AN73503

PSoC® USB HID ブートローダ

著者: Robert Murphy, Keith Mikoleit

関連製品ファミリ: USB 対応のすべての PSoC 3、PSoC 4 L シリーズ、および PSoC 5LP 製品

関連するサンプル コードおよび他の資料: 完全なリストについては、ここをクリックしてください。

更にサンプル コードをお求めでしょうか？以下の通りご対応いたします。

PSoC のサンプルコードのリストにアクセスするには、サンプル コード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。

本アプリケーション ノート (AN73503) では、USB ヒューマン インターフェース デバイス (HID) クラスを使用して PsoC デバイス用に USB ブートローダを実装する方法について説明します。また、Windows ベースの USB ブートローダのホスト プログラムをビルドする方法も説明します。

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 2
1.3 ブートローダの機能フロー 3
1.4 USB ブートローダに関する注意事項 3
1.5 サンプル コード 4
2 ブートローダ プロジェクト 4
2.1 プロジェクトの作成 4
2.2 ブートローダ コンポーネントの設定 5
2.3 USB コンポーネントの設定 5
2.4 入力ピンの設定 6
2.5 ピンの配置およびクロックの設定 6
2.6 ブートローダ ファームウェア 8
3 ブートローダブル (アプリケーション) プロジェクト 8
3.1 プロジェクトの作成 8
3.2 ブートローダブル コンポーネントの設定 9
3.3 プロジェクトの残り部分の設定 9
4 PSoC Creator のブートローダ ホスト 10
5 ブートローダ ホストのビルド 11
5.1 必要なリソース 11
5.2 ブートローダ ホスト アプリケーションの作成 12
6 まとめ 15
7 関連リソース 16
A 付録 A – USBFS HID コンフィギュレーション 17
B 付録 B – ブートローダとデバイス リセット 26
C 付録 C – ホスト コア API 29
改訂履歴 30
ワールドワイド販売と設計サポート 31
製品 31
PSoC®ソリューション 31
サイプレス開発者コミュニティ 31
テクニカル サポート 31

1 はじめに

ブートローダは MCU システム設計で一般的に使用されます。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することができます。工場では、製品への初期ファームウェア プログラミングは一般的に MCU の Joint Test Action Group (JTAG) または Serial Wire Debugger (SWD) のインターフェースを介して実行されます。しかし、これらのインターフェースは通常、現場では利用できません。

そこでブートローディングを活用します。ブートローディングは USB や I^2C などの標準通信インターフェースを経由してシステム ファームウェアのアップグレードを可能にします。ブートローダはホストと通信し、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の方法について説明します:

- PSoC 3、PSoC 4 L シリーズ、または PSoC 5LP デバイスに USB ブートローダを追加
- ブートローディング用にアプリケーション プロジェクトを準備
- PSoC Creator と共に提供されるブートローダ ホスト プログラムを使用
- ユーザー独自の Windows ベースのブートローダ ホスト プログラムを作成

本アプリケーション ノートは PSoC および PSoC Creator IDE の経験者を対象としています。PSoC の初心者である場合、AN54181、AN79953、および AN77759 の PSoC 入門アプリケーション ノートをご参照ください。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。

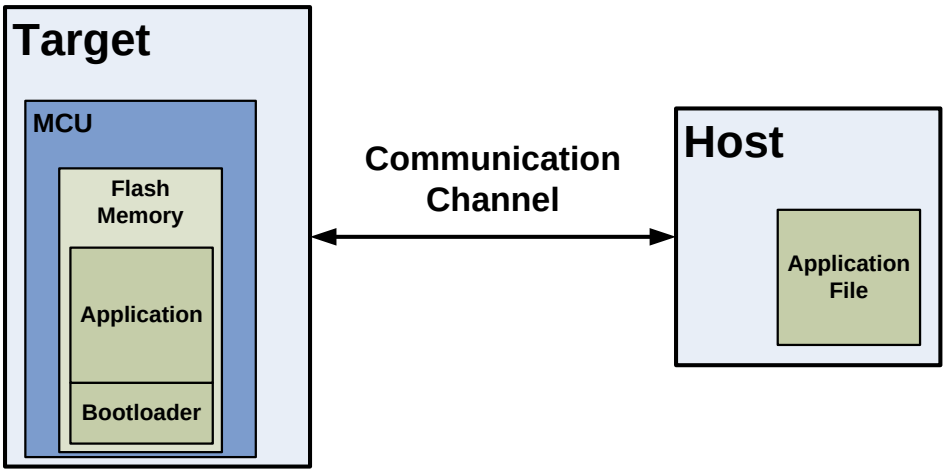
本アプリケーション ノートは、読者がブートローダの概念を理解していることも想定しています。これらの概念をご存知ない場合、「AN73854, PSoC Introduction to Bootloaders」のアプリケーション ノートをご参照ください。

最後に、本アプリケーション ノートは読者が USB に慣れていることを想定しています。USB または USB HID クラスが初めてであれば、「AN57294 - USB 101: An Introduction to Universal Serial Bus 2.0」または「AN57473 - USB HID Basics with PSoC」のアプリケーション ノートをご参照ください。

1.1 用語および定義

図 1 に示す通り、製品に組み込まれたファームウェアは通常動作とアプリケーション更新という 2 つの異なる目的で通信ポートを使用することを示します。アプリケーションを更新する組み込みファームウェアの部分は**ブートローダ**と呼ばれています。

図 1. ブートローダ システムのブロック図



一般に、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。ホストは外部の PC またはターゲットと同じ PCB にある別の MCU 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**、または単に**ブートロード**と呼ばれます。フラッシュに配置されるデータは**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もう 1 つの用語は**インシステム プログラミング(ISP)** です。サイプレスには似た名前の In-System Serial Programmer (インシステム シリアル プログラマ) (ISSP) と呼ばれる製品、および Host-Sourced Serial Programming (ホストソース シリアル プログラミング) (HSSP) と呼ばれるオペレーションがあります。詳細については、アプリケーション ノート AN73054 または「AN84858, PSoC 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ための最初のステップは、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製品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。もし、ターゲットから「OK」の応答が送信されると、ブートローディングは開始できます。

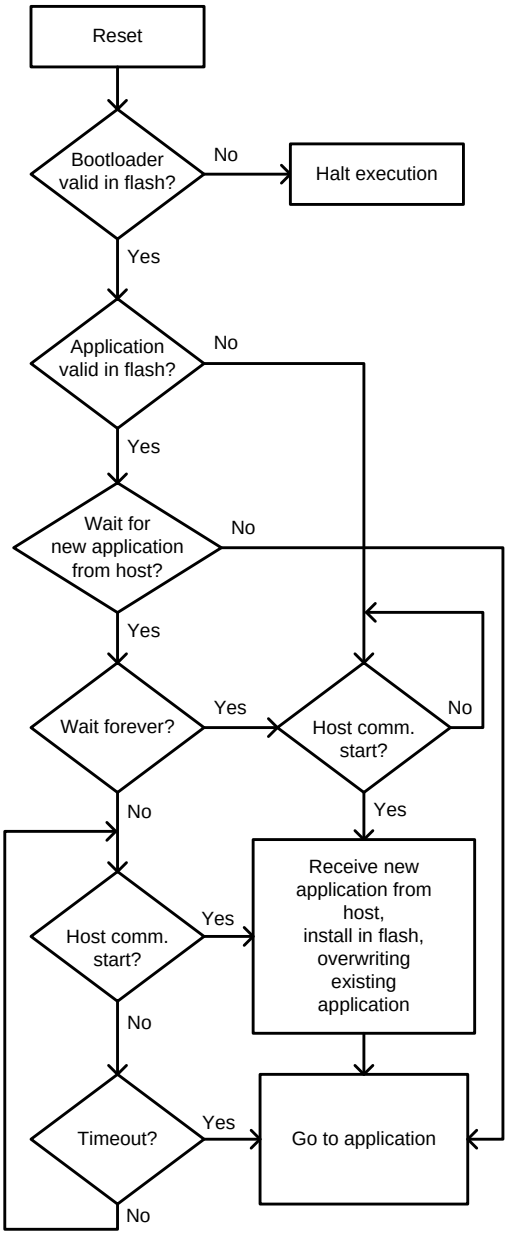
ブートローディング中、ホストは新しいアプリケーション用のファイルを読み出し、フラッシュ書き込みコマンドにそれを変換し、それらのコマンドをブートローダに送信します。ファイル全体が送信された後、ブートローダは新しいアプリケーションに制御を渡すことができます。

1.3 ブートローダの機能フロー

ブートローダは通常、リセット時に最初に実行されます。図 2 に示すように、次の動作が実行できます。

- 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／フラッシュの更新動作を実行
- 最後に、アプリケーションに制御を渡す

図 2. ブートロード処理のフローチャート



1.4 USB ブートローダに関する注意事項

ブートローダ通信インターフェースとして USB ポートを使用する際、考慮すべきいくつかの項目があります。USB HID クラスでは、ドライバーを必要とせずに、デバイスがあらゆるホスト オペレーティング システムで動作できるという利点があります。

ブートローダがアプリケーションに制御を渡す前に限られた時間待機し、USB デバイスが列挙するために時間が掛かることに留意してください。もし、USB 列挙に時間が掛かり過ぎると、ホストはターゲットデバイスがリセットした後にブートロード動作を開始するチャンスを逃すことがあります。

アプリケーションに制御が渡された後に、ブートローダに制御を戻すいくつかの方法があります。

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 API 関数 (Bootloadable_Load()) を備えています。これによりアプリケーションがブートローダに制御権を渡し返すことができます。これは製品を使用するユーザーがファームウェアの更新を開始することができる良い方法です。

この方法の問題はアプリケーションの更新を実行することが、アプリケーション コードに依存することです。もし、アプリケーションにブートローダの転送ができないバグがあった場合、それはどうなりますか？ファームウェアをアップグレードおよび修正する能力を確保するには、ブートローダ プロジェクトにブートローダを起動するフェイルセーフ手法を置くべきです。

起動時に無限の待機時間でブートローダの起動

USB ブートローダで破損したアプリケーションや長い USB 列挙時間などの問題に対処するために、ホスト コマンドが受信されるまでブートローダに留まる方法を追加することができます。Bootloader_Start() を呼び出して通常のルーチンを実行する前に、いくつかのユーザー入力を確認するようにブートローダ プロジェクトをカスタマイズすることができます。

詳細については、ブートローダ ファームウェアセクションをご参照ください。

1.5 サンプル コード

次のセクションは、PSoC Creator ブートローダとブートローダブル プロジェクトの作成手順を説明します。サンプル コード CE95391 を利用すれば完全なプロジェクトを入手できます。

2 ブートローダ プロジェクト

PSoC ブートローダ システムには最低 2 個の PSoC Creator プロジェクト- 1 つ目はブートローダ プロジェクトで、残りは少なくとも 1 個のブートローダブル (アプリケーション) プロジェクトが含まれています。本セクションはブートローダ プロジェクトの作成方法について説明します。次のセクションはブートローダブル プロジェクトの作成方法について説明します。

ブートローダ プロジェクト デザインは図 3 に示すように、PSoC Creator ブートローダおよび USBFS コンポーネントを含みます。USBFS コンポーネントは PC ホストと通信して、コマンドおよび新しいアプリケーション イメージを取得します。ブートローダ コンポーネントはフラッシュ プログラミング、ホスト コマンド／応答プロトコルを行い、ブートローダブル アプリケーションを起動します。

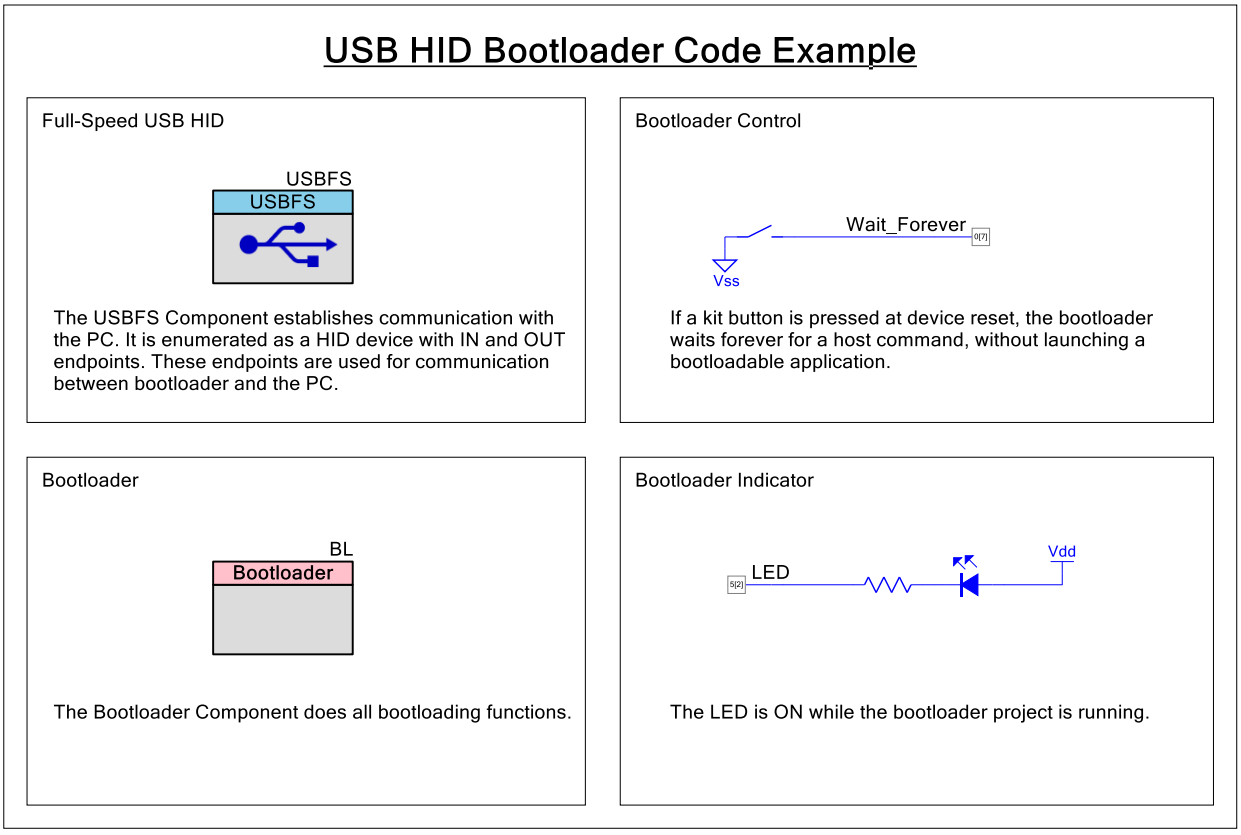
2.1 プロジェクトの作成

開始するには、新しい PSoC Creator プロジェクトを作成します。Create Project ダイアログからターゲットのハードウェアやデバイスを選択します。

ワークスペースを「PSoCx_USB_Bootloader」などの名前にします。プロジェクトを「USB _Bootloader」などの名前にします。

次に、PSoC Creator コンポーネント – USBFS、ブートローダ、および他のコンポーネント – を回路図に追加します。図 3 に示す通り、コンポーネントの名前を変更することもできます。

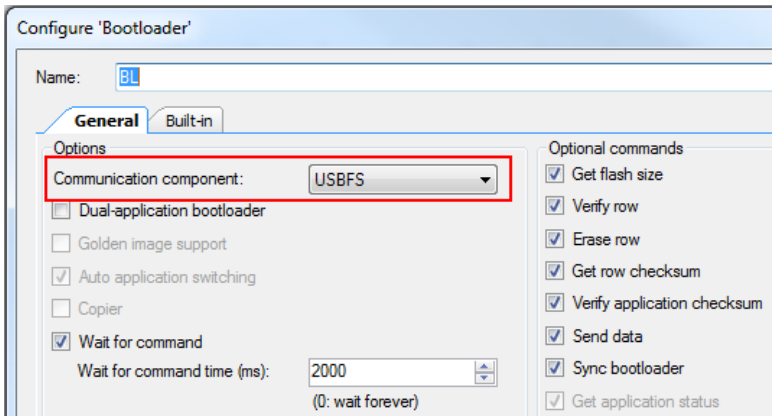
図 3. サンプル コード CE95391 のブートローダ プロジェクトの回路図



2.2 ブートローダ コンポーネントの設定

図 4 に示すように、ブートローダ コンポーネントの必要となる変更は「Communication component」を USBFS にするだけのことです。これはブートローダが USB とインターフェースするために必要な機能を生成するように PSoC Creator に指示します。

図 4. ブートローダ コンポーネントのカスタマイザ



また、「**Wait for command**」の設定を確認することも必要です。USB デバイスの列挙時間を考慮する必要があるため、これらは USB ブートローダに特に重要です。

- 「**Wait For Command**」**:** デバイスのリセット時に、ブートローダはホストからコマンドが入力されるまで待機するか、または直ちにアプリケーション コードにジャンプします。このオプションを選択すると、ブートローダは **Wait for command time** パラメーターで指定したタイムアウトまでホストからのコマンドを待機します。タイムアウト期間内にブートローダがホストからコマンドを受信しない場合、アプリケーション コードにジャンプします。
- 「**Wait for command time (ms)**」**:** 上記のオプションが有効の場合、このパラメーターはブートローダがアプリケーション コードにジャンプする前の待機時間です。0 値は、「Wait forever (永久に待機)」と解釈されます。デフォルト値は 2 秒のタイムアウトです。

デバイスのフラッシュに有効なアプリケーションがインストールされていない場合、ブートローダはこれらの設定を問わずにホスト コマンドを永久に待機します。

ブートローダ コンポーネントの完全な設定情報については、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。

2.3 USB コンポーネントの設定

次は、ブートローダ通信をサポートするために USBFS コンポーネントを設定します。外部で提供されるドライバー ファイルをホスト PC に必要としないように USB HID クラスを使用します。

ダブルク リックして、USBFS コンポーネントの設定を始めます。そして、コンフィギュレーション ダイアログで、デフォルトの Device Descriptor ルートを削除します。図 5 に示すように、**Device Descriptor** タブをクリックして、「**X**」ボタンをクリックします。

図 5. デフォルトの USB 設定の削除

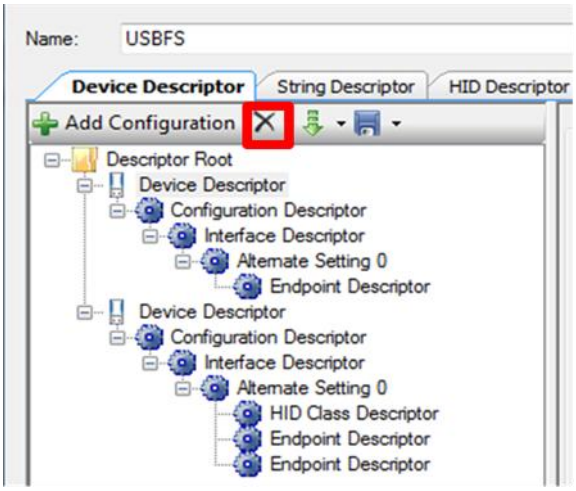
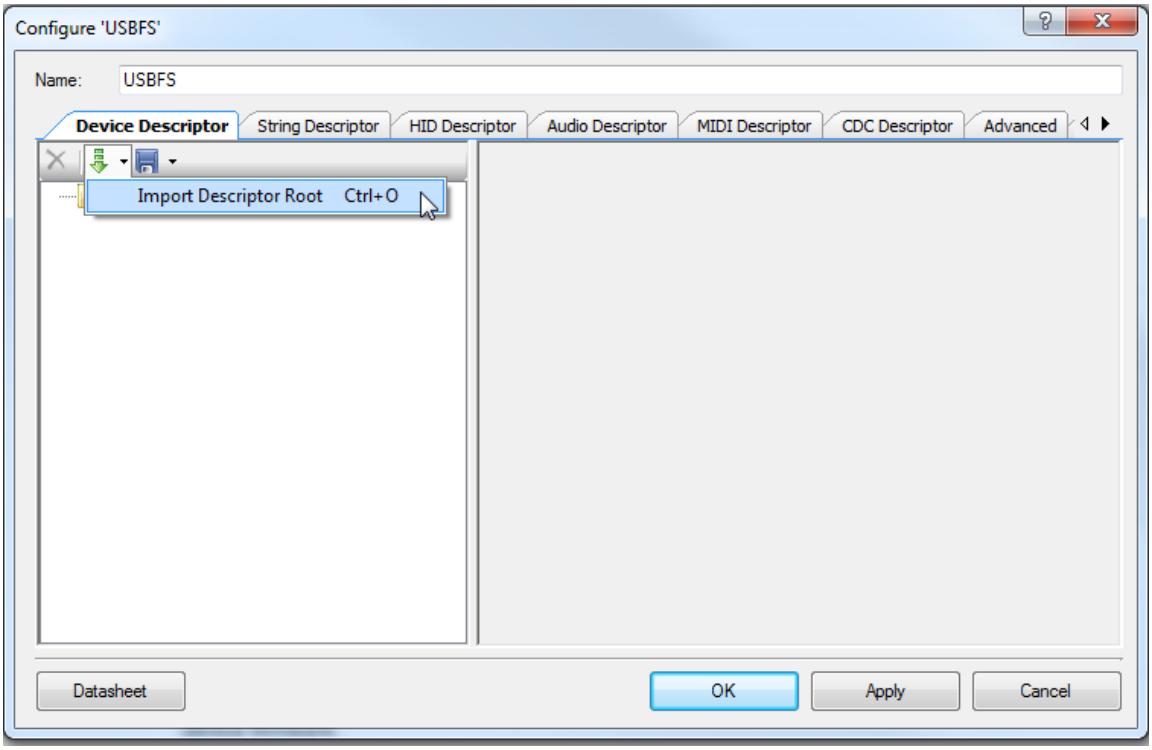


図 6 に示すように、次に「Insert Configuration」オプションを使用して、ブートローダ用 USB 設定をインポートします。ブートローダ用の完全な USB 設定は*.xml* ファイルで保存され、PSoC Creator に含まれます:

<PSoC Creator InstallDir> \ psoc \ content \ cycomponentlibrary \ CyComponentLibrary.cylib \ USBFS_v_2_80 \ Custom \ template \ Bootloader.root.xml

その設定がインポートされた後、PSoC Creator は自動的に必要な USB ディスクリプタのロードを開始します。USB 設定の詳細については、付録 A をご参照ください。

図 6. USB の Insert Configuration



2.4 入力ピンの設定

前述のように、ブートローダを起動するためにバックドア オプションを提供した方が良いです。これを行う 1 つの方法は、起動時にピンの状態を確認することです。エンド アプリケーションにスイッチなどのユーザー インターフェースが既にあった場合に有効な方法です。

Digital Input Pin コンポーネントを使ってピンの状態を読み出します。サンプル コード CE95391 ではピンがキットのボタンに接続されます。ボタンは押された時、グランドに短絡するため、ピン コンポーネントをプルアップ抵抗 (Resistive pull up) に設定します。そのため、ボタンを押した時に、ピン入力の状態が「0」になり、ボタンを離した時に、ピン入力の状態が「1」になります。

2.5 ピンの配置およびクロックの設定

全てのコンポーネントを設定した後、クロック リソースを設定し、ピンを配置します。

ワークスペース エクスプローラ ウィンドウで *USB_Bootloader.cydwr* をダブルクリックします。Pin Selection タブが表示されます。Wait_Forever と他のピン コンポーネントを、ご使用のキットにあるデバイスの物理ピンに割り当てます。CE95391 を例としてご参照ください。

注: PSoC Creator によって、USB D+および D-データ ラインは自動的に割り当てられます。

注: ブートローダ プロジェクトのピン コンポーネントとアプリケーションのピン コンポーネントを同じ物理ピンに割り当てることが可能です。ブートローダとアプリケーションは独立したプロジェクトなので、同じピン名を使用することもできます。

次は **Clocks** タブをクリックし、いずれかのクロックをダブルクリックし、クロック コンフィギュレーション ウィザードを開きます。USB 用にクロックを設定します (図 7 は PSoC 4 L シリーズの場合で、図 8 は PSoC 3 および PSoC 5LP の場合です)。

図 7. PSoC 4 L シリーズの USB クロックの設定

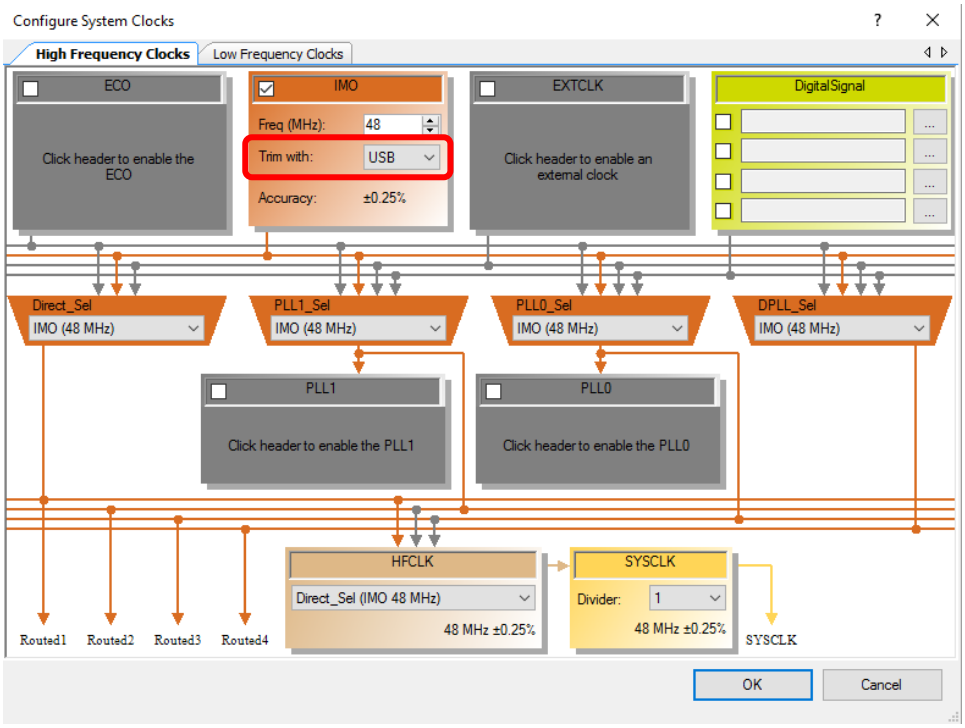
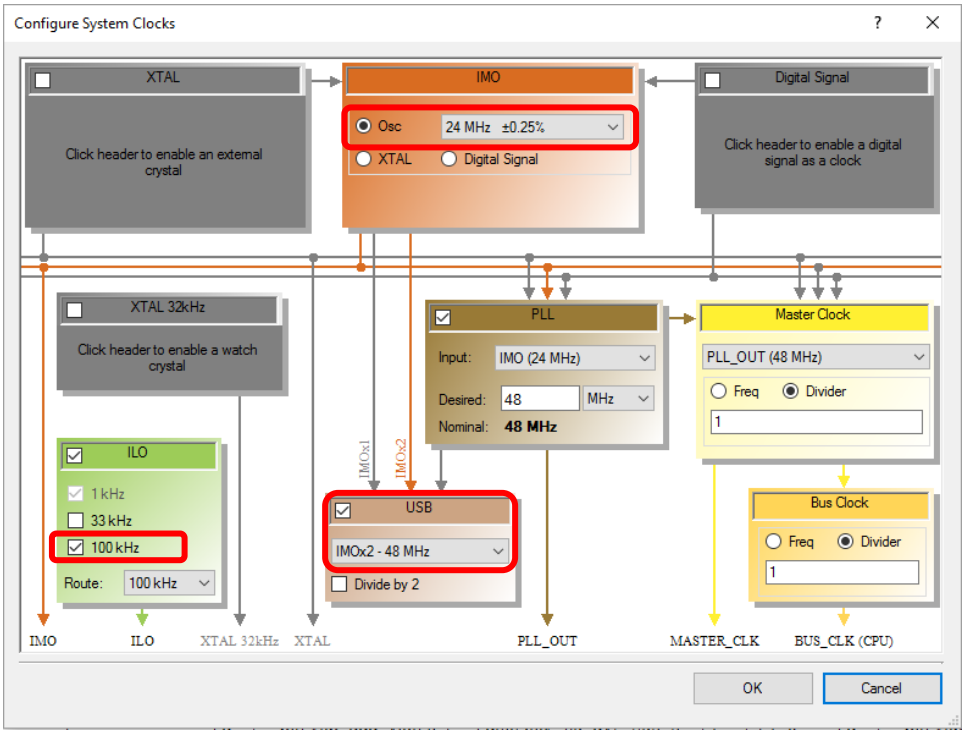


図 8. PSoC 3 および PSoC 5LP の USB クロックの設定



2.6 ブートローダ ファームウェア

プロジェクトのすべてのコンポーネントを設定した後、ブートローダ動作を制御するためのファームウェアを追加します。(プロジェクトの **Build** > **Generate Application** を実行した後ファームウェアを追加することを推奨します。)

多くの場合には、*main.c* に `Bootloader_Start()`のブートローダ コンポーネント API 呼び出し関数を追加するだけで済みます。(呼び出し関数の「Bootloader」部分をご使用のブートローダ コンポーネントの名前 (デフォルトの値から変更済みの場合) に置き換えます。)

`Bootloader_Start()`関数はすべてのブートロード機能を担当します。戻り値はなく、アプリケーションに制御権を渡す前に、PSoC デバイスをリセットします。ブートローダとデバイスのリセットの詳細については、付録 B をご覧ください。

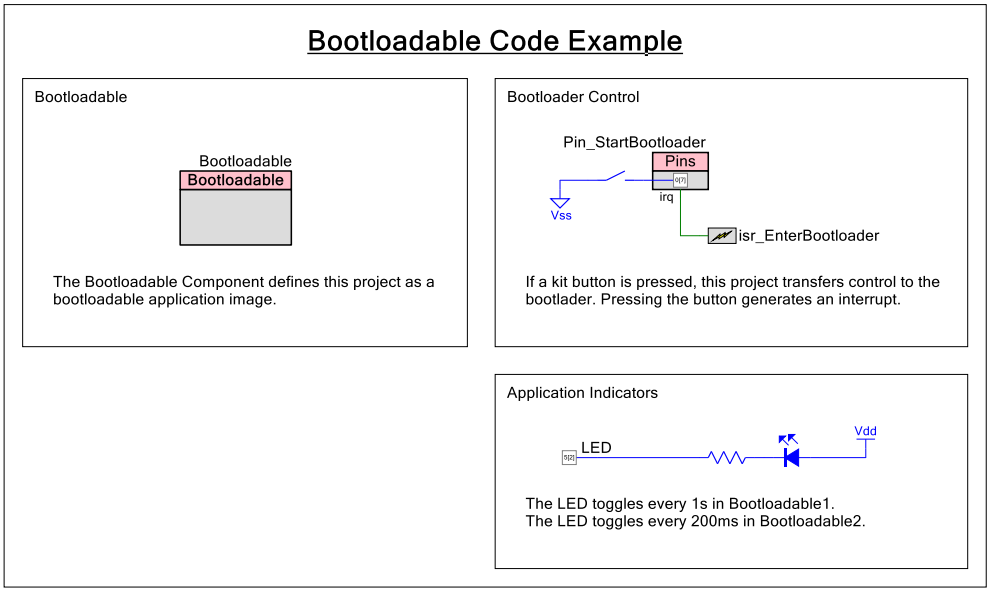
図 3 に示すブートローダ インジケータおよび制御ピンの使用方法についてより複雑な例については、サンプル コード CE95391 をご参照ください。

3 ブートローダブル (アプリケーション) プロジェクト

ブートローダブル プロジェクトは図 9 に示すように、PSoC Creator ブートローダブル コンポーネントを含みます。他のコンポーネントは希望アプリケーションを実装します。

サンプル コード CE95391 には、キットの LED を異なる周波数で点滅させる 2 個のブートローダブル プロジェクトがあります。また、プロジェクトはキットのボタン スイッチも読み出します。スイッチが閉じると、プロジェクトは制御権をブートローダに転送します。

図 9. ブートローダブル プロジェクトの回路図



3.1 プロジェクトの作成

新しい PSoC Creator プロジェクトを作成します。ターゲットのハードウェアやデバイスをブートローダ プロジェクトと同様に選択します。プロジェクトを「USB _Bootloadable」などの名前にします。ブートローダ プロジェクトのワークスペースに追加するか異なるワークスペースに保存することができます。[1]

次に、PSoC Creator コンポーネント – ブートローダブルおよび他のコンポーネント – を回路図に追加します。図 9 に示す通り、コンポーネントの名前を変更することもできます。

[1] 1 つの PSoC Creator のワークスペースは複数のプロジェクトを持つことができます。多くの場合、ブートローダ プロジェクトはその関連するブートローダブルと同じワークスペースに存在します。しかし、そのようにする必要はありません—ブートローダとブートローダブルは別のワークスペースや、PC 上の別の場所に存在することができます。PSoC を始める前に、全体的なシステムの開発ニーズのために、ワークスペース／プロジェクト計画を考えておくことをお勧めします。

3.2 ブートローダブル コンポーネントの設定

図 10 に示すように、ブートローダブル プロジェクトは常に関連ブートローダ プロジェクトの出力「*.hex*」および「*.elf*」ファイルに依存します。「*.hex*」ファイルを選択すると、関連の「*.elf*」ファイルが自動的に選択されます。

ブートローダブル コンポーネントを構成する前に、関連のブートローダ プロジェクトを完全にビルドする必要があります。

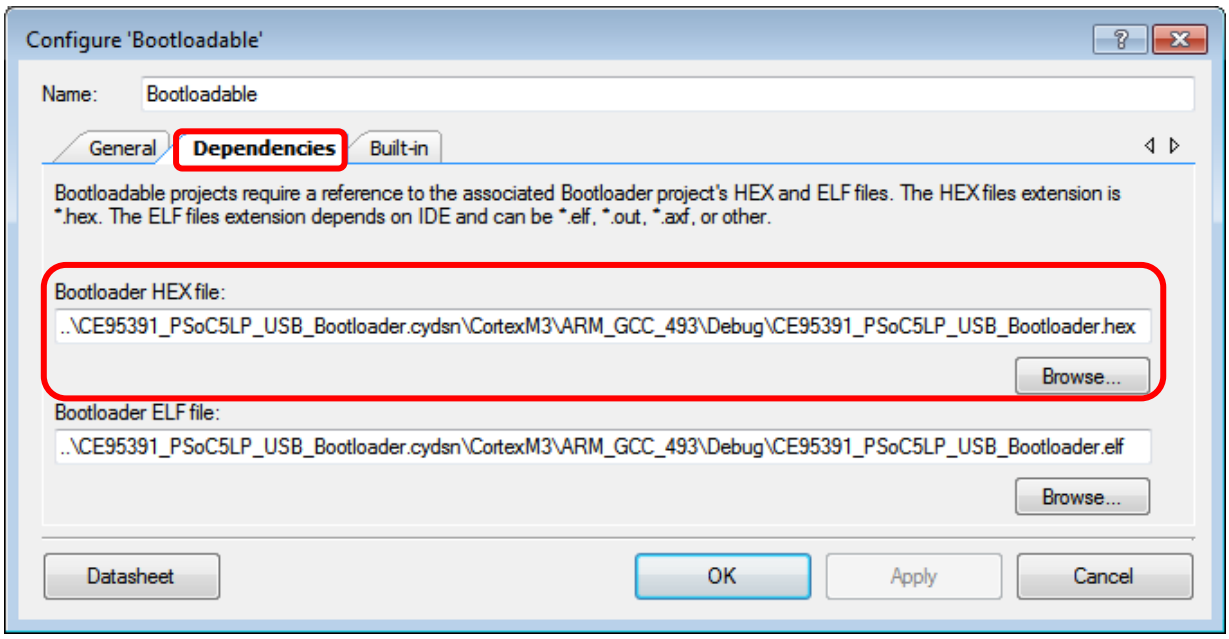
図 10 に、ブートローダ「*.hex*」ファイルが PSoC Creator ワークスペース内の相対パスにあることを示します。ただし、これは必須ではありません。ブートローダおよびブートローダブル プロジェクトは異なるワークスペースにあっても構いません。どちらの場合にも、ブートローダ プロジェクトのコンパイラ出力のためのフォルダ内に*.hex* ファイルがあります。

<project folder>\USB_Bootloader.cydsn\DP8051\DP8051_Keil_903\Debug\ (PSoC 3 の場合)

<project folder>\USB_Bootloader.cydsn\CortexM3\ARM_GCC_493\Debug\ (PSoC 4 L シリーズと PSoC 5LP の場合)

ブートローダおよびブートローダブル ファイルの詳細については、「AN73854 - PSoC Introduction to Bootloaders」をご参照ください。

図 10 ブートローダブル コンポーネントの設定



3.3 プロジェクトの残り部分の設定

ブートローダブル コンポーネントが設定された後、ブートローダブル プロジェクトの残り部分を、標準の (非ブートローダブル) プロジェクトと同様な方法でビルドできます。他のコンポーネントを必要に応じて回路図に追加し、設定してからファームウェアを追加します。サンプル コード CE95391 を例としてご参照ください。

ブートローダブル プロジェクトを複数作成し、それぞれを以前作成したブートローダ プロジェクトと結合させることができます。ブートローダ プロジェクトがターゲットのハードウェアにインストールされた後、異なるブートローダブル プロジェクトをダウンロードすることでターゲットの機能を変更できます。

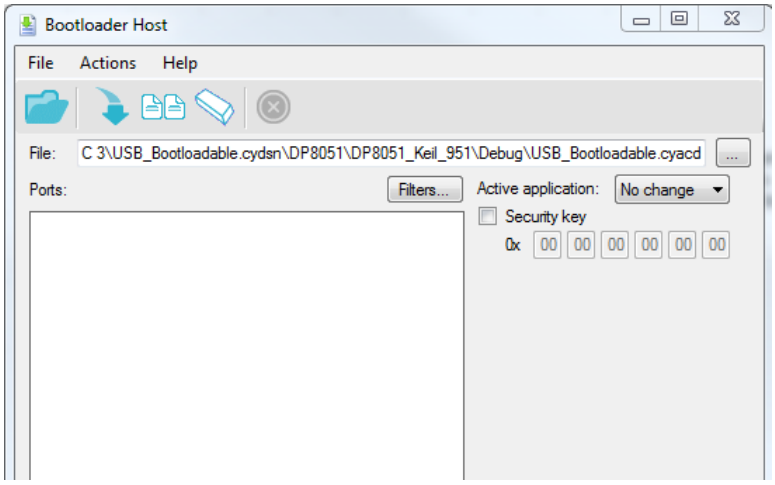
これより以降の本アプリケーション ノートでは、PSoC Creator と共に提供されるブートローダ ホスト プログラムの使用法、および PC ベースのブートローダ ホストのビルド方法について説明します。

4 PSoC Creator のブートローダ ホスト

PSoC Creator は、ブートローダおよびブートローダブル プロジェクトの使用を容易にするためにブートローダ ホスト プログラムを提供します。以下の手順に従って行ってください:

1. ブートローダ プロジェクトをビルドして、ターゲットとなる PSoC にプログラムします。
2. ブートローダ ホスト ツールを開きます (図 11)。PSoC Creator メニューから **Tools** > **Bootloader Host** を選択します。

図 11. PSoC Creator のブートローダ ホスト プログラム



3. 図 12 に示す通り、**Filters** をクリックし、ブートロードしようとする USB デバイス (ご使用の PSoC ターゲット) を識別するためにフィルターを追加します。

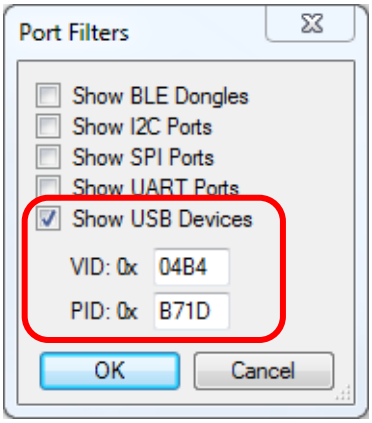
   ベンダ ID (VID) および製品 ID (PID) がブートローダ コンポーネントの設定に定義されているものと一致することを確かめてください。

4. **Open File** アイコンをクリックし、ブートローダブル ファイルを参照します。ファイルは *.cyacd* の形式で、ブートローダブル プロジェクトのコンパイラ出力用のフォルダにあります。

   ブートローダおよびブートローダブル ファイルの詳細については、「AN73854 - PSoC Introduction to Bootloaders」をご参照ください。

5. ブートロードを開始するために **Program** をクリックするか、または F5 キーを押します。

図 12. USB フィルターの設定



ブートロード処理が完了した後、既定のタイムアウト期間が経過したら、制御権はブートローダからブートローダブル アプリケーションに渡されます。

5 ブートローダ ホストのビルド

本節に、カスタムの USB ブートローダ ホストを実装する Windows 用のグラフィカル ユーザー インターフェース (GUI) アプリケーションの作成法について説明します。ここでは重要なステップのみ説明します。Windows のアプリケーション開発または Microsoft Visual Studio のご経験がない方は、関連リソース節をご参照ください。

ご参考のために、完全なブートローダ ホスト アプリケーションの Visual Studio プロジェクト - *USBBootloaderHost_VC2015.zip* を本アプリケーション ノートに添付しております。コードをプロジェクトのソース ファイルから開発中のプロジェクトにコピー&ペーストすることも可能です。

5.1 必要なリソース

ブートローダ ホストを作成するには以下のものが必要です:

Visual Studio

Visual Studio 版は、様々な開発ツールを提供し、C#や C++などのプログラミング言語に対応しています。

本アプリケーション ノートにある説明は Visual Studio Community 2015 用であることに、ご注意ください。別のバージョンの Visual Studio は異なる手順を必要とすることがあります。

Visual C# Express 2015

これは、C#プログラミング言語を使って.NET アプリケーションを開発するための Microsoft 社の無償の IDE です。本アプリケーション ノート内の説明は無料版のものですが、完全版 (Visual Studio) を使用することもできます。

Visual C++ Express 2015

これは、.NET アプリケーションを開発するための Microsoft 社のもう 1 つの IDE です。これは C++プログラミング言語を使用します。本アプリケーション ノートでは、Visual C++ Express は、C モジュールを使用してダイナミック リンク ライブラリ (DLL) を生成するために使用されます。

CYUSB.dll

CYUSB.dll は、Visual Studio アプリケーションをサイプレスの USB デバイスにインターフェースするためにサイプレスが開発し保持する.NET ダイナミック リンク ライブラリ (DLL) です。この DLL は、Visual Studio 用の USB 開発ツールの完全なセットであるサイプレスの SuiteUSB パッケージとセットになっています。SuiteUSB をサイプレスの Cypress SuiteUSB からダウンロードします。

PSoC Creator とともに提供されるブートローダ API

ホスト プログラムの作成に使用される 4 つの API モジュールは PSoC Creator の以下の場所にあります:

<install folder> \ PSoC Creator \ 3.3 \ PSoC Creator \ cybootloaderutils.

これらのモジュールは、サイプレスのブートローダ コマンド／応答プロトコルを使って、ホストが PSoC Creator ブートローダ コンポーネントとのインターフェースのために必要なコードをすべて含んでいます。このプロトコルの詳細についてはブートローダ コンポーネント データシートまたはシステム リファレンス ガイドをご参照ください。

4 つのモジュールがあり、それぞれは一組の.c / .h ファイルです:

- cybtldr_api.c / .h
  このモジュールはブートロード動作の開始、フラッシュ行のプログラミング、行の消去、行の検証、およびブートロード動作の終了を行う低レベル関数を含んでいます。
- cybtldr_api2.c / .h
  このモジュールはブートロード プロセス全体を管理するより高いレベルの API です。デバイスのプログラミング、デバイスの消去、デバイスの検証およびブートロード動作の中止を行う関数を含んでいます。
- cybtldr_command.c / .h
  このモジュールはブートローダに送信するコマンド パケットを構成し、ブートローダからの応答パケットを解析します。
- 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 イメージを含む*.*cyacd* ファイルを解析します。

5.2 ブートローダ ホスト アプリケーションの作成

このプロセスは次の重要な段階があります:

段階 1: PSoC Creator とともに提供されるブートローダ ユーティリティ関数で DLL (ダイナミック リンク ライブラリ) を作成。

段階 2: C# Windows フォーム アプリケーション (すなわち、GUI) を作成。

段階 3: 通信関数の定義

段階 4: 段階 1 で作成した DLL から必要なブートローダ関数をインポート

段階 5: フォーム関数を修正

段階 6: ホスト エラー コードを定義

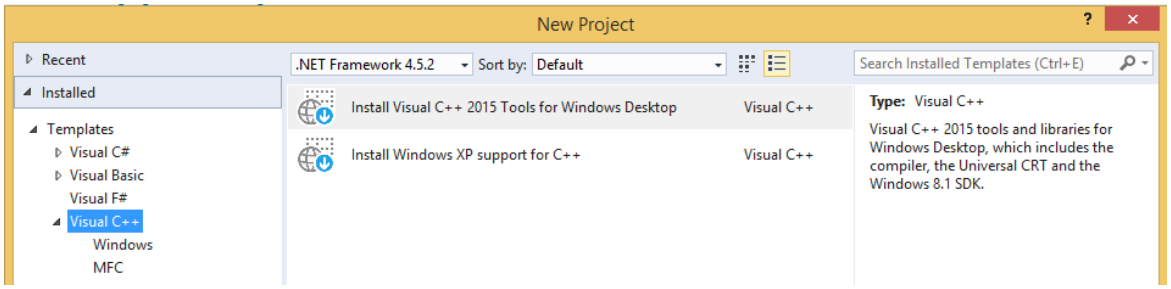
次の節では、それぞれの段階を詳細に説明します:

5.2.1 段階 1: DLL の作成

まず、PSoC Creator のブートローダ C ファイルを含む DLL を作成します。詳細については、付録 C – ホスト コア API をご参照ください。

1. Visual Studio Express を起動します。
2. C++ DLL プロジェクトを新規作成します。
   図 13 に示すように、Win32 コンソール アプリケーションを見るためには Windows Desktop 用の Visual C++ 2015 ツールをインストールする必要があることもあります。メニューから **File** > **New** > **Project…** > **Visual C++** > **Win32 Console Application** を選択します。

図 13. Windows Desktop 用の Visual C++ 2015 ツールをインストールします。



3. プロジェクトに Bootloader_Utils などの適切な名前を付けます。**OK** をクリックして **Next** をクリックします。
4. **Application type** を DLL に、**Additional options** をエンプティ プロジェクトに設定します。**Finish** をクリックして、これらの設定を有効にし、新しいプロジェクトを作成します。
5. ソリューション エクスプローラ ウィンドウで、プロジェクト名 (「Bootloader_Utils」) を右クリックして **Add** > **Existing Item…** を選択します。PSoC Creator フォルダ *<install folder>* \ *PSoC Creator* \ *3.3* \ *PSoC Creator* \ *cybootloaderutils* から次のファイルを追加します:
   - cybtldr_api.h、cybtldr_api.c
   - cybtldr_api2.h、cybtldr_api2.c
   - cybtldr_parse.h、cybtldr_parse.c
   - cybtldr_command.h、cybtldr_command.c
   - cybtldr_utils.h
6. プロジェクト名(「Bootloader_Utils」) を右クリックして、プロジェクト プロパティにプリプロセッサの定義を追加します。次のメニュー項目を選択します:
   Project Properties > **Configuration Properties** > **C/C++** > **Preprocessor** > **Preprocessor Definitions** > **Edit…**
   プリプロセッサ定義の末尾に「_CRT_SECURE_NO_WARNINGS」を挿入します。
7. プロジェクトをコンパイル – **Build** > **Build Solution** を選択します。これは *Bootloader_Utils.dll* ファイルを作成します。

5.2.2 段階 2: Windows フォーム アプリケーションの作成

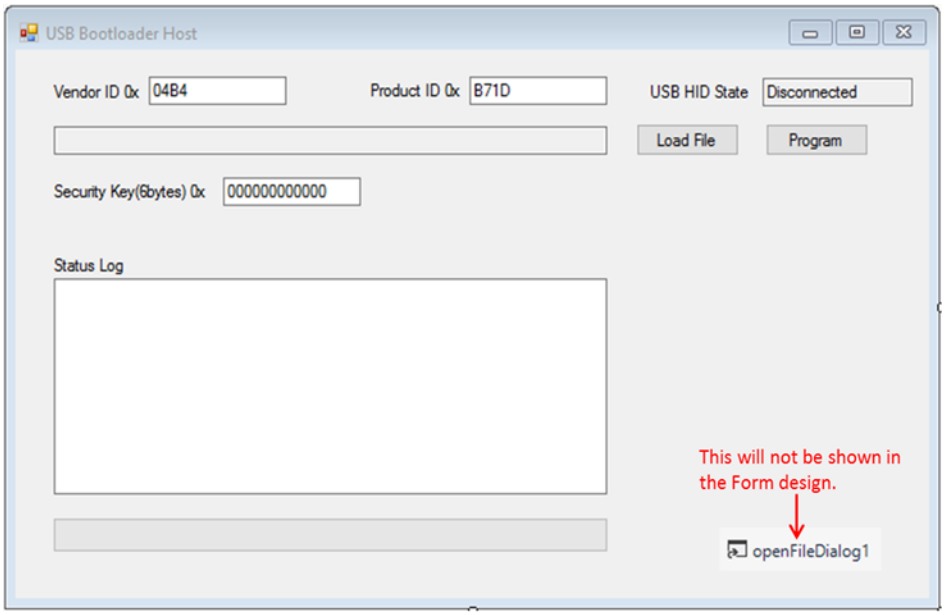
次の段階はブートローダ ホストに GUI を作成します。GUI は Visual C#を使用して作成します。主な要件は次の通りです:

- デバイスの VID と PID に基づいて認識し、HID デバイスに接続します。詳細については、関連リソースの USB アプリケーション ノートをご参照ください。
- ユーザーがセキュリティ キーを入力する入力ボックスを配置します。
- デバイス上にブートロードする新しい*.cyacd* ファイルをユーザーが選択できるようにします。
- *.cyacd* ファイルをデバイスにプログラムします。
- x86 プラットフォーム ターゲット (**Project Properties** > **Build** = > **Platform target** > **x86**) を使用します。
- いくつかのシステムには.NET Framework 4.x でビルドのエラーが発生する場合があるため、.NET Framework 3.5 を使用してください。メニューの **Project Properties** > **Application** > **Target framework** > **.NET Framework 3.5** を選択します。

以下の手順に従って行ってください:

1. Visual Studio を起動します。
2. C#プロジェクトの新規作成します。次のメニュー項目を選択します:
   File > **New** > **Project…** > **Visual C#** > **Windows Forms Application**.
3. プロジェクトに USBBootloaderHost などの適切な名前を付け、**OK** をクリックします。
4. そのフォームをデザインします。図 14 に示すように、5 つのラベル、5 つのテキスト ボックス、2 個のボタン、1 個の進捗バー、および 1 個の openFileDialog をツールボックス からメイン フォームに配置します。適切な名前とデフォルト値を与えます。

図 14. カスタム ブートローダ ホスト GUI



label1.Text = “Vendor ID 0x”
label2.Text = “Product ID 0x”
label3.Text = “USB HID State”
label4.Text = “Security Key(6bytes) 0x”
label5.Text = “Status Log”
textBox1.Text = “04B4”
textBox2.Text = “B71D”
textBox3.Text = “Disconnected”
textBox3.ReadOnly = true
textBox4.Text = “000000000000”
textBox5.Text = “”, textBox5.Multiline = true
textBox6.Text = “”, textBox6.ReadOnly = true

5.2.3 段階 3: 通信関数の定義

次の段階では、C#アプリケーションに *CyUSB.dll* から CyUSB 関数をインポートします。CyUSB 関数は.NET フレームワークにより作成されていないためアンマネージ コードと見なされることにご注意ください。アンマネージ コードの詳細については、Secure Coding Guidelines for Unmanaged Code をご参照ください。

関数をインポートするために、以下のステップに従ってください:

1. *CyUSB.dll* をプロジェクト フォルダにコピーします。Cypress SuiteUSB がインストール済みの場合、*CyUSB.dll* の格納場所は次の通りです:
   C:\Cypress\Cypress Suite USB 3.4.7\CyUSB.NET\lib.
2. プロジェクト名や References を右クリックして、CyUSB.dll を選びます:
   References > **Add Reference…** > **Browse…** > **CyUSB.dll**
3. CyUSB.dll の前のチェックボックスにチェックを入れ **OK** をクリックします。
4. 新しいクラス ファイルを追加します: プロジェクト名を右クリックして、**Add** > **New Item…** > **Class** を選択します。クラス ファイルを *Bootloader_Utils.cs* などの適切な名前にします。
5. 15 ページのコード 1 に示すように *Bootloader_Utils.cs* にコードを追加します。また、本アプリケーション ノートに添付される完全なブートローダ ホスト Visual Studio プロジェクト - *USBBootloaderHost_VC2015.zip* の *Bootloader_Utils.cs* からコードをコピー&ペーストすることもできます。

5.2.4 段階 4: ブートローダ関数のインポート

次の段階は、DLL からのブートローダ関数 (*Bootloader_Utils.dll*) を C#アプリケーションにインポートします。これは **dllimport** モディファイヤを使用して実現されます (コード 1 を参照)。

1. コード 1 に示すように、コードを *Bootloader_Utils.cs* に追加します。また、本アプリケーション ノートに添付される完全なブートローダ ホスト アプリケーション Visual Studio プロジェクト - *USBBootloaderHost_VC2015.zip* の *Bootloader_Utils.cs* ファイルからコードをコピー&ペーストすることもできます。
2. 段階 1 で作成した *BootLoader_Utils.dll* をプロジェクト の出力フォルダ (: ..\bin\x86\Debug) にコピーします。

5.2.5 段階 5: フォーム関数を修正

いくつかのイベントとプロセスのコードがフォームに追加される必要があります。以下のようなものが挙げられます:

- USB HID 接続および通信の設定
- ファイル管理
- 進捗表示

本アプリケーション ノートに添付される完全なブートローダ ホスト アプリケーション Visual Studio プロジェクト - *USBBootloaderHost_VC2015.zip* の *Form1.cs* ファイルからコードをコピー&ペーストすることが一番簡単な方法です。その後、コードを自分のアプリケーション用に修正できます。

5.2.6 段階 6: ホスト エラーコードを定義

クラス ファイルを作成し、*Bootloader_enum.cs* などの適切な名前にします。enum の定義は *cybtldr_utils.h* ファイルのものと一致する必要があります。本アプリケーション ノートに添付される完全なブートローダ ホスト アプリケーション Visual Studio プロジェクト - *USBBootloaderHost_VC2015.zip* の *Bootloader_enum.cs* ファイルからコードをコピー&ペーストすることが一番簡単な方法です。

コード 1. Bootloader_Utils.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Runtime.InteropServices;

namespace USBBootloaderHost
{
    class Bootloader_Utils
    {
        // Communication control structure
        [StructLayout(LayoutKind.Sequential)]
        public struct CyBtldr_CommunicationsData
        {
            public OpenConnection_USB OpenConnection;
            public CloseConnection_USB CloseConnection;
            public ReadData_USB ReadData;
            public WriteData_USB WriteData;
            public uint MaxTransferSize;
        };

        // Unmanaged code handling
        [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
        public delegate int OpenConnection_USB();

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

        [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
        public delegate int ReadData_USB(IntPtr buffer, int size);

        [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
        public delegate int WriteData_USB(IntPtr buffer, int size);

        // DLL importing
        [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
        public delegate int CyBtldr_ProgressUpdate(byte arrayID, ushort rowNum);
        [DllImport("BootLoader_Utils.dll", CallingConvention =
                   CallingConvention.Cdecl)]
        public static extern int CyBtldr_Program(string file, byte[] securityKey,
                                                 byte appId,
                                                 ref CyBtldr_CommunicationsData comm,
                                                 CyBtldr_ProgressUpdate update);
    }
}
```

6 まとめ

ブートローダは製品の現場での改良に有益な方法です。各ブートローダはホストと通信するためにハードウェア通信インターフェースが必要です。USB は通信インターフェースとして広範に使用可能です。

このアプリケーション ノートおよびサンプル コード CE95391 に説明された PSoC USB HID ブートローダは立ち上げと稼動に迅速かつ信頼性のあるソリューションです。USBFS コンポーネントは HID クラスとして設定されているため、特別なドライバーは不要で、どのオペレーティング システムにもインターフェースできます。

Windows ブートローダ ホスト プログラムは PSoC Creator と共に提供されます。また、本アプリケーション ノートに、サイプレスが提供したファイルを利用して自分の Windows ブートローダ ホストのアプリケーションを作成する方法についても説明しました。

7 関連リソース

サンプル コード

- CE95391 – USB HID Bootloader Code Example

アプリケーション ノート

- AN54181 – Getting Started with PSoC 3
- AN79953 – Getting Started with PSoC 4
- AN77759 – Getting Started With PSoC 5LP
- AN73854 – PSoC 3, PSoC 4, and PSoC 5LP Introduction to Bootloaders
- AN60317 – PSoC 3 and PSoC 5LP I^2C Bootloader
- AN86526 – PSoC 4 I^2C Bootloader
- AN68272 – PSoC 3, PSoC 4, and PSoC 5LP UART Bootloader
- AN84401 – PSoC 3 and PSoC 5LP SPI Bootloader
- AN57473 – USB HID Basics with PSoC 3 and PSoC 5LP (Fundamentals with Mouse and Joystick)
- AN58726 – USB HID Intermediate with PSoC 3 and PSoC 5LP (with Keyboard and Composite Device)
- AN56377 – PSoC 3 and PSoC 5LP - Introduction to Implementing USB Data Transfers

その他の情報

- SuiteUSB - Visual Studio 用 USB 開発ツール
- EZ-USB FX3 ソフトウェア開発キット

著者について

氏名: Robert Murphy
役職: スタッフ アプリケーション エンジニア
経歴: Robert Murphy は、パデュー大学を卒業し、電気工学技術の学士号を取得しました。

氏名: Keith Mikoleit
役職: シニア アプリケーション エンジニア
経歴: Keith Mikolei は、ウェスタン ワシントン大学を卒業し、電気工学技術の学士号を取得しました。

A 付録 A – USBFS HID コンフィギュレーション

本節に USB ブートローダ ディスクリプタの段階的な設定方法を示します。USB コンポーネントの設定に説明した通りに、ブートローダ テンプレート ファイルの *bootloader.root.xml* からディスクリプタをインポートできます。

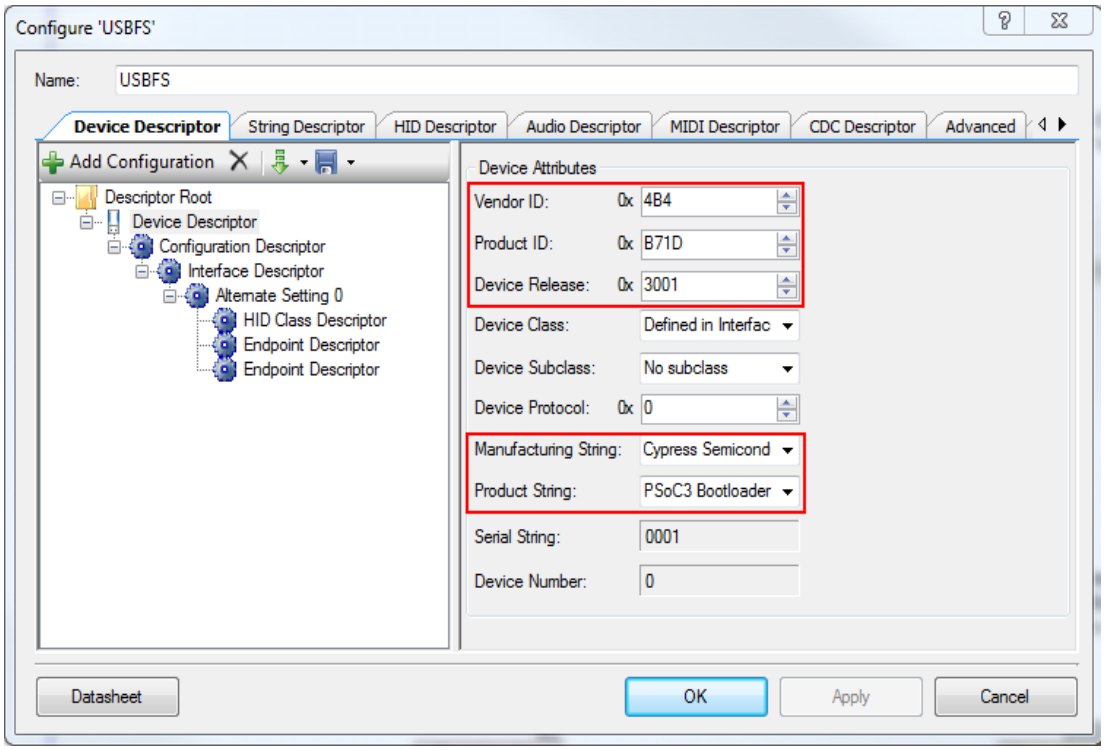
本アプリケーション ノートでは、外部で提供されるドライバー ファイルを必要としないように USB HID クラスを使用します。

USBFS コンポーネントを設定するために、PSoC Creator プロジェクトの回路図にあるコンポーネント シンボルをダブルクリックしてください。

A.1 デバイス ディスクリプタの作成

1. **Descriptor** ツリーで、**Device Descriptor** をクリックします。図 15 に示すように、**Device Attributes** のオプションを設定します:
   - **Vendor ID**: 0x04B4
   - **Product ID**: 0xB71D
   - **Device Release**: 0x0101
   - **Manufacturing String**: Cypress Semiconductor
   - **Product String**: PSoCx Bootloader

図 15. USBFS デバイス ディスクリプタ

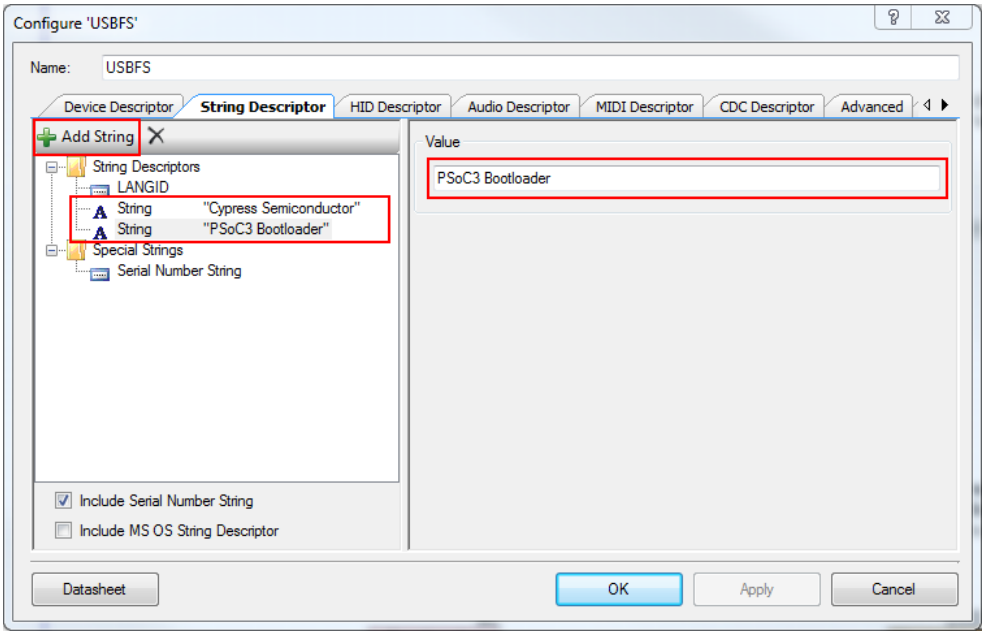


ハード要求はベンダーID (VID) と製品 ID (PID) の変更だけです。使用される VID (0x04B4) はサイプレス セミコンダクタ社特定の VID で、この例では利用可能です。しかし、生産向けアプリケーションを開発する時には貴社に割り当てられた VID を使用しなければなりません。選択された PID はこのアプリケーションに一意のものです。ブートローダ ホスト アプリケーションはデバイスを認識するために VID と PID を使用します。

製品文字列や製造文字列などの文字列を自由に変更することができます。

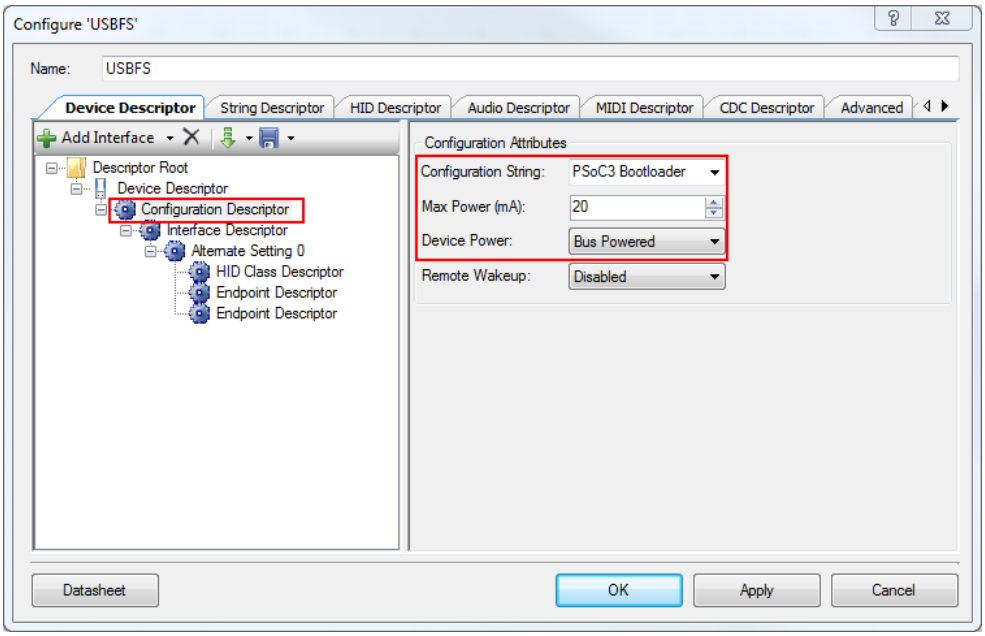
2. **String Descriptor** ツリーから、**Add String** をクリックします。図 16 に示すように、文字列を追加します:

図 16. String Descriptors



3. **Device Descriptor** ツリーで、**Configuration Descriptor** をクリックします。**Configuration Attributes** のオプションを図 17 に示すように設定してください:

図 17. USBFS コンフィギュレーション ディスクリプタ

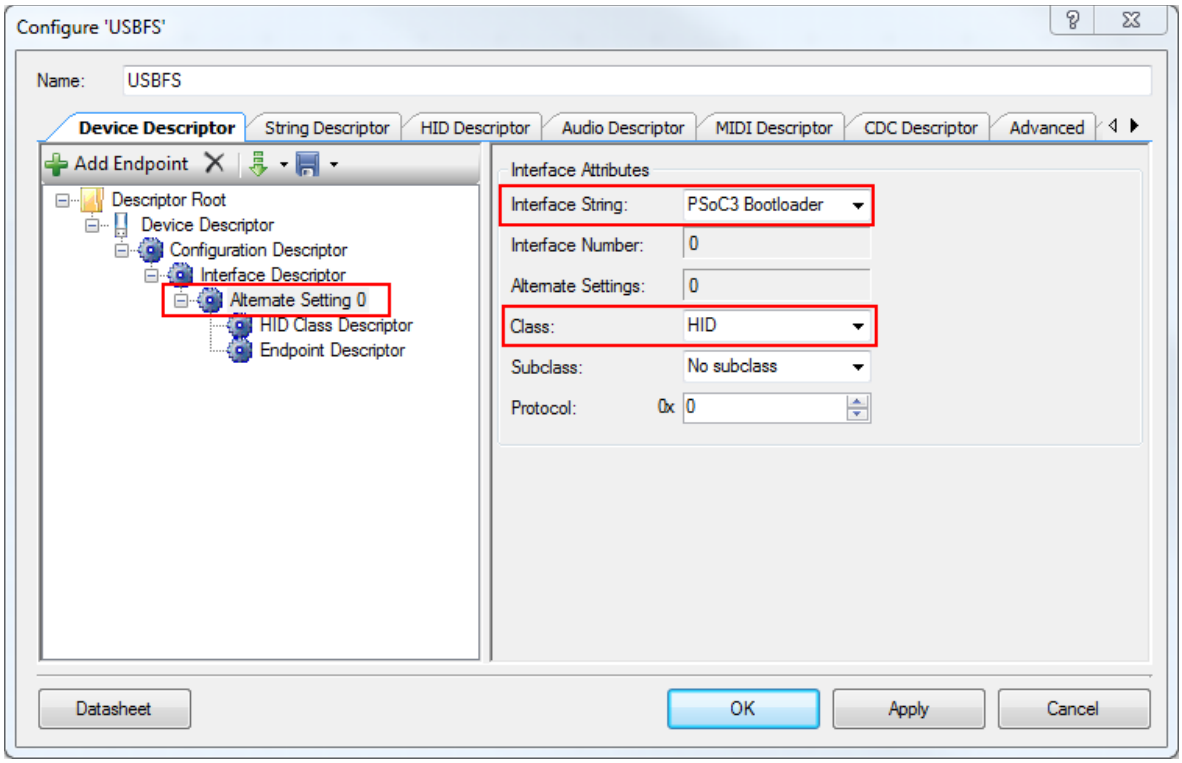


このステップでの主な設定は、バスにより給電されるように USB デバイスを定義し、ホストから 20mA の消費電力を要求することです。アプリケーションに 20mA 未満を供給できますが、この要件を超えてはいけません。

Remote Wakeup 機能は必要ないため無効 (disabled) にします。

4. **Interface Descriptor** ツリーで、**Alternate Setting 0** をクリックします。**Interface Attributes** を図 18 に示すように設定してください:
   - **Interface String**: PSoCx Bootloader
   - **Class**: HID
   - **Subclass**: No subclass

図 18. USBFS Alternate Setting



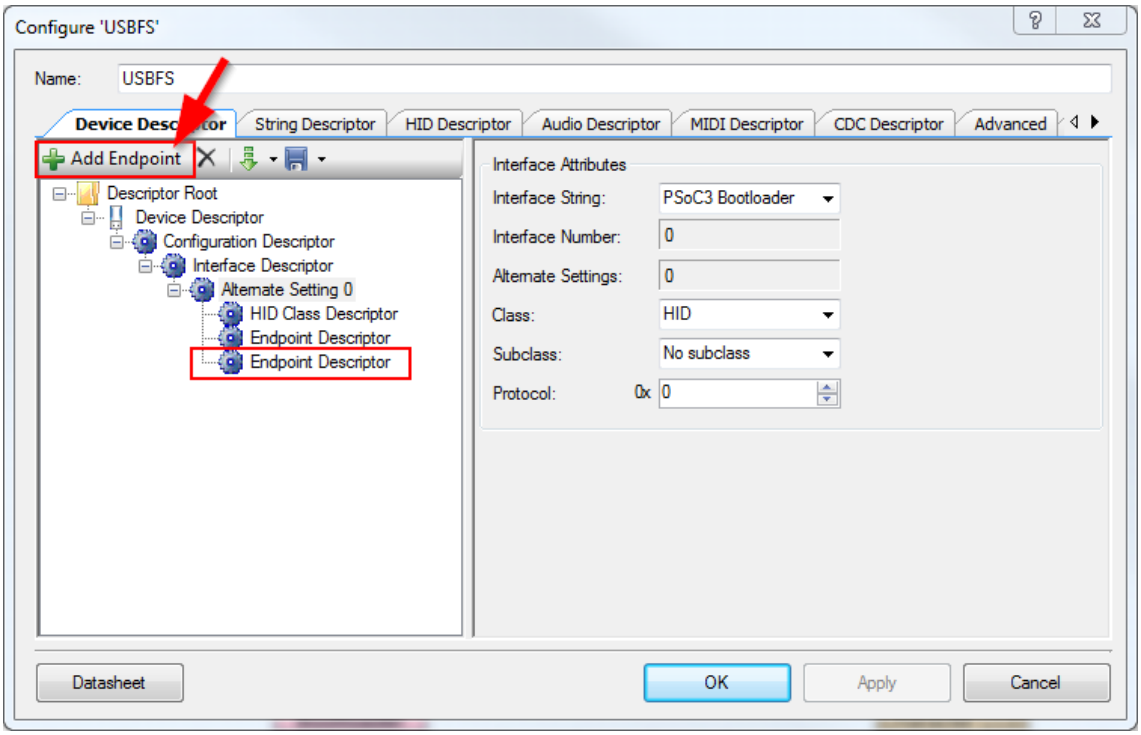
各インターフェースは、複数のエンドポイント コンフィギュレーションのために、複数の代替設定を持つことができます。ここでは、代替設定が 1 つのみ必要です (デフォルト)。

ここでデバイスが HID クラスに準拠することを指定します。Class を **Undefined** から **HID** に変更すると、Descriptor Root ツリーで「HID Class Descriptor」という追加ディスクリプタが出現します。

HID レポートをまだ設定していないことにご注意ください。この時、OK または Apply ボタンをクリックすると、HID レポートが定義されていないため、エラー メッセージが出ます。このレポートの作成は次に説明します。

5. **Add Endpoint** ボタンをクリックしてください。図 19 に示すように、**Alternate Setting 0** ツリーで追加の **Endpoint Descriptor** が出ます。

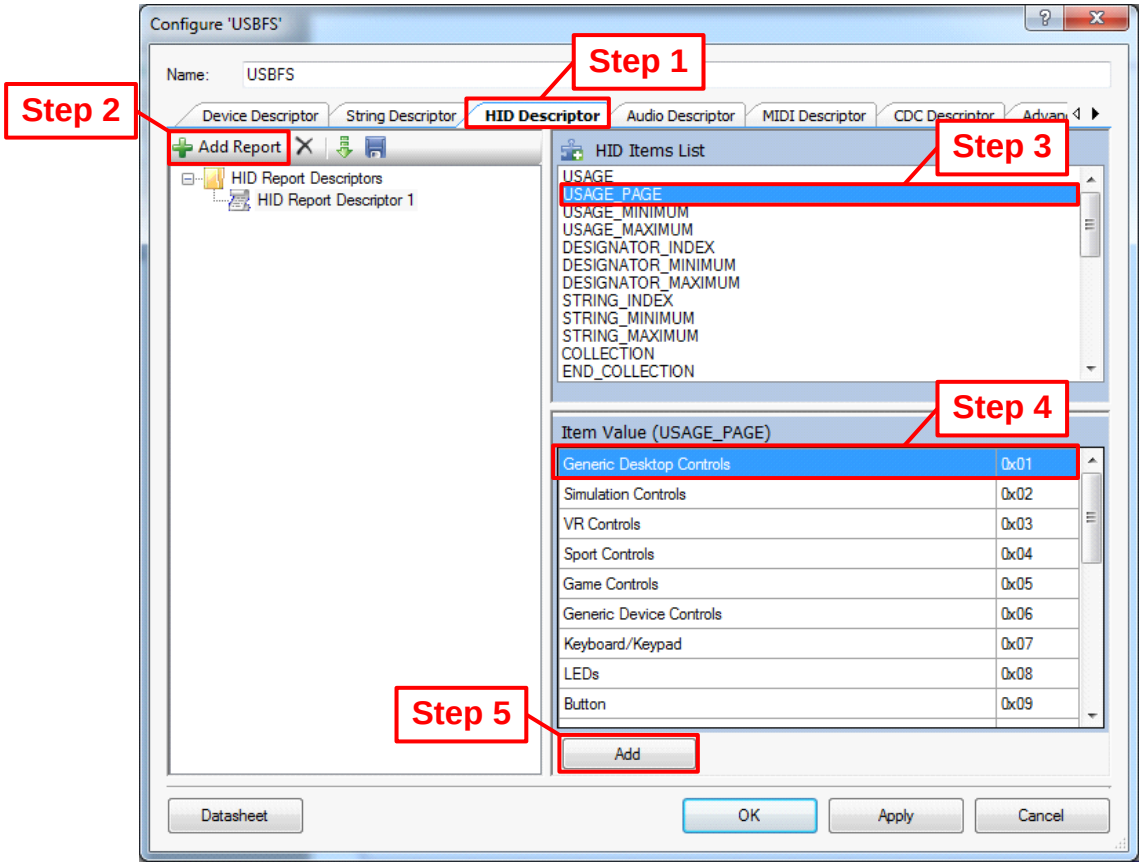
図 19. USBFS エンドポイントの追加



A.2 HID Descriptor の作成

1. 図 20 に示すように、HID レポート ディスクリプタを作成するために **HID Descriptor** インターフェースを使用します:

図 20. HID ディスクリプタ



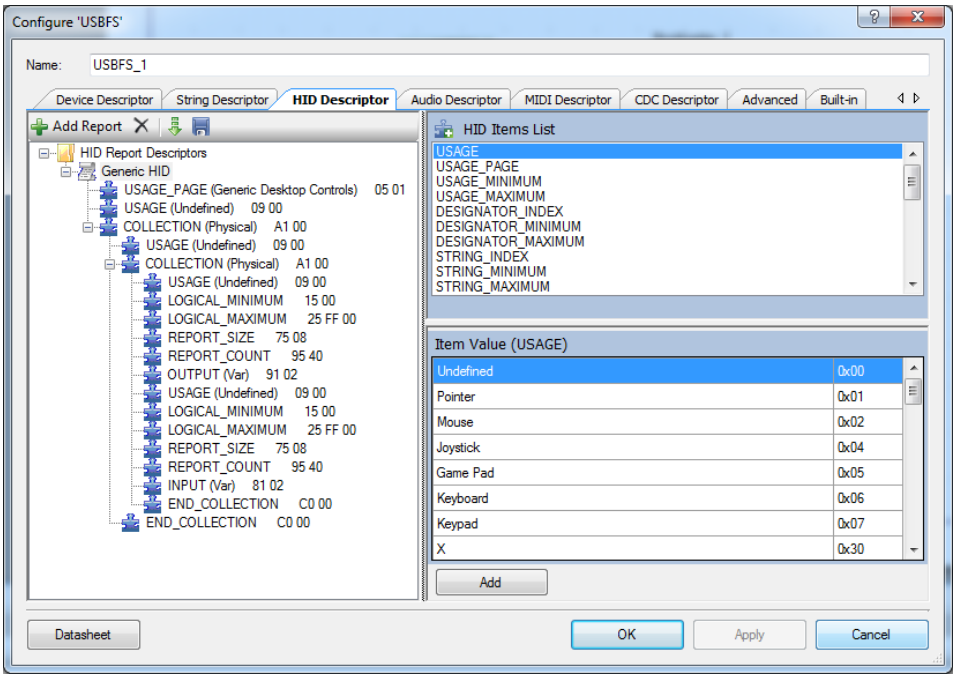
1. **HID Descriptor** タブをクリックします。
2. **Add Report** ボタンをクリックします。

   ステップ 3 からステップ 5 については、22 ページの表 1 をご参照ください。レポート項目は、表に示す順序で追加する必要があります。
3. **HID Item List** から一つの項目を選択します。
4. 表に示すように、**Item Value** リストから一つの値を選択します。
5. Item Value ボックスが特定の選択済み項目にテキスト フィールドを含む場合、10 進数または 16 進数のラジオ オプションをクリックして、フィールドに必要な値を入力します。
6. レポート ディスクリプタが 22 ページの図 21 のスクリーンショット画像のようになるまで、ステップ 3 からステップ 5 を繰り返します。
7. HID ディスクリプション名を「Generic HID」に変更します。

表 1. HID ディスクリプタ項目

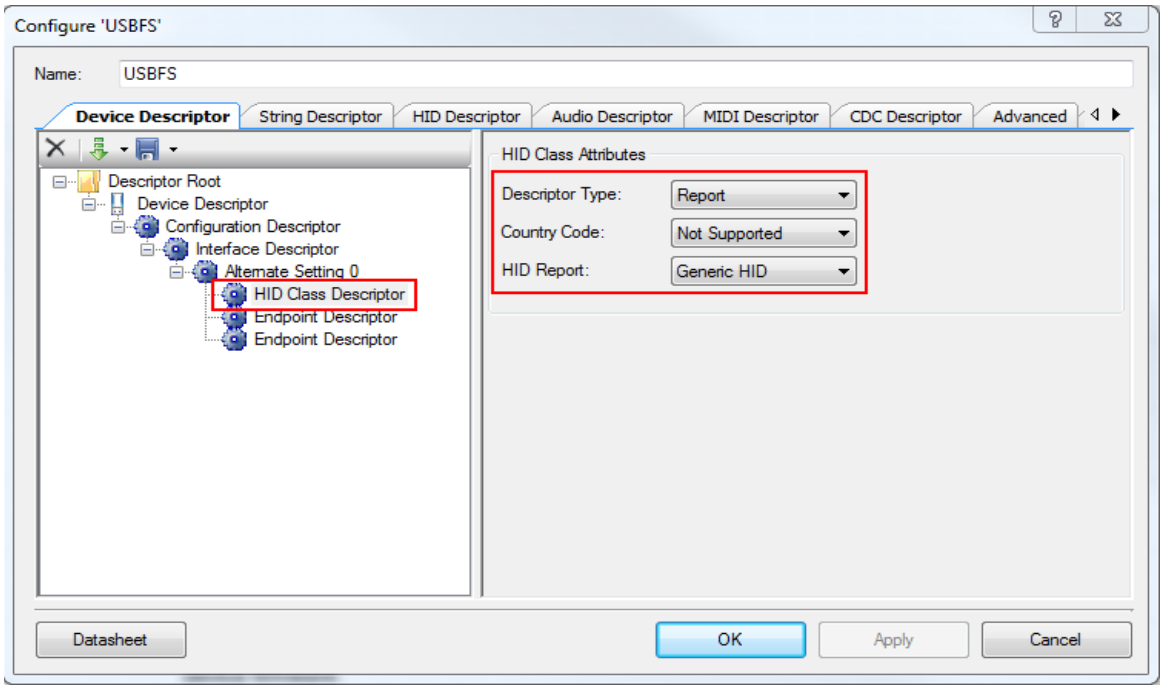
HID Item	Item Value (USAGE)
USAGE PAGE (05)	Generic Desktop Controls 0x01
USAGE (09)	Undefined 0x00
COLLECTION (A1)	Physical 0x00
USAGE (09)	Undefined 0x00
COLLECTION (A1)	Physical 0x00
USAGE (09)	Undefined 0x00
LOGICAL_MINIMUM (15)	0x00
LOGICAL_MAXIMUM (25)	0xFF
REPORT_SIZE (75)	0x08
REPORT_COUNT (95)	0x40
OUTPUT (91)	Bit 1 – Variable
USAGE (09)	Undefined 0x00
LOGICAL_MINIMUM (15)	0x00
LOGICAL_MAXIMUM (25)	0xFF
REPORT_SIZE (75)	0x08
REPORT_COUNT (95)	0x40
INPUT (81)	Bit 1 – Variable
END_COLLECTION (C0)	NA
END_COLLECTION (C0)	NA

図 21. 完成した HID ディスクリプタ



2. **Device Descriptor** タブをクリックします。**Descriptor** ツリーで、**HID クラス ディスクリプタ**をクリックします。図 22 に示すように、**Device Attributes** のオプションを設定します:
   - **Descriptor Type**: Report
   - **Country Code**: Not Supported
   - **HID Report**: Generic HID

図 22. USBFS HID デバイス属性



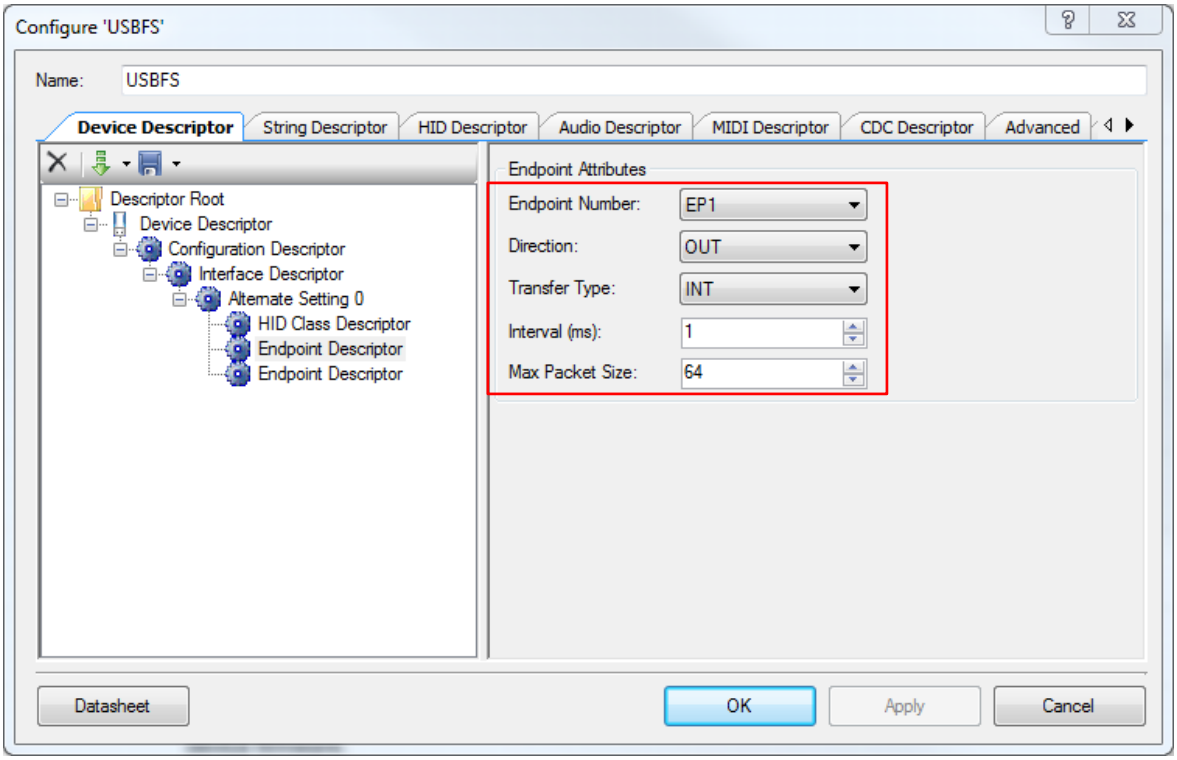
プロジェクトはレポート ディスクリプタを使用するため、ディスクリプタの種類を「Report」に設定します。

プロジェクトは特定の国で使用するものではないので、国コードを「Not Supported」にします。

最後に、HID クラス ディスクリプタは、前のステップで作成した HID レポート ディスクリプタを指す必要があります。このリンクを作るために、HID レポートを HID レポート ディスクリプタ (この例では Generic HID) に一致させます。

3. **Descriptor** ツリーで、最初の **Endpoint Descriptor** エントリをクリックします。図 23 に示すように、**Endpoint Attributes** のオプションを設定します:
   - **Endpoint Number**: EP1
   - **Diretion**: OUT
   - **Transfer Type**: INT
   - **Interval**: 10
   - **Max Packet Size**: 64

図 23. USBFS エンドポイント 1



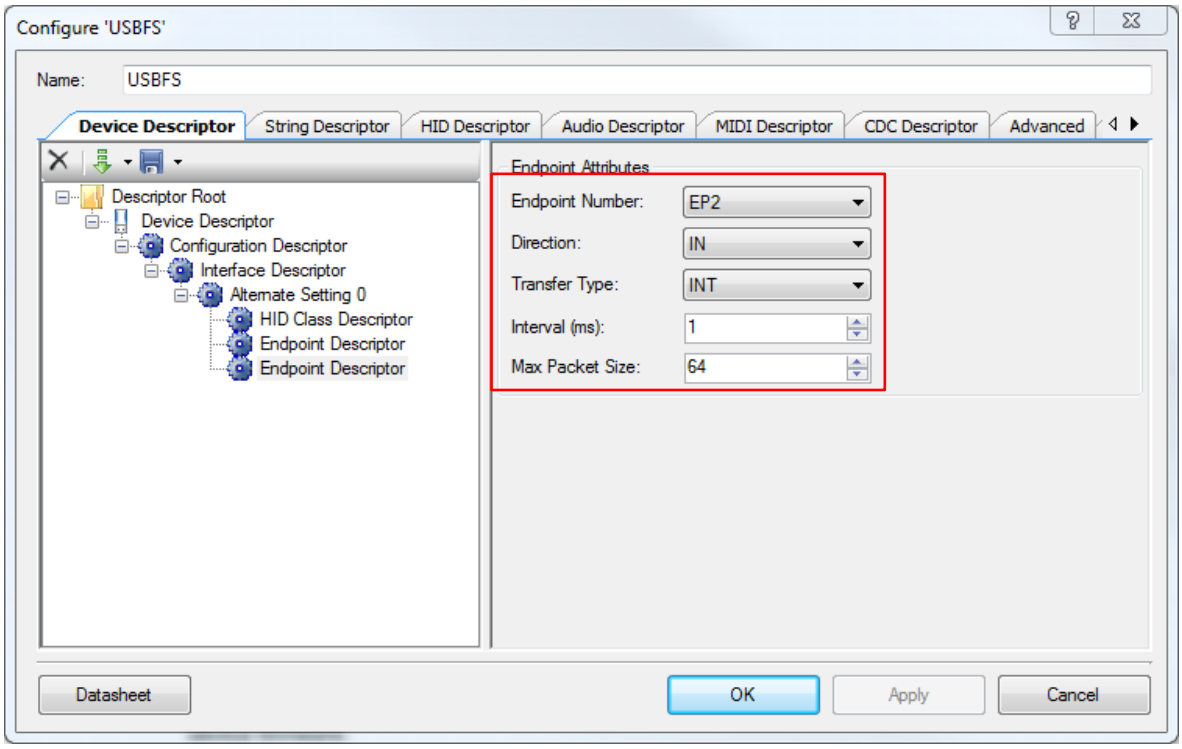
最初のエンドポイント (EP) は OUT エンドポイントとして使用されます。ホストから受信したデータのバッファの役割をします。USB IN (入力) と OUT (出力) の方向は、常に USB ホストから見た向きになります。

先ず、このエンドポイントをエンドポイント 1 (EP1) として定義します。次に、エンドポイントの方向を OUT に設定します。

アプリケーションが HID のため、割り込み (INT) 転送を使用する必要があります。

4. EP2 はホストに送信されるデータのバッファとして機能します。次のエンドポイントに対しても前述のステップを繰り返し、それを IN エンドポイントとして次の属性があるように設定します (図 24 を参照):
   - **Endpoint Number**: EP2
   - **Direction**: IN
   - **Transfer type**: INT
   - **Interval**: 10
   - **Max Packet Size**: 64

図 24. USBFS エンドポイント 2



5. **Apply** ボタンをクリックしてから **OK** ボタンをクリックし、コンフィギュレーション ウィザードを閉じます。

B 付録 B – ブートローダとデバイス リセット

このアプリケーション ノートで述べたように、ブートローダとブートローダブル間の制御の遷移は常にデバイス リセットで行われます。システムが極めて重要な機能を実行しながら、他のプログラムに変更する場合には、このことを考慮しなければなりません。このセクションでは、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のデバイス性能に影響を与えることなど、詳細に説明します。

B.1 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、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必要があります。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計を有します。そのため、一つのプロジェクトから他のプロジェクトに変更する際には全ての PSoC デバイスのハードウェア機能を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。特に、PSoC の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にビットを設定する他、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なければなりません。そうしないと、新しいコンフィギュレーションは動作しなくなり、デバイスを破壊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 プロジェクトの間で変更するとき、デバイス ソフトウェア リセット (SRES) を行います。これにより、全ての 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 レジスタは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低減できます。

B.2 PSoC 3 および PSoC 5LP デバイスの 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 3 and PSoC 5LP Startup Procedure」で説明した通りに、リセットおよび起動プロセスの実行時に PSoC 3 と PSoC 5LP の I/O ピンは表 2 に示すように、3 種類の駆動モードで動作します。PSoC 4 L シリーズの I/O ピンも同様です。

表 2. デバイス リセット中の PSoC 3 および PSoC 5LP 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 (標準値)		コメント
		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 リセットが解除される	HI-Z アナログ	40µs		リセットがアクティブの間、I/O が HI-Z アナログ モードに保持される
不揮発性ラッチ (NVL) が I/O ポートにコピーされる コードの実行が開始	NVL 設定: HI-Z アナログ、プルアップまたはプルダウン	約 12ms	約 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 ポートおよびピンは設定される	PSoC Creator プロジェクトのコンフィギュレーション	該当なし		8 つの可能な駆動モード。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
コードは main() に到達	コードは I/O ピンの機能を変更可能	該当なし		

PSoC 3 および PSoC 5LP での NVL 使用の詳細についてはデバイスの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。

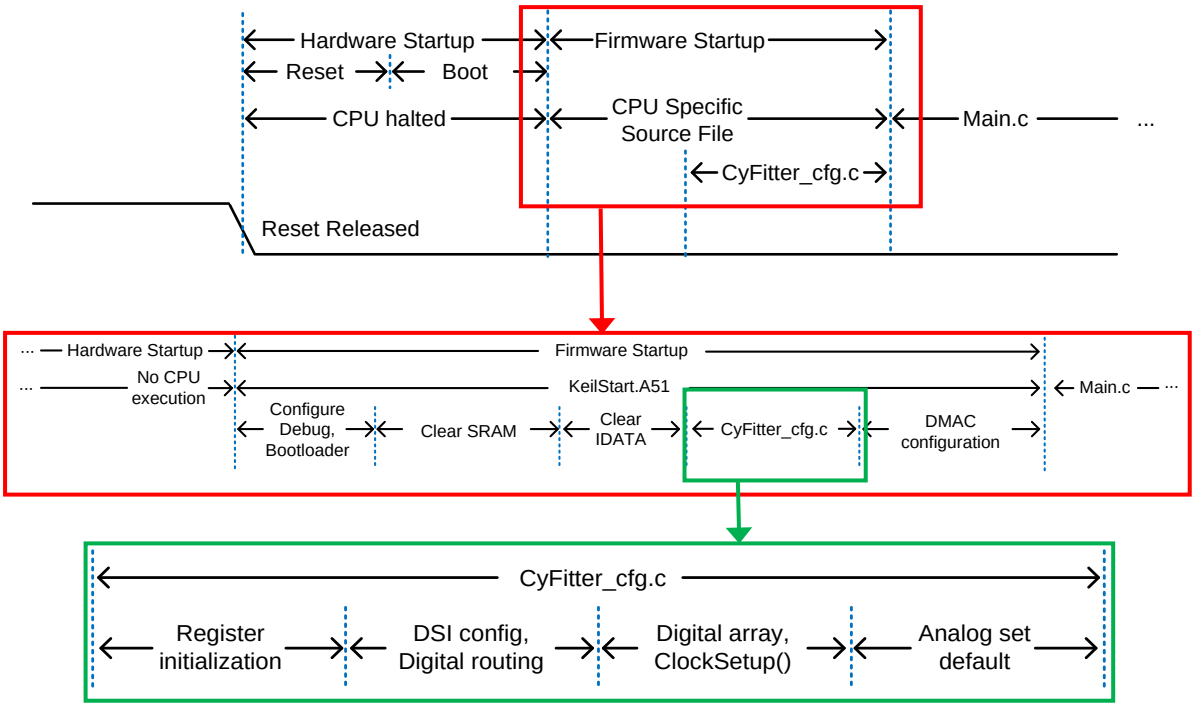
- I/O ポート用の **Reset** タブ、個別のピン コンポーネントのコンフィギュレーション
- 他のすべての NVL 用には **System** タブ、デザイン全体リソースのウィンドウ (DWR)

デバイスがプロジェクトを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 の設定は関連するブートローダ プロジェクト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します。

図 25 に、デバイスの起動およびコンフィギュレーション用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 L シリーズおよび 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 3 and PSoC 5LP Startup Procedure をご参照ください。

図 25. デバイスの起動プロセス図



B.3 他の機能への影響

デバイス リセットの時に、汎用デジタル ブロック (UDB) レジスタはリセットされたため、全ての UDB ベースのコンポーネントがなくなり、その機能が停止されます。同様なことは PSoC 3 および PSoC 5LP のコンフィギュレーション可能な SC/CT ブロック、および PSoC 4 L シリーズの CTBm ブロックに基づいたアナログ コンポーネントにも当てはまります。

すべての固定ペリフェラル (デジタルおよびアナログ ペリフェラル) がその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。IMO 以外の全てのクロックは停止します。

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。これにより、すべてのデジタルおよびアナログ スイッチが開いて、デバイス内のすべての接続を切ります。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 25 を参照) プロジェクトの main() 関数の実行を開始する時、すべてのファームウェアの機能が復元されます。

B.4 例: ファン制御

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てみましょう。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ウェブ 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 3 に示す通りに、ファンはデバイスがリセットしている間、ブートローダとブートローダブル間の転送中に動作し続けることができます。

表 3. 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	コメント
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。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
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。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
PSoC Creator のプロジェクトの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。PWM コンポーネントはアクティブになるが、動作しない
Main() の実行を開始	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。 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積極的に開始

C 付録 C – ホスト コア API

cybtldr_api2.c / .h

これは、ブートロード動作全体を処理する、より高いレベルの API です。ファイルを開閉する関数があります。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 ホストをビルドする時に使用されます。

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、列のプログラム、列の消去、列のベリファイ、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 4 は、この API の関数の詳細を説明します。

表 4.*cybtldr_api.c /.h* の関数

関数	説明
CyBtldr_StartBootloadOperation	• 通信インターフェースを有効にし、「ブートロード開始」コマンドをターゲットに送信します。 • 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します。
CyBtldr_ProgramRow	• 最初に列を確認します。つまり、ターゲットフラッシュの特定のアレイ ID のために、Get Flash Size コマンドをターゲットに送信します。これに応えて、ターゲットはそのアレイにあるブートローダブルのフラッシュ部の開始と終了の列番号を返します。ホストはこの応答を読み出し、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します。 • 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 コマンドでターゲットに送信します。 • 列データの最終部と共に、Program Row コマンドをターゲットに送信します。
CyBtldr_VerifyRow	• この関数は最初に特定のアレイ ID と列番号のために列を確認します。 • 列の確認が成功したら、確認されたフラッシュ列の Verify Row コマンドを送信します。このコマンドの応答で、ターゲットは列のチェックサムを返します。 • 返されたチェックサムは期待されるチェックサム値と比較されます。
CyBtldr_EraseRow	• この関数も最初に特定のアレイ ID と列番号のために列を確認します。 • 列の確認が成功したら、確認されたフラッシュ列の Erase Row コマンドを送信します。
CyBtldr_EndBootloadOperation	Exit Bootload コマンドを送信し、通信インターフェースを無効化します。

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 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関数もあります。

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_utils.h

このヘッダはブートローダのバージョン情報、およびステータスおよびエラーの定数を提供します。

改訂履歴

文書名: AN73503 - PSoC® USB HID ブートローダ

文書番号: 002-11097

版	ECN	変更者	発行日	変更内容
**	5165186	HZEN	03/08/2016	これは英語版 001-73503 Rev. *I を翻訳した日本語版 002-11097 Rev. **です。
*A	5827698	AESATMP8	07/21/2017	ロゴと著作権を更新しました。

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保持しています。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。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。