USBFS ブートローダ データシート BootLdrUSBFS V 1.30

リソース	PSoC® ブロック			API メモリ（バイト）		ピン（外部 I/O あたり）
	デジタル	アナログ CT	アナログ SC	Flash	RAM	
CY8C24x94、CY8CLED04、CY7C64215、CY8C20x66、CY8C20x36、CY8C20x46、CY8C20x96、CY7C643xx、CYONS2000、CYONS2110、CY8CTST120、CY8CTMG120、CY8CTMA120、CY8CTST200、CY8CTMG2xx						
HID サポート	0	0	0	4615-4982	46	2
HID サポートなし	0	0	0	4516-4982	46	2

Note インターフェース、HID クラス、その他の USBFS 拡張を追加すると Flash と RAM を多く消費することを考慮しておいてください。ブートローダが稼動しているときは、プログラムデータをダウンロードするために、より大きな RAM 領域を使用しますが、終了時にはそれを解放します。アプリケーションのオペレーションは、ブートローダのオペレーションとは独立していて、この RAM 要件は無視することが出来ます。ROM/Flash は、USB インターフェースに使用されます。ブートローダでは、USB そのものが通常使用する約 1.9K のコードサイズ用件に対して 2K バイト上回ります。

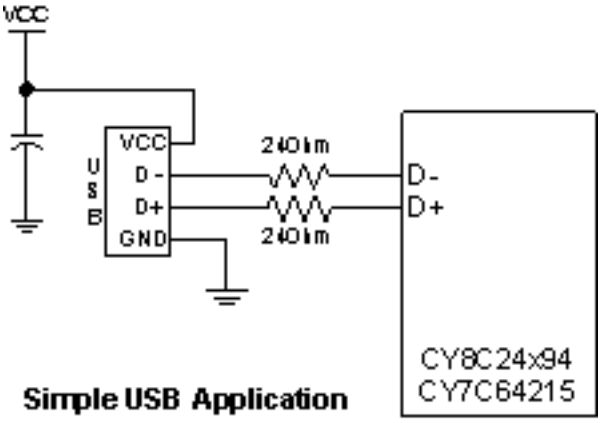
特徴と概要

- 柔軟なメモリマップ
- エンジニアリングツールを使用しないで、デバイスの再プログラミング
- 自身による再プログラミング
- コードのオーバーヘッドを低減するために統合された通信インターフェース
- フィールドにおけるファームウェア更新
- USB フルスピード デバイス インターフェース ドライバ
- インタラプトトランスファーとコントロールトランスファーのサポート
- デスクリプタを簡単かつ正確につくるためのセットアップウィザード
- デスクリプタ選択のためのランタイムサポート
- オプション :USB 文字列デスクリプタ
- オプション :USB HID クラスサポート

USB ブートローダは、インダストリースタンダードなコミュニケーションインターフェイスによってデバイスを再プログラムするための機能を持っています。また、エラーを検出するための機能ももっています。

複数の USB デバイス記述子はシステムに内在しており、実行中のデバイスが自ら再構成、再プログラミングするコマンドを発行できます。再構成中、ホストの通信をサポートするようコア USB は維持され、その間、プログラムデータの転送と保存が行われます。構成プロセスの最後に、デバイスは自動的にリセットし、新しいプログラムを検証した上で実行します。

Figure 1. USBFS ブートローダのブロック ダイアグラム



クイックスタート

1. この Data Sheet を確認します。ブートローダプロジェクトの実施を成功させるには、まずこの情報を理解することが必要です。
2. プロジェクトにユーザ モジュールを追加します。
3. ユーザ モジュールを配置し、HID または非 HID クラスのアプリケーションを選択します。
4. ユーザ モジュール アイコンを右クリックします。 **Boot Loader Tools** を選択します。 **Get Files** を選択します。終了すると、boot.tpl, custom.lkp、HTLinkOpts.lkp、flashsecurity.example のファイルがプロジェクトのルートディレクトリに配置されます。
5. ユーザ モジュール アイコンを右クリックします。選択します。 **Device:> > Application USB Setup Wizard… を選択します。Strings** エリアに少なくとも 1 つの文字列があることを確認します。デフォルトでは、少なくとも 1 つの文字列が存在しなければならず、存在しない場合は 1 文字の文字列を加えます。 **[OK]** を選択します。

   Note ステップ 3 で HID クラスアプリケーションを選択した場合、次のセットアップが必要です。HID アプリケーションでない場合、次のセットアップをスキップし、ステップ 6 に進みます。

   - **[Import HID Report Template]** をクリックします。
   - 3 button mouse template を選択します。
   - テンプレート右側の **Apply** をクリックします。
   - HID クラスデスクリプタの編集：3 ボタンマウスで [HID Report] フィールドを選択します。
   - **[OK]** をクリックして USB デスクリプタ情報を保存します。
6. ユーザ モジュール アイコンを右クリックします。 **Device> > Bootloader USB Setup Wizard… を選択します。**何も変更する必要はありません。 **[OK]** をクリックします。
7. (Image Craft コンパイラのみ) **Project > Setting > > Linker >** ダイアログボックスで、**Relocatable code start address** に ApplicationCode_Start_Block X デバイスブロックサイズを乗算して得られる値を設定します。これは、予期せずに、ブートローダの ROM 領域にアプリケーションコードを作成してしまわないようにするためです。この設定によって ROM 領域に未使用部分が残った場合には、設定を後で最適化できます。リンカは、メモリ重複エラーの場合、役に立たないメッセージを発行します。リンカの問題が最小限に抑えられれば、初期プロジェクト開発は、よりスムーズに進行できます。Relocatable Code Start Address を 0x1700 に設定すれば、ほとんどの場合適切なデフ

ォルト設定となります。CYONS2xxx の例（ブロックサイズが 0x80 バイト）: 0x2e*0x80 = 0x1700。CY8C24794 の例（ブロックサイズが 0x40 バイト）: 0x5c*0x40 = 0x1700。

8. Generate source code をし、プロジェクトをコンパイルします。
9. 出力ファイル <project>.mp <project 名 >.mp と <project>.hex <project 名 >.hex をチェックし、プロジェクトの構成を確認します。
10. エラーなしにコンパイル出来るプロジェクトが完成したら、[ファームウェアの例] セクションに進みます。例で示されているコードを変更・適用します。

機能説明

USBFS ユーザ モジュールには次のような特徴があります。

- USB フルスピードで Chapter 9 準拠のデバイスフレームワーク。
- USB ホストからの要求をデコード・発行する Control endpoint 用、低級ドライバ。
- デスクリプタを簡単に構成できるようにする USBFS セットアップウィザード。

HID ベースのデバイスまたは汎用 USB デバイスの作成を選択できます。これは、USBFS ユーザ モジュールを選定する際に選びます。最初の選択を変更するには、現在の USBFS ユーザ モジュールのインスタンスを削除し、新しいインスタンスを追加します。

ユーザ モジュールのブートローダ部分は、メモリマップと主要なコード機能ブロックを、デバイスリプログラミング可能な領域に配置する方法を提供します。プロジェクトのメモリ構成は、従来の PSoC デザイナプロジェクトと大幅に異なっています。デバイスアプリケーションをリプログラミング最中に最低限の機能をもったデバイスとして動作するためには、メモリマップの変更が必要です。特に、プロジェクトにブートローダを組み込んだ場合には、2 つのプログラムがそれぞれ異なる機能をサポートすることになります。メモリのマップを 示図 2 します。

ブートローダを統合したプロジェクトを実行すると、赤にハイライトされているメモリに配置されている部分はリプログラミングされることはありません。アプリケーションコードとチェックサムは、ブートローダの実行によって変更されます。プログラム空間の最初の 2 ブロック以外は、その他の主要なコード機能グループを、ユーザが指定する場所に移動できます。

ユーザ モジュール内で調整されたパラメータに加えて、2 つの重要な機能があります。デバイスマネージャビューのブートローダアイコンを右クリックすると、内蔵ツールセットを査定できます。さらに、ホストアプリケーション（ソースコードを含む）と、その設定方法、システム上での使用方法の説明もあり、ブートローダの機能を明示しています。

仕様、リソース例、フォーラムなどの USB に関する情報は、www.usb.org を参照してください。

動作の原理

ブートローダを用いてプロジェクトを作成するには、PSoC Designer の標準的なプロジェクトに対していくつかの変更が必要です。これを効率的に行うために、ブートローダユーザ モジュールはカスタイマズされたファイルと専用ツールを提供し、ブートローダープロジェクトの開発をサポートします。この特殊ツールは、Device Editor ビューに切り替え、USBFS ブートローダ ユーザーモジュールアイコンを右クリックすることでアクセスできます。ユーザ モジュールの一部として提供されるツールとファイルのほかに、ホストアプリケーションの例では、ブートローダの基本機能が理解できるユーザ モジュールの導入について説明されています。PC ベースのアプリケーションと Microsoft Visual Studio® 2005 のソースコードが、PSoC Programmer 3 のインストールディレクトリにある .zip ファイルに含まれています。

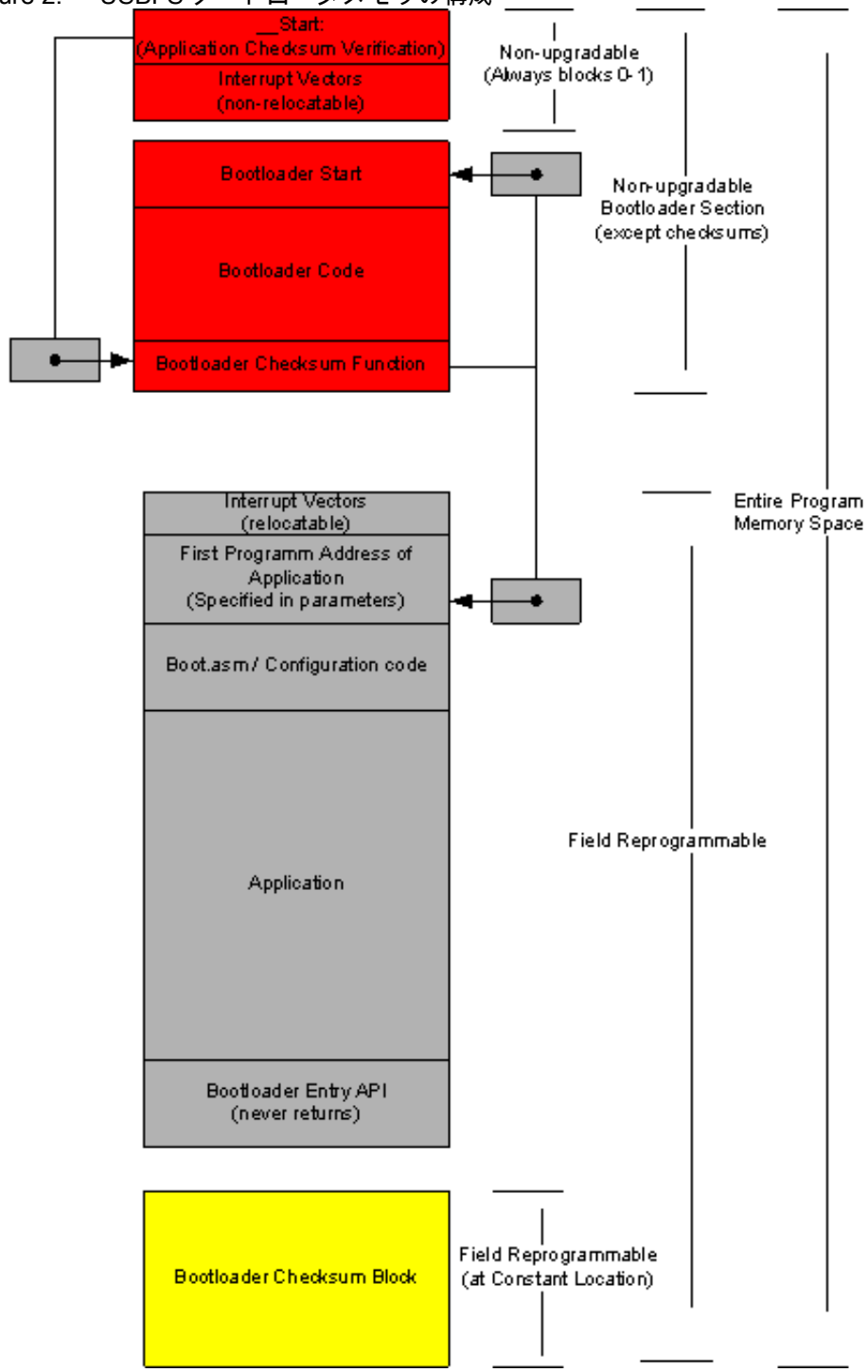
<install_path>\Cypress\Programmer\3.xx\Bootloaders\BootLdrUSBFS\USB_BootLoaderHostApp\...

ホストのデモアプリケーションを使用するには、汎用 USB ドライバのインストールとわずかなカスタマイズが必要です。このファイルは、インストールの一部として提供され、ブートローダデバイスを初回

動作時に登録されます。Windows の手動インストールを使用し、先に示した場所の 「\USB driver」ディレクトリに含まれるドライバの場所を指定します

ユーザが選択したブートローダデバイスの VID と PID をこのディレクトリに含まれる driver.inf ファイルを変更し正しく指定できるようにします。注：この変更は、driver.inf ファイル内のファイルはじめのに近い位置と最後近くの、2 つの場所に対してしなければなりません。

Figure 2. USBFS ブートローダメモリの構成



USBFS ブートローダメモリの構成

PSoC Designer ™は、標準化されたファイル、パーツファミリごとの Built in data、特定デバイスの属性を使用して、コンパイル可能なプロジェクトと API を作成します。ブートローダを用いたプロジェクトには、標準的 PSoC デザイナプロジェクトのものとは大幅に異なるメモリマップが必要です。メモリ領域の選定によりコアデザインが決定し、これはデザインが完了するまで維持されます。ブートローダの要件を持たないプロジェクトでは、単に、コンパイラとリンカが RAM や ROM を配置できます。ただし、新しいアプリケーションの読み込み中にプログラムがクラッシュしないように、ブートローダの RAM と ROM を特定の領域にグループ化しなければなりません。

示図 2 すメモリのレイアウトでは、次のように 5 つの主要な ROM の管理領域があります。

- 第 1 は、ブロック 0 と 1 の ROM です。これらのブロックには、重要な割込みベクタと再起動ベクタが含まれます。オペレーティングデバイスによりこれらのブロックへの読み取りアクセスを制御することは不可能に近いため、削除や再プログラミングされることはありません。ROM の最初の 2 ブロックは変更不可能で、別の位置に移動することもできません。
- 定義するべきメモリの第 2 の領域は、ブートローダコードそのものを含む領域です。この領域は、ROM の別の位置に移動することができます。ただし、ブートローダを含むプロジェクトやデバイスが展開された後は、この領域は再プログラミング不可となり、フィールド更新はできません。
- 第 3 のメモリ領域は、再配置可能な割込みテーブルです。このテーブルは、1 つまたは 2 つのブロックから構成されますが、これはデバイスのアーキテクチャによって異なります。この領域には、割込みと汎用ベクタが含まれ、ブートローダを使用して新しいアプリケーションを読み込むときに変更できる、割込みやコードエントリ用のジャンプテーブルを提供します。例えば、この領域には、アプリケーション開始アドレスを含むことができます。ブートローダはこのアドレスを使用して、電源オン時にチェックサムを検証したあと、新しいアプリケーションを再起動させます。この領域の位置はデザインする際に決定できます。アプリケーションとブートローダの実装後、この領域は書き換えることができますが、その位置を変更することはできません。この領域の特徴は、本セクションで後述されるチェックサム領域に似ています。
- 第 4 のメモリ領域は、アプリケーション領域です。アプリケーション領域には、条件に準拠した再プログラミングが可能な割込みベクタセットが含まれ、書き換え後はアクセス不可となります。実行中に実行コードが変更された場合、プログラムに問題が発生することが予測されます。この要件は、ブートロード中にすべてのアプリケーション割込みをオフにすることで、簡単に満たすことができます。ブートローダが起動している場合、これは自動的に行われます。割込みベクタに加え、アプリケーション領域にはデバイスブートコードの大部分と、すべてのフォアグラウンドランタイムアプリケーションが含まれます。
- 第 5 の領域は、チェックサム領域です。この領域には、ブートローダソフトウェアがフォアグラウンドアプリケーションのダウンロード· 検証に使用する重要なデータが含まれています。このチェックサム領域には、フォアグラウンドアプリケーションのブロックにおける開始アドレスとサイズが含まれます。チェックサムの最初の 2 バイトは、チェックサムブロックそのもののチェックサムで、最後の 2 バイトは、ランタイムアプリケーションのチェックサムです。チェックサムブロックの構成には、ブートローダが使用するスペースのほかに、ユーザがユーザデータを定義するためのスペースもあります。この構成は、C 言語の構造体で定義されており、これは、ブートローダユーティリティがブロック内で変更や再配置にならない限り、変更が可能です。

ブートロード中も含めて、常時作動していることが要求されるアプリケーションがある場合、ブートローダユーザ モジュールのデザインは、それに対応するようカスタマイズできます。これは、AREA ディレクティブを使用してブートローダの ROM 領域にコードを加えることにより行えます。ブートロードプロセス中にコードが使用する RAM はすべて、ブートローダが定義する RAM 領域に加えなければなりません。

ユーザ モジュールパラメータのメモリ領域の定義

USBFS ブートローダユーザ モジュールパラメータを使用すると、ROM に配置された主要なプログラム要素をカスタマイズできます。ユーザ モジュールのデフォルトは動作する初期設定を提供します。プロジェクトのコンパイルが完全に成功するまで、この設定をします。プロジェクトのコンパイルが完了したら、プログラムのメモリマップと .hex 出力ファイルを見て、プログラム構成の最適化の方法を検討します。パラメータを再構成し、手違いでメモリ領域に矛盾ができた場合、有効なメモリマップを見ないで正しい位置を特定することは難しいかもしれません。

ブートローダのユーティリティ

ブートローダユーザ モジュールには、ユーザーのフォアグラウンドアプリケーションと共存できる完全な機能を提供します。デバイスを再起動やリセットした場合、かならずブートローダユーティリティが起動します。システムの起動時に起動したブートローダは、フォアグラウンドアプリケーションの ROM 領域のチェックサムを計算することにより、フォアグラウンドアプリケーションを検証します。計算されたチェックサムは、チェックサムブロックに保存されているチェックサム（アプリケーションで作成）と比較されます。2 つのチェックサムが同じ場合、ブートローダユーティリティはフォアグラウンドアプリケーションの実行を許可します。2 つのチェックサムが異なる場合、ブートローダは待機ループに入り、ホストアプリケーションが有効なアプリケーションをダウンロードするのを待ちます。また自らの USB サブシステムも有効化し、ホストによるデータ転送を許可します。このインターフェースが有効化されたことをホストシステムが検知すると、ホストは自らのアプリケーションセットを実行する場合もあります。デフォルトの USB デスクリプタは、ここに示す例で実行が可能になっていますが、ホストもしくは PSoC デバイス上のパラメータを変更することも可能です。VisualStudio 2005 のソースコードがホストアプリケーション用に含まれています。アプリケーションとソースコードの例が、PSoCProgrammer3 のインストールディレクトリにはいっています。
<install_path>\Cypress\Programmer\3.xx\Bootloaders\BootLdrUSBFS\USB_BootLoaderHostApp\...

ブートローダツール

いくつかのツールはショートカットメニューで使用でき、ユーザ モジュールアイコンを右クリックすることでアクセスできます。

boot.tpl、custom.lkp、HTLinkOpts.lkp の特別バージョンは、プロジェクトに配置したり、削除したりできます。メインメニューで、Tools > Restore Default Boot files を選択します。USBFS ブートローダユーザ モジュールを削除すると、デフォルトブートファイル復元のオプションは、PSoC デザイナの File メニューに移動します。

チェックサムの作成 – プロジェクトが正しく構成できたら、ブートローダツールを使用してチェックサムを作成･ 自動検証できます。ブートローダツール選択の画面が表示されると、プロジェクトコードが作成され、プロジェクト全体のコンパイルが実行されます。次にチェックサム計算が行われ、その結果が hex ファイルに保存され、ユーザ モジュールに保存されているチェックサムと比較されます。チェックサムが適合しない場合、メッセージが表示されます。適宜、チェックサムを計算し、保存しなおすことができます。ブートローダツールから自動生成、自動ビルドされたコードからビルド、コンパイルエラーが発生し、hex ファイルが作成できない場合、エラーが報告されますが、PSoC Designer の構成ダイアログにはエラーデバック情報は表示されません。自動インターフェースにより生成とビルドが起動されると、エラー報告は抑制されます。デバッグ構成エラーでは、従来型の構成と作成プロセスをブートローダツールメニューの外で使用することが必要です。

dld ファイルの作成 – このツールアイテムは、hex プロジェクトの出力ファイルからダウンロードファイルを取り出します。このファイルには、チェックサムブロックを含めて、ブートローダによって再プログラミングされた hex ブロックだけが含まれます。ホストのデモアプリケーションは、このファイルを読み取り、ブートローダを使用している組み込みプロジェクトにダウンロードできます。このダウンロードファイルは、フィールドアプリケーションに実装して、PSoC デバイスの更新に使用できます。

チェックサム半自動作成

このプロジェクトがエラーなしにビルド・コンパイルされると、アプリケーションのチェックサムが作成されます。アプリケーションチェックサムは、Device Editor View の中で Bootloader User Module アイコンを右クリックして Bootloader Tools を選択することで利用できるユーティリティを使って、作成することが可能です。アプリケーションのチェックサム（事前計算済みまたはデフォルト）は、ユーザモジュールパラメータに隠されて保存されています。ブートローダツール メニューページが起動すると、以前のチェックサムは現在の出力 \<prj_name>.hex ファイルで計算されたチェックサムと比較検証されます。当然、コンパイルが正常に完了するまで、チェックサムの作成はできないことになります。チェックサムが作成されると、コンパイルされたファイルに統合されます。これにより、2 回目のコンパイルが必要になります。そして最後に、チェックサムが保存されているコードのブロックを検証するには、ブロックのチェックサム自体を提供します。これにより、3 回目のコンパイルが必要になります。チェックサムパラメータ中の構成はコンパイル後にのみ計算できる事前コンパイル工レメントなので、数回にわたるコンパイルが必要です。

特別ファイルの提供

[**Bootloader Tools**] メニューを開き [**Get Files**] を選択すると、いくつかの重要なファイルにアクセスできます。デバイス別 boot.tpl ファイルは、custom.lkp （ImageCraft）、HTLinkOpts.lkp（Hi Tech）という名称のファイル、および事前定義された flashsecurity.txt ファイルを含むメインプロジェクトディレクトリに配置されます。ここで各ファイルについて簡単に説明します（これらのファイルのオリジナル版はプロジェクトバックアップディレクトリに保存されています）。

Boot.tpl. – ファイルには再配置可能なものと、再配置不可能なインタラプトベクタテーブル及びデバイス固有のブートセットアップが含まれており、標準の boot.tpl に記述されている固定位置ではなく再配置可能な ROM の領域が指定されています。

Custom.lkp – ソースの作成が行われると、ユーザ モジュールパラメータに定義されているとおり、自動作成された主要なコードブロック用 ROM 領域に custom.lkp ファイルが配置されます。custom.lkp ファイルで、下記の名称のコードブロックは変更しないでください。

- -bSSCParmBlk – Flash 操作で使用される重要な指定 RAM を含んでいます。
- -bBootloader
- -bBLChecksum
- -bUserAPP – 最後の 3 行に変更があるとエラーダイアログが表示され、正しい custom.lkp を検知するプロジェクトの機能が無効になっていることを示します。

コード作成中は、custom.lkp ファイルの最後の 3 行が、1 行ずつ、コード作成ソフトウェア制御の下で書き換えられます。最後の 3 行に変更があると、エラーが発生するか、機能が停止します。custom.lkp ファイルのその他の場所には変更を加えることができます。プロジェクトのメモリ配置をデバッグするには、最初のスペースにセミコロンを挿入して、前述の 3 行をコメント化します。これにより、リンカがコードを自動的に配置します。また、アプリケーションのコードサイズの特定に役立つ場合もあります。

HTLinkOpts.lkp – ソースの作成が行われると、ユーザ モジュールパラメータに定義されているとおり、自動作成された主要なコードブロック用 ROM 領域に HTLinkOpts.lkp ファイルが配置されます。HTLinkOpts.lkp ファイルで、下記の名称のコードブロックは変更しないでください。

- -L-ACODE... & -L-AROM... 行には ROM 全体のサイズを提示するデータが含まれます。
- -L-PPD_startup... には、ブートローダ特有の ROM 領域を検知するリンカ指令が含まれます。
- -L-P

- -L-Pbss0= 最後の数行に変更があるとエラーダイアログが表示され、正しい HTLinkOpts.lkp を検知するプロジェクトの機能が無効になっていることを示します。

コード作成中は、HTLinkOpts.lkp ファイルの最後の数行が、コード作成ソフトウェア制御の下で書き換えられます。最後の 3 行またはそれ以前に変更があると、エラーが発生するか、機能が停止します。

Flashsecurity.example – これは、デフォルトのユーザ モジュールパラメータで指定されたデフォルトメモリマップに準拠して作成されたデフォルトファイルです。このファイルはプロジェクトの最後の作成するもので、場合によっては、実装されたデバイスとファームウェアに適合した最終メモリマップとアプリケーションサイズに準拠して、手動で変更しなければなりません。注：このファイルは、PSoC デザイナで直接使用するものではありません。何らかの理由で、プロジェクトがデータ外ファイルを用いて更新されたり、タグ付けされたりしている場合、flashsecurity ファイルは上書きされません。このファイルを編集し、名前を flashsecurity.example に変更することは可能です。

Flashsecurity.txt – これは PSoC デザイナが提供するデフォルトのファイルです。このファイルのデータは .hex ファイルに加えられ、内部 ROM メモリへのアクセス管理方法をデバイスに指示します。メモリブロックが書き込みアクセスから保護されている場合、ブートローダは機能しません。プログラミングされた PSoC には読み込みと書き込み保護がかかっているため、このファイルはブートローダの最初の実施前に正しく構成しなければなりません。

USB　デスクリプタ

標準的な USBFS ユーザ モジュールには、実行時間アプリケーションで使用される USB 記述子を開発するツールが統合されています。ブートローダは、デフォルト USB_Bootloader の記述子の開発または変更を可能にするもう一つのツールです。これらの 2 つの記述子は、ROM の異なる領域に保存されています。フォアグラウンドアプリケーション用の記述子は、アプリケーションを用いて更新する必要がある場合があります。USB_Bootloader 記述子は、ブートローダの ROM 領域の一部で、フィールドで更新できます。コア機能を維持するために、主要 USB コードもブートローダ ROM 領域に配置されます。これは上書き（よいプログラミングの手法とはいえません）される実行コードの問題を解決することが目的です。

自己電源供給デバイスに対する USB 準拠

*USB 準拠チェックリスト*に、「V_{BUS} が " 高 " のとき、デバイスのプルアップは " アクティブのみ " ですか？」という質問があります。

この質問は、*ユニバーサルシリアルバス仕様リビジョン 2.0* のセクション 7.1.5 に参考として記載されています。このセクションの一部には、「V_{BUS} が取り外されたときに、プルアップ抵抗器が接続されているデータラインに電流を供給しないように、プルアップ抵抗器の電源は USB ケーブルから得るか、または制御されなければなりません。」と説明されています。

作成中のデバイスが自己電源供給型である場合、抵抗膜式ネットワークを通じて、GPIO ピンを V_{BUS} に接続し、GPIO のステータスをモニタするファームウェアを書く必要があります。アプリケーション ノート AN15813*EZ-USB FX2LP VBUS をモニタする*では、必要なハードウェアとソフトウェアが説明されています。D+ ラインのプルアップ抵抗器を制御するには、USB IO 制御レジスタ 1 を使用します (USBIO_CR1) 。

ブートローダ VID および PID

UBS デバイス実装の最終段階では、ベンダ ID と製品 ID を割当てます。これらの割り当ては、USB 開発者からの依頼に応えて、USB 標準組織によって割り当てられます。開発の目的では、ホスト上の既存の VID と PID と矛盾しない VID や PID がプロジェクトのデバックに使用できます。しかし、プロジェクトリリースや開発の目的には、サイプレスに割り当てられている VID や PID を使用してはなりません。

パラメータのブロックエントリ

メモリパラメータはすべて、16K デバイスでは 0x00 ~ 0xFF、32K デバイスでは 0x00 ~ 0x1FF の番号が付いているブロックでブートローダに入力されます。これはメモリアドレスに入力するには最も便利なフォーマットではありませんが、部分ブロックアドレスがメモリマップの様々なセクションに間違って割り当てられるのを防ぐことができます。ここに該当する PSoC デバイスは、64 バイトの Flash ブロックの保存だけができ (20x45 では 128 バイト)、これがプロジェクトコードの異なるセクション間の境界線を簡単に維持できる方法です。

ホストアプリケーションのデバッグ

ブートローダが内蔵されているアプリケーションは、デバッグが困難な場合があります。このため、ブートローダユーザ モジュールファイル内で追加調整できるものがあります。これらはファイル BootLdrUSBFS_Bt_loader.inc に含まれています。あるセクションには次の式が含まれています。

```
BOOT_TIMEOUT:           EQU    40   ;set to zero to make timeout infinite
CHECKSUM_ON_CKSUMBLK:   EQU     1   ;Apply a checksum to the checksum block
                                   ;(adds compile steps and code to verify)
```

この BOOT_TIMEOUT 式を使用すると、ユーザコマンドがブートローダを呼び出した後ホストから何もこない場合、タイムアウトする無限コードを延長、短縮、作成できます。これはホストアプリケーションの開発またはデバッグで役立つことがあります。

第 2 の式は、チェックサムブロック内でチェックサムの使用を制御します。この式が 0 にセットされている場合、チェックサムブロック内のチェックサムには検証が行われません。ただし、ユーザ モジュールパラメータで指定されているとおり、ユーザアプリケーション領域全体のチェックサム検証は行われます。

タイミング

USBFS ユーザ モジュールは、CY8C24x94、CY8CLED04、CY7C64215、CY8C20xx6、CY7C643xx、CYONS2000、CYONS2100、CYONS2110 デバイス上の USB 2.0 フルスピード動作をサポートします。

USBFS セットアップウィザード

USBFS ブートローダユーザ モジュールは、個別化のための PSoC デザイナ パラメータ グリッド表示は使用せず、アプリケーション用に USB デスクリプタを指定するフォーム方式の USBFS セットアップウィザードを使用します。デスクリプタによって、ウィザードはユーザ モジュールを個別化します。

ユーザ モジュールは、USBFS セットアップウィザードで作成された情報によって操作されます。ウィザードは、USB 記述子の構造を促進し、生成された情報をデバイス エニュメレーションに使用されるドライブファームウェアに統合します。USBFS ブートローダユーザ モジュールは、まずウィザードを実行し、適切な属性を選択し、コードを作成しなければ機能しません。

アドレスの代用としてブロックを使用

大部分の PSoC パーツは 64 バイトのブロックサイズです。最近では、128 バイトブロックの PSoC パーツも一部で導入されてきています。ここでは次のような 2 つの重要なファクトがあります。ブートローダは Flash に書き入むことが必要で、PSoC はブロック別にのみ Flash 書き入みができます。そのため、ブートローダアプリケーションにとっては、メモリを書き込みされる 「ブロック」のグループと考えるほうが便利です。

ブロックから絶対アドレスに変換するには、次の乗算を行います。Abs_addr = block_number X ブロックサイズ。Block_0 は addr 0 から開始し、Block_n は n x Block_size. から開始します。すべてのブロックはブートローダパラメータ用に 16 進数で区切られており、16 進アドレスは 0x40 (64-byte ブロック) または 0x80 (128-byte ブロック) の乗算によって求められます。

16 進数の出力ファイルには、各行の絶対アドレスが含まれています。該当するデバイスのブロックサイズ（0x40/0x80）に関わらず、16 進数の出力ファイルでは、コードを各行 64(d)/0x40 バイトに分割します。従って、各行の 64byte ブロックのデバイスは、コードのブロックを意味します。128 バイトのブロックデバイスでは、hex ファイルから 2 行が 1 ブロックに挿入されます（ブロック 0 はアドレス 0 で開始し、128 バイトのブロックは常に、下位（アドレス）半分を示す「偶数」半分と、上位（アドレス）半分を示す「奇数」半分を持つと考えられます）。

hex ファイルを見て、作業中のパーツの Flash ブロックサイズに慣れてください。

よくある問題

このセクションでは、ブートローダプロジェクトを作成する際によく発生する問題と、その対処方法について説明します。

ブートローダプロジェクトの更新、サービスパックのアップグレードとコンパイラ

ブートローダアプリケーションを使用している間は、PSoC 開発者環境を変更することは避けてください。これは PSoC デザイナ、ブートローダユーザ モジュール、コンパイラを更新しないことも含みます。

この理由を理解するには、ブートローダとアプリケーションはもともと一緒にコンパイルされましたが、ブートロード可能なシステムの実装後はアプリケーションセクションだけが再プログラミングされたということを理解しておく必要があります。新しいアプリケーションや改定アプリケーションは、もともとの実装のブートローダと一致するために、ブートローダユーザ モジュールの同一バージョンでコンパイルする必要があります。開発環境におけるすべてのエレメントのバージョンが適合することが理想ですが、ブートローダの場合、適合性を維持することは必須です。開発環境を変えなければ、互換性のリスクは取り除かれます。

USB ベースのブートローダは、API として USB サブシステムをアプリケーションに提示します。これは、コードサイズを縮小することが目的です。これらの機能の提示は、リダイレクトされた呼び出しテーブルを介しては行われません。この戦略の意義は、アプリケーションがブートローダ内の特定アドレスにダイレクト呼び出しを発行するということです。ブートローダとアプリケーションは一緒にコンパイルされるため、USB API 機能のアドレス変更を生じるブートローダへの変更は、ブートローダのその他のバージョンでは非適合という結果を招きます。

PSoC デザイナでは複数のコンパイラがサポートされていますが、1 つのコンパイラでコンパイルしたブートローダが、他のコンパイラでコンパイルしたアプリケーションとも適合すると想定すべきではありません。RAM 割り当てに関する仮定に関して重要な違いが一つあります。RAM ページングの実施は、コンパイラによって異なる可能性があります。特に困難なのは、ブートローダとアプリケーションが一緒にコンパイルされるため、使用された開発ツールで不適合があったブートローダ / アプリケーションの組み合わせをデバッグすることは不可能だということです。

ウォッチドッグタイマの内部使用

ウォッチドッグタイマとの調整は、グローバル パラメータ WATCHDOG_ENABLE と関連付けられており、ファイル globalparams.inc に含まれています。プロジェクトでウォッチドッグタイマを使用する場合、グローバル パラメータに関連付けられているコードを条件付きでコンパイルし、ブートロードチェックサムとダウンロード操作の間に、ウォッチドッグを自動的に設定します。CPU クロック速度は、ウォッチドッグタイマの更新速度に影響を及ぼします。ウォッチドッグタイマの現実的な最小設定は約 0.125 秒です。

Flashsecurity.txt における不適切な設定

このファイルのデフォルト設定は、プロジェクトが作成された際にセットされます。構成例はファイル「Flashsecurity.example」に記載されています。Flashsecurity.example は、[ブートローダツール] - [フ

ァイルの取得] ユーザ モジュールメニュー項目にあります。マップにより、ブートロードされたすべての位置で Flash 書き込みが可能になります。1 つの方法は、すべてのブロックを書き込み可能にすることです。また別の方法は、まずメモリマップをレイアウトし、それに合わせてファイルを編集することです。どちらの方法を選んでも、プロジェクトの最初でアクションを起こすほうが、あとでデバックするよりも簡単です。ブートローダ実行可能ファイルで使用されるコードの領域は、書き込み保護しなければなりません。Flash セキュリティを正しくマッピングできない場合、システムが機能不全となり、非常に困難なデバッグ作業を強いられることとなります。

不適切な再配置可能コード開始アドレス（リンカパラメータ ImageCraft コンパイラのみ）

ブートローダプロジェクト用のメモリマップは、従来のプロジェクト用のメモリマップと大幅に異なるため、通常、再配置可能なコードの開始アドレスも変更する必要があります。これは、同じアドレスに 1 つ以上のブロックコードを書き込もうと試みたときに、リンカによって生成されるエラーの一般的な原因となります。このパラメータは、[プロジェクト] > [設定] > [リンカ] タブでファイルされた再配置コード開始アドレスで変更できます。絶対 16 進数開始アドレスをブートローダコードが使用する最高ブロックよりわずかに大きくなるように、または ROM の未使用領域を使用するように計算します。ブートローダの USB バージョンでは、この値に、パラメータを乗算して得た答えを設定します。

ApplicationCode_Start_Block X ブロックサイズ = 再配置コード開始アドレス

この値は、適宜、実際の計算結果より大きい値を説明することができます。一部のデバイスでは、ブロックサイズが 0x80 バイトや 0x40 バイトのこともあります。

メモリの重複

再配置可能コード開始アドレス（前セクションを参照）を修正するには、先行するセミコロンを使用して、custom.lkp ファイルの最後の 3 行コメントアウトし、ファイルの構成をもう一度試み、結果として生じたメモリマップを検証します。メモリ重複は、出力ファイルの生成を妨げるため、診断が難しい問題です。custom.lkp ファイルの変更により、リンカが目標ブロックを配置できるようになる場合もあります。これにより、メモリ重複の原因を是正する起点が得られます。

電力安定化

Flash プログラミングで診断が困難な問題が発生する原因には、電源雑音、グリッチ、電圧低下、遅い電源の立ち上がり、接続の不安定などがあります。出力急昇と比較するとプログラムの実行は速く、場合によっては、Flash プログラミングが実行されていてもまだ電力レベルが変化していることもあります。一例として、電源投入時の Flash への書き込みステータスの種類があります。使用中のモデルと、Flash 動作中に電源の状態を変える可能性を評価します。電源安定性が低いと、パーツの機能不全につながり、Flash 維持力がそこなわれる可能性があります。

ブートロードプロセスで完全に停止していないアプリケーションや割込み

新しいブートロードアプリケーションに置き換えられるアプリケーションは、ブートロード操作が実行される前に完全に終了しなければなりません。アプリケーションの割込みをオフにすることは特に重要です。ブートロードプロセスが実行されるとき、割込みベクタアドレスは書き換えられる前に、ゼロに変更されます。これが無効になっていないと、割込みの実行で無作為なリセット（ベクタを 0）が発生します。注：これはブートローダで使用される特定の通信割込みには適用されません。

USB インターフェースには 2 種類あり、1 つはアプリケーションが、もう 1 つはブートローダが使用します。USB アプリケーションインターフェースをシャットダウンし、ブートローダのインターフェースをオンにする手段を講じる必要があります。これには実行中のアプリケーションを完全にシャットダウンすることも含まれます。本 Data Sheet で後述すサンプルコードには、この手順の例が説明されています。

デバイスの作動を停止する新ファイルのダウンロード

ブートローダユーティリティを開始する手段がなくとも、アプリケーションを構成することは可能です。意図せずにこれをしてしまうことはよくあります。例えば、単純な while(1); を持つ main() 機能では、ループから返らず、ブートローダに入ります。その結果、実行開始後は再プログラミングできません（チェックサムが正しい場合）。この問題の対処策はいくつかありますが、このユーザ モジュールにはデフォルトの手段は記載されていません。推奨される手段の例：

1. デバイスの起動時にブートローダが有効な場合、一定の時間を許可するリセット条件を適用します。タイムアウトパラメータを設定することにより、リセット時にブートローダを開始し、タイムアウトが時間切れになるとフォアグラウンドアプリケーションを終了するように、デバイスを構成できます。
2. デバイスがブートローダを開始するように、コードのある時点でテストを設定します。これはスイッチのクロージャであったり、ポートピンを Low,High に維持することもできます。
3. 機能のレポートやスペアエンドポイントなどの USB 機能の構成を使用して、ブートロードモードを開始するために、デバイスへ送信できる USB 通信を定義します。このコマンドが送信されると、デバイスは USB バスを一時的に停止し、戻ったときにはブートローダとなっています。
4. 定期的な点検を受けていない場合、ウォッチドッグタイマを使用してデバイスをリセットできます。これは上記の手段のうち 1 つと組み合わせて、ウォッチドッグタイマ割込みによってブートロード可能な状態を作ることができます。ウォッチドッグタイマをリセットすると、ウォッチドッグタイマと関連するステータスビットをモニタし、それがリセット条件の発生原因であるかどうかを特定します。詳細については、Technical Reference Manual を参照してください。
5. 2 つのプロジェクトは開発された場合、各プロジェクトのブートローダは細かいところで異なります。「ブートロード」とは、デバイスのプログラミングパーツ部分が実行されることを意味します。これは、2 つの手動で再プログラミングされたアプリケーションのブートローダ実装部分は同一であるということです。すべてのブートローダパラメータと再配置可能コード開始アドレスも、同一です（これは第 1 のアプリケーションブロックとは異なります）。このプログラムのデバッグ手段の 1 つは、ブートロードが使用する 16 進数コードの領域に特に注意を払いながら、該当する 2 つの 16 進数ファイルを比較することです。また別の手段は <project>.lst ファイルを比較することです。ブートローダは、リダイレクトベクタの一部を使用して、特定のアプリケーションアドレスパラメータの変更を許可します。これらのジャンプベクタはすべて、アプリケーションとブートローダで適合しなければなりません。ブートローダをフィールドアプリケーション用に実装すると、そのコードを変えることはできません。将来のアプリケーションも、相互に使用するジャンプベクタが保存されている場所については、適合していなければなりません。

パラメータおよびリソース

USBFS ブートローダは、完全に機能する USBFS ユーザ モジュールで統合されたブートローダユーティリティで構成されます。

ブートローダ用に定義されたパラメータによって、プログラムがコンパイルされ、関連付けられた際に、主要なプログラム領域がどこであるかが特定できます。

ユーザ モジュール名の変更

このユーザ モジュールはソースファイルの記述および / または上書きをするのにウィザードの使用が必要なため、ユーザ モジュール名を変更するにはユーザによるアクションが必要な場合があります。ウィザードにより生成されたファイル内のウィザードが生成した変数名は、すべてのケースを更新することはできません。内部変数名を再生成するには、ウィザードを 1 つずつ開き、[OK] または [適用] を選択します。変数名が更新されないためにコンパイルエラーが引き続き発生する場合は、プログラムから問題のファイルを削除し、ウィザードを再び開き、[OK] または [適用] を選択してプロジェクトを再構成します。このファイルは更新された変数名で置き換えられます。

デフォルトパラメータ

デフォルトパラメータは参照のみを目的としています。プロジェクトのデフォルトは、使用中のパーツのブロックサイズやコード領域の十分なサイズに適合させるために変更することがあります。プロジェクトのコンパイルとテストが完了したら、開発者はメモリ使用を最適化するためにブロックサイズの調整を選ぶことができます。

Figure 3. 0x80 (128 バイト) ブロックのデバイス用デフォルトパラメータ、HID 選択オプション

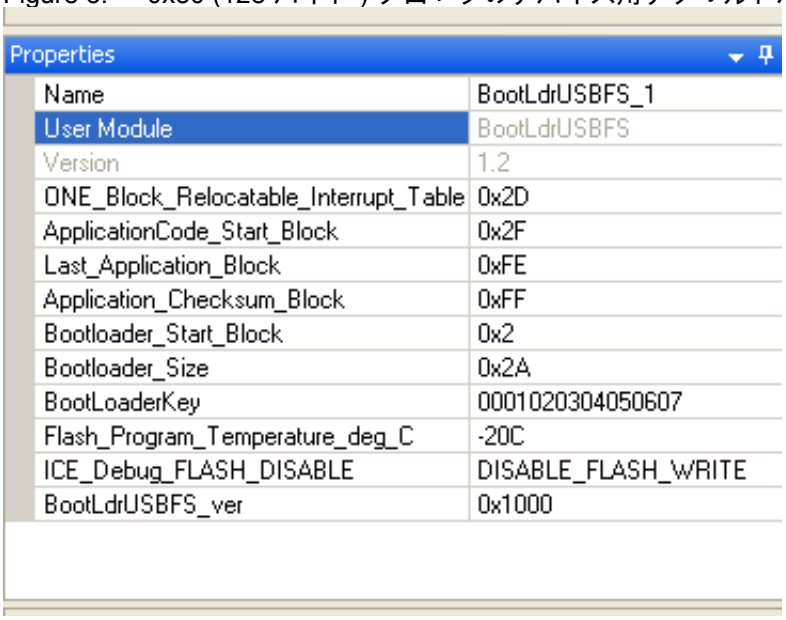
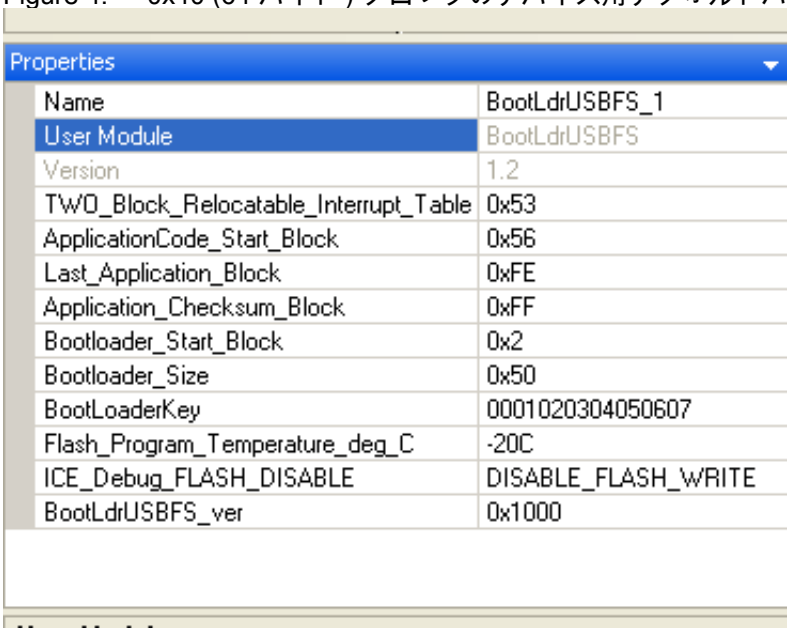


Figure 4. 0x40 (64 バイト) ブロックのデバイス用デフォルトパラメータ、HID 選択オプション



TWO_Block Relocatable_Interrupt_Table

ONE_Block Relocatable_Interrupt_Table

大部分の PSoC パーツは 64 バイトのブロックサイズです。これらのパーツでは、ROM 領域の 2 ブロックは、PSoC ROM のブロック 0 と 1 で常に存在するテーブルに適合する割込みテーブルのセットを定義するために使用されます。PSoC デバイスの中には 128 バイトブロックを持ち、テーブルの保持に ROM ブロックが 1 つだけ必要なものもあります。ブロック 0 と 1 の割込みテーブルがここにリダイレクトされるため、これらのテーブルは、メモリの同じ位置に維持される必要があります。これらの再配置可能なテーブルは、アプリケーションプログラムに対しては再配置可能な割込みベクタを意味します。

またこのパラメータは、ブートローダツールによって使用され、.dld ファイル用の処理に使用するコードブロックと、チェックサムの計算に使用するコードブロックを指定します。この変数は、ブートローダユーティリティが自動的にアプリケーションのチェックサムを確認した際に使用するチェックサムブロックに割り当てられます。

ApplicationCode_Start_Block

これはユーザアプリケーションに割り当てられる最初のコードブロックです。このコードはブートロード可能です。またこのパラメータは、ブートローダツールによって使用され、.dld ファイル用の処理に使用するコードブロックと、チェックサムの計算に使用するコードブロックを指定します。この変数は、ブートローダユーティリティが自動的にアプリケーションのチェックサムを確認した際に使用するチェックサムブロックに割り当てられます。

Last_Application_Block

これはユーザアプリケーションに割り当てられる最後のコードブロックです。またこのパラメータは、ブートローダツールによっても使用され、<prj_name>.dld ファイル用の処理に使用するコードブロックと、チェックサムの計算に使用するコードブロックを指定します。この変数は、ブートローダユーティリティが自動的にアプリケーションのチェックサムを確認した際に使用するチェックサムブロックに割り当てられます（絶対アドレスの変換後）。アプリケーション完了後、チェックサム操作の間処理時間を節約し、ROM の空ブロックでブートロードすることを避けるために、この値は低減される可能性があります。アップグレードされたアプリケーションが大きくなっている場合、大きなコードスペースの要件を満たすため、使用可能な ROM 領域の限界までこの値は増加することがあります。

Application_Checksum_Block

これはチェックサム記憶装置ブロックの位置です。アプリケーションが変更になっても、このブロックは書き換えることができます。ただし、ブートローダを実装せずに、位置を移動させてはなりません。

このブロックはアプリケーション領域に配置してはなりません。アプリケーション領域に配置すると、アプリケーションをチェックサムブロックの「周辺」で関連付けることが必要となりますが、これは現段階のリンカの機能では対応できません。そのため、ブロックはじゃまにならない、アプリケーションから離れた場所に配置します。配置の位置候補：ブートローダとアプリケーションの間の 1 ブロック、または ROM の最後のブロックのうちの 1 つ。

Bootloader_Start_Block

これは、ブートローダユーティリティの最初のブロックです。通常、この位置を変更する必要はありません。ただし、特殊なケースで、現在実装されているカスタムブートローダとの整合性を維持するために、移動が必要になる場合があります。

Bootloader_Size

これは、ブートローダアプリケーション用ブロックのサイズです。通常、このサイズは調整を必要としません。ブートローダにコードを追加する場合、このサイズを大きくすることができます。他のブロックも同様に調整できます。

BootLoaderKey

これは、ブートローダアプリケーションに送信されるトランザクションに添付されたキー値です。これは、ブートローダアップグレードユーティリティが間違って起動しないことを保証するための追加検証ステップです。

デフォルト値は 「0001020304050607」です。

Flash_Program_Temperature_deg_C

これは、デバイスが再プログラミングされた際に期待される一般的なプログラミング温度です。このパラメータで指定された以外の温度でデバイスをプログラミングすると、プログラムの保持力に悪影響が及ぶ場合があります。

ブートロード中にプログラムの温度パラメータを実際の温度に適合させると、メモリ保持と書き込み周期の最大数に影響が及びます。温度が下がると、PSoC の Flash 書き込みは強くなります。パラメータの設定より大幅に低い温度でブートロードする場合、メモリ保持機能が低下することもあります。このため、ブートローダをこのパラメータ値より 20 度以上差のある温度で操作しないようご注意ください。詳細については、サイプレスデバイス仕様を参照してください。

ICE_Debug_Flash_DISABLE

このパラメータは、USB リソースがオンで操作中に SSC を実行している場合、ICE のデバッグの状況で示された異常に対応するために使用します。SSC 操作が呼び出される（Flash 書き込み中）と、USB SIE は無効になります。Flash 書き込みを無効にすると、コードを Flash に書き込む必要なしに、アプリケーションを完全にテストできます。

デフォルト値は 「Flash Write DISABLE」です。

BootLdrUSBFS_ver

これはブートローダのバージョンです。現在内部ファームウェアでは使用されていませんが、チェックサムブロックの一部として利用可能です。この値は、ブートローダ実行可能コードの正しいバージョンであることを検証するために設定･使用できます。

Bootload_when_CKSUM_fails

通常、リセット時間でアプリケーションチェックサムが不正な場合、ブートローダがアクティブとなり、新しいアプリケーションがブートロードされるのを待ちます。ICE またはデバッガを用いたコード開発では、コンパイルを終えるたびにチェックサムを修正するというステップを追加すると、不便な場合があります。この機能を使用すると、ブートローダの操作を考慮することなく、アプリケーションを実行･デバッグできます。チェックサム機能は、アプリケーションフィールドでの実装用に再び有効にします。

アプリケーション プログラミング インタフェース (API)

このセクションでは、ブートローダ用の API、およびブートローダ内に含まれる USB 機能について説明します。ブートローダの主目的がユーザアプリケーションを完全に削除し、置き換えることであるため、ブートローダには、非常に限定された API セットが含まれています。

ユーザ モジュールを作成するたびに、インスタンス名が発行されます。デフォルトでは、PSoC デザイナが、そのプロジェクトのユーザ モジュールの第 1 インスタンスに BootLdrUSBFS_1 を割り当てます。これは、識別名のシンタックス規則に従った独自の値に変更できます。割り当てられたインスタンスは、グローバル関数名、変数、定数記号のプリフィクスとなります。次の説明では、簡単にするために、インスタンス名は省略されて単に「BootLdrUSBFS」となっています。一部の API には、インスタンス名が添付されていません。*ENTER_BOOTLOADER* など、これらのルーチンは不変です。

ブートローダ API

ENTER_BOOTLOADER()

GenericBootloaderEntry

説明：

ブートローダホストとデバイスの通信が開始しない場合、ブートローダアプリケーションを入力し、タイムアウト後に戻します（タイムアウトが設定されている場合）。実装されたパーツの寿命の限り、固定アドレスにある一般的なパラメータが定義されます。この機能は、既知の 16 進アドレスへのダイレクト呼び出しによって実装される場合があります。

この機能は、GenericBootloaderEntry へのジャンプを実行し、0x7C に存在します。

C プロトタイプ：

```
void ENTER_BOOTLOADER(void)
```

アセンブリ：

```
lcall  ENTER_BOOTLOADER ; Call the Start Function
```

代替案：

```
GenericHardDefinition: equ (0x7C)
lcall  GenericHardDefinition            ; Call the Start Function
```

パラメータ：

なし

GenericApplicationStart

説明：

boot.asm の始めにアプリケーションを入力します。warm boot に似ています。

C プロトタイプ：

```
void GenericApplicationStart(void)
```

アセンブリ：

```
lcall  GenericApplicationStart             ; Call the Start Function
```

パラメータ：

なし

bootLoaderVerify

説明：

アプリケーション記憶装置領域のチェックサム検証を実行します。チェックサム検証が成功しなかった場合、ブートローダが開始され、この機能は戻ってきません。これ以外では、フォアグラウンドアプリケーションも boot.asm から実行されます。

C プロトタイプ：

```
void bootLoaderVerify(void)
```

アセンブリ：

```
lcall bootLoaderVerify              ; Call the Start Function
```

パラメータ：

なし。

特殊作用：

この関数、または今後この関数の実装によって、A および X レジスタが変更される場合があります。現在変更されているのは、IDX_PP と CUR_PP のページポインタレジスタだけです。

USBFS API

このセクションにおけるアプリケーション プログラミング インタフェース（API）のルーチンは、USBFS ユーザ モジュールのプログラムにより制御が可能です。以下のセクションでは、デスクリプタの生成と統合、および基本的なデバイス特有 API 機能について説明しています。アプリケーションを開発するには、USB プロトコルの基本的な理解と、USB 2.0 仕様の知識、特に 「第 9 章：USB デバイス フレームワーク」を熟知していることが必要です。

USBFS ユーザ モジュールは、制御、割込み、バルク、アイソクロナス転送をサポートします。LoadInEP や EnableOutEP など、個別の機能やグループ機能の一部は、バルクおよび割込みエンドポイントとともに使用するようにデザインされています。BootLdrUSBFS_LoadINISOCEP などその他の機能は、等時性エンドポイントとともに使用するようにデザインされています。これらの転送タイプの操作方法に関する詳細は、Technical Reference Manual （TRM）を参照してください。

Note

1. ** すべてのユーザ モジュール API では、API 関数を呼び出すことで A と X レジスタの値が変更されることがあります。A と X の値が呼び出し後に必要な場合は、呼び出し元関数で A と X の値を保存してください。効率を高めるために、この「レジスタを volatile」とするポリシーが採択されました。C コンパイラは、自動的にこの条件で処理されています。アセンブラ言語のプログラマは、コードがこのポリシーを遵守していることも確認する必要があります。一部のユーザ モジュール API 関数では A と X は変更されないこともありますが、将来も変更されないという保証はありません。
2. USB ユーザモジュール用の API ルーチンは、再入可能ではありません。これらは RAM の内部グローバル変数に依存するため、これらのルーチンを割込みから実行することは、このユーザ モジュールに同梱の API ではサポートされていません。デザイン要件によりこの機能をご希望のユーザは、お近くのサイプレス フィールド アプリケーション エンジニアまでお問い合わせください。

関数	説明
void BootLdrUSBFS_Start(BYTE bDevice, BYTE bMode)	デバイスと特定の電圧モードで使用するためにユーザ モジュールをアクティブにします。
void BootLdrUSBFS_Stop(void)	ユーザ モジュールを無効にします。
BYTE BootLdrUSBFS_bCheckActivity(void)	USB バス アクティビティフラグをチェックし、クリアします。USB が最後のチェックからアクティブの場合 1 を返します。その他の場合は 0 を返します。
void BootLdrUSBFS_SetPowerStatus(BYTE bPowerStatus)	電源を自己供給またはバスにより供給を受けるように設定します。
BYTE BootLdrUSBFS_bGetConfiguration(void)	現在割り当てられている構成を返します。デバイスが構成されていない場合、0 を返します。
BYTE BootLdrUSBFS_bGetEPState(BYTE bEPNumber)	指定の USBFS エンドポイントの現在の状態を返します。 2 = NO_EVENT_ALLOWED 1 = イベントが保留中です 0 = NO_EVENT_PENDING
BYTE BootLdrUSBFS_bGetEPAckState(BYTE bEPNumber)	ゼロ以外の値を返すことにより、ACK が設定されているかどうかを特定します。
BYTE BootLdrUSBFS_wGetEPCount(BYTE bEPNumber)	指定の USBFS エンドポイントの現在のバイトカウントを返します。
void BootLdrUSBFS_LoadInEP(BYTE bEPNumber, BYTE *pData, WORD wLength, BYTE bToggle)	IN 転送用に特定の USBFS エンドポイントを読み込み、有効にします。
void USB_LoadInISOCEP(BYTE bEPNumber, BYTE *pData, WORD wLength, BYTE bToggle)	
BYTE BootLdrUSBFS_bReadOutEP(BYTE bEPNumber, BYTE *pData, WORD wLength)	エンドポイント RAM から特定のバイト数を読み出し、pSrc が示す RAM アレイに配置します。関数は、ホストが送信したバイト数を返します。

関数	説明
void USB_EnableOutEP(BYTE bEPNumber)	OUT 転送を受け入れるために。指定 USB エンドポイントを有効にします。
void USB_EnableOutISOCEP(BYTE bEPNumber)	
void BootLdrUSBFS_DisableOutEP(BYTE bEPNumber)	OUT 転送を否定応答するために、指定の USB エンドポイントを無効にします。

BootLdrUSBFS_Force(BYTE bState)	USB D+/D- ピンに J、K、または SE0 の状態を強制します。通常はリモート ウェークアップに使用されます。
	bState パラメータ :
	`USB_FORCE_SE0  0xC0`
	`USB_FORCE_J    0xA0`
	`USB_FORCE_K    0x80`
	`USB_FORCE_NONE 0x00`
	
	注：この API 関数とポート 1 (P1.2-P1.7) の GPIO ピンを使用している場合、アプリケーションは一貫性のあるデータ処理を保証するために、Port_1_Data_SHADE シャドーレジスタを使用します。アセンブリ言語からは、Port_1_Data_SHADERAM 位置に直接アクセスします。C 言語からは、次の外部レファレンスを含めます。
	`extern BYTE Port_1_Data_SHADE;`

関数	説明
BYTE BootLdrUSBFS_UpdateHIDTimer(BYTE bInterface)	指定インターフェース用に HID レポートタイマを更新し、タイマが時間切れとなっていたら 1 を、それ以外は 0 を返します。タイマが時間切れとなった場合、タイマを再読み込みします。
BYTE BootLdrUSBFS_bGetProtocol(BYTE bInterface)	指定インターフェース用のプロトコルを返します。

BootLdrUSBFS_Start(ユーザ定義によるアプリケーションデバイス

BootLdrUSBFS_Start(ブートローダのデバイス) 0xFF

説明：

USBFS ユーザ モジュール用にすべての必要な初期化を実行します。フォアグラウンド USB デバイスも、ブートローダ指定の USB デバイスも、このコマンドを使用して開始できます。アクティブにできる USB デバイス構成は一度に 1 つだけです。ブートローダデバイスを開始するには、bDevice の値を -1 (0xFF) に設定します。

C プロトタイプ：

```
void BootLdrUSBFS_Start(BYTE bDevice, BYTE bMode)
```

アセンブリ：

```
mov   A, 0xFF                ; The bootloader device descriptor
mov   A, 0                   ; Select application device descriptor
mov   X, USB_5V_OPERATION    ; Select the Voltage level
lcall  BootLdrUSBFS_Start     ; Call the Start Function
```

パラメータ：

レジスタ A: USBFS セットアップウィザードに入力されている任意のデバイス記述子セットからのデバイス番号を含みます。

レジスタ X: チップが実行される操作電圧を含みます。5V 操作用に電圧調整器が有効か、あるいはパススルーモードが 3.3V 操作で使用されているかを判断します。C およびアセンブラで記号名が登録されていますが、関連する値を以下の表に示します。

マスク	値	説明
USB_3V_OPERATION	0x02	プルアップ用に電圧調整器とパススルー vcc を無効にします。
USB_5V_OPERATION	0x03	プルアップ用に電圧調整器を無効にし、使用します。

戻り値：

なし

特殊作用：

この関数、または今後この関数の実装によって、A および X レジスタが変更される場合があります。現在変更されているのは、IDX_PP と CUR_PP のページポインタレジスタだけです。

BootLdrUSBFS_Stop

説明：

USBFS ユーザ モジュールに必要なシャットダウンタスクをすべて実行します。

C プロトタイプ：

```
void  BootLdrUSBFS_Stop(void)
```

アセンブリ：

```
lcall BootLdrUSBFS_Stop
```

パラメータ：

なし

戻り値：

なし

特殊作用：

この関数、または今後この関数の実装によって、A および X レジスタが変更される場合があります。現在変更されているのは、CUR_PP のページポインタレジスタだけです。

BootLdrUSBFS_bCheckActivity

説明：

USBFS バス アクティビティを確認します。

C プロトタイプ：

```
BYTE BootLdrUSBFS_bCheckActivity(void)
```

アセンブリ：

```
lcall BootLdrUSBFS_bCheckActivity
```

パラメータ：

なし

戻り値：

USB が最後のチェックからアクティブの場合 A に 1 を返します。その他の場合は 0 を返します。

特殊作用：

この関数、または今後この関数の実装によって、A および X レジスタが変更される場合があります。

BootLdrUSBFS_bGetConfiguration

説明：

USB デバイスの現在の構成を取得します。

C プロトタイプ：

```
BYTE BootLdrUSBFS_bGetConfiguration(void)
```

アセンブリ：

```
lcall BootLdrUSBFS_bGetConfiguration
```

パラメータ：

なし

戻り値：

A に現在割り当てられている構成を返します。デバイスが構成されていない場合は 0 を返します。

特殊作用：

この関数、または今後この関数の実装によって、A および X レジスタが変更される場合があります。これは大容量メモリ モデルで、すべての RAM ページ ポインタ レジスタについて同じです。

必要に応じて、fastcall16 関数を呼び出す値を維持するのは、呼び出す関数が行います。現在変更されているのは、CUR_PP のページポインタレジスタだけです。

BootLdrUSBFS_bGetEPState

説明：

指定したエンドポイント用のエンドポイント状態を取得します。フォアグラウンドアプリケーションの検知から、エンドポイントの状態を記述します。エンドポイントには 3 つの状態のうち、1 つが該当します。2 つの状態が該当した場合、IN と OUT のエンドポイントにとって異なる意味があります。次のテーブルには、考えられる状態と、IN と OUT のエンドポイントに対するその意味が記載されています。

C プロトタイプ：

```
BYTE BootLdrUSBFS_bGetEPState(BYTE bEPNumber)
```

アセンブリ：

```
MOV A, 1                            ; Select endpoint 1
lcall BootLdrUSBFS_bGetEPState
```

パラメータ：

レジスタ A はエンドポイント番号を含みます。

戻り値：

指定の USBFS エンドポイントの現在の状態を返します。C およびアセンブラで記号名が登録されていますが、関連する値を以下の表に示します。送受信での処理に際して、ISR コードなどのエンドポイントの状態を変更するために、コードを書く際にはこれらを使用してください。

状態	値	説明
NO_EVENT_PENDING	0x00	エンドポイントが SIE のアクションを待っていることを示します。
EVENT_PENDING	0x01	エンドポイントが CPU のアクションを待っていることを示します。
NO_EVENT_ALLOWED	0x02	エンドポイントがアクセスからロックされていることを示します。
IN_BUFFER_FULL	0x00	IN エンドポイントが読み込まれ、モードは ACK IN に設定されています。
IN_BUFFER_EMPTY	0x01	IN トランザクションが発生し、より多くのデータを読み込むことができます。
OUT_BUFFER_EMPTY	0x00	OUT エンドポイントは、ACK OUT を設定するためのもので、データを待っています。
OUT_BUFFER_FULL	0x01	OUT トランザクションが発生し、データを読むことができます。

特殊作用：

この関数、または今後この関数の実装によって、A および X レジスタが変更される場合があります。これは大容量メモリ モデルで、すべての RAM ページ ポインタ レジスタについて同じです。必要に応じて、fastcall16 関数を呼び出す値を維持するのは、呼び出す関数が行います。現在変更されているのは、IDX_PP のページポインタレジスタだけです。

BootLdrUSBFS_bGetEPAckState

説明：

エンドポイントの制御レジスタの ACK ビットを読むことで、ACK トランザクションがこのエンドポイントで発生したかどうかを判断します。この関数は ACK ビットをクリアしません。

C プロトタイプ：

```
BYTE BootLdrUSBFS_bGetEPState(BYTE bEPNumber)
```

アセンブリ：

```
MOV A, 1                            ; Select endpoint 1
lcall BootLdrUSBFS_bGetEPState
```

パラメータ：

レジスタ A はエンドポイント番号を含みます。

戻り値：

ACK トランザクションが発生した場合、この関数はゼロ以外の値を戻します。それ以外の場合は、0 を返します。

特殊作用：

この関数、または今後この関数の実装によって、A および X レジスタが変更される場合があります。これは大容量メモリ モデルで、すべての RAM ページ ポインタ レジスタについて同じです。必要に応じて、fastcall16 関数を呼び出す値を維持するのは、呼び出す関数が行います。

BootLdrUSBFS_wGetEPCount

説明：

この関数は、エンドポイントカウントレジスタの値を返します。シリアルインターフェースエンジン (SIE) には、カウント内のチェックサムでの 2 バイトを含んでいます。この関数は、値を戻す前にカウントから 2 を引きます。USB_GetEPState への呼び出しが EVENT_PENDING. を返したあと、OUT エンドポイント用にこの機能を呼び出します。

C プロトタイプ：

```
WORD BootLdrUSBFS_wGetEPCount(BYTEbEPNumber)
```

アセンブリ：

```
MOV A, 1                            ; Select endpoint 1
lcall BootLdrUSBFS_bGetEPCount
```

パラメータ：

レジスタ A はエンドポイント番号を含みます。

戻り値：

A と X にある指定の USBFS エンドポイントの現在のバイトカウントを返します。

特殊作用：

この関数、または今後この関数の実装によって、A および X レジスタが変更される場合があります。これは大容量メモリ モデルで、すべての RAM ページ ポインタ レジスタについて同じです。必要に応じて、fastcall16 関数を呼び出す値を維持するのは、呼び出す関数が行います。

BootLdrUSBFS_LoadInEP

BootLdrUSBFS_LoadInISOCEP

説明：

IN 割込みまたはバルク転送（..._LoadInEP）および等時性転送（..._LoadInISOCEP）用の指定 USB エンドポイントを読み込み・有効にします。

C プロトタイプ：

```
void BootLdrUSBFS_LoadInEP(BYTE bEPNumber, BYTE * pData, WORD wLength, BYTE bToggle)
void BootLdrUSBFS_LoadInISOCEP(BYTE bEPNumber, BYTE * pData, WORD wLength, BYTE
bToggle)
```

アセンブリ：

```
mov A, USBFS_TOGGLE
push A
mov A, 0
push A
mov A, 32
push A
mov A, >pData
push A
mov A, <pData
push A
mov A, 1
push A
lcall BootLdrUSBFS_LoadInEP
```

パラメータ：

bEPNumber – 1 と 4 の間のエンドポイント番号。

pData – データアレイへのポインタ。データエンドポイント用のデータは pData に指定されたデータアレイから読み込まれます。

wLength – IN 要求の結果として、アレイから転送されるバイト数。有効値は 0 ~ 256 です。

bToggle – カウントレジスタで設定する前にデータトグルビットがトグルするか否かを示すフラグ。IN トランザクションでは、データ転送が成功したあとは毎回データビットをトグルします。これは、同じパケットが繰り返されたり、紛失したりすることがないようにするためです。フラグ用の記号が C とアセンブリで提供されます。

マスク	値	説明
USB_NO_TOGGLE	0x00	データトグルは変わりません。
USB_TOGGLE	0x01	データビットは転送前にトグルされます。

戻り値：

なし

特殊作用：

この関数、または今後この関数の実装によって、A および X レジスタが変更される場合があります。これは大容量メモリ モデルで、すべての RAM ページ ポインタ レジスタについて同じです。必要に応じて、fastcall16 関数を呼び出す値を維持するのは、呼び出す関数が行います。現在変更されているのは、IDX_PP と CUR_PP のページポインタレジスタだけです。

BootLdrUSBFS_bReadOutEP

説明：

エンドポイント RAM からデータ RAM への指定のバイト数を移動します。エンドポイント RAM からデータ RAM へ実際に転送されるバイト数は、ホストから送信される実際のバイト数と wCount 引数から要求されるバイト数の小さいほうです。

C プロトタイプ：

```
BYTE BootLdrUSBFS_bReadOutEP(BYTE bEPNumber, BYTE * pData, WORD wLength)
```

アセンブリ：

```
mov A, 0
push A
mov A, 32
push A
mov A, >pData
push A
mov A, <pData
push A
mov A, 1
push A
lcall BootLdrUSBFS_bReadOutEP
```

パラメータ：

bEPNumber – 1 と 4 の間のエンドポイント番号。

pData – エンドポイントスペースは、このポインタが指定するデータアレイから読み込まれます。

wLength – アレイから転送され、次に IN 要求の結果として送信されるバイト数。有効値は 0 ~ 256 です。ホストから転送されるバイト数への要求が少ない場合、関数の移動もこれより少なくなります。

戻り値：

ホストが USB データに送信したバイト数を返します。これは要求されたバイト数に対して少し多いか少ない場合があります。

副作用：

この関数、または今後この関数の実装によって、A および X レジスタが変更される場合があります。これは大容量メモリ モデルで、すべての RAM ページ ポインタ レジスタについて同じです。必要に応じて、fastcall16 関数を呼び出す値を維持するのは、呼び出す関数が行います。現在変更されているのは、IDX_PP と CUR_PP のページポインタレジスタだけです。

BootLdrUSBFS_EnableOutEP

USBFS_EnableOutISOCEP

説明：

OUT バルクまたは割込み転送（..._EnableOutEP）および等時性転送（..._EnableOutISOCEP）用の指定エンドポイントを有効にします。IN エンドポイントに関してこれらの関数を呼び出してはなりません。

C プロトタイプ：

```
void USBFS_EnableOutEP(BYTE bEPNumber)
void USBFS_EnableOutISOCEP(BYTE bEPNumber)
```

アセンブリ：

```
MOV A, 1
lcall USBFS_EnableOutEP
```

パラメータ：

レジスタ A はエンドポイント番号を含みます。

戻り値：

なし

特殊作用：

この関数、または今後この関数の実装によって、A および X レジスタが変更される場合があります。現在変更されているのは、IDX_PP のページポインタレジスタだけです。

BootLdrUSBFS_DisableOutEP

説明：

指定の USBFS OUT エンドポイントを無効にします。IN エンドポイントに関してこの関数を呼び出してはなりません。

C プロトタイプ：

```
void BootLdrUSBFS_DisableEP(BYTE bEPNumber)
```

アセンブリ：

```
MOV A, 1                            ; Select endpoint 1
lcall BootLdrUSBFS_DisableEP
```

パラメータ：

レジスタ A はエンドポイント番号を含みます。

戻り値：

なし

特殊作用：

この関数、または今後この関数の実装によって、A および X レジスタが変更される場合があります。これは大容量メモリ モデルで、すべての RAM ページ ポインタ レジスタについて同じです。必要に応じて、fastcall16 関数を呼び出す値を維持するのは、呼び出す関数が行います。

BootLdrUSBFS_Force

説明：

D+/D- 行に USB J、K、または SE0 の状態を強制します。この関数は、USB デバイスアプリケーションが USB リモート ウェークアップ機能を実施するために必要なメカニズムを提供します。詳細については、USB 2.0 仕様の一時停止と再開機能の詳細を参照してください。

C プロトタイプ：

```
void BootLdrUSBFS_Force(BYTE bState)
```

アセンブリ：

```
mov A, BootLdrUSB_FORCE_K
lcall BootLdrUSBFS_Force
```

パラメータ：

bState は 4 つのバス状態のうち有効にする 1 つを示すバイトです。C およびアセンブラで記号名が登録されていますが、関連する値を以下の表に示します。

状態	値	説明
USB_FORCE_SE0	0xC0	D+/D- 行にシングル エンドの 0 を強制します。
USB_FORCE_J	0xA0	D+/D- 行に J の状態を強制します。
USB_FORCE_K	0x80	D+/D- 行に K の状態を強制します。
USB_FORCE_NONE	0x00	SIE 制御にバスを返します。

戻り値：

なし

特殊作用：

この関数、または今後この関数の実装によって、A および X レジスタが変更される場合があります。これは大容量メモリ モデルで、すべての RAM ページ ポインタ レジスタについて同じです。必要に応じて、fastcall16 関数を呼び出す値を維持するのは、呼び出す関数が行います。

BootLdrUSBFS_UpdateHIDTimer

説明：

HID レポート アイドルタイマを更新し、期間満了ステータスを返します。タイマが期限切れになったら、再び読み込みます。

C プロトタイプ：

```
BYTE BootLdrUSBFS_UpdateHIDTimer(BYTE bInterface)
```

アセンブリ：

```
MOV A, 1                            ; Select interface 1
lcall BootLdrUSBFS_UpdateHIDTimer
```

パラメータ：

レジスタ A はインターフェース番号を含みます。

戻り値：

HID タイマの状態が A に返されます。C およびアセンブラで記号名が登録されていますが、関連する値を以下の表に示します。

状態	値	説明
USB_IDLE_TIMER_EXPIRED	0x01	タイマが期限切れです。
USB_IDLE_TIMER_RUNNING	0x02	タイマは作動しています。
USB_IDLE_TIMER_IDEFINITE	0x00	データまたは状態が変更され、レポートが送信されると返されます。

特殊作用：

この関数、または今後この関数の実装によって、A および X レジスタが変更される場合があります。

BootLdrUSBFS_bGetProtocol

説明：

選択されたインターフェース用の HID プロトコルを返します。

C プロトタイプ：

```
BYTE BootLdrUSBFS_bGetProtocol(BYTE bInterface)
```

アセンブリ：

```
MOV A, 1                            ; Select interface 1
lcall BootLdrUSBFS_bGetProtocol
```

パラメータ：

bInterface には、インターフェース番号が含まれます。

戻り値：

レジスタ A にはプロトコル値が含まれます。

特殊作用：

この関数、または今後この関数の実装によって、A および X レジスタが変更される場合があります。これは大容量メモリ モデルで、すべての RAM ページ ポインタ レジスタについて同じです。必要に応じて、fastcall16 関数を呼び出す値を維持するのは、呼び出す関数が行います。

BootLdrUSBFS_SetPowerStatus

説明：

現在の電源ステータスを設定します。電源ステータスは、自己供給の場合は 1 で、バス供給の場合は 0 です。デバイスはこの値に基づいて USBGET_STATUS 要求に反応します。これにより、デバイスは、USB 仕様書第 9 章に規定されている準拠の状態を適正にレポートできます。デバイスは、いつでも電源を自己からバス供給に変更する可能性があり、現在の電源をデバイスステータスの一部としてレポートします。この関数はいつでも呼び出すことができます。デバイスは自己からバス供給に、またはその逆に変更し、適切にステータスを設定します。

C プロトタイプ：

```
void BootLdrUSBFS_SetPowerStatus(BYTE bPowerStaus);
```

アセンブリ：

```
MOV   A, 1                          ; Select self powered
lcall  BootLdrUSBFS_SetPowerStatus
```

パラメータ：

bPowerStatus は任意の電源ステータスを含んでおり、1 が自己供給で、0 がバス供給です。C およびアセンブラで記号名が登録されていますが、関連する値を以下の表に示します。

状態	値	説明
USB_DEVICE_STATUS_BUS_POWERED	0x00	デバイスの電源をバス供給に設定します。
USB_DEVICE_STATUS_SELF_POWERED	0x01	デバイスの電源を自己供給に設定します。

戻り値：

なし

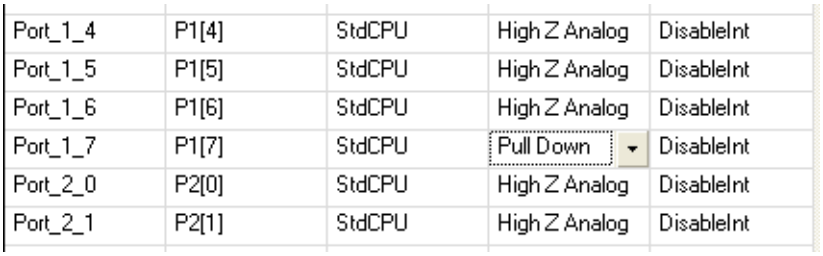
特殊作用：

この関数、または今後この関数の実装によって、A および X レジスタが変更される場合があります。これは大容量メモリ モデルで、すべての RAM ページ ポインタ レジスタについて同じです。必要に応じて、fastcall16 関数を呼び出す値を維持するのは、呼び出す関数が行います。

ファームウェア ソースコードの例

この C 言語とアセンブリ言語のプロジェクト例では、ブートローダユーザ モジュールパラメータをデフォルト状態のままにしています（図 3-4）。

プルダウンとして正しいピンが構成されており、ブートローダを開始するためにスイッチが閉じたことが認識されていることを確認します。この例はユーザが押すことができるボタンに依存しています。このボタンは Port1_7 を高に設定し、プログラムがブートローダとして再構成されます。デバイスによっては、「プルダウン」に設定されるときもあり、また別のデバイスでは、「オープン - ドレイン - ロー」である場合もあります。この例でターゲットとして使用されるテストハードウェアの構成を正確に考慮する必要があります。また技術レファレンス マニュアルを参照して、このプロジェクトを正しく構成しましょう。



Port_1_4	P1[4]	StdCPU	High Z Analog	DisableInt
Port_1_5	P1[5]	StdCPU	High Z Analog	DisableInt
Port_1_6	P1[6]	StdCPU	High Z Analog	DisableInt
Port_1_7	P1[7]	StdCPU	Pull Down	DisableInt
Port_2_0	P2[0]	StdCPU	High Z Analog	DisableInt
Port_2_1	P2[1]	StdCPU	High Z Analog	DisableInt

アプリケーションをブートロードする前に、*flashsecurity.txt* ファイルを変更します。このファイルは、アプリケーション、チェックサム、書き込みと再配置可能な割込みベクタ領域を構成します。例の flashsecurity.txt ファイルは、例中では次の数値を示しています。デバイスによって、外観が異なることもありますが、すべてのデバイスが同じ基本パターンを許可します。

```
; to the end of the line.

; 0 40 80 C0 100 140 180 1C0 200 240 280 2C0 300 340 380 3C0 (+) Base Address

  W  W  W  W  W  W  W  W  W  W  W  W  W  W  W  W ;    Base Address 0
  W  W  W  W  W  W  W  W  W  W  W  W  W  W  W  W ;    Base Address 400
  W  W  W  W  W  W  W  W  W  W  W  W  W  W  W  W ;    Base Address 800
  W  W  W  W  W  W  W  W  W  W  W  W  W  W  W  W ;    Base Address C00
  W  W  W  W  W  W  W  W  W  W  W  W  W  W  U  U ;    Base Address 1000
  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U ;    Base Address 1400
  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U ;    Base Address 1800
  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U ;    Base Address 1C00
  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U ;    Base Address 2000
  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U ;    Base Address 2400
  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U ;    Base Address 2800
  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U ;    Base Address 2C00
  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U ;    Base Address 3000
  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U ;    Base Address 3400
  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U ;    Base Address 3800
  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U ;    Base Address 3C00
; End 16K parts
```

注：ホスト PC プロジェクト例には 04b4 の VID と E006 の PID があります。これらはサイプレス占有 ID で、ローカルデバッグには使用できますが、生産用のリリースに使用することは許可されていません。

これは、C 言語で書かれた USB ブートローダユーザ モジュールの実施例です。このサンプルコードは CY7C64215 デバイスファミリ用に開発されました。このユーザ モジュールがサポートする別のデバイスファミリでは、使用可能ポート、ピン、ドライブモードが異なっている場合があります。ご使用のデバイスで作動するために、このコードをカスタマイズする必要があるかもしれません。

```
//
//emulate a mouse that causes the cursor to move in a square
//

#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules

signed char bXInc = 0;  // X-Step Size
signed char bYInc = 0;  // Y-Step Size

#define USB_INIT     0   // Initialized state
#define USB_UNCONFIG 1   // Unconfigured state
#define USB_CONFIG   2   // Configured state

// Mouse movemet states
#define MOUSE_DOWN   0
#define MOUSE_LEFT   1
#define MOUSE_UP     2
#define MOUSE_RIGHT  3

#define POSMASK      0x03 // Mouse position state mask
#define BOX_SIZE     32   // Transfers per side of the box
#define bCursorStep  4    // Step size

BYTE bConfigState = 0;    // Configuration state
BYTE bDirState = 0;       // Mouse diretion state

BYTE abMouseData[3] = {0,0,0};  // Endpoint 1, mouse packet array
BYTE bButton;                   // Used for button
BYTE boxLoop = 0;               // Box loop counter

const char abDirection[4][6] = {"DOWN "};
extern const  USB_pAppChkSumBlk;
WORD blversion;
void main(void)
{
M8C_EnableGInt;                     //Enable Global Interrupts
USB_Start(0, USB_5V_OPERATION);     //Start USB Operation usgin device 0


PRT1DR = 0;

while(1) {                                  // Main loop
   if(PRT1DR & 0x80) {
   USB_Stop();
   while(PRT1DR & 0x80);
   USB_EnterBootloader();
   }
   switch(bConfigState) {                   // Check state
   case USB_INIT:                           // Initialize state
      bConfigState = USB_UNCONFIG;

      break;

   case USB_UNCONFIG:                       // Unconfigured state
```

```
      if(USB_bGetConfiguration() != 0) {      // Check if configuration set
         bConfigState = USB_CONFIG;

         USB_LoadInEP(1, abMouseData, 3, USB_NO_TOGGLE);  // Load a dummy mouse packet
      }
      break;

   case USB_CONFIG:                           // Configured state time to move the mouse
      if(USB_bGetEPAckState(1) != 0) {
         boxLoop++;
       if(boxLoop > BOX_SIZE) {              // Change mouse direction every 32 packets
            boxLoop = 0;
            bDirState++;    // Advance box state
            bDirState &= POSMASK;

         }

        switch(bDirState) {                     // Determine current direction state

         case MOUSE_DOWN:                          // Down
            bXInc = 0;
            bYInc = bCursorStep;
            //asm("nop");
            break;

         case MOUSE_LEFT:                          // Left
            bXInc = -bCursorStep;
            bYInc = 0;
            break;

         case MOUSE_UP:                            // up
            bXInc = 0;
            bYInc = -bCursorStep;
            break;

         case MOUSE_RIGHT:                         // Right
            bXInc = bCursorStep;
            bYInc = 0;
            break;

         }
         abMouseData[1] = bXInc;                    // Load the packet array
         abMouseData[2] = bYInc;
         abMouseData[0] = 0;   // No buttons pressed
         USB_LoadInEP(1, abMouseData, 3, USB_TOGGLE); // Load and cock Endpoint1


      }  // End if Endpoint ready
      break;

   }  // End Switch
}   // End While
}
```

これは、アセンブリ言語で書かれた USB ブートローダユーザ モジュールの実施です。

ここで図示されているアセンブリコードは、単純な HID アプリケーションにおける BootLdrUSBFS ユーザ モジュールの使用法を示しています。PC ホストに接続されている場合、デバイスは 3 ボタンマウスとして再構成されます。コードが実行されると、マウスのカーソルが右から左へジグザグに動きます。このコードは、BootLdrUSBFS セットアップウィザードがユーザ モジュールを構成するやり方を図示しています。このプロジェクトは、USBFS ユーザ モジュールと同一ですが、ブートローダが加わったことだけが異なる点です。

```
;-------------------------------------------------------------------------
; Assembly main line
;-------------------------------------------------------------------------

include "m8c.inc" ; part specific constants and macros
include "memory.inc" ; Constants & macros for SMM/LMM and Compiler
include "PSoCAPI.inc" ; PSoC API definitions for all User Modules



export _main
export i
export abMouseData

area bss(RAM) // inform assembler that variables follow
abMouseData: blk 3 // USBFS data variable
i: blk 1 // count variable

area text(ROM,REL) // inform assembler that program code follows

_main:
OR F,1
; Start USBFS Operation using device 0
PUSH X
MOV X,3
MOV A,0
LCALL BootLdrUSBFS_Start
POP X

; Wait for Device to enumerate
.no_device:
PUSH X
LCALL BootLdrUSBFS_bGetConfiguration
POP X
CMP A,0
JZ .no_device
; Enumeration is completed load endpoint 1. Do not toggle the first time
; BootLdrUSBFS_LoadInEP(1, abMouseData, 3, USB_TOGGLE);
PUSH X
MOV A,1
PUSH A
MOV A,0
PUSH A
MOV A,3
PUSH A
```

```
MOV A,>abMouseData
PUSH A
MOV A,<abMouseData
PUSH A
MOV A,1
PUSH A
LCALL BootLdrUSBFS_LoadInEP
ADD SP,250
POP X

.endless_loop:



;implement bootloader entry
mov reg[PRT1DR], 0   ;load reg[PRT1DR] with  0
;   if(PRT1DR & 0x80) {
;   USB_Stop();
;   while(PRT1DR & 0x80);
;   BootLdrUSBFS_EnterBootloader();
;   }
push A
mov A, reg[PRT1DR]
and A, 0x80
jz .Exit_BOOTLOAD_TEST
;****    IMPORTANT,      configure prt0.7 as a stdcpu/pulldown     IMPORTANT     *****
;if PRT1DR.7 is pulled high, (configure for a pull down, set data to zero)
; wait for the port pin to be released (back to zero) to debounce
; immeciately un-enumerate by releasing the D+ pullup
lcall BootLdrUSBFS_Stop
.wait_for_bit_low:
tst reg[PRT1DR], 0x80
jnz .wait_for_bit_low
; once it goes low enter the bootloader
pop A
ljmp BootLdrUSBFS_EnterBootloader ;;never returns
halt

.Exit_BOOTLOAD_TEST:
pop A


;;; mouse operations



PUSH X
MOV A,1
LCALL BootLdrUSBFS_bGetEPAckState
POP X
CMP A,0
JZ .endless_loop
; ACK has occurred, load the endpoint and toggle the data bit
; BootLdrUSBFS_LoadInEP(1, abMouseData, 3, USB_TOGGLE);
PUSH X
```

```
MOV A,1
PUSH A
MOV A,0
PUSH A
MOV A,3
PUSH A
MOV A,>abMouseData
PUSH A
MOV A,<abMouseData
PUSH A
MOV A,1
PUSH A
LCALL BootLdrUSBFS_LoadInEP
ADD SP,250
POP X

; When our count hits 128
CMP [i],128
JNZ .move_left
; Start moving the mouse to the right
MOV [abMouseData+1],5
JMP .increment_i
; When our counts hits 255
.move_left:
CMP [i],255
JNZ .increment_i
; Start moving the mouse to the left
MOV [abMouseData+1],251

.increment_i:
INC [i]

JMP .endless_loop

.terminate:
jmp .terminate
```

USBFS セットアップとコード例の対応

1. BootLdrUSBFS ユーザ モジュールでサポートされるベースパーツを用いて、新しいプロジェクトを作成します（例：CY8C24894）
2. デバイス エディタで、[**プロトコル**] をクリックします。BootLdrUSBFS アイコンをダブルクリックするか、右クリックして [**Select**] を選んで、BootLdrUSBFS ユーザ モジュールを加えます。
3. ヒューマンインターフェースデバイス（HID）のラジオボタンの選択 操作の手順：ユーザ モジュールの名前をサンプルコードに準じて、BootLdrUSBFS_1 から BootLdrUSBFS に変更します。これは、モジュールを右クリックして [**名前の変更**] を選択することにより行います。
4. デバイス エディタ中の USBFS ユーザ モジュールアイコンを右クリックして、「デバイス：[アプリケーション USB セットアップウィザード ...] を選択します。
   - [HID レポートテンプレートをインポートする] 操作をクリックし、名前を [HID レポートテンプレートをインポートする（斜体）] に変更し、ラベルであることを示します。
   - 3 ボタンマウステンプレートを選択します。
   - テンプレート右側の [適用] をクリックします。
   - [文字列を追加する] を選択し、製造元と製品名の文字列を加えます。
   - デバイス属性：ベンダ ID、製品 ID を編集し、文字列を選択します。
   - インターフェース属性の編集：クラスフィールドで HID を選択します。
   - HID クラス記述子の編集：3 ボタンマウスで [HID レポート] フィールドを選択します。
5. **[OK]** をクリックして USB 記述子情報を保存します。
6. デバイス エディタ中の USBFS ユーザ モジュールアイコンを右クリックして、「デバイス：[ブートローダ USB セットアップウィザード] を開きます。
7. 正しい VID（ベンダ ID）と PID（製品 ID）をウィザードに入力します。注：アプリケーションとブートローダ用の VID と PID は同一であってはなりません。
8. **[OK]** をクリックして USB BootLoader 記述子情報を保存します。
9. アプリケーションの作成
10. サンプルコードをコピーし、main.c にペーストします。
11. [**すべて再構成**] を実行します。

記述子	データ
USB ユーザ モジュールデスクリプタルート	デバイス名
デバイス記述子	デバイス
デバイス属性	
ベンダ ID	企業 VID を使用
製品 ID	製品 PID を使用
デバイスリリース（bcdDevice）	0000
デバイスクラス	インターフェース記述子による定義
サブクラス	サブクラスなし

記述子	データ
製造者文字列	My company
製品文字	My mouse
シリアル番号文字列	文字列なし
コンフィグレーション　デスクリプタ	構成
構成属性	
構成文字列	文字列なし
最大電源値	100
デバイス電源	バス供給
リモート ウェークアップ	無効
インターフェース記述子	インターフェース
インターフェース属性	
インターフェース文字列	文字列なし
クラス	HID
サブクラス	サブクラスなし
HID クラス記述子	
記述子クラス	レポート
国コード	サポートされていません
HID レポート	3-button マウス
エンドポイント記述子	ENDPOINT_NAME
エンドポイント属性	
エンドポイント番号	1
方向	IN
転送タイプ	INT
間隔	10
最大パケットサイズ	8
文字列 /LANGID	
文字列記述子	USBFS
LANGID	
文字列	My company

記述子	データ
文字列	My mouse
記述子	
HID レポート記述子ルート	USBFS
HID レポート記述子	USBFS

付録 A – USBFS トピック

USB　スタンダード　デバイス　リクエスト

次のセクションでは、USB ユーザ モジュールがサポートする要求について説明しています。USB ユーザ モジュールがサポートしない要求は、リクエストエラーを示す通常ストールが返されます。

標準デバイス要求	記述子をサポートする USB ユーザ モジュール	USB 2.0 仕様 セクション
CLEAR_FEATURE	デバイス：	9.4.1
	インターフェース：サポートされていません	
	エンドポイント	
GET_CONFIGURATION	現在のデバイス構成値を返します。	9.4.2
GET_DESCRIPTOR	指定された記述子を返します。	9.4.3
GET_INTERFACE	指定されたインターフェース用に選択された代替インターフェース設定を返します。	9.4.4
GET_STATUS	デバイス：	9.4.5
	インターフェース：	
	エンドポイント：	
SET_ADDRESS	今後のすべてのデバイスアクセスに	9.4.6
SET_CONFIGURATION	デバイス構成を設定します。	9.4.7
SET_DESCRIPTOR	この要求はオプションですが、サポートされていません。	9.4.8

標準デバイス要求	記述子をサポートする USB ユーザ モジュール	USB 2.0 仕様セクション
SET_FEATURE	デバイス： DEVICE_REMOTE_WAKEUP サポートは bRemoteWakeUp ユーザ モジュールパラメータによって選択されます。 TEST_MODE はサポートされていません。	9.4.9
	インターフェース：サポートされていません	
	エンドポイント：指定されたエンドポイントは中止されます。	
SET_INTERFACE	サポートされていません	9.4.10
SYNCH_FRAME	サポートされていません ユーザ モジュールの将来のバージョンでは、この要求をサポートして、反復フレームパターンで等時性転送を有効にします。	9.4.11

HID クラス要求

クラス要求	記述子をサポートする USB ユーザ モジュール	HID のデバイスクラス記述子 - セクション
GET_REPORT	制御パイプの形で、レポートの受取をホストに許可します。	7.2.1
GET_IDLE	具体的な入力レポートについて、現在のアイドル速度を読み込みます。	7.2.3
GET_PROTOCOL	現在アクティブなプロトコルを読み込みます（ブートかレポートプロトコル）。	7.2.5
SET_REPORT	デバイスへのレポートの送信をホストに許可します。また入力、出力、関数制御の状態も設定できる可能性もあります。	7.2.2
SET_IDLE	新しいイベントが起こるまで、または一定の時間が経つまで、Interrupt In パイプ上の特定レポートをしないようにします。	7.2.4
SET_PROTOCOL	ブート プロトコルとレポートプロトコル（またはその逆）の間で切り替えます。	7.2.6

USBFS セットアップウィザード

このセクションは、USBFS ユーザ モジュールで提供されるすべての USBFS 記述子について詳細に説明します。説明には記述子の書式とユーザ モジュールパラメータが記述子データにどのようにマッピングされるか、も含まれます。

USBFS セットアップウィザードは、USB デザインでエンジニアをサポートするためにサイプレスが提供するものです。セットアップウィザードは、デバイス記述子構造を表示し、展開時には、標準 USB 記述子の定義の一部である次のフォルダが表示されます。

- デバイス属性
- 構成記述子
- インターフェース記述子
- HID クラス記述子

- エンドポイント記述子
- 文字列 /LANGID
- HID 記述子

セットアップウィザードにアクセスするには、デバイス エディタで USB ユーザ モジュールアイコンを右クリックし、[USB セットアップウィザード] メニュー項目をクリックします。

デバイス記述子構造をすべて展開すると、セットアップウィザードのオプションすべてが表示されます。左側は記述子名、中央はデータ、左は特定の記述子に対して利用可能な動作が表示されます。場合によっては、記述子に利用可能なオプションを示すプルダウンメニューが表示されることもあります。

記述子	データ		操作
USBFS ユーザ モジュール記述子ルート	「デバイス名」		デバイスの追加
デバイス記述子	DEVICE_1		構成の削除 / 追加
デバイス属性			
ベンダ ID	FFFF		
製品 ID	FFFF		
デバイスリリース（ bcdDevice ）	0000		
デバイスクラス	未定義	プルダウン	
サブクラス	サブクラスなし	プルダウン	
プロトコル	なし	プルダウン	
製造者文字列	文字列なし	プルダウン	
製品文字列	文字列なし	プルダウン	
シリアル番号文字列	文字列なし	プルダウン	
構成記述子	CONFIG_NAME		インターフェースの削除 / 追加
構成属性			
構成文字列	文字列なし	プルダウン	
最大電源値	100		
デバイス電源	バス供給	プルダウン	
リモート ウェークアップ	無効	プルダウン	
インターフェース記述子	INTERFACE_NAME		エンドポイントの削除 / 追加
インターフェース属性			
インターフェース文字列	文字列なし	プルダウン	
クラス	ベンダ固有	プルダウン	
サブクラス	サブクラスなし	プルダウン	
HID クラス記述子			

記述子	データ		操作
記述子タイプ	レポート	プルダウン	
国コード	サポートされていません	プルダウン	
HID レポート	なし	プルダウン	
エンドポイント記述子	ENDPOINT_NAME		削除
エンドポイント属性			
エンドポイント番号	0		
方向	IN	プルダウン	
転送タイプ	CNTRL	プルダウン	
間隔	10		
最大パケットサイズ	8		
文字列 /LANGID			
文字列記述子	デバイス名		文字列の追加
LANGID		プルダウン	
文字列	選択された文字列名		削除
記述子			
HID 記述子	デバイス名		HID レポートテンプレートのインポート

USB セットアップウィザードの説明

USB セットアップウィザードのウィンドウには、プログラミングの 2 つの主要な領域を示すテーブルが表示されます。第 1 の領域は記述子 USBFS ユーザ モジュール、第 2 の領域は文字列 /LANGID、第 3 の領域は記述子 HID レポートです。テーブルの下にある 2 つのボタンを使用して、選択されたコマンドを実行します。

最初のセクションには記述子があります。第 2 のセクションは文字列 /LANGID です。文字列 ID が必要なとき、この領域は文字列の入力に使用されます。USB デバイス用に文字列を追加するには、[**文字列の追加**] をクリックします。ソフトウェアは、行を増やし、ユーザの文字列をここで編集するよう指示します。新しい文字列を入力して、[**保存 / 作成**] をクリックします。文字列を保存したら、記述子セクションでプルダウンメニューから使用可能となります。保存を選択しないと、文字列は失われます。

第 3 の領域は HID レポート記述子ルートです。この領域から、選択したデバイス用に HID レポートを加えたり、インポートしたりできます。

USB ユーザ モジュール記述子ルート

第 1 コラムには、展開したり畳んだりできるフォルダが表示されます。ここでの説明を理解するために、構造をすべて展開してみてください。すべてのオプションが表示されます。セットアップウィザードを利用すると、中央のデータコラムが表示されます。プルダウンメニューがある場合は、そこから異なるオプションを選択できます。プルダウンメニューがなく、データがある場合、カーソルを使ってデータをハイライト· 選択し、そのデータを別の値やテキストオプションで上書きします。すべての値が USB 2.0 の第 9 章仕様に適合しなければなりません。

第 1 のフォルダは *USB User Module Descriptor Root* の上に表示されます。これには、データコラムのユーザ モジュール名があります（これはソフトウェアで提供したユーザ モジュール名）。このユーザ モジュールは Interconnect View に配置されます。右コラムの [デバイスの追加] で、記述に必要とされる様々なフィールドがすべて揃っているもう一つの USB を加えることができます。新しい USB デバイス記述子は、エンドポイント記述子の後に記載されています。**OK** をクリックして保存します。新しく追加したデバイスを保存しない場合、これは使用できません。

デバイス記述子は、データとして DEVICE_NUMBER を持っており、削除されるか、構成が加えられる場合があります。既存データを上書きする、またはプルダウンメニューを使用して、特定の USB デバイスに関する情報をすべて入力することができます。

プルダウンメニューを使用するか、適切なスポットにアルファベットと数字でテキストをタイプしてデータ入力が完了したら、[**OK**] をクリックして保存します。

USB サスペンド、リジューム、リモート ウェークアップ

USBFS ユーザ モジュールは、USB サスペンド、リジューム、リモート ウェークアップをサポートします。これらの関数はユーザアプリケーションと緊密に関わっているため、USBFS ユーザ モジュールは API 関数を提供します。

USFS アクティビティのモニタリング

BootLdrUSBFS_bCheckActivityAPI 関数を使うと、USB バス アクティビティが発生したかどうかがチェックできます。アプリケーションは、この関数を使用して、USB 一時停止の条件が満たされているかを判定します。

USBFS 一時停止

USB 一時停止を入力する条件が適合したら、アプリケーションは、サスペンド電流要件を満たすように現在の消費電流を低減する適切な手順を踏みます。USB SIE とトランシーバを電源モードにすると、アプリケーションは BootLdrUSBFS_SuspendAPI と BootLdrUSBFS_bCheckActivityAPI を呼び出して USB アクティビティを検知します。この関数は、USBFS ブロックを無効にします。ただし、現在の USB アドレスを維持します（USBIO_CR0 レジスタで）。デバイスは消費電力を低減するために、スリープ機能を使用しています。

USBFS 再開

一時停止している間、デバイスは一時停止状態を終了する条件が満たされたかどうかを判定するために、定期的にチェックを行います。再開条件を確認する 1 つの方法は、スリープタイマを使用して定期的にデバイスを起こすことです。再開条件が満たされれば、アプリケーションは BootLdrUSBFS_ResumeAPI 関数を呼び出します。この関数は、USBFS SIE とトランシーバを有効にし、電源ダウンモードを終了します。これは USBCR レジスタの USB アドレスフィールドを変更せず、ホストが事前に割り当てた USB アドレスを維持しています。

USBFS リモート ウェークアップ

リモート ウェークアップをサポートするデバイスの場合、アプリケーションは、ホストが BootLdrUSBFS_bRWUEnabledAPI を用いてリモート ウェークアップを有効にしたかどうかを判定できます。デバイスが一時停止され、リモート ウェークアップ開始の条件が満たされたと判断した場合、アプリケーションは BootLdrUSBFS_ForceAPI 機能を使用して、適切な J と K の状態を USB バスに強制し、リモート ウェークアップの信号を発します。

ベンダ指定デバイス要求の作成と既存の要求の上書き

USBFS ユーザ モジュールは、セットアップパケット要求の配送ルーチンを準備することにより、ベンダ指定のデバイス要求をサポートします。また、供給された標準やクラス指定ルーチンを上書きする独自ルーチンを作成することも、サポートされていない要求のタイプを有効にすることもできます。

USBFS デバイス要求の処理

ベンダ指定やデバイス上書き要求などを含むすべての制御転送は、次のように構成されます。

- ホストからデバイスに要求情報が転送されるセットアップステージ。
- セットアップステージにおいてゼロ以上のデータトランザクションによって構成され、データの方向が指定されたデータステージ。
- 転送を帰結させるステータス ステージ。

BootLdrUSBFS ユーザ モジュールで、エンドポイント 0 割込みサービスルーチンによって処理されるすべての制御転送（ BootLdrUSBFS_EP0_ISR).

エンドポイント 0 割込みサービス ルーチンは、すべてのセットアップパケットの制御を配送ルーチンに転送します。要求はそこから bmRequestType フィールドの適切なハンドラに引き渡されます。ハンドラは特定のユーザ モジュールデータ構成を初期化し、制御をエンドポイント 0 ISR に戻します。ベンダ指定または上書きデータ要求用のハンドラは、アプリケーションによって提供されます。ユーザ モジュールは、アプリケーションを関与させることなく、データと転送のステータスステージを処理します。転送が完了したら、ユーザ モジュールは完了ステータスブロックを更新します。ステータスブロックはアプリケーションがモニタし、ベンダ指定のデバイス要求が完了したかどうかを診断します。

すべてのセットアップパケットは BootLdrUSBFS_EP0_ISR を入力し、これにより BootLdrUSBFS_bmRequestType_Dispatch ルーチンへセットアップパケットが転送されます。すべての標準デバイス要求とベンダ指定デバイス要求は、ここから配送されます。デバイス要求ハンドラは、制御書き込み用のデータを受け取るためにアプリケーションを準備する、または制御読み取りのためにホストへデータを転送する準備をしなければなりません。データなしの制御転送の場合、ハンドラはセットアップパケットそのものから情報を抜き出します。

USBFS ユーザ モジュールは、すべての要求とまったく同じ形で、データとステータスステージを処理します。データステージの場合、データは転送の方向によって、制御エンドポイントバッファとの間でコピーされます（ レジスタ EP0DATA0-EP0DATA7)。

ベンダ指定データ要求配送ルーチン

アプリケーション要件によって、USBFS ユーザ モジュールはセットアップパケットの bmRequestType フィールドに基づき、最高 8 タイプのベンダ指定デバイス要求を配送します。USB デバイス要求と bmRequestType フィールドの説明については、USB2.0 仕様書のセクション 9.3 を参照してください。USBFS ユーザ モジュールが配送する最高 8 タイプのベンダ指定デバイス要求を、表 1 に一覧表示します。

Table 1.　ベンダ指定データ要求配送ルーチン名

方向	受信先	配送ルーチンエントリポイント	有効にするフラグ
ホストからデバイス（制御書き込み）	デバイス	USB_DT_h2d_vnd_dev_Dispatch	USB_CB_h2d_vnd_dev
	インターフェース	USB_DT_h2d_vnd_ifc_Dispatch	USB_CB_h2d_vnd_ifc
	エンドポイント	USB_DT_h2d_vnd_ep_Dispatch	USB_CB_h2d_vnd_ep
	その他	USB_DT_h2d_vnd_oth_Dispatch	USB_CB_h2d_vnd_oth
デバイスからホスト（制御書き込み）	デバイス	USB_DT_d2h_vnd_dev_Dispatch	USB_CB_d2h_vnd_dev
	インターフェース	USB_DT_d2h_vnd_ifc_Dispatch	USB_CB_d2h_vnd_ifc
	エンドポイント	USB_DT_d2h_vnd_ep_Dispatch	USB_CB_d2h_vnd_ep
	その他	USB_DT_d2h_vnd_oth_Dispatch	USB_CB_d2h_vnd_oth

アプリケーションが、ベンダ固有のデバイス要求に対してアセンブリ言語配送ルーチンを提供するには、これらの手順に従わなければなりません。

1. *USBFS.inc* ファイルでは、有効化によるサポートがベンダ指定配送ルーチンに提供されます。配送ルーチンの有効化フラグを見つけ、EQU に 1 を設定します。
2. 適切な名前を持つアセンブリ言語ルーチンを書いて、デバイス要求を処理します。表 1 に記載されているエントリポイントを使用します。

既存の要求ルーチンを上書きする

標準またはクラス指定デバイス要求を上書きする、またはサポートされていないデバイス要求を有効にするには、次の手順に従います。

1. *USBFS.inc* ファイルで、USB_APP_SUPPLIED. として、指定デバイス要求を再定義します。
2. 適切な名前を持つアセンブリ言語関数を書いて、デバイス要求を処理します。アセンブリ言語関数名は、「APP_」の次にデバイス名を加えます。

例えば、供給された HID クラスのセットレポート要求 USB_CB_SRC_h2d_cls_ifc_09 を上書きするには、以下の変更を *USBFS.inc* に加えるルーチンを有効にします。

```
;@PSoC_UserCode_BODY_1@ (Do not change this line.)
   ;---------------------------------------------------
   ; Insert your custom code below this banner
   ;---------------------------------------------------
   ;   NOTE: interrupt service routines must preserve
   ;   the values of the A and X CPU registers.

; Enable an override of the HID class Set Report request.
USB_CB_SRC_h2d_cls_ifc_09: EQU USB_APP_SUPPLIED

   ;---------------------------------------------------
   ; Insert your custom code above this banner
   ;---------------------------------------------------
   ;@PSoC_UserCode_END@ (Do not change this line.)
```

次に、APP_USB_CB_SRC_h2d_cls_ifc_09. というアセンブリ言語ルーチンを書きます。デバイス要求名は USB の bmRequestType と bRequest も値から取得します（ USB 仕様：表 9-2)。

このコードは、以前の例のアセンブリルーチンのスタブです。

```
export APP_USB_CB_SRC_h2d_cls_ifc_09
APP_USB_CB_SRC_h2d_cls_ifc_09:

;Add your code here.

; Long jump to the appropriate return entry point for your application.
LJMP BootLdrUSBFS_InitControlWrite
```

付録 B – ブートローダに関して

次のセクションでは、USB ブートローダの作成時に役立つ追加情報を説明します。

配送と上書きルーチン要件

最低でも、配送と上書きルーチンは、表 2 に記載されているエンドポイント 0 ISR 戻り点のうちの 1 つへ、LJMP によって、エンドポイント 0 ISR へ制御を戻さなければなりません。このルーチンによって、A と X レジスタが破壊される可能性がありますが、スタック ポインタ（ SP) とその他の関連性のあるコンテキストは、ISR に制御を戻す前に再保存しなければなりません。

Table 2.　エンドポイント 0 ISR 戻り点

戻りエントリポイント	必要なデータ項目	説明
USB_Not_Supported	この戻り点は、要求がサポートされないときに使います。要求を停止します。	
	データ項目：なし	
USB_InitControlRead	この戻り点は、制御読み込み転送を開始するために使用します。	
	BootLdrUSBFS_DataSource (BYTE)	データソースは RAM または ROM です (USB_DS_RAM または USB_DS_ROM).。これは、ソース ROMX や MOV からのデータ移動に難しい指示が使用されているため、必要です。
	BootLdrUSBFS_TransferSize (WORD)	転送するデータバイト数。
	BootLdrUSBFS_DataPtr (WORD)	データの RAM または ROM のアドレス。
	BootLdrUSBFS_StatusBlockPtr (WORD) オプション	USB_XFER_STATUS_BLOCK マクロ内で割り当てられているステータスブロックのアドレス。
BootLdrUSBFS_InitControlWrite	この戻り点は、制御書き込み転送を開始するために使用します。	
	BootLdrUSBFS_DataSource (BYTE)	USB_DS_RAM (制御書き込み先は必ず RAM でなければなりません)。

戻りエントリポイント	必要なデータ項目	説明
	BootLdrUSBFS_TransferSize (WORD)	データを受信するアプリケーションバッファのサイズ
	BootLdrUSBFS_DataPtr (WORD)	データを受信するアプリケーションバッファの RAM アドレス
	BootLdrUSBFS_StatusBlockPtr (WORD) オプション	USB_XFER_STATUS_BLOCK マクロ内で割り当てられているステータスブロックのアドレス。
USB_InitNoDataControlTransfer	この戻り点を使用しても、制御読み込み転送は開始しません。	
	BootLdrUSBFS_StatusBlockPtr (WORD) オプション	USB_XFER_STATUS_BLOCK マクロ内で割り当てられているステータスブロックのアドレス。

ステータス完了ブロック

ステータス完了ブロックには、2 つのデータ項目があり、その 1 つは 1 バイトの完了ステータス、もう 2 つは 2 バイトの転送長です。「主要」アプリケーションは、完了ステータスをモニタしながら、どのように進むかを判断します。次の表には、完了ステータスコードが一覧表示されています。転送長は転送されるデータの実際のバイト数です。

Table 3. USBFS 転送完了コード

完了コード	説明
USB_XFER_IDLE (0x00)	USB_XFER_IDLE は、関連データバッファに有効なデータがなく、アプリケーションがバッファを使用するべきでないことを示します。実際のデータ転送は完了コードが USB_XFER_IDLE の間に行われます。ただし、これは転送が進行している様子を示していません。
USB_XFER_STATUS_ACK (0x01)	USB_XFER_STATUS_ACK は、制御転送ステータスステージが完了したことを示します。このとき、アプリケーションは関連データバッファとそのコンテンツを使用します。
USB_XFER_PREMATURE (0x02)	USB_XFER_PREMATURE は、制御転送が次の制御転送のセットアップによって割り込まれたことを示します。制御書き込みでは、関連データバッファのコンテンツに前回完了までのデータが含まれています。
USB_XFER_ERROR (0x03)	USB_XFER_ERROR は、予期されるステータスステージのトークンを受信していないことを示します。

HID クラスレポート記憶装置領域のカスタマイズ

オプションの HID クラスサポートを有効にすると、セットアップウィザードは HID クラスでからデータレポート用の固定サイズレポート記憶エリアを作成します。IN、OUT、FEATURE レポート用に別個のレポート領域を用意します。これは、レポート記述子にレポート ID 項目のタグがなく、入力、出力、機能レポート構成だけが存在する場合に十分な領域です。レポートストレージサイズをより細かく制御したい場合、または複数のレポート ID をサポートしたい場合には、次の操作が必要です。

1. ウィザードを使用してデバイスの記述、エンドポイント、HID レポートを特定し、次にアプリケーションを生成します。
2. ウィザード定義のレポートストレージエリアを無効にします（ *USB_descr.asm* ）。
3. ウィザードが作成した、レポート記憶装置領域を定義するコードをコピーします。

4. それを保護されているユーザコード領域（ *USB_descr.asm* ）、あるいはアセンブリ言語ファイルにペーストします。
5. レポート記憶装置領域を定義するコードをカスタマイズします。

デバイスの特定とアセンブリの生成

ウィザードを使用してデバイスの記述、エンドポイント、HID レポートを特定します。PSoC デザイナの [**アプリケーションの生成**] ボタンをクリックします。

ウィザード定義のレポート記憶装置領域を無効にする

USB_descr.asm ファイルで、カスタムコード領域の WIZARD_DEFINED_REPORT_STORAGE ラインのコメントを外すことにより、ウィザード定義による記憶装置領域を無効にします。そのコードを下に示します。

```
WIZARD: equ 1
WIZARD_DEFINED_REPORT_STORAGE: equ 1
   ;-------------------------------------------------
   ;@PSoC_UserCode_BODY_1@ (Do not change this line.)
   ;-------------------------------------------------
   ; Insert your custom code below this banner
   ;-------------------------------------------------
   ; Redefine the WIZARD equate to 0 below by
   ; uncommenting the WIZARD: equ 0 line
   ; to allow your custom descriptor to take effect
   ;-------------------------------------------------

   ; WIZARD: equ 0
   WIZARD_DEFINED_REPORT_STORAGE: equ 0
   ;-------------------------------------------------
   ; Insert your custom code above this banner
   ;-------------------------------------------------
   ;@PSoC_UserCode_END@ (Do not change this line.)
```

ウィザード作成によるコードのコピー

このコードを *USB_descr.asm* で見つけます。

```
;-----------------------------------------------------------------------
; HID IN Report Transfer Descriptor Table for ()
;-----------------------------------------------------------------------
IF WIZARD_DEFINED_REPORT_STORAGE
AREA  UserModules     (ROM,REL,CON)
.LITERAL
USB_D0_C1_I0_IN_RPTS:
  TD_START_TABLE 1                                ; Only 1 Transfer Descriptor
  TD_ENTRY       USB_DS_RAM, USB_HID_RPT_3_IN_RPT_SIZE, USB_INTERFACE_0_IN_RPT_DATA,
NULL_PTR
.ENDLITERAL
ENDIF ; WIZARD_DEFINED_REPORT_STORAGE
```

これには IN、OUT、FEATURE レポート用の 3 つのセクションがあります。3 セクションすべてをコピーします。

コードを保護付きユーザコード領域にペーストする

コードを保護付きのユーザコード領域 *USB_descr.asm* 、あるいはアセンブリ言語ファイルにペーストすることができます。

```
   ;-------------------------------------------------
   ;@PSoC_UserCode_BODY_2@ (Do not change this line.)
   ;-------------------------------------------------
   ; Redefine your descriptor table below. You might
   ; cut and paste code from the WIZARD descriptor
   ; above and then make your changes.
   ;-------------------------------------------------

   ;-------------------------------------------------
   ; Insert your custom code above this banner
   ;-------------------------------------------------
   ;@PSoC_UserCode_END@ (Do not change this line.)
; End of File USB_descr.asm
```

レポート記憶装置領域を定義するコードのカスタマイズ

レポート記憶装置領域を定義するには、ユーザ独自の転送記述子テーブルエントリを書く必要があります。このテーブルには、必要なデータ項目用の記憶装置スペースを定義するエントリが含まれます。テーブル中の各転送記述子エントリが、新しいレポート ID を作成します。ID はゼロから開始して連続番号が付けられています。レポート ID 0 は、USB 仕様の予約領域でユーザは使用できませんが、ID 0 用に定義された転送記述子エントリは、レポート ID がレポート記述子にない場合に使用されます。コード効率を高めるために、ID 1 の開始番号でレポート ID を使用しなければなりません。

Table 4.　転送記述子テーブルエントリ

テーブルエントリ	必要なデータ項目	説明
TD_START_TABLE	USB_NumberOfTableEntries	定義されたレポート ID 数。ID は 0 から連続番号が振られています。レポート ID 0 は使用されません。
TD_ENTRY		
	USB_DataSource	データソースは RAM または ROM です。(USB_DS_RAM または USB_DS_ROM).
	USB_TransferSize	データ転送サイズの単位はバイトです。第 1 バイトはレポート ID です。
	USB_DataPtr	データ転送の RAM または ROM アドレス。
	USB_StatusBlockPtr	USB_XFER_STATUS_BLOCK マクロ内で割り当てられているステータスブロックのアドレス。

次の例は、未使用のレポート ID 0 と、異なるサイズの 2 つの IN レポートを設定します。注：条件付アセンブリステートメントは、保護付きユーザコード領域にコードをおいたときのみ必要となります *USB_descr.asm.*

```
;-------------------------------------------------------------------------
; HID IN Report Transfer Descriptor Table for ()
;-------------------------------------------------------------------------
IF WIZARD_DEFINED_REPORT_STORAGE
ELSE

_ID0_RPT_SIZE:   EQU 0       ; 7 data bytes + report ID = 8 bytes (unused)
_SM_RPT_SIZE:    EQU 3       ; 2 data bytes + report ID = 3 bytes
_LG_RPT_SIZE:    EQU 5       ; 4 data bytes + report ID = 5 bytes

AREA data (RAM, REL, CON)

EXPORT _ID0_RPT_PTR
_ID0_RPT_PTR: BLK 0         ; Allocates space for report ID0 (unused)
EXPORT _SM_RPT_PTR
_SM_RPT_PTR:    BLK 3       ; Allocates space for report ID1
EXPORT _LG_RPT_PTR
_LG_RPT_PTR:    BLK 5       ; Allocates space for report ID2

AREA bss (RAM, REL, CON)

EXPORT _SM_RPT_STS_PTR
_SM_RPT_STS_PTR: USB_XFER_STATUS_BLOCK
EXPORT _LG_RPT_STS_PTR
_LG_RPT_STS_PTR: USB_XFER_STATUS_BLOCK

AREA  UserModules     (ROM,REL,CON)
.LITERAL
EXPORT USB_D0_C1_I0_IN_RPTS:
  TD_START_TABLE 3
  TD_ENTRY  USB_DS_RAM, _ID0_RPT_SIZE, _ID0_RPT_PTR, NULL_PTR ; ID0 unused
  TD_ENTRY  USB_DS_RAM, _SM_RPT_SIZE, _SM_RPT_PTR, _SM_RPT_STS_PTR ; ID1
  TD_ENTRY  USB_DS_RAM, _LG_RPT_SIZE, _LG_RPT_PTR, _LG_RPT_STS_PTR ; ID2
.ENDLITERAL

ENDIF ; WIZARD_DEFINED_REPORT_STORAGE
```

ブートローダ USB ダウンロードプロトコル

次の図には、2 つのサンプルダウンロードレコード（最初と最後）が示されています。これらのレコードは、USB マスタとブートロードになるスレーブの間で転送される実際のデータから構成されます。レコードの書式を下記に示します。

Figure 5.　ブートローダ開始コマンドと第 1 データブロック

Packet data (BULK OUT):

FF 38 00 01 02 03 04 05 06 07 00 00 00 00 00 00 — Enter bootloader FF, 38
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Status data (BULK IN):

20 20 02 00 00 00 00 AA 00 00 00 00 00 00 00 00 — Status response
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Packet data (BULK OUT):

FF 39 00 01 02 03 04 05 06 07 00 4E 00 30 30 30 — Write block command FF, 39
First half of the data block
30 30 7E 30 30 7E 30 30 30 7E 30 30 30 7E 30 30
30 30 30 30 30 30 30 30 7E 30 30 30 28 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Status data (BULK IN):

20 20 03 00 00 4E 00 AA 00 00 20 00 00 00 00 00 — Status response
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
30 30 30 30 30 7E 30 30 7E 30 30 30 7E 30 30 30

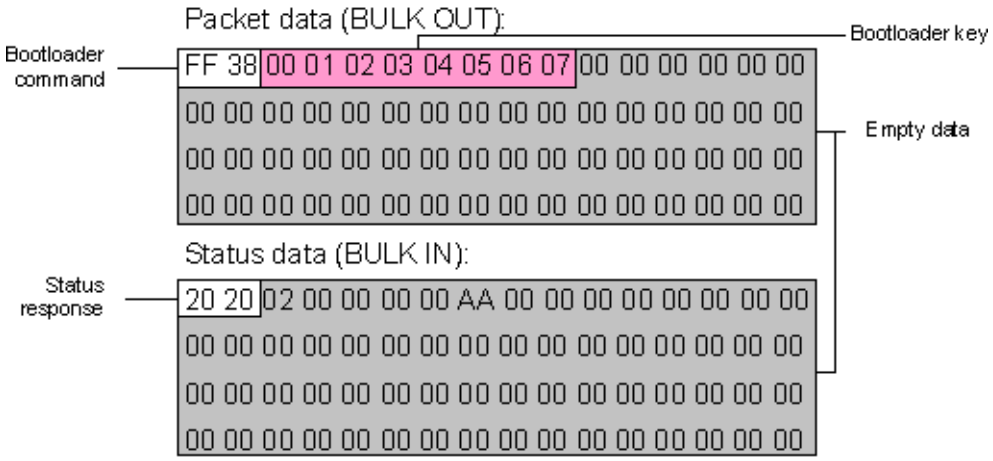
Packet data (BULK OUT):

FF 39 00 01 02 03 04 05 06 07 00 4E 01 7E 30 30 — Write block command FF, 39
Second half of the data block
30 7E 30 30 30 7E 30 30 30 7E 30 30 30 30 30 30
30 30 30 30 30 30 30 30 30 30 30 30 DB 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Status data (BULK IN):

20 20 03 01 00 4E 00 AA 00 00 40 00 00 00 00 00 — Status response
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
30 30 30 30 30 7E 30 30 7E 30 30 30 7E 30 30 30

ブートローダへの各コマンドには、それぞれブートローダからの応答に従います。次の図は、ブートローダ開始コマンドの書式を示しています。



第 1 行はブートローダコマンド FF38、すなわちブートローダ開始から開始されています。このコマンドの次にはブートローダキーがあります。すべてのブートローダコマンドには、ブートローダキーが発行されます。ブートローダは適切なキーを持たないコマンドは無視します。ブートローダキーは Bootloader_Key パラメータで設定できます。その他のブートローダコマンドには次のようなものがあります。

コマンド	意味
FF38	ブートローダ開始
FF39	書き込みブロック
FF3A	Flash 検証
FF3B	ブートローダ終了
FF3C	チェックサム更新

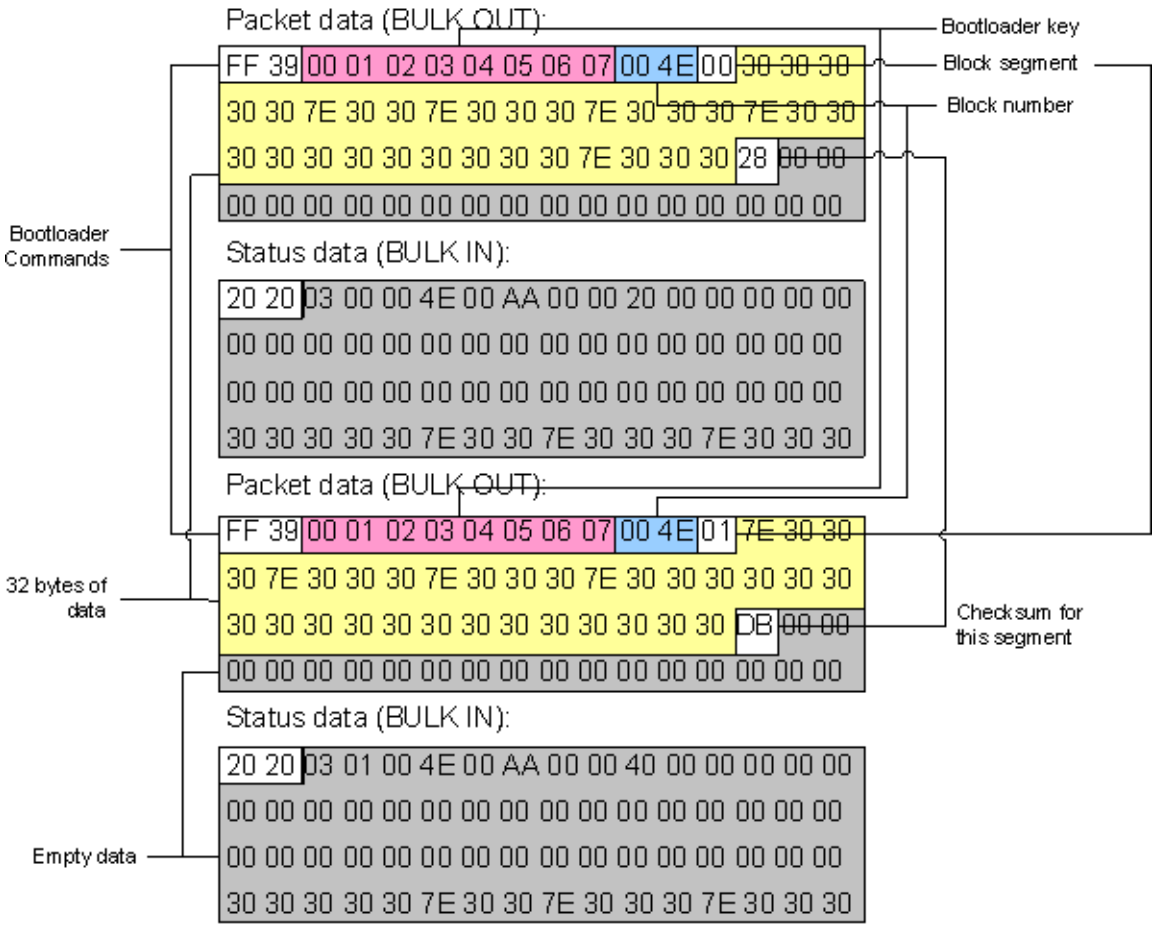
ブートローダへの各コマンドには、それぞれブートローダからの応答に従います。送信されたコード 0x20 は、ブートローダが正常に開始したことを示します。その他の応答ステータスには次のようなものがあります。

コード	意味
0x20	ブートロードモード（成功）
0x01	ブート完了 OK
0x02	画像検証エラー
0x04	Flash チェックサムエラー

コード	意味
0x08	Flash 保護エラー
0x10	一般的チェックサムエラー
0x40	無効なブートローダキー
0x80	無効なコマンドエラー

ブートローダ書き込みブロックコマンド

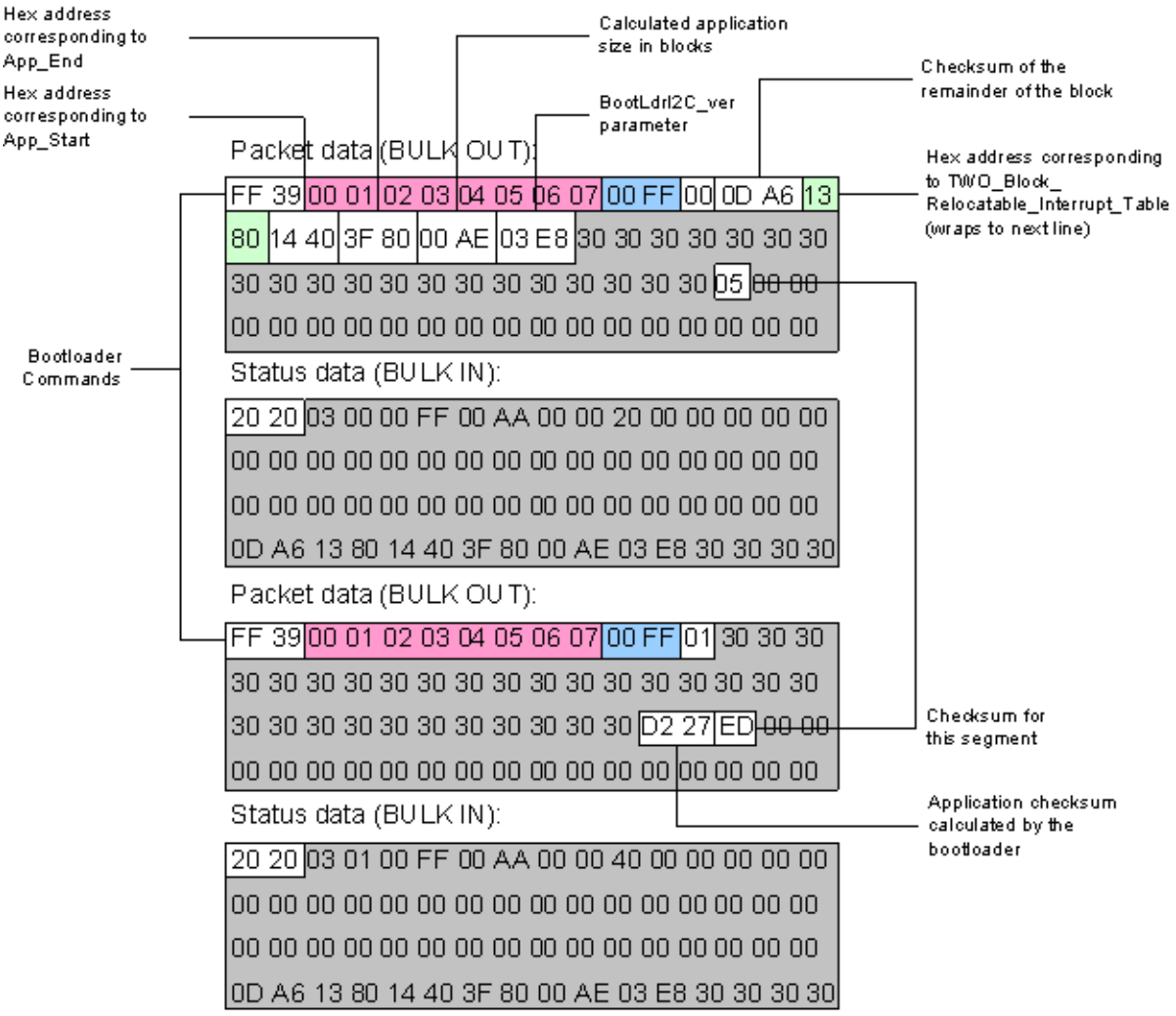
ブートローダに送信されるほとんどのコマンドは、書き込みブロックコマンドです。書き込みブロックコマンドはすべて同じ書式になっています。各 64-byte ブロックが 2 つの 32-byte パケットに分割されます。各コマンドはスレーブからのステータス反応を要求します。次の図は、64-byte ブロックの転送を示したものです。



第 1 パケットの第 1 行は、書き込みブロックコマンドとブートローダキーから構成されており、次に転送されるブロック番号が続きます。各ブロックとも 2 つに分割されるため、ブロック番号の次にはブロックセグメント番号が続きます。第 1 セグメントの場合は 0x00、第 2 セグメントの場合は 0x01 です。第 1 行の最後の 3 バイト、第 2 行の 16 バイト全部、第 3 行の最初の 13 バイトで有効な 32 バイトのデータを構成し、次にセグメントデータのチェックサムが続きます。ブロックの残りは 64 バイトにセグメントをパッドする空データです。

ステータスの応答は、2 度転送されたステータスバイトとセグメントを 64 バイトにパッドする 62 バイトの空データから構成されています。

ブロックの第 2 セグメントの書式は、第 1 セグメントと同一です。すべての転送されたデータブロックはこの書式ですが、最後のブロックだけ例外です。最後のブロックは、チェックサムとブートローダ操作用のその他の必要データから構成されています。最後のデータブロックの書式を次の図に示します。



この例では、チェックサムブロックが最後のレコードに含まれています（注：この例でのブロック番号は 0x00ff ですが、最後のブロックは 0x00ff でなくともかまいません）。

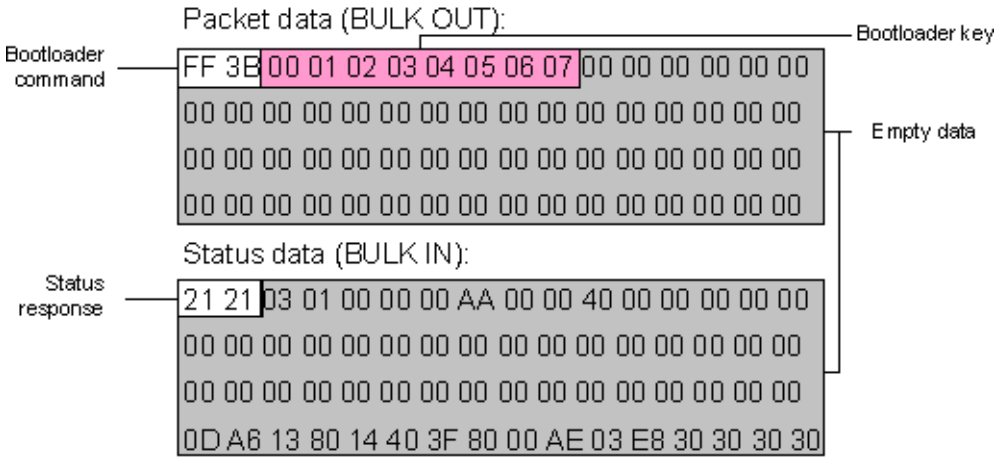
第 1 行は、前述のレコードと同様に、ブートローダ書き込みブロックコマンド、ブートローダキー、ブロック番号、ブロックセグメントで構成されています。次の 2 バイトは、ブロックの残り用のチェックサムで、この例では 0x0DA6 です。第 1 行の最後のバイトはと第 2 行の最初のバイトには、TWO_Block_Relocatable_Interrupt_Table パラメータ用にブロック 0x4E から計算された 16 進アドレスが含まれます。

第 2 行は、ブロック 0x45 から計算された App_Start ユーザ モジュールの 16 進アドレスを示す 2 バイト値を含みます。次の 2 バイトは、ブロック 0xFE から計算された App_End ユーザ モジュールの 16 進ア

ドレスである 2 バイト値です。その次には、ブロックのアプリケーションサイズである 2 バイトが続きます。この行にある実際のデータ値の最後の 2 バイトは、BootLdrI2C_ver パラメータからのブートローダ式番号です。行の残りと第 2 行のほとんどは、空データスペースです。セグメント用のチェックサムが、その他のパケットで占有したものと同じパケット位置を占有しています。パケットの残りは空スペースです。

チェックサムブロックの第 2 パケットはその他のパケットと同様に開始しますが、そこに含まれるデータは第 3 行のアプリケーションチェックサムとセグメントチェックサムだけです。

チェックサムブロックの直後に、ブートローダ終了コマンドが続きます。



ブートローダ終了コマンドは、ブートローダ終了コマンド 0xFF3B とブートローダキーで構成されています。

最後のパケットは最終ステータス反応です。

改訂履歴

バージョン	作成者	説明
1.30	DHA	jmp 命令周りの .Literal/.Endliteral 指令を削除 USB_cls_hid.asm における USBFS_bGetProtocol と USBFS_UpdateHIDTimer 関数用の SECTION と .ENDSECTION ディレクティブ削除 .

Note PSoC Designer 5.1 により、全ユーザ モジュール データシートが改訂されます。このセクションは、現在および以前のユーザ モジュール バージョン間の差を高レベルに説明するものです。