外部マイクロコントローラーを使用した PSoC™ 4 プログラミング (HSSP)

本書について

適用範囲と目的

AN84858 は、モジュール化された C コードを使用した外部マイクロコントローラーによる PSoC™ 4 のデバイス プログラミングの実装について説明します。ホスト ソース シリアル プログラミング (HSSP) と呼ばれるこのプロセスでは、ホスト マイクロコントローラーがシリアル ワイヤ デバッグ (SWD) インターフェースを介して PSoC™ 4 をプログラムします。C コードは、最小限の変更で任意のマイクロコントローラーにも移植できるように書かれていますので、PSoC™ 4 用の HSSP アプリケーション開発をスピードアップします。

対象者

本書の対象者は、電子システムの設計エンジニア, 技術者, および開発者です。

目次

本書について 1
目次 2
1 はじめに 4
1.1 用語および定義 5
2 HSSP ファームウェアのアーキテクチャ 6
2.1 SWD プロトコル物理層 7
2.2 SWD プロトコル パケット層 7
2.3 プログラミング データ フェッチ 7
2.4 HSSP プログラミング ステップ 8
2.5 HSSP タイムアウト パラメーター 9
2.6 HSSP プログラミング データ 9
2.7 メイン アプリケーション コード 9
2.8 HSSP エラー ステータス 10
3 HSSP プログラミングのハードウェア接続 12
4 ホスト プログラマへの HSSP アプリケーションの移植 13
4.1 移植対象のファイル 13
4.2 移植時に必要なコード変更 13
5 HSSP タイムアウト パラメーターの計算 15
5.1 DEVICE_ACQUIRE_TIMEOUT 15
5.2 SROM_POLLING_TIMEOUT 16
5.3 XRES_PULSE_100US 17
6 HSSP プログラミング データの受信インターフェース 19
7 HSSP タイミング検証 21
8 パワー サイクル モード プログラミング 22
9 サンプル プロジェクトのテスト 23
9.1 PSoC™ 5LP プログラマ搭載のキット (KitProg) 23
9.2 PSoC™ 5LP プログラマ搭載のキット (KitProg2) 26
10 HSSP 問題をデバッグするためのヒントとコツ 31
11 まとめ 32
12 添付プロジェクトの一覧 33
13 付録 A. Hex File Parser アプリケーション 34
13.1 Hex File Parser アプリケーションの使用方法 34
13.2 PSoC™ Creator のサンプル プロジェクトへの生成したファイルの追加 35
13.2.1 PSoC™ 4xxxL/PSoC™ 41x8_BLE/42x8_BLE デバイス ファミリ用の HexImage.c と HexImage.h ファイルの使用 37
14 付録 B. SROM 要求用のステータス コード 38
15 付録 C. HSSP エラー ステータス レジスタのビット フィールド定義 40
15.1 ビット[2:0]: SWD 認識応答 (SWD ACK[2:0]) 40
15.2 ビット 3: SWD 読み出しデータ パリティ エラー 40
15.3 ビット 4: ポート取得タイムアウト 40
15.4 ビット 5: SROM ポーリング タイムアウト エラー 41
15.5 ビット 6: 検証失敗 41
15.6 ビット 7: 遷移エラー 41
16 付録 D. HSSP 関数 42
関連資料 45

目次

改訂履歴 46
免責事項 47

1 はじめに

PSoC™ 4 デバイス プログラミングは、特に外部ホスト プログラマを使用する、PSoC™ 4 内の不揮発性メモリのプログラミングを指します。ホストは、インフィニオンが提供するプログラマ (CY8CKIT-002 MiniProg3)、サードパーティーのプログラマ、あるいはカスタム プログラマ (例えば、基板に搭載したマイクロコントローラー) です。本アプリケーション ノートは、PSoC™ 4 デバイスをプログラムするためのホスト プログラマの実装方法について説明します。PSoC™ 4 アーキテクチャの詳細情報および PSoC™ Creator™ソフトウェアを使って PSoC™ 4 のプロジェクトを作成する方法は、「AN79953 – Getting started with PSoC™ 4」を参照してください。

プログラマは、デバイスのプログラム (フラッシュ) メモリにバイナリ プログラム (16 進ファイル) を保存するハードウェア ソフトウェア システムです。プログラマは、エンジニアのプロトタイピング環境に不可欠なコンポーネント、または製造環境 (大量プログラミング) の不可欠な要素です。Figure 1 に、開発環境の概要を示します。

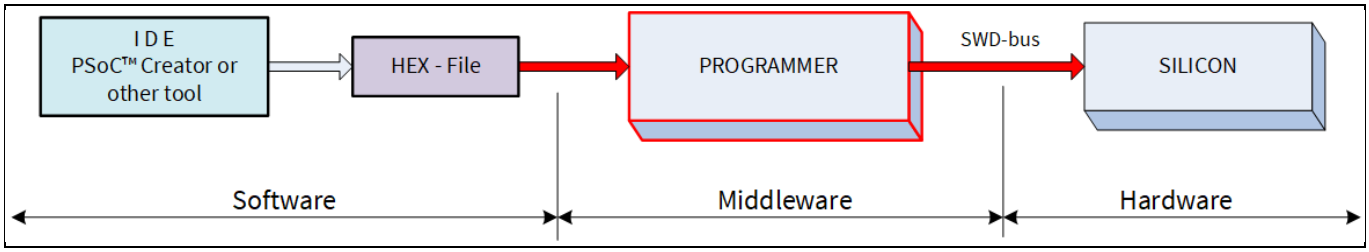


Figure 1　開発環境におけるプログラマ

製造環境では、統合開発環境 (IDE) ブロックは存在しません。その主な目的は 16 進ファイルを作成することだからです。プログラマは、次の 3 つの機能を実行します。

- 16 進ファイルを解析し、必要な情報を抽出する
- シリアル ワイヤ デバッグ (SWD) マスターとしてシリコンに接続する
- 16 進データを SWD 信号に変換してプログラミング アルゴリズムを実装する

このドキュメントでは、プログラマの具体的な実装については説明しません。データ フロー、物理接続、およびアルゴリズムに焦点を当てています。具体的には、プログラマの 3 つの機能に対応する次のトピックを取り上げます。

- プログラムするデータ
- チップとのインターフェース
- ターゲット デバイスのプログラムに使用するアルゴリズム

詳細については、PSoC™ programming solution を参照してください。

1.1 用語および定義

- **シリアル ワイヤ デバッグ (SWD):** Arm®社の開発製品である SWD プロトコルは、SWDCLK (クロック) と SWDIO(双方向データ ライン) という 2 本の信号だけを使ってプログラムとデバッグを行います。
- **デバッグ アクセス ポート (DAP):** DAP は SWD と PSoC™ 4 内の CPU 間のプログラム/デバッグ インターフェースです。これはデバッグ ポート (DP) とアクセス ポート (AP) を含みます。
  - DP はプログラマ/デバッガへの物理接続を提供します。
  - AP は、DAP モジュールと Cortex®-M0 CPU、フラッシュメモリなどとの間のインターフェースを提供します。
- **HSSP:** HSSP はホスト マイクロコントローラーを使用して基板上のターゲット デバイスをプログラムすることを指します。ターゲットは SWD インターフェースを介してプログラムされます。本アプリケーション ノートでは、HSSP はビットバンギングの実装を使用してターゲット デバイスをプログラムします。ビット バンギング プログラミングとは、ホストプログラマに在るソフトウェア コードを使ってプログラミング ピンを操作する技術を指します。
- ブートロードと HSSP の違い**:** 組込みシステムでは、ブートローダはシステム ファームウェアの更新にも使用されます。ブートローディングと HSSP の主な違いは次のとおりです。
  - ブートローダは、標準的な通信プロトコルを介してデバイスのフラッシュ メモリを更新するために使用されます。ブートローダでは、ブートローダブル領域と指定されているフラッシュ メモリの特定部分のみを更新できます。
  - 一方、HSSP はターゲット デバイスのフラッシュ メモリの完全なプログラミングをサポートします。
  - ブートローダは、任意の標準的な通信インターフェース (例えば USB, I^2C, SPI, および UART) を使用してファームウェアを更新できますが、HSSP は SWD または JTAG インターフェースを使用してフラッシュ メモリをプログラムします。PSoC™ 4 は SWD インターフェースのみに対応します。

2 HSSP ファームウェアのアーキテクチャ

ターゲット デバイス用の HSSP は、モジュール化された C コードを使用して複数の層で実装されています。これらの階層は以下のとおりです。

- SWD Protocol Physical Layer (SWD プロトコル物理層)
- SWD Protocol Packet Layer (SWD プロトコル パケット層)
- HSSP Programming Steps Layer (HSSP プログラミング ステップ層)

各階層間の制御フローについては、Figure 2 を参照してください。

Note: このファームウェアの実装については、本アプリケーション ノートに添付されている、PSoC™ 5LP を外部のホストプログラマとして使用する A_Hssp_Programmer プロジェクトを参照してください。

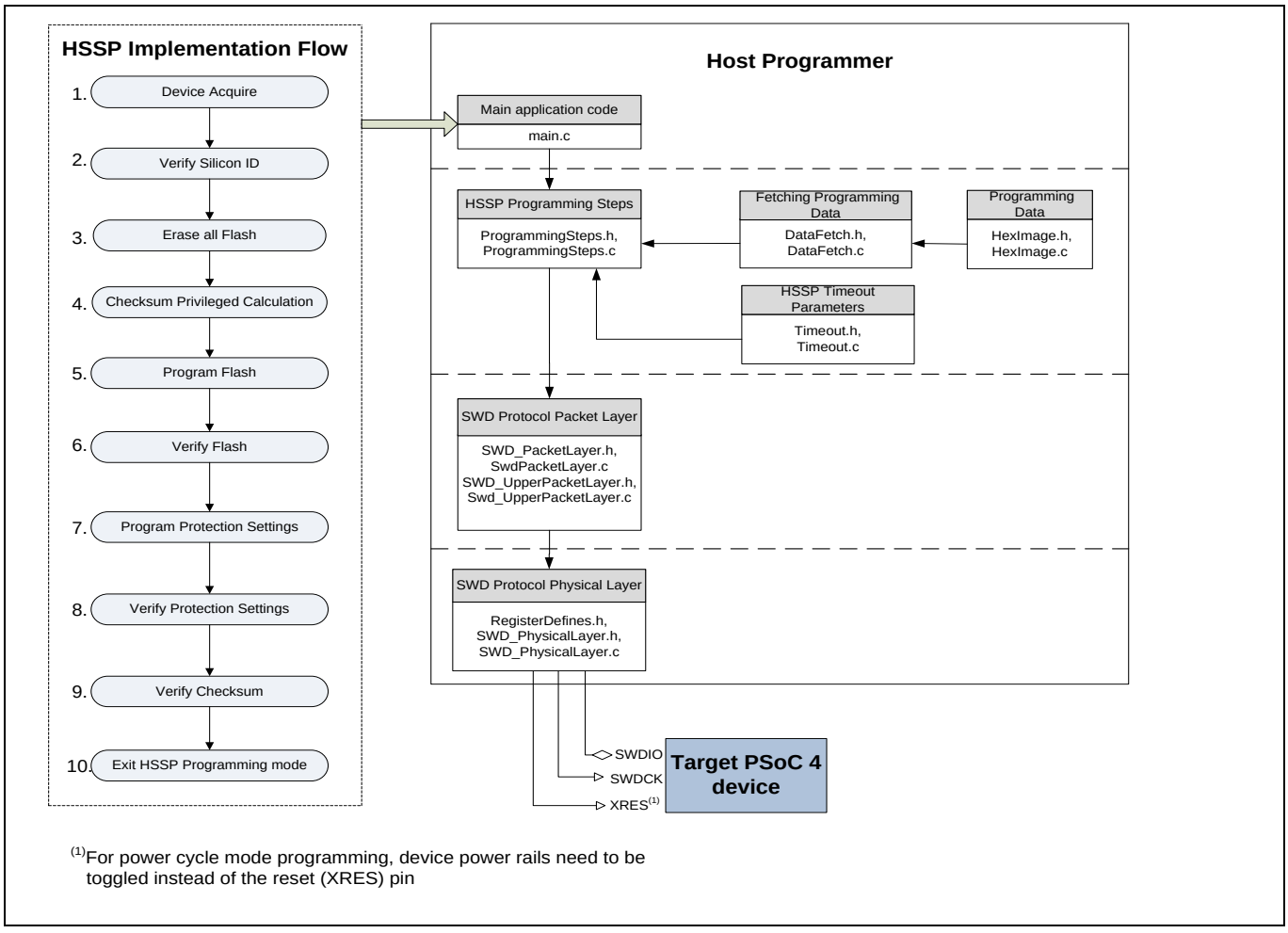


Figure 2 HSSP ファームウェア アーキテクチャ

HSSP プログラミング ステップ層は、「Fetching Programming Data」(プログラミング データ フェッチ) インターフェースを介してデータのソース (例えば I^2C, SPI, UART, USB など任意の通信インターフェースまたは (Figure 2 に示すように) ホストフラッシュによって提供された HEX ファイル データ) からプログラミング データを抽出します。また、HSSP プログラミングステップ層は、「HSSP Timeout Parameters」(HSSP タイムアウト パラメーター) インターフェースを使用して、プログラミング API 内でタイムアウトを設定します。

Figure 2 に示すファームウェア アーキテクチャで使用されるすべての層は、次の節で説明します。

2.1 SWD プロトコル物理層

SWD プロトコル物理層を構成するファイルは Table 1 に記載されています。

Table 1 SWD プロトコル物理層のファイル

ソース ファイル	説明
RegisterDefines (*.h* ファイル)	このファイルは、プログラミング ピンのポート番号、ピン番号、入力/出力レジスタ、および駆動モード レジスタを定義します。
SWD_PhysicalLayer (*.c* および *.h* ファイル)	これらのファイルはプログラミング ピンを操作するマクロと関数を含みます。ピンは *RegisterDefines.h* ファイル内で定義されています。

上記のファイル内のコードは、ホスト マイクロコントローラーとしての PSoC™ 5LP 向けに書かれています。これらのファイルを他のマイクロコントローラーに移植する場合、すべての関数とマクロを適切に修正する必要があります。

Note: ホスト側のピン配置の詳細は関連資料に記載されている各デバイスの該当プログラミング仕様書の「Pin Names and Requirements」を参照してください。

2.2 SWD プロトコル パケット層

SWD プロトコル パケット層を構成するファイルを Table 2 に示します。

Table 2 SWD プロトコル パケット層のファイル

ソース ファイル	説明
SWD_PacketLayer (*.c* および *.h* ファイル)	これらのファイルは、SWD プロトコルに従って SWD 読み出しパケットと SWD 書き込みパケットを送信するためのパケット ルーチンを定義します。
SWD_UpperPacketLayer (*.c* および *.h* ファイル)	これらのファイルは、SWD_PacketLayer 内で定義された関数を使って、DAP レジスタと CPU アドレス空間を直接読み書きする関数を実装します。

この層で定義された関数は *ProgrammingSteps.c* ファイル内の関数から直接呼び出されます。

Figure 1 に示すように、これらすべての SWD パケット関数は、最上層のファイル内の関数でアクセスされる `swd_PacketHeader`、`swd_PacketAck` および `swd_PacketData[]` という 3 つのグローバル変数を使用します。

2.3 プログラミング データ フェッチ

上位層の関数にプログラミング データをフェッチするために使用するファイルを Table 3 に示します。

Table 3 データフェッチ層のファイル

ソース ファイル	説明
DataFetch (*.c* および *.h* ファイル)	これらのファイルは、プログラミング データを *HexImage.c* ファイルからフェッチして *ProgrammingSteps.c* ファイル内の関数に渡すルーチンを含みます。 プログラミング データは、フラッシュ行データ, フラッシュ保護データ, チップ保護データ, シリコン ID, チェックサム, およびフラッシュ行の総数を含みます。

HSSP プログラミング データを取得する方法に基づいて、関数の定義を変更します。

Note: HSSP ソース コードを変更する方法についての情報は、HSSP プログラミング データの受信インターフェースを参照してください。

2.4 HSSP プログラミング ステップ

この層には *ProgrammingSteps.c* と *ProgrammingSteps.h* と名付けられたファイルがあります。これらのファイルは、HSSP アプリケーションのトップレベルの関数を含みます。これらの関数は下記のとおりに説明します。

1. **デバイスの取得**: このステップでは、デバイス リセットの後に、特定のシーケンスを SWD インターフェースを介して送信することでデバイスが取得されます。結果として、ホスト プログラマは Cortex M0-CPU、および SRAM やレジスタなどの他のシステム リソースを制御できます。PSoC™ 40xx、PSoC™ 4xx7_BLE、PSoC™ 40xxS、PSoC™ 41xxS および PSoC™ 4100PS デバイス ファミリは、フラッシュ消去/書き込み動作の前に内部主発振器 (IMO) の周波数を 48MHz に設定する必要があります。この動作は、これらのデバイス用のデバイス取得ルーチンにも含まれています。

Note: PSoC™ 4000 ファミリのいくつかのデバイスは専用リセット (XRES) ピンがないため、デバイスをリセットさせるためにデバイスのパワー レールをトグルする必要があります。これをパワー サイクル モード プログラミングと呼びます。パワー サイクル モード プログラミングに対応するためにコードを修正するために必要な変更の詳細は、パワー サイクル モード プログラミングを参照してください。

2. **シリコン ID の検証**: このステップでは、取得したデバイスが、HEX ファイルが生成されたものと同じであることを検証します。
3. **全フラッシュの消去:** このステップでは、すべてのユーザー行および対応するフラッシュ保護設定を消去します。
4. **特権行のチェックサム計算**: すべてのユーザー行を消去した後、このステップでは、特権行のチェックサムを計算します。このチェックサムは、ステップ 9 でユーザー行のチェックサムを検証するために使用します。
5. **フラッシュのプログラム**: このステップでは、HEX ファイル内のプログラミング データと SROM API 呼び出しを使用してフラッシュをプログラムします。
6. **フラッシュの検証**: このステップでは、前のステップで HEX ファイル内のデータでプログラムされたフラッシュのデータを検証します。このステップはオプションですが、強く推奨します。
7. **保護設定のプログラム:** このステップでは、HEX ファイルからの行保護設定とチップ保護設定を、特定のフラッシュ領域に書き込みます。
8. **保護設定の検証:** 両方の保護設定が HEX ファイル内の設定と一致しているかを検証します。
9. **チェックサムの検証**: このステップでは、フラッシュ内のユーザー データのチェックサムが HEX ファイル内のチェックサムと一致しているかを検証します。これはステップ 4 で計算した特権行のチェックサムを使用します。
10. **HSSP プログラミング モードの終了:** このステップでは、ターゲット デバイスをプログラミング モードから解放します。

これらの各ステップは、基本 SWD 命令で構成される関数によって記述されています。詳細は関連資料に記載されている各デバイスの該当プログラミング仕様書を参照してください。

ProgrammingSteps.h ファイル内で宣言された関数は、3 つの層 (SWD プロトコル パケット層、データ フェッチ層、タイムアウト層) から関数、定義、およびグローバル変数にアクセスします。*ProgrammingSteps.c* と *ProgrammingSteps.h* のファイルで定義された関数は、ターゲット デバイスのプログラミングに必要なすべてのステップをカバーします。

2.5 HSSP タイムアウト パラメーター

Table 4 タイムアウト層のファイル

ソース ファイル	説明
Timeout (*.c* および *.h* ファイル)	これらのファイルは、HSSP で使用するタイムスタンプの定義および遅延ルーチンを含みます。

タイムスタンプの定義はデバイスのプログラミング仕様書に記載されている電気的タイミング仕様から派生します。*Timeout.h* ファイル内のこれらのタイムスタンプ パラメーター定義の値は、63MHz のクロック周波数で動作する PSoC™ 5LP ホスト プログラマ向けのものです。

タイムスタンプ定義と遅延ルーチンは *ProgrammingSteps.c* ファイルの関数定義に使用されます。

特定のホスト プログラマのタイムスタンプ パラメーターを計算する方法については、HSSP タイムアウト パラメーターの計算を参照してください。

2.6 HSSP プログラミング データ

ホスト プログラマに格納するプログラミング データを含むファイルを Table 5 に示します。

Table 5 プログラミング データを含むファイル

ソース ファイル	説明
HexImage (*.c* および *.h* ファイル)	これらのファイルはターゲット デバイスにプログラムするデータを含みます。また、シリコン ID, チェックサム, フラッシュ行の総数, およびフラッシュ行あたりのバイト数などの HSSP プログラミングに使用するターゲット デバイスのパラメーターも格納しています。

これらのファイルにあるデータは定数の配列として PSoC™ 5LP のフラッシュ メモリに格納されます。

これらのファイルは、本アプリケーション ノートで提供された Hex File Parser という C#アプリケーションによって生成されます。このアプリケーションでは、ターゲット デバイスの HEX ファイルを入力として受けることによって、これらのファイルを生成します。このアプリケーションの詳細は、付録 A に記載されています。

HEX ファイル フォーマットの詳細については、本書の関連資料に記載される各デバイスの該当プログラミング仕様書の「Appendix B. Intel Hex File Format」を参照してください。

プログラミング データを内蔵メモリに格納するためのメモリ容量が足りないホスト プログラマには、*HexImage.c* と *HexImage.h* ファイルは不要です。その場合、HSSP プログラミング データは通常、I^2C、UART や USB などの通信インターフェースを介してパケットとしてホストに送信されます。

Note: プログラミング データの取得方法に従って HSSP ソース コードを修正する詳細情報については、HSSP プログラミング データの受信インターフェースを参照してください。

2.7 メイン アプリケーション コード

main.c ファイルは、Figure 1 に示す順番で、トップレベルの HSSP プログラミング ステップを呼び出すメインのアプリケーションコードです。*main.c* 内の `ProgramDevice()`関数はすべてのステップを実行します。次のステップに進む前に、各ステップが正常に実行されなければいけません。

どのステップの後にも、FAILURE が返されると、HSSP 動作は中止します。`ReadHsspErrorStatus()` 関数で返されたエラー ステータスが、エラーの原因を特定するために使用されます。HSSP 動作の状態はエラー ステータス レジスタとともに、添付している HSSP プロジェクトのキャラクタ LCD 上に表示されます。

Note: キャラクタ LCD と `Pin_Start` ルーチンは、PSoC™ 5LP のホスト プログラムに固有のものであり、他のホスト プログラマでは必要に応じて変更する必要があります。

2.8 HSSP エラー ステータス

HSSP アプリケーションのトップレベル ステップのいずれかが Failure 状態を返したとき、`ReadHsspErrorStatus()`関数がメイン アプリケーション コードから呼び出され、エラーの詳細を取得します。

この関数はステータス バイトを返します。このステータス バイトを使って、ビット フィールド定義からエラー詳細を推測します。この関数によって返されたビット フィールド値については Figure 3 を参照してください。

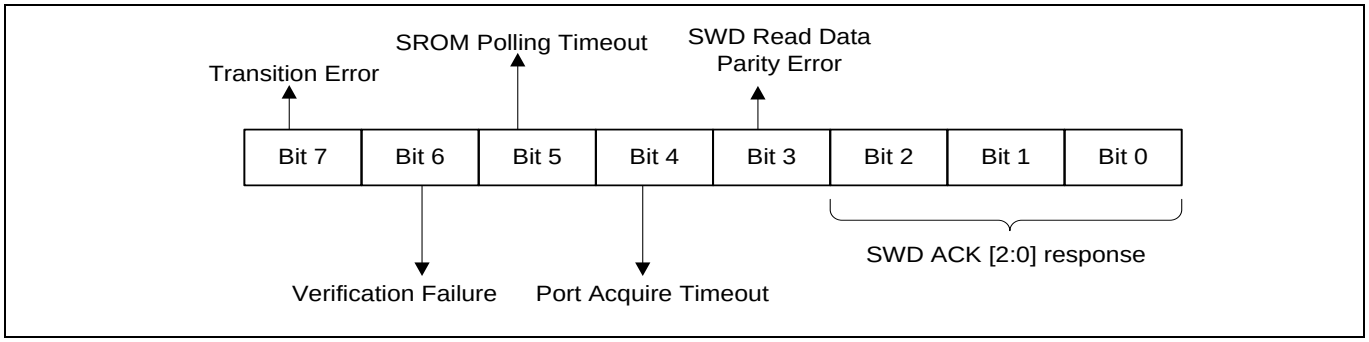


Figure 3 HSSP エラー ステータス レジスタ

Table 6 はこのステータス レジスタのビット フィールド定義を説明します。

Table 6 HSSP エラー ステータス レジスタ

ビット	フィールド名	説明
[2:0]	SWD ACK	これらの 3 ビットは前回の SWD トランザクションの確認応答を格納します。
3	SWD Read Data Parity Error (SWD 読み出しデータ パリティ エラー)	このビットがセットされていれば、ホストで受信したデータ内のパリティ エラーが発生したことを示します。
4	Port Acquire Timeout (ポート取得タイムアウト)	このビットがセットされていれば、デバイスがタイムアウト時間内で取得されなかったことを示します。
5	SROM Polling Timeout (SROM ポーリング タイムアウト)	このビットがセットされていれば、SROM の動作が 1 秒を超えたことを示します。
6	Verification Failure (検証失敗)	このビットは複数のステップでセットされます。失敗が発生したステップに応じて、その原因を推測できます。
7	Transition Error (遷移エラー)	このビットがセットされていれば、チップおよび HEX ファイルから読み出されたチップ保護設定が誤った遷移を示すことを示します。

このレジスタおよびビット フィールドの詳細は付録 C. HSSP エラー ステータス レジスタのビット フィールドの定義を参照してください。

3 HSSP プログラミングのハードウェア接続

PSoC™ 4 の HSSP プログラミングは、SWD インターフェース ピン (SWDIO と SWDCK) および外部リセット ピン (XRES) を使用します。ホスト プログラマのピン駆動モードの要件は、本書の関連資料に記載されているプログラミング仕様書の「Physical Layer」で説明されています。Figure 4 は、ホスト プログラマとターゲット デバイス間に必要となる基本ハードウェア接続を示します。

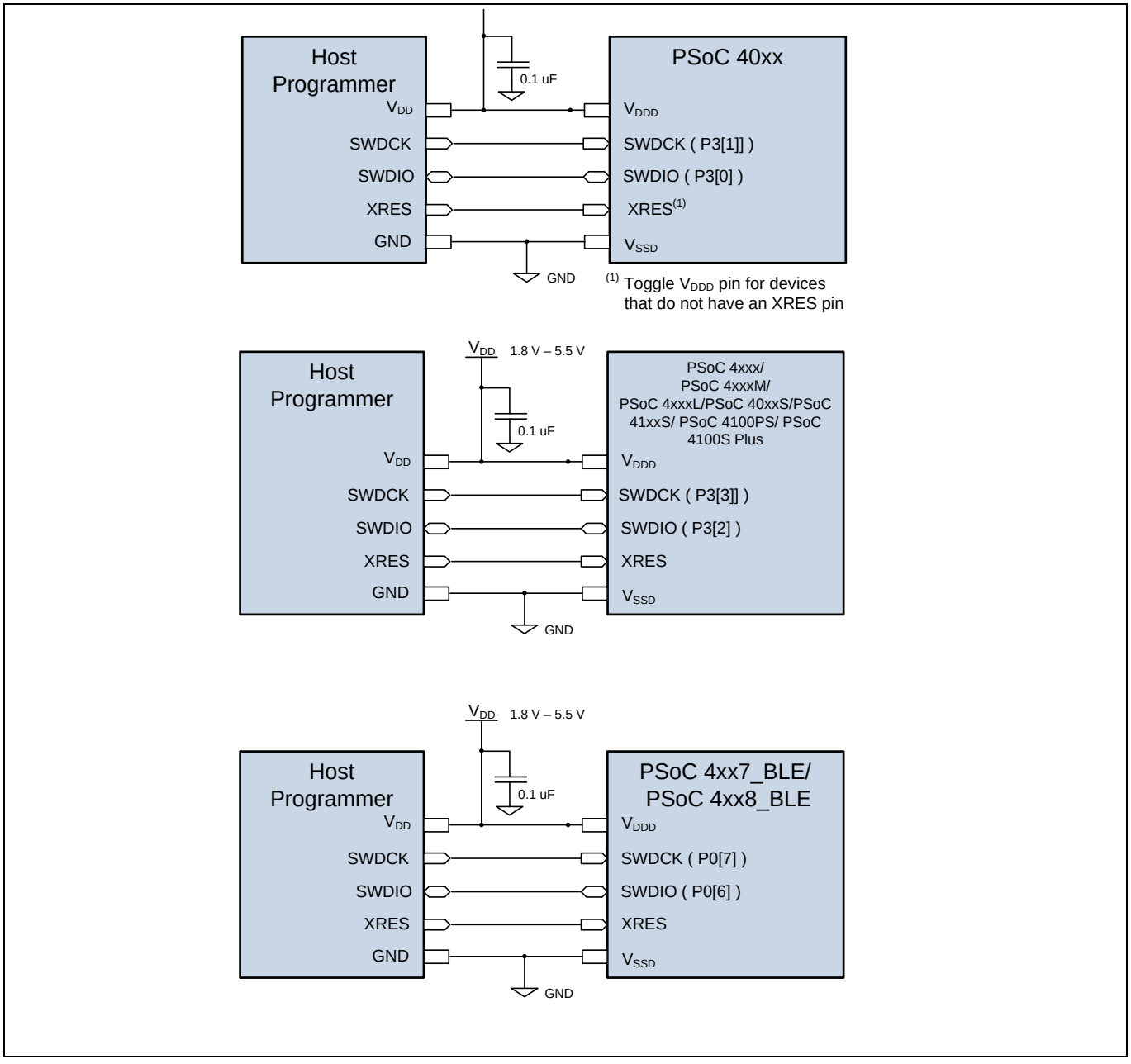


Figure 4 ホスト/ターゲットの基本接続

4 ホスト プログラマへの HSSP アプリケーションの移植

本アプリケーション ノートで提供された A_Hssp_Programmer プロジェクトは、PSoC™ 5LP をターゲット デバイスのホストプログラマとして使用します。HSSP アプリケーションでは、ホスト プログラマは任意のマイクロコントローラーにすることができます。本節は、ターゲット デバイスをプログラムするために使用される特定のホストに HSSP アプリケーション コードを移植するために必要な変更を説明します。

4.1 移植対象のファイル

PSoC™ 5LP のホスト プログラマに基づいたプロジェクトは PSoC™ 5LP に固有のファイルを含みます。HSSP アプリケーション コードを他のホスト プログラマに移植する際には、Table 7 に記載されているファイルのみを移植してください。

Table 7 移植対象のファイル

移植対象のヘッダ ファイル	移植対象のソース ファイル
SWD_PhysicalLayer.h	SWD_PhysicalLayer.c
SWD_PacketLayer.h	SWD_PacketLayer.c
SWD_UpperPacketLayer.h	SWD_UpperPacketLayer.c
Timeout.h	Timeout.c
HexImage.h	HexImage.c
DataFetch.h	DataFetch.c
ProgrammingSteps.h	ProgrammingSteps.c
RegisterDefines.h	

4.2 移植時に必要なコード変更

添付の HSSP アプリケーション コードを PSoC™ 5LP 以外のホスト プログラマに移植する際には、各ファイルに対して次の変更を行ってください。移植時に修正が必要なファイルは僅かであることに注意してください。

- ***RegisterDefines.h***: 使用するホスト プログラマに応じて、ヘッダ ファイル内のポート番号、ピン番号、マスク値、出力レジスタ、入力レジスタおよび駆動モード レジスタの定義を変更します。
- ***SWD_PhysicalLayer.h***: ヘッダ ファイル内で定義されたすべてのビット バンギング マクロは、*SWD_PhysicalLayer.c* ファイル内の関数定義に使用されます。使用するホスト プログラマに応じてこれらのマクロを変更します。
- ***SWD_PhysicalLayer.c***: 使用するホスト プログラマに応じてこのファイルで定義されたすべてのビット バンギング関数を変更します。
- ***Timeout.h***: 使用するホスト プログラマに応じて、Table 8 で説明した 3 つのタイムアウト パラメーター定義を変更します。

Table 8 タイミング パラメーター

No.	タイミング パラメーター
1	XRES_PULSE_100US
2	DEVICE_ACQUIRE_TIMEOUT
3	SROM_POLLING_TIMEOUT

特定のホスト プログラマ用のタイムアウト パラメーターを計算する方法については、HSSP タイムアウト パラメーターの計算を参照してください。

- ***HexImage.c*、*HexImage.h***: これらのファイルは、定数の配列として定義された、ターゲット デバイスにプログラムするデータを含みます。PSoC™ 5LP ホスト プログラマの場合、プログラムするデータは、ホスト PSoC™ 5LP のフラッシュ メモリに格納されます。

ホスト プログラマによっては、プログラミング データを内蔵メモリに格納するための容量が足りない場合があります。その代わりに、USB、SPI や UART などの通信インターフェースを使用してプログラミング データを取得できます。その場合、これらのファイルを削除してください。

- ***DataFetch.c***: 関数の定義は、プログラミング データを取得する方法に基づいて変更しなければなりません。

プログラミング データの取得方法に従って HSSP ソース コードを修正する詳細情報については、HSSP プログラミング データの受信インターフェースを参照してください。

- ***main.c***: *main.c* ファイル内のキャラクタ LCD と Pin_Start ピン用の API は、PSoC™ 5LP のホスト プログラマに固有のものです。したがって、他のホスト プログラマに移植する際には、それらを削除するか修正する必要があります。

上記で指定されたすべてのファイル内では、次のコメントはホスト プログラマに基づいて修正する必要のあるコードの前に置かれています。これは、修正するコードを特定する際に役立ちます。

```
/******USER ATTENTION REQUIRED******/
/******HOST PROCESSOR SPECIFIC******/
```

5 HSSP タイムアウト パラメーターの計算

使用するホスト プログラマに応じて、*Timeout.h* 内で定義したタイムアウト パラメーターの値を変更します。

「C_Hssp_TimeoutCalc」という別のテスト プロジェクトが本アプリケーション ノートで提供されています。このプロジェクトは、PSoC™ 5LP ホスト プログラマ用のタイムアウト パラメーターの計算手順を示します。他のホスト プログラマの場合、タイムアウト値を計算するために、これと同様なテスト プロジェクトを作成してください。

テスト プロジェクトは、*TimeoutCalc.h* と *TimeoutCalc.c* の 2 つのファイルでテスト関数を提供します。これらのテスト関数は、コード実行中、テスト ピンをトグルし続けます。タイムアウト パラメーターは、オシロスコープを使ってテスト ピン上の信号の LOW パルス幅を測定することで測定されます。

これらのタイムアウト パラメーターを計算するには、プロジェクトの *TimeoutCalc.h* ヘッダ ファイルにあるマクロ定義の説明を参照してください。それぞれのタイムアウト値の重要性を検討して見ましょう。

5.1 DEVICE_ACQUIRE_TIMEOUT

このマクロは *ProgrammingSteps.c* ファイル内の `DeviceAcquire()` 関数で使用されます。デバイスをプログラムするには、XRES ピンを使用してデバイスをリセットした後、デバイス取得最大時間ウィンドウ内にデバイスを取得する必要があります。Table 9 に、デバイスを取得するための最大時間ウィンドウを定義します。

Table 9 デバイス取得用のタイムアウト値

デバイス ファミリ	タイムアウト (ms)[1]
PSoC™40xx	2.0
PSoC™41/42xx	1.5
PSoC™4xxxM	2.0
PSoC™ 4xxxL	2.0
PSoC™4xx7_BLE/PSoC™4xx8_BLE	2.0
PSoC™ 4000S/PSoC™ 4100S/PSoC™ 4100PS	2.0
PSoC™ 4100S Plus	5.0

Note: デバイス取得のタイミング要件を満たす SWDCK クロックの推奨最小周波数は 1.5MHz です。詳細については、関連資料に記載されている各デバイスの該当プログラミング仕様書の「Step 1. Acquire Chip」項を参照してください。

デバイス取得シーケンスには、2 ステップがあります。

1. デバッグ アクセス ポート (DAP) をリセットするための標準 Arm®コマンドであるラインのリセットを実行します。
2. DAP を読み出します。

1 パワー サイクル モード プログラミングの場合、デバイス取得用のタイムアウトは、より長くなります(約 30ms)。

Table 9 に示すように、DEVICE_ACQUIRE_TIMEOUT は、デバイス リセット後の特定の時間ウィンドウの間にデバイス取得シーケンスを送信できる最大回数を示します。

このマクロを計算するには、プロジェクトの *main.c* ファイルの `TestModeTimeout()` 関数のコメントを削除して、デバイスをプログラムします。

デバイス取得シーケンスの 1 回の反復に対応したテスト ピン上の LOW パルス幅をオシロスコープを使って測定します。その後、次のようにマクロを計算します。

DEVICE_ACQUIRE_TIMEOUT = (Xms/LOW パルス幅)。X は、Table 9 に示すようにデバイス取得の最大時間ウィンドウです。

Code Listing 1　DEVICE_ACQUIRE_TIMEOUT パラメーターの計算

```
Unsignedshort timestamp = 0;
Unsignedlongchip_DAP_Id = 0;

/* Make the pin LOW before sending SWD clock train */
TESTPIN_OUTPUT_LOW;

for(timestamp = 0; timestamp < 1; timestamp++)
{
       SwdLineReset()
       Read_DAP(DPACC_DP_IDCODE_READ, &chip_DAP_Id);
}
/* Make the pin HIGH after sending SWD clock train */
TESTPIN_OUTPUT_HIGH;
```

5.2　SROM_POLLING_TIMEOUT

このマクロは HSSP プログラミング中に SROM ポーリング動作に使用されます。これは、ターゲット デバイス内で SROM 要求による不揮発性メモリの読み書き動作の結果をポーリングするときに使用されます。

ホストは、SWD インターフェースを介して SROM システム コールを要求したとき、ステータスを得るために CPUSS_SYSREQ レジスタを読み出す中に最大 1 秒待機します。応答が受けられなかった場合、ホストは HSSP 動作を中止します。

SROM_POLLING_TIMEOUT は、CPUSS_SYSREQ レジスタが 1 秒間に読み出せる最大回数を示します。

このマクロを計算するには、プロジェクトの *main.c* ファイルの `TestSromPollingTimeout()` 関数のコメントを削除して、デバイスをプログラムします。

SROM ポーリング シーケンスの 10 回の反復を送信した後、オシロスコープを使ってテスト ピン上の LOW パルス幅を測定します。その後、次のようにマクロを計算します。

SROM_POLLING_TIMEOUT = (1s/LOW パルス幅)*10

Code Listing 2　SROM_POLLING_TIMEOUT パラメーターの計算

```
Unsignedshort timestamp = 0;
Unsignedlongstatuscode = 0;

/* Make the pin low before sending SWD clock train */
TESTPIN_OUTPUT_LOW;

for(timestamp = 0; timestamp < 10; timestamp++)
{
       Read_IO (CPUSS_SYSREQ, &statusCode);

/*performing SROM_SYSREQ_BIT | SROM_PRIVILEGED_BIT */
statusCode &= (SROM_SYSREQ_BIT | SROM_PRIVILEGED_BIT);
}
/* Make the pin high after sending SWD clock train */
TESTPIN_OUTPUT_HIGH;
```

このマクロは、1 秒の間に CPUSS_SYSREQ を読み出して SROM_SYSREQ_BIT と SROM_PRIVILEGED_BIT が 0 にセットされているかをチェックする回数を示します。

5.3　XRES_PULSE_100US

ターゲット デバイスをリセットするには、ホストは XRES ラインに最小パルス幅 5µs のアクティブ LOW 信号を生成します。HSSP コードでは、100µs のパルス幅が XRES ピン上に生成されます。

`TestDelayHundredUs()` 関数は *TimeoutCalc.c* ファイルで定義されています。

本アプリケーション ノートで提供された C_Hssp_TimeoutCalc プロジェクトでは、この関数は `CyDelayUs()`API を使用して実装されます。PSoC™以外のマイクロコントローラーをホスト プログラマとして使用してこの関数を実装するためには、*TimeoutCalc.h* 内の次の行のコメントを削除します。

```
#define XRES_PULSE_100US 600
```

Code Listing 3 に示すように、*TimeoutCalc.c* で定義された `TestDelayHundredUs()`のコードを変更します。

このマクロを計算するには、プロジェクトの main.c ファイルにある *`TestDelayHundredUs()`*関数のコメントを削除して、デバイスをプログラムします。オシロスコープでテスト ピンのパルス幅を測定します。

Code Listing 3　遅延ルーチン

```
unsignedshort timestamp;

/* Make the pin low before start of the delay */
```

```
TESTPIN_OUTPUT_LOW;

/* For loop to introduce the 100 us delay
for(timestamp = 0; timestamp < XRES_PULSE_100US; timestamp++)
{
       /* Do Nothing */
}

/* Make the pin high after end of the delay */
TESTPIN_OUTPUT_HIGH;
```

A_HSSP_Programmer および B_HSSP_Pioneer プロジェクトは、`CyDelayUs()`API を使用して 100µs の遅延を実装します。PSoC™以外のマイクロコントローラーをホスト プログラマとして使用する場場合、*Timeout.h* に `XRES_PULSE_100US` パラメーターを定義し、*Timeout.c* の `DelayHundredUs` 関数でこのパラメーターを使用して遅延ルーチンを実装する必要があります。

パワー サイクル モード プログラミングの場合、遅延ルーチンはデバイス パワー レールがオフである時間を示します。パワー サイクル モード プログラミングでは、このパルス幅要件はありません。パワー レールをオンにする前に 1ms の遅延を取ることを推奨します。

6 HSSP プログラミング データの受信インターフェース

DataFetch.c と *DataFetch.h* ファイルは、ターゲット デバイスにプログラムするデータをフェッチする関数を含みます。

サンプル プロジェクトでは、プログラミング データは PSoC™ 5LP ホスト プログラマの内蔵フラッシュ メモリの *HexImage.c* と *HexImage.h* ファイルに格納されています。データ フェッチ ルーチンは、PSoC™ 5LP フラッシュ メモリからこのデータをアクセスして HSSP を実行します。

ただし、すべてのホスト プログラマが、HSSP プログラミング データを格納するための内蔵メモリを持っているわけではありません。この場合には、プログラマは USB、SPI や UART などの通信インターフェースを使用してプログラミング データを取得できます。また、*DataFetch.c* ファイルのすべての関数定義を適切に変更する必要があります。

次の例は、`Hex_ReadRowData()` 関数に必要な変更を示す参考例です。他の関数も同様に変更できます。

オリジナル コード

Code Listing 4　フラッシュ データを取得するオリジナル コード

```
void HEX_ReadRowData(unsignedshortrowCount, unsignedchar * rowData)
{
/* Maximum value of 'i' can be 256 */
unsignedshort i;

for(i = 0; i<BYTES_PER_FLASH_ROW; i++)
    {
rowData[i] = FlashData_HexFile[rowCount][i];
    }
}
```

変更済みコード

プログラミング データが通信インターフェースを介して受信された場合、変更済みコードは次のようになります。

Code Listing 5　フラッシュ データを取得する変更済みコード

```
void HEX_ReadRowData(unsignedshortrowCount, unsignedchar * rowData)
{
/* Maximum value of 'i' can be 256 */
unsignedshort i;

/* ADD WAITING CODE HERE FOR THE UART BUFFER TO GET THE FLASH DATA */
```

Code Listing 5　フラッシュ データを取得する変更済みコード

```
for(i = 0; i<BYTES_PER_FLASH_ROW; i++)
    {
rowData[i] = /* PLACE THE UART BUFFER ARRAY HERE */
    }
}
```

7 HSSP タイミング検証

ホスト プログラマは、堅牢な HSSP 実装を実現するために、ターゲット デバイス プログラミングのタイミング仕様を満たさなければなりません。これらの仕様は、関連資料に掲載されている各デバイスの該当プログラミング仕様書の「Appendix D. Timing Specifications of the SWD Interface」に記載されています。

ホスト プログラマは、SWD インターフェースとプログラミング モード開始用に指定されたタイミング パラメーターの仕様を満たさなければなりません。タイミングを検証するには、オシロスコープで SWDIO, SWDCK, および XRES 信号を取り込みます。パワー サイクル モード プログラミングでは、デバイス パワー レールは XRES ピンの代わりに監視する必要があります。取り込んだ波形を使用して、プログラミング仕様で提供された該当値に対してタイミング パラメーターを検証できます。

8 パワー サイクル モード プログラミング

デバイスをプログラムする際、まずデバイスをリセットして取得し、プログラミング モードに移行させます。ホスト側からデバイスをリセットするために、デバイスの XRES ピンをトグルすることを推奨します。ただし、PSoC™ 4000 ファミリの少ピン数デバイスには XRES ピンを備えていないものがあるため、デバイスをリセットするためにホストはデバイスの V_{dd} ピンをトグルする必要があります (パワー サイクル モード)。開発キット上の PSoC™ 5LP ホスト プロセッサはターゲット PSoC™ 4 デバイスの電源をトグルするためのハードウェア接続を備えていないため、本アプリケーション ノートで提供されたプロジェクトはすべて XRES プログラミング モードに対応しています。したがって、プロジェクトをホスト プロセッサに移植し、パワー サイクル モードのサポートを追加する際、プロジェクトには次の変更と考慮事項を適用する必要があります。

1. DEVICE_ACQUIRE_TIMEOUT で説明した DEVICE_ACQUIRE_TIMEOUT 値は、パワー サイクル モード プログラミングに必要な 30ms のタイムアウト ウィンドウを反映するように変更する必要があります。例えば、DEVICE_ACQUIRE_TIMEOUT 値は 2ms の XRES モード取得タイミングで 20 と計算された場合、30ms のパワーサイクル モード取得タイミングのためには 300 に変更する必要があります。
2. ホスト側の XRES ピン トグルに関連する関数定義は、デバイス パワー レールをトグルするために必要に応じて変更する必要があります。これらの関数は `SetXresHigh()` と `SetXresLow()` です。
3. ホスト プロセッサの I/O ピンが PSoC™ 4 デバイスに直接電源供給し、ホスト側の電源をトルクするために使用された場合、該当する PSoC™ 4 デバイス データシートに指定されているように、I/O ピンがデバイス動作に必要な電流を供給できることを保証してください。I/O ピンが必要な電流を供給できない場合、出力電圧 (V_{OH}) が一般的に減少し、PSoC™ 4 デバイスの最小動作電圧よりも低くなってしまい、プログラミングが失敗する可能性があります。そのような電圧降下のシナリオはオシロスコープでパワー レール電圧を監視することにより職別できます。

9 サンプル プロジェクトのテスト

プロジェクト ファイル *HexImage.h* は CY8C40xx_FAMILY, CY8C4xxxM_FAMILY, CY8C4xx7_BL_FAMILY, CY8C4xx8_BL_FAMILY, CY8C4xxxL_FAMILY, CY8C40xxS_FAMILY, CY8C41xxS_FAMILY, CY8C41xxS_PLUS_FAMILY, および CY8C41xxPS_FAMILY のパラメーターを定義します。該当ファミリの HEX ファイル パーサ アプリケーションが *HexImage.c* と *HexImage.h* ファイルを生成するために使用されている場合は、これらのパラメーターは自動的に 1 にセットされます。

9.1 PSoC™ 5LP プログラマ搭載のキット (KitProg)

Table 10 にリストされているキットで HSSP プロジェクトをテストするには、このアプリケーション ノートに添付されている B_Hssp_Pioneer プロジェクトを使用します。このキットを使用すると、外部ホスト マイクロコントローラーは必要ありません。PSoC™ 5LP はオンボード マイクロコントローラーとして存在します。

オンボード PSoC™ 5LP には、USB 経由で新しいブートローダブル アプリケーションをロードして実行できるブートローダ ファームウェアがあります。したがって、HSSP プロジェクトはブートローダブル プロジェクトとして構築され、B_Hssp_Pioneer.cyacd プロジェクトを USB 経由で PSoC™ 5LP にダウンロードして、HSSP ホスト プログラマとして使用できます。

このプロジェクトでは、USB-UART コンポーネントを使用して、シリアル通信に使用される標準プログラムであるハイパーターミナルにプログラミング出力を表示します。

このプロジェクトでは、CY8CKIT-040, CY8CKIT-042, CY8CKIT-044, CY8CKIT-042-BLE で動作するために、Table 10 にリストされている変更が必要です。対応する変更は、プロジェクト内の RegisterDefines.h ファイルで行う必要があることに注意してください。

Table 10 各種キットのピン割り当て

ホストピン	CY8CKIT-040	CY8CKIT-042	CY8CKIT-043	CY8CKIT-044	CY8CKIT-046	CY8CKIT-042-BLE
Pin_SWDCK	P12[3]	P2[1]	P12[3]	P12[3]	P12[3]	P12[3]
Pin_SWDIO	P12[2]	P2[0]	P12[2]	P12[2]	P12[2]	P12[2]
Pin_XRES	P12[4]	P2[4]	P12[4]	P12[4]	P12[4]	P12[4]

このプロジェクトをテストするために、次のステップに従ってください。

1. **B_Hssp_Pioneer プロジェクトを準備します。**
   a) HexFile Parser を使用してターゲット PSoC™ 4 デバイス用のプログラミング データを含むファイル (*HexImage.c*、*HexImage.h*) を生成します。
   b) 既存の *HexImage.c* と *HexImage.h* ファイルを、生成されたファイルに置き換えます。
   c) *TopDesign.cysch* でのブートローダブル コンポーネントをダブルクリックします。
   d) **Dependencies** タブを選択し、Figure 5 に示すようにブートローダ HEX ファイルに対応する「Browse」ボタンをクリックします。
   e) アプリケーション ノートに含まれている KitProg フォルダ (PSoC™ 5LP Bootloader Files > KitProg) に移動して、*KitProg_Bootloader.hex* を選択します。

ブートローダ HEX ファイルを選択すると、ブートローダ ELF ファイルが自動的に生成されます。

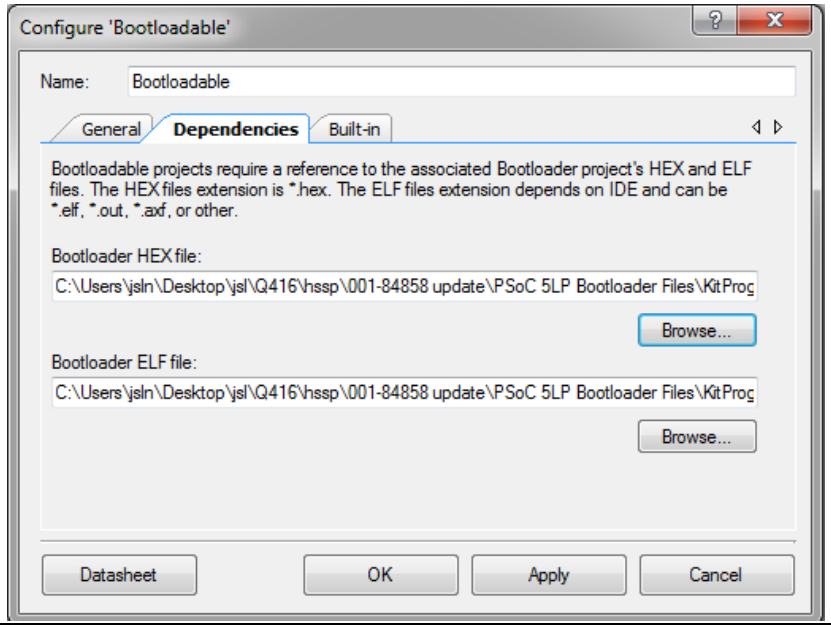


Figure 5　KitProg のブートローダブル依存関係

f) B_Hssp_Pioneer プロジェクトをビルドします。

2. **Pioneer Kit の PSoC™ 5LP ブートローダを起動します。**
   a) USB ケーブルを抜いて電源供給を切ります。
   b) キットのリセット スイッチ (SW1) を押下したまま USB ケーブルを差し込みます。

ステータス LED が点滅し始め、PSoC™ 5LP がブートローダ モードに入ったことを示します。Figure 6 に、PSoC™ 4 Pioneer Kit およびステータス LED とリセット スイッチの位置を示します。



Figure 6　PSoC™ 4 Pioneer Kit

3. **HSSP プロジェクトを PSoC™ 5LP にブートロードします。**
   a) ブートローダ ホスト ツールを開きます。PSoC™ Creator で **Tools** > **Bootloader Host** を選択します。
   b) **Filter** ボタンをクリックして、**Show USBDevices** チェックボックスにチェックを入れます。
   c) **PID** フィールドに「0xF13B」を入力します (Figure 7 を参照してください)。
   PSoC™ 5LP ブートローダはポート一覧に「USB Human Interface Device」 (USB ヒューマン インターフェースデバイス) として表示されます。

d) GUI 上に **Open** をクリックして、次のパスから *B_Hssp_Pioneer.cyacd* ファイルを選択します。
..\ AN84858 \ B_Hssp_Pioneer.cydsn \ CortexM3 \ARM_GCC_441 \Debug \B_Hssp_Pioneer.cyacd

e) **Program** をクリックしてファイルを PSoC™ 5LP にブートロードします。

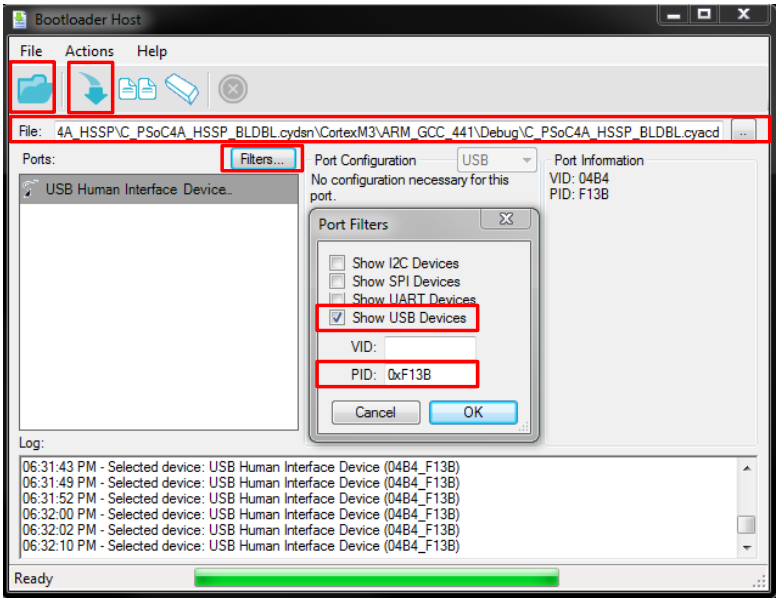


Figure 7　ブートローダ ホスト

4. **キット用の USBUART ドライバーをインストールします。**

a) USB ケーブルを抜いて、再度差し込みます。

b) Windows が USBUART ドライバーのインストールに失敗した場合、**Device Manager** を開きます。**Other Devices** リストに **USBUART** が表示されることを確認します。

c) **USBUART** をダブルクリックして、そのプロパティを開きます。

d) **Update Driver** をクリックします。

e) 「**Browse my computer for driver software**」オプションを選択します。

f) ドライバーは添付されたプロジェクトの「USBUART」フォルダにあります。本アプリケーション ノートに添付されたプロジェクトを保存した場所に移動し、*USBUART driver* フォルダ内のドライバーを選択します。

g) **Next** をクリックします。このドライバー ファイルが署名されていないので、Windows は警告を出します。その警告を無視して進みます。これで、USBUART ドライバーがユーザーのマシンにインストールされました。

5. **HyperTerminal を使って HSSP プログラミングを開始します。**

a) ユーザーのコンピューターで HyperTerminal を開きます。もし HyperTerminal がなければ、インターネットから何らかのシリアル通信用のターミナル アプリケーションをダウンロードします。

b) UART コンフィギュレーション ウィンドウで、ボー レートを 9600、データ ビットを 8、ストップ ビットを 1、パリティを「No Parity」、ハードウェア制御を「None」にセットします。

c) キーボードの任意の英数字のボタンを押してプログラミングを開始します。動作が成功すれば、ターミナルは「HSSP Success」と表示します (Figure 8 を参照してください)。
プログラミング動作が失敗した場合、ターミナルはエラーが発生したステップとエラー コードを表示します。ユーザーはそれを使ってデバッグできます (Figure 8 を参照してください)。

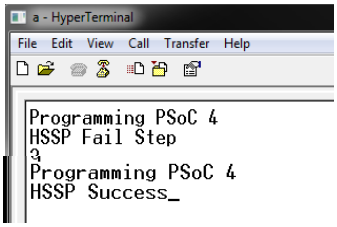


Figure 8 ターミナルの表示

6. **キットのファームウェアで PSoC™ 5LP を再プログラムします。**
   a) PSoC™ Programmer を開きます。
   b) ステップ 2 で説明したとおりにブートローダを起動します。
   c) **Utilities** タブに移動し、**Upgrade Firmware** をクリックしてオリジナルのキット ファームウェアで PSoC™ 5LP を再プログラムします (Figure 9 を参照してください)。

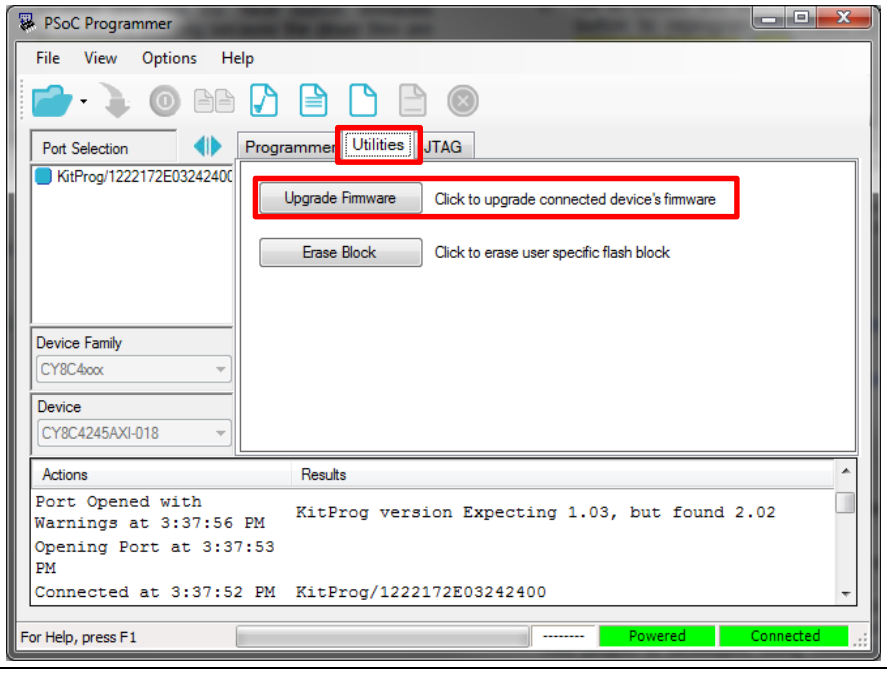


Figure 9 **PSoC™ Programmer を使用したファームウェア アップグレード**

9.2 PSoC™ 5LP プログラマ搭載のキット (KitProg2)

Table 10 に記載されているキットで HSSP プロジェクトをテストするには、本アプリケーション ノートに添付されている B_Hssp_Pioneer プロジェクトを使用します。このキットを使用する場合、外部のホスト マイクロコントローラーが不要です。PSoC™ 5LP は基板搭載のマイクロコントローラーとして機能します。

基板搭載の PSoC™ 5LP デバイスは、新しいブートローダブル アプリケーションを USB 経由でロードして実行するブートローダ ファームウェアを備えています。したがって、HSSP プロジェクトはブートローダブル プロジェクトとしてビルドされます。PSoC™ 5LP を HSSP ホスト プログラマとして使用するために、*B_Hssp_Pioneer.cyacd* プロジェクトを USB 経由で PSoC™ 5LP にダウンロードできます。

9 サンプル プロジェクトのテスト

このプロジェクトは USB-UART コンポーネントを使用して、シリアル通信用の標準プログラムである HyperTerminal にプログラミングの出力を表示します。

CY8CKIT-041-40xx キットで動作するには、このプロジェクトは Table 10 のとおりに変更する必要があります。プロジェクトの *RegisterDefines.h* ファイルでの該当する変更を行う必要があることに注意してください。

Table 11　異なるキット用のピン割り当て

ホスト ピン	CY8CKIT-041-40xx	CY8CKIT-149	CY8CKIT-147
Pin_SWDCK	P12[3]	P12[3]	P12[3]
Pin_SWDIO	P12[2]	P12[2]	P12[2]
Pin_XRES	P12[4]	P12[4]	P12[4]

このプロジェクトをテストするために、次のステップに従ってください。

1. **B_Hssp_Pioneer プロジェクトを準備します。**
   a) HexFile Parser を使用してターゲット デバイス用のプログラミング データを含むファイル (*HexImage.c*、*HexImage.h*) を生成します。
   b) 既存の *HexImage.c* と *HexImage.h* ファイルを、生成されたファイルに置き換えます。
   c) TopDesign.cysch でのブートローダブル コンポーネントをダブルクリックします。
   d) **Dependencies** タブを選択し、Figure 10 に示すようにブートローダ HEX ファイルに対応する **Browse** ボタンをクリックします。
   e) アプリケーション ノートに含まれている KitProg2 フォルダ (**PSoC™ 5LP Bootloader Files** > **KitPog2**) に移動して、Bootloader HEX file フィールドで *KitProg2_Bootloader.hex* を選択します。

ブートローダ HEX ファイルを選択すると、ブートローダ ELF ファイルが自動的に生成されます。

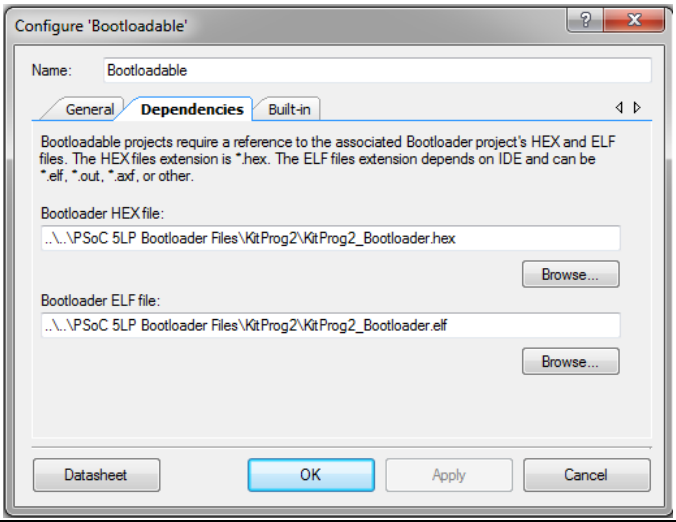


Figure 10　KitProg2 のブートローダブル依存関係

   f) B_Hssp_Pioneer プロジェクトをビルドします。
2. **Pioneer Kit の PSoC™ 5LP ブートローダを起動します。**
   a) USB ケーブルを抜いて電源供給を切ります。
   b) キットの SW1 を押下したまま USB ケーブルを差し込みます。

LED2 が点滅し始め、PSoC™ 5LP がブートローダ モードに入ったことを示します。Figure 11 は PSoC™ CY8CKIT-041-040xx キットおよび LED2 と SW3 の位置を示します。

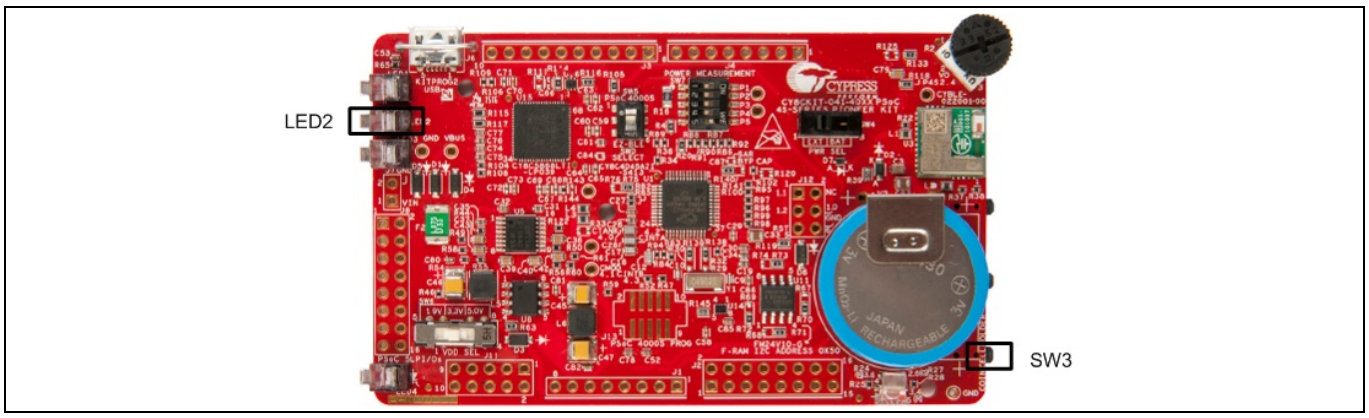


Figure 11　CY8CKIT-041-040xx

3. **HSSP プロジェクトを PSoC™ 5LP にブートロードします。**
   a) ブートローダ ホスト ツールを開きます。PSoC™ Creator で **Tools** > **Bootloader Host** を選択します。
   b) Filter ボタンをクリックして、**Show USB Devices** チェックボックスにチェックを入れます。
   c) PID フィールドに「0XF146」を入力します。
   d) PSoC™ 5LP ブートローダはポート一覧に USB Human Interface Device (USB ヒューマン インターフェースデバイス) として表示されます。
   e) GUI 上に **Open** をクリックして、次のパスから B_Hssp_Pioneer.cyacd ファイルを選択します。
   ..\ AN84858 \ B_Hssp_Pioneer.cydsn \ CortexM3 \ARM_GCC_441 \Debug \B_Hssp_Pioneer_2.cyacd
   f) **Program** をクリックしてファイルを PSoC™ 5LP にブートロードします。

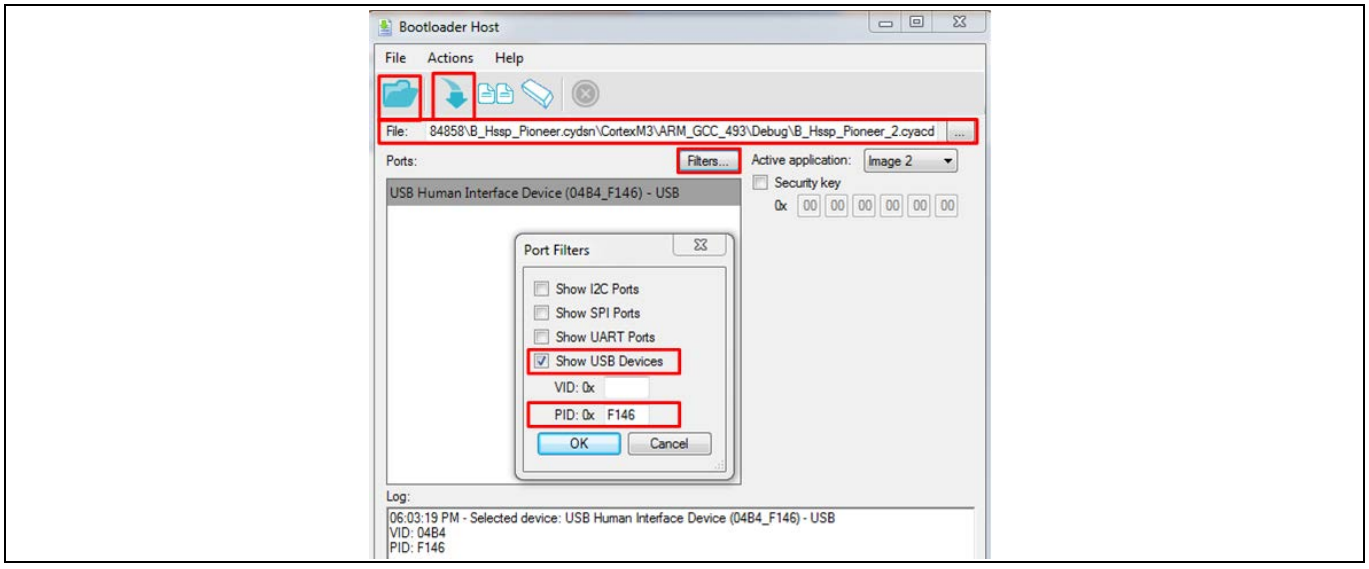


Figure 12　ブートローダ ホスト アプリケーション

4. **キット用に USBUART ドライバーをインストールします。**
   a) USB ケーブルを抜いて、再度差し込みます。
   b) Windows が USBUART ドライバーのインストールに失敗した場合、以下を行ってください。

a. **Device Manager** を開きます。**Other Devices** リストに **USBUART** が表示されることを確認する。

b. USBUART をダブルクリックして、そのプロパティを開きます。

c. **Update Driver** をクリックします。

d. **Browse my computer for driver software** オプションを選択します。

e. 本アプリケーション ノートに添付されたプロジェクトを保存した場所に移動し、*USBUART driver* フォルダ内のドライバーを選択します。

f. **Next** をクリックします。このドライバー ファイルが署名されていないので、Windows は警告を出します。その警告を無視して進みます。これで、USBUART ドライバーがユーザーのマシンにインストールされました。

5. **HyperTerminal を使って HSSP プログラミングを開始します。**

a) ユーザーのコンピューターで HyperTerminal を開きます。もし HyperTerminal がなければ、インターネットから何らかのシリアル通信用のターミナル アプリケーションをダウンロードします。

b) UART コンフィギュレーション ウィンドウで、ボー レートを 9600, データ ビットを 8, ストップ ビットを 1, パリティを No Parity, ハードウェア制御を None にセットします。

c) キーボードの任意の英数字のボタンを押してプログラミングを開始します。動作が成功すれば、ターミナルは「HSSP Success」と表示します。

d) プログラミング動作が失敗した場合、ターミナルはエラーが発生したステップとエラー コードを表示します。ユーザーはそれを使ってデバッグできます。

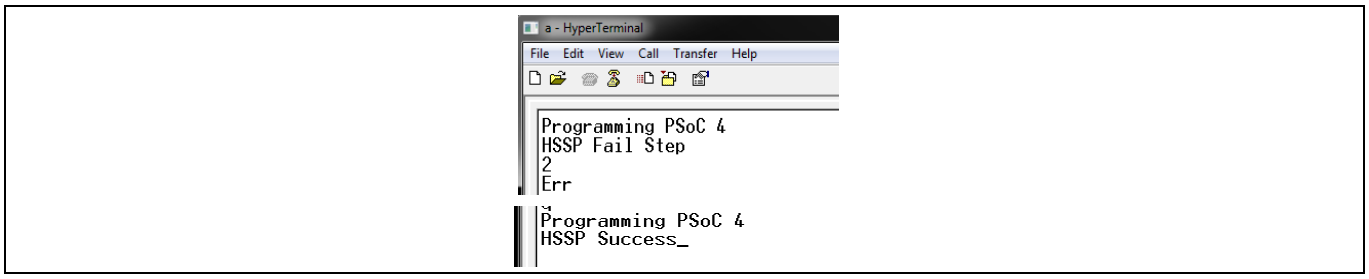


Figure 13　ターミナルの表示

6. **キットのファームウェアで PSoC™ 5LP を再プログラムします。**

a) PSoC™ Programmer を開きます。

b) ステップ 2 で説明したとおりにブートローダを起動します。

c) **Utilities** タブに移動し、**Upgrade Firmware** をクリックしてオリジナルのキット ファームウェアで PSoC™ 5LP を再プログラムします。

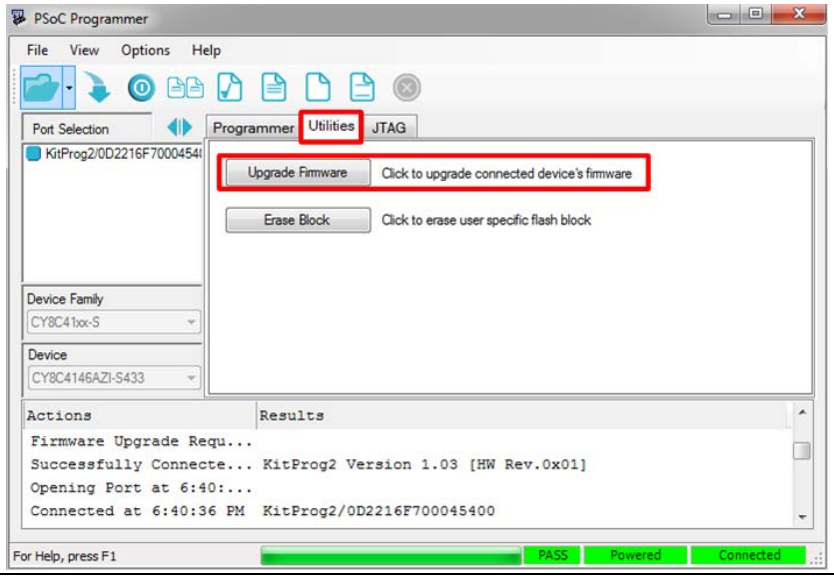


Figure 14 PSoC™ Programmer を使用したファームウェア アップグレード

10 HSSP 問題をデバッグするためのヒントとコツ

サンプル コードで使用された PSoC™ 5LP ホスト プロセッサから HSSP コードをユーザーのプロセッサ アーキテクチャに移植することは、ホスト プロセッサ側の他のシステム レベル制約によって複雑な作業になることがあります。本節では、ユーザーのハードウェア プラットフォーム向けに HSSP アプリケーションを開発する際に最も一般に発生する問題に対するトラブルシューティング方法について説明します。

- **ハードウェア セットアップの検証**: 最初のステップは、ハードウェア接続が HSSP 動作用に適切に実施されていることを確かめることです。このために、ホスト プロセッサとターゲット PSoC™ 4 デバイスの間でピンを正しく接続し、すべての PSoC™ 4 電圧ドメインに電源供給し、ホスト SWD ピンの駆動モードが適切に設定されていることを確実にしてください。
- **タイミングの検証**: ホスト PSoC™ 5LP コードをユーザーのホスト プロセッサに移植する際、コードで使用するタイムアウト パラメーターがホスト プロセッサ コード タイミングを反映するよう変更されたことを確認してください。コード移植時のタイムアウト パラメーターの変更に関する情報は、HSSP タイムアウト パラメーターの計算を参照してください。

HSSP の最初のステップである「デバイスの取得」では、プログラミング モードへの移行に関して厳しいタイミング要件があります。重要な要件の 1 つは、取得ウィンドウ タイミングを満たすために SWDCK クロック ラインの周波数が少なくとも 1.5MHz であることです。ホスト プロセッサ側での「デバイスの取得」ステップを完了するために要するコード実行時間に影響を与えないように、ホスト プロセッサ側に割り込みイベントが無いことを保証してください。ホスト プロセッサがデバイスのプログラミング仕様書の「Step 1. Acquire Chip」で説明されているすべてのタイミング要件を満たすことを確実にしてください。

- **HSSP アルゴリズムの検証**:
  - HSSP コードを移植する際、Figure 1 に示す SWD パケット層ファイルが変更された場合、SWD パケット フォーマットがデバイスのプログラミング仕様書の「Serial Wire Debug (SWD) Format」節のとおりであることを確実にしてください。
  - MiniProg3 や KitProg などのサイプレス認定プログラマは Figure 1 の「フラッシュの消去」と「フラッシュのプログラム」などのステップを検証およびデバッグするために使用されます。例えば、ホスト プロセッサがフラッシュ メモリ全体を消去したかを確認するために、MiniProg3 プログラマと PSoC™ Programmer GUI の「Read」オプションを使用してフラッシュの全データがゼロになったことを検証できます。PSoC™ Programmer GUI の「Checksum」オプションを使用すると、デバイスにプログラムされたフラッシュ データのチェックサムが GUI の入力ファイルとしての HEX ファイルのチェックサムと一致することを保証できます。さらに、PSoC™ Programmer GUI の「Patch Image」オプションを使用すると、デバイス フラッシュ内容と HEX ファイルのデータ不一致が発生したフラッシュ行の数を特定できます。

11 まとめ

HSSP アプリケーションは、PSoC™ 4 デバイス用のインシステム プログラミングのソリューション開発に役立ちます。基板に搭載された組み込みマイクロコントローラーをホスト プログラマとして使用することは、PSoC™ 4 をプログラムするための安価で堅牢な方法を提供します。本アプリケーション ノートで提供されている移植可能なモジュール化 C コードにより、そのような HSSP アプリケーションの開発に要する時間を大幅に短縮させます。

12 添付プロジェクトの一覧

AN84858.cywrk: このワークスペースは、HSSP アプリケーションをデモする 3 つのプロジェクトを含みます。

- A_Hssp_Programmer: これは PSoC™ 4 の HSSP アプリケーション プロジェクトです。ホスト プログラマの役割を果たしてターゲット PSoC™ 4 デバイスをプログラムする PSoC™ 5LP デバイスがあります。このプロジェクトは、関連資料に掲載されている各デバイスの該当プログラミング仕様書に説明されている手順に従い、モジュール化された C コードを使用してデバイスをプログラムします。
- B_Hssp_Pioneer: これは、PSoC™ 4 Pioneer Kit でテストするための PSoC™ 4 HSSP アプリケーション プロジェクトです。基板搭載の PSoC™ 5LP プログラマを使って PSoC™ 4 デバイスをプログラムします。
- C_Hssp_TimeoutCalc: これは、PSoC™ 4 の HSSP プロジェクトで使用されるタイムスタンプ パラメーターを計算するためのプロジェクトです。

また、HEX ファイルから情報を抽出する C#アプリケーションがアプリケーション ノートに添付されています。

- Hex File Parser アプリケーション: この C#アプリケーションは必要な情報を HEX ファイルから抽出し、*.c/h* ファイルに解析します。このファイルはマイクロコントローラーのフラッシュに格納され、プログラミング データに直接アクセスするために使用されます。

13 付録 A. Hex File Parser アプリケーション

PSoC™ 4 をプログラムするデータは、関連資料に掲載されている各デバイスの該当プログラミング仕様書の「Appendix B. Intel Hex File Format」で説明されている HEX ファイル (.hex) フォーマットにあります。

HEX ファイルは、ターゲット デバイスをプログラムするためにホストが利用できるフォーマットではありません。PSoC™ Creator ソフトウェアによって生成されたこのファイルには、16 進形式でのプログラミング データとメタデータの両方があります。メタデータは HEX ファイルのレコード型の情報、拡張リニア アドレスのデータなどを含みます。メタデータはプログラミングデータをフラッシュ コード領域データ, フラッシュ コンフィギュレーション領域データ, フラッシュ保護データなどに分類するために使用されます。

この C#アプリケーションは、Visual Studio 開発環境で開発され、HEX ファイルを解析して、HEX ファイルからのプログラミング データのみを格納する *.c* と *.h* (*HexImage.c* と *HexImage.h*) ファイルを生成します。プログラミング データは定数の配列として *HexImage.c* と *HexImage.h* ファイルに格納されます。

ソース コードを含む C#アプリケーションは本アプリケーション ノートとともに提供されています。C#アプリケーションを使用するには、.NET フレームワーク 3.5 以降をコンピューターにインストールする必要があります。

Note: PSoC™ 4000, PSoC™ 4100/4200, PSoC™ 4100M/4200M, PSoC™ 4xx7_BLE, PSoC™ 4xx8_BLE, PSoC™ 4xxxL, PSoC™ 40xxS, PSoC™ 41xxS, PSoC™ 41xxS Plus, および PSoC™ 41xxPS デバイス ファミリごとにそれぞれの Hex File Parser アプリケーションが提供されています。

C#アプリケーション名: Hex File Parser

開発環境: Microsoft Visual Studio Express 2013 (バージョン 12.0.31101.00 (4 回目の更新))

ソース コード: 詳細は C#ソース プロジェクトを参照してください。プロジェクトのソース コードは、無料の Microsoft Visual Studio 2013 Express Edition をダウンロードしてインストールすることにより、表示し変更できます。**Solution Explorer** ウィンドウで *Form1.cs* ファイルを単に右クリックして、**View Code** オプションを選択します。これで、*Form1.cs* のソースコードが開きます。

13.1 Hex File Parser アプリケーションの使用方法

「C# Application」フォルダには、9 組のパーサ アプリケーションがあります。ご使用の PSoC™ 4 ファミリに応じて、適切なアプリケーションを選択してください。

添付する *.zip* ファイル内の *C# Application\HexFile Parser.exe* フォルダにある Hex File Parser アプリケーションの実行ファイルを開きます。Figure 15 に示すように GUI 画面がポップアップ表示されます。

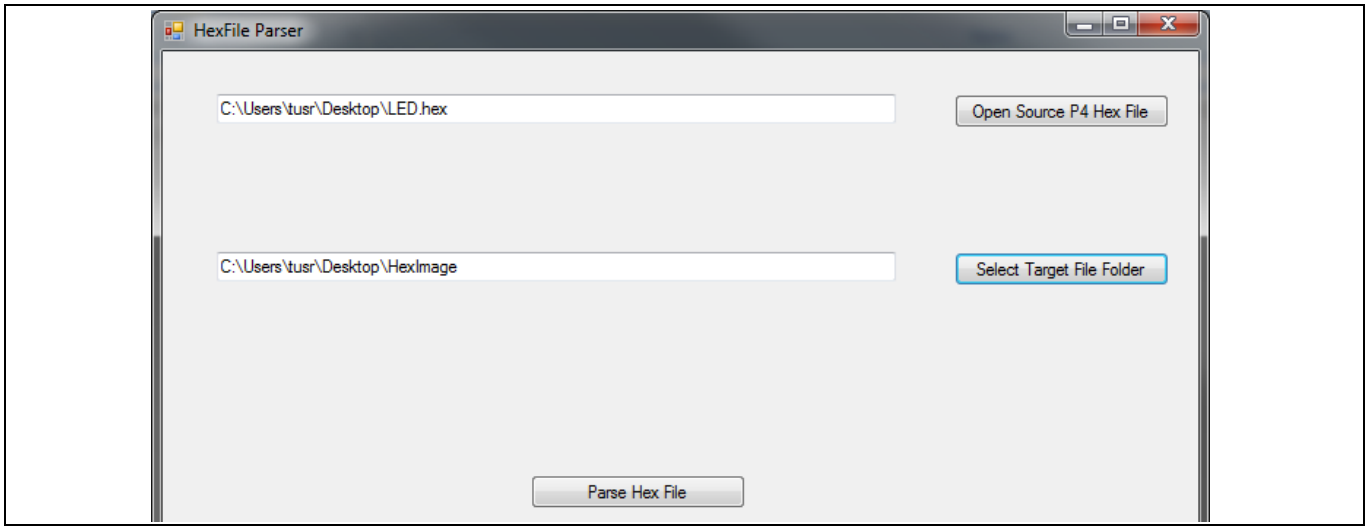


Figure 15　Hex File Parser アプリケーション

1. **Open Source P4 Hex File** をクリックしてファイルの場所を指定することで、プログラム対象の HEX ファイルを選択します。
2. 解析された.*c* と.*h* ファイル (*HexImage.h* と *HexImage.c*) が作成されるデスティネーション フォルダの場所を選択します。**Select Target File Folder** をクリックしてフォルダの場所を選択します。

Note: HexImage.h、HexImage.c の名前を持つファイルがターゲット フォルダの場所にないことを確認してください。それらのファイルがあれば、削除するかまたは新しいフォルダ場所を選択します。ターゲット フォルダの場所に同じファイル名のファイルが既に存在している場合、通知するメッセージが表示されます。

3. HEX ファイルとターゲット フォルダの場所を選択した後、**Parse Hex File** をクリックして.*c* と.*h* ファイルを生成します。解析が完了後、メッセージが表示されます。

13.2　PSoC™ Creator のサンプル プロジェクトへの生成したファイルの追加

生成した *HexImage.c*、*HexImage.h* ファイルを、本アプリケーション ノートで提供された PSoC™ Creator サンプルプロジェクトに追加するために、本節での手順に従ってください。PSoC™ 5LP をホスト プログラマとして使用してターゲットデバイスに HEX ファイルをプログラムする必要がある場合は、これらのステップが必要です。これらのステップは PSoC™ 4 HSSP 用の A_Hssp_Programmer と B_Hssp_Pioneer プロジェクトに適用します。

1. Figure 15 に示すように、GUI でターゲット ファイル フォルダの場所を選択します。これは、プロジェクトの *main.c* ファイルが配置されているフォルダです。
2. *HexImage.c* と *HexImage.h* ファイルが上記の場所に生成された後、Figure 16 に示すように、ワークスペース エクスプローラーで **Add Existing Item** をクリックして、それらのファイルを PSoC™ Creator 内のプロジェクト ワとークスペースに追加します。

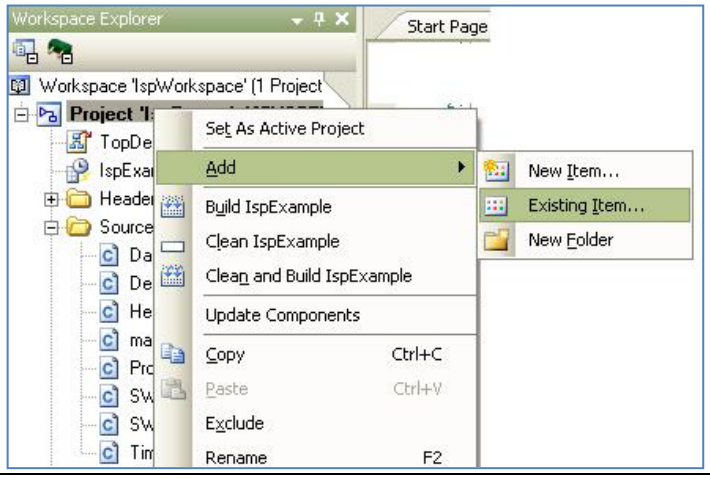


Figure 16　PSoC™ Creator プロジェクトへのファイルの追加

3. ファイルが選択された後、Figure 17 に示すように、Workspace Explorer ウィンドウが現れます。

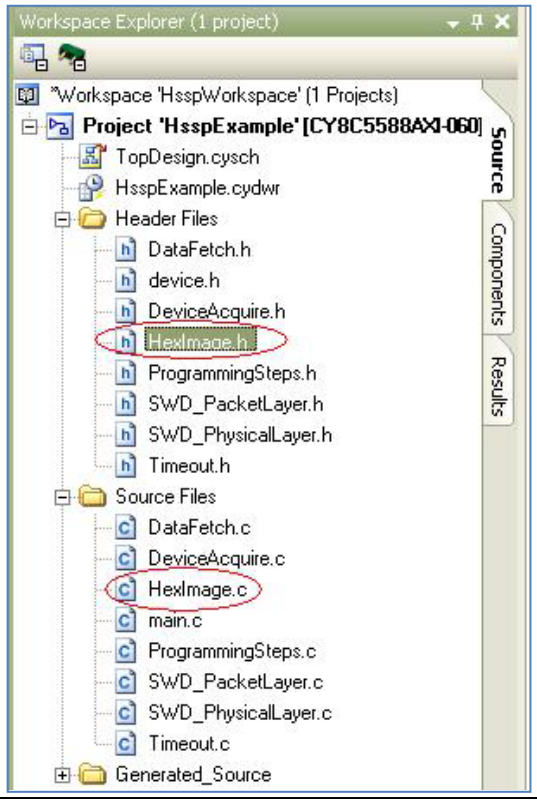


Figure 17　HexImage.c と HexImage.h ファイルを表示しているプロジェクト ワークスペース

別の HEX ファイルに対応するもので、プロジェクト内でこれらのファイルを置き換える必要がある場合、既存のファイルを右クリックして削除することで、それらのファイルを Workspace Explorer とプロジェクト フォルダから削除します。その後、上記の説明のとおりに、ステップ 1 から Figure 16 に従って新しいファイルをプロジェクトに追加します。

13.2.1 PSoC™ 4xxxL/PSoC™ 41x8_BLE/42x8_BLE デバイス ファミリ用の HexImage.c と HexImage.h ファイルの使用

1. HSSP サンプル コードは、PSoC™ 5LP デバイスをホスト プログラマとして使用し、ターゲット PSoC™ 4 デバイスをプログラムします。ホスト PSoC™ 5LP の HSSP アルゴリズムが 256KB のフラッシュ メモリの一部を占めるため、より高容量のフラッシュ メモリ デバイス (256KB PSoC™ 4xxxL/PSoC™ 41x8_BLE/42x8_BLE ファミリのデバイスなど) の HEX ファイル データはホスト PSoC™ 5LP のフラッシュ メモリにすべて格納できません。
2. B_HSSP_Pioneer プロジェクトでは、PSoC™ 41xxS Plus デバイスの 128KB 全体は、デュアル アプリケーション イメージブートロードを備えた KitProg2 を使用する CY8CKIT-159 の基板搭載の PSoC™ 5 に格納できません。
3. コード サイズの要件を満たすために、最後の数フラッシュ行に対応するプログラミング データを *HexImage.c* と *HexImage.h* ファイルから削除します。対応する論理も *DataFetch.c* ファイルの HEX_ReadRowData(...) 関数に組み込まれています。変更は以下のとおりです。正確な変更については、プロジェクト コードを参照してください。

1. *HexImage.h* ファイルでは、以下のように配列宣言を編集することで、配列宣言内のフラッシュ行数を 256 減らします。

```
extern unsigned char const
flashData_HexFile[(NUMBER_OF_FLASH_ROWS_HEX_FILE -
256)][FLASH_ROW_BYTE_SIZE_HEX_FILE];
```

フラッシュ行数を 256 に選択したら 、すべてのターゲット デバイスに対してプロジェクトのコード サイズはホスト PSoC™ 5LP フラッシュ メモリ容量に合います。

2. *HexImage.c ファイル*では、*HexImage.h* ファイル内の配列宣言で行った変更に応じて、最後の 256 フラッシュ行が配列の定義 `flashData_HexFile[][]` から削除されます。
3. DataFetch.c ファイルの HEX_ReadRowData (...) 関数の定義では、フラッシュ行番号がターゲット デバイスの最後の 256 行にある場合、フラッシュ行データ全体はゼロでロードされます。

14 付録 B. SROM 要求用のステータス コード

プログラミングに関連したすべての動作はシステム コール関数として実行されます。それらは、特権動作モードで SROM から実行されます。

ユーザーは、SROM コードへの読み出し/変更のアクセスがありません。DAP または Cortex®-M0 CPU は、特定のレジスタに関数オペコードと関数パラメーターを書き込んで、SROM が関数を実行するよう要求することで、システム コールの要求を行います。関数オペコードに基づいて、SROM はそのメモリから対応するシステム コールを実行して、レジスタで関数実行ステータスを更新します。DAP または CPU は関数実行の成功/失敗の結果を取得するためにこのステータス レジスタを読み出す必要があります。

CPUSS_SYSREQ レジスタの SROM_SYSREQ_BIT (ビット 31) と SROM_PRIVILEGED_BIT (ビット 28) がクリアされることは、システム コールが完了したことを示します。システム コールの成功/失敗の状態を確認するには、CPUSS_SYSARG レジスタを読み出します。

レジスタから読み出した 32 ビット値が 0xAXXXXXXX (X はドント ケア値を示す) の場合、システム コールは成功です。レジスタから読み出した値が、0xF00000YY 形式の場合は失敗を意味し、その内 YY は失敗の原因を示します。Table 12 はエラー コードの一覧を示します。

詳細については、関連資料に記載されている該当 PSoC™ 4 アーキテクチャ テクニカル リファレンス マニュアル (TRM) の「Nonvolatile Memory Programming」章を参照してください。

Table 12 エラー ステータス コードおよび失敗の原因

ステータス コード (CPUSS_SYSARG レジスタ内の 32 ビット値)	説明
0xAXXXXXXX	成功: API が CPUSS_SYSARG レジスタにパラメーターを直接返さない限り、「X」の値は、SROM から返された「0」を含む「ドント ケア」値を表します。
0x F0000001	無効チップ保護モード: この API は現行のチップ保護モード中は使用できません。
0x F0000003	無効ページ ラッチ アドレス: ページ ラッチ バッファ内のアドレスは範囲外にあるか、または提供したサイズがページ アドレスより大きすぎます。
0x F0000004	無効アドレス: 提供した行 ID またはバイト アドレスは利用可能なメモリ範囲外です。
0x F0000005	行の保護: 与えられた行 ID は保護されている行の ID です。
0x F0000006	SRAM アドレス無効: SRAM アドレスは範囲外です。
0x F0000007	再開完了: すべてのノンブロッキング API の実行は完了しました。再開 API は次のノンブロッキング API が実行されるまで呼び出せません。
0x F0000008	保留再開: ノンブロッキング API が開始しました。他の API が呼び出される前に再開 API を呼び出してこの API を完了させる必要があります。
0x F0000009	システム コール実行中: 再開またはノンブロッキング API がまだ実行中です。SPC ISR は、次の再開を試行する前に発生する必要があります。
0x F000000A	チェックサム ゼロ失敗: 計算したチェックサムはゼロではありません。
0x F000000B	無効オペコード: オペコードは有効な API オペコードではありません。
0x F000000C	キー オペコード不一致: 与えられたオペコードは key1 と key2 に一致しません。
0x F000000E	無効開始アドレス: 開始アドレスは与えられた終了アドレスよりも大きいです。

14 付録 B. SROM 要求用のステータス コード

ステータス コード (CPUSS_SYSARG レジスタ内の 32 ビット値)	説明
0xF0000012	無効フラッシュ クロック: CY8C40xx デバイス ファミリは、書き込み/消去動作前に、IMO を 48MHz に、HF クロック ソースを IMO クロックに設定する必要があります。

15 付録 C. HSSP エラー ステータス レジスタのビット フィールド定義

HSSP エラー ステータス レジスタは現行の HSSP 動作の状態を含みます。HSSP アプリケーションのトップレベルステップのいずれかが Failure 状態を返したとき、`ReadHsspErrorStatus()`関数がメイン アプリケーション コードから呼び出され、エラーの詳細を取得します。この関数はこのレジスタの内容を返します。この関数によって返されたビットフィールド値については Figure 3 を参照してください。

正常な HSSP 動作のためには、ビット 0 を除いて 8 ビット レジスタのすべてのビットはゼロでなければいけません。Table 13 に示すように、ビット 0 がセットされることは SWD パケットが OK ACK を受信したことを意味します。

15.1 ビット[2:0]: SWD 認識応答 (SWD ACK[2:0])

これは、ターゲット デバイスからホスト プログラマに送信した SWD パケットの 3 ビットの確認応答です。あり得る ACK コードを Table 13 に示します。

Table 13 SWD ACK 応答コード

ACK[2:0]	ACK 応答の意味	説明
001	OK (SUCCESS)	前回の SWD トランザクションが成功したことを意味します。
010	WAIT	このエラー コードは、ターゲットが 5 つの WAIT ACK 応答を連続的に返したことを示します。
100	FAULT	このエラー コードは、前回の SWD 書き込みパケットの実施中に、ホストから送信した 4 バイトのデータ パケット内にパリティ エラーが発生したことを示します。
他のコード	未定義コード	FAULT 応答として扱います。

OK ACK 応答以外のすべての応答に対しては、ホストが HSSP 動作を中止し、最初のステップからやり直す必要があります。OK ACK 応答の場合であっても、Figure 3 内のステータス レジスタの残りのビット フィールド (ビット 3～ビット 6) をセットしてはいけません。OK ACK 応答の場合で他のビット フィールドのいずれかがセットされていれば、HSSP 動作を中止して最初からやり直す必要があります。

ホストプログラマがプログラミング仕様で指定されている SWDCK の最大値よりも高い周波数で SWDCK クロックを供給しようとすると、WAIT ACK が受信されます。

15.2 ビット 3: SWD 読み出しデータ パリティ エラー

ターゲット デバイスから受信したデータにパリティ エラーが発生した場合、ホスト プログラマはこのビットをセットします。ホストは HSSP 動作を中止して再試行する必要があります。

15.3 ビット 4: ポート取得タイムアウト

ターゲット デバイス取得の一部である SWD パケット (ステップ 1. *ProgrammingSteps.c* ファイルにある `DeviceAcquire()`関数) が正常に完了しなかった場合、このビットがセットされます。このビットがセットされたら、HSSP 動作を中止し、再試行する必要があります。

このタイムアウト エラーの原因には次の 2 つがあります: ホスト プログラマとターゲット デバイス間のハードウェア接続に問題があったか、あるいはホスト プログラマが、ターゲット デバイスのプログラミング モードに移行するタイミング要件を満たさなかった場合です。

PSoC™ 4 プログラミング モードに移行するタイミング要件については、関連資料に掲載されている各デバイスの該当プログラミング仕様書の「Step 1: Acquire Chip」を参照してください。

15.4 ビット 5: SROM ポーリング タイムアウト エラー

PSoC™ 4 内のフラッシュ プログラミングは SROM API により実行されます。`PollSromStatus()`関数がエラー状態を返すと、このビットがセットされます。ステータス コードを知るために CPUSS_SYSREQ レジスタは 1 秒間ポーリングされます。1 秒を超えたか、またはこの関数によって返されたステータス コードが FAILURE に該当した場合、このビットがセットされます。この関数は、SROM システム要求のステータスを読み出すためにすべての HSSP ステップで呼び出されます。

このビットがセットされた場合、ホスト プログラマはポーリング中に返されたステータス コードの値を読み出し、表示するために *ProgrammingSteps.h* ファイル内の `ReadSromStatus()`関数を呼び出さなければいけません。

SROM ステータス コードの詳細については、付録 B. SROM 要求用のステータス コードを参照してください。

15.5 ビット 6: 検証失敗

Table 14 で説明するように、このエラー ビットは複数のステップでセットされ、複数の原因で検証に失敗することがあります。

Table 14 検証エラー: ステップと原因

エラー ステップ	エラー	エラーの原因
デバイスの取得 (ステップ 1)	デバイス ID 検証エラー	デバイスから返された IDCODE が Cortex®-M0 DAP ID (0x0BB11477) と一致しません。
シリコン ID の検証 (ステップ 2)	シリコン ID 検証エラー	HEX ファイル内のシリコン ID 情報がターゲット デバイスから読み出されたシリコン ID と一致しません。
フラッシュの検証 (ステップ 6)	フラッシュ データ検証エラー	フラッシュ データが HEX ファイル内のデータと一致しません。
保護設定の検証 (ステップ 7)	フラッシュ保護データ検証エラー	シリコン チップから読み出された行保護データまたはチップ保護データが HEX ファイル内のデータと一致しません。
チェックサムの検証 (ステップ 9)	チェックサム検証エラー	ターゲット デバイスのフラッシュ データのチェックサム値が HEX ファイル内のチェックサム データと一致しません。

上記の条件から見ると、ビット 6 は多くの検証エラー ケースでセットされることが分かります。そのビットがどのステップでセットされるかによって、検証失敗の原因を推測できます。例えば、このビットが「シリコン ID の検証」ステップでセットされると、ホスト プログラマ アプリケーションはシリコン ID の不一致がエラーの原因であると判断できます。

15.6 ビット 7: 遷移エラー

チップから読み出されたチップ保護設定と HEX ファイルに格納されたチップ保護設定が間違った遷移を示すと、このビットがセットされます。

保護モードおよび有効と無効な遷移の状態遷移図については、関連資料に掲載されている各デバイスの該当プログラミング仕様書の「Appendix A. Chip-Level Protection」を参照してください。

16 付録 D. HSSP 関数

下表に HSSP ファームウェア アーキテクチャの各層に定義されたパブリック関数を示します。これらの関数は、Figure 2 に示すように HSSP ファームウェアの各層間の通信に使用されます。

Table 15 SWD_PhysicalLayer.h ファイル内の関数

関数名	説明
`SetSwdckHigh()`	ホストの SWDCK ピンを HIGH にセットします。
`SetSwdioHigh()`	ホストの SWDIO ピンを HIGH にセットします。
`SetXresHigh()`	ホストの XRES ピンを HIGH にセットします。
`SetSwdckLow()`	ホストの SWDCK ピンを LOW にセットします。
`SetSwdioLow()`	ホストの SWDIO ピンを LOW にセットします。
`SetXresLow()`	ホストの XRES ピンを LOW にセットします。
`SetSwdckCmosOutput()`	ホストの SWDCK ピンを CMOS 出力駆動モードに設定します。
`SetSwdioCmosOutput()`	ホストの SWDIO ピンを CMOS 出力駆動モードに設定します。
`SetXresCmosOutput()`	ホストの XRES ピンを CMOS 出力駆動モードに設定します。
`SetSwdckHizInput()`	ホストの SWDCK ピンを High-Z デジタル入力駆動モードに設定します。
`SetSwdioHizInput()`	ホストの SWDIO ピンを High-Z デジタル入力駆動モードに設定します。
`SetXresHizInput()`	ホストの XRES ピンを High-Z デジタル入力駆動モードに設定します。
`ReadSwdio()`	SWDIO 入力ピンの現在の状態を返します。

Table 16 SWD_PacketLayer.h ファイル内の関数

関数名	説明
`Swd_WritePacket()`	SWD 書き込みパケットを送信します。この関数は swd_PacketHeader、swd_PacketAck および swd_PacketData[] のグローバル変数を使用します。
`Swd_ReadPacket()`	1 つの SWD 読み出しパケットを送信します。この関数は swd_PacketHeader, swd_PacketAck, および swd_PacketData[] のグローバル変数を使用します。この関数は特定のアドレスから読み出すために使用されます。
`SwdLineReset()`	SWDIO ラインが HIGH 状態で 51 回の SWDCK クロック サイクルを送信することで SWD ラインをリセットします。プログラミングのステップ 1 でデバッグ アクセス ポート (DAP) を取得するために使用されます。

Table 17 SWD_UpperPacketLayer.h ファイル内の関数

関数名	説明
`Read_DAP`	特定の DAP レジスタから 32 ビット データを読み出して `Swd_PacketData()`に書き込みます。この関数は `Swd_ReadPacket()` 関数を使用してデータを読み出します。
`Write_DAP`	特定の DAP レジスタに 32 ビット データを書き込みます。この関数は `Swd_WritePacket()`関数を使用してデータを書き込みます。
`Read_IO`	CPU アドレス空間の特定のアドレスから 32 ビット データを読み出します。この関数は `Read_DAP()`と `Write_DAP()`関数を使用して実行されます。すべての SWD トランザクションが正常に実行された (ACK された) 場合、「True」を返します。

関数名	説明
Write_IO	CPU アドレス空間の特定のアドレスに 32 ビット データを書き込みます。この関数は Write_DAP()関数を使用して実行されます。すべての SWD トランザクションが正常に実行された場合、「True」を返します。

Table 18 Timeout.h ファイル内の関数

関数名	説明
DelayHundredUs()	XRES ピン上に 100µs のアクティブ LOW パルス信号を生成するために、100µs の遅延を入れます。

Table 19 DataFetch.h ファイル内の関数

関数名	説明
HEX_ReadSiliconId()	*HexImage.c* ファイルからデバイス シリコン ID データを指定先配列にコピーします。
HEX_ReadRowData()	*HexImage.c* ファイルからフラッシュ行データを指定先配列にコピーします。フラッシュ行の番号もこの関数のパラメーターとして渡されます。
HEX_ReadRowProtectionData()	*HexImage.c* ファイルからフラッシュ行保護データを指定先配列にコピーします。保護データのバイト サイズもこの関数のパラメーターとして渡されます。
HEX_ReadChipProtectionData	*HexImage.c* ファイルからチップ保護データを指定先にコピーします。
HEX_ReadChecksumData()	*HexImage.c* ファイルからチェックサム データを指定先配列にコピーします。
GetFlashRowCount()	*HexImage.c* ファイルからターゲット デバイスのフラッシュ行の総数を返します。

Table 20 ProgrammingSteps.h ファイル内の関数

関数名	説明
DeviceAcquire()	プログラミング モードに移行してターゲット デバイスを取得します。
VerifySiliconId()	ターゲット デバイスのシリコン ID が HEX ファイルと一致するかを検証します。
EraseAllFlash()	フラッシュ保護データを含む、ターゲット デバイスのフラッシュ メモリをすべて消去します。
ChecksumPrivileged()	フラッシュ内の特権データのチェックサム値を計算します。
ProgramFlash()	ターゲット デバイスのフラッシュ メモリをプログラムします。
VerifyFlash()	ターゲット デバイスにプログラムされたフラッシュ データが HEX ファイルのフラッシュ データと一致するかを検証します。
ProgramProtectionSettings()	ターゲット デバイスにフラッシュ行保護データとチップ保護データをプログラムします。
VerifyProtectionSettings()	ターゲット デバイスにプログラムされたフラッシュ行保護データとチップ保護データが HEX ファイル内のデータと一致するかを検証します。

関数名	説明
VerifyChecksum()	ターゲット デバイスから読み出されたチェックサム データが HEX ファイル内のチェックサム データと一致するかを検証します。
ExitProgrammingMode()	XRES ピン上にアクティブ LOW パルス信号を生成することで、ターゲット デバイスのプログラミング モードを終了します。
ReadHsspErrorStatus()	HSSP 動作のエラー ステータスを返します。
ReadSromStatus()	SROM ポーリング タイムアウト エラー状態の場合、CPUSS_SYSARG ステータス レジスタの値を返します。

関連資料

[1] アプリケーションノート

- AN73054 – PSoC™ 3 / PSoC™ 5LP programming using an external microcontroller (HSSP)
- AN44168 – PSoC™ 1 Device Programming using External Microcontroller (HSSP)

[2] サンプルコード

Modus Toolbox™

[3] プログラミング仕様

CY8C4XXX, CYBLXXXX Programming specifications

[4] アーキテクチャ リファレンス マニュアル

- PSoC™ 4000 family: PSoC™ 4 architecture reference manual
- PSoC™ 4100 and 4200 Family: PSoC™ 4 architecture reference manual
- PSoC™ 4100M/4200M Family: PSoC™ 4 architecture reference manual
- PSoC™ 41X7_BLE/42X7_BLE Family: PSoC™ 4 BLE architecture reference manual
- PSoC™ 4100S and PSoC™ 4100S Plus: PSoC™ 4 architecture reference manual
- PSoC™ 4000S: PSoC™ 4 architecture reference manual
- PSoC™ 4100PS: PSoC™ 4 architecture reference manual

[5] Register reference manuals

[6] Training videos

改訂履歴

版数	発行日	変更内容
**	2013-10-01	これは英語版 001-84858 Rev. **を翻訳した日本語版 001-89490 Rev. **です。
*A	2015-04-23	これは英語版 001-84858 Rev. *E を翻訳した日本語版 001-89490 Rev. *A です。
*B	2015-11-19	これは英語版 001-84858 Rev. *H を翻訳した日本語版 001-89490 Rev. *B です。
*C	2017-07-19	ロゴと著作権を更新しました。
*D	2020-01-08	これは英語版 001-84858 Rev. *M を翻訳した日本語版 001-89490 Rev. *D です。
*E	2021-05-10	テンプレートの変更を実施。 これは英語版 001-84858 Rev. *N を翻訳した日本語版 Rev. *E です。
*F	2024-09-02	これは英語版 001-84858 Rev. *O を翻訳した日本語版 001-89490 Rev. *F です。

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

Edition 2024-09-02
Published by
Infineon Technologies AG
81726 Munich, Germany

© 2024 Infineon Technologies AG. All Rights Reserved.

Do you have a question about this document?
Email: erratum@infineon.com

Document reference
001-89490 Rev. *F

重要事項
本手引書に記載された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。