I2C ブートローダ データシート BootLdrI2C V 2.00

Copyright © 2008-2011 Cypress Semiconductor Corporation. All Rights Reserved.

リソース	PSoC® ブロック				API メモリ （バイト数）		ピン
	CapSense®	I2C/SPI	タイマ	コンパレータ	フラッシュ	RAM	
CY8C20x96, CY8C20x66, CY8C20x46, CY8C20x36, CY8C20336AN, CY8C20436AN, CY8C20636AN, CY7C643/4/5xx, CY7C60413, CY7C60424, CY7C6053x; CY8CTST200, CY8CTMG200, CYONS2xxx, CYONSFN2xxx, CYONSTB2010, CYONSTB2011, CY8CTMA30xx							
スレーブ （フル API サポート） 12 バイト		1			2134	12 バイト	2
スレーブ （API サポートなし） 5 バイト		1			2017	5 バイト	2
SW のみのスレーブ （API サポートなし） I2C リソース使用なし					2518	5 バイト	

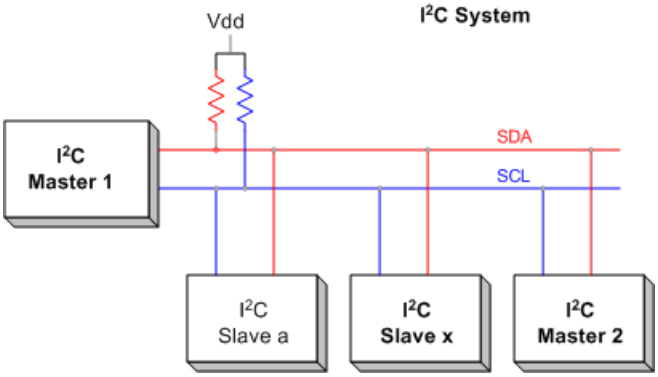
機能および概要

- 業界標準の フィリップス社 I2C バス互換インターフェイス
- インシステム プログラミング ピンではなく I2C システム バス経由で PSoC デバイスを再プログラミング可能

I2C ブートローダ ユーザ モジュールは、I2C インターフェース上で PSoC デバイスを再プログラムできるブートローダを実装します。PSoC デバイスは、デバイスに新しいコードをダウンロードするためのインシステム シリアル プログラミング インタフェース （ISSP）をすでに提供しています。しかし、このブートローダは、I2C のような業界標準の通信インターフェースによって、コードの更新を可能にします。このユーザ モジュールはフィールドでの再プログラムを必要とする、どんなデバイスにもお使い頂けます。ブートロードする情報は CY3240 （USB - I2C ブリッジ）やインシステム ホスト プロセッサのような I2C マスタ デバイスを通して送信することができます。

I2C ブートローダでは I2C ハードウェア ユーザ モジュールを使用する必要があります。これによって PSoC デバイス内の他の機能で I2C バスが使えなくなるということはありません。I2C ブートローダは、関連する機能に別の I2C アドレスを使用します。I2C ブートローダ向けのすべてのコードは、EEPROM の保護領域にプログラムされ、間違って上書きされることはありません。

Figure 1. I²C ブロック ダイアグラム



利用可能な構成

3 つの構成が利用できます。そのうち 2 つは、I²C リソースを使用するもので、アプリケーションで、同じ I²C リソースを使用する別のユーザ モジュールの使用はできません。3 つ目の構成は、割り込みを使用せず、I²C リソースを消費しないソフトウェアのみのブートローダ用です。これはアプリケーション デザイナにより高い柔軟性を提供します。ソフトウェアのみのユーザ モジュールでは、オペレーション アーキテクチャにこの他の改良も加えられています。これらは本データシートの末尾にまとめて記載されています。

クイック スタート

1. このデータシートを確認します。ブートローダ プロジェクトの実装を成功させるには、データシートの内容をよく理解することが必要です。
2. 正しく作動するために、このユーザ モジュールは、メモリ位置、およびプロジェクト全体をコンパイラで処理する方法を変えます。ブートローダプロジェクトを実施する前に、データシートの内容を理解することが重要です。
3. プロジェクトに、ユーザ モジュールを追加します。
4. ユーザ モジュールを配置します。ブートローダのみの I²C、またはブートローダを用いたフル I²C API サポートを選択します。
5. メニューバーの **Project （プロジェクト）> Settings （設定）>** ダイアログ ボックスを開き、**OK** をクリックしてプロジェクト パラメータを保存します。
6. ユーザ モジュール アイコンを右クリックし、**Boot Loader Tools （ブートローダツール）を選択します。**
7. **Get Files （ファイルを取得）を**クリックします。ファイル *boot.tpl*、*custom.lkp*、および *flashsecurity.example* 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。
8. 右上隅の X のアイコンをクリックして **Boot Loader Tools** （ブートローダツール）ウィザードを閉じます。
9. ユーザ モジュールには、大きな配列のメモリを使用したり、ページ 0 で大量の RAM を使用したりするものがあります。その結果、ページ 0 でメモリを使い切ってしまうこともありえます。このことが問題になる時は、Project （プロジェクト）-> Settings （設定）-> Compiler （コンパイラ）を選択して、HI-TECH のオプションボックスに 1 行加えてください。

   ```
   --AUTOBANK=1
   ```

   こうすれば C 言語自動変数をメモリ ページ 1 に移動することができます。使用しているデバイスの最大限度まで、メモリ ページを選択できます。--AUTOBANK オプションの詳細については、HI-TECH マニュアルを参照してください。

[+] Feedback

10. ソースコードを生成し、プロジェクトをコンパイルします。
11. 出力ファイルを見直して、*<project>.mp* と *<project>.hex* プロジェクトがどのようにビルドされているか確認します。
12. エラーなしでコンパイルしたプロジェクトが作成できたら、「Sample Firmware Code （サンプル ファームウェア コード）」セクションに移動します。サンプルで提供されているコードを変更し、適用してください。
13. PSoC Designer 5.1 には詳しいチュートリアルがありますのでご利用頂けます。ブートローダ チュートリアルにアクセスするには、メニュー バーから **Help（ヘルプ）> Documentation （ドキュメンテーション）> Supporting Documents**（サポート ドキュメント）をクリックします。

機能説明

ブートローダは、ユーザが UM パラメータを使用して定義できるフラッシュ メモリのセクション内にあります。このメモリ空間は、うっかり変更したり破損したりすることのないよう、書込み保護されています。リセット ベクターは、プロセッサがリセットされるとブートローダが実行されるように変更されます。

ブートローダは以下の処理を行います。

1. リセット時、ブートローダ は、フラッシュ ユーザ コードのチェックサムを計算し、フラッシュ メモリの最後の 2 バイトに書き込まれているチェックサムに一致するか、確認します。チェックサムが一致した場合、前回のプログラミングは成功したと判断し、ブートローダがユーザ コードの最初に分岐して、ユーザ コードが実行できます。
2. チェックサムが一致しない場合、ブートローダは、システムの重要なタスクを実行する（ファンをオンにするなど）カスタマイズ可能なユーザコードを実行し、そしてマスタからの 10-byte ブートローダ キーが送られてくるまで待機するブートローダ モードに入ります。前回のブートロードが失敗した場合、（たとえば、過渡電力が生じたなど）プログラムは、チェックサムの不一致により、ブートローダ モードに入ります。
3. マスタから有効なブートローダ キーを受け取ると、ブートローダは、マスタにフラッシュ イメージを受信する準備ができたことを通知するステータス バイトで応答します。
4. マスタは、エンコーディング バイトを含む可変長パケット内にある更新されたユーザ コードを送信します。
5. ブートローダは、フラッシュにユーザ コードを書き込みます。すべてのフラッシュ ページが正しく書き込まれると、ブートローダは、フラッシュ確認処理、次いでソフトウェアのリセットを実行し、ユーザ コードを開始します。

   Note I^2C マスタは、各ブロックへの書き込み作業後、デバイスがフラッシュのブロック書き込み操作を実行できるように 100 ms 待ってからブロックのステータス バイトを読み取る必要があります。デバイスがフラッシュ書き込みを待っている間は、割り込みに応答することはできません。128 バイトのフラッシュブロックを持つデバイスは、フラッシュブロックが完成するまで、パケットを集め、正しいパケットの構成を検証します。

ユーザ モジュールのブートローダ部分は、メモリ マップと主なコードの機能ブロックをデバイスの再プログラミングと互換性のある領域に構成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 Designer プロジェクトの構成とはかなり異なっています。デバイス アプリケーションの再プログラム中に、デバイスの最小機能要件を満たすためにメモリ マップを変更する必要があります。実際には、ブートローダが組み込まれたプロジェクトには、異なる機能をサポートする二つの独立したプログラムが含まれています。メモリ マップは以下に掲載しています。

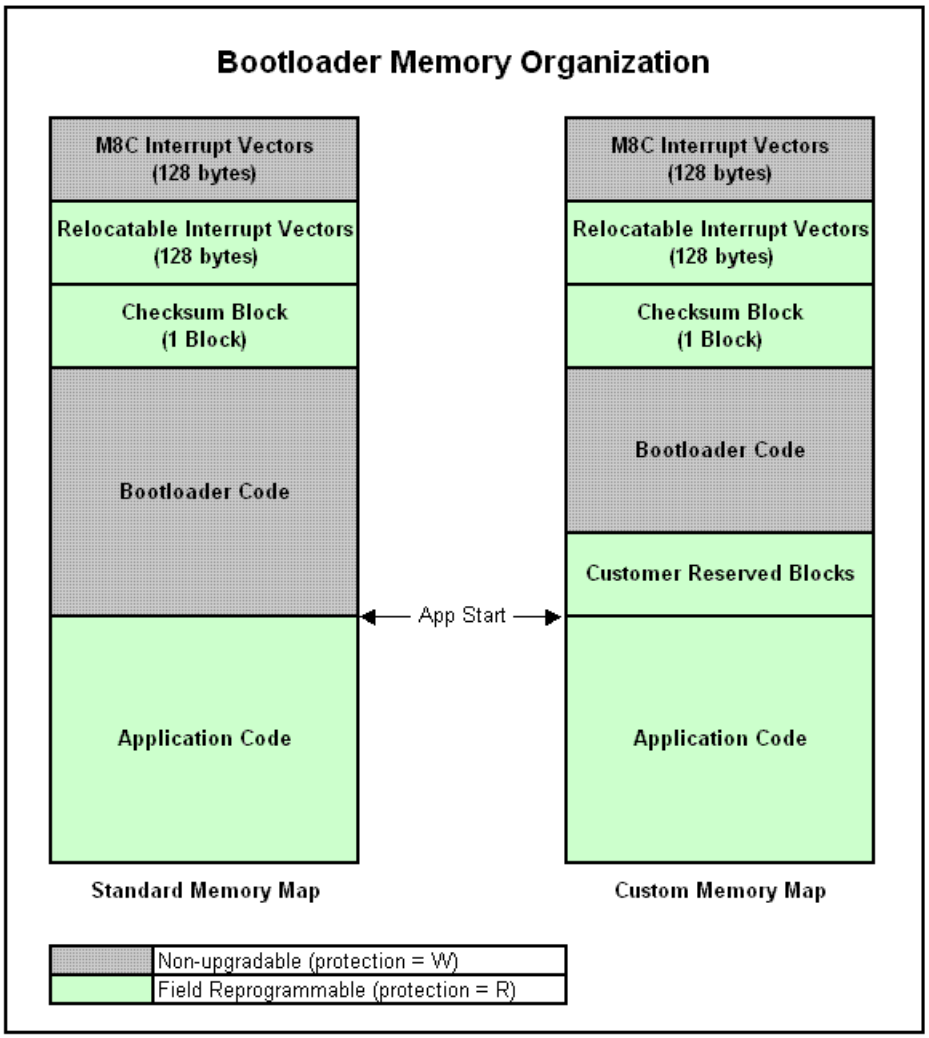
ブートローダが組み込まれたプロジェクトをいったん展開すると、灰色で強調表示されている、メモリの場所は二度と再プログラムされません。緑色で強調表示されているメモリの場所は、ブートローダを実行することにより変更できます。

動作理論

ブートローダでプロジェクトを作成するには、PSoC Designer の標準モデルに、非標準的な変更をいくつか加える必要があります。この作業を容易にするために、ブートローダ ユーザ モジュールは、ブートローダ プロジェクトの開発に役立つカスタマイズ ファイルと特殊ツールを提供しています。特殊ツールは、「Device Editor（デバイス エディタ）」ビューに切り替えて、BootLdrI2C ユーザ モジュール アイコンを右クリックすればアクセスできます。ブートローダの基本機能を示すユーザ モジュールのインストールの一環として、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に加え、ホスト アプリケーション サンプルが提供されています。この 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 プログラマ 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\BootLdrI2C\BootLoaderHostApp\...

Figure 2.　ブートローダ メモリマップ



概要

PSoC Designer は、標準化ファイル、パーツファミリ、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を作成し、API 定義を修正します。ブートローダを含むプロジェクトは、標準の PSoC Designer ™ プロジェクトとは大幅に異なるメモリ マップを必要とします。メモリ領域の選択は、設計の開始から完了までを通して、設計の中核的な決定要因になります。ブートローダを必要としないプロジェクトでは、コンパイラとリンカが単純に RAM と ROM を割り当てますが、ブートローダは新規アプリケーションのロード中にプログラムがクラッシュしないよう、RAM と ROM を特定の領域でグループ化しなければなりません。

メモリ レイアウトには、管理される ROM の 6 つの重要な領域が存在します。

- 第 1 は、ROM のブロック 0 と 1 です。これらのブロックには、非常に重要な割り込みベクトルと再起動ベクトルが含まれます。いかなる使用デバイスをもってしても、これらブロックへの読み取りアクセスを制御することはほぼ不可能なので、この二つのブロックを消去したり再プログラムしたりすることは決してできません。この ROM の最初の 2 ブロックは、変更してはならず、他の場所へ配置することもできません。
- 第 2 のメモリ領域は、再配置可能な割り込みテーブルです。このテーブルは一つまたは二つのブロックでできており、ブロックの数はデバイス アーキテクチャによって決まります。この領域には割り込みベクトルと汎用ベクトルが含まれており、ブートローダを使用して新規アプリケーションをロードする際に変更可能な割り込みエントリ、またはコード エントリのためのジャンプ　テーブルを提供します。
  たとえば、この領域はアプリケーション開始アドレスを含みます。ブートローダは、電源投入時にチェックサムが確認されたら、新規アプリケーションを開始するためのこのアドレスを使用することができます。この領域はブロック 2 と 3 に配置されています。ブートローダとアプリケーションが展開されたら、この領域のコンテンツは再書き込み可能ですが、場所を変えてはなりません。この領域の特徴は、次のセクションで説明するチェックサム領域の特徴と似ています。
- ここで定義される ROM の 第 3 の領域は、チェックサム領域です。この領域はブロック 4 に配置され、フォアグラウンド アプリケーションをダウンロードして認証するためにブートローダ ソフトウェアが使用する、重要なデータを含みます。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開始アドレスとブロックのサイズが含まれます。チェックサム ブロックの最初の 2 バイトは、このブロック自体のチェックサムです。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 ブロックの構造には、ブートローダが使用するデータ以外に、ユーザ独自のデータを定義するスペースが含まれています。この構造は、C 言語の構造体の定義として公開され、ブートローダ ユーティリティによって使用されるデータを変更したり、ブロック内で再配置したりしない限り、変更できます。
- 定義が必要な第 4 のメモリ領域は、ブートローダ コード自体を含む領域です。この領域はブロック 5 から始まります。ブートローダを含むプロジェクトまたはデバイスがいったん展開されると、この領域を再プログラムすることおよびフィールド アップグレードすることはできなくなります。
- 第 5 の領域は、カスタマー データのために確保されています。この領域は、ブートロード経由でアップグレードしている間、維持していなければならない設定データを含んでいることがあります。メモリ マップで示されているように、この領域はオプショナルです。カスタムデータがない場合は、標準メモリマップを使用することもできます。
- 第6番のメモリ領域はアプリケーション領域です。ここにはアプリケーション イメージがあります。「Bootloader Code（ブートローダ コード）」のコード サイズは拡張可能なため、この領域の開始アドレスは調整可能です。「Appliaction_Start_Block」パラメータ（プロパティ ウィンドウ内にある）を使えば、ユーザが適宜、アプリケーション開始アドレスを設定することができます。すべての残存メモリは、通常この領域に占められています。

ユーザのアプリケーションに、ブートロード プロセス中を含め、常に動作していなければならないようなコードがある場合、ブートローダ ユーザ モジュールの設計により、これに対応できる十分なカスタマイズが可能になっています。これを達成するために最適な方法は、アセンブラ AREA ディレクティブを使用して、ブートローダの ROM 領域にこのコードを追加することです。ブートローダ プロセス中にユーザのコードにより使用される RAM は、ブートローダに対して定義した RAM 領域に追加する必要があります。

ユーザ モジュール パラメータでのメモリ領域の定義

BootLdrI2C ブートローダ ユーザ モジュールでは、メインプログラムを ROM のどこに配置するかをカスタマイズするためのパラメータを用意しています。ユーザ モジュールのデフォルトは、通常、初期設定として作動するようになっています。プロジェクトのコンパイルが完了するまで、この設定を使用します。プロジェクトをコンパイルしたら、プログラムのメモリマップと .hex 出力ファイルを見て、プログラム構造を最適化する方法を決定します。パラメータを再構築し、誤ってメモリ領域の競合が生じてしまったら、参照できる有効なメモリ マップがなければ、正しい場所を決定することは困難になる可能性があります。

ブートローダ ユーティリティ

ブートローダ ユーザ モジュールは、フォアグラウンド（プライマリ）アプリケーションと共存するための完全なサブシステムアプリケーションを提供します。デバイスを起動またはリセットすると、常にブートローダ ユーティリティが呼び出されます。ブートローダは、システムの起動時に呼び出されると、フォアグラウンド アプリケーションの ROM 領域上でチェックサムを計算することで、フォアグラウンド アプリケーションを認証します。計算されたチェックサムは、チェックサム ブロックに格納されているチェックサム（アプリケーションを使用して作成されたもの）と比較されます。2 つのチェックサムが同等の場合、ブートローダ妥当性確認機能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。二つのチェックサムが等しくない場合、ブートローダは、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 待機ループに入ります。また、ホストがデータを送信できるように、ブートローダが I^2C サブシステムをイネーブルにします。ホスト システムがこのインタフェースの有効なことを認識すると、独自のアプリケーション セットの実行を選択できます。アプリケーションをターゲットにダウンロードするための 2 つのダウンロード ファイル フォーマットが、自動的にユーザ モジュール ツールによりサポートされます。一つ目は <project_name>.txt 、二つ目は <project_name>.dld という名称のついた出力ファイルです。二つのダウンロード ファイルはそれぞれ、異なったデモンストレーション ツールによりサポートされています。

ダウンロードの方法

1. .txt ファイルは、CY3240 USB-I^2C ブリッジ キットを使えばダウンロードできます。AN45683 で解説しているように、関連ツールは .txt ファイル フォーマットをダウンロードするために使用できます。さらに詳しい説明と、CY3240 USB-I^2C ブリッジ の使用については、AN2352 をご覧ください。この手法の説明は、本書の付録に記載されています。

2. アプリケーションをダウンロードするための二つ目の方法もサポートされています。これは前述の方法より複雑ですが、完成品のためにカスタム I2C ダウンロード アプリケーションを開発するにはより参考になるでしょう。ホスト アプリケーションの簡単な説明を以下に示します。サンプル アプリケーションとソース コードは、PSoC Programmer 3 のインストール ディレクトリ内で提供されています。

<install_path>\Cypress\Programmer\3.xx\Bootloaders\BootLdrI2C\BootLoaderHostApp\...

I^2C ブートローダの有効性を説明するには、2 つのアプリケーションが必要です。1 つ目は PSoC 27000 ベースのアプリケーション（完成した PSoC プロジェクトを含む）で、組み込みブートローダのダウンロード レコードを含むブートローダ アプリケーションを RS-233 通信から I^2C パケットに変換すること

ができます。この PSoC プロジェクト用に提供されたソース コードは、通常は他の PSoC デバイスのアーキテクチャに簡単に適応できます。

2 つ目のアプリケーションは Microsoft Visual Studio （マイクロソフト ビジュアル スタジオ）アプリケーションの、I²C Bootloader Host （ブートローダ ホスト）で、インストーラおよび修正のためのソースコードと共に提供されます。これは、ダウンロードファイル <fiilename>.dld を読み取り、解析し、前述の PSoC プロジェクトに送信することができます。Microsoft Visual Studio 2005 で使用できるこのアプリケーションのために、ソース コードも提供されています。このアプリケーションはデモンストレーション目的でのみ提供されており、量産利用や再販を意図するものではありません。

Note このアプリケーションを使用するには、ファイル mscomm32.ocx をコンピュータにインストールしなければならない場合もあります。このファイルはマイクロソフトのウェブサイトで、Windows/accessories（アクセサリ）/command prompt（コマンド プロンプト）ウィンドウおよび以下のコマンドを使用して、ダウンロードし、インストールできます。

> regsvr32 mscomm32.ocx

ファイルは *regsvr32.exe* Windows のインストール フォルダ windows/system32 (winXP) にあります。

ファイル *mscomm32.ocx* をマイクロソフトのウェブサイトからダウンロードするには、ファイル名「mscomm32.ocx」を検索します。

ブートローダ ツール

ユーザ モジュール アイコンを右クリックしてアクセスするショートカット メニューから、いくつかのツールを使用できます。

特別バージョンの boot.tpl と custom.lkp を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、[ツール] > [デフォルトブートファイルの復元] を選択します。BootLdrI2C ユーザ モジュール を削除すると、ユーザ モジュール アイコンからはデフォルトのブート ファイルを復元するオプションを使用できなくなりますが、PSoC Designer のメイン メニューのツール タブからはアクセスできます。

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 ツール選択画面での入力時に、プロジェクト コードが生成され、プロジェクト全体の完全なコンパイルが実行されます。実行の結果生成される hex ファイル上でチェックサムが計算され、ユーザ モジュールに格納されたチェックサムと比較します。チェックサムが一致しない場合は、メッセージが表示されます。必要に応じて新しいチェックサムを再計算し、格納することもできます。ビルドまたはコンパイル エラーがブートローダ ツールで呼び出される自動生成およびビルドで発生し、hex ファイルが正常に作成されない場合、エラーは報告されますが、PSoC Designer のビルド ダイアログにはエラー デバッグ情報は表示されません。生成およびビルドが自動化インタフェースから呼び出される場合、エラーの報告は行われません。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。3 つのタイプのダウンロードファイルが生成されます（<project_name>.txt、<project_name>.dld、<project_name>.iic ）。<project_name>.txt と <project_name>.dld はブートローダのみ用 I2C またはフル I2CAPI サポートオプション用に生成されます。<project_name>.iic は、SW のみブートローダオプション用に生成されます。これらのファイルには、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 デモンストレーション アプリケーションは、このファイルを読み取り、ブートローダが組み込まれている作業中のプロジェクトにダウンロードすることができます。これらのダウンロードはいずれも、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。

Note 現在、<project_name>.txt ファイルは本データシートに記載されているデバイスにのみ対応しています（他のデータシートに記載がある場合を除く ）。

[+] Feedback

チェックサムの半自動生成

プロジェクトがエラーなしに構成･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（デバイス エディタ）」ビューの I²C ブートローダ ユーザ モジュール アイコンを右クリックし、「**Bootloader Tools**（ブートローダ ツール）」を選択してアクセスできるユーティリティのうちの 1 つを使用して作成します。アプリケーション チェックサム（事前に計算されたもの、またはデフォルト）は、非表示のユーザ モジュール パラメータとして格納され、ブートローダ ツール メニュー ページが呼び出されるとすぐに、以前のチェックサムはすべて、すぐに現在の出力 output\<prj_name>.hex ファイルで計算されたチェックサムと照らし合わせて確認されます。当然のことながら、コンパイルが正常に行われなければチェックサムは生成されません。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。したがって、2 回目のコンパイルが必要になります。

提供される特殊ファイル

いくつかの重要なファイルが、「**Bootloader Tools**（ブートローダ）」メニューにアクセスして「**Get Files**（ファイルを取得）」を選択すると得られます。デバイス固有の boot.tpl ファイルは、custom.lkp、および事前定義されている flashsecurity.txt ファイルと共にメインのプロジェクト ディレクトリに配置されています。これら各ファイルのオリジナル バージョンは、プロジェクト バックアップ ディレクトリに配置されています。以下に各ファイルの目的を簡単に説明します。

Boot.tpl. – このファイルには、割り込みベクター テーブルの再配置可能な定義および再配置不可能な定義、および標準 boot.tpl ファイルで指定されている固定された場所ではなく、ROM の再配置可能な領域で指定されるデバイス固有のブート セットアップが含まれています。

Custom.lkp – ソースが生成されると、ユーザ モジュール パラメータで定義されているように、主なコード ブロックの自動生成された ROM 領域が、custom.lkp ファイルに入力されます。以下に示す custom.lkp ファイルのコード ブロックは変更しないでください。

- -bSSCParmBlk – フラッシュ処理中に使用される重要な特定の RAM を含みます。
- -bBootloader
- -bBLChecksum
- -bUserAPP – 最後の 3 行のいずれかを変更すると、プロジェクトが正しい custom.lkp ファイルを検出できないことを示すエラー ダイアログが表示されます。

コード生成中、custom.lkp ファイルの最後の 3 行はそれぞれ、コード生成ソフトウェアに制御され、再書き込みされます。最後の 3 行内またはそれ以降に加えられた変更は、エラーの原因となるか、単に失われます。custom.lkp ファイルの残りの部分も変更することができます。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 行をコメント化することができます。これにより、リンカは自動的にコードを配置できるので、アプリケーション コードのサイズ要件を決定する際に役立つことがあります。

HTLinkOpts.lkp – ソースが生成されると、ユーザ モジュール パラメータで定義されているように、主なコード ブロックの自動生成された ROM 領域が、HTLinkOpts.lkp ファイルに入力されます。HTLinkOpts.lkp ファイルのコード ブロックは変更しないでください。

- -L-ACODE... & -L-AROM... 行は、ROM 全体のサイズを示すデータを含みます。
- -L-PPD_startup... は、ブートローダ固有の ROM 領域の位置を示すリンカ ディレクティブを含みます。
- -L-P
- -L-Pbss0= 最後の数行のいずれかを変更すると、プロジェクトが正しい HTLinkOpts.lkp ファイルを検出できないことを示すエラー ダイアログが表示されます。

コード生成中、HTLinkOpts.lkp ファイルの最後の数行は、コード生成ソフトウェアに制御され、再書き込みされます。最後の 3 行内またはそれ以降に加えられた変更は、エラーの原因となるか、単に失われます。

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であり、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、手動による変更が必要な場合があります。なお、このファイルは PSoC Designer によって直接使用されるわけではありません。何らかの理由でデータのないファイルのためにプロジェクトを更新またはタグ付けする場合、Flashsecurity ファイルを上書きするのは不都合になることがあります。それは、開発者が後で繰り返し変更する必要が出てくるからです。しかしここで提供される flashsecurity.example ファイルは、必要に応じて編集も名前の変更も可能です。

Flashsecurity.txt – これは PSoC Designer によって提供されるデフォルト ファイルです。このファイル内のデータは .hex ファイルに追加され、内部 ROM メモリへのアクセスの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は、ブートローダは機能しません。読み取りおよび書き込み保護はプログラムされている PSoC にビルトインされているため、ブートローダを最初に展開する前に、このファイルを正しく構成する必要があります。

I²C 割り込み処理

標準的な BootLdrI2C ユーザ モジュールは、オプションで I²C 割り込み処理モジュールのフォアグラウンド コピーを提供します。このコードは、I²C スレーブを動作させるために要する API と共にアップグレード可能なメモリ内に配置されます。ブートローダ自体も、ブートローダにアドレス指定される I²C データを処理する内部ユーティリティを維持しています。これは、再書き込みされるコードを実行するという問題を解決します (これは好ましいプログラミング慣行ではありません)。

パラメータのブロック入力

メモリパラメータはすべて、0x00 ~ 0xFF 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 (ブロックの最大数は 0x4f ~ 0x1FF ですが、これはデバイスファミリによって異なります)。これは、メモリ アドレスを入力するのに最も便利なフォーマットではありませんが、部分的なブロック アドレスが、メモリ マップの異なる領域に誤って割り当てられるのを防ぎます。PSoC デバイスには、64 バイトのフラッシュブロックのみを格納できるものと 128 バイトブロックを使用できるものがあります。ブロックを使用することは、プロジェクトコードの異なるセクション間の境界維持に現在使用できる方法のうち、簡単なものの 1 つです。

PSoC のパーツはほとんど、ブロックサイズが 64 バイトです。本データシートに記載されている PSoC デバイスは 128 バイトブロックです。以下は重要事項です。1. ブートローダはすべて、フラッシュに書き込む必要があります。2. PSoC は、1 ブロックごとにしかフラッシュに書き込めません。したがってブートローダ アプリケーションにとっては、メモリは書き込むべきブロックの集まりであると考えたほうが良いでしょう。

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 進出力ファイルは各行に絶対アドレスを含みます。問題になっているデバイスのブロックサイズ (0x40/0x80) とは関係なく、16 進出力ファイルはコードを、 1 行につき 64(d)/0x40 バイトの行に分割します。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。128byte ブロックのデバイスでは、hex ファイルの 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、作業中のパーツのフラッシュブロックサイズに慣れてください。

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たアプリケーションは、デバッグが難しい場合があります。このため、ブートローダ ユーザ モジュール ファイル内でさらに調整することができます。これらは、ファイル BootLdrI2C_Bootloader.inc に含まれており、このファイルは次の等式を含んでいます。

```
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)
```

BOOT_TIMEOUT 式を使用すると、ユーザは、ユーザのコマンドがブートローダを呼び出した後、ホストから通信を受信しない場合にタイムアウトとなるコードを長くしたり、短くしたり、無限にしたりできます。この定義は、ホスト アプリケーションを開発またはデバッグする場合に役立つことがあります。

2 番目の定義は、チェックサム ブロック内部のチェックサムの使用を制御します。この式を 0 に設定すると、チェックサム ブロック内部に含まれるチェックサムは確認されません。ユーザ モジュール パラメータに定義されているように、ユーザ アプリケーション領域全体に対するチェックサム確認は、この場合でも実行されます。

DC 電気的特性と AC 電気的特性

デバイスの電気的特性については、PSoC デバイスのデータシートを参照してください。

配置

このユーザ モジュールは I2C/SPI 通信ブロックを消費します。ユーザ モジュールを配置する際には、**ブートローダのみ用 I2C** と **ブートローダを用いたフル I2CAPI サポート**のうちから選択できます。I^2C の使用目的がデバイスのブートローディングのみである場合は、**ブートローダのみ用 I2C** を選択します。API はブートローダが使用する機能のみに縮小されていて、フラッシュと RAM の使用も低減されています。

パラメータおよびリソース

デフォルトパラメータは、情報の提供のみを目的としたものです。ユーザのプロジェクト内のデフォルト設定は、使用されているパーツのブロックサイズに合わせて調整できます。または、コード領域の適切なサイズを提供するために、すでに調節してある場合もあります。プロジェクトのコンパイルとテストが完了したら、ユーザ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.　デフォルト パラメータ

Full API Support option

Parameters - BootLdrI2C_1

Name	BootLdrI2C_1
User Module	BootLdrI2C
Version	1.1
Slave_Addr_HEX	0x1
Boot_Loader_Addr_HEX	0x0
Read_Buffer_Types	RAM ONLY
Communication_Service_T	Interrupt
ApplicationCode_Start_Blo	0x14
Bootload_when_CKSUM_I	ENABLE_(deployment)
BootLoaderKey	0001020304050607
Flash_Program_Temperatu	-40C
Ignore_N_I2C_Prefix_Byte	2
BootLdrI2C_ver	0x1000
I2C Clock	100K Standard
I2C_Pin	P[1]0-P[1]1

SW Only option

Parameters - BootLdrI2C_1

Name	BootLdrI2C_1
User Module	BootLdrI2C
Version	1.1
Boot_Loader_Addr_HEX	0x0
I2C Clock	100kHz
I2C_Pin	P1[0]-P1[1]
Interrupt_out	Disable
Interrupt_out_Port	None
Interrupt_out_Pin	None
ApplicationCode_Start_Blo	0x14
Bootload_when_CKSUM_I	ENABLE_(deployment)
Run_App_if_CKSUM_OK	ENABLE
BootLoaderKey	0001020304050607
Flash_Program_Temperatu	-20C
Ignore_N_I2C_Prefix_Byte	2
BootLdrI2C_ver	0x1000

上図には、デフォルトパラメータの目的の例が挙げてあります。ユーザモジュールのソースコード内で、これらのパラメータは定期的に更新されることがあり、例とは違っている可能性もあります。

I^2C マスタによる使用を説明するために、すべてのバッファ名が書き込まれます。たとえば、I2Cs_pRead_Buf は、I^2C マスタが読み取るデータを含んだ RAM 内の場所を意味しています。

Slave_Addr_HEX

これは Slave （スレーブ）および MultiMasterSlave （マルチ マスタ スレーブ）パラメータです。スレーブ モードでスレーブまたは MultiMasterSlave をアドレス指定するために、I^2C マスタが使用する 7-bit スレーブアドレスを選択します。有効な選択肢は、0x00-0x7F からです。これはアドレスの上位 7 ビットなので、実際のアドレスはコード内部で 2 倍に見えます。

Boot_Loader_Addr_HEX

これは Slave （スレーブ）および MultiMasterSlave （マルチ マスタ スレーブ）パラメータです。スレーブ モードでスレーブまたは MultiMasterSlave をアドレス指定するために、I ^{2} C マスタが使用する 7-bit スレーブアドレスを選択します。有効な選択肢は 0-7Fh からです。これはアドレスの上位 7 ビットなので、実際のアドレスはコード内部で 2 倍に見えます。

Read_Buffer_Types

データ読み取りのためにサポートされるバッファのタイプを選択します。二つの選択肢があります。RAM ONLY （RAM のみ）または RAM OR FLASH （RAM かフラッシュ）です。RAM ONLY を選択すると、フラッシュ ROM の直接読み取りをサポートするために必要なコードと変数が削除されます。RAM OR FLASH は、RAM バッファまたはフラッシュ ROM バッファのいずれかを読み取って、データをマスタに送信するためのコードと変数をサポートする場合に選択します。API 呼び出しのサポートをイネーブルにして、RAM またはフラッシュ読み取りバッファを使用するには、RAM または FLASH を選択します。

Interrupt_out/Interrupt_out_Port/Interrupt_out_Pin

(SW のみブートローダのみ) ブートローダが 「enter-bootloader」または 「run-app」コマンドを待っている間、構成可能な入力パルスは有効です。ブートローダに入ると、各ブロックが転送された後フラッシュが書き込まれている間、パルスは低になっています。パルスのデューティサイクルと期間は、取り込みファイルを介して構成できます。本ユーザ モジュール データシートの SW のみブートローダの説明を参照してください。

Communication_Service_Type

Note ソフトウェアのみブートローダでは利用できません。

このパラメータを使用すると、割り込みベースのデータ処理方式とポーリング方式のいずれかを選択できます。割り込みベースの方式では、事前に定義したバッファに対して転送が開始されます。それからバックグラウンドで、可能な限り迅速にデータがバッファに入力されたりバッファから出力されたりします。データの移動を処理する ISR ルーチンが含まれています。ポーリング データ処理方式を選択した場合は、データをいつ移動するか決められます。ポーリング方式を実装するには、関数 BootLdrI2C_Poll() を定期的に呼び出さねばなりません （正確なインスタンス名については、I2C.h ファイルを参照のこと)。ポーリング関数を呼び出すたびに、1 バイトが転送されます。他の I^{2}C 関数も同様に使用します。ポーリング通信方式は、割り込みレイテンシが非常に重要である （かつ、非同期通信割り込みが問題を引き起こす可能性がある）状況で使用できます。また、データの転送タイミングを完全に制御したいという場合にも使用できます。ポーリングの欠点は、I^{2}C ステートマシンをイネーブルにした場合、各バイトの後、ポーリング関数が呼び出されるまで、バスが自動的にストールされることです。ポーリング関数は、I^{2}C の Slave および MultiMasterSlave 実装でのみ使用できます。Single Master 実装は、バイト単位のデータ転送をサポートする API 関数を提供します。

Clock （クロック）I2C

I^{2}C インタフェースを実行するクロック速度を指定するものです。クロックレートには 3 つあります。

- 50 k 標準
- 100 k 標準
- 400 k 高速 （ CPU_Clk_speed が 6 MHz を上回る場合)

起動時に無効なチェックサムが検出されると、ブートローダのデフォルトクロック速度は 100Khz です。これは、ブートローダ asm を永久的に修正することにより、カスタマイズできます （タグ UserCode_BODY6 を検索)。

I2C_Pin

ポート 1 から I²C 信号のために使用されるピンを選択します。PSoC Designer が自動で行うため、ピンの正しいドライブ モードを選択する必要がありません。

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コードの最初のブロックです。このコードは、ブートロード可能および書き込み可能でなければなりません。このパラメータはブートローダ ツールが、.dld ファイルのために処理するコードのブロック、およびチェックサムを計算するコードのブロックを決定するためにも使用します。ブートローダ ユーティリティがアプリケーション チェックサムを自動的に確認する場合、この変数は、使用するためにチェックサム ブロックに伝播されます。

このデフォルト値は前述の図に示されています。

Bootloader_when_CKSUM_fails

開発の目的では、アプリケーションのデバッグ時はチェックサム機能を無効にするほうが便利かもしれません。アプリケーションのチェックサムがチェックサムブロックに保存されているものと適合しない場合、このパラメータは、起動時にブートローダに自動挿入される機能を無効にするために使用できます。

デフォルト値は Enable_(Deployment) です。

Bootloader_Key

これはブートローダ アプリケーションに送信したトランザクションに付加されたキー値で、追加確認手順を表します。これはブートローダグレードユーティリティが手違いで起動しないようにするためです。

デフォルト値は 「0001020304050607」です。

Flash_Program_Temperature_Deg_C

これは、デバイスを再プログラミングする場合に予想される典型的なプログラミング温度です。このパラメータで指定し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。PSoC は、温度がより低い場合に、より強力なフラッシュ書き込みを実装します。パラメータ設定よりも大幅に低い温度でブートロードを行うと、メモリ保持力が落ちる可能性があります。したがって、このパラメータの値から 20 度以上異なる温度でブートローダが決して動作しないように予防措置を講じなければなりません。詳細は、サイプレスのデバイス仕様書を参照してください。

Run_App_if_CKSUM_OK

(SW のみブートローダのみ) このパラメータは、チェックサムが認証された場合、アプリケーションが自動的に実行されるのを許可します。この機能が有効になっていると、ブートローダに入る唯一の方法はアプリケーションの BootLdrI2C_bootLoaderStart() 関数を用いることです。ソフトウェアのみブートローダのフローチャートを参照してください。

Ignore_N_I2C_Prefix_Bytes

Note RS-232 to I²C トランスレータ プロジェクトは、デバイスに 2 バイトのプリフィックスを送信します。したがって、提供されているデモンストレーション アプリケーションを使用する場合、このパラメータの正しい設定は、2 になります。このパラメータは、I²C に基づく、一部の SM バス プロトコルの場合も使用されます。このパラメータを使用すると、可変的な接頭語の バイト数を無視するようにブートローダを設定できます。

SW のみブートローダと併用すると、このパラメータには特別な意味が付加されます。SW のみブートローダは、プロトコルバイトそのものとして、これらのバイトのうち 2 つを使用します。これは、ホストプロトコルバイトが、SW のみブートローダで使用される 2 バイトに追加されなければならないということを意味します。SW のみブートローダは、2 バイト m 未満の接頭語は許可しません。

BootLdrI2C_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によって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は使用できます。これは、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。

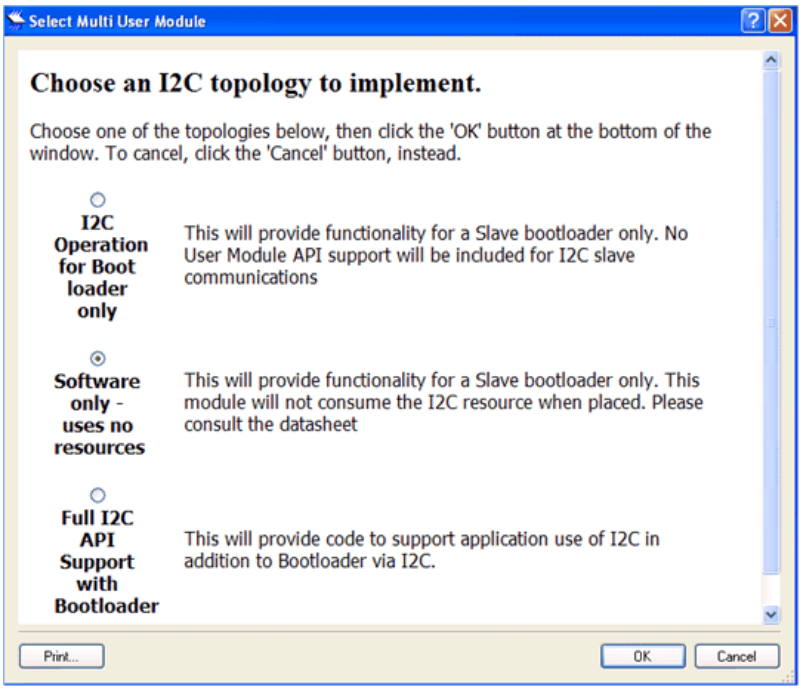
I²C トポロジ選択オプション

ブートローダ ユーザ モジュールを配置するとき、ブートローダプロジェクトのために、どの I²C トポロジを実装するか決定しなければなりません。

ブートローダのみ I²C 操作 --- ブートローダへの I²C 通信を提供するオプションです。I2C スレーブ通信には、ユーザ モジュール API サポートは含まれません。ユーザのアプリケーションが、ブートロード以外の目的で I²C 通信を使用しない場合は、このオプションを選択します。

SW のみ（リソースの使用なし）--- ブートローダのみへの I²C 通信を提供するオプションです。I²C スレーブ通信には、ユーザ モジュール API サポートは含まれません。初めのオプションとは異なり、このオプションは I²C リソースを消費しません。ブートローディングのみに I²C 通信を使用し、その他の目的に I²C を使用する場合、このオプションを選択します。

ブートローダによるフル I²C API サポート --- このオプションは、I²C 経由のブートローダに加えて、I²C のアプリケーション使用をサポートするコードを提供します。ブートロード以外の目的で、ユーザのアプリケーションの I²C 通信を使用する予定があるときは、このオプションを選択します。



よくある問題

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、およびコンパイラ

ブートローダアプリケーションを使用している間は、PSoC 開発環境を変更することは避けてください。これは PsoC Designer とブートローダユーザ モジュールの更新、コンパイラの変更を行わないことも含みます。

最初は、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ます。しかし、ブートロード可能なシステムが実装されると、アプリケーションセクションのみが再プログラミング可能となります。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。理想的には、開発環境内のコンポーネントの全バージョンに整合性が望まれますが、ブートローダでは整合性の維持が基本です。開発環境の互換性を変更しなければ、リスクは回避できます。

マルチコンパイラは PSoC Designer にサポートされていますが、だからといって 1 つのコンパイラでコンパイルしたブートローダが、別のコンパイラでコンパイルしたアプリケーションと互換性があるという想定はできません。RAM 割り当てに関する想定には一つの決定的な違いがあります。RAM ページングの実装は、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。

HI-TECH コンパイラの RAM 割り当てに関する問題

ユーザ モジュールには、大きな配列のメモリを使用したり、ページ 0 で大量の RAM を使用したりするものがあります。

- ユーザ モジュール メモリのデフォルト位置はページ 0 です。
- ブートローダは、ページ 0 の最低位置にメモリがあることを必要とします。
- HI-TECH ローカル変数および割り込み RAM のデフォルトはページ 0 です。

これらすべてが、RAM のページ 0 上で場所をとろうとしているため、このページのメモリが足りなくなる可能性があります。これが問題なら、Project （プロジェクト）-> Settings （設定）-> Compiler （コンパイラ）を選択して、HI-TECH のオプションボックスに 1 行加えてください。

```
--AUTOBANK=1
```

こうすれば C 言語自動変数をメモリ ページ 1 に移動することができます。使用しているデバイスの最大限度まで、メモリ ページを選択できます。--AUTOBANK オプションの詳細については、HI-TECH マニュアルを参照してください。

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、ファイル globalparams.inc 内のグローバルパラメータ WATCHDOG_ENABLE にリンクしています。プロジェクトがウォッチドッグ タイマを使用する場合、グローバル パラメータにリンクしている条件付きでコンパイルされた コードが、ブートロード チェックサムおよびダウンロードの処理中に、自動的にウォッチドッグを設定します。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します。ウォッチドッグ タイマの実際の最小設定は、約 0.125s です。

Flashsecurity.txt の不正な設定

このファイルのデフォルト設定は、プロジェクトの作成時に設定されます。構成例はファイル「Flashsecurity.example」に記載されています。Flashsecurity.example は、「BootLoader Tools （ブートローダ ツール）」- 「Get Files （ファイルを取得）」ユーザ モジュール メニュー項目から得られます。マップは、最終的にブートロードされるすべての場所でフラッシュ書き込みできるものでなければ

なりません。一つのやり方は、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。どちらのやり方にしても、プロジェクトの最初に対処したほうが、後でデバッグするより早くすみます。実行可能なブートローダが使用するコードの領域を書き込み保護にするのは、ユーザの責任です。フラッシュ セキュリティを正しくマッピングできないと、システムの故障や非常に困難なデバッグ タスクの原因となる場合があります。

不正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 プロジェクトのメモリ マップは、従来のプロジェクトのものとは大幅に異なるため、再配置可能なコード開始アドレスは、通常、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のリンクを試みたときに、リンカによって生成された エラーの一般的な原因となります。このパラメータはメニュータブの **Project（プロジェクト）> Settings（設定）> Linker（リンカ）** にあります。ブートローダ コードが使用する最も高いブロックより少し大きくなるように、または ROM の未使用領域を占めるように 16 進数の絶対開始アドレスを計算します。ブートローダの I^2C バージョンの場合、通常はこの値に x8x900 を設定すれば十分です（他のパラメータのデフォルト値が使用されている場合）。

メモリのオーバラップ

再配置可能なコード開始アドレスを修正するには、先頭のセミコロンを使用して、custom.lkp ファイルの最後の 3 行をコメント化し、ファイルの再ビルドを試み、結果生成されるメモリ マップを検討します。メモリのオーバラップ問題は、出力ファイルの生成を妨げるため、診断が困難です。custom.lkp ファイルを変更すると、リンカはオブジェクト ブロックを配置し、これが、メモリがオーバラップする根本的な原因を修正する開始点を提供する可能性があります。

電源の安定性

電源ノイズ、グリッチ、電圧低下、遅い電源電圧上昇、および接続不良によって、フラッシュ プログラミング問題の診断が困難になる場合があります。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、フラッシュ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、電源投入時のフラッシュへのステータス書き込みがあります。使用モデル、およびフラッシュ操作中に電源供給状態が変更される可能性がある場合は十分に検討しなければなりません。電源の安定性が低いと、部品が機能しない一因となったり、低いフラッシュ保持力の原因となったりする場合があります。

アプリケーションがブートロードプロセスで完全に停止していない

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。

I^2C インターフェースを使用すると、複数のアドレスを認識できます。アプリケーションが不完全なオフの状態で、ブートローダへの通信を開始することがあり得ます。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を含めて、ブートローダを明確に挿入する手段を講じる必要があります。

新しいファイルのダウンロードによるデバイスの機能停止

ブートローダ ユーティリティに入る機能がないアプリケーションを構築する可能性もあります。特に、これは意図せずに行われがちです。たとえば、単純な while(1) を含む main{} 関数では、ループは決して戻らず、決してブートローダには入りません。したがって、実行開始後、再プログラムすることはできません (正しいチェックサムがあることが条件)。この問題に対応するには複数の方法があります。このユーザ モジュールにはデフォルトの方法は含まれていません。以下に、いくつか方法を提案します。

1. デバイスが最初に起動したときにブートローダがイネーブルにされる時間を許容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に入るようにコード内のあるポイントでテストを行う。このポイントとしては、スイッチ閉やポート ピンの Low または High 保持が考えられます。
3. I^2C アプリケーション リソースをイネーブルにし、デバイスがブートローダに入るようにする I^2C コマンドを作成します。一般的に、I^2C がメイン ルーチンによってイネーブルにされると、ブートローダ のアドレスはデバイスがブートローダに入るように設定できます。
4. 定期的に処理されない場合は、ウォッチドッグ タイマを利用してデバイスをリセットします。これは、WDT 割り込みがブートロード可能な状態を開始できるようし、上記の方法のいずれかと組み合わせることができます。ウォッチドッグ タイマのリセット状態からリセットする時には、ウォッチドッグ タイマに関連するステータス ビットを監視し、それがリセット状態の原因であるかどうかを判断できます。『Technical Reference Manual』を参照してください。
5. 二つのプロジェクトが開発されていますが、それぞれのブートローダは微妙に異なります。ブートロードは、デバイスのプログラミング *部分*が実行されていることを示唆することに留意してください。これは、二つの相互に再プログラム可能な各アプリケーションのブートローダの実装は同一でなければならないことを示唆しています。すべてのブートローダパラメータは同一でなければなりません。また再配置可能コード開始アドレスも同一でなければなりません。(これは最初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。<project>.lst ファイルを比較するという方法もあります。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に適合しなければなりません。ブートローダをフィールド アプリケーションに展開すると、その内部のコードを変更することはできません。将来のアプリケーションでも、相互に使用されるジャンプ ベクターの格納場所について " 合致 " させなければなりません。
6. ブートロード処理が失敗した場合は、PSoC ベースのトランスレータ アプリケーションがハングアップします。PSoC ベースのトランスレータが、可変的なソフトウェア ループの後、通信の試行を中断できるように設定できる if-def ベースのタイムアウトがあります。デバッグするために、このソフトウェア スイッチをオンまたはオフにすることができます。タイムアウト スイッチについては、プロジェクトのソース コードを検討してください。
7. 電源ノイズ、グリッチ、電圧低下、遅い電源電圧上昇、および接続不良。これらすべての電源に関する問題のせいで、フラッシュ プログラミングの問題の診断が困難になる場合があります。プログラムの実行は電源電圧上昇に関連して急速で、場合によっては、フラッシュのプログラミング中に、部品がまだ電源電圧を変えている場合が考えられます。一例として、電源投入時のフラッシュへのステータス書き込みがあります。使用モデル、およびフラッシュ操作中に電源供給状態が変更される可能性がある場合は十分に検討しなければなりません。電源の安定性が低いと、部品が機能しない一因となったり、フラッシュ保持力が低下する可能性があります。

アプリケーション プログラミング インタフェース

アプリケーション プログラミング インタフェース （API）ファームウェアは、複数バイトの転送の送受信をサポートする高レベルのコマンドを提供します。読み取りバッファは、RAM またはフラッシュメモリ内に設定できます。書き込みバッファは、RAM メモリにのみ設定できます。

ユーザ モジュールを配置するたびに、インスタンス名が割り当てられます。デフォルトでは、PSoC Designer プロジェクトで、このユーザ モジュールの最初のインスタンスに BootLdrI2C_1 を割り当てます。これは識別子の構文ルールに従った一意の値に変更できます。割り当てたインスタンス名が、全てのグローバル関数名、変数、および定数記号の接頭語になります。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 「BootLdrI2C」となっています。

Note ここでは、全てのユーザ モジュール API と同じように、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。関数を呼び出す場合、呼出し後に A と X の値が必要になるならば、必ず呼び出し前に A と X の値を保存してください。PSoC Designer のバージョン 1.0 以降では、効率を高めるために、この 「registers are volatile (レジスタの揮発性)」ポリシーが選択され、実施されています。C コンパイラは自動的にこの条件を処理します。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しなければなりません。ユーザ モジュール API 関数の中には、A と X を変更しないものもありますが、今後も変更されないという保証はありません。

ENTER_BOOTLOADER

説明

ブートローダを完全に設定し、新規アプリケーション プログラムのダウンロードを準備するルーチン。このルーチンは、いったん呼び出されると、タイムアウトまたはリセットが発生するまで戻りません。

GenericBootloaderEntry （汎用ブートローダ エントリ）の関数名 は、常に同じ物理 ROM アドレスに配置されるので、後からコンパイルされるアプリケーションも、やはりこの関数呼び出しを使用してブートローダに入ることができます。

C プロトタイプ :

```
void ENTER_BOOTLOADER(void);
```

アセンブラ :

```
lcall ENTER_BOOTLOADER
```

パラメータ :

なし

戻り値 :

なし

副作用 :

A および X レジスタは、今回、または今後、この関数を実装することによって変更される可能性があります。

BootLdrI2C_Start

説明

互換性のために提供されている空のルーチン。

C プロトタイプ :

```
void  BootLdrI2C_Start(void);
```

アセンブラ :

```
lcall BootLdrI2C_Start
```

パラメータ :

なし

戻り値 :

なし

副作用 :

A および X レジスタは、今回、または今後、この関数を実装することによって変更される可能性があります。

BootLdrI2C_DisableInt

説明

SDA 割り込みをディスエーブルにすることで、I^2C スレーブをディスエーブルにします。I2Cs_Stop. と同じように動作します。

C プロトタイプ :

```
void  BootLdrI2C_DisableInt(void);
```

アセンブラ :

```
lcall BootLdrI2C_DisableInt
```

パラメータ :

なし

戻り値 :

なし

副作用 :

A および X レジスタは、今回、または今後、この関数を実装することによって変更される可能性があります。

BootLdrI2C_EnableInt

説明

I²C 割り込みを有効にして、開始状態の検出を可能にします。以下のマクロを使って、グローバル割り込みのイネーブル関数を有効にしてください。M8C_EnableGInt.

C プロトタイプ :

```
void  BootLdrI2C_EnableInt(void);
```

アセンブラ :

```
lcall BootLdrI2C_EnableInt
```

パラメータ :

なし

戻り値 :

なし

副作用 :

A および X レジスタは、今回、または今後、この関数を実装することによって変更される可能性があります。

BootLdrI2C_Poll() および BootLdrI2C_BootLdr_Poll()

説明

Communication_ Service_Type パラメータに Polled が設定されている場合に使用します。この関数は、I/O 処理ルーチンへのユーザが制御する入力を提供します。Communication_Service_Type パラメータが Interrupt （割り込み）に設定されている場合、この関数は何も実行しません。

C プロトタイプ :

```
void BootLdrI2C_Poll(void);
void BootLdrI2C_BootLdr_Poll(void);
```

アセンブラ :

```
lcall BootLdrI2C_Poll
lcall BootLdrI2C_BootLdr_Poll
```

パラメータ :

なし

戻り値 :

なし

副作用 :

このルーチンが呼び出され、ステータス変数が更新されるたびに、一つの I²C イベントが処理されます。イベントは、エラー条件、I/O バイト、または、場合によっては、停止条件で構成されます。このルーチンの呼び出しの結果は、三つ考えられます。

1. 利用できるデータがない場合は何も実行されません。
2. 1 つが使用可能な場合は、アドレスまたはデータバイトが受信または送信されます。

3. 外部マスタが書き込み処理を完了すると、停止 「イベント」が処理されます。停止状態が書き込み処理の最後に処理される場合、ステータス変数だけが更新されます。停止状態が処理される時点で I²C バイトが保留中の場合は、I2CHW_Poll 関数を再度呼び出して処理する必要があります。

   I2CHW_Poll() が割り込みに設定されている場合、Communication_Service_Type 関数は何の効果も持ちません。バス上でスタート / リスタート条件およびアドレスが検出されると、I2CHW_Poll() 関数が呼び出されるまで、バスはストールされます。アドレスが NAK された場合は、別のスタート / リスタートおよびアドレスが検出されるまで、続いて送信されたバイトは無視されます。それ以外の場合は、I2CHW_Poll() 関数が呼び出されるまで、各データ バイトで I²C バスがストールします。

   Communication_Service_Type が Polled に設定されている場合、I²C データは I²C ハードウェアによってストールされます。受信データの場合、バスはバイトの最後に、および SCL （クロック）行を口ーに保つことで ACK （承認）応答 / NAK 応答が生成される前に、ストールされます。送信データの場合、バスは、ACK 応答 / NAK 応答ビットが外部で生成された直後にストールされます。

   これら二つの関数は、以下の制限の下、互換性を持ちます。ブートローダがアクティブで、外部アプリケーションが入ることができる場合は、I²C コマンド、API BootLdrI2C_BootLdr_Poll() を使用しなければなりません。ブートローダのアドレスではない I²C アドレスが検出された場合は、このアドレスはテストされ、さらにアドレスをテストするフォアグラウンド I²C 割り込みプロセスに伝えられます。ブートローダが非作動状態の場合は、BootLdrI2C_Poll() API を使用しても、I²C アドレスは、内部ブートローダのデータ プロセス ルーチンに提供されません。その代わりに、アドレスとそれ以降のデータが、フォアグラウンド ルーチンに直接伝えられます。

BootLdrI2C_Stop

説明

I²C 割り込みをディスエーブルにすることで、I2CHW をディスエーブルにします。

C プロトタイプ :

```
void  BootLdrI2C_Stop(void);
```

アセンブラ :

```
lcall BootLdrI2C_Stop
```

パラメータ :

なし

戻り値 :

なし

副作用 :

A および X レジスタは、今回、または今後、この関数を実装することによって変更される可能性があります。

BootLdrI2C_EnableSlave

説明

I2C_CFG レジスタで Enable Slave（スレーブをイネーブルにする）ビットを設定して I^2C HW ブロックの I^2C スレーブ関数をイネーブルにします。

C プロトタイプ：

```
void  BootLdrI2C_EnableSlave(void);
```

アセンブラ：

```
lcall BootLdrI2C_EnableSlave
```

パラメータ：

なし

戻り値：

なし

副作用：

A および X レジスタは、今回、または今後、この関数を実装することによって変更される可能性があります。

BootLdrI2C_DisableSlave

説明

I2C_CFG レジスタで Enable Slave（スレーブをイネーブルにする）ビットを解除して、I^2C スレーブ関数をディスエーブルにします。

C プロトタイプ：

```
void  BootLdrI2C_DisableSlave(void);
```

アセンブラ：

```
lcall BootLdrI2C_DisableSlave
```

パラメータ：

なし

戻り値：

なし

副作用：

A および X レジスタは、今回、または今後、この関数を実装することによって変更される可能性があります。

BootLdrI2C_InitWrite

説明

API は FullAPISupport（フル API サポート）にのみ使用できます。

BootLdrI2C_InitWrite ルーチンは、スレーブがデータ保存のために使用できるようデータ バッファ ポインタを初期化し、同じバッファのカウント バイトの値をゼロにします。

C プロトタイプ :

```
void  BootLdrI2C_InitWrite(BYTE * pBootLdrI2C_WriteBuf, BYTE  BootLdrI2C_Write_Count);
```

アセンブラ :

```
mov A, Write_Count
push A
move A, >pWriteBuf
push A
mov A, <pWriteBuf
push A
lcall BootLdrI2C_InitWrite
```

パラメータ :

pWriteBuf: RAM バッファの位置へのポインタ。Write_Count: 書き込みバッファの長さ。

戻り値 :

なし

副作用 :

A および X レジスタは、今回、または今後、この関数を実装することによって変更される可能性があります。

BootLdrI2C_InitRamRead

説明

API は FullAPISupport （ フル API サポート ）にのみ使用できます。

BootLdrI2C_InitRamRead ルーチンは、スレーブがデータを検索するために使用できるようデータ バッファ ポインタを初期化し、同じバッファのカウント バイトの値をゼロにします。

C プロトタイプ :

```
void BootLdrI2C_InitRamRead(BYTE * pBootLdrI2C_ReadBuf, BYTE  BootLdrI2C_Read_Count);
```

アセンブラ :

```
mov A, Read_Count
push A
move A, >pReadBuf
push A
mov A, <pReadBuf
push A
lcall BootLdrI2C_InitRamRead
```

パラメータ :

pReadBuf: RAM バッファの位置へのポインタ。Read_Count: 読み取りバッファの長さ。

戻り値 :

なし

副作用 :

A および X レジスタは、今回、または今後、この関数を実装することによって変更される可能性があります。

BootLdrI2C_InitFlashRead

説明

API は FullAPISupport（フル API サポート）にのみ使用できます。

BootLdrI2C_InitFlashRead ルーチンは、スレーブがデータを検索するために使用できるようフラッシュ データ バッファ ポインタを初期化し、同じバッファのカウント バイトの値をゼロにします。

C プロトタイプ：

```
void  BootLdrI2C_InitFlashRead(const BYTE * pBootLdrI2C_flashaddr,  unsigned int
BootLdrI2C_Read_CountHI);
```

アセンブラ：

```
mov A, >Read_Count
push A
mov A, <Read_Count
push A
move A, >pflashaddr
push A
mov A, <pflashaddr
push A
lcall BootLdrI2C_InitFlashRead
```

パラメータ：

pflashaddr: フラッシュ データ バッファの位置へのポインタ。Read_Count: 読み取りバッファの長さ。

戻り値：

なし

副作用：

A および X レジスタは、今回、または今後、この関数を実装することによって変更される可能性があります。

読み取りステータス ビットはクリアされます。

BootLdrI2C_bReadI2CStatus

説明

API は FullAPISupport（フル API サポート）にのみ使用できます。

I2CStatus 変数の値を返します。

C プロトタイプ：

```
BYTE  BootLdrI2C_bReadI2CStatus(void);
```

アセンブラ：

```
lcall BootLdrI2C_bReadI2CStatus ; Accumulator contains the status on return
```

パラメータ：

なし

戻り値：

bI2CStatus - ステータス データ

定数	値	説明
I2CHW_RD_NOERR	01h	データがマスタに読み取られました。ISR を通常に終了します。
I2CHW_RD_ OVERFLOW	02h	使用可能な量よりも多くのデータ バイトをマスタが読み取りました
I2CHW_RD_ COMPLETE	04h	読み取りが開始され、完了しました。
I2CHW_READFLASH	08h	フラッシュの場所で、次の読み取り操作が実行されます。
I2CHW_WR_NOERR	10h	マスタによってデータが正常に作成されました
I2CHW_WR_OVERFLOW	20h	マスタが作成バッファのために作成したバイトが多すぎます
I2CHW_WR_COMPLETE	40h	マスタの作成作業が新しいアドレスまたは停止信号により完了しました。
I2CHW_ISR_ACTIVE	80h	I^2C ISR がまだ終了しておらず、有効です。

副作用 :

A および X レジスタは、今回、または今後、この関数を実装することによって変更される可能性があります。

BootLdrI2C_ClrRdStatus

説明

API は FullAPISupport （ フル API サポート ） にのみ使用できます。

制御 / ステータス レジスタのステータス ビットをクリアしますが、バッファ アドレス、カウント、またはフラッシュ / RAM 読み取りビットは変更しません。

C プロトタイプ :

```
void  BootLdrI2C_ClrRdStatus(void);
```

アセンブラ :

```
lcall BootLdrI2C_ClrRdStatus
```

パラメータ :

なし

戻り値 :

なし

副作用 :

A および X レジスタは、今回、または今後、この関数を実装することによって変更される可能性があります。

BootLdrI2C_ClrWrStatus

説明

API は FullAPISupport （フル API サポート）にのみ使用できます。

制御 / ステータス レジスタのステータス ビットをクリアしますが、バッファ アドレス、カウント、またはフラッシュ / RAM 読み取りビットは変更しません。

C プロトタイプ：

```
void  BootLdrI2C_ClrWrStatus(void);
```

アセンブラ：

```
lcall BootLdrI2C_ClrWrStatus
```

パラメータ：

なし

戻り値：

なし

副作用：

A および X レジスタは、今回、または今後、この関数を実装することによって変更される可能性があります。

SW のみブートローダ （I²C リソース、割り込みの使用なし）ベータ

I²C ブートローダの新バージョンは 100% ソフトウェアベースです。動作中は割り込みは使用せず、配置されても I²C リソースを消費しません。これにより、アプリケーション内で使用可能なリソースが増えます。

SW のみブートローダの実用モデルは、リセットで実行され、新しいアプリケーションプログラムの読み込みを許可する独立したアプリケーションとして理解してください。アプリケーションの実行中は、ブートローダはほとんど目に見えません。

SW のみブートローダがサポートする機能

通信フォーマットは EZI2C プロトコルに適合します。読み込みトランザクションの場合：

<I2C_WRaddr>, <EZI2CSubaddr>, EZI2CSubaddr で書き込みを開始する < データ >,< ストップコンディション >

読み取りトランザクションの場合：

<I2C_RDaddr>, 前の書き込みで指定され、EZI2CSubaddr で読み取りを開始する < データ >,< ストップコンディション >

ブートローダは I²C リソースを使用しないため、I²C を使用するアプリケーションが簡単に実行できます。

チェックサムが OK の場合、アプリケーションを自動的に実行するオプション機能。

ブートローダのブートロード操作が開始したらアプリケーションの内容を消去するオプション機能。

チェックサム処理を迅速にするオプション機能。チェックサムの確認後、値がフラッシュに保存されます。これは一回ごとの起動で実行する必要がないことを示します。これにより、起動が速く行われるようになります。

ブートローダがコマンドを待っている間、構成可能な出力パルスは有効になっています。ブートローダに入ると、各ブロックが転送された後フラッシュが書き込まれている間、パルスは高になっています。このパルスはデューティサイクルで、周期の構成が可能です。

I²C アドレスとデータを認識する場合、ハードウェアの I²C バッファはソフトウェアのレイテンシを削除するために使用されます。

初回の起動では、ホストとの初回通信で状態反応がプリロードされます。

制限（SW のみブートローダ）

SW のみブートローダの複数インスタンスはサポートされません。複数インスタンスの場合、SW のみブートローダのユーザ モジュールが正しく作動することは保証されていません。

ブートローダがアクティブで、HW I²C バッファにデータが書き込まれると、状態を要求する前にホストは HW バッファの読み取りを待たなければなりません（約 10 mSec）。

SW のみブートローダが提供する API 関数は 1 つだけです。

初回 STATUS (16 バイト)

ブートローダが挿入されると、ホストへの状態応答がクエリにプリロードされます。状態は 16 バイトで構成されます。応答は、bootloader.inc ファイルの値を修正することで、エンドユーザがカスタマイズできます。

まずブートローダアドレスにサブアドレス 0 を書き込み、次に I²C バッファからデータを読み取ることにより状態を獲得します。

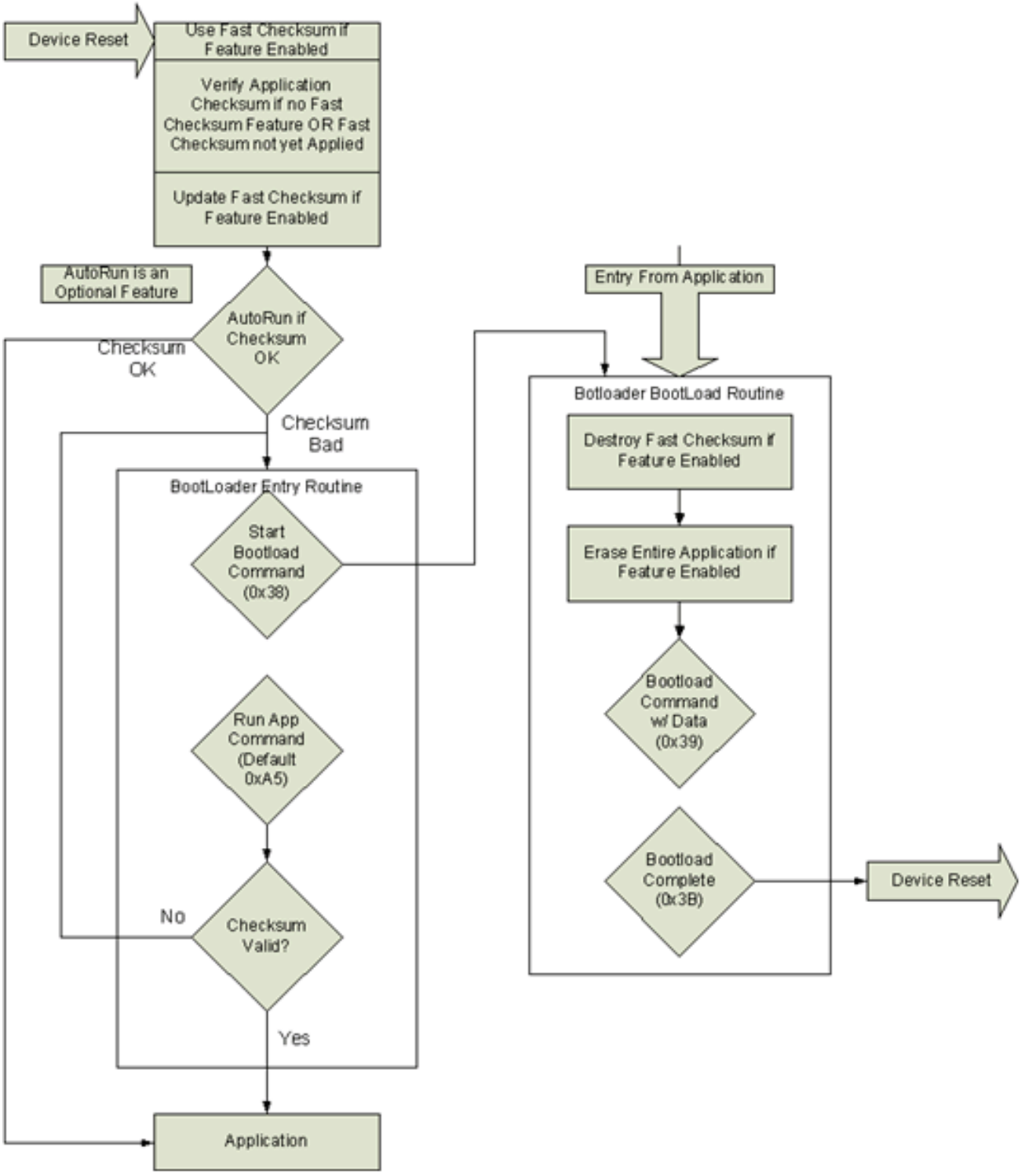
これらの値は下表で示すように初期化されます。

address	Name	Bit 7	Bit 6		Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Master/ PSoC Access
BL,00	HST_MODE	Always 0 (defined later by host)								W/R: 00
BL,01	BootLoaderStatus	0	0	0	BootLdr Mode	1, 1 during ERASE APP		Watchdog reset	Checksum valid	R/W:00
BL,02	BootErrorCode	Inalid Command	Invalid decurity Key	BootLoad Mode	Comm Cksum Error	Flash Protection Error	Flash Checksum Error	Image Verify Error	Boot Completed OK	R/W:00
BL,03	BL_VER_Bhi	Bootloader Version Defined by Bootloader (lmsb) (default 0x10)								R/W:00
BL,04	BL_VER_Blo	Bootloader Version Defined by Bootloader (lsb) (default 0x00)								R/W:00
BL,05	BL_VER_Ah	Bootloader Version Defined by Application (msb) (default 0xff)								R/W:00
BL,06	BL_VER_Alo	Bootloader Version Defined by Application (lsb) (default 0xff)								R/W:00
BL,07	DIP_VERhi	Design IP Version (msb) (default 0xff)								R/W:00
BL,08	DIP_VERlo	Design IP Version (lsb) (default 0xff)								R/W:00
BL,09	APP_IDhi	Application ID (msb) (default 0xff)								R/W:00
BL,0a	APP_IDlo	Application ID (lsb) (default 0xff)								R/W:00
BL,0b	APP_VERhi	Application Version (msb) (default 0xff)								R/W:00
BL,0c	APP_VERlo	Application Version (lsb) (default 0xff)								R/W:00
BL,0d	CID_0	Custom ID #0 (default 0xC1)								R/W:00
BL,0e	CID_1	Custom ID #1 (default 0xC2)								R/W:00
BL,0f	CID_2	Custom ID #2 (default 0xC3)								R/W:00

この表は、デフォルト値を示しています。エンドユーザは、「bootloader.inc」ファイルの「BL_VER***", "DIP_VER***」、「APP_***」、「CID_**"」の値を修正できます。しかし、「bootloader.inc」ファイルで、「BOOT_LOADER_DEFAULTS」を 0x00 に変更するときには、これらの値に 「TO_DO_Developer」変数を代入します。

SW のみブートローダの動作

Figure 4. SW のみブートローダ動作のフローチャート



SW のみブートローダのカスタマイズ

PSoC Designer は、コード生成器によるアップデートからコードのブロックを保護することを可能にします。ブートローダ全体が保護されます。そのため、ユーザはブートローダのコードをアップデートでき、ユーザによる変更はコード生成器によって上書きされません。このアプローチは、過去の問題を緩和するための妥協策です。

1. 実装されたブートローダは新しいバージョンにアップグレードしないでください。フィールドではブートローダの異なるバージョンを混合することができない場合があります。
2. 2. バグ修正や、改良、その他のカスタマイズ手法を用いてもかまいません。これがコードにあるときは必ず、このコードを保存してください。

取り込みファイル構成

一部の構成変更は、ファイル UM_Name_bootloader.inc 内の永続 EQU 値を変更することで、実行できます。

Bootloader.inc ファイルは、SW のみブートローダの機能の一部を構成するために使用されます。コードが再生成された際に変更されない構成の修正を加えることができる保護領域があります。

ブートローダ RUN_APP

このコマンドの値は、booloader.inc ファイルでデフォルト 0xA5 に設定されます。必要に応じて、取り込みファイルの定義を修正して、コマンド値を変更することもできます。

PULSE_COUNT: EQU 0xA00 (デフォルト) 「わたしは生きている」割り込みパルス周期の経験的調整 (合法値、0-65535)。

PULSE_COMPARE: EQU 0x10 (デフォルト) 「わたしは生きている」割り込みパルス幅の経験的調整 (合法値 0-255)。

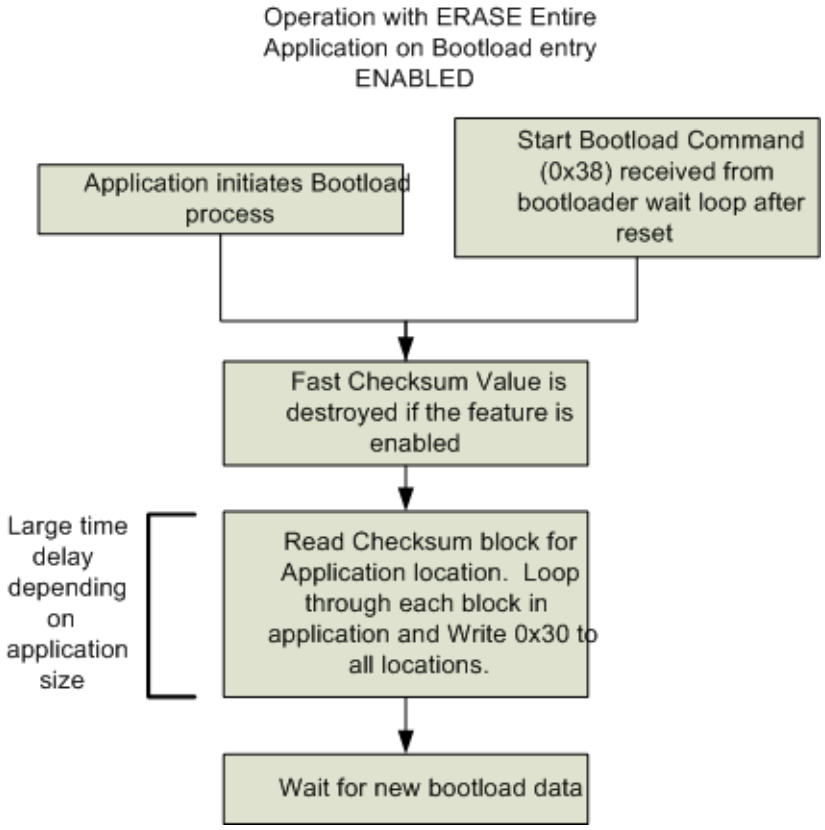
パルスはソフトウェアタイミングループを介して動作するため、以前の 2 つ値については、パルスは 0xA00 ループの 0x10 ループです (32/320)。

BOOT_TIMEOUT: EQU 00 (デフォルト 0x40) ブートローダループのある地点で、カウンタは初期化・減少して無限ループの発生を抑えます。

カウンタがゼロまで減少すると、ループから出ます。この値をゼロに設定すると、ループから出ることを阻止できます。これは、デバッグやブートロード処理の進行に役立ちます。

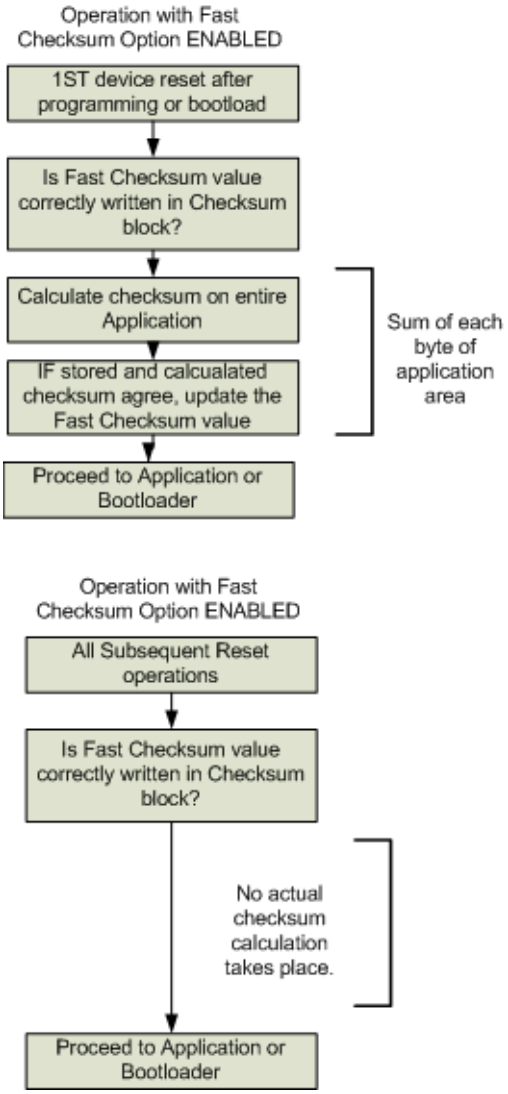
ERASE_APP_BEFORE_BOOTLOAD: EQU 0 (デフォルト 0/ オフ) ブートロード開始コマンドを受信すると、アプリケーション全体が消去されます。この場合、最初ブロックを削除するループに入り、次に値 0x30 を伴うアプリケーションスペースとして定義される各ブロックを初期化します (停止)。この操作は数秒かかる場合があります。アプリケーションを削除する間、デバイスは I^2C のトラフィックにいつも反応するとは限りません。I2C_Interrupt_out ピンは、デバイスがフラッシュに書き込まれる間、アサートされます。この機能を USB-I2C ブリッジとともに使用することは特に困難ですが、それはブリッジが固定期間分だけ遅延する能力しか持たないからです。アプリケーションの削除にかかる時間は様々です。最良の開発アプローチは、アプリケーションサイズで削除時間を計り、デバイスのブートロードに使用する遅延をカスタマイズすることです。ブートロードデータの転送にインテリジェントなホストを使用すると、処理を続ける前にピントグルを待つこともあります。

Figure 5.　ブートロード上のアプリケーションの削除



CHECKSUM_FAST_VERIFICATION: EQU 1 (デフォルト 1/ オン) チェックサムの確認は、3 MHz クロックパルスでは約 1.3 秒かかり、チェックサムまでの ROM スペースは約 30K 必要です。最初のチェックサム確認後にこの機能が有効にされると、チェックサムが通過したことを示すチェックサムブロックで値が更新されます。後続のチェックサムの確認は、この値が設定されているかどうかを見るためだけに行います。この操作は、確認時間を、3 MHz 命令クロックレートで約 260 uSec までに短縮します。

Figure 6.　高速チェックサムオプション



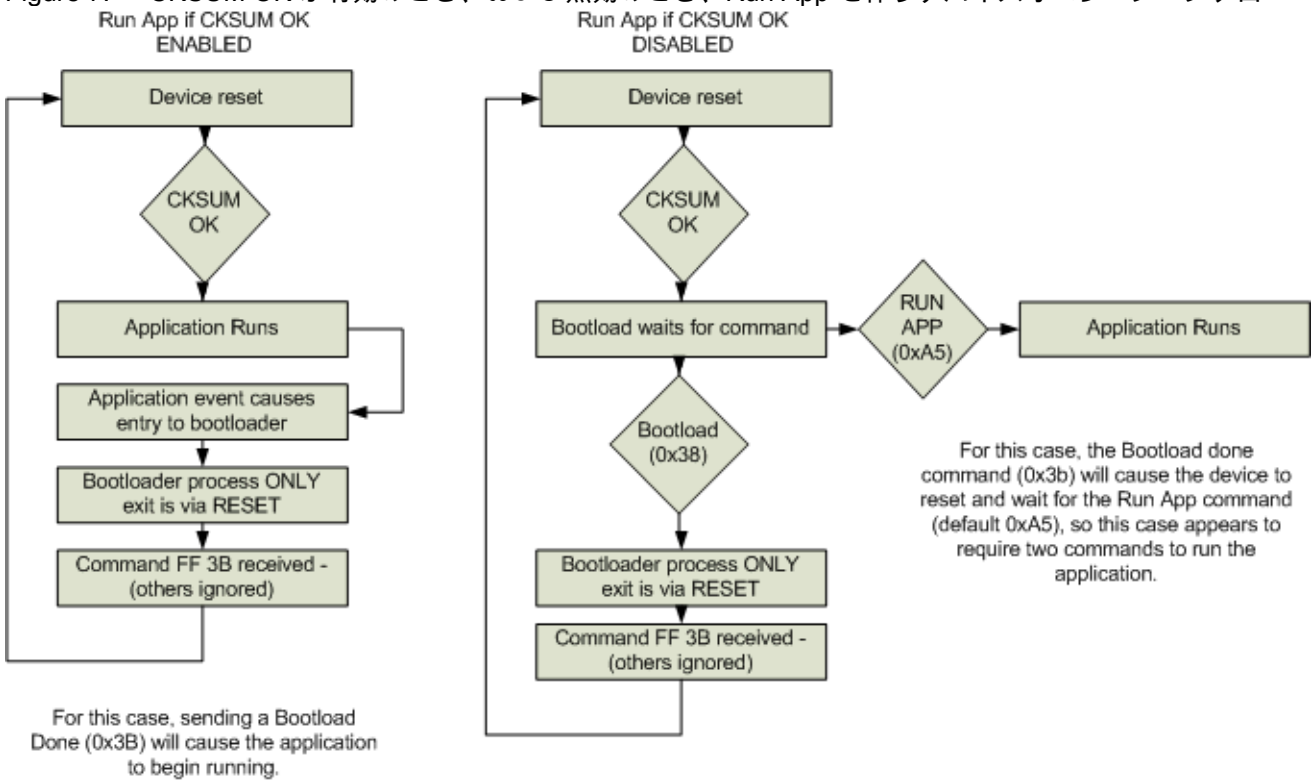
Run_App_if_CKSUM_OK: EQU 1 (デフォルト 1/ オン)

この機能を有効にすると、チェックサムが正しい場合には、起動時に自動的にアプリケーションが実行できます。この機能は、高速チェックサム確認機能とともに使用すると、デバイスをすばやく起動できるようになります。ブートローダは、前述の EnterBootloader() API を使用して挿入できます。この機能を使用すると、ブートローダを退去するコマンドシーケンスがわずかに変わり、ブートロード後のアプリケーションを開始させます。この機能によって可能となる操作の詳細については、下図を参照してください。

フローチャートは SW ブートローダ操作を説明しています。

左のケースは、オプションの 「Auto Run App on CKSUM OK」が有効になったことを想定してます。右のケースは CKSUM OK パラメータが無効な場合の Run App を想定しています。

Figure 7.　CKSUM OK が有効のとき、および無効のとき、Run App を伴うデバイスオペレーションフロー

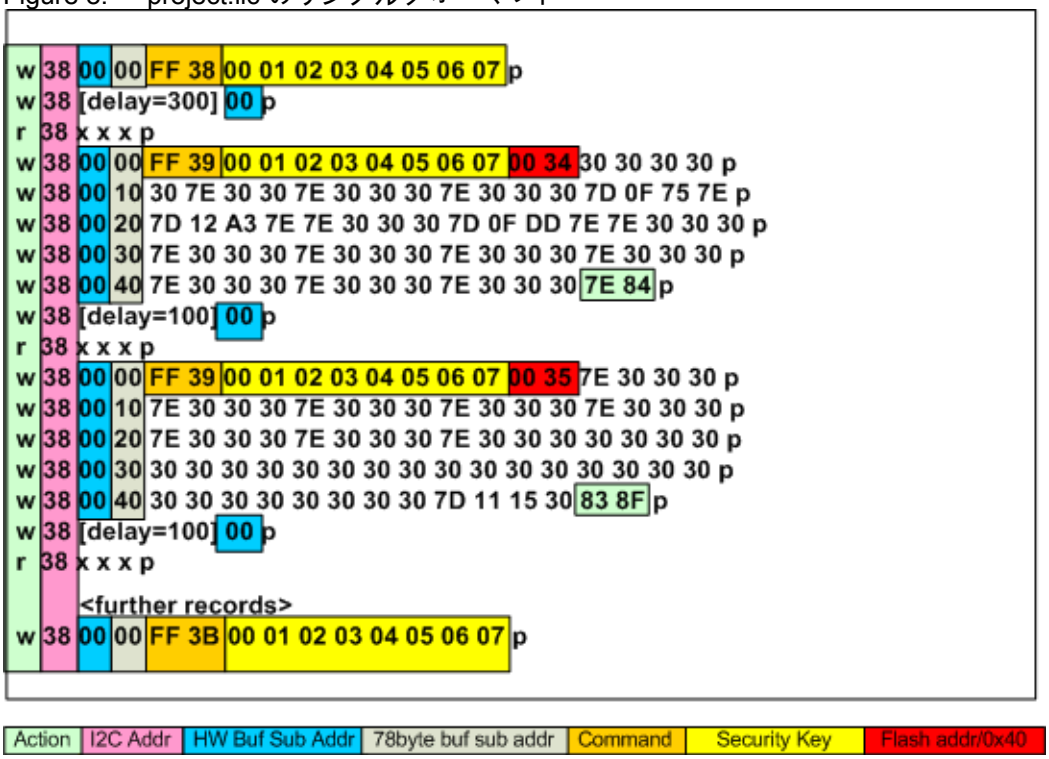


[+] Feedback

SW のみブートローダ用に作成されたダウンロードファイル

新しいフォーマットのダウンロードファイルが、SW のみブートローダ用に作成されました。このファイルは .txt ファイル出力に似ています。SW のみブートローダの出力ファイルは、project_name.iic という拡張子が付いています。

Figure 8. project.iic のサンプルフォーマット



Action	I2C Addr	HW Buf Sub Addr	78byte buf sub addr	Command	Security Key	Flash addr/0x40

ファームウェア ソースコードの例

SW のみブートローダの例

以下の図に示されている値を用いて、アセンブリ言語と C 言語のユーザ モジュール パラメータ例を設定します。

Figure 9. SW のみパラメータ設定

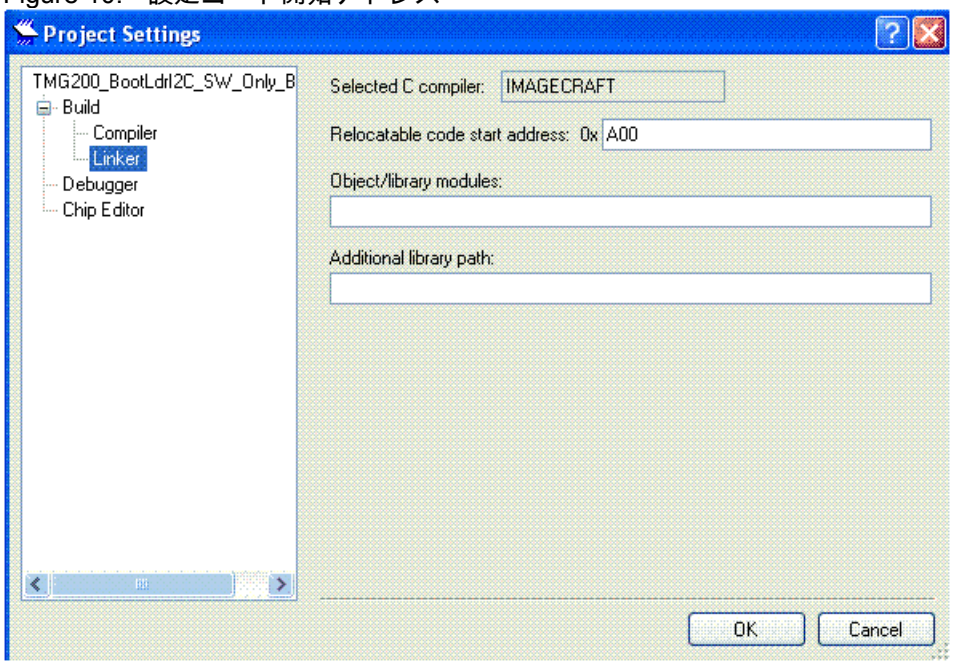
SW Only Bootloader parameters setting

Parameters - BootLdrI2C_1

Name	BootLdrI2C_1
User Module	BootLdrI2C
Version	1.1
Boot_Loader_Addr_HEX	0x0
I2C Clock	100kHz
I2C_Pin	P1[0]-P1[1]
Interrupt_out	Disable
Interrupt_out_Port	None
Interrupt_out_Pin	None
ApplicationCode_Start_Blo	0x14
Bootload_when_CKSUM_I	ENABLE_(deployment)
Run_App_if_CKSUM_OK	ENABLE
BootLoaderKey	0001020304050607
Flash_Program_Temperatu	-20C
Ignore_N_I2C_Prefix_Byte	2
BootLdrI2C_ver	0x1000

コード開始アドレスが正しく設定されていることを確認します（IMAGECRAFT コンパイラのみ）。

Figure 10. 設定コード開始アドレス



次に、ブートローダ ユーザ モジュールが正しく作動していることを示すサンプルを 2 つ挙げます。このテストでは、2 つの LED がそれぞれ P0[0] と P0[1] に接続されています。まず、P0[0] ピン上の LED を点灯する例 1 を伴うデバイスをプログラムします。次に、P0[1] ピン上の LED を点灯する例 2 を伴うデバイスをブートロードします (USBtoIIC ブリッジ GUI を使用)。ブートローダ ユーザ モジュールが正しく作動していると、P0[1] の LED はオンで、P0[0] の LED はオフになります。これは、本データシートのクイックスタートセクションに従って、プロジェクトにブートローダ ユーザ モジュールを配置したことを想定しています。

```
//-------------------------------------------------------------------------
// Example 1
//-------------------------------------------------------------------------

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


void main(void)
{
    PRT0DM0 = 0xff;
    PRT0DM1 = 0x00;

    while(1)
    {
        PRT0DR = 0x01;
    }
}


//-------------------------------------------------------------------------
// Example 2
//-------------------------------------------------------------------------

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


void main(void)
{
    PRT0DM0 = 0xff;
    PRT0DM1 = 0x00;

    while(1)
    {
        PRT0DR = 0x02;
    }
}
```

ブートローダを用いたフル I²C API サポートの例

ここに、C 言語で書かれた「ブートローダのみ向けの I²C 操作」と「ブートローダを用いたフル I²C API サポート」オプション用 I²C ブートローダ ユーザ モジュールの実装例を示します。

Figure 11. フル I²C API サポート用のパラメータ設定

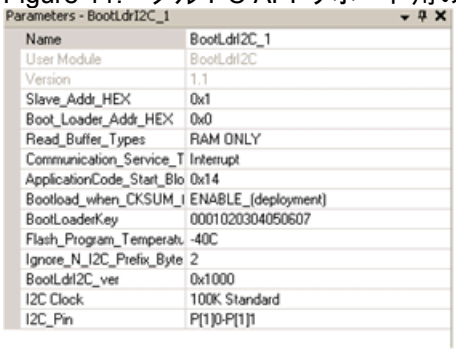
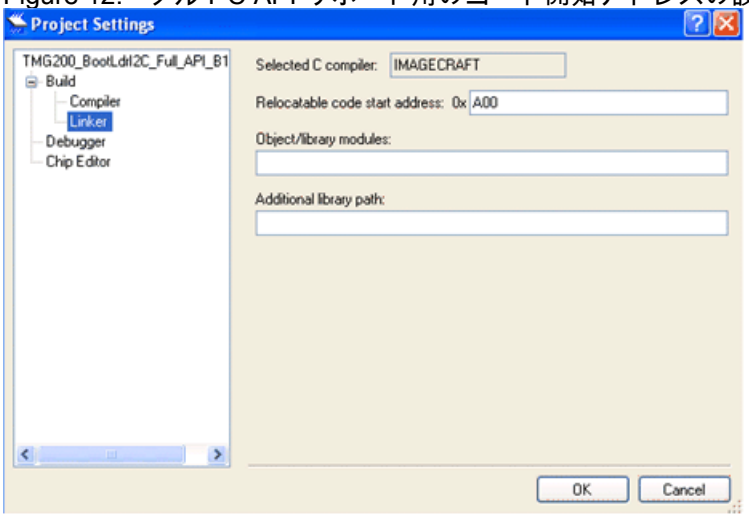


Figure 12. フル I²C API サポート用のコード開始アドレスの設定




```
//-------------------------------------------------------------------------
// C main line
//-------------------------------------------------------------------------
#include <m8c.h> // part specific constants and macros
#include "PSoCAPI.h" // PSoC API definitions for all User Modules
BYTE result;
WORD wAddr, wByteCount, cTemperature, wByteReadCount;
BYTE pbDataDest[10], pbData[10];
void main()
{
    //example application consists of an EEPROM UM, an LED UM,
    //and a 16-bit timer UM.
    //the EEPROM demonstrates that the EEPROM UM can co-exist
    //with the bootloader, the timer sets a duty cycle and the
    //LED blinks at the set duty cycle.
    //The bootloader UM provides the capability to modify
    //the project (LED duty cycles are conveniently visible.)
    //and use the bootloader to download the modified project.
```

```
    //by the time the main() function is executed the project
    //has already been checksummed and verified by the bootloader.
    //init EEPROM data
    wAddr = 0;
    wByteCount = 64;
    wByteReadCount = 10;
    cTemperature = 25;

    //start the bootloader running in the background
    BootLdrI2C_1_Start();
    BootLdrI2C_1_EnableSlave();
    BootLdrI2C_1_EnableInt();

    //start blinking the LED
    Counter24_1_Start();
    Counter24_1_EnableInt();
    LED_1_Start();
    M8C_EnableGInt;

#define INCLUDE_LIB_API
#ifdef INCLUDE_LIB_API
    E2PROM_1_Start();
    result = E2PROM_1_bE2Write( wAddr, pbData, wByteCount, cTemperature);
    E2PROM_1_E2Read( wAddr, pbDataDest, wByteReadCount);
    // Insert your main routine code here.
#endif

#define BUSMODE 0xf5
    while(1)
    {
        asm("nop");
    }
}
```

アセンブリ言語で書かれた I²C ブートローダ ユーザ モジュールの実装を以下に示します。

```
;-------------------------------------------------------------------
; 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
_main:
    ; Insert your main assembly code here.
    lcall BootLdrI2C_1_Start
    lcall BootLdrI2C_1_EnableSlave
    lcall BootLdrI2C_1_EnableInt

    //start blinking the LED
    lcall Counter24_1_Start
    lcall Counter24_1_EnableInt
    lcall LED_1_Start
    M8C_EnableGInt
.terminate:
    jmp .terminate
```

コンフィグレーション レジスタ

このセクションでは、I²C ブートローダ ユーザ モジュールが使用または変更する PSoC リソース レジスタについて説明します。

Table 1. リソース I2C_CFG：バンク 0 reg[D6] 設定レジスタ

ビット	7	6	5	4	3	2	1	0
値	予約	PinSelect	Bus Error IE (バス エラー IE)	Stop IE (ストップ IE)	Clock (クロック) Rate[1]	Clock (クロック) Rate[0]	Enable Master (マスタのイネーブル化)	スレーブの有効化

ピン選択 : P1[5]/P1[7] または P1[0]/P1[1] として、SCL または SDA のいずれかを選択します。

バス エラー割り込みイネーブル : バス エラー時の I²C 割り込み生成をイネーブルにします。

停止エラー割り込みのイネーブル I²C 停止条件での I²C 割り込みをイネーブルにします。

クロック レート [1,0]: 3 つの有効なクロック レート、50、100、および 400 kbps （ CPU_Clk_speed が 6 MHz を上回る場合は 400 kbps ）から選択します。

マスタのイネーブル : I²C HW ブロックをバス マスタとしてイネーブルにします。

スレーブのイネーブル : I²C HW ブロックをバス スレーブとしてイネーブルにします。

Table 2. リソース I2C_SCR：バンク 0、reg[D7] ステータス制御レジスタ

ビット	7	6	5	4	3	2	1	0
値	Bus Error (バス エラー)	Lost Arb (Arb 消滅)	Stop Status (停止ステータス)	ACK out (ACK 出力)	Address (アドレス)	Transmit (送信)	Last Recd Bit (LRB) (最後の受信ビット : LRB)	Byte Complete (バイト完了)

バス エラー : バスエラー条件が検出されたことを示します。

アービトレーション消滅 : MultiMaster モードでは、このデバイスのアービトレーションが消滅した （ デバイスがバスを制御しない ）ことを示します

停止ステータス : I²C 停止条件が検出されています。

ACK 出力 : I²C ブロックを受信バイトの認識 (1) または非認識 (0) にダイレクトします。

アドレス : 受信または送信されるバイトがアドレスです。

最後に受信したビット (LRB): 送信シーケンスの最後に受信されたビット （ ビット 9 ）の値、送信先デバイスからの ACK 応答 / NAK 応答のステータスです。

バイト完了 : 8 データ ビットが受信されました。受信モードでは、バスが Ack/Nac を待機するためストールします。送信モードの場合、ACK 応答 / NAK 応答も受信されており （ LRB を参照)、バスは次の動作を待ってストールします。

Table 3. リソース I2C_DR：バンク 0、reg[D8] データ レジスタ

ビット	7	6	5	4	3	2	1	0
値	データ							

受信または送信したデータ。データ送信のためには、I2C_SCR レジスタ書き込みの前にこのレジスタをロードしなければなりません。受信データは、このレジスタから読み取られます。このレジスタには、アドレスまたはデータが含まれる場合があります。

Table 4. リソース I2C_MSCR：バンク 0 reg[D9] マスタ ステータス制御レジスタ

ビット	7	6	5	4	3	2	1	0
値	予約	予約	予約	予約	Bus Busy（バス ビジー）	Master Mode（マスタ モード）	Restart Gen（再開生成）	Start Gen（開始生成）

バス ビジー : マスタ専用、バス開始条件が検出されると設定され、停止条件が検出されるとクリアされます。

マスタ モード : デバイスが現在バス マスタとして動作していることを示します。

再開生成 : マスタ専用。I^2C バスの反復開始を生成するために設定できます。

開始生成 : マスタ専用。バスがアイドル状態になると、データ レジスタ （I2C_DR).）内のデータを使用して、I^2C バス開始を生成し、I^2C アドレスを転送します。

付録

次のセクションでは、I²C ブートローダの作成時に役立つ追加情報を説明します。

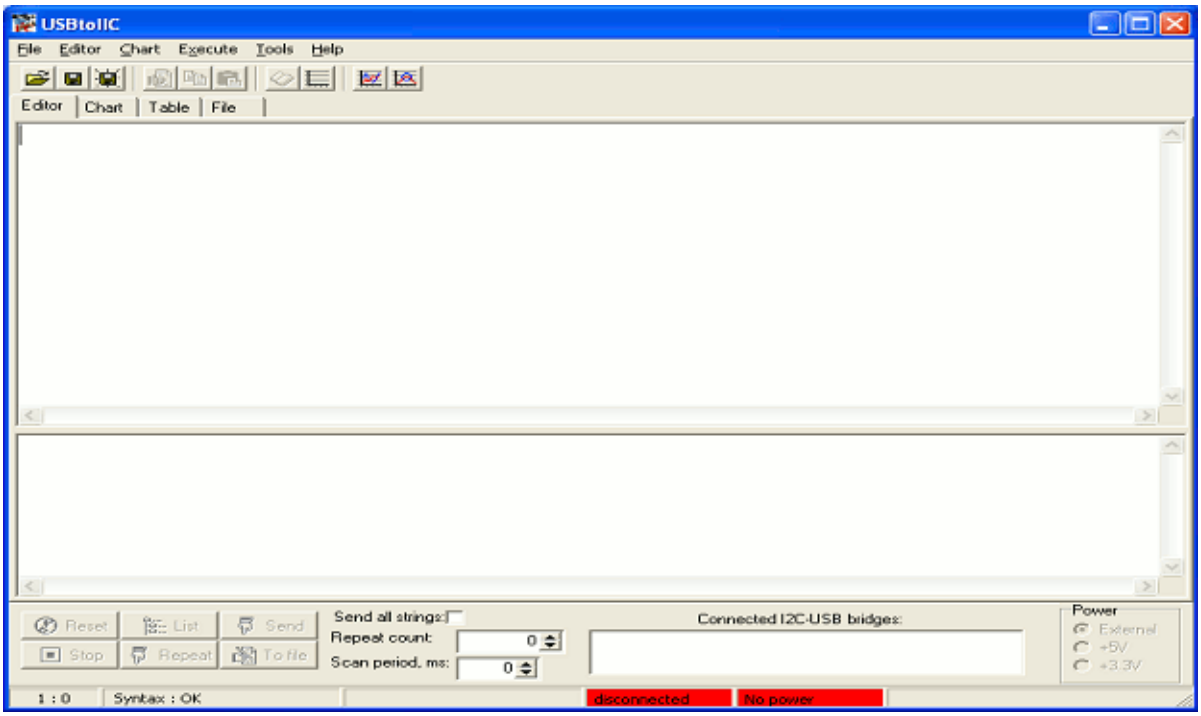
USBtoIIC ブリッジ および GUI アプリケーションの使用

USBtoIIC ブリッジと関連 GUI が、ブートローダのダウンロードに好まれる方法です。

詳細は、アプリケーション ノート 「CY3240 I²C-USB ブリッジを使用した I²C ブートローダ」をご覧ください。AN45683 で、<project_name>.txt ファイルのフォーマット、および プロジェクトのブートロードに使用する手順が説明されています。アプリケーション ノートには、.dld フォーマットを .txt フォーマットにするためのツールについて説明があります。これは、このデータシートにあるデバイスのためには必要ではありません。<project_name>.txt ファイルは自動的に生成されます。

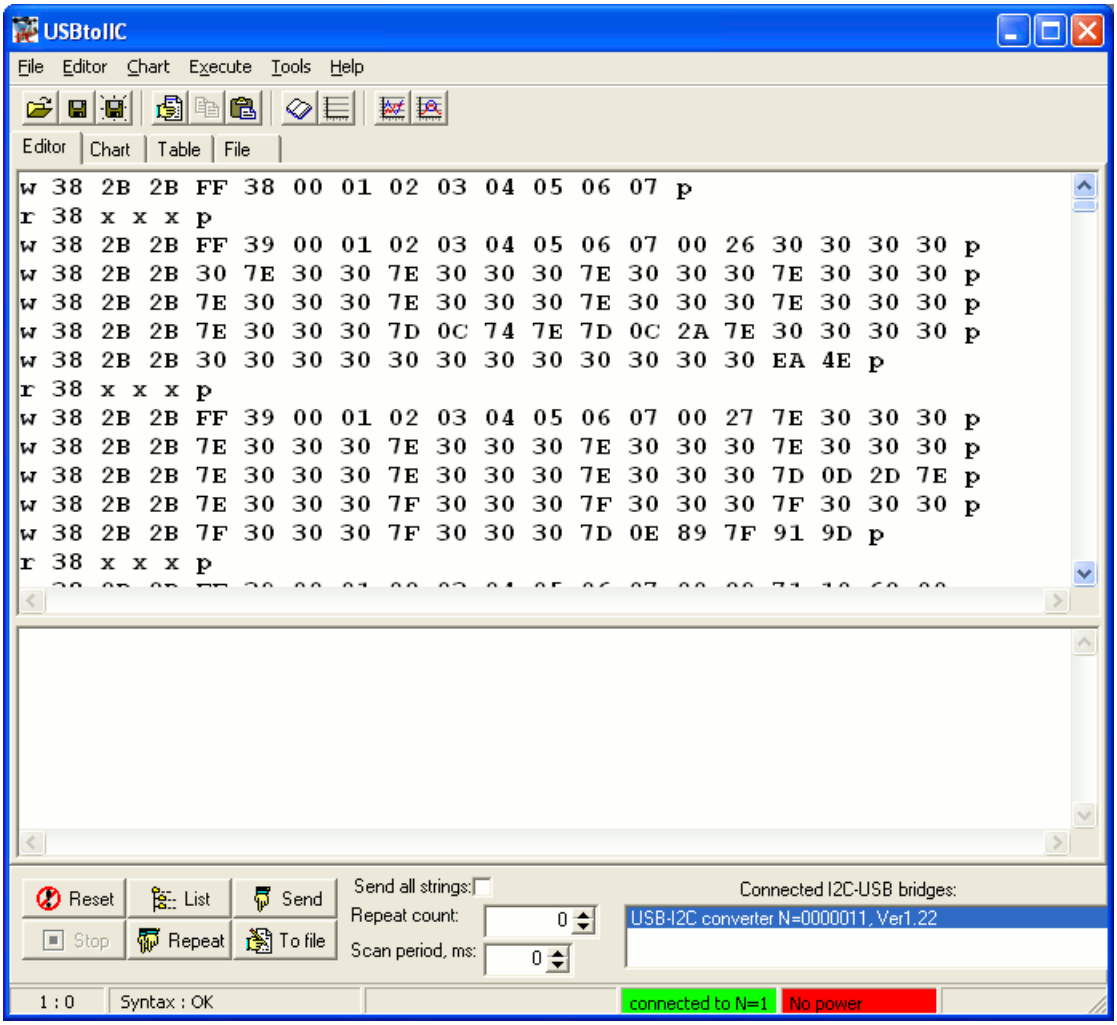
次に USBtoIIC ブリッジの使用を簡単に説明します。

CY3240 USBtoIIC ブリッジのアプリケーションプログラムを開始してください。GUI の操作を次の図に示します。



<projectname>.txt ファイルを、USBtoIIC ブリッジ GUI にインポートします。

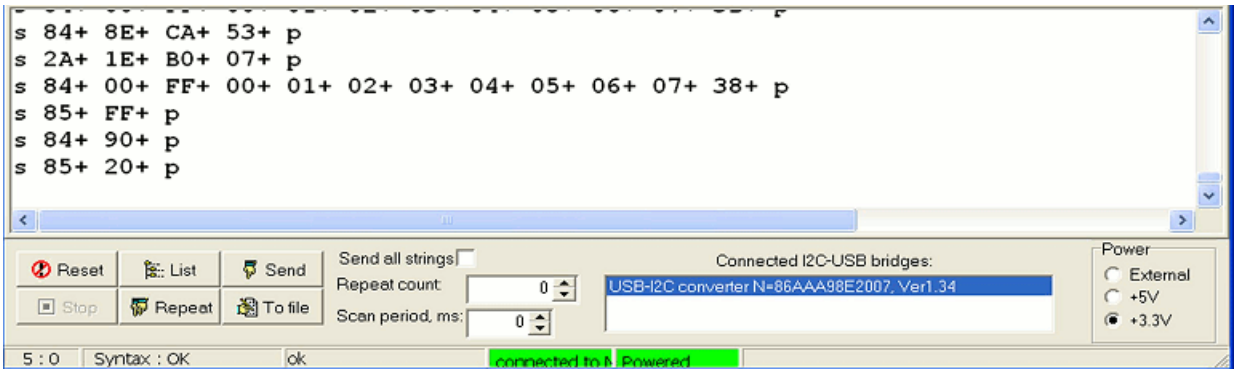
File（ファイル）> > Open（開く）を選択し、ブートロードするプロジェクトの出力ディレクトリを開きます。<projectname>.txt というファイルを探します。ファイルブラウザウィンドウで、ファイルの種類を、「すべてのファイル」にする必要がある場合もあります。このファイルが存在しない場合は、本書で説明しているブートローダ ツールを使用して再生成する必要があります。開くファイルを選択してから、読み込むのに数秒かかることもあります。ファイルの読み込みが完了したかどうか確認するために、下方のウィンドウを右クリックします。メニューが表示されたら、GUI は準備完了です。



ターゲットのシステムに、CY3240 USBtoIIC ブリッジ を接続します。GUI を使用して、任意のレベルに電源電圧を設定します。これはターゲットのシステムが必要とする電力に応じて、電源電圧を供給するために使うことができます。GUI 一番下のステータスバーは、ブリッジが接続され、電力が供給されていることを示していなければなりません。すべてのダウンロードファイルを送信するために、USBtoIIC GUI の一番下にある 「Send all strings（全文字列を送信）」ボックスがチェックしてあるか確認します。 「Send all strings（全文字列を送信）」ボックスのチェックをはずすと、ダウンロードファイルの小片が強調表示され、テストのために送信されることがあります。



送信ボタンをクリックして新しいコードをダウンロードすると、ステータス領域に様々な I2C トランザクションのステータスが表示されます。「+」は トランザクションの完了を表します。



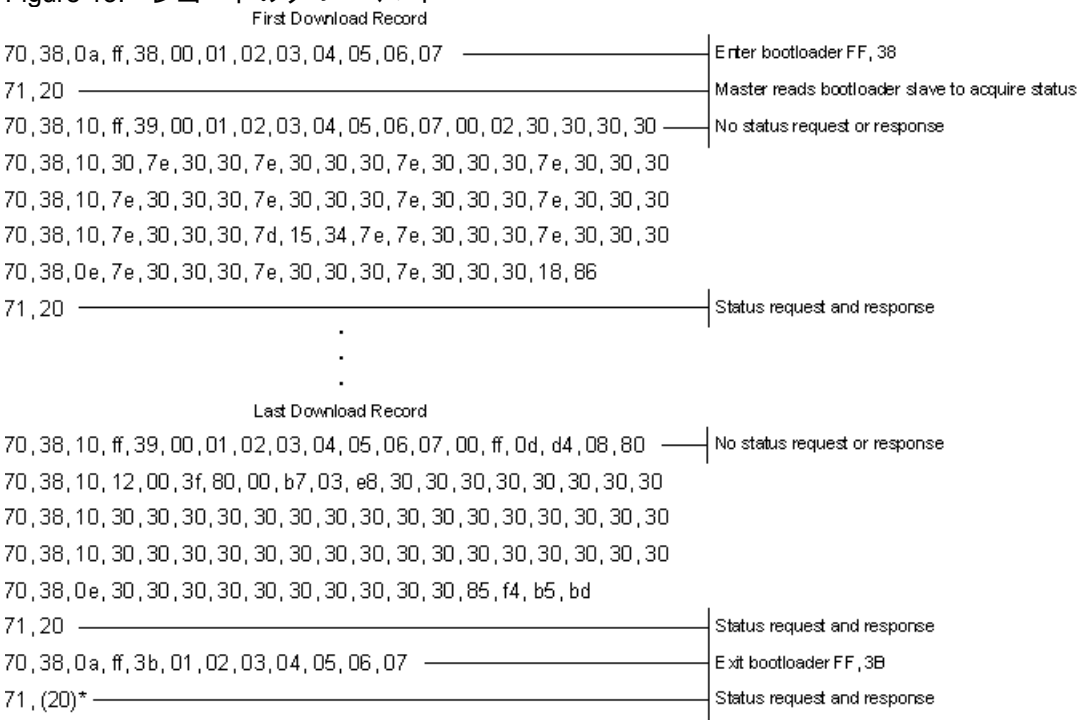
ブートローダ I²C ダウンロードプロトコル

アプリケーション ノート 「CY3240 I²*C-USB ブリッジ*を使用した *I²C ブートローダ」および CY3240 では*、>.txt ファイルのフォーマット、および プロジェクトのブートロードに使用する手順が説明されています。アプリケーション ノートには、.dld フォーマットを .txt フォーマットにするためのツールについて説明があります。これは、このデータシートにあるデバイスのためには必要ではありません。<project_name>.txt ファイルは自動的に生成されます。

次にファイル <project_name>.dld のフォーマットについて説明します。注； 64-byte と 128-byte のフラッシュブロックサイズを持つデバイス用のファイルフォーマットは変わりません。プログラムされるブロックのサイズはブートローダの内部でサポートされます。

ここに、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、ブートロードされる I²C マスタとスレーブ間を転送される実際のデータで構成されています。レコードのフォーマットを次の図に示します。

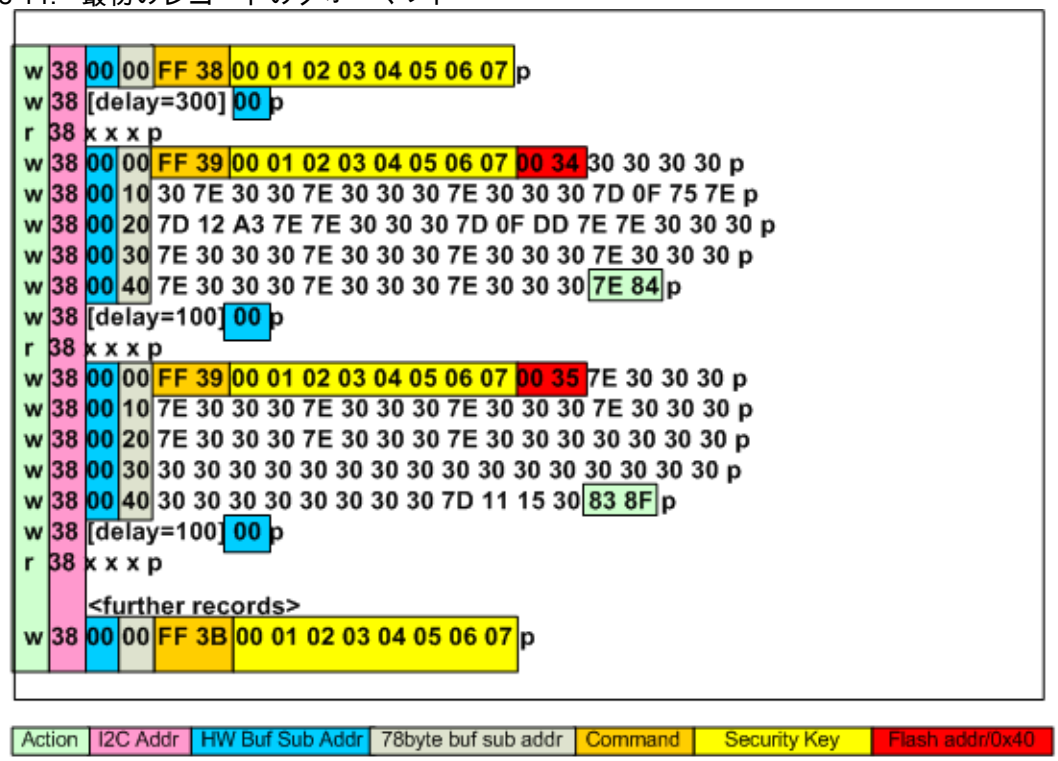
Figure 13. レコードのフォーマット



[+] Feedback

次の図は、最初のレコートのフォーマットを示しています。

Figure 14. 最初のレコードのフォーマット



address	Name	Bit 7	Bit 6		Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Master/ PSoC Access
BL,00	HST_MODE	Always 0 (defined later by host)								W/R: 00
BL,01	BootLoaderStatus	0	0	0	BootLdr Mode	1, 1 during ERASE APP		Watchdog reset	Checksum valid	R/W:00
BL,02	BootErrorCode	Inalid Command	Invalid decurity Key	BootLoad Mode	Comm Cksum Error	Flash Protection Error	Flash Checksum Error	Image Verify Error	Boot Completed OK	R/W:00
BL,03	BL_VER_Bhi	Bootloader Version Defined by Bootloader (lmsb) (default 0x10)								R/W:00
BL,04	BL_VER_Blo	Bootloader Version Defined by Bootloader (lsb) (default 0x00)								R/W:00
BL,05	BL_VER_Ah	Bootloader Version Defined by Application (msb) (default 0xff)								R/W:00
BL,06	BL_VER_Alo	Bootloader Version Defined by Application (lsb) (default 0xff)								R/W:00
BL,07	DIP_VERhi	Design IP Version (msb) (default 0xff)								R/W:00
BL,08	DIP_VERlo	Design IP Version (lsb) (default 0xff)								R/W:00
BL,09	APP_IDhi	Application ID (msb) (default 0xff)								R/W:00
BL,0a	APP_IDlo	Application ID (lsb) (default 0xff)								R/W:00
BL,0b	APP_VERhi	Application Version (msb) (default 0xff)								R/W:00
BL,0c	APP_VERlo	Application Version (lsb) (default 0xff)								R/W:00
BL,0d	CID_0	Custom ID #0 (default 0xC1)								R/W:00
BL,0e	CID_1	Custom ID #1 (default 0xC2)								R/W:00
BL,0f	CID_2	Custom ID #2 (default 0xC3)								R/W:00

各行の先頭は制御バイトです。ダウンロード プロトコルが使用する 2 つの制御バイトを以下に示します。

- 70 - スレーブ アドレス 38 への書き込み。アドレスは、バイト カウントの一部とはみなされません。
- 71 - スレーブ アドレス 38 からの読み取り。スレーブ アドレス読み取りに対して予想される応答は、0x20 で、正常終了です。その他の応答例 :

Table 5. スレーブ アドレス読み取りの応答

コード	意味
0x20	ブートロードモード (正常終了)
0x02	イメージ確認エラー
0x04	フラッシュ チェックサム エラー
0x08	フラッシュ保護エラー
0x10	共通チェックサム 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

スレーブ アドレス書き込みコマンドは応答を必要としないので、各スレーブ アドレス書き込み行の次の 2 バイトは、ブートローダが無視する I^2C プリフィックスです。アプリケーションで使用するプリフィックスのバイト数を設定するには、Ignore_N_I2C_Prefix_Bytes パラメータを使用します。

サンプル ダウンロード レコードの 1 行目と 3 行目には、ブートローダ コマンドが含まれます。次のブートローダコマンドが使用されます。

Table 6. ブートローダ コマンド

コマンド	意味
FF38	ブートローダに入る
FF39	ブロックを書き込む
FF3B	ブートローダを終了する

すべてのブートローダ コマンドは、ブートローダ キーと共に送信しなければなりません。ブートローダは、適切なキーと共に送信されていないコマンドを無視します。ブートローダ キーは、Bootloader_Key パラメータで設定します。

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コマンドの大部分は、書き込みブロック コマンドです。各書き込みブロック コマンドのフォーマットは同一です。ブロックの 1 つ、すなわちこの場合最後のブロックには、チェックサム情報が含まれます。最後の（チェックサム）ブロックのフォーマットについては、このセクションに記載しています。他の各書き込みブロック コマンドは、64-byte 16 進数レコードを、合計 78 バイト（アドレスおよび破棄されるプリフィックスを無視）になる 5 つのパケットでブートローダに送信します。128 バイトブロックを持つデバイスでは、これらの 64byte 転送のうち 2 つを収集します。ブロック組織が、最初に偶数ブロックで、次に奇数ブロックで構成されていることが検証されます。ブロックが完全に収集されると、フラッシュが書き込まれます。

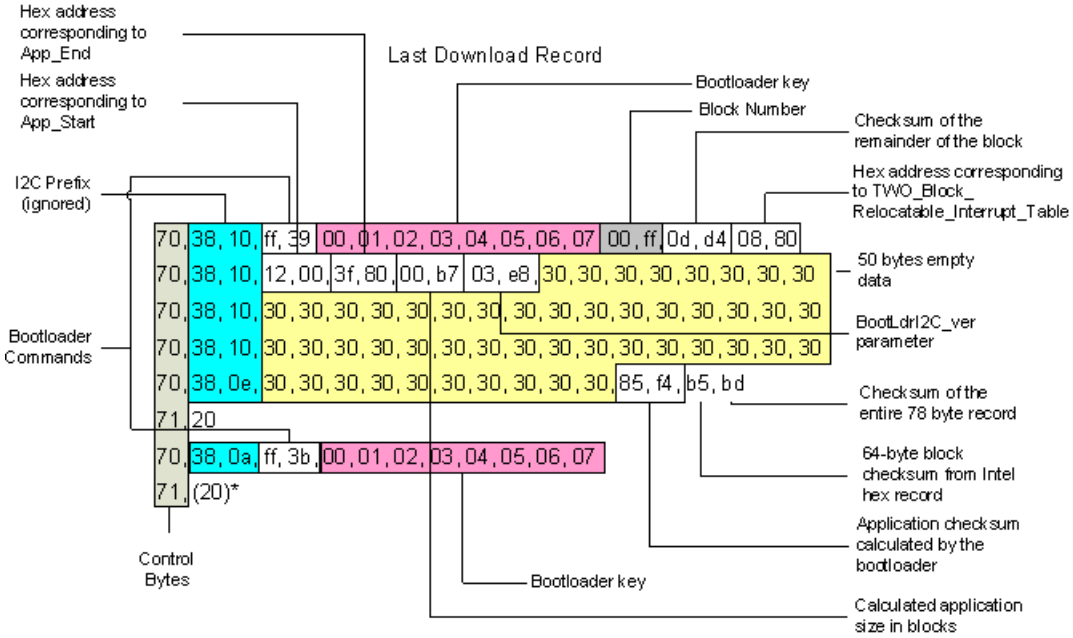
書き込みブロック コマンドの最初の行には、制御バイト、無視される 2-byte I^{2}C プリフィックス、書き込みブロック コマンド、ブートローダ キー、送信されるブロック番号、および最初の 4 バイトのデータが含まれます。この例のブロック番号は 0x0002 で、これは ROM アドレス 0x0080 に対応します。

次の 3 行には、制御バイト、I^{2}C プリフィックス、および 16 バイトのデータだけが含まれます。書き込みブロック コマンドの最後の行には、制御バイト、I^{2}C プリフィックス、データの最後の 12 バイト、および 1-byte チェックサムが 2 つ含まれています。最初のチェックサム、すなわちこの例では 0x18 は、このレコードのデータ バイトのチェックサムです。2 番目のチェックサム、すなわちこの例では 0x86 は、アドレス バイトとプリフィックスを除く 78 バイトのレコード全体のチェックサムです。アドレス バイトとプリフィックスは、受信時にブートローダによって内部で確認されます。

書き込みブロック コマンドの終了時には別のステータス要求が送信され、応答が表示されることになります。

最後に送信されるブロックのフォーマットも同じですが、この例では、チェックサム情報を含んでいることを確認できます。次の図は、最後のレコードのフォーマットを示しています。

Figure 15. 最後のレコードのフォーマット



最後のレコードには、この例のチェックサム ブロックが含まれています（なお、このレコードのブロック番号は 0x00ff です）。ここで、レコード内のデータについて説明します。この説明は、チェックサム ブロックを含む 66 バイトのレコードに当てはまります。

最初の行には、他のレコードとまったく同じように、制御バイト、I^{2}C プリフィックス、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、およびブロック番号が含まれています。次の 2 バイト

[+] Feedback

には、ブロック残り部分のオプションのチェックサム、すなわちこの場合は 0x0 DD4、が含まれています。この行の最後の 2 バイトには、TWO_Block_Relocatable_Interrupt_Table パラメータ向けのブロック 0x22 から計算された 16 進数アドレスが含まれています。

レコードの 2 行目には、制御バイトと I^2C プリフィックスに続いて、ブロック 0x48 から計算される App_Start ユーザ モジュール パラメータの 16 進数アドレスを表す 2 バイトの値が含まれています。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 パラメータの 16 進数アドレスです。それから、ブロック単位のアプリケーション サイズである 2 バイトが続きます。この行最後の 2 バイトの実際のデータ値は、BootLdrI2C_ver パラメータからのブートローダ バージョン番号です。この行の残りは、空のデータ スペースです。

次の 2 行には、さらに空のデータ スペースが含まれています。チェックサム ブロックの最後の行には、行の最後の 4 バイトまで空のデータ スペースが含まれています。最後の 4 バイトには、ブートローダによって計算される 2 バイトのアプリケーション チェックサム、Intel 16 進数レコードからの 1 バイトのブロック チェックサム、および最後に、アドレス バイトとプリフィックスを除いた 64 バイトのレコード全体のチェックサムが含まれています。アドレス バイトとプリフィックスは、受信時にブートローダによって内部で確認されます。

次の行には、別のステータス要求および応答が含まれます。

ブートローダ終了コマンドは、制御バイト 70、2 バイトのプリフィックス、ブートローダ終了コマンド 0xFF3B、およびブートローダ キーで構成されています。

最後の行は最終ステータス要求で、応答の場合もあります。ブートローダ終了コマンドを受け取ると、ターゲット システムは内部リセットを即座に実行し、チェックサムブロックのパラメータを使用して、ダウンロードしたアプリケーションの チェックサム確認を開始します。ホスト システムの速度によって違いますが、ブートローダは、マスタが有効なステータス バイトを受信できる前に、すでにリセット プロセスを開始している場合もあります。したがって、必ずしもステータス（0x20）が存在するとはいえません。その代わりに、アドレス 0x71 は単に NAK 応答される場合もあります。

バージョン ヒストリー

バージョン	著者	説明
2.00	DHA	1. メモリ マップを再構成。 2. 再配置可能割り込みベクターテーブルをアドレス 0x0080 に配置。 3. チェックサムブロックをアドレス 0x0100 に配置。 4. ブートローダ開始をアドレス 0x0180 に配置。 5. 確認済みアプリケーションブロックサイズは 256 より小さい。 6. ブートローダ API ジャンプ テーブルを追加。 7. ユーザ モジュール パラメータ テーブルを更新。

Note PSoC Designer 5.1 は、すべてのユーザ モジュール データシートにおいてバージョン ヒストリーを導入しています。このセクションでは、ユーザ モジュールの過去のバージョンと現在のバージョンとの違いに関して高度な解説を掲載しています。

Copyright © 2008-2011 © Cypress Semiconductor Corporation. 本文書に記載される情報は、予告なく変更される場合があります。Cypress Semiconductor Corporation (サイプレス セミコンダクタ社) は、サイプレス 製品に組み込まれた回路以外のいかなる回路を使用することに対しても一切の責任を負いません。特許またはその他の権限下で、ライセンスを譲渡または暗示することもありません。サイプレス 製品は、サイプレスとの書面による合意に基づくものでない限り、医療、生命維持、救命、重要な管理、または安全の用途のために使用することを保証するものではなく、また使用することを意図したものでもありません。さらにサイプレスは、誤動作や故障によって使用者に重大な傷害をもたらすことが合理的に予想される、生命維持システムの重要なコンポーネントとしてサイプレス製品を使用することを許可していません。生命維持システムの用途にサイプレス 製品を供することは、製造者がそのような使用におけるあらゆるリスクを負うことを意味し、その結果サイプレスはあらゆる責任を免除されることを意味します。

PSoC® は Cypress Semiconductor Corp の登録商標であり、PSoC Creator ™ および Programmable System-on-Chip ™ は Cypress Semiconductor Corp. の商標です。本文書で言及するその他すべての商標または登録商標は、各社の所有物です。

全てのソース コード (ソフトウェアおよび / またはファームウェア) はサイプレス セミコンダクタ社 (以下「サイプレス」) が所有し、全世界の特許権保護 (米国およびその他の国)、米国の著作権法ならびに国際協定の条項により保護され、かつそれらに従います。サイプレスが本書面によりライセンシーに付与するライセンスは、個人的、非独占的かつ譲渡不能のライセンスであって、適用される契約で指定されたサイプレスの集積回路と併用されるライセンシーの製品のみをサポートするカスタム ソフトウェアおよび / またはカスタムファームウェアを作成する目的に限って、サイプレスのソース コードの派生著作物をコピー、使用、変更そして作成するためのライセンス、ならびにサイプレスのソース コードおよび派生著作物をコンパイルするためのライセンスです。上記で指定された場合を除き、サイプレスの書面による明示的な許可なくして本ソース コードを複製、変更、変換、コンパイル、または表示することは全て禁止されます。

免責条項 : サイプレス は、明示的または黙示的を問わず、本資料に関するいかなる種類の保証も行いません。これには、商品性または特定目的への適合性の黙示的な保証か゛含まれますが、これに限定されません。サイプレスは、本文書に記載される資料に対して今後予告なく変更を加える権利を留保します。サイプレスは、本文書に記載されるいかなる製品または回路を適用または使用したことによって生ずるいかなる責任も負いません。サイプレスは、誤動作や故障によって使用者に重大な傷害をもたらすことが合理的に予想される生命維持システムの重要なコンポーネントとしてサイプレス製品を使用することを許可していません。生命維持システムの用途にサイプレス製品を供することは、製造者がそのような使用におけるあらゆるリスクを負うことを意味し、その結果サイプレスはあらゆる責任を免除されることを意味します。

ソフトウェアの使用は、適用されるサイプレス ソフトウェア ライセンス契約によって制限され、かつ制約される場合があります。