ADC Sequencing Using DMA and EVENT System on PIC32C and SAM MCUs Using MPLAB Harmony v3 and MCC

TB3367

Introduction

A high CPU load can be caused by using interrupts to manage a large number of ADC conversions across multiple channels and transferring the resulting data from peripheral registers to a user buffer. With the help of an Event System and a DMA, the data can be transferred to the specified memory location without the intervention of the CPU.

This document describes how to configure and use the ADC peripheral library in MPLAB® Harmony v3 with the timer, Event System (EVSYS), and DMA data transfer features.

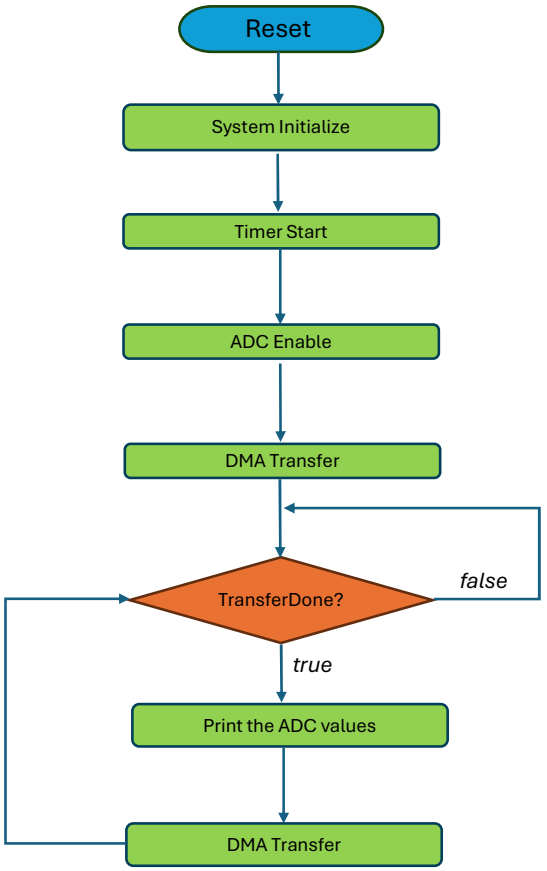
Notes: This configuration can be implemented with the following family of devices:

- PIC32CM family of microcontrollers
- PIC32CX family of microcontrollers
- PIC32CK family of microcontrollers
- PIC32CZ family of microcontrollers
- SAM C family of microcontrollers
- SAM D family of microcontrollers
- SAM L family of microcontrollers
- SAM D5x/E5x family of microcontrollers

1. ADC Operation in Automatic Sequencing Mode with DMA Transfer Mode

Automatic sequencing of an Analog-to-Digital Converter (ADC) using Direct Memory Access (DMA) is used to efficiently transfer data from the ADC to memory without involving the CPU for each data transfer.

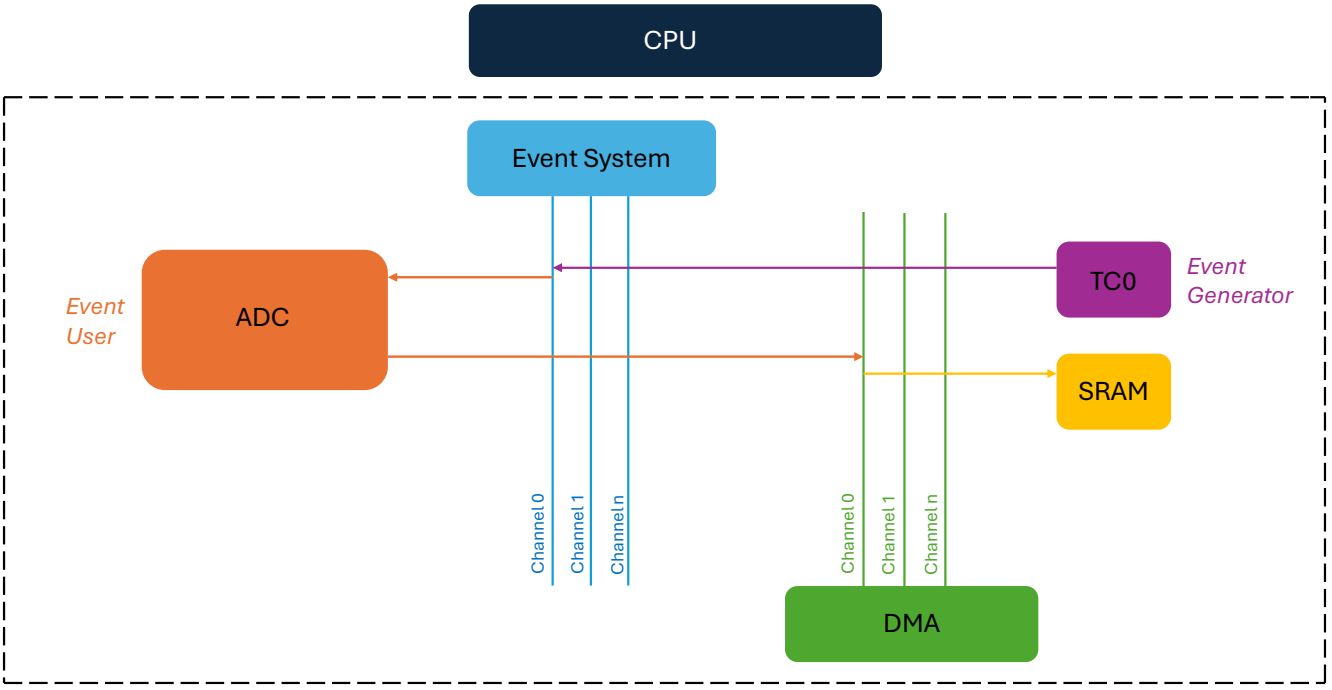
Figure 1-1. Flow Chart



The process initiates with the configuration of the timer, specifying a sampling period in either milliseconds or seconds. The timer then starts counting from zero to a predefined top value (100 millisecond). When the timer reaches the top value, it triggers the timer (Event Generator), which in turn triggers the ADC (Event User) through the Event System (EVSYS).

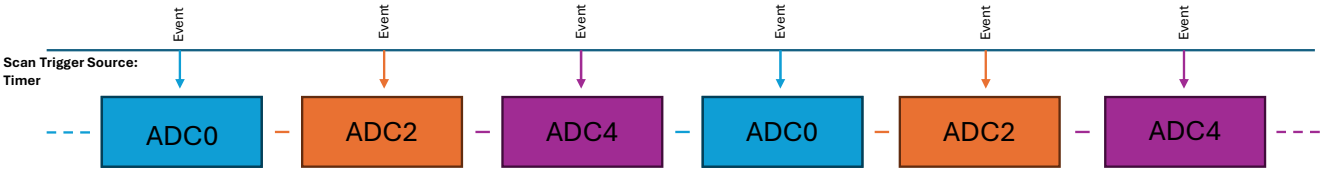
The process continues with the ADC configuration, where multiple channels are selected to be sampled in a specific sequence, often defined by setting up a sequence register. The ADC then starts conversions based on a specific trigger source, which, in this case, refers to the timer. Next, the DMA transfer is initiated.

Figure 1-2. Block Diagram



After the DMA transfer is initiated, the converted ADC data from the ADC result register is transferred to the specific location (SRAM). The DMA controller uses the ADC result register as the source, and a specific memory location (SRAM) or buffer as the destination. The number of data transfers (samples) to be moved from the ADC to memory is specified.

Figure 1-3. ADC Sequencing



Once the ADC is triggered to start the conversion process, the ADC samples the first channel, converts the analog signal to a digital value and stores the result in its result register. The DMA controller fetches the new data available in the ADC result register and transfers it to the specified memory location (SRAM). The ADC proceeds to the next channel in the sequence and repeats the conversion process. At the same time, the DMA continues to transfer each data to the memory location until the maximum number of transfers is reached.

2. Creating the Application on PIC32CM MCU Using MPLAB Harmony v3 and MCC

The following software and hardware tools are used for this application:

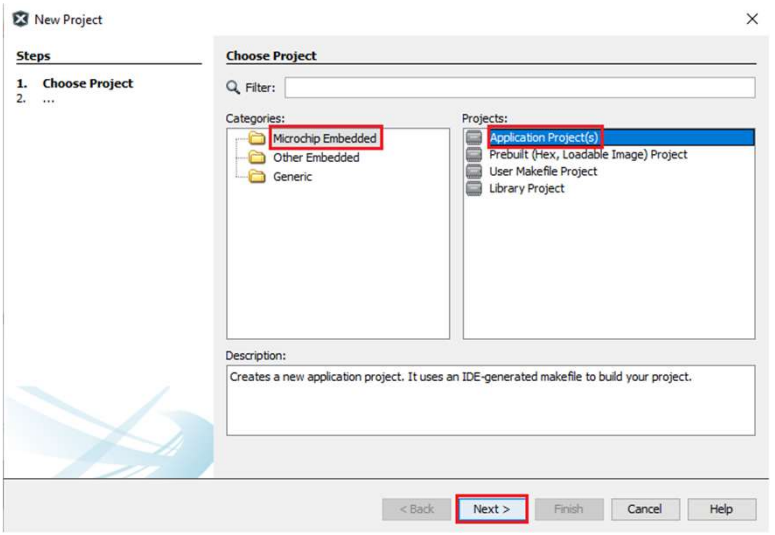
- MPLAB X IDE v6.20
- MPLAB Code Configurator Plugin v5.5.1
- MPLAB XC32 Compiler v4.45
- csp v3.20.0
- PIC32CM JH01 Curiosity Nano+ Touch Evaluation Kit

Note: Updated versions of the above-listed tools can also be used to create the application, and users are not restricted to using the older versions.

To create an MPLAB Harmony v3-based project, follow these steps:

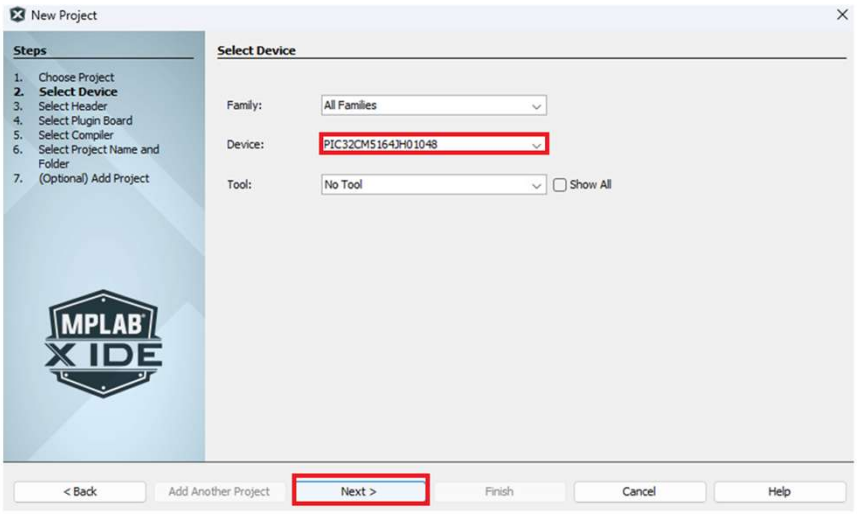
1. From the Start menu launch MPLAB X IDE.
2. Click *File > New Project* or click on the *New Project* icon.
3. In the **Choose Project** properties page:
   a. Categories: Select **Microchip Embedded**.
   b. Projects: Select **Application Project(s)**.

Figure 2-1. Choose Project



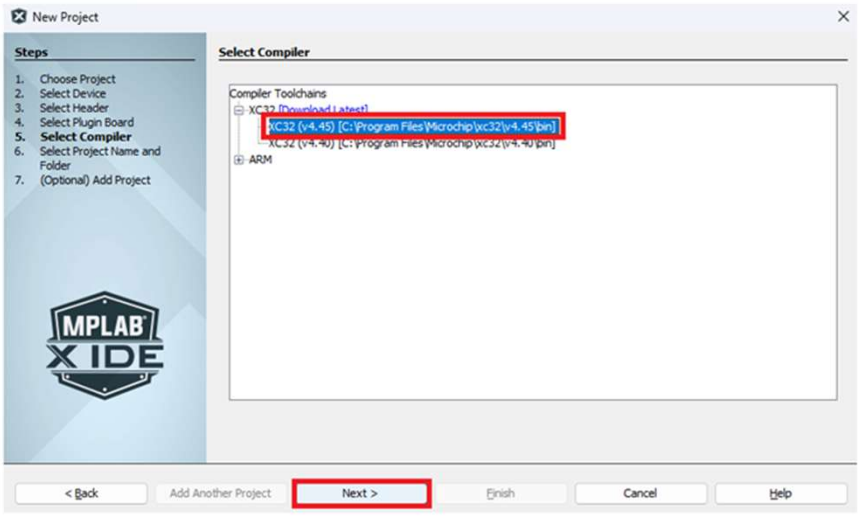
4. Click **Next.**
5. Click **Select Device** and in the right **Select Device** properties page:
   a. For Device, select **PIC32CM5164JH01048** for creating the project on the PIC32CM JH01 Curiosity Nano+ Touch Evaluation Kit.
   Note: The selected device will reflect under *Target Device.*

Figure 2-2. Device Selection



6. Click **Next**.
7. Click **Select Compiler**, and in the right **Select Compiler** properties page:
   a. Under Compiler Toolchains, click and expand XC32, and then select **XC32 (v4.45)** as shown below.

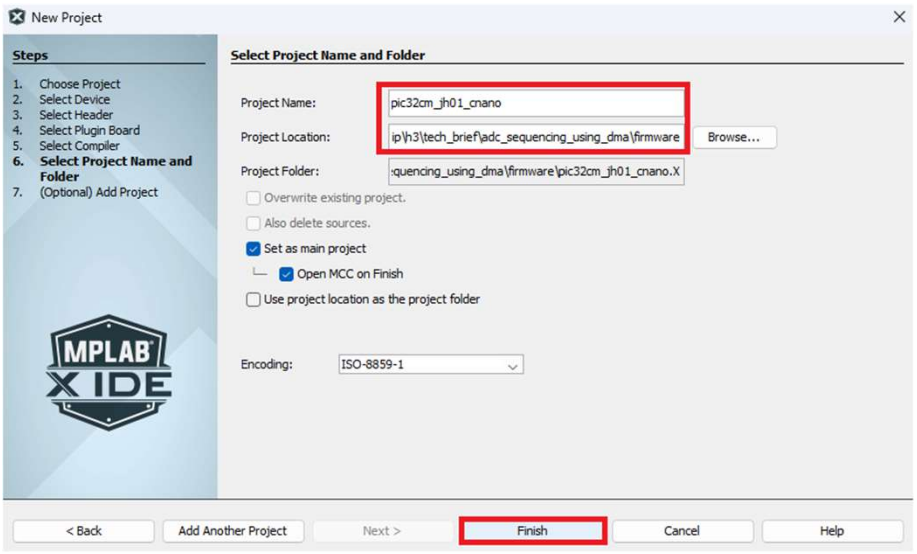
Figure 2-3. XC32 Compiler Selection



8. Click **Next.**
9. Click **Select Project Name and Folder**, and in the right **Select Project Name and Folder** properties page, enter these details:
   a. **Project Name**: Enter `pic32cm_jh01_cnano`. This indicates the name of the project that will be shown in MPLAB X IDE to set the project's name.
   b. **Location Project**: Enter *C:\microchip\h3\tech_brief\adc_seqeuencing_using_dma\firmware*. This indicates the path to the root folder of the new project. All project files will be placed in this folder. The project location can be any valid path.

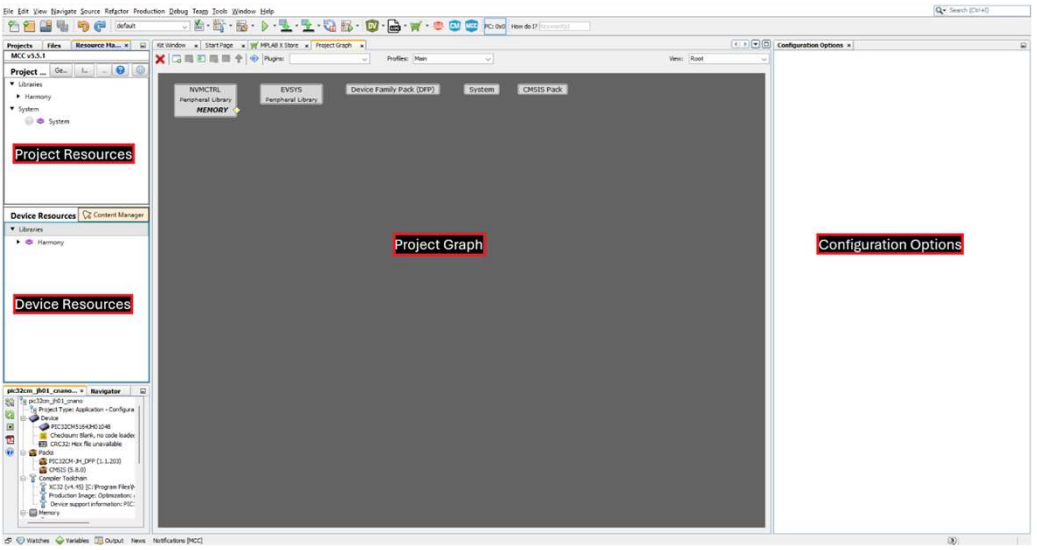
c. **Project Folder**: Read-only content, which automatically updates when users make changes to the above entries.

Figure 2-4. Project Name and Folder Settings



10. Click **Finish** to launch MCC.
11. The MCC plugin will launch in a new window as shown below:

Figure 2-5. MPLAB Code Configurator Window

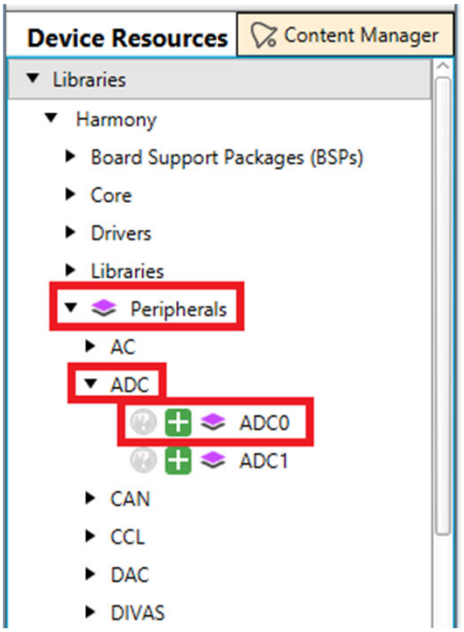


3. Adding and Configuring MPLAB Harmony Components

To add and configure the MPLAB Harmony components using the MCC, follow these steps:

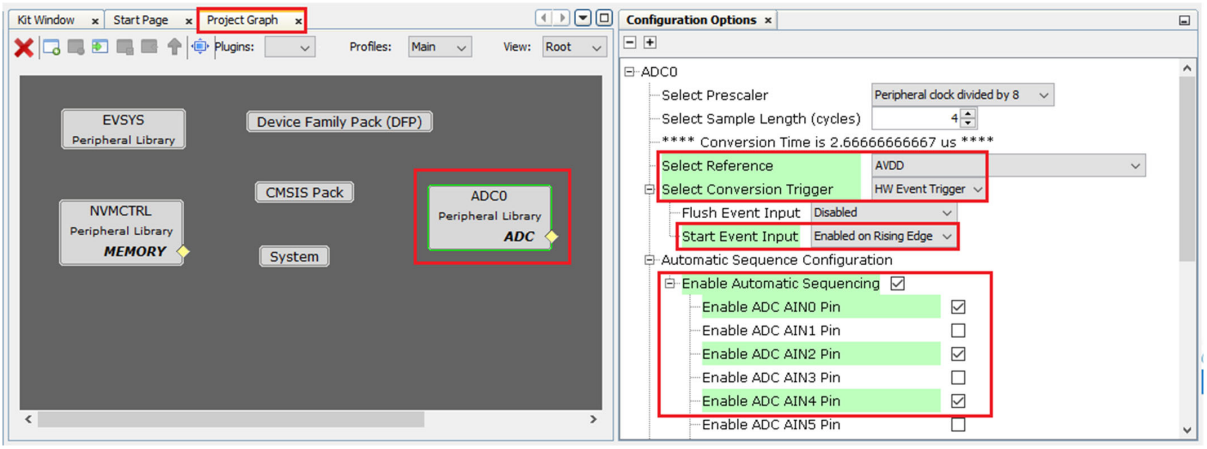
1. In the MCC window, under **Device Resources**, click and expand *Harmony > Peripherals > ADC*, and then select **ADC0.**

Figure 3-1. Selecting ADC Module



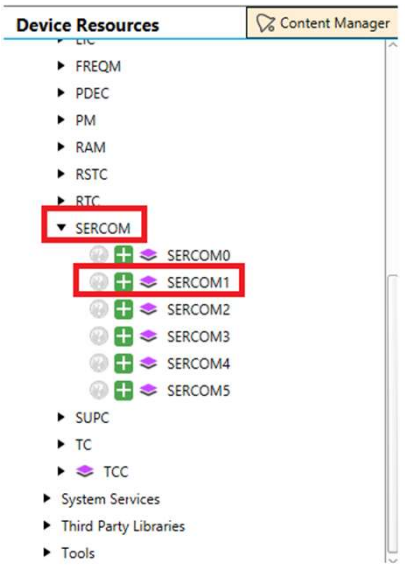
2. To configure ADC0 module, click **Project Graph**, and then select **ADC0** from the left navigation pane.
3. In the **Configuration Options** property page, click and expand ADC0 and then select required options as shown below:

Figure 3-2. ADC0 Configuration



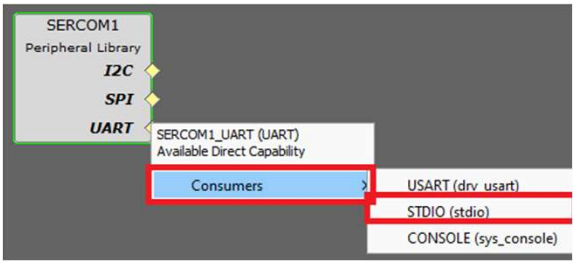
4. Under **Device Resources**, click and expand *Harmony > Peripherals > SERCOM,* and then select **SERCOM1**.

Figure 3-3. SERCOM1 Module



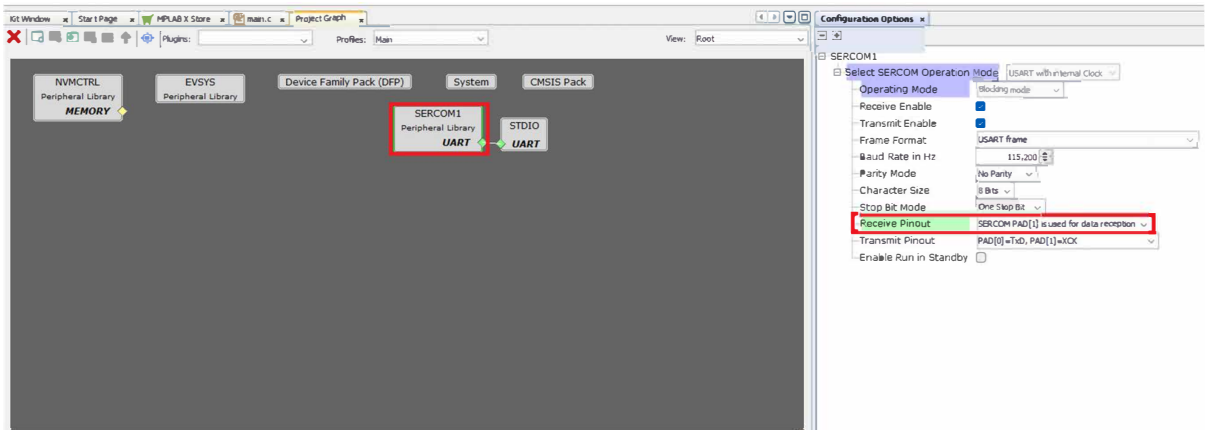
5. Right-click on the **SERCOM1 UART**, and then select *Consumers > STDIO (stdio).*

Figure 3-4. Adding STDIO Module in SERCOM1



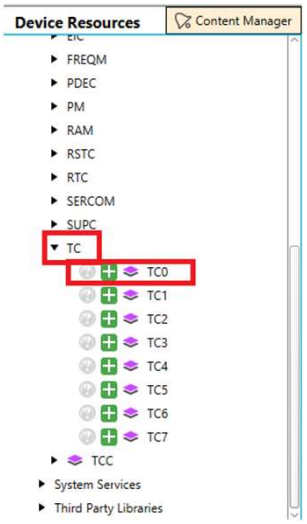
6. To configure the SERCOM1 module, click **Project Graph** and then select **SERCOM1 Peripheral Library UART**.
7. In the **Configuration Options** property page, click and expand **SERCOM1** and then configure as shown below:

Figure 3-5. SERCOM1 Configuration



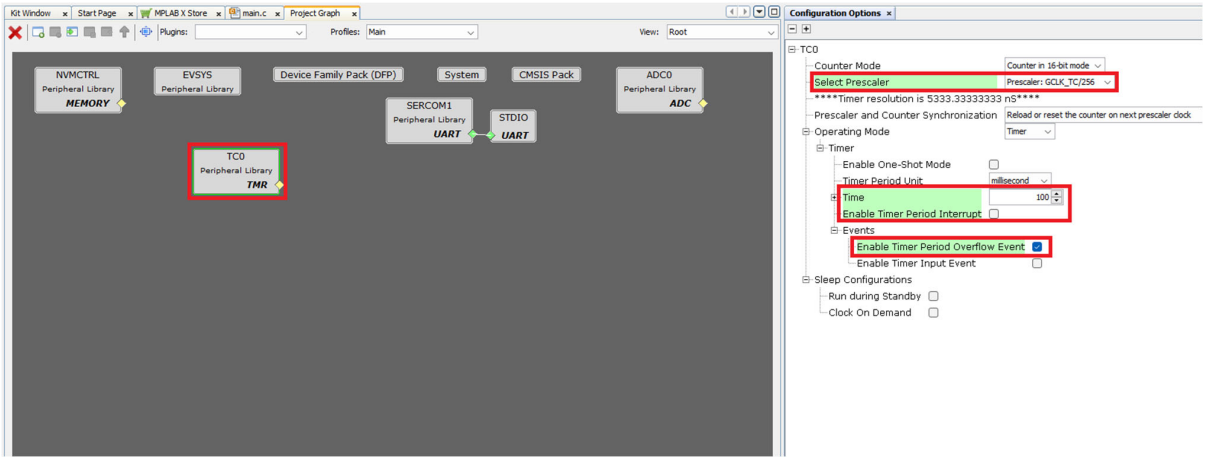
8. Under **Device Resources**, click and expand *Harmony* > *Peripherals* > *TC*, and then select **TC0**.

Figure 3-6. TC0 Module



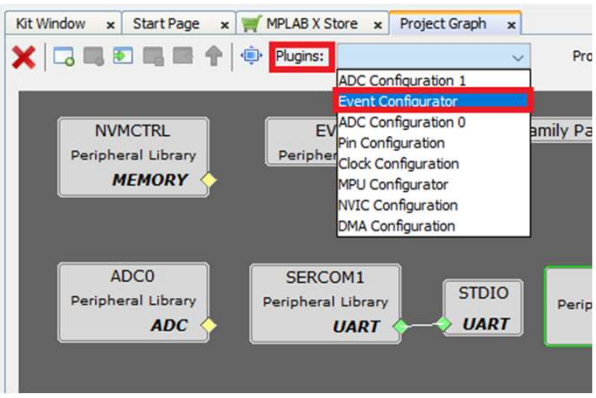
9. To configure **TC0**, select **TC0 Peripheral Library TMR**, and in the right **Configuration Options** property page select options as shown below:

Figure 3-7. Configuration of TC0 Module



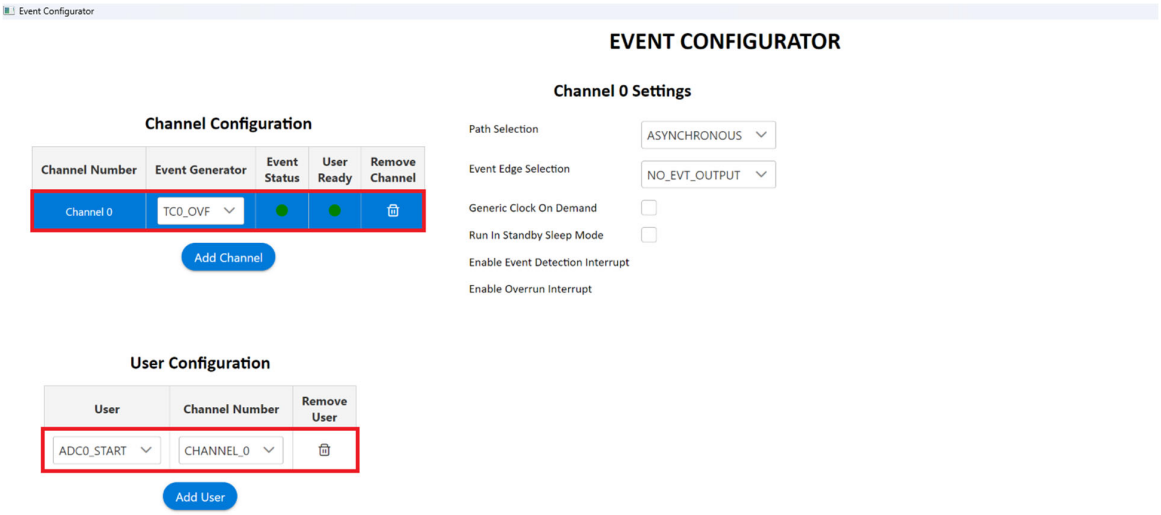
10. From the **Plugins** drop-down list, select **Event Configurator**.

Figure 3-8. Event Configurator



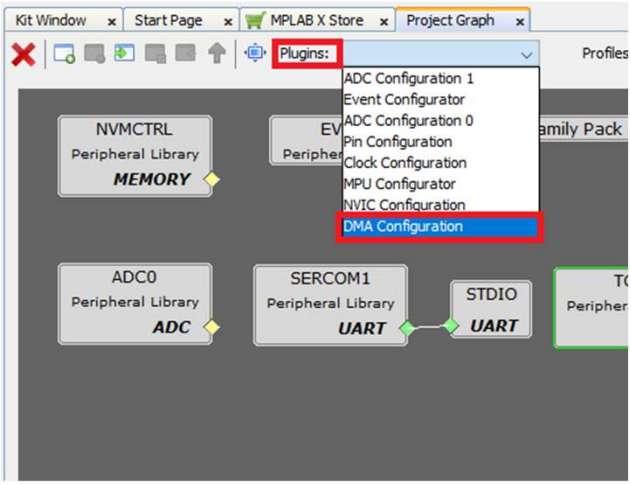
11. To configure Event Configurator, follow these instructions:
    a. Select **Add Channel** under Channel Configuration, and then set it up as illustrated.
    b. Select **Add User** under User Configuration, and then configure it as shown in the following figure.

Figure 3-9. Configuration of Event Configurator



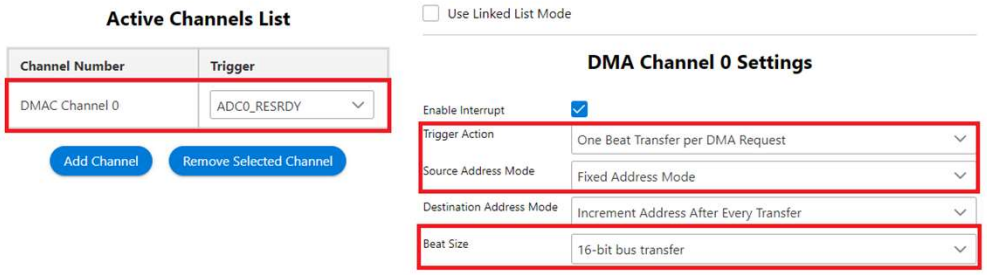
12. From the **Plugins** drop-down list, select **DMA Configuration**.

Figure 3-10. DMA Configuration



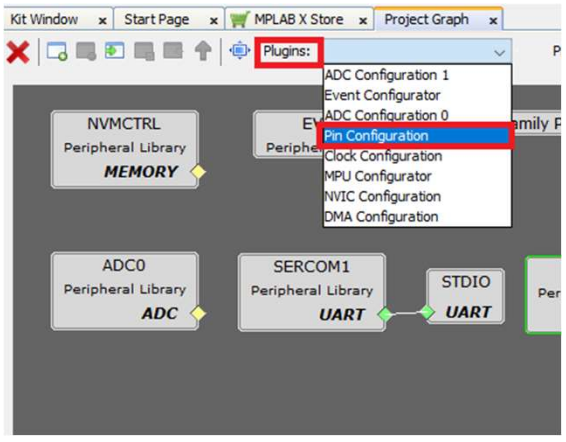
13. To configure the DMA, follow these steps:
    a. Select **Add Channel** under the Active Channels List, and then select the trigger as **ADC0_RESRDY**.
    b. Configure **DMA Channel 0 Settings** as shown in the following figure.

Figure 3-11. Configuration of DMA



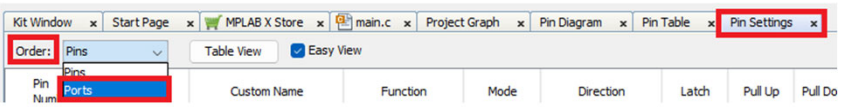
14. From the **Plugins** drop-down list, select **Pin Configuration**, and then click **Pin Settings.**

Figure 3-12. Pin Configuration



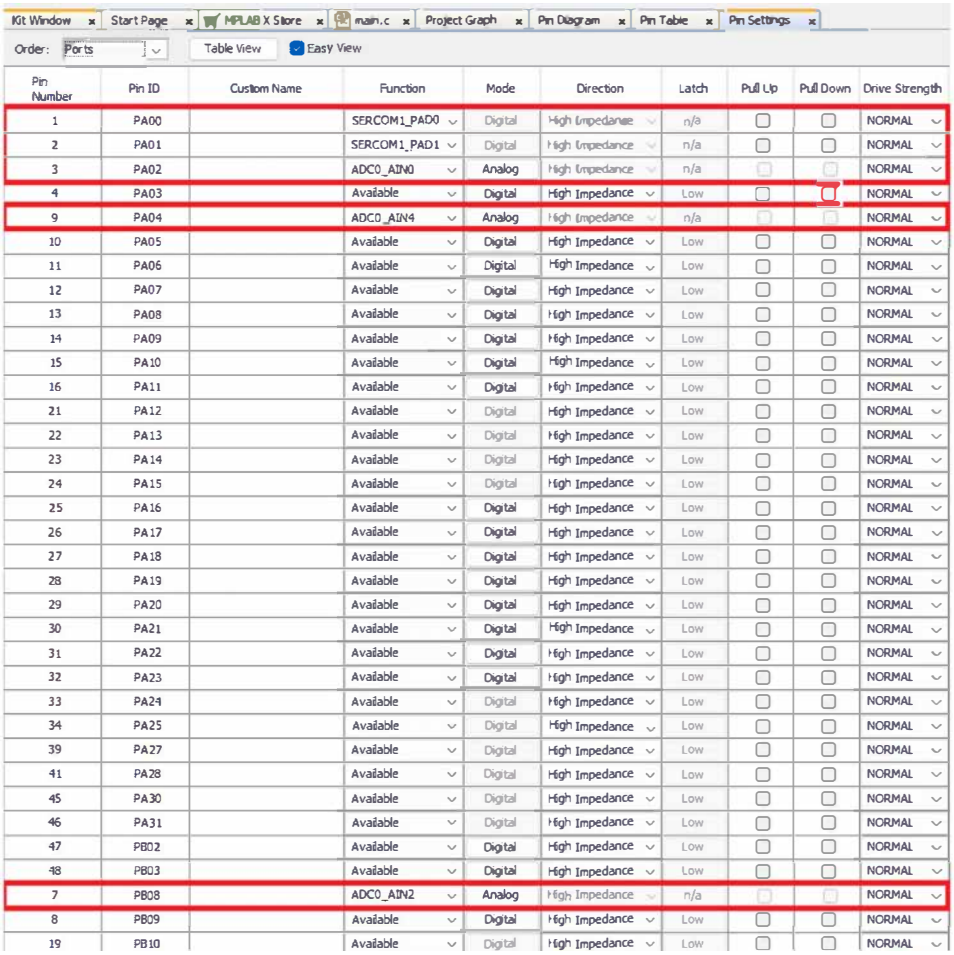
15. From the **Order** menu, select **Ports.**

Figure 3-13. Selecting Ports from Order Menu



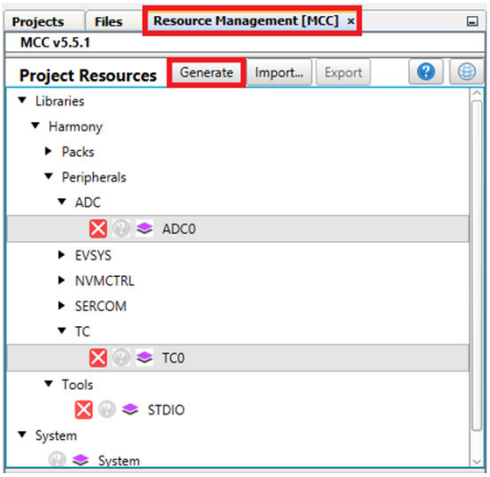
16. Configure pins as shown in the following figure.

Figure 3-14. Configuration of Pins



17. Click **Resource Management [MCC]**, and then click **Generate** to generate the code.

Figure 3-15. Generate the Project



4. Adding Application Logic to the Project

To develop and run the application, follow these steps:

1. Add the required macro and variables outside the `main()` function in the `main.c` file.

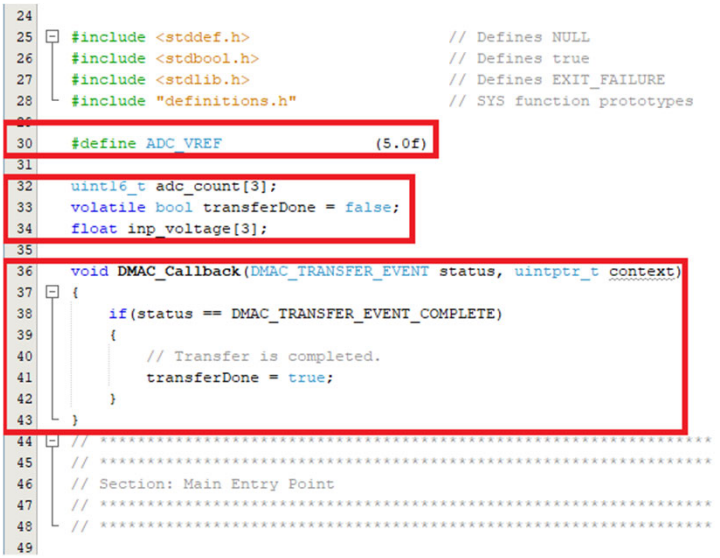
```
#define ADC_VREF                (5.0f)

uint16_t adc_count[3];
volatile bool transferDone = false;
float inp_voltage[3];
```

2. Add a DMAC callback function outside the `main()` function in the `main.c` file.

```
void DMAC_Callback(DMAC_TRANSFER_EVENT status, uintptr_t context)
{
    if(status == DMAC_TRANSFER_EVENT_COMPLETE)
    {
        // Transfer is completed.
        transferDone = true;
    }
}
```

Figure 4-1. Adding Macro and DMAC Callback Function and Variables



3. Add the `printf` statements, DMAC callback register function, DMAC channel transfer function, ADC enable function, and the Timer Start function inside the `main()` function as shown below:

```
printf("\n\r-----------------------------------------------------------");
printf("\n\r                  ADC Sequencer Demo                      ");
printf("\n\r-----------------------------------------------------------\n\r");

TC0_TimerStart();

ADC0_Enable();

DMAC_ChannelCallbackRegister(DMAC_CHANNEL_0, DMAC_Callback, 0);
DMAC_ChannelTransfer(DMAC_CHANNEL_0, (const void *)&ADC0_REGS->ADC_RESULT, &adc_count,
sizeof(adc_count));
```

4. Inside the while loop, add the application logic as shown below:

```
if(transferDone == true)
        {
            transferDone = false;
            for(int i = 0; i<3;i++)
```

```
            {
                /* The below formula is used to find the voltage for the
                   corresponding ADC value.
                   Input_Volt = (ADC_count * Ref_Volt)/(2^No.of Bits);
                   Where Ref_Volt = 5V and No. of Bits = 12*/
               inp_voltage[i] = (float)adc_count[i] * ADC_VREF / 4095U;
            }
            printf("\rADC0 Count = 0x%x, ADC0 Voltage = %d.%02d V, "
                   "ADC1 Count = 0x%x, ADC1 Voltage = %d.%02d V, "
                   "ADC2 Count = 0x%x, ADC2 Voltage = %d.%02d V ",
                   adc_count[0], (int)inp_voltage[0], (int)((inp_voltage[0] -
(int)inp_voltage[0])*100.0),
                   adc_count[1], (int)inp_voltage[1], (int)((inp_voltage[1] -
(int)inp_voltage[1])*100.0),
                   adc_count[2], (int)inp_voltage[2], (int)((inp_voltage[2] -
(int)inp_voltage[2])*100.0));
            DMAC_ChannelTransfer(DMAC_CHANNEL_0, (const void *)&ADC0_REGS->ADC_RESULT,
              &adc_count, sizeof(adc_count));
        }
```

Note: The formula used in `printf` statements is to print the integer and the decimal values. In the expression %d.%02d, %d represents the integer value and .%02d represents the decimal value rounded to two places.

Figure 4-2. Adding Application Logic to `main.c`

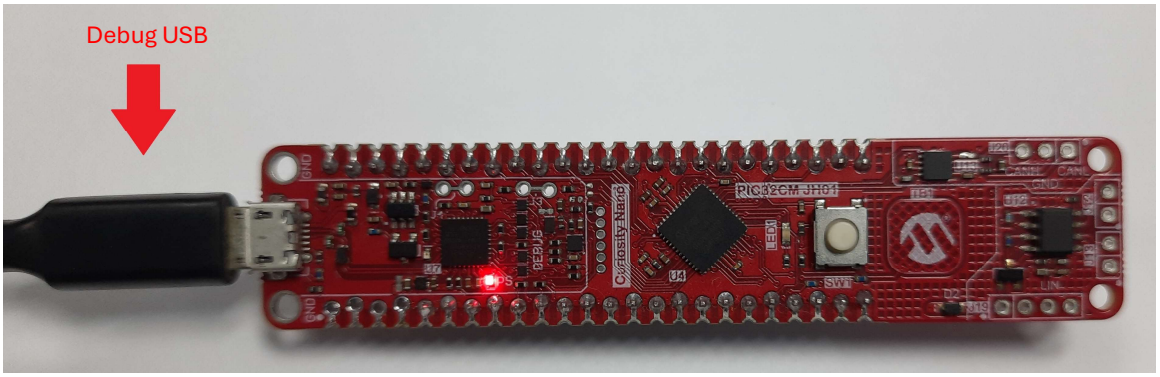
```
50  int main ( void )
51  {
52      /* Initialize all modules */
53      SYS_Initialize ( NULL );
54
55      printf("\n\r-------------------------------------------------------");
56      printf("\n\r                ADC Sequencer Demo                   ");
57      printf("\n\r-------------------------------------------------------\n\r");
58
59      TC0_TimerStart();
60
61      ADC0_Enable();
62
63      DMAC_ChannelCallbackRegister(DMAC_CHANNEL_0, DMAC_Callback, 0);
64      DMAC_ChannelTransfer(DMAC_CHANNEL_0, (const void *)&ADC0_REGS->ADC_RESULT, &adc_count, sizeof(adc_count));
65
66      while ( true )
67      {
68          /* Maintain state machines of all polled MPLAB Harmony modules. */
69          SYS_Tasks ( );
70
71          if(transferDone == true)
72          {
73              transferDone = false;
74
75              for(int i = 0; i<3;i++)
76              {
77                 inp_voltage[i] = (float)adc_count[i] * ADC_VREF / 4095U;
78              }
79
80              printf("\rADC0 Count = 0x%x, ADC0 Voltage = %d.%02d V, "
81                     "ADC1 Count = 0x%x, ADC1 Voltage = %d.%02d V, "
82                     "ADC2 Count = 0x%x, ADC2 Voltage = %d.%02d V ",
83                     adc_count[0], (int)inp_voltage[0], (int)((inp_voltage[0] - (int)inp_voltage[0])*100.0),
84                     adc_count[1], (int)inp_voltage[1], (int)((inp_voltage[1] - (int)inp_voltage[1])*100.0),
85                     adc_count[2], (int)inp_voltage[2], (int)((inp_voltage[2] - (int)inp_voltage[2])*100.0));
86
87              DMAC_ChannelTransfer(DMAC_CHANNEL_0, (const void *)&ADC0_REGS->ADC_RESULT,
88                &adc_count, sizeof(adc_count));
89          }
90      }
91      /* Execution should not come here during normal operation */
92
93      return ( EXIT_FAILURE );
94  }
```

Note: The example code shown above is in the main routine inside while (1). This example code can be run in Standby mode or other sub-routine, where the CPU intervention and CPU load reduction can be observed.

5. Building and Programing the Application

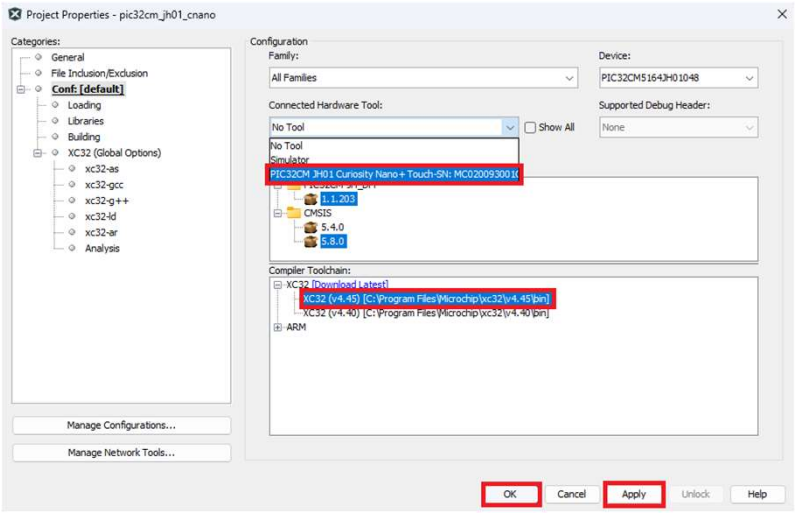
1. The PIC32CM JH01 Curiosity Nano+ Touch Evaluation Kit supports debugging using an Embedded Debugger (EDBG). Connect the Type-A male to micro-B USB cable to the micro-B USB port on the PIC32CM JH01 Curiosity Nano+ Touch Evaluation Kit and Type-A male to the PC.

Figure 5-1. Hardware



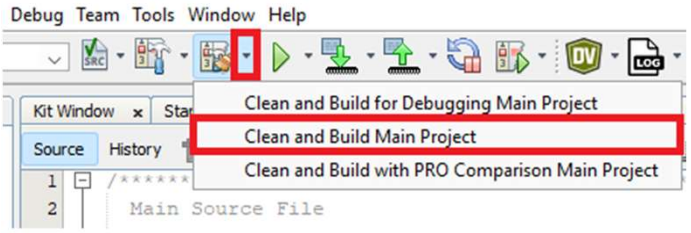
2. In the MPLAB X IDE **Project Properties** window perform these actions:
   a. Under the left **Categories** navigation box, select **Conf: [default]**.
   b. In the right **Configuration** properties sheet, select **Connected Hardware Tool** and **Compiler Toolchain**.

Figure 5-2. Project Properties



3. Click **Apply**, and then click **OK**.
4. To build the project, click on the *Clean and Build* icon or select **Clean and Build Main Project** from the drop-down item list and verify that the project builds successfully. At this point, the application code can be implemented.

Figure 5-3. Clean and Build

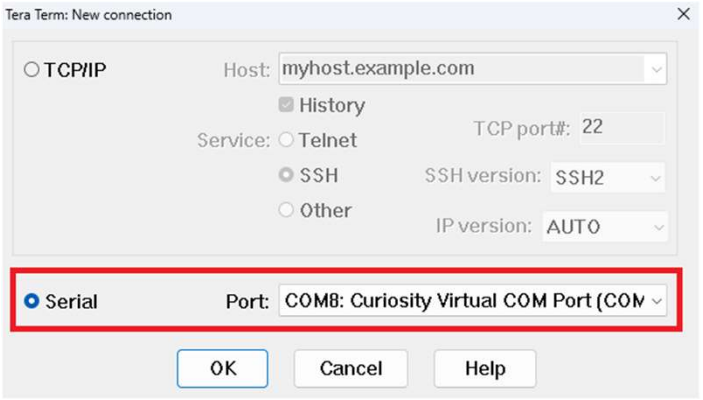


5. Program the application by clicking the highlighted icon below.

Figure 5-4. Program the Device

6. After the application builds and completes programming, open the Tera Term tool on the PC. Select the serial port and set the baud rate to 115200.

Figure 5-5. Selection of Serial Port



7. Click **OK**.
8. Connect the following pins in the PIC32CM JH01 Curiosity Nano+ Touch Evaluation Kit to observe the output as shown in the following figure:
   - PA02 (ADC0) - GND
   - PB08 (ADC2) - GND
   - PA04 (ADC4) - V_EDGE (5V)

Figure 5-6. Hardware Setup

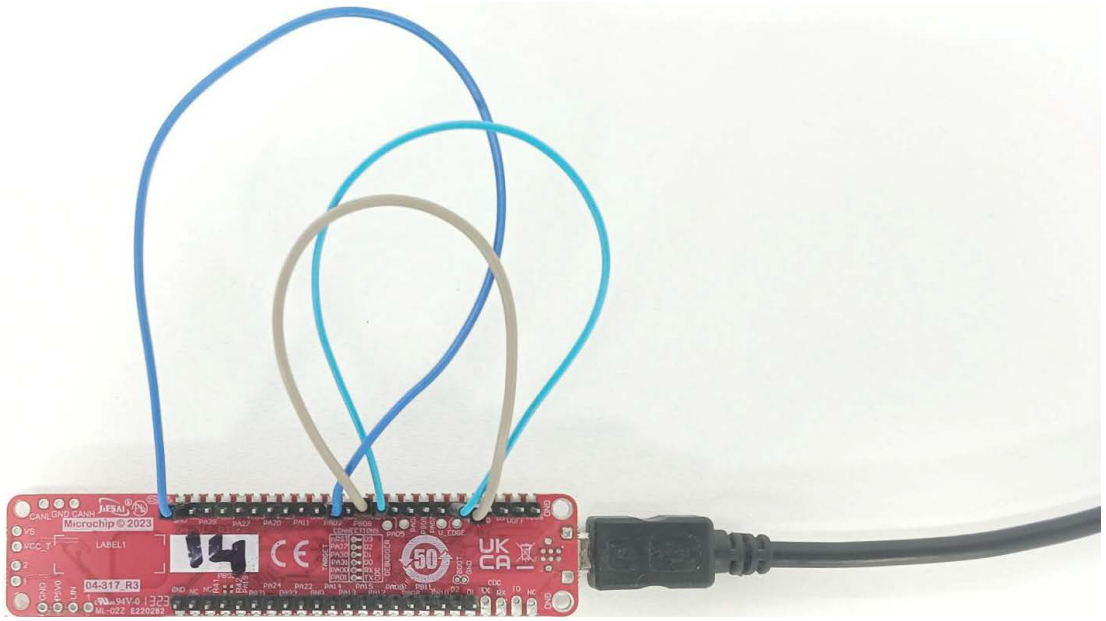


Figure 5-7. ADC Output - 1

9. Connect the following pins to observe the output as shown below:
   - PA02 (ADC0) - GND
   - PB08 (ADC2) - V_EDGE (5V)
   - PA04 (ADC4) - GND

Figure 5-8. ADC Output - 2

6. Resources

- PIC32CM JH01 Curiosity Nano+ Touch Evaluation Kit User's Guide
- Creating the First Application on PIC32CM JH01 Microcontrollers Using MPLAB Harmony v3 with MPLAB Code Configurator (MCC)
- For Document section go to PIC32CM JH01 Curiosity Nano+ Touch Evaluation Kit
- For the example application, refer to Getting Started Application on PIC32CM JH01 Curiosity Nano+ Touch Evaluation Kit under the “Software” heading
- For additional information on MPLAB® Harmony v3, refer to the Microchip web site: https://www.microchip.com/en-us/tools-resources/configure/mplab-harmony and https://developerhelp.microchip.com/xwiki/bin/view/software-tools/harmony/
- For more information on various applications, refer to: github.com/Microchip-MPLAB-Harmony/reference_apps
- For additional information about 32-bit Microcontroller Collaterals and Solutions, refer to the 32-bit Microcontroller Collateral and Solutions Reference Guide: ww1.microchip.com/downloads/aemDocuments/documents/MCU32/ProductDocuments/ReferenceManuals/32-bit-Microcontroller-Collateral-and-Solutions-Reference-Guide-DS70005534.pdf
- For other relevant information, refer to the Microchip web site: www.microchip.com/

Microchip Information

Trademarks

The “Microchip” name and logo, the “M” logo, and other names, logos, and brands are registered and unregistered trademarks of Microchip Technology Incorporated or its affiliates and/or subsidiaries in the United States and/or other countries (“Microchip Trademarks”). Information regarding Microchip Trademarks can be found at https://www.microchip.com/en-us/about/legal-information/microchip-trademarks.

ISBN: 979-8-3371-0356-3

Legal Notice

This publication and the information herein may be used only with Microchip products, including to design, test, and integrate Microchip products with your application. Use of this information in any other manner violates these terms. Information regarding device applications 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. Contact your local Microchip sales office for additional support or, obtain additional support at www.microchip.com/en-us/support/design-help/client-support-services.

THIS INFORMATION IS PROVIDED BY MICROCHIP “AS IS”. 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 ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE, OR WARRANTIES RELATED TO ITS CONDITION, QUALITY, OR PERFORMANCE.

IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE, INCIDENTAL, OR CONSEQUENTIAL LOSS, DAMAGE, COST, OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE INFORMATION OR ITS USE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE FULLEST EXTENT ALLOWED BY LAW, MICROCHIP’S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THE INFORMATION OR ITS USE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THE INFORMATION.

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.

Microchip Devices Code Protection Feature

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

- Microchip products meet the specifications contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is secure when used in the intended manner, within operating specifications, and under normal conditions.
- Microchip values and aggressively protects its intellectual property rights. Attempts to breach the code protection features of Microchip products are strictly prohibited and may violate the Digital Millennium Copyright Act.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of its code. Code protection does not mean that we are guaranteeing the product is “unbreakable”. Code protection is constantly evolving. Microchip is committed to continuously improving the code protection features of our products.