TRAVEO™ T2G ファミリのセキュアシステムの構築

本書について

適用範囲と目的

このドキュメントでは、TRAVEO™ T2G ファミリでのセキュアシステム構築に必要なブートプロセスからアプリケーションソフトウェア実行までについて説明します。

対象者

このドキュメントは、TRAVEO™ T2G ファミリを使用してセキュアシステムを構成するすべての人を対象とします。

関連製品ファミリ

TRAVEO™ T2G ファミリ

目次

目次

目次

1 はじめに

このアプリケーションノートは、システムが信頼できるソースからのみコードを実行する方法、および TRAVEO™ T2G ファミリ MCU を使用してセキュアな組込みシステムを構築する方法について説明します。起動プロセスとセキュアシステムの関係についても学習します。

これは高度なアプリケーションのため、基本的な TRAVEO™ T2G アーキテクチャに精通していることを前提とします。(デバイスデータシート[1]またはアーキテクチャテクニカルリファレンスマニュアル(TRM)[2]を参照してください。)

メーカーは自社の IP を保護し市場を維持する、または第三者からの悪意ある攻撃によって引き起こされる危険な操作からエンドユーザを保護するなど、システムが安全でなければならない多くの理由があります。

製品がハッキングされる主な 3 つの方法を示します。

- **デバッグポートへの直接アクセス**: TRAVEO™ T2G は、Arm® アーキテクチャに基づきます。共通のデバッガを使用しファームウェアへのアクセスや再プログラミング、または内部データの調査を簡単に行えます。デバイスが保護されていない場合、製品のハッキングやリバースエンジニアリングは簡単です。
- **SPI, I2C, CAN, LIN, または UART など通信ポートへの直接アクセス**: これらのポートは、ECU 間、または ECU と車内の他コンポーネント間の通信に使用されます。また、ECU のソフトウェア更新や情報取得にも使用されます。デバイスが保護されていない場合、許可されていないアクセスによってファームウェアの読出し、更新ができます。
- **OTA などのワイヤレス接続**: 物理的な接触は必要としないため、ハッキングの標準的な方法です。デバイスが保護されていない場合、ハッカーは、どこからでも車にアクセスし不正なファームウェアの更新や制御ができます。

2 セキュアシステムとは？

セキュアシステムの定義は、アプリケーションによって異なります。一部のシステムでは、デバイスすべてのアクセスをブロックする必要がありますが、ファームウェア破損のみを確認するシステムもあります。TRAVEO™ T2G MCU では、プロジェクトに必要なセキュリティレベルを定義できます。プロジェクトごとに要件が異なるため、完全な方法はありません。以下に、さまざまなセキュリティ目的を持つプロジェクトを示します。

- **ハードウェアデバッガのみを使用した信頼できるファームウェアの更新**: 通常、セキュアシステムとはみなされませんが、第三者が直接アクセスできないようハードウェアが実装されている時、セキュアである場合があります。ファームウェアからのフラッシュ書込みコマンドを無効化して、内部ハッキングによってアプリケーションの変更および置換えを不可能にします。デバイスは、デバイスがリセットからの復帰ごとにファームウェアが公開鍵で強制的に認証されるセキュアモードでデバッグポートをオープン状態にできます。
- **デバッグポートへのアクセスなし、ファームウェア更新のサポート**: デバッグポートは、すべてのメモリへのアクセスとデバイスの再プログラムを提供します。ほとんどの場合、実際のセキュアシステムではデバッグポートを無効化する必要があります。ファームウェア更新の唯一の方法は、アプリケーションが UART, I2C, CAN, LIN または、SPI などの通信ポートを使用し新しいプログラムデータをダウンロードします。この通信ポートの安全性は、設計者に依存します。
- **ファームウェアをロックし、更新しない**: デバッグポートは無効化され、ブートローダも用意されません。最も安全ですが、不具合の修正や将来的な機能拡張ができません。
- **信頼できるファームウェア更新と IP の保護**: IP を完全に保護するためには、デバッグポートを無効化する必要があります。このため、ユーザは新しいファームウェア更新のためブートローダを用意する必要があります。これは通常、UART, SPI, または I2C などのシリアルポートによって実装されます。IP 保護のため、ブートローダは転送データを暗号化する必要があります。TRAVEO™ T2G MCU は、暗号化と復号化を高速に行うための暗号ブロックを持ちます。工場でデバイスに実装されたセキュリティキーを使用して、コードを認証し、ブートプロセス中に転送データを復号化できます。

***注**: TRAVEO™ T2G ファミリ MCU が提供するすべてのセキュリティ関連機能は、Logical attack に対する保護のみを目的としています。これらは、通常のインターフェイスを使用して、デバイスと予期せぬ方法で通信する攻撃であり、リモートでも実行できます。典型的な例としては、バッファオーバーフロー、認証バイパス、悪意のあるソフトウェアのインジェクション、デバッグ機能やインターフェースの悪用などがあります。*

***注**: Physical attack は範囲外とみなされるため、TRAVEO™ T2G ファミリ MCU には、Non-invasive attack (非侵襲攻撃)、Semi-invasive attack (半侵襲攻撃)、または Invasive attack (侵襲攻撃) に対するハードウェア対策は含まれていません。サイドチャネル攻撃やフォールト インジェクション攻撃への対策は、ソフトウェアで実装する必要があります。*

3 基本的な定義

ここでは、ドキュメントで使用するいくつかの用語について説明します。

- **アプリケーションフラッシュ(ユーザ)**: アプリケーションコードを格納のため使用するフラッシュメモリです。
- **信頼の連鎖 (Chain of Trust:CoT)**: 信頼の基点 (The root of trust) は、ROM に格納されるコードから開始され、これを変更することはできません。CoT は、ROM に格納される信頼の基点からのソフトウェアブロックを実行開始前に検証することによって確立されます。
- **Cipher-based Message Authentication Code (CMAC)**: ブロック暗号に基づいたメッセージ認証コードアルゴリズムです。(例:AES)
- **CySAF**: サイプレス セキュア アプリケーション フォーマット
- **デバッグアクセスポート (DAP)**: 書込みおよびデバッグ用の外部デバッガ/プログラマと TRAVEO™ T2G MCU 間のインタフェース。CM0_AP, CM4_AP (または CM7_AP), および System_AP の 3 つのアクセスポート(AP)のいずれかに接続できます。System_AP は、SRAM, フラッシュ, および MMIO にのみアクセスでき、CPU にはアクセスできません。
- **DAP アクセス制限**: デバッグアクセスポートのアクセス制限を決定します。保護状態に応じて Normal, Secure, および DEAD の 3 つの状態があります。各状態はユーザが設定可能です。各保護状態でのアクセス制限の格納場所については、セキュリティエンハンス PPU コンフィグレーションを参照してください。
- **Digital Digest/Signature**: データブロックを SHA-256 によって処理し生成した署名。
- **電子制御ユニット (ECU)**: 電子回路を使用してシステム制御するためのユニット。主に自動車に搭載され、ボディ制御, エンジン制御, およびブレーキ制御など用途に応じてさまざまな ECU があります。
- **eFuse**: ワンタイムプログラマブル (OTP) メモリ。デフォルトでは“0”であり、“0”から“1”にのみ変更可能です。eFuse ビットは個別にプログラムできますが消去はできません。
- **eFuse 読出し保護ユニット (ERPU)**: SWPU(ソフトウェア保護ユニット、以下の定義を参照)の一部です。ERPU は、eFuse からの読出しアクセスを制限するために使用します。
- **eFuse 書込み保護ユニット (EWPU)**: SWPU の一部です。EWPU は、eFuse への書込みアクセスを制限するために使用します。
- **フラッシュブート**: 2 つの基本的なタスクを実行するブートシステムの一部です。
  - ライフサイクルステージに応じてデバッグポートを設定します。
  - 実行前にユーザアプリケーションを検証します。
- **フラッシュ書込み保護ユニット (FWPU)**: SWPU の一部です。FWPU は、フラッシュメモリへの書込みアクセスを制限するために使用します。
- **ハッシュ (Hash)**: 所定のデータブロックに対し反復可能な一意のダイジェストを生成する暗号アルゴリズムです。これは不可逆的です。
- **IP (Intellectual Property)**: 知的財産。デバイスに格納されるコードとデータの両方を指します。
- **IPC (Inter-Processor Communication)**: プロセッサ間通信。2 つの CPU コア間の通信を容易にするプロセッサ間通信ハードウェア。
- **ライフサイクル**: デバイスが動作しているセキュリティモードです。TRAVEO™ T2G MCU は、NORMAL_PROVISIONED, SECURE, SECURE_W_DEBUG, RMA, および CORRUPTED の 5 つのステージがあります。ユーザにとって対象となるのは、NORMAL_PROVISIONED, SECURE, および SECURE_W_DEBUG の 3 つです。
- **メインユーザアプリケーション**: フラッシュブートで認証されないユーザアプリケーションの一部です。主に CM4 または CM7 によって実行されます。CoT では、セキュアイメージによって認証される必要があります。
- **メモリ保護ユニット (Memory Protection Unit:MPU)**: MPU は同一の CPU で実行される異なるソフトウェアコンポーネントのメモリセクションを分離するために使用します。MPU は、バスマスタ固有です。
- **MMIO (Memory-Mapped Input/Output)**: メモリ上にマッピングされ、通常はハードウェア入出力を制御するレジスタを指します。

3 基本的な定義

- **非セキュア (Non-Secure:NS)**: セキュアアクセスと非セキュアアクセスを区別するために使用する保護属性です。非セキュア設定では、セキュアアクセスも許可されます。NS 属性は、SMPU, PPU, および SWPU によって許可または制限されます。
- **Over the Air (OTA)**: ワイヤレス通信を介したデータの送受信を指します。
- **保護コンテキスト (Protection Context:PC)**: 保護ユニットの設定変更なしにさまざまな保護属性をバスマスタアクセスに適用できます。PC 属性は、SMPU, PPU, および SWPU によって許可または制限されます。PC は多くの場合、プログラムカウンタを指しますが、このドキュメントでは保護コンテキストを指します。
- **周辺保護ユニット (Peripheral Protection Unit:PPU)**: 周辺機能または周辺機能セットへのアクセスを 1 つまたは特定のバスマスタセットのみに制限するために使用します。
- **保護状態 (Protection State)**: ライフサイクルステージに応じて、Normal, Secure, DEAD, および Virgin 状態があります。ROM ブートは、保護状態に応じてアクセス制限を構成します。
- **保護ユニット (Protection Unit)**: メモリ保護ユニット(MPU)、共有メモリ保護ユニット(SMPU)、周辺保護ユニット(PPU)、およびソフトウェア保護ユニット(SWPU)の 4 種類の保護ユニットがあります。MPU, SMPU, PPU はハードウェアによって実装され、SWPU はソフトウェアによって実装されます。
- **公開鍵暗号 (Public-Key Cryptography:PKC)**: 非対称暗号とも呼ばれ、安全なデータ通信のために公開鍵と秘密鍵のペア (または非対称鍵) アルゴリズムを使用する暗号化技術です。メッセージまたはデータブロックをデコードするために使用します。秘密鍵はデータの復号化に使用され、安全に保管する必要があります。一方、公開鍵はデータの暗号化に使用され、広く配布できます。
  - **公開鍵 (Public Key)**: 公開鍵は共有できますが、変更できないように認証または保護する必要があります。
  - **秘密鍵 (Private Key)**: 秘密鍵は安全な場所に保管し、公開または窃取されないようにする必要があります。関連する公開鍵を使用して、暗号化されたデータのブロックを復号化するために使用します。
- **RMA**: 返却品認証 (Returned Material Authorization)
- **ROM (Read Only Memory)**: 製造プロセスの一部としてプログラムされ、再プログラム不可の読出し専用メモリ。
- **RSA-nnnn**: 2 つの鍵を使用する非対称暗号化システム。1 つは秘密であり、共有すべきではありません。もう 1 つは公開可能で、セキュリティを失うことなく読み出せます。通常、暗号化/復号化は 2048, 3072, または 4096 ビット長のキー (RSA-2048, RSA-3072, または RSA-4096) によって制御されます。
- **セキュアイメージ**: TRAVEO™ T2G MCU の HSM ファームウェアなどのセキュリティ機能をセットアップするために使用されるソフトウェアです。主に CM0+により実行されます。開発者は特定のセキュリティポリシーを実装するために変更できます。
- **セキュリティポリシー**: 外部の改ざんまたは CPU 間での保護されるリソース決定のために設計者によって課せられるルール。
- **シリアルメモリインタフェース (SMIF)**: NOR フラッシュ, SRAM, および不揮発性 SRAM などの SPI (シリアルペリフェラルインタフェース) 通信インタフェースです。
- **SFlash**: スーパバイザフラッシュメモリ。フラッシュ内のこのメモリパーティションには、システムトリム値, フラッシュブート, 公開鍵などの領域が含まれます。デバイスが SECURE ライフサイクルステージに移行すると変更できません。
- **SHA-256**: データまたはコードのブロックダイジェストを作成するために使用する暗号化ハッシュアルゴリズムです。データブロックのサイズに関係なく、一意の 256 ビットダイジェストを生成します。
- **共有メモリ保護ユニット (Shared Memory Protection Unit:SMPU)**: 特定のメモリ空間 (フラッシュ, SRAM, またはレジスタ) へのアクセスを 1 つまたは特定のバスマスタセットにのみ許可するために使用します。
- **ソフトウェア保護ユニット(SWPU)**: フラッシュ書込み、および eFuse 読出しと書込みアクセスの制限を実装するために使用します。
- **システムコール**: Arm® Cortex® M0+ CPU (CM0+) が ROM から実行するフラッシュ書込み機能などの関数群。

3 基本的な定義

- **TOC1**: トリム値、フラッシュブートエントリポイントなどへのポインタを格納するために使用する SFlash 領域です。ROM ブートコードのみが使用し、設計者は編集できません。
- **TOC2**: 2 つのアプリケーションブロック (セキュアイメージとメインユーザアプリケーション) へのポインタを格納するために使用する SFlash 領域です。また、システム設計者が構成できるブートパラメータを含みます。

4 ライフサイクルステージ

デバイスのライフサイクルは、TRAVEO™ T2G MCU セキュリティにとって重要です。ライフサイクルステージは、eFuse の切断 (ヒューズ値を「0」から「1」に変更) によって決定される厳格で不可逆的な遷移です。このシステムは、顧客が必要とするレベルで内部デバイスのコードとデータを保護するために使用します。図 1 に、TRAVEO™ T2G MCU がサポートするライフサイクルステージを示します。

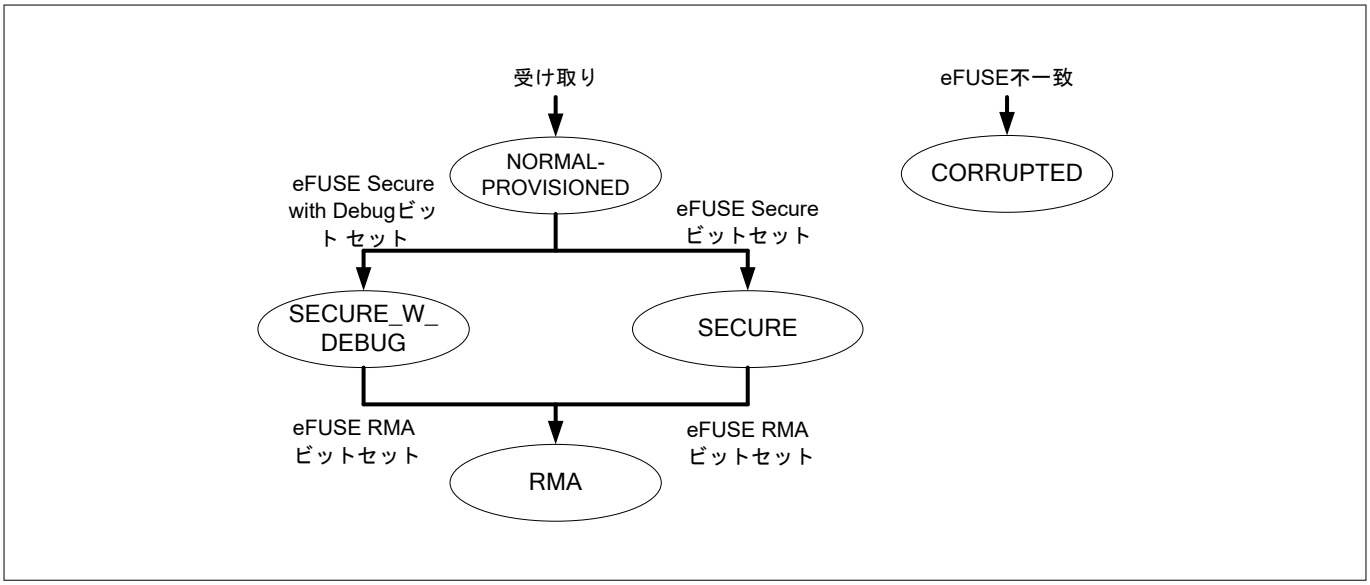


図 1 デバイスライフサイクルステージ

4.1 NORMAL_PROVISIONED

工場でトリミングとテストが完了したデバイスのライフサイクルステージです。すべての設定およびトリミング情報が完了し、有効なフラッシュブートコードが SFlash にプログラムされます。トリム, フラッシュブート, およびその他のオブジェクトデータの整合性を OEM によりチェックするため、これらオブジェクトのハッシュ (128 ビットに切り捨てた SHA-256) が eFuse に格納されます。このハッシュは FACTRY_HASH と呼ばれます。お客様はこの段階で部品を受け取ります。

4.2 SECURE

セキュアなデバイスライフサイクルステージです。このステージに移行する前に、SECURE_HASH が eFuse に組み込まれ有効なアプリケーションコードがコードフラッシュにプログラムされている必要があります。保護状態は SECURE に設定され、SECURE アクセス制限が適用されます。SECURE デバイスは、フラッシュブートとアプリケーションコードの認証が成功した場合のみ起動します。SECURE_HASH は、NORMAL_PROVISIONED ライフサイクルステージから SECURE または SECURE_W_DEBUG ライフサイクルステージに移行する時、SROM ファームウェアによって計算され、eFuse に書き込まれます。SECURE ライフサイクルステージへの移行は、Appendix H - プログラムとスクリプト例を参照してください。

***注**: ライフサイクルステージは元に戻せません。したがって、SECURE ライフサイクルステージに移行する前に、プログラミングプロセスを評価し、アプリケーションプログラムを認証することを推奨します。*

4.3 SECURE_W_DEBUG

デバイスがデバッグ可能なことを除き、SECURE ライフサイクルと同じです。このステージに移行する前に、SECURE_HASH が eFuse に組み込まれ有効なアプリケーションコードがコードフラッシュにプログラムされている必要があります。保護状態は SECURE に設定されますが、デバッグを有効にするためにノーマルアクセス制限が適用されます。ROM ブートまたは、フラッシュブート中に認証エラーが発生した場合、保護状態は「SECURE」に

設定されますが、NORMAL アクセス制限が有効となり、SWD/JTAG ピンが有効になり「DEAD」状態には移行しません。SECURE_W_DEBUG から SECURE への移行はできません。SECURE_W_DEBUG に設定された部品は、開発者とソフトウェア評価者のみが使用し、このライフサイクルステージでは、デバイスを出荷しないでください。SECURE_W_DEBUG ライフサイクルステージへの移行は、Appendix H - プログラムとスクリプト例を参照してください。

4.4 RMA

障害分析 (FA) を実行できます。故障解析をする場合、顧客は部品を RMA ライフサイクルステージに移行させます。顧客は、部品を RMA に移行するシステムコール (TransitiontoRMA API) を実行する前にすべての機密データを消去します。

システムコールによって RMA に移行する場合、顧客は、RMA ライフサイクルステージ移行のため、部品固有のユニーク ID とともに証明書を作成する必要があります。証明書は、ユーザアプリケーションイメージの署名に使用される秘密鍵を使用して、顧客によって署名されます。署名の検証は、フラッシュブートがユーザアプリケーション認証に使用するアルゴリズムと同じものが使用され、SFlash に格納される同じ公開鍵 (OEM によって書き込まれたもの) を使用します。

RMA ライフサイクルステージで部品がリセットされると、ブートは、DAP が System_AP、システムコールのため IPC MMIO レジスタ、および通信用の適切な RAM のみにアクセスするようアクセス制限を設定します。次に、DAP から認証用の証明書とシステムコール (OpenRMA API) を待ちます。OpenRMA API が正常に実行されるまで、ブートプロセスはファームウェアを開始しません。このステージに移行後、eFuse に格納されたライフサイクルステージを RMA から変更できません。部品をリセットするごとに、使用する前に OpenRMA API を実行しなければいけません。OpenRMA API を実行するために、顧客は秘密鍵を使用して署名した証明書を作成する必要があります。RMA ライフサイクルステージへの移行は、Appendix E - RMA ライフサイクルステージへの移行を参照してください。

***注**: DEAD 保護状態または CORRUPTED ライフサイクルステージのデバイスは、RMA ライフサイクルステージに移行できません。*

***注**: RMA への移行は、SECURE または SECURE_W_DEBUG ライフサイクルステージからのみ可能です。したがってデバイスが、NORMAL_PROVISIONED ライフサイクルステージにある場合、SECURE または SECURE_W_DEBUG に移行後、RMA に移行する必要があります。*

***注**: TransitiontoRMA API 実行する場合、RSA キーペア(秘密鍵と公開鍵)が必要です。秘密鍵は証明書の作成に使用され、公開鍵は証明書の認証に使用されます。公開鍵は事前にデバイスに書き込んでおく必要があります。*

***注**: OpenRMA API を実行するには、RSA キー ペア(秘密鍵と公開鍵)が必要になる場合があります。秘密鍵は証明書の作成に使用され、公開鍵は証明書の認証に使用されます。公開鍵は事前にデバイスに書き込んでおく必要があります。ただし、デバイスが SECURE_WITH_DEBUG ライフサイクル ステージから RMA ライフサイクル ステージに移行した場合、OpenRMA はスキップされ、デバイスは OpenRMA の実行を待機しません。したがって、デバイスへのフル アクセスは証明書なしでロック解除され、ユーザアプリケーションが実行されます。*

***注**: TransitiontoRMA および OpenRMA API の実行中に ECC エラーが発生する可能性があります。したがって、ユーザソフトウェアは、TransitiontoRMA の実行中および OpenRMA API の実行後の障害処理に注意してください。詳細については、RMA ライフサイクルステージ移行の要件を参照してください。*

ハードウェア障害や機密データの消去などの意図しない認証失敗によりデバイスが DEAD 保護状態に移行し、RMA ライフサイクルステージに移行できない場合、RMA ライフサイクルステージへの移行のみを管理するセカン

ドアプリケーションソフトウェアを準備することで解決できる場合があります。開始アドレスは TOC2 で定義できます。ファーストアプリケーションソフトウェアが失敗する、またはファーストアプリケーションをすべて消去した場合、フラッシュブートはセカンドアプリケーションを実行します。ただし、セカンドアプリケーションソフトウェアの認証も破損している場合、デバイスは RMA ライフサイクルステージに移行できません。セカンドアプリケーションは RMA 管理専用として使用できます。ファーストアプリケーションのみ消去できるようセカンドアプリケーションはファーストアプリケーションと異なるセクタに配置することを推奨します。TOC2 の詳細については TOC2 を、セカンドアプリケーションの実装については Appendix I - セカンドアプリケーションソフトウェア実装を参照してください。

しかしながら、デバイスがブートローダの有効化条件を満たしている場合、ファーストアプリケーションがすべて消去されると、セカンドアプリケーションを起動できません。ブートローダの有効化条件については、TOC2 を参照してください。

ファーストアプリケーションがすべて消去された場合、DEAD 状態に移行する場合があります。このため、2 つのオプションがあります。

- セカンドアプリケーションが実装されていない場合
  - 消去後、ファーストアプリケーションソフトウェアの代わりに、署名されたダミーコードおよびデジタル署名を準備し再プログラムします。このダミーコードは、RMA 管理用として使用します。
- セカンドアプリケーションが実装されている場合
  - 消去後、ファーストアプリケーションソフトウェアの代わりに、ダミーのアプリケーションヘッダを準備し、再プログラムします。アプリケーションヘッダの実装については、Appendix J – ダミーアプリケーションヘッダの実装を参照してください。
  - ブートローダを無効にします。(TOC2_FLAGS.FB_BOOTLOADER_CTL=0x2) これは、NORMAL_PROVISIONED ライフサイクルでのみ可能です。

4.4.1 RMA ライフサイクルステージ移行の要件

RMA ライフサイクルステージの移行に使用する TrransitiontoRMA および OpenRMA API の実行条件について説明します。

4.4.1.1 TransitiontoRMA API

TransitiontoRMA API の実行中に ECC エラーが発生する場合があります。したがって、ユーザソフトウェアは、TransitiontoRMA API をトリガする前に Crypto および SRAM0 ECC エラーのための Fault report を設定しないでください。そうでない場合、ソフトウェアは TransitiontoRMA API 実行中に通知された ECC エラーを無視してください。以下の異常をマスクする必要があります。

- (フォールト番号は CYT2B のものです)
  - CPUSS_RAM0_C_ECC (フォールト番号=58)
  - CPUSS_RAM0_NC_ECC (フォールト番号=59)
  - CPUSS_CRYPTO_C_ECC (フォールト番号=64)
  - CPUSS_CRYPTO_NC_ECC (フォールト番号=65)

フォールト番号については、デバイス固有のデータシート[1]を、障害のマスク例は、API の実行を参照してください。

さらに、RMA ライフサイクルステージのデバイスは、リセットごとに OpenRMA API が必要です。そのためデバイス固有のハードウェア障害などによって Fault Report がリセットをトリガする場合、OpenRMA 実行後、デバイスは RMA に移行できません。この場合、TransitiontoRMA API を実行する前に 2 つのオプションがあります。

- 事前にリセットを発行するデバイス固有の障害をマスクします。同様に OpenRMA 後に実行されるアプリケーションソフトウェアもマスクします。
- RMA 管理のみを実行するダミーコードを再プログラムします。

TransitiontoRMA API の実行には、証明書とデジタル署名を SRAM に書込む必要があります。TransitiontoRMA API を使用してデバイスを RMA ライフサイクルステージに移行する場合、証明書やデジタル署名などのパラメー

タは、[システム RAM0 開始アドレス+4KB]以降に配置する必要があります。証明書とデジタル署名については、Appendix E - RMA ライフサイクルステージへの移行を参照してください。

4.4.1.2 OpenRMA API

OpenRMA API 実行中に ECC エラーが発生する場合もあります。したがって、OpenRMA API 後に実行されるソフトウェアは、Crypto および SRAM0 の ECC エラーを Fault structure に設定してはいけません。そうでない場合、ソフトウェアは通知された ECC エラーを無視してください。マスクする必要のある異常については、TransitiontoRMA API を、障害のマスク例については、API の実行を参照してください。

RMA ライフサイクルステージの保護状態は、VIRGIN です。したがって、ソフトウェアは、CPUSS_PROTECTION レジスタを使用してデバイスが RMA ライフサイクルステージかどうかを知ることができます。詳細についてはレジスタ TRM[2]を参照してください。

OpenRMA API の実行には、証明書とデジタル署名を SRAM に書込む必要があります。デバイスが RMA ライフサイクルステージに移行後、Sys_DAP は Sys_DAP MPU によって IPC MMIO とシステム RAM0 の 1/16 にのみアクセスできます。OpenRMA を使用する場合、証明書やデジタル署名などのパラメータは以下のように配置する必要があります。

- SRAM0 サイズが 64KB より大きいデバイス：パラメータは、[システム RAM0 開始アドレス+ 4KB]から[システム RAM0 開始アドレス+1/16 システム RAM0 サイズ]まで配置する必要があります。
- SRMA0 が 64KB 以下のデバイス：パラメータは、[システム RAM0 開始アドレス+ 2KB]から 600 バイト以内に配置する必要があります。証明書と署名アドレスは 24 bytes で、デジタル署名は 512bytes です。(RSA-4K の場合)

証明書とデジタル署名については、Appendix E - RMA ライフサイクルステージへの移行を参照してください。

障害番号とシステム RAM0 サイズについてはデバイス固有のデータシート[1]を参照してください。

4.5 CORRUPTED

ライフサイクルを決定する eFuse 読出し時にエラーを検出すると、デバイスはこのライフサイクルステージに移行します。デバイスは保護状態 DEAD に移行し、SYS-AP を介して IPC MMIO のみを読み出せます。他のアクセスは許可されません。

5 ブートシーケンスと信頼の連鎖

信頼の連鎖(CoT)は本質的にブートシーケンスの一部です。これは ROM に保存される初期ブートコードであり、変更できない信頼されたルートから始まります。

5.1 ブートシーケンス

多くの場合、起動シーケンスと検証シーケンスは同じです。図 2 に、リセット後の CM0+動作を示します。リセット後、CM0+は ROM ブートから実行を開始し、SFlash を検証します。SFlash の検証が完了すると、フラッシュブートにジャンプし、保護状態に応じて DAP を構成します。データとコードのあるメモリの種類を色分けで示します。

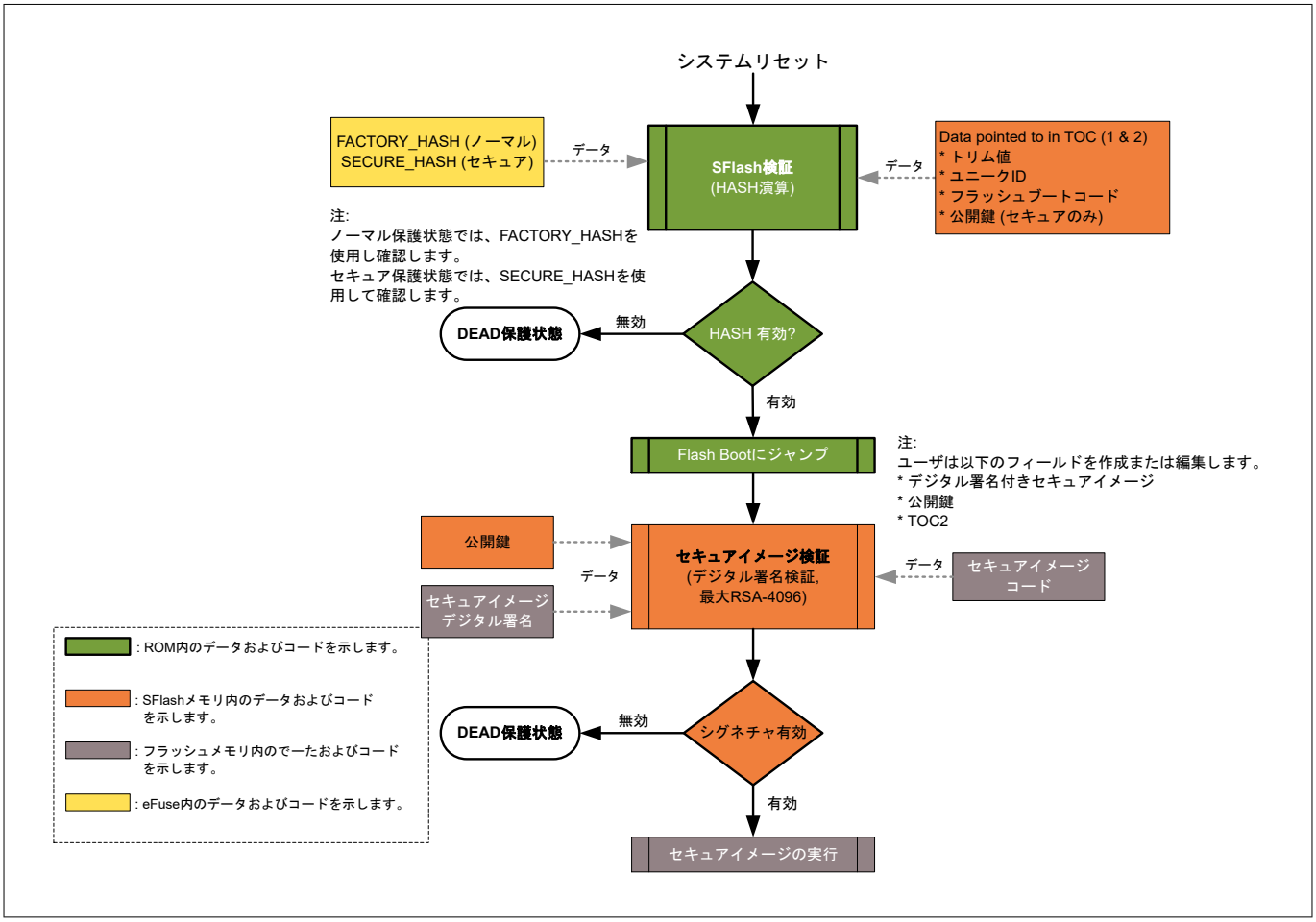


図 2 TRAVEO™ T2G MCU の CoT ブートシーケンス

次にフラッシュブートは、TOC2 に記述されるファーストアプリケーションを検証し、検証後、そのエントリポイントにジャンプします。このアプリケーションノートで定義されるセキュアシステムでは、ファーストユーザアプリケーションはセキュアイメージを指します。セキュアイメージは、システムを保護するハードウェアを構成し、必要に応じてメインユーザアプリケーションを検証します。

SFlash またはセキュアイメージの無効または破損を検出した場合、デバイスは DEAD 保護状態に遷移し、デバイスがリセットされるまで DEAD 保護状態のままです。

注: デバイスが DEAD 保護状態に遷移すると、RMA ライフサイクルステージに移行できないため、障害分析を実行できません。

5.2 信頼の連鎖 (Chain of Trust:CoT)

CoT の基礎は、ROM など変更できないメモリに依存します。残りの部分は、この事実に依存します。ROM コードは、変更できず次の実行ブロック (この場合は、フラッシュブート) の検証に使用します。

フラッシュブートコード, トリム値, および TOC1 は再プログラムできない SFlash にあります。その多くは、工場で事前にプログラムされます。このセクションのハッシュ値は、FACTORY_HASH と呼ばれ eFuse に格納されます。これにより、MCU の提供後、フラッシュブートコード, トリム値, および TOC1 が改ざんされていないことが保証されます。

ライフサイクルステージが、NORMAL から SECURE に移行後、SFlash ブロックは、「SECURE_HASH」と呼ばれる別のハッシュ値によって検証されます。この値も eFuse に格納されプログラム後は変更できません。SFlash のこれらのブロックは、TOC2 と公開鍵が含まれます。

NORMAL ライフサイクルステージで SFlash 検証に使用する FACTORY_HASH 値は eFuse に格納され、変更できません。別の場所を使用しプログラムされた追加のアイテムと SECURE_HASH を使用して SFlash を検証します。これは eFuse の異なるセクションに書込まれます。

SECURE ライフサイクルステージでは、デバイスがリセットから復帰するごとに、SFlash ブロック全体が SECURE_HASH で検証されます。SFlash 検証中にエラーが検出された場合、デバイスはブートシーケンスを完了せず、DEAD 状態になります。

検証が成功した場合、SFlash 全体が信頼できます。これは、検証がメモリ (eFuse) に基づくため、ROM での SFlash 検証中の異常検出なしに変更できないためです。

SFlash 内に格納される公開鍵はセキュアであり、そして同様に検出なしに変更できません。ブートプロセスの次のステップを検証するために公開鍵はフラッシュブートで使用します。フラッシュブートは、コードブロックの最後に配置されるデジタル署名を含むセキュアイメージブロックコードを検証します。フラッシュブートは、SHA-256 ハッシュ関数を使用して、セキュアイメージブロックのデジタル署名を計算します。セキュアイメージブロックに添付されたデジタル署名は、最大 RSA-4096 ビット暗号化を使用して、SFlash 内の公開鍵と関連する秘密鍵を使用して暗号化されます。次に、計算および格納され暗号化されたデジタル署名がチェックされ、一致するかどうかを確認します。一致した場合、セキュアイメージは検証成功です。セキュアイメージ内でも同様のプロセスを使用して、ユーザアプリケーションブロックを検証できます。図 4 と図 5 にセキュアイメージの署名と検証のフローを示します。

図 3 に、データおよびコード検証の CoT を示します。

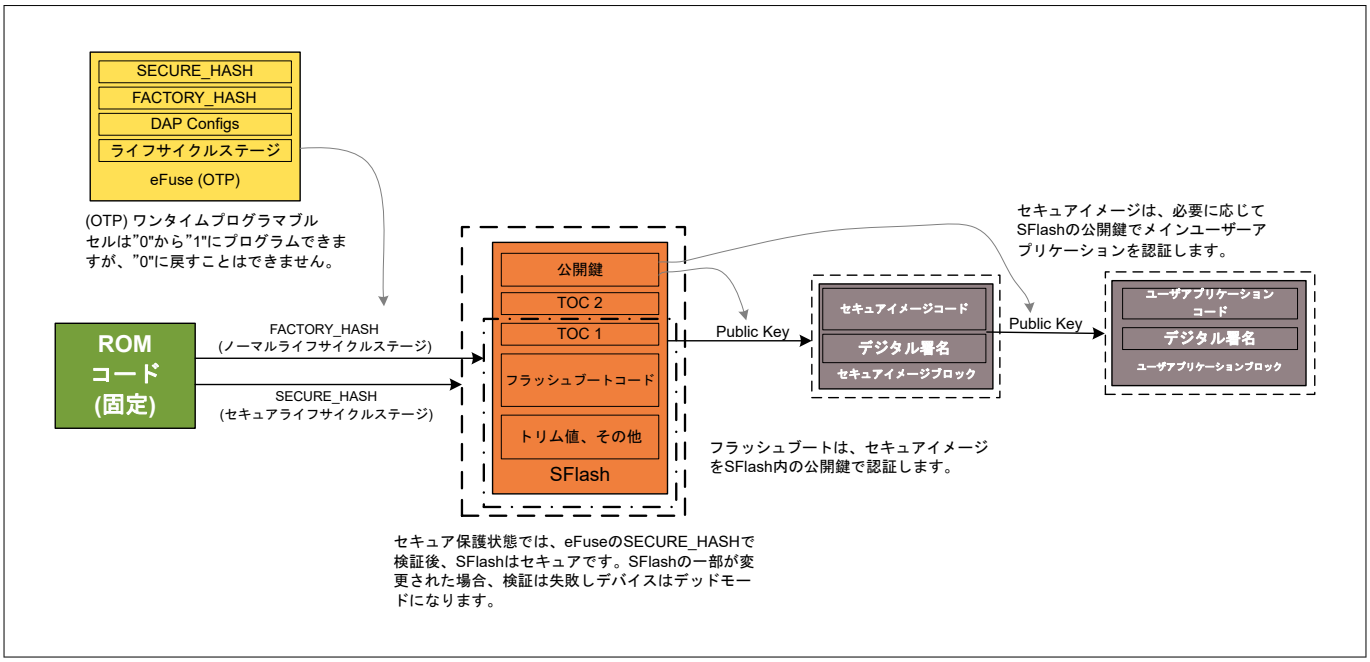


図 3 基本的な CoT

6 コードの署名と検証

ここでは、起動時にコードブロックを検証するためコードブロックに署名するプロセスについて説明します。

秘密鍵と公開鍵を使用する公開鍵暗号 (PKC) を暗号化方式として使用します。秘密鍵を安全な場所に保管し、公な領域に侵入しないように注意する必要があります。秘密鍵が公開された場合、システムのセキュリティが危険にさらされます。顧客は、秘密鍵への非常に制限されたアクセスを許可する方法を作成する必要があります。一方、公開鍵はだれでも参照できます。唯一の要件は、公開鍵を変更できないようロックするか、または検証し公開鍵の変更を検出できるようにします。TRAVEO™ T2G MCU では、公開鍵は SFlash に保存され、上記の CoT セクションで定義される SECURE_HASH で検証します。秘密鍵と公開鍵の生成と使用方法については、Appendix A - 公開鍵と秘密鍵の作成例で説明します。

6.1 コードの署名

起動時にユーザアプリケーションなどのコードを検証するためには、デジタル署名を作成し、ビルド時にコードに付属する必要があります。コード自体は暗号化された形式でフラッシュに格納されませんが、デジタル署名は暗号化されます。デジタル署名は SHA-256 ハッシュ関数により生成され (a)、最大 RSA-4096 ビット暗号化の秘密鍵を使用して暗号化されます (b)。デジタル署名が暗号化されるのは、秘密鍵にアクセスできない第三者によって有効なコードに署名を付属させないためです。図 4 を参照してください。

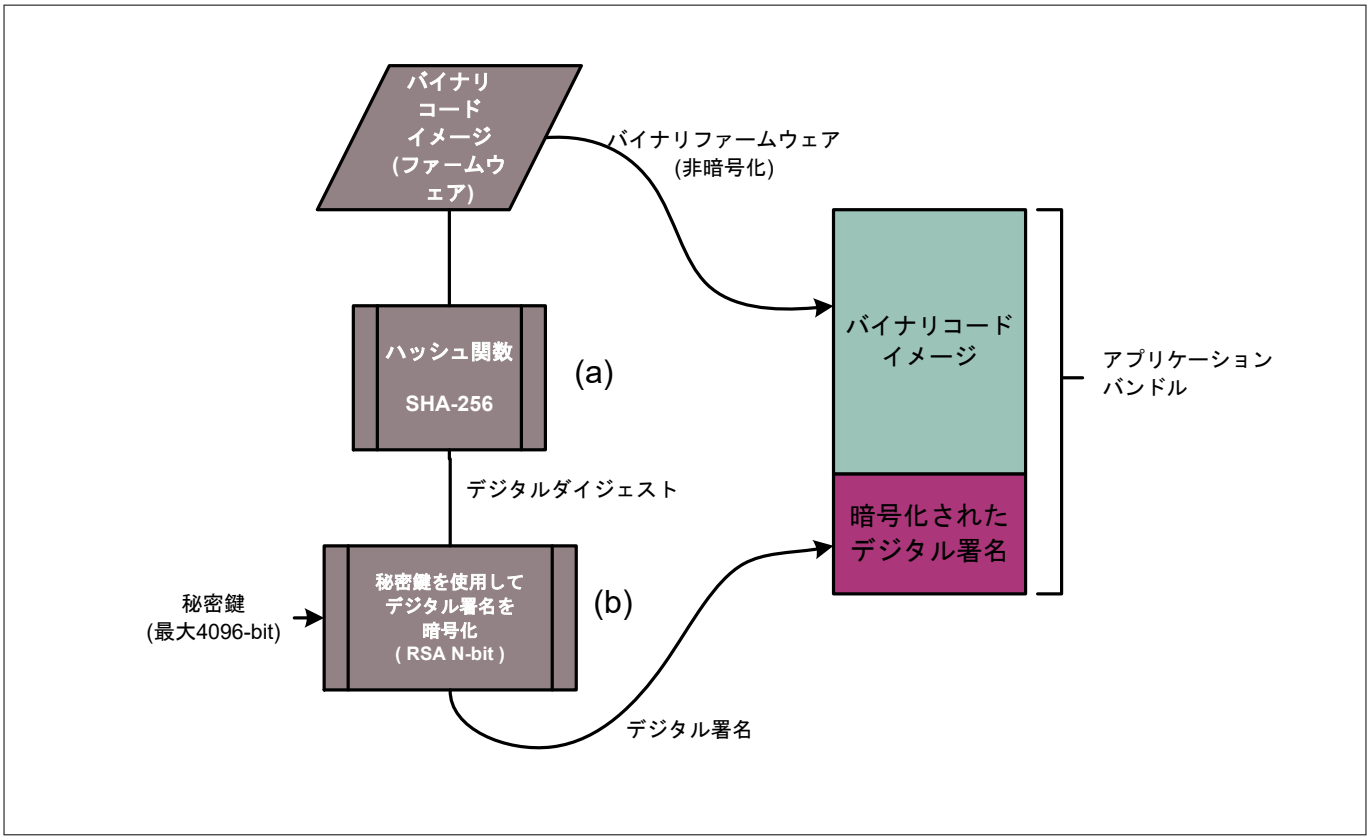


図 4 暗号化されたデジタル署名の生成

注: RSA-2048, 3072, および 4096 のサポートについては、デバイス固有のデータシート[1] (Part Number/ Ordering Code Nomenclature, Hardware option のセクション) を参照してください。

6.2 コード検証

セキュアシステムは、製造元または信頼できるソースによって作成されていないコードを検出する必要があります。信頼できないコードが検出された場合、既知のパスを介して安全状態に遷移する必要があります。これは、ファームウェアが意図的または偶発的に破損していないことも検証します。

検証には、データ, 署名, および暗号鍵の 3 つの要素が必要です。データと署名はペアです。鍵は変更できない場所に格納されます。

- **データ**: 組込みシステムのファームウェアを構成する実行可能コードと定数の両方が含まれます。通常、フラッシュメモリに格納され、一括して変更可能です。システムによっては、変更または更新します。そのため、このデータが既知のソースからのもので、偶発または悪意あるイベントによって破損してないか判定する必要があります。
- **デジタル署名**: データブロックのハッシュで、使用されるハッシュアルゴリズムは SHA-256 です。デジタル署名のみで、データの完全性を検証できます。デジタル署名を暗号鍵で暗号化し、データが信頼できるソースからのものか否かを判別します。
- **暗号鍵**: 公開鍵または秘密鍵です。このアプリケーションノートで説明するシステムでは、公開鍵はデバイスに格納され、秘密鍵は開発者によって保護されます。公開鍵は次の 2 つの方法のいずれかで保護する必要があります。
  1. *鍵のソースを検証する。*既知のソースまたはサーバとある種の通信で実現可能です。必要なときに既知のサーバと容易に通信できないデバイスには実用的ではありません。
  2. *変更できないメモリに格納される鍵を使用し、鍵自体を検証する。*組込みシステムでは最も可能性のあるオプションです。TRAVEO™ T2G MCU では、ハッシュは公開鍵, フラッシュブートコード, およびトリム値を含む領域によって計算されます。このハッシュはワンタイムプログラマブルの eFuse に格納され「SECURE_HASH」と呼ばれます。

ユーザアプリケーションブロックのバイナリコードブロック (アプリケーションバンドル) には、ビルド時に生成された暗号化デジタル署名が含まれます。セキュアイメージアプリケーションもこの形式を使用します。コードブロック検証のため、バイナリコードイメージにハッシュ関数 (SHA-256) を適用しデジタル署名またはダイジェストを計算します。次に、暗号化されたデジタル署名は公開鍵を使用して復号化しデジタル署名を生成します。最後に計算されたデジタルダイジェストと復号化されたデジタルダイジェストが一致することをチェックします。一致した場合、コード検証は完了です。図 5 を参照してください。

6 コードの署名と検証

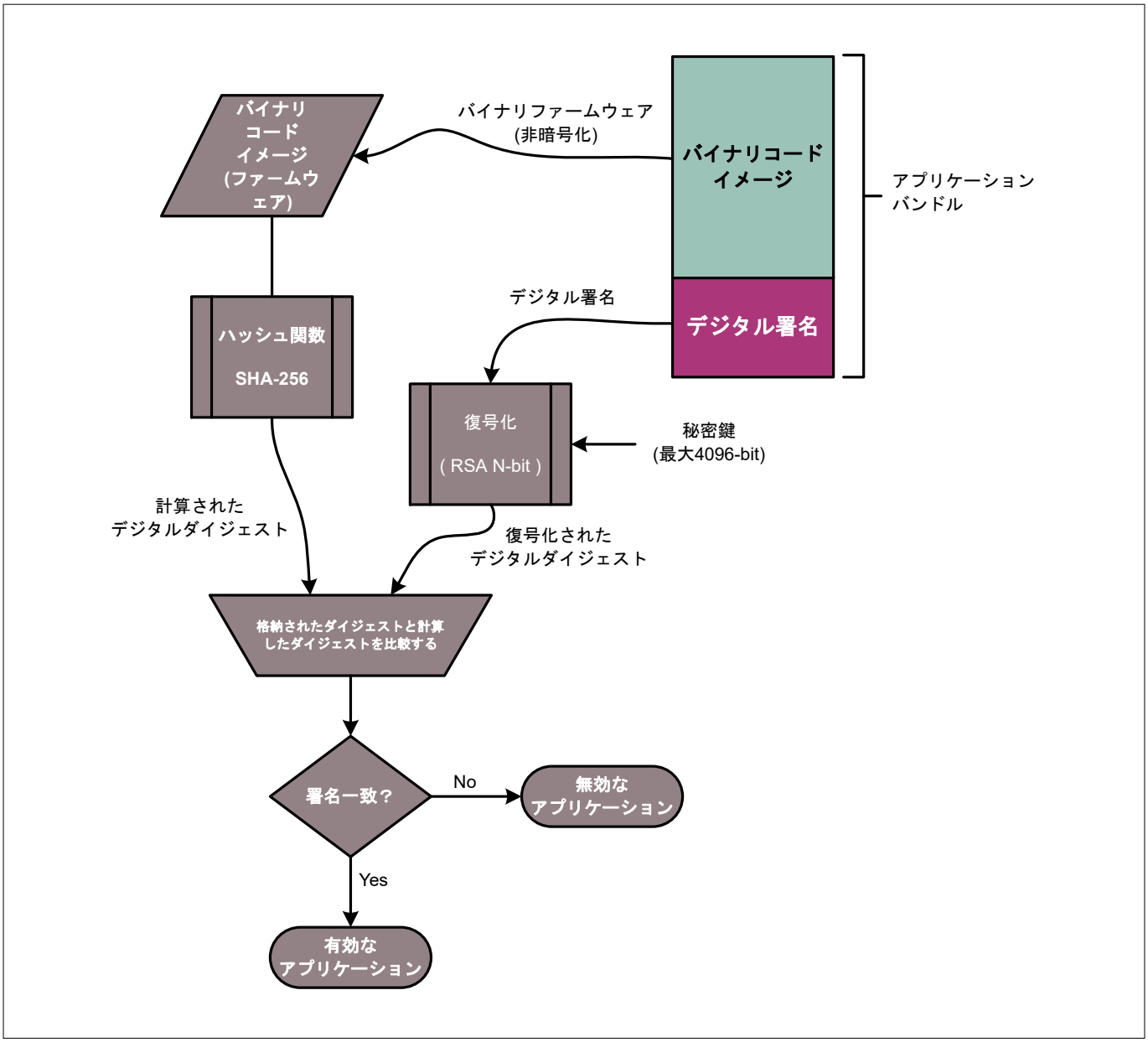


図 5　　コード検証

注:　Crypto メモリ バッファの不適切な初期化が原因で、アプリケーション認証による起動後に Crypto ECC エラーが設定される場合があります。したがって、ユーザ ソフトウェアは、アプリケーション認証による起動中に生成される Crypto ECC エラーをクリアまたは無視する必要があります。

7 リソース保護

リソース保護は、実行中にメモリ領域またはレジスタ領域にアクセス可能なバスマスタまたはタスクのみアクセスできることを意味します。アクセスは、読出し, 書込み, または実行の組合せです。TRAVEO™ T2G MCU は、この機能を追加できる「保護ユニット」と呼ばれるブロックがあります。保護ユニットは、フラッシュ, SRAM, 周辺機能, および I/O を含む複数の保護領域を作成できます。保護領域は、CPU, タスク, または両方によって制限できます。保護ユニットには主に、MPU, SMPU, PPU, および SWPU の 4 つのタイプがあります。

- 各 CPU は独自の MPU を持ちます。MPU は、他の保護ユニットとは異なり、単一のバスマスタに関連する保護属性のみに対応します。MPU には保護コンテキストパラメータはありません。保護属性は、ユーザ/特権、読出し/書込み/実行です。MPU は各バスマスタ固有であり、さまざまなスレッドまたはタスクからのリソース保護を提供します。
- SMPU は、複数のマスタによって使用されるメモリ領域を保護し、MPU のすべての属性, 保護コンテキスト, および非セキュア属性を持ちます。セキュアイメージは、SMPU を使用し、非セキュアアプリケーションからセキュアセクションメモリへのアクセスを制限します。フラッシュ書き込み操作に使用されるレジスタは、SROM コードのみがこれらの操作にアクセスできるよう制限されています。これにより、フラッシュメモリへの偶発的な書込みおよび消去を防止します。また、暗号化操作に使用する SRAM およびレジスタは安全のため保護されます。
- PPU は、周辺レジスタの保護するため特別に設計され、SMPU に似ています。特定の周辺領域保護のためにハードウェア実装され、固定機能です。したがって、このタイプの PPU はアドレスとサイズのパラメータは設定できませんが、保護属性を設定できます。
- SWPU は、フラッシュ (プログラム/消去) および eFuse (読出し/書込み) へのアクセス制限を実装できます。SWPU は 2 つの部分に分割され、SFlash に格納されます。1 つは、ブートプロセスによって構成され、変更できません。2 つめは、アプリケーション固有のアクセス制限を追加するためアプリケーションソフトウェアで使用できます。SFlash の特定の領域に書込み NORMAL_PROVISIONED ライフサイクルステージで更新できます。また、SROM API を使用して更新も可能です。ROM/フラッシュブートは、SFlash から SWPU の 2 つの部分を読み出して設定します。SWPU は、FWPU, ERPU, および EWPU で構成されます。

7.1 ブートプロテクション

一部の保護ユニットは、起動プロセス中に設定され、再設定はできません。これらの保護ユニットはセキュアなシステムを提供し、システムコールへの信頼性の高いアクセスを提供するために必要です。ブートプロテクションの詳細は、アーキテクチャ TRM[2]の BootROM 章を参照してください。

7.2 アプリケーション保護

SWPU を使用して、ブートプロセス中にフラッシュと eFuse へのアクセス保護を設定でき、アプリケーション保護ユニットと呼びます。アプリケーション保護ユニットは、デバイスが NORMAL_PROVISIONED ライフサイクル状態の SFlash に設定できます。アプリケーション保護 SWPU は、最大 16 領域の FWPU と最大 4 領域の ERPU と EWPU を持ちます。SWPU の詳細については、アーキテクチャ TRM[2]の Protection Unit 章を参照してください。

表 1 に、アプリケーション保護のデフォルト値を示します。

表 1 アプリケーション保護 SWPU のデフォルト値

SWPU レイアウト	説明	デフォルト値
PU_OBJECT_SIZE (4 バイト)	SWPU オブジェクトサイズ	0x00000030
N_FWPU (4 バイト) 最大 16 エントリ	FWPU 数	0x00000000
N_ERPU	ERPU 数	0x00000001
ERPU0_SL_OFFSET (4 バイト)	保護オフセットアドレス設定	0x00000068

(続く)

表 1 (続き) アプリケーション保護 SWPU のデフォルト値

SWPU レイアウト	説明	デフォルト値
ERPU0_SL_SIZE (4 バイト)	領域サイズと ERPU0 のイネーブル	0x80000018
ERPU0_SL_ATT (4 バイト)	スレーブ属性	0x00FF0007
ERPU0_SL_ATT (4 バイト)	マスタ属性	0x00FF0007
N_EWPU	ERPU 数	0x00000001
EWPU0_SL_OFFSET (4 バイト)	保護オフセットアドレス設定	0x00000068
EWPU0_SL_SIZE (4 バイト)	領域サイズと ERPU0 のイネーブル	0x80000018
EWPU0_SL_ATT (4 バイト)	スレーブ属性	0x00FF0007
EWPU0_SL_ATT (4 バイト)	マスタ属性	0x00FF0007

アプリケーション保護の設定方法については、Appendix F - アプリケーション保護設定を参照してください。

7.3 セキュリティエンハンス PPU コンフィグレーション

マジックナンバーがセキュリティマーカ(TOC2_SECURITY_UPDATES_MARKER)に設定されている場合、ブートプロセスはセキュリティと安全性の強化のため、以下の PPU を設定します。この機能は TRAVEO™ T2G Body Controller Entry/High デバイスで有効であり、TRAVEO™ T2G Cluster デバイスではセキュリティマーカ設定なしに適用されます。表 2 にセキュリティマーカによって設定される PPU を示します。

表 2 セキュリティマーカによる PPU 設定一覧

PPU 名	開始アドレス	サイズ (バイト)	Access for PC > 0(スレーブ属性)	Access for PC > 0(マスタ属性)
Programmable PPU 11	0x40201000	32	PC1: Full access PC1: Full access	PC1: Full access PC1: Full access
Programmable PPU 12	0x402013c8	4	PC1: Full access PC1: Full access	PC1: Full access PC1: Full access
Programmable PPU 13	0x40201300	256	PC1: Full access PC1: Full access	PC1: Full access PC1: Full access
PERI_MS_PPU_FX_PERI_GR2_GROUP (PPU index=4)	0x40004050	4	PC1: Read Only PC1: Read Only	PC1: Read Only PC1: Read Only

Programmable PPU 11 と 13 は、PERI_MS_PPU_FX_CPUSS_CM0 と組合せ、HSM ソフトウェアとアプリケーションソフトウェアの分離に役立ちます。例えば、Programmable PPU 11 と 13 はアプリケーションソフトウェアのアクセスを許可し、PERI_MS_PPU_FX_CPUSS_CM0 は HSM ソフトウェアのアクセスを許可します。その結果、CPUSS_AP_CTL レジスタは HSM ソフトウェアによって排他的に制御され、CPUSS_CM0_CLOCK_CTL、RAM0_PWR_CTL および RAM1_PWR_CTL はアプリケーションソフトウェアによって制御できます。

Programmable PPU 12 は、CPUSS_ECC_CTL レジスタを保護するために使用します。このレジスタは ECC エラー挿入機能を提供します。ECC エラー挿入は、メモリの ECC ロジックをテストするための重要なツールです。TRAVEO™ T2G MCU では、メモリが複数の CPU で共有される他のマイクロコントローラ同様、ECC エラー挿入機能が 1 つの CPU またはデバッガによって悪用され、他の CPU が使用するメモリ領域のデータを操作してセキュリティを損なう可能性があります。このようなリスクを軽減するために少なくとも HSM で使用されるメモリでは、ECC エラー挿入機能を無効にするか、その使用を制限することを推奨します。

ECC ロジックはデバイスの起動時のみテストされると想定され、起動完了後の通常デバイス操作では、ECC エラー挿入機能は必要ありません。セキュリティリスクを軽減するためには、適切な PPU 構成を使用して ECC エラー挿入制御レジスタへのアクセスをブロックし、ECC テスト完了後に ECC エラー挿入ロジックを無効にすることを推奨します。より厳密には、ユーザは ECC エラー挿入制御レジスタへの書込み不可を確認する必要があります。起動後に ECC エラー挿入制御レジスタへの書込みアクセスが必要な場合、ユーザは信頼できるソフトウェアのみがこれらのレジスタに書込みできるようにする必要があります。

図 6 に Programmable PPU 11, 12, 13, および PERI_MS_PPU_FX_CPUSS_CM0 の使用例を示します。Programmable PPU は、Fixed PPU よりも高い優先度を持ちます。同じ PPU グループでは番号が小さいほど高い優先度を持ちます。

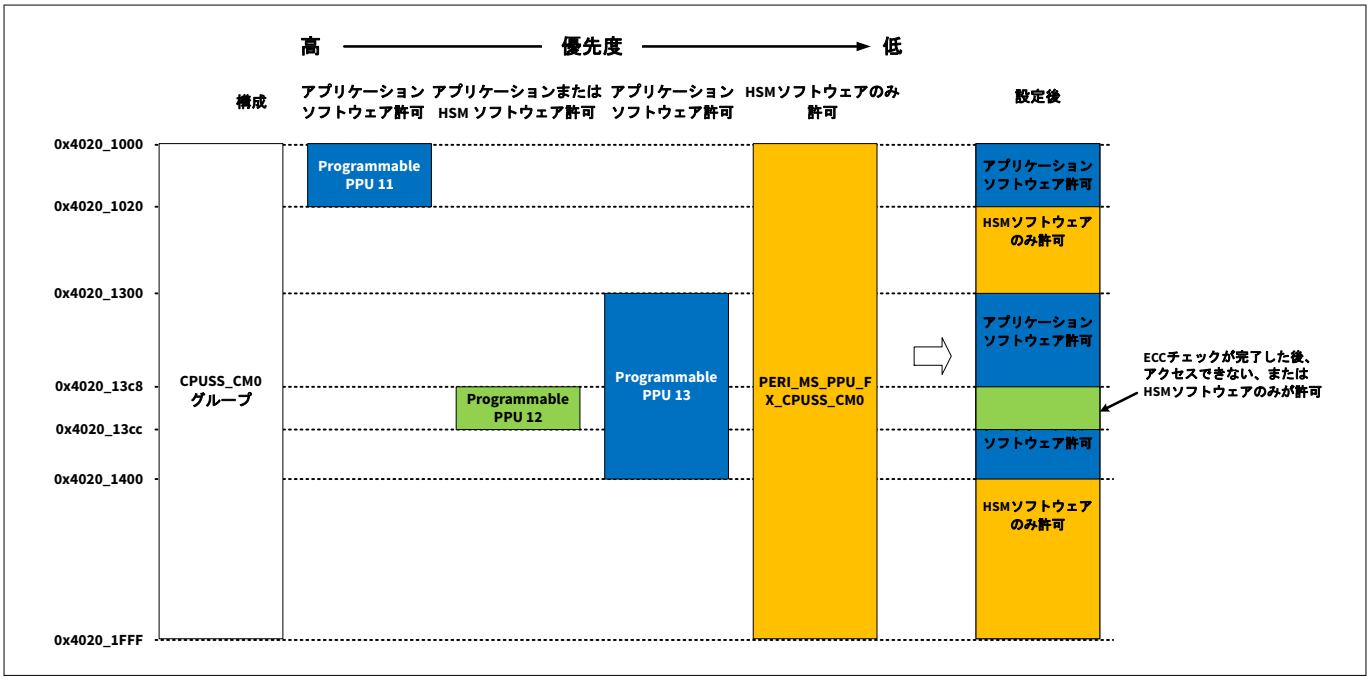


図 6　　PPU の使用例

***注**:　　セキュアシステムのために、これらの PPU は、HSM ソフトウェアで設定することを推奨します。*

PERI_GR2_SL_CTL レジスタへの誤った書込みにより、MCU コア機能ブロックへのクロックが停止する可能性があります。PERI_MS_PPU_FX_PERI_GR2_GROUP は、PERI_GR2_SL_CTL への偶発的な書込みアクセスから保護します。

TRAVEO™ T2G Body Controller Entry/High デバイスでセキュリティマーカが設定されていない場合、Programmable PPU 11、12、および 13 は構成されません。また、PERI_MS_PPU_FX_PERI_GR2_GROUP はデフォルト値です。

セキュリティマーカの配置については、TOC2 を参照してください。この設定例は、セキュリティエンハンス PPU 設定例を参照してください。

7.4　　デバッグおよびテストアクセスポート制限

TRAVEO™ T2G MCU は、ライフサイクルステージと保護状態に応じて、ブートプロセス中にテストアクセスポート設定のため MPU とデバッグアクセスポートを構成します。表 3 に、各アクセス制限の保管場所と展開条件を示します。

表 3　　デバッグおよびテストアクセスポート制限

アクセス制限	コンフィグレーション	デフォルト値	格納場所	展開するライフサイクルステージ	保護状態
Normal Access Restriction	SFlash へのプログラミングにより設定	0x00000080 表 4 を参照。	SFlash	NORMAL_PROVISIONED または SECURE_W_DEBUG	NORMAL_PROVISIONED の NORMAL 保護状態、または SECURE_W_DEBUG の SECURE 保護状態
Normal Dead Access Restriction		0x00000000 表 4 を参照。			NORMAL、SECURE、DEAD の保護状態。次の注を参照してください。
Secure Access Restriction	SECURE への TransitiontoSecure API 実行により設定	0x00000000 表 4 を参照。	eFuse	SECURE	SECURE 保護状態
Secure Dead Access Restriction		0x00000000 表 4 を参照。			DEAD 保護状態

***注**:　Normal および Normal Dead Access Restriction は更新できますが、Secure および Secure Dead Access Restriction は更新できません。*

***注**:　SECURE_WITH_DEBUG、NORMAL_PROVISIONED では、フラッシュブート時に既存の保護モードが DEAD ブランチに保持されます。詳細については、アーキテクチャ TRM [2] の Flash Boot Flow を参照してください。*

表 4 にアクセス制限の設定を示します。設定パラメータは、すべてのアクセス制限に共通です。

表 4　　アクセス制限パラメータ

フィールド名	ビット	説明
AP_CTL_M0_DISABLE	[1:0]	00 – Enable M0-DAP 01 – Disable M0-DAP 1x – Permanently Disable M0-DAP 表 5 を参照。
AP_CTL_M4_DISABLE	[3:2]	00 – Enable M4-DAP 01 – Disable M4-DAP 1x – Permanently Disable M4-DAP 表 5 を参照。
AP_CTL_SYS_DISABLE	[5:4]	00 – Enable SYS-DAP 01 – Disable SYS-DAP 1x – Permanently Disable SYS-DAP 表 5 を参照。

(続く)

表 4 (続き) アクセス制限パラメータ

フィールド名	ビット	説明
SYS_AP_MPU_ENABLE	[6]	メイン/ワークフラッシュ, RAM0, SFlash, および MMIO フィールドの設定に従い、ブートプロセスがシステムアクセスポートの MPU を設定し、ロックすることを示します。 これら 6 つのフィールド (TCM や ROM など) でカバーされないメモリ領域へのアクセスは無効にする必要があります。 0 –無効:ブートプロセスによって MPU を設定しません。アプリケーションソフトウェアは MPU を設定できます。 1 –有効:ブートプロセスによって設定に従い MPU を設定します。MPU は PPU によって保護され、アプリケーションソフトウェアは MPU を再設定できません。
DIRECT_EXECUTE_DISABLE	[7]	DirectExecute システムコールを無効にします (ソフトウェアで実装)。 このフィールドは、NAR では"1"に固定されています。
FLASH_ENABLE	[10:8]	システムアクセスポートを介してアクセス可能なメインフラッシュを示します。エリアの下部から始まるフラッシュの一部のみがアクセスできます。エンコードは次のとおりです。 “0”: 全領域 “1”: 7/8th “2”: 3/4th “3”: 1/2 “4”: 1/4th “5”: 1/8th “6”: 1/16th “7”: なし
RAM0_ENABLE	[13:11]	システムアクセスポートを介してアクセス可能な SRAM0 を示します。エリアの下部から始まるフラッシュの一部のみがアクセスできます。エンコードは FLASH_ENABLE と同じです。
WORK_FLASH_ENABLE	[15:14]	システムアクセスポートを介してアクセス可能なワークフラッシュを示します。エリアの下部から始まるフラッシュの一部のみがアクセスできます。エンコードは次のとおりです。 “0”: 全領域 “1”: 1/2 “2”: 1/4th “3”: なし
SFLASH_ENABLE	[17:16]	システムアクセスポートを介してアクセス可能な SFlash を示します。エリアの下部から始まるフラッシュの一部のみがアクセスできます。エンコードは次のとおりです。 “0”:全領域 “1”: 1/2 “2”: 1/4th “3”: なし

(続く)

表 4 (続き) アクセス制限パラメータ

フィールド名	ビット	説明
MMIO_ENABLE	[19:18]	システムアクセスポートを介してアクセス可能な MMIO 領域を示します。エンコードは次のとおりです。 “0”: すべての MMIO レジスタ “1”: アクセス可能な IPC MMIO レジスタのみ (システムコール用) “2”, “3”: MMIO アクセスなし

表 5 にデバッグポートアクセス制限設定について示します。

表 5 デバッグポートアクセス制限

要素	説明
Enable M0/M4/SYS-DAP	対応する DAP は有効です。
Disable M0/M4/SYS-DAP	対応する DAP は一時的に無効です。DAP はアプリケーションソフトウェアによって再度有効にできます。
Permanently Disable M0/M4/SYS-DAP	対応する DAP は永久的に無効です。

注: Normal および Normal Dead Access Restriction は更新できますが、制限を緩和する設定はできません。例えば、Disable DAP 設定を Enable DAP 設定に変更できません。

7.4.1 デバッガによるシステムコール起動の最小要件

RMA 移行のシステムコールを実行するため、必要なアクセスの有効化にはいくつかの考慮する点があります。

セキュアシステムを検討する場合、初期アクセス制限設定の一部として、すべての DAP を一時的に無効にすることを推奨します。また、Enable M4-DAP 設定の場合、CM4 CPU の電源投入後、任意のメモリアドレスからコード実行を開始できることに注意してください。これは、ROM/フラッシュブートが、SWD/JTAG ピンを構成する場合において可能です。最悪のシナリオでは、CM0+アプリケーション起動前に発生する場合があります。

CM0+アプリケーション(例えば、HSM ソフトウェアなど)が、メインアプリケーション(例えば、フラッシュの HSM 部分に格納されたキーなど)から保護する必要があるなど特定のアセットを持つ場合、アクセス制限により CPU アクセスポートを有効化しないことを推奨します、そうでない場合、CM0+が保護を構成にする前に、CM4 CPU を使用して、HSM メモリにアクセスする可能性があります。

Sys-DAP を有効するために、ユーザアプリケーションは以下の手順を実行します。

これらの手順を実行するためには、すべての DAP を「Disable」(一時的に無効)に設定し、SYS_AP_MPU_ENABLE を「0」(無効)に設定する必要があります。

1. 必要なすべての DAP ピンを設定してください。
2. Sys_DAP MPU を必要なリソースにアクセスできるように設定してください。(例えば、IPC MMIO、SRAM など)
3. Sys_DAP MPU を TransitiontoRMA API によって追加で使用される SRAM 領域へのアクセスしないよう設定してください。これは、SRAM0+2KB から始まる 2KB の SRAM 領域です。
4. Sys_DAP MPU を保護する PPU を設定してください。
5. IPC MMIO および SRAM などアクセスが必要なリソースを PPU/SMPU 設定が許可していることを確認してください。
6. ユーザソフトウェアで割込みを初期化してください。システムコール割込み処理のため IRQ0 および IRQ1 を有効にします。両割り込みのベクタを定義します。
7. 必要なすべての DAP を有効にしてください。

***注**: 上記の設定のいずれかが失敗している場合、システムコールは使用できません。*

***注**: DAP を有効にする前に、手順 1 から 5 を実行してください。*

***注**: セキュリティ要件に基づいて、個々の CPU へこれらの手順の割当てを決定する必要があります。*

詳細については、デバッグアクセスポート認証を参照してください。

8 セキュアシステムの構築

ここでは、セキュアシステム構成の各部分について説明し、それらを生成する方法について説明します。

CoT を使用して完全なセキュアシステムの構築は、単純なアプリケーションを生成するよりも複雑です。ユーザコードだけでなく、単純な (非セキュア) システムでは通常必要としないいくつかの要素を含める必要があります。以下に、セキュアシステムの構築にプログラムする必要のあるメモリセクションを示します。

- SECURE_HASH (eFuse)
- DAP 設定 (eFuse)
- ライフサイクルステージ (eFuse)
- 公開鍵 (SFlash)
- TOC2 (SFlash)
- セキュアイメージブロック (ユーザフラッシュ)
- ユーザアプリケーションブロック (ユーザフラッシュ)

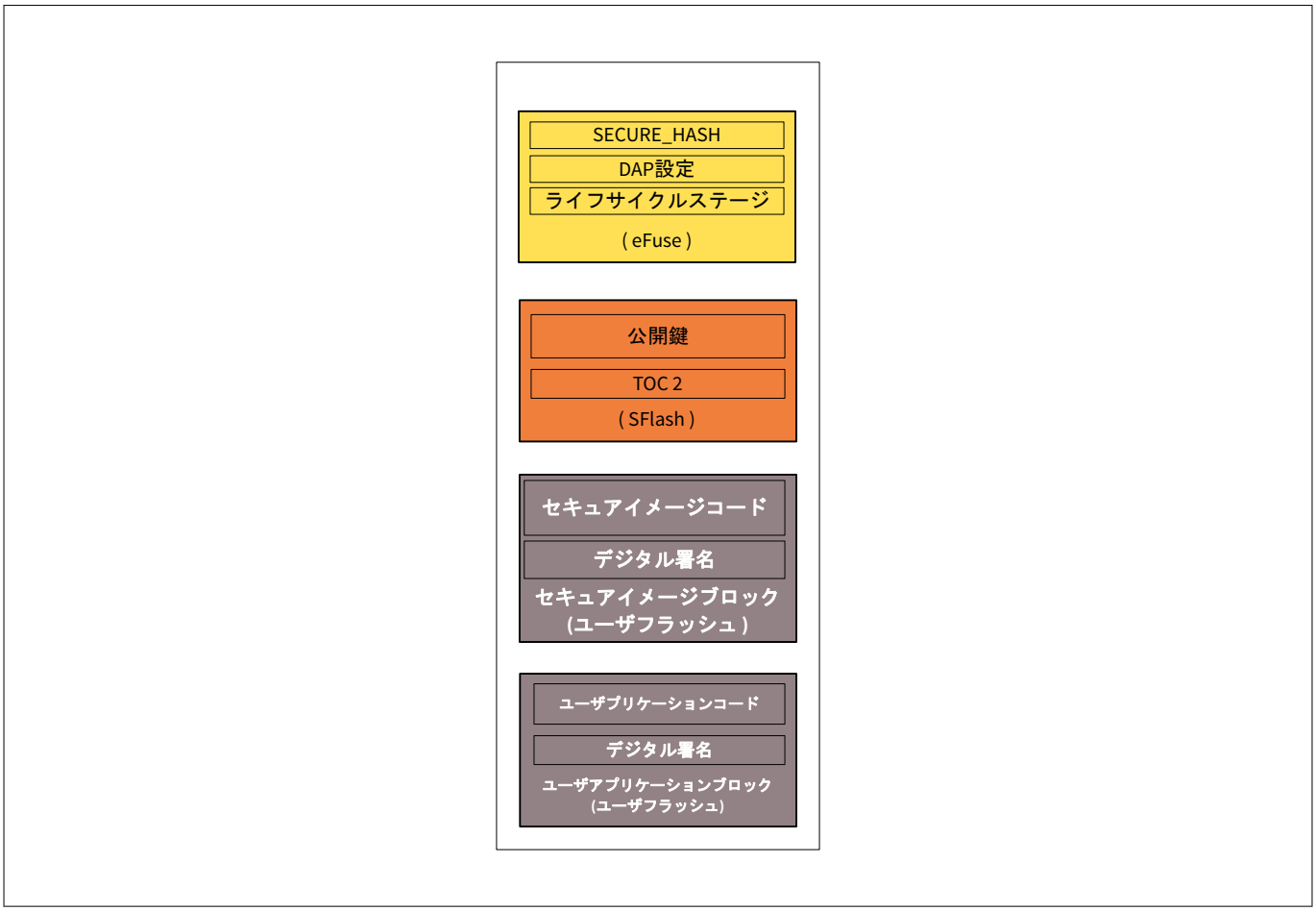


図 7 セキュアシステムの構築

1 つのブロックの構築は、別のブロックの構築に依存する場合があります。例えば、TOC2 は、セキュアイメージブロックとユーザアプリケーションブロックの両方の開始アドレスを含みます。eFuse に格納される SECURE_HASH は、SFlash のすべてに依存します。

いくつかの項目は、適切な SROM ファームウェアを介して暗黙的にプログラムされることに注意してください。

8.1 TOC2

SFlash には、TOC1 と TOC2 の 2 つのテーブルセクションがあります。TOC1 は、内部目的で使用しユーザは編集できません。TOC2 は、各アプリケーションの配置を示すために使用します。TOC2 では、セキュアイメージは「ユーザアプリケーション」、メインユーザアプリケーションは「CM4 または CM7 コア x ユーザアプリケーション」と呼びます。(x は CPU 番号を示します)

オフセット 0x1F8 (表 6 を参照) は、起動シーケンス高速化のために変更の可能性がある 2 つのパラメータ、ブートクロック周波数パラメータの「CLOCK_CONFIG」およびデバッグパラメータの「LISTEN_WINDOW」があります。TOC2 ではこれらを表に定義される値に変更できます。

表 6 に、TOC2 の要素を示します。

表 6 TOC2 の要素

オフセット	説明	デフォルト値
0x00	オフセット 0x00 から始まる CRC 計算のオブジェクトサイズ(バイト数)	0x00001FC
0x04	マジックナンバー(Fixed: 0x01211220)	0x01211220
0x08	SMIF 構成構造を表すポインタの Null 終了テーブル	0x00000000
0x0C	ファーストユーザアプリケーションオブジェクトのアドレス	0x10000000
0x10	ファーストユーザアプリケーションオブジェクトのフォーマット 0: Basic、1: CySAF、2: Simplified	0x00000000 セキュアブート時は“1”に設定します。
0x14	セカンドユーザアプリケーションオブジェクトのアドレス ファーストアプリケーション検証が失敗した場合、セカンドユーザアプリケーションを検証します。	0x00000000
0x18	ファーストユーザアプリケーションオブジェクトのフォーマット 0: Basic、1: CySAF、2: Simplified	0x00000000 セキュアブート時は“1”に設定します。
0x1C	ファースト CM4 または CM7 コア 1 ユーザアプリケーションオブジェクトのアドレス	0x00000000
0x20	セカンド CM4 または CM7 コア 1 ユーザアプリケーションオブジェクトのアドレス	0x00000000
0x24	ファースト CM4 または CM7 コア 2 ユーザアプリケーションオブジェクトのアドレス	0x00000000
0x28	セカンド CM4 または CM7 コア 2 ユーザアプリケーションオブジェクトのアドレス	0x00000000
0xFC	有効なマジックナンバーの場合、セキュリティ強化用の保護ユニットを有効にします。 詳細については、セキュリティエンハンス PPU コンフィグレーションを参照してください。 TRAVEO™ T2G クラスタデバイスでは無効です。	0x00000000 マジックナンバー = 0xFEDEEDDF
0x100	SECURE_HASH で検証する追加オブジェクトの数	0x00000003
0x104	署名検証鍵のアドレス(ない場合は”0”) オブジェクトは署名固有の鍵です。RSA の場合は公開鍵です。	0x00000000 SFlash に配置する場合: 0x17006400

(続く)

表 6 (続き) TOC2 の要素

オフセット	説明		デフォルト値
0x108	アプリケーション保護のアドレス (この領域は変更しないでください)		0x17007600
0x10C	予約済み (この領域は変更しないでください)		0x00000000
0x110-0x1F0	必要に応じてオブジェクト追加、ない場合は“0”		0x00000000
0x1F8	TOC2_FLAGS: デフォルト設定を制御します。 Bits [1:0]: CLOCK_CONFIG クロック周波数構成を示す。フラッシュブート実行後、クロックは同じままです。		デフォルト値はレジスタ TRM[2]を参照してください。
	値 [1:0]	**説明**	
	0x0	8 MHz, IMO, no FLL	
	0x1	25 MHz, IMO + FLL	
	0x2	50 MHz, IMO + FLL	
	0x3	ROM ブート設定を使用する (100 MHz)	
	Bits [4:2]: LISTEN_WINDOW デバッグポート接続に十分な時間確保のため Listen window を決定する		
	値 [4:2]	**説明**	
	0x0	20 ms	
	0x1	10 ms	
	0x2	1 ms	
	0x3	0 ms (Listen window なし)	
	0x4	100 ms	
	その他	予約済み	
	Bits [6:5]: SWJ_PINS_CTL SWJ ピンがフラッシュブートによって SWJ モードで設定するかどうかを決定する。		
	値 [6:5]	**説明**	
	0x0	フラッシュブートで SWJ ピンを有効にしない。Listen window はスキップされます。	
	0x1	フラッシュブートで SWJ ピンを有効にしない。Listen window はスキップされます。	
	0x2	フラッシュブートで SWJ ピンを有効にする。	
	0x3	フラッシュブートで SWJ ピンを有効にしない。Listen window はスキップされます。	

(続く)

表 6 (続き) TOC2 の要素

オフセット	説明		デフォルト値
	Bits [8:7]: APP_AUTH_CTL アプリケーションイメージのデジタル署名検証 (認証) の実行を決定する		
	値 [8:7]	**説明**	
	0x0	認証有効	
	0x1	認証無効	
	0x2	認証有効 (推奨)	
	0x3	認証有効	
	Bits [10:9]: FB_BOOTLOADER_CTL フラッシュブートの内部ブートローダの有効/無効を決定する		
	値 [10:9]	**説明**	
	0x0	内部ブートローダ無効	
	0x1	ブートローダ条件が一致した場合、内部ブートローダを起動します。 以下の条件を参照してください。	
	0x2	内部ブートローダ無効	
	0x3	内部ブートローダ無効	

ブートローダは次の条件で有効になります。

- フラッシュのコード開始アドレスの 2 つのワードが'0xFFFFFFFF'である必要があります。
- TOC2 が有効であり内部ブートローダが TOC2_FLAGS.FB_BOOTLOADER_CTL ビットによって有効 (デフォルト) になっている、または TOC2 が無効である。
- 保護状態が、SECURE でも SECURE_DEAD のいずれでもない。
- 1 秒間の待機時間中にデバッガ接続が検出されない。

ブートローダの有効化条件が満たされている場合、ファーストアプリケーションを消去すると、ブートローダが起動されセカンドアプリケーションは起動されません。したがって、不要な場合は、ブートローダを無効 (TOC2_FLAGS.FB_BOOTLOADER_CTL=0x2) にすることを推奨します。

8.1.1 コンフィグレーション

適切な TOC2 を生成するために、以下のコードをプロジェクトに追加できます。

```
/** Flashboot parameters */
#define CY_SI_FLASHBOOT_FLAGS ((CY_SI_FLASHBOOT_CLK_100MHZ << CY_SI_TOC_FLAGS_CLOCKS_POS) \
                               | (CY_SI_FLASHBOOT_WAIT_20MS << CY_SI_TOC_FLAGS_DELAY_POS) \
                               | (CY_SI_FLASHBOOT_SWJ_ENABLE << CY_SI_TOC_FLAGS_SWJEN_POS) \
                               | (CY_SI_FLASHBOOT_VALIDATE_ENABLE <<
CY_SI_TOC_FLAGS_APP_VERIFY_POS) \
                               | (CY_SI_FLASHBOOT_FBLOADER_DISABLE <<
CY_SI_TOC_FLAGS_FBLOADER_ENABLE_POS))
/** TOC2 in SFlash */
CY_SECTION(".cy_toc_part2") __USED static const cy_stc_si_toc_t cy_toc2 =
{
    .objSize         = CY_SI_TOC2_OBJECTSIZE,           /* Offset+0x00: Object Size (Bytes)
excluding CRC */
    .magicNum        = CY_SI_TOC2_MAGICNUMBER,          /* Offset+0x04: TOC2 ID (magic number)
*/
    .smifCfgAddr     = 0UL,                             /* Offset+0x08: SMIF config list
pointer */
    .cm0pappAddr1    = CY_SI_SECURE_FLASH_BEGIN,        /* Offset+0x0C: App1 (CM0+ First User
App Object) addr */
    .cm0pappFormat1  = CY_SI_APP_FORMAT_CYPRESS,        /* Offset+0x10: App1 Format */
    .cm0pappAddr2    = CY_SI_USERAPP_FLASH_BEGIN,       /* Offset+0x14: App2 (CM0+ Second User
App Object) addr */
    .cm0pappFormat2  = CY_SI_APP_FORMAT_BASIC,          /* Offset+0x18: App2 Format */
    .cm4_71appAddr1  = CY_SI_CM471_1stAPP_FLASH_BEGIN,  /* Offset+0x1C: App3 (CM4/CM7_1 1st
User App Object) addr */
    .cm4_71appAddr2  = CY_SI_CM471_2ndAPP_FLASH_BEGIN,  /* Offset+0x20: App4 (CM4/CM7_1 2nd
User App Object) addr */
    .cm72appAddr1    = CY_SI_CM72_1stAPP_FLASH_BEGIN,   /* Offset+0x24: App5 (CM7_2 1st User
App Object) addr */
    .cm72appAddr2    = CY_SI_CM72_2ndAPP_FLASH_BEGIN,   /* Offset+0x28: App6 (CM7_2 1st User
App Object) addr */
    .reserved1       = 0UL,                             /* Offset+0x2C-0xFB: Reserved area
212Bytes */
    .securityMarker  = CY_SECURITY_NOT_ENHANCED,        /* Offset+0xFC Security Enhance Marker
*/
    .shashObj        = 3UL,                             /* Offset+0x100: Number of verified
additional objects */
    .sigKeyAddr      = CY_SI_PUBLIC_KEY,                /* Offset+0x104: Addr of signature
verification key */
    .swpuAddr        = CY_SI_SWPU_BEGIN,                /* Offset+0x108: Addr of SWPU Objects */
    .toc2Addr        = 0UL,                             /* Offset+0x10C:
TOC2_OBJECT_ADDR_UNUSED */
    .addObj          = 0UL,                             /* Offset+0x110-0x1F4: Reserved area
232Bytes */
    .tocFlags        = CY_SI_FLASHBOOT_FLAGS,           /* Flashboot flags stored in TOC2 */
    .crc             = 0UL,                             /* Offset+0x1FC: Reserved area 1Byte */
};
```

8.2 ユーザアプリケーションブロック

セキュアなアプリケーションは、公開暗号鍵で検証する必要があり、アプリケーションはデジタル署名を含むサイプレスセキュアアプリケーションフォーマット(CySAF)を使用する必要があります。ユーザアプリケーションフォーマットは、TOC2 で「ファースト/セカンドユーザアプリケーションオブジェクトのフォーマット(オフセット=0x10/0x18)」で指定されます。

これにより、フラッシュブートは、アプリケーション実行前のブートプロセス中に検証できます。アプリケーションフォーマットは、アプリケーションバイナリ, アプリケーションメタデータ, および暗号化されたデジタル署名をカプセル化します。このフォーマットには、CM0+と CM4 の両方のイメージを格納できますが、セキュアイメージでは CM0+のイメージのみが必要です。ユーザアプリケーションには両方のイメージが含まれます。図 8 を参照してください。

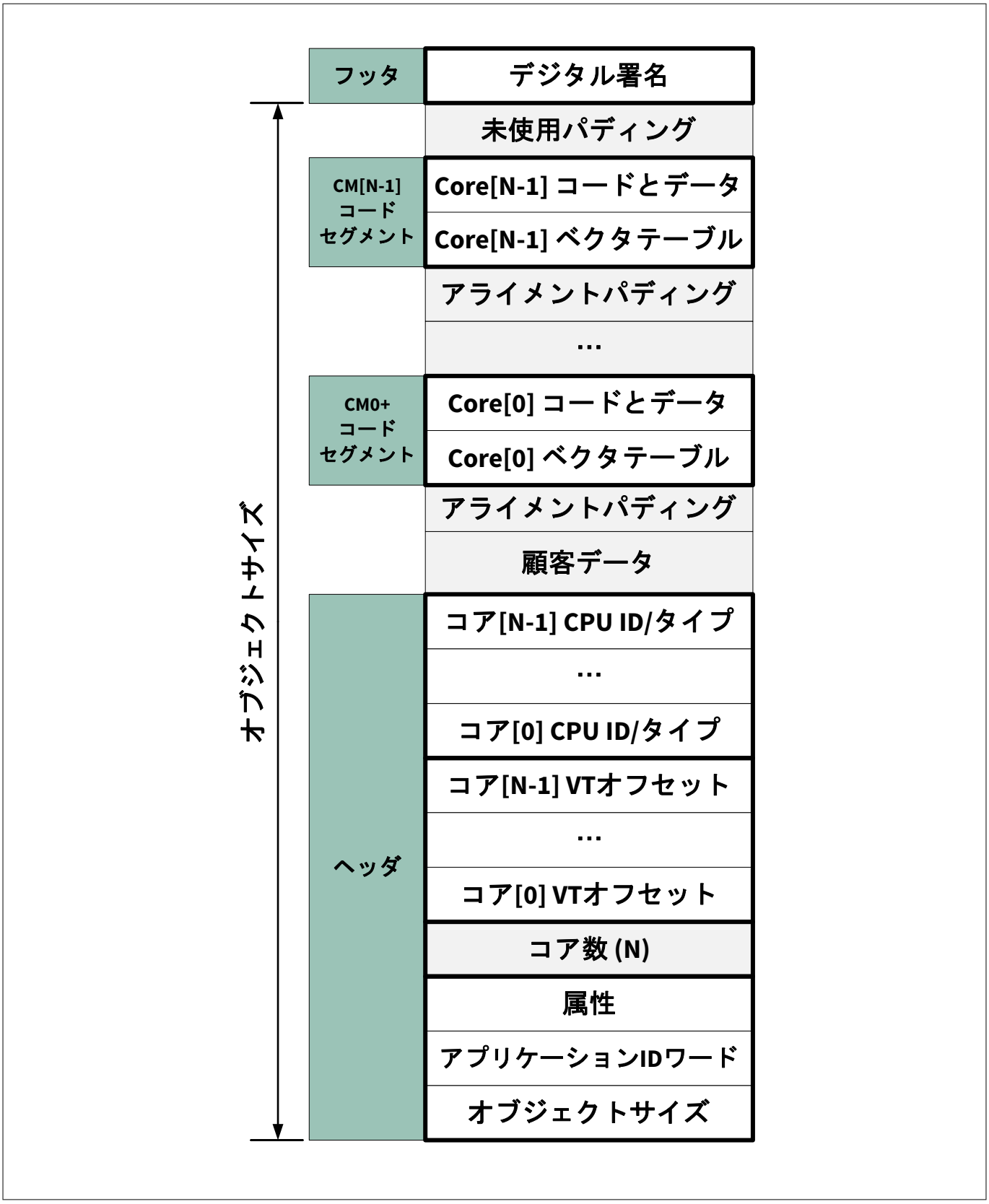


図 8 セキュアアプリケーションフォーマット

注: *ライフサイクルステージを SECURE または SECURE_W_DEBUG に移行する場合、CySAF が必要です。したがって、ライフサイクルステージを NORMAL_PROVISIONED から RMA に移行する場合においても、CySAF が必要です。ライフサイクルステージを NORMAL_PROVISIONED から RMA に移行するためには、Appendix E - RMA ライフサイクルステージへの移行を参照してください。*

表 7 に、CySAF のヘッダセクションの詳細を示します。合計サイズ, コア数, アプリケーションのタイプ, および各コアアプリケーションのベクタテーブルへのオフセットを定義します。

表 7 ヘッダ詳細

オフセット	サイズ	アイテム	説明
0x00	4 バイト	オブジェクトサイズ	フラッシュイメージサイズ (アプリケーションサイズ: バイト数)
0x04	4 バイト	アプリケーション ID/バージョン	この値は、フラッシュイメージのタイプを識別します。 Bit 31 - 28: 常に “0” Bit 27 - 24: メジャーバージョン Bit 23 - 16: マイナーバージョン Bit 15 - 0: アプリケーション ID 例 0x0000 – ユーザアプリケーション 0x8001 – フラッシュブート 0x8002 – セキュリティライブラリ 0x8003 – ブートローダ その他 – 予約済み
0x08	4 バイト	属性	将来的な機能拡張のため予約
0x0C	4 バイト	コア数(N)	アプリケーションによって使用されるコアの数
0x10 + (4*i)	4 バイト	コア(i) VT オフセット	このアドレスからコア(i) コードセグメントへのベクタテーブルへのオフセット
0x10+(4*N)+(4*i)	4 バイト	コア(i) CPU ID とコアインデックス	ユーザが割り当てた CPU ID とコアインデックス Bit 31 - 20: CPU ID. これは、Arm®デバイスの CPU ID[15:4]レジスタ値の一部です。 Bit 7 - 0: コアインデックス コアインデックスは、同タイプの複数コアを識別するために使用します。例えば、CM0+と 2 つの CM7_0/CM7_1 で構成されるシステムでは、CM0+は CPUID=0xC60 およびコアインデックス=0 で識別されます。1 つ目の CM7_0/CM7_1 は、CPUID=0xC24 およびコアインデックス=0 で識別されます。2 つ目の CM7_0/CM7_1 は、CPUID=0xC24 およびコアインデックス=1 で識別されます。

適切なアプリケーションヘッダを生成するために、以下のコードをプロジェクトに追加できます。

```
/** Secure Application header */
CY_SECTION(".cy_app_header") __USED static const cy_stc_si_appheader_t cy_si_appHeader =
{
    .objSize        = CY_M0PLUS_SI_SIZE,
    .appId          = (CY_SI_APP_VERSION | CY_SI_APP_ID_SECUREIMG),
    .appAttributes  = 0UL,                          /* Reserved */
    .numCores       = 1UL,                          /* Only CM0+ */
    .core0Vt        = CY_SI_VT_OFFSET,              /* CM0+ VT offset */
    .core0Id        = CY_SI_CPUID | CY_SI_CORE_IDX,   /* CM0+ core ID */
};

/** Secure Image Digital signature (Populated by cymcuelftool) */
CY_SECTION(".cy_app_signature") __USED CY_ALIGN(4)
static const uint8_t cy_si_appSignature[CY_SI_SECURE_DIGSIG_SIZE] = {0u};
```

8.3 セキュアブート RSA 公開鍵形式

暗号化アルゴリズムは、署名検証スキームの SHA-256 + RSASSA PKCS v1.5 アルゴリズムです。RSA 鍵の格納場所はユーザが定義できます。ユーザは TOC2 に鍵へのポインタを定義する必要があります。公開鍵の内容は、SFlash に配置すると ROM ブートの HASH 計算によって変更の有無がチェックされます。

SFlash 領域には、公開鍵がバイナリ形式で格納されます。Modulus, Exponent, および 3 つの Coefficient は、検証を高速化するために事前に計算されます。図 9 に形式を示します。

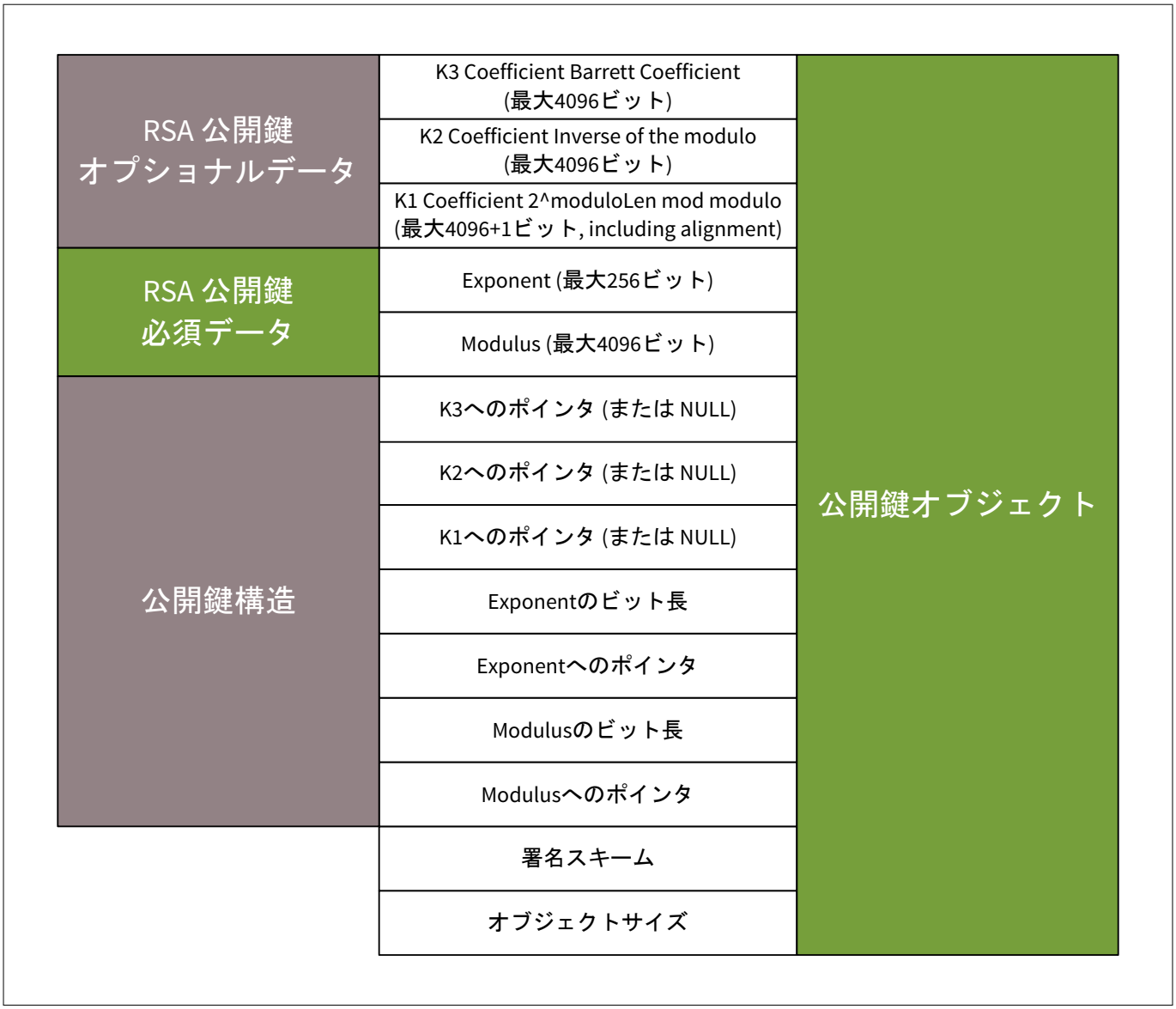


図 9　　公開鍵形式

鍵は 3 つの構造で格納します。最初の構造の「鍵」は、SECURE_HASH 計算に容易に含めることができるオブジェクトとして格納します。「署名スキーム」は鍵の構造を定義します。「オブジェクトサイズ」は、3 つの構造全体を含む公開鍵オブジェクトのフルサイズが含まれます。2 番目の構造は、公開鍵の個々の部分 (coefficient (K1, K2, K3), exponent (E), modulus (N)) が含まれます。これらは、リトルエンディアンのバイトリストに格納する必要があります。3 番目の構造は、公開鍵の各部分へのポインタリストです。これは、SROM ファームウェアの呼出しに必要な形式です。

適切な公開鍵形式を生成する場合、以下のコードをプロジェクトに追加してください。

8 セキュアシステムの構築

秘密鍵と公開鍵の生成と使用については、Appendix A - 公開鍵と秘密鍵の作成例を参照してください。

```
/** Public key in SFlash */
CY_SECTION(".cy_SFlash_public_key") __USED const cy_si_stc_public_key_t cy_publicKey =
{
    .objSize = sizeof(cy_si_stc_public_key_t),
    .signatureScheme = 0UL,
    .publicKeyStruct =
    {
        .moduloAddr         = CY_SI_PUBLIC_KEY + offsetof(cy_si_stc_public_key_t, moduloData),
        .moduloSize         = CY_SI_PUBLIC_KEY_SIZEOF_BYTE * CY_SI_PUBLIC_KEY_MODULOLENGTH,
        .expAddr            = CY_SI_PUBLIC_KEY + offsetof(cy_si_stc_public_key_t, expData),
        .expSize            = CY_SI_PUBLIC_KEY_SIZEOF_BYTE * CY_SI_PUBLIC_KEY_EXPLENGTH,
        .barrettAddr        = CY_SI_PUBLIC_KEY + offsetof(cy_si_stc_public_key_t, barrettData),
        .inverseModuloAddr  = CY_SI_PUBLIC_KEY + offsetof(cy_si_stc_public_key_t,
inverseModuloData),
        .rBarAddr           = CY_SI_PUBLIC_KEY + offsetof(cy_si_stc_public_key_t, rBarData),
    },
    .moduloData =
    { // N(Modulus)
    },
    .expData =
    { // E(Exponent)
    },
    .barrettData =
    { // K1
    },
    .inverseModuloData =
    { // K2
    },
    .rBarData =
    { // K3
    },
};
```

9 Appendix A - 公開鍵と秘密鍵の作成例

このプロジェクトには、rsa_private.txt と rsa_public.txt の 2 つファイルが用意されています。これらのファイルには、秘密鍵および公開鍵のサンプルが含まれます。これらはビルドシステムを適切に機能させるためのサンプルとしてのみ提供されます。実際の環境に移行する前に、独自のファイルに置換える必要があります。ここでは、公開鍵と秘密鍵のペアを生成し、それらを C 形式に変換し、ソースファイルを新しい鍵で更新する方法を説明します。

さらに、このアプリケーション ノートでは、サンプル ドライバ ライブラリ (SDL) を使用したサンプル コードについて説明します。このアプリケーション ノートのコードは SDL の一部です。SDL については関連ドキュメントを参照してください。セキュアシステム構築のセットアップについては、sdl_additional_code_examples の 16_AN228680_Secure_configuration フォルダにある Setup for secure configuration.pptx を参照してください。このサンプルコードは、GHS MULTI 環境向けの CYT2B7 シリーズ MCU に適用されます。

***注**: 本書に記載されているスクリプトの一部が正しく動作しない場合があるため、最新の GHS プローブ ファームウェアの使用を推奨します。このアプリケーション ノートでは、プローブ ファームウェア Ver.6.4.4 を使用しています。*

9.1 必要な追加のツール

1. OpenSSL v1.0.2 以降
2. Python 3 (公開鍵の形式の変換に使用されるスクリプトに必要です)
3. SDL 7.5 以降

RSA の秘密鍵と公開鍵を生成する方法はいくつかあります。次の方法では、コンピュータに OpenSSL/Python をインストールする必要があります。これら OpenSSL のソースまたはバイナリは、インターネット上のいくつかのソースからダウンロードできます。

9.2 スクリプト

このプロジェクトは、OpenSSL からの出力を C と互換のある形式に変換する 2 つのスクリプトとセキュアイメージに公開鍵を格納するために使用する構造を提供します。これらのスクリプトは、次のパスにあります。

<user>\tviibe1m\04_Util\Scripts\Key_2K, Key_3K or Key_4K

<user>: サンプルプロジェクトが格納されたフォルダ

バッチスクリプトの rsa_keygen.bat は OpenSSL 関数を使用します。そのため、OpenSSL がコンピュータにインストールされている必要があります。バッチファイルは、「keys_generated」ディレクトリを作成し、OpenSSL で生成された秘密鍵と公開鍵を含む 2 つのファイルを生成します。秘密鍵は ”rsa_private.txt”、公開鍵は「rsa_public.txt」に出力します。

次に、バッチファイルは Python スクリプト「rsa_to_c.py」を呼び出します。このスクリプトは、生成された公開鍵ファイルから、C および公開鍵形式と互換性のあるデータに変換します。出力は、公開鍵ファイルと秘密鍵ファイルとともに、keys_generated ディレクトリの rsa_to_c_generated.txt に出力します。

9 Appendix A - 公開鍵と秘密鍵の作成例

以下に RSA-2048 秘密鍵と公開鍵を生成する rsa_keygen.bat を示します。

```
set OUT_DIR="%~dp0\keys_generated"
set PRIV_NAME=rsa_private_generated.txt
set PUB_NAME=rsa_public_generated.txt
set PRIV_RNAME=rsa_private.txt
set PUB_RNAME=rsa_public.txt
set MOD_NAME=rsa_to_c_generated.txt

if not exist %OUT_DIR% mkdir %OUT_DIR%

:: Generate the RSA-2K public and private keys
openssl genrsa -out %OUT_DIR%\%PRIV_NAME% 2048
openssl rsa -in %OUT_DIR%\%PRIV_NAME% -outform PEM -pubout -out %OUT_DIR%\%PUB_NAME%

copy %OUT_DIR%\%PRIV_NAME% %OUT_DIR%\%PRIV_RNAME%
copy %OUT_DIR%\%PUB_NAME% %OUT_DIR%\%PUB_RNAME%

:: Create C-code ready public key
%~dp0\rsa_to_c.py %OUT_DIR%\%PUB_NAME% > %OUT_DIR%\%MOD_NAME%
```

RSA-3072 の秘密鍵と公開鍵を生成する場合は、2048 (赤) のテキストを 3072 に変更してください。RSA-4096 の場合は、テキストを 4096 に変更してください。

9.3 スクリプトの実行

バッチファイル rsa_keygen.bat はコマンドラインインタフェースから、または Windows7 または 10 環境でダブルクリック (a) して呼び出せます。実行後、以下のファイルが keys_generated フォルダ (b) に出力されたことを確認します。

- rsa_private.txt (RSA 秘密鍵)
- rsa_public.txt (RSA 公開鍵)
- rsa_to_c_generated.txt (C 形式の公開鍵)

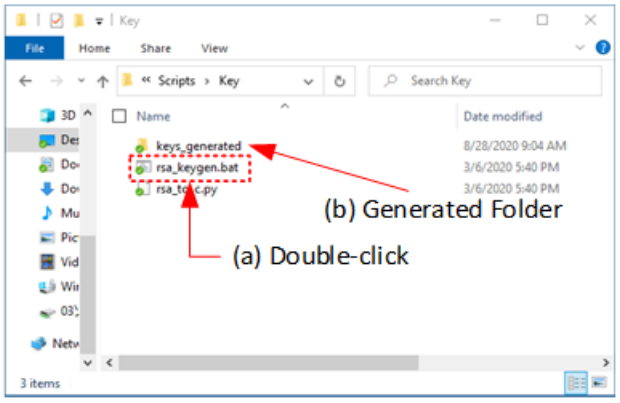


図 10 鍵の生成

9 Appendix A - 公開鍵と秘密鍵の作成例

秘密鍵 (RSA-2048) ファイルの例を以下に示します。

```
-----BEGIN RSA PRIVATE KEY-----
MIIEogIBAAKCAQEAs1LSkjDJbsnXvHdMbMYB4SMCN2QbMRwJmYytIVpc7kylFZN0
iK+uj1gB3UZzQshY6IjrAjCJy7l9xr8arzWu/waYntSshQmts4Bdd7UWefZtNjou
IRK6+N+geefXXEJFWzZd0/5Bdur+Z6gI8xUuc0BrgpBAEtVDf+/uBnPmPD3Eh7tu
60NbqkB22P9bwY0oyjKaGZUU0D0b8zCPwmp+RXHmLcK+D7q8CwotUkmYxpX/8PaV
ho6jwbCggI7FSWAF2m0KrtB1mkiAlKEavM+7mpsysYWApOghULrlqEy9svXAYDch
jaOmD1c7P2WGYxgmR/9acj1cMisemBCcMQmqOQIDAQABAoIBAG3OopjYfRB1UsDa
gJb+7Pmy2VGX4DrDQ3GePgn1boCRuGks4SguBANMzd90m3HigubpqJavqEFy9Xz1
Wt69TcpmfKCN9JKGqbIIQbPi1L+cUaXqHIMuGAX70iWz/SFGh3fiI7SDtgyeu2g3
8jr/1nDGms7ZjBK8Otgl0Khs4igyV8BXN+6xcoc5mzUW1T52vtsJFl1NpTN9IthD
Rm5WAb729mZNnWEixOHQ1BJZxJHFtUCZzK2O15rTMriWtgHo0teF/QHRpZzWxYhj
p4poIUHep7ZMQOAp5mVxWoIyUa1et3vVMfReVFBdDysi76lFvRsA7gTriJbAbEhX
hpuA3YECgYEA41EA6I98VTm07w6KyujmdIGwI4zLvRy25DSt8LaBNntUAFBJdBJB
/WGoPMfXMcj5d42wkpToO6FHqxqDZK+zNlYSCUAd7FeiRRlruRV7h1vFrAZt83el
UfuBLSmfDMUiagXE3I5qTm1Ccl/fk+v2xqz08BdNLHzEsRUgZWyii+kCgYEAyfOB
eYfayufO2K4MhTphEwBrcDaHM2Wbkqgcx+AT3ejFMA1ujHMDHSh0zaigodjl3ekY
FiYvXx1zi19wdpkOrjP+GMzhei70X8IcciszMB5ACmqWMDymGN58gP4616vbTkYa
/e4bZpta8E66xHbaf8iTFNGXrerdA+NAV5zwSdECgYBEbU3os8ipnwVoMOh2zEUT
LTbofKmX89zaNUFnBIlA7T8MGR8DYbubmpoaMc4Fnon6Axz0Av8ldBcu2nxTAtDf
umkTf8mCK58dTdmvjVfyFx4Z+pV+hLWvSBZKkgzeuzjw0WvBON2nXhxycYnL7WwL
wmS5IUqJ6cULyWBHJ7yGgQKBgHhosoLmiIJAUikJJTfuReDReD2QlW4EoAyCJZ7a
sJ234pIy//3HuUySSYoxh4zYSL3V8+GI5Oe+JJ0ty107BvDA2TiqN6nlAxlxIArG
TR/ceWx8fo3GK3ZaeTtj2Wur8Pcrf351kGrOKBttpZfsEXzs9x0LlndAuIRP45YZ
YX9xAoGARcsEcXuU9Mm7fprdKUjlxp1Tc/Ge+f3zYMCf9Arw+VnENUdUqMfeO4Z8
tjMd6eFC6/eaGpgOkhv85T22bi8xHNcQaM768nk4jb8sok/g+JG3k1+X0AulRTCY
Lbn8U9D4rYaRVvHeWKHzXxROlImqoJ7qe/T1MdltDJNR5Sy8Tm8=
-----END RSA PRIVATE KEY-----
```

公開鍵 (RSA-2048) ファイルの例を以下に示します。

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAs1LSkjDJbsnXvHdMbMYB
4SMCN2QbMRwJmYytIVpc7kylFZN0iK+uj1gB3UZzQshY6IjrAjCJy7l9xr8arzWu
/waYntSshQmts4Bdd7UWefZtNjouIRK6+N+geefXXEJFWzZd0/5Bdur+Z6gI8xUu
c0BrgpBAEtVDf+/uBnPmPD3Eh7tu60NbqkB22P9bwY0oyjKaGZUU0D0b8zCPwmp+
RXHmLcK+D7q8CwotUkmYxpX/8PaVho6jwbCggI7FSWAF2m0KrtB1mkiAlKEavM+7
mpsysYWApOghULrlqEy9svXAYDchjaOmD1c7P2WGYxgmR/9acj1cMisemBCcMQmq
OQIDAQAB
-----END PUBLIC KEY-----
```

9.4 公開鍵の実装

セキュアイメージの公開鍵を更新する最後の手順は、生成した `rsa_to_c_generated.txt` ファイルのコードを、セキュアイメージプロジェクトの一部である `main_cm0plus.c` ソースファイルにコピーすることです。更新後のファイ

9 Appendix A - 公開鍵と秘密鍵の作成例

ルの例を以下に示します。生成された鍵データによる置換コード部を緑色で示します。以下は RSA-2048 の例です。

```
/** Public key in SFlash */
CY_SECTION(".cy_SFlash_public_key") __USED const cy_si_stc_public_key_t cy_publicKey =
{
    .objSize = sizeof(cy_si_stc_public_key_t),
    .signatureScheme = 0UL,
    .publicKeyStruct =
    {
        .moduloAddr         = CY_SI_PUBLIC_KEY + offsetof(cy_si_stc_public_key_t, moduloData),
        .moduloSize         = CY_SI_PUBLIC_KEY_SIZEOF_BYTE * CY_SI_PUBLIC_KEY_MODULOLENGTH,
        .expAddr            = CY_SI_PUBLIC_KEY + offsetof(cy_si_stc_public_key_t, expData),
        .expSize            = CY_SI_PUBLIC_KEY_SIZEOF_BYTE * CY_SI_PUBLIC_KEY_EXPLENGTH,
        .barrettAddr        = CY_SI_PUBLIC_KEY + offsetof(cy_si_stc_public_key_t, barrettData),
        .inverseModuloAddr  = CY_SI_PUBLIC_KEY + offsetof(cy_si_stc_public_key_t,
inverseModuloData),
        .rBarAddr           = CY_SI_PUBLIC_KEY + offsetof(cy_si_stc_public_key_t, rBarData),
    },
    .moduloData =
    {
0x39u, 0xAAu, 0x09u, 0x31u, 0x9Cu, 0x10u, 0x98u, 0x1Eu,
0x2Bu, 0x32u, 0x5Cu, 0x3Du, 0x72u, 0x5Au, 0xFFu, 0x47u,
0x26u, 0x18u, 0x63u, 0x86u, 0x65u, 0x3Fu, 0x3Bu, 0x57u,
0x0Fu, 0xA6u, 0xA3u, 0x8Du, 0x21u, 0x37u, 0x60u, 0xC0u,
0xF5u, 0xB2u, 0xBDu, 0x4Cu, 0xA8u, 0xE5u, 0xBAu, 0x50u,
0x21u, 0xE8u, 0xA4u, 0x80u, 0x85u, 0xB1u, 0x32u, 0x9Bu,
0x9Au, 0xBBu, 0xCFu, 0xBCu, 0x1Au, 0xA1u, 0x94u, 0x80u,
0x48u, 0x9Au, 0x75u, 0xD0u, 0xAEu, 0x0Au, 0x6Du, 0xDAu,
0x05u, 0x60u, 0x49u, 0xC5u, 0x8Eu, 0x80u, 0xA0u, 0xB0u,
0xC1u, 0xA3u, 0x8Eu, 0x86u, 0x95u, 0xF6u, 0xF0u, 0xFFu,
0x95u, 0xC6u, 0x98u, 0x49u, 0x52u, 0x2Du, 0x0Au, 0x0Bu,
0xBCu, 0xBAu, 0x0Fu, 0xBEu, 0xC2u, 0x2Du, 0xE6u, 0x71u,
0x45u, 0x7Eu, 0x6Au, 0xC2u, 0x8Fu, 0x30u, 0xF3u, 0x1Bu,
0x3Du, 0xD0u, 0x14u, 0x95u, 0x19u, 0x9Au, 0x32u, 0xCAu,
0x28u, 0x8Du, 0xC1u, 0x5Bu, 0xFFu, 0xD8u, 0x76u, 0x40u,
0xAAu, 0x5Bu, 0x43u, 0xEBu, 0x6Eu, 0xBBu, 0x87u, 0xC4u,
0x3Du, 0x3Cu, 0xE6u, 0x73u, 0x06u, 0xEEu, 0xEFu, 0x7Fu,
0x43u, 0xD5u, 0x12u, 0x40u, 0x90u, 0x82u, 0x6Bu, 0x40u,
0x73u, 0x2Eu, 0x15u, 0xF3u, 0x08u, 0xA8u, 0x67u, 0xFEu,
0xEAu, 0x76u, 0x41u, 0xFEu, 0xD3u, 0x5Du, 0x36u, 0x5Bu,
0x45u, 0x42u, 0x5Cu, 0xD7u, 0xE7u, 0x79u, 0xA0u, 0xDFu,
0xF8u, 0xBAu, 0x12u, 0x21u, 0x2Eu, 0x3Au, 0x36u, 0x6Du,
0xF6u, 0x79u, 0x16u, 0xB5u, 0x77u, 0x5Du, 0x80u, 0xB3u,
0xADu, 0x09u, 0x85u, 0xACu, 0xD4u, 0x9Eu, 0x98u, 0x06u,
0xFFu, 0xAEu, 0x35u, 0xAFu, 0x1Au, 0xBFu, 0xC6u, 0x7Du,
0xB9u, 0xCBu, 0x89u, 0x30u, 0x02u, 0xEBu, 0x88u, 0xE8u,
0x58u, 0xC8u, 0x42u, 0x73u, 0x46u, 0xDDu, 0x01u, 0x58u,
0x8Fu, 0xAEu, 0xAFu, 0x88u, 0x74u, 0x93u, 0x15u, 0xA5u,
0x4Cu, 0xEEu, 0x5Cu, 0x5Au, 0x21u, 0xADu, 0x8Cu, 0x99u,
0x09u, 0x1Cu, 0x31u, 0x1Bu, 0x64u, 0x37u, 0x02u, 0x23u,
0xE1u, 0x01u, 0xC6u, 0x6Cu, 0x4Cu, 0x77u, 0xBCu, 0xD7u,
0xC9u, 0x6Eu, 0xC9u, 0x30u, 0x92u, 0xD2u, 0x52u, 0xB3u,
```

9 Appendix A - 公開鍵と秘密鍵の作成例

```
    },
    .expData =
    {
0x01u, 0x00u, 0x01u, 0x00u,
    },
    .barrettData =
    {
0xA2u, 0x78u, 0x5Cu, 0x68u, 0x49u, 0xBBu, 0x85u, 0xD6u,
0xF0u, 0x36u, 0xE3u, 0xAAu, 0xF7u, 0x33u, 0x48u, 0x40u,
0xC2u, 0xE2u, 0x75u, 0x03u, 0x7Eu, 0x18u, 0xCAu, 0x0Bu,
0x21u, 0xF3u, 0xDFu, 0x70u, 0xF4u, 0x73u, 0xBCu, 0x4Bu,
0xA2u, 0xFDu, 0x98u, 0x3Cu, 0x71u, 0x20u, 0xD3u, 0xECu,
0x57u, 0xC4u, 0xFEu, 0xE5u, 0xBBu, 0x38u, 0xEEu, 0x0Bu,
0x38u, 0x25u, 0xA5u, 0x0Au, 0xABu, 0xF5u, 0x88u, 0xE5u,
0x8Eu, 0x98u, 0xA7u, 0xA6u, 0x6Du, 0x2Fu, 0x12u, 0x40u,
0xC3u, 0x2Du, 0xD5u, 0x34u, 0x15u, 0x7Du, 0x6Au, 0x18u,
0xE8u, 0x64u, 0x3Au, 0x47u, 0x1Eu, 0xAFu, 0x0Cu, 0x8Eu,
0x75u, 0xE0u, 0x39u, 0x2Cu, 0x09u, 0x8Cu, 0xE0u, 0x96u,
0x6Du, 0xD4u, 0xB4u, 0x9Bu, 0x77u, 0xF0u, 0xA8u, 0xDAu,
0x7Cu, 0x60u, 0x09u, 0xF0u, 0x82u, 0xACu, 0x68u, 0x14u,
0x46u, 0xEEu, 0x1Du, 0xF7u, 0xCCu, 0x45u, 0xE8u, 0xCAu,
0x83u, 0x5Au, 0x19u, 0x74u, 0x1Bu, 0xEFu, 0xBAu, 0x98u,
0x4Bu, 0xC7u, 0x20u, 0x97u, 0x15u, 0xC8u, 0x8Bu, 0x17u,
0x09u, 0x06u, 0xB3u, 0x6Fu, 0x85u, 0x7Du, 0xC5u, 0x72u,
0xDCu, 0xD3u, 0x8Du, 0x14u, 0x12u, 0x8Bu, 0x6Cu, 0x81u,
0x33u, 0x6Fu, 0x57u, 0xF2u, 0x3Bu, 0x1Fu, 0x66u, 0x1Cu,
0xF9u, 0x3Au, 0xE3u, 0xE3u, 0x3Eu, 0x1Du, 0x86u, 0xDCu,
0xDCu, 0x85u, 0x29u, 0xD2u, 0x83u, 0x35u, 0x83u, 0x1Du,
0x44u, 0x51u, 0xD3u, 0x68u, 0x74u, 0x6Au, 0xBFu, 0xAEu,
0x3Eu, 0xCDu, 0x2Bu, 0xC6u, 0x7Fu, 0xDDu, 0xB5u, 0xB8u,
0x3Eu, 0x6Au, 0xEFu, 0x72u, 0x14u, 0xE9u, 0x56u, 0xBEu,
0xD0u, 0xD2u, 0xA0u, 0xA5u, 0x0Du, 0x68u, 0xA4u, 0x4Du,
0x76u, 0x7Au, 0x1Fu, 0xDFu, 0xD8u, 0x19u, 0x84u, 0x4Cu,
0x5Eu, 0xE4u, 0x5Fu, 0x1Au, 0xD7u, 0x7Bu, 0x79u, 0xCEu,
0xF9u, 0xFFu, 0x2Fu, 0x0Au, 0xFFu, 0xC5u, 0x3Au, 0xA8u,
0xFAu, 0x62u, 0xC5u, 0xDEu, 0x75u, 0xE7u, 0x22u, 0x01u,
0x4Du, 0x48u, 0x15u, 0x76u, 0x79u, 0x35u, 0x25u, 0x9Du,
0x33u, 0x0Fu, 0xFAu, 0xA5u, 0xE7u, 0x41u, 0xEDu, 0x06u,
0xD0u, 0x83u, 0x4Bu, 0xC4u, 0xA4u, 0x5Du, 0x76u, 0x6Du,
0x01u, 0x00u, 0x00u, 0x00u,
    },
    .inverseModuloData =
    {
      0xF7u, 0xDBu, 0x7Eu, 0xBBu, 0x40u, 0x73u, 0x6Eu, 0x72u,
      0xEFu, 0xA6u, 0x8Au, 0x7Fu, 0x8Au, 0x28u, 0x8Du, 0xB5u,
      0x35u, 0x2Fu, 0xD7u, 0x6Cu, 0x67u, 0x0Au, 0xBAu, 0xE3u,
      0x0Cu, 0xFEu, 0x8Fu, 0xDBu, 0x86u, 0xA7u, 0x3Cu, 0xC4u,
      0xACu, 0x26u, 0xF9u, 0x57u, 0x82u, 0xCAu, 0x66u, 0xC9u,
      0x76u, 0x9Fu, 0x3Bu, 0x36u, 0x38u, 0x14u, 0x72u, 0xF2u,
      0x28u, 0xFCu, 0xBDu, 0x2Eu, 0xFDu, 0x65u, 0x89u, 0x35u,
      0x78u, 0x7Du, 0x99u, 0x07u, 0x1Au, 0x53u, 0xC8u, 0x3Eu,
      0x51u, 0xD3u, 0xF2u, 0xFDu, 0xCEu, 0x92u, 0x8Fu, 0x10u,
      0xD2u, 0x27u, 0xC7u, 0xCCu, 0x0Fu, 0xF4u, 0xC9u, 0xAEu,
```

```
    0xCEu, 0x50u, 0x68u, 0x8Cu, 0x76u, 0xE9u, 0x91u, 0xD9u,
    0x42u, 0x55u, 0x1Fu, 0x25u, 0x04u, 0xB1u, 0xBDu, 0xABu,
    0xA1u, 0x16u, 0xBCu, 0xD7u, 0x2Cu, 0x8Bu, 0x55u, 0xC2u,
    0x02u, 0x96u, 0x04u, 0x44u, 0xB4u, 0x71u, 0x88u, 0xF9u,
    0x79u, 0xD0u, 0xF0u, 0x2Du, 0x58u, 0xF9u, 0x93u, 0xD5u,
    0x91u, 0x24u, 0xB8u, 0x2Bu, 0xA9u, 0x3Eu, 0x6Au, 0xE3u,
    0x07u, 0x44u, 0xDCu, 0xD5u, 0x8Du, 0xB1u, 0xA3u, 0xC3u,
    0x09u, 0x57u, 0xC5u, 0x9Au, 0xAEu, 0x93u, 0x0Cu, 0xEEu,
    0x29u, 0xEAu, 0x03u, 0x41u, 0xD0u, 0xE6u, 0xA1u, 0xFFu,
    0x65u, 0x02u, 0x17u, 0x7Eu, 0x31u, 0x3Cu, 0x00u, 0x4Cu,
    0xA9u, 0x32u, 0xF3u, 0xC6u, 0x8Du, 0xA9u, 0x33u, 0xDBu,
    0x62u, 0x23u, 0x4Eu, 0xE3u, 0x1Au, 0xEAu, 0x97u, 0x60u,
    0xA8u, 0x34u, 0xE3u, 0x3Bu, 0x96u, 0xBCu, 0xE5u, 0x2Fu,
    0xC2u, 0x66u, 0x40u, 0xE6u, 0xFFu, 0x92u, 0x84u, 0xF6u,
    0x38u, 0xB7u, 0x59u, 0x81u, 0x96u, 0xEFu, 0x1Fu, 0xD9u,
    0xA9u, 0x20u, 0x8Bu, 0xB2u, 0x77u, 0x49u, 0x0Fu, 0xA9u,
    0x0Fu, 0x7Fu, 0x60u, 0xD4u, 0x6Bu, 0xBAu, 0xC6u, 0x73u,
    0xA2u, 0x25u, 0x44u, 0xA2u, 0xEAu, 0x91u, 0xDBu, 0xA3u,
    0xC2u, 0x8Cu, 0x27u, 0x38u, 0xFCu, 0xEAu, 0xFEu, 0x00u,
    0x6Du, 0x93u, 0xB6u, 0x0Du, 0xF8u, 0x74u, 0xFDu, 0x14u,
    0xC7u, 0xD5u, 0xE7u, 0x7Du, 0x32u, 0x08u, 0x52u, 0x8Du,
    0xACu, 0x66u, 0x03u, 0x4Fu, 0xA9u, 0x33u, 0xA0u, 0x7Bu,
  },
  .rBarData =
  {
    0xC7u, 0x55u, 0xF6u, 0xCEu, 0x63u, 0xEFu, 0x67u, 0xE1u,
    0xD4u, 0xCDu, 0xA3u, 0xC2u, 0x8Du, 0xA5u, 0x00u, 0xB8u,
    0xD9u, 0xE7u, 0x9Cu, 0x79u, 0x9Au, 0xC0u, 0xC4u, 0xA8u,
    0xF0u, 0x59u, 0x5Cu, 0x72u, 0xDEu, 0xC8u, 0x9Fu, 0x3Fu,
    0x0Au, 0x4Du, 0x42u, 0xB3u, 0x57u, 0x1Au, 0x45u, 0xAFu,
    0xDEu, 0x17u, 0x5Bu, 0x7Fu, 0x7Au, 0x4Eu, 0xCDu, 0x64u,
    0x65u, 0x44u, 0x30u, 0x43u, 0xE5u, 0x5Eu, 0x6Bu, 0x7Fu,
    0xB7u, 0x65u, 0x8Au, 0x2Fu, 0x51u, 0xF5u, 0x92u, 0x25u,
    0xFAu, 0x9Fu, 0xB6u, 0x3Au, 0x71u, 0x7Fu, 0x5Fu, 0x4Fu,
    0x3Eu, 0x5Cu, 0x71u, 0x79u, 0x6Au, 0x09u, 0x0Fu, 0x00u,
    0x6Au, 0x39u, 0x67u, 0xB6u, 0xADu, 0xD2u, 0xF5u, 0xF4u,
    0x43u, 0x45u, 0xF0u, 0x41u, 0x3Du, 0xD2u, 0x19u, 0x8Eu,
    0xBAu, 0x81u, 0x95u, 0x3Du, 0x70u, 0xCFu, 0x0Cu, 0xE4u,
    0xC2u, 0x2Fu, 0xEBu, 0x6Au, 0xE6u, 0x65u, 0xCDu, 0x35u,
    0xD7u, 0x72u, 0x3Eu, 0xA4u, 0x00u, 0x27u, 0x89u, 0xBFu,
    0x55u, 0xA4u, 0xBCu, 0x14u, 0x91u, 0x44u, 0x78u, 0x3Bu,
    0xC2u, 0xC3u, 0x19u, 0x8Cu, 0xF9u, 0x11u, 0x10u, 0x80u,
    0xBCu, 0x2Au, 0xEDu, 0xBFu, 0x6Fu, 0x7Du, 0x94u, 0xBFu,
    0x8Cu, 0xD1u, 0xEAu, 0x0Cu, 0xF7u, 0x57u, 0x98u, 0x01u,
    0x15u, 0x89u, 0xBEu, 0x01u, 0x2Cu, 0xA2u, 0xC9u, 0xA4u,
    0xBAu, 0xBDu, 0xA3u, 0x28u, 0x18u, 0x86u, 0x5Fu, 0x20u,
    0x07u, 0x45u, 0xEDu, 0xDEu, 0xD1u, 0xC5u, 0xC9u, 0x92u,
    0x09u, 0x86u, 0xE9u, 0x4Au, 0x88u, 0xA2u, 0x7Fu, 0x4Cu,
    0x52u, 0xF6u, 0x7Au, 0x53u, 0x2Bu, 0x61u, 0x67u, 0xF9u,
    0x00u, 0x51u, 0xCAu, 0x50u, 0xE5u, 0x40u, 0x39u, 0x82u,
    0x46u, 0x34u, 0x76u, 0xCFu, 0xFDu, 0x14u, 0x77u, 0x17u,
    0xA7u, 0x37u, 0xBDu, 0x8Cu, 0xB9u, 0x22u, 0xFEu, 0xA7u,
    0x70u, 0x51u, 0x50u, 0x77u, 0x8Bu, 0x6Cu, 0xEAu, 0x5Au,
```

```
        0xB3u, 0x11u, 0xA3u, 0xA5u, 0xDEu, 0x52u, 0x73u, 0x66u,
        0xF6u, 0xE3u, 0xCEu, 0xE4u, 0x9Bu, 0xC8u, 0xFDu, 0xDCu,
        0x1Eu, 0xFEu, 0x39u, 0x93u, 0xB3u, 0x88u, 0x43u, 0x28u,
        0x36u, 0x91u, 0x36u, 0xCFu, 0x6Du, 0x2Du, 0xADu, 0x4Cu,
    },
};
```

Cy_FB_Isvalidkey は、公開鍵構造が有効かどうかを確認します。詳細については、アーキテクチャ TRM[2]の Flash Boot の章を参照してください。

RSA 鍵サイズに応じて次のファイルで公開鍵形式の Modulus ビット長を変更する必要があります。

<user>\ tviibe1m\src\cy_si_keystorage.h

<user>: サンプルプロジェクトが格納されたフォルダ

```
#define CY_SI_PUBLIC_KEY_MODULOLENGTH   (256UL) /**< Modulus length of the RSA 2K key */
//#define CY_SI_PUBLIC_KEY_MODULOLENGTH   (384UL) /**< Modulus length of the RSA 3K key */
//#define CY_SI_PUBLIC_KEY_MODULOLENGTH   (512UL) /**< Modulus length of the RSA 4K key */
```

10 Appendix B - セキュアイメージの作成

このプロジェクトは、セキュアなコードに署名する Secure_Complete_SREC_to_CYP.bat スクリプトを提供します。これらのスクリプトは、次のパスにあります。

<user>\tviibe1m\04_Util\Scripts

<user>: サンプルプロジェクトが格納されたフォルダ

このスクリプトは、デジタル署名を CM0+ ELF ファイルに追加し、CYP_Scripts フォルダに CM0+ SREC ファイルを生成します。また、CM4 ELF/SREC ファイルを CYP_Scripts フォルダにコピーします。

このスクリプトは、cymcuelftool.exe ツールを呼び出します。このツールは、デジタル署名を生成し CM0+ ELF ファイルに追加します。このツールは、<user>\ tviibe1m\04_Util\Tools に格納されます。

このスクリプトは、秘密鍵を使用してデジタル署名を生成し、デジタル署名をセキュアコードに追加します。生成されたデジタル署名は、SFlash に保存されている公開鍵を使用して復号化され、フラッシュブートによって検証されます。

以下は、cymcuelftool.exe ツールの使用例です。

```
> cymcuelftool.exe -sign [in.elf] [HASH ALGORITHM] -encrypt [ENC_ALGORITHM] -key [PrivateKey] -
output [out.elf]
```

また、in.elf に CM4 ELF ファイルを選択して、デジタル署名を CM4 ELF ファイルにも追加できます。

10.1 ビルドとプログラミング

1. GHS Multi を使用して Flash プロジェクトをビルドします。

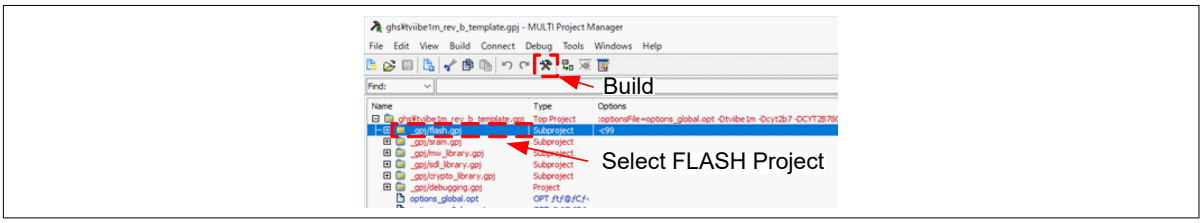


図 11 フラッシュプロジェクトのビルド

2. Secure_Complete_SREC_to_CYP_2K，Key_3K，または Key_4K.bat スクリプトを実行します。デジタル署名が追加された CM0+ SREC ファイル (cm0plus_si.elf.srec) および CM4 SREC ファイル(cm4_si.elf.srec) は、CYP_Scripts フォルダに保存されます。

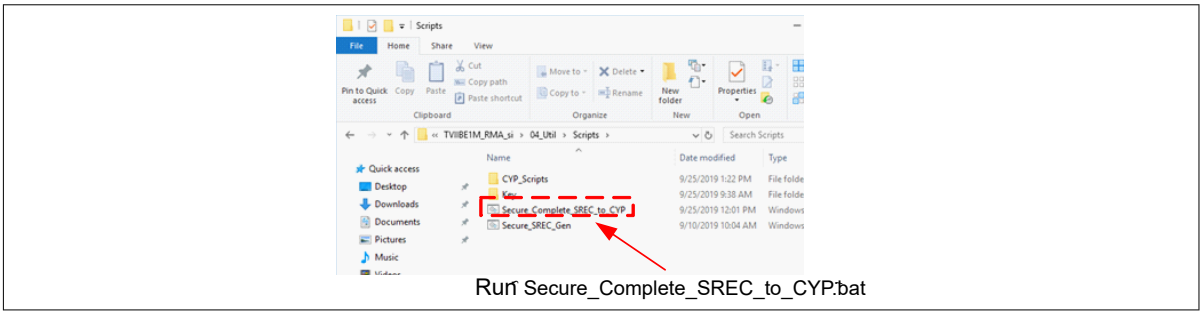


図 12 Secure_Complete_SREC_to_CYP.bat の実行

3. MiniProg4 をターゲットボードに接続します。
4. enable_sflashandprogram.bat を実行します。CM0+ SREC ファイルと CM4 SREC ファイルが MCU デバイスにプログラムされます。

10 Appendix B - セキュアイメージの作成

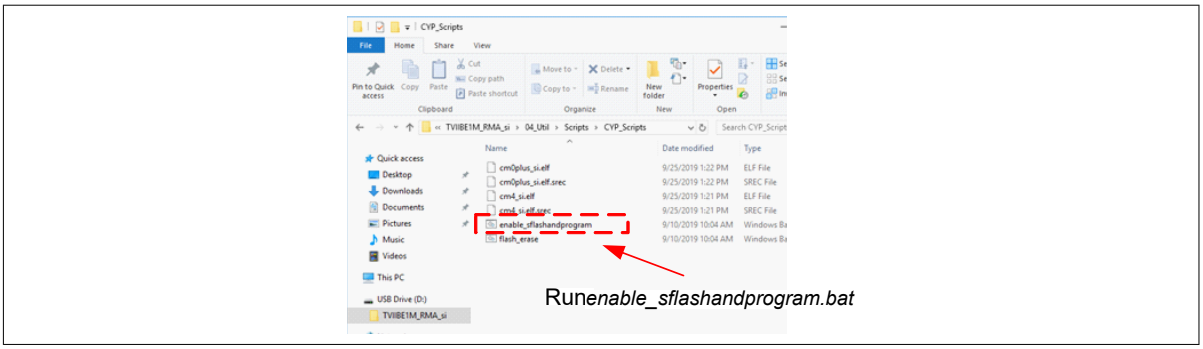


図 13　　enable_sflashandprogram.bat の実行

11 Appendix C - デジタル署名生成の要件

cymcuelftool.exe は、デジタル署名の生成に以下のシンボル/セクションが必要です。

1. “.cy_app_signature” セクション:
   このセクションは、デジタル署名が書き込まれる場所を指定します。

```
/** Secure Image Digital signature (Populated by cymcuelftool) */
                CY_SECTION(".cy_app_signature") __USED CY_ALIGN(4)
                static const uint8_t cy_si_appSignature[CY_SI_SECURE_DIGSIG_SIZE] = {0u};
```

2. 「__cy_app_verify_start」および「__cy_app_verify_length」シンボル:
   このシンボルは、デジタル署名が計算されるメモリ領域の開始アドレスとサイズを定義します。

```
__cy_app_verify_start = 0x10000000;
                __cy_app_verify_length = 0xFE00;
```

この例では、デジタル署名の計算領域は、0x10000000～0x1000FE00 です。

このスクリプトによって生成されるデジタル署名は、計算領域中のブランクは「0」を想定していることに注意してください。「0」埋めする SREC ファイルの生成については、Appendix J – ダミーアプリケーションヘッダの実装を参照してください。

12 Appendix D - メインユーザアプリケーションの認証

メインユーザアプリケーションは、セキュアイメージにより認証します。CoT では、メイン CPU (CM4 または CM7) のアクティブ前に、メインユーザアプリケーションを認証する必要があります。

TRAVEO™ T2G MCU は、ユーザアプリケーションを認証するための Cy_FB_VerifyApplication 関数をサポートします。これはフラッシュブートに含まれ、ユーザコードから実行でき、RSASSA-PKCS1-v1.5 を使用して他のコード認証に使用できます。詳細については、アーキテクチャ TRM[2]の Flash Boot の章を参照してください。

この例では、ファーストユーザアプリケーションと同じ方法を使用しデジタル署名が生成され、SFlash にある公開鍵を使用して認証が実行されます。

```
typedef bool (*pFB_VerifyApp)(uint32_t, uint32_t, uint32_t, uint32_t);

int main(void)
{
    bool isAppValid;
    uint32_t publicKeyAddr;
    uint32_t appDataAddr;
    uint32_t appDataLength;
    uint32_t appSignatureAddr;
    uint32_t toc2Addr;

    pFB_VerifyApp VerifyApplication = (pFB_VerifyApp)*(uint32_t*)(0x17002040);

    SystemInit();

    /* Enable global interrupts. */
    __enable_irq();

    /* Run RSA */
    appDataAddr       = CY_SI_SECURE_FLASH_BEGIN_CM4;
    appDataLength     = CY_M4_SI_SIZE;
    appSignatureAddr  = CY_SI_SIGNATURE_ADDR;
    publicKeyAddr     = CY_SI_PUBLIC_KEY + CY_FB_PBKEY_STRUCT_OFFSET;

    isAppValid = VerifyApplication (appDataAddr, appDataLength, appSignatureAddr,
publicKeyAddr);

    /* RSA Verify Error Handling Here */
    while(isAppValid == false);

    /* Enable CM4.  */
    Cy_SysEnableApplCore(CY_CORTEX_M4_APPL_ADDR);

    while(1)
    {
    }
}
```

表 8 に、この関数に必要なパラメータを示します

12 Appendix D - メインユーザアプリケーションの認証

表 8 Cy_FB_VerifyApplication 関数パラメータ

パラメータ	説明
appDataAddr	認証するデータイメージの開始アドレス
appDataLength	データイメージの長さ
appSignatureAddr	データイメージの署名開始アドレス
publicKeyAddr	公開鍵のアドレス

13 Appendix E - RMA ライフサイクルステージへの移行

部品の故障解析をする場合、顧客は、部品を RMA ライフサイクルステージに移行します。

RMA ライフサイクルステージに移行するためには、デバイスのユニーク ID と SFlash に格納されている公開鍵とペアの秘密鍵が必要です。UART, SPI, I2C, CAN LIN などを介してデバイスの外部から送信できる 2 つの専用のコマンドを実装する必要があります。1 つは、内部のユニーク ID の読み出し、もう 1 つは、RMA ライフサイクルへの移行させることです。Sys-DAP が使用可能な場合、これら 2 つの特別なコマンドは、Sys-DAP を介して実行できます。

顧客は、固有のユニーク ID を持つ部品を RMA ライフサイクルステージに移行するために証明書を作成する必要があります。部品が RMA に移行後、故障解析のために顧客は秘密鍵を使用して署名した別の証明書を提供する必要があります。

***注**: RMA ライフサイクルステージに移行する場合、ROM ブートのデフォルトクロック構成を使用することを推奨します。詳細についてはアーキテクチャ TRM[2] の Blow Fuse Bit を参照してください。*

***注**: TransitiontoRMA と OpenRMA API を実行する場合、部品ごとに異なる ID(ユニーク ID)が必要なため、部品ごとに異なる証明書が必要です。したがって、故障解析のため複数の部品を返却する場合、マーキングなどにより部品と証明書をリンクすることを推奨します。*

デバイスを RMA ステージに移行するためには、以下の手順を実行します。

1. デバイスが NORMAL_PROVISIONED ライフサイクルステージにある場合:
   - デバイスに公開鍵が格納されていない場合、公開鍵を SFlash に書込みます。
   - デバイスのユーザコードが CySAF 形式ではない場合、SECURE または SECURE_W_DEBUG ライフサイクルステージに移行する前に、CySAF 形式で再プログラムするか、ダミーの CySAF 形式をプログラムする必要があります。詳細については、ユーザアプリケーションブロックを参照してください。
   - デバイスを SECURE または SECURE_W_DEBUG ライフサイクルステージに移行します。

     デバイスがすでに SECURE または SECURE_W_DEBUG ライフサイクルステージにある場合、このプロセス(1)は不要です。
2. デバイスに保存されている機密または専用コードをすべて消去します。これは、特別なコマンドまたは前述の特別なコードイメージを使用して実行できます。ただし、保護状態が DEAD のデバイスは、RMA ライフサイクルステージに移行できないことに注意してください。機密性の高いコードや独自のコードを消去した場合、DEAD 保護状態に移行する場合があります。SFlash に保存されている公開鍵は、RMA ライフサイクルステージへの移行、および RMA 実行に使用するため消去できません。

   ***注**: セキュアイメージまたは認証用のデジタル署名を消去することで、デバイスが DEAD 状態になる場合があります。その結果、RMA ライフサイクルステージに移行できません。コードを消去する必要がある場合、署名済みのダミーコードとデジタル署名を準備するか、事前にセカンドアプリケーションを実装する必要があります。詳細については、RMA を参照してください。*
3. SFlash に保存されるデバイスのユニーク ID を読み出します。これは、システムコール 0x1F (ReadUniqueID) を呼び出し、通信インタフェースまたは Sys-DAP を介して ID を送信することで実行できます。
4. ユニーク ID と、SFlash 内に格納される公開鍵とペアの顧客の秘密鍵を使用し証明書を生成します。表 9 に、証明書の形式を示します。
5. 証明書を含むコマンドをデバイスに送信します。ユーザは、RMA システムコール (0x28) への移行を呼び出し、この証明書を受入れるコードを実装し、証明書をパラメータとして渡す必要があります。

注: TransitiontoRMA API 実行条件については、RMA ライフサイクルステージ移行の要件を参照してください。

6. デバイスはリセットまたは電源の再投入後、デバッグポートからの RMA 開始のための単一のコマンドを待ちます(システムコール 0x29)。デバイスは OpenRMA API が正常に実行されるまで OpenRMA API のみ許可されます。

   注: OpenRMA API 実行条件については、RMA ライフサイクルステージ移行の要件を参照してください。

表 9 RMA 証明書の形式

オブジェクト	バイト数
オブジェクトサイズ	4 バイト
コマンド ID	4 バイト TransitiontoRMA: 0x120028F0 OpenRMA: 0x120029F0
ユニーク ID	11 バイト
“0”パディング	1 バイト
デジタル署名	RSA-2048 で最大 256 バイト RSA-3072 で最大 384 バイト RSA-4096 で最大 512 バイト

注: TransitiontoRMA API および OpenRMA API の証明書とデジタル署名は異なります。

13.1 証明書の生成

ここでは、Open SSL を使用して Linux 上で証明書を作成する方法について説明します。

1. `ReadUniqueID` API (システムコール 0x1F) を使用して、SFlash のユニーク ID を読み出します。以下は、readuniqueID の結果の例です。

   0xa00dfe10: 最初のバイトはシステムコールステータスです (0xa0:成功/0xf0:失敗)その他: ユニーク ID_0

   0x000a0a03: ユニーク ID_1

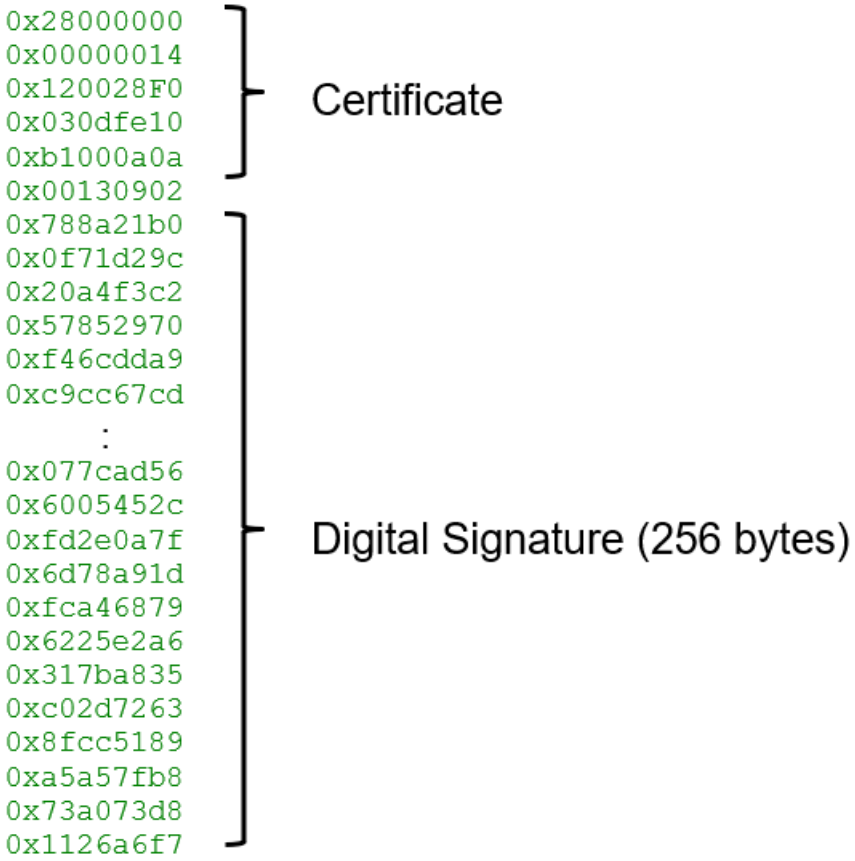
   0x130902b1: ユニーク ID_2

2. Linux 上でコマンドを入力します。以下は、RMA 移行の証明書とデジタル署名の生成例です。

```
echo 14000000   F0280012       10fe0d03 0a0a00b1 02091300 | xxd -r -p > _data.bin
Object Size     Command ID     Unique ID               Zero PAdding   /
openssl dgst -sha256 -sign rsa_private.txt _data.bin > _signature.bin
cat _signature.bin | xxd -p -c 64 > _tmp1.hex
echo 00000028 > _tmp2.hex
echo 14000000 F0280012 10fe0d03 0a0a00b1 02091300 >> _tmp2.hex
cat _tmp1.hex >> _tmp2.hex
sed -r "s/\s*(\w{2})(\w{2})(\w{2})(\w{2})/0x\4\3\2\1\n/g" _tmp2.hex | sed -r
"/^$/d" > _output_transtorma.hex
rm _data.bin _tmp1.hex _tmp2.hex
```

出力される`_output_transtorma.hex` ファイルを以下に示します。

13 Appendix E - RMA ライフサイクルステージへの移行



14 Appendix F - アプリケーション保護設定

このプロジェクトは、アプリケーション保護の設定手段を提供します。ここでは、顧客システムのアプリケーション保護である SWPU を設定および追加する方法について説明します。SWPU は、FWPU, ERPU, および EWPU で構成されます。アプリケーション保護は、FWPU は最大 16 エントリ、ERPU および EWPU は最大 4 エントリで設定できます。

各 SWPU はマスタ/スレーブ構造を持ち、保護構造によって保護構造を保護します。スレーブ属性はリソースへのアクセス属性を示し、マスタ属性はスレーブへのアクセス属性を示します。したがって、SWPU スレーブ属性の変更には、マスタ定義の属性が必要です。

アプリケーション保護は、次の要素で構成されます。詳細については、アーキテクチャ TRM[2]の Protection Unit の章を参照してください。

- PU_OBJECT_SIZE: 設定する要素数。(4 バイト)
- N_FWPU: FWPU オブジェクト数。FWPU は最大 16 領域設定できます。(4 バイト)
- FWPUx_SL_ADDR: FWPUx ベースアドレスを設定。(4 バイト)
- FWPUx_SL_SIZE: FWPUx 領域サイズおよび FWPUx イネーブルを設定。(4 バイト)
- FWPUx_SL_ATT: FWPUx スレーブ属性を設定。(4 バイト)
- FWPUx_MS_ATT: FWPUx マスタ属性を設定。(4 バイト)
- N_ERPU: ERPU オブジェクト数。ERPU は最大 4 領域設定できます。(4 バイト)
- ERPUy_SL_OFFSET: ERPUy ベースアドレスオフセットを設定。(4 バイト)
- ERPUy_SL_SIZE: ERPUy 領域サイズおよび ERPUy イネーブルを設定。(4 バイト)
- ERPUy_SL_ATT: ERPUy スレーブ属性を設定。(4 バイト)
- ERPUy_MS_ATT: ERPUy マスタ属性を設定。(4 バイト)
- N_EWPU: EWPU オブジェクト数。EWPU は最大 4 領域設定できます。(4 バイト)
- EWPUy_SL_OFFSET: EWPUy ベースアドレスオフセットを設定。(4 バイト)
- EWPUy_SL_SIZE: EWPUy 領域サイズおよび ERPUy イネーブルを設定。(4 バイト)
- EWPUy_SL_ATT: EWPUy スレーブ属性を設定。(4 バイト)
- EWPUy_MS_ATT: EWPUy マスタ属性を設定。(4 バイト)

サフィックス「x」は 0～15 を示し、サフィックス「y」は 0～3 を示します。

14.1 コンフィグレーション

以下に、アプリケーション保護を設定する手順を示します。このセクションアドレスは、TOC2 の「アプリケーション保護のアドレス (オフセット= 0x108)」と一致する必要があります。セキュリティため、デフォルト値 (0x17007600) を変更しないでください。

1. 各 SWPU の領域数を設定します。

```
// deification of application protection
#define N_FWPU                (0UL)  /**< Number of flash write protection Max 16 */
#define N_ERPU                (1UL)  /**< Number of efuse read  protection Max  4 */
#define N_EWPU                (1UL)  /**< Number of efuse write protection Max  4 */
```

2. SWPU 数に応じて、各 SWPU を設定します。

```
   /*************************************
   *   Application Protection
   *************************************/

   CY_SECTION(".cy_SFlash_app_prot") __USED static const cy_stc_si_app_prot_t
cy_si_appprot =
   {
    .objSize                = OBJECT_SIZE,            /* Application Protection Object
Size (in bytes) */
    .n_fwpu                 = N_FWPU,                 /* Number of FWPU Max 16 */
    .fwpu0_adr.addr30       = 0x10000000>>2ul,        /* Add region if you need */
    .fwpu0_size.region_size = 0x200,                  /* in bytes (multiple of 4) */
    .fwpu0_size.enable      = APP_PROT_ENABLE,        /* FWPU0 enable */
    .fwpu0_sl_att.urw       = APP_PROT_ALLOW,         /* FWPU0 Slave Attribute */
    .fwpu0_sl_att.prw       = APP_PROT_ALLOW,         /* FWPU0 Slave Attribute */
    .fwpu0_sl_att.ns        = APP_PROT_ALLOW,         /* FWPU0 Slave Attribute */
    .fwpu0_sl_att.pc_mask   = 0x00FF,                 /* FWPU0 Slave Attribute */
    .fwpu0_ms_att.urw       = APP_PROT_ALLOW,         /* FWPU0 Master Attribute */
    .fwpu0_ms_att.prw       = APP_PROT_ALLOW,         /* FWPU0 Master Attribute */
    .fwpu0_ms_att.ns        = APP_PROT_ALLOW,         /* FWPU0 Master Attribute */
    .fwpu0_ms_att.pc_mask   = 0x00FF,                 /* FWPU0 Master Attribute */
    .n_erpu                 = N_ERPU,                 /* Number of ERPU Max 4 */
    .erpu0_offset.offset    = 0x68,                   /* ERPU0 offset (Default) */
    .erpu0_size.region_size = 0x18,                   /* ERPU0 region size (Default) */
    .erpu0_size.enable      = APP_PROT_ENABLE,        /* ERPU0 enable (Default) */
    .erpu0_sl_att.urw       = APP_PROT_ALLOW,         /* ERPU0 Slave Attribute (Default)
*/
    .erpu0_sl_att.prw       = APP_PROT_ALLOW,         /* ERPU0 Slave Attribute (Default)
*/
    .erpu0_sl_att.ns        = APP_PROT_ALLOW,         /* ERPU0 Slave Attribute (Default)
*/
    .erpu0_sl_att.pc_mask   = 0x00FF,                 /* ERPU0 Slave Attribute (Default)
*/
    .erpu0_ms_att.urw       = APP_PROT_ALLOW,         /* ERPU0 Master Attribute
(Default) */
    .erpu0_ms_att.prw       = APP_PROT_ALLOW,         /* ERPU0 Master Attribute
(Default) */
    .erpu0_ms_att.ns        = APP_PROT_ALLOW,         /* ERPU0 Master Attribute
(Default) */
    .erpu0_ms_att.pc_mask   = 0x00FF,                 /* ERPU0 Master Attribute
(Default) */
    .n_ewpu                 = N_EWPU,                 /* Number of EWPU Max 4 */
    .ewpu0_offset.offset    = 0x68,                   /* EWPU0 offset (Default) */
```

```
    .ewpu0_size.region_size = 0x18,                 /* EWPU0 region size (Default) */
    .ewpu0_size.enable      = APP_PROT_ENABLE,      /* EWPU0 enable (Default) */
    .ewpu0_sl_att.urw       = APP_PROT_ALLOW,       /* EWPU0 Slave Attribute (Default)
*/
    .ewpu0_sl_att.prw       = APP_PROT_ALLOW,       /* EWPU0 Slave Attribute (Default)
*/
    .ewpu0_sl_att.ns        = APP_PROT_ALLOW,       /* EWPU0 Slave Attribute (Default)
*/
    .ewpu0_sl_att.pc_mask   = 0x00FF,               /* EWPU0 Slave Attribute (Default)
*/
    .ewpu0_ms_att.urw       = APP_PROT_ALLOW,       /* EWPU0 Master Attribute
(Default) */
    .ewpu0_ms_att.prw       = APP_PROT_ALLOW,       /* EWPU0 Master Attribute
(Default) */
    .ewpu0_ms_att.ns        = APP_PROT_ALLOW,       /* EWPU0 Master Attribute
(Default) */
    .ewpu0_ms_att.pc_mask   = 0x00FF,               /* EWPU0 Master Attribute
(Default) */
  };
```

表 10 に、各要素の詳細を示します。

表 10　　要素の説明

要素	説明
.objeSize	設定する要素数
.n_fwpu	FWPU オブジェクト数
.n_erpu	ERPU オブジェクト数
.n_ewpu	EWPU オブジェクト数
.fwpu0_adr.addr30	FWPU0 ベースアドレス (4 バイトアライン).
.fwpu0_size.region_size	FWPU0 領域サイズ
.fwpu0_size.enable	FWPU0 イネーブル 0: ディセーブル 1: イネーブル
.fwpu0/erpu0/ewpu0_sl_att.urw	FWPU0/EWPU0 スレーブ属性 ユーザライト 0: 禁止 1: 許可 EWPU0 スレーブ属性 ユーザリード 0: 禁止 1: 許可

(続く)

14 Appendix F - アプリケーション保護設定

表 10 (続き) 要素の説明

要素	説明
.fwpu0/erpu0/ewpu0_sl_att.prw	FWPU0/EWPU0 スレーブ属性 特権ライト. 0: 禁止 1: 許可 EWPU0 スレーブ属性 特権リード 0: 禁止 1: 許可
.fwpu0/erpu0/ewpu0_sl_att.ns	FWPU0/EWPU0 スレーブ属性 非セキュアライト 0: 禁止 1: 許可 EWPU0 スレーブ属性 非セキュアリード 0: 禁止 1: 許可
.fwpu0/erpu0/ewpu0_sl_att.pc_mask	FWPU0/ERPU0/EWPU0 スレーブ属性 PC。 PC 番号に対応するビット 0: 禁止 1: 許可(pc_mask が 0x0005 の時, PC0 および PC2 が許可)
.fwpu0_ms_att.urw	FWPU0/EWPU0 マスタ属性 ユーザライト 0: 禁止 1: 許可 EWPU0 マスタ属性 ユーザリード 0: 禁止 1: 許可
.fwpu0_ms_att.prw	FWPU0/EWPU0 マスタ属性 特権ライト 0: 禁止 1: 許可 EWPU0 マスタ属性 特権リード 0: 禁止 1: 許可
.fwpu0_ms_att.ns	FWPU0/EWPU0 マスタ属性 非セキュアライト 0: 禁止 1: 許可 EWPU0 マスタ属性 非セキュアリード 0: 禁止 1: 許可
.fwpu0_ms_att.pc_mask	FWPU0/ERPU0/EWPU0 マスタ属性 PC。PC 番号に対応するビット 0: 禁止 1: 許可 (pc_mask が 0x0005 の時, PC0 および PC2 が許可)

(続く)

14 Appendix F - アプリケーション保護設定

表 10 (続き) 要素の説明

要素	説明
.erpu0/ewpu0_offset.offset	ERPU0 ベースアドレスオフセット
.erpu0/ewpu0_size.region_size	ERPU0 領域サイズ
.erpu0/ewpu0_size.enable	ERPU0 イネーブル. 0: ディセーブル 1: イネーブル

15 Appendix G - ノーマルアクセス制限

このプロジェクトは、ノーマルアクセス制限 (NAR)の設定を提供します。NAR は、NORMAL_PROVISIONED および SECURE_W_DEBUG の DAP 制限を設定します。NAR はフラッシュブートで設定されます。アーキテクチャ TRM[2] の BootROM の章およびセキュリティエンハンス PPU コンフィグレーションを参照してください。

このプロジェクトでは、M0+_DAP, M4_DAP, および System_DAP のノーマルアクセス制限とノーマルデッドアクセス制限を設定できます。

15.1 設定

以下のコードは、ノーマルアクセス制限を設定する手順を示します。このセクションは、SFlash の 0x17001A00 にあります。

```
        #define CY_SI_CM0_ENABLE                (0UL)  /**< CM0 ACCESS PORT ENABLE */
        #define CY_SI_CM0_DISABLE_TMP           (1UL)  /**< CM0 ACCESS PORT TEMPORARY DISABLE
*/
        #define CY_SI_CM0_DISABLE               (2UL)  /**< CM0 ACCESS PORT
PERMANENTLY_DISABLE */
        #define CY_SI_CM4_ENABLE                (0UL)  /**< CM4 ACCESS PORT ENABLE */
        #define CY_SI_CM4_DISABLE_TMP           (1UL)  /**< CM4 ACCESS PORT TEMPORARY DISABLE
*/
        #define CY_SI_CM4_DISABLE               (2UL)  /**< CM4 ACCESS PORT
PERMANENTLY_DISABLE */
        #define CY_SI_SYS_ENABLE                (0UL)  /**< SYS ACCESS PORT ENABLE */
        #define CY_SI_SYS_DISABLE_TMP           (1UL)  /**< SYS ACCESS PORT TEMPORARY DISABLE
*/
        #define CY_SI_SYS_DISABLE               (2UL)  /**< SYS ACCESS PORT
PERMANENTLY_DISABLE */


        /* Access Restriction */
        #define CY_SI_NAR_NORMALACCESSRESTRICTION ((CY_SI_CM0_ENABLE << CY_SI_CM0_AP_POS) \
                                | (CY_SI_CM4_ENABLE << CY_SI_CM4_AP_POS) \
                                | (CY_SI_SYS_ENABLE << CY_SI_SYS_AP_POS)) \
                                | 0x80                                          /* Fixed
value */

        #define CY_SI_NAR_NORMALDEADACCESSRESTRICTION ((CY_SI_CM0_ENABLE << CY_SI_CM0_AP_POS) \
                                | (CY_SI_CM4_ENABLE << CY_SI_CM4_AP_POS) \
                                | (CY_SI_SYS_ENABLE << CY_SI_SYS_AP_POS))

        CY_SECTION(".cy_SFlash_nar") __USED static const cy_stc_si_nar_t cy_nar =
        {
    .nar         = CY_SI_NAR_NORMALACCESSRESTRICTION,       /* Normal Access Restrictions */
    .ndar        = CY_SI_NAR_NORMALDEADACCESSRESTRICTION,   /* Normal Dead Access Restrictions
*/
        };
```

表 11 に、各要素の詳細を示します。

15 Appendix G - ノーマルアクセス制限

表 11 NAR 設定

要素	説明
CY_SI_CM0/CM4/SYS_ENABLE	M0/M4/SYS-DAP をイネーブルに設定
CY_SI_CM0/CM4/SYS_DISABLE_TMP	M0/M4/SYS-DAP をディセーブルに設定
CY_SI_CM0/CM4/SYS_DISABLE	M0/M4/SYS-DAP を永久的にディセーブルに設定

設定の詳細は、表 5 を参照してください。

16 Appendix H - プログラムとスクリプト例

サンプルプロジェクトには、以下のサンプルプログラムとスクリプトがあります。ここに示すサンプルプログラムとスクリプトは CYT2B7 シリーズを使用しています。また、GHS MULTI を使用してスクリプトを実行します。

16.1 サンプルコード

サンプルプロジェクトには、以下のサンプルプログラムがあります。

- デバッグアクセスポート認証
- TransitiontoSecure API の実行
- ReadUniqueID API の実行
- TransitiontoRMA API の実行
- OpenRMA API の実行

16.1.1 デバッグアクセスポート認証

このサンプルプログラムでは、電源投入後、すべての DAP を無効に設定し起動します。CM0+は Sys_DAP MPU を設定し、Sys_DAP を有効にします。次に、Sys_DAP から MCU に入力される特定の 128 ビットキーを、あらかじめ MCU に保存されている 128 ビットキーと比較します。一致した場合、CM0+_DAP と CM4_DAP が有効になります。以下は、このサンプルプログラムの概要です。

- CM0+は DAP 制限を制御します
- NAR 設定
  - AP_CTL_M0_DISABLE: 1 (=無効)
  - AP_CTL_M4_DISABLE: 1 (=無効)
  - AP_CTL_SYS_DISABLE: 1 (=無効)
  - SYS_AP_MPU_ENABLE: 0 (=無効)
- Sys_DAP MPU 設定
  - リージョン 0: バックグラウンドリージョン
    - ベースアドレス: 0x00000000
    - サブリージョン: 0x00
    - サイズ: 4 GB
    - 属性: アクセスなし、セキュア
  - リージョン 1: MMIO リージョン (IPC チャネル構造 0/1/2 のみ)
    - ベースアドレス: 0x40220000
    - サブリージョン: 0xf8
    - サイズ: 256 バイト
    - 属性: オールアクセス、非セキュア
  - リージョン 6: SRAM 領域 (0x08001000～0x08003FFF)
    - ベースアドレス: 0x08000000
    - サブリージョン: 0x03 (SRAM の最初の 4KB は無効になっています)
    - サイズ: 16 キロバイト
    - 属性: オールアクセス、非セキュア
- 128 ビットキーは WorkFlash に保存されます (アドレス= 0x14012000)
- Sys_DAP から入力されたパラメータ (128 ビットキーを含む) は、IPC2 を介して渡されます。図 14 に、パラメータの構造を示します。

16 Appendix H - プログラムとスクリプト例

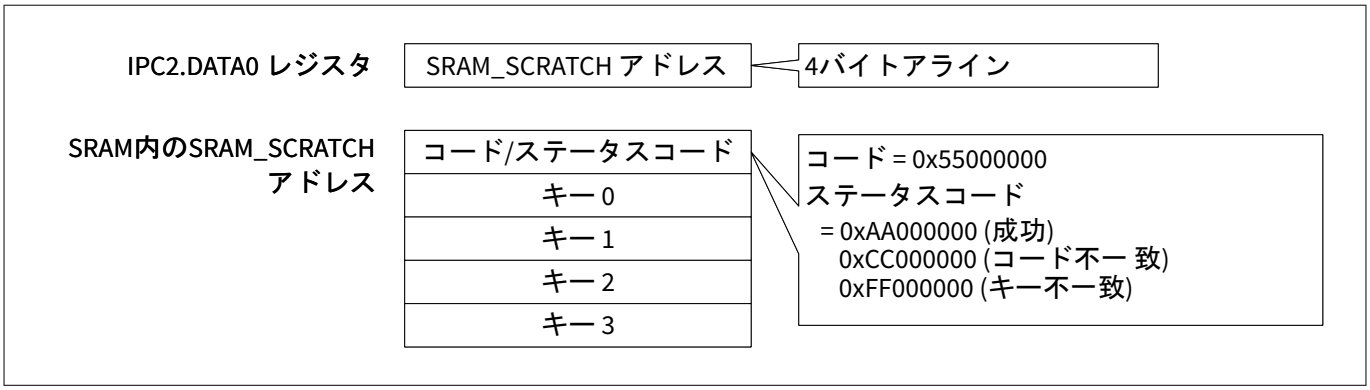


図 14　　パラメータ構造

これらのパラメータは、スクリプトを使用して GHS MULTI から Sys-DAP を介して入力されます。

- IPC2 は、パラメータの設定後、CM0+への notify 割込みを生成します。
- IPC2 notify 割込みは IRQ3 (優先度 3) にマップされます。

図 15 に、このサンプルプログラムの割込み接続を示します。

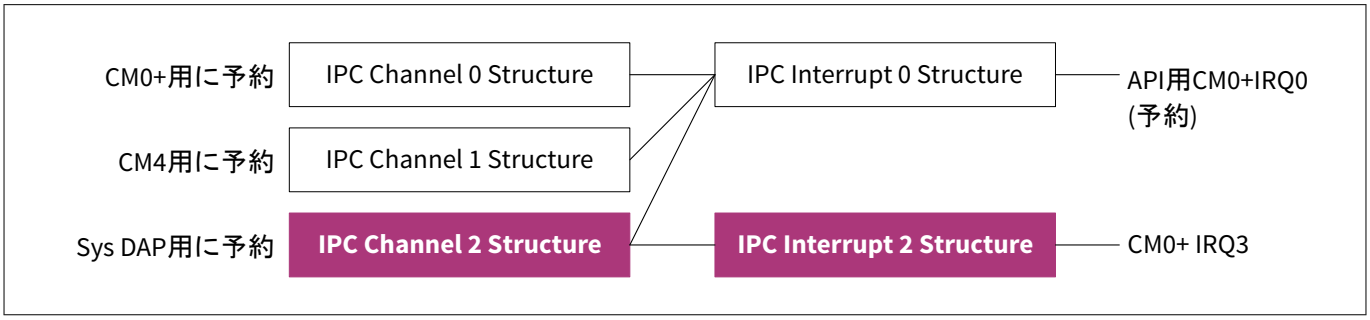


図 15　　割込み接続

注: CYT2B シリーズでは、IPC channel 2 は Sys_DAP 用に予約されています。IPC channel 2 は、Sys_DAP から API を起動するために割り当てられた IPC チャネル番号です。製品によって割り当てられた IPC チャネルについては、アーキテクチャ TRM[2] を参照してください。

動作フロー

図 16 にサンプルプログラムのソフトウェアフローを示します。

16 Appendix H - プログラムとスクリプト例

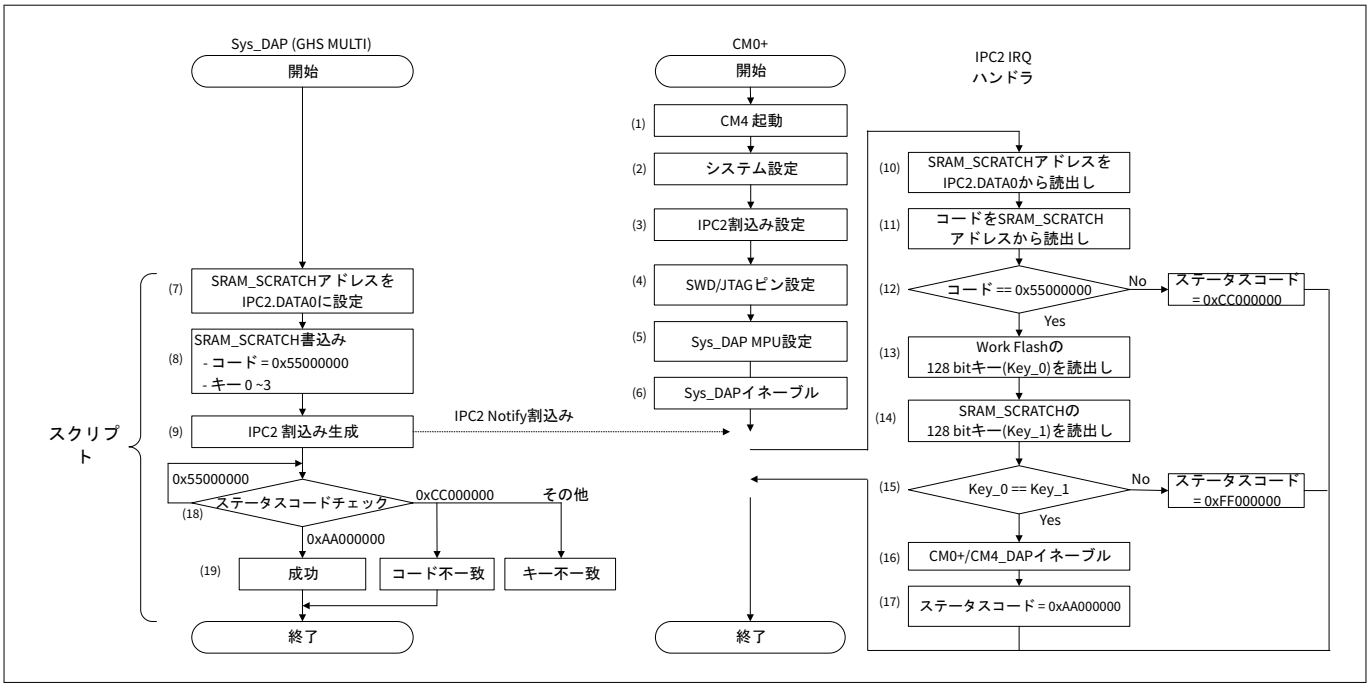


図 16　　DAP 認証のソフトウェアフロー

1. CM0+は CM4 を起動します。
2. CM0+は初期設定を行います。
3. CM0+は IPC2 および割込みコントローラを設定します。
4. CM0+は SWD/JTAG ピンを設定します
5. CM0+は Sys_MPU を設定します。
6. Sys_DAP をイネーブルにします。
7. Sys_DAP から IPC2.DATA0 レジスタに SRAM_SCRATCH アドレスを書き込みます。
8. Sys_DAP から SRAM_SCRATCH アドレスにコードとキーデータを書き込みます。
9. IPC2.Notify を割込み生成に設定します。
10. CM0+は割込みを受付けると、IPC2.DATA0 から SRAM_SCRATCH アドレスが読み出されます。
11. SRAM_SCRATCH アドレスからコードを読み出します。
12. コードが 0x55000000 かどうかチェックし、0x55000000 でない場合は、0xCC000000 をステータスコードに設定し、割込みハンドラから戻ります。
13. Work Flash からキー (Key_0) を読み出します。
14. SRAM_SCRATCH アドレスからキー (Key_1) を読み出します。
15. Key_0 と Key_1 を比較します。一致しない場合、0xFF000000 をステータスコードに設定し、割込みハンドラから戻ります。
16. Key_0 と Key_1 が一致した場合、CM0+_DAP と CM4_DAP をイネーブルにします。
17. 0xAA000000 をステータスコードに設定し、割込みハンドラから戻ります。
18. ステータスコードをチェックします。
19. ステータスコードが 0xAA000000 の場合、DAP のイネーブルは成功です。ステータスコードが 0xCC000000 または 0xFF000000 の場合、DAP のイネーブルに失敗しています。

注:　　これは例です。セキュリティ要件に基づいて個々の CPU へコードを割当てる必要があります。

設定

DAP 認証の設定手順は以下のとおりです。これらのプログラムコードとスクリプトは以下にあります。
<user>\tviibe1m\src\DAP_restrict

<user>: サンプルプロジェクトが格納されたフォルダ

16 Appendix H - プログラムとスクリプト例

1. キーデータを設定してください。
Sys_DAP から入力される 128 ビットキーと比較するキーデータを Work Flash に設定してください。

```
uint8_t  programData[SIZE_128_BIT_IN_BYTE] = {      --> Key Data in Work Flash
0x33, 0x22, 0x11, 0x00,                             --> Key Data 1
0x77, 0x66, 0x55, 0x44,                             --> Key Data 2
0xbb, 0xaa, 0x99, 0x88,                             --> Key Data 3
0xff, 0xee, 0xdd, 0xcc,                             --> Key Data 4
};
```

2. NAR を設定してください。
システムに従い NAR を設定してください。

```
/* Access Restriction */
/* CY_SI_DIRECT_EXE_DISABLE in CY_SI_NAR_NORMALACCESSRESTRICTION is fixed value */
#define CY_SI_NAR_NORMALACCESSRESTRICTION ((CY_SI_CM0_DISABLE_TMP << CY_SI_CM0_AP_POS) \
                                         | (CY_SI_CM4_DISABLE_TMP << CY_SI_CM4_AP_POS) \
                                         | (CY_SI_SYS_DISABLE_TMP << CY_SI_SYS_AP_POS) \
                                         | (CY_SI_MPU_DISABLE << CY_SI_AP_MPU_POS)\
                                         | (CY_SI_DIRECT_EXE_DISABLE <<
CY_SI_DIRECT_EXECUTE_POS) \
                                         | (CY_SI_FLASH_ ENABLE << CY_SI_FLASH_POS) \
                                         | (CY_SI_RAM0_ENABLE << CY_SI_RAM0_POS) \
                                         | (CY_SI_WORK_FLASH_ ENABLE <<
CY_SI_WORK_FLASH_POS) \
                                         | (CY_SI_SFLASH_ENABLE << CY_SI_SFLASH_POS) \
                                         | (CY_SI_MMIO_ENABLE << CY_SI_MMIO_POS))
```

この設定によりすべての DAP はディセーブルになります。Sys_DAP MPU は、アプリケーション ソフトウェアによって設定されます。

***注**: CM0+および CM4 のアプリケーションソフトウェアをプログラミング後、DAP を無効に設定することを推奨します。そうでない場合、アプリケーションソフトウェアがプログラムできない場合があります。*

3. SRAM_SCRATCH パラメータを設定してください。
パラメータとして SRAM に書込むデータを作成してください。これらのパラメータはテキストデータです。

```
0x55000000              --> Code
0x00112233              --> Key Data 1
0x44556677              --> Key Data 2
0x8899aabb              --> Key Data 3
0xccddeeff              --> Key Data 4
```

4. 実行してください。

- スクリプトファイルを<user>\tviibe1m\tools\ghs にコピーしてください。
- GHS MULTI デバッグウィンドウのコマンドペインに次のコマンドを入力してください。

```
MULTI> python OpenDAP.py [file name] [SRAM_SCRATCH Address]
```

ここで、
[file name] は、コードとキーが書込まれたファイルです。
[SRAM_SCRATCH Address] は、有効な領域を指定する必要があります。

16.1.2 TransitiontoSecure API 実行

この API は、MCU のライフサイクルステージを「SECURE」または「SECURE_W_DEBUG」に移行します。この API は FACTORY_HASH を検証し、SECURE_HASH, セキュアアクセス制限, およびデッドアクセス制限を eFuse にプログラムします。

この API の実行は、MCU のライフサイクルステージが「NORMAL_PROVISIONED」である必要があります。さらに、ライフサイクルステージが「SECURE」または「SECURE_W_DEBUG」に移行した後、MCU は「NORMAL_PROVISIONED」ライフサイクルステージに戻れないことに注意してください。

注: SECURE または SECURE_W_DEBUG ライフサイクルステージに移行する場合は、ROM ブートのデフォルトクロック設定で呼び出すことを推奨します。詳細についてはアーキテクチャ TRM[2] の Blow Fuse Bit を参照してください。

16.1.2.1 サンプルプログラム

このサンプルプログラムは以下にあります。<user>\tviibe1m\src\APIs_usage

<user>: サンプルプロジェクトが格納されたフォルダ

この API は以下のパラメータを持ち、CM0+から実行されます。

SECURE 移行のパラメータ設定は以下です。

```
#define SECURE                  (0x0ul)
#define SECURE_ACC_RESTRICT     (0x00000000ul) // For SECURE
#define DEAD_ACC_RESTRICT       (0x00000000ul) // For SECURE
```

SECURE_W_DEBUG 移行のパラメータは以下です。

```
#define SECURE_WITH_DEBUG       (0x1ul)
#define SECURE_WD_ACC_RESTRICT  (0x00000080ul) // For SECURE WITH DEBUG
#define DEAD_WD_ACC_RESTRICT    (0x00000000ul) // For SECURE WITH DEBUG
```

16.1.2.2 スクリプト

このスクリプトは以下にあります。<user>\tviibe1m\tools\ghs\Script

<user>: サンプルプロジェクトが格納されたフォルダ

実行手順を以下に示します。

1. スクリプトファイルを<user>\tviibe1m\tools\ghs にコピーしてください。
2. GHS MULTI デバッグウィンドウのコマンドペインに次のコマンドを入力してください。

```
MULTI> python TransitionToSecure.py(/TransitionToSecure_with_Debug.py) [SECURE_ACCESS_RESTRICT]
[DEAD_ACCESS_RESTRICT] [SRAM_SCRATCH Address]
```

16.1.3 ReadUniqueID API 実行

この API は、SFlash からチップのユニーク ID を読み出します。ユニーク ID は、RMA ライフサイクルステージの移行に使用します。

16.1.3.1 サンプルプログラム

このサンプルプログラムは以下にあります。<user>\tviibe1m\src\APIs_usage

<user>: サンプルプロジェクトが格納されたフォルダ

この API を実行後、ユニーク ID が SRAM_SCRACH に格納されます。この API は CM4 から実行します。

表 12 SRAMSCRACH のユニーク ID

SRAM_SCRATCH アドレス	ビット[31:28]	ビット[23:16]	ビット[15:8]	ビット[7:0]
0x00	ステータスコード 0xA0: 成功 0xF0: 失敗	ユニーク ID_0		
0x04	ユニーク ID_1			
0x08	ユニーク ID_2			

16.1.3.2 スクリプト

このスクリプトは以下にあります。<user>\tviibe1m\tools\ghs\Script

<user>: サンプルプロジェクトが格納されたフォルダ

実行手順を以下に示します。

1. スクリプトファイルを<user>\tviibe1m\tools\ghs にコピーしてください。
2. GHS MULTI デバッグウィンドウのコマンドペインに次のコマンドを入力してください。

```
MULTI> python ReadUniqueID.py [SRAM_SCRATCH Address]
```

3. ユニーク ID を読み出してください。

```
MULTI> target t 0 mr 4 [SRAM_SCRATCH Address]
MULTI> target t 0 mr 4 [SRAM_SCRATCH Address+4]
MULTI> target t 0 mr 4 [SRAM_SCRATCH Address+8]
```

16.1.4 TransitiontoRMA API 実行

この API は、ライフサイクルステージを「RMA」に移行します。TransitiontoRMA API は、部品を SECURE または SECURE_W_DEBUG から RMA ライフサイクルステージに移行します。OpenRMA API を実行すると、RMA ライフサイクルステージの部品を分析できます。また、有効な公開鍵が SFlash に格納されている必要があります。

16.1.4.1 サンプルプログラム

このサンプルプログラムは以下にあります。<user>\tviibe1m\src\APIs_usage

<user>: サンプルプロジェクトが格納されたフォルダ

この API は以下のパラメータを持ち、CM0+から実行されます。パラメータは RSA-2048 の例です。

```
Parameter setting for Transition to RMA.
// Transition To RMA and Open RMA parameters
#define OBJECT_SIZE_TRANSITION_TO_RAM  (0x00000014ul)
#define COMMND_ID_TRAN_TO_RMA          (0x120028F0ul)
#define UNIQUE_ID_0                    (0x028992f2ul) // Need to change per chip
#define UNIQUE_ID_1                    (0xb3000101ul) // Need to change per chip
#define UNIQUE_ID_2                    (0x00140708ul) // Need to change per chip

uint32_t Digital_Signature_TransitionToRMA[64]
=  {0x5c1bbb5dul, 0xc6bae36dul, 0xe48c4044ul, 0xf8dd8131ul, 0x57230ea4ul, 0x4ede3ce5ul,
0xba0b8960ul, 0x5cb4d66ful,
     0x2362e0b7ul, 0x88e6a3f1ul, 0x353caae9ul, 0xf8630ad6ul, 0x2f388ca7ul, 0x9ee16a77ul,
0xce92ec23ul, 0xec16037aul,
     0x00c3b4feul, 0x891409fcul, 0xe0242d69ul, 0xa00b6018ul, 0x2e677254ul, 0x4d64188bul,
0xcef8a86dul, 0x03a0fdf6ul,
     0xf4b511b6ul, 0x123a3df3ul, 0x19f41b52ul, 0xecbad2a7ul, 0x43e3a1cbul, 0xe1ec4fecul,
0xd6c78392ul, 0x63de12b8ul,
     0x1a1f08aaul, 0xd3cdf2a8ul, 0xf0430eb9ul, 0x109decfbul, 0x22a05b23ul, 0x9a3d8181ul,
0x0f4c4f1ful, 0x1c62a64cul,
     0xed971c41ul, 0x57e74366ul, 0x991cb9a4ul, 0xfa8e4bfeul, 0x19ed9cccul, 0xd2fd7e5eul,
0xd517af36ul, 0x66679d89ul,
     0x004ec3f4ul, 0xafcbf55eul, 0xaf0a4a85ul, 0xb09f362cul, 0x7b7abe4aul, 0x626e826dul,
0x56511b05ul, 0x7089b60eul,
     0xe15cac61ul, 0x83bd7290ul, 0x45beac77ul, 0x1eb7d67aul, 0xc886dc0bul, 0x05736051ul,
0xf8105b97ul, 0x680a5d41ul,};
```

注: TansitiontoRMA API の実行条件については、RMA ライフサイクルステージ移行の要件を参照してください。

16.1.4.2 スクリプト

このスクリプトは以下にあります。<user>\tviibe1m\tools\ghs\Script

<user>: サンプルプロジェクトが格納されたフォルダ

実行手順を以下に示します。

1. 証明書とデジタル署名を作成してください。TransitiontoRMA API の証明書およびデジタル署名の生成は、Appendix E - RMA ライフサイクルステージへの移行を参照してください。
2. スクリプトファイルを<user>\tviibe1m\tools\ghs にコピーしてください。
3. GHS MULTI デバッグウィンドウのコマンドペインに次のコマンドを入力してください。

```
MULTI> python TransitionRMA.py [file name] [SRAM_SCRATCH Address] [DIGITAL_SIGNATURE
Address]
```

ここで、

[file name]は、TransitiontoRMA API の証明書とデジタル署名が書込まれたファイルです。

注: *TansitiontoRMA API の実行条件については、RMA ライフサイクルステージ移行の要件を参照してください。*

注: *Sys_DAP から TransitiontoRMA API を実行する場合、デバッガによるシステムコール起動の最小要件を参照してください。*

16.1.5 OpenRMA API の実行

RMA ライフサイクルステージのデバイスは、OpenRMA API を実行することで解析可能になります。この API を実行するためには証明書とデジタル署名が必要です。また、有効な公開鍵が SFlash に格納されている必要があります。

16.1.5.1 スクリプト

このスクリプトは以下にあります。<user>\ tviibe1m\tools\ghs\Script

<user>: サンプルプロジェクトが格納されたフォルダ

1. 実行手順を以下に示します。証明書とデジタル署名を作成してください。TransitiontoRMA API の証明書およびデジタル署名の生成は、Appendix E - RMA ライフサイクルステージへの移行を参照してください。
2. スクリプトファイルを<user>\tviibe1m\tools\ghs にコピーしてください。
3. GHS MULTI デバッグウィンドウのコマンドペインに次のコマンドを入力してください。

```
MULTI> python TransitionRMA.py [file name] [SRAM_SCRATCH Address] [DIGITAL_SIGNATURE
Address]
```

ここで、

[file name]は、OpenRMA API の証明書とデジタル署名が書込まれたファイルです。

注: *OpenRMA API の実行条件については、RMA ライフサイクルステージ移行の要件を参照してください。*

16.2 API の実行

このサンプルプロジェクトでは、main_cm0plus.c ソースファイルの「ExecuteAPI」で API の実行を選択できます。表 13 に、ExecuteAPI 設定と API の実行を示します。

表 13 ExecuteAPI パラメータ

ExecuteAPI	実行する API
= 1	TransitiontoRMA API
= 3	TransitiontoSecure API (for SECURE)
= 4	TransitiontoSecure API (for SECURE_W_DEBUG)
その他	ReadUniqueID API (デフォルト: 0)

(続く)

表 13 (続き) ExecuteAPI パラメータ

ExecuteAPI	実行する API

```
//    SystemInit();
//    Fault report structure setting in User Application
// Masked ECC errors for OpenRMA
// Need to select Fault report structure according to your system (Fault report Structure 0 is
used as an example)
if(CPUSS->unPROTECTION.stcField.u3STATE == 1ul) {
 Cy_SysFlt_ClearMaskByIdx(FAULT_STRUCT0, CY_SYSFLT_RAMC0_C_ECC);   // Mask RAM0 correctable ECC
violation
 Cy_SysFlt_ClearMaskByIdx(FAULT_STRUCT0, CY_SYSFLT_RAMC0_NC_ECC);  // Mask RAM0 non-correctable
ECC violation
 Cy_SysFlt_ClearMaskByIdx(FAULT_STRUCT0, CY_SYSFLT_CRYPTO_C_ECC);  // Mask Crypto memory
correctable ECC violation
 Cy_SysFlt_ClearMaskByIdx(FAULT_STRUCT0, CY_SYSFLT_CRYPTO_NC_ECC); // Mask Crypto memory non-
correctable ECC violation
}

switch (ExecuteAPI) {
case 1:
// Masked ECC errors for TransitiontoRMA
// Need to select Fault report structure according to your system (Fault report Structure 0 is
used as an example)
  Cy_SysFlt_ClearMaskByIdx(FAULT_STRUCT0, CY_SYSFLT_RAMC0_C_ECC);   // Mask RAM0 correctable
ECC violation
  Cy_SysFlt_ClearMaskByIdx(FAULT_STRUCT0, CY_SYSFLT_RAMC0_NC_ECC);  // Mask RAM0 non-
correctable ECC violation
  Cy_SysFlt_ClearMaskByIdx(FAULT_STRUCT0, CY_SYSFLT_CRYPTO_C_ECC);  // Mask Crypto memory
correctable ECC violation
  Cy_SysFlt_ClearMaskByIdx(FAULT_STRUCT0, CY_SYSFLT_CRYPTO_NC_ECC); // Mask Crypto memory non-
correctable ECC violation

  params.TransitionToRMA.arg0.Opcode = OPCODE_TRANSITION_TO_RMA;
  params.TransitionToRMA.arg1.Objsize = OBJECT_SIZE_TRANSITION_TO_RAM;
  params.TransitionToRMA.arg2.CommandId = COMMND_ID_TRAN_TO_RMA;
  params.TransitionToRMA.arg3.UniqueID_0 = UNIQUE_ID_0;
  params.TransitionToRMA.arg4.UniqueID_1 = UNIQUE_ID_1;
  params.TransitionToRMA.arg5.UniqueID_2 = UNIQUE_ID_2;
  params.TransitionToRMA.arg6.dataAddr = (uint32_t)Digital_Signature_TransitionToRMA;
  break;
case 3:
  params.TransitionToSecure.arg0.Opcode = OPCODE_TRANSITION_TO_SECURE;
  params.TransitionToSecure.arg0.Debug  = SECURE;
  params.TransitionToSecure.arg1.Acc_restrict = SECURE_ACC_RESTRICT;
  params.TransitionToSecure.arg2.Dead_Acc_restrict = DEAD_ACC_RESTRICT;
  break;
case 4:
  params.TransitionToSecure.arg0.Opcode = OPCODE_TRANSITION_TO_SECURE;
  params.TransitionToSecure.arg0.Debug  = SECURE_WITH_DEBUG;
  params.TransitionToSecure.arg1.Acc_restrict = SECURE_WD_ACC_RESTRICT;
```

```
    params.TransitionToSecure.arg2.Dead_Acc_restrict = DEAD_WD_ACC_RESTRICT;
    break;
  default:
    params.RdUnId.arg0.Opcode = OPCODE_READ_UNIQUEID;
    break;
  }

  Cy_IPC_Drv_SendMsgWord(syscall_ipc_struct, CY_SROM_DR_IPC_NOTIFY_STRUCT, (uint32_t)&params);
```

はじめに、デバイスが RMA であるかどうかを確認します。保護状態が「1」の場合、つまり VIRGIN は、ECC エラーフォールト要因をマスクします。次に、ExecuteAPI が「1」の場合、デバイスは TransitiontoRMA API を実行します。TransitiontoRMA API を実行する前にも、ECC エラーフォールト要因をマスクします。詳細については、RMA ライフサイクルステージ移行の要件を参照してください。

16.3 セキュリティエンハンス PPU 設定例

セキュリティエンハンスメントが有効な場合、プログラマブル PPU11, 12, 13, および PERI_MS_PPU_FX_CPUSS_CM0 (図 6 を参照してください)を設定するサンプルプログラムを示します。セキュアシステムでは、これらの PPU 設定は、HSM ソフトウェアを使用して行う必要があります。

図 17 に、このサンプルプログラムのソフトウェアフローを示します。

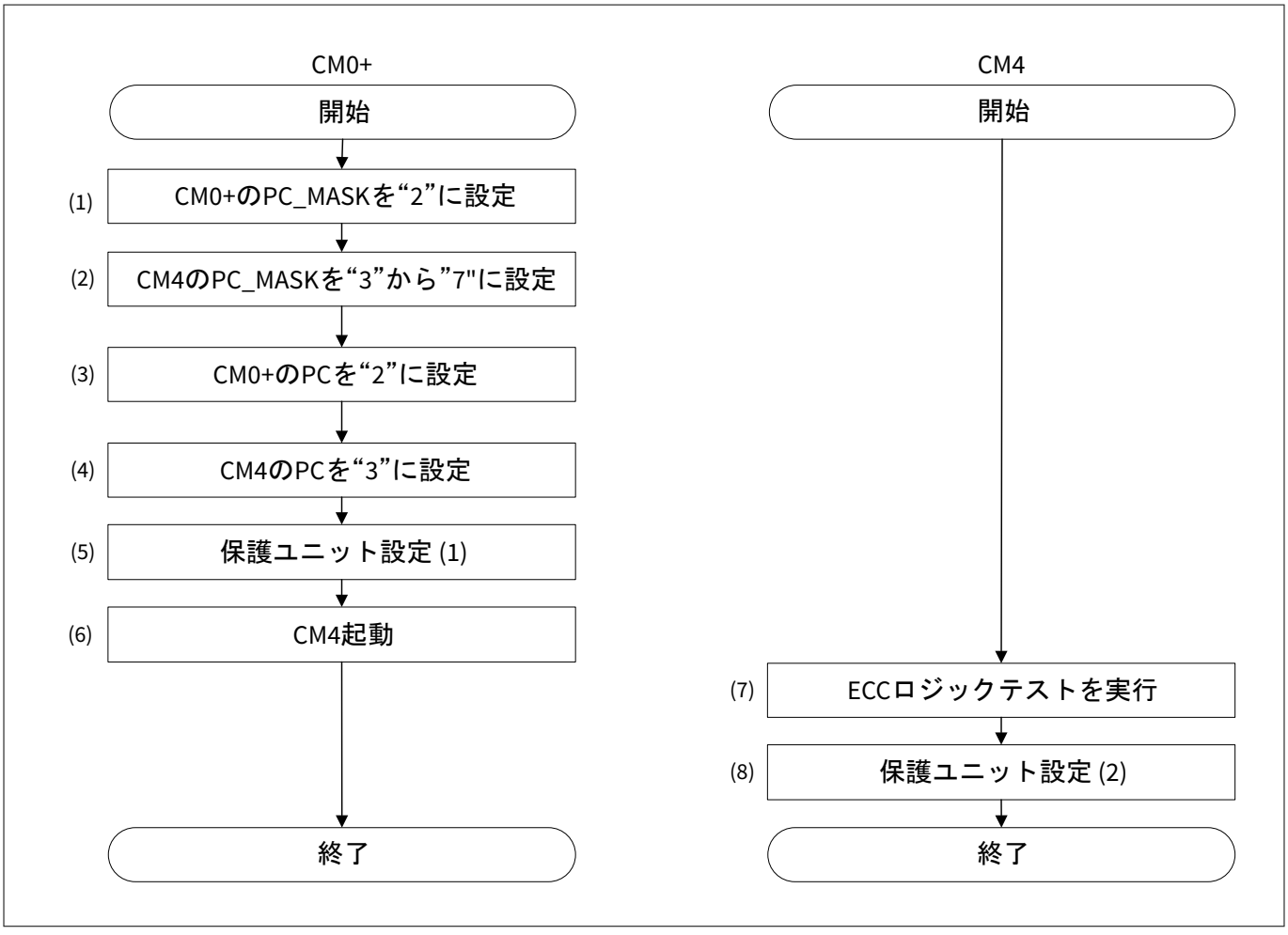


図 17 セキュリティエンハンスメントでのプロテクションユニット設定

16 Appendix H - プログラムとスクリプト例

以下の設定は CM0+ (HSM ソフトウェア) によって実行されます。

1. CM0+の PC_MASK を「2」に設定します。CM0+は PC=「2」のみ設定できます。
2. CM4 の PC_MASK を「3」から「7」に設定します。CM4 は PC=「3」から「7」を設定できます。
3. CM0+の PC を「2」に設定します。HSM ソフトウェアは PC=「2」を使用します。
4. CM4 の PC を「3」に設定します。アプリケーションソフトウェアは PC=「3」を使用します。
5. 保護ユニットを設定します。(1)
6. CM4 を起動します。
   以下の設定は CM4 によって実行されます。
7. ECC ロジックテストを実行します。
8. 保護ユニットを設定します。(2)

表 14 に、各 CPU によって実行される PPU 設定を示します。

表 14　　保護ユニット設定

PPU 名	CM0+による保護ユニット設定(1) (マスタ/スレーブ)	CM4 による保護ユニット設定(2) (マスタ/スレーブ)
プログラマブル PPU 11	PC3: Read / Write PC1, 2, 4-7: Read only	-
プログラマブル PPU 12	PC2, 3: Read / Write PC1, 4-7: Read only	PC2: Read / Write PC1, 3-7: Read only
プログラマブル PPU 13	PC3: Read / Write PC1, 2, 4-7: Read only	-
PERI_MS_PPU_FX_CPUSS_CM0 (PPU index=17)	PC2: Read / Write PC1, 3-7: Read only	-

表 15 にパラメータ、表 16 に保護ユニット設定のための SDL 関数を示します。

表 15　　設定パラメータ

パラメータ	説明	値
CY_SECURITY_ENHANCED	セキュリティエンハンスメントマーカ	0xFEDEEDDF
CY_PROT_PC2 to PC7	保護コンテキスト値	2u ～ 7u
PC_MASK_OF_PC2 to PC7	保護コンテキストマスク値	1u<<(CY_PROT_PCx-1u) x = 2 ～ 7
CY_PROT_PERM_R	リードオンリ属性	0x01u
CY_PROT_PERM_RW	リードライト属性	0x03u
CPUSS_MS_ID_CM0	CM0+バスマスタ ID	0
CPUSS_MS_ID_CM4	CM4 バスマスタ ID	14
PERI_MS_PPU_PR11	プログラマブル PPU#11 ベースアドレス	0x400102C0
PERI_MS_PPU_PR12	プログラマブル PPU#12 ベースアドレス	0x40010300

(続く)

16 Appendix H - プログラムとスクリプト例

表 15 (続き) 設定パラメータ

パラメータ	説明	値
PERI_MS_PPU_PR13	プログラマブル PPU#13 ベースアドレス	0x40010340
PERI_MS_PPU_FX_CPUSS_CM0	PERI_MS_PPU_FX_CPUSS_CM0 PPU ベースアドレス	0x40010C40
ppuFixedAttr_ReadOnly.userPermission	固定 PPU ユーザリードオンリ属性	CY_PROT_PERM_R
ppuFixedAttr_ReadOnly.privPermission	固定 PPU 特権リードオンリ属性	CY_PROT_PERM_R
ppuFixedAttr_ReadOnly.secure	固定 PPU セキュアリードオンリ属性	0ul
ppuFixedAttr_Readwrite.userPermission	固定 PPU ユーザリードライト属性	CY_PROT_PERM_RW
ppuFixedAttr_Readwrite.privPermission	固定 PPU 特権リードライト属性	CY_PROT_PERM_RW
ppuFixedAttr_Readwrite.secure	固定 PPU セキュアリードライト属性	0ul
ppuProgAttr_ReadOnly.userPermission	プログラマブル PPU ユーザリードオンリ属性	CY_PROT_PERM_R
ppuProgAttr_ReadOnly.privPermission	プログラマブル PPU 特権リードオンリ属性	CY_PROT_PERM_R
ppuProgAttr_ReadOnly.secure	プログラマブル PPU セキュアリードオンリ属性	0ul
ppuProgAttr_Readwrite.userPermission	プログラマブル PPU ユーザリードライト属性	CY_PROT_PERM_RW
ppuProgAttr_Readwrite.privPermission	プログラマブル PPU 特権リードライト属性	CY_PROT_PERM_RW
ppuProgAttr_Readwrite.secure	プログラマブル PPU セキュアリードライト属性	0ul

表 16 保護設定関数

関数	説明	補足
Cy_Prot_ConfigBusMaster (busMaster, privileged, secure, pcMask)	保護コンテキストマスク設定	-
Cy_Prot_SetActivePC(busMaster, pc)	保護コンテキスト設定	-
Init_ProtectionUnit_Secure_Enahnce()	保護ユニット (1) 設定	-
Cy_Prot_ConfigPpuProgSlaveStructAttr(base, setPC, config)	プログラマブル PPU スレーブ設定	-
Cy_Prot_ConfigPpuProgMasterStructAttr (base, setPC, config)	プログラマブル PPU マスタ設定	-
Cy_Prot_ConfigPpuFixedSlaveStruct (base, setPC, config)	固定 PPU スレーブ設定	-
Cy_Prot_ConfigPpuFixedMasterStruct (base, setPC, config)	固定 PPU スレーブマスタ設定	-
Invalidate_ProtectionUnit_PPU12()	保護ユニット (2)設定	-

16 Appendix H - プログラムとスクリプト例

以下は、SDL ドライバのレジスタ表記例を示します。

- addrMpu->unMS_CTL.u32Register はレジスタ TRM[2]の PROT_MPUx_MS_CTL register を示します。他のレジスタも同様に記述されます。「x」はバスマスタ ID を示します。
- パフォーマンス改善策
- レジスタ設定のパフォーマンス向上のため、SDL は完全な 32 ビットデータをレジスタに書き込みます、各ビットフィールドは、ビット書込み可能なバッファであらかじめ生成され、最終的に 32 ビットデータとしてレジスタに書き込まれます。

```
tempSL_ATT0.u32Register = base->unSL_ATT0.u32Register;
tempSL_ATT0.stcField.u1PC1_UR = (config->userPermission & CY_PROT_PERM_R);
    tempSL_ATT0.stcField.u1PC1_UW = (config->userPermission & CY_PROT_PERM_W) >> 1;
    tempSL_ATT0.stcField.u1PC1_PR = (config->privPermission & CY_PROT_PERM_R);
    tempSL_ATT0.stcField.u1PC1_PW = (config->privPermission & CY_PROT_PERM_W) >> 1;
    tempSL_ATT0.stcField.u1PC1_NS = !(config->secure);
    base->unSL_ATT0.u32Register = tempSL_ATT0.u32Register;
```

Code Listing 1 ～ Code Listing 10 の保護ユニット設定のサンプルコードを参照してください。

16 Appendix H - プログラムとスクリプト例

Code Listing 1 保護ユニット設定 (CM0+)

```
CY_SECTION(".cy_toc_part2") __USED static const cy_stc_si_toc_t cy_toc2 =
{
:
   /*  Set Security marker to valid  */
.securityMarker = CY_SECURITY_ENHANCED,          /* Offset+0xFC Security Enhance Marker */
:
};

/*******************************************************************************/
/* Programmable PPU11, 12, 13, and FixePPU 17 Configuration */
/*******************************************************************************/
#define PC_MASK_OF_PC2 (1u<<(CY_PROT_PC2-1u))   /*  Define PC_MASK value  */
#define PC_MASK_OF_PC3 (1u<<(CY_PROT_PC3-1u))
#define PC_MASK_OF_PC4 (1u<<(CY_PROT_PC4-1u))
#define PC_MASK_OF_PC5 (1u<<(CY_PROT_PC5-1u))
#define PC_MASK_OF_PC6 (1u<<(CY_PROT_PC6-1u))
#define PC_MASK_OF_PC7 (1u<<(CY_PROT_PC7-1u))

const cy_stc_ppu_gr_cfg_t  ppuFixedAttr_ReadOnly =   /*  Fixed PPU attribute (Read only)  */
{
    .userPermission = CY_PROT_PERM_R,
    .privPermission = CY_PROT_PERM_R,
    .secure         = 0ul,
};
const cy_stc_ppu_gr_cfg_t  ppuFixedAttr_Readwrite =   /*  Fixed PPU attribute (Read/Write)   */
{
    .userPermission = CY_PROT_PERM_RW,
    .privPermission = CY_PROT_PERM_RW,
    .secure         = 0ul,
};
const cy_stc_ppu_prog_attr_cfg_t  ppuProgAttr_ReadOnly =   /*  Programmable PPU attribute (Read
only)   */
{
    .userPermission = CY_PROT_PERM_R,
    .privPermission = CY_PROT_PERM_R,
    .secure         = 0ul,
};
const cy_stc_ppu_prog_attr_cfg_t  ppuProgAttr_Readwrite =   /*  Programmable PPU attribute
(Read/Write)   */
{
    .userPermission = CY_PROT_PERM_RW,
    .privPermission = CY_PROT_PERM_RW,
    .secure         = 0ul,
};

int main(void)
{
:
/* Protection Context Setting HSM(CM0+) = PC2, Application(CM4) = PC3-7 */
Cy_Prot_ConfigBusMaster(CPUSS_MS_ID_CM0, true, true, PC_MASK_OF_PC2);   /*  (1)CM0+ PC_MASK
setting   */
```

16 Appendix H - プログラムとスクリプト例

```
   /*  (2)CM4 PC_MASK setting   */
Cy_Prot_ConfigBusMaster(CPUSS_MS_ID_CM4, true, true, (PC_MASK_OF_PC3|PC_MASK_OF_PC4|
PC_MASK_OF_PC5|PC_MASK_OF_PC6|PC_MASK_OF_PC7));


Cy_Prot_SetActivePC(CPUSS_MS_ID_CM0, CY_PROT_PC2);   /*  (3)CM0+ PC setting   */
Cy_Prot_SetActivePC(CPUSS_MS_ID_CM4, CY_PROT_PC3);    /*  (4)CM4 PC setting   */

Init_ProtectionUnit_Secure_Enahnce();   /*  (5)Configure protection unit. See Code Listing 2.
*/
:

Cy_SysEnableApplCore(CY_CORTEX_M4_APPL_ADDR);   /*  (6)Activate CM4  */

while(1)
    {
    }
}
```

16 Appendix H - プログラムとスクリプト例

Code Listing 2 Init_ProtectionUnit_Secure_Enahnce()

```
void Init_ProtectionUnit_Secure_Enahnce(void)
{


// Programmable PPU11 Slave structure(PC1-7) Setting
   /*  Configure slave programmable PPU#11. (Slave) See Code Listing 7.  */
Cy_Prot_ConfigPpuProgSlaveStructAttr(PERI_MS_PPU_PR11, CY_PROT_PC1, &ppuProgAttr_ReadOnly);
Cy_Prot_ConfigPpuProgSlaveStructAttr(PERI_MS_PPU_PR11, CY_PROT_PC3, &ppuProgAttr_Readwrite);
Cy_Prot_ConfigPpuProgSlaveStructAttr(PERI_MS_PPU_PR11, CY_PROT_PC4, &ppuProgAttr_ReadOnly);
Cy_Prot_ConfigPpuProgSlaveStructAttr(PERI_MS_PPU_PR11, CY_PROT_PC5, &ppuProgAttr_ReadOnly);
Cy_Prot_ConfigPpuProgSlaveStructAttr(PERI_MS_PPU_PR11, CY_PROT_PC6, &ppuProgAttr_ReadOnly);
Cy_Prot_ConfigPpuProgSlaveStructAttr(PERI_MS_PPU_PR11, CY_PROT_PC7, &ppuProgAttr_ReadOnly);
// Set the current PC last
Cy_Prot_ConfigPpuProgSlaveStructAttr(PERI_MS_PPU_PR11, CY_PROT_PC2, &ppuProgAttr_ReadOnly);







// Programmable PPU11 Master structure(PC1-7) Setting
   /*  Configure programmable PPU#11. (Master) See Code Listing 8.  */
Cy_Prot_ConfigPpuProgMasterStructAttr(PERI_MS_PPU_PR11, CY_PROT_PC1, &ppuProgAttr_ReadOnly);
Cy_Prot_ConfigPpuProgMasterStructAttr(PERI_MS_PPU_PR11, CY_PROT_PC3, &ppuProgAttr_Readwrite);
Cy_Prot_ConfigPpuProgMasterStructAttr(PERI_MS_PPU_PR11, CY_PROT_PC4, &ppuProgAttr_ReadOnly);
Cy_Prot_ConfigPpuProgMasterStructAttr(PERI_MS_PPU_PR11, CY_PROT_PC5, &ppuProgAttr_ReadOnly);
Cy_Prot_ConfigPpuProgMasterStructAttr(PERI_MS_PPU_PR11, CY_PROT_PC6, &ppuProgAttr_ReadOnly);
Cy_Prot_ConfigPpuProgMasterStructAttr(PERI_MS_PPU_PR11, CY_PROT_PC7, &ppuProgAttr_ReadOnly);
// Set the current PC last
Cy_Prot_ConfigPpuProgMasterStructAttr(PERI_MS_PPU_PR11, CY_PROT_PC2, &ppuProgAttr_ReadOnly);


   /*  Configure programmable PPU#12 (Slave)  */
// Programmable PPU12 Slave structure(PC1-7) Setting. This PPU need to reconfigurs after ECC
checking is completed
Cy_Prot_ConfigPpuProgSlaveStructAttr(PERI_MS_PPU_PR12, CY_PROT_PC1, &ppuProgAttr_ReadOnly);
Cy_Prot_ConfigPpuProgSlaveStructAttr(PERI_MS_PPU_PR12, CY_PROT_PC3, &ppuProgAttr_Readwrite);
Cy_Prot_ConfigPpuProgSlaveStructAttr(PERI_MS_PPU_PR12, CY_PROT_PC4, &ppuProgAttr_ReadOnly);
Cy_Prot_ConfigPpuProgSlaveStructAttr(PERI_MS_PPU_PR12, CY_PROT_PC5, &ppuProgAttr_ReadOnly);
Cy_Prot_ConfigPpuProgSlaveStructAttr(PERI_MS_PPU_PR12, CY_PROT_PC6, &ppuProgAttr_ReadOnly);
Cy_Prot_ConfigPpuProgSlaveStructAttr(PERI_MS_PPU_PR12, CY_PROT_PC7, &ppuProgAttr_ReadOnly);
// Set the current PC last
Cy_Prot_ConfigPpuProgSlaveStructAttr(PERI_MS_PPU_PR12, CY_PROT_PC2, &ppuProgAttr_Readwrite);


   /*  Configure programmable PPU#12 (Master)  */
// Programmable PPU12 Master structure(PC1-7) Setting. This PPU need to reconfigurs after ECC
checking is completed
Cy_Prot_ConfigPpuProgMasterStructAttr(PERI_MS_PPU_PR12, CY_PROT_PC1, &ppuProgAttr_ReadOnly);
Cy_Prot_ConfigPpuProgMasterStructAttr(PERI_MS_PPU_PR12, CY_PROT_PC3, &ppuProgAttr_Readwrite);
Cy_Prot_ConfigPpuProgMasterStructAttr(PERI_MS_PPU_PR12, CY_PROT_PC4, &ppuProgAttr_ReadOnly);
```

```
Cy_Prot_ConfigPpuProgMasterStructAttr(PERI_MS_PPU_PR12, CY_PROT_PC5, &ppuProgAttr_ReadOnly);
Cy_Prot_ConfigPpuProgMasterStructAttr(PERI_MS_PPU_PR12, CY_PROT_PC6, &ppuProgAttr_ReadOnly);
Cy_Prot_ConfigPpuProgMasterStructAttr(PERI_MS_PPU_PR12, CY_PROT_PC7, &ppuProgAttr_ReadOnly);
// Set the current PC last
Cy_Prot_ConfigPpuProgMasterStructAttr(PERI_MS_PPU_PR12, CY_PROT_PC2, &ppuProgAttr_Readwrite);


// Programmable PPU13 Slave structure(PC1-7) Setting
   /*  Configure programmable PPU#13 (Slave)  */
Cy_Prot_ConfigPpuProgSlaveStructAttr(PERI_MS_PPU_PR13, CY_PROT_PC1, &ppuProgAttr_ReadOnly);
Cy_Prot_ConfigPpuProgSlaveStructAttr(PERI_MS_PPU_PR13, CY_PROT_PC3, &ppuProgAttr_Readwrite);
Cy_Prot_ConfigPpuProgSlaveStructAttr(PERI_MS_PPU_PR13, CY_PROT_PC4, &ppuProgAttr_ReadOnly);
Cy_Prot_ConfigPpuProgSlaveStructAttr(PERI_MS_PPU_PR13, CY_PROT_PC5, &ppuProgAttr_ReadOnly);
Cy_Prot_ConfigPpuProgSlaveStructAttr(PERI_MS_PPU_PR13, CY_PROT_PC6, &ppuProgAttr_ReadOnly);
Cy_Prot_ConfigPpuProgSlaveStructAttr(PERI_MS_PPU_PR13, CY_PROT_PC7, &ppuProgAttr_ReadOnly);
// Set the current PC last
Cy_Prot_ConfigPpuProgSlaveStructAttr(PERI_MS_PPU_PR13, CY_PROT_PC2, &ppuProgAttr_ReadOnly);


// Programmable PPU13 Master structure(PC1-7) Setting
   /*  Configure programmable PPU#13 (Master)  */
Cy_Prot_ConfigPpuProgMasterStructAttr(PERI_MS_PPU_PR13, CY_PROT_PC1, &ppuProgAttr_ReadOnly);
Cy_Prot_ConfigPpuProgMasterStructAttr(PERI_MS_PPU_PR13, CY_PROT_PC3, &ppuProgAttr_Readwrite);
Cy_Prot_ConfigPpuProgMasterStructAttr(PERI_MS_PPU_PR13, CY_PROT_PC4, &ppuProgAttr_ReadOnly);
Cy_Prot_ConfigPpuProgMasterStructAttr(PERI_MS_PPU_PR13, CY_PROT_PC5, &ppuProgAttr_ReadOnly);
Cy_Prot_ConfigPpuProgMasterStructAttr(PERI_MS_PPU_PR13, CY_PROT_PC6, &ppuProgAttr_ReadOnly);
Cy_Prot_ConfigPpuProgMasterStructAttr(PERI_MS_PPU_PR13, CY_PROT_PC7, &ppuProgAttr_ReadOnly);
// Set the current PC last
Cy_Prot_ConfigPpuProgMasterStructAttr(PERI_MS_PPU_PR13, CY_PROT_PC2, &ppuProgAttr_ReadOnly);




// Fixed PPU17 Slave structure(PC1-7) Setting
   /*  Configure PERI_MS_PPU_FX_CPUSS_CM0. (Slave). See Code Listing 9.  */
Cy_Prot_ConfigPpuFixedSlaveStruct(PERI_MS_PPU_FX_CPUSS_CM0, CY_PROT_PC1,
&ppuFixedAttr_ReadOnly);
Cy_Prot_ConfigPpuFixedSlaveStruct(PERI_MS_PPU_FX_CPUSS_CM0, CY_PROT_PC3,
&ppuFixedAttr_ReadOnly);
Cy_Prot_ConfigPpuFixedSlaveStruct(PERI_MS_PPU_FX_CPUSS_CM0, CY_PROT_PC4,
&ppuFixedAttr_ReadOnly);
Cy_Prot_ConfigPpuFixedSlaveStruct(PERI_MS_PPU_FX_CPUSS_CM0, CY_PROT_PC5,
&ppuFixedAttr_ReadOnly);
Cy_Prot_ConfigPpuFixedSlaveStruct(PERI_MS_PPU_FX_CPUSS_CM0, CY_PROT_PC6,
&ppuFixedAttr_ReadOnly);
Cy_Prot_ConfigPpuFixedSlaveStruct(PERI_MS_PPU_FX_CPUSS_CM0, CY_PROT_PC7,
&ppuFixedAttr_ReadOnly);
// Set the current PC last
Cy_Prot_ConfigPpuFixedSlaveStruct(PERI_MS_PPU_FX_CPUSS_CM0, CY_PROT_PC2,
&ppuFixedAttr_Readwrite);
```

```
// Fixed PPU17 Master structure(PC1-7) Setting
   /*  Configure PERI_MS_PPU_FX_CPUSS_CM0. (Master) See Code Listing 10.  */
Cy_Prot_ConfigPpuFixedMasterStruct(PERI_MS_PPU_FX_CPUSS_CM0, CY_PROT_PC1,
&ppuFixedAttr_ReadOnly);
Cy_Prot_ConfigPpuFixedMasterStruct(PERI_MS_PPU_FX_CPUSS_CM0, CY_PROT_PC3,
&ppuFixedAttr_ReadOnly);
Cy_Prot_ConfigPpuFixedMasterStruct(PERI_MS_PPU_FX_CPUSS_CM0, CY_PROT_PC4,
&ppuFixedAttr_ReadOnly);
Cy_Prot_ConfigPpuFixedMasterStruct(PERI_MS_PPU_FX_CPUSS_CM0, CY_PROT_PC5,
&ppuFixedAttr_ReadOnly);
Cy_Prot_ConfigPpuFixedMasterStruct(PERI_MS_PPU_FX_CPUSS_CM0, CY_PROT_PC6,
&ppuFixedAttr_ReadOnly);
Cy_Prot_ConfigPpuFixedMasterStruct(PERI_MS_PPU_FX_CPUSS_CM0, CY_PROT_PC7,
&ppuFixedAttr_ReadOnly);


// Set the current PC last
Cy_Prot_ConfigPpuFixedMasterStruct(PERI_MS_PPU_FX_CPUSS_CM0, CY_PROT_PC2,
&ppuFixedAttr_Readwrite);
}
```

Code Listing 3 保護ユニット設定 (CM4)

```
/*******************************************************************************/
/* Programmable PPU12 Configuration */
/*******************************************************************************/
const cy_stc_ppu_prog_attr_cfg_t  ppuProgAttr_ReadOnly =   /*  Programmable PPU attribute (Read
only)   */
{
.userPermission = CY_PROT_PERM_R,
.privPermission = CY_PROT_PERM_R,
.secure         = 0ul,
};

int main(void)
{
:
/*****************************/
   /*  (7)ECC logic test   */
/*   User ECC checking here   */
/*****************************/

   /*  (8)Configure Protection unit. See Code Listing 4.  */
Invalidate_ProtectionUnit_PPU12();     // Read only setting for PC1,3-7

for(;;)
{
}
}
```

16 Appendix H - プログラムとスクリプト例

Code Listing 4 Invalidate_ProtectionUnit_PPU12()

```
void Invalidate_ProtectionUnit_PPU12(void)
{
// Programmable PPU12 Slave structure(PC3) Setting.   /*  Configure programmable PPU#12
(Slave)  */
Cy_Prot_ConfigPpuProgSlaveStructAttr(PERI_MS_PPU_PR12, CY_PROT_PC3, &ppuProgAttr_ReadOnly);

// Programmable PPU12 Master structure(PC3) Setting.   /*  Configure programmable PPU#12
(Master)  */
Cy_Prot_ConfigPpuProgMasterStructAttr(PERI_MS_PPU_PR12, CY_PROT_PC3, &ppuProgAttr_ReadOnly);
}
```

Code Listing 5 Cy_Prot_ConfigBusMaster()

```
cy_en_prot_status_t Cy_Prot_ConfigBusMaster(en_prot_master_t busMaster, bool privileged, bool
secure, uint32_t pcMask)
{
    cy_en_prot_status_t status = CY_PROT_SUCCESS;
    un_PROT_SMPU_MS0_CTL_t tProtSmpuMs0Ctl = {0};
    uint32_t * addrMsCtl = (uint32_t *)(PROT_BASE + (uint32_t)((uint32_t)busMaster <<
CY_PROT_MSX_CTL_SHIFT));

    if((uint32_t)(pcMask & CY_PROT_MPU_PC_LIMIT_MASK) != 0UL)
    {
        status = CY_PROT_BAD_PARAM;
    }
    else
    {
        tProtSmpuMs0Ctl.stcField.u1NS = !secure;
        tProtSmpuMs0Ctl.stcField.u1P = privileged;
        tProtSmpuMs0Ctl.stcField.u15PC_MASK_15_TO_1 = pcMask;

        *addrMsCtl = tProtSmpuMs0Ctl.u32Register; // regVal;
        status = ((*addrMsCtl != tProtSmpuMs0Ctl.u32Register) ? CY_PROT_FAILURE :
CY_PROT_SUCCESS);
    }

    return status;
}
```

16 Appendix H - プログラムとスクリプト例

Code Listing 6 Cy_Prot_SetActivePC ()

```
cy_en_prot_status_t Cy_Prot_SetActivePC(en_prot_master_t busMaster, uint32_t pc)
{
    cy_en_prot_status_t status = CY_PROT_SUCCESS;
    un_PROT_MPU_MS_CTL_t tProtMpuMsCtl = {0};
    volatile stc_PROT_MPU_t* addrMpu = (stc_PROT_MPU_t*)(&PROT->CYMPU[busMaster]);

    if(pc > (uint32_t)CY_PROT_MS_PC_NR_MAX)
    {
        status = CY_PROT_BAD_PARAM;
    }
    else
    {
        tProtMpuMsCtl.stcField.u4PC = pc;
        addrMpu->unMS_CTL.u32Register = tProtMpuMsCtl.u32Register;
        status = ((addrMpu->unMS_CTL.stcField.u4PC != pc) ? CY_PROT_FAILURE : CY_PROT_SUCCESS);
    }
    return status;
}
```

16 Appendix H - プログラムとスクリプト例

Code Listing 7 Cy_Prot_ConfigPpuProgSlaveStructAttr ()

```
cy_en_prot_status_t Cy_Prot_ConfigPpuProgSlaveStructAttr(volatile stc_PERI_MS_PPU_PR_t* base,
cy_en_prot_pc_t setPC, const cy_stc_ppu_prog_attr_cfg_t* config)
{
    cy_en_prot_status_t status = CY_PROT_SUCCESS;
    un_PERI_MS_PPU_PR_SL_ATT0_t tempSL_ATT0 = { 0 };
    un_PERI_MS_PPU_PR_SL_ATT1_t tempSL_ATT1 = { 0 };
    un_PERI_MS_PPU_PR_SL_ATT2_t tempSL_ATT2 = { 0 };
    un_PERI_MS_PPU_PR_SL_ATT3_t tempSL_ATT3 = { 0 };
:
   switch(setPC)
    {
:
    case CY_PROT_PC2:
        tempSL_ATT0.u32Register = base->unSL_ATT0.u32Register;
        tempSL_ATT0.stcField.u1PC2_UR = (config->userPermission & CY_PROT_PERM_R);
        tempSL_ATT0.stcField.u1PC2_UW = (config->userPermission & CY_PROT_PERM_W) >> 1;
        tempSL_ATT0.stcField.u1PC2_PR = (config->privPermission & CY_PROT_PERM_R);
        tempSL_ATT0.stcField.u1PC2_PW = (config->privPermission & CY_PROT_PERM_W) >> 1;
        tempSL_ATT0.stcField.u1PC2_NS = !(config->secure);
        base->unSL_ATT0.u32Register = tempSL_ATT0.u32Register;
:
        break;
:
    default:
        return CY_PROT_BAD_PARAM;
    }

    return status;
}
```

16 Appendix H - プログラムとスクリプト例

Code Listing 8 Cy_Prot_ConfigPpuProgMasterStructAttr ()

```
cy_en_prot_status_t Cy_Prot_ConfigPpuProgMasterStructAttr(volatile stc_PERI_MS_PPU_PR_t* base,
cy_en_prot_pc_t setPC, const cy_stc_ppu_prog_attr_cfg_t* config)
{
    cy_en_prot_status_t status = CY_PROT_SUCCESS;
    un_PERI_MS_PPU_PR_MS_ATT0_t tempMS_ATT0 = { 0 };
    un_PERI_MS_PPU_PR_MS_ATT1_t tempMS_ATT1 = { 0 };
    un_PERI_MS_PPU_PR_MS_ATT2_t tempMS_ATT2 = { 0 };
    un_PERI_MS_PPU_PR_MS_ATT3_t tempMS_ATT3 = { 0 };
:
   switch(setPC)
    {
:
    case CY_PROT_PC2:
        tempMS_ATT0.u32Register = base->unMS_ATT0.u32Register;
        tempMS_ATT0.stcField.u1PC2_UR = (config->userPermission & CY_PROT_PERM_R);
        tempMS_ATT0.stcField.u1PC2_UW = (config->userPermission & CY_PROT_PERM_W) >> 1;
        tempMS_ATT0.stcField.u1PC2_PR = (config->privPermission & CY_PROT_PERM_R);
        tempMS_ATT0.stcField.u1PC2_PW = (config->privPermission & CY_PROT_PERM_W) >> 1;
        tempMS_ATT0.stcField.u1PC2_NS = !(config->secure);
        base->unMS_ATT0.u32Register = tempMS_ATT0.u32Register;
:
        break;
:
    default:
        return CY_PROT_BAD_PARAM;
    }
    return status;
}
```

16 Appendix H - プログラムとスクリプト例

Code Listing 9 Cy_Prot_ConfigPpuFixedSlaveStruct ()

```
cy_en_prot_status_t Cy_Prot_ConfigPpuFixedSlaveStruct(volatile stc_PERI_MS_PPU_FX_t* base,
cy_en_prot_pc_t setPC, const cy_stc_ppu_gr_cfg_t* config)
{
    cy_en_prot_status_t status = CY_PROT_SUCCESS;
    un_PERI_MS_PPU_PR_SL_ATT0_t tempSL_ATT0 = { 0 };
    un_PERI_MS_PPU_PR_SL_ATT1_t tempSL_ATT1 = { 0 };
    un_PERI_MS_PPU_PR_SL_ATT2_t tempSL_ATT2 = { 0 };
    un_PERI_MS_PPU_PR_SL_ATT3_t tempSL_ATT3 = { 0 };
:
   switch(setPC)
    {
:
    case CY_PROT_PC2:
        tempSL_ATT0.u32Register = base->unSL_ATT0.u32Register;
        tempSL_ATT0.stcField.u1PC2_UR = (config->userPermission & CY_PROT_PERM_R);
        tempSL_ATT0.stcField.u1PC2_UW = (config->userPermission & CY_PROT_PERM_W) >> 1;
        tempSL_ATT0.stcField.u1PC2_PR = (config->privPermission & CY_PROT_PERM_R);
        tempSL_ATT0.stcField.u1PC2_PW = (config->privPermission & CY_PROT_PERM_W) >> 1;
        tempSL_ATT0.stcField.u1PC2_NS = !(config->secure);
        base->unSL_ATT0.u32Register = tempSL_ATT0.u32Register;
:
        break;:
    default:
        return CY_PROT_BAD_PARAM;
    }

    return status;
}
```

16 Appendix H - プログラムとスクリプト例

Code Listing 10 Cy_Prot_ConfigPpuFixedMasterStruct ()

```
cy_en_prot_status_t Cy_Prot_ConfigPpuFixedMasterStruct(volatile stc_PERI_MS_PPU_FX_t* base,
cy_en_prot_pc_t setPC, const cy_stc_ppu_gr_cfg_t* config)
{
    cy_en_prot_status_t status = CY_PROT_SUCCESS;
    un_PERI_MS_PPU_PR_MS_ATT0_t tempMS_ATT0 = { 0 };
    un_PERI_MS_PPU_PR_MS_ATT1_t tempMS_ATT1 = { 0 };
    un_PERI_MS_PPU_PR_MS_ATT2_t tempMS_ATT2 = { 0 };
    un_PERI_MS_PPU_PR_MS_ATT3_t tempMS_ATT3 = { 0 };
:
   switch(setPC)
    {
:
    case CY_PROT_PC2:
        tempMS_ATT0.u32Register = base->unMS_ATT0.u32Register;
        tempMS_ATT0.stcField.u1PC2_UR = (config->userPermission & CY_PROT_PERM_R);
        tempMS_ATT0.stcField.u1PC2_UW = (config->userPermission & CY_PROT_PERM_W) >> 1;
        tempMS_ATT0.stcField.u1PC2_PR = (config->privPermission & CY_PROT_PERM_R);
        tempMS_ATT0.stcField.u1PC2_PW = (config->privPermission & CY_PROT_PERM_W) >> 1;
        tempMS_ATT0.stcField.u1PC2_NS = !(config->secure);
        base->unMS_ATT0.u32Register = tempMS_ATT0.u32Register;
:
        break;
:
    default:
        return CY_PROT_BAD_PARAM;
    }

    return status;
}
```

これらのプログラムコードとスクリプトは以下にあります。

<user>\ tviibe1m\src\enhance_marker

<user>: サンプルプロジェクトが格納されたフォルダ

17 Appendix I - セカンドアプリケーションソフトウェア実装

DEAD 状態のデバイスは、RMA ライフサイクルに移行できません。ハードウェア障害など意図しないアプリケーションソフトウェア認証の失敗により DEAD 状態に遷移する場合があります。この場合、RMA ライフサイクルステージへの移行のみを許可するセカンドアプリケーションソフトウェアを準備することで解決できる場合があります。ここでは、セカンドアプリケーションソフトウェアの実装について説明します。

注: セカンドアプリケーションソフトウェアは NORMAL_PROVISIONED ライフサイクルステージで実装する必要があります。SECURE または SECURE_W_DEBUG ライフサイクルステージでは実装できません。

注: ブートローダの有効化条件が満たされた場合、ブートローダが起動されセカンドアプリケーションは有効化されません。ブートローダの有効化条件は、TOC2 を参照してください。

図 18 にフラッシュブートによるアプリケーションソフトウェア実行のフローチャートを示します。

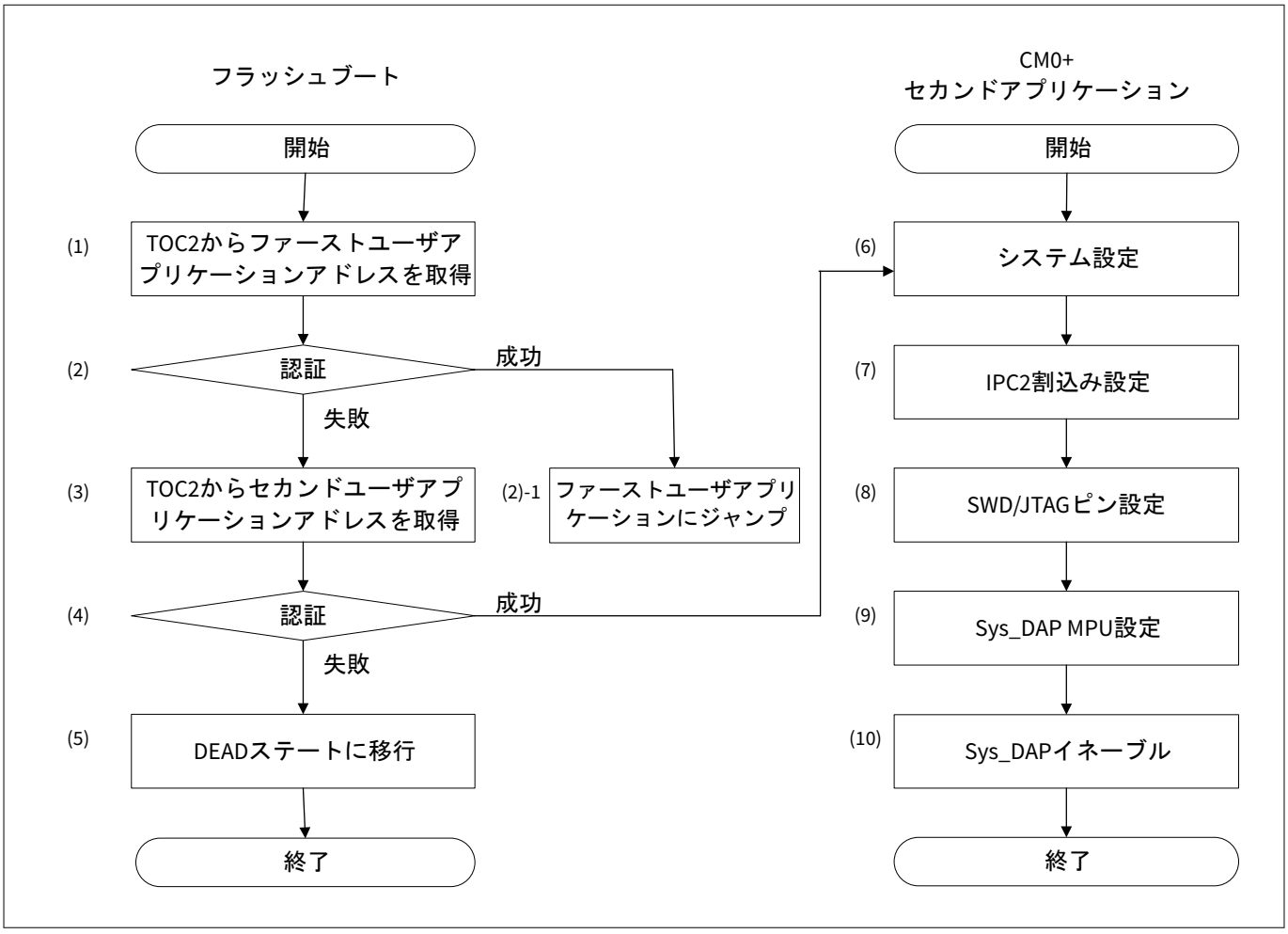


図 18 フラッシュブートのアプリケーション実行

1. フラッシュブートは、TOC2 からファーストユーザアプリケーションのアドレスを取得します (オフセット = 0x0c)。
2. ファーストユーザアプリケーションを認証します。認証が成功するとフラッシュブートはファーストユーザアプリケーションにジャンプします。
3. ファーストユーザアプリケーションの認証が失敗した場合、フラッシュブートはセカンドユーザアプリケーションソフトウェアのアドレスを取得します(オフセット = 0x14) 。認証の失敗には、ファーストユーザアプリケーションの全消去も含まれます。

4. セカンドユーザアプリケーションソフトウェアを認証します。認証が成功した場合、フラッシュブートはセカンドユーザアプリケーションにジャンプします。
5. 認証が失敗した場合、デバイスは DEAD 状態になります。この場合、デバイスは RMA ライフサイクルステージに移行できません。
6. セカンドユーザアプリケーションソフトウェアは、初期設定をします。必要に応じて、推奨されるクロック設定に変更します。詳細についてはアーキテクチャ TRM[2]の Blow Fuse Bit を参照してください。
7. セカンドユーザアプリケーションソフトウェアは、IPC2 と割込みコントローラを設定します。
8. セカンドユーザアプリケーションソフトウェアは、SWD/JTAG ピンを設定します。
9. セカンドユーザアプリケーションソフトウェアは、Sys_DAP MPU を設定します。
10. Sys_DAP をイネーブルにします。

17.1 実装

セカンドアプリケーション実装用のプロジェクトファイルを提供します。これは以下にあります。

<user>\tviibe1m\tools\ghs

<user>: サンプルプロジェクトが格納されたフォルダ

このサンプルプロジェクトの設定は以下のとおりです。

- ファーストユーザアプリケーションアドレス: 0x10000000
- ファーストユーザアプリケーションの署名のアドレス: 0x1000fe00
- セカンドユーザアプリケーションアドレス: 0x10010000
- セカンドユーザアプリケーションの署名のアドレス: 0x1001fe00

システムに応じてこれらのパラメータを変更してください。ここに示すサンプルプログラムとスクリプトは CYT2B7 シリーズを使用しています。

1. プロジェクトの設定
   a. プロジェクトウィンドウを開いてください。
   b. **tviibe1m_common.gpj** を右クリックしてください。
   c. **Add File into tviibe1m_common.gpj...**を選択してください。
   d. **_gpj_2nd** フォルダから cm0plus_flash_2nd.gpj を選択してください。

17 Appendix I - セカンドアプリケーションソフトウェア実装

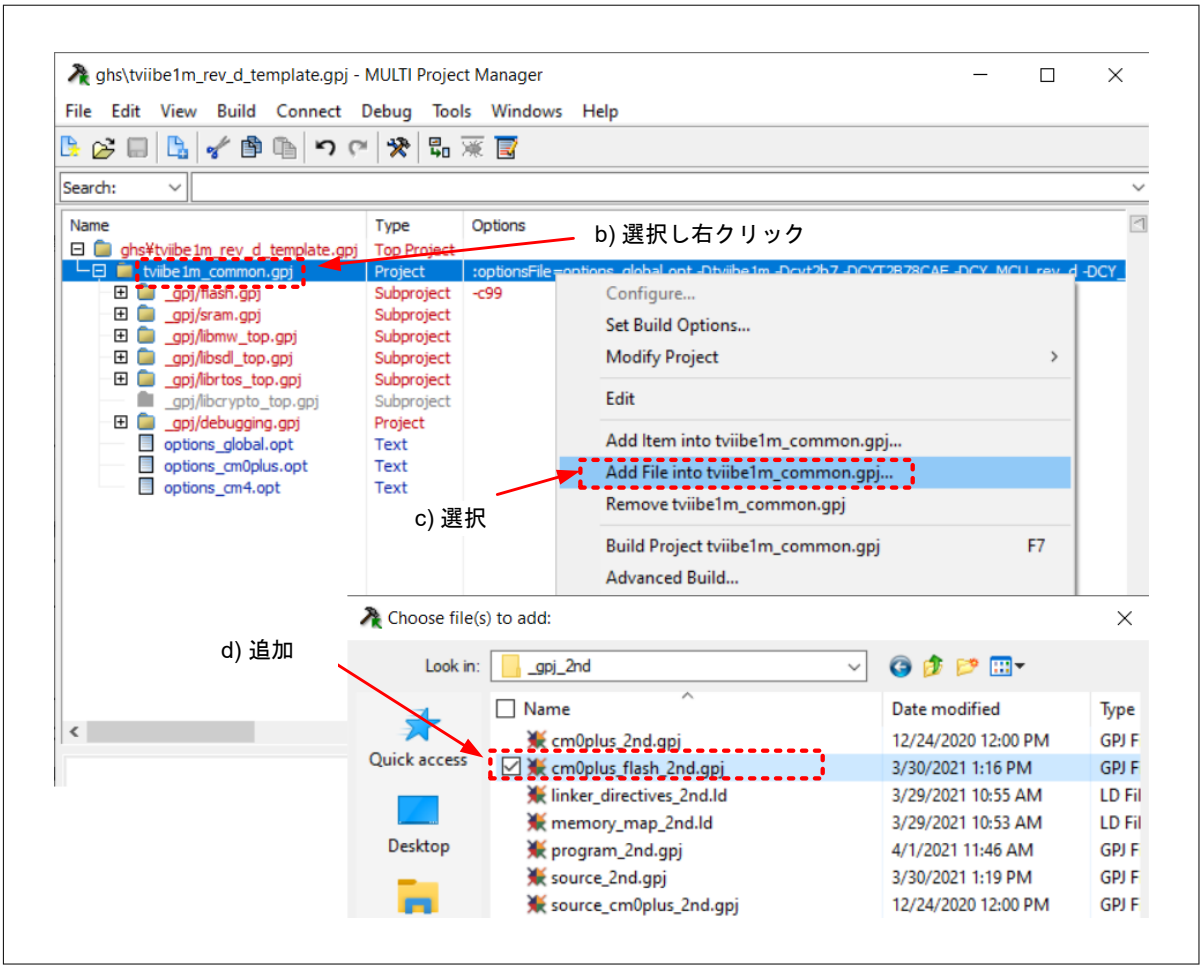


図 19　　セカンドアプリケーション用プロジェクト設定

2. ソースファイルの設定

a. <user>\tviibe1m\src\second_app フォルダから<user>\tviibe1m\src フォルダへ main_cm0plus_2nd.c ファイルをコピーしてください。

b. main_cm0plus.c を設定してください。

「.cm0pappAddr2」をセカンドアプリケーションソフトウェアの開始アドレス、「.cm0pappFormat2」を CySAF に設定してください。

```
/** TOC2 in SFlash */
CY_SECTION(".cy_toc_part2") __USED static const cy_stc_si_toc_t cy_toc2 =
{
:
    .cm0pappAddr2    = CY_SI_SECOND_APP_FLASH_BEGIN,    /* Offset+0x14: App2 (CM0+ Second
User App Object) addr */
    .cm0pappFormat2  = CY_SI_APP_FORMAT_CYPRESS,        /* Offset+0x18: App2 Format */
:
};
```

3. ビルドとプログラム

a. GHS Multi を使用して、フラッシュプロジェクトと cm0plus_flash_2nd プロジェクトをビルドしてください。

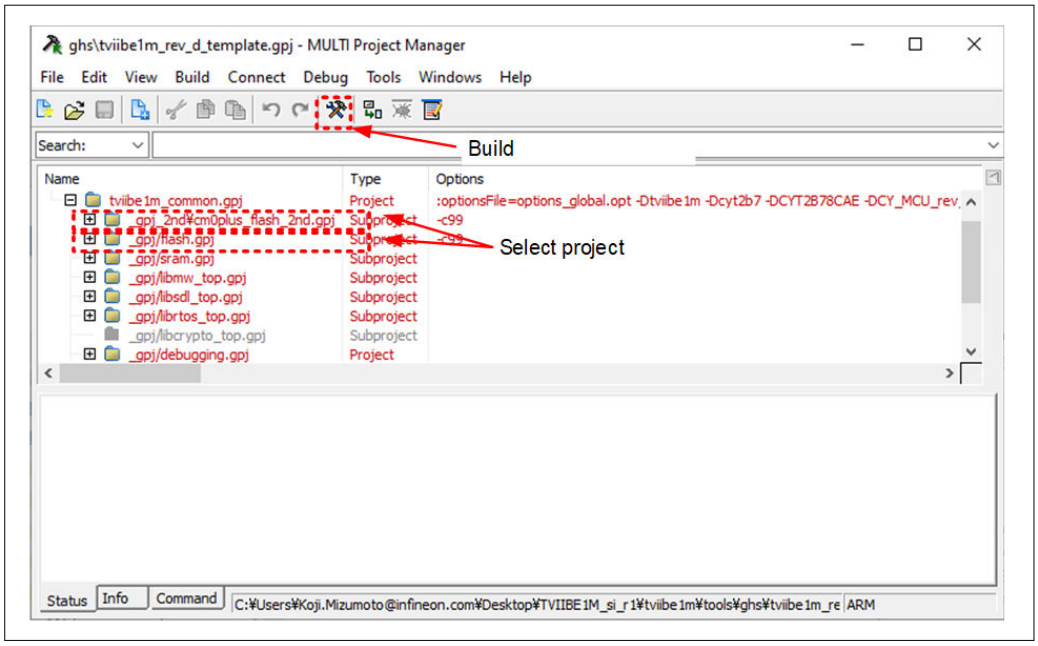


図 20　　セカンドアプリケーションプロジェクトのビルド

b. *Secure_Complete_SREC_to_CYP_withCM4and2ndapp_2K, _3K, または_4K.bat* スクリプトを実行してください。

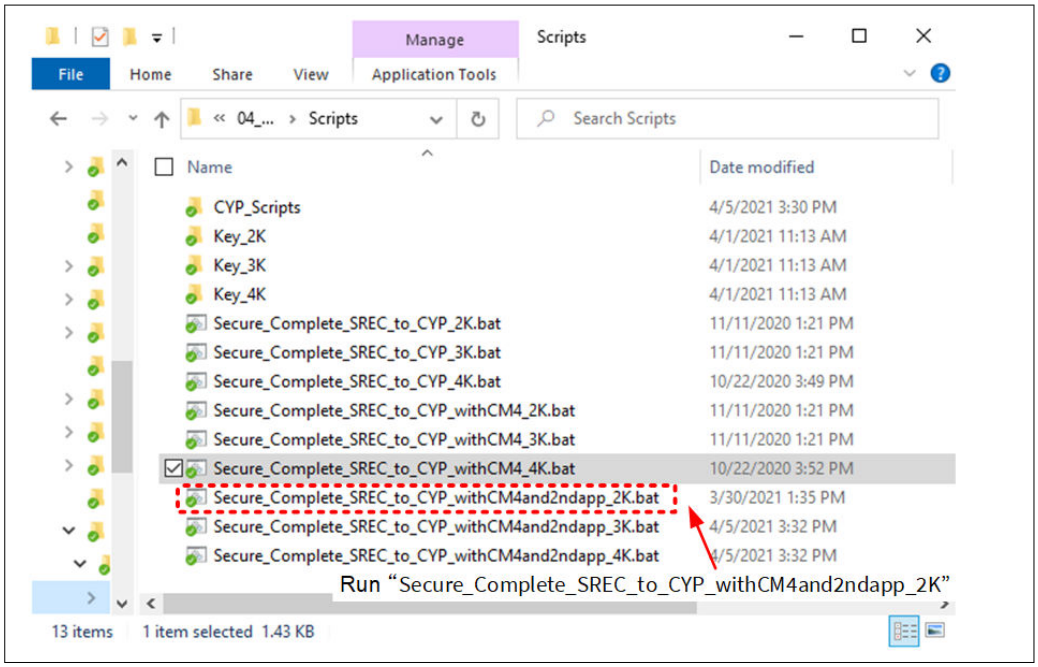


図 21　　Secure_Complete_SREC_to_CYP_with CM4and2ndapp_2K, 3K, または _4K.bat の実行

この *bat* ファイルは、以下を実行します。CM0+のファーストアプリケーション ELF ファイルおよびセカンドアプリケーション ELF ファイルに署名します。

- ファーストアプリケーションとセカンドアプリケーションの ELF ファイルをマージします。
- マージされた ELF ファイルから cm0plus_si.elf.srec ファイルを生成します。

- CM4 ELF ファイルに署名し、署名された CM4 ELF ファイルから cm4_si.elf.srec を生成します。
- SREC ファイルが `CYP_Scripts` にあることを確認します。

c. MiniProg4 をターゲットボードに接続し、**CYP_Scripts** フォルダの `enable_sflashandprogram.bat` を実行してください。

18 Appendix J – ダミーアプリケーションヘッダの実装

ブートローダの有効化条件が満たされた場合、デバイスはファーストアプリケーションソフトウェアを消去すると DEAD 状態に移行する場合があります。その結果、セカンドアプリケーションソフトウェアを実装しても起動されません。このような場合、ファーストアプリケーションソフトウェアの代わりにダミーアプリケーションヘッダをプログラムするか、ブートローダを無効にすることでセカンドアプリケーションを起動できます。ここでは、ダミーのアプリケーションヘッダの実装について説明します。

18.1 実装

ダミーアプリケーションヘッダ実装用のファイルを提供します。<user>\tviibe1m\tools\ghs

<user>: サンプルプロジェクトが格納されたフォルダ

このサンプルプロジェクトの設定は以下のとおりです。

- アプリケーションヘッダアドレス: 0x10000000
- オブジェクトサイズ: 0xFE00
- アプリケーション ID/バージョン: 0x00010000
- 属性: 0
- コア数(N): 1
- コア (i) Vt オフセット: 0xF0
- コア (i) CPU ID とコアインデックス: 0xC6000000

システムに応じてこれらのパラメータを変更してください。ここに示すサンプルプログラムとスクリプトは CYT2B7 シリーズを使用しています。

1. プロジェクトの設定
   a. プロジェクトウィンドウを開いてください。
   b. <user> \tviibe1m\tools\ghs_gpj_app_header フォルダから<user>\ tviibe1m\tools\ghs フォルダへ *tviibe1m_common_app_header.gpj* および *options_global_app_header.opt* ファイルをコピーしてください。

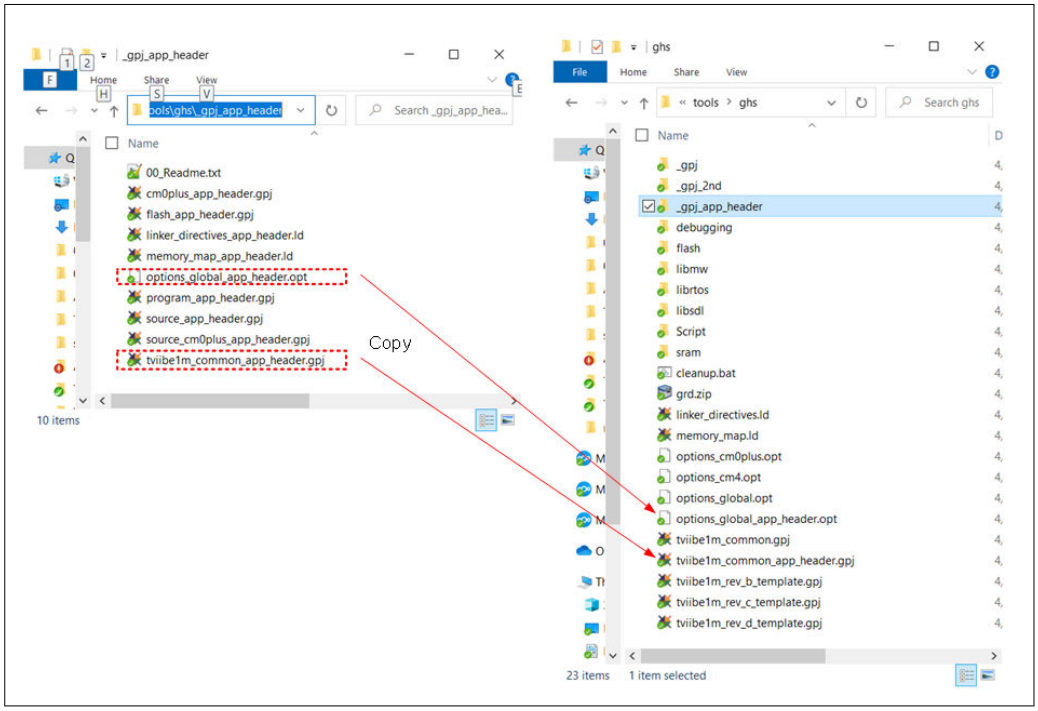


図 22 アプリケーションヘッダ用プロジェクトの設定 1

   c. **tviibe1m_rev_d_template.gpj** を右クリックしてください。

d. **Add File into tviibe1m_common.gpj…**を選択してください。

e. **Tviibe1m_common_app_header.gpj** を追加してください。

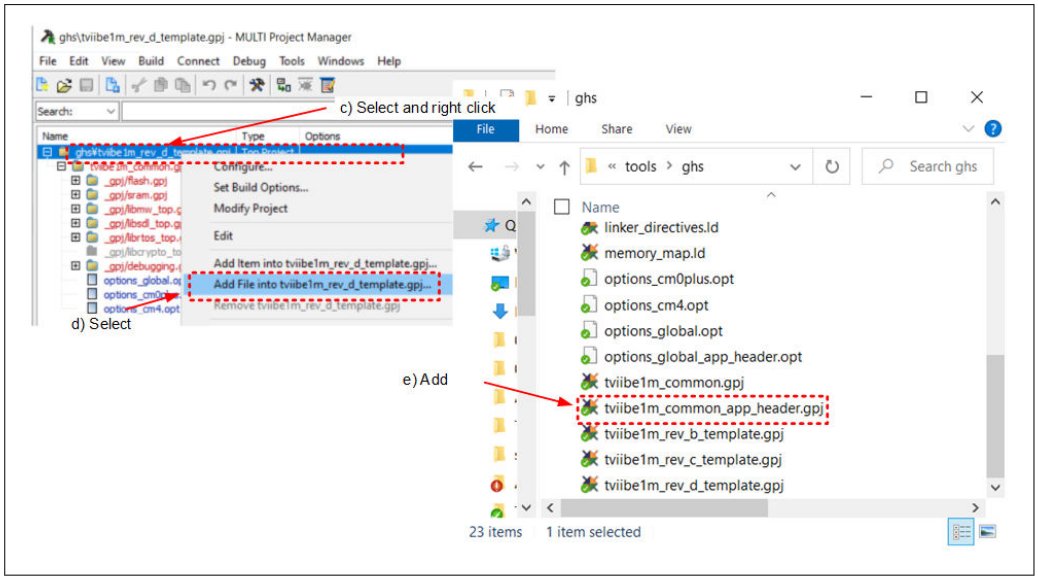


図 23　　アプリケーションヘッダ用プロジェクトの設定 2

2. ソースファイルの準備

a. `<user>\ tviibe1m\src\dummy_first_app_hdr` フォルダから`<user>\ tviibe1m\src` フォルダへ main_cm0plus_app_header.c ファイルをコピーしてください。

3. ビルドとプログラム

a. GHS Multi を使用して、flash_app_header プロジェクトをビルドしてください。

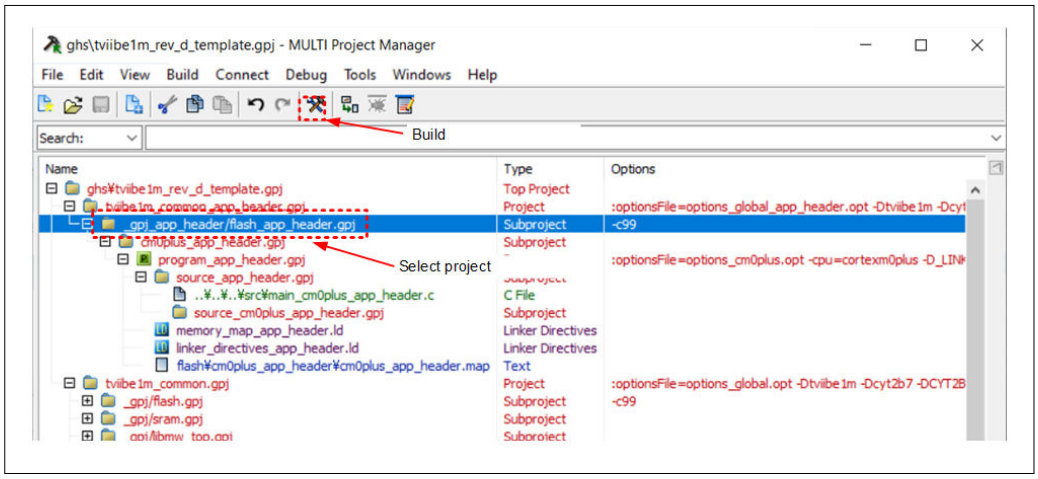


図 24　　アプリケーションヘッダプロジェクトのビルド

以下はアプリケーションヘッダの SREC ファイル例です。

```
S00F0000636D30706C75735F61700000FC
S3151000000000FE0000000001000000000001000000DA
S30D10000010F0000000000000C61C
S50500000002F8
S70500000000FA
```

b. MiniProg4 をターゲットボードに接続し、CYP_Scripts フォルダの *enable_appheaderprogram.bat* を実行してください。この bat ファイルは以下を実行します。

- cm0plus_app_header.elf.srec ファイルをコピーします。
- ファーストアプリケーションソフトウェア領域(0x10000000 から 0x1000FFFF)を消去します。
- cm0plus_app_header.elf.srec ファイルをプログラムします。

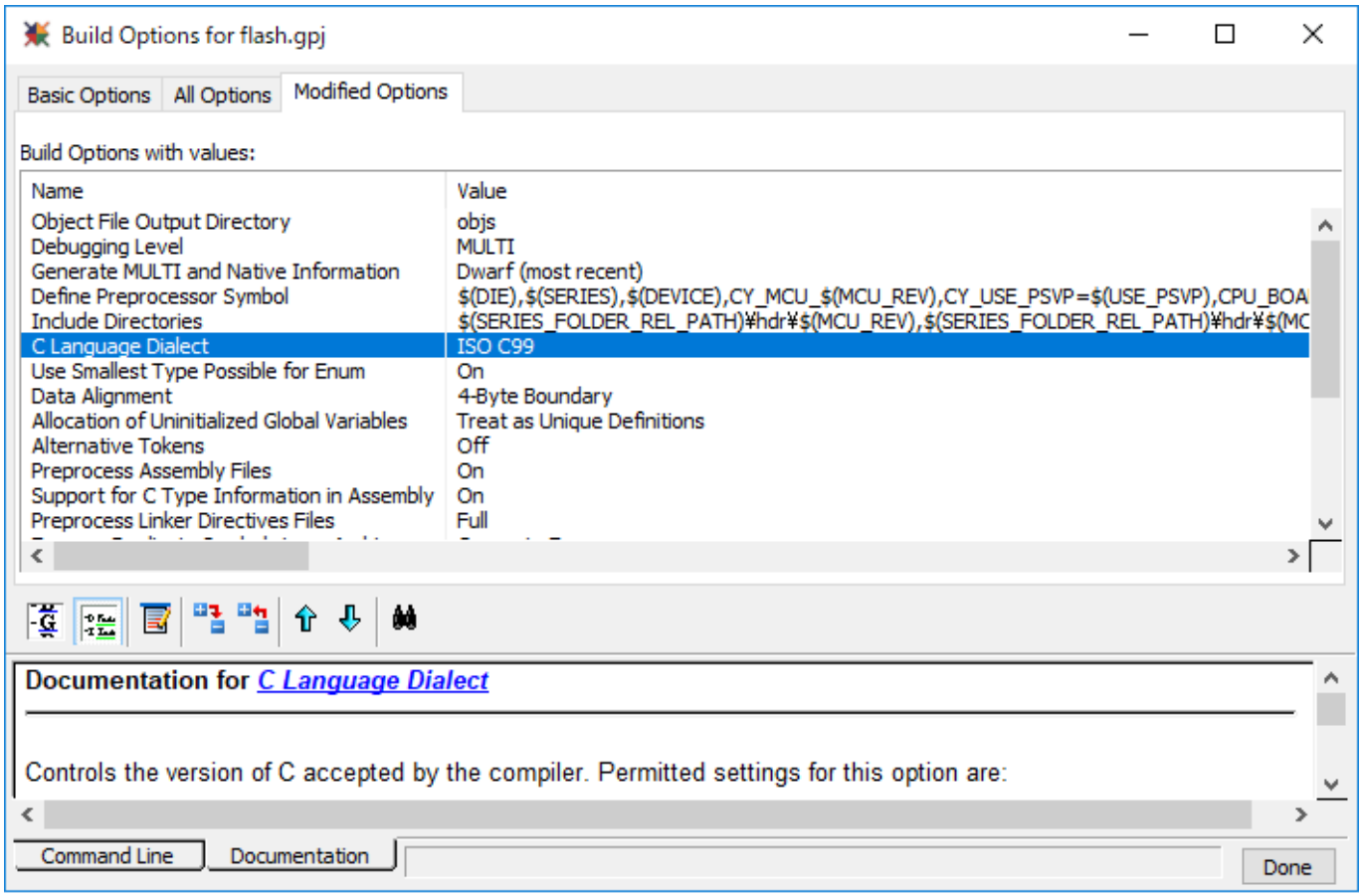
19 Appendix K - コンパイルおよびリンカオプション

Green Hills MULTI を使用しセキュアブートをビルドする場合、以下のオプションが必要です。

- コンパイルオプション: C Language Dialect を ISO C99 に設定

この設定は、サンプルプロジェクトを使用してセキュアイメージを設定するために必要です。

1. Target project を選択してください。
2. **Edit** > **Set Build Options....**を選択してください。

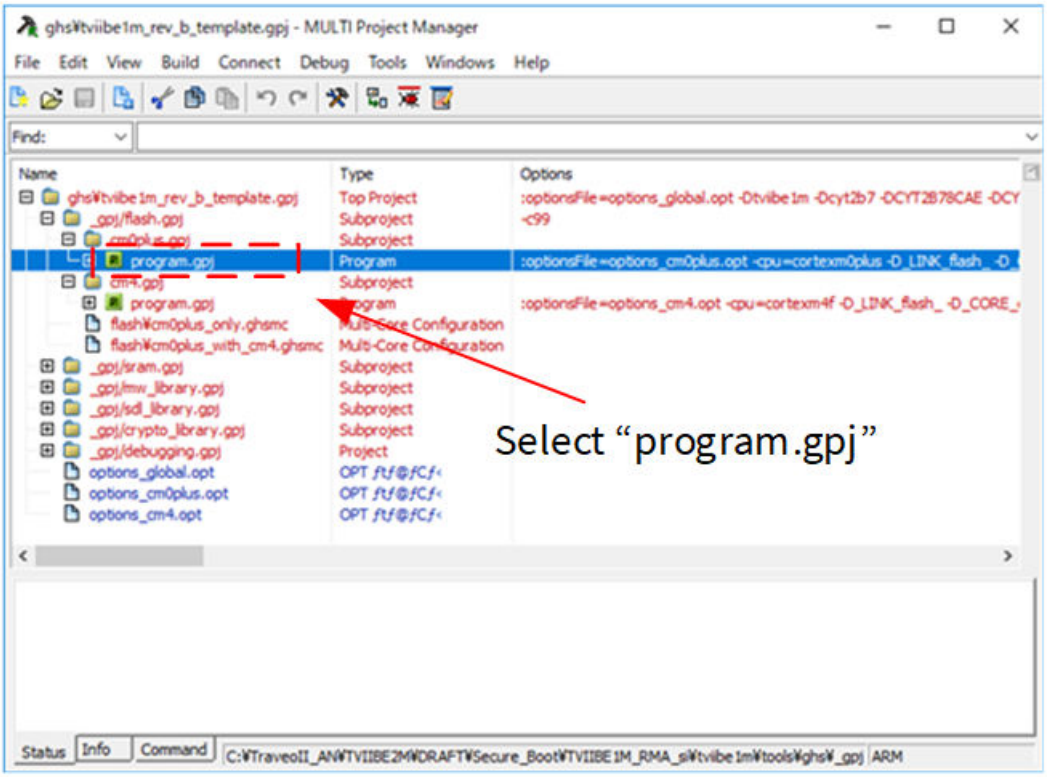


- リンカオプション: physical address offset from the Virtual Address を「0」に設定

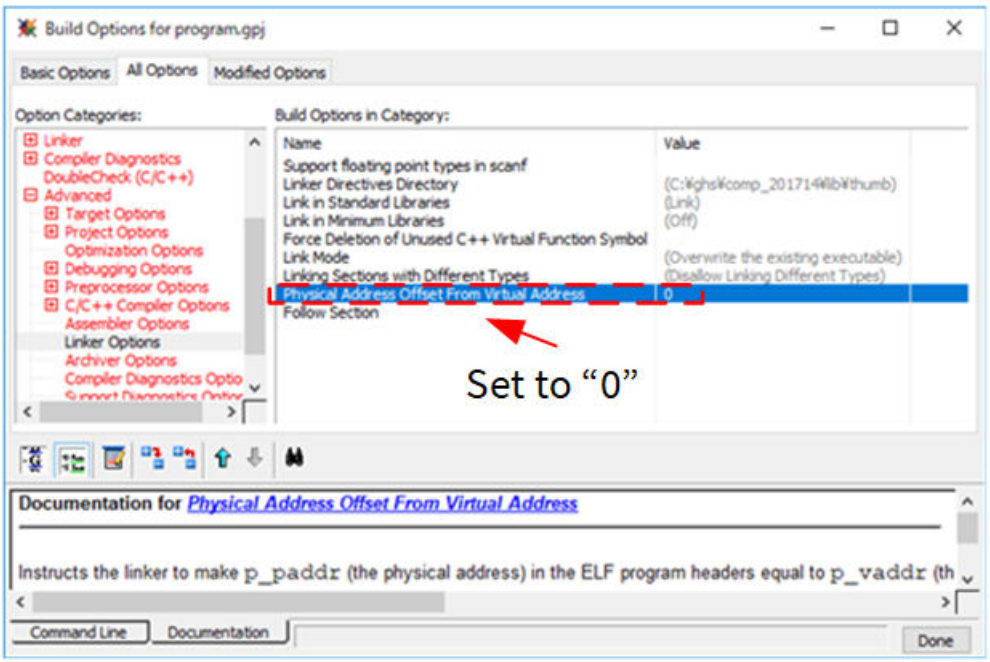
この設定は、cymcuelftool.exe ツールを使用してデジタル署名を生成するために必要です。

1. **cm0plus.gpj** の **program.gpj** を選択してください。

19 Appendix K - コンパイルおよびリンカオプション



2. **Edit** > **Set Build Options....**の **All Options** タブを選択してください。
3. **Option Categories** リストの **Advanced** グループを展開し、**Linker Options** を選択してください。



4. 同様に *cm4.gpj* の **program.gpj** を設定してください。

- SREC ファイルの生成

19 Appendix K - コンパイルおよびリンカオプション

前述のように、デジタル署名認証する場合、コードの空白領域は「0」でなければなりません。以下に、GHS MULTI の SREC ファイルの空白領域を「0」埋めする方法を示します。SREC ファイルは、gsrec ユーティリティを使用して生成されます。

```
> gsrec.exe input_file (*.elf) -o out_file (*.srec) -bytes 16 -fill1 0x10000000 0x1000FEFF 0x00
```

20 関連ドキュメント

以下は、TRAVEO™ T2G ファミリシリーズのデータシートとテクニカルリファレンスマニュアルです。このアプリケーション ノートで使用されているドキュメント、コード スニペット、サンプル ドライバ ライブラリ (SDL)、およびツールの入手についてはテクニカルサポートに連絡してください。なお、入手したサンプルプログラムは車載規格に準拠したものではなく、セキュアシステム構成の一例であり、製品目的として使用できません。

[1] デバイスデータシート

- CYT2B7 datasheet 32-bit Arm® Cortex®-M4F microcontroller TRAVEO™ T2G family
- CYT2B9 datasheet 32-bit Arm® Cortex®-M4F microcontroller TRAVEO™ T2G family
- CYT4BF datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family
- CYT4DN datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family (Doc No. 002-24601)
- CYT3BB/4BB datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family
- CYT3DL datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family (Doc No. 002-27763)

[2] テクニカルリファレンスマニュアル

- Body Controller Entry ファミリ
  - TRAVEO™ T2G Automotive Body Controller Entry family architecture technical reference manual (TRM)
  - TRAVEO™ T2G Automotive Body Controller Entry registers technical reference manual (TRM) for CYT2B7
  - TRAVEO™ T2G Automotive Body Controller Entry registers technical reference manual (TRM) for CYT2B9
- Body Controller High ファミリ
  - TRAVEO™ T2G Automotive Body Controller High family architecture technical reference manual (TRM)
  - TRAVEO™ T2G Automotive Body Controller High registers technical reference manual (TRM) for CYT4BF
  - TRAVEO™ T2G Automotive Body Controller High registers technical reference manual (TRM) for CYT3BB/4BB
- Cluster 2D ファミリ
  - TRAVEO™ T2G Automotive Cluster 2D family architecture technical reference manual (TRM) (Doc No. 002-25800)
  - TRAVEO™ T2G Automotive Cluster 2D registers technical reference manual (TRM) for CYT4DN (Doc No. 002-25923)
  - TRAVEO™ T2G Automotive Cluster 2D registers technical reference manual (TRM) for CYT3DL (Doc No. 002-29854)

改訂履歴

版数	発行日	変更内容
**	2020-05-20	これは英語版 002-28680 Rev. **を翻訳した日本語版 002-30209 Rev. **です。英語版の改訂内容: New Application Note.
英語版*A	-	この版は英語版のみです。英語版の改訂内容: 7.3 章を追加 RMA ライフサイクルステージへ遷移について注意事項を追加 RSA-3072/4096 を追加 Appendix G,H を変更 Appendix I,J を追加 MOVED TO INFINEON TEMPLATE.
*A	2021-11-10	これは英語版 002-28680 Rev. *B を翻訳した日本語版 002-30209 Rev. *Aです。英語版の改訂内容: ブートローダに関する記載を 8.1 章に追加 7.3 および 16.3 章を追加 TOC2 テーブルにセキュリティマーカを追加 CYT3BB/4BB および CYT3DL を追加
*B	2022-08-23	これは英語版 002-28680 Rev. *C を翻訳した日本語版 002-30209 Rev. *Bです。英語版の改訂内容: Added a note in RMA for Open RMA behavior in the SECURE_WITH_DEBUG lifecycle stage. Added a note in コード検証 for boot authentication of application software. Changed sample program folder location and file name.
英語版*D	-	この版は英語版のみです。英語版の改訂内容: Added note in セキュアシステムとは？
英語版*E	2023-11-16	この版は英語版のみです。英語版の改訂内容: Template update;no content update
*C	2024-08-13	これは英語版 002-28680 Rev. *F を翻訳した日本語版 002-30209 Rev. *Cです。英語版の改訂内容: Added protection condition of Normal Dead Access Restriction in 表 3, and 注 Added 注 about GHS probe version in Appendix A - 公開鍵と秘密鍵の作成例

Trademarks

All referenced product or service names and trademarks are the property of their respective owners.

Edition 2024-08-13

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about any aspect of this document?

Email: erratum@infineon.com

Document reference
IFX-efo1681902030894

重要事項

本手引書に記載された情報は、本製品の使用に関する手引きとして提供されるものであり、いかなる場合も、本製品における特定の機能性能や品質について保証するものではありません。本製品の使用の前に、当該手引書の受領者は実際の使用環境の下であらゆる本製品の機能及びその他本手引書に記された一切の技術的情報について確認する義務が有ります。インフィニオンテクノロジーズはここに当該手引書内で記される情報につき、第三者の知的所有権の不侵害の保証を含むがこれに限らず、あらゆる種類の一切の保証および責任を否定いたします。

本文書に含まれるデータは、技術的訓練を受けた従業員のみを対象としています。本製品の対象用途への適合性、およびこれら用途に関連して本文書に記載された製品情報の完全性についての評価は、お客様の技術部門の責任にて実施してください。

警告事項

技術的要件に伴い、製品には危険物質が含まれる可能性があります。当該種別の詳細については、インフィニオンの最寄りの営業所までお問い合わせください。

インフィニオンの正式代表者が署名した書面を通じ、インフィニオンによる明示の承認が存在する場合を除き、インフィニオンの製品は、当該製品の障害またはその使用に関する一切の結果が、合理的に人的傷害を招く恐れのある一切の用途に使用することはできないこと予めご了承ください。