AN68272

PSoC® 3、PSoC 4、PSoC 5LP および PSoC アナログ コプロセッサ UART ブートローダ

著者: Anu M D、Siddalinga Reddy

関連プロジェクト: あり

関連製品ファミリー: CY8C3xxx、CY8C42xx、CY8C4Axx、CY8C40xxS、CY8C41xxS、CY8C5xxx

ソフトウェア バージョン: PSoC Creator™ 3.3 SP2 以降

関連アプリケーション ノート: 完全なリストについては、関連アプリケーション ノートをご参照ください

更にコード用例をお求めでしょうか？以下の通り対応いたします。

PSoC のコード用例のリストにアクセスするには、コード用例のウェブページをご覧ください。PSoC 4 のビデオ ライブラリについてはここからご覧ください。

AN68272 は PSoC® 3、PSoC 4、PSoC 5LP および PSoC アナログ コプロセッサ用の UART ベースのブートローダについて説明します。本アプリケーション ノートは、PSoC Creator™を使用して UART ベースのブートローダ プロジェクトとブートローダブル プロジェクトを素早く簡単に構築する方法について説明します。また、UART ベースの組み込みブートローダ ホスト プログラムおよび C#ベースのブートローダ アプリケーションを構築する方法も示しています。

目次

1 はじめに 1
1.1 用語および定義 2
1.2 ブートローダの使用 3
1.3 ブートローダ機能フロー 3
1.4 ブートローダに移行する技術 4
2 プロジェクト 5
2.1 UART ブートローダ 5
2.2 PSoC 3 および PSoC 5LP ブートローダブル 13
2.3 PSoC 4 ブートローダブル 16
2.4 PC のホストを使用してブートローディング 19
2.5 組み込みホストを使用したブートローディーグ 22
3 プロジェクトのテスト 26
3.1 キットの設定 26
3.2 PSoC 3 のブートローディング 27
3.3 PSoC 4/PSoC アナログ コプロセッサをブートローディング 27
4 まとめ 27
5 関連アプリケーション ノート 27
6 関連のプロジェクト 28
A 付録 A – メモリ 29
B 付録 B – プロジェクト ファイル 34
C 付録 C – ホスト／ターゲットの通信 35
D 付録 D – ホスト コア API 38
E 付録 E – ブートローダおよびデバイス リセット 39
F 付録 F – その他のトピック 42
G 付録 G – C#ブートローダ ホスト アプリケーション 45
H 付録 H – キットの選択 49
ワールドワイド販売と設計サポート 51
製品 51
PSoC®ソリューション 51
サイプレス開発者コミュニティ 51
テクニカル サポート 51

1 はじめに

ブートローダは MCU システム設計の共通部分です。ブートローダにより、製品のファームウェアを現場で更新できます。工場では、ファームウェアは一般的に MCU の Joint Test Action Group (JTAG) あるいは ARM Serial Wire Debug (SWD) のインターフェースを介して最初に製品にプログラムされます。しかし、これらのインターフェースは通常現場ではアクセスすることができません。

ここがブートローディングが活躍するところです。ブートローディングは、USB、I²C、UART または SPI などの標準通信インターフェースを経由してユーザーがシステム ファームウェアをアップグレードすることを可能にするプロセスです。ブートローダはホストと通信して、新しいアプリケーション コードやデータを取得し、デバイスのフラッシュ メモリに書き込みます。

本アプリケーションノートでは、以下の項目について説明します:

- PSoC Creator を使用し、UART ブートローダを作成する方法
- ブートローダ ホストのトピック:
  - ブートローダ ホスト ツールの使用方法
  - ブートローダ ホスト システムの基本的な構築ブロックと機能
  - PSoC 5LP を使用し、組み込み UART ブートローダ ホストを作成する方法
  - PC ブートローダ アプリケーションの作成方法

本アプリケーション ノートはユーザーが PSoC および PSoC Creator 統合設計環境 (IDE) に精通していることを前提としています。PSoC 3、PSoC 4、PSoC 5LP、または PSoC アナログ コプロセッサの概念をご存知ない場合、AN54181 – Getting Started with PSoC 3、AN79953 – Getting Started with PSoC 4、AN77759 – Getting Started with PSoC 5LP、または AN211293 – Getting Started with PSoC Analog Coprocessor をそれぞれご参照ください。PSoC Creator が初めての方は PSoC Creator ホームページをご参照ください。

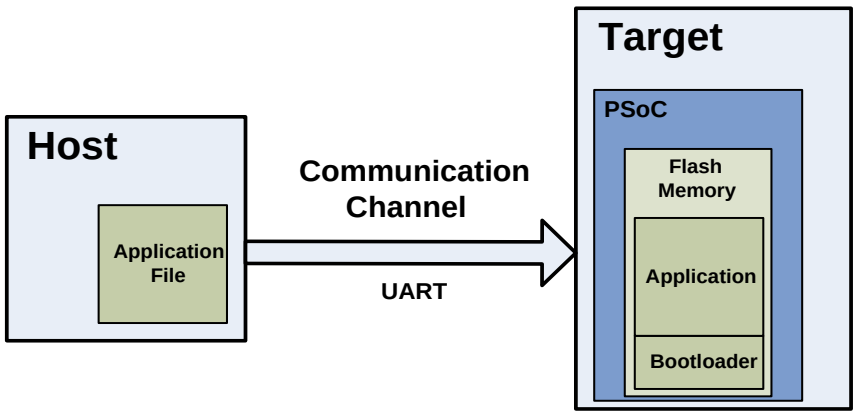
本アプリケーション ノートは、読者がブートローダの概念を理解していることも想定しています。これらの概念に慣れていない場合、AN73854 – PSoC 3, PSoC 4, and PSoC 5LP Introduction to Bootloaders をご参照ください。ブートローディングに関わる他のアプリケーションノート全ての一覧については、関連アプリケーション ノートをご参照ください。

最後に、本アプリケーション ノートは、読者が UART プロトコルと PSoC Creator UART コンポーネントに慣れていることを想定しています。UART コンポーネントに慣れていない場合、PSoC Creator UART コンポーネント データシートをご参照ください。PSoC Creator での UART コンポーネントを右クリックすることでもデータシートを取得できます。

1.1 用語および定義

図 1 はブートローダ システムの主な要素を図にしたものです。これは、製品に組み込まれたファームウェアが、通常動作とフラッシュ更新という 2 つの異なる目的で、通信ポートを使用できなくてはならないことを示します。フラッシュの更新方法を知る組み込みファームウェアの部分は bootloader (ブートローダ) と呼ばれています。図 1 に表示している他の用語は以下の段落の通り定義されます。

図 1. ブートローディングのシステム図



フラッシュ を更新するデータを提供するシステムは「ホスト」、更新されるシステムは「ターゲット」と呼ばれます。ホストは外部 PC (PC ホスト) またはターゲットと同じ PCB にある別の MCU (PSoC 5LP デバイスなどの組み込みホスト) です。

ホストからターゲットのフラッシュにデータを転送する動作は「ブートローディング」、「ブートロード動作」、または略して「ブートロード」と呼ばれます。フラッシュに配置されるファームウェアは「アプリケーション」または「ブートローダブル」と呼ばれます。

ブートローディングの別の一般的な用語はインシステム プログラミング (ISP) です。サイプレスは、似た名前ですが、「In-system serial programming (ISSP)」と呼ばれる異なる機能と「Host-Sourced Serial Programming (HSSP)」と呼ばれる動作を備えた製品があります。詳細については、AN73054 – PSoC 3 and PSoC 5LP Programming Using an External Microcontroller (HSSP) をご参照ください。

1.2 ブートローダの使用

ブートローダの通信ポートは、通常、ブートローダと実際のアプリケーション間で共有されます。ブートローダを使用する最初のステップは、アプリケーションではなくブートローダが実行されるようにターゲットを操作することです。

ブートローダが実行されると、ホストは通信チャネルを経由して「ブートロード開始」コマンドを送信できます。ブートローダが「OK」応答を送信すると、ブートローディングが開始できます。

ブートローディング中、ホストは新しいアプリケーション用のファイルを読み出し、それを解析してフラッシュ書き込みコマンドにし、それらのコマンドをブートローダに送信します。ファイル全体が送信された後、ブートローダは新しいアプリケーションに制御を渡すことができます。

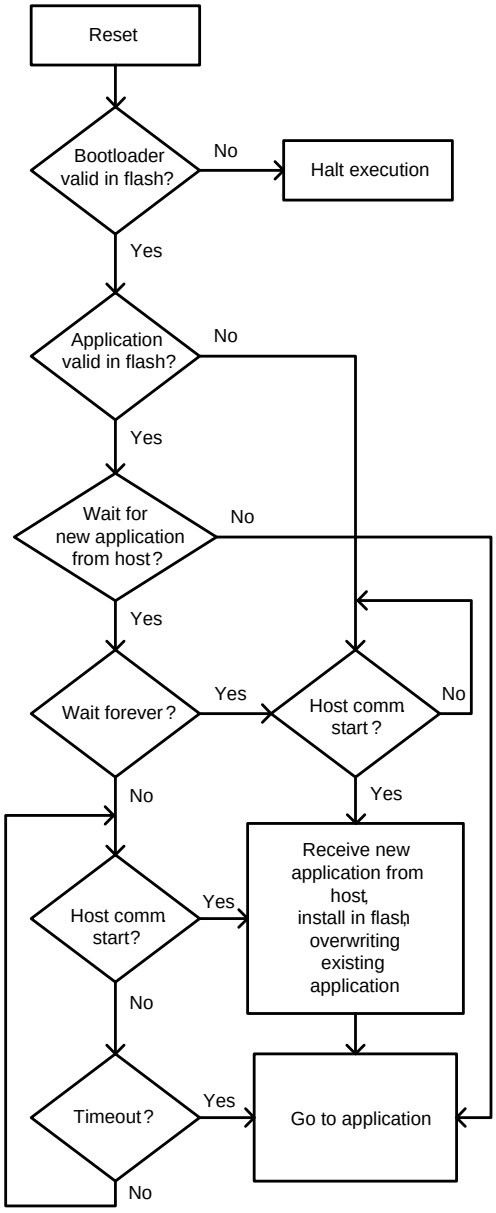
1.3 ブートローダ機能フロー

デバイスをリセットした時、ブートローダは通常最初に実行する機能です。その後、次の動作を実行します:

- アプリケーションを実行させる前に、その妥当性をチェック
- ホストとの通信を開始するタイミングを管理
- ブートロード／フラッシュ更新動作を実行
- アプリケーションに制御を渡す

図 2 は標準ブートローダ機能を示します。

図 2. ブートローダ機能フロー



1.4 ブートローダに移行する技術

前に説明したように、ブートローダはリセット時に実行する最初の機能です。図 2 に示すように、ブートローダ コードはアプリケーションに制御を渡す前に、短期間ホスト (からのコマンド) を待ちます。このことで、ホストはブートロード処理を開始する機会を逃すことがあります。しかし、ブートローディングを開始する別の方法があります。それは、アプリケーションまたはブートローダブルからブートローダに制御戻すことです。

1.4.1 ブートローダブル API

PSoC Creator のブートローダブル コンポーネントは、ブートローダを開始するためのアプリケーション プログラミング インターフェース (API) 関数 (`Bootloadable_Load()`) を備えています。これにより、ホストはいつでもブートロード動作を開始できます。

この方法の問題は、アプリケーションの更新を実行するためにアプリケーション コードに依存しなければならないことです。アプリケーションにブートローダへの制御の移管を妨げる欠陥があると、何が起こるでしょうか？

1.4.2 ブートローダのカスタマイズ

それよりも、ブートローダに無限の時間ホストを待たせる方が良いでしょう。それを行うためには、`Bootloader_Start()` を呼び出してその通常のルーチンを実行する前に、ブートローダ プロジェクトをカスタマイズしユーザー入力を確認します。

例えば、ブートローダは `Bootloader_Start()` を呼び出す前に、UART をモニターしユーザー コマンドをずっと待ちます。詳細については、AN73854 – PSoC 3, PSoC 4, and PSoC 5LP Introduction to Bootloaders をご参照ください。

2 プロジェクト

本節では、下記の PSoC Creator プロジェクトを作成する手順について説明します:

- UART ブートローダ
- ブートローダブル
- 組み込みブートローダ ホスト

このプロジェクトはサイプレス開発キットと一緒に使用するように設計されています。キットの選択はターゲット デバイスに基づきます。詳細は付録 H – キットの選択をご参照ください。キットに応じてピン接続の変更を必要とする場合があります。接続のタイプについては特定のキットに関する資料をご確認ください。プロジェクトは他のカスタム基板に容易に適用できます。

2.1 UART ブートローダ

本節では、UART ベースのブートローダ プロジェクトを作成し構築します。このプロジェクトの特徴の 1 つは、ブートロード中にキットの LED が点滅することです。

1. 新しい PSoC Creator プロジェクトを作成して、それを「UART_Bootloader」と名付けます。ターゲット デバイスを選択し、プロジェクトの新しいワークスペースを作成します。

注: PSoC Creator 3.1 以前のバージョンである場合、プロジェクトを作成するときにアプリケーション タイプを指定する必要があります。これを行うには、**Advanced** タブの隣にある「+」ボタンをクリックして、設定オプションを拡張します。アプリケーション タイプとして **Bootloader** を選択します。

2. 図 3 および図 4 に示すように、UART コンポーネントをトップ デザインの回路図に追加します。

図 3. PSoC 3 および PSoC 5LP 用の UART コンポーネント

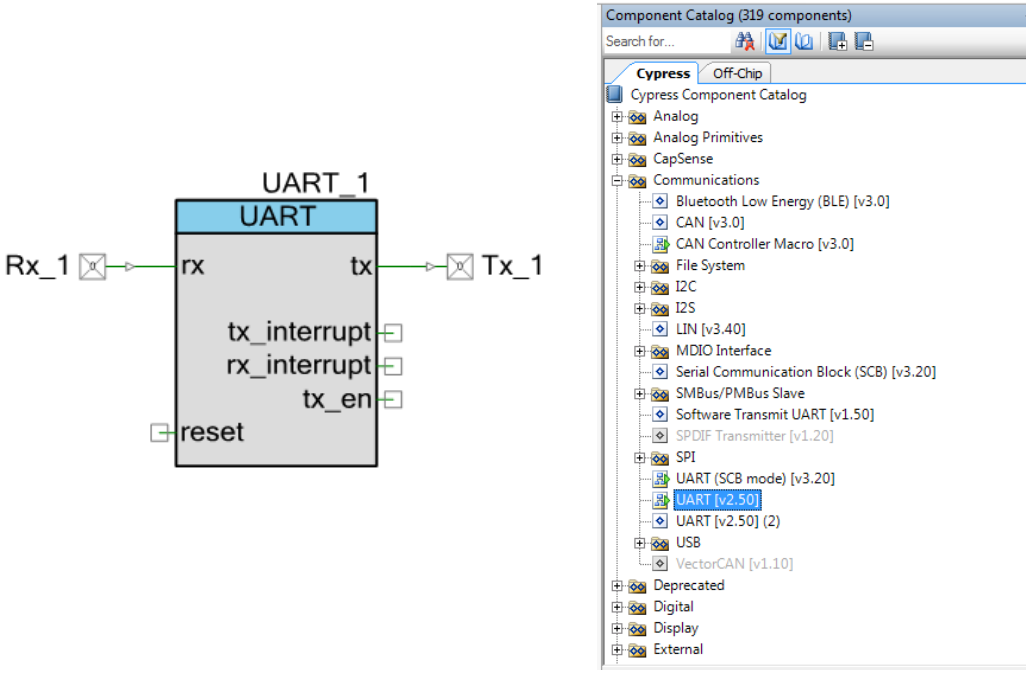
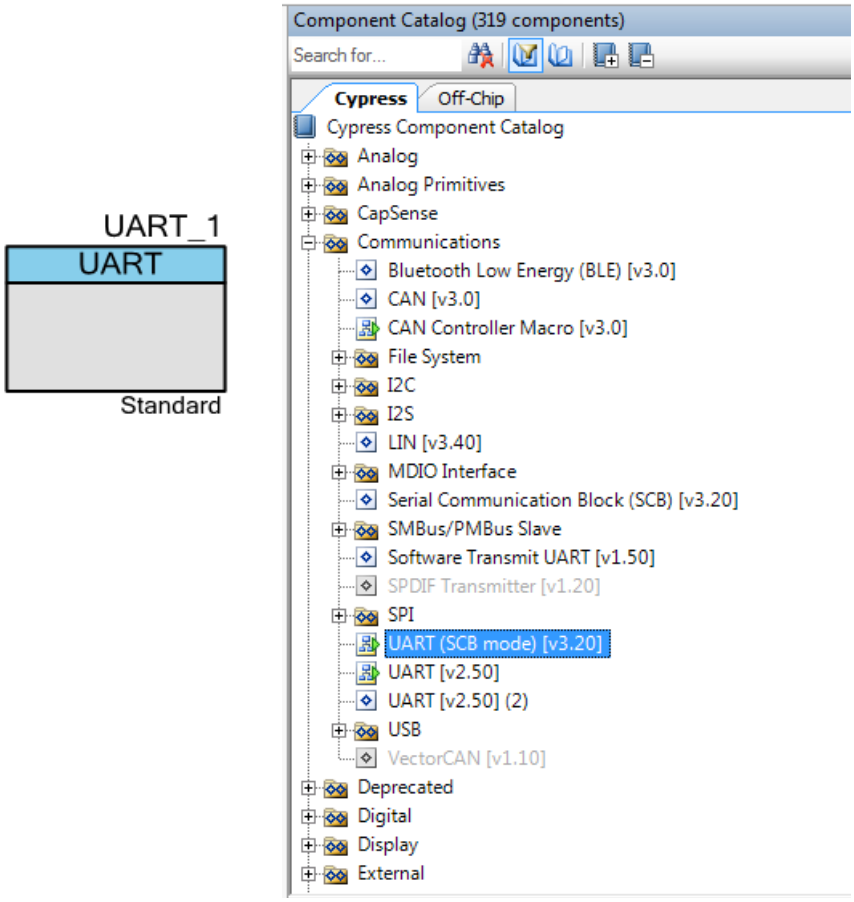
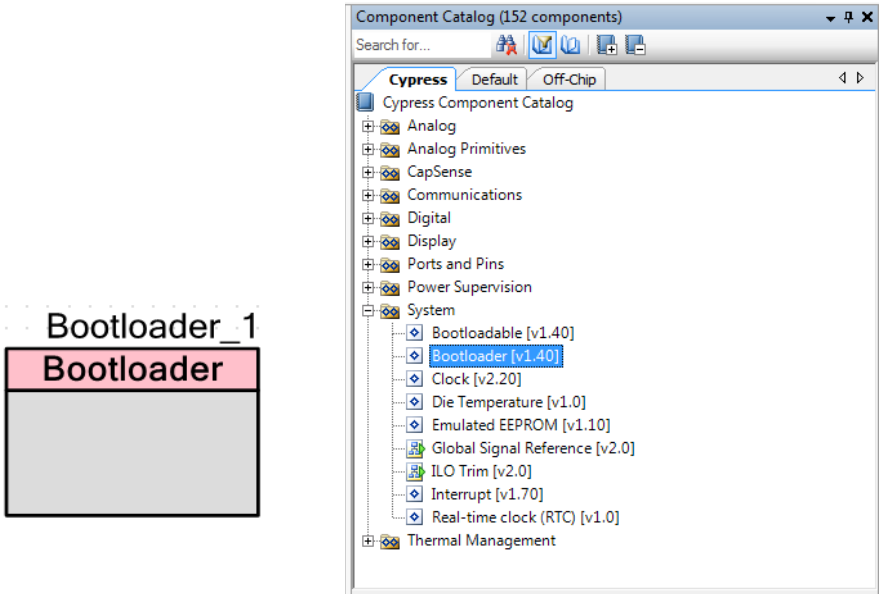


図 4. PSoC 4 および PSoC アナログ コプロセッサ用の UART コンポーネント



3. 図 5 に示すように、ブートローダ コンポーネントをトップ デザインの回路図に追加します。

図 5. ブートローダ コンポーネント



4. LED を点滅させるために、PWM (PSoC 4 および PSoC アナログ コプロセッサ用の TCPWM)、クロックおよびデジタル出力ピン コンポーネントを回路図に追加します。
5. 表 1 に示すように、コンポーネントとピンの名称を変更します。

表 1. ブートローダ プロジェクトのコンポーネント名

コンポーネント	名称
Bootloader_1	Bootloader
UART_1	UART
Rx_1	Rx
Tx_1	Tx
Clock_1	Clock
Pin_1	Pin_LED
PWM_1 / TCPWM_1	PWM

6. LED と抵抗が注釈コンポーネントとして追加されると、PSoC 3 および PSoC 5LP 用のプロジェクトのトップデザインは図 6 と同様のようになり、PSoC 4 および PSoC アナログ コプロセッサ用のトップデザインは図 7 と同様のようになります。

図 6. PSoC 3 および PSoC 5LP 用の UART_Bootloader プロジェクトのトップ デザイン

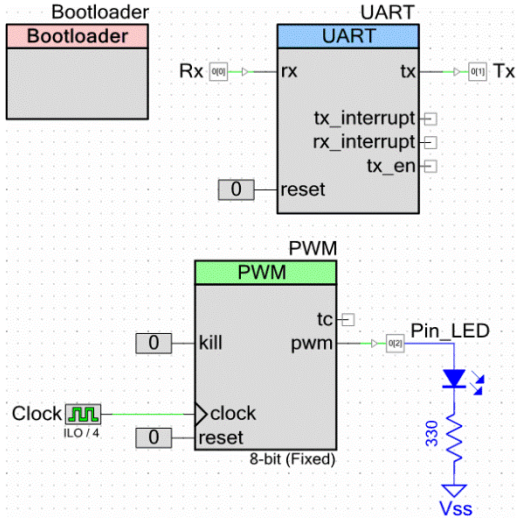
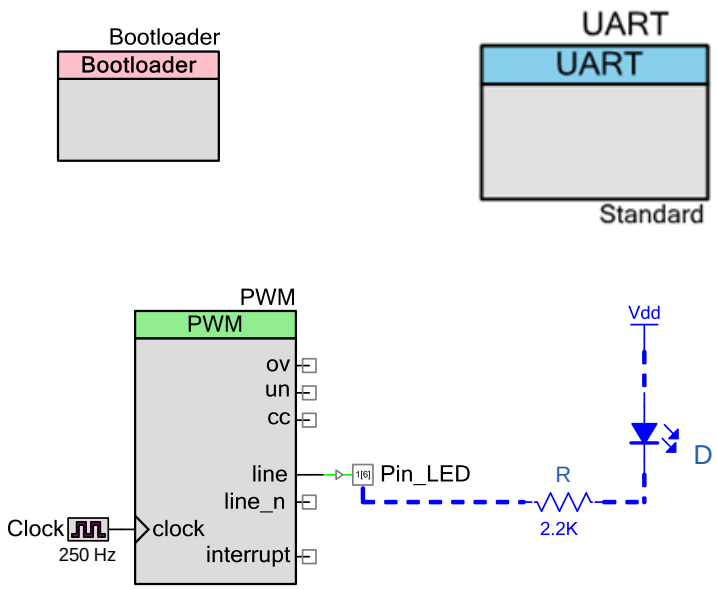


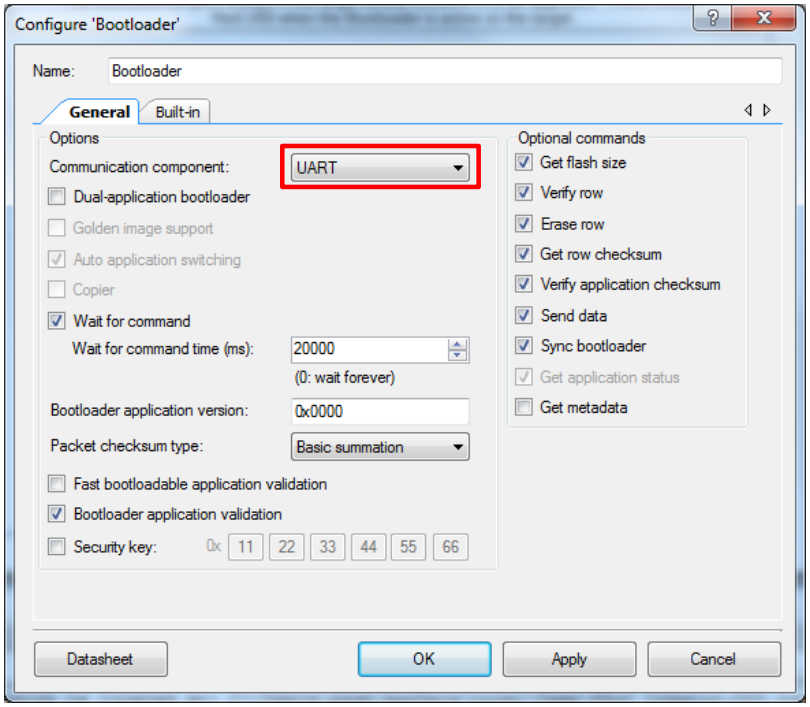
図 7. PSoC 4 および PSoC アナログ コプロセッサ用の UART ブートローダ プロジェクトのトップ デザイン



この例では、UART はリセットする必要がなく、そのため、リセット端末は論理 LOW「0」コンポーネントに接続されます。

7. ブートローダを設定するために、コンポーネント上をダブルクリックします。
8. 図 8 に示すように、**UART** を **Communication component** として選択します。他のパラメーターは初期設定のままにしてください。これらの設定パラメーターの詳細については、ブートローダ コンポーネント データシートをご参照ください。

図 8. ブートローダの設定



9. UART コンポーネントを設定するために、その上をダブルクリックします。デフォルトでは、フル UART モードであり、データレートが 57,600bps です。すべてのパラメータを初期設定のままにしてください。図 9 および図 10 は UART コンポーネントの基本設定タブを示します。

図 9. PSoC 3 および PSoC 5LP 用の UART の基本設定

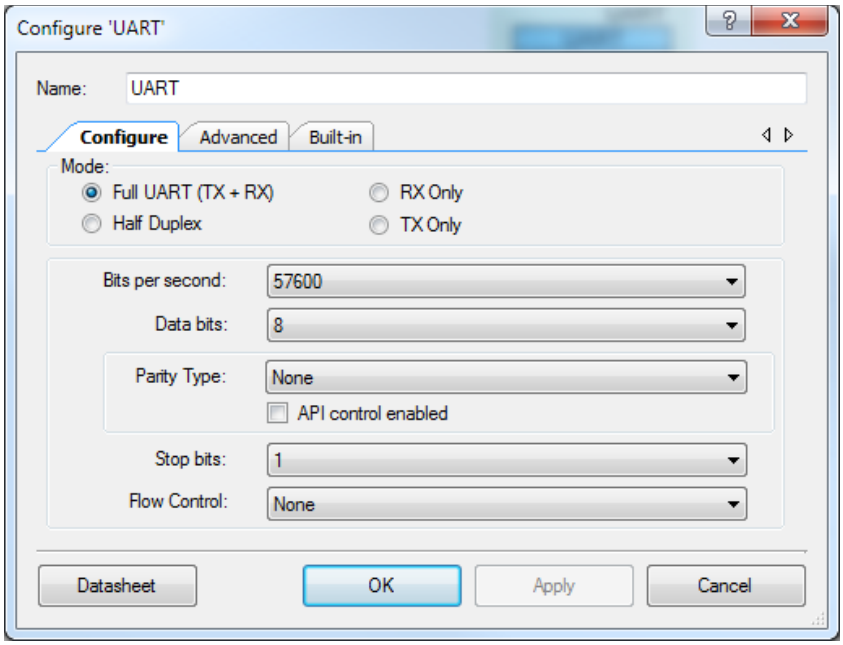
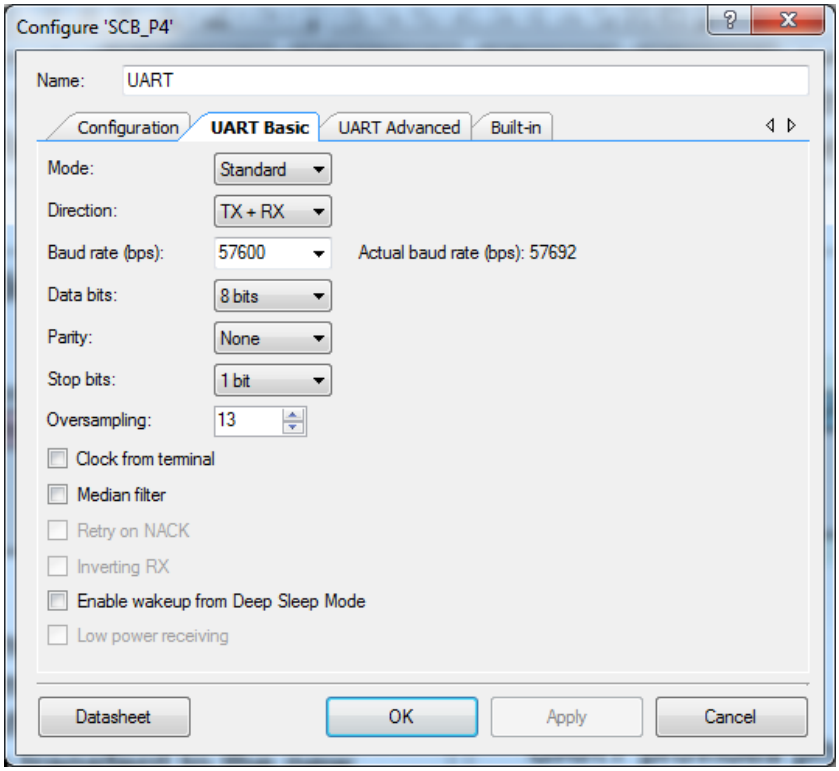


図 10. PSoC 4 および PSoC アナログ コプロセッサ用の UART の基本設定



10. UART コンフィギュレーション ウィンドウの **Advanced** タブ上をクリックします。
11. 通信オーバーフローを防止するために、受信 (Rx) および送信 (Tx) バッファ サイズの両方を 64 に設定します (ホストのパケット サイズは最大 64 バイト)。他のパラメーターは初期設定のままにしてください。図 11 および図 12 をご覧ください。

図 11. PSoC 3 および PSoC 5LP 用の UART の高度設定

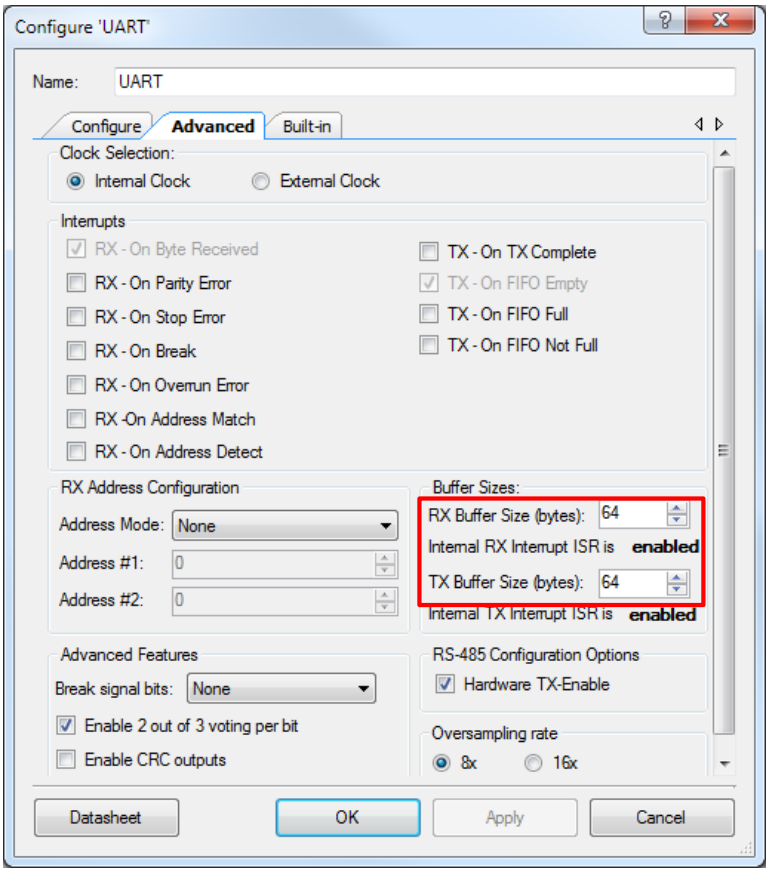
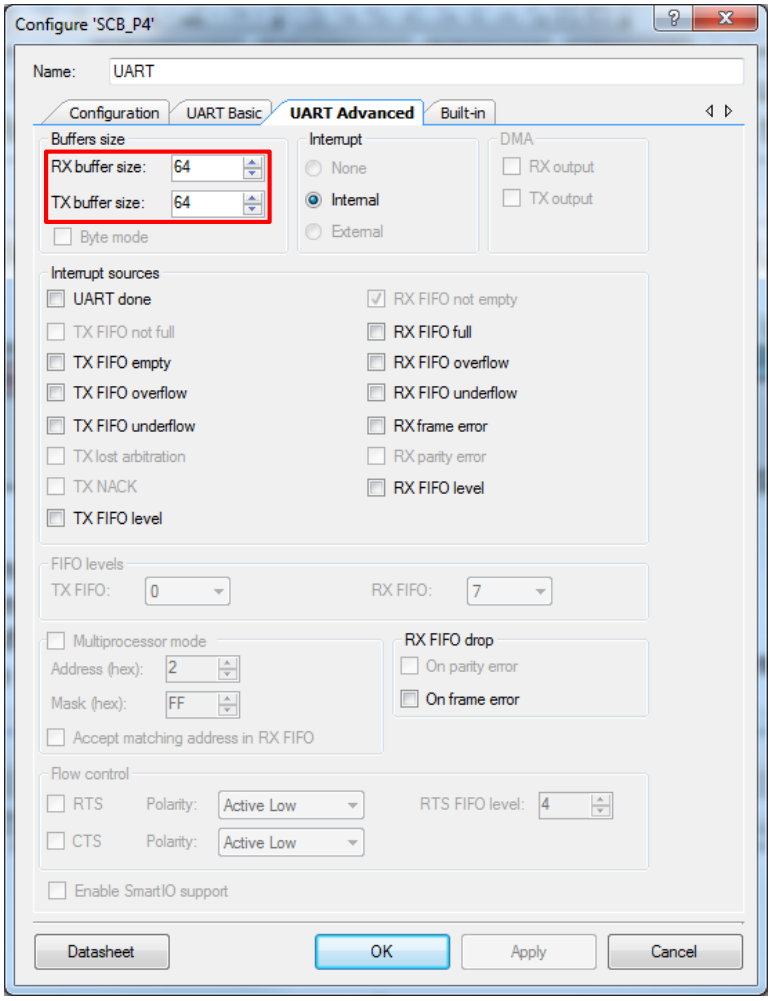


図 12. PSoC 4 および PSoC アナログ コプロセッサの UART の高度設定



12. PWM コンポーネントを設定するために、その上をダブルクリックします。**Period** を **255** に、**Compare** を **127** に設定します。他のパラメーターは初期設定のままにしてください。
13. クロック コンポーネントを設定するために、その上をダブルクリックします。**Frequency** を **ILO/4**、または～250Hz に設定します。他のパラメーターは初期設定のままにしてください。
14. Pin_LED コンポーネントに対しては、それらのパラメーターを初期設定のままにします。
15. ピン コンポーネントを物理ピンに割り当てます。**Workspace Explorer** ウインドウ内で、「*UART*_Bootloader.*cydwr*」ファイルをダブルクリックし、**Pins** タブ上をクリックします。表 2 は異なるキット用のピン割り当てを示します。ピン割り当てはキットに依存します。詳細はキット ユーザー ガイドをご参照ください。

表 2. 各キットに対する UART ブートローダ プロジェクトのピン割り当て

ピン名	CY8CKIT-030	CY8CKIT-050	CY8CKIT-042	CY8CKIT-041-40xx	CY8CKIT-041-41xx	CY8CKIT-048
\UART:rx\	P0[0]	P0[0]	P4[0]	P0[4]	P0[4]	P0[4]
\UART:rx\	P0[1]	P0[1]	P4[1]	P0[5]	P0[5]	P0[5]
Pin_LED	P6[2]	P6[2]	P1[6]	P3[4]	P3[4]	P1[4]

16. `Bootloader_Start()` 関数を *main.c* ファイルに追加します。この API 関数は全ブートロード動作を行います。これはリターンせず、ソフトウェアのデバイス リセットで終了します。そのため、この API 呼び出しの後のコードは実行されません。

    コード 1 が示すように、`PWM_Start()` 関数を追加して main()の PWM を初期化します。このコンポーネント API の詳細については、PWM コンポーネント データシートをご参照ください。

コード 1. ブートローダ内における PWM 初期化

```
void main()
{
  /* Initialize PWM */
  PWM_Start();

  Bootloader_Start();

  /* Uncomment this line to enable
     global interrupts. */
  /* CyGlobalIntEnable; */

  for(;;)
  {
    /* Place your code here. */
  }
}
```

17. プロジェクトをビルドし、ユーザーのターゲット デバイスの選択に基づく指定のキットにプログラムします。デバイスの選択に基づくキット関連の情報を入手するために、付録 H – キットの選択をご参照ください。

これで簡単な UART ベースのブートローダを作成しました。それはホストと通信してブートローダブル プロジェクトをダウンロードすることができます。 ブートローダはさまざまな方法で拡張とカスタマイズすることができます。詳細はブートローダ、UART コンポーネント データシートおよび AN73854 – PSoC 3, PSoC 4 and PSoC 5LP Introduction to Bootloaders をご参照ください。

注: ブートローダは PSoC フラッシュの一部を占有するため、アプリケーション用のフラッシュ量は削減されます。詳細については付録 F をご参照ください。

これで、このブートローダとともに使用されるブートローダブル アプリケーションを作成する方法を了解されることでしょう。

2.2 PSoC 3 および PSoC 5LP ブートローダブル

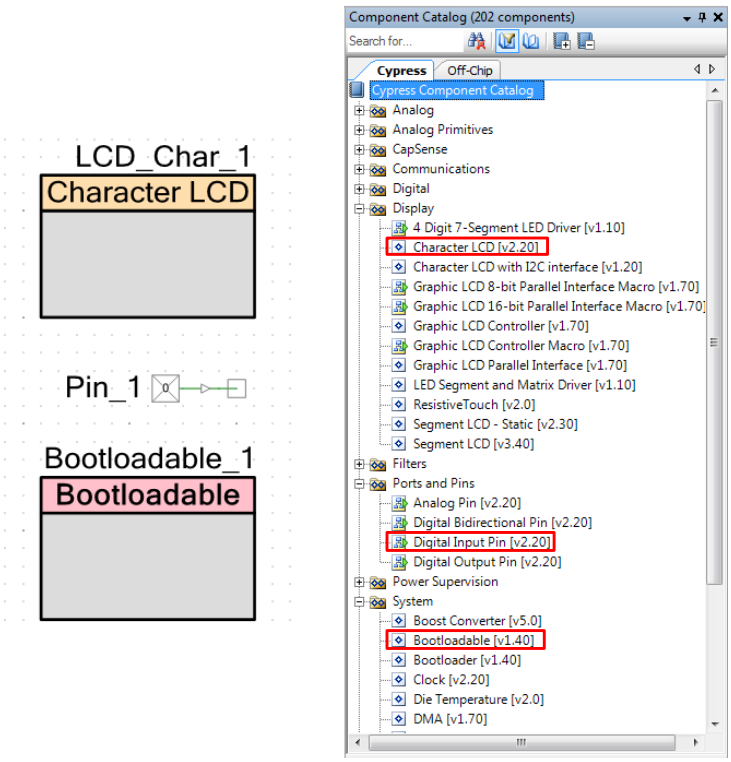
本節では、2 つのブートローダブル プロジェクトを作成します。それらは非常に似ています: 1 つはキットのキャラクタ LCD に「Hello」を表示し、もう 1 つは「Bye」を表示します。本節はこれらのブートローダブル プロジェクトを作成する手順を説明します。

1. 新しい PSoC Creator プロジェクトを作成し、「Bootloadable1」と名付けます。本プロジェクトと UART_Bootloader プロジェクト用のデバイスは同じである必要があります。

   注: PSoC Creator 3.1 以前のバージョンである場合、プロジェクトを作成するときにアプリケーション タイプを指定する必要があります。これを行うには、**Advanced** タブの隣にある「+」ボタンをクリックして、設定オプションを拡張します。アプリケーション タイプとして **Bootloadable** を選択します。

2. 本プロジェクトに対しては、ブートローダブル、デジタル出力ピン、および LCD コンポーネントが必要です。図 13 に示すように、これらのコンポーネントをトップ デザインの回路図に追加します。

図 13. Bootloadable1 プロジェクトのコンポーネント



3. 表 3 に従って、コンポーネント名を変更します。

表 3. ブートローダブル プロジェクト コンポーネント名

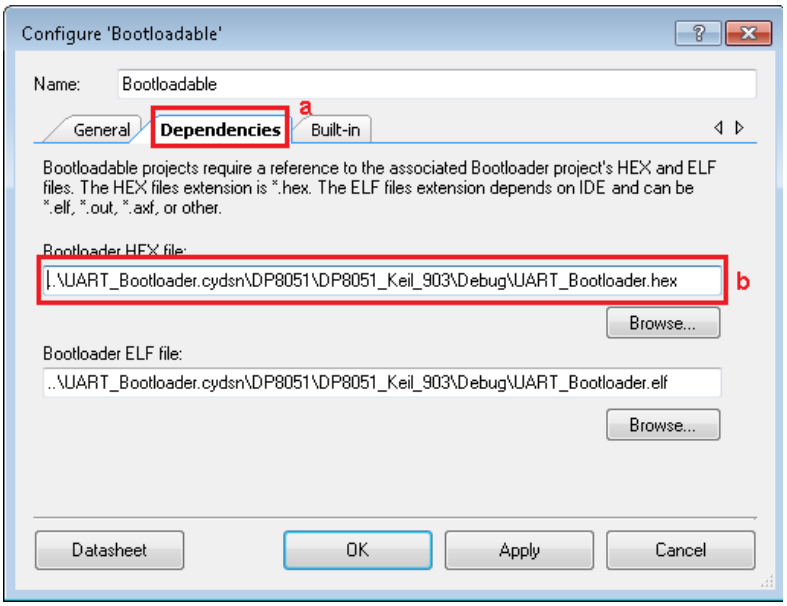
コンポーネント	名称
Bootloadable_1	Bootloadable
Pin_1	Pin_StartBootloader
LCD_Char_1	LCD_Char

4. ブートローダブル コンポーネントを設定するために、ダブルクリックします。

   A. ブートローダブル プロジェクトは常にブートローダ プロジェクトの*.hex* ファイルにリンクします。図 14 に示すように、コンポーネントを設定するために、ブートローダブル コンポーネントのコンフィギュレーション ウィンドウの **Dependencies** タブに移動します。

B. 図 14 に示すように、*UART_Bootloader.hex* ファイルを選択します。ブートローダ コンポーネントの設定の詳細情報については、ブートローダ コンポーネント データシートをご参照ください。

図 14. ブートローダブル コンポーネントの設定



UART_Bootloader.hex ファイルを、ブートローダ プロジェクトの Debug または Release フォルダに見つけることができます:

PSoC 3 がブートローダの場合、

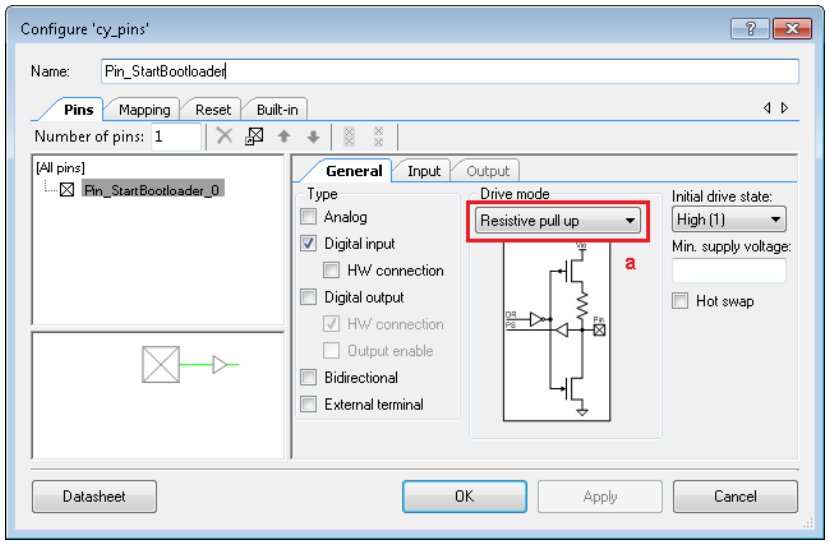
..\UART_Bootloader\UART_Bootloader.cydsn\DP8051\DP8051_Keil_951\Debug\UART_Bootloader.hex

PSoC 5LP がブートローダの場合、

..\UART_Bootloader\UART_Bootloader.cydsn\CortexM3\ARM_GCC_493\Debug\UART_Bootloader.hex

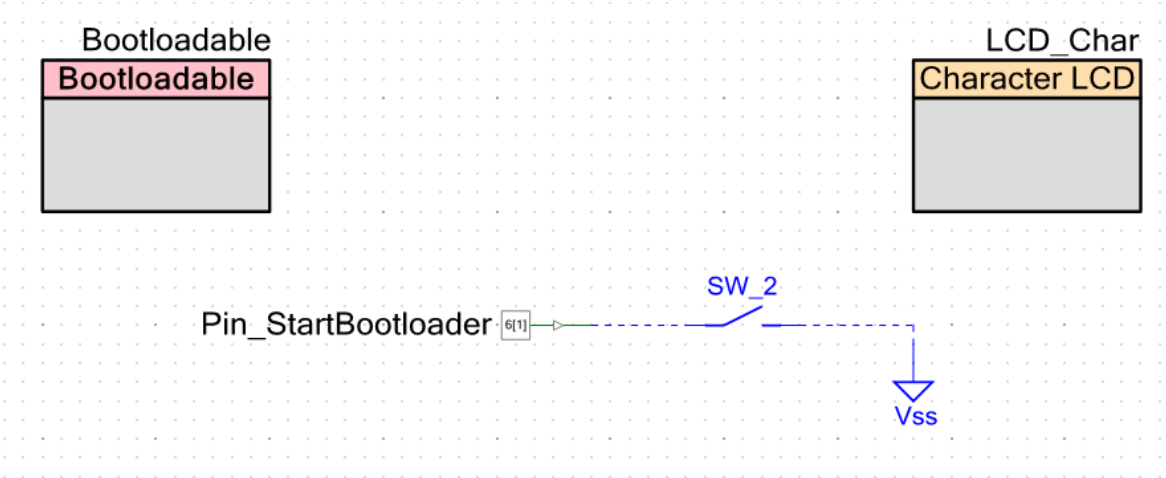
5. デジタル入力ピン「Pin_StartBootloader」は、アプリケーションからブートローダに戻すために使用されます。DVK ボタンを押すとグランドに短絡するので 、図 15 に示すようにピンの駆動モードを **Resistive Pull Up** に設定します。

図 15. デジタル入力ピンの設定



6. ボタンと入力ピンへの注釈コンポーネントが追加されて、トップ デザインが完了します。それは図 16 と同じようになります。

図 16. PSoC 3 および PSoC 5LP 用の Bootloadable1 プロジェクトのトップ デザイン



7. ピン コンポーネントを物理ピンに割り当てます。**Workspace Explorer** ウィンドウ内で、*Bootloadable1.cydwr* ファイルをダブルクリックして、図 17 の通りにピンを割り当てます。

図 17. Bootloadable1 プロジェクトのピン割り当て

Name	Port
\LCD_Char:LCDPort[6:0]\	P2[6:0]
Pin_StartBootloader	P6[1]

CY8CKIT-030 および CY8CKIT-050 キット基板では、LCD ピンが P2[6:0]にハードワイヤ接続され、SW2 が P6[1]にハードワイヤ接続されます。

8. 完成した Bootloadable1 プロジェクトは本アプリケーション ノートと関連付けられています。この関連するプロジェクトの *main.c* ファイルからのコードリストを、ユーザー プロジェクトの *main.c* ファイルに挿入してください。

   `main()` 関数は Pin_StartBootloader の状態を連続して確認します。このピンをグランドに短絡すると、`Bootloadable_Load()` API 関数がブートローダを起動するために呼び出されます。ブートローダは、ホストがブートロード動作を開始するのを無期限に待ちます。

9. プロジェクトを構築します。ブートローダブルプロジェクトがビルドされると、PSoC Creator は *.cyacd* ファイルを作成します。これはターゲットにブートロードされたファイルです。このファイルと内容についての詳細は、付録 B をご参照ください。
10. 「Bye」を表示する他のブートローダブル プロジェクトを作成するために、本節の前のステップを繰り返します。「Bootloadable2」プロジェクトと名付けます。2 つのプロジェクトの唯一の違いは、*main.c* のコードが「Hello」の代わりに「Bye」を表示することです。

注: PSoC Creator 3.0 以前のバージョンは、ブートローダが更新されると、ブートローダ プロジェクトに依存するブートローダブル プロジェクトをすべて再ビルドする必要があります。「Clean and Build」オプションをご使用ください。

PC ホストを使ってこのプロジェクトをブートロードするために、PC ホストを使用してのブートローディング をご覧ください。

2.3 PSoC 4 ブートローダブル

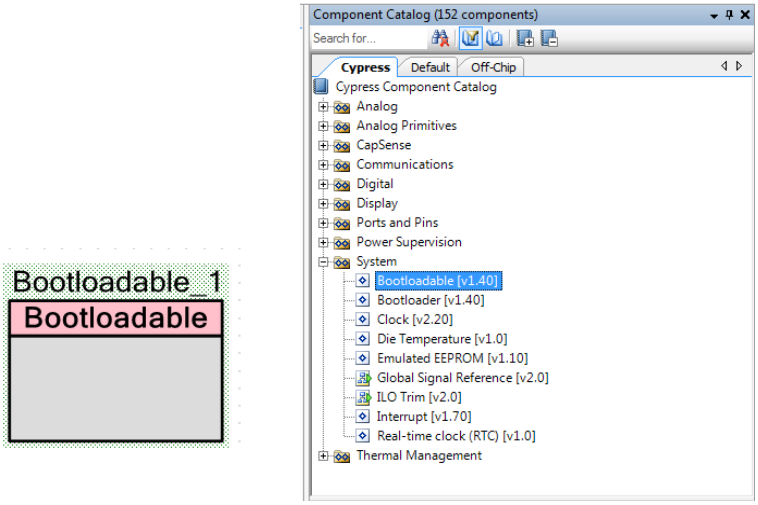
本節では、PSoC 4 および PSoC アナログ コプロセッサ用の 2 つのブートローダブル プロジェクトを作成します。これらのプロジェクトは CY8CKIT-042 キットに設計されますが、他の PSoC 4 および PSoC アナログ コプロセッサ デバイスを使用する別のキットに容易に適用されることができます。キット関連の情報については、付録 H – キットの選択およびキット ユーザー マニュアルをご参照ください。これらのプロジェクトは非常に似ています: 1 つはキット上の緑色 LED を点滅させ、もう 1 つは青色 LED を点滅させます。本節はこれらのブートローダブル プロジェクトを作成する手順を説明します。

1. 新しい PSoC Creator プロジェクトを作成し、「Bootloadable1」と名付けます。本プロジェクトと UART_Bootloader プロジェクト用のデバイスは同じである必要があります。

   注: PSoC Creator 3.1 バージョンの場合、プロジェクトを作成するときにアプリケーション タイプを指定する必要があります。これを行うには、**Advanced** タブの隣にある「+」ボタンをクリックして、設定オプションを拡張します。アプリケーション タイプとして **Bootloadable** を選択します。

2. 本プロジェクトに対しては、ブートローダブル、デジタル入力ピンとデジタル出力ピン コンポーネントが必要です。図 18 に示すように、これらのコンポーネントをトップ デザインの回路図に追加します。

図 18. Bootloadable1 プロジェクトのコンポーネント



3. 表 4 に従って、コンポーネント名を変更します。

表 4. Bootloadable1 コンポーネント名

コンポーネント	名称
Bootloadable_1	Bootloadable
Pin_1	Green_LED
Pin_2	Pin_StartBootloader

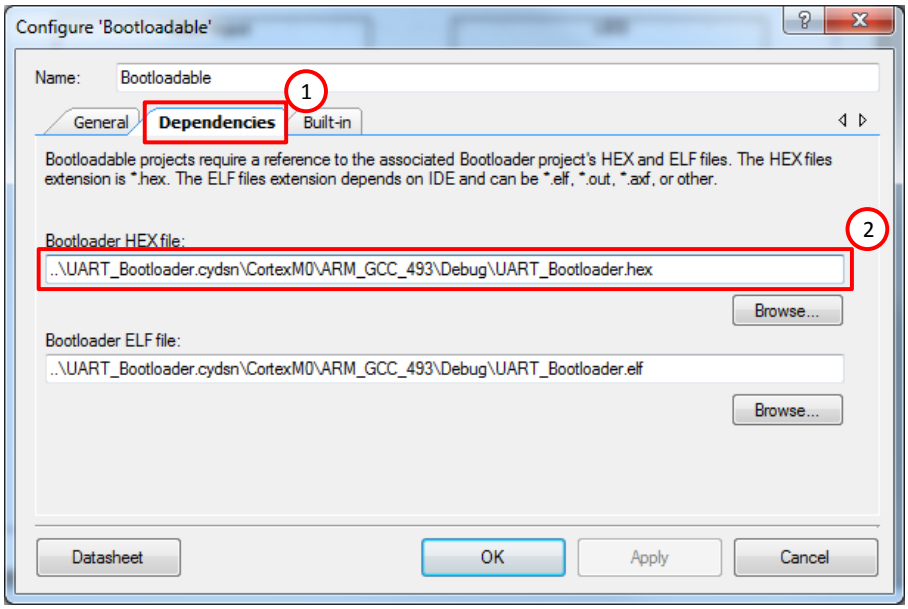
次のステップでは、これらのコンポーネントを設定します。

4. ブートローダブル コンポーネントを設定するために、ダブルクリックします。

   A. ブートローダブル プロジェクトは常にブートローダ プロジェクトの *.hex* ファイルにリンクします。図 19 に示すように、コンポーネントを設定するために、ブートローダブル コンポーネントのコンフィギュレーション ウィンドウの **Dependencies** タブに移動します。

   B. *UART_Bootloader.hex* ファイルを選択します。ブートローダ コンポーネントの設定の詳細情報については、ブートローダ コンポーネント データシートをご参照ください。

図 19. ブートローダブル コンポーネントの設定



UART_Bootloader.hex ファイルをブートローダ プロジェクトの Debug または Release フォルダで見つけられます:

PSoC 42xx の場合、

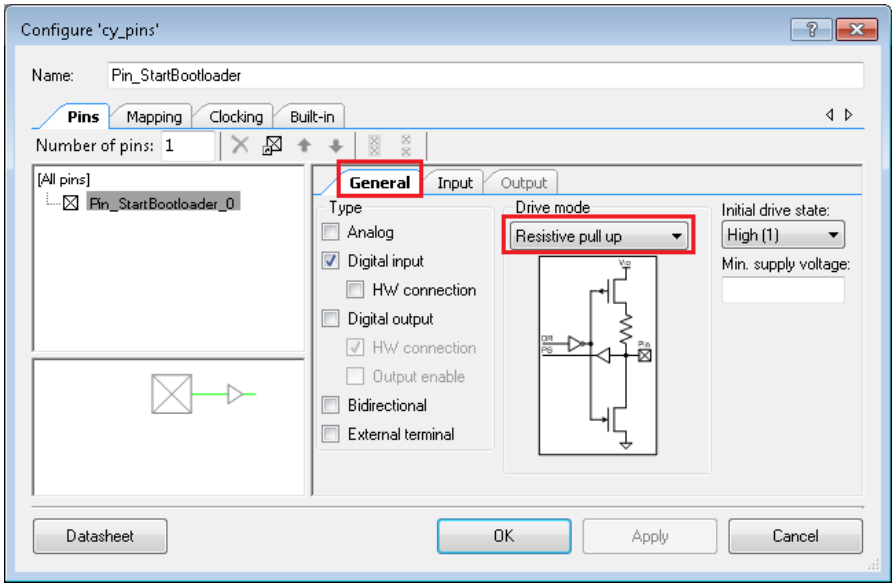
..\UART_Bootloader\UART_Bootloader.cydsn\Cortex M0\ARM_GCC_493\Debug\UART_Bootloader.hex

PSoC CY8C4Axx、CY8C40xxS、CY8C41xxS の場合、

..\UART_Bootloader\UART_Bootloader.cydsn\Cortex M0\ARM_GCC_493\Debug\UART_Bootloader.hex

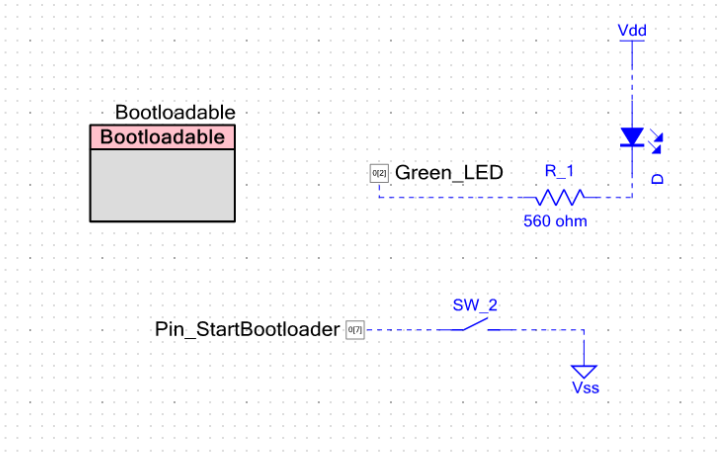
5. デジタル入力ピン (Pin_StartBootloader) はアプリケーションからブートローダに戻すために使用されます。DVK ボタンを押すとグランドに短絡するので、図 20 に示すようにピンの駆動モードを **Resistive Pull Up** に設定します。

図 20. デジタル入力ピンの設定



6. ボタンと入力ピン用の注釈コンポーネントが追加されて、トップ デザインが完了します。それは図 21 と同じようになります。

図 21. Bootloadable1 プロジェクトのトップ デザイン



7. ピン コンポーネントを物理ピンに割り当てます。**Workspace Explorer** ウインドウで、*Bootloadable1.cydwr* ファイルをダブルクリックして、ピンを割り当てます。表 5 は、異なるキットに対する bootloable1 プロジェクトのピン割り当てを示します。

表 5. キット用の Bootloadable1 のピン割り当て

ピン名	CY8CKIT-042	CY8CKIT-041-40xx	CY8CKIT-041-41xx	CY8CKIT-048
Green_LED	P0[2]	P2[6]	P2[6]	P2[6]
Pin_StartBootloader	P0[7]	P0[7]	P0[7]	P0[3]

CY8CKIT-042 キット基板では、緑色 LED が P0[2]にハードワイヤ接続され、SW2 が P0[7]にハードワイヤ接続されます。

8. PSoC 4 用の完成したブートローダブル プロジェクトは本アプリケーション ノートと関連付けられています。この関連するプロジェクトの *main.c* ファイルからのコードのリストを、ユーザー プロジェクトの *main.c* ファイルに挿入してください。

   `main()` 関数は Pin_StartBootloader の状態を連続して確認します。このピンをグランドに短絡する場合、`Bootloadable_Load()` API 関数はブートローダを起動するために呼び出されます。ブートローダは、ホストがブートロード動作を開始するのを無期限に待ちます。

9. プロジェクトを構築します。ブートローダブルプロジェクトがビルドされると、PSoC Creator は *.cyacd* ファイルを作成します。これはターゲットにブートロードされるファイルです。このファイルと内容についての詳細は、付録 B をご参照ください。

10. 青色 LED を点滅する他のブートローダブル プロジェクトを作成するために、本セクションの前のステップを繰り返します。「Bootloadable2」プロジェクトと名付けます。2 つのプロジェクトの唯一の違いはデジタル出力ピン (Blue_LED) の接続が異なることです。

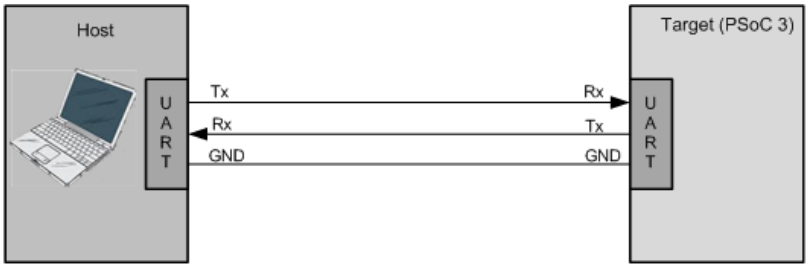
    注: PSoC Creator 3.0 以前のバージョンは、ブートローダが更新されると、ブートローダ プロジェクトに依存するブートローダブル プロジェクトを再ビルドする必要があります。「Clean and Build」オプションをご使用ください。

    これで、UART ブートローダ ホスト アプリケーション (PC ホスト) を使用して、このプロジェクトを対象の PSoC 4 または PSoC アナログ コプロセッサにブートロードすることができでいることでしょう。

2.4 PC ホストを使用してのブートローディング

図 22 に示すように、ブートローダ ホスト実行可能ファイルには、PC ホストからアプリケーションをブートロードするための本アプリケーション ノートが備えられています。*UARTBootloaderHost.exe* は *AN68272.zip* ファイル内の *Bootloader_Host_GUI_exe* フォルダにあります。それを吟味し、このツールを実行するために必要なソフトウェアをインストールするために、*Prerequisites.txt* ファイルにある指示に従って下さい。64 ビットおよび 32 ビットの Windows プラットフォーム用の実行ファイルに使用されるダイナミック リンク ライブラリ (DLL) は、*Bootloader_Host_GUI_exe* フォルダ内のそれぞれのフォルダにあります。この実行ファイルがマルチアプリケーション ブートローディングに対応しないことに注意してください。この目的には、PSoC Creator で用意されているブートローダ ホストを使用します。このツールをアクセスするには、**Tools** > **Bootloader Host** を選択してください。

図 22. PC ホストを使用してブートローディング



ブートローダ ホスト アプリケーションを使用して、アプリケーションをブートロードするには、以下の手順に従って下さい。 前述した通り、ブートロード動作を開始する前に、PSoC デバイスにブートローダ プロジェクトをプログラムする必要があります。

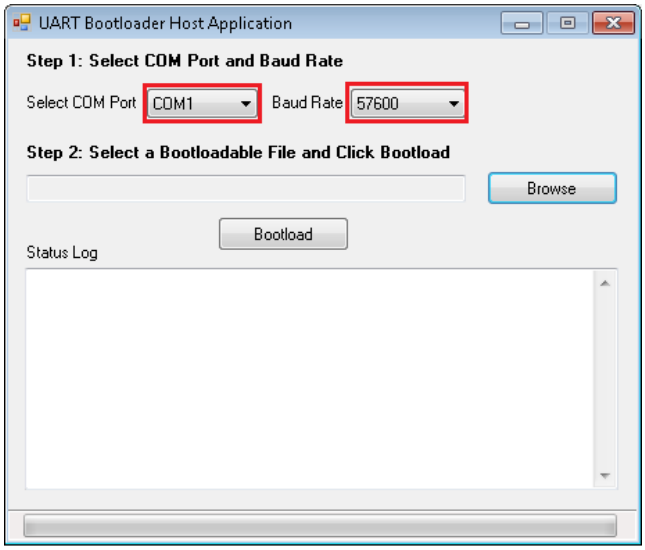
1. PSoC 3 または PSoC 5LP をブートロードするために、RS-232 シリアル ケーブルを CY8CKIT-030/CY8CKIT-050 のポート (P7) に接続します。

   PSoC 4 をブートロードするには、PSoC 4 および PSoC アナログ コプロセッサ キット上の PSoC 5LP に USB–UART ブリッジが用意されているため、RS-232 シリアル ケーブルは必要とされません。そのため、PSoC 4 の UART ポートピンとキット上のオンボード PSoC 5LP 間の外部接続の提供のみが必要となります。例えば、CY8CKIT-042 では、P0[0]をヘッダ J8 のピン 10 に接続し、P0[1]をヘッダ J8 のピン 9 に接続します。

2. ブートローダ ホスト アプリケーション (*UARTBootloaderHost.exe*) を開きます。図 23 に示すように、適切な COM ポートおよびボー レートを選択します。選択したボー レートはブートローダ プロジェクトの UART コンポーネントで設定されたボー レートと同じである必要があります (9 の図 9 をご参照ください)。

   ご自身のコンピューターの COM ポートは、デバイス マネージャのポート (COM および LPT) カテゴリに一覧表示されます。USB-to-UART ブリッジを使用する場合、それはエニュメレーション後、このカテゴリに表示されます。COM ポート番号は COM ポート名に続いて括弧内表記されます。

図 23. ブートローダ ホスト アプリケーション



3. ブートローダブル プロジェクトの Debug/Release フォルダ内の適切なブートローダブル ファイルを選択します:

   PSoC 3 がブートローダの場合、

   ...\Bootloadable1.cydsn\DP8051\DP8051_Keil_951\Debug\Bootloadable1.cyacd

   PSoC 5LP がブートローダの場合、

   ...\Bootloadable1.cydsn\CortexM3\ARM_GCC_493\Debug\Bootloadable1.cyacd

   PSoC 4000S/4100S または PSoC アナログ コプロセッサがブートローダの場合、

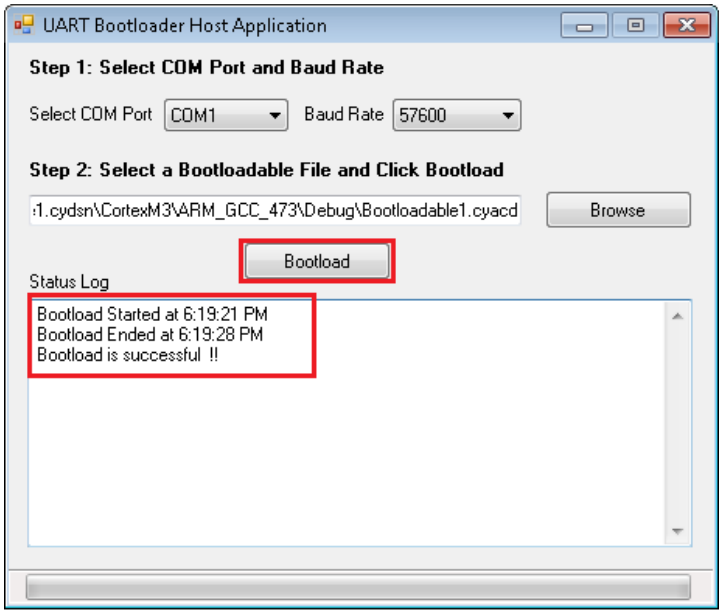
   ...\Bootloadable1.cydsn\CortexM0p\ARM_GCC_493\Debug\Bootloadable1.cyacd

   PSoC 4200 がブートローダの場合、

   ...\Bootloadable1.cydsn\CortexM0\ARM_GCC_493\Debug\Bootloadable1.cyacd

4. *.cyacd* ファイルをブラウズし、**Bootload** ボタンをクリックしてブートローディングを開始します。図 24 に示すように、ブートローディーグ状態は **Status Log** セクションに表示されます。

図 24. ブートローダブルプロジェクトのダウンロード



5. ブートロード動作が成功して完了した後、PSoC 3 または PSoC 5LP の場合、「Hello」メッセージが CY8CKIT-030/CY8CKIT-050 の LCD 上に表示されます。PSoC 4 および PSoC アナログ コプロセッサの場合、キット上の緑色 LED の点滅を開始します。

   注: KitProg 2.16 では、ブートロード動作が成功して完了した後、ブートロードしたアプリケーションに制御を渡すためにデバイス リセットが必要となります。これは既知の問題で、KitProg の後のバージョンで解決される予定です。

6. 再度ブートロードするために、DVK 上の SW2 を押してください。これにより、PSoC デバイスはブートローダ モードになります。ステップ 3 から 5 まで繰り返てし再度ブートロードをします。

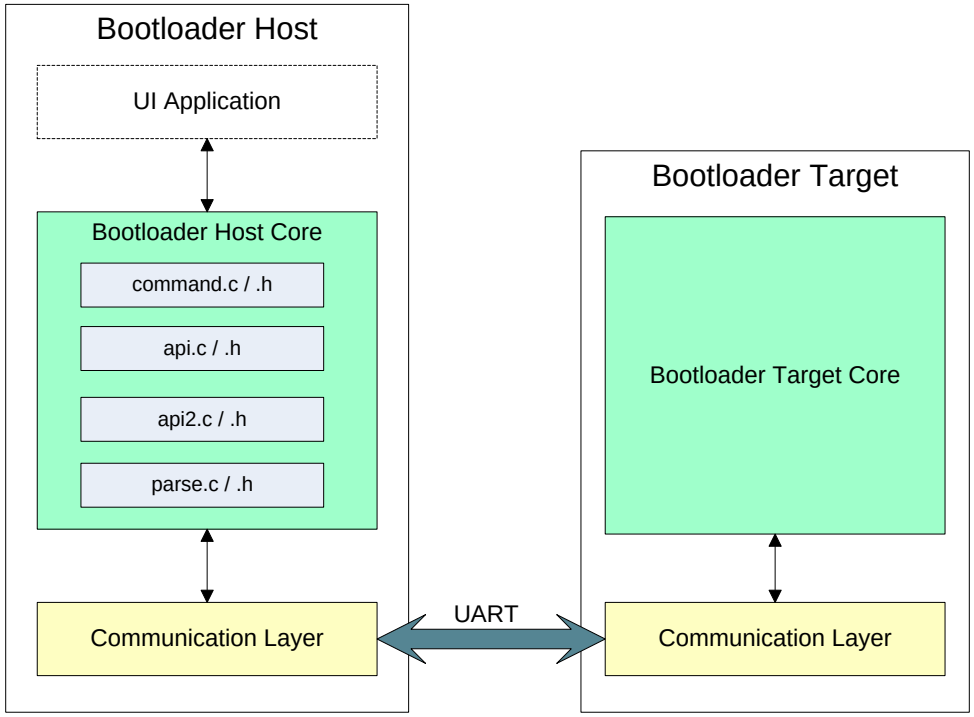
2.5 組み込みホストを使用したブートローディング

用例のプロジェクトを学ぶことに加えて、ブートローダ ホスト プログラムの一般的な構造を理解することも有用です。これにより、ユーザーが独自のブートローダ ホスト システムをビルドできます。

2.5.1 ブートローダ ホスト プログラム

図 25 はブートローダ システムのプロトコルレベルでの図を説明します。ブートローダのホストとターゲットはそれぞれ 2 つのブロック (コアと通信層) があります。

図 25.ブートローディングのプロトコルレベル図



- ブートローダ ホスト コアは全てのブートローディング動作を実施します。つまり、コマンド パケットとフラッシュ データをターゲットに送信します。ターゲットからの応答により、ブートローディングを続けるかどうか決定します。
- ブートローダのターゲット コアはホストからのコマンドを解読し、フラッシュ ルーチン (行の消去、行のプログラム、行の確認など) を呼び出すことでコマンドを実行し、応答パケットを形成します。
- ホストとターゲットの両方にある通信層は、ブートローディング プロトコルへの物理層サポートを提供します。これはこの機能を実行するための固有 API である通信プロトコル (UART) を含んでいます。この層はホストとターゲット間のプロトコル パケットの送受信を担当します。

2.5.2 ブートローダ システム API

ターゲット コアと通信層用の全ての API は、ユーザーがブートローダ プロジェクトをビルドする時に、PSoC Creator により自動的に生成されます。

ホスト コアに対応した API も PSoC Creator により提供され、以下のリンクに見つけることができます:

<インストール フォルダ> \ PSoC Creator \ 3.3 \ PSoC Creator \ cybootloaderutils

これら API ファイルの詳細情報については、付録 D をご参照ください。

ユーザーが書き込む必要がある唯一のコードは、ファイルのペア *communication_api.c /.h* 内にある、通信層に対するホスト側の API 関数です。関数は 4 つあります: `OpenConnection()`、`CloseConnection()`、`ReadData()` および `WriteData()`。これらは「CyBtldr_CommunicationsData」構造体内の関数ポインタで指定され、*cybtldr_api.h* 内にて定義されます。

これらの API を使って C#でユーザーご自身のホストを作成する詳細については、付録 F – その他のトピックをご覧ください。

2.5.3 UART ブートローダ ホスト プロジェクトの作成手順

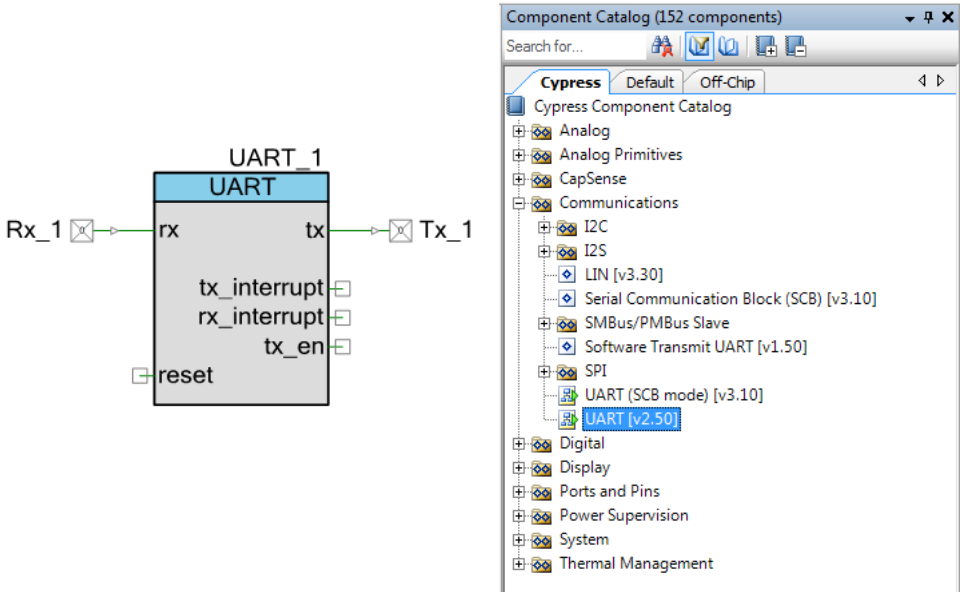
本節では、PSoC 5LP を使用して、他の PSoC デバイスをブートロードできる、組み込み UART ブートローダ ホスト プロジェクトの作成方法について説明します。このプロジェクトにより、スイッチを交互に押下する度に、ホストは 2 つの異なるブートローダブル ファイル (*.cyacd* ファイル) をブートロードできます。

1. 新規の PSoC Creator プロジェクトを作成してから、PSoC 5LP をターゲット デバイスとして選択し、プロジェクトを「UART_Bootloader_Host」と名付け、そのプロジェクト用の新規のワークスペースを作成します。

   注: PSoC Creator 3.1 バージョンの場合、プロジェクトを作成するときにアプリケーション タイプを指定する必要があります。これを行うには、**Advanced** タブの隣にある「+」ボタンをクリックして、設定オプションを拡張します。アプリケーション タイプとして **Normal** を選択します

2. 図 26 に示すように、UART コンポーネントをトップ デザインの回路図に追加します。また、デジタル入力ピンおよびキャラクタ LCD コンポーネントをトップ デザインに追加します。

図 26. UART コンポーネント



3. コンポーネントの名称を表 6 の通りに変更する。

表 6. UART ブートローダ ホスト プロジェクトのコンポーネント リスト

コンポーネント	名称
UART_1	UART
Rx_1	Rx
Tx_1	Tx
Pin_1	Pin_Switch
LCD_Char_1	LCD_Char

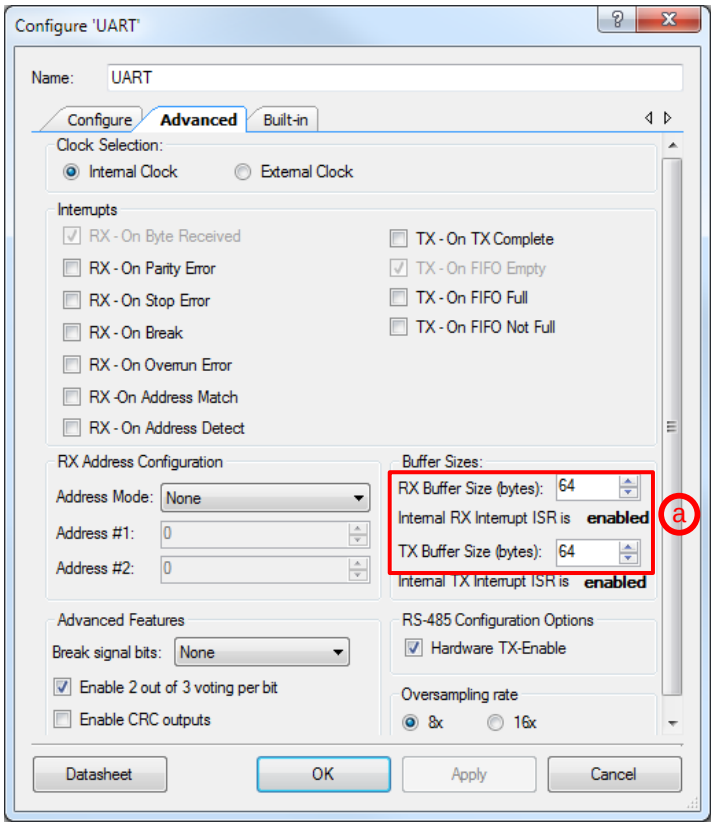
次のステップでは、これらのコンポーネントを設定します。

4. UART コンポーネントを設定するために、ダブルクリックします。デフォルトでは、フル UART モードであり、データ レートが 57,600bps です。すべてのパラメータを初期設定のままにしてください。

   注: プロジェクトはサポートされている任意のデータ レートで実行することができますが、そのデータ レートはブートローダ プロジェクトのデータ レートと同じである必要があります。

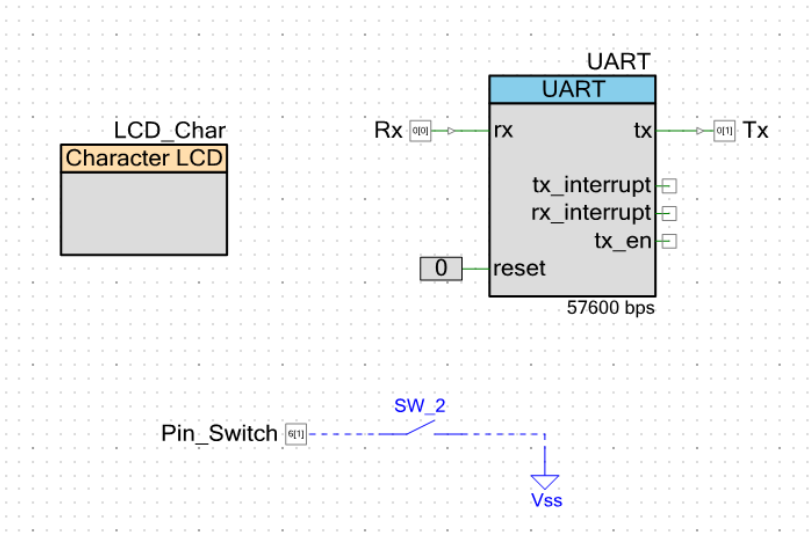
5. 図 27 に示すように、コンポーネント コンフィギュレーション ウィンドウの **Advanced** タブで、通信オーバーフローを避けるために送信 (Tx) および受信 (Tx) のバッファ サイズを 64 に設定してください (ホスト パケットは最大 64 バイト) 。

図 27. UART 高度設定



6. デジタル入力ピンの Pin_Switch は、ホスト内でのブートローディング動作を起動するのに使用されます。キットのボタンを押すと、ピンがグランドに短絡します。そのため、このピンがプルアップ抵抗を持つように設定する必要があります。
7. ボタンへの注釈コンポーネントが追加されて、このプロジェクトのトップデザインは、図 28 と同じようになります。

図 28. UART_Bootloader_Host プロジェクトのトップデザイン



8. 入力と出力のピンを割り当てます。**Workspace Explorer** ウインドウ内で、*UART_Bootloader_Host.cydwr* ファイルをダブルクリックし、ピンを図 29 のように割り付けます。

図 29. UART_Bootloader_Host プロジェクトのピン割り当て

Name	Port
\LCD_Char:LCDPort[6:0]\	P2[6:0]
Pin_Switch	P6[1]
Rx	P0[0]
Tx	P0[1]

CY8CKIT-050 キット基板では、LCD ピンが P2[6:0]にハードワイヤ接続され、SW2 が P6[1]ハードワイヤ接続されています。指定されたポート ピン (P4) を TX と RX (P5) に接続するようにキット基板の配線を行います。

9. ファームウェアをこのプロジェクトに追加します。UART_Bootloader_Host プロジェクトは本アプリケーション ノートに添付されています。この関連するプロジェクトの *main.c* ファイルから、ユーザー プロジェクトの *main.c* ファイルに、リストされたコードを挿入してください。

   main.c の `main()` 関数は Pin_Switch の状態を連続的に確認します。Pin_Switch はグランドに接続した時、ブートローディングが起動されます。*main.c* ファイルには `BootloadStringImage()` と呼ばれている関数があり、これは *device.h* で定義されています. この関数は、ブートローダ ホスト API ファイル (ホスト コア、ページ 22 の図 25 をご参照ください) を使用し、*.cyacd* ファイルをブートロードします。

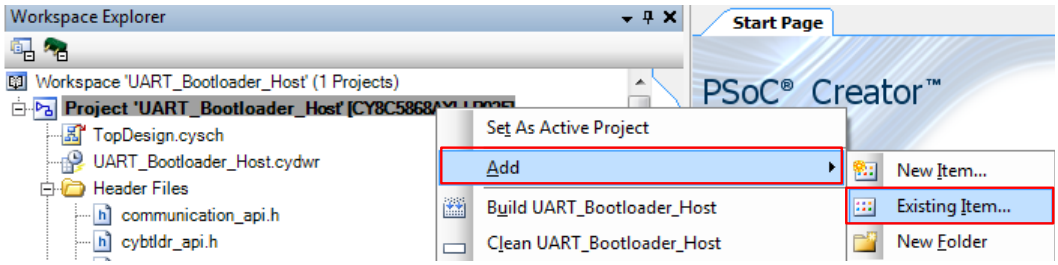
   `main()` 関数は「toggle」と呼ばれる別の変数があります。ボタンを押下する度に、「0」と「1」の間で切り替わります。これにより、ホストに交互のブートローダブル ファイルを選択させます。

10. 前に説明した通り、ブートローダ ホスト コアは 4 つの API ファイルでビルドされています。これらのファイルは全てのホスト ブートローディング動作を行います。これらのファイルをユーザは、ユーザのプロジェクト内に含める必要があります。これらの API ファイルを次の場所に見出してください:

    <インストール フォルダ> \ *PSoC Creator* \ *3.3* \ *PSoC Creator* \ *cybootloaderutils*

    図 30 に示すように、これらのファイルを取り込むために、**Workspace Explorer** ウィンドウへ行き、プロジェクト名を右クリックして、**Add** > **Existing Item** を選択します。PSoC Creator が提供する次のファイルを追加します: *cybtldr_api.c /.h*、*cybtldr_command.c /.h*、*cybtldr_parse.c / .h* および *cybtldr_utils.h*。

図 30. API ファイルの追加



11. API ファイルのブートローディングに加えて、ホストは通信層のサポートも必要とします。このサポートは *communication_api.c /.h* ファイルを追加することで提供されます。ユーザーは、本アプリケーション ノートに関連する UART_Bootloader_Host プロジェクトからこれらのファイルの内容を取り込んでください (これらのファイルをプロジェクトに追加するには、前述のステップに従ってください)。本アプリケーション ノートに添付されたプロジェクトからコピーし、これらのファイルを更新します。

12. ここで、ブートローダブルファイルをホスト システムに組み込みます。ブートローダブル ファイルを構築すると、*.cyacd* ファイルが生成されます。ファイルは.*hex* の出力ファイルと似ています。*.cyacd* ファイルの詳細情報については、付録 B をご参照ください。

    A. 各ラインが配列の要素であるように、このファイルの内容を文字列の配列の形式でコピーします。ユーザーは 2 つのブートローダブル ファイルを持っているため、「StringImage1」および「StringImage2」と名付けられた 2 つの配列を定義する必要があります。各配列に対しては、その配列にある行数を格納するマクロを定義します。これらの配列を *StringImage.h* という別のファイルに定義します (文字列を定義する前に、このファイルをプロジェクトに追加する必要があります)。

    B. 本アプリケーション ノートに関連する UART_Bootloader_Host プロジェクト内にある *StringImage.h* ファイルをご参照ください。

または、本アプリケーション ノートと共に提供されている Windows C#アプリケーションを使用し、*StringImage.h* ファイルを生成します。

13. プロジェクトをビルドし、CY8CKIT-050 キットの PSoC 5LP デバイスにプログラムします。

3 プロジェクトのテスト

UART_Bootloader_Host プロジェクトの *main.c* ファイルには「TARGET_DEVICE」と呼ばれるマクロがあります。このマクロは PSoC 3、PSoC 4、PSoC アナログ コプロセッサおよび PSoC 5LP の中から対象のデバイスを選択するのに使用されます。デフォルトでは、PSoC_3 (TARGET_DEVICE を含む同じファイルにあるマクロ) として定義されます。PSoC 4、PSoC アナログ コプロセッサ、または PSoC 5LP を対象のデバイスとして使用する場合は、このマクロの定義をそれぞれ「PSoC_4」、「PSoC_AC」または「PSoC_5LP」に変更します。

3.1 キットの設定

プロジェクトをテストするために、次のようにキットを設定します:

CY8CKIT-030 には:

1. PSoC 3 デバイスを UART_Bootloader プロジェクトでプログラムします。
2. J10 および J11 ジャンパを 5V にセットします。
3. キャラクタ LCD をポート 2[6:0]に接続します。

PSoC 4 または PSoC アナログ コプロセッサベースのキットには:

1. PSoC 4 または PSoC アナログ コプロセッサを UART_Bootloader プロジェクト (PSoC 4 および PSoC アナログ コプロセッサ プロジェクト) でプログラムします。
2. ジャンパを 5V にセットします。

CY8CKIT-050 は:

1. PSoC 5LP を UART_Bootloader_Host プロジェクトでプログラムします。
2. J10 および J11 ジャンパを 5V にセットします。
3. キャラクタ LCD をポート 2[6:0]に接続します。

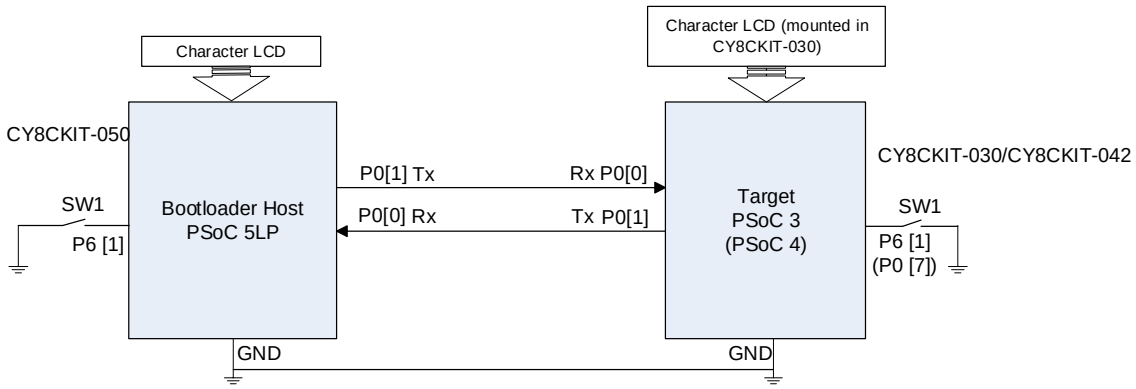
2 つの DVK 間で次の接続を行います。この例では CY8CKIT-030 および CY8CKIT-042 キットは対象のデバイスとして見なされます。

1. CY8CKIT-030 (CY8CKIT-042) の P0[0] を CY8CKIT-050 の P0[1]に接続します。
2. CY8CKIT-030 (CY8CKIT-042) の P0[1]を CY8CKIT-050 の P0[0]に接続します。
3. キットのグランド ピンを短絡します。

これらの接続を 図 31 に図示します。この接続はブートローダプロジェクトで選択された UART ピンにより変わる可能性があります。

注: 図 31 では、対象のデバイスは PSoC 5LP デバイス (CY8CKIT-050) となる可能性があり、その場合、ピン接続は CY8CKIT-030 と同じです。

図 31. ホスト／ターゲットの接続



3.2 PSoC 3 のブートローディング

DVK が設定された後、用例のプロジェクトを以下のようにテストできます:

- CY8CKIT-050 でのボタン (P6[1]) を一度押すと、*Bootloadable1.cyacd* ファイルが対象の PSoC 3 デバイスにブートロードされます。正常終了の時、「Bootloaded - Hello」のメッセージが CY8CKIT-050 LCD で表示され、「Hello」のメッセージが CY8CKIT-030 LCD で表示されます。
- 引き続いてのブートローディング動作のために、CY8CKIT-030 でのボタン(P6[1]) を押します。そうすると、PSoC 3 デバイスはブートローダに入り、新しいアプリケーションをブートロードできる状態になります。LED が点滅し始めます。
- CY8CKIT-050 でのボタンを次に押すと、*Bootloadable_2.cyacd* ファイルが対象の PSoC 3 デバイスにブートロードされます。正常終了の時、「Bootloaded - Bye」のメッセージが CY8CKIT-050 LCD に表示され、「Bye」のメッセージが CY8CKIT-030 LCD に表示されます。

3.3 PSoC 4/PSoC アナログ コプロセッサをブートローディング

- UART_Bootloader_Host プロジェクトの *main.c* ファイルでは、PSoC 4 または PSoC アナログ コプロセッサのために TARGET_DEVICE マクロをそれぞれ PSoC_4 あるいは PSoC_AC に変更します。プロジェクトを再ビルドし、CY8CKIT-050 での PSoC 5LP デバイスにプログラムします。
- CY8CKIT-050 でのボタン (P6[1]) を一度押すと、*Bootloadable1.cyacd* ファイルは対象の PSoC 4/PSoC アナログ コプロセッサにブートロードされます。正常終了の時、PSoC 4/PSoC アナログ コプロセッサ キット上の緑色 LED が点滅し始めます。
- 次のブートローディング動作のために、Pin_Startbootloader ボタン(P6[1])を押します。そうすると、PSoC 4/PSoC アナログ コプロセッサはブートローダに入り、新しいアプリケーションをブートロードできる状態になります。PSoC 4/PSoC アナログ コプロセッサ キット上の赤色 LED が点滅を始めます。
- CY8CKIT-050 でのボタンを次に押すと、*Bootloadable2.cyacd* ファイルが対象の PSoC 4/PSoC アナログ コプロセッサにブートロードされます。正常終了の時、PSoC 4/PSoC アナログ コプロセッサ キット上の青色 LED が点滅し始めます。

4 まとめ

本アプリケーション ノートは UART を通信インターフェースとして使用し、PSoC 3、PSoC 4 および PSoC 5LP をブートロードする方法について説明した。また、ブートローダホストの基本的な構成ブロックも紹介し、組み込み UART ブートローダ ホストの構築方法を示しました。

ブートローダはフィールドでの更新を実施するための標準的な方法です。ユーザーのために全ての設定を行う PSoC Creator を用いることにより、PSoC 用のブートローダの作成が容易です。

詳細情報については、付録 A – メモリ節および「PSoC 3、PSoC 4 と PSoC 5LP テクニカル リファレンス マニュアル」をご参照ください。

5 関連アプリケーション ノート

ブートローダとフラッシュのプログラミングについて、よりよく理解していただくために、次のアプリケーションノートをご参照ください。

- AN73854 – PSoC 3, PSoC 4, and PSoC 5LP Introduction to Bootloaders
- AN60317 – PSoC 3 and PSoC 5LP I2C Bootloader
- AN73503 – USB HID Bootloader for PSoC 3 and PSoC 5LP
- AN84401 – PSoC 3 and PSoC 5LP SPI Bootloader
- AN86526 – PSoC 4 I2C Bootloader
- AN73054 – PSoC 3 and PSoC 5LP Programming Using an External Microcontroller (HSSP)
- AN61290 – PSoC 3 and PSoC 5LP Hardware Design Considerations

- AN54181 – Getting Started with PSoC 3
- AN79953 – Getting Started with PSoC 4
- AN77759 – Getting Started with PSoC 5LP
- AN211293 – Getting Started with PSoC Analog Coprocessor

PSoC の他の多くの特徴と機能に関わるアプリケーションノート全ての一覧は、ここをクリックしてください。

6 関連のプロジェクト

このアプリケーション ノートに添付されたプロジェクトは、表 7 の通りにまとめられています。

表 7. 本アプリケーション ノートに関連付けられているプロジェクト

設計プロジェクト名	説明
UART_Bootloader_Host	これは、UART を通信チャネルとして用い、PSoC 5LP が PSoC 3、PSoC 4、PSoC アナログ コプロセッサ または PSoC 5LP をブートロードすることを示す、組み込みブートローダ ホスト プロジェクトである。
UART_Bootloader	このブートローダ プロジェクトは UART を通信チャネルとする。ブートローダは LED を点滅させる
Bootloadable1	PSoC 3/PSoC 5LP に対し、このプロジェクトは「Hello」のメッセージを対象のデバイスのキャラクタ LCD に表示 PSoC 4/PSoC アナログ コプロセッサに対して、このプロジェクトは対象のキット上の緑色 LED を点滅させる
Bootloadable2	PSoC 3/PSoC 5LP に対し、このプロジェクは「Bye」のメッセージを対象のデバイスのキャラクタ LCD に表示 PSoC 4/PSoC アナログ コプロセッサに対して、このプロジェクトは対象のキット上の青色 LED を点滅させる

著者について

氏名:	Anu M D
役職:	シニア アプリケーション エンジニア
経歴:	Anu M D 氏は PSoC アプリケーションに特化したサイプレスセミコンダクタの プログラマブルシステム部のアプリケーションエンジニアです。
氏名:	Siddalinga Reddy
役職:	アプリケーション エンジニア
経歴:	Siddalinga Reddy 氏は PSoC のアプリケーションに特化したサイプレスセミ コンダクタのプログラマブル システム部のアプリケーション エンジニアです。

A 付録 A – メモリ

フラッシュメモリの詳細

フラッシュ メモリはファームウェア、バルク データ、ECC データ、デバイスの設定データ、工場出荷時の設定データおよびユーザー定義のフラッシュ プロテクション データの記憶領域を提供します。図 32 は PSoC 3、PSoC 4 および PSoC 5LP にあるフラッシュメモリの物理的な構造を示します。

PSoC のフラッシュは「アレイ」と呼ばれるブロックに分けられます。アレイはアレイ ID によって個別に識別されます。PSoC 3 および PSoC 5LP では、各アレイに 256 列のフラッシュ メモリがあります。各列には 256 データ バイト、且つ、有効な場合、32 ECC (エラー訂正コード) バイトがあります。32 ECC バイトは、エラー訂正データの代わりにコンフィギュレーション データを格納するために使用することができます。これにより、ひとつのアレイは、命令とデータの記憶用に 64KB あるいは 72KB 持つことができます。

PSoC 4 および PSoC アナログ コプロセッサには、各アレイには 128 列または 256 列のフラッシュメモリがあります。各列には 128 データ バイトがあります。これにより、ひとつのアレイは、命令とデータの記憶用に 16KB または 32KB を持つことができます。

フラッシュ アレイの数はデバイスと品番によって異なります。PSoC 3 および PSoC 4100S は、最大フラッシュ 64KB を有し、アレイはひとつだけであり、唯一の有効なアレイ ID は 0 です。PSoC 4100/4200、PSoC アナログ コプロセッサおよび PSoC 4000S デバイスは、最大フラッシュ 32KB を有し、アレイはひとつだけであり、唯一の有効なアレイ ID は 0 です。また、PSoC 5LP は最大フラッシュ 256KB、すなわち 4 つのフラッシュ アレイを有し、有効なアレイ ID は 0～3 です。

フラッシュ メモリは 1 度に 1 列がプログラムされます。フラッシュメモリは一度に 64 列セクタごとまたはフラッシュ全体を消去できます。列はアレイ ID と列番号の固有の組み合わせで識別されます。

図 32 は、フラッシュの最初の X 列はブートローダで占められていることを示します。X は、以下の項目に十分な空間があるように設定されます。:

- address 0 から始まるブートローダのベクタ テーブル (PSoC 4 および PSoC 5LP のみ)
- ブートローダ プロジェクトの設定バイト
- ブートローダ プロジェクトのコードとデータ
- フラッシュのブートローダ部のチェックサム

  PSoC 4 および PSoC 5LP の場合、ベクタ テーブルは、ブートローダ プロジェクトの初期スタック ポインタ (SP) 値、およびブートローダ プロジェクト コードの開始アドレスを含みます。また、ブートローダが使用する、例外および割り込み用のベクタも含みます。PSoC 3 で、割り込みベクタはフラッシュ内にありません。それらは割り込みコントローラにより提供されます。

  ブートローダブル プロジェクトは、ブートローダの次にある最初の 256 バイト境界から始まるフラッシュ領域を占有しています (PSoC 4 は最初の 128 バイト境界です)。このフラッシュ領域は以下の項目を含みます:

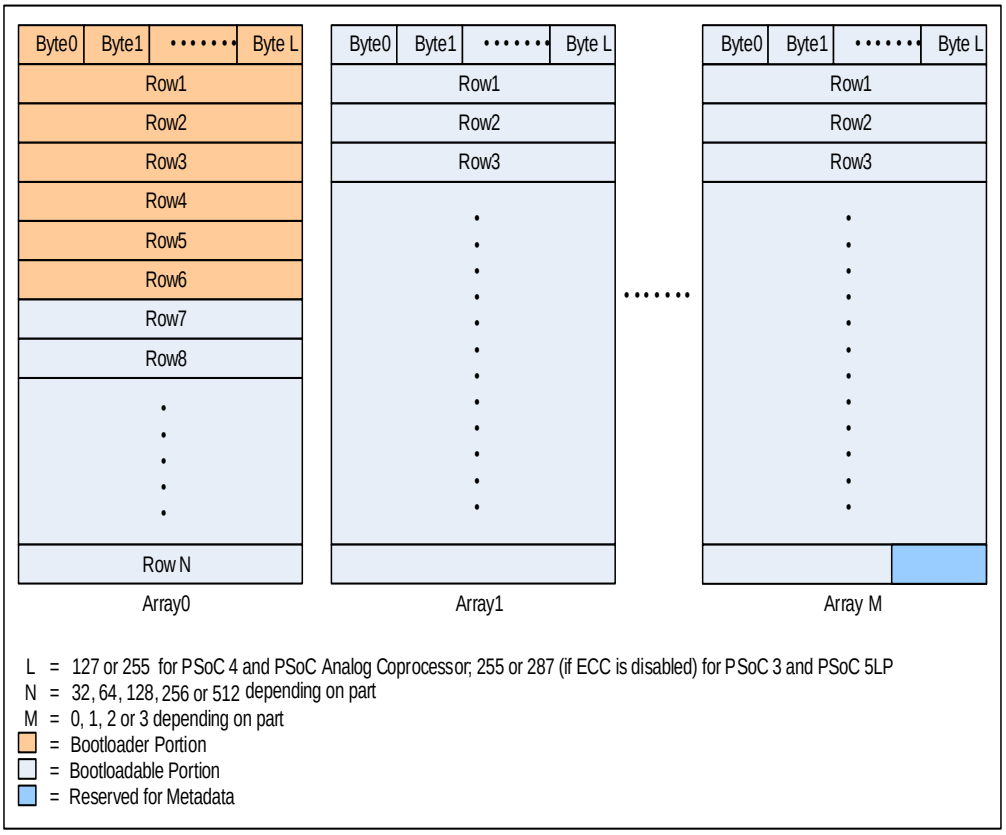
  - ブートローダブルプロジェクトのベクタテーブル (PSoC 4 および PSoC 5LP のみ)
  - ブートローダブル プロジェクトのコードとデータ

フラッシュ メモリの最上位の 64 バイト ブロックは、2 つのプロジェクトの共有領域として使用されます。このブロックに保存されるパラメーターは以下の通りです:

- ブートローダブル プロジェクトのフラッシュ メモリ内のエントリ (4 バイトのアドレス)
- ブートローダブル プロジェクトが占有するフラッシュ メモリ量 (フラッシュの列数)
- フラッシュのブートローダブルブロックのチェックサム (1 バイト)
- フラッシュ メモリのブートローダブル ブロックのバイト単位のサイズ (4 バイト)

フラッシュ メモリ内のメタデータの場所にある詳細情報は、フラッシュ メモリ内のメタデータの配置をご参照ください。

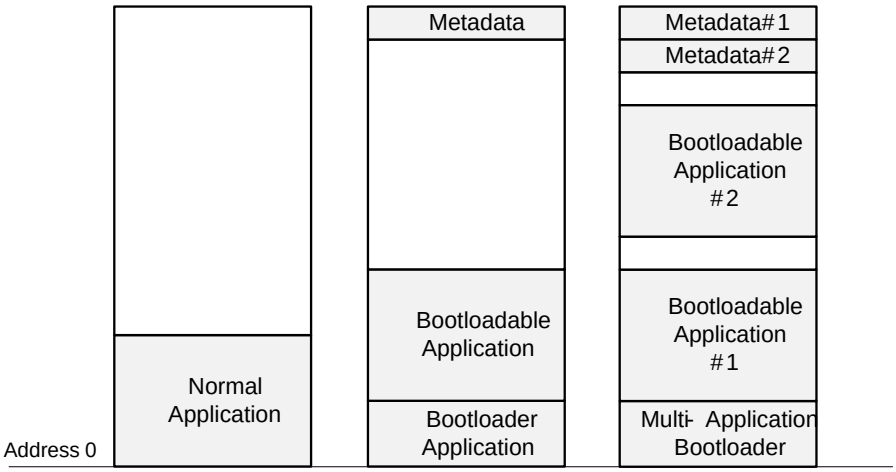
図 32. PSoC 内のフラッシュ メモリの物理的な構造



PSoC でのメモリ使用量

ブートローダ プロジェクトには 2 種類あります。それは標準とマルチアプリケーションです。マルチアプリケーション ブートローダは、実行可能な有効なアプリケーションが常に存在する保証が必要な設計に役立ちます。しかし、この保証には、各アプリケーションがフラッシュ メモリ領域を半分しか使用できない制限があります。図 33 は各々のタイプの PSoC Creator プロジェクトにおけるフラッシュ メモリ使用量を示します。

図 33.フラッシュメモリ使用量



フラッシュ メモリ内のメタデータの配置

メタデータセクションは、フラッシュの最上位 64 バイトのブロックで、図 33 に示すように、ブートローダとブートローダブル プロジェクトのために共通して使用されます。表 8 に示すように、使用されるデバイスに応じて、様々なパラメータがこのブロックに格納されます。マルチアプリケーション ブートローダには 2 セットのメタデータがあります。

表 8. メタデータの配置

アドレス	PSoC 3	PSoC 4/PSoC 5LP
0x00	アプリケーション チェックサム	アプリケーション チェックサム
0x01	予約	アプリケーション アドレス
0x02		
0x03	アプリケーション アドレス	
0x04		
0x05	該当なし	最後のブートローダ列
0x06	該当なし	
0x07	最後のブートローダ列	
0x08		
0x09		アプリケーションの長さ
0x0A		
0x0B	アプリケーションの長さ	
0x0C		
0x0D		該当なし
0x0E		該当なし
0x0F	該当なし	該当なし
0x10	アプリケーション アクティブ	アプリケーション アクティブ
0x11	アプリケーション ベリファイ	アプリケーション ベリファイ
0x12	ブートローダ アプリケーションのバージョン	ブートローダ アプリケーションのバージョン
0x13	ブートローダブル アプリケーションの ID	ブートローダブル アプリケーションの ID
0x14		
0x15	ブートローダブル アプリケーションのバージョン	ブートローダブル アプリケーションのバージョン
0x16		
0x17	ブートローダブル アプリケーションのカスタム ID	ブートローダブル アプリケーションのカスタム ID
0x18		
0x19～0x3F	該当なし	該当なし

注: マルチアプリケーションブートローダでは、メタデータ用の最後のブートローダ列 (イメージ 2) は、ブートローダ列ではなく、フラッシュ セクションにあるブートローダブル 1 の最後の列を意味します。

フラッシュの保護

ブートローダ コードが無効になると、製品が使用できなくなります。そのため、フラッシュのブートローダ部を不慮の上書きから保護することが重要です。

PSoC デバイスは柔軟なフラッシュ保護システムを備えています。この機能は、所有権者のコードの複製やリバース エンジニアリングを防止するように設計されています。しかし、これはフラッシュのブートローダ部への不注意による書き込みから守るためにも使用されます。

表 9 に示すように、フラッシュ メモリには 4 段階の保護レベルが提供されています。フラッシュ メモリの各列は異なる保護レベルを設定でき、PSoC Creator (*.cydwr* ファイルのフラッシュ セキュリティ タブ) を使用してセットすることができます。

表 9. フラッシュ メモリ保護レベル

保護レベル	許可	不可
非保護	▪ 外部読み出しおよび書き込み ▪ 内部読み出しおよび書き込み	–
工場アップグレード	▪ 外部書き込み ▪ 内部読み出しおよび書き込み	外部読み出し
現場アップグレード	内部読み出しおよび書き込み	外部読み出しおよび書き込み
完全保護	内部読み出し	▪ 外部読み出しおよび書き込み ▪ 内部書き込み

注: PSoC 4 は、非保護および完全保護という 2 つのフラッシュの保護レベルのみをサポートします。

フラッシュ メモリのブートローダ部が「完全保護」の保護レベルに設定されると、その設定を現場で変更できません。保護レベルまたはブートローダ コードを変更する方法は、フラッシュ メモリを全て消去することと、JTAG/SWD インターフェースを使って再プログラムすることのみです。

以下はブートローダのフラッシュ メモリ保護の例です。

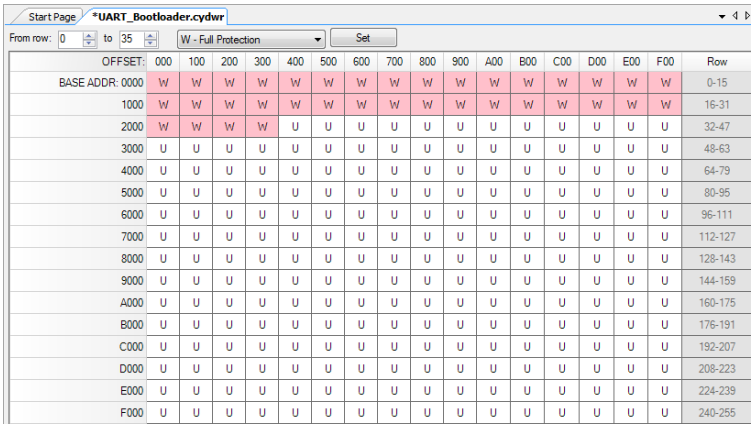
フラッシュ保護の例

ブートローダ プロジェクトが構築されると、PSoC Creator **出力**ウインドウが使用したフラッシュ メモリ量を示します。例えば、UART_Bootloader プロジェクトが占めるフラッシュ メモリが 9250 バイトの場合、(64KB フラッシュの PSoC 3 の場合) 出力は以下の通りです:

```
フラッシュ メモリの使用量: 9250／65536 バイト (14.11%)。
```

この例では、ブートローダはフラッシュ メモリの 37 列 (9250/256) を占めます。すなわち、フラッシュ メモリ内の場所は、0x0000～0x2300 です。(PSoC Creator にある.*cydwr* ファイルの **Flash Security** タブで) これらの列のフラッシュ メモリ保護レベルを「完全保護」に設定します。残りの列の保護レベルは図 34 に示すように非保護 (デフォルト) またはフィールド アップグレードにできます。フラッシュ メモリ保護ダイアログの使用方法の詳細については、PSoC Creator のヘルプ記事「Flash Security Editor」をご参照ください。

図 34. PSoC Creator 内のフラッシュ保護



不揮発性ラッチ設定

不揮発性ラッチ (NVL) 設定は、ブートローダ プロジェクト、またはその他の標準 PSoC Creator プロジェクトに適用できますが、ブートローダブル プロジェクトには適用できません。これは、NVL の設定が、デバイスの起動時に常にロードされるためです。デバイスの起動時に、ブートローダ プロジェクトが最初に実行し、ブートローダブル コードが続きます。そのため、ブートローダブル プロジェクトの NVL 設定は、それが関連付けられているブートローダ プロジェクトの設定です。

いくつかの PSoC Creator デザイン ワイド リソース (*.cydwr*) 設定はユーザーNVL を使用してプログラムされます。ブートローダブル プロジェクト用のいくつかの *.cydwr* 設定がブートローダ プロジェクトの設定と異なる場合には、警告またはエラーメッセージが表示されます。

B 付録 B – プロジェクト ファイル

ブートローダブル出力ファイル

PSoC Creator プロジェクトが構築されると、*.hex* タイプの出力ファイルが生成されます。これは、JTAG/SWD インターフェースを使ったプログラミング時に PSoC デバイスへダウンロードされるファイルです。

ブートローダブル プロジェクトの場合、*.hex* ファイルはブートローダブル プロジェクトと関連するブートローダ プロジェクトが結合された*.hex* ファイルです。このファイルは通常、プロダクション環境で両方のプロジェクトを JTAG/SWD 経由でダウンロードするために使用されます。

*.cyacd のファイル フォーマット

ブートローダブル プロジェクトが構築されると、*.cyacd* タイプ (アプリケーション コードとデータ) の追加ファイルも生成されます。このファイルには、ヘッダがあり、その後にフラッシュ メモリのデータ行が続きます。ヘッダを除くと、.cyacd ファイルの各行は、それぞれフラッシュ データの 1 列全体を示します。データは、ビッグ エンディアン形式の ASCII データとして保存されます。従って、ブートロードの時、このファイルの内容は構文解析される (ASCII から hex に変換される) 必要があります。構文解析は*.hex* タイプのファイルをプログラムする時には必要ありません。

このファイルのヘッダは次のフォーマットになっています:

```
[4 bytes Silicon ID] [1 byte Silicon rev] [1 byte checksum type]
```

フラッシュ メモリの列は次のフォーマットになっています:

```
[1 byte array ID] [2 bytes row number] [2 bytes data length] [N bytes of data] [1 byte checksum]
```

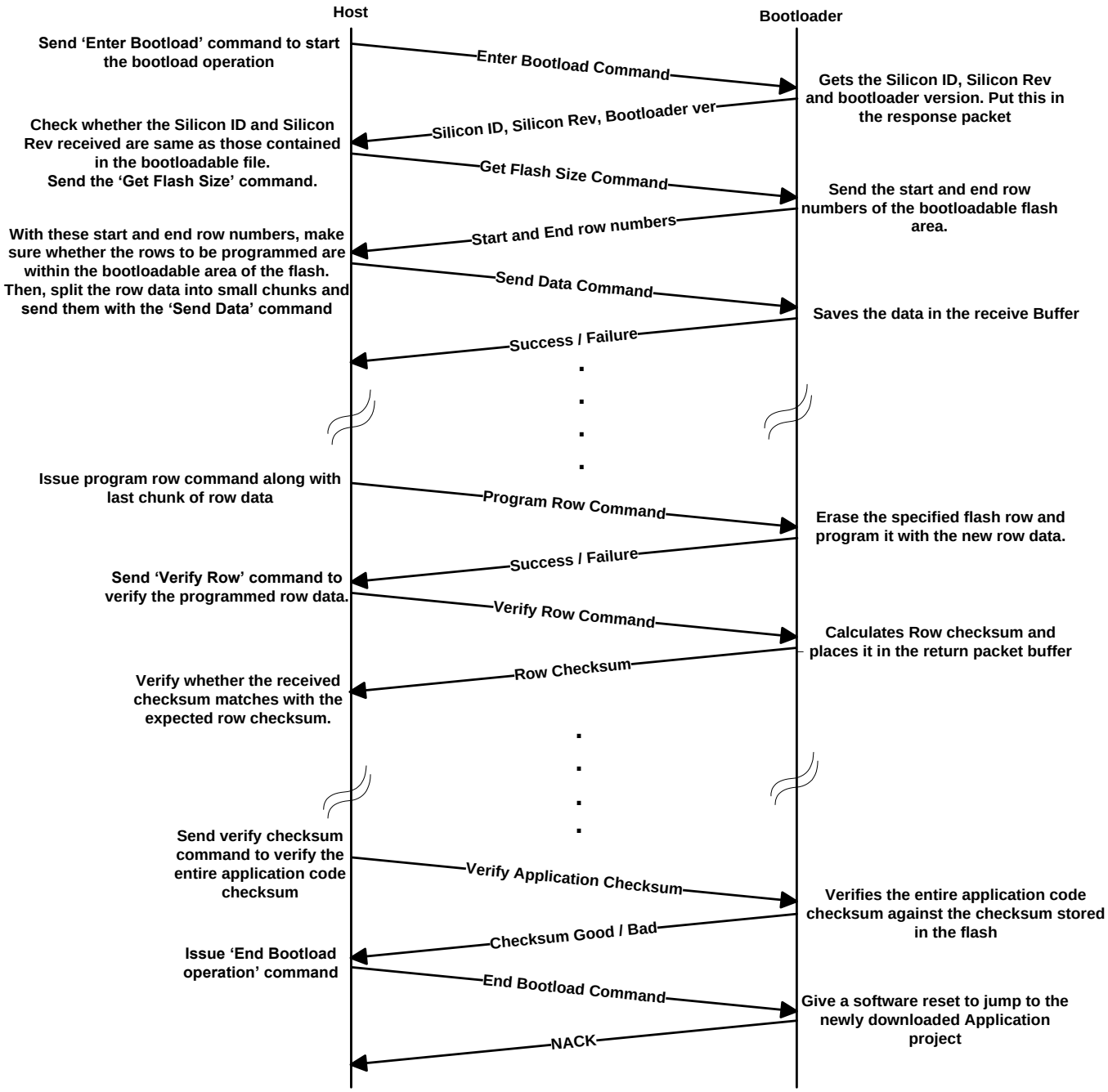
ヘッダ内のチェックサムタイプは、ブートローディング動作中にブートローダのホストとブートローダとの間で送信されるパケットに使用されるチェックサムのタイプを示します。このバイトが 0 の場合、チェックサムは単純な合計値です。このバイトが 1 の場合、チェックサムは CRC-16 です。

C 付録 C – ホスト／ターゲットの通信

通信フロー

ブートローダ ファンクション フロー節は PSoC 内のブートローダ動作を説明し、「組み込みホストによるブートローディング」節はブートローダ ホストのビルディング ブロックを紹介します。これを背景にして、図 35 はブートローディング中のホストとターゲット間の通信フローについて説明します。これは、コマンドがターゲットに発行され、応答が受信される順序を示します。ブートロードコマンドの完全なリスト、そのコードと期待される応答については、コマンドとステータス／エラーコードをご参照ください。

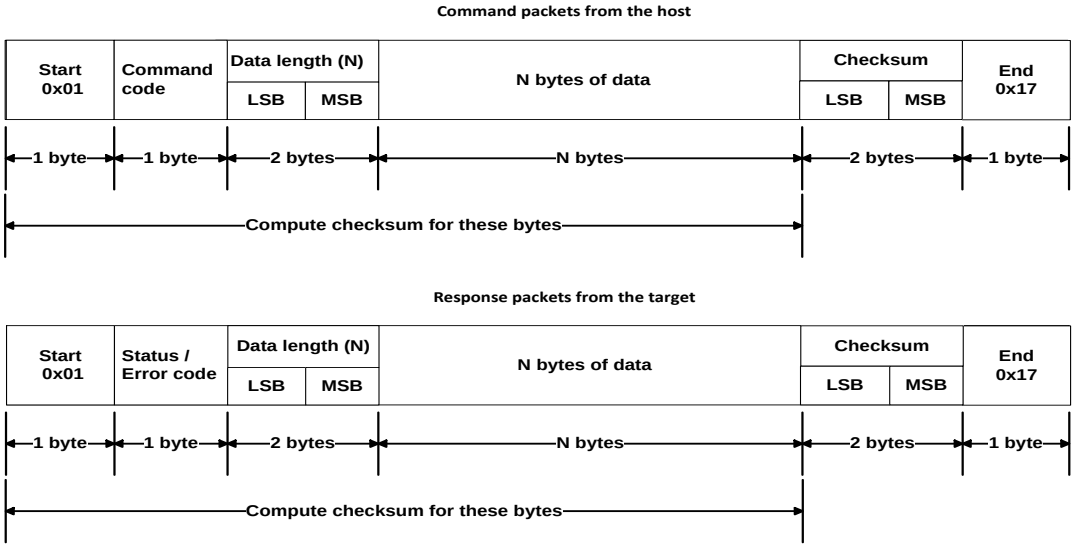
図 35. ブートローディング中の通信フロー



プロトコル パケットのフォーマット

ブートローディング動作はホストとターゲット間のコマンドと応答パケットの交換を含みます。これらのパケットは図 36 に示すように、特定のフォーマットを持っています。

図 36. ブートローディング パケットのフォーマット



それぞれのパケットはチェックサムバイトを含みます。チェックサムは、ブートローダ プロジェクトの設定によって、単純合計 (2 の補数) または CRC-16 のいずれかにできます。データ長とチェックサムなどのマルチバイト データを送信する時、最下位バイトが先に送信されます。

ブートローダは応答パケットでホストからのコマンドに応答します。応答パケットのフォーマットは、コマンドコードの代わりにステータス／エラーコードがあることを除けば、コマンドパケットのフォーマットと似ています。重要なコマンドとデータバイトおよびブートローダ応答パケットデータは表 10 に示されています。

コマンドとステータス／エラーコード

前の節で説明した通り、コマンドと応答パケットの構造は同様です。唯一の違いは、第 2 バイトがコマンド コードまたはステータス／エラー コードを含むことです。

表 10 は、コマンドのリストとそれらの予期される応答の一覧です。表 11 はステータスとエラーコードの一覧です。

表 10. ブートローディング コマンド

コマンド バイト	コマンド	コマンド パケット内の データ バイト	予期される応答データ バイト
0x31	Verify Checksum (チェックサムの検証)	該当なし	1 バイト: 「0」またはゼロ以外。 ゼロ以外のバイトの場合、アプリケーション チェックサムが一致し、それは有効なアプリケーション ゼロのバイトの場合、チェックサムが間違い、アプリケーションは無効
0x32	フラッシュサイズを取得	フラッシュアレイ ID、 1 バイト	ブートローダブルフラッシュ領域の最初の列番号、2 バイト、 ブートローダブル フラッシュ領域の最終の列番号、2 バイト これらの番号は要求されたアレイ ID 用のものである
0x33	アプリケーションステータスを取得 (マルチアプリケーション ブートローダのみに有効)	アプリケーションの番号、 1 バイト	有効なアプリケーションの番号、1 バイト、 アクティブアプリケーションの番号、1 バイト。 指定されたアプリケーションが有効でアクティブかどうかチェックする

コマンド バイト	コマンド	コマンド パケット内の データ バイト	予期される応答データ バイト
0x34	列を消去	フラッシュアレイ ID、1 バイト、 フラッシュ列の番号、2 バイト	該当なし 指定されたフラッシュ メモリ列の内容を消去
0x35	ブートローダの同期 (Sync bootloader)	該当なし	該当なし ブートローダをクリーンな状態にリセットします。バッファ内のいかなるデータも廃棄される。このコマンドは、ブートローダとホストが互いに同期しなくなる場合にのみ必要である
0x36	Set Active Application (アクティブ アプリケーションを設定) (マルチ アプリケーション ブートローダにのみ有効)	アプリケーションの番号、 1 バイト	該当なし 指定されたアプリケーションをアクティブとして設定
0x37	Send Data (データの送信)	送信される N バイト データ	該当なし 受信されたデータ バイトは、プログラム列コマンドを見越して、ブートローダによってバッファされる
0x38	ブートローダに入る	該当なし	シリコン ID、4 バイト、 シリコン リビジョン、1 バイト、 ブートローダ バージョン、3 バイト 全てのコマンドは、このコマンドが受信されるまで無視される
0x39	Program Row (列をプログラム)	フラッシュアレイ ID、1 バイト、 フラッシュ列の番号、2 バイト、 送信される N バイト データ	該当なし 複数バイトのデータをデータ送信コマンドを使用してブートローダに送信した後、データの最終チャンク (塊) がこのコマンドと共に送信される
0x3A	Verify Row (列の検証)	フラッシュアレイ ID、1 バイト、 フラッシュ列の番号、2 バイト	列のチェックサム、1 バイト 指定された列のチェックサムを返す
0x3B	ブートローダを終了する	該当なし	該当なし このコマンドは認識されない

表 11. ブートローディング ステータス／エラーコード – コマンドへの可能な応答

ステータス／エラーコード	ラベル	説明
0x00	CYRET_SUCCESS	コマンドは正しく受信され、実行された
0x02	BOOTLOADER_ERR_VERIFY	フラッシュのベリファイに失敗した
0x03	BOOTLOADER_ERR_LENGTH	利用可能なデータ量は予想される範囲外
0x04	BOOTLOADER_ERR_DATA	データが正しい形式ではない
0x05	BOOTLOADER_ERR_CMD	コマンドが認識されない
0x06	BOOTLOADER_ERR_DEVICE	期待されるデバイスと検出したデバイスが一致しない
0x07	BOOTLOADER_ERR_VERSION	検出されたブートローダのバージョンはサポートされていない
0x08	BOOTLOADER_ERR_CHECKSUM	チェックサムは想定した値と異なる
0x09	BOOTLOADER_ERR_ARRAY	フラッシュ アレイ ID が有効でない
0x0A	BOOTLOADER_ERR_ROW	フラッシュ行番号が有効でない
0x0C	BOOTLOADER_ERR_APP	アプリケーションは有効ではなく、アクティブに設定できない
0x0D	BOOTLOADER_ERR_ACTIVE	アプリケーションは現在アクティブとしてマークされていない
0x0F	BOOTLOADER_ERR_UNK	未知のエラーが発生した

D 付録 D – ホスト コア API

cybtldr_api2.c /.h

これは、ブートロード動作の全てを処理する、より高レベルの API です。ファイルを開く関数とファイルを閉じる関数があります。これはブートロード動作のために、*cybtldr_api.c / .h* API の関数を呼び出します。この API は GUI ベースのブートローダ ホストを構築する時に使用されます。

cybtldr_parse.c /.h

このモジュールは、デバイスに送信するブートローダブルのイメージを含む*.cyacd* ファイルの構文解析を行います。ファイルへのアクセス設定、ヘッダの読み出し、列のデータの読み出し、およびファイルを閉じるための関数も持っています。

cybtldr_api.c /.h

これは 1 回に 1 つのデータ列をブートローダのターゲットに送信する低レベル API ファイルです。ブートロード動作の設定、行の消去、行のプログラミング、行の検証、およびブートロード動作の終了を行う関数が含まれています。表 12 は、この API ファイルの関数の詳細を説明します。

表 12. cybtldr_api.c /.h の関数

関数	説明
CyBtldr_StartBootloadOperation	▪ この関数は通信インターフェースを有効にし、Enter Bootloader コマンドをターゲットに送信する。 ▪ 受信された応答パケットから、ターゲット デバイスのシリコン ID、シリコン リビジョン、およびブートローダのバージョンを確認する
CyBtldr_ProgramRow	▪ この関数は最初に列を確認する。つまり、Get Flash Size コマンドをターゲットフラシュを特定するアレイ ID のために、ターゲットに送信する。それに対応して、ターゲットはそのアレイにあるブートローダブル フラッシュ部の先頭と末尾の列番号を返す。ホストはこの応答を読み込み、指定した列がフラッシュのブートローダブルの領域にあるかどうか確認する ▪ 列の確認が成功したら、ホストは列データをより細かい部分に分割して、Send Data コマンドを使用してそれらをターゲットに送信する ▪ 列データの最終部と共に、この関数は Program Row コマンドをターゲットに送信する
CyBtldr_VerifyRow	▪ この関数は最初に特定アレイ ID と列番号のために列を確認する ▪ 列の確認が成功したら、確認されたフラッシュ列に対する Verify Row コマンドを送信する。それに応答して、ターゲットは列のチェックサムを返す ▪ 返されたチェックサムは期待されるチェックサム値に対して検証される
CyBtldr_EraseRow	▪ この関数は最初に特定アレイ ID と列番号のために列を確認する ▪ 列の確認が成功したら、確認されたフラッシュ列に対する Erase Row コマンドを送信する
CyBtldr_EndBootloadOperation	この関数は Exit Bootload コマンドを送信し、通信インターフェースを無効化する

cybtldr_command.c /.h

この API は、ターゲットへのコマンドパケットの構成を処理し、ターゲットから受信した応答パケットを解析します。cybtldr_api.c /.h API は、この API の関数を呼び出します。例えば、Enter Bootload コマンドを送信するために、`CyBtldr_StartBootloadOperation()`は、この API の `CyBtldr_CreateEnterBootloadCmd()`関数を呼び出します。また、ターゲットに送信する前に、コマンド パケットのチェックサムを計算する機能もあります。

E 付録 E – ブートローダおよびデバイス リセット

前述の通り、制御をブートローダからブートローダブルに (またはその逆に) 遷移することは常にデバイス リセットで行われます。あるプログラムから別のプログラムに変更しながら、システムが極めて重要な機能を実行しなければならない場合には、このことを考慮しなければなりません。この節では、リセットを使用しなければならない理由、およびアプリケーションにおけるリセットのデバイス性能に対する影響を詳細に説明します。

なぜデバイス リセットが必要なのか?

デバイス リセットが必要とされる理由を理解するためには、システムのブートローダ プロジェクトおよびブートローダブル プロジェクトのいずれもが完全な自己完結型 PSoC Creator プロジェクトであることに注意することが重要です。各プロジェクトは固有のデバイス コンフィギュレーション設定を有します。そのため、あるプロジェクトから他のプロジェクトに変更する際には、PSoC デバイスのハードウェア機能を全て再定義することができます。

複雑なカスタム機能を実装するために、デバイス コンフィギュレーションで数千の PSoC レジスタ設定が行われます。これは特に、PSoC デジタルとアナログ ルーティング機能に当てはまります。レジスタとルーティングを設定する時、新しいコンフィギュレーション用ビットの設定に加えて、古いコンフィギュレーションのビットをリセットすることを確認してください。さもなければ、新しいコンフィギュレーションは動作せず、デバイスを損傷する可能性さえあります。

そのため、ブートローダとブートローダブル プロジェクトの間の切り替えには、デバイス ソフトウェア リセット (SRES) を行ってください。これによって、すべての PSoC レジスタはデフォルト状態にリセットされます。そして、新しいプロジェクトのコンフィギュレーションを開始することができます。すべての PSoC レジスタがそのデバイスのリセットのデフォルト状態に初期化されると想定すると、コンフィギュレーション時間とフラッシュ メモリ使用量の両方を削減できます。

デバイス I/O ピンへの影響

アプリケーション ノート AN61290 – PSoC 3, PSoC 5LP Hardware Design Considerations および AN60616 – PSoC Startup Process で説明した通りに、リセットおよび起動プロセス時に PSoC I/O ピンは表 13 に示すように、3 種類の別個の駆動モードにあります。

表 13. デバイス リセット中の PSoC I/O ピンの駆動モード

起動イベント	I/O ピンの駆動モード	期間 (Typ)		備考
		低速 IMO (12MHz)	高速 IMO (48MHz)	
デバイス リセット (SRES) がアクティブ デバイス リセットが解除	HI-Z アナログ	40µs		リセットがアクティブの時、I/O が HI-Z アナログ モードに維持される
NVL が I/O ポートにコピーされる コードの実行開始	NVL 設定: HI-Z アナログ、 プルアップまたはプルダウン	～12ms	～4ms	期間はコード実行速度およびコンフィギュレーションの複雑度に左右される
I/O ポートおよびピンが設定される	PSoC Creator プロジェクト コンフィギュレーション	該当なし		8 個の駆動モード。詳細はデバイス データシートをご参照ください
コードは main()に到達	コードは I/O ピン 機能を変更可能	該当なし		

PSoC での NVL 使用の詳細については、デバイス データシートをご参照ください。PSoC Creator プロジェクトでは、NVL は 2 箇所で設定されます:

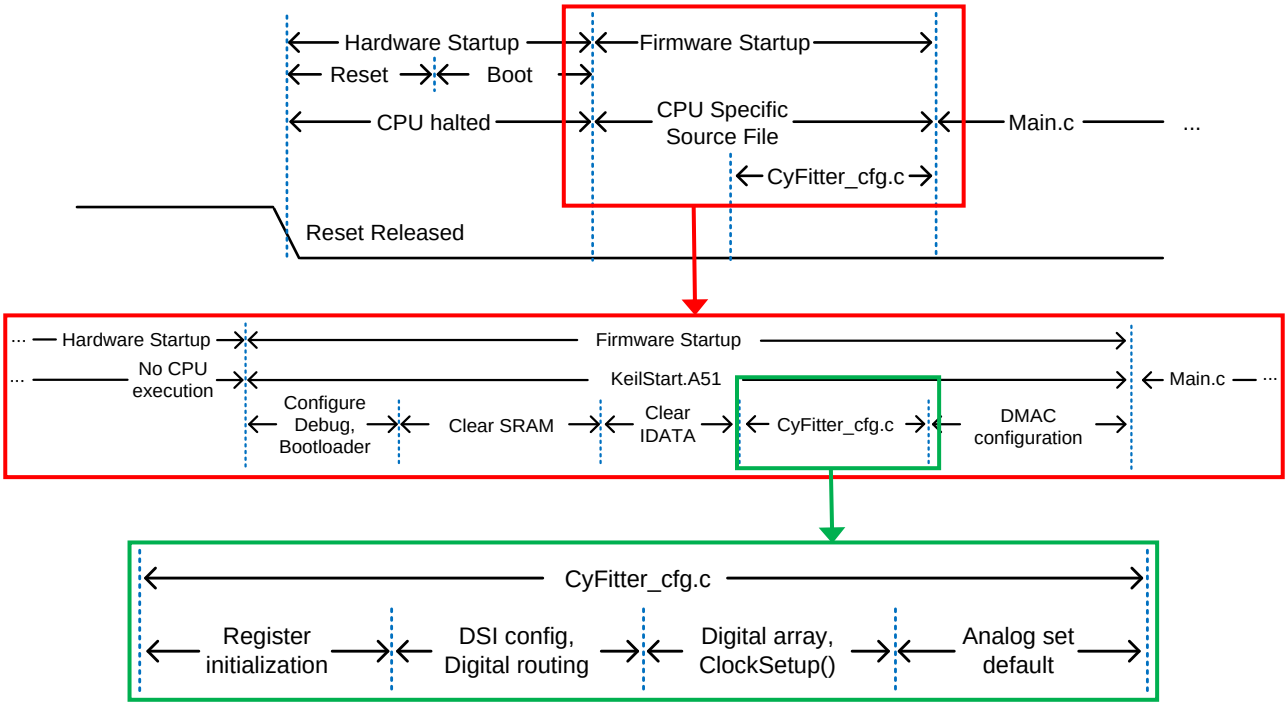
- I/O ポートには **Reset** タブ、個別のピン コンポーネントのコンフィギュレーション
- 他のすべての NVL には **System** タブ、Design-Wide Resources (DWR) ウィンドウ

デバイスがユーザーのプロジェクトにプログラムされる時、NVL が更新されます。ブートローダブル プロジェクトで NVL をセットすることができない点に注意してください。その DWR 設定は関連するブートローダ プロジェクトのものと一致する必要があります。

最終 I/O 駆動モードは個別のピン コンポーネントのコンフィギュレーションでセットされます。

図 37 は、デバイスの起動およびコンフィギュレーションのためのタイミング図を示します。図の中央にある例は PSoC 3 用のものです。PSoC 4 および PSoC 5LP には同様のプロセスが適用されます。詳細については AN60616 – PSoC Startup Process をご参照ください。

図 37. デバイス起動プロセス図



他機能への影響

デバイス リセット時に、汎用デジタル ブロック (UDB) レジスタはリセットされるため、全ての UDB ベース コンポーネントは消滅し、その機能は停止されます。同様なことは、コンフィギュレーション可能な SC/CT ブロックに基づいたアナログ コンポーネントおよび汎用アナログ ブロック (UAB) ベースのコンポーネントにも当てはまります。

デジタルおよびアナログの両方のすべての固定ペリフェラルはアイドル状態にリセットされます。これは DMA、DFB、タイマー (TCPWM)、I^2C、USB、CAN、ADC、DAC、コンパレータ、およびオペアンプを含みます。 内部主発振器 (IMO) を除いてすべてのクロックは停止されます。

すべてのデジタルおよびアナログ ルーティング制御レジスタがリセットされます。これは、すべてのデジタルとアナログのスイッチをオープンし、デバイス内のすべての接続を切断します。これは NVL 以外のすべての I/O との接続を含みます。

すべてのハードウェア ベースの機能はコンフィギュレーション後に復元されます (図 37 をご参照ください)。プロジェクトの main()関数が実行を開始する時、すべてのファームウェア機能は復元されます。

例: ファン制御

本節では、ブートローダおよび関連するデバイス リセットをファン制御などの代表的な用途内に統合する方法を検証します。PSoC Creator は、PWM、タコメーター入力キャプチャ タイマー、制御レジスタ、ステータス レジスタ、DMA チャネルまたは割り込みなどの必要なハードウェア ブロックをすべて含む、ファン コントローラー コンポーネントを提供します。詳細については、ファン コントローラー アプリケーションのページをご参照ください。

ファン制御アプリケーションはブートローダブル プロジェクトにあります。オプションで、ブートローダはブートロード中にファンの動作を維持するためにカスタマイズされることができます。

表 14 に示すように、ファンはブートローダとブートローダブル間移行中のデバイスがリセットされている間も、動作し続けることができます。

表 14. ファン コントローラー用のデバイス リセット中の PSoC I/O ピンの駆動モード

I/O ピンの駆動モード	説明
HI-Z アナログ	オプションで、100%デューティ比のために、PWM ピンに外部プルアップまたはプルダウン抵抗を追加する。ファンは慣性によって回り続ける可能性があるため、これは不要とされることもある
NVL 設定: HI-Z アナログ、プルアップまたはプルダウン	オプションで、100%デューティ比のために、PWM ピン コンポーネントのリセット値をプルアップまたはプルダウンに設定する。ファンは慣性によって回り続ける可能性があるため、これは不要とされることもある。
PSoC Creator プロジェクト コンフィギュレーション	100%デューティ比のために、PWM ピン コンポーネントの駆動モードおよび初期状態を設定する。PWM コンポーネントはアクティブになるが、動作しない
Main()の実行開始	PWM_Start() を呼び出す時、PWM はコンポーネントのデフォルトのデューティ比で PWM ピンの駆動を開始する。ファームウェアはタコメーターのデータを読み出し、デューティ比の制御を能動的に開始する

F 付録 F – その他のトピック

ブートローダ対 HSSP

ブートローダは、通信インターフェースを介してシステムファームウェアのアップグレードを可能にします。しかし、ブートローダのフラッシュ領域を含む完全なフラッシュのアップグレードのためには、JTAG/SWD プログラマ (HSSP) を使用する必要があります。HSSP を作成するための ISSP 仕様は AN62391 (PSoC 3) に記載されています。

ブートロード動作中に電源が切れたら、どうなるのか？

ブートロード処理中に電源が切れたら、次のリセットで、ブートローダブル プロジェクトのチェックサムが期待値 (フラッシュの最終列に格納されるブートローダブル プロジェクトのチェックサム) と一致せず、そのブートローダブル プロジェクトが無効であると見なされます。ブートロードが成功するまでプログラムの実行はブートローダ内に留まります。ブートロー ダ ホストは、ブートロード動作を再開するために start bootload コマンドを送信する必要があります。

なぜ、ブートローダとブートローダブル プロジェクトの間の切り替えにリセットが必要なのか？

PSoC は非常にコンフィギュアラブルなデバイスです。ブートローダは、オンチップ ハードウェア リソースおよびファームウエアの変更を可能にします。その高度なコンフィギュアラブルなアーキテクチャにより、ハードウェアの再設定 (配置、配線、ファンクション) はリセット状態からのみ可能です。従って、ブートローダはブートローダとブートローダブル プロジェクト間をジャンプするためにはリセットが必要になります。付録 E – ブートローダおよびデバイス リセットをご参照ください。

通常アプリケーション プロジェクトのブートローダブル プロジェクトへの変換

標準 (通常) のアプリケーション プロジェクトを既に作成し、これをブートローダブル プロジェクトに変換したい場合は、14 ページの図 14 に示すように、トップ デザインにブートローダブル コンポーネントを追加し、ブートローダ プロジェクトの.*hex* ファイルを付属するものとして追加してください。

あるプロジェクトが通常のプロジェクトとして作成された後、ブートローダ コンポーネントをトップ デザインに追加することでブートローダ プロジェクトに変更される場合、ブートローダ プロジェクトが期待されるように動作するためには、`Bootloader_Start()`呼び出し関数を *main.c* に挿入する必要があります。

注: PSoC Creator 3.1 については、標準 (通常) のプロジェクトをブートローダブル／ブートローダ プロジェクトに変換したい場合、プロジェクトのアプリケーション タイプを「Bootloadable/Bootloader」に変更してください。これを行うには、プロジェクトを右クリックして、**Build** > **Code Generation** > **General** を選択します。そして、アプリケーション タイプを変更し、ブートローダブル／ブートローダー コンポーネントをトップデザインに追加します。

ブートローダブル プロジェクトのデバッグ

PSoC Creator ブートローダ システムでは、最初にブートローダ プロジェクトが (デバイス リセットで) 実行し、続いてブートローダブル プロジェクトが実行します。ブートローダからブートローダブル プロジェクトへのジャンプはソフトウェアが制御するデバイス リセットによって実施されます。これにより、デバッガ インターフェースをリセットします。つまり、ブートローダブル プロジェクトはデバッガ モードでは実行できません。

ブートローダブル プロジェクトをデバッグするには、「Application Type」を「Normal」に変更して、デバッグを実行します。デバッグが完了した後、それを「Bootloadable」に戻します。

他のオプションは、ブートローダブル プロジェクトの.*hex* ファイルをデバイスにプログラムすることです。そのあと、ブートローダブルプロジェクトが実行中に、「Attach to running target」オプションを使用してデバッグを実行します。この場合には、デバッガがデバイスに取り付けられた時点からのみ、ブートローダブル プロジェクトのデバッグを実行できます。

マルチアプリケーション ブートローダ

マルチアプリケーション ブートローダ (MABL) が 2 つのブートローダブルのアプリケーションを同時にフラッシュに配置するために使用されます。2 つのアプリケーションは、デバイスのフラッシュに常時有効なひとつのアプリケーションが存在することを確認するために、同じものにすることができます。または、2 つのアプリケーションは異なるものであることも可能で、ブートローダ コマンドを使用して切り替えることができます。この機能は各アプリケーションがフラッシュメモリの半分しか使用できないとのわかる通りの制限付きです。30 ページの図 33 は、フラッシュ メモリ内の 2 つのアプリケーションの配置を示します。

MABL は、標準ブートローダのアプリケーションとは異なる次のステップに従って実装できます:

1. MABL ブートローダ プロジェクトを新規作成します。ブートローダ コンフィギュレーション ウィンドウで「Multi-App Bootloader」チェックボックスを選択します

注: PSoC Creator 3.1 の場合、アプリケーション タイプを「Multi-App Bootloader」に設定します。

2. 例えば Project_A と Project_B という 2 つのブートローダブルプロジェクトをワークスペースに追加します。各プロジェクトでは、MABL プロジェクトに付属物を追加します。2 つの*.cyacd* ファイルが各プロジェクトのために生成されます。ひとつはフラッシュの下位領域用のもので、もうひとつはフラッシュの上位領域用のものです:
   - *Project_A_1.cyacd* と *Project_A_2.cyacd*
   - *Project_B_1.cyacd* と *Project_B_2.cyacd*
3. 1 のサフィックスを付ける*.cyacd* ファイルは常にフラッシュの前半分を、2 のサフィックスを付ける*.cyacd* ファイルはフラッシュの後半分を占めます。このようにして、*.cyacd* ファイルの特定の組み合わせのみが使用されることができます。これらの組み合わせは以下の通りです:
   - *Project_A_1.cyacd* と *Project_A_2.cyacd*
   - *Project_B_1.cyacd* と *Project_B_2.cyacd*
   - *Project_A_1.cyacd* と *Project_B_2.cyacd*
   - *Project_B_1.cyacd* と *Project_A_2.cyacd*
4. デバイスをマルチアプリケーション ブートローダ プロジェクトでプログラムし、上記のいずれかの 1 つの組み合わせにより、アプリケーション (*.cyacd* ファイル) を順次ブートロードします。
5. アプリケーション間の切り替えのために、アクティブ アプリケーション設定のコマンドをブートローダーに送信します。`CyBtldr_CreateSetActiveAppCmd()` の API 関数を使用してこのコマンドを作成できます。アクティブ アプリケーション設定のコマンドを送信する前に、ブートローダ モード入場のコマンドを送信します。すべてのコマンドを送信した後、ブートローダ モード退出のコマンドを送信します。これらの API の詳細情報については、*CyBtldr_Command.c / .h* ファイルをご参照ください。

ブートローダ用に必要なメモリ量

すべてのオプション コマンドを含む標準 UART ブートローダ プロジェクトは、Keil 8051 コンパイラ最適化レベル 5 を含む、約 7KB の PSoC 3 フラッシュを占めます。

GCC コンパイラの最適化が「Size」 (サイズ) に設定された PSoC 4 フラッシュの約 4.6KB を占めます。そして GCC コンパイラの最適地が「Size」に設定された PSoC 5LP フラッシュの約 5.4KB を占めます。 ユーザーがプロジェクトをビルドする時、ブートローダ プロジェクトにより使用されるメモリを出力ウインドウに見ることができます。ブートローダ プロジェクトで使用される RAM メモリは、ブートローダブル プロジェクトによって再使用されることができます。

図 38 に示すように、ブートローダ プロジェクトのメモリ使用量は、ブートローダ コンポーネントがサポートするオプションのコマンドを取り除くことにより、少し削減できます。

図 39 に示すように、*.cydwr* **System** タブで **Device Configuration Mode** を **Compressed** に設定してフラッシュ メモリの使用量を最小限に抑えます。起動時間がコード サイズより重要である場合、**Device Configuration Mode** を **DMA** に設定します。

図 38. ブートローダ コンポーネントでのオプション コマンドのチェック解除

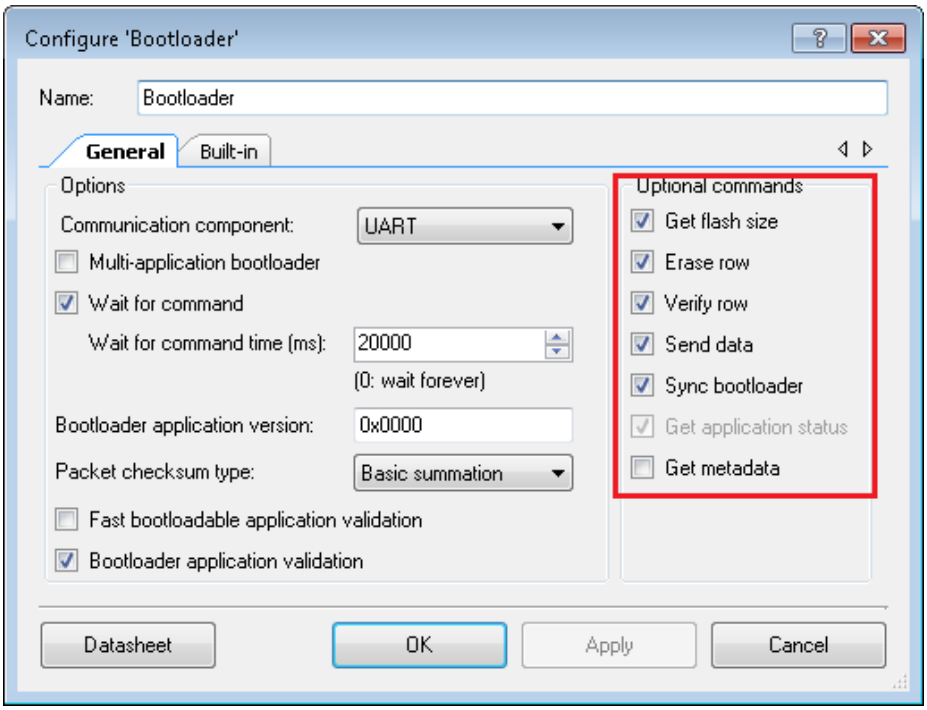
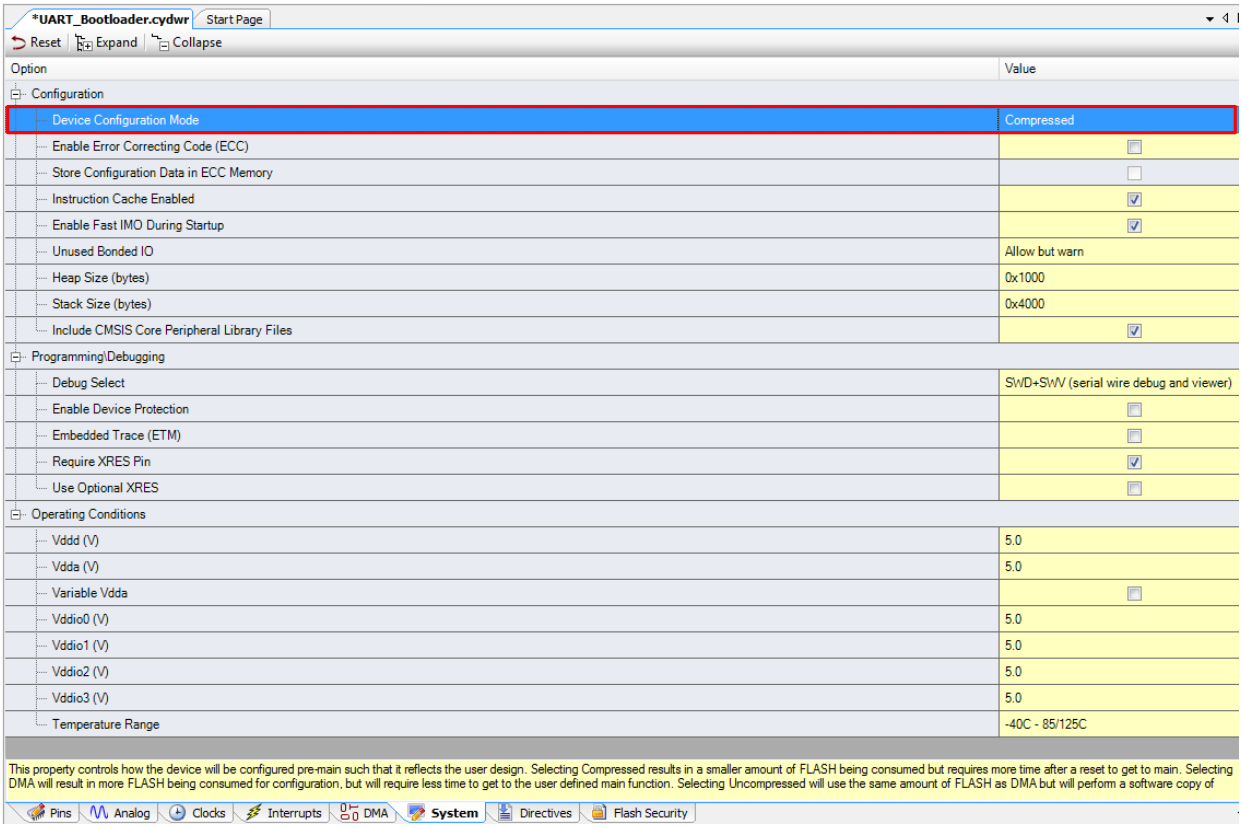


図 39. デバイス コンフィギュレーション モード



G 付録 G – C#ブートローダ ホスト アプリケーション

ブートローダ ホスト アプリケーション用に GUI を開発するために、以下のソフトウェアをインストールする必要があります:

- Visual C# 2010 Express Edition および Visual C++ 2010 Express Edition
- Visual Studio 2010 Complete

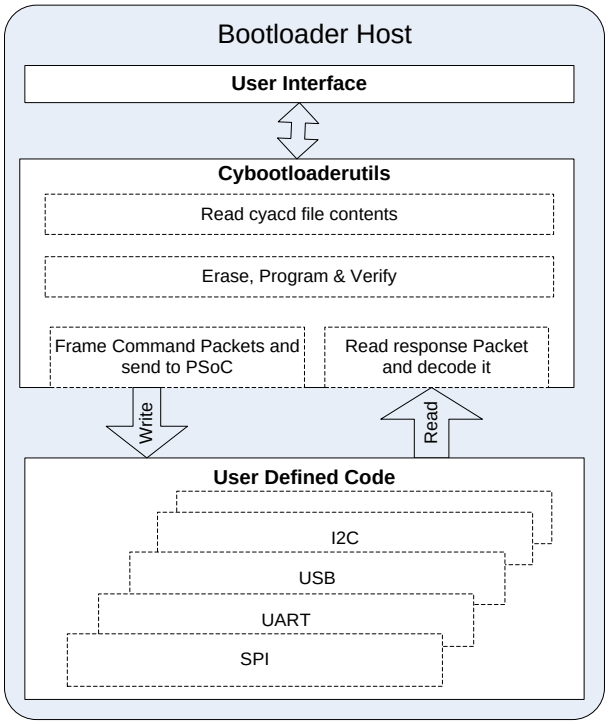
ソフトウェアの Professional 版を Microsoft 社から購入することができます。

64 ビット Windows プラットフォーム用の DLL を対象とする場合、この MSDN ページをご覧ください。

本節で説明した GUI を実装するには、Visual C#または Visual C++について多少経験を持つ必要があります。

PSoC 3 または PSoC 5LP ブートローダ ホストのアーキテクチャが図 40 に示されています。

図 40. ブートローダ ホスト アーキテクチャ



C#ブートローダ ホスト アプリケーションを作成するステップは以下の通りです:

1. PSoC Creator 提供の C 関数 (cybootloader Utils) 用に DLL を作成します。
2. C# GUI (ユーザー インターフェース) を作成します。
3. ステップ 1 で作成した DLL から必須のブートローダ関数をインポートします。
4. シリアル ポート (UART) 通信インターフェースを使用して、通信機能に定義を提供します。
5. Windows Forms アプリケーションを完了します。

次の節では、それぞれのステップを詳細に説明します。

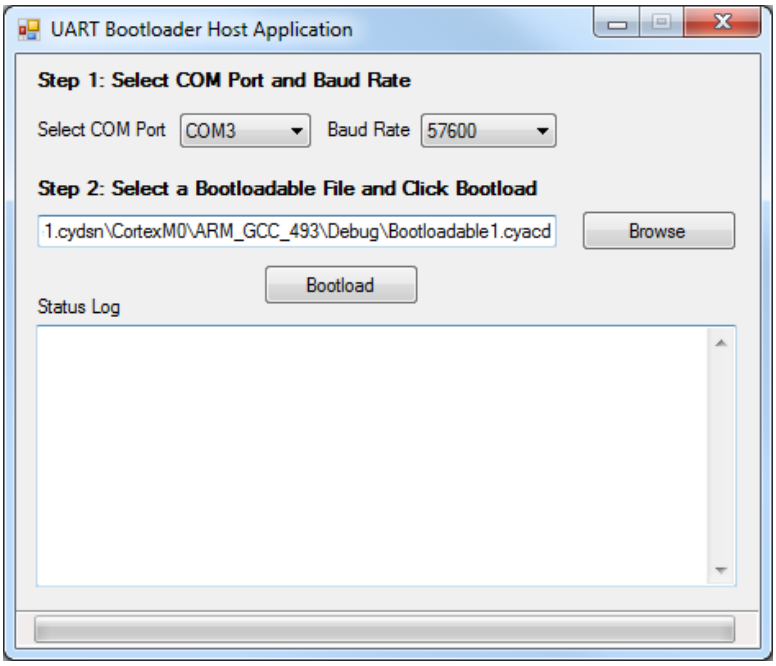
ステップ 1: bootloaderutils.dll の作成

PSoC Creator 提供の C 関数用に DLL を作成します。様々な Windows プラットフォーム用に Visual C++2010 Express Edition で作成された DLL (*BootLoad_Utils.dll*) のコピーは、このアプリケーション ノート (*BootLoad_Utils_dll*) に添付されています。

ステップ 2: C# Windows Forms アプリケーションの作成

C# Windows Forms アプリケーションを作成します。ツールボックスから serialPort などの必要な Windows フォームをインクルードします。図 41 は、本書で用意されている UART ブートローダ用に作成された GUI のスクリーン ショットを示します。

図 41. UART ブートローダ ホスト GUI



ステップ 3: ステップ 1 で作成した DLL から必須のブートローダ関数をインポート

BootLoad_Utils.dll によって提供されるメソッドとオブジェクトは、Windows が提供するプラットフォーム呼び出し (pInvoke) ユーティリティを使い、C#アプリケーションからアクセスされます。プラットフォーム呼び出しは、マネージ コードが DLL で実装されるアンマネージ関数を呼び出す[12]ことを可能にするサービスです。これは、エクスポートされた関数を探し出し呼び出し、そして、その関数の引数 (整数、文字列、アレイ、構造など) をマーシャリングします。エクスポートされた DLL 関数を使用するためには:

a. C#ホスト アプリケーションによって直接呼び出される DLL 中の関数 (`CyBtldr_Program()`、`CyBtldr_Erase()`、`CyBtldr_Verify()` および `CyBtldr_Abort()` など) を識別します。

b. DLL 関数を含むクラスを作成します。クラスで、関数の名前とそれらを含む DLL の名前を指定します。

   既存クラスを使用すること、各アンマネージ関数に個々のクラスを作成すること、またはアンマネージ関数のセットを含む一個のクラスを作成すること、上記のいずれも可能です。

c. マネージ コードにプロトタイプを作成します。

   C#では DllImport 属性を使用して DLL や関数を識別します。メソッドを「static」および「extern」モディファイアでマークします。これらのプロトタイプは単純であり、必要な DllImport 属性を定義しないことに注意してください。詳細はソース コードをご参照ください。

```
[[DllImport("BootLoad_Utils.dll")]
int CyBtldr_Program(string file, ref CyBtldr_CommunicationsData comm,
CyBtldr_ProgressUpdate update);
```

[1] .NET ランタイムの制御下で実行されるコードは「マネージ コード」と呼ばれます。

[2] ランタイムの制御なしで実行されるコードは「アンマネージ コード」です。

```
[DllImport("BootLoad_Utils.dll")]
int CyBtldr_Erase(string file, ref CyBtldr_CommunicationsData comm,
CyBtldr_ProgressUpdate update);

[DllImport("BootLoad_Utils.dll")]
int CyBtldr_Verify(string file, ref CyBtldr_CommunicationsData comm,
CyBtldr_ProgressUpdate update);

[DllImport("BootLoad_Utils.dll")]
int CyBtldr_Abort();
```

d. DLL関数を呼び出します。ご自身のC#コードで、他のメソッドと呼び出すようにこのメッソドを呼び出します。

ステップ 4: Visual C＃で通信機能のための定義を提供

通常、ブートローダはブートローダ コマンドを送信するために特別な通信プロトコルを必要とします。PSoC Creator 内蔵の C ファイルは、*.cyacd* ファイルを読み出しブートローダ パケットをフレーム化する機能を持つ、C コードを提供します。通信関数を定義する必要があるだけです。したがって、UART 通信に対応する以下の 4 つのアンマネージ関数は、希望の通信コンポーネント (この例では、UART) を使用して C#で管理される必要があります。

- OpenConnection()
- CloseConnection()
- ReadData()
- WriteData()

それらを管理するには、まず OpenConnection、CloseConnection、ReadData および WriteData を行う関数を C#で定義します。そして、これらの関数を *BootLoad_Utils.dll* での OpenConnection()、CloseConnection()、 ReadData()、WriteData() の関数へのデリゲートとして使用します。

例えば、CloseConnection 関数を実装するために、下記のステップを行ってください:

a. 関数の実装を行うためにデリゲートが使用されることを指示します:

```
[UnmanagedFunctionPointer(CallingConvention.Cdecl)]
public delegate int OpenConnection_UART();
```

b. C#でその関数を定義します:

```
public int OpenConnection()
{
/*Open communication channel*/
serialPort.Open();
}
```

これらのステップはすべてのデリゲートした関数に必要とされます。

c. *cybtldr.h* での CyBtldr_CommunicationsData 構造も C#プログラムで宣言されなければなりません。これはクラス内で行われます。添付 C#コードの「Bootload_Utils」クラスをご参照ください。

```
[StructLayout(LayoutKind.Sequential)]
public struct CyBtldr_CommunicationsData
{
public OpenConnection_UART OpenConnection;
public CloseConnection_UART CloseConnection;
public ReadData_UART ReadData;
public WriteData_UART WriteData;
public uint MaxTransferSize;
};
```

d. 上記で定義した構造のインスタンス (comm_data) を作成します:

```
Bootload_Utils.CyBtldr_CommunicationsData comm = new
Bootload_Utils. CyBtldr_CommunicationsData();
```

e. 構造のメンバーをデリゲートします:

```
comm.OpenConnection = OpenConnection;
comm.CloseConnection = CloseConnection;
comm_data.ReadData = ReadData;
comm_data.WriteData = WriteData;
comm_data.MaxTransferSize = 64;
```

その次に、UART ブートローダに対する各機能が行うことについての簡単な説明:

```
public delegate int OpenConnection_UART();
```

UART ブートローダには、この関数は選択した COM ポートへの接続を行います。

```
public delegate int CloseConnection_ UART ();
```

この関数は COM ポートの接続を閉じます。

```
public delegate int ReadData_ UART (IntPtr buffer, int size);
```

この関数は、データが入力バッファで利用可能となるまでタイムアウト期間を待ちます。データ長は上記の API の「size」で指定され、バッファへのポインタは IntPtr で指定されます。

```
public delegate int WriteData_ UART(IntPtr buffer, int size);
```

この関数は、選択されたシリアル ポートにデータを書き込みます。送信されるデータはバッファにプリロードされます。データ長は上記の API の「size」で指定され、バッファへのポインタは IntPtr で指定されます。

```
public delegate void CyBtldr_ProgressUpdate(byte arrayID, ushort rowNum);
```

この機能は、ブートローダ処理の進捗を可視化する方法を提供します。この機能は、進捗の割合を示すテキスト ボックスを更新するのと同じくらい簡単であり得るが、プログレス バーを更新する重要なものです。

通信機能の定義は、付属の C#プロジェクト中の、*delegated_functions.cs* と *BootLoad_Utils_NativeCode.cs* に提供されます。

ステップ 5: Windows Forms アプリケーションの完了

必要な API がインポートされ、上記の通信 API に定義が与えられた後、これらは様々な制御動作を実現するために使用されます。例えば、Bootload ボタンを押すと、ブートロード動作が開始されます。そのため、ボタン クリックのイベントでは、`Bootload_Utils.CyBtldr_Program` の関数が呼び出される必要があります。

ホストの詳細な実装については、UART ブートローダ ホスト C#プロジェクトをご参照ください。

H 付録 H – キットの選択

表 15. 対象デバイスに対応するキットの選択

対象デバイス	キット名	ユーザー ガイド
CY8C42xx	CY8CKIT-042 PSoC 4 Pioneer Kit	CY8CKIT-042
CY8C4Axx	CY8CKIT-048 PSoC アナログ コプロセッサ Pioneer Kit	CY8CKIT-048
CY8C40xx-S	CY8CKIT-041-40xx 4 S シリーズ Pioneer Kit	CY8CKIT-041
CY8C41xx-S	CY8CKIT-041-41xx 4 S シリーズ Pioneer Kit	CY8CKIT-041
CY8C3xxx	CY8CKIT-030 PSoC 3 開発キット	CY8CKIT-030
CY8C5xxx	CY8CKIT-050 PSoC 5 開発キット	CY8CKIT-050

改訂履歴

文書名: AN68272 - PSoC® 3、PSoC 4、PSoC 5LP、および PSoC アナログ コプロセッサ UART ブートローダ

文書番号: 002-15786

版	ECN	変更者	発行日	変更内容
**	5414610	HZEN	09/06/2016	これは英語版 001-68272 Rev. *J を翻訳した日本語版 002-15786 Rev. **です。
*A	5803817	AESATMP9	07/07/2017	更新されたロゴと著作権。

ワールドワイド販売と設計サポート

サイプレスは、事業所、ソリューション センター、メーカー代理店および販売代理店の世界的なネットワークを持っています。お客様の最寄りのオフィスについては、サイプレスのロケーション ページをご覧ください。

製品

ARM® Cortex® Microcontrollers	cypress.com/arm
車載用	cypress.com/automotive
クロック&バッファ	cypress.com/clocks
インターフェース	cypress.com/interface
IoT (モノのインターネット)	cypress.com/iot
メモリ	cypress.com/memory
マイクロコントローラ	cypress.com/mcu
PSoC	cypress.com/psoc
電源用 IC	cypress.com/pmic
タッチ センシング	cypress.com/touch
USB コントローラー	cypress.com/usb
ワイヤレス	cypress.com/wireless

PSoC®ソリューション

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6

サイプレス開発者コミュニティ

フォーラム | WICED IOT Forums | Projects | ビデオ | ブログ | トレーニング | Components

テクニカル サポート

cypress.com/support

本書で言及するその他すべての商標または登録商標は、それぞれの所有者に帰属します。

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

©Cypress Semiconductor Corporation, 2011-2017. 本書面は，Cypress Semiconductor Corporation 及び Spansion LLC を含むその子会社（以下「Cypress」という。）に帰属する財産である。本書面（本書面に含まれ又は言及されているあらゆるソフトウェア若しくはファームウェア（以下「本ソフトウェア」という。）を含む）は，アメリカ合衆国及び世界のその他の国における知的財産法令及び条約に基づき Cypress が所有する。Cypress はこれらの法令及び条約に基づく全ての権利を留保し，本段落で特に記載されているものを除き，その特許権，著作権，商標権又はその他の知的財産権のライセンスを一切許諾しない。本ソフトウェアにライセンス契約書が伴っておらず，かつ Cypress との間で別途本ソフトウェアの使用方法を定める書面による合意がない場合，Cypress は，(1) 本ソフトウェアの著作権に基づき，(a) ソースコード形式で提供されている本ソフトウェアについて，Cypress ハードウェア製品と共に用いるためにのみ，かつ組織内部でのみ，本ソフトウェアの修正及び複製を行うこと，並びに (b) Cypress のハードウェア製品ユニットに用いるためにのみ，（直接又は再販売者及び販売代理店を介して間接のいずれかで）本ソフトウェアをバイナリーコード形式で外部エンドユーザーに配布すること，並びに (2) 本ソフトウェア（Cypress により提供され，修正がなされていないもの）が抵触する Cypress の特許権のクレームに基づき，Cypress ハードウェア製品と共に用いるためにのみ，本ソフトウェアの作成，利用，配布及び輸入を行うことについての非独占的で譲渡不能な一身専属的ライセンス（サブライセンスの権利を除く）を付与する。本ソフトウェアのその他の使用，複製，修正，変換又はコンパイルを禁止する。

適用される法律により許される範囲内で，Cypress は，本書面又はいかなる本ソフトウェア若しくはこれに伴うハードウェアに関しても，明示又は黙示をとわず，いかなる保証（商品性及び特定の目的への適合性の黙示の保証を含むがこれらに限られない）も行わない。いかなるコンピューティングデバイスも絶対に安全ということはない。従って，Cypress のハードウェアまたはソフトウェア製品に講じられたセキュリティ対策にもかかわらず，Cypress は，Cypress 製品への権限のないアクセスまたは使用といったセキュリティ違反から生じる一切の責任を負わない。加えて，本書面に記載された製品には，エラッタと呼ばれる設計上の欠陥またはエラーが含まれている可能性があり，公表された仕様とは異なる動作をする場合がある。適用される法律により許される範囲内で，Cypress は，別途通知することなく，本書面を変更する権利を留保する。Cypress は，本書面に記載のある，いかなる製品若しくは回路の適用又は使用から生じる一切の責任を負わない。本書面で提供されたあらゆる情報（あらゆるサンプルデザイン情報又はプログラムコードを含む）は，参照目的のためのみに提供されたものである。この情報で構成するあらゆるアプリケーション及びその結果としてのあらゆる製品の機能性及び安全性を適切に設計，プログラム，かつテストすることは，本書面のユーザーの責任において行われるものとする。Cypress 製品は，兵器，兵器システム，原子力施設，生命維持装置若しくは生命維持システム，蘇生用の設備及び外科的移植を含むその他の医療機器若しくは医療システム，汚染管理若しくは有害物質管理の運用のために設計され若しくは意図されたシステムの重要な構成部分としての使用，又は装置若しくはシステムの不具合が人身傷害，死亡若しくは物的損害を生じさせるようなその他の使用（以下「本目的外使用」という。）のためには設計，意図又は承認されていない。重要な構成部分とは，それの不具合が装置若しくはシステムの不具合を生じさせるか又はその安全性若しくは実効性に影響すると合理的に予想できるような装置若しくはシステムのあらゆる構成部分をいう。Cypress 製品のあらゆる本目的外使用から生じ，若しくは本目的外使用に関連するいかなる請求，損害又はその他の責任についても，Cypress はその全部又は一部をとわず一切の責任を負わず，かつ Cypress はそれら一切から本書により免除される。Cypress は Cypress 製品の本目的外使用から生じ又は本目的外使用に関連するあらゆる請求，費用，損害及びその他の責任（人身傷害又は死亡に基づく請求を含む）から免責補償される。

Cypress，Cypress のロゴ，Spansion，Spansion のロゴ及びこれらの組み合わせ，WICED，PSoC，CapsSense，EZ-USB，F-RAM，及び Traveo は，米国及びその他の国における Cypress の商標又は登録商標である。Cypress のより完全な商標のリストは，cypress.com を参照すること。その他の名称及びブランドは，それぞれの権利者の財産として権利主張がなされている可能性がある。