APPLICATION NOTE

AT11626: SAM D SERCOM USART Configuration

ATSAMD21J18

Introduction

This application note explains the various features of SERCOM USART in the Atmel® SAM D microcontrollers and its configurations with example codes and corresponding scope shots.

For demonstration purpose two SAM D21 Xplained Pro boards will be used.

Table of Contents

1 Introduction to Serial Communication Interfaces (USART, I^{2}C, and SPI)

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 differs from each other. USART, I^{2}C, and SPI are the common serial modules used in embedded systems.

1.1 USART

USART (Universal Synchronous/Asynchronous Receiver/Transmitter) is based on the RS232 protocol where it can operate in both synchronous and asynchronous modes. It is full-duplex in operation. The limitation can be the lower data rates.

1.2 I^{2}C

I^{2}C is a two-wire protocol utilizing just two wires for its operation. I^{2}C is a true multi-master bus providing arbitration and collision detection. It is half-duplex in communication. Different transfer rates are available depending on the speed mode. The I^{2}C speed rate is higher than the USART but lesser than the SPI. I^{2}C is mainly preferred in embedded applications where limited number of pins are available for communication and several devices have to be connected in a single bus.

1.3 SPI

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

2 SERCOM Implementation in SAM D Microcontrollers

Generally 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^{2}C will be a separate peripheral with its own dedicated pins.

In SAM D microcontrollers, all the serial peripherals are designed into a single module as serial communication interface (SERCOM). A SERCOM module can be either configured as USART, I^{2}C, or SPI selectable by 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 purpose and the SERCOM module will not control them unless it is configured to be used by the SERCOM module.

For example, SERCOM0 can be configured as USART mode with PAD0 as transmit pad and PAD1 as receive pad. Other unused pads (PAD2 and PAD3) can be either used as GPIO pins or can be assigned to some other peripherals. The assignment of SERCOM functionality for different pads is highly flexible making the SERCOM module more advantageous compared to the typical serial communication peripheral implementation.

2.1 SERCOM Overview

The serial communication interface (SERCOM) can be configured to support three different modes: I^{2}C, 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.

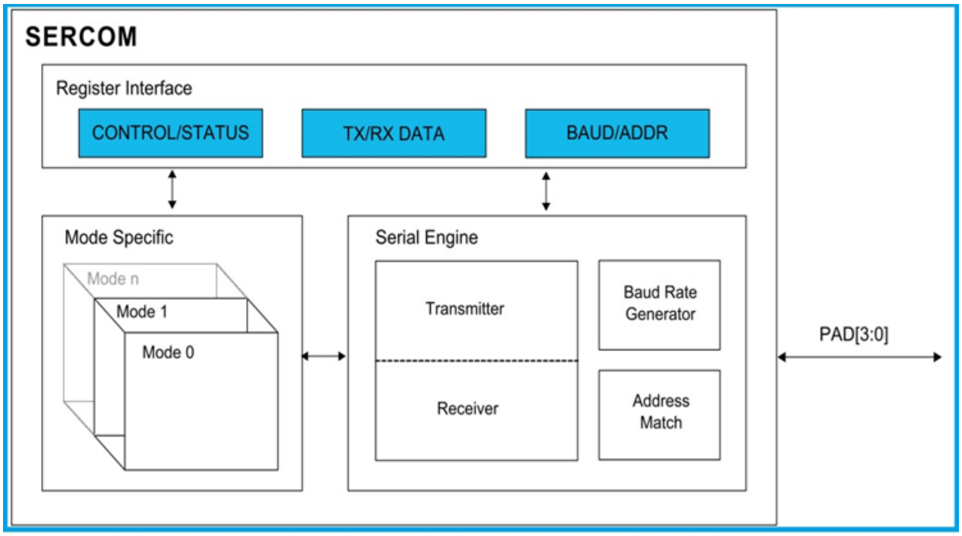
2.2 Features

- Combined interface 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
- Address match/mask logic
- Operational in all sleep modes
- Can be used with DMA (not supported in SAM D20 MCUs)

2.3 Block Diagram

Figure 2-1 depicts the block diagram of a SERCOM module. The module mainly consists of a serial engine handling the actual data transfers and mode specific IPs implementing the corresponding protocol.

Figure 2-1. SERCOM Block Diagram



2.4 Clocks

SERCOM module needs below clocks for its operation:

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

SERCOM bus clock (CLK_SERCOMx_APB) is disabled by default, but can be enabled and disabled in the Power Manager (PM) module.

Two generic clocks are used by the SERCOM module, namely GCLK_SERCOMx_CORE and GCLK_SERCOMx_SLOW.

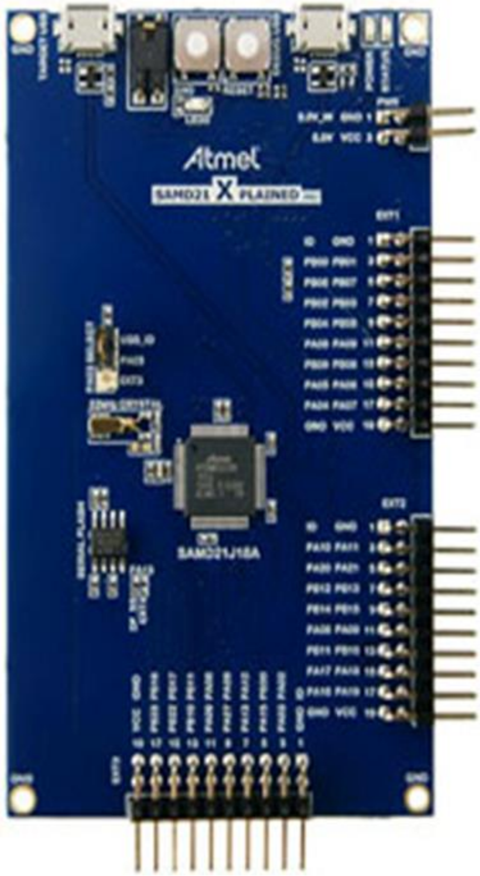
The core clock (GCLK_SERCOMx_CORE) is required to clock the SERCOM while operating as a master, while the slow clock (GCLK_SERCOMx_SLOW) is only required for certain functions like I^2C timeouts.

Note: In this application note only the SERCOM bus clock and core clock (GCLK_SERCOMx_CORE) are used.

3 Hardware and Software Requirements

The application demonstration needs two SAM D21 Xplained Pro boards. One board will be Master/Transmitter and the other board will be Slave/Receiver.

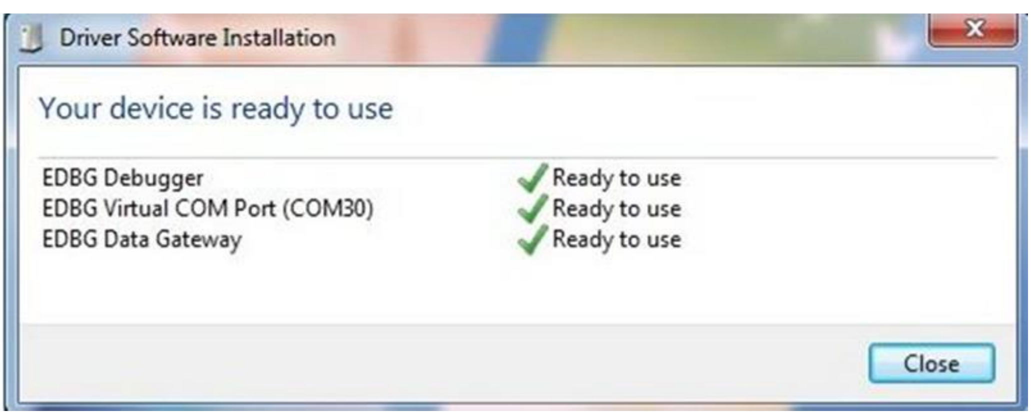
Figure 3-1. SAM D21 Xplained Pro Board



There are two USB ports in the SAM D21 Xplained Pro board; DEBUG USB and TARGET USB. For debugging using the Embedded debugger EDBG, the DEBUG USB port has to be connected.

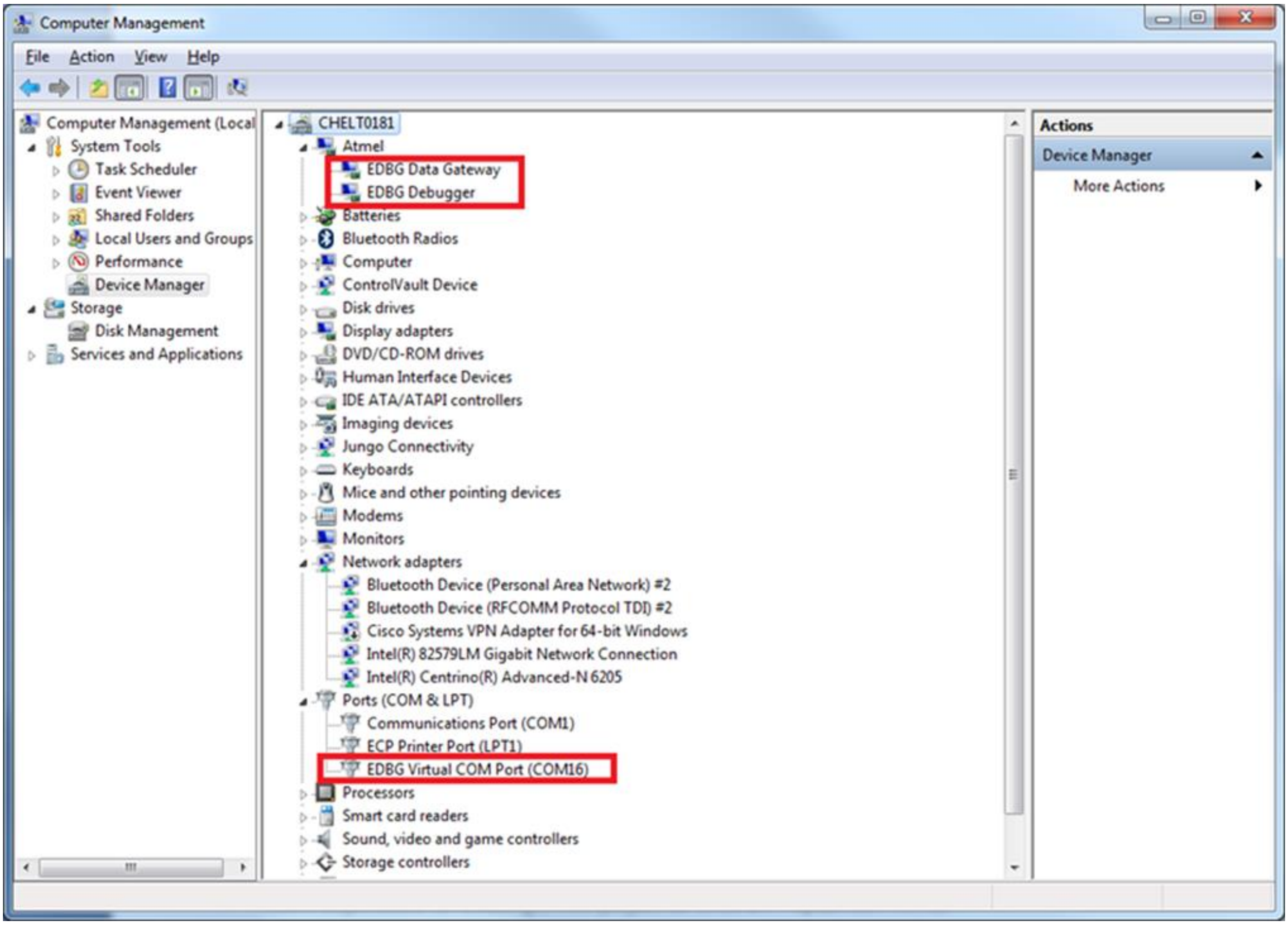
Once the kit is successfully connected the Windows® Task bar will pop-up a message as shown in Figure 3-2.

Figure 3-2. SAM D21 Xplained Pro Driver Installation



If the driver installation is proper, the EDBG will be listed in the Device Manager as shown in Figure 3-3.

Figure 3-3. Successful EDBG Driver Installation



Application codes are tested in Atmel Studio 6.2 with ASF version 3.22.0 and above.

Two projects are needed for implementing the functionalities, one for Master/Transmitter and the other for Slave/Receiver.

GCC C ASF Board project from Atmel studio is used for the implementation.

To create an ASF board project for SAM D21 Xplained Pro, go to File menu → New → Project and select “GCC C ASF Board project” in the new project wizard.

Figure 3-4. New Project in Atmel Studio

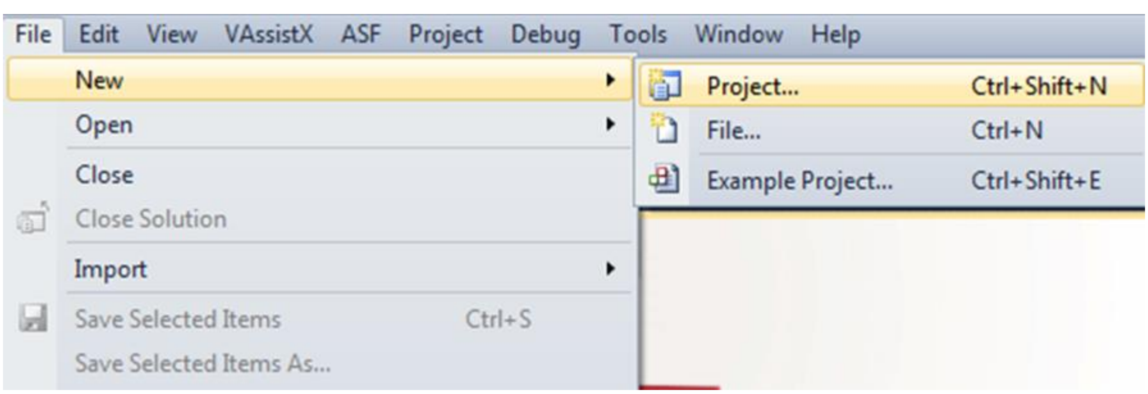
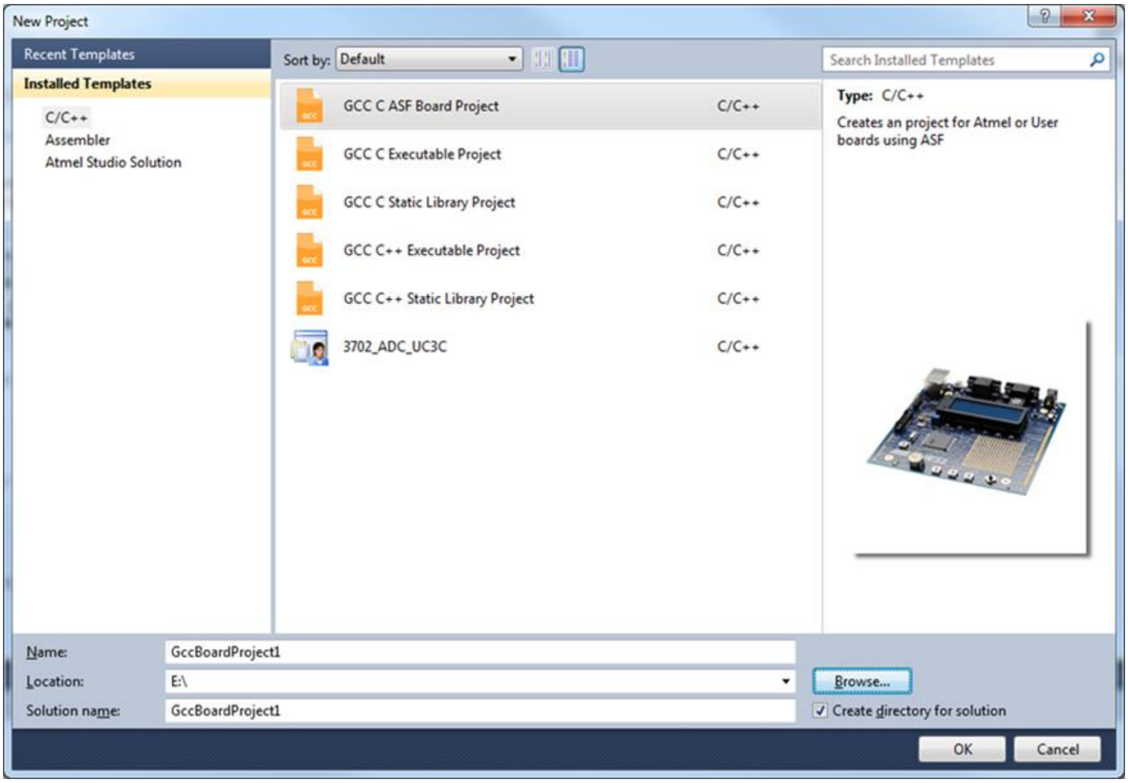
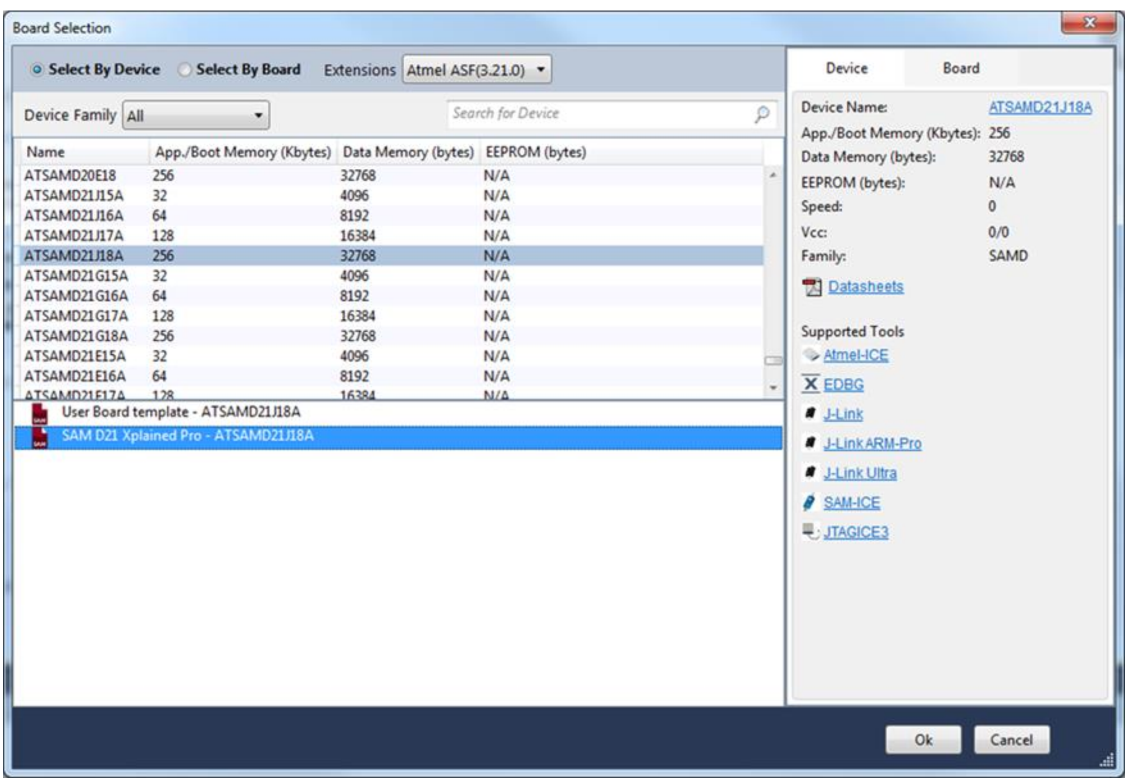


Figure 3-5. ASF Board Project



In the next window, select the device family as "SAM D", scroll down and select the device "ATSAMD21J18A" and board as "SAM D21 Xplained PRO - ATSAMD21J18A", and click on "OK" to create the new project.

Figure 3-6. Device and Board Selection



The new project by default has a minimal application that will turn on or off the LED in SAM D21 Xplained Pro based on the state of the SW0 button. Pressing the SW0 button will turn the LED on, and releasing the button will turn the LED off. To verify that the SAM D21 Xplained Pro is connected correctly this application can be run and checked whether it produces the expected output.

4 Application Demonstration

This chapter will demonstrate the various features of the SERCOM USART module of SAM D21 with different example codes.

- SERCOM USART Examples
  - Basic Configuration
  - Fraction baud configuration
  - Hardware Handshaking configuration (not available in SAM D20 devices)
  - SOF detection and wakeup configuration

Note: This chapter assumes that the user has previous knowledge on programming/debugging a SAM D21 device by using the Atmel Studio IDE.

For easier understanding, the examples will use register level coding for SERCOM module configuration. The clock configuration will, however, use the ASF functions.

4.1 Main Clock

In SAM D21 devices, the output from GCLK Generator 0 will be used as the main clock. The Generic Clock Generator 0, also called GCLK_MAIN, is the clock feeding the Power Manager used to generate synchronous clocks. The GCLK Generator 0 can have one of the SYSCTRL oscillators as its source clock. The following are the SYSCTRL clock sources.

- XOSC
- OSCULP32K
- OSC32K
- XOSC32K
- OSC8M
- DFLL48M
- FDPLL96M (not available in SAM D20 devices)

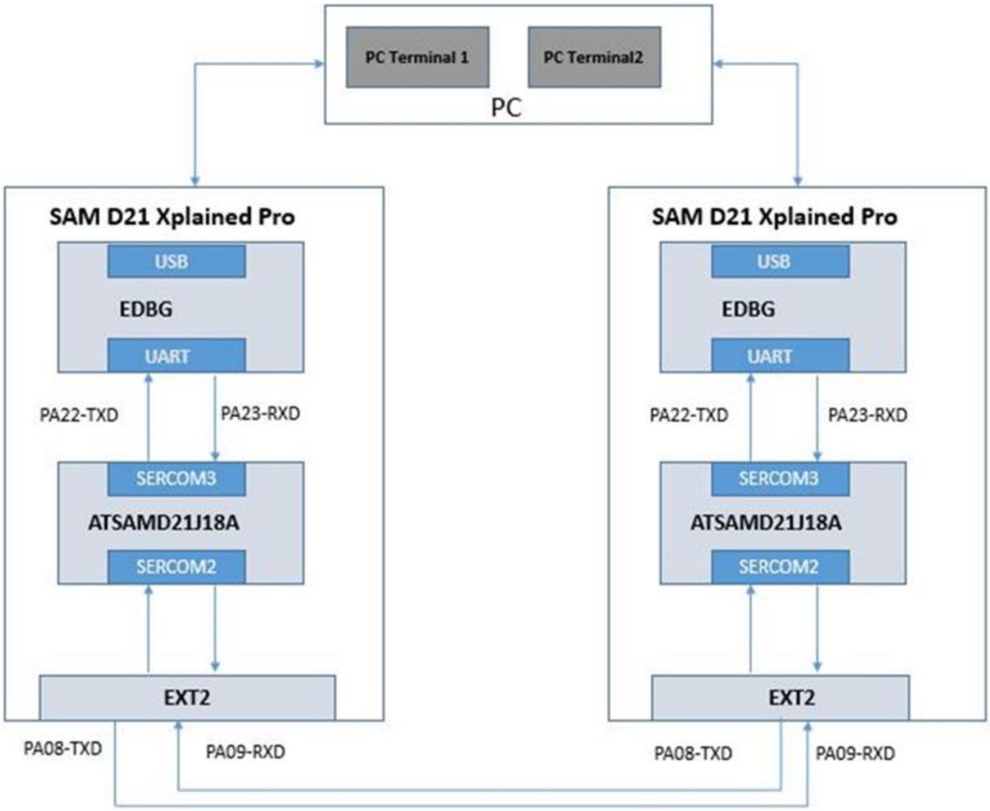
The application uses OSC8M as the clock source for Generator 0. The following lines in the conf_clocks.h will initialize the source clock for generator 0.

```
/* Configure GCLK generator 0 (Main Clock) */
#  define CONF_CLOCK_GCLK_0_ENABLE                true
#  define CONF_CLOCK_GCLK_0_RUN_IN_STANDBY        false
#  define CONF_CLOCK_GCLK_0_CLOCK_SOURCE          SYSTEM_CLOCK_SOURCE_OSC8M
#  define CONF_CLOCK_GCLK_0_PRESCALER             1
#  define CONF_CLOCK_GCLK_0_OUTPUT_ENABLE         false
```

4.2 Basic Configuration

Two SAM D21 Xplained Pro boards are connected to each other by SERCOM USART lines (TXD, RXD) through EXT2 connector and connected to the PC terminal through EDBG port as shown in Figure 4-1.

Figure 4-1. Block Diagram



Basic configuration section will have pin initialization, clock initialization, and the SERCOM USART initialization functions. Below are the function calls for the Basic configuration.

```
system_init()
edbg_usart_clock_init()
edbg_usart_pin_init()
edbg_usart_init()
ext_usart_clock_init()
ext_usart_pin_init()
ext_usart_init()
```

The below sections will summarize each function in detail.

4.2.1 System Initialization

`system_init()` is an ASF function which is used to configure the generic clocks and clock sources as per the settings in the conf_clocks.h file. The main clock will be configured as stated in Section 4.1. It also initializes the board hardware of SAM D21 Xplained Pro and the event system.

4.2.2 EDBG USART Clock Initialization

SERCOM3 is connected to the EDBG USART lines through which the SAM D21 Xplained pro will communicate to the PC terminal application. The following function initializes the clock for the SERCOM3 module.

```
/* EDBG UART(SERCOM3) bus and generic clock initialization */
void edbg_usart_clock_init(void)
{
	struct system_gclk_chan_config gclk_chan_conf;
	uint32_t gclk_index = SERCOM3_GCLK_ID_CORE;
```

```
        /* Turn on module in PM */
        system_apb_clock_set_mask(SYSTEM_CLOCK_APB_APBC, PM_APBCMASK_SERCOM3);
        /* Turn on Generic clock for USART */
        system_gclk_chan_get_config_defaults(&gclk_chan_conf);
        /*Default is generator 0. Other wise need to configure like below */
         /* gclk_chan_conf.source_generator = GCLK_GENERATOR_1; */
        system_gclk_chan_set_config(gclk_index, &gclk_chan_conf);
        system_gclk_chan_enable(gclk_index);

}
```

- A structure variable gclk_chan_conf is declared. This structure is used to configure the generic clock for the SERCOM used.
- EDBG USART is connected to SERCOM3, so SERCOM3 generic clock "`SERCOM3_GCLK_ID_CORE`" and bus clock "`SYSTEM_CLOCK_APB_APBC`" is configured
- Generic clock "`SERCOM3_GCLK_ID_CORE`" uses GCLK Generator 0 as source (generic clock source can be changed as per the user needs), so the SERCOM3 clock runs at 8MHz from OSC8M
- `system_gclk_chan_set_config` will set the generic clock channel configuration
- `system_gclk_chan_enable` will enable the generic clock index "`SERCOM3_GCLK_ID_CORE`"

4.2.3 EDBG USART Pin Initialization

SERCOM3 USART lines are connected to the EDBG. `edbg_usart_pin_init` function will initialize pins PA22 and PA23 to the SERCOM peripheral function.

```
/* EDBG UART (SERCOM3) pin initialization */
void edbg_usart_pin_init(void)
{
        /* PA22 and PA23 set into peripheral function C*/
        pin_set_peripheral_function(PINMUX_PA22C_SERCOM3_PAD0);
        pin_set_peripheral_function(PINMUX_PA23C_SERCOM3_PAD1);
}
```

`edbg_usart_pin_init` function calls the `pin_set_peripheral_function` to assign I/O lines PA22 and PA23 to the SERCOM peripheral function.

The function *pin_set_peripheral_function()* will switch the GPIO functionality of an I/O pin to peripheral functionality and assigns the given peripheral function to the pin. The function takes a 32-bit pinmux value as its argument. The 32-bit pinmux value contains the pin number in its 16-bit MSB part and the peripheral function number in its 16-bit LSB part. So each 32-bit pinmux value is unique per pin per peripheral function. The function first identifies the PORT group from the pin number (MSB 16-bit) and updates the PMUX register with the peripheral number (LSB 16-bit).

Note: `pin_set_peripheral_function` will assign the I/O pin to the alternate function only. The functionality of individual pin mapping should be done in the peripheral control register.

4.2.4 EDBG USART Initialization

The `edbg_usart_init` function will initialize the USART function by configuring the control registers, baud registers, and setting the respective interrupt enable bits.

```
/* EDBG(SERCOM3) UART initialization */
void edbg_usart_init(void)
{
        uint16_t baud_value;
        baud_value = calculate_baud_value(USART_BAUD_RATE,sys-
tem_gclk_chan_get_hz(SERCOM3_GCLK_ID_CORE),
                                                                 USART_SAMPLE_NUM);
```

```c
    /* By setting the DORD bit LSB is transmitted first and setting the RXPO bit as 1
    corresponding SERCOM PAD[1] will be used for data reception, PAD[0] will be used as TxD
    pin by setting TXPO bit as 0, 16x over-sampling is selected by setting the SAMPR bit as
    0,
    Generic clock is enabled in all sleep modes by setting RUNSTDBY bit as 1,
    USART clock mode is selected as USART with internal clock by setting MODE bit into 1.
    */
    SERCOM3->USART.CTRLA.reg = SERCOM_USART_CTRLA_DORD        |
                               SERCOM_USART_CTRLA_RXPO(0x1) |
                               SERCOM_USART_CTRLA_TXPO(0x0) |
                               SERCOM_USART_CTRLA_SAMPR(0x0)|
                               SERCOM_USART_CTRLA_RUNSTDBY   |
                               SERCOM_USART_CTRLA_MODE_USART_INT_CLK ;

    /*baud register value corresponds to the device communication baud rate */
    SERCOM3->USART.BAUD.reg = baud_value;

    /* 8-bits size is selected as character size by setting the bit CHSIZE as 0,
    TXEN bit and RXEN bits are set to enable the Transmitter and receiver*/
    SERCOM3->USART.CTRLB.reg = SERCOM_USART_CTRLB_CHSIZE(0x0) |
                               SERCOM_USART_CTRLB_TXEN        |
                               SERCOM_USART_CTRLB_RXEN ;
    /* synchronization busy */
    while(SERCOM3->USART.SYNCBUSY.bit.CTRLB);
    /* SERCOM3 handler enabled */
    system_interrupt_enable(SERCOM3_IRQn);
    /* receive complete interrupt set */
    SERCOM3->USART.INTENSET.reg = SERCOM_USART_INTFLAG_RXC;
    /* SERCOM3 peripheral enabled */
    SERCOM3->USART.CTRLA.reg |= SERCOM_USART_CTRLA_ENABLE;
    /* synchronization busy */
    while(SERCOM3->USART.SYNCBUSY.reg & SERCOM_USART_SYNCBUSY_ENABLE);
}
```

- The communication between the SAM D21 Xplained Pro and PC terminal is done with the arithmetic baud rate of `USART_BAUD_RATE` – 9600 bps. The value in the baud register is calculated by taking the values of `USART_BAUD_RATE`, SERCOM3 generic clock, `USART_SAMPLE_NUM`.

  Arithmetic BAUD Rate formulae is given by the below equation.

  $F_{BAUD} = (f_{REF}/ S) (1 – BAUD/65{,}536)$

  F_{BAUD} = baud frequency

  f_{ref} – SERCOM generic clock frequency

  S – Number of samples per bit

  BAUD – BAUD register value

- `calculate_baud_value` function is used to do this manipulation
- `system_gclk_chan_get_hz` (`SERCOM3_GCLK_ID_CORE`) function will return the SERCOM3 generic clock frequency
- The CTRLA register is used to configure data order transmission, TxD and RxD pads, sampling rate, run in standby mode, and USART clock selection. In the above the function data order is set as MSB, SERCOM PAD[1] is used as RxD line, SERCOM PAD[0] is used as TxD line,16x over sampling is used, Generic clock is enabled in all sleep modes, and internal clock is used for the USART.
- The CTRLB register is used to configure character size and transmitter and receiver enable. In the above the function character size is configured as eight bits and the application needs to transmit and receive so both the transmitter and the receiver are enabled.

- Each peripheral has dedicated interrupt line which is connected to the **Nested Vector Interrupt Controller** in the Cortex®-M0+ core
- In the above function SERCOM3 interrupt request line (IRQ - 12) is enabled.
- The INTENSET register is used to enable the required interrupts. In the above function RXC - Receive complete interrupt will be set. This is data from PC terminal that will be received by the EDBG, so once receiving the data, interrupt should be triggered to notify the CPU.
- CTRLA, CTRLB, and BAUD registers can be written only when the USART is disabled because these registers are enable protected. So once configuring these registers, the USART is enabled.
- Due to the asynchronicity between CLK_SERCOMx_APB and GCLK_SERCOMx_CORE, some registers must be synchronized when accessed. CTRLA register is Write-Synchronized so it needs to check the synchronization busy.

4.2.5 External USART Clock Initialization

SAM D21 Xplained Pro communicates with the other SAM D21 Xplained pro boards through EXT2 connector present in it as shown in Figure 4-1. SERCOM2 lines are connected to the EXT2 connector.

```
/* External connector(SERCOM2) UART bus and generic clock initialization */
void ext_usart_clock_init(void)
{
	struct system_gclk_chan_config gclk_chan_conf;
	uint32_t gclk_index = SERCOM2_GCLK_ID_CORE;
	/* Turn on module in PM */
	system_apb_clock_set_mask(SYSTEM_CLOCK_APB_APBC, PM_APBCMASK_SERCOM2);
	/* Turn on Generic clock for USART */
	system_gclk_chan_get_config_defaults(&gclk_chan_conf);
	//Default is generator 0. Other wise need to configure like below
	/* gclk_chan_conf.source_generator = GCLK_GENERATOR_1; */
	system_gclk_chan_set_config(gclk_index, &gclk_chan_conf);
	system_gclk_chan_enable(gclk_index);
}
```

This section is same as Section 4.2.2. The change is SERCOM2 core clock will be used by the application.

4.2.6 External USART Pin Initialization

SERCOM2 USART lines are connected to the EXT2 connector. The `edbg_usart_pin_init` function will initialize pins PA08 and PA09 to SERCOM peripheral function.

```
/* External connector(SERCOM2) pin initialization */
void ext_usart_pin_init(void)
{
	/* PA08 and PA09 set into peripheral function C */
	pin_set_peripheral_function(PINMUX_PA08D_SERCOM2_PAD0);
	pin_set_peripheral_function(PINMUX_PA09D_SERCOM2_PAD1);
}
```

The `ext_usart_pin_init` function calls the `pin_set_peripheral_function` to assign I/O lines PA08 and PA09 into the SERCOM peripheral function.

4.2.7 External USART Initialization

The `ext_usart_init` function will initialize the USART function by configuring the control registers, baud registers, and setting the respective interrupt flags.

```
/* External connector(SERCOM2) UART initialization */
void ext_usart_init(void)
{
	uint16_t baud_value;
```

```
        baud_value = calculate_baud_value(USART_BAUD_RATE,sys-
tem_gclk_chan_get_hz(SERCOM2_GCLK_ID_CORE),
        USART_SAMPLE_NUM);
        /* By setting the DORD bit LSB is transmitted first and setting the RXPO bit as 1 correspond-
        ing SERCOM PAD[1] will be used for data reception RXD, PAD[0] will be used as TxD pin by set-
        ting TXPO bit as 0,16x over-sampling is selected by setting the SAMPR bit as 0,
         Generic clock is enabled in all sleep modes by setting RUNSTDBY bit as 1,
         USART clock mode is selected as USART with internal clock by setting MODE bit into 1.
                */
        SERCOM2->USART.CTRLA.reg = SERCOM_USART_CTRLA_DORD      |
                                   SERCOM_USART_CTRLA_RXPO(0x1) |
                                   SERCOM_USART_CTRLA_TXPO(0x0) |
                                   SERCOM_USART_CTRLA_SAMPR(0x0)|
                                   SERCOM_USART_CTRLA_RUNSTDBY  |
                                   SERCOM_USART_CTRLA_MODE_USART_INT_CLK ;
        /* baud register value corresponds to the device communication baud rate */
        SERCOM2->USART.BAUD.reg = baud_value;
        /* 8-bits size is selected as character size by setting the bit CHSIZE as 0,
        TXEN bit and RXEN bits are set to enable the Transmitter and receiver*/
        SERCOM2->USART.CTRLB.reg = SERCOM_USART_CTRLB_CHSIZE(0x0) |
                                    SERCOM_USART_CTRLB_TXEN       |
                                    SERCOM_USART_CTRLB_RXEN ;
        /* synchronization busy */
        while(SERCOM2->USART.SYNCBUSY.bit.CTRLB);
        /* SERCOM2 handler enabled */
        system_interrupt_enable(SERCOM2_IRQn);
        /* receive complete interrupt set */
        SERCOM2->USART.INTENSET.reg = SERCOM_USART_INTFLAG_RXC;
        /* SERCOM2 peripheral enabled */
        SERCOM2->USART.CTRLA.reg |= SERCOM_USART_CTRLA_ENABLE;
        /* synchronization busy */
        while(SERCOM2->USART.SYNCBUSY.reg & SERCOM_USART_SYNCBUSY_ENABLE);

}
```

This section is same as Section 4.2.4. Here SERCOM2 PADS will be configured for USART and SERCOM2 interrupt line will be enabled.

4.2.8 SERCOM Interrupt Handlers

Following are the SERCOM2 and SERCOM3 handlers used in the application.

```
/*ext_usart handler*/
void SERCOM2_Handler()
{
        if (SERCOM2->USART.INTFLAG.bit.RXC){
                ext_rx_data = SERCOM2->USART.DATA.reg;
                if (SERCOM3->USART.INTFLAG.bit.DRE)
                {
                        SERCOM3->USART.DATA.reg = ext_rx_data;
                }
        }

}

/*edbg_usart handler*/
void SERCOM3_Handler()
{
        if (SERCOM3->USART.INTFLAG.bit.RXC){
                edbg_rx_data = SERCOM3->USART.DATA.reg;
```

```
                if (SERCOM2->USART.INTFLAG.bit.DRE)
                {
                        SERCOM2->USART.DATA.reg = edbg_rx_data;
                }
        }

}
```

In this application, character from PC terminal 1 is sent to SAM D21 Xplained pro (Transmitter) through EDBG USART (SERCOM3). This character will reach the other SAM D21 Xplained pro (receiver) through EXT2 connector between two boards.

Now the character from SAM D21 Xplained pro (receiver) will reach the other PC terminal 2.

In short, character from PC terminal 1 reach the PC terminal 2 through the SAM D21 boards. Similarly character from PC terminal 2 reach the PC terminal 1 in the same way.

Once pressing a character in the PC terminal 1 it will reach the SAM D21 Xplained pro (Transmitter) through EDBG. Once the EDBG USART receives the character since the RXC interrupt of the SERCOM3 is enabled, the application executes the `SERCOM3_Handler`.

- In the `SERCOM3_Handler` it will check for the RXC flag set condition in INTFLAG register for SERCOM3. This bit will be set when the character pressed in terminal 1 is received completely by SERCOM3.
- The received character will be in the DATA register of SERCOM3 and it is read into the variable `edbg_rx_data`.
- Now DRE – Data Register Empty interrupt flag of SERCOM2 which is connected to EXT2 connector will be checked for the set condition. This bit will be set when the DATA register of SERCOM2 is empty and ready to be written.
- If this bit DRE is set then the data `edbg_rx_data` will be placed in the SERCOM2 DATA register. This data is passed into the other SAM D21 Xplained pro (receiver).
- Now RXC flag of SERCOM2 will be set in the INTFLAG register of other SAM D21 board and the `SERCOM2_Handler` will be serviced
- In the SERCOM2_Handler once checking the RXC set condition, the data from the SERCOM2 DATA register will be written into variable `ext_rx_data`
- Now DRE – Data Register Empty of SERCOM3 which is connected to EDBG will be checked for the set condition. This bit will be set when the DATA register of SERCOM3 is empty and ready to be written.
- If this bit DRE is set then the data `ext_rx_data` will be placed in the SERCOM3 DATA register. This data is passed to the PC terminal 2 through EDBG.

When a character is pressed in the PC terminal 2 the same above sequence will happen and the character will reach the PC terminal 1.

The application code can handle both transmission and reception, so both the SAM D21 Xplained boards can be flashed with the same binary.

Note: This application is also tested with the file like .Txt apart from character.

The final application “Basic Configuration” in main.c file will be as below.

```
#include <asf.h>
#define USART_BAUD_RATE 9600
#define USART_SAMPLE_NUM 16
#define SHIFT 32
uint8_t edbg_rx_data,ext_rx_data;

/* function prototype */
void edbg_usart_clock_init(void);
void edbg_usart_pin_init(void);
void edbg_usart_init(void);
```

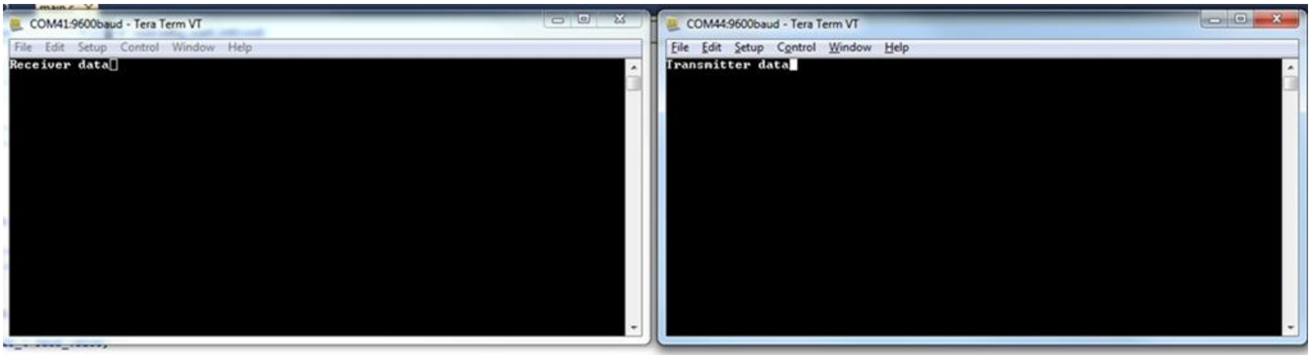
```c
void ext_usart_clock_init(void);
void ext_usart_pin_init(void);
void ext_usart_init(void);
uint16_t calculate_baud_value(const uint32_t baudrate,const uint32_t peripheral_clock,
                                                                    uint8_t sample_num);

/*ext_usart handler*/
void SERCOM2_Handler()
{
       if (SERCOM2->USART.INTFLAG.bit.RXC){
              ext_rx_data = SERCOM2->USART.DATA.reg;
              if (SERCOM3->USART.INTFLAG.bit.DRE)
              {
                     SERCOM3->USART.DATA.reg = ext_rx_data;
              }
       }

}

/*edbg_usart handler*/
void SERCOM3_Handler()
{
       if (SERCOM3->USART.INTFLAG.bit.RXC){
              edbg_rx_data = SERCOM3->USART.DATA.reg;
              if (SERCOM2->USART.INTFLAG.bit.DRE)
              {
                     SERCOM2->USART.DATA.reg = edbg_rx_data;
              }
       }

}

/*Assigning pin to the alternate peripheral function*/
static inline void pin_set_peripheral_function(uint32_t pinmux)
{
       uint8_t port = (uint8_t)((pinmux >> 16)/32);
       PORT->Group[port].PINCFG[((pinmux >> 16) - (port*32))].bit.PMUXEN = 1;
       PORT->Group[port].PMUX[((pinmux >> 16) - (port*32))/2].reg &= ~(0xF << (4 * ((pinmux >>
16) & 0x01u)));
       PORT->Group[port].PMUX[((pinmux >> 16) - (port*32))/2].reg |= (uint8_t)((pinmux &
0x0000FFFF) << (4 * ((pinmux >> 16) & 0x01u)));
}

/*
 * internal Calculate 64 bit division, ref can be found in
 * http://en.wikipedia.org/wiki/Division_algorithm#Long_division
 */
static uint64_t long_division(uint64_t n, uint64_t d)
{
       int32_t i;
       uint64_t q = 0, r = 0, bit_shift;
       for (i = 63; i >= 0; i--) {
              bit_shift = (uint64_t)1 << i;

              r = r << 1;

              if (n & bit_shift) {
                     r |= 0x01;
              }
```

```
                if (r >= d) {
                        r = r - d;
                        q |= bit_shift;
                }
        }

        return q;
}

/*
 * \internal Calculate asynchronous baudrate value (UART)
*/
uint16_t calculate_baud_value(
                const uint32_t baudrate,
                const uint32_t peripheral_clock,
                uint8_t sample_num)
{
        /* Temporary variables  */
        uint64_t ratio = 0;
        uint64_t scale = 0;
        uint64_t baud_calculated = 0;
        uint64_t temp1;


                /* Calculate the BAUD value */
                temp1 = ((sample_num * (uint64_t)baudrate) << SHIFT);
                ratio = long_division(temp1, peripheral_clock);
                scale = ((uint64_t)1 << SHIFT) - ratio;
                baud_calculated = (65536 * scale) >> SHIFT;

        return baud_calculated;
}

/* EDBG UART(SERCOM3) bus and generic clock initialization */
void edbg_usart_clock_init(void)
{
                struct system_gclk_chan_config gclk_chan_conf;
                uint32_t gclk_index = SERCOM3_GCLK_ID_CORE;
                /* Turn on module in PM */
                system_apb_clock_set_mask(SYSTEM_CLOCK_APB_APBC, PM_APBCMASK_SERCOM3);

                /* Turn on Generic clock for USART */
                system_gclk_chan_get_config_defaults(&gclk_chan_conf);
                /*Default is generator 0. Other wise need to configure like below */
                 /* gclk_chan_conf.source_generator = GCLK_GENERATOR_1; */
                system_gclk_chan_set_config(gclk_index, &gclk_chan_conf);
                system_gclk_chan_enable(gclk_index);
}

/* EDBG UART(SERCOM3) pin initialization */
void edbg_usart_pin_init(void)
{
    /* PA22 and PA23 set into peripheral function C */
        pin_set_peripheral_function(PINMUX_PA22C_SERCOM3_PAD0);
        pin_set_peripheral_function(PINMUX_PA23C_SERCOM3_PAD1);
}

/* EDBG(SERCOM3) UART initialization */
```

```
void edbg_usart_init(void)
{
	uint16_t baud_value;
	baud_value = calculate_baud_value(USART_BAUD_RATE,sys-
tem_gclk_chan_get_hz(SERCOM3_GCLK_ID_CORE),
	USART_SAMPLE_NUM);
	/* By setting the DORD bit LSB is transmitted first and setting the RXPO bit as 1 cor-
responding SERCOM PAD[1] will be used for data reception, PAD[0] will be used as TxD pin by
setting TXPO bit as 0,16x over-sampling is selected by setting the SAMPR bit as 0,
 Generic clock is enabled in all sleep modes by setting RUNSTDBY bit as 1,
USART clock mode is selected as USART with internal clock by setting MODE bit into 1.
	*/
	SERCOM3->USART.CTRLA.reg = SERCOM_USART_CTRLA_DORD	|
				SERCOM_USART_CTRLA_RXPO(0x1) |
				SERCOM_USART_CTRLA_TXPO(0x0) |
				SERCOM_USART_CTRLA_SAMPR(0x0)|
				SERCOM_USART_CTRLA_RUNSTDBY	|
				SERCOM_USART_CTRLA_MODE_USART_INT_CLK ;

	/*baud register value corresponds to the device communication baud rate */
	SERCOM3->USART.BAUD.reg = baud_value;

	/* 8-bits size is selected as character size by setting the bit CHSIZE as 0,
	TXEN bit and RXEN bits are set to enable the Transmitter and receiver*/
	SERCOM3->USART.CTRLB.reg = SERCOM_USART_CTRLB_CHSIZE(0x0) |
				SERCOM_USART_CTRLB_TXEN	|
				SERCOM_USART_CTRLB_RXEN ;
	/* synchronization busy */
	while(SERCOM3->USART.SYNCBUSY.bit.CTRLB);
	/* SERCOM3 handler enabled */
	system_interrupt_enable(SERCOM3_IRQn);
	/* receive complete interrupt set */
	SERCOM3->USART.INTENSET.reg = SERCOM_USART_INTFLAG_RXC;
	/* SERCOM3 peripheral enabled */
	SERCOM3->USART.CTRLA.reg |= SERCOM_USART_CTRLA_ENABLE;
	/* synchronization busy */
	while(SERCOM3->USART.SYNCBUSY.reg & SERCOM_USART_SYNCBUSY_ENABLE);
}

/* External connector(SERCOM2) UART bus and generic clock initialization */
void ext_usart_clock_init(void)
{
	struct system_gclk_chan_config gclk_chan_conf;
	uint32_t gclk_index = SERCOM2_GCLK_ID_CORE;
	/* Turn on module in PM */
	system_apb_clock_set_mask(SYSTEM_CLOCK_APB_APBC, PM_APBCMASK_SERCOM2);
	/* Turn on Generic clock for USART */
	system_gclk_chan_get_config_defaults(&gclk_chan_conf);
	//Default is generator 0. Other wise need to configure like below
	/* gclk_chan_conf.source_generator = GCLK_GENERATOR_1; */
	system_gclk_chan_set_config(gclk_index, &gclk_chan_conf);
	system_gclk_chan_enable(gclk_index);
}

/* External connector(SERCOM2) pin initialization */
void ext_usart_pin_init(void)
{
	/* PA08 and PA09 set into peripheral function*/
	pin_set_peripheral_function(PINMUX_PA08D_SERCOM2_PAD0);
```

```
        pin_set_peripheral_function(PINMUX_PA09D_SERCOM2_PAD1);
}

/* External connector(SERCOM2) UART initialization */
void ext_usart_init(void)
{
        uint16_t baud_value;
        baud_value = calculate_baud_value(USART_BAUD_RATE,sys-
tem_gclk_chan_get_hz(SERCOM2_GCLK_ID_CORE),
        USART_SAMPLE_NUM);
                /* By setting the DORD bit LSB is transmitted first and setting the RXPO bit as
1 corresponding SERCOM PAD[1] will be used for data reception RXD, PAD[0] will be used as TxD
pin by setting TXPO bit as 0, 16x over-sampling is selected by setting the SAMPR bit as 0,
Generic clock is enabled in all sleep modes by setting RUNSTDBY bit as 1,
USART clock mode is selected as USART with internal clock by setting MODE bit into 1.
                */
        SERCOM2->USART.CTRLA.reg = SERCOM_USART_CTRLA_DORD      |
                                    SERCOM_USART_CTRLA_RXPO(0x1) |
                                   SERCOM_USART_CTRLA_TXPO(0x0) |
                                   SERCOM_USART_CTRLA_SAMPR(0x0)|
                                   SERCOM_USART_CTRLA_RUNSTDBY   |
                                  SERCOM_USART_CTRLA_MODE_USART_INT_CLK ;

        /* baud register value corresponds to the device communication baud rate */
        SERCOM2->USART.BAUD.reg = baud_value;
        /* 8-bits size is selected as character size by setting the bit CHSIZE as 0,
        TXEN bit and RXEN bits are set to enable the Transmitter and receiver*/
        SERCOM2->USART.CTRLB.reg = SERCOM_USART_CTRLB_CHSIZE(0x0) |
                                    SERCOM_USART_CTRLB_TXEN        |
                                    SERCOM_USART_CTRLB_RXEN ;
        /* synchronization busy */
        while(SERCOM2->USART.SYNCBUSY.bit.CTRLB);
        /* SERCOM2 handler enabled */
        system_interrupt_enable(SERCOM2_IRQn);
        /* receive complete interrupt set */
        SERCOM2->USART.INTENSET.reg = SERCOM_USART_INTFLAG_RXC;
        /* SERCOM2 peripheral enabled */
        SERCOM2->USART.CTRLA.reg |= SERCOM_USART_CTRLA_ENABLE;
        /* synchronization busy */
        while(SERCOM2->USART.SYNCBUSY.reg & SERCOM_USART_SYNCBUSY_ENABLE);

}
int main (void)
{
        system_init();
        edbg_usart_clock_init();
        edbg_usart_pin_init();
        edbg_usart_init();
        ext_usart_clock_init();
        ext_usart_pin_init();
        ext_usart_init();
        while(1);
}
```

The PC terminal output will look like below.



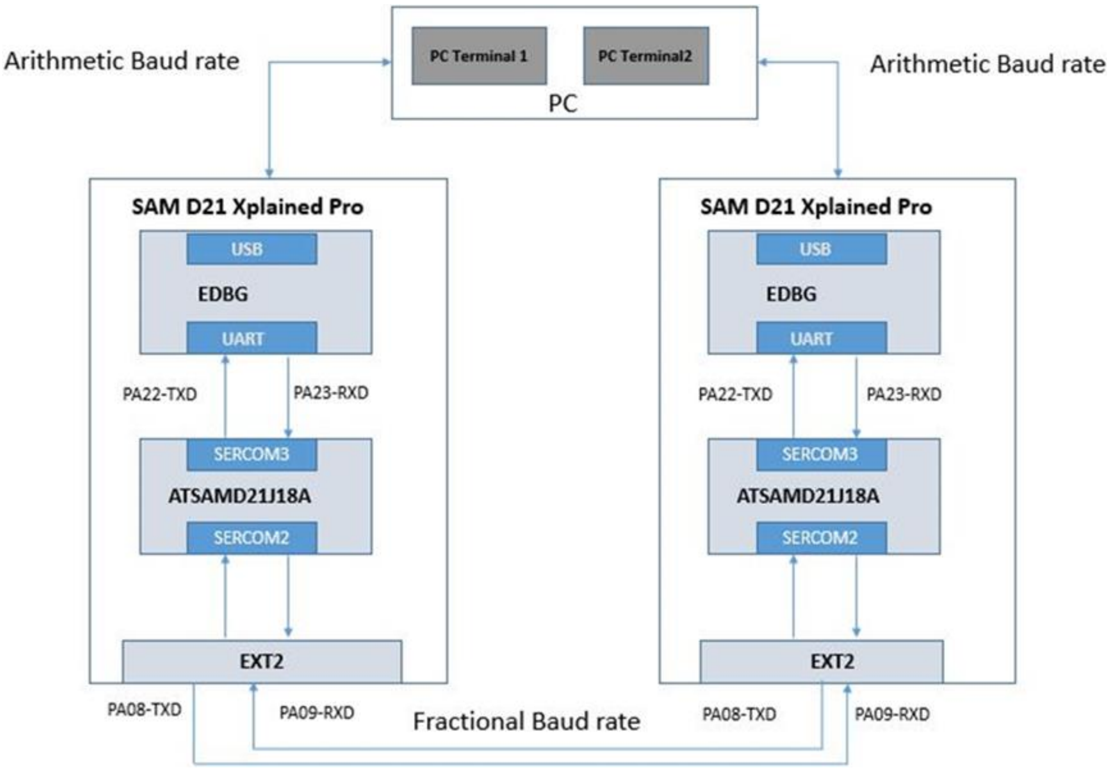
The Complete Project solution can be found in the zipped folder attachment that comes with this application note.

4.3 Fraction Baud Configuration

This section is similar to the Basic configuration except the baud rate configuration between the two SAM D21 Xplained Pro boards.

The Communication baud rate between the PC terminal and SAM D21 Xplained pro board will be arithmetic baud rate, whereas the communication between the two SAM D21 Xplained Pro boards are fractional baud rate.

Figure 4-2. Block Diagram



Only the EXT2 connector, which is of SERCOM2 will be in the fractional baud rate.

Note: The PC terminal will not support non-standard baud rates.

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.

Fractional baud equation should be used to calculate the baud value.

Fractional baud $f_{baud} = f_{ref} / S(BAUD+(FP/8))$

f_{baud} – fractional baud frequency

f_{ref} – 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 $= f_{ref} / (f_{baud} \times S)$

= 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

Here the BAUD value corresponds to the BAUD [12:0] and the FP value corresponds to FP [2:0] in the BAUD register.

```
/* fractional baud value calculation */
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)(bau
drate*sample_num);
       baud = mul_ratio/1000;
       fp = ((mul_ratio - (baud*1000))*8)/1000;
}
```

The below line in the application code is used to configure the BAUD register with the fractional baud value.

```
SERCOM2->USART.BAUD.reg = SERCOM_USART_BAUD_FRAC_BAUD(baud) | SERCOM_USART_BAUD_FRAC_FP(fp);
```

The application code can handle both transmission and reception, so both the SAM D21 Xplained boards can be flashed with the same binary.

The final application “Fractional baud Configuration” in main.c file will be as below.

```
#include <asf.h>
#define USART_BAUD_RATE 9600
#define FRAC_BAUD_RATE 11000
#define USART_SAMPLE_NUM 16
#define SHIFT 32

uint8_t edbg_rx_data,ext_rx_data;
uint16_t baud;
uint8_t fp;

/* Function prototype */
```

```
void edbg_usart_clock_init(void);
void edbg_usart_pin_init(void);
void edbg_usart_init(void);
void ext_usart_clock_init(void);
void ext_usart_pin_init(void);
void ext_usart_init(void);
uint16_t calculate_baud_value(const uint32_t baudrate,const uint32_t peripheral_clock,
                                                                    uint8_t sample_num);
void calculate_fractional_baud_value(const uint32_t baudrate,const uint32_t periphe-
ral_clock,uint8_t sample_num);

/*ext_usart handler*/
void SERCOM2_Handler()
{
       if (SERCOM2->USART.INTFLAG.bit.RXC){
              ext_rx_data = SERCOM2->USART.DATA.reg;
              if (SERCOM3->USART.INTFLAG.bit.DRE)
              {
                     SERCOM3->USART.DATA.reg = ext_rx_data;
              }
       }

}

/*edbg_usart handler*/
void SERCOM3_Handler()
{
       if (SERCOM3->USART.INTFLAG.bit.RXC){
              edbg_rx_data = SERCOM3->USART.DATA.reg;
              if (SERCOM2->USART.INTFLAG.bit.DRE)
              {
                     SERCOM2->USART.DATA.reg = edbg_rx_data;
              }
       }

}

/*Assigning pin to the alternate peripheral function*/
static inline void pin_set_peripheral_function(uint32_t pinmux)
{
       uint8_t port = (uint8_t)((pinmux >> 16)/32);
       PORT->Group[port].PINCFG[((pinmux >> 16) - (port*32))].bit.PMUXEN = 1;
       PORT->Group[port].PMUX[((pinmux >> 16) - (port*32))/2].reg &= ~(0xF << (4 * ((pinmux >>
16) & 0x01u)));
       PORT->Group[port].PMUX[((pinmux >> 16) - (port*32))/2].reg |= (uint8_t)((pinmux &
0x0000FFFF) << (4 * ((pinmux >> 16) & 0x01u)));
}

/*
 * internal Calculate 64 bit division, ref can be found in
 * http://en.wikipedia.org/wiki/Division_algorithm#Long_division
 */
static uint64_t long_division(uint64_t n, uint64_t d)
{
       int32_t i;
       uint64_t q = 0, r = 0, bit_shift;
       for (i = 63; i >= 0; i--) {
              bit_shift = (uint64_t)1 << i;
```

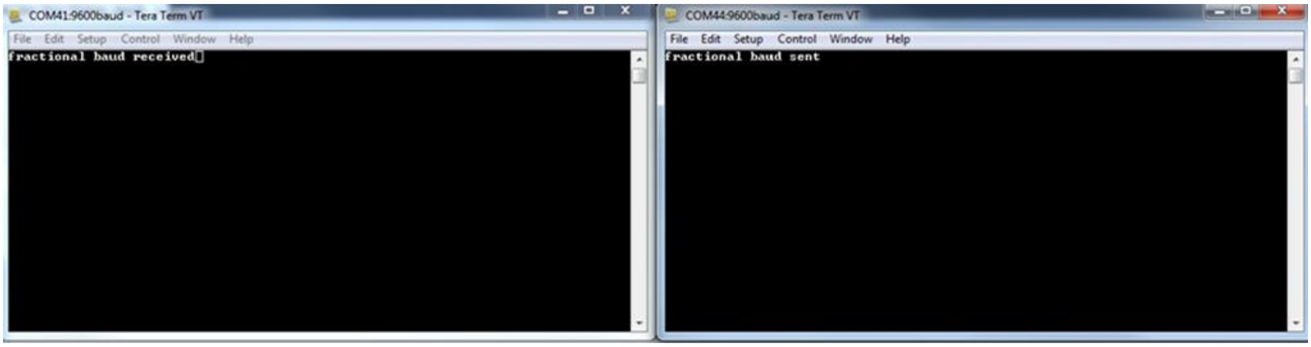
```
                r = r << 1;

                if (n & bit_shift) {
                        r |= 0x01;
                }

                if (r >= d) {
                        r = r - d;
                        q |= bit_shift;
                }
        }

        return q;
}

/*
 * internal Calculate asynchronous baudrate value (UART)
*/
uint16_t calculate_baud_value(
                const uint32_t baudrate,
                const uint32_t peripheral_clock,
                uint8_t sample_num)
{
        /* Temporary variables  */
        uint64_t ratio = 0;
        uint64_t scale = 0;
        uint64_t baud_calculated = 0;
        uint64_t temp1;


                /* Calculate the BAUD value */
                temp1 = ((sample_num * (uint64_t)baudrate) << SHIFT);
                ratio = long_division(temp1, peripheral_clock);
                scale = ((uint64_t)1 << SHIFT) - ratio;
                baud_calculated = (65536 * scale) >> SHIFT;

        return baud_calculated;
}


/* As per the table 24-2 Baud rate equations */
/* Fbaud = Fref/S(BAUD+(FP/8))
   fbaud - fractional baud frequency
   fref  - SERCOM generic clock frequency
   S    - Number of samples per bit
   BAUD - BAUD value
   FP - fractional part of baud value
 */
/* fractional baud value calculation */
void calculate_fractional_baud_value(const uint32_t baudrate,const uint32_t periph-
eral_clock,uint8_t sample_num)
{
        uint64_t mul_ratio;
        mul_ratio = (uint64_t)((uint64_t)peripheral_clock*(uint64_t)1000)/(uint64_t)(bau-
drate*sample_num);
        baud = mul_ratio/1000;
        fp = ((mul_ratio - (baud*1000))*8)/1000;
}
```

```
/* EDBG UART(SERCOM3) bus and generic clock initialization */
void edbg_usart_clock_init(void)
{

                struct system_gclk_chan_config gclk_chan_conf;
                uint32_t gclk_index = SERCOM3_GCLK_ID_CORE;
                /* Turn on module in PM */
                system_apb_clock_set_mask(SYSTEM_CLOCK_APB_APBC, PM_APBCMASK_SERCOM3);
                /* Turn on Generic clock for USART */
                system_gclk_chan_get_config_defaults(&gclk_chan_conf);
                //Default is generator 0. Other wise need to configure like below
                 /* gclk_chan_conf.source_generator = GCLK_GENERATOR_1; */
                system_gclk_chan_set_config(gclk_index, &gclk_chan_conf);
                system_gclk_chan_enable(gclk_index);
}

/* EDBG UART(SERCOM3) pin initialization */
void edbg_usart_pin_init(void)
{
       /* PA22 and PA23 set into peripheral function C*/
       pin_set_peripheral_function(PINMUX_PA22C_SERCOM3_PAD0);
       pin_set_peripheral_function(PINMUX_PA23C_SERCOM3_PAD1);
}

/* EDBG UART(SERCOM3) initialization */
void edbg_usart_init(void)
{
       uint16_t baud_value;
       baud_value = calculate_baud_value(USART_BAUD_RATE,sys-
tem_gclk_chan_get_hz(SERCOM3_GCLK_ID_CORE),
                                                                     USART_SAMPLE_NUM);
       /* By setting the DORD bit LSB is transmitted first and setting the RXPO bit as 1 cor-
responding SERCOM PAD[1] will be used for data reception RXD, PAD[0] will be used as TxD pin
by setting TXPO bit as 0,16x over-sampling is selected by setting the SAMPR bit as 0,
Generic clock is enabled in all sleep modes by setting RUNSTDBY bit as 1,
USART clock mode is selected as USART with internal clock by setting MODE bit into 1.
       */
       SERCOM3->USART.CTRLA.reg = SERCOM_USART_CTRLA_DORD      |
                                  SERCOM_USART_CTRLA_RXPO(0x1) |
                                  SERCOM_USART_CTRLA_TXPO(0x0) |
                                  SERCOM_USART_CTRLA_SAMPR(0x0)|
                                  SERCOM_USART_CTRLA_RUNSTDBY  |
                                  SERCOM_USART_CTRLA_MODE_USART_INT_CLK ;
       /* baud register value corresponds to the device communication baud rate */
       SERCOM3->USART.BAUD.reg = baud_value;
       /* 8-bits size is selected as character size by setting the bit CHSIZE as 0,
       TXEN bit and RXEN bits are set to enable the Transmitter and receiver*/
       SERCOM3->USART.CTRLB.reg = SERCOM_USART_CTRLB_CHSIZE(0x0) |
                                  SERCOM_USART_CTRLB_TXEN        |
                                  SERCOM_USART_CTRLB_RXEN ;
       /* synchronization busy */
       while(SERCOM3->USART.SYNCBUSY.bit.CTRLB);
       /* SERCOM3 handler enabled */
       system_interrupt_enable(SERCOM3_IRQn);
       /* receive complete interrupt set */
       SERCOM3->USART.INTENSET.reg = SERCOM_USART_INTFLAG_RXC;
       /* SERCOM3 peripheral enabled */
       SERCOM3->USART.CTRLA.reg |= SERCOM_USART_CTRLA_ENABLE;
       /* synchronization busy */
```

```
        while(SERCOM3->USART.SYNCBUSY.reg & SERCOM_USART_SYNCBUSY_ENABLE);
}

/* External connector(SERCOM2) UART bus and generic clock initialization */
void ext_usart_clock_init(void)
{
        struct system_gclk_chan_config gclk_chan_conf;
        uint32_t gclk_index = SERCOM2_GCLK_ID_CORE;
        /* Turn on module in PM */
        system_apb_clock_set_mask(SYSTEM_CLOCK_APB_APBC, PM_APBCMASK_SERCOM2);
        /* Turn on Generic clock for USART */
        system_gclk_chan_get_config_defaults(&gclk_chan_conf);
        //Default is generator 0. Other wise need to configure like below
        /* gclk_chan_conf.source_generator = GCLK_GENERATOR_1; */
        system_gclk_chan_set_config(gclk_index, &gclk_chan_conf);
        system_gclk_chan_enable(gclk_index);
}

/* External connector(SERCOM2) pin initialization */
void ext_usart_pin_init(void)
{
        /* PA08 and PA09 set into peripheral function D */
        pin_set_peripheral_function(PINMUX_PA08D_SERCOM2_PAD0);
        pin_set_peripheral_function(PINMUX_PA09D_SERCOM2_PAD1);
}

/* External connector(SERCOM2) UART initialization */
void ext_usart_init(void)
{
        calculate_fractional_baud_value(FRAC_BAUD_RATE,sys-
tem_gclk_chan_get_hz(SERCOM2_GCLK_ID_CORE),
        USART_SAMPLE_NUM);
        /* Fractional baud and Baud value */
        SERCOM2->USART.BAUD.reg = SERCOM_USART_BAUD_FRAC_BAUD(baud) |
SERCOM_USART_BAUD_FRAC_FP(fp);
        /* By setting the DORD bit LSB is transmitted first and setting the RXPO bit as 1 cor-
responding SERCOM PAD[1] will be used for data reception RXD, PAD[0] will be used as TxD pin
by setting TXPO bit as 0,16x over-sampling is selected by setting the SAMPR bit as 0,
Generic clock is enabled in all sleep modes by setting RUNSTDBY bit as 1,
USART clock mode is selected as USART with internal clock by setting MODE bit into 1.
        */
        SERCOM2->USART.CTRLA.reg = SERCOM_USART_CTRLA_DORD      |
                                   SERCOM_USART_CTRLA_RXPO(0x1) |
                                   SERCOM_USART_CTRLA_TXPO(0x0) |
                                   SERCOM_USART_CTRLA_SAMPR(0x1)|
                                   SERCOM_USART_CTRLA_RUNSTDBY  |
                                   SERCOM_USART_CTRLA_MODE_USART_INT_CLK ;

        /* 8-bits size is selected as character size by setting the bit CHSIZE as 0,
        TXEN bit and RXEN bits are set to enable the Transmitter and receiver*/
        SERCOM2->USART.CTRLB.reg = SERCOM_USART_CTRLB_CHSIZE(0x0) |
                                   SERCOM_USART_CTRLB_TXEN        |
                                   SERCOM_USART_CTRLB_RXEN ;
        /* synchronization busy */
        while(SERCOM2->USART.SYNCBUSY.bit.CTRLB);
        /* SERCOM2 handler enabled */
        system_interrupt_enable(SERCOM2_IRQn);
        /* receive complete interrupt set */
        SERCOM2->USART.INTENSET.reg = SERCOM_USART_INTFLAG_RXC;
```

```
        /* SERCOM2 peripheral enabled */
        SERCOM2->USART.CTRLA.reg |= SERCOM_USART_CTRLA_ENABLE;
        /* synchronization busy */
        while(SERCOM2->USART.SYNCBUSY.reg & SERCOM_USART_SYNCBUSY_ENABLE);
}

int main (void)
{
        system_init();

        edbg_usart_clock_init();

        edbg_usart_pin_init();

        edbg_usart_init();

        ext_usart_clock_init();

        ext_usart_pin_init();

        ext_usart_init();

        while(1);

}
```

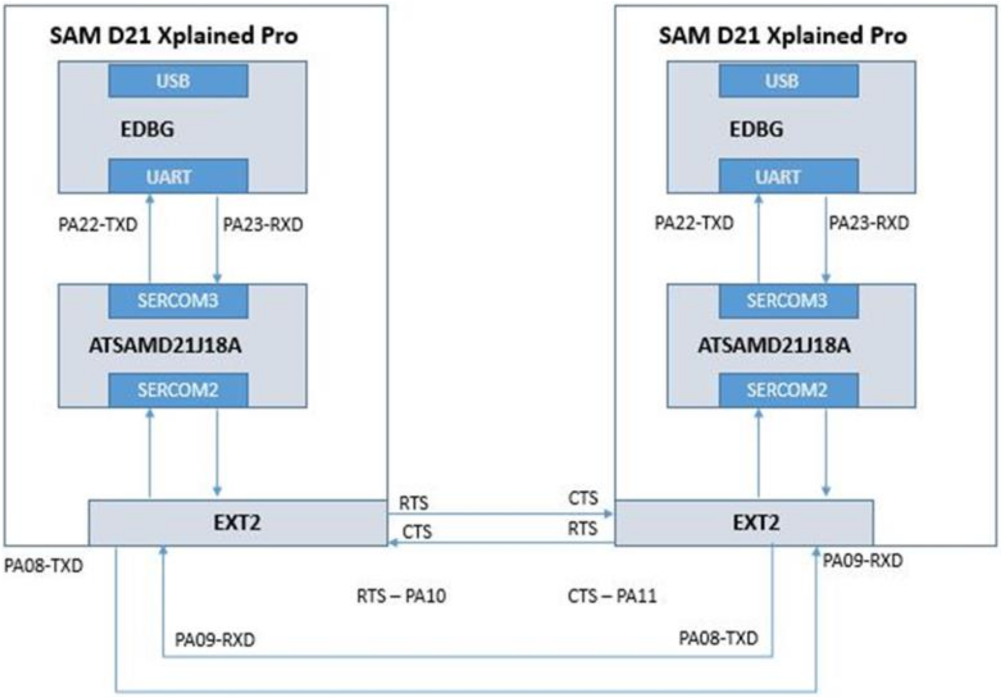
The PC terminal output will look like below.



4.4 Hardware Handshaking Configuration

The USART features an out-of-band hardware handshaking flow control mechanism, implemented by connecting the RTS and CTS pins with the remote device, as shown in Figure 4-3.

Figure 4-3. Block Diagram



In this application, only EXT2 SERCOM2 will be used. In this demonstration the defined data 0xAA will be sent by both the SAM D21 Xplained Pro boards with handshaking protocol.

This section is similar to the basic configuration but with additional configuration of the hardware flow control signal lines and a few changes in the code execution. Only changes will be explained in this section. In the EXT2 SERCOM line, the hardware flow control signal lines need to be configured as below.

```
/* By setting the DORD bit LSB is transmitted first and setting the RXPO bit as 1 correspond-
ing SERCOM PAD[1] will be used for data reception RXD, PAD[0] will be used as TxD pin, PAD[2]
as RTS pin,PAD[3] as CTS signal pin by setting TXPO bit as 2,16x over-sampling is selected by
setting the SAMPR bit as 0,Generic clock is enabled in all sleep modes by setting RUNSTDBY bit
as 1,USART clock mode is selected as USART with internal clock by setting MODE bit into 1.
*/
SERCOM2->USART.CTRLA.reg = SERCOM_USART_CTRLA_DORD        |
                           SERCOM_USART_CTRLA_RXPO(0x1)  |
                           SERCOM_USART_CTRLA_TXPO(0x2)  |
                           SERCOM_USART_CTRLA_SAMPR(0x0)|
                           SERCOM_USART_CTRLA_RUNSTDBY   |
                           SERCOM_USART_CTRLA_MODE_USART_INT_CLK ;
```

Note: The RTS signal of board 1 should be connected to the CTS line of board 2, the CTS line of board 1 should be connected to the RTS line of board 2.

- The DRE bit – Data register empty flag is set in the INTENSET register.

```
SERCOM2->USART.INTENSET.reg = SERCOM_USART_INTENSET_DRE;
```

4.4.2 SERCOM Handler

```
void SERCOM2_Handler()
{
        if (SERCOM2->USART.INTFLAG.bit.DRE) {
                SERCOM2->USART.DATA.reg = ext_tx_data;
        }

}
```

DRE interrupt will be set only when RTS and CTS line is low. During the start of application, RTS and CTS line will be low so the control will service the interrupt handler SERCOM2_Handler.

In the handler, DRE flag is checked for set condition and the defined data 0xAA will be written into the DATA register. Now both the boards will send the 0xAA data. Now hardware control signal lines will be in High condition and control reaches the main function.

```
int main (void)
{
	system_init();

	delay_init();

	ext_usart_clock_init();

	ext_usart_pin_init();

	ext_usart_init();

	while(1)
	{
		delay_ms(1);
		if (SERCOM2->USART.INTFLAG.bit.RXC)
			ext_rx_data = SERCOM2->USART.DATA.reg;
	}

}
```

In the main function, it will check for the RXC flag condition and the received data will be written into variable `ext_rx_data`. Again the SERCOM2 handler will be serviced when the flow control signal reaches low state.

Note: This Application functionality can be checked only by capturing the UART lines in the oscilloscope.

The final application “Hardware Handshaking Configuration” will look like below.

```
#include <asf.h>
#define USART_BAUD_RATE 115200
#define USART_SAMPLE_NUM 16
#define SHIFT 32


uint8_t ext_rx_data, ext_tx_data = 0xAA;

/* Function Prototype */
void ext_usart_clock_init(void);
void ext_usart_pin_init(void);
void ext_usart_init(void);
uint16_t calculate_baud_value(const uint32_t baudrate,const uint32_t peripheral_clock,
                                                                   uint8_t sample_num);

/*ext_usart handler*/
void SERCOM2_Handler()
{
	if (SERCOM2->USART.INTFLAG.bit.DRE) {
		SERCOM2->USART.DATA.reg = ext_tx_data;
	}

}

/*Assigning pin to the alternate peripheral function*/
static inline void pin_set_peripheral_function(uint32_t pinmux)
{
```

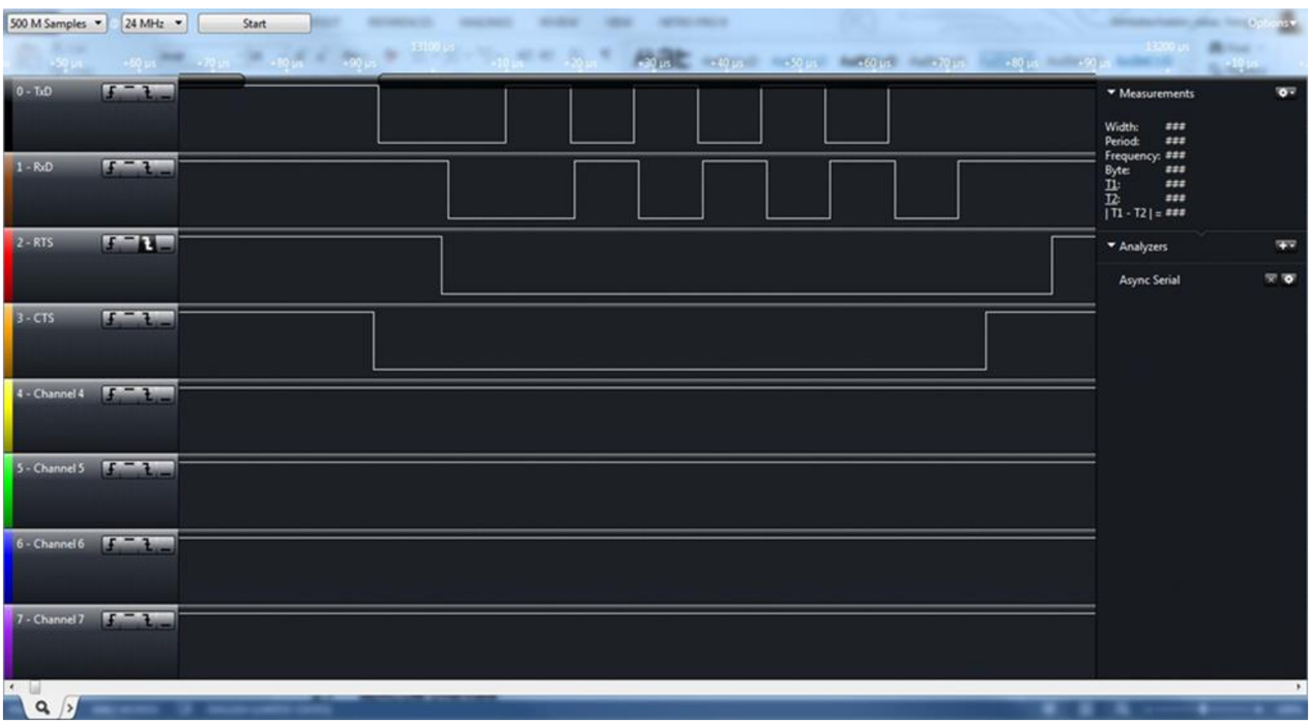
```
	uint8_t port = (uint8_t)((pinmux >> 16)/32);
	PORT->Group[port].PINCFG[((pinmux >> 16) - (port*32))].bit.PMUXEN = 1;
	PORT->Group[port].PMUX[((pinmux >> 16) - (port*32))/2].reg &= ~(0xF << (4 * ((pinmux >>
16) & 0x01u)));
	PORT->Group[port].PMUX[((pinmux >> 16) - (port*32))/2].reg |= (uint8_t)((pinmux &
0x0000FFFF) << (4 * ((pinmux >> 16) & 0x01u)));
}

/*
 * internal Calculate 64 bit division, ref can be found in
 * http://en.wikipedia.org/wiki/Division_algorithm#Long_division
 */
static uint64_t long_division(uint64_t n, uint64_t d)
{
	int32_t i;
	uint64_t q = 0, r = 0, bit_shift;
	for (i = 63; i >= 0; i--) {
		bit_shift = (uint64_t)1 << i;

		r = r << 1;

		if (n & bit_shift) {
			r |= 0x01;
		}

		if (r >= d) {
			r = r - d;
			q |= bit_shift;
		}
	}

	return q;
}

/*
 * internal Calculate asynchronous baudrate value (UART)
*/
uint16_t calculate_baud_value(
		const uint32_t baudrate,
		const uint32_t peripheral_clock,
		uint8_t sample_num)
{
	/* Temporary variables  */
	uint64_t ratio = 0;
	uint64_t scale = 0;
	uint64_t baud_calculated = 0;
	uint64_t temp1;


		/* Calculate the BAUD value */
		temp1 = ((sample_num * (uint64_t)baudrate) << SHIFT);
		ratio = long_division(temp1, peripheral_clock);
		scale = ((uint64_t)1 << SHIFT) - ratio;
		baud_calculated = (65536 * scale) >> SHIFT;

	return baud_calculated;
}
```

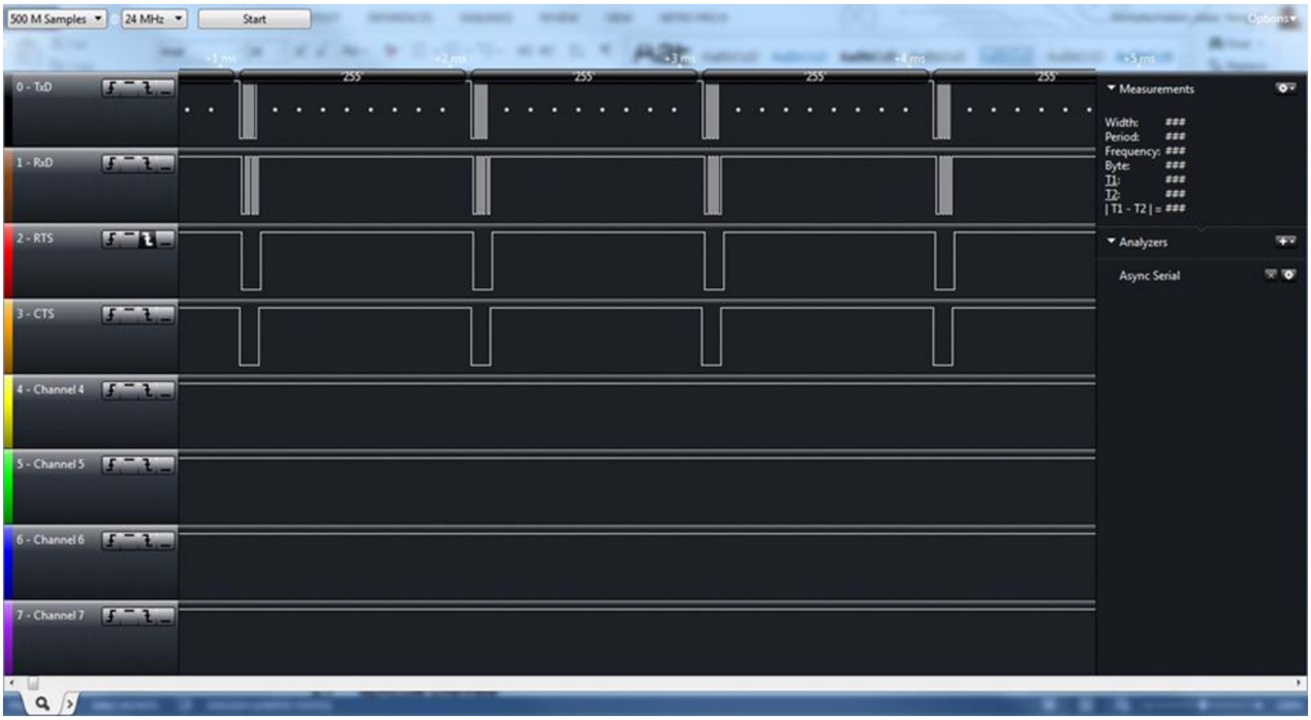
```c
/* External connector(SERCOM2) UART bus and generic clock initialization */
void ext_usart_clock_init(void)
{

       struct system_gclk_chan_config gclk_chan_conf;
       uint32_t gclk_index = SERCOM2_GCLK_ID_CORE;
       /* Turn on module in PM */
       system_apb_clock_set_mask(SYSTEM_CLOCK_APB_APBC, PM_APBCMASK_SERCOM2);
       /* Turn on Generic clock for USART */
       system_gclk_chan_get_config_defaults(&gclk_chan_conf);
       //Default is generator 0. Other wise need to configure like below
       gclk_chan_conf.source_generator = GCLK_GENERATOR_1;
       system_gclk_chan_set_config(gclk_index, &gclk_chan_conf);
       system_gclk_chan_enable(gclk_index);

}
/* External connector(SERCOM2) pin initialization */
void ext_usart_pin_init(void)
{
       /* PA08,PA09,PA10,PA11 set into peripheral function*/
       pin_set_peripheral_function(PINMUX_PA08D_SERCOM2_PAD0);  //TXD
       pin_set_peripheral_function(PINMUX_PA09D_SERCOM2_PAD1);  //RXD
       pin_set_peripheral_function(PINMUX_PA10D_SERCOM2_PAD2);  //RTS
       pin_set_peripheral_function(PINMUX_PA11D_SERCOM2_PAD3);  //CTS
}
/* External connector(SERCOM2) UART initialization */
void ext_usart_init(void)
{
       uint16_t baud_value;
       baud_value = calculate_baud_value(USART_BAUD_RATE,sys-
tem_gclk_chan_get_hz(SERCOM2_GCLK_ID_CORE),
       USART_SAMPLE_NUM);
              /* By setting the DORD bit LSB is transmitted first and setting the RXPO bit as
1 corresponding SERCOM PAD[1] will be used for data reception RXD, PAD[0] will be used as TxD
pin, PAD[2] as RTS pin,PAD[3] as CTS signal pin by setting TXPO bit as 2,
16x over-sampling is selected by setting the SAMPR bit as 0,
Generic clock is enabled in all sleep modes by setting RUNSTDBY bit as 1,
USART clock mode is selected as USART with internal clock by setting MODE bit into 1.
              */
       SERCOM2->USART.CTRLA.reg = SERCOM_USART_CTRLA_DORD         |
                                  SERCOM_USART_CTRLA_RXPO(0x1) |
                                  SERCOM_USART_CTRLA_TXPO(0x2) |
                                  SERCOM_USART_CTRLA_SAMPR(0x0)|
                                  SERCOM_USART_CTRLA_RUNSTDBY  |
                                  SERCOM_USART_CTRLA_MODE_USART_INT_CLK ;
       /* baud register value corresponds to the device communication baud rate */
       SERCOM2->USART.BAUD.reg = baud_value;
       /* 8-bits size is selected as character size by setting the bit CHSIZE as 0,
       TXEN bit and RXEN bits are set to enable the Transmitter and receiver*/
       SERCOM2->USART.CTRLB.reg = SERCOM_USART_CTRLB_CHSIZE(0x0) |
                                   SERCOM_USART_CTRLB_TXEN        |
                                   SERCOM_USART_CTRLB_RXEN ;
       /* synchronization busy */
       while(SERCOM2->USART.SYNCBUSY.bit.CTRLB);
       //SERCOM2->USART.INTENSET.bit.RXC = 1;
       /* SERCOM2 handler enabled */
       system_interrupt_enable(SERCOM2_IRQn);
       SERCOM2->USART.INTENSET.reg = SERCOM_USART_INTENSET_DRE;
       SERCOM2->USART.CTRLA.reg |= SERCOM_USART_CTRLA_ENABLE;
```

```c
}
int main (void)
{
	system_init();

	delay_init();

	ext_usart_clock_init();

	ext_usart_pin_init();

	ext_usart_init();

	while(1)
	{
		/* Delay of 1 ms is added to emulate that the CPU is busy doing something */
		delay_ms(1);
		if (SERCOM2->USART.INTFLAG.bit.RXC)
			ext_rx_data = SERCOM2->USART.DATA.reg;
	}

}
```

Figure 4-4 shows the output of the SEROM USART hardware handshaking application.

Figure 4-4. SERCOM USART – Hardware Handshaking Output



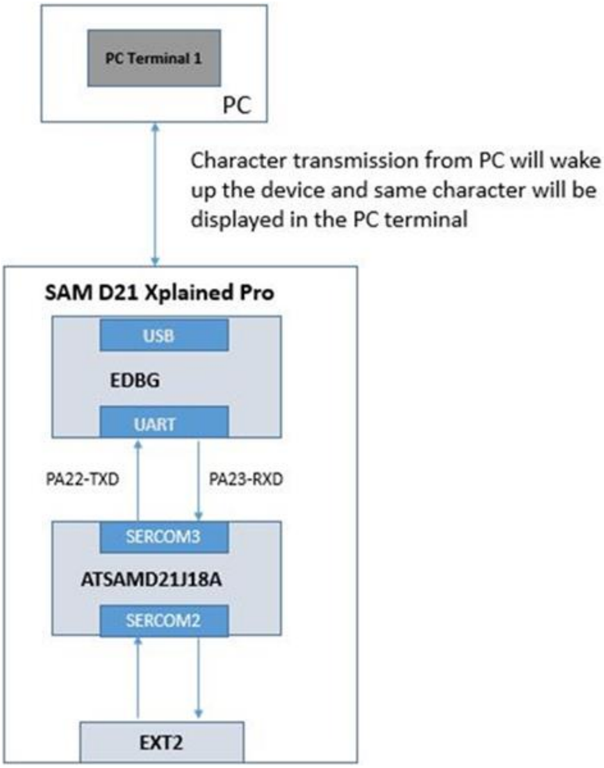


4.5 SOF Detection and Wakeup Configuration

The USART start-of-frame 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 into 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.

Figure 4-5. Block Diagram



This application uses only one SAM D21 Xplained Pro connected to the PC terminal through EDBG. Here only SERCOM3 is used.

Basic configuration section will be used here with few changes for SOF detection and wakeup configuration.

The below configurations are the changes from Basic configuration application.

```
/* 8-bit, Even parity one stop bit 8 bits, Transmitter, receiver enabled
       Start of frame detection enabled */
SERCOM3->USART.CTRLB.reg = SERCOM_USART_CTRLB_CHSIZE(0x0) |
                            SERCOM_USART_CTRLB_TXEN       |
                            SERCOM_USART_CTRLB_RXEN       |
                            SERCOM_USART_CTRLB_SFDE;
```

- By setting the SFDE bit – Start of Frame Detection Enable, device will wake up from the sleep once detecting the start bit on the RxD line.

Below are the wake up settings for SFDE.

Table 4-1. Start of Frame Detection

SFDE	INTENSET.RXS	INTENSET.RXC	Description
0	X	X	Start-of-frame detection disabled
1	0	0	Reserved
1	0	1	Start-of-frame detection enabled. RXC wakes up the device from all sleep modes.
1	1	0	Start-of-frame detection enabled. RXS wakes up the device from all sleep modes.
1	1	1	Start-of-frame detection enabled. Both RXC and RXS wake up the device from all sleep modes.

- The `system_set_sleepmode (SYSTEM_SLEEPMODE_STANDBY)` function will set the device sleep mode as standby sleep mode
- The STANDBY sleep mode allows very low power consumption
- In standby sleep mode all the clocks will be stopped except those kept running if requested by a running module with ONDEMAND bit set
- If the `CONF_CLOCK_OSC8M_ON_DEMAND` bit is true in the conf_clocks.h file, then only on demand the clock will be supplied. It takes the clock startup delay to supply the clock.
- In the SOF detection application, `CONF_CLOCK_OSC8M_ON_DEMAND` bit will be set as false to avoid the loss of first character which wake up the device from sleep mode
- The application enables both RXC and RXS interrupt
- RXS flag is set when a start condition is detected on the RxD line and start-of-frame detection is enabled
- The application uses only SERCOM3 handler – EDBG USART

Boolean type variable `rx_started` is used to flag the interrupt. Initial value is configured as false.

Main function will be like below.

```
int main (void)
{
	system_init();

	edbg_usart_clock_init();

	edbg_usart_pin_init();

	edbg_usart_init();

	system_set_sleepmode(SYSTEM_SLEEPMODE_STANDBY);

	while(1)
	{
		if (!rx_started)
		{
			usart_send_string("\r\n Device entered into standby sleep mode");
			while(!SERCOM3->USART.INTFLAG.bit.TXC);
			system_sleep();
		}

		while(rx_started);

		usart_send_string("\r\n Character received after wakeup : ");
		while(!SERCOM3->USART.INTFLAG.bit.DRE);
		SERCOM3->USART.DATA.reg = edbg_rx_data;
	}

}
```

- Once setting the sleep mode as standby sleep mode, in the while loop Boolean variable is checked for the false condition, then the function `usart_send_string` is used to print the message in the terminal window
- The `system_sleep` function will make the device to enter into the sleep mode defined
- Once pressing a character on the PC, the device wake up from sleep mode and enters into the `SERCOM3_Handler` function

```
void SERCOM3_Handler()
```

```
{
        if (SERCOM3->USART.INTFLAG.bit.RXS){
              SERCOM3->USART.INTFLAG.bit.RXS = 1;
              rx_started = true;

        }
        if (SERCOM3->USART.INTFLAG.bit.RXC){
              edbg_rx_data = SERCOM3->USART.DATA.reg;
              rx_started = false;
        }

}
```

- In the handler, the interrupt flag condition RXS – Receive start will be checked. This flag will be set when a start condition is detected on the RxD line and start of frame detection is enabled (CTRLB.SFDE is one).
- If the RXS flag is set, the Boolean variable `rx_started` is set to true
- Now the RXC – receive complete flag is checked for set condition and the character pressed is read from DATA register into the buffer `edbg_rx_data` and Boolean variable `rx_started` is set to false.
- The control reaches the main function and character press value from the buffer `rx_started` is written into the DATA register. Now this value will be displayed in the PC terminal.

The final application "SOF detection and wakeup configuration" in main.c file will be as below.

```
#include <asf.h>
#define USART_BAUD_RATE 9600
#define USART_SAMPLE_NUM 16
#define SHIFT 32


uint8_t edbg_rx_data;
volatile bool rx_started = false;

/* Function prototype */
void edbg_usart_clock_init(void);
void edbg_usart_pin_init(void);
void edbg_usart_init(void);
void usart_send_string(const char *str_buf);

uint16_t calculate_baud_value(const uint32_t baudrate,const uint32_t peripheral_clock,
                                                                 uint8_t sample_num);

/*edbg_usart handler*/
void SERCOM3_Handler()
{

        if (SERCOM3->USART.INTFLAG.bit.RXS){
              SERCOM3->USART.INTFLAG.bit.RXS = 1;
              rx_started = true;
        }
        if (SERCOM3->USART.INTFLAG.bit.RXC){
              edbg_rx_data = SERCOM3->USART.DATA.reg;
              rx_started = false;
        }

}

/*Assigning pin to the alternate peripheral function*/
static inline void pin_set_peripheral_function(uint32_t pinmux)
```

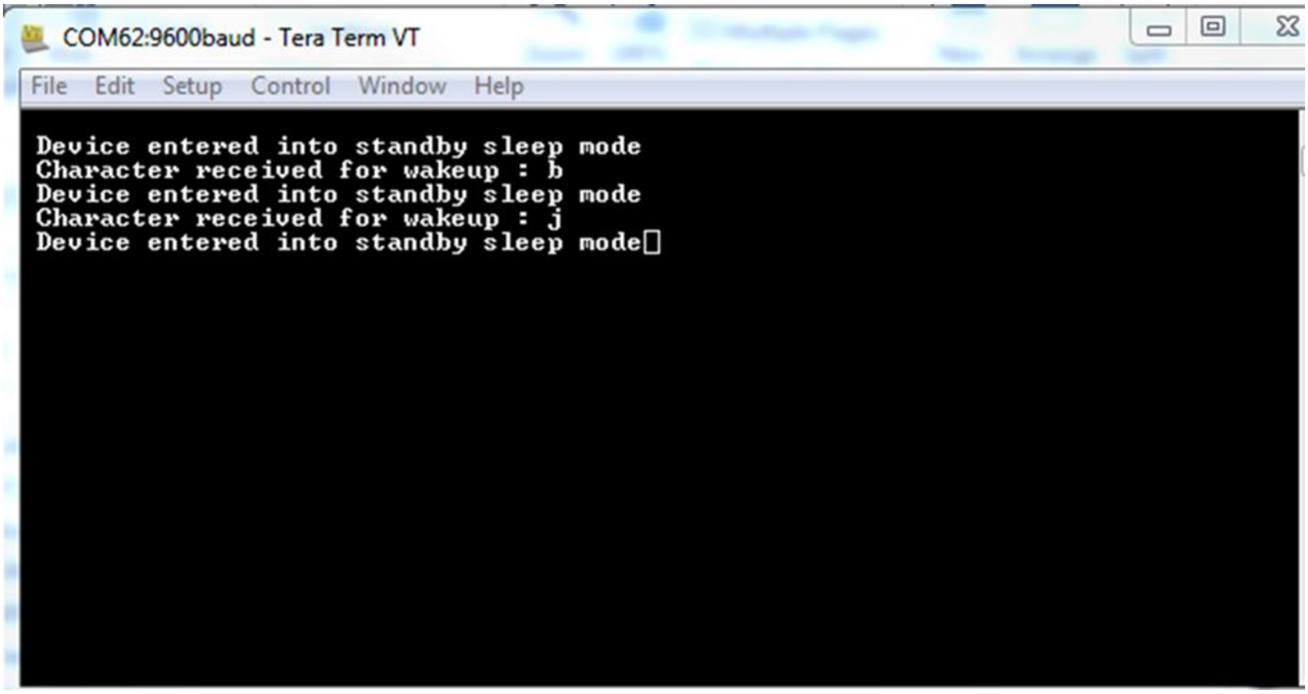
```
{
        uint8_t port = (uint8_t)((pinmux >> 16)/32);
        PORT->Group[port].PINCFG[((pinmux >> 16) - (port*32))].bit.PMUXEN = 1;
        PORT->Group[port].PMUX[((pinmux >> 16) - (port*32))/2].reg &= ~(0xF << (4 * ((pinmux >>
16) & 0x01u)));
        PORT->Group[port].PMUX[((pinmux >> 16) - (port*32))/2].reg |= (uint8_t)((pinmux &
0x0000FFFF) << (4 * ((pinmux >> 16) & 0x01u)));
}

/*
 * internal Calculate 64 bit division, ref can be found in
 * http://en.wikipedia.org/wiki/Division_algorithm#Long_division
 */
static uint64_t long_division(uint64_t n, uint64_t d)
{
        int32_t i;
        uint64_t q = 0, r = 0, bit_shift;
        for (i = 63; i >= 0; i--) {
                bit_shift = (uint64_t)1 << i;

                r = r << 1;

                if (n & bit_shift) {
                        r |= 0x01;
                }

                if (r >= d) {
                        r = r - d;
                        q |= bit_shift;
                }
        }

        return q;
}

/*
 * internal Calculate asynchronous baudrate value (UART)
*/
uint16_t calculate_baud_value(
                const uint32_t baudrate,
                const uint32_t peripheral_clock,
                uint8_t sample_num)
{
        /* Temporary variables  */
        uint64_t ratio = 0;
        uint64_t scale = 0;
        uint64_t baud_calculated = 0;
        uint64_t temp1;


                /* Calculate the BAUD value */
                temp1 = ((sample_num * (uint64_t)baudrate) << SHIFT);
                ratio = long_division(temp1, peripheral_clock);
                scale = ((uint64_t)1 << SHIFT) - ratio;
                baud_calculated = (65536 * scale) >> SHIFT;

        return baud_calculated;
}
```

```
/* EDBG UART(SERCOM3) bus and generic clock initialization */
void edbg_usart_clock_init(void)
{

              struct system_gclk_chan_config gclk_chan_conf;
              uint32_t gclk_index = SERCOM3_GCLK_ID_CORE;
              /* Turn on module in PM */
              system_apb_clock_set_mask(SYSTEM_CLOCK_APB_APBC, PM_APBCMASK_SERCOM3);
              /* Turn on Generic clock for USART */
              system_gclk_chan_get_config_defaults(&gclk_chan_conf);
              //Default is generator 0. Other wise need to configure like below
               /* gclk_chan_conf.source_generator = GCLK_GENERATOR_1; */
              system_gclk_chan_set_config(gclk_index, &gclk_chan_conf);
              system_gclk_chan_enable(gclk_index);

}

/* EDBG UART(SERCOM3) pin initialization */
void edbg_usart_pin_init(void)
{
       /* PA22 and PA23 set into peripheral function C*/
       pin_set_peripheral_function(PINMUX_PA22C_SERCOM3_PAD0);
       pin_set_peripheral_function(PINMUX_PA23C_SERCOM3_PAD1);
}

/* EDBG UART(SERCOM3) initialization */
void edbg_usart_init(void)
{
       uint16_t baud_value;
       baud_value = calculate_baud_value(USART_BAUD_RATE,sys-
tem_gclk_chan_get_hz(SERCOM3_GCLK_ID_CORE),
                                                                      USART_SAMPLE_NUM);
       /* By setting the DORD bit LSB is transmitted first and setting the RXPO bit as 1 cor-
responding SERCOM PAD[1] will be used for data reception, PAD[0] will be used as TxD pin by
setting TXPO bit as 0,16x over-sampling is selected by setting the SAMPR bit as 0,
USART clock mode is selected as USART with internal clock by setting MODE bit into 1.
              */
       SERCOM3->USART.CTRLA.reg = SERCOM_USART_CTRLA_DORD      |
                                   SERCOM_USART_CTRLA_RXPO(0x1) |
                                   SERCOM_USART_CTRLA_TXPO(0x0) |
                                   SERCOM_USART_CTRLA_SAMPR(0x0)|
                                   SERCOM_USART_CTRLA_MODE_USART_INT_CLK ;
       /*baud register value corresponds to the device communication baud rate */
       SERCOM3->USART.BAUD.reg = baud_value;

       /* 8-bits size is selected as character size by setting the bit CHSIZE as 0,
       TXEN bit and RXEN bits are set to enable the Transmitter and receiver*/
       SERCOM3->USART.CTRLB.reg = SERCOM_USART_CTRLB_CHSIZE(0x0) |
                                   SERCOM_USART_CTRLB_TXEN        |
                                   SERCOM_USART_CTRLB_RXEN        |
                                   SERCOM_USART_CTRLB_SFDE;

       /* synchronization busy */
       while(SERCOM3->USART.SYNCBUSY.bit.CTRLB);
       /* SERCOM3 handler enabled */
       system_interrupt_enable(SERCOM3_IRQn);
       /* receive complete interrupt,receive start interrupt set */
       SERCOM3->USART.INTENSET.reg = SERCOM_USART_INTFLAG_RXC | SERCOM_USART_INTFLAG_RXS;
       /* SERCOM3 peripheral enabled */
```

```
        SERCOM3->USART.CTRLA.reg |= SERCOM_USART_CTRLA_ENABLE;
        /* synchronization busy */
        while(SERCOM3->USART.SYNCBUSY.reg & SERCOM_USART_SYNCBUSY_ENABLE);


}

/* function used to print the character in the terminal */
void usart_send_string(const char *str_buf)
{
while (*str_buf != '\0')
        {
                while(!SERCOM3->USART.INTFLAG.bit.DRE);
                SERCOM3->USART.DATA.reg = *str_buf;
                str_buf++;
}
}
int main (void)
{
        system_init();

        edbg_usart_clock_init();

        edbg_usart_pin_init();

        edbg_usart_init();

        system_set_sleepmode(SYSTEM_SLEEPMODE_IDLE_2);

        while(1)
        {
                if (!rx_started)
                {
                        usart_send_string("\r\n Device entered into standby sleep mode");
                        while(!SERCOM3->USART.INTFLAG.bit.TXC);
                        system_sleep();
                }

                while(rx_started);

                usart_send_string("\r\n Character received for wakeup : ");
                while(!SERCOM3->USART.INTFLAG.bit.DRE);
                SERCOM3->USART.DATA.reg = edbg_rx_data;
        }

}
```

Figure 4-6 shows the output of the SEROM USART hardware handshaking application.

Figure 4-6. SERCOM USART – SOF Detection and Wake-up Configuration Output



5 References

SAM D21 Device Datasheet - http://www.atmel.com/Images/Atmel-42181-SAM-D21_Datasheet.pdf.

SAM D21 Xplained Pro user guide and schematics link - http://www.atmel.com/tools/atsamd21-xpro.aspx?tab=documents.

6 Revision History

Doc Rev.	Date	Comments
42539A	09/2015	Initial document release.

Atmel | Enabling Unlimited Possibilities®

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | **www.atmel.com**