchip Harmony Configurator (MHC), care must be taken when generating the code to *not* erase the USB descriptors in the `system_init.c` file.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`berkeley_tcp_client.X`	`<install-dir>/apps/tcpip/berkeley_tcp_client/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx_eth_sk	pic32mx_eth_sk	Demonstrates the Berkeley TCP Client on the PIC32 Ethernet Starter Kit.
pic32mx_eth_sk2	pic32mx_eth_sk2	Demonstrates the Berkeley TCP Client on the PIC32 Ethernet Starter Kit II.
pic32mz_ec_sk	pic32mz_ec_sk	Demonstrates the Berkeley TCP Client on the PIC32MZ EC Starter Kit.
pic32mz_ef_sk	pic32mz_ef_sk	Demonstrates the Berkeley TCP Client on the PIC32MZ EF Starter Kit.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 Ethernet Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32 Ethernet Starter Kit II

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ EC Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

This section provides instructions on how to build and run the demonstration.

Description

For PIC32M-based Starter Kits

1. Connect a USB cable from the computer to the micro-B USB connector on the bottom of the starter kit in use.
2. When the demonstration runs, it will create a virtual com with USB CDC device on the USB bus.
3. Open a standard terminal application on the computer (such as HyperTerminal or Tera Term).
4. Set the baud rate to 921600 baud in the terminal application.
5. Establish a connection between the router or switch with the PIC32M Starter Kit using the RJ45 connector.

For all hardware, ensure the router or switch is connected to the Internet.

There is only one command available in the demonstration from the serial port:

`openurl <url>` - The <url> argument must be a fully formed URL; for instance, http://www.microchip.com/

After that one command is input, the demonstration will make a DNS query, and then open a connection to the requested URL and perform a simple HTTP PUT command. The results will be sent to the serial port.

berkeley_tcp_server

Before using this demonstration, please see the important notes in the *TCP/IP Demonstrations* > *Introduction*.

This configuration demonstrates creating an Internet server that uses the Berkeley API to create a TCP/IP echo server on port 9764.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Berkeley TCP Server Demonstration.

Description

To build this project, you must open the `berkeley_tcp_server.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/tcpip/berkeley_tcp_server`.

Warning

When using the Microchip Harmony Configurator (MHC), care must be taken when generating the code to *not* erase the USB descriptors in the `system_init.c` file.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`berkeley_tcp_server.X`	`<install-dir>/apps/tcpip/berkeley_tcp_server/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx_eth_sk	pic32mx_eth_sk	Demonstrates the Berkeley TCP Server on the PIC32 Ethernet Starter Kit.
pic32mx_eth_sk2	pic32mx_eth_sk2	Demonstrates the Berkeley TCP Server on the PIC32 Ethernet Starter Kit II.
pic32mz_ec_sk	pic32mz_ec_sk	Demonstrates the Berkeley TCP Server on the PIC32MZ EC Starter Kit.
pic32mz_ef_sk	pic32mz_ef_sk	Demonstrates the Berkeley TCP Server on the PIC32MZ EF Starter Kit.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 Ethernet Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32 Ethernet Starter Kit II

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ EC Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

This section provides instructions on how to build and run the demonstration.

Description

For PIC32M-based Starter Kits

1. Connect a USB cable from the computer to the micro-B USB connector on the bottom of the starter kit in use.
2. When the demonstration runs, it will create a virtual com with USB CDC device on the USB bus.
3. Open a standard terminal application on the computer (such as HyperTerminal or Tera Term).
4. Set the baud rate to 921600 baud in the terminal application.
5. Establish a connection between the router or switch with the PIC32M Starter Kit using the RJ45 connector.

The demonstration does not offer any additional functionality through the serial port; however, the current IP can be checked. As soon as a valid IP has been assigned through DHCP to the demonstration, it is then ready to accept a TCP/IP connection on 9760. The demonstration will echo back everything it receives along the connection.

berkeley_udp_client

Before using this demonstration, please see the important notes in the *TCP/IP Demonstrations* > *Introduction*.

This configuration demonstrates creating an Internet client that uses the Berkeley API to create a UDP/IP connection to a specified port.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Berkeley UDP Client Demonstration.

Description

To build this project, you must open the `berkeley_udp_client.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/tcpip/berkeley_udp_client`.

Warning

When using the Microchip Harmony Configurator (MHC), care must be taken when generating the code to *not* erase the USB descriptors in the `system_init.c` file.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`berkeley_udp_client.X`	`<install-dir>/apps/tcpip/berkeley_udp_client/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx_eth_sk	pic32mx_eth_sk	Demonstrates the Berkeley UDP Client on the PIC32 Ethernet Starter Kit.
pic32mx_eth_sk2	pic32mx_eth_sk2	Demonstrates the Berkeley UDP Client on the PIC32 Ethernet Starter Kit II.
pic32mz_ec_sk	pic32mz_ec_sk	Demonstrates the Berkeley UDP Client on the PIC32MZ EC Starter Kit.
pic32mz_ef_sk	pic32mz_ef_sk	Demonstrates the Berkeley UDP Client on the PIC32MZ EF Starter Kit.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 Ethernet Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32 Ethernet Starter Kit II

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ EC Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

This section provides instructions on how to build and run the demonstration.

Description

For PIC32M-based Starter Kits

1. Connect a USB cable from the computer to the micro-B USB connector on the bottom of the starter kit in use.
2. When the demonstration runs, it will create a virtual com with USB CDC device on the USB bus.
3. Open a standard terminal application on the computer (such as HyperTerminal or Tera Term).
4. Set the baud rate to 921600 baud in the terminal application.
5. Establish a connection between the router or switch with the PIC32M Starter Kit using the RJ45 connector.

There are three sequential commands that can be used from the console:

- `setudppacketoptions <hostname> <port> <message>` - This command specifies where to send the UDP packet and what to have in the message
- `getudppacketoptions` - This command displays the current options
- `sendudppacket` - This command sends a UDP packet

After the `sendudppacket` command is input, the demonstration will make a DNS query to look up the host name and send a UDP packet to that host.

The output message will be received by the UDP server and by the UDP port that is configured by the command setudppacketoptions.

berkeley_udp_relay

Before using this demonstration, please see the important notes in the *TCP/IP Demonstrations* > *Introduction*.

This application demonstrates the use of multiple sockets for both sending and receiving. There are three different sub-functions of this application:

- UDP Relay, which accepts UDP packets on one socket, and sends the packets out on a different socket
- UDP Relay Client, which generates UDP traffic that is compatible with the UDP Relay Server
- UDP Relay Server, which receives and checks traffic for a packet count and reports is any packets are dropped

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Berkeley UDP Relay Demonstration.

Description

To build this project, you must open the `berkeley_udp_relay.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/tcpip/berkeley_udp_relay`.

Warning

When using the Microchip Harmony Configurator (MHC), care must be taken when generating the code to *not* erase the USB descriptors in the `system_init.c` file.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`berkeley_udp_relay.X`	`<install-dir>/apps/tcpip/berkeley_udp_relay/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx_eth_sk	pic32mx_eth_sk	Demonstrates the Berkeley UDP Relay on the PIC32 Ethernet Starter Kit.
pic32mx_eth_sk2	pic32mx_eth_sk2	Demonstrates the Berkeley UDP Relay on the PIC32 Ethernet Starter Kit II.
pic32mz_ec_sk	pic32mz_ec_sk	Demonstrates the Berkeley UDP Relay on the PIC32MZ EC Starter Kit.
pic32mz_ef_sk	pic32mz_ef_sk	Demonstrates the Berkeley UDP Relay on the PIC32MZ EF Starter Kit.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 Ethernet Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32 Ethernet Starter Kit II

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ EC Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

This section provides instructions on how to build and run the demonstration.

Description

For PIC32M-based Starter Kits

1. Connect a USB cable from the computer to the micro-B USB connector on the bottom of the starter kit in use.
2. When the demonstration runs, it will create a virtual com with USB CDC device on the USB bus.
3. Open a standard terminal application on the computer (such as HyperTerminal or Tera Term).
4. Set the baud rate to 921600 baud in the terminal application.
5. Establish a connection between the router or switch with the PIC32M Starter Kit using the RJ45 connector.

This application demonstrates a simple UDP packet relay. Functionality has also been put in place to generate packets and to receive packets on the same device. IPv6 has not been tested.

Demonstration Commands

There is are several different commands available in the demonstration from the console port:

General Application Commands:

- current - Displays the current configuration
- start - Starts the packet relay service
- stop - Stops the packet relay service
- reportinterval <seconds> - Sets the interval between reports to the console

Relay Service Configuration:

- relayhost <host name> - Sets the host to which packets are to be relayed
- relayport <port number> - Sets the port to which packets are to be relayed
- ipv4port <port number> - Sets the IPv4 port that the relay server will listen to for packets to relay
- ipv6port <port number> - Sets the IPv6 port that the relay server will listen to for packets to relay

Relay Client Configuration and Commands:

- relayclienthost <host name> - Sets the host to which packets are to be sent
- relayclientport <port number> - Sets the port to which packets are to be sent
- relayclientiter <number> - The number of packets to generate
- relayclientstart - Starts the relay client. This command must be used after the general application start. After a start is called, and the first packet is received by either the relay or the relay server, periodic updates will be sent to the console with information about the number of packets and bytes received.

berkeley_udp_server

Before using this demonstration, please see the important notes in the *TCP/IP Demonstrations* > *Introduction*.

This configuration demonstrates creating an Internet server that uses the Berkeley API to create a UDP/IP echo server on port 9764.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Berkeley UDP Server Demonstration.

Description

To build this project, you must open the `berkeley_udp_server.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/tcpip/berkeley_udp_server`.

Warning

When using the Microchip Harmony Configurator (MHC), care must be taken when generating the code to *not* erase the USB descriptors in the `system_init.c` file.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`berkeley_udp_server.X`	`<install-dir>/apps/tcpip/berkeley_udp_server/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx_eth_sk	pic32mx_eth_sk	Demonstrates the Berkeley UDP Server on the PIC32 Ethernet Starter Kit.
pic32mx_eth_sk2	pic32mx_eth_sk2	Demonstrates the Berkeley UDP Server on the PIC32 Ethernet Starter Kit II.
pic32mz_ec_sk	pic32mz_ec_sk	Demonstrates the Berkeley UDP Server on the PIC32MZ EC Starter Kit.
pic32mz_ef_sk	pic32mz_ef_sk	Demonstrates the Berkeley UDP Server on the PIC32MZ EF Starter Kit.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 Ethernet Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32 Ethernet Starter Kit II

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ EC Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

This section provides instructions on how to build and run the demonstration.

Description

For PIC32M-based Starter Kits

1. Connect a USB cable from the computer to the micro-B USB connector on the bottom of the starter kit in use.
2. When the demonstration runs, it will create a virtual com with USB CDC device on the USB bus.
3. Open a standard terminal application on the computer (such as HyperTerminal or Tera Term).
4. Set the baud rate to 921600 baud in the terminal application.
5. Establish a connection between the router or switch with the PIC32M Starter Kit using the RJ45 connector.

The demonstration does not offer any additional functionality through the serial port; however, the current IP can be checked. As soon as a valid IP has been assigned through DHCP to the demonstration, it is then ready to accept a UDP/IP connection on 9760. The demonstration will echo back everything it receives along the connection.

snmpv3_nvm_mpfs

Before using this demonstration, please see the important notes in the *TCP/IP Demonstrations* > *Introduction*.

SNMPv3 NVM MPFS demonstration.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the SNMPv3 NVM MPFS Demonstration.

Description

To build this project, you must open the `snmpv2_nvm_mpfs.X` project in MPLAB X IDE, and then select the desired configuration.

The Non-Volatile Memory (NVM) Microchip Proprietary File System (MPFS) has the `snmp.bib` file along with other web page files stored in

internal Flash and are accessed through the MPFS API.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/tcpip/snmpv3_nvm_mpfs`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`snmpv3_nvm_mpfs.X`	`<install-dir>/apps/tcpip/snmpv3_nvm_mpfs/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx_eth_sk	pic32mx_eth_sk2	Demonstrates the SNMPv3 NVM MPFS on the PIC32 Ethernet Starter Kit II in Interrupt mode and dynamic operation.
pic32mz_ef_sk	pic32mz_ef_sk	Demonstrates the SNMPv3 NVM MPFS on the PIC32MZ EF Starter Kit in Interrupt mode and dynamic operation.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32 Ethernet Starter Kit II

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

This section provides instructions on how to build and run the demonstration.

Description

Use the following procedure to run the demonstration:

1. Load the demonstration project into MPLAB X IDE.
2. Connect the mini-B debugger port on-board the starter kit in use to a USB port on the development computer using the USB cable provided in the kit.
3. Connect the RJ-45 Ethernet port on the starter kit board to a network hub or an Ethernet port on the development computer using the Ethernet patch cord provided in the kit.
4. Build, download, and run the demonstration project on the target board.
5. A SNMP and SNMPv3 server is hosted by the demonstration application.
6. Run tcpip_discoverer to get the IPv4 and IPv6 address for the board.
7. Open a SNMP manager (iREASONING SNMP manager is recommended) and configure the IPv4 or IPv6 address.

Notes:

1. Refer to the iREASONING Networks MIB Browser section in the Third-Party help for complete details on using and configuring the application using the iREASONING SNMP Manager.

For PIC32M-based Starter Kits

1. Connect a USB cable from the computer to the micro-B USB connector on the bottom of the starter kit in use.
2. When the demonstration runs, it will create a virtual com with USB CDC device on the USB bus.
3. Open a standard terminal application on the computer (such as HyperTerminal or Tera Term).
4. Set the baud rate to 921600 baud in the terminal application.

SNMP MIB Browser

Several SNMP MIB browsers are available. Users can also install a customized MIB browser specific to their application.

SNMP Get, GetNext, GetBulk, Set request and response are working as expected for SNMP v1/v2/v3 versions.

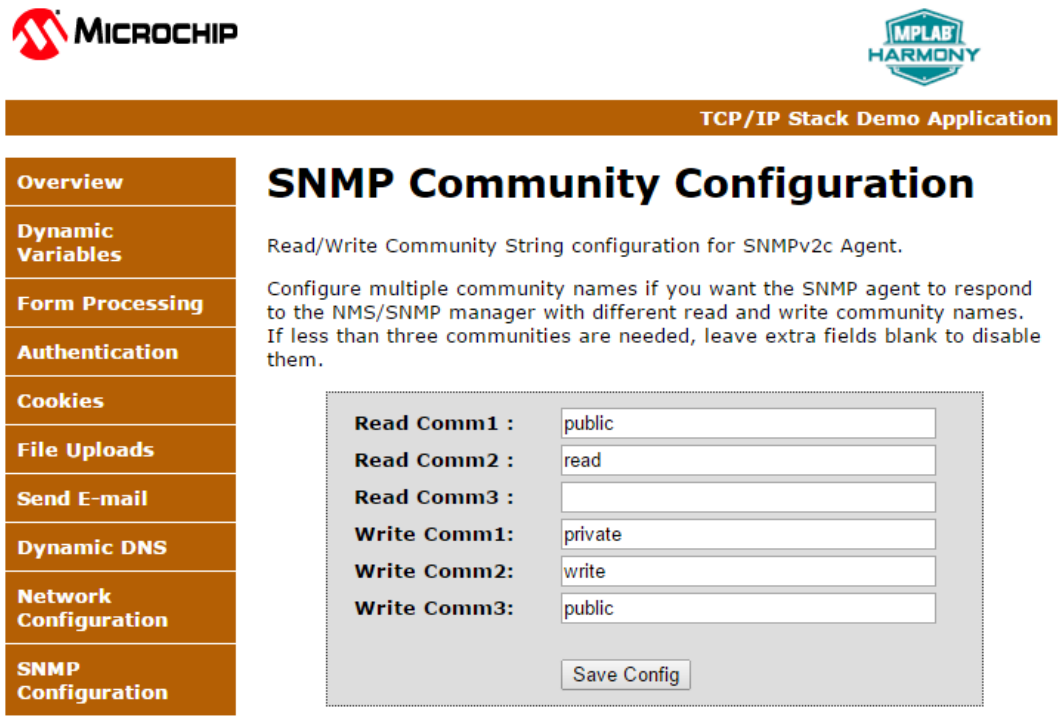
- For SNMP v2c , the Agent is configured with three Read communities ("public", "read", " ") and three Write communities ("private","write","public").
- For SNMP v3, the Agent is configured as per the following table:

Type	USER 1	USER 2	USER 3
USM User	microchip	SnmpAdmin	root
Security Level	auth, priv	auth, no priv	no auth, no priv
Auth Algorithm	MD5	SHA1	N/A
Auth Password	auth12345	ChandlerUS	N/A
Privacy Algorithm	AES	N/A	N/A
Privacy Password	priv12345	N/A	N/A

The Microchip SNMP Stack supports both TRAP version 1 and TRAP version 2. This demonstration trap output is a multi-varbind SNMPv3 TRAP version 2. Users may be required to configure the Trap receiver as per the SNMP browser selection.

HTTP Configuration for SNMPv2c Community

It is possible to dynamically configure the Read and Write community names through the SNMP Configuration web page. Access the web page using http://mchpboard_e/mpfsupload or http://<Board IP address>(for IPv6 it should be http://<Ipv6 address>:80/index.html), and then access the SNMP Configuration web page through the navigation bar. Use "admin" for the username and "microchip" for the password.



Warning

When using the Microchip Harmony Configurator (MHC), care must be taken when generating the code to *not* erase the USB descriptors in the `system_init.c` file.

snmpv3_sdcard_fatfs

Before using this demonstration, please see the important notes in the *TCP/IP Demonstrations* > *Introduction*.

SNMPv3 SD Card FAT File System demonstration.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the SNMPv3 SD Card FAT FS Demonstration.

Description

To build this project, you must open the `snmpv3_sdcard_fatfs.X` project in MPLAB X IDE, and then select the desired configuration.

The SD Card FAT FS has the `snmp.bib` file with other web pages stored in an external SD card and is accessed through a FAT FS API.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/tcpip/snmpv3_sdcard_fatfs`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`snmpv3_sdcard_fatfs.X`	`<install-dir>/apps/tcpip/snmpv3_sdcard_fatfs/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx_eth_sk2_sd_mmc_pictail	pic32mx_eth_sk2	Demonstrates the access of a SNMP file on a microSD card through the FAT file system on the PIC32 Ethernet Starter Kit using the Starter Kit I/O Expansion Board with the PICtail daughter board for SD and MMC cards. The demonstration runs in Interrupt mode and dynamic operation.
pic32mz_ef_sk	pic32mz_ef_sk+meb2	Demonstrates the access of a SNMP file on a microSD card through the FAT file system on the PIC32MZ EF Starter Kit and the MEB II board combination. The demonstration runs in Interrupt mode and dynamic operation.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MZ EC Starter Kit or PIC32MZ EF Starter Kit with the MEB II

1. Connect the starter kit to the application board connector on the MEB II.
2. Make sure a microSD card is formatted and loaded with the `snmp.bib` file along with the web pages provided in the `<install-dir>/apps/tcpip/web_server_sdcard_fatfs/firmware/src/web_pages` folder.
3. Insert the microSD card with the web pages into the microSD card slot (J8) on the MEB II.

PIC32 Ethernet Starter Kit II with the Starter Kit Expansion Board

1. Connect the PIC32 Ethernet Starter Kit II to the I/O expansion board.
2. Make sure a SD card is formatted and loaded with the `snmp.bib` file along with the web pages provided within the `<install-dir>apps/tcpip/snmpv3_sdcard_fatfs/firmware/src/web_pages` folder.
3. Insert the SD card into the PICtail Daughter Board for SD and MMC cards with the `snmp.bib` file along with web pages into the SPI1 slot (J4 - starts slot count from 1) of the PIC32 I/O Expansion Board.

Note: The SD card on the PICtail daughter board should face the PIC32 Ethernet Starter Kit II.

Running the Demonstration

This section provides instructions on how to build and run the demonstration.

Description

Please refer to the Running the Demonstration section for the snmpv3_nvm_mpfs configuration, as the process is the same for this configuration.

Note: Ensure that the SD card with the `snmp.bib` file and the Web pages is inserted as detailed in Configuring the Hardware.

tcpip_tcp_client

Before using this demonstration, please see the important notes in the *TCP/IP Demonstrations* > *Introduction*.

This configuration demonstrates creating an Internet client that uses the MPLAB Harmony TCP API to create a TCP/IP connection to a web server.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the TCP/IP TCP Client Demonstration.

Description

To build this project, you must open the `tcpip_tcp_client.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/tcpip/tcpip_tcp_client`.

Warning When using the Microchip Harmony Configurator (MHC), care must be taken when generating the code to *not* erase the USB descriptors in the `system_init.c` file.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`tcpip_tcp_client.X`	`<install-dir>/apps/tcpip/tcpip_tcp_client/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx_eth_sk2	pic32mx_eth_sk2	Demonstrates the TCP/IP TCP Client on the PIC32 Ethernet Starter Kit II.
pic32mz_ef_sk	pic32mz_ef_sk	Demonstrates the TCP/IP TCP Client on the PIC32MZ EF Starter Kit.
pic32mx_eth_sk2_enc28j60	pic32mx_eth_sk2	Demonstrates the TCP/IP TCP Client on the PIC32 Ethernet Starter Kit II connected to the 10 Mbps Ethernet PICtail Plus Daughter Board and Starter Kit I/O Expansion Board using the ENC28J60 Driver Library.
pic32mx_eth_sk2_encx24j600	pic32mx_eth_sk2	Demonstrates the TCP/IP TCP Client on the PIC32 Ethernet Starter Kit II connected to the Fast 100Mbps Ethernet PICtail Plus Daughter Board and Starter Kit I/O Expansion Board using the ENCx24J600 Driver Library.
pic32mz_da_sk_intddr	pic32mz_da_sk_intddr	Demonstrates the TCP/IP TCP Client on the PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 Ethernet Starter Kit II

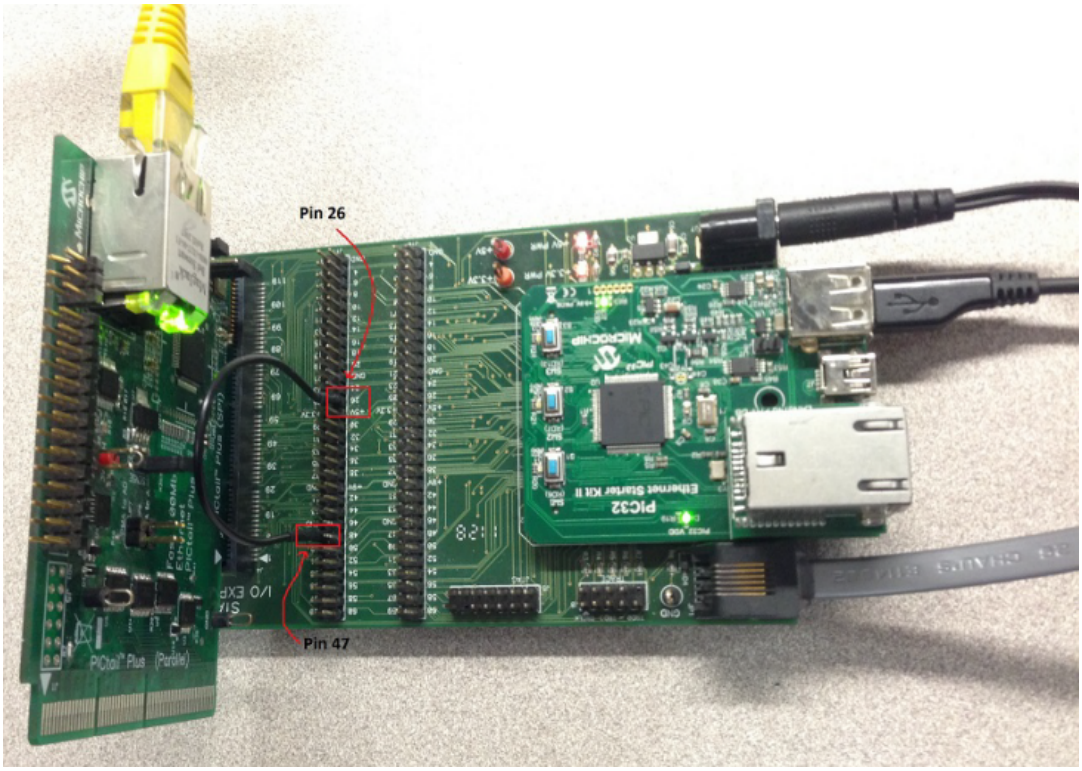
No hardware related configuration or jumper setting changes are necessary.

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32 Ethernet Starter Kit II, Fast 100Mbps Ethernet PICtail Plus Daughter Board, and Starter Kit I/O Expansion Board

The Fast 100Mbps Ethernet PICtail Plus Daughter Board is connected to J4 on the Starter Kit I/O Expansion Board board by sliding the J2 PICtail Plus (SPI) card edge into the top of the connector so that the white arrows on the two boards line up. The PICtail daughter board is inserted so that it uses SPI1. Pins 26 and 47 on J11 need to be jumpered to allow the CS line to be controlled by the PIC32. ThePIC32 Ethernet Starter Kit II is connected to J1 on the Starter Kit I/O Expansion board. Please refer to the following figure for more detail.



PIC32 Ethernet Starter Kit II, Ethernet PICtail Plus Daughter Board, and Starter Kit I/O Expansion Board

The 10 Mbps Ethernet PICtail Plus Daughter Board is connected to J4 on the Starter Kit I/O Expansion Board board by sliding the J2 PICtail Plus (SPI) card edge into the top of the connector. The PICtail daughter board is inserted so that it uses SPI1. Pins 26 and 47 on J11 need to be jumpered to allow the CS line to be controlled by the PIC32. The PIC32 Ethernet Starter kit II is connected to J1 on the Starter Kit I/O Expansion board.

Using the previous figures as a reference, replace the Fast 100 Mbps Ethernet PICtail Plus Daughter Board with the Ethernet PICtail Plus Daughter Board for this configuration. The RJ45 of the PICtail should be towards the edge connectors of the Starter Kit I/O Expansion Board.

Running the Demonstration

This section provides instructions on how to build and run the demonstration.

Description

For PIC32M-based Starter Kits

1. Connect a USB cable from the computer to the micro-B USB connector on the bottom of the starter kit in use.
2. When the demonstration runs, it will create a virtual com with USB CDC device on the USB bus.
3. Open a standard terminal application on the computer (such as HyperTerminal or Tera Term).
4. Set the baud rate to 921600 baud in the terminal application.
5. Establish a connection between the router or switch with the PIC32M Starter Kit using the RJ45 connector.

For all hardware, ensure that the router or switch is connected to the Internet.

There is only one command available in the demonstration from the serial port:

`openurl <url>` - The <url> argument must be a fully formed URL; for instance, http://www.microchip.com/.

After that one command is input, the demonstration will make a DNS query, and then open a connection to the requested URL and perform a simple HTTP PUT command. The results will be sent to the serial port.

tcpip_tcp_client_server

Before using this demonstration, please see the important notes in the *TCP/IP Demonstrations* > *Introduction*.

This configuration demonstrates creating an Internet client and an Internet server that uses the MPLAB Harmony TCP API. This demonstration is a combination of the TCP/IP Client and TCP/IP Server application.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the TCP/IP TCP Client Server Demonstration.

Description

To build this project, you must open the `tcpip_tcp_client_server.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/tcpip/tcpip_tcp_client_server`.

Warning

When using the Microchip Harmony Configurator (MHC), care must be taken when generating the code to *not* erase the USB descriptors in the `system_init.c` file.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`tcpip_tcp_client_server.X`	`<install-dir>/apps/tcpip/tcpip_tcp_client_server/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx_eth_sk2	pic32mx_eth_sk2	Demonstrates the TCP/IP TCP Client Server on the PIC32 Ethernet Starter Kit II.
pic32mz_ef_sk	pic32mz_ef_sk	Demonstrates the TCP/IP TCP Client Server on the PIC32MZ EF Starter Kit.
pic32mz_ef_curiosity	pic32mz_ef_curiosity	Demonstrates the TCP/IP TCP Client Server on the PIC32MZ EF Curiosity Development Board, with the PIC32MZ2048EFM100 microcontroller. This configuration is generated for standalone mode. All necessary files are copied under its configuration folder, and so it can be built and run without the MPLAB Harmony framework.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 Ethernet Starter Kit II

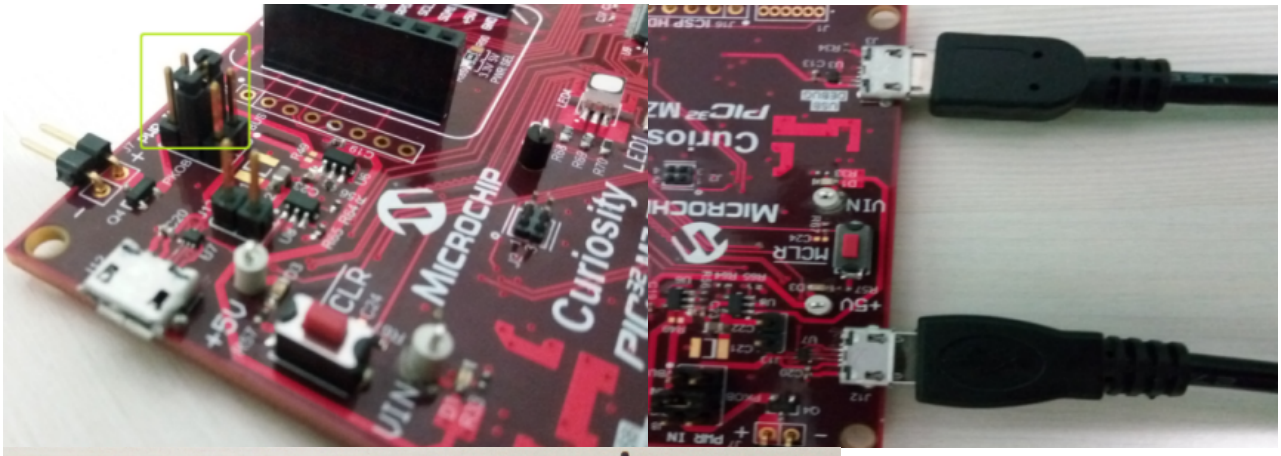
No hardware related configuration or jumper setting changes are necessary.

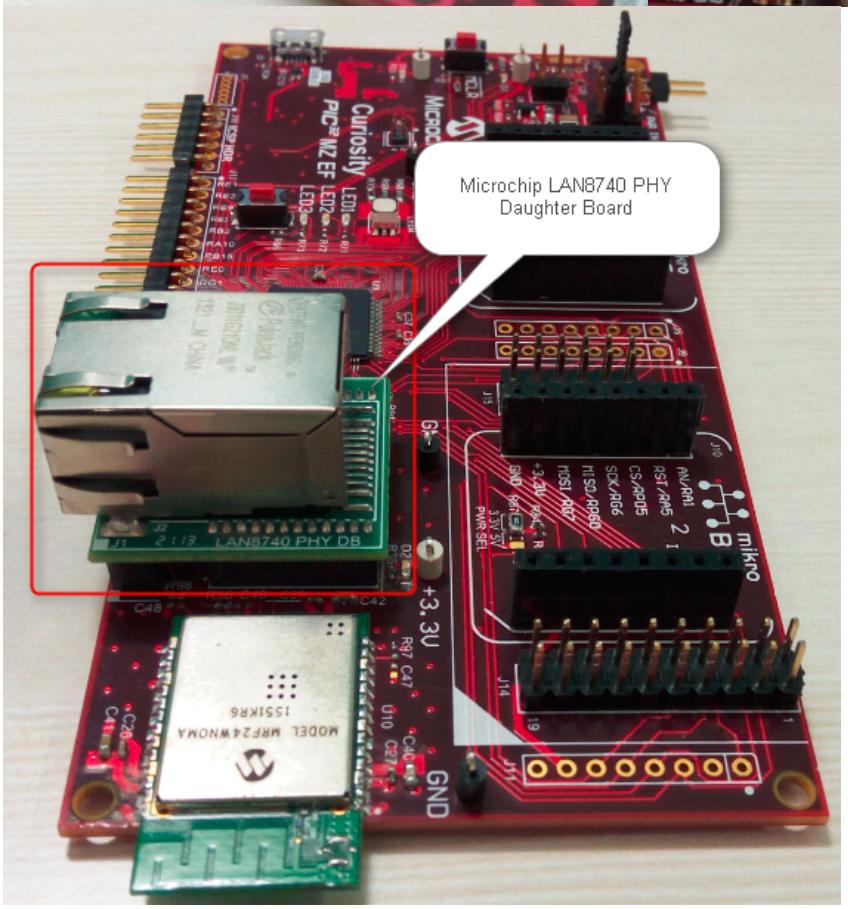
PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ EF Curiosity Development Board

1. Ensure that a jumper is placed at 4-3 on J8, to select supply from debug USB connector.
2. Power the PIC32MZ EF Curiosity Development Board from a Host PC through a Type-A male to micro-B USB cable connected to micro-B port (J3).
3. Ensure that jumper is not present in the J13 header to use the Curiosity board in device mode.
4. Plug in a USB cable with a micro-B type connector to Micro-B port (J12), and plug the other end into your computer.
5. Ensure that you have a LAN8740 Ethernet PHY DB installed on the PIC32MZ EF Curiosity Development Board (header J18).



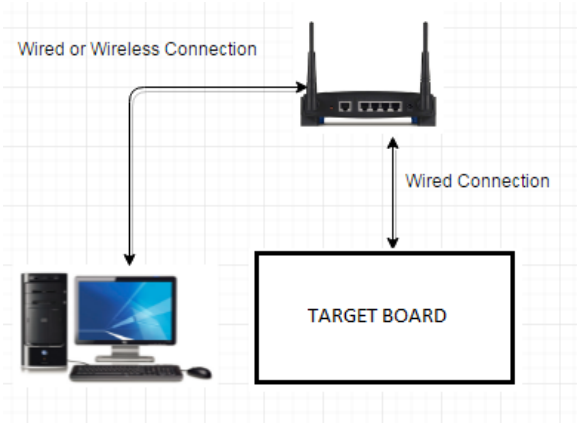


Running the Demonstration

This section provides instructions on how to build and run the demonstration.

Description

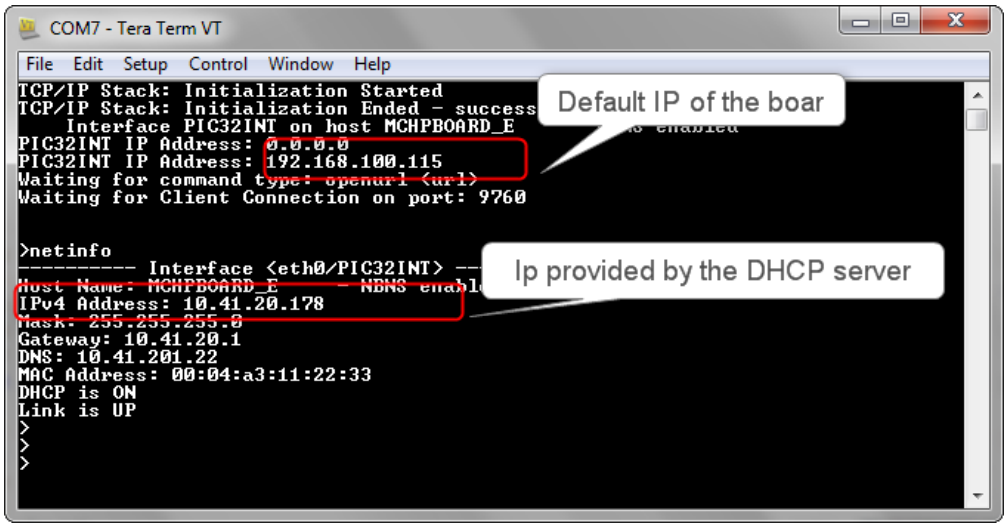
For the purpose of this demo, both the Target board and the Host PC should be in the same network. The host PC can be connected to a router via an Ethernet cable or Wi-Fi. The target board should be connected to the router via an Ethernet cable. Please refer to the following connection diagram.



For PIC32M-based Starter Kits

1. Connect a USB cable from the computer to the micro-B USB connector on the bottom of the starter kit in use.
2. When the demonstration runs, it will create a virtual com with USB CDC device on the USB bus.
3. Open a standard terminal application on the computer (such as HyperTerminal or Tera Term).
4. Set the baud rate to 921600 baud in the terminal application.
5. Establish a connection between the router or switch with the PIC32M Starter Kit using the RJ45 connector.

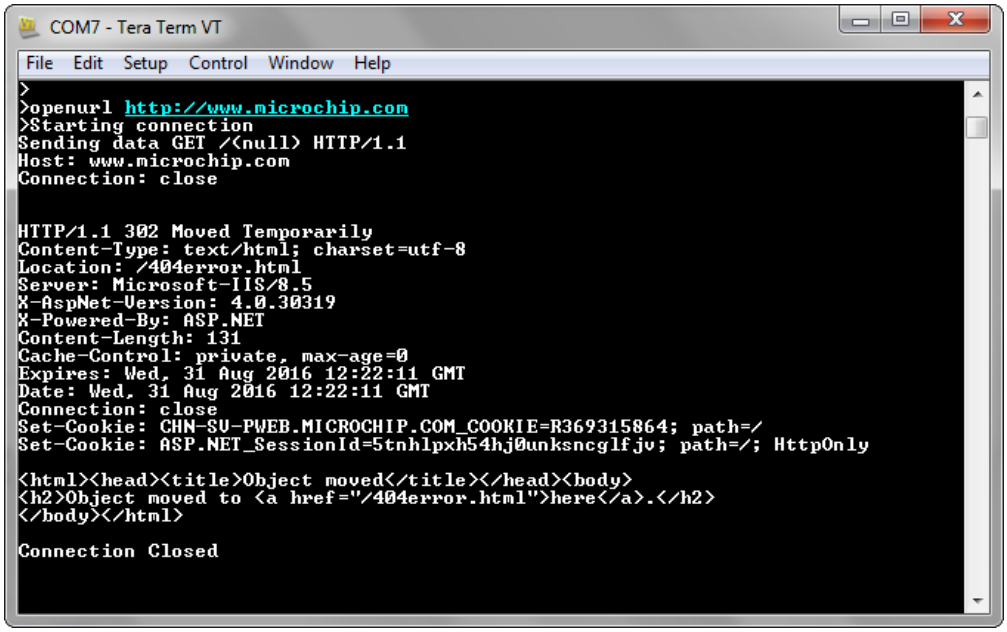
For all hardware, ensure that the router or switch is connected to the Internet.



There is only one command available in the demonstration from the serial port:

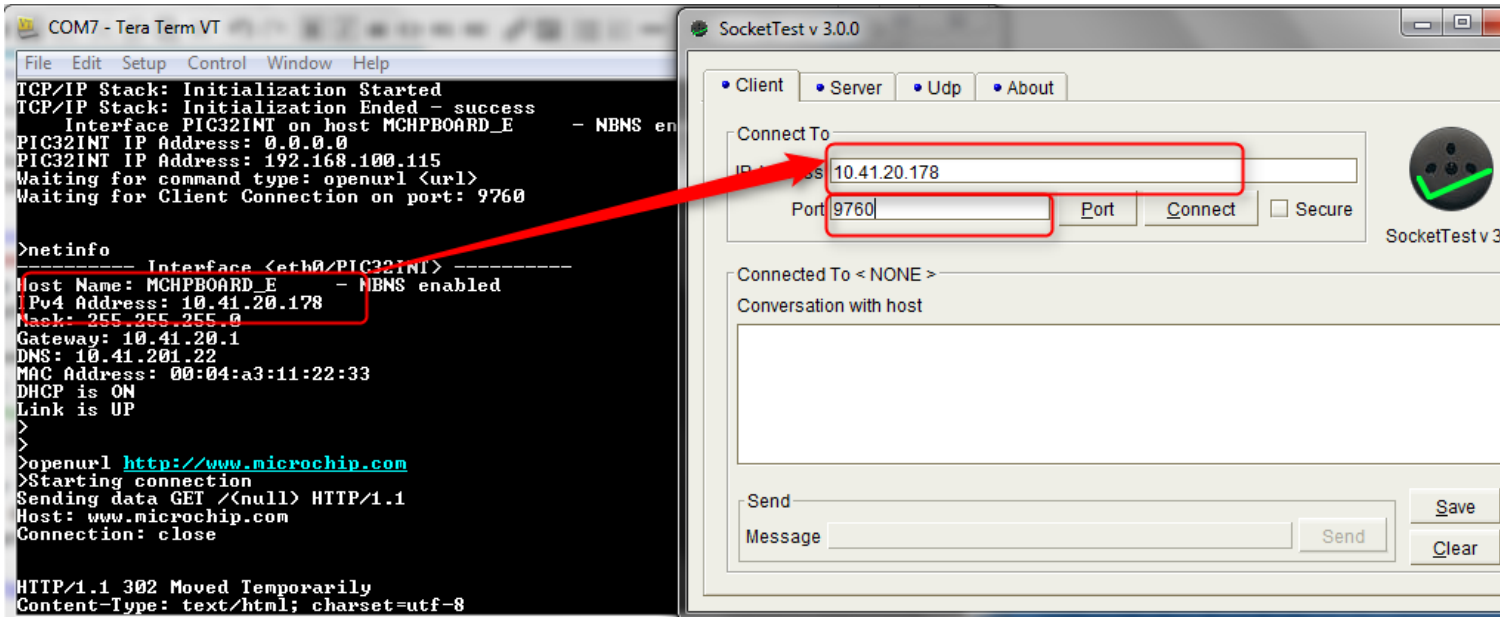
`openurl <url>` - The <url> argument must be a fully formed URL; for instance, http://www.microchip.com/

After that one command is input, the demonstration will make a DNS query, and then open a connection to the requested URL and perform a simple HTTP PUT command. The results will be sent to the serial port.



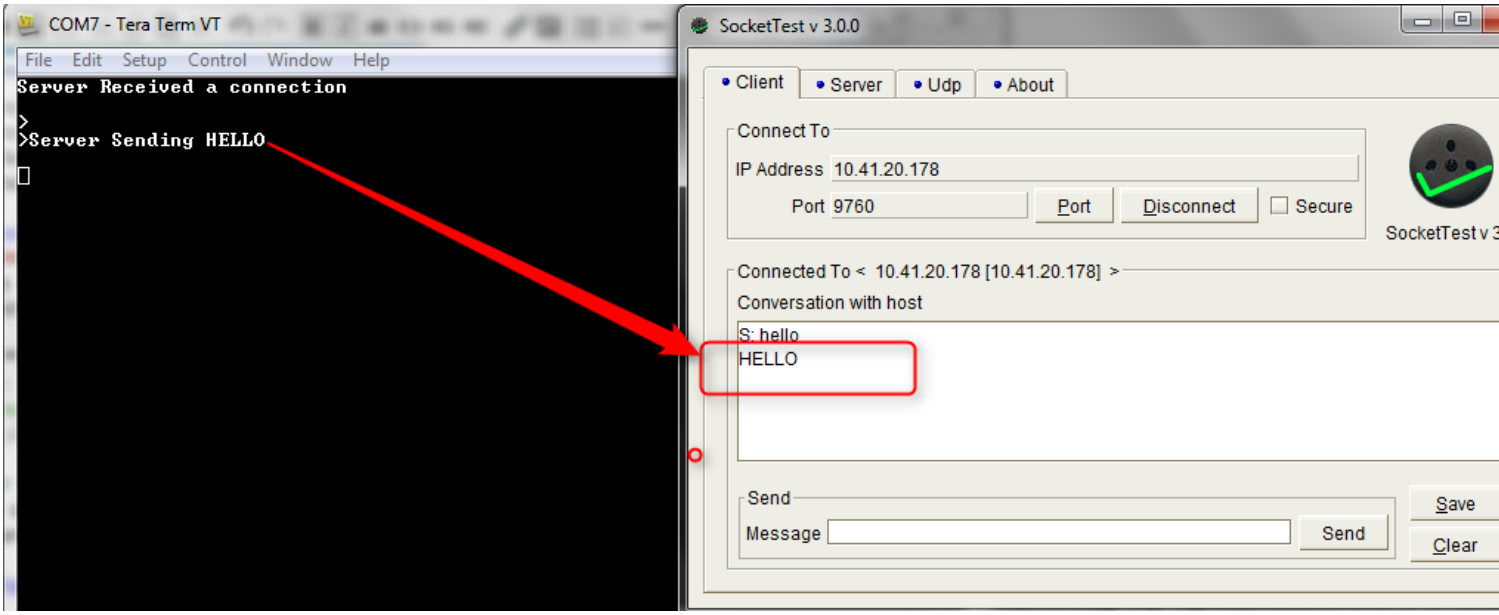
To test the Server part of the demo, we require a program that acts as a TCP/IP client. In this demonstration, we use the program, SocketTest (http://sockettest.sourceforge.net/). This demonstration has been tested with SocketTest v3.0.

1. Open the SocketTest software and set the configuration as shown in the following figure.



2. Press the Connect button on the SocketTest software after setting the configuration. The serial terminal indicates that the connection has been established.

3. Type any message in the message box of the SocketTest program, and press the Send button. The Server running on the Curiosity development board will echo back the message to the SocketTest program.



tcpip_tcp_server

Before using this demonstration, please see the important notes in the *TCP/IP Demonstrations* > *Introduction*.

This configuration demonstrates creating an Internet server that uses the MPLAB Harmony TCP API to create a TCP/IP echo server on port 9764.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the TCP/IP TCP Server Demonstration.

Description

To build this project, you must open the `tcpip_tcp_server.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/tcpip/tcpip_tcp_server`.

Warning

When using the Microchip Harmony Configurator (MHC), care must be taken when generating the code to *not* erase the USB descriptors in the `system_init.c` file.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`tcpip_tcp_server.X`	`<install-dir>/apps/tcpip/tcpip_tcp_server/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx_eth_sk2	pic32mx_eth_sk2	Demonstrates the TCP/IP TCP Server on the PIC32 Ethernet Starter Kit II.
pic32mz_ef_sk	pic32mz_ef_sk	Demonstrates the TCP/IP TCP Server on the PIC32MZ EF Starter Kit.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 Ethernet Starter Kit II

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

This section provides instructions on how to build and run the demonstration.

Description

For PIC32M-based Starter Kits

1. Connect a USB cable from the computer to the micro-B USB connector on the bottom of the starter kit in use.
2. When the demonstration runs, it will create a virtual com with USB CDC device on the USB bus.
3. Open a standard terminal application on the computer (such as HyperTerminal or Tera Term).
4. Set the baud rate to 921600 baud in the terminal application.
5. Establish a connection between the router or switch with the PIC32M Starter Kit using the RJ45 connector.

The demonstration does not offer any additional functionality through the serial port; however, the current IP can be checked. As soon as a valid IP has been assigned through DHCP to the demonstration, it is then ready to accept a TCP/IP connection on 9764. The demonstration will echo back everything it receives along the connection.

tcpip_udp_client

Before using this demonstration, please see the important notes in the *TCP/IP Demonstrations* > *Introduction*.

This configuration demonstrates creating an Internet client that uses the MPLAB Harmony UDP API to create a UDP/IP connection to a specified port.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the TCP/IP UDP Client Demonstration.

Description

To build this project, you must open the `tcpip_udp_client.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/tcpip/tcpip_udp_client`.

Warning

When using the Microchip Harmony Configurator (MHC), care must be taken when generating the code to *not* erase the USB descriptors in the `system_init.c` file.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`tcpip_udp_client.X`	`<install-dir>/apps/tcpip/tcpip_udp_client/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx_eth_sk2	pic32mx_eth_sk2	Demonstrates the TCP/IP UDP Client on the PIC32 Ethernet Starter Kit II.
pic32mz_ef_sk	pic32mz_ef_sk	Demonstrates the TCP/IP UDP Client on the PIC32MZ EF Starter Kit.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 Ethernet Starter Kit II

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

This section provides instructions on how to build and run the demonstration.

Description

For PIC32M-based Starter Kits

1. Connect a USB cable from the computer to the micro-B USB connector on the bottom of the starter kit in use.
2. When the demonstration runs, it will create a virtual com with USB CDC device on the USB bus.
3. Open a standard terminal application on the computer (such as HyperTerminal or Tera Term).
4. Set the baud rate to 921600 baud in the terminal application.
5. Establish a connection between the router or switch with the PIC32M Starter Kit using the RJ45 connector.

Ensure a UDP Server is running on the network that can respond to client requests.

There are three commands that can be used from the console:

- `setudppacketoptions <hostname> <port> <message>` - This command specifies where to send the UDP packet and what to have in the message
- `getudppacketoptions` - This command displays the current options
- `sendudppacket` - This command sends a UDP packet

After the `sendudppacket` command is input, the demonstration will make a DNS query to look up the host name and send a UDP packet to that host.

tcpip_udp_client_server

Before using this demonstration, please see the important notes in the *TCP/IP Demonstrations* > *Introduction*.

This configuration demonstrates creating an Internet client and an Internet server that uses the MPLAB Harmony UDP API. This demonstration shows how the UDP/IP loopback works, and is a combination of the TCP/IP UDP Client and TCP/IP UDP Server application.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the TCP/IP UDP Client Server Demonstration.

Description

To build this project, you must open the `tcpip_udp_client_server.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/tcpip/tcpip_udp_client_server`.

Warning

When using the Microchip Harmony Configurator (MHC), care must be taken when generating the code to *not* erase the USB descriptors in the `system_init.c` file.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`tcpip_udp_client_server.X`	`<install-dir>/apps/tcpip/tcpip_udp_client_server/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx_eth_sk2	pic32mx_eth_sk2	Demonstrates the TCP/IP UDP Client Server on the PIC32 Ethernet Starter Kit II.
pic32mz_ef_sk	pic32mz_ef_sk	Demonstrates the TCP/IP UDP Client Server on the PIC32MZ EF Starter Kit.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 Ethernet Starter Kit II

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

This section provides instructions on how to build and run the demonstration.

Description

For PIC32M-based Starter Kits

1. Connect a USB cable from the computer to the micro-B USB connector on the bottom of the starter kit in use.
2. When the demonstration runs, it will create a virtual com with USB CDC device on the USB bus.
3. Open a standard terminal application on the computer (such as HyperTerminal or Tera Term).
4. Set the baud rate to 921600 baud in the terminal application.
5. Establish a connection between the router or switch with the PIC32M Starter Kit using the RJ45 connector.

There are three commands that can be used from the console:

- `setudppacketoptions <hostname> <port> <message>` - This command specifies where to send the UDP packet and what to have in the message
- `getudppacketoptions` - This command displays the current options
- `sendudppacket` - This command sends a UDP packet

After the `sendudppacket` command is input, the demonstration will make a DNS query to look up the host name and send a UDP packet to that host.

The UDP Client can be configured to send UDP data to the host, configured using the commands previously described.

The UDP server in this demonstration waits for the client connection and data at port 9760.

As the server receives the data from the external UDP client, the data is shared to the UDP Client to transmit back to the external UDP Server.

The data received over UDP by the server is looped backed using the UDP Client.

tcpip_udp_server

Before using this demonstration, please see the important notes in the *TCP/IP Demonstrations* > *Introduction*.

This configuration demonstrates creating an Internet server that uses the MPLAB Harmony UDP API to create a UDP/IP echo server on port 9760.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the TCP/IP UDP Server Demonstration.

Description

To build this project, you must open the `tcpip_udp_server.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/tcpip/tcpip_udp_server`.

Warning

When using the Microchip Harmony Configurator (MHC), care must be taken when generating the code to *not* erase the USB descriptors in the `system_init.c` file.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`tcpip_udp_server.X`	`<install-dir>/apps/tcpip/tcpip_udp_server/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx_eth_sk2	pic32mx_eth_sk2	Demonstrates the TCP/IP UDP Server on the PIC32 Ethernet Starter Kit II.
pic32mz_ef_sk	pic32mz_ef_sk	Demonstrates the TCP/IP UDP Server on the PIC32MZ EF Starter Kit.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 Ethernet Starter Kit II

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

This section provides instructions on how to build and run the demonstration.

Description

For PIC32M-based Starter Kits

1. Connect a USB cable from the computer to the micro-B USB connector on the bottom of the starter kit in use.
2. When the demonstration runs, it will create a virtual com with USB CDC device on the USB bus.
3. Open a standard terminal application on the computer (such as HyperTerminal or Tera Term).
4. Set the baud rate to 921600 baud in the terminal application.
5. Establish a connection between the router or switch with the PIC32M Starter Kit using the RJ45 connector.

The demonstration does not offer any additional functionality through the serial port; however, the current IP can be checked. As soon as a valid IP has been assigned through DHCP to the demonstration, it is then ready to accept a UDP/IP connection on 9760. The demonstration will echo back everything it receives along the connection.

web_net_server_nvm_mpfs

Before using this demonstration, please see the important notes in the *TCP/IP Demonstrations* > *Introduction*.

Web Net Server Non-volatile Memory (NVM) MPFS TCP/IP demonstration.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the TCP/IP Web Net Server Demonstration.

Description

To build this project, you must open the `pic32_eth_web_server.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/tcpip/web_net_server_nvm_mpfs`.

Warning

When using the Microchip Harmony Configurator (MHC), care must be taken when generating the code to *not* erase the USB descriptors in the `system_init.c` file.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`pic32_eth_web_server.X`	`<install-dir>/apps/tcpip/web_net_server_nvm_mpfs/firmware`
`pic32_eth_wifi_web_server.X`	`<install-dir>/apps/tcpip/web_net_server_nvm_mpfs/firmware`
`pic32_wifi_web_server.X`	`<install-dir>/apps/tcpip/web_net_server_nvm_mpfs/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx_eth_sk2	pic32mx_eth_sk2	Demonstrates the web server hosted on internal Flash through the Microchip Proprietary File System (MPFS) on the PIC32 Ethernet Starter Kit II.
pic32mz_ef_sk	pic32mz_ef_sk	Demonstrates the web server hosted on internal Flash through the MPFS on the PIC32MZ EF Starter Kit.
pic32mz_ef_sk_16b	pic32mz_ef_sk	Demonstrates the web net server hosted on internal Flash through the MPFS on the PIC32MZ EF Starter Kit.
pic32mz_ef_sk_ioexp_winc_freertos	pic32mz_ef_sk	Demonstrates the web server hosted on internal Flash through the MPFS on the PIC32MZ EF Starter Kit and the WINC1500 PICtail/PICtail Plus Daughter Board.
pic32mx795_pim_e16_winc_freertos	pic32mx795_pim+e16	Demonstrates the web net server hosted on internal Flash through the MPFS on the PIC32MX795F512L CAN-USB PIM with the Explorer 16/32 Development Board and the WINC1500 PICtail/PICtail Plus Daughter Board.
pic32mx795_pim_e16_wincclick_freertos	pic32mx795_pim+e16	Demonstrates the web net server hosted on internal Flash through the MPFS on the PIC32MX795F512L CAN-USB PIM with the Explorer 16/32 Development Board and the MIKROE-2046 Wi-Fi 7 Click Board.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 Ethernet Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32 Ethernet Starter Kit II

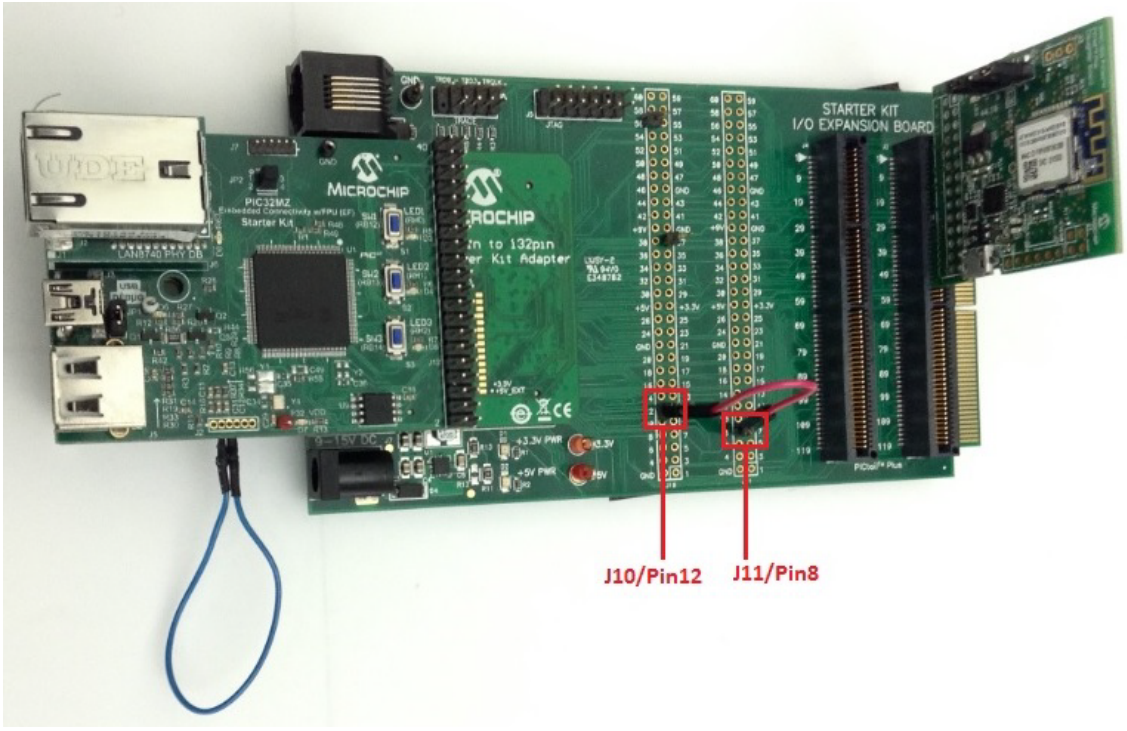
No hardware related configuration or jumper setting changes are necessary.

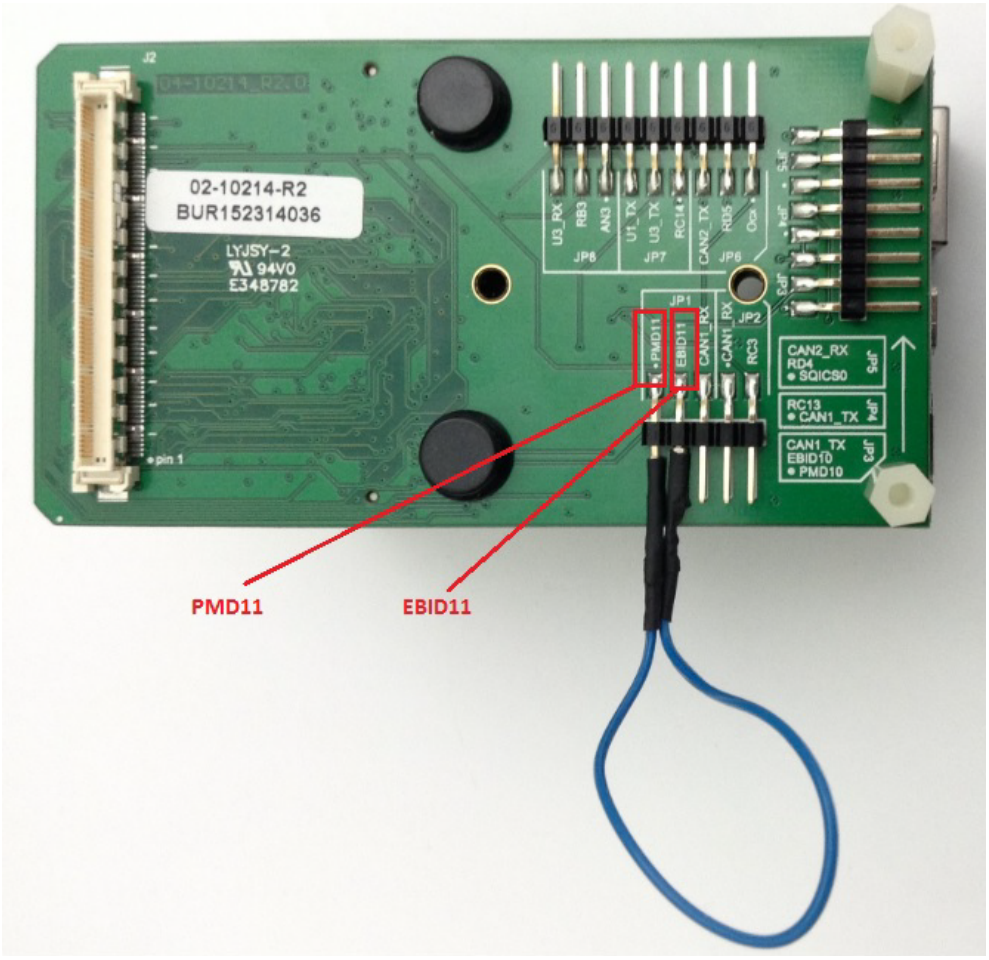
PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ EF Starter Kit with the Starter Kit I/O Expansion Board, WINC1500 Wi-Fi PICtail/PICtail Plus Daughter Board, and PIC32MZ Starter Kit Adapter Board

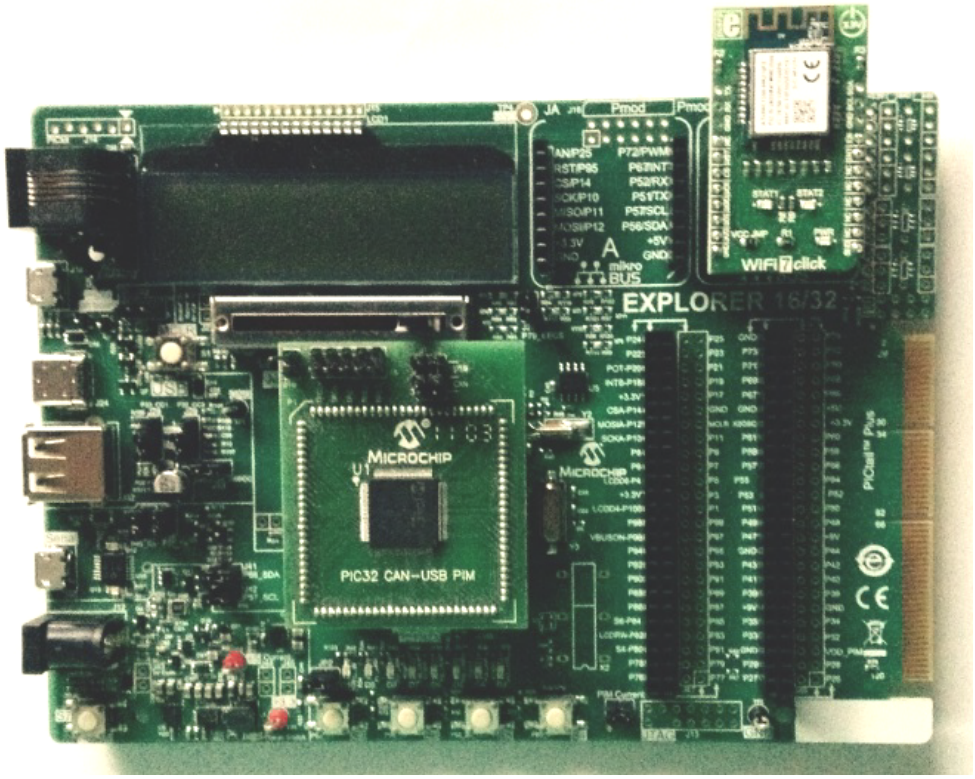
The following figures show the necessary configuration.





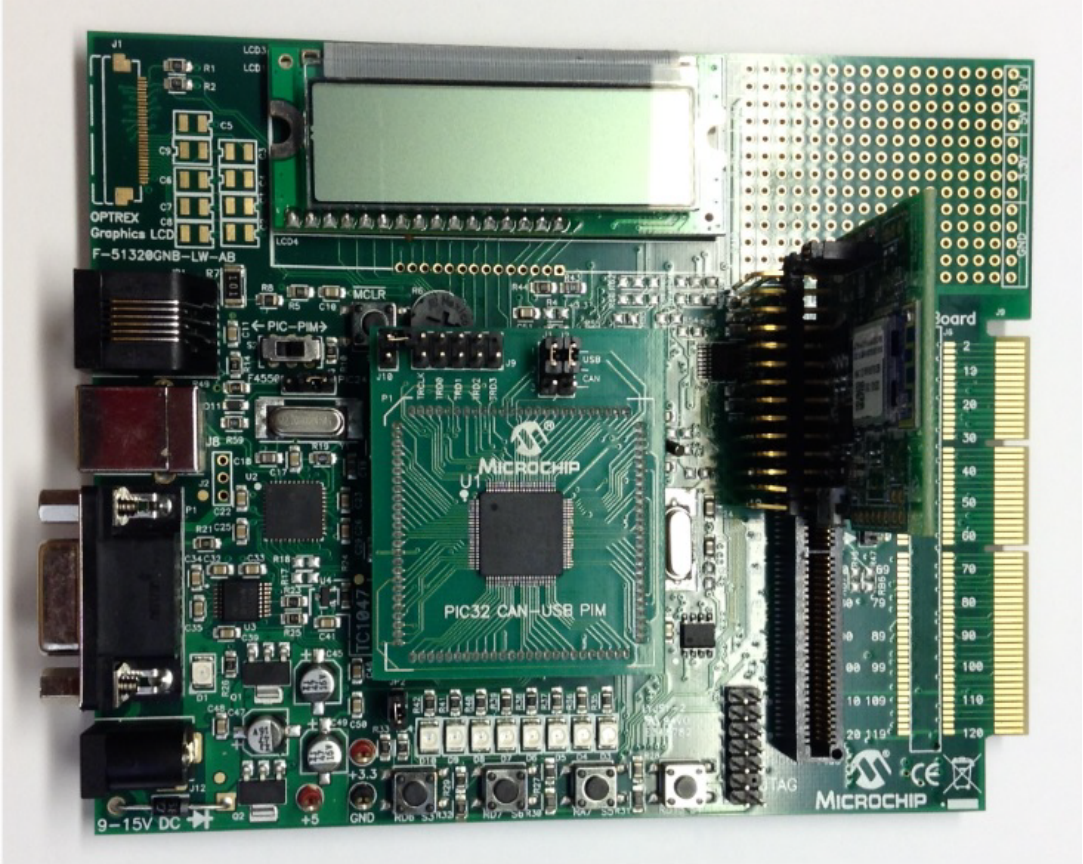
Explorer 16/32 Development Board with the PIC32MX795F512L CAN-USB PIM and the MikroElektronika WiFi 7 Click Board (with on-board WINC1500 Wi-Fi module

The following figure shows the necessary configuration.



Explorer 16 Development Board with the PIC32MX795F512L CAN-USB PIM and WINC1500 Wi-Fi PICtail/PICtail Plus Daughter Board

The following figure shows the necessary configuration.



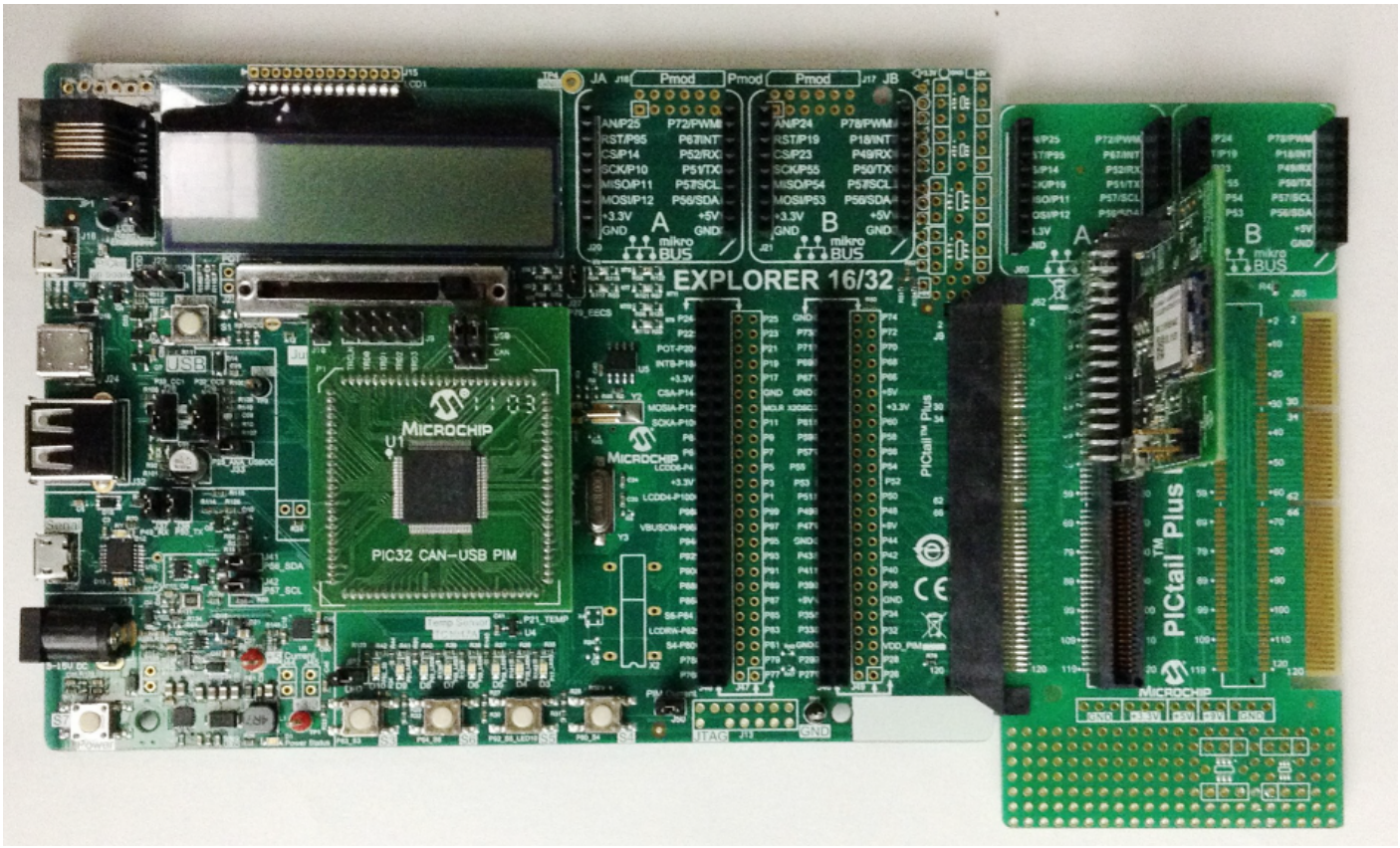
Explorer 16/32 Development Board with the PIC32MX795F512L CAN-USB PIM, PICtail Plus Expansion Board, and WINC1500 Wi-Fi PICtail/PICtail Plus Daughter Board

- Connect the WINC1500 Wi-FI PICtail/PICtail Daughter Board into the PICtail Plus slot (J63). The module should face into the main board and

be placed all the way to the end on the pin 1 side of the connector.

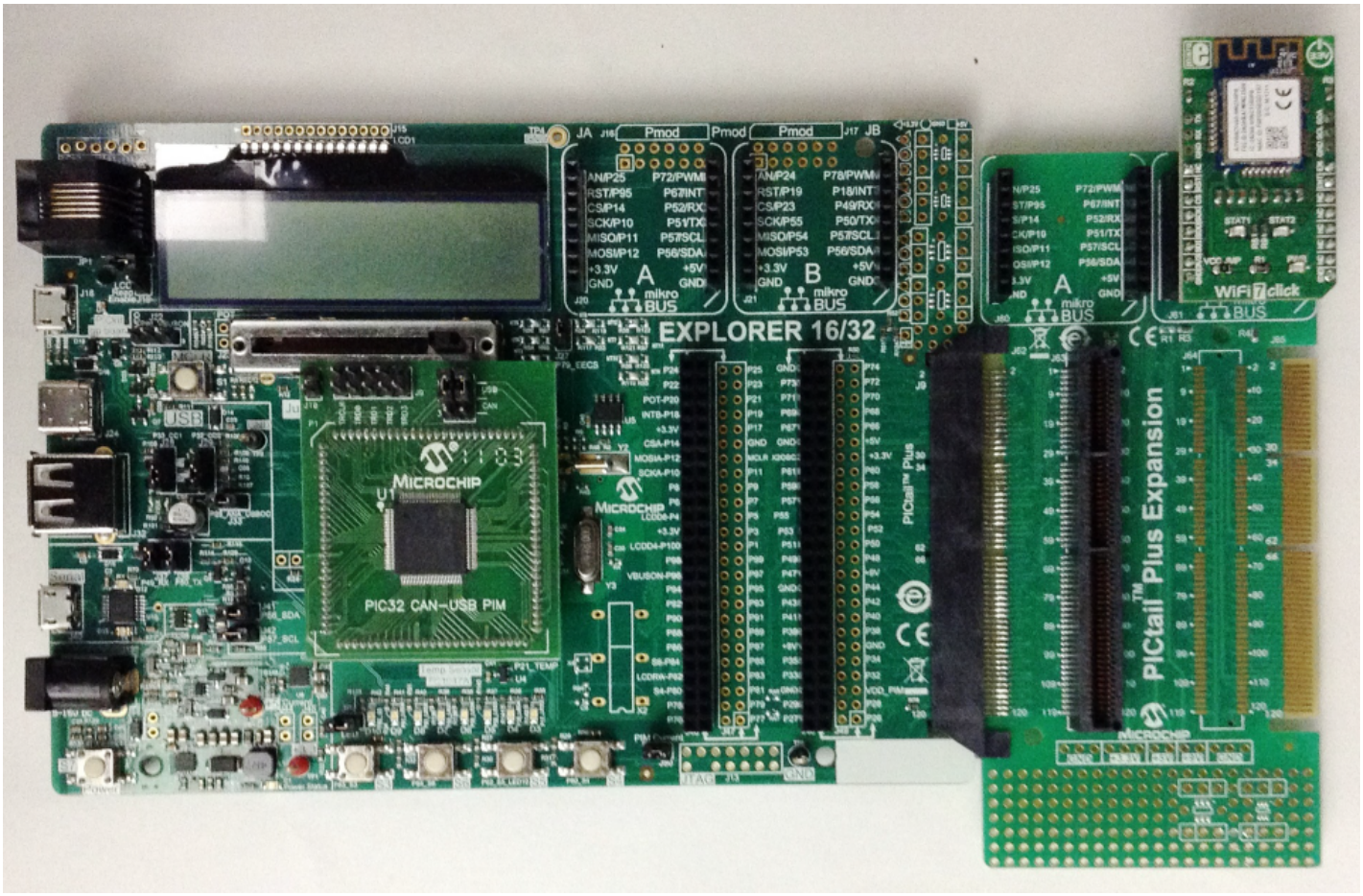
- The console output uses the mini_USB/Serial connector on the board (J40) at 115200 baud, 8-bit data, No parity, 1 Stop bit, with No Flow Control.

The following figure shows the hardware configuration.



Explorer 16/32 Development Board with the PIC32MX795F512L CAN-USB PIM, PICtail Plus Expansion Board, and the MikroElektronika WiFi 7 Click Board (with on-board WINC1500 Wi-Fi module).

The following figure shows the hardware configuration.



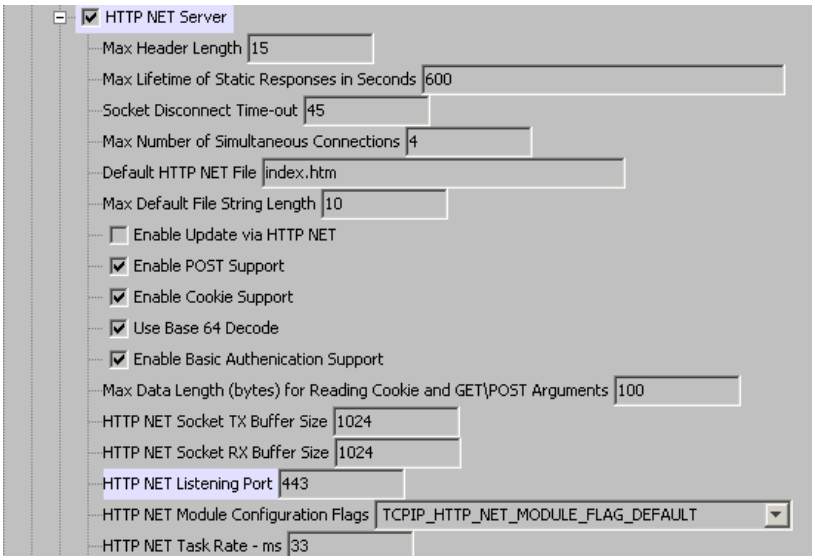
Configuring the MHC

Provides information on the MHC configuration for the demonstration.

Description

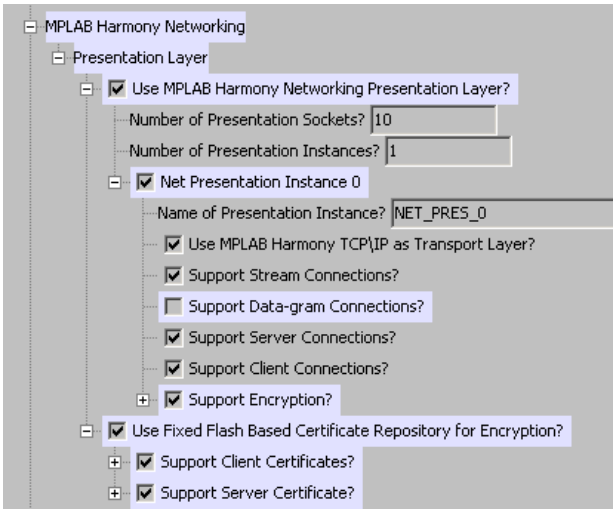
MHC Configuration:

1. From *Harmony Framework Configuration* > *TCPIP Stack* select **HTTP NET Server**.
2. Leave the default settings. The HTTP server listening port should be already set to 443 for encrypted connections, as shown in the following figure; however, change this value to 80 if unencrypted connection is required:

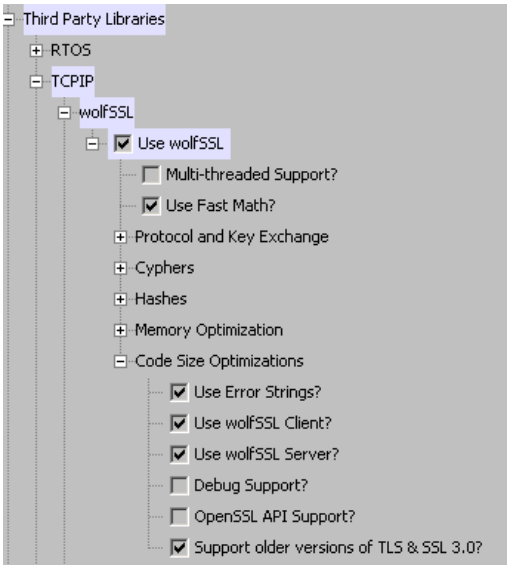


3. Enable the MPLAB Harmony Networking Presentation Layer, as follows:
   - Ensure that the TCP/IP stack is used as transport layer

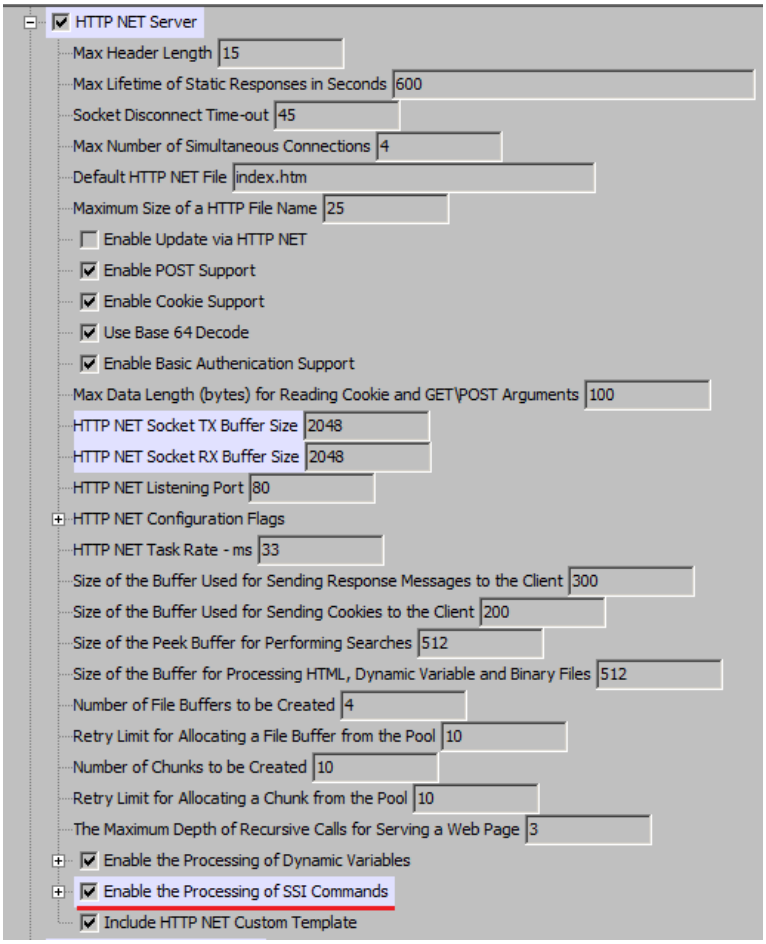
- Select **Support Stream Connections?** (for TCP support)
- Select **Support Server Connections?** (for HTTPS support)
- Select **Support Client Connections?** (for encrypted SMTP support)
- Select **Support Client Certificate?** and **Support Server Certificate?** (as appropriate)



4. Enable *Third Party Libraries* > *TCPIP* > *wolfSSL* > *Use wolfSSL*. Ensure that the wolfSSL client and wolfSSL server are enabled depending on your HTTP and SMTP selection, as shown in the following figure:

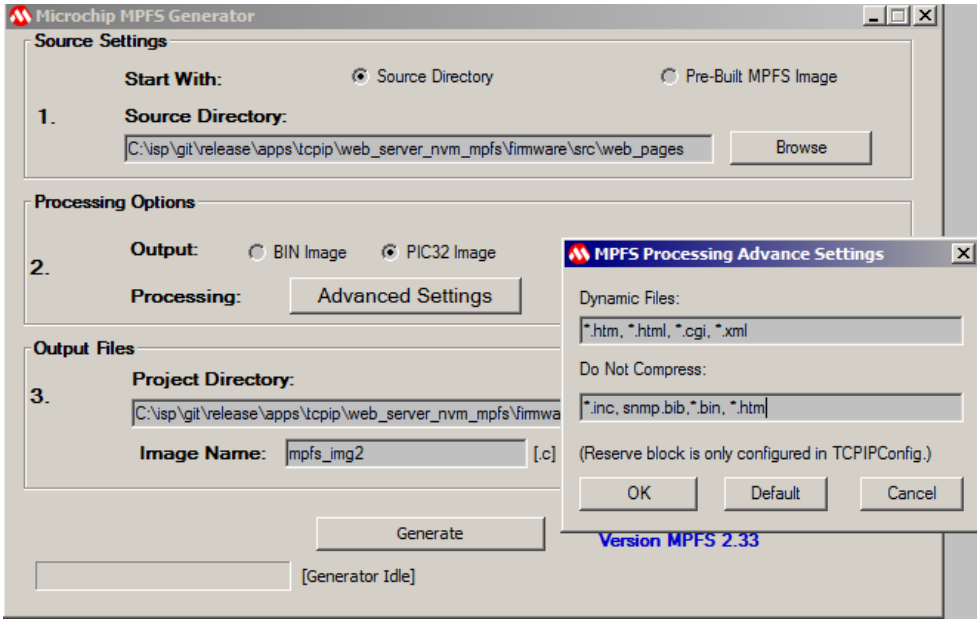


5. As shown in the following figure, select **HTTP Net Server** from *Harmony Framework Configuration* >*TCPIP Stack*. Ensure that the “Enable the Processing SSI Commands” is enabled. Leave the default settings or expand the item and adjust to your application needs.

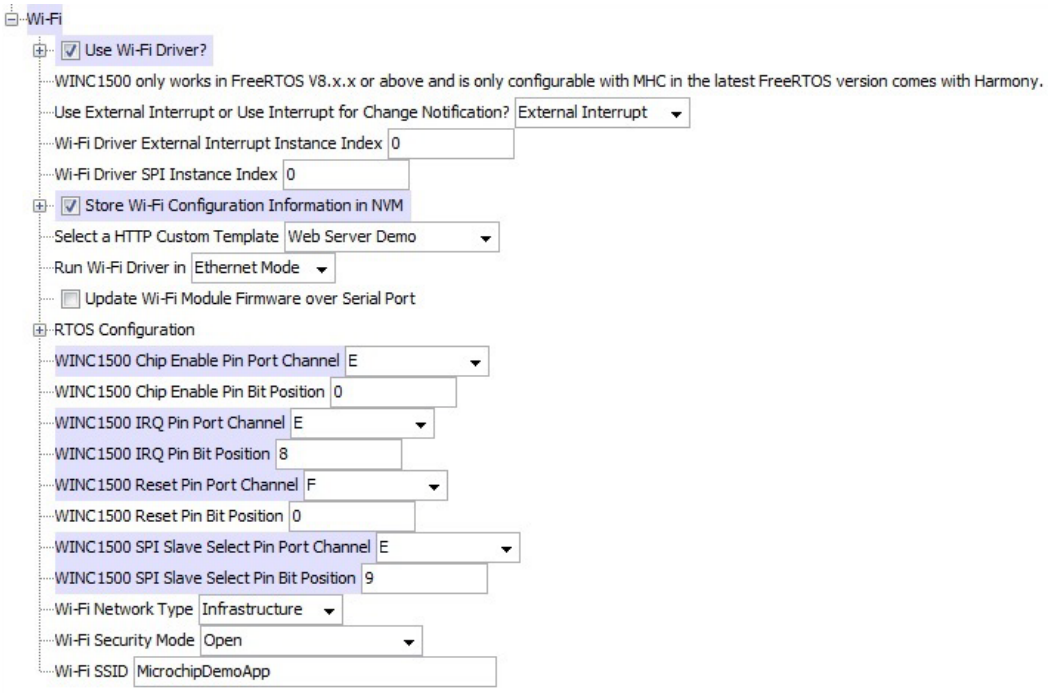


6. This version of the HTTP Net MHC configuration allows for the explicit selection of the dynamic variables processing. Ensure that this option is also selected.

Note: For demonstrations that use SSI, the file inclusion is now done in a standard way using `.htm` files.
For example, `<!--#include virtual="header.htm">`. These files may contain dynamic variables, other SSI commands, or include other files. As shown in the following figure, ensure that when using the `mpfs2.jar` image generation tool that `*.htm` is added using *Advanced Settings > Do Not Compress.*



7. Set the following MHC configuration to enable Wi-Fi network connection.



Running the Demonstration

This section provides instructions on how to build and run the PIC32 Ethernet Web Net Server demonstration.

Description

To view the web page hosted by the demonstration application, open a web browser and direct it to the board running the HTTP server by typing the URL in the address bar (for example, http://mchpboard_e or http://mchpboard_c), and then pressing **Enter**.

Notes:

1. The NetBIOS name of the TCP/IP application is specified at the time the TCP/IP stack is initialized, which is usually in the hostName member of the tcpip_stack_init.c:: TCPIP_HOSTS_CONFIGURATION structure. The NetBIOS service must be enabled for the PIC32 demonstration to respond to NetBIOS queries. Alternatively, you can use the IPv4 or IPv6 address (if IPv6 is enabled) of the board directly, for example, http://192.168.1.131 or http://[fdfe:dcba:9876:1:204:a3ff:fe12:128e].
2. The IPv4 and IPv6 addresses can be obtained from running the TCP/IP Discovery application on the PC side. It requires that the TCP/IP Announce module is enabled when building the stack.
3. With Wi-Fi network connection, the HTTP NET Server demonstration is accessible in duo-port mode (i.e., Ethernet and Wi-Fi). To access the HTTP NET Server pages on a web browser through the Wi-Fi connection, use the IPv4/IPv6 address assigned to the Wi-Fi connection.

Demonstration Process

Please use the following procedure to run the demonstration:

1. Load the demonstration project into MPLAB X IDE.
2. Connect the USB debugger port on-board the starter kit in use to a USB port on the Development computer using the USB cable provided in the kit.
3. Connect the RJ-45 Ethernet port on the starter kit board to a network hub or an Ethernet port on the development computer using the Ethernet patch cord provided in the kit.
4. Build, download, and run the demonstration project on the target board.
5. A HTTP server is hosted by the demonstration application. Open a web browser and direct it to the board running the HTTP server by typing the URL in the address bar (for example, http://mchpboard_e or http://mchpboard_c), and then pressing **Enter**.

The demonstration application features following:

- *Real-time Hardware Control and Dynamic Variables* - On the Overview page the LEDs can be clicked to toggle the LEDs on the PIC32 Ethernet Starter Kit. The buttons on the PIC32 Ethernet Starter Kit can be pressed to toggle the Buttons on the web page. The dynamic variables can be updated in real-time on the HTTP server.

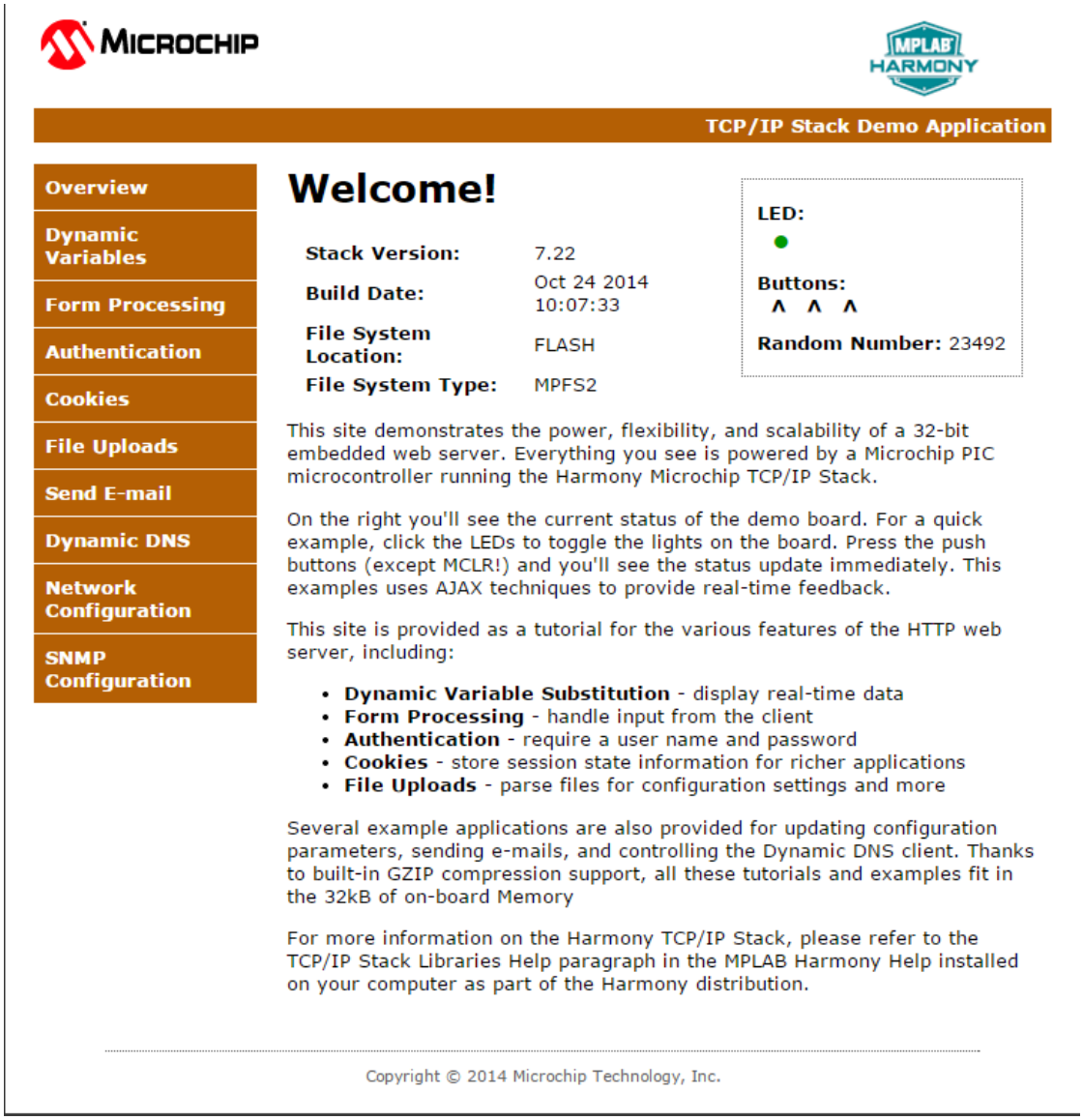
Note: The LED functionality portion of the demonstration is somewhat limited due to issue related to the functional multiplexing on GPIO and Ethernet pins on different supported hardware.

- *Form Processing* - Input can be handled from the client by using GET and POST methods (this functionality controls the on-board LEDs and will be operational only on Explorer 16 Development Board)
- *Authentication* - Shows an example of the commonly used restricted access feature

- *Cookies* - Shows an example of storing small text stings on the client side
- *File Uploads* - Shows an example of file upload using the POST method. The HTTP server can accept a user-defined MPFS/MPFS2 image file for web pages.
- *Send E-mail* - Shows simple SMTP POST methods
- *Dynamic DNS* - Exercises Dynamic DNS capabilities
- *Network Configuration* - The MAC address, host name, and IP address of the PIC32 Ethernet Starter Kit can be viewed in the Network Configuration page and some configurations can be updated
- *MPFS Upload* - A new set of web pages can be uploaded to the web server using this feature, which is accessed through http://mchpboard_e/mpfsupload.

Notes:

1. The location of the MPFS image is fixed at the beginning of the Flash page (aligned to the page boundary). The size of the MPFS upload is limited to 64K in the demonstration, which it can be expanded by changing NVM_MEDIA_SIZE to the desired size (restricted based on the available size) and overriding the EBASE address using the following linker command:
   - --defsym=_ebase_address=0x9D0xxxx (where, xxxx = 9D000000+NVM_MEDIA_SIZE)
2. The MPFS UPLOAD functionality has to be enabled when the project is built.



web_photoframe_demo

Before using this demonstration, please see the important notes in the *TCP/IP Demonstrations* > *Introduction*.

The demonstration application creates a Web Photoframe that displays images, in the form of a slideshow on a web browser by connecting to a web server hosted on PIC32 Development board, over Ethernet or Wi-Fi interface. The images are stored on a micro SDCARD connected to the PIC32 development board.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the PIC32 Web Photoframe Demonstration.

Description

To build this project, you must open the `pic32_eth_wifi_photoframe.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/tcpip/web_photoframe`.



Warning

When using the Microchip Harmony Configurator (MHC), care must be taken when generating the code to *not* erase the USB descriptors in the `system_init.c` file.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`pic32_eth_web_server.X`	`<install-dir>/apps/tcpip/web_photoframe/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within ./firmware/src/system_config.

Project Configuration Name	BSP Used	Description
pic32mz_ef_curiosity	pic32mz_ef_curiosity	FreeRTOS version of the demonstration running on the PIC32MZ EF Curiosity Development board with the on-board MRF24WN module.

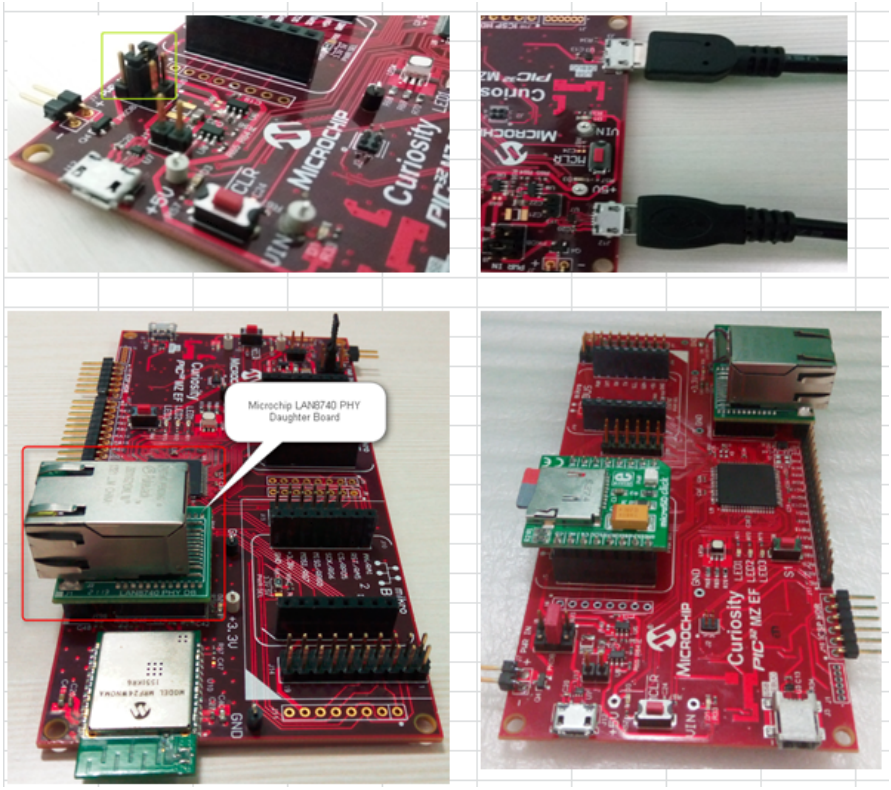
Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MZ EF Curiosity Development Board with on-board MRF24WN0MA module

- Ensure that a jumper is placed at 4-3 on J8, to select supply from debug USB connector.
- Power the PIC32MZ EF Curiosity Development Board from a Host PC through a Type-A male to micro-B USB cable connected to Micro-B port (J3)
- Ensure that a jumper is not present in the J13 header to use the PIC32MZ EF Curiosity Board in Device mode
- Connect a USB cable with a micro-B type connector to Micro-B port (J12), and connect the other end to your computer
- Ensure that you have a LAN8740 Ethernet PHY DB installed on the PIC32MZ EF Curiosity Development Board (header J18)
- Ensure that you have a microSD click board on the microBUS1 (J5) connector of the PIC32MZ EF Curiosity Development Board



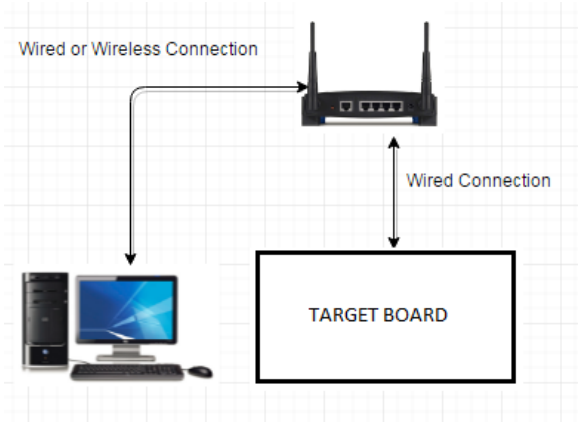
Running the Demonstration

This section provides instructions on how to build and run the demonstration.

Description

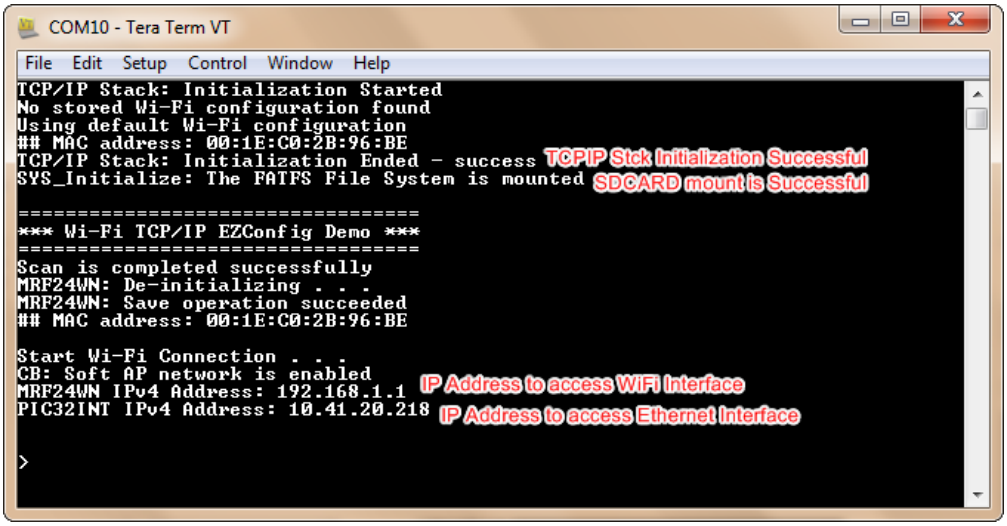
For the purpose of this demo, both the Target board and the Host PC should be in the same network.

The host PC can be connected to a router via an Ethernet cable or WiFi. The Target board should be connected to the router via an Ethernet cable. Please refer to the connection diagram shown below.

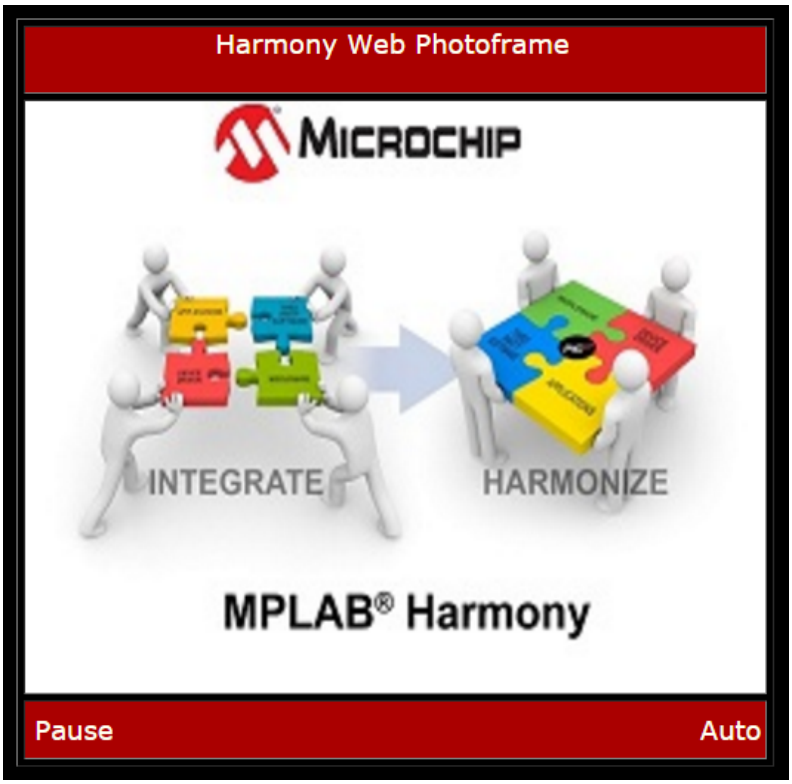


The SDCARD should be loaded with `index.htm` and the images folder with `.jpg` images. The `index.htm` and the `images` folder can be found at the following location: `<install-dir>/apps/tcpip/web_photoframe/firmware/src/web_pages`.

1. Load the demonstration project into MPLAB X IDE.
2. Connect the USB debugger port on-board the starter kit to a USB port on the development computer using the USB cable provided in the kit.
3. A USB cable needs to be connected to the micro-B USB connector on the bottom of the starter kit in use. When the demonstration runs, it will create a USB CDC device on the USB bus. The demonstration can be executed once you have connected to this device through a standard terminal program, set the baud rate to 921,600 baud, and a valid IP address has been received by the device.
4. Copy the files `index.htm`.
5. Build, download, and run the demonstration project on the board.
6. Connect to the board through a standard terminal program.



7. In the previous image we can see that the board scans for local Access Points and outputs the results to the serial console. After the scan results, the MRF24WN goes into SoftAP mode.
8. To run the demonstration on an Ethernet Interface, open a web browser on the host PC and type the IP address for the Ethernet Interface (PIC32INT IPv4 Address) obtained from the output of the serial terminal.
9. The web browser will load the application web page hosted by the SDCARD. And the `.jpg` images stored in the SDCARD will be displayed on the web page in a scrolling fashion. Press the “Pause” button to pause the slideshow.



10. From the host PC, connect to the MCHPSoftAP access point, which is the SoftAP network started by the demonstration. Then, bring up a web page by entering the IP address of the SoftAP network into the browser. This is the IP address displayed in step 6 (e.g., 192.168.1.1). The application web page will be displayed with the slideshow of the images.

web_server_nvm_mpfs

Before using this demonstration, please see the important notes in the *TCP/IP Demonstrations* > *Introduction*.

Web Server Non-volatile Memory (NVM) MPFS TCP/IP demonstrations.

pic32_eth_web_server

This section describes the steps necessary to begin using the PIC32 Ethernet Web Server Demonstration Application.

Description

This demonstration exercises the HTTP web server running on PIC32 devices. The Non-Volatile Memory (NVM) Microchip Proprietary File System (MPFS) web server demonstration has the web pages stored in internal Flash and are accessed through the MPFS API.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the TCP/IP Web Server Demonstration.

Description

To build this project, you must open the `pic32_eth_web_server.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/tcpip/web_server_nvm_mpfs`.



Warning

When using the Microchip Harmony Configurator (MHC), care must be taken when generating the code to *not* erase the USB descriptors in the `system_init.c` file.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`pic32_eth_web_server.X`	`<install-dir>/apps/tcpip/web_server_nvm_mpfs/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx_eth_sk2	pic32mx_eth_sk2	Demonstrates the web server hosted on internal Flash through Microchip proprietary file system on the PIC32 Ethernet Starter Kit II.
pic32mz_ef_sk	pic32mz_ef_sk	Demonstrates the web server hosted on internal Flash through Microchip proprietary file system on the PIC32MZ EF Starter Kit.
pic32mz_ef_sk_16b	pic32mz_ef_sk	Demonstrates the web server hosted on internal Flash through the MPFS on the PIC32MZ EF Starter Kit.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 Ethernet Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32 Ethernet Starter Kit II

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

This section provides instructions on how to build and run the PIC32 Ethernet Web Server demonstration.

Description

To view the web page hosted by the demonstration application, open a web browser and direct it to the board running the HTTP server by typing the URL in the address bar (for example, http://mchpboard_e or http://mchpboard_c), and then pressing **Enter**.

Notes:

1. The NetBIOS name of the TCP/IP application is specified at the time the TCP/IP stack is initialized, which is usually in the hostName member of the tcpip_stack_init.c:: TCPIP_HOSTS_CONFIGURATION structure. The NetBIOS service must be enabled for the PIC32 demonstration to respond to NetBIOS queries. Alternatively, you can use the IPv4 or IPv6 address (if IPv6 is enabled) of the board directly, for example, http://192.168.1.131 or http://[fdfe:dcba:9876:1:204:a3ff:fe12:128e].
2. The IPv4 and IPv6 addresses can be obtained from running the TCP/IP Discovery application on the PC side. It requires that the TCP/IP Announce module is enabled when building the stack.

Demonstration Process

Please use the following procedure to run the demonstration:

1. Load the demonstration project into MPLAB X IDE.
2. Connect the USB debugger port on-board the starter kit in use to a USB port on the Development computer using the USB cable provided in the kit.
3. Connect the RJ-45 Ethernet port on the starter kit board to a network hub or an Ethernet port on the development computer using the Ethernet patch cord provided in the kit.
4. Build, download, and run the demonstration project on the target board.
5. A HTTP server is hosted by the demonstration application. Open a web browser and direct it to the board running the HTTP server by typing the URL in the address bar (for example, http://mchpboard_e or http://mchpboard_c), and then pressing **Enter**.

The demonstration application features following:

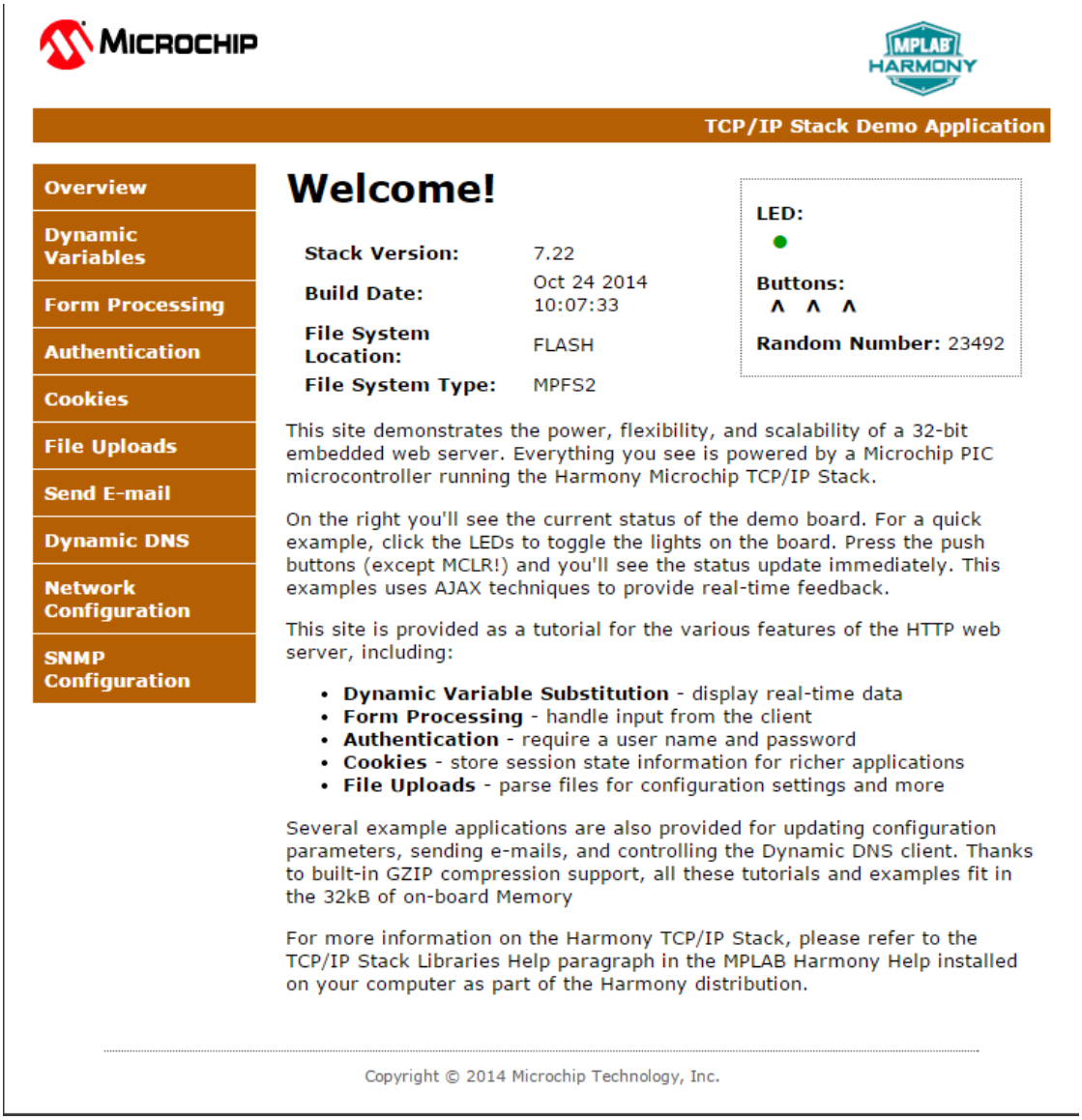
- *Real-time Hardware Control and Dynamic Variables* - On the Overview page the LEDs can be clicked to toggle the LEDs on the PIC32 Ethernet Starter Kit. The buttons on the PIC32 Ethernet Starter Kit can be pressed to toggle the Buttons on the web page. The dynamic variables can be updated in real-time on the HTTP server.

Note: The LED functionality portion of the demonstration is somewhat limited due to issue related to the functional multiplexing on GPIO and Ethernet pins on different supported hardware.

- *Form Processing* - Input can be handled from the client by using GET and POST methods (this functionality controls the on-board LEDs and will be operational only on Explorer 16 Development Board)
- *Authentication* - Shows an example of the commonly used restricted access feature
- *Cookies* - Shows an example of storing small text stings on the client side
- *File Uploads* - Shows an example of file upload using the POST method. The HTTP server can accept a user-defined MPFS/MPFS2 image file for web pages.
- *Send E-mail* - Shows simple SMTP POST methods
- *Dynamic DNS* - Exercises Dynamic DNS capabilities
- *Network Configuration* - The MAC address, host name, and IP address of the PIC32 Ethernet Starter Kit can be viewed in the Network Configuration page and some configurations can be updated
- *MPFS Upload* - A new set of web pages can be uploaded to the web server using this feature, which is accessed through http://mchpboard_e/mpfsupload.

Notes:

1. The location of the MPFS image is fixed at the beginning of the Flash page (aligned to the page boundary). The size of the MPFS upload is limited to 64K in the demonstration, which it can be expanded by changing NVM_MEDIA_SIZE to the desired size (restricted based on the available size) and overriding the EBASE address using the following linker command:
   - --defsym=_ebase_address=0x9D0xxxx (where, xxxx = 9D000000+NVM_MEDIA_SIZE)
2. The MPFS UPLOAD functionality has to be enabled when the project is built.



pic32_eth_wifi_web_server

This section describes the steps necessary to begin using the PIC32 Ethernet Wi-Fi Web Server Demonstration Application.

Description

The Wi-Fi Web Server demonstration (`apps\tcpip\web_server_nvm_mpfs\firmware`) exercises the HTTP web server running on PIC32 devices. The Non-Volatile Memory (NVM) Microchip Proprietary File System (MPFS) web server demonstration has the web pages stored in internal Flash and are accessed through the MPFS API.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the PIC32 Ethernet Wi-Fi Web Server Demonstration.

Description

To build this project, you must open the `pic32_eth_wifi_web_server.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/tcpip/web_server_nvm_mpfs`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
pic32_eth_wifi_web_server.X	<install-dir>/apps/tcpip/web_server_nvm_mpfs/firmware

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx_eth_sk_ioexp_11n_freertos	pic32mx_eth_sk	Demonstrates the web server hosted on internal Flash through the MPFS on the PIC32 Ethernet Starter Kit and the MRF24WN0MA PICtail/PICtail Plus Daughter Board.
pic32mx_eth_sk_ioexp_winc_freertos	pic32mx_eth_sk	Demonstrates the web server hosted on internal Flash through the MPFS on the PIC32 Ethernet Starter Kit and the WINC1500 PICtail/PICtail Plus Daughter Board.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

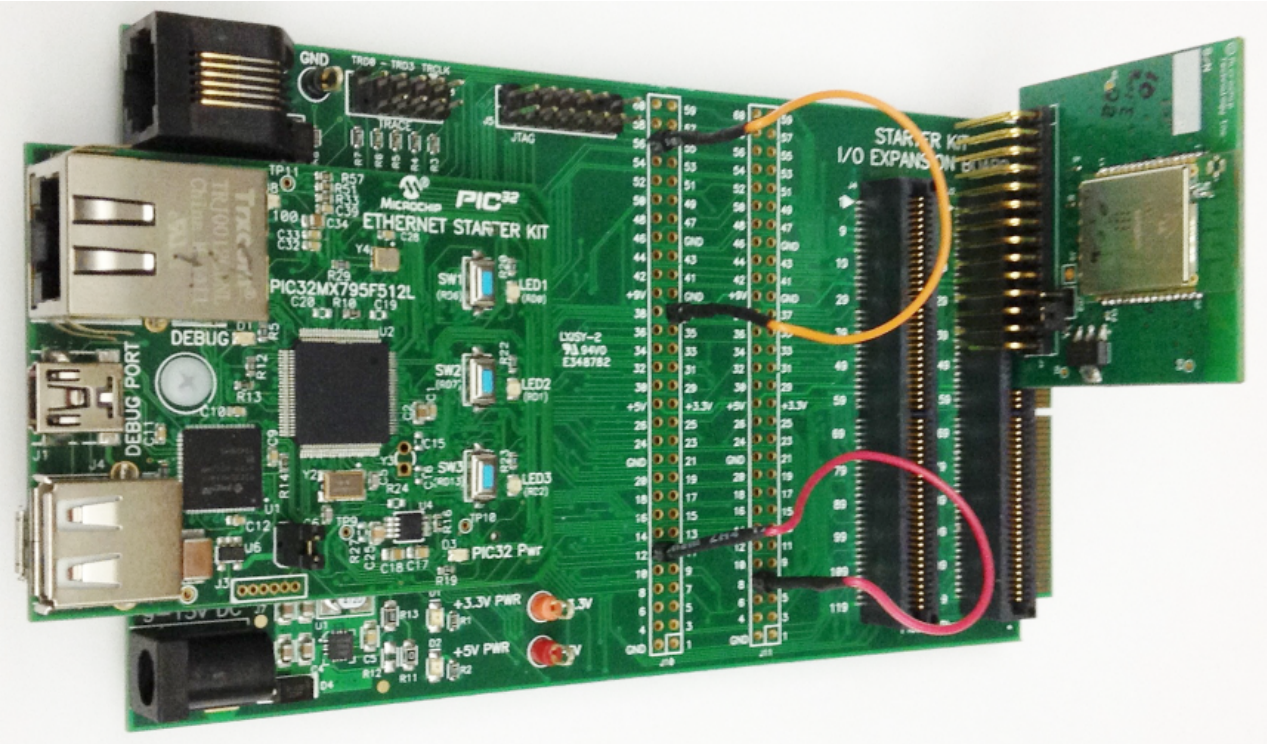
PIC32 Ethernet Starter Kit with PIC32MX795F512L, MRF24WN0MA Wi-Fi PICtail/PICtail Plus Daughter Board, and Starter Kit I/O Expansion Board

(shown in the following figure below)

Note: For use with the WINC1500 module, replace the MRF24WN module shown in the following figure with the WINC1500 Wi-Fi PICtail/PICtail Plus Daughter Board.

Configure the Starter Kit I/O Expansion Board jumper modifications, as described in the following figure:

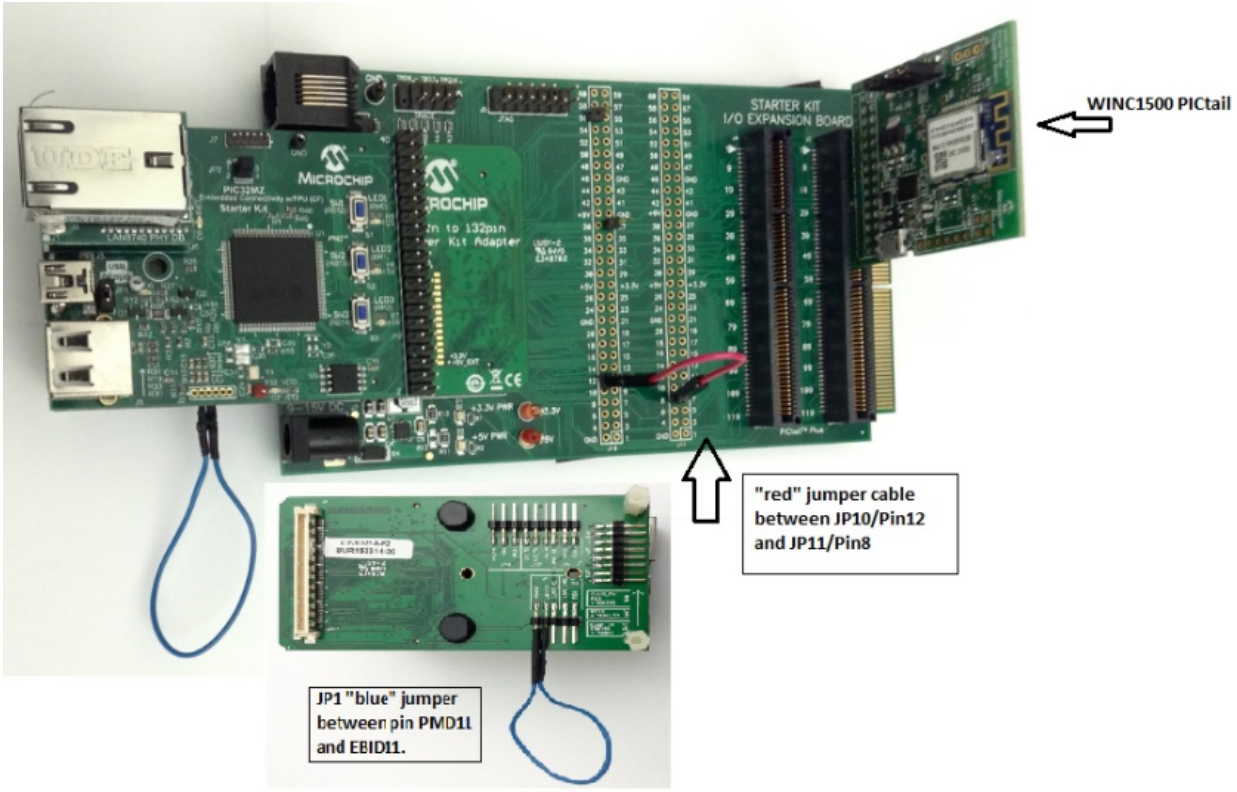
On-board Jumpers: J10/pin 12 to J11/pin 8 (red jumper cable), and J10/pin 56 to J10/pin 37 (orange jumper cable).



PIC32MZ EF Starter Kit with PIC32MX2048EFM144, WINC1500 Wi-Fi PICtail/PICtail Plus Daughter Board, and Starter Kit I/O Expansion Board.

Configure the hardware, as shown in the following figure.

On-board jumpers: JP10/pin 12 to JP11/pin 8 (red jumper cable), and JP1 between pin PMD11 and pin EBID11 (blue jumper cable)



Running the Demonstration

This section provides instructions on how to build and run the PIC32 Ethernet Wi-Fi Web Server demonstration.

Description

To view the web page hosted by the demonstration application, open a web browser and direct it to the board running the HTTP server by typing the URL in the address bar (for example, http://mchpboard_w), and then pressing **Enter**.

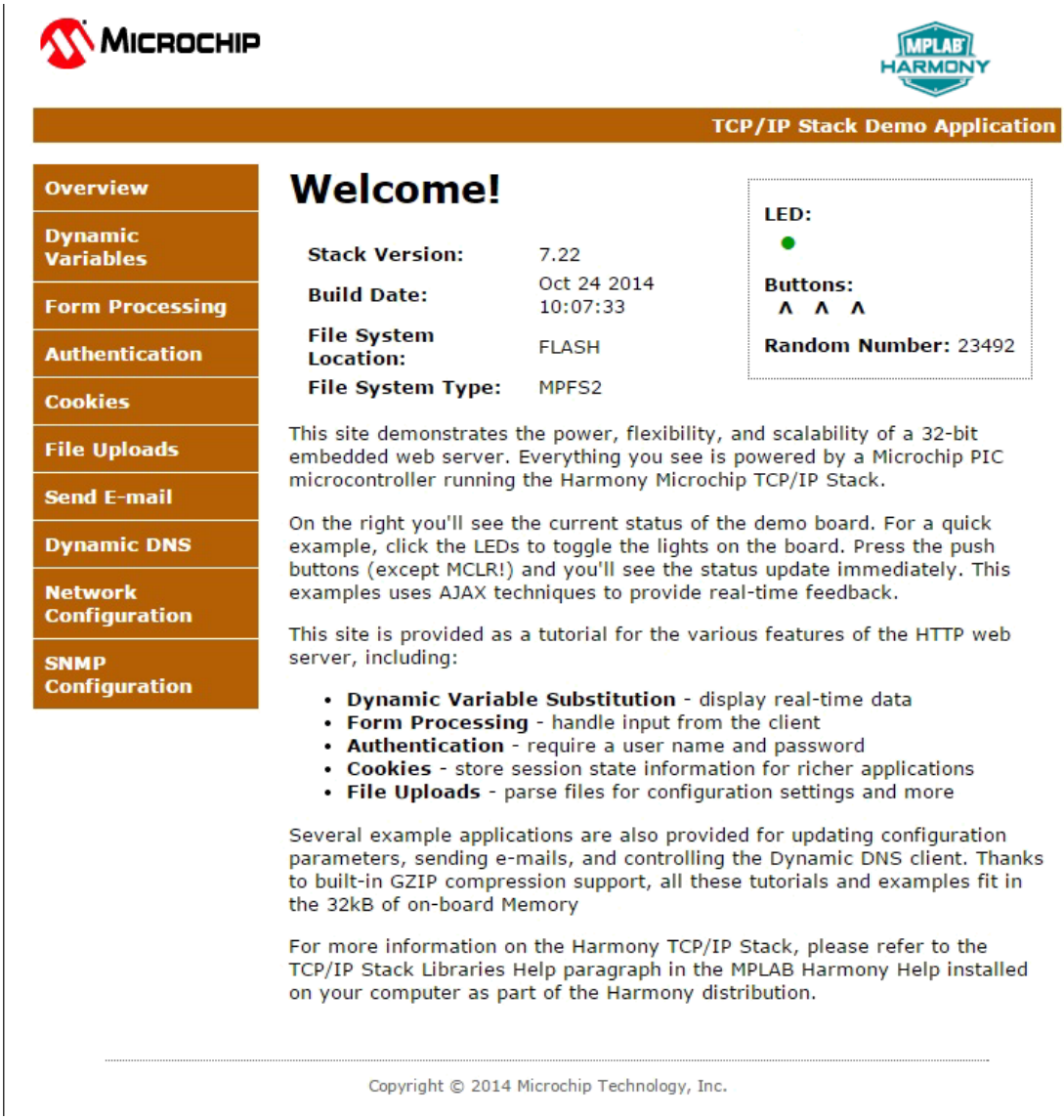
This demonstration runs on two interfaces, Ethernet and Wi-Fi. A second web browser tab could be opened pointing to the other interface (http://mchpboard_e) to have both interfaces running simultaneously.

Notes:

1. The NetBIOS name of the TCP/IP application is specified at the time the TCP/IP stack is initialized, which is usually in the hostName member of the tcpip_stack_init.c:: TCPIP_HOSTS_CONFIGURATION structure. The NetBIOS service must be enabled for the PIC32 demonstration to respond to NetBIOS queries. Alternatively, you can use the IPv4 or IPv6 address (if IPv6 is enabled) of the board directly, for example, http://192.168.1.131 or http://[fdfe:dcba:9876:1:204:a3ff:fe12:128e].
2. The IPv4 and IPv6 addresses can be obtained from running the TCP/IP Discovery application on the PC side. It requires that the TCP/IP Announce module is enabled when building the stack.

Please refer to the **Demonstration Process** in the Running the Demonstration section for the pic32_eth_web_server configuration, as the process is the same for this configuration.

Note: Refer to Wi-Fi Console Commands for information on the commands that enable control over the Wi-Fi settings.



pic32_wifi_web_server

This section describes the steps necessary to begin using the PIC32 Wi-Fi Web Server Demonstration Application.

Description

The Wi-Fi Web Server demonstration (`<install-dir>\apps\tcpip\web_server_nvm_mpfs\firmware`) exercises the HTTP Web Server running on PIC32 devices. The Non-Volatile Memory (NVM) Microchip Proprietary File System (MPFS) Web Server demonstration has the Web pages stored in internal Flash and are accessed through the MPFS API.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the PIC32 Wi-Fi Web Server Demonstration.

Description

To build this project, you must open the `pic32_wifi_web_server.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/tcpip/web_server_nvm_mpfs`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
pic32_wifi_web_server.X	<install-dir>/apps/tcpip/web_server_nvm_mpfs/firmware

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx795_pim_e16_11n_freertos	pic32mx795_pim+e16	Demonstrates the web server hosted on internal Flash through the MPFS on the PIC32MX795F512L CAN-USB PIM with the Explorer 16/32 Development Board and the MRF24WN0MA PICtail/PICtail Plus Daughter Board.
pic32mx795_pim_e16_winc_freertos	pic32mx795_pim+e16	Demonstrates the web server hosted on internal Flash through the MPFS on the PIC32MX795F512L CAN-USB PIM with the Explorer 16/32 Development Board and the WINC1500 PICtail/PICtail Plus Daughter Board.
pic32mx795_pim_e16_wincclick_freertos	pic32mx795_pim+e16	Demonstrates the web server hosted on internal Flash through the MPFS on the PIC32MX795F512L CAN-USB PIM with the Explorer 16/32 Development Board and the MIKROE-2046 Wi-Fi 7 Click Board.

Configuring the Hardware

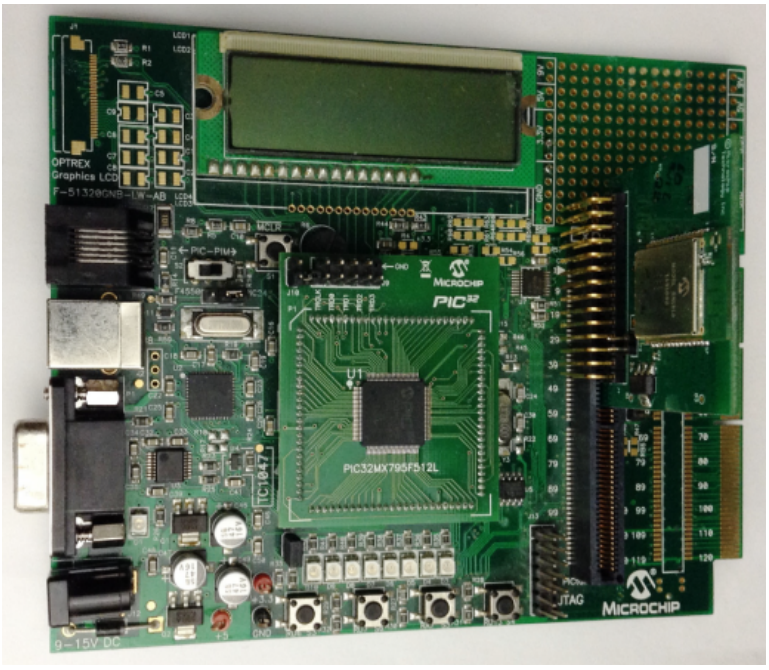
Describes how to configure the supported hardware.

Description

PIC32MX795F512L PIM with the Explorer 16 Development Board and MRF24WN PICtail Daughter Board

- Connect the MRF24WN PICtail Daughter Board into the PICtail Plus slot closest to the MCU. The module should face into the board and be placed all the way to the end on the pin 1 side of the connector.
- The console output uses the DB9 connector on the board at 115200 baud, 8-bit data, No parity, 1 Stop bit, with No Flow Control

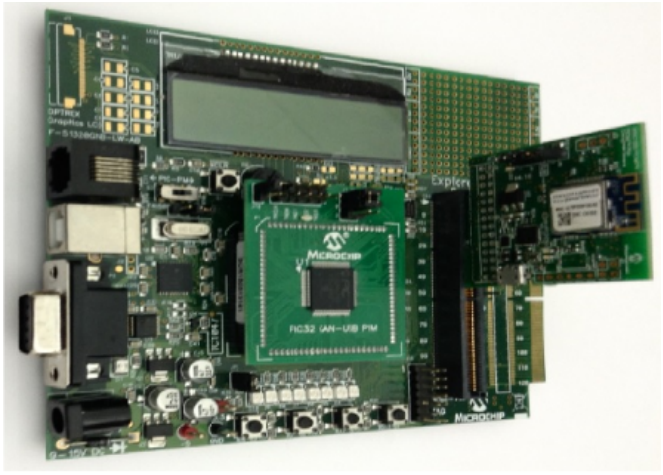
The following figure shows the hardware configuration.



PIC32MX795F512L PIM with the Explorer 16 Development Board and WINC1500 PICtail/PICtail Plus Daughter Board

- Connect the WINC1500 PICtail Daughter Board into the PICtail Plus slot closest to the MCU. The module should face into the board and be placed all the way to the end on the pin 1 side of the connector.
- The console output uses the DB9 connector on the board at 115200 baud, 8-bit data, No parity, 1 Stop bit, with No Flow Control

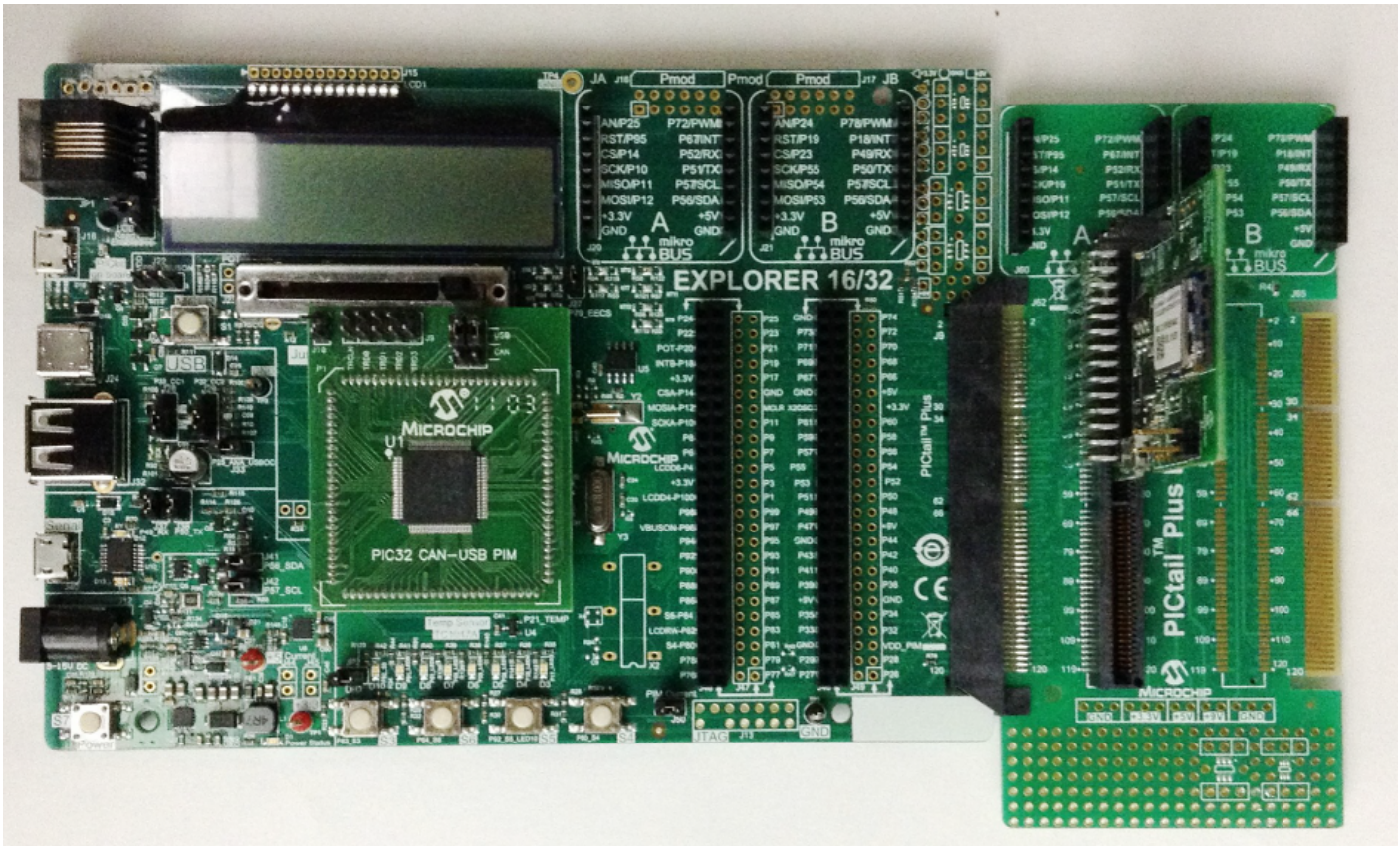
The following figure shows the hardware configuration.



Explorer 16/32 Development Board with the PIC32MX795F512L CAN-USB PIM, PICtail Plus Expansion Board, and WINC1500 Wi-Fi PICtail/PICtail Plus Daughter Board

- Connect the WINC1500 Wi-FI PICtail/PICtail Plus Daughter Board into the PICtail Plus slot (J63) . The module should face into the main board and be placed all the way to the end on the pin 1 side of the connector.
- The console output uses the mini_USB/Serial connector on the board (J40) at 115200 baud, 8-bit data, No parity, 1 Stop bit, with No Flow Control.

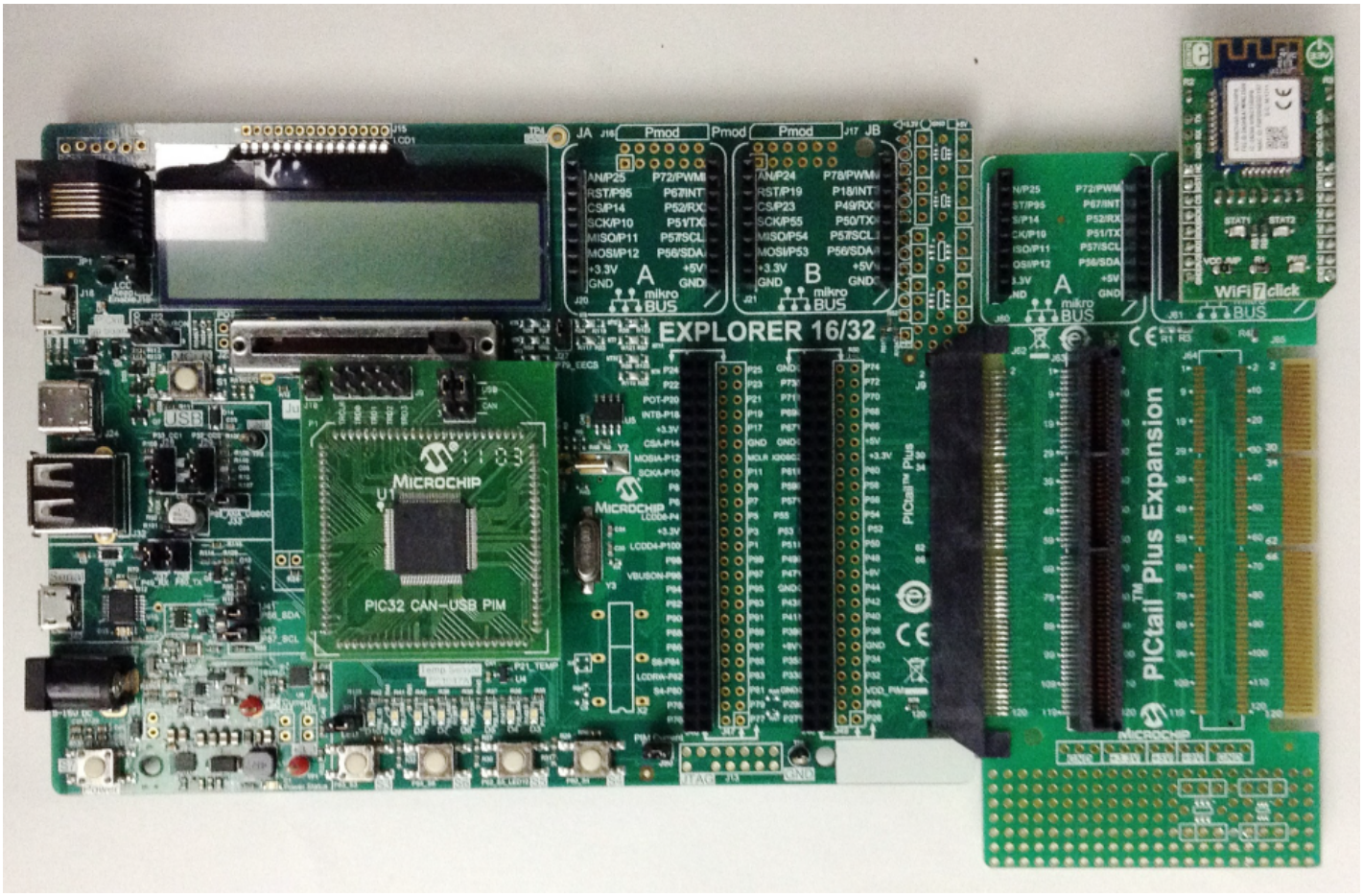
The following figure shows the hardware configuration.



Explorer 16/32 Development Board with the PIC32MX795F512L CAN-USB PIM, PICtail Plus Expansion Board, and the MikroElektronika WiFi 7 Click Board (with on-board WINC1500 Wi-Fi module).

- Connect the WINC1500 Wi-Fi PICtail/PICtail Plus Daughter Board into the PICtail Plus slot (J63) . The module should face into the main board and be placed all the way to the end on the pin 1 side of the connector.
- The console output uses the mini_USB/Serial connector on the board (J40) at 115200 baud, 8-bit data, No parity, 1 Stop bit, with No Flow Control

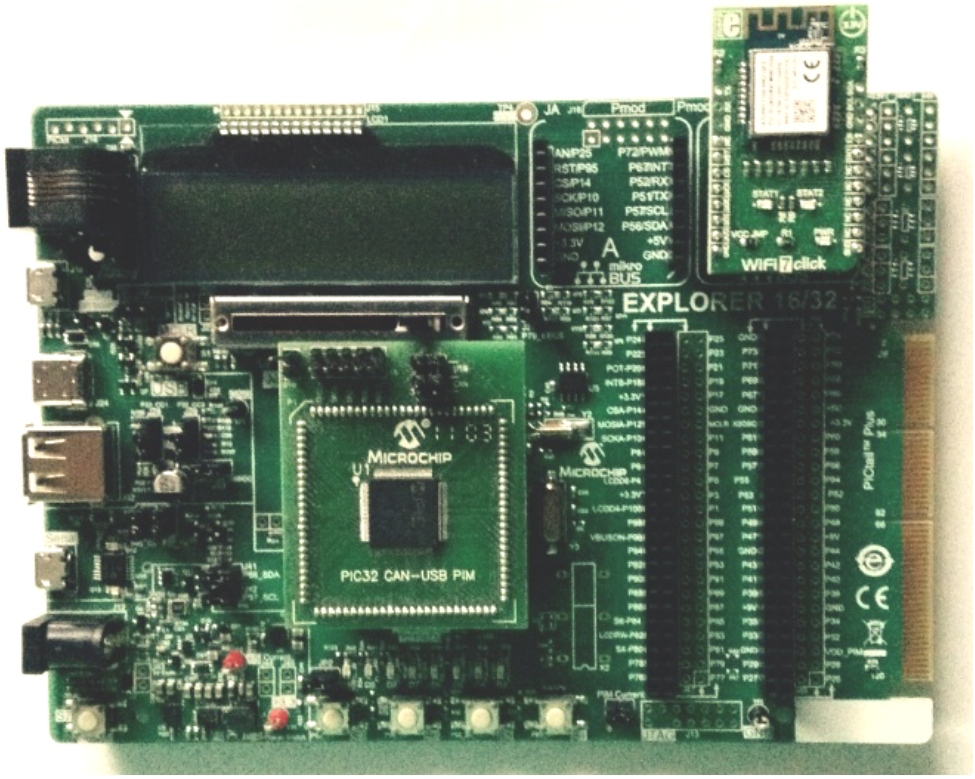
The following figure shows the hardware configuration.



Explorer 16/32 Development Board with the PIC32MX795F512L CAN-USB PIM and the MikroElektronika WiFi 7 Click Board (with on-board WINC1500 Wi-Fi module).

- Connect the WINC1500 Wi-Fi PICtail/PICtail Plus Daughter Board into the PICtail Plus slot. The module should face into the main board and be placed all the way to the end on the pin 1 side of the connector.
- The console output uses the mini_USB/Serial connector on the board (J40) at 115200 baud, 8-bit data, No parity, 1 Stop bit, with No Flow Control

The following figure shows the hardware configuration.



Running the Demonstration

This section provides instructions on how to build and run the PIC32 Wi-Fi Web Server demonstration.

Description

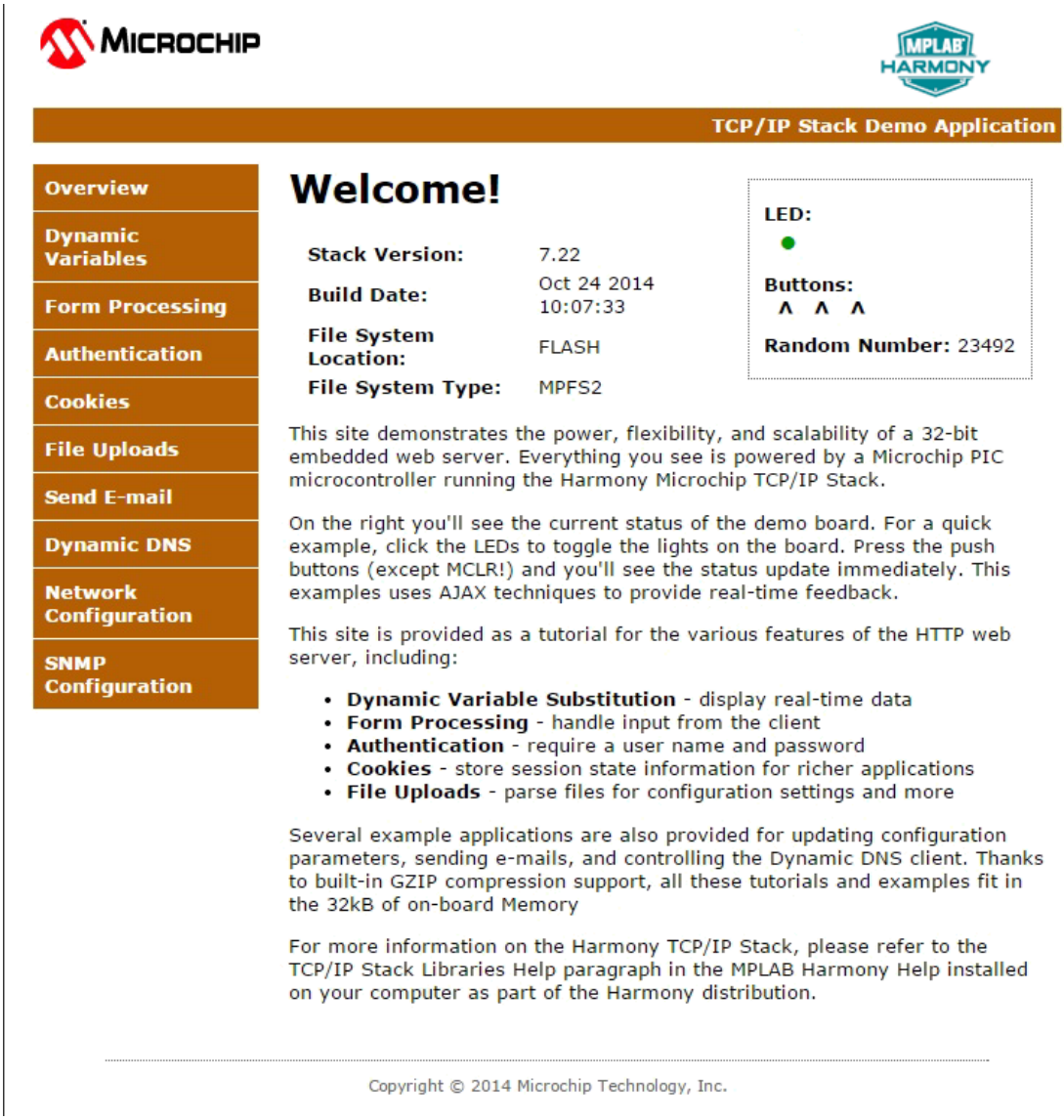
To view the web page hosted by the demonstration application, open a web browser and direct it to the board running the HTTP server by typing the URL in the address bar (for example, http://mchpboard_w), and then pressing **Enter**.

Notes:

1. The NetBIOS name of the TCP/IP application is specified at the time the TCP/IP stack is initialized, which is usually in the hostName member of the tcpip_stack_init.c:: TCPIP_HOSTS_CONFIGURATION structure. The NetBIOS service must be enabled for the PIC32 demonstration to respond to NetBIOS queries. Alternatively, you can use the IPv4 or IPv6 address (if IPv6 is enabled) of the board directly, for example, http://192.168.1.131 or http://[fdfe:dcba:9876:1:204:a3ff:fe12:128e].
2. The IPv4 and IPv6 addresses can be obtained from running the TCP/IP Discovery application on the PC side. It requires that the TCP/IP Announce module is enabled when building the stack.

Please refer to the **Demonstration Process** in the Running the Demonstration section for the pic32_eth_web_server configuration, as the process is the same for this configuration.

Note: Refer to Wi-Fi Console Commands for information on the commands that enable control over the Wi-Fi settings.



web_server_sdcard_fatfs

Before using this demonstration, please see the important notes in the *TCP/IP Demonstrations* > *Introduction*.

Web Server SD Card FAT File System TCP/IP demonstration.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Web Server SD Card FAT FS Demonstration.

Description

To build this project, you must open the `pic32_eth_web_server.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/tcpip/web_server_sdcard_fatfs`.

Warning

When using the Microchip Harmony Configurator (MHC), care must be taken when generating the code to *not* erase the USB descriptors in the `system_init.c` file.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
pic32_eth_web_server.X	<install-dir>/apps/tcpip/web_server_sdcard_fatfs/firmware

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within ./firmware/src/system_config.

Project Configuration Name	BSP Used	Description
pic32mx_eth_sk2_sd_mmc_pictail	pic32mx_eth_sk2	Demonstrates the Web Server hosted on a microSD card through the FAT file system on the PIC32 Ethernet Starter Kit II and the PICtail Daughter Board for SD and MMC.
pic32mz_ef_sk_meb2	pic32mz_ef_sk+meb2	Demonstrates the Web Server hosted on a microSD card through the FAT file system on the PIC32MZ EF Starter Kit and the MEB II combination.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 Ethernet Starter Kit with the Starter Kit I/O Expansion Board and PICtail Daughter Board for SD and MMC

- Plug the PIC32 Ethernet Starter Kit into application board connector (J1) on the Starter Kit I/O Expansion Board
- Plug the PICtail Daughter Board for SD and MMC into the PICtail connector (J4) on the Starter Kit I/O Expansion Board
- Make sure a SD card is formatted and loaded with the web pages provided within the apps/tcpip/web_server_sdcard_fatfs/firmware/src/web_pages2_sdcard folder
- Insert the SD card with the web pages into the SD card slot (J1) on the PICtail Daughter Board for SD and MMC into the PICtail connector
- Connect a USB cable to the micro-B USB connector on the bottom of the PIC32 Ethernet Starter Kit. When the demonstration runs, it will create a USB CDC device on the USB bus. Connect to this device though a standard terminal program, and set the baud rate to 921,600 baud. You can observe the IP address details and query the stack using the console interface.

PIC32 Ethernet Starter Kit II with the Starter Kit I/O Expansion Board and PICtail Daughter Board for SD and MMC

- Plug the PIC32 Ethernet Starter Kit II into application board connector (J1) on the Starter Kit I/O Expansion Board
- Plug the PICtail Daughter Board for SD and MMC into the PICtail connector (J4) on the Starter Kit I/O Expansion Board
- Make sure a SD card is formatted and loaded with the web pages provided within the apps/tcpip/web_server_sdcard_fatfs/firmware/src/web_pages2_sdcard folder
- Insert the SD card with the web pages into the SD card slot (J1) on the PICtail Daughter Board for SD and MMC into the PICtail connector
- Connect a USB cable to the micro-B USB connector on the bottom of the PIC32 Ethernet Starter Kit II. When the demonstration runs, it will create a USB CDC device on the USB bus. Connect to this device though a standard terminal program, and set the baud rate to 921,600 baud. You can observe the IP address details and query the stack using the console interface.

PIC32MZ EC Starter Kit or PIC32MZ EF Starter Kit with the MEB II

- Plug the desired starter kit into the application board connector on the MEB II
- Ensure a microSD card is formatted and loaded with the web pages provided within the apps/tcpip/web_server_sdcard_fatfs/firmware/src/web_pages2_sdcard directory.
- Insert the microSD card with the web pages into the microSD card slot (J8) on the MEB II
- Connect a USB cable to the micro-B USB connector on the bottom of the PIC32MZ EC Starter Kit
- When the demonstration runs, it will create a USB CDC device on the USB bus. Connect to this device though a standard terminal program, and set the baud rate to 921600 baud. You can observe the IP address details and query the stack using the console interface.

Running the Demonstration

This section provides instructions on how to build and run the TCP/IP SD Card FAT FS Web Server demonstration.

Description

To view the web page hosted by the demonstration application, open a web browser and direct it to the board running the HTTP server by typing the URL in the address bar (for example, http://mchpboard_e or http://mchpboard_e), and then pressing **Enter**.

Notes:

1. The NetBIOS name of the TCP/IP application is specified at the time the TCP/IP stack is initialized, which is usually in the hostName member of the tcpip_stack_init.c:: TCPIP_HOSTS_CONFIGURATION structure. The NetBIOS service must be enabled for the PIC32 demonstration to respond to NetBIOS queries. Alternatively, you can use the IPv4 or IPv6 address (if IPv6 is enabled) of the board directly, for example, http://192.168.1.131 or http://[fdfe:dcba:9876:1:204:a3ff:fe12:128e].
2. The IPv4 and IPv6 addresses can be obtained from running the TCP/IP Discovery application on the PC side. It requires that the TCP/IP Announce module is enabled when building the stack.

Please refer to the **Demonstration Process** in the Running the Demonstration section for the pic32_eth_web_server configuration of the web_server_nvm_mpfs demonstration, as the process is the same for this configuration.

wifi_ap_demo

Before using this demonstration, please see the important notes in the *TCP/IP Demonstrations* > *Introduction*.

The Wi-Fi AP mode demonstration showcases how to start a PIC32WK Wi-Fi module in AP mode and connect with a third-party STA.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the PIC32WK Wi-Fi AP mode demonstration.

Description

To build this project, you must open the wifi_ap_demo.X project in MPLAB X IDE, and then select the desired configuration. The following tables list and describe the project and supported configurations. The parent folder for these files is

`<install-dir>/apps/tcpip/wifi_ap_demo.`

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`wifi_ap_demo.X`	`<install-dir>/apps/tcpip/wifi_ap_demo/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are within

`./firmware/src/system_config.`

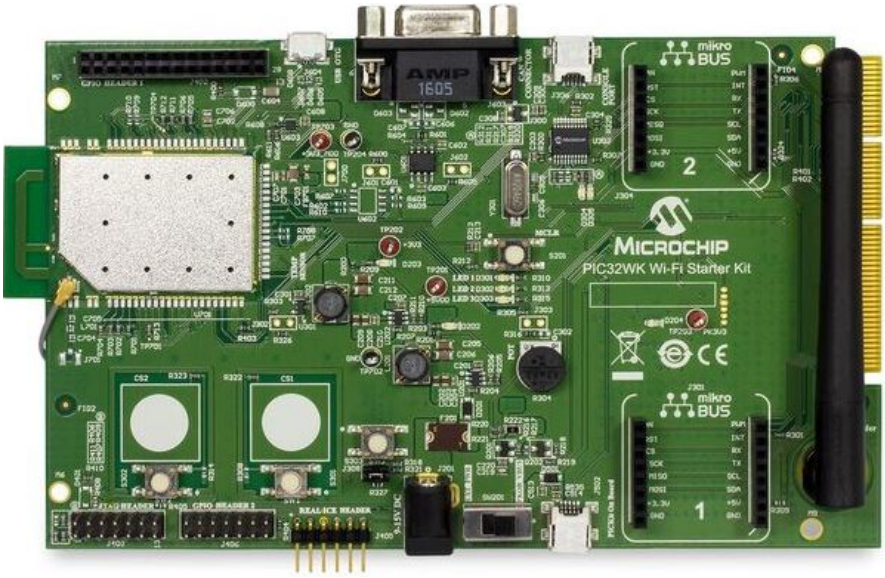
Project Configuration Name	BSP(s) Used	Description
pic32wk_ap	pic32wk_gpb_gpd_sk+module	Demonstrates the AP functionality with WM32 Wi-Fi Starter Kit.

Configuring the Hardware

This section describes how to configure the supported hardware.

Description

The following is the PIC32WK Wi-Fi Starter Kit mounted with Wi-Fi module (WM32S2057GXEAS-I/RM)



The Starter Kit has a UART to USB converter (MCP2200) that enables communication between a host PC and the Wi-Fi module. This is done by connecting a mini-B USB cable from the host PC to the USB port (J306) in the Starter Kit.

The console output uses the UART to USB converter on the board at 230400 baud, 8-bit data, No parity, 1 Stop bit, with No Flow Control.

Running the Demonstration

This section provides instructions about how to build and run the PIC32WK Wi-Fi AP mode demonstration with the WM32 module.

Description

The demonstration does the following:

- Starts the device in AP mode, with the configuration stored in flash memory.
- PIC32WK Wi-Fi AP accepts connections from another third-party STA and accepts up to eight connections.
- The AP can be started in different modes: Open and Security (WPA and WPA2)
- The default configuration of the AP is stored in flash memory, and this configuration can be updated with console commands.
- The connected STA can verify the connection status by using PING process.
- In the connected STA, you can access the AP IP-address in any browser that serves the Web pages.
- That web page will display the Welcome screen along with a random seed number on the page.
- From the connected STA, you can scan for the wireless networks to see the list of APs in the BSS.

Use the following procedure to run the demonstration:

1. Load the demonstration project into MPLAB X IDE.

2. Connect the mini-B debugger port on-the starter kit to a USB port on the development computer by using the USB cable provided in the kit.

3. Build, download, and run the demonstration project on the board.

- The demonstration starts an AP, sends beacons in the given channel, and responds with the Probe Response for the Probe Request received from other STA devices.

4. Run **netinfo** to display the following network details.

```
---------- Interface <wlan0/PIC32WK> ----------
Host Name: MCHPBOARD_WK - NBNS disabled
IPv4 Address: 192.168.3.1
Mask: 255.255.255.0
Gateway: 192.168.3.1
DNS: 192.168.3.1
MAC Address: 54:8c:a0:00:f3:4e
dhcps is ON
dhcp is disabled
Link is UP
```

5. Run **wlan open** to open the access to give Wi-Fi commands, and then run **wlan macinfo** to see the Wi-Fi module status.

```
----WIFI MAC configuration----
BootMode: AP
HTTP: Enable
OTA: Enable
```

```
SSID: AP_PINGDEMO
Security: NONE
The above Configuration is Saved in Flash
************************************************
----MAC Started as AP with following Details----
SSID from MAC: AP_PINGDEMO
Security: NONE
```

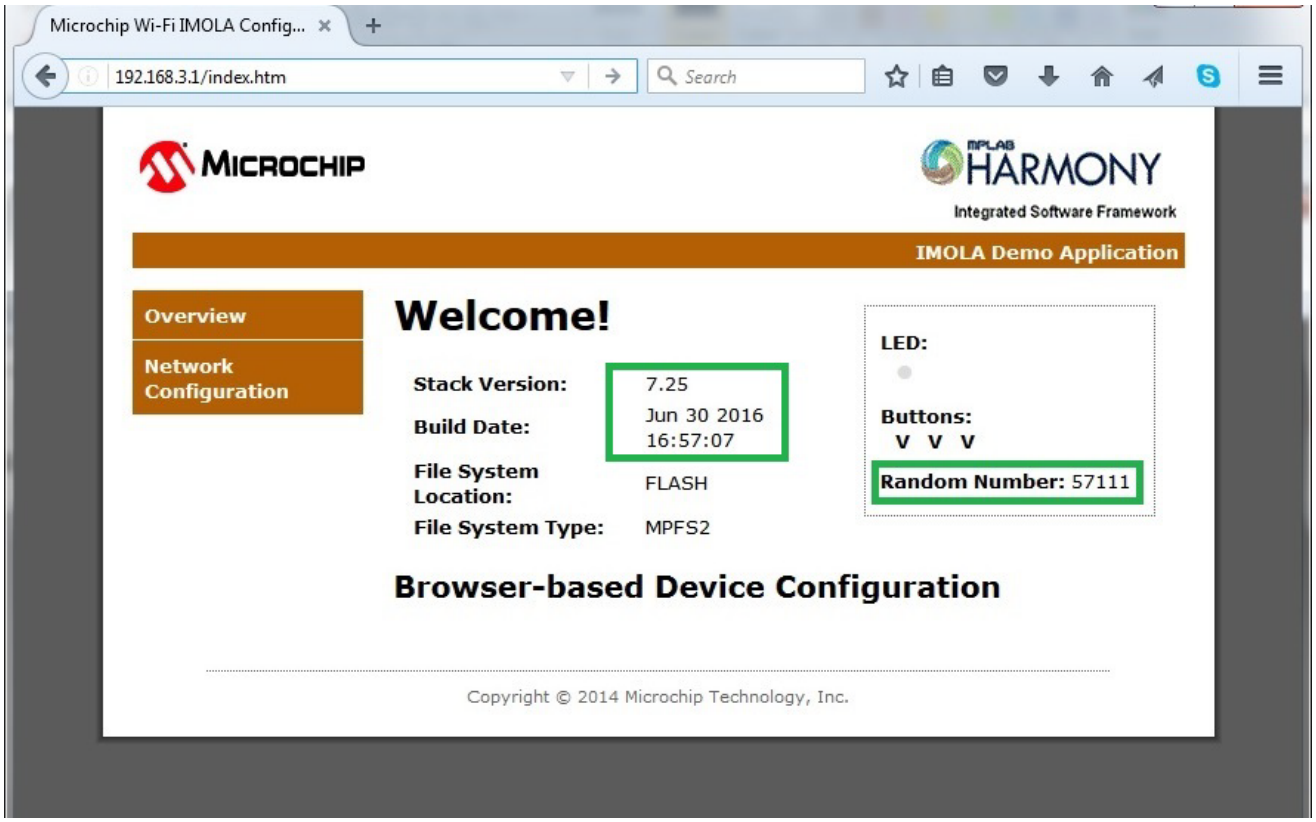
- The AP starts with IP address as 192.168.3.1

6. Use console commands to update the AP details.

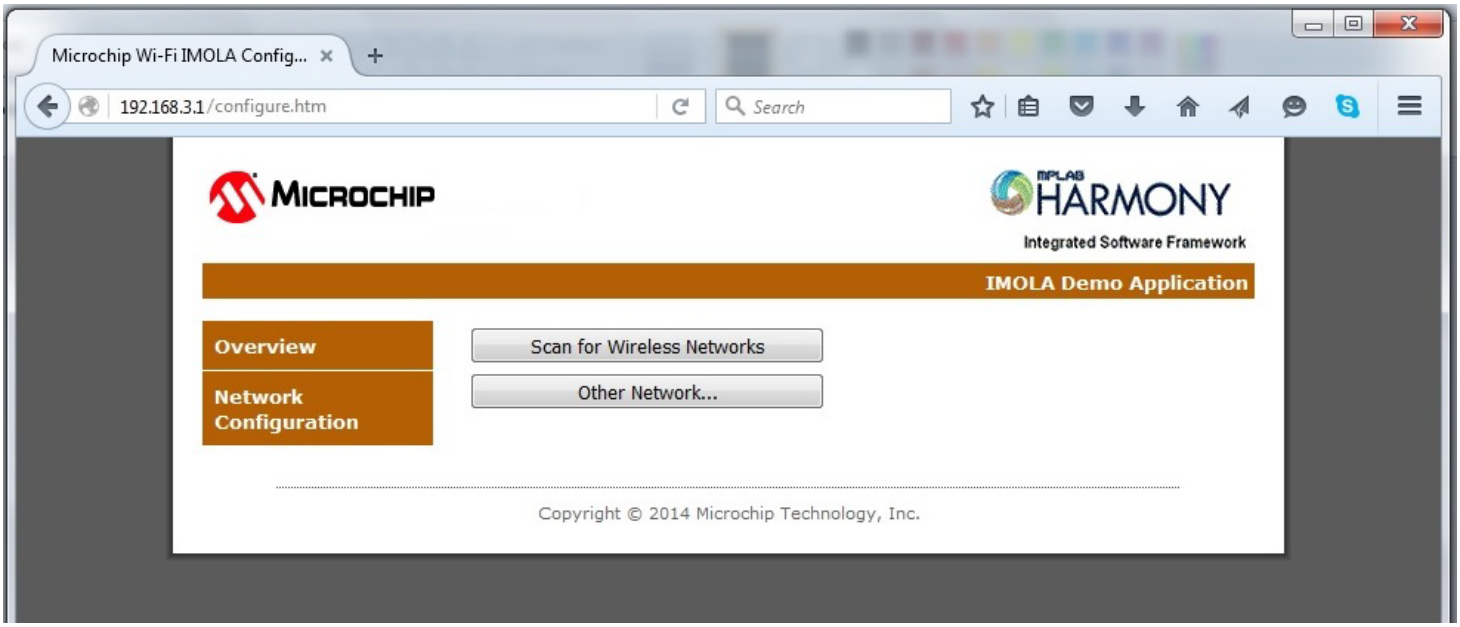
7. After connecting the third-party personal computer/laptop to the DUT AP, run **ping** to verify the simple data connection process between the DUT and the computer/laptop.

8. Enter the IP-address of the AP in the browser to get served with Web pages.

- The default Web-page as shown below gives firmware build details and a random number.



- The Network Configuration page gives the option to trigger the scan in the Wi-Fi module. After a successful scan, the page updates with the scan results.



wifi_easy_configuration

Wi-Fi Easy Configuration TCP/IP demonstration.

Description

This demonstration shows how to connect a MRF24WN or WINC1500 Wi-Fi device with no keyboard or display to a wireless network.

Before using this demonstration, please see the important notes in the *TCP/IP Demonstrations* > *Introduction*.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the PIC32 Wi-Fi Easy Configuration Demonstration.

Description

To build this project, you must open the `wifi_easy_configuration.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/tcpip/wifi_easy_configuration`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`wifi_easy_configuration.X`	`<install-dir>/apps/tcpip/wifi_easy_configuration/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx795_pim_e16_11n_freertos	pic32mx795_pim+e16	FreeRTOS version of the demonstration running on the Explorer 16 Development Board with PIC32MX795F512L CAN-USB PIM and the MRF24WN PICtail Daughter Board.
pic32mx795_pim__e16__winc__freertos	pic32mx795_pim+e16	FreeRTOS version of the demonstration running on the Explorer 16 or Explorer 16/32 Development Board, or the Explorer 16/32 Development Board with the PIC32MX795F512L CAN-USB PIM, PICtail Plus Expansion Board, and the WINC1500 Wi-Fi PICtail/PICtail Plus Daughter Board.

pic32mz_ef_sk_winc_freertos	pic32mz_ef_sk	FreeRTOS version of the demonstration running on the PIC32MZ EF Starter Kit connected to the I/O Expansion board with the WINC1500 PICtail Daughter Board.
pic32mz_ef_sk_ioexp_11n_freertos	pic32mz_ef_sk	FreeRTOS version of the demonstration running on the PIC32MZ EF Starter Kit connected to the I/O Expansion board with the MRF24WN PICtail Daughter Board.
pic32mz_ef_curiosity	pic32mz_ef_curiosity	FreeRTOS version of the demonstration running on the PIC32MZ EF Curiosity Development board, with on-board MRF24WN module. This configuration is generated for standalone mode. All necessary files are copied under its configuration folder, and so it can be built and run without the MPLAB Harmony framework.
pic32mx795_pim__e16__wincclick__freertos	pic32mx795_pim+e16	FreeRTOS version of the demonstration running on the Explorer 16 or Explorer 16/32 Development Board, or the Explorer 16/32 Development, PIC32MX795F512L CAN-USB PIM, and PICtail Plus Expansion Board with the MikroElektronica WiFi 7 Click Board (with on-board WINC1500 Wi-Fi module).

Configuring the Hardware

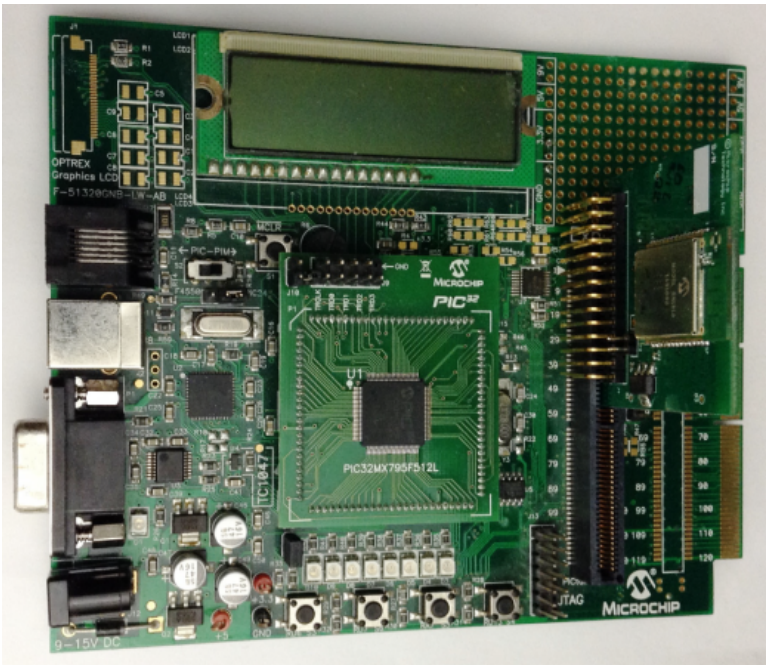
Describes how to configure the supported hardware.

Description

PIC32MX795F512L CAN-USB PIM with the Explorer 16 Development Board and MRF24WN PICtail Daughter Board

- Connect the MRF24WN PICtail Daughter Board into the PICtail Plus slot closest to the MCU. The module should face into the board and be placed all the way to the end on the pin 1 side of the connector.
- The console output uses the DB9 connector on the board at 115200 baud, 8-bit data, No parity, 1 Stop bit, with No Flow Control

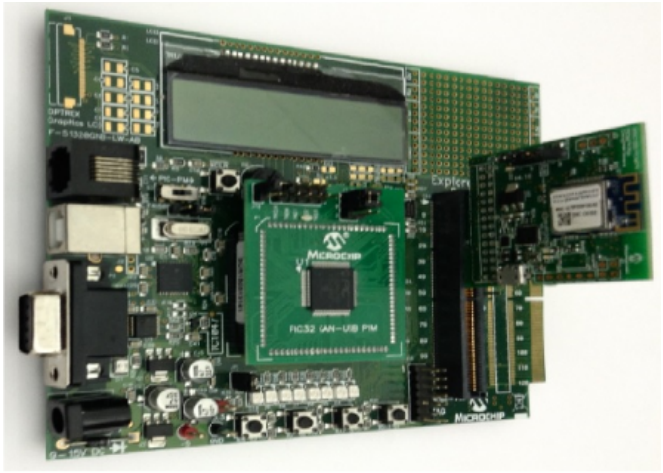
The following figure shows the hardware configuration.



PIC32MX795F512L CAN-USB PIM with theExplorer 16 Development Board and WINC1500 Wi-Fi PICtail/PICtail Plus Daughter Board

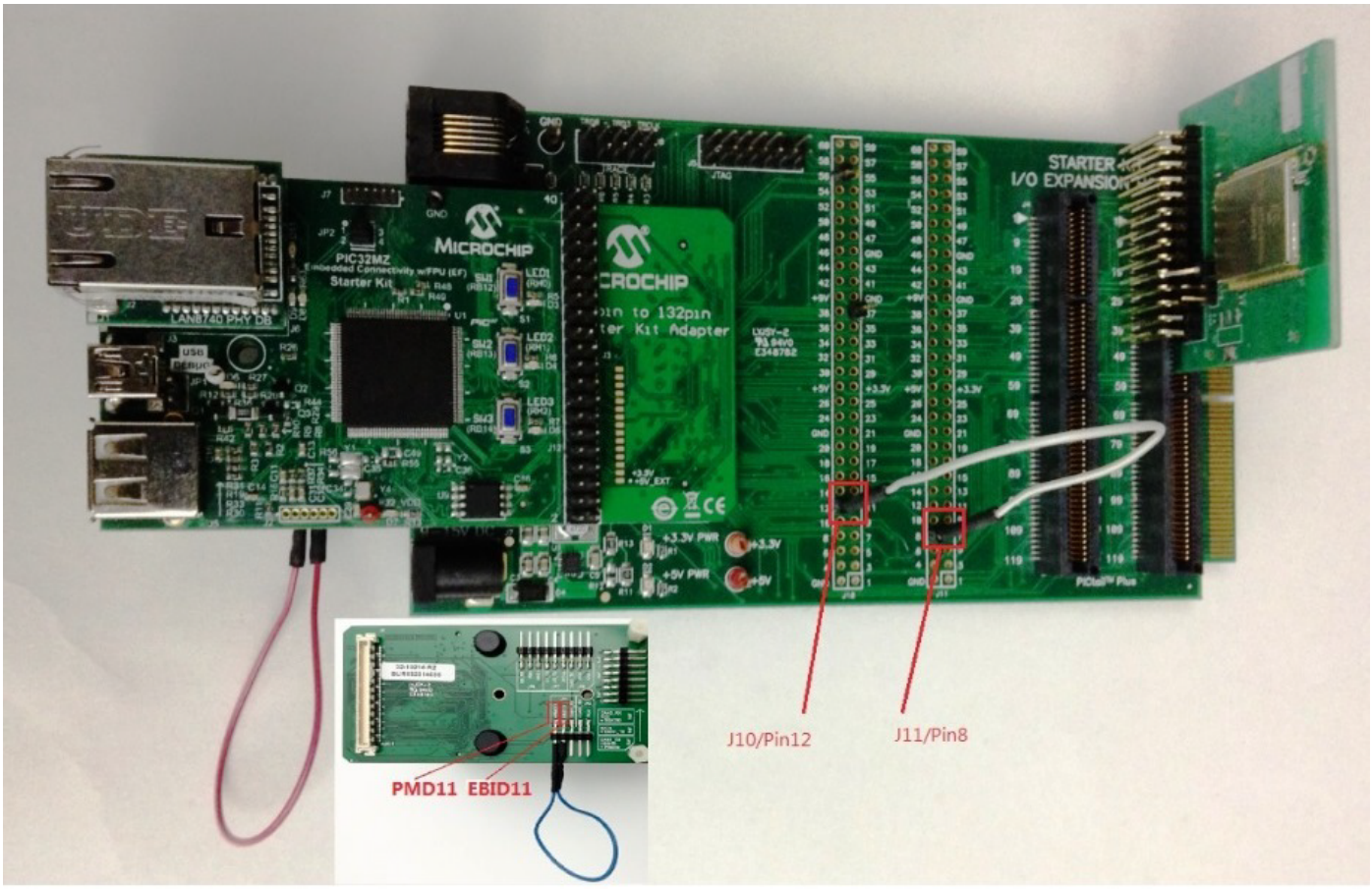
- Connect the WINC1500 PICtail Daughter Board into the PICtail Plus slot closest to the MCU. The module should face into the board and be placed all the way to the end on the pin 1 side of the connector.
- The console output uses the DB9 connector on the board at 115200 baud, 8-bit data, No parity, 1 Stop bit, with No Flow Control

The following figure shows the hardware configuration.



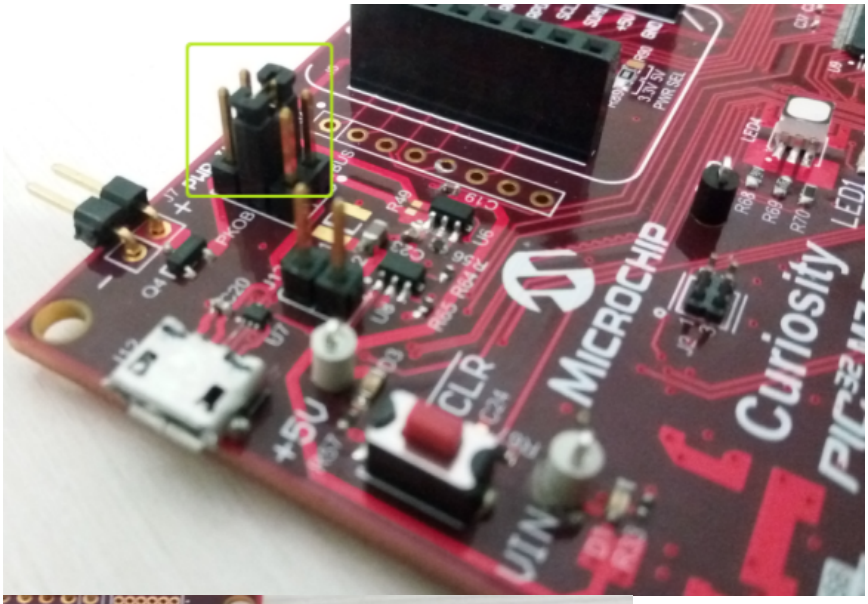
PIC32MZ EF Starter Kit with PIC32MZ2048ECH144, MRF24WN0MA Wi-Fi PICtail/PICtail Plus Daughter Board, and Starter Kit I/O Expansion Board

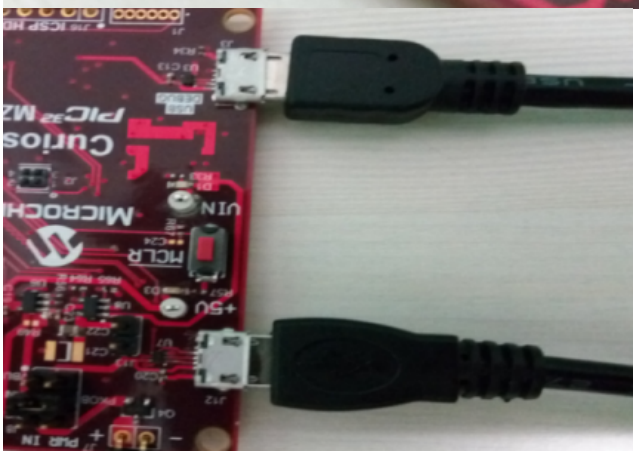
Configure the hardware, as shown in the following figure:



PIC32MZ EF Curiosity Development Board with on-board MRF24WN0MA module.

- Ensure that a jumper is placed at 4-3 on J8, to select supply from debug USB connector.
- Power the PIC32MZ EF Curiosity Development Board from a Host PC through a Type-A male to micro-B USB cable connected to Micro-B port (J3).
- Ensure that jumper is not present in the J13 header to use the Curiosity board in device mode.
- Plug in a USB cable with a micro-B type connector to Micro-B port (J12), and plug the other end into your computer.

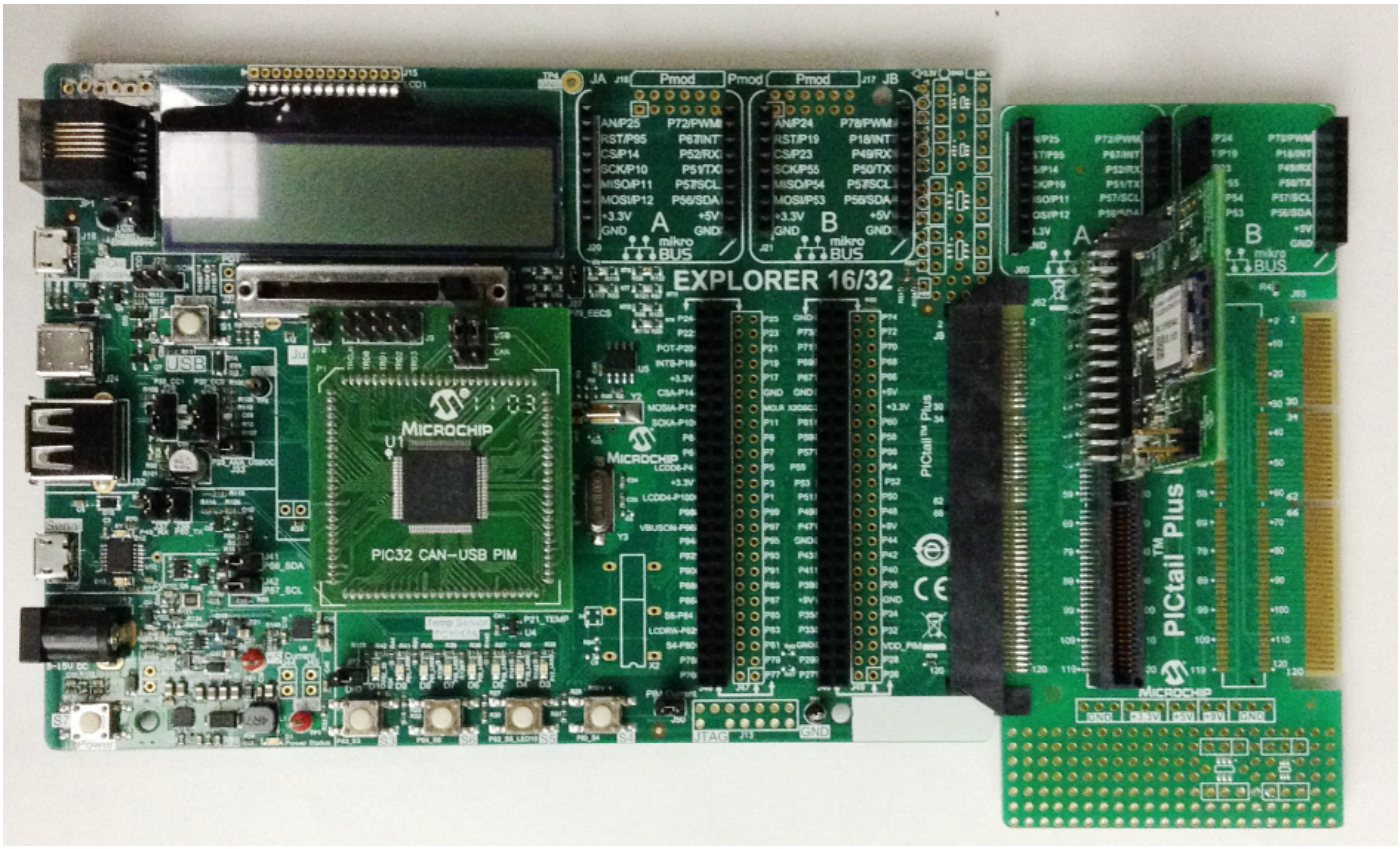




Explorer 16/32 Development Board with the PIC32MX795F512L CAN-USB PIM, PICtail Plus Expansion Board, and WINC1500 Wi-Fi PICtail/PICtail Plus Daughter Board

- Connect the WINC1500 Wi-FI PICtail/PICtail Plus Daughter Board into the PICtail Plus slot (J63) . The module should face into the main board and be placed all the way to the end on the pin 1 side of the connector.
- The console output uses the mini_USB/Serial connector on the board (J40) at 115200 baud, 8-bit data, No parity, 1 Stop bit, with No Flow Control.

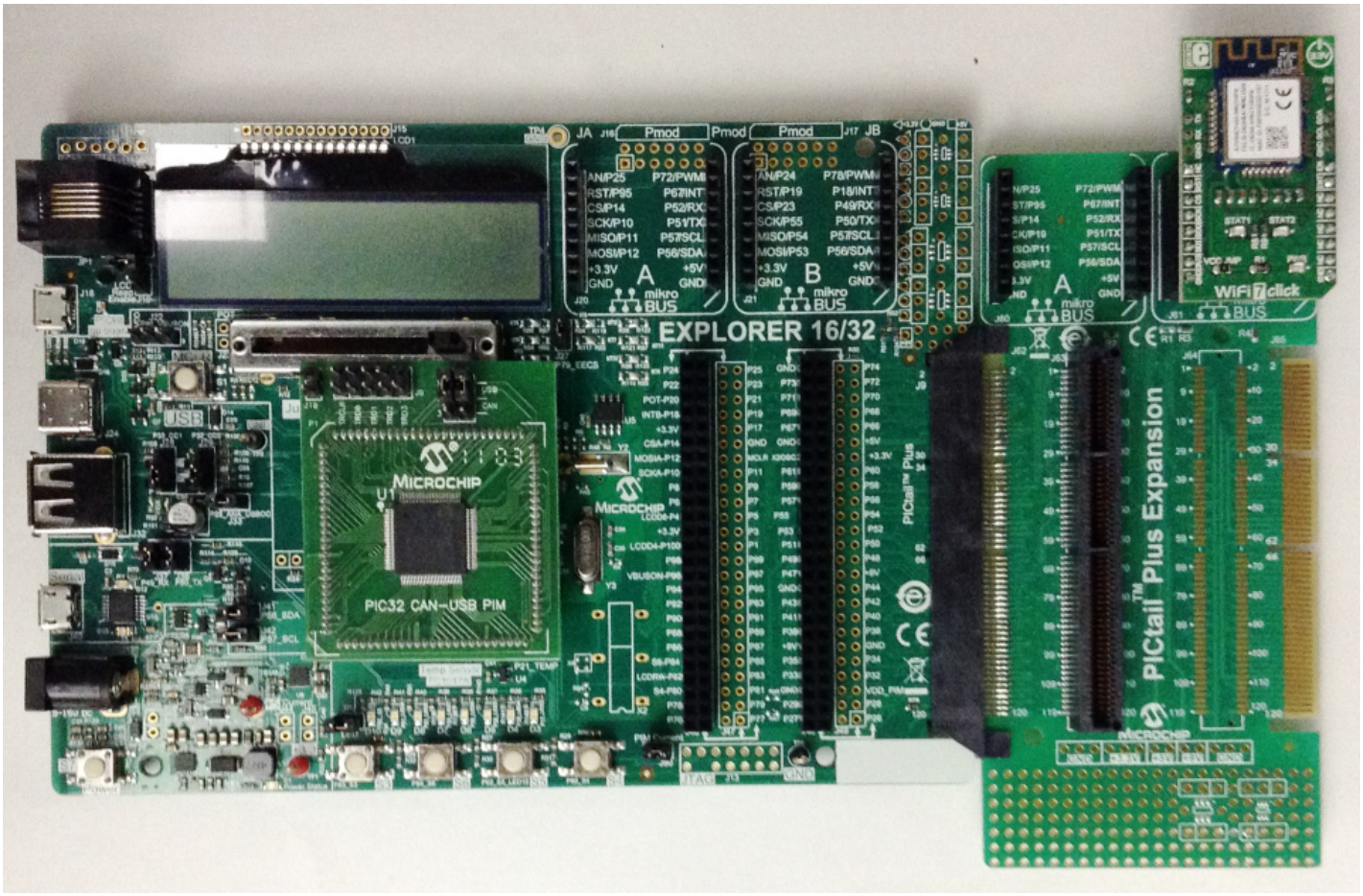
The following figure shows the hardware configuration.



Explorer 16/32 Development Board with the PIC32MX795F512L CAN-USB PIM, PICtail Plus Expansion Board, and the MikroElektronika WiFi 7 Click Board (with on-board WINC1500 Wi-Fi module).

- Connect the WINC1500 Wi-Fi PICtail/PICtail Plus Daughter Board into the PICtail Plus slot (J63) . The module should face into the main board and be placed all the way to the end on the pin 1 side of the connector.
- The console output uses the mini_USB/Serial connector on the board (J40) at 115200 baud, 8-bit data, No parity, 1 Stop bit, with No Flow Control

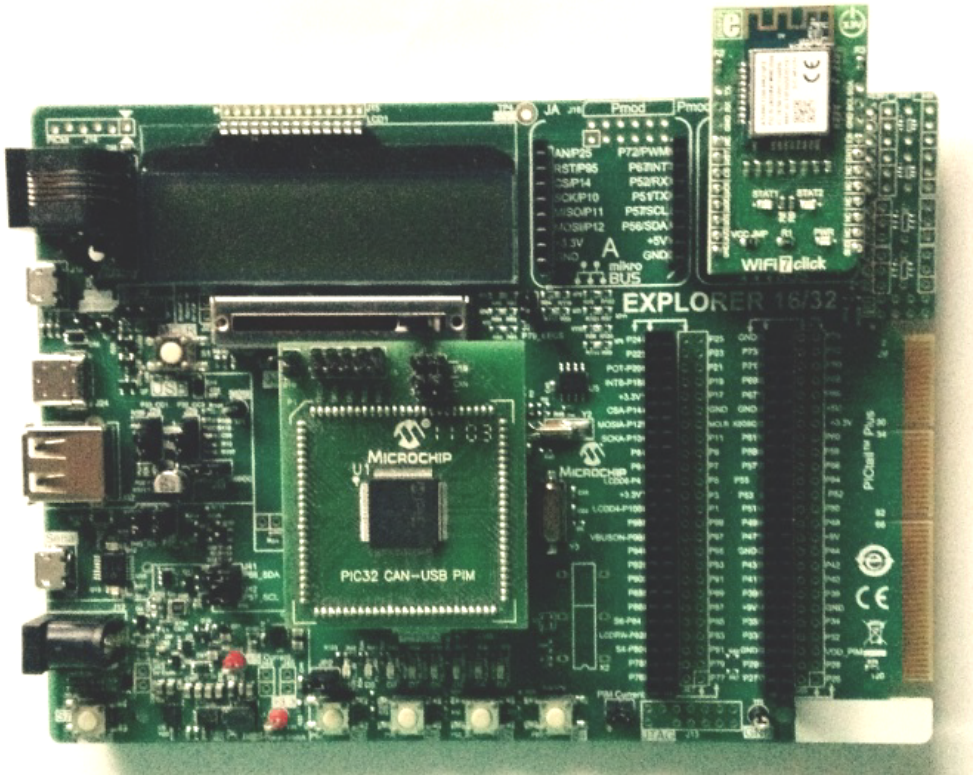
The following figure shows the hardware configuration.



Explorer 16/32 Development Board with the PIC32MX795F512L CAN-USB PIM and the MikroElektronika WiFi 7 Click Board (with on-board WINC1500 Wi-Fi module).

- Connect the WINC1500 Wi-Fi PICtail/PICtail Plus Daughter Board into the PICtail Plus slot. The module should face into the main board and be placed all the way to the end on the pin 1 side of the connector.
- The console output uses the mini_USB/Serial connector on the board (J40) at 115200 baud, 8-bit data, No parity, 1 Stop bit, with No Flow Control

The following figure shows the hardware configuration.



Running the Demonstration

This section provides instructions on how to build and run the Wi-Fi Easy Configuration demonstration with the MRF24WN or WINC1500 Wi-Fi module.

Description

Notes:

1. Refer to Wi-Fi Console Commands for information on the commands that enable control over the Wi-Fi settings.
2. WINC1500 SoftAP mode can support only one Station client connection.

The demonstration does the following:

- Scans the area and stores the list of Access Points in memory
- Switches to SoftAP mode allowing another device to connect to it (smartphone or personal computer)
- After a smartphone or personal computer wirelessly connects to the MRF24WN or WINC1500, a web page is served to it
- That web page will display the access point (AP) list (from step 1).
- From the smartphone you can select the desired AP and command the MRF24WN or WINC1500 to connect to that AP
- The MRF24WN or WINC1500 will connect to the selected AP and store the configuration information in non-volatile memory

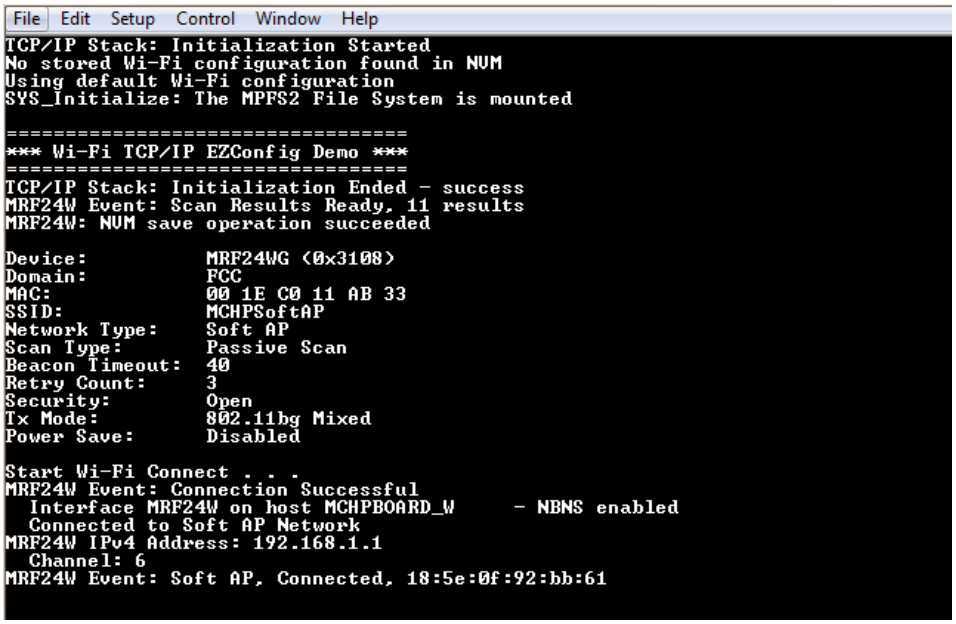
Use the following procedure to run the demonstration:

1. Load the demonstration project into MPLAB X IDE.

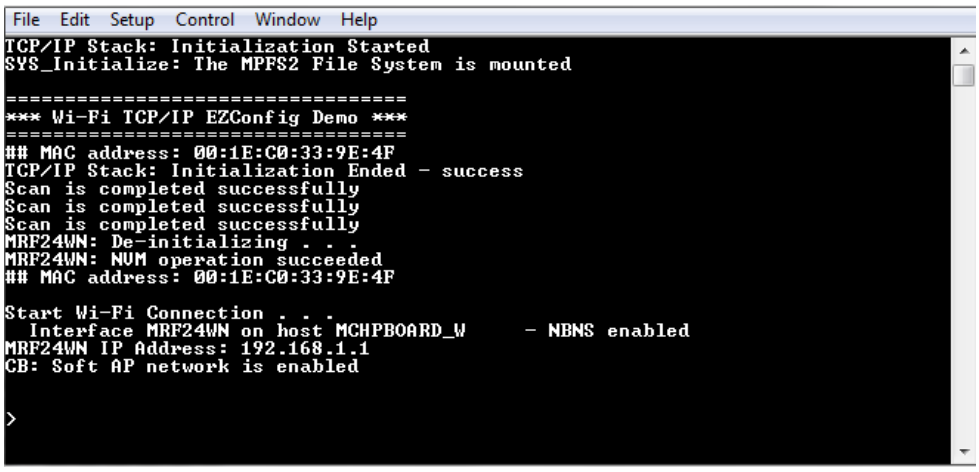
2. Connect the mini-B/micro B debugger port on-board the starter kit to a USB port on the development computer by using the USB cable provided in the kit.

3. Build, download, and run the demonstration project on the board.

4. When the demonstration runs, it scans for local access points and outputs the results to the serial console. After the scan results, the MRF24WN or WINC1500 goes into SoftAP mode (it behaves like an access point) and outputs the following to the serial console:



The output of the serial console for the MRF24WN module is shown as follows:



5. From a smartphone or personal computer, connect to the 'MCHPSoftAP' network, which is the SoftAP network started by the demonstration. Then, bring up a web page by entering the IP address of the SoftAP network into the smartphone browser. This is the IP address displayed in step 4 (e.g., 192.168.1.1). When the web page is displayed, on the right top corner of the page, there are three widgets to verify connectivity with the board.

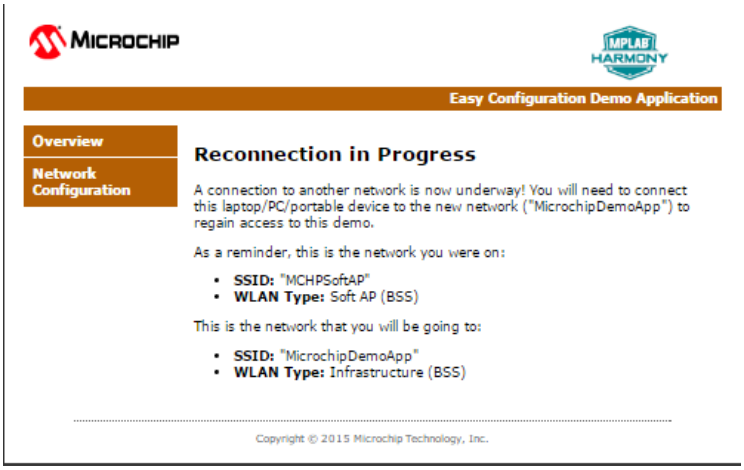
- LED - This is in blinking mode.
- Button - This changes position when the buttons on the target board is pressed.
- Random Number - This displays random numbers generated by the target board.

On the web page:

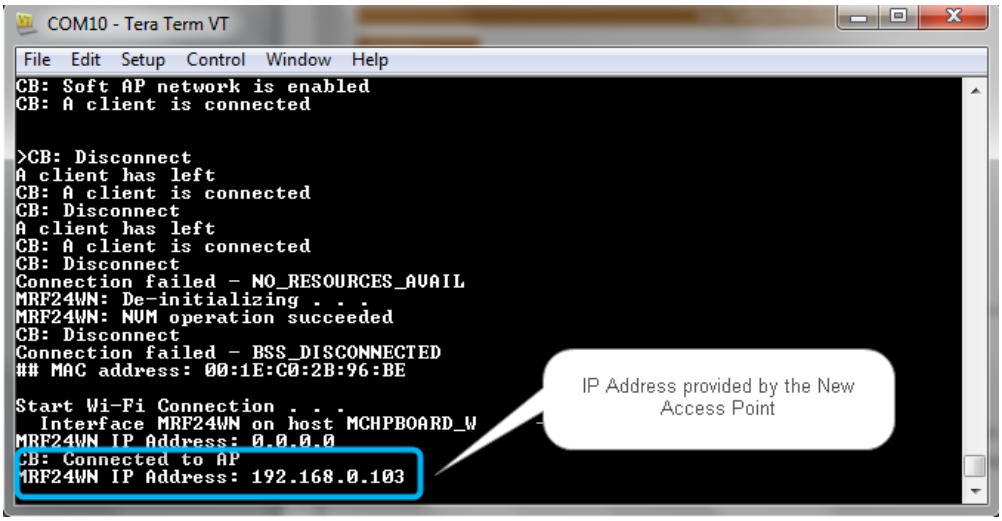
a. Select **Network Configuration**, and then **Scan for Wireless Networks**. The MRF24WN or WINC1500 will display the list of wireless networks on the web page.

b. Select the desired AP to which the MRF24WN or WINC1500 should connect by clicking the name of the AP.

c. The MRF24WN or WINC1500 will then connect to that Access Point and write the configuration information to non-volatile memory.



d. The console output will show the new connection taking place.



6. If you rerun the demonstration, it will automatically connect the selected AP, as the configuration data stored in non-volatile memory will be used to reconnect to the desired AP.

a. Connect the PC to the same AP to which the board is connected.

b. Open the serial terminal as described in step 2, and get the IP address that the board had gotten from the newly selected AP.

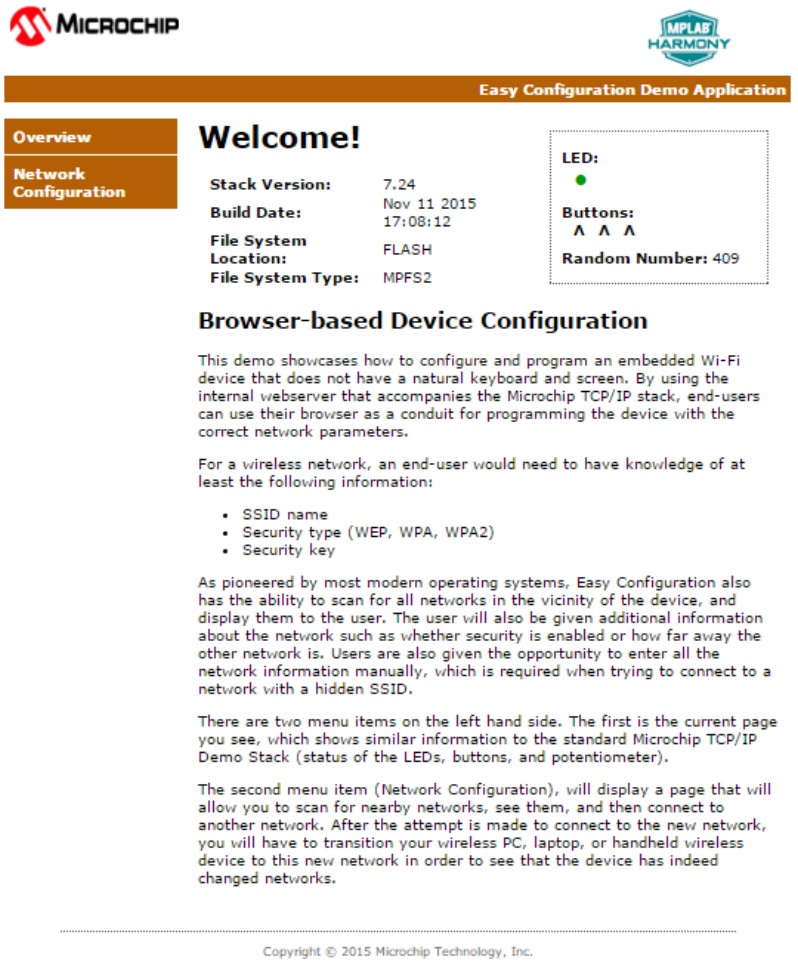
c. Type this IP address into the browser and the demo web page should come up on the browser.

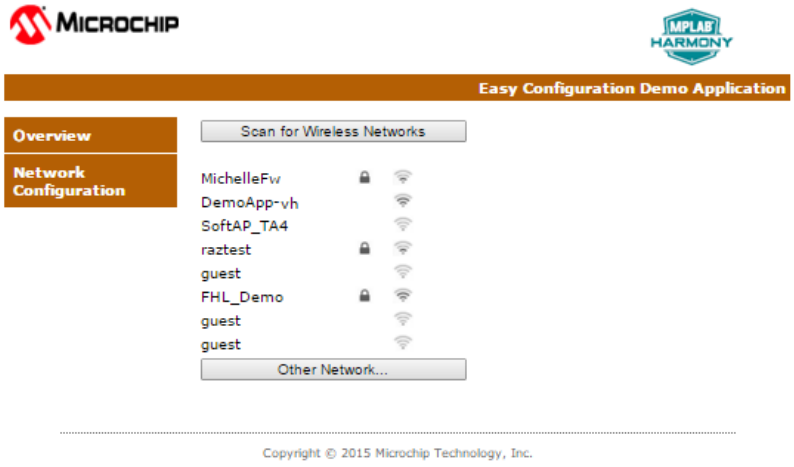
d. Try monitoring the LED and Switch through the newly established connection.

7. To reset and run the demonstration from the beginning:

a. Erase the stored configuration by bringing up the demonstration.

b. At the command line type **deleteconf**





wifi_rgb_easy_configuration

Before using this demonstration, please see the important notes in the *TCP/IP Demonstrations* > *Introduction*.

The WiFi RGB Easy Configuration demo showcases how to configure an embedded WiFi device that does not have a natural keyboard and screen. By using the internal Web server that accompanies the Microchip TCP/IP stack, end-users can use their browser as a conduit for programming the WiFi device with the correct network parameters.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the PIC32 Wi-Fi RGB

Easy Configuration Demonstration.

Description

To build this project, you must open the wifi_rgb_easy_configuration.X project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/tcpip/wifi_rgb_easy_configuration`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`wifi_rgb_easy_configuration.X`	`<install-dir>/apps/tcpip/wifi_rgb_easy_configuration/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mz_ef_curiosity	pic32mz_ef_curiosity	FreeRTOS version of the demonstration running on the PIC32MZ EF Curiosity Development board, with on-board MRF24WN module. This configuration is generated for standalone mode. All necessary files are copied under its configuration folder, and so it can be built and run without the MPLAB Harmony framework.

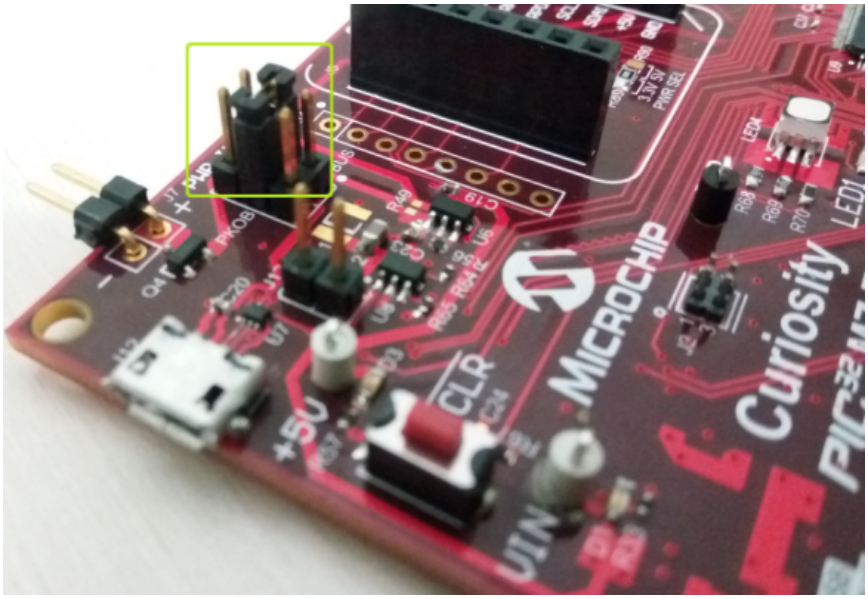
Configuring the Hardware

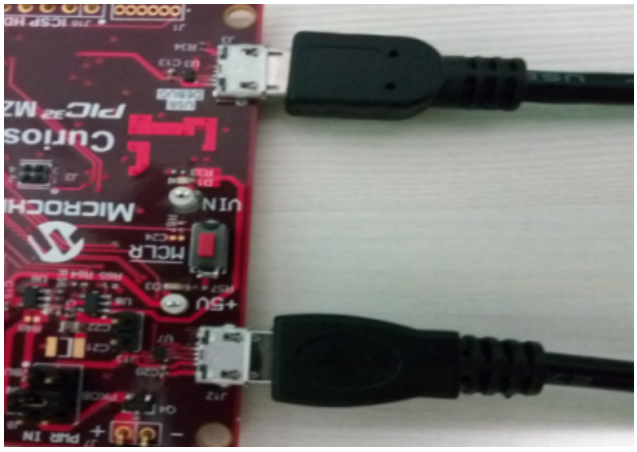
Describes how to configure the supported hardware.

Description

PIC32MZ EF Curiosity Development Board with on-board MRF24WN0MA module.

- Ensure that a jumper is placed at 4-3 on J8, to select supply from debug USB connector.
- Power the PIC32MZ EF Curiosity Development Board from a Host PC through a Type-A male to micro-B USB cable connected to Micro-B port (J3).
- Ensure that jumper is not present in the J13 header to use the Curiosity board in device mode.
- Plug in a USB cable with a micro-B type connector to Micro-B port (J12), and plug the other end into your computer.





Running the Demonstration

This section provides instructions on how to build and run the Wi-Fi RGB Easy Configuration demonstration with the MRF24WN Wi-Fi module.

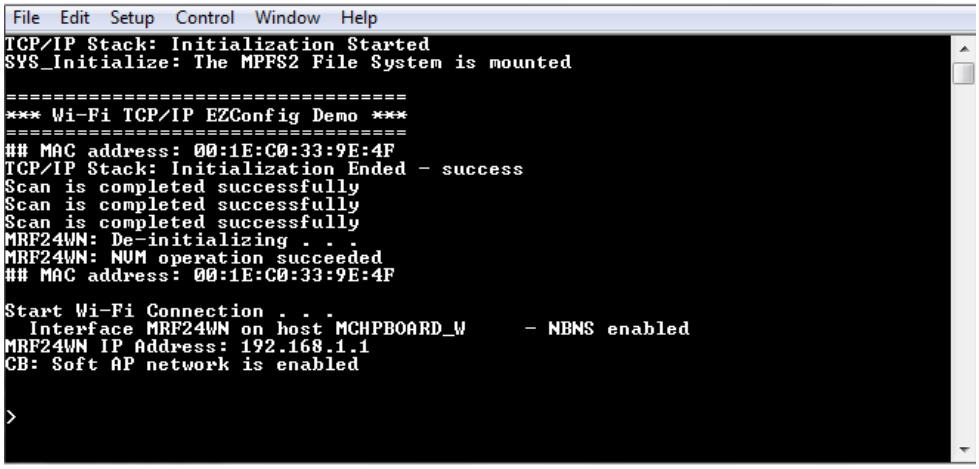
Description

1. Load the demonstration project into MPLAB X IDE.

2. Connect the USB debugger port on-board the starter kit to a USB port on the development computer using the USB cable provided in the kit.

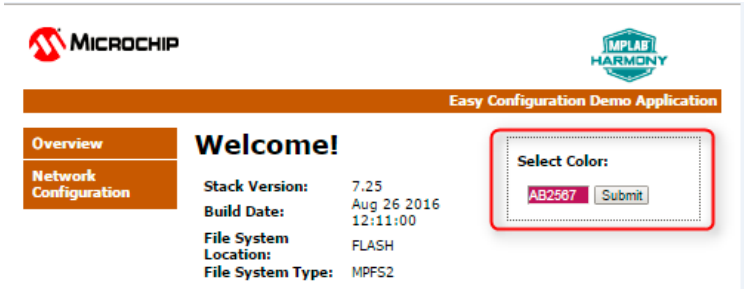
3. Build, download, and run the demonstration project on the board.

4. When the demonstration runs, it scans for local access points and outputs the results to the serial console. After the scan results, the MRF24WN goes into SoftAP mode (it behaves as an access point) and outputs the following to the serial console:

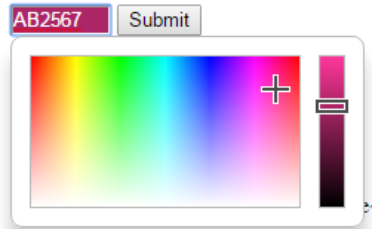


5. From a smartphone or personal computer, connect to the Curiosity_RGBLED_AP network, which is the SoftAP network started by the demonstration. Then, bring up a web page by entering the IP address of the SoftAP network into the smartphone browser. This is the IP address displayed in step 2 (e.g., 192.168.1.1). When the web page is displayed:

a. On the right top corner of the page, there is a widget through which we can change the color of the RGB LED which is on the Curiosity board.



Note: Click the text box to display a pop-up for selecting the color of the RGB LED. Click the **Submit** button to set the color of the RGB LED on the Curiosity board.

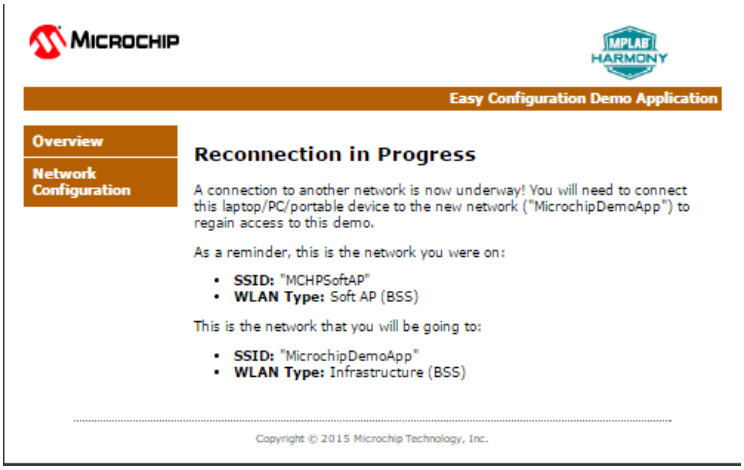


b. Select **Network Configuration**, and then **Scan for Wireless Networks**. The MRF24WN will display the list of wireless networks on the web page.

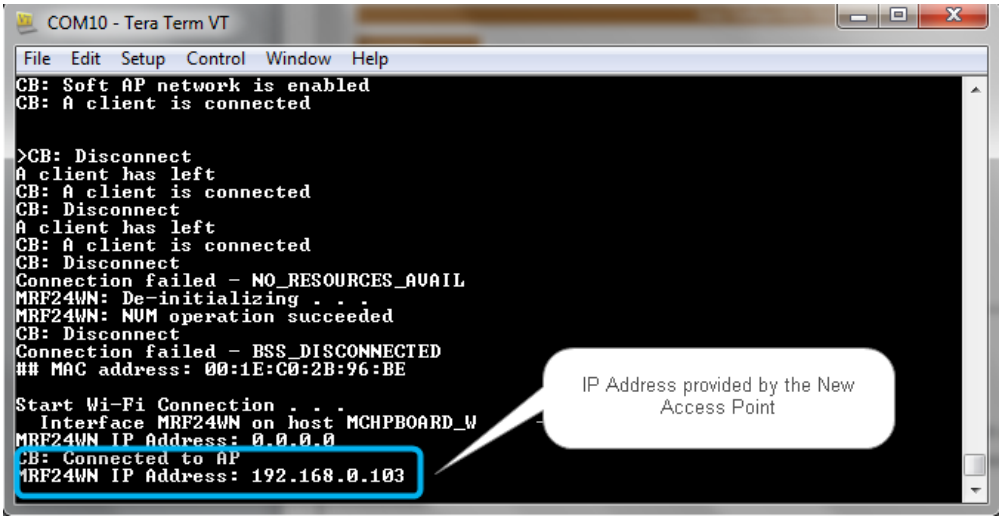


c. Select the desired access point (AP) to which the MRF24WN should connect by clicking the name of the AP. The MRF24WN will then connect to that AP and write the configuration information to non-volatile memory.

Note: This demo does not support connection to Secured APs. Make sure that the AP that you want to connect is unsecured.



d. The console output will show the new connection taking place.



wifi_sta_demo

Before using this demonstration, please see the important notes in the *TCP/IP Demonstrations* > *Introduction*.

The Wi-Fi Station (STA) mode demonstration showcases how to connect a PIC32WK Wi-Fi module with Home AP, checking connection status using PING process and discovering PIC32WK Wi-Fi module.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the PIC32WK Wi-Fi STA mode configuration demonstration.

Description

To build this project, you must open the wifi_sta_demo.X project in MPLAB X IDE, and then select the desired configuration. The following tables list and describe the project and supported configurations. The parent folder for these files is

`<install-dir>/apps/tcpip/wifi_sta_demo.`

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`wifi_sta_demo.X`	`<install-dir>/apps/tcpip/wifi_sta_demo/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are within

`./firmware/src/system_config.`

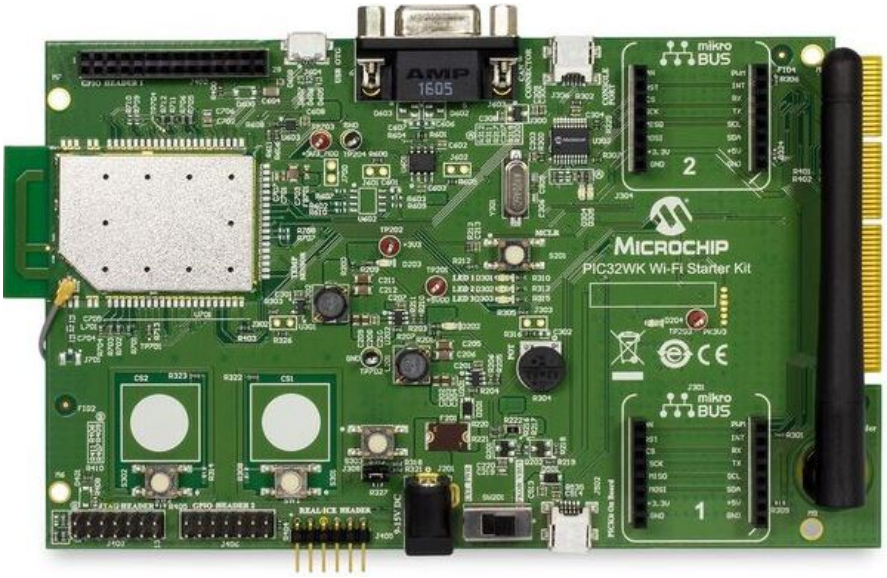
Project Configuration Name	BSP(s) Used	Description
pic32wk_sta	pic32wk_gpb_gpd_sk+module	Demonstrates basic STA functionality.
pic32wk_sta_freertos	pic32wk_gpb_gpd_sk+module	FreeRTOS version of the demonstration with basic STA functionality.

Configuring the Hardware

This section describes how to configure the supported hardware.

Description

The following is the PIC32WK Wi-Fi Starter Kit mounted with Wi-Fi module (WM32S2057GXEAS-I/RM)



The Starter Kit has a UART to USB converter (MCP2200) that enables communication between a host PC and the Wi-Fi module. This is done by connecting a mini-B USB cable from the host PC to the USB port (J306) in the Starter Kit.

The console output uses the UART to USB converter on the board at 230400 baud, 8-bit data, No parity, 1 Stop bit, with No Flow Control.

Running the Demonstration

This section provides instructions about how to build and run the PIC32WK Wi-Fi STA mode demonstration with the WM32 module.

Description

The demonstration does the following:

- Scans the area and triggers the connection with the Home AP.
- The default configuration of the home AP is stored in the Flash and this can be updated with console commands or by using the Web page.
- After connecting with the Home AP, connection status can be verified by running **ping**.
- After connecting the DUT and the third-party personal computer/laptop to the Home AP, the DUT can **ping** the third-party computer/laptop, and the computer/laptop can **ping** the DUT.
- The DUT also supports TCP/IP discovery, in which the WM32 module can be discovered by the Microchip TCPIP Discoverer tool on the third-party personal computer/laptop.

Use the following procedure to run the demonstration:

1. Load the demonstration project into MPLAB X IDE.

2. Connect the mini-B debugger port on the starter kit to a USB port on the development computer by using the USB cable provided in the kit.

3. Build, download, and run the demonstration project on the board.

- When the demonstration runs, it scans for APs and connects with the Home AP saved configuration. If the configured AP is not available in the scan list, the device will be in the scanning phase only.
- After connecting with the Home AP, the DUT (PIC32WK Wi-Fi) gets the IP-address from AP (if DHCP is enabled) or default IP-address, 192.168.1.150

4. Use console commands to get the connection status details. Run **netinfo** to display these connection details.

```
---------- Interface <wlan0/PIC32WK> ----------
Host Name: MCHPBOARD_WK - NBNS disabled
IPv4 Address: 192.168.1.101
Mask: 255.255.255.0
Gateway: 192.168.1.1
DNS: 192.168.1.1
MAC Address: 54:8c:a0:00:f3:82
dhcp is ON
Link is UP
```

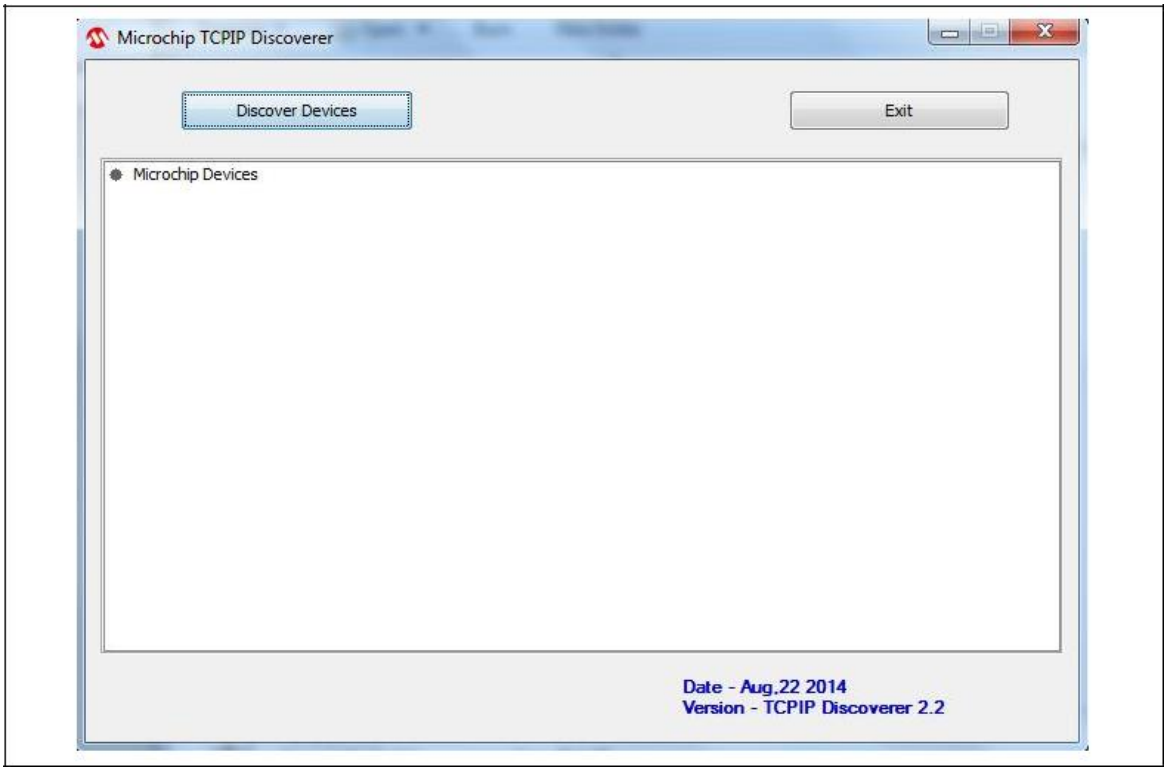
5. Run **wlan open** to open the access to give Wi-Fi commands, and then run **wlan macinfo** to see the Wi-Fi module status.

```
----WIFI MAC configuration----
BootMode: STA
HTTP: Disable
OTA: Disable
SSID: PINGD
```

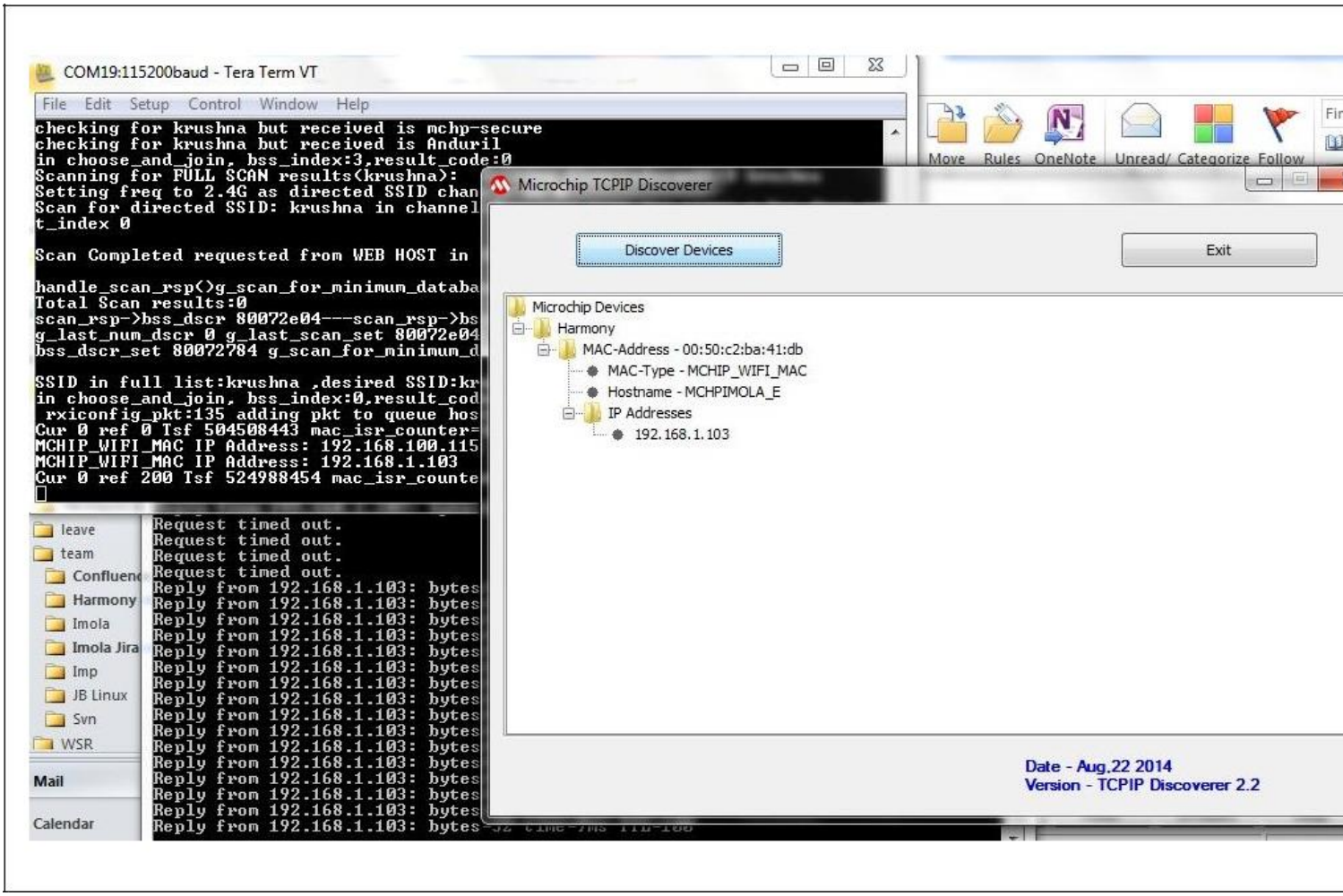
```
Security: NONE
The above Configuration is Saved in Flash
************************************************
----MAC Connected to AP with following Details----
SSID from MAC:PINGD
Security: NONE
```

6. After connecting the third-party computer/laptop to the same Home AP, run **ping** to verify the simple data connection process between the DUT and the computer/laptop.

7. To discover the DUT from the third-party computer/laptop, run the Microchip TCPIP Discoverer tool before the DUT connects to the Home AP. This tool is located in the MPLAB Harmony code base path on the computer/laptop: `<install-dir>/utilities/tcpip_discoverer/tcpip_discoverer.jar`.



8. After the DUT connects to the Home AP, Click **Discover Devices** on the Microchip TCPIP Discoverer tool to discover the DUT. The third-party computer/laptop shows above result for discovered DUT.



wifi_staap_demo

Before using this demonstration, please see the important notes in the *TCP/IP Demonstrations* > *Introduction*.

The WiFi AP mode (Out Of Box) and STA mode support demonstration helps to perform the out-of-box configuration of the Home AP using the DUT as the out-of-box access point (OOBAP). On reboot, the DUT switches to station mode and automatically connects to the configured Home AP.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the PIC32WK Wi-Fi OOBAP and STA mode demonstration.

Description

To build this project, you must open the `wifi_staap_demo.X` project in MPLAB X IDE, and then select the desired configuration. The following tables list and describe the project and supported configurations. The parent folder for these files is:`<install-dir>/apps/tcpip/wifi_staap_demo`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`wifi_staap_demo.X`	`<install-dir>/apps/tcpip/wifi_staap_demo/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are within `./firmware/src/system_config.`

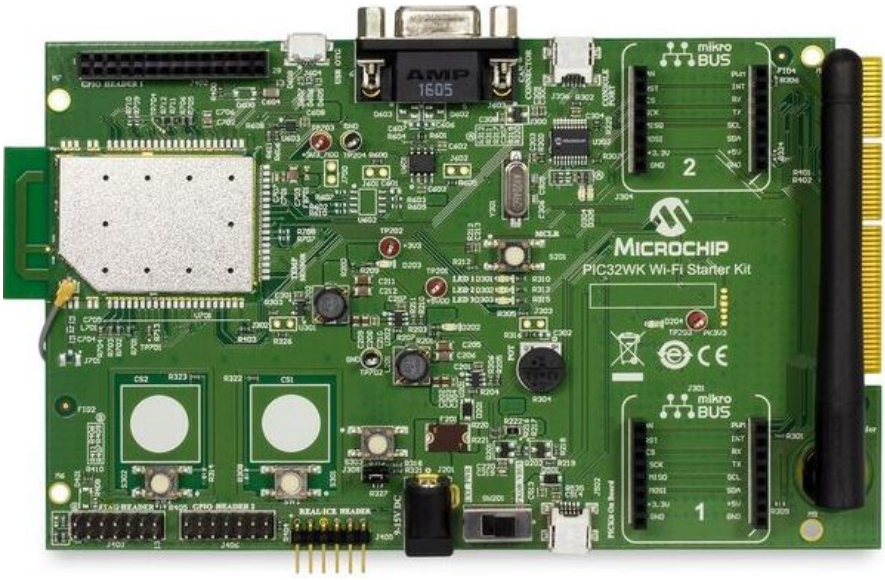
Project Configuration Name	BSP(s) Used	Description
pic32wk_staap	pic32wk_gpb_gpd_sk+module	Demonstrates OOBAP and STA functionality with WM32 Wi-Fi Starter Kit.

Configuring the Hardware

This section describes how to configure the supported hardware.

Description

The following is the PIC32WK Wi-Fi Starter Kit mounted with Wi-Fi module (WM32S2057GXEAS-I/RM)



The Starter Kit has a UART to USB converter (MCP2200) that enables communication between a host PC and the Wi-Fi module. This is done by connecting a mini-B USB cable from the host PC to the USB port (J306) in the Starter Kit.

The console output uses the UART to USB converter on the board at 230400 baud, 8-bit data, No parity, 1 Stop bit, with No Flow Control.

Running the Demonstration

This section provides instructions about how to build and run the PIC32WK Wi-Fi OOBAP and STA mode demonstration with the WM32 module.

Description

The demonstration does the following:

The PIC32WK Wi-Fi DUT starts as OOBAP (out-of-box Access Point) for configuring the Home AP credentials (such as SSID and password), and then switches to STA mode for automatically connecting to the configured Home Access Point (AP) without any user trigger.

AP (OOBAP) mode

- Starts the device in OOBAP mode, with the configuration stored in flash memory, or with the default configuration if the flash configuration is not available.
- The PIC32WK Wi-Fi OOBAP accepts connections from another third-party STA and accepts only one connection.
- The OOBAP can be started only in Open mode.
- The default configuration of the AP is stored in flash memory, and this configuration can also be updated with console commands.
- The connection status can be verified by the connected STA by running **ping**.
- In connected STA, the user can access the OOBAP IP address in any browser that will serve the Web pages.
- From the Web pages, the user can configure the Home AP credentials.
- The configuration details are stored in in-package flash (IPF).

STA mode

- Scans the area and triggers the connection with the Home AP stored in flash memory.
- The default configuration of the Home AP is stored in flash memory, and this configuration can be updated with console commands or by using the Web page.
- After connecting with the Home AP, connection status can be verified by running **ping**.
- After connecting the DUT (PIC32WK) and the third-party personal computer/laptop to the Home AP, the demonstration can be verified.
- The third-party personal computer/laptop can access the DUT IP-address in any browser that will serve the Web pages. That Web page will

display the Welcome screen along with a random seed number on the page.

- From the third-party personal computer/laptop, you can scan for the wireless networks and select the desired AP, or directly give the desired AP configuration and request the Wi-Fi module to connect to that AP. The DUT will connect to that AP and store the configuration information in flash memory.

Use the following procedure to run the demonstration:

1. Load the demonstration project into MPLAB X IDE.
2. Connect the mini-B debugger port on-the starter kit to a USB port on the development computer by using the USB cable provided in the kit.
3. Build, download, and run the demonstration project on the board.

- When the demonstration runs, the DUT boots-up in OOBAP mode for configuring the Home AP credential.

4. Run **netinfo** to display the network details as shown below

```
---------- Interface <wlan0/PIC32WK> ----------
Host Name: MCHPBOARD_WK - NBNS disabled
IPv4 Address: 192.168.3.1
Mask: 255.255.255.0
Gateway: 192.168.3.1
DNS: 192.168.3.1
MAC Address: 54:8c:a0:00:f3:4e
dhcps is ON
dhcp is disabled
Link is UP
```

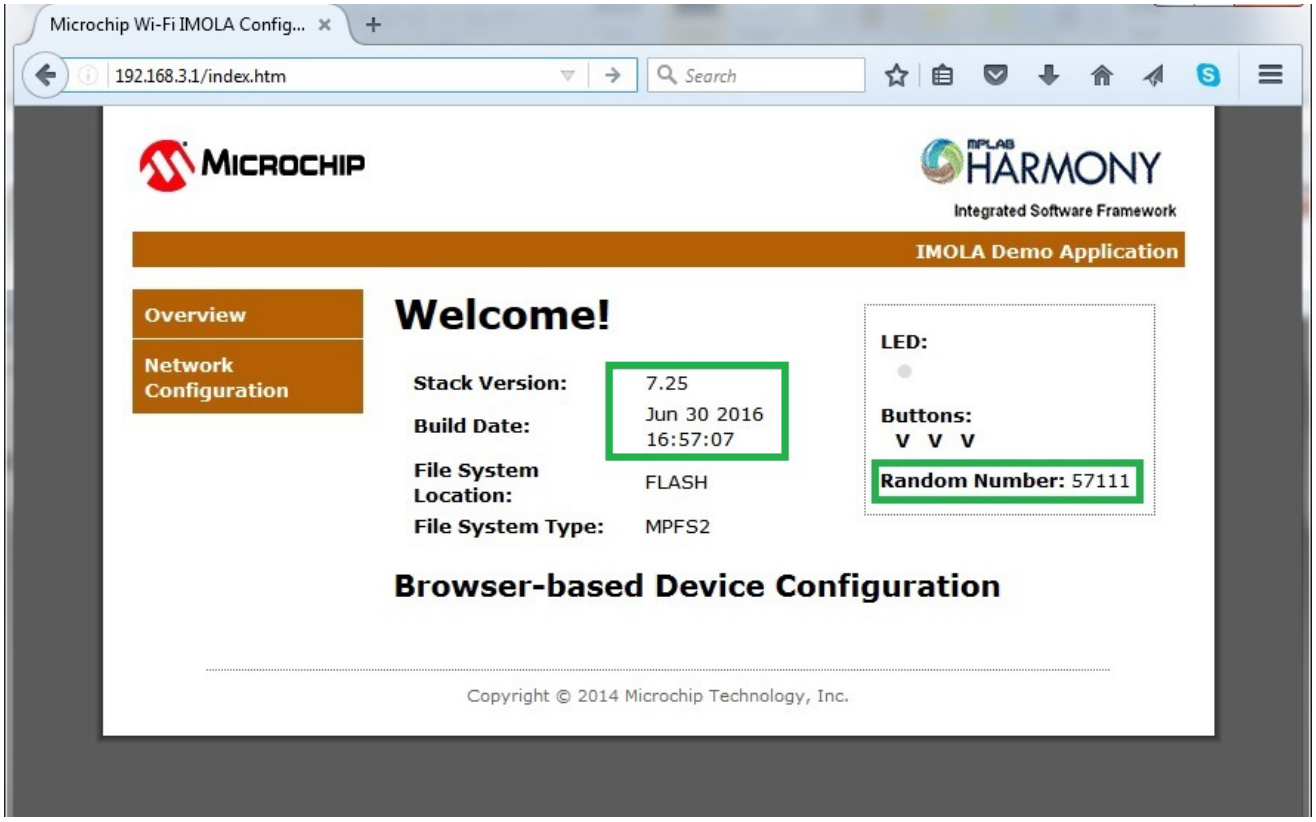
5. Run **wlan open** to open the access to give Wi-Fi commands, and then run **wlan macinfo** to see the Wi-Fi module status.

```
----WIFI MAC configuration----
BootMode: AP
HTTP: Enable
OTA: Enable
SSID: AP_PINGDEMO
Security: NONE
The above Configuration is Saved in Flash
************************************************
----MAC Started as AP with following Details----
SSID from MAC: AP_PINGDEMO
Security: NONE
```

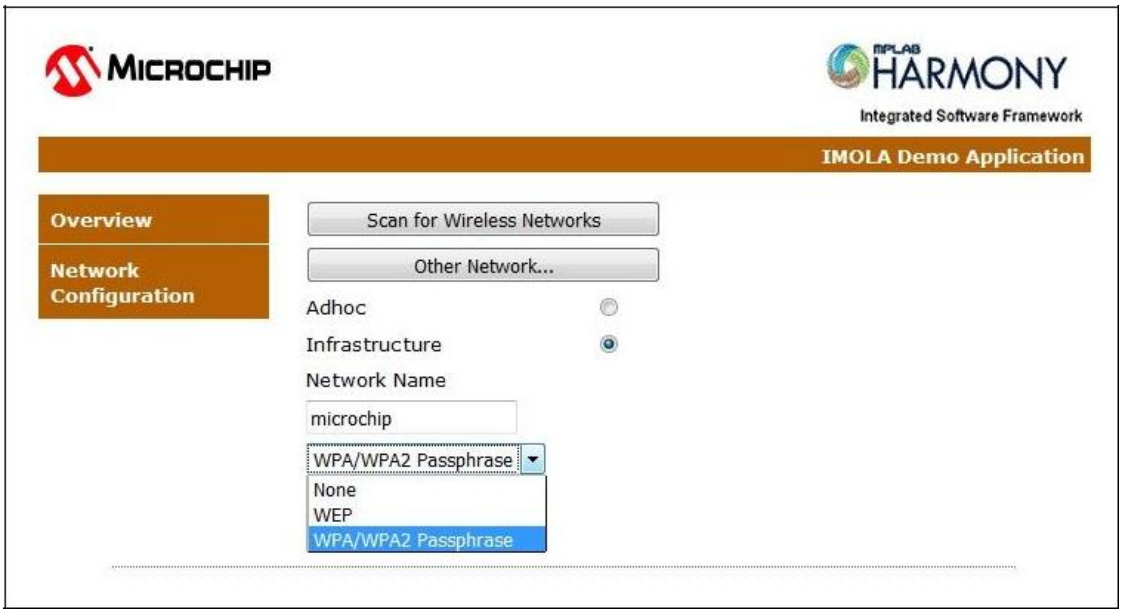
- OOBAP default SSID is “microchip” and IP address is “192.168.3.1”.

6. Connect the third-party personal computer/laptop (STA) to the PIC32WK OOBAP.
7. In the third-party personal computer/laptop, enter the IP-address of the OOBAP in the browser to get served with Web pages.

- The following default web-page displays firmware build details and a random number.

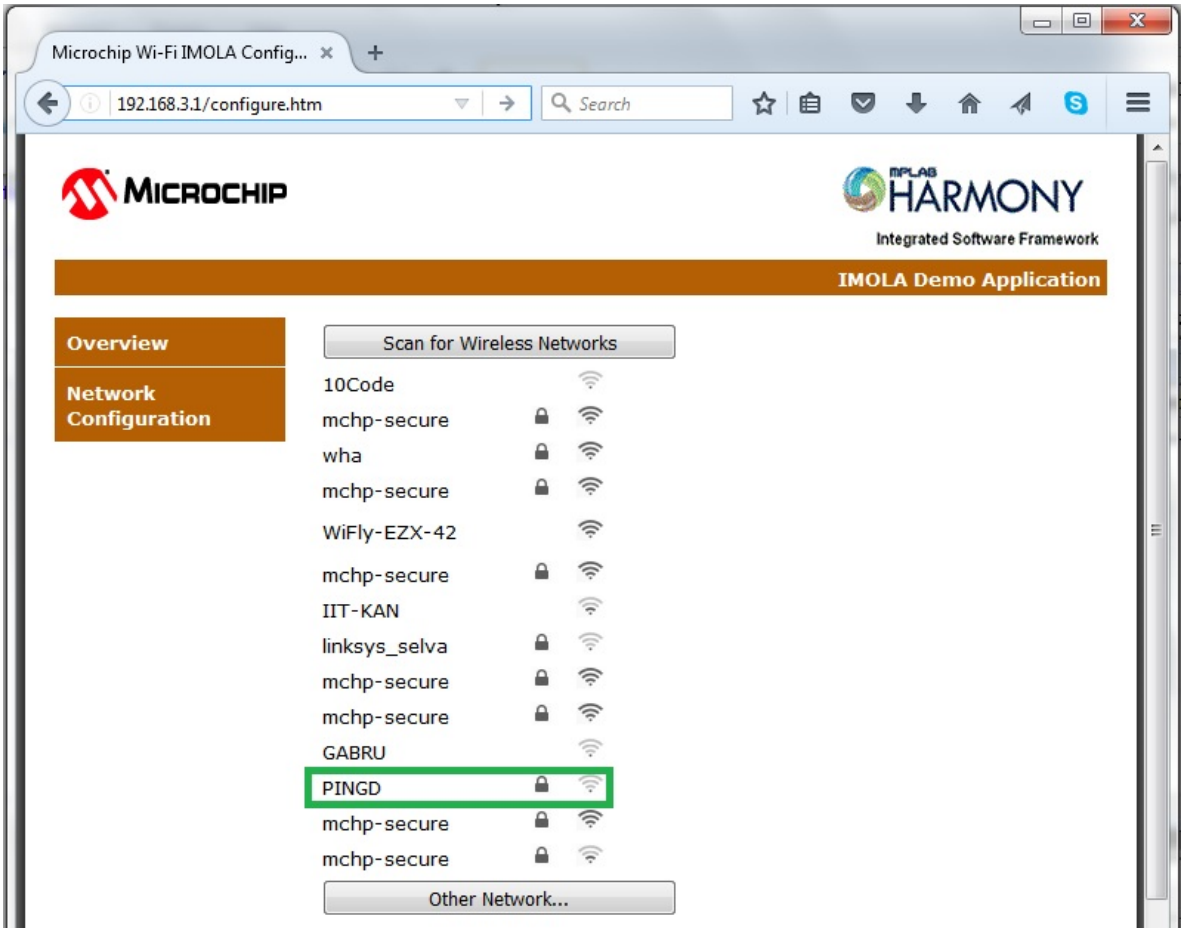


- The Network Configuration page, shown in the following, gives the option to configure the OOBAP manually.



8. Enter the Home AP Wi-Fi configuration details (SSID and Security) and click OK to save the details in in-package flash(IPF). The DUT OOBAP will automatically trigger a soft reboot.

9. If the user wants to scan to get the Home AP availability in the vicinity of the DUT, trigger the scan from **Scan for Wireless Networks**. After a successful scan, the Web page updates the scan results. In the scan results list, click the Home AP. This will send the configuration details (SSID and Security) and save the details in in-package flash (IPF). The DUT OOBAP will automatically trigger a soft reboot.



- After the reboot, the DUT starts in STA mode and starts the connection procedure to the above configured Home AP. The DUT scans for the configured Home AP. If the Home AP is available in the scan results, the DUT will then connect to it. Otherwise the device will be in scanning phase only.
- After connecting with the Home AP, the DUT will get the IP address from the AP (if DHCP is enabled) or the default IP address, 192.168.1.150.

10. Use console commands to get the connection status details. Run **netinfo** to show these connection details.

```
---------- Interface <wlan0/PIC32WK> ----------
Host Name: MCHPBOARD_WK - NBNS disabled
IPv4 Address: 192.168.1.101
Mask: 255.255.255.0
Gateway: 192.168.1.1
DNS: 192.168.1.1
MAC Address: 54:8c:a0:00:f3:82
dhcp is ON
Link is UP
```

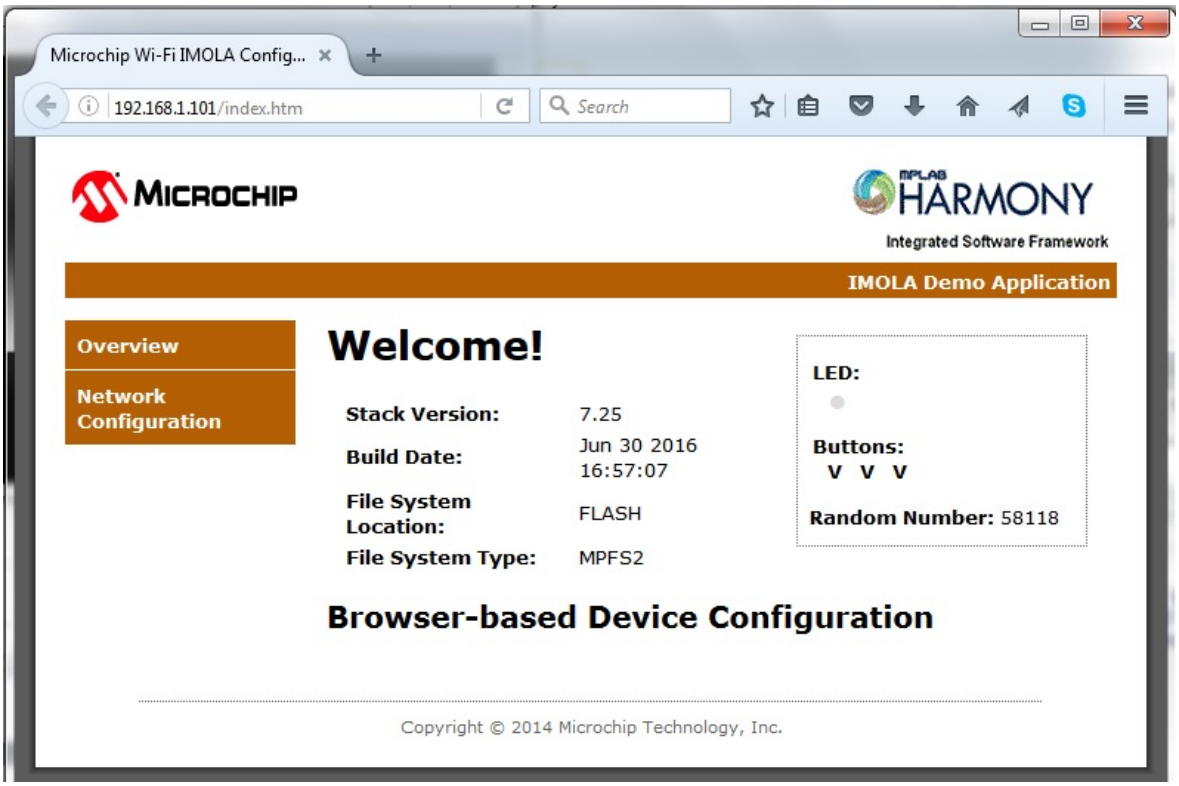
11. Run **wlan open** to open the access to give Wi-Fi commands, and then run **wlan macinfo** to see the Wi-Fi module status.

```
----WIFI MAC configuration----
BootMode: STA
HTTP: Enable
OTA: Enable
SSID: PINGD
Security: NONE
The above Configuration is Saved in Flash
************************************************
----MAC Connected to AP with following Details----
SSID from MAC:PINGD
Security: NONE
```

12. After connecting the third-party personal computer/laptop to the same Home AP, run **ping** to verify the simple data connection process between the DUT and the computer/laptop.

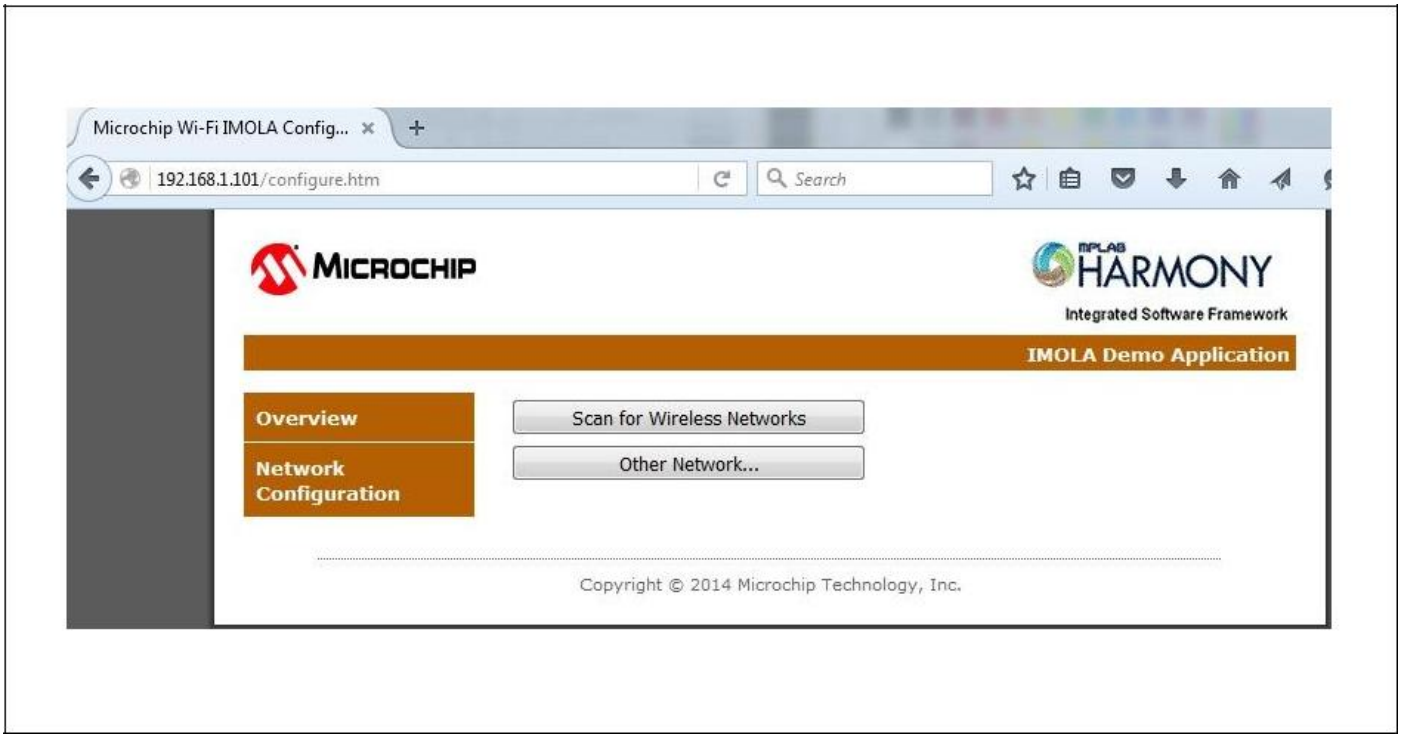
13. Enter the IP-address of the DUT in the browser to get served with Web pages.

- The following default web-page displays firmware build details and a random number.



- The Network Configuration page gives the option to trigger the scan in the Wi-Fi module. After a successful scan, the page updates with scan results.

14. On the Network Configuration page, click **Scan for Wireless Networks** to start the scan. After a successful scan, the Web page updates the scan results. You can choose the desired AP from the scan list to connect the DUT, or click **Other Network** to enter the AP configuration manually.



- The DUT will connect with the given configuration and save the configuration information in flash memory.

Note: When rerunning the demonstration, the Wi-Fi module connects with the last saved configuration.

wifi_sta_http_demo

Before using this demonstration, please see the important notes in the *TCP/IP Demonstrations* > *Introduction*.

The WiFi STA mode with HTTP support demonstration shows how to connect a PIC32WK Wi-Fi module with Home AP and also to open the Web page using HTTP for accessing Wi-Fi module for Scan and updating configuration.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the PIC32WK Wi-Fi STA mode with HTTP configuration demonstration.

Description

To build this project, you must open the wifi_sta_http_demo.X project in MPLAB X IDE, and then select the desired configuration. The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/tcpip/wifi_sta_http_demo.`

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`wifi_sta_http_demo.X`	`<install-dir>/apps/tcpip/wifi_sta_http_demo/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are within `./firmware/src/system_config.`

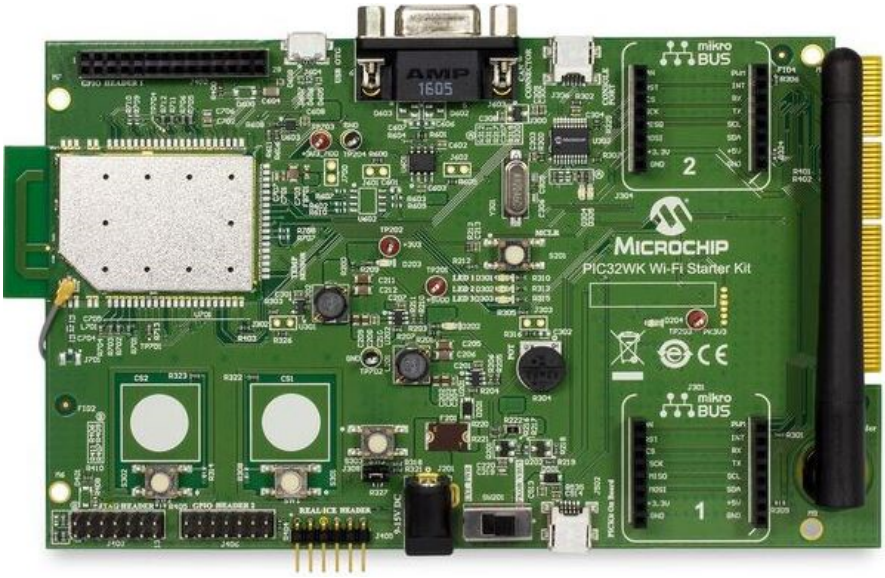
Project Configuration Name	BSP(s) Used	Description
pic32wk_sta_http	pic32wk_gpb_gpd_sk+module	Demonstrates STA functionality and Web page support for Scan and Connect with Home AP with WM32 Wi-Fi Starter Kit.

Configuring the Hardware

This section describes how to configure the supported hardware.

Description

The following is the PIC32WK Wi-Fi Starter Kit mounted with Wi-Fi module (WM32S2057GXEAS-I/RM)



The Starter Kit has a UART to USB converter (MCP2200) that enables communication between a host PC and the Wi-Fi module. This is done by connecting a mini-B USB cable from the host PC to the USB port (J306) in the Starter Kit.

The console output uses the UART to USB converter on the board at 230400 baud, 8-bit data, No parity, 1 Stop bit, with No Flow Control.

Running the Demonstration

This section provides instructions about how to build and run the PIC32WK Wi-Fi STA mode with HTTP demonstration with the WM32 module.

Description

The demonstration does the following:

- Scans the area and triggers the connection with the Home Access Point (AP).
- The default configuration of the Home AP is stored in flash memory. This configuration can be updated through console commands or the Web page.
- After connecting with the Home AP, the connection status can be verified by running **ping**.
- After connecting the DUT and the third-party personal computer/laptop to the Home AP, the demonstration can be verified.
- In the third-party personal computer/laptop, you can access the DUT IP-address in any browser that serves the Web pages.
- That web page will display the Welcome screen along with a random seed number on the page.
- From the third-party personal computer/laptop, you can scan for the wireless networks and select the desired AP, or directly give the desired AP configuration and command the Wi-Fi module to connect to that AP. The WM32 module will connect to that AP and store the configuration information in flash memory.

Use the following procedure to run the demonstration:

1. Load the demonstration project into MPLAB X IDE.

2. Connect the mini-B debugger port on the starter kit to a USB port on the development computer by using the USB cable provided in the kit.

3. Build, download, and run the demonstration project on the board.

- When demonstration runs, it scans for APs and connects with the saved configuration of the Home AP. If the configured AP is not available in the scan list, the device will be in scanning phase only.
- After connecting with the Home AP, the DUT (PIC32WK Wi-Fi) gets the IP-address from the AP (if DHCP is enabled) or the default IP-address, 192.168.1.150

4. Use console commands to get the connection status details. Run **netinfo** to display these connection details.

```
---------- Interface <wlan0/PIC32WK> ----------
Host Name: MCHPBOARD_WK - NBNS disabled
IPv4 Address: 192.168.1.101
Mask: 255.255.255.0
Gateway: 192.168.1.1
```

```
DNS: 192.168.1.1
MAC Address: 54:8c:a0:00:f3:82
dhcp is ON
Link is UP
```

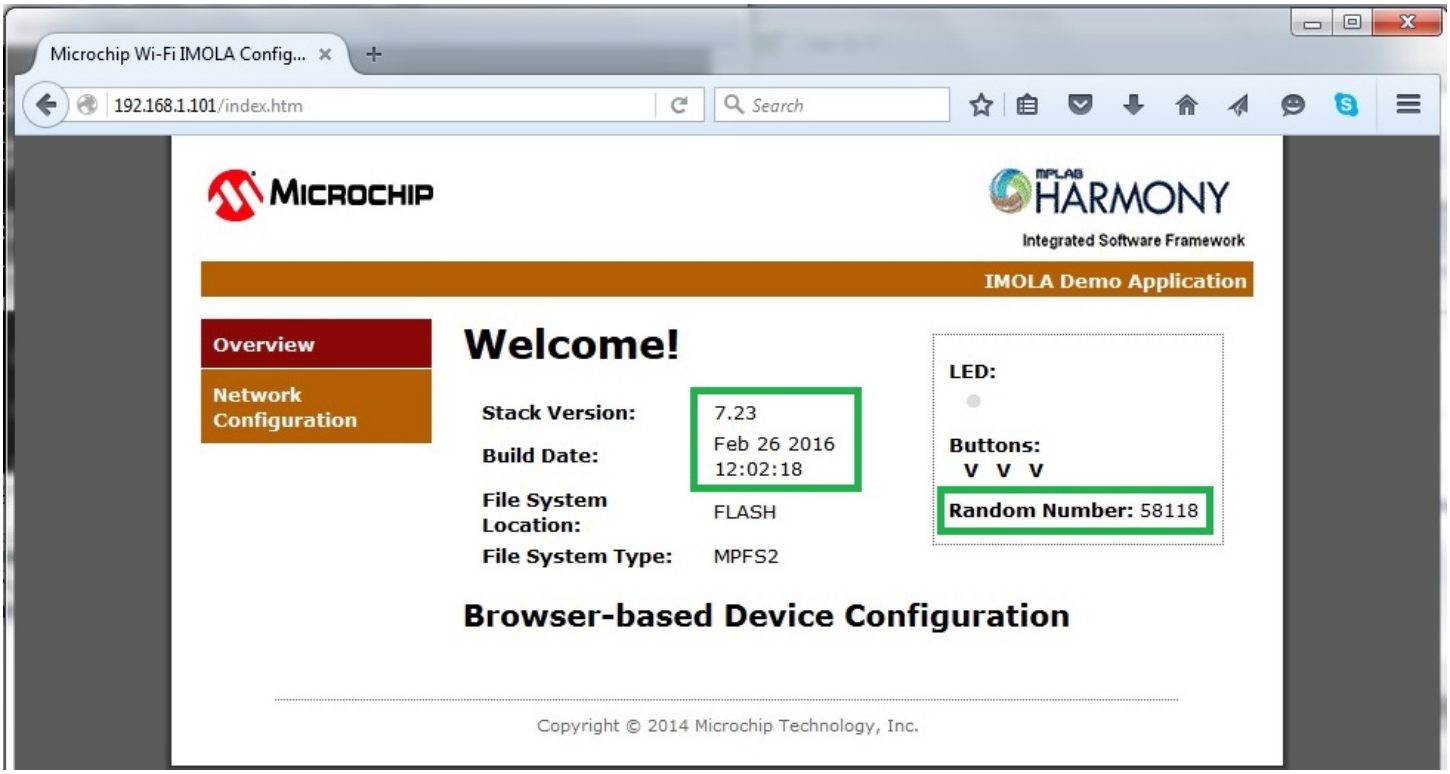
5. Run **wlan open** to open the access to give Wi-Fi commands, and then run **wlan macinfo** to see the Wi-Fi module status.

```
----WIFI MAC configuration----
BootMode: STA
HTTP: Enable
OTA: Enable
SSID: PINGD
PSK: 12345678
Mode: 0x79
The above Configuration is Saved in Flash
************************************************
----MAC Connected to AP with following Details----
SSID from MAC:PINGD
PSK: 12345678
Mode: 0x79
```

6. After connecting the third-party personal computer/laptop to the same Home AP, run **ping** to verify the simple data connection process between the DUT and the computer/laptop.

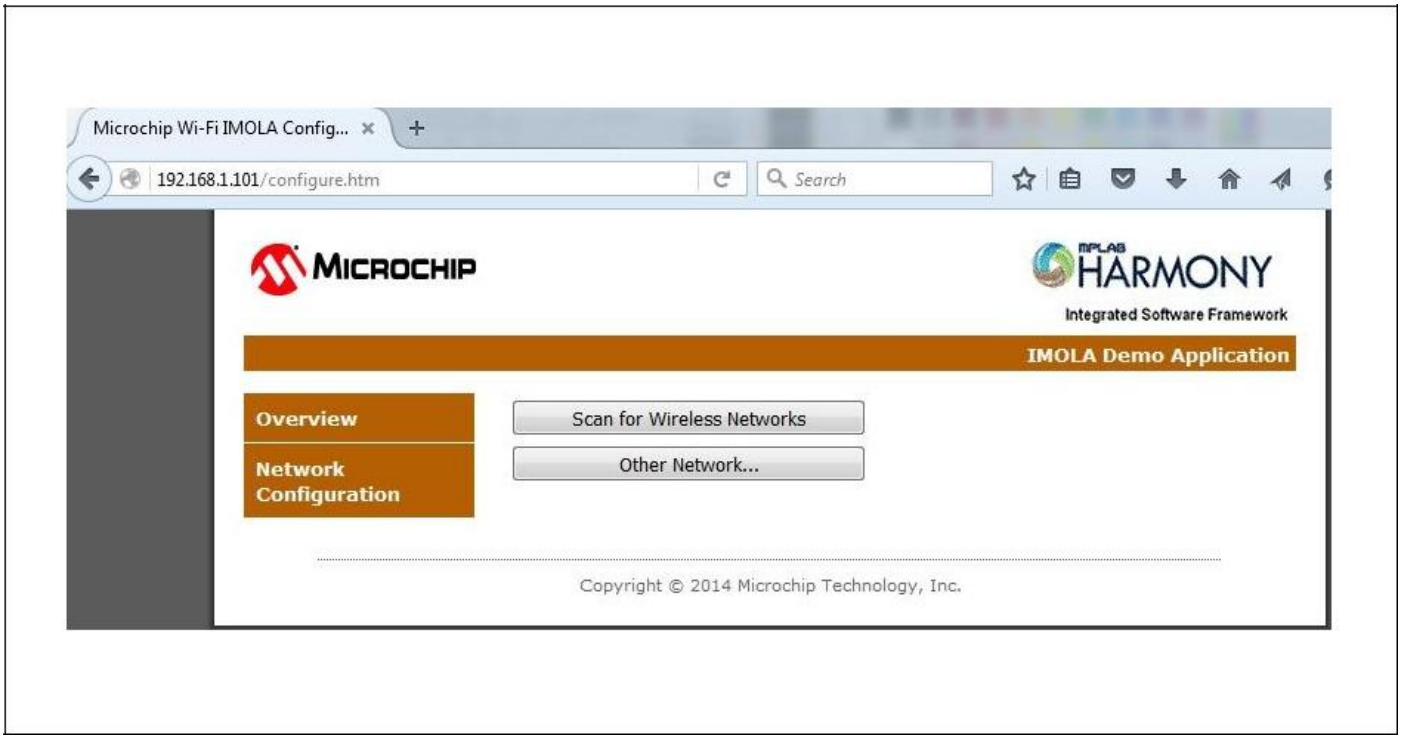
7. Enter the IP-address of the DUT in the browser to get served with web pages.

- The following default web-page gives firmware build details and a random number.



- The “Network Configuration” page gives the option to trigger the scan in the Wi-Fi module. After a successful scan, the page updates with scan results.

8. On the Network Configuration page, click **Scan for Wireless Networks** to start the scan. After a successful scan, the Web page updates the scan results. You can choose the desired AP from the scan list to connect the DUT, or click **Other Network** to enter the AP configuration manually.



- The DUT will connect with the given configuration and save the configuration information in flash memory.

Note: When rerunning the demonstration, the Wi-Fi module connects with the last saved configuration.

wifi_sta_ota_demo

Before using this demonstration, please see the important notes in the *TCP/IP Demonstrations* > *Introduction*.

The WiFi STA mode with OTA support demonstration shows how to connect a PIC32WK Wi-Fi module with Home AP and also open the Web page by using HTTP for OTA (Over the Air) update.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the PIC32WK Wi-Fi STA mode with OTA support demonstration.

Description

To build this project, you must open the wifi_sta_ota_demo.X project in MPLAB X IDE, and then select the desired configuration. The following tables list and describe the project and supported configurations. The parent folder for these files is

`<install-dir>/apps/tcpip/wifi_sta_ota_demo`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`wifi_sta_ota_demo.X`	`<install-dir>/apps/tcpip/wifi_sta_ota_demo/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are within

`./firmware/src/system_config.`

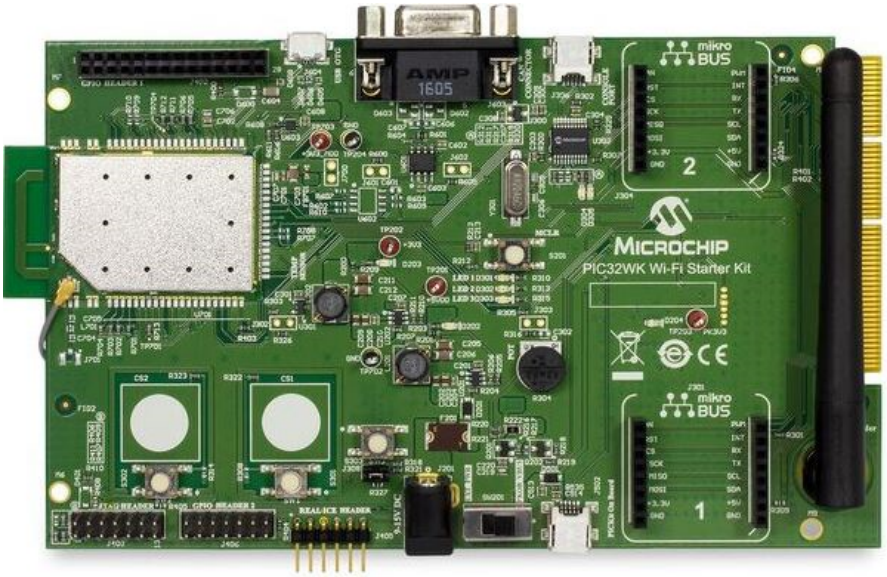
Project Configuration Name	BSP(s) Used	Description
pic32wk_sta_http_ota	pic32wk_gpb_gpd_sk+module	Demonstrates STA functionality and Web page support for demonstration of how to update the device over OTA using HTTP.

Configuring the Hardware

This section describes how to configure the supported hardware.

Description

The following is the PIC32WK Wi-Fi Starter Kit mounted with Wi-Fi module (WM32S2057GXEAS-I/RM)



The Starter Kit has a UART to USB converter (MCP2200) that enables communication between a host PC and the Wi-Fi module. This is done by connecting a mini-B USB cable from the host PC to the USB port (J306) in the Starter Kit.

The console output uses the UART to USB converter on the board at 230400 baud, 8-bit data, No parity, 1 Stop bit, with No Flow Control.

Running the Demonstration

This section provides instructions about how to build and run the PIC32WK Wi-Fi STA mode with OTA demonstration with the WM32 module.

Description

The demonstration does the following:

- Scans the area and triggers the connection with the Home Access Point (AP) stored in flash memory.
- The default configuration of the Home AP is stored in flash memory, and this configuration can be updated through console commands or the Web page.

- After connecting with the Home AP, connection status can be verified by running **ping**.
- After connecting the DUT and the third-party personal computer/laptop to the Home AP, the OTA demonstration can be verified.
- In the third-party personal computer/laptop, the DUT IP-address for starting OTA procedure can be accessed in any browser that serves the Web pages.
- That web page gives the option to upload the file.
- OTA updates through the HTTP server.

Use the following procedure to run the demonstration:

1. Load the demonstration project into MPLAB X IDE.

2. Connect the mini-B debugger port on-the starter kit to a USB port on the development computer by using the USB cable provided in the kit.

3. Build, download, and run the demonstration project on the board.

- When demonstration runs, it scans for APs and connects with the saved configuration of the Home AP. If the configured AP is not available in the scan list, the device will be in scanning phase only.
- After connecting with the Home AP, the DUT (PIC32WK Wi-Fi) will get the IP-address from the AP (if DHCP is enabled) or the default IP-address, 192.168.1.150

4. Use console commands to get the connection status details. Run **netinfo** to display these connection details.

```
---------- Interface <wlan0/PIC32WK> ----------
Host Name: MCHPBOARD_WK - NBNS disabled
IPv4 Address: 192.168.1.101
Mask: 255.255.255.0
Gateway: 192.168.1.1
DNS: 192.168.1.1
MAC Address: 54:8c:a0:00:f3:82
dhcp is ON
Link is UP
```

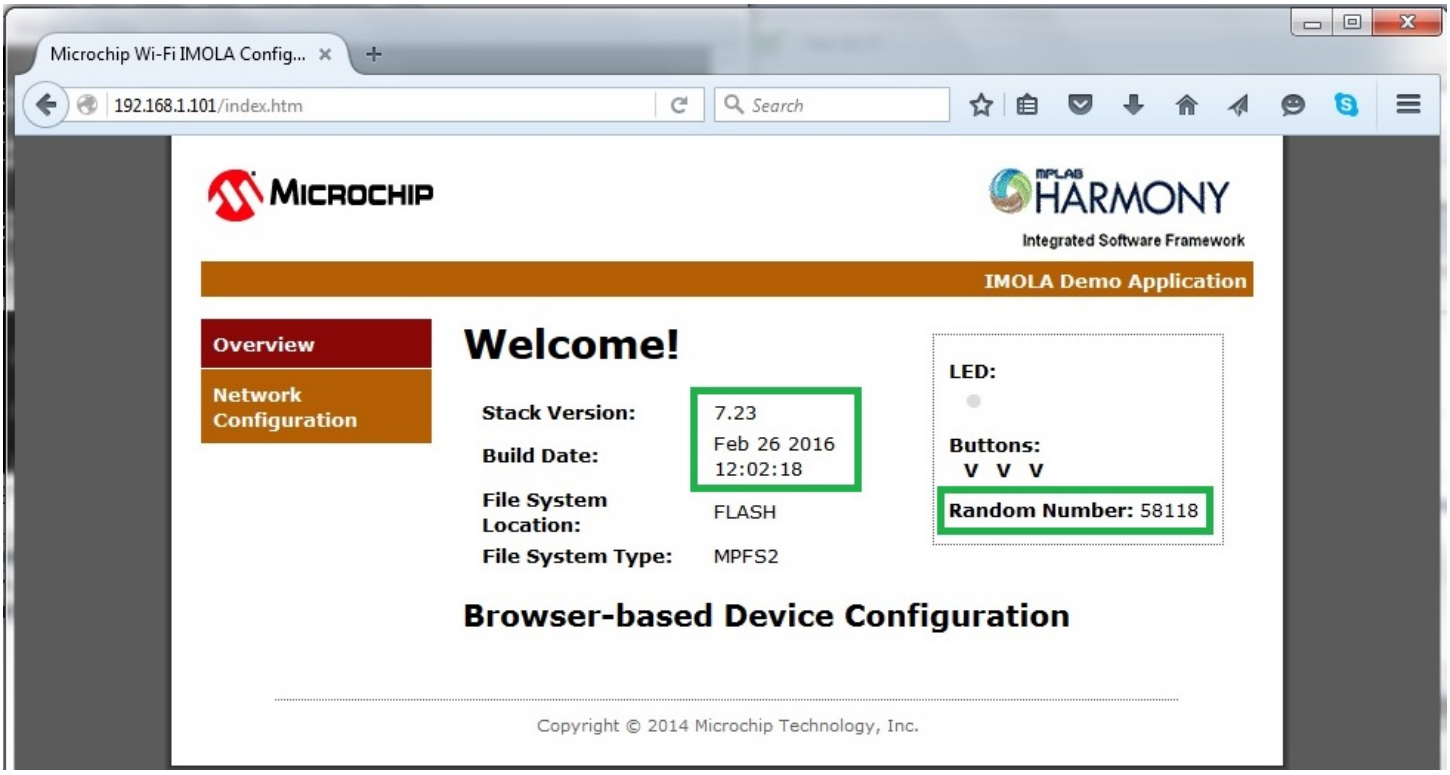
5. Run **wlan open** to open the access to give Wi-Fi commands, and then run **wlan macinfo** to see the Wi-Fi module status.

```
----WIFI MAC configuration----
BootMode: STA
HTTP: Enable
OTA: Enable
SSID: PINGD
PSK: 12345678
Mode: 0x79
The above Configuration is Saved in Flash
************************************************
----MAC Connected to AP with following Details----
SSID from MAC:PINGD
PSK: 12345678
Mode: 0x79
```

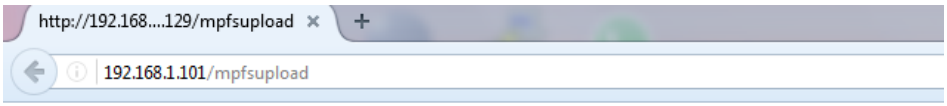
6. After connecting the third-party personal computer/laptop to the same Home AP, run **ping** to verify the simple data connection process between the DUT and the computer/laptop.

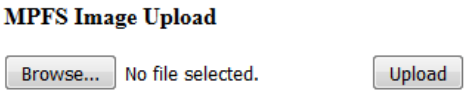
7. Enter the IP-address of the DUT in the browser to get served with web pages.

- The default web-page as shown below gives firmware build details and a random number.



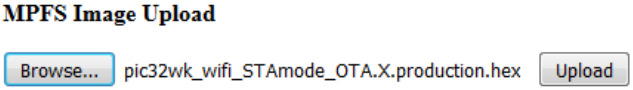
8. In the browser, type the IP address appended with "/mpfsupload" to get the upload page. Refer to the following.





9. In local storage, browse to and select the hex file to be uploaded, and then click **Upload**.

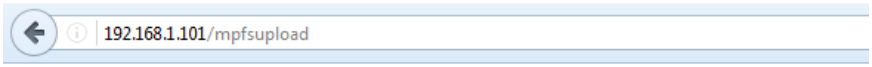




- When the upload starts, you can observe the progress in the command console. After upload is complete, the command console outputs “Done” followed by “OTA Completed”.



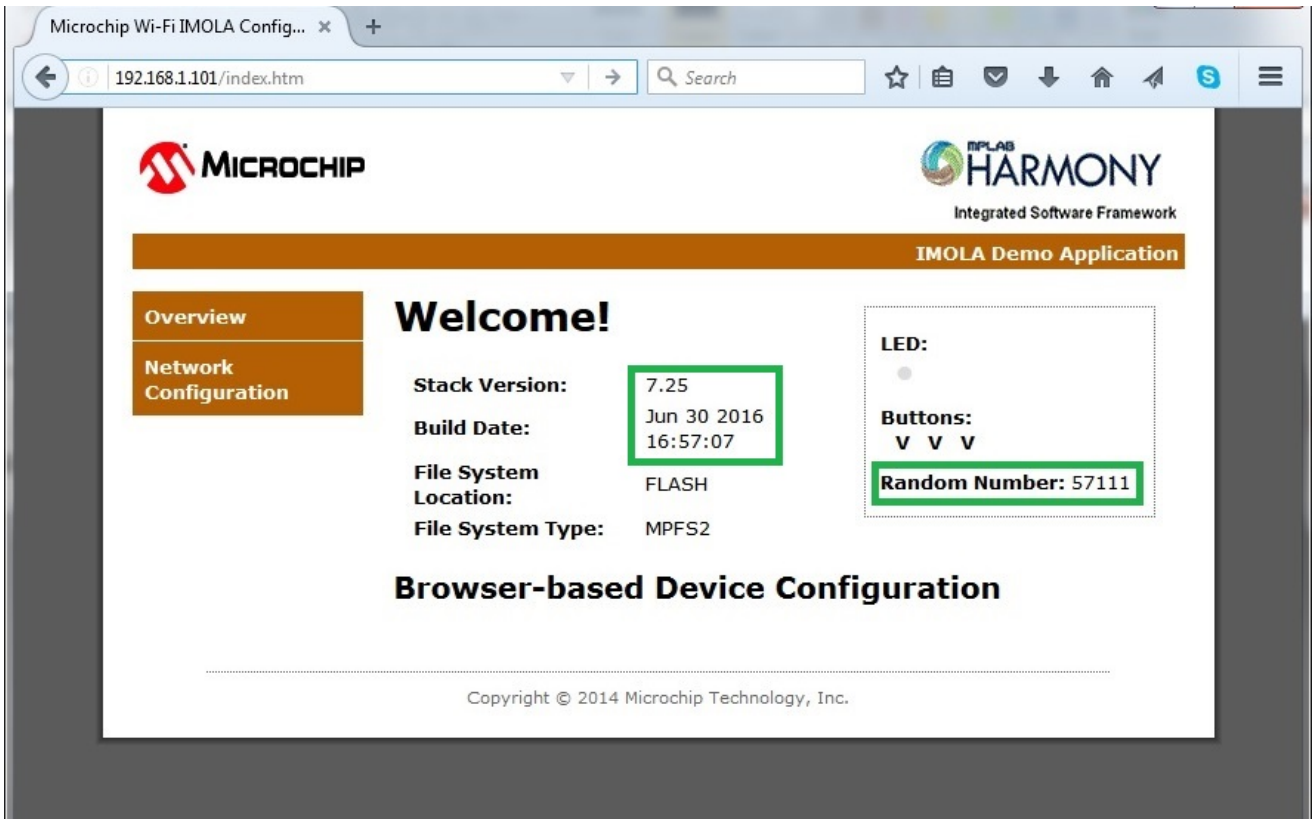
- The browser displays a message that the OTA completion is successful.



MPFS Update Successful

Site main page

10. Click the Site main page link to be redirected to the home page. After successful upload of the firmware image, you can see the new build details on the home page.



wifi_sta_wolfssl_demo

Before using this demonstration, please see the important notes in the *TCP/IP Demonstrations* > *Introduction*.

This Wi-Fi STA and wolfSSL TCP/IP Client demonstration.showcases how to connect a PIC32WK Wi-Fi module with Home AP and access a secure Web site using wolfSSL.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the PIC32WK Wi-Fi STA mode with SSL demonstration.

Description

To build this project, you must open the wifi_sta_wolfssl_demo.Xproject in MPLAB X IDE, and then select the desired configuration. The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/tcpip/wifi_sta_wolfssl_demo.`

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`wifi_sta_wolfssl_demo.X`	`<install-dir>/apps/tcpip/wifi_sta_wolfssl_demo/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are within `./firmware/src/system_config.`

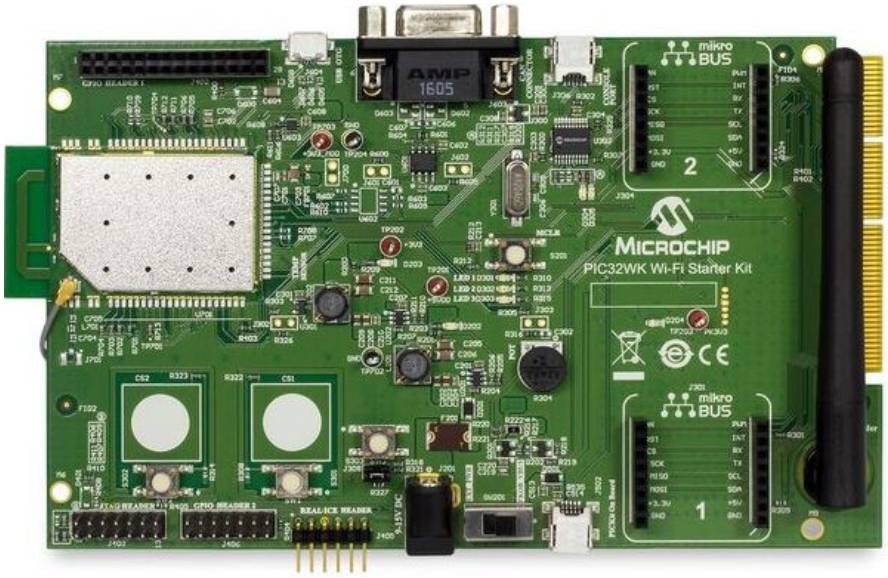
Project Configuration Name	BSP(s) Used	Description
pic32wk_sta_wolfssl	pic32wk_gpb_gpd_sk+module	Demonstrates STA with SSL functionality with WM32 Wi-Fi Starter Kit.

Configuring the Hardware

This section describes how to configure the supported hardware.

Description

The following is the PIC32WK Wi-Fi Starter Kit mounted with Wi-Fi module (WM32S2057GXEAS-I/RM)



The Starter Kit has a UART to USB converter (MCP2200) that enables communication between a host PC and the Wi-Fi module. This is done by connecting a mini-B USB cable from the host PC to the USB port (J306) in the Starter Kit.

The console output uses the UART to USB converter on the board at 230400 baud, 8-bit data, No parity, 1 Stop bit, with No Flow Control.

Running the Demonstration

This section provides instructions about how to build and run the PIC32WK Wi-Fi STA mode SSL demonstration with the WM32 module.

Description

The demonstration does the following:

- Scans the area and triggers the connection with the Home AP stored in flash memory.
- The default configuration of the Home AP is stored in flash memory, and this configuration can be updated with console commands.
- After connecting with the Home AP, the connection status can be verified by running **ping**.
- Opens the secure web page by running the wolfSSL command.

Use the following procedure to run the demonstration:

1. Load the demonstration project into MPLAB X IDE.

2. Connect the mini-B debugger port on- the starter kit to a USB port on the development computer using the USB cable provided in the kit.

3. Build, download, and run the demonstration project on the board.

- When the demonstration runs, it scans for APs and connects with the saved configuration of the Home AP. If the configured AP is not available in the scan list, the device will be in scanning phase only.
- After connecting with the Home AP, the DUT (PIC32WK Wi-Fi) will get the IP address from the AP (if DHCP is enabled) or the default IP-address, 192.168.1.150.

4. Use console commands to get the connection status details. Run **netinfo** to display these connection details.

```
---------- Interface <wlan0/PIC32WK> ----------
Host Name: MCHPBOARD_WK - NBNS disabled
IPv4 Address: 192.168.1.101
Mask: 255.255.255.0
Gateway: 192.168.1.1
DNS: 192.168.1.1
MAC Address: 54:8c:a0:00:f3:82
dhcp is ON
Link is UP
```

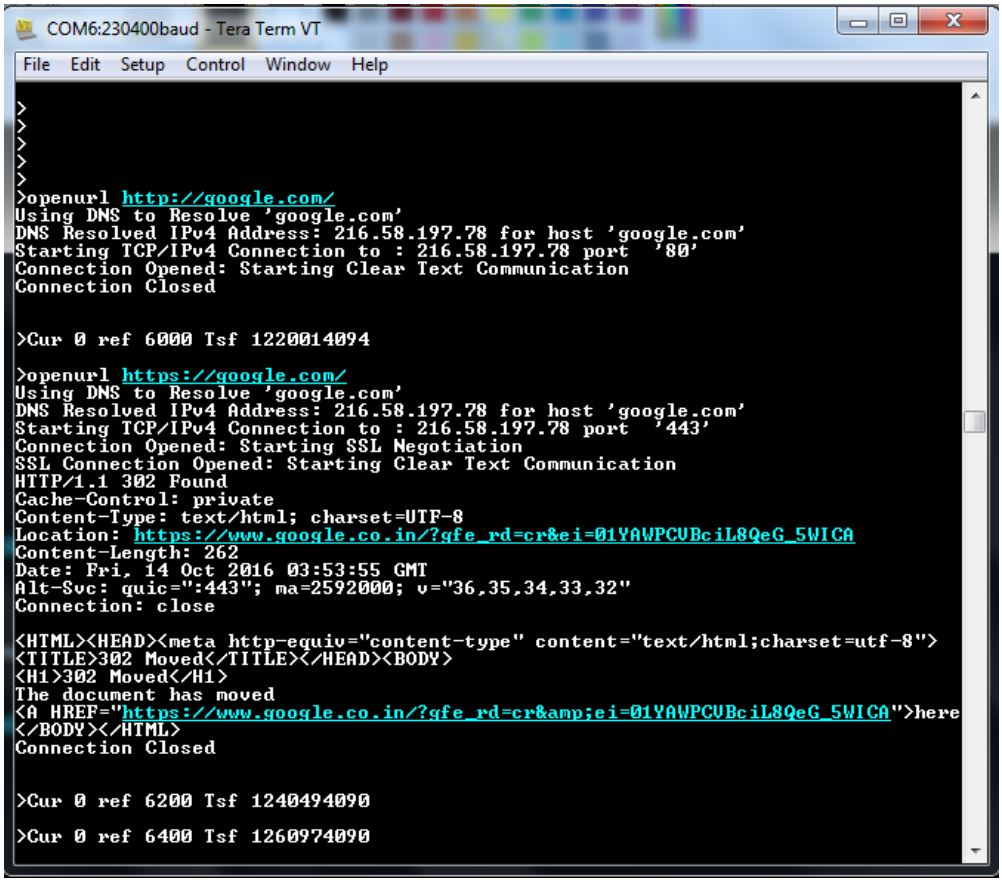
5. Run **wlan open** to open the access to give Wi-Fi commands, and then run **wlan macinfo** to see the Wi-Fi module status.

```
----WIFI MAC configuration----
BootMode: STA
HTTP: Enable
OTA: Enable
SSID: PINGD
PSK: 12345678
Mode: 0x79
The above Configuration is Saved in Flash
************************************************
----MAC Connected to AP with following Details----
SSID from MAC:PINGD
PSK: 12345678
Mode: 0x79
```

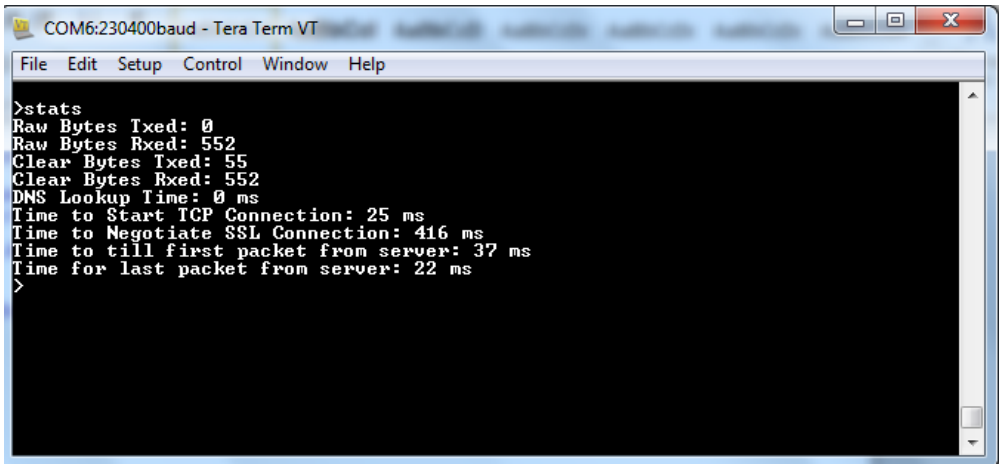
6. After connecting the third-party personal computer/laptop to the same Home AP, run **ping** to verify the simple data connection process between the DUT and the computer/laptop.

7. To access the content of the Web page, run **openurl *<url>***, where <*url*> must be a fully formed URL.

- Example: **openurl http://google.com/** or **openurl https://google.com/**
- The following shows an example of accessing a secure and a non-secure Web page by running the **openurl** command.



8. Run **stats** to retrieve the statistics from running the previous **openurl** command. Examples of statistics are how long each phase of the connection took, and how many bytes were transferred.



9. After running the openurl command, the demonstration makes a DNS query. Open a connection to the requested URL and run an http PUT command.

- The results are sent to the serial port. If an https URL is specified, the connection will first undergo SSL negotiation before running the http PUT command.

wifi_wilc1000

This section describes how to use a simple Wi-Fi application using the WILC1000 firmware.

Description

This application demonstrates a simple Wi-Fi Application using the WILC1000 firmware on the WINC1500 Wi-Fi PICtail/PICtail Plus Daughter Board, with different configurations; STA mode, AP mode, and the wolfSSL Client. The ping from DUT to Home AP works in STA mode, and ping from DUT to a connected Station works in AP mode.

Libraries Used

Other than the common libraries used, following are the libraries have been used for the demonstration:

- MPLAB Harmony TCP/IP Stack
- FreeRTOS
- WILC1000 Wi-Fi Driver Library

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the wifi_wilc1000 Demonstration.

Description

To build this project, you must open the `wifi_wilc1000.X` project in MPLAB X IDE, and then select the desired configuration. The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/tcpip/wifi_wilc1000`.

MPLAB X IDE Project

The following table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`wifi_wilc1000.X`	`<install-dir>/apps/tcpip/wifi_wilc1000/firmware`

MPLAB X IDE Project Configurations

The following table describes the supported configurations of the demonstration, which are within `./firmware/src/system_config`.

Project Configuration Name	BSP(s) Used	Description
pic32mx795_pim_e16_wilc_freertos_sta	pic32mx795_pim + e16	FreeRTOS version of demonstration running STA Mode with on WILC1000 firmware running on the WINC1500 Wi-Fi PICtail/PICtail Plus Daughter Board on the Explorer 16 Development Board with PIC32MX795F512L CAN-USB Plug-in Module (PIM).
pic32mx795_pim_e16_wilc_freertos_ap	pic32mx795_pim + e16	FreeRTOS version of demonstration running AP Mode with on WILC1000 firmware running on WINC1500 Wi-Fi PICtail/PICtail Plus Daughter Board on the Explorer 16 Development Board with PIC32MX795F512L CAN-USB Plug-in Module (PIM).
pic32mz_ef_sk__ioexp__wilc__freertos_wolfssl_client	pic32mz_ef_sk	FreeRTOS version of demonstration running the wolfSSL Client with the WILC1000 firmware running on WINC1500 Wi-Fi PICtail/PICtail Plus Daughter Board on the PIC32MZ EF Starter Kit with a PIC32MZ2048 device.
pic32mx795_pim__e16__wilc__sta	pic32mx795_pim + e16	Non-OS version of demonstration running STA Mode on PIC32 PIM with WILC1000 firmware running on the WINC1500 Wi-Fi PICtail/PICtail Plus Daughter Board on the Explorer 16 Development Board. This demo is similar to the demo pic32mx795_pim__e16__wilc__freertos_sta in terms of running it.
pic32mx795_pim__e16__wilc__freertos_ap_scan	pic32mx795_pim + e16	FreeRTOS version of Demonstration Scanning APs on PIC32 PIM with WILC1000 firmware running on the WINC1500 Wi-Fi PICtail/PICtail Plus Daughter Board on the Explorer 16 Development Board.The demo scans for the APs and prints the list of scanned APs, on the console.
pic32mx795_pim__e16__wilc__freertos_mac_address_chip_info	pic32mx795_pim + e16	FreeRTOS version of demonstration for getting MAC Address and Chip Info on PIC32 PIM with WILC1000 firmware running on the WINC1500 Wi-Fi PICtail/PICtail Plus Daughter Board on the Explorer 16 Development Board. The demo prints the MAC Address and Chip Info, on the console.
pic32mx795_pim__e16__wilc__freertos_p2p_client	pic32mx795_pim + e16	FreeRTOS version of Demonstration running P2P Client on PIC32 PIM with WILC1000 firmware running on the WINC1500 Wi-Fi PICtail/PICtail Plus Daughter Board on the Explorer 16 Development Board.
pic32mx795_pim__e16__wilc__freertos_mode_change	pic32mx795_pim + e16	FreeRTOS version of demonstration Changing Mode from AP to P2P Client on PIC32 PIM with WILC1000 firmware running on the WINC1500 Wi-Fi PICtail/PICtail Plus Daughter Board on the Explorer 16 Development Board.

pic32mx795_pim__e16__wilc__freertos_tcp_client	pic32mx795_pim + e16	FreeRTOS version of demonstration running TCP Client on PIC32 PIM with WILC1000 firmware running on the WINC1500 Wi-Fi PICtail/PICtail Plus Daughter Board on the Explorer 16 Development Board.
pic32mx795_pim__e16__wilc__freertos_tcp_server	pic32mx795_pim + e16	FreeRTOS version of demonstration running TCP Server on PIC32 PIM with WILC1000 firmware running on the WINC1500 Wi-Fi PICtail/PICtail Plus Daughter Board on the Explorer 16 Development Board.
pic32mx795_pim__e16__wilc__freertos_udp_server	pic32mx795_pim + e16	FreeRTOS version of Demonstration running UDP Server on PIC32 PIM with WILC1000 firmware running on the WINC1500 Wi-Fi PICtail/PICtail Plus Daughter Board on the Explorer 16 Development Board.
pic32mz_ef_sk__ioexp__wilc__freertos_wolfmqtt_client	pic32mz_ef_sk	FreeRTOS version of demonstration running WolfMqtt Client on PIC32 EF Starter Kit with WILC1000 firmware running on the WINC1500 Wi-Fi PICtail/PICtail Plus Daughter Board.
pic32mx795_pim__e16__wilc__freertos_wps	pic32mx795_pim + e16	FreeRTOS version of Demonstration running WPS on PIC32 PIM with WILC1000 firmware running on the WINC1500 Wi-Fi PICtail/PICtail Plus Daughter Board on the Explorer 16 Development Board. The Demo runs WPS Push Button Mode by default, and if one needs to run WPS PIN, he needs to make MAIN_WPS_PUSH_BUTTON_FEATURE as ‘false’ in wifi_wilc1000\firmware\src\wps.c file
pic32mx795_pim__e16__wilc__freertos_udp_client	pic32mx795_pim + e16	FreeRTOS version of Demonstration UDP Client on PIC32 PIM with WILC1000 firmware running on the WINC1500 Wi-Fi PICtail/PICtail Plus Daughter Board on the Explorer 16 Development Board.

Configuring the Hardware

This section describes how to configure the hardware using the WINC1500 PICtail Daughter Board and the PICtail Plus slot.

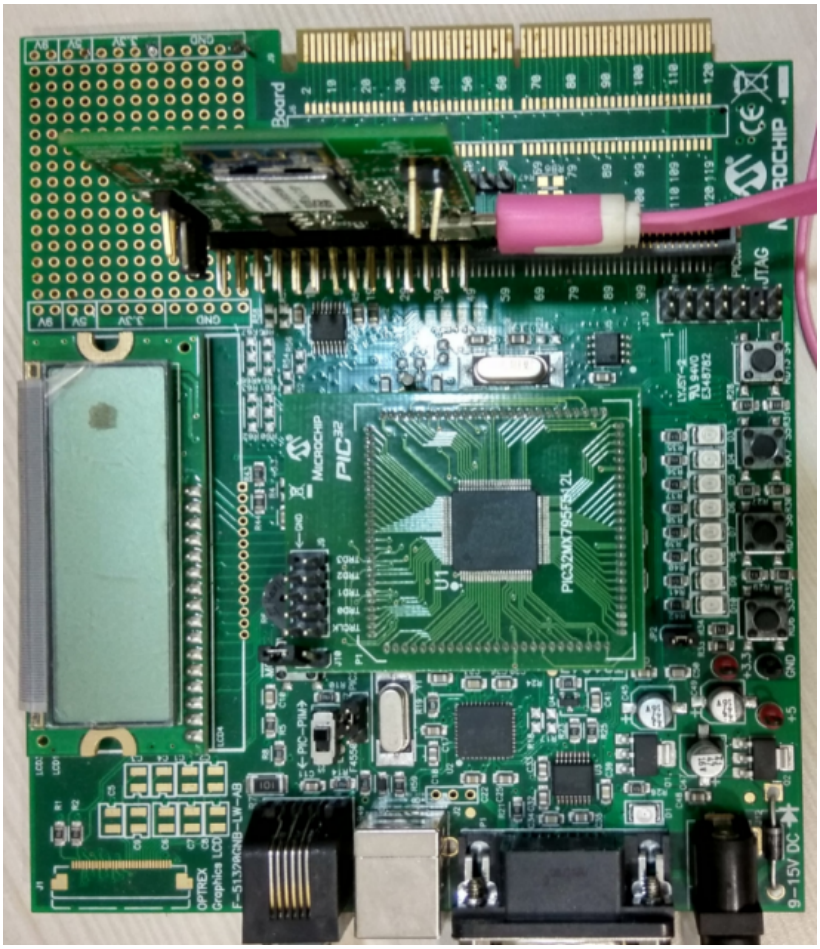
Description

The hardware is configured with the following steps:

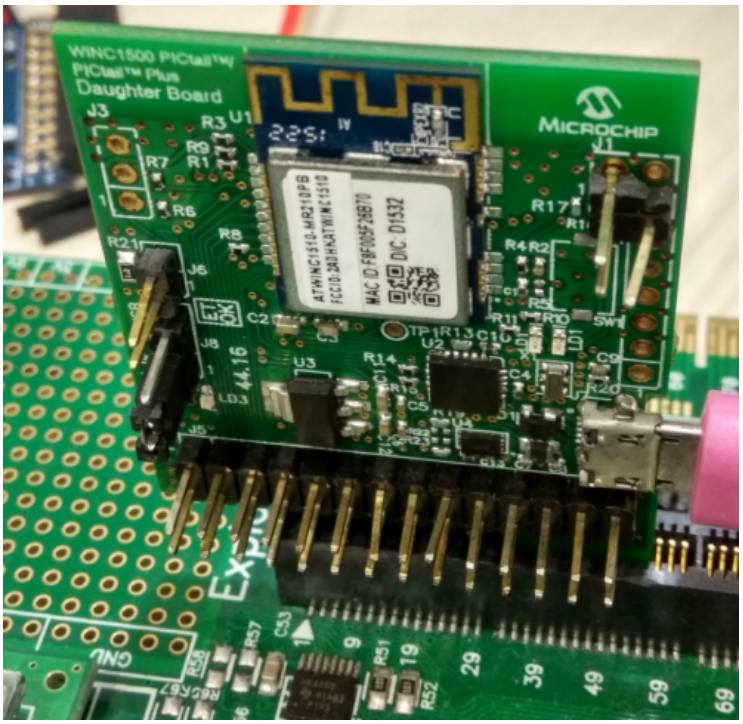
1. Connect the WINC1500 PICtail Daughter Board into the PICtail Plus slot closest to the MCU. The module should face into the board and be placed all the way to the end on the pin 1 side of the connector.
2. The console output uses the DB9 connector on the board at 115200 baud, 8-bit data, No parity, 1 Stop bit, with No Flow Control.

The following figure shows the hardware configuration.

Explorer 16 Development Board with PIC32MX795512L PIM



WINC1500 Wi-Fi PICtail/PICtail Plus Daughter Board



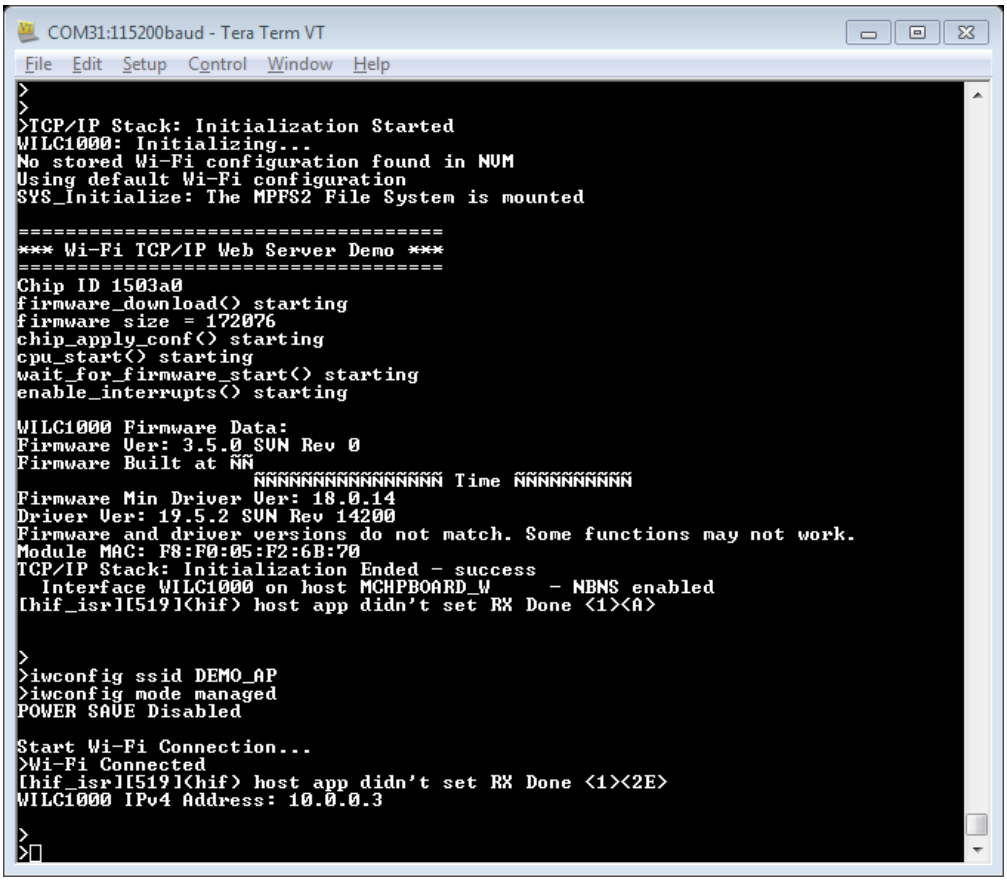
Running the Demonstration

This section provides instructions on how to run the wifi_wilc1000 demonstration.

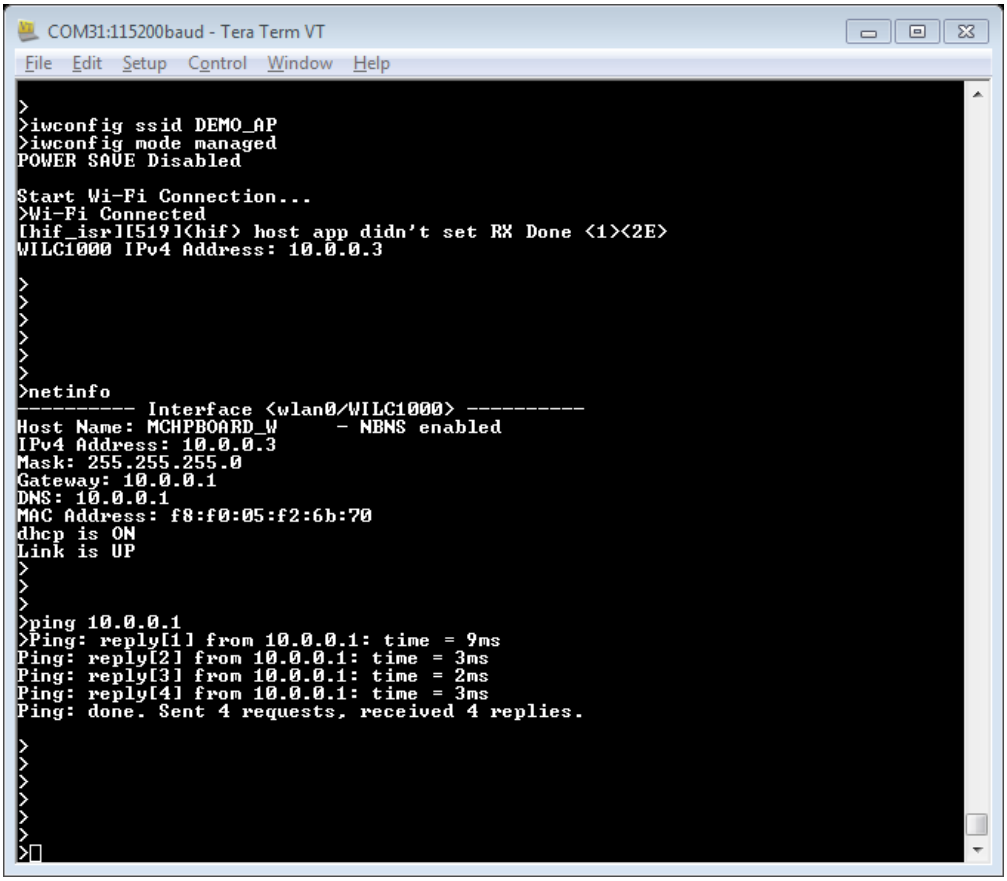
Description

This demonstration exercises various encryption, decryption, hashing, and random number functions.

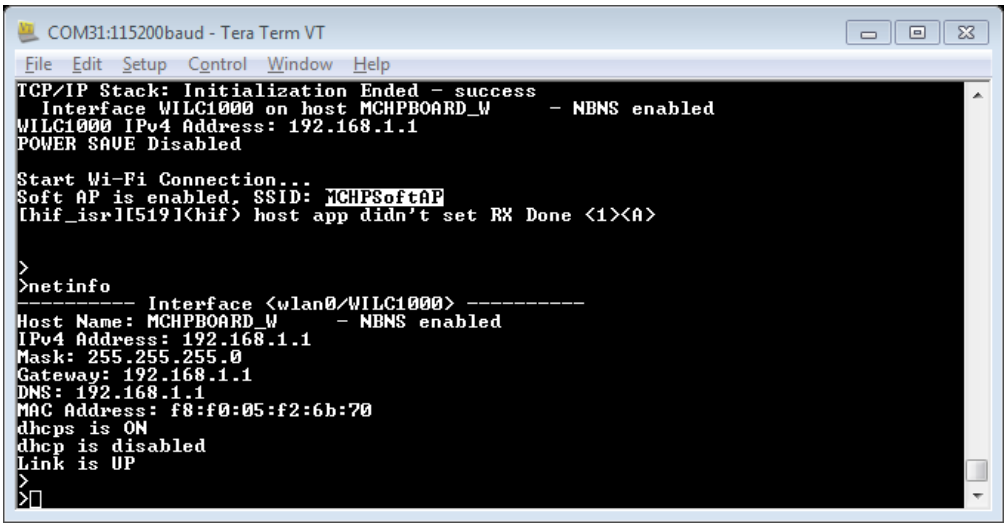
1. After compiling and programming the target device, select the configuration as pic32mx795_pim_e16_wilc_freertos_sta.
2. When the UART is connected by the serial to the USB cable, configure the Home AP configuration using the `iwconfig` console command. Refer to DRV WILC1000 WIFI Library Interface for additional information.



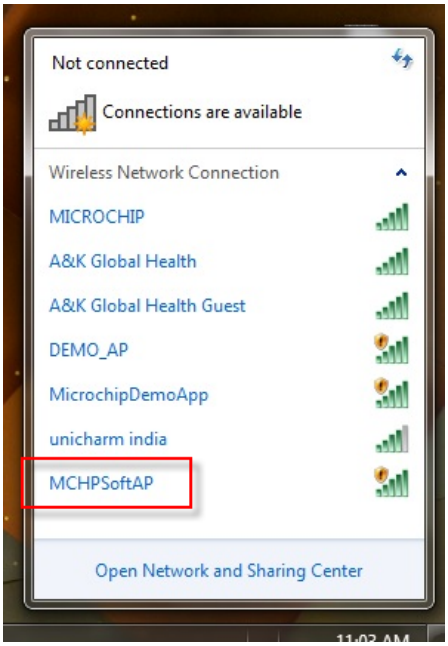
3. When receiving the IP from the Home AP, ping the Home AP.



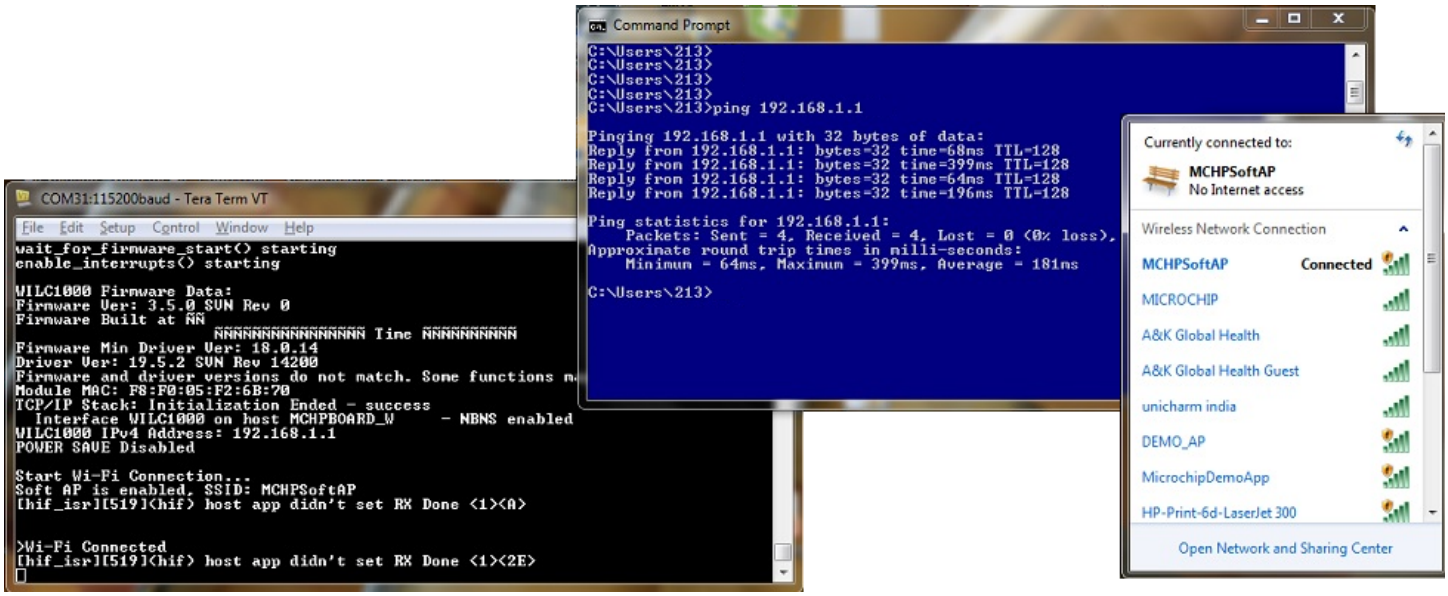
4. When creating the device to act as the AP, to compile and program the target device, select the configuration as pic32mx795_pim_e16_wilc_freertos_ap.
5. When the device appears as Soft AP, the AP can be found in the scan with SSID as 'MCHPSoftAP.'



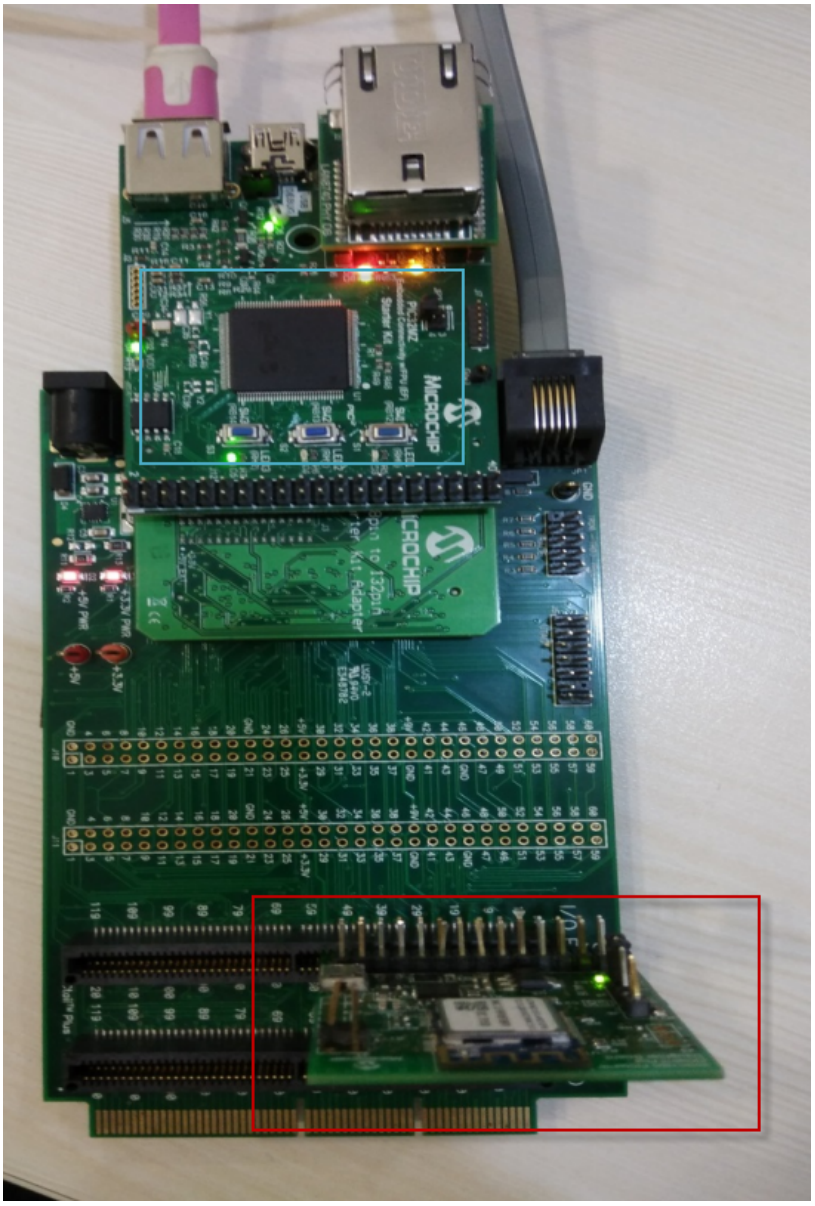
6. The SSID 'MCHPSoftAP' can be found in the scan list.



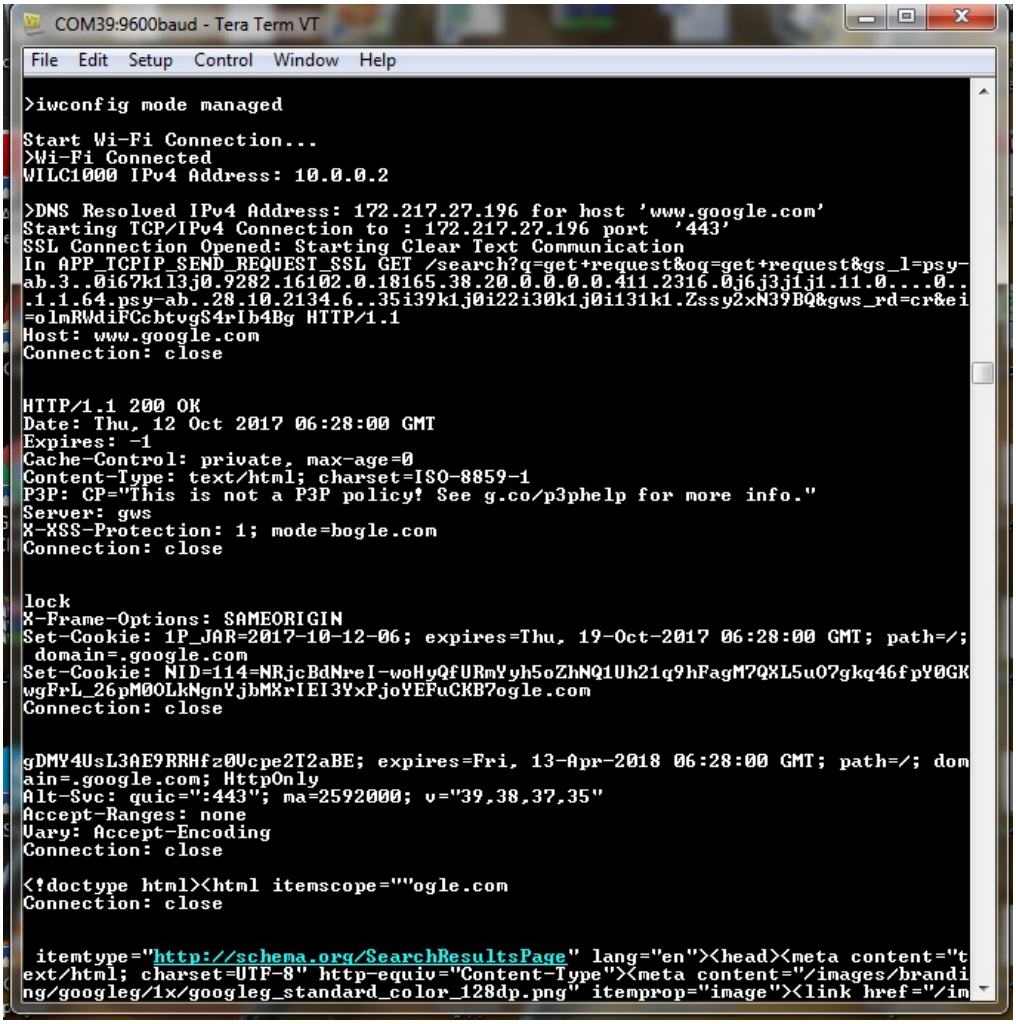
7. Any station can connect, and ping on the AP IP address, 192.168.1.1.



8. When the device serves as a wolfSSL client, to compile and program the target device, select the configuration pic32mz_ef_sk__ioexp__wilc__freertos_wolfssl_client.
9. For this configuration, the hardware is different. In this case, use the WINC1500 PICtail with the starter kit containing PIC32MZ2048. The highlighted red box is the WINC1500 Wi-Fi PICtail/PICtail Plus Daughter Board. The highlighted blue box is the PIC32MZ2048 Processor.



10. For serial communication and console output connect a mini-USB cable between the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Family Starter Kit and laptop. The serial connection is configured for 115200 baud, 8-bit data, No parity, 1 Stop bit, with No Flow Control. Configure the home AP using the `iwconfig` console command. Refer to DRV WILC1000 WIFI Library Interface for additional information.
11. When receiving the IP from the home AP, the device makes a DNS query to resolve the IP address for the host, “https://www.google.com."
12. The device then proceeds with SSL negotiations as the URL is secured (https).
13. The UART displays the contents of the web page.



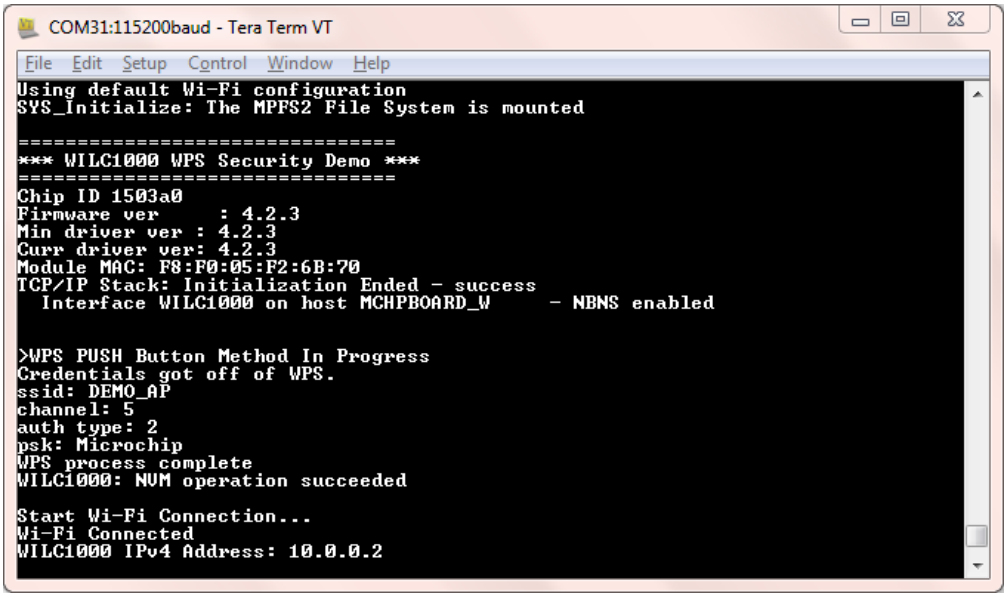
Running the Demonstration Using Various Configurations

This section demonstrates how to run the wifi_wilc1000 Demonstrations using other configurations.

Description

pic32mx795_pim__e16__wilc__freertos_wps

1. The demonstration comes as WPS in Push Button Mode.
2. The user needs to push the WPS Button on the AP for the device to connect to it.



3. In case the user wants to use the WPS PIN Mode, changes need to be made in the `wifi_wilc1000\firmware\src\wps.c` file
   - Macro MAIN_WPS_PUSH_BUTTON_FEATURE should be changed to *false*
   - Macro MAIN_WPS_PIN_NUMBER needs to be set

pic32mx795_pim__e16__wilc__freertos_p2p_client

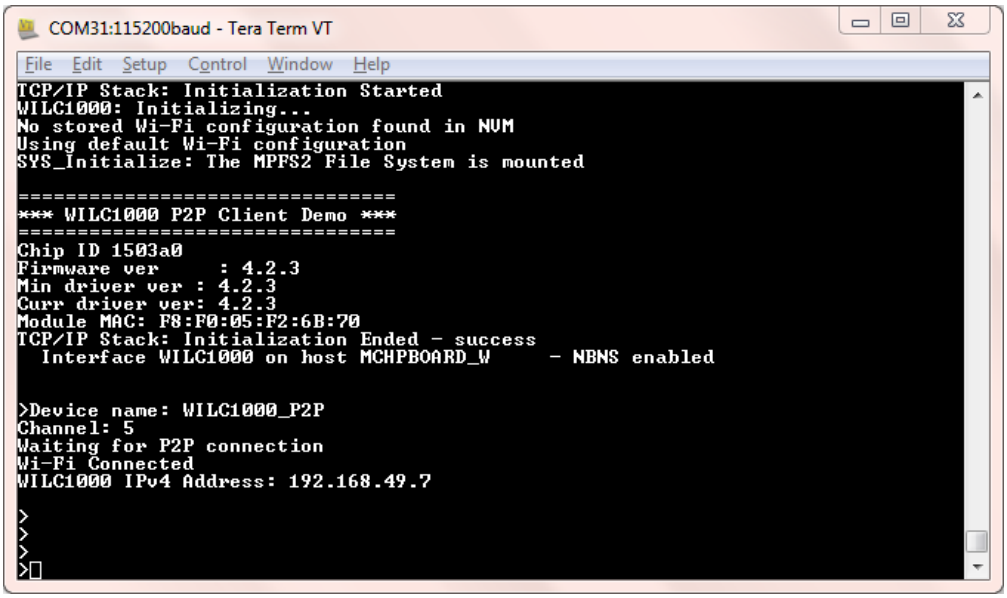
1. The P2P client comes up with the device name as *WILC1000_P2P*.
2. The other device can scan and find the same in its list.



3. The other device needs to initiate the connection.

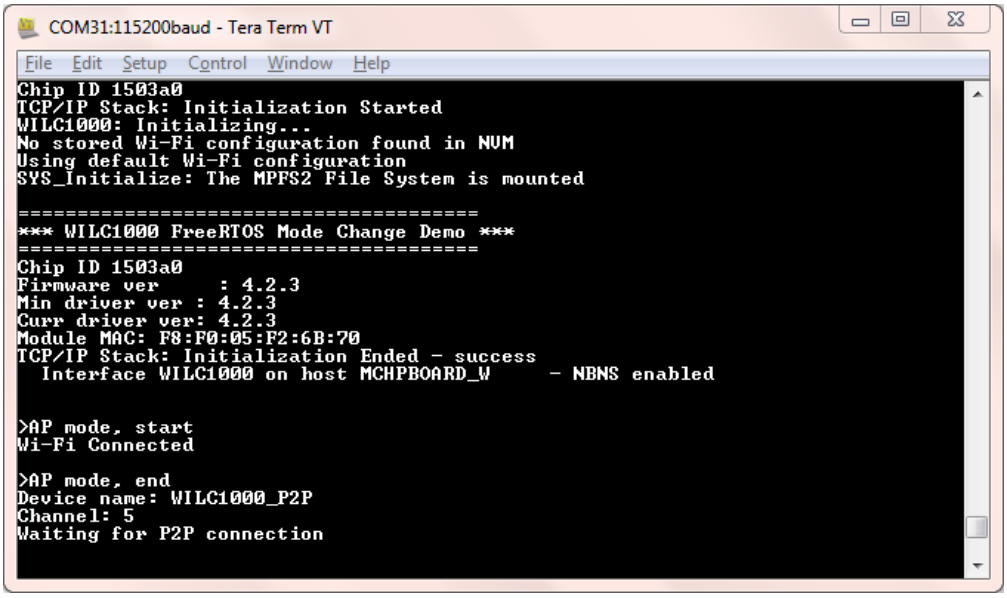


4. The Logs on the device will be as follows:



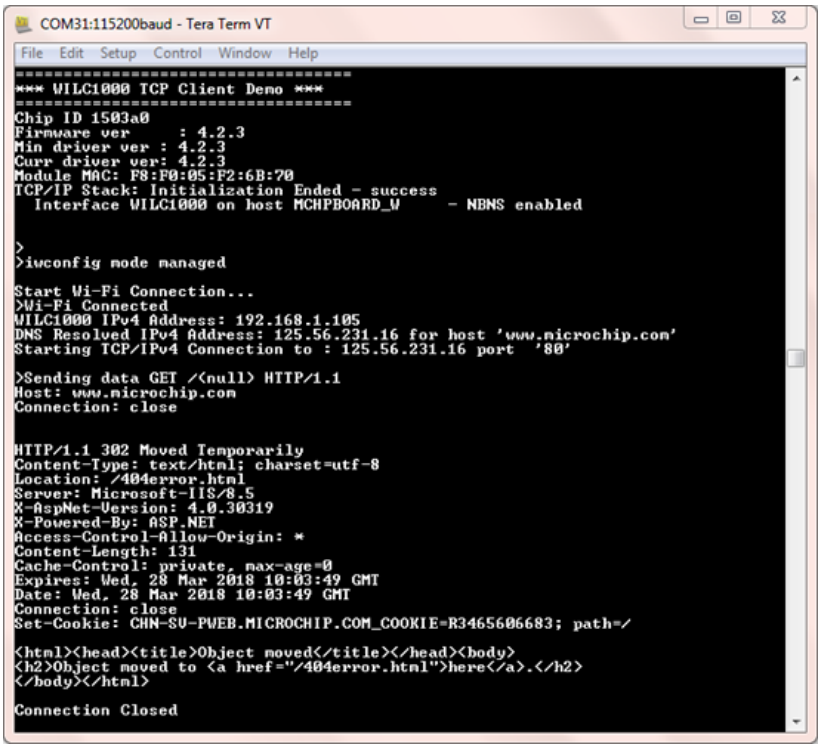
pic32mx795_pim__e16__wilc__freertos_mode_change

1. The device comes up in AP Mode with SSID *WILC1000_AP*.
2. The device remains in AP Mode for around a minute, and then changes the mode to P2P Client.



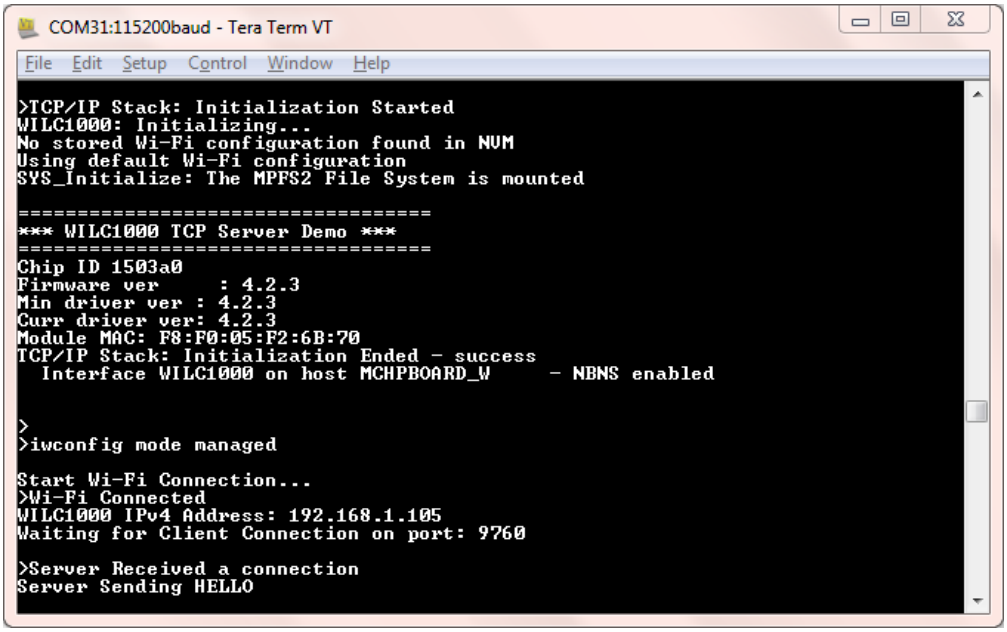
pic32mx795_pim__e16__wilc__freertos_tcp_client

1. When the device comes up, connect it to the AP using the command *iwconfig mode managed*.
2. Once connected, the demonstration will make a DNS query, and then open a TCP connection to microchip.com, and perform a simple HTTP GET command.

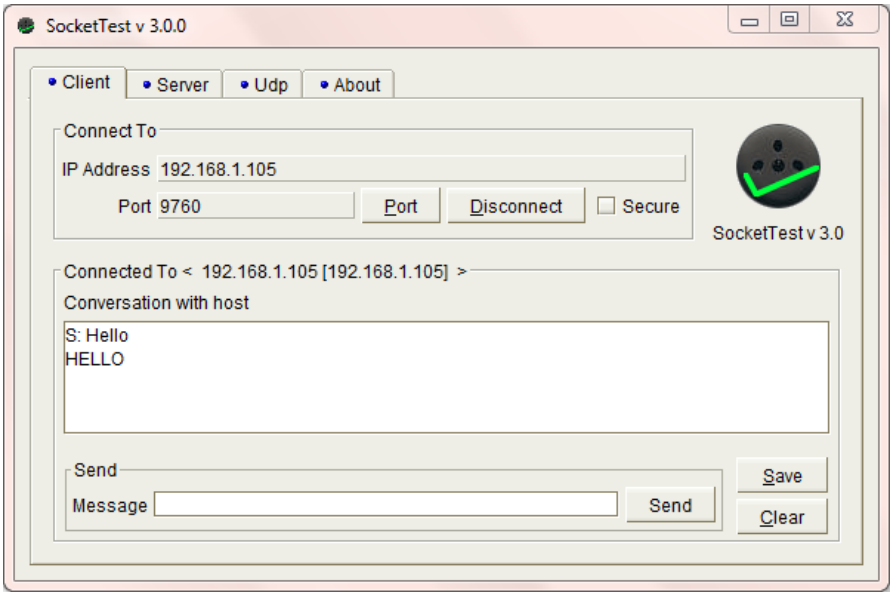


pic32mx795_pim__e16__wilc__freertos_tcp_server

1. When the device comes up, connect it to the AP using the command *iwconfig mode managed*.
2. Once connected, the demonstration starts a TCP Server on port 9760 with the IP assigned by the AP.



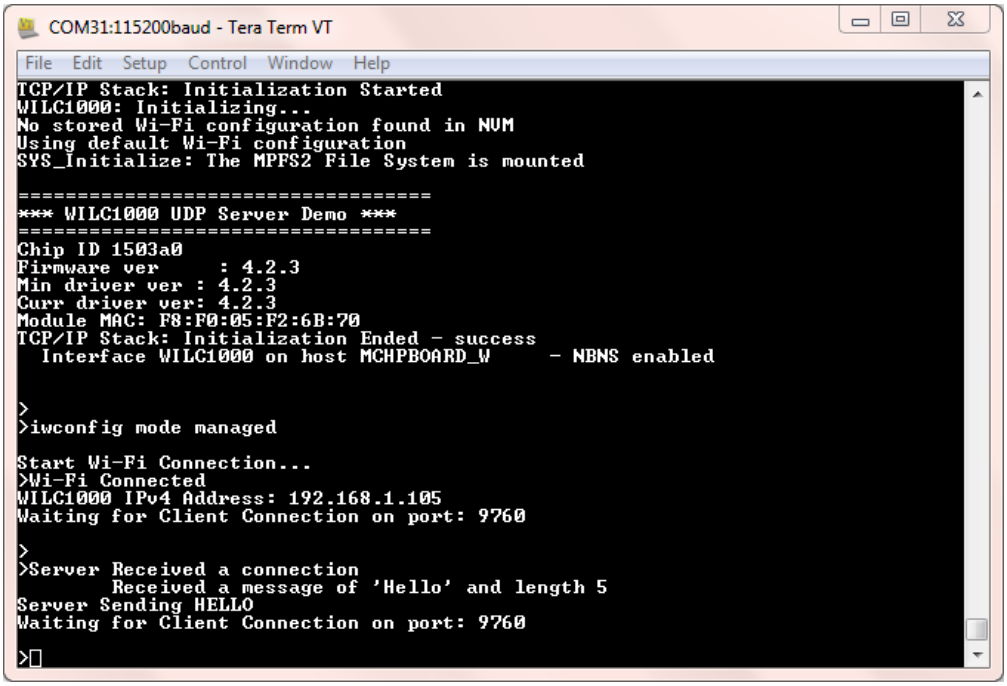
3. Start a TCP Client on another device (SocketTest3 in this example), and connect to this TCP Server.



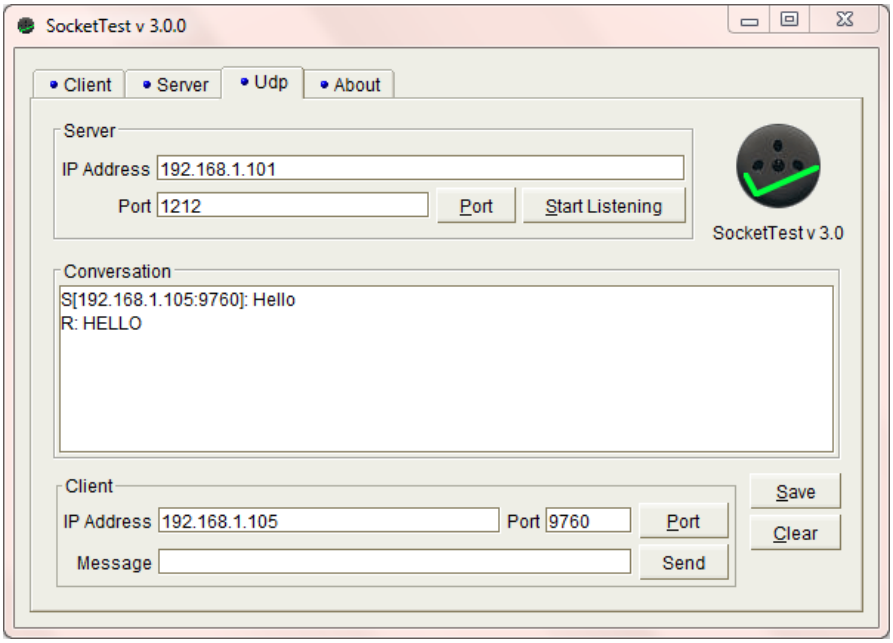
4. The TCP server running on the device shall send back the string received from the Client in CAPS. Therefore, if the TCP Client sends the Server “Hello”, the Server shall send “HELLO” back to the Client.

pic32mx795_pim__e16__wilc__freertos_udp_server

1. When the device comes up, connect it to the AP using the command *iwconfig mode managed*.
2. Once connected, the demonstration starts a UDP Server on port 9760 with the IP assigned by the AP.



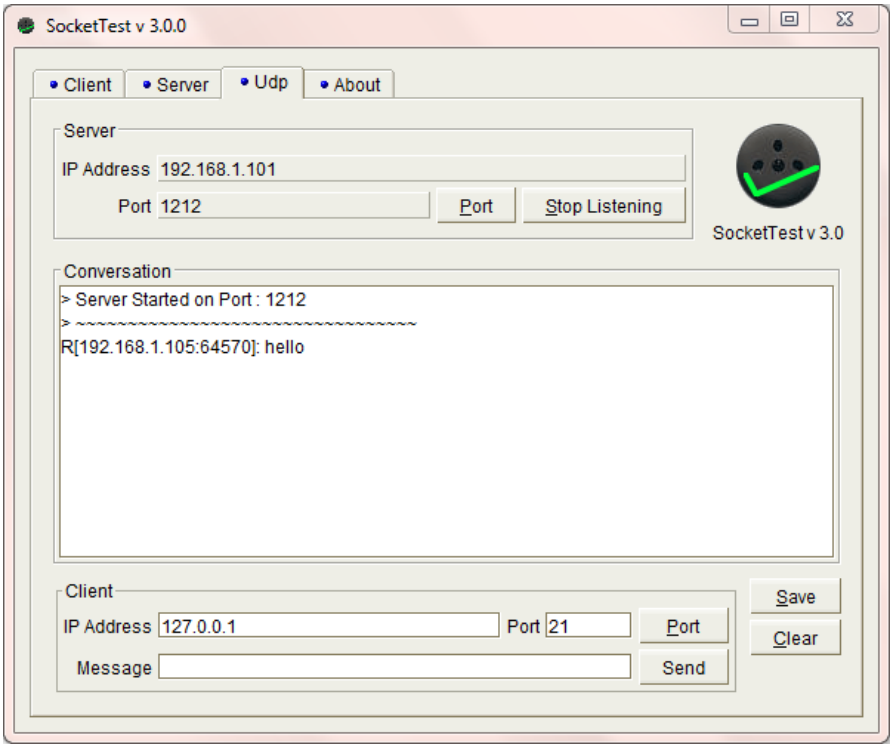
3. Start a UDP Client on another device (SocketTest3 in this example), and connect to this UDP Server.



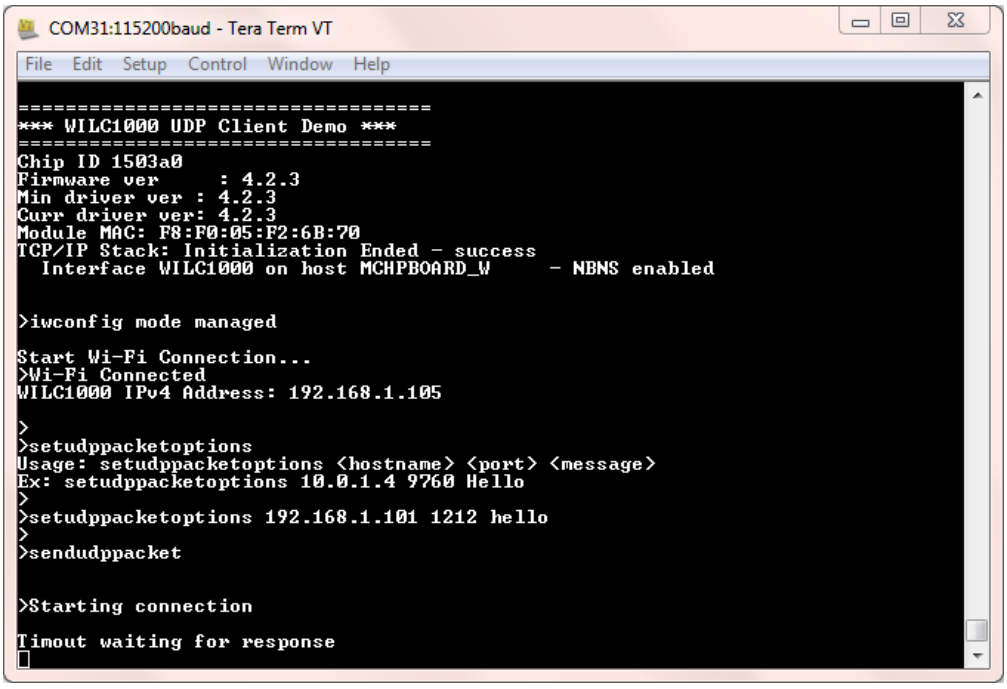
4. The UDP server running on the device shall send back the string received from the Client in CAPS. Therefore, if the UDP Client sends the Server “Hello," the Server shall send “HELLO” back to the Client.

pic32mx795_pim__e16__wilc__freertos_udp_client

1. When the device comes up, connect it to the AP using the command *iwconfig mode managed*.
2. Start a UDP Server on another device (SocketTest3 in this example).

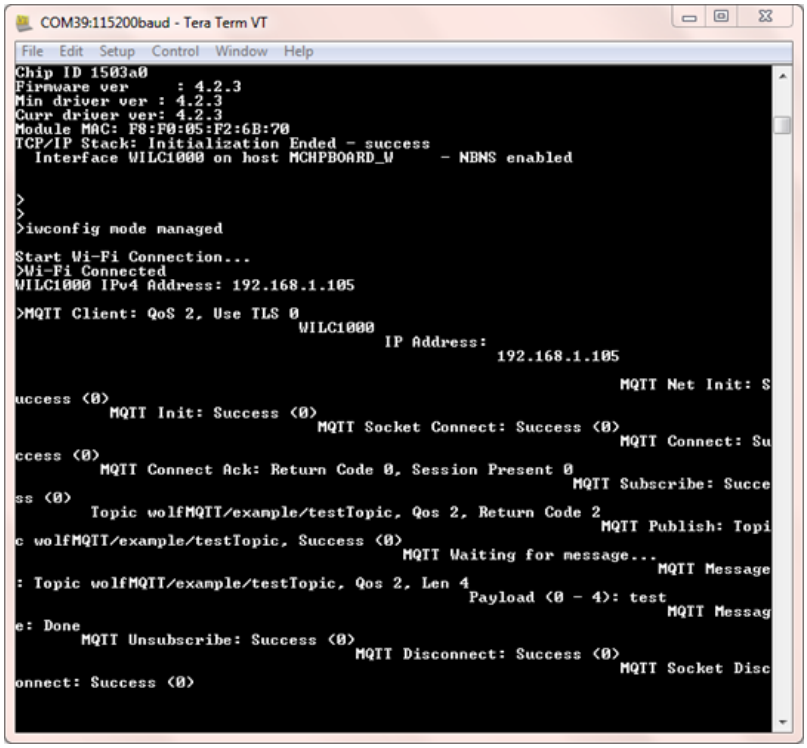


3. This demonstration has three commands:
   - setudppacketoptions - Sets the current hostname, port, and message
   - getudppacketoptions - Gets the hostname, port and message
   - sendudppacket - Sends the UDP Packet
4. Use “setudppacketoptions” to set the IP, port of the UDP Server and the message to be sent to the UDP Server. In this example, setudppacketoptions 192.168.1.101 1212 hello.
5. After setting the options, send the packet to the server using the command *sendudppacket*.



pic32mz_ef_sk__ioexp__wilc__freertos_wolfmqtt_client

1. Connect the device to the Access Point using the command *iwconfig mode managed*.
2. After connecting to the AP, the MQTT Protocol gets triggered automatically and Client connects to the Server.



wifi_winc1500_socket

This section provides information on the WINC1500 Socket Mode Driver demonstrations.

Description

Before using this demonstration, please see the important notes in the *TCP/IP Demonstrations* > *Introduction*.

This demonstration has many examples that can be configured and run individually and demonstrate the features support by the WINC1500 Wi-Fi module. The examples are defined in the file `app.h`, and the detail for each example is described in their related `.c` files.

Demo Example Configurations (app.h)	Description	.c file
#define AP_SCAN_EXAMPLE	Scans APs around you and displays the results, and then connects to the target AP.	ap_scan.c
#define CHIP_INFO_GET_EXAMPLE	Gets the chip information of WINC1500.	chip_info_get.c
#define EMAIL_SEND_EXAMPLE	Demonstrates email sending.	email_send.c
#define HTTP_DOWNLOAD_EXAMPLE	Demonstrates file downloading by HTTP client.	http_download.c
#define IP_ADDR_LOCATE_EXAMPLE	Gets current location where my IP is used.	ip_addr_locate.c
#define MAC_ADDRESS_GET_EXAMPLE	Gets the MAC address of WINC150.	mac_address_get.c
#define MDNS_EXAMPLE	Demonstrates mDNS server with service discovery support.	mdns.c
#define MODE_AP_EXAMPLE	Demonstrates starting SoftAP with WPA, WEP or open security.	mode_ap.c
#define MODE_CLIENT_STA_EXAMPLE	Demonstrates client mode connection with WPA.	mode_client_sta.c
#define MULTI_SOCKET_EXAMPLE	Demonstrates running multiple TCP clients at the same time.	multi_socket_example.c
#define P2P_EXAMPLE	Demonstrates Wi-Fi Direct GC function.	p2p.c
#define POWER_SAVE_EXAMPLE	Demonstrates power save features.	power_save.c
#define PROVISION_AP_EXAMPLE	Demonstrates provisioning using SoftAP function through Android application.	provision_ap.c
#define PROVISION_HTTP_EXAMPLE	Demonstrates provisioning using SoftAP function through webpage.	provision_http.c
#define PUBNUB_CLOUD_EXAMPLE	Demonstrates publishing and subscribing using PubNub.	pubnub_cloud.c
#define SECURITY_WEP_WPA_EXAMPLE	Demonstrates infrastructure network connection with WPA, WEP or open security.	security_wep_wpa.c
#define SIMPLE_GROWL_EXAMPLE	Demonstrates notification transmitting among WINC1500, public remote server and smartphone application.	simple_growl.c

#define SSL_CLIENT_EXAMPLE	Demonstrates SSL client.	ssl_client.c
#define SSL_SERVER_EXAMPLE	Demonstrates SSL server.	ssl_server.c
#define TCP_CLIENT_EXAMPLE	Demonstrates TCP client.	tcp_client.c
#define TCP_SERVER_EXAMPLE	Demonstrates TCP server.	tcp_server.c
#define TIME_CLIENT_EXAMPLE	Demonstrates SNTP client.	time_client.c
#define UDP_EXAMPLE	Demonstrates UDP server and client.	udp.c
#define UDP_CLIENT_EXAMPLE	Demonstrates UDP client.	udp_client.c
#define UDP_SERVER_EXAMPLE	Demonstrates UDP server.	udp_server.c
#define WEATHER_CLIENT_EXAMPLE	Weather client retrieves weather information of the target location using HTTP query.	weather_client.c
#define WPS_CONNECT_EXAMPLE	Demonstrates WPS security in client mode.	wps_connect.c
#define FW_UPDATE_OTA	Supports FW update over the air (OTA).	fw_update_ota.c
#define FW_UPDATE_OVER_SERIAL	Supports FW update over serial port.	fw_update_over_serial.c

WINC1500 Socket Examples Guide

This section introduces the WINC1500 Socket examples and describes how to run each example on the Explorer 16 Development Board.

Organization of WINC1500 Socket Examples

Provides information on the organization of the socket examples for the WINC1500 Wi-Fi Demonstration.

Description

Basic Examples

These examples describe basic Wi-Fi operation in a ‘how-to’ manner:

- How to read chip ID (to identify WINC1500 H/W revision)
- How to get MAC address of the Wi-Fi module
- How to start Wi-Fi in specific operation mode, such as:
  - STA Mode (Station mode, known as a Wi-Fi client)
  - AP mode (Access Point mode)
  - P2P mode (Peer-to-Peer mode, also known as Wi-Fi Direct®)
- How to switch mode among STA, AP, and P2P modes during the runtime
- How to scan APs that are nearby
- How to set deep sleep mode
- How to connect to secure Wi-Fi using WEP or WPA/WPA2 personal security
- How to connect to WPA/WPA2 enterprise security network
- How to connect to security WPS
- How to get RF signal status by reading RSSI value
- How to set AP provision
- How to set HTTP provision

Protocol Examples

After basic code examples, user may want to explore how to send and receive network packets. Here are protocol examples that can be extended for IoT application.

- UDP protocol example
  - Server and Client
  - Client
  - Server
- TCP protocol example
  - Client
  - Server
- NTP Time client – retrieve network time for IoT application
- Send email – send an email through SMTP server

Advanced Examples

These examples demonstrate more complex functions like:

- Weather client – get the current weather information of the network provider and utilize the IO1 sensor device
- SSL connection - Set up an SSL connection
- Multi-Socket - Use Ethernet and Wi-Fi sockets
- PubNub cloud – Access cloud device
- Zeroconfig or mDNS – Service or Device discovery

MPLAB Harmony WINC1500 Socket Examples

Provides information on the socket examples provided in your installation of MPLAB Harmony.

Description

The MPLAB Harmony WINC1500 socket demonstration project provides many example configurations, but only one example at a time can be configured and run. The following table lists the available examples in MPLAB Harmony, which can be located in the file, `app.h`.

WINC1500 Demonstration Examples

Example Configuration	Description
AP_SCAN_EXAMPLE	Scans APs around you and displays the results, and then connects to the target AP
CHIP_INFO_GET_EXAMPLE	Gets the chip information of WINC1500
EMAIL_SEND_EXAMPLE	Demonstrates email sending
HTTP_DOWNLOAD_EXAMPLE	Demonstrates file downloading by HTTP client
IP_ADDR_LOCATE_EXAMPLE	Gets current location where my IP is used
MAC_ADDRESS_GET_EXAMPLE	Gets the MAC address of WINC1500
MDNS_EXAMPLE	Demonstrates mDNS server with service discovery support
MODE_AP_EXAMPLE	Demonstrates starting SoftAP with WPA, WEP or open security
MODE_CLIENT_STA_EXAMPLE	Demonstrates client mode connection with WPA
MULTI_SOCKET_EXAMPLE	Demonstrates running multiple TCP clients at the same time
P2P_EXAMPLE	Demonstrates Wi-Fi Direct GC function
POWER_SAVE_EXAMPLE	Demonstrates power save features
PROVISION_AP_EXAMPLE	Demonstrates provisioning using SoftAP function
PUBNUB_CLOUD_EXAMPLE	Demonstrates publish and subscribe using PubNub
SECURITY_WEP_WPA_EXAMPLE	Demonstrates client mode connection with WPA
SSL_CLIENT_EXAMPLE	Demonstrates SSL client
SSL_SERVER_EXAMPLE	Demonstrates SSL server
TCP_CLIENT_EXAMPLE	Demonstrates TCP client
TCP_SERVER_EXAMPLE	Demonstrates TCP server
TIME_CLIENT_EXAMPLE	Demonstrates SNTP client
UDP_EXAMPLE	Demonstrates UDP server and client
UDP_CLIENT_EXAMPLE	Demonstrates UDP client
UDP_SERVER_EXAMPLE	Demonstrates UDP server
WEATHER_CLIENT_EXAMPLE	Weather clients retrieves weather information of the target location using HTTP query
WPS_CONNECT_EXAMPLE	Demonstrates WPS security in client mode
IOT_SUPPORT	Enables additional IOT supporting features, for instance, HTTP client

Prerequisites

Provides information on the WINC1500 socket example prerequisites.

Description

Development Platform

Development Platforms	MCU	WINC1500	Comment
Option 1: • Microchip Explorer 16 Development Board (DM240001) • MPLAB ICD3 In-Circuit Debugger (DV164035) • 9V Power supply (AC002014) • DB9 Serial Cable or USB/Serial Cable • Windows7 PC or laptop • Wi-Fi Access Point (AP) • Android 5.0 or later smart device • Internet Access	PIC32MX795F512L USB/CAN Plug-in Module (MA320003)	WINC1500 PICtail Plus module	Hardware Platform support in MPLAB Harmony v2.03b for Windows.

MPLAB Harmony WINC1500 Development Platform with a Laptop Running Windows



WINC1500 Wi-Fi PICtail Module Connected to the Explorer 16 Development Board

Software Prerequisites for Windows

- WINC1500 MPLAB Harmony v2.0.4b or later
- MPLAB X IDE v4.0 or later
- MPLAB XC32 C/C++ Compiler v1.43 or later
- A Terminal console, such as TeraTerm

Internet Services

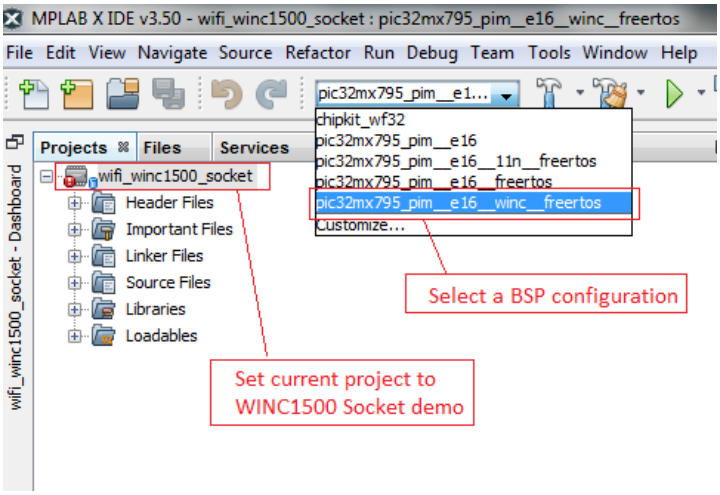
- Cloud Service
- Weather Server
- NDP Server

Assigning the IDC and XC32 Compilers

Provides information on assigning the IDC and compiler.

Description

Refer to the following diagram to assign the IDC and XC32 compiler to the active demonstration project, wifi_winc1500_socket. The MPLAB X IDE detects your IDC automatically.



Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the PIC32 WINC1500 Socket demonstration.

Description

The WINC1500 socket demonstration contains many examples and only one example can be built and run at a time. The example can be selected from the `app.h` file. For example, to run the demonstration with the AP_SCAN_EXAMPLE, edit the `app.h` file and set:

```
#define AP_SCAN_EXAMPLE     1
```

To build this project, you must open the `wifi_winc1500_socket.X` project in MPLAB X IDE, and then select the desired configuration. The following tables list and describe the project and supported configurations. The parent folder for these files is

`<install-dir>/apps/tcpip/wifi_winc1500_socket.`

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`wifi_winc1500_socket.X`	`<install-dir>/apps/tcpip/wifi_winc1500_socket/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are within

`./firmware/src/system_config.`

Project Configuration Name	BSP(s) Used	Description
pic32mx795_pim_e16_winc_freertos	pic32mx795_pim+e16	FreeRTOS version of the demonstration running on the Explorer 16 or Explorer 16/32 Development Board, or the Explorer 16/32 Development Board with the PICtail Plus Expansion Board, PIC32MX795F512L CAN-USB PIM and the WINC1500 Wi-Fi PICtail/PICtail Plus Daughter Board.
pic32mz_ef_sk_ioexp_winc_freertos	pic32mz_ef_sk	FreeRTOS version of the demonstration running on the PIC32MZ EF Starter Kit connected to an I/O Expansion board with the WINC1500 PICtail Daughter Board.
pic32mx795_pin_e16_wincclick_freertos	pic32mx795_pim+e16	FreeRTOS version of the demonstration running on the Explorer 16/32 Development Board, or the Explorer 16/32 Development board with the PIC32MX795F512L CAN-USB PIM, PICtail Plus Expansion Board, and the MikroElektronika WiFi 7 Click Board (with on-board WINC1500 Wi-Fi module).

Configuring the Hardware

This section describes how to configure the supported hardware.

Description

PIC32MX795F512L CAN-USB PIM with the Explorer 16 Development Board and WINC1500 Wi-Fi PICtail Daughter Board

- Connect the WINC1500 PICtail Daughter Board into the PICtail Plus slot closest to the MCU. The module should face into the board and be placed all the way to the end on the pin 1 side of the connector.
- The console output uses the DB9 connector on the board at 115200 baud, 8-bit data, No parity, 1 Stop bit, with No Flow Control

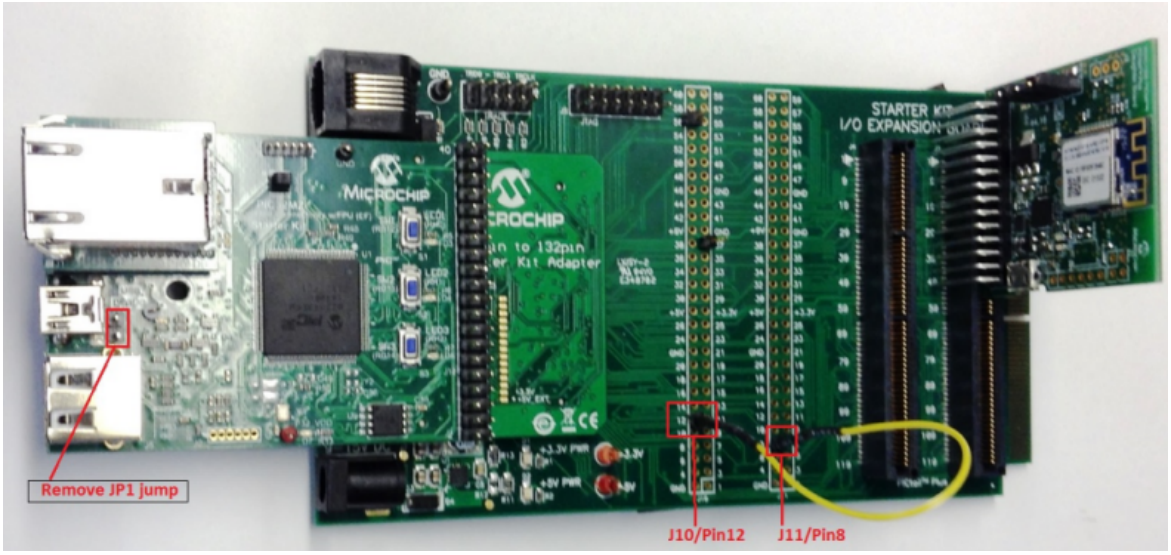
The following figure shows the hardware configuration.

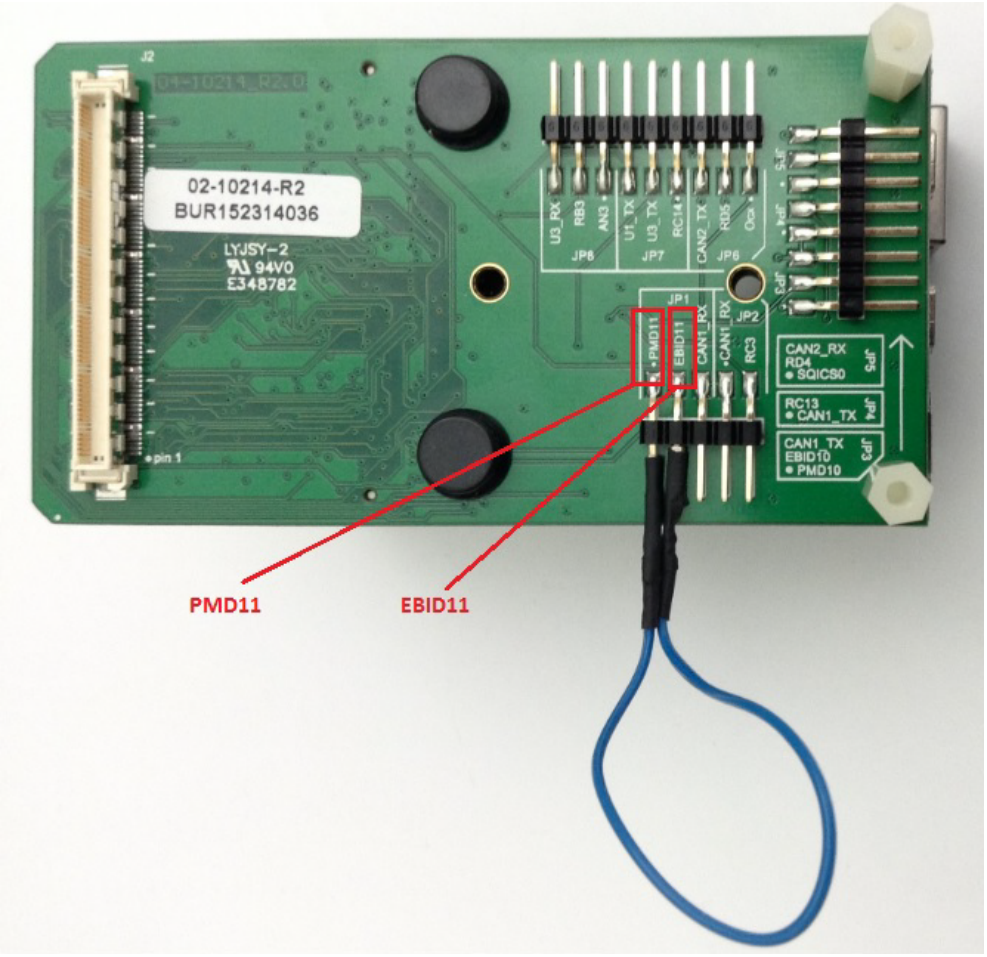


PIC32MZ EF Starter Kit connected to an Starter Kit I/O Expansion Board with the WINC1500 Wi-Fi PICtail Daughter Board

- Connect the WINC1500 PICtail Daughter Board into the J2 PICtail Plus slot on the IO Expansion board. The module should face into the board and be placed all the way to the end on the pin 1 side of the connector.
- Connect a jump cable (yellow cable in the picture below) between J10/Pin12 to J11/Pin8 on the I/O Expansion board. This jumper is required for reset of the WINC1500 module.
- Connect the AC adapter to the I/O Expansion board. This external power is required as power supply from USB host connection may not be efficient.
- Remove the JP1 jump on the PIC32MZ EF Starter Kit board.
- For serial communication and console output connect a mini-USB cable between the PIC32MZ EF Starter Kit board and laptop. The serial connection is configured for 115200 baud, 8-bit data, No parity, 1 Stop bit, with No Flow Control.

The following figure shows the hardware configuration.

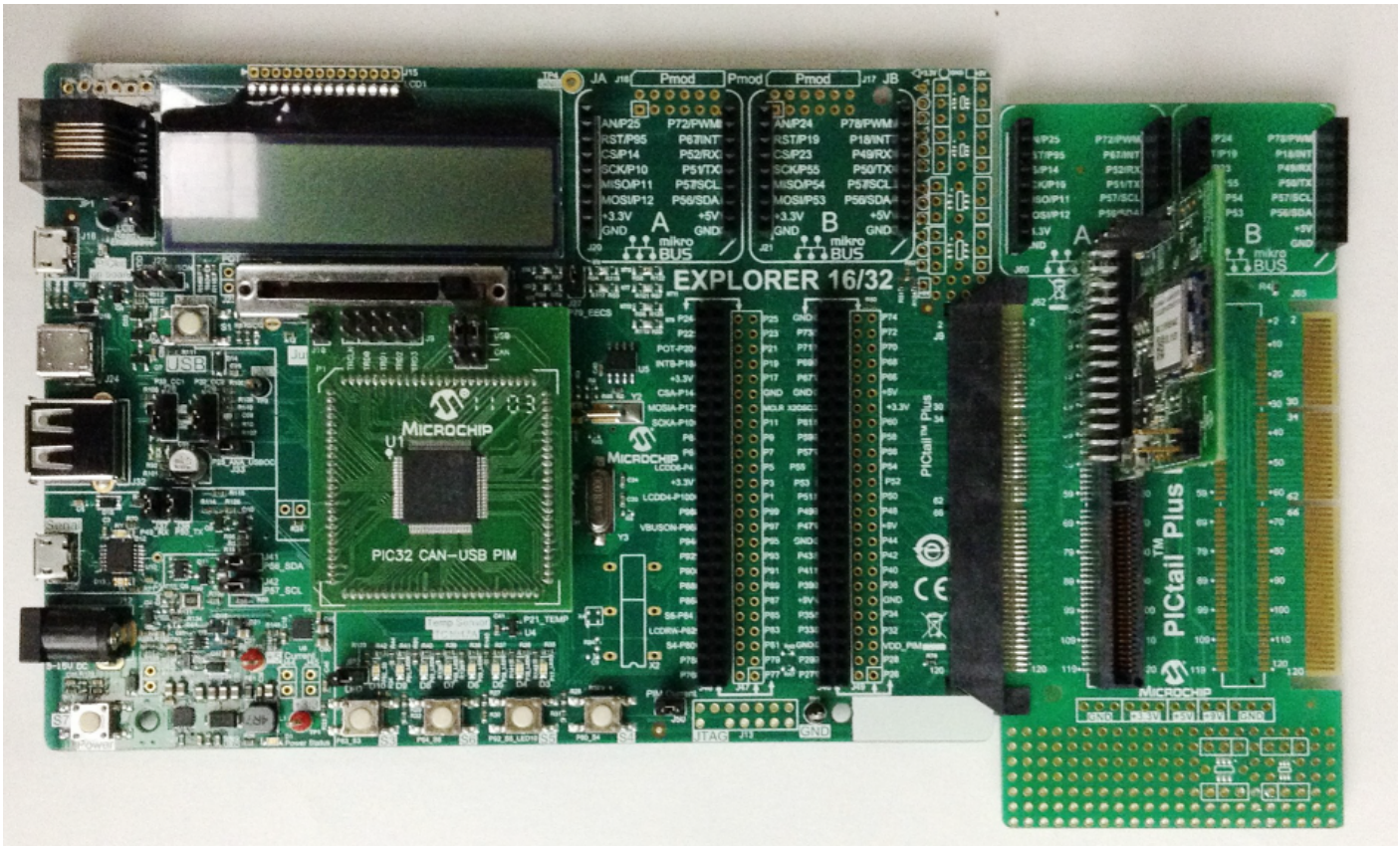




Explorer 16/32 Development Board with the PIC32MX795F512L CAN-USB PIM, PICtail Plus Expansion Board, and WINC1500 Wi-Fi PICtail/PICtail Plus Daughter Board

- Connect the WINC1500 Wi-FI PICtail/PICtail Plus Daughter Board into the PICtail Plus slot (J63). The module should face into the main board and be placed all the way to the end on the pin 1 side of the connector.
- The console output uses the mini_USB/Serial connector on the board (J40) at 115200 baud, 8-bit data, No parity, 1 Stop bit, with No Flow Control.

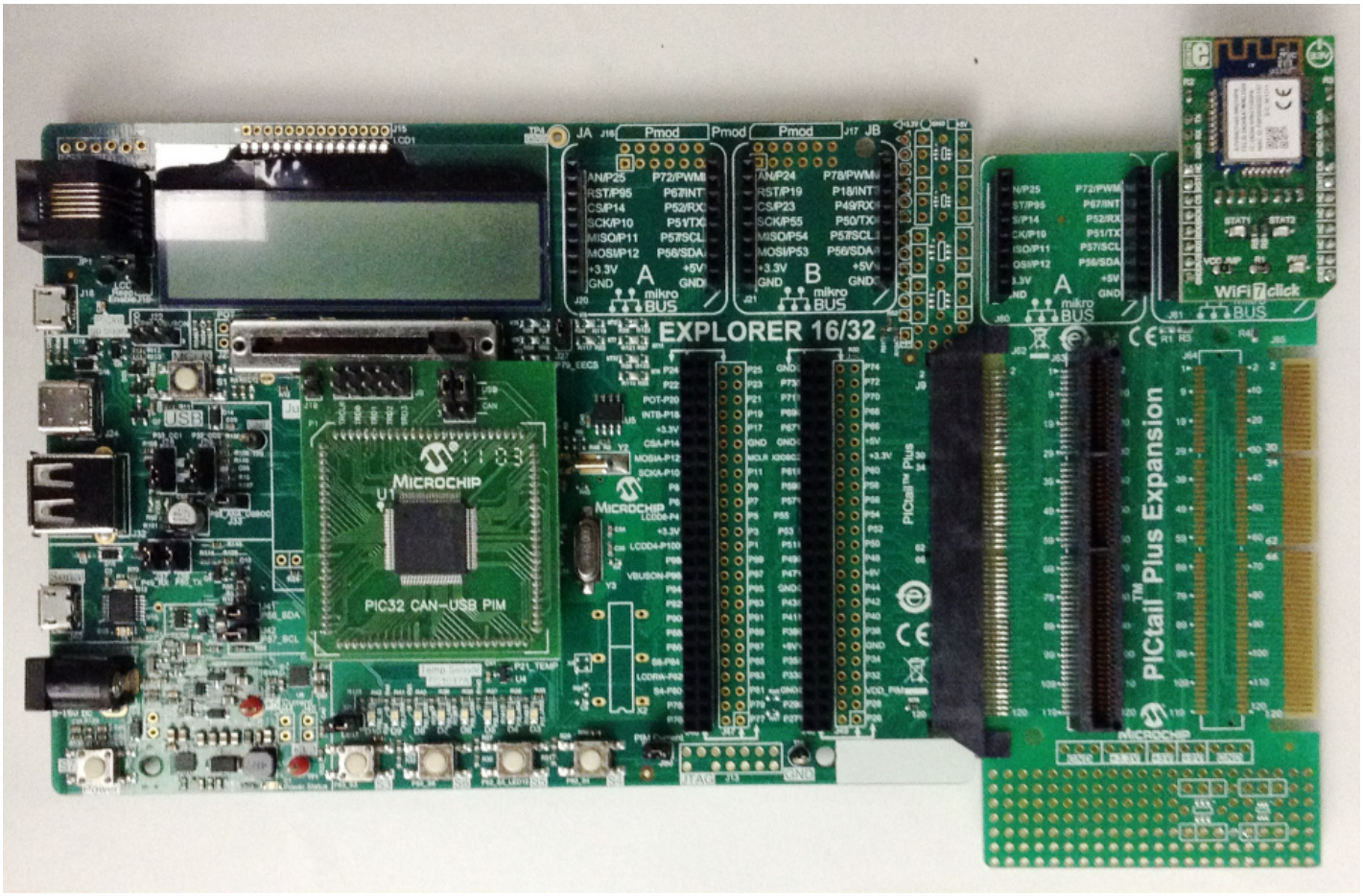
The following figure shows the hardware configuration.



Explorer 16/32 Development Board with the PIC32MX795F512L CAN-USB PIM, PICtail Plus Expansion Board, and the MikroElektronika WiFi 7 Click Board (with on-board WINC1500 Wi-Fi module).

- Connect the WINC1500 Wi-Fi PICtail/PICtail Plus Daughter Board into the PICtail Plus slot (J63) . The module should face into the main board and be placed all the way to the end on the pin 1 side of the connector.
- The console output uses the mini_USB/Serial connector on the board (J40) at 115200 baud, 8-bit data, No parity, 1 Stop bit, with No Flow Control

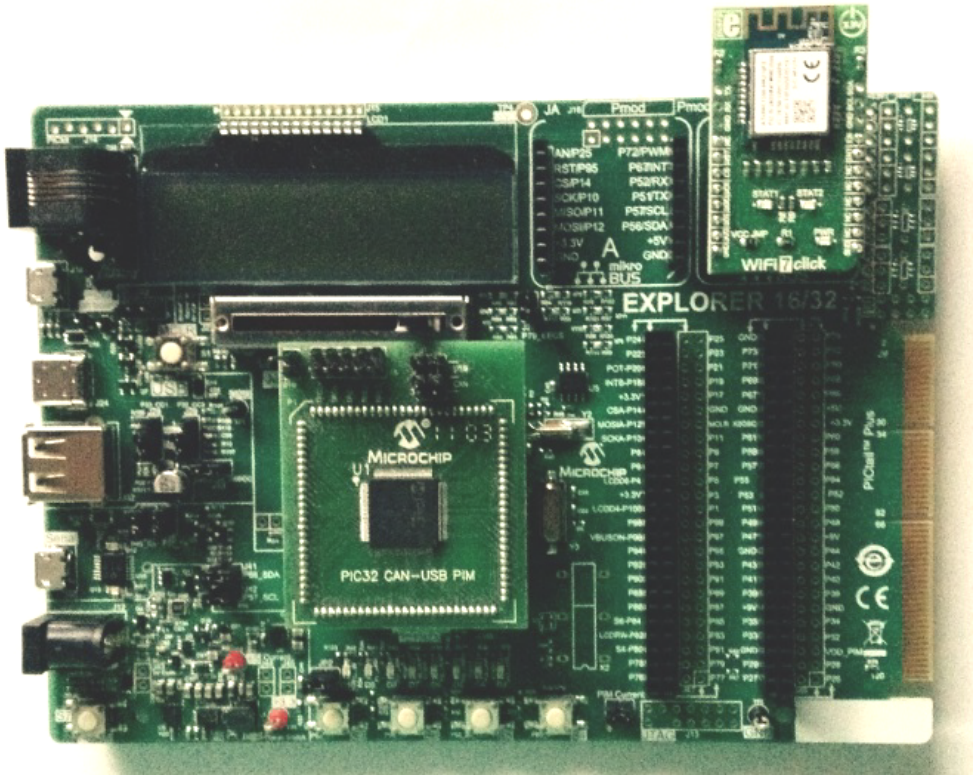
The following figure shows the hardware configuration.



Explorer 16/32 Development Board with the PIC32MX795F512L CAN-USB PIM and the MikroElektronika WiFi 7 Click Board (with on-board WINC1500 Wi-Fi module).

- Connect the WINC1500 Wi-Fi PICtail/PICtail Plus Daughter Board into the PICtail Plus slot. The module should face into the main board and be placed all the way to the end on the pin 1 side of the connector.
- The console output uses the mini_USB/Serial connector on the board (J40) at 115200 baud, 8-bit data, No parity, 1 Stop bit, with No Flow Control

The following figure shows the hardware configuration.



Running the Demonstration

This section provides instructions on how to build and run the WINC1500 Socket demonstration with AP-SCAN-EXAMPLE. Other examples can be run by following the similar procedure steps describe below. Detailed instructions for the example can be found in the comments in the example `.c` file.

Description

To run the AP-SCAN-EXAMPLE demonstration follows these steps:

1. In MPLAB X IDE, load the project file `wifi_winc1500_socket.X`.
2. Select the project configuration BSP.
3. Edit the file `app.h` and set:

```
#define AP_SCAN_EXAMPLE     1
```

4. Edit the file `ap_scan.c` and set the configuration for the example:

```
#if AP_SCAN_EXAMPLE
#define WLAN_SSID                    "DEMO_AP" /* Target AP */
#define WLAN_AUTH                    M2M_WIFI_SEC_WPA_PSK /* AP Security */
#define WLAN_PSK                     "12345678" /* Security Passphrase (If WPA Security Used) */
#define WLAN_WEP_KEY                 "1234567890" /* Security Key (If WEP Security Used) */
#define WLAN_WEP_KEY_INDEX          1 /* Security Key Index (If WEP Security Used) */

#define PING_ADDRESS                 "192.168.1.1" /* Address to Ping after Connection */
#define PING_COUNT                   3 /* Number of Times to Ping */
#define PING_INTERVAL                100 /* Wait 100ms between Pings */
```

5. Build and run the demonstration.
6. The example console outputs can be captured with TeraTerm, as shown in the following figure.



wolfssl_tcp_server

wolfSSL TCP Server demonstration.

Description

This configuration demonstrates creating a simple Internet Web server, that operates with clear text (TCP Port 80), and with encrypted text (TCP Port 443). If IPv6 is enabled than the demonstration also serves both types of connections on IPv6. The Web server only serves one page with the text 'Nothing Here' to all Web clients.

Before using this demonstration, please see the important notes in the *TCP/IP Demonstrations* > *Introduction*.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the wolfSSL TCP Client Demonstration.

Description

To build this project, you must open the `wolfssl_tcp_server.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/tcpip/wolfssl_tcp_server`.

Warning

When using the Microchip Harmony Configurator (MHC), care must be taken when generating the code to *not* erase the USB descriptors in the `system_init.c` file.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`wolfssl_tcp_server.X`	`<install-dir>/apps/tcpip/wolfssl_tcp_server/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx_eth_sk	pic32mx_eth_sk	Demonstrates the wolfSSL TCP Server on the PIC32 Ethernet Starter Kit.
pic32mx_eth_sk2	pic32mx_eth_sk2	Demonstrates the wolfSSL TCP Server on the PIC32 Ethernet Starter Kit II.
pic32mz_ec_sk	pic32mz_ec_sk	Demonstrates the wolfSSL TCP Server on the PIC32MZ EC Starter Kit.
pic32mz_ef_sk	pic32mz_ef_sk	Demonstrates the wolfSSL TCP Server on the PIC32MZ EF Starter Kit.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 Ethernet Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32 Ethernet Starter Kit II

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ EC Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

This section provides instructions on how to build and run the demonstration.

Description

For PIC32M-based Starter Kits

1. Connect a USB cable from the computer to the micro-B USB connector on the bottom of the starter kit in use.
2. When the demonstration runs, it will create a virtual com with USB CDC device on the USB bus.
3. Open a standard terminal application on the computer (such as HyperTerminal or Tera Term).
4. Set the baud rate to 921600 baud in the terminal application.
5. Establish a connection between the router or switch with the PIC32M Starter Kit using the RJ45 connector.

The demonstration does not offer any additional functionality through the serial port; however, the current IP can be checked. As soon as a valid IP has been assigned through DHCP to the demonstration, it is ready to serve Web pages. Use any Web browser (i.e., Chrome, Internet Explorer, Firefox, etc.) to connect to the Web server with either http:// or https://.

wolfssl_tcp_client

wolfSSL TCP Client demonstration.

Description

This configuration demonstrates creating an Internet client that uses the MPLAB Harmony TCP API to create a TCP/IP connection to a Web server. The connection can either be clear text, or it can use SSL to encrypt the connection with wolfSSL. The demonstration can use either IPv4 or IPv6.

Before using this demonstration, please see the important notes in the *TCP/IP Demonstrations* > *Introduction*.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the wolfSSL TCP Client Demonstration.

Description

To build this project, you must open the wolfssl_tcp_client.X project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is <install-dir>/apps/tcpip/wolfssl_tcp_client.

Warning

When using the Microchip Harmony Configurator (MHC), care must be taken when generating the code to *not* erase the USB descriptors in the system_init.c file.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
wolfssl_tcp_client.X	<install-dir>/apps/tcpip/wolfssl_tcp_client/firmware

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within ./firmware/src/system_config.

Project Configuration Name	BSP Used	Description
pic32mx_eth_sk	pic32mx_eth_sk	Demonstrates the wolfSSL TCP Client on the PIC32 Ethernet Starter Kit.
pic32mx_eth_sk2	pic32mx_eth_sk2	Demonstrates the wolfSSL TCP Client on the PIC32 Ethernet Starter Kit II.
pic32mz_ec_sk	pic32mz_ec_sk	Demonstrates the wolfSSL TCP Client on the PIC32MZ EC Starter Kit.
pic32mz_ef_sk	pic32mz_ef_sk	Demonstrates the wolfSSL TCP Client on the PIC32MZ EF Starter Kit.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 Ethernet Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32 Ethernet Starter Kit II

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ EC Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

This section provides instructions on how to build and run the demonstration.

Description

For PIC32M-based Starter Kits

1. Connect a USB cable from the computer to the micro-B USB connector on the bottom of the starter kit in use.
2. When the demonstration runs, it will create a virtual com with USB CDC device on the USB bus.
3. Open a standard terminal application on the computer (such as HyperTerminal or Tera Term).
4. Set the baud rate to 921600 baud in the terminal application.
5. Establish a connection between the router or switch with the PIC32M Starter Kit using the RJ45 connector.

There are three commands available in the demonstration from the serial port:

- openurl <url> - The <url> argument must be a fully formed URL; for instance, http://www.microchip.com/
- ipmode <mode> - The <mode> argument selects the IP version. 0 - Any IP version, 4 - IPv4 only, 6 - IPv6 only

- `stats` - Output the statistics of the previous `openurl` run. Statistics such as how long each phase of the connection took, and how many bytes were transferred.

After the `openurl` command is input, the demonstration will make a DNS query, and then open a connection to the requested URL and perform a simple HTTP PUT command. The results will be sent to the serial port. If a https URL is specified, the connection will first undergo SSL negotiation before sending the HTTP PUT command.

If `ipmode` is set to '`0`' (Any), the demonstration will favor IPv6 over IPv4, which means it will look for the IPv6 address before the IPv4 address.

USB Demonstrations

This section provides descriptions of the USB demonstrations.

MPLAB Harmony is available for download from the Microchip website by visiting: http://www.microchip.com/mplabharmony. Once you are on the site, click the Downloads tab to access the appropriate download for your operating system. For additional information on this demonstration, refer to the “Applications Help” section in the MPLAB Harmony Help.

Introduction

USB Library Demonstration Applications Help

Description

This distribution package contains a variety of USB-related firmware projects that demonstrate the capabilities of the MPLAB Harmony USB stack. This section describes the hardware requirement and procedures to run these firmware projects on Microchip demonstration and development boards.

To know more about the MPLAB Harmony USB stack and configuring the USB stack and the APIs provided by the USB stack, refer to the USB Library documentation.

Program, Data Memory, and Stack Component Memory

Refer to USB Device Stack Demonstration Application Program and Data Memory Requirements and USB Device Stack Component Memory Requirements for important memory information.

Pen Drive Tests

Refer to USB MSD Host USB Pen Drive Tests for information on the tests conducted on USB Flash devices.

USB Device Stack Demonstration Application Program and Data Memory Requirements

Provides information on program and data memory requirements, as well as pen drive test specifications.

Description

Program Memory and Data Memory Requirements with -O1 Optimization

The following table shows the program memory and data memory requirements of the USB Device Stack demonstration applications. All size figures are in bytes. Demonstration applications were compiled with the MPLAB XC32 C/C++ Compiler, v1.40, with –O1 optimization.

Demonstration Name		Program Memory Components			Data Memory Components	
-O1 Optimization		USB Stack	Other Drivers	System and Application	USB Stack	Others
cdc_com_port_single	PIC32MX	14048	0	5880	262	718
	PIC32MZ	18924	4328	11632	318	1410
cdc_com_port_dual	PIC32MX	13980	0	5776	262	1250
	PIC32MZ	18856	4328	10908	318	2166
cdc_serial_emulator	PIC32MX	13976	9100	5916	262	858
	PIC32MZ	N/A	N/A	N/A	N/A	N/A
cdc_serial_emulator_msd	PIC32MX	20380	12560	39540	438	1950
	PIC32MZ	NA	NA	NA	NA	NA
cdc_msd_basic	PIC32MX	20392	3460	39444	494	1810
	PIC32MZ	25264	7788	45404	494	20438
hid_basic	PIC32MX	13988	0	5440	263	601
	PIC32MZ	18764	4328	10460	319	893
hid_joystick	PIC32MX	13432	0	5332	263	485
	PIC32MZ	18208	4328	10368	319	777
hid_keyboard	PIC32MX	14016	0	6196	263	581
	PIC32MZ	18792	4328	11048	319	873
hid_mouse	PIC32MX	13460	0	5972	263	497
	PIC32MZ	18236	4328	10964	319	793
hid_msd_basic	PIC32MX	20352	3460	39172	495	1861
	PIC32MZ	25232	7788	44216	495	20445
msd_basic	PIC32MX	17848	3460	38108	492	1348
	PIC32MZ	22632	7788	43996	492	20068
vendor	PIC32MX	12560	0	5660	324	560
	PIC32MZ	17356	4328	12080	324	1748

Note: The msd_basic, cdc_msd_basic, and the hid_msd_basic demonstrations use the PIC32 program Flash memory as the MSD storage media. The difference in Data Memory requirements between the PIC32MX and PIC32MZ microcontrollers for these demonstration examples, is due to an application demonstration buffer whose size is equal to the erase page size of the PIC32 microcontroller. On the PIC32MX795F512L, this size is 4096 bytes. On the PIC32MZ2048ECH144, the erase page size is 16 KB.

Program Memory and Data Memory Requirements with -Os Optimization

The following table shows the program memory and data memory requirements of the USB Device Stack demonstration applications. All size figures are in bytes. Demonstration applications were compiled with the MPLAB XC32 C/C++ Compiler, v1.40, with –Os optimization.

Demonstration Name		Program Memory Components			Data Memory Components	
Optimization -Os		USB Stack	Other Drivers	System and Application	USB Stack	Others
cdc_com_port_single	PIC32MX	12556	0	5656	262	718
	PIC32MZ	16840	4144	11076	318	1410
cdc_com_port_dual	PIC32MX	12548	0	5620	262	1250
	PIC32MZ	16832	4144	10448	318	2166
cdc_serial_emulator	PIC32MX	12504	7744	5880	262	858
	PIC32MZ	N/A	N/A	N/A	N/A	N/A
cdc_serial_emulator_msd	PIC32MX	20380	12560	35080	438	1950
	PIC32MZ	N/A	N/A	N/A	N/A	N/A
cdc_msd_basic	PIC32MX	17896	2884	39232	494	1810
	PIC32MZ	22172	7012	44868	494	20438
hid_basic	PIC32MX	12656	0	5344	263	601
	PIC32MZ	16784	4144	10060	319	893
hid_joystick	PIC32MX	12144	0	5304	263	485
	PIC32MZ	16272	4144	10040	319	777
hid_keyboard	PIC32MX	12664	0	6060	263	581
	PIC32MZ	16792	4144	10604	319	873
hid_mouse	PIC32MX	12152	0	5820	263	497
	PIC32MZ	16280	4144	10512	319	793
hid_msd_basic	PIC32MX	18060	2884	39068	495	1861
	PIC32MZ	20352	7012	45788	495	20445
msd_basic	PIC32MX	15776	2884	38044	492	1348
	PIC32MZ	19090	7012	44426	492	20068
vendor	PIC32MX	11452	0	5540	324	560
	PIC32MZ	15584	4144	10948	324	1748

USB Device Stack Component Memory Requirements

Provides memory requirements.

Description

The following table shows the Program and Data Memory requirements for individual components in the MPLAB Harmony USB Device Stack.

Device Stack Component	Program Memory	Data Memory
Device Layer	5688	184
CDC Function Driver	2420	64 + (36 * Queue Size)
MSD Function Driver	5352	217
HID Function Driver	2376	40 + (36 * Queue Size)
Vendor	912	8 + (36 * Queue Size)
PIC32MX USB Driver	5636	144 + (32 * Number of Endpoints)
PIC32MZ USB Driver	10244	192 + (32 * Number of Endpoints)

Notes:

1. Memory requirements (in bytes) for a single instance.
2. Size measured for USB Device Stack Components in MPLAB Harmony.
3. Data Memory does not include function call stack memory size.

USB MSD Host USB Pen Drive Tests

Provides pen drive test specifications.

Description

USB MSD Host USB Pen Drive Tests

The following table lists the commercially available USB pen drives, which have been tested to successfully enumerate with the MSD Host Driver

in the MPLAB Harmony USB Host. Note that if the USB pen drive you are using in not included in the table, this indicates that this USB pen drive has not been tested with the MSD Host Driver. However, the USB pen drive could still potentially work with MSD Host Driver. Some USB pen drives in this table did not have their manufacturer or model data available. The USB Pen drives were tested with the msd_basic USB Host demonstration in the latest version of the MPLAB Harmony USB Host Stack.

VID	PID	Manufacturer	Model/Drive Capacity
0x1B1C	0x1A0F	Corsair Components	Flash Voyager Go 8 GB
0x03F0	0x0AB7	Hewlett-Packard	64 GB
0xABCD	0x1234	Microchip Technology Inc.	4 GB
0x125F	0xCB10	Adata	Dashdrive UV100 8 GB
0x8644	0x8003	Verico	T Series 16 GB
0x8564	0x1000	Transcend	USB 3.0 32 GB
0x0951	0x16A7	Dell	Kingston Technology 16 GB
0x0718	0x0704	Imation	16 GB Pen Drive
0x048D	0x1168	iBall	Jaldi 16 GB Pen Drive
0x058F	0x6366	Alcor	Micro AXL 32 GB
0x154B	0x005B	PNY	Cube 16 GB
0x0930	0x6544	Toshiba	Hatabusa Pen Drive 8 GB
0x058F	0x6387	Alcor	ZipMem 16 GB
0x090C	0x1000	Silicon Motion Inc.	Axl 8GB
0x18A5	0x0245	Verbatim	Store N Go Audio USB 8 GB
0x05DC	0xC75C	Lexar	USB Pen Drive 8 GB
0x1005	0xb113	Apacer	8 GB (AH233)
0x054C	0x06B0	Sony	8 GB
0x054C	0x0862	Sony	Micro Vault USM-V 8 GB
0x0781	0x557c	SanDisk	8 GB
0x1E4E	0x3257	Etron	iBall 16 GB
0x1EC9	0x0101	Moserbaer	Swivel 16 GB Pen Drive
0x0BDA	0x0109	SanDisk	Standard A and Mini-B connector 16 GB
0x1908	0x1320	ZBEL	Wrist Band Flash Drive 4 GB
0x0951	0x1665	Kingston	Data Traveler SE9 16 GB

USB HID Host Keyboard and Mouse Tests

Provides information on tested USB keyboard and mouse devices.

Description

The following table lists the commercially available USB keyboard and mouse devices, which have been tested to successfully enumerate with the HID Host Driver in the MPLAB Harmony USB Host. Note that if the USB HID device you are using in not included in the table, this indicates that this USB HID device has not been tested, but could still potentially work with the HID Host Driver.

Type	Manufacturer	VID	PID	Device Details
Keyboard	Microsoft	0x045E	0x07B9	Microsoft wired 200.
	ProHT	0x1A2C	0x0C21	110 keys.
	HP	0x04D9	0x1702	K1500 standard keyboard.
	Zebronics	0x1A2C	0x0027	Standard keyboard.
	Logitech	0x046D	0xC31D	112 keys.
	Gear Head	0x04D9	0x2BA0	82 key, mini-USB keyboard plus integrated mouse.
	Intex	0x1C4F	0x0002	122 keys.
Mouse	HP	0x0461	0x4D0F	Three button mouse.
	Intex	0x1BCF	0x0007	Three button mouse.
	Anker	0x1BCF	0x0005	Five button mouse.
	TeraByte	0x10C4	0x8103	Three button mouse.
	Kensington	0x1BCF	0x0002	Five button mouse.
	Logitech	0x046D	0xC069	Five button mouse.
	Microsoft	0x045E	0x0797	Optical mouse 200, three buttons.
	Logitech	0x046D	0xC246	Nine button mouse.
	HP	0x03F0	0x0941	Three button mouse.
	Lenovo	0x1BCF	0x000A	Five button mouse.

Note: The above tests have been performed only on the PIC32M family of devices.

Demonstration Application Configurations

This topic provides information on the available USB demonstration project configurations.

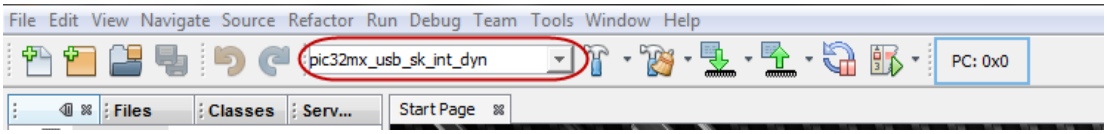
Description

The available USB Demonstration application MPLAB X IDE projects feature support for multiple configurations. Selecting these configurations allow for the demonstration projects to run across different PIC32 microcontrollers and development boards. The following project configurations are available:

Configuration name	Description
pic32mx_usb_sk2_int_dyn	Selecting this configuration will set up the demonstration application to run on the PIC32 USB Starter Kit II development board, with the PIC32MX795F512L microcontroller. The USB Stack will be configured for Interrupt mode operation and the USB Driver will be configured for Dynamic operation mode.
pic32mx_usb_sk2_poll_dyn	Selecting this configuration will set up the demonstration application to run on the PIC32 USB Starter Kit II development board, with the PIC32MX795F512L microcontroller. The USB Stack will be configured for Polled mode operation and the USB driver will be configured for Dynamic operation mode.
pic32mx_usb_sk3_int_dyn	Selecting this configuration will set up the demonstration application to run on the PIC32 USB Starter Kit III development board, with the PIC32MX470F512L microcontroller. The USB Stack will be configured for Interrupt mode operation and the USB Driver will be configured for Dynamic operation mode.
pic32mx_bt_sk_int_dyn	Selecting this configuration will set up the demonstration application to run on the PIC32 Bluetooth Starter Kit development board, with the PIC32MX270F256D microcontroller. The USB Stack will be configured for Interrupt mode operation and the USB Driver will be configured for dynamic operation mode.
pic32mz_da_sk_intddr_int_dyn	Selecting this configuration will set up the demonstration application to run on the PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit development board, with the PIC32MZ2064DAH169 microcontroller. The USB Stack will be configured for Interrupt mode operation and the USB Driver will be configured for Dynamic operation mode.
pic32mz_ec_sk_int_dyn	Selecting this configuration will set up the demonstration application to run on the PIC32MZ EC Starter Kit development board, with the PIC32MZ2048ECH144 microcontroller. The USB Stack will be configured for Interrupt mode operation and the USB Driver will be configured for Dynamic operation mode.
pic32mz_ec_sk_poll_dyn	Selecting this configuration will set up the demonstration application to run on the PIC32MZ EC Starter Kit development board, with the PIC32MC2048ECH144 microcontroller. The USB Stack will be configured for Polled mode operation and the USB Driver will be configured for Dynamic operation mode.
pic32mz_ec_sk_meb2_int_dyn	Selecting this configuration will set up the demonstration application to run on the PIC32MZ EC Starter Kit, with the PIC32MZ2048ECH144 microcontroller board attached to the MEB II. The USB Stack will be configured for Interrupt mode operation and the USB Driver will be configured for Dynamic operation mode.
pic32mz_ef_sk_int_dyn	Selecting this configuration will set up the demonstration application to run on the PIC32MZ EF Starter Kit, with the PIC32MZ2048EFM144 microcontroller. The USB Stack will be configured for Interrupt mode operation and the USB Driver will be configured for Dynamic operation mode.

pic32mz_ef_sk_poll_dyn	Selecting this configuration will set up the demonstration application to run on the PIC32MZ EF Starter Kit development board, with the PIC32MZ2048EFM144 microcontroller. The USB Stack will be configured for Polled mode operation and the USB Driver will be configured for Dynamic operation mode.
pic32mx795_pim_e16_int_dyn	Selecting this configuration will set up the demonstration application to run on the Explorer 16 Development Board along with the PIC32MX795F512L microcontroller Plug In Module and USB PICtail Plus Daughter Board. The USB Stack will be configured for Interrupt mode operation and the USB Driver will be configured for Dynamic operation mode.
pic32mx460_pim_e16_int_dyn	Selecting this configuration will set up the demonstration application to run on the Explorer 16 Development Board along with the PIC32MX460F512L microcontroller Plug In Module and USB PICtail Plus Daughter Board. The USB Stack will be configured for Interrupt mode operation and the USB Driver will be configured for Dynamic operation mode.
pic32mx470_curiosity	Selecting this configuration will set up the demonstration application to run on the PIC32MX470 Curiosity Development Board, with the PIC32MX470F512H microcontroller. The USB Stack will be configured for Interrupt mode operation and the USB Driver will be configured for Dynamic operation mode.
pic32mz_ef_curiosity	Selecting this configuration will set up the demonstration application to run on the PIC32MZ EF Curiosity Development Board, with the PIC32MZ2048EFM100 microcontroller. The USB Stack will be configured for Interrupt mode operation and the USB Driver will be configured for Dynamic operation mode.
pic32mk_evk_int_dyn	Selecting this configuration will set up the demonstration application to run on the PIC32MK GP Development Board, with the PIC32MK1024GPE100 microcontroller. The USB Stack will be configured for Interrupt mode operation and the USB Driver will be configured for Dynamic operation mode.
pic32mx_xlp_sk_int_dyn	Selecting this configuration will set up the demonstration application to run on the PIC32MX XLP Starter Kit, with the PIC32MX274F256D microcontroller. The USB Stack will be configured for Interrupt mode operation and the USB Driver will be configured for Dynamic operation mode.
chipkit_wf32	Selecting this configuration will set up the demonstration application to run on the chipKIT WF32 Wi-Fi Development Board, with the PIC32MZ2048EFG100 microcontroller. The USB Stack will be configured for Interrupt mode operation and the USB Driver will be configured for Dynamic operation mode.
chipkit_wifire	Selecting this configuration will set up the demonstration application to run on the chipKIT Wi-FIRE Development Board, with the PIC32MX275F256D microcontroller. The USB Stack will be configured for Interrupt mode operation and the USB Driver will be configured for Dynamic operation mode.

The following figure shows how a configuration can be selected in MPLAB X IDE.



Alternatively, the active configuration can be selected in the Project Properties.

USB Device Demonstrations Matrix

The following table shows the availability of a configuration across available USB Device demonstration applications. **Green** indicates support. Red indicates no support.

Demo Apps \ Configurations	pic32mx_usb_sk2_int_dyn	pic32mz_ec_sk_int_dyn	pic32mx795_pim_e16_int_dyn	pic32mx_bt_sk_int_dyn	pic32mx_usb_sk2_poll_dyn	pic32mx460_pim_e16_int_dyn	pic32mx_usb_sk2_int_sta	pic32mx_usb_sk3_int_dyn	pic32mz_bt_audio_int_dyn	pic32mz_ef_sk_poll_dyn	pic32mz_ef_sk_int_dyn	pic32mx_125_sk_int_dyn	chipkit_wf32	chipkit_wifire	pic32mx470_curiosity	pic32mz_ef_curiosity	pic32mx_xlp_sk_int_dyn	pic32mz_ef_sk_int_dyn_micromips	pic32mz_da_sk_int_dyn
cdc_com_port_single																			
cdc_com_port_dual																			
cdc_serial_emulator																			
hid_basic																			
hid_mouse																			
hid_keyboard																			
hid_joystick																			
msd_basic																			
vendor																			
hid_msd_basic																			
cdc_serial_emulator_msd																			
cdc_msd_basic																			
msd_sdcard																			
msd_fs_spiflash																			
msd_multiple_luns																			

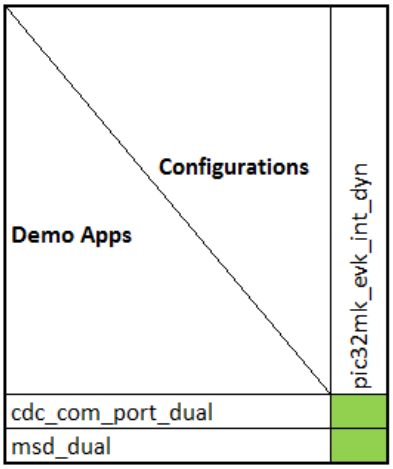
USB Host Demonstration Matrix

The following table shows the availability of a configuration across available USB Host demonstration applications. Green indicates support. Red indicates no support.

Demo Apps \ Configurations	pic32mx_usb_sk2_int_dyn	pic32mz_ec_sk_int_dyn	pic32mz_ec_sk_meb2_int_dyn	pic32mz_ef_sk_int_dyn	pic32mz_ef_sk_meb2_int_dyn	pic32mx795_pim_e16_int_dyn	chipkit_wf32	chipkit_wifire	pic32mz_ef_curiosity	pic32mz_ef_curiosity	pic32mx_xlp_sk_int_dyn
cdc_basic											
msd_basic											
cdc_msd											
hid_basic_mouse											
audio_speaker											
hub_msd											
hub_cdc_hid											
hid_basic_keyboard											

USB Multiple Controller Demonstration Matrix

The following table shows the availability of a configuration across available USB Multiple Controller Demonstration applications. Green indicates support. Red indicates no support.



Demonstrations

The USB Demonstrations are grouped into USB Device Stack, USB Host Stack, USB Dual Role, and USB demonstrations that make use of multiple USB controllers on certain PIC32 family devices.

Device

This section describes the USB Device demonstrations.

Description

The MPLAB Harmony USB Device Stack demonstration applications uses LEDs on the development board to indicate the USB state of the device. The following table provides details on the development board specific LEDs and the USB Device State these indicate when active. This indication scheme is implemented by all USB Device Stack Demonstration applications.

USB Device State and LED Indication

Demonstration Board	Reset State	Configured State	Suspended State
Explorer 16 Development Board and PIM	D3, D4	D5	D4, D5
PIC32 USB Starter Kit II	LED1, LED2	LED3	LED2, LED3
PIC32MZ Embedded Connectivity (EC) Starter Kit	LED1, LED2	LED3	LED2, LED3
PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit	LED1, LED2	LED3	LED2, LED3
PIC32 USB Starter Kit III	LED1, LED2	LED3	LED2, LED3
PIC32 Bluetooth Starter Kit	Red LED, Green LED	Blue LED	Green LED, Blue LED
PIC32MX470 Curiosity Development Board	LED1, LED2	LED3	LED2, LED3
PIC32MZ EF Curiosity Development Board	LED1, LED2	LED3	LED2, LED3

cdc_com_port_dual

Demonstrates a USB CDC device, emulating dual serial COM ports - one looping back into the other.

Description

This demonstration application creates a USB CDC Device that enumerates as two serial ports on the USB Host personal computer. This application demonstrates the ability of the MPLAB Harmony USB Device Stack to support multiple instances of the same Device class.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the USB CDC Device Dual COM Port Demonstration.

Description

To build this project, you must open the `cdc_com_port_dual.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/usb/device/cdc_com_port_dual`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`cdc_com_port_dual.X`	`<install-dir>/apps/usb/device/cdc_com_port_dual/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx460_pim_e16_int_dyn	pic32mx460_pim+e16	Select this MPLAB X IDE project configuration to run the demonstration on the Explorer 16 Development Board configured for Interrupt mode and dynamic operation. This configuration also requires PIC32MX460F512L Plug-In Module (PIM) and the USB PICtail Plus Daughter Board.
pic32mx_bt_sk_int_dyn	pic32mx_bt_sk	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32 Bluetooth Starter Kit configured for Interrupt mode and dynamic operation.
pic32mx_usb_sk3_int_dyn	pic32mx_usb_sk3	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32 USB Starter Kit III configured for Interrupt mode and dynamic operation.
pic32mx_usb_sk2_int_dyn	pic32mx_usb_sk2	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32 USB Starter Kit II configured for Interrupt mode and dynamic operation.
pic32mx_xlp_sk_int_dyn	pic32mx_xlp_sk	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32MX XLP Starter Kit configured for Interrupt mode and dynamic operation.
pic32mz_ef_sk_int_dyn	pic32mz_ef_sk	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit configured for Interrupt mode and dynamic operation.
pic32mx470_curiosity	pic32mx470_curiosity	Select this MPLAB X IDE project configuration to run the demonstration application to run on the PIC32MX470 Curiosity Development Board, with the PIC32MX470F512H microcontroller. The USB Stack will be configured for Interrupt mode operation and the USB Driver will be configured for Dynamic operation mode.
pic32mz_ef_curiosity	pic32mz_ef_curiosity	Select this MPLAB X IDE project configuration to run the demonstration application to run on the PIC32MZ EF Curiosity Development Board, with the PIC32MZ2048EFM100 microcontroller. The USB Stack will be configured for Interrupt mode operation and the USB Driver will be configured for Dynamic operation mode.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 USB Starter Kit II

Remove jumper JP2.

PIC32MZ EC Starter Kit

Remove jumper JP1.

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32MX XLP Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32 USB Starter Kit III

Remove jumper JP1.

PIC32 Bluetooth Starter Kit

Jumper J8 should either be shorted between pins 2 and 3 or should be completely open.

PIC32MX460F512L PIM

Jumper J10 should be removed. This plug-in module should be used along with the Explorer 16 Development Board and the USB PICtail Plus daughter board. The microcontroller PIM should be plugged into the PIM socket on the board. The USB PICtail Plus daughter board should be connected to the edge connector J9.

On the Explorer 16 Development Board:

- Switch S2 should be set to PIM
- Jumper JP2 should be in place

On the USB PICtail Plus Daughter Board:

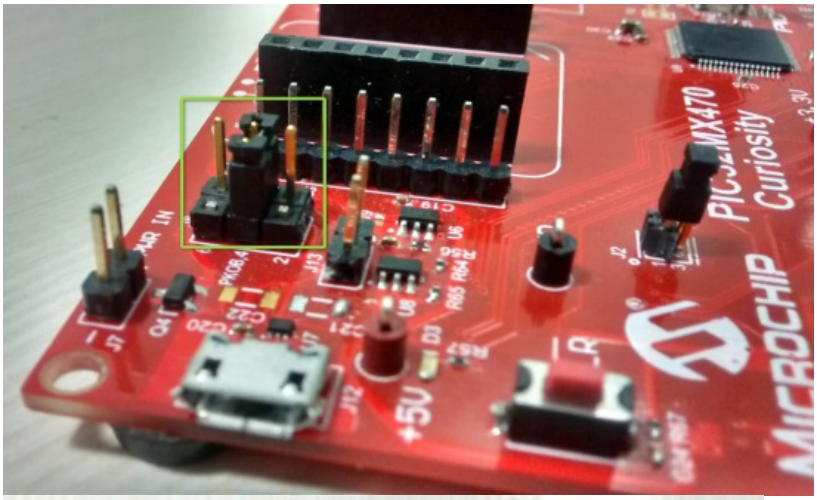
- Jumper JP1 should be in place
- Jumper JP2 and JP4 should be removed

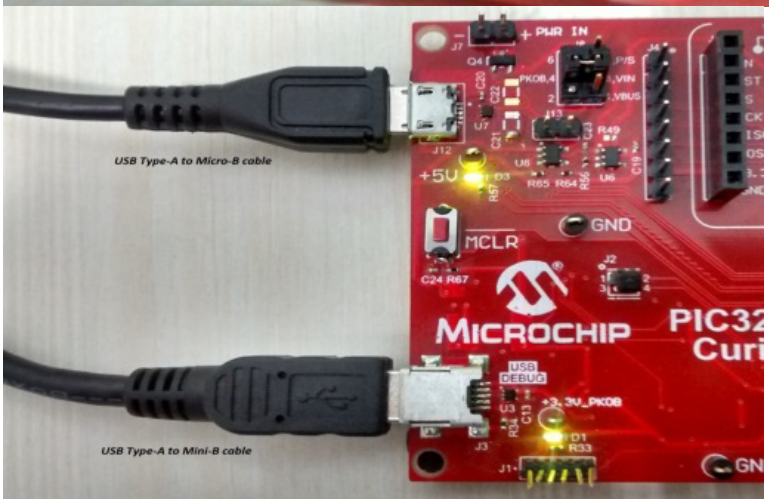
On the PIC32MX460F512L PIM:

- Keep jumper J10 open
- Keep all jumpers in J9 open

PIC32MX470 Curiosity Development Board

- Ensure that a jumper is placed at 4-3 on J8, to select supply from debug USB connector.
- Power the PIC32MX470 Curiosity Development Board from a Host PC through a Type-A male to mini-B USB cable connected to Mini-B port (J3).
- Ensure that jumper is not present in the J13 header to use the Curiosity board in device mode.
- Plug in a USB cable with a micro-B type connector to Micro-B port (J12), and plug the other end into your computer.

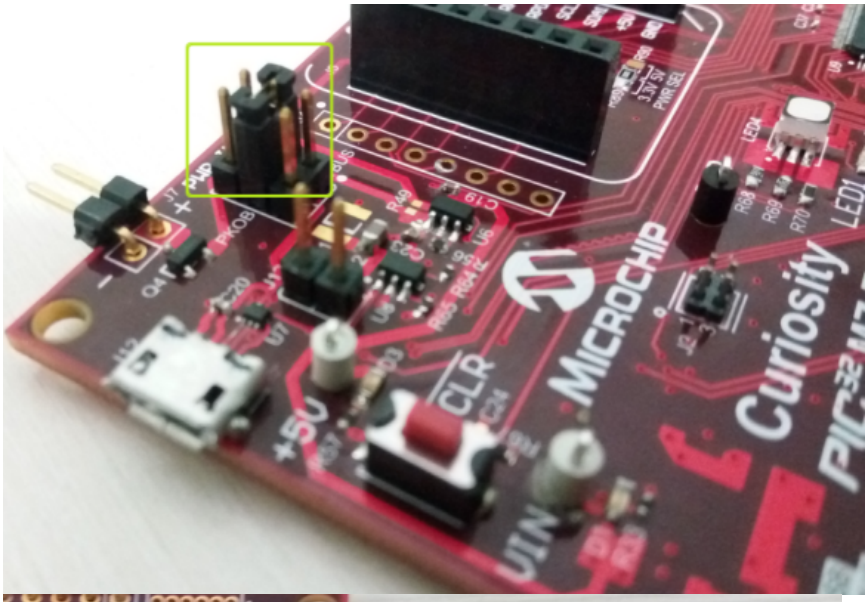


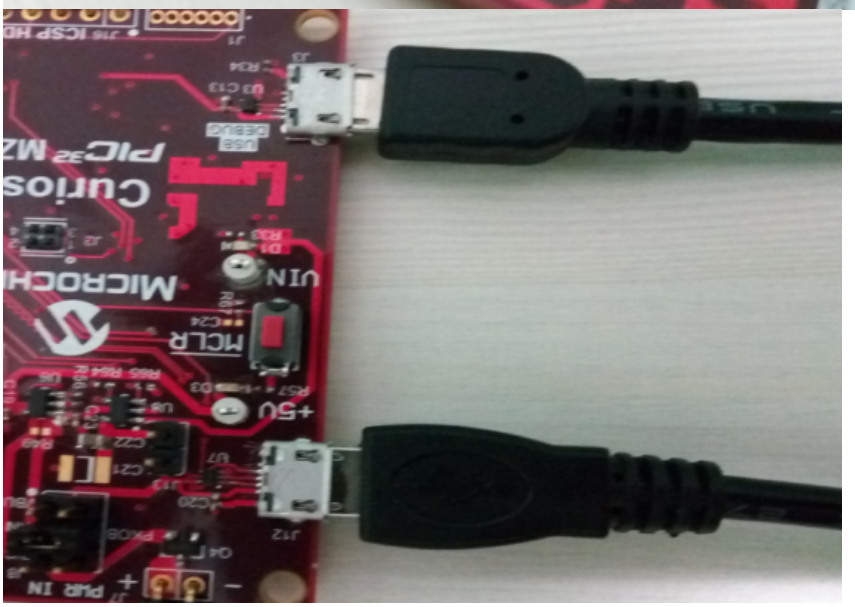


PIC32MZ EF Curiosity Development Board

- Ensure that a jumper is placed at 4-3 on J8, to select supply from debug USB connector.
- Power the PIC32MZ EF Curiosity Development Board from a Host PC through a Type-A male to micro-B USB cable connected to Micro-B port (J3).

- Ensure that jumper is not present in the J13 header to use the Curiosity board in device mode.
- Plug in a USB cable with a micro-B type connector to Micro-B port (J12), and plug the other end into your computer.





Running the Demonstration

Provides instructions on how to build and run the CDC Dual COM Port demonstration.

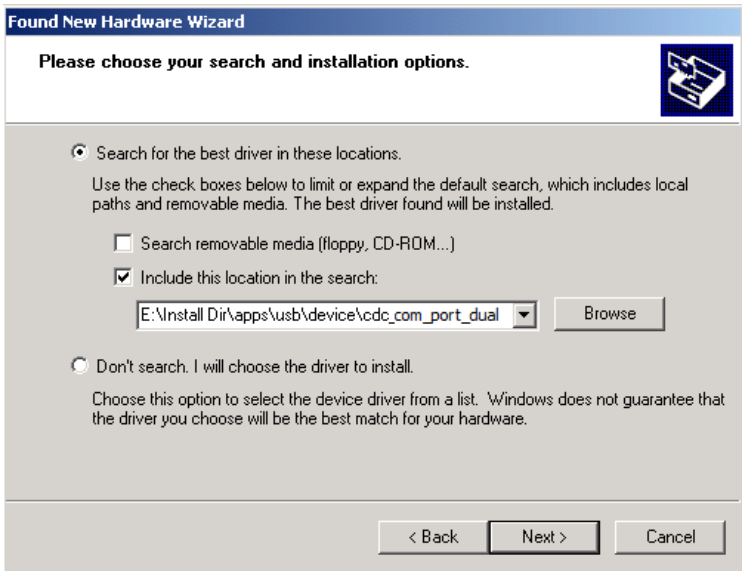
Description

This demonstration allows the device to appear like dual serial (COM) ports to the host. Do the following to run this demonstration:

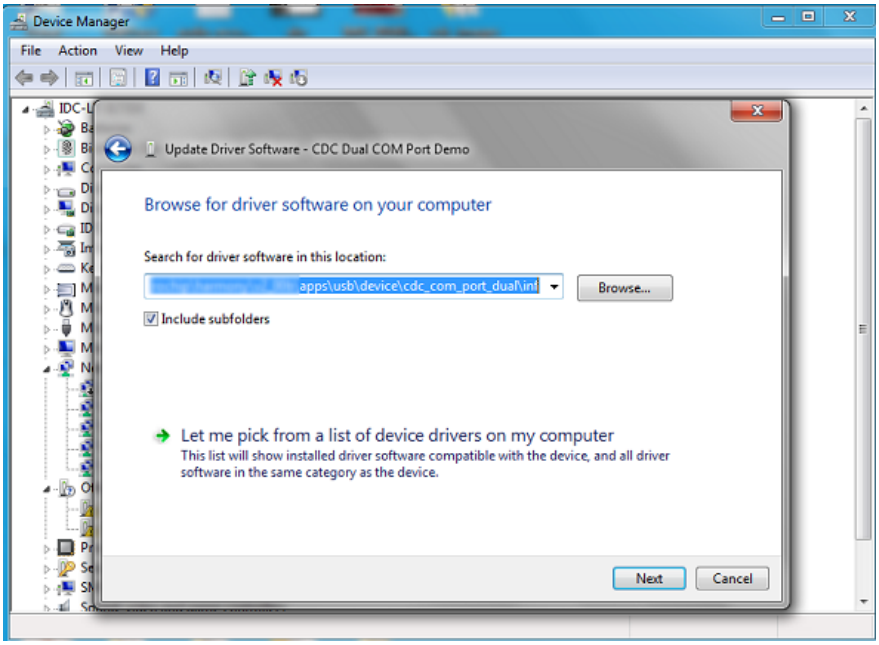
1. First compile and program the target device. While compiling, select the appropriate MPLAB X IDE project configuration based on the demonstration board. Refer to Building the Application for details.
2. Attach the device to the host. If the host is a personal computer and this is the first time you have plugged this device into the computer you may be prompted for a `.inf` file.



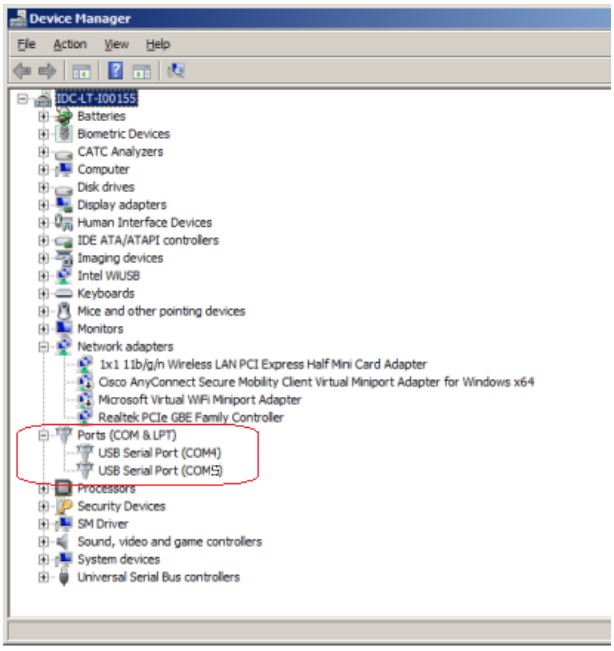
3. Select the "Install from a list or specific location (Advanced)" option. Specify the `<install-dir>/apps/usb/device/cdc_com_port_dual/inf` directory.



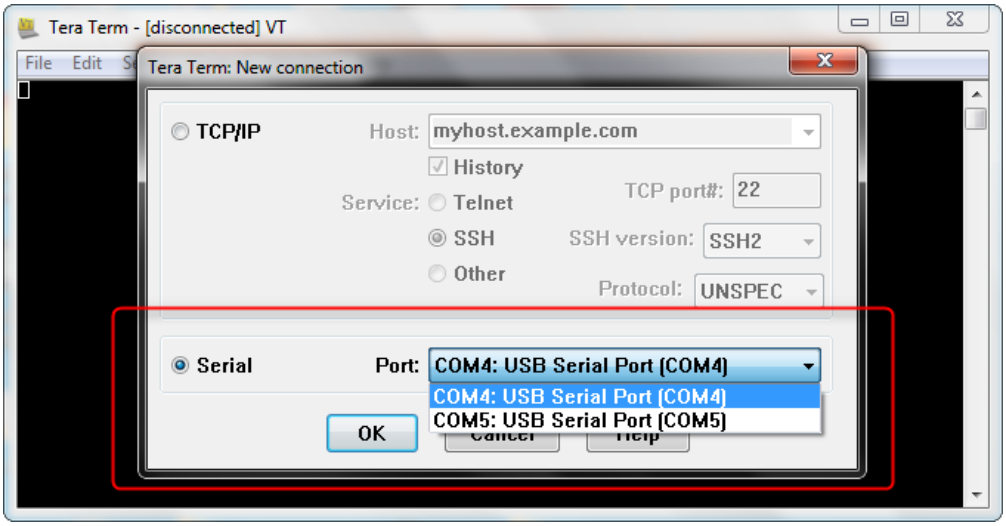
Note: As an option, to specify the driver, you may open the device manager and expand the Ports (COM & LPT) tab, and right click on "Update Driver Software…"



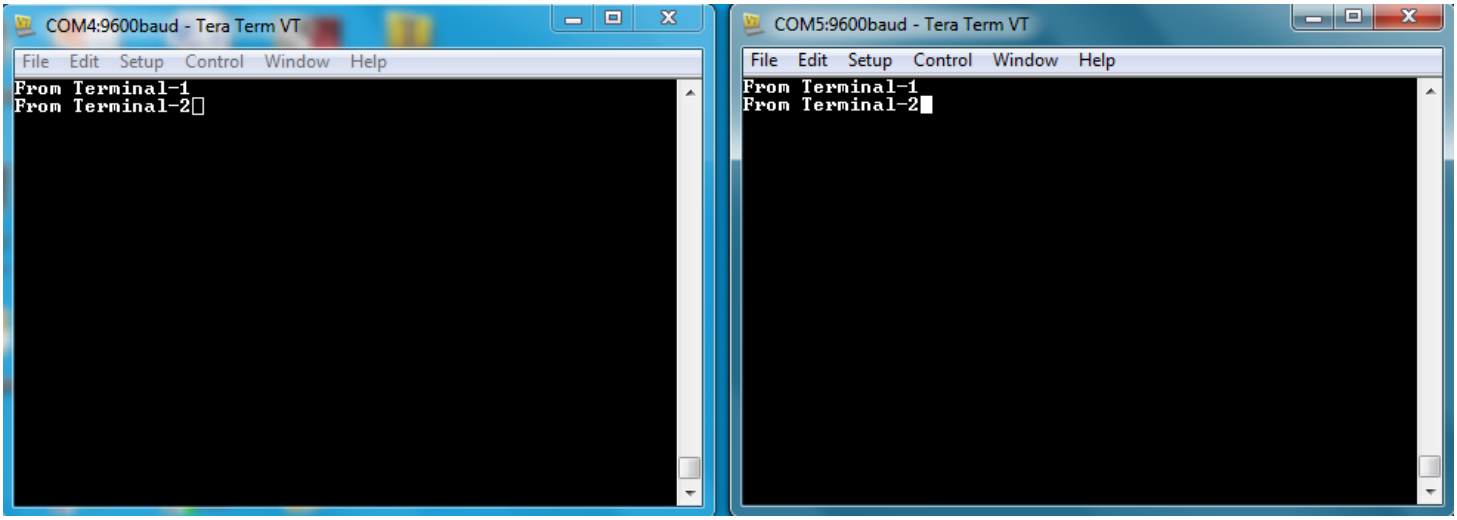
Verify that the enumerated USB device is seen as a virtual USB serial comport in Device Manager.



4. Once the device is successfully installed, open up two instances of a terminal program, such as HyperTerminal. Select the appropriate COM port for each of these terminal instances. The following screen shot shows the COM port selection for the Tera Term terminal program.



5. The LEDs on the demonstration board will indicate the USB state of the device, as described in the USB Device State and LED Indication Table in the Device section.
6. To run the demonstration, turn on local echo on both the terminals. For Tera Term terminal application, navigate to Setup->Terminal to turn on local echo. Type a character or string in one terminal window. The same character or string appears on the second terminal window. Similarly, any character typed in the second window appears in the first window. The following screen shot shows two instances of Tera Term.



Note: Some terminal programs, like HyperTerminal, require users to click the disconnect button before removing the device from the computer. Failing to do so may result in having to close and open the program again to reconnect to the device.

cdc_com_port_single

Demonstrates a USB CDC device, emulating a serial COM port.

Description

This demonstration application creates a USB CDC Device that enumerates as a single COM port on the host personal computer. The application demonstrates two-way communication between the USB device and the personal computer host.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the USB CDC Device Single COM Port Demonstration.

Description

To build this project, you must open the `cdc_com_port_single.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/usb/device/cdc_com_port_single`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`cdc_com_port_single.X`	`<install-dir>/apps/usb/device/cdc_com_port_single/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx460_pim_e16_int_dyn	pic32mx460_pim+e16	Select this MPLAB X IDE project configuration to run the demonstration on the Explorer 16 Development Board configured for Interrupt mode and dynamic operation. This configuration also requires PIC32MX460F512L Plug-In Module (PIM) and the USB PICtail Plus Daughter Board.
pic32mx_usb_sk2_poll_dyn	pic32mx_bt_sk	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32 USB Starter Kit II with the USB Device Stack configured for Polled mode and dynamic operation.
pic32mx_usb_sk3_int_dyn	pic32mx_usb_sk3	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32 USB Starter Kit III configured for Interrupt mode and dynamic operation.
pic32mx_usb_sk2_int_dyn	pic32mx_usb_sk2	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32 USB Starter Kit II configured for Interrupt mode and dynamic operation.
pic32mz_da_sk_intddr_int_dyn	pic32mz_da_sk_intddr	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit configured for Interrupt mode and dynamic operation.
pic32mz_ef_sk_int_dyn_micromips	pic32mz_ef_sk	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit configured in microMIPS mode for Interrupt mode and dynamic operation.
pic32mz_ef_sk_int_dyn	pic32mz_ef_sk	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit configured for Interrupt mode and dynamic operation.
pic32mz_ef_sk_poll_dyn	pic32mz_ef_sk	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit with the USB Device Stack configured for Polled mode and dynamic operation.
pic32mx_125_sk_int_dyn	pic32mx_125_sk	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32MX1/2/5 Starter Kit with the USB Device Stack configured for Interrupt mode and dynamic operation.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 USB Starter Kit II

Remove jumper JP2.

PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32 USB Starter Kit III

Remove jumper JP1.

PIC32MX460F512L PIM

Jumper J10 should be removed. This plug-in module should be used along with the Explorer 16 Development Board and the USB PICtail Plus daughter board. The microcontroller PIM should be plugged into the PIM socket on the board. The USB PICtail Plus daughter board should be connected to the edge connector J9.

On the Explorer 16 Development Board:

- Switch S2 should be set to PIM
- Jumper JP2 should be in place

On the USB PICtail Plus Daughter Board:

- Jumper JP1 should be in place
- Jumper JP2 and JP4 should be removed

On the PIC32MX460F512L PIM:

- Keep jumper J10 open
- Keep all jumpers in J9 open

PIC32WK Wi-Fi Starter Kit

No hardware related configuration or jumper setting changes are necessary.

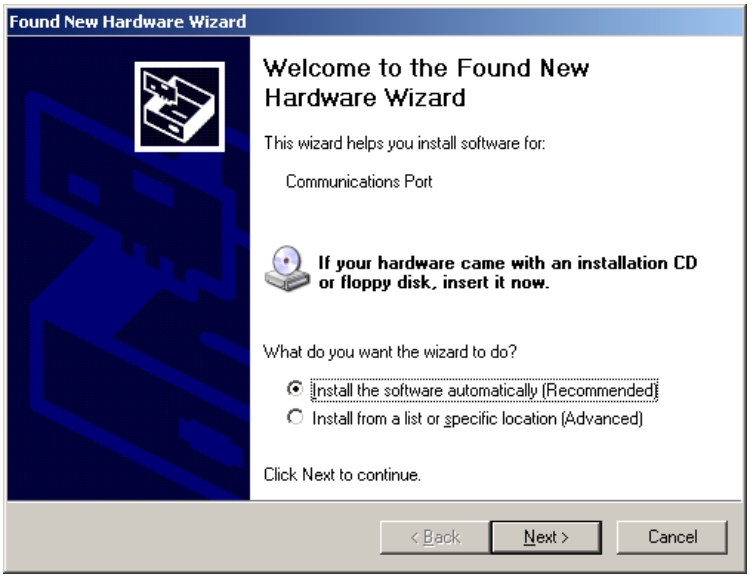
Running the Demonstration

Provides instructions on how to build and run the CDC Single COM Port demonstration.

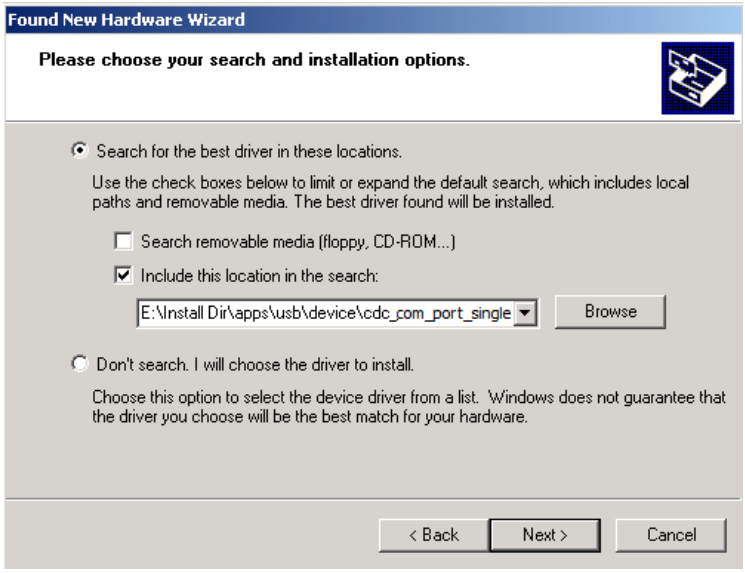
Description

This demonstration allows the device to appear like a serial (COM) port to the host. Do the following to run this demonstration:

1. First compile and program the target device. While compiling, select the appropriate MPLAB X IDE project configuration based on the demonstration board. Refer to Building the Application for details.
2. Attach the device to the host. If the host is a personal computer and this is the first time you have plugged this device into the computer, you may be prompted for a `.inf` file.



3. Select the "Install from a list or specific location (Advanced)" option. Specify the `<install-dir>/apps/usb/device/cdc_com_port_single/inf` directory.



4. Once the device is successfully installed, open up a terminal program, such as HyperTerminal and select the appropriate COM port. On most machines this will be COM5 or higher. Set the communication properties to 9600 baud, 1 Stop bit and No parity, with Flow Control set to None.
5. The LEDs on the demonstration board will indicate the USB state of the device, as described in the USB Device State and LED Indication Table in the Device section.
6. Once connected to the device, there are two ways to run this example project:
   - a) Typing a key in the terminal window will result in the attached device echoing the next letter. Therefore, if the letter 'b' is pressed, the device will echo 'c'.
   - b) If the push button is pressed, the device will echo "PUSH BUTTON PRESSED" to the terminal window.

The following table shows the switch buttons to be pressed for different demonstration boards.

Demonstration Board	Button
PIC32 USB Starter Kit II PIC32 USB Starter Kit III PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit PIC32MZ Embedded Connectivity (EC) Starter Kit PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit PIC32WK Wi-Fi Starter Kit	SW1
Explorer 16 Development Board	S3

Note: Some terminal programs, like HyperTerminal, require users to click the disconnect button before removing the device from the computer. Failing to do so may result in having to close and open the program again to reconnect to the device.

cdc_msd_basic

Demonstrates a composite USB device emulating a COM port and Flash drive.

Description

This demonstration application creates a composite USB Device that enumerates as a COM port and as Flash drive simultaneously.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the USB CDC MSD Composite Device Demonstration.

Description

To build this project, you must open the `cdc_msd_basic.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/usb/device/cdc_msd_basic`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
cdc_msd_basic.X	<install-dir>/apps/usb/device/cdc_msd_basic/firmware

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within ./firmware/src/system_config.

Project Configuration Name	BSP Used	Description
pic32mx_usb_sk2_int_dyn	pic32mx_usb_sk2	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32 USB Starter Kit II configured for Interrupt mode and dynamic operation.
pic32mz_ef_sk_int_dyn	pic32mz_ef_sk	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit configured for Interrupt mode and dynamic operation.

Configuring the Hardware

Description

PIC32 USB Starter Kit II

Remove jumper JP2.

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the USB CDC MSD Composite Device demonstration.

Description

This demonstration application creates a composite USB Device that works simultaneously as a CDC and as a MSD device. This application combines the functionality of the cdc_com_port_single and msd_basic demonstration applications into one device.

Refer to Running the Demonstration section of the cdc_com_port_single demonstration and the Running the Demonstration section of the msd_basic demonstration for details on exercising the CDC and MSD device features, respectively.

The LEDs on the demonstration board will indicate the USB state of the device, as described in the USB Device State and LED Indication Table in the Device section.

cdc_serial_emulator

This application demonstrates the use of the CDC device class in implementing a USB-to-Serial Dongle.

Description

This application demonstrates the use of the CDC device class in implementing a USB-to-Serial Dongle. The application enumerates a COM port on the personal computer. Data received through the CDC USB interface is forwarded to a UART. Data received on the UART is forwarded to the CDC USB interface. This emulates a USB-to-Serial Dongle.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the USB CDC Device USB-to-Serial Demonstration.

Description

To build this project, you must open the cdc_serial_emulator.X project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is <install-dir>/apps/usb/device/cdc_serial_emulator.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
cdc_serial_emulator.X	<install-dir>/apps/usb/device/cdc_serial_emulator/firmware

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within ./firmware/src/system_config.

Project Configuration Name	BSP Used	Description
pic32mx795_pim_e16_int_dyn	pic32mx795_pim+e16	Select this MPLAB X IDE project configuration to run the demonstration on the Explorer 16 Development Board configured for Interrupt mode and dynamic operation. This configuration also requires the PIC32MX795F512L Plug-In Module (PIM) and the USB PICtail Plus Daughter Board.
pic32mz_ef_sk_int_dyn	pic32mz_ef_sk	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32MZ EF Starter Kit configured for Interrupt mode and dynamic operation.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32MX795F512L CAN-USB PIM

Jumper J10 should be removed. Jumper J1 and J2 should connect to positions 1 and 2. This PIM should be used along with the Explorer 16 Development Board and the USB PICtail Plus daughter board. The microcontroller PIM should be plugged into the PIM socket on the board. The USB PICtail Plus daughter board should be connected to the edge connector J9.

On the Explorer 16 Development Board:

- Switch S2 should be set to PIM
- Jumper JP2 should be in place

On the USB PICtail Plus Daughter Board:

- Jumper JP1 should be in place
- Jumper JP2 and JP4 should be removed

On the PIC32MX795F512L PIM:

- Keep jumper J10 open
- Keep all jumpers in J9 open
- Jumper J1 should be shorted between positions 1 and 2. This configuration is only applicable for the PIC32MX795F512L USB CAN PIM (MA320003), and not the PIC32MX795F512L USB PIM (MA320002).
- Jumper J2 should be shorted between positions 1 and 2. This configuration is only applicable for the PIC32MX795F512L USB CAN PIM (MA320003) and not the PIC32MX795F512L USB PIM (MA320002).

Running the Demonstration

Provides instructions on how to build and run the CDC Serial Emulator Demonstration.

Prior to using this demonstration, it is recommended to review the MPLAB Harmony Release Notes for any known issues. A PDF copy of the release notes is provided in the <install-dir>/doc folder of your installation.

Description

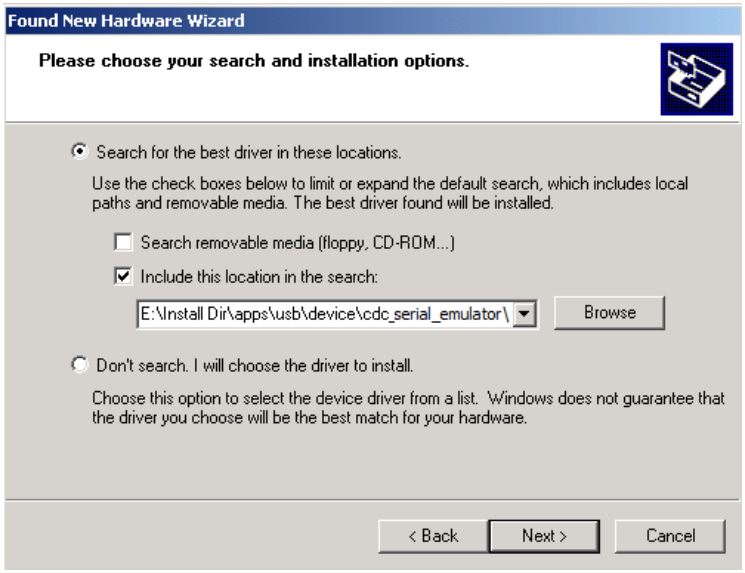
This application demonstrates the use of the CDC Device class in implementing a USB-to-Serial Dongle. The application enumerates a COM port on the personal computer. Data received through the CDC USB interface is forwarded to a UART. Data received on the UART is forwarded to the CDC USB interface. This emulates a USB-to-Serial Dongle.

1. Open the project in MPLAB X IDE and select the desired configuration.
2. Build the code and program the device.
3. Depending on the hardware in use, do one of the following:

- If you are using the Explorer 16 board, connect the mini-B device connector on the USB PICtail Plus Daughter Board to the personal computer
- If you a are using the PIC32MZ EF starter kit, connect the micro-USB device connector to the personal computer



7. Select the "Install from a list or specific location (Advanced)" option. Specify the `<install-dir>/apps/usb/device/cdc_serial_emulator/inf` directory.



The LEDs on the demonstration board will indicate the USB state of the device, as described in the USB Device State and LED Indication Table in the Device section.

8. Open a terminal emulation program of your choice and select the enumerated USB COM port.

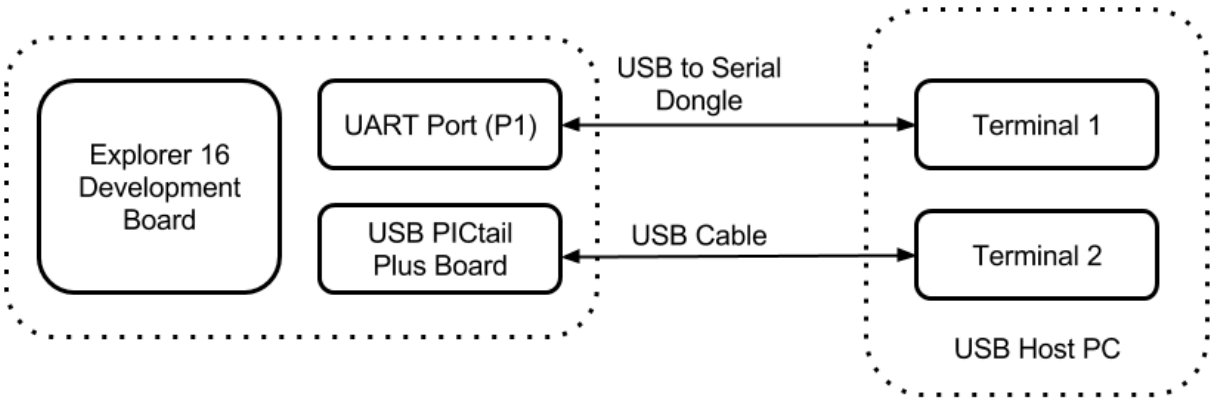
9. Connect the USB-to-Serial Dongle to the same personal computer.

10. Open another instance of the terminal emulation program and select the USB-to-Serial Dongle.

11. Connect the serial connector of the USB-to-Serial Dongle to the UART connector (P1) on the Explorer 16 Development Board.

12. Choose a baud rate of 9600, 1 Stop bit and no parity while opening both of the terminal emulation programs.

The setup should be similar to the following diagram.



Any text entered into the terminal 1 program will be echoed on terminal 2 and vice versa.

cdc_serial_emulator_msd

Demonstrates a USB to Serial Dongle combined with a MSD class.

Description

This demonstration application creates a USB Device that combines the functionality of the cdc_serial_emulator and msd_basic demonstration applications.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the this demonstration application.

Description

To build this project, you must open the `cdc_serial_emulator_msd.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/usb/cdc_serial_emulator_msd`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`cdc_serial_emulator_msd.X`	`<install-dir>/apps/usb/device/cdc_serial_emulator_msd/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx795_pim_e16_int_dyn	pic32mx795_pim+e16	Select this MPLAB X IDE project configuration to run the demonstration on the Explorer 16 Development Board configured for Interrupt mode and dynamic operation. This configuration also requires the PIC32MX795F512L Plug-In Module (PIM) and the USB PICtail Plus Daughter Board.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MX795F512L CAN-USB PIM

Jumper J10 should be removed. Jumper J1 and J2 should connect to positions 1 and 2. This PIM should be used along with the Explorer 16 Development Board and the USB PICtail Plus daughter board. The microcontroller PIM should be plugged into the PIM socket on the board. The USB PICtail Plus daughter board should be connected to the edge connector J9.

On the Explorer 16 Development Board:

- Switch S2 should be set to PIM
- Jumper JP2 should be in place

On the USB PICtail Plus Daughter Board:

- Jumper JP1 should be in place
- Jumper JP2 and JP4 should be removed

On the PIC32MX795F512L PIM:

- Keep jumper J10 open.
- Keep all jumpers in J9 open
- Jumper J1 should be shorted between positions 1 and 2
- Jumper J2 should be shorted between positions 1 and 2

Running the Demonstration

Provides instructions on how to build and run the demonstration.

Description

This demonstration functions as a composite USB Device that combines the features of the devices created by the cdc_serial_emulator and the msd_basic demonstration applications. Refer to Running the Demonstration section of the cdc_serial_emulator demonstration and Running the Demonstration section of the msd_basic demonstration for details on exercising the CDC and MSD functions, respectively.

The LEDs on the demonstration board will indicate the USB state of the device, as described in the USB Device State and LED Indication Table in the Device section.

hid_basic

This demonstration application creates a custom HID device that can be controlled by a personal computer-based utility.

Description

This application creates a custom HID device that can be controlled by a personal computer-based utility. The device allows the USB Host utility to control the LEDs on the board and query the status of a switch.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the USB HID Basic Demonstration.

Description

To build this project, you must open the `hid_basic.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/usb/device/hid_basic`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`hid_basic.X`	`<install-dir>/apps/usb/device/hid_basic/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx460_pim_e16_int_dyn	pic32mx460_pim+e16	Select this MPLAB X IDE project configuration to run the demonstration on the Explorer 16 Development Board configured for Interrupt mode and dynamic operation. This configuration also requires the PIC32MX460F512L Plug-In Module (PIM) and the USB PICtail Plus Daughter Board.
pic32mx_usb_sk3_int_dyn	pic32mx_usb_sk3	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32 USB Starter Kit III configured for Interrupt mode and dynamic operation.
pic32mx_usb_sk2_int_dyn	pic32mx_usb_sk2	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32 USB Starter Kit II configured for Interrupt mode and dynamic operation.

pic32mz_ef_sk_int_dyn	pic32mz_ef_sk	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit configured for Interrupt mode and dynamic operation.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 USB Starter Kit II

Remove jumper JP2.

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32 USB Starter Kit III

Remove jumper JP1.

PIC32MX460F512L PIM

Jumper J10 should be removed. This plug-in module should be used along with the Explorer 16 Development Board and the USB PICtail Plus daughter board. The microcontroller PIM should be plugged into the PIM socket on the board. The USB PICtail Plus daughter board should be connected to the edge connector J9.

On the Explorer 16 Development Board:

- Switch S2 should be set to PIM
- Jumper JP2 should be in place

On the USB PICtail Plus Daughter Board:

- Jumper JP1 should be in place
- Jumper JP2 and JP4 should be removed

On the PIC32MX460F512L PIM:

- Keep jumper J10 open
- Keep all jumpers in J9 open

Running the Demonstration

Provides instructions on how to build and run the HID Basic demonstration.

Description

This demonstration uses the selected hardware platform as a HID class USB device, but uses the HID class for general purpose I/O operations. While compiling, select the appropriate MPLAB X IDE project configuration based on the demonstration board. Refer to Building the Application for details.

Typically, the HID class is used to implement human interface products, such as mice and keyboards. The HID protocol, is however, quite flexible, and can be adapted and used to send/receive general purpose data to/from a USB device. Using the HID class for general purpose I/O operations is quite advantageous, in that it does not require any kind of custom driver installation process. HID class drivers are already provided by and are distributed with common operating systems. Therefore, upon plugging in a HID class device into a typical computer system, no user installation of drivers is required, the installation is fully automatic.

The LEDs on the demonstration board will indicate the USB state of the device, as described in the USB Device State and LED Indication Table in the Device section.

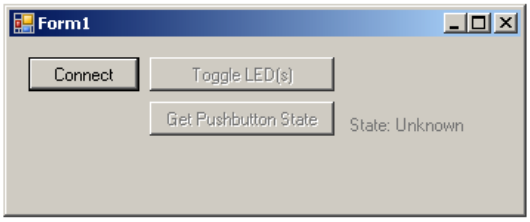
HID devices primarily communicate through one interrupt IN endpoint and one interrupt OUT endpoint. In most applications, this effectively limits the maximum achievable bandwidth for full speed HID devices to 64 kBytes/s of IN traffic, and 64 kBytes/s of OUT traffic (64 kB/s, but effectively "full duplex").

The `GenericHIDSimpleDemo.exe` program, and the associated firmware demonstrate how to use the HID protocol for basic general purpose USB data transfer.

Before you can run the `GenericHIDSimpleDemo.exe` executable, you will need to have the Microsoft® .NET Framework Version 2.0 Redistributable Package (later versions are probably acceptable, but have not been tested) installed on your computer. Programs that were built in the Visual Studio® .NET languages require the .NET redistributable package. The redistributable package can be freely downloaded from Microsoft's website. Users of Windows Vista® operating systems will not need to install the .NET framework, as it comes preinstalled as part of the operating system.

Launching the Application

To launch the application, simply double click the executable `GenericHIDSimpleDemo.exe` in the `<install-dir>\apps\usb\device\hid_basic\bin` directory. A property sheet similar to the following should appear:



Note: If instead of this window, an error message appears while trying to launch the application, it is likely the Microsoft .NET Framework Version 2.0 Redistributable Package has not yet been installed. Please install it and try again.

Send/Receive Packets

To begin sending/receiving packets to the device, you must first find and connect to the device. As configured by default, the application is looking for HID class USB devices with VID = 0x04D8 and PID = 0x003F. The device descriptor in the firmware project meant to be used with this demonstration uses the same VID/PID. If you plug in a USB device programmed with the correct precompiled `.hex` file, and click **Connect**, the other push buttons should become enabled. If clicking **Connect** has no effect, it is likely the USB device is either not connected, or has not been programmed with the correct firmware.

Clicking **Toggle LED(s)** should send a single packet of general purpose generic data to the HID class USB peripheral device. The data will arrive on the interrupt OUT endpoint. The firmware has been configured to receive this generic data packet, parse the packet looking for the Toggle LED(s) command, and should respond appropriately by controlling the LED(s) on the demonstration board.

The Get Pushbutton State option will send one packet of data over the USB to the peripheral device (to the interrupt OUT endpoint) requesting the current push button state. The firmware will process the received Get Pushbutton State command, and will prepare an appropriate response packet depending upon the pushbutton state.

The following table shows the button that has to be pressed on the demonstration board to see the change in the push button state.

Demonstration Board	Button
PIC32 USB Starter Kit II PIC32 USB Starter Kit III PIC32MZ Embedded Connectivity (EC) Starter Kit PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit	SW1
Explorer 16 Development Board	S3

hid_joystick

Demonstrates a USB HID device emulating a joystick.

Description

This demonstration application creates a custom HID joystick. This application is only intended to demonstrate creation of Joystick HID Report descriptors and may not be a definite end solution. The end application requirements may need the report descriptor to be modified.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the USB HID Joystick Demonstration.

Description

To build this project, you must open the `hid_joystick.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/usb/device/hid_joystick`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`hid_joystick.X`	`<install-dir>/apps/usb/device/hid_joystick/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx460_pim_e16_int_dyn	pic32mx460_pim+e16	Select this MPLAB X IDE project configuration to run the demonstration on the Explorer 16 Development Board configured for Interrupt mode and dynamic operation. This configuration also requires the PIC32MX460F512L Plug-In Module (PIM) and the USB PICtail Plus Daughter Board.
pic32mx_usb_sk3_int_dyn	pic32mx_usb_sk3	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32 USB Starter Kit III configured for Interrupt mode and dynamic operation.
pic32mx_usb_sk2_int_dyn	pic32mx_usb_sk2	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32 USB Starter Kit II configured for Interrupt mode and dynamic operation.
pic32mz_ef_sk_int_dyn	pic32mz_ef_sk	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit configured for Interrupt mode and dynamic operation.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 USB Starter Kit II

Remove jumper JP2.

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32 USB Starter Kit III

Remove jumper JP1.

PIC32MX460F512L PIM

Jumper J10 should be removed. This plug-in module should be used along with the Explorer 16 Development Board and the USB PICtail Plus daughter board. The microcontroller PIM should be plugged into the PIM socket on the board. The USB PICtail Plus daughter board should be connected to the edge connector J9.

On the Explorer 16 Development Board:

- Switch S2 should be set to PIM
- Jumper JP2 should be in place

On the USB PICtail Plus Daughter Board:

- Jumper JP1 should be in place
- Jumper JP2 and JP4 should be removed

On the PIC32MX460F512L PIM:

- Keep jumper J10 open
- Keep all jumpers in J9 open

Running the Demonstration

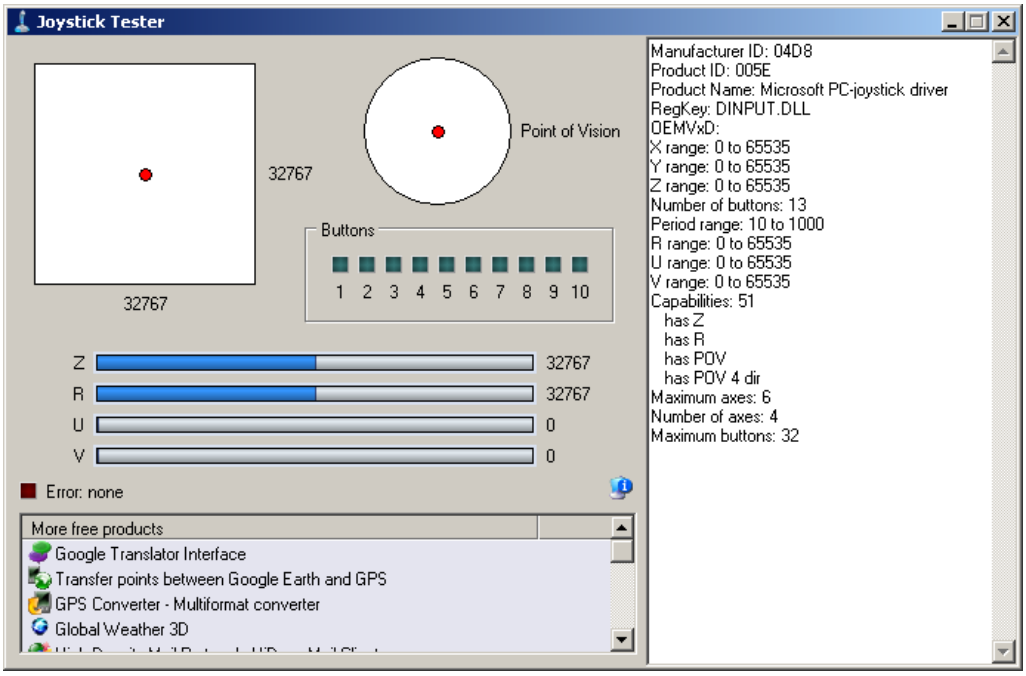
Provides instructions on how to build and run the USB HID Joystick demonstration.

Description

This demonstration uses the selected hardware platform as a USB Joystick. Select the appropriate MPLAB X IDE project configuration based on the demonstration board. Refer to Building the Application for details.

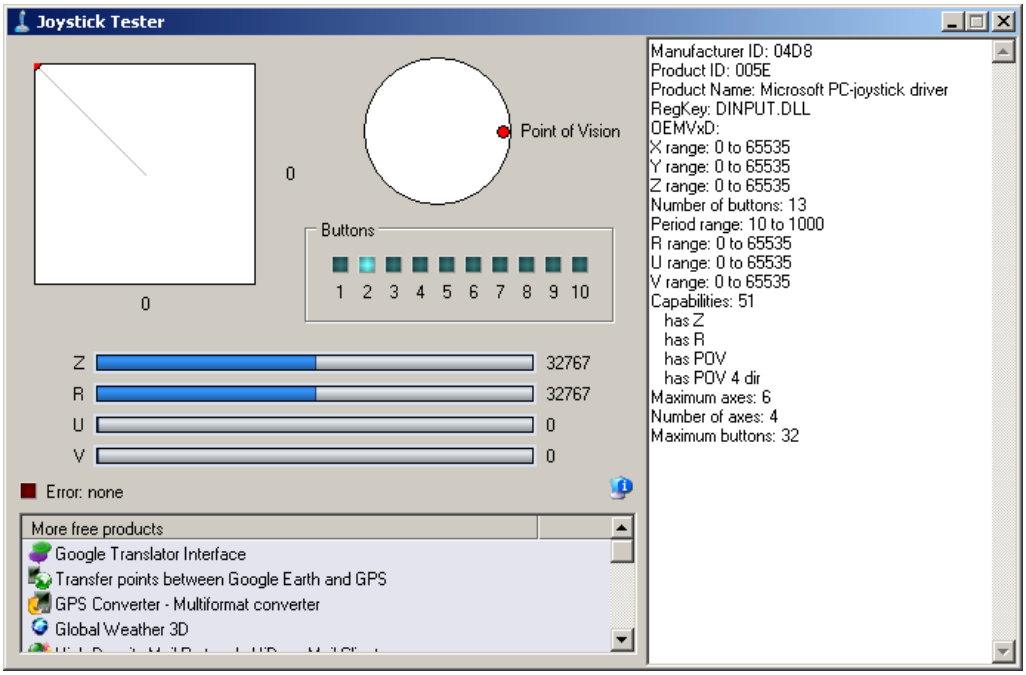
The LEDs on the demonstration board will indicate the USB state of the device, as described in the USB Device State and LED Indication Table in the Device section.

To test the joystick feature, navigate to the `<install-dir>/apps/usb/device/hid_joystick/bin` directory and open `JoystickTester.exe`:



Pressing the button will cause the device to:

- Indicate that the "x" button is pressed, but no others
- Move the hat switch to the "east" position
- Move the X and Y coordinates to their extreme values



The Following table shows the button that has to be pressed on the demonstration board to emulate the joystick.

Demonstration Board	Button
PIC32 USB Starter Kit II PIC32 USB Starter Kit III PIC32MZ Embedded Connectivity (EC) Starter Kit PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit	SW1
Explorer 16 Development Board	S3

hid_keyboard

Demonstrates a USB HID device, emulating a keyboard.

Description

This demonstration application creates a Generic HID keyboard. Pressing a key on the board emulates a keyboard key press.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the USB HID Keyboard Demonstration.

Description

To build this project, you must open the `hid_keyboard.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/usb/device/hid_keyboard`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`hid_keyboard.X`	`<install-dir>/apps/usb/device/hid_keyboard/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx460_pim_e16_int_dyn	pic32mx460_pim+e16	Select this MPLAB X IDE project configuration to run the demonstration on the Explorer 16 Development Board configured for Interrupt mode and dynamic operation. This configuration also requires the PIC32MX460F512L Plug-In Module (PIM) and the USB PICtail Plus Daughter Board.
pic32mx_usb_sk3_int_dyn	pic32mx_usb_sk3	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32 USB Starter Kit III configured for Interrupt mode and dynamic operation.
pic32mx_usb_sk2_int_dyn	pic32mx_usb_sk2	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32 USB Starter Kit II configured for Interrupt mode and dynamic operation.
pic32mz_ef_sk_int_dyn	pic32mz_ef_sk	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit configured for Interrupt mode and dynamic operation.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 USB Starter Kit II

Remove jumper JP2.

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32 USB Starter Kit III

Remove jumper JP1.

PIC32MX460F512L PIM

Jumper J10 should be removed. This plug-in module should be used along with the Explorer 16 Development Board and the USB PICtail Plus daughter board. The microcontroller PIM should be plugged into the PIM socket on the board. The USB PICtail Plus daughter board should be

connected to the edge connector J9.

On the Explorer 16 Development Board:

- Switch S2 should be set to PIM
- Jumper JP2 should be in place

On the USB PICtail Plus Daughter Board:

- Jumper JP1 should be in place
- Jumper JP2 and JP4 should be removed

On the PIC32MX460F512L PIM:

- Keep jumper J10 open
- Keep all jumpers in J9 open

Running the Demonstration

Provides instructions on how to build and run the USB HID Keyboard demonstration.

Description

This demonstration uses the selected hardware platform as a USB keyboard. While compiling, select the appropriate MPLAB X IDE project configuration based on the demonstration board. Refer to Building the Application for details.

The LEDs on the demonstration board will indicate the USB state of the device, as described in the USB Device State and LED Indication Table in the Device section.

Before pressing the button, select a window in which it is safe to type text freely. Pressing the button on the demonstration board will cause the device to print a character on the screen.

The following table shows the button that has to be pressed on the demonstration board to print a character.

Demonstration Board	Button
PIC32 USB Starter Kit II PIC32 USB Starter Kit III PIC32MZ Embedded Connectivity (EC) Starter Kit PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit	SW1
Explorer 16 Development Board	S3

hid_mouse

Demonstrates a USB HID device, emulating a mouse pointing device.

Description

This demonstration application creates a USB HID based two-button mouse device. When connected, the device emulates mouse operation by moving the cursor in a circular pattern.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the USB HID Mouse Demonstration.

Description

To build this project, you must open the `hid_mouse.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/usb/device/hid_mouse`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`hid_mouse.X`	`<install-dir>/apps/usb/device/hid_mouse/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx460_pim_e16_int_dyn	pic32mx460_pim+e16	Select this MPLAB X IDE project configuration to run the demonstration on the Explorer 16 Development Board configured for Interrupt mode and dynamic operation. This configuration also requires the PIC32MX460F512L Plug-In Module (PIM) and the USB PICtail Plus Daughter Board.
pic32mx_usb_sk3_int_dyn	pic32mx_usb_sk3	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32 USB Starter Kit III configured for Interrupt mode and dynamic operation.
pic32mx_usb_sk2_int_dyn	pic32mx_usb_sk2	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32 USB Starter Kit II configured for Interrupt mode and dynamic operation.
pic32mz_ef_sk_int_dyn	pic32mz_ef_sk	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit configured for Interrupt mode and dynamic operation.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 USB Starter Kit II

Remove jumper JP2.

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32 USB Starter Kit III

Remove jumper JP1.

PIC32MX460F512L PIM

Jumper J10 should be removed. This plug-in module should be used along with the Explorer 16 Development Board and the USB PICtail Plus daughter board. The microcontroller PIM should be plugged into the PIM socket on the board. The USB PICtail Plus daughter board should be connected to the edge connector J9.

On the Explorer 16 Development Board:

- Switch S2 should be set to PIM
- Jumper JP2 should be in place

On the USB PICtail Plus Daughter Board:

- Jumper JP1 should be in place
- Jumper JP2 and JP4 should be removed

On the PIC32MX460F512L PIM:

- Keep jumper J10 open
- Keep all jumpers in J9 open

Running the Demonstration

Provides instructions on how to build and run the HID Mouse Demonstration.

Description

This demonstration uses the selected hardware platform as a USB mouse. While compiling, select the appropriate MPLAB X IDE project configuration based on the demonstration board. Refer to Building the Application for details.

The LEDs on the demonstration board will indicate the USB state of the device, as described in the USB Device State and LED Indication Table in the Device section.

Before connecting the board to the computer through the USB cable please be aware that the device will begin moving the mouse cursor on the computer. There are two ways to stop the device from allowing the cursor to continue to move. The first way is to disconnect the device from the computer. The second is to press the correct button on the hardware platform. Pressing the button again will cause the mouse cursor to start moving in a circle again.

The following table shows the button that has to be pressed on the demonstration board to stop the circular motion:

Demonstration Board	Button
PIC32 USB Starter Kit II PIC32 USB Starter Kit III PIC32MZ Embedded Connectivity (EC) Starter Kit PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit	SW1
Explorer 16 Development Board	S3

hid_msd_basic

Demonstrates a HID Device Class and MSD class composite USB Device.

Description

This demonstration application creates a USB Device that combines the functionality of the hid_basic and msd_basic demonstration applications.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the this demonstration application.

Description

To build this project, you must open the `hid_msd_basic.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/usb/device/hid_msd_basic`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`hid_msd_basic.X`	`<install-dir>/apps/usb/device/hid_msd_basic/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx_usb_sk2_int_dyn	pic32mx_usb_sk2	Select this configuration to run the demonstration application on the PIC32 USB Starter Kit II configured for Interrupt mode and dynamic operation.
pic32mz_ef_sk_int_dyn	pic32mz_ef_sk	Select this configuration to run the demonstration application on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit configured for Interrupt mode and dynamic operation.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 USB Starter Kit II

Remove jumper JP2.

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the demonstration.

Description

This demonstration functions as composite USB Device that combines the features of the devices created by the hid_basic and the msd_basic demonstration applications. Refer to Running the Demonstration section of the hid_basic demonstration and Running the Demonstration section of the msd_basic demonstration for details on exercising the HID and MSD functions, respectively.

The LEDs on the demonstration board will indicate the USB state of the device, as described in the USB Device State and LED Indication Table in the Device section.

msd_basic

Demonstrates a USB MSD Device emulating a Flash Drive.

Description

This demonstration application creates a Flash drive using the Mass Storage Device Class.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the USB MSD Basic Demonstration.

Description

To build this project, you must open the `msd_basic.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/usb/device/msd_basic`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`msd_basic.X`	`<install-dir>/apps/usb/device/msd_basic/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx_bt_sk_int_dyn	pic32mx_bt_sk	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32 Bluetooth Starter Kit configured for Interrupt mode and dynamic operation.
pic32mx_usb_sk2_poll_dyn	pic32mx_usb_sk2	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32 USB Starter Kit II with the USB Device Stack configured for Polled mode and dynamic operation..
pic32mx_usb_sk2_int_dyn	pic32mx_usb_sk2	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32 USB Starter Kit II configured for Interrupt mode and dynamic operation.
pic32mx_usb_sk3_int_dyn	pic32mx_usb_sk3	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32 USB Starter Kit III configured for Interrupt mode and dynamic operation.
pic32mz_ef_sk_int_dyn	pic32mz_ef_sk	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit configured for Interrupt mode and dynamic operation.
pic32mz_ef_sk_poll_dyn	pic32mz_ef_sk	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit configured for Polled mode and dynamic operation.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 USB Starter Kit II

Remove jumper JP2.

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32 USB Starter Kit III

Remove jumper JP1.

PIC32 Bluetooth Starter Kit

No hardware related configuration or jumper settings required.

PIC32MX460F512L PIM

Jumper J10 should be removed. This plug-in module should be used along with the Explorer 16 Development Board and the USB PICtail Plus daughter board. The microcontroller PIM should be plugged into the PIM socket on the board. The USB PICtail Plus daughter board should be connected to the edge connector J9.

On the Explorer 16 Development Board:

- Switch S2 should be set to PIM
- Jumper JP2 should be in place

On the USB PICtail Plus Daughter Board:

- Jumper JP1 should be in place
- Jumper JP2 and JP4 should be removed

Running the Demonstration

Provides instructions on how to build and run the USB MSD Basic demonstration.

Description

This demonstration uses the selected hardware platform as a logical drive on the computer using the internal Flash of the device as the drive storage media. Connect the hardware platform to a computer through a USB cable. The device should appear as a new drive on the computer named "Drive Name". The drive can used to store files.

The LEDs on the demonstration board will indicate the USB state of the device, as described in the USB Device State and LED Indication Table in the Device section.

Note: Reprogramming the development board will cause any stored files to be erased.

msd_fs_spiflash

This application demonstrates accessing the SPI Flash connected to the PIC32 device as a media by multiple clients.

Description

This application demonstrates accessing the SPI Flash connected to the PIC32 device as a media by multiple clients. When connected via USB to the Host Computer, the SPI Flash is shown as the storage media. The Host writes files to the media, which is later accessed by the application running on the PIC32 device using the File System.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the USB MSD File System SPI Flash Demonstration.

Description

To build this project, you must open the `msd_fs_spiflash.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/usb/device/msd_fs_spiflash`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`msd_fs_spiflash.X`	`<install-dir>/apps/usb/device/msd_fs_spiflash/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
bt_audio_dk_int_dyn	bt_audio_dk	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32 Bluetooth Audio Development Kit.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 Bluetooth Audio Develoment Kit

Ensure that switch S1 is set to PIC32_MCLR.

Running the Demonstration

Provides instructions on how to build and run the USB MSD File System SPI Flash demonstration.

Description

This demonstration shows an example of:

- Accessing the media attached to PIC32 by multiple clients
- Application running on the PIC32 firmware accesses the media using the MPLAB Harmony File System

When connected to the USB Host the very first time, the user is expected to format the media and create a file named `FILE.TXT` in the root directory of the media. The user can update the file to provide input for the application to glow the LEDs present on the development kit. The application running on the PIC32 reads and interprets the data present in the file and accordingly turns ON or OFF the LEDs LED8 and LED9 of the development kit. The format of input in the file `FILE.TXT` should be as follows:

- For turning ON an LED:
  - LED8:1
  - LED9:1
- For turning OFF an LED:
  - LED8:0
  - LED9:0

After having set the appropriate values in the file, the user can then press and release the wwitch SW1 located on the development kit for the MPLAB Harmony File System running on the PIC32 to act upon the contents of the file.

The FS state machine of the demonstration is only triggered by the switch SW1. When the user presses and releases SW1 the following occurs:

- LED5 is turned ON to indicate that the FS state machine is running
- The USB is detached
- The file system on the SPI Flash is mounted
- The contents of `FILE.TXT` is read and acted upon. Depending on the values set in the file, the LEDs are either turned ON or OFF.
- Next, the file system is unmounted and the USB is reattached
- LED5 is turned OFF to indicate that FS state machine is no longer running
- If LED6 is turned ON during any part of the demonstration, this indicates the demonstration has failed

msd_multiple_luns

This topic demonstrates data transfer between two storage media - SD card and non-volatile memory (NVM) - and a computer through USB Mass Storage Device (MSD).

Description

This application demonstrates the creation of a USB device with multiple logical units. The storage media, SD Card, acts as one logical unit, and the NVM acts as the second logical unit. Data transfer between a computer and the logical units (SD Card / NVM) takes place through USB MSD.

Building the Application

This section identifies the MPLAB X IDE project name and location, and then lists and describes the available configurations for the USB MSD multiple LUNs demonstration.

Description

To build this project, you must open the `msd_multiple_luns.X` project in MPLAB X IDE, and then select the desired configuration. The following tables lists and describes the project and the supported configurations. The parent folder for these files is

`<install-dir>/apps/usb/device/msd_multiple_luns`

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`msd_multiple_luns.X`	`<install-dir>/apps/usb/device/msd_multiple_luns/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx470_curiosity	pic32mx470_curiosity	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32MX470 Curiosity board with the USB device stack configured for Interrupt mode and full speed operation. The LUN0 media type is configured as SD Card and LUN1 media type is configured as NVM.
pic32mz_ef_curiosity	pic32mz_ef_curiosity	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32MZ EF Curiosity board with the USB device stack configured for Interrupt mode and high speed operation. The LUN0 media type is configured as SD Card and LUN1 media type is configured as NVM.

Configuring the Hardware

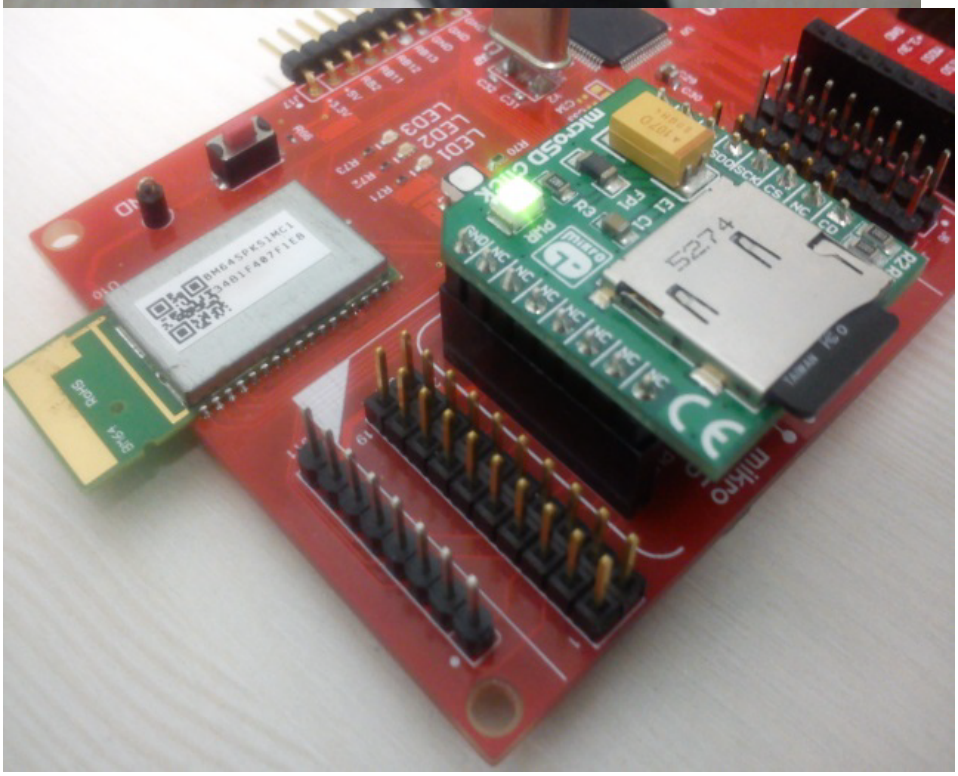
This section describes how to configure the supported hardware.

Description

PIC32MX470 Curiosity Development Board

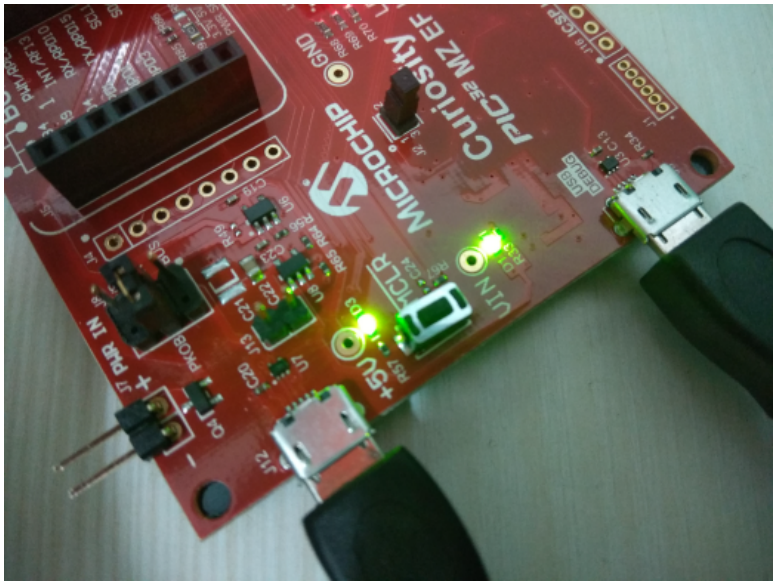
1. Ensure that a jumper is placed at 4-3 on J8, to select supply from debug USB connector.
2. Mount the SD Click board, "microSD click" from MikroElektronika (http://www.mikroe.com/click/microsd/) on the mikro bus interface J10.
3. Plug a micro SD card into the microSD click board card slot.
4. Power the PIC32MX470 Curiosity Development Board from a Host PC through a Type-A male to mini-B USB cable connected to Mini-B port (J3).
5. Connect a Type-A male to micro USB cable to the micro USB port (J12) on PIC32MX470 Curiosity Development Board.

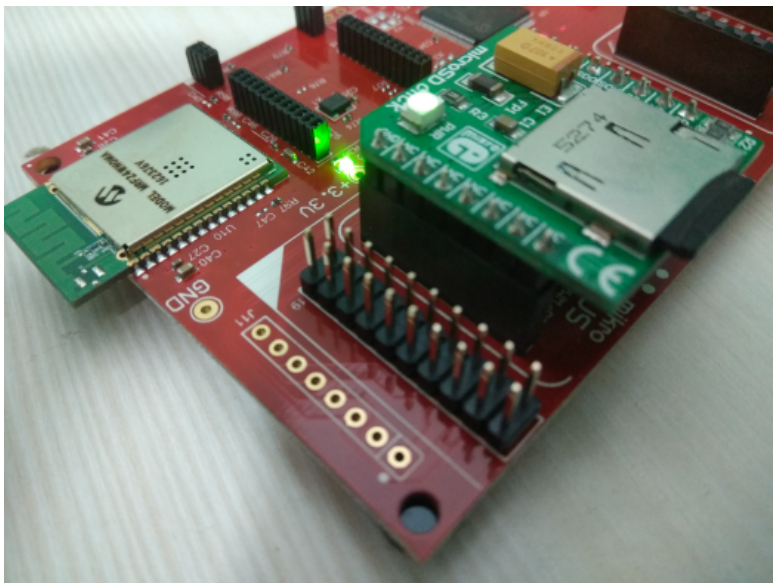




PIC32MZ EF Curiosity Development Board

1. Ensure that a jumper is placed at 4-3 on J8, to select supply from debug USB connector.
2. Mount the SD Click board, "microSD click" from MikroElektronika (http://www.mikroe.com/click/microsd/) on the mikro bus interface J10.
3. Plug a micro SD card into the microSD click board card slot.
4. Power the PIC32MZ EF Curiosity Development Board from a Host PC through a Type-A male to micro USB cable connected to micro USB port (J3).
5. Connect a Type-A male to micro USB cable to the micro USB port (J12) on PIC32MZ EF Curiosity Development Board.





Running the Demonstration

This section provides instructions about how to build and run the USB MSD Multiple LUNs demonstration.

Description

This demonstration uses SD card and NVM as drive storage media and shows them as two logical drives on the computer.

- Connect the hardware platform to a computer through a USB cable.
  - The device should appear as two new drives on the computer.
  - The NVM media should appear as "Drive Name" and should have a sample “FILE.txt” file. The drive name for the SD card media depends on the micro SD card vendor. The drives can then be used to store files.
  - The LEDs on the demonstration board will indicate the USB state of the device, as described in the USB Device State and LED Indication Table in the Device section.

Note: Reprogramming the development board will cause any stored files in the NVM media to be erased.

msd_sdcard

Demonstrates data transfer from a SD card and a computer through USB MSD.

Description

This application demonstrates the usage of a SD card reader through the USB Mass Storage Device (MSD) class to transfer data between a computer and SD card. High-Speed USB is used for communication between the Host computer and the PIC32 device, while a SD card is used as the storage medium.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the USB MSD SD Card Demonstration.

Description

To build this project, you must open the `msd_sdcard.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/usb/device/msd_sdcard`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`msd_sdcard.X`	`<install-dir>/apps/usb/device/msd_sdcard/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mz_ec_sk_int_dyn	pic32mz_ec_sk+meb2	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32MZ EC Starter Kit connected to the MEB II. The media drivers are configured for Interrupt mode and dynamic operation.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MZ Embedded Connectivity (EC) Starter Kit and Multimedia Expansion Board II (MEB II)

No hardware related configuration or jumper settings required.

Running the Demonstration

Provides instructions on how to build and run the USB MSD SD Card demonstration.

Description

This demonstration uses the selected hardware platform as a logical drive on the computer using the SD card as the drive storage media. Connect the hardware platform to a computer through a USB cable. The device should appear as a new drive on the computer named "Drive Name". The drive can then be used to store files.

The LEDs on the demonstration board will indicate the USB state of the device, as described in the USB Device State and LED Indication Table in the Device section.

vendor

Demonstrates a custom USB Device created by using the USB Device Layer Endpoint functions.

Description

This demonstration application creates a custom USB device using the USB Device Layer Endpoint functions.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Vendor USB Device Demonstration.

Description

To build this project, you must open the `vendor.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/usb/device/vendor`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`vendor.X`	`<install-dir>/apps/usb/device/vendor/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx460_pim_e16_int_dyn	pic32mx460_pim+e16	Select this MPLAB X IDE project configuration to run the demonstration on the Explorer 16 Development Board configured for Interrupt mode and dynamic operation. This configuration also requires the PIC32MX460F512L Plug-In Module (PIM) and the USB PICtail Plus Daughter Board.
pic32mx_usb_sk3_int_dyn	pic32mx_usb_sk3	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32 USB Starter Kit III configured for Interrupt mode and dynamic operation.
pic32mx_usb_sk2_int_dyn	pic32mx_usb_sk2	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32 USB Starter Kit II configured for Interrupt mode and dynamic operation.
pic32mz_ef_sk_int_dyn	pic32mz_ef_sk	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit configured for Interrupt mode and dynamic operation.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 USB Starter Kit II

Remove jumper JP2.

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32 USB Starter Kit III

Remove jumper JP1.

PIC32MX460F512L PIM

Jumper J10 should be removed. This plug-in module should be used along with the Explorer 16 Development Board and the USB PICtail Plus daughter board. The microcontroller PIM should be plugged into the PIM socket on the board. The USB PICtail Plus daughter board should be connected to the edge connector J9.

On the Explorer 16 Development Board:

- Switch S2 should be set to PIM
- Jumper JP2 should be in place

On the USB PICtail Plus Daughter Board:

- Jumper JP1 should be in place

- Jumper JP2 and JP4 should be removed

On the PIC32MX460F512L PIM:

- Keep jumper J10 open
- Keep all jumpers in J9 open

Running the Demonstration

Provides instructions on how to build and run the Vendor USB Device demonstration.

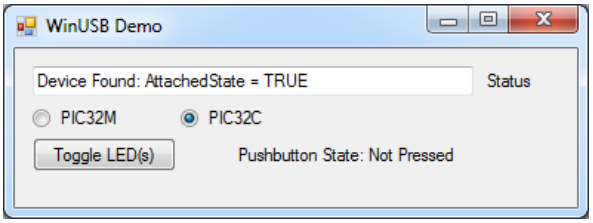
Description

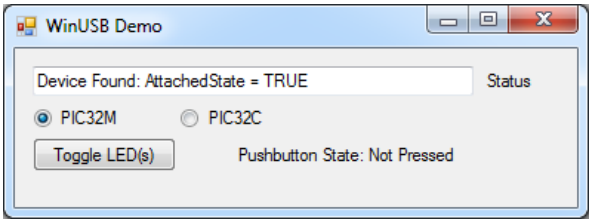
The Vendor device can be exercised by using the WinUSB PnP Demonstration application, which is provided in your installation of MPLAB Harmony.

The LEDs on the demonstration board will indicate the USB state of the device, as described in the USB Device State and LED Indication Table in the Device section.

This application allows the state of the LEDs on the board to be toggled and indicates the state of a switch (pressed/released) on the board.

To launch the application, double click `WinUSB PnP Demo.exe` located in `<install dir>/apps/usb/device/vendor/bin`. A dialog box similar to the following should appear:





The appropriate device family that is under testing should be selected in the utility. Pressing the Toggle LED button will cause the LED on the board to toggle. The Pushbutton State field in the application indicates the state of a button on connected USB Device. Pressing the switch on the development board will update the Pressed/Not Pressed status of the Pushbutton State field.

Demonstration Board	Button
PIC32 USB Starter Kit II PIC32 USB Starter Kit III PIC32MZ Embedded Connectivity (EC) Starter Kit PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit	SW1
Explorer 16 Development Board	S3

Note: The device family under test should be selected appropriately. An incorrect selection will result in an invalid push button status.

Host

This section describes the USB Host demonstrations.

audio_speaker

This application demonstrates the use of the Audio v1.0 Host Class Driver to enumerate and operate an audio speaker device.

Description

This application demonstrates the use of the Audio v1.0 Host Class Driver to enumerate and an audio speaker device. The application uses the USB Host Layer and Audio 1.0 class driver to enumerate an Audio v1.0 USB device. The demonstration host application then operates and uses the functionality of the attached audio speaker device.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the USB Host Audio Speaker Demonstration.

Description

To build this project, you must open the `audio_speaker.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/usb/host/audio_speaker`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`audio_speaker.X`	`<install-dir>/apps/usb/host/audio_speaker/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx_usb_sk2_int_dyn	pic32mx_usb_sk2	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32 USB Starter Kit II configured for Interrupt mode and dynamic operation.
pic32mz_ef_sk_int_dyn	pic32mz_ef_sk	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit configured for Interrupt mode and dynamic operation.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 USB Starter Kit II

JP2 should be in place.

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the USB Host Audio v1.0 Basic Demo.

Description

This application demonstrates the use of the Audio v1.0 Host Class Driver to enumerate and operate an Audio v1.0 Device. The application uses the USB Host layer and Audio v1.0 class driver to enumerate a Audio v1.0 USB device. The demonstration host application then operates and uses the functionality of the attached Audio v1.0 Device.

Prior to using this demonstration, it is recommended to review the MPLAB Harmony Release Notes for any known issues. A PDF copy of the release notes is provided in the `<install-dir>/doc` folder of your installation.

1. Open the project in MPLAB X IDE and select the desired project configuration.
2. Build the code and program the device.
3. Attach a commercially available USB speaker to the board.
4. LED1 is turned ON if the attached device is accepted by the Audio 1.0 class driver.
5. The speaker should produce a 1 kHz sine wave.
6. LED2 will continue blinking if the demonstration application cannot accept the device.
7. Press switch SW1 to mute the audio.
8. Press switch SW2 to unmute the audio

cdc_basic

This application demonstrates the use of the CDC Host Class Driver to enumerate and operate a CDC Device.

Description

This application demonstrates the use of the CDC Host Class Driver to enumerate and operate a CDC Device. The application uses the USB Host layer and CDC class driver to enumerate a CDC USB device. The demonstration host application then operates and uses the functionality of the attached CDC Device.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the USB CDC Host Basic Demonstration.

Description

To build this project, you must open the `cdc_basic.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/usb/host/cdc_basic`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`cdc_basic.X`	`<install-dir>/apps/usb/host/cdc_basic/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx_usb_sk2_int_dyn	pic32mx_usb_sk2	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32 USB Starter Kit II configured for Interrupt mode and dynamic operation.
pic32mx_usb_sk2_poll_dyn	pic32mx_usb_sk2	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32 USB Starter Kit II configured for Polled mode and dynamic operation.
pic32mz_ef_sk_int_dyn	pic32mz_ef_sk	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit configured for Interrupt mode and dynamic operation.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 USB Starter Kit II

JP2 should be in place.

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the USB Host CDC Basic Demo.

Description

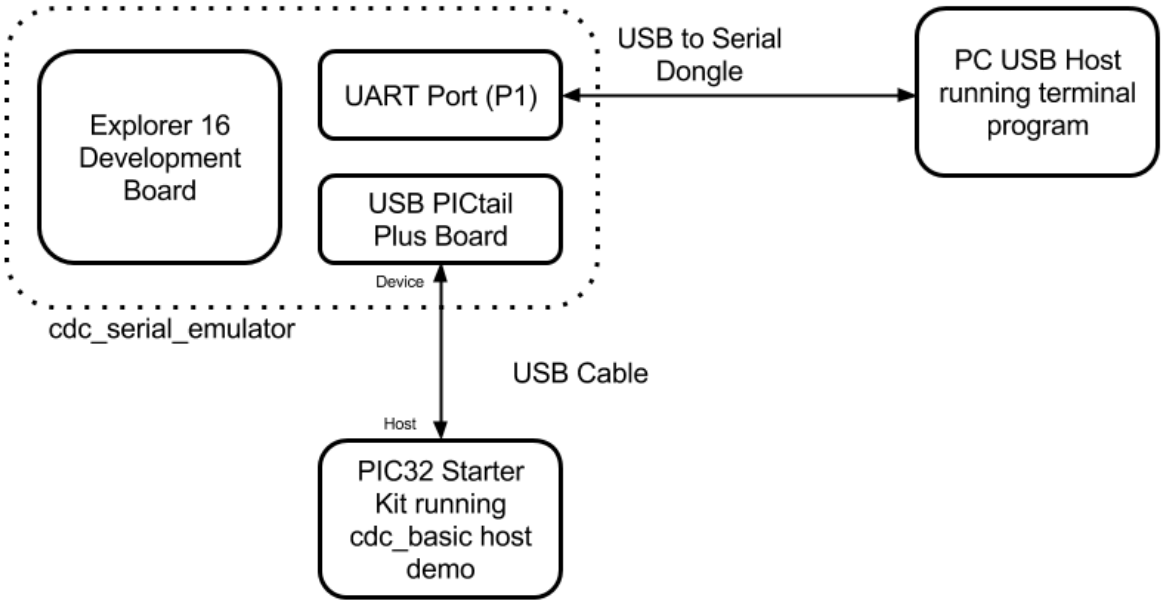
This application demonstrates the use of the CDC Host Class Driver to enumerate and operate a CDC Device. The application uses the USB Host layer and CDC class driver to enumerate a CDC USB device. The demonstration host application then operates and uses the functionality of the attached CDC Device.

Prior to using this demonstration, it is recommended to review the MPLAB Harmony Release Notes for any known issues. A PDF copy of the

release notes is provided in the `<install-dir>/doc` folder of your installation.

1. Open the project in MPLAB X IDE and select the desired project configuration.
2. Build the code and program the device.
3. Follow the directions for setting up and running the cdc_serial_emulator USB device demonstration.
4. Connect the UART (P1) port on the Explorer 16 Development Board (running the cdc_serial_emulator demonstration) to a USB Host personal computer via a commercially available Serial-to-USB Dongle.
5. Start a terminal program on the USB Host personal computer and select the Serial-to-USB Dongle as the communication port. Select the baud rate as 9600, no parity, 1 Stop bit and no flow control.
6. Connect the mini – B connector on the USB PICtail Plus Daughter Board, of the cdc_serial_emulator demonstration setup, to the USB host connector on the starter kit. For PIC32M-based starter kits, connect to the on-board Type-A connector.
7. A prompt (LED :) will be displayed immediately on the terminal emulation program.
8. Pressing either the 1, 2, or 3 key on the USB Host keyboard will cause LEDs on the PIC32 starter kit (running the USB CDC Host application) to switch on, respectively. On PIC32M-based starter kits, the LEDs are LED1, LED2, and LED3.
9. The prompt will again be displayed on terminal emulation program, and step 8 can be repeated.

The setup should be similar to the following diagram.



The cdc_serial_emulator demonstration emulates a USB-to-Serial Dongle. The CDC Host (running the cdc_basic demonstration application) sends the prompt message to the CDC device. The CDC device forwards the prompt to the UART port from where it is transmitted to the personal computer USB Host through the USB-to-Serial Dongle. A key press on the personal computer USB Host is transmitted to the CDC device, which in turn presents the key press data to the CDC host. The cdc_basic demonstration then analyzes the key press data and switches on the respective LED.

cdc_msd

Demonstrates host support for multiple device classes.

Description

This demonstration application creates a USB Host that can support different device classes in one application.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for this USB CDC MSD Host Demonstration.

Description

To build this project, you must open the `cdc_msd.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/usb/host/cdc_msd`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
cdc_msd.X	<install-dir>/apps/usb/host/cdc_msd/firmware

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within ./firmware/src/system_config.

Project Configuration Name	BSP Used	Description
pic32mx_usb_sk2_int_dyn	pic32mx_usb_sk2	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32 USB Starter Kit II configured for Interrupt mode and dynamic operation.
pic32mz_ef_sk_int_dyn	pic32mz_ef_sk	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit configured for Interrupt mode and dynamic operation.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 USB Starter Kit II

JP2 should be in place.

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the USB CDC MSD demonstration.

Description

This demonstration application creates a USB Host application that enumerates a CDC and a MSD device. This application combines the functionality of the Host cdc_basic and msd_basic demonstration applications into one application. If a CDC device is connected, the demonstration application behaves like the cdc_basic host application. If a MSD device is connected, the demonstration application behaves like the msd_basic host application.

Refer to Running the Demonstration section of the host cdc_basic demonstration and the Running the Demonstration section of the host msd_basic demonstration for details on exercising the CDC and MSD host aspects of the demonstration.

hid_basic_keyboard

Demonstrates using the USB HID Host Client driver with the Keyboard Usage driver to facilitate the use of a USB HID Keyboard with a PIC32 USB Host.

Description

This application demonstrates the use of the USB HID Host Client Driver to enumerate and operate a HID keyboard device. The application uses the USB Host layer, HID Client driver and HID Keyboard Usage driver to enumerates a USB keyboard and understand keyboard press release events.

The keyboard events are displayed using a terminal emulator on a personal computer.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the USB HID Basic Keyboard Demonstration.

Description

To build this project, you must open the hid_basic_keyboard.X project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/usb/host/hid_basic_keyboard`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`hid_basic_keyboard.X`	`<install-dir>/apps/usb/host/hid_basic_keyboard/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx795_pim_e16_int_dyn	pic32mx795_pim+e16	Select this MPLAB X IDE project configuration to run the demonstration configured for Interrupt mode and dynamic operation on the PIC32MX795F512L PIM connected to the Explorer 16 Development Board with the USB PICtail Plus Daughter Board attached.
pic32mz_ef_sk_int_dyn	pic32mz_ef_sk	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit configured for Interrupt mode and dynamic operation.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit

No hardware related configuration or jumper setting changes are necessary.

Explorer 16 Development Board

- Switch S2 should be set to PIM

USB PICtail Plus Daughter Board

- Jumper the Host Enable pins
- Device Enable and OTG Enable should be open

PIC32MX795F512L CAN-USB PIM

- Keep jumper J10 open
- Keep all jumpers in J9 open
- Jumper J1 should be shorted between positions 1 and 2. This configuration is only applicable for the PIC32MX795F512L USB CAN PIM (MA320003), and not the PIC32MX795F512L USB PIM (MA320002).
- Jumper J2 should be shorted between positions 1 and 2. This configuration is only applicable for the PIC32MX795F512L USB CAN PIM (MA320003) and not the PIC32MX795F512L USB PIM (MA320002).

For the pic32mx795_pim_e16_int_dyn configuration:

1. Ensure that the PIC32MX795F512L PIM is connected properly to the PIM socket on the Explorer 16 Development Board.
2. Connect the Serial Port connector on the Explorer 16 Development Board to a PC using a Serial-to-USB converter cable.
3. Connect the USB PICtail Plus Daughter Board to the horizontal edge connector (J9) of the Explorer 16 Development Board.

For the pic32mz_ef_sk_int_dyn configuration:

Connect the USB to the UART connector (J11) on the PIC32MZ EF Starter Kit to a PC using a USB micro cable.

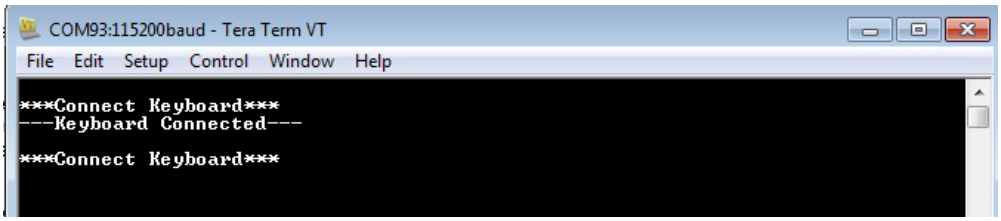
Running the Demonstration

Provides instructions on how to build and run the USB HID Basic Keyboard demonstration.

Description

1. Open the project in MPLAB X IDE and select the project configuration.
2. Build the code and program the device.
3. Launch a terminal emulator, such as Tera Term, and select the appropriate COM port and set the serial port settings to 115200-N-1.
4. If a USB keyboard is not connected to the PIC32 USB Host, the terminal emulator window will show the *Connect Keyboard* prompt.
5. Attach a USB keyboard to the Host connector of the target hardware. The message, *Keyboard Connected*, will appear in the terminal emulator window.

6. Begin typing on the keyboard and the appropriate keys should be displayed on the serial terminal. Subsequent press and release of modifier keys (i.e., CAPS LOCK, NUM LOCK, etc.) will result in the appropriate keyboard LEDs to turning ON and OFF.
7. Disconnecting the keyboard will result in the message, *Connect Keyboard*.



hid_basic_mouse_usart

This topic demonstrates USB Host support for a USB HID Mouse.

Description

This application demonstrates the use of the USB HID Host Client Driver to enumerate and operate a HID mouse device. The application uses the USB Host layer, HID Client driver and HID Mouse Usage driver to enumerate USB mouse and decode mouse-generated data.

Mouse-specific movements events are demonstrated by displaying relative coordinate changes using a serial terminal emulator on a personal computer. Mouse button clicks are indicated by LEDs.

Building the Application

This section does the following:

- Identifies the MPLAB X IDE project name and location.
- Lists and describes the available configurations for the USB HID Basic Mouse USART demonstration.

Description

To build this project, you must open the `hid_basic_mouse_usart.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/usb/host/hid_basic_mouse_usart`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`hid_basic_mouse_usart.X`	`<install-dir>/apps/usb/host/hid_basic_mouse_usart/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mz_ef_sk_meb2	pic32mz_ef_sk+meb2	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit attached to Multimedia Expansion Board II (MEB II) board.

Configuring the Hardware

This section describes how to configure the supported hardware.

Description

1. Ensure that the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit is securely fastened into the MEB II expansion board.
2. Connect the USB to the UART connector (J11) on the PIC32MZ EF Starter Kit to a PC using a USB micro cable.

Note: No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

This section provides instructions about how to build and run the USB HID Mouse USART demonstration.

Description

1. Open the project in MPLAB X IDE and select the project configuration.
2. Build the code and program the device.
3. Launch a terminal emulator, such as Tera Term. Select the appropriate COM port and set the serial port settings to 115200-N-1.
   - If a USB mouse is not connected to the Host connector by using J5 on the PIC32 MZ EF Starter Kit, the serial terminal emulator window will show the "Connect Mouse" prompt.
4. Attach a USB mouse to the Host connector of the target hardware. The message, "Mouse Connected", will display in the serial terminal emulator window.
5. Begin moving the mouse and the appropriate relative coordinate changes for X,Y, and Z axes should be displayed in the serial terminal window.
6. Click the mouse button to toggle LEDs on the MEB II board as shown in the following table.

Mouse Click	MEB II LED
Left	D3
Right	D4
Middle	D5
Lower Left	D6
Lower Right	D7

- Disconnecting the mouse will result in the message, "Connect Mouse", to reappear on the serial console.



hub_cdc_hid

Demonstrates the enumeration of a HID mouse and CDC emulator device via an external hub.

Description

This application demonstrates the capability of the USB Host Stack to access and manage multiple USB Devices through a Hub. The demonstration application enumerates a HID mouse and CDC emulator device via an external hub. The host will demonstrate the communication from the CDC emulator device and the HID mouse.

Building the Application

This topic identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the USB Host HUB CDC HID Demonstration.

Description

To build this project, you must open the `hub_cdc_hid.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/usb/host/hub_cdc_hid`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`hub_cdc_hid.X`	`<install-dir>/apps/usb/host/hub_cdc_hid/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx_usb_sk2_int_dyn	pic32mx_usb_sk2	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32 USB Starter Kit II configured for Interrupt mode and dynamic operation.
pic32mz_ef_sk_int_dyn	pic32mz_ef_sk	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32MZ EF Starter Kit configured for Interrupt mode and dynamic operation.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 USB Starter Kit II

JP2 should be in place.

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the USB Host HUB CDC HID demonstration.

Description

This application demonstrates the capability of the USB Host Stack to access and manage multiple USB Devices through a Hub. The demonstration application enumerates a HID mouse and CDC emulator device via an external hub. The host will demonstrate the communication from the CDC emulator device and the HID mouse.

1. Open the project in MPLAB X IDE and select the desired project configuration.
2. Build the code and program the device.
3. Connect a hub to the Type A Host connector on the desired board.
4. Connect a mouse to a spare port on the hub.
5. Connect the CDC emulator device to another spare port on the hub.
6. Click the mouse to toggle LEDs on the starter kit.
7. On the personal computer, open a terminal emulator. At the prompt, (LED:), enter 1, 2, or 3 to toggle the LEDs on the starter kit.

hub_msd

This application demonstrates the capability of the USB Host stack to support multiple MSD device through a hub.

Description

This application demonstrates the use of the Hub Driver and the MSD Host Client Driver, with File System, to support multiple MSD devices and Hub. The demonstration application copies a file from one pen driver into another pen drive.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the USB Host Hub MSD Demonstration.

Description

To build this project, you must open the `hub_msd.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/usb/host/hub_msd`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`hub_msd.X`	`<install-dir>/apps/usb/host/hub_msd/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mx_usb_sk2_int_dyn	pic32mx_usb_sk2	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32 USB Starter Kit II configured for Interrupt mode and dynamic operation.
pic32mz_ef_sk_int_dyn	pic32mz_ef_sk	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit configured for Interrupt mode and dynamic operation.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 USB Starter Kit II

JP2 should be in place.

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the USB Host Hub MSD demonstration.

Description

This application demonstrates the capability of the USB Host Stack to access and manage multiple USB Devices through a Hub. The demonstration application copies a file from one USB pen drive (i.e., a USB Flash storage device) to another USB pen drive, where these pen drives are attached to a hub.

Note: The demonstration will search for a file named `file.txt` on any of the connected pen drives. Such a file should be created on one of the pen drives through any suitable method.

1. Open the project in MPLAB X IDE and select the desired project configuration.
2. Build the code and program the device.
3. Connect a hub to the Type A Host connector on the desired board.
4. Connect a USB Pen drive containing an arbitrary file named `file.txt` to a spare port on the hub.
5. Connect another USB pen drive to another spare port on the hub.
6. The application will copy the file `file.txt` from the drive containing this file to the other drive. The copied file will be renamed as `newfile.txt`. LED 2 on the demonstration board will illuminate to indicate completion of the file transfer.
7. Disconnect the drives and confirm demonstration success by inserting them into a personal computer and verifying the file transfer completed as expected.

The demonstration application will always be in state where it waits for two pen drives to be connected to the hub and at least one of these pen drives contains a file named `file.txt`.

msd_basic

This application demonstrates the use of the MSD Host Class Driver to write a file to USB Flash Drive.

Description

This application demonstrates the use of the MSD Host Class Driver to write a file to a USB Flash drive. The application uses the USB Host layer , MSD class driver and the MPLAB Harmony File System Framework to enumerate a USB Flash drive and to write a file to it.

Building the Application

This section identifies the MPLAB X IDE project name and location, and lists and describes the available configurations for the USB MSD Host Class Driver Demonstration.

Description

To build this project, you must open the `msd_basic.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/usb/host/msd_basic`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`msd_basic.X`	`<install-dir>/apps/usb/host/msd_basic/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
chipkit_wf32	chipkit_wf32	Demonstration running on the chipKIT WF32 Development Board.
chipkit_wifire	chipkit_wifire	Demonstration running on the chipKIT Wi-FIRE Development Board.
pic32mx_usb_sk2_int_dyn	pic32mx_usb_sk2	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32 USB Starter Kit II configured for Interrupt mode and dynamic operation.
pic32mx_usb_sk3_int_dyn	pic32mx_usb_sk3	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32 USB Starter Kit III with the PIC32MX470F512L microcontroller configured for Interrupt mode and dynamic operation.
pic32mz_da_sk_intddr_int_dyn	pic32mz_da_sk_intddr	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit configured for Interrupt mode and dynamic operation.
pic32mz_ef_sk_int_dyn	pic32mz_ef_sk	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit configured for Interrupt mode and dynamic operation.
pic32mx_xlp_sk_int_dyn	pic32mx_xlp_sk	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32MX XLP Starter Kit configured for Interrupt mode and dynamic operation.
pic32wk_sk_int_dyn	pic32wk_gbp_gpd_sk+module	Select this MPLAB X IDE project configuration to run the demonstration application to run on the PIC32WK Wi-Fi Starter Kit, with the WM32 Wi-Fi module. The USB Stack will be configured for Interrupt mode operation and the USB Driver will be configured for Dynamic operation mode.
pic32mx470_curiosity	pic32mx470_curiosity	Select this MPLAB X IDE project configuration to run the demonstration application to run on the PIC32MX470 Curiosity Development Board, with the PIC32MX470F512H microcontroller. The USB Stack will be configured for Interrupt mode operation and the USB Driver will be configured for Dynamic operation mode.
pic32mz_ef_curiosity	pic32mz_ef_curiosity	Select this MPLAB X IDE project configuration to run the demonstration application to run on the PIC32MZ EF Curiosity Development Board, with the PIC32MZ2048EFM100 microcontroller. The USB Stack will be configured for Interrupt mode operation and the USB Driver will be configured for Dynamic operation mode.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32 USB Starter Kit II

JP2 should be in place.

PIC32 USB Starter Kit III

JP1 should be in place.

PIC32MZ Embedded Graphics with Internal DRAM (DA) Starter Kit

No hardware related configuration or jumper setting changes are necessary.

PIC32MZ EC Starter Kit

JP1 should be in place and the Ethernet plug-in board should be removed.

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

chipKIT WF32 Wi-Fi Development Board

No hardware related configuration or jumper setting changes are necessary.

chipKIT Wi-FIRE Development Board

No hardware related configuration or jumper setting changes are necessary.

PIC32MX470 Curiosity Development Board

- Ensure that a jumper is placed at 4-3 on J8, to select supply from debug USB connector.
- Power the PIC32MX470 Curiosity Development Board from a Host PC through a Type-A male to mini-B USB cable connected to Mini-B port (J3)
- Place a jumper on J13 to drive VBUS in Host mode
- Plug in a USB peripheral with a micro-A USB connector, or use a micro USB OTG to USB adapter.



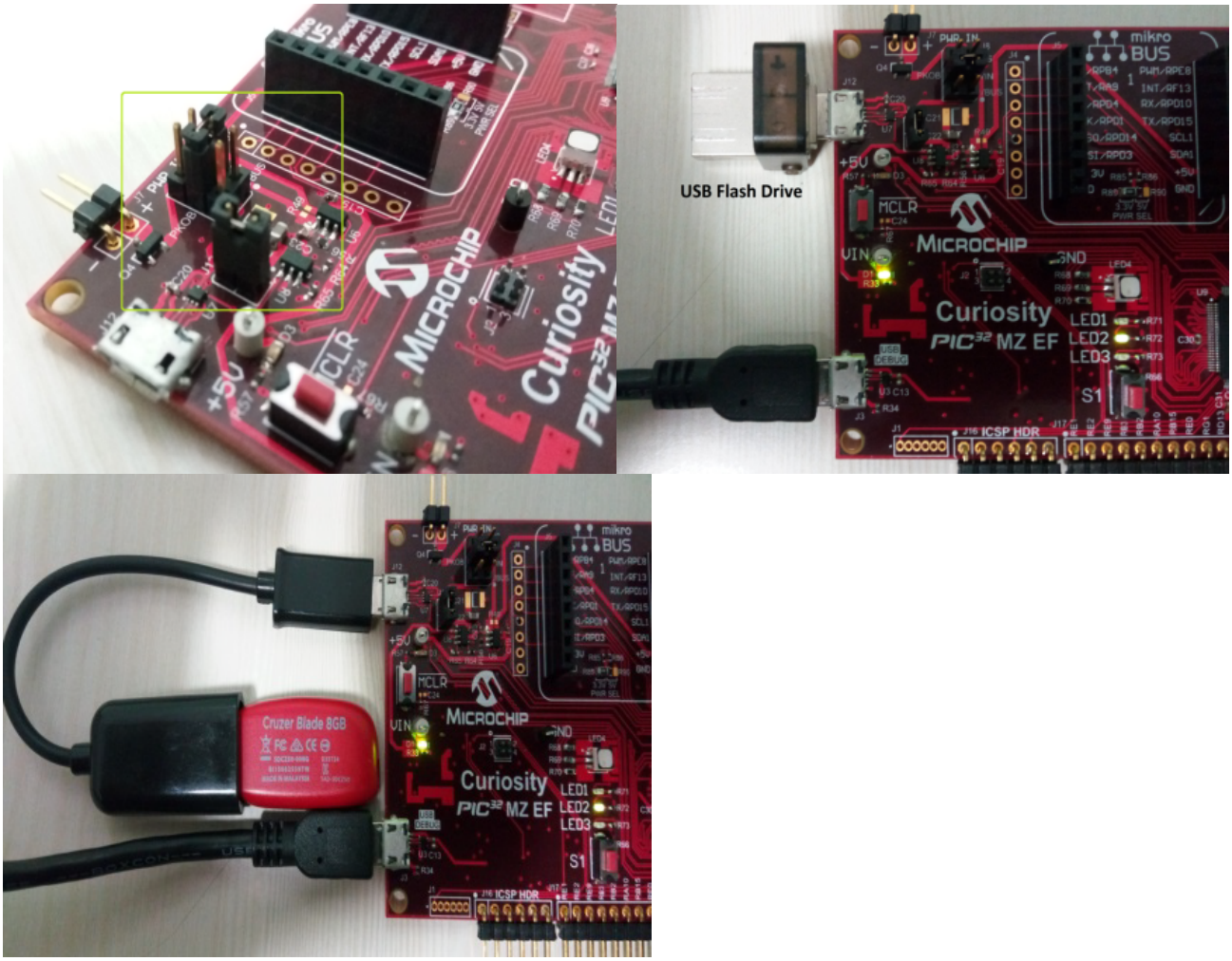


PIC32MZ EF Curiosity Development Board

- Ensure that a jumper is placed at 4-3 on J8, to select supply from debug USB connector.
- Power the PIC32MZ EF Curiosity Development Board from a Host PC through a Type-A male to micro-B USB cable connected to Micro-B port

(J3).

- Place a jumper on J13 to drive VBUS in Host mode.
- Plug in a USB peripheral with a micro-A USB connector, or use a micro USB OTG to USB adapter.



PIC32WK Wi-Fi Starter Kit

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions on how to build and run the USB Host MSD Basic demonstration.

Description

This application demonstrates the use of the MSD Host Class Driver to write a file to USB Flash drive. The application uses the USB Host layer, MSD class driver and the MPLAB Harmony File System Framework to enumerate a USB Flash drive and to write a file to it.

Prior to using this demonstration, it is recommended to review the MPLAB Harmony Release Notes for any known issues. A PDF copy of the release notes is provided in the `<install-dir>/doc` folder of your installation.

1. Open the project in MPLAB X IDE and select the desired project configuration.
2. Build the code and program the device.
3. With the code running, attach a USB Flash drive to the Host connector on the desired starter kit.
4. The demonstration application will then create a file named `file.txt`. It will then write the text "Hello World" to this file, and then close the file.
5. The demonstration will then move to Idle mode, which is indicated when the LED on the starter kit illuminates. On PIC32M-based starter kits the LED is LED2.
6. The USB Flash drive can then be attached to a USB Host personal computer to verify the demonstration application operation.
7. Steps 3 through 6 can be repeated.
8. If the USB Flash drive already contains a file with the name `file.txt`, the demonstration application will append the text "Hello World" to the end of the file contents.
9. The LED on the starter kit illuminates if the file creation or write failed. On PIC32M-based starter kits, the LED is LED1.

Multiple USB Controller

This section describes the demonstrations that make use of multiple USB controllers on certain PIC32 microcontrollers.

cdc_com_port_dual

This application demonstrates dual USB Device operation on a PIC32 microcontroller with two USB Controllers.

Description

This application demonstrates dual USB Device operation on a PIC32 microcontroller with Two USB Controllers. In this demonstration both of the USB controllers act as CDC devices.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Multiple USB CDC Device Dual COM Port Demonstration.

Description

To build this project, you must open the `cdc_com_port_dual.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/usb/multi_usb/cdc_com_port_dual`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`cdc_com_port_dual.X`	`<install-dir>/apps/usb/multi_usb/cdc_com_port_dual/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mk_gp_db_int_dyn	pic32mk_gp_db	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32MK Evaluation Kit configured for interrupt mode and dynamic operation.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MK General Purpose (GP) Development Board

Switch S4 should be set to the Device position.

Running the Demonstration

This section provides instructions on how to build and run the USB Multiple Controller CDC Com Port Dual demonstration.

Description

This application demonstrates dual USB Device operation on a PIC32 microcontroller with two USB Controllers. The MPLAB Harmony USB Stack is capable of handling multiple USB controllers. In this demonstration, both of the USB controllers act as CDC devices.

This demonstration allows the each controller on the PIC32 to appear like a serial (COM) port to the host. Do the following to run this demonstration:

1. First compile and program the target device. Refer to Building the Application for details.
2. Attach both USB connectors J15 and J13 to the host.
3. Refer to the Running the Demonstration section of the USB Device cdc_com_port_single demonstration for details on exercising the CDC device features.

msd_dual

This application demonstrates the capability of a PIC32 microcontroller and MPLAB Harmony USB Host stack to work with two USB Controllers in an application.

Description

This application demonstrates the capability of a PIC32 microcontroller and the MPLAB Harmony USB Host stack to work with two USB Controllers in an application. The MPLAB Harmony USB Stack is capable of handling multiple USB controllers.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the Dual MSD demonstration.

Description

To build this project, you must open the `msd_dual.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/usb/multi_usb/dual_msd`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`msd_dual.X`	`<install-dir>/apps/usb/multi_usb/msd_dual/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mk_gp_deb_int_dyn	pic32mk_gp_db	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32MK Evaluation Kit configured for interrupt mode and dynamic operation.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MK General Purpose (GP) Development Board

- Switch S4 should be set to Host position
- Jumper J28 must be installed
- USB Connector J12 must be connected to a USB Host for powering the board
- USB Flash drive should be attached to Connector J15 and J14 after programming the microcontroller

Running the Demonstration

This section provides instructions on how to build and run the USB Multiple Controller Dual MSD demonstration.

Description

This application demonstrates the capability of a PIC32 microcontroller and the MPLAB Harmony USB Host stack to work with two USB Controllers in an application. The MPLAB Harmony USB Stack is capable of handling multiple USB controllers. The application uses the USB Host_layer, MSD class driver, and the MPLAB Harmony File System Framework to enumerate a USB Flash drive and to write a file to it.

Prior to using this demonstration, it is recommended to review the MPLAB Harmony Release Notes for any known issues. A PDF copy of the release notes is provided in the `<install-dir>/doc` folder of your installation. Do the following to run this demonstration:

1. Open the project in MPLAB X IDE and select the desired project configuration.
2. Build the code and program the device.
3. With the code running, attach a USB Flash drive with a file “file.txt” in it to one of the Host connector on the board.

4. Connect another USB Flash drive to other Host connector on the board. Ensure this flash drive does not contain any file named `newfile.txt`.
5. The application will copy the file file.txt from the drive containing this file to the other drive. The copied file will be renamed as `newfile.txt`. LED2 on the demonstration board will illuminate to indicate completion of the file transfer.
6. Disconnect the drives and confirm demonstration success by inserting them into a personal computer and verifying the file transfer completed as expected.

Dual Role

This section describes the USB Dual Role Demonstrations. These demonstrations project demonstrate operation of the USB Host and the USB Device stack in the same project.

host_msd_device_hid

This application demonstrates role switching between USB Host MSD Stack and USB Device HID function. The role switch is trigger by a switch press.

Description

This application demonstrates role switching between USB Host MSD Stack and USB Device HID function. The role switch is trigger by a switch press. In the USB Host mode, the application performs read and write operations to a USB pen drive. In the USB Device mode, the application emulates a HID mouse.

Building the Application

This section identifies the MPLAB X IDE project name and location and lists and describes the available configurations for the USB Host MSD and USB HID Mouse Device Dual Role application.

Description

To build this project, you must open the `host_msd_device_hid.X` project in MPLAB X IDE, and then select the desired configuration.

The following tables list and describe the project and supported configurations. The parent folder for these files is `<install-dir>/apps/usb/dual_role/host_msd_device_hid`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

Project Name	Location
`host_msd_device_hid.X`	`<install-dir>/apps/usb/dual_role/host_msd_device_hid/firmware`

MPLAB X IDE Project Configurations

This table lists and describes the supported configurations of the demonstration, which are located within `./firmware/src/system_config`.

Project Configuration Name	BSP Used	Description
pic32mz_ef_sk_int_dyn	pic32mz_ef_sk	Select this MPLAB X IDE project configuration to run the demonstration on the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit configured for Interrupt mode and dynamic operation.

Configuring the Hardware

Describes how to configure the supported hardware.

Description

PIC32MZ EF Starter Kit

No hardware related configuration or jumper setting changes are necessary.

Running the Demonstration

Provides instructions no how to build and run the USB Host MSD and USB HID Mouse Device Dual Role application.

Description

This application demonstrates the Dual Role capability of the MPLAB Harmony USB Stack. The application project includes both, the USB Host and Device Stacks. Both the stacks are initialized during application initialization. During operation, the application polls the switch SW2 on the starter kit to trigger a USB role switch. Note that the application cannot simultaneously operate as a host and device. The one USB role is exclusive of the other.

Prior to using this demonstration, it is recommended to review the MPLAB Harmony Release Notes for any known issues. A PDF copy of the release notes is provided in the `<install-dir>/doc` folder of your installation.

1. Open the project and in MPLAB X IDE and select the desired project operation
2. Build the code and program the device. The application initially will not operate in any USB role.
3. Press SW2 on the starter kit. This places the application in a USB Device mode.
4. Connect a USB cable between micro USB connector (J4) on the starter kit and a PC USB host. The application will emulate a USB HID mouse function. The cursor on the PC will rotate. Pressing SW1 will enable and disable the cursor movements. Exercise device plug-n-play operation to confirm USB Device operation
5. Now try switching the USB role. Disconnect the USB cable between micro USB connector (J4) on the starter kit and a PC USB host. Press SW2 on the starter kit.
6. The application now will be in USB Host role. Connect a USB pen drive to the Type-A USB Host connector (J5) on the starter kit. The application will create a file (`file.txt`) on the pen drive. The completion of the operation is indicated by LED2 on the starter kit. Disconnect the pen driver and connect it to a PC to verify the contents of the file.
7. Repeat steps 3 through 6 to exercise the role switching capability.

Volume IV: MPLAB Harmony Development

This volume provides information about how to develop MPLAB Harmony-compatible libraries and applications and how to best distribute and integrate them into an existing installation. Information on porting and updating is also included.

Description



Information on the tools, techniques, and knowledge required to develop, distribute, and integrate MPLAB Harmony-compatible libraries and applications into existing installations, as well as updating existing MPLAB Harmony projects to a newer version of MPLAB Harmony is provided throughout the help documentation. This section also introduces key topics and provides a starting point for new developers.

MPLAB Harmony Driver Development Guide

This guide provides information on developing MPLAB Harmony device drivers.

Introduction

Describes how to develop MPLAB Harmony device drivers.

Description

This development guide describes how to develop device drivers for MPLAB Harmony. MPLAB Harmony device drivers, or simply "drivers", typically utilize MPLAB Harmony Peripheral Libraries (PLIBS) to access and control built-in peripheral hardware. A driver is the *glue* logic between an application and the peripheral library. The peripheral library provides a low-level interface to a specific peripheral, but use of that interface would place a lot of responsibility on the application for maintaining the state of the device and ensuring that other (usually unrelated) modules do not interfere with its operation. Instead, these functions become the responsibility of the driver, freeing the application from managing devices and considerably simplifying the interface to the peripheral.

Drivers provide simple C-language interfaces (see **Note**). A driver's interface should be highly abstracted and provide file system style "*open*" and "*close*", and "*read*" and "*write*" functions (for data transfer peripherals) that allow applications (or other client modules) to easily interact with them in a consistent manner. Most drivers also provide additional functions that are unique to a particular type of driver or peripheral, but are independent of the details of how that peripheral is implemented on any specific hardware or how many instances of that driver or peripheral exist in a given system.

Note: MPLAB Harmony has not been tested with C++; therefore, support for this programming language is not supported.

Drivers can also indirectly support external peripheral hardware, which has no peripheral library, by accessing another driver. For example, a SD Card driver may use a SPI driver to access an external Flash device. Or, a driver may be completely abstracted, utilizing no peripheral hardware at all and simply providing some device-like service to higher layers of software.

Regardless of the type or lack of hardware a MPLAB Harmony driver manages, it has the following fundamental responsibilities:

- Providing a common system-level interface to a peripheral
- Providing a highly-abstracted file system style client interface to a peripheral
- Controlling access to a peripheral
- Managing the state of a peripheral

Information on how a driver can fulfill these responsibilities and other key concepts and requirements for design and development of MPLAB Harmony device drivers is provided in the following sections.

Using This Document

Describes how to best use this document, depending upon your experience level and familiarity with MPLAB Harmony.

Description

Experienced MPLAB Harmony Driver Developer

If you are an experienced MPLAB Harmony driver developer, you can skip to the Checklist and Information section at the end of this document for a list of the tasks necessary to develop a MPLAB Harmony driver and a handy reminder of the file and folder naming and organization conventions.

Experienced Embedded Software Developer

If you are an experienced embedded software developer who is familiar with modular design and state-machine based development, but are not familiar with MPLAB Harmony driver development, you can briefly scan the Key Design Concepts section and jump to the System Interface and Client Interface sections. Also, be sure to review the Interrupt and Thread Safety section for examples of recommended methods to use in MPLAB Harmony drivers and read the remaining sections in detail.

MPLAB Harmony User Who Has Yet to Develop a Driver

If you are a MPLAB Harmony user who would like to start developing MPLAB Harmony drivers, please read this entire document.

New to MPLAB Harmony

If you are new to MPLAB Harmony, please read the What is MPLAB Harmony? section first.

System Interface

Describes the system interface requirements for driver design.

Description

In MPLAB Harmony, almost everything running in the system is considered a module. A module usually has an internal state machine that runs when the system runs. Exactly what it means to *run* in the system depends on the configuration that is selected. In the simplest polled bare metal configuration a MPLAB Harmony system runs modules in a single polled super loop, implemented in the main function, as shown in the following example.

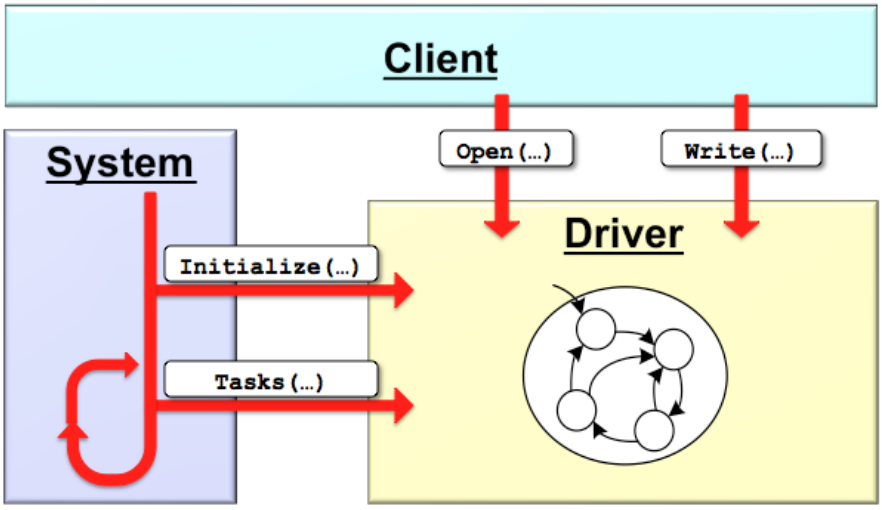
The Main Function

```
int main ( void )
{
    SYS_Initialize(NULL);

    while( true )
    {
        SYS_Tasks();

    }

    return(EXIT_FAILURE);
}
```

The SYS_Initialize and SYS_Tasks functions are implemented in system configuration-specific files, normally generated by the MPLAB Harmony Configurator (MHC). The SYS_Initialize function calls the initialization functions of all modules in the system. The initialization function of a module must (at a minimum) initialize the state machine of that module so that its tasks function can be safely called. Then, the SYS_Tasks function calls the tasks functions of all modules in the system. This continues indefinitely, keeping all of the modules in the system running. Effectively, the main loop and the SYS_Initialize and SYS_Tasks functions implement a simple system kernel scheduler as a round-robin polled super loop.

The signatures of these functions are always consistent, using the same parameters and return values. Only the names change from module to module. This provides a consistent execution model for polled, interrupt-driven, and RTOS-based environments and supports the ability to implement system executives, power mangers, and test harnesses that initialize, deinitialize and maintain multiple modules.

In any module, the system interface should be thought of as conceptually separate from the application or client Interface (what is commonly referred to as the API), as shown in the following diagram.



These two interfaces serve completely separate purposes. The system interface allows system kernel or scheduler to initialize and run the module (as described previously) and the client interface allows the application or other client module to interact with the driver or module. The system code does not normally interact with the client-level API of the module and the application or any other client should not call the system interface functions or have access to the object handle (explained in the following sections). Once the system has been initialized, client or application code can call the driver's client interface functions (such as open, read, and write functions). The client interface is described in detail in the Client Interface section.

Once a driver module has been initialized, its state machine must be placed into its initial state and it can be considered ready to use. Although its internal state machine may have still several initial transitions to complete, its other system and client interface routines may be called.

The initial power state of the module can be defined by build-time configuration parameters or be dependent upon the hardware initialization data passed into the initialize operation. Driver modules can be initialized in a power-on, full running state or a power-off or low-power state and reinitialized later under system control to a power-on state when they are needed. Drivers may also be reinitialized to refresh the hardware state.

A driver can be deinitialized if it does not need to be used any longer, after which none of its other interface functions may be called without first calling the initialize function again. The initialize function must not be called more than once without first calling the deinitialize function. The reinitialize operation can be called any time the module is in a ready state.

A driver's system interface consists of the following functions, where `<module>` matches the module abbreviation for the driver or peripheral.

Driver's System Interface Functions

Function	Description
DRV_<module>_Initialize	Place the driver in its initial state.
DRV_<module>_Tasks	Manage the running state of the driver.
DRV_<module>_Reinitialize	Change a running driver's initial parameters.
DRV_<module>_Deinitialize	Disable the driver and stop it from running.
DRV_<module>_Status	Provide the current status of the driver

A driver can have only one of each of these functions, except for the DRV_<module>_Tasks function. It is possible for a driver to have multiple different tasks functions, each maintaining a different state machine within the driver. The naming format for device-driver system module routines adds the DRV_ prefix to identify that the function belongs to a device driver module and to be consistent with the driver's other interface routines.

Full descriptions of the driver-module interface routines that implement these operations are provided in the following sections.

Module Initialization

Describes the details of the module initialization function.

Description

The function signature of a module's initialize function is defined by a pointer data type that is defined by the system module interface header, in the <install-dir>/framework/system/common/sys_module.h file, as shown in the following example.

Example: Module Initialization Function Signature

```
/*********************************************************************
  System Module Initialization Function Pointer

  Function Pointer:
    SYS_MODULE_OBJ (* SYS_MODULE_INITIALIZE_ROUTINE) (
                        const SYS_MODULE_INDEX index,
                        const SYS_MODULE_INIT * const init )

  Description:
    This data type is a pointer to a function that initializes a system module
    (driver, library, or system-maintained application).

  Preconditions:
    The low-level processor and board initialization must be completed before the
    system can call the initialization functions for any modules.

  Parameters:
    index           - Zero-based index of the module instance to be initialized.

    init            - Pointer to the data structure containing any data
                      necessary to initialize the module. This pointer may
                      be null if no data is required.

  Returns:
    A handle to the instance of the module that was initialized.  This handle is
    a necessary parameter to all of the other system level routines for that
    module.

  Remarks:
    This function will normally only be called once during system initialization.
*/

typedef SYS_MODULE_OBJ (* SYS_MODULE_INITIALIZE_ROUTINE) (
                            const SYS_MODULE_INDEX index,
                            const SYS_MODULE_INIT * const init );
```

An implementation of a module's initialization function might look like the following example.

Example: Sample Module Initialization Function

```
SYS_MODULE_OBJ SAMPLE_Initialize ( const SYS_MODULE_INDEX index,
                                   const SYS_MODULE_INIT * const init )
{
    SAMPLE_MODULE_DATA      *pObj  = (SAMPLE_MODULE_DATA *)&gObj[index];
    SAMPLE_MODULE_INIT_DATA *pInit = (SAMPLE_MODULE_INIT_DATA *)init;
```

```
    /* Initialize module object. */

    pObj->state                = SAMPLE_STATE_INITALIZE;
    pObj->status               = SYS_STATUS_BUSY;
    pObj->dataNewIsValid       = false;
    pObj->dataProcessedIsValid = false;

    if ( null != init )
    {
        pObj->dataNew          = pInit->dataSome;
        pObj->dataNewIsValid   = true;
    }

    return (SYS_MODULE_OBJ)pObj;
}
```

Ignoring the index parameter and the return value for now, you can see from the sample code above that a module's initialization function accepts a pointer to an initial data (`init`) structure, casts the pointer to a new type, and then stores data from the `init` structure into an internal module object structure. Because of the requirement for consistency in function signature, the initialize function for any module must use a common data type for the `init` data pointer parameter. This data type, shown in the following example, is also defined in the `sys_module.h` header file.

SYS_MODULE_INIT Structure

```
typedef union
{
    uint8_t         value;

    struct
    {
        uint8_t     powerState  : 4;

        uint8_t     reserved    : 4;
    }sys;

} SYS_MODULE_INIT;
```

The SYS_MODULE_INIT structure allows the system to pass in a pass in a parameter (the `powerState` member) to identify the requested initial power state of the module. The following values and labels are predefined (in `system_module.h`) to provide a common set of power states for all modules.

Predefined Power States

Volume	Label	Description
0	SYS_MODULE_POWER_OFF	Module Power-off state code.
1	SYS_MODULE_POWER_SLEEP	Module Sleep state code.
2	SYS_MODULE_POWER_IDLE_STOP	Module Idle-Stop state code.
3	SYS_MODULE_POWER_IDLE_RUN	Module Idle-Run state code.
4 through 14	<Module-specific Definition>	Module-specific meaning.
15	SYS_MODULE_POWER_RUN_FULL	Module Run-Full state code.

Note:

Note: Refer to the Help documentation for each individual label for a more detailed description of what each value indicates. See Framework Help > System Service Libraries Help > System Service Overview.

Using a pointer to this structure as the `init` parameter allows system code to treat all modules in a consistent way. However, any specific module or implementation of a module may have its own unique `init` data requirements and may define its own unique structure type. Unfortunately, the C language does not provide a syntactical mechanism for managing this sort of polymorphism. Polymorphism is an object oriented programming (OOP) concept that allows different types (or classes) of data (or other objects) to support multiple forms. To achieve this flexibility in the C language, the module must cast the pointer to an internally defined data type. But, it is reasonable to think of the SYS_MODULE_INIT structure as a base class, extended as necessary by any individual module class or implementation to contain the specific additional initialization data it requires. While this is a slight abuse of the C language, it works as required as long as the first member of any module's extended `init` structure is a SYS_MODULE_INIT structure, which is (of course) a requirement of any MPLAB Harmony module.

Whether or not a module requires any initialization data, the primary purpose of its initialization function is to place the module's state machine into its initial state. In the sample module initialization function above, the following line of code does this.

```
pObj->state = SAMPLE_STATE_INITALIZE;
```

In this line, state is simply a structure member variable used to keep track of the current state of the module's state machine and the SAMPLE_STATE_INITIALIZE value is its initial state. This variable is contained within in a structure and accessed using the `pObj` pointer along with all other variables that are specific to a single instance of the driver module. This allows an instance of a driver to be referenced by a single

driver *object* pointer, which is cast to the SYS_MODULE_OBJ data type and returned from the driver's initialize function to be used by the system to identify an instance of the driver and passed into the driver's other system interface functions to access its instance-specific data.

Note: Any module that has a state machine must implement an initialize function.

Module Tasks

Describes the details of the module "Tasks" function.

Description

The function signature of a module's tasks function is defined by a pointer data type that is defined by the system module interface header, in the `<install-dir>/framework/system/common/sys_module.h` file, as shown in the following example.

Example: Module Tasks Function Signature

```
// *****************************************************************************
/* System Module Tasks Routine Pointer

  Function:
    void (* SYS_MODULE_TASKS_ROUTINE) ( SYS_MODULE_OBJ object )

  Summary:
    Pointer to a routine that performs the tasks necessary to maintain a state
    machine in a module.

  Description:
    This data type is a pointer to a routine that performs the tasks necessary
    to maintain a state machine in a module (driver, library, or application).

  Preconditions:
    The low-level board initialization must have been completed and the module's
    initialization function must have been called before the system can call the
    tasks routine for any module.

  Parameters:
    object          - Handle to the module instance

  Returns:
    None.

  Remarks:
    If the module is interrupt driven, the system will call this routine from
    an interrupt context.
*/

typedef void (* SYS_MODULE_TASKS_ROUTINE) ( SYS_MODULE_OBJ object );
```

An implementation of a module's tasks function might look like the following example.

Example: Sample Module Tasks Function

```
void SAMPLE_Tasks( SYS_MODULE_OBJ object )
{
    SAMPLE_MODULE_DATA     *pObj  = (SAMPLE_MODULE_DATA *)object;

    // Sample Module State Machine
    switch ( pObj->state )
    {
        case SAMPLE_STATE_INITIALIZE:
        {
            pObj->status = SYS_STATUS_READY;
            pObj->state  = SAMPLE_STATE_PROCESS;
            break;
        }

        case SAMPLE_STATE_PROCESS:
        {
            if (pObj->dataNewIsValid && !pObj->dataProcessedIsValid)
            {
                pObj->dataProcessed         = pObj->dataNew;
```

```
                pObj->dataNewIsValid        = false;
                pObj->dataProcessedIsValid  = true;
            }
            break;
        }

        default:
        {
            pObj->status = SYS_STATUS_ERROR;
            break;
        }
    }

    return;
}
```

As shown by the previous sample code, the object handle returned from the initialize function is passed into the tasks function and cast back into a pointer to the module's internal data structure type. This allows the function to access the instance-specific data it contains. The module may then utilize the data to determine what its next appropriate action may be. This is often implemented using a state variable (`pObj->state`) and a switch statement with different states defined by an enumeration.

Each case in the switch statement corresponds to a different state transition that the module's state machine can make. The module's initialize function placed the state machine in its initial state (SAMPLE_STATE_INITIALIZE). The initialize state transitions to the next state (SAMPLE_STATE_PROCESS) automatically and, as a side effect, changes the module's status to ready (SYS_STATUS_READY). Therefore, the next time the tasks function is called, it is in the process state and is ready to process data.

In the process state, the sample module checks to see if it has any valid new data (using the Boolean flag `pObj->dataNewIsValid`) and if it is able to process that new data. It can only process one data item at a time. So, if it does not currently have any processed data (as indicated by the `pObj->dataProcessedIsValid` Boolean flag being false), it can then it processes the new data item and update the Boolean flags.

In this sample module, the initialize state transition is not really necessary and could have been eliminated by appropriately setting the module's status variable in the initialize function. Also, the default case is unnecessary and should never occur, although it can be used for error handling. The only active state transition occurs in the process state and it never transitions out of that state to a new one.

It is important to notice that the state machine checks status flags before taking any action. This prevents it from taking action until some change has occurred, which allows it to be polled from the main super loop. The flags in the sample module are modified by the module's API functions, when a client calls them. However, these flags could just as easily have been hardware interrupt flags. If that were the case, this state machine could just as easily be called from an ISR, which is exactly how it would be called in an interrupt-driven configuration. In that configuration, testing the interrupt flag could be redundant. But, doing so is safer as it avoids taking inappropriate actions if spurious interrupts occur or if the interrupt vector is shared between multiple interrupt sources.

Note: Use of the switch-case technique is not a requirement, but it is a convenient way to explain the purpose of the tasks function. Simpler and more sophisticated techniques are possible. Any method that correctly implements the necessary state machine logic is acceptable.

For a complete working example, refer to the sample module library, located in the `<install-dir>/framework/sample` folder.

A module's tasks function may assume that the initialize function has been called once (and only once) before the tasks function is called. But, it must not associate any meaning to when or how often the tasks function is called. It cannot assume that it is called before or after any other tasks has been called and it cannot assume it has been called once for every time any other tasks function has been called. It is entirely possible (particularly in a RTOS-based system) that the module's tasks function is running at a different priority level and is called more frequently or less frequently than other tasks functions in the system. It is also possible that it may be called from within an ISR, but only if it was designed to support an ISR. It is possible to design a tasks function that has no associated interrupt. Every time it is called, a module's tasks function must use current state or status to determine the appropriate state transition to make or if it needs to make any transition at all.

Note: Any module that has a state machine must implement a tasks function.

Module Status

Describes the details of the module "Status" function.

Description

The function signature of a module's status function is defined by a pointer data type that is defined by the system module interface header, in the `<install-dir>/framework/system/common/sys_module.h` file, as shown in the following example.

Example: Module Status Function Signature

```
// *****************************************************************************
/* System Module Status Routine Pointer

  Function:
    SYS_STATUS (* SYS_MODULE_STATUS_ROUTINE) ( SYS_MODULE_OBJ object )
```

```
  Summary:
    Pointer to a function that gets the current status of a module.

  Description:
    This data type is a pointer to a function that gets the current status of a
    system module (driver, library, or application).

  Preconditions:
    The low-level board initialization must have been completed and the module's
    initialization function must have been called before the system can call the
    status function for any module.

  Parameters:
    object          - Handle to the module instance

  Returns:
    One of the possible status codes from SYS_STATUS

  Remarks:
    A module's status function can be used to determine when any of the
    other system level operations has completed as well as to obtain general
    status of the module.
    If the status function returns SYS_STATUS_BUSY, a previous operation
    has not yet completed. Once the status function returns SYS_STATUS_READY,
    any previous operations have completed.
    The value of SYS_STATUS_ERROR is negative (-1). A module may define
    module-specific error values of less or equal SYS_STATUS_ERROR_EXTENDED
    (-10).
    The status function must NEVER block.
    If the status function returns an error value, the error may be cleared by
    calling the reinitialize function. If that fails, the deinitialize
    function will need to be called, followed by the initialize function to
    return to normal operations.
*/

typedef SYS_STATUS (* SYS_MODULE_STATUS_ROUTINE) (  SYS_MODULE_OBJ object );
```

An implementation of a module's status function might look like the following example.

Example: Sample Module Status Function

```
SYS_STATUS SAMPLE_Status ( SYS_MODULE_OBJ object )
{
    SAMPLE_MODULE_DATA     *pObj  = (SAMPLE_MODULE_DATA *)object;

    return pObj->status;
}
```

In most cases, a module's status function will just return the current value of the status variable in the module-instance data structure referred to by the object handle. However, it is possible to deduce the module instance's current status using more complex logic. But, be aware of potential race conditions that could be caused by checking multiple individual variables and/or hardware status flags.

A module's status function must return a value from the SYS_STATUS enumeration (or an extension of it), as shown in the following example.

SYS_STATUS Enumeration

```
// *****************************************************************************
/* System Module Status

  Summary:
    Identifies the current status/state of a system module

  Description:
    This enumeration identifies the current status/state of a system module
    (driver, library, or application).

  Remarks:
    This enumeration is the return type for the system-level status routine
    defined by each driver, library, or application (for example, DRV_I2C_Status).
*/

typedef enum
{
```

```
    /* Indicates that a non-system defined error has occurred.  The caller
       must call an extended status routine for the module in question to
       identify the error. */
    SYS_STATUS_ERROR_EXTENDED   = -10,

    /* An unspecified error has occurred. */
    SYS_STATUS_ERROR            = -1,

    /* The module has not yet been initialized. */
    SYS_STATUS_UNINITIALIZED    = 0,

    /* An operation is currently in progress. */
    SYS_STATUS_BUSY             = 1,

    /* Any previous operations have completed and the module is ready for
       additional operations. */
    SYS_STATUS_READY            = 2,

    /* Indicates that the module is in a non-system defined ready/run state.
       The caller must call an extended status routine for the module in
       question to identify the state. */
    SYS_STATUS_READY_EXTENDED   = 10

} SYS_STATUS;
```

However, a module may define its own module-specific status enumerations that extend the system status enumeration by equating their values and defining new values in the extended error and ready ranges, as shown in the following example.

Example: Sample Module Specific Status Enumeration

```
typedef enum
{
    SAMPLE_STATUS_ERROR_PARITY     = SYS_STATUS_ERROR_EXTENDED - 2,
    SAMPLE_STATUS_ERROR_UNDERRUN   = SYS_STATUS_ERROR_EXTENDED - 1,
    SAMPLE_STATUS_ERROR_OVERFLOW   = SYS_STATUS_ERROR_EXTENDED,
    SAMPLE_STATUS_ERROR            = SYS_STATUS_ERROR,
    SAMPLE_STATUS_UNINITIALIZED    = SYS_STATUS_UNINITIALIZED,
    SAMPLE_STATUS_BUSY             = SYS_STATUS_BUSY,
    SAMPLE_STATUS_READY            = SYS_STATUS_READY,
    SAMPLE_STATUS_READY_BUS_IDLE   = SYS_STATUS_READY_EXTENDED,
    SAMPLE_STATUS_READY_RECEIVING  = SYS_STATUS_READY_EXTENDED + 1,
    SAMPLE_STATUS_READY_SENDING    = SYS_STATUS_READY_EXTENDED + 2

} SAMPLE_STATUS;
```

This allows the module to utilize its own status labels internally in its implementation code and allows its clients to do the same when using the module's API while still allowing the system to utilize the standard values or to check for error ranges below SYS_STATUS_ERROR or ready ranges above SYS_STATUS_READY.



Note: Any module that has a state machine must implement a status function.

Module Deinitialize

Describes the details of the module "Deinitialize" function.

Description

The function signature of a module's deinitialize function is defined by a pointer data type that is defined by the system module interface header, in the `<install-dir>/framework/system/common/sys_module.h` file, as shown in the following example.

Example: Module Deinitialize Function Signature

```
/*********************************************************************
  Function:
    void (* SYS_MODULE_DEINITIALIZE_ROUTINE) (  SYS_MODULE_OBJ object )

  Summary:
    Pointer to a routine that deinitializes a system module.

  Description:
    This data type is a pointer to a routine that deinitializes a system
    module (driver, library, or application).
```

```
  Preconditions:
    The low-level board initialization must have (and will be) completed
    and the module's initialization function will have been called before
    the system will call the deinitialization function for any module.

  Parameters:
    object -  Handle to the module instance

  Returns:
    None.

  Remarks:
    If the module instance has to be used again, the module's "initialize"
    function must first be called.
*/

typedef void (* SYS_MODULE_DEINITIALIZE_ROUTINE) (  SYS_MODULE_OBJ object );
```

An implementation of a module's deinitialize function might look like the following example.

Example: Sample Module Deinitialize Function

```
void SAMPLE_Deinitialize ( SYS_MODULE_OBJ object )
{
    SAMPLE_MODULE_DATA *pObj  = (SAMPLE_MODULE_DATA *)object;

    pObj->dataNewIsValid        = false;
    pObj->dataProcessedIsValid  = false;
    pObj->status                = SYS_STATUS_UNINITIALIZED;

    return;
}
```

In the previous example, the deinitialize function for the sample module simply clears a few key flags and returns. For a simple module, where its other system and client interface routines take no action and return appropriate values when uninitialized, this may be all that is necessary. Other, more complex modules may need their state machines to go through a deinitialize sequence before the modules can be safely considered deinitialized. Such modules must return SYS_STATUS_BUSY from their status functions until the sequence has completed to signal to the system that it must continue calling the module's tasks function(s). Once the system has called a module's deinitialize function and received a SYS_STATUS_UNINITIALIZED status from its status function, it may stop calling the module's state machine. However, it is safer to not make that assumption and place the module in an uninitialized state in which its state machine does nothing. See the Module Tasks section for details.

Note: This function is optional. If a module is not intended to be deinitialized in normal operation (i.e., only initialized after a reset and always running thereafter) it does not need to implement the deinitialize function.

Module Reinitialize

Describes the details of the module "Reinitialize" function.

Description

The function signature of a module's reinitialize function is defined by a pointer data type that is defined by the system module interface header, in the `<install-dir>/framework/system/common/sys_module.h` file, as shown in the following example.

Example: Module Reinitialize Function Signature

```
// *****************************************************************************
/* System Module Reinitialization Function Pointer

  Function:
    void (* SYS_MODULE_REINITIALIZE_ROUTINE) ( SYS_MODULE_OBJ object,
                                               const SYS_MODULE_INIT * const init)

  Summary:
    Pointer to a routine that reinitializes a module.

  Description:
    This data type is a pointer to a routine that reinitializes a system
    module (driver, library, or application).

  Preconditions:
    The low-level board initialization must have been completed and the
    module's initialization function must have been called before the system
```

```
    will call the reinitialization function for any module.

  Parameters:
    object          - Handle to the module instance

    init            - Pointer to the data structure containing any data
                      necessary to initialize the module. This pointer may
                      be null if no data is required and default initialization
                      is to be used.

  Returns:
    None.

  Remarks:
    This function uses the same initialization data structure as the Initialize
    function.
    This function can be used to change the power state of a module by passing
    in a different set of initial data values to reconfigure the module to a
    different power level.
    This function can also be used to refresh the hardware state as defined
    by the initialization data by passing in initial data values that match the
    previously given initial data values.  Thus, this function should guarantee
    that all hardware state is refreshed.
    This function can be called multiple times to reinitialize the module.
*/

typedef void (* SYS_MODULE_REINITIALIZE_ROUTINE) ( SYS_MODULE_OBJ object,
                                                   const SYS_MODULE_INIT * const init );
```

An implementation of a module's reinitialize function might look like the following example.

Example: Sample Module Reinitialize Function

```
void SAMPLE_Reinitialize ( SYS_MODULE_OBJ                object,
                           const SYS_MODULE_INIT * const init )
{
    SAMPLE_MODULE_DATA         *pObj  = (SAMPLE_MODULE_DATA *)object;
    SAMPLE_MODULE_INIT_DATA    *pInit = (SAMPLE_MODULE_INIT_DATA *)init;

    if (NULL == pInit)
    {
        pObj->status            = SYS_STATUS_READY;
        pObj->dataNewIsValid    = false;
    }
    else
    {
        pObj->status            = SYS_STATUS_BUSY;
        pObj->dataNew           = pInit->dataSome;
        pObj->dataNewIsValid    = true;
    }

    return;
}
```

The previous example initialize function for the sample module simulates resetting the hardware by resetting the `dataNew` value. If no `init` data is provided, it simply invalidates the current value by clearing the `dataNewIsValid` flag to false. However, if `init` data is provided, it stores the value and sets the library into a busy status that will be cleared once the library's tasks function has processed the data.

This may not be a particularly useful example, because it is possible to lose data given by the client if the `dataNew` value was currently valid. However, it does illustrate that the decision of what is preserved and what is not preserved when a module is reinitialized is a module-specific design decision that will depend on the type of module and how it operates safely.

Note: This function is optional. Any module that does not support power management or the ability to dynamically change initial settings while it is running does not need to implement this function.

Client Interface

Describes the client interface requirements for driver design.

Description

A driver's client interface is what is commonly thought of as its Application Program Interface (API). It is the interface through which any application

or other client interacts with the driver. It should be considered conceptually separate from the system interface (see the System Interface section). It represents the highest-level of abstraction at which an application or other module directly interacts with a specific peripheral device.

Middleware layers may implement protocol modules that simplify usage of specific types of peripherals (such as network interfaces or storage devices), but the driver interface is the highest level at which a client will interact directly with a peripheral. As such, a device driver should have a very simple usage model, especially for the most common uses of the device. The basic driver operations should allow an application to access the device, read and write data as if it were a simple file. In some cases, this will be all that is required. However, in most cases, device-type specific operations will also be required.

Driver-Client Usage Model

MPLAB Harmony drivers are required to provide a consistent "open/close" driver-client usage model.

Description

MPLAB Harmony device drivers follow a file system style device driver model, similar to that of POSIX-based operating systems, with a few primary differences. First, clients can directly access the drivers, through their own driver-specific functions instead of (or in addition to) indirectly through file system functions. Second, instead of having a single read/write data transfer model, there are multiple common data transfer models and some drivers may provide their own unique data transfer models. Third, instead of grouping all other input/output control operations into a single I/O control or IOCTL function, MPLAB Harmony drivers each provide a set of functions used to control the device.

In fact, the primary distinguishing feature of a MPLAB Harmony driver is that it uses a consistent driver-client usage model. This means that it has an open function and (optionally) a close function, and that before a client may use a device driver, it must call the driver's open function to obtain a driver handle. The handle is then passed into all other client-level interface functions as a parameter to identify the instance of the client that is calling and the instance of the driver (and, by implication the instance of the peripheral device) it is using, as shown in the following example.

Example: Driver-Client Usage Model

```
DRV_HANDLE  myUsart;
char       *message = “Hello World\n”;

/* Obtain an open handle to the USART driver. */
myUsart = DRV_USART_Open(MY_UART_INDEX, DRV_IO_INTENT_READWRITE);

/* Interact with the USART driver. */
DRV_USART_Write(myUsart, message, sizeof(message));

/* Continue using other USART driver client-interface functions as needed. */

/* Close USART driver if no longer needed. */
DRV_USART_Close(myUsart);
```

This example is somewhat simplified for the purpose of explanation. In normal usage, a client should test the value of the handle returned from the open function to ensure that it is not invalid (equal to DRV_HANDLE_INVALID). If the value of the handle returned is invalid, the caller should retry the open function later as it is possible that the driver is not yet ready for client usage. A client that requires the usage of a driver will normally prevent its state machine from advancing until a valid open handle has been obtained or it may eventually time-out and go into an error handling state.

The requirements of the open and close functions are described in the following section. The need for this model and the driver's internal usage of the handle are described in the Single Client vs. Multiple Client section.

Driver Client Interface Functions

Describes the format and naming convention for the interface functions of a driver client.

Description

The interface functions for the client of a driver follow a consistent format and naming convention, where `<module>` matches the module abbreviation for the driver or peripheral and `<operation>` is the name of the driver-specific operation to be performed.

Function	Description
`DRV_<module>_Open`	Open a link to the driver and start using it.
`DRV_<module>_<operation>`	Perform some driver-specific operation.
`DRV_<module>_Close`	Close link to the driver and stop using it.

The usage of the open and close functions is described previously in the Driver-Client Usage Model section and example implementations and key concepts are described in the Single Client vs. Multiple Client section. The details of the interface requirements (parameter data types, return values, etc.) are described in the following section. Requirements of data transfer operations are described in the Common Data Transfer Models section and general client interface requirements are described in the General Guidelines section.

Open

Describes the interface requirements for driver *open* functions.

Description

The purpose of a driver's *open* function is described in the Single Client vs. Multiple Client section. Driver *open* functions must meet the following interface requirements.

Function:

```
DRV_HANDLE DRV_<module>_Open ( const SYS_MODULE_INDEX index, const DRV_IO_INTENT intent )
```

Summary:

Opens a driver for use and provides an open-instance handle.

Descriptions:

This function opens a driver for use by a client module and provides an open-instance handle that must be provided to all of the other client-interface operations to identify the caller and the instance of the driver module.

Required or Optional:

Required.

Preconditions:

The driver module's initialize operation must have been called.

Parameters:

index	A zero-based index, identifying the instance of the driver to be opened. This value matches the index passed to the driver's initialize function.
intent	Flags parameter identifying the intended use of the driver: One of: • DRV_IO_INTENT_READ – Driver opened in read-only mode • DRV_IO_INTENT_WRITE – Driver opened in write-only mode • DRV_IO_INTENT_READWRITE – Driver opened in read-write mode One of: • DRV_IO_INTENT_NON_BLOCKING – Routines return immediately • DRV_IO_INTENT_BLOCKING – Routines return after operation is complete One of: • DRV_IO_INTENT_EXCLUSIVE – Will support only a single client. • DRV_IO_INTENT_SHARED – Can be used by multiple clients concurrently One flag from each group may be ORed together to fully define the intended use. However, the zero values and thus the default for each group is: DRV_IO_INTENT_READ, DRV_IO_INTENT_BLOCKING, and DRV_IO_INTENT_SHARED.

Returns:

If successful, the function returns a valid open-instance handle (an opaque value identifying both the caller and the driver instance). If an error occurs, the value returned is DRV_HANDLE_INVALID.

Example:

```
#define MY_I2C  0

handle = DRV_I2C_Open(MY_I2C, DRV_IO_BLOCKING|DRV_IO_RW|DRV_IO_NON_BUFFERED);
if (DRV_HANDLE_INVALID == handle)
{
    // Handle error
}
```

Blocking Behavior:

This function (and other client interface functions) may block on IO operations in an OS environment if DRV_IO_INTENT_BLOCKING (the default) is passed in the `intent` parameter. However, it (and other client interface functions) should never block waiting on I/O in a non-OS environment or it will block the entire system.

Remarks:

To support blocking behavior, the driver must be appropriately configured and built.

Drivers that are opened with a mode that is not supported must fail the open call by returning DRV_HANDLE_INVALID.

The default mode (if no flags are set, i.e., zero (0) is passed in the intent parameter), is blocking, read access only, and shared access.

Close

Describes the interface requirements for driver *close* functions.

Description

The purpose of a driver's *close* function is described in the Single Client vs. Multiple Client section. Driver *close* functions must meet the following interface requirements.

Function:

```
void DRV_<module>_Close ( DRV_HANDLE handle )
```

Summary:

Closes an opened-instance of a driver.

Descriptions:

This routine closes an opened-instance of a driver, invalidating the handle provided.

Required or Optional:

Optional – Not required if the driver is designed to never be closed.

Preconditions:

The driver's initialize function must have been called and the driver's open function must have returned a valid open-instance handle.

Parameters:

handle	A valid open-instance handle, returned from the driver's open function.

Returns:

None.

However, the driver's system-level status function will return SYS_STATUS_BUSY until the close operation has completed.

Example:

```
// Close the driver
DRV_I2C_Close(handle);
```

Blocking Behavior:

This function (and other client interface functions) may block on I/O operations in an OS environment if DRV_IO_INTENT_BLOCKING (the default) is passed in the `intent` parameter of the open function. However, it (and other client interface functions) should never block waiting on I/O in a non-OS environment or it will block the entire system.

Remarks:

Once this routine has been called, the handle provided will become invalid.

Common Data Transfer Models

Describes common data transfer usage models used by MPLAB Harmony drivers.

Description

MPLAB Harmony drivers for data transfer (or data source or data sink) peripherals normally follow one or more consistent data transfer usage models, described in this section. Each data transfer model has its own advantages and disadvantages, depending on the type of usage required and the execution environment. A single MPLAB Harmony driver may provide multiple data transfer models, depending upon the needs of the peripheral device. Usually, one particular model is the base or normal model, and others can be selected as optional features for broader compatibility.

Byte-by-Byte (Single Client)

Describes the byte-by-byte (single client) data transfer model.

Description

The byte-by-byte data transfer model provides a very simple mechanism for transferring data to and receiving data from a driver. It is very similar to the method commonly used by simple serial devices such as a USART that have transmitter and receiver FIFO buffers. This method transfers data one byte (or word) at a time until the driver's FIFO is either full (if transmitting data) or empty (if receiving data), as shown in the following examples.

Example: Reading Data Using the Byte-by-Byte Model

```
char buffer[MY_BUFFER_SIZE];
int  count;

for (count=0; count < MY_BUFFER_SIZE; count++)
```

```
{
    if (DRV_USART_ReceiverBufferIsEmpty(myUsart))
    {
        break;
    }
    else
    {
        buffer[count] = DRV_USART_ByteRead(myUsart);
    }
}
```

The previous example code assumes the USART driver has been successfully opened and the handle was stored in the `myUsart` variable. The `for` loop counts from 0 through MY_BUFFER_SIZE, unless the driver runs out of data. Each time through the loop, the code calls the DRV_USART_ReceiverBufferIsEmpty function to see if there is any data available. If there is no data available in the driver's receiver buffer FIFO (indicated by DRV_USART_ReceiverBufferIsEmpty returning true), the loop is aborted and the count is not incremented. If data is available in the driver's FIFO (indicated by DRV_USART_ReceiverBufferIsEmpty returning false), the example calls the driver's DRV_USART_ByteRead function and stores the data returned into the current position in the buffer, indexed by the count variable. Then, the `for` loop increments `count` before checking the loop exit condition and potentially starting over. When this loop exits, either because `count` reached MY_BUFFER_SIZE or because the driver had no more data available, the buffer contains count bytes of data received from the driver.

A very similar method is used to transmit data to the driver.

Example: Writing Data Using the Byte-by-Byte Model

```
char *buffer = "Hello World\n";
int   count;

for (count=0; count < strlen(buffer); count++)
{
    if (DRV_USART_TransmitBufferIsFull(myUsart))
    {
        break;
    }
    else
    {
        DRV_USART_ByteWrite(myUsart, buffer[count]);
    }
}
```

Again, it is assumed that the USART driver was previously opened and a valid `myUsart` handle obtained. The `for` loop counts from 0 through `strlen(buffer)`, unless it fills the driver's transmitter buffer FIFO first. Each time through the loop, it checks to see if the driver's transmitter FIFO is full. If it is (as indicated by DRV_USART_TransmitBufferIsFull returning true), it aborts the loop and does not increment the `count` variable. If the driver's transmitter FIFO buffer is not full, it will then call DRV_USART_ByteWrite to send the byte of data in `buffer` currently indexed by the `count` variable. It will then increment the `count` variable, check the loop exit condition and potentially start over. When the loop exits, `count` bytes of data from `buffer` have been sent from `buffer` to the driver's transmitter FIFO.

This data transfer model has the advantage that it is usually very lightweight and simple to implement, resulting in very little RAM and Flash required by the driver. However, this data transfer model is only safe for usage with single client drivers or with multiple client drivers that have been successfully opened in DRV_IO_INTENT_EXCLUSIVE mode. It is not safe for use with multiple clients or in a preemptive multi-tasking environment because it requires calling multiple functions to completely read or write a buffer of data. Refer to the Interrupt and Thread Safety section for an explanation on the types of issues that could occur in that environment.

Also, this data transfer model is not particularly easy to use, as it requires the caller to manage and adjust its current buffer pointer each time the loop exits. A synchronous file system style read/write data transfer model, described in the next section, may be simpler and safer from a caller's point of view, especially when used in a RTOS environment.

File System Read/Write

Describes the "file system style" read/write data transfer model.

Description

The synchronous, file system style, read/write data transfer model is intended to be similar to the POSIX read and write (and fread and fwrite) operations, as shown in the following example.

Example: Reading Data Using the File System Style Model

```
char    buffer[MY_BUFFER_SIZE];
size_t  count;

count = DRV_USART_Read(myUsart, buffer, MY_BUFFER_SIZE);
```

In the previous data read example, the single DRV_USART_Read function is called to transfer the entire contents of the `buffer` array. In a RTOS or file system model environment (described as follows), this function should not return until all data has been transferred or some form of error or time-out occurs. The success of this operation will be indicated by the return value stored in the `count` variable. If it is equal to the value of MY_BUFFER_SIZE, the operation completed successfully.

Example: Writing Data Using the File System Style Model

```
char    buffer = “Hello World\n”;
size_t  count;

count = DRV_USART_Write(myUsart, buffer, strlen(buffer));
```

In the previous data write example, the single DRV_USART_Write function is called to transfer the entire contents of the buffer string. Like the data reading example, in a RTOS or file system model environment, this function should not return until all data has been transferred or some form of error or time-out occurs. The success of this operation is indicated by the return value stored in the `count` variable. If its value is equal to `strlen(buffer)`, the operation completed successfully.

For simple data stream peripherals, such as UART, SPI, I2S, etc., this is normally the most basic data transfer model that a driver should support. It has the advantage that it provides a simple single-function interface and, depending on the configuration and execution environment, it manages advancing through the caller's buffer automatically and does not return until all data has been transferred.

If an error occurs, most drivers will return (`(size_t)-1`) (equivalent to a maximum unsigned integer value) from these functions. Or, in a non-blocking environment, they will only transfer the amount of data that can be buffered by the driver or hardware. So, this data transfer model has the disadvantage that the caller should always check the return count and may need to call a driver-specific status function to identify if an error has occurred. If no error has occurred, the caller may need to call the function again (potentially several times) to complete the desired transfer. So, this model is most suitable for configurations that provide sufficient buffering or that support a RTOS or where blocking behavior is acceptable, such as a file system model environment where operation of other modules is not required.

Buffer Queuing

Describes the buffer queuing data transfer model.

Description

The buffer queuing data transfer model is an asynchronous transfer mode. It is always non-blocking, so it allows the client to call the buffer add function multiple times, without waiting for each transfer to complete. This allows the caller to queue up more than one buffer at a time, potentially before the first buffer has finished (depending on buffer size and data transfer speed). The following examples show how this is done.

Example: Reading Data Using the Buffer Queuing Model

```
DRV_USART_BUFFER_HANDLE handle1;
DRV_USART_BUFFER_HANDLE handle2;
char buffer1[BUFFER_1_SIZE];
char buffer2[BUFFER_2_SIZE];

DRV_USART_BufferAddRead(myUsart, &handle1, buffer1, BUFFER_1_SIZE);
DRV_USART_BufferAddRead(myUsart, &handle2, buffer2, BUFFER_2_SIZE);
```

The previous example shows the caller queuing up two buffers to read data from the USART driver. When the first call to DRV_USART_BufferAddRead occurs, the driver will place the address and size of `buffer1` into its queue. Then, it will store a unique handle, identifying the data transfer request, into the `handle1` variable and begin copying data into the buffer (unless the driver was already busy placing data into a different buffer). Then the call will return. When the second call to DRV_USART_BufferAddRead occurs, the process repeats. If the driver has not yet finished filling `buffer1`, it will add the address and size of `buffer2` into its queue, provide a handle to it, and return.

Example: Writing Data Using the Buffer Queuing Model

```
DRV_USART_BUFFER_HANDLE handle1;
DRV_USART_BUFFER_HANDLE handle2;
char                    buffer1 = “Hello World\n”;
char                    buffer2 = “Hello Again\n”;

DRV_USART_BufferAddWrite(myUsart, &handle1, buffer1, strlen(buffer1));
DRV_USART_BufferAddWrite(myUsart, &handle2, buffer2, strlen(buffer2));
```

Similarly, the previous example shows the caller queuing up two buffers to write data to the USART driver. When the first call to DRV_USART_BufferAddWrite occurs, the driver will place the address and size of `buffer1` into its queue. Then, it will store a unique handle, identifying the data transfer request, into the `handle1` variable and begin copying data from the buffer (unless the driver was already busy copying data from a different buffer). Then, the call will return. When the second call to DRV_USART_BufferAddWrite occurs, the process repeats. If the driver has not yet finished copying all data from `buffer1`, it will add the address and size of `buffer2` into its queue, provide a handle to it, and return.

A driver that supports the buffer queuing model usually provides either a callback notification function or a status function (or both) so that a client can determine when the data transfer request has completed, as shown by the following examples.

Example: Using Callback Notification

```
DRV_USART_BUFFER_HANDLE handle1;
char                    buffer1 = “Hello World\n”;

DRV_USART_BufferEventHandlerSet(myUsart, MyUsartCallback, NULL);

DRV_USART_BufferAddWrite(myUsart, &handle1, buffer1, strlen(buffer1));
```

In the previous example, the client registers a callback function called MyUsartCallback with the USART driver before calling the DRV_USART_BufferAddWrite function to transmit of the contents of buffer1 on the USART. When the driver has completely transferred all data

from buffer1 by the USART, it will call the MyUsartCallback function, as shown in the following example.

Example: Callback Implementation

```
void MyUsartCallback ( DRV_USART_BUFFER_EVENT event,
                       DRV_USART_BUFFER_HANDLE bufferHandle,
                       uintptr_t context )
{
    switch ( event )
    {
        case DRV_USART_BUFFER_EVENT_COMPLETE:
        {
            if (bufferHandle == handle1)
            {
                 /* buffer1 data transfer complete */
            }
            break;
        }

        /* Handle other possible transfer events. */
    }
}
```

The MyUsartCallback function (described previously) is an example of how the client might handle the callback from the USART driver. In this example, the USART driver passes the DRV_USART_BUFFER_EVENT_COMPLETE ID in the `event` parameter to indicate that the data from `buffer1` has been completely transmitted by the USART. The value of the `bufferHandle` parameter will match the value assigned into the `handle1` parameter of the DRV_USART_BufferAddWrite function call so that the client can verify which buffer transfer has completed. (Recall that the driver can queue multiple transfers. It may, in fact, have multiple read and multiple write transfers queued at the same time.) For now, ignore the `context` parameter, which is explained in the Single Client vs. Multiple Client section.

The callback mechanism allows a client to synchronize to the timing of when the data transfers have completed. However, this adds some additional complexity to the interface and has a few potential subtle concerns. For one, a very short transfer may actually complete and the callback may occur before the DRV_USART_BufferAddWrite function actually returns. This is a potential *race* condition and it is why the transfer handle value is given as an output parameter and not as a return value. It allows the driver to ensure that the client has a valid handle before it starts the transfer so that the handle value is valid when the callback occurs. Also, in an interrupt-configuration, the callback may occur in an ISR context. Therefore, the client should be careful to not call anything that may block. In particular, the client should be careful to not call any of the driver's own API functions as they are not normally designed to be called from within the driver's own ISR.

Alternately, client may not require hard real time notification of the completion of the transfer of the buffer data. It may just need to know when it can safely reuse the buffer. If that is the case, the driver may also provide an interface function that will allow the client to poll the driver at its convenience to determine if the buffer has completed, as shown by the following example.

Example: Checking for Buffer Status

```
DRV_USART_TRANSFER_STATUS status;

status = DRV_USART_BufferStatusGet(myUsart, handle1);
if (status == DRV_USART_BUFFER_COMPLETE)
{
    /* Buffer Transfer Has Completed */
}
```

The previous example shows how the client can call the driver, passing in the `handle1` value given by the DRV_USART_BufferAddWrite function, to poll at its leisure to find out when the driver has completed using `buffer1` and has transferred all the data it contained. Most drivers will provide both the buffer status function and the callback mechanisms so that the client can choose the most appropriate one.

Using the buffer queuing model, a client can keep a driver at 100% throughput utilization by queuing up at least two buffers. When the first buffer completes, the client has until the second buffer completes to queue another buffer. Using either the byte-by-byte or file system style models requires the client to respond within the time it takes to fill or empty the driver's built-in FIFO buffer to keep a continuous stream of data transfers. This can be a very short period, potentially as short as the time it takes to transfer a single byte on the peripheral. However, the buffer queuing method is somewhat more complex to use and usually requires more RAM and Flash to implement. Therefore, a driver may not offer it or may only offer it as an optional feature, unless the normal operation of the peripheral requires continuous, uninterrupted data transfer.

General Guidelines

Provides general guidelines for device driver client interface design.

Description

In addition to the open, close, and data transfer functions, most device drivers will require a number of device type-specific interface functions. What these functions do will depend on exactly what type of device is being controlled. It is generally best to provide a solution for the most basic usage of a given type of peripheral first. It is easy to add unique optional capabilities later, but it is hard to fix a bad interface without breaking existing client code. Keep in mind that the driver should manage the state of the device so the client does not have to. Intermediate steps that are part of normal operation should be hidden by the interface as much as possible. The following list provides more general guidelines that may help when defining a driver's interface.

1. A driver that supports only a single type of peripheral is easier to maintain and more flexible to use. Only integrate drivers for different types of peripherals together when the merged driver serves another purpose and cannot expose any of the underlying functionality. For example, if the merged driver is a touch screen driver that fully utilizes the underlying Timer and ADC resources, leaving neither available for other uses. Otherwise, the merged driver will need to expose multiple driver interfaces so as to be transparent to clients.
2. Define the interface based on the client's point of view of an idealized and abstracted version of the peripheral, not on a detailed understanding of the device. Any details that may be different from one implementation of a device to another should be hidden from the client. Features that exist on one piece of hardware and not on another that need to be exposed to the interface can be added or removed as configuration selections. Operations related to features that do not exist on one part can be grouped together and removed from the interface when not supported.
3. Although it may be appropriate for a driver to maintain its own buffer(s) in which to collect data, it is generally preferable to use the caller's buffer. This places decisions about buffer size and allocation with the caller, who has better knowledge of exactly how the data is to be used. Ownership of the buffer is passed to the driver when a data transfer function is called and released from the driver either when the call returns or when the transfer completion status has been given to the client.
4. Many driver functions will perform operations that require some time to complete (usually waiting for some status to change). These functions should report status to indicate whether an operation is complete or not and provide a handle or identifier with which the client can identify the operation later when it completes.
5. An interface should be appropriately sized (i.e., it should not contain too many operations or too few operations). A smaller interface is generally better, if it can support all of the required features. However, performance, ease of use, and compatibility are more important. Do not sacrifice any of these considerations to eliminate interface functions.
6. Try to use data types that can be easily ported to an appropriate size (8-, 16-, 32-bit) if the data value range or processor changes for parameters or return types, unless the usage model of the driver requires a specific bit width. When a specific data size is required, use the C99 data types defined in `stdint.h`, `stdbool.h`, and `stddef.h`.
7. If DMA is supported (for peripherals that would benefit from it), it should be hidden behind the same data transfer operations used when it is not available and either enabled as a build-time configuration option or enabled and disabled by a setup function at run-time, as appropriate.
8. Drivers must use system services for memory allocation, interrupt control, system clocks and timers, power management, physical/virtual address conversion, etc. They must also use the OSAL for thread safety and synchronization. Generally, these facts should be hidden from the client interface. (Refer to the System Services and OSAL help for information on what services are available and how to use them.)
9. Interrupt specifics, such as the interrupt ID and vector numbers, should be abstracted away as build-time configuration options (for static implementations) or instance-specific initialization options (for dynamic implementations) unless there is a need to change them dynamically at run-time.

As always, it is best to follow generally accepted programming practices and a consistent programming style. Remember that source code frequently outlives its original purpose and well-designed and easily readable and maintainable code will be a joy to work with long after the original project is completed.

Interrupt and Thread Safety

Describes key concepts and concerns related to safe driver operation when using interrupts or RTOS threads.

Description

MPLAB Harmony allows libraries to be configured for any one of several different environments. This is accomplished by designing libraries that comprehend the restrictions of all supported environments and that are configurable for each.

The basic concept is to always consider the context in which a function can be called. Since a driver has both system and client interface functions, the developer must consider both of these interfaces. In particular a driver's tasks function(s) may be called from any one of the following three contexts (but only from one of them in any given configuration).

- The main polling loop
- An ISR, if an appropriate one is available
- A loop in a RTOS thread or task function

It is also important to keep in mind that a driver's client interface functions are normally called from an application's tasks function (or some other client module's tasks function), which is called from a potentially different one of the previous three contexts. And, if a driver supports callback functions, the developer must consider from what context it will call the client's function (again, from a potentially different one of the previous three contexts).

Supporting (and testing) the capability to configure a driver for any of these environments, while potentially challenging, helps to produce robust and flexible drivers that can be used and reused in the widest range applications and that are easily portable to future projects. It also helps to ensure that they are reliable in the project for which they were originally developed by helping to find corner cases and ensure reliable operation.

Of the three contexts listed previously, most developers consider the polled environment is first when designing and implementing libraries. This environment is the easiest to understand, but it is also the most restrictive and limiting in terms of system capabilities and real-time responsiveness. So, most systems will utilize at least some interrupt-driven code or use a RTOS, requiring the developer to consider the issues described in this section.

Atomicity

Describes atomic code sequences and data types at a fundamental level.

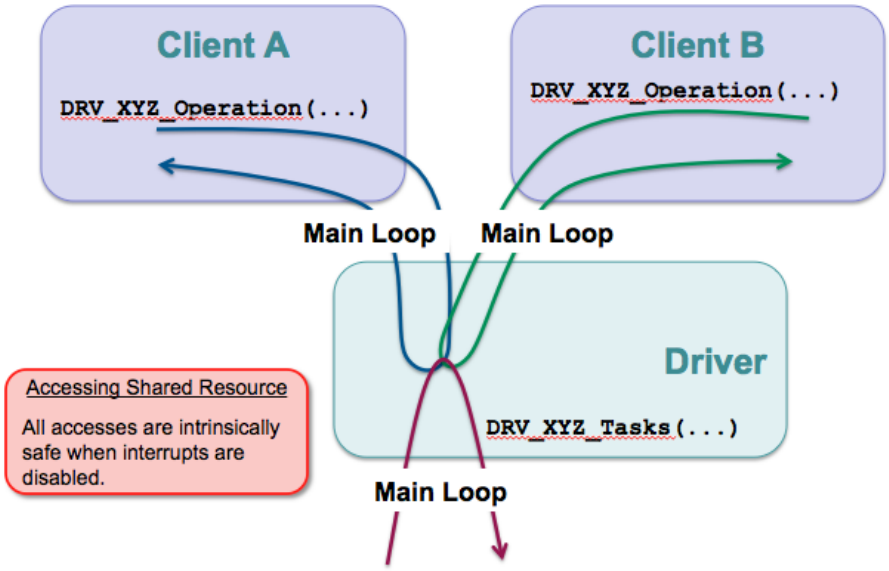
Description

The English word *atomic* derives from a Greek word meaning *indivisible*. A sequence of instructions that is indivisible once started and cannot be

interrupted until it has completed is called atomic. A data item is considered atomic if it cannot be subdivided as it is read or written. These are critical concepts for all software, including real-time embedded systems, because interrupts and context switches can occur at particularly inopportune times and potentially cause data corruption or incorrect behavior or even complete system failure. Such possibilities must be prevented to guarantee correct and robust functioning of the system.

When a processor is reset, interrupts are disabled and the processor executes instructions one after the next and will continue to do so until powered off or reset. Therefore, all code is effectively atomic in a polled environment where interrupts are globally disabled. In such a strictly polled configuration, conflicts due to non-atomic accesses to resources shared between clients and drivers do not occur.

Polled Configuration



Even though different clients may call the same driver function and access, the same resources that are also accessed by the driver's tasks function, all functions are called from the context of the main system loop and only one such access will occur at a time and it will complete without interruption.

However, once interrupts are enabled, there is a very limited set of situations where atomicity can be guaranteed. The execution of a single instruction is atomic. When an interrupt occurs, an instruction will either execute completely or it will not be started. Interrupts are synchronized to the instruction flow and they effectively occur between the instructions. Or, more accurately, the CPU can only respond to an interrupt after completing the instruction it is currently decoding and executing. That may seem to be obvious, but it is the basis for all atomicity of both data items and sequences of instructions in a computing system.

Atomicity is also guaranteed when reading or writing a single data word with a single instruction. A data word contains the same number of bits as the data bus width. In a correctly functioning system, a data value that is the width of the processor's data bus (or less if half word or smaller values are supported, as they are on PIC32 microcontrollers) will be read or written in its entirety. The data word will never be partially read or partially written.

However, if a variable, data structure, or array is larger than a single data word (for example, a 64-bit value on a 32-bit processor), it will take more than one instruction to read or write the entire value and interrupts could occur in between those instructions. If that happens, it is possible that part of the data value could be changed by the interrupt and may not be consistent with the part that was read before the interrupt (or written after the interrupt) when the sequence of instructions completes. Such data types should not be considered atomic. Instruction sequences that access them should be protected to guarantee that there is no possibility of corruption.

Even if a data type is atomic (only one word or less in size), it may still be necessary to protect a code sequence that performs a read-modify-write operation on it because almost all read-modify-write operations require execution of multiple instructions, which could be interrupted. This concern applies to both data stored in memory (in variables, structures, arrays, and buffers) as well as registers (or SFRs). If non-atomic accesses are made to these resources from both an ISR and the main loop or from more than a single thread context when using a RTOS, that resource is considered shared and access to it must be made atomic by guarding or protecting it as described in the following sections.

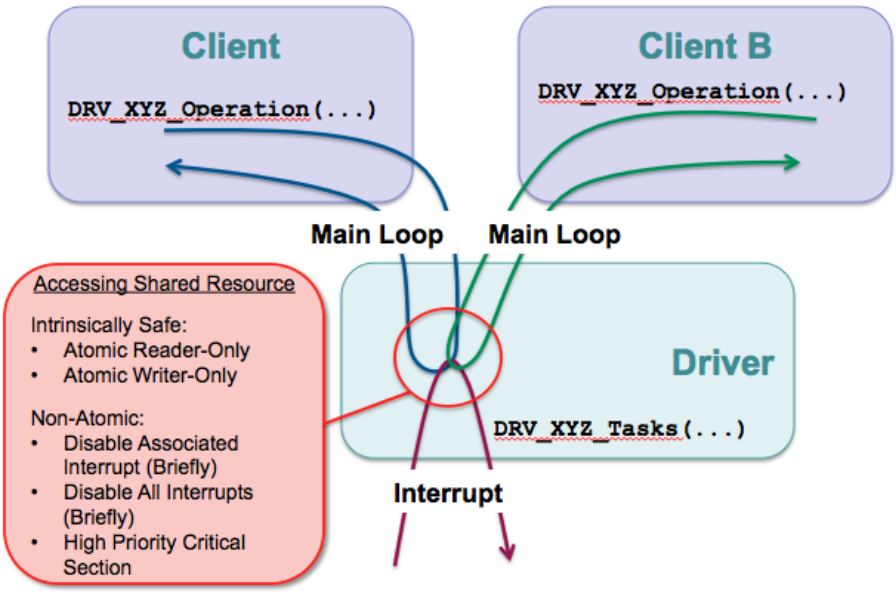
Interrupt Safety

Describes how to guard against the potential conflicts that interrupts can cause.

Description

If a sequence of code must be atomic (indivisible and uninterruptible), the relevant interrupts must be disabled before entering the sequence. In MPLAB Harmony, interrupts can be disabled globally by using the interrupt system service or by using a high-priority mutex. However, MPLAB Harmony libraries are modular and by convention they respect the abstractions of other libraries, never attempting to directly access their internal resources. Because of this convention, the only code in the system that should ever attempt to access the internal resources owned by a driver is the driver code itself, as shown in the following diagram.

Interrupt-Driven (No RTOS)



This means that it is not necessary to globally disable interrupts in most cases to guarantee correct and reliable operation of a MPLAB Harmony driver. It is usually sufficient for a driver to temporarily mask just the interrupt(s) of the peripheral it owns while performing non-atomic accesses to its own data structures or peripheral hardware. Doing so will prevent the driver's own interrupt-driven tasks function(s) from potentially corrupting data that is also accessed by the driver's interface function(s).

This can be done using the interrupt system service and it is more efficient than globally disabling all interrupts because it allows higher priority interrupts (which do not affect the driver in question) to occur, protecting their response time latency. This can be done using the Interrupt System Service, as shown in the following example.

Example: Temporarily Disabling an Interrupt Source

```
#define MY_INTERRUPT_SOURCE  INT_SOURCE_TIMER_2

bool enabled;

enabled = SYS_INT_SourceDisable(MY_INTERRUPT_SOURCE);

/* Access resource shared with interrupt-driven tasks function. */

if (enabled)
{
    SYS_INT_SourceEnable(MY_INTERRUPT_SOURCE);
}
```

This method is safe to use, even if the driver in question is not running in an interrupt-driven mode because it only re-enables the interrupt source if it was enabled before the sequence was entered. However, when a driver is not configured for interrupt-driven operation, it must not enable its own interrupt and the code that is necessary to disable the interrupt and restore its previous state is not necessary and could be removed to save code space. Fortunately, this can be accomplished fairly easily by abstracting the interrupt management code behind functions that switch implementations depending upon the configuration of the driver, as shown in the following example.

Example: Interrupt Management Functions

```
#if (SAMPLE_MODULE_INTERRUPT_MODE == true)

    #define _SAMPLE_InterruptDisable(s)    SYS_INT_SourceDisable(s)

#else

    #define _SAMPLE_InterruptDisable(s)    false

#endif


#if (SAMPLE_MODULE_INTERRUPT_MODE == true)

    static inline void _SAMPLE_InterruptRestore ( INT_SOURCE source, bool enabled )
    {
        if (enabled)
        {
            SYS_INT_SourceEnable(source);
        }
```

```
    }

#else

    #define _SAMPLE_InterruptRestore(s,e)

#endif
```

This method effectively compiles away the interrupt management code, adding little or no object code when used in a polled configuration (when SAMPLE_MODULE_INTERRUPT_MODE is false). But, when used in an interrupt-driven configuration (when SAMPLE_MODULE_INTERRUPT_MODE is true), these functions use the system interrupt service to manage the driver's interrupt source(s). These functions can then be used to guard non-atomic accesses by the driver's interface functions to resources that are shared with the driver's ISR, as shown in the following code example.

Example: Interrupt Management

```
bool SAMPLE_DataGet ( const SYS_MODULE_INDEX index, int *data )
{
    SAMPLE_MODULE_DATA *pObj;
    bool                intState;
    bool                result = false;

    pObj = (SAMPLE_MODULE_DATA *)&gObj[index];

    // Guard against interrupts
    intState = _SAMPLE_InterruptDisable(pObj->interrupt);

    if (pObj->dataProcessedIsValid)
    {
        // Provide data
        *data = pObj->dataProcessed;
        pObj->dataProcessedIsValid = false;
        result = true;
    }

    // Restore interrupt state.
    _SAMPLE_InterruptRestore(pObj->interrupt, intState);

    return result;
}
```

In this code example, the driver interface function SAMPLE_DataGet calls _SAMPLE_InterruptDisable before checking a flag and potentially updating an internal data structure, which is a non-atomic process that will take multiple instructions. When SAMPLE_MODULE_INTERRUPT_MODE is true, the _SAMPLE_InterruptDisable function will call the SYS_INT_SourceDisable system service function to atomically disable the interrupt source and capture its current state (whether it was enabled or disabled before being disabled). Once it is done accessing the data structure, it calls the _SAMPLE_InterruptRestore function to restore the interrupt to its previous state. This ensures that the ISR cannot fire and call the tasks function to modify this data until the interface function has finished with the data. However, if SAMPLE_MODULE_INTERRUPT_MODE is false, the _SAMPLE_InterruptDisable function will be replaced with a constant false value (to avoid a syntax error in the assignment of the function return value) and the _SAMPLE_InterruptRestore function will be completely removed.

This method works to ensure safe access to shared resources without disabling interrupts globally when using a bare-metal configuration. Of course, if a section of code must be truly atomic (uninterruptible), interrupts can be globally disabled for a short period of time using either the Interrupt System Service functions or a high-priority critical section.

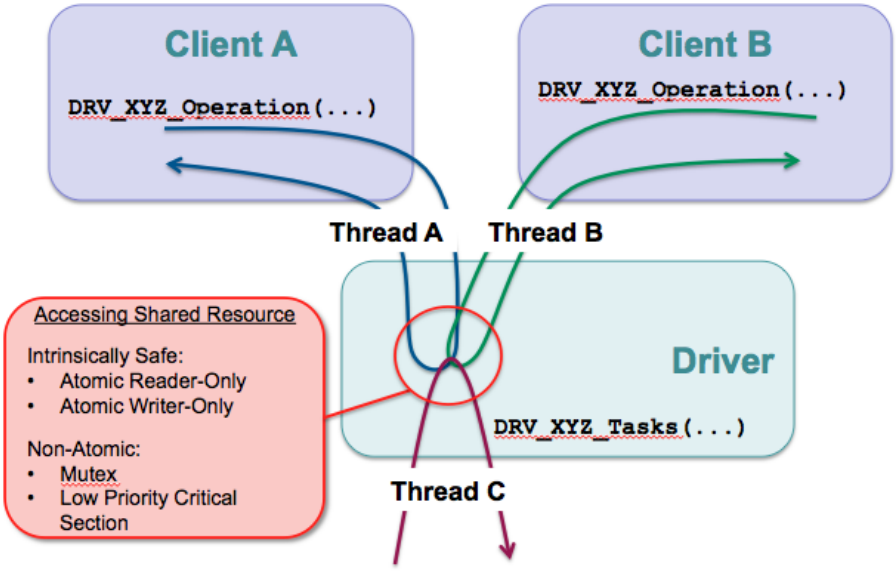
RTOS Thread Safety

Describes how to protect accesses to shared resources and critical sections of code from corruption by simultaneous access by multiple RTOS threads (tasks) and interrupts.

Description

When utilizing a RTOS, it is possible that a driver and its clients may each run in its own RTOS thread. If the RTOS is preemptive, it is possible that the scheduler may interrupt any of these threads at any time and switch to another. If the other thread happens to also non-atomically access the same shared resource or execute the same critical section of code (as shown by the following diagram), that section of code must be guarded and made atomic using the methods provided by the MPLAB Harmony Operating System Abstraction Layer (OSAL).

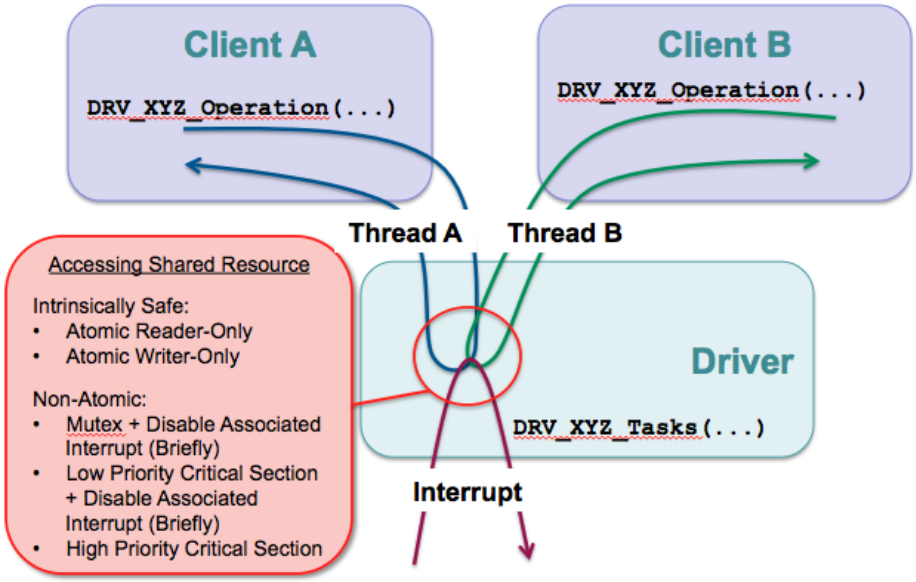
RTOS Polled



In a RTOS configuration, where the MPLAB Harmony driver and all of its clients are running strictly polled in their own threads, a mutex or low-priority critical section is sufficient to protect non-atomic accesses to shared resources or critical sections of code. Both of these mechanisms will instruct the RTOS scheduler to only allow a single thread to access to the resource or execute the critical code sequence at a time and will block all other threads, making them idle until the first thread has released the mutex or exited the critical section.

However, MPLAB Harmony drivers normally run most efficiently when interrupt driven. So, even when utilizing a RTOS, it is common for a driver's tasks function to be called from an interrupt context, as shown in the following diagram.

RTOS Interrupt-Driven



In this situation, a mutex or a low-priority critical section can still be used to guard against simultaneous access to shared resources by different threads (for example client A's thread and client B's thread, as shown previously). However, neither will prevent an ISR from accessing the shared resource or critical code section right in the middle of the thread's attempt to access it. So, if either of these methods is used, they must be augmented by also temporarily disabling (masking) the associated interrupt source, exactly the same way it is done in a bare metal environment as described in the Interrupt Safety topic. It is also possible to simply disable interrupts globally and prevent context switches by using a high-priority critical section. But, this is a brute force solution that is not recommended unless the timing of the code sequence is absolutely critical, and then only for very brief periods (see the Blocking Guidelines section for recommendations on what is an acceptably brief period).

The following example uses the _SAMPLE_InterruptDisable and _SAMPLE_InterruptRestore functions from the previous (Interrupt Safety) topic in conjunction with the OSAL mutex functions to effectively guard non-atomic accesses to shared resources from within client interface function code.

Example: Guarding Shared Resources

```
result = false;

if (OSAL_MUTEX_Lock(&pObj->mutex, SAMPLE_MODULE_TIMEOUT) == OSAL_RESULT_TRUE)
{
    /* Guard against interrupts */
    intState = _SAMPLE_InterruptDisable(pObj->interrupt);
```

```
    /* Check for storage space */
    if (!pObj->dataNewIsValid)
    {
        /* Store data */
        pObj->dataNew           = data;
        pObj->dataNewIsValid    = true;
        pObj->status            = SYS_STATUS_BUSY;
        result                  = true;
    }

    /* Restore interrupt state and unlock module object. */
    _SAMPLE_InterruptRestore(pObj->interrupt, intState);
    OSAL_MUTEX_Unlock(&pObj->mutex);
}
```

In the previous code example, the OSAL_MUTEX_lock function is called first to lock the mutex. When using a RTOS, this is the point at which it will block any subsequent thread entering the sequence before the first exits. So, the locked mutex protects the current thread against accesses by multiple clients. However, the non-atomic access sequence still needs to be guarded against ill-timed interrupts by calling the local _SAMPLE_InterruptDisable function and passing in the appropriate interrupt flag ID. When interrupt-driven, this will disable the interrupt and (atomically) capture the previous status of the interrupt sourced passed in. Then, once finished accessing the shared resource, this example calls the inverse functions (_SAMPLE_InterruptRestore and OSAL_MUTEX_Unlock) in reverse order to restore the previous state, unlock the mutex and continue safely.

In a bare metal configuration, if the mutex is already locked, the OSAL_MUTEX_lock function will return OSAL_RESULT_FALSE and the if will fail the result variable will stay false, allowing the interface function that contains this code to provide a negative result to the caller. If the mutex is not locked, it will become locked and the `if` clause will be taken, emulating the behavior of a RTOS. Of course, as described in the Interrupt Safety section, the interrupt enable and disable functions also change implementation depending on whether the driver is configured for interrupt-driven or polling operation. So, this method works for client interface functions in all possible execution environments (bare metal polled, bare metal interrupt-driven, RTOS polled and RTOS interrupt-driven). It has full flexibility and configurability when implementing client-interface functions. However, a driver's tasks function requires a slightly different set of behavior, as described in the following paragraph.

When interrupt-driven, the driver's tasks function is called from within the ISR. In an ISR, the associated interrupt source is already masked and it will be automatically unmasked when the ISR returns. This means that the driver's tasks function does not need to disable and restore its own interrupt source, only the driver's other functions need to do that. However, driver's tasks function can only call potentially blocking OSAL functions when in a polled configuration (RTOS-based or bare metal). When built in an interrupt-driven configuration, any interrupt-driven tasks functions must not call OSAL functions that might block. This means that any OSAL functions called from a tasks function must switch implementations and compile away to nothing when the driver is built in an interrupt-driven configuration. The following example shows a simple way to do this based on the interrupt configuration option.

Example: OSAL Use in Interrupt Tasks Functions

```
#if (SAMPLE_MODULE_INTERRUPT_MODE == false)

    static inline bool _SAMPLE_TasksMutexLock (SAMPLE_MODULE_DATA *pObj)
    {
        if (OSAL_MUTEX_Lock(&pObj->mutex, SAMPLE_MODULE_TIMEOUT) == OSAL_RESULT_TRUE)
        {
            return true;
        }

        return false;
    }

#else

    #define _SAMPLE_TasksMutexLock(p)      true

#endif


#if (SAMPLE_MODULE_INTERRUPT_MODE == false)

    static inline void _SAMPLE_TasksMutexUnlock ( SAMPLE_MODULE_DATA *pObj )
    {
        OSAL_MUTEX_Unlock(&pObj->mutex);
    }

#else

    #define _SAMPLE_TasksMutexUnlock(p)

#endif
```

These functions effectively compile away when used in an interrupt-driven configuration (when SAMPLE_MODULE_INTERRUPT_MODE is true), adding little or no object code. But, when used in an interrupt-driven configuration (when SAMPLE_MODULE_INTERRUPT_MODE is false), these functions translate to the desired OSAL mutex functions and can be used to guard non-atomic accesses in the tasks function, as shown in the following example.

Note: The OSAL functions themselves map to the appropriate RTOS-specific or bare metal implementation, so no additional steps need be taken to ensure flexibility to use or not use a RTOS. Only the interrupt and non-interrupt behavior needs to be mapped by functions defined in the driver's source code.

Example: Guarding Shared Resources in Interrupt-Driven Tasks

```
void SAMPLE_Tasks( SYS_MODULE_OBJ object )
{
    SAMPLE_MODULE_DATA     *pObj  = (SAMPLE_MODULE_DATA *)object;

    SYS_ASSERT(_ObjectIsValid(object), "Invalid object handle");

    if (!_SAMPLE_TasksMutexLock(pObj))
    {
        return;
    }

    // Process data when ready.
    if (pObj->dataNewIsValid && !pObj->dataProcessedIsValid)
    {
        pObj->dataProcessed         = pObj->dataNew;
        pObj->dataNewIsValid        = false;
        pObj->dataProcessedIsValid  = true;
        pObj->status                = SYS_STATUS_READY;
    }

    _SAMPLE_TasksMutexUnlock(pObj);

    return;
}
```

This previous example shows how to use the mapped mutex lock functions within an ISR-driven tasks function. Note that if the _SAMPLE_TasksMutexLock function returns false, it is potentially an error condition and the tasks function will be unable to perform its task because the resources are unavailable. So, be sure to perform appropriate error recovery or management if necessary.

A task function that manages an interrupt is, by nature, interrupt safe because it is either called from the appropriate ISR or it is polled and the associated interrupt source is disabled. And, the previous method can be used to also make them thread safe. So, it provides all four combinations of ISR and thread safety and can be used to make fully configurable drivers for all four combinations of interrupt versus polled and RTOS versus non-RTOS configurations.

Note: Tasks functions that do not manage an interrupt (such as tasks functions for applications or second-level tasks function) and that can only be polled should be treated like client interface functions. Non-atomic accesses to resources that can be accessed from other thread contexts and/or interrupts must be protected using the same methods as used by client interface functions.

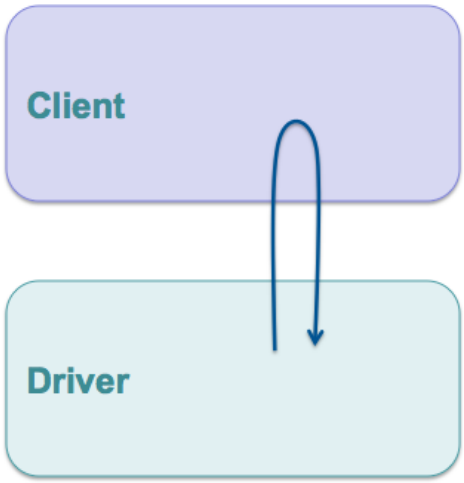
Callback Functions

Describes callback functions and the potential interrupt and thread-safety concerns they may cause.

Description

Normally, a client calls the driver's interface functions to interact with it. But, a callback function is a client function called by the driver back to the client, instead of the other way around, as shown in the following diagram.

Callback Functions



Callback functions are usually dynamically registered with a driver (or other server library). To do this, the driver provides an interface function that the client can call and pass in a pointer to the function it wants the driver to call back (the MyCallback function in the following example).

Example: Registering a Callback Function

```
#define PERIOD    1000
#define NO_REPEAT false

if (DRV_TMR_AlarmRegister(pObj->tmrHandle, PERIOD, NO_REPEAT, pObj, MyCallback))
{
    /* Successfully registered “MyCallback” function */
}
```

A callback is different from functions that are statically linked and called from the driver by name. Statically linked functions are considered dependencies. Dependencies are requirements of the driver. It will not build without an implementation of the dependency. Dependencies should be limited to only the things that the driver needs to use to do its job. They should not be part of the client interface. If they are, they force the driver to be static and single client. Dynamically registering a callback function (by a function pointer) instead of statically linking it to the driver allows the driver to be static or dynamic, or single client or multiple client.

Note: Sometimes two libraries may be mutually dependent on each other or a library may have dependencies upon functions defined in configuration files that are implemented as part of the system configuration. But, a library should not be dependent upon a client.

The callback function whose address is passed to the callback registration function probably needs to do something when it is called back. Likely it will need to set a flag (or semaphore) or capture some status value (see the following example).

Example: Callback Function

```
void MyCallback ( uintptr_t context, uint32_t alarmCount )
{
    MY_DATA_OBJECT *pObj = (MY_DATA_OBJECT *)context;

    pObj->alarmCount += alarmCount;
}
```

Caution should be taken when designing the usage model of a driver callback. This is because in an interrupt-driven configuration the callback function might be called from the driver's ISR or in a RTOS configuration the callback might be called from a different thread context. This places additional complexity on the client and on the driver, especially if the client then needs to call the driver's interface functions from the callback function.

Also, it is important to carefully document the context in which a callback function can be called because a client cannot disable an interrupt owned by a driver, or any other module, because each module manages its own interrupts. It would be a violation of the driver's abstraction. The client should not know what interrupts the driver uses or if it uses any interrupts at all. In this situation, a client may not be able to make non-atomic accesses to its own internal data structures if they are accessed by the callback and its own state machine or interface functions without globally disabling interrupts. For example, the `pObj->alarmCount += alarmCount` line in the previous example is a non-atomic (read-modify-write) access so the client using this driver would need to stop the timer callbacks from happening before it attempted to perform a non-atomic access to the `alarmCount` variable in its MY_DATA_OBJECT structure from any of its other functions, as shown in the following example.

Example: Temporarily Disabling a Callback

```
bool previousAlarmEnable;

alarmWasDisabled = DRV_TMR_AlarmDisable(pObj->myTimer);

if (pObj->alarmCount > MY_MAX_ALARM_COUNT)
{
    /* Reset my alarm count. */
    pObj->alarmCount = 0;
}
```

```
DRV_TMR_AlarmEnable(pObj->myTimer, previousAlarmEnable);
```

In the previous example, the `if` statement reads the `alarmCount` member of the current module's structure and, depending on its value, the assignment inside the `if` statement modifies and writes it. Since this takes more than one instruction, the caller cannot allow alarm callbacks to occur in this time. So, the Timer Driver provides client interface functions to conveniently disable and restore the callback functionality. If the driver did not provide these functions the client would have to deregister the callback and/or stop the timer to ensure that the `alarmCount` variable would not be corrupted by a simultaneous access by its own interface or tasks functions.

Another concern that the driver developer must take care to avoid when providing a callback function is a common race condition that can occur. Most callback functions are used as synchronization methods. (See Synchronization for details.) An interface call-in function will usually start some process that takes time and a callback function will notify the client when the process has completed. The danger is that, if the process completes too quickly, it may cause an interrupt to occur immediately and call the callback function before the call-in function returns. That can be a serious problem if the client needs to use the return value of the interface call-in function from within the callback function. Unfortunately, that is exactly what would happen when a transfer handle is returned from a data transfer function, as shown in the following example.

Example: Transfer Synchronization Callback

```
void MyBufferEventHandler ( DRV_USART_BUFFER_EVENT  event,
                            DRV_USART_BUFFER_HANDLE bufferHandle,
                            uintptr_t               context )
{
    MY_OBJ *pObj = (MY_OBJ *)context;

    if (pObj->myBufferHandle == bufferHandle)
    {
        switch(event)
        {
            case DRV_USART_BUFFER_EVENT_COMPLETE:
            {
                /* Clean up after my buffer transfer is complete. */
            }

            /* Handle other events for my buffer */
        }
    }
}
```

Example: Interface With an Intrinsic Race Condition

```
char buffer[] = “Hello World\n”;

pObj->bufferHandle = DRV_USART_BufferAddWrite(pObj->myUsart, buffer, strlen(buffer));
```

In the previous example, the DRV_USART_BufferAddWrite function adds the buffer containing the `Hello World\n` string to the USART driver's write buffer queue and returns a handle identifying the request to write that buffer. When the transfer completes, the driver will call the MyBufferEventHandler function and pass in the DRV_USART_BUFFER_EVENT_COMPLETE event, the buffer handle returned from the DRV_USART_BufferAddWrite function, and the context value passed in when the callback was registered. (The context is used to identify the instance of the caller that registered the callback. In most cases, a caller will pass in a pointer to its instance data structure so the callback can recover it, as shown in this example.)

In this example, as long as the DRV_USART_BufferAddWrite function returns the buffer handle before transfer finishes and the MyBufferEventHandler function is called back, this will work just fine. But, what happens if the write queue is empty, the buffer is only one byte in size and the baud rate really is high? It is possible that the transfer will finish and the callback will happen when the interrupt occurs before the DRV_USART_BufferAddWrite function has had time to return and the buffer handle value has been assigned to the `pObj->bufferHandle` variable. If that should happen, the value of the `bufferHandle` parameter passed into the MyBufferEventHandler callback function will not match the value in the value stored in the `pObj->bufferHandle` variable because it has not yet been stored there. When that occurs, the MyBufferEventHandler function will incorrectly decide that the event was not for buffer it was looking for and it will not correctly perform whatever clean-up logic it was designed to perform. So, whether or not this callback works correctly depends on who wins the race, the client or the driver.

To avoid this common race condition, it is necessary to put the timing of the assignment of the buffer handle into the hands of the driver where it can be managed successfully. A better driver interface design would make the transfer handle an output parameter instead of a return value by passing the address of the variable to receive the value of the buffer handle as a parameter, as shown in the following example.

Example: Interface Without an Intrinsic Race Condition

```
char buffer[] = “Hello World\n”;

DRV_USART_BufferAddWrite(pObj->myUsart, buffer, strlen(buffer), &pObj->bufferHandle);
```

In this example, the USART driver can now eliminate the potential race condition that was intrinsic in the previous DRV_USART_BufferAddWrite and callback interface. By passing the address of the `pObj->bufferHandle` variable into the DRV_USART_BufferAddWrite function, the driver can ensure that it assigns the correct `buffer` handle value to the variable before it starts transferring the buffer. This guarantees that, no matter how quickly the buffer transfer finishes and how quickly the callback occurs, the client's variable has the correct value so it will match the value passed in by the driver.

Callback functions can be a very useful mechanism for synchronizing between a driver and its client, but care must be taken make sure the client has a working usage model or the driver may not be useful in some configurations.

Synchronization

Describes how to use the OS Abstraction Layer (OSAL) to synchronize between threads and ISRs to manage blocking behavior.

Description

Some driver interface functions have an intrinsically blocking usage model. For example, most developers would expect the file system style read and write functions to block and not return until the entire transfer had completed. While this cannot be accomplished in a bare-metal environment, it can be accomplished in a RTOS configuration in a way that still allows usage in a non-RTOS environment by using the OSAL semaphore support.

Example: Blocking Function

```
size_t DRV_MYDEV_Write( DRV_HANDLE handle, void *buffer, size_t size)
{
    size_t        count;
    DRV_MYDEV_OBJ pObj  = (DRV_MYDEV_OBJ *)handle;

    count = 0;
    while(count < size)
    {
        count += PLIB_MYDEV_Transmit(pObj->devIndex, buffer, size-count);
        if (count < size)
        {
            if (OSAL_SEM_Pend(pObj->txSemaphore) == OSAL_RESULT_TRUE)
            {
                /* Exit loop if semaphore fails or if no RTOS */
                break;
            }
        }
    }

    return count;
}
```

In the previous example, the DRV_MYDEV_Write function attempts to loop, repeatedly filling the fictitious MYDEV device's transmit FIFO until it has sent all size bytes of data pointed to by the buffer parameter by calling the PLIB_MYDEV_Transmit function (assuming that is what this function does and that it returns the actual number of bytes copied to the transmitter FIFO). If the buffer passed in contains more data bytes than will fit into the device's FIFO, `count` will be less than `size` and the code will call the OSAL_SEM_Pend function. In a RTOS configuration, assuming that no other code has previously posted the `txSemaphore`, this will cause the thread that called the DRV_MYDEV_Write to block (be suspended by the OS scheduler) until some other thread or ISR posts the semaphore.

Presumably, this fictitious device will set an interrupt flag when its transmitter FIFO is empty (or as shown in a following section, a given watermark level). If that is the case, the driver's tasks function will need to call either the OSAL_SEM_Post or OSAL_SEM_PostISR function (depending upon whether or not it is configured for polled or interrupt-driven operation), passing in the `txSemaphore`, to signal that the transmitter is ready to accept more data. When that happens, the previous call to OSAL_SEM_Pend will return with an OSAL_RESULT_TRUE value. This causes the loop to continue until it has successfully transmitted all data (when `count` equals `size`) or until OSAL_SEM_Pend returns some other value.

If this code is built in a non-RTOS configuration, the OSAL_SEM_Pend function will instead return OSAL_RESULT_FALSE. When that occurs, the example will break out of the loop and the DRV_MYDEV_Write function will return the current count of how much data was successfully written to the device's transmitter FIFO. While not ideal (after all, the caller was hoping all of its data would be written), this behavior is still consistent with the expected behavior of the DRV_MYDEV_Write function and is completely safe so long as the caller appropriately checks the return value and adjusts accordingly.

When using a technique like this, blocking behavior becomes an optimization that is available when a RTOS is used, which is why the naturally blocking file system style read and write functions operate best in a RTOS environment and they are a bit inefficient (but still work) in a bare metal environment. Similar techniques can be used to synchronize between any interface functions and the associated tasks function(s) in any driver (or other library). Using this technique also illustrates that a driver is best designed to block in its interface functions and not in its tasks function(s). Using this technique best synchronizes clients, calling a driver's interface routines, with the operation of the driver's state machine.

Configuration and Implementations

Describes the different configuration capabilities that a MPLAB Harmony device driver developer must comprehend.

Description

MPLAB Harmony is very flexible and can support a number of ways in which a device driver (or other library) may be configured and customized to suit the individual needs of a specific system. These usually methods fall into one of the following categories.

- Optional Feature Sets
- Configuration Options
- Multiple Implementations

These different methods are described in the following sections. However, there are two important overriding concerns that the driver (or other library) developer must keep in mind when utilizing this flexibility.

First, all options should be managed and presented to the user for selection by the MPLAB Harmony Configurator (MHC). The MHC is the utility that manages MPLAB Harmony libraries and integrates them into the MPLAB X IDE development environment. While it is certainly possible to develop MPLAB Harmony-compatible drivers and add them to an application directly (in source code) with no MHC integration, doing so is missing out on the power and convenience of the MHC and will quickly and inevitably result in a need to manage changes in less effective ways. Adding MHC support for the libraries and their configuration options will dramatically simplify the tasks necessary to manage the addition, removal, and configuration of a driver (or any library) to one or more MPLAB Harmony projects and is will worth the effort.

Note: Please refer to the MPLAB Harmony Configurator Developer's Guide for information on how to support the MHC.

Second, the most important thing about any configuration or implementation of a MPLAB Harmony driver (or other library) is that its interface must stay consistent with the interface of all other configurations and implementations of the same driver (or library). This does not mean that every implementation of a driver must support every function defined in the driver's interface (see optional features). But, it does mean that if an implementation supports a feature then it must provide the same interface to that feature that all other implementations or configurations provide to that feature. An implementation of a driver that does not follow the interface defined in the help and by the driver's interface header is not an alternate implementation or configuration of the same driver. It is an entirely different driver or library.

These two considerations are important to all driver development. If they are not properly comprehended, the end result cannot be easily managed in the same way as other MPLAB Harmony drivers. The user will not be able to treat the driver as a building block module and he will likely need to either modify his application to reuse the driver or directly modify the driver itself. Properly managing these concerns is vital to developing highly reusable MPLAB Harmony drivers.

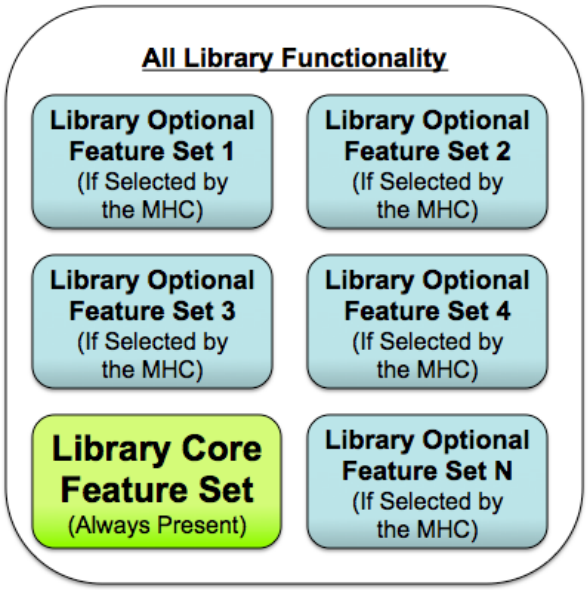
Optional Feature Sets

Describes how to group MPLAB Harmony driver features into sets and manage them as build configuration options.

Description

Any MPLAB Harmony library or driver should have a core feature set and interface to that feature set. This is the simplest and most basic functionality that is always provided by that driver, without which, there is no reason to use the driver in a system. All other features that may be provided by the driver can be considered optional and should be broken into sets, as shown in the following diagram.

Library Feature Sets



These feature sets should be identified and described in the **Configuring the Library** section of the driver's Help documentation, along with a description of how to select and configure each feature set. Any implementation of a driver that supports a feature set should support all features that are part of the set. Configuration options may affect how that feature set is supported (for example, buffer sizes used or minimums and maximums supported), but the inclusion or exclusion of that feature must happen as a single unit. Either all features in a set are included in the project when support for the feature is selected or they are all excluded from the project, based on a single MHC selection. Driver implementations that support *rogue* features that are undocumented or are not part of any defined feature set dramatically complicate the usage of the driver, make it difficult to represent its configuration in the MHC and prevent the user from treating different implementations of the driver as a simple building blocks.

In general, it is best to develop a driver so that it builds upon the core feature set in a clean and modular way. A good way to do this is to think of a feature set as a sub-module of its own and implement all code for that one feature set in a common source file. The source file for the core feature set will always be included in the project whenever the driver is included. The source file for an optional feature can then be included or excluded from the project, depending on whether or not the user selects the feature, as shown by the following examples.

Example: Core Feature (drv_mydev.c)

```
MY_RETVAL DRV_MYDEV_CoreFunction1 ( DRV_MYDEV_HANDLE handle, DRV_MYDEV_A *data )
{
    /* Implementation of core interface function 1 */
}

MY_RETVAL DRV_MYDEV_CoreFunction2 ( DRV_MYDEV_HANDLE handle, DRV_MYDEV_B *data )
{
    /* Implementation of core interface function 2 */
}

/* Additional core interface and internal functions... */

void __attribute__((weak)) _DRV_MYDEV_TasksOpt1 ( DRV_MYDEV_OBJ obj )
{
    return;
}

void DRV_MYDEV_Tasks ( DRV_MYDEV_OBJ obj )
{
    /* Implementation of core tasks state machine. */

    _DRV_MYDEV_TasksOpt1(obj);
}
```

Example: Optional Feature (drv_mydev_opt1.c)

```
MY_RETVAL DRV_MYDEV_Option1Function1 (DRV_MYDEV_HANDLE handle, DRV_MYDEV_A *data )
{
    /* Implementation of option 1 interface function 1 */
}

MY_RETVAL DRV_MYDEV_InterfaceFunction2 ( DRV_MYDEV_HANDLE handle, DRV_MYDEV_B *data )
{
    /* Implementation of option 1 interface function 2 */
}

/* Additional option 1 interface and internal functions... */

void _DRV_MYDEV_TasksOpt1 ( DRV_MYDEV_OBJ obj )
{
    /* Implementation of optional feature set 1 state machine. */
}
```

In the previous examples, the any interface or internal functions that are specific to the optional feature are implemented in a separate source file (drv_mydev_opt1.c) from the core feature set functions implemented in the driver's primary source file (drv_mydev.c). The core driver implementation file is always included in a project if the *mydev* driver is used. If the optional feature set is selected, the optional source file is also included. This provides implementations of the optional interface functions (DRV_MYDEV_InterfaceFunction1 and DRV_MYDEV_InterfaceFunction1, for example) as well as any internal and tasks or sub-tasks functions.

The _DRV_MYDEV_TasksOpt1 function shows the technique of using a weak function in the core feature set's implementation file to implement a sub-tasks state machine function. This can then be called from within the main state machine's tasks function. If the optional feature file (drv_mydev_opt1.c) is not included in the build, the weak implementation of the function will be called (and likely removed from the build, depending on the level of optimization chosen). However, if the optional feature set is selected and the implementation of the _DRV_MYDEV_TasksOpt1 function is built, the linker will override the weak definition in drv_mydev.c and link the call to the full non-weak implementation in the drv_mydev_opt1.c file calling the optional feature's state machine when the driver's core state machine is called.

This is the preferred method for implementing optional features (in separate files), but if this method is not feasible or if it adds more complexity than it saves, it is acceptable to use preprocessor macros to switch implementations of a function or short sequence of code based upon the selection of a configuration option, using the mapping method shown in the following example.

Example: Macro Mapping Function Implementations

```
#if defined(DRV_MYDEV_USE_OPT1)

    #define _DRV_MYDEV_TasksOpt1(obj)    _DRV_MYDEV_TasksOpt1Implementation(obj)

#else

    #define _DRV_MYDEV_TasksOpt1(obj)

#endif
```

This method allows the optional function's code to be removed even in builds with no optimizations, but it is more confusing and can be harder to debug. It is most useful an optional feature requires one or two short code sequences, when the first method is too much. This method can also be

used to wrap data allocations (see the OSAL_MUTEX_DECLARE and OSAL_SEM_DECLARE macros for examples).

In general, it is best to work to minimize the use of preprocessor `#if` directives as much as possible as they tend complicate the code and obfuscate the logic. If they must be used (as shown previously), it is best to group them into a single local mapping header included (for example the previous macros might be defined in a `drv_mydev_local_mapping.h` file). If they must be used directly in source code, it is best to indent them and the contents as if they were normal `if` statements. Do not force them to align at column 0. That is an old C-language standard that is no longer required and only serves to render code harder to read.

Configuration Options

Describes how to create and manage static build-time configuration options for MPLAB Harmony drivers.

Description

One of the primary ways in which MPLAB Harmony libraries are configured is by the definition and utilization of static configuration options defined at build-time. In most cases, these configuration options take the form of name-value pairs, defined in the system-wide `system_config.h` header using C preprocessor `#define` statements, as shown in the following example.

Example: Static Configuration Option Definitions

```
/* DRV USART Configuration Options */
#define DRV_USART_QUEUE_DEPTH_COMBINED              20
#define DRV_USART_CLIENTS_NUMBER                    6
#define DRV_USART_INSTANCES_NUMBER                  2

/* DRV USART 0 Initialization */
#define DRV_USART_PERIPHERAL_ID_IDX0                USART_ID_2
#define DRV_USART_BRG_CLOCK_IDX0                    80000000
#define DRV_USART_BAUD_RATE_IDX0                    9600

/* DRV USART 1 Initialization */
#define DRV_USART_PERIPHERAL_ID_IDX0                USART_ID_2
#define DRV_USART_BRG_CLOCK_IDX0                    80000000
#define DRV_USART_BAUD_RATE_IDX0                    9600
```

These macros are utilized in one of two ways.

- To define implementation-specific options
- To define instance-specific options

Implementation-specific macros are often used to control allocation of internal arrays or buffers and to control logic that manages them, as shown by the following example.

Example: Using Implementation-Specific Options

```
DRV_USART_BUFFER_OBJ gDrvUSARTBufferObj[DRV_USART_QUEUE_DEPTH_COMBINED];
unsigned int i;

/* Search the buffer pool for a free buffer object */
for(i = 0; i < DRV_USART_QUEUE_DEPTH_COMBINED; i++)
{
    if(!gDrvUSARTBufferObj[i].inUse)
    {
        /* Initialize buffer object. */
        gDrvUSARTBufferObj[i].inUse = true;
        break;
    }
}

if(i >= DRV_USART_QUEUE_DEPTH_COMBINED)
{
    /* Could not find a buffer. */
}
```

In the previous example, the USART driver keeps a common pool of buffer objects in an array, the size of which is determined by the DRV_USART_QUEUE_DEPTH_COMBINED configuration option. When a buffer object is needed, the driver logic searches through the array, looking for one that has not been allocated by checking its `inUse` flag. This option affects both the amount of RAM statically allocated by this driver and the code generated when it is built.

Note: This sequence of code is simplified for explanation. A real implementation would need to perform additional initializations and be protected for interrupt and thread safety.

Instance specific macros are used in `system_init.c` to initialize a dynamic driver's `init` data structure, as shown in the following example, or are built directly into instance-specific static driver implementation, using the same method shown previously.

Example: Using Instance-Specific Options

```
const DRV_USART_INIT drvUsart0InitData =
{
    .usartID  = DRV_USART_PERIPHERAL_ID_IDX0,
    .brgClock = DRV_USART_BRG_CLOCK_IDX0,
    .baud     = DRV_USART_BAUD_RATE_IDX0,

    /* Initialize other “init” data members. */
};

sysObj.drvUsart0 = DRV_USART_Initialize(DRV_USART_INDEX_0,
                                        (SYS_MODULE_INIT *)&drvUsart0InitData);
```

Since the user must provide the values of these options, the MHC must be made aware of them To make the MHC aware of an option, you must provide the appropriate *config* definitions in the Hconfig file hierarchy. And, to enable code generation based on these options, you must develop the necessary FreeMarker templates as described in the MPLAB Harmony Configurator Developer's Guide. Please refer to that document for details on developing Hconfig and FreeMarker template files.

Additionally, since static configuration options are defined in the `system_config.h` header, there are a few key guidelines governing this file to keep in mind.

Key `system_config.h` Guidelines:

- Any driver (or other source file) that uses any build-time configuration options supported by the MHC must include this header. The MHC always adds the path to this header to the compiler's include file search path. So, it is included without any path information (e.g., `#include "system_config.h"`).
- The `system_config.h` header must not contain any data type or function prototypes definitions. It must only define pre-processor name-value macros or header file include file dependency loops that cannot be resolved may occur.
- It is acceptable for configuration options to utilize symbol names that have not yet been defined because the macros it defines are not instantiated until used in a source file

Note: Type definitions used by system configuration code are defined in the `system_definitions.h` header file. This file should only be used by the system configuration code as it defines data types and external references for system configuration code.

While it is possible to define callable macros that emulate functions (or that map function call names and parameters to different selectable functions) in the `system_config.h`, it is best to implement such macros in the library code and select their implementations based upon name-value macro definition(s) defined by MHC, as shown in the following example. This simplifies the configuration files and template code and keeps the knowledge and complexity of the macro's implementation encapsulated in the library.

Example: Defining "Callable" Macros

```
#if (SAMPLE_MODULE_INTERRUPT_MODE == true)

    #define _InterruptDisable(s)    SYS_INT_SourceDisable(s)

#else

    #define _InterruptDisable(s)    false

#endif
```

The previous example shows how to map a local function used to disable the sample module's interrupt source to the appropriate system service when the library is configured for interrupt-driven operation (`SAMPLE_MODULE_INTERRUPT_MODE == true`) or to an implementation that provides an appropriate return constant when it is not. This technique allows the driver developer to capture the knowledge of the necessary implementation variants while providing the higher-level choice (of interrupt-driven or polled, in this example) to the user.

Multiple Implementations

Describes how to define multiple implementations of MPLAB Harmony drivers.

Description

As described in Interface vs. Implementation, the features and functionality provided by a driver are defined by its interface, not by any specific implementation of that driver. And, due to the flexibility and configurability provided by MPLAB Harmony and the MHC, it is possible to provide multiple implementations of the driver that are optimized for different hardware or different purposes. To support the selection and management of such implementations, the MHC provides the ability to select different source files, based upon configuration choices made by the user. (Refer to the MPLAB Harmony Configurator Developer's Guide for instructions on how to develop the Hconfig files to support this capability.)

Because different implementation variants provided the same interface (and thus, define the same interface functions), they are mutually exclusive. Only one implementation of a specific diver (or other library) can be included in the system at a time. Variant implementations of a MPLAB Harmony driver (or other library) are usually defined for one of three reasons:

- Targeted/optimized usage
- Integration of dependencies
- Static implementations

A targeted implementation is optimized for a specific usage or hardware selection. It can make use of hardware acceleration (for example built-in

DMA support) or simply eliminate functionality that is not required for a specific usage.

Integrated implementations may combine multiple libraries or driver *stacks* into a single driver, improving efficiency and reducing code size at the cost of the flexibility provided by a stack. For example, a SPI Codec driver would normally utilize the SPI driver to access its Codec over the SPI bus so that it can switch SPI bus drivers, if necessary. However, a integrated SPI Codec driver may directly utilize the SPI peripheral itself, integrating the SPI driver functionality to save code size and achieve better performance at the cost of being able to switch to a different type of SPI peripheral.

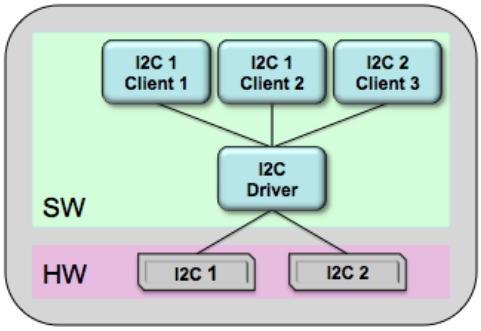
Refer to Static Implementations for details.

Implementing Multiple Client Drivers

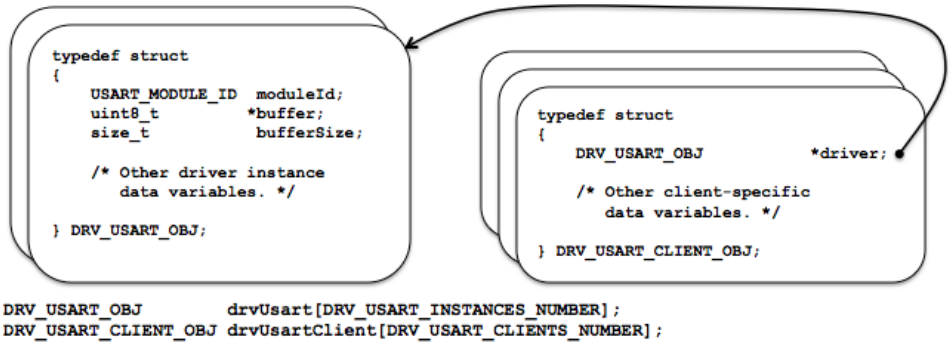
Describes how to implement multiple client drivers.

Description

When the concept of managing multiple clients is combined with the concept of managing multiple instances, the full picture of a MPLAB Harmony driver emerges, as shown in the following diagram. See Single Client vs. Multiple Client and Implementation vs. Instances for more information.



One method of implementing the ability to manage multiple instances of the peripheral hardware and multiple independent clients within the same driver is described in the following diagram, using the familiar USART as an example. Other methods may be possible, but this example illustrates the requirement.



The diagram shows the definitions of two data structures. The DRV_USART_OBJ structure is used to store all of the data required to manage a single instance of the peripheral hardware. (This structure is described in the Implementation vs. Instances section.) The DRV_USART_CLIENT_OBJ structure stores all of the data required to keep track of an individual client.

Since the driver manages multiple instances of the peripheral, there will be one instance of the DRV_USART_OBJ structure per peripheral instance, as defined by the DRV_USART_INSTANCES_NUMBER configuration parameter and allocated by the `drvUsart` array shown previously. Since the driver manages multiple clients, there will also be a number of client data structures, as defined by the DRV_USART_CLIENTS_NUMBER configuration parameter and allocated by the `drvUsartClient` array. One structure form the array will be assigned to each client that calls the driver's open function until they are all allocated.

One item of particular importance in the client object structure is the pointer that associates a client with the driver instance. This pointer (and usually other data) will be initialized when a client calls the driver's open function. The following example shows a possible implementation of this function (assuming the previous structure definitions).

Example: Driver Open Function

```
DRV_HANDLE DRV_USART_Open( const SYS_MODULE_INDEX index,
                           const DRV_IO_INTENT    ioIntent )
{
    int i;
    DRV_USART_CLIENT_OBJ pClient = (DRV_USART_CLIENT_OBJ *)DRV_HANDLE_INVALID;

    for (i=0; i < DRV_USART_CLIENTS_NUMBER; i++)
    {
        if (drvUsartClient[i].driver == NULL)
```

```
        {
            pClient = &drvUsartClient[i];
            pClient->driver = &drvUsart[index];
            break;
        }
    }

    return (DRV_HANDLE)pClient;
}
```

This implementation of the DRV_USART_Open function does a linear search through the array of client objects. The first one it finds with a NULL driver pointer is assumed to be unallocated and available for use. It then assigns the address of the driver object structure in the `drvUsart` array that is identified by the function's index parameter. Doing this simultaneously allocates that client object and associates it with the specified driver instance. An open function would normally store some additional data and maybe do some other preparation to get ready to service the client, but this simple example shows how a unique opened driver handle can be created that identifies a client, how a client object structure might provide a storage location for client-specific data, and how the driver can associate the handle with a specific instance of the driver and peripheral.

Note: This example is not RTOS safe. A RTOS safe implementation would protect the for loop with a mutex. Refer to the Interrupt and Thread Safety section for more in formation on RTOS safety.

Once the driver has been opened and the association between the client and the driver instance has been made, the driver handle can be returned to the client and then later be used by other client interface functions to interact with the peripheral safely, as shown in the following example.

Example: Client API Function Implementation

```
void DRV_USART_BufferAddRead ( const DRV_HANDLE handle,
                               DRV_USART_BUFFER_HANDLE * const bufferHandle,
                               void * buffer, const size_t size )
{
    DRV_USART_CLIENT_OBJ *client = (DRV_USART_CLIENT_OBJ *)handle;
    DRV_USART_OBJ        *driver = (DRV_USART_OBJ *)client->driver;

    if (driver->buffer == NULL)
    {
        driver->buffer       = buffer;
        driver->bufferSize   = size;
        bufferHandle         = buffer;

        /* Start the data transfer process. */

    }
    else
    {
        *bufferHandle = DRV_USART_BUFFER_HANDLE_INVALID;
    }

    return;
}
```

This example shows how a buffer queuing read might work. However, it is somewhat oversimplified because it only maintains a queue size of one. This is because the driver structure only keeps a single buffer pointer and `bufferSize` variable (instead of a queue of several of them). So, in this example, the driver checks to see if its buffer pointer is NULL (which, presumably, that is how it was initialized during the driver's initialize function and how it is reset whenever a transfer completes). However, if the buffer pointer is not NULL, it means that the driver is currently busy transferring a previous request. This causes the driver to return an invalid buffer handle value (DRV_USART_BUFFER_HANDLE_INVALID) to the caller. When this happens, the client calling will have to try again later because the queue (of one entry) is full. If the buffer pointer is NULL, it is not in use (i.e., the queue is not full) and the function will save the caller's buffer pointer and size information and do whatever is necessary to start the data transfer, likely interacting with the hardware through the peripheral library at that point.

While this example is incomplete and somewhat limited, it does demonstrate how a client API function might use the opened driver handle to identify the link to the peripheral instance and prevent conflicts between peripherals. In this case, a rather brute force method of only allowing a single transfer to occur at a time is used; but, that method is completely valid as regardless of how big the queue is, it can always become full. However, more sophisticated methods (such as implementing an actual transfer queue) would let the driver provide better throughput.

Static Implementations

Describes how to develop static MPLAB Harmony driver implementations.

Description

As described in Static vs. Dynamic, a dynamic driver implementation manages multiple instances of a particular type of peripheral and a static driver implementation only manages one. This allows a single static implementation of a driver to be smaller than the equivalent dynamic

implementation, saving code space by hard coding values that can be made constant. The following examples show the basic differences between the two types of implementations.

Example: Dynamic Implementation

```
SYS_MODULE_OBJ DRV_USART_Initialize ( const SYS_MODULE_INDEX  index,
                                      const SYS_MODULE_INIT * const init )
{
    DRV_USART_OBJ  *pObj  = (DRV_USART_OBJ *)&gDrvUsartObj[index];
    DRV_USART_INIT *pInit = (DRV_USART_INIT *)init;

    /* Initialize data for this instance */
    pObj->usartId               = pInit->usartId;
    pObj->interruptSourceTx     = pInit->interruptSourceTx;
    pObj->interruptSourceRx     = pInit->interruptSourceRx;
    pObj->interruptSourceErr    = pInit->interruptSourceErr;
    pObj->queueSizeCurrentRead  = 0;
    pObj->queueSizeCurrentWrite = 0;
    pObj->queueRead             = NULL;
    pObj->queueWrite            = NULL;

    /* Initialize USART Hardware */
    PLIB_USART_Disable(pObj->usartId);
    PLIB_USART_HandshakeModeSelect(pObj->usartId, pInit->handshake);
    PLIB_USART_BaudSetAndEnable(pObj->usartId, pInit->brgClock, pInit->baud);
    PLIB_USART_LineControlModeSelect(pObj->usartId, pInit->lineControl);

    /* Clear and enable the interrupts */
    SYS_INT_SourceStatusClear(pObj->interruptSourceTx);
    SYS_INT_SourceStatusClear(pObj->interruptSourceRx);
    SYS_INT_SourceStatusClear(pObj->interruptSourceErr);
    _InterruptSourceEnable(pObj->interruptSourceErr);

    /* Ready! */
    pObj->status = SYS_STATUS_READY;
    PLIB_USART_Enable(pObj->usartId);
    return (SYS_MODULE_OBJ)pObj;
}
```

The previous dynamic example shows that the driver's initialization function must capture any initialization data that could be different from one instance to another and that it must use a pointer (`pObj`) to the desired instance of its global data object/structure (`gDrvUsartObj[index]`) and clear the references that point to access of any global data in the object. It must also clear the reference to the `init` pointer (cast to `pInit`) to access any initialization data, whether or not it is stored in the global data object.

Note: The _InterruptSourceEnable function is a locally mapped function that switches implementations depending upon whether or not the driver was built in Interrupt-driven mode or Polled mode, as shown in Interrupt Safety.

The following equivalent static example shows how a static implementation saves both code and data space.

Example: Static Implementation

```
void DRV_USART0_Initialize ( void )
{
    /* Initialize data for this instance */
    gDrvUsart0Obj.queueSizeCurrentRead  = 0;
    gDrvUsart0Obj.queueSizeCurrentWrite = 0;
    gDrvUsart0Obj.queueRead             = NULL;
    gDrvUsart0Obj.queueWrite            = NULL;

    /* Initialize USART Hardware */
    PLIB_USART_Disable(DRV_USART_ID_IDX0);
    PLIB_USART_HandshakeModeSelect(DRV_USART_ID_IDX0,
                                   DRV_USART_HANDSHAKE_MODE_IDX0);
    PLIB_USART_BaudSetAndEnable(DRV_USART_ID_IDX0,
                                DRV_USART_BRG_CLOCK_IDX0,
                                DRV_USART_BAUD_RATE_IDX0);
    PLIB_USART_LineControlModeSelect(DRV_USART_ID_IDX0,
                                     DRV_USART_LINE_CNTRL_IDX0);

    /* Clear and enable the interrupts */
    SYS_INT_SourceStatusClear(DRV_USART_INT_SRC_TX_IDX0);
    SYS_INT_SourceStatusClear(DRV_USART_INT_SRC_RX_IDX0);
    SYS_INT_SourceStatusClear(DRV_USART_INT_SRC_ERR_IDX0);
```

```
    _InterruptSourceEnable(DRV_USART_INT_SRC_ERR_IDX0);

    /* Ready! */
    gDrvUsart0Obj.status = SYS_STATUS_READY;
    PLIB_USART_Enable(DRV_USART_ID_IDX0);
}
```

In the previous static example, the function would be implemented differently for different driver instances (DRV_USART0_Initialize, DRV_USART1_Initialize, etc.). So, any initialization data that is different from one instance to another can be defined by different configuration macros (such as DRV_USART_ID_IDX0) and hard-coded directly into the function's implementation. This reduces code and data size because it eliminates the need to store these items in the driver's global data structure instance and it eliminates the need to clear the reference to a pointer and access a variable when using these values.

And, using a constant instead of a variable greatly reduces the amount of code generated by PLIB functions because PLIB functions are implemented as C-language inline functions. When a constant is passed to an inline function, the compiler can optimize them by performing calculations before generating the object code instead of generating object code instructions to do the calculations. This eliminates a significant amount of object code, especially when each PLIB function would otherwise index to the appropriate SFRs for the instance of the peripheral passed in as a variable.



Note: The mapping functions shown in Static vs. Dynamic shows how the parameters are dropped and the return value is provided. Refer to this section for an explanation and example of mapping the dynamic driver interface functions to the static implementation functions.

Creating a static driver implementation requires development of a FreeMarker template. (Note that it does not require any additional Hconfig file development, since the dynamic and static implementations both utilize the same configuration options.) Creating a static implementation from a dynamic implementation is primarily a matter of removing the unnecessary code and marking up the dynamic implementation using FreeMarker syntax to parameterize the driver's source code and insert the appropriate values where necessary, as shown by the following example.

Example: FreeMarker Code for Static Implementation

```
<#macro make_drv_usart_initialize_function DRV_INSTANCE>
void DRV_USART${DRV_INSTANCE}_Initialize ( void )
{
    /* Initialize data for this instance */
    gDrvUsart${CONFIG_DRV_INSTANCE}Obj.queueSizeCurrentRead  = 0;
    gDrvUsart${DRV_INSTANCE}Obj.queueSizeCurrentWrite = 0;
    gDrvUsart${DRV_INSTANCE}Obj.queueRead             = NULL;
    gDrvUsart${DRV_INSTANCE}Obj.queueWrite            = NULL;

    /* Initialize USART Hardware */
    PLIB_USART_Disable(DRV_USART_ID_IDX${DRV_INSTANCE});
    PLIB_USART_HandshakeModeSelect(DRV_USART_ID_IDX${DRV_INSTANCE},
                                   DRV_USART_HANDSHAKE_MODE_IDX${DRV_INSTANCE});
    PLIB_USART_BaudSetAndEnable(DRV_USART_ID_IDX${DRV_INSTANCE},
                                DRV_USART_BRG_CLOCK_IDX${DRV_INSTANCE},
                                DRV_USART_BAUD_RATE_IDX${DRV_INSTANCE});
    PLIB_USART_LineControlModeSelect(DRV_USART_ID_IDX${DRV_INSTANCE},
                                     DRV_USART_LINE_CNTRL_IDX${DRV_INSTANCE});

    /* Clear and enable the interrupts */
    SYS_INT_SourceStatusClear(DRV_USART_INT_SRC_TX_IDX${DRV_INSTANCE});
    SYS_INT_SourceStatusClear(DRV_USART_INT_SRC_RX_IDX${DRV_INSTANCE});
    SYS_INT_SourceStatusClear(DRV_USART_INT_SRC_ERR_IDX${DRV_INSTANCE});
    _InterruptSourceEnable(DRV_USART_INT_SRC_ERR_IDX${DRV_INSTANCE});

    /* Ready! */
    gDrvUsart${DRV_INSTANCE}Obj.status = SYS_STATUS_READY;
    PLIB_USART_Enable(DRV_USART_ID_IDX${DRV_INSTANCE});
}

</#macro>
<#list index = 0..CONFIG_DRV_USART_INSTANCES_NUMBER>
  <@make_drv_usart_initialize_function DRV_INSTANCE=index/>
</#list>
```

The previous example defines a FreeMarker macro called make_drv_usart_initialize_function to define the static (instance specific) USART driver initialization functions. Within that function, it uses a local FreeMarker macro variable (DRV_INSTANCE) to indicate the driver's instance index. It places this variable within the function's source code wherever an instance index number is required. Then, it defines a list that increments from 0 to less than the CONFIG_DRV_USART_INSTANCES_NUMBER value defined by the MHC when the user selected how many USART driver instances he wanted. From within that list, it calls the macro, passing in the index value iterated by the list, generating as many static implementations of the USART driver's initialize function as desired.

The previous examples are simplified to aid understanding. Implementing an entire static driver may become more complicated, requiring some

skill with the FreeMarker language. However, the principles remain the same. All configuration variables defined by the MHC are available to the FreeMarker code and may be used as needed. Refer to MPLAB Harmony Configurator Developer's Guide for details on the Hconfig and FreeMarker languages necessary to develop static driver implementations.

Multiple Client Static Drivers

Describes multiple client static drivers.

Description

Supporting multiple clients also has an affect on how static drivers are implemented, particularly if mapping functions are used (see the Static vs. Dynamic section for a description of a static implementation). Other than the open function, all client interface functions require the use of an opened driver. For static implementations, this can be done the same way it is done in a dynamic implementation. The main difference is that the driver object instance is identified in the driver structure name, so the index parameter is ignored, as shown in the following example.

Example: Static Multiple Client Open Function

```
DRV_USART_OBJ        drvUsart0;
DRV_USART_CLIENT_OBJ drvUsartClient[DRV_USART_CLIENTS_NUMBER];


DRV_HANDLE DRV_USART0_Open( const SYS_MODULE_INDEX index,
                            const DRV_IO_INTENT ioIntent )
{
    int i;
    DRV_USART_CLIENT_OBJ pClient = (DRV_USART_CLIENT_OBJ *)DRV_HANDLE_INVALID;

    for (i=0; i < DRV_USART_CLIENTS_NUMBER; i++)
    {
        if (drvUsartClient[i].driver == NULL)
        {
            pClient = &drvUsartClient[i];
            pClient->driver = &drvUsart0;
            break;
        }
    }

    return (DRV_HANDLE)pClient;
}
```

If multiple static implementations are defined, the static driver's client API mapping functions can use the driver object pointer in the client object structure to identify driver instance. The following example shows how this can be done.

Example: Multiple Client Dynamic-to-Static Mapping Function

```
inline void DRV_USART_BufferAddRead ( const DRV_HANDLE handle,
                                      DRV_USART_BUFFER_HANDLE * const bufferHandle,
                                      void * buffer, const size_t size )
{
    DRV_USART_CLIENT_OBJ *client = (DRV_USART_CLIENT_OBJ *)handle;

    switch (client->driver)
    {
        case &drvUsart0:
        {
            DRV_USART0_BufferAddRead(handle, bufferHandle, buffer, size);
            break;
        }

        case &drvUsart1:
        {
            DRV_USART1_BufferAddRead(handle, bufferHandle, buffer, size);
            break;
        }

        default:
        /* invalid instance. */
    }

    return;
}
```

However, as stated in the Static vs. Dynamic section, it normally does not make sense to implement multiple static driver instances for the same system. (If you need multiple instances of a driver, using a dynamic implementation would be more efficient.) So, if only a single instance of a

static driver is implemented and if static and dynamic driver implementations are never used together in the same system (which they should not be because nothing is gained), the open function can be simplified by removing the driver object pointer altogether, as shown in the following example.

Example: Simplified Static Multiple Client Open Function

```
DRV_USART_OBJ        drvUsart0;
DRV_USART_CLIENT_OBJ drvUsartClient[DRV_USART_CLIENTS_NUMBER];


DRV_HANDLE DRV_USART_Open( const SYS_MODULE_INDEX index,
                           const DRV_IO_INTENT ioIntent )
{
    int i;
    DRV_USART_CLIENT_OBJ pClient = (DRV_USART_CLIENT_OBJ *)DRV_HANDLE_INVALID;

    for (i=0; i < DRV_USART_CLIENTS_NUMBER; i++)
    {
        if (drvUsartClient[i].driver == NULL)
        {
            pClient = &drvUsartClient[i];
            break;
        }
    }

    return (DRV_HANDLE)pClient;
}
```

Note: If this method is used, some method of marking a client object structure as assigned to a client, like an "in use" Boolean flag, must be used because you cannot rely on a NULL value for the driver pointer as the indicator.

Also, the mapping function is unnecessary because there is only one mapping. Instead, the client API functions can utilize the exact same names as the dynamic driver's client API functions (meaning both cannot be used together in the same system), as shown by the following example.

Example: Simplified Static Client API Function Implementation

```
void DRV_USART_BufferAddRead ( const DRV_HANDLE handle,
                               DRV_USART_BUFFER_HANDLE * const bufferHandle,
                               void * buffer, const size_t size )
{
    DRV_USART_CLIENT_OBJ *client = (DRV_USART_CLIENT_OBJ *)handle;

    if (drvUsart0.buffer == NULL)
    {
        drvUsart0.buffer     = buffer;
        drvUsart0.bufferSize = size;
        bufferHandle         = buffer;

        /* Start the data transfer process. */

    }
    else
    {
        *bufferHandle = DRV_USART_BUFFER_HANDLE_INVALID;
    }

    return;
}
```

Notice that in this example, the client API function does not need to retrieve the driver instance structure pointer from the client object. Since there is only one driver object structure (`drvUsart0`), it is always used. No other driver object instance is possible. When used throughout the client interface routines in a static implementation of the driver, the elimination of repeated cleared pointer references will reduce the size of the generated object code.

Review and Testing

Describes how to test MPLAB Harmony drivers and references the resources available to help.

Description

The MPLAB Harmony Driver Development Guide (this document) describes how to develop a MPLAB Harmony device driver. To do so, it explains specific the requirements of a MPLAB Harmony driver, including support for the system and client interfaces as well as key design concepts, interrupt and thread safety concerns, and support for multiple configurations and implementations. In addition to this guide, the MPLAB Harmony

Compatibility Guide describes general modularity and flexibility guidelines and provides a compatibility checklist worksheet, available in a separate fillable PDF form in the following documentation folder in the MPAB Harmony installation.

MPLAB Harmony Compatibility Checklist Worksheet Location

`<install-dir>/doc/harmony_compatibility_worksheet.pdf`

Any MPLAB Harmony driver developed should be reviewed, tested and evaluated against the rules described in this guide and checklist. Providing a completed copy of this worksheet along with the documentation of any MPLAB Harmony compatible library will help the user to determine the environments and configuration limits supported by the library.

Additionally, each driver implementation and superset configuration should be thoroughly tested in all supported execution environments (bare metal polled, bare metal interrupt-driven, RTOS multi-threaded polled, RTOS multi-threaded interrupt-driven) to ensure correct and robust operation in all supported usages. To facilitate such testing, MPLAB Harmony provides a Test Harness that is a useful tool for developing and iterating through test in a controlled way that allows easy capturing of failures using MPLAB Harmony debug capabilities. The Test Harness is a library that is included in the MPLAB Harmony installation.

MPLAB Harmony Test Harness Library Location

`<install-dir>/framework/test`

Refer to the MPLAB Harmony Test Harness User's Guide for information on how to utilize the test harness to validate your drivers and libraries.

A good way to provide transparency to the customer is to provide the test applications and test results generated by them with the driver implementation. If a customer can reproduce the test results, it provides great confidence in your libraries and improves the customer's understanding of how the library works and how to use it.

Checklist and Information

Provides a quick reference checklist and important reference information for developing MPLAB Harmony drivers.

Description

At a high-level, the process of developing a MPLAB Harmony device driver is fairly simple. The following checklist describes the basic work flow. The individual steps are described in detail in other sections in this guide.

MPLAB Harmony Development Checklist

Done	Step	Description
☐	1	Define (and document) System interface functions.
☐	2	Define (and document) Client interface functions.
☐	2a	Define data transfer functions (following common models if appropriate).
☐	2b	Define driver-specific functions.
☐	3	Develop Hconfig and FreeMarker template support for initialization, tasks, and other system functions as needed to test and develop.
☐	4	Develop and test initialize, tasks, and core functions.
☐	4a	Start in a polled environment.
☐	4b	Update & test for interrupt-driven environment (retest polled).
☐	4c	Update & test OS support (retest polled & interrupt-driven).
☐	5	Define (and document) MHC support for static configuration options.
☐	6	Define (and document) optional feature sets and functions (one at a time).
☐	6a	Implement & test in all three environments (polled, interrupt-driven, & OS-driven).
☐	6b	Define MHC support for optional feature sets.
☐	7	Develop alternate implementations (particularly static/optimized implementations).

When Implementing a MPLAB Harmony driver, adhere to the following folder layout and file/folder naming conventions.

MPLAB Harmony Drier File and Folder Layout Conventions

Path	Description
`<library>`	Root folder of the MPLAB Harmony driver library. Contains all library header, source files, configuration, and template files.
`<library>/<library>.h`	Library interface header file.
`<library>/<library>_mapping.h`	Library interface level (dynamic-to-static) mapping file.
`<library>/src`	Library C-language source code folder.
`<library>/src/*.c, *.h`	Library implementation files and headers.

<library>/config	Library configuration folder.
<library>/config/*.h	C-language configuration header example (primarily for documentation purposes).
<library>/config/*.hconfig	MHC Hconfig files defining all library configuration options.
<library>/config/*.hconfig.ftl	MHC Hconfig files that are preprocessed by FreeMarker before being integrated into the Hconfig tree of MHC.
<library>/templates/*.ftl	MHC FreeMarker template files for MHC code generation of static implementations and system configuration code necessary integrate the library into a MPLAB Harmony system.

Index

1

12-bit High-Speed SAR ADC (ADCHS) Peripheral Library Examples 215

A

a2dp_avrcp 160
ADC Peripheral Library Examples 213
adc_pot 213
adc_pot_dma 214
adchs_3ch_dma 215
adchs_oversample 217
adchs_pot 218
adchs_sensor 220
adchs_touchsense 221
adcp_cal 223
Adding a New Font File to the Application 452
Additional Bluetooth Resources 104
Applications Help 3
aria_adventure 303
aria_basic_motion 307
aria_benchmark 311
aria_coffee_maker 317
aria_counter 324
aria_external_resources 329
aria_flash 337
aria_image_viewer 344
aria_oven_controller 350
aria_quickstart 354
aria_radial_menu 378
aria_scrolling 385
aria_showcase 391
aria_showcase_reloaded 400
aria_splash_screen 410
aria_touchadc_calibrate 421
aria_video_player 429
aria_weather_forecast 438
Assigning the IDC and XC32 Compilers 605
Atomicity 687
Audio Demonstrations 3
audio_microphone_loopback 3
audio_speaker 653
audio_tone 7

B

basic 162, 487, 489, 496, 497, 499, 503
berkeley_tcp_client 507
berkeley_tcp_server 508
berkeley_udp_client 509
berkeley_udp_relay 511
berkeley_udp_server 512
blank_quickstart 444
ble_rn4871_comm 135
blinky_leds 250
Bluetooth Demonstrations 103
BM64 Driver Demonstrations 104
BM64_a2dp_hfp 104
BM64_ble_comm 109
BM64_bootloader 119
BMX Peripheral Library Examples 224
Bootloader Demonstrations 162
bt_data_voice_control 153
Buffer Queuing 685
Building the Application 6, 12, 21, 26, 31, 35, 42, 48, 56, 63, 69, 75, 81, 86, 93, 100, 106, 111, 121, 135, 141, 146, 153, 160, 162, 168, 170, 171, 172, 174, 179, 186, 192, 193, 195, 197, 198, 203, 206, 207, 208, 209, 214, 216, 217, 219, 220, 221, 223, 225, 226, 229, 235, 236, 238, 239, 240, 241, 243, 244, 246, 247, 249, 250, 251, 252, 254, 255, 257, 258, 259, 260, 262, 263, 265, 267, 274, 278, 280, 281, 282, 284, 285, 286, 288, 290, 292, 294, 297, 299, 301, 304, 308, 313, 320, 326, 333, 341, 346, 352, 368, 380, 389, 394, 403, 415, 426, 432, 441, 447, 450, 465, 469, 472, 475, 477, 480, 487, 488, 489, 490, 491, 493, 496, 497, 498, 500, 502, 503, 504, 507, 508, 509, 511, 512, 513, 516, 517, 519, 523, 524, 525, 526, 527, 537, 540, 542, 545, 550, 552, 555, 564, 568, 571, 577, 580, 585, 588, 606, 612, 613, 622, 629, 631, 632, 635, 636, 638, 641, 642, 644, 645, 646, 647, 651, 652, 654, 655, 656, 657, 659, 660, 661, 662, 666, 667, 668
Byte-by-Byte (Single Client) 683

C

Callback Functions 693
CAN Peripheral Library Examples 225
can_display 228
cdc_basic 655
cdc_com_port_dual 490, 500, 622, 666
cdc_com_port_single 628
cdc_msd 656
cdc_msd_basic 491, 502, 631
cdc_serial_emulator 632
cdc_serial_emulator_msd 635
Checklist and Information 707
Class B Library Demonstrations 170
ClassBDemo 171
Client Interface 680
Close 683
cn_interrupt 251
Command Processor System Service Examples 265
command_appio 265
Common Data Transfer Models 683
Comparator Peripheral Library Examples 234
Configuration and Implementations 696
Configuration Options 699
Configuring the Hardware 6, 13, 21, 26, 31, 36, 42, 48, 57, 63, 70, 75, 82, 86, 94, 100, 107, 112, 122, 136, 142, 146, 154, 161, 163, 169, 170, 171, 173, 174, 180, 187, 192, 194, 196, 198, 199, 203, 206, 207, 209, 210, 214, 215, 216, 217, 219, 220, 222, 223, 225, 226, 230, 235, 237, 238, 239, 241, 242, 244, 245, 246, 248, 249, 250, 251, 253, 254, 256, 257, 258, 259, 261, 262, 264, 266, 267, 275, 278, 281, 282, 283, 284, 285, 286, 289, 291, 293, 295, 297, 299, 301, 302, 305, 309, 314, 321, 326, 334, 341, 346, 353, 369, 381, 389, 394, 404, 416, 427, 432, 442, 447, 451, 466, 470, 473, 476, 477, 481, 487, 488, 490, 491, 492, 493, 497, 498, 499, 500, 501, 502, 504, 505, 507, 509, 510, 511, 513, 514, 516, 517, 519, 523, 525, 526, 527, 528, 537, 540, 543, 546, 551, 552, 556, 565, 568, 572, 578, 581, 585, 589, 607, 613, 614, 623, 629, 632, 633, 635, 637, 639, 641, 643, 644, 645, 647, 648, 651, 652, 654, 655, 657, 658, 659, 661, 662, 663, 666, 667, 668
    USB Device Demonstration (hid_msd_basic) 644
Configuring the MHC 532
Console System Service Examples 266
Crypto Demonstrations 172
CVREF Peripheral Library Examples 236

D

Data Demonstrations 135
Data EEPROM Driver Demonstration 191
data_basic 140
data_temp_sens_rgb 144
DDR Peripheral Library Examples 237
Debug System Service Examples 274
debug_uart 274
debug_usb_cdc_2 277
Demonstration Application Configurations 619
Demonstration Functionality 103
Demonstrations 3, 103, 162, 171, 172, 192, 193, 195, 197, 207, 208, 213, 215, 223, 225, 226, 235, 236, 237, 239, 240, 241, 243, 244, 246, 247, 248, 249, 252, 253, 255, 257, 260, 262, 263, 265, 267, 274, 280, 283, 286, 288, 303, 472, 506, 622
- ADC Peripheral Library 213
- ADCHS Peripheral Library 215
- Audio Demonstrations (audio_microphone_loopback) 3
- BMX Peripheral Library 225
- Bootloader 162
- CAN Library 226
- Class B Library 171
- Command Processor System Service Library 265
- Comparator Peripheral Library 235
- Console System Service Library 267
- Crypto Library 172
- CVREF Peripheral Library 236
- Data EEPROM Driver 192
- DDR Peripheral Library 237
- Debug System Service Library 274
- Device Control System Service Library 280
- DMA Peripheral Library 239
- DMA System Service Library 283
- EBI Peripheral Library 240
- File System 288
- Graphics Library 303
- I2C Driver 193
- I2C Peripheral Library 241
- Input Capture Peripheral Library 243
- Motor Control 472
- NVM Driver 195
- NVM Peripheral Library 244
- OSC Peripheral Library 247
- Output Compare Peripheral Library 246
- PIC32 Bluetooth Stack Library 135
- Pipelined ADC Peripheral Library 223
- PMP Peripheral Library 248
- Ports Peripheral Library 249
- Power Peripheral Library 252
- Reset Peripheral Library 253
- RTCC System Service Library 286
- SPI Driver 197
- SPI Flash Driver 207
- SPI Peripheral Library 255
- SQI Peripheral Library 257
- TCPIP 506
- TMR Peripheral Library 260
- USART Driver 208
- USART Peripheral Library 262
- USB 622
- WDT Peripheral Library 263

devcon_cache_clean 280
devcon_cache_invalidate 281
devcon_sys_config_perf 282
Device 622
Device Control System Service Examples 279
DMA Peripheral Library Examples 238
DMA System Service Examples 283
dma_crc 283
dma_led_pattern 239
dma_mem2mem 285
Driver Client Interface Functions 681
Driver Demonstrations 191
Driver-Client Usage Model 681
Dual Role 668
dualshunt_pll_foc_mclv2_ext_opamp 472
dualshunt_pll_foc_mclv2_int_opamp 475

E

EBI Peripheral Library Examples 240
ecc_asymmetric 177
ecc_symmetric 185
echo_send 226
eeprom_read_write 192
emwin_media_player 19
emwin_multilanguage 448
emwin_quickstart 463
emwin_showcase 468
encrypt_decrypt 172
Examples 213
Express Logic ThreadX Demonstrations 487

F

File System Demonstrations 287
File System Read/Write 684
Flash/NVM Peripheral Library Examples 244
flash_modify 244
flash_read_dma_mode 257
flash_read_pio_mode 258
flash_read_xip_mode 259
FreeRTOS Demonstrations 489

G

General Guidelines 686
Graphics Demonstrations 302

H

hid_basic 636
hid_basic_keyboard 657
hid_basic_mouse_usart 659
hid_joystick 638
hid_keyboard 641
hid_mouse 642
hid_msd_basic 644
Host 653
host_msd_device_hid 668
hub_cdc_hid 660

hub_msd 661

I

I2C Driver Demonstrations 193
I2C Peripheral Library Examples 241
i2c_interrupt 241
i2c_rtcc 193
ic_basic 243
Implementing Multiple Client Drivers 701
Input Capture Peripheral Library Examples 243
integrated_pfc_foc_mchv3_int_opamp 480
integrated_pfc_foc_mhcv3_ext_opamp 477
Interrupt and Thread Safety 687
Interrupt Safety 688
Introduction 3, 103, 162, 170, 172, 191, 193, 195, 197, 207, 208, 213, 215, 223, 224, 225, 235, 236, 237, 238, 240, 241, 243, 244, 246, 247, 248, 249, 252, 253, 255, 256, 260, 261, 263, 265, 266, 274, 280, 283, 286, 287, 302, 472, 486, 505, 615, 671
- Crypto Library Demonstrations 172
- Peripheral Library Example Applications 213
- PIC32 Bluetooth Stack Library Demonstrations 103
- USART Driver Demonstration 208

L

large_hash 173
LiveUpdate_App 168
LiveUpdate_Switcher 169

M

mac_audio_hi_res 23
mem_partition 225
Micrium uC/OS-III Demonstrations 497
Micrium uC_OS_II Demonstrations 496
Module Deinitialize 678
Module Initialization 673
Module Reinitialize 679
Module Status 676
Module Tasks 675
Motor Control Demonstrations 471
MPLAB Harmony Driver Development Guide 671
MPLAB Harmony WINC1500 Socket Examples 604
msd_basic 645, 662
msd_dual 667
msd_fs_spiflash 646
msd_multiple_luns 647
msd_sdcard 650
multi_instance_console 267
Multiple Client Static Drivers 705
Multiple Implementations 700
Multiple USB Controller 666
my_first_app 213

N

NVM Driver Demonstration 195
nvm_fat_single_disk 288
nvm_mpfs_single_disk 290
nvm_read_write 195
nvm_sdcard_fat_mpfs_multi_disk 292
nvm_sdcard_fat_multi_disk 294

O

oc_pwm 246
Open 682
OPENRTOS Demonstrations 499
Optional Feature Sets 697
Organization of WINC1500 Socket Examples 603
osc_config 247
Oscillator Peripheral Library Examples 247
Output Compare Peripheral Library Examples 246

P

Peripheral Library Examples 213
pic32_eth_web_server 539
pic32_eth_wifi_web_server 542
pic32_wifi_web_server 545
Pipelined ADC (ADCP) Peripheral Library Examples 223
PMP Peripheral Library Examples 248
pmp_lcd 248
Ports Peripheral Library Examples 249
Power Peripheral Library Examples 252
Premium Demonstrations 159
Prerequisites 604

R

real_time_fft 29
Reset Peripheral Library Examples 253
reset_handler 254
Review and Testing 706
RTCC System Service Examples 286
rtcc_timestamps 286
RTOS Demonstrations 486
RTOS Thread Safety 690
Running the Application 101
Running the Demonstration 7, 14, 22, 27, 32, 40, 43, 51, 58, 63, 71, 76, 82, 87, 95, 108, 113, 122, 137, 142, 146, 154, 161, 164, 169, 170, 172, 173, 176, 180, 187, 192, 194, 196, 198, 202, 205, 206, 207, 209, 212, 214, 215, 216, 218, 219, 221, 222, 224, 225, 227, 233, 235, 237, 238, 240, 241, 242, 244, 245, 247, 248, 249, 251, 252, 253, 254, 256, 258, 259, 261, 263, 264, 266, 268, 275, 278, 281, 282, 283, 284, 285, 287, 289, 291, 293, 295, 298, 300, 301, 302, 305, 309, 315, 321, 327, 335, 343, 347, 353, 377, 382, 390, 395, 405, 418, 427, 435, 443, 448, 462, 467, 470, 473, 476, 478, 481, 488, 490, 491, 492, 493, 497, 498, 499, 500, 501, 502, 504, 505, 508, 509, 510, 512, 513, 514, 516, 518, 520, 524, 525, 526, 527, 535, 538, 540, 544, 549, 551, 553, 561, 566, 569, 572, 578, 581, 586, 590, 611, 613, 614, 625, 630, 632, 633, 636, 637, 639, 642, 643, 644, 646, 647, 650, 651, 653, 654, 655, 657, 658, 660, 661, 662, 665, 666, 667, 668
- Audio Demonstrations (audio_microphone_loopback) 7
- Motor Control Demonstration (encrypt_decrypt) 473
- USB Device Demonstration (hid_keyboard) 641
- USB Device Demonstrations (audio_speaker) 654
- USB Device Demonstrations (hid_keyboard) 642

Running the Demonstration Using Various Configurations 595

S

sdcard_fat_single_disk 296
sdcard_msd_fat_multi_disk 299
sdcard_player 34
sdcard_usb_audio 40
SEGGER embOS Demonstrations 503
Selecting the Decoders Using MHC 50

serial_eeprom 197
simple_comparator 235
sleep_mode 252
snmpv3_nvm_mpfs 513
snmpv3_sdcard_fatfs 515
SPI Driver Demonstrations 196
SPI Flash Driver Demonstrations 206
SPI Peripheral Library Examples 255
spi_loopback 198, 255
spi_multislave 203
spi_self_loopback 205
SQI Peripheral Library Examples 256
sqi_fat 300
sram_read_write 240
sst25_fat 301
sst25vf020b 207
Static Implementations 702
Synchronization 696
System Interface 671
System Service Library Examples 265

T

TCP/IP Demonstrations 505
tcpip_client_server 493
tcpip_tcp_client 517
tcpip_tcp_client_server 518
tcpip_tcp_server 523
tcpip_udp_client 524
tcpip_udp_client_server 525
tcpip_udp_server 526
Timer Peripheral Library Examples 260
timer3_interrupt 260
triangle_wave 236

U

uart_basic 262
universal_audio_decoders 45
universal_audio_encoders 54
USART Driver Demonstrations 208
USART Peripheral Library Examples 261
usart_echo 208
usart_loopback 209
usb 488, 498, 504
USB Demonstrations 615
USB Device Stack Component Memory Requirements 617
USB Device Stack Demonstration Application Program and Data Memory Requirements 615
USB HID Host Keyboard and Mouse Tests 618
USB MSD Host USB Pen Drive Tests 617
usb_headset 59
usb_host_headset 66
usb_microphone 72
usb_microphone_multirate 78
usb_smart_speaker 85
usb_speaker 90
usb_speaker_hi_res 97
Using This Document 671

V

vendor 651
Volume I: Getting Started With MPLAB Harmony Libraries and Applications 2
Volume IV: MPLAB Harmony Development 670

W

WDT Peripheral Library examples 263
wdt_timeout 263
web_net_server_nvm_mpfs 527
web_photoframe_demo 536
web_server_nvm_mpfs 539
web_server_sdcard_fatfs 550
Wi-Fi Console Commands 506
wifi_ap_demo 552
wifi_easy_configuration 555
wifi_rgb_easy_configuration 564
wifi_sta_demo 568
wifi_sta_http_demo 577
wifi_sta_ota_demo 580
wifi_sta_wolfssl_demo 585
wifi_staap_demo 571
wifi_wilc1000 587
wifi_winc1500_socket 602
WINC1500 Socket Examples Guide 603
wolfssl_tcp_client 613
wolfssl_tcp_server 612
write_read_ddr2 237

X

X2C-Scope Plug-in 481