Adding DMA Support for Peripherals on PIC32CZ CA and SAM E70/S70/V7x MCUs using MPLAB Harmony v3 and MCC

TB3370

www.microchip.com
Product Page Links

Introduction

In modern computing systems, efficient data transfer mechanisms are crucial for maintaining high performance and responsiveness, especially in applications that handle large volumes of data. Direct Memory Access (DMA) is one such mechanism that is pivotal in optimizing data movement. The DMA significantly reduces processing bottlenecks by allowing hardware subsystems to access the system independently of the CPU. Additionally, the use of a dedicated DMA controller ensures that data transfers are managed seamlessly further enhancing system efficiency and performance.

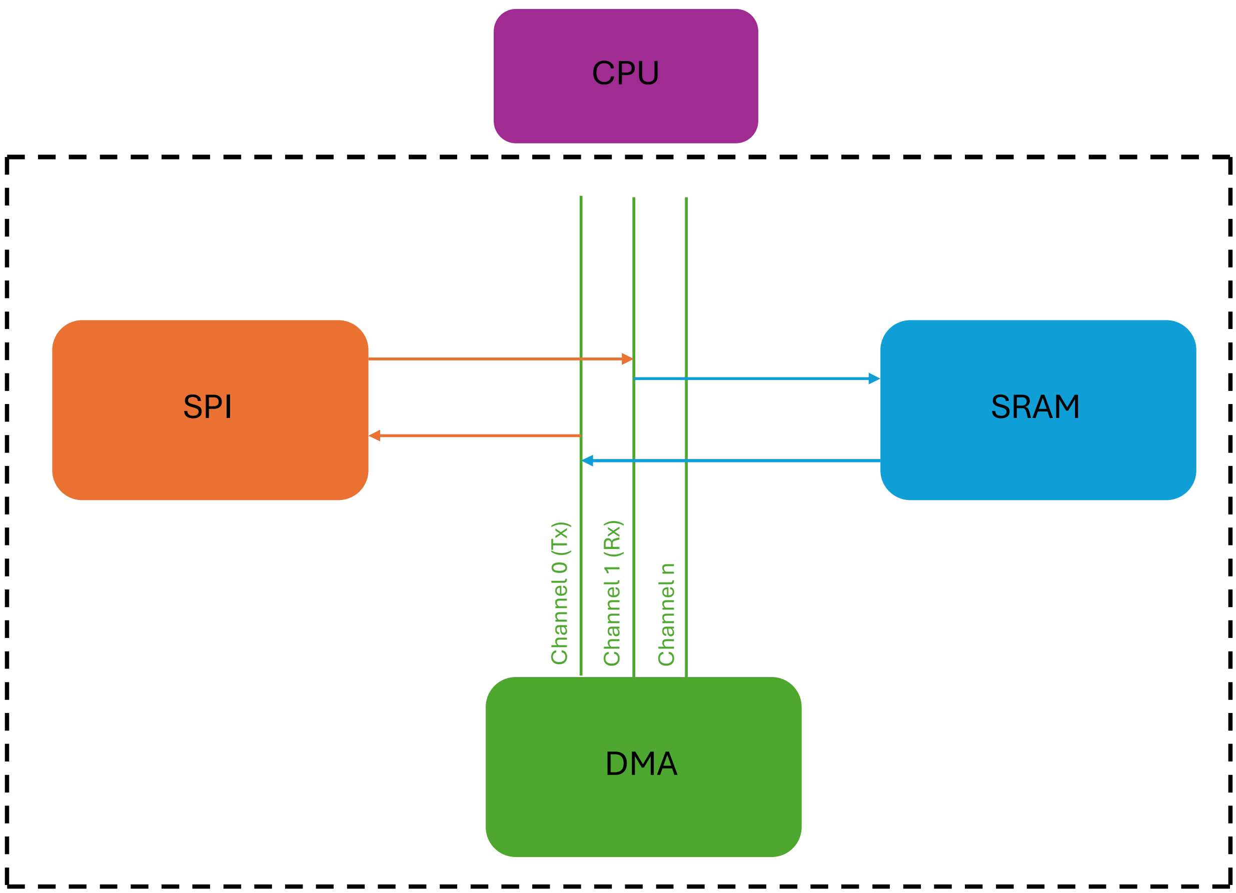
This document describes how to configure and use DMA with peripherals, such as the SPI on PIC32CZ CA and SAM E/S/V family of devices using MPLAB® Harmony v3 and MCC.

1. What is DMA?

Direct Memory Access (DMA) is an essential feature allowing specific hardware subsystems to access the system memory (RAM) independently of the central processing unit (CPU). This capability significantly enhances the efficiency and performance of data transfers, particularly in applications requiring the rapid movement of large data volumes, such as multimedia, networking, and storage systems.

The DMA process is managed by a dedicated hardware component known as the DMA controller, which can be integrated into the CPU or exist as a separate chip. The controller typically includes multiple channels, each capable of handling separate data transfer operations, and utilizes registers to store memory addresses, transfer counts, and control information. The primary advantages of the DMA include improved efficiency, faster data transfer rates, and reduced CPU overhead, making it indispensable in applications like audio and video streaming and real-time embedded systems.

Figure 1-1. DMA Block Diagram



1.1 Working Principle of DMA

The DMA transfer can be started only when a DMA transfer request is detected. The transfer requests may be software, peripheral, or an event. The DMA operation begins with the CPU initializing the DMA controller, followed by a peripheral device requesting a transfer. The DMA controller then arbitrates between multiple requests, if necessary, and takes control of the system bus to perform the data transfer directly between memory and the peripheral device, thereby freeing the CPU to perform other tasks. Upon completion, the DMA controller notifies the CPU through an interrupt.

1.2 Types of Data Transfers in DMA

The DMA data transfers are essential for optimizing system performance by enabling efficient communication between different components. These data transfers are categorized based on the source and destination of the data. It is categorized into the following four categories:

- Peripheral-to-Memory transfer
- Memory-to-Peripheral transfer
- Peripheral-to-Peripheral transfer
- Memory-to-Memory transfer

Table 1-1. Types of Data Transfers in DMA

Types	Source Location	Destination Location
Peripheral-to-Memory	Peripheral	Memory
Memory-to-Peripheral	Memory	Peripheral
Peripheral-to-Peripheral	Peripheral	Peripheral
Memory-to-Memory	Memory	Memory

Note: The SAM E/S/V family of devices does not support Peripheral-to-Peripheral type of data transfer.

The DMA transfer mode can also be categorized based on the size of the data being transferred by the components:

- **Beat transfer**: Size of one data bus transfer.
- **Block transfer**: Amount of data one transfer descriptor can transfer.
- **Burst transfer**: Back-to-back DMA transfer without CPU intervention.
- **DMA transaction**: Complete transfer of all data in a linked list of descriptors.
- **Cycle Stealing**: The DMA controller interrupts the CPU after every cycle to transfer data.

2. Creating the Application Using MPLAB Harmony v3 and MCC

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

- MPLAB X IDE v6.20
- MPLAB Code Configurator Plugin v5.5.1
- MPLAB XC32 Compiler v4.45
- csp v3.20.0
- PIC32CZ CA90 Curiosity Ultra Development Board
- PIC32CZ CA80 Curiosity Ultra Development Board
- SAM E70 Xplained Ultra Evaluation Kit
- SAM V71 Xplained Ultra Evaluation Kit

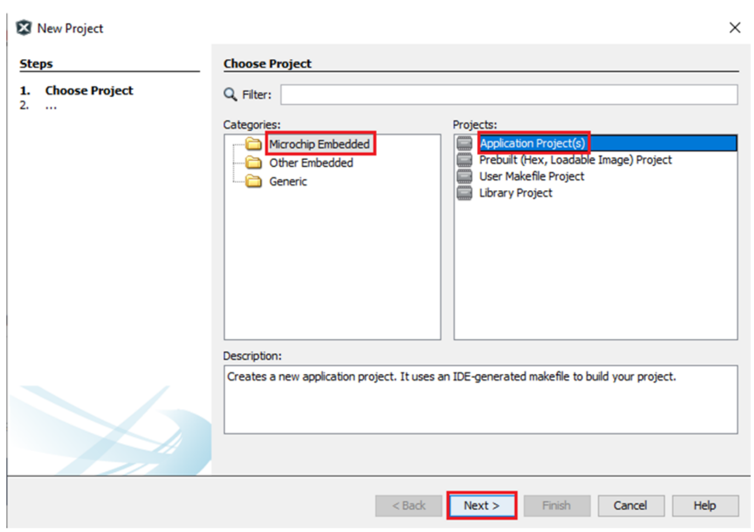
Note: The updated versions of the above listed tools can also be used to create the application, and users are not restricted to use the older versions. Also, only one of the above mentioned boards is needed for creating the application.

2.1 Creating Demo Application Using PIC32CZ CA90 Curiosity Ultra Development Board

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

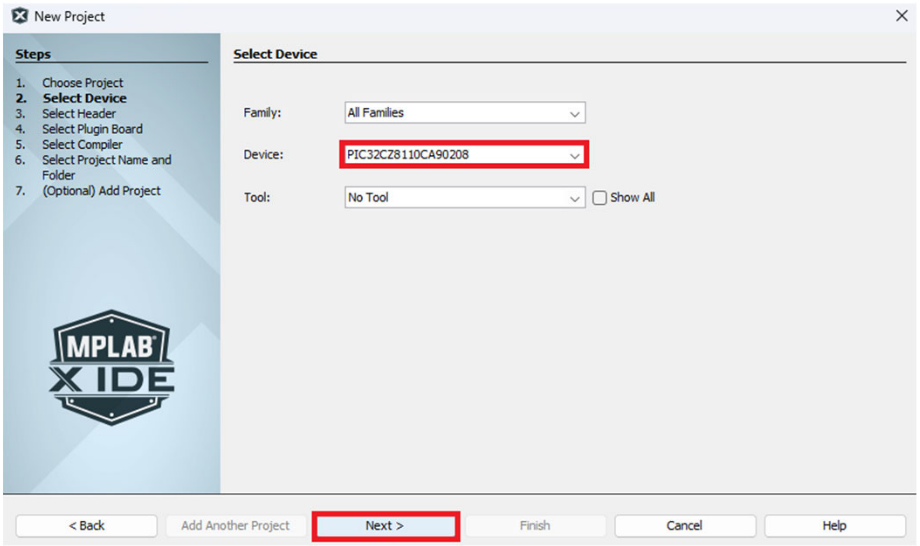
1. From the **Start** menu launch MPLAB X IDE.
2. On the **File** menu, click **New Project** or click on the *New Project* icon.
3. The **New Project** window will be displayed. From the **Steps** navigation pane, click **Choose Project**.
4. In the right **Choose Project** property page:
   a. Categories select **Microchip Embedded**.
   b. Projects select **Application Project(s)**.
5. Click **Next.**

Figure 2-1. New Project Window



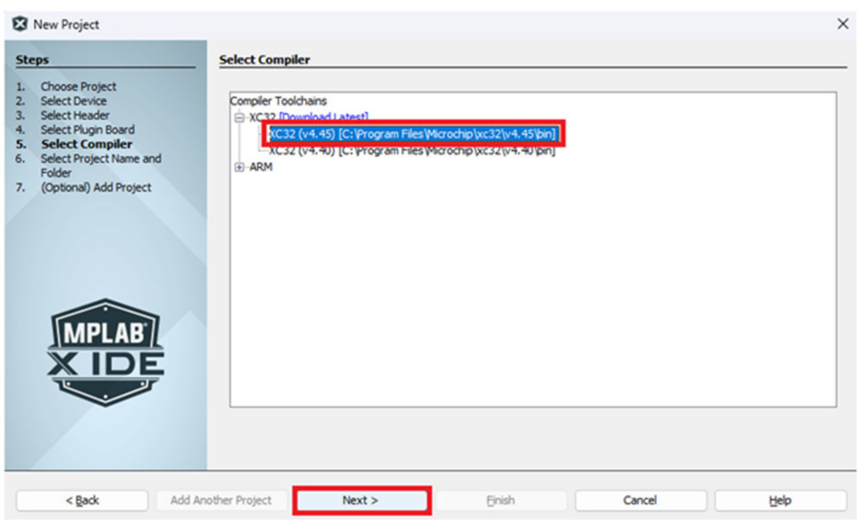
6. Click **Select Device** and in the right **Select Device** property page, for **Device** select **PIC32CZ8110CA90208** for creating the project on the PIC32CZ CA90 Curiosity Ultra Development Board (the selected device will be reflected under the Target Device).

Figure 2-2. Device Selection



7. Click **Next**.
8. Click **Select Compiler**, and in the right **Select Compiler** property page, under **Compiler Toolchains** click and expand **XC32**, and then select **XC32 (v4.45)**.

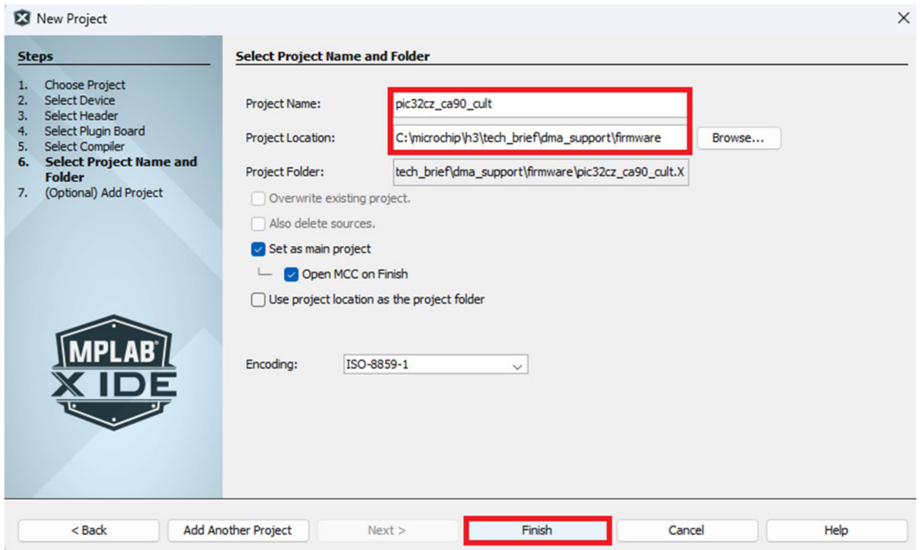
Figure 2-3. Select Compiler



9. Click **Next**.
10. Click **Select Project Name and Folder** and in the right **Select Project Name and Folder** property page:
    a. **Project Name**: Enter *pic32cz_ca90_cult* (Indicates the name of the project that will be shown in MPLAB X IDE to set the project's name).

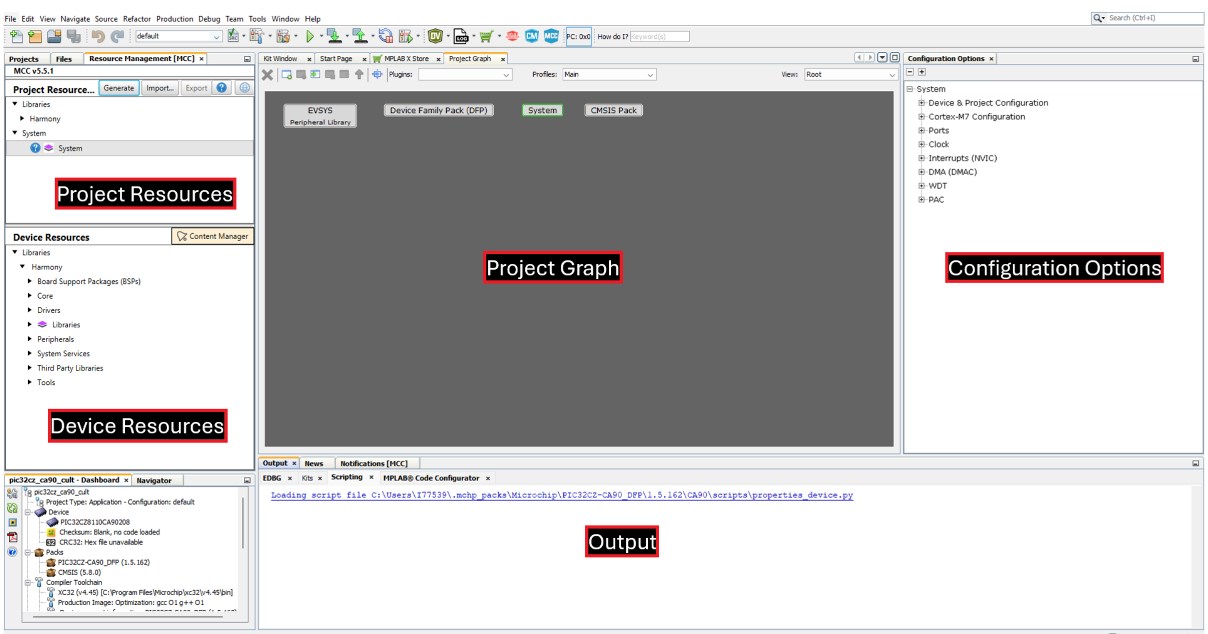
b. **Project Location**: Enter *C:\microchip\h3\tech_brief\dma_support\firmware* (Indicates the path to the root folder of the new project. All project files will be placed in this folder. The project location can be any valid path).
c. **Project Folder**: Read-only content (Automatically updates when users make changes to the above entries).

Figure 2-4. Project Name and Folder Settings



11. Click **Finish** to launch MCC.
12. The MCC plugin will open in a new window as shown in the following figure:

Figure 2-5. MPLAB Code Configurator Window

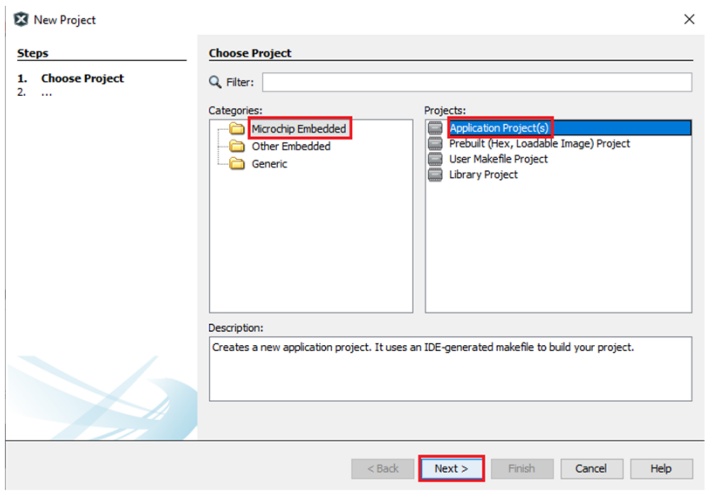


2.2 Creating Demo Application Using SAM E70 Xplained Ultra Evaluation Kit

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

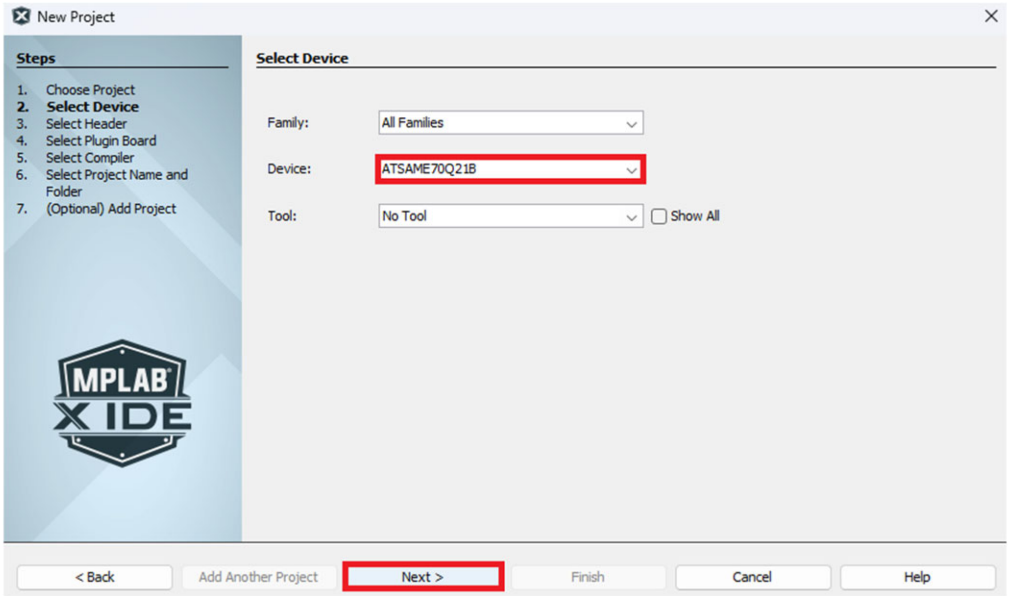
1. From the **Start** menu launch MPLAB X IDE.
2. On the **File** menu, click **New Project** or click on the *New Project* icon.
3. The **New Project** window will be displayed. From the **Steps** navigation pane, click **Choose Project**.
4. In the right **Choose Project** property page:
   a. Categories select **Microchip Embedded**.
   b. Projects select **Application Project(s)**.
5. Click **Next**.

Figure 2-6. New Project Window



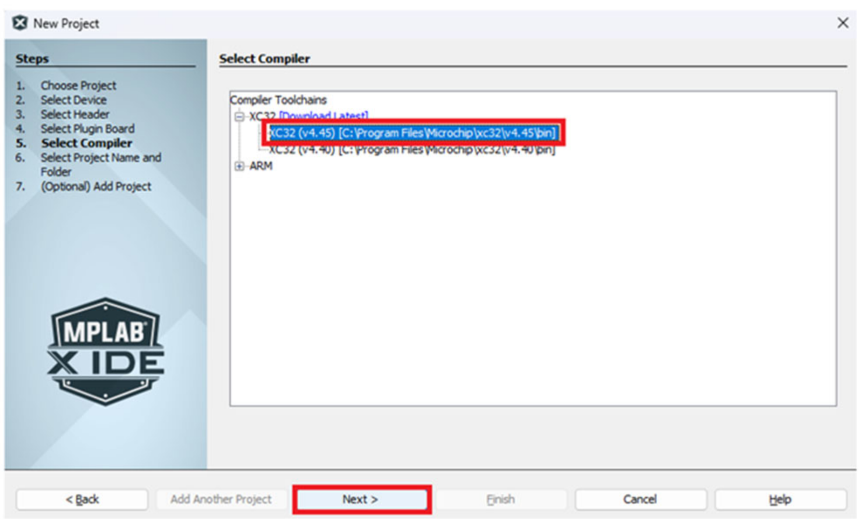
6. Click **Select Device**, and in the right **Select Device** property page, for **Device** select **ATSAME70Q21B** for creating the project on the SAM E70 Xplained Ultra Evaluation Kit (the device entry will be reflected under the Target Device).

Figure 2-7. Device Selection



7. Click **Next**.
8. Click **Select Compiler**, and in the right **Select Compiler** property page, under Compiler Toolchains click and expand **XC32** and then select **XC32 (v4.45)**.

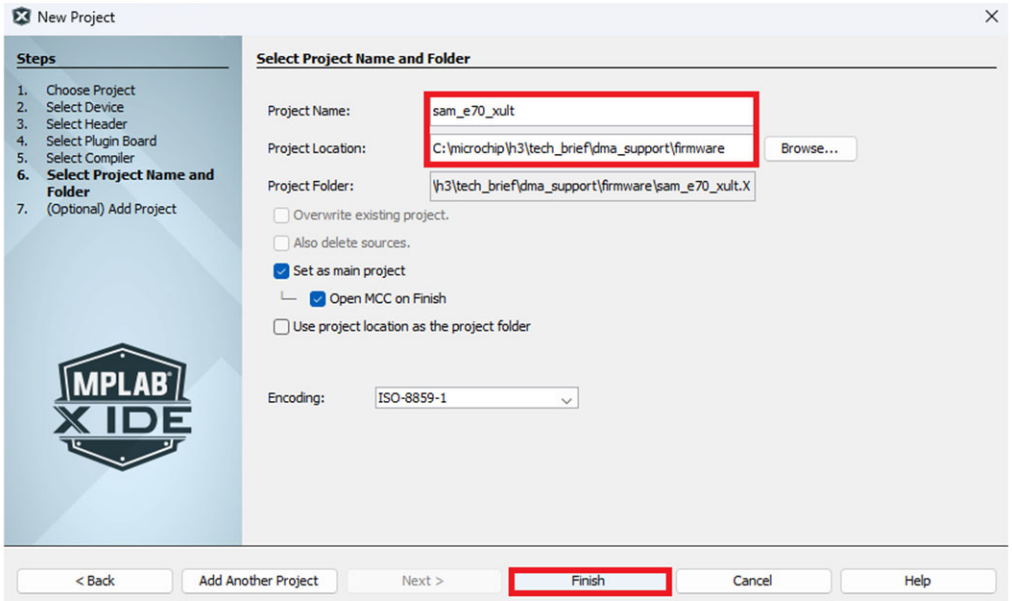
Figure 2-8. Selecting Compiler



9. Click **Next.**
10. Click **Select Project Name and Folder**, and in the right **Select Project Name and Folder** property page:
    a. **Project Name**: Enter *sam_e70_xult* (Indicates the name of the project that will be shown in MPLAB X IDE to set the project's name).
    b. **Project Location**: Enter *C:\microchip\h3\tech_brief\dma_support\firmware* (Indicates the path to the root folder of the new project. All project files will be placed in this folder. The project location can be any valid path).

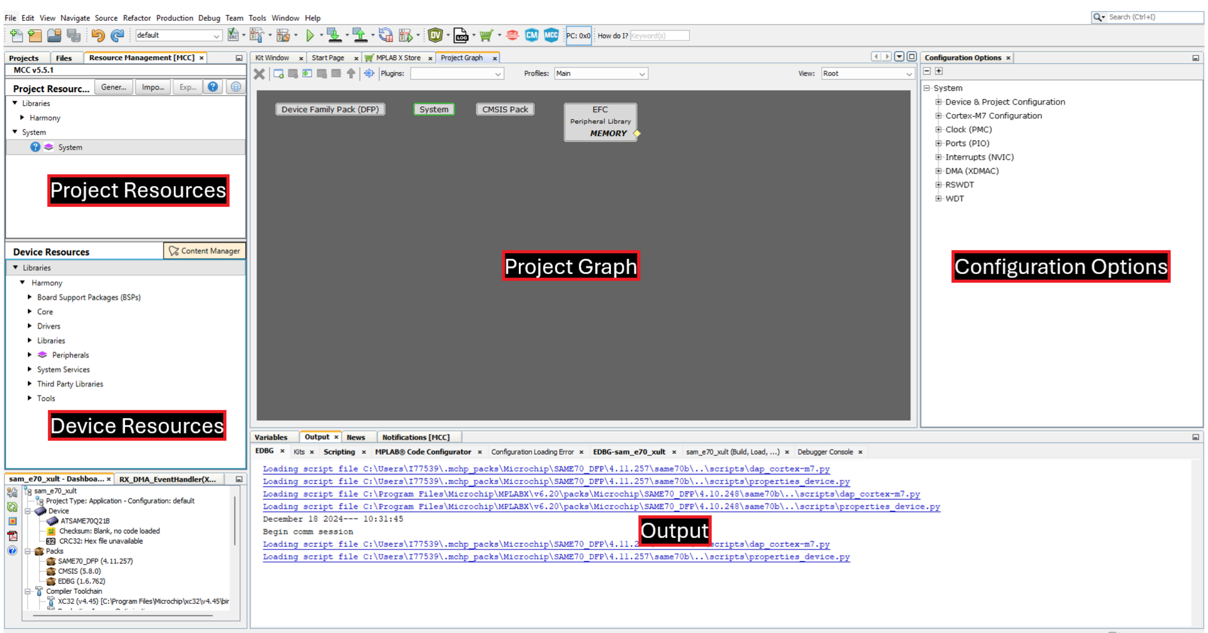
c. **Project Folder**: Read-only content (Automatically updates when users make changes to the above entries).

Figure 2-9. Project Name and Folder Settings



11. Click **Finish** to launch the MCC.
12. The MCC plugin will open in a new window as shown in the following figure:

Figure 2-10. MPLAB Code Configurator Window



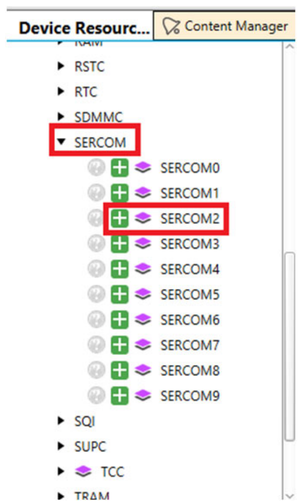
3. Adding and Configuring MPLAB Harmony Components

3.1 PIC32CZ CA90 Curiosity Ultra Development Board

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

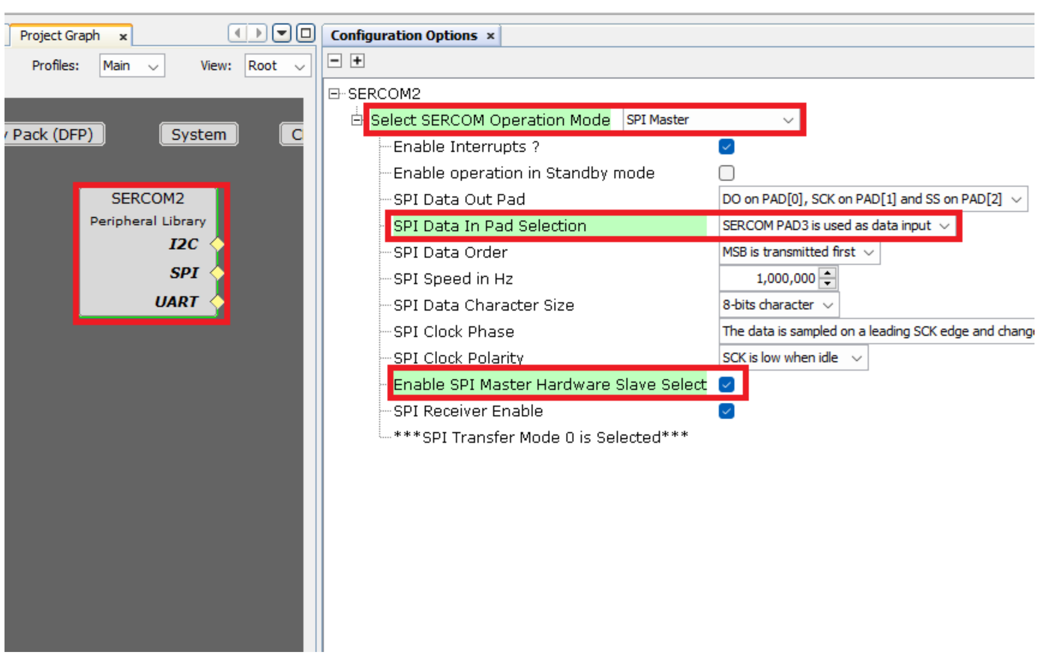
1. In the MCC window, under Device Resources, click and expand the list of options *Harmony* > *Peripherals* > *SERCOM.*
2. Click **SERCOM2** and observe that the SERCOM2 Peripheral Library block is added in the Project Graph section.

Figure 3-1. SERCOM Module



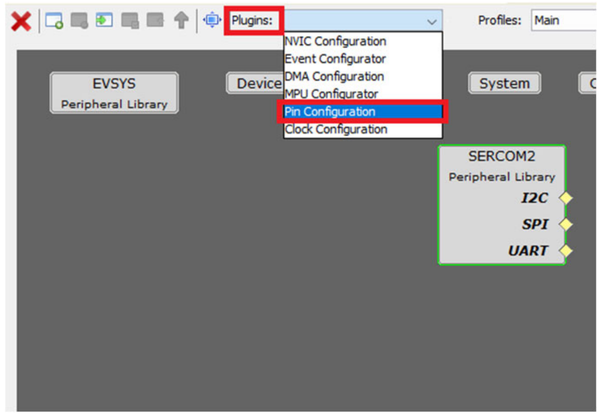
3. Click **SERCOM2** and in the **Configuration Options** property page, click and expand **SERCOM2** and configure SERCOM2 module as SPI Master as shown below.

Figure 3-2. SERCOM2 Configuration



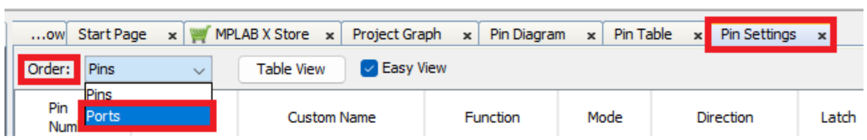
4. From the **Plugins** drop-down list, select **Pin Configuration**.

Figure 3-3. Plugins - Pin Configuration



5. Click **Pin Settings** and then sort entries by selecting **Ports** from the Order list.

Figure 3-4. Selecting Ports from the Order Menu



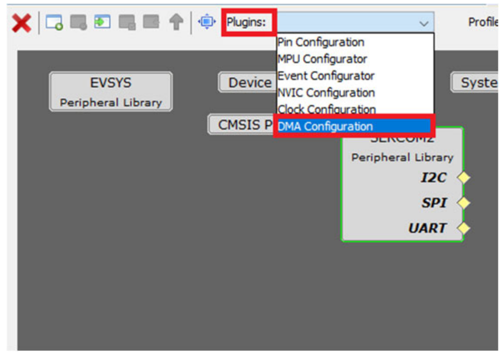
6. Configure the pins as shown below:

Figure 3-5. Configuration of Pins

Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch	Pull Up	Pull Down	Open Drain	Slew Rate
V15	PB31		Available	Digital	High Impedance	Low	☐	☐	☐	FAST
P15	PC00		Available	Digital	High Impedance	Low	☐	☐	☐	FAST
T17	PC01		Available	Digital	High Impedance	Low	☐	☐	☐	FAST
T18	PC02		Available	Digital	High Impedance	Low	☐	☐	☐	FAST
R17	PC03		Available	Digital	High Impedance	Low	☐	☐	☐	FAST
R18	PC04		Available	Digital	High Impedance	Low	☐	☐	☐	FAST
N15	PC05		Available	Digital	High Impedance	Low	☐	☐	☐	FAST
N17	PC06		Available	Digital	High Impedance	Low	☐	☐	☐	FAST
N18	PC07		Available	Digital	High Impedance	Low	☐	☐	☐	FAST
M13	PC08		SERCOM2_PAD0	Digital	High Impedance	n/a	☐	☐	☐	FAST
M18	PC09		SERCOM2_PAD1	Digital	High Impedance	n/a	☐	☐	☐	FAST
H17	PC10		SERCOM2_PAD2	Digital	High Impedance	n/a	☐	☐	☐	FAST
H15	PC11		SERCOM2_PAD3	Digital	High Impedance	n/a	☐	☐	☐	FAST
F18	PC12		Available	Digital	High Impedance	Low	☐	☐	☐	FAST
F17	PC13		Available	Digital	High Impedance	Low	☐	☐	☐	FAST
E17	PC14		Available	Digital	High Impedance	Low	☐	☐	☐	FAST

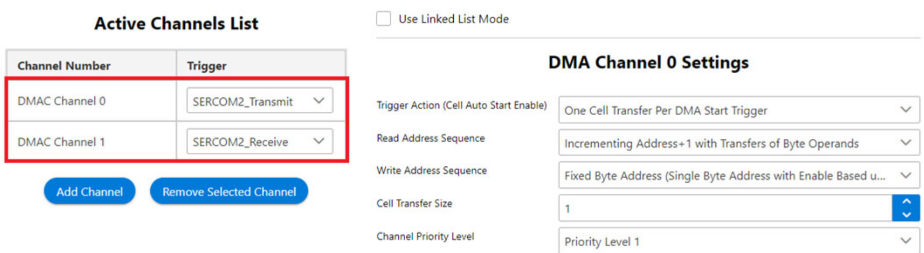
7. In the **Plugins** drop-down list, select **DMA Configuration**.

Figure 3-6. Plugins - DMA Configuration



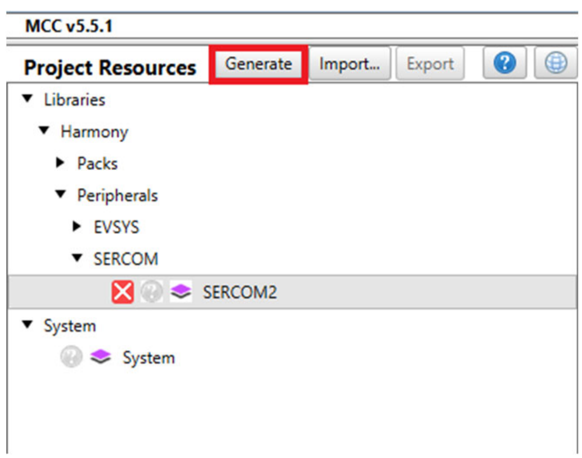
8. In the DMA Configuration, click **Add Channel** to add DMAC Channel 0 and choose the trigger as SERCOM2_Transmit. Repeat the same steps, but for DMAC Channel 1 choose SERCOM2_Receive as the trigger.

Figure 3-7. DMA Configuration



9. Click **Generate** to generate the code.

Figure 3-8. Generating the Code

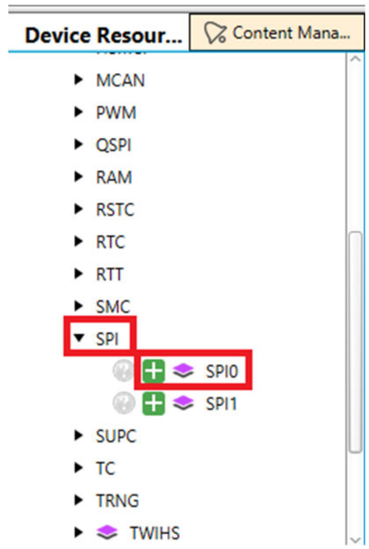


3.2 SAM E70 Xplained Ultra Evaluation Kit

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

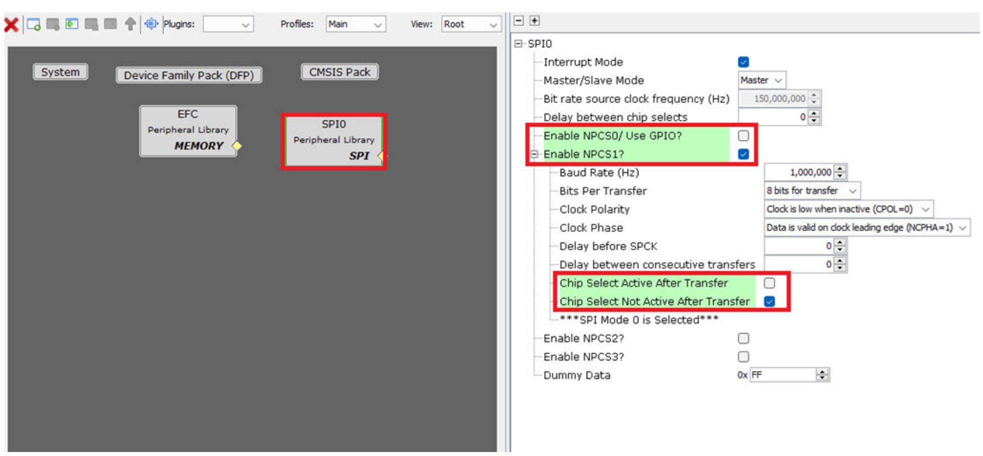
1. In the MCC window, under Device Resources, click and expand the list of options *Harmony* > *Peripherals* > *SPI.*
2. Click **SPI0** and observe that the SPI0 Peripheral Library block is added in the Project Graph section.

Figure 3-9. SPI Module



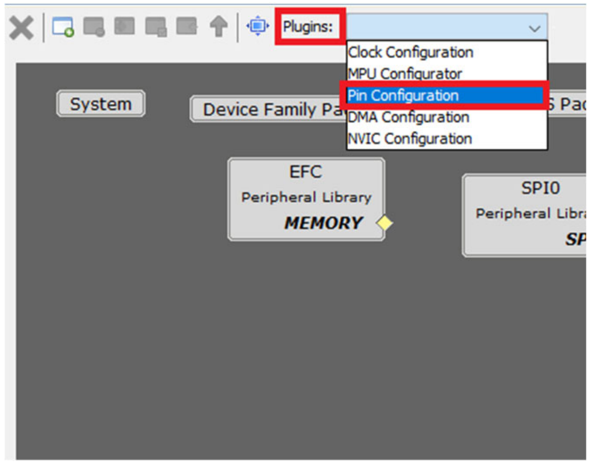
3. Click SPI0, and to configure SPI0 module, click and expand SP10 and then configure it as shown below.

Figure 3-10. SPI0 Configuration



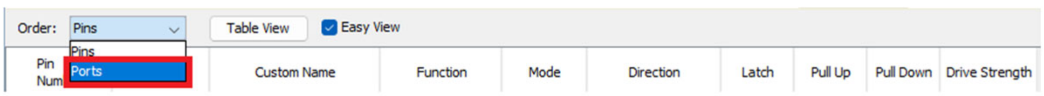
4. From the **Plugins** drop-down list, select **Pin Configuration**.

Figure 3-11. Plugins - Pin Configuration



5. From the **Order** drop-down list, select **Ports** to sort the entries.

Figure 3-12. Selecting Ports from the Order Menu



6. From the **Plugins** drop-down list, select **DMA Configuration**. In the **DMA Configuration** property page, click **Add Channel** to add DMAC Channel 0, and set the trigger to SPI0_Transmit. Repeat the same steps, but for DMAC Channel 1, set the trigger to SPI0_Receive.

Figure 3-13. Plugins - DMA Configuration

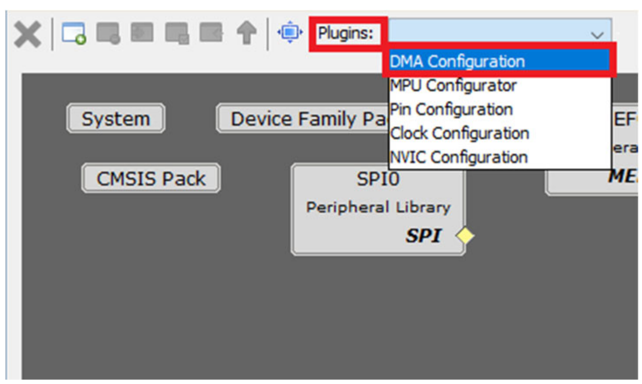
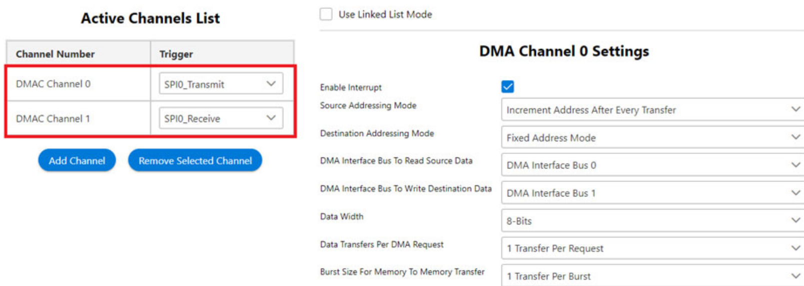
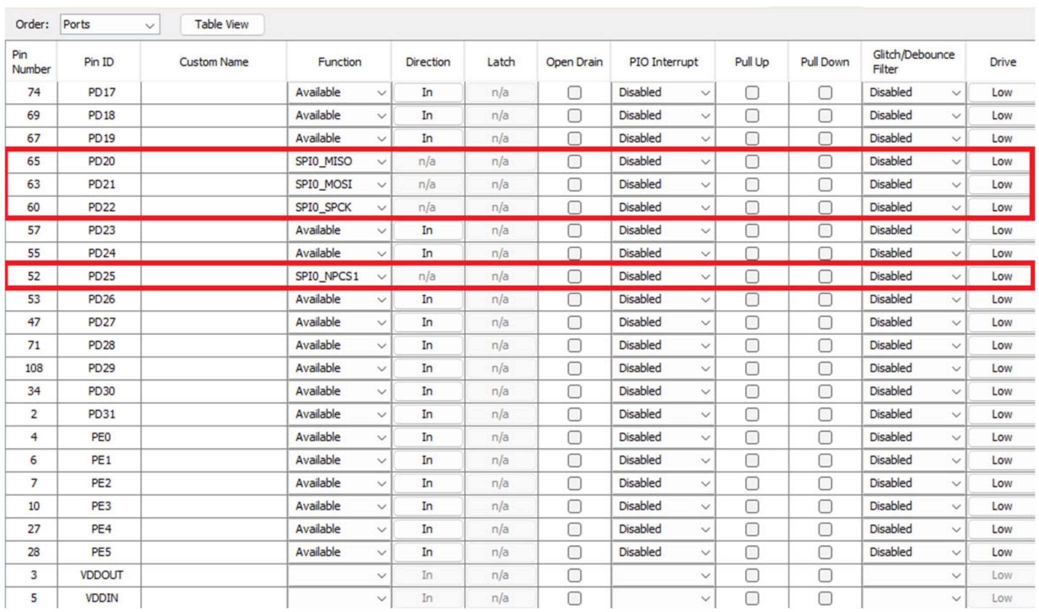


Figure 3-14. DMA Configuration



7. Configure the pins as shown below.

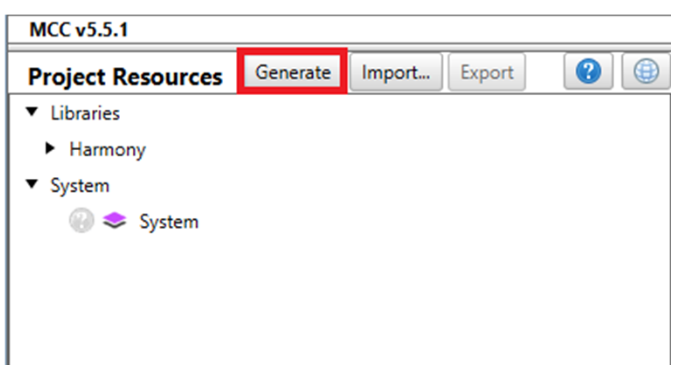
Figure 3-15. Configuration of Pins



Pin Number	Pin ID	Custom Name	Function	Direction	Latch	Open Drain	PIO Interrupt	Pull Up	Pull Down	Glitch/Debounce Filter	Drive
74	PD17		Available	In	n/a		Disabled			Disabled	Low
69	PD18		Available	In	n/a		Disabled			Disabled	Low
67	PD19		Available	In	n/a		Disabled			Disabled	Low
65	PD20		SPI0_MISO	n/a	n/a		Disabled			Disabled	Low
63	PD21		SPI0_MOSI	n/a	n/a		Disabled			Disabled	Low
60	PD22		SPI0_SPCK	n/a	n/a		Disabled			Disabled	Low
57	PD23		Available	In	n/a		Disabled			Disabled	Low
55	PD24		Available	In	n/a		Disabled			Disabled	Low
52	PD25		SPI0_NPCS1	n/a	n/a		Disabled			Disabled	Low
53	PD26		Available	In	n/a		Disabled			Disabled	Low
47	PD27		Available	In	n/a		Disabled			Disabled	Low
71	PD28		Available	In	n/a		Disabled			Disabled	Low
108	PD29		Available	In	n/a		Disabled			Disabled	Low
34	PD30		Available	In	n/a		Disabled			Disabled	Low
2	PD31		Available	In	n/a		Disabled			Disabled	Low
4	PE0		Available	In	n/a		Disabled			Disabled	Low
6	PE1		Available	In	n/a		Disabled			Disabled	Low
7	PE2		Available	In	n/a		Disabled			Disabled	Low
10	PE3		Available	In	n/a		Disabled			Disabled	Low
27	PE4		Available	In	n/a		Disabled			Disabled	Low
28	PE5		Available	In	n/a		Disabled			Disabled	Low
3	VDDOUT			In	n/a						Low
5	VDDIN			In	n/a						Low

8. Click **Generate** to generate the code.

Figure 3-16. Generating the Project



4. Adding Application Logic to the Project

4.1 PIC32CZ CA90 Curiosity Ultra Development Board

To develop and run the application, follow these steps:

1. Click **Projects** and then under Source files open the `main.c` file of the project and add the required variables outside the `main()` function.

```
uint8_t __attribute__ ((aligned(32))) Tx[10] = {0x1F, 0x2F, 0x3F, 0x4F, 0x5F, 0x6F, 0x7F,
0x8F, 0x9F, 0xAF};
uint8_t __attribute__ ((aligned(32)))Rx[10];

/* transfer done flag */
volatile bool Rx_transfer_done = false;
volatile bool Tx_transfer_done = false;
```

2. Add the DMA Event Handler for both Tx and Rx outside the `main()` function.

```
/* This is called after transfer is done */
void Tx_DMA_EventHandler(DMA_TRANSFER_EVENT event, uintptr_t context)
{
    if (event == DMA_TRANSFER_EVENT_BLOCK_TRANSFER_COMPLETE)
    {
        Tx_transfer_done = true;
    }
}

void Rx_DMA_EventHandler(DMA_TRANSFER_EVENT event, uintptr_t context)
{
    if (event == DMA_TRANSFER_EVENT_BLOCK_TRANSFER_COMPLETE)
    {
        Rx_transfer_done = true;
    }
}
```

Figure 4-1. Adding Macros, Variables, and Event Handlers

3. Add the DMA Callback register function, cache invalidate function, and DMA Channel transfer function.

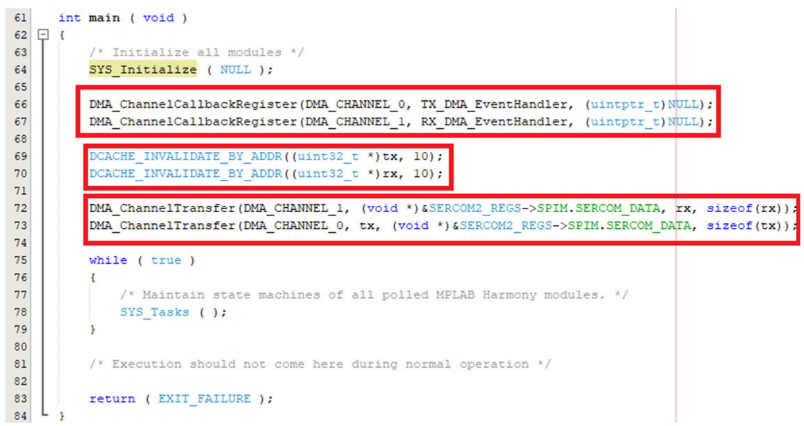
```
DMA_ChannelCallbackRegister(DMA_CHANNEL_0, Tx_DMA_EventHandler, (uintptr_t)NULL);
    DMA_ChannelCallbackRegister(DMA_CHANNEL_1, Rx_DMA_EventHandler, (uintptr_t)NULL);

    DCACHE_INVALIDATE_BY_ADDR((uint32_t *)Tx, 10);
    DCACHE_INVALIDATE_BY_ADDR((uint32_t *)Rx, 10);

    DMA_ChannelTransfer(DMA_CHANNEL_1, (void *)&SERCOM2_REGS->SPIM.SERCOM_DATA, Rx, 
sizeof(Rx));
    DMA_ChannelTransfer(DMA_CHANNEL_0, Tx, (void *)&SERCOM2_REGS->SPIM.SERCOM_DATA, 
sizeof(Tx));
```

Note: Cache invalidation is crucial for maintaining data consistency and accuracy, especially in dynamic systems where data frequently changes. By invalidating the cache, it ensures that the next time the data is requested, it will be fetched from the original source rather than the cache.

Figure 4-2. Application Logic



4.2 SAM E70 Xplained Ultra Evaluation Kit

To develop and run the application, follow these steps:

1. Open the `main.c` file of the project and add the required variables outside the `main()` function.

```
uint8_t __attribute__ ((aligned(32))) Tx[10] = {0x1F, 0x2F, 0x3F, 0x4F, 0x5F, 0x6F, 0x7F, 
0x8F, 0x9F, 0xAF};
uint8_t __attribute__ ((aligned(32)))Rx[10];

/* transfer done flag */
volatile bool Rx_transfer_done = false;
volatile bool Tx_transfer_done = false;
```

2. Add the DMA Event Handler for both Tx and Rx outside of the `main()` function.

```
/* This is called after transfer is done */
void Tx_DMA_EventHandler(XDMAC_TRANSFER_EVENT event, uintptr_t context)
{
    if (event == XDMAC_TRANSFER_COMPLETE)
    {
        Tx_transfer_done = true;
    }
}

void Rx_DMA_EventHandler(XDMAC_TRANSFER_EVENT event, uintptr_t context)
{
    if (event == XDMAC_TRANSFER_COMPLETE)
    {
        Rx_transfer_done = true;
    }
}
```

Figure 4-3. Adding Macros, Variables, and Event Handlers

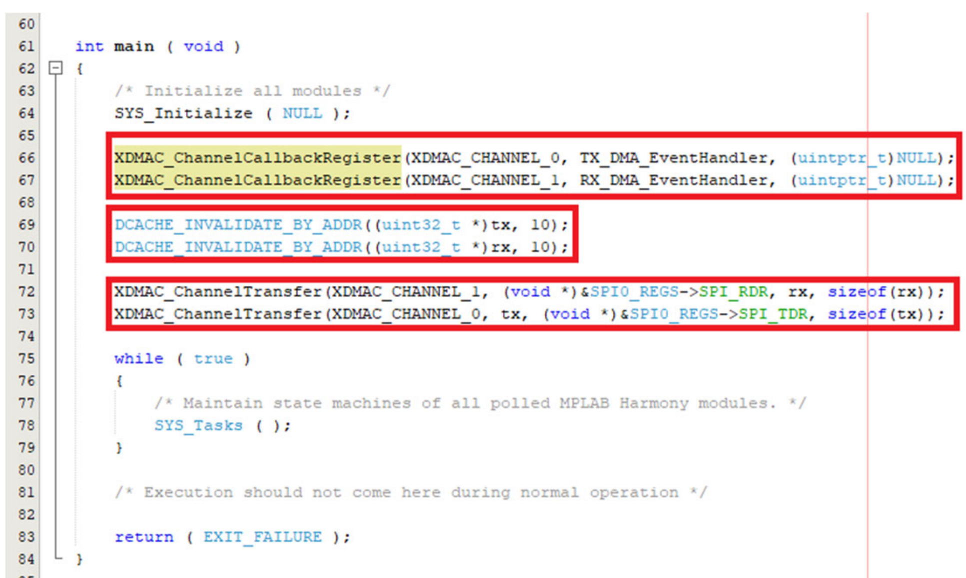
3. Add the DMA Callback register function, cache invalidate function, and DMA Channel transfer function.

```
XDMAC_ChannelCallbackRegister(XDMAC_CHANNEL_0, Tx_DMA_EventHandler, (uintptr_t)NULL);
XDMAC_ChannelCallbackRegister(XDMAC_CHANNEL_1, Rx_DMA_EventHandler, (uintptr_t)NULL);

DCACHE_INVALIDATE_BY_ADDR((uint32_t *)Tx, 10);
DCACHE_INVALIDATE_BY_ADDR((uint32_t *)Rx, 10);

XDMAC_ChannelTransfer(XDMAC_CHANNEL_1, (void *)&SPI0_REGS->SPI_RDR, Rx, sizeof(Rx));
XDMAC_ChannelTransfer(XDMAC_CHANNEL_0, Tx, (void *)&SPI0_REGS->SPI_TDR, sizeof(Tx));
```

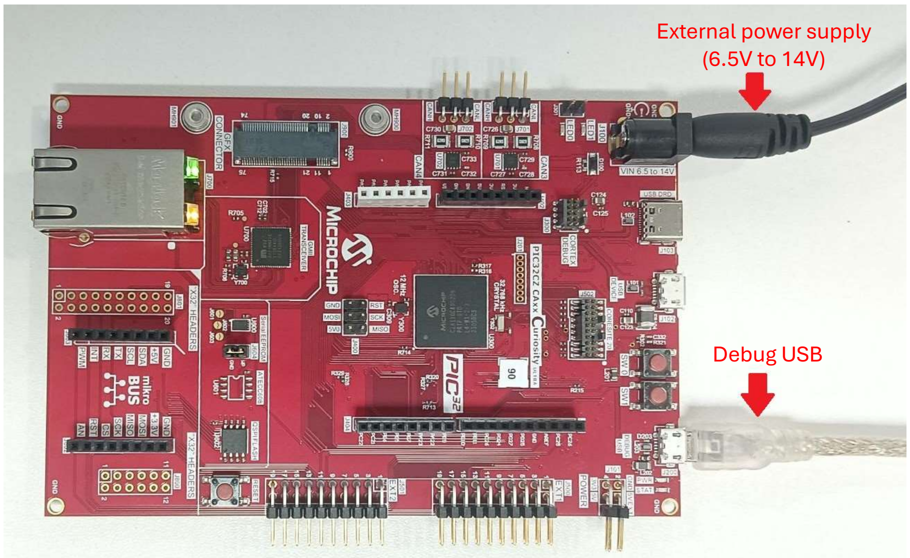
Figure 4-4. Application Logic



5. Building and Debugging the Application

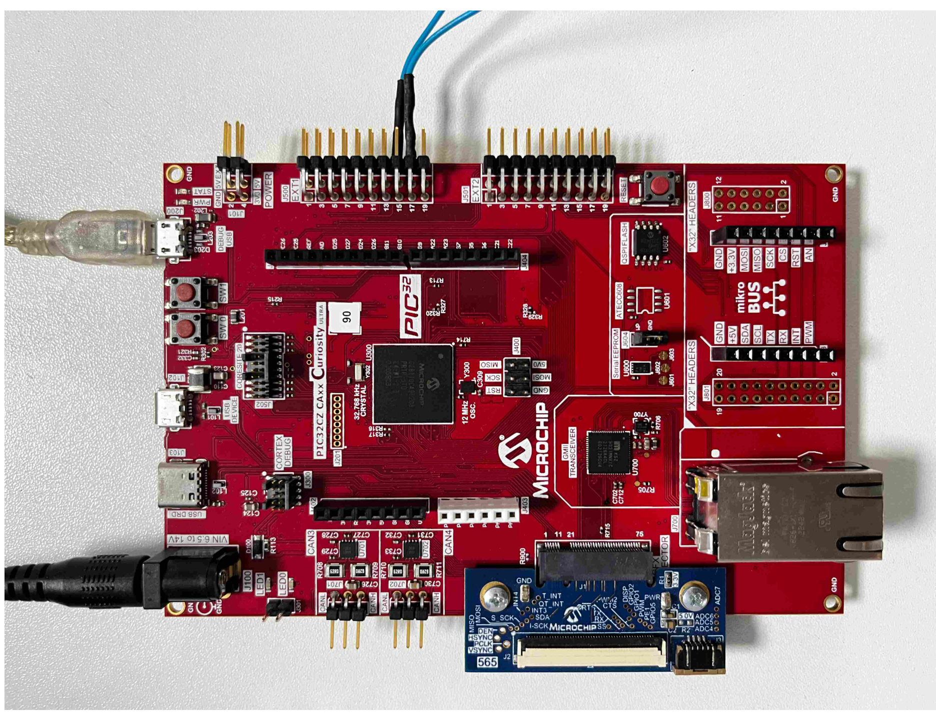
1. The PIC32CZ CA90 Curiosity Ultra Development Board supports debugging using an Embedded Debugger (EDBG). Connect the Type-A male to micro-B USB cable to the micro-B USB port on the PIC32CZ CA90 Curiosity Ultra Development Board and connect the Type-A male end to the PC. Additionally, connect an external power supply (6.5V-14V) to power up the board.

Figure 5-1. Hardware - PIC32CZ CA90 Curiosity Ultra Development Board



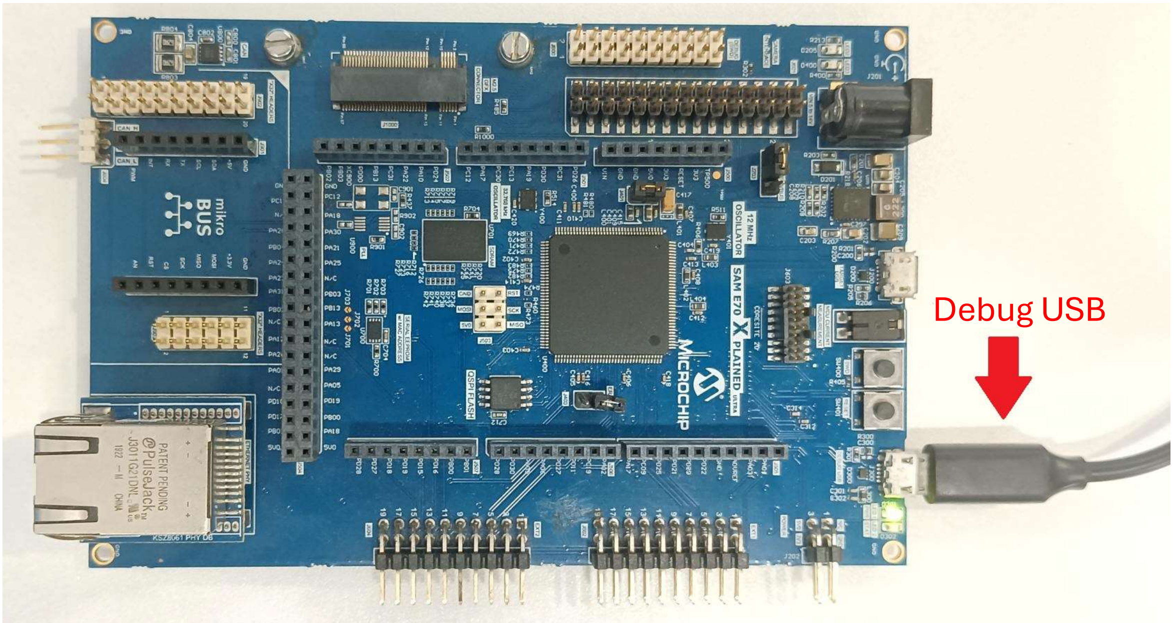
2. Short the MISO (PC11 – Pin 17) and MOSI (PC08 – Pin 16) pins present in EXT1 in the PIC32CZ CA90 Curiosity Ultra Development Board using a wire.

Figure 5-2. Hardware Setup - PIC32CZ CA90 Curiosity Ultra Development Board



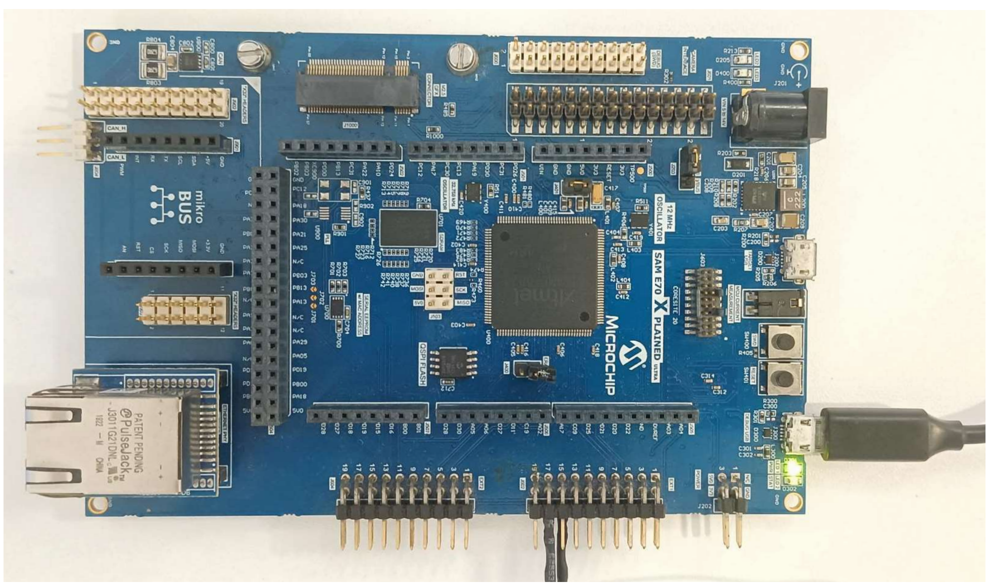
3. Connect the Type-A male to the micro-B USB cable to the micro-B debug USB port to power and debug the SAM E70 Xplained Ultra Evaluation Kit.

Figure 5-3. Hardware - SAM E70 Xplained Ultra Evaluation Kit



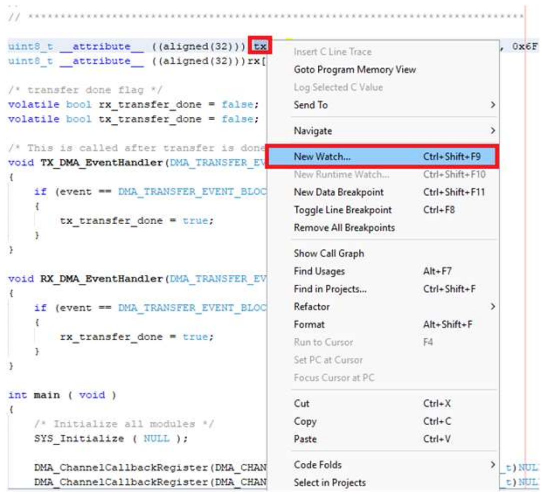
4. Short the MISO (PD20 – Pin 17) and MOSI (PD21 – Pin 16) pins present in EXT1 in the SAM E70 Xplained Ultra Evaluation Kit using a wire.

Figure 5-4. Hardware Setup - SAM E70 Xplained Ultra Evaluation Kit



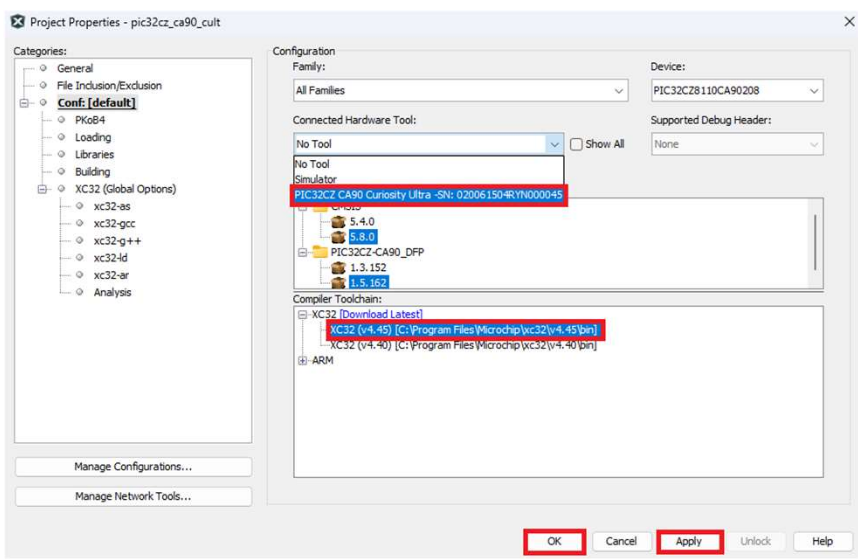
5. Select the **Tx** variable in the code and right-click on the selected text, then select **New Watch**.
   Note: Follow the same for **Rx**.

Figure 5-5. Adding a New Watch



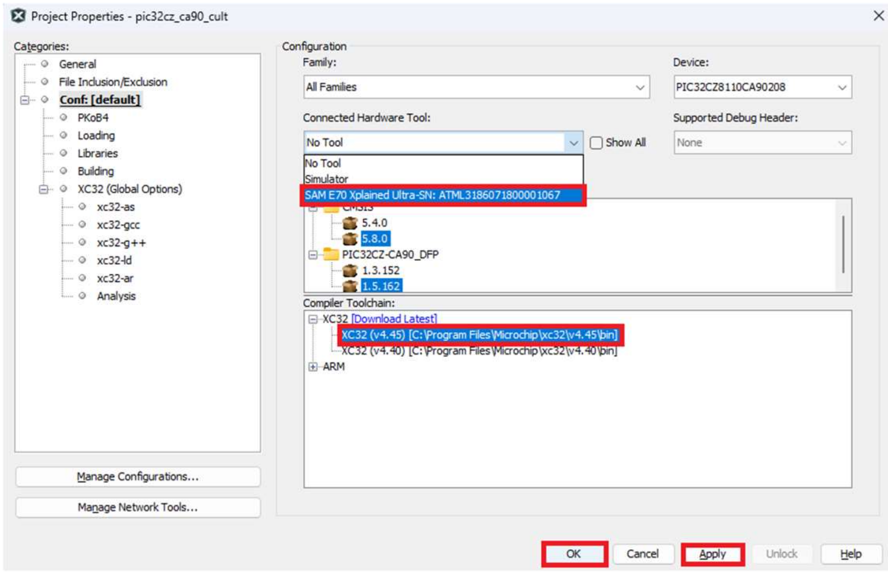
6. In the MPLAB X IDE Project Properties window perform these actions.
   a. Under the left Categories section, select **Conf: [default]**, and in the right **Configuration** properties page, select the Connected Hardware Tool and Compiler Toolchain as shown below.

Figure 5-6. Project Properties - PIC32CZ CA90 Curiosity Ultra Development Board



7. For SAM E70 Xplained Ultra Evaluation Kit, the hardware tool needs to be changed as follows.

Figure 5-7. Project Properties - SAM E70 Xplained Ultra Evaluation Kit



Note: The following steps are applicable for both the boards (the PIC32CZ CA90 Curiosity Ultra Development Board and the SAM E70 Xplained Ultra Evaluation Kit).

8. Click **Apply** and then click **OK**.
9. Click on the highlighted position as shown below to put a breakpoint.
   Note: Breakpoint is used in the reception transfer handler to visualize the received data.

Figure 5-8. Adding the Breakpoint in Reception Complete

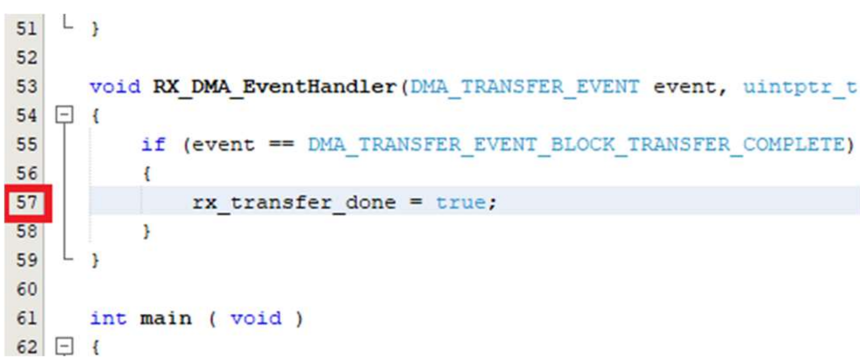
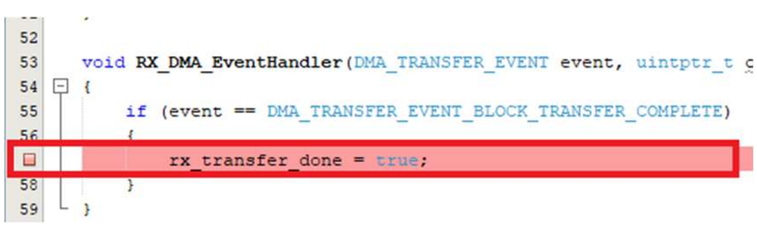


Figure 5-9. Breakpoint on Receive Complete Callback



10. Click on the **Debug main Project** button.

Figure 5-10. Debug Main Project

Note: If the Variable window does not appear, go to *Window* > *Debugging* > *Variables* to open the Variable window.

Figure 5-11. Window Menu details

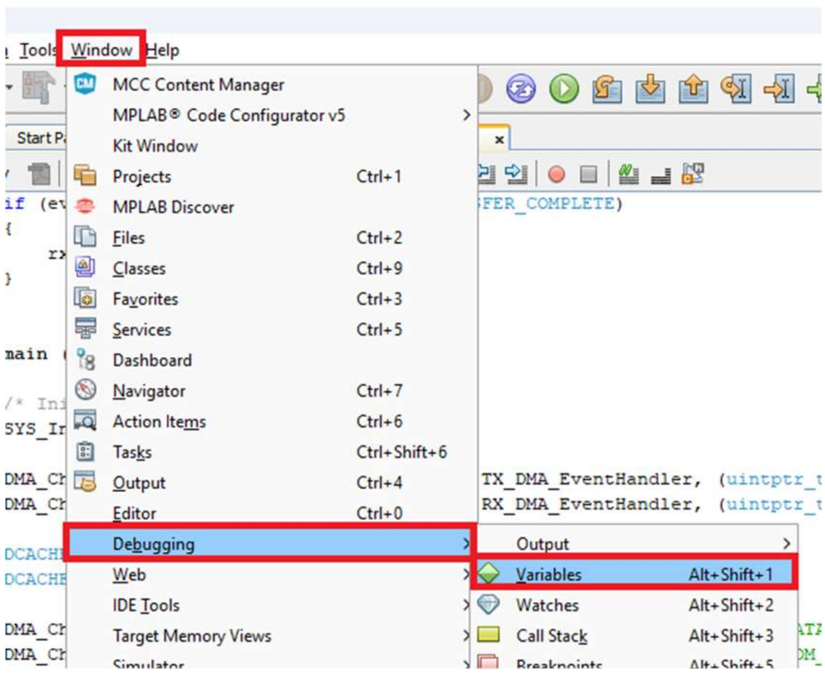
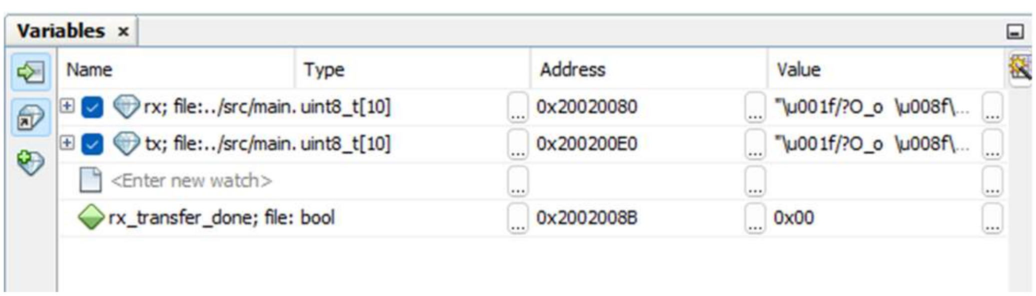
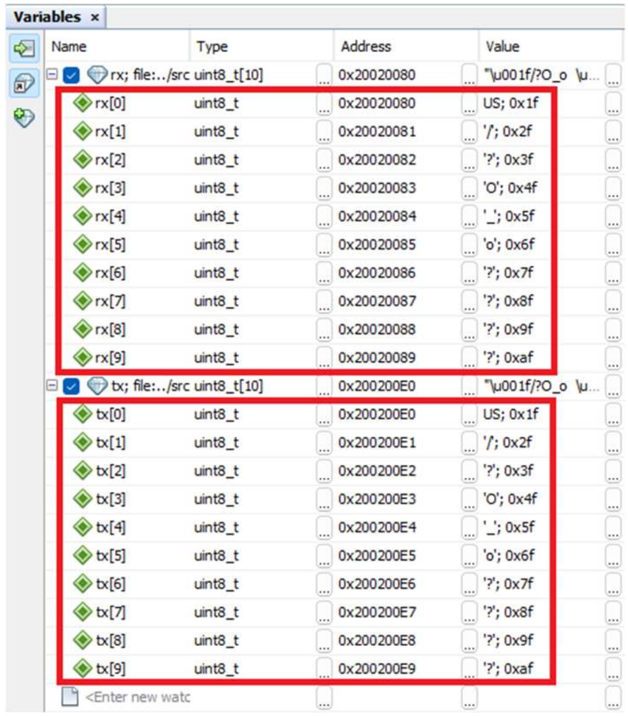


Figure 5-12. Variables Window



11. The transmitted data (Tx) matches with the received data (Rx).

Figure 5-13. Output



6. References

- PIC32CZ CA90 Curiosity Ultra Development Board
- SAM E70 Xplained Ultra Evaluation Kit
- PIC32CZ CA80/CA90 Curiosity Ultra User Guide (*DS70005522*)
- SAM E70 Xplained Ultra User Guide (*DS70005389)*
- Demonstrating Application Development with PIC32CZ CA90 Curiosity Ultra Evaluation Board
- Getting Started Extended Application on PIC32CZ CA90 Curiosity Ultra Development Board
- Create Your First Project with SAM E70 using MPLAB® Harmony v3
- Getting Started with MPLAB® Harmony v3 Drivers and System Services on SAM E70/S70/V70/V71 MCUs
- For additional information about 32-bit Microcontroller Collaterals and Solutions, refer to: 32-bit Microcontroller Collateral and Solutions Reference Guide (*DS70005534*)
- For additional information on MPLAB® Harmony v3, refer to the Microchip web site: https://www.microchip.com/en-us/tools-resources/configure/mplab-harmony and https://developerhelp.microchip.com/xwiki/bin/view/software-tools/harmony/
- For more information on various applications, refer to: github.com/Microchip-MPLAB-Harmony/reference_apps
- For other relevant information, refer to the Microchip web site: www.microchip.com/

7. Revision History

7.1 Revision A - January 2025

This is the initial release of this document.

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: 979-8-3371-0498-0

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

ATSAME51G18A, ATSAME51G19A, ATSAME51J18A, ATSAME51J19A, ATSAME51J20A, ATSAME51N19A, ATSAME51N20A, ATSAME53J18A, ATSAME53J19A, ATSAME53J20A, ATSAMS70J19, ATSAMS70J20, ATSAMS70J21, ATSAMS70N19, ATSAMS70N20, ATSAMS70N21, ATSAMS70Q19, ATSAMS70Q20, ATSAMS70Q21, ATSAMV70J19, ATSAMV70J20, ATSAMV70N19, ATSAMV70N20, ATSAMV70Q19, ATSAMV70Q20, ATSAMV71J19, ATSAMV71J20, ATSAMV71J21, ATSAMV71N19, PIC32CZ2051CA70064, PIC32CZ2051CA70100, PIC32CZ2051CA70144, PIC32CZ2051CA80100, PIC32CZ2051CA80144, PIC32CZ2051CA80176, PIC32CZ2051CA80208, PIC32CZ2051CA90100, PIC32CZ2051CA90144, PIC32CZ2051CA90176