Microchip Cortex® （SAM 和 PIC32C）器件上的 SERCOM USART

AN5807

产品页链接

简介

本应用笔记介绍了基于 Arm® Cortex®-M 内核的 PIC32C 和 SAM 系列器件（例如 SAM D、SAM E、SAM L、PIC32CK、 PIC32CM、PIC32CX 和 PIC32CZ）中 SERCOM USART 的各种功能，以及其带有示例代码和相应输出的配置。

本文档中探讨的示例使用两个 SAM D21 Curiosity Nano 评估工具包和两个 SAM E70 Xplained Ultra 评估工具包。

注：相同的配置和步骤可以轻松复制到其他 Arm 系列，如 SAM D 单片机中的 SERCOM 实现中所示。

目录

1. 串行通信接口简介

串行通信接口在嵌入式系统中多个单片机和其他器件之间交换数据时发挥着关键作用。数据交换支持半双工和全双工，具体取决于串行模块的规范。串行模块的数据速率和连接各不相同。USART、I^2C、SPI、LIN、LON 是嵌入式系统中常见的串行模块。

1.1. USART

通用同步/异步收发器（Universal Synchronous/Asynchronous Receiver/Transmitter，USART）是嵌入在单片机内用于实现串行数据交换的完整通信模块。这一多功能组件可以管理同步和异步串行协议，按顺序逐位发送数据。

USART 具有高度的可配置性，允许使用多种数据帧结构、波特率和工作模式。它支持全双工操作。这种适应性使它们非常适合与各种设备（包括传感器、计算机和其他单片机）连接。

1.2. I^2C

I^2C 是一种双线协议。I^2C 是一种提供仲裁和冲突检测功能的多主器件总线。该协议采用半双工通信。根据速度模式，可提供不同的传输速率。I^2C 的速度比 USART 快，但比 SPI 慢。I^2C 主要用于嵌入式应用，因为嵌入式应用中可用于通信的引脚数量有限，并且必须在单个总线上连接多个器件。

1.3. SPI

SPI 是一种使用四条物理线进行通信的同步串行总线。它支持全双工操作。SPI 支持更高的数据速率。SPI 可以与单个主器件和一个或多个从器件一起运行，每个器件都有单独的芯片选择线。

1.4. LIN

局域互联网络（Local Interconnect Network，LIN）模式可方便地在 LIN 总线上实现主器件节点与从器件节点之间的连接。LIN 是一种串行通信协议，用于高效管理分布式汽车系统中机电节点的控制。在不需要 CAN 的更高带宽和灵活性的场景下，LIN 为总线通信提供了一种经济高效的解决方案。

1.5. LON

局域操作网络（Local Operating Network，LON）模式可方便地实现与 LON 基础设施的集成。该模式包含全面的 OSI（开放系统互连）模型，涵盖了从物理连接（包括有线、电力线、射频和 Internet 协议）到应用层和中间层的全部七层。LON 模式的设计初衷是一种强大的通信控制平台，支持高效交换物理协议数据单元（Physical Protocol Data Unit，PPDU）帧，只需消耗极少的单片机资源。

2. SAM D 单片机中的 SERCOM 实现

通常，单片机具有单独的串行通信模块，每个模块有不同的引脚分配。每个模块都提供了独立的专用外设和用户寄存器。例如，USART 是一个有专用引脚来实现其功能的独立外设，而 I²C 是一个具有专用引脚的独立外设。

在 SAM D 单片机中，所有串行外设都集成到单个模块中，用作串行通信接口（SERCOM）。SERCOM 模块可由用户选择配置为 USART、I²C 或 SPI。每个 SERCOM 将被分配从 PAD0 到 PAD3 的四个焊盘。每个焊盘的功能可根据所使用的 SERCOM 模式进行配置。未使用的焊盘可用于其他目的，并且 SERCOM 模块不会控制这些焊盘，除非配置它们以供 SERCOM 模块使用。例如，SERCOM0 可配置为 USART 模式，其中 PAD0 作为发送焊盘，PAD1 作为接收焊盘。其他未使用的焊盘（PAD2 和 PAD3）可用作 GPIO 引脚或分配给其他外设。不同焊盘的 SERCOM 功能分配方案高度灵活，这让 SERCOM 模块比典型的串行通信外设实现更具优势。

注：这些配置也可以在以下器件系列中实现：

- SAM L 系列单片机
- SAM C 系列单片机
- SAM D 系列单片机
- SAM D5X/E5X 系列单片机
- PIC32CK 系列单片机
- PIC32CM 系列单片机
- PIC32CX 系列单片机
- PIC32CZ 系列单片机

2.1. 概述

串行通信接口（SERCOM）可配置为支持三种不同的模式：I²C、SPI 和 USART。配置并使能后，所有 SERCOM 资源都专用于所选模式。SERCOM 串行引擎包括发送器和接收器、波特率发生器和地址匹配功能。它可以配置为使用内部通用时钟或外部时钟，从而可在所有休眠模式下运行。

2.2. 特性

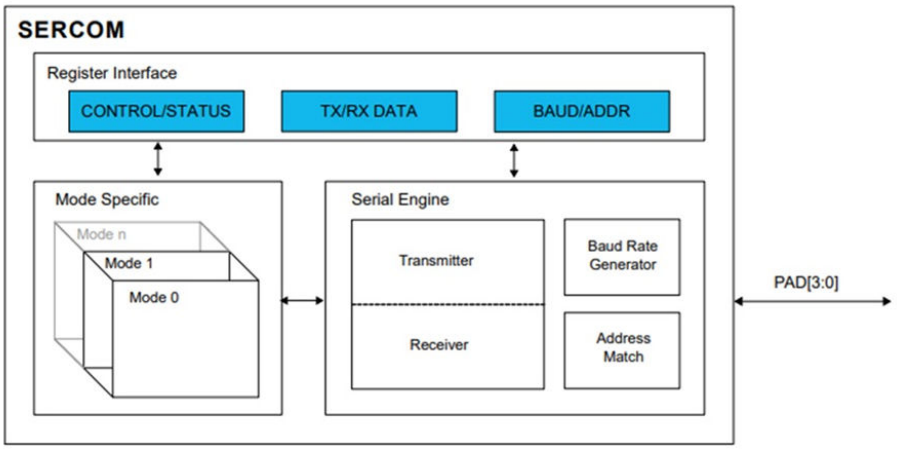
模块的主要特性包括：

- 组合接口可配置为以下接口之一：
  - I²C——双线串行接口（兼容 SMBus）
  - SPI——串行外设接口
  - USART——通用同步/异步收发器
- 单发送缓冲区，双接收缓冲区
- 波特率发生器（Baud Rate Generator，BRG）
- 地址匹配或掩码逻辑
- 可在所有休眠模式下工作
- 可与 DMA 配合使用（SAM D20 MCU 中不受支持）

2.3. 框图

下图给出了 SERCOM 模块的框图。

图 2-1. SERCOM 框图



2.4. 时钟

SERCOM 模块的运行需要以下时钟：

- SERCOM 总线时钟
- SERCOM 内核通用时钟
- SERCOM 慢速通用时钟

默认情况下，可以在功耗管理器（Power Manager，PM）模块中使能和禁止 SERCOM 总线时钟（CLK_SERCOMx_APB）。SERCOM 模块使用两个通用时钟：GCLK_SERCOMx_CORE 和 GCLK_SERCOMx_SLOW。在 SERCOM 作为主器件运行时，需要内核时钟（GCLK_SERCOMx_CORE）为其提供时钟，而仅某些功能（如 I^2C 超时）需要慢速时钟（GCLK_SERCOMx_SLOW）。

3. SAM E70 单片机中的 USART 实现

3.1. 概述

通用同步/异步收发器（USART）提供了一条全面的全双工同步或异步串行连接。其数据帧配置高度可定制，包括数据长度、奇偶校验和停止位选项，确保与各类标准兼容。接收器具有奇偶校验错误、帧错误和溢出错误的检测能力。此外，还包括接收器超时功能以有效管理可变长度帧，同时设置发送器时间保护功能以方便地与低速远程设备通信。

USART 设计了三种诊断测试模式：远程环回、本地环回和自动回显。它支持多种工作模式，可与 RS485、LIN、LON 和 SPI 总线接口。它兼容 ISO7816 T = 0 或 T = 1 智能卡协议、红外收发器和调制解调器端口连接。硬件握手功能利用 RTS 和 CTS 线实现了带外流控制。

此外，USART 兼容直接存储器访问（Direct Memory Access，DMA）控制器连接，可在发送器和接收器之间实现高效的数据传输。DMA 控制器提供高级缓冲区管理功能，无需处理器干预即可运行。

3.2. 特性

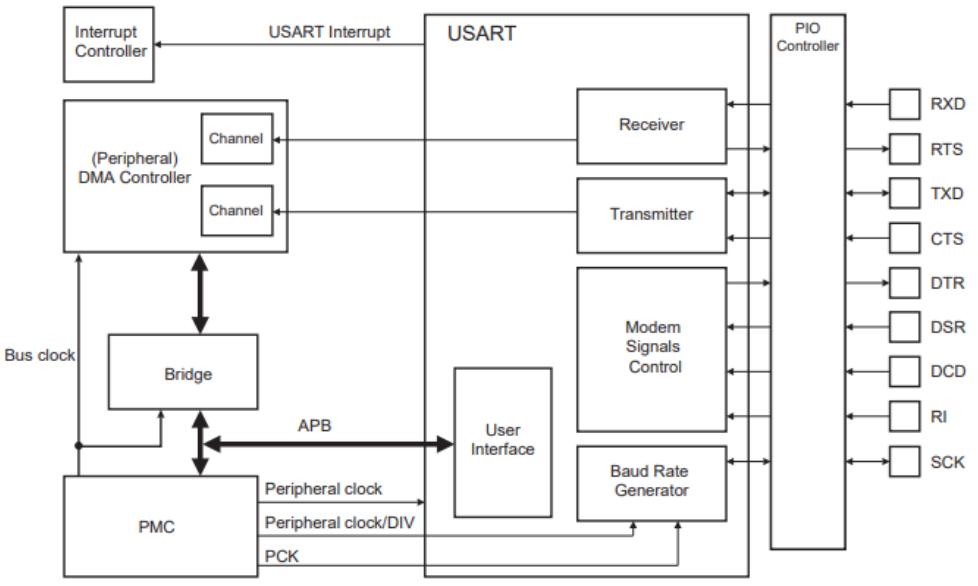
USART 模块的主要特性包括：

- 带驱动器控制信号的 RS485
- SPI 模式
- LIN 模式
- LON 模式
- 寄存器写保护

3.3. 框图

下图给出了 USART 模块的框图：

图 3-1. USART 框图



3.4. 时钟

USART 的时钟不是连续提供的；相反，它提供从各种来源选择内部时钟的选项，具体来源包括慢速时钟（SLCK）、主时钟（MAINCK）、任何可用的锁相环（PLL）时钟和主器件时钟（MCK）。通过配置可编程时钟（PMC）控制状态寄存器（PMC_PCKx）的时钟源选择（CSS）字段，可以选择具体来源。

此外，可以通过设置 PMC_PCKx 预分频器（PRES）来调整这些时钟源的频率。可以通过向相应的电源管理控制器系统时钟启用寄存器（PMC_SCER.PCKx）或系统时钟禁止寄存器（PMC_SCDR.PCKx）写入“1”来激活或禁用每个可编程时钟输出（PCKx）。这些内部时钟的运行状态反映在 PMC 系统时钟状态寄存器（PMC_SCSR.PCKx）中。在可编程时钟寄存器中设置后，PMC 状态寄存器（PMC_SR.PCKRDYx）标志可用于指示可编程内部时钟是否准备就绪。对于 USART 操作，指定 PCK4。

电源管理控制器（Power Management Controller，PMC）通过 PMC 外设控制寄存器（PMC_PCR）管理嵌入式外设的时钟。该寄存器使用户能够管理各种外设时钟的激活和禁止，包括分配给每个外设并从主器件时钟（MCK）派生的外设时钟（periph_clk [PID]），以及专门分配给 I2SC0 和 I2SC1 的通用时钟（GCLK[PID]）。这些时钟独立于内核和总线时钟（HCLK、MCK 和 periph_clk [PID]）运行，并通过时钟源（如 SLCK、MAINCK、UPLLCKDIV、PLLACK 和 MCK）选择和分频生成。要配置外设的时钟，必须将 PMC_PCR.CMD 设置为 1，并且必须为 PMC_PCR.PID 分配目标外设的索引。必须准确定义所有其他配置字段才能正确设置。

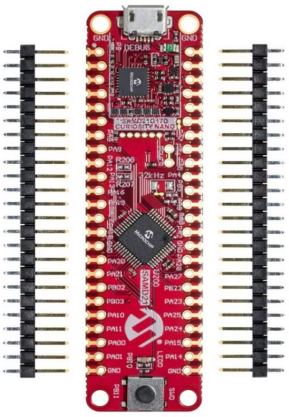
4. 硬件和软件要求

该应用演示需要两个 SAM D21 Curiosity Nano 评估工具包和两个 SAM E70 Xplained Ultra 评估工具包。一个工具包将作为主器件/发送器，另一个工具包将作为从器件/接收器。

本应用程序使用以下软件和硬件工具：

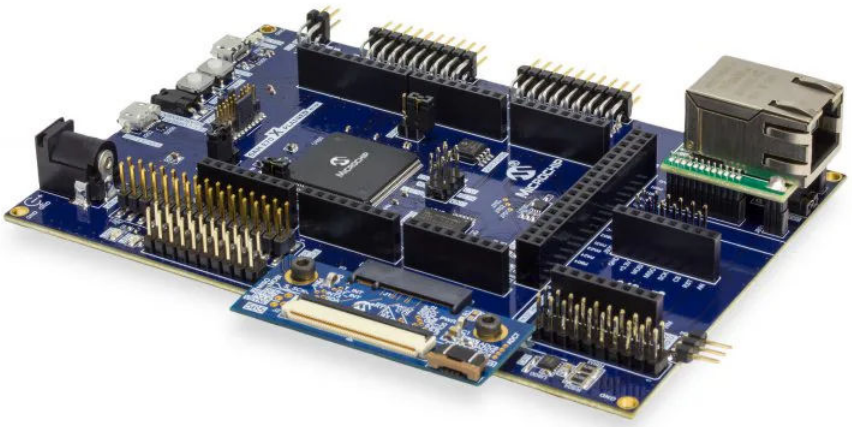
- MPLAB® X IDE v6.20
- MPLAB 代码配置器插件 v5.5.1
- MPLAB XC32 编译器 v4.45
- csp v3.20.0
- SAM D21 Curiosity Nano 评估工具包
- SAM E70 Xplained Ultra 评估工具包

图 4-1. SAM D21 Curiosity Nano 评估工具包



SAM D21 Curiosity Nano 评估工具包配备 USB 端口 DEBUG USB。要使用嵌入式调试器 EDBG 进行调试，必须连接 DEBUG USB 端口。

图 4-2. SAM E70 Xplained Ultra 评估工具包

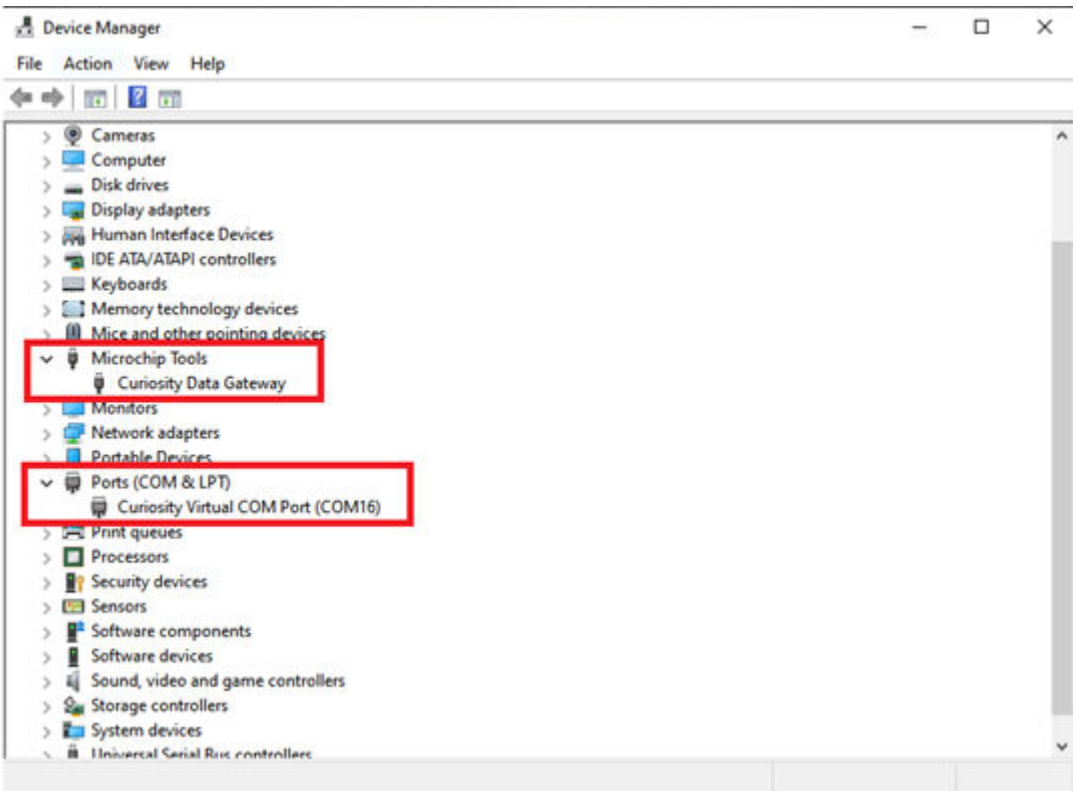


SAM E70 Xplained Ultra 评估工具包具有两个 USB 端口：DEBUG USB 和 TARGET USB。要使用嵌入式调试器（EDBG）进行调试，必须连接 DEBUG USB 端口。

如果驱动程序安装成功，设备管理器中将列出 EDBG，如下图所示。

注：如果上面列出的工具有更新版本，也可用于创建应用程序。

图 4-3. EDBG 驱动程序安装成功



注：在计算机的搜索栏中搜索设备管理器，以验证 Microchip 工具和端口是否被识别。

5. 应用演示

本章通过不同的示例代码演示了 SAM D21 Curiosity Nano 评估工具包和 SAM E70 Xplained Ultra 评估工具包的 SERCOM USART 模块的各种功能。

后续章节中将实现以下配置：

- 基本配置
- 小数波特率配置
- 硬件握手配置
- SOF 检测和唤醒配置（SAM E70 器件中未提供）

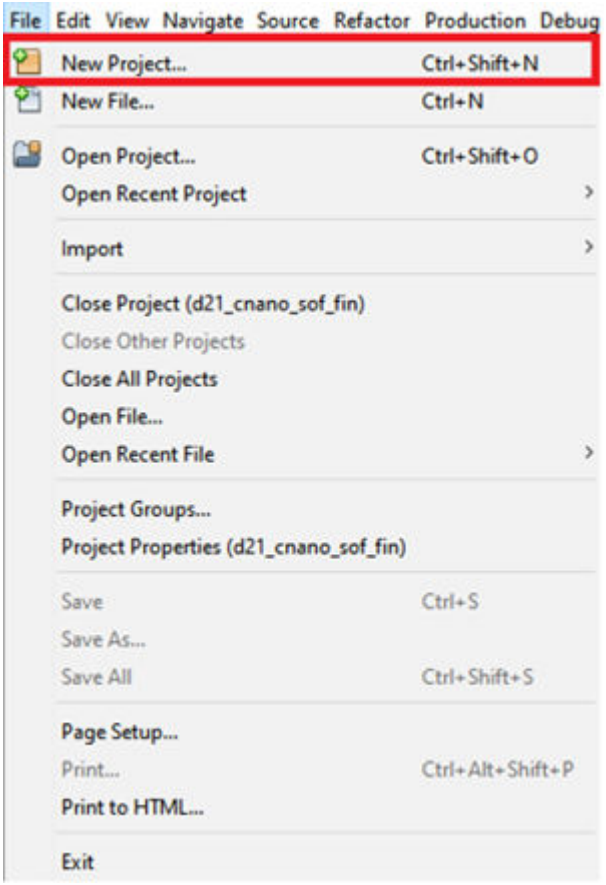
5.1. 创建项目

实现这些功能需要两个项目：一个用于主器件/发送器，另一个用于从器件/接收器。

要创建基于 MPLAB Harmony v3 的项目，请按照以下步骤操作：

1. 从 **Start**（开始）菜单中启动 MPLAB X IDE。
2. 单击 **File**（文件）菜单，选择 **New Project**（新建项目）或单击 *New Project* 图标。

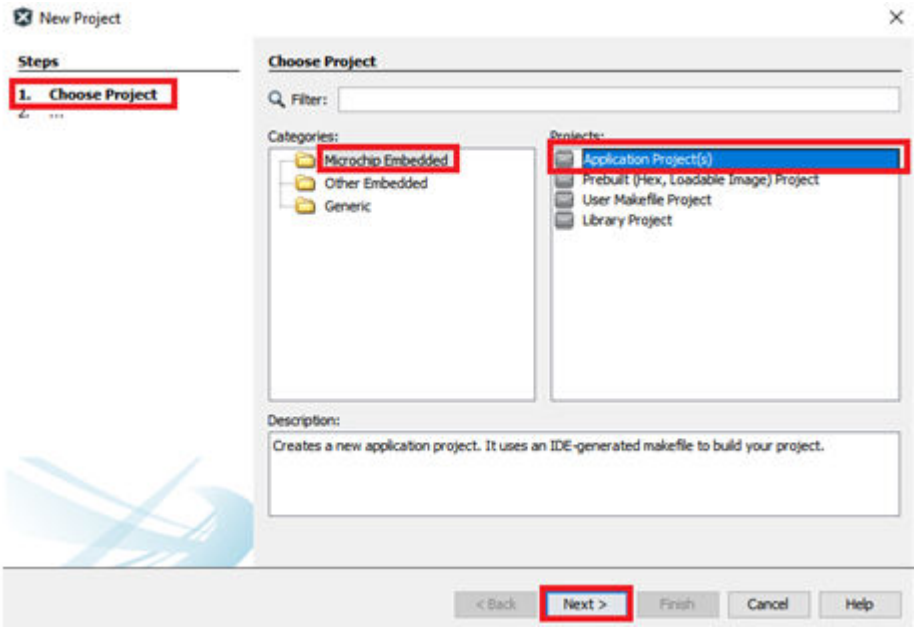
图 5-1. MPLAB X IDE v6.20 中的新项目



3. 在 **New Project** 窗口左侧导航栏中的 Steps（步骤）下单击 **Choose Project**（选择项目）。
4. 在 **Choose Project** 属性页面中：
   a. 在 Categories（类别）下，选择 **Microchip Embedded**（Microchip 已安装工具）。

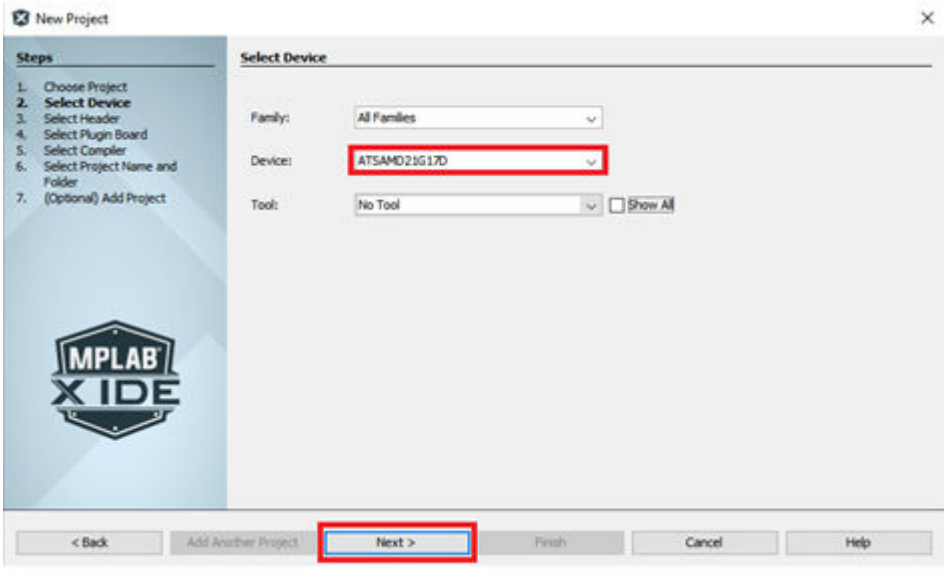
b. 在 Projects（项目）下，选择 **Application Project(s)**（应用程序项目）。

图 5-2. 选择项目



5. 单击 **Next**（下一步）。
6. 在左侧导航栏中，单击 **Select Device**（选择器件）。
7. 在 **Select Device** 属性页的 **Device**（器件）框中，为 SAM D21 Curiosity Nano 评估工具包输入或选择器件 **ATSAMD21G17D**。
   注：对于 SAM E70 Xplained Ultra 评估工具包，选择器件 ATSAME70Q21B。

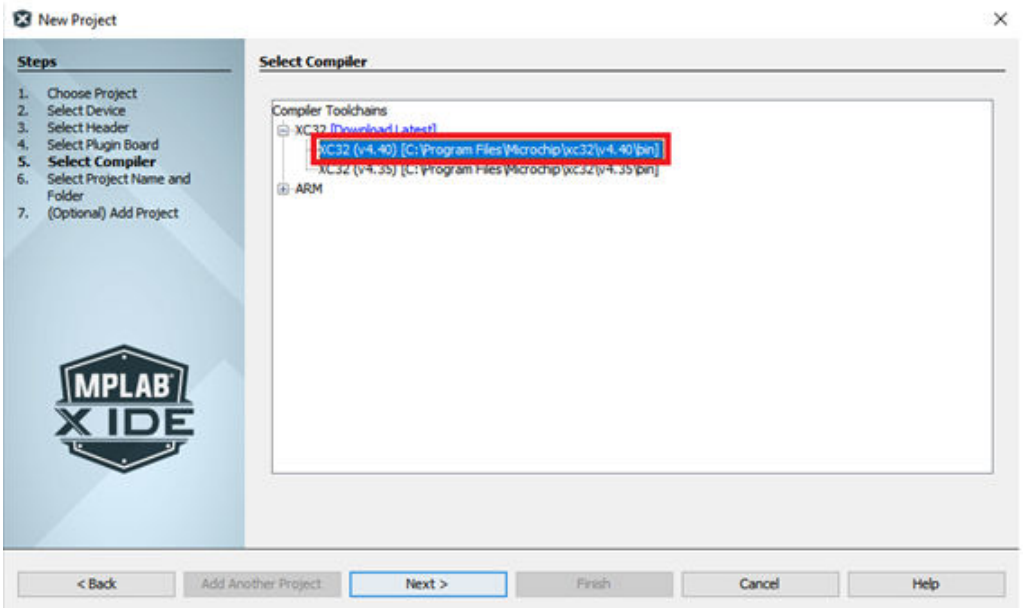
图 5-3. 器件选择



8. 单击 **Next**。

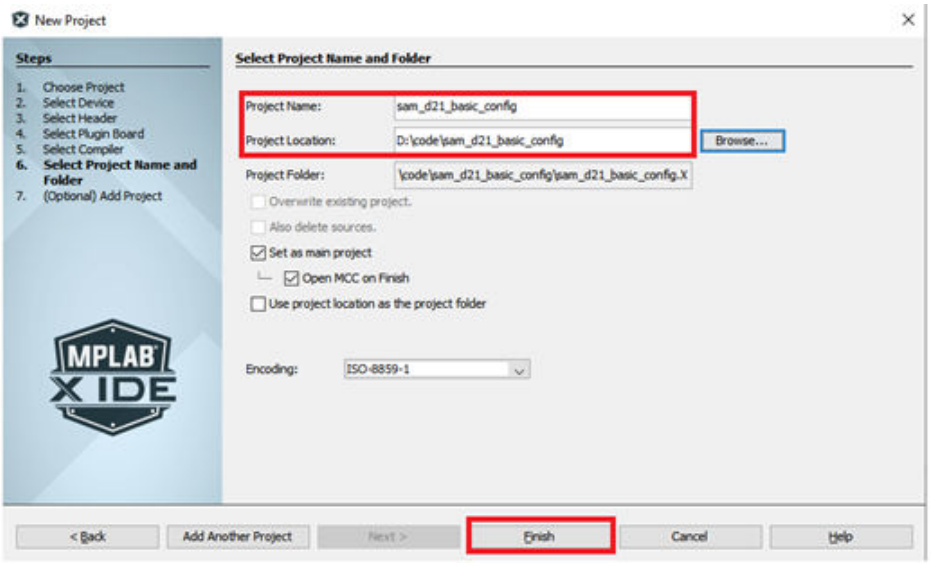
9. 在左侧导航栏中，单击 **Select Compiler**（选择编译器）。
10. 在 **Select Compiler** 属性页面中的 Compiler Toolchains（编译器工具链）下，单击并展开 XC32 选项列表，然后选择 **XC32 (v4.45)**。

图 5-4. XC32 编译器选择



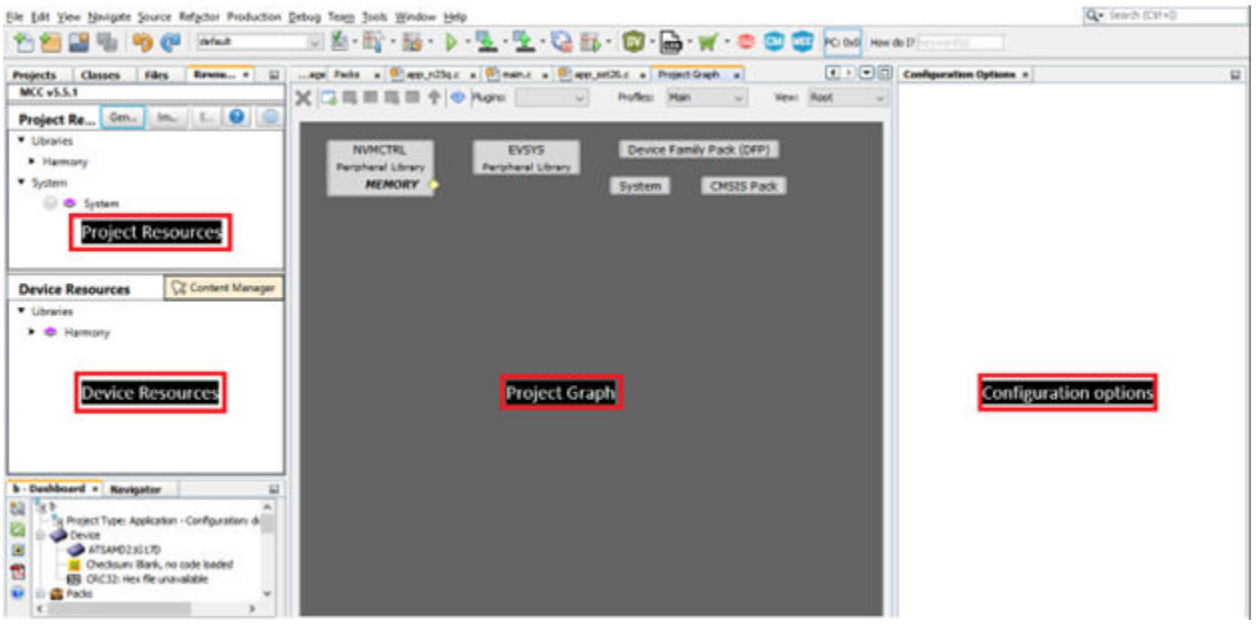
11. 单击 **Next**。
12. 在左侧导航栏中，单击 **Select Project Name and Folder**（选择项目名称和文件夹）。
13. 在 **Select Project Name and Folder** 属性页面中，输入项目名称和项目位置。

图 5-5. 项目名称和文件夹设置



14. 单击 **Finish**（完成）。随后将显示 MPLAB 代码配置器窗口。该界面包含 Project Resources（项目资源）、Device Resources（器件资源）、Project Graph（项目图）和 Configuration（配置）选项。

图 5-6. MPLAB 代码配置器窗口



注：可以使用创建项目一节中所述的操作步骤在 SAM E70 Xplained Ultra 评估工具包上创建一个项目，并在步骤 6 中进行修改。

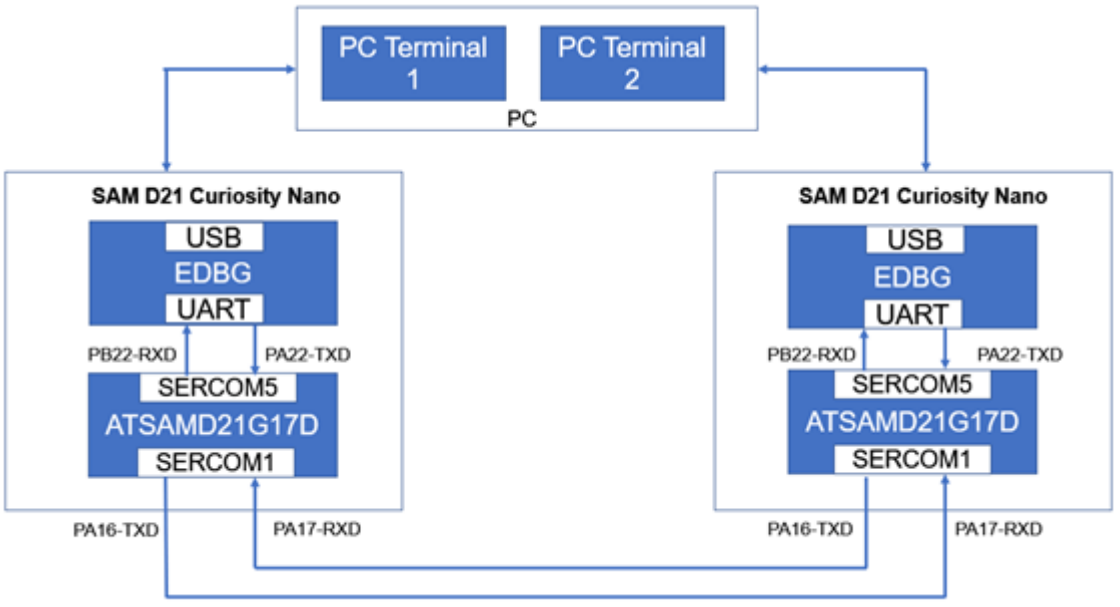
5.2. 基本配置

5.2.1. SAM D21 Curiosity Nano 评估工具包

两个 SAM D21 Curiosity Nano 评估工具包通过 SERCOM USART 线（TxD 和 RxD）相互连接，并通过 EDBG 端口连接到 PC 终端。

注：单击此处下载此应用程序配置的源代码。此外，也可在 GitHub 的 reference_apps 资源库中获取该代码。

图 5-7. 框图（SAM D21 Curiosity Nano 评估工具包）



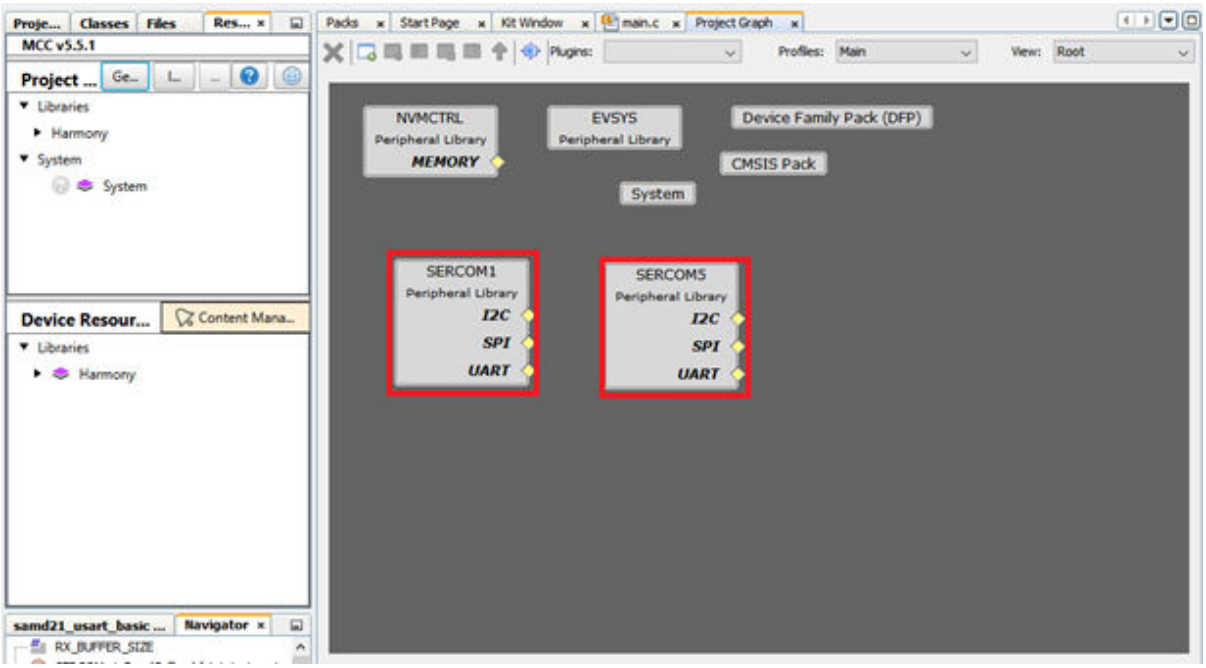
要使用 MCC 添加和配置 MPLAB Harmony 组件，请按照以下步骤操作：

1. 要创建项目，请参见创建项目。
2. 在 MCC 窗口中，单击 **Project Graph**。
3. 在 **Device Resources** 下，单击并展开选项列表 *Harmony* > *Peripherals* > *SERCOM*（Harmony > 外设 > SERCOM）。

图 5-8. 器件资源

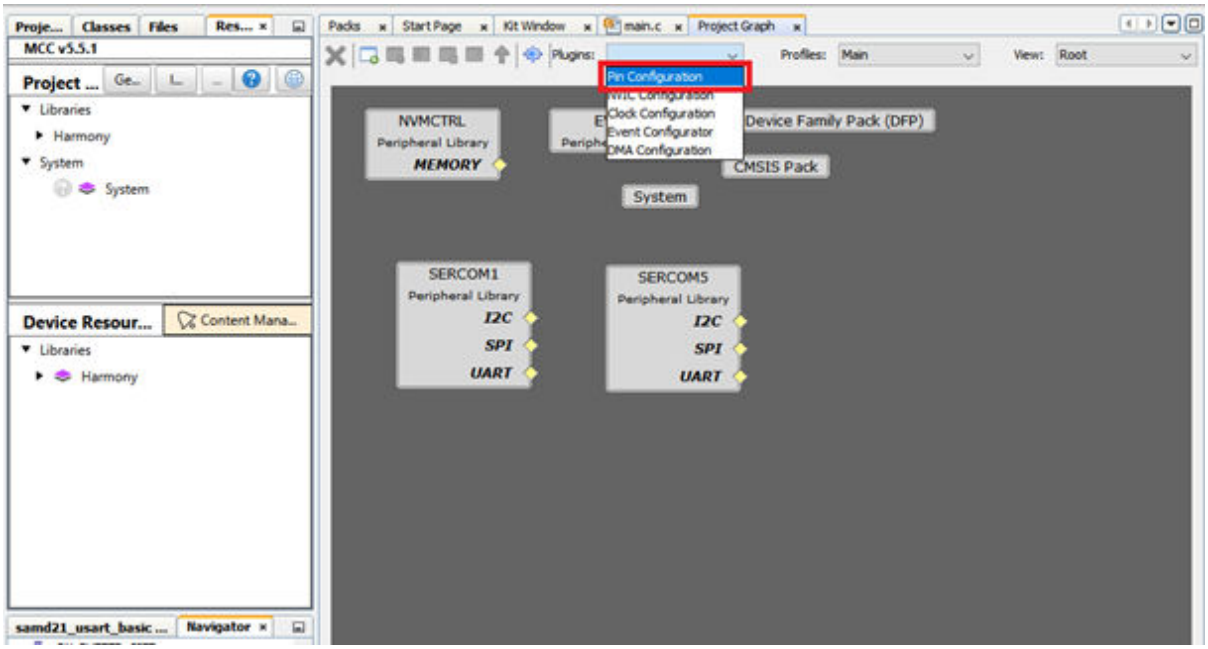
4. 单击 **SERCOM1** 和 **SERCOM5**。
5. 可观察到 Project Graph 窗口中已添加 SERCOM1 和 SERCOM5 外设库模块。

图 5-9. Project Graph 窗口中添加的 SERCOM1 与 SERCOM5



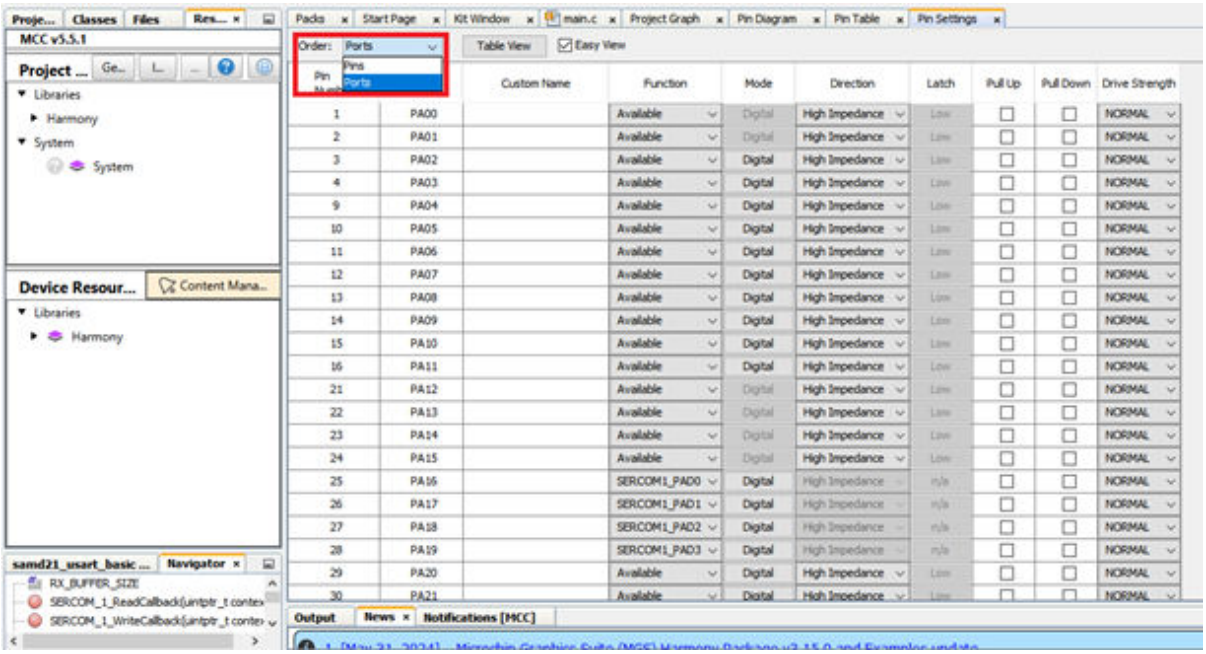
6. 从 **Plugins**（插件）下拉列表中，选择 **Pin Configuration**（引脚配置），然后单击 **Pin Settings**（引脚设置）（请参见图 5-12）。

图 5-10. 选择插件



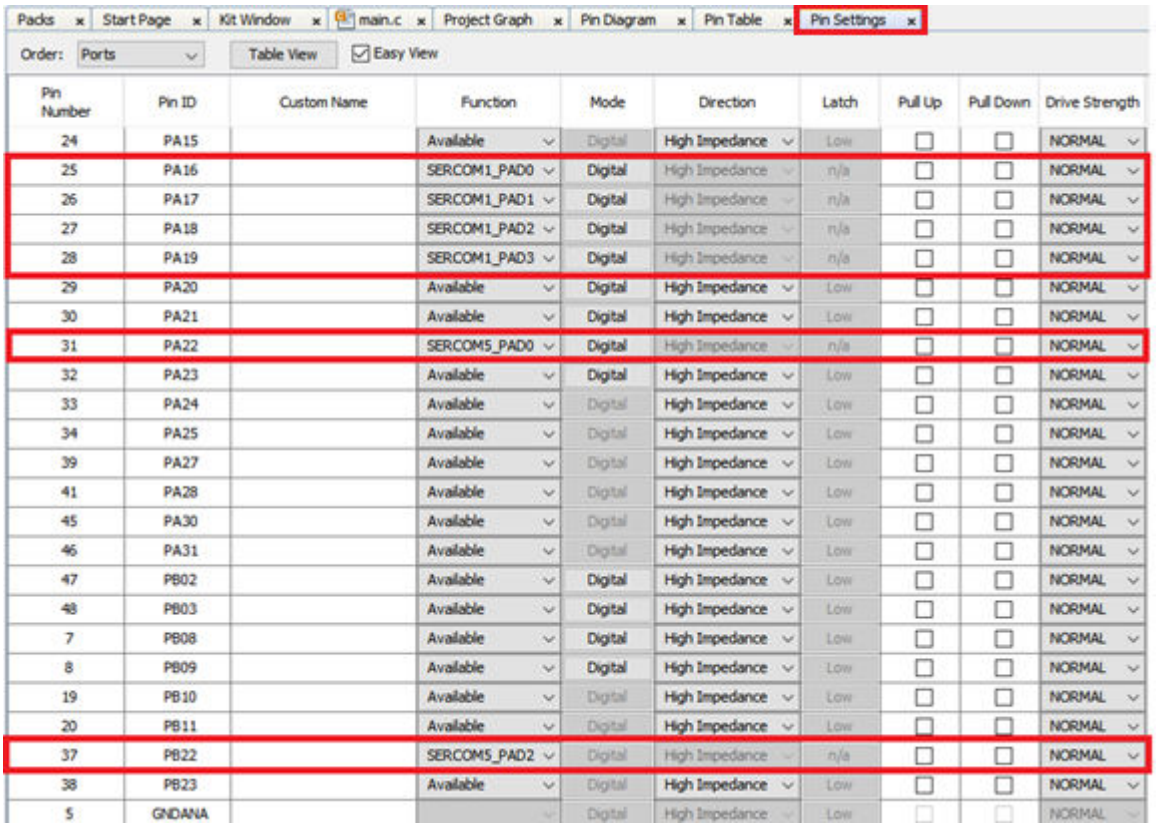
7. 从 **Order**（顺序）下拉列表中选择 **Ports**（端口）以根据应用程序编译配置，如下所示。

图 5-11. 引脚设置



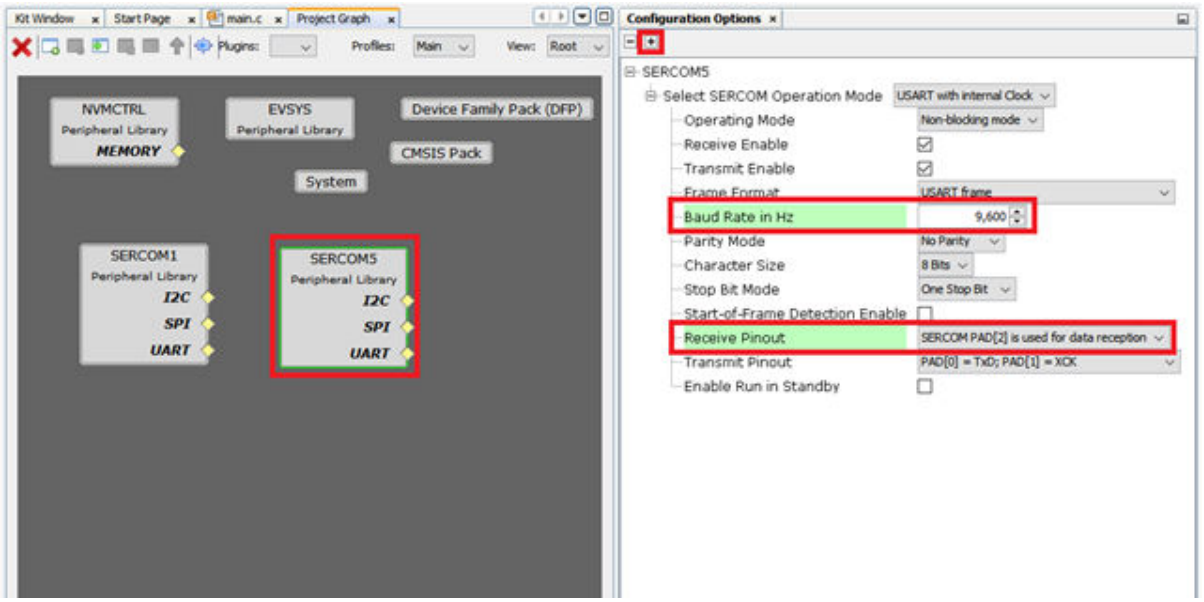
8. 将以下引脚 PA16、PA17、PA18、PA19、PA22 和 PB22 分别配置为 SERCOM1_PAD0、SERCOM1_PAD1、SERCOM1_PAD2、SERCOM1_PAD3、SERCOM5_PAD0 和 SERCOM5_PAD2。

图 5-12. 引脚配置



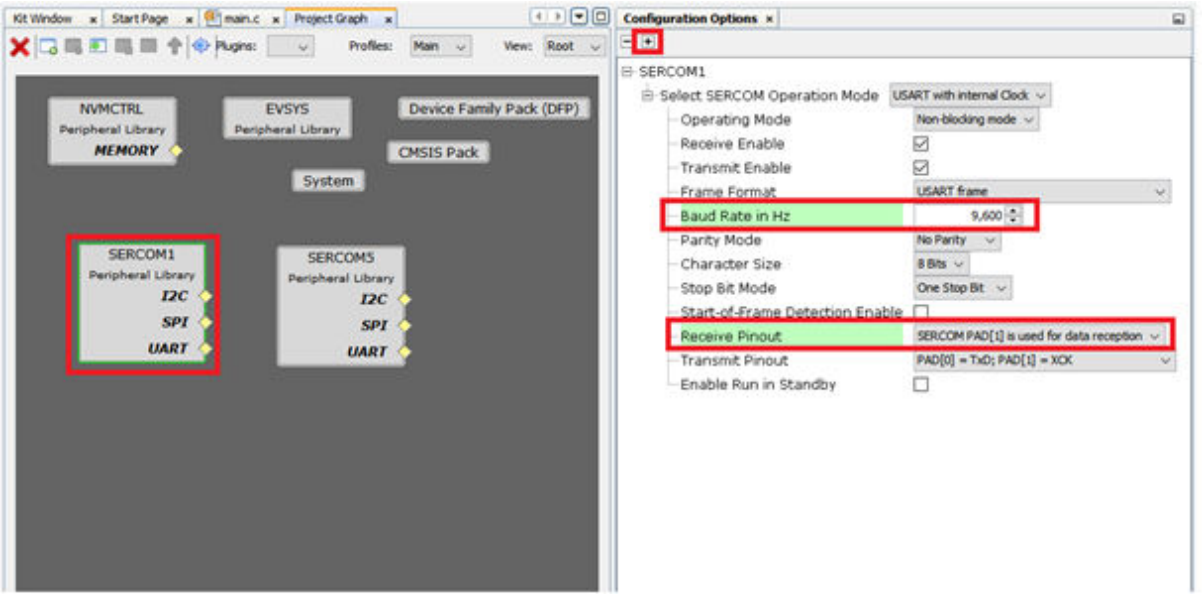
9. 在 **Project Graph** 窗口左侧导航栏中，选择 **SERCOM5 Peripheral Library**（SERCOM5 外设库），在右侧 **Configuration Options**（配置选项）属性页面中，进行如下配置以将数据按照 9600 的波特率打印在串行控制台上。

图 5-13. 更改 SERCOM5 中的 PAD 和波特率



10. 选择 **SERCOM1 Peripheral Library**（SERCOM1 外设库），然后在右侧 **Configuration Options** 属性页面中，进行如下配置以将数据按照 9600 的波特率打印在串行控制台上。

图 5-14. 更改 SERCOM1 中的 PAD 和波特率



11. 配置完外设后，单击 Project Resources 下的 **Generate**（生成）（如下图所示）。

图 5-15. 生成代码

注：SERCOM5 连接到 EDBG USART 线，SAM D21 Curiosity Nano 评估工具包将通过该线路与 PC 终端应用程序通信。

12. 将生成 `main.c` 文件，随即可实现应用程序的逻辑。

5.2.1.1. 应用程序逻辑

要开发和运行该应用程序，请按照以下步骤操作：

1. 打开位于源文件文件夹中的项目的 `main.c` 文件。在 main()函数外添加以下代码：

```
#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. 在下面的代码示例中，声明了读取回调和写入回调。这些函数设置指向从器件函数的指针，当给定 USART 的写入或读取事件发生时将调用该函数。这些回调描述了写入或读取事件完成时所要调用函数的指针。通过将该指针设置为 NULL，可将其禁止。

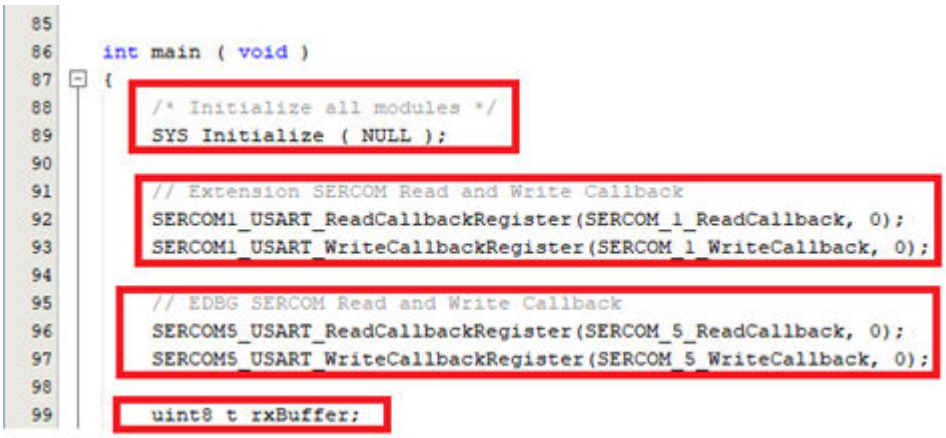
图 5-16. 事件处理程序

```
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. 在图 5-17 中，SYS_Initialize (NULL) 初始化各模块，如端口、时钟、SERCOM USART、嵌套向量中断控制器（Nested Vector Interrupt Controller，NVIC）和非易失性存储器控制器（Non-Volatile Memory Controller，NVMCTRL）。此外，还声明了用于读写操作的回调寄存器函数。
4. 在 main()函数内添加以下代码：

```
/* 初始化所有模块 */
    SYS_Initialize ( NULL );

    // 扩展 SERCOM 读写回调
    SERCOM1_USART_ReadCallbackRegister(SERCOM_1_ReadCallback, 0);
    SERCOM1_USART_WriteCallbackRegister(SERCOM_1_WriteCallback, 0);

    // EDBG SERCOM 读写回调
    SERCOM5_USART_ReadCallbackRegister(SERCOM_5_ReadCallback, 0);
    SERCOM5_USART_WriteCallbackRegister(SERCOM_5_WriteCallback, 0);

    uint8_t rxBuffer;
```

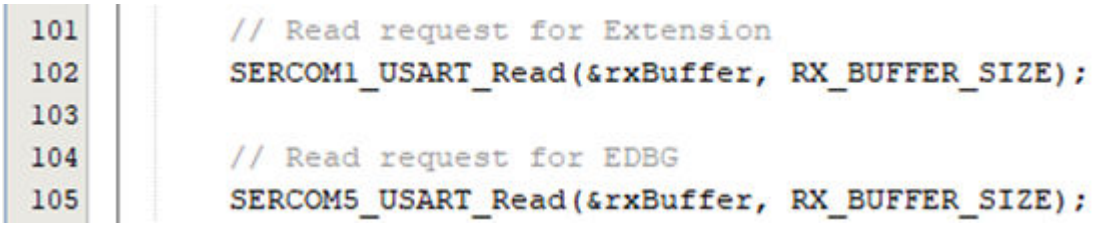
图 5-17. 模块初始化



5. 通过 SERCOM1（用于扩展）和 SERCOM5（用于 EDBG）的读请求启动实现。

```
// 扩展的读请求
    SERCOM1_USART_Read(&rxBuffer, RX_BUFFER_SIZE);

// EDBG 的读请求
    SERCOM5_USART_Read(&rxBuffer, RX_BUFFER_SIZE);
```

图 5-18. 读请求



6. 在图 5-19 中，如果扩展已准备好读取（Rx）数据（即 SERCOM_1_readStatus == true），则将接收数据发送到 EDBG。以及，如果扩展已准备好写入（Tx）数据（即 SERCOM_1_writeStatus == true），则 EDBG 将启动读取（Rx）请求。类似地，如果 EDBG 已准备好读取（Rx）或已准备好写入（Tx），则反之亦然。将以下代码添加到 while 循环内：

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

            //发送从 EDBG 接收的字节
            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;

            //发送从扩展接收的字节
            SERCOM1_USART_Write(&rxBuffer, RX_BUFFER_SIZE);
        }

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

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

图 5-19. 实现基本配置

```
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     }
```

图 5-20. 主器件端

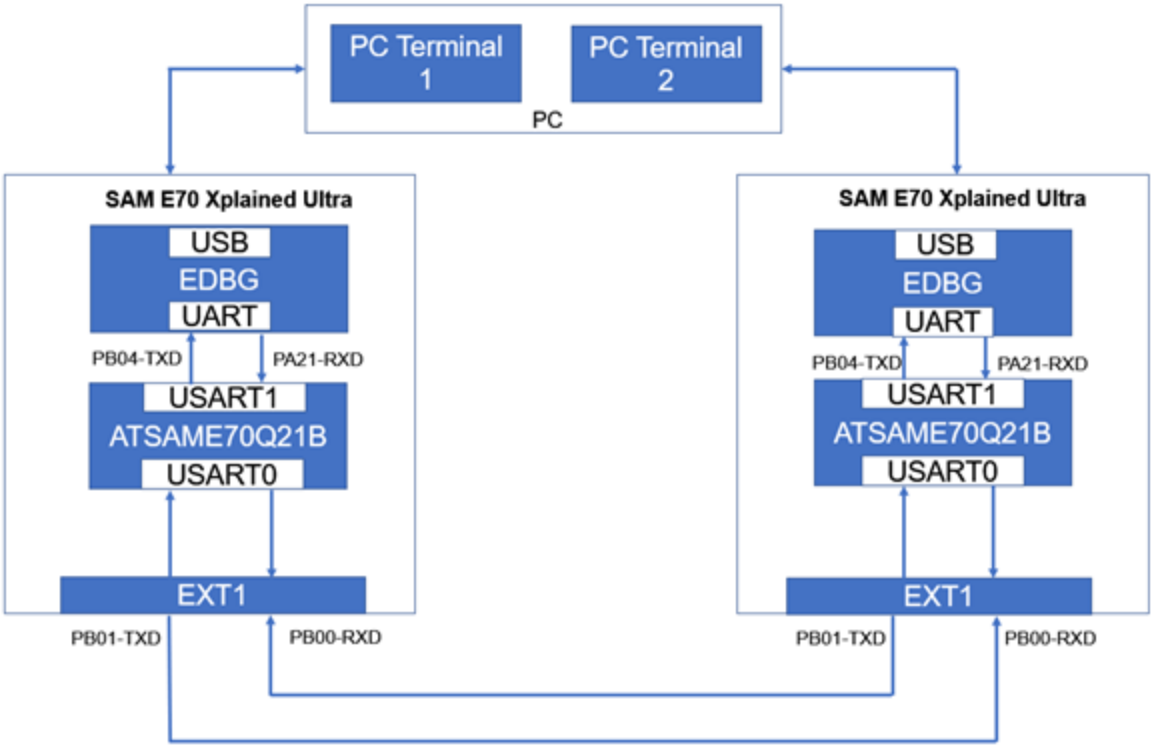
图 5-21. 从器件端

5.2.2. SAM E70 Xplained Ultra 评估工具包

两个 SAM E70 Xplained Ultra 评估工具包通过 EXT1 连接器的 SERCOM USART 线（TxD 和 RxD）互连，并通过 EDBG 端口连接到 PC 终端。下图给出了框图。

注：单击此处下载此应用程序配置的源代码。此外，也可在 GitHub 的 reference_apps 资源库中获取该代码。

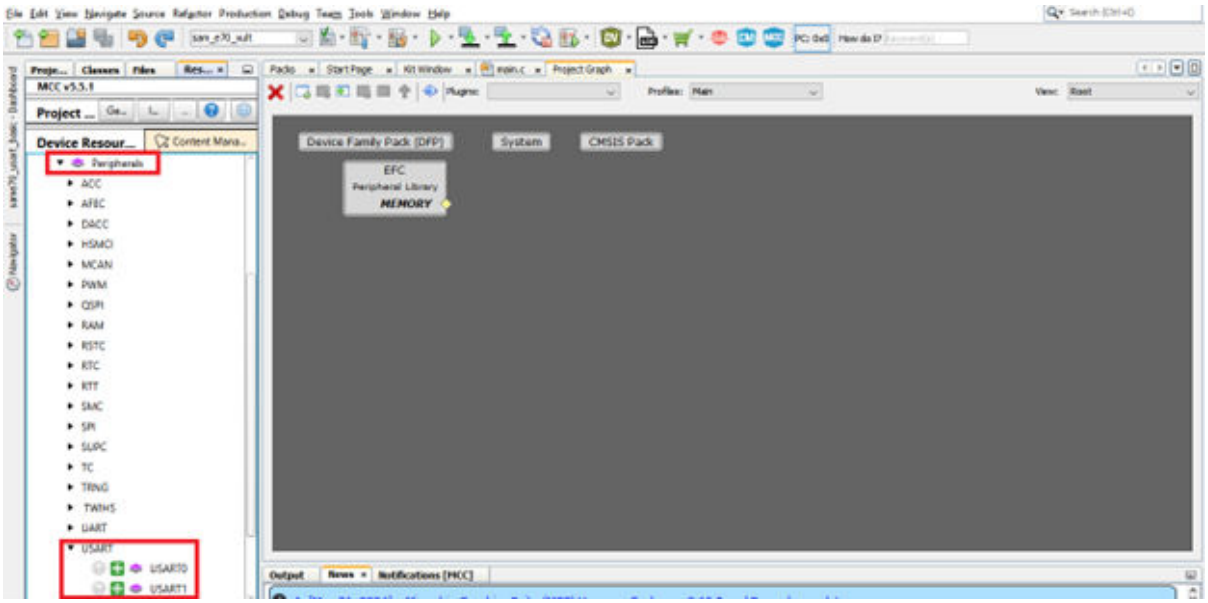
图 5-22. 框图——SAM E70 Xplained Ultra 评估工具包



要使用 MCC 添加和配置 MPLAB Harmony 组件，请按照以下步骤操作：

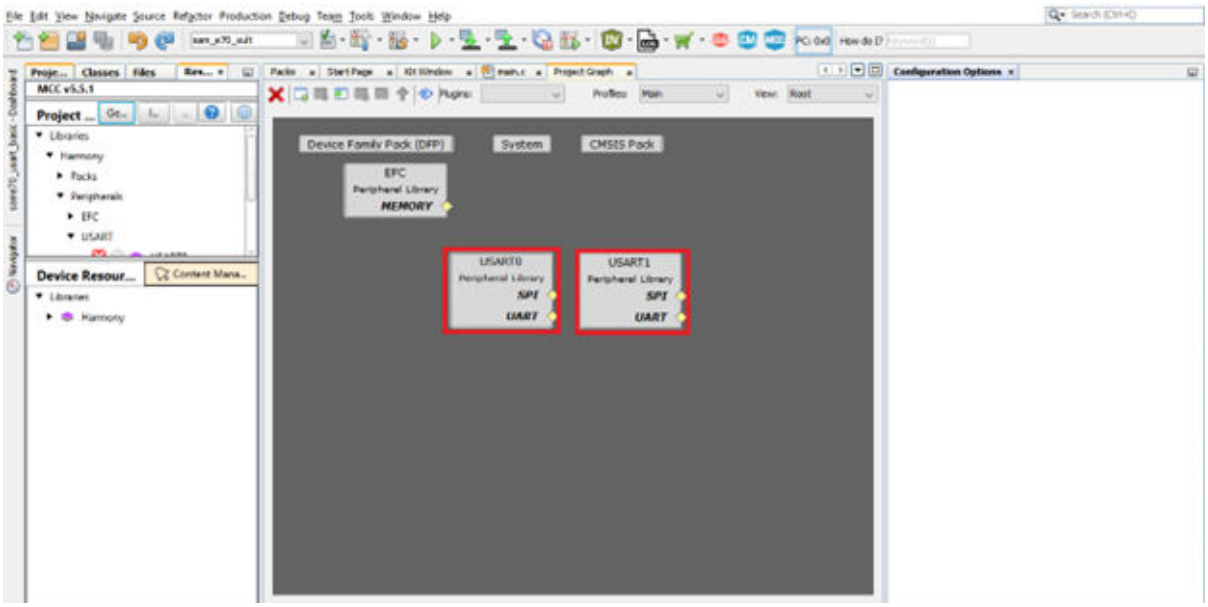
1. 要创建项目，请参见创建项目。
2. 在 MCC 窗口中，单击 **Project Graph**。
3. 在 **Device Resources** 下，单击并展开选项列表 *Harmony* > *Peripherals* > *USART*（Harmony > 外设 > USART）。

图 5-23. 器件资源



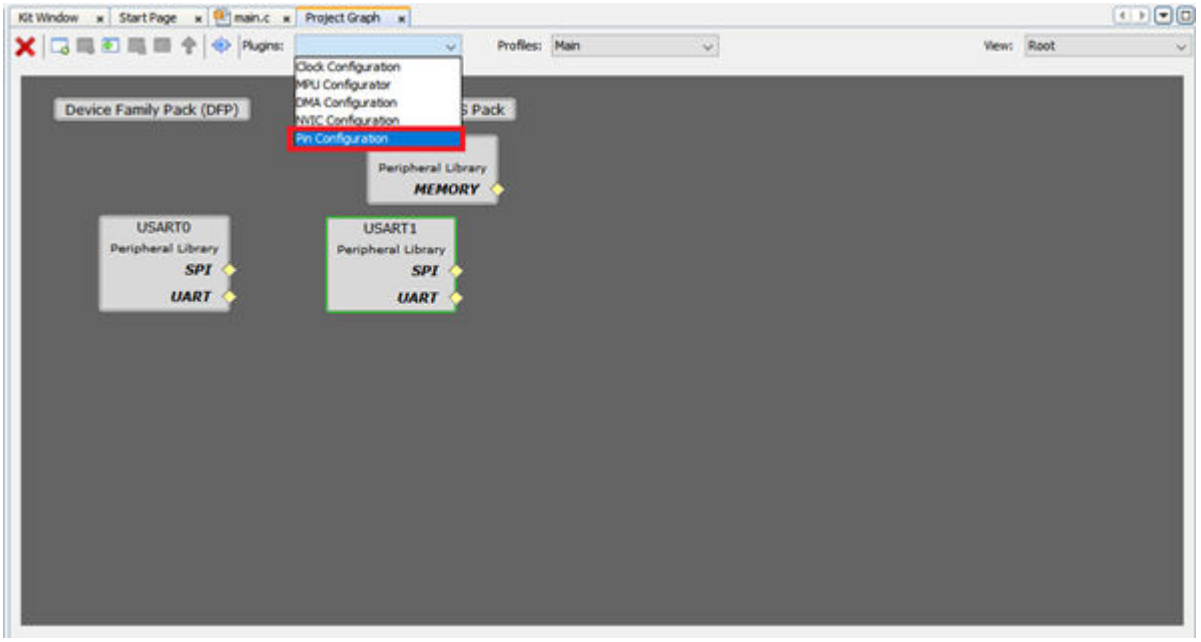
4. 单击 **USART0** 和 **USART1**。
5. 可观察到 Project Graph 窗口中已添加 USART0 和 USART1 外设库模块。

图 5-24. 添加 USART0 和 USART1 模块



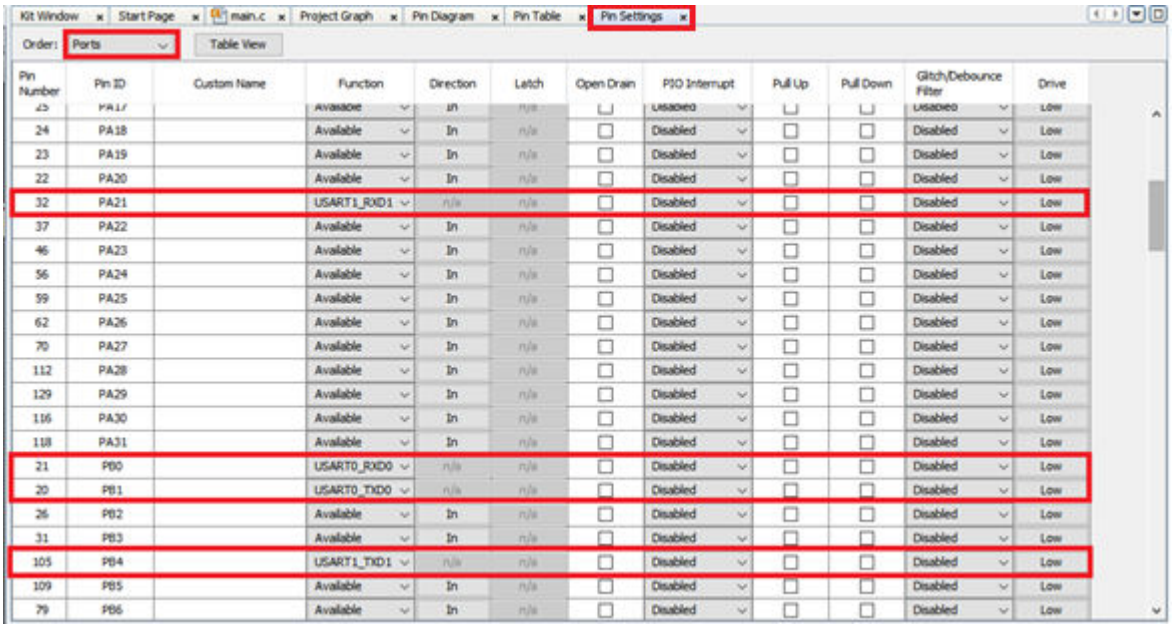
6. 从 **Plugins** 下拉列表中，选择 **Pin Configuration**，然后单击 **Pin Settings**（请参见图 5-26）。

图 5-25. 选择插件



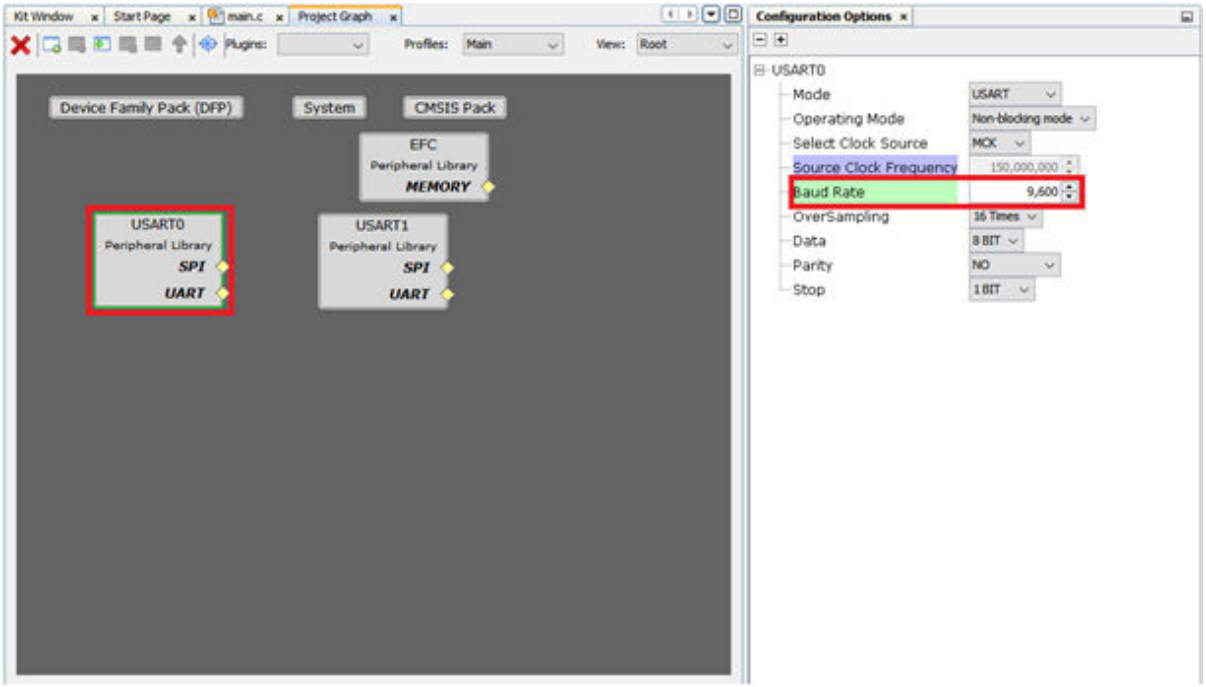
7. 从 **Order** 下拉列表中选择 **Ports**。根据下面所示的应用程序编译配置。
将以下引脚 PA21、PB0、PB1、PB4 分别配置为 USART1_RXD1、USART0_RXD0、USART0_TXD0、USART1_TXD1。

图 5-26. 引脚配置



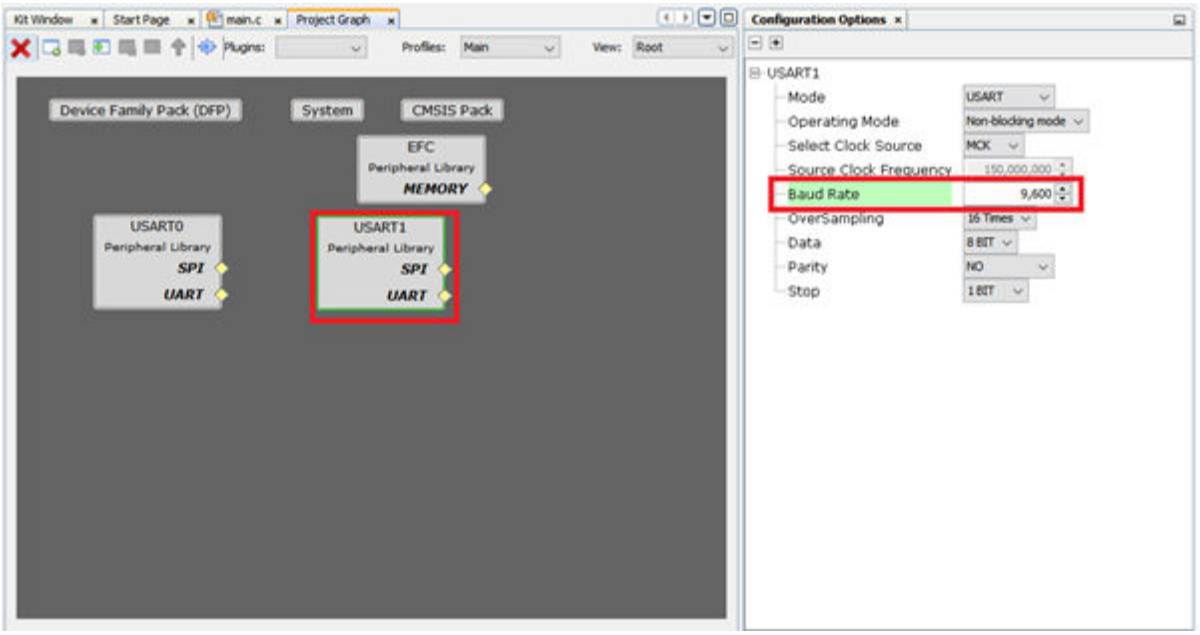
8. 在 **Project Graph** 窗口左侧导航栏中，选择 **USART0 Peripheral Library**（USART0 外设库），在右侧 **Configuration Options** 属性页面中，进行如下配置以将数据按照 9600 的波特率打印在串行控制台上。

图 5-27. 更改 USART0 的波特率



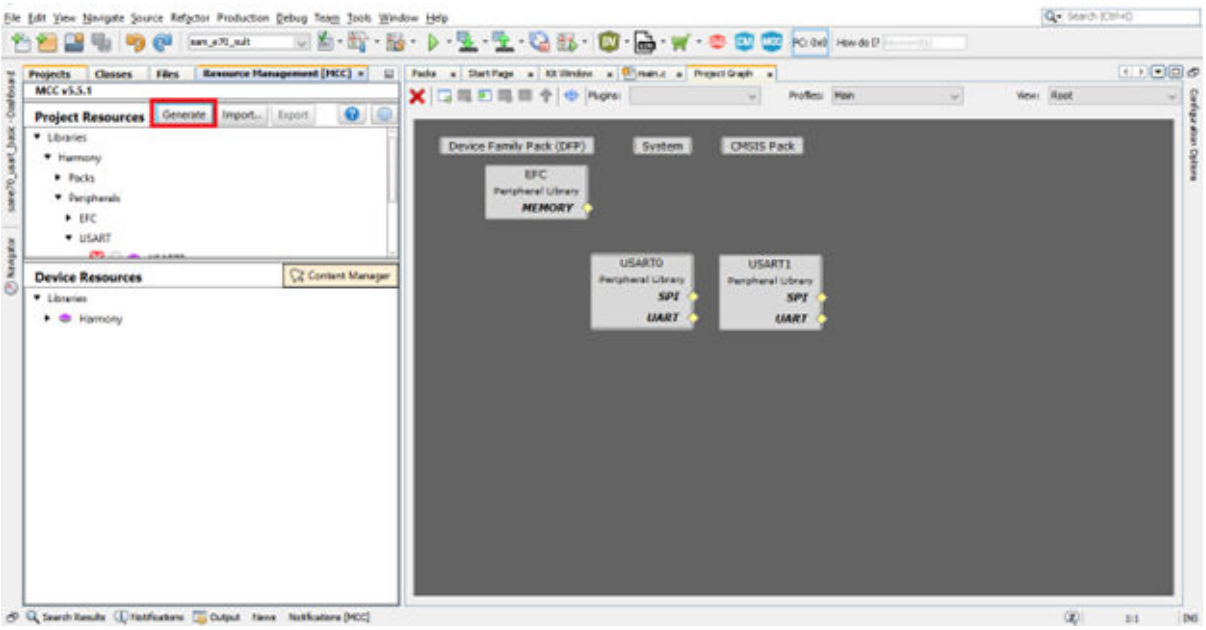
9. 选择 **USART1 Peripheral Library**（USART1 外设库），然后在右侧 **Configuration Options** 属性页面中，进行如下配置以将数据按照 9600 的波特率打印在串行控制台上。

图 5-28. 更改 USART1 的波特率



10. 配置完外设后，单击 Project Resources 下的 **Generate**（生成）（如下图所示）。

图 5-29. 生成代码



注：USART1 连接到 EDBG USART 线，SAM E70 Xplained Ultra 评估工具包将通过该线路与 PC 终端应用程序通信。

11. 将生成 `main.c` 文件，随即可实现应用程序的逻辑。

5.2.2.1. 应用程序逻辑

要开发和运行该应用程序，请按照以下步骤操作：

1. 打开位于源文件文件夹中的项目的 `main.c` 文件。在 main()函数外添加以下代码：

```
#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. 在下图中，读写事件处理程序的声明方式与 SAM D21 Curiosity Nano 评估工具包中的回调寄存器相同。

图 5-30. 事件处理程序

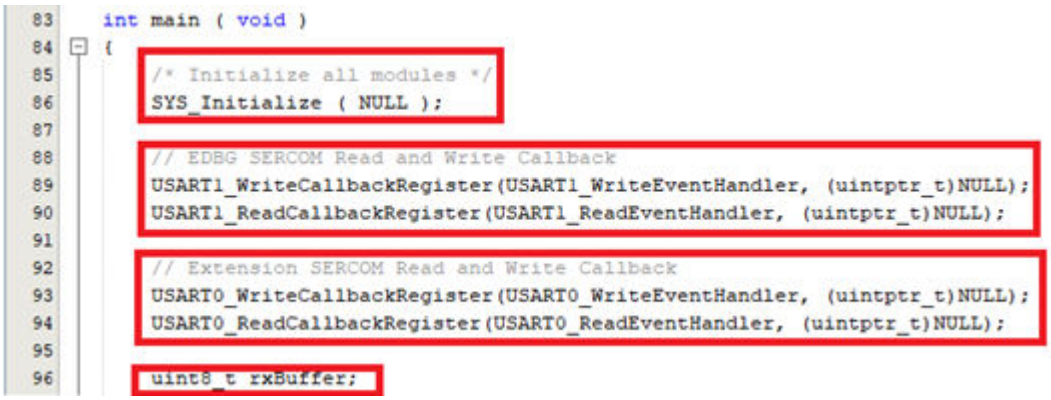
```
57  #define RX_BUFFER_SIZE 1
58  bool USART1_writeStatus = false;
59  bool USART1_readStatus = false;
60
61  bool USART0_writeStatus = false;
62  bool USART0_readStatus = false;
63
64  void USART1_WriteEventHandler ( uintptr_t context )
65  {
66      USART1_writeStatus = true;
67  }
68
69  void USART0_WriteEventHandler ( 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. 在 main()函数内添加以下代码：

```
/*初始化所有模块*/
    SYS_Initialize ( NULL );

    // EDBG SERCOM 读写回调
    USART1_WriteCallbackRegister(USART1_WriteEventHandler, (uintptr_t)NULL);
    USART1_ReadCallbackRegister(USART1_ReadEventHandler, (uintptr_t)NULL);

    // 扩展 SERCOM 读写回调
    USART0_WriteCallbackRegister(USART0_WriteEventHandler, (uintptr_t)NULL);
    USART0_ReadCallbackRegister(USART0_ReadEventHandler, (uintptr_t)NULL);

    uint8_t rxBuffer;
```

4. 在下面的代码示例中，`SYS_Initialize (NULL)`初始化时钟、嵌套向量中断控制器（NVIC）、嵌入式闪存控制器（Embedded Flash Controller，EFC）、并行输入/输出控制器（Parallel In/Out Controller，PIO）和 USART 等模块。此外，还声明了用于读写操作的回调寄存器函数。

图 5-31. 模块初始化

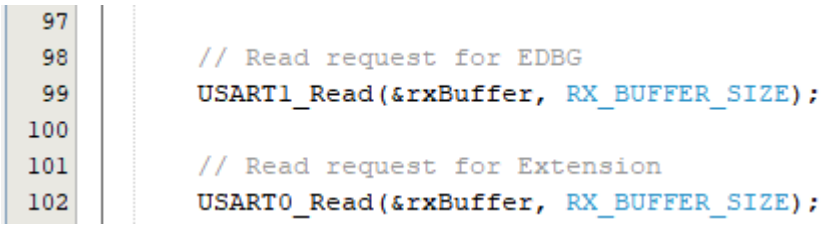


5. 在模块初始化后添加如下代码：

```
// EDBG 的读请求
    USART1_Read(&rxBuffer, RX_BUFFER_SIZE);

// 扩展的读请求
    USART0_Read(&rxBuffer, RX_BUFFER_SIZE);
```

6. 通过 USART0（用于扩展）和 USART1（用于 EDBG）的读请求启动实现。

图 5-32. 读请求



7. 将以下代码添加到 while 循环内：

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

            //发送从 EDBG 接收的字节
            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;
            //发送从扩展接收的字节
            USART0_Write(&rxBuffer, RX_BUFFER_SIZE);
        }
        if(USART0_writeStatus == true)
        {
            USART0_writeStatus = false;
            USART1_Read(&rxBuffer, RX_BUFFER_SIZE);
        }
```

8. 在下图中，如果扩展已准备好读取（Rx）数据（即 `USART0_readStatus == true`），则会将接收到的数据发送到 EDBG；如果扩展已准备好写入（Tx）数据（即 `USART1_writeStatus ==`

true），则 EDBG 将启动读取（Rx）请求。类似地，如果 EDBG 已准备好读取（Rx）或已准备好写入（Tx），则反之亦然。

图 5-33. 基本配置的实现

```
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     }
```

图 5-34. 主器件端

图 5-35. 从器件端

5.3. 小数波特率配置

除了两个 SAM D21 Curiosity Nano 评估工具包与两个 SAM E70 Xplained Ultra 评估工具包之间的波特率配置外，本节的其余部分与基本配置类似。

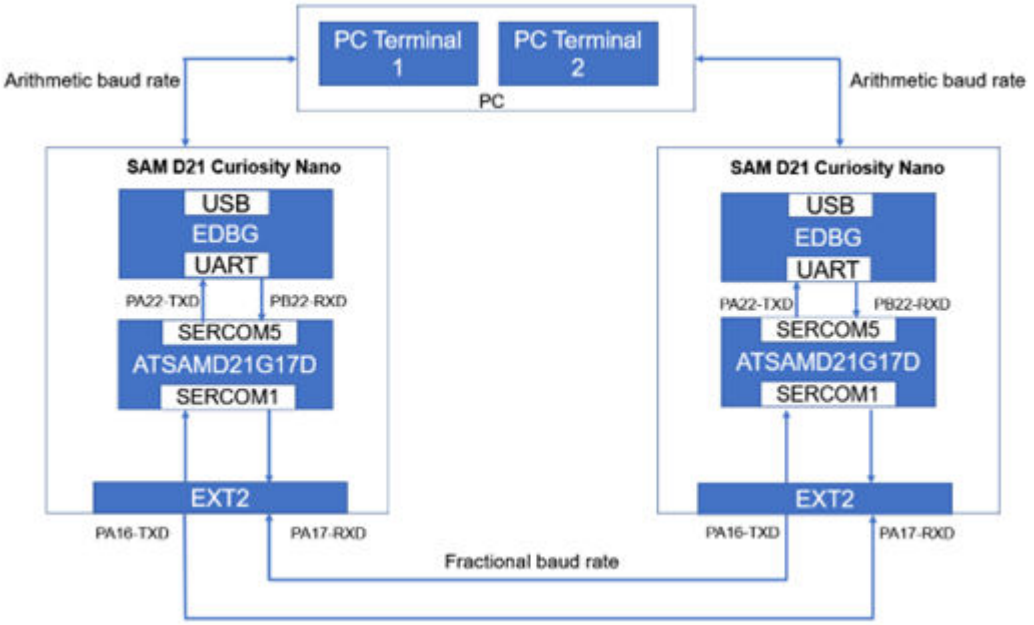
PC 终端与开发板（SAM D21 Curiosity Nano 评估工具包或 SAM E70 Xplained Ultra 评估工具包）之间的通信波特率为算术波特率。相比之下，两个开发板（SAM D21 Curiosity Nano 评估工具包或 SAM E70 Xplained Ultra 评估工具包）之间的通信为小数波特率。

5.3.1. SAM D21 Curiosity Nano 评估工具包

两个 SAM D21 Curiosity Nano 评估工具包通过 SERCOM USART 线（TxD 和 RxD）相互连接，并通过 EDBG 端口连接到 PC 终端。

注：单击此处下载此应用程序配置的源代码。此外，也可在 GitHub 的 reference_apps 资源库中获取该代码。

图 5-36. 框图（SAM D21 Curiosity Nano 评估工具包）



必须使用小数波特率公式来计算波特率值：

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

其中，

fbaud——小数波特率频率
fref——SERCOM 通用时钟频率
S——每位采样数
BAUD——波特率值
FP——波特率值的小数部分

根据小数波特率公式，

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

这里的整数部分对应于 BAUD 值，小数部分对应于小数部分。

BAUD = 45，

FP/8 = .454

FP = 3.6，即 3

5.3.1.1. 应用程序逻辑

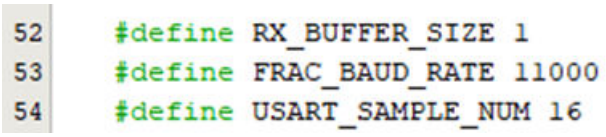
本节仅说明小数波特率部分，其余内容将在基本配置一节中介绍。应用程序中使用 11000 bps 的小数波特率。宏 `FRAC_BAUD_RATE` 包含小数波特率值。遵循 SAM D21 Curiosity Nano 评估工具包基本配置的步骤。

1. 要使用 MCC 添加和配置 MPLAB Harmony 组件，请参见 SAM D21 Curiosity Nano 评估工具包。

2. 添加以下宏：

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

图 5-37. 定义小数波特率的宏



3. 添加以下变量：

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

4. 添加事件处理程序，请参见应用程序逻辑一节的中步骤 1。
5. 在 main()函数外添加以下代码：

```
// 计算小数波特率值的函数
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;
}

// 使用小数波特率设置初始化 USART
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. 在以下代码示例中，使用 `calculate_fractional_baud_value()` 函数计算小数波特率的值。在 `ext_usart_init()` 函数中，SERCOM1 可被禁止和使能，还可设置 CTRLA 和 BAUD 寄存器。使用 `calculate_fractional_baud_value()` 函数计算小数波特率的值。在 `ext_usart_init()` 函数中，SERCOM1 可被禁止和使能，还可设置 CTRLA 和 BAUD 寄存器。

图 5-38. 小数波特率配置的实现

```
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. 按照应用程序逻辑部分的步骤 4 所示添加其余代码。

图 5-39. 主器件端

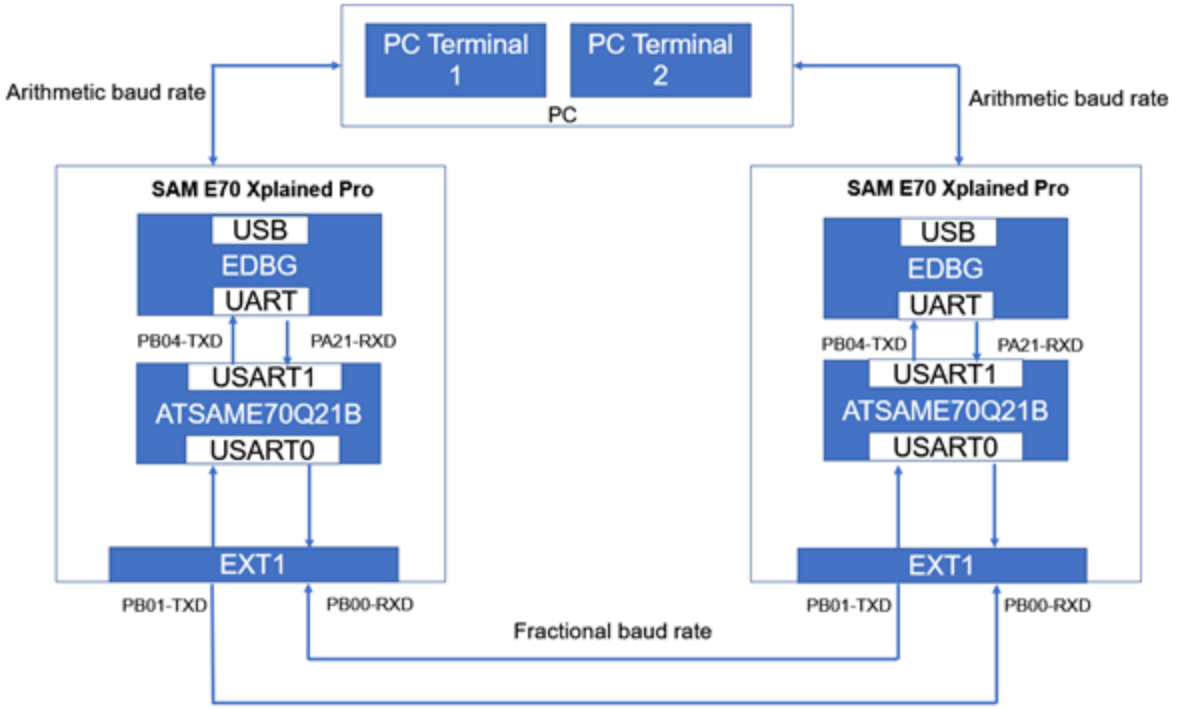
图 5-40. 从器件端

5.3.2. SAM E70 Xplained Ultra 评估工具包

本节仅说明小数波特率部分，其余内容与基本配置一节相同。应用程序中使用 11000 bps 的小数波特率。宏 FRAC_BAUD_RATE 包含小数波特率值。遵循 SAM E70 Xplained Ultra 评估工具包基本配置的步骤。下图给出了框图。

注：单击此处下载此应用程序配置的源代码。此外，也可在 GitHub 的 reference_apps 资源库中获取该代码。

图 5-41. 框图（SAM E70 Xplained Ultra 评估工具包）



5.3.2.1. 应用程序逻辑

1. 要使用 MCC 添加和配置 MPLAB Harmony 组件，请参见 SAM E70 Xplained Ultra 评估工具包。
2. 添加以下宏：

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

图 5-42. 定义小数波特率的宏

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

3. 添加以下变量：

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

4. 添加事件处理程序，请参见应用程序逻辑中的步骤 1。

5. 在 main()函数外添加以下代码：

```
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. 在以下代码示例中，使用 `calculate_fractional_baud_value()` 函数计算小数波特率的值。为实现小数波特率配置，寄存器（US_CR 和 US_BRGR）操作在 `ext_usart_init()` 函数中完成。

图 5-43. 小数波特率配置的实现

```
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. 添加其余代码，请参见应用程序逻辑中的步骤 4。

图 5-44. 主器件端

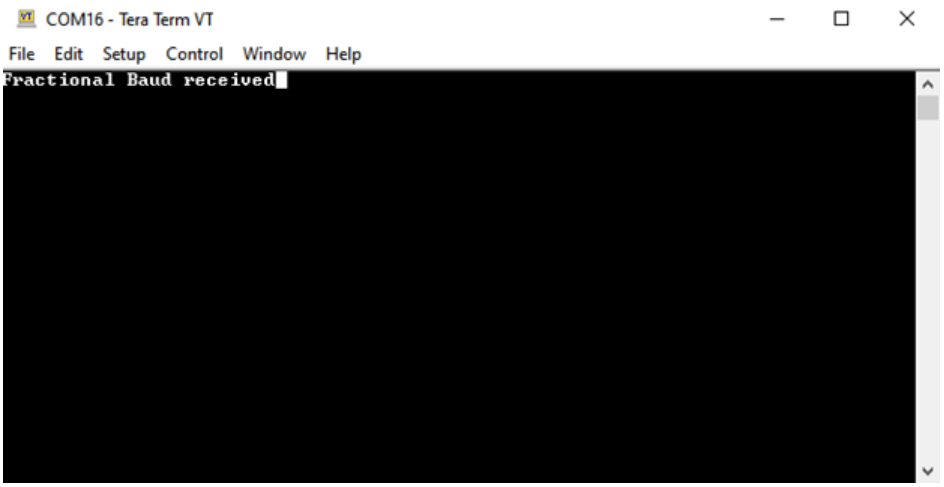


图 5-45. 从器件端

5.4. 硬件握手配置

USART 具有带外硬件握手流控制机制，通过将 RTS 和 CTS 线与远程设备连接可实现这种机制。这种方法可确保在设备之间可靠地发送和接收数据。它通常涉及使用 RTS（请求发送）和 CTS（允许发送）等附加控制线来管理数据流。当一个设备准备好发送数据时，它会将 RTS 线置为有效，而接收设备在准备好接收数据时，会通过将 CTS 线置为有效做出响应。

5.4.1. SAM D21 Curiosity Nano 评估工具包

两个 SAM D21 Curiosity Nano 评估工具包通过 SERCOM USART 线（TxD、RxD、RTS 和 CTS）相互连接，并通过 EDBG 端口连接到 PC 终端。

注：单击此处下载此应用程序配置的源代码。此外，也可在 GitHub 的 reference_apps 资源库中获取该代码。

图 5-46. 框图（SAM D21 Curiosity Nano 评估工具包）

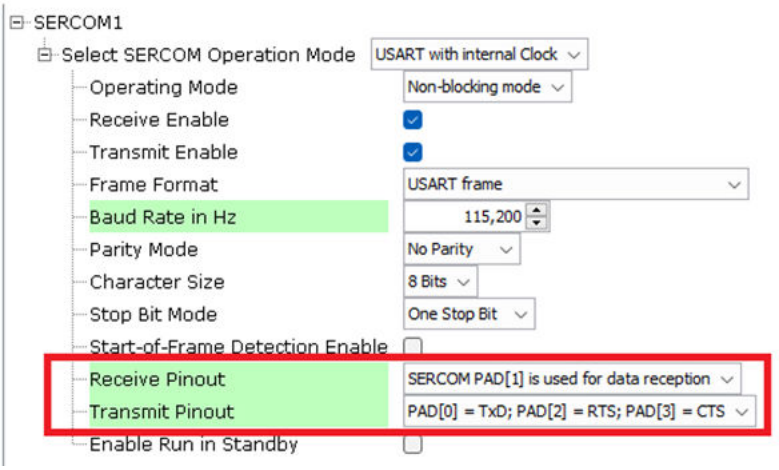
在此应用程序中，仅使用扩展 SERCOM1。在此演示中，定义的数据 0xAA 将由两个 SAM D21 Curiosity Nano 板通过握手协议发送。

本节与基本配置类似，但需要额外配置硬件流控制信号线并在代码执行中进行一些修改。本节仅讨论这些更改。

用户需要遵循 SAM D21 Curiosity Nano 评估工具包基本配置的以下步骤。

1. 要使用 MCC 添加和配置 MPLAB Harmony 组件，请参见 SAM D21 Curiosity Nano 评估工具包。
2. 单击并展开 **SERCOM1**，然后配置 SERCOM1 外设库块，如下所示：

图 5-47. SERCOM1 配置



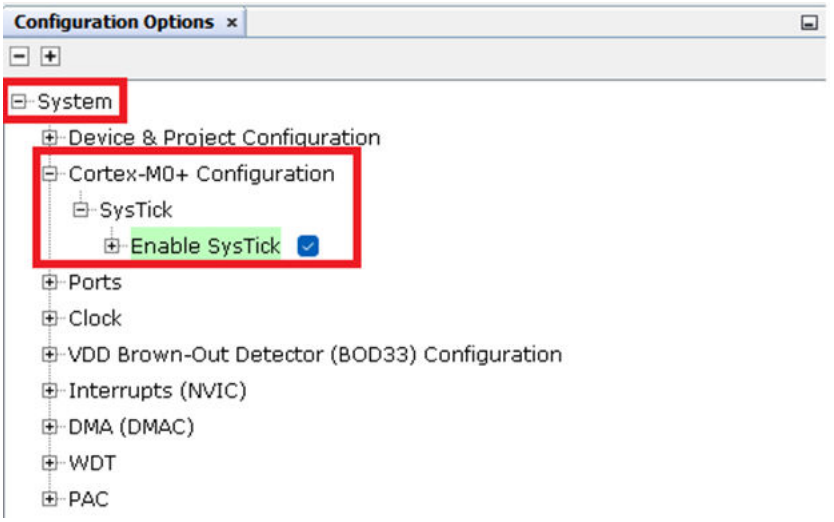
3. 按如下所述配置引脚：

图 5-48. 引脚配置

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. 在 MCC 的 **Configuration Options** 中，单击并展开 **System**（系统），然后选择 **Enable SysTick**（使能 SysTick）。

图 5-49. 使能 Systick 定时器



5. 配置外设后，单击 Project Resources 下的 **Generate**，如 SAM D21 Curiosity Nano 评估工具包一节的步骤 9 所示。

5.4.1.1. 应用程序逻辑

要开发和运行该应用程序，请按照以下步骤操作：

1. 打开位于源文件文件夹中的项目的 `main.c` 文件。在 main()函数外添加以下代码：

```
#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. 下图中声明了读取回调和写入回调。这些函数设置指向从器件函数的指针，当给定 USART 的写入或读取事件发生时将调用该函数。读取完成后将调用写入回调，该回调中的写函数会将数据写入控制台。这些回调描述了写入或读取事件完成时所要调用函数的指针。通过将该指针设置为 NULL，可将其禁止。

图 5-50. 事件处理程序

```
#define RX_BUFFER_SIZE 1
#define TX_BUFFER_SIZE 1
// *****************************************************************************
// *****************************************************************************
// Section: Main Entry Point
// *****************************************************************************
// *****************************************************************************

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;
}
```

3. 在 main()函数内添加以下代码：

```
SYSTICK_TimerStart();

 // 扩展 SERCOM 读写回调
 SERCOM1_USART_WriteCallbackRegister(APP_SERCOM_1_WriteCallback, 0);
 SERCOM1_USART_ReadCallbackRegister(APP_SERCOM_1_ReadCallback, 0);

 // 扩展的读取请求
 SERCOM1_USART_Write(&txBuffer, TX_BUFFER_SIZE);
 SERCOM1_USART_Read(&rxBuffer, RX_BUFFER_SIZE);
```

4. 在下图中，SYS_Initialize (NULL) 初始化各模块，如端口、时钟、SERCOM USART、嵌套向量中断控制器（NVIC）和非易失性存储器控制器（NVMCTRL）。声明了用于读写操作的回调寄存器函数。此外，还实现了 SYSTICK 定时器和 SERCOM1（用于扩展）的读写请求。

图 5-51. 模块初始化

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

5. 将以下代码添加到 while 循环内：

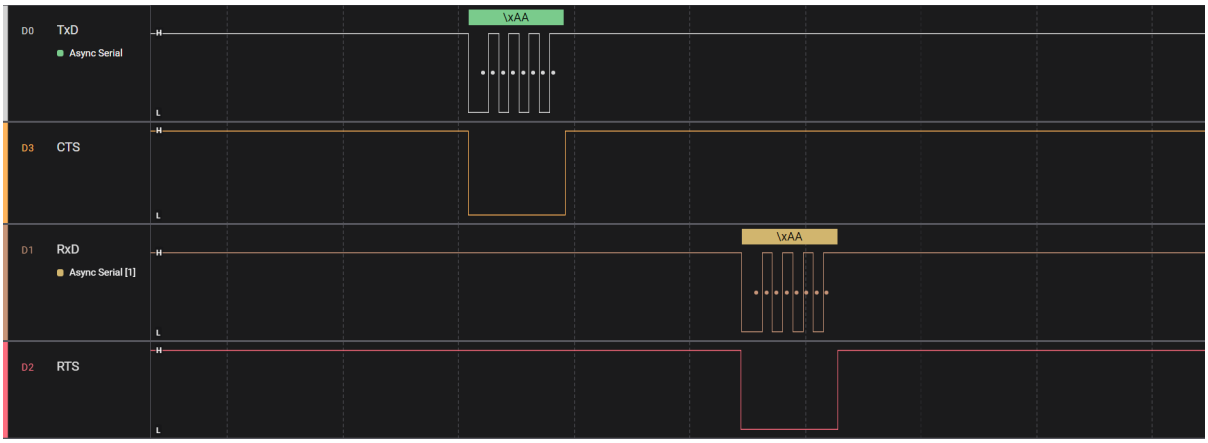
```
SYSTICK_DelayMs(1U);

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

6. 在下面的代码示例中，为可视化使用 RTS 和 CTS 线的硬件流控制，在代码中引入了 1 毫秒的延迟。

图 5-52. 硬件握手配置的实现

```
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     }
```

图 5-53. SAM D21 Curiosity Nano——硬件握手输出

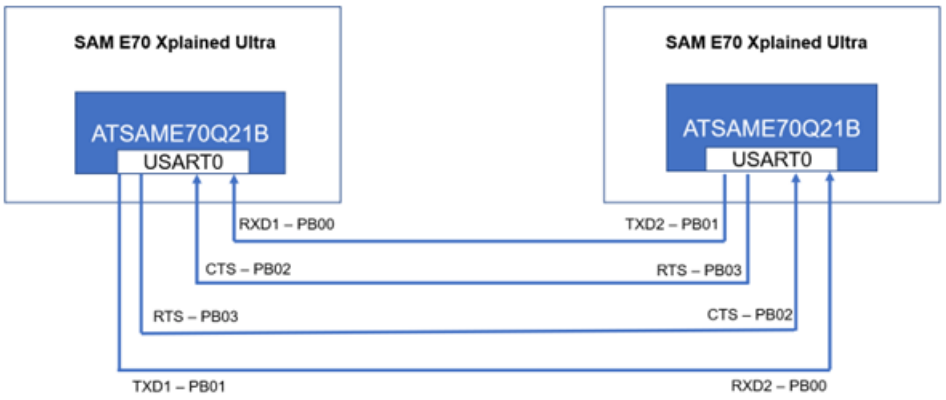


5.4.2. SAM E70 Xplained Ultra 评估工具包

两个 SAM E70 Xplained Ultra 评估工具包由 EXT1 连接器通过 SERCOM USART 线（TxD、RxD、RTS 和 CTS）相互连接，并通过 EDBG 端口连接到 PC 终端。

注：单击此处下载此应用程序配置的源代码。此外，也可在 GitHub 的 reference_apps 资源库中获取该代码。

图 5-54. 框图（SAM E70 Xplained Ultra 评估工具包）



本节仅说明硬件握手部分，其余内容在基本配置一节中介绍。其中，RTS 和 CTS 线是手动控制的，而在 SAM D21 中则不是。遵循 SAM E70 Xplained Ultra 评估工具包基本配置的步骤。

1. 要使用 MCC 添加和配置 MPLAB Harmony 组件，请参见 SAM E70 Xplained Ultra 评估工具包。
2. 在 **Pin Configuration** 中，为 RTS 和 CTS 配置 PB2 和 PB3 引脚。

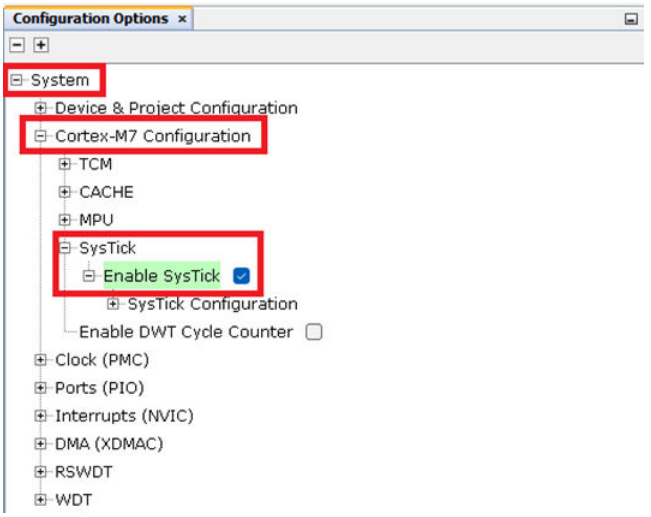
图 5-55. 引脚配置

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. 在 MCC 配置选项中，单击并展开 *System* > *Contex-M7 Configuration* > *SysTick*（系统 > Contex-M7 配置 > SysTick）。
4. 选择 **Enable SysTick**。

图 5-56. 使能 Systick 定时器



5. 单击 Project Resources 下的 **Generate**，如 SAM E70 Xplained Ultra 评估工具包一节的步骤 9 所示。

5.4.2.1. 应用程序逻辑

1. 在 main()函数外添加以下宏和变量：

```
#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;
```

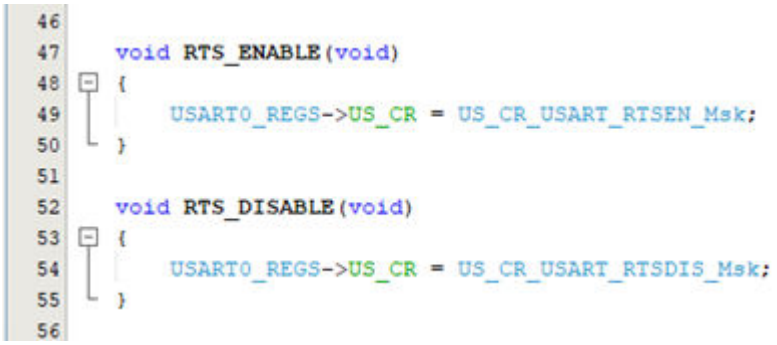
图 5-57. 添加宏和变量

```
24
25  #include <stddef.h>                    // Defines NULL
26  #include <stdbool.h>                   // Defines true
27  #include <stdlib.h>                    // Defines EXIT_FAILURE
28  #include "definitions.h"               // SYS function prototypes
29
30  #define RX_BUFFER_SIZE 1
31  #define TX_BUFFER_SIZE 1
32
33  // *****************************************************************************
34  // *****************************************************************************
35  // Section: Main Entry Point
36  // *****************************************************************************
37  // *****************************************************************************
38  volatile bool USART1_writeStatus = false;
39  volatile bool USART1_readStatus = false;
40
41  volatile bool USART0_writeStatus = false;
42  volatile bool USART0_readStatus = false;
43
44  uint8_t rxBuffer;
45  uint8_t txBuffer = 0xAA;
46
```

2. 在 main()函数外添加 RTS_ENABLE 和 RTS_DISABLE 函数以手动控制 RTS 线：

```
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;
}
```

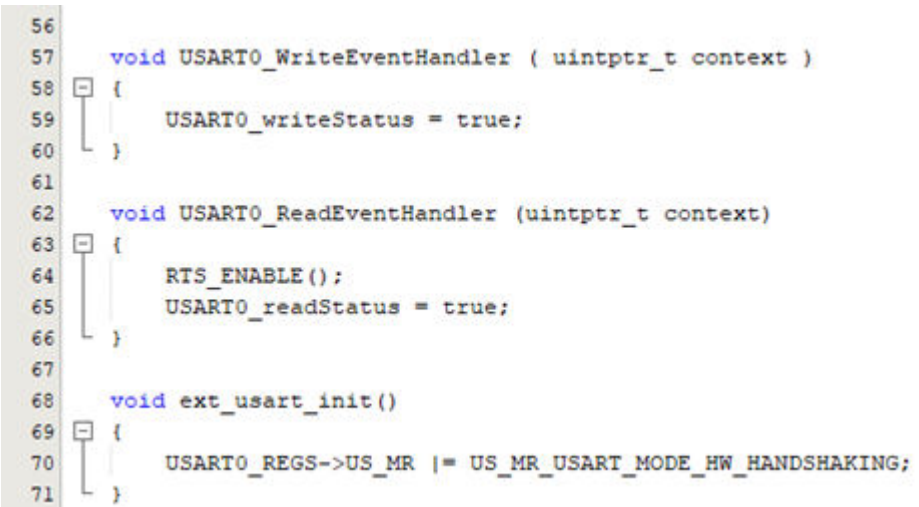
图 5-58. 添加 RTS 函数



3. 在 main()函数外添加事件处理程序并使能硬件握手模式：

```
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;
}
```

图 5-59. 添加事件处理程序



4. 在 main()函数内部调用必要的函数和回调寄存器：

```
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);
```

图 5-60. 模块初始化

```
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. 在 main()函数的 `while` 循环内添加硬件握手配置逻辑：

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

//发送从 EDBG 接收的字节
USART0_Write(&txBuffer, TX_BUFFER_SIZE);
}

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

//接收 EDBG 发送的字节
RTS_DISABLE();
USART0_Read(&rxBuffer, RX_BUFFER_SIZE);
}
```

图 5-61. 添加应用程序逻辑

```
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 }
```

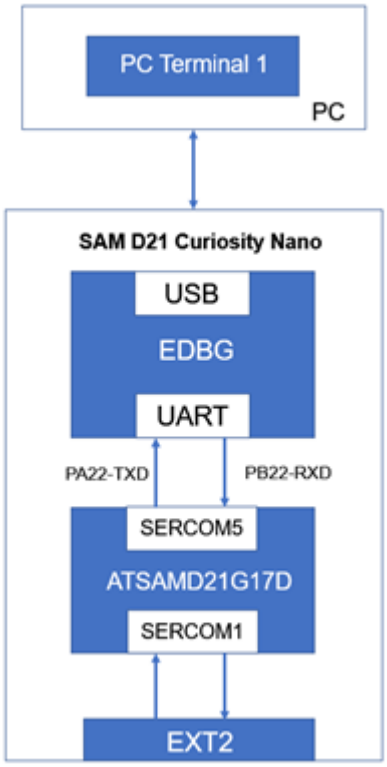
图 5-62. SAM E70 Xplained Ultra——硬件握手输出

5.5. SOF 检测和唤醒配置

当检测到起始位时，USART 帧起始（Start-Of-Frame，SOF）检测器可从待机休眠模式下唤醒 MCU。在待机休眠模式下，必须选择内部快速启动振荡器作为 GCLK_SERCOMx_CORE 源。应用程序进入待机休眠模式，PC 按键按下字符将设备从休眠模式唤醒并在终端上显示字符。

注：单击此处下载此应用程序配置的源代码。此外，也可在 GitHub 的 reference_apps 资源库中获取该代码。

图 5-63. 框图（SAM D21 Curiosity Nano 评估工具包）

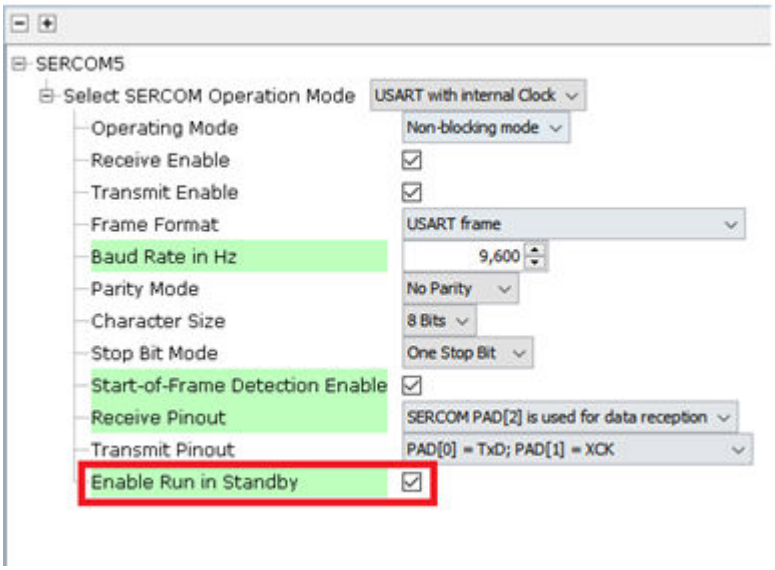


此应用程序仅使用一个通过 EDBG 连接到 PC 终端的 SAM D21 Curiosity Nano 评估工具包。本节仅介绍 SOF 检测和唤醒配置部分，其余内容在基本配置一节中介绍。

要使用 MCC 添加和配置 MPLAB Harmony 组件，请按照以下步骤操作：

1. 要创建项目，请参见 SAM D21 Curiosity Nano 评估工具包。
2. 单击并展开 **SERCOM5**，选择 **Enable Run in Standby**（使能在待机模式下运行），然后单击 **Generate**。

图 5-64. 配置选项



3. 在 main()函数外添加以下代码：

```
#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. 在以下代码示例中，当写函数（在 APP_SERCOM_5_Writecallback()中执行）完成时，tx_done 将变为 true；当读函数（在 APP_SERCOM_5_Raedcallback()中执行）完成时，rx_done 将变为 true。此外，如果 RXS_Msk 和 SERCOM_INTFLAG 的值等于 RXS_Msk，则需设置 RXS 标志。函数 usart_send_string()用于向控制台发送字符串。

图 5-65. SOF 检测与唤醒配置的实现——例 1

```
#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));
}
```

5. 在 main()函数内添加以下代码：

```
/* 初始化所有模块 */
    SYS_Initialize ( NULL );
    SERCOM5_REGS->USART_INT.SERCOM_INTENSET =  (uint8_t)SERCOM_USART_INT_INTENSET_RXS_Msk;

    // EDBG SERCOM 读写回调
    SERCOM5_USART_ReadCallbackRegister(APP_SERCOM_5_ReadCallback, 0);
    SERCOM5_USART_WriteCallbackRegister(APP_SERCOM_5_WriteCallback, 0);

    // EDBG 的读请求
    SERCOM5_USART_Read(&edbg_rx_data, RX_BUFFER_SIZE);
```

6. 在下图中，设置特定的 RXS 中断并调用回调寄存器，如基本配置中所示。此外，还会发出读请求。

图 5-66. SOF 检测与唤醒配置的实现——例 2

```
int main ( void )
{
    /* 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);
```

注：在 SOF 检测和唤醒配置中仅使用 SERCOM5（EDBG）。

7. 将以下代码添加到 `while` 循环内：

```
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));

            // 进入待机休眠模式。
            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. 在下图中，当输入字符时，器件将从休眠模式唤醒并输出输入的字符。在 TXC 为 false 期间（即发送完成前），程序将持续执行 while 循环本身。当 TXC 变为 true 时（即发送完成后），器件将进入待机休眠模式。

图 5-67. SOF 检测与唤醒配置的实现——例 3

```
98      while ( true )
99      {
100         if (!rx_done)
101         {
102             tx_done = false;
103             usart_send_string("\r\n Device entered into standby sleep mode");
104             while(!(SERCOM5_REGS->USART_INT.SERCOM_INTFLAG & SERCOM_USART_INT_INTFLAG_TXC_Msk));
105
106             // Enters standby sleep mode.
107             PM_StandbyModeEnter();
108         }
109         while(!rx_done);
110
111         tx_done = false;
112         usart_send_string("\r\n Character received after wakeup :");
113         while(!tx_done);
114
115         tx_done = false;
116         SERCOM5_USART_Write(&edbg_rx_data, RX_BUFFER_SIZE);
117         while(!tx_done);
118
119         rx_done = false;
120         SERCOM5_USART_Read(&edbg_rx_data, RX_BUFFER_SIZE);
121         SYS_Tasks ( );
122     }
123
124
125     return ( EXIT_FAILURE );
126 }
```

图 5-68. SERCOM USART——SOF 检测和唤醒配置输出

注：SAM E70/S70/V7x 系列器件未提供 SOF 检测和唤醒配置功能。

6. 参考资料

以下文档用于参考：

- 在 SAM D21 MCU 上使用 MPLAB Harmony v3 外设库入门
- 利用 FreeRTOS 在 SAM D21 MCU 上使用 MPLAB Harmony v3 驱动程序入门
- 在 SAM E70/S70/V70/V71 MCU 上使用 MPLAB Harmony v3 外设库入门
- 利用 FreeRTOS 在 SAM E70/S70/V70/V71 MCU 上使用 MPLAB Harmony v3 驱动程序入门
- SAM D21 Curiosity Nano 评估工具包
- SAM E70 Xplained Ultra 评估工具包
- SAM D21/DA1 Family Data Sheet（*DS40001882*）
- SAM D21 Curiosity Nano 用户指南（DS70005409D_CN）
- SAM E70/S70/V70/V71 Family Data Sheet（*DS60001527*）
- SAM E70 Xplained Ultra 用户指南（DS70005389B_CN）
- 有关 32 位单片机资料和解决方案的更多信息，请参见：32 位单片机相关资料和解决方案参考指南（DS70005534A_CN）
- 有关 MPLAB Harmony v3 的更多信息，请访问 Microchip 网站： www.microchip.com/en-us/tools-resources/configure/mplab-harmony 和 https:// developerhelp.microchip.com/xwiki/bin/view/software-tools/harmony/
- 有关各种应用程序的更多信息，请参见：github.com/Microchip-MPLAB-Harmony/reference_apps
- 有关更多信息，请访问 Microchip 网站或联系当地的 Microchip 销售代表。

7. 版本历史

7.1. 版本 A——2025 年 2 月

这是本文档的初始版本。

Microchip 信息

商标

“Microchip”名称和徽标、“M”徽标及其他名称、徽标和品牌均为 Microchip Technology Incorporated 或其关联公司和/或子公司在美国和/或其他国家/地区的注册商标和未注册商标（“Microchip 商标”）。有关 Microchip 商标的信息，可访问 https://www.microchip.com/en-us/about/legal-information/microchip-trademarks。

ISBN： 979-8-3371-1370-8

法律声明

提供本文档的中文版本仅为了便于理解。请勿忽视文档中包含的英文部分，因为其中提供了有关 Microchip 产品性能和使用情况的有用信息。Microchip Technology Inc.及其分公司和相关公司、各级主管与员工及事务代理机构对译文中可能存在的任何差错不承担任何责任。建议参考 Microchip Technology Inc.的英文原版文档。

本出版物及其提供的信息仅适用于 Microchip 产品，包括设计、测试以及将 Microchip 产品集成到您的应用中。以其他任何方式使用这些信息都将被视为违反条款。本出版物中的器件应用信息仅为您提供便利，将来可能会发生更新。您须自行确保应用符合您的规范。如需额外的支持，请联系当地的 Microchip 销售办事处，或访问 www.microchip.com/en-us/support/design-help/client-support-services。

Microchip“按原样”提供这些信息。Microchip 对这些信息不作任何明示或暗示、书面或口头、法定或其他形式的声明或担保，包括但不限于针对非侵权性、适销性和特定用途的适用性的暗示担保，或针对其使用情况、质量或性能的担保。

在任何情况下，对于因这些信息或使用这些信息而产生的任何间接的、特殊的、惩罚性的、偶然的或间接的损失、损害或任何类型的开销，Microchip 概不承担任何责任，即使 Microchip 已被告知可能发生损害或损害可以预见。在法律允许的最大范围内，对于因这些信息或使用这些信息而产生的所有索赔，Microchip 在任何情况下所承担的全部责任均不超出您为获得这些信息向 Microchip 直接支付的金额（如有）。如果将 Microchip 器件用于生命维持和/或生命安全应用，一切风险由买方自负。买方同意在由此引发任何一切损害、索赔、诉讼或费用时，会维护和保障 Microchip 免于承担法律责任。除非另外声明，在 Microchip 知识产权保护下，不得暗中或以其他方式转让任何许可证。

Microchip 器件代码保护功能

请注意以下有关 Microchip 产品代码保护功能的要点：

- Microchip 的产品均达到 Microchip 数据手册中所述的技术规范。
- Microchip 确信：在正常使用且符合工作规范的情况下，Microchip 系列产品非常安全。
- Microchip 注重并积极保护其知识产权。严禁任何试图破坏 Microchip 产品代码保护功能的行为，这种行为可能会违反《数字千年版权法案》（Digital Millennium Copyright Act）。
- Microchip 或任何其他半导体厂商均无法保证其代码的安全性。代码保护并不意味着我们保证产品是“牢不可破”的。代码保护功能处于持续发展中。Microchip 承诺将不断改进产品的代码保护功能。

产品页链接

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 和 PIC32CZ8110CA90208