Lab Manual for Creating Simple Embedded Applications with 32-bit MCUs/MPUs using the MPLAB® Harmony Software Framework

Table Of Contents

Introduction:

The lab exercises in this manual are intended to reinforce usage of MPLAB® Harmony v3 Peripheral Libraries.

This manual provides specific instructions to configure the MPLAB Harmony projects on a SAM MCU to successfully complete the exercises. Upon completion of the class you are encouraged to use additional resources (identified in the class materials) to further your knowledge and understanding of MPLAB Harmony v3 and SAM MCUs/MPUs.

Required Development Tools:

1. ATSAML10-XPRO: SAM L10 Xplained Pro evaluation kit
2. ATIO1-XPRO: I/O1 Xplained Pro Extension Kit
3. MPLAB X IDE v5.45 or higher
4. MPLAB® XC32 Compiler v2.50 or higher
5. MPLAB Harmony CSP ,DEV_PACKS
6. MPLAB Code Configurator Plugin v5.0 or higher

Upon completion, you will:

- Have hands-on experience with the MPLAB Harmony.
- Have hands-on experience with MPLAB Code Configurator
- Understand key benefits and features of MPLAB Harmony

Prerequisites:

The lab material assumes you have prior experience with:

- MPLAB X IDE basic usage
- Basic understanding of MPLAB Harmony
- MPLAB based Programming/Debugging fundamentals
- C language programming
- Basic knowledge on SAM product family

Lab 1 - Create an application to read temperature sensor data periodically and print it on a serial terminal using Harmony Peripheral Libraries

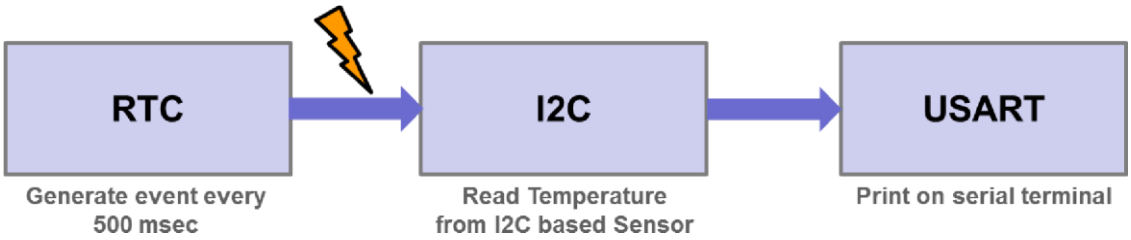
Purpose:

After completing Lab 1, you will This tutorial shows you how to use MCC to create an application that gets you started in developing applications on SAM L10 MCUs using the MPLAB Harmony v3 software framework. In the process, you will understand the project structure and learn how to develop applications using PLIBs.

Overview:

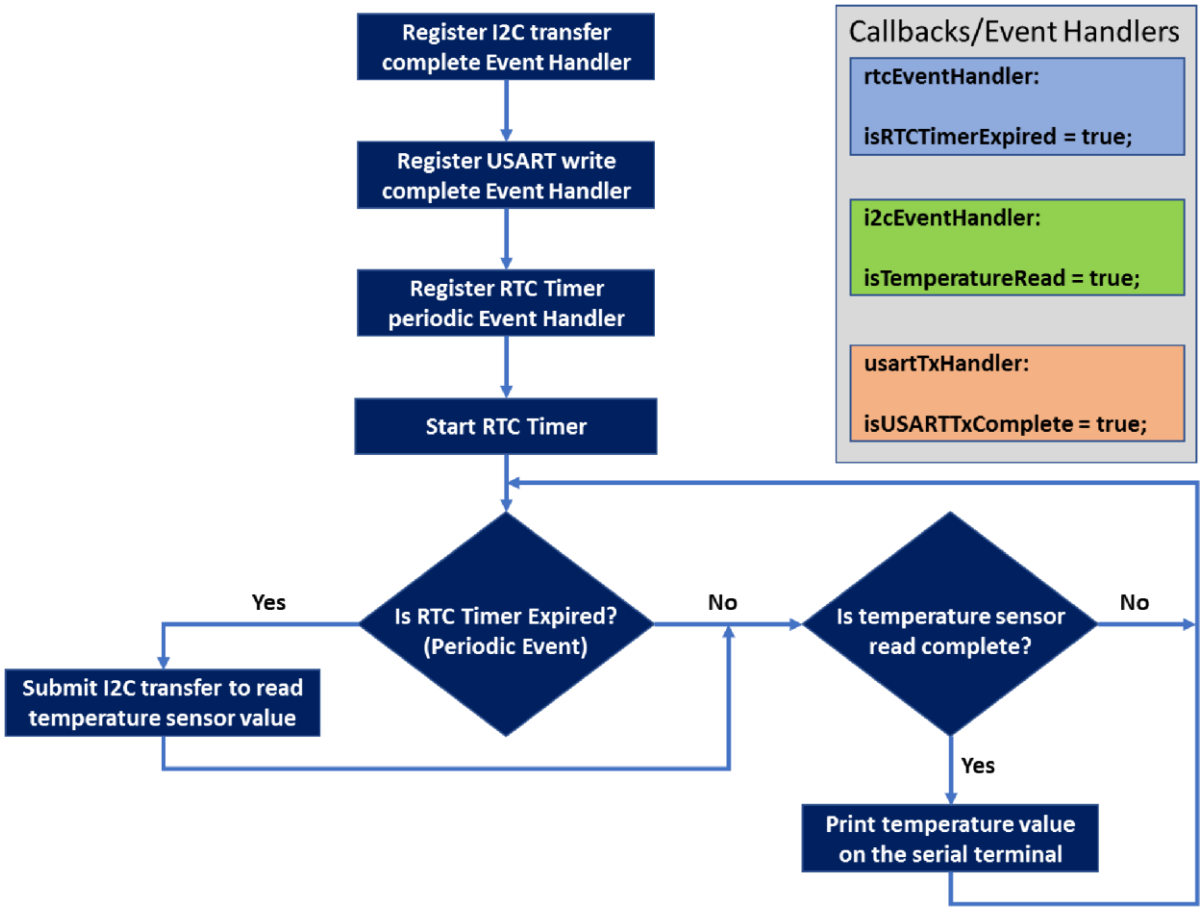
In this lab you will create a MPLAB Harmony project using MCC. The application you create will read temperature from an I2C based temperature sensor and print it on a serial terminal, every 500 milliseconds. The application will use:

- RTC Peripheral Library to generate periodic events
- SERCOM-I2C Peripheral Library to read temperature from an I2C based temperature sensor
- SERCOM-USART Peripheral Library to print the temperature values on a serial port terminal application running on a PC

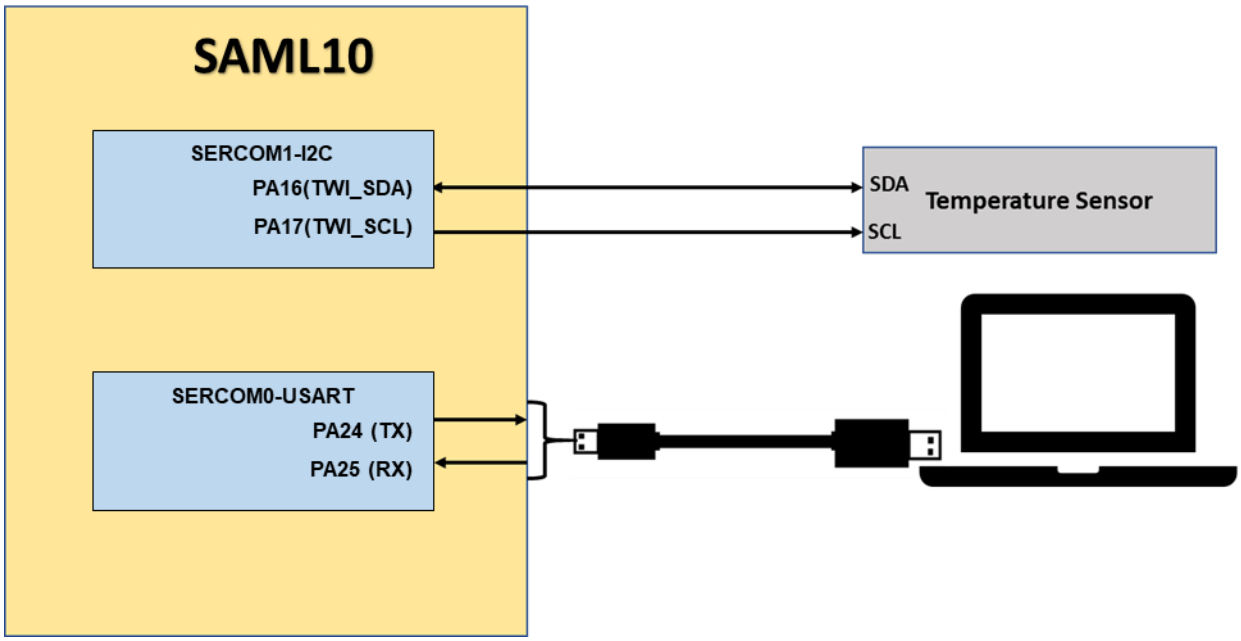


In the process, the lab will also demonstrate the use of callbacks functions.

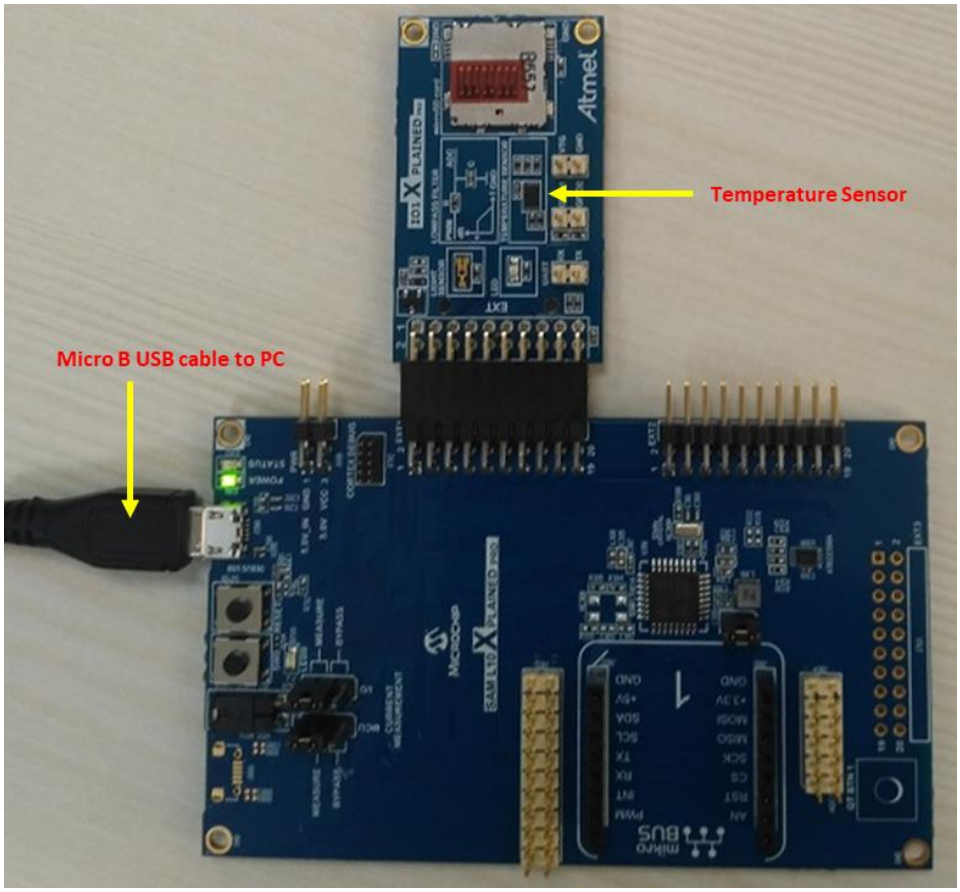
Flowchart:



Connection Diagram:



Hardware Setup:



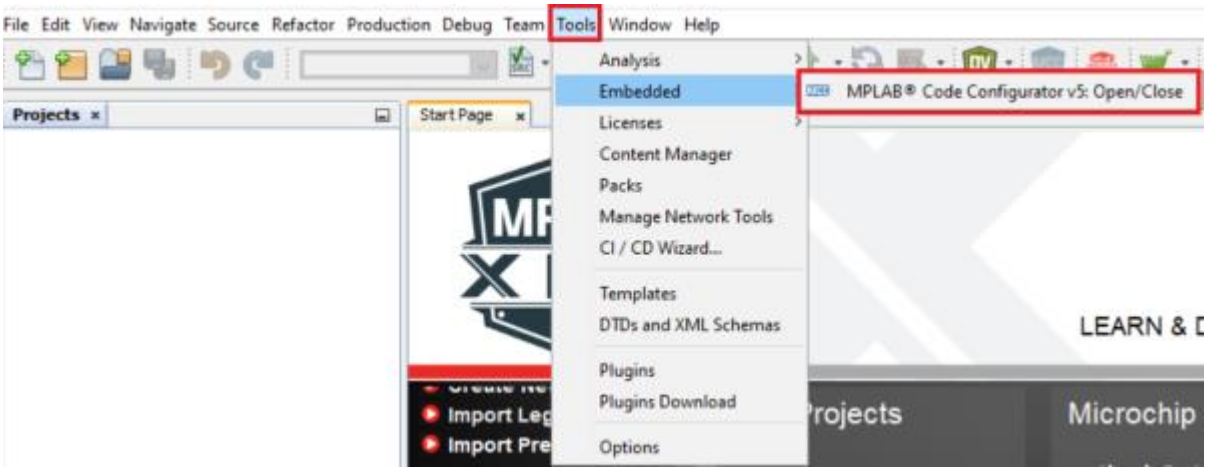
Note: IO1 Xplained Pro extension board features Microchip's AT30TSE758 temperature sensor chip with an 8kbit serial EEPROM. The temperature sensor has two I2C addresses, one for the temperature sensor and one for the EEPROM

This lab is completed in the following steps:
Part 1: Create project and configure SAML10
STEP 1: Install the MPLAB® Code Configurator (MCC) Plugin in the MPLAB X IDE
STEP 2 Create MCC Harmony Project using MPLAB X IDE
STEP 3: Verify Clock Settings and setup RTC clock
STEP 4: Configure RTC Peripheral Library
STEP 5: Configure I2C Peripheral Library and I2C pins
STEP 6: Configure USART Peripheral Library and USART pins
STEP 7: Configure LED pin
STEP 8: Generate code
Part 2: Add application code and build the application
STEP 9: Add application code to the project
STEP 10: Build and Run the Application

Procedure:

STEP 1: Install the MPLAB® Code Configurator (MCC) Plugin in the MPLAB X IDE

- Launch MPLAB X IDE from the Windows® Start Menu. Close any projects and files that are currently open
- Go to Tools > Embedded.
- You will see MPLAB® Code Configurator in the menu. If you don't see it in the menu, please Install MCC



Note:

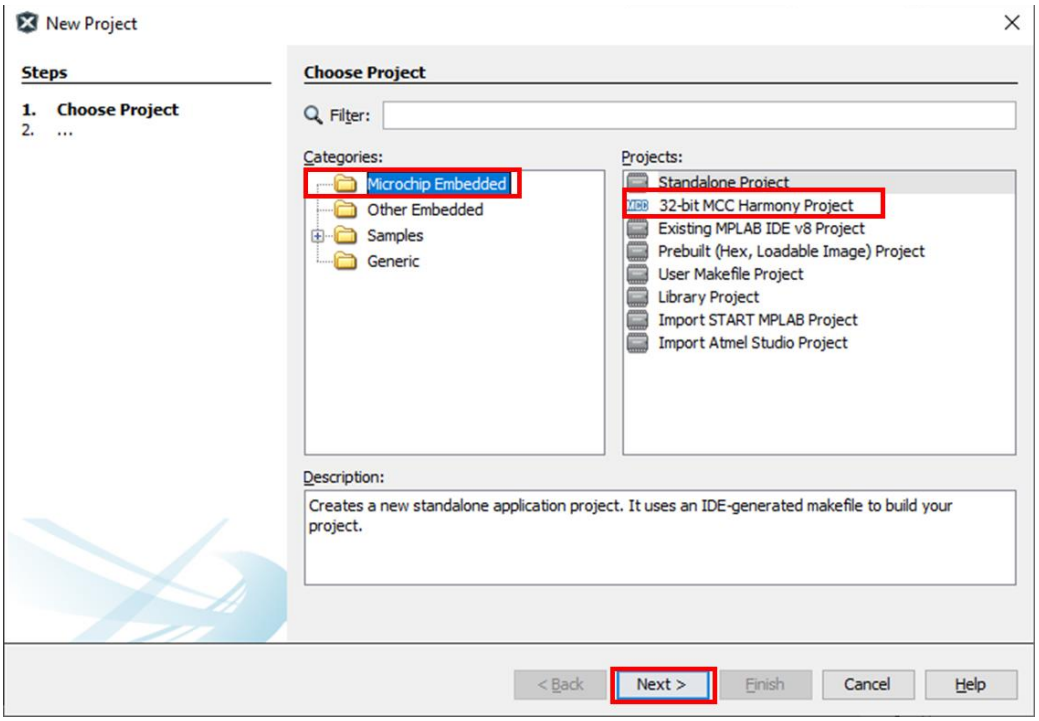
- When you launch MCC for the first time, it displays a prompt asking for the mode in which you would like to use MCC.
  - Standalone mode (as a separate window)
  - Native mode (as an embedded window in MPLAB X IDE)
- Standalone mode is the default mode.
- MCC will launch in the operating mode selected the first time, every time you launch MCC.
- For this lab, MCC is configured to operate in Standalone mode.

STEP 2: Create MPLAB Harmony Project using MPLAB X IDE

Step 2a: Close any currently Opened project in MPLAB IDE

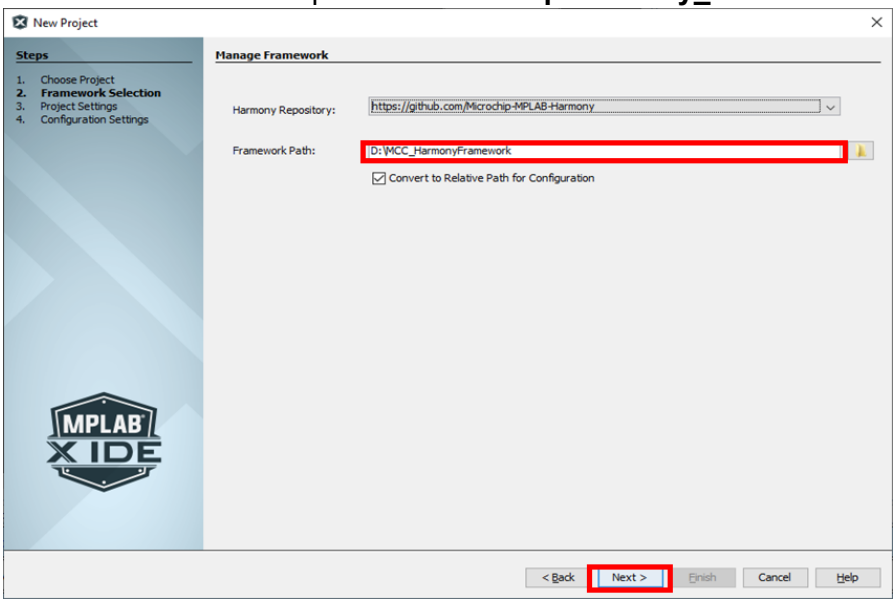
Step 2b: Select *File > New Project* from the main IDE menu.

Step 2c: In the Categories pane of the New Project dialog, select Microchip Embedded. In the Projects pane, select 32-bit MCC Harmony Project, and then click Next.



Note: If “32-Bit MCC Harmony Project” is not displayed, please download and install the MCC before continuing with further steps. Refer Step 1 for details.

Step 2d: In “Framework Path” edit box, enter the path to the folder in which MPLAB Harmony 3 packages are downloaded. For Example: **“D:\microchip\harmony_3”.**

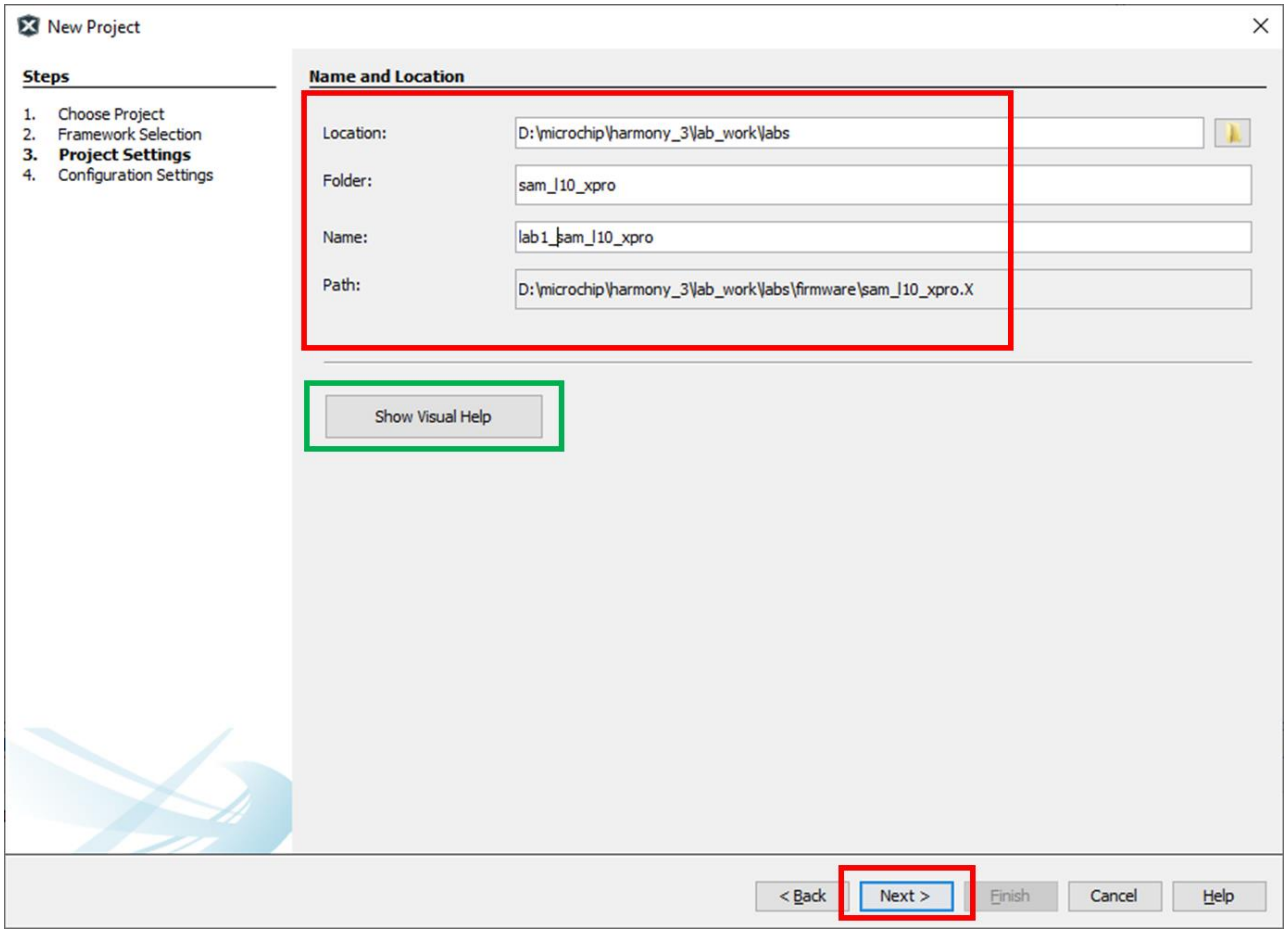


Step 2e: In the New Project window, perform the following settings:

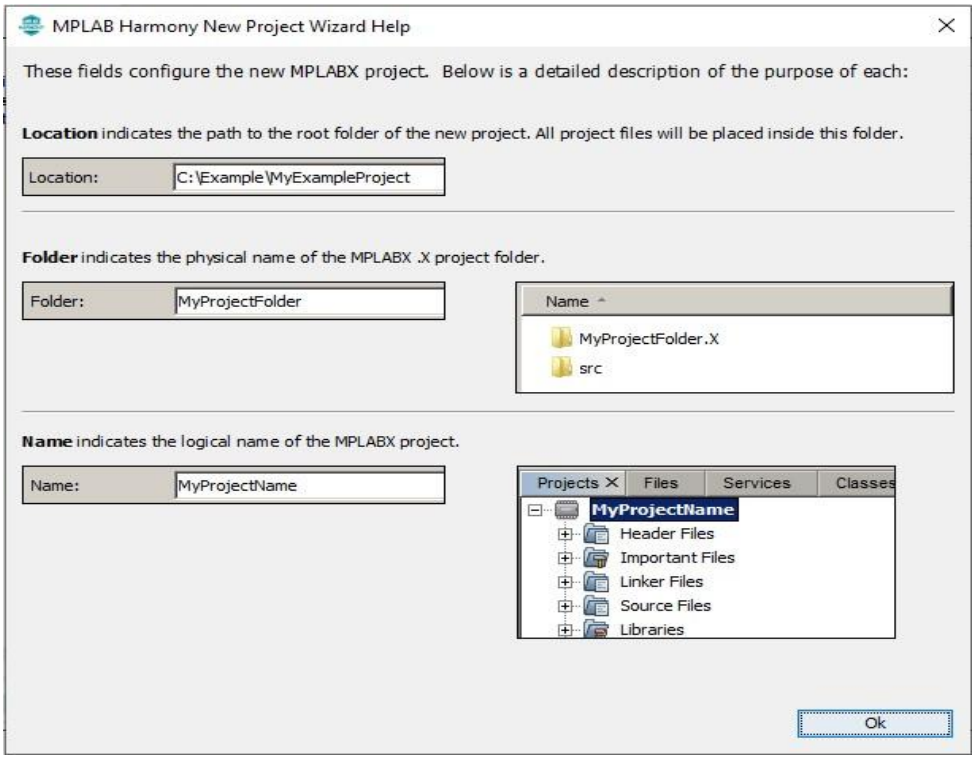
- *Location*: Indicates the path to the root folder of the new project. All project files will be placed inside this folder. The project location can be any valid path, for example: **"D:\microchip\harmony_3\lab_work\labs".**
- *Folder*: Indicates the name of the MPLABX .X folder. Enter "**sam_l10 _xpro**" to create a sam_l10_xpro.X folder.
  Note: This must be a valid directory name for your operating system.
- *Name*: Enter the project's logical name as **"lab1_sam_l10_xpro.X"**. This is the name that will be shown from within the MPLAB X IDE.

Note: The *Path* box is read-only. It will update as you make changes to the other entries.

- Click **Next** to proceed to Configuration Settings.

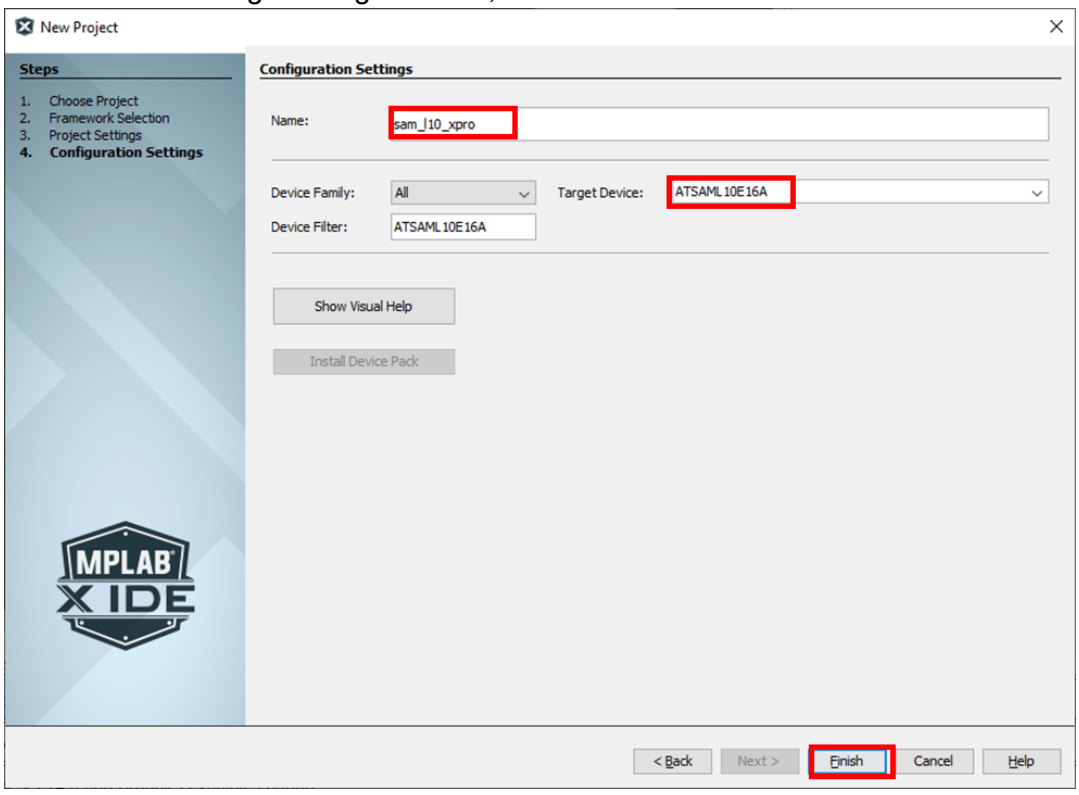


Note: Clicking on the "Show Visual Help" button will open a help window providing a detailed description of the various fields in the *Project Settings* window.



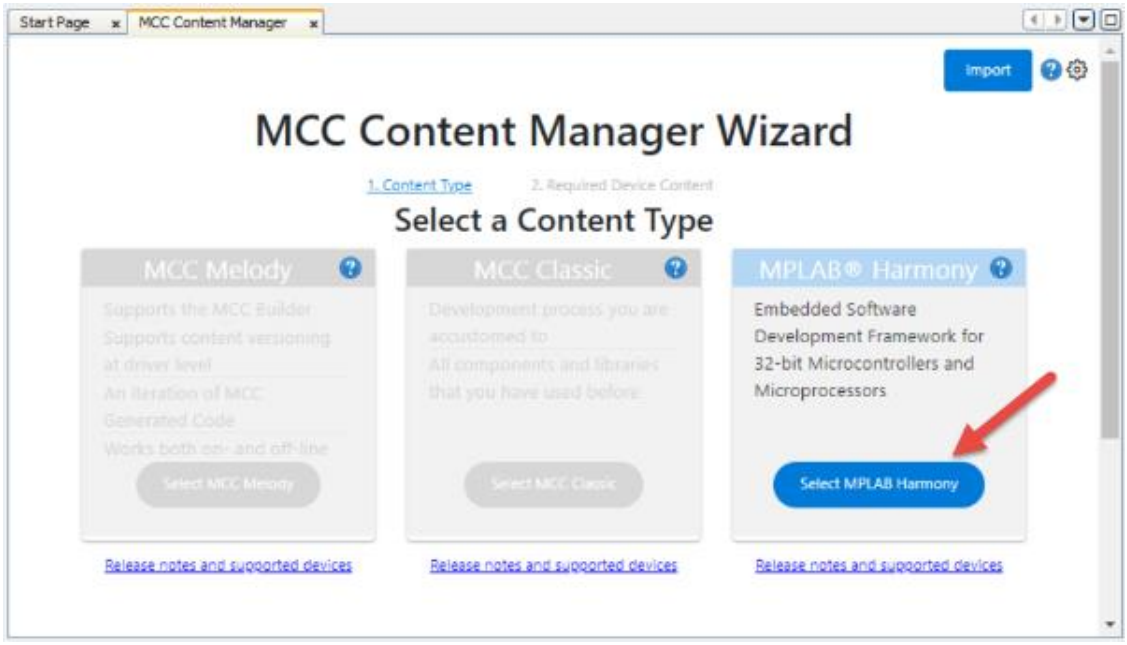
Step 2f: Follow the below steps to setup the project's *Configuration Settings*

- *Name*: Enter the configuration name as "**sam_l10_xpro**".
- *Target Device*: Select "**ATSAML10E16A**" as the target device.
  Note: You can select the Device Family or enter a partial device name to filter the list in Target Devices in order to make it easier to find the desired device.
- After selecting the target device, click **Finish** to launch.



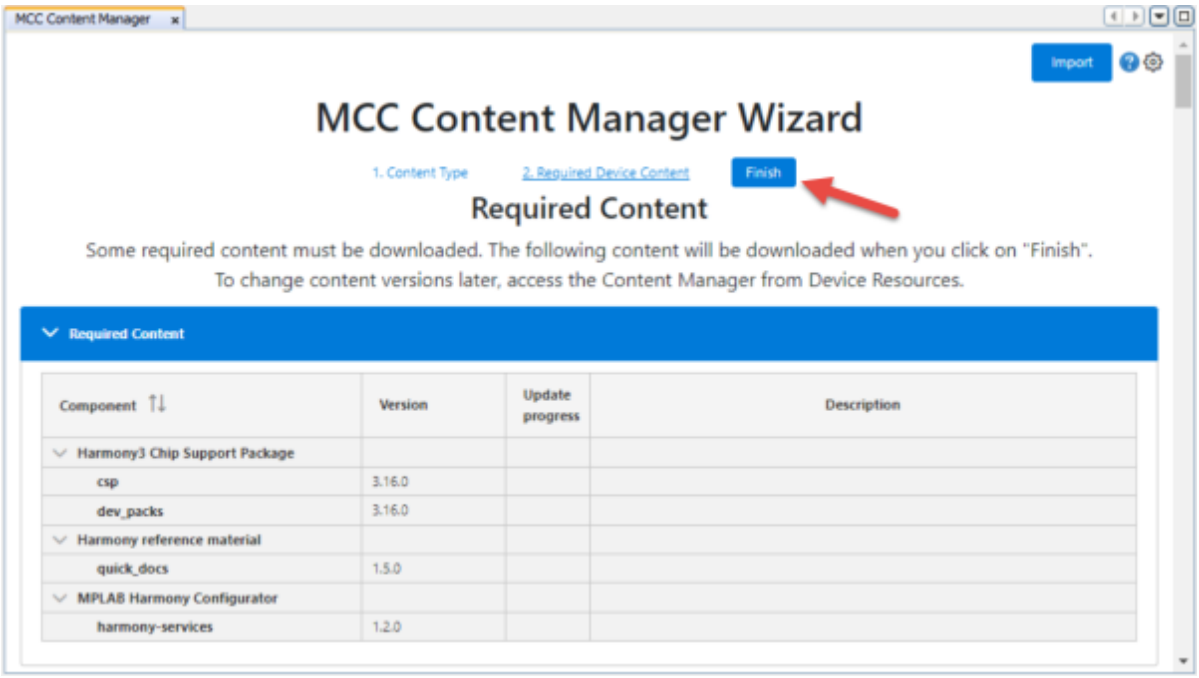
Step 2g: Download the MPLAB Harmony framework (if using the first time)

When the project opens, the MCC Content Manager Wizard window will be displayed. Click the **Select MPLAB Harmony** button to select which MPLAB Harmony framework components you want to download.



The MCC Content Manager Wizard will compare the Harmony Repository with the Framework Path (as specified above), and provide you with a list of components you may want to download. This list will include updated components and never-downloaded components.

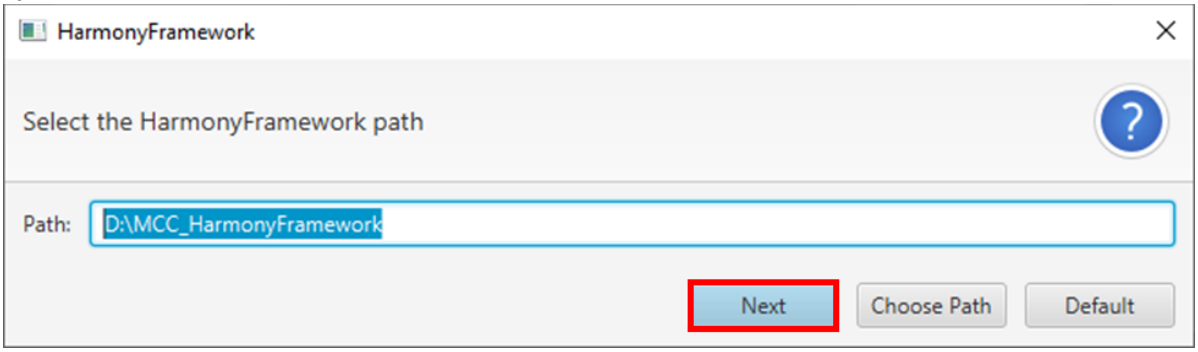
If this is your first time downloading the framework or if you've chosen a new framework path, the required components will already be selected for you. To keep your first Harmony project simple, just click **Finish** to download only the required components (about 5 GB):



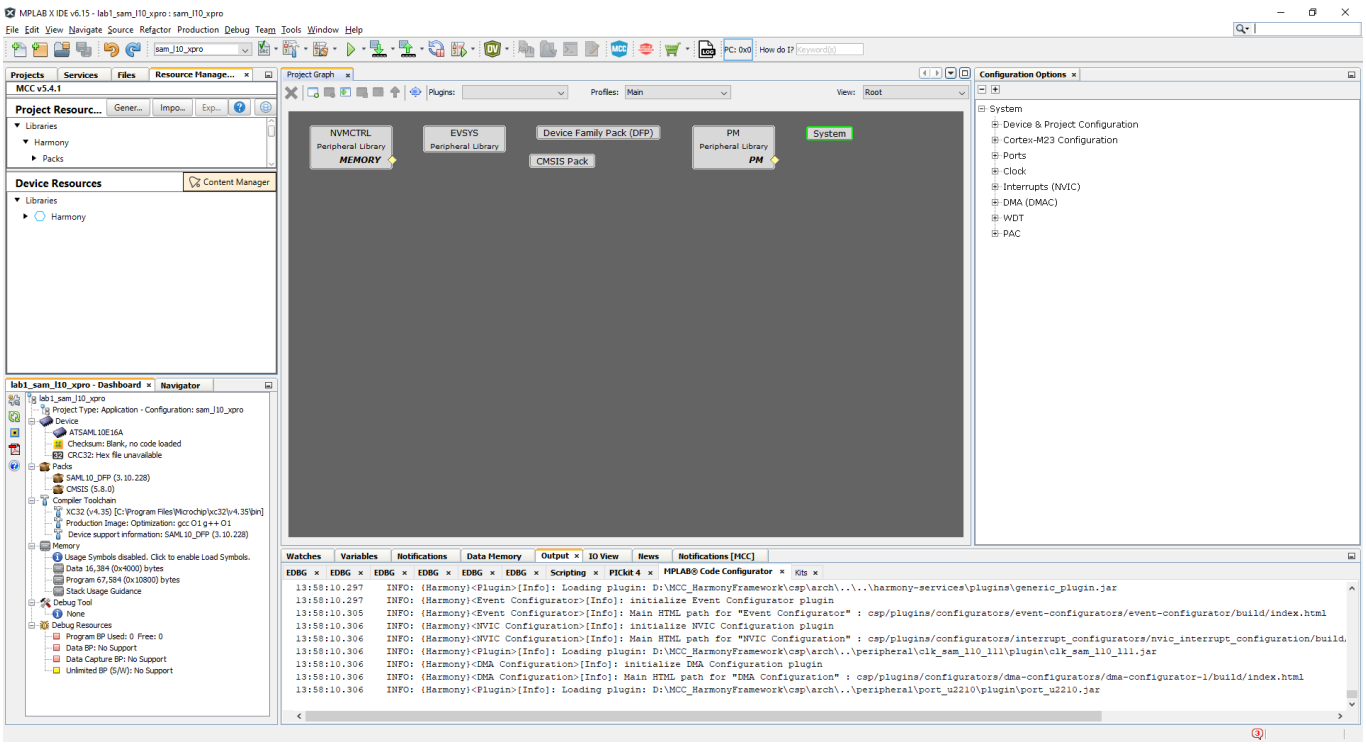
- Harmony 3 Chip Support Package
  - csp (Chip Support Package includes low-level Harmony Peripheral Libraries (PLIBs))
  - dev_packs (includes ARM CMSIS and Microchip Device Family Packs)
- Harmony reference material
  - quick_docs component provides Harmony help through standalone HTML pages
- MPLAB Harmony Configurator
  - harmony-services provides all Harmony plugins for the MPLAB X IDE

MCC will automatically start after the download (about 5 GB) completes.

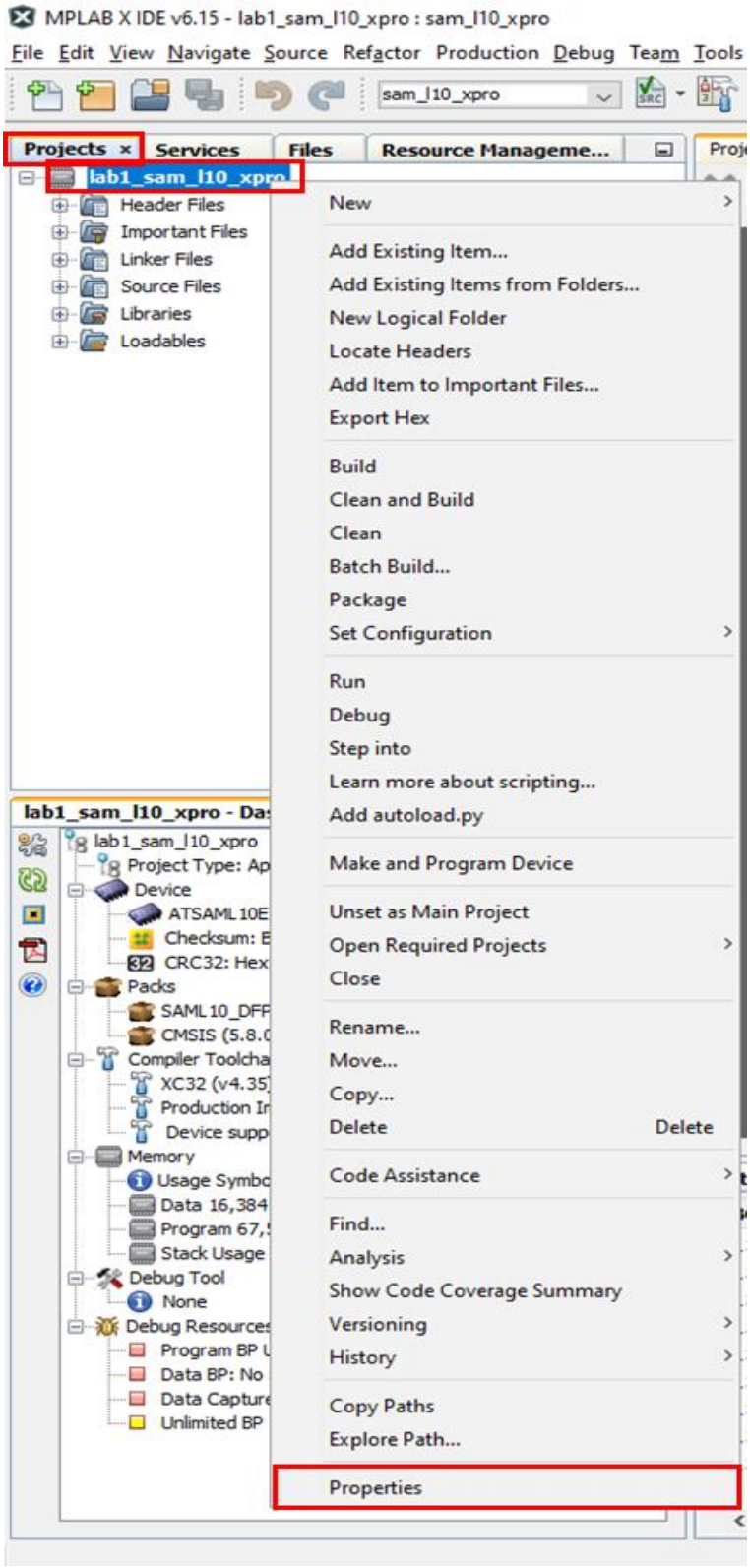
If already installed confirm the path as shown below



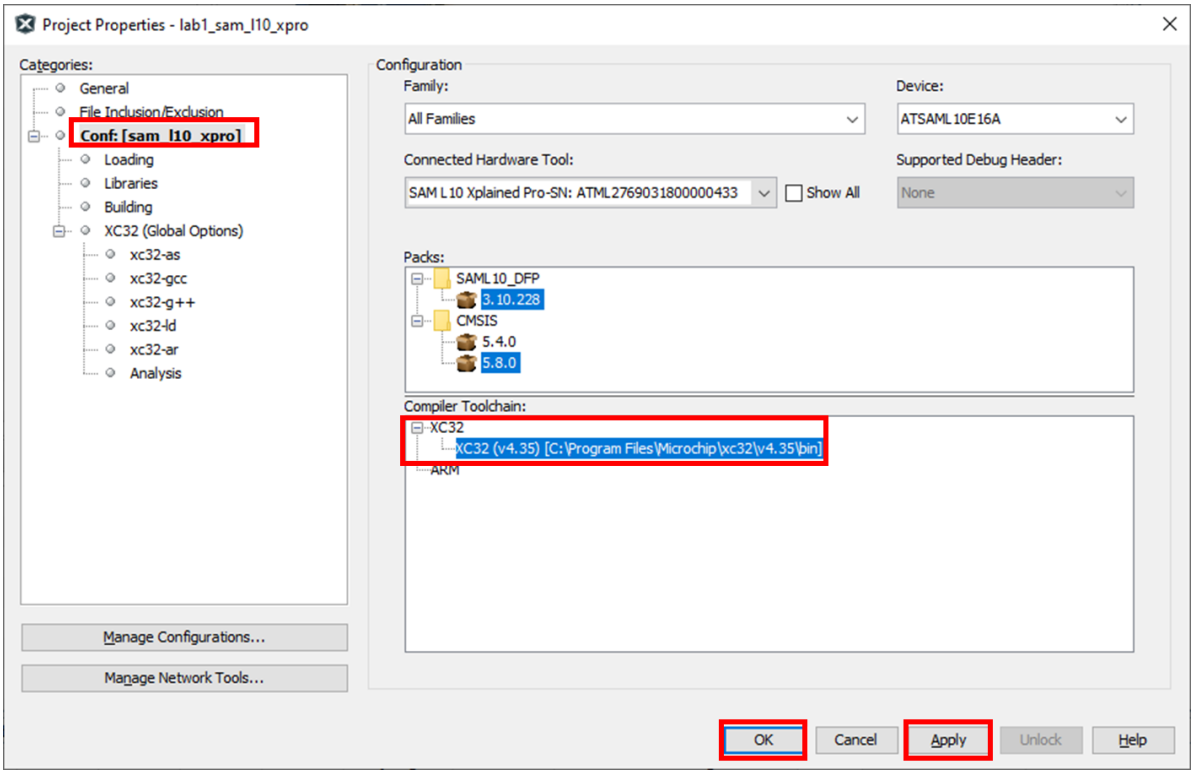
Step 2h: The MCC plugin's main window for the project will be displayed.



Step 2i: Before proceeding, setup the compiler toolchain. Click on the “Projects” tab on the top left pane. Right click on the project name “**lab1_sam_l10_xpro**” and go to “**Properties**”.



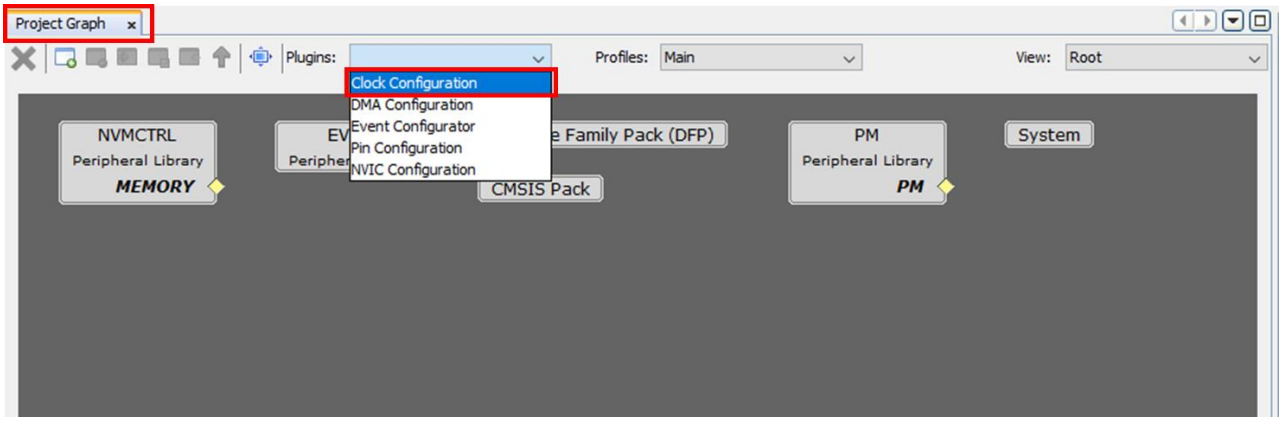
Make sure that **XC32 (v4.35)** is selected as the Compiler Toolchain.
Click on **Apply** and then click on **OK.**



Tool Tip: - If you closed the Project graph accidently, and would like to open it again, go to *Tools* ▶ *Embedded* ▶ *MPLAB Code Configurator* v5:Open/Close

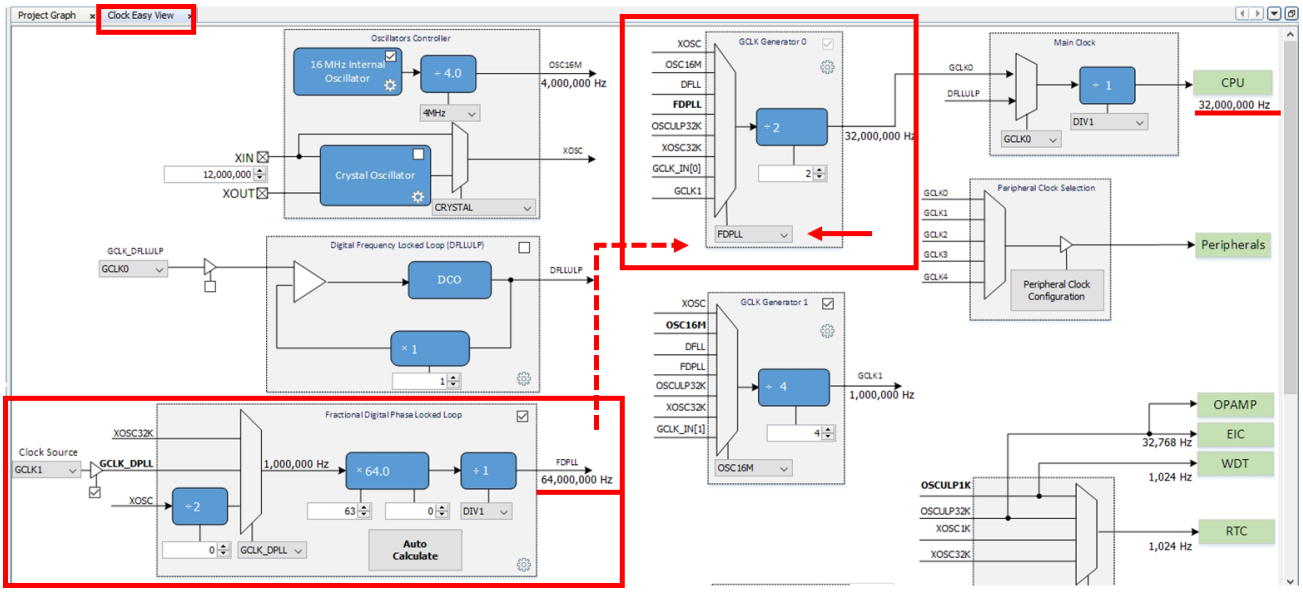
STEP 3: Configure clock peripheral

Step 3a: Launch Clock Easy View by going to **Project Graph** tab in MPLABX IDE and then selecting **Plugins > Clock Configuration**



A new window “Clock Easy View” is opened in the project’s Main Window.

Step 3b: Click on the **Clock Easy View** tab, scroll to the right and verify that the **Main Clock** is set to 32 MHz

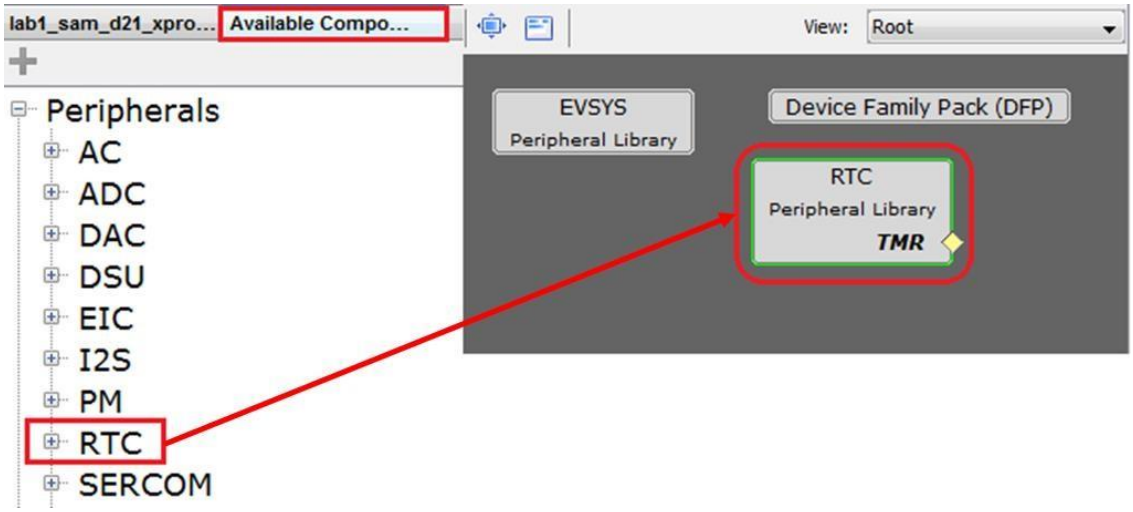


***Tool Tip*: –** Double click on "**Clock Easy View**" tab to maximize the window.

Note: The input clock source to the clock generator *GCLK Generator 0* is *F*DPLL (Fractional Digital Phased Locked Loop)

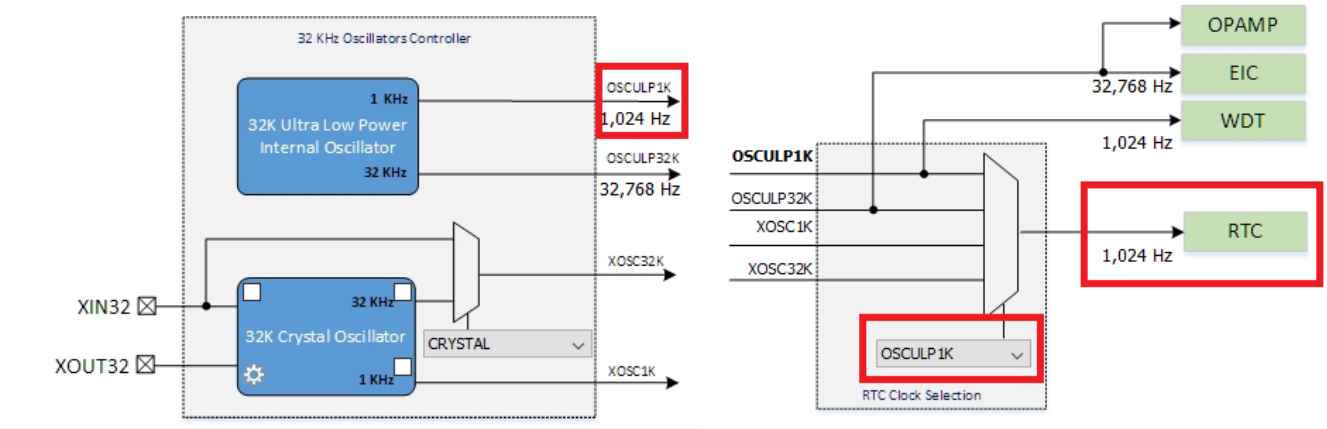
STEP 4: Configure RTC Peripheral Library

Step 4a: Under the bottom left tab "**Available Components**", expand **Peripherals > RTC**. Select and double click on RTC to add the RTC peripheral library (PLIB) to the project graph.



Step 4b: Verify the default RTC clock under **Clock Easy View**.

When a module is added to the project graph, MCC automatically enables the clock to the module.
Go to the **Clock Easy View** tab and verify the input clock to the RTC module. The default RTC clock source is OSCULP1K which is set to 1024 Hz.

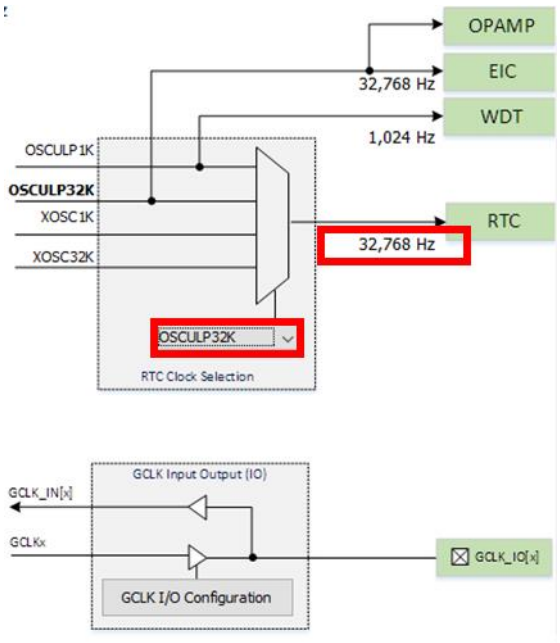


Step 4c: Setup the RTC clock to run from the 32 kHz ultra-low power oscillator.

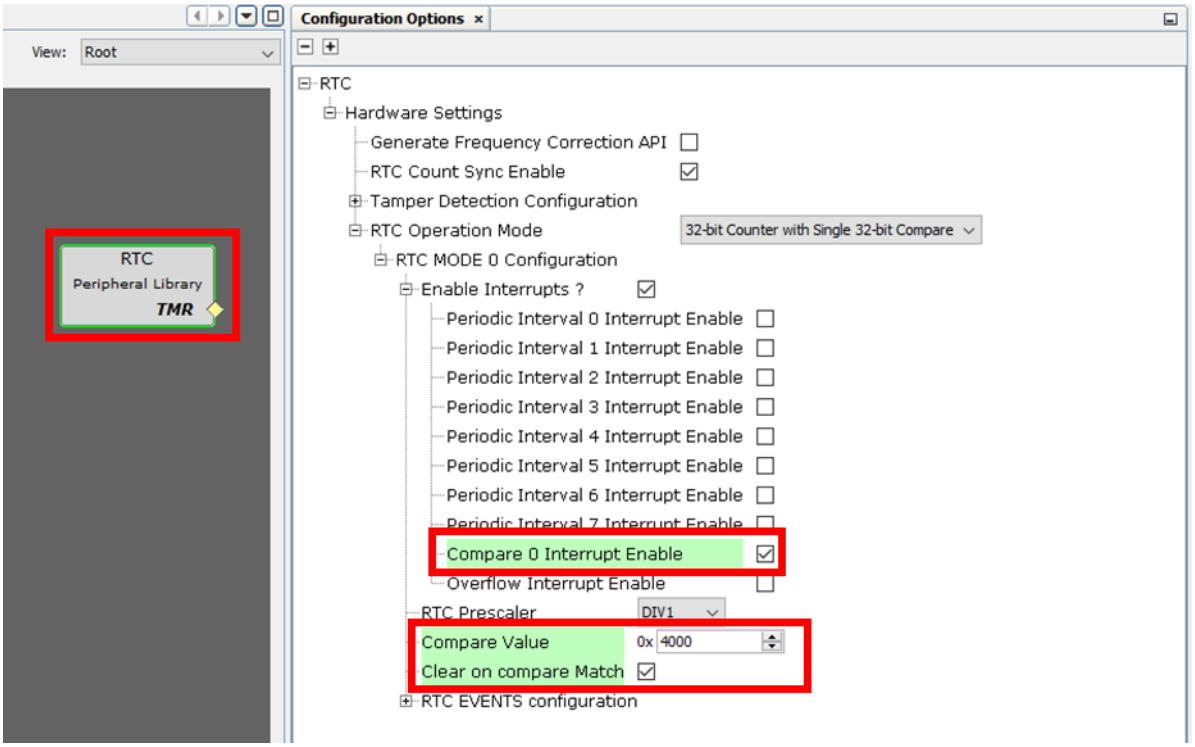
To optimize the power consumption, setup the RTC to run from the internal 32 kHz ultra-low power oscillator (OSCULP32K), which is always on.

For RTC clock selection, select the input oscillator as OSCULP32K to generate 32768 Hz clock output.

Click on the drop down menu to change the RTC clock source to OSCULP32K.



Step 4d: Go back to the project graph and configure the RTC PLIB to generate a compare interrupt once every 500 milliseconds.

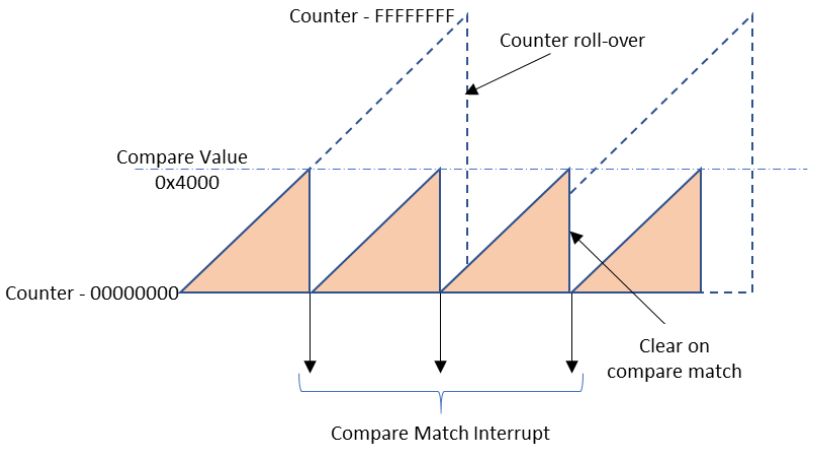


Note:

- The RTC can operate in either counter or calendar mode. By default, it is configured for 32-bit counter mode operation.
- The Compare Value is set as 0x4000. This compare value generates an RTC compare interrupt every 500 milliseconds.
  - RTC clock = 32768 Hz o RTC Prescaler = 1
  - Required Interrupt rate = 500 ms

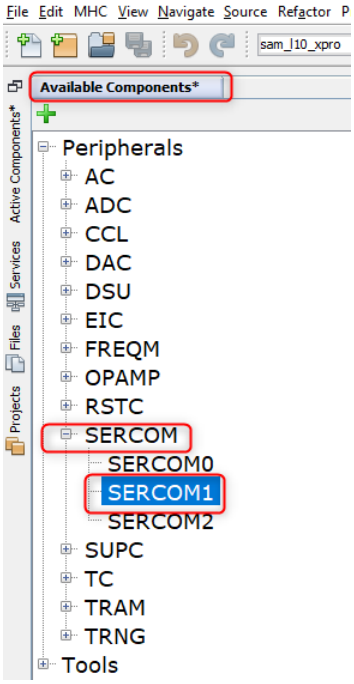
Hence, Compare Value = $(500 \times 10^{-3}) \times 32768 = 16384$ (0x4000)

- Clear on compare match – Clears counter on compare match, thereby configuring the RTC to generate periodic interrupts.

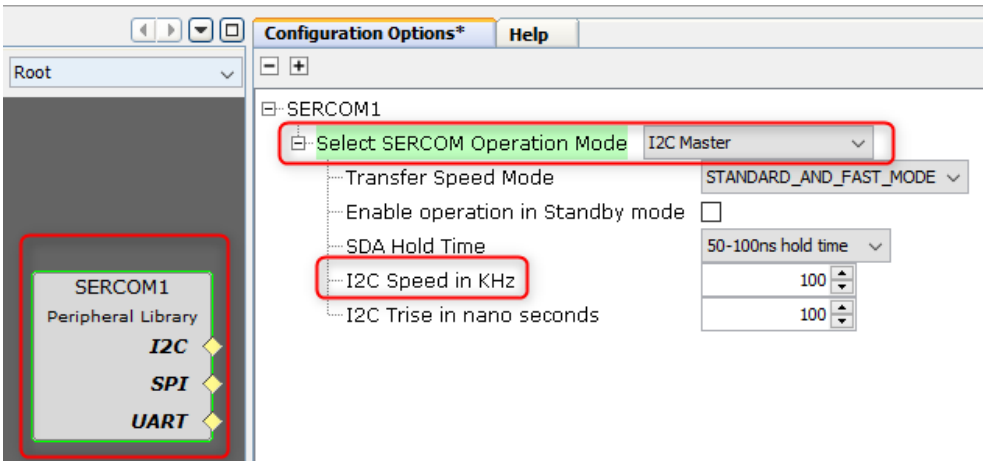


STEP 5: Configure I2C Peripheral Library and I2C pins

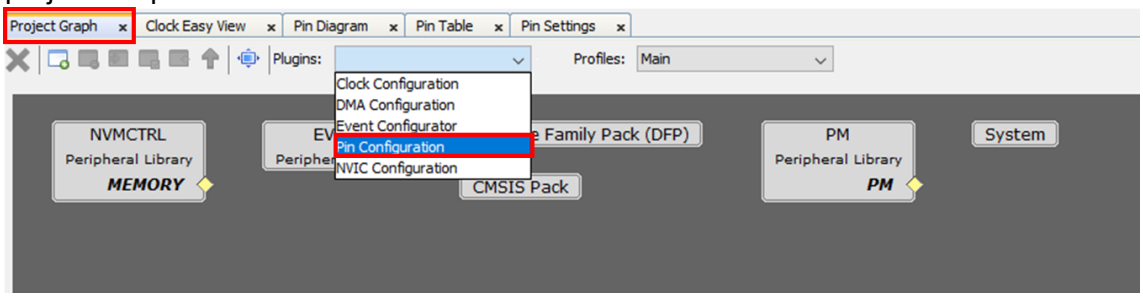
Step 5a: Under the bottom left tab "**Available Components**", Expand **Peripherals > SERCOM**
Select and double click on SERCOM1 to add the SERCOM instance 1 to the project.



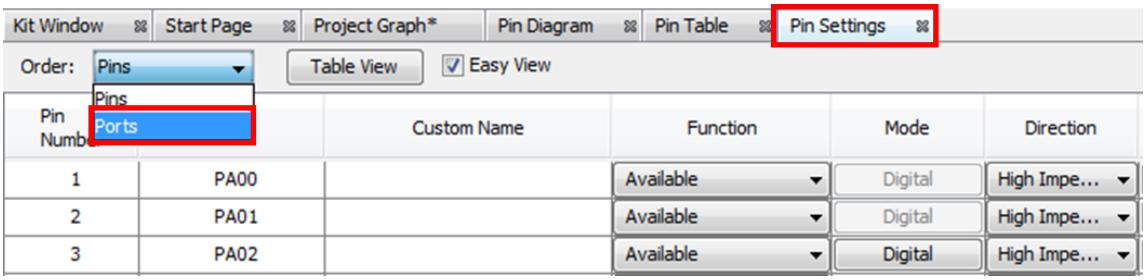
Select the SERCOM 1 Peripheral library and configure it for I2C Master protocol. Also, configure the I2C Speed to 100 kHz.



Step 5b: Open the Pin Configuration tabs by clicking **Plugins> Pin Configuration** in the project Graph



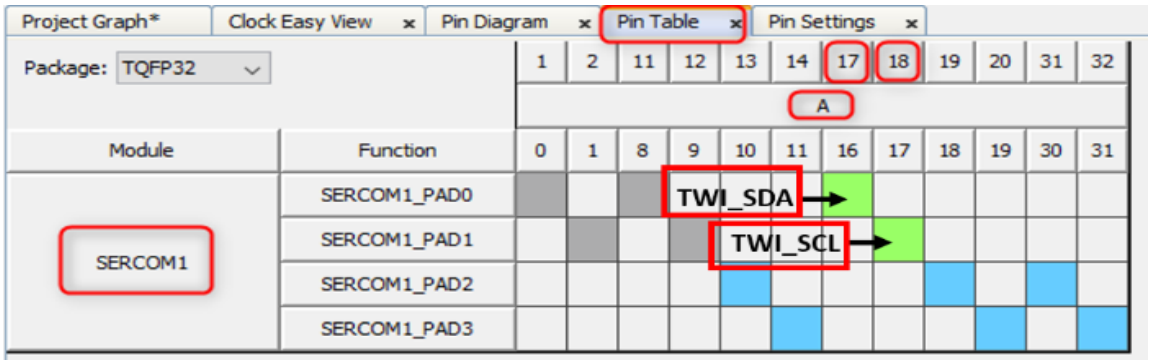
Step 5c: Select the **Pin Settings** tab and sort the entries by **Port** names as shown below.



Now, Select the **Pin Table** tab and then scroll down to the **SERCOM1** module as shown below.

a. Enable I2C Clock (**TWI_SCL**) on PA17 (Pin #18)

b. Enable I2C Data (**TWI_SDA**) on PA16 (Pin #17)

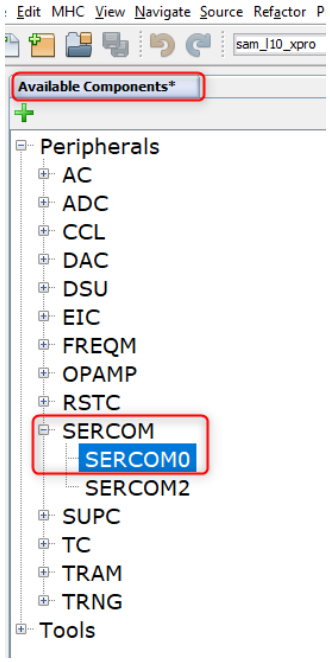


Note: The I2C pull-up resistors are mounted on the SAM L10 Xplained Pro board.
Note: The I2C pin configuration can also be done from the Pin Settings tab by selecting SERCOM1_PAD0 and SERCOM1_PAD1 function for pins PA16 and PA17 respectively.

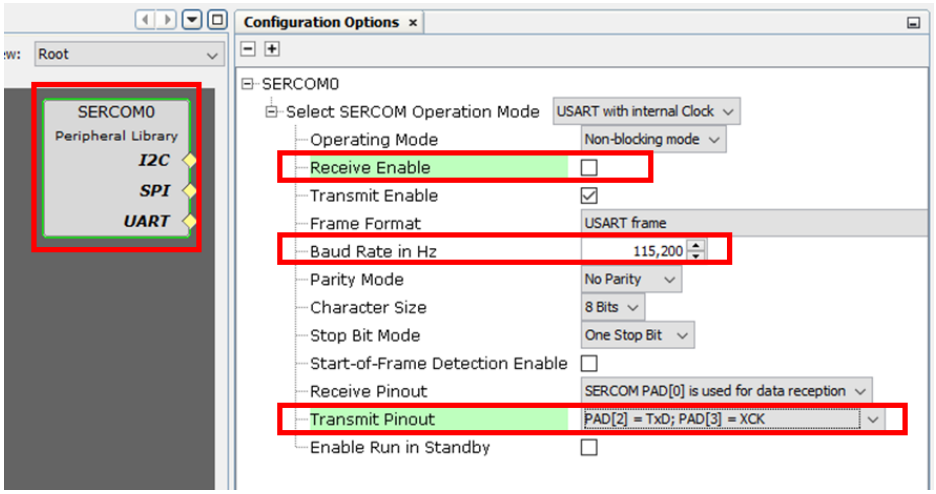
STEP 6: Configure USART Peripheral Library and USART pins

Step 6a: Under the bottom left tab "**Available Components**", Expand **Peripheral > SERCOM**

Select and double click on **SERCOM0** to add the SERCOM instance 0 to the project.



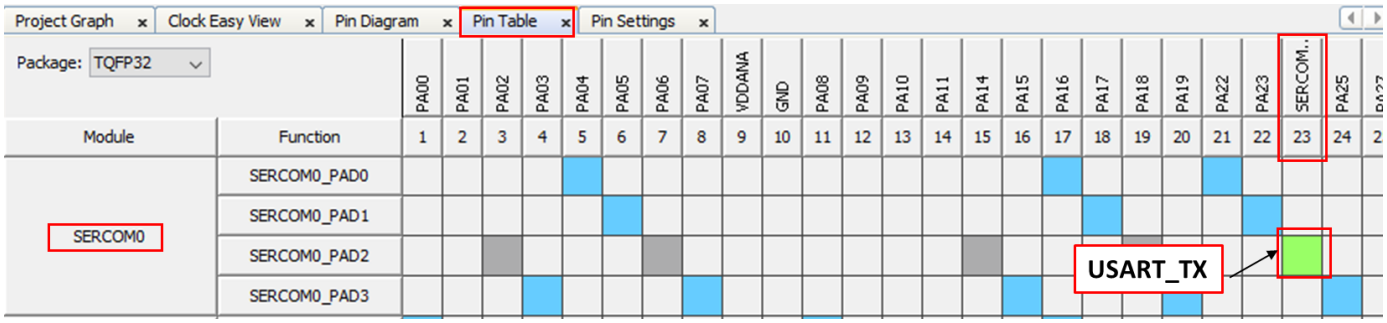
Select the SERCOM0 PLIB in the Project Graph and configure it for USART protocol as shown below.



Verify the Operating Mode is set to "Non-blocking mode" and the Baud Rate is set to 115200 Hz. Also, as per the "SAM L10 Xplained Pro Evaluation Kit" design, **SERCOM0 PAD2** is used for SERCOM0 (as USART) data transmission.

Note: The application will use the SERCOM0 USART PLIB for printing messages on the serial terminal. Hence, only the transmit functionality is enabled and the receive functionality is disabled.

Step 6b: Select the **Pin Table** tab and then scroll down to the **SERCOM0** module as shown below. Enable USART_TX (SERCOM0_PAD2) on **PA24** (Pin #23)



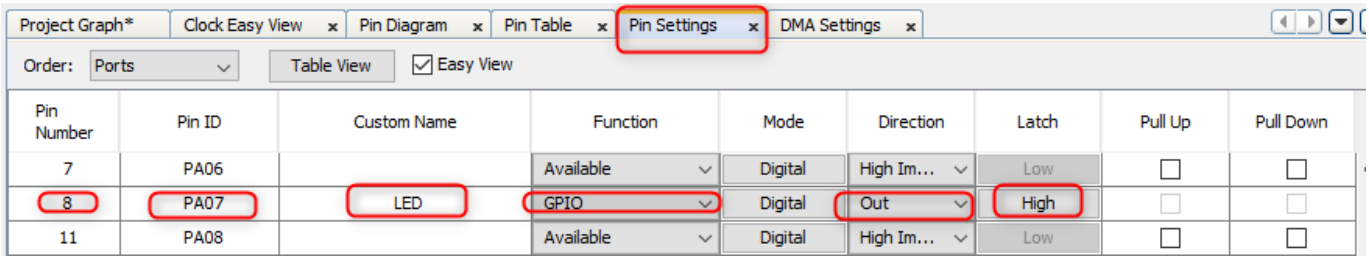
Note: In the SERCOM0 USART configuration, USART is enabled for transmit functionality alone. Hence, the USART receive pin is not configured. The IO pins may also be configured using the Pin Settings tab.

Note: The UART transmit pin can also be configured from the Pin Settings tab by selecting SERCOM0_PAD2 function for pin PA24.

STEP 7:Configure LED pin

Select the **Pin Settings** tab and then scroll down to the **Pin Number 8** and configure the PORT pin **PA07** as a GPIO output pin for LED functionality as shown below.

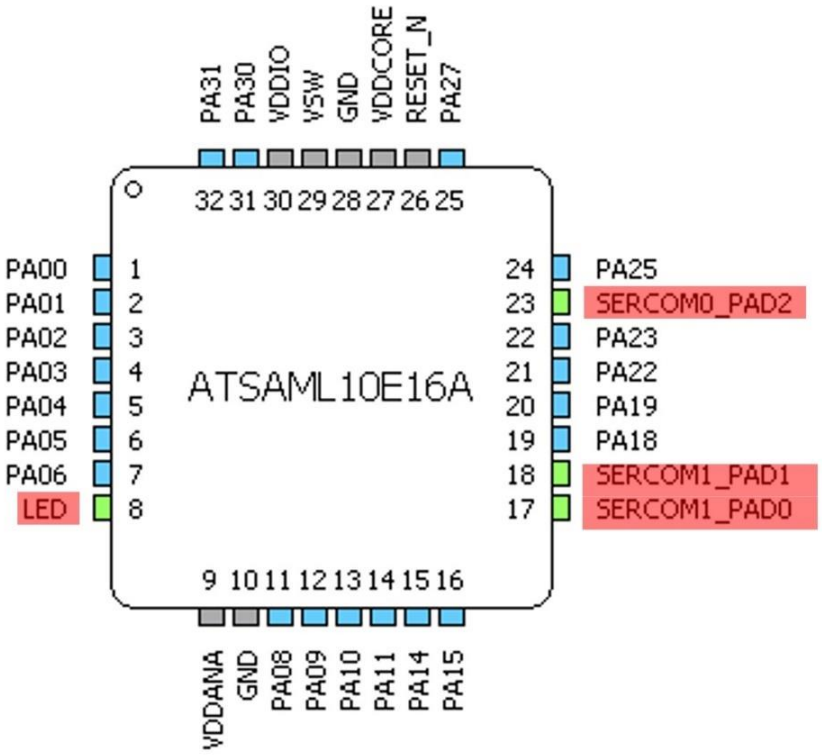
The LED on the SAM L10 Xplained Pro board is active low. Configure the LED in default OFF state by configuring the latch value to logic High.



Note that here you have provided a custom name “LED” to refer to the PA07 GPIO pin. Once the code is generated the following macros will be generated by the MCC:

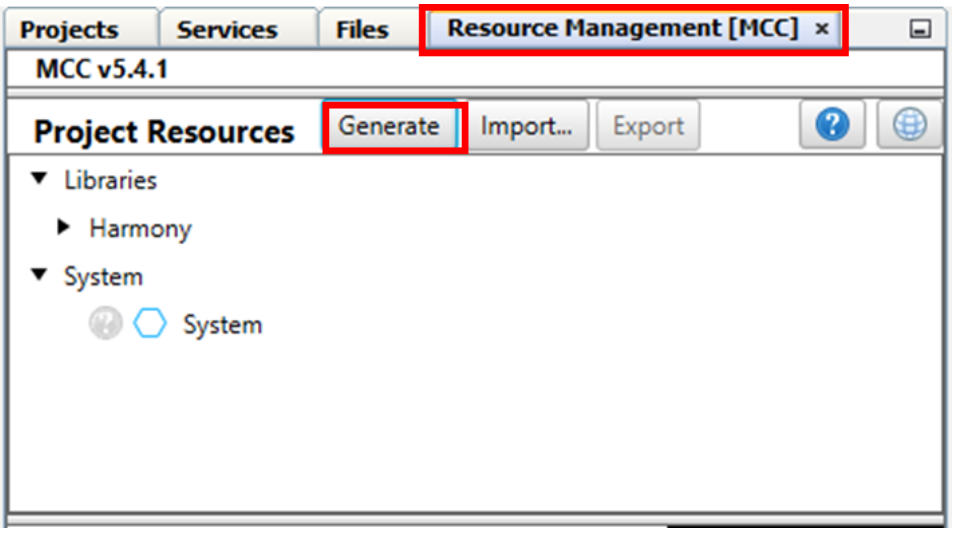
```
/*** Macros for LED pin ***/
#define LED_Set()               (PORT_REGS->GROUP[0].PORT_OUTSET = ((uint32_t)1U << 7U))
#define LED_Clear()             (PORT_REGS->GROUP[0].PORT_OUTCLR = ((uint32_t)1U << 7U))
#define LED_Toggle()            (PORT_REGS->GROUP[0].PORT_OUTTGL = ((uint32_t)1U << 7U))
#define LED_OutputEnable()      (PORT_REGS->GROUP[0].PORT_DIRSET = ((uint32_t)1U << 7U))
#define LED_InputEnable()       (PORT_REGS->GROUP[0].PORT_DIRCLR = ((uint32_t)1U << 7U))
#define LED_Get()               (((PORT_REGS->GROUP[0].PORT_IN >> 7U)) & 0x01U)
#define LED_PIN                  PORT_PIN_PA07
```

Note: All the pin assignments can be verified by selecting the Pin Diagram tab as shown below. All the assigned pins are indicated in Green (Assigned), while all the available pins are indicated in Blue color.



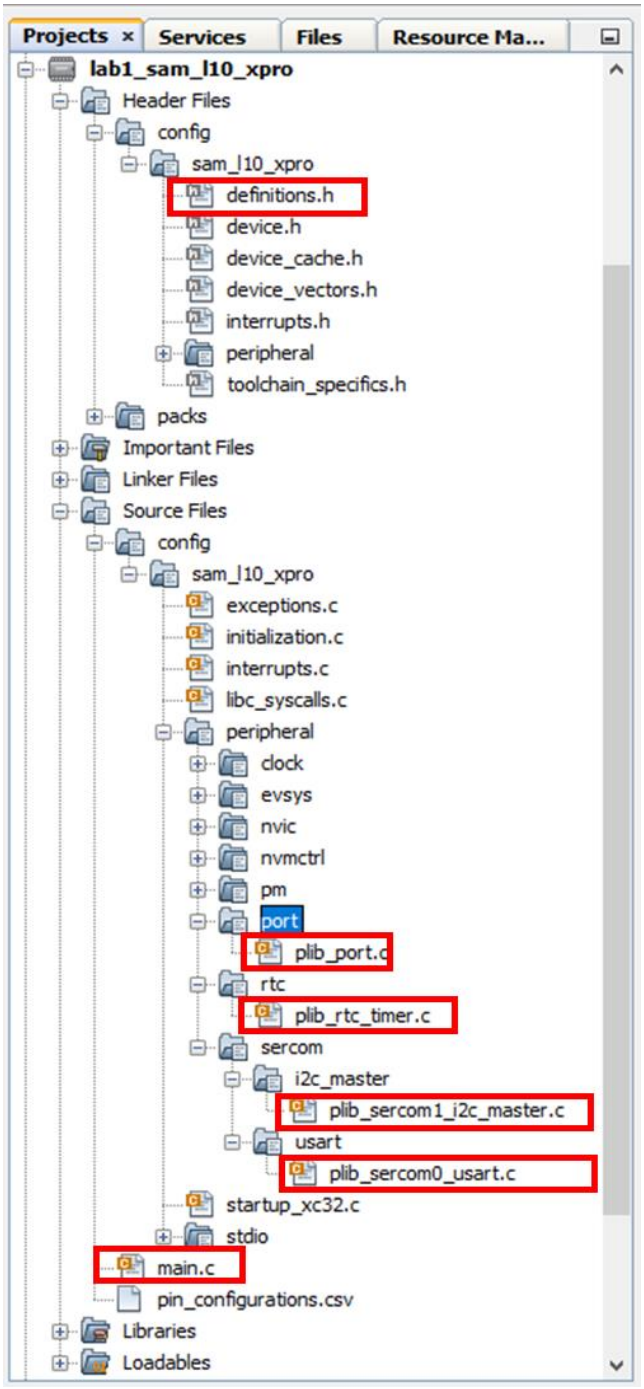
STEP 8: Generate code

Step 8a: Click on the **Generate** button in the Generate icon on Resouce management[MCC] window, as shown below.



Note: The MCC will include all the MPLAB® Harmony library files and generate the code based on the selections.

Note: The MCC will include all the MPLAB® Harmony library files and generate the code based on the MCC selections.
The generated code will add files and folders in your Harmony project as shown below.



Among the generated code notice the source and header files generated for RTC, SERCOM1-I2C and SERCOM0-USART and PORT peripherals. MCC also generates a template main.c file.

Note: Navigate to **Projects** tab to view the project tree structure.

Step 8c: Build the code by clicking on the "Clean and Build" icon  and verify the project builds successfully.
At this point you are ready to start implementing your application code.

PART 2: Add application code and build the application

STEP 9: Add application code to the project

Note: The application is already developed and is available in **main.c** file under **"help/saml10/lab1**" folder. The **main.c** file contains the application logic. It also contains empty event handlers and placeholders that you will populate with necessary code in the next step.

- Go to the "**lab_work/help/saml10/lab1**" folder and copy the pre-developed **main.c** file.
- Replace (over-write) the **main.c** file of your project available at "**<your-project-path>/ firmware/src**" with the copied file.
- Open **main.c** in MPLAB® IDE and add application code by following the below mentioned steps.

Tip: Search for the string **"Step #"** in **main.c** file to locate the position where the code snippets need to be enabled by un-commenting it.

Note: For all the labs, the code snippets have already been added to the application. You will enable it by un-commenting them. The purpose of this exercise is to understand how to use the peripheral libraries, register and handle callbacks and understand the overall application flow.

9a: Open **main.c** file. Register an event handler (callback) with the I2C PLIB. The event handler is called by the I2C PLIB when the I2C transfer is complete. The below API is used to register a callback:

```
void SERCOM1_I2C_CallbackRegister(
        SERCOM_I2C_CALLBACK callback,
        uintptr_t contextHandle
);
```

where, SERCOM_I2C_CALLBACK is a pointer to a function defined as:

```
typedef void (*SERCOM_I2C_CALLBACK) (uintptr_t contextHandle);
```

Search for "Step #1" and un-comment the below code snippet.

```
/*Register I@C Transfer complete event handler. Here ---> Step 1*/
SERCOM1_I2C_CallbackRegister(i2cEventHandler, 0);
```

Note: If the application does not want to use callbacks, it can always poll the status of the I2C transfer by calling the SERCOM1_I2C_IsBusy() API.

Step 9b: Register an event handler (callback) with the USART PLIB. The event handler is called by the USART PLIB when a USART write is complete.

Search for "Step #2" and un-comment the below code snippet.

```
/*Register USART Write Complete event handler. Here ---> Step 2*/
SERCOM0_USART_WriteCallbackRegister(usartTxHandler, 0);
```

Note: If the application does not want to use callbacks, it can always poll the status of the USART write operation by calling the SERCOM0_USART_WriteIsBusy() API.

In this lab, the callback event for USART write is not consumed. However, it could be used to indicate the completion of a write operation.

Step 9c: Register an RTC handler (callback) with the RTC PLIB. The event handler is called by the RTC PLIB when 500 milliseconds time period has elapsed. This will be used by the application to read the temperature every 500 milliseconds.

Search for "Step #3" and un-comment the below code snippe

```
/*Register RTC Compare Match event handler. Here ---> Step 3*/
RTC_Timer32CallbackRegister(rtcEventHandler, 0);
```

9d: Start the RTC timer. The RTC PLIB will call the event-handler (registered in the step 3 above) when a period of 500 milliseconds elapses.

Search for "Step #4" and un-comment the below code snippet.

```
/* Start RTC timer. Here -----> Step #4 */
RTC_Timer32Start();
```

Step 9e: In the RTC event handler (**rtcEventHandler**), set a flag to indicate that the compare match event (a period of 500 milliseconds) has occurred.

Search for "Step #5" and un-comment the below code snippet.

```
static void rtcEventHandler (RTC_TIMER32_INT_MASK intCause, uintptr_t context)
{
    if (intCause & RTC_TIMER32_INT_MASK_COMPARE_MATCH)
    {
        /* RTC timer compare match. Here -----> Step #5 */
        isRTCTimerExpired = true;
    }
}
```

Step 9f: Submit an I2C transfer to read temperature sensor value when a time period of 500 milliseconds has elapsed. The I2C PLIB calls back the event-handler (registered in the step 1 above) when the submitted request is complete.

The application reads two bytes of temperature data from the address location 0x00.

The SERCOM1 I2C Write-Read API is given below:

```
bool SERCOM1_I2C_WriteRead(
        uint16_t address,
        uint8_t* wrData,
        uint32_t wrLength,
        uint8_t* rdData,
        uint32_t rdLength
);
```

Search for "Step #6" and un-comment the below code snippet.

```
if(isRTCTimerExpired == true)
{
    isRTCTimerExpired=false;

    /*Submit I2c transfer to read temperature sensor value Here ---> Step 6*/
    status=SERCOM1_I2C_WriteRead(TEMP_SENSOR_SLAVE_ADDR, &i2cWrData, 1, i2cRdData, 2);

    if(status==false)
    {
        /*Handle error condition*/
    }
}
```

Step 9g: In the I2C event handler (**i2cEventHandler**), set a flag to indicate that the I2C read (of the temperature sensor value) is complete.

Search for "Step #7" and un-comment the below code snippet.

```
static void i2cEventHandler(uintptr_t contextHandle)
{
    if (SERCOM1_I2C_ErrorGet() == SERCOM_I2C_ERROR_NONE)
    {
        /*I2c read complete. Here ---> Step #7*/
        isTemperatureRead = true;
    }
}
```

Step 9h: Add code to transfer the latest temperature value over USART to be printed on the serial terminal.

Search for "Step #8" and un-comment the below code snippet.

```
temperatureVal = getTemperature(i2cRdData);

sprintf((char*)uartTxBuffer, "Temperature = %02d F\r\n", temperatureVal);

/*Submit USART write to display the latest temperature value and
 * toggle LE same time. Here --->Step #8  */
status=SERCOM0_USART_Write(uartTxBuffer,strlen((const char*)uartTxBuffer));
```

Step 9i: Also add code to toggle the user LED (LED0) every time the temperature value is printed on the serial terminal.

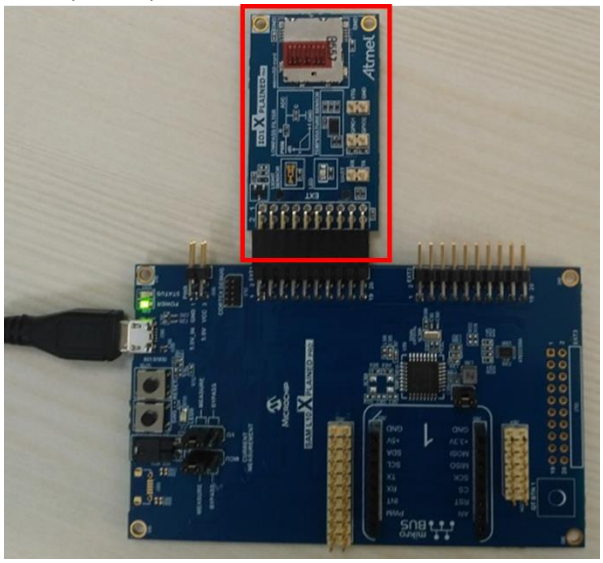
Search for "Step #9" and un-comment the below code snippet.

```
/* Toggle LED. Here -----> Step #9 */
LED_Toggle();
```

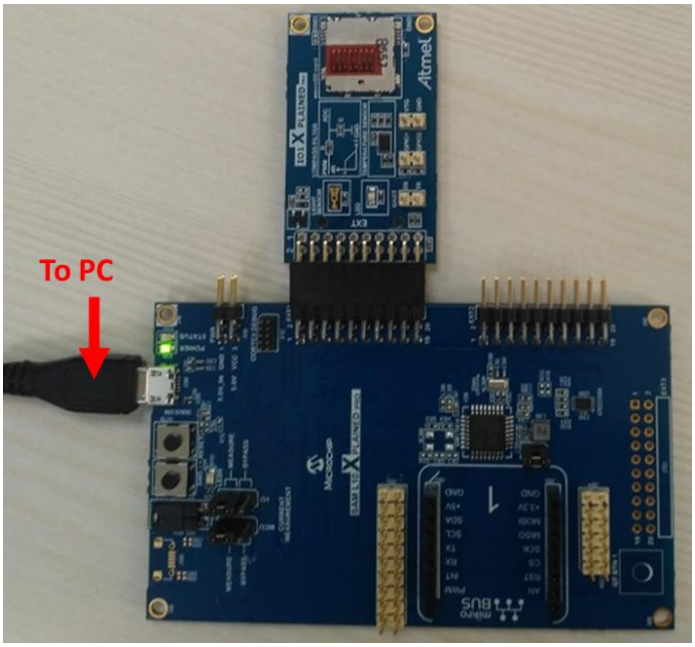
Congratulations! You're almost done. You are ready to build and run your first application.

STEP 10: Build and Run the Application

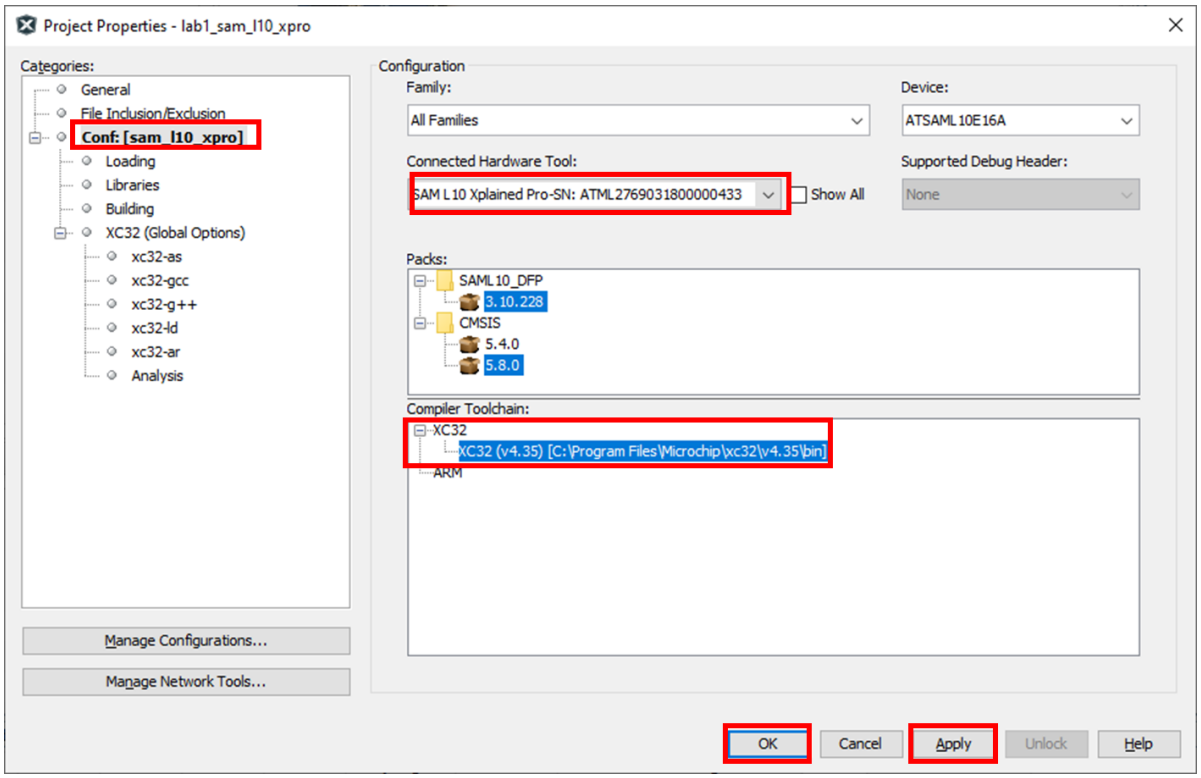
Step 10a: Verify that the temperature sensor (**I/O1 Xplained Pro Extension Kit**) is connected to Extension Header 1 (EXT1) on the **SAM L10 Xplained Pro Evaluation Kit**.



Step 10b: The **SAM L10 Xplained Pro Evaluation Kit** allows using the EDBG (Embedded Debugger) for debugging. Connect the Type-A male to micro-B USB cable to DEBUG USB port to power and debug the **SAM L10 Xplained Pro Evaluation Kit**.



Step 10c: Go to **File > Project Properties** and make sure that the EDBG is selected as the debugger under the **Hardware Tools** and XC32 (v2.50) is selected as the **Compiler Toolchain**.



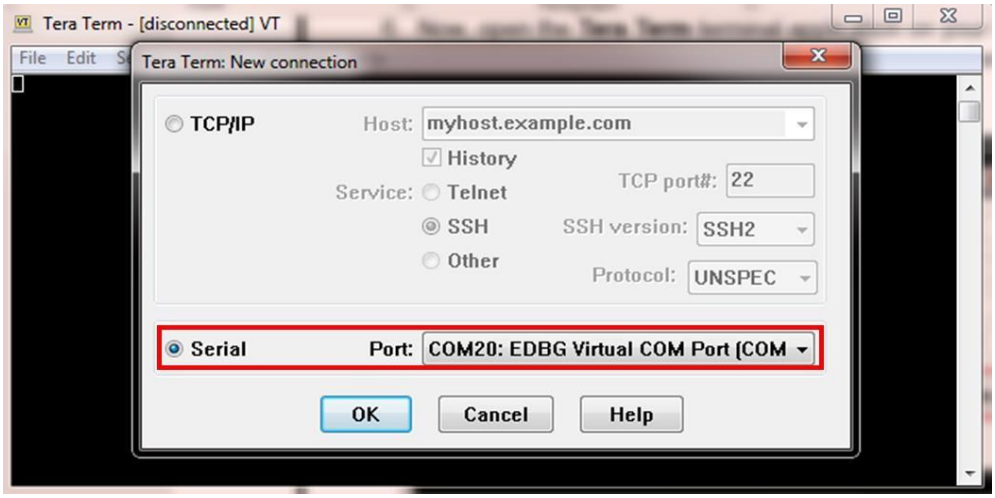
Step 10d: Clean and build your application by clicking on the “Clean and Build” button as shown below.

Step 10e: Program your application to the device, by clicking on the “Make and Program” button as shown below.

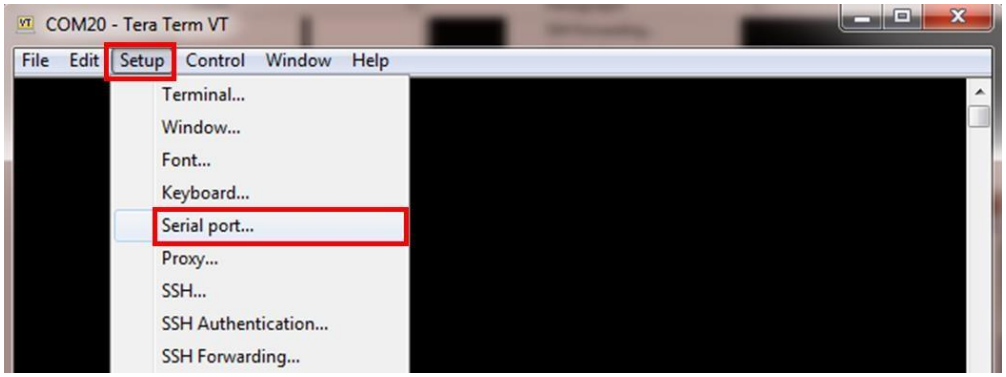
The Lab should build and program successfully.

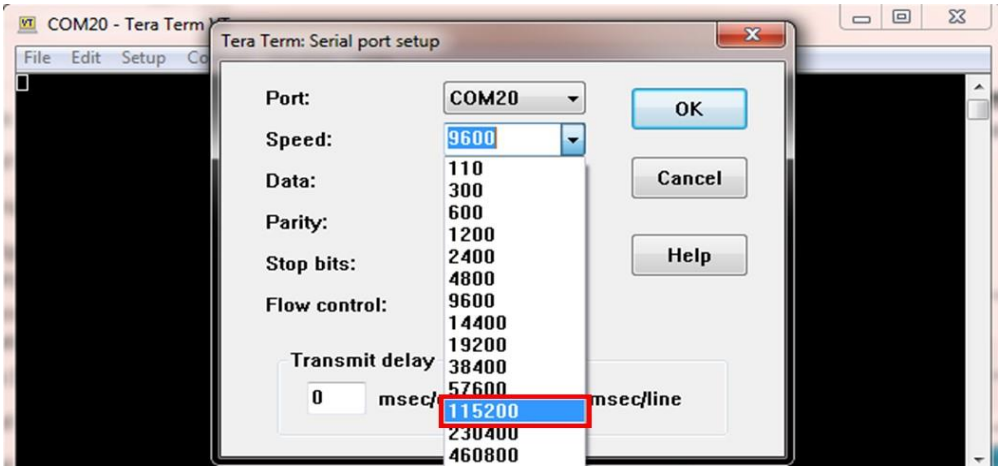
Results:

- Open the **Tera Term** terminal application on your PC (from the windows Start menu by pressing the Start button). Select the EDBG Virtual COM Port as shown below.

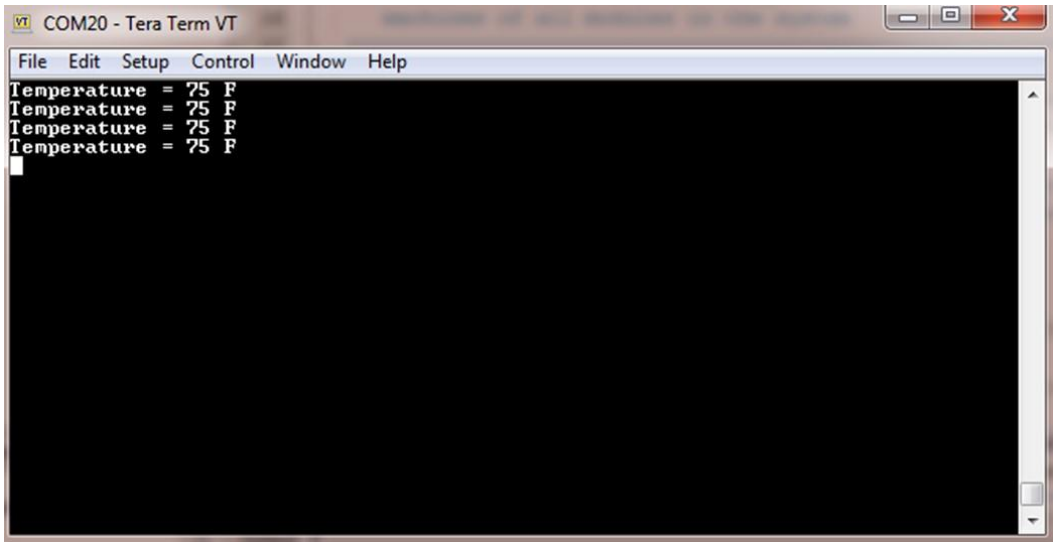


- Change the baud rate to 115200



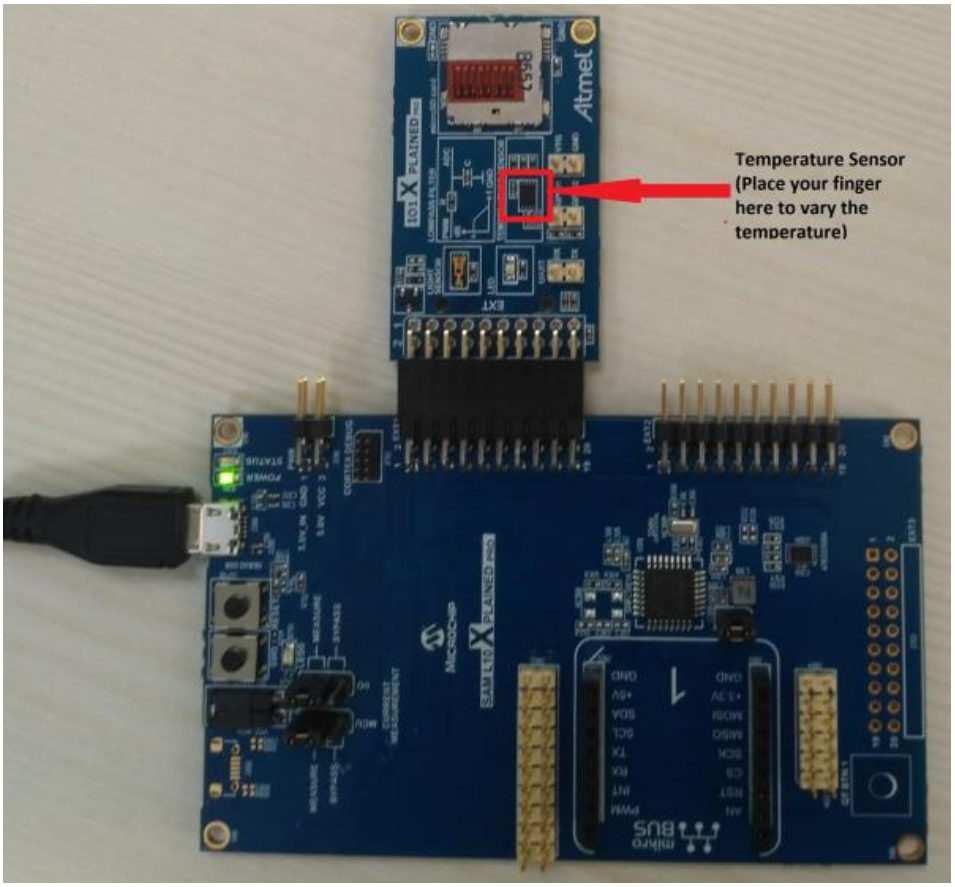


- You should see the temperature values (in °F) being printed on the terminal every 500 milliseconds, as shown below.



- Also notice the LED0 blinking at 500 millisecond rate.

- You may vary the temperature by placing your finger on the temperature sensor (for a few seconds).



Summary:

Using MPLAB® Harmony, you were able to create a sensor based application to periodically read temperature from a temperature sensor.

The RTC PLIB was configured to generate an event once every 500 milliseconds.

The SERCOM1-I2C PLIB was used to read temperature from the temperature sensor.
The I2C was configured (by default) for master mode operation at 100 kHz baud.

The SERCOM0-USART PLIB was used to print temperature values on the serial terminal. The SERCOM0-USART PLIB was configured (by default) for 115200 baud.

In the process you got a hands-on experience on various MCC configurators like Clock Configurator, Pin Configurator and how to add and configure different peripheral libraries, generate code and develop an application.

In the next lab, you will realize the existing application to read temperature periodically using USART with DMA.

Lab 2 - Use DMA to print temperature sensor data on serial terminal

Purpose:

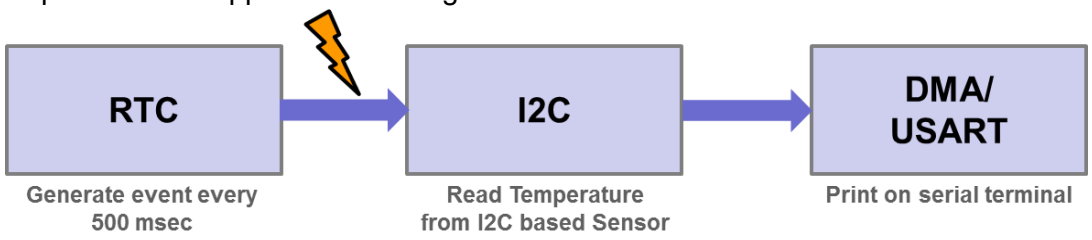
After completing Lab 2, you will understand how to configure and use the DMA PLIB.

Overview:

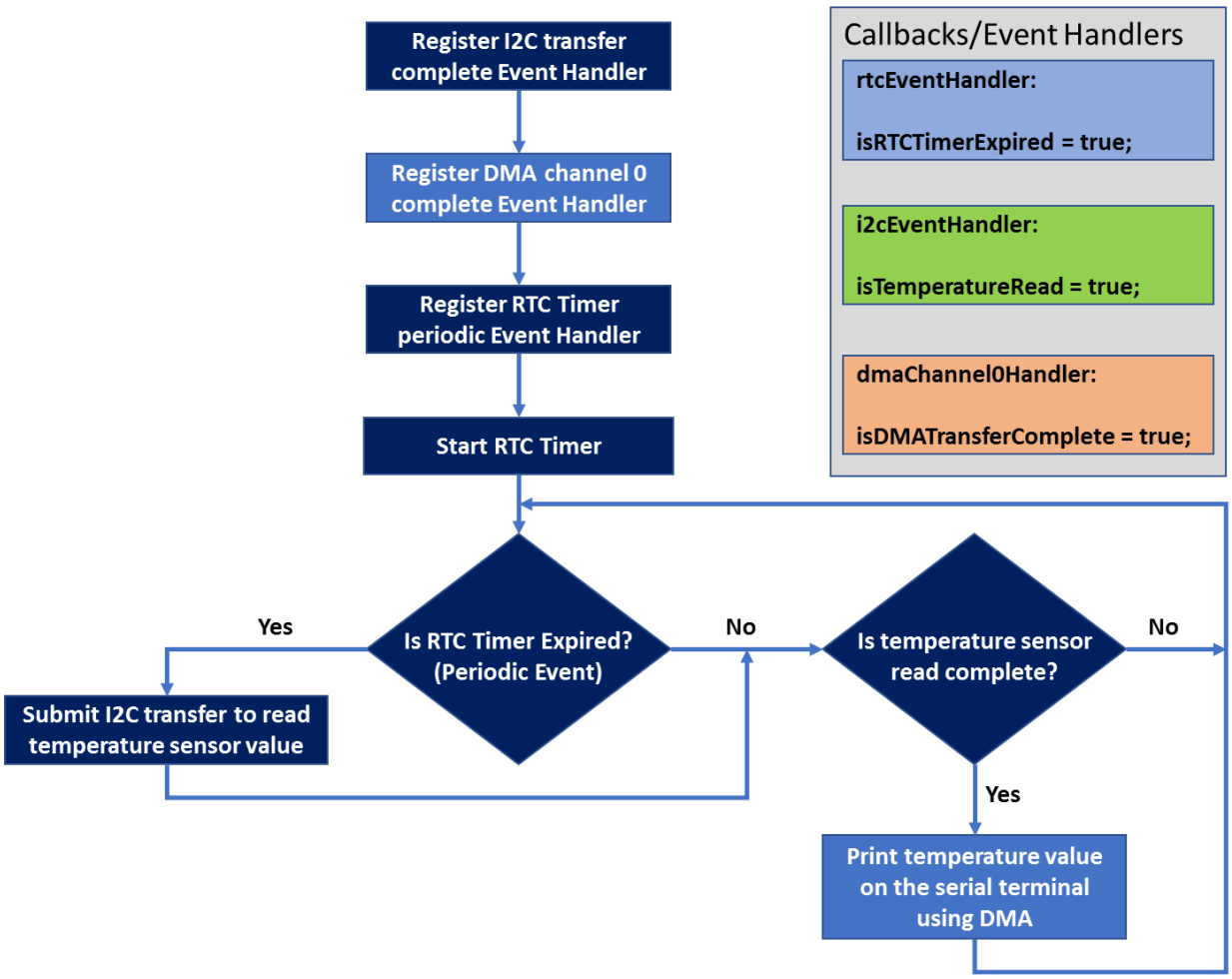
In this lab you will use the DMA peripheral library to realize Lab 1.

The application uses:

- RTC Peripheral Library to generate periodic notifications
- SERCOM-I2C Peripheral Library to read temperature from a temperature sensor
- SERCOM-USART and DMA Peripheral Libraries to print the temperature values on a serial port terminal application running on a PC.



Flowchart


Register I2C transfer complete Event Handler
Register DMA channel 0 complete Event Handler
Register RTC Timer periodic Event Handler
Start RTC Timer
Callbacks/Event Handlers
rtcEventHandler:
isRTCTimerExpired = true;
i2cEventHandler:
isTemperatureRead = true;
dmaChannel0Handler:
isDMATransferComplete = true;
Is RTC Timer Expired? (Periodic Event)
Yes
No
Submit I2C transfer to read temperature sensor value
Is temperature sensor read complete?
No
Yes
Print temperature value on the serial terminal using DMA

Procedure:

This lab is completed in the following steps:
Part 1: Configure DMA Peripheral Library and disable USART interrupt
STEP 1: Configure DMA
STEP 2: Disable USART interrupt
STEP 3: Generate code
Part 2: Add application code and build the application
STEP 4: Add code to your Application
STEP 5: Build and Run the Application

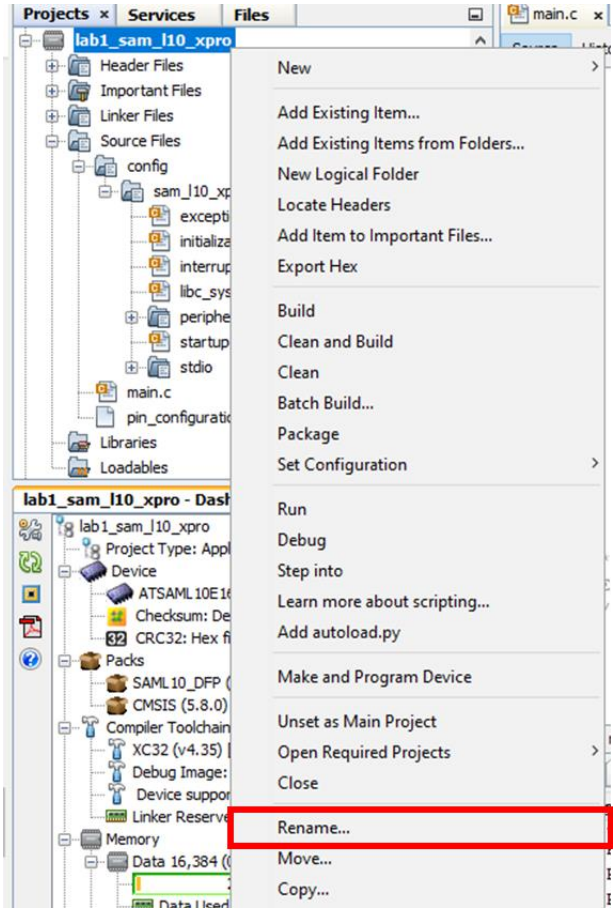
Note: You will develop Lab2 by extending the Lab1. If for some reason you were unable to complete the Lab1 successfully, then follow the below procedure (***otherwise, skip these steps***):

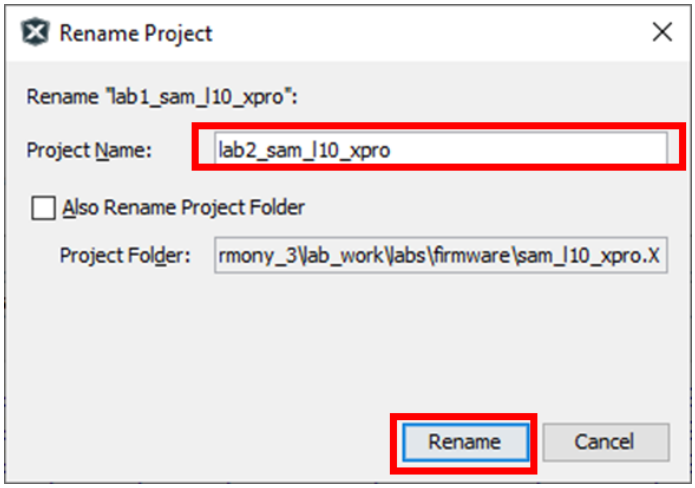
1. Close the Lab1 project (**lab1_sam_l10_xpro**) in MPLAB® X IDE by going to **File>Close All Projects**
2. Delete the **firmware** folder created by you at **<your-project-path>** location. For example, for the path used in this lab manual, the firmware folder will be under **/lab_work/labs** folder.
3. Copy the ready-made solution for the Lab1 available under the **lab_work/solutions/saml10/lab1** folder and paste the solution (**firmware** folder) to the **<your-project-path>** location**.** For example, for the path used in this lab manual, paste the **firmware** folder under **lab_work/labs** folder.
4. Drag and drop the MPLAB X project file (**sam_l10_xpro.X**) available under **<your-project-path>/firmware/** to the MPLAB X IDE
5. In MPLAB X IDE, right click on the project and set it as the main project.
6. In MPLAB X IDE, go to **Tools > Embedded** and open the **MPLAB Harmony 3 Configurator**.

Tool Tip:- If you closed MHC accidently, and would like to open, refer section **Appendix A – Launching MCC**

Part 1: Configure DMA Peripheral Library and disable USART interrupt

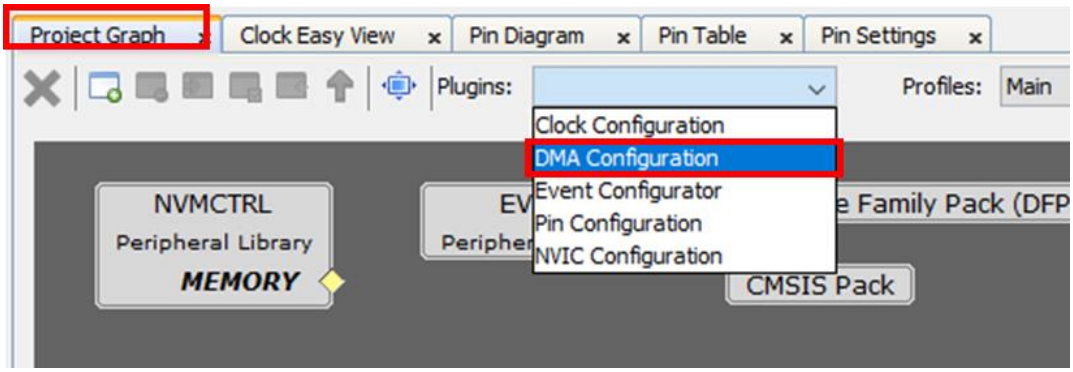
Since you are working on Lab2, change the name of the project under project tree from **lab1_sam_l10_xpro** to **lab2_sam_l10_xpro.**
Click on the “Projects” tab on the top left pane. Right click on the project name “**lab1_sam_l10_xpro**” and click on “**Rename…**”.





STEP 1: Configure DMA

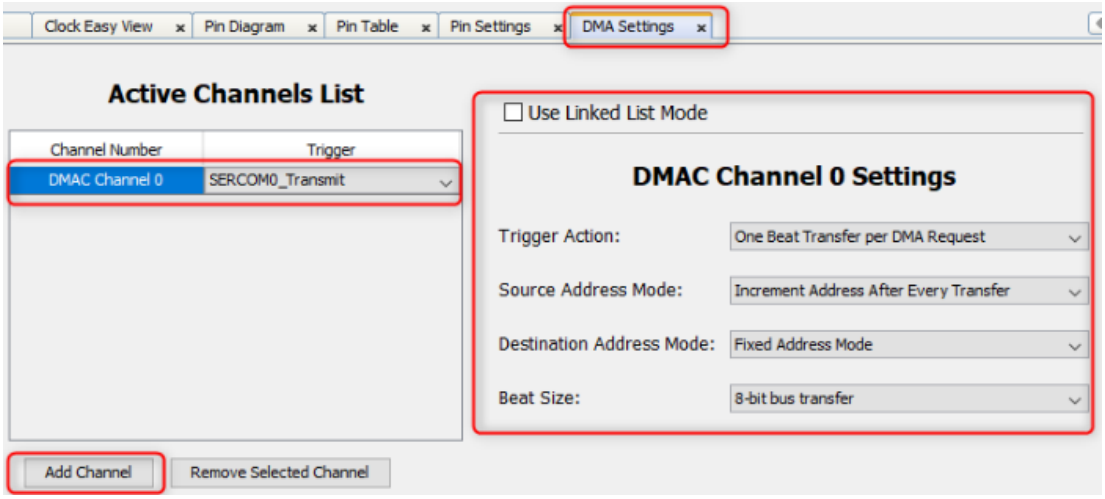
Step 1a: Launch DMA Configurator by going to **Project Graph** tab in MPLABX IDE and then selecting **Plugins > DMA Configuration**.



Step 1b: Click on the **DMA Settings** tab.

Click on *Add Channel* to add DMA Channel 0. Set the *Trigger* source to SERCOM0_Transmit.

This configures the DMA Channel 0 to transfer application buffer to USART TX register. The DMA transfers one byte from application buffer to USART transmit buffer on each trigger.



Based on the Trigger source, the DMA channel configuration is automatically set by the MCC.

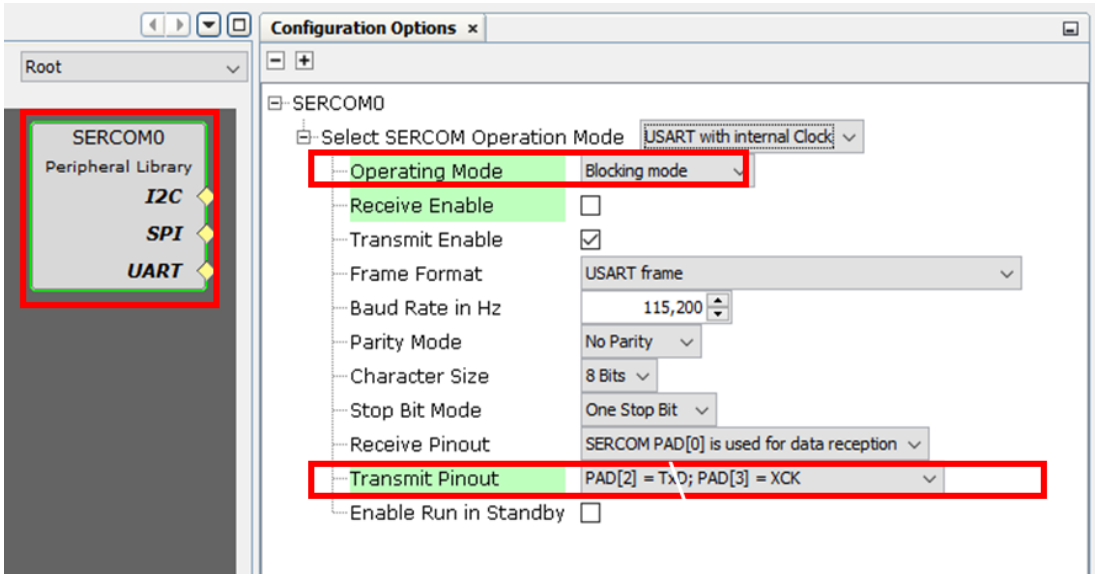
- Trigger Action – Action taken by DMA on receiving a trigger o One beat transfer – can be used during a memory to peripheral or peripheral to memory transfer. o One block transfer – can be used during memory to memory transfer on a software trigger.
  - o One transaction transfer – can used for memory to memory transfer with linked list to copy multiple blocks on a software trigger.

- Source Address Mode, Destination Address Mode –
  Select whether to increment Source/Destination Address after every transfer. Automatically set by MCC based on the trigger type.
  Example:
  - If the trigger source is USART transmit, then the Source Address is incremented, and the Destination Address is fixed
  - If the trigger source is USART receive, then the Source Address is fixed, and the Destination Address is incremented

- Beat Size – Size of one beat. The default value is 8-bits.
  Example:
  - o If the SPI peripheral is configured for 16-bit/32-bit mode, then the beat size must be set to 16-bits/32-bits respectively

STEP 2: Disable USART interrupt

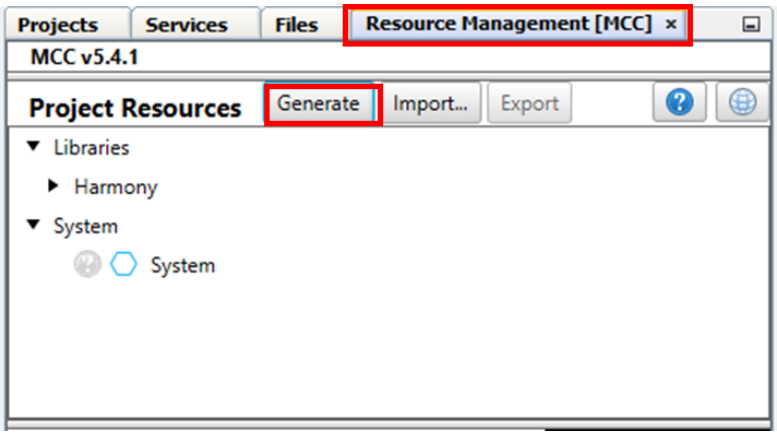
Under the **Project Graph** tab, click on the **SERCOM0** block. In the **Configuration Options**, select the Operating Mode to **Blocking mode**.



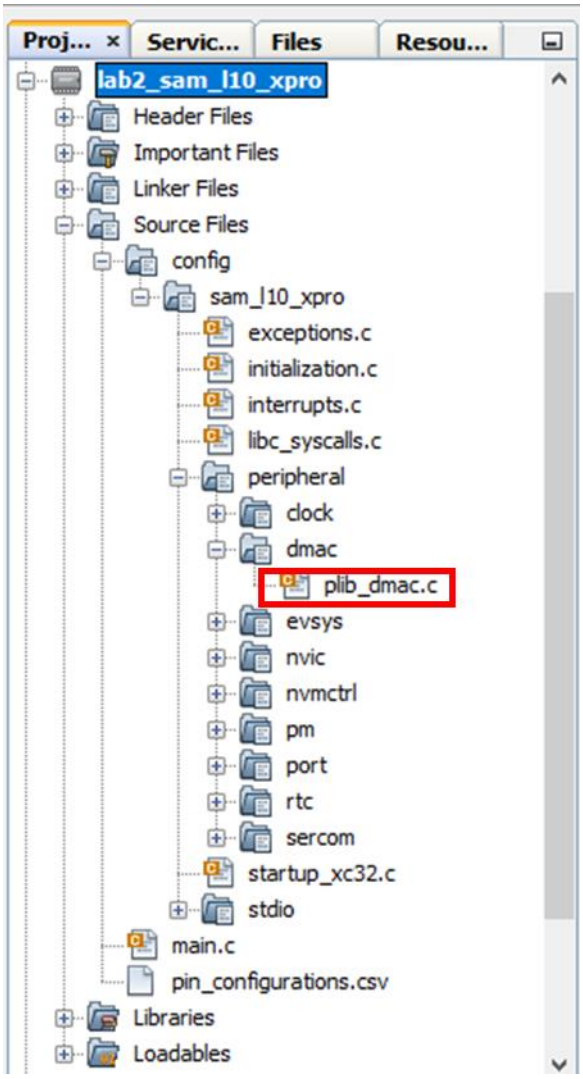
Note: In this lab, the SERCOM0-USART interrupt is disabled as the application does not need a callback on USART transfer complete. A USART transmit buffer empty event triggers DMA to transfer one byte of data from source (user buffer) to destination (USART TX register). When all the requested bytes are transmitted, DMA PLIB notifies the application by calling the registered DMA event handler.

STEP 3: Generate code

Step 3a: Click on the **Generate** button in the Generate icon on Resouce management[MCC] window, as shown below.



Note: The MCC will include all the MPLAB® Harmony library files and generate the code based on the selections.



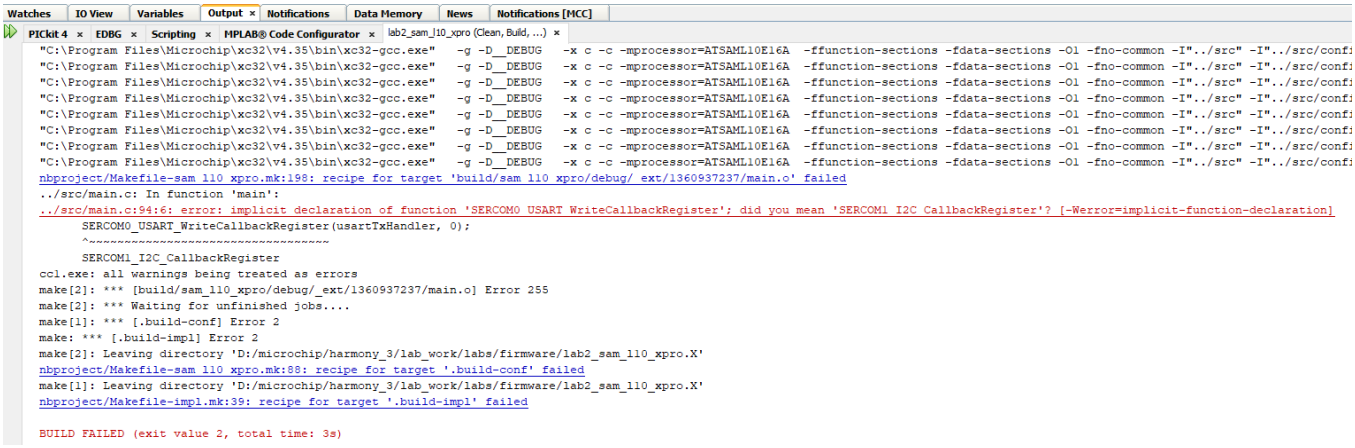
The generated code would add files and folders as shown below to your Harmony project.

In the generated code, notice the files generated for DMA Peripheral Library (**plib_dmac.c)**

Note: Navigate to the **Projects** tab to view the project tree structure.

Step 3b: Build the code by clicking on the "Clean and Build" icon  .

The build result will throw the following error:



The error seen in the application file (main.c developed in Lab1) is because the USART interrupt is disabled. The Harmony v3 configurator generates code only for the configured options. Since in this lab, the USART transmit interrupt is disabled the corresponding code (APIs and macros) to register USART transmit callback is not generated, resulting in build failure.

This build error will be resolved after modifying the application code, in the next steps.

Part 2: Add application code and build the application

STEP 4: Add code to your Application

Note: Like Lab1, the application is already developed and is available in **main.c** file under **"lab_work/help/saml10/lab2**" folder.

Replace the **main.c** file in your project, with the pre-developed file:

- ✓ Go to the "**lab_work/help/saml10/lab2**" folder and copy the pre-developed file **main.c**.
- ✓ Over-write the **main.c** file of your project with the pre-developed file.

Open **main.c** and add application code by following the below steps.

Note: For this lab the code snippets have already been added to the application. You will enable it by un-commenting them. The purpose of this exercise is to understand how easy it is to modify an application and implement it using DMA.

Tip: Search for the string **"Step #"** in **main.c** file to locate the position where the code snippets need to be enabled by un-commenting it.

Step 4a: Open **main.c** file. Register an event handler (callback) with the DMA PLIB. The event handler is called by the DMA PLIB when the DMA transfer (of temperature sensor data to serial terminal) is complete.

The below API is used to register a callback with the DMA PLIB:

```
void DMAC_ChannelCallbackRegister(
    DMAC_CHANNEL channel,
    const DMAC_CHANNEL_CALLBACK callback,
    const uintptr_t context
);
```

where, DMAC_CHANNEL_CALLBACK is a pointer to a function of type:

```
typedef void (*DMAC_CHANNEL_CALLBACK) (DMAC_TRANSFER_EVENT event, uintptr_t contextHandle);
```

Search for "Step #1" and un-comment the below code snippet.

```
/*Register DMA Channel 0 transfer complete event handler. Here---> Step #1*/
DMAC_ChannelCallbackRegister(DMAC_CHANNEL_0,dmaChannel0Handler,0);
```

Note: In this lab, the callback event for DMAC is not consumed. However, it could be used to indicate the completion of the DMA operation.

Step 4b: Add code to transfer the buffer containing the latest temperature value in the format "Temperature = XX F\r\n", over USART using DMA.

The below API is used to initiate a DMA transfer:

```
bool DMAC_ChannelTransfer(
    DMAC_CHANNEL channel,
    const void* srcAddr,
    const void* destAddr,
    size_t blockSize
);
```

Search for "Step #2" and un-comment the below code snippet.

```
/*Submit DMA transfer from memory to USART to display the latest
 *temperature value and toggle the LED at the same time. Here ---> Step #2 */
status=DMAC_ChannelTransfer(
        DMAC_CHANNEL_0,
        uartTxBuffer,
        (const void *)&(SERCOM0_REGS->USART_INT.SERCOM_DATA),
        strlen((const char*)uartTxBuffer)
);
```

Step 4c: Add code to implement the DMA channel 0 event handler. The event handler is called when the DMA channel 0 transfer completes.

Search for "Step #3" and un-comment the below code snippet.

```
/*DMA Channel 0 event handler. Here ---> Step #3  */
static void dmaChannel0Handler(DMAC_TRANSFER_EVENT event, uintptr_t contextHandle)
{
    if (event == DMAC_TRANSFER_EVENT_COMPLETE)
    {
        isDMATransferComplete = true;
    }
}
```

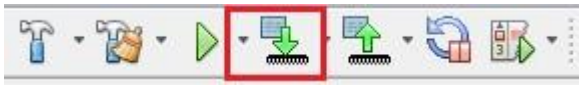
Congratulations! You are ready to build and run your second application.

Note: The **main.c** file for this lab is same as that of the previous lab (Lab1) except for the changes covered in the 3 steps shown above.

STEP 5: Build and Run the Application

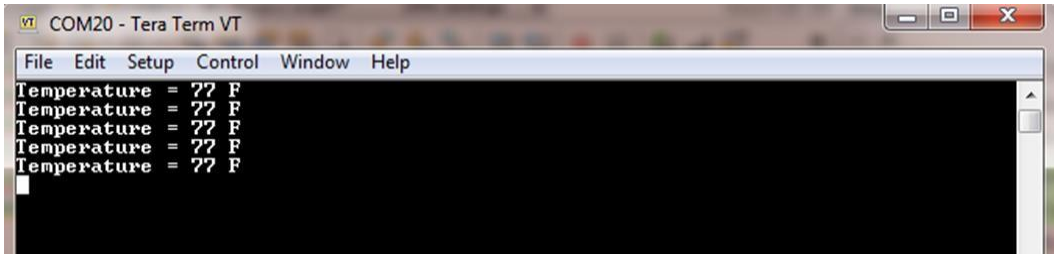
Step 5a: Clean and build your application by clicking on the “Clean and Build” button as shown below.

Step 5b: Program your application to the device, by clicking on the “Make and Program” button as shown below.



Results:

- Open the Tera Term terminal application on your PC. (If already opened, then close the previous session of Tera Term)
- Once opened, the terminal should print the temperature values every 500 milliseconds.



- Also notice the LED0 blinking at every 500-millisecond interval.

Summary:

Using MPLAB® Harmony, you were able to add DMA functionality to your existing project. The new application uses DMA PLIB to transfer data from the user buffer to USART. In the process you learnt how to add and configure DMA channels using the DMA configurator.

In the next lab, you will add ADC PLIB and learn how to configure and use the Event System peripheral. You will also put the CPU in sleep mode to optimize power consumption.

Lab 3 - Periodically print temperature sensor data on serial terminal when the light sensor is covered (by placing a hand over it). Enter sleep mode when the light sensor is not covered.

Purpose:

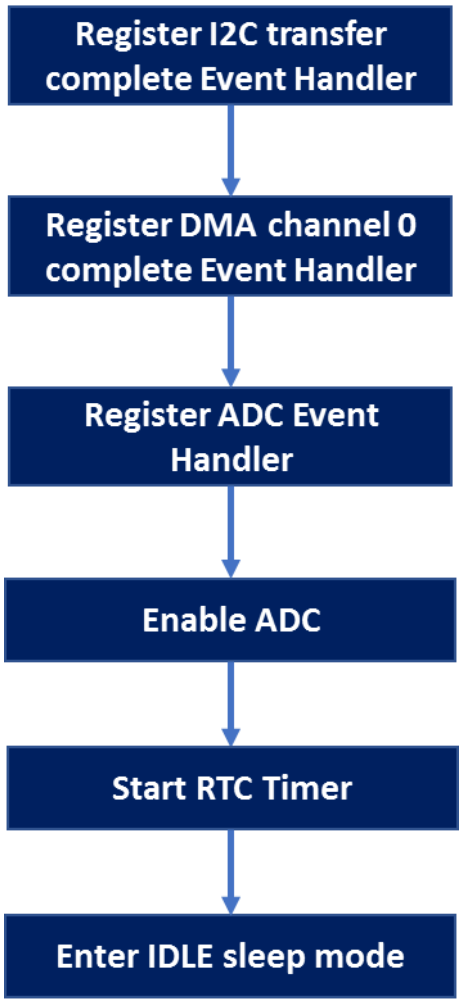
After completing Lab 3, you will learn how to add and configure the ADC, Event System and Power Management peripheral libraries.

Overview:

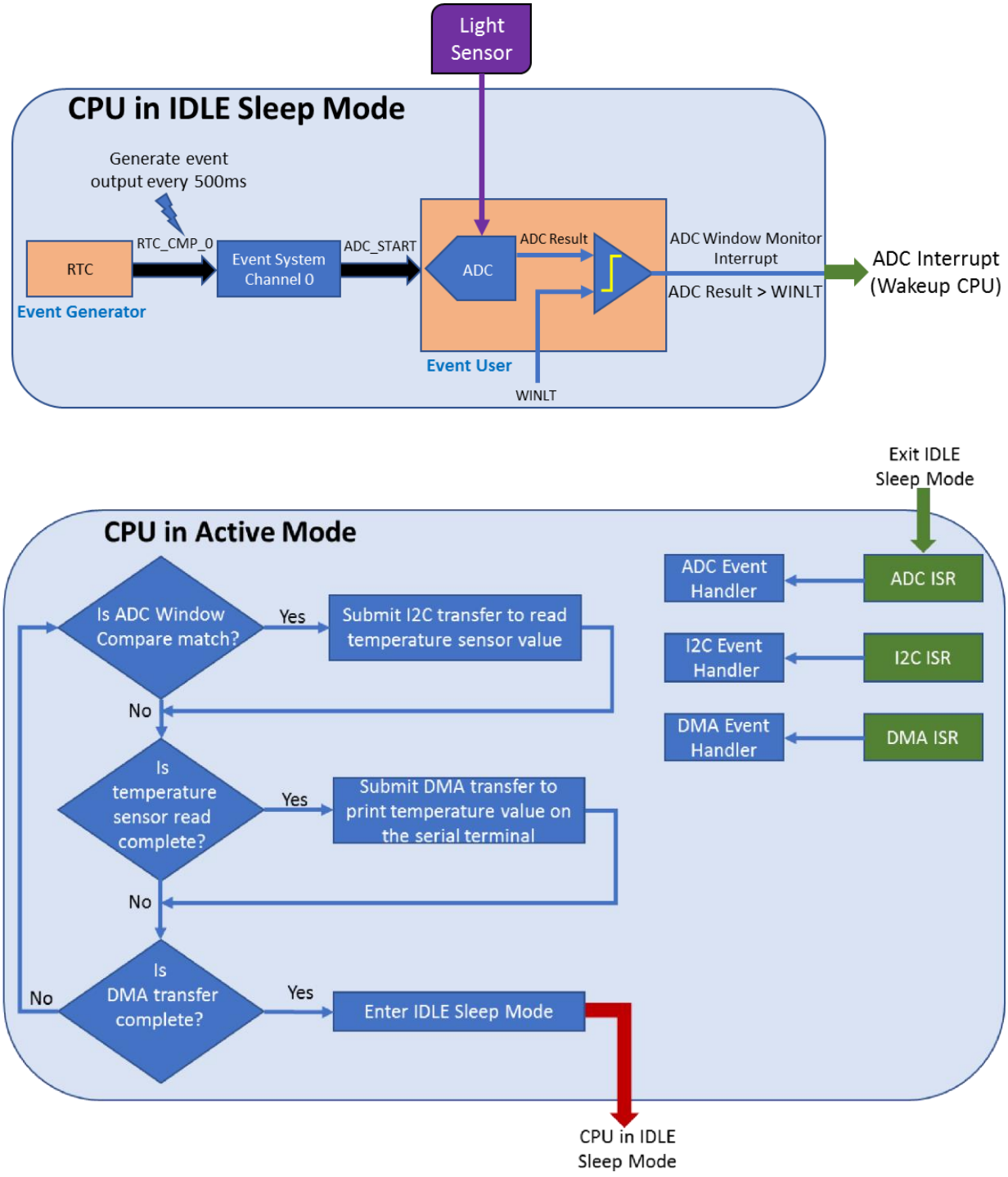
In this Lab you will add ADC, Event system and PM peripheral libraries. The application you create, will utilize:

- ADC Peripheral Library to sample the light sensor analog input and detect whether the light sensor is covered or not.
- RTC Peripheral Library to periodically generate event for ADC to sample the light sensor.
- Event System Peripheral Library to trigger the start of ADC conversion on every RTC compare match event. The Event System allows peripheral-peripheral communication without CPU intervention. This reduces the burden on the CPU and other resources when compared to the conventional interrupt-based systems.
- SERCOM-I2C Peripheral Library to read temperature from the temperature sensor.
- SERCOM-USART and DMA Peripheral Libraries to print the temperature values on a COM (serial) port terminal application running on a PC.
- PM (Power Manager) Peripheral Library to put the CPU in IDLE Sleep mode.

Initialization Flowchart:



CPU in IDLE Sleep Mode and Active Mode:



Procedure:

This lab is completed in the following steps.

Part 1: Configure ADC, RTC, Event System and PM Peripheral libraries

STEP 1**:** Configure ADC Peripheral Library

STEP 2: Disable RTC interrupt and enable Compare Event Output

STEP 3: Configure Event System

STEP 4: Configure Power Module (PM) Peripheral Library

STEP 5: Generate code

Part 2: Add application code and build the application

STEP 6: Add application code

STEP 7: Build and Run the Application

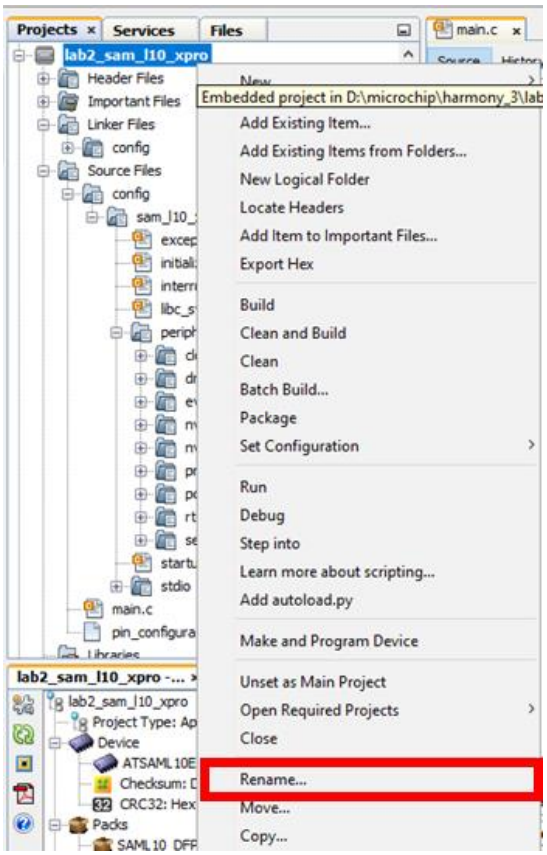
Note: You will develop Lab3 by extending the Lab2. If for some reason you were unable to complete the Lab2 successfully, then follow the below procedure (***otherwise, skip these steps***):

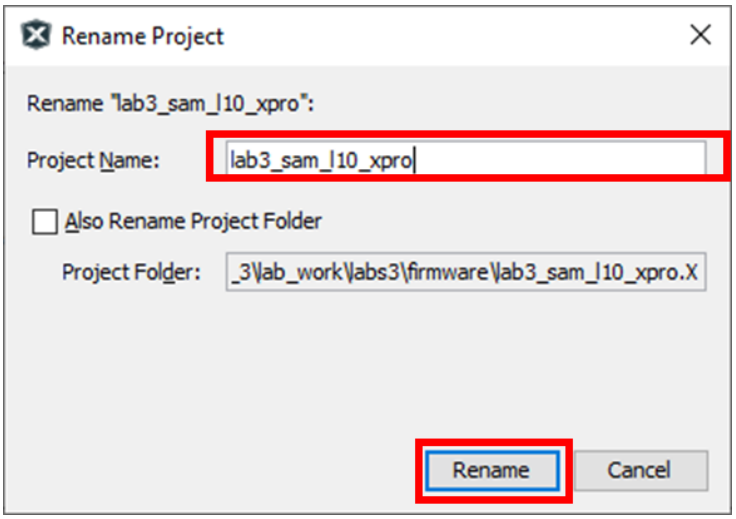
1. Close the Lab2 project (**lab2_sam_l10_xpro**) in MPLAB® X IDE by going to **File>Close All Projects**
2. Delete the **firmware** folder created by you at **<your-project-path>** location. For example, for the path used in this lab manual, the firmware folder will be under **/lab_work/labs** folder.
3. Copy the ready-made solution for the Lab2 available under the **lab_work/solutions/saml10/lab2** folder and paste the solution (**firmware** folder) to the **<your-project-path>** location**.** For example, for the path used in this lab manual, paste the **firmware** folder under **lab_work/labs** folder.
4. Drag and drop the MPLAB X project file (**sam_l10_xpro.X**) available under **<your-project-path>/firmware/** to the MPLAB X IDE
5. In MPLAB X IDE, right click on the project and set it as the main project.
6. In MPLAB X IDE, go to **Tools > Embedded** and open the **MPLAB Harmony 3 Configurator**.

Tool Tip:- If you closed MHC accidently, and would like to open, refer section **Appendix A – Launching MCC**

Part 1: Configure ADC, RTC, Event System and PM Peripheral libraries

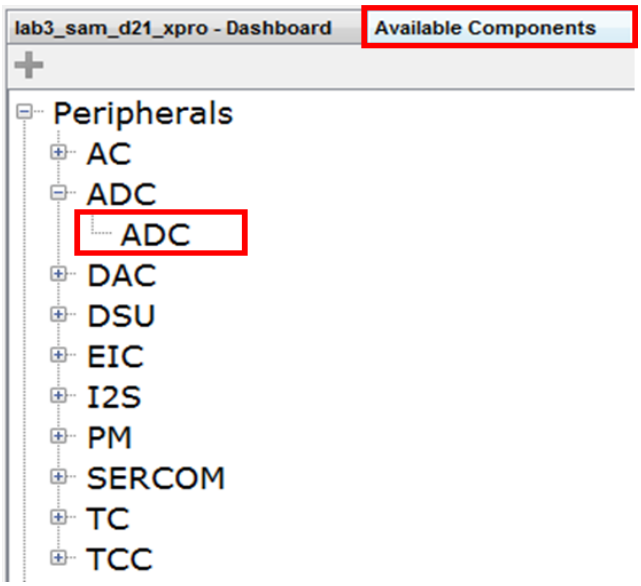
Since you are working on Lab3, change the logical name of the project under project tree from **lab2_sam_l10_xpro** to **lab3_sam_l10_xpro.**



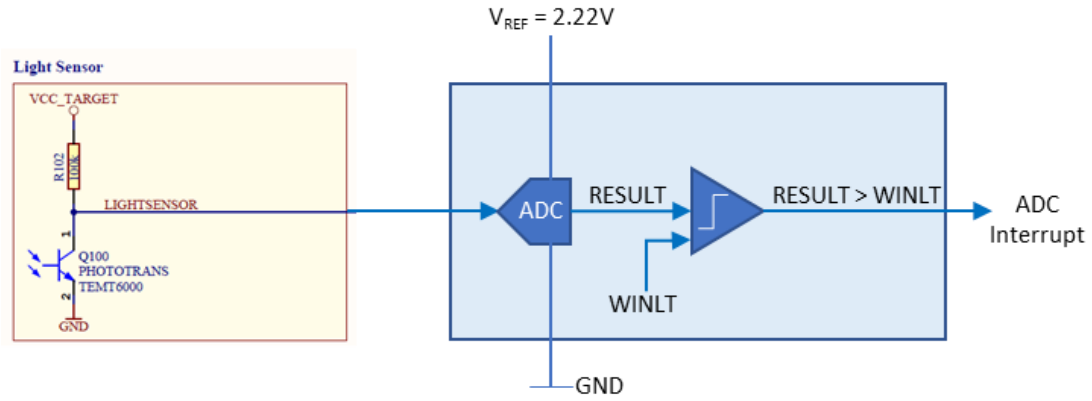


STEP 1: Configure ADC Peripheral Library

Step 1a: Under the bottom left tab "**Available Components**", Expand **Peripherals > ADC**
Select and double click on ADC to add the ADC module to the project.



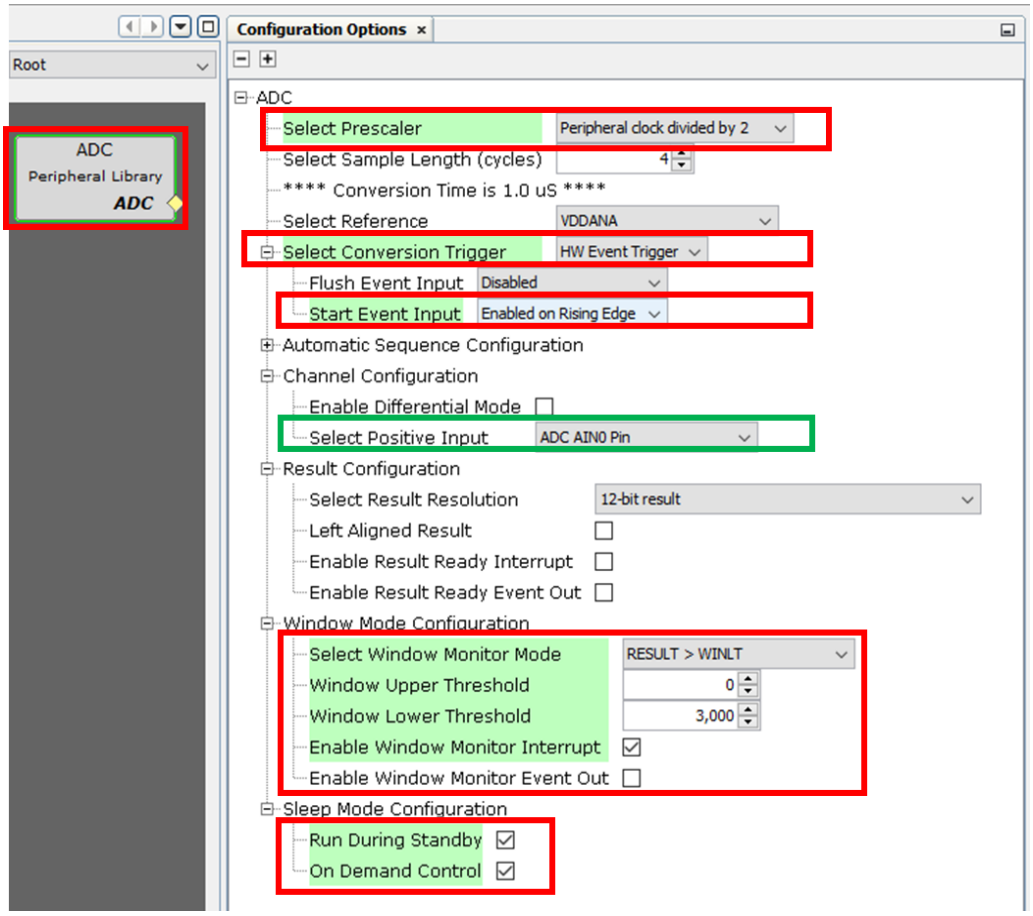
You will configure ADC PLIB to sample and convert the light sensor input. You will also configure the ADC to generate an interrupt (and thereby wakeup CPU), when the ADC Result is greater than a set threshold value.



When light sensor is not covered (light is falling on the sensor), the phototransistor is turned on. The ADC input is ~0V and the ADC RESULT register is close to 0x00.
When the light sensor is covered (light is not falling on the sensor), the phototransistor is turned off. The ADC input is ~3.3V and the ADC RESULT register will be saturated (0xFF). The ADC RESULT register is compared with the Window Lower Threshold (WINLT) which is set to 0xFE.
When the light sensor is covered (ADC RESULT > WINLT), a ADC Window Monitor interrupt is generated. This interrupt is used to bring the CPU out of IDLE sleep mode, when the user covers the light sensor.

Light Sensor Covered?	Phototransistor	ADC input V	ADC Result	ADC Result > WINLT(0xFE) ?	ADC Window Interrupt?
No	On	~0V	~0x00	No	No
Yes	Off	~3.3V	0xFF	Yes	Yes

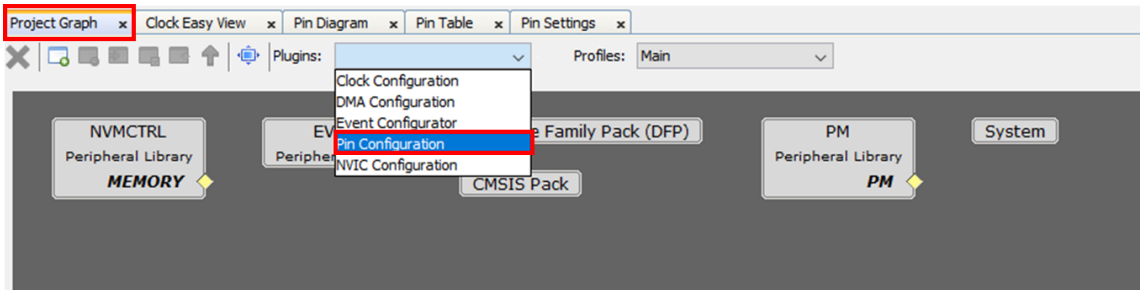
Select the **ADC PLIB** module and configure as shown below.



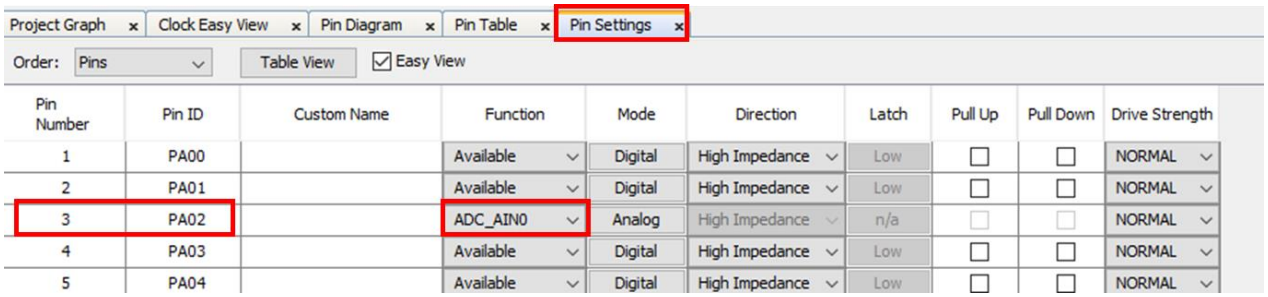
Notes:

1. Select Prescaler: Peripheral clock divided by 2 divides the ADC input clock with the configured pre-scaler value and provides more sampling time (CLKADC = 1 MHz/2 = 500 kHz).
2. Select Reference: **VDDANA**.
3. Select Conversion Trigger: **HW Event Trigger**.
4. Start Event Input: **Enabled on Rising Edge**.
5. Select Positive Input: **ADC AIN0 Pin** (the light sensor is connected to AIN0 pin of ADC).
6. Select Result Resolution: **12-bit result** (ADC conversion value range from 0 to 4096).
7. Open the **Window Mode Configuration** panel:
    - Select Window Monitor mode: **Result > WINLT**. A Window Monitor Interrupt is generated when the ADC result is greater than the configured WINLT value.
    - Window Upper Threshold: Sets the upper threshold of the window comparator. It is set to **0** as the ADC resolution is 12-bit.
    - Window Lower Threshold: Sets the lower threshold of the window comparator. It is set to **3000**. This means that the ADC Window Monitor Interrupt will be generated when the ADC result is greater than 3000. This value is decided based on the light sensor voltage when we cover the light sensor; it approximately generates 2.2 V. Hence, the threshold is set near to this voltage.
    - Enable Window Monitor Interrupt: A Window Monitor Interrupt is generated when the ADC result is greater than the configured WINLT.
    - Window Lower Threshold – Sets the lower threshold of the window comparator. It is set to 254. This means that ADC Window Monitor Interrupt will be generated when the ADC result is 255 (0xFF).

Step 1b: Open the Pin Configuration tabs by clicking **Plugins> Pin Configuration** in the project Graph



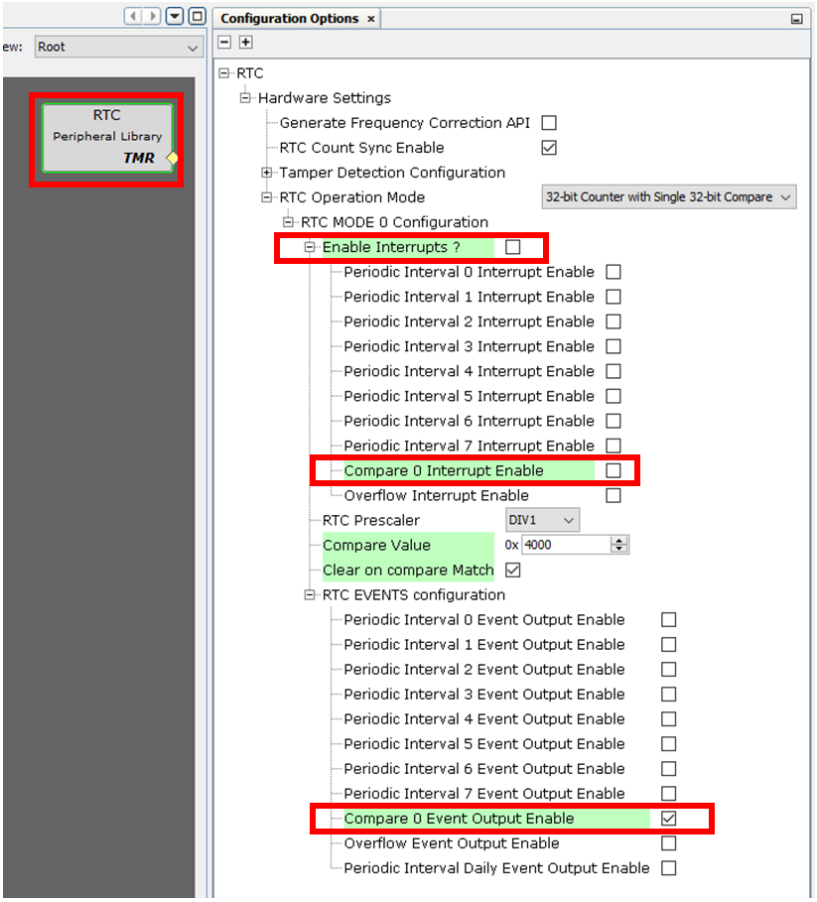
Go to the Pin Settings tab and configure PA02 (Pin #3) as ADC_AIN0 pin.



Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch	Pull Up	Pull Down	Drive Strength
1	PA00		Available	Digital	High Impedance	Low	☐	☐	NORMAL
2	PA01		Available	Digital	High Impedance	Low	☐	☐	NORMAL
3	PA02		ADC_AIN0	Analog	High Impedance	n/a	☐	☐	NORMAL
4	PA03		Available	Digital	High Impedance	Low	☐	☐	NORMAL
5	PA04		Available	Digital	High Impedance	Low	☐	☐	NORMAL

STEP 2: Disable RTC interrupt and enable Compare Event Output

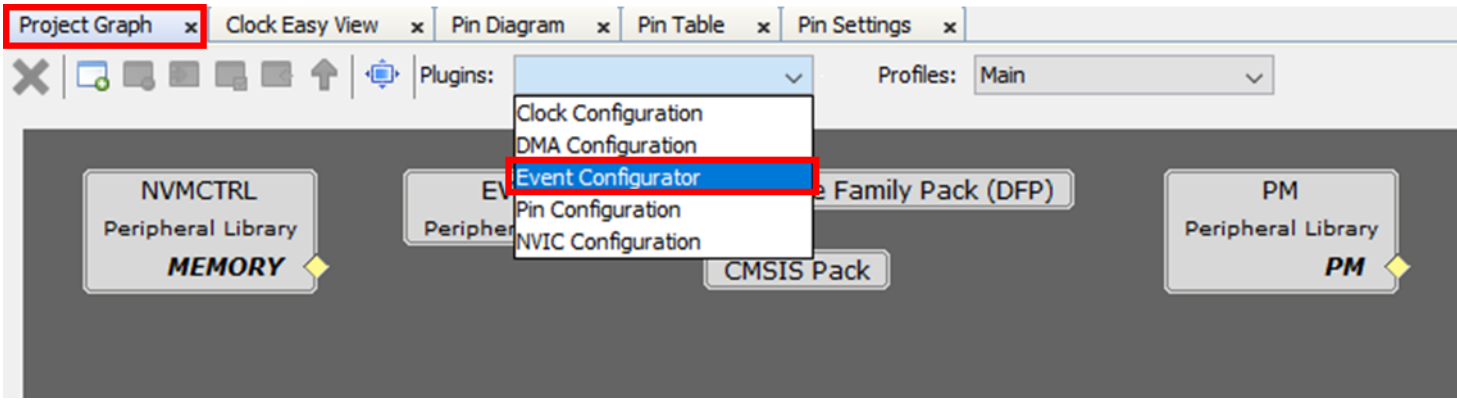
Under the **"Project Graph"** select the **RTC PLIB** block and disable the RTC Compare match interrupt. Also, enable the Compare 0 Event Output as shown below.



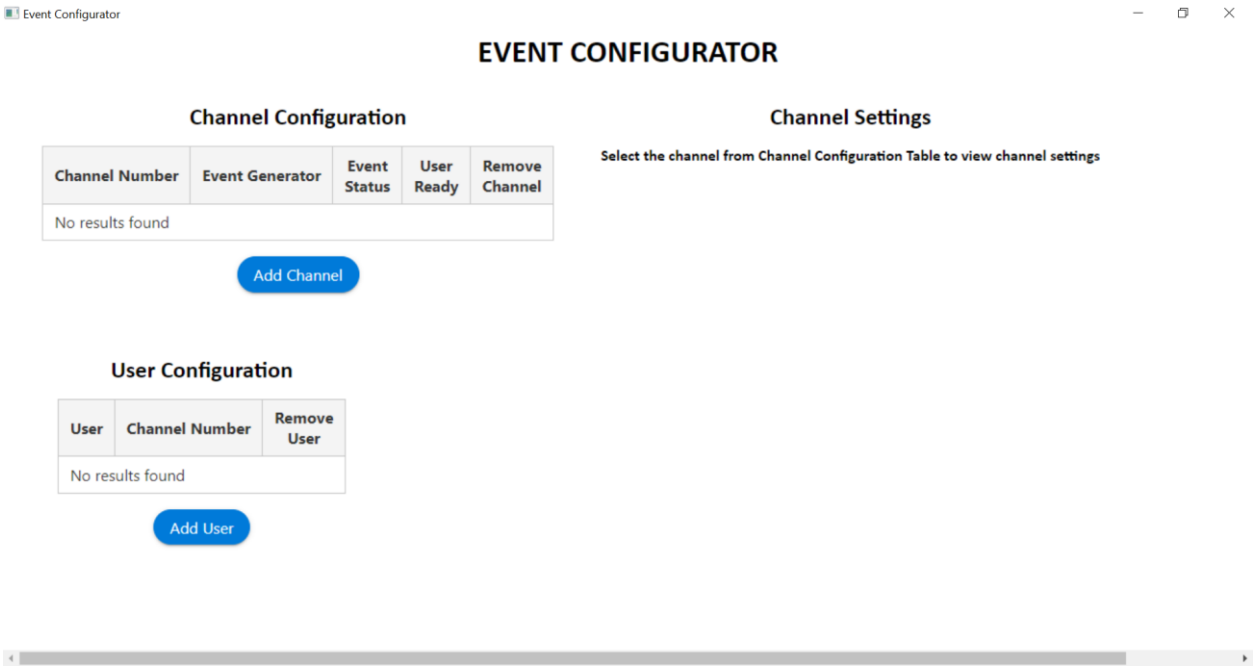
Note: The RTC Compare 0 Event (when RTC counter matches the compare value) output will serve as an input to the Event System module to trigger ADC to start conversion of the light sensor. As a result, CPU intervention is not required to start the ADC conversion on RTC interrupt. Hence, the RTC interrupt is disabled.

STEP 3: Configure Event System

Step 3a: Launch Event System Configuration window by going to **Project Graph** tab in MPLABX IDE and then selecting **Plugins > Event System Configuration**



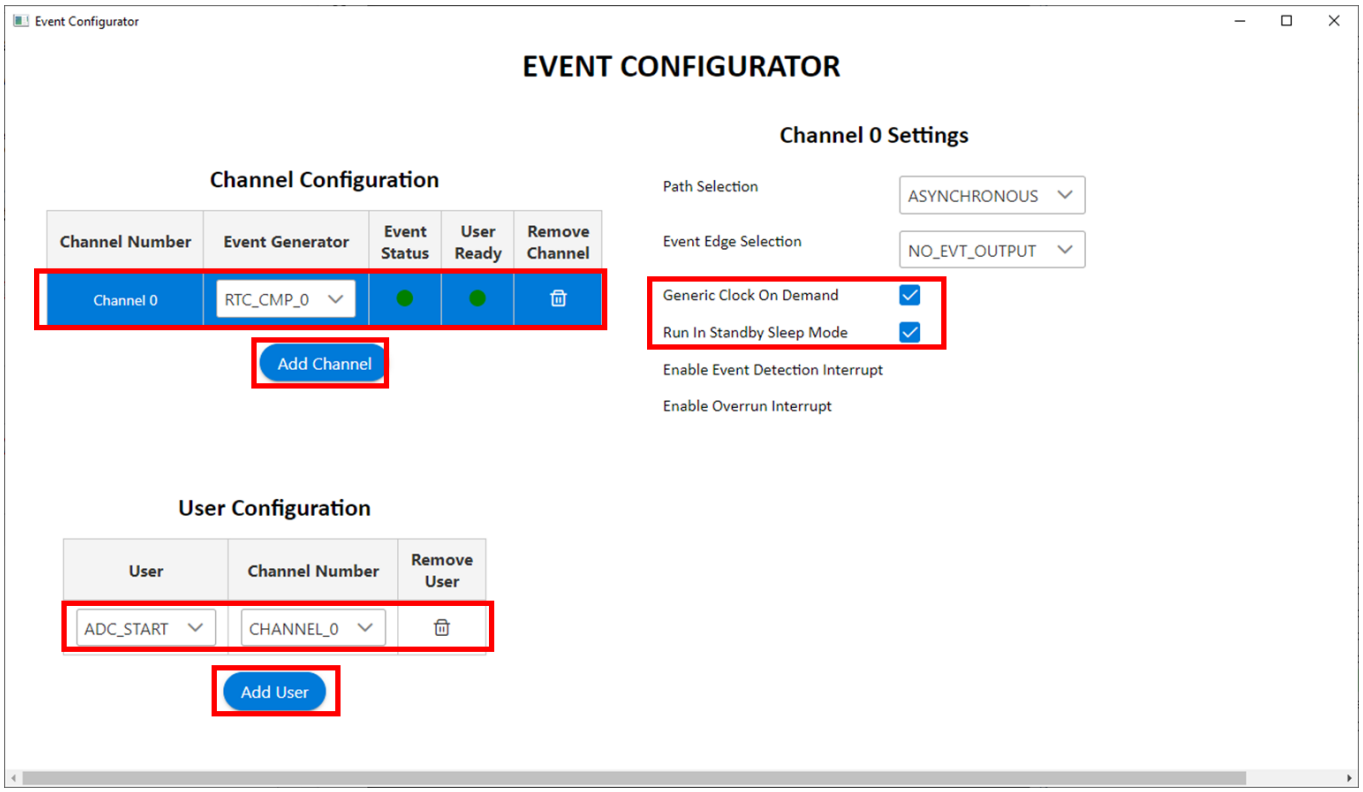
A new widow tab “EVENT0 Easy View” is opened.



Step 3b: Add and configure Event System Channel 0 with

- RTC compare 0 match as the event generator
- ADC Start of Conversion as the event user

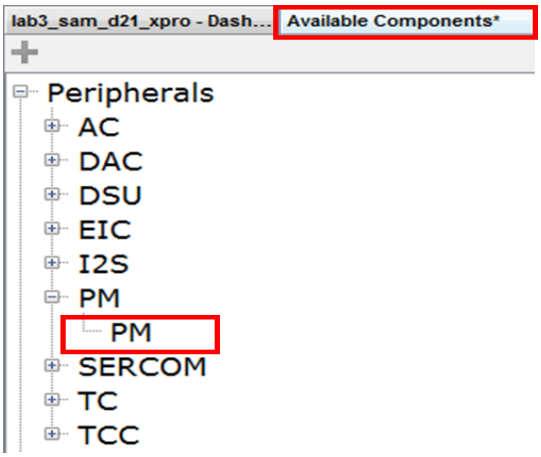
Note: Make sure that the status of the Event and User (Event Status and User Ready) is indicated Green in color. If it is Red, verify if the Event Output and Event Input is enabled in the respective (RTC and ADC) PLIB configuration.



STEP 4: Configure Power Module (PM) Peripheral Library

Step 4a: Under the bottom left tab "**Available Components**", Expand **Peripherals > PM**

Select and double click on PM to add the PM module to the project

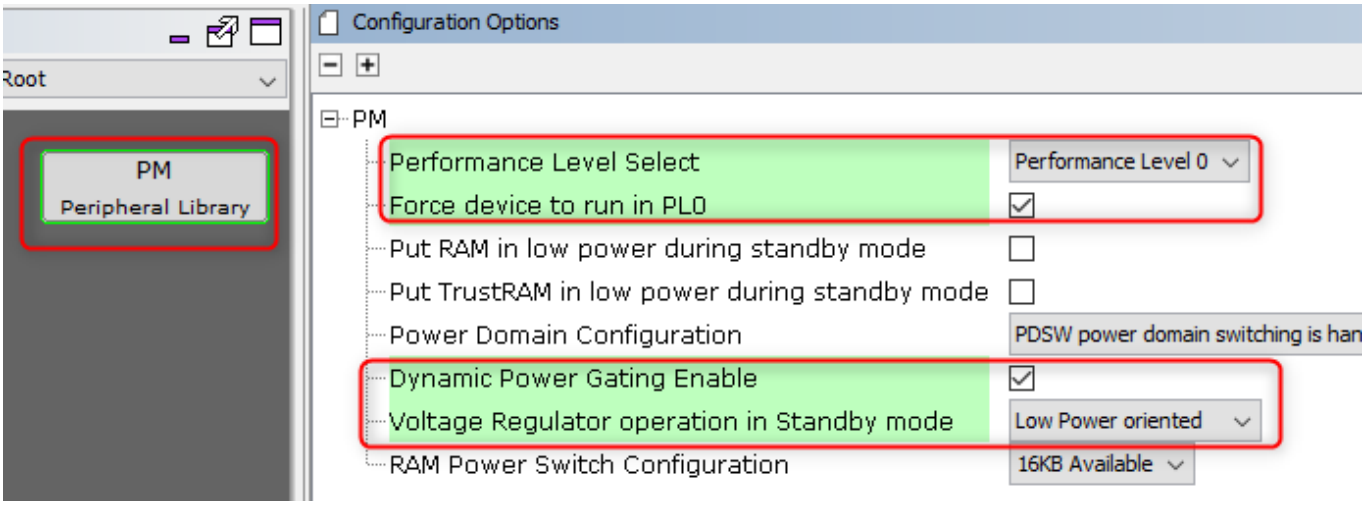


Step 4b: Keep the default configuration for the PM module.

The IDLE Sleep mode allows power optimization with the fastest wake-up time. The CPU is stopped. The CPU enters IDLE mode by executing the WFI instruction and exists the IDLE mode when any interrupt is generated.

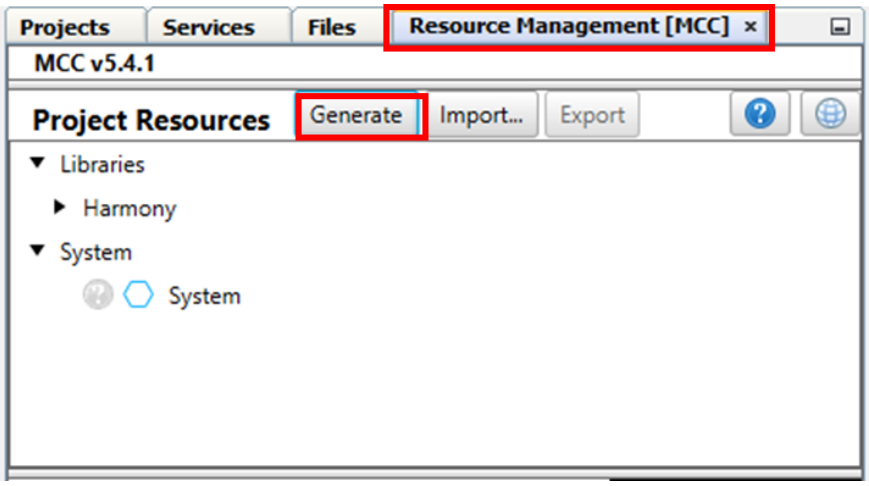
Select the **PM** peripheral library in the **Project Graph** window and configure it as shown below.

- Set the Performance Level Select to **Performance Level 0** to reduce the power consumption of the device during Standby mode.
- Enable **Dynamic Power Gating**: These options allow the device to turn off a power domain during Standby mode if no peripheral contained in this power domain requests its clock.
- Set the Voltage Regulator operation in Standby Mode to operate in **Low Power oriented** mode during standby to reduce the power consumption of the device even more.



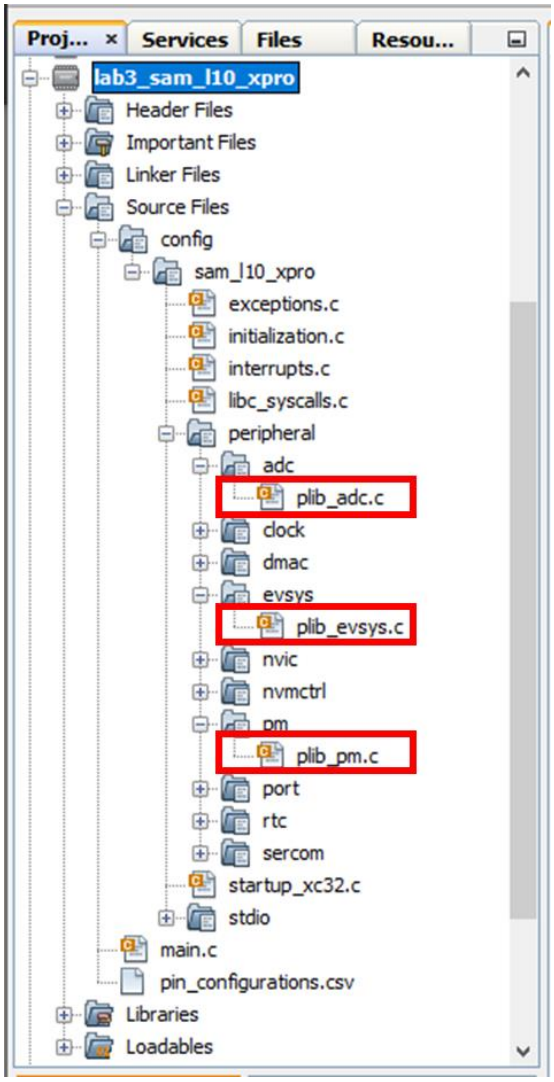
STEP 5: Generate code

Step 5a: Click on the **Generate** button in the Generate icon on Resouce management[MCC] window, as shown below.



Note: The MCC will include all the MPLAB® Harmony library files and generate the code based on the selections.

The MCC will include all the MPLAB® Harmony library files and generate the code based on the MCC selections.



Analyze the generated code:

The generated code adds all the necessary files for the selected PLIBs, and configures it based on the user selections. Notice the files added for the ADC, Event System and Power Management modules.

Note: Navigate to the **Projects** tab to view the project tree structure.

Step 5c: Build the code by clicking on the "Clean and Build" icon  .

The build result will throw the following errors:

The errors seen in the application file (main.c developed in Lab2) are because the RTC interrupt is disabled in this lab. The Harmony v3 configurator generates code only for the configured options. Since in this lab, the RTC interrupt is disabled the corresponding code (APIs and macros) to register RTC callback is not generated, resulting in build failure.

These errors will be addressed in the following section.

Part 2: Add application code and build the application

STEP 6: Add Application Code

Note: Like the earlier labs, the application file (main.c) is already developed and is available under "**help/saml10/lab3**" folder. Copy the pre-developed main.c file from "**help/saml10/lab3**" folder and replace (over-write) the corresponding main.c file in your project.

Tip: Search for the string **"Step #"** to locate the position where the code snippets need to be enabled by un-commenting it.

Note: Like the previous lab, the code snippets have already been added to the application. You will enable it by un-commenting them. The purpose of this exercise is to understand how to use the ADC, Event System and PM Peripheral Libraries in your application.

Step 6a: Open **main.c** file and register an event handler (callback) with the ADC PLIB. The event handler is called by the ADC PLIB when the ADC Window Monitor event occurs. The ADC Window Monitor event occurs when the ADC result exceeds the lower threshold value of 254, which was set when configuring the ADC PLIB.

Search for "Step #1" and un-comment the below line of code.

```
/*Register ADC Window Compare match Event Handler. Here ---> Step #1 */
ADC_CallbackRegister (adcEventHandler, 0);
```

Step 6b: Enable ADC module by calling the *ADC_Enable* API. ADC is used to sample the light sensor when a trigger is received from the Event System Channel 0, every 500 milliseconds.

Search for "Step #2" and un-comment the below line of code.

```
/*Start ADC. Here ---> Step #2*/
ADC_Enable();
```

Step 6c: Start the RTC timer. The event output from RTC is fed as input to the Event System and is used to start the ADC conversion.

Search for "Step #3" and un-comment the below line of code.

```
/*Start RTC Timer. Here ---> Step #3*/
RTC_Timer32Start();
```

Step 6d: Add code to put the CPU to IDLE Sleep mode. After a call to the *PM_IdleModeEnter* API, the CPU will enter IDLE Sleep mode. The ADC Window Monitor interrupt will bring the CPU out of the Sleep mode.

Search for "Step #4" and un-comment the below line of code.

```
/* Move the CPU to Idle Mode. Here -----> Step #4 */
PM_IdleModeEnter();
```

Step 6e: Add code to implement the ADC Window Monitor event handler. The event handler is called when the ADC result of the light sensor is greater than the window mode match value (254).

Search for "Step #5" and un-comment the below line of code.

```
/* ADC Window Monitor Event Handler. Here -----> Step #5  */
static void adcEventHandler(ADC_STATUS status, uintptr_t context)
{
    if (status & ADC_STATUS_WINMON)
    {
        adcResult = ADC_ConversionResultGet();
        isADCWinMonitorEvent = true;
    }
}
```

Step 6f: Submit I2C transfer to read temperature sensor value once the ADC Window Monitor event flag is set. The I2C PLIB calls back the I2C event-handler when the submitted request is complete (as shown in Step #7 below).

Search for "Step #6" and un-comment the below line of code.

```
if (isADCWinMonitorEvent == true)
{
    isADCWinMonitorEvent = false;

    /*Submit I2C Transfer to read temperature sensor value. Here ---> Step #6 */
    status=SERCOM1_I2C_WriteRead (TEMP_SENSOR_SLAVE_ADDR, &i2cWrData, 1, i2cRdData, 2);
```

Step 6g: Add code to set a flag in the I2C Event Handler to indicate that the I2C transfer is (reading of the temperature sensor) complete.

Search for "Step #7" and un-comment the below line of code.

```
static void i2cEventHandler(uintptr_t contextHandle)
{
    if (SERCOM1_I2C_ErrorGet() == SERCOM_I2C_ERROR_NONE)
    {
        /*I2C read complete. Here ---> Step #7 */
        isTemperatureRead = true;
    }
}
```

Step 6h: Add code to print the latest temperature value on the serial terminal using the DMA peripheral.

Search for "Step #8" and un-comment the below line of code.

```
sprintf((char*)uartTxBuffer, "Temperature = %02d F\r\n", temperatureVal);

/*Submit DMA transfer from memory to USART to display the latest
*temperature value and toggle the LED at the same time. Here ---> Step #8 */

status=DMAC_ChannelTransfer(
        DMAC_CHANNEL_0,
        uartTxBuffer,
        (const void *)&(SERCOM0_REGS->USART_INT.SERCOM_DATA),
        strlen((const char*)uartTxBuffer)
);
```

Step 6i: Add code to put the CPU back into IDLE Sleep mode by calling the API *PM_IdleModeEnter* when the DMA transfer is complete. The CPU comes out of the Sleep mode when the ADC Window Monitor event triggers an interrupt.

Search for "Step #9" and un-comment the below line of code.

```
if(isDMATransferComplete == true)
{
    isDMATransferComplete=false;

    LED_Set();

    /*Move the CPU to Idle Mode. Here ---> Step #9*/
    PM_IdleModeEnter ();
}
```

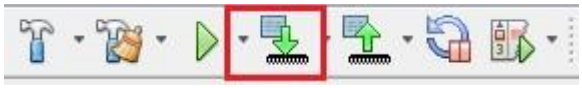
Congratulations! You have added ADC, Event System and PM Peripheral Libraries to your application. You have also enhanced your application to sample temperature sensor data periodically only when the light sensor is covered. You are now ready to build and run your enhanced application!

STEP 7: Build and Run the Application

Step 7a: If already open, then close the previous session of Tera Term.

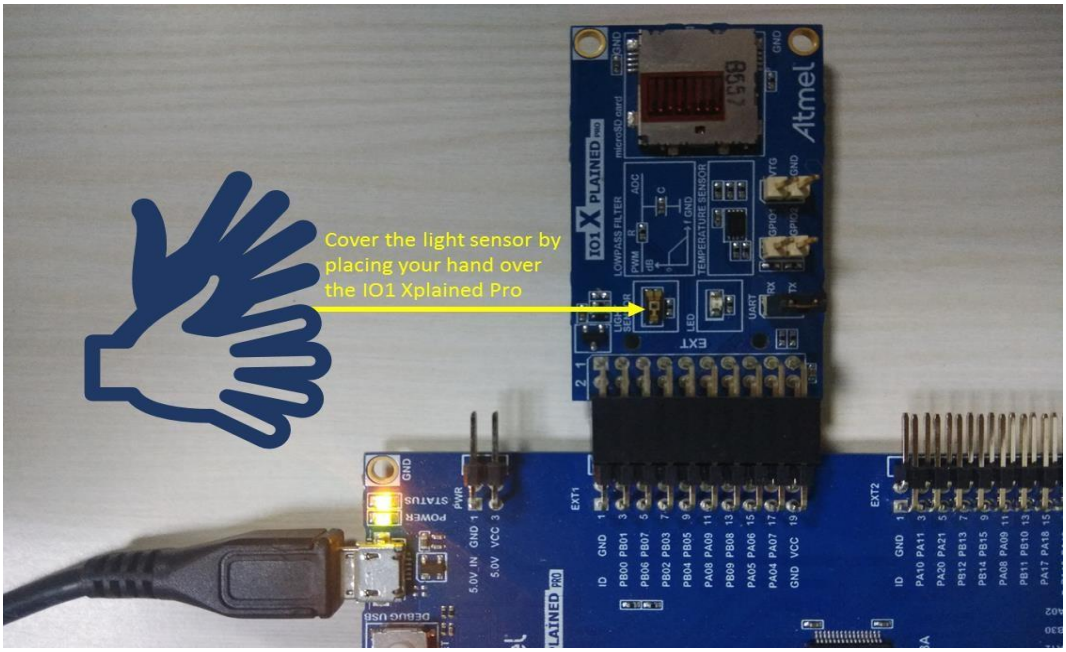
Step 7b: Clean and build your application by clicking on the “Clean and Build” button as shown below.

Step 7c: Program your application to the device, by clicking on the “Make and Program” button as shown below.

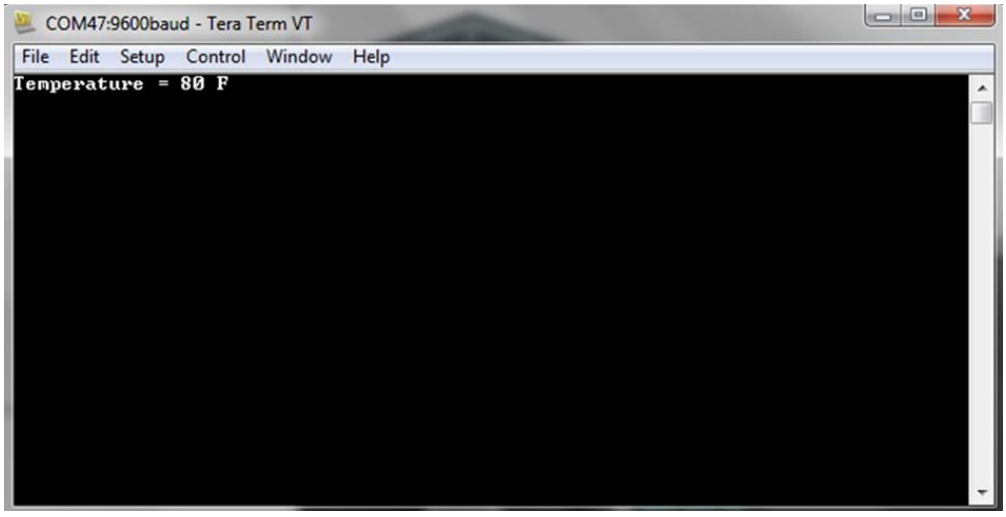


Results:

- Open the Tera Term terminal application on your PC.
- Cover the light sensor on the IO Xplained Pro board (by placing your hand over it) to print the temperature on the terminal.



You should see the temperature values (in °F) getting printed on the terminal every 500 milliseconds for the duration the light sensor is covered, as shown below.



Also notice that LED0 is ON for the duration the CPU is active. The LED0 turns OFF when the application enters sleep mode. When you continuously place your hand on the light sensor, the LED0 blinks and temperature values are printed on the console every 500millisecond interval.

Summary:

Using MPLAB® Harmony, you were able to enhance your application to add more functionality. This lab created a sensor based application to read temperature periodically from a temperature sensor only when the user covers the light sensor.

Most of the application work was off-loaded to the Event System. Event System provides efficient way to communicate between peripherals without CPU intervention. This reduces the load on the CPU and other system resources, compared to a traditional interrupt-based system, and lets the CPU to remain in sleep mode for longer durations.

RTC PLIB was configured to generate a compare match event output. ADC PLIB was configured to start ADC conversion on RTC event input. The Event System was configured with RTC as the event generator and ADC as the event user.

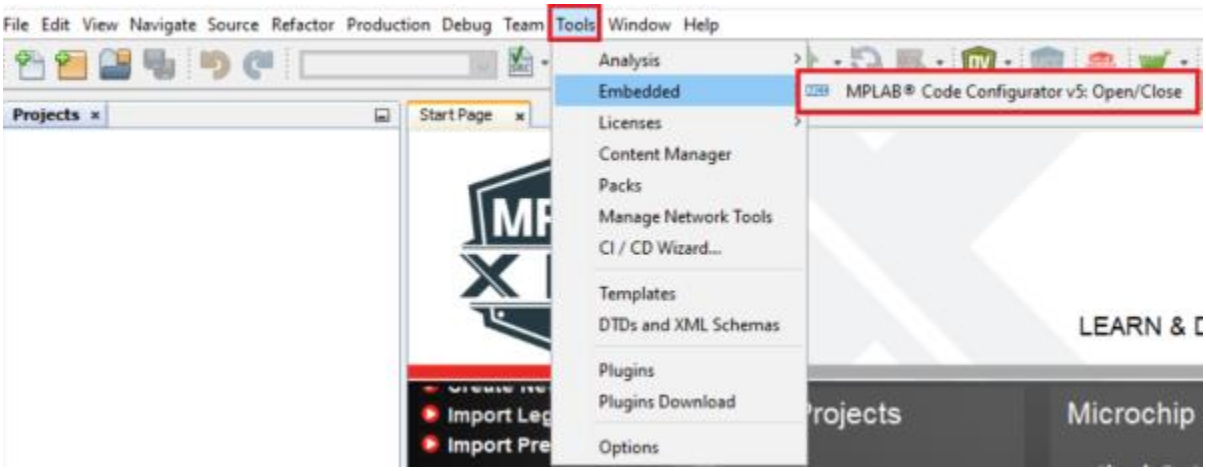
Further, the ADC PLIB was configured to generate an interrupt only when the ADC result was above a configured threshold. The ADC interrupt was used to bring the CPU out of the sleep mode.

Appendix A – Launching MCC

Follow the below steps to launch MCC.

1. Go to **Tools > Embedded**.

• You will see MPLAB® Code Configurator in the menu. If you don't see it in the menu, please install MCC.



2. The MCC plugin's main window for the project will be displayed