I²C ブートローダデータシート BootLdrI2C V 2.20

Copyright © 2007-2012 Cypress Semiconductor Corporation. All Rights Reserved.

リソース	PSoC® ブロック			API メモリ（バイト）		ピン（外部入出力ごと）
	デジタル	アナログ CT	アナログ SC	フラッシュ	RAM	
CY7C603xx, CY7C64215, CY8C21x12, CY8C21x45, CY8C22x45, CY8C23x33, CY8C24x9x, CY8C28x43, CY8C28x52, CY8C29/27/24/21x3x, CY8CPLC20, CY8CLED04/08/16, CY8CLED0xD, CY8CLED0xG, CY8CLED16P01, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120						
スレーブ（フル API サポート）	0	0	0	2560	6-128	2
スレーブ (API サポートなし)	0	0	0	2144	6-128	2

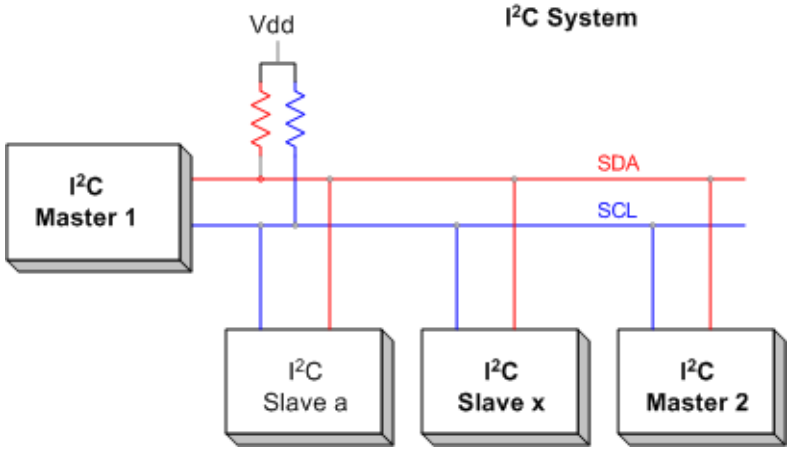
機能と概要

- 業界標準の Philips I²C バス互換インターフェース
- 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 を再プログラムすることが可能。

BootLdrI2C 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 モジュールは使用現場で再プログラムを必要とする機器にお使いいただけます。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が 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、その機能に別の I²C アドレスを使用します。I²C ブートローダのすべてのコードは、EEPROM の保護領域にプログラムされ、誤って上書きされることはありません。

Figure 1. I²C ブロック図



クイック スタート

1. このユーザモジュール データシートをよく読んでください。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. ユーザ モジュールをプロジェクトに追加します。
3. [I²C for Bootloader Only (ブートローダ専用の I²C)] または [Full I²C API Support with Bootloader (ブートローダを含むフル I²C API サポート)] のいずれかを選択して、ユーザ モジュールを配置します。
4. 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。
5. 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。.
6. Get Files (ファイルを取得) をクリックします。Get Files (ファイルを取得) を . boot.tpl, custom.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。*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。
7. **BootLoader Tools** ウィザードを閉じます。
8. 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。
9. 出力ファイル <project>.mp と <project>.hex を点検し、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。*<project>.mp* および *<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。
10. エラーなく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。サンプルとして提供されているコードを変更し、適用してください。
11. 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いただけます。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 パラメータによって）定義したフラッシュ メモリのセクションに配置されます。このメモリ空間は、誤って変更したり破損することのないよう、書込み保護されています（されていなければなりません）。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を確認します。チェックサムが一致した場合、前回のプログラムが成功したと判断し、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（例えばファンをオンにするなど）を実行する「カスタマイズ可能なユーザコード」を実行します。そしてマスタから 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、（例えば電源異常の発生など）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 64-byte のパケットとして更新されたユーザ 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。

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。図 2 はブートローダのメモリ構成を示します。

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。

I2C とスリープ

I2C をスリープ状態の設定があるプロジェクトと共に使用する場合は、特別な注意が必要です。プロジェクトがスリープ状態に入る前に、適切にスリープに入り I2C を処理するために、次の手順に従ってください。

1. すべての I2C 通信が完了していることを確認します。
2. Stop API を呼び出し、I2C を無効にします。
3. I2C ピンをアナログハイ Z ドライブモードに設定します。

スリープ状態から復帰したら、以下の手順に従います。

1. アクティブな I2C 通信が存在しないことを確認します。
2. Start API を呼び出し、I2C を有効にします。

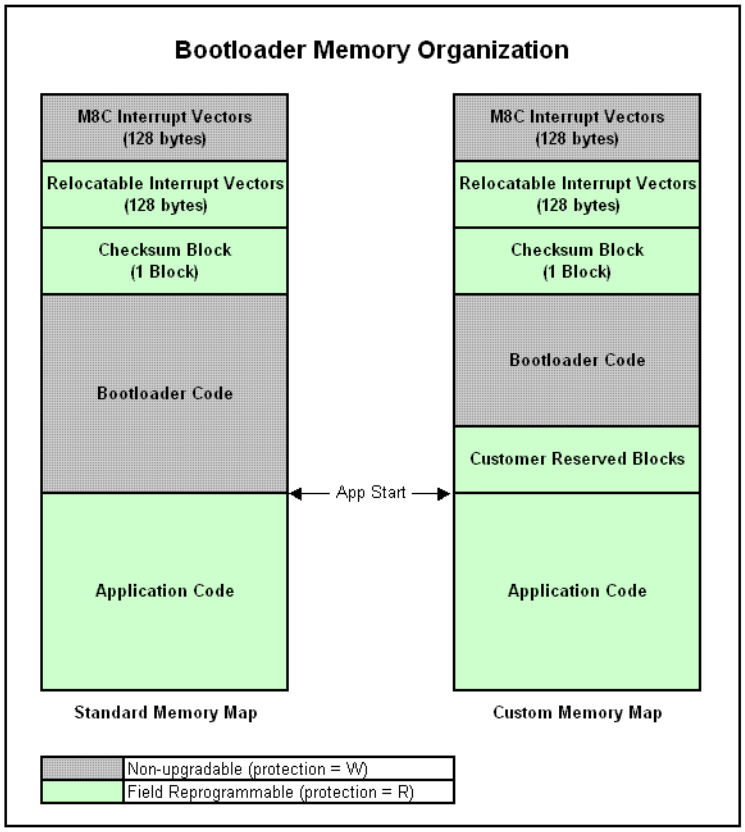
3. I2C ピンを Open-Drain Drives Low ドライブモードに設定します。
4. 割り込みを許可します。

動作理論

ブートローダ付きプロジェクトを作成するには、通常の PSoC Designer モデルに、標準的ではない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、BootLdrI2C 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。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 サンプルも提供されています。これによりブートローダの基本機能を確認できます。この PC ベースのアプリケーションおよび Microsoft Visual Studio® 2005 向けのソース コードは、PSoC Programmer 3 のインストール ディレクトリ内にある .zip ファイルに含まれています。

<install_path>\Cypress\Programmer\3.00\Bootloaders\BootLdrI2C\BootLoaderHostApp\…

Figure 2.　ブートローダメモリ配置



概要

PSoC Designer は、標準化ファイル、デバイスファミリに関する内蔵データおよびデバイス固有の属性を使い、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、フォアグランド アプリケーションをダウンロードして内容を確認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 データがなければ、Standard Memory Map（標準メモリマップ）が使用できます。
- 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、BootLdrI2C 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 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 モジュールの初期設定は、動作する設定になっています。プロジェクトが完全にコンパイルされるまで、この設定を使います。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

BootLdrI2C 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ユーティリティを提供します。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、フォアグランド アプリケーションの内容を確認します。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。二つのチェックサムが等しい場合、ブートローダユーティリティは、フォアグランド 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まで待機します。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 で説明したように、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。
CY3240 USB-I^2C ブリッジを使用する方法の詳細については、アプリケーションノート AN2352 “PSoC®1 Communication - I2C-USB Bridge Usage” を参照してください。この方法は、このユーザ モジュール データシートの付属資料でも説明します。

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\3.00\Bootloaders\BootLdrI2C\BootLoaderHostApp\...

I^2C ブートローダの実験のために、二つのアプリケーションが提供されています。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 プロジェクトにはソースコードが提供され、他の PSoC デバイスアーキテクチャに簡単に適用できます。

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変更のためのソースコードと共に提供されます。ダウンロードファイル <filename>.dld の読み取りと解析、さらにこのセクションで前述した 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実証目的のためのみに使用され、生産のために使用したり再販は意図していません。

Note 場合によっては、ユーザのコンピュータに mscomm32.ocx ファイルをインストールする必要があります。*mscomm32.ocx* インストールする必要があることもあります。このファイルはマイクロソフトのウェブサイトからダウンロードし、Windows/accessories（アクセサリ）/command prompt（コマンド プロンプト）ウィンドウを使ってインストールしてください。

```
> regsvr32 mscomm32.ocx
```

ファイル mscomm32.ocx は *regsvr32.exe* Windows のインストール フォルダ windows/system32 にあります。(winXP の場合)

ファイル mscomm32.ocx はマイクロソフトのサイトでファイル名 "mscomm32.ocx" で検索できます。

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。ドロップダウンメニューから、BootLoader Tools（ブートローダツール）を選択します。

「Get Files（ファイルを取得)」の選択肢では boot.tpl の特殊バージョンを追加し、custom.lkp および HTLinkOpts.lkp はプロジェクトに配置することや削除することができます。メイン メニューから、これらのファイルを削除するために Tools Restore Default Boot files（デフォルト ブート ファイルの復元ツール）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にアクセスすると、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。次に、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から自動的に呼び出されるアプリケーションコードの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。アプリケーションコードの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。ビルド エラーをデバッグするには、ブートローダツール メニュー外の従来のビルドおよびアプリケーションコード生成プロセスを使用する必要があります。

Generate dld file（dld ファイルの生成）- このツール項目は、hex プロジェクト出力ファイルからダウンロード ファイルを抽出します。このファイルには、チェックサム ブロックを含むブートローダによって再プログラムされる 16 進数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展開して、PSoC デバイスをアップグレードできます。

dld と txt 形式のダウンロードファイルが BootldrI2C ユーザモジュールツールにより生成され、Workspace Explorer の Output Files (出力ファイル) に表示されます。

チェックサムの半自動生成

エラーなくプロジェクトをビルドおよびコンパイルできたら、アプリケーション チェックサムを生成する必要があります。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 BootldrI2C 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（以前計算済みまたはデフォルト）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ます。ブートローダツールメニューページが表示されると、以前のチェックサムが現在の output\<prj_name>.hex ファイルで計算されたチェックサムに対して検証されます。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。

特殊ファイル

いくつかの重要なファイルを取得するには、「**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）」を選択します。デバイス固有の boot.tpl ファイルは、custom.lkp（ImageCraft）ファイル、HTLinkOpts.lkp（Hi Tech）、および、事前に定義されている 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。各ファイルの目的は以下で簡潔に説明しています。

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: -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割り当てをデバッグするために、上記の 3 行すべてをコメント化することも可能です。各行頭にセミコロンを挿入します。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... 行には、全体の ROM サイズのデータが含まれています。
- -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 パラメータによって指定されているデフォルトのメモリ マップに従ってレイアウトされる、デフォルト ファイルです。最終的なプロジェクトの作成には、展開したデバイスとファームウェアの最終的なメモリ マップとアプリケーション サイズに従い、手作業でこのファイルを変更しなければならないこともあります。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。何らかの理由でプロジェクトを更新またはデータファイル以外にタグ付けされた場合は、Flashsecurity ファイルを上書きすることはしないでください。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。

I^2C 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 I^2C 割り込み処理モジュールのフォアグランド コピーを提供します。このコードは、I^2C 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 I^2C 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

すべてのメモリ パラメータは、16K デバイスの場合は 0x00 ~ 0xFF 、8K デバイスの場合は 0x00 ~ 0x7F、32K デバイスの場合は 0x00 ~ 0x1FF という番号のブロック内のブートローダに入力されます。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。問題となる PSoC デバイスは、64 バイトのフラッシュ ブロック（いくつかのデバイス ファミリでは 128 バイト）を格納することしかできないので、これはプロジェクト コードの異なる領域間の境界を正しく維持する簡単な方法です。

PSoC 製品はほとんど、ブロックサイズが 64 バイトです。新しい PSoC デバイスの中には、128 バイト ブロックのものもあります。重要な 2 つのことは、

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。

ブロックから絶対的なアドレスに変換するには、以下のように乗算します。Abs_addr = block_number X ブロック サイズ。Block_0 は addr 0 で開始、Block_n はアドレス n x Block_size. で開始。全ブロックはブートローダパラメータのために 16 進数で区切られているので、16 進アドレスは 0x40（64-byte ブロック）または 0x80（128-byte ブロック）で乗算して得られます。

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。したがって、64 バイトのブロックデバイスでは各行が 1 ブロックのコードを表します。128 バイトのブロックデバイスでは、16 進ファイルから 2 行が 1 ブロックに入ります (ブロック 0 はアドレス 0 から始まり、128 バイトのブロックは常に、" 偶数の " 半分がアドレスの下位半分を、" 奇数の " 半分が上位半分を表すとみなされねばならないからです)。

hex ファイルを見て、作業中製品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

ビルトイン ブートローダを搭載したアプリケーションは、デバッグしにくくなっています。このため、BootLdrI2C 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 BootLdrI2C_Bootloader.inc ファイルに含まれています。このファイルには以下のような式を含むセクションがあります。

```
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 定義を使用すると、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。

配置

I2CHW ユーザ モジュールには SCL および SDA P1[5]/P1[7] または P1[0]/P1[1] が選択でき、デジタルおよびアナログの PSoC ブロック も不要です。配置制限はありません。複数の I^2C モジュール配置は不可能で、それは I^2C モジュールが専用の PSoC リソースブロックと割り込みを使用するためです。

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。ユーザがプロジェクトをコンパイ

ルし、テストもした後は、ブロック サイズを調節してメモリの使用を最適化するという選択も可能です。

Figure 3.　デフォルト パラメータ

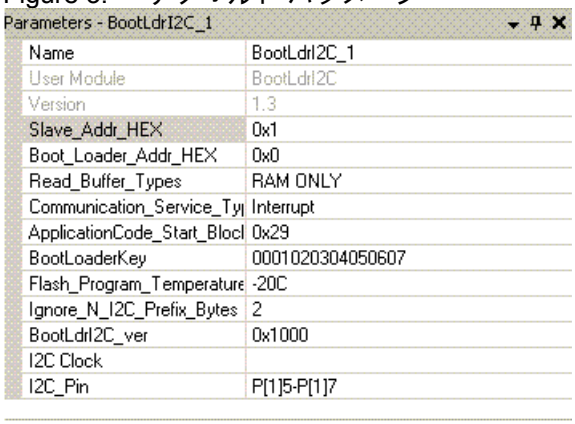


図 3 は、デフォルトのユーザ モジュール パラメータを示します。ユーザモジュールのソースコード内で、これらのパラメータは定期的に更新されることがあり、例とは違っている可能性もあります。

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

I2C マスタによって使用される 7-bit スレーブ アドレスを選択し、I2C ブートローダスレーブデバイスを指定します。有効な選択肢は 0-7Fh 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。パラメータ値は Slave_Addr_HEX パラメータ値と違うものでなければなりません。

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かどうかを選択する場合も、RAM OR FLASH を選択します。

Communication_Service_Type

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 I²C 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、I²C 状態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、I²C 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。

ポーリング関数を、割り込みから使用することは推奨されていません。ポーリング関数を呼び出すためにタイマー割り込みを定義すると、関数を何度も呼び出すことになり、他のデータ処理ができなくなります。ポーリング サービスは再入可能ではなく、関数は処理が完了するまで呼び出すことができません。

I2C Clock

I²C インタフェースを動作させるクロック速度を指定します。3 種類の I2C_Clock クロック速度を選択できます。

- 50K Standard
- 100K Standard
- 400K Fast （ CPU_Clk_speed が 6 MHz より速い場合 ）

I2C_Pin

ポート 1 から I²C 信号として使用するピンを選択します。PSoC Designer が自動的に適切なドライブ モードを選択します。SCL、SDA および割り込みピン以外、ポート 1 上のピンはすべて、ブートロード中は High Z Analog モードに設定されますのでご注意ください。

CPU_Clk_speed_(CY8C27xA)

Note このパラメータは、CY8C27xxx ファミリの製品にのみ存在します。

CY8C27xxx シリコンのリビジョン A デバイスが動作する CPU クロック速度の範囲を指定します。CY8C27xA は、新規設計に推奨されない古い CY27xxx デバイスのことです。PSoC Designer のデバイスカタログでは、CY8C27x シリコンのリビジョン A はカタログの一番下にあり、"Not Recommended for New Designs"（新規設計に非推奨）という警告が前書きされています。新しいバージョンの製品名には 「X」という文字が含まれているため、古いバージョンとの区別が可能になっています。文字 X は、鉛フリー製品を示すために使用されます。このパラメータは、高い CPU クロック速度では特殊なコード処理を挿入し、低い CPU クロック周波数ではそれが不要なオーバヘッドとならないよう取り除くために必要です。

CPU_Clk_speed_(CY8C27xA) の値	用途
6 MHz or less(6 MHz 以下)	CPU クロック速度が 6 MHz 以下の場合に使用します。これにより不要なコードが排除されます。
Above 6 MHz (6 MHz 超（CY8C27x A のみ）)	この設定は、CPU クロック速度が 6 MHz を超え、使用するチップが CY8C27xxx シリコンのリビジョン A ファミリの場合に使用します。これによりオーバヘッドがある程度追加される代わりに、ユーザ モジュールの正しい動作が保証されます。
Not CY8C27xA (CY8C27xA 以外)	この設定は CY8C27xxx シリコンのリビジョン A デバイス ファミリ以外のチップに使用します。これは、デバイスカタログの製品番号に 「X」の文字を含む CY8C29/24/22xxx 製品または CY8C27xxx 製品に適用されます。この値を選択すると、不要なコードが排除されます。

ブートローダに対して定義するパラメータにより、開発者は、プログラムをコンパイルおよびリンクするときに主なプログラム ブロックを配置する場所を定義することができます。場合によっては、対象製品がシリコンのより新しいバージョンであっても、プロジェクトが、シリコンのより古いリビジョン（したがって、このようなクロック速度固有の設定が必要となります）を使用する 27xxx POD で開発されることもありえます。新しいデバイスに、古いシリコンに適したクロック速度パラメータを含めても、プロジェクトに深刻な影響はありません。

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。

パラメータブロックによって指定されているデフォルト アドレスは、デバイスのブロック サイズ（ 0x40 or 0x80）にパラメータのブロックを乗算して計算できる場合があります。

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。この手順により、ブートローダアップグレード ユーティリティが誤って呼び出されないことを確認します。

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温度パラメータを実際の温度に合わせると、メモリ保持および最大書き込みサイクル数に影響を及ぼします。PSoC は低温で、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。

Ignore_N_I2C_Prefix_Bytes

Note RS-232 to I^2C 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、I^2C. に基づく、いくつかの 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、可変的なプリフィックスの 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。

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では使用されませんが、チェックサムブロックの一部として利用可能です。このパラメータはユーザが設定でき、ブートローダの実行可能コードの正しいバージョンを確認するために使用できます。

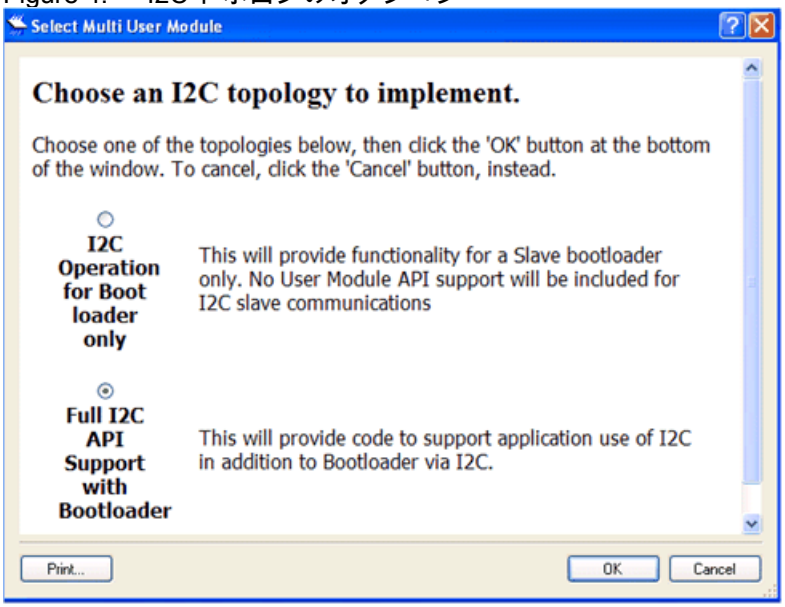
I^2C トポロジ選択オプション

BootLdrI2C 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 I^2C トポロジを実装するか決定しなければなりません。

I²C Operation for Bootloader only: このオプションは、ブートローダの I²C コミュニケーションのみを実行します。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。

Full I²C API Support with Bootloader: 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。

Figure 4. I2C トポロジのオプション



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ときは、PSoC の開発環境を変更しないでください。これには、PSoC Designer および BootLdrI2C ユーザ モジュールを更新しないこと、コンパイラを変更しないことが含まれます。

この背景には、はじめに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ていても、後にブートローダシステムが配置されると、アプリケーションセクションのみが再プログラムされることがあります。新規アプリケーションが最初の展開時のブートローダに一致するよう、新規の I2C、または修正したアプリケーションは、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しなければなりません。開発環境の要素の、全バージョンが互換性を持っているのが理想的です。しかしブートローダの場合は、その互換性を維持することが不可欠です。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。

PSoC Designer は複数のコンパイラをサポートしますが、だからといってある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 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。もうひとつの問題は、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ているために、使用した開発ツールで不一致があったブートローダとアプリケーションのペアをデバッグすることができないということです。

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、globalparams.inc. ファイルに含まれるグローバル パラメータ WATCHDOG_ENABLE にリンクされています。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。設定例は、ファイル「Flashsecurity.example」で提供されています。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。もう一つは、時間を取って、この時点でメモリ マップをレイアウトし、このファイルを適切に編集することです。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しなければなりません。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、システムの故障やデバッグ作業が非常に困難になる場合があります。

開発とデバッグのために、アプリケーション領域には 'U'（未保護）のフラッシュセキュリティを推奨します。最終生産では、外部からの読み取り· 書き込みを防ぐために、アプリケーション領域に 'R'（読み取り保護）のフラッシュセキュリティ設定を推奨します。

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、同じアドレスに複数のブロック コードを書き込もうとするときに、リンカが起こすエラーの一般的な原因です。このパラメータは、「Project（プロジェクト）」> 「Settings（設定）」>「Linker（リンカ）」タブにファイルされている再配置コード開始アドレスで変更できます。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。ブートローダ設定の I^2C バージョンの場合、通常はこの値に 0xA40 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）。

Note ブーストローダー I2C ユーザモジュールの配置を解除する場合、再配置可能なコード開始アドレスを元の値にリセットしません。手動で復元してください。

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（上記を参照）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 ブロックを配置します。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手かがりにな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。プログラムの実行は電源電圧上昇に比べて急速で、場合によってはフラッシュのプログラミング中に製品の電源電圧がまだ変動していることが考えられます。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、製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。

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。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、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と、リセット時にブートローダに入り、タイムアウト時間が過ぎると、フォアグランド アプリケーションに出るようにデバイスを設定できます。
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。ポート ピンを Low/High することでスイッチを開閉するようにできます。
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に、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（詳細はテクニカルリファレンスマニュアル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意味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および、再配置可能なコード開始アドレスを同一にし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）。この問題を解決するためのデバッグ方法には、ブートローダが使用する 16 進数コードの領域に特に注意を払いながら、問題になっている 2 つの hex ファイルを比較します。<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、特定のアプリケーションのアドレス パラメータを変更できるように、いくつかのリダイレクト ベクトルを使用できます。これらのすべてのジャンプ ベクトルは、アプリケーションとブートローダで一致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。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比べて急速で、場合によってはフラッシュのプログラミング中に製品の電源電圧がまだ変動していることが考えられます。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、製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 メモリ内にのみ設定できます。

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、効率性の観点から、この「registers are volatile（レジスタの揮発性）」ポリシーが採用されています。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。

大容量メモリデバイスでは、CUR_PP、IDX_PP、MVR_PP 及び MVW_PP レジスタの値を保護するのは呼び出し元の責任です。今すぐ修正されないレジスタもありますが、今後のリリースにこのケースが留まる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しない限り戻りません。

GenericBootloaderEntry の関数名 は常に同じ物理 ROM アドレスに配置されるので、後日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。この 3 つのルーチンはすべて、同じアドレス ベクトルを表しています。「GenericBootloaderEntry」ベクトルは、ユーザ モジュールのインスタンス名が変更されても変化しないエントリ ポイントがあるように、提供されています。

C プロトタイプ

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

過去に示された別名称の API 関数名でも呼び出せます。

アセンブラ：

```
lcall ENTER_BOOTLOADER
```

パラメータ：

なし

戻り値：

なし

注意事項：

関数の実行により A および X レジスタの内容は変更される可能性があり保証されません。大容量メモリ モデル (CY8C29xxx) の全ての RAM ページ ポインタ レジスタにも同じことが言えます。必要に応じ、fastcall16 関数を呼び出してこれらの値を保存してください。

BL_SetTemp

説明：

この関数は、測定されたダイ温度をブートローダに通知するために使用します。この場合、アプリケーションはダイ温度を測定し、この関数を使用してブートローダに渡します。次にブートローダイベントが生じるとブートローダは、この関数を使用して引き渡された温度に基づきフラッシュを最適にプログラムします。

ランタイム中は、定期的にデバイスのダイ温度を測定することを推奨します。ダイ温度を測定するたびに、この関数を用いてブートローダに引き渡すべきです。ブートローダはダイ温度を元にブート期間中のフラッシュプログラミングの消去と書き込み期間を最適に変化させます。これにより、フラッシュの保存期間と耐久性が最適化します。その結果、ブートロードの実行に要する時間は、ブートローダに渡された温度の値により異なります。

ダイ温度は、デバイスのオンチップ温度センサを使用するユーザモジュールを用いて測定することが可能です。または外部の温度センサーなどから温度を読み取ります。

この関数は 「Flash_Program_Temperature_Deg_C」ユーザモジュールパラメータにより設定されたダイ温度値を上書きします。

C プロトタイプ

```
void BL_SetTemp (CHAR cTemp);
```

アセンブラ：

```
mov A, cTemp
lcall BL_SetTemp
```

コード例：

```
void main(void)
{
CHAR cDieTemp = -20;  // Allocate a variable to hold the die temperature
// Use -20C as the default value
...
// Measure die temperature here and copy to cDieTemp variable
BL_SetTemp(cDieTemp); // Update Bootloader with real die temperature
ENTER_BOOTLOADER();  // Run the BootLoader
...
}
```

パラメータ：

cTemp: ダイ温度（摂氏）

戻り値：

なし

注意事項：

関数の実行により A および X レジスタの内容は変更される可能性があり保証されません。大容量メモリ モデル (CY8C29xxx) の全ての RAM ページ ポインタ レジスタにも同じことが言えます。必要に応じ、fastcall16 関数を呼び出してこれらの値を保存してください。

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

注意事項：

関数の実行により A および X レジスタの内容は変更される可能性があり保証されません。大容量メモリ モデル (CY8C29xxx) の全ての RAM ページ ポインタ レジスタにも同じことが言えます。必要に応じ、fastcall16 関数を呼び出してこれらの値を保存してください。

BootLdrI2C_DisableInt

説明：

SDA 割り込みをディセーブルにすることで、I^2C スレーブをディセーブルにします。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

注意事項：

関数の実行により A および X レジスタの内容は変更される可能性があり保証されません。大容量メモリ モデル (CY8C29xxx) の全ての RAM ページ ポインタ レジスタにも同じことが言えます。必要に応じ、fastcall16 関数を呼び出してこれらの値を保存してください。

BootLdrI2C_EnableInt

説明：

I^2C 割り込みを有効にして、開始状態の検出を可能にします。M8C_EnableGInt マクロを使って、グローバル割り込みをイネーブルにしてください。

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

注意事項：

関数の実行により A および X レジスタの内容は変更される可能性があり保証されません。大容量メモリ モデル (CY8C29xxx) の全ての RAM ページ ポインタ レジスタにも同じことが言えます。必要に応じ、fastcall16 関数を呼び出してこれらの値を保存してください。

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

注意事項：

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、また場合によってはストップ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. データがない場合、処理は行われません。
2. データがある場合は、アドレスまたはデータ バイトの受信または送信が行われます。
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 / 反復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（否定）応答されると、そのトランザクションに対して転送されていたそれ以降のバイトは、別のスタート / 反復スタート条件およびアドレスが検出されるまで無視されます。それ以外の場合は、I2CHW_Poll() 関数が呼び出されるまで、データ バイトごとに I²C バスがストールされます。Communication_Service_Type が Polled に設定されている場合この関数が呼び出されるまで、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、および SCL 線を L に保ち ACK 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付きで互換性を持ちます。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 割り込みをディセーブルにすることで、I2CHW をディセーブルにします。

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

注意事項：

関数の実行により A および X レジスタの内容は変更される可能性があり保証されません。大容量メモリ モデル (CY8C29xxx) の全ての RAM ページ ポインタ レジスタにも同じことが言えます。必要に応じ、fastcall16 関数を呼び出してこれらの値を保存してください。

BootLdrI2C_EnableSlave

説明：

I2C_CFG レジスタで Enable Slave ビットをセットして I^2C HW ブロックの I^2 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

注意事項：

関数の実行により A および X レジスタの内容は変更される可能性があり保証されません。大容量メモリ モデル (CY8C29xxx) の全ての RAM ページ ポインタ レジスタにも同じことが言えます。必要に応じ、fastcall16 関数を呼び出してこれらの値を保存してください。

BootLdrI2C_DisableSlave

説明：

I2C_CFG レジスタで Enable Slave ビットをクリアして、I^2C スレーブ関数をディセーブルにします。

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

注意事項：

関数の実行により A および X レジスタの内容は変更される可能性があり保証されません。大容量メモリ モデル (CY8C29xxx) の全ての RAM ページ ポインタ レジスタにも同じことが言えます。必要に応じ、fastcall16 関数を呼び出してこれらの値を保存してください。

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

BootLdrI2C_InitWrite

説明：

BootLdrI2C_InitWrite ルーチンは、スレーブのデータ保存のためデータ 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 0 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

注意事項：

関数の実行により A および X レジスタの内容は変更される可能性があり保証されません。大容量メモリ モデル (CY8C29xxx) では、すべての RAM ページ ポインタ レジスタでも同じです。必要に応じ、fastcall16 関数を呼び出してこれらの値を保存してください。

BootLdrI2C_InitRamRead

説明：

BootLdrI2C_InitRamRead ルーチンは、スレーブがデータを検索するためにデータ 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 0 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

注意事項：

関数の実行により A および X レジスタの内容は変更される可能性があり保証されません。大容量メモリ モデル (CY8C29xxx) の全ての RAM ページ ポインタ レジスタにも同じことが言えます。必要に応じ、fastcall16 関数を呼び出してこれらの値を保存してください。

BootLdrI2C_InitFlashRead

説明：

BootLdrI2C_InitFlashRead ルーチンは、スレーブがデータを検索するためにフラッシュ 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 0 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

注意事項：

関数の実行により A および X レジスタの内容は変更される可能性があり保証されません。大容量メモリ モデル (CY8C29xxx) の全ての RAM ページ ポインタ レジスタにも同じことが言えます。必要に応じ、fastcall16 関数を呼び出してこれらの値を保存してください。

読み取りステータス ビットはクリアされます。

BootLdrI2C_bReadI2CStatus

説明：

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	変更内容
I2CHW_RD_NOERR	01h	データがマスタに読み取られ、ISR を通常に終了した。
I2CHW_RD_ OVERFLOW	02h	利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った。
I2CHW_RD_ COMPLETE	04h	読み取りが完了した。
I2CHW_READFLASH	08h	次の読み込みはフラッシュメモリ内の位置から。
I2CHW_WR_NOERR	10h	データがマスタによって正常に書き込まれた。
I2CHW_WR_OVERFLOW	20h	マスタが書き込みバッファに対して過剰にデータを書き込んだ。
I2CHW_WR_COMPLETE	40h	マスタの書き込みが、新しいアドレスまたはストップ条件によって完了した。
I2CHW_ISR_ACTIVE	80h	I^2C ISR が作動中。

注意事項：

関数の実行により A および X レジスタの内容は変更される可能性があり保証されません。大容量メモリ モデル (CY8C29xxx) の全ての RAM ページ ポインタ レジスタにも同じことが言えます。必要に応じ、fastcall16 関数を呼び出してこれらの値を保存してください。

BootLdrI2C_ClrRdStatus

説明：

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

注意事項：

関数の実行により A および X レジスタの内容は変更される可能性があり保証されません。大容量メモリ モデル (CY8C29xxx) の全ての RAM ページ ポインタ レジスタにも同じことが言えます。必要に応じ、fastcall16 関数を呼び出してこれらの値を保存してください。

BootLdrI2C_ClrWrStatus

説明：

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

注意事項：

関数の実行により A および X レジスタの内容は変更される可能性があり保証されません。大容量メモリ モデル (CY8C29xxx) の全ての RAM ページ ポインタ レジスタにも同じことが言えます。必要に応じ、fastcall16 関数を呼び出してこれらの値を保存してください。

ファームウェア ソースコードの例

以下の図に示すようにユーザ モジュール パラメータを設定します。アセンブリ言語と C 言語両方の例です。

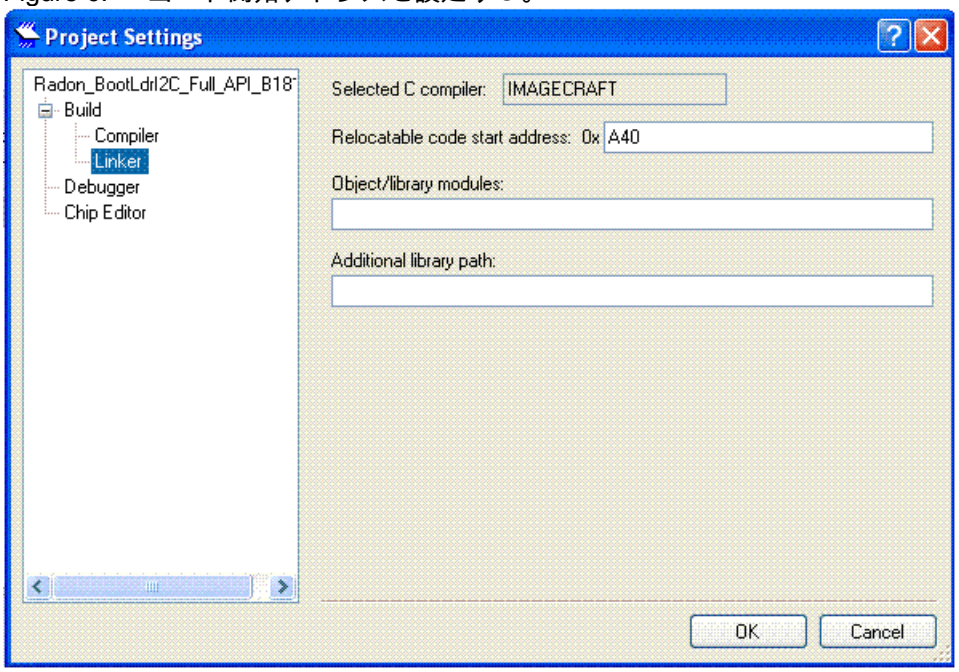
Figure 5.　ユーザ モジュール パラメータ

Parameters - BootLdrI2C_1

Name	BootLdrI2C_1
User Module	BootLdrI2C
Version	1.3
Slave_Addr_HEX	0x1
Boot_Loader_Addr_HEX	0x0
Read_Buffer_Types	RAM ONLY
Communication_Service_T	Interrupt
ApplicationCode_Start_Blo	0x29
BootLoaderKey	0001020304050607
Flash_Program_Temperatu	-20C
Ignore_N_I2C_Prefix_Byte	2
BootLdrI2C_ver	0x1000
I2C Clock	100K Standard
I2C_Pin	P[1]5-P[1]7

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てください。

Figure 6.　コード開始アドレスを設定する。



C 言語で書いた BootLdrI2C ユーザ モジュールの実装例。

```
//-------------------------------------------------------------------------
// 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(void)
{
//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 BootLdrI2C ユーザ モジュールの実装例。

```
;-------------------------------------------------------------------------
; 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

BootLdrI2C ユーザーモジュールが取り扱う PSoC のレジスタです。

Table 1. 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	予約済み	PinSelect	Bus Error IE	Stop IE	Clock Rate[1]	Clock Rate[0]	Enable Master	Enable Slave

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

Bus Error Interrupt Enable(バス エラー割り込みイネーブル): バス エラー時の I^2C 割り込み発生をイネーブルにします。

Stop Error Interrupt Enable(停止エラー割り込みのイネーブル): I^2C ストップ条件での I^2C 割り込みをイネーブルにします。

Clock Rate(クロック レート)[1,0]): 50、100 および 400 kbps からクロック レートを選択します。(400 kbps は CPU_Clk_speed が 6 MHz を超える場合に可能)

Enable Master(マスタイネーブル): I^2C HW ブロックをバス マスタとしてイネーブルします。

Enable Slave(スレーブイネーブル): I^2C 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. 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	Lost Arb	Stop Status	ACK out	Address	Transmit	Last Rec'd Bit (LRB)	Byte Complete

Bus Error(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。

Lost Arbitration(アービトレーションロスト): MultiMaster モードで、アービトレーションロスト （ このデバイスがバスを制御しない) を示します。

Stop Status(ストップステータス): I^2C ストップ条件の検出を示します。

ACK out(ACK 出力) : 受信した 1 バイトに関して I^2C ブロックに ACK(1) または NAK(0) の出力を指示します。

Address(アドレス): 受信されたまたは送信されるバイトがアドレスであることを示します。

Last Received Bit (最後に受信したビット)(LRB): 送信シーケンス最後の第 9 ビットの値、送信先デバイスからの ACK / NAK 応答のステータスです。

Byte Complete(バイト完了) : 8 つのデータ ビットが受信されたことを示します。受信モードでは、バスが ACK/NAK を待機するためストールします。送信モードでは、ACK/NAK 応答は受信されるので (LRB を参照)、バスはそれ以後次の動作を待ってストールします。

Table 3. 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	Data（データ）							

受信または送信データ。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. 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	予約済み	予約済み	予約済み	予約済み	Bus Busy	Master Mode	Restart Gen	Start Gen

Bus Busy(バス ビジー): マスタ専用、バススタート条件が検出されるとセットされ、ストップ条件が検出されるとクリアされます。

Master Mode(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。

Restart Gen(反復スタート条件発行): マスタ専用。I^2C バスの反復スタート条件を発行するためにセットします。

Start Gen(スタート条件発行): 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 バススタート条件を発行し、I^2C アドレスを転送します。

付属資料

以下のセクションには、I^2C ブートローダ作成時に役立つ可能性のある情報を記載します。

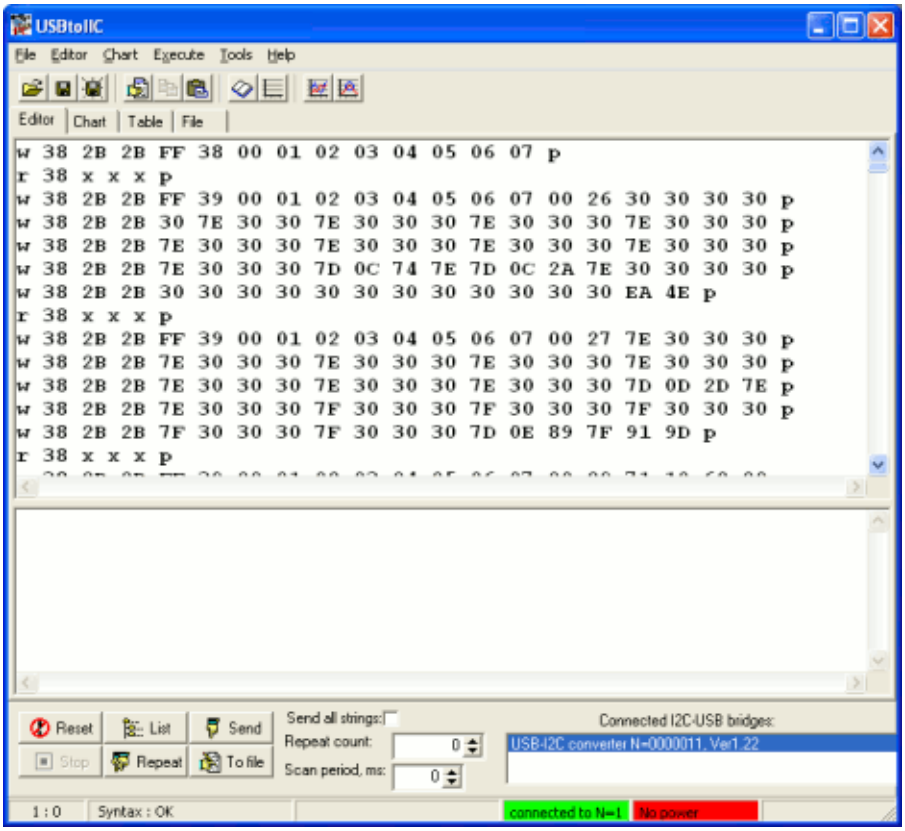
USBtoIIC ブリッジ GUI アプリケーションの使用

USBtoIIC ブリッジ とその GUI は、ブートローダへのダウンロードに適した方法です。

詳細は、アプリケーションノート 「I^2C Bootloader Using CY3240 I^2C-USB Bridge」を参照してください AN45683。このアプリケーションノートでは、<project_name>.txt ファイルの書式とプロジェクトをブートロードする手順について説明します。さらに、.dld 書式を .txt 書式に変換するツールについても説明します。この操作は、このユーザーモジュールデータシートに記載されたデバイスに必須ではありません。<project_name>.txt ファイルは自動的に生成されます。

以下、USBtoIIC ブリッジについて簡潔に説明します。

1. 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。
2. <projectname>.txt ファイルを、USBtoIIC ブリッジ GUI にインポートします。
3. File（ファイル）> Open（開く）を選択し、ブートロードしたいプロジェクトの出力ディレクトリを参照します。<projectname>.txt という名前のファイルを選択します。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、このデータシート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。ファイルを選択してからロードするのに二三秒かかることがあります。ロード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。



4. 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、所望のレベルに電源電圧を設定します。この電源はターゲットシステムの電源条件に応じて電源電圧を供給できます。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。



5. Send ボタンをクリックして、新コードをダウンロードします。各種 I2C トランザクションのステータスがステータス領域に表示されます。 「+」は トランザクションの完了を表します。

BootLdrI^{2}C ダウンロード (.dld file) フォーマット

このセクションでは、<project_name>.dld ファイルのフォーマットについて簡潔に説明します。

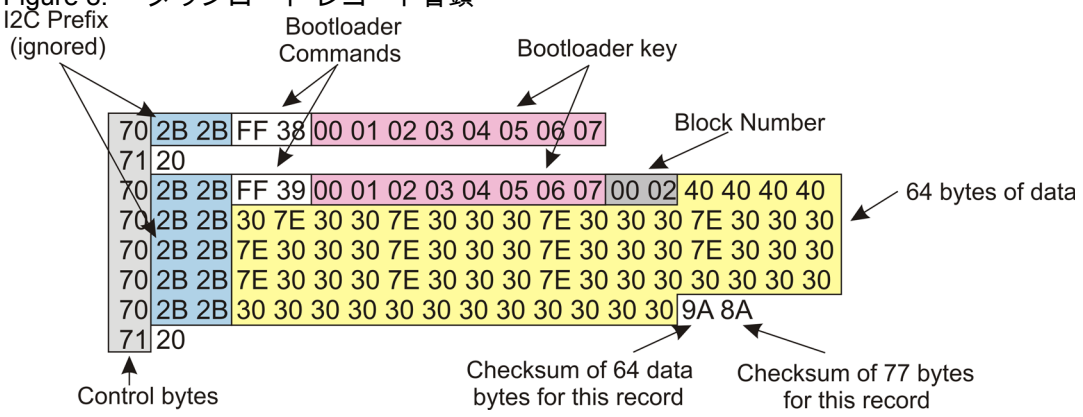
以下は 2 つのダウンロードレコード例で、冒頭、三番目そして最後の部分です。これらのレコードは、ブートロードされる I^{2}C マスタとスレーブ間を転送される実際のデータで構成されています。レコードのフォーマットはこのセクションに記載されています。

Figure 7. レコード例

```
            First Download Record
70 2B 2B FF 38 00 01 02 03 04 05 06 07                    Enter bootloader FF, 38
71 20                                                     Master reads bootloader slave to acquire status
70 2B 2B FF 39 00 01 02 03 04 05 06 07 00 02 40 40 40 40  No status request  or response
70 2B 2B 30 7E 30 30 7E 30 30 30 7E 30 30 30 7E 30 30 30
70 2B 2B 7E 30 30 30 7E 30 30 30 7E 30 30 30 7E 30 30 30
70 2B 2B 7E 30 30 30 7E 30 30 30 7E 30 30 30 30 30 30 30
70 2B 2B 30 30 30 30 30 30 30 30 30 30 30 30 9A 8A
71 20                                                     Status request and response
        ..............................................................
            Last Download Record
70 2B 2B FF 39 00 01 02 03 04 05 06 07 00 FF 30 30 30 30  No status request  or response
70 2B 2B 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30
70 2B 2B 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30
70 2B 2B 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30
70 2B 2B 30 30 30 30 30 30 30 30 30 30 30 30 00 53
71 20                                                     Status request and response
70 2B 2B FF 3B 00 01 02 03 04 05 06 07                    Exit bootloader FF, 3B
71 20*                                                    Status request and response
```

図 8 はレコード冒頭のフォーマット詳細です。

Figure 8. ダウンロード レコード冒頭



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。他に考えられる応答を表 5 に示します。

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。表 6 のブートローダコマンドを使用します。

Table 6. ブートローダの 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 パラメータで設定できます。

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。3 番目のブロックにはチェックサム情報が含まれます。チェックサムブロックの書式は、このセクションの後半で説明します。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。

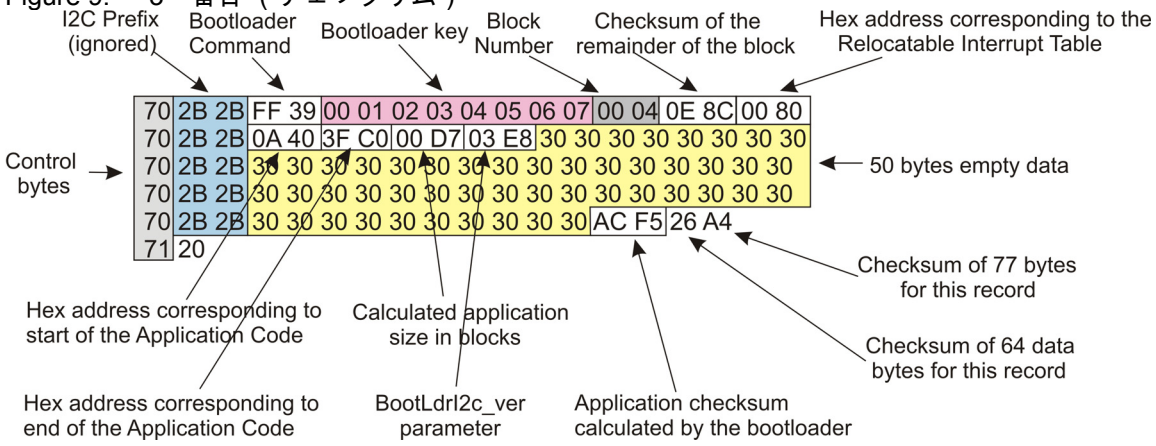
書き込みブロック コマンドの最初の行には、制御バイト、無視される 2-byte I^2C プリフィックス、書き込みブロック コマンド、ブートローダキー、送信されるブロック番号、および最初の 4 バイトのデータが含まれます。この例のブロック番号は 0x0002 で、これは ROM アドレス 0x0080 に対応します。

次の 3 行には、制御バイト、I^2C 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^2C 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 0x9 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 0x8A は、アドレス バイトとプリフィックスを除く 77 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。

送信されたブロックはすべて、同じ書式を持ちます。3 番目のブロックにはチェックサム情報が含まれます。図 9 はこのレコードのフォーマット詳細です。

Figure 9. 3 番目 (チェックサム)



3 番目のレコードにはチェックサムブロックが含まれます (ブロック番号は常に 0x0004 です)。レコードのデータについてはこのセクションで説明しています。

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^2C 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 0x0E8C、が含まれています。行の最後の 2 バイトには再配置可能な割り込みベクタテーブルの 16 進数のアドレスが含まれます (詳細は図 2 を参照)。

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 0x29 から計算される ApplicationCode_Start_Block 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 0xFF から計算されたアプリケーションノートコードの終了の 16 進数のアドレスです。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。

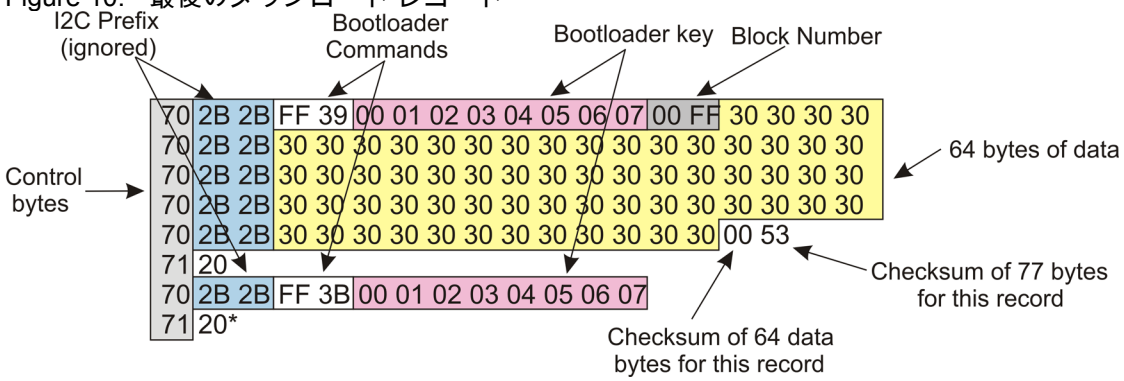
次の 2 行には、さらに空のデータ スペースが含まれています。チェックサム ブロックの最後の行には、行の最後の 4 バイトまで空のデータ スペースが含まれています。最後の 4 バイトは、ブートローダによって計算される 2 バイトのアプリケーション チェックサム、Intel ヘキサレコードからの 1 バイトのブロック チェックサム、最後にアドレス バイトとプリフィックスを除いた 77 バイトのレコード全体のチェックサムが含まれています。アドレス バイトとプリフィックスは、受信時にブートローダによって内部で確認されます。

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。

図 10 は、最後のダウンロードレコードを示します。

Figure 10. 最後のダウンロード レコード



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、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。これはチェックサム ブロックで提供されるパラメータを使用して行います。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。

BootLdrI2C と E2PROM ユーザモジュールの共存

ブートローダプロジェクトに E2PROM ユーザモジュールを配置する場合、カスタマ保護ブロック領域に E2PROM ブロックを割り当てます。この領域はブートローダコード領域とアプリケーションコード領域の間です。詳細はブートローダメモリ構成を参照してください。これにより、E2PROM ブロックはアプリケーションコード領域ではなく、アプリケーションチェックサムの一部として計算されないことが保証されます。

変更履歴

バージョン	担当	変更内容
1.2	DHA	変更履歴を追加
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 0x0140 に配置。 5. SetTemperature() 関数を追加。 6. BootLdrAPI 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。
2.10	DHA	1. SSC 呼び出し全体に .nocc を伴う .Literal と .EndLiteral のステートメントを再配置。 2. エクスポート `@INSTANCE_NAME`_EnterBootloader ステートメントを削除。 3. I2C アドレス比較カスタマイズにユーザコードセクションを追加。
2.20	DHA	1. Application_Checksum_Block と TWO_Block_Relocatable_Interrupt_Table. の初期化を更新。 2. Workspace Explorer でブートローダ出力ファイルの表示サポートを追加。 3. 「Flashsecurity.txt における不適切な設定」セクションの説明を更新 4. 「I2C およびスリープ」セクションを追加。 5. CY8C27x43 デバイスの flashsecurity.example, custom.lkp および boot.tpl テンプレートへのパスを訂正。

Note PSoC Designer 5.1 から、全ユーザ モジュール データシートに変更履歴を追加しました。このセクションでは、ユーザー 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ついて簡単な解説を掲載しています。

Copyright © 2007-2012 © 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。