Getting Started With the Arm® Cortex®-M0+ Based PIC32CM Microcontroller Families

AN6368

Product Page Links

Introduction

This document provides a comprehensive introduction to PIC32CM microcontrollers (MCUs) — based on the Arm® Cortex®-M0+ processor — and demonstrates how to efficiently evaluate, prototype, and develop applications using the hardware and software tools supported by Microchip. To accelerate development, this guide outlines the essential tools, workflows, and resources required to establish a robust development environment using MPLAB® Tools for Visual Studio Code (VS Code®), Common Microcontroller Software Interface Standard (CMSIS) framework, and MPLAB Code Configurator (MCC). The document also demonstrates how to replicate the same process using MPLAB X Integrated Development Environment (IDE).

This document helps readers get started with the Arm Cortex-M0+ based PIC32CM MCU families by covering device selection, toolchain configuration, and providing a step-by-step guide for creating example applications using the Curiosity Nano platform and various software development tools. It further shows where to find code examples that use Arm Cortex-M0+ based MCUs, enabling a deep-dive exploration into the architectural features and key peripherals of these microcontrollers. For detailed device-specific characteristics, refer to the respective data sheets and silicon errata. In this document, the PIC32CM family microcontroller is used as a reference, however, the information is relevant for all PIC32CM microcontroller families.

Table of Contents

1. Get the Device Data Sheet

All Arm Cortex-M0+ based PIC32CM microcontroller families have dedicated family pages, and each microcontroller has a product page where users can access data sheets and errata documents.

For instance, the useful web pages for documentation related to the PIC32CM PL10 family are:

- Family Overview Page: www.microchip.com/en-us/products/microcontrollers/32-bit-mcus/pic32-sam/pic32cm-pl10

The documentation for the PIC32CM microcontroller family is divided into two types:

- Data sheet (includes device description, number of peripherals, pinout, and electrical characteristics)
- Errata (includes known issues for the device)

Find the latest information on all Arm Cortex-M0+ based PIC32CM microcontrollers at https://www.microchip.com/en-us/products/microcontrollers/32-bit-mcus/pic32-sam.

2. Tools and Software

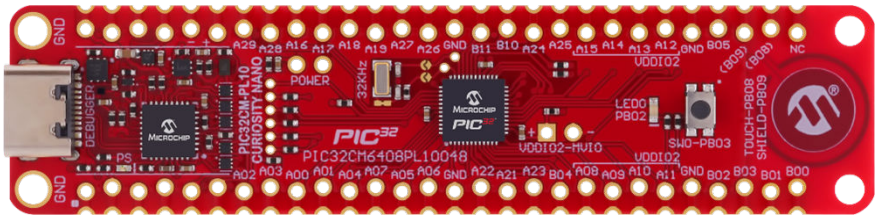
VS Code and MPLAB X IDE with the appropriate extensions can be used as development environments to start working with the PIC32CM family, both utilizing the XC32 compiler.

2.1. Get Evaluation Kits

2.1.1. Curiosity Nano Evaluation Kit

The Curiosity Nano is a compact hardware platform for rapid prototyping and hands-on evaluation of Microchip microcontrollers. It features an on-board debugger and breaks out the MCU pins to an edge connector, enabling easy access for testing and development.

Figure 2-1. Overview of the PIC32CM Curiosity Nano Evaluation Kit



Web page: www.microchip.com/en-us/development-tool/ev10p22a

Get the kit: www.microchipdirect.com/ProductSearch.aspx?Keywords=EV10P22A

For additional information, refer to the PIC32CM PL10 Curiosity Nano Hardware User Guide (*DS50004003*).

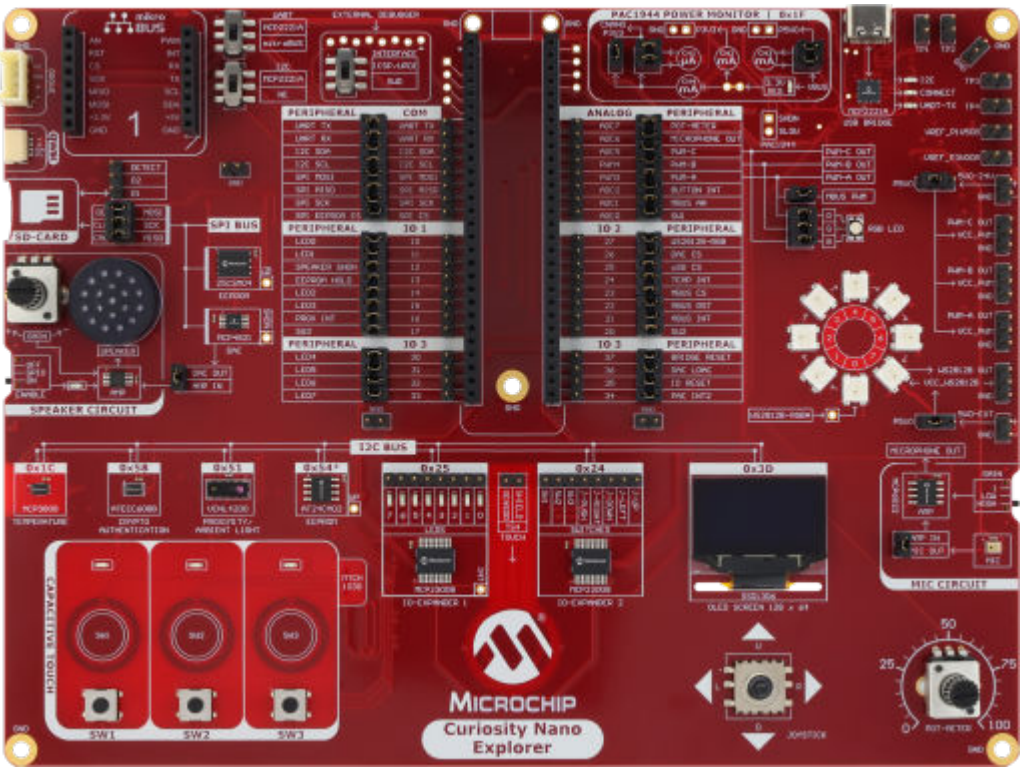
PIC32CM PL10 Curiosity Nano Key Features:

- PIC32CM6408PL10048 Microcontroller
- One Yellow User LED
- One Mechanical Switch
- One Capacitive Touch Button
- 32.768 kHz Crystal
- USB Powered
- On-Board Debugger:
  - Board identification in MPLAB X IDE and VS Code
  - One green power and status LED
  - Programming and debugging
  - Virtual COM port (CDC)
  - One debug GPIO channel (DGI GPIO)

2.1.2. Microchip Curiosity Nano Explorer Evaluation Kit

The Microchip Curiosity Nano Explorer lives up to its name by providing extensive on-board features that allow users to explore and experiment with the microcontroller peripherals of their Curiosity Nano development board. It also serves as an ideal platform to become familiar with Microchip's software offerings, including MPLAB Code Configurator (MCC).

Figure 2-2. Overview of Curiosity Nano Explorer Evaluation Kit



Web page: www.microchip.com/DevelopmentTools/ProductDetails/PartNO/EV58G97A

Get the kit: www.microchipdirect.com/dev-tools/EV58G97A

For additional information, refer to the Curiosity Nano Explorer User Guide (*DS50003716*).

2.2. Get Programmers and Debuggers

Microchip offers a range of programmers, debuggers, and extensions to support all device architectures. All solutions, including in-circuit debuggers and programmers, are USB-powered and fully integrated into their respective environments. The MPLAB ICD 5 In-Circuit Debugger offers debugging and hardware features suitable for most users. The MPLAB Snap In-Circuit Debugger and MPLAB PICkit™ 5 In-Circuit Debuggers are economical choices for basic debugging functionalities. Choosing a programmer or debugger depends on the type of development the engineer plans to perform. The MPLAB Snap is a low-cost, low-feature programmer/debugger; the PICkit 5 is a mid-range with more features than the Snap; and the high-feature functioning programmer/debugger is intended for engineers developing products for commercial or industrial applications.

- MPLAB PICkit 5 In-Circuit Debugger
  - Product Page
  - Buy Here
  - User Guide
  - Microchip Developer Help

Figure 2-3. PICkit5 In-Circuit Debugger

- MPLAB ICD 5 In-Circuit Debugger
  - Product Page
  - Buy Here
  - User Guide
  - Microchip Developer Help

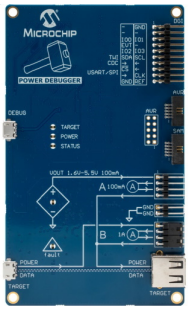
Figure 2-4. In-Circuit Debugger (ICD)

- Atmel ICE
  - Product Page
  - Buy Here
  - User Guide
  - Microchip Developer Help

Figure 2-5. Atmel ICE

- Power Debugger
  - Product Page
  - Buy Here
  - User Guide
  - Microchip Developer Help

Figure 2-6. Power Debugger



2.3. Get Device Support for MPLAB Code Configurator (MCC)

Web page: MPLAB Code Configurator

- MCC Harmony

MCC Harmony can be utilized within MPLAB X IDE as a graphical programming interface that generates peripheral and library code for the PIC32CM family.

For a step-by-step guide on setting up the necessary tools — including MPLAB X IDE, XC32 Compiler, MCC, and accessing the Harmony v3 framework — check out this video: How to Set up the Tools Required to Get Started with MPLAB® Harmony v3 and MCC.

2.4. MPLAB Tools for VS Code

The MPLAB Tools for VS Code are available for download from the official MPLAB Tools website or directly through the VS Code Marketplace. Both platforms offer a complete list of MPLAB extensions, which can be installed individually or collectively as part of the MPLAB Extensions Pack. Installation is performed through the VS Code Extensions interface.

A step-by-step guide for getting started with the MCC plugin in VS Code is available on the official website, including links to the Extensions Developer Help and the MPLAB Tools for VS Code video playlist.

2.4.1. MPLAB Plugins

To enhance the functionality of MPLAB, users are encouraged to install the following additional plugins:

- Clangd for MPLAB: Provides support for Clang-based compilation and advanced code analysis
- Code Configurator (MCC) for Microchip devices: Enables graphical configuration of peripherals and code generation for supported devices
- Toolchain Support for MPLAB: Integrates the XC8, XC16, and XC32 compilers for optimized code generation for Microchip devices

2.4.2. Get Device Support for VS Code

The MPLAB extension handles pack installation for PIC32 devices, so users do not need to explicitly install any packs.

2.5. Get Code Examples from MPLAB Discover

MPLAB Discover bundles available resources from multiple sources into one portal, such as code examples, data sheets, evaluation boards, videos, and blogs.

Web page: MPLAB Discover

Code Examples

Code examples for devices in the PIC32CM family can be found by searching for the device name, e.g., PIC32CM6408PL10048, in the search bar.

Users can download code examples as a `.zip` file directly from MPLAB Discover. Repositories ending with *"mplab-mcc"* can be opened in MPLAB X.

When a code example is hosted on GitHub, MPLAB Discover provides an *"Open with GitHub"* link. Download the example code from there or use the Git tool on the PC to create a local repository clone.

2.6. Get MPLAB X IDE

Web page: MPLAB X IDE

- MPLAB X

MPLAB X can be used as an IDE for developing and debugging firmware for the PIC32CM family. For device support, refer to the Get Device Support for MPLAB X IDE section.

2.7. Get Device Support for MPLAB X IDE

Use the MPLAB Pack Manager, found under *Tools>Packs*, to add support for new devices in MPLAB X.

For the PIC32CM family, update to the latest version by performing the following steps:

1. Click **Check for Updates**.
2. Select the latest available version of PIC32CM Series Device Support. For instance, for the PIC32CM-PL10 family, select the latest version of PIC32CM-PL series device support.
3. Click **Install**.

For offline installers, go to packs.download.microchip.com. Double-click the installer file and follow the instructions to install the package. Any open MPLAB X window must be closed for the installation to take effect.

2.8. Get IAR Embedded Workbench® for Arm

Web page: www.iar.com/iar-embedded-workbench/

2.9. Get Arm Keil MDK for Cortex -M Processors

Web page: https://developer.arm.com/Tools%20and%20Software/Keil%20MDK

2.10. Get Segger Embedded Studio

Web page: https://www.segger.com/products/development-tools/embedded-studio/

2.11. Configuring Compilers

For the PIC32CM family of devices, which are 32-bit microcontrollers, use the XC32 compiler.

Web page: XC32 Compilers

3. Getting Started PIC32CM MCU With VS Code and MCC

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

- VS Code
- VS Code Plugins:
  a. MPLAB
  b. Toolchain Support for MPLAB
  c. Code Configurator (MCC) for Microchip devices
- MPLAB Harmony v3 repository: csp v3.25 or newer
- PIC32CM Curiosity Nano Evaluation Kit

Note: The latest versions of these tools can also be used to develop the application.

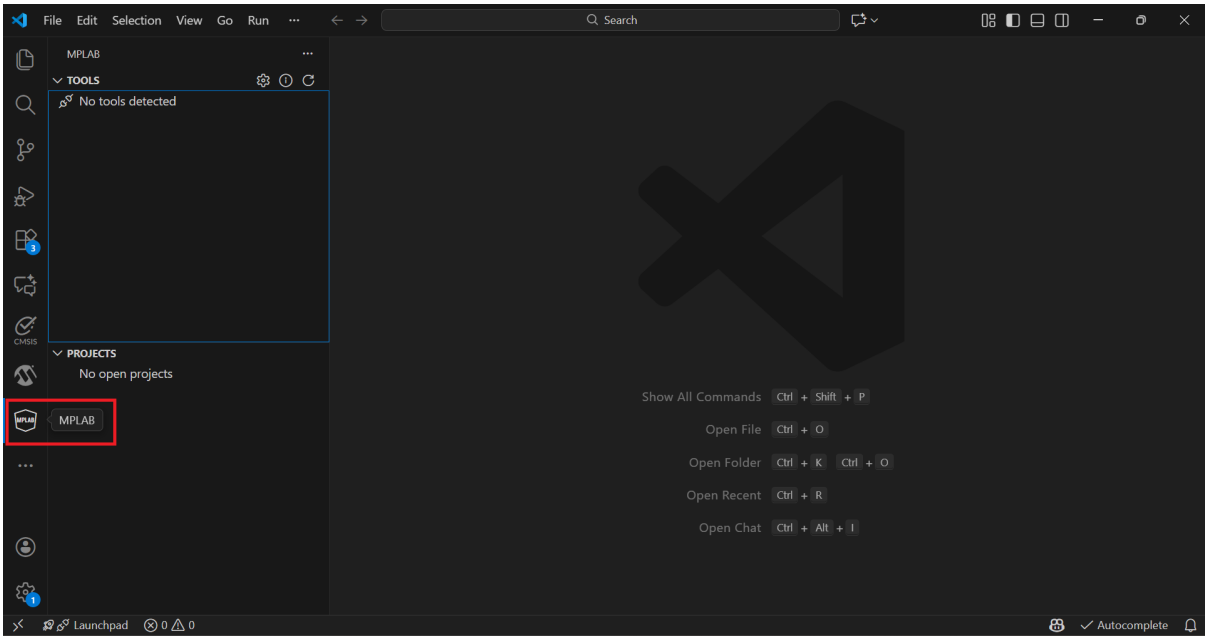
3.1. Creating the First Application on the PIC32CM MCU

For new projects, it is recommended to use the MPLAB Tools for VS Code. If not already installed, VS Code can be downloaded from the official Visual Studio Code website and installed following the standard setup procedure. Once installed, the MPLAB Tools for VS Code can be downloaded from the official MPLAB Tools website or directly from the VS Code Marketplace. Both resources offer a complete list of MPLAB extensions, which can be installed individually or collectively as part of the MPLAB Extensions Pack. Installation is performed through the VS Code Extensions interface. Additionally, tutorials are available on the official website, including links to the Extensions Developer Help and the MPLAB Tools for VS Code video playlist. Refer to the Tools and Software section for instructions on installing the basic VS Code setup.

To create a VS Code-based project, follow these steps:

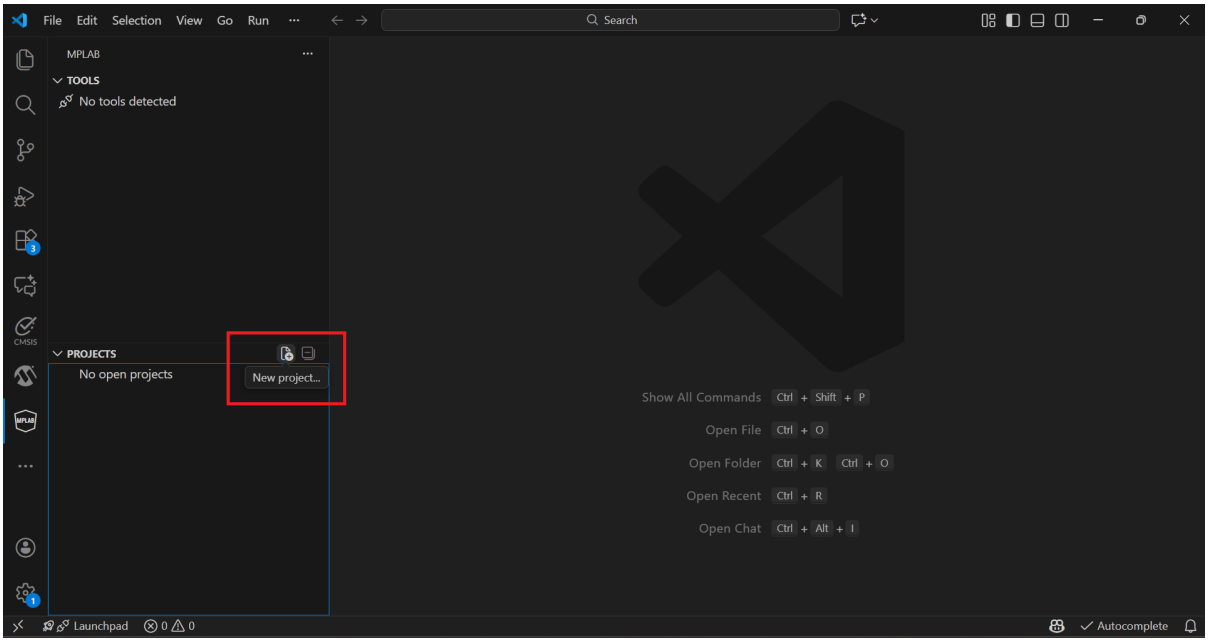
1. On the Start menu, launch VS Code.
2. On the left navigation bar, click the MPLAB plugin.

Figure 3-1. MPLAB Plugin Selection



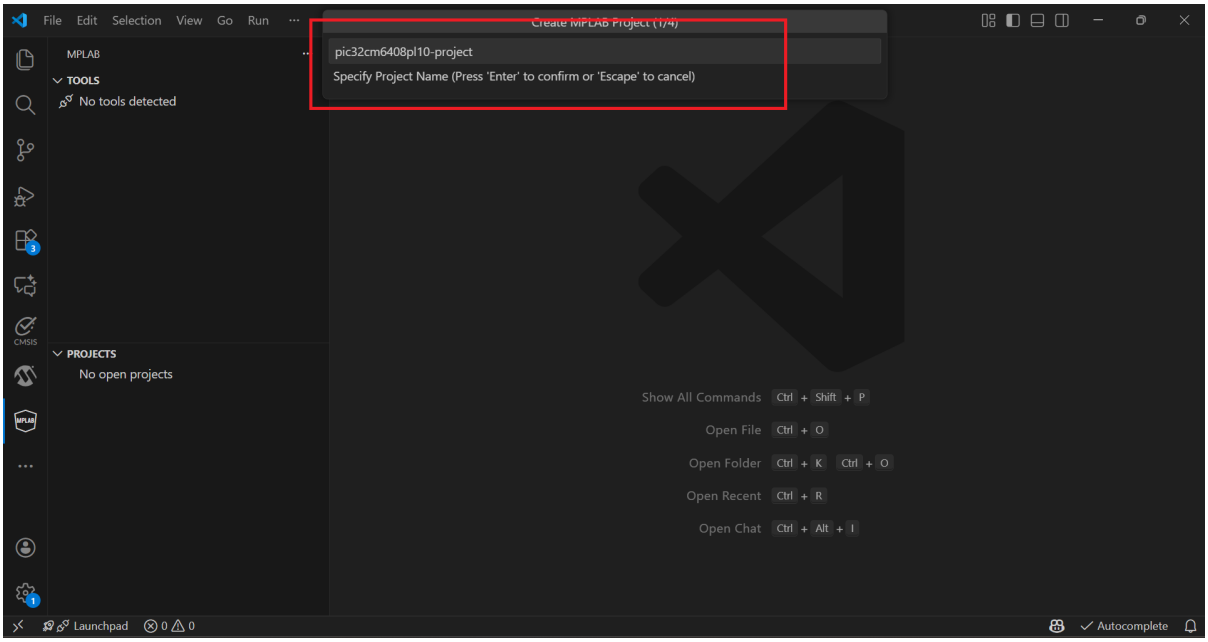
3. Under the **Project** tab window, click the New Project option to create a new MPLAB project.

Figure 3-2. New Project Window



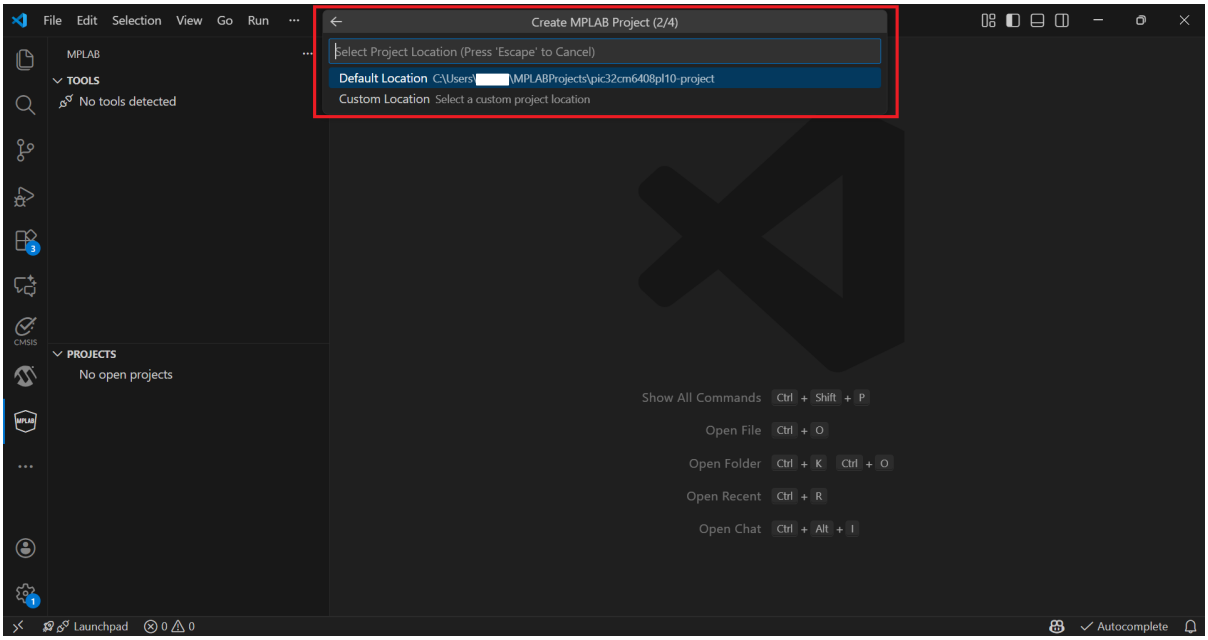
4. Enter the project title in the search window and press **Enter**.

Figure 3-3. Creating New Project



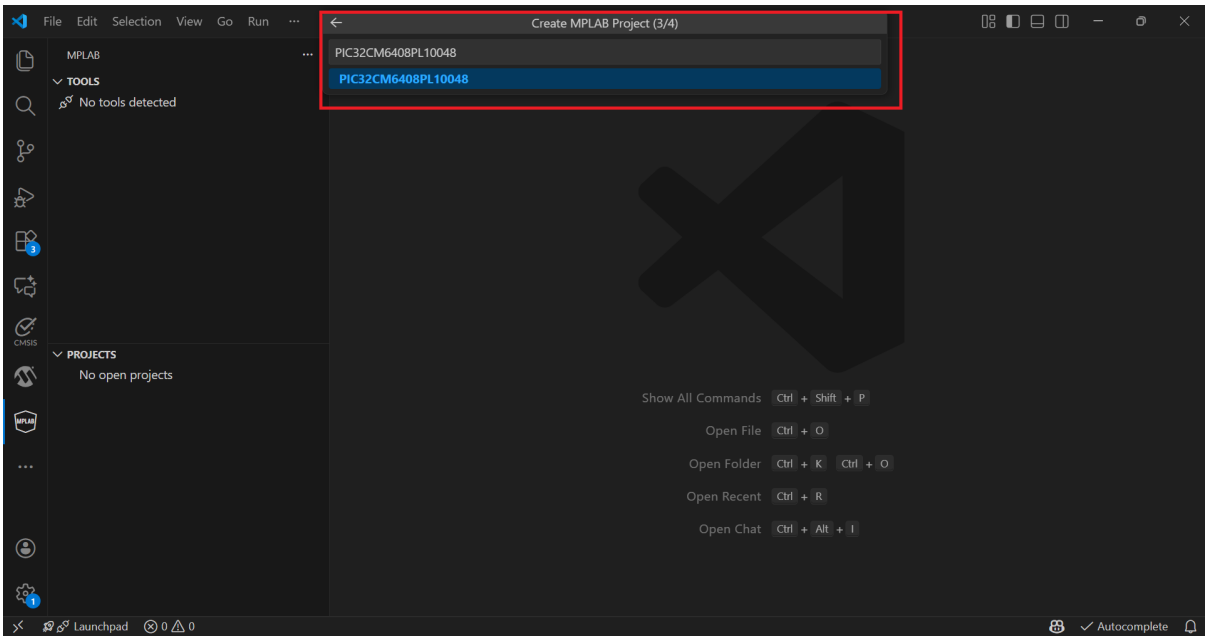
5. Select the location where the new project will be created. Here, the default location is selected. Press **Enter** to proceed.

Figure 3-4. Selection of New Project Location



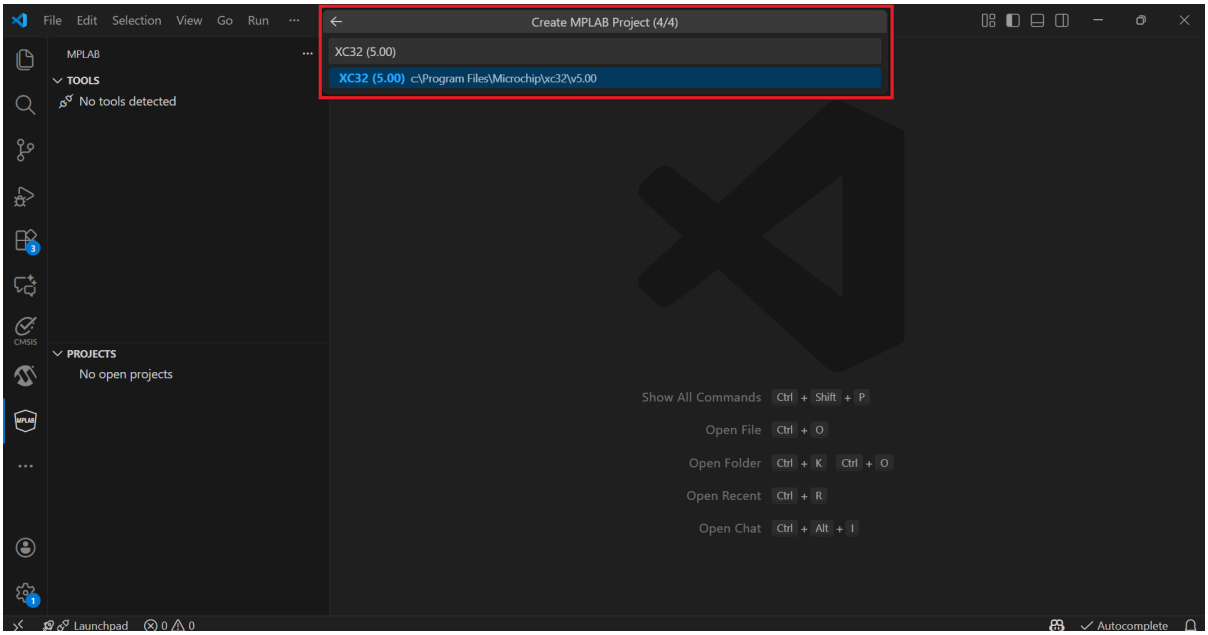
6. Select the `PIC32CM6408PL10048` device and press **Enter**.

Figure 3-5. Device Selection



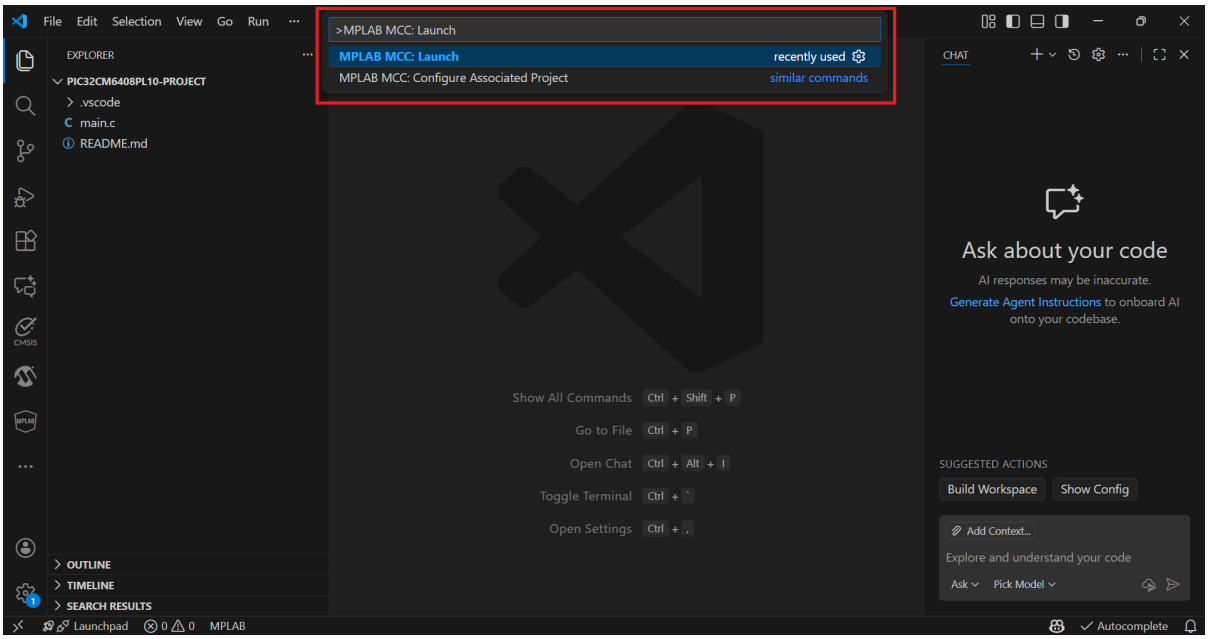
7. Select the latest compiler version (`XC32 (5.00)`) and press **Enter**.

Figure 3-6. Compiler Selection



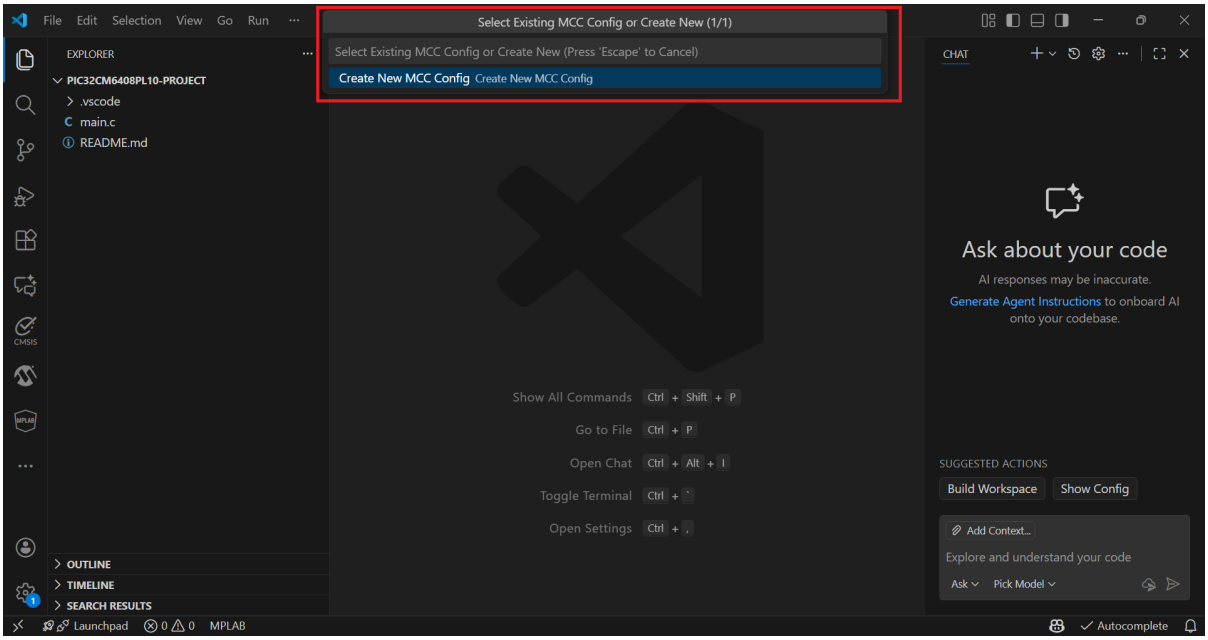
8. Type `>MPLAB MCC: Launch` in the search window.

Figure 3-7. Launch MCC



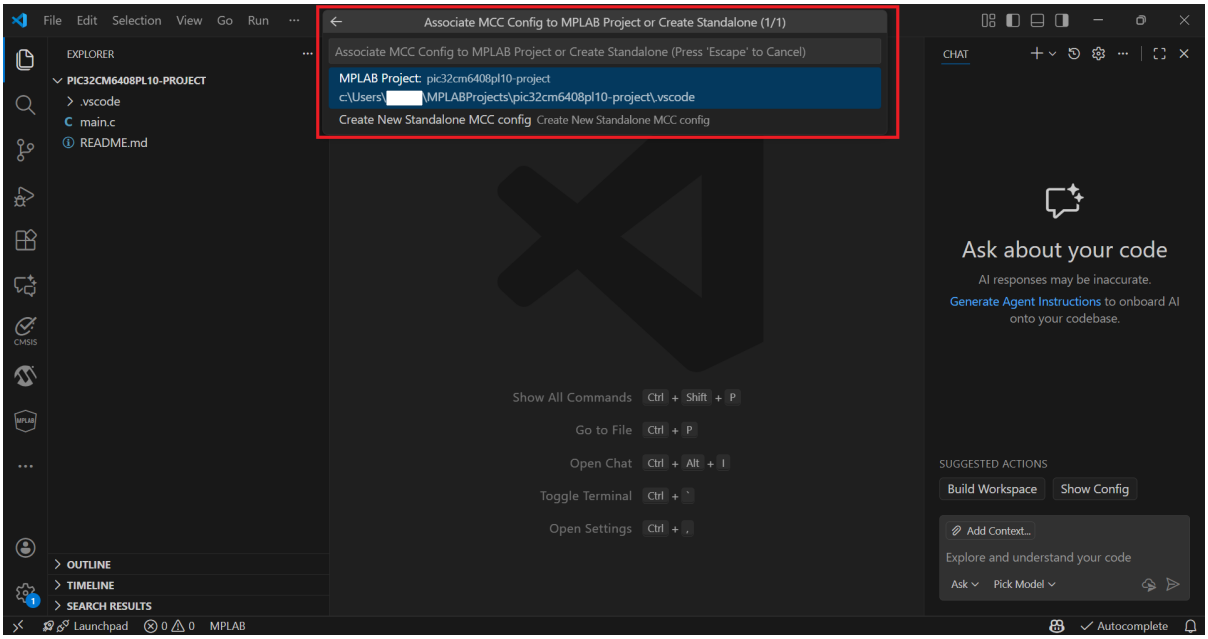
9. Select the Create New MCC Config option.

Figure 3-8. Select New MCC Config



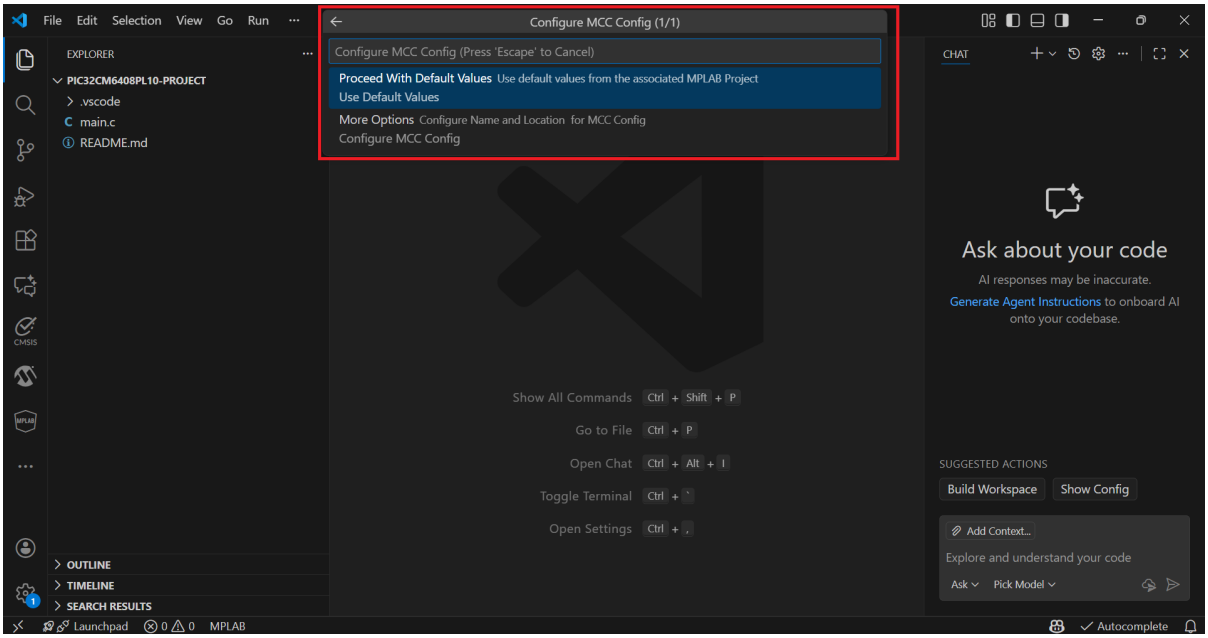
10. Link the MPLAB project created with MCC.

Figure 3-9. Associate MCC Config to MPLAB Project



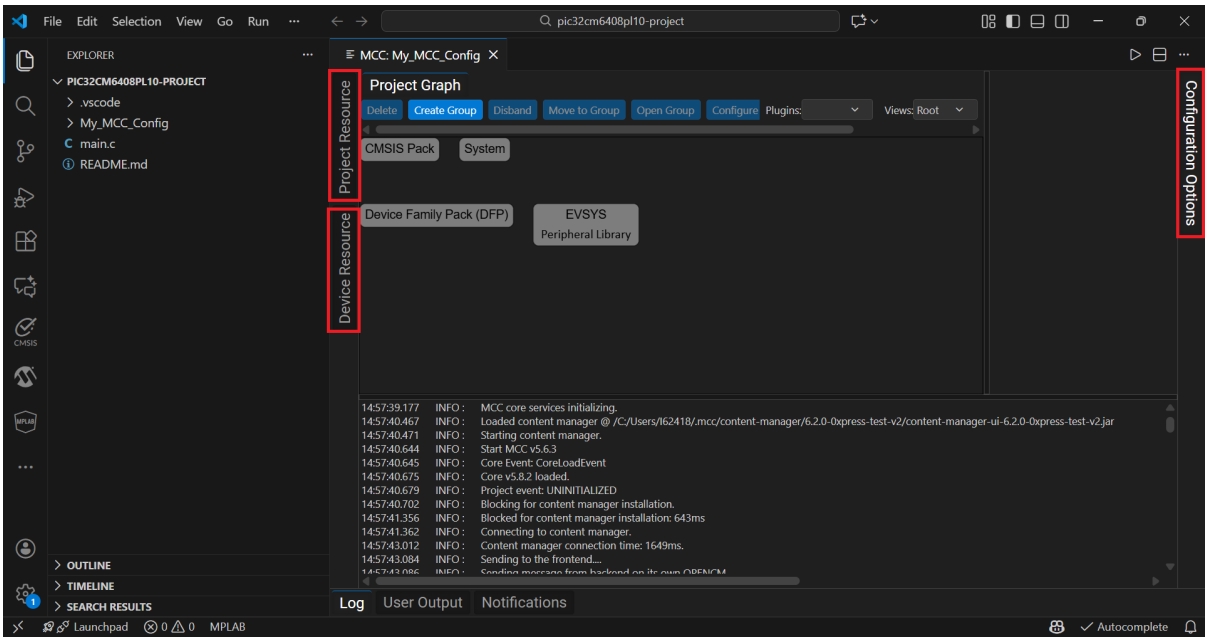
11. Configure the MCC configuration and select the Proceed With Default Values option.

Figure 3-10. Configure MCC Config



12. Before launching MCC, the Configuration Database Setup window will be displayed, where the Device Family Pack (DFP) and Common Microcontroller Software Interface Standard (CMSIS) path can be changed if required. For this demonstration, the default settings are used.
13. The MCC plugin will open in a new window as shown in the following figure.

Figure 3-11. MPLAB Code Configurator Window

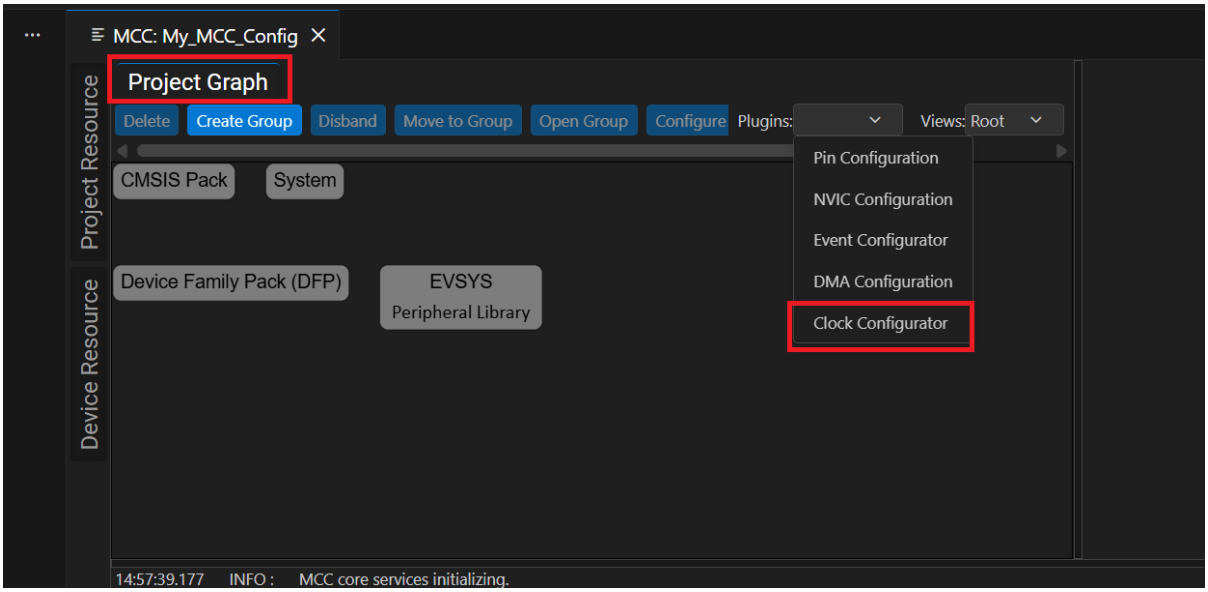


3.2. Adding and Configuring the MPLAB Harmony Component

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

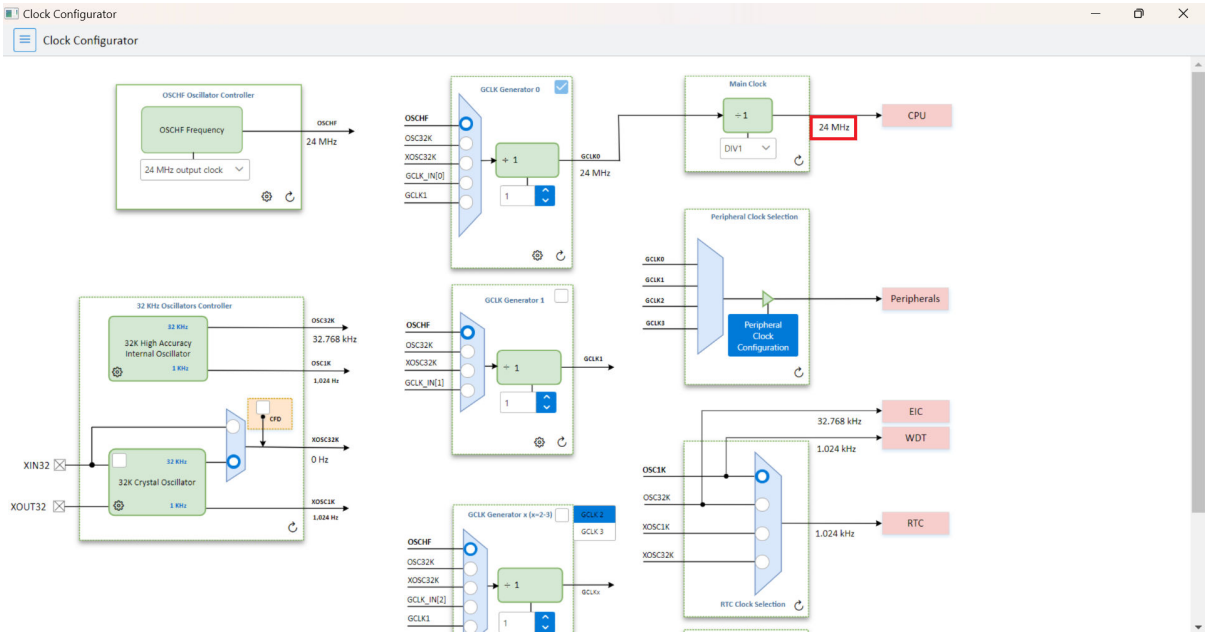
1. In the MCC window, click Project Graph.

Figure 3-12. MPLAB Code Configurator



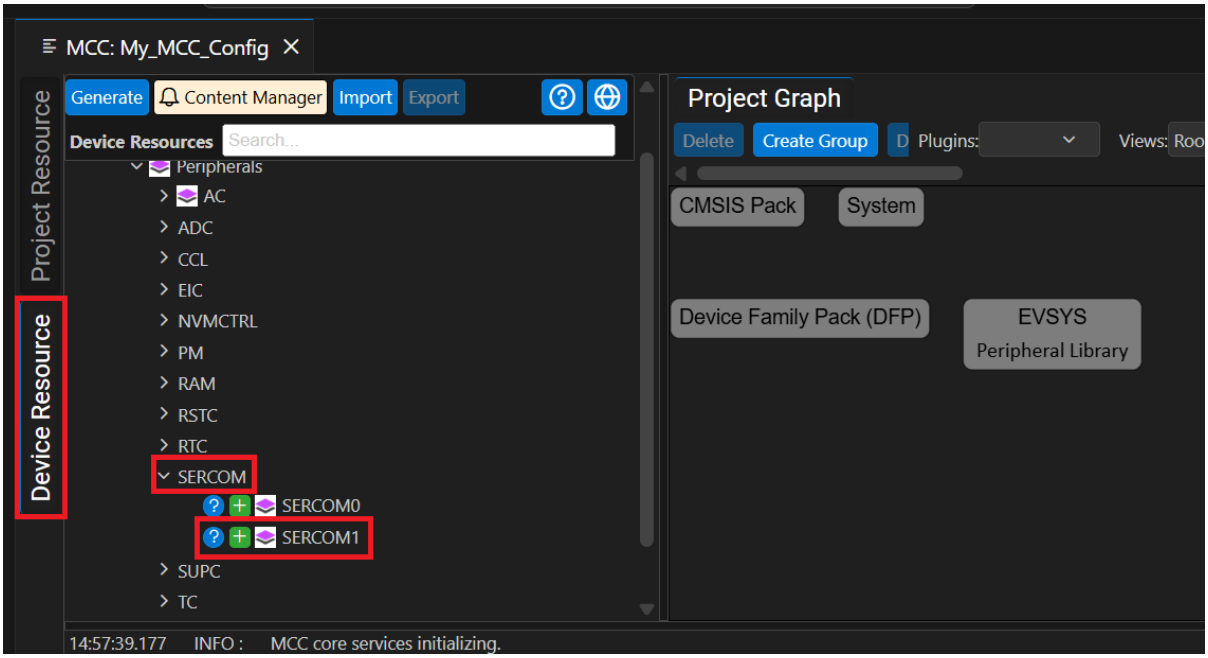
2. In the Plugins drop-down list, select Clock Configuration. The Clock Easy View window will be displayed, verify that the Main Clock is set to 24 MHz.
   Note: Ensure that the following modification is made for GCLK Generator 0.

Figure 3-13. MPLAB Code Configurator - GCLK Generator 0



3. Under Device Resources, expand the list of options: *Harmony>Peripherals>SERCOM*.
4. Click SERCOM and observe that the SERCOM1 Peripheral Library block is added to the Project Graph window.

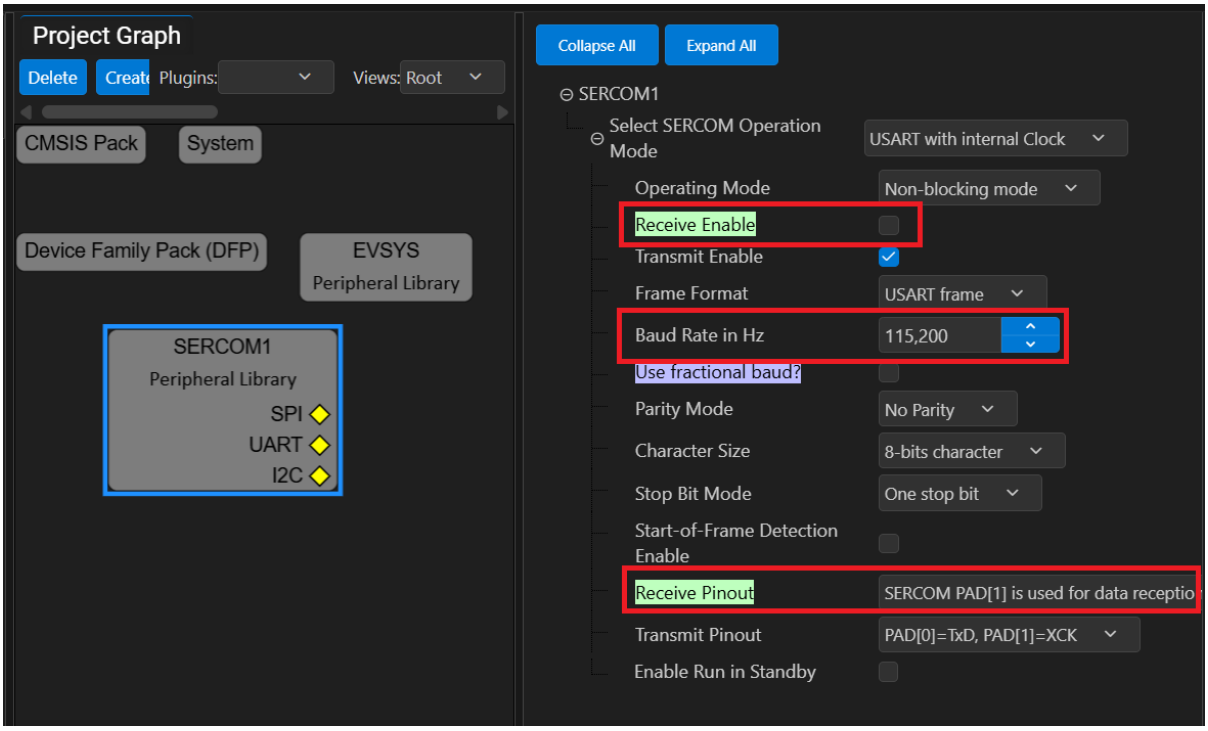
Figure 3-14. MPLAB Code Configurator - Selection of Peripheral



Note: Users can also select other peripherals under Device Resources: *Harmony>Peripherals*.

5. In the Project Graph window, on the left navigation bar, select the SERCOM1 Peripheral Library. In the Configuration Options property page on the right, configure it as follows to print the data on the Serial Console at a baud rate of 115,200.

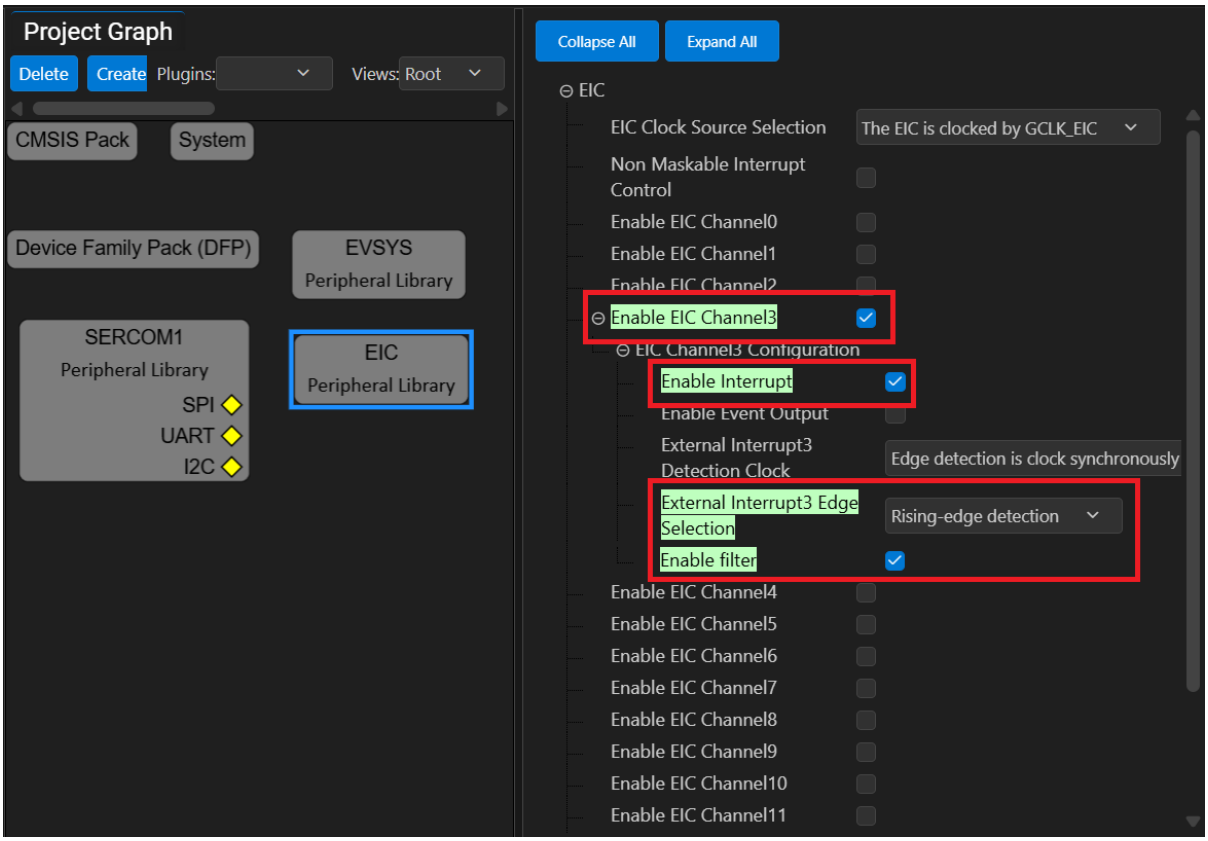
Figure 3-15. MPLAB Code Configurator – SERCOM1 Configuration



6. Under Device Resources, expand the list of options: *Harmony>Peripherals>EIC*. Click EIC and observe that the EIC Peripheral Library block is added in the Project Graph window.

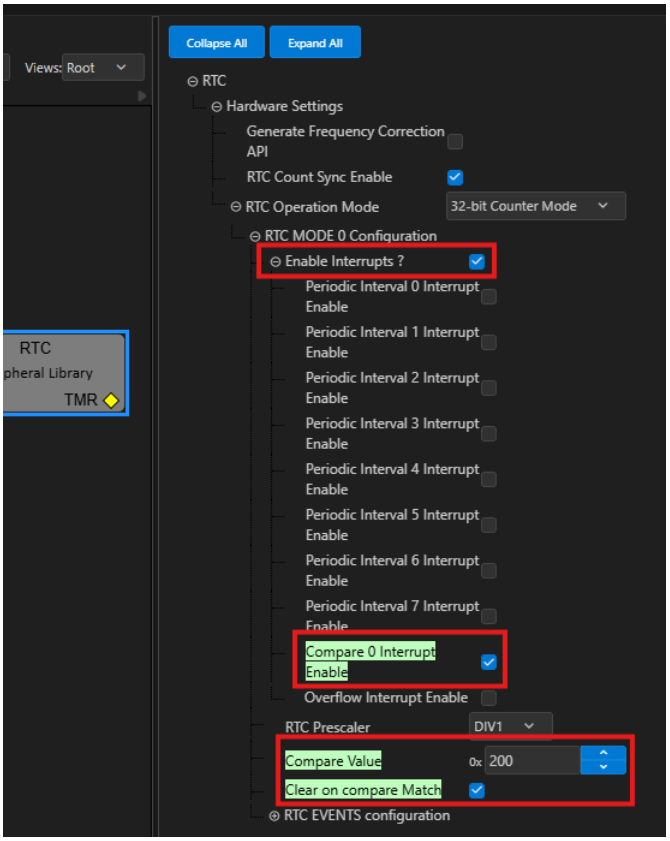
7. In the Project Graph window, on the left navigation bar, select the EIC Peripheral Library. In the Configuration Options property page on the right, select the Enable Interrupt checkbox and the Enable EIC Channel3 checkbox for the switch press event.

Figure 3-16. MPLAB Code Configurator – EIC Configuration



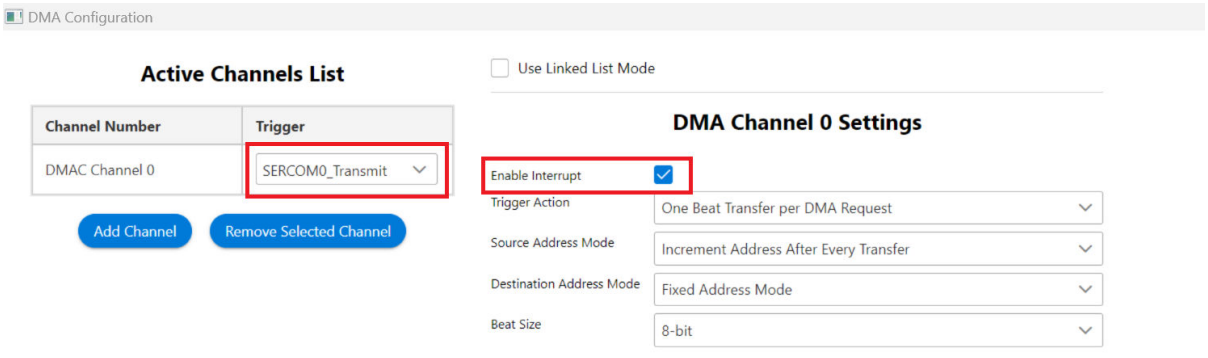
8. Under Device Resources, expand the list of options: *Harmony>Peripherals>RTC*. Click RTC and observe that the RTC Peripheral Library block is added to the Project Graph window to generate a compare interrupt every 500 ms. Select the Clear on compare Match checkbox.
   Note: The compare value is set as 0x200. This value generates an RTC compare interrupt every 500 ms.
   RTC Clock = 1024 Hz
   RTC Prescaler = 1
   Required Interrupt rate = 500 ms
   Therefore, the compare value = (500/1000) x 1024 = 512 (i.e., 0x200).

Figure 3-17. MPLAB Code Configurator – RTC Configuration



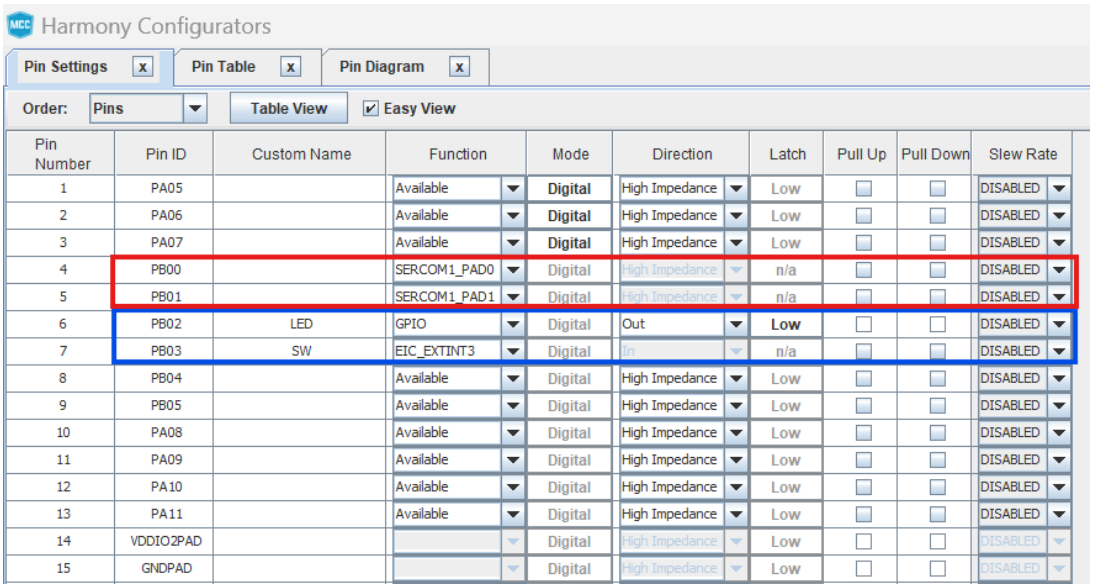
9. From the Plugins drop-down list, select Add Channel and then select DMA Configuration. Configure DMA Channel 0 to transmit the application buffer to the USART TX register. The DMA transfers one byte from the user buffer to the USART transmit buffer on each trigger.

Figure 3-18. MPLAB Code Configurator – DMA Configuration



10. From the Plugins drop-down list, select Pin Configuration and then click **Pin Settings**.
11. In the Order box, type or select Ports. Build configurations according to the application as indicated below. Change the Custom Name of the pin IDs PB02 and PB03, as shown in the following figure.

Figure 3-19. MPLAB Code Configurator – Pin Configuration

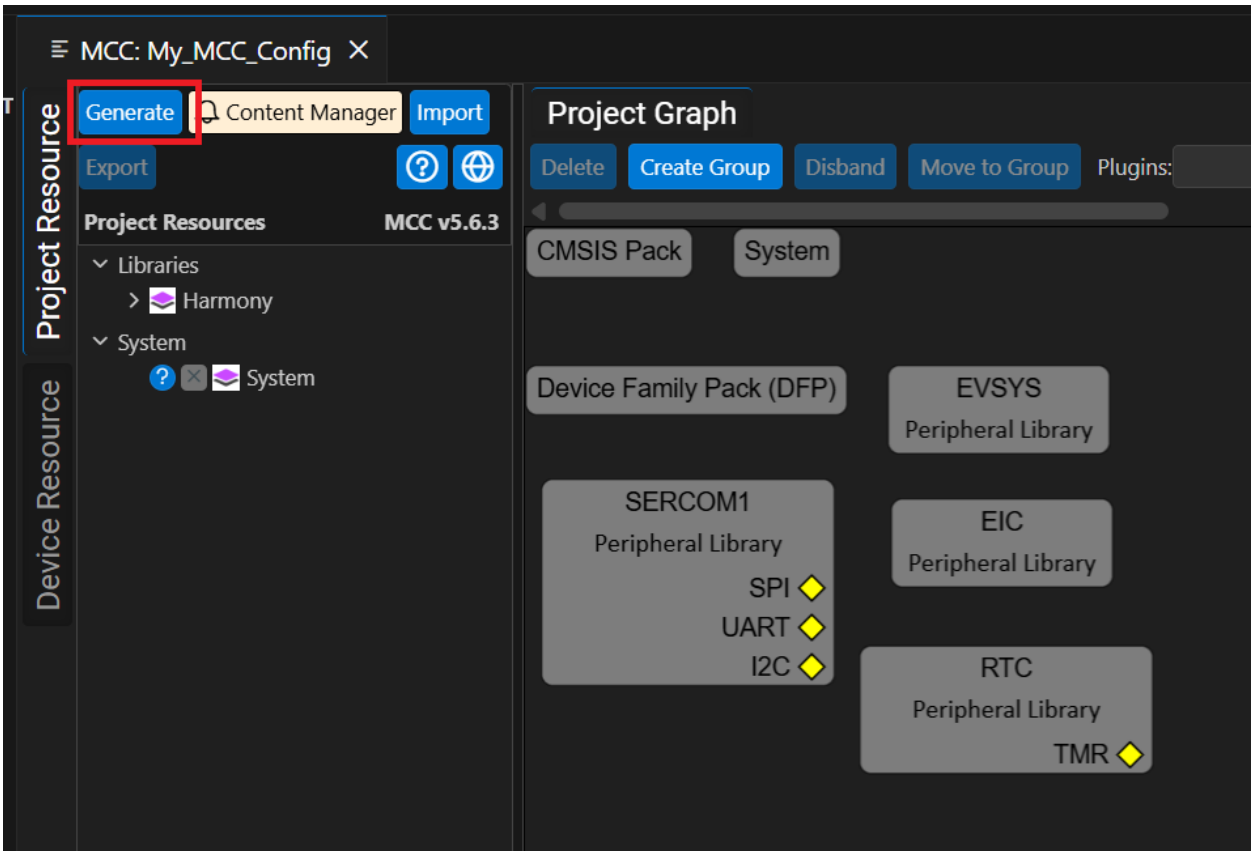


Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch	Pull Up	Pull Down	Slew Rate
1	PA05		Available	Digital	High Impedance	Low			DISABLED
2	PA06		Available	Digital	High Impedance	Low			DISABLED
3	PA07		Available	Digital	High Impedance	Low			DISABLED
4	PB00		SERCOM1_PAD0	Digital	High Impedance	n/a			DISABLED
5	PB01		SERCOM1_PAD1	Digital	High Impedance	n/a			DISABLED
6	PB02	LED	GPIO	Digital	Out	Low			DISABLED
7	PB03	SW	EIC_EXTINT3	Digital	In	n/a			DISABLED
8	PB04		Available	Digital	High Impedance	Low			DISABLED
9	PB05		Available	Digital	High Impedance	Low			DISABLED
10	PA08		Available	Digital	High Impedance	Low			DISABLED
11	PA09		Available	Digital	High Impedance	Low			DISABLED
12	PA10		Available	Digital	High Impedance	Low			DISABLED
13	PA11		Available	Digital	High Impedance	Low			DISABLED
14	VDDIO2PAD			Digital	High Impedance	Low			DISABLED
15	GNDPAD			Digital	High Impedance	Low			DISABLED

3.3. Code Generation

After configuring the peripherals as shown in the following figure, click **Generate** under Resource Management [MCC].

Figure 3-20. Code Generation



Notes:

1. The generated code will add files and folders to the 32-bit MCC Harmony project. In the generated code, notice the Peripheral Library files generated for the Real-Time Clock (RTC), External Interrupt Controller (EIC), PORT peripherals, SERCOM1 (as Universal Synchronous Asynchronous Receiver Transmitter (USART)), and the Direct Memory Access (DMA) peripherals. MCC also generates the `main.c` file.
2. MCC provides an option to change the generated file name. By default, the file name `main.c` is used if a name is not assigned.

3.4. Adding Application Logic

To develop and run the application, follow these steps:

1. Open the `main.c` file of the project and add the following application code.

Figure 3-21. Logic for Register Callback Event Handlers

```
int main ( void )
{
    /* Initialize all modules */
    SYS_Initialize ( NULL );

    /* Register callback for DMA, RTC, and external interrupt */
    DMAC_ChannelCallbackRegister(DMAC_CHANNEL_0, uartDmaChannelHandler_Tx, 0);
    RTC_Timer32CallbackRegister(rtcEventHandler, 0);
    EIC_CallbackRegister(EIC_PIN_3, SW_userHandler, 0);

    /* Print initial message */
    snprintf((char*)uartTxBuffer, TX_BUFFER_SIZE, "************** Printing Toggling LED rate**************\r\n");
    isUARTTxComplete = false;
    DMAC_ChannelTransfer(DMAC_CHANNEL_0, uartTxBuffer, (const void *)&(SERCOM1_REGS->USART.SERCOM_DATA), strlen(

    /* Start the timer */
    RTC_Timer32Start();
```

2. Implement the registered callback event handlers for the peripherals by adding the following code before the `main()` function.

```
static void SW_userHandler(uintptr_t context)
{
    changeTempSamplingRate = true;
}
static void rtcEventHandler (RTC_TIMER32_INT_MASK intCause, uintptr_t context)
{
    if (intCause & RTC_MODE0_INTENSET_CMP0_Msk)
    {
        isRTCExpired = true;
    }
}
static void uartDmaChannelHandler_Tx(DMAC_TRANSFER_EVENT event, uintptr_t contextHandle)
{
    if (event == DMAC_TRANSFER_EVENT_COMPLETE)
    {
        isUARTTxComplete = true;
    }
}
```

3. According to the status of the isRTCExpired and isUARTTxComplete flags (these flags are handled by the rtcEventHandler and uartDmaChannelHandler_Tx event handlers when the RTC timer expires and when UART completes the data transfer), LED0 is toggled at a default rate of 500 ms. To change the toggling rate, if the user presses the SW switch, the toggling rate changes to 1s, 2s, and 4s and then returns to 500 ms with subsequent switch press events. The SW_userHandler is responsible for changing the toggling rate when the user presses the SW switch on the board.

Inside the while loop, delete `SYS_Tasks()` and add the following code to toggle the LED at the default rate of 500 ms.

```
if ((isRTCExpired == true) && (true == isUARTTxComplete))
        {
            isRTCExpired = false;
            isUARTTxComplete = false;
            LED_Toggle();
            sprintf((char*)(uartTxBuffer), "Toggling LED at %s rate
            \r\n",&timeouts[(uint8_t)tempSampleRate][0]);
            DMAC_ChannelTransfer(DMAC_CHANNEL_0, uartTxBuffer,(const void *)&
            (SERCOM1_REGS->USART.SERCOM_DATA),strlen((const char*)uartTxBuffer));
        }
```

4. Add the following code immediately after the code above to change the toggling rate when a switch press event occurs.

```
if(changeTempSamplingRate == true)
        {
            changeTempSamplingRate = false;
            if(tempSampleRate == TEMP_SAMPLING_RATE_500MS)
            {
                tempSampleRate = TEMP_SAMPLING_RATE_1S;
                RTC_Timer32CompareSet(PERIOD_1S);
            }
            else if(tempSampleRate == TEMP_SAMPLING_RATE_1S)
            {
                tempSampleRate = TEMP_SAMPLING_RATE_2S;
                RTC_Timer32CompareSet(PERIOD_2S);
            }
            else if(tempSampleRate == TEMP_SAMPLING_RATE_2S)
            {
                tempSampleRate = TEMP_SAMPLING_RATE_4S;
                RTC_Timer32CompareSet(PERIOD_4S);
            }
            else if(tempSampleRate == TEMP_SAMPLING_RATE_4S)
            {
                tempSampleRate = TEMP_SAMPLING_RATE_500MS;
                RTC_Timer32CompareSet(PERIOD_500MS);
            }
            else
            {
            ;
            }
            RTC_Timer32CounterSet(0);
            sprintf((char*)uartLocalTxBuffer, "LED Toggling rate is changed to %s\r\n",
            &timeouts[(uint8_t)tempSampleRate][0]);
            DMAC_ChannelTransfer(DMAC_CHANNEL_0, uartLocalTxBuffer, (const void *)&
            (SERCOM1_REGS->USART.SERCOM_DATA), strlen((const char*)uartLocalTxBuffer));
        }
```

5. Add the following code to include the necessary header files and define the macros for different RTC compare values.

```
#include <stddef.h>                     // Defines NULL
#include <stdbool.h>                    // Defines true
#include <stdlib.h>                     // Defines EXIT_FAILURE
#include "definitions.h"                // SYS function prototypes
#include <string.h>
#include <stdio.h>
```

This code declares the various flags whose status is monitored and changed by the event handlers in the application. It also includes declarations and definitions of arrays used to print the LED toggling rate on the console.

```
static void SW_userHandler(uintptr_t context);
static void rtcEventHandler (RTC_TIMER32_INT_MASK intCause, uintptr_t context);
static void uartDmaChannelHandler_Tx(DMAC_TRANSFER_EVENT event, uintptr_t contextHandle);

/* Timer Counter Time period match values for input clock of 4096 Hz */
#define PERIOD_500MS        (512)
#define PERIOD_1S           (1024)
#define PERIOD_2S           (2048)
#define PERIOD_4S           (4096)
```

```
#define TX_BUFFER_SIZE      (100)

static volatile bool isRTCExpired = false;
static volatile bool changeTempSamplingRate = false;
static volatile bool isUARTTxComplete = true;
static volatile bool isUARTRxComplete = false;

static uint8_t uartTxBuffer[TX_BUFFER_SIZE] = {0};
```

3.5. Building and Programming Application

To build and program the application, follow these steps:

1. The PIC32CM Curiosity Nano evaluation kit supports debugging using a debugger. Connect the USB Type-C cable to the PIC32CM Curiosity Nano evaluation kit to power and debug the PIC32CM PL10 Curiosity Nano board.
2. Ensure that the compiler optimization is set to the default value (i.e., 1). To check this, select the Fileset option under the **Project** tab. Then select the C compiler option and verify the optimization level.

Figure 3-22. Project Properties

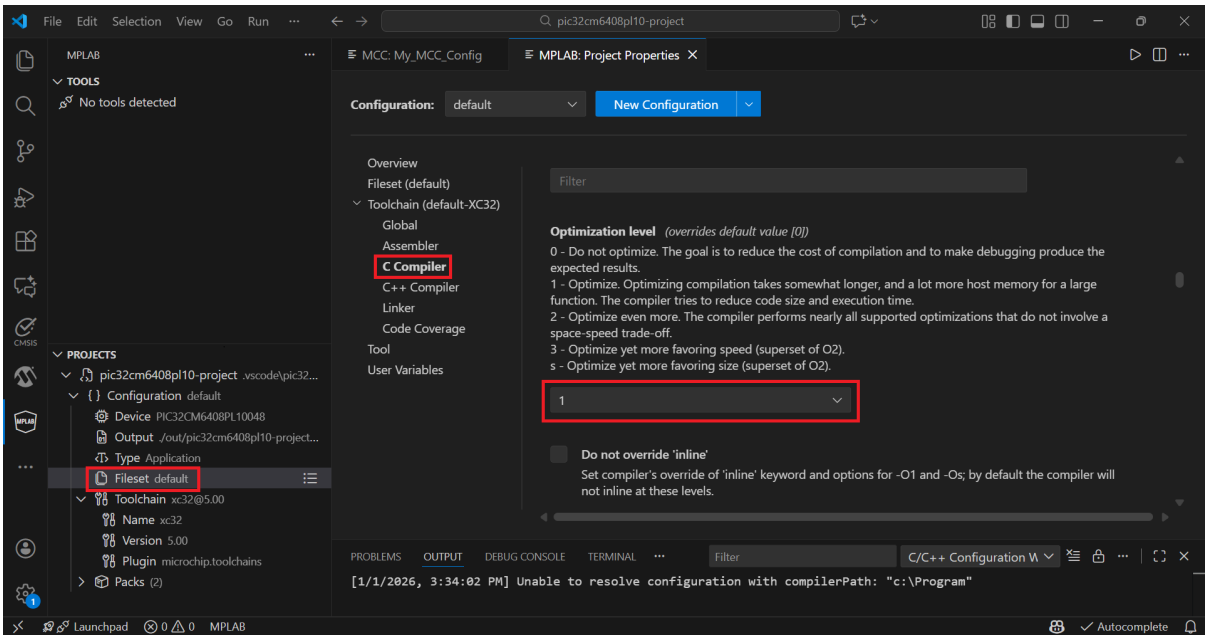
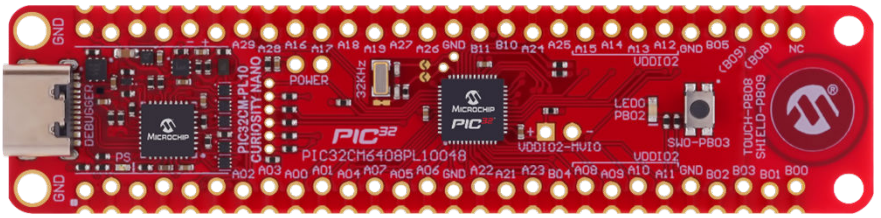
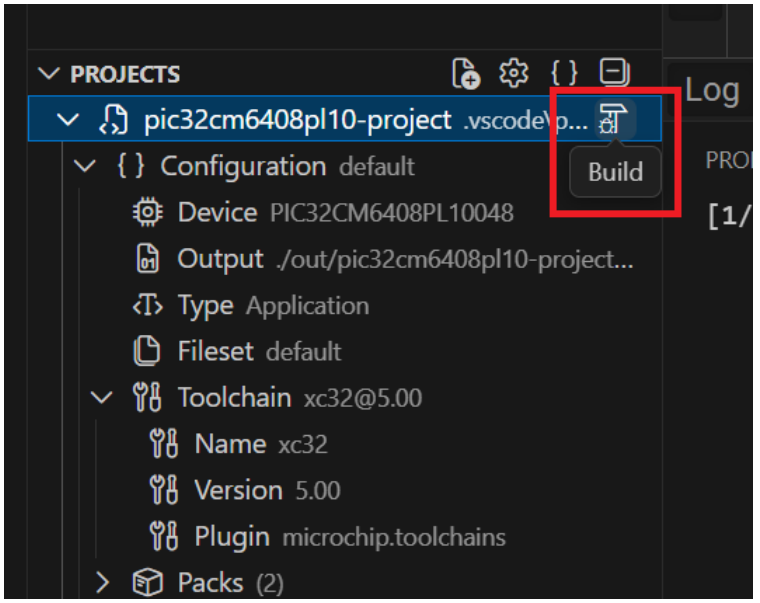


Figure 3-23. Hardware Setup



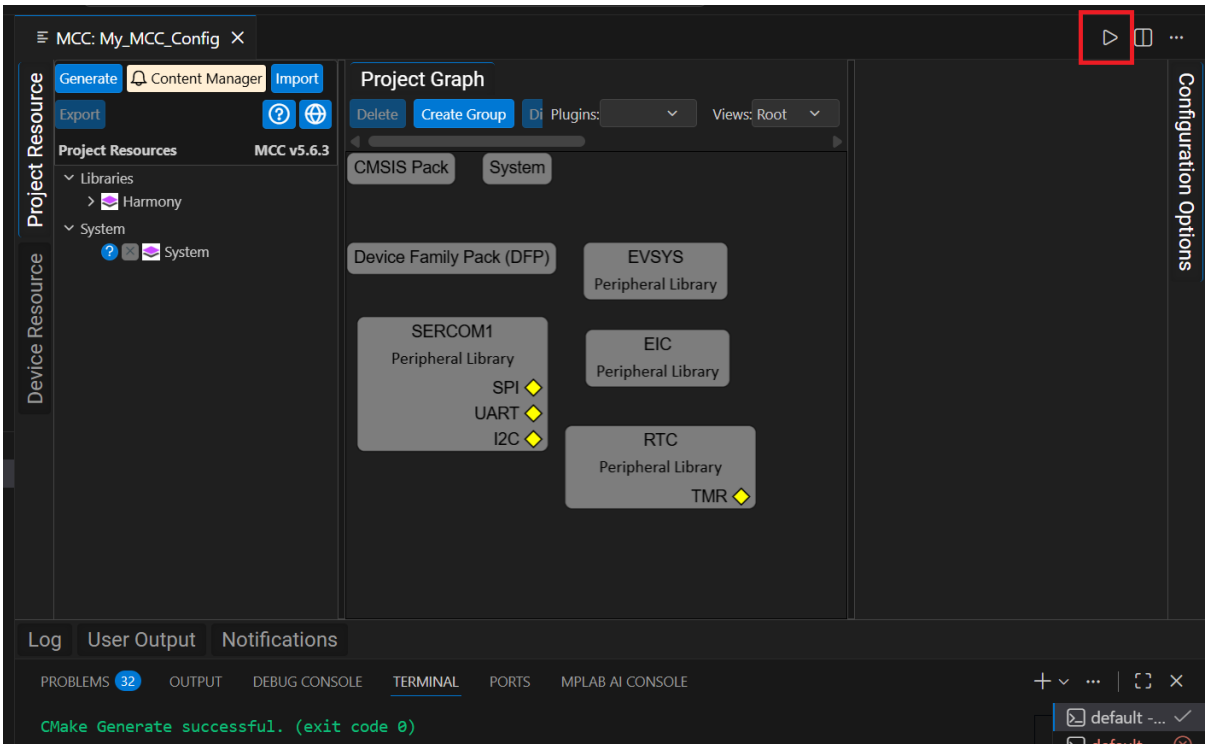
3. Set `pic32cm6408pl10-project` as the main project, and in Project Properties, select the latest compiler version (`v5.00`). Clean and build the project by clicking the highlighted icon.

Figure 3-24. Build Main Project



4. Program the application by clicking the highlighted icon.

Figure 3-25. Program the Application

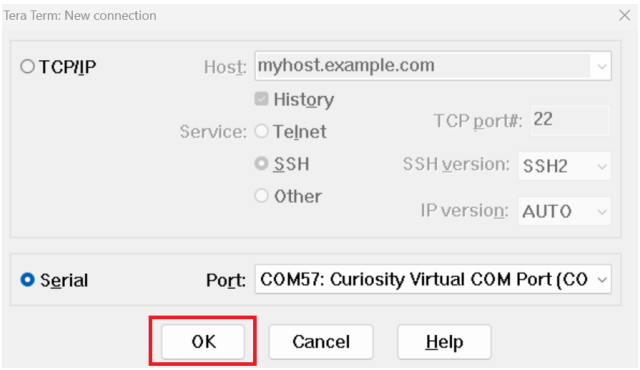


3.6. Observing the Output on the Board and Serial Terminal

To observe the output on the board and serial terminal, follow these steps:

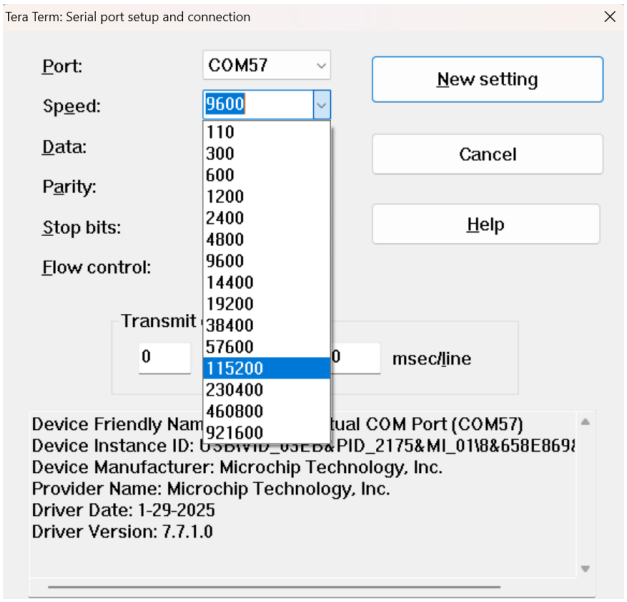
1. Open any terminal window (TeraTerm in this case).
2. Select the required serial port and then click **OK**.

Figure 3-26. Selection of Serial COM Port



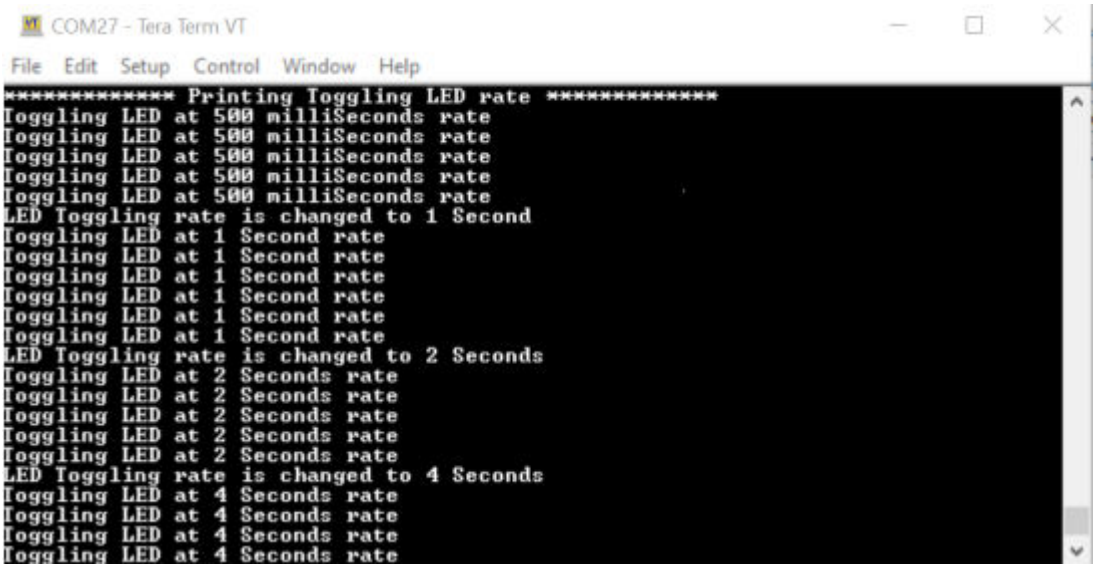
3. In the TeraTerm serial port setup and connection dialog box, type or select 115200 as the baud rate in the Speed box.

Figure 3-27. Setting the Baud Rate



4. An LED on the PIC32CM Curiosity Nano evaluation kit toggles on a timeout basis, with a default periodicity of 500 ms.
5. The LED toggling rate is displayed on the serial terminal.
6. Press the SW switch on the PIC32CM Curiosity Nano evaluation kit to change the default timeout periodicity to 1s.
7. Each subsequent press of the SW switch on the PIC32CM Curiosity Nano evaluation kit changes the periodicity of the timeout periodicity to 2s, 4s, 500 ms, and back to 1s in a cyclic order.

Figure 3-28. Output Window



As the LED toggling rate displayed on the serial terminal changes with each subsequent switch press, observe the same change in the toggling rate of LED0 on the evaluation kit.

4. Getting Started PIC32CM MCU With VS Code and CMSIS Drivers

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

- VS Code
- VS Code Plugins
  - Arm CMSIS Solution
  - Arm CMSIS Debugger
  - Keil Studio Pack
- PIC32CM Curiosity Nano Evaluation Kit

4.1. CMSIS Environment Setup

Refer to this link for a step-by-step guide to installing CMSIS-Toolbox. Use a terminal window to run the commands below.

Follow the steps below to set up the environment for CMSIS based code generation:

1. Install the `cmsis-toolbox` bin folder to the system path under Environment Variables.

```
wget https://artifacts.tools.arm.com/cmsis-toolbox/2.12.0/cmsis-toolbox-windows-amd64.zip
-o cmsis-toolbox-windows-amd64.zip
```

```
tar -xf cmsis-toolbox-windows-amd64.zip
```

2. To install the Device Family Pack (DFP), run the command below on the terminal window.

```
cpackget init https://artifacts.microchip.com/artifactory/pack-index/mdk/index.pidx
```

3. Verify that the installed DFP pack is the latest public released version by running the command.

```
cpackget add Microchip::PIC32CM-PL_DFP
```

4. For the Generic Software Pack (GSP), initialize `cpackget` with the Microchip pack index.

```
cpackget init https://artifacts.microchip.com/artifactory/pack-index/mdk/index.pidx
```

5. Update to the latest Microchip CMSIS Driver Pack using the command below.

```
cpackget add Microchip::CMSIS-Driver_PIC32CM-PL
```

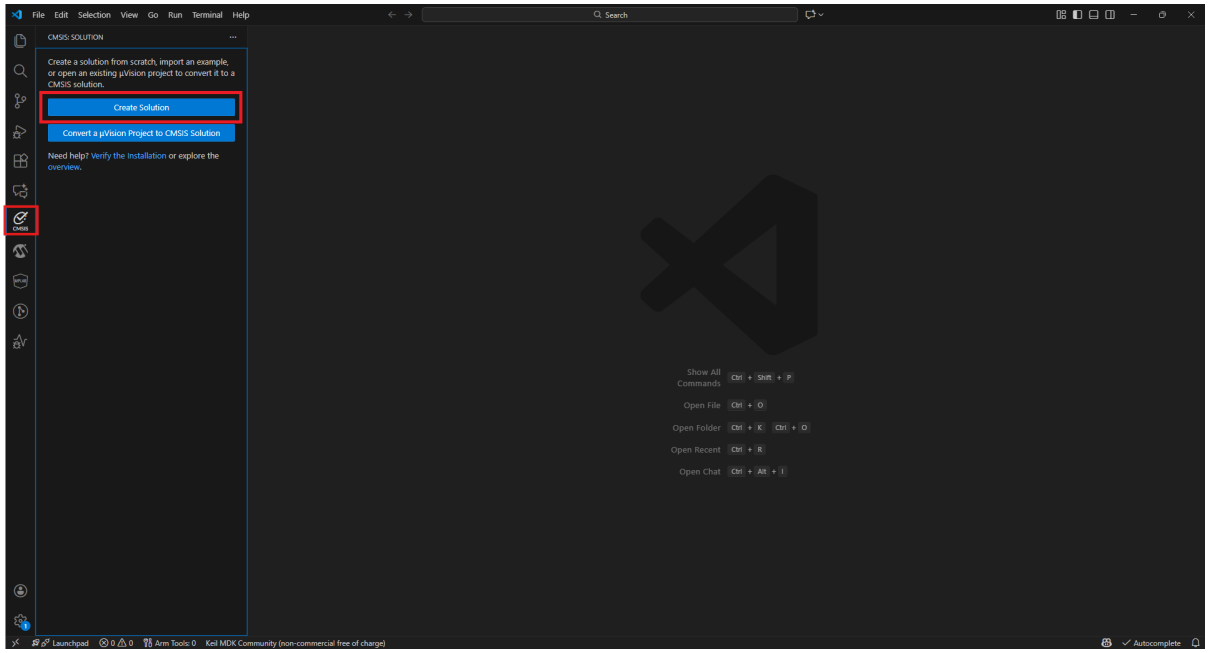
6. To apply the changes, restart the terminal window.

4.2. Creating the First Application on the PIC32CM MCU

To create a CMSIS based project, follow these steps:

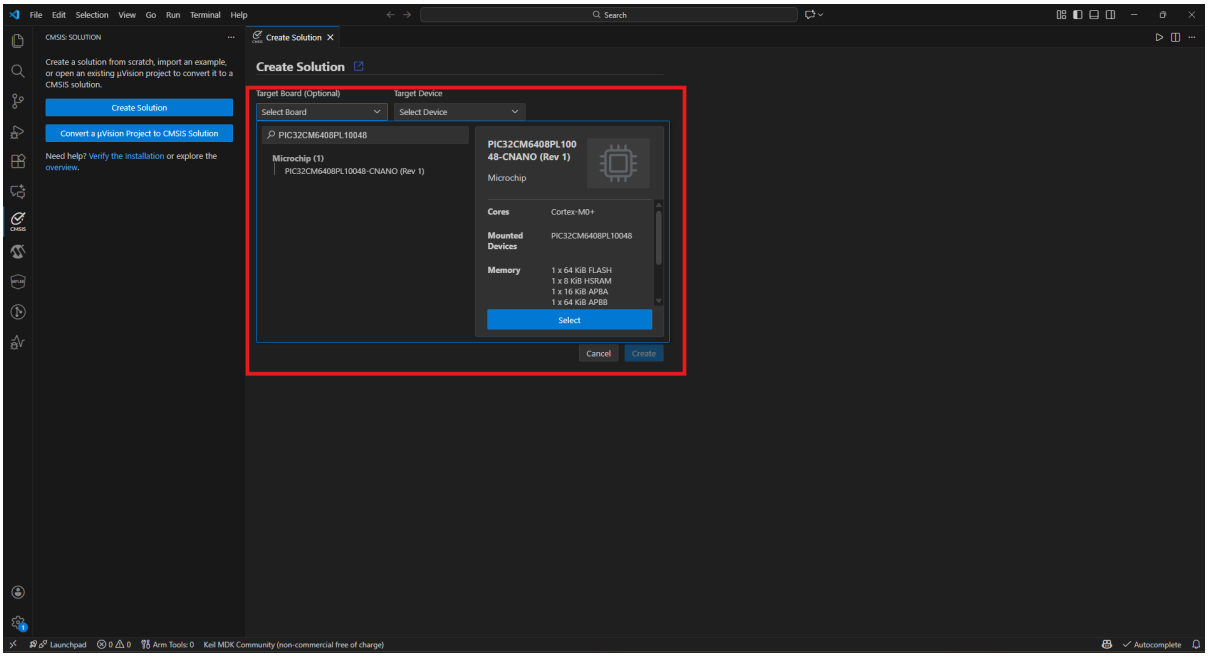
1. On the Start menu, launch VS Code.
2. On the left navigation bar, click the CMSIS plugin and select **Create Solution**.

Figure 4-1. Navigate to CMSIS and Solution Creation



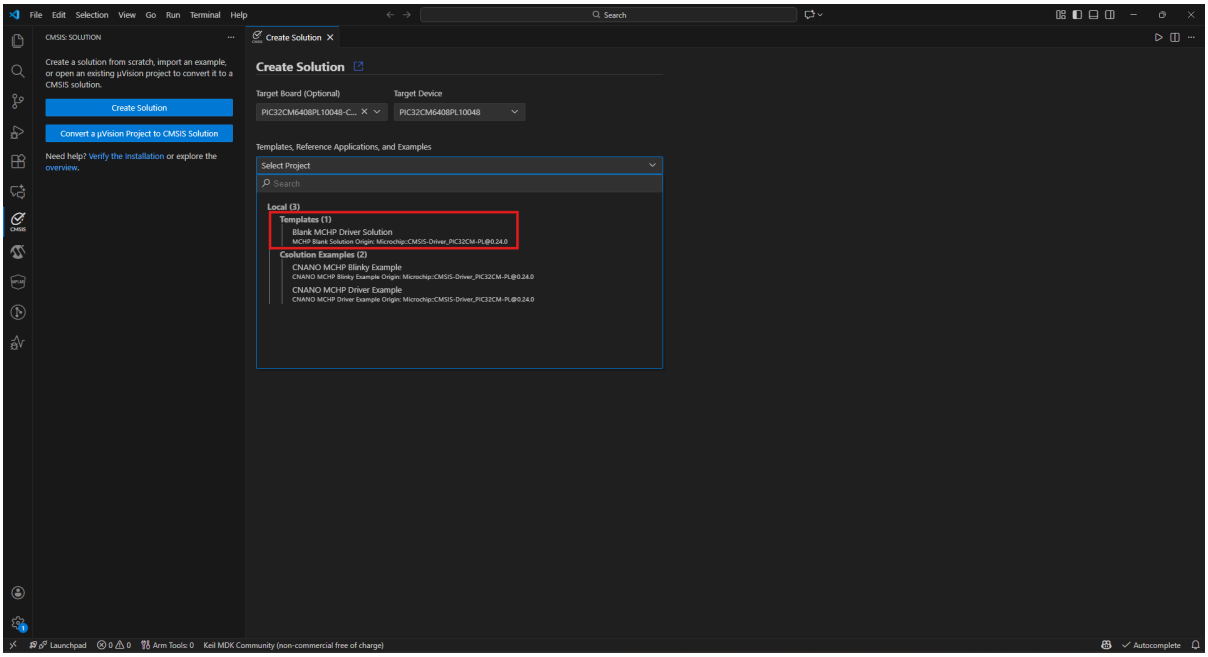
3. Select the target board as PIC32CM6408PL10048 and press **Select**.

Figure 4-2. Device Selection



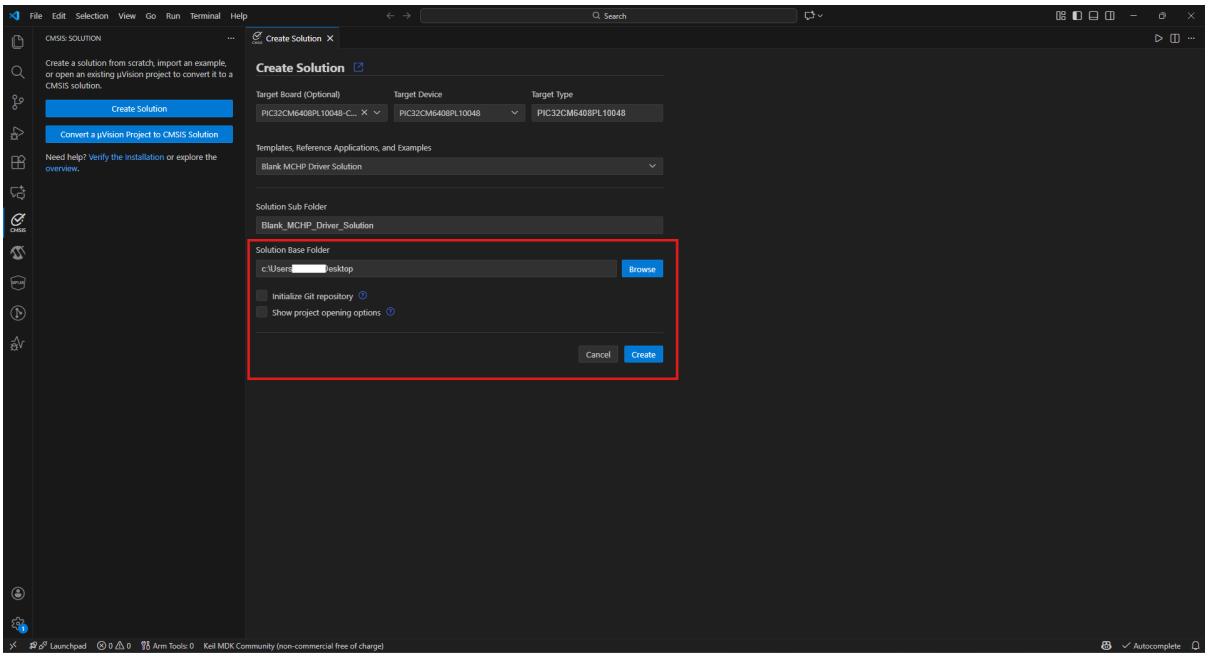
4. Select the Blank MCHP Driver Solution template.

Figure 4-3. Template Selection



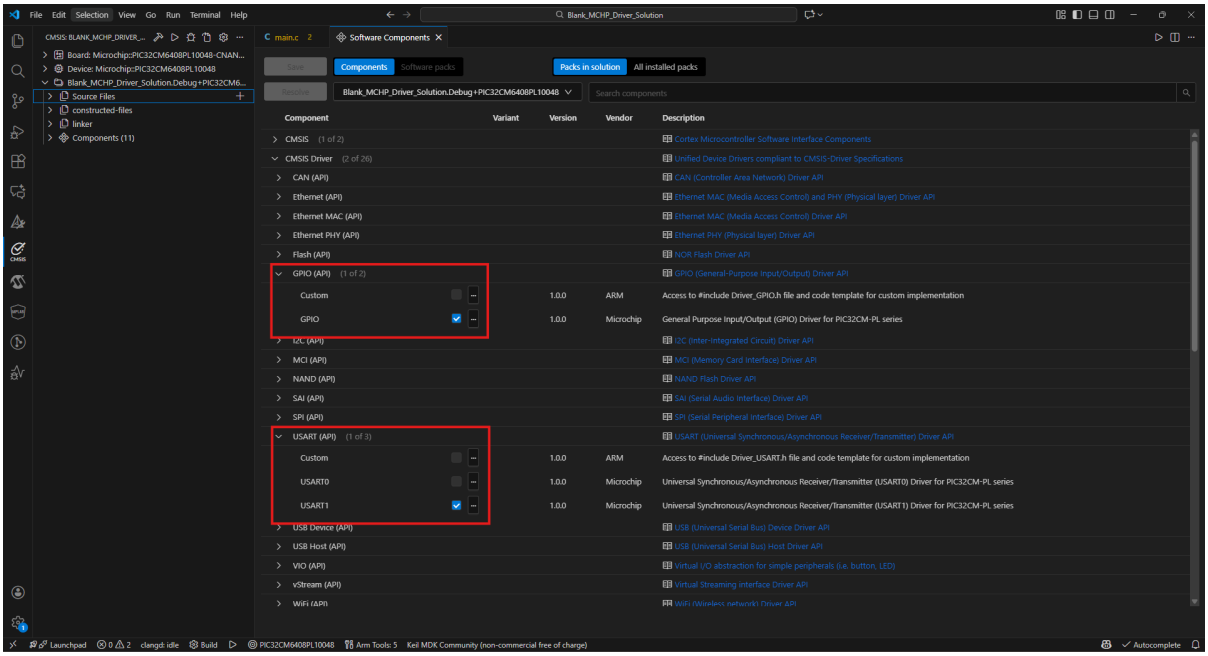
5. Select the project location and click **Create**.

Figure 4-4. Project Location Selection



6. Under the Components tab, select the GPIO and USART drivers from the CMSIS Driver drop-down. Select the dependent plibs from the MCHP-PLIB drop-down option (PORT and SERCOM_USART1), along with GCLK and MCLK, and click **Save**.

Figure 4-5. CMSIS Driver and MCHP PLIB Selection



4.3. Adding Application Logic

To develop and run the application, use the following steps:

1. Open the `main.c` file of the project and add the following application code.

```
while (1)
{
    /* -------- Switch Handling -------- */

    curr_sw_state = GPIOdrv->GetInput(SW0);

    if ((prev_sw_state == 1) && (curr_sw_state == 0))
    {
        blink_index = (blink_index + 1) % BLINK_TABLE_SIZE;
        blink_delay = blink_table[blink_index];

        USARTdrv->Send(switch_msg, strlen((char *)switch_msg));

        delay_ms(DEBOUNCE_TIME_MS);    // Mechanical debounce
    }

    prev_sw_state = curr_sw_state;

    /* -------- LED Blink Timing -------- */

    if ((g_ms_ticks - last_blink_time) >= blink_delay)
    {
        last_blink_time = g_ms_ticks;

        led_state ^= 1U;
        GPIOdrv->SetOutput(LED0, led_state);

        snprintf(speed_msg, sizeof(speed_msg),"Blink Speed: %u ms\r\n", blink_delay);

        USARTdrv->Send(speed_msg, strlen(speed_msg));
    }
}
```

2. Add peripheral initialization function calls and variable declarations inside `main.c`, before the `while(1)` loop.

```
/******** System Initialization ********/
    configure_clock();
    configure_systick();
```

```
    /******** GPIO Configuration ********/
    // LED Output
    GPIOdrv->Setup(LED0, NULL);
    GPIOdrv->SetDirection(LED0, ARM_GPIO_OUTPUT);
    GPIOdrv->SetOutputMode(LED0, ARM_GPIO_PUSH_PULL);

    // Switch Input (Active LOW)
    GPIOdrv->Setup(SW0, NULL);
    GPIOdrv->SetDirection(SW0, ARM_GPIO_INPUT);
    GPIOdrv->SetPullResistor(SW0, ARM_GPIO_PULL_UP);

    /******** Interrupt Enable ********/

    __DMB();
    __enable_irq();
    NVIC_EnableIRQ(SERCOM1_IRQn);

    /******** USART Configuration ********/

    PORT_SetPeripheralMuxing(PIN_PB00, PORT_PERIPHERAL_FUNC_D); // TX
    PORT_SetPeripheralMuxing(PIN_PB01, PORT_PERIPHERAL_FUNC_D); // RX

    USARTdrv->Initialize(NULL);
    USARTdrv->PowerControl(ARM_POWER_FULL);

    USARTdrv->Control(
        ARM_USART_MODE_ASYNCHRONOUS |
        ARM_USART_PARITY_NONE |
        ARM_USART_STOP_BITS_1 |
        ARM_USART_DATA_BITS_8,
        USART1_DEFAULT_BAUD_RATE);

    USARTdrv->Control(ARM_USART_CONTROL_TX, 1);

    /******** Application Variables ********/

    uint8_t blink_index = 0;
    uint32_t blink_delay = blink_table[0];

    uint8_t prev_sw_state = 1;
    uint8_t curr_sw_state;

    uint32_t led_state = 0;
    uint32_t last_blink_time = 0;
```

3. Add header files, hardware pin definitions, global variables, macros, and driver declarations at the start of the code.

```
/*******************************************************************************
 * Includes
 ******************************************************************************/
#include "RTE_Components.h"
#include CMSIS_device_header

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/*******************************************************************************
 * Hardware Pin Definitions
 ******************************************************************************/
#define LED0    PIN_PB02      // Status LED
#define SW0     PIN_PB03      // User switch

/*******************************************************************************
 * Application Configuration Macros
 ******************************************************************************/
#define DEBOUNCE_TIME_MS    20U
#define BLINK_TABLE_SIZE   4U

/*******************************************************************************
 * CMSIS Driver Declarations
 ******************************************************************************/
extern ARM_DRIVER_GPIO Driver_GPIO;
static ARM_DRIVER_GPIO *GPIOdrv = &Driver_GPIO;
```

```
extern ARM_DRIVER_USART Driver_USART1;
static ARM_DRIVER_USART *USARTdrv = &Driver_USART1;

/*******************************************************************************
 * Global Variables
 ******************************************************************************/
static const uint32_t blink_table[BLINK_TABLE_SIZE] = {500, 1000, 2000, 4000};
static uint8_t switch_msg[] = "Switch Press Detected\r\n";
static char speed_msg[50];
volatile uint32_t g_ms_ticks = 0;    // System millisecond tick counter
```

4. Add function prototypes and handlers below the global variables.

```
/*******************************************************************************
 * Function Prototypes
 ******************************************************************************/
void configure_clock(void);
void configure_systick(void);
void delay_ms(uint32_t ms);

/*******************************************************************************
 * SysTick Interrupt Handler
 * Increments 1ms system tick counter
 ******************************************************************************/
void SysTick_Handler(void)
{
    g_ms_ticks++;
}
```

5. Add function definitions below main function.

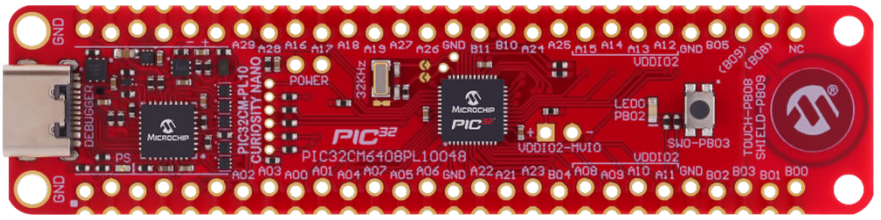
```
/*******************************************************************************
 * Clock Configuration
 ******************************************************************************/
void configure_clock(void)
{
    OSCCTRL_OSCHF_EnableAutotune();
    OSCCTRL_OSCHF_SetFrequency(OSCCTRL_OSCHF_FREQ_4M);

    // SERCOM1 clock routing
    GCLK_SetPeripheralChannelGenSrc(GCLK_PCHCCTRL_8, GCLK_GENERATOR_0);
    GCLK_EnablePeripheralChannel(GCLK_PCHCCTRL_8);

    MCLK_EnableAPBCClock(MCLK_APBC_SERCOM1);
}

/*******************************************************************************
 * SysTick Configuration (1ms time base)
 ******************************************************************************/
void configure_systick(void)
{
    uint32_t core_clock_hz = OSCCTRL_OSCHF_GetFrequency();

    SysTick->LOAD = (uint32_t)((core_clock_hz / 1000U) - 1U);
    SysTick->VAL  = 0;

    SysTick->CTRL = SysTick_CTRL_CLKSOURCE_Msk |
                    SysTick_CTRL_TICKINT_Msk |
                    SysTick_CTRL_ENABLE_Msk;
}

/*******************************************************************************
 * Blocking Delay Function
 * Used only for short debounce timing
 ******************************************************************************/
void delay_ms(uint32_t ms)
{
    uint32_t start = g_ms_ticks;

    while ((g_ms_ticks - start) < ms)
    {
        ;   // wait
    }
}
```

4.4. Building and Programming Application

To build and program the application, follow these steps:

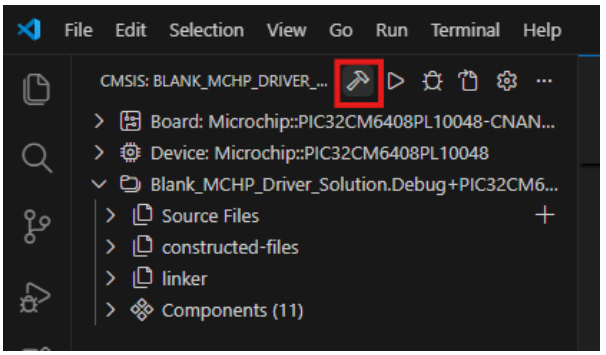
1. The PIC32CM Curiosity Nano evaluation kit supports debugging using a debugger. Connect the USB Type-C cable to power and debug the PIC32CM Curiosity Nano evaluation kit.

Figure 4-6. Hardware Setup



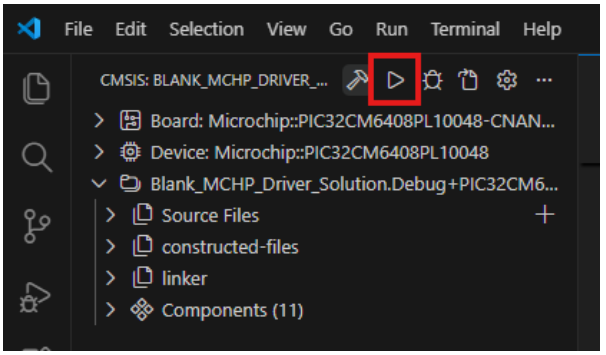
2. Build the project by clicking the highlighted icon.

Figure 4-7. Build Solution



3. Program the application by clicking the highlighted icon.

Figure 4-8. Program Device

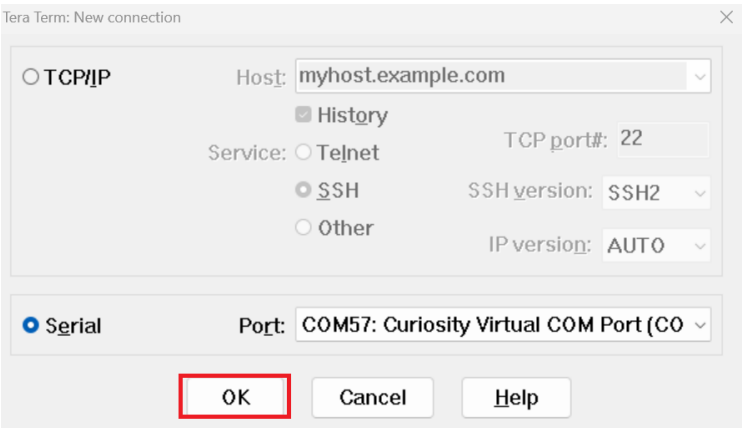


4.5. Observing the Output on the Board and Serial Terminal

To observe the output on the board and serial terminal, follow these steps:

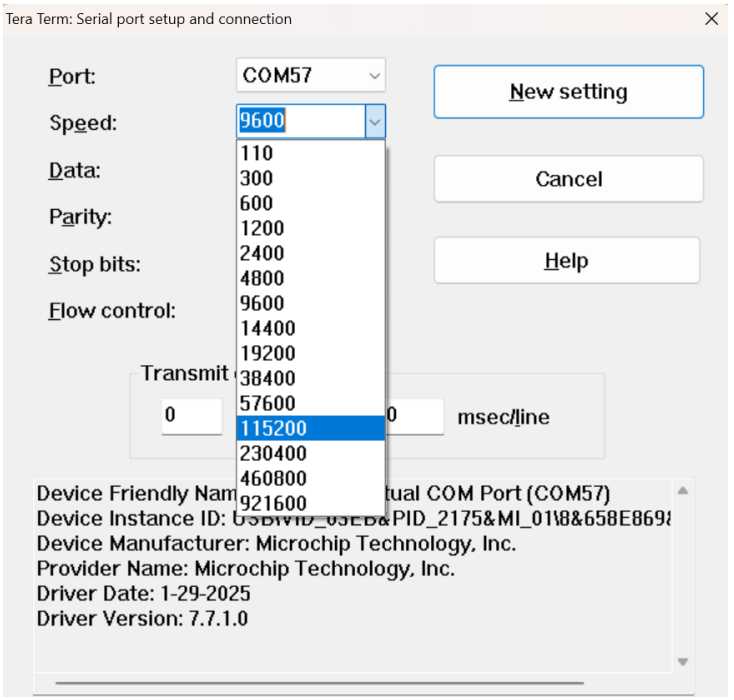
1. Press the **Start** button to open any terminal window (TeraTerm in this case).
2. Select the required serial port and then click **OK**.

Figure 4-9. Selection of Serial COM Port



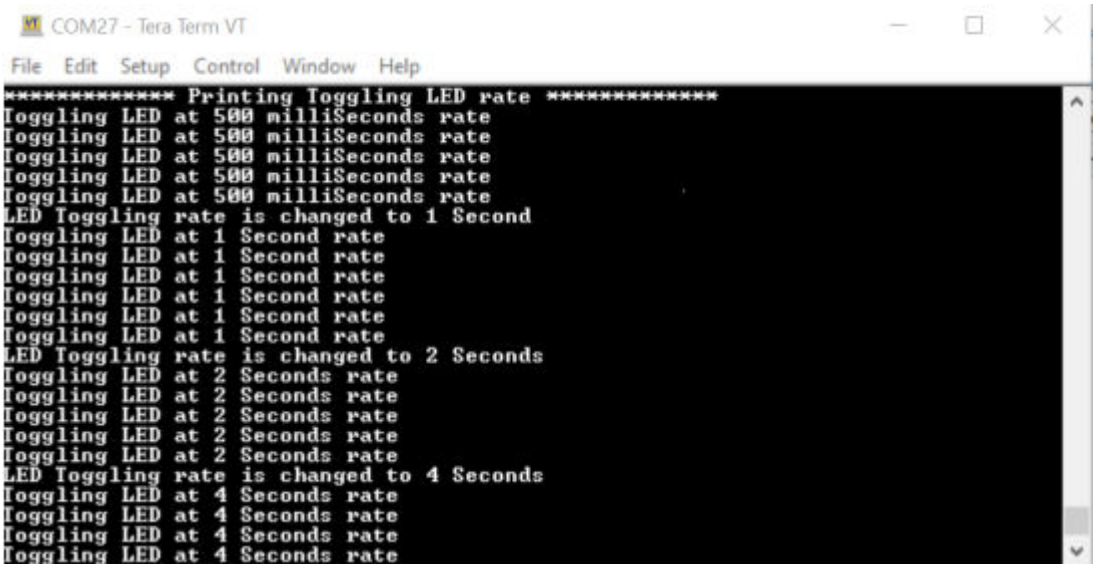
3. In the TeraTerm serial port setup and connection dialog box, type or select 115200 as the baud rate in the Speed box.

Figure 4-10. Setting the Baud Rate



4. An LED on the PIC32CM Curiosity Nano evaluation kit toggles on a timeout basis, with a default periodicity of 500 ms.
5. The LED toggling rate is displayed on the Serial Terminal.
6. Press the SW switch on the PIC32CM Curiosity Nano evaluation kit to change the default timeout periodicity to 1s.
7. Each subsequent press of the SW switch on the PIC32CM Curiosity Nano evaluation kit changes the timeout periodicity to 2s, 4s, 500 ms, and back to 1s in a cyclic order.

Figure 4-11. Output Window



As the LED toggling rate displayed on the serial terminal changes with each subsequent switch press, observe the same change in the toggling rate of LED0 on the evaluation kit.

5. Getting Started PIC32CM MCU With MPLAB X IDE

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

- MPLAB X IDE v6.25 or newer
- MPLAB Code Configurator Plug-in v5.6.3 or newer
- MPLAB XC32 Compiler v5.00 or newer
- MPLAB Harmony v3 repository: csp v3.25 or newer
- MCC Core Version v5.8.3 or newer
- PIC32CM Curiosity Nano Evaluation Kit

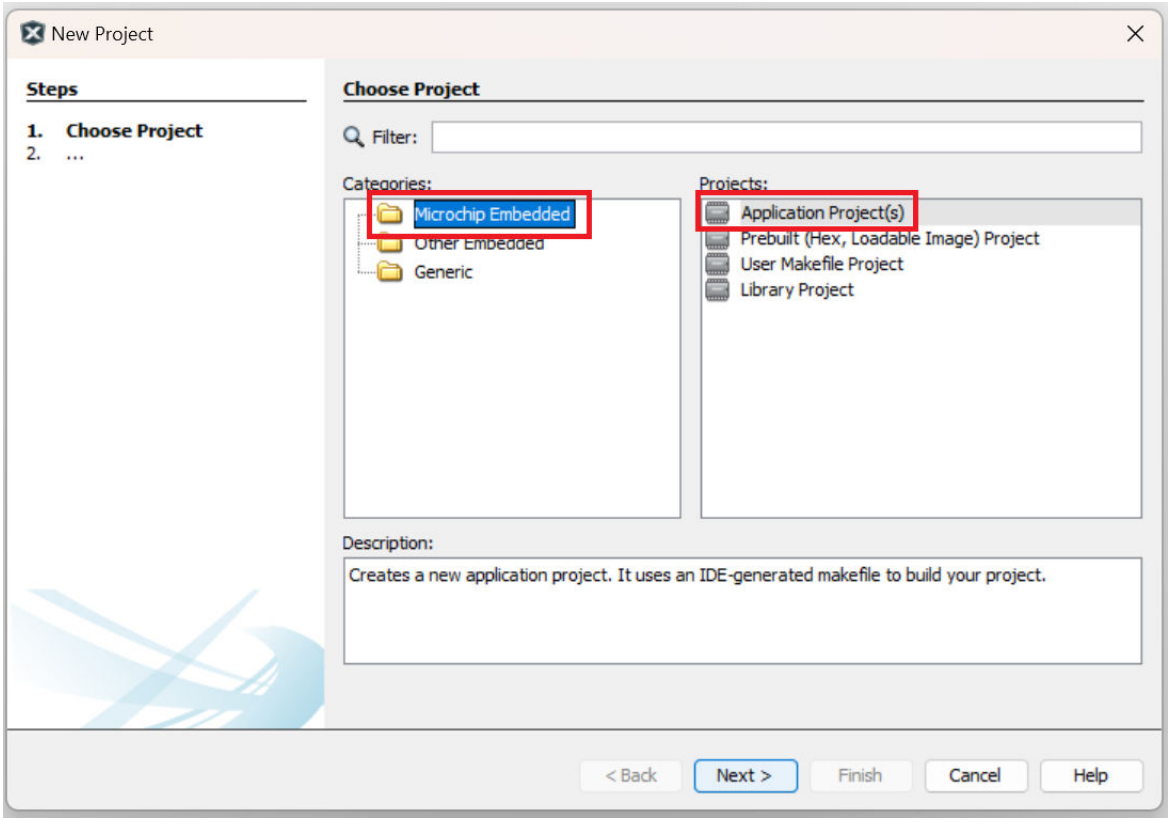
Note: The latest versions of these tools can also be used to develop the application.

5.1. Creating the First Application on the PIC32CM MCU

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

1. On the Start menu, launch MPLAB X IDE.
2. On the File menu, click New Project or click the New Project icon.
3. In the New Project window, on the left navigation bar, under Steps, click Choose Project.

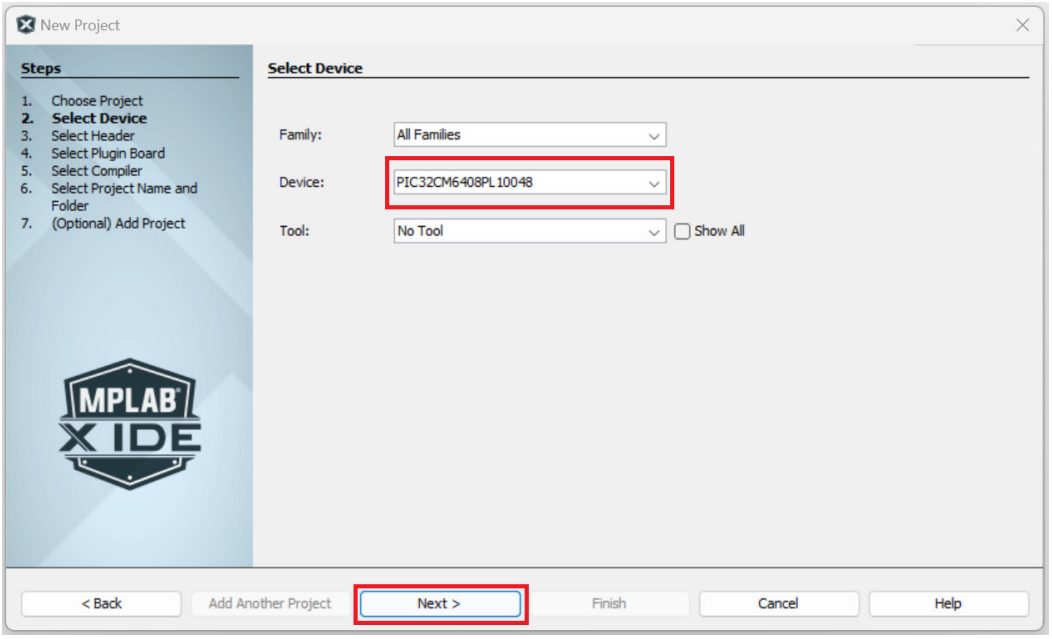
Figure 5-1. Choose Project Window



4. In the Choose Project property page:
   a. Categories: Select Microchip Embedded.
   b. Project: Select Application Project(s).
5. Click **Next**.

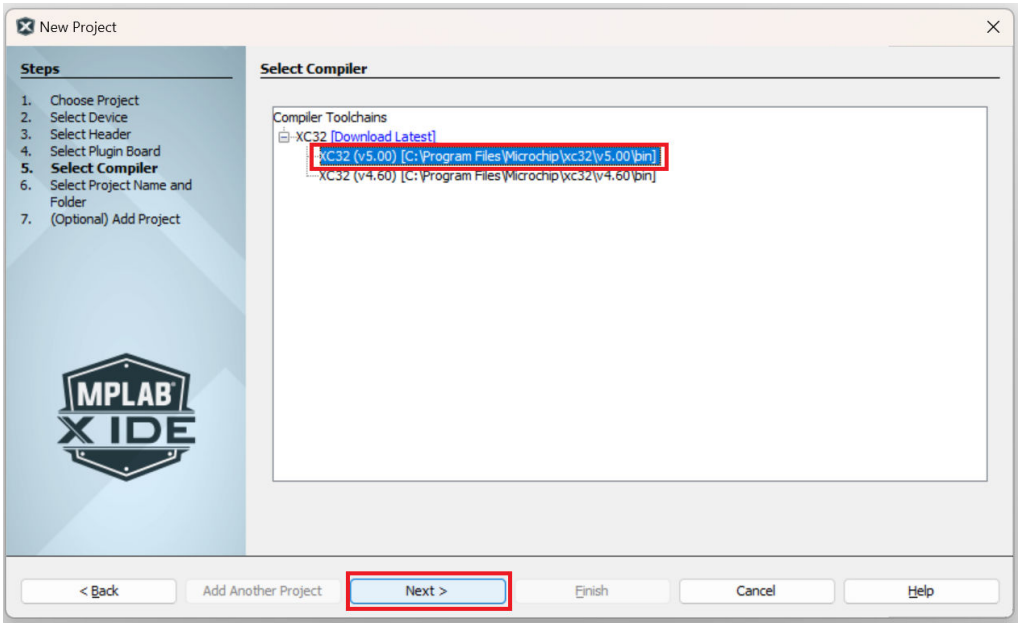
6. In the left navigation bar, under Steps, click Select Device.

Figure 5-2. Device Selection Window



7. In the Select Device property page, type or select the device (e.g., PIC32CM6408PL10048) in the Device box.
8. Click **Next**.
9. In the left navigation bar, under Steps, click Select Compiler.
10. In the Select Compiler property page, expand the XC32 list of options, and then select the compiler (e.g., XC32 (v5.00)).

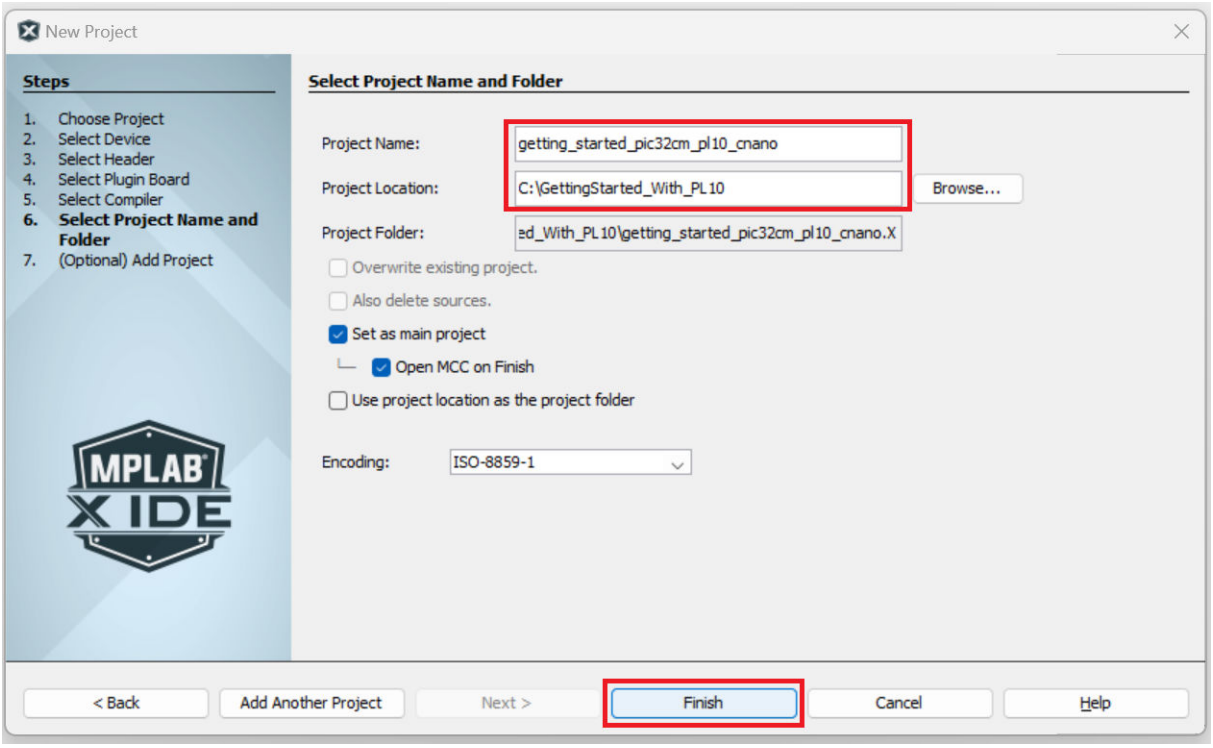
Figure 5-3. Compiler Selection Window



11. Click **Next**.

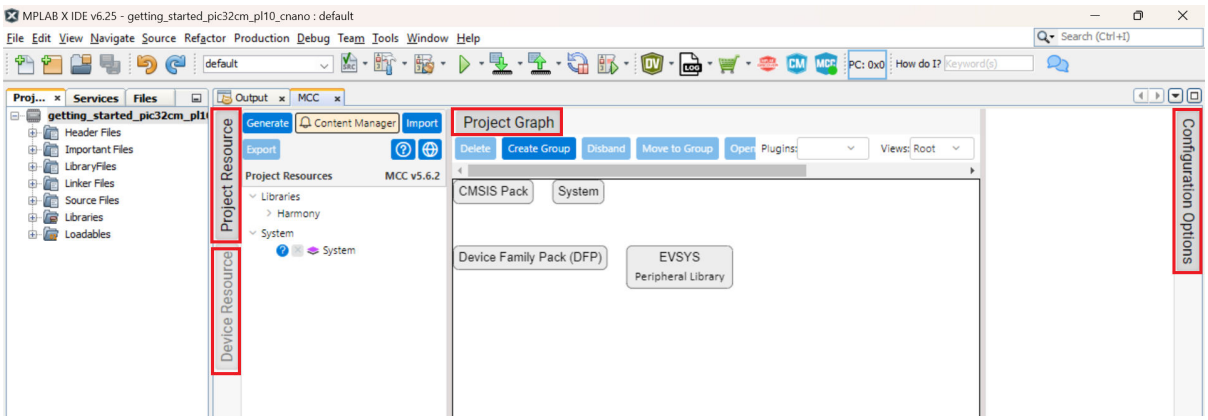
12. In the left navigation bar, click Select Project Name and Folder.
13. In the Select Project Name and Folder property page:
    a. Project Name: Enter `getting_started_pic32cm_pl10_cnano`.
    b. Project Location: Click the **Browse** button and choose `C:\GettingStarted_With_PL10`.

Figure 5-4. Project Name and Folder Selection Window



14. Click **Finish**.
15. Before launching MCC, the Configuration Database Setup window will be displayed, where the Device Family Pack (DFP) and Cortex Microcontroller Software Interface Standard (CMSIS) path can be changed if required. For this demonstration, the default settings are used.
16. The MCC plugin will open in a new window, as shown in the following figure.

Figure 5-5. MPLAB Code Configurator Window

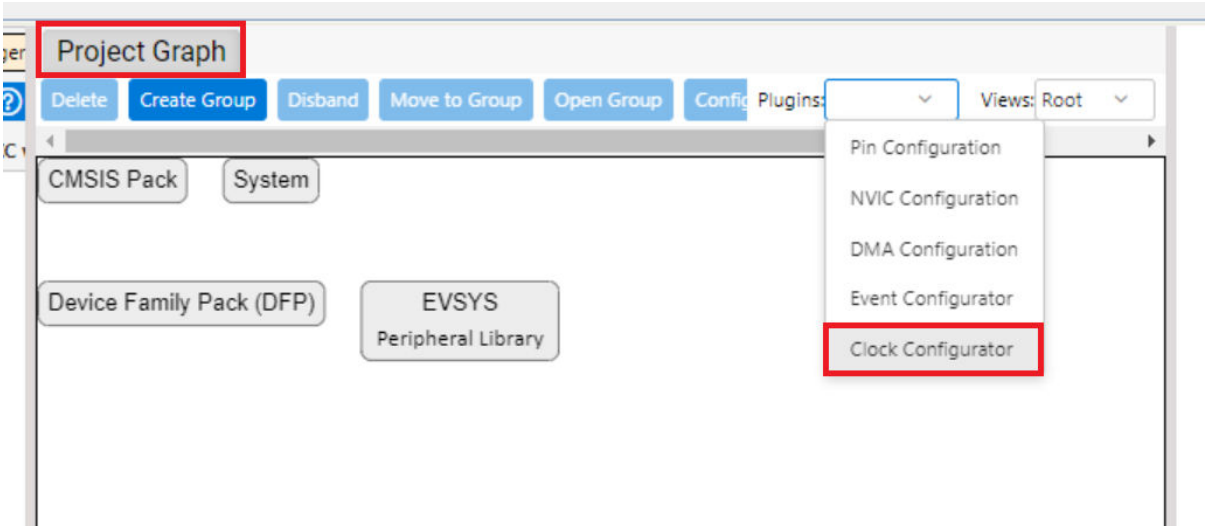


5.2. Adding and Configuring the MPLAB Harmony Component

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

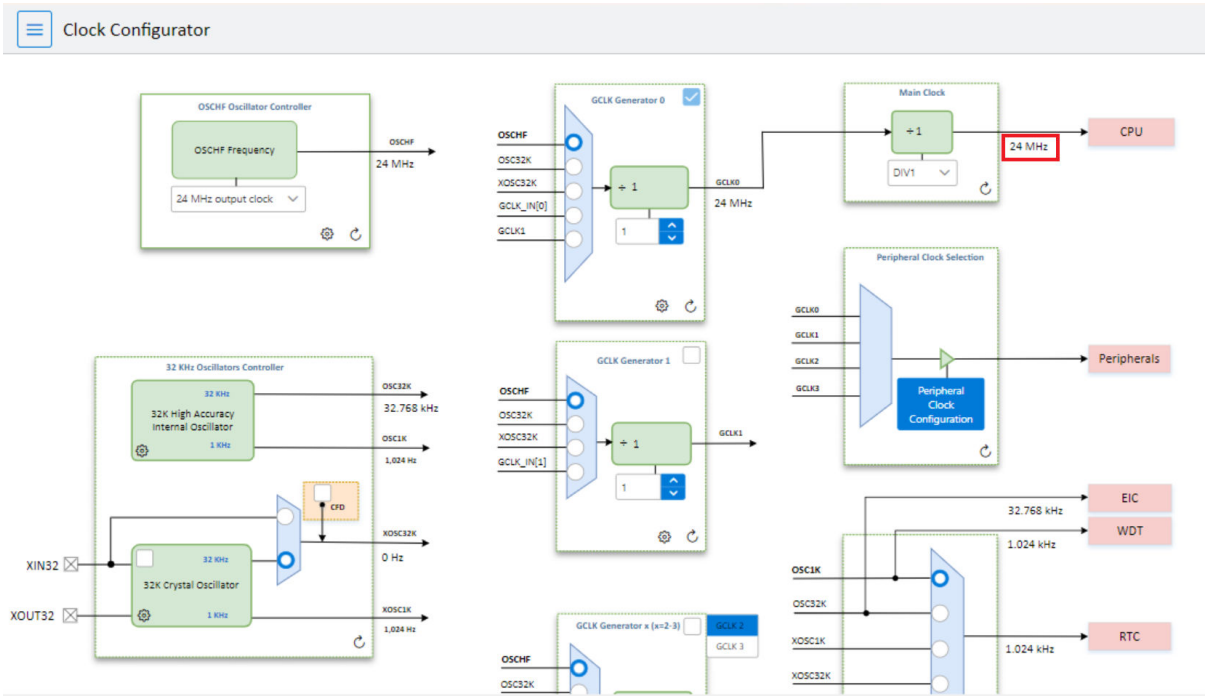
1. In the MCC window, click Project Graph.

Figure 5-6. MPLAB Code Configurator



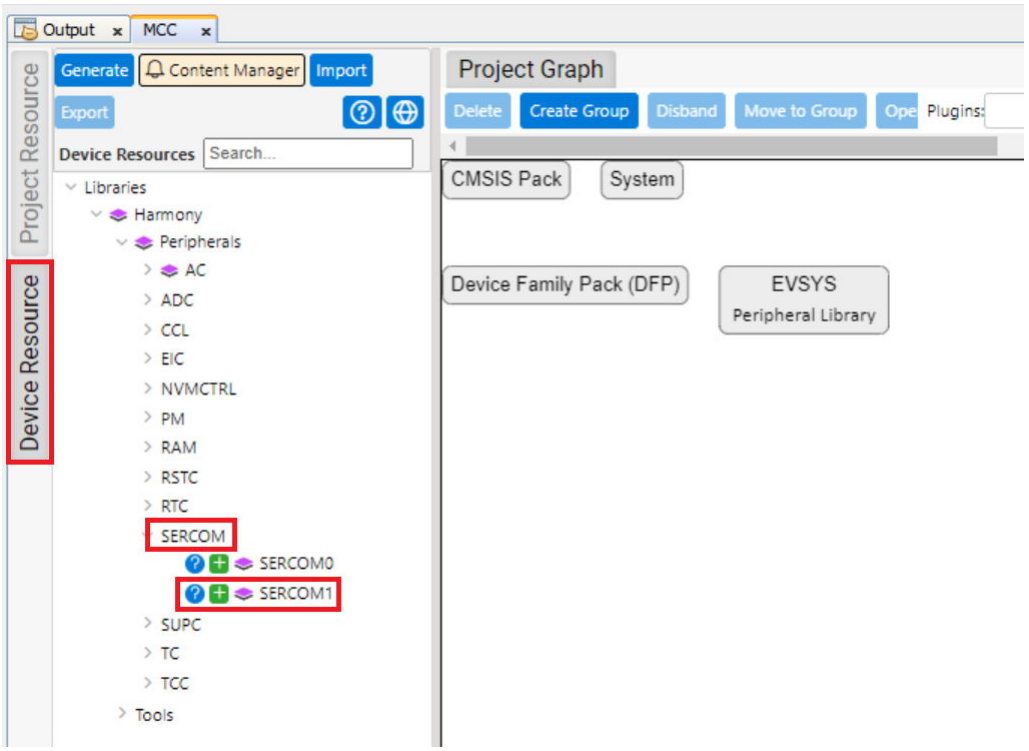
2. In the Plugins drop-down list, select Clock Configuration. The Clock Easy View window will be displayed, verify that the Main Clock is set to 24 MHz.
Note: Ensure that the following modification is made for GCLK Generator 0.

Figure 5-7. MPLAB Code Configurator - GCLK Generator 0



3. Under Device Resources, expand the list of options: *Harmony>Peripherals>SERCOM*.
4. Click SERCOM and observe that the SERCOM1 Peripheral Library block is added to the Project Graph window.

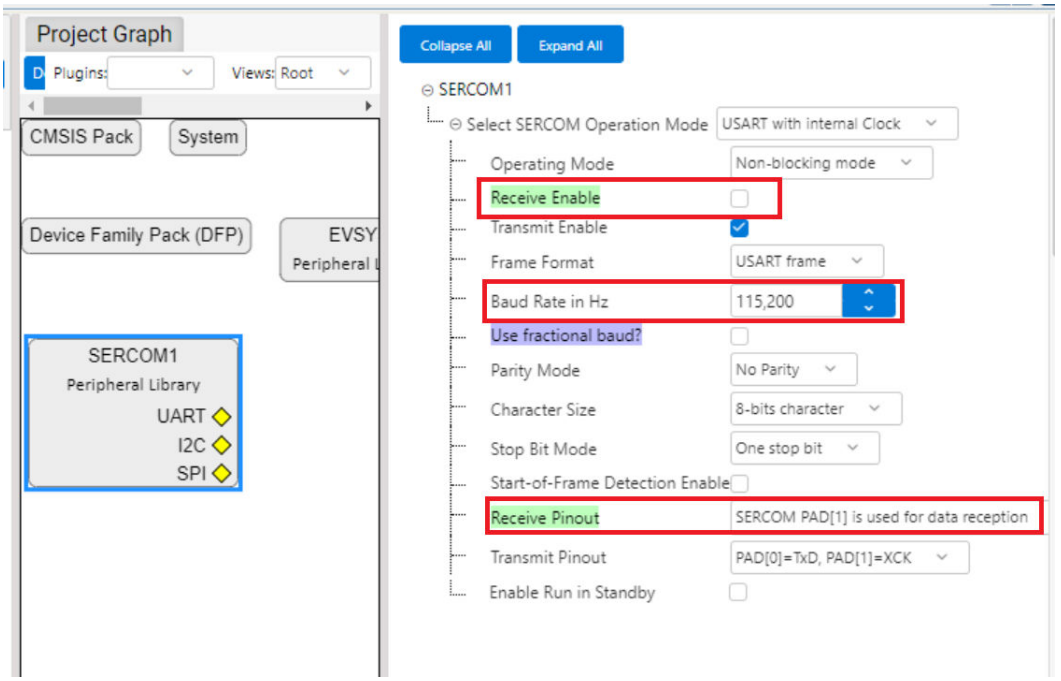
Figure 5-8. MPLAB Code Configurator - Selection of Peripheral



Note: Users can also select other peripherals under Device Resources: *Harmony>Peripherals*.

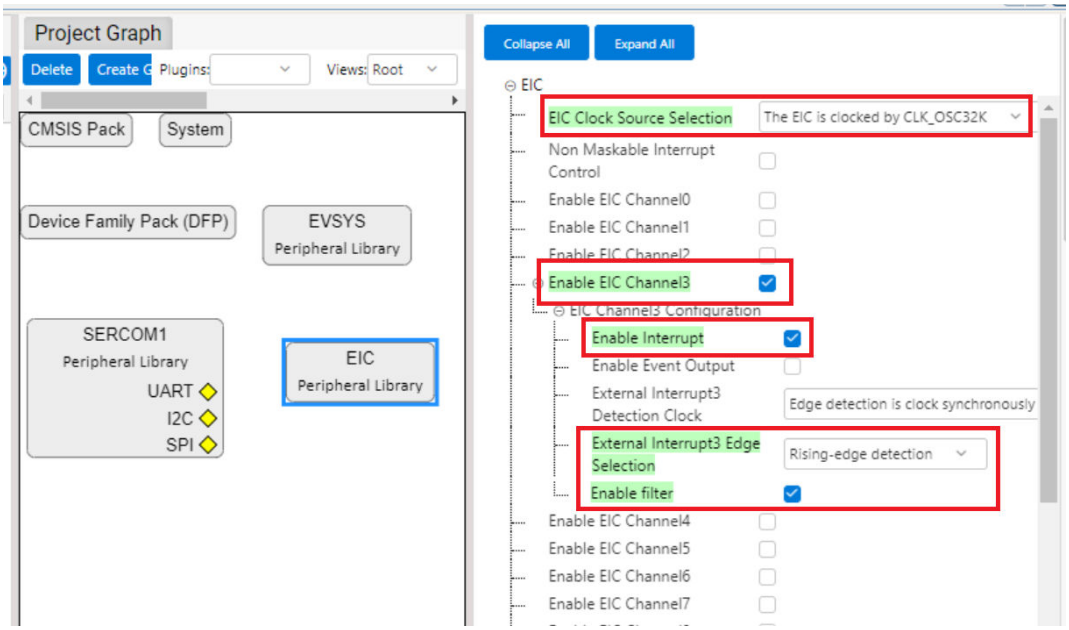
5. In the Project Graph window, on the left navigation bar, select the SERCOM1 Peripheral Library. In the Configuration Options property page on the right, configure it as follows to print data on the serial console at a baud rate of 115,200.

Figure 5-9. MPLAB Code Configurator – SERCOM1 Configuration



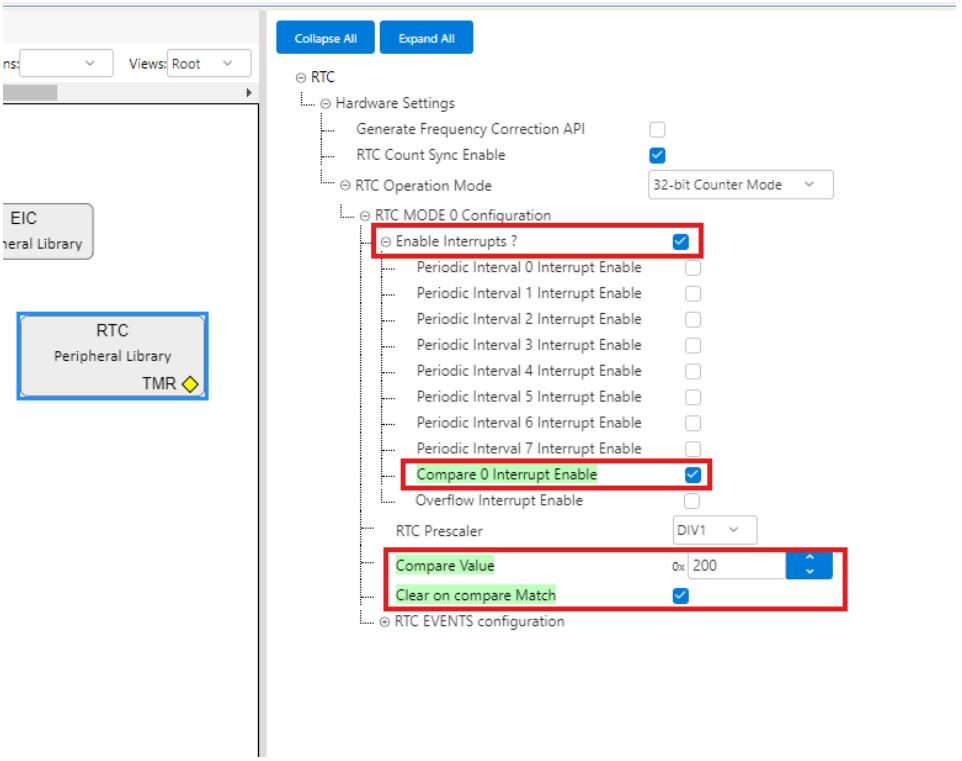
6. Under Device Resources, expand the list of options: *Harmony>Peripherals>EIC*. Click EIC and observe that the EIC Peripheral Library block is added to the Project Graph window.
7. In the Project Graph window, on the left navigation bar, select the EIC Peripheral Library. In the Configuration Options property page on the right, select the Enable Interrupt checkbox and the Enable EIC Channel 3 checkbox for the switch press event.

Figure 5-10. MPLAB Code Configurator – EIC Configuration



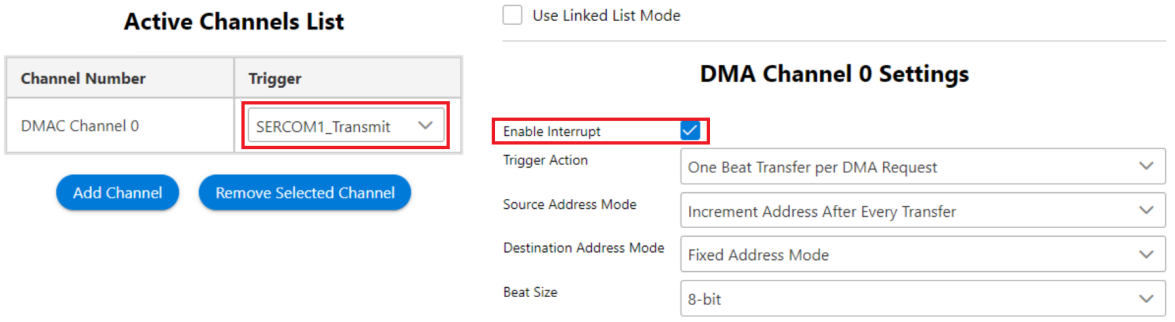
8. Under Device Resources, expand the list of options: *Harmony>Peripherals>RTC*. Click RTC and observe that the RTC Peripheral Library block is added to the Project Graph window to generate a compare interrupt every 500 ms. Select the Clear on compare Match checkbox.
   Note: The compare value is set to 0x200. This value generates an RTC compare interrupt every 500 ms.
   RTC Clock = 1024 Hz
   RTC Prescaler = 1
   Required Interrupt rate = 500 ms
   Therefore, the compare value = (500/1000) x 1024 = 512 (i.e., 0x200).

Figure 5-11. MPLAB Code Configurator – RTC Configuration



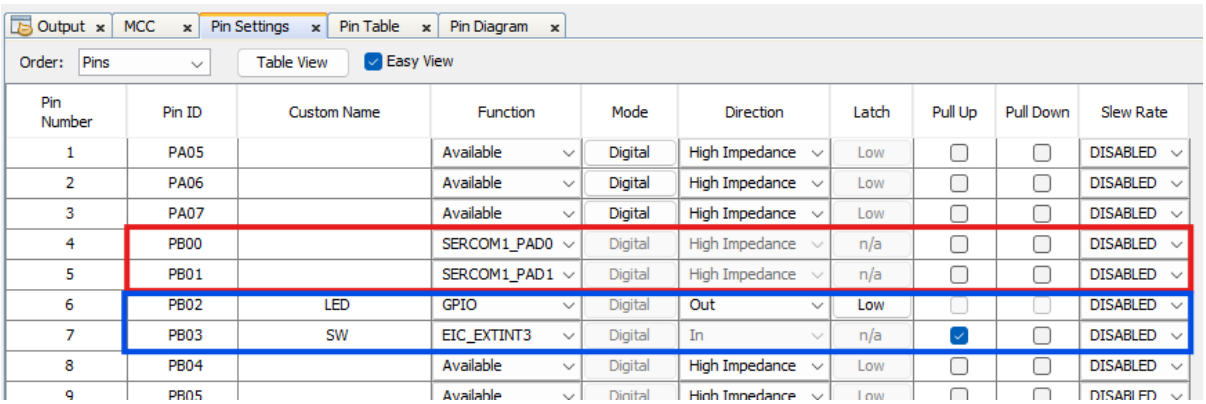
9. From the Plugins drop-down list, select Add Channel and then select DMA Configuration. Configure DMA Channel 0 to transmit the application buffer to the USART TX register. The DMA transfers one byte from the user buffer to the USART transmit buffer on each trigger.

Figure 5-12. MPLAB Code Configurator – DMA Configuration



10. From the Plugins drop-down list, select Pin Configuration, and then click **Pin Settings**.
11. In the Order box, type or select Ports. Build configurations according to the application as indicated below. Change the Custom Name of the pin IDs PB02 and PB03, as shown in the following figure.

Figure 5-13. MPLAB Code Configurator – Pin Configuration

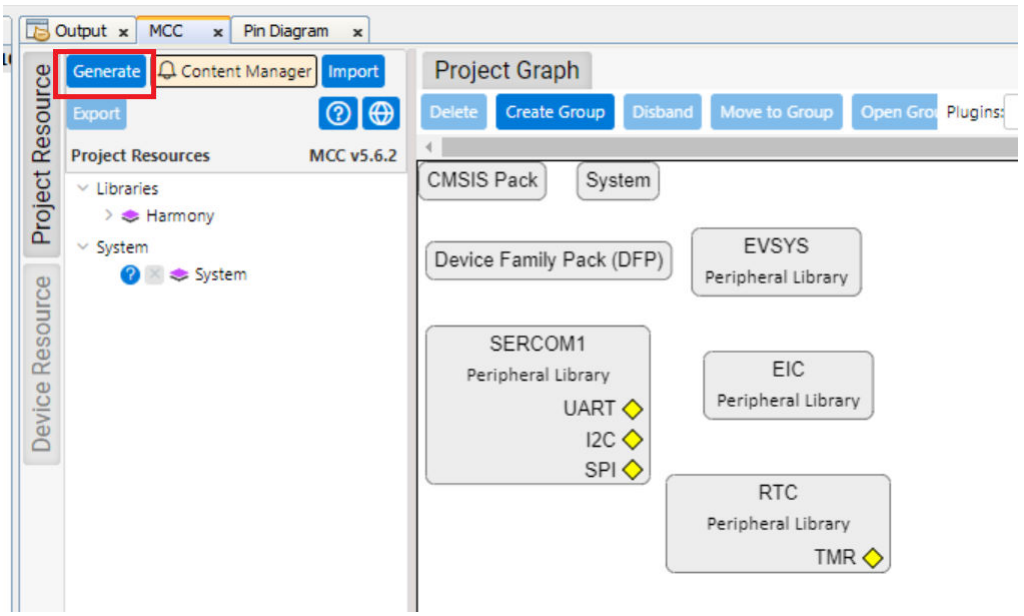


Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch	Pull Up	Pull Down	Slew Rate
1	PA05		Available	Digital	High Impedance	Low	☐	☐	DISABLED
2	PA06		Available	Digital	High Impedance	Low	☐	☐	DISABLED
3	PA07		Available	Digital	High Impedance	Low	☐	☐	DISABLED
4	PB00		SERCOM1_PAD0	Digital	High Impedance	n/a	☐	☐	DISABLED
5	PB01		SERCOM1_PAD1	Digital	High Impedance	n/a	☐	☐	DISABLED
6	PB02	LED	GPIO	Digital	Out	Low	☐	☐	DISABLED
7	PB03	SW	EIC_EXTINT3	Digital	In	n/a	☑	☐	DISABLED
8	PB04		Available	Digital	High Impedance	Low	☐	☐	DISABLED
9	PB05		Available	Digital	High Impedance	Low	☐	☐	DISABLED

5.3. Code Generation

After configuring the peripherals as shown in the following figure, click **Generate** under Resource Management [MCC].

Figure 5-14. Code Generation



Notes:

1. The generated code will add files and folders to the 32-bit MCC Harmony project. In the generated code, notice the Peripheral Library files generated for the Real-Time Clock (RTC), External Interrupt Controller (EIC), PORT peripherals, SERCOM1 (as Universal Synchronous Asynchronous Receiver Transmitter (USART)), and the Direct Memory Access (DMA) peripherals. MCC also generates the `main.c` file.
2. MCC provides an option to change the generated file name. By default, the file name `main.c` is used if a name is not assigned.

5.4. Adding Application Logic

To develop and run the application, follow these steps:

1. Open the `main.c` file of the project and add the following application code.

Figure 5-15. Logic for Register Callback Event Handlers

```
int main ( void )
{
    /* Initialize all modules */
    SYS_Initialize ( NULL );

    /* Register callback for DMA, RTC, and external interrupt */
    DMAC_ChannelCallbackRegister(DMAC_CHANNEL_0, uartDmaChannelHandler_Tx, 0);
    RTC_Timer32CallbackRegister(rtcEventHandler, 0);
    EIC_CallbackRegister(EIC_PIN_3, SW_userHandler, 0);

    /* Print initial message */
    snprintf((char*)uartTxBuffer, TX_BUFFER_SIZE, "************** Printing Toggling LED rate**************\r\n");
    isUARTTxComplete = false;
    DMAC_ChannelTransfer(DMAC_CHANNEL_0, uartTxBuffer, (const void *)&(SERCOM1_REGS->USART.SERCOM_DATA), strlen(

    /* Start the timer */
    RTC_Timer32Start();
```

2. Implement the registered callback event handlers for the peripherals by adding the following code before the `main()` function.

```
static void SW_userHandler(uintptr_t context)
{
    changeTempSamplingRate = true;
}
static void rtcEventHandler (RTC_TIMER32_INT_MASK intCause, uintptr_t context)
{
    if (intCause & RTC_MODE0_INTENSET_CMP0_Msk)
    {
        isRTCExpired = true;
    }
}
static void uartDmaChannelHandler_Tx(DMAC_TRANSFER_EVENT event, uintptr_t contextHandle)
{
    if (event == DMAC_TRANSFER_EVENT_COMPLETE)
    {
        isUARTTxComplete = true;
    }
}
```

3. According to the status of the isRTCExpired and isUARTTxComplete flags (these flags are handled by the rtcEventHandler and uartDmaChannelHandler_Tx event handlers when the RTC timer expires and when UART completes the data transfer), LED0 is toggled at a default rate of 500 ms. To change the toggling rate, if the user presses the SW switch, the toggling rate changes to 1s, 2s, and 4s and then returns to 500 ms with subsequent switch press events. The SW_userHandler is responsible for changing the toggling rate when the user presses the SW switch on the board.

   Inside the while loop, delete `SYS_Tasks()` and add the following code to toggle the LED at a default rate of 500 ms.

```
if ((isRTCExpired == true) && (true == isUARTTxComplete))
        {
            isRTCExpired = false;
            isUARTTxComplete = false;
            LED_Toggle();
            sprintf((char*)(uartTxBuffer), "Toggling LED at %s rate
            \r\n",&timeouts[(uint8_t)tempSampleRate][0]);
            DMAC_ChannelTransfer(DMAC_CHANNEL_0, uartTxBuffer,(const void *)&
            (SERCOM1_REGS->USART.SERCOM_DATA),strlen((const char*)uartTxBuffer));
        }
```

4. Add the following code immediately after the code above to change the toggling rate when a switch press event occurs.

```
if(changeTempSamplingRate == true)
        {
            changeTempSamplingRate = false;
            if(tempSampleRate == TEMP_SAMPLING_RATE_500MS)
            {
```

```
                tempSampleRate = TEMP_SAMPLING_RATE_1S;
                RTC_Timer32CompareSet(PERIOD_1S);
            }
            else if(tempSampleRate == TEMP_SAMPLING_RATE_1S)
            {
                tempSampleRate = TEMP_SAMPLING_RATE_2S;
                RTC_Timer32CompareSet(PERIOD_2S);
            }
            else if(tempSampleRate == TEMP_SAMPLING_RATE_2S)
            {
                tempSampleRate = TEMP_SAMPLING_RATE_4S;
                RTC_Timer32CompareSet(PERIOD_4S);
            }
            else if(tempSampleRate == TEMP_SAMPLING_RATE_4S)
            {
                tempSampleRate = TEMP_SAMPLING_RATE_500MS;
                RTC_Timer32CompareSet(PERIOD_500MS);
            }
            else
            {
            ;
            }
            RTC_Timer32CounterSet(0);
            sprintf((char*)uartLocalTxBuffer, "LED Toggling rate is changed to %s\r\n",
            &timeouts[(uint8_t)tempSampleRate][0]);
            DMAC_ChannelTransfer(DMAC_CHANNEL_0, uartLocalTxBuffer, (const void *)
            &(SERCOM1_REGS->USART.SERCOM_DATA), strlen((const char*)uartLocalTxBuffer));
        }
```

5. Add the following code to include the necessary header files and define the macros for different RTC compare values.

```
#include <stddef.h>                     // Defines NULL
#include <stdbool.h>                    // Defines true
#include <stdlib.h>                     // Defines EXIT_FAILURE
#include "definitions.h"                // SYS function prototypes
#include <string.h>
#include <stdio.h>
```

This code declares the various flags whose status is monitored and changed by the event handlers in the application. It also includes declarations and definitions of arrays used to print the LED toggling rate on the console.

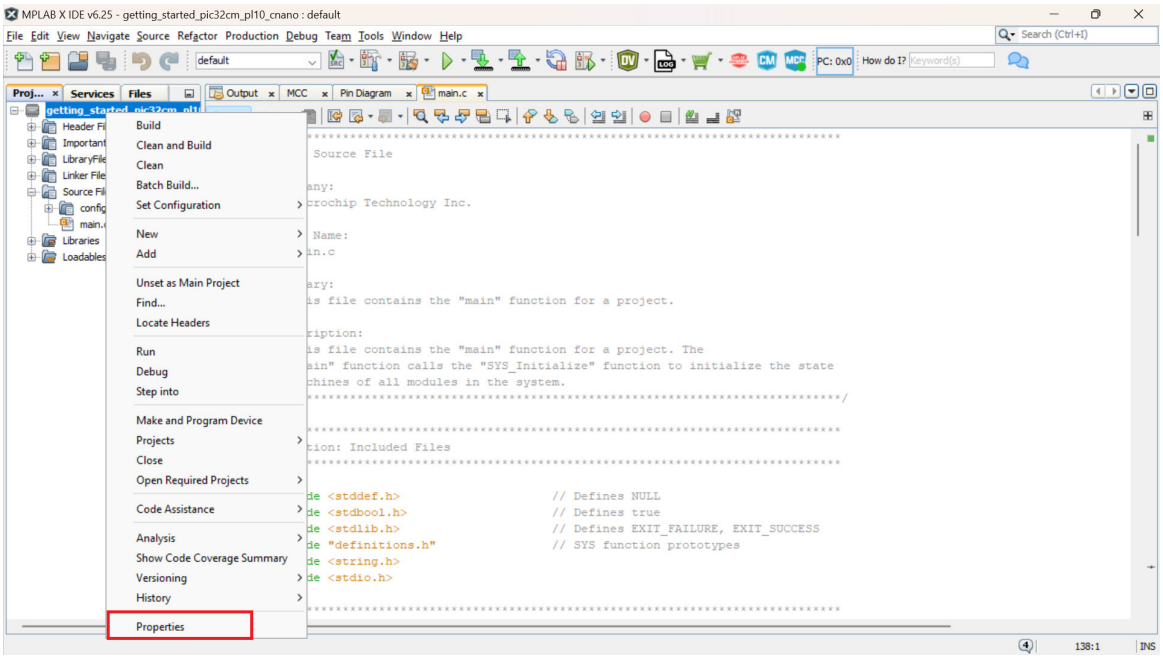
```
static void SW_userHandler(uintptr_t context);
static void rtcEventHandler (RTC_TIMER32_INT_MASK intCause, uintptr_t context);
static void uartDmaChannelHandler_Tx(DMAC_TRANSFER_EVENT event, uintptr_t contextHandle);

/* Timer Counter Time period match values for input clock of 4096 Hz */
#define PERIOD_500MS        (512)
#define PERIOD_1S           (1024)
#define PERIOD_2S           (2048)
#define PERIOD_4S           (4096)

#define TX_BUFFER_SIZE      (100)

static volatile bool isRTCExpired = false;
static volatile bool changeTempSamplingRate = false;
static volatile bool isUARTTxComplete = true;
static volatile bool isUARTRxComplete = false;

static uint8_t uartTxBuffer[TX_BUFFER_SIZE] = {0};
```

5.5. Building and Programming Application

To build and program the application, follow these steps:

1. The PIC32CM Curiosity Nano evaluation kit supports debugging using a debugger. Connect the USB Type-C® cable to the PIC32CM Curiosity Nano evaluation kit to power and debug the PIC32CM PL10 Curiosity Nano board.
2. Ensure that the compiler optimization is set to 1. To check this, follow these steps:
   a. Right-click on the project `getting_started_pic32cm_pl10_cnano`, a shortcut menu will appear. Click Properties.

Figure 5-16. Project Properties



b. In the Project Properties window, under Option Categories, select Optimization, and from the optimization-level item list, select 1.

c. Click **OK** to close the Project Properties window.

Figure 5-17. Compiler Optimization Level

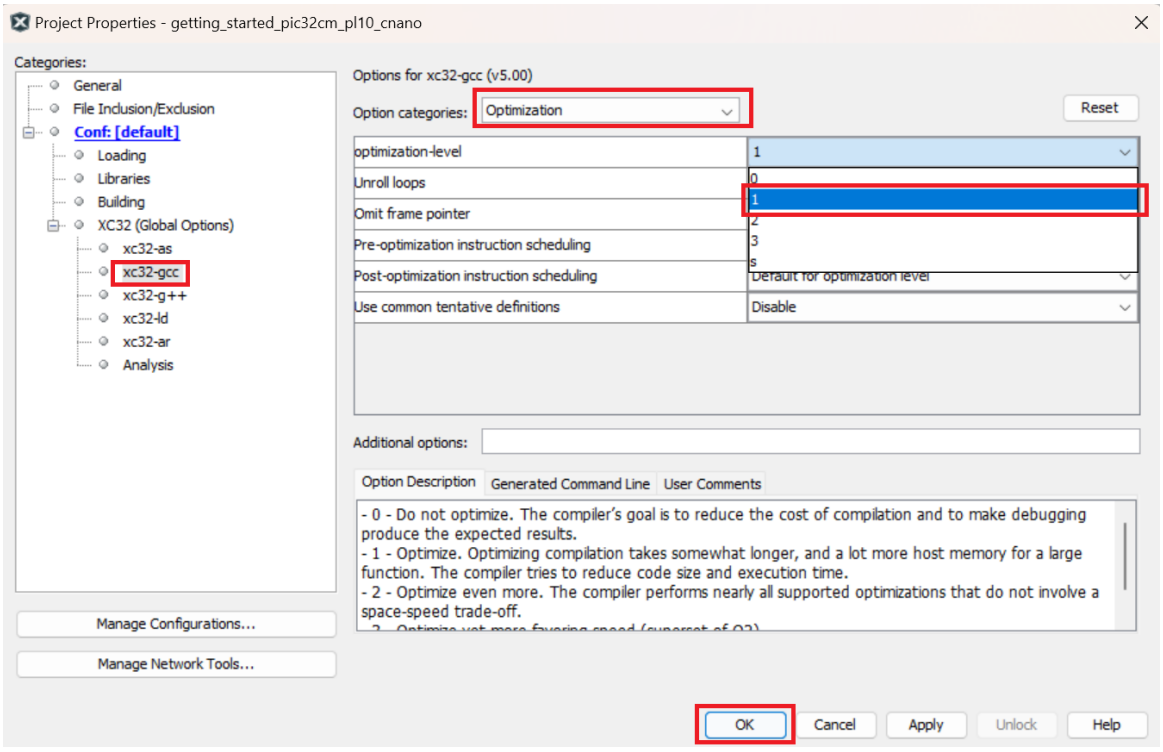
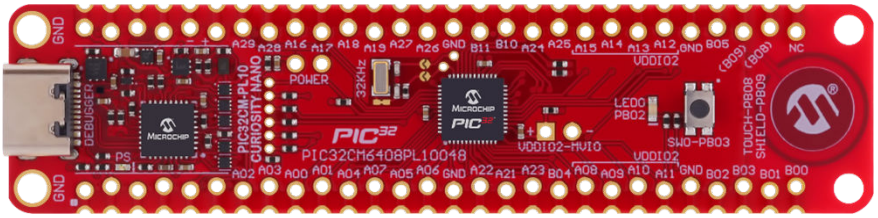
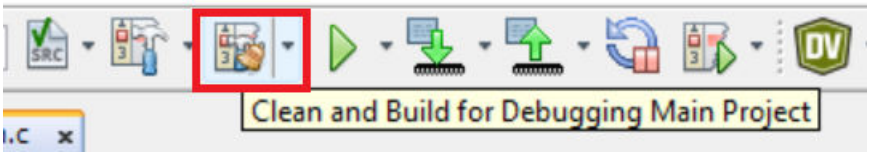


Figure 5-18. Hardware Setup



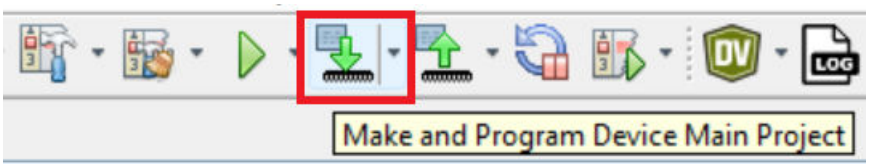
3. Set `getting_started_pic32cm_pl10_cnano` as the main project, and in Project Properties, select the latest compiler version (`v5.00`). Clean and build the project by clicking the highlighted icon.

Figure 5-19. Clean and Build Main Project



4. Program the application by clicking the highlighted icon.

Figure 5-20. Make and Program Device

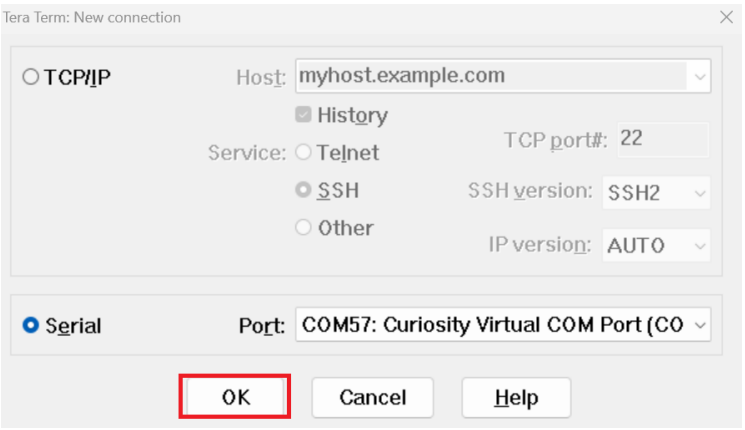


5.6. Observing the Output on the Board and Serial Terminal

To observe the output on the board and serial terminal, follow these steps:

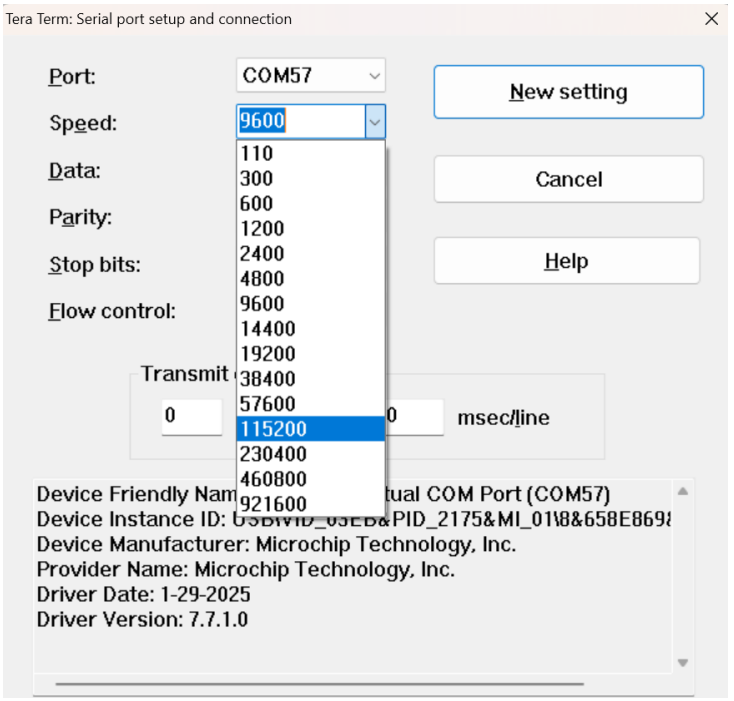
1. Open any terminal window (TeraTerm in this case).
2. Select the required serial port and then click **OK**.

Figure 5-21. Selection of Serial COM Port



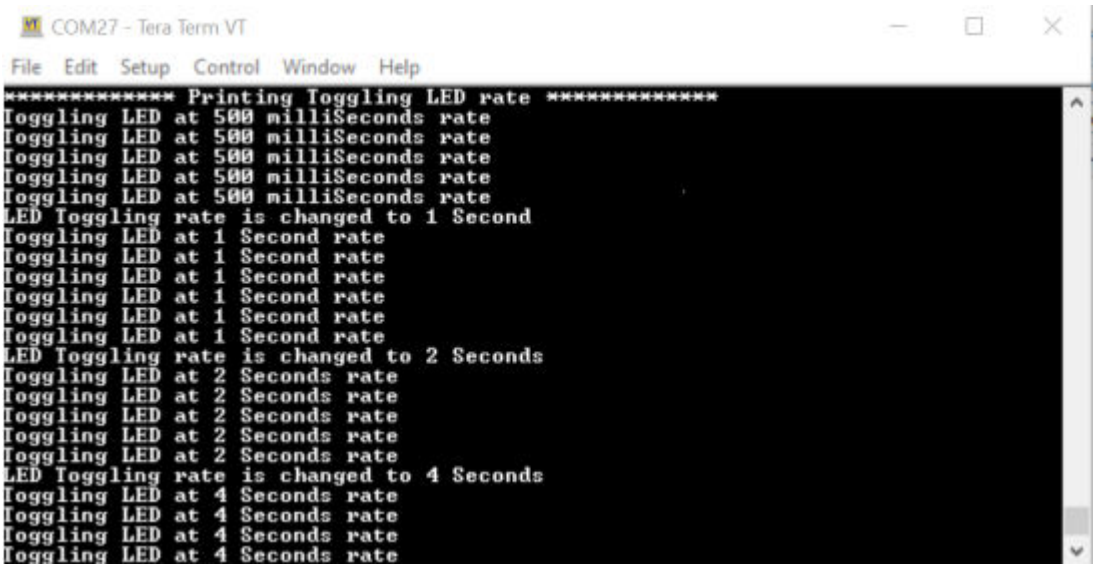
3. In the TeraTerm serial port setup and connection dialog box, type or select 115200 as the baud rate in the Speed box.

Figure 5-22. Setting the Baud Rate



4. An LED on the PIC32CM Curiosity Nano evaluation kit toggles on a timeout basis, with a default periodicity of 500 ms.
5. The LED toggling rate is displayed on the serial terminal.
6. Press the SW switch on the PIC32CM Curiosity Nano evaluation kit to change the default timeout periodicity to 1s.
7. Each subsequent press of the SW switch on the PIC32CM Curiosity Nano evaluation kit changes the timeout periodicity to 2s, 4s, 500 ms, and back to 1s in cyclic order.

Figure 5-23. Output Window



As the LED toggling rate displayed on the serial terminal changes with each subsequent switch press, observe the same change in the toggling rate of LED0 on the evaluation kit.

6. What's Next

Now that you have learned where to find information on Microchip products and how to get started, download the software and purchase the hardware tools needed to begin developing your products. For additional information on related PIC32 products and IDE, refer to the links below:

Software:

- VS Code with extensions
- MPLAB X IDE
- MPLAB Code Configurator
- XC32 Compiler

Firmware:

- MPLAB Discover examples
- GitHub examples

Recommended Programming/Debugging Tools:

MPLAB PICkit 5:

- Buy Here

MPLAB ICD 5:

- Buy Here

Atmel ICE:

- Buy Here

Power Debugger:

- Buy Here

7. References

- PIC32CM6408PL10048 Curiosity Nano Hardware User Guide (*DS50004003*)
- For additional information on MPLAB Harmony, refer to the Microchip website
- Developer Help: Get Started for MPLAB® X IDE Users New to Visual Studio Code (VS Code®)
- Microchip University Course: Introduction to MPLAB® X IDE
- Microchip University Course: Overview of the MPLAB® Code Configurator (MCC) Content Manager (CM)
- Microchip University Course: MPLAB Data Visualizer (for Curiosity Nano)
- Additional online technical documentation concerning various products: onlinedocs.microchip.com
- Microchip Technical Support: support.microchip.com/s/
- For the example application, refer to the “Getting Started with the Arm® Cortex®-M0+ based PIC32CM Microcontroller Families”: https://www.microchip.com/en-us/development-tool/ev10p22a
- For more information on various applications, refer to the discover-Microchip-PIC32-PL10
- Migration Guide From AVR® to PIC32CM MCUs (*DS40002662*)
- Migration Guide from AVR® and PIC16/PIC18F to PIC32CM Development Tools Ecosystem (*DS40002663*)
- For additional info about 32-bit Microcontroller Collaterals and Solutions, refer to the 32-bit Microcontroller Collateral and Solutions Reference Guide (*DS70005534*)

8. Revision History

Doc. Rev.	Date	Comments
A	02/2026	Initial document release

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: 979-8-3371-2745-3

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.

Product Page Links

PIC32CM6408PL10048