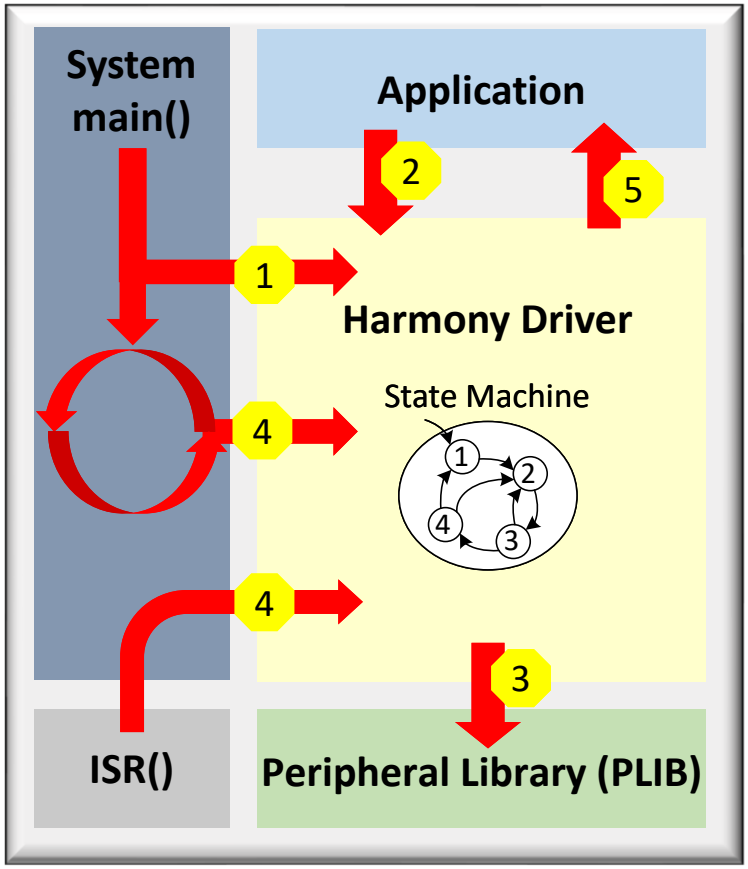
The Differences Between MPLAB Harmony v3 Synchronous and Asynchronous Drivers and When to Use Them

Introduction

MPLAB Harmony v3 drivers provide a simple and abstracted 'C' language interface to the peripherals and other system resources. Some functions are similar across on all the device drivers, while other functions are unique to a specific driver or peripheral. Driver interface functions are generally independent of the details of how a given peripheral is implemented on any specific hardware, or how many instances of that peripheral exist in each system. Applications can control and easily interact with the peripherals by calling the driver interface.

The following diagram represents the MPLAB Harmony v3 Driver Execution flow:

Figure 1. MPLAB Harmony v3 Driver Execution Flow Diagram



1. System Initializes the device driver.
2. Application calls the driver APIs.

3. Driver starts its operation using peripheral libraries which is interacting with hardware.
4. The driver state machine is run from the interrupt context or through continuously polling the device. For example, most of the MPLAB Harmony v3 peripheral drivers are interrupt driven, which means that the driver state machine doesn't wait in a loop for an operation to complete. The driver is notified through an interrupt, when the operation is complete. However, some drivers like the SD Card requires determining whether the SD Card is inserted or not, and most middle-ware has a state machine that runs from a super-loop.
5. Driver finishes the operation and notifies the application.

MPLAB Harmony v3 allows users to configure the drivers in any one of these operating modes: *Synchronous* (Blocking) or *Asynchronous* (Non-Blocking). This document describes *Synchronous* and *Asynchronous* operating modes and how to use them.

1. MPLAB Harmony v3 Driver Features

Asynchronous and Synchronous Driver Modes

- **Asynchronous Mode**
  - Non-blocking Application Program Interfaces (APIs).
  - Allows queuing of multiple requests. Each instance of an *Asynchronous* driver has a dedicated queue.
  - Works seamlessly in bare-metal and RTOS environment.
  - Interrupt and thread-safe.
- **Synchronous Mode**
  - Blocking APIs
  - Support only RTOS environment
  - Interrupt and thread-safe

Note: For additional information on *Asynchronous* and *Synchronous* driver modes, refer to the section Asynchronous and Synchronous Drivers.

Multiple Client Support

This feature seamlessly handle client-specific differences. Both *Synchronous* and *Asynchronous* drivers allow multiple clients to a driver instance. For example, there can be multiple application clients to a SPI driver instance having multiple SPI slaves. The SPI slave specific information, such as clock phase, clock polarity, clock speed, and chip select are all handled by the SPI driver based on the client that submitted the request.

Multiple Instance Support

This feature seamlessly handles the multiple instances of a peripheral. For example, there may be three instances of SPI: SPI0, SPI1, and SPI2. The SPI driver manages three peripheral instances by creating three driver instances and attaching a client to each of them.

Cache Management

Drivers manage cache-related operations on parts (for example, on PIC32MZ, SAME70 and SAMV71) that have a data cache, thereby simplifying the application development.

2. Key Concepts of MPLAB Harmony v3 Drivers

Atomicity
Once started, an operation (sequence of instructions) is considered atomic, if it is indivisible and non-interruptible. A data item is considered atomic if it cannot be subdivided as it is read or written.

Atomicity is necessary for shared data or instructions that need to have exclusive access of the CPU. The sequence of instructions that needs to be protected from shared access is called a critical *code section*, and the data that needs to be protected from shared access is called *critical data*.

Atomicity is achieved by guarding the critical code or critical data. One of the approaches to guard the critical regions is to use a lock, which is set before accessing the shared resource and then released when done.

Interrupt Safety
A sequence of code is said to be “interrupt-safe” when the occurrence of an interrupts does not alter the output or functional behavior of the code. To be “interrupt-safe”, the sequence of code in consideration must be atomic (indivisible and uninterruptible), and the relevant interrupts must be disabled before entering the sequence.

In MPLAB Harmony v3, interrupts can be disabled globally by using the Interrupt System Service API, that is, `SYS_INT_Disable()` can be enabled by using the API, `SYS_INT_Enable()` and can restore the saved state by using the API, `SYS_INT_Restore()`. However, MPLAB Harmony v3 libraries are modular and by convention they respect the abstractions of other libraries, never attempting to directly access their internal resources. Due to 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.

This means it is not necessary to globally disable interrupts in most cases to guarantee correct and reliable operation of a MPLAB Harmony v3 driver. While performing a non-atomic access to data structures or peripheral hardware, a driver temporarily masks the interrupts of the peripheral it owns. This prevents the driver’s own interrupt-driven tasks functions from potentially corrupting data that is also accessed by the driver’s interface functions.

This can be done using the interrupt system service and it is efficient than globally disabling all the interrupts because it allows higher priority interrupts (which do not affect the driver) to occur, protecting their response time latency. This can be done using the Interrupt System Service, as shown in the following example.

```
// Disables the interrupt for SPI0.
Bool spiIntStatus; = false;
spiIntStatus = SYS_INT_SourceDisable(SPI0_IRQn);

// Restore the interrupt for SPI0.
SYS_INT_SourceRestore(spiIntStatus, SPI0_IRQn);
```

These functions can also be used to guard non-atomic accesses by the driver’s interface functions to resources that are shared with the driver’s ISR. This method works to ensure safe access to a shared resource without disabling interrupts globally when using an *Asynchronous* driver with bare-metal configuration.

If a section of code must be truly atomic (uninterruptible), interrupts can be globally disabled for a short period of time using the Interrupt System Service functions, as shown in the following example.

```
Bool interruptState;

// Save global interrupt state and disable interrupt
interruptState = SYS_INT_Disable();
// Critical Section

// Restore interrupt state
SYS_INT_Restore(interruptState);
```

Thread Safety
In an RTOS-based environment, 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 access the same shared resource or execute the same critical section of the

code, that section of the code must be guarded and made atomic. The sequence of code with such guards is known as “thread-safe” code. In the MPLAB Harmony v3 framework, thread safety is achieved by using the methods provided by the MPLAB Harmony Operating System Abstraction Layer (OSAL).

MPLAB Harmony v3 drivers run efficiently in an interrupt driven system. For an interrupt driven system running in an RTOS-based environment, it is common for the driver’s task functions to be called from an interrupt context. A mutex can be used to guard against simultaneous access to shared resources by different threads. However, this will not prevent an ISR from accessing the shared resources or critical code. If a mutex is used to accomplish thread safety, it must be augmented by temporarily disabling (masking) the associated interrupt source as described for a bare-metal environment in the Interrupt Safety section above.

Callback Functions

In the Non-Blocking method, instead of status polling, the callback mechanism can also be used to check the transfer status. For example, the application registers a callback function with a driver and makes the transfer request. The driver calls back the registered function on the completion of the request submitted by the driver client.

MPLAB Harmony v3 provides driver specific APIs to register the callback functions. For example, the *Asynchronous* SPI driver provides API, `DRV_SPI_TransferEventHandlerSet()`, this allows a client to set a transfer event handling function for the driver to call back when a queued transfer has finished.

Notes: Because callbacks are called from the interrupt context, the following guidelines must be followed while implementing a callback function:

- Must be treated like an ISR
- Must be short
- Do not call application functions that are not interrupt safe
- Do not call other driver’s interface functions

Blocking API

A blocking API hangs up execution flow until it has performed its function and returns a result.

For example, the following code shows the I^{2}C driver *Synchronous* mode API, `DRV_I2C_WriteTransfer`.

```
DRV_I2C_WriteTransfer(app_eepromData.i2cHandle, APP_EEPROM_I2C_SLAVE_ADDR, (void
*)app_eepromData.i2cTxBuffer, 2);
```

The execution flow blocks when the API, `DRV_I2C_WriteTransfer()` is called until the requested bytes are transferred to the EEPROM.

Non-Blocking API

A non-blocking API receives the application request and returns immediately without providing the result. The result of the non-blocking API call is provided separately through an *Asynchronous* event. The application verifies the event to take further action.

The following code example shows the usage of the I^{2}C driver *Asynchronous* mode API, `DRV_I2C_WriteReadTransferAdd()`, which is a non-blocking API.

```
void APP_EEPROM_I2CEventHandler ( DRV_I2C_TRANSFER_EVENT event, DRV_I2C_TRANSFER_HANDLE
transferHandle, uintptr_t context)
{
    switch(event)
    {
        case DRV_I2C_TRANSFER_EVENT_COMPLETE:
            /* I2C Transfer Complete. */
            app_eepromData.reqStatus = APP_EEPROM_REQ_STATUS_DONE;
        break;

        case DRV_I2C_TRANSFER_EVENT_ERROR:
            app_eepromData.reqStatus = APP_EEPROM_REQ_STATUS_ERROR;
        break;

        default:
        break;
```

```
    }
}

int main ( void )
{
    /* Initialize all modules */
     ……

    /* Open the I2C Driver */
    app_eepromData.i2cHandle = DRV_I2C_Open( DRV_I2C_INDEX_0, DRV_IO_INTENT_READWRITE );
    if(app_eepromData.i2cHandle == DRV_HANDLE_INVALID)
    {
        app_eepromData.state = APP_STATE_ERROR;
    }
    else
    {
        /* Register I2C transfer complete Event Handler for EEPROM. */
        DRV_I2C_TransferEventHandlerSet(app_eepromData.i2cHandle, APP_EEPROM_I2CEventHandler,
0);

        /* Submit I2C transfer to read stored temperature values from EEPROM. */
        DRV_I2C_WriteReadTransferAdd(app_eepromData.i2cHandle,
                APP_EEPROM_I2C_SLAVE_ADDR, app_eepromData.i2cTxBuffer, 1,
                app_eepromData.i2cRxBuffer, 5, &app_eepromData.transferHandle);

        /* Display the read temperature on console */
    }

     ……
}
```

In the main function, the application registers a callback function (event handler "`APP_EEPROM_I2CEventHandler`") with the I²C driver. The control immediately returns after the submission of the I²C request, calling the API `DRV_I2C_WriteReadTransferAdd()`. The application is notified by the driver on the completion of the request through an event to the callback function (event handler).

Synchronization

MPLAB Harmony v3 provides driver interface functions having both a blocked and non-blocking usage model. There are situations when developers expect a blocking model. For example, the file system style read and write functions block and not return until the entire transfer has completed.

The requirement to block is challenging to accomplish in a bare-metal environment. The RTOS-based design provides flexibility to implement the blocking requirement. MPLAB Harmony v3 provides the synchronous drivers to implement the blocking requirement of application. The Synchronous drivers are blocking in nature. Although the APIs are blocking, the data transfer still takes place from the interrupt context.

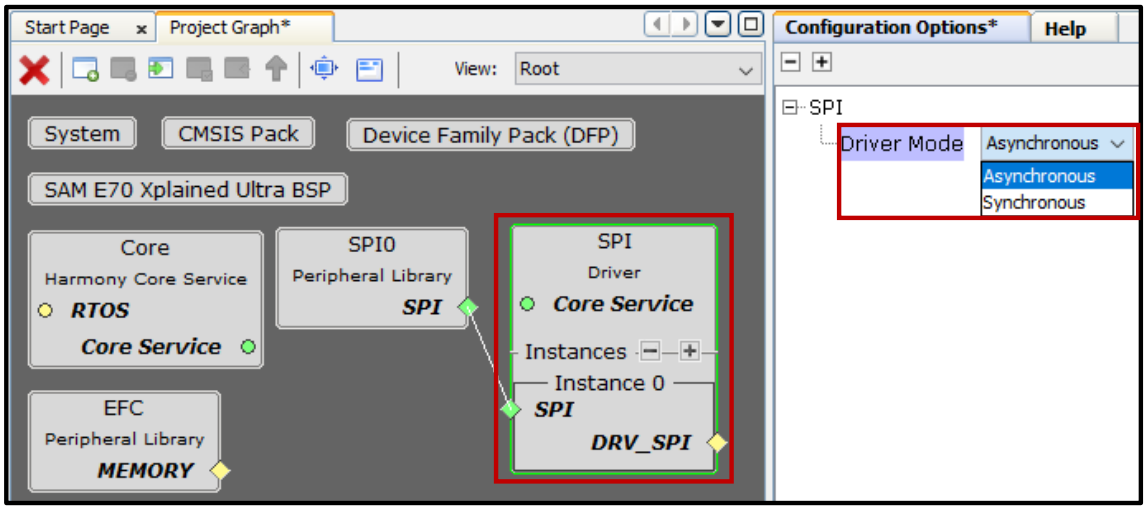
Note: MPLAB Harmony v3 OSAL provides a consistent interface to MPLAB Harmony v3 framework components (drivers, middleware, and so on). It takes care of the underlying differences between the available or supported RTOS Kernels, and ensures correct operation in the bare-metal and RTOS environment.

3. Asynchronous and Synchronous Drivers

Configuration

MPLAB Harmony v3 allows the user to configure the drivers in *Synchronous* or *Asynchronous* mode using the MPLAB Harmony v3 Configurator (MHC) Project Graph driver configuration. For example, the following figure shows how to configure the SPI Driver Operating mode.

Figure 3-1. MPLAB Harmony v3 SPI Driver Mode Configuration



Note: To configure a driver in *Synchronous* or *Asynchronous* mode, in the MHC project graph, the user should click on the driver block. For example, in the figure above, clicking on the highlighted SPI block shows the driver configuration modes.

Differences

The following table provides the SPI *Synchronous* and *Asynchronous* driver APIs to perform the Read and Write operations.

Table 3-1. MPLAB Harmony v3 SPI Asynchronous and Synchronous Driver APIs and SPI PLIB APIs

SPI Asynchronous Driver APIs	SPI Synchronous Driver APIs	SPI PLIB APIs
DRV_SPI_WriteTransferAdd	DRV_SPI_WriteTransfer	SPI0_Write (…)
		SPI1_Write (…)
DRV_SPI_ReadTransferAdd	DRV_SPI_ReadTransfer	SPI0_Read (…)
		SPI1_Read (…)

Note: It does not matter which driver mode is selected (*Synchronous* or *Asynchronous*), the SPI driver uses the same underlying PLIB APIs.

The following table provides the differences in the SPI driver APIs that are generated based on the driver mode selection.

Table 3-2. Differences in SPI Asynchronous and Synchronous Driver APIs

SPI Asynchronous Driver APIs	SPI Synchronous Driver APIs
DRV_SPI_WriteTransferAdd	DRV_SPI_WriteTransfer
DRV_SPI_ReadTransferAdd	DRV_SPI_ReadTransfer

.....continued	
SPI Asynchronous Driver APIs	**SPI Synchronous Driver APIs**
DRV_SPI_WriteReadTransferAdd	DRV_SPI_WriteReadTransfer
DRV_SPI_TransferEventHandlerSet	N/A
DRV_SPI_TransferStatusGet	N/A

The following table provides the common SPI driver APIs that are generated regardless of the driver mode selected.

Table 3-3. Common SPI Asynchronous and Synchronous Driver APIs

SPI Driver APIs
DRV_SPI_Initialize
DRV_SPI_Status
DRV_SPI_Open
DRV_SPI_Close
DRV_SPI_TransferSetup

The following table shows the summary of differences between Synchronous and Asynchronous drivers.

Table 3-4. Summary of the Differences Between Synchronous and Asynchronous Drivers

Asynchronous Drivers	Synchronous Drivers
Non-Blocking APIs	Blocking APIs
Works seamlessly in bare-metal and supports in an RTOS environment	Suitable to use in RTOS environment
Each instance of a driver has a dedicated queue. The driver allows queuing of multiple requests.	A request to driver blocks the application until the submitted request is serviced. Therefore a queue is not needed.
A transfer request is identified through a transfer handle. The client can get a transfer request status using a polling or callback mechanism by passing the transfer handle to the APIs. For example: *Polling Mechanism*: The API, DRV_SPI_TransferStatusGet(), can be used to poll the status of the queued transfer request. *Callback Mechanism*: The API, DRV_SPI_TransferEventHandlerSet(), is called to register the callback function with the driver to notify with an event when the transfer request is completed.	Because the API is blocking, and the return value of the API indicates the transfer status, there is no need for the transfer handle.
Interrupt and Thread Safe	Interrupt and Thread Safe

Note: The MPLAB Harmony v3 Configurator (MHC) takes care of generating the PLIB APIs regardless of the driver mode selected.
Both Synchronous (only in RTOS environment) and Asynchronous drivers are supported for UART, I^2C, SPI, SDSPI, and memory device (MX25L, AT25DF, NVM, and SST26) peripherals.

Only the Asynchronous driver is supported for SDMMC, external EEPROM (AT24 and AT25), SPI Flash (AT25DF), SQI Flash (MX25L and SST26).

4. Usage Recommendation of Asynchronous and Synchronous Drivers

Synchronous drivers are suitable for use in an RTOS environment, whereas *Asynchronous* drivers are suitable in a Bare-metal environment.

Asynchronous drivers (Non-Blocking API) are suitable for an application which runs in a Bare-metal or non-RTOS based environment. In a Bare-metal environment, the suggested way to implement an application is by implementing the state machine programming model. In a state machine programming model, an application avoids busy waiting loops so as to not waste CPU bandwidth. Therefore, the module's functions must not block waiting for an external operation to complete (especially on anything that has any possibility of never completing) or it may block the entire system. If the function must wait for an external operation, the module must break up the operation into smaller tasks to be performed later. The multiple smaller tasks run in the super loop. The task running in the super loop includes application tasks and driver tasks. The application task checks the completion of the driver operations through status flags.

MPLAB Harmony v3 drivers running in *asynchronous* mode provide the ability to acquire the driver task completion status through the implementation of *asynchronous* callback events (as discussed in Callback Functions) that the application registers with the driver. In addition to the callback mechanism the MPLAB Harmony v3 drivers running in *asynchronous* mode provide status functions for the application task to check the driver task completion status. For example, the SPI driver provides the status check API,`DRV_SPI_TransferStatusGet()`.

Synchronous driver (Blocking API) is suitable in an RTOS-based application environment. In an RTOS-based application environment, the blocking interface is acceptable and desirable. When the *synchronous* driver API in a thread blocks waiting for the completion of a task (for example, the I^2C driver API, `DRV_I2C_WriteTransfer()`, blocks for the transfer of bytes), the RTOS scheduler ensures that the thread in contention does not block the control flow indefinitely. The RTOS, particularly with preemptive scheduling, ensures that the next high-priority thread runs immediately when the current thread has blocked. Refer to Interrupt Safety, Thread Safety, Synchronization and Blocking API for additional information on blocking behavior and the implementation using MPLAB Harmony v3 Synchronous drivers.

5. References

1. MPLAB Harmony GitHub: github.com/Microchip-MPLAB-Harmony

2. SAM E70 Xplained Ultra User's Guide:ww1.microchip.com/downloads/en/DeviceDoc/SAM-E70-Xplained-Ultra-User-Guide-70005389A.pdf

3. SAM E70/S70/V70/V71 Family Data Sheet: ww1.microchip.com/downloads/en/DeviceDoc/SAM-E70-S70-V70-V71-Family-Data-Sheet-DS60001527D.pdf

4. Getting Started with Harmony v3 Drivers on SAM E70/S70/V70/V71 MCUs Using FreeRTOS: microchipdeveloper.com/harmony3:same70-getting-started-tm-drivers-freertos

5. Getting Started with Harmony v3 Drivers and Middleware on PIC32MZ EF MCUs using FreeRTOS: microchipdeveloper.com/harmony3:pic32mz-get-start-tm-drvr-middlware-freertos

6. MPLAB Harmony v3 Application Development Guide for MPLAB Harmony v2 Users: ww1.microchip.com/downloads/en/Appnotes/MPLAB_Harmonyv3_Application_Development_%20Guide_for_%20MPLAB_Harmonyv2_Users_DS00003388A.pdf

7. MPLAB Harmony Core Help: microchip-mplab-harmony.github.io/core/frames.html?frmname=topic&frmfile=index.html

8. How to Setup MPLAB Harmony v3 Software Development Framework: ww1.microchip.com/downloads/en/DeviceDoc/How_to_Setup_MPLAB_%20Harmony_v3_Software_Development_Framework_DS90003232C.pdf

9. Creating a "Hello World" Application on SAM Microcontrollers Using Harmony 3 MPLAB Harmony Configurator (MHC): ww1.microchip.com/downloads/en/DeviceDoc/Creating_Hello_World_%20Application_on_SAM_Using_MHC_DS90003231A.pdf

10. Creating a Hello World Application on PIC32 Microcontrollers Using the MPLAB Harmony v3's MPLAB Harmony Configurator (MHC): ww1.microchip.com/downloads/en/DeviceDoc/Creating_Hello_World_Applicatio_%20on_PIC32_Microcontrollers_Using_MPLAB_Harmonyv3_MHC_DS90003259A.pdf

The Microchip Website

Microchip provides online support via our website at www.microchip.com/. This website is used to make files and information easily available to customers. Some of the content available includes:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQs), technical support requests, online discussion groups, Microchip design partner program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Product Change Notification Service

Microchip's product change notification service helps keep customers current on Microchip products. Subscribers will receive email notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, go to www.microchip.com/pcn and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Embedded Solutions Engineer (ESE)
- Technical Support

Customers should contact their distributor, representative or ESE for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in this document.

Technical support is available through the website at: www.microchip.com/support

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Legal Notice

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with

your specifications. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Microchip disclaims all liability arising from this information and its use. Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

Trademarks

The Microchip name and logo, the Microchip logo, Adaptec, AnyRate, AVR, AVR logo, AVR Freaks, BesTime, BitCloud, chipKIT, chipKIT logo, CryptoMemory, CryptoRF, dsPIC, FlashFlex, flexPWR, HELDO, IGLOO, JukeBlox, KeeLoq, Kleer, LANCheck, LinkMD, maXStylus, maXTouch, MediaLB, megaAVR, Microsemi, Microsemi logo, MOST, MOST logo, MPLAB, OptoLyzer, PackeTime, PIC, picoPower, PICSTART, PIC32 logo, PolarFire, Prochip Designer, QTouch, SAM-BA, SenGenuity, SpyNIC, SST, SST Logo, SuperFlash, Symmetricom, SyncServer, Tachyon, TempTrackr, TimeSource, tinyAVR, UNI/O, Vectron, and XMEGA are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

APT, ClockWorks, The Embedded Control Solutions Company, EtherSynch, FlashTec, Hyper Speed Control, HyperLight Load, IntelliMOS, Libero, motorBench, mTouch, Powermite 3, Precision Edge, ProASIC, ProASIC Plus, ProASIC Plus logo, Quiet-Wire, SmartFusion, SyncWorld, Temux, TimeCesium, TimeHub, TimePictra, TimeProvider, Vite, WinPath, and ZL are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Adjacent Key Suppression, AKS, Analog-for-the-Digital Age, Any Capacitor, AnyIn, AnyOut, BlueSky, BodyCom, CodeGuard, CryptoAuthentication, CryptoAutomotive, CryptoCompanion, CryptoController, dsPICDEM, dsPICDEM.net, Dynamic Average Matching, DAM, ECAN, EtherGREEN, In-Circuit Serial Programming, ICSP, INICnet, Inter-Chip Connectivity, JitterBlocker, KleerNet, KleerNet logo, memBrain, Mindi, MiWi, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, MultiTRAK, NetDetach, Omniscient Code Generation, PICDEM, PICDEM.net, PICkit, PICtail, PowerSmart, PureSilicon, QMatrix, REAL ICE, Ripple Blocker, SAM-ICE, Serial Quad I/O, SMART-I.S., SQI, SuperSwitcher, SuperSwitcher II, Total Endurance, TSHARC, USBCheck, VariSense, ViewSpan, WiperLock, Wireless DNA, and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

The Adaptec logo, Frequency on Demand, Silicon Storage Technology, and Symmcom are registered trademarks of Microchip Technology Inc. in other countries.

GestIC is a registered trademark of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries.

All other trademarks mentioned herein are property of their respective companies.

ISBN: 978-1-5224-6375-7

Quality Management System

For information regarding Microchip's Quality Management Systems, please visit www.microchip.com/quality.

Worldwide Sales and Service

AMERICAS	ASIA/PACIFIC	ASIA/PACIFIC	EUROPE
Corporate Office 2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel: 480-792-7200 Fax: 480-792-7277 Technical Support: www.microchip.com/support Web Address: www.microchip.com **Atlanta** Duluth, GA Tel: 678-957-9614 Fax: 678-957-1455 **Austin, TX** Tel: 512-257-3370 **Boston** Westborough, MA Tel: 774-760-0087 Fax: 774-760-0088 **Chicago** Itasca, IL Tel: 630-285-0071 Fax: 630-285-0075 **Dallas** Addison, TX Tel: 972-818-7423 Fax: 972-818-2924 **Detroit** Novi, MI Tel: 248-848-4000 **Houston, TX** Tel: 281-894-5983 **Indianapolis** Noblesville, IN Tel: 317-773-8323 Fax: 317-773-5453 Tel: 317-536-2380 **Los Angeles** Mission Viejo, CA Tel: 949-462-9523 Fax: 949-462-9608 Tel: 951-273-7800 **Raleigh, NC** Tel: 919-844-7510 **New York, NY** Tel: 631-435-6000 **San Jose, CA** Tel: 408-735-9110 Tel: 408-436-4270 **Canada - Toronto** Tel: 905-695-1980 Fax: 905-695-2078	**Australia - Sydney** Tel: 61-2-9868-6733 **China - Beijing** Tel: 86-10-8569-7000 **China - Chengdu** Tel: 86-28-8665-5511 **China - Chongqing** Tel: 86-23-8980-9588 **China - Dongguan** Tel: 86-769-8702-9880 **China - Guangzhou** Tel: 86-20-8755-8029 **China - Hangzhou** Tel: 86-571-8792-8115 **China - Hong Kong SAR** Tel: 852-2943-5100 **China - Nanjing** Tel: 86-25-8473-2460 **China - Qingdao** Tel: 86-532-8502-7355 **China - Shanghai** Tel: 86-21-3326-8000 **China - Shenyang** Tel: 86-24-2334-2829 **China - Shenzhen** Tel: 86-755-8864-2200 **China - Suzhou** Tel: 86-186-6233-1526 **China - Wuhan** Tel: 86-27-5980-5300 **China - Xian** Tel: 86-29-8833-7252 **China - Xiamen** Tel: 86-592-2388138 **China - Zhuhai** Tel: 86-756-3210040	**India - Bangalore** Tel: 91-80-3090-4444 **India - New Delhi** Tel: 91-11-4160-8631 **India - Pune** Tel: 91-20-4121-0141 **Japan - Osaka** Tel: 81-6-6152-7160 **Japan - Tokyo** Tel: 81-3-6880- 3770 **Korea - Daegu** Tel: 82-53-744-4301 **Korea - Seoul** Tel: 82-2-554-7200 **Malaysia - Kuala Lumpur** Tel: 60-3-7651-7906 **Malaysia - Penang** Tel: 60-4-227-8870 **Philippines - Manila** Tel: 63-2-634-9065 **Singapore** Tel: 65-6334-8870 **Taiwan - Hsin Chu** Tel: 886-3-577-8366 **Taiwan - Kaohsiung** Tel: 886-7-213-7830 **Taiwan - Taipei** Tel: 886-2-2508-8600 **Thailand - Bangkok** Tel: 66-2-694-1351 **Vietnam - Ho Chi Minh** Tel: 84-28-5448-2100	**Austria - Wels** Tel: 43-7242-2244-39 Fax: 43-7242-2244-393 **Denmark - Copenhagen** Tel: 45-4485-5910 Fax: 45-4485-2829 **Finland - Espoo** Tel: 358-9-4520-820 **France - Paris** Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79 **Germany - Garching** Tel: 49-8931-9700 **Germany - Haan** Tel: 49-2129-3766400 **Germany - Heilbronn** Tel: 49-7131-72400 **Germany - Karlsruhe** Tel: 49-721-625370 **Germany - Munich** Tel: 49-89-627-144-0 Fax: 49-89-627-144-44 **Germany - Rosenheim** Tel: 49-8031-354-560 **Israel - Ra'anana** Tel: 972-9-744-7705 **Italy - Milan** Tel: 39-0331-742611 Fax: 39-0331-466781 **Italy - Padova** Tel: 39-049-7625286 **Netherlands - Drunen** Tel: 31-416-690399 Fax: 31-416-690340 **Norway - Trondheim** Tel: 47-72884388 **Poland - Warsaw** Tel: 48-22-3325737 **Romania - Bucharest** Tel: 40-21-407-87-50 **Spain - Madrid** Tel: 34-91-708-08-90 Fax: 34-91-708-08-91 **Sweden - Gothenberg** Tel: 46-31-704-60-40 **Sweden - Stockholm** Tel: 46-8-5090-4654 **UK - Wokingham** Tel: 44-118-921-5800 Fax: 44-118-921-5820