PIC32CM LS00 MCU 上的软件攻击防护

AN5880

产品页链接

简介

现代嵌入式系统越来越容易受到软件攻击，这些攻击是指利用软件漏洞来获取未经授权的访问、窃取数据、破坏服务或造成其他形式损害的恶意活动。同时，保护知识产权仍然至关重要。

本文档提供了有关使用 PIC32CM LS00 Curiosity Nano+ Touch 评估工具包保护 PIC32CM LS00 MCU 免受软件攻击的设计指导。通过利用引导 ROM 的安全哈希算法 2（SHA-256）身份验证，PIC32CM LS00 可以识别非安全存储器中未经授权的代码片段，并将其替换为来自安全存储器区域的相同代码的可信副本。

目录

1. 硬件和软件要求

1.1. PIC32CM LS00 Curiosity Nano+ Touch 评估工具包

PIC32CM LS00 Curiosity Nano+ Touch 评估工具包非常适合用于对安全、超低功耗的 PIC32CM LS00 Arm® Cortex®-M23 单片机进行评估和原型设计。该 MCU 集成了 Arm TrustZone®技术和增强型外设触摸控制器（Peripheral Touch Controller，PTC）以及智能模拟功能，例如运放、ADC、DAC 和模拟比较器。

该工具包包括一个板上 Nano 嵌入式调试器（nEDBG），无需借助外部工具即可进行编程或调试。PIC32CM LS00 MCU 的主要特性如下：

- 48 MHz Arm Cortex-M23 内核
- 512 KB 闪存和 64 KB SRAM
- 不可变的安全引导、加密加速器和防篡改检测

PIC32CM LS00 Curiosity Nano+ Touch 评估工具包可从 Microchip 直销网站获取。

1.2. MPLAB® X 集成开发环境（IDE）和 MPLAB XC 编译器

MPLAB X IDE 是一款可扩展且高度可配置的软件程序，包含多种功能强大的工具，可用于发现、配置、开发、调试和验证大多数 Microchip 单片机的嵌入式设计。

- MPLAB X IDE 可从 Microchip 网站获取。本文档介绍了 MPLAB X IDE 版本 6.20。
- MPLAB XC 编译器可从 Microchip 网站获取。本文档介绍了 MPLAB XC32 版本 4.45。

1.3. MPLAB Harmony v3

MPLAB Harmony v3 是一款完全集成的软件开发框架，提供灵活且可互操作的软件模块，使开发人员能够将精力集中于创建 32 位 PIC®和 SAM 器件应用程序的开发，而无需耗费大量时间处理器件细节、复杂通信协议以及多种软件库的集成问题。

它包括 MPLAB 代码配置器（MPLAB Code Configurator，MCC），这是一种易于使用的开发工具，带有图形用户界面（Graphical User Interface，GUI），可显著简化器件初始化、软件库选择、系统配置以及应用程序开发过程。MCC 可作为插件与 MPLAB X IDE 集成，同时也提供独立的 Java 可执行程序，可在其他开发环境中单独运行和使用。

本文档中讨论的应用程序使用以下 MPLAB Harmony v3 资源库。这些资源库可从 GitHub 下载：

- csp v3.20.0 或更高版本（MPLAB Harmony v3 芯片支持包）
- bootloader v3.7.0
  或
- 使用 MCC 内容管理器下载上述资源库

2. 利用 PIC32CM LS00 MCU 防止软件攻击

2.1. 引导 ROM 功能

PIC32CM LS00/LS60 系列集成了硬件或软件加密加速器（CRYA），通过一套标准化 API 接口，支持以下加密与认证功能：高级加密标准（Advanced Encryption Standard，AES）加密和解密、安全哈希算法 2（SHA-256）身份验证以及 Galois 计数器模式（Galois Counter Mode，GCM）加密和身份验证。

CRYA 加密加速器在 IOBUS 端口上配置为客户端，并由 CPU 通过存储在引导 ROM 中的汇编代码控制。

高级加密标准（AES）遵循美国联邦信息处理标准（Federal Information Processing Standard，FIPS）出版物 197 号规范。AES 以 128 位数据块处理数据。AES 密码的密钥长度决定了将输入明文转换为最终输出（称为密文）所需的变换轮数。AES 采用对称密钥算法，这意味着加密和解密使用相同的密钥。

SHA-256 是一种加密哈希函数，它从数据块生成 256 位哈希值，该数据块以 512 位块为单位进行处理。

Galois/计数器模式（GCM）是 AES 的一种工作模式，它将计数器（CTR）模式与身份验证哈希函数相结合。

2.2. 安全哈希算法 2（SHA-256）身份验证

哈希函数的主要用途是为特定的数据集生成独有的数字标识符，类似于指纹。与错误检测码不同，每个数据集都必须与惟一的标识符关联。

从实际角度看，哈希函数接收不同长度的输入并生成固定大小的输出（称为报文摘要）。报文摘要具有几个重要属性，包括优秀的扩散性，可确保即使输入发生微小的变化，输出也会产生显著差异。

尽管哈希函数的输出大小固定，这在理论上限制了为每个可能的数据生成惟一摘要的能力，但其设计使得找到两个能产生相同摘要的报文变得极其困难，从而在实际应用中有效创造了惟一性的表象。

2.3. 使用引导 ROM 中的 SHA-256 API

加密加速器（CRYA）API 位于专用的引导 ROM 区域。该区域是仅执行的，这意味着 CPU 不能进行任何装载，但可以调用 API。引导 ROM 存储空间是一个安全区域，只有安全应用程序才能直接调用这些 API。

表 2-1. CRYA API 地址

CRYA API	地址
AES Encryption	0x02006804
AES Decryption	0x02006808
SHA256 Init	0x02006810
SHA256 Update	0x02006814
SHA256 Final	0x02006818
SHA256 Process（传统 API）	0x02006800
GCM Process	0x0200680C

API 由以下函数组成，必须按特定顺序调用：

1. SHA-256 Init 用于初始化 SHA256_CTX 结构体。
2. SHA-256 Update 用于添加要在摘要中计算的报文。
3. SHA-256 Final 用于计算摘要。

注：如果要将多条报文包含在摘要计算中，可以多次调用 SHA-256 Update。

要定义的 SHA-256 结构体称为 SHA56_CTX：

```
typedef struct
{
    /* Digest result of SHA256 */
```

```
    uint32_t digest[8];
    /* 报文长度 */
    uint64_t length;
    /* 存放数据剩余部分的大小 */
    uint32_t remain_size;
    /* 数据剩余部分的缓冲区（512 位数据块） */
    uint8_t remain_ram[64];
    /* crya_sha_process 使用的 256 字节 RAM 缓冲区 */
    uint32_t process_buf[64];

} SHA256_CTX;
```

SHA-256 Init 函数入口点位于引导 ROM 地址 0x02006810：

```
typedef void (*crya_sha256_init_t) (SHA256_CTX *context);

#define crya_sha256_init ((crya_sha256_init_t) (0x02006810 | 0x1))
```

SHA-256 Update 函数入口点位于引导 ROM 地址 0x02006814：

```
typedef void (*crya_sha256_update_t) (SHA256_CTX *context, const unsigned char *data, size_t
length);

#define crya_sha256_update ((crya_sha256_update_t) (0x02006814 | 0x1))
```

SHA-256 Final 函数入口点位于引导 ROM 地址 0x02006818：

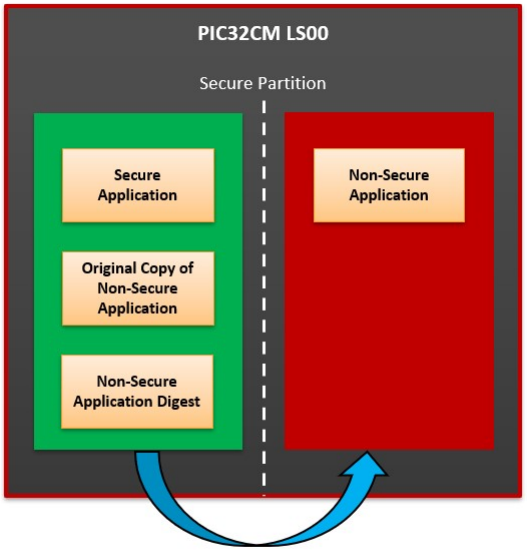
```
typedef void (*crya_sha256_final_t) (SHA256_CTX *context, unsigned char output[32]);

#define crya_sha256_final ((crya_sha256_final_t) (0x02006818 | 0x1))
```

2.4. 防止非安全区域受到软件攻击

在器件启动期间，安全应用程序会为非安全固件计算一个唯一标识符（摘要），并将其存储在安全存储器（安全数据闪存）中。必须执行定期验证以确保非安全固件的完整性。定时器将以特定的时间间隔检查固件的真实性。

如果将恶意软件或未经授权的代码注入到非安全应用程序中，更新后固件的计算摘要将与真正的非安全应用程序的预期摘要不匹配。因此，安全应用程序将从安全存储器中把原始副本恢复到非安全闪存区域，从而防止系统停机。

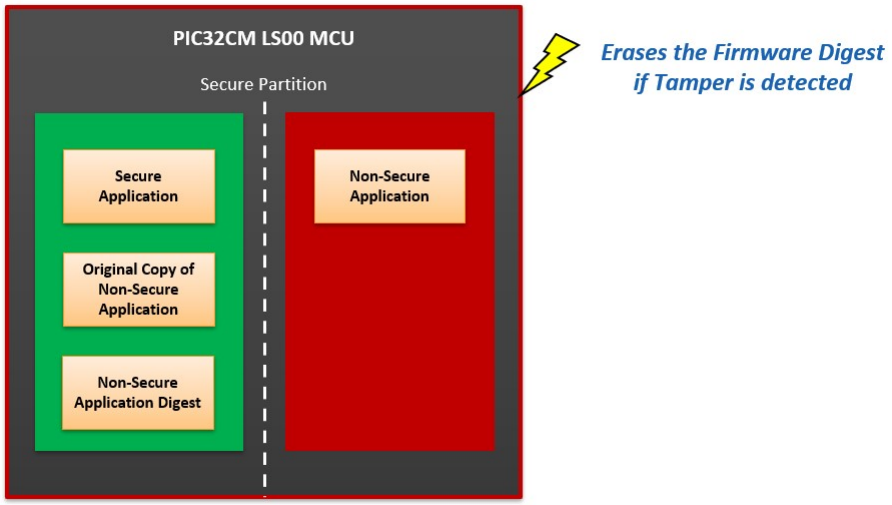
图 2-1. 非安全固件的软件攻击保护



2.4.1. 软件攻击模拟

PIC32CM LS00/LS60 系列器件在实时时钟（Real Time Clock，RTC）外设中具有带防篡改擦除安全功能的防篡改检测功能。为了模拟软件攻击，采用防篡改擦除选项来擦除存储在安全存储器区域中的数据。在检测到任何篡改时，安全应用程序中的 RTC 外设会触发防篡改擦除操作，以删除安全数据闪存区域中的内容（固件摘要）。

图 2-2. 使用防篡改检测的软件攻击模拟

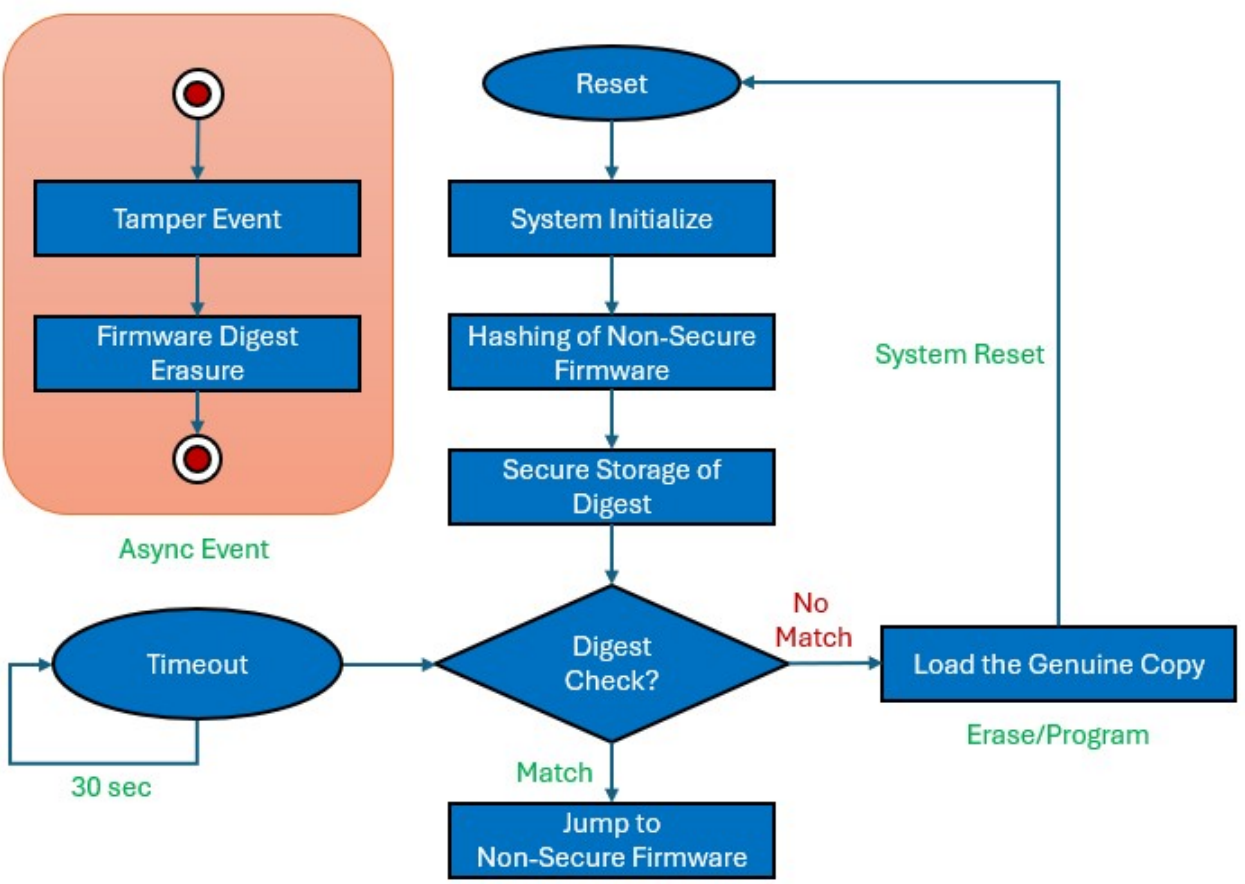


2.4.2. 执行流程

安全固件执行流程

下图说明了软件攻击保护应用程序中安全固件的系统级执行流程。

图 2-3. 安全应用程序执行流程



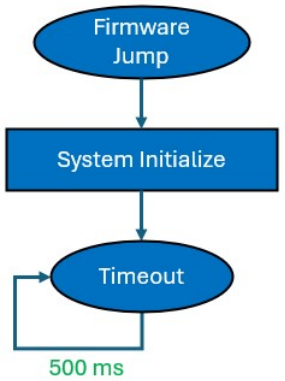
安全应用程序按以下顺序执行：

1. 系统复位后，应用程序将启动非安全固件哈希处理过程以生成固件摘要。
2. 计算出的摘要存储在安全区域内的数据闪存存储器中。
3. 非安全应用程序的固件摘要将与安全闪存存储器中的真实副本进行对比验证。
4. 验证成功后，执行将跳转到非安全应用程序。
5. 如果验证失败，则会擦除非安全应用程序，并将真实副本装入非安全闪存区域。
6. 每 30 秒重新生成一次固件摘要，并与真实副本进行交叉验证，以确保真实性。

非安全固件执行流程

下图说明了软件攻击保护应用程序中非安全固件的系统级执行流程。

图 2-4. 非安全应用程序执行流程



非安全应用程序按以下顺序执行：

1. 从安全应用程序进行固件跳转后，非安全固件将初始化非安全外设。
2. 在 PIC32CM LS00 Curiosity Nano+ Touch 评估工具包上每 500 毫秒切换一次 LED1。

软件攻击模拟执行流程

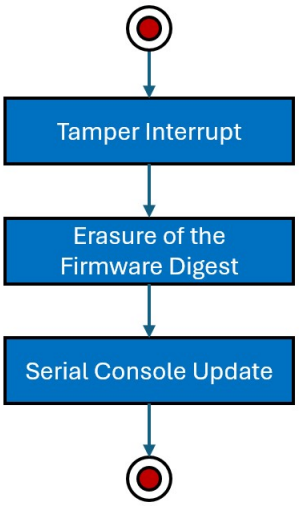
软件攻击的模拟按以下顺序进行：

1. 按下 PIC32CM LS00 Curiosity Nano+ Touch 评估工具包上的 SW1 按钮，通过触发篡改事件来模拟软件攻击。
2. 在篡改事件之后，RTC 启动数据闪存内容擦除过程。
3. 在 RTC 篡改处理程序中，一条指示软件攻击启动的报文会发送到串行控制台。

注：此执行发生在安全固件的 RTC 中断处理程序内。

下图说明了安全固件中软件攻击的系统级执行流程。

图 2-5. 软件攻击执行流程

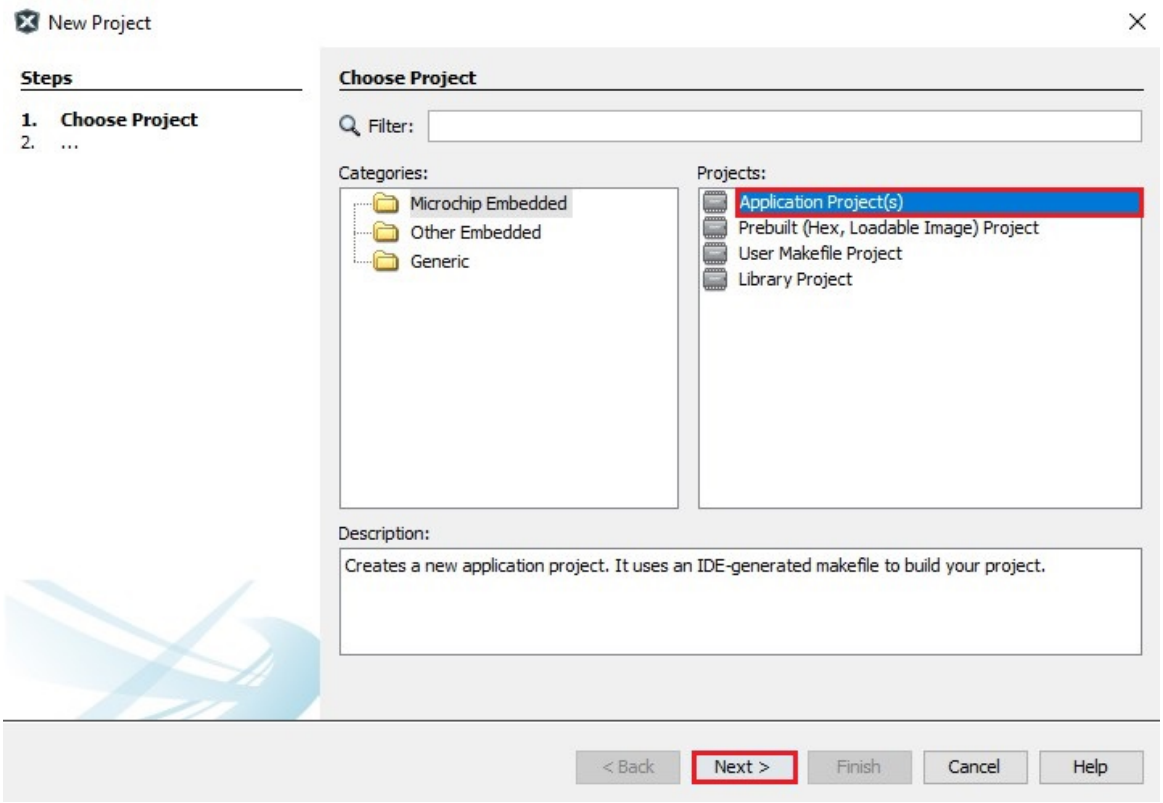


3. 使用 MPLAB Harmony v3 和 MCC 在 PIC32CM LS00 Curiosity Nano+ Touch 评估工具包上实现软件攻击保护

要创建基于 MPLAB Harmony v3 的项目，请按照以下步骤操作，或者在此处下载预先开发的演示项目。

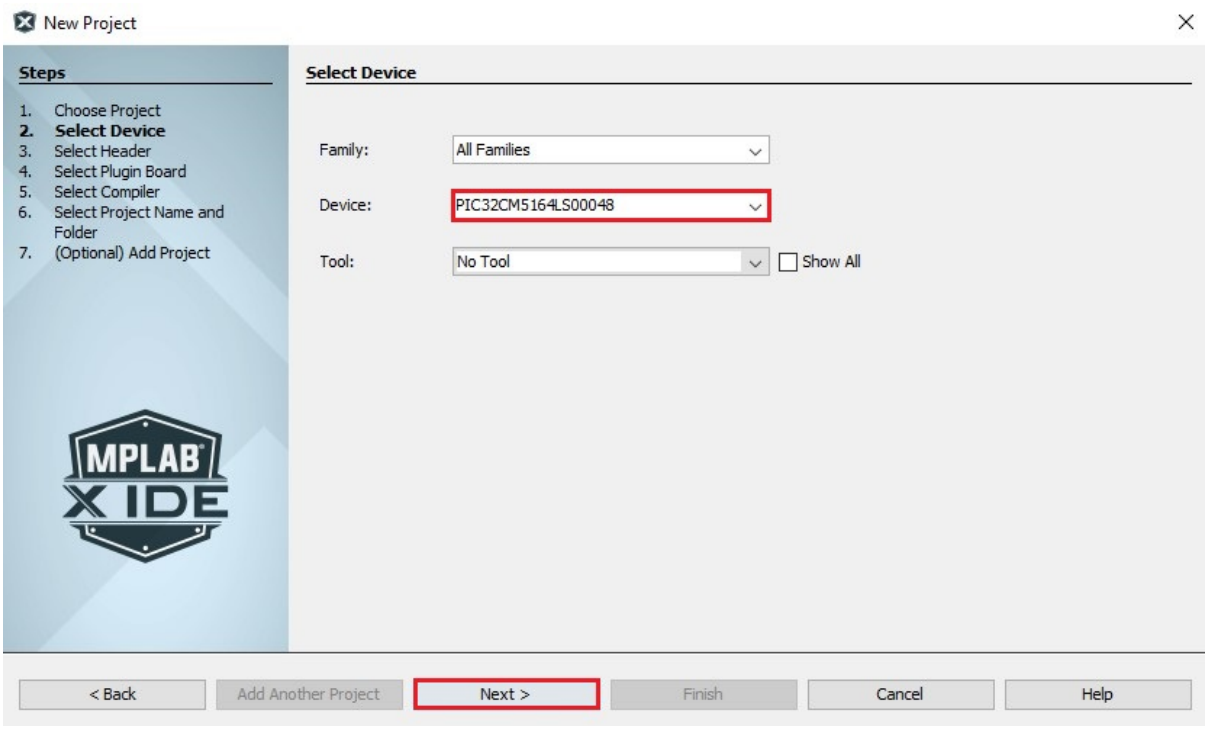
1. 从 **Start**（开始）菜单中启动 **MPLAB X IDE**。
2. MPLAB X IDE 打开后，在 File（文件）菜单中，单击 **New Project**（新建项目）或单击 *New Project* 图标。
3. 在 **New Project** 窗口左侧导航栏中的 Steps（步骤）下，选择 **Choose Project**（选择项目）。
4. 在右侧 Choose Project 属性页面中：
   a. 在 Categories（类别）下，选择 **Microchip Embedded**（Microchip 已安装工具）。
   b. 在 Projects（项目）下，选择 **Application Project**（应用程序项目）。

图 3-1. 新项目创建



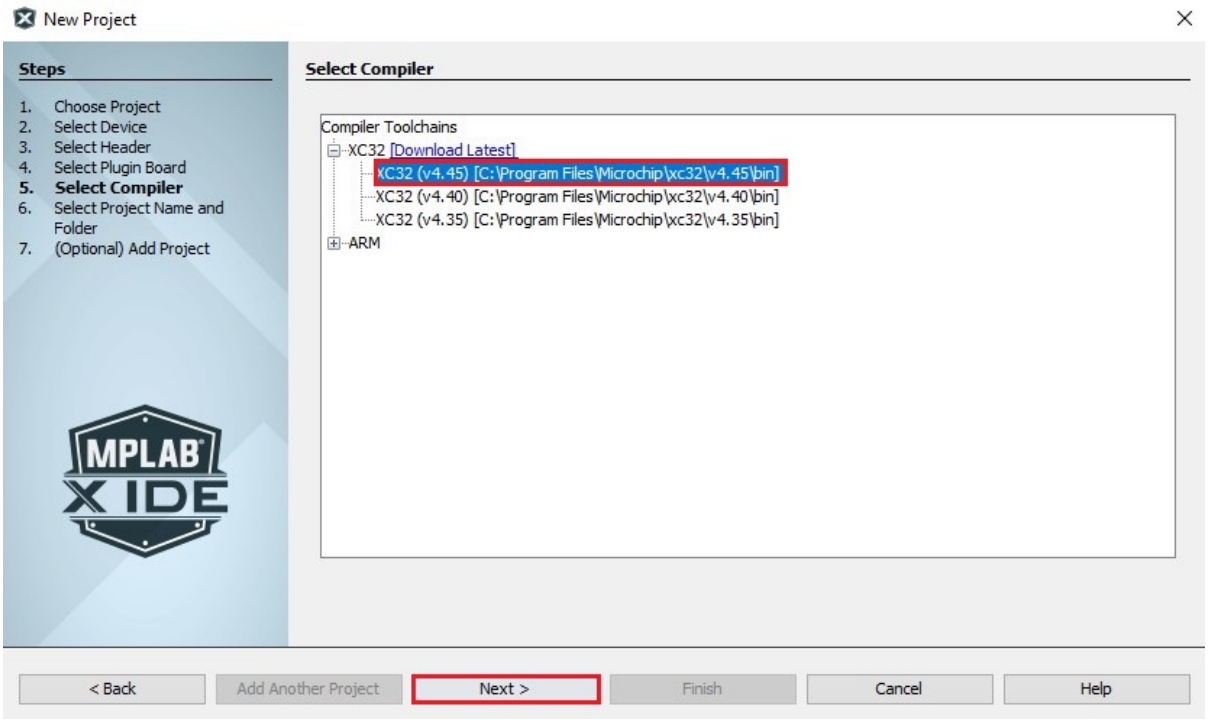
5. 单击 **Next**（下一步）。
6. 在 Steps 下，选择 **Select Device**（选择器件），然后从右侧的 Select Device Properties（选择器件属性）页面中，为 Device（器件）选择 **PIC32CM5164LS00048**，以在 PIC32CM LS00 Curiosity Nano+ Touch 评估工具包上创建项目（器件条目将反映在器件下）。

图 3-2. 器件选择



7. 单击 **Next**。
8. 选择 **Select Compiler**（选择编译器），接着从右侧的 Select Compiler 属性页面中单击并展开 XC32，然后选择 **XC32 Compiler**（XC32 编译器）。

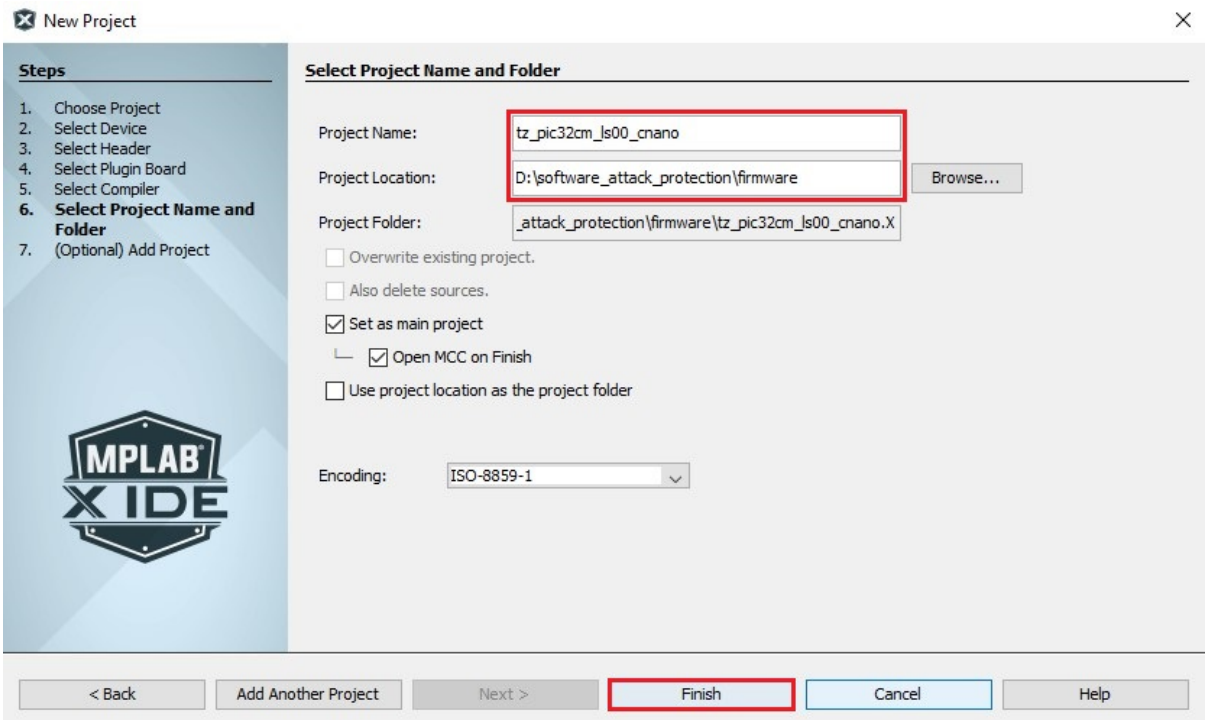
图 3-3. XC32 编译器选择



9. 单击 **Next**。

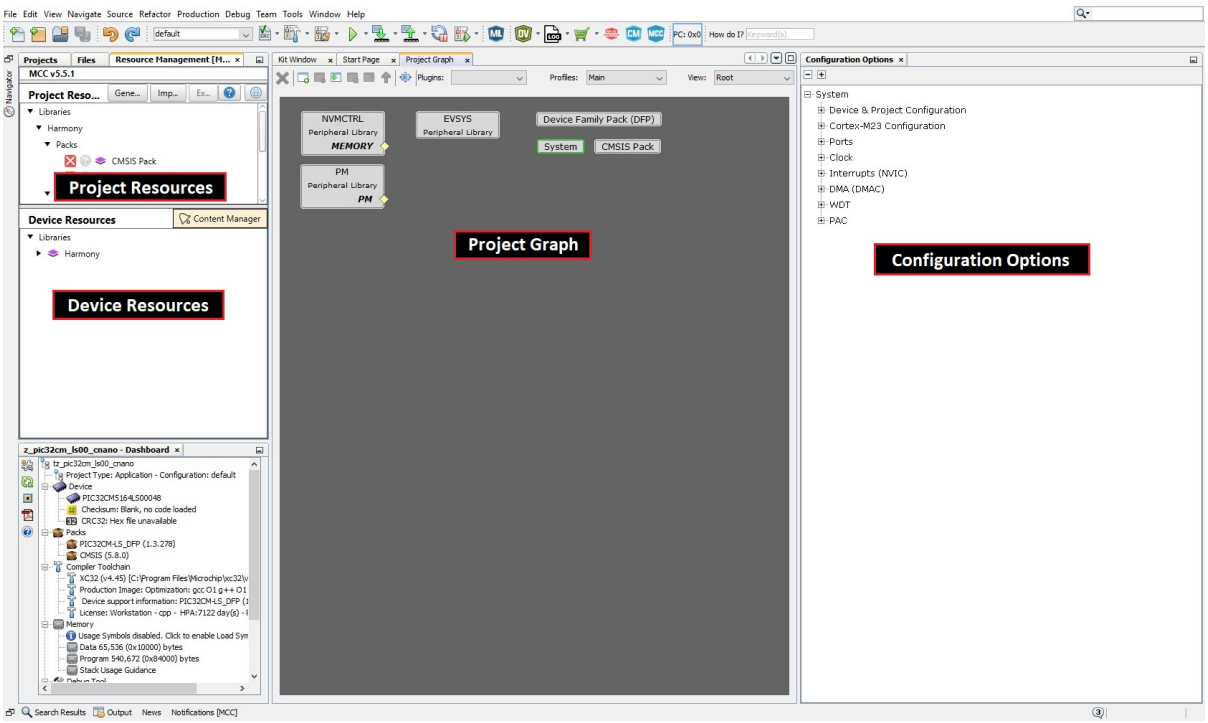
10. 选择 **Select Project Name and Folder**（选择项目名称和文件夹），然后在 Select Project Name and Folder 属性页面中输入以下详细信息：
    - Project Name（项目名称）：输入 *tz_pic32cm_ls00_cnano*（指示将在 MPLAB X IDE 中显示的项目名称，即设置项目的名称）。
    - Project Location（项目位置）：输入 *D:\software_attack_protection\firmware*（指示新项目根文件夹的路径）。所有项目文件均将置于该文件夹中。项目位置可以是任何有效路径。
    - Project Folder（项目文件夹）：只读内容（在用户更改上述条目时自动更新）。

图 3-4. 项目名称和文件夹设置



11. 单击 **Finish**（完成）以启动 MCC。
12. MCC 插件将在新窗口中打开，如下图所示：

图 3-5. MPLAB®代码配置器窗口

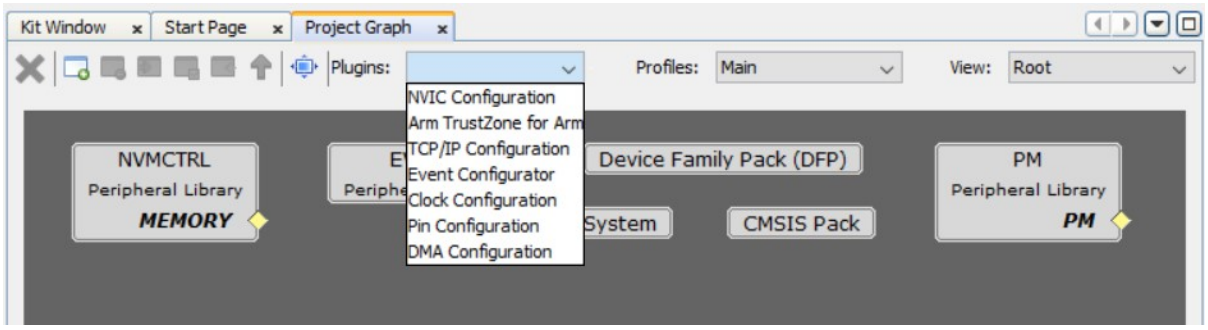


3.1. 添加和配置 MPLAB Harmony 组件

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

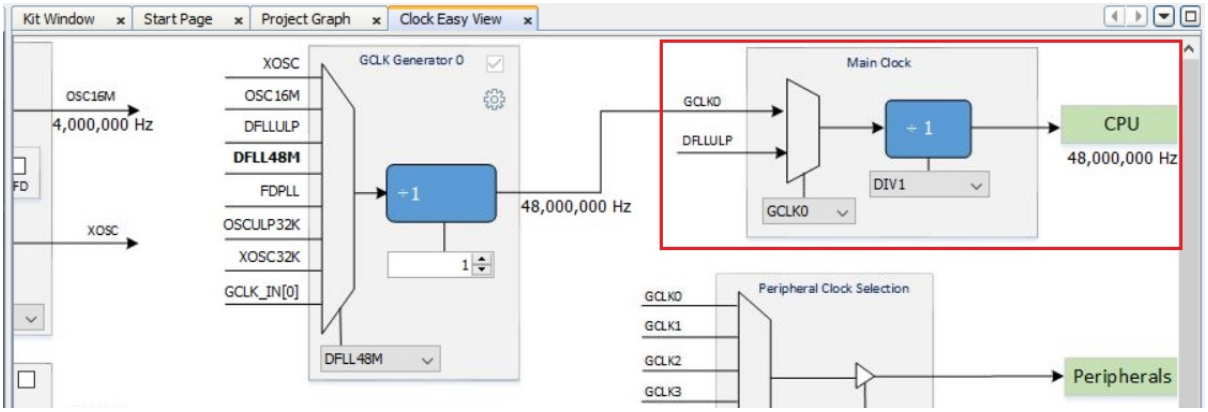
1. 在 MCC 窗口的 Plugins（插件）下拉列表中，选择所需的配置窗口。

图 3-6. MPLAB®代码配置器——插件列表



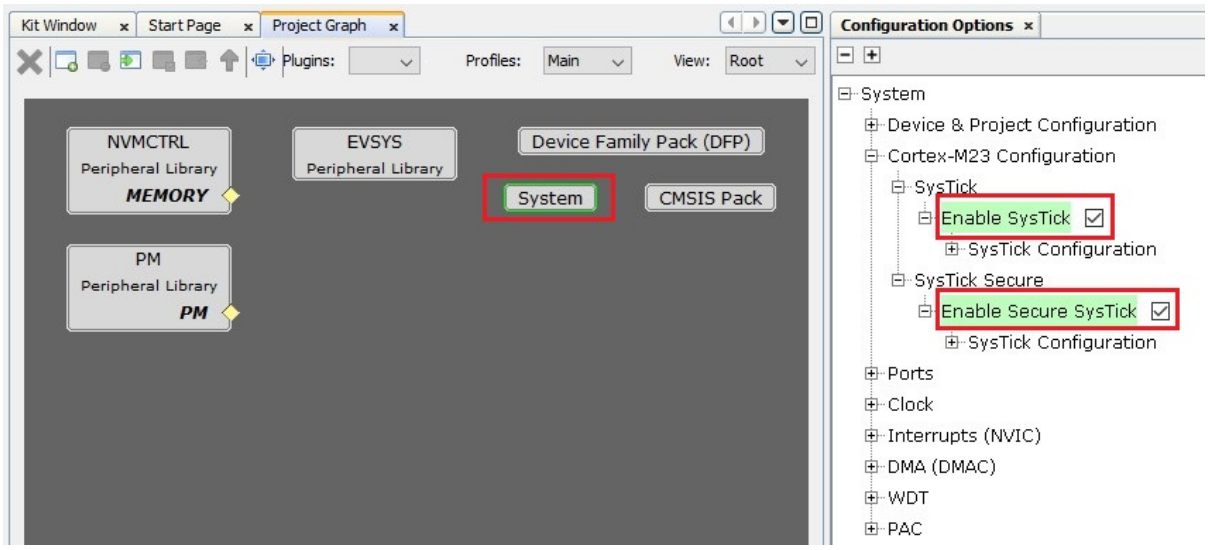
2. 选择 **Clock Configuration**（时钟配置）以打开 Clock Easy View（时钟简化视图）窗口，然后查看其中的主时钟是否已设置为 48 MHz。

图 3-7. MPLAB®代码配置器——GCLK 发生器 0



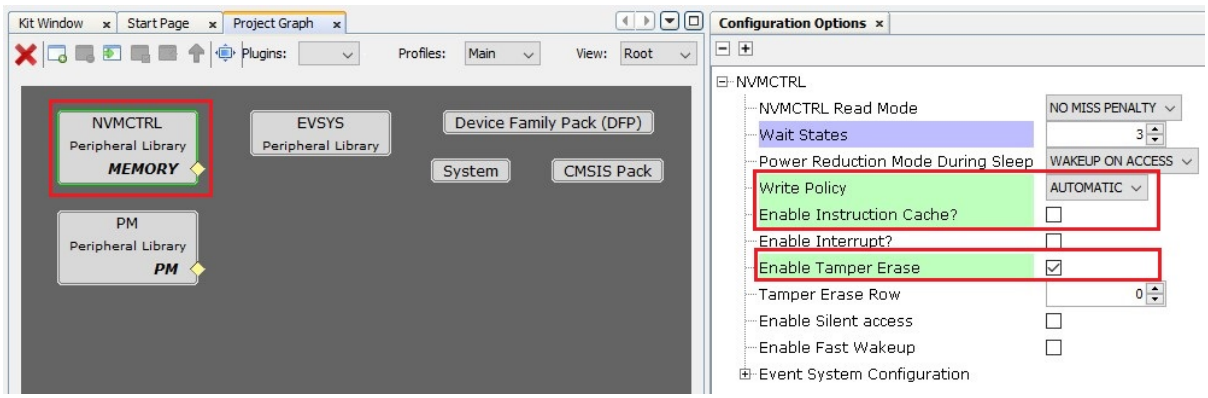
3. 单击 **Project Graph**（项目图），然后选择 **System**（系统）模块。在 Configuration Options（配置选项）属性页面中，进行如下配置以使能 SysTick 定时器创建安全和非安全延时。

图 3-8. MPLAB®代码配置器——SysTick 配置



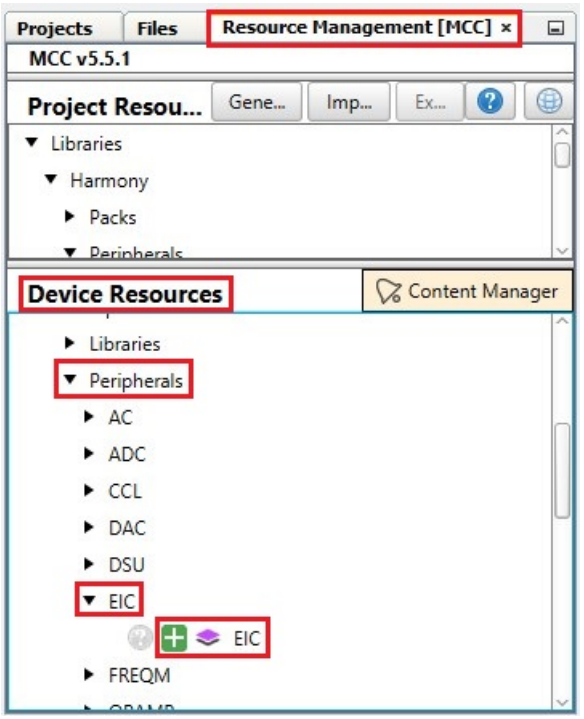
4. 选择 **NVMCTRL Peripheral Library MEMORY**（NVMCTRL 外设库存储器），然后在 Configuration Options 属性页面中，进行如下配置以使能防篡改擦除功能。

图 3-9. MPLAB®代码配置器——NVMCTRL 配置



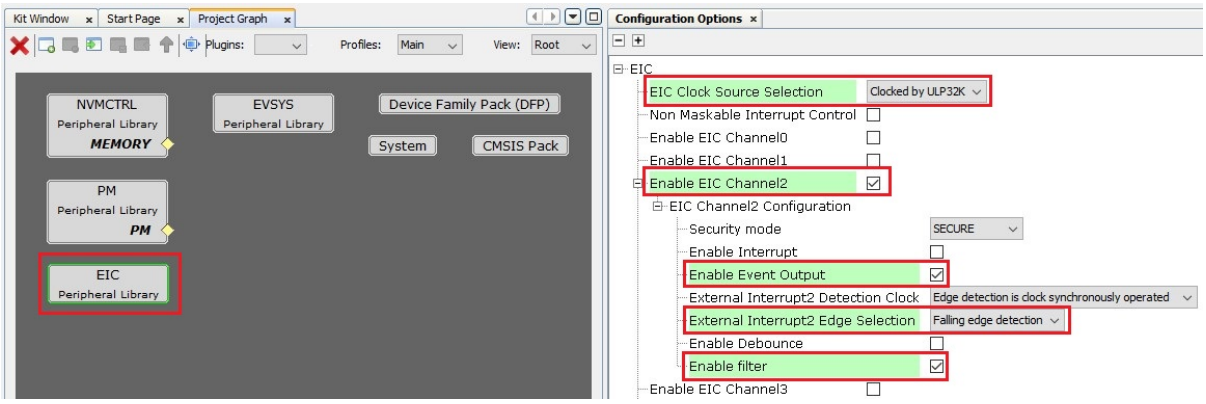
5. 单击 **Resource Management (MCC)**（资源管理（MCC）），然后在 Device Resources（器件资源）下，单击并展开 *Harmony* > *Peripherals* > *EIC*（Harmony > 外设 > EIC）。单击 **EIC**，随后可观察到 EIC 外设库模块已添加到 Project Graph 窗口中。

图 3-10. MPLAB®代码配置器——EIC 外设选择



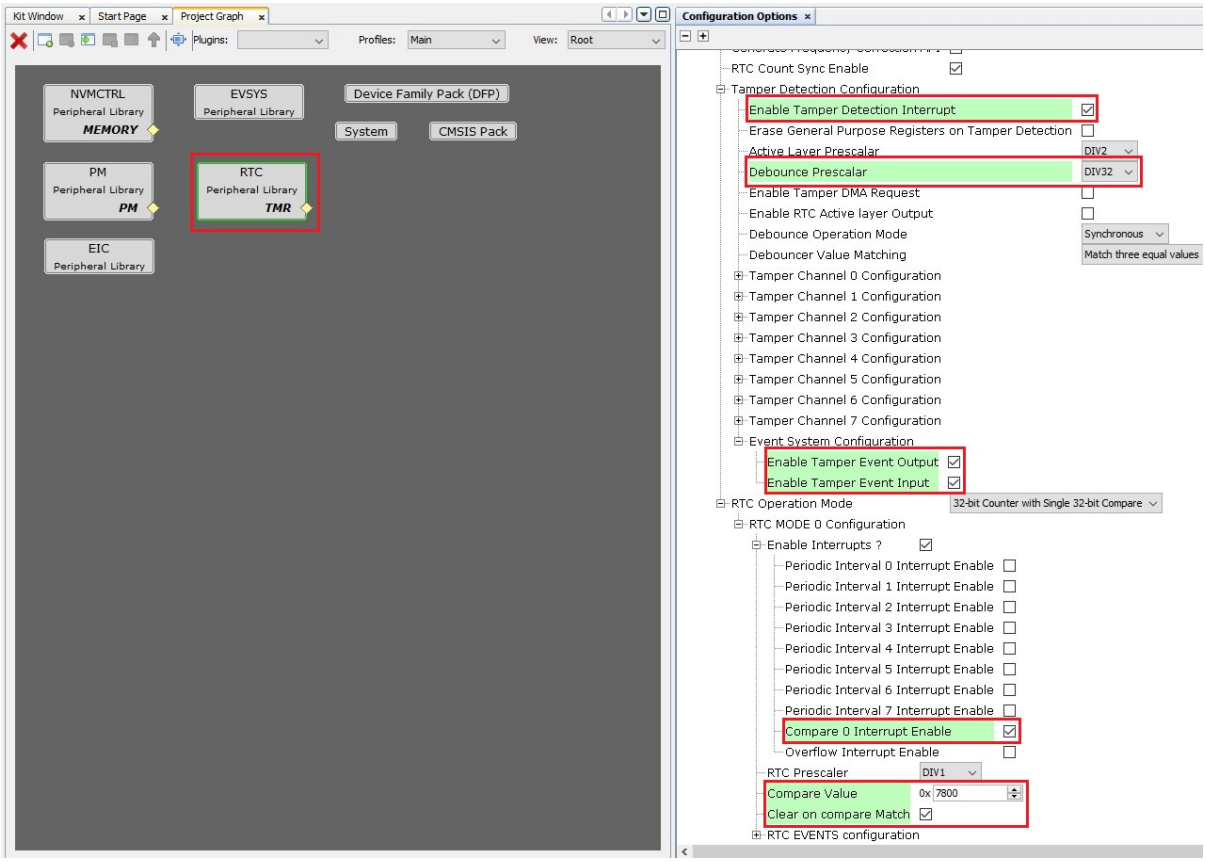
6. 选择 **EIC Peripheral Library**（EIC 外设库），然后在右侧的 Configuration Options 属性页面中进行如下配置，以将 EIC 通道 2（SW1）用作防篡改输入。

图 3-11. MPLAB®代码配置器——EIC 配置



7. 在 Device Resources 下，单击并展开 *Harmony* > *Peripherals* > *RTC*（Harmony > 外设 > RTC）单击 **RTC**，随后可观察到 RTC 外设库模块已添加到 Project Graph 窗口中。
8. 选择 **RTC Peripheral Library**（RTC 外设库），然后在 Configurations Options 属性页面中进行如下配置，以每 30 秒产生一次比较中断，并允许篡改中断和事件。

图 3-12. MPLAB®代码配置器——RTC 配置



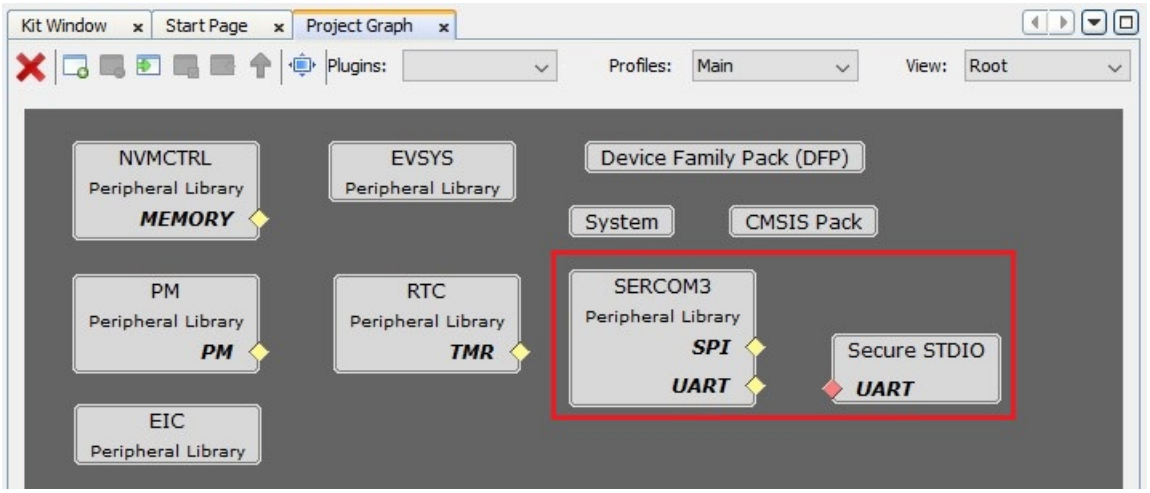
注：比较值设置为 0x7800。此值每 30 秒生成一次 RTC 比较中断。

- RTC 时钟 = 1024 Hz
- RTC 预分频比 = 1
- 所需中断速率 = 30s

因此，比较值 = 30 x 1024 = 30,720（即 0x7800）。

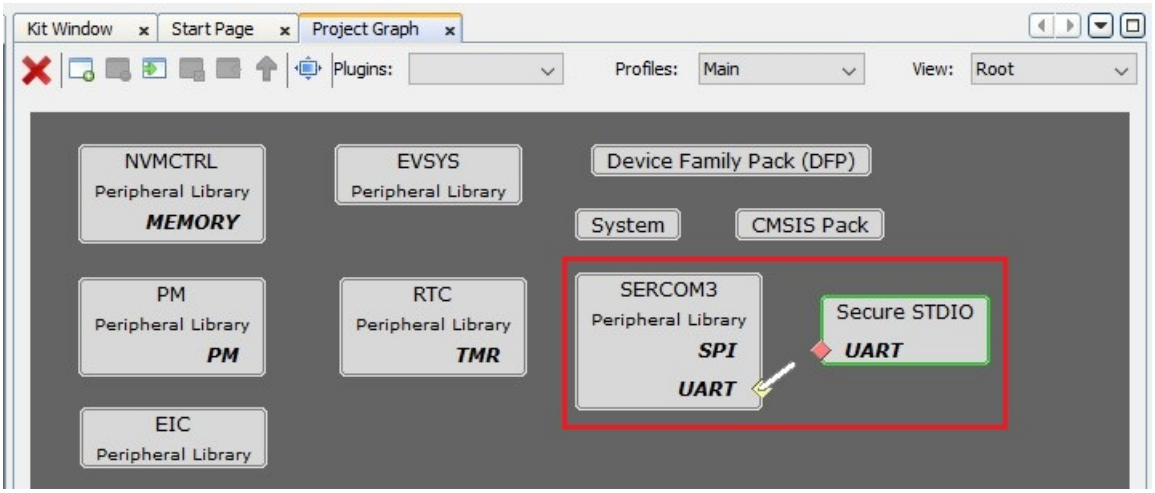
9. 在 Device Resources 下：
   a. 单击并展开 *Harmony* > *Peripherals* > *SERCOM*（Harmony > 外设 > SERCOM）。单击 **SERCOM3**，随后可观察到 SERCOM3 模块已添加到 Project Graph 窗口中。
   b. 单击并展开 *Harmony* > *Peripherals* > *Tools*（Harmony > 外设 > 工具）。单击 **Secure STDIO**，随后可观察到 Secure STDIO 模块已添加到 Project Graph 窗口中。

图 3-13. MPLAB®代码配置器——SERCOM 和 Secure STDIO 选择



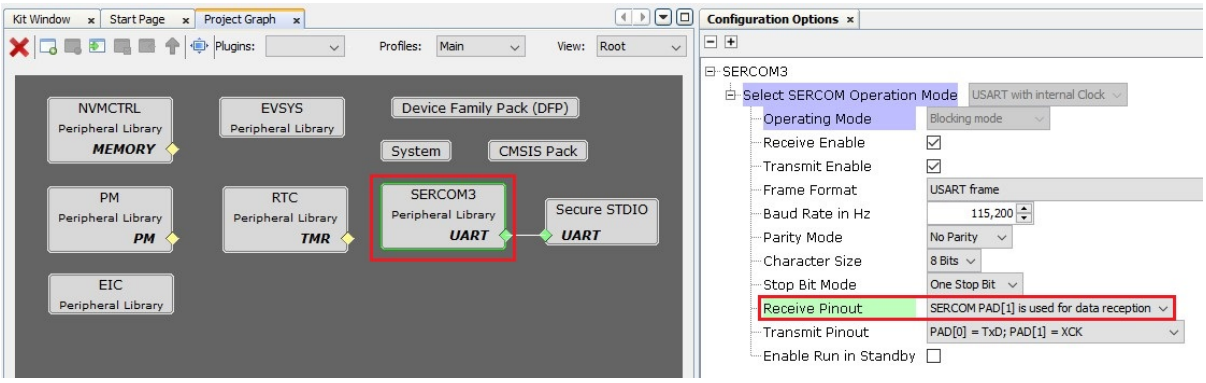
10. 通过将 UART 黄色菱形拖到 Secure STDIO 模块中的红色菱形上，连接 SERCOM3 与 Secure STDIO 模块。

图 3-14. MPLAB®代码配置器——SERCOM 和 Secure STDIO 选择



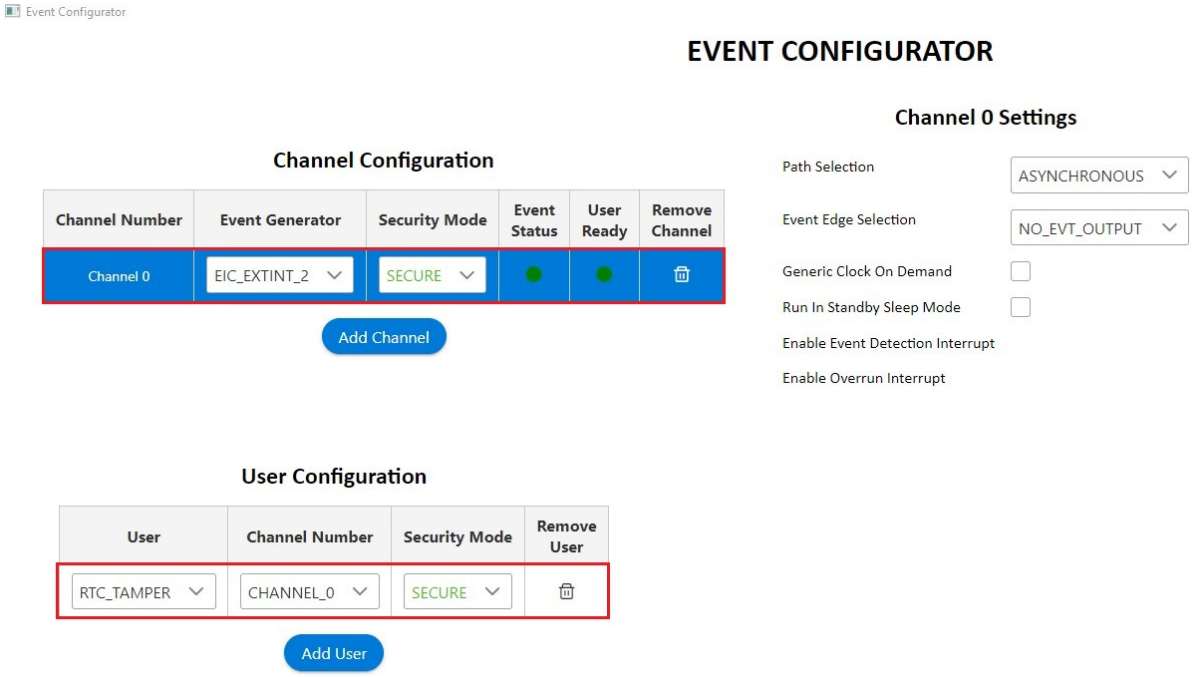
11. 在左侧窗格中，选择 **SERCOM3 Peripheral Library**（SERCOM3 外设库）。在 Configuration Options 属性页面中，进行如下配置以将数据按照 115200 的波特率打印在串行控制台上。

图 3-15. MPLAB®代码配置器——SERCOM3 配置



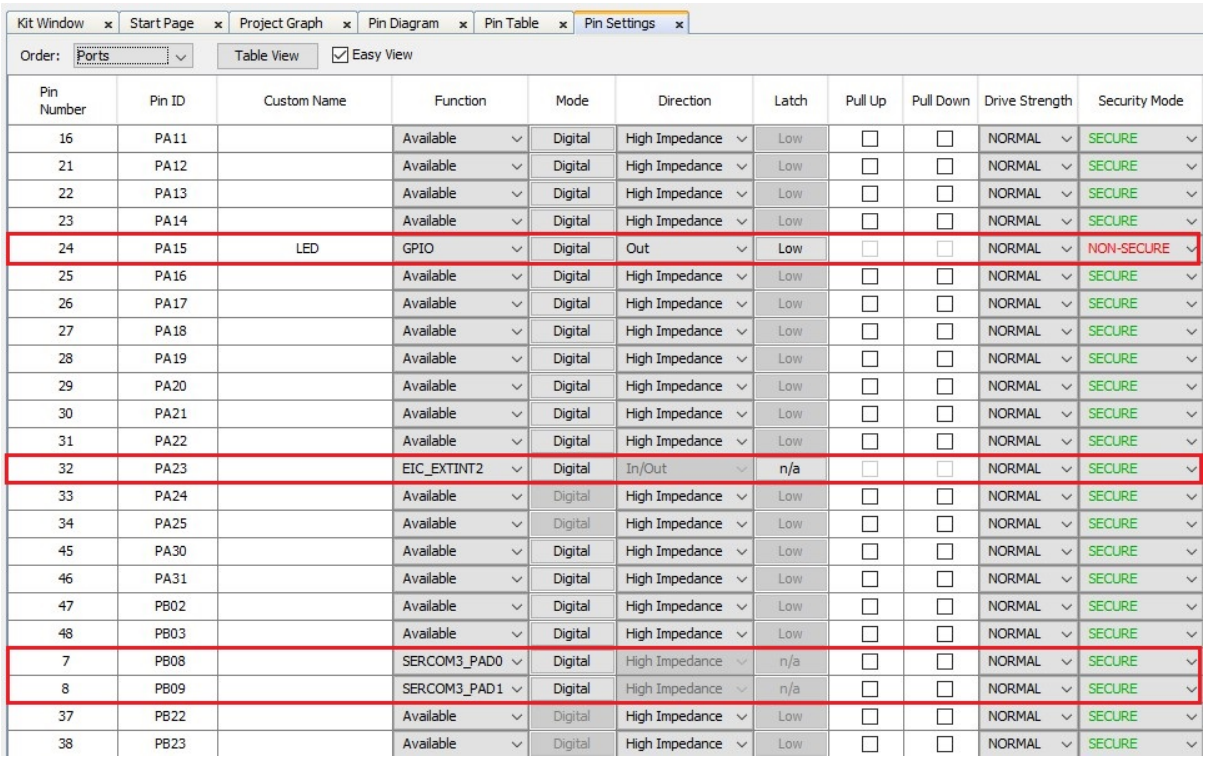
12. 从 **Plugins** 下拉列表中选择 *Event Configurator*（事件配置器）。添加防篡改输入的事件生成器和事件用户，如下图所示。

图 3-16. MPLAB®代码配置器——事件配置



13. 从 **Plugins** 下拉列表中，选择 **Pin Configuration**（引脚配置），然后单击 **Pin Settings**（引脚设置）选项卡。将顺序更改为 *Ports*（端口）。根据下面所示的应用程序进行引脚配置。

图 3-17. Pin Settings 窗口——引脚配置



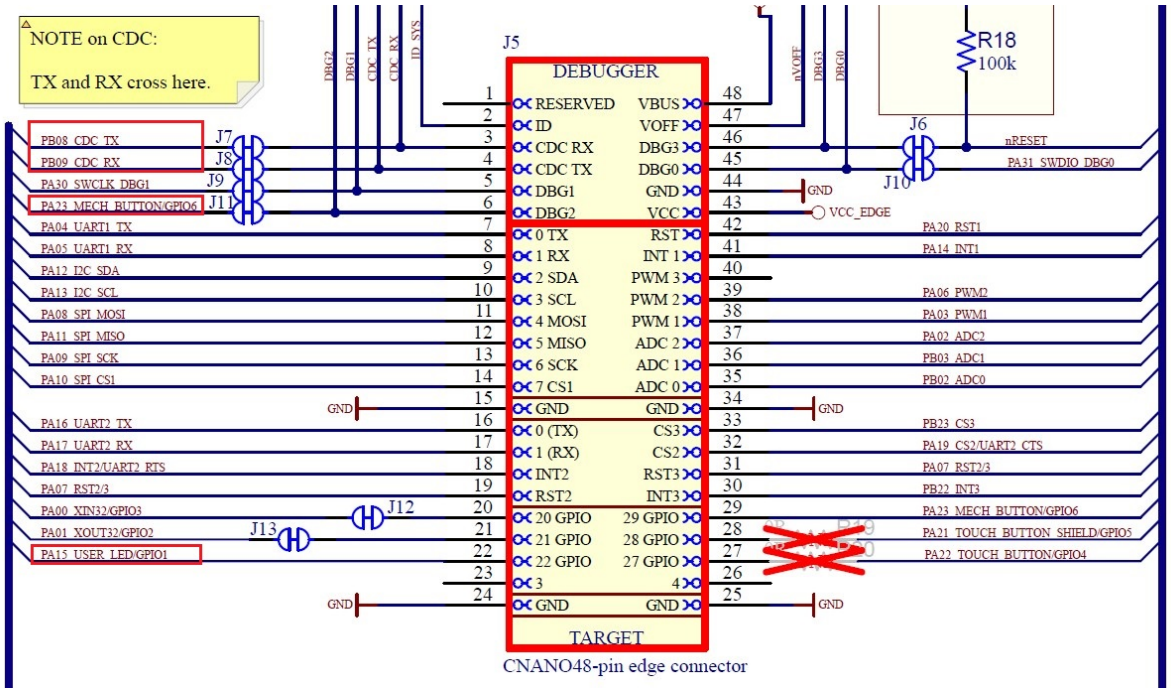
Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch	Pull Up	Pull Down	Drive Strength	Security Mode
16	PA11		Available	Digital	High Impedance	Low	☐	☐	NORMAL	SECURE
21	PA12		Available	Digital	High Impedance	Low	☐	☐	NORMAL	SECURE
22	PA13		Available	Digital	High Impedance	Low	☐	☐	NORMAL	SECURE
23	PA14		Available	Digital	High Impedance	Low	☐	☐	NORMAL	SECURE
24	PA15	LED	GPIO	Digital	Out	Low	☐	☐	NORMAL	NON-SECURE
25	PA16		Available	Digital	High Impedance	Low	☐	☐	NORMAL	SECURE
26	PA17		Available	Digital	High Impedance	Low	☐	☐	NORMAL	SECURE
27	PA18		Available	Digital	High Impedance	Low	☐	☐	NORMAL	SECURE
28	PA19		Available	Digital	High Impedance	Low	☐	☐	NORMAL	SECURE
29	PA20		Available	Digital	High Impedance	Low	☐	☐	NORMAL	SECURE
30	PA21		Available	Digital	High Impedance	Low	☐	☐	NORMAL	SECURE
31	PA22		Available	Digital	High Impedance	Low	☐	☐	NORMAL	SECURE
32	PA23		EIC_EXTINT2	Digital	In/Out	n/a	☐	☐	NORMAL	SECURE
33	PA24		Available	Digital	High Impedance	Low	☐	☐	NORMAL	SECURE
34	PA25		Available	Digital	High Impedance	Low	☐	☐	NORMAL	SECURE
45	PA30		Available	Digital	High Impedance	Low	☐	☐	NORMAL	SECURE
46	PA31		Available	Digital	High Impedance	Low	☐	☐	NORMAL	SECURE
47	PB02		Available	Digital	High Impedance	Low	☐	☐	NORMAL	SECURE
48	PB03		Available	Digital	High Impedance	Low	☐	☐	NORMAL	SECURE
7	PB08		SERCOM3_PAD0	Digital	High Impedance	n/a	☐	☐	NORMAL	SECURE
8	PB09		SERCOM3_PAD1	Digital	High Impedance	n/a	☐	☐	NORMAL	SECURE
37	PB22		Available	Digital	High Impedance	Low	☐	☐	NORMAL	SECURE
38	PB23		Available	Digital	High Impedance	Low	☐	☐	NORMAL	SECURE

注：

- PB08 和 PB09：SERCOM3 TX 和 RX 引脚
- PA15：LED
- PA23：开关

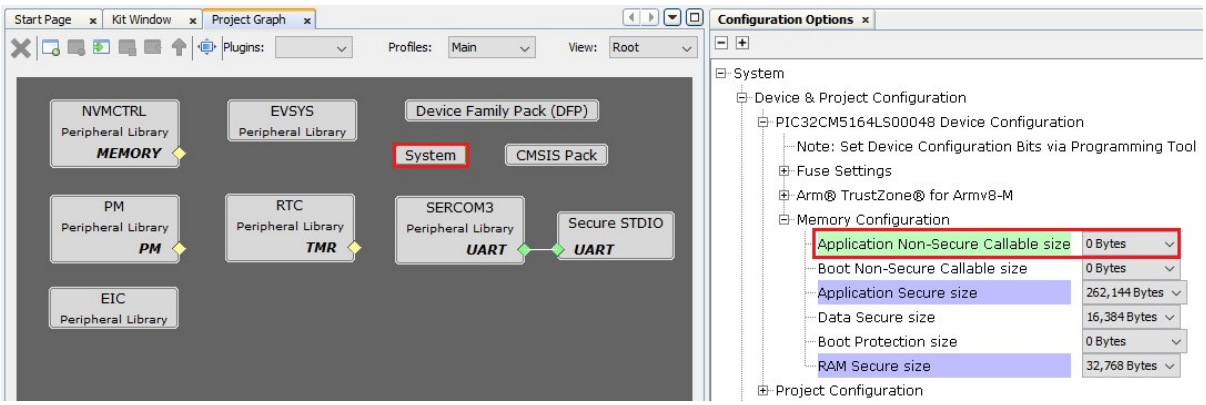
更多信息，请参见 PIC32CM LS00 Curiosity Nano+ Touch 评估工具包用户指南（*DS70005567B_CN*）。

图 3-18. PIC32CM LS00 Curiosity Nano+ Touch 开发板引脚分配



14. 单击项目图中的 **System**。在 Configuration Options 属性页面中，进行如下配置以将非安全可调用大小的存储器配置设置为零。

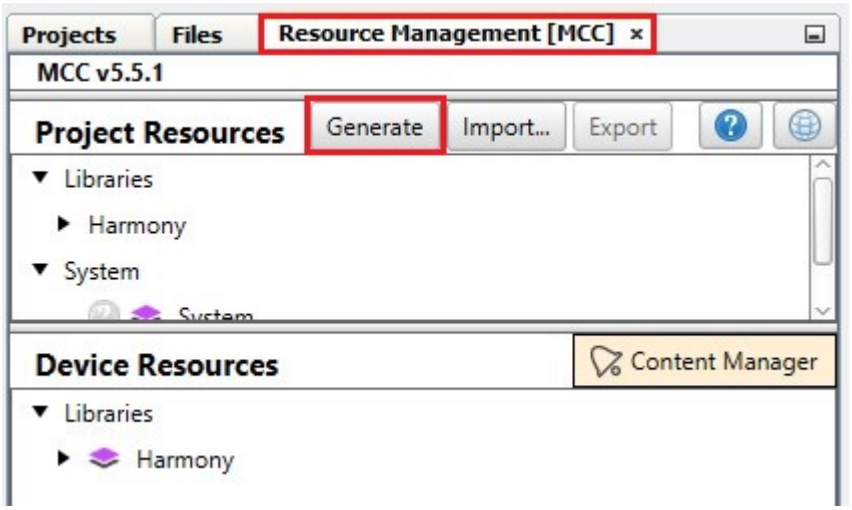
图 3-19. MPLAB®代码配置器——存储器配置



3.2. 生成代码

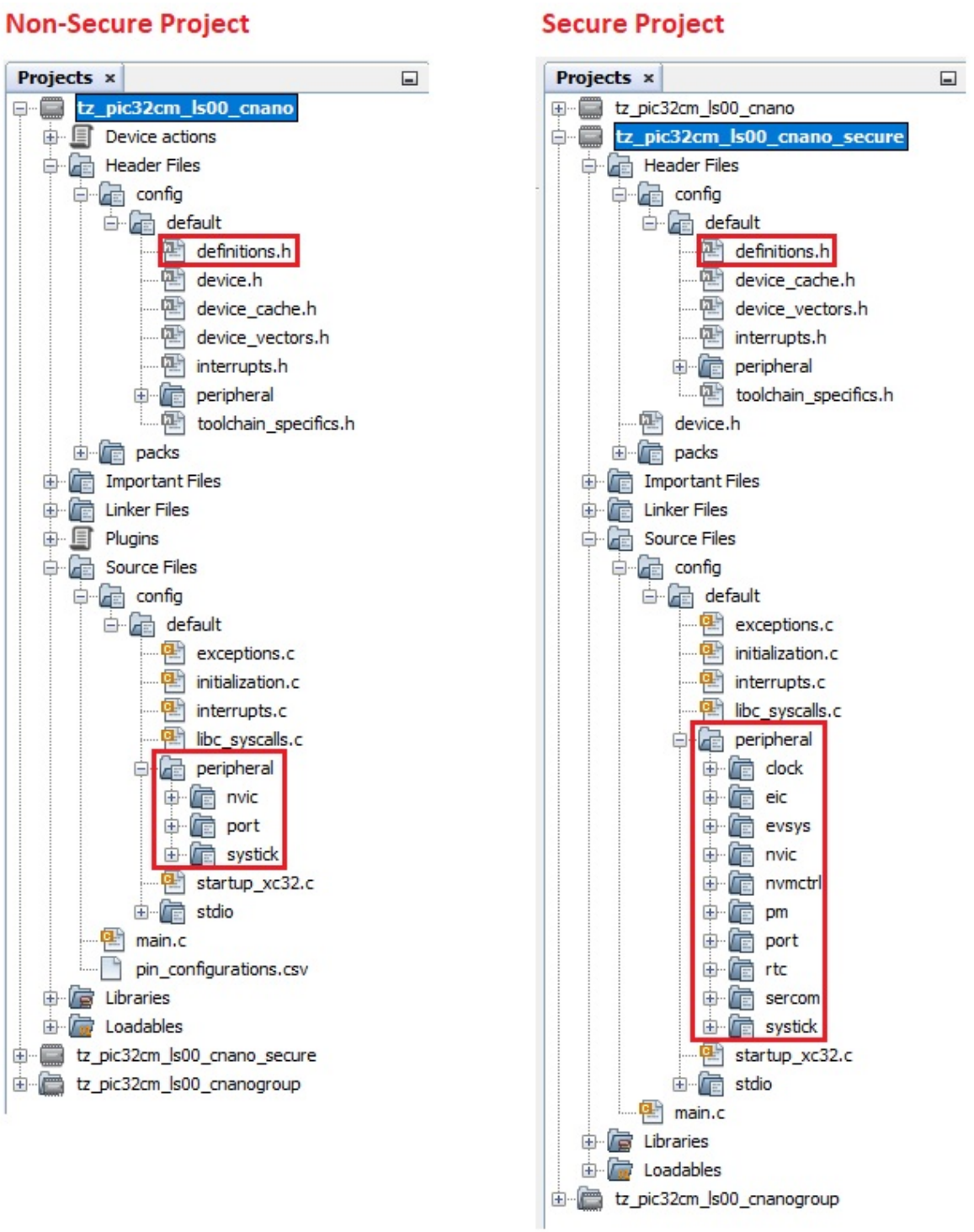
1. 配置外设后，单击 *Resource Management [MCC]*（资源管理[MCC]），然后单击 *Generate*（生成）选项卡。

图 3-20. 生成代码



2. 生成的代码会将文件和文件夹添加到 32 位 MCC Harmony v3 项目中。在生成的代码中，请注意针对 SysTick、SERCOM、EIC、NVMCTRL、RTC、事件系统和 PORT 外设生成的外设库文件。

图 3-21. 非安全和安全项目中生成的代码



注：

- MCC 在安全和非安全项目中生成单独的 `main.c` 文件。
- MCC 提供了一个选项来更改生成的文件名，如果不使用此选项，则生成的文件名默认为 `main.c`。

4. 将应用程序逻辑添加到非安全和安全项目

4.1. 添加非安全应用程序逻辑

要开发和运行该应用程序，请按照以下步骤操作：

1. 打开非安全项目（tz_pic32cm_ls00_cnano.X）的 `main.c` 文件，并在 SYS_Initialize()之后添加以下代码：

```
SYSTICK_TimerStart();
```

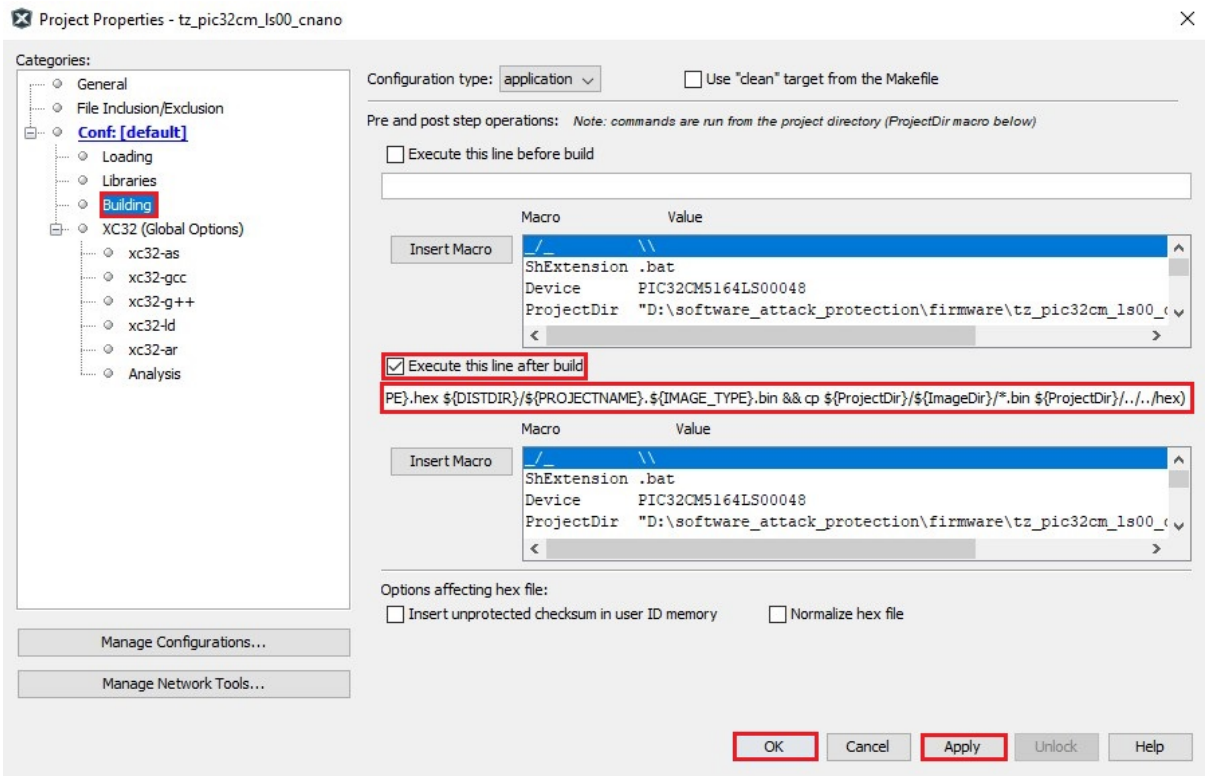
2. 在 while 循环内，添加以下代码，使 LED 以默认速率 500 ms 进行切换：

```
LED_Toggle();
SYSTICK_DelayMs(500);
```

3. 转到非安全项目属性，输入用于生成非安全固件真实副本的编译后命令：
   a. 在 MPLAB X IDE 的 Project Properties（项目属性）窗口中，执行这些操作。
   b. 在左侧的 Categories 部分下，选择 **Building**（编译），然后在右侧的 Configuration（配置）属性页面中，选中 **Execute this line after build**（在编译之后执行该行）复选框。
   c. 在复选框下方输入以下后处理命令。

```
rm -rf ${ProjectDir}/../../hex && mkdir ${ProjectDir}/../../hex&& cp
${ProjectDir}/${ImageDir}/*.hex ${ProjectDir}/../../hex &&
${MP_CC_DIR}"/xc32-objcopy" -I ihex -O binary
${DISTDIR}/${PROJECTNAME}.${IMAGE_TYPE}.hex
${DISTDIR}/${PROJECTNAME}.${IMAGE_TYPE}.bin && cp
${ProjectDir}/${ImageDir}/*.bin ${ProjectDir}/../../hex
```

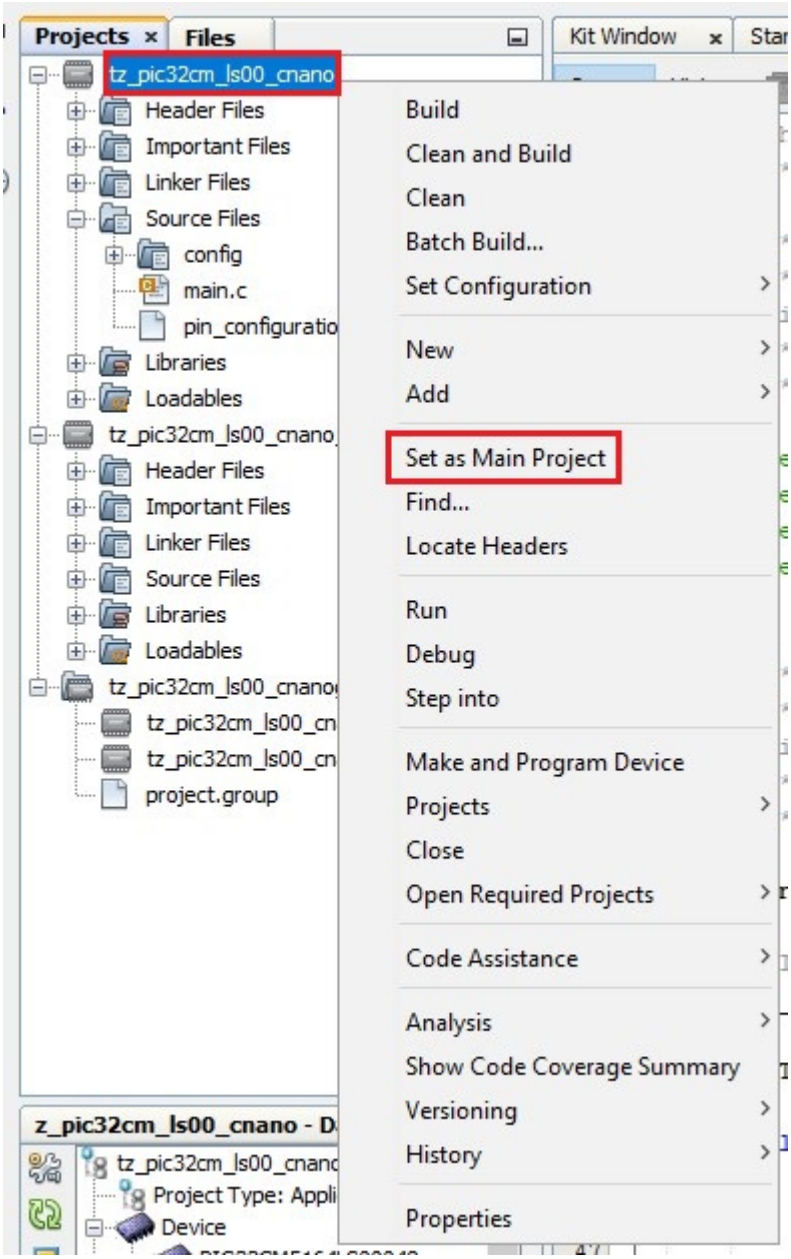
图 4-1. 生成非安全固件真实副本



4. 单击 **Apply**（应用），然后单击 **OK**（确定）。

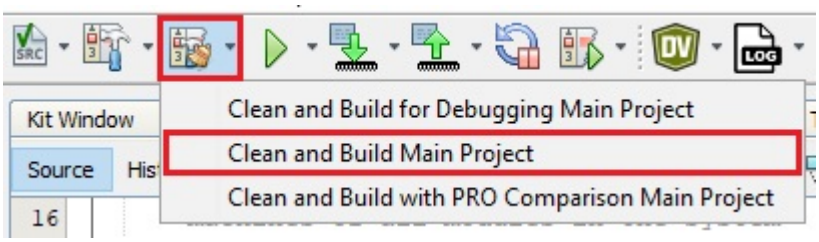
5. 在 Projects 下，右键单击 **tz_pic32cm_ls00_cnano**，然后单击 **Set as Main Project**（设置为主项目）。

图 4-2. 将非安全项目设置为主项目



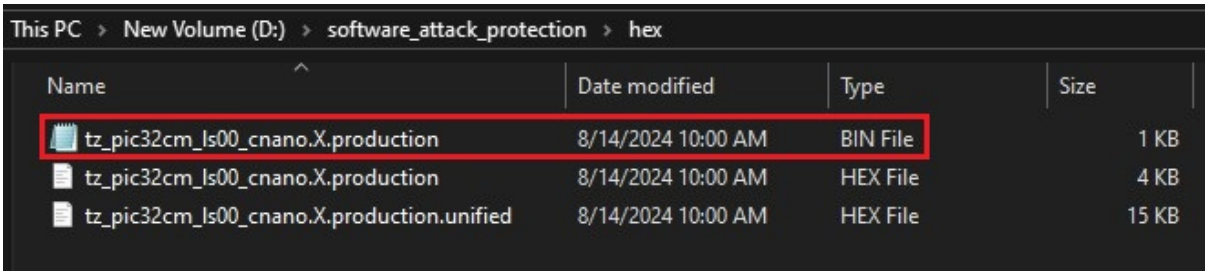
6. 通过单击 *Clean and Build*（清除并编译）图标或从下拉列表中选择 **Clean and Build Main Project**（清除并编译主项目）编译项目，然后验证项目是否成功编译。

图 4-3. 清除并编译



7. 检查非安全项目的二进制文件是否在 hex 文件夹位置（路径：*D:/software_attack_protection/hex*）。

图 4-4. 所生成二进制文件的位置



8. 打开命令提示符并导航到以下位置。
路径：*<Harmony 文件夹路径>/bootloader/tools*

注：如果在 Harmony 文件夹中未找到 bootloader 文件夹，请使用 MPLAB 内容管理器下载自举程序包（v3.7.0 或更高版本）。

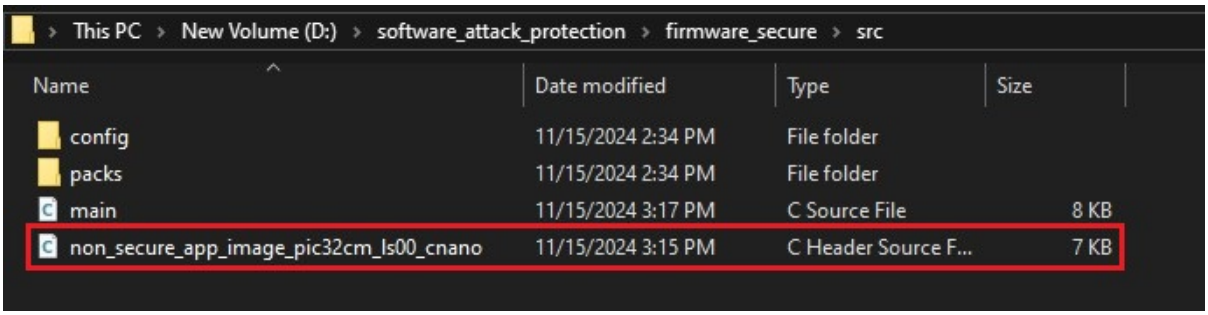
9. 运行 Python 脚本 `btl_bin_to_c_array.py`，将非安全应用程序二进制文件转换为包含十六进制输出的 C 语言风格数组。

```
python btl_bin_to_c_array.py -b
D:\software_attack_protection\hex\tz_pic32cm_ls00_cnano.X.production.bin -o
D:\software_attack_protection\firmware_secure\src\non_secure_app_image_pic32cm_ls00_cnano.h
 -d PIC32CM
```

图 4-5. 运行 Python 脚本

10. 脚本成功执行后，可以在安全项目的源文件夹中找到非安全应用程序映像（真实副本）的头文件。

图 4-6. 非安全应用程序映像的真实副本



4.2. 添加安全应用程序逻辑

要开发和运行该应用程序，请按照以下步骤操作：

1. 在 `main.c` 文件中声明安全应用程序使用的以下变量和宏。

```
#include <string.h>
#include "non_secure_app_image_pic32cm_ls00_cnano.h"

#define APP_IMAGE_SIZE          sizeof(image_pattern)
#define APP_IMAGE_END_ADDR      (APP_IMAGE_START_ADDR + APP_IMAGE_SIZE)
#define NON_SECURE_APP_ADDR     (TZ_START_NS)

static uint8_t *appStart = (uint8_t *)NON_SECURE_APP_ADDR;
static uint8_t *dataStart = (uint8_t *)NVMCTRL_DATAFLASH_START_ADDRESS;

uint8_t firmware_digest_0[64];
uint8_t firmware_digest_1[32];
```

图 4-7. 变量和宏声明

```
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  /* typedef for non-secure callback functions */
31  typedef void (*funcptr_void) (void) __attribute__((cmse_nonsecure_call));
32
33  #include <string.h>
34  #include "non_secure_app_image_pic32cm_ls00_cnano.h"
35
36  #define APP_IMAGE_SIZE          sizeof(image_pattern)
37  #define APP_IMAGE_END_ADDR      (APP_IMAGE_START_ADDR + APP_IMAGE_SIZE)
38  #define NON_SECURE_APP_ADDR     (TZ_START_NS)
39
40  static uint8_t *appStart = (uint8_t *)NON_SECURE_APP_ADDR;
41  static uint8_t *dataStart = (uint8_t *)NVMCTRL_DATAFLASH_START_ADDRESS;
42
43  uint8_t firmware_digest_0[64];
44  uint8_t firmware_digest_1[32];
```

2. 在 `main.c` 文件中添加引导 ROM API 以按如下方式访问它们。

```
typedef struct
{
    /* Digest result of SHA256 */
    uint32_t digest[8];
    /* 报文长度 */
    uint64_t length;
```

```
    /* 存放数据剩余部分的大小 */
    uint32_t remain_size;
    /* 数据剩余部分的缓冲区（512 位数据块） */
    uint8_t remain_ram[64];
    /* crya_sha_process 使用的 256 字节 RAM 缓冲区 */
    uint32_t process_buf[64];

} SHA256_CTX;

SHA256_CTX sha256_ctx;

typedef void (*crya_sha256_init_t) (SHA256_CTX *context);
typedef void (*crya_sha256_update_t) (SHA256_CTX *context, const unsigned char *data,
size_t length);
typedef void (*crya_sha256_final_t) (SHA256_CTX *context, unsigned char output[32]);

#define crya_sha256_init ((crya_sha256_init_t) (0x02006810 | 0x1))
#define crya_sha256_update ((crya_sha256_update_t) (0x02006814 | 0x1))
#define crya_sha256_final ((crya_sha256_final_t) (0x02006818 | 0x1))
```

3. 在 main.c 文件中包含 flash_write API，以便在非安全闪存区域中烧录非安全固件，并在安全数据闪存中写入固件摘要。

```
static void flash_write(uint32_t addr, uint8_t *buf, uint32_t size)
{
    uint32_t end_addr = addr + size;

    if((addr & NVMCTRL_DATAFLASH_START_ADDRESS) == NVMCTRL_DATAFLASH_START_ADDRESS)
    {
        /* 解锁安全数据闪存区域 */
        NVMCTRL_RegionUnlock(NVMCTRL_SECURE_MEMORY_REGION_DATA);
        while(NVMCTRL_IsBusy());
    }

    else
    {
        /* 解锁非安全闪存区域 */
        NVMCTRL_RegionUnlock(NVMCTRL_MEMORY_REGION_APPLICATION);
        while(NVMCTRL_IsBusy());
    }

    do
    {
        if(addr % NVMCTRL_FLASH_ROWSIZE == 0)
        {
            /* 擦除行 */
            NVMCTRL_RowErase(addr);
            while(NVMCTRL_IsBusy());
        }

        /* 编程 64 字节页 */
        NVMCTRL_PageWrite((uint32_t *)(buf), addr);
        while(NVMCTRL_IsBusy());

        addr += NVMCTRL_FLASH_PAGESIZE;
        buf += NVMCTRL_FLASH_PAGESIZE;

    }while (addr < end_addr);
}
```

4. 将 SHA-256 哈希和非安全固件验证 API 添加到 main.c 文件中，以计算非安全固件摘要。

```
static void sha256_hash(SHA256_CTX *ctx, const uint8_t *message, uint32_t length,
        unsigned char digest[32])
{
    uint8_t dataBuf[64];

    uint32_t bufIdx = 0;

    crya_sha256_init(ctx);

    do
    {
        memcpy(dataBuf, &message[bufIdx], 64);
```

```
        crya_sha256_update(ctx, dataBuf, sizeof(dataBuf));
        bufIdx += 64;

    }while (bufIdx < APP_IMAGE_SIZE);

    crya_sha256_final(ctx, digest);
}
```

```
static bool non_secure_app_verify(void)
{
    sha256_hash(&sha256_ctx, appStart, APP_IMAGE_SIZE, firmware_digest_1);

    if(memcmp(dataStart, firmware_digest_1, 32) != 0)
    {
        printf("Firmware is Corrupted....!");
        printf("\n\r\n\r");

        printf("Firmware Digest after tamper detection:");
        printf("\n\r\n\r");

        for(int i=0; i<32; i++)
        {
            printf("0x%X ", dataStart[i]);

            if((i%8 == 0) && (i != 0))
            {
                printf("\n\r");
            }
        }
        flash_write(TZ_START_NS, (uint8_t *)&image_pattern, sizeof(image_pattern));

        sha256_hash(&sha256_ctx, image_pattern, APP_IMAGE_SIZE, firmware_digest_1);

        printf("\n\r\n\r");
        printf("Restored Firmware Digest:");
        printf("\n\r\n\r");

        for(int i=0; i<32; i++)
        {
            printf("0x%X ", firmware_digest_1[i]);

            if((i%8 == 0) && (i != 0))
            {
                printf("\n\r");
            }
        }
        printf("\n\r\n\r");
        printf("Genuine Firmware is restored");

    }
    else
    {
        return false;
    }

    return true;
}
```

5. 在 `main.c` 中包含用于防篡改中断和 30 秒超时的 RTC 回调。

```
void timeout_handler(RTC_TIMER32_INT_MASK intCause, uintptr_t context)
{
    if(RTC_TIMER32_INT_MASK_CMP0  == (RTC_TIMER32_INT_MASK_CMP0 & intCause ))
    {
        if(non_secure_app_verify() == true)
        {
            SYSTICK_DelayMs(2000);

            NVIC_SystemReset();
        }
    }

    if (RTC_TIMER32_INT_MASK_TAMPER == (intCause & RTC_TIMER32_INT_MASK_TAMPER))
    {
        RTC_REGS->MODE2.RTC_TAMPID = RTC_TAMPID_Msk;
```

```
        printf("Software Attack Detected");
        printf("\n\r\n\r");
    }
}
```

6. 在 main.c 中的 SYS_Initialize API 之后添加以下代码片段。

```
sha256_hash(&sha256_ctx, image_pattern, APP_IMAGE_SIZE, firmware_digest_0);

flash_write(NVMCTRL_DATAFLASH_START_ADDRESS, firmware_digest_0, sizeof(firmware_digest_0));
```

注：

- sha256_hash：计算非安全固件的摘要。
- flash_write：将固件摘要存储在安全数据闪存区域中。

```
SYSTICK_TimerStart();

    RTC_Timer32CallbackRegister(timeout_handler,0);
    RTC_Timer32Start();
    printf("\n\r-------------------------------------------------------------");
    printf("\n\r           Software Attack Protection Demo                 ");
    printf("\n\r-------------------------------------------------------------\n\r");

    if(non_secure_app_verify() != true)
    {
        printf("\n\rFirmware is Genuine");
        printf("\n\r\n\r");
    }
```

注：

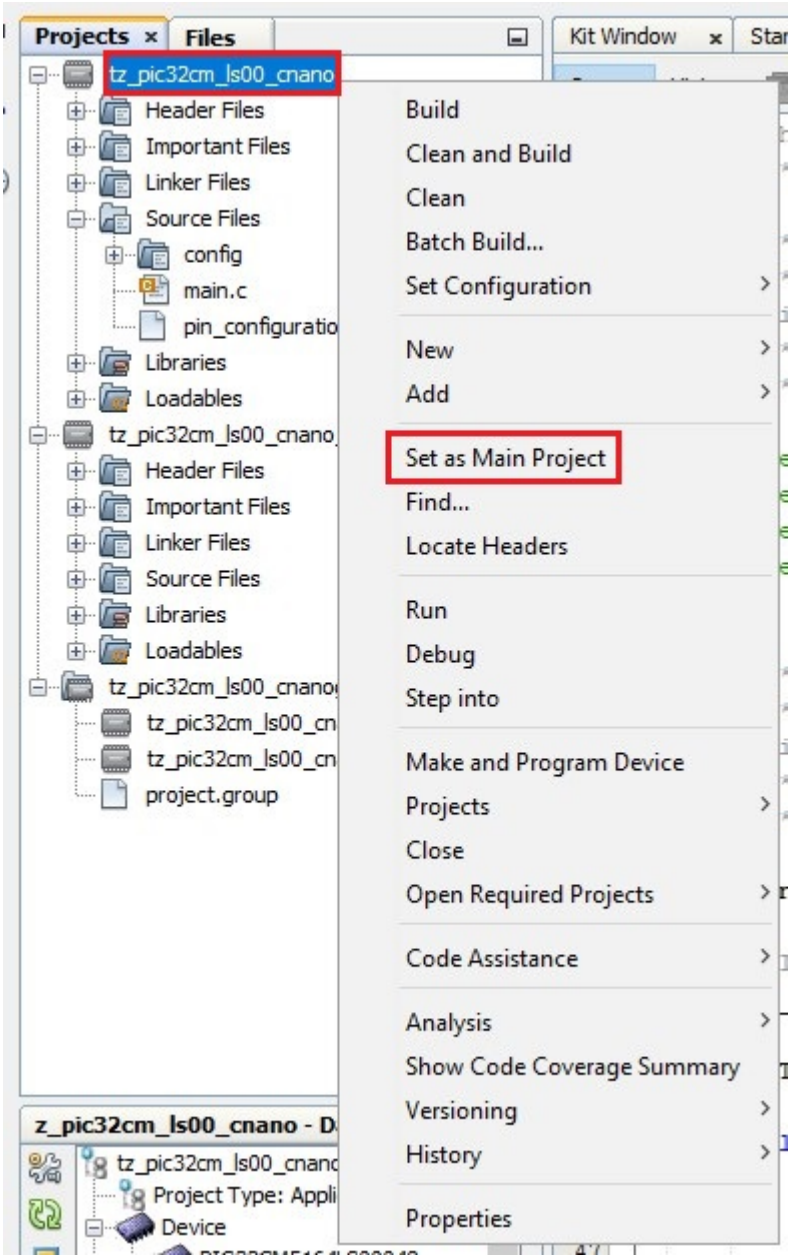
- non_secure_app_verify：验证非安全固件，如果验证失败，则将真实副本编程到非安全闪存区域中。

5. 编译并运行应用程序

请按照以下步骤在 PIC32CM LS00 Curiosity Nano+ Touch 评估工具包上编程软件攻击保护应用程序。

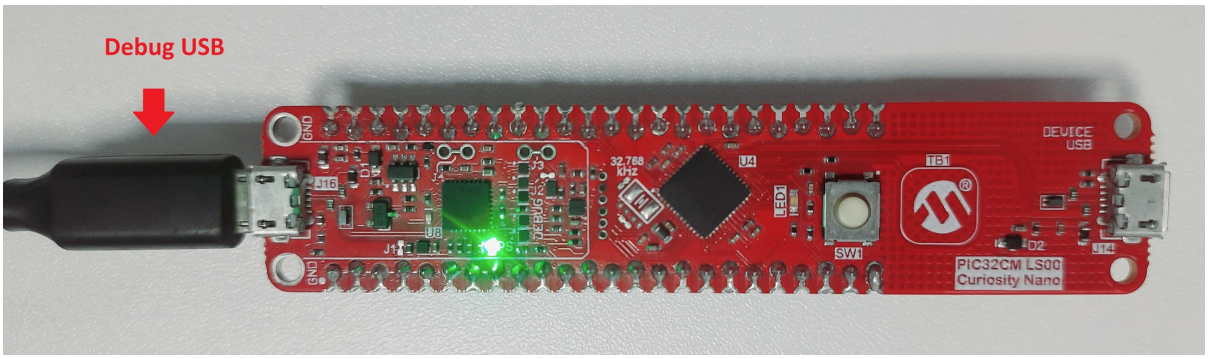
1. 通过右键单击项目并选择 **Set as Main Project**，将 *tz_pic32cm_ls00_cnano* 项目设置为主项目。

图 5-1. 将非安全项目设置为主项目



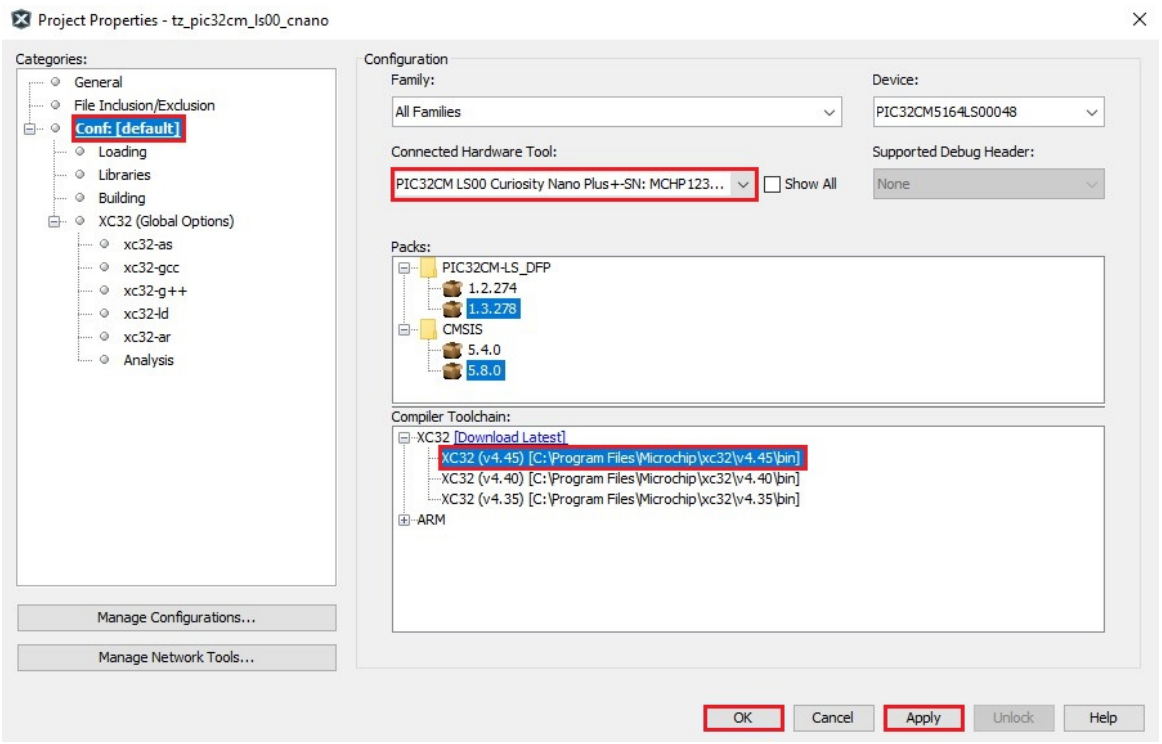
2. PIC32CM LS00 Curiosity Nano+ Touch 评估工具包支持使用 Nano 嵌入式调试器（nEDBG）进行调试。将 *Type-A 公头转 micro-B* USB 线缆连接到 PIC32CM LS00 Curiosity Nano+ Touch 评估工具包上的 *micro-B* USB 端口，以便对 PIC32CM LS00 Curiosity Nano+ Touch 评估工具包进行供电和调试。

图 5-2. PIC32CM LS00 Curiosity Nano+ Touch 评估工具包硬件设置



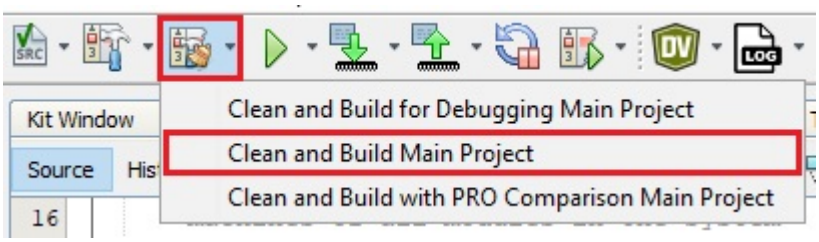
3. 转到 Project Properties，选择硬件工具和编译器：
   a. 在 MPLAB X IDE 的 Project Properties 窗口中，执行这些操作。
   b. 在左侧的 Categories 部分下，选择 *Conf: [default]*，然后在右侧 Configuration 属性页面中选择 Connected Hardware Tool（连接的硬件工具）和 Compiler Toolchain（编译器工具链）。

图 5-3. 项目属性——PIC32CM LS00 Curiosity Nano+ Touch 评估工具包



4. 单击 **Apply**，然后单击 **OK**。
5. 通过单击 *Clean and Build* 图标或从下拉列表中选择 **Clean and Build Main Project** 编译项目，然后验证项目是否成功编译。

图 5-4. 清除并编译



6. 单击下图中红框部分的图标以编程应用程序。

图 5-5. 编程器件

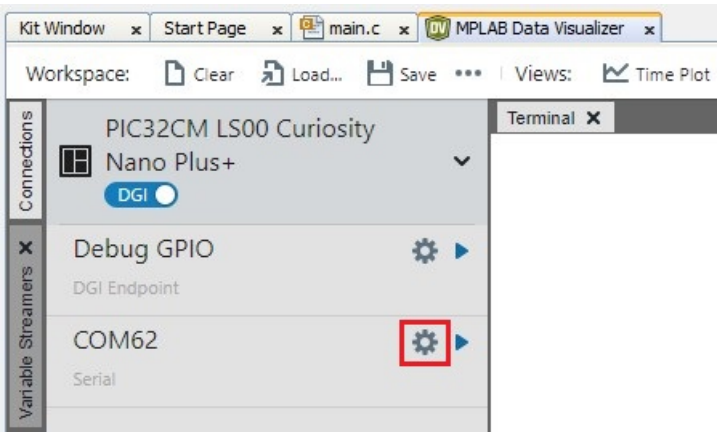
6. 观察 MPLAB 数据可视化器上的输出

1. 编译应用程序并完成编程后，单击下图中红框部分的图标打开 MPLAB 数据可视化器。

图 6-1. 启动 MPLAB 数据可视化器

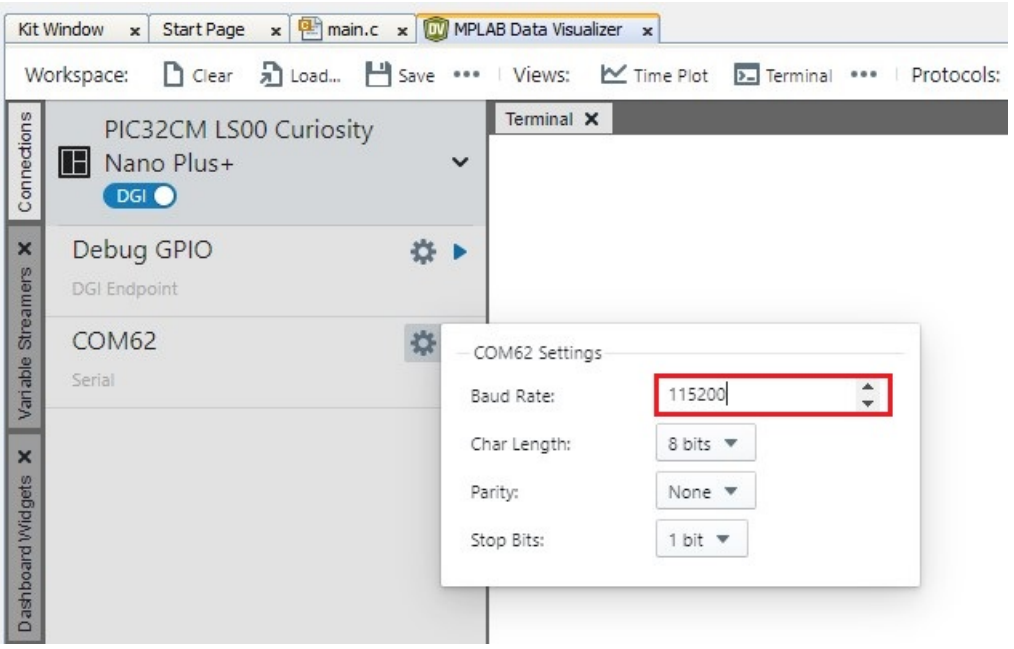
2. 单击下图所示的*齿轮*图标，配置 PIC32CM LS00 Curiosity Nano+ Touch 评估工具包的串行端口设置。

图 6-2. 串行端口设置



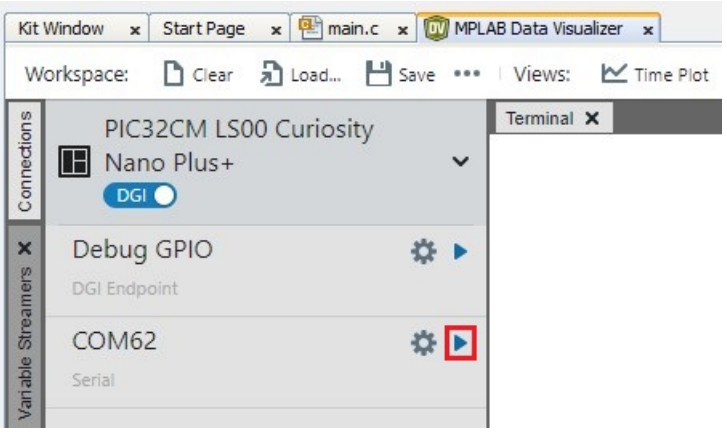
3. 在 COM 设置中将波特率设置为 115200。

图 6-3. 波特率配置



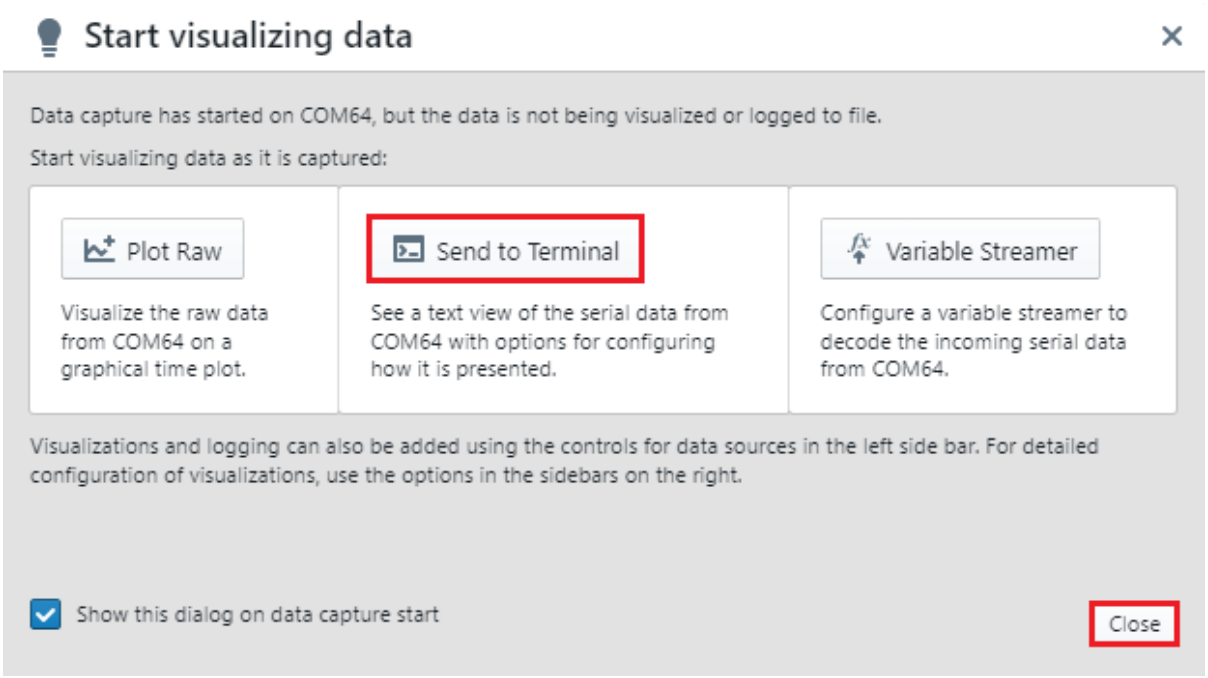
4. 单击下图所示的*播放*图标，打开 PIC32CM LS00 Curiosity Nano+ Touch 评估工具包的串行端口。

图 6-4. 打开串行 COM 端口



5. 单击 **Send to Terminal**（发送到终端）查看串行控制台消息，然后单击 **Close**（关闭）。

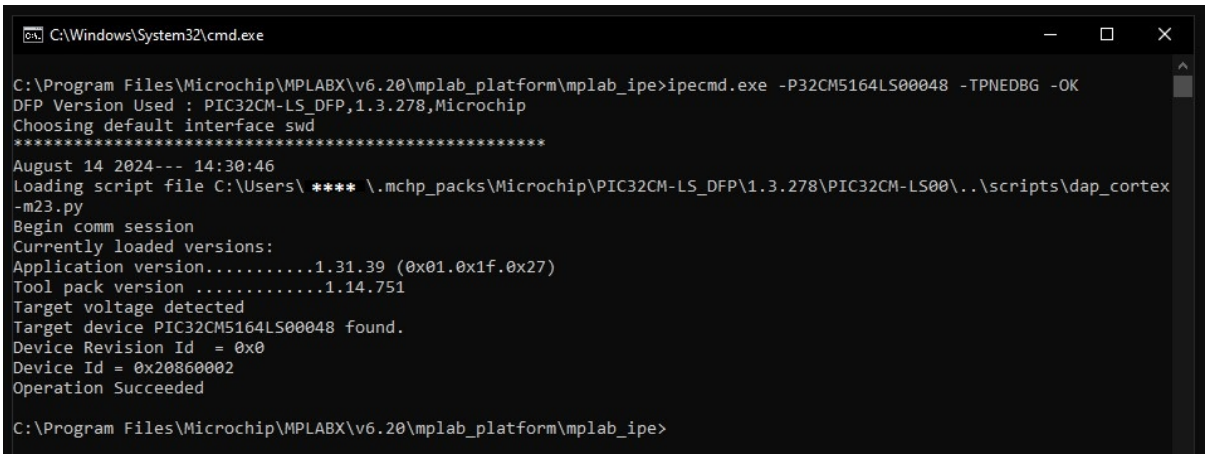
图 6-5. 选择终端选项



6. 打开命令提示符并导航到以下位置：*C:\Program Files\Microchip\MPLABX\v6.20\mplab_platform\mplab_ipe*。
注：PIC32CM LS00 Curiosity Nano+ Touch 评估工具包没有用于复位 MCU 的复位按钮。要重启开发板，可借助 MPLAB IPECMD，将 `reset` 命令发送到 nEDBG 以复位 MCU。
7. 运行以下命令以复位 PIC32CM LS00 Curiosity Nano+ Touch 评估工具包。

```
ipecmd.exe -P32CM5164LS00048 -TPNEDBG -OK
```

图 6-6. 复位 PIC32CM LS00 Curiosity Nano+ Touch 评估工具包



8. 在 MPLAB 数据可视化器上观察启动控制台消息，PIC32CM LS00 Curiosity Nano+ Touch 评估工具包上的 LED1 将切换。

图 6-7. 启动控制台消息

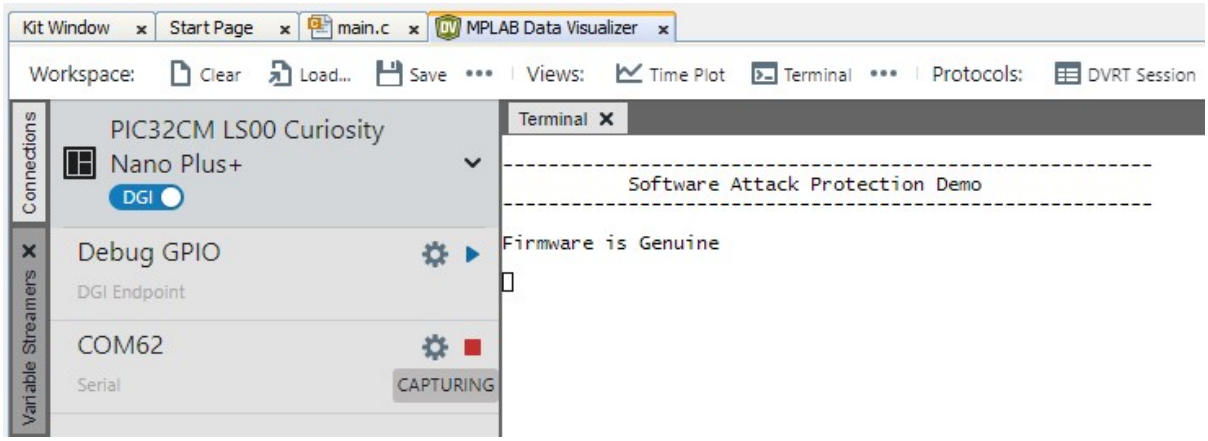
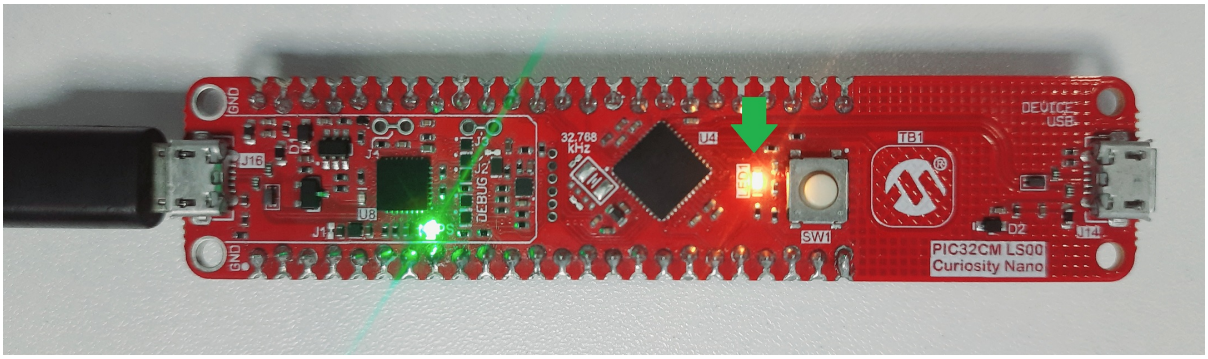
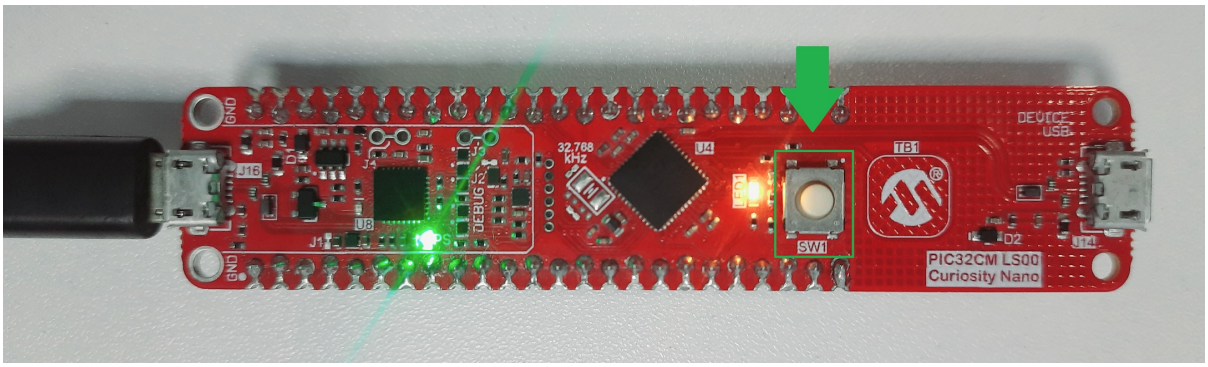


图 6-8. PIC32CM LS00 Curiosity Nano+ Touch 评估工具包上的 LED1 切换



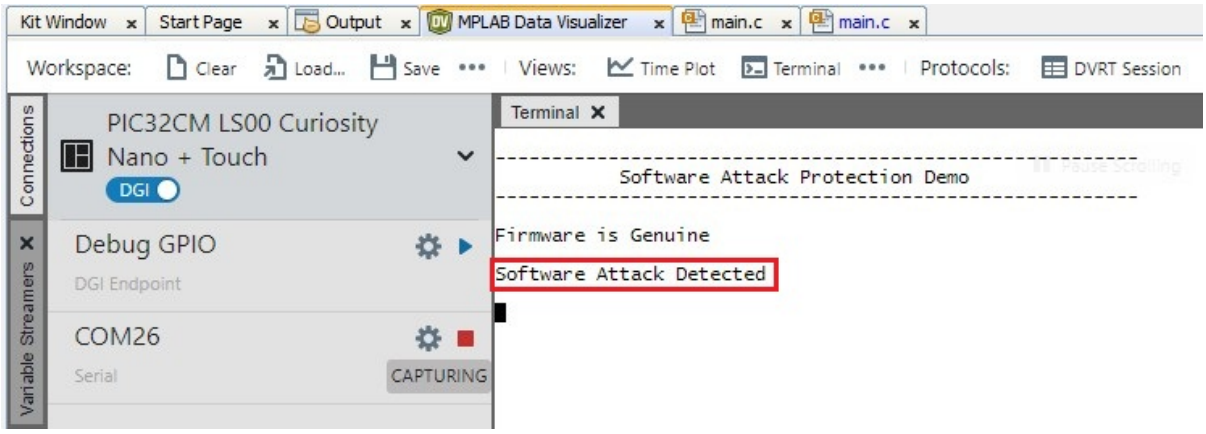
9. 按下 PIC32CM LS00 Curiosity Nano+ Touch 评估工具包上的 SW1 按钮以模拟软件攻击。

图 6-9. 软件攻击模拟



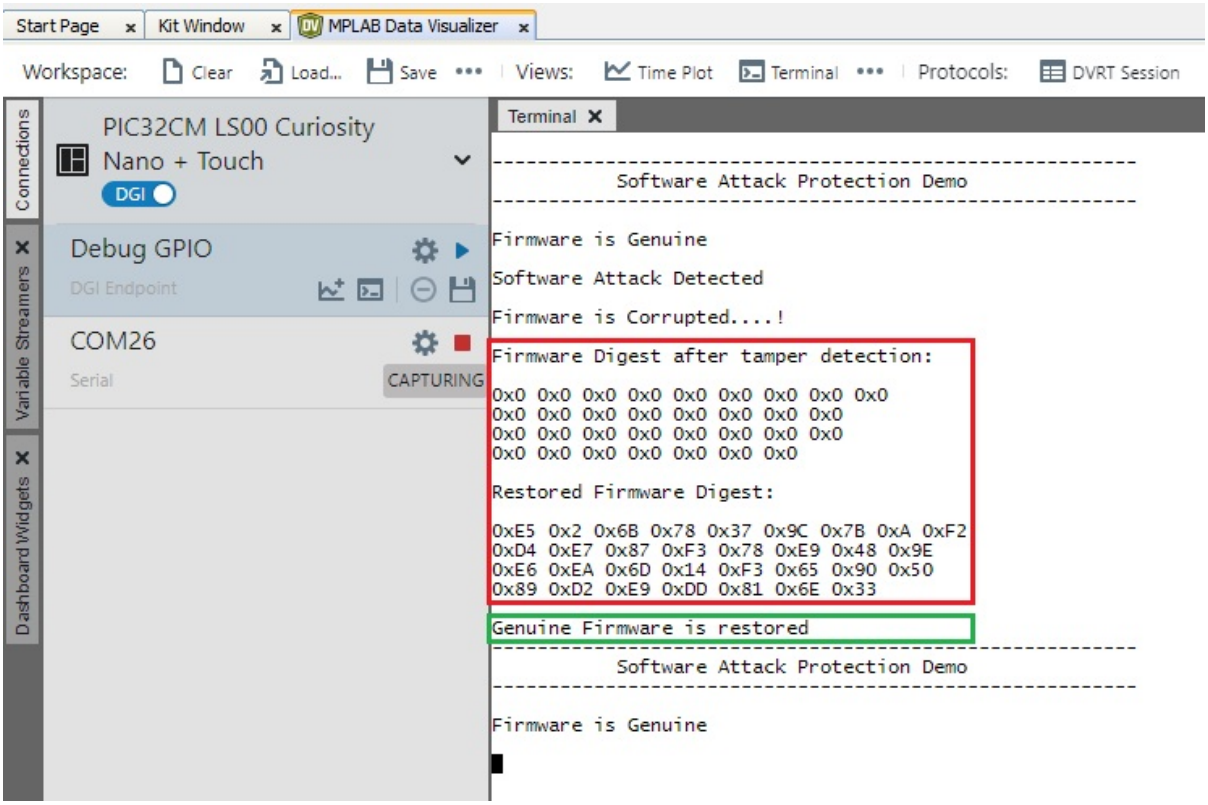
10. 观察 MPLAB 数据可视化器上的软件攻击启动控制台消息。

图 6-10. 软件攻击启动



11. 一旦达到 RTC 超时，安全应用程序就会开始验证非安全固件。如果验证失败，则会将非安全固件的真实副本编程到非安全闪存区域。

图 6-11. 非安全固件验证



注：成功编程真实副本后，安全应用程序将启动软件复位。

7. 资源

- 可从 Microchip 网站（www.microchip.com）下载以下文档：
  - PIC32CM LS00 Curiosity Nano+ Touch 评估工具包用户指南（*DS70005567B_CN*）
  - PIC32CM LS00/LS60 Security Reference Guide（*DS00003992*）
- PIC32CM LS00 Curiosity Nano+ Touch 评估工具包
- 使用 MPLAB® Harmony v3 软件框架在 PIC32CM LS60 Curiosity Pro 评估工具包上实现安全引导
- 有关 MPLAB Harmony v3 的更多信息，请访问 Microchip 网站： 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 https://github.com/Microchip-MPLAB-Harmony/reference_apps
- 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*）

8. 版本历史

版本 A——2025 年 4 月

这是本文档的初始版本。

Microchip 信息

商标

“Microchip”的名称和徽标组合、“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/或其他国家或地区的注册商标或商标（“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： 979-8-3371-3054-5

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、偶然的或附带的损失、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

PIC32CM2532LS00048、PIC32CM2532LS00064、PIC32CM2532LS00100、PIC32CM2532LS60048、PIC32CM2532LS60064、PIC32CM2532LS60100、PIC32CM5164LS00048、PIC32CM5164LS00064、PIC32CM5164LS00100、PIC32CM5164LS60048、PIC32CM5164LS60064 和 PIC32CM5164LS60100