注意 : この日本語版文書は参考資料としてご利用ください。
最新情報は必ずオリジナルの英語版をご参照願います。

PIC32CM LS00 MCU のソフトウェア攻撃対策

AN5880

製品ページへのリンク

はじめに

現在の組み込みシステムはソフトウェア攻撃の影響をますます受けやすくなっています。ソフトウェア攻撃とは、ソフトウェアの脆弱性につけ込んで不正アクセス、データ盗用、サービス妨害等の損害を与える事を企む悪意ある行為です。同時に知的財産を保護する事も極めて重要です。

本書は、PIC32CM LS00 Curiosity Nano+ Touch 評価用キットを使って PIC32CM LS00 MCU をソフトウェア攻撃から保護するためのガイドラインを提供します。PIC32CM LS00 は、ブート ROM のセキュア ハッシュ アルゴリズム 2 (SHA-256)認証を使って非セキュアメモリ内の不正コード部分を識別し、それらをセキュアメモリ領域からの正規(真正)コピーで置き換えます。

目次

1. ハードウェアおよびソフトウェア要件

1.1. PIC32CM LS00 Curiosity Nano+ Touch 評価用キット

PIC32CM LS00 Curiosity Nano+ Touch 評価用キットは、セキュアかつ超低消費電力の PIC32CM LS00 Arm® Cortex®-M23 マイクロコントローラを使った回路の評価および試作用に理想的です。この MCU は Arm TrustZone®テクノロジ、強化されたペリフェラル タッチ コントローラ(PTC)、スマートアナログ モジュール (オペアンプ、ADC、DAC、アナログ コンパレータ等)を内蔵しています。

本評価用キットは Nano 組み込みデバッガ(nEDBG)を内蔵するため、プログラミングおよびデバッグ用の外部ツールは不要です。PIC32CM LS00 MCU の主な特長は以下の通りです。

- 48 MHz Arm Cortex-M23 コア
- 512 KB フラッシュと 64 KB SRAM
- 改変不可能なセキュアブート、暗号処理アクセラレータ、タンパー検出機能

PIC32CM LS00 Curiosity Nano+ Touch 評価用キットは Microchip 社オンラインストアでご購入になれます。

1.2. MPLAB® X 統合開発環境(IDE)と MPLAB XC コンパイラ

MPLAB X IDE は拡張と柔軟な設定が可能なソフトウェア プログラムです。この IDE はほとんど全ての Microchip 社製マイクロコントローラをサポートし、組み込み回路の研究、設定、開発、デバッグ、評価に役立つ強力なツールを備えています。

- MPLAB X IDE は Microchip 社ウェブサイトからダウンロードできます。本書の説明は MPLAB X IDE バージョン 6.20 に基づきます。
- MPLAB XC コンパイラは Microchip 社ウェブサイトからダウンロードできます。本書の説明は MPLAB XC32 バージョン 4.45 に基づきます。

1.3. MPLAB Harmony v3

MPLAB Harmony 3 は、柔軟かつ相互運用性に優れたソフトウェア モジュールを提供する統合型の組み込みソフトウェア開発フレームワークです。これによりデバイスの細部、複雑なプロトコル、ライブラリの組み込み等に煩わされる事なく、専用のリソースを使って 32 ビット PIC®および SAM デバイス向けアプリケーションを容易に開発できます。

MPLAB® Harmony 3 には MPLAB Harmony Configurator (MHC)が含まれます。このツールは使いやすいグラフィカル ユーザ インターフェイス(GUI)を備えており、デバイスの設定、ライブラリの選択、アプリケーション コードの開発が容易に行えます。MCC は MPLAB X IDE に統合されるプラグインとして提供されますが、他の開発環境でスタンドアロンとして使うための Java 実行ファイルも別に備えています。

本書で説明する応用例では、以下の MPLAB Harmony v3 リポジトリを使います。これらのリポジトリは GitHub からダウンロードできます。

- csp v3.20.0 (MPLAB Harmony v3 チップサポート パッケージ)
- ブートローダ v3.7.0

これらのリポジトリは MCC Content Manager を使ってダウンロードする事もできます。

2. PIC32CM LS00 MCU を使ったソフトウェア攻撃対策

2.1. ブート ROM 機能

PIC32CM LS00/LS60 シリーズはハードウェアまたはソフトウェア暗号処理アクセラレータ(CRYA)を備えており、一連の API を使って先進的暗号化標準(AES)による暗号化/復号、セキュア ハッシュ アルゴリズム 2 (SHA-256)による認証、ガロア カウンタモード(GCM)による暗号化/認証が容易に行えます。

CRYA 暗号処理アクセラレータは IOBUS 上のクライアントとして設定され、ブート ROM 内に保存されたアセンブリコードを使って CPU により制御されます。

AES は米国の連邦情報処理標準(FIPS)が規定する FIPS 197 仕様を順守します。AES はデータを 128 ビットのブロック単位で処理します。入力平文を最終出力(暗号文)に変換するために必要な変換ラウンドの数は AES 暗号の鍵長によって決まります。AES は対称鍵アルゴリズムを使います。これは暗号化と復号の両方で同じ鍵を使う事を意味します。

SHA-256 は 1 つのデータブロックから 256 ビットハッシュを生成する暗号学的ハッシュ関数であり、512 ビット単位で処理されます。

ガロア カウンタモード(GCM)は AES の動作モードの 1 つであり、カウンタ(CTR)モードを認証ハッシュ関数と組み合わせます。

2.2. セキュアハッシュ アルゴリズム 2 (SHA-256)による認証

ハッシュ関数の主な目的は、データの特定セットに対して特異なデジタル識別子(指紋のような物)を生成する事です。誤り検出コードと異なり、各データセットは一意の識別子に対応付けられる必要があります。

実際には、ハッシュ関数は可変長の入力から固定長の出力(メッセージ ダイジェストと呼ぶ)を生成します。ハッシュ関数は各種の重要特性を持ち、これには「優れた拡散性」(入力がわずかでも変化すると出力が大きく異なる事を保証する特性)が含まれます。

出力が固定長である事から、存在し得る全ての入力データに対して完全一意のダイジェストを生成する事は理論的にできませんが、同一ダイジェストを生成する 2 つのメッセージを発見する事が極めて困難となるよう(実用上は実質的に一意と見なせるよう)にハッシュ関数が設計されます。

2.3. ブート ROM からの SHA-256 API の使用

暗号処理アクセラレータ(CRYA) API は専用のブート ROM 領域内に配置されます。この領域は「実行専用」です。すなわち、CPU は API を呼び出せますが、この領域に書き込む事は一切できません。ブート ROM メモリ空間はセキュア領域であり、セキュア アプリケーションのみがこれらの API を直接呼び出せます。

表 2-1. CRYA API のアドレス

CRYA API	アドレス
AES Encryption	0x02006804
AES Decryption	0x02006808
SHA256 Init	0x02006810
SHA256 Update	0x02006814
SHA256 Final	0x02006818
SHA256 Process (レガシーAPI)	0x02006800
GCM Process	0x0200680C

API は下記の関数で構成され、これらは決められた順番で呼び出される必要があります。

1. SHA-256 Init: SHA256_CTX 構造体を初期化します。
2. SHA-256 Update: ダイジェストの計算に含めるメッセージを追加します。
3. SHA-256 Final: ダイジェストを計算します。

Note: ダイジェストの計算に複数のメッセージが含められる場合、SHA-256 Update は複数回呼び出される可能性があります。

SHA-256 構造体は下記の通り「SHA56_CTX」として定義されます。

```
typedef struct
{
    /* Digest result of SHA256 */
    uint32_t digest[8];
    /* Length of the message */
    uint64_t length;
    /* Holds the size of the remaining part of data */
    uint32_t remain_size;
    /* Buffer of remaining part of data (512 bits data block) */
    uint8_t remain_ram[64];
    /* RAM buffer of 256 bytes used by crya_sha_process */
    uint32_t process_buf[64];

} SHA256_CTX;
```

SHA-256 Init 関数のエントリポイントは、ブート ROM アドレス 0x02006810 に配置されます。

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

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

SHA-256 Update 関数のエントリポイントは、ブート ROM アドレス 0x02006814 に配置されます。

```
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 関数のエントリポイントは、ブート ROM アドレス 0x02006818 に配置されます。

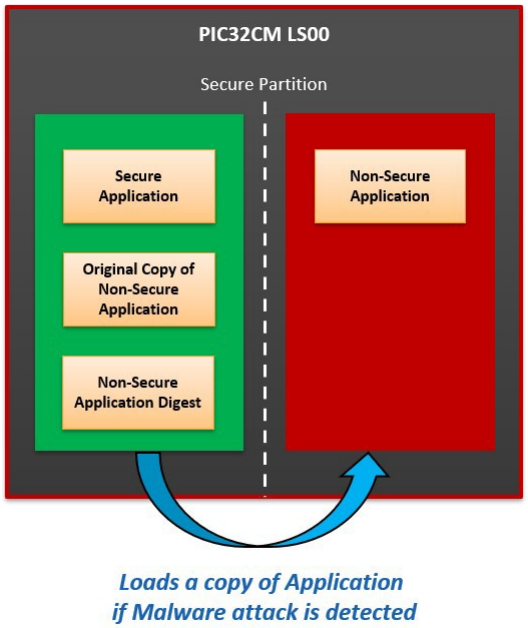
```
typedef void (*crya_sha256_final_t) (SHA256_CTX *context, unsigned char output[32]);

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

2.4. ソフトウェア攻撃に対する非セキュア領域の保護

デバイスの起動中に、セキュア アプリケーションは非セキュア ファームウェアに対して一意識別子(ダイジェスト)を計算し、それをセキュアメモリ (セキュア データフラッシュ)内に保存します。非セキュア ファームウェアの真正性を確保するには周期的な検証が必要です。このためタイマを使って特定時間間隔でファームウェアの信頼性をチェックします。

非セキュア アプリケーションにマルウェアまたは不正コードが注入された場合、改変されたファームウェアから計算されたダイジェストは真正の非セキュア アプリケーションから得られるはずのダイジェストと一致しません。この場合、セキュア アプリケーションは、システム動作を停止する事なく、セキュアメモリ領域に保存された非セキュア アプリケーションのオリジナルコピーを非セキュア フラッシュ領域に書き込みます(図 2-1 参照)。

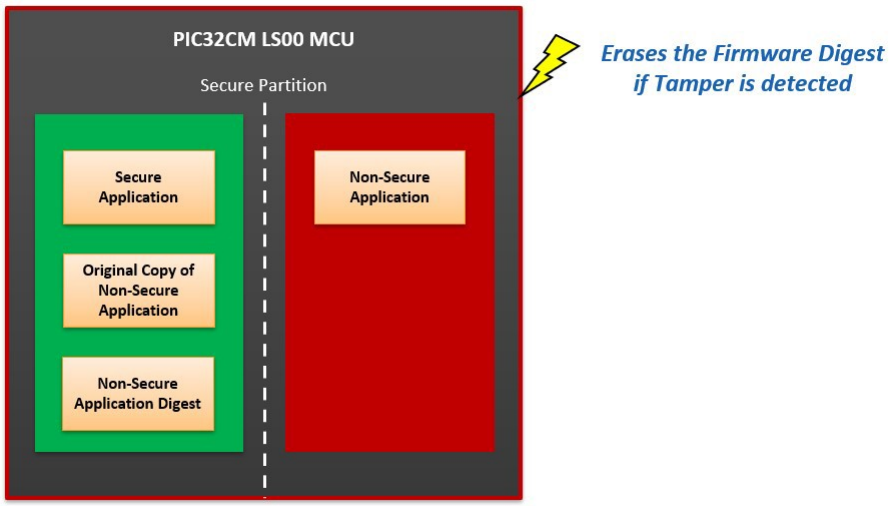
図 2-1. 非セキュア ファームウェアのソフトウェア攻撃対策



2.4.1. ソフトウェア攻撃のシミュレーション

PIC32CM LS00/LS60 ファミリのデバイスは、タンパー消去セキュリティ機能を持つタンパー検出をリアルタイム クロック(RTC)モジュール内に備えています。ソフトウェア攻撃をシミュレートするため、セキュアメモリ領域に保存されたデータを消去するタンパー消去オプションが使われます。改ざんが検出されると、セキュア アプリケーション内の RTC モジュールによってタンパー消去動作がトリガされ、セキュア データフラッシュ領域の内容(ファームウェア ダイジェスト)が削除されます。

図 2-2. タンパー検出機能を使ったソフトウェア攻撃のシミュレーション

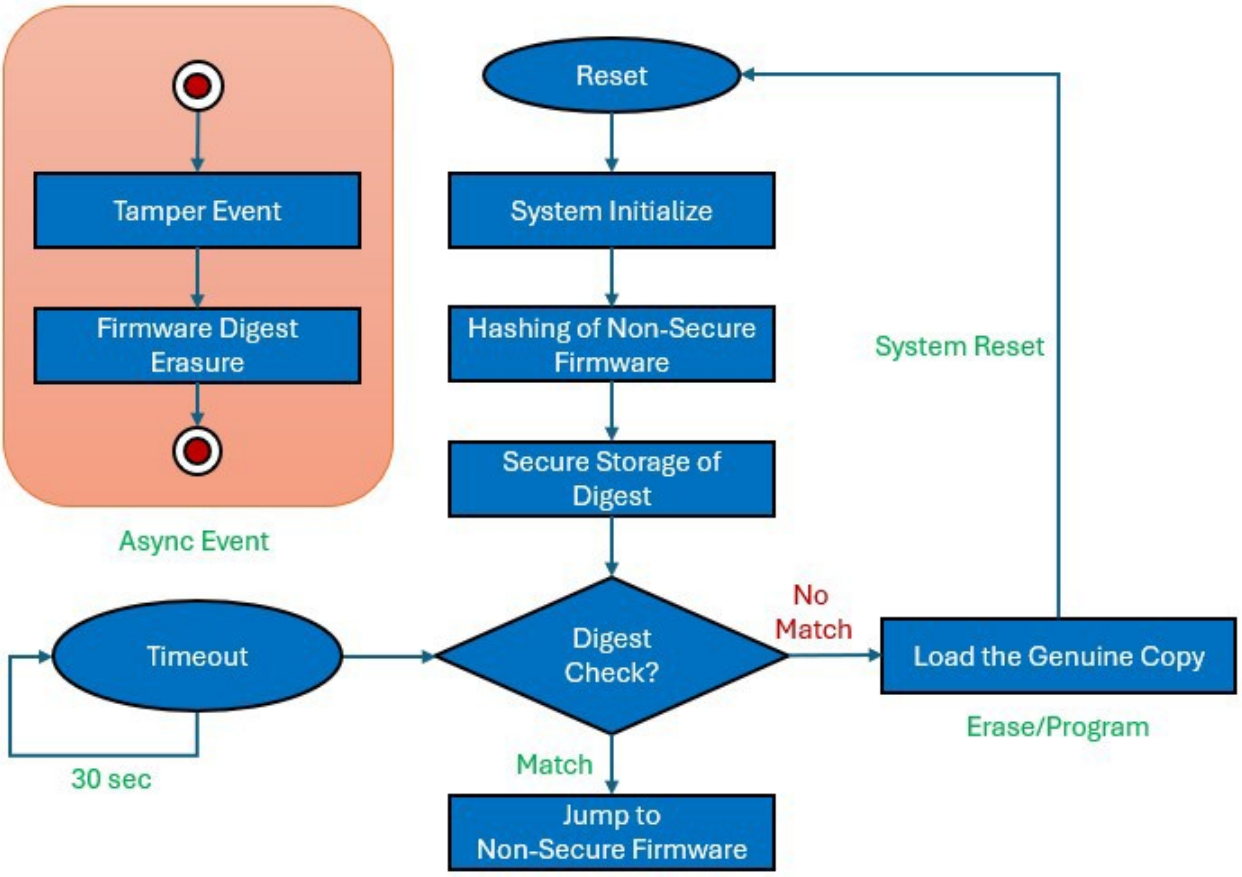


2.4.2. 実行フロー

セキュア ファームウェアの実行フロー

図 2-3 に、ソフトウェア攻撃対策アプリケーションにおけるセキュア ファームウェアのシステムレベル実行フローを示します。

図 2-3. セキュア アプリケーションの実行フロー



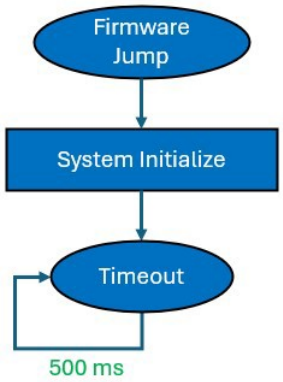
セキュア アプリケーションは以下のシーケンスで実行されます。

1. システムリセット後に、セキュア アプリケーションは非セキュア ファームウェアのハッシング処理を実行してファームウェア ダイジェストを生成します。
2. 計算されたダイジェストは、セキュア領域内のデータ フラッシュメモリに保存されます。
3. 非セキュア アプリケーションのファームウェア ダイジェストは、セキュア フラッシュメモリ内の真正コピーに対して検証されます。
4. 検証に成功すると、実行は非セキュア アプリケーションへジャンプします。
5. 検証に失敗すると非セキュア アプリケーションは消去され、真正コピーが非セキュア フラッシュ領域に書き込まれます。
6. 30 秒ごとにファームウェア ダイジェストが再生成され、真正コピーとの照合検証により真正性が確保されます。

非セキュア ファームウェアの実行フロー

図 2-4 に、ソフトウェア攻撃対策アプリケーションにおける非セキュア ファームウェアのシステムレベル実行フローを示します。

図 2-4. 非セキュア アプリケーションの実行フロー



非セキュア アプリケーションは以下のシーケンスで実行されます。

1. セキュア アプリケーションからのファームウェア ジャンプ後に、非セキュア ファームウェアは非セキュア周辺モジュールを初期化します。
2. PIC32CM LS00 Curiosity Nano+ Touch 評価用キット上で LED1 を 500 ms 周期でトグルします。

ソフトウェア攻撃シミュレーションの実行フロー

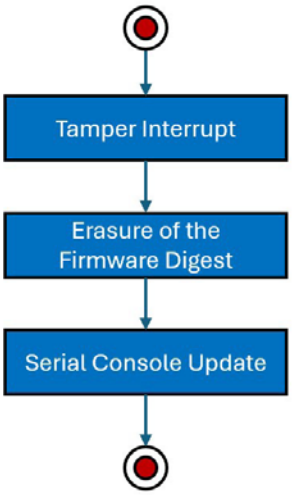
ソフトウェア攻撃のシミュレーションは以下のシーケンスで実行されます。

1. PIC32CM LS00 Curiosity Nano+ Touch 評価用キット上の SW1 ボタンを押すと、タンパーイベントがトリガされる事によりソフトウェア攻撃がシミュレートされます。
2. タンパーイベント後に、RTC はデータフラッシュの内容消去プロセスを開始します。
3. RTC タンパーハンドラ内で、ソフトウェア攻撃の開始を示すメッセージがシリアル コンソールへ送信されます。

Note: この実行はセキュア ファームウェアの RTC 割り込みハンドラ内で発生します。

図 2-5 に、セキュア ファームウェアにおけるソフトウェア攻撃応答処理のシステムレベル実行フローを示します。

図 2-5. ソフトウェア攻撃応答処理の実行フロー

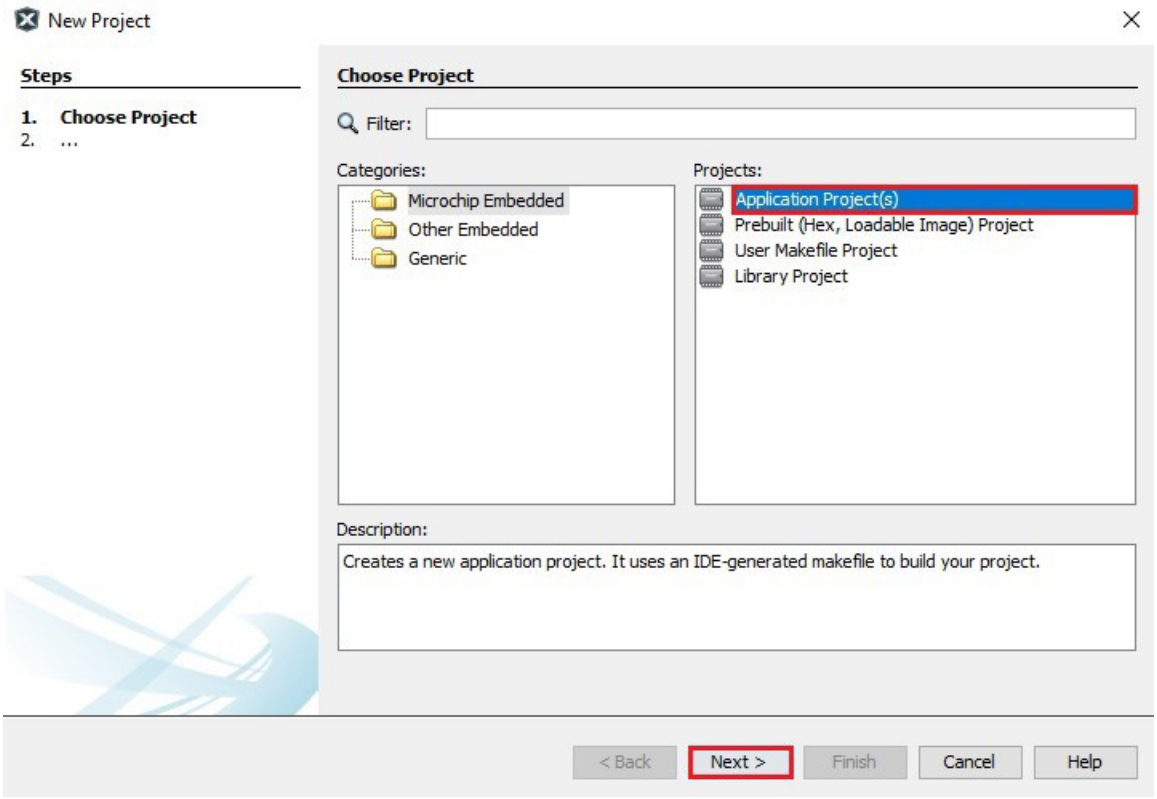


3. MPLAB Harmony v3 と MCC を使って PIC32CM LS00 Curiosity Nano+ Touch 評価用キット上でソフトウェア攻撃対策を実装する方法

MPLAB Harmony v3 ベースのプロジェクトを作成するには、以下の手順に従うか、開発済みデモプロジェクトを こちら からダウンロードします。

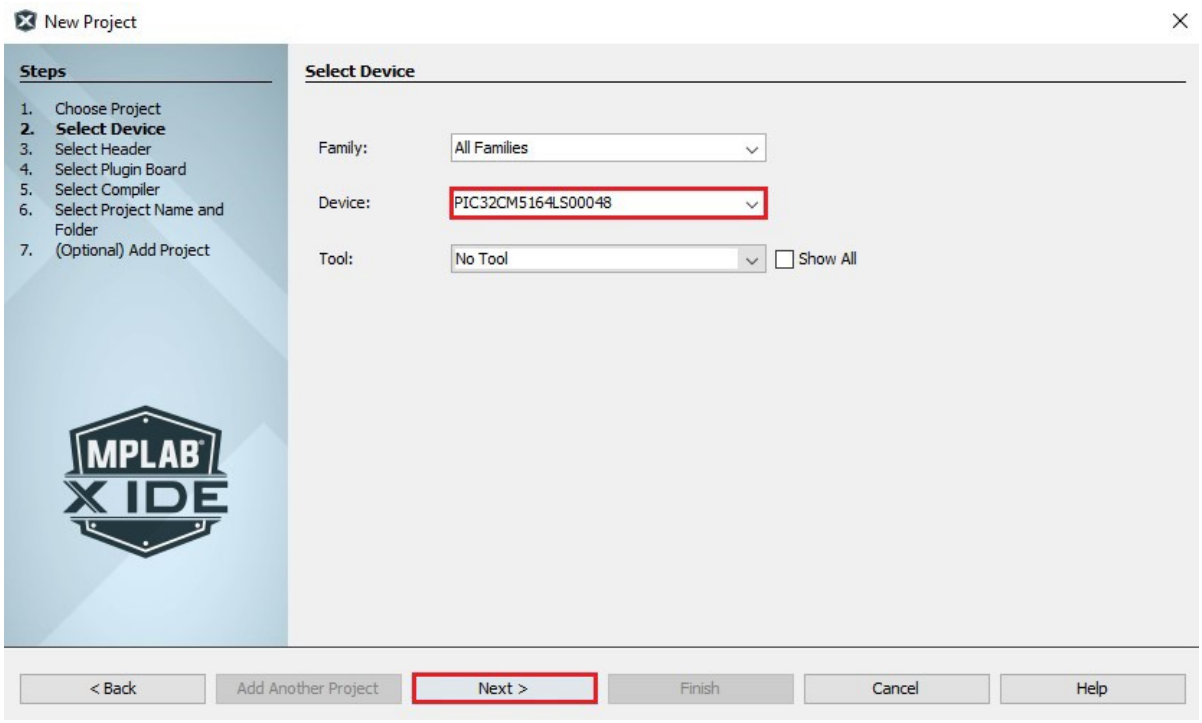
1. **[Start]**メニューから **MPLAB X IDE** を起動します。
2. MPLAB X IDE が開いたら、**[File]**メニューから**[New Project]**をクリックするか、**[New Project]**アイコンをクリックします。
3. **「New Project」**ウィンドウ内の左側の**「Steps」**の下で**[Choose Project]**を選択します。
4. 右側の**「Choose Project」**プロパティページ内で、
   a. **「Categories:」**の下で**[Microchip Embedded]**を選択します。
   b. **「Projects:」**の下で**[Application Project(s)]**を選択します。

図 3-1. 新規プロジェクトの作成



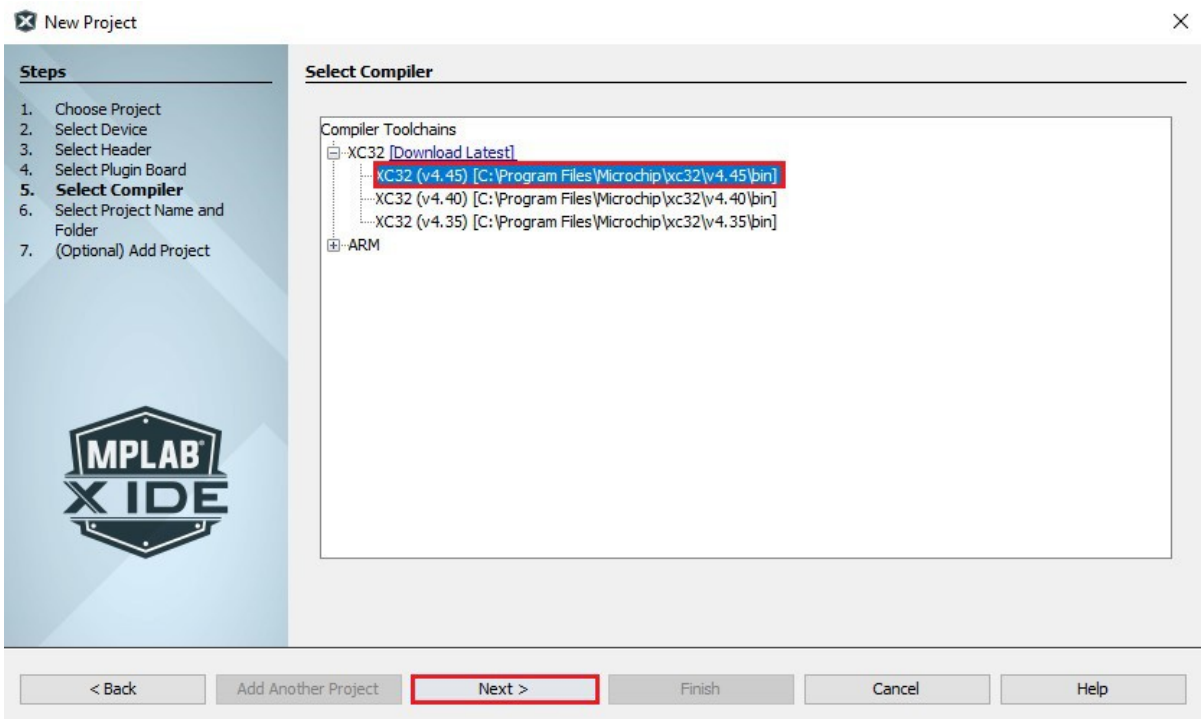
5. **[Next >]**をクリックします。
6. PIC32CM LS00 Curiosity Nano+ Touch 評価用キット上のプロジェクトを作成するため、左側の**「Steps」**の下で**[Select Device]**を選択し、右側の**「Select Device」**プロパティページの**「Device:」**で **PIC32CM5164LS00048** を選択します(図 3-2 参照)。

図 3-2. デバイスの選択



7. **[Next >]**をクリックします。
8. 左側の「**Steps**」の下で**[Select Compiler]**を選択し、右側の「Select Compiler」プロパティページ内で[XC32]を展開して使用する **XC32 コンパイラ**を選択します(図 3-3 参照)。

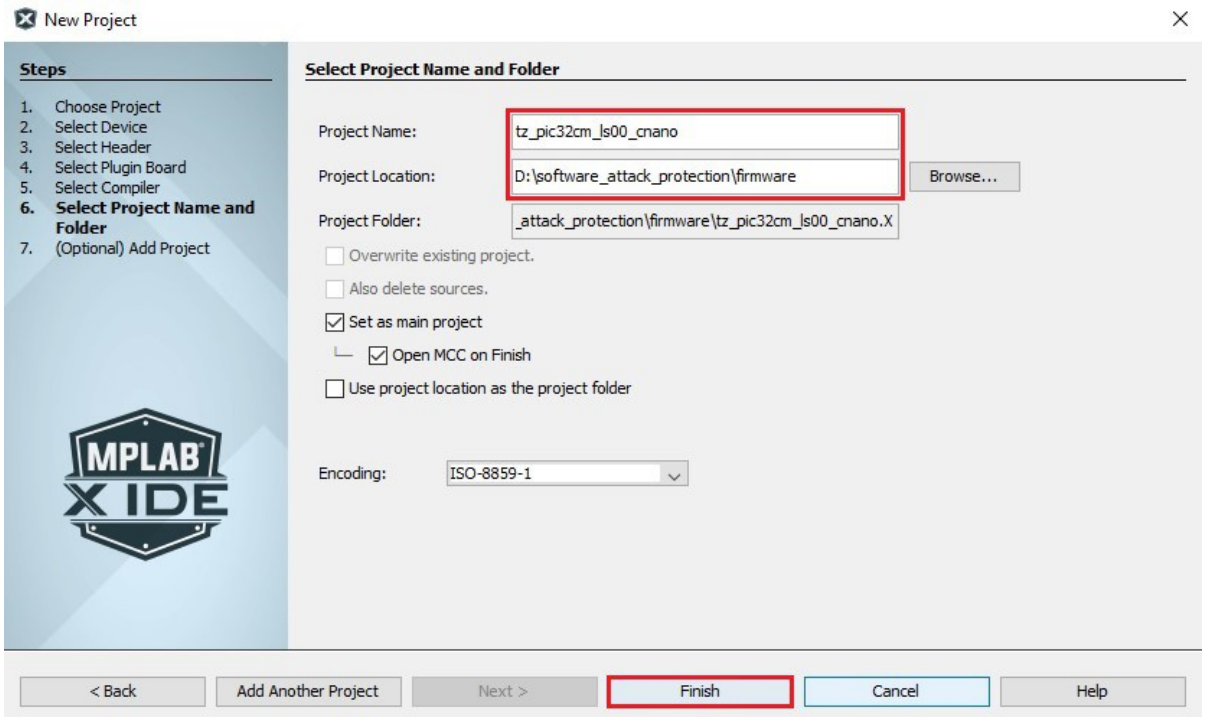
図 3-3. XC32 コンパイラの選択



9. **[Next>]**をクリックします。

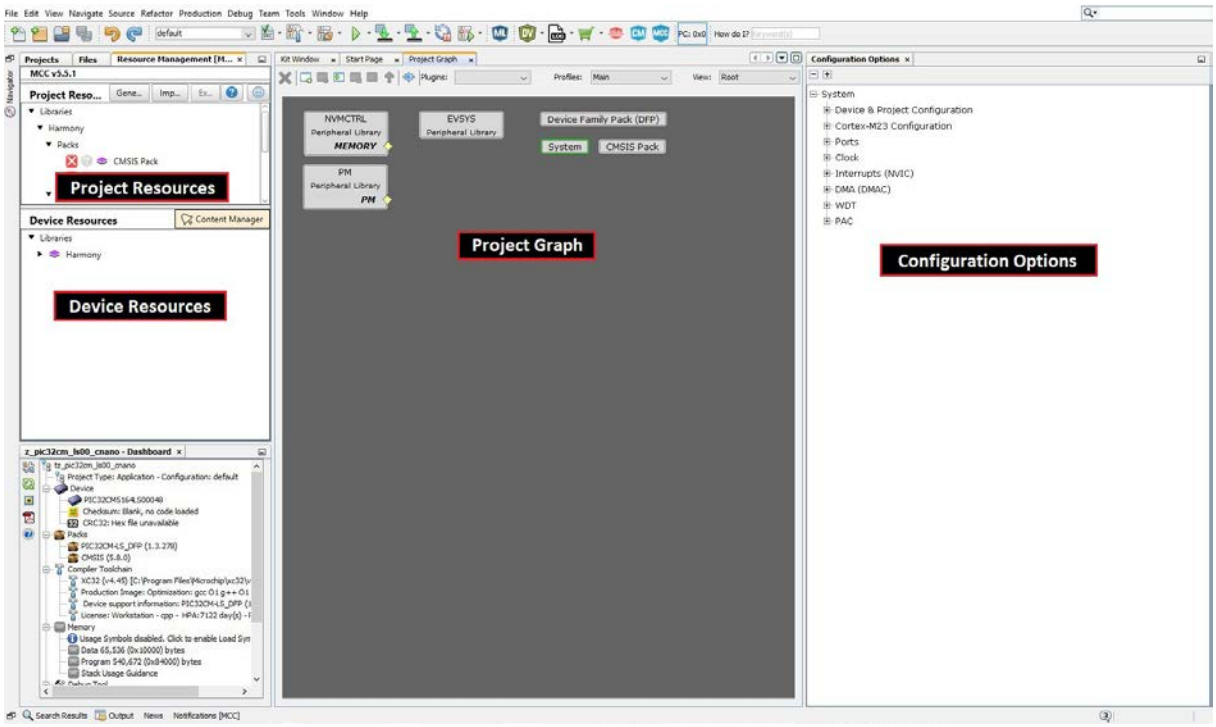
10. 左側の「**Steps**」の下で**[Select Project Name and Folder]**を選択し、右側の「**Select Project Name and Folder**」プロパティページ内で下記を選択します(図 3-4 参照)。
    - **Project Name:** *tz_pic32cm_ls00_cnano* と入力します。これはプロジェクトの名前を設定する際に MPLAB X IDE のプロジェクトに表示される名前です。
    - **Location Project:** *D:\software_attack_protection\firmware* と入力します。これは新規プロジェクトのルートフォルダへのパスです。全てのプロジェクト ファイルはこのフォルダに格納されます。有効な任意のパスを指定できます。
    - **Project Folder:** このフォルダはリードオンリーです。上記の 2 項目の入力に応じて自動的に更新されます。

図 3-4. プロジェクト名とフォルダの設定



11. **[Finish]**をクリックして MCC を起動します。
12. MCC プラグインが新しいウィンドウに表示されます(図 3-5 参照)。

図 3-5. MPLAB® Code Configurator ウィンドウ

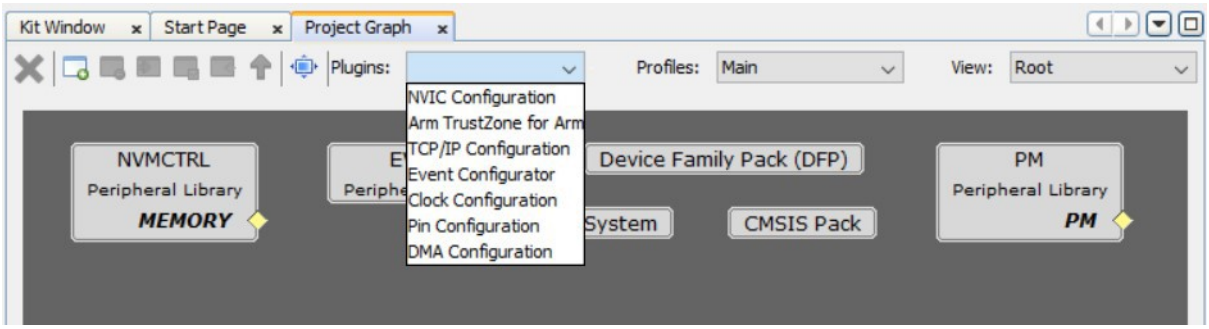


3.1. MPLAB Harmony コンポーネントの追加と設定

MCC を使って MPLAB Harmony コンポーネントを追加および設定するには以下の手順を実行します。

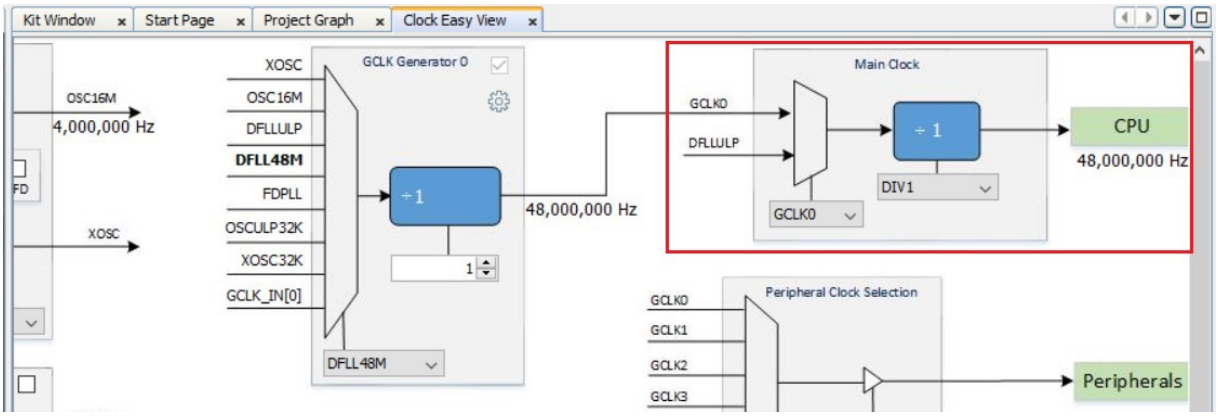
1. MCC ウィンドウ内の**[Plugins:]**ドロップダウン リストから必要な設定ウィンドウを選択します。

図 3-6. MPLAB Code Configurator – プラグインのリスト



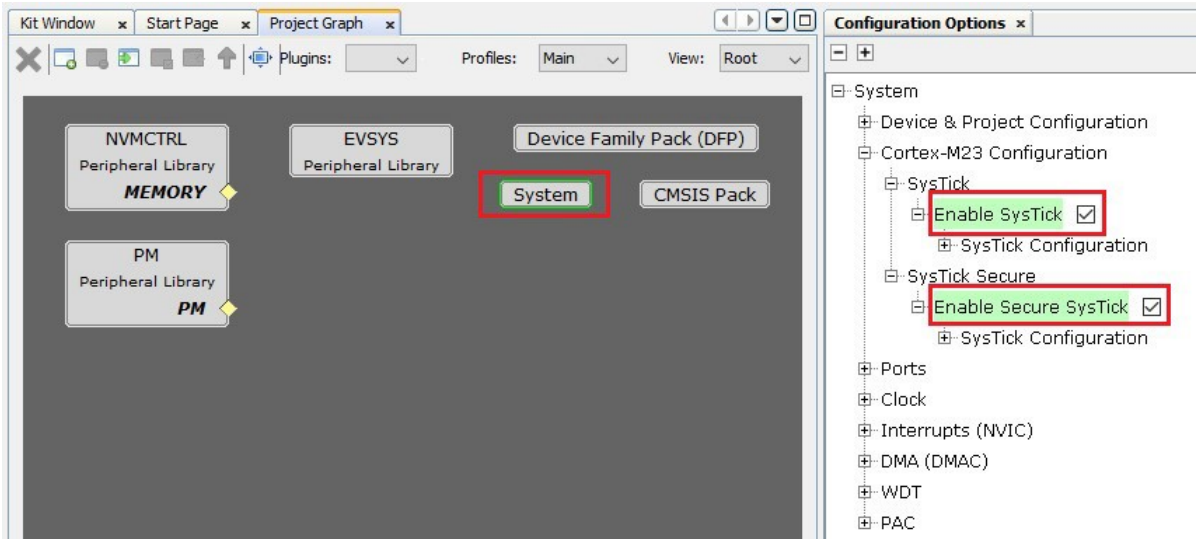
2. ドロップダウン リストから**[Clock Configuration]**を選択して**「Clock Easy View」**ウィンドウを開き、メインクロックが 48 MHz に設定されている事を確認します(図 3-7 参照)。

図 3-7. MPLAB Code Configurator - GCLK Generator 0



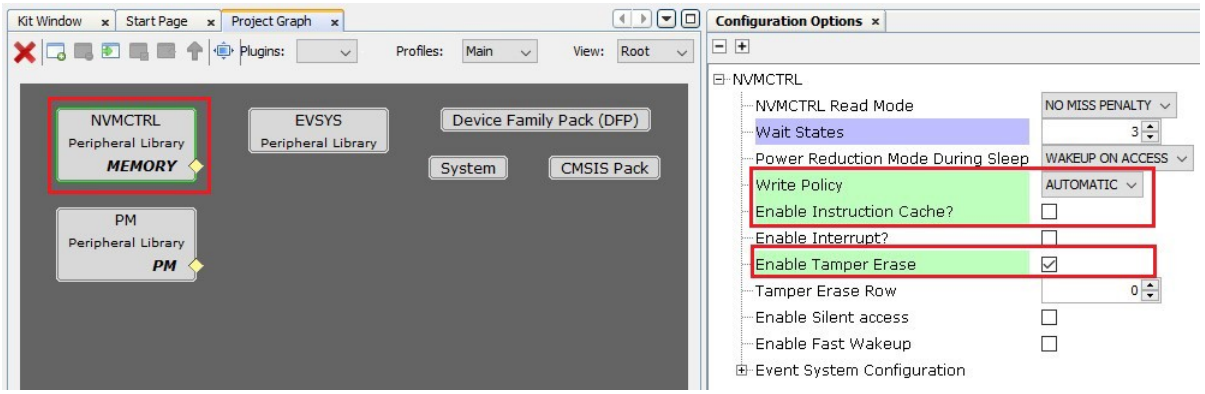
3. **[Project Graph]**をクリックしてから**[System]**モジュールを選択します。右側の**[Configuration Options]**プロパティページ内で図 3-8 の通りに設定する事により、セキュアおよび非セキュア時間遅延用に SysTick タイマを有効にします。

図 3-8. MPLAB Code Configurator – SysTick の設定



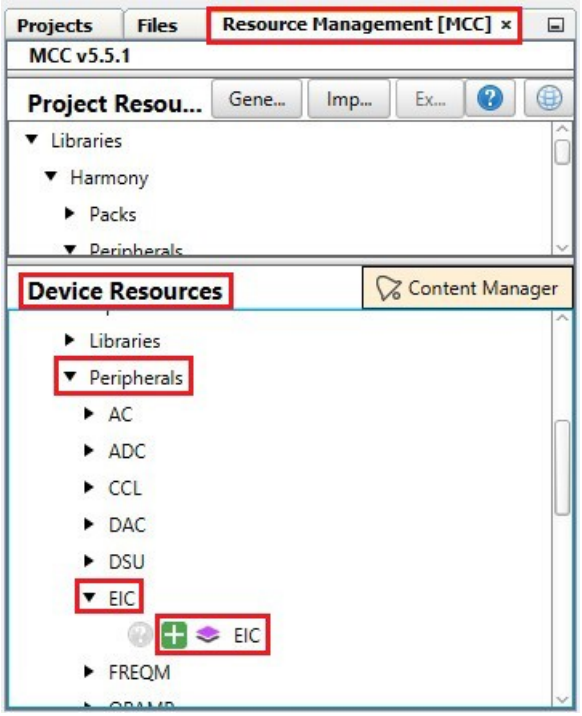
4. 「**Project Graph**」ウィンドウ内で**[NVMCTRL Peripheral Library MEMORY]**を選択し、右側の**[Configuration Options]**プロパティページ内で図 3-9 の通りに設定する事により、タンパー消去機能を有効にします。

図 3-9. MPLAB Code Configurator – NVMCTRL の設定



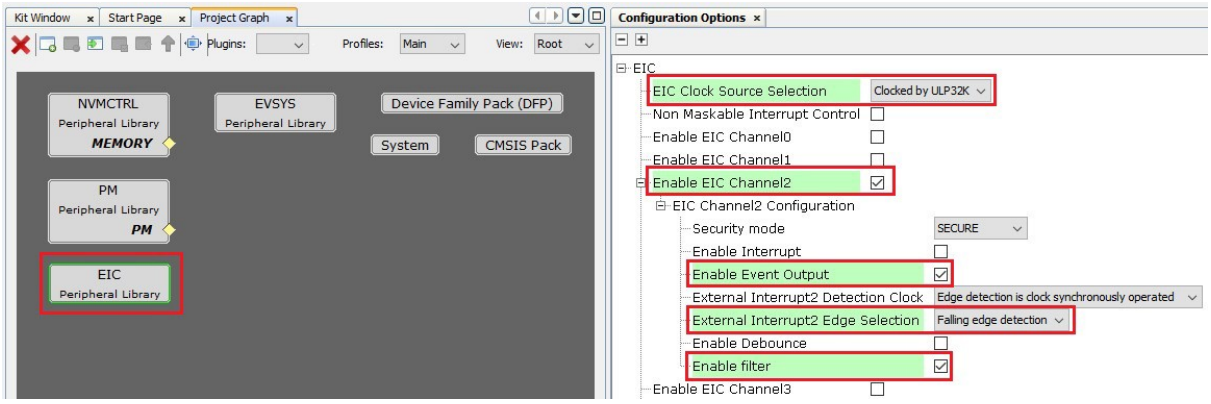
5. **[Resource Management (MCC)]**をクリックし、**[Device Resources]**（図 3-10 参照）の下で *[Peripherals]* > *[EIC]*とクリックして展開します。**[EIC]**をクリックし、**[EIC Peripheral Library]**ブロックが**「Project Graph」**ウィンドウに追加された事を確認します。

図 3-10. MPLAB Code Configurator - EIC モジュールの選択



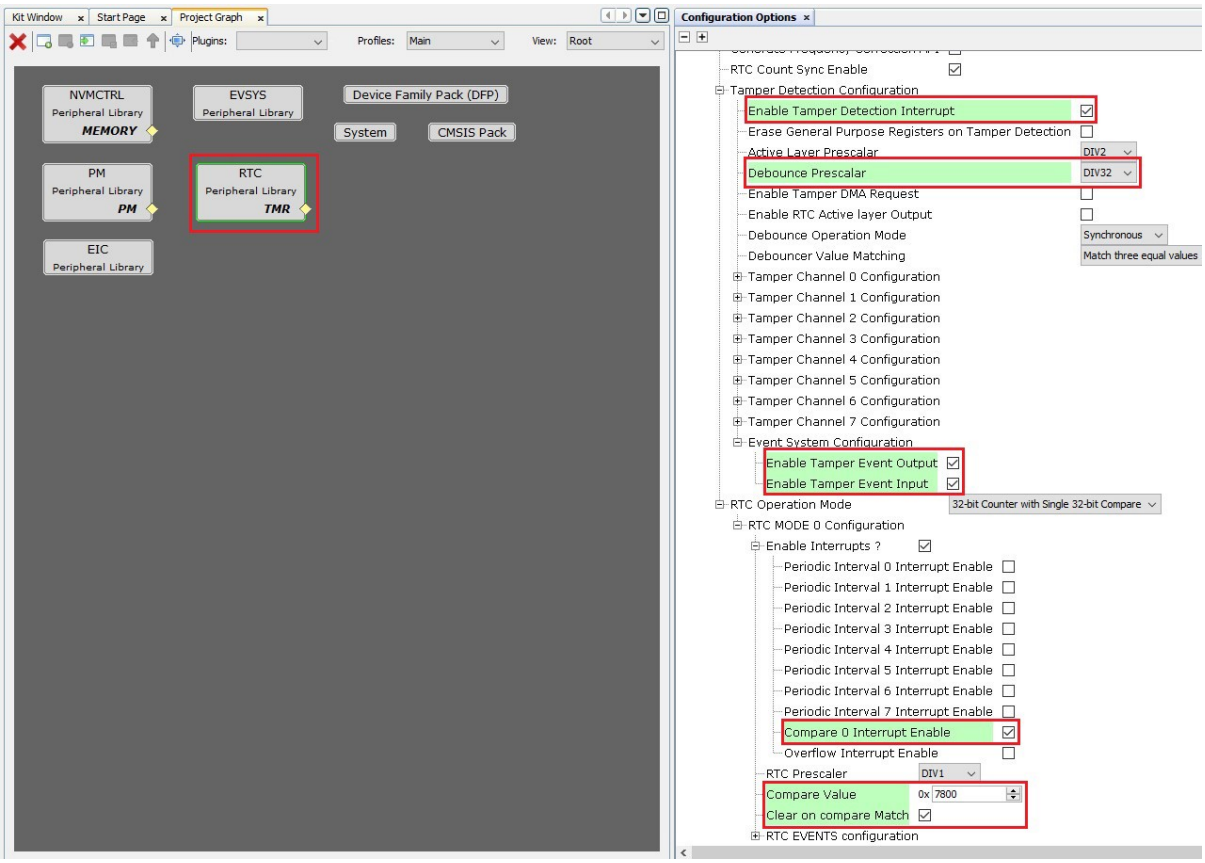
6. **「Project Graph」**ウィンドウ内で**[EIC Peripheral Library]**を選択し、右側の**「Configuration Options」**プロパティ ページ内で図 3-11 に示す通りに設定する事により、EIC チャンネル 2 (SW1) をタンパー入力として使います。

図 3-11. MPLAB Code Configurator - EIC の設定



7. **「Device Resources」**の下で*[Peripherals]* > *[RTC]*とクリックして展開します。**[RTC]**をクリックし、**「RTC Peripheral Library」**ブロックが**「Project Graph」**ウィンドウに追加された事を確認します。
8. **「Project Graph」**ウィンドウ内で**[RTC Peripheral Library]**をクリックし、右側の**「Configurations Options」**プロパティページ内で図 3-12 の通りに設定する事により、コンペア割り込みの生成周期を 30s に設定し、タンパー割り込みおよびイベントを有効にします。

図 3-12. MPLAB Code Configurator - RTC の設定



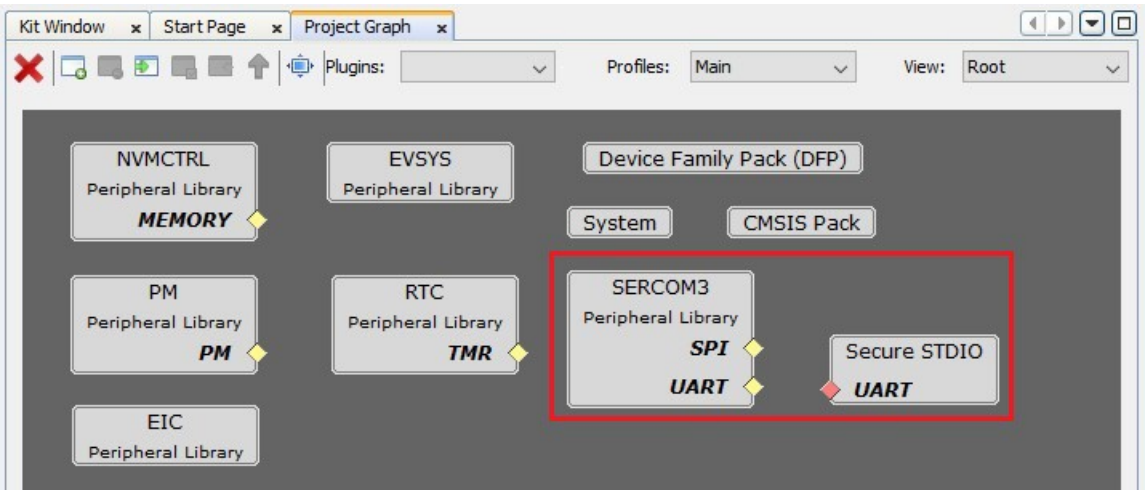
Note: コンペア値は 0x7800 として設定されます。この値により、RTC コンペア割り込みは 30 秒周期で生成されます。

- RTC クロック = 1024 Hz
- RTC プリスケーラ = 1
- 要求割り込みレート = 30s

従って、コンペア値 = 30 x 1024 = 30,720 (= 0x7800)

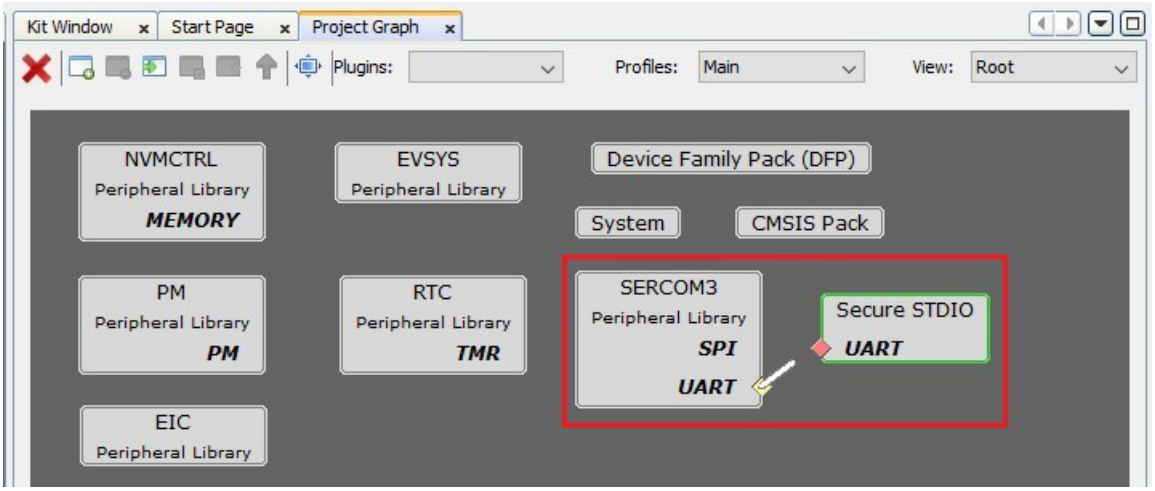
9. 「**Device Resources**」の下で下記を設定します。
   a. *[Peripherals]* > *[SERCOM]*とクリックして展開します。**[SERCOM3]**をクリックし、**[SERCOM3]**ブロックが「**Project Graph**」ウィンドウに追加された事を確認します。
   b. *[Harmony]* > *[Peripherals]* > *[Tools]*とクリックして展開します。**[Secure STDIO]**をクリックし、[Secure STDIO]ブロックが「Project Graph」ウィンドウに追加された事を確認します(図 3-13 参照)。

図 3-13. MPLAB Code Configurator – SERCOM と Secure STDIO の選択



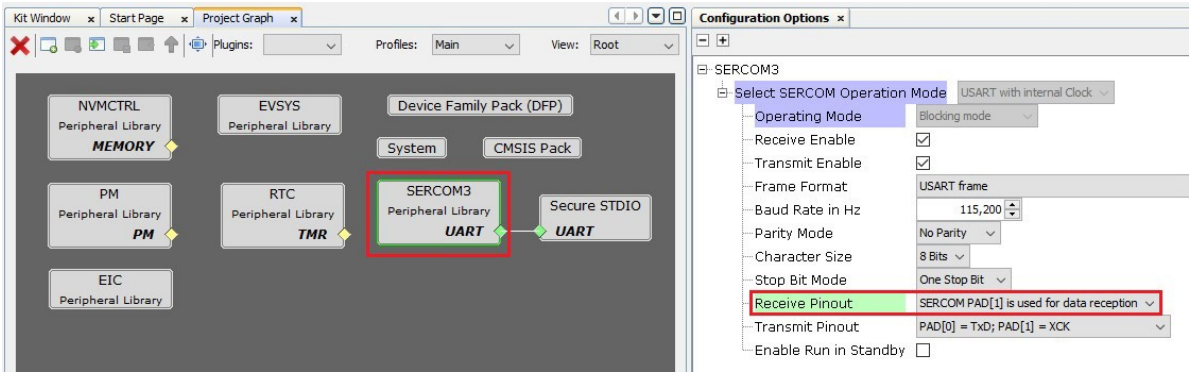
10. 図 3-14 に示す通り、**[SERCOM3]**ブロックと**[Secure STDIO]**ブロックを接続します(SERCOM3 側の UART(黄色の◆)を Secure STDIO 側の UART(赤色の◆) へドラッグ)。

図 3-14. MPLAB Code Configurator – SERCOM と Secure STDIO の接続



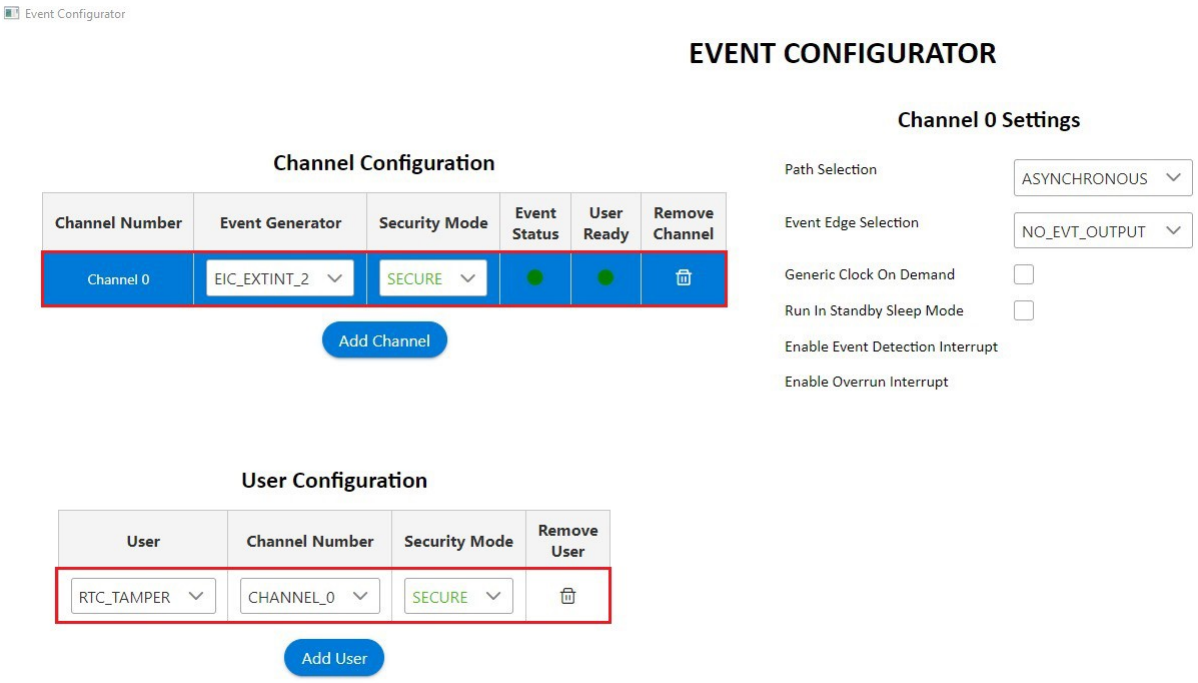
11. 左側の「**Project Graph**」ウィンドウ内で**[SERCOM3 Peripheral Library]**を選択します。右側の「**Configuration Options**」プロパティページ内で図 3-15 に示す通りに設定する事により、データをシリアルコンソール上に 115200 baud レートで出力します。

図 3-15. MPLAB Code Configurator – SERCOM3 の設定



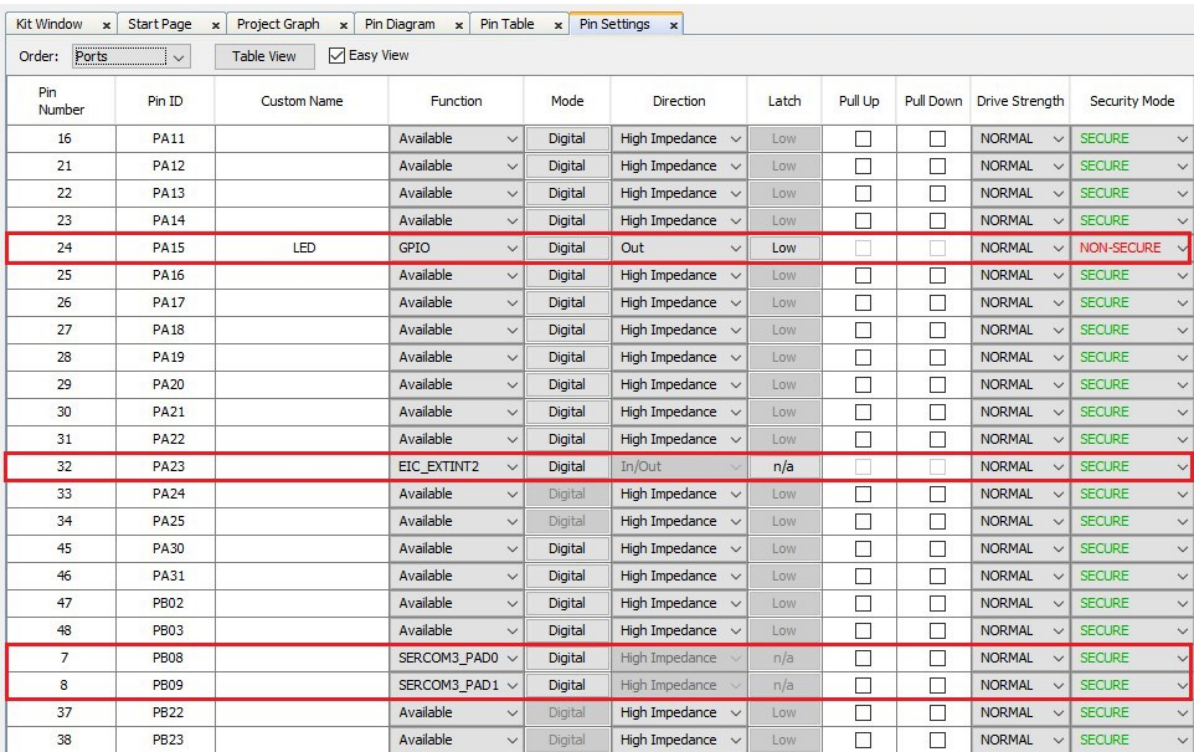
12. **[Plugins]**ドロップダウン リストから**[Event Configurator]**を選択します。図 3-16 に示す通りに、タンパー入力向けにイベント ジェネレータとイベントユーザを追加します。

図 3-16. MPLAB Code Configurator – Event Configurator



13. **[Plugins]**ドロップダウン リストから**[Pin Configuration]**を選択した後に**[Pin Settings]**タブをクリックします。**[Pin Settings]**タブ内の**「Order:」**フィールドの選択を「Ports」に変更します(図 3-17 参照)。アプリケーションに従って図 3-17 の通りにピンを設定します

図 3-17. 「Pin Settings」ウィンドウ - ピンの設定



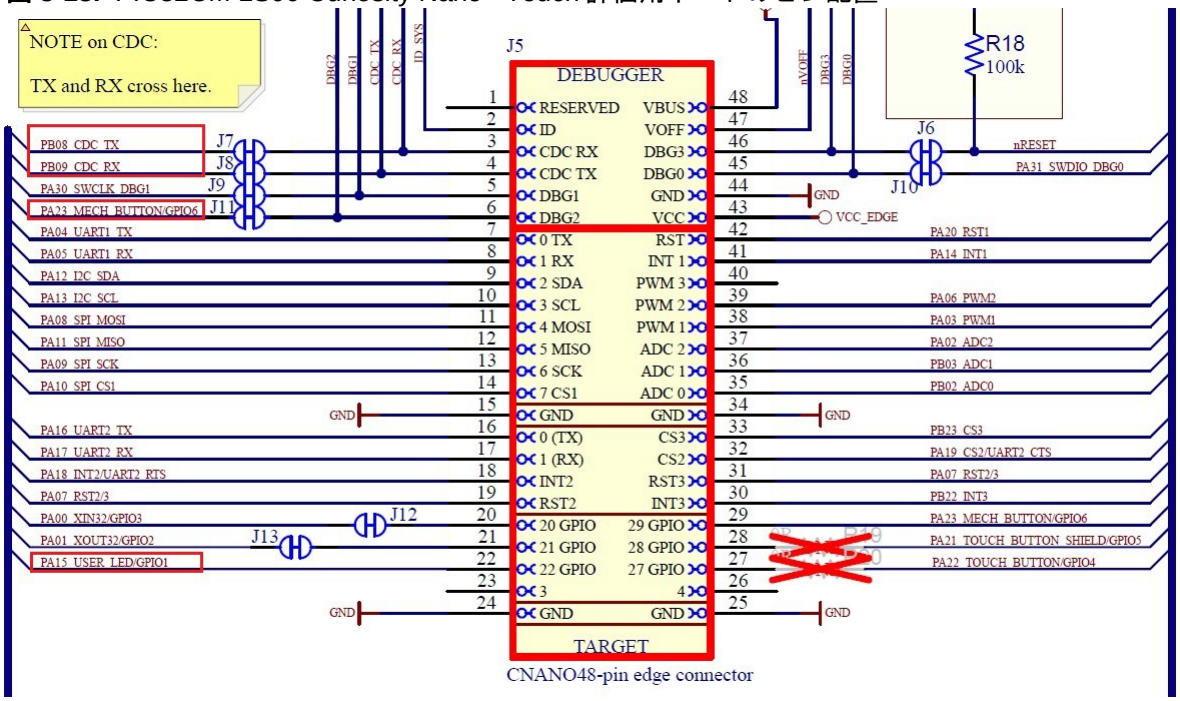
Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch	Pull Up	Pull Down	Drive Strength	Security Mode
16	PA11		Available	Digital	High Impedance	Low	☐	☐	NORMAL	SECURE
21	PA12		Available	Digital	High Impedance	Low	☐	☐	NORMAL	SECURE
22	PA13		Available	Digital	High Impedance	Low	☐	☐	NORMAL	SECURE
23	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

Note:

- PB08、PB09: SERCOM3 TX および RX ピン
- PA15: LED
- PA23: スイッチ

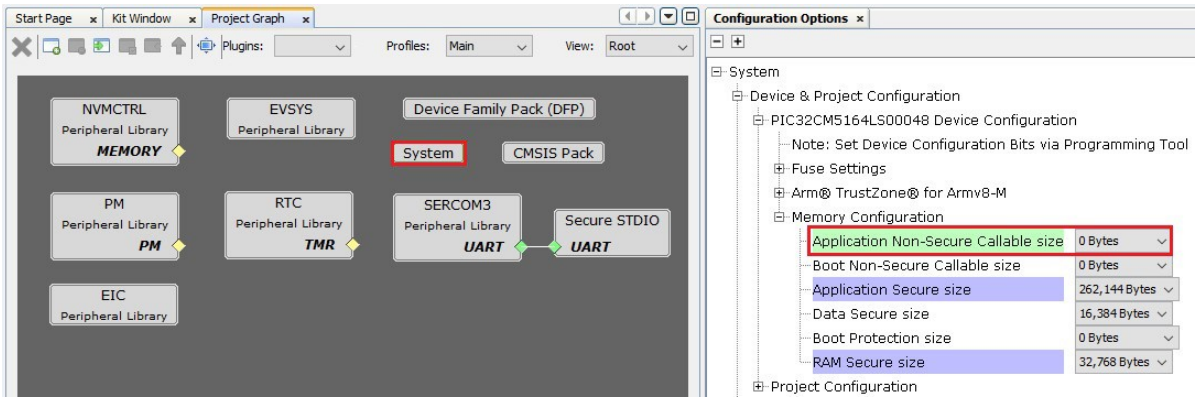
追加の情報は『PIC32CM LS00 Curiosity Nano+ Touch Evaluation Kit User Guide』 (DS70005567)を参照してください。

図 3-18. PIC32CM LS00 Curiosity Nano+ Touch 評価用ボードのピン配置



14. 「**Project Graph**」ウィンドウ内で**[System]**を選択します。「**Configuration Options**」プロパティページ内で図 3-19 の通りに設定します([Memory Configuration]の下の[Application Non-Secure Callable Size]を 0 バイトに設定)。

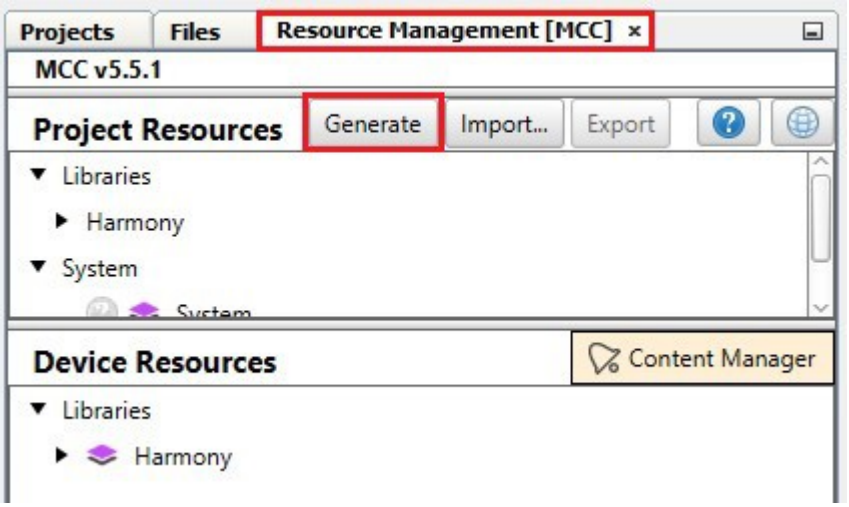
図 3-19. MPLAB Code Configurator – メモリ設定



3.2. コードの生成

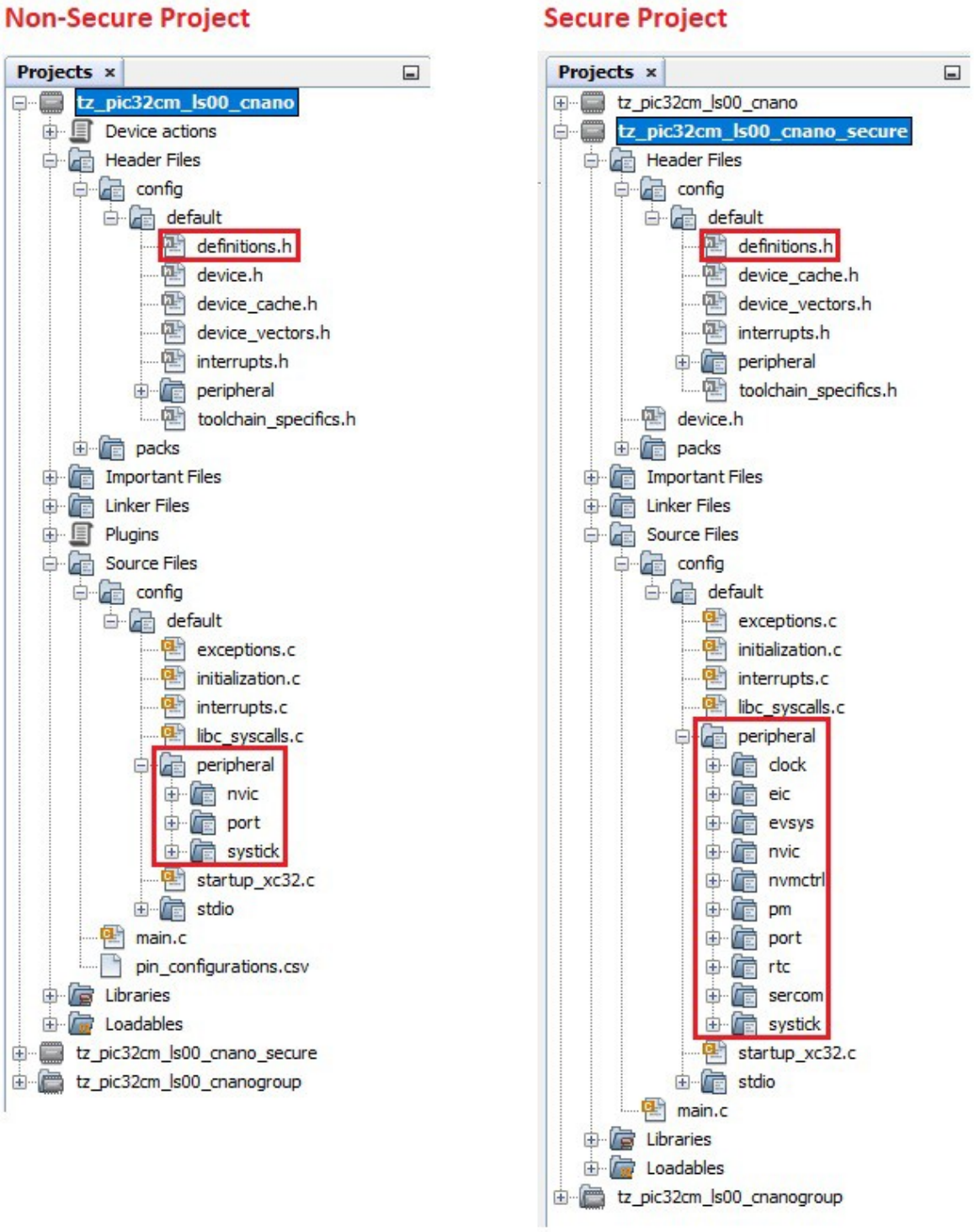
1. 周辺モジュールを設定した後に、**[Resource Management [MCC]]** をクリックしてから**[Generate]** タブをクリックします(図 3-20 参照)。

図 3-20. コードの生成



2. コードが生成されると、32 ビット MCC Harmony v3 プロジェクトにファイルとフォルダが追加されます。生成されたコード内で SysTick、SERCOM、EIC、NVMCTRL、RTC、Event System、PORT モジュール向けの周辺モジュール ライブラリ ファイルが生成されている事を確認します。

図 3-21. 非セキュアおよびセキュア プロジェクトで生成されたコード



Note:

- MCC はセキュア プロジェクトと非セキュア プロジェクト向けに別々の `main.c` を生成します。
- MCC では生成されるファイルの名前をオプションで変更できます。変更しない場合、既定値でファイル名 `main.c` が生成されます。

4. 非セキュアおよびセキュア プロジェクトに対するアプリケーション ロジックの追加

4.1. 非セキュア アプリケーション ロジックの追加

アプリケーションを開発して実行するには以下の手順を実行します。

1. 非セキュア プロジェクト(tz_pic32cm_ls00_cnano.X)の `main.c` ファイルを開き、下記のコードを `SYS_Initialize()` の後に追加します。

```
SYSTICK_TimerStart();
```

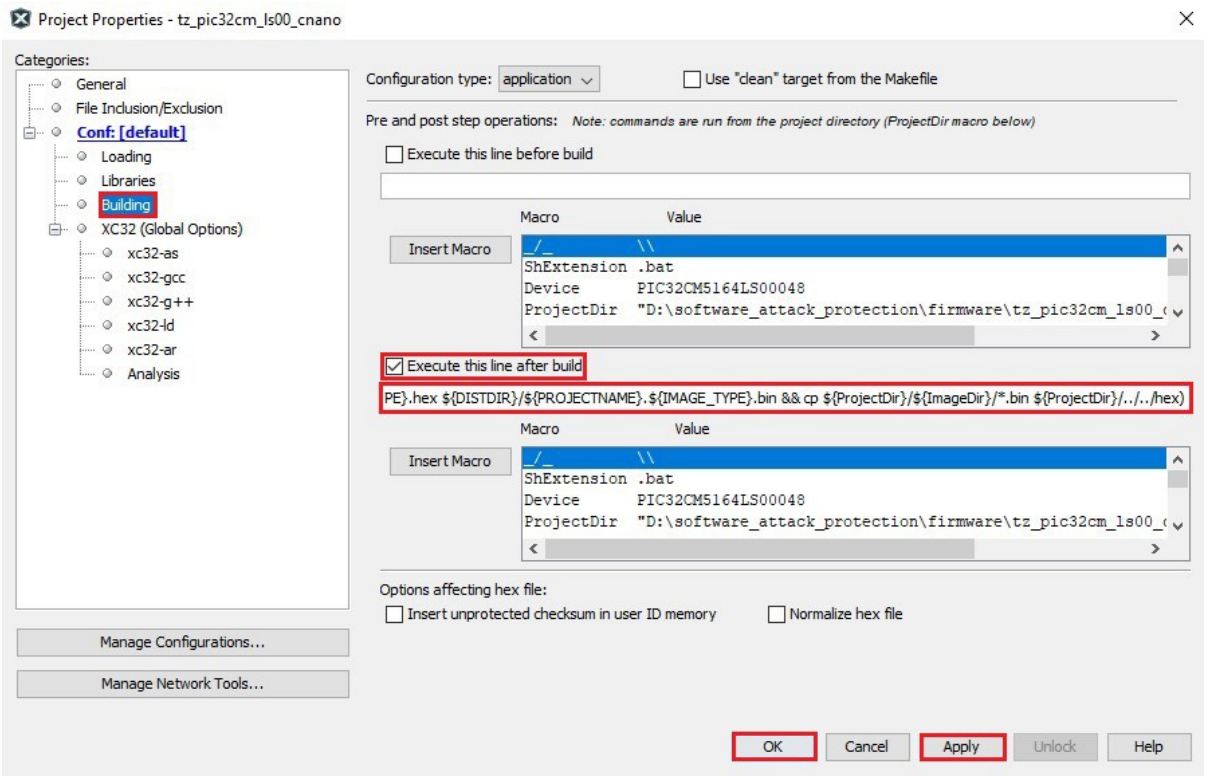
2. LED を既定値レート(500 ms)でトグルするために、`while` ループ内に下記のコードを追加します。

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

3. 「Non-Secure Project Properties」へ移動し、非セキュア ファームウェアの真正コピーを生成するためのポストビルド コマンドを入力します。
   a. MPLAB X IDE の「Project Properties」ウィンドウ内で下記を実行します(図 4-1 参照)。
   b. 左側の**「Categories:」**の下で**[Building]**を選択し、右側の**「Configuration」**プロパティページ内で**[Execute this line after build]**チェックボックスを選択します。
   c. チェックボックスの下に下記のポストコマンドを入力します。

```
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
```

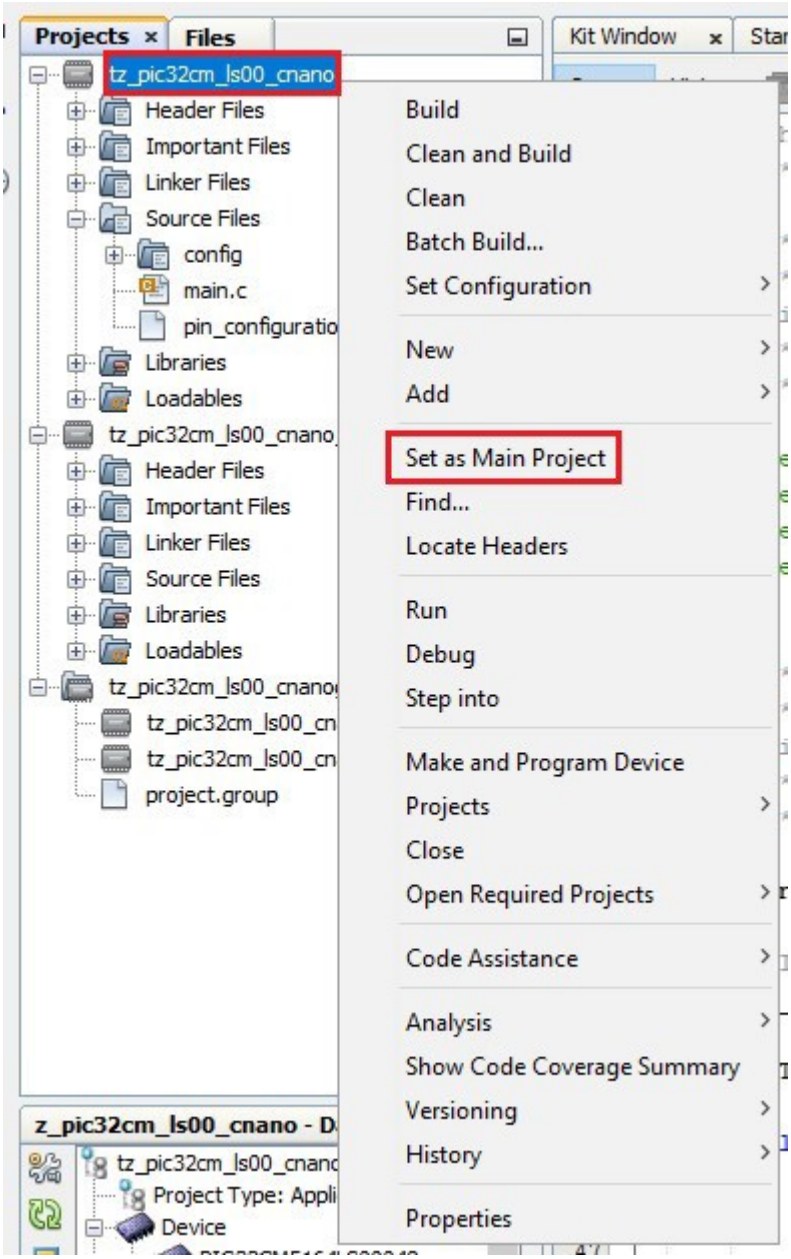
図 4-1. 非セキュア ファームウェアの真正コピーの生成



4. **[Apply]**をクリックしてから**[OK]**をクリックします。

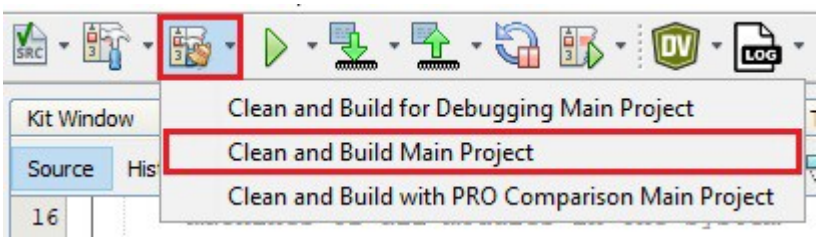
5. 「**Projects**」の下で**[tz_pic32cm_ls00_cnano]**をクリックしてから**[Set as Main Project]**を選択します。

図 4-2. 非セキュアプロジェクトを main プロジェクトとして設定する



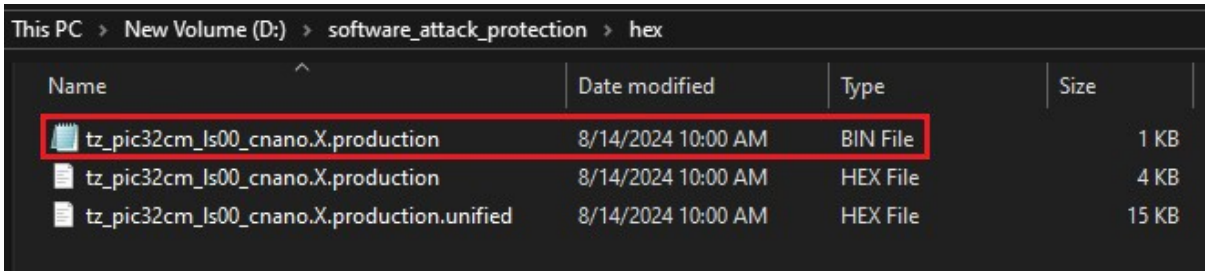
6. **[Clean and Build]**アイコンをクリックするか、ドロップダウン リストから**[Clean and Build Main Project]**を選択する事によりプロジェクトをビルドし、ビルドが成功したかどうか確認します。

図 4-3. クリーンとビルド



7. 非セキュア プロジェクトのバイナリファイルが「hex」フォルダ内(パス: *D:/software_attack_protection/hex*)に格納されている事を確認します。

図 4-4. 生成されたバイナリファイルの格納位置



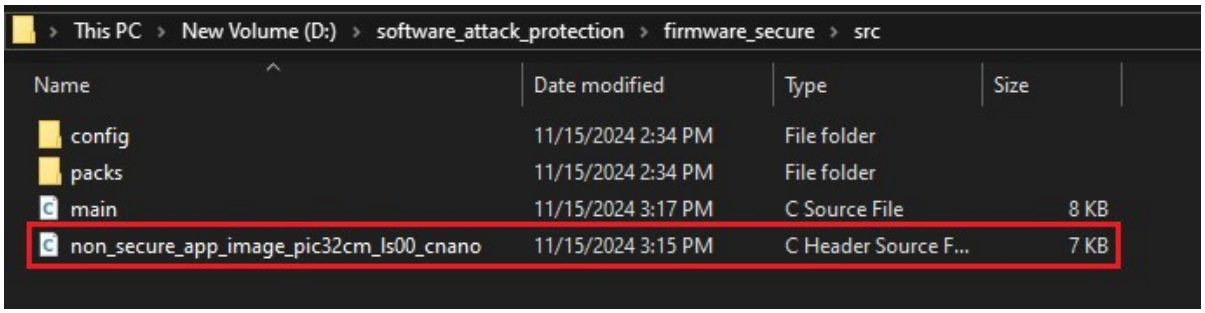
8. コマンド プロンプトを開き、下記のパスへ移動します。
パス: *<Harmony フォルダのパス>/ bootloader/tools*
Note: 「bootloader」フォルダが「Harmony」フォルダ内に存在しない場合、MPLAB Content Manager を使ってブートローダ パッケージ(v3.7.0 以上)をダウンロードします。
9. Python スクリプト `btl_bin_to_c_array.py` を実行する事により、非セキュア アプリケーションのバイナリファイルを Hex 出力を格納した C 言語形式の配列へ変換します。

```
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
```

図 4-5. Python スクリプトの実行

10. スクリプトが正常に実行されると、非セキュア アプリケーション イメージ(真正コピー)のヘッダファイルがセキュア プロジェクトの「source」フォルダに格納されます。

図 4-6. 非セキュア アプリケーション イメージの真正コピー



4.2. セキュア アプリケーション ロジックの追加

アプリケーションを開発して実行するには以下の手順を実行します。

1. `main.c` ファイル内で、セキュア アプリケーションによって使われる変数とマクロを下記の通りに宣言します。

```
#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];
```

図 4-7. 変数とマクロの宣言

```
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. 下記の通りに main.c ファイル内にブート ROM API を追加してそれらへアクセスできるようにします。

```
typedef struct
{
    /* Digest result of SHA256 */
    uint32_t digest[8];
    /* Length of the message */
    uint64_t length;
    /* Holds the size of the remaining part of data */
    uint32_t remain_size;
    /* Buffer of remaining part of data (512 bits data block) */
    uint8_t remain_ram[64];
    /* RAM buffer of 256 bytes used by crya_sha_process */
    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. 非セキュア フラッシュ領域に非セキュア ファームウェアをプログラミングしファームウェア ダイジェストをセキュアデータ フラッシュに書き込むために、flash_write API を main.c ファイルに組み込みます。

```
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)
    {
        /* Unlock the Secure Data Flash region */
        NVMCTRL_RegionUnlock(NVMCTRL_SECURE_MEMORY_REGION_DATA);
        while(NVMCTRL_IsBusy());
    }

    else
    {
        /* Unlock the Non-Secure Flash region */
        NVMCTRL_RegionUnlock(NVMCTRL_MEMORY_REGION_APPLICATION);
        while(NVMCTRL_IsBusy());
    }

    do
    {
        if(addr % NVMCTRL_FLASH_ROWSIZE == 0)
        {
            /* Erase the row */
            NVMCTRL_RowErase(addr);
            while(NVMCTRL_IsBusy());
        }

        /* Program 64 byte page */
        NVMCTRL_PageWrite((uint32_t *)(buf), addr);
        while(NVMCTRL_IsBusy());

        addr += NVMCTRL_FLASH_PAGESIZE;
        buf += NVMCTRL_FLASH_PAGESIZE;

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

4. 非セキュア ファームウェア ダイジェストを計算するために、SHA-256 ハッシュ API と非セキュア ファームウェア検証 API を `main.c` ファイルに組み込みます。

```
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. タンパー割り込みと 30s タイムアウトのために、RTC コールバックを `main.c` ファイルに組み込みます。

```
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. 下記の短いコードを `main.c` 内の SYS_Initialize API の後に追加します。

```
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));
```

Note:

- `sha256_hash`: 非セキュア ファームウェアのダイジェストを計算します。
- `flash_write`: ファームウェア ダイジェストをセキュアデータ フラッシュ領域に保存します。

```
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");
    }
```

Note:

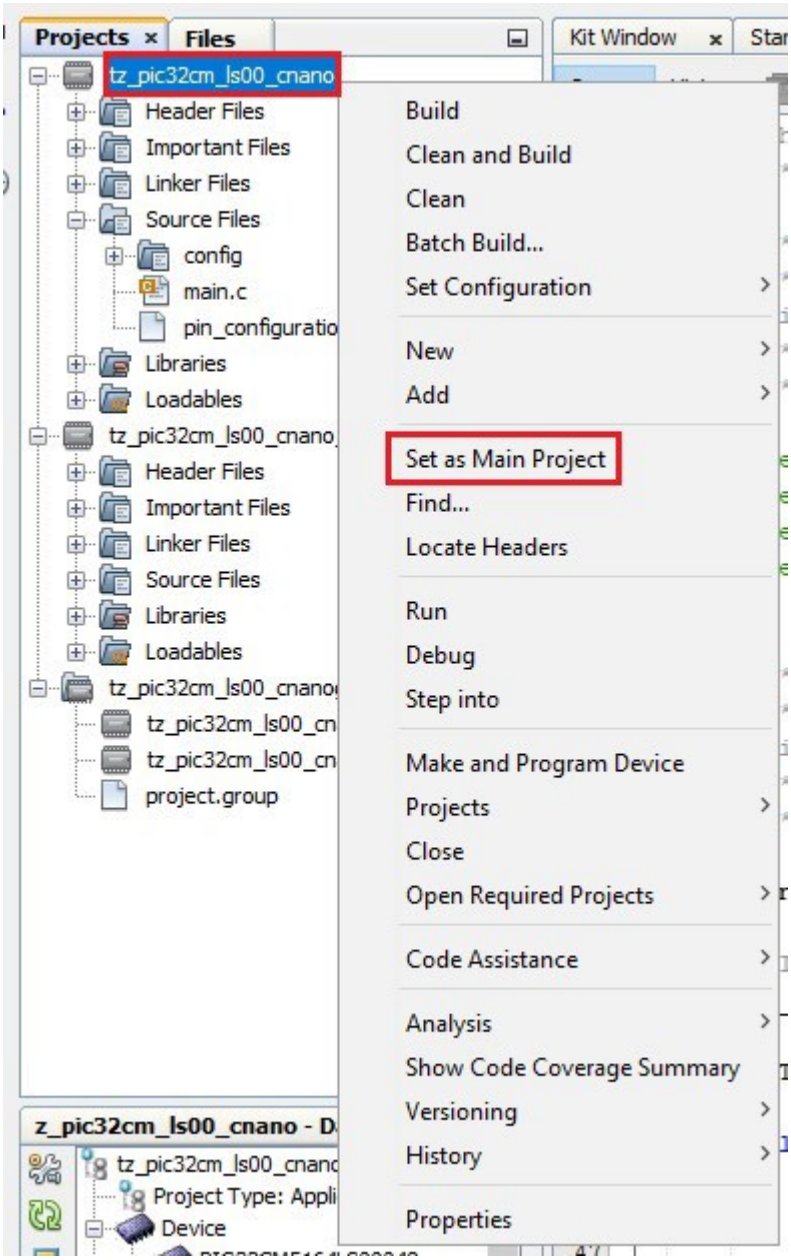
- non_secure_app_verify: 非セキュア ファームウェアを検証し、検証に失敗した場合は真正コピーを非セキュア フラッシュ領域にプログラミングします。

5. アプリケーションのビルドと実行

以下の手順により、ソフトウェア攻撃対策アプリケーションを PIC32CM LS00 Curiosity Nano+ Touch 評価用キット上でプログラミングします。

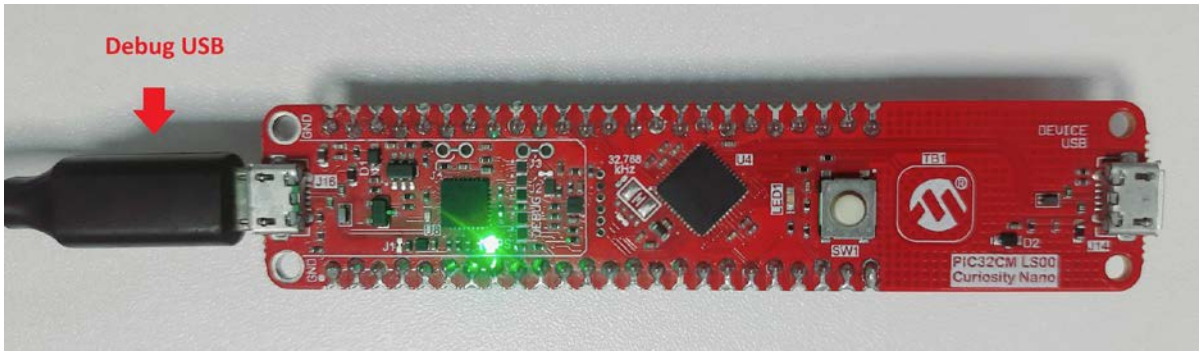
1. **tz_pic32cm_ls00_cnano** プロジェクトを右クリックして**[Set as Main Project]**を選択する事により、このプロジェクトを Main プロジェクトとして設定します。

   図 5-1. 非セキュアプロジェクトを Main プロジェクトとして設定する



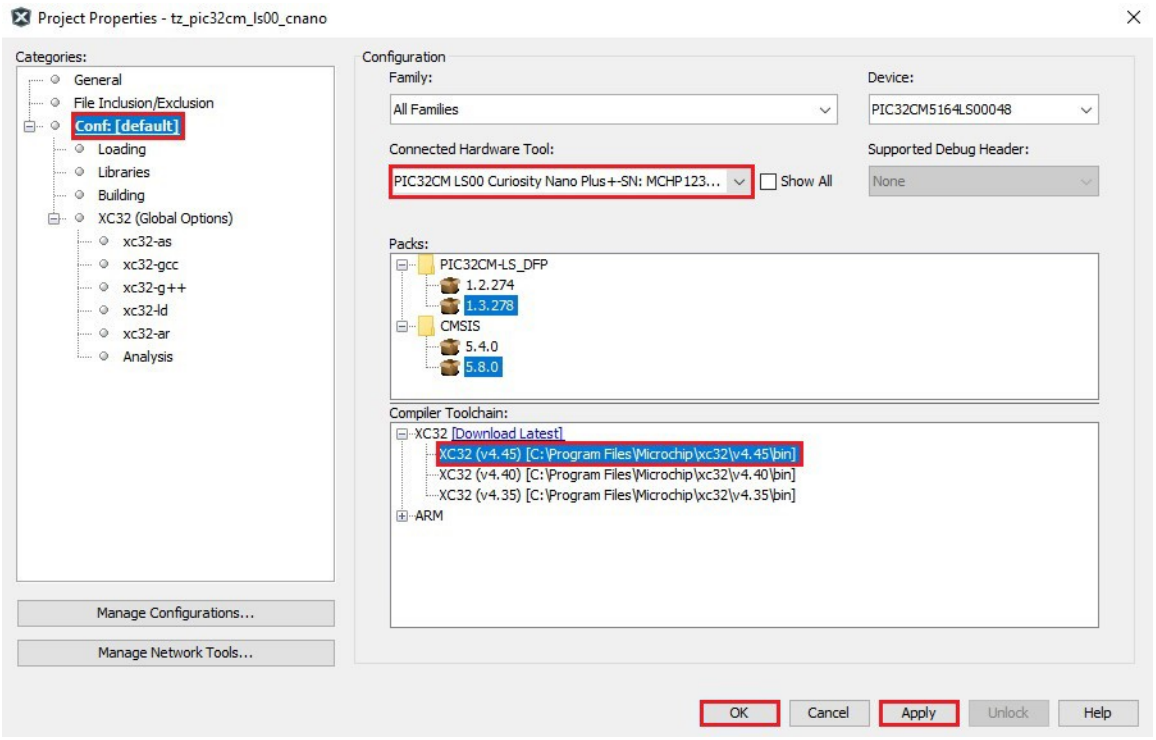
2. PIC32CM LS00 Curiosity Nano+ Touch 評価用キットは、Nano 組み込みデバッガ(nEDBG)を使ったデバッグをサポートします。PIC32CM LS00 Curiosity Nano+ Touch 評価用キットへの給電とデバッグのために、Type-A オス- micro-B USB ケーブルをキット上の micro-B USB ポートに接続します。

図 5-2. PIC32CM LS00 Curiosity Nano+ Touch 評価用キットのハードウェア セットアップ



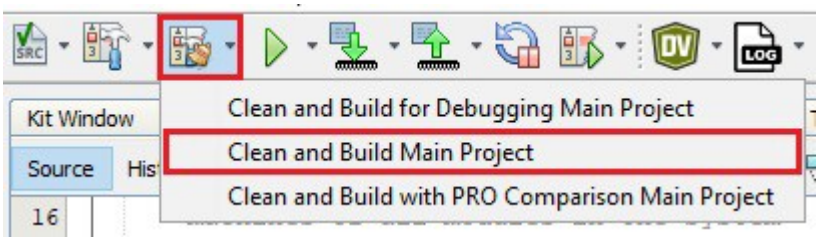
3. 「**Project Properties**」へ移動してハードウェア ツールとコンパイラを選択します。
   a. MPLAB X IDE の「**Project Properties**」ウィンドウ内で下記を実行します。
   b. 左側の「**Categories:**」の下で**[Conf: [default]]**を選択し、右側の「**Configuration** 」プロパティシート内で接続したハードウェアツールとコンパイラ ツールチェーンを選択します(図 5-3 参照)。

図 5-3. Project Properties - PIC32CM LS00 Curiosity Nano+Touch 評価用キット



4. **[Apply]**をクリックしてから**[OK]**をクリックします。
5. **[Clean and Build]**アイコンをクリックするか、ドロップダウン リストから**[Clean and Build Main Project]**を選択する事によりプロジェクトをビルドし、ビルドが成功したかどうか確認します。

図 5-4. Clean and Build



6. 図 5-5 内の赤枠で囲んだアイコンをクリックする事によりアプリケーションをプログラミングします。

図 5-5. デバイスのプログラミング

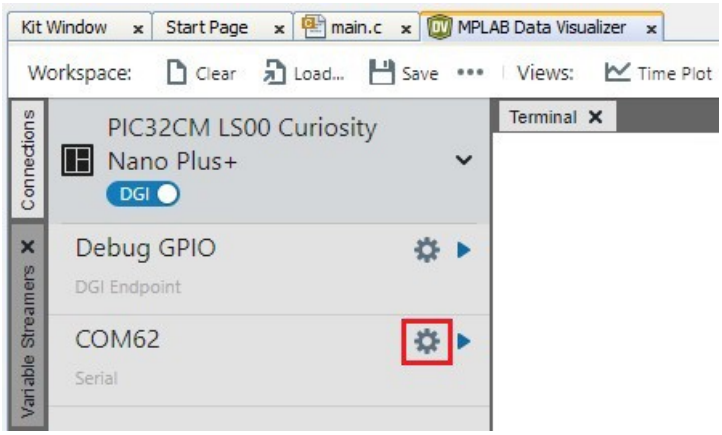
6. MPLAB Data Visualizer による出力の観察

1. アプリケーションをビルドしてプログラミングが完了した後に、図 6-1 内の赤枠で囲んだアイコンをクリックする事により、MPLAB Data Visualizer を開きます。

図 6-1. MPLAB® Data Visualizer の起動

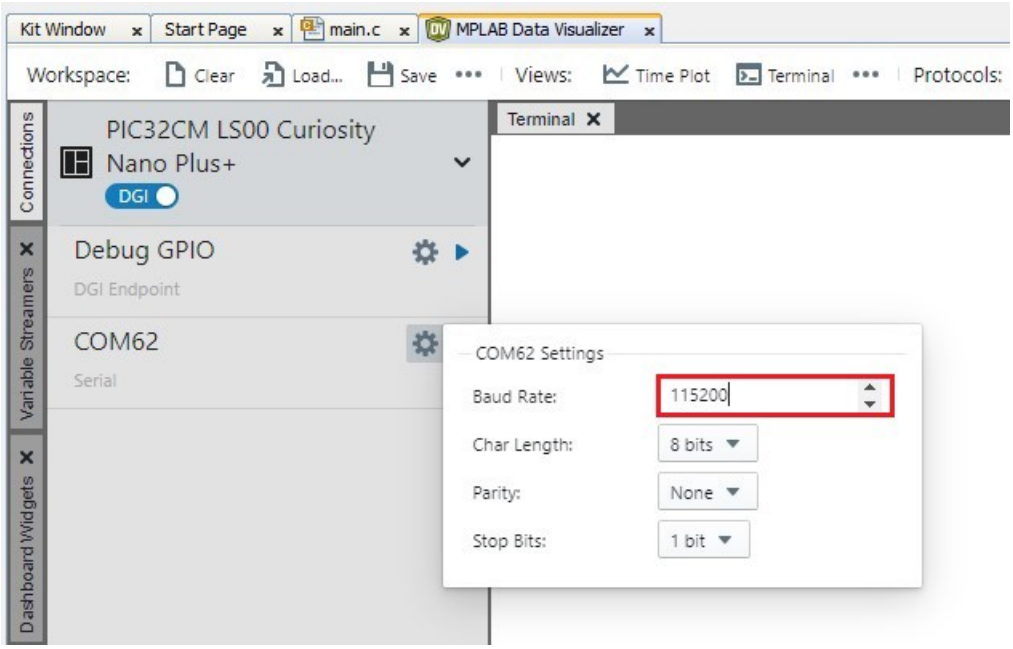
2. 図 6-2 内の**ギア**アイコンをクリックする事により、PIC32CM LS00 Curiosity Nano+ Touch 評価用キットのシリアルポートを設定します。

図 6-2. シリアルポートの設定



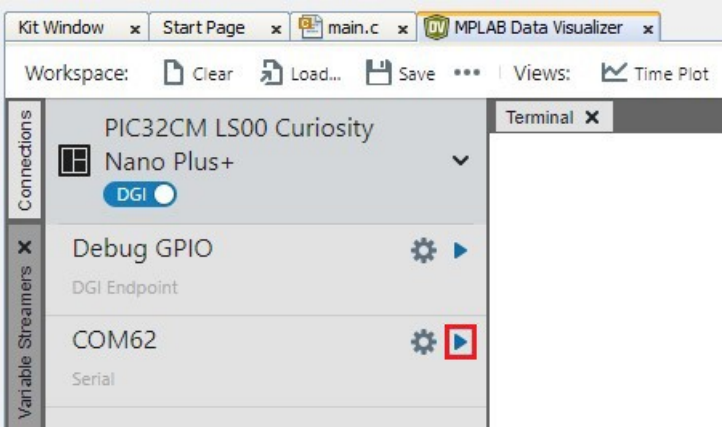
3. **「COM Settings」**内で baud レートを 115200 に設定します。

図 6-3. baud レートの設定



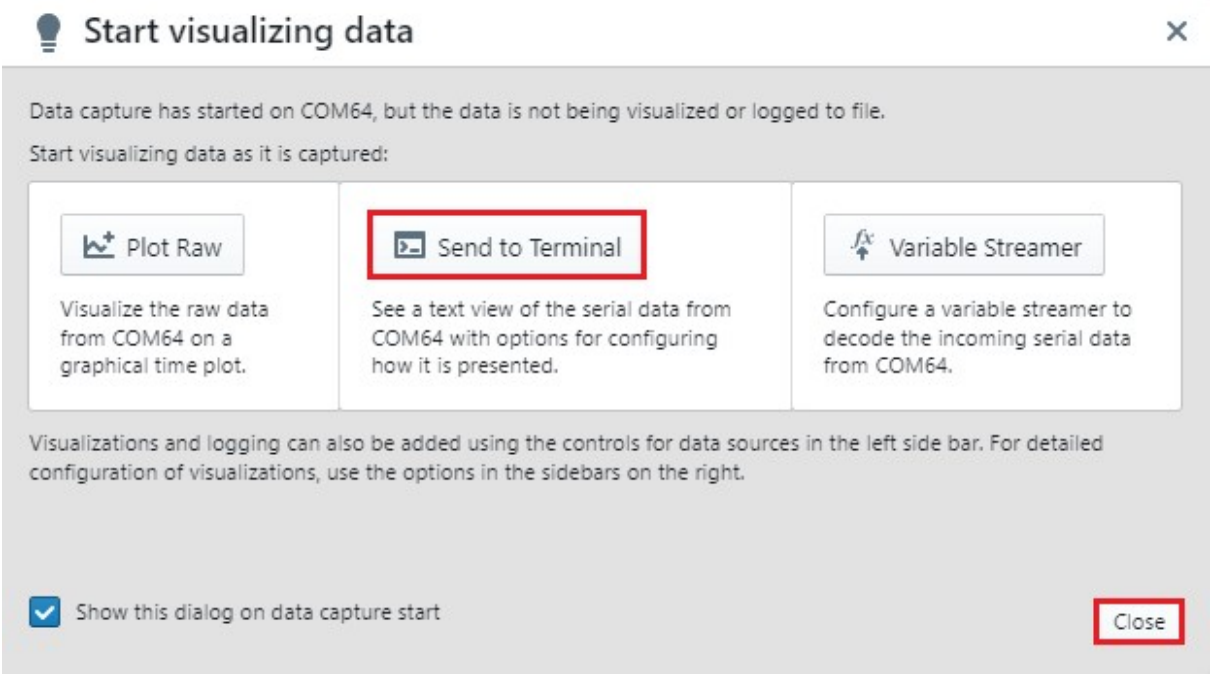
4. 図 6-4 内の右向き三角アイコンをクリックする事により、PIC32CM LS00 Curiosity Nano+ Touch 評価用キットのシリアルポートを開きます。

図 6-4. シリアル COM ポートを開く



5. **[Send to Terminal]**をクリックし、シリアル コンソール メッセージを確認した後に**[Close]**をクリックします。

図 6-5. ターミナル オプションの選択



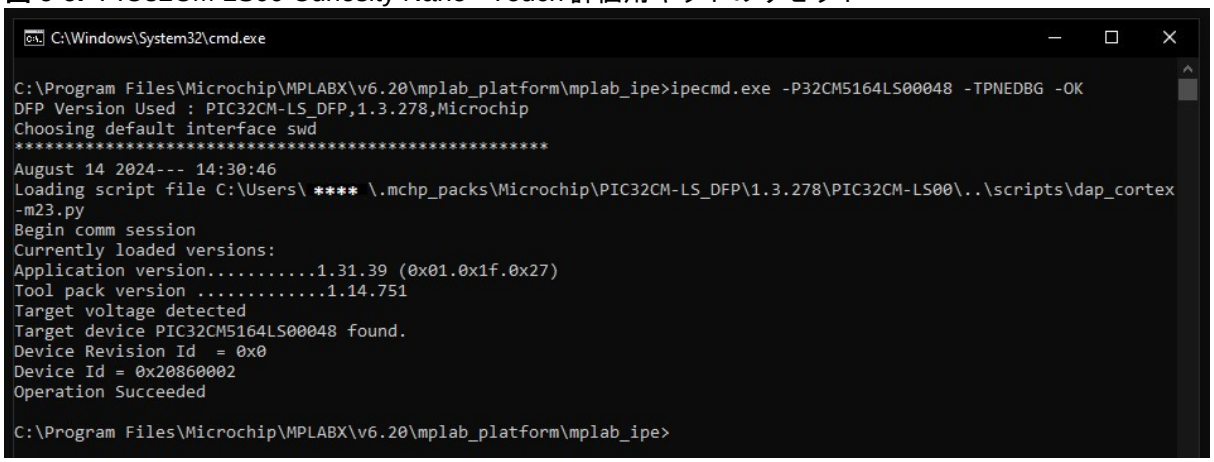
6. コマンド プロンプトを開き、下記のパスへ移動します。
   C:\Program Files\Microchip\MPLABX\v6.20\mplab_platform\mplab_ipe.

   Note: PIC32CM LS00 Curiosity Nano+ Touch 評価用キットは、MCU をリセットするためのボタンを備えていません。本ボードをリブートするには、MPLAB IPECMD を使って `reset` コマンドを nEDBG へ送信する事により MCU をリセットします。

7. PIC32CM LS00 Curiosity Nano+ Touch 評価用キットをリセットするには下記のコマンドを実行します。

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

図 6-6. PIC32CM LS00 Curiosity Nano+ Touch 評価用キットのリセット



8. MPLAB Data Visualizer 上の起動時コンソール メッセージと、PIC32CM LS00 Curiosity Nano+ Touch 評価用キット上の LED1 のトグルを確認します。

図 6-7. 起動時コンソール メッセージ

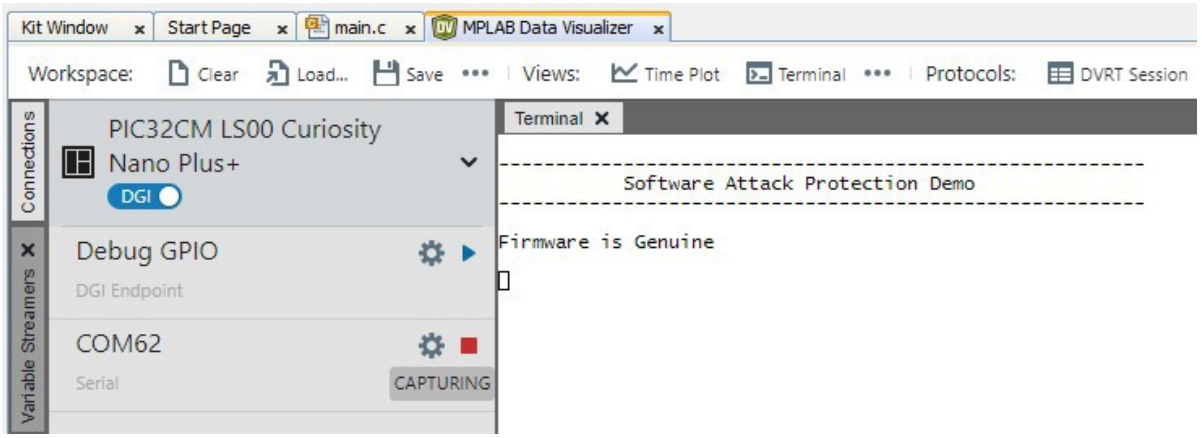
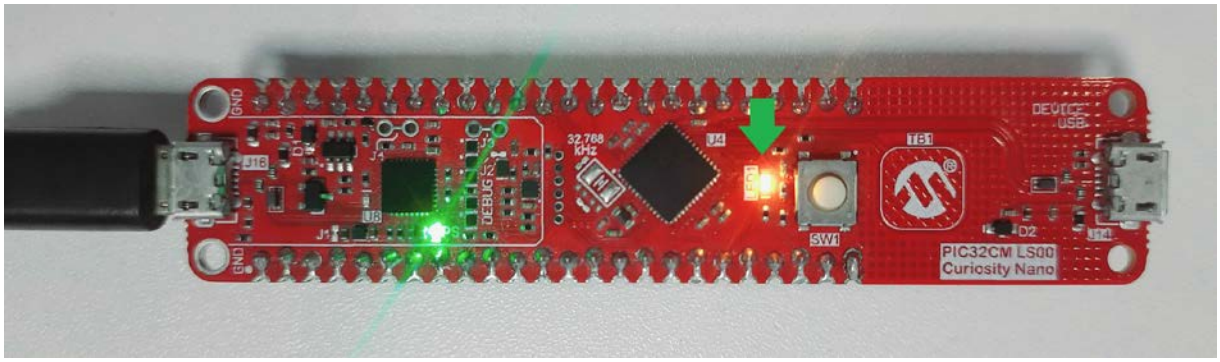
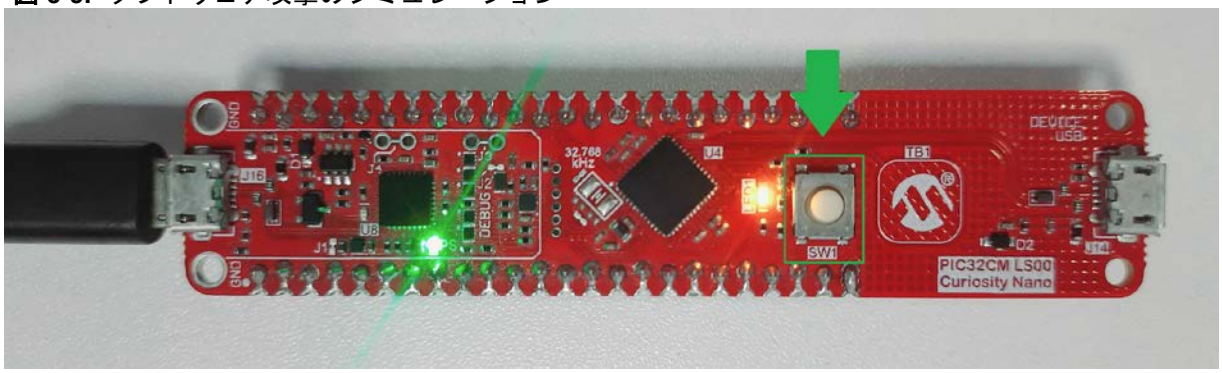


図 6-8. PIC32CM LS00 Curiosity Nano+ Touch 評価用キット上の LDE1 のトグル



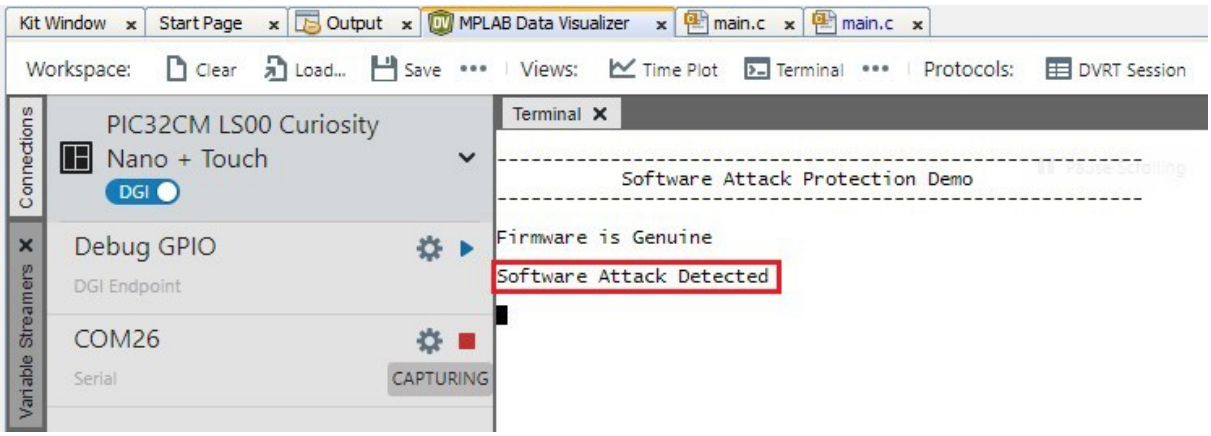
9. PIC32CM LS00 Curiosity Nano+ Touch 評価用キット上の SW1 ボタンを押す事により、ソフトウェア攻撃をシミュレートします。

図 6-9. ソフトウェア攻撃のシミュレーション



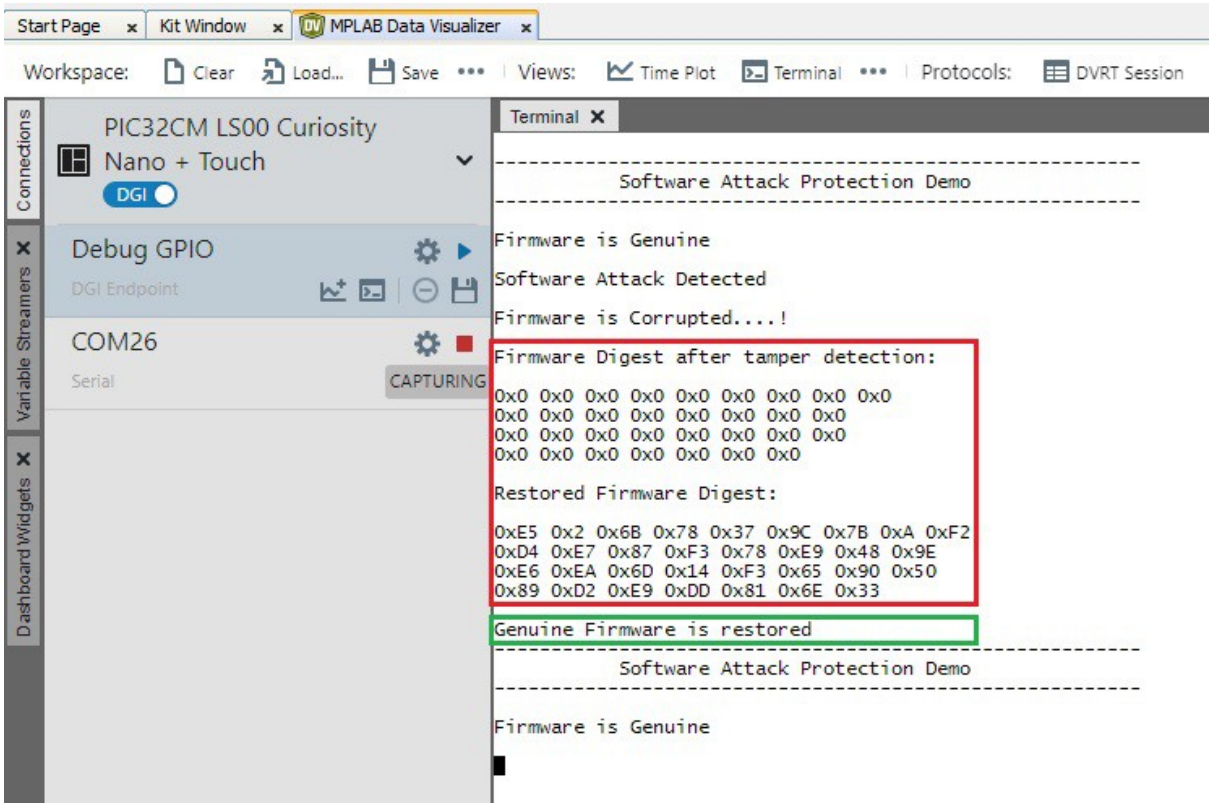
10. MPLAB Data Visualizer 上でソフトウェア攻撃開始のメッセージを確認します。

図 6-10. ソフトウェア攻撃開始



11. RTC がタイムアウトすると、セキュア アプリケーションは非セキュア ファームウェアの検証を開始します。検証に失敗すると、非セキュア ファームウェアの真正コピーが非セキュア フラッシュ領域にプログラミングされます。

図 6-11. 非セキュア ファームウェアの検証



Note: 真正コピーが正常にプログラミングされた後に、セキュア アプリケーションはソフトウェア リセットを開始します。

7. 参考資料

- 以下の文書は、Microchip 社ウェブサイト(www.microchip.com)からダウンロードできます。
  - PIC32CM LS00 Curiosity Nano+ Touch Evaluation Kit User Guide (DS70005567)
  - PIC32CM LS00/LS60 Security Reference Guide (DS00003992)
- PIC32CM LS00 Curiosity Nano+ Touch 評価用キットの製品ページ
- Secure Boot on PIC32CM LS60 Curiosity Pro Evaluation Kit Using MPLAB® Harmony v3 Software Framework
- MPLAB® Harmony v3 に関する情報は以下を参照してください。
  - Microchip 社ウェブサイト: developerhelp.microchip.com/xwiki/bin/view/software-tools/harmony/
- 各種アプリケーションに関する情報は以下を参照してください。
  - https://github.com/Microchip-MPLAB- Harmony/reference_apps
- 32 ビット マイクロコントローラ関連情報およびソリューションは以下を参照してください。
  - 32-bit Microcontroller Collateral and Solutions Reference Guide (DS70005534)

8. 改訂履歴

リビジョン A - 2025 年 4 月

本書は初版です。

Microchip 社の情報

商標

「Microchip」社の名称とロゴ、「M」のロゴ、およびその他の名称、ロゴ、ブランドは、米国およびその他の国における Microchip Technology Incorporated またはその関連会社および/または子会社の登録商標および未登録商標です(「Microchip 社の商標」)。「Microchip 社の商標」に関する情報は https://www.microchip.com/en-us/about/legal- information/microchip-trademarks に記載されています。

ISBN: 979-8-3371-2216-8

法律上の注意点

本書および本書に記載されている情報は、Microchip 社製品を設計、テスト、お客様のアプリケーションと統合する目的を含め、Microchip 社製品に対してのみ使用する事ができます。
それ以外の方法でこの情報を使用する事はこれらの条項に違反します。デバイス アプリケーションの情報は、ユーザの便宜のためにのみ提供されるものであり、更新によって変更となる事があります。お客様のアプリケーションが仕様を満たす事を保証する責任は、お客様にあります。その他のサポートについては、弊社または代理店にお問い合わせになるか、www.microchip.com/en-us/support/design-help/client-support-services をご覧ください。

Microchip 社は本書の情報を「現状のまま」で提供しています。
Microchip 社は明示的、暗黙的、書面、口頭、法定のいずれであるかを問わず、本書に記載されている情報に関して、非侵害性、商品性、特定目的への適合性の暗黙的保証、または状態、品質、性能に関する保証をはじめとするいかなる類の表明も保証も行いません。

いかなる場合も Microchip 社は、本情報またはその使用に関連する間接的、特殊的、懲罰的、偶発、的または必然的損失、損害、費用、経費のいかんにかかわらず、また Microchip 社がそのような損害が生じる可能性について報告を受けていた場合あるいは損害が予測可能であった場合でも、一切の責任を負いません。法律で認められる最大限の範囲を適用しようとも、本情報またはその使用に関連する一切の申し立てに対する Microchip 社の責任限度額は、使用者が当該情報に関連して Microchip 社に直接支払った額を超えません。法律で認められる最大限の範囲を適用しようとも、本情報またはその使用に関連する一切の申し立てに対する Microchip 社の責任限度額は、使用者が当該情報に関連して Microchip 社に直接支払った額を超えません。

Microchip 社の明示的な書面による承認なしに、生命維持装置あるいは生命安全用途に Microchip 社の製品を使用する事は全て購入者のリスクとし、また購入者はこれによって発生したあらゆる損害、クレーム、訴訟、費用に関して、Microchip 社は擁護され、免責され、損害をうけない事に同意するものとします。特に明記しない場合、暗黙的あるいは明示的を問わず、Microchip 社が知的財産権を保有しているライセンスは一切譲渡されません。

Microchip 社のデバイスコード保護機能

Microchip 社製品のコード保護機能について以下の点にご注意ください。

- Microchip 社製品は、該当する Microchip 社データシートに記載の仕様を満たしています。
- Microchip 社では、通常の条件ならびに仕様に従って使った場合、Microchip 社製品のセキュリティレベルは、現在市場に流通している同種製品の中でも最も高度であると考えています。
- Microchip 社はその知的財産権を重視し、積極的に保護しています。Microchip 社製品のコード保護機能の侵害は固く禁じられており、デジタル ミレニアム著作権法に違反します。
- Microchip 社を含む全ての半導体メーカーで、自社のコードのセキュリティを完全に保証できる企業はありません。コード保護機能とは、Microchip 社が製品を「解読不能」として保証するものではありません。コード保護機能は常に進歩しています。Microchip 社では、常に製品のコード保護機能の改善に取り組んでいます。

製品ページへのリンク

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