参考資料

Application Note

MSPM0 ブートローダーの実装

Chao Gao

概要

このアプリケーション ノートは、MSPM0 デバイス向けブートローダー (BSL) のアプリケーション レベルでの説明を提供します。このドキュメントでは、MSPM0 の BSL に関するリソースを要約し、SDK に含まれる BSL のサンプルやツールの使用手順を順を追って説明します。ROM ベースの BSL の詳細については、*MSPM0 ブートローダー ユーザー ガイド*を参照してください。

目次

図の一覧

English Document: SLAAE88

表の一覧

商標

Code Composer Studio™ is a trademark of Texas Instruments.
すべての商標は、それぞれの所有者に帰属します。

1 はじめに

1.1 ブートローダの概念

マイコンのブートローダは、ユニバーサル非同期レシーバ/トランスミッタ (UART) や集積回路間 (I2C) などの一般的なインターフェイスを使用して MCU の内部メモリにプログラムするために利用できます。ブートローダは、デバイスのライフサイクル全体を通して迅速かつ容易にプログラミングを可能にします。BSL によるファームウェア更新の構造を図 1-1 に示します。図 1-1 に基づき、新しいファームウェアは、UART や I2C などのインターフェイスを備えた PC またはプロセッサをホストとする BSL を介して MSPM0 デバイスにダウンロードできます。

このアプリケーション ノートでは、プログラムされる MCU をターゲット、更新を実行するデバイスまたはツールをホストと呼びます。

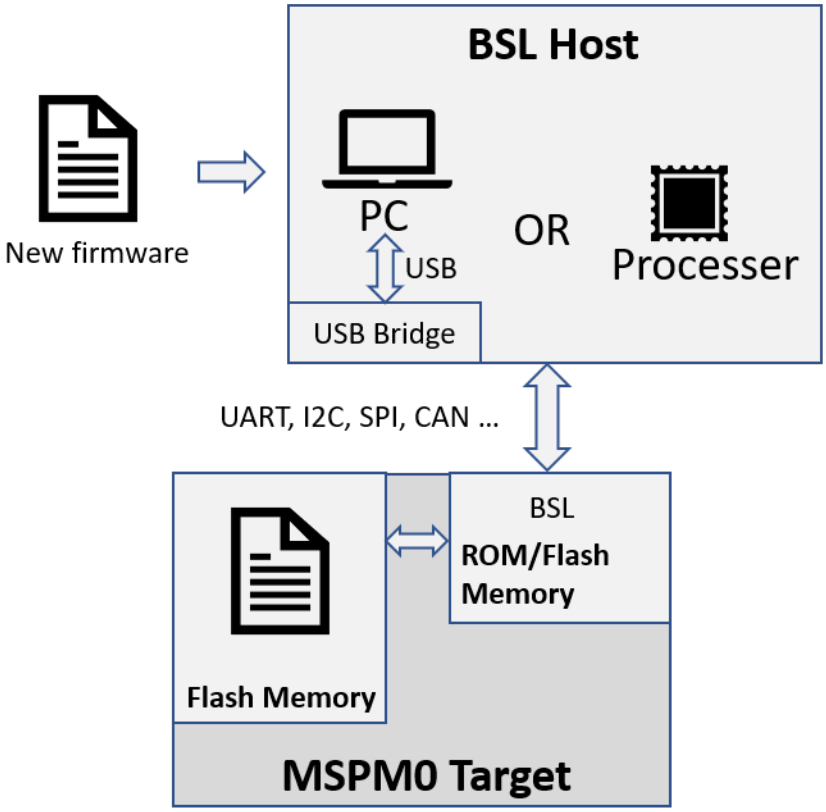

図 1-1. BSL を使用するファームウェア更新の構造

1.2 MSPM0 ブートローダの構造

MSPM0 デバイスには、3 種類のブートローダ設計が用意されています。ROM ベースの BSL、フラッシュベースのプラグイン インターフェイス付き ROM ベースの BSL、フラッシュベースのセカンダリ BSL。アプリケーションの要件に基づいて、3 種類のデザインのいずれかを選択するだけです。両方の設計は、汎用入出力 (GPIO) 起動、未書き込みデバイス検出、ソフトウェア起動といった同じ起動モードを使用します。NON-MAIN フラッシュで設定が必要なパラメータが存在します。詳細については、セクション 3 を参照してください。

表 1-1. MSPM0L と MSPM0G の BSL 設計の概要

BSL Designs リファレンス デザイン	ROM コスト	フラッシュ コスト (デフォルト)	インターフェイス	ハードウェア呼び出しとともに使用されるピン	ソフトウェア起動で使用されるピン	使用事例
ROM ベースの BSL	5K	該当なし	UART	4	2	TI のプロトコルおよび UART/I2C の設定に従う必要があります。
			I2C	4	2	
プラグイン インターフェイス付き ROM ベースの BSL	5K (BSL コア セクションのみを使用)	~ 1.6K	UART	4	2	TI のプロトコルには従う必要がありますが、インターフェイス レベルは完全にオープンソースです。
		~ 1.3K	I2C	4	2	
		~ 1.6K	SPI	6	4	
		~ 6K	CAN	4	2	
フラッシュベースのセカンダリ BSL	該当なし	~ 5K	UART	4	2	完全にオープンソースです。
		~ 5K	I2C	4	2	
		~ 5K	SPI	6	4	
		~ 9K	CAN	4	2	

注

ハードウェア起動はソフトウェア起動より 2 本多くのピンを必要とし、そのピンはリセットピンと GPIO 起動ピンです。

表 1-2. MSPM0C/H 設計の概要

BSL Design リファレンス デザイン	フラッシュ コスト	インターフェイス	ハードウェア呼び出しとともに使用されるピン	ソフトウェア起動で使用されるピン	使用事例
フラッシュベースの BSL	~ 4K	UART	4	2	完全にオープンソースです。
	~ 4K	I2C	4	2	

注

フラッシュコストは、次のような機能が制限されています: 一括消去、デバイス ID の取得、プログラム。他の機能は、コードで有効にできます。

図 1-2 に、MSPM0 での BSL の構造が示されています。

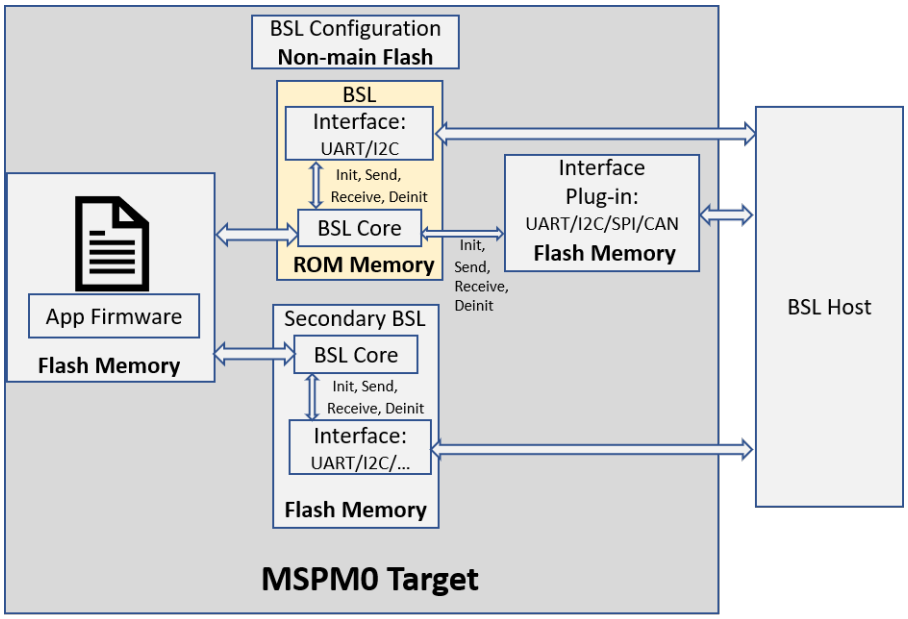

図 1-2. MSPM0 の BSL 構造

1.2.1 ROM ベースの BSL

MSPM0 L&G デバイスは、UART と I2C をサポートする高度にカスタマイズ可能な ROM ベースのブートローダを搭載して出荷されています。

ROM ベースの BSL は、BSL コアとインターフェイスが搭載されています。ホストとターゲット間でデータ パケットを送受信するために使用されるインターフェイス。BSL コアは、プロトコルに基づいてインターフェイスから送られてくるパケット データを解釈するために使用されます。一部のパラメータは Non-main フラッシュで構成でき、例えば BSL パスワードや UART/I2C ピンの割り当てなどがあります。

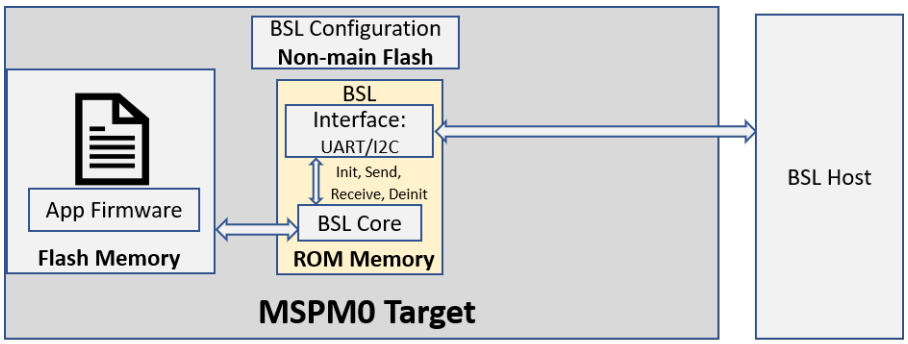

図 1-3. ROM ベースの BSL 構造

1.2.2 フラッシュ ベースのプラグイン インターフェイスを備えた ROM ベースの BSL

ROM ベースの通信インターフェイス (UART/I2C) がアプリケーションに適合しない場合、UART、I2C、CAN、シリアル ペリフェラル インターフェース (SPI) などのフラッシュベースのインターフェイス プラグイン デモがあり、これらは完全にオープンソースで必要に応じて変更できます。プラグイン インターフェイス デモは、パケットを解釈するために ROM ベースの BSL コアを共有しいるため、フラッシュ メモリを節約できます。

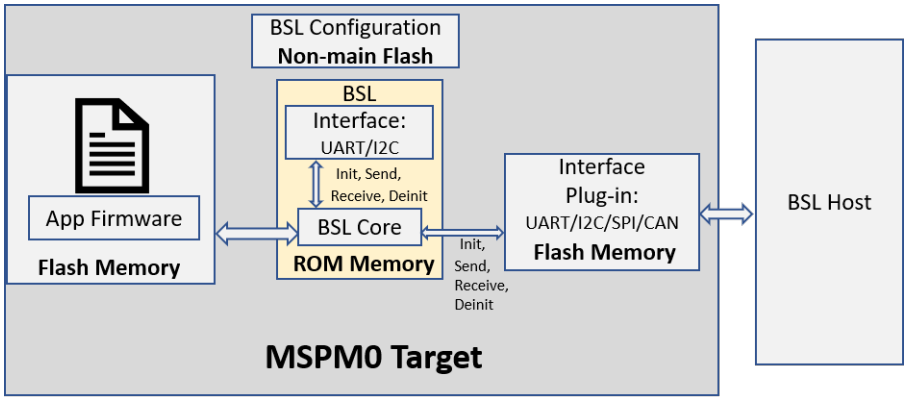

図 1-4. フラッシュ ベースのプラグイン インターフェイス構造を採用した ROM ベースの BSL

1.2.3 フラッシュベースのセカンダリ BSL

プライベート プロトコルが必要な場合、ROM ベースの BSL コアはこれ以上使用できず、セカンダリ BSL デモを参照できます。プロトコルを容易に変更できる、完全にオープンソース化されたセカンダリ BSL デモが SDK に用意されています。セカンダリ BSL デモのデフォルト プロトコルは、ROM ベースの BSL で同じです。

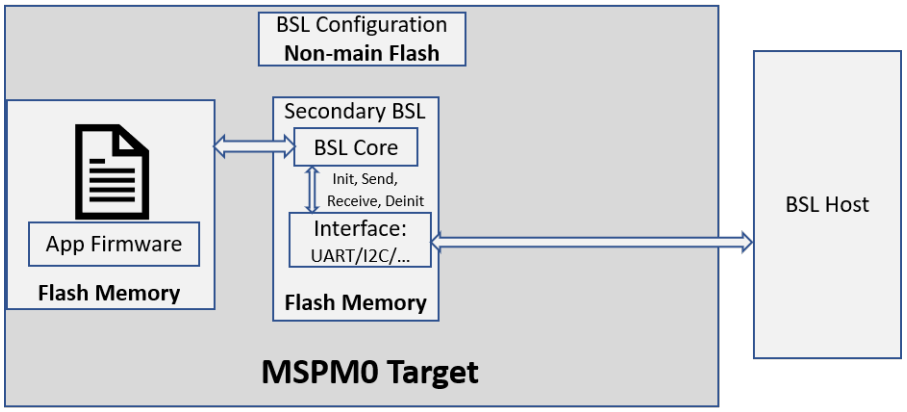

図 1-5. フラッシュベースのセカンダリ BSL の構造

ここでは、図 1-6 に示すように、2 種類のセカンダリ BSL デモも説明しています。MSPM0G3507 を例として使用します:

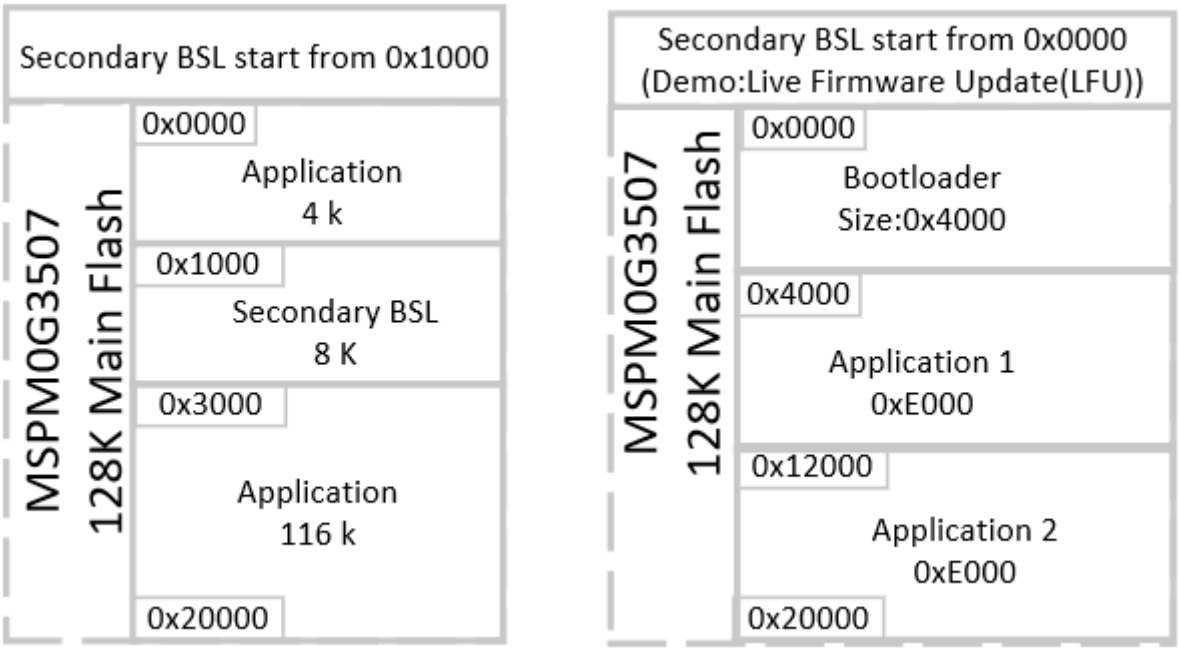

図 1-6. セカンダリ BSL 設計

- セカンダリ BSL を 0x1000 から開始する場合、0x0 を除いてフラッシュ内の任意の場所に配置できます。アプリケーション コードは 0x0 アドレスから開始する必要があるためです。この条件では、デバイスが起動またはリセットされたときに、デバイスはブートコード内の BSL 起動条件を確認し、アプリケーション コードを実行するか BSL コードを実行するかを決定します。このデモでは、ROM ベースの BSL のトリガ リソースを再利用しました。(ハードウェア、ソフトウェア、および未書き込みデバイスの検出。詳細については *MSPM0 ブートローダ ユーザ ガイド*のセクション 3.2 を参照してください)。図 1-7 に、セカンダリ BSL デモの実行フローを示します。

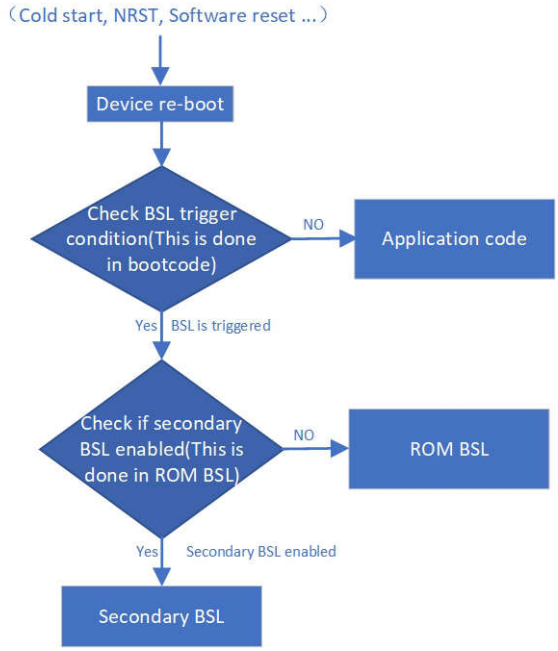

図 1-7. セカンダリ BSL 実行フロー

- 0 アドレスから開始するセカンダリ BSL の場合、MCU は起動またはリセットのたびにセカンダリ BSL を実行します。セカンダリ BSL では、カスタムチェック判定を使用して、ファームウェア更新のために BSL に留まるか、アプリケーションに移行するかを決定します。この設計の利点は、顧客が GPIO や未書き込みデバイス検出に限定されない特別な判定を使用できる点です。例えば、アプリケーション コードにジャンプする前にアプリケーションの CRC を確認し、アプリケーション コードの整合性を確保する必要があります。もう一つのユースケースは、MSPM0C のように ROM BSL を持たない一部の MSPM0 デバイス向けであり、この点については SDK にデモ コードが用意されています。このデモでは、起動条件を確認した後、アプリケーションにジャンプする必要がある場合、PC をアプリケーションの開始アドレスに設定できます。詳細については、セクション 5.3.2 を参照してください。セカンダリ BSL で FreeRTOS を使用し、ライブ ファームウェア更新を実現できるデモもあります。このデモは、アプリケーション コードを停止せずに、セカンダリ BSL ファームウェア アップデートが実行中であることを意味します。詳細については、*MSPM0 ライブ ファームウェア アップデート (LFU) ブート ローダの実装*を参照してください。

1.3 MSPM0 BSL の機能とデモの概要

表 1-3. MSPM0 BSL 機能の概要

デバイス ファミリ		M0C1104/3	M0C1106/5 M0H3216/5	MSPM0L	MSPM0G
BSL 全般	BSL メモリ タイプ	フラッシュ	フラッシュ	ROM	ROM
	BSL メモリのサイズ	3.5K 超過	3.9K 超過	5K	5K
	メイン以外のフラッシュでのユーザー構成	✓	✓	✓	✓
	UART	✓	✓	✓	✓
	I2C	✓	✓	✓	✓
プラグイン インターフェイスのデモ	UART			✓	✓
	I2C			✓	✓
	SPI			✓	✓
	CAN				✓
BSL 起動	GPIO 起動	✓	✓	✓	✓
	ブランク デバイスの検出	✓	✓	✓	✓
	ソフトウェア起動	✓	✓	✓	✓
	ROM サポートにおける起動検出		✓	✓	✓
ハードウェア ツール	UART 搭載 XDS110	✓	✓	✓	✓
ソフトウェア ツール	SDK 内の MSPM0_BSL_GUI	✓	✓	✓	✓
	Uniflash			✓	✓
セキュリティ	256 ビットのパスワード保護	✓	✓	✓	✓

SDK にはいくつかの BSL コード例があり、表 1-4 に示すように要約できます。

表 1-4. MSPM0 BSL デモの概要

	デモ タイプ	デモ名	使用事例
ターゲット側のデモ	プラグイン インターフェイスのデモ	bsl_spi_flash_interface	ROM ベースの通信インターフェイス構成や種類が要件を満たさない場合 (インターフェイスとして UART1 モジュールが必要、または SPI が必要な場合)、TI のデフォルト BSL プロトコルを使用できます
		bsl_uart_flash_interface	
		bsl_i2c_flash_interface	
		bsl_can_flash_interface	
	セカンダリ BSL デモ	secondary_bsl (uart/i2c/spi/can) flash_bsl (MSPM0C 用)	TI のデフォルト BSL プロトコルが要件を満たせない場合、MSPM0C 用の flash_bsl デモを除き、ROM ベース BSL と同じトリガ条件が再利用されます。
	アプリケーションのデモ	bsl_software_invoke_app_demo (uart/i2c/spi/can)	アプリケーションのサンプル コードは、ROM ベースの BSL、フラッシュベースのセカンダリ BSL デモ、またはフラッシュベースのインターフェイス プラグイン デモと連携して動作でき、ソフトウェア トリガ機能も含まれています。
ホスト側のデモ	ホストとしての MCU またはプロセッサ	bsl_host_mcu_to_m0x_target (uart/i2c/spi/can)	MCU またはプロセッサをホストとして使用し、TI のデフォルト BSL プロトコルに従う場合。これは、ROM BSL およびデフォルトのセカンダリ BSL デモとともに使用できます。
	PC をホストとして使用	MSPM0_BSL_GUI/Uniflash	PC を UART ホストとして使用し、TI のデフォルト BSL プロトコルに従う場合。つまり、これは ROM ベースの UART BSL、デフォルトの UART プラグイン インターフェイス デモ、またはデフォルトのセカンダリ BSL UART デモに使用できます。

表 1-5. MSPM0 BSL Demos Co-Work MCU をホストとして使用

ターゲット側				ホスト側
	メモリ位置	BSL コード デモ	アプリケーション コードのデモ	MCU/プロセッサ ホスト
ROM BSL	ROM	/	bsl_software_invoke_app_demo (uart/i2c/spi/can)	bsl_host_mcu_to_m0x_target (uart/i2c/spi/can)
プラグイン インターフェイスのデモ	メイン フラッシュ (ROM BSL との連携が必要)	bsl_spi_flash_interface		
		bsl_uart_flash_interface		
		bsl_i2c_flash_interface		
		bsl_can_flash_interface		
セカンダリ BSL デモ	メイン フラッシュ	secondary_bsl (uart/i2c/spi/can)		

表 1-6. MSPM0 BSL Demos Co-Work-PC をホストとして使用し

ターゲット側				ホスト側
	メモリ位置	BSL コード デモ	アプリケーション コードのデモ	PC ホスト
ROM BSL	ROM	/	bsl_software_invoke_app_demo (uart/i2c/spi/can)	MSPM0_BSL_GUI/Uniflash
プラグイン インターフェイスのデモ	メイン フラッシュ (ROM BSL との連携が必要)	bsl_spi_flash_interface		該当なし
		bsl_uart_flash_interface		MSPM0_BSL_GUI/Uniflash
		bsl_i2c_flash_interface		該当なし
		bsl_can_flash_interface		該当なし
セカンダリ BSL デモ	メイン フラッシュ	secondary_bsl (uart/i2c/spi/can)		該当なし

2 BSL ホストの実装の概要

本アプリケーション ノートでは、2 種類のホストの実装について説明しています。1 つは XDS110 などのインターフェース ブリッジを備えた PC、もう 1 つは MCU またはプロセッサです。図 2-1 に、信号接続図を示します。

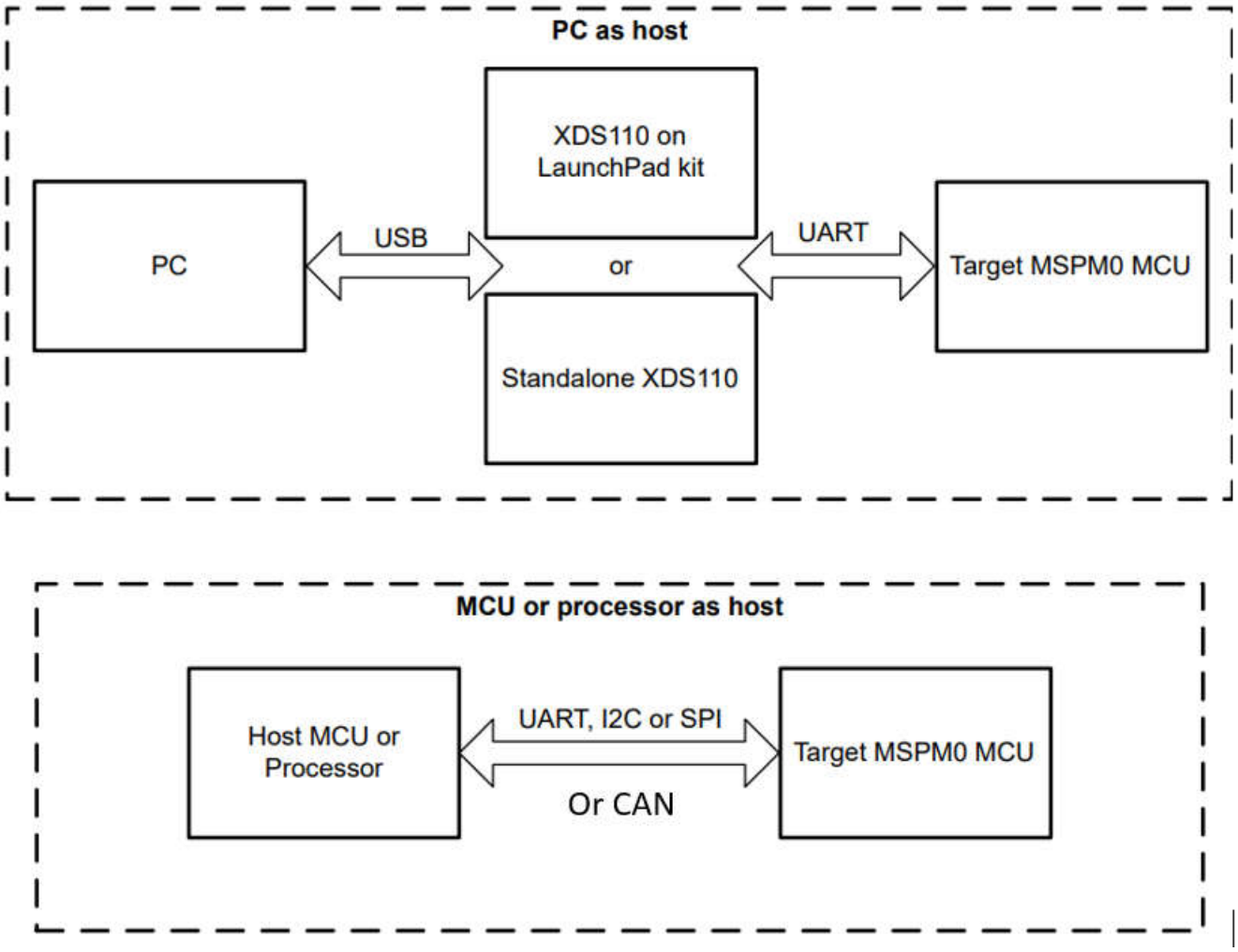

図 2-1. BSL ファームウェア アップデート システム ブロック図

PC をホストとして使用する場合、ダウンロード処理を行うために python 3 をベースに開発された GUI があります。事前にビルドされた Windows 実行ファイル (Win10 64 ビットでテスト済み) が含まれており、GUI のソースコードも SDK に含まれています。Uniflash は PC 側でも使用できます。

MCU またはプロセッサをホストとして使用する場合、MSPM0 をホスト MCU として動作させ、別の MSPM0 デバイスのファームウェアを更新するためのデモがいくつか用意されています。

3 Non-Main (構成 NVM) における BSL 構成

3.1 Non-Main の紹介

MSPM0 デバイスには 3 種類のフラッシュ メモリがあります。

表 3-1. フラッシュ メモリ領域

フラッシュ メモリ領域	リージョンの内容	実行可能ファイル	使用者	プログラム実行者
FACTORY	デバイス ID およびその他のパラメータ	なし	アプリケーション	TI のみ (変更不可)
NON-MAIN	デバイス ブート構成 (BCR および BSL)	なし	ブート ROM	TI、ユーザー
MAIN	アプリケーション コードおよびデータ	あり	アプリケーション	ユーザー

NON-MAIN は、デバイスをブートするために BCR および BSL が使用する設定データを格納する、フラッシュメモリの専用領域です。この領域は、他の目的では使用されません。BCR と BSL にはそれぞれ設定ポリシーがあり、開発や評価時には通常デフォルト値のままにしておくことができますが、量産プログラミング時には特定の目的に応じて NON-MAIN フラッシュ領域に書き込まれた値を変更することで修正できます。MSPM0C シリーズは ROM ベースの BSL を搭載していないため、MSPM0C シリーズ デバイスの NON-MAIN 内には BSL 関連の設定項目は含まれていません。

表 3-2. NON-MAIN 領域の概要

NON-MAIN セクション	開始アドレス	終了アドレス
BCR の構成	41C0.0000h	41C0.005Bh
BCR の構成 CRC	41C0.005Ch	41C0.005Fh
BSL の構成	41C0.0100h	41C0.0153h
BSL の構成 CRC	41C0.0154h	41C0.0157h

メインの BSL パラメータは、表 3-3 で設定できます。ファミリごとに、Non-Main フラッシュの内容は異なります。詳細については TRM を参照してください。

表 3-3. NON-MAIN フラッシュ BSL 構成の主なパラメータ

パラメータの使用例	パラメータ	説明
コモン	BSLCONFIGID	BSL 構成 ID
	BSLPW	256 ビット BSL アクセス パスワード。(セカンダリ BSL 用は任意)
	BSLCONFIG0	BSL 呼び出しピンの構成とメモリ読み出しポリシー。(メモリ読み出しポリシーはセカンダリ BSL では任意)
	BSLAPPVER	アプリケーション バージョン ワードのアドレス。
	BSLCONFIG1	BSL セキュリティ構成。(セカンダリ BSL では任意)
	BSLCRC	NON-MAIN メモリ内の BSL_CONFIG 部分の CRC ダイジェスト (CRC-32)。
ROM ベースの BSL	BSLPINCFG0	BSL UART ピンの構成
	BSLPINCFG1	BSL I2C ピン構成
フラッシュ ベースのプラグイン インターフェイスを備えた ROM ベースの BSL	BSLPLUGINCFG	MAIN フラッシュ メモリ内における BSL プラグインの有無と種類を定義します。
	BSLPLUGINHOOK	プラグインの初期化、受信、送信、終了処理関数の関数ポインタ
フラッシュベースのセカンダリ BSL	PATCHHOOKID	代替 BSL 構成
	SBLADDRESS	代替 BSL のアドレス。

NON-MAIN フラッシュの詳細については、*MSPM0 L シリーズ 32 MHz マイクロコントローラ 技術リファレンス マニュアル*、または *MSPM0 G シリーズ 80 MHz マイクロコントローラ 技術リファレンス マニュアル*を参照してください。

3.2 例 – Sysconfig で PA18 BSL 起動ピンを無効化

NON-MAIN 構成は Sysconfig で行うことができます。以下に、NON-MAIN フラッシュで PA18 の BSL 呼び出し機能を無効化する例を示します。なお、PA18 はデフォルト設定では NON-MAIN にて BSL 呼び出し用に割り当てられています。アプリケーションで PA18 を BSL 呼び出しに使用しない場合、このピンをプルダウンするか、NON-MAIN で BSL 呼び出し機能を無効化して、電源投入時やリセット時にデバイスが BSL モードに入るのを防ぐ必要があります。

1. Sysconfig を開き、NVM の設定を追加すると、NON-MAIN フラッシュを有効化するリスクを知らせるエラーが表示されます。ステップ 2 で設定リスクを承認すると、エラーを取り除くことができます。

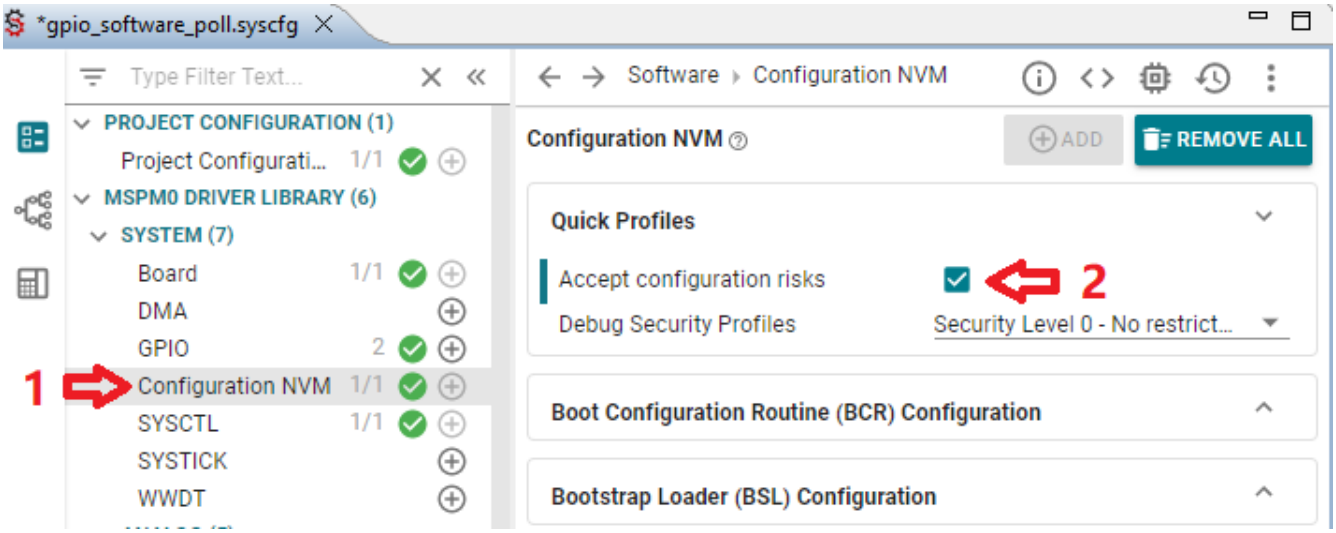

図 3-1. PA18 BSL 起動ピンを無効化するステップ 1

2. 図 3-2 に示されている PA18 BSL 起動関数をディセーブルにするか、図 3-3 に示されている別の BSL 起動ピンを選択します。

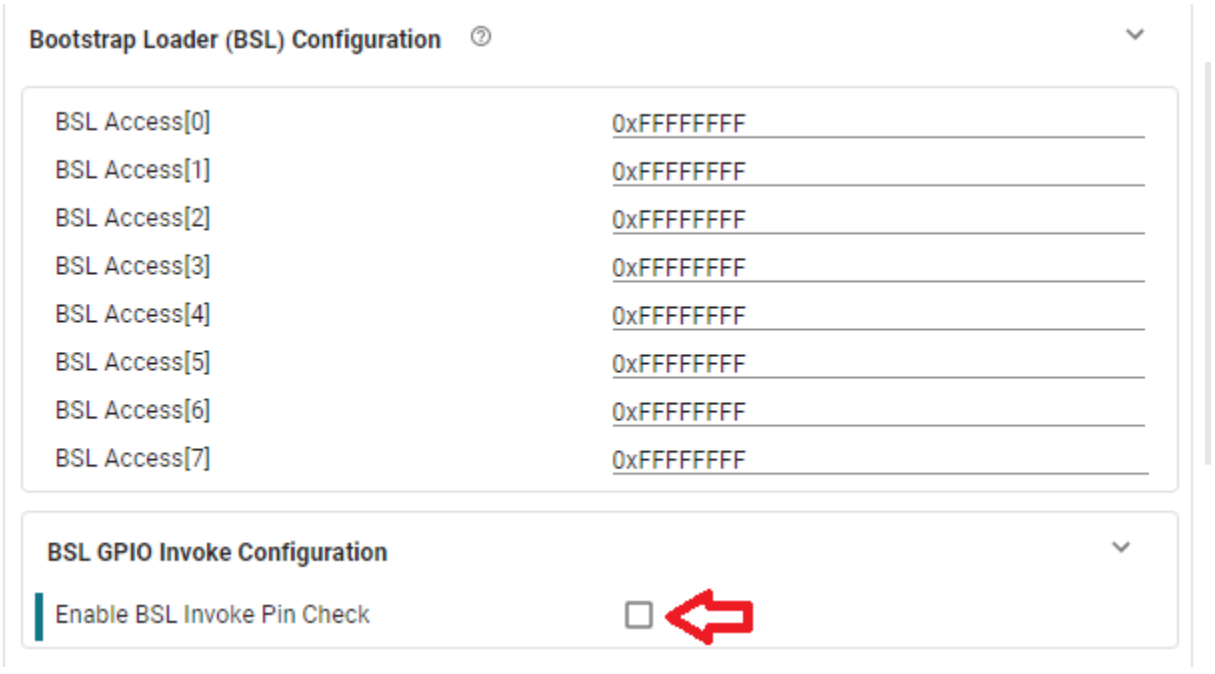

図 3-2. PA18 BSL 起動関数を無効化

図 3-3. BSL 起動として他のピンを選択し

3. Code Composer Studio™ (CCS)、または Keil でプロジェクトをビルドし、その後コードをフラッシュにダウンロードします。画像をダウンロードするために重要なことは、メイン以外のフラッシュ消去を有効にすることです。例えば CCS では、図 3-4 でこれを有効にします。

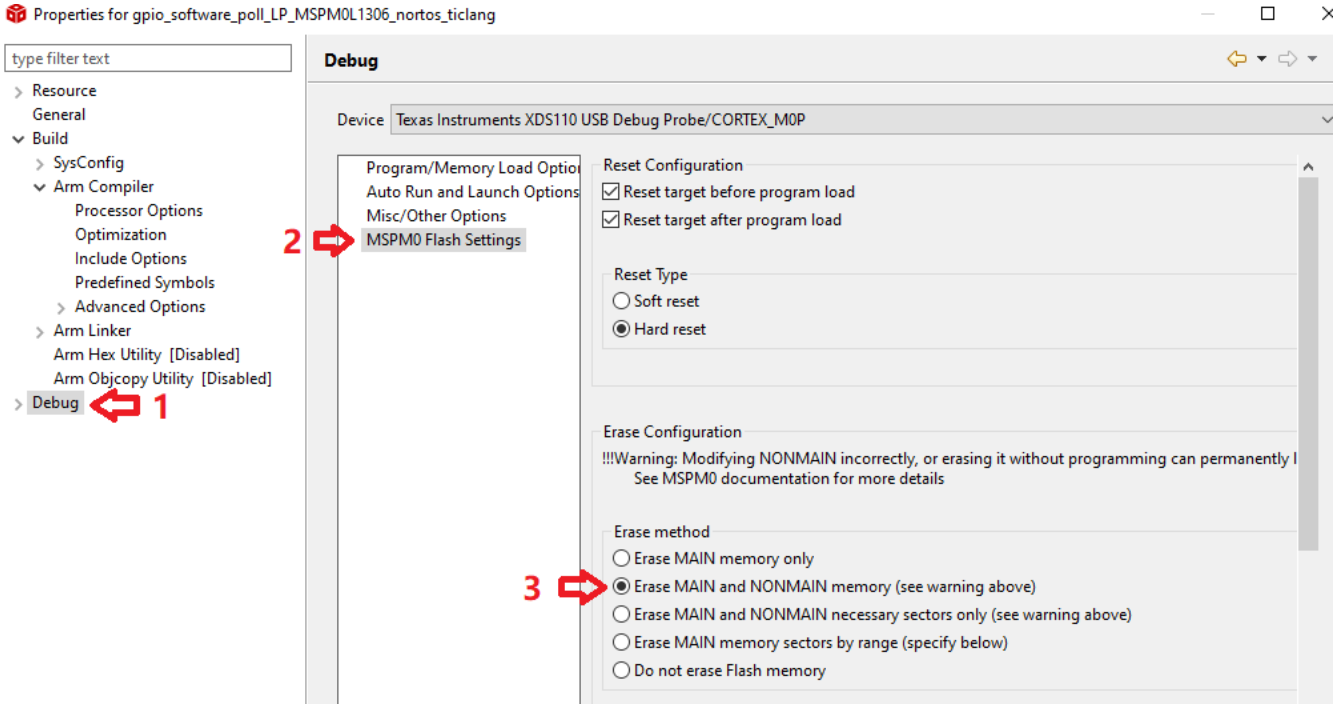

図 3-4. NON-MAIN フラッシュ消去を有効化

4 ブートローダーホスト

4.1 MCU ホスト コードの概要

Code Composer Studio™ (CCS) をベースにした MCU ホスト デモは、次のフォルダ内にあります

< …\mspm0_sdk_xxxx\examples\nortos\LP_MSPM0xxxx\bsl >

これらのデモは、UART、I2C、SPI、または CAN を介してターゲットの MSPM0 デバイスを更新できます。BSL ホスト デモのソース コードには、ターゲット デバイスのファームウェアが含まれており、これは SDK の GUI ツールによって .txt イメージ ファイルから変換された application_image.h ファイルに格納されています。詳細については、セクション 4.1.2 を参照してください。これには、main.c ファイル内の BSL_PW_RESET 配列として定義されている BSL パスワードも含まれています。ターゲット側のパスワードは Non-main フラッシュ内、つまり BSL 構成領域の BSLPW に定義されています。図 4-1 に、ホスト BSL プロジェクトのフロー チャートを示します。

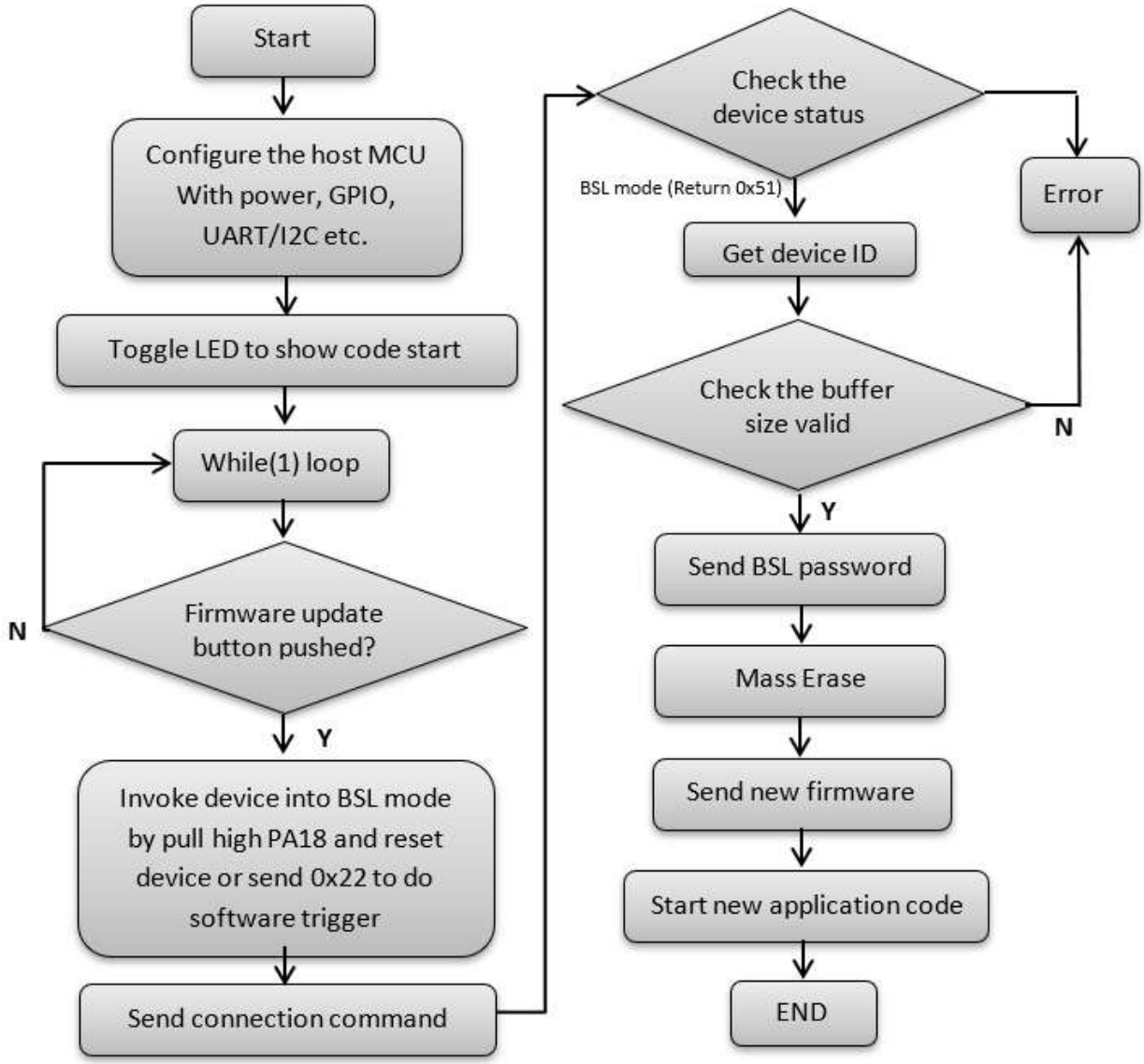

図 4-1. ホスト プロジェクトのフロー図

ホスト デモは、PA18 ピンを High にしてからリセットを行うハードウェア トリガをサポートできます。また、このデモはソフトウェア呼び出しにも対応しており、0x22 コマンドを送信するだけで BSL を起動できます。

注

ソフトウェア トリガを使用する場合は、ソフトウェア トリガ機能を備えたアプリケーション デモを先にダウンロードする必要があります。

4.1.1 ハードウェア接続

ホスト デモ コードは、MSPM0 をホスト MCU としても使用します。ホストとターゲットの間のハードウェア信号接続を表 4-1 に示します。

表 4-1. ハードウェア信号接続

信号	LP-MSPM0G3507		LP-MSPM0L1306	
	ホスト デバイス	ターゲット デバイス	ホスト デバイス	ターゲット デバイス
リセット	PB0	NRST ピン	PA3	NRST ピン
呼び出し	PB16	PA18	PA7	PA18
UART	PB7/UART1_RX	PA10/UART0_TX	PA9/UART0_RX	PA23/UART0_TX
	PB6/UART1_TX	PA11/UART0_RX	PA8/UART0_TX	PA22/UART0_RX
I2C	PB2/I2C1_SCL	PA1/I2C0_SCL	PA11/I2C0_SCL	PA1/I2C0_SCL
	PB3/I2C1_SDA	PA0/I2C0_SDA	PA10/I2C0_SDA	PA0/I2C0_SDA
SPI	PB9/SPI1_SCLK	PB9/SPI1_SCLK	PA6/SPI0_SCLK	PA6/SPI0_SCLK
	PB8/SPI1_PICO	PB8/SPI1_PICO	PA5/SPI0_PICO	PA5/SPI0_PICO
	PB7/SPI1_POCI	PB7/SPI1_POCI	PA4/SPI0_POCI	PA4/SPI0_POCI
	PB6/SPI1_CS	PB6/SPI1_CS	PA8/SPI0_CS0	PA8/SPI0_CS
CANFD	PA12/CAN_TX	PA13/CAN_RX	\	\
	PA13/CAN_RX	PA12/CAN_TX	\	\

注

- UART、I2C、SPI のいずれか 1 つの通信インターフェイスのみを接続します。ターゲット側のピンは既定の構成ピンですが、Non-main フラッシュで変更できます。
- ソフトウェア呼び出しを使用する場合、リセット信号および呼び出し信号を接続する必要はありません。
- CANFD を使用する場合、ホスト側およびターゲット側の MSPM0 の両方にトランシーバを接続する必要があります。

.

表 4-2. MSPM0C1104 のハードウェア信号接続

信号	LP-MSPM0C1104	
	ホスト デバイス	ターゲット デバイス
リセット	PA2	NRST ピン
呼び出し	PA4	PA18
UART	PA24/UART0_RX	PA27/UART0_TX
	PA27/UART0_TX	PA26/UART0_RX
I2C	PA11/I2C0_SCL	PA11/I2C0_SCL
	PA0/I2C0_SDA	PA0/I2C0_SDA

注

ソフトウェア呼び出しを使用する場合、リセット信号および呼び出し信号を接続する必要はありません。

注

UART インターフェイスを使用する場合は、J101 上で XDS110 UART バック チャネルに接続されているジャンパを取り外す必要があります。

表 4-3. MSPM0H3216 のハードウェア信号接続

信号	LP-MSPM0H3216	
	ホスト デバイス	ターゲット デバイス
リセット	PA13	NRST ピン

表 4-3. MSPM0H3216 のハードウェア信号接続 (続き)

信号	LP-MSPM0H3216	
呼び出し	PA12	PA11
UART	PA18/UART1_RX	PA27/UART0_TX
	PA17/UART1_TX	PA30/UART0_RX
I2C	PA9/I2C0_SCL	PA27/I2C0_SCL
	PA8/I2C0_SDA	PA30/I2C0_SDA

注

ソフトウェア呼び出しを使用する場合、リセット信号および呼び出し信号を接続する必要はありません。

注

ターゲット ボードの LED2 を制御するには、PA30 のジャンパを取り外す必要があります。

表 4-4. MSPM0C1106 のハードウェア信号接続

信号	LP-MSPM0C1106	
	ホスト デバイス	ターゲット デバイス
リセット	PA13	NRST ピン
呼び出し	PA12	PA18
UART	PA11/UART0_RX	PA10/UART0_TX
	PA10/UART0_TX	PA11/UART0_RX
I2C	PB2/I2C0_SCL	PA11/I2C0_SCL
	PB3/I2C0_SDA	PA10/I2C0_SDA

注

ソフトウェア呼び出しを使用する場合、リセット信号および呼び出し信号を接続する必要はありません。

4.1.2 TXT からヘッダ ファイルへの変換

MCU ホスト ファームウェアには、ターゲット アプリケーションのイメージがヘッダ ファイル (application_image.h) として含まれています。このヘッダ ファイルは、新しいアプリケーション ファームウェアであり、MSPM0 ターゲット デバイスに書き込む必要があります。ヘッダ ファイルを取得するには、GUI MSPM0_BSL_GUI.exe に次のパスの変換ユーティリティがあります:

<...\mspm0_sdk_xxxx\tools\bsl\BSL_GUI_EXE>。または、ここからダウンロードします

1. [MoreOption] メニューで [TXT_to_H] を選択します。
2. 変換する TI-TXT 形式ファイルを選択します。入力フォルダに用意されている、いくつかのシンプルなアプリケーション デモ ファイルを使用できます。
3. 出力ファイル用のフォルダを選択します (例: Output という名前のフォルダを選択)。
4. [Run] ボタンをクリックして、ソフトウェアの実行を開始します。

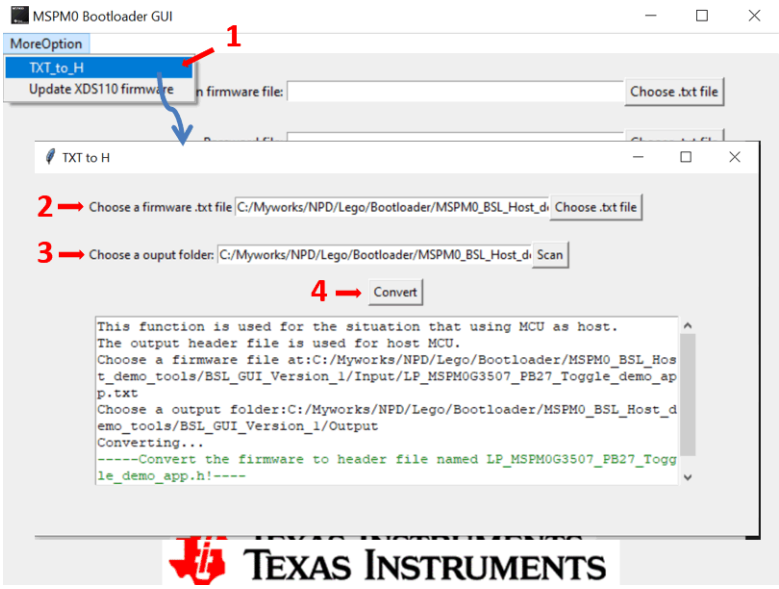

図 4-2. TXT ファイルをヘッダ ファイルに変換する手順

4.1.3 デモを使用する手順

以下の手順では、ホストとして LP-MSPM0G3507 を使用して MSPM0 MCU をプログラムする方法を説明します。MSPM0G3507 をターゲット デバイスとして使用し、このデモではハードウェア BSL 起動と UART 通信を使用します。同様のプロセスを使用して、UART、I2C、SPI のいずれかを介して他の MSPM0 デバイスをプログラムできます (表 4-1 を参照)。

注

誤った BSL パスワードを送信すると、デバイスは 2 秒間スリープ状態に入り、インターフェース (UART/I2C) は無効になります。UART の場合、コアの応答の後に 0x00 を送信し、ホスト側はこのバイトを無視するように見えます。インターフェイスはデフォルト設定で 2 秒後に有効になりますが、事前に UART ボーレートを変更した場合は、再度これを再設定する必要があります。

1. ハードウェア信号を図 4-3 に示すように接続します。この例では UART を使用しているため、I2C 信号を接続する必要はありません。

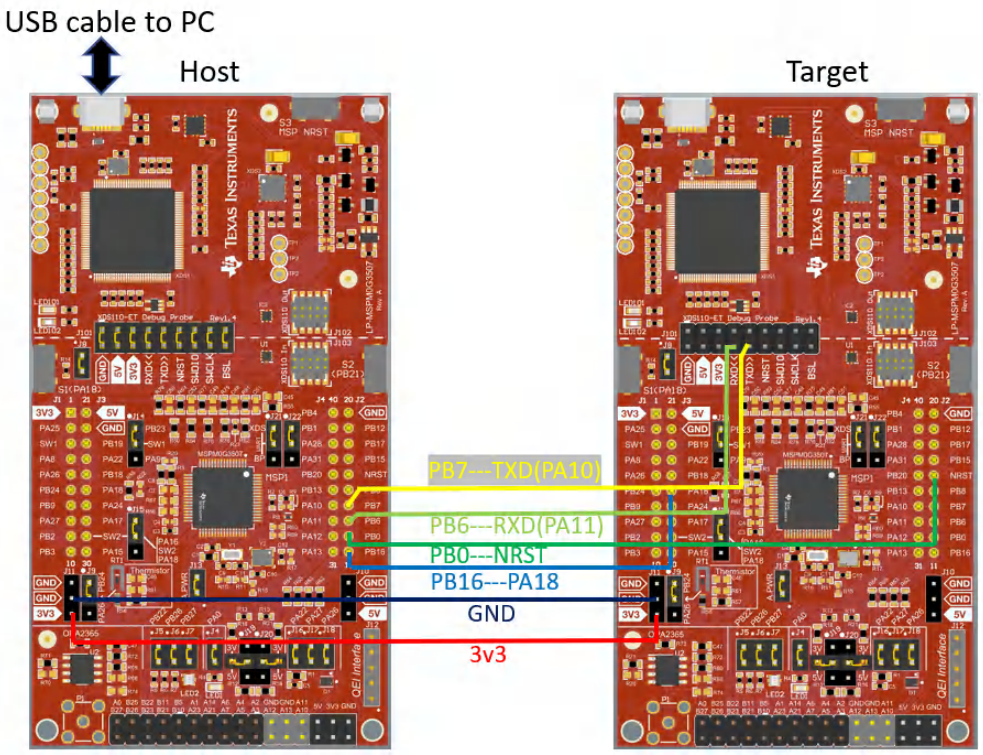

図 4-3. ハードウェア信号接続

2. 表 4-5 に示すように、ジャンパを接続します。

表 4-5. ジャンパ接続

ボード	モード	接続するためのジャンパ	接続解除するためのジャンパ
LP-MSPM0G3507	ホスト	J101 (電源とデバッグ)、J4、J7 (LED)	なし
LP-MSPM0G3507	ターゲット	J7、(LED) J21、J22 (UART～J101 XDS110)	すべて J101

注

LP-MSPM0L1306 をターゲット ボードとして使用する場合、J6 上のジャンパを取り外す必要があります。

3. → フォルダ < … \mspm0_sdk_xxxx\examples\nortos\LP_MSPM0G3507\bsl\bsl_host_mcu_to_mspm0g1x0x_g3x0x_target_uart> にある UART デモ付き BSL ホストを CCS にインポートします。

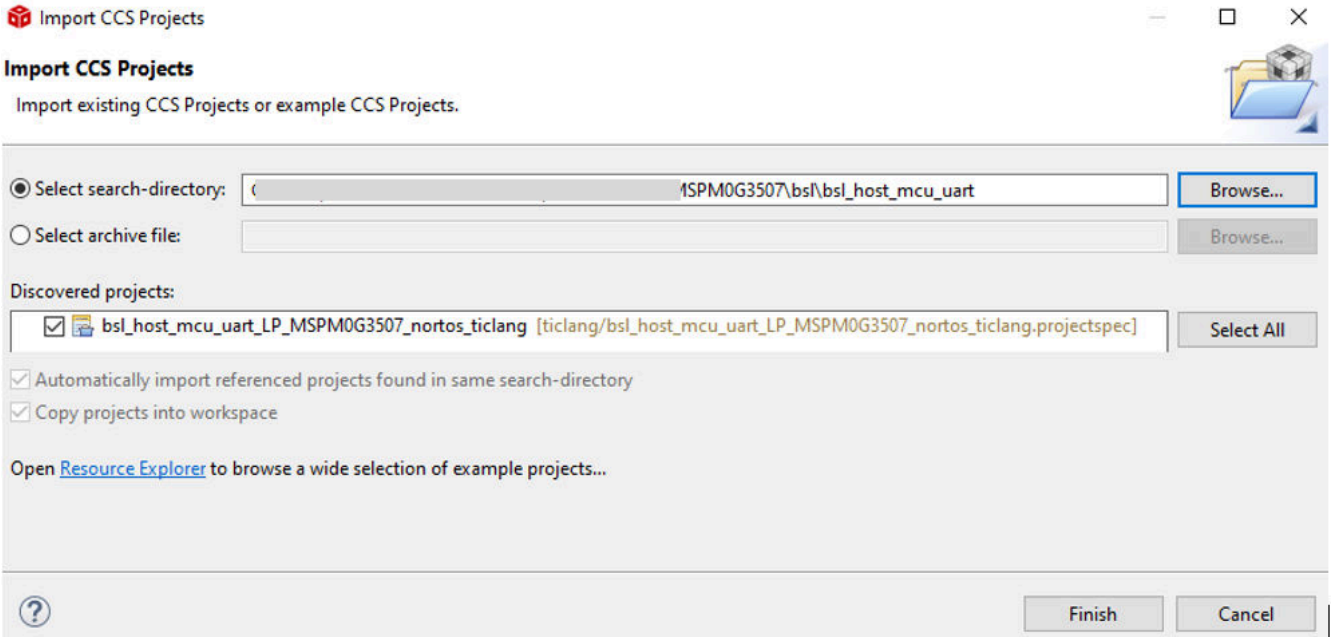

図 4-4. プロジェクトを CCS にインポート

4. 必要に応じて、main.c の bsl_password 配列内のパスワードを変更します。デフォルトのパスワードは、すべて 32 バイトの 0xFF です。ターゲット BSL パスワードは、メイン以外のメモリで定義されます。詳細については、テクニカル リファレンス マニュアル [1] または [2]、ブート ローダ ユーザー ガイド [3] を参照してください。
5. モを実行するだけでアプリケーション コードを変更する必要がない場合、BSL ホスト デモにはデフォルトのファームウェア ファイル application_image_uart.h が含まれており、これは bsl_software_invoke_app_demo_uart という名前のデモから生成されたものです。この場合、手順 6 から 8 は省略できます。
6. アプリケーション コード (ここでは bsl_software_invoke_app_demo_uart デモを使用可能) を CCS にインポートし、ターゲット デバイス用のファームウェアを TI-TXT hex 形式で生成します (図 4-5 を参照)。

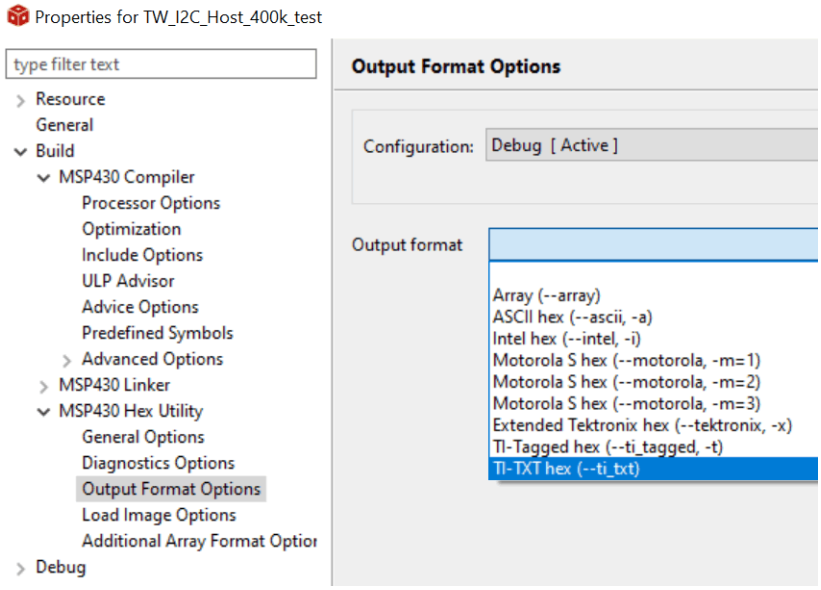

図 4-5. CCS で TI-TXT Hex ファイルを生成する

7. GUI ツール MSPM0_BSL_GUI.exe を実行して、ターゲット デバイスのファームウェア .txt 形式をヘッダ ファイルに変換します。詳細については、セクション 4.1.2 を参照してください。

8. GUI によって生成された出力ファイル xxx.h の内容を、ホスト プロジェクトのファイル application_image.h にコピーします。
9. ホスト プロジェクトをビルドし、LP-MSPM0G3507 にダウンロードします。
10. ホスト ボードのボタン S2 を押して、ファームウェアの更新を開始します。エラーが発生すると、LED1 が点灯します。

4.2 PC ホストの例

ホストとなる PC には、ソフトウェア GUI (MSPM0_BSL_GUI.exe または Uniflash) と USB-UART ブリッジが必要です。2 種類のハードウェア ブリッジが含まれており (MSPM0_BSL_GUI.exe で選択可能)、1 つは MSPM0 LaunchPad キット上の XDS110、もう 1 つはスタンドアロンの XDS110 です。どちらのブリッジも、USB-UART ブリッジとして使用できるバックチャネル UART をサポートしています。LaunchPad キット上の XDS110 は、NRST ピンと BSL 呼び出しピンの制御をサポートしており、GUI から MCU を BSL モードで起動するために使用できます。スタンドアロンの XDS110 では、AUX 接続ポートの 2 つの GPIO 出力ピン (IOOUT0 と IOOUT1) を使用して、ターゲット デバイス上の NRST ピンと BSL 呼び出しピンを制御し、BSL モードを開始できます。(これは MSPM0_BSL_GUI.exe を使用して実装されています)。

注

- LP-MSPM0C1104 のオンボード XDS110 では BSL 呼び出しピンが配線されていないため、GUI を使用したソフトウェア呼び出しのみがサポートされています。
- ソフトウェア リリースの制限により、MSPM0_BSL_GUI.exe を SDK に含めることはできません。SDK 内のソース ファイルで生成するか、ここからダウンロードできます

4.2.1 イメージ ファイルとパスワード ファイルの準備

GUI を使用してファームウェアをダウンロードする前に、アプリケーション ファームウェア ファイルと BSL パスワード ファイルの 2 つを準備します。

GUI (MSPM0_BSL_GUI.exe) は TI-TXT 形式のみをサポートしています。この形式のイメージ ファイルを CCS で生成する方法の詳細については、セクション 4.1.3 内の 6 を参照してください。

パスワードファイルの形式は、図 4-6 に示すように TI-TXT の形式と類似しています。BSL パスワードは Non-Main メモリに定義されています。詳細については、テクニカル リファレンス マニュアル[1] [2]またはブートローダーのユーザー ガイド[3]を参照してください。BSL パスワードがデフォルト値 (すべて 0xFF) でない場合は、パスワード ファイルを修正します。BSL_Password32_Default.txt という名前のデフォルトのパスワード ファイルは、次のフォルダにあります: <...\mspm0_sdk_xxxx\tools\bsl\BSL_GUI_EXE\Input>。

```
@00000000
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
q
```

図 4-6. BSL デフォルト パスワード ファイル (BSL_Password32_default.txt)

4.2.2 GUI を使用する手順

1. ターゲット デバイスと XDS110 を PC に接続します。LaunchPad キットに統合された XDS110 を使用する場合、マイクロ USB ケーブルを PC に接続してください (図 4-7 を参照)。

   MSPM0G3507 の ROM ベース BSL UART ピンは PA10 と PA11 であり、これらのピンは XDS110 バックチャネル UART に直接接続されているため、J101 の全ジャンパが必要です (表 4-7 を参照)。

   LP-MSPM0L1306 では、XDS110 バックチャネル UART ピンは BSL UART ピンと異なるため、J101 の TXD および RXD を切り離し、ジャンパワイヤで PA22 と PA23 を接続します (表 4-7 を参照)。

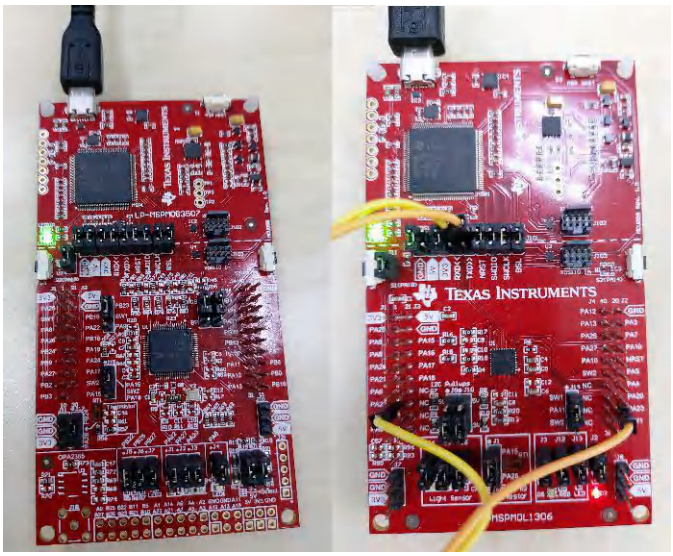

図 4-7. LaunchPad キットの接続 (左: LP-MSPM0G3507、右:LP-MSPM0L1306)

表 4-6. ジャンパ接続

基板	モード	ジャンパの実装が必要	ジャンパの未実装が必要
LP-MSPM0G3507	ターゲット	J101 (電源、UART ピン、リセットおよび BSL 起動ピン)、J4、J7 (LED)、J21、J22 (UART ～ J101 XDS)	該当なし
LP-MSPM0L1306	ターゲット	J101 (GND、3V3、NRST、BSL)、J2、J3 (LED)	J101 (TXD、RXD)

スタンドアロン XDS110 の場合、補助インターフェイス (AUX) は表 4-7 の信号接続を使用します。

表 4-7. スタンドアロン信号接続

信号	スタンドアロン XDS110		ターゲット デバイス		
	信号	AUX ポート	信号	LP-MSPM0G3507	LP-MSPM0L1306
NRST	IO 出力	IOOUT0	NRST	NRST ピン	NRST ピン
呼び出し	IO 出力	IOOUT1	デフォルト:ピンを起動	PA18	PA18
UART	RXD	UARTRX	TXD	PA10/UART0_TX	PA23/UART0_TX
	TXD	UARTTX	RXD	PA11/UART0_RX	PA22/UART0_RX

2. GUI を使用して、イメージをターゲットにダウンロードします。
   a. ダウンロードする必要のある TI-TXT 形式の画像ファイルを選択します。(input という名前のフォルダに 2 つのデモ イメージがあります)
   b. TI-TXT 形式のパスワード ファイルを選択します (デフォルトのファイルは *input* フォルダ内にあります)。このファイルの準備の詳細については、セクション 4.2.1 を参照してください。
   c. ハードウェア ブリッジを選択します。

d. [Download] (ダウンロード) ボタンをクリックします。

GUI が自動的に BSL を起動するため、この操作中に手動で BSL を起動する必要はありません。

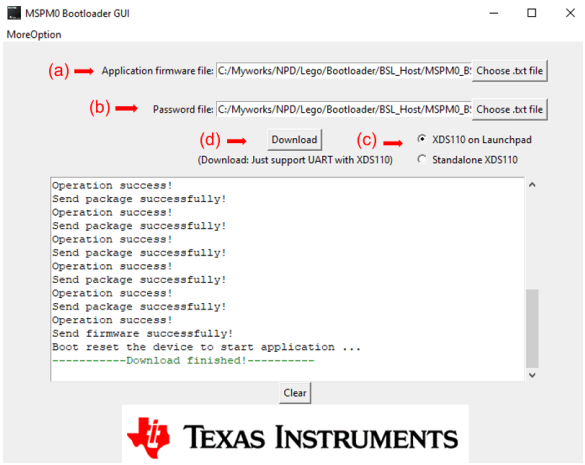

図 4-8. UART を使用して GUI でイメージをダウンロードする手順

3. XDS110 を使用する場合、この GUI は XDS110 ファームウェアバージョン firmware_3.0.0.20 以上をサポートします。イメージのダウンロード中にエラーが発生した場合は、XDS110 ファームウェアを更新します。

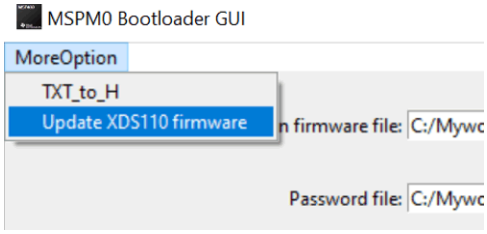

図 4-9. XDS110 ファームウェアを更新し

5 ブートローダーのターゲット

5.1 デフォルトの ROM ベースの BSL

一部の MSPM0 デバイスには ROM ベースの BSL が含まれています。ROM ベースの BSL は、UART と I2C インターフェイスのみをサポートします。これは変更できませんが、一部の機能は NON-MAIN フラッシュで構成できます。たとえば、UART/I2C ピン割り当てや、I2C アドレスなどです。詳細については、*MSPM0 ブートローダー ユーザー ガイド*を参照してください。

5.1.1 UART インターフェイス

MSPM0 ROM ベースの BSL UART は、次の構成でイネーブルになります。次の構成でイネーブルになります:

- ボーレート:9600 bps (一部のデバイスでは NON-MAIN または BSL コア コマンドで変更可能)
- データ幅:8 ビット
- 1 ストップ ビット
- パリティなし

UART ピンの割り当てとボーレートは (一部のデバイスのみ) NON-MAIN フラッシュで構成できます。

5.1.2 I2C インターフェイス

MSPM0 ROM ベースの BSL I2C は、次の構成でイネーブルになります:

- アドレス:0x48 (NON-MAIN でも変更可能)
- アドレス幅:7 ビット

I2C ピンの割り当てとスレーブ アドレスは NON-MAIN フラッシュで構成できます。

5.2 フラッシュベースのプラグイン インターフェイスのデモ

ROM ベースの BSL インターフェイス セクションにおいて、設定がアプリケーションの要件を満たしている場合。フラッシュベースのプラグイン インターフェイスのデモを使用できます。通信セクション用のコードはどちらもオープンソースであるため、顧客はコード内の任意の設定を変更できます。フラッシュベースのプラグイン インターフェイス デモは、BSL パケットを受信することはできますが、パケットを解析することはできない点に注意してください。そのため、プラグイン インターフェイス デモでは、ROM BSL に格納されている BSL コアと連携してコマンドを解析する必要があります。

5.2.1 UART インターフェイス

MSPM0 フラッシュベース UART は、既定で以下の構成で有効化されています。

- ボーレート:9600bps
- データ幅:8 ビット
- 1 ストップ ビット
- パリティなし

5.2.1.1 デモの使用手順

MSPM0G3507 用のフラッシュベース UART プラグイン インターフェイス デモを使用する手順は以下のとおりです:

1. SDK から CCS にフラッシュ ベースの UART プラグイン インターフェイス デモをインポートします。

   <…\mspm0_sdk_xxxx\examples\nortos\LP_ MSPM0G3507\bsl\ bsl_uart_flash_interface >
2. 必要な変更を加え、プロジェクトをビルドします。
3. NON-MAIN 内の静的なフラッシュ書き込み保護設定をクリアするため、ファクトリ リセットを実行します。デバイスが空白の場合、この手順はスキップできます。この操作を実行する手順の詳細については、セクション 6.2 を参照してください。
4. UART プラグイン コードをフラッシュにダウンロードします。イメージをダウンロードする際の重要な点は、図 3-4 に示されている NON-MAIN フラッシュ消去を有効にすることです。プラグイン インターフェイス デモは直接デバッグすることはできません。詳細については、セクション 5.2.1.2 を参照してください。
5. LP-MSPM0G3507 launchpad を 1 台用意し、BSL ホスト デモを使用してファームウェアを更新します。詳細については、セクション 4.1.3 (MCU をホストとする場合) または セクション 4.2.2 (PC をホストとする場合) を参照してください。

5.2.1.2 プラグイン インターフェイス コードをデバッグする方法

プラグイン インターフェース デモ コードを変更してデバッグを行う際のガイドラインは以下のとおりです:

1. 必要に応じて変更を加え、プラグイン インターフェイス プロジェクトをビルドします。
2. セクション 3.2 のステップ C と同様に NON-MAIN を消去した状態でデバイスにダウンロードし、その後電源を再投入してください。
3. 図 5-1 に示すようにデバイスを起動します。ステップ 2 で、CCXML ファイルを右クリックします。

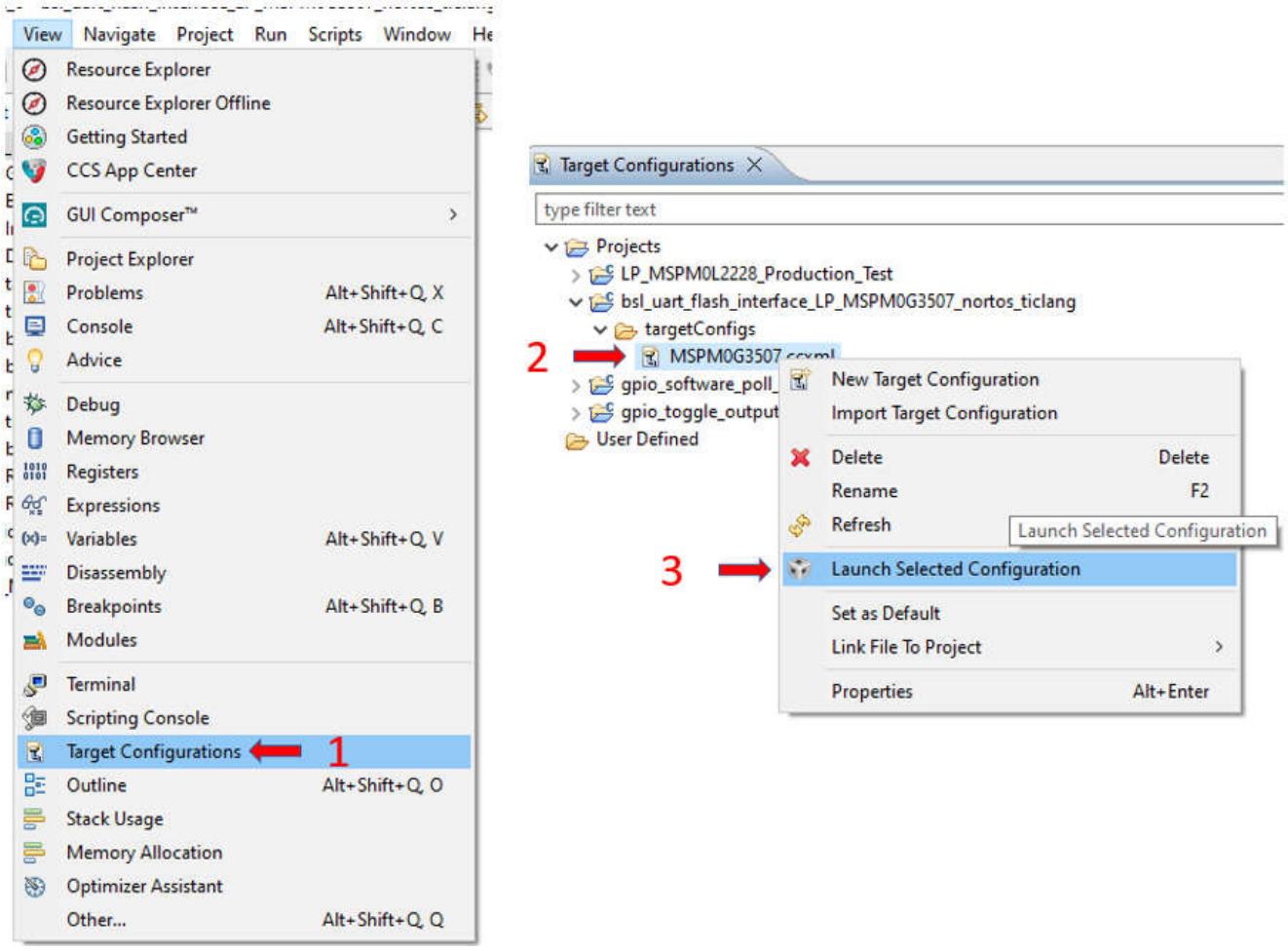

図 5-1. CCS でデバイスを起動し

4. ターゲットを接続します。

図 5-2. デバイスを CCS に接続し

5. プラグイン インタフェイス コードのシンボルをロードし、必要なブレークポイントを入力します。

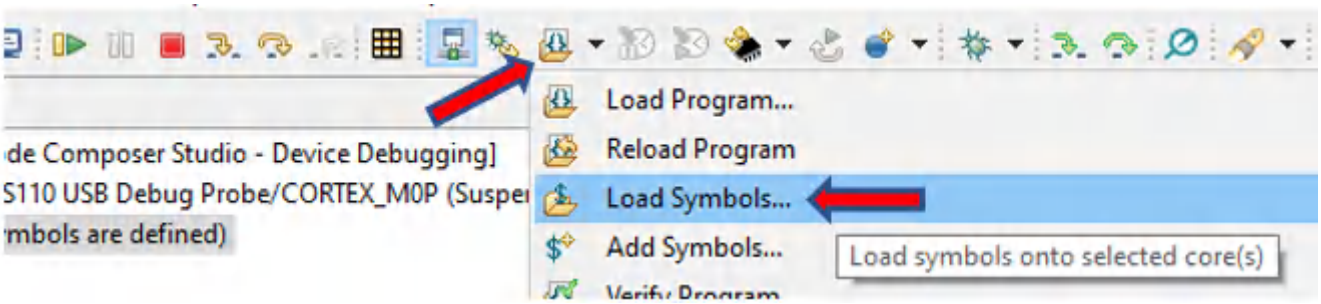

図 5-3. CCS にシンボルをロード

6. デバッグを実行するためにコードを実行し続けます。アプリケーション領域が空になると、デバイスは自動的に BSL モードに移行します。

5.2.2 I2C インターフェイス

BSL の I2C インターフェイスは、I2C ターゲットまたはスレーブとして動作します。

- I2C ターゲット アドレスはデフォルトで 0x48 です
- SCL および SDA ラインには外部プルアップ抵抗が必要です。

デモの動作の詳細は、UART プラグイン インターフェイスで同様です。ステップ バイ ステップの動作とデモのデバッグ方法については、セクション 5.2.1.1 およびセクション 5.2.1.2 を参照してください

5.2.3 SPI インターフェイス

SPI プラグイン デモは、SPI をターゲット モードに構成し、他のデフォルト設定を以下のように構成します:

- Motorola 4 ワイヤ接続
- 最初のクロック エッジでキャプチャされたデータ
- クロック極性 Low
- ビット順序 MSB

デモの動作の詳細は、UART プラグイン インターフェイスで同様です。ステップ バイ ステップの動作とデモのデバッグ方法については、セクション 5.2.1.1 およびセクション 5.2.1.2 を参照してください

5.2.4 CAN インターフェイス

CAN プラグイン デモは、デフォルトで次のように、CAN モジュールによって構成されます:

- この例は、最初は 1Mbps で CAN モードで動作するように設定されています。
- ホストから取得した構成に基づき、change baudrate コマンドを使用して通信のビットレートを変更します。

change baudrate コマンドのデータ セクションは、図 5-4 に示す形式と一致していることが予想されます。

Padding (5)	DRP (5)	DSJW (4)	DTSEG2 (4)	DTSEG1 (5)	NRP (9)	NSJW (7)	NSEG2 (7)	NTSEG1 (8)	BRS (1)	FD (1)

図 5-4. Change Baudrate コマンドのデータ セクション

- CAN モードを CAN FD に変更すると、送信遅延補償属性値をキャリブレーションするために任意の CAN フレームが CAN バスに挿入されます。ID 値は必要に応じて変更できます。
- BSL プラグインで受け入れられるメッセージ識別子は 0x003 です
- BSL プラグインから送信されたメッセージ識別子は 0x004 です

デモの動作の詳細は、UART プラグイン インターフェイスで同様です。ステップ バイ ステップの動作とデモのデバッグ方法については、セクション 5.2.1.1 およびセクション 5.2.1.2 を参照してください

5.3 セカンダリ BSL デモ

プライベート プロトコルが必要な場合、ROM ベースの BSL コアは使用できなくなり、また ROM BSL を搭載していない MSPM0C の場合には、セカンダリ BSL デモを参照できます。プロトコルを容易に変更できる、完全にオープンソース化されたセカンダリ BSL デモが SDK に用意されています。セカンダリ BSL デモのデフォルト プロトコルは、ROM ベースの BSL で同じです。図 1-6 で説明しているセカンダリ BSL デモにはいくつかの種類があります。

表 5-1. MSPM0 セカンダリ BSL デモの概要

デモ	SDK 内のプロジェクト	使用事例
セカンダリ BSL は 0x1000 から開始	<...\mspm0_sdk_xx\examples\nortos\LP_MSPM0L1306 (または LP_MSPM0G3507)\bsl\secondary_bsl_uart/i2c/spi/can>	ブート コードで BSL 起動検出を行えるデバイス (通常は ROM BSL 搭載デバイス) かつプライベート プロトコルが必要な場合にのみ使用できます。
MSPM0C 向けフラッシュベース BSL は 0x0 から開始	<...\mspm0_sdk_xx\examples\nortos\LP_MSPM0C1104\bsl\flash_bsl>	ROM ベースの BSL を搭載していない MSPM0、または起動やリセットのたびにアプリケーション領域の CRC を実行するなど、アプリケーションにジャンプする前の判定条件を変更する必要がある場合。
ライブ ファームウェア アップデート BSL	該当なし	ライブ ファームウェア アップデートが必要です

従来のフラッシュベース BSL 設計は、MSPM0C 向けフラッシュベース BSL が 0x0 から開始するデモにより近いものです。このような設計では、アプリケーション コードに直接ジャンプし、PC をアプリケーション コードの開始アドレスに設定できます。ただし、予期しない状況でスタック競合の問題が発生する可能性があります。0x1000 から開始するセカンダリ BSL の設計では、リセットを利用してアプリケーション コードにジャンプし、アプリケーション コードに入る前にすべてのレジスタや SRAM をリセットできるため、より安定した動作が可能です。ROM ベースの BSL を搭載した MSPM0 デバイスでプライベート プロトコルも必要な場合は、0x1000 から開始するセカンダリ BSL デモを使用することを強く推奨します。

5.3.1 フラッシュベースのセカンダリ BSL を 0x1000 から開始

0x1000 から開始するセカンダリ BSL は、0x0 以外のフラッシュ領域の任意の場所に配置できます。アプリケーション コードは 0x0 から開始する必要があるからです。セカンダリ BSL デモの実行フローを図 1-6 に示します。このデバイスがサポートしている場合、UART、I2C、SPI、CAN インターフェイスをサポートできます。デモのステップ バイ ステップ動作は、セクション 5.2.1.1 に示すものと同じです。

変更後にコードをデバッグする必要がある場合は、セクション 5.2.1.2 の手順に従います。

セカンダリ BSL では、割り込みベクタ テーブルのオフセット アドレスが 0x1000 からに移動されており (コードが 0x1000 から開始するため)、リセット ハンドラは startup_mspm0xxxx_ticlang という名前のファイルに配置されています。

セカンダリ BSL を別のフラッシュ領域に移動しようとするときは、cmd ファイルで実行できます。たとえば、セカンダリ BSL を 0x4000 から開始し、図 5-5 に示すように cmd ファイルを変更します。

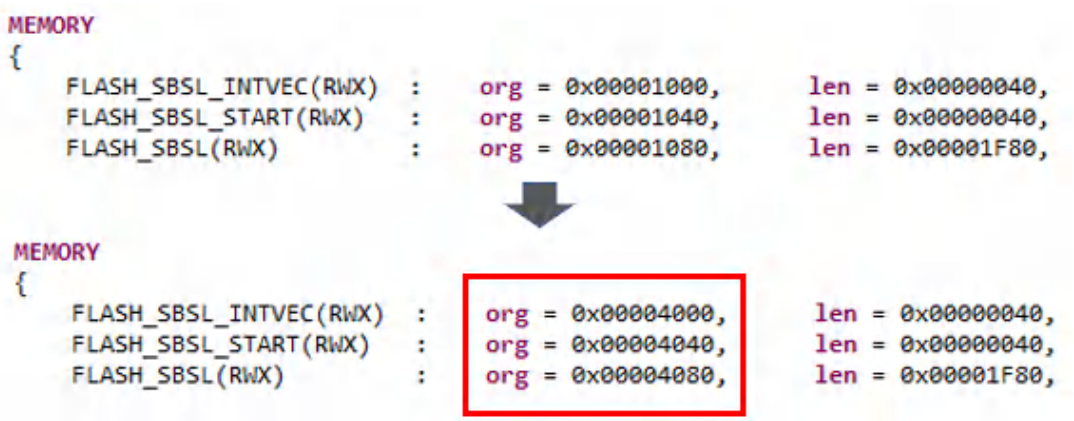

図 5-5. cmd ファイル修正による 0x4000 への移動

図 5-6 に示すように、フラッシュの静的書き込み保護パラメータと、代替 BSL の開始アドレスも、Sysconfig ファイルで変更する必要があります。

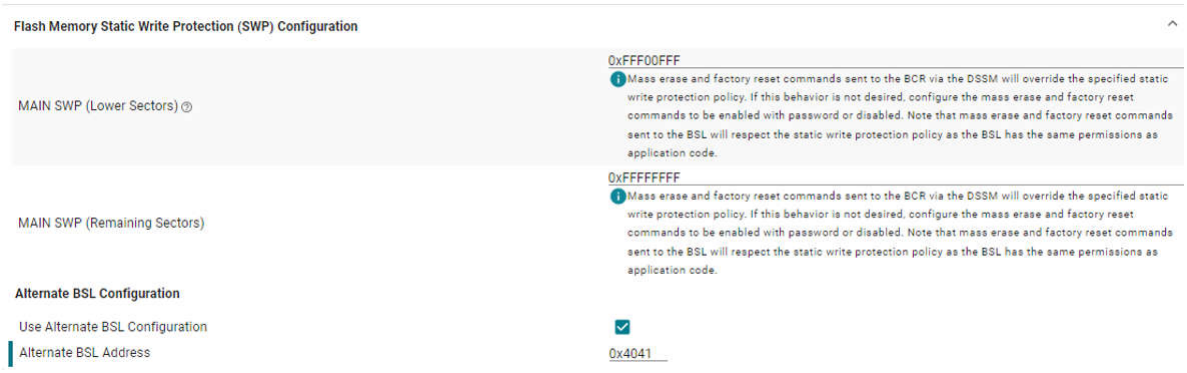

図 5-6. Sysconfig ファイル修正による 0x4000 への移動

セカンダリ BSL の修正に加え、アプリケーションの cmd ファイルも修正し、セカンダリ BSL が使用するフラッシュ領域を再利用しないようにする必要があります。

5.3.2 フラッシュベースのセカンダリ BSL を 0x0000 から開始

0 アドレスから開始するセカンダリ BSL の場合、MCU は起動またはリセットのたびにセカンダリ BSL を実行できます。セカンダリ BSL では、カスタムチェック判定を使用して、ファームウェア更新のために BSL に留まるか、アプリケーションに移行するかを決定します。この設計の利点は、顧客が GPIO や未書き込みデバイス検出に限定されない特別な判定を使用できる点です。例えば、アプリケーション コードにジャンプする前にアプリケーションの CRC を確認し、アプリケーション コードの整合性を確保する必要があります。もう一つの使用ケースは、MSPM0C のように ROM BSL を搭載していない一部の MSPM0 デバイス向けであり、この点に関するデモコードが SDK に用意されています。アプリケーションにジャンプしたら、PC をアプリケーションの開始アドレスに設定できます。

この種の BSL を使用するには、セカンダリ BSL 用とアプリケーション用の 2 つのプロジェクトを作成します。フラッシュ領域は分離する必要があります。各プロジェクトには 2 つの割り込みテーブルがあります。BSL では、BSL からアプリケーション コードにジャンプする際に現在の割り込みテーブルを有効にするため、ベクタ テーブル オフセット レジスタ (SCB->VTOR) を設定する必要があります (アプリケーションから BSL へのジャンプはリセットを使用するため、ベクタ テーブル オフセット レジスタは自動的にリセットされます)。

ライブ ファームウェア アップデートをサポートできるセカンダリ BSL デモ コードもあります。このデモは、アプリケーション コードを停止せずに、セカンダリ BSL ファームウェア アップデートが実行中であることを意味します。詳細については、*MSPM0 ライブ ファームウェア アップデート(LFU) ブートローダの実装*を参照してください。

5.3.2.1 MSPM0C 用のフラッシュベース 0x0 アドレス BSL デモ

MSPMC デバイスは ROM ベースの BSL を搭載していないため、フラッシュベースの BSL を使用する必要があり、起動またはリセットのたびに起動検出コードを実行するために 0x0 アドレスから開始する必要があります。

このデモでは、デバイスが起動またはリセットされると、まずセカンダリ BSL コードに入ります。BSL リセットハンドラ内でデバイスは BSL 起動条件 (未書き込み検出、GPIO 起動、ソフトウェア起動) を検出し、ファームウェア更新のために BSL コードに留まるか、PC をアプリケーションの開始アドレスに設定してアプリケーション コードに移行するかを決定します。このデモについては、アプリケーション コードの CRC チェックは付属していません。*MSP430FRBoot – メイン メモリ ブートローダおよび MSP430™ FRAM 大容量メモリ モデル デバイス向け OTA アップデート*アプリケーション ノートを参照してください。

現在このデモでは、リセット ハンドラ ISR 内でアプリケーションの開始アドレスに PC を設定するために、次のコードが使用されています。

```
uint32_t *appResetHandler = (uint32_t *) (MAIN_APP_START_ADDR + VTOR_RESET_HANDLER_OFFSET);
appPointer FlashBSL_applicationStart = (appPointer) * (appResetHandler);
/* Before branch check if the address of reset handler is a valid Flash address */
if ((*((uint32_t *) MAIN_APP_RESET_VECTOR_ADDR) >= MAIN_APP_START_ADDR) &&
(*((uint32_t *) MAIN_APP_RESET_VECTOR_ADDR) < (MAIN_APP_START_ADDR + DEVICE_FLASH_SIZE))) {
FlashBSL_applicationStart(); }
```

ジャンプを行うもう一つの簡単な方法は以下の通りです (APP_AREA_START_ADDR は割り込みベクタテーブルを格納したアプリケーション領域の開始アドレスであり、4 バイト シフトすることでリセット ハンドラのアドレスを取得します)

```
/*! Jumps to application using the reset vector address */
#define TI_MSPBoot_APPMGR_JUMPTOAPP()     {((void (*)()) (*(uint32_t *)(APP_AREA_START_ADDR + 4)))
();}
```

ジャンプに問題がある場合は、以下のアセンブリ コードを試してください。このコードは、ブート コードとアプリケーション コードが SRAM 領域を共有している場合に、アプリケーションの開始アドレスにジャンプする前に RAM をクリアします。

```
 __asm(
#if defined(__GNUC__)
 ".syntax unified\n" /* Load SRAMFLASH register*/
#endif
 "ldr r4, = 0x41C40018\n" /* Load SRAMFLASH register*/
 "ldr r4, [r4]\n"
 "ldr r1, = 0x03FF0000\n" /* SRAMFLASH.SRAM_SZ mask */
 "ands r4, r1\n" /* Get SRAMFLASH.SRAM_SZ */
 "lsrs r4, r4, #6\n" /* SRAMFLASH.SRAM_SZ to kB */
#if defined ECC
 "ldr r1, = 0x20300000\n" /* Start of ECC-code */
 "adds r2, r4, r1\n" /* End of ECC-code */
 "movs r3, #0\n"
 "init_ecc_loop: \n" /* Loop to clear ECC-code */
 "str r3, [r1]\n"
 "adds r1, r1, #4\n"
 "cmp r1, r2\n"
 "blo init_ecc_loop\n"
#endif
 "ldr r1, = 0x20200000\n" /* Start of NON-ECC-data */
 "adds r2, r4, r1\n" /* End of NON-ECC-data */
 "movs r3, #0\n"
 "init_data_loop:\n" /* Loop to clear ECC-data */
 "str r3, [r1]\n"
 "adds r1, r1, #4\n"
 "cmp r1, r2\n"
 "blo init_data_loop\n"
 //Jump to Reset_Handler
 "ldr r0, = 0x7004\n" //FLASH_SBSL_INTVEC in .cmd file+ 4
 "ldr r0, [r0]\n"
 "blx r0\n"
 );
```

デバイスが ECC SRAM をサポートしている場合は、ECC の定義を追加します。このデモではアプリケーションの開始アドレスは 0x7004 に保存されています。自身のアプリケーションの開始アドレスに基づいてこの値を変更します。

ジャンプをペリフェラル初期化の後に配置する場合 (main() 関数を実行する前にジャンプ関数を呼び出さない場合)、いくつか注意すべき点があります:

- ISR にジャンプしないでください(リセット ハンドラ ISR を除きます)。
- グローバル割り込みを無効化 (__disable_irq を使用、アプリケーション コードで __enable_irq を呼び出して有効化) → 使用済みのペリフェラルをリセット → 保留中の NVIC IRQ をクリア (NVIC_ClearPendingIRQ(IRQn_Type IRQn) を呼び出し) → 必要に応じて RAM をクリア → アプリケーション コード開始アドレスにジャンプ。

5.3.2.2 ライブ ファームウェア アップデート

ライブ ファームウェア アップデート (LFU) は、ファームウェア更新中にアプリケーション コードを実行するために使用されます。このアップデートでは FreeRTOS を使用し、ファームウェア更新とアプリケーション コードの双方を同時に実行します。詳細については、*MSPM0 ライブ ファームウェア アップデート (LFU) ブート ローダの実装*を参照してください。

6 よくある質問

6.1 リンカ ファイルの変更

現在提供されているデモは CCS をベースとしており、ほとんどのデモではメモリ配置を行うためにリンカーファイルを修正する必要があります。この作業は CCS の cmd ファイルで処理されます。cmd リンカ ファイルの導入方法の詳細については、こちらの Web ページ *TI リンカ コマンド ファイル入門*を参照してください。

6.2 デバイスを回復するための CCS によるファクトリ リセット

デバイスにアクセスできない場合は、CCS のファクトリ リセットを実行してデバイスを回復してください。手順は次のとおりです:

1. ハードウェア接続:MSPM0 デバイス対応の XDS110。

   必要な信号:GND、SWDIO、SWCLK、NRST
2. ターゲット構成を開きます。

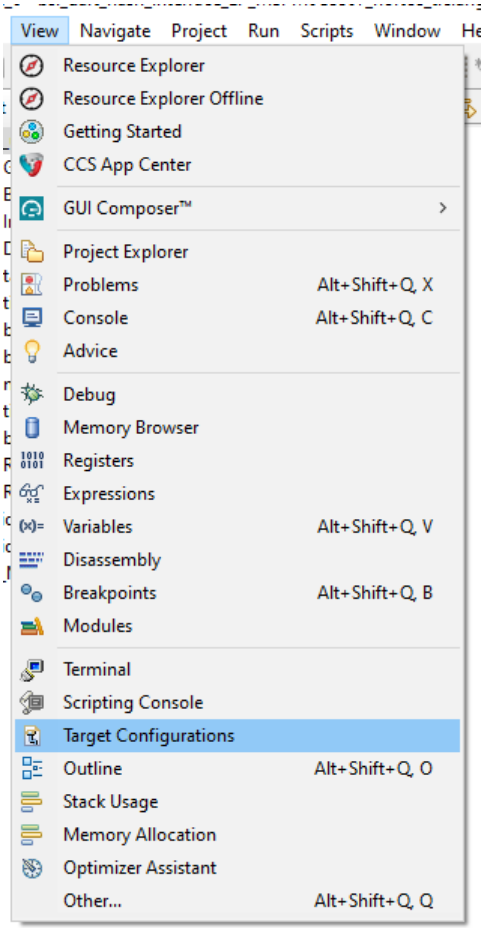

図 6-1. ターゲット構成を開く

3. ターゲット構成ウィンドウで、現在の MSPM0 プロジェクトを見つけ、フォルダを展開して .ccxml ファイルを見つけます:

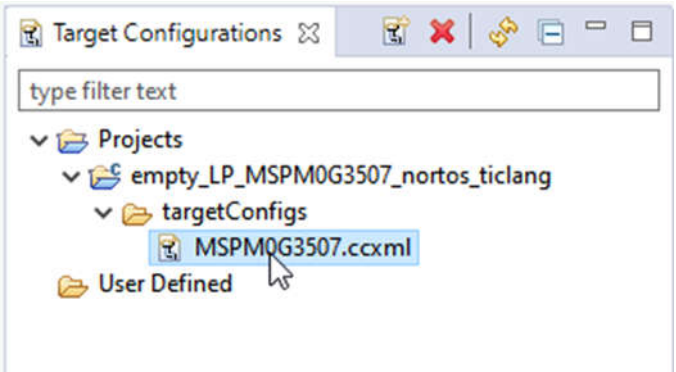

図 6-2. cxml ファイルを見つける

4. .ccxml ファイルを右クリックし、選択した構成の起動をクリックします。

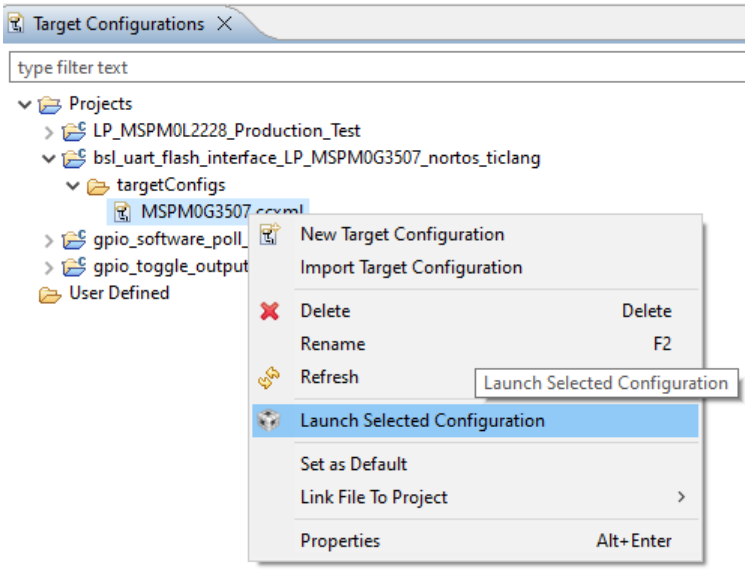

図 6-3. 選択した構成を起動し

5. Scripts → MSPM0G3507 Commands → MSPM0_Mailbox_FactoryReset_Auto をクリックします。

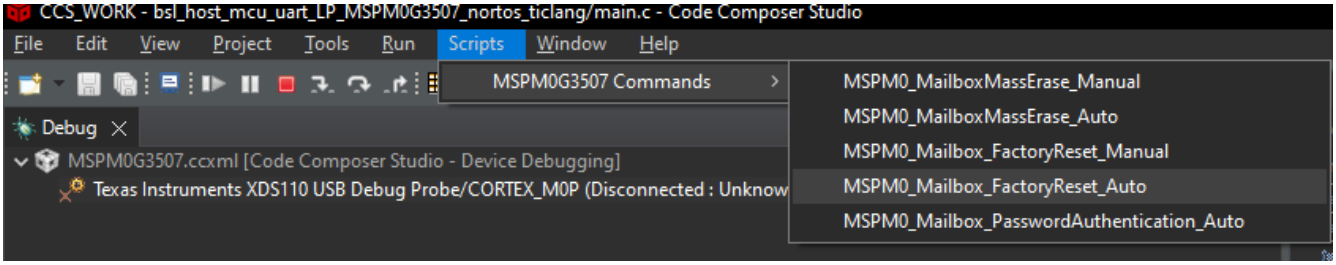

図 6-4. スクリプトでファクトリ リセットを実行

6. コンソールには、次の情報が表示されます:

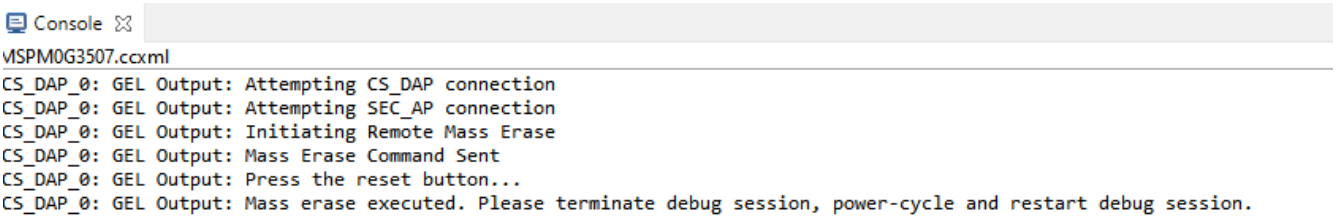

図 6-5. コンソールに情報を記録し

7. それでも動作しない場合は、デバイスを強制的に BSL モードにして、上記の手順 b ～ e を実行します。デバイスを強制的に BSL モードにするには、Non-main フラッシュで既定の BSL 呼び出しピン (PA18) を変更していない場合、デバイスの電源投入前に PA18 を High にし、そのまま保持します。Launchpad を使用する場合は、ボードを PC に接続するときに、PA18 に接続されたボタンを押し続けるだけで構いません。

7 参考資料

1. テキサス・インスツルメンツ: *MSPM0 G シリーズ 80MHz マイコン テクニカル リファレンス マニュアル*
2. テキサス・インスツルメンツ: *MSPM0 L シリーズ 32MHz マイコン テクニカル リファレンス マニュアル*
3. テキサス・インスツルメンツ: *MSPM0 ブートローダー ユーザー ガイド*

8 改訂履歴

Changes from Revision C (September 2024) to Revision D (September 2025)	Page
• ドキュメント全体にわたって表、図、相互参照の採番方法を更新	1
• ドキュメント全体にわたり、M0C1106 および M0H3216 に関する情報を追加	1
• BSL GUI exe ファイルのダウンロード リンクを追加	1

重要なお知らせと免責事項

テキサス・インスツルメンツは、技術データと信頼性データ (データシートを含みます)、設計リソース (リファレンス デザインを含みます)、アプリケーションや設計に関する各種アドバイス、Web ツール、安全性情報、その他のリソースを、欠陥が存在する可能性のある「現状のまま」提供しており、商品性および特定目的に対する適合性の黙示保証、第三者の知的財産権の非侵害保証を含むいかなる保証も、明示的または黙示的にかかわらず拒否します。

これらのリソースは、テキサス・インスツルメンツ製品を使用する設計の経験を積んだ開発者への提供を意図したものです。(1) お客様のアプリケーションに適した テキサス・インスツルメンツ製品の選定、(2) お客様のアプリケーションの設計、検証、試験、(3) お客様のアプリケーションに該当する各種規格や、その他のあらゆる安全性、セキュリティ、規制、または他の要件への確実な適合に関する責任を、お客様のみが単独で負うものとします。

上記の各種リソースは、予告なく変更される可能性があります。これらのリソースは、リソースで説明されている テキサス・インスツルメンツ製品を使用するアプリケーションの開発の目的でのみ、テキサス・インスツルメンツはその使用をお客様に許諾します。これらのリソースに関して、他の目的で複製することや掲載することは禁止されています。テキサス・インスツルメンツや第三者の知的財産権のライセンスが付与されている訳ではありません。お客様は、これらのリソースを自身で使用した結果発生するあらゆる申し立て、損害、費用、損失、責任について、テキサス・インスツルメンツおよびその代理人を完全に補償するものとし、テキサス・インスツルメンツは一切の責任を拒否します。

テキサス・インスツルメンツの製品は、テキサス・インスツルメンツの販売条件、または ti.com やかかる テキサス・インスツルメンツ製品の関連資料などのいずれかを通じて提供する適用可能な条項の下で提供されています。テキサス・インスツルメンツがこれらのリソースを提供することは、適用されるテキサス・インスツルメンツの保証または他の保証の放棄の拡大や変更を意味するものではありません。

お客様がいかなる追加条項または代替条項を提案した場合でも、テキサス・インスツルメンツはそれらに異議を唱え、拒否します。

郵送先住所:Texas Instruments, Post Office Box 655303, Dallas, Texas 75265

重要なお知らせと免責事項

TI は、技術データと信頼性データ (データシートを含みます)、設計リソース (リファレンス デザインを含みます)、アプリケーションや設計に関する各種アドバイス、Web ツール、安全性情報、その他のリソースを、欠陥が存在する可能性のある「現状のまま」提供しており、商品性および特定目的に対する適合性の黙示保証、第三者の知的財産権の非侵害保証を含むいかなる保証も、明示的または黙示的にかかわらず拒否します。

これらのリソースは、TI 製品を使用する設計の経験を積んだ開発者への提供を意図したものです。(1) お客様のアプリケーションに適した TI 製品の選定、(2) お客様のアプリケーションの設計、検証、試験、(3) お客様のアプリケーションに該当する各種規格や、その他のあらゆる安全性、セキュリティ、規制、または他の要件への確実な適合に関する責任を、お客様のみが単独で負うものとします。

上記の各種リソースは、予告なく変更される可能性があります。これらのリソースは、リソースで説明されている TI 製品を使用するアプリケーションの開発の目的でのみ、TI はその使用をお客様に許諾します。これらのリソースに関して、他の目的で複製することや掲載することは禁止されています。TI や第三者の知的財産権のライセンスが付与されている訳ではありません。お客様は、これらのリソースを自身で使用した結果発生するあらゆる申し立て、損害、費用、損失、責任について、TI およびその代理人を完全に補償するものとし、TI は一切の責任を拒否します。

TI の製品は、TI の販売条件 、 TI の総合的な品質ガイドライン 、 ti.com または TI 製品などに関連して提供される他の適用条件に従い提供されます。TI がこれらのリソースを提供することは、適用される TI の保証または他の保証の放棄の拡大や変更を意味するものではありません。 TI がカスタム、またはカスタマー仕様として明示的に指定していない限り、TI の製品は標準的なカタログに掲載される汎用機器です。

お客様がいかなる追加条項または代替条項を提案する場合も、TI はそれらに異議を唱え、拒否します。

最終更新日：2025 年 10 月