SERCOM USART on Microchip Cortex Devices (SAM and PIC32C)

AN5807

www.microchip.com
Product Page Links

Introduction

This application note explains the various features of SERCOM USART in PIC32C and SAM family of devices that are powered by Arm® Cortex®-M, such as SAM D, SAM E, SAM L, PIC32CK, PIC32CM, PIC32CX, and PIC32CZ, and its configurations with example codes and corresponding outputs.

The examples discussed in this document uses two SAM D21 Curiosity Nano Evaluation Kits and two SAM E70 Xplained Ultra Evaluation Kits.

Note: The same configuration and steps can be easily replicated for other Arm families as shown in SERCOM Implementation in SAM D Microcontrollers.

Table of Contents

1. Introduction to Serial Communication Interfaces

The serial communication interface plays a key role in exchanging data between several microcontrollers and other devices in an embedded system. The exchange of data can be half-duplex or full-duplex, depending on the serial module specification. The data rate and connections of the serial module differ from each other. USART, I^2C, SPI, LIN, and LON are the common serial modules used in embedded systems.

1.1 USART

The Universal Synchronous/Asynchronous Receiver/Transmitter (USART) is an integral communication module embedded within microcontrollers to facilitate serial data exchange. This versatile component can manage both synchronous and asynchronous serial protocols, transmitting data sequentially, bit by bit.

USARTs offer a high degree of configurability, allowing for an array of data frame structures, baud rates, and modes of operation. It is a full-duplex in operation. This adaptability renders them highly suitable for interfacing with various devices, including sensors, computers, and additional microcontrollers.

1.2 I^2C

I^2C is a two-wire protocol. I^2C is a multi-host bus that provides arbitration and collision detection. It is half-duplex in communication. Different transfer rates are available depending on the speed mode. The I^2C speed rate is higher than the USART, but less than the SPI. I^2C is mainly preferred in embedded applications where a limited number of pins are available for communication and several devices must be connected in a single bus.

1.3 SPI

SPI is a synchronous serial bus using four physical lines for its communication. It is a full-duplex in operation. SPI supports higher data rates. The SPI can operate with a single host device and one or more client devices, each with separate chip select lines.

1.4 LIN

The Local Interconnect Network (LIN) mode facilitates connectivity between the Host node and the Client node on a LIN bus. The LIN is a serial communication protocol designed to effectively manage the control of mechatronic nodes within distributed automotive systems. LIN offers a cost-effective solution for bus communication in scenarios where the higher bandwidth and flexibility of CAN are not necessary.

1.5 LON

The Local Operating Network (LON) mode facilitates integration with the LON infrastructure. This mode encompasses the comprehensive OSI (Open Systems Interconnection) model, addressing all seven layers from the physical connections, including wired, power line, radio frequency, and Internet Protocol, to the application layer, and the intermediate layers. Engineered from the ground up to serve as a robust platform for control communications, the LON mode allows for the efficient exchange of Physical Protocol Data Unit (PPDU) frames, requiring minimal microprocessor involvement.

2. SERCOM Implementation in SAM D Microcontrollers

Generally, the microcontroller will have separate serial communication modules with different pinouts for each module. Separate dedicated peripherals and user registers will be available for each module. For example, USART will be a separate peripheral with dedicated pins for its function and I^2C will be a separate peripheral with its dedicated pins.

In SAM D microcontrollers, all serial peripherals are integrated into a single module, functioning as a serial communication interface (SERCOM). A SERCOM module can be configured as USART, I^2C, or SPI selectable by the user. Each SERCOM will be assigned four pads from PAD0 to PAD3. The functionality of each pad is configurable depending on the SERCOM mode used. Unused pads can be used for other purposes and the SERCOM module will not control them unless they are configured to be used by the SERCOM module, for example, SERCOM0 can be configured as USART mode with PAD0 as the transmit pad and PAD1 as the receive pad. Other unused pads (PAD2 and PAD3) can be used as GPIO pins or assigned to some other peripherals. The assignment of SERCOM functionality for different pads is highly flexible, making the SERCOM module more advantageous than the typical serial communication peripheral implementation.

Notes: These configurations can also be implemented in the following device families:

- SAM L family of microcontrollers
- SAM C family of microcontrollers
- SAM D family of microcontrollers
- SAM D5X/E5X family of microcontrollers
- PIC32CK family of microcontrollers
- PIC32CM family of microcontrollers
- PIC32CX family of microcontrollers
- PIC32CZ family of microcontrollers

2.1 Overview

The serial communication interface (SERCOM) can be configured to support three different modes: I^2C, SPI, and USART. Once configured and enabled, all SERCOM resources are dedicated to the selected mode. The SERCOM serial engine consists of a transmitter and receiver, baud-rate generator and address-matching functionality. It can be configured to use the internal generic clock or an external clock, making operation in all sleep modes possible.

2.2 Features

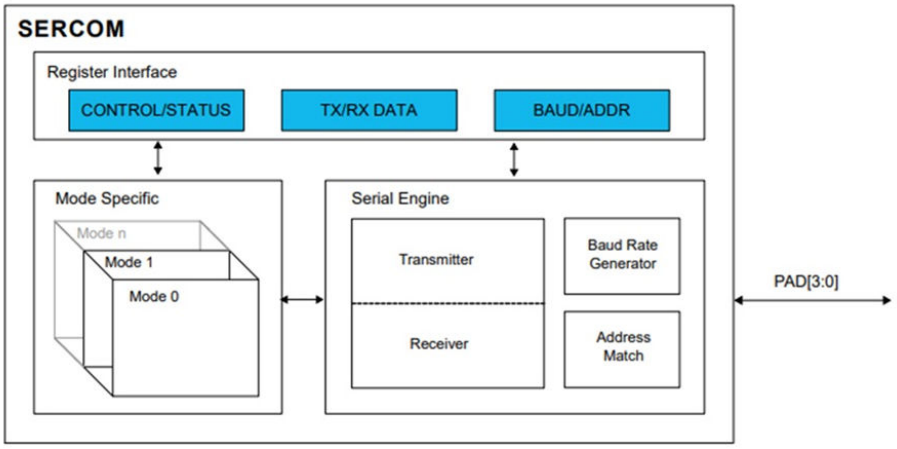
The following are key features of the module:

- The combined interface is configurable as one of the following:
  - I^2C – Two-wire serial interface (SMBus compatible)
  - SPI – Serial Peripheral Interface
  - USART – Universal Synchronous/Asynchronous Receiver/Transmitter
- Single transmit buffer and double receive buffers
- Baud Rate Generator (BRG)
- Address match or mask logic
- Operational in all sleep modes
- Can be used with DMA (not supported in the SAM D20 MCUs)

2.3 Block Diagram

The following figure illustrates the block diagram of the SERCOM module.

Figure 2-1. SERCOM Block Diagram



2.4 Clocks

The SERCOM module requiress the following clocks for its operation:

- SERCOM bus clock
- SERCOM CORE generic clock
- SERCOM SLOW generic clock

By default, the SERCOM bus clock (CLK_SERCOMx_APB) is disabled but can be enabled and disabled in the Power Manager (PM) module. Two generic clocks are used by the SERCOM module: GCLK_SERCOMx_CORE and GCLK_SERCOMx_SLOW. The core clock (GCLK_SERCOMx_CORE) is required to clock the SERCOM while operating as a host, while the slow clock (GCLK_SERCOMx_SLOW) is only required for certain functions like I^2C timeouts.

3. USART Implementation in SAM E70 Microcontroller

3.1 Overview

The Universal Synchronous Asynchronous Receiver Transceiver (USART) offers a comprehensive full-duplex synchronous or asynchronous serial connection. Its data frame configuration is highly customizable, including data length, parity, and stop-bit options to ensure compatibility with a broad range of standards. The receiver is equipped with detection capabilities for parity errors, framing errors, and overrun errors. Additionally, a receiver timeout feature is included to manage variable-length frames effectively, while a transmitter timeguard is in place to facilitate communication with slower remote devices.

The USART is designed with three diagnostic test modes: Remote Loopback, Local Loopback, and Automatic Echo. It supports various operational modes to interface with RS485, LIN, LON, and SPI buses. This includes compatibility with ISO7816 T = 0 or T = 1 smart card protocols, infrared transceivers, and modem port connections. The hardware handshaking capability, utilizing RTS and CTS lines, allows for out-of-band flow control.

In addition, the USART is compatible with Direct Memory Access (DMA) Controller connections, facilitating efficient data transfers to the transmitter and from the receiver. The DMA Controller offers advanced buffer management, which operates without requiring processor intervention.

3.2 Features

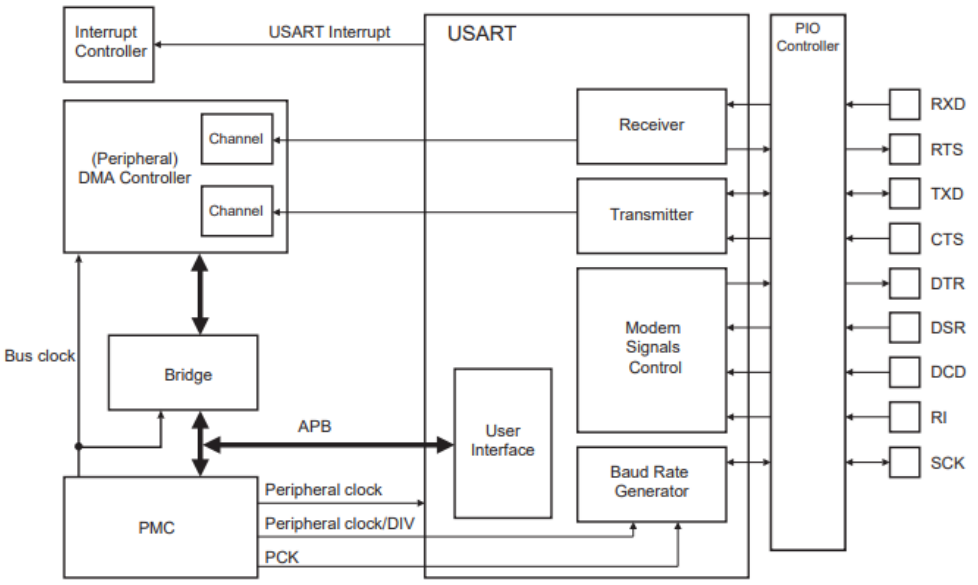
The following are key features of the USART module:

- RS485 with Driver Control Signal
- SPI Mode
- LIN Mode
- LON Mode
- Register Write Protection

3.3 Block Diagram

The following figure illustrates the block diagram of the USART module:

Figure 3-1. USART Block Diagram



3.4 Clocks

The USART is not continuously clocked; instead, it provides the option to select internal clocks from various sources, including the Slow Clock (SLCK), Main Clock (MAINCK), any available Phase-Locked Loop (PLL) clock, and the Host Clock (MCK). This selection is achieved through the configuration of the Programmable Clock (PMC) Control Status Register's (PMC_PCKx) Clock Source Selection (CSS) field.

Additionally, the frequency of these clock sources can be adjusted by setting the PMC_PCKx Prescaler (PRES). Each programmable clock output (PCKx) can be activated or deactivated by writing a '1' to the corresponding Power Management Controller System Clock Enable Register (PMC_SCER.PCKx) or System Clock Disable Register (PMC_SCDR.PCKx). The operational status of these internal clocks is reflected in the PMC System Clock Status Register (PMC_SCSR.PCKx). The PMC Status Register (PMC_SR. PCKRDYx) flag is used to indicate the readiness of the programmable internal clock after being set in the Programmable Clock Registers. For USART operations, PCK4 is designated.

The Power Management Controller (PMC) governs the clocking of embedded peripherals through the PMC Peripheral Control Register (PMC_PCR). This register enables users to manage the activation and deactivation of various peripheral clocks, including the Peripheral Clocks (periph_clk [PID]), which are distributed to each peripheral and derived from the Host Clock (MCK), and the Generic Clocks (GCLK[PID]), specifically allocated to I2SC0 and I2SC1. These clocks function independently from the core and bus clocks (HCLK, MCK, and periph_clk [PID]) and are generated through the selection and division of sources, such as SLCK, MAINCK, UPLLCKDIV, PLLACK, and MCK. To configure a peripheral's clocks, the PMC_PCR.CMD must be set to '1', and the PMC_PCR.PID must be assigned the index of the targeted peripheral. It is imperative that all other configuration fields are accurately defined for proper setup.

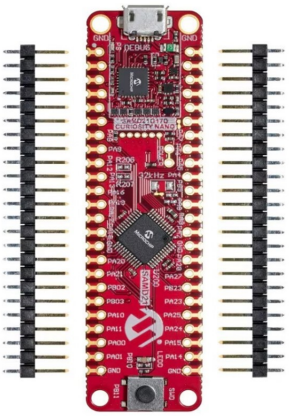
4. Hardware and Software Requirements

The application demonstration needs two SAM D21 Curiosity Nano Evaluation Kits and two SAM E70 Xplained Ultra Evaluation Kits. One kit will be Host/Transmitter, and the other kit will be Client/Receiver.

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

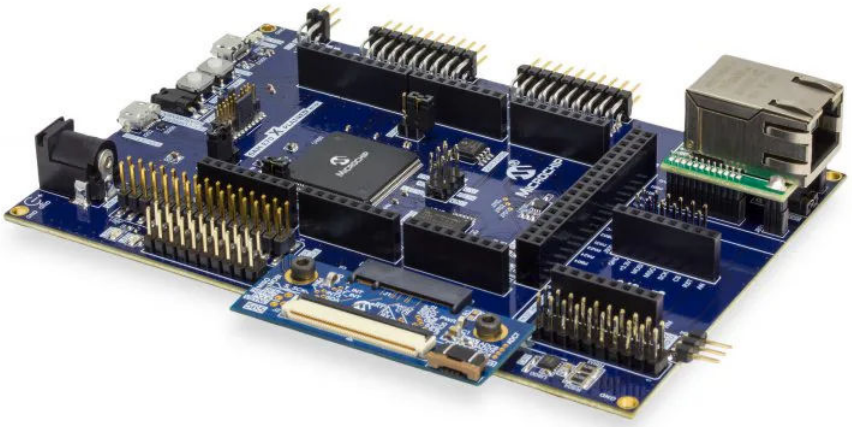
- MPLAB® X IDE v6.20
- MPLAB Code Configurator Plugin v5.5.1
- MPLAB XC32 Compiler v4.45
- csp v3.20.0
- SAM D21 Curiosity Nano Evaluation Kit
- SAM E70 Xplained Ultra Evaluation Kit

Figure 4-1. SAM D21 Curiosity Nano Evaluation Kit



The SAM D21 Curiosity Nano Evaluation Kit is equipped with a USB port, DEBUG USB. For debugging using the embedded debugger EDBG, the DEBUG USB port must be connected.

Figure 4-2. SAM E70 Xplained Ultra Evaluation Kit

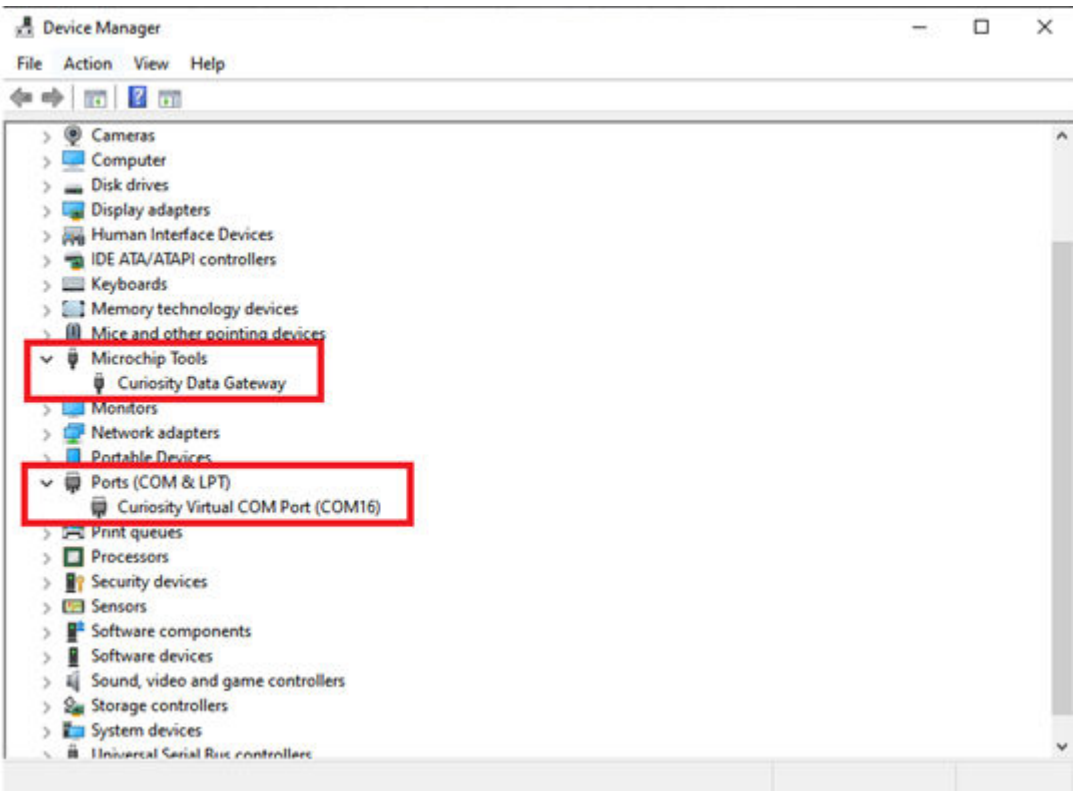


The SAM E70 Xplained Ultra Evaluation Kit features two USB ports: DEBUG USB and TARGET USB. For debugging using the embedded debugger (EDBG), the DEBUG USB port must be connected.

If the driver installation is proper, the EDBG will be listed in the Device manager as shown in the following figure.

Note: The latest versions of the above listed tools can also be used to create the application.

Figure 4-3. Successful EDBG Driver Installation



Note: Search for Device Manager in the search bar of the computer to verify the microchip tools and ports are identified.

5. Application Demonstration

This chapter demonstrates the various features of the SERCOM USART module of the SAM D21 Curiosity Nano Evaluation Kit and SAM E70 Xplained Ultra Evaluation Kit with different example codes.

The following configurations are implemented in the subsequent sections:

- Basic configuration.
- Fraction Baud configuration.
- Hardware Handshaking configuration.
- SOF detection and wakeup configuration (not available in the SAM E70 devices).

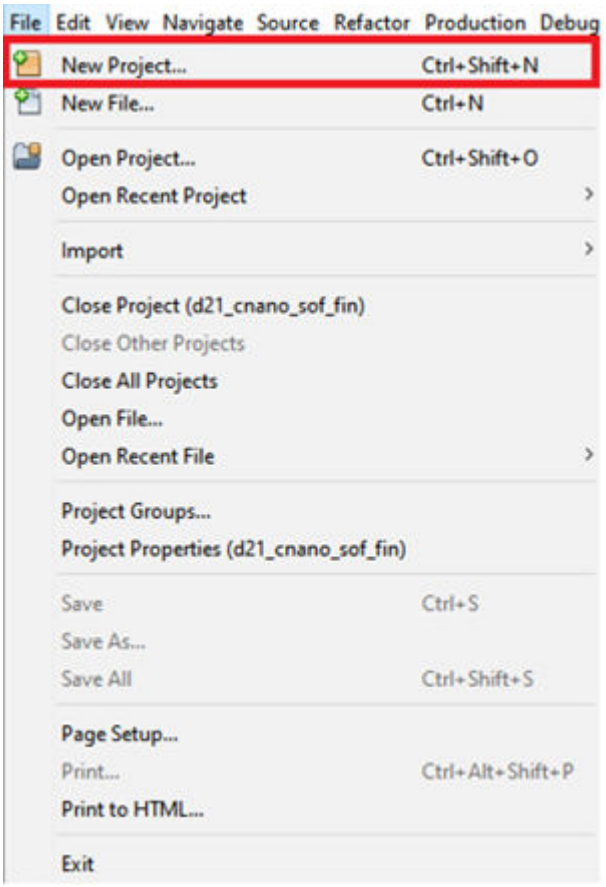
5.1 Creating the Project

Two projects are required for implementing the functionalities: one for host/transmitter and the other for client/receiver.

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, select **New Project** or click on the *New Project* icon.

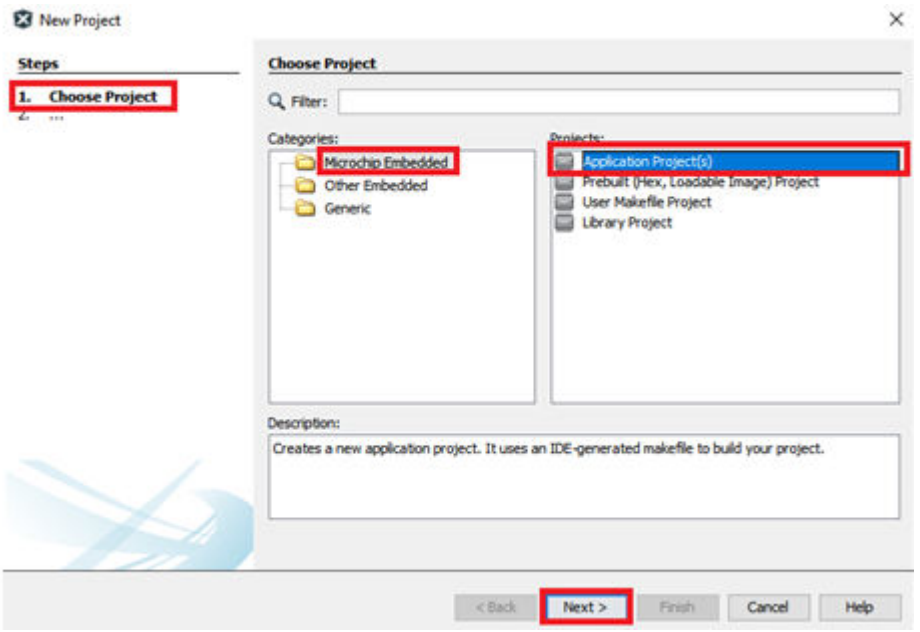
Figure 5-1. New Project in MPLAB X IDE v6.20



3. In the **New Project** window, in the left Navigation bar, under Steps click **Choose Project**.
4. In the **Choose Project** property page:
   a. For categories, select **Microchip Embedded**.

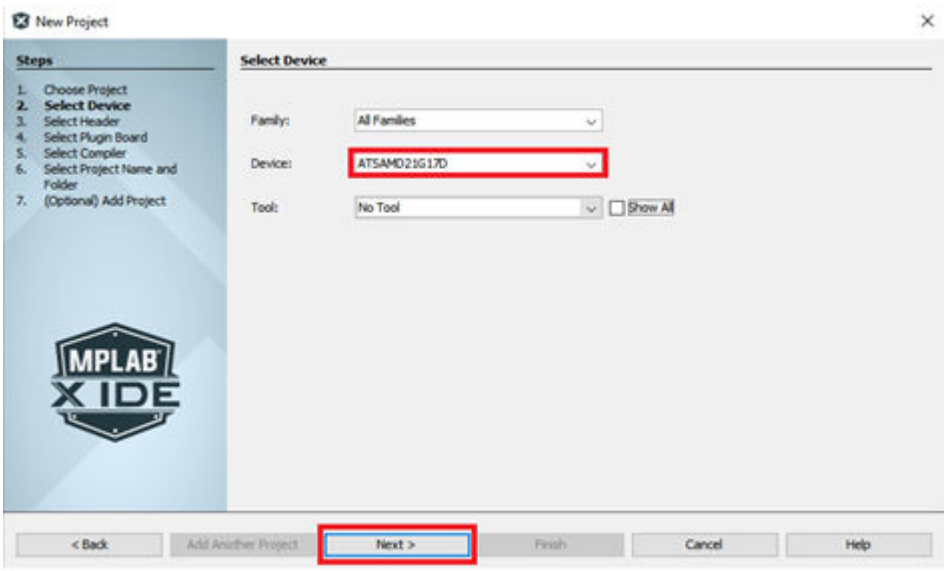
b. For projects, select **Application Project(s)**.

Figure 5-2. Choose Project



5. Click **Next**.
6. In the left Navigation bar, click **Select Device**.
7. In the **Select Device** property page, in the **Device** box, type or select the device **ATSAMD21G17D** for the SAM D21 Curiosity Nano Evaluation Kit.
   Note: For the SAM E70 Xplained Ultra Evaluation Kit, select the device ATSAME70Q21B.

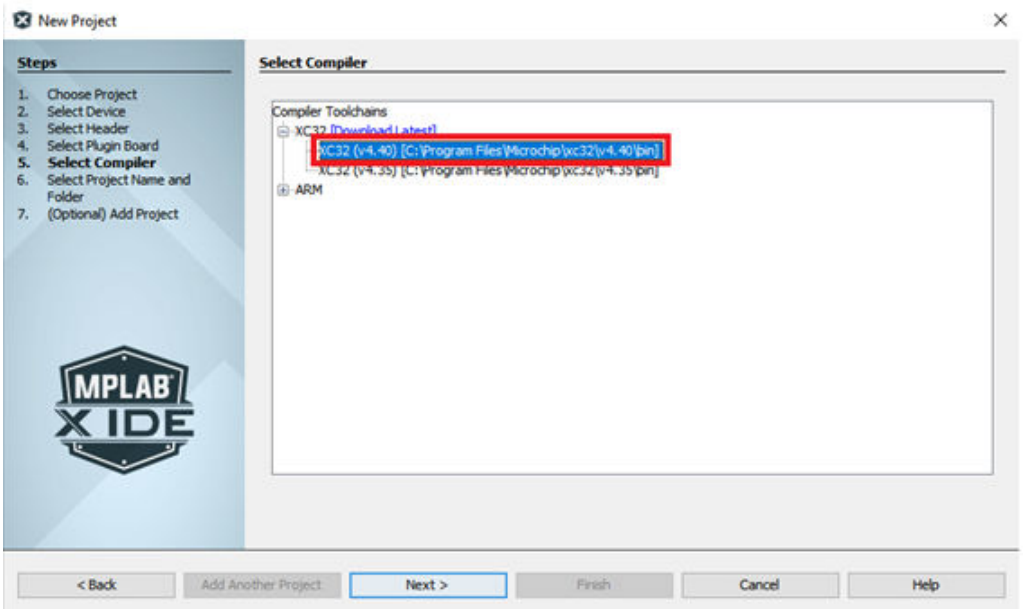
Figure 5-3. Device Selection



8. Click **Next**.
9. In the left Navigation bar, click **Select Compiler**.

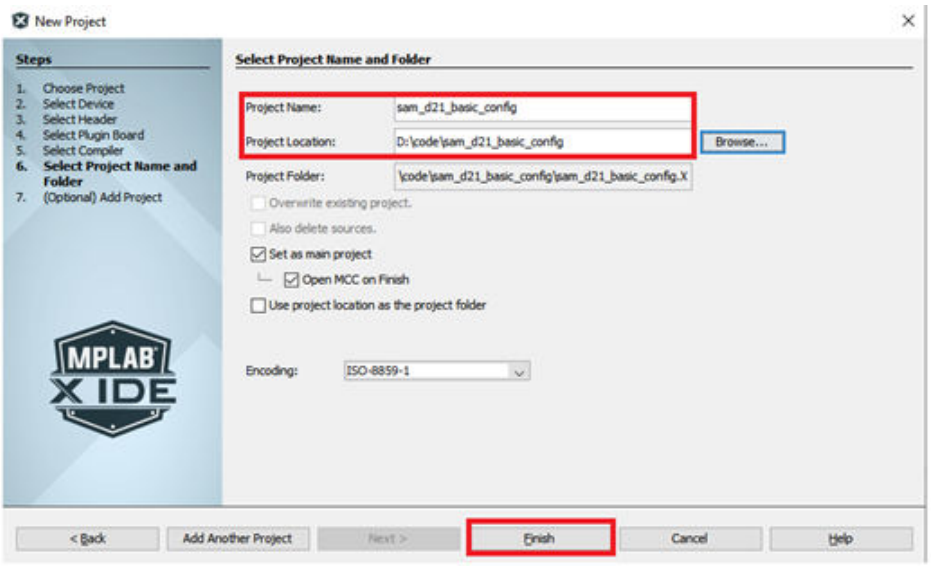
10. In the **Select Compiler** property page, under Compiler Toolchains, click and expand XC32 list of options, and then select **XC32 (v4.45)**.

Figure 5-4. XC32 Compiler Selection



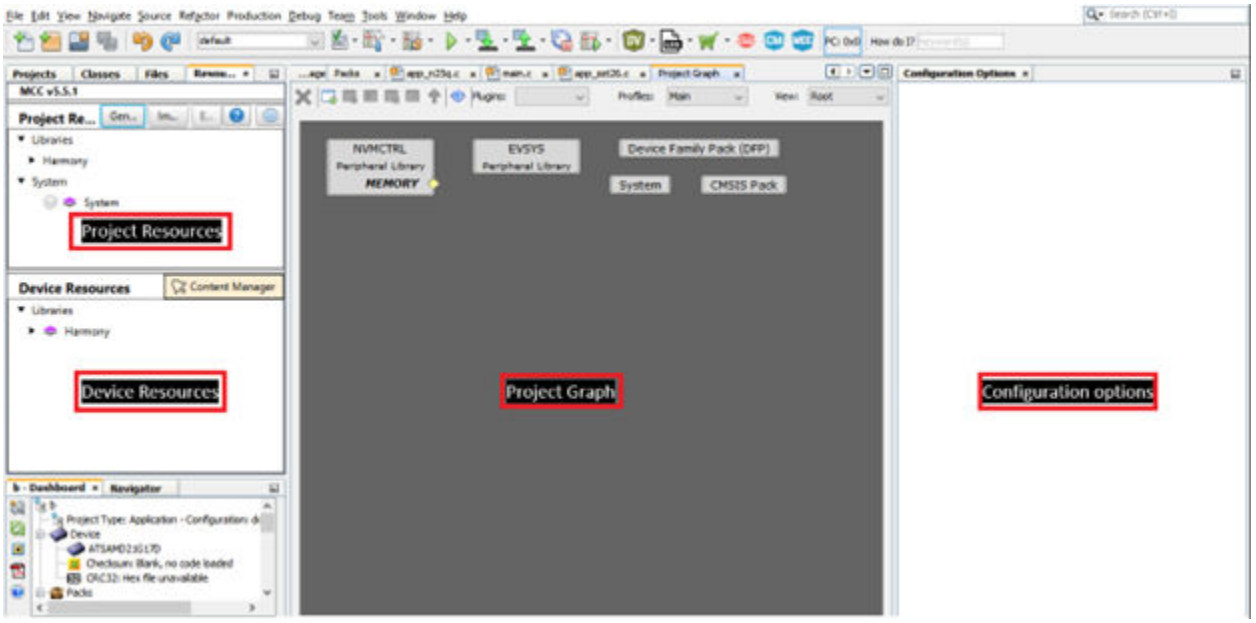
11. Click **Next**.
12. In the left Navigation bar, click **Select Project Name and Folder**.
13. In the **Select Project Name and Folder** property page, enter the Project Name and Project Location.

Figure 5-5. Project Name and Folder Settings



14. Click **Finish**. Then, MPLAB Code Configurator Window will be displayed. It comprises of Project Resources, Device Resources, Project Graph, and Configuration options.

Figure 5-6. MPLAB Code Configurator Window



Note: The procedural steps mentioned in the section Creating the Project can be used to create a project on the SAM E70 Xplained Ultra Evaluation Kit with the changes mentioned in Step 6.

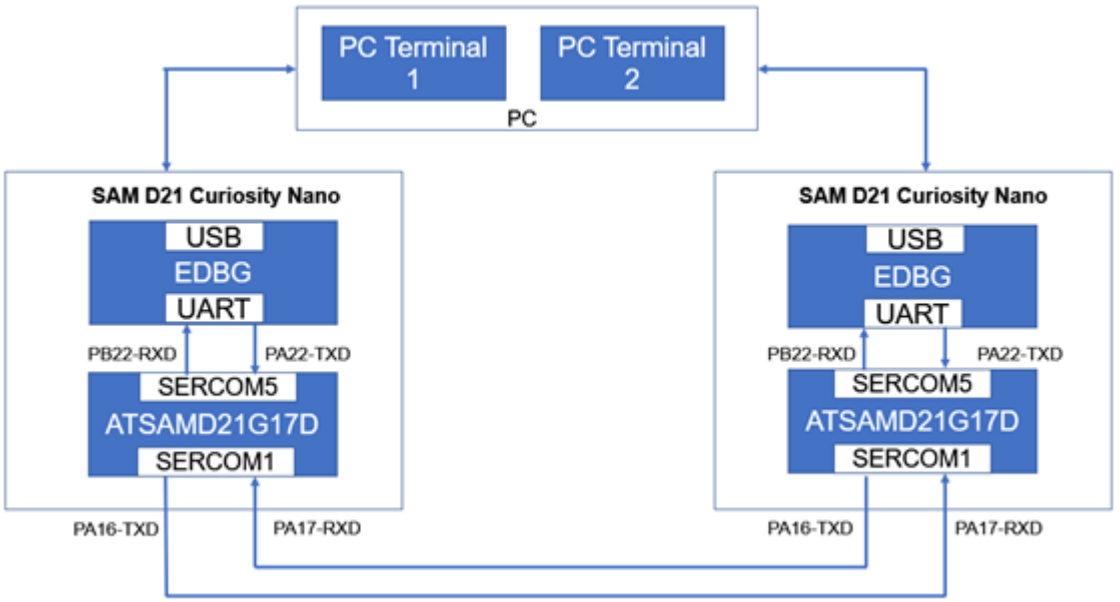
5.2 Basic Configuration

5.2.1 SAM D21 Curiosity Nano Evaluation Kit

Two SAM D21 Curiosity Nano Evaluation Kits are connected to each other by the SERCOM USART lines (TxD, RxD) and it is connected to the PC terminal through the EDBG port.

Note: Click here to access the source code for this application configuration. Alternatively, it is also available in the GitHub reference_apps repository.

Figure 5-7. Block Diagram (SAM D21 Curiosity Nano Evaluation Kit)



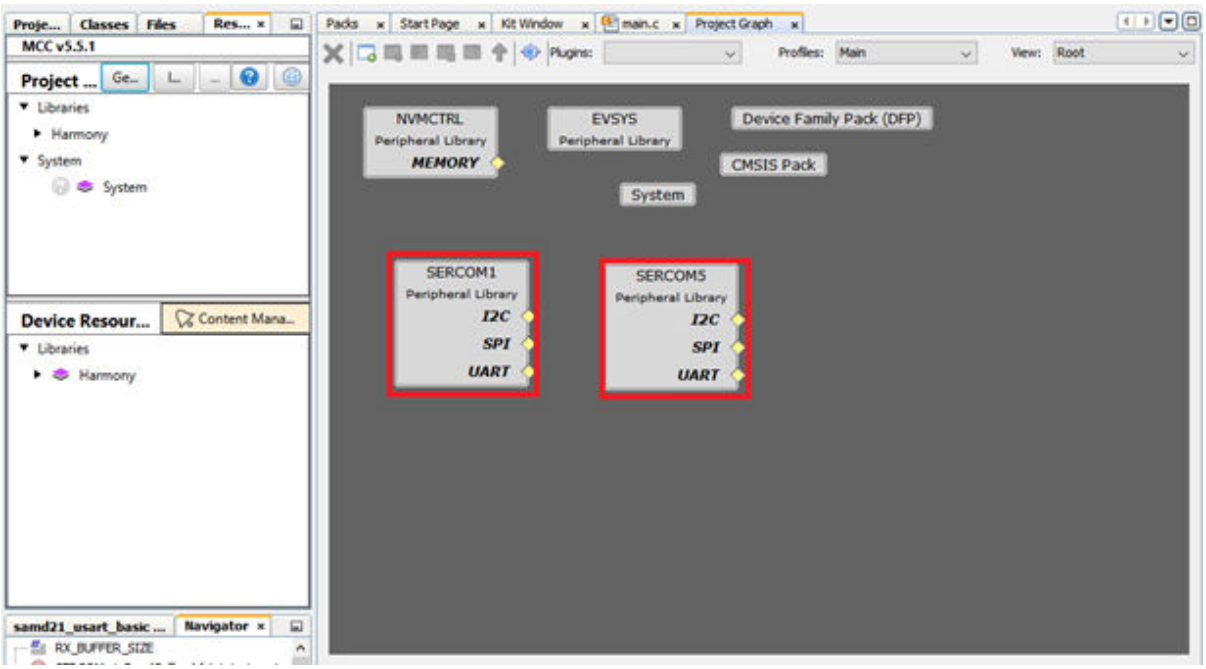
To add and configure MPLAB Harmony components using MCC, follow these steps:

1. To create the project, refer to Creating the Project.
2. In the MCC window, click **Project Graph**.
3. Under **Device Resources**, click and expand the list of options *Harmony* > *Peripherals* > *SERCOM.*

Figure 5-8. Device Resources

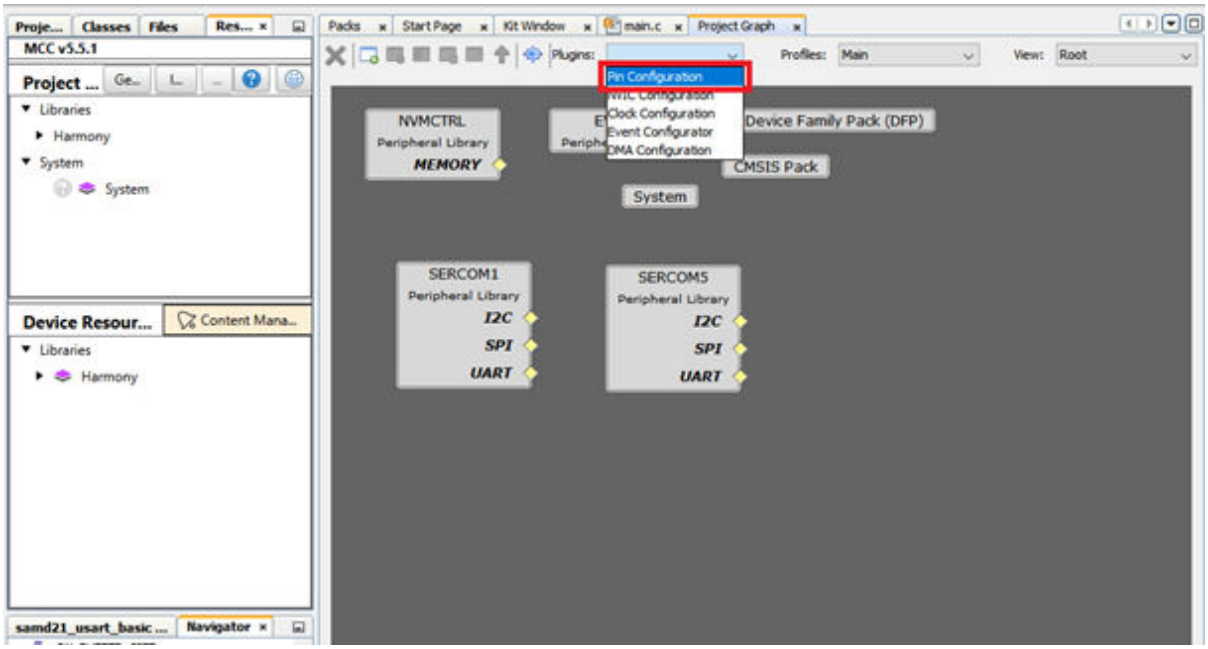
4. Click **SERCOM1** and **SERCOM5**.
5. Observe that the SERCOM1 and SERCOM5 Peripheral Library blocks are added in the Project Graph window.

Figure 5-9. SERCOM1 and SERCOM5 Added in the Project Graph Window



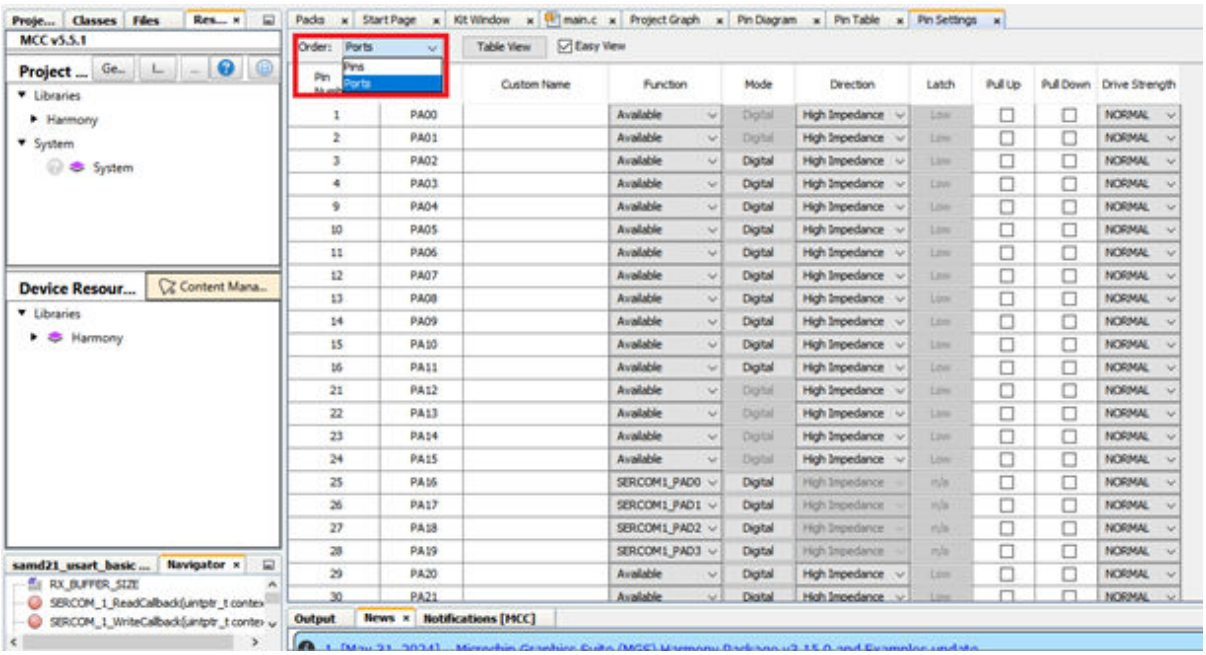
6. From the **Plugins** drop-down list, select **Pin Configuration** and then click **Pin Settings** (see Figure 5-12).

Figure 5-10. Select the Plugin



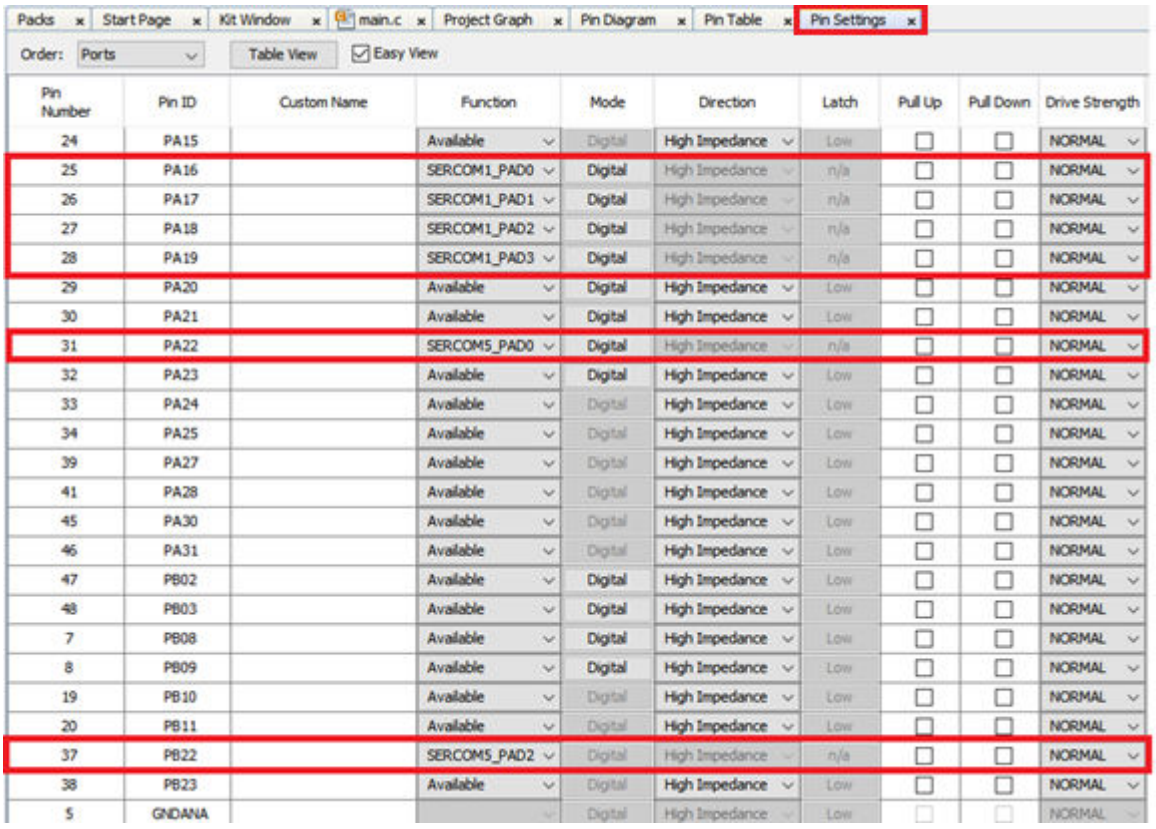
7. From the **Order** drop-down list, select **Ports** to build configurations according to the application as shown below.

Figure 5-11. Pin Settings



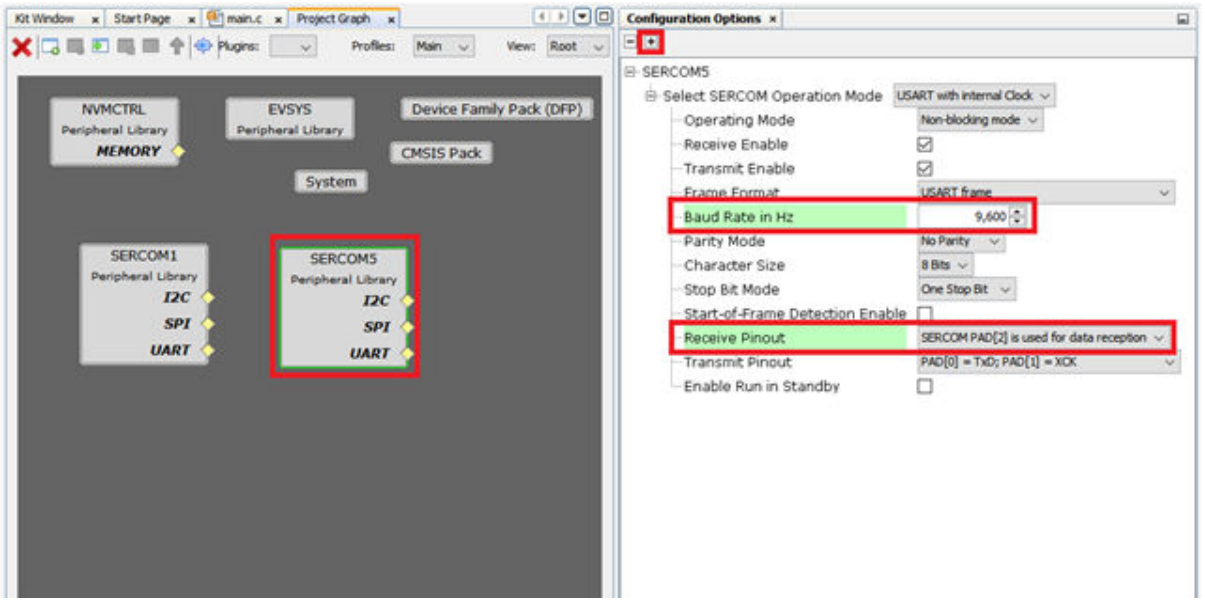
8. Configure the following pins PA16, PA17, PA18, PA19, PA22, and PB22 as SERCOM1_PAD0, SERCOM1_PAD1, SERCOM1_PAD2, SERCOM1_PAD3, SERCOM5_PAD0, and SERCOM5_PAD2, respectively.

Figure 5-12. Pin Configuration



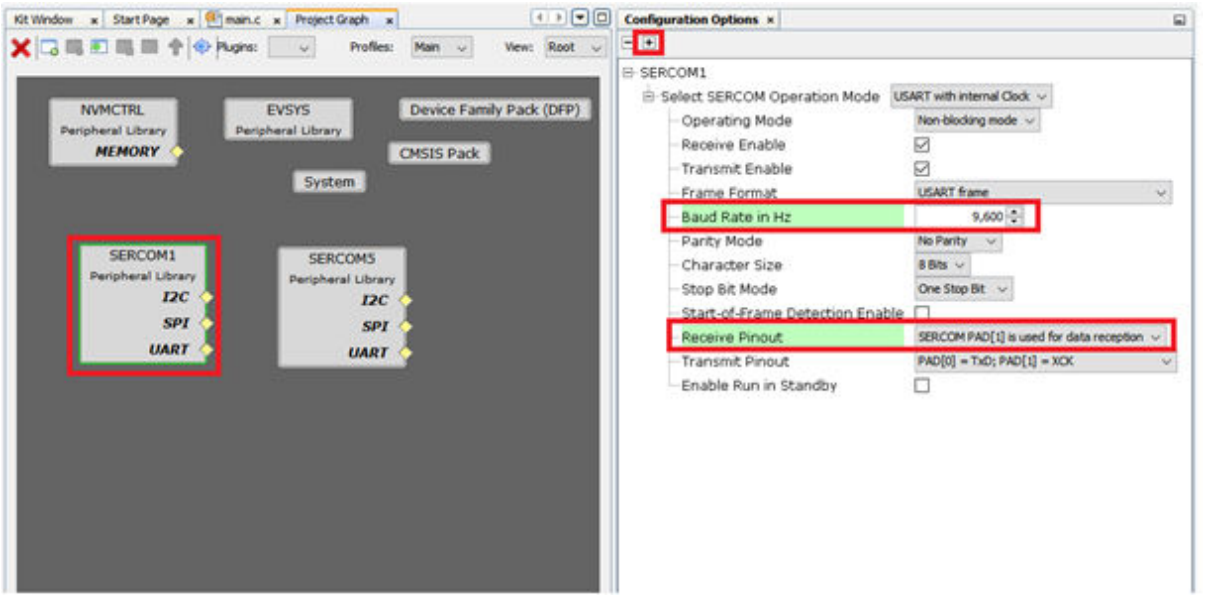
9. In the **Project Graph** window, in the left Navigation bar, select **SERCOM5 Peripheral Library** and in the right **Configuration Options** property page, configure it as showin below to print the data on the Serial Console at 9600 baud rate.

Figure 5-13. Altering the PAD and Baud Rate in SERCOM5



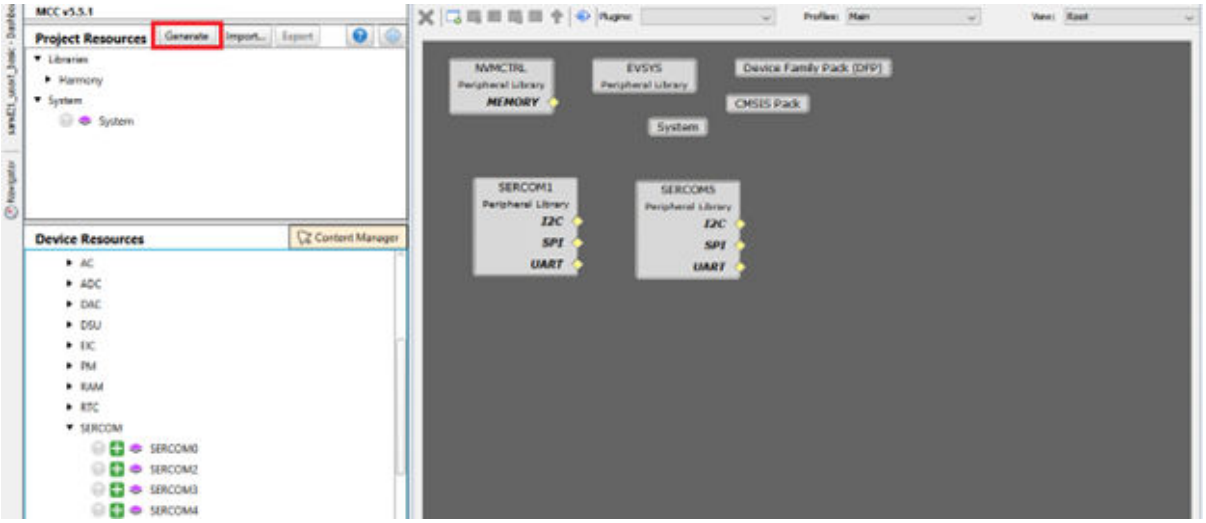
10. Select **SERCOM1 Peripheral Library** and in the right **Configuration Options** property page, configure it as shown below to print the data on the Serial Console at 9600 baud rate.

Figure 5-14. Altering the PAD and Baud Rate in SERCOM1



11. After configuring the peripherals, as shown in the following figure, click **Generate** under Project Resources.

Figure 5-15. Generation of Code



Note: SERCOM5 is connected to EDBG USART lines through which the SAM D21 Curiosity Nano Evaluation Kit will communicate with the PC terminal application.

12. The `main.c` file will be generated and the logic of the application can be implemented.

5.2.1.1 Application Logic

To develop and run the application, follow these steps:

1. Open the `main.c` file of the project located in the ource files folder. Add the following code outside the main() function:

```
#define RX_BUFFER_SIZE 1

volatile bool SERCOM_1_readStatus = false;
volatile bool SERCOM_1_writeStatus = false;

volatile bool SERCOM_5_readStatus = false;
volatile bool SERCOM_5_writeStatus = false;

void SERCOM_1_WriteCallback(uintptr_t context)
{
    SERCOM_1_writeStatus = true;
}

void SERCOM_1_ReadCallback(uintptr_t context)
{
    SERCOM_1_readStatus = true;
}

void SERCOM_5_WriteCallback(uintptr_t context)
{
    SERCOM_5_writeStatus = true;
}

void SERCOM_5_ReadCallback(uintptr_t context)
{
    SERCOM_5_readStatus = true;
}
```

2. In the following code example, the read callback and write callback are declared. These functions set the pointer to a client function to be called when the given USART's write or read event occurs. These callbacks describe the pointer to the function to be called when the write or read event has completed. By setting this to NULL, it can be disabled.

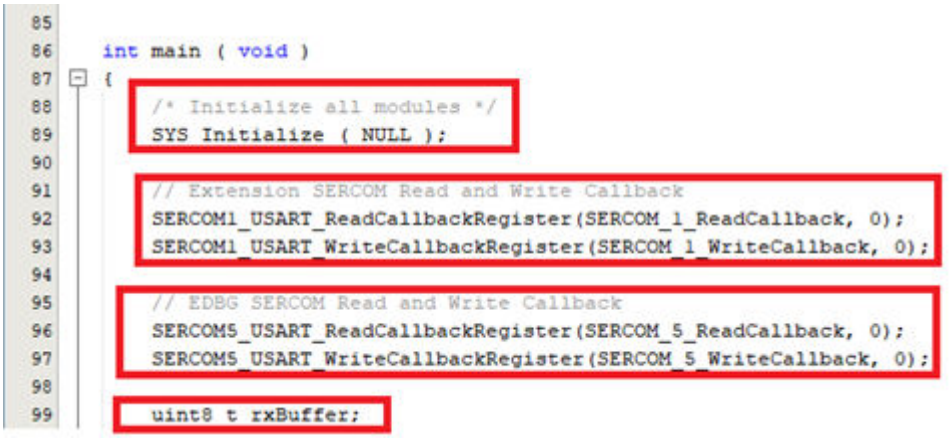
Figure 5-16. Event Handlers

```
57  #define RX_BUFFER_SIZE 1
58
59  volatile bool SERCOM_1_readStatus = false;
60  volatile bool SERCOM_1_writeStatus = false;
61
62  volatile bool SERCOM_5_readStatus = false;
63  volatile bool SERCOM_5_writeStatus = false;
64
65  void SERCOM_1_WriteCallback(uintptr_t context)
66  {
67      SERCOM_1_writeStatus = true;
68  }
69
70  void SERCOM_1_ReadCallback(uintptr_t context)
71  {
72      SERCOM_1_readStatus = true;
73  }
74
75  void SERCOM_5_WriteCallback(uintptr_t context)
76  {
77      SERCOM_5_writeStatus = true;
78  }
79
80  void SERCOM_5_ReadCallback(uintptr_t context)
81  {
82      SERCOM_5_readStatus = true;
83  }
```

3. In Figure 5-17, `SYS_Initialize (NULL)` initializes the modules, such as ports, clock, SERCOM USART, Nested Vector Interrupt Controller (NVIC), and Non-Volatile Memory Controller (NVMCTRL). Also, the callback register functions for read and write operations are declared.
4. Inside the main() function, add the following code:

```
/* Initialize all modules */
    SYS_Initialize ( NULL );

    // Extension SERCOM Read and Write Callback
    SERCOM1_USART_ReadCallbackRegister(SERCOM_1_ReadCallback, 0);
    SERCOM1_USART_WriteCallbackRegister(SERCOM_1_WriteCallback, 0);

    // EDBG SERCOM Read and Write Callback
    SERCOM5_USART_ReadCallbackRegister(SERCOM_5_ReadCallback, 0);
    SERCOM5_USART_WriteCallbackRegister(SERCOM_5_WriteCallback, 0);

    uint8_t rxBuffer;
```

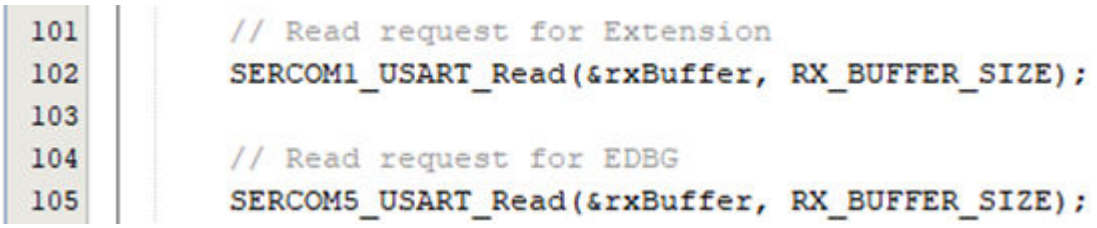
Figure 5-17. Initialization of Modules



5. Start the implementation with a read request for SERCOM1 (for Extension) and SERCOM5 (for EDBG).

```
// Read request for Extension
    SERCOM1_USART_Read(&rxBuffer, RX_BUFFER_SIZE);

// Read request for EDBG
    SERCOM5_USART_Read(&rxBuffer, RX_BUFFER_SIZE);
```

Figure 5-18. Read Requests



6. In Figure 5-19, if the Extension is ready to read (Rx) the data (i.e., `SERCOM_1_readStatus == true`), then it transmits the received data to EDBG. And, if the Extension is ready to write (Tx) the data (i.e., `SERCOM_1_writeStatus == true`), then EDBG will start a read (Rx) request. Similarly, vice versa if the EDBG is ready to read (Rx) or if the EDBG is ready to write (Tx). Add the following code inside the while loop:

```
if(SERCOM_1_readStatus == true)
        {
            SERCOM_1_readStatus = false;

            //Transmit received bytes from EDBG
            SERCOM5_USART_Write(&rxBuffer, RX_BUFFER_SIZE);
        }
if(SERCOM_5_writeStatus == true)
        {
            SERCOM_5_writeStatus = false;
            SERCOM1_USART_Read(&rxBuffer, RX_BUFFER_SIZE);
        }

if(SERCOM_5_readStatus == true)
        {
            SERCOM_5_readStatus = false;

            //Transmit received bytes from Extension
            SERCOM1_USART_Write(&rxBuffer, RX_BUFFER_SIZE);
        }

if(SERCOM_1_writeStatus == true)
        {
            SERCOM_1_writeStatus = false;
```

```
            SERCOM5_USART_Read(&rxBuffer, RX_BUFFER_SIZE);
        }
```

Figure 5-19. Implementing the Basic Configuration

```
106     while ( true )
107     {
108         if(SERCOM_1_readStatus == true)
109         {
110             SERCOM_1_readStatus = false;
111
112             //Transmit received bytes from EDBG
113             SERCOM5_USART_Write(&rxBuffer, RX_BUFFER_SIZE);
114         }
115
116         if(SERCOM_5_writeStatus == true)
117         {
118             SERCOM_5_writeStatus = false;
119             SERCOM1_USART_Read(&rxBuffer, RX_BUFFER_SIZE);
120         }
121
122         if(SERCOM_5_readStatus == true)
123         {
124             SERCOM_5_readStatus = false;
125
126             //Transmit received bytes from Extension
127             SERCOM1_USART_Write(&rxBuffer, RX_BUFFER_SIZE);
128         }
129
130         if(SERCOM_1_writeStatus == true)
131         {
132             SERCOM_1_writeStatus = false;
133             SERCOM5_USART_Read(&rxBuffer, RX_BUFFER_SIZE);
134         }
135         SYS_Tasks ( );
136     }
```

Figure 5-20. Host Side

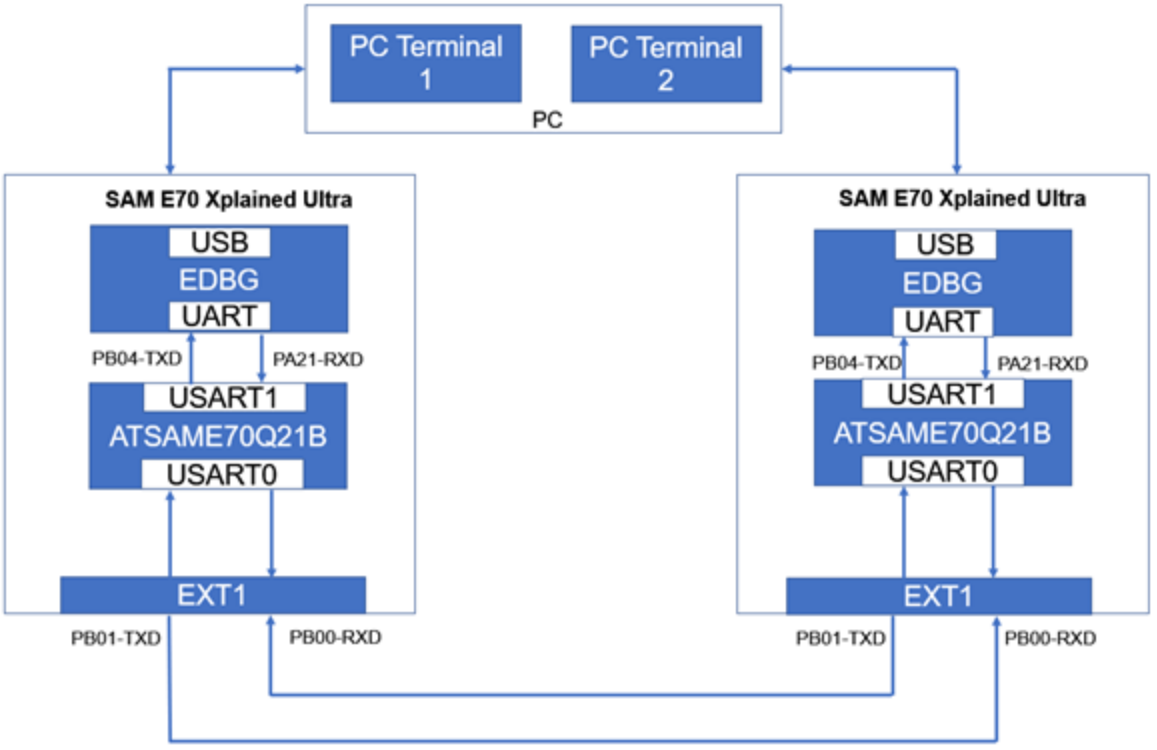
Figure 5-21. Client Side

5.2.2 SAM E70 Xplained Ultra Evaluation Kit

Two SAM E70 Xplained Ultra Evaluation Kits are interconnected by SERCOM USART lines (TxD, RxD) through the EXT1 connector, and connected to the PC terminal through the EDBG port. The following figure illustrates the block diagram.

Note: Click here to access the source code for this application configuration. Alternatively, it is also available in the GitHub reference_apps repository.

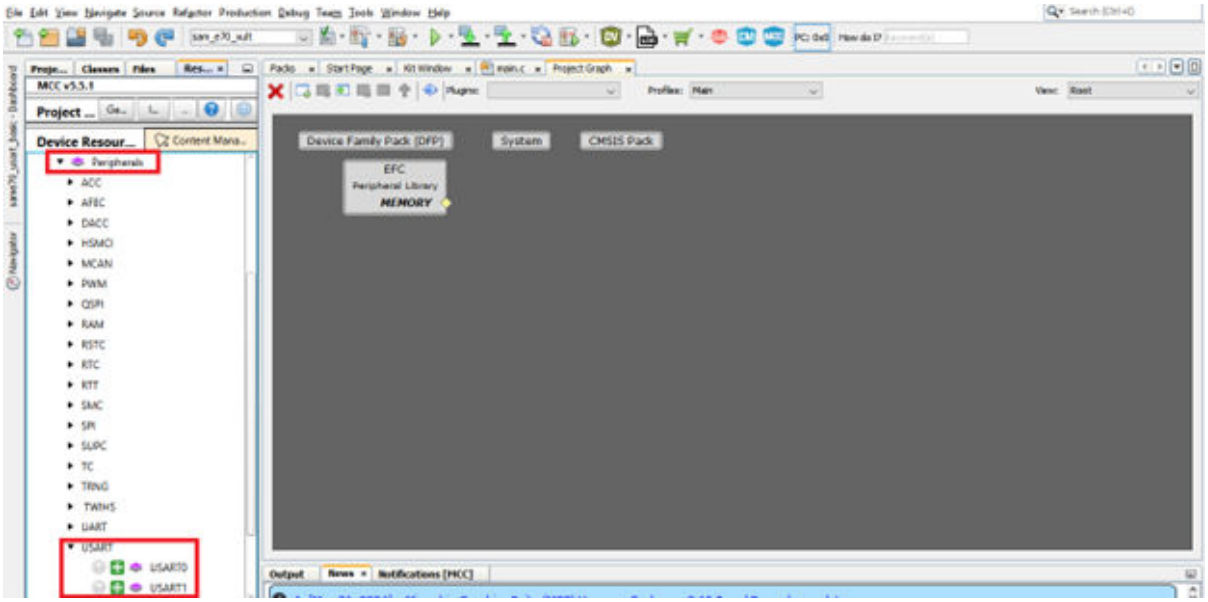
Figure 5-22. Block Diagram - SAM E70 Xplained Ultra Evaluation Kit



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

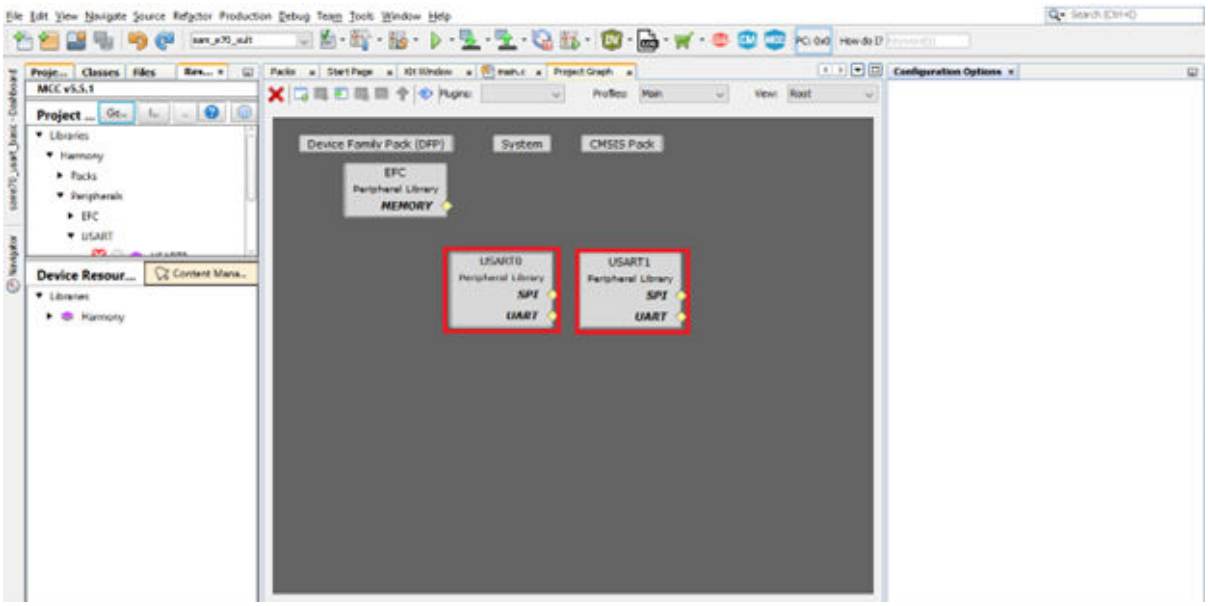
1. To create the project, see Creating the Project.
2. In the MCC window, click **Project Graph**.
3. Under **Device Resources**, click and expand the list of options *Harmony* > *Peripherals* > *USART.*

Figure 5-23. Device Resources



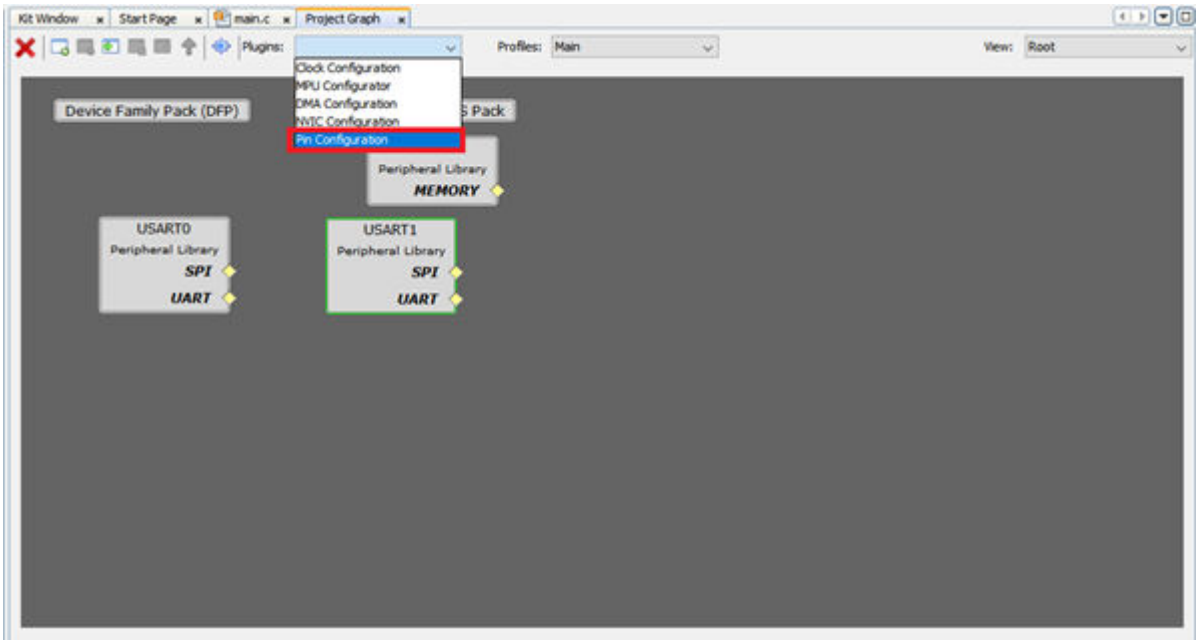
4. Click **USART0** and **USART1**.
5. Observe that the USART0 and USART1 Peripheral Library blocks are added in the Project Graph window.

Figure 5-24. Addition of USART0 and USART1 Modules



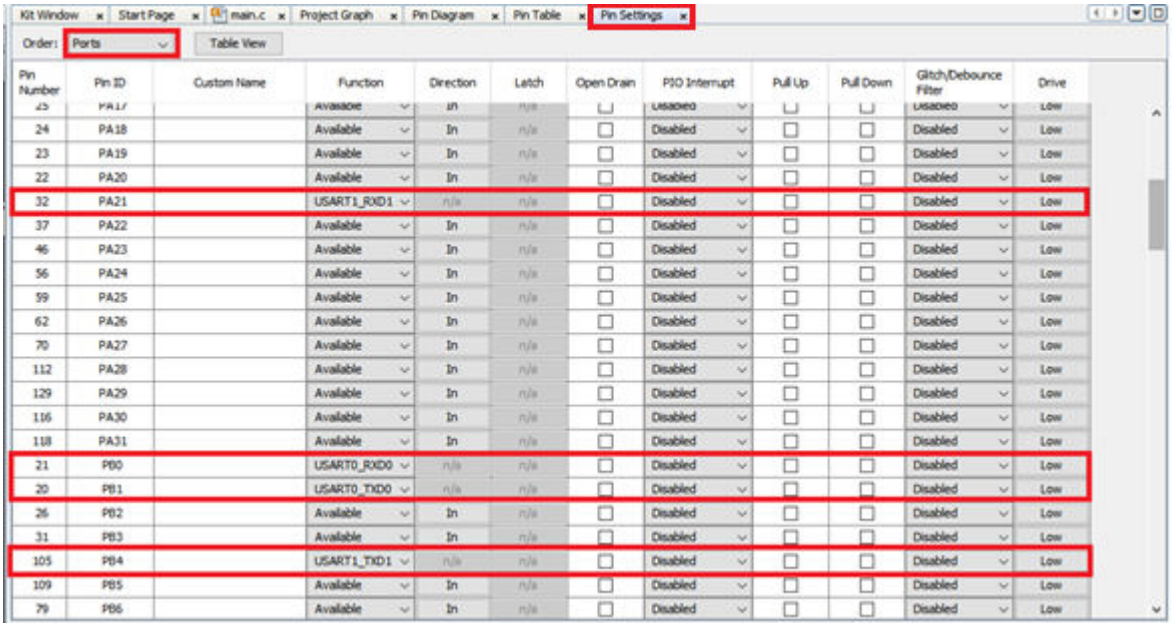
6. From the **Plugins** drop-down list, select **Pin Configuration**, and then click **Pin Settings** (see Figure 5-26).

Figure 5-25. Select the Plugin



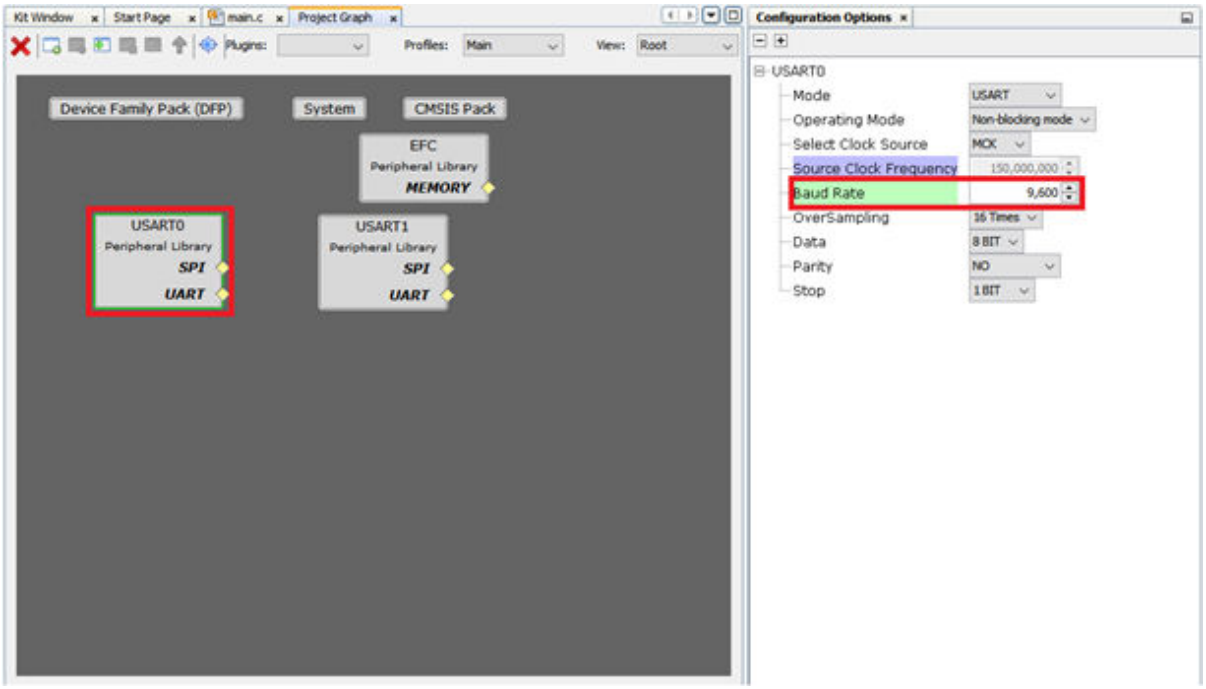
7. From the **Order** drop-down list, select **Ports**. Build configurations according to the application as shown below.
   Configure the following pins PA21, PB0, PB1, and PB4 as USART1_RXD1, USART0_RXD0, USART0_TXD0, and USART1_TXD1, respectively.

Figure 5-26. Pin Configuration



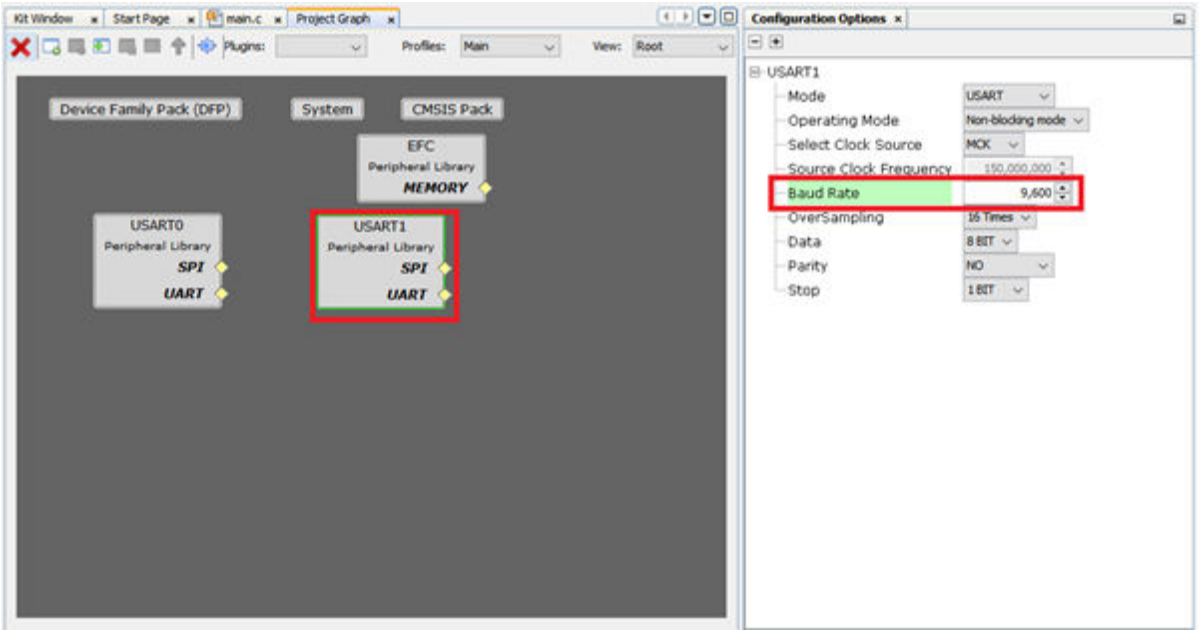
8. In the **Project Graph** window, in the left Navigation bar, select **USART0 Peripheral Library** and in the right **Configuration Options** property page, configure it as shown below to print the data on the Serial Console at 9600 baud rate.

Figure 5-27. Altering the Baud Rate for USART0



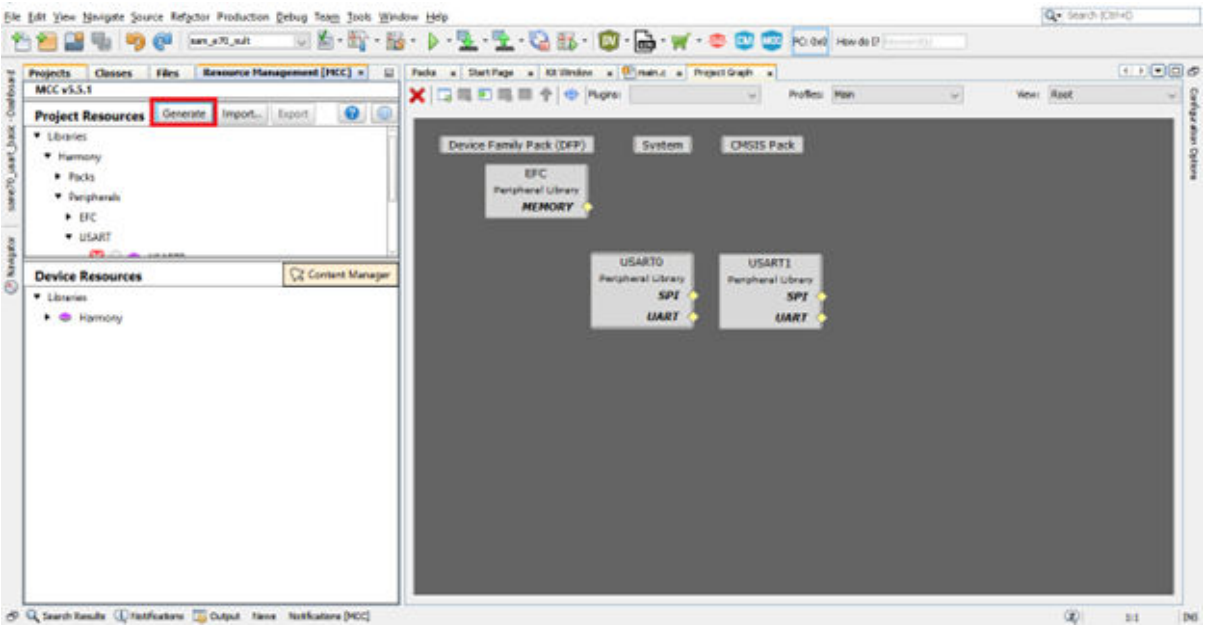
9. Select **USART1 Peripheral Library** and in the right **Configuration Options** property page, configure it as follows to print the data on the Serial Console at 9600 baud rate.

Figure 5-28. Altering the Baud Rate for USART1



10. After configuring the peripherals, as shown in the following figure, click **Generate** under Project Resources.

Figure 5-29. Generation of Code



Note: USART1 is connected to EDBG USART lines through which the SAM E70 Xplained Ultra Evaluation Kit will communicate to the PC terminal application.

11. The `main.c` file will be generated and the logic of the application can be implemented.

5.2.2.1 Application Logic

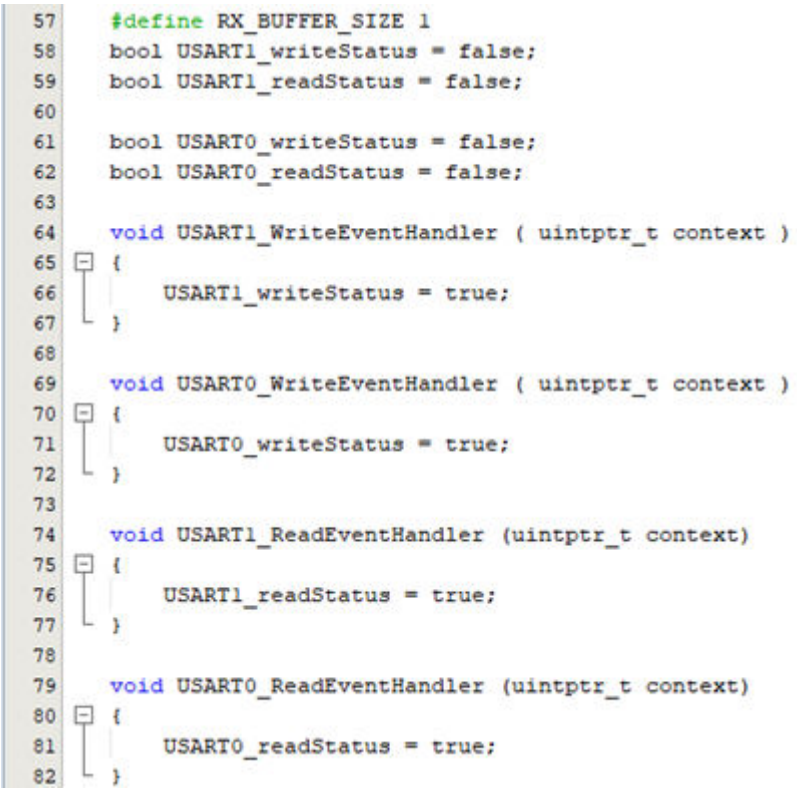
To develop and run the application, follow these steps:

1. Open the `main.c` file of the project located in the source files folder. Add the following code outside the main() function:

```
#define RX_BUFFER_SIZE 1
bool USART1_writeStatus = false;
bool USART1_readStatus = false;

bool USART0_writeStatus = false;
bool USART0_readStatus = false;

void USART1_WriteEventHandler ( uintptr_t context )
{
    USART1_writeStatus = true;
}

void USART0_WriteEventHandler ( uintptr_t context )
{
    USART0_writeStatus = true;
}

void USART1_ReadEventHandler (uintptr_t context)
{
    USART1_readStatus = true;
}

void USART0_ReadEventHandler (uintptr_t context)
{
    USART0_readStatus = true;
}
```

2. In the following figure, the read and write event handlers are declared same as the callback registers seen in the SAM D21 Curiosity Nano Evaluation Kit.

Figure 5-30. Event Handlers



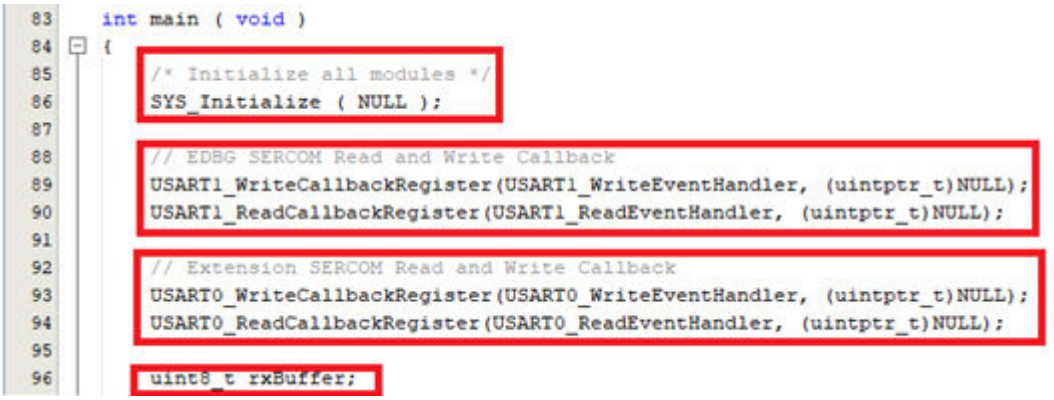
```
57  #define RX_BUFFER_SIZE 1
58  bool USART1_writeStatus = false;
59  bool USART1_readStatus = false;
60
61  bool USART0_writeStatus = false;
62  bool USART0_readStatus = false;
63
64  void USART1_WriteEventHandler ( uintptr_t context )
65  {
66      USART1_writeStatus = true;
67  }
68
69  void USART0_WriteEventHandler ( uintptr_t context )
70  {
71      USART0_writeStatus = true;
72  }
73
74  void USART1_ReadEventHandler (uintptr_t context)
75  {
76      USART1_readStatus = true;
77  }
78
79  void USART0_ReadEventHandler (uintptr_t context)
80  {
81      USART0_readStatus = true;
82  }
```

3. Add the following code inside the main() function:

```
/* Initialize all modules */
    SYS_Initialize ( NULL );

    // EDBG SERCOM Read and Write Callback
    USART1_WriteCallbackRegister(USART1_WriteEventHandler, (uintptr_t)NULL);
    USART1_ReadCallbackRegister(USART1_ReadEventHandler, (uintptr_t)NULL);

    // Extension SERCOM Read and Write Callback
    USART0_WriteCallbackRegister(USART0_WriteEventHandler, (uintptr_t)NULL);
    USART0_ReadCallbackRegister(USART0_ReadEventHandler, (uintptr_t)NULL);

    uint8_t rxBuffer;
```

4. In the following code examples, `SYS_Initialize (NULL)` initializes the modules like clock, Nested Vector Interrupt Controller (NVIC), Embedded Flash Controller (EFC), Parallel In/Out Controller (PIO), and USART. Also, the callback register functions for read and write operations are declared.

Figure 5-31. Initialization of Modules

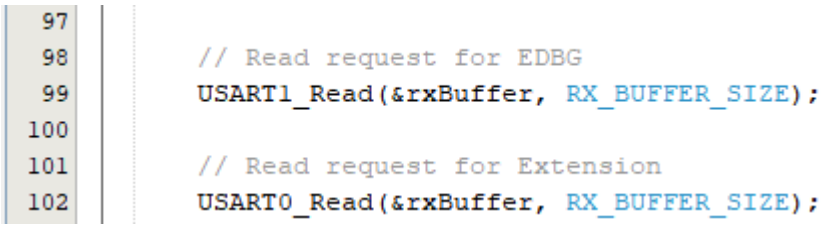


5. Add the following code after the initialization of modules:

```
// Read request for EDBG
    USART1_Read(&rxBuffer, RX_BUFFER_SIZE);

// Read request for Extension
    USART0_Read(&rxBuffer, RX_BUFFER_SIZE);
```

6. Start the implementation with a read request for USART0 (for Extension) and USART1 (for EDBG).

Figure 5-32. Read Requests



7. Add the following code inside the while loop:

```
if(USART0_readStatus == true)
        {
            USART0_readStatus = false;

            //Transmit received bytes from EDBG
            USART1_Write(&rxBuffer, RX_BUFFER_SIZE);
        }

        if(USART1_writeStatus == true)
        {
            USART1_writeStatus = false;
            USART0_Read(&rxBuffer, RX_BUFFER_SIZE);
        }

        if(USART1_readStatus == true)
        {
            USART1_readStatus = false;
            //Transmit received bytes from Extension
            USART0_Write(&rxBuffer, RX_BUFFER_SIZE);
        }
        if(USART0_writeStatus == true)
        {
            USART0_writeStatus = false;
            USART1_Read(&rxBuffer, RX_BUFFER_SIZE);
        }
```

8. In the following figure, if the Extension is ready to read (Rx) the data (i.e., `USART0_readStatus == true`), then it transmits the received data to EDBG, and if the Extension is ready to write (Tx) the data (i.e., `USART1_writeStatus == true`), then EDBG will start a read (Rx) request. Similarly, vice versa if the EDBG is ready to read (Rx) or if the EDBG is ready to write (Tx).

Figure 5-33. Implementation of Basic Configuration

```
103     while ( true )
104     {
105         if(USART0_readStatus == true)
106         {
107             USART0_readStatus = false;
108
109             //Transmit received bytes from EDBG
110             USART1_Write(&rxBuffer, RX_BUFFER_SIZE);
111         }
112
113         if(USART1_writeStatus == true)
114         {
115             USART1_writeStatus = false;
116             USART0_Read(&rxBuffer, RX_BUFFER_SIZE);
117         }
118
119         if(USART1_readStatus == true)
120         {
121             USART1_readStatus = false;
122             //Transmit received bytes from Extension
123             USART0_Write(&rxBuffer, RX_BUFFER_SIZE);
124         }
125         if(USART0_writeStatus == true)
126         {
127             USART0_writeStatus = false;
128             USART1_Read(&rxBuffer, RX_BUFFER_SIZE);
129         }
130         SYS_Tasks ( );
131     }
```

Figure 5-34. Host Side

Figure 5-35. Client Side

5.3 Fractional Baud Configuration

This section is similar to the Basic configuration except for the baud rate configuration between the two SAM D21 Curiosity Nano Evaluation Kits and two SAM E70 Xplained Ultra Evaluation Kits.

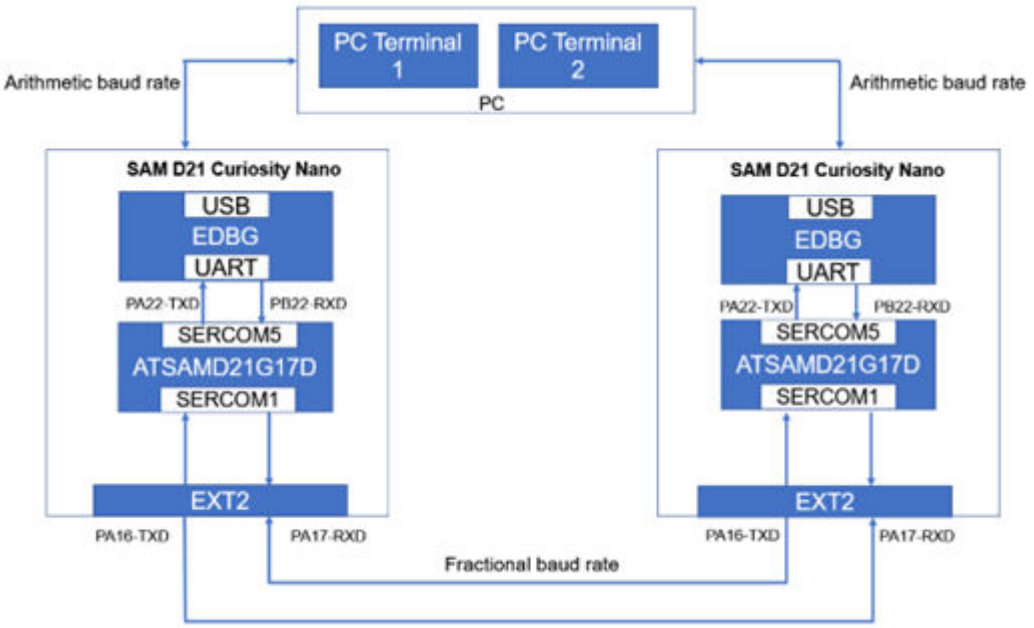
The Communication baud rate between the PC terminal and a board (the SAM D21 Curiosity Nano Evaluation Kit or the SAM E70 Xplained Ultra Evaluation Kit) will be arithmetic baud rate. In contrast, the communication between the two boards (the SAM D21 Curiosity Nano Evaluation Kit or the SAM E70 Xplained Ultra Evaluation Kit) is fractional baud rate.

5.3.1 SAM D21 Curiosity Nano Evaluation Kit

Two SAM D21 Curiosity Nano Evaluation Kit are connected to each other by SERCOM USART lines (TxD and RxD), and connected to the PC terminal through EDBG port.

Note: Click here to access the source code for this application configuration. Alternatively, it is also available in the GitHub reference_apps repository.

Figure 5-36. Block Diagram (SAM D21 Curiosity Nano Evaluation Kit)



The fractional baud equation must be used to calculate the baud value:

fbaud = fref /S(BAUD+(FP/8))

Where,

fbaud – fractional baud frequency
fref – SERCOM generic clock frequency
S – Number of samples per bit
BAUD – BAUD value
FP – fractional part of baud value

From the Fractional baud equation,

BAUD + FP/8 = fref / (fbaud x S)
= 8000000 / (11000 x 16)
= 45.454

Here the integer part corresponds to the BAUD value and the decimal part corresponds to the fractional part.

BAUD = 45,

FP/8 = .454

FP = 3.6, which is 3

5.3.1.1 Application Logic

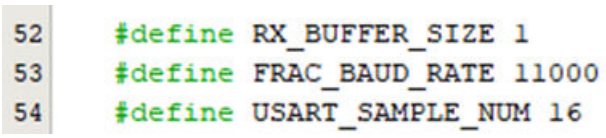
In this section, only fractional baud rate part will be explained, whereas the remaining explanation is covered in the Basic Configuration section. The fractional baud rate of 11000 bps is used in the application. The macro `FRAC_BAUD_RATE` contains the fractional baud value. Adhere to the steps outlined for the basic configuration of the SAM D21 Curiosity Nano Evaluation Kit.

1. To add and configure MPLAB Harmony components using the MCC, see SAM D21 Curiosity Nano Evaluation Kit.

2. Add the following macros:

```
#define RX_BUFFER_SIZE 1
#define FRAC_BAUD_RATE 11000
#define USART_SAMPLE_NUM 16
```

Figure 5-37. Defining the Macro for Fractional Baud Rate



3. Add the following variables:

```
uint16_t baud;
uint8_t fp;
```

4. Add event handlers, see Step 1 in the Application Logic section.
5. Add the following code outside the main() function:

```
// Function to calculate the fractional baud value
void calculate_fractional_baud_value(const uint32_t baudrate, const uint32_t
peripheral_clock, uint8_t sample_num)
{
    uint32_t mul_ratio;
    mul_ratio = (uint64_t)((uint64_t)peripheral_clock * (uint64_t)1000) / (uint64_t)
(baudrate * sample_num);
    baud = mul_ratio / 1000;
    fp = ((mul_ratio - (baud * 1000)) * 8) / 1000;
}

// USART initialization with fractional baud rate settings
void ext_usart_init(void)
{
    calculate_fractional_baud_value(FRAC_BAUD_RATE, SERCOM1_USART_FrequencyGet(),
USART_SAMPLE_NUM);

    SERCOM1_USART_Disable();
    SERCOM1_REGS->USART_INT.SERCOM_CTRLA |= SERCOM_USART_INT_CTRLA_SAMPR(1UL);
    SERCOM1_REGS->USART_INT.SERCOM_BAUD = SERCOM_USART_INT_BAUD_FRAC_BAUD(baud) |
SERCOM_USART_INT_BAUD_FRAC_FP(fp);
    SERCOM1_USART_Enable();
}
```

6. In the following code examples, the value of the fractional baud rate is calculated using the function, `calculate_fractional_baud_value ()` . In the `ext_usart_init ()` function, SERCOM1 is disabled, enabled, and also sets the CTRLA and BAUD registers. The value of the fractional baud rate is calculated using the function, `calculate_fractional_baud_value ()` . In the `ext_usart_init ()` function, SERCOM1 is disabled, enabled, and also sets the CTRLA and BAUD registers.

Figure 5-38. Implementation of Fraction Baud Rate Configuration

```
92  void calculate_fractional_baud_value(const uint32_t baudrate,const uint32_t peripheral_clock,uint8_t sample_num)
93  {
94      uint32_t mul_ratio;
95      mul_ratio = (uint64_t)((uint64_t)peripheral_clock*(uint64_t)1000)/(uint64_t)(baudrate*sample_num);
96      baud = mul_ratio/1000;
97      fp = ((mul_ratio - (baud*1000))*8)/1000;
98  }
99
100 void ext_usart_init(void)
101 {
102     calculate_fractional_baud_value(FRAC_BAUD_RATE,SERCOM1_USART_FrequencyGet(),USART_SAMPLE_NUM);
103
104     SERCOM1_USART_Disable();
105     SERCOM1_REGS->USART_INT.SERCOM_CTRLA |= SERCOM_USART_INT_CTRLA_SAMPR(1UL) ;
106     SERCOM1_REGS->USART_INT.SERCOM_BAUD = SERCOM_USART_INT_BAUD_FRAC_BAUD(baud) | SERCOM_USART_INT_BAUD_FRAC_FP(fp);
107
108     SERCOM1_USART_Enable();
109 }
110
111 int main ( void )
112 {
113     /* Initialize all modules */
114     SYS_Initialize ( NULL );
115
116     ext_usart_init();
```

7. Add the remaining code as shown in the Step 4 of the Application Logic section.

Figure 5-39. Host Side

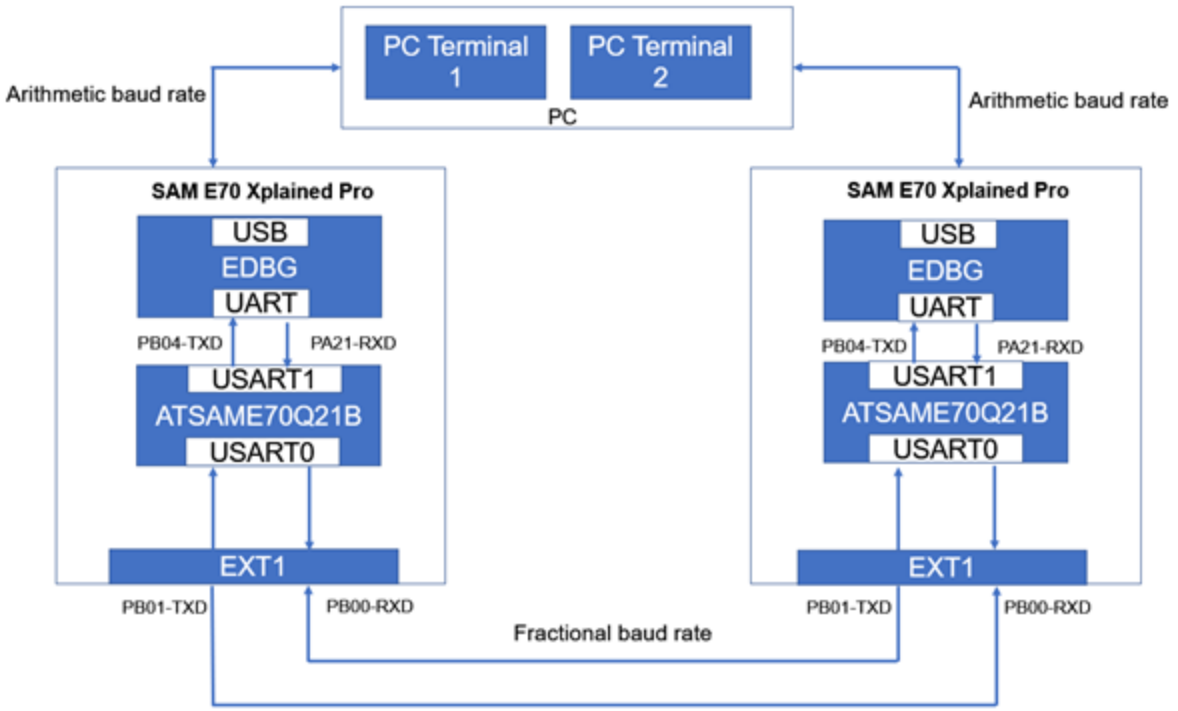
Figure 5-40. Client Side

5.3.2 SAM E70 Xplained Ultra Evaluation Kit

In this section, only fractional baud rate part will be explained whereas the remaining are the same as the Basic configuration section. Fractional baud rate of 11000 bps is used in the application. The macro `FRAC_BAUD_RATE` contains the fractional baud value. Adhere to the steps outlined for the basic configuration of the SAM E70 Xplained Ultra Evaluation Kit. The following figure illustrates the block diagram.

Note: Click here to access the source code for this application configuration. Alternatively, it is also available in the GitHub reference_apps repository.

Figure 5-41. Block Diagram (SAM E70 Xplained Ultra Evaluation Kit)



5.3.2.1 Application Logic

1. To add and configure MPLAB Harmony components using the MCC, see SAM E70 Xplained Ultra Evaluation Kit.
2. Add the following macros:

```
#define RX_BUFFER_SIZE 1
#define FRAC_BAUD_RATE 11000
#define USART_SAMPLE_NUM 16
```

Figure 5-42. Defining the Macro for Fractional Baud Rate

```
51
52  #define RX_BUFFER_SIZE 1
53  #define FRAC_BAUD_RATE 11000
54  #define USART_SAMPLE_NUM 16
```

3. Add the following variables:

```
uint16_t cd;
uint8_t fp;
```

4. Add event handlers, see Step 1 in the Application Logic.
5. Add the following code outside the main() function:

```
void calculate_fractional_baud_value(const uint32_t baudrate, const uint32_t
peripheral_clock,uint8_t sample_num)
{
    uint32_t mul_ratio;
    mul_ratio = (uint64_t)((uint64_t)peripheral_clock*(uint64_t)1000)/(uint64_t)
(baudrate*sample_num);
    cd = mul_ratio/1000;
    fp = ((mul_ratio - (cd*1000))*8)/1000;
}

void ext_usart_init(void)
{
    calculate_fractional_baud_value(FRAC_BAUD_RATE,USART0_FrequencyGet(),USART_SAMPLE_NUM);

    USART0_REGS->US_CR = (US_CR_USART_RXDIS_Msk & US_CR_USART_TXDIS_Msk);

    USART0_REGS->US_BRGR = US_BRGR_CD(cd) | US_BRGR_CD(fp);

    USART0_REGS->US_CR = (US_CR_USART_TXEN_Msk | US_CR_USART_RXEN_Msk);
}
```

6. In the following code examples, value of the fractional baud rate is calculated in the `calculate_fractional_baud_value ()` function. In order to implement the fractional baud rate configuration, the register (US_CR, US_BRGR) operations are done in the `ext_usart_init ()` function.

Figure 5-43. Implementation of Fraction Baud Rate Configuration

```
89  void calculate_fractional_baud_value(const uint32_t baudrate,const uint32_t peripheral_clock,uint8_t sample_num)
90  {
91      uint32_t mul_ratio;
92      mul_ratio = (uint64_t)((uint64_t)peripheral_clock*(uint64_t)1000)/(uint64_t)(baudrate*sample_num);
93      cd = mul_ratio/1000;
94      fp = ((mul_ratio - (cd*1000))*8)/1000;
95  }
96
97  void ext_usart_init(void)
98  {
99      calculate_fractional_baud_value(FRAC_BAUD_RATE,USART0_FrequencyGet(),USART_SAMPLE_NUM);
100
101     USART0_REGS->US_CR = (US_CR_USART_RXDIS_Msk & US_CR_USART_TXDIS_Msk);
102
103     USART0_REGS->US_BRGR = US_BRGR_CD(cd) | US_BRGR_CD(fp);
104
105     USART0_REGS->US_CR = (US_CR_USART_TXEN_Msk | US_CR_USART_RXEN_Msk);
106 }
107 int main ( void )
108 {
109     /* Initialize all modules */
110     SYS_Initialize ( NULL );
111     ext_usart_init();
112
```

7. Add the remaining code, see Step 4 in the Application Logic.

Figure 5-44. Host Side

Figure 5-45. Client Side

5.4 Hardware Handshaking Configuration

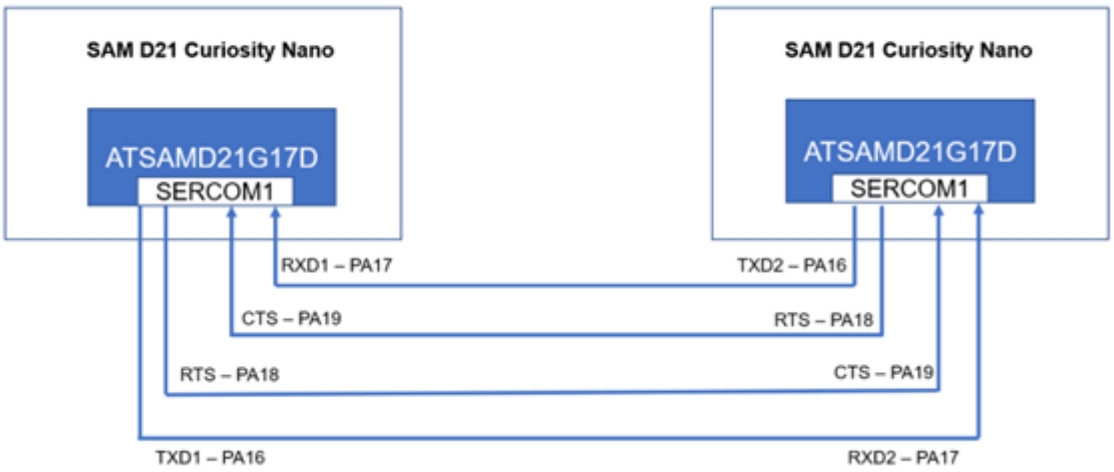
The USART features an out-of-band hardware handshaking flow control mechanism, implemented by connecting the RTS and CTS lines with the remote device. This method ensures reliable data transmission and reception between devices. It typically involves the use of additional control lines, such as RTS (Request to Send) and CTS (Clear to Send), to manage the data flow. When one device is ready to send data, it asserts the RTS line, and the receiving device responds by asserting the CTS line if it is ready to receive data.

5.4.1 SAM D21 Curiosity Nano Evaluation Kit

Two SAM D21 Curiosity Nano Evaluation Kit are connected to each other by SERCOM USART lines (TxD, RxD, RTS, and CTS), and connected to the PC terminal through EDBG port.

Note: Click here to access the source code for this application configuration. Alternatively, it is also available in the GitHub reference_apps repository.

Figure 5-46. Block Diagram (SAM D21 Curiosity Nano Evaluation Kit)



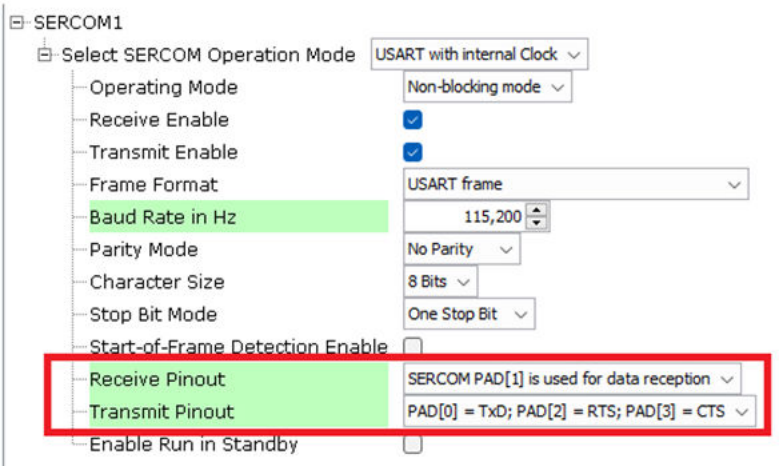
In this application, only extension SERCOM1 will be used. In this demonstration the defined data 0xAA will be sent by both the SAM D21 Curiosity Nano boards with handshaking protocol.

This section is similar to the Basic Configuration, but with additional configuration of the hardware flow control signal lines and some modifications in the code execution. This section will only address the changes.

Users need to follow these steps outlined for the basic configuration of the SAM D21 Curiosity Nano Evaluation Kit.

1. To add and configure MPLAB Harmony components using the MCC, see SAM D21 Curiosity Nano Evaluation Kit.
2. Click and expand **SERCOM1** and then configure the SERCOM1 Peripheral Library block as shown below:

   Figure 5-47. SERCOM1 Configuration



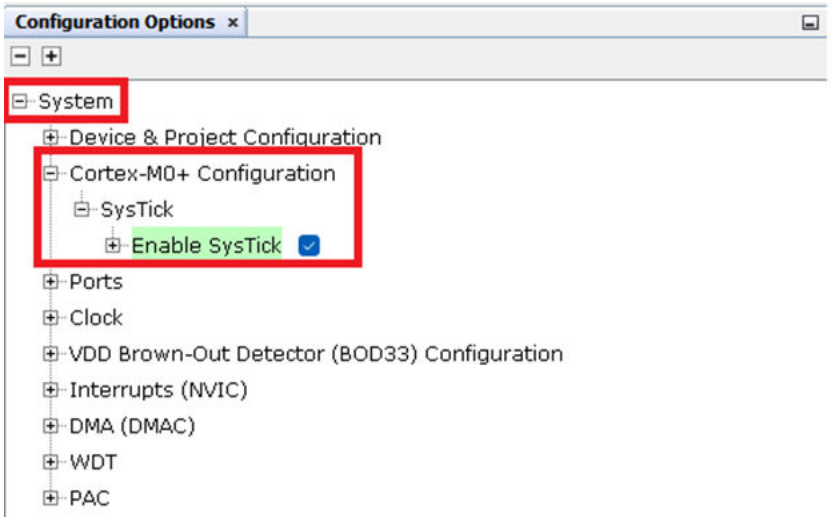
3. Configure the pin configuration as shown below:

Figure 5-48. Pin Configuration

Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch	Pull Up	Pull Down	Drive Strength
27	PA18		SERCOM1_PAD2	Digital	High Impedance	n/a	☑	☐	NORMAL
28	PA19		SERCOM1_PAD3	Digital	High Impedance	n/a	☑	☐	NORMAL

4. In the MCC **Configuration Options**, click and expand **System**, and then select **Enable SysTick**.

Figure 5-49. Enable the Systick Timer



5. After configuring the peripherals, click **Generate** under Project Resources as shown in Step 9 of the SAM D21 Curiosity Nano Evaluation Kit section.

5.4.1.1 Application Logic

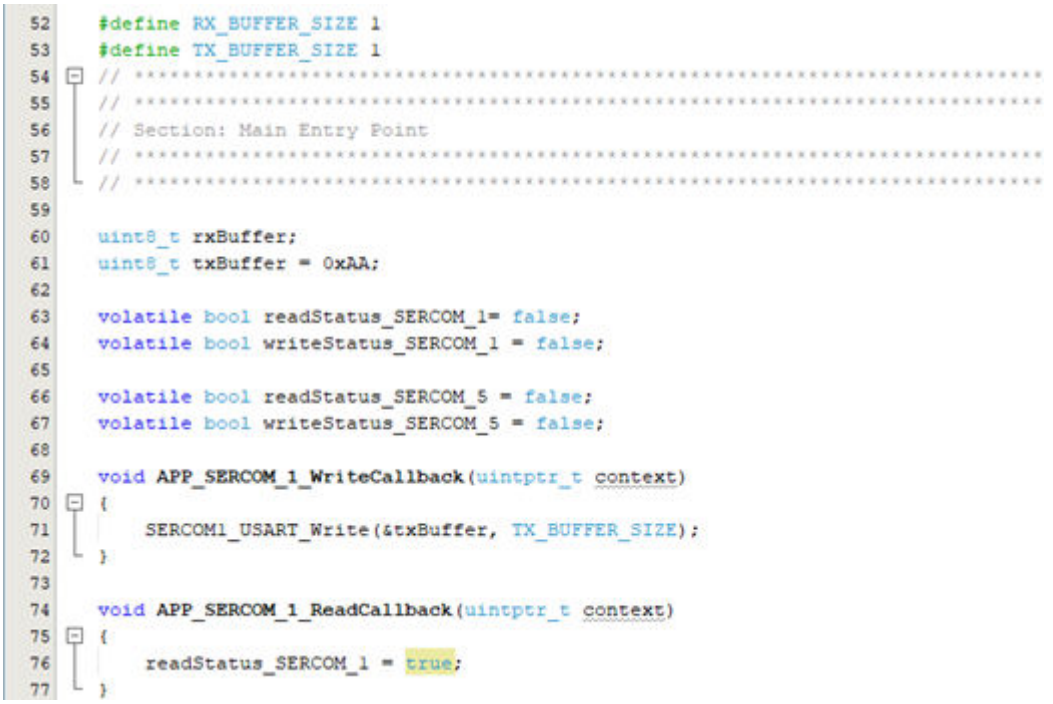
To develop and run the application, follow these steps:

1. Open the `main.c` file of the project located in the source files folder. Add the following code outside the main() function:

```
#define RX_BUFFER_SIZE 1
#define TX_BUFFER_SIZE 1

uint8_t rxBuffer;
uint8_t txBuffer = 0xAA;

volatile bool readStatus_SERCOM_1= false;
volatile bool writeStatus_SERCOM_1 = false;

volatile bool readStatus_SERCOM_5 = false;
volatile bool writeStatus_SERCOM_5 = false;

void APP_SERCOM_1_WriteCallback(uintptr_t context)
{
SERCOM1_USART_Write(&txBuffer, TX_BUFFER_SIZE);
}

void APP_SERCOM_1_ReadCallback(uintptr_t context)
{
readStatus_SERCOM_1 = true;
}
```

2. In the following figure, the read callback and write callback is declared. These functions set the pointer to a client function to be called when the given USART's write or read event occurs. Once the read is completed, the write callback is called and the write function inside this callback writes the data in the console. These callbacks describe the pointer to the function to be called when the write or read event has completed. By setting this to NULL, it will be disabled.

Figure 5-50. Event Handlers



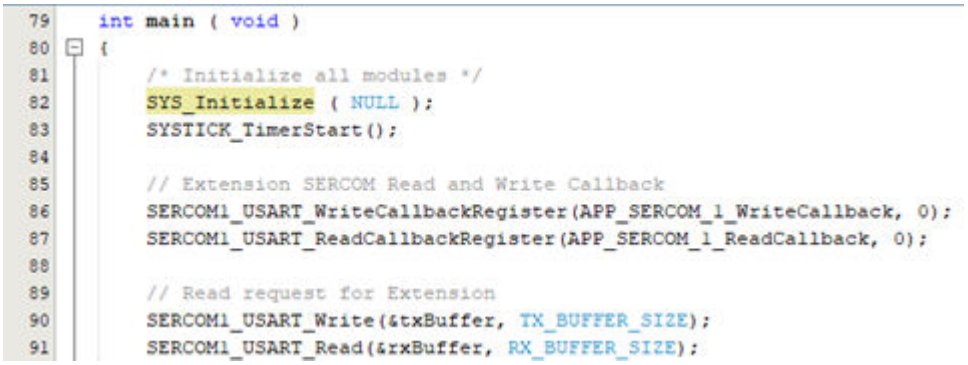
```
52  #define RX_BUFFER_SIZE 1
53  #define TX_BUFFER_SIZE 1
54  // *****************************************************************************
55  // *****************************************************************************
56  // Section: Main Entry Point
57  // *****************************************************************************
58  // *****************************************************************************
59
60  uint8_t rxBuffer;
61  uint8_t txBuffer = 0xAA;
62
63  volatile bool readStatus_SERCOM_1= false;
64  volatile bool writeStatus_SERCOM_1 = false;
65
66  volatile bool readStatus_SERCOM_5 = false;
67  volatile bool writeStatus_SERCOM_5 = false;
68
69  void APP_SERCOM_1_WriteCallback(uintptr_t context)
70  {
71      SERCOM1_USART_Write(&txBuffer, TX_BUFFER_SIZE);
72  }
73
74  void APP_SERCOM_1_ReadCallback(uintptr_t context)
75  {
76      readStatus_SERCOM_1 = true;
77  }
```

3. Inside the main() function, add the following code:

```
SYSTICK_TimerStart();

 // Extension SERCOM Read and Write Callback
SERCOM1_USART_WriteCallbackRegister(APP_SERCOM_1_WriteCallback, 0);
SERCOM1_USART_ReadCallbackRegister(APP_SERCOM_1_ReadCallback, 0);

 // Read request for Extension
SERCOM1_USART_Write(&txBuffer, TX_BUFFER_SIZE);
SERCOM1_USART_Read(&rxBuffer, RX_BUFFER_SIZE);
```

4. In the following figure, `SYS_Initialize (NULL)` initializes the modules, such as ports, clock, SERCOM USART, Nested Vector Interrupt Controller (NVIC), and Non-Volatile Memory Controller (NVMCTRL). The callback register functions for read and write operations are declared. Also, the SYSTICK times and the read and write request for SERCOM1 (for extension) is implemented.

Figure 5-51. Initialization of Modules



```
79  int main ( void )
80  {
81      /* Initialize all modules */
82      SYS_Initialize ( NULL );
83      SYSTICK_TimerStart();
84
85      // Extension SERCOM Read and Write Callback
86      SERCOM1_USART_WriteCallbackRegister(APP_SERCOM_1_WriteCallback, 0);
87      SERCOM1_USART_ReadCallbackRegister(APP_SERCOM_1_ReadCallback, 0);
88
89      // Read request for Extension
90      SERCOM1_USART_Write(&txBuffer, TX_BUFFER_SIZE);
91      SERCOM1_USART_Read(&rxBuffer, RX_BUFFER_SIZE);
```

5. Add the following code inside the while loop:

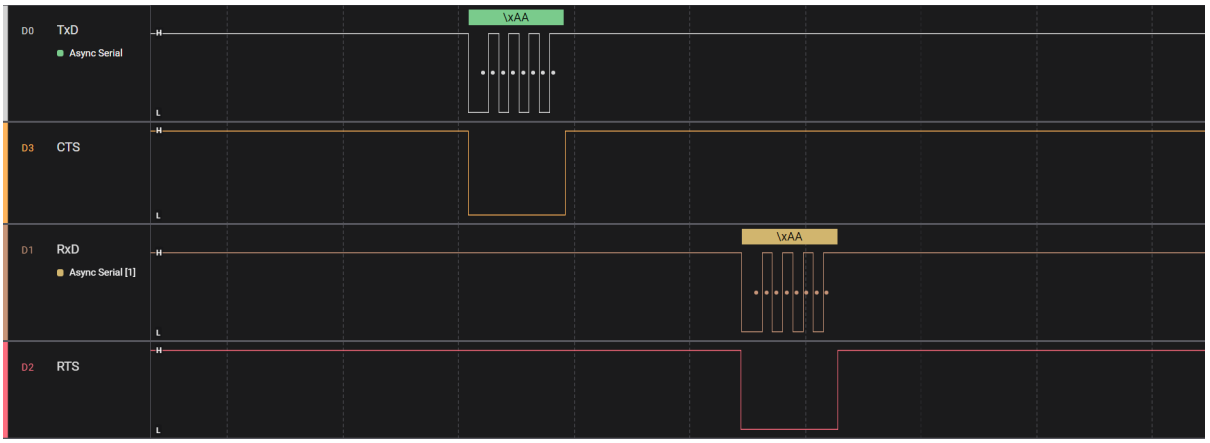
```
SYSTICK_DelayMs(1U);

if(readStatus_SERCOM_1 == true)
{
readStatus_SERCOM_1 = false;
SERCOM1_USART_Read(&rxBuffer, RX_BUFFER_SIZE);
}
```

6. In the following code example, to visualize the hardware flow control using the RTS and CTS lines, a delay of one millisecond is introduced in the code.

Figure 5-52. Implementation of Hardware Handshaking Configuration

```
93      while ( true )
94      {
95          SYSTICK_DelayMs(1U);
96
97          if(readStatus_SERCOM_1 == true)
98          {
99              readStatus_SERCOM_1 = false;
100             SERCOM1_USART_Read(&rxBuffer, RX_BUFFER_SIZE);
101         }
102
103         SYS_Tasks ( );
104     }
```

Figure 5-53. SAM D21 Curiosity Nano – Hardware Handshaking Output



5.4.2 SAM E70 Xplained Ultra Evaluation Kit

Two SAM E70 Xplained Ultra Evaluation Kits are connected to each other by SERCOM USART lines (TxD, RxD, RTS, and CTS) through EXT1 connector and connected to the PC terminal through EDBG port.

Notes: Click here to access the source code for this application configuration. Alternatively, it is also available in the GitHub reference_apps repository.

Figure 5-54. Block Diagram (SAM E70 Xplained Ultra Evaluation Kit)

In this section, only hardware handshaking part will be explained whereas the remaining are covered in the Basic configuration section. Here, RTS and CTS lines are controlled manually, whereas in SAM D21 it is not. Adhere to the steps outlined for the basic configuration of the SAM E70 Xplained Ultra Evaluation Kit.

1. To add and configure MPLAB Harmony components using the MCC, see SAM E70 Xplained Ultra Evaluation Kit.
2. In **Pin Configuration**, configure the PB2 and PB3 pins for RTS and CTS.

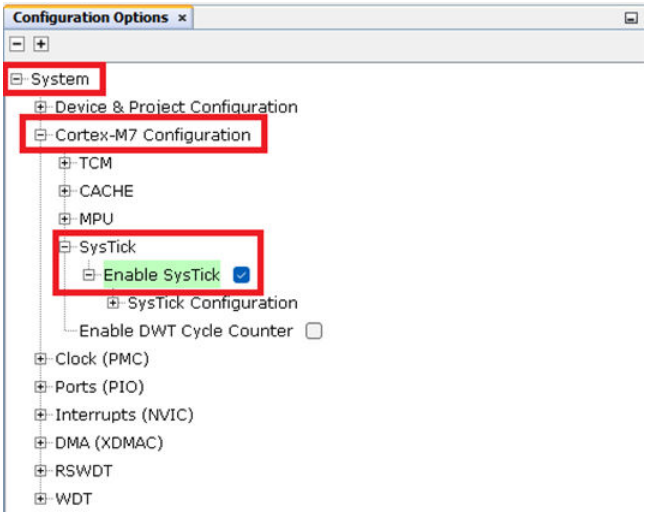
Figure 5-55. Pin Configuration

Order: Ports | Table View

Pin Number	Pin ID	Custom Name	Function	Direction	Latch	Open Drain	PIO Interrupt	Pull Up	Pull Down	Glitch/Debounce Filter	Drive
49	PA15		Available	In	n/a		Disabled			Disabled	Low
45	PA16		Available	In	n/a		Disabled			Disabled	Low
25	PA17		Available	In	n/a		Disabled			Disabled	Low
24	PA18		Available	In	n/a		Disabled			Disabled	Low
23	PA19		Available	In	n/a		Disabled			Disabled	Low
22	PA20		Available	In	n/a		Disabled			Disabled	Low
32	PA21		USART1_RXD1	n/a	n/a		Disabled			Disabled	Low
37	PA22		Available	In	n/a		Disabled			Disabled	Low
46	PA23		Available	In	n/a		Disabled			Disabled	Low
56	PA24		Available	In	n/a		Disabled			Disabled	Low
59	PA25		Available	In	n/a		Disabled			Disabled	Low
62	PA26		Available	In	n/a		Disabled			Disabled	Low
70	PA27		Available	In	n/a		Disabled			Disabled	Low
112	PA28		Available	In	n/a		Disabled			Disabled	Low
129	PA29		Available	In	n/a		Disabled			Disabled	Low
116	PA30		Available	In	n/a		Disabled			Disabled	Low
118	PA31		Available	In	n/a		Disabled			Disabled	Low
21	PB0		USART0_RXD0	n/a	n/a		Disabled			Disabled	Low
20	PB1		USART0_TXD0	n/a	n/a		Disabled			Disabled	Low
26	PB2		USART0_CTS0	n/a	n/a		Disabled			Disabled	Low
31	PB3		USART0_RTS0	n/a	n/a		Disabled			Disabled	Low
105	PB4		USART1_TXD1	n/a	n/a		Disabled			Disabled	Low
109	PB5		Available	In	n/a		Disabled			Disabled	Low
79	PB6		Available	In	n/a		Disabled			Disabled	Low
89	PB7		Available	In	n/a		Disabled			Disabled	Low
141	PB8		Available	In	n/a		Disabled			Disabled	Low
142	PB9		Available	In	n/a		Disabled			Disabled	Low

3. In MCC Configuration Options, click and expand *System* > *Contex-M7 Configuration* > *SysTick.*
4. Select **Enable SysTick.**

Figure 5-56. Enable the Systick Timer



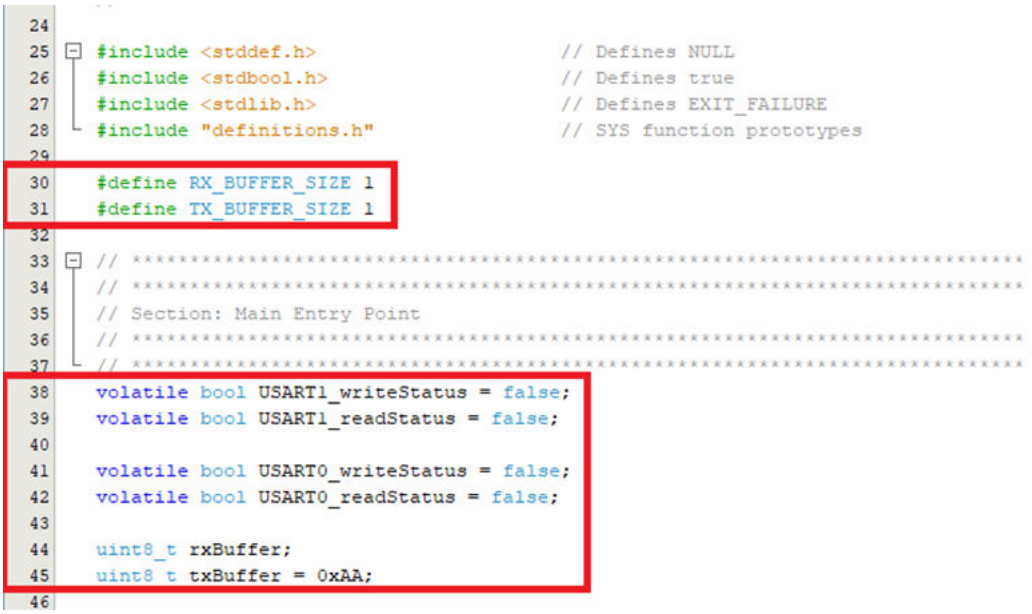
5. Click **Generate** in Project Resources as shown in Step 9 of the SAM E70 Xplained Ultra Evaluation Kit section.

5.4.2.1 Application Logic

1. Add the following macros and variables outside the main() function:

```
#define RX_BUFFER_SIZE 1
#define TX_BUFFER_SIZE 1

volatile bool USART1_writeStatus = false;
volatile bool USART1_readStatus = false;

volatile bool USART0_writeStatus = false;
volatile bool USART0_readStatus = false;

uint8_t rxBuffer;
uint8_t txBuffer = 0xAA;
```

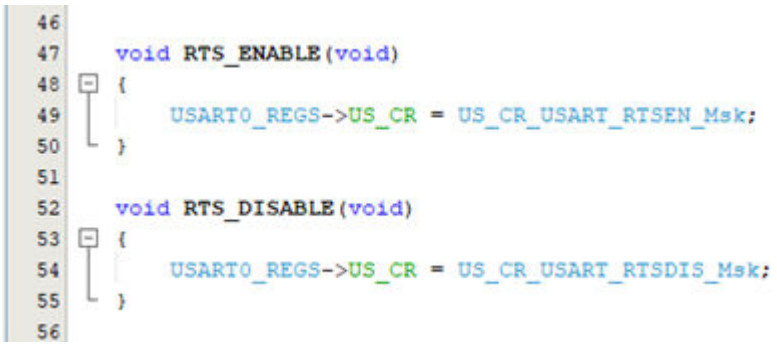
Figure 5-57. Adding the Macros and Variables



2. Add the `RTS_ENABLE` and `RTS_DISABLE` functions outside the main() function to manually control the RTS lines:

```
void RTS_ENABLE(void)
{
    USART0_REGS->US_CR = US_CR_USART_RTSEN_Msk;
}
void RTS_DISABLE(void)
{
    USART0_REGS->US_CR = US_CR_USART_RTSDIS_Msk;
}
```

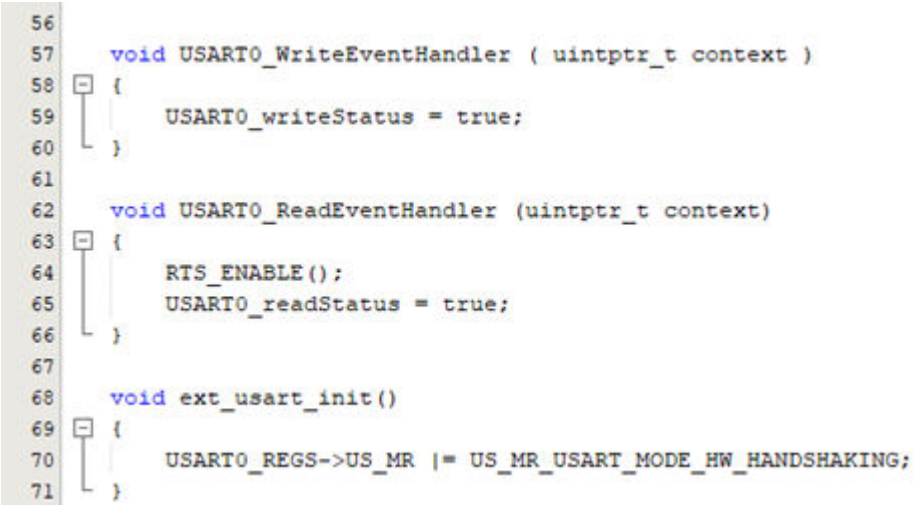
Figure 5-58. Adding the RTS Functions



3. Add event handlers and enable the hardware handshaking mode outside the main() function:

```
void USART0_WriteEventHandler ( uintptr_t context )
{
    USART0_writeStatus = true;
}

void USART0_ReadEventHandler (uintptr_t context)
{
    RTS_ENABLE();
    USART0_readStatus = true;
}

void ext_usart_init()
{
    USART0_REGS->US_MR |= US_MR_USART_MODE_HW_HANDSHAKING;
}
```

Figure 5-59. Adding Event Handlers



4. Call the necessary functions and callback registers inside the main() function:

```
SYSTICK_TimerStart();
ext_usart_init();

USART0_WriteCallbackRegister(USART0_WriteEventHandler, (uintptr_t)NULL);
USART0_ReadCallbackRegister(USART0_ReadEventHandler, (uintptr_t)NULL);

RTS_DISABLE();
USART0_Read(&rxBuffer, RX_BUFFER_SIZE);

USART0_Write(&txBuffer, TX_BUFFER_SIZE);
```

Figure 5-60. Initialization of Modules

```
72
73    int main ( void )
74    {
75        /* Initialize all modules */
76        SYS_Initialize ( NULL );
77
78        SYSTICK_TimerStart();
79        ext_usart_init();
80
81        USART0_WriteCallbackRegister(USART0_WriteEventHandler, (uintptr_t)NULL);
82        USART0_ReadCallbackRegister(USART0_ReadEventHandler, (uintptr_t)NULL);
83
84        RTS_DISABLE();
85        USART0_Read(&rxBuffer, RX_BUFFER_SIZE);
86
87        USART0_Write(&txBuffer, TX_BUFFER_SIZE);
```

5. Add the hardware handshaking configuration logic inside the `while` loop in the main() function:

```
SYSTICK_DelayUs(500U);
if(USART0_writeStatus == true)
{
USART0_writeStatus = false;

//Transmit received bytes from EDBG
USART0_Write(&txBuffer, TX_BUFFER_SIZE);
}

if(USART0_readStatus == true)
{
USART0_readStatus = false;

//Receive transmitted bytes from EDBG
RTS_DISABLE();
USART0_Read(&rxBuffer, RX_BUFFER_SIZE);
}
```

Figure 5-61. Adding the Application Logic

```
87      USART0_Write(&txBuffer, TX_BUFFER_SIZE);
88
89      while ( true )
90      {
91          SYSTICK_DelayUs(500U);
92          if(USART0_writeStatus == true)
93          {
94              USART0_writeStatus = false;
95
96              //Transmit received bytes from EDBG
97              USART0_Write(&txBuffer, TX_BUFFER_SIZE);
98          }
99
100         if(USART0_readStatus == true)
101         {
102             USART0_readStatus = false;
103
104             //Receive transmitted bytes from EDBG
105             RTS_DISABLE();
106             USART0_Read(&rxBuffer, RX_BUFFER_SIZE);
107         }
108
109         /* Maintain state machines of all polled MPLAB Harmony modules. */
110         SYS_Tasks ( );
111     }
112
113     /* Execution should not come here during normal operation */
114
115     return ( EXIT_FAILURE );
116 }
```

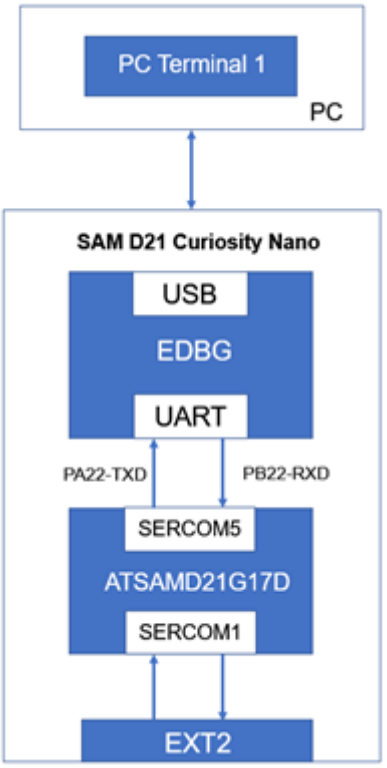
Figure 5-62. SAM E70 Xplained Ultra – Hardware Handshaking Output

5.5 SOF Detection and Wakeup Configuration

The USART start-of-frame (SOF) detector can wake up the CPU from Standby Sleep mode when it detects a Start bit. In Standby Sleep mode, the internal fast start-up oscillator must be selected as the `GCLK_SERCOMx_CORE` source. The application enters the Standby Sleep mode, and the PC key press character will wake up the device from the Sleep mode and displays the character on the terminal.

Note: Click here to access the source code for this application configuration. Alternatively, it is also available in the GitHub reference_apps repository.

Figure 5-63. Block diagram (SAM D21 Curiosity Nano Evaluation Kit)

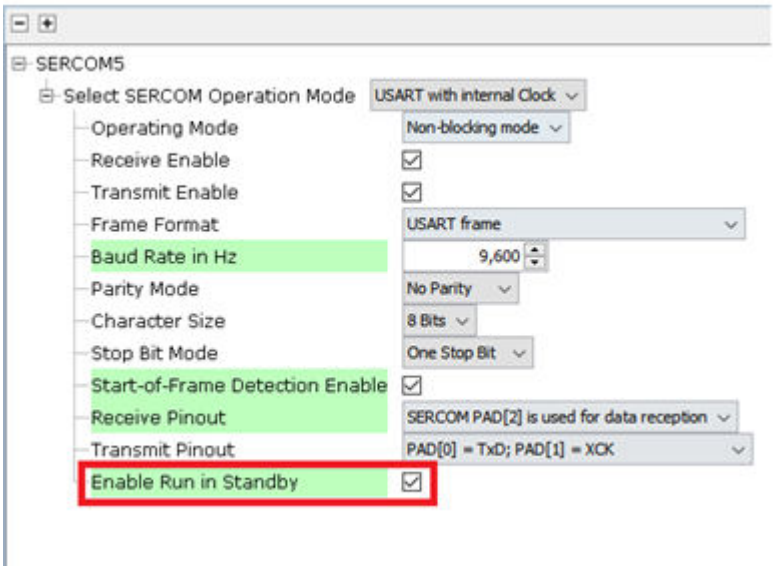


This application uses only one SAM D21 Curiosity Nano Evaluation Kit connected to the PC terminal through EDBG. In this section, only SOF detection and wakeup configuration part will be explained whereas the remaining availabe in the Basic Configuration section.

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

1. To create the project, see SAM D21 Curiosity Nano Evaluation Kit.
2. Click and expand **SERCOM5** and then select **Enable Run in Standby**, and then click **Generate**.

Figure 5-64. Configuration Options



3. Add the following code outside the main() function:

```
#define RX_BUFFER_SIZE 1
volatile bool rx_done = false;
uint8_t edbg_rx_data;
volatile bool tx_done = false;

void APP_SERCOM_5_WriteCallback(uintptr_t context)
{
   tx_done = true;
}

void APP_SERCOM_5_ReadCallback(uintptr_t context)
{
    if((SERCOM5_REGS->USART_INT.SERCOM_INTFLAG & SERCOM_USART_INT_INTFLAG_RXS_Msk) ==
SERCOM_USART_INT_INTFLAG_RXS_Msk)
    {
        SERCOM5_REGS->USART_INT.SERCOM_INTFLAG |=
(uint8_t)SERCOM_USART_INT_INTFLAG_RXS_Msk;

    }

    rx_done = true;
}
void usart_send_string(const char *str)
{
    SERCOM5_USART_Write((void *)&str[0], strlen(str));
}
```

4. In the following code example, the `tx_done` becomes true once the write function (which is happening in `APP_SERCOM_5_Writecallback ()` is completed and `rx_done` becomes true once read function (`APP_SERCOM_5_Raedcallback ()` is completed. Also, set the RXS flag if the value of `RXS_Msk` and `SERCOM_INTFLAG` equals `RXS_Msk`. The function `usart_send_string()` is used to send strings to the console.

Figure 5-65. Implementation of SOF Detection and Wakeup Configuration - Example 1

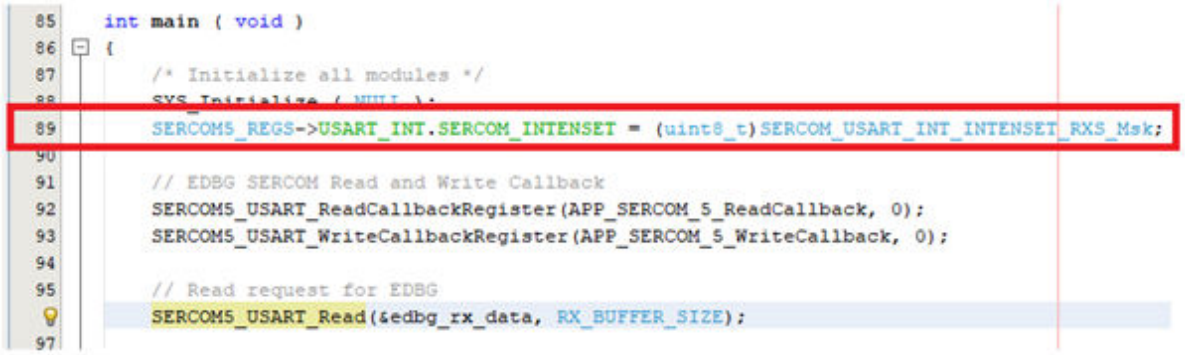
```
59  #define RX_BUFFER_SIZE 1
60  volatile bool rx_done = false;
61  uint8_t edbg_rx_data;
62  volatile bool tx_done = false;
63
64  void APP_SERCOM_5_WriteCallback(uintptr_t context)
65  {
66     tx_done = true;
67  }
68
69  void APP_SERCOM_5_ReadCallback(uintptr_t context)
70  {
71      if((SERCOM5_REGS->USART_INT.SERCOM_INTFLAG & SERCOM_USART_INT_INTFLAG_RXS_Msk) == SERCOM_USART_INT_INTFLAG_RXS_Msk)
72      {
73          SERCOM5_REGS->USART_INT.SERCOM_INTFLAG |= (uint8_t)SERCOM_USART_INT_INTFLAG_RXS_Msk;
74
75      }
76
77      rx_done = true;
78  }
79
80  void usart_send_string(const char *str)
81  {
82      SERCOM5_USART_Write((void *)&str[0], strlen(str));
83  }
```

5. Add the following code inside the main() function:

```
/* Initialize all modules */
    SYS_Initialize ( NULL );
    SERCOM5_REGS->USART_INT.SERCOM_INTENSET =  (uint8_t)SERCOM_USART_INT_INTENSET_RXS_Msk;

    // EDBG SERCOM Read and Write Callback
    SERCOM5_USART_ReadCallbackRegister(APP_SERCOM_5_ReadCallback, 0);
    SERCOM5_USART_WriteCallbackRegister(APP_SERCOM_5_WriteCallback, 0);

    // Read request for EDBG
    SERCOM5_USART_Read(&edbg_rx_data, RX_BUFFER_SIZE);
```

6. In the following figure, set the particular RXS interrupt, and the callback registers are called as seen in the basic configuration. Also, a read request is also given.

Figure 5-66. Implementation of SOF Detection and Wakeup Configuration - Example 2



Note: In SOF detection and wakeup configuration only SERCOM5 (EDBG) is used.

7. Add the following code inside the `while` loop:

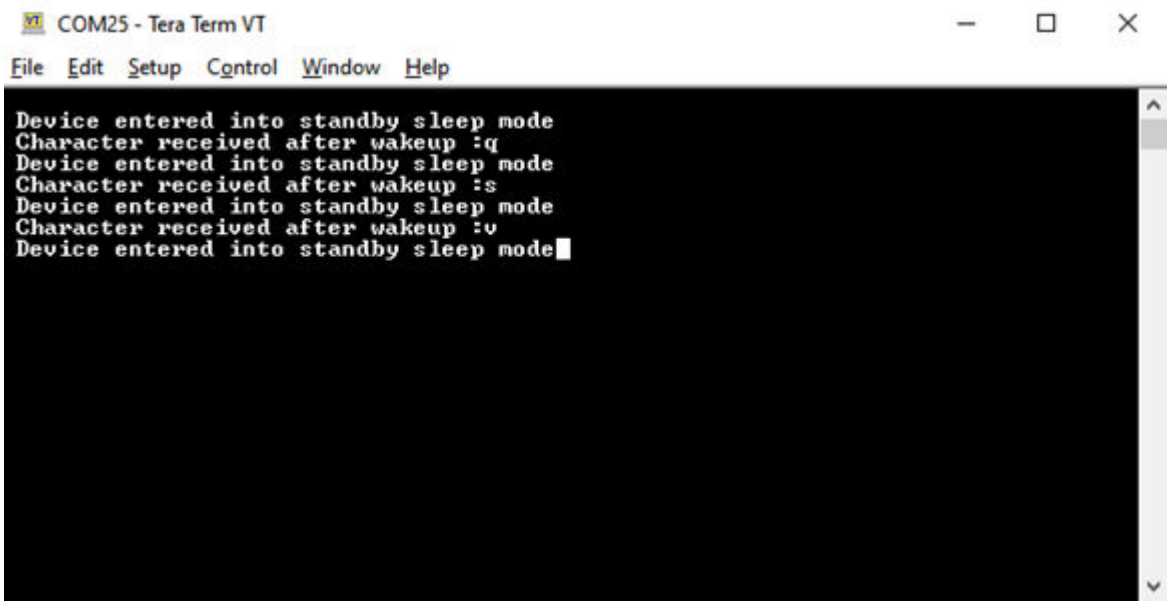
```
if (!rx_done)
        {
            tx_done = false;
            usart_send_string("\r\n Device entered into standby sleep mode");
            while(!(SERCOM5_REGS->USART_INT.SERCOM_INTFLAG &
SERCOM_USART_INT_INTFLAG_TXC_Msk));

            // Enters standby sleep mode.
            PM_StandbyModeEnter();
        }
        while(!rx_done);

        tx_done = false;
        usart_send_string("\r\n Character received after wakeup :");
        while(!tx_done);

        tx_done = false;
        SERCOM5_USART_Write(&edbg_rx_data, RX_BUFFER_SIZE);
        while(!tx_done);

        rx_done = false;
        SERCOM5_USART_Read(&edbg_rx_data, RX_BUFFER_SIZE);
        SYS_Tasks ( );
    }
```

8. In the following figure, when a character is entered, it wakes up from Sleep mode and print the character that is entered. Until the TXC is false (i.e., until the transmission is complete) it loops in the while loop itself. When it becomes true (i.e., once the transmission is complete) it enters Standby Sleep mode.

Figure 5-67. Implementation of SOF Detection and Wakeup Configuration - Example 3

```
    while ( true )
    {
        if (!rx_done)
        {
            tx_done = false;
            usart_send_string("\r\n Device entered into standby sleep mode");
            while(!(SERCOM5_REGS->USART_INT.SERCOM_INTFLAG & SERCOM_USART_INT_INTFLAG_TXC_Msk));

            // Enters standby sleep mode.
            PM_StandbyModeEnter();
        }
        while(!rx_done);

        tx_done = false;
        usart_send_string("\r\n Character received after wakeup :");
        while(!tx_done);

        tx_done = false;
        SERCOM5_USART_Write(&edbg_rx_data, RX_BUFFER_SIZE);
        while(!tx_done);

        rx_done = false;
        SERCOM5_USART_Read(&edbg_rx_data, RX_BUFFER_SIZE);
        SYS_Tasks ( );
    }


    return ( EXIT_FAILURE );
}
```

Figure 5-68. SERCOM USART – SOF Detection and Wake-up Configuration Output



Note: The SOF detection and wakeup configuration is not available in the SAM E70/S70/V7x family of devices.

6. References

The following documents are used as reference:

- Getting started with MPLAB Harmony v3 Peripheral Libraries on SAM D21 MCUs
- Getting started with MPLAB Harmony v3 Drivers on SAM D21 MCUs Using FreeRTOS
- Getting started with MPLAB Harmony v3 Peripheral Libraries on SAM E70/S70/V70/V71 MCUs
- Getting started with MPLAB Harmony v3 Drivers on SAM E70/S70/V70/V71 MCUs Using FreeRTOS
- SAM D21 Curiosity Nano Evaluation Kit
- SAM E70 Xplained Ultra Evaluation Kit
- SAM D21/DA1 Family Data Sheet (*DS40001882*)
- SAM D21 Curiosity Nano User Guide (*DS70005409*)
- SAM E70/S70/V70/V71 Family Data Sheet (*DS60001527*)
- SAM E70 Xplained Ultra User Guide (*DS70005389*)
- 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: www.microchip.com/en-us/tools-resources/configure/mplab-harmony and https://developerhelp.microchip.com/xwiki/bin/view/software-tools/harmony/
- For more information on various applications, refer to: github.com/Microchip-MPLAB-Harmony/reference_apps
- For additional information, visit the Microchip website or contact a local Microchip Sales Representative

7. Revision History

7.1 Revision A - February 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: 979-8-3371-0711-0

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

ATSAMD21E15, ATSAMD21E15L, ATSAMD21E16, ATSAMD21E16L, ATSAMD21E17, ATSAMD21E17L, ATSAMD21E18, ATSAMD21G15, ATSAMD21G16, ATSAMD21G16L, ATSAMD21G17, ATSAMD21G17L, ATSAMD21G18, ATSAMD21J15, ATSAMD21J16, ATSAMD21J17, ATSAMD21J18, ATSAME70J19, ATSAME70J20, ATSAME70J21, ATSAME70N19, ATSAME70N20, ATSAME70N21, ATSAME70Q19, ATSAME70Q20, ATSAME70Q21, ATSAML10D14A, ATSAML10D15A, ATSAML10D16A, ATSAML10E14A, ATSAML10E15A, ATSAML10E16A, ATSAML11D14A, ATSAML11D15A, ATSAML11D16A, ATSAML11E14A, ATSAML11E15A, ATSAML11E16A, ATSAML21E15B, ATSAML21E16B, ATSAML21E17B, ATSAML21E18B, ATSAML21G16B, ATSAML21G17B, ATSAML21G18B, ATSAML21J16B, ATSAML21J17B, ATSAML21J18B, ATSAML22G16A, ATSAML22G17A, ATSAML22G18A, ATSAML22J16A, ATSAML22J17A, ATSAML22J18A, ATSAML22N16A, ATSAML22N17A, ATSAML22N18A, PIC32CK0512GC00064, PIC32CK0512GC00100, PIC32CK0512GC01064, PIC32CK0512GC01100, PIC32CK0512SG00064, PIC32CK0512SG00100, PIC32CK0512SG01064, PIC32CK0512SG01100, PIC32CK1025GC00064, PIC32CK1025GC00100, PIC32CK1025GC01064, PIC32CK1025GC01100, PIC32CK1025GC01144, PIC32CK1025SG00064, PIC32CK1025SG00100, PIC32CK1025SG01064, PIC32CK1025SG01100, PIC32CK1025SG01144, PIC32CK2051GC00064, PIC32CK2051GC00100, PIC32CK2051GC00144, PIC32CK2051GC01064, PIC32CK2051GC01100, PIC32CK2051GC01144, PIC32CK2051SG00064, PIC32CK2051SG00100, PIC32CK2051SG00144, PIC32CK2051SG01064, PIC32CK2051SG01100, PIC32CK2051SG01144, PIC32CM1216MC00032, PIC32CM1216MC00048, PIC32CM2532JH00032, PIC32CM2532JH00048, PIC32CM2532JH00064, PIC32CM2532JH00100, PIC32CM2532JH01032, PIC32CM2532JH01048, PIC32CM2532JH01064, PIC32CM2532JH01100, PIC32CM2532LE00048, PIC32CM2532LE00064, PIC32CM2532LE00100, PIC32CM2532LS00048, PIC32CM2532LS00064, PIC32CM2532LS00100, PIC32CM2532LS60048, PIC32CM2532LS60064, PIC32CM2532LS60100, PIC32CM5112GC00048, PIC32CM5112GC00064, PIC32CM5112GC00100, PIC32CM5112SG00048, PIC32CM5112SG00064, PIC32CM5112SG00100, PIC32CM5164JH00032, PIC32CM5164JH00048, PIC32CM5164JH00064, PIC32CM5164JH00100, PIC32CM5164JH01032, PIC32CM5164JH01048, PIC32CM5164JH01064, PIC32CM5164JH01100, PIC32CM5164LE00048, PIC32CM5164LE00064, PIC32CM5164LE00100, PIC32CM5164LS00048, PIC32CM5164LS00064, PIC32CM5164LS00100, PIC32CM5164LS60048, PIC32CM5164LS60064, PIC32CM5164LS60100, PIC32CM6408MC00032, PIC32CM6408MC00048, PIC32CX1025MTC, PIC32CX1025MTG, PIC32CX1025MTSH, PIC32CX1025SG41128, PIC32CX1025SG60100, PIC32CX1025SG60128, PIC32CX1025SG61100, PIC32CX1025SG61128, PIC32CX2051MTC, PIC32CX2051MTG, PIC32CX2051MTSH, PIC32CX5109BZ31032, PIC32CX5109BZ31048, PIC32CX5112MTC, PIC32CX5112MTG, PIC32CX5112MTSH, PIC32CX-BZ2, PIC32CX-BZ3, PIC32CX-BZ6, PIC32CZ2051CA70064, PIC32CZ2051CA70100, PIC32CZ2051CA70144, PIC32CZ2051CA80100, PIC32CZ2051CA80144, PIC32CZ2051CA80176, PIC32CZ2051CA80208, PIC32CZ2051CA90100, PIC32CZ2051CA90144, PIC32CZ2051CA90176, PIC32CZ2051CA90208, PIC32CZ2051MC70064, PIC32CZ2051MC70100, PIC32CZ2051MC70144, PIC32CZ4010CA80100, PIC32CZ4010CA80144, PIC32CZ4010CA80176, PIC32CZ4010CA80208, PIC32CZ4010CA90100, PIC32CZ4010CA90144, PIC32CZ4010CA90176, PIC32CZ8110CA80100, PIC32CZ8110CA80144, PIC32CZ8110CA80176, PIC32CZ8110CA80208, PIC32CZ8110CA90100, PIC32CZ8110CA90144, PIC32CZ8110CA90176, PIC32CZ8110CA90208