AN73854

PSoC – ブートローダの導入

著者: Mark Ainsworth

関連製品ファミリ:すべての PSoC® 3, PSoC 4, および PSoC 5LP

関連アプリケーションノート: 関連文書を参照してください

さらにサンプルコードをお求めでしょうか？以下のとおり対応いたします。

PSoC のサンプルコードのリストにアクセスするには、サンプル コードのウェブページをご覧ください。Cypressのビデオ ライブラリについてはここからご覧ください。

AN73854 ではブートローダの理論と技術の概要を紹介します。また、PSoC Creator™を使用して PSoC® 3, PSoC 4, および PSoC 5LP MCU にブートローダをすばやく簡単に実装する方法を示します。

目次

1 はじめに

このアプリケーションノートでは、ブートローダの基本とデザイン原則の概要を示し、PSoC Creator のプロジェクトで PSoC 3、PSoC 4、および PSoC 5LP にどのように実装されるかを示します。

注: PSoC 6 MCU の導入により、サイプレスは統合開発環境 (IDE) である ModusToolbox™ と同様にデバイスファームウェアアップデート (DFU) SDK を開発しました。このアプリケーションノートの対象は PSoC 3、PSoC 4、および PSoC 5LP に限定しています。PSoc 6 での DFU の詳細については AN213924, PSoC 6 MCU デバイス ファームウェア アップデート ソフトウェア開発キットガイド、および関連するサンプルコードを参照してください。

このアプリケーションノートは PSoC および PSoC Creator 統合開発環境 (IDE) に精通していることを前提としています。PSoC を初めて使用される場合は以下のいずれかの入門アプリケーションノートを参照してください。

- AN54181 - PSoC® 3 入門
- AN79953 - PSoC 4 入門
- AN77759 - PSoC 5LP 入門

PSoC Creator を初めて使用される場合は PSoC Creator ホームページを参照ください。

ブートローダ全般が初めて使用される場合、「ブートローダとは?」および「ブートローダ設計の一般的な考慮事項」で説明されている基本概念および設計原則を参照してください。

ブートローダに精通していて、PSoC Creator を使用して PSoC デバイスにどのように実装するのかを参照したい場合は、PSoC ブートローダ – 使用方法を参照してください。

ご使用の PSoC Creator のプロジェクトへブートローダを追加する方法については PSoC Creator へのブートローダの追加を参照してください。

I²C、UART、SPI、および USB に関連するブートローダ アプリケーションノートのリストは、関連文書を参照してください。このセクションにリストされている各ブートローダのアプリケーションノートには、サンプルコードが関連付けられています。

ブートローダに関連するサンプルプロジェクトは PSoC Creaotr のメニューオプション **File** > **Code Example** からもアクセスできます。ポップアップウィンドウで「ブートローダ」を検索してブートローダに関連するプロジェクトをフィルタします。

PSoC 3、PSoC 4、および PSoC 5LP のサンプルコードのリストについてはここ をクリックしてください。

2 サイプレスのリソース

サイプレスは www.cypress.com で豊富なデータを提供しており、最適なデバイスの選定、迅速で効果的な設計への統合に役立ちます。以下は、このアプリケーションノートに関連するリソースの簡略リストです。

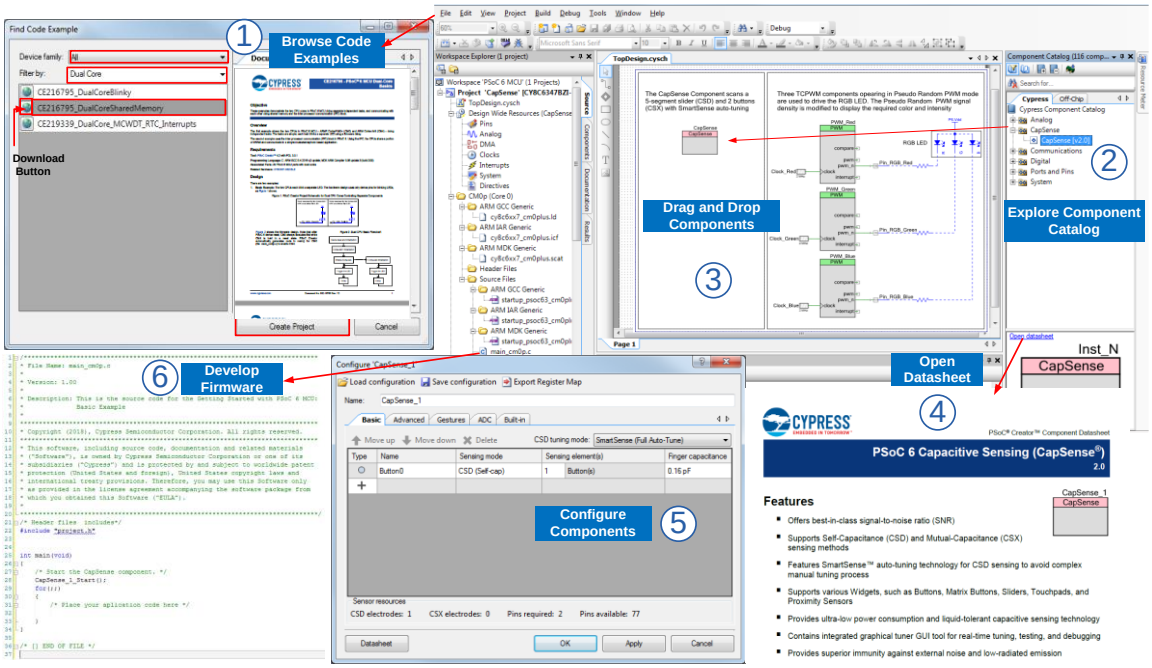
- **概要:** MCU ポートフォリオ, ロードマップ
- **製品セレクタ:** PSoC 1、PSoC 3、PSoC 4、PSoC 5LP、または PSoC 6。加えて PSoC Creator も製品選定ツールに含む。
- **データシート:** PSoC デバイスファミリの電気的仕様を記述し提供
- **アプリケーションノート:** 基本レベルから上級レベルまでの幅広いトピックを提供
- **サンプルコード:** PSoC 3、PSoC 4、および PSoC 5LP 用、または PSoC 6 用を用意
- PSoC **テクニカルリファレンスマニュアル (TRM):** PSoC デバイスファミリのアーキテクチャおよびレジスタについての詳細情報を提供
- **トレーニングビデオ:** これらのビデオでは様々なサイプレス製品ファミリやツールを用いた入門用のガイダンスを提供します。
- **CapSense デザインガイド:** 容量性タッチセンシングアプリケーションの設計方法を学習できます。
- **開発キット:** 以下を含むいくつかの例があります。
  - CY8CKIT-042 および CY8CKIT-040 パイオニアキット: 使いやすく廉価な PSoC 4 の開発プラットフォーム
  - CY8CKIT-049: は PSoC 4 デバイスをサンプリングする非常に低価格なプロトタイプ プラットフォーム
  - CY8CKIT-001: すべての PSoC ファミリデバイスに共通の開発プラットフォーム
  - MiniProg3 デバイス: フラッシュプログラミングおよびデバッグ用のインターフェースを提供

3 PSoC Creator

PSoC Creator は無料で利用できる Windows ベースの統合開発環境 (IDE) です。このツールにより PSoC 3、PSoC 4、PSoC 5 LP、および PSoC 6 をベースにしたハードウェアとファームウェアのシステムを同時に設計できます。PSoC Creator を使用すれば、以下のことができます。

1. 図 1 に示すように、File > Code Example のメニューからサンプルコード集をブラウズ
2. 100 以上のコンポーネントのライブラリを探索
3. ビルドするハードウェアシステム設計のコンポーネントをメイン設計ワークスペースにドラッグアンドドロップ
4. コンポーネントのデータシートをレビュー
5. コンフィグレーションツールを使用してコンポーネントをコンフィグレーション
6. PSoC のハードウェアでアプリケーションのファームウェアを同時に設計

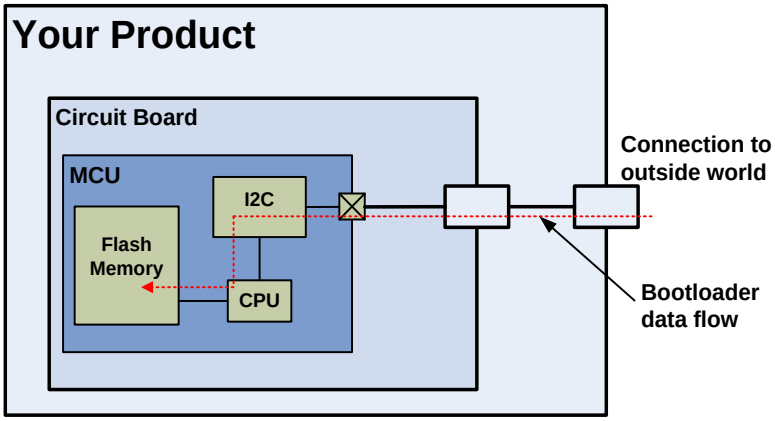
図 1. PSoC Creator の機能



4 ブートローダとは?

ブートローダは MCU システム設計の共通的な部品です。ブートローダにより製品のファームウェアをフィールドで更新できます。
一般的な製品ではファームウェアは MCU のフラッシュメモリに組み込まれています。図 2 に示すように MCU は PCB 上に実装されて製品に組み込まれます。

図 2. ブートローダデータフローのブロックダイヤグラム

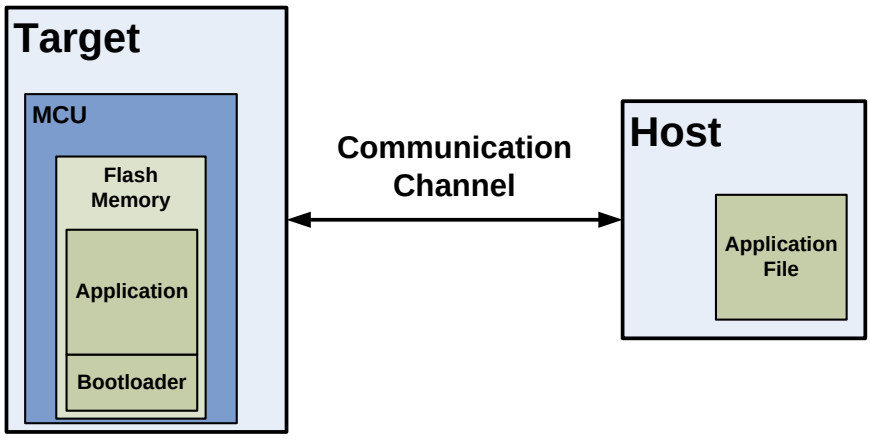


工場では、製品へのファームウェアの初期プログラミングは、一般的に MCU の Joint Test Action Group (JTAG) またはシリアル ワイヤー デバッガ (SWD) のインターフェースを介して実行されます。しかしながら、これらのインターフェースは通常、フィールドでは利用できません。製品を開封して PCB に直接アクセスするのは難しく、費用も大きくかかります。より良い方法の 1 つは、製品と外部との間の既存の接続を用いることです。接続には I^2C、USB、または UART のような一般的なものやカスタムプロトコルを用いることもできます。

4.1 用語および定義

図 2 は、製品の組込みファームウェアが、通常動作とアプリケーション更新という 2 つの異なる目的で通信ポートを使用することを示します。図 3 に示すように、フラッシュの更新方法を認識する組込みファームウェアの部分は、**ブートローダ**と呼ばれます。.

図 3. ブートローダシステム



一般に、フラッシュを更新するデータを提供するシステムは**ホスト**、更新するシステムは**ターゲット**と呼ばれます。ホストは外部の PC またはターゲットと同じ PCB にある別の MCU です。ホストからターゲットのフラッシュにデータを転送する動作は**ブートローディング**、**ブートロード動作**、または単に**ブートロード**と呼ばれます。フラッシュに配置されるデータは**アプリケーション**または**ブートローダブル**と呼ばれます。

ブートローディングのもう 1 つの共通用語は**インシステム プログラミング (ISP)** です。サイプレスには似た名前のインシステム シリアル プログラマ (In-System Serial Programmer: ISSP) と呼ばれる製品、およびホストソース シリアル プログラミング (Host-Sourced Serial Programming: HSSP) と呼ばれる動作があります。詳細については、ブートローダの代替を参照してください。

4.2 ブートローダの使用

通信ポートは通常、ブートローダとアプリケーション間で共有されます。ブートローダを使用するための最初のステップは、アプリケーションではなくブートローダが実行するように製品を操作することです。

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

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

4.3 ブートローダの機能フロー

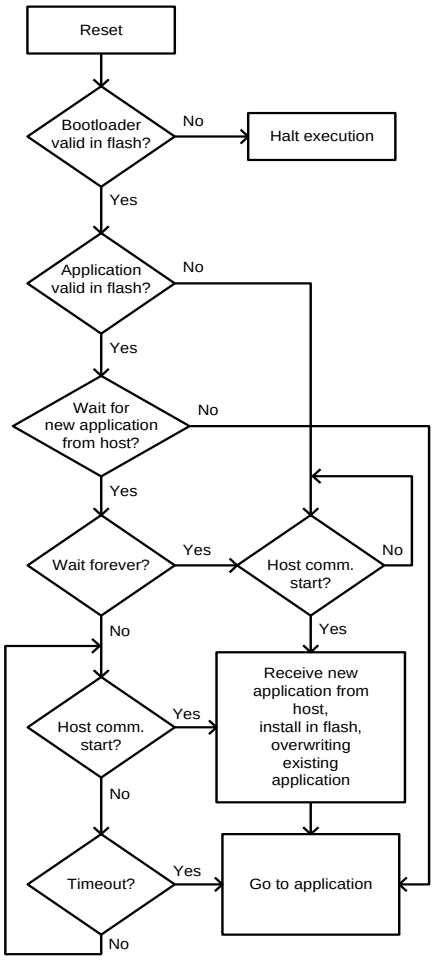
ブートローダは通常リセット時に実行されます。(メモリ仕様とモジュラー設定を参照してください) 実行後は次の動作が実行できます。

- 実行する前にアプリケーションの有効性をチェック
- ホストとの通信を開始するタイミングを管理
- ブートロード／フラッシュ更新動作を実行
- 最後に、アプリケーションに制御を渡す

図 4 は実行され方をフロー図で示したものです。

注: PSoC Creator はデュアルアプリケーションオプションをサポートしています。図 4 の“Go to application”ではより複雑に動作します。 詳細についてはアプリケーション起動プロセスを参照してください。

図 4. Bootloader 機能フロー



5 ブートローダ設計の一般的な考慮事項

ブートローダシステムの設計を行う際には留意すべき多くの考慮事項があります。

5.1 ブートローダの代替

前項で示したように、ブートローダは、フィールドで製品のファームウェア更新を可能にします。

この問題を解決する方法は他にもあります。例えば、フラッシュ更新機能はアプリケーション自体にコード化できます。しかしながら、アプリケーションは一部またはアプリケーション自体を上書きできる必要があるため、複雑さが増します。ブートローダを個別のモジュールまたはプログラムとして分割して設計することを推奨します。

ブートローダの代わりとしては、その他に HSSP を使用する方法があります。この方法は MCU の JTAG または SWD ピンを直接外部のホストから制御して、新しいアプリケーションをフラッシュにプログラムします。サイプレスの CY8CKIT-002 MiniProg3 およびその他のプログラマはこの方法を使っています。

けれども HSSP は一般的に開発中や工場でのプログラミングに用いられるもので、フィールドでは一般的ではありません。フィールドで最も頻繁に HSSP が使用されるのは、複数の MCU を搭載した PCB で、1 つの MCU が直接もう一方の MCU をプログラムする場合です。

PSoC の JTAG/SWD ピンの詳細は以下を参照してください。

- AN61290 – PSoC 3 and PSoC 5LP Hardware Design Considerations
- AN88619 – PSoC 4 and Hardware Design Considerations
- 表 1 に示した要求デバイスのプログラミング仕様

表 1. プログラミング仕様

デバイスファミリ	プログラミング仕様
PSoC 3	PSoC 3 Programming Specifications
PSoC 5LP	PSoC 5LP Programming Specifications
PSoC® 4000S, PSoC 4100M, PSoC 4100S, PSoC 4200D, PSoC 4200M and PSoC 4100PS	PSoC® 4000S, PSoC 4100M, PSoC 4100S, PSoC 4200D, PSoC 4200M, and PSoC 4100PS Programming Specifications
PSoC 4000	CY8C4000 Programming Specifications
PSoC 4100/4200	CY8C41XX CY8C42XX Programming Specifications
CYBL10X6X, CY8C4127_BL, CY8C4247_BL	CYBL10X6X, CY8C4127_BL, CY8C4247_BL Programming Specifications
CYBL10x7x, CY8C4128_BL, CY8C4248_BL (256K), CY8C4246_L, CY8C4247_L, CY8C4248_L	CYBL10x7x, CY8C4128_BL, CY8C4248_BL (256K), CY8C4246_L, CY8C4247_L, CY8C4248_L Programming Specifications

HSSP の詳細については、以下を参照してください。

- AN73054 – PSoC 3 and PSoC 5LP Programming Using an External Microcontroller (HSSP)
- AN84858 – 外部マイクロコントローラーを使用した PSoC 4 プログラミング (HSSP)

5.2 メモリ仕様とモジュラー設定

前項ではブートローダのコードはアプリケーションのコードとは分けておく必要があることを注意点としてあげました。多くの場合それらは完全に分けられたモジュールとして設計されます。両方のモジュールがフラッシュに置かれる場合、どの領域に置けばよいでしょうか? いくつかの MCU は、図 5 に示すように、フラッシュとは別にハードコードされたブートロード読み出し専用メモリ (ROM) が含まれます。他の MCU は、図 6 に示すように、ブートローダにフラッシュの一部を使用します。

図 5. ブートローダが ROM にある場合

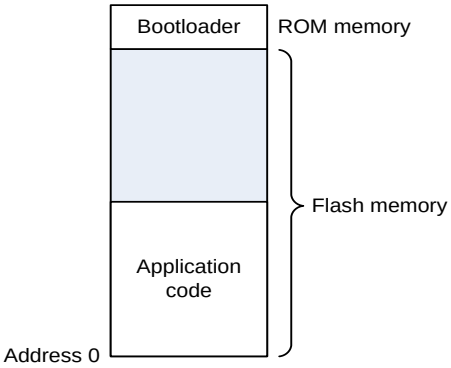
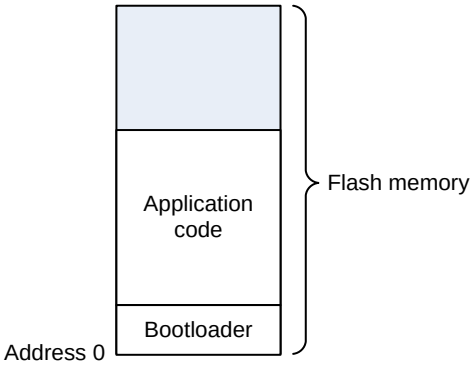


図 6. フラッシュメモリ内のブートローダ



ROM 方式では、アプリケーションはフラッシュのすべてを使用できます。フラッシュ方式では、柔軟なブートローダ設計が可能です。フラッシュ方式が一般的に好まれます。ブートローダは通常、アドレス 0 から始まるようにフラッシュに配置されます。ほとんどの CPU はアドレス 0 からコードを実行するため、ブートローダはデバイスのリセット時に最初に実行されます。

フラッシュ方式の潜在的な問題の 1 つは、アプリケーションが使用できるメモリをブートローダが使用することです。これは、アプリケーションの設計、または MCU がより多くのメモリが必要になった場合のコストに影響する可能性があります。そのためフラッシュ方式では、ブートローダをできるかぎり小さく設計しなければなりません。サイズを削減し、要件と設計をシンプルに保つにはメモリ使用とモジュラー設定を参照してください。もし追加の機能を含めなければならない場合、ブートローダを小さく保つのは難しくなります。カスタマイズを参照してください。

5.3 ブートローダ - ホストタイミング

ブートローダの設計で重要な考慮事項としてホストとの通信を開始するタイミングがあります。図 4 で示すように、アプリケーションが有効であると判断した後、ホストが新たなブートロード動作を開始するまで、ブートローダは一定時間待機できます。

待機時間は通常 50～500ms です。待機時間が短かすぎる場合には、ホストが確実に通信を開始できない場合があります。長すぎる場合には、製品全体の起動時間が長くなりすぎる場合があります。

タイミングの問題への別の解決策は、アプリケーションにブートローダを呼び出させることです。次に、アプリケーションは、ボタンの押下やホストからのメッセージなどの外部イベントに応答して、ブートロード動作を開始できます。

5.4 通信ポート

ほとんどの場合、図 3 に記載されている通信ポートの仕様は製品全体の要件に従って設定されます。ブートローダシステムの強力なサポートのためには、さらにポートを以下のようにする必要があります。

- **パケットベースのデータ通信** ポートはパケットを解析しないでください。ブートローダとホストがそれを行います。
- **パケットエラー検知** ポートは不正なデータを含むパケットを検知して報告できるようにしてください。システムの残りの部分は不正なパケット報告を処理できるようにしてください。
- **コマンド応答プロトコル** 通常、ホストはコマンドパケットをブートローダへ送信後、成功／失敗／ステータス応答パケットを待ちます。
- **中速転送** フラッシュの 1 行を書き込むのに数ミリ秒かかる場合があるため、高速ポートを使用しても、全体のブートロード時間を大幅に改善できない場合があります。
- **フラッシュ書き込み中に発生する転送** 前の行がフラッシュに書き込まれている間にデータの行をダウンロードできます。

組込みシステムで一般的に用いられているプロトコルの中で USB は最もこれらの機能をサポートしますが、USB のコードは大量のフラッシュを使用します。UART、I^2C、および SPI はよりシンプルですが、パケット管理には追加のコードが必要になります。I^2C はマスタ側 (一般的にホスト) によってのみ制御されるため、コマンド応答のプロトコルを実装することはより難しくなります。

5.5 障害からの回復

ブートロード動作中に電源障害、通信の途絶、およびフラッシュ書き込みエラーなどが発生した場合に、ブートローダはエラーの検出、通知、および適切な処理ができなければなりません。

いくつかのチェックビット (チェックサムや CRC) をアプリケーションでフラッシュに格納することが、よく行われる方法としてあります。ブートロード動作が開始されると、ビットはクリアされます。アプリケーションのダウンロードとインストールが成功するとそのビットは更新されます。例えば、電源障害がブートロード中に発生した場合、リセット時にブートローダは不正なチェックビットを検出し、部分的にロードされたアプリケーションに制御を渡しません。そのかわりに、ホストが別のブートロード動作が開始するのを待ちます。

5.6 将来性

もう 1 つの設計上の考慮事項は、インストール後、ブートローダはフィールドで更新する必要がないことです。フィールドで自分自身を上書きまたは更新できるブートローダを作成することは可能ですが、それは複雑であり、回避するのが最善です。ブートローダを堅牢で将来性のあるものにするための鍵は、要件と設計をシンプルに保つことです。欠陥を回避するには、ベストプラクティスでのコーディングと、コードレビューを実施することです。

ブートローダとアプリケーションは別のモジュールのため異なるコンパイラの使用や、ビルド用に異なる開発システムでさえも使用できます。コンパイラなどのツールはそれぞれバージョンが変更になる場合があるため、2 つのモジュール間の転送制御メカニズムが変わりないかどうか確認してください。また、将来開発ツールをアップグレードする場合は、古いブートローダで新しいアプリケーションをロードできるかどうか確認してください。

5.6.1 アプリケーション管理

ブートローダとアプリケーションは別のモジュールのためアプリケーションを変更できます。ブートローダからアプリケーションへ制御を移行する最適な方法を考慮する必要があります。以下にいくつかの方法を示します。

1. アプリケーションがいつもスタートする決まった場所へジャンプさせる。この方法はシンプルですが、ブートローダまたはアプリケーションを将来変更する際の柔軟さに欠けます。
2. アプリケーションの開始アドレスをフラッシュの共通領域に保持する。ブートローダは、その場所をアプリケーションの開始アドレスへのポインタとして使用します。
3. 共通の開発システムを用いてアプリケーションをブートローダにリンクさせる。ブートローダはそこへジャンプするシンボリックアドレスを持ちます。

2 番目の方法が最もシンプルで柔軟性のある組合せであり、一般的に好まれるソリューションです。

5.6.2 フラッシュ保護

ブートローダは、フラッシュメモリ内の自身のイメージが有効かどうか確認する必要があります。有効でない場合、実行を止めなければなりません。残念ながら、これにより製品は使用できなくなります。

ブートローダをフラッシュ内で有効な状態に保つための最適な方法は、ブートローダがファームウェアによって上書きされないようにハードウェアを使用することです。これを行う 1 つの方法は、ブートローダフラッシュへの偶発的な上書きを防ぐフラッシュ書き込み保護回路を使用することです。PSoC への実装の詳細については フラッシュ保護 を参照してください。

5.7 カスタマイズ

ブートローダは異なる製品アプリケーション向けに容易に変更できるように設計する必要があります。例えば、ブートローダシステムは複数の通信ポートが使われる場合も含めて異なる通信ポートを容易に使用できるようにしておく必要があります。

また、ブートローダシステムは高い信頼性が必要な製品で使用される場合があります。以下の 3 つの主要な側面があります。

- ブートローダからアプリケーションへの転送中に、重要なピンの状態を保持する必要がある場合があります。転送がデバイスのリセットを通じて行われる場合には問題となる可能性があります。 詳細については、付録 A を参照してください。
- 重要なタスクがブートロードと同時に完了する必要がある場合があります。マルチタスクのシステムを可能にするため、ブートローダに追加のコードが必要になる場合があります。11 ページのカスタマイズ を参照してください。
- 複数 (通常は 2 つ) のアプリケーションイメージがフラッシュに格納されています。もしそのうちの 1 つがフラッシュ内で破損した場合、ブートローダはもう一方のイメージへ制御を渡すことができ、製品の平均故障間隔 (MTBF) を削減します。PSoC Creator は、デュアルイメージ ブートローダに対応します。

6 PSoC ブートローダ – 使用方法

前のセクションでは、一般的なブートローダの機能と設計上の考慮事項について説明しました。ここで、PSoC Creator を使用して、PSoC 3、PSoC 4、および PSoC 5LP でこれらの原則がどのように実践されるかを見てみましょう。

注: PSoC 6 MCU の導入に伴い、サイプレスは、デバイスファームウェアアップデート (DFU) SDK と ModusToolbox 統合開発環境 (IDE) を開発しました。このアプリケーションノートの対象は、PSoC 3、PSoC 4、および PSoC 5LP に限ったままとしています。PSoC 6 での DFU の詳細についてには、 AN213924, PSoC 6 MCU Device Firmware Update Software Development Kit Guide および関連するサンプルコードを参照してください。

PSoC デバイスはメモリおよび設定可能なペリフェラルハードウェアを持っているため、非常に有能で柔軟なブートローダシステムの構築を可能にします。PSoC Creator を使用して開発してください。これは、PSoC ベースのソリューションをビルドするために使用するサイプレスが提供する無償の IDE です。PSoC デバイスの詳細については以下を参照してください。

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

PSoC Creator の情報については, PSoC Creator のホームページを参照してください。

注: PSoC 3、PSoC 4、および PSoC 5LP へのブートローダシステムの実装は PSoC 1 とは異なります。PSoC 1 ブートローダの詳細については AN2100, Bootloader: PSoC 1 を参照してください。

すべてのサイプレスの PSoC 製品および対応する IDE と同様に、PSoC Creator は基本システムの機能の実装を自動化することにより、設計時間を短縮しようとします。ブートローダも例外ではありません。シンプルな I^2C のブートローダをプロジェクトに追加するのは、文字通り数分でできます。追加方法の詳細については PSoC Creator プロジェクトへのブートローダの追加を参照してください。

6.1 PSoC Creator ブートローダプロジェクト

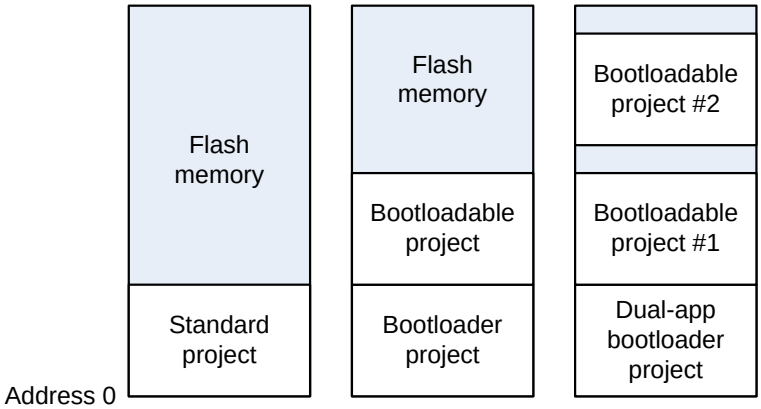
PSoC Creator は「プロジェクト」という用語を使用し、完全な自己完結型アプリケーションを定義します。CPU コードに加えて、プロジェクトにはアプリケーション用の PSoC デバイスのアナログおよびデジタルペリフェラルの設定用のデータも含まれます。

PSoC Creator では、ブートローダとアプリケーションは完全に別のプロジェクトとして実装されることを覚えておくことが重要です。使用可能なプロジェクトのタイプは、**Standard** (または「normal」、ブートローダなし)、**Bootloader**、および **Bootloadable** があります。4 番目のプロジェクトのタイプとして、**Dual-App Bootloader** があり、カスタマイズで説明した高信頼性が必要なアプリケーション向けに 2 つのアプリケーションのイメージに対応します。

ブートローダブルプロジェクトはブートローダプロジェクトに関連付ける必要があります。ブートローダプロジェクトは複数のブートローダブルプロジェクトに関連付けられます。

PSoC にはブートロード ROM を持っていませんので、図 7 に示すように、ブートローダはフラッシュに配置されます。ブートローダプロジェクトは、フラッシュの開始アドレスのゼロに配置され、デバイスがリセットされた際に最初に実行されます。5 ページの図 4 にプログラム実装のフローを示します。

図 7. PSoC Creator プロジェクトとフラッシュメモリ使用例



6.2 ブートローダオプション

PSoC Creator は、ブートローダ動作時の挙動設定オプションを持つブートローダコンポーネントを提供します。いくつかのオプションを以下に示します。

- **Wait for Command:** ブートローダブルへ制御を渡す前にホストからのコマンドを待つ (Yes) または 待たない (No)。
- **Wait for Command Time:** 1～2550 ms のタイムアウト、または無限。Wait for Command が Yes の場合にのみ有効。
- **Communication Component:** ブートローダが使用する通信コンポーネント。PSoC Creator は様々な種類の通信ポートに対応しており、カスタムオプションも含みます。
- **Checksum Type:** ホストとの間のパケットで使用するチェックビットのタイプ。Checksum または CRC。

詳細については 13 ページの図 9 および Bootloader Component データシートを参照してください。

6.3 通信コンポーネント

PSoC Creator のブートローダプロジェクトは、少なくとも 1 つのブートローダ互換の通信コンポーネントを含む必要があります。現在、I²C スレーブ、UART、SPI、および USBFS コンポーネントが標準としてブートローダに対応します。

標準以外の通信チャネルをブートロードに使用したい場合は、カスタムコンポーネントを容易に作成できます。スタート、ストップ、リセット、読み出し、および書き込みの 5 つの機能に対応するようにコンポーネントに API を記述しなければなりません。ブートローダのカスタム通信コンポーネントの作成方法の詳細については PSoC Creator の Component Author Guide を参照してください。

6.4 障害からの回復

PSoC Creator のブートローダ コンポーネントはアプリケーション (または dual-app option では両方のアプリケーション) のデータを格納するためにフラッシュの最上位行 (64 バイト、128 バイト、または 256 バイト) を使用します。このデータは、各アプリケーションのチェックサムと有効性ビットを含みます。ブートロードが開始されると、これらのビットはクリアされます。ブートロードが完全に成功したとき、それらは再計算され更新されます。

ブートロード動作中の電源異常や通信が途絶えた場合、ブートローダブルプロジェクトのチェックサムは、次のデバイスリセット時に不正な値となります。その後、ブートローダはホストからの別のブートロード動作を開始するコマンドを待ちます。

6.5 下位互換性

PSoC Creator は、ブートローダブルプロジェクトが新しいバージョンでビルドされても、古いバージョンでビルドしたブートローダとリンクし、互換性が保たれるよう設計されています。

6.6 ブートローダメモリ使用量

前述のように PSoC Creator ブートローダはアプリケーションで使用されるメモリを使用します。これはアプリケーションの設計、またはコストへ影響があります。そのためブートローダメモリ使用量は重要な考慮事項となります。PSoC Creator ブートローダコンポーネントのメモリ使用量は以下に応じて大きく異なります。

- 使用する通信コンポーネント
- ブートローダコンポーネント設定オプションの選択 (図 9 参照)
- ターゲットデバイス – PSoC 3、PSoC 4、および PSoC 5LP のそれぞれの CPU である 8051、Cortex-M0/M0+、および Cortex-M3
- コンパイラおよび最適化設定

詳細は Bootloader Component データシートの仕様表を参照してください。特定のブートローダプロジェクトについては、プロジェクトをビルドした後、コンパイラによって生成された.*map* ファイルをチェックして、正確なメモリ使用量を判別してください。

6.7 フラッシュ保護

すべての PSoC 3、PSoC 4、PSoC 5LP デバイスは柔軟なフラッシュメモリへのアクセスを制限するフラッシュ保護システムを含みます。この機能は独自のコードを保護するように設計されていますが、フラッシュメモリのブートローダ部分への不注意な書込みから保護するためにも使用できます。

フラッシュメモリは、デバイスファミリに応じて、64～256 バイトの行で構成されます。各行に対して 4 つ (PSoC 4 は 2 つのレベル) の保護レベルから 1 つを割り当てられます。表 1 を参照してください。フラッシュ保護レベルの変更は、完全にフラッシュの消去によってのみ可能です。詳細については、それぞれのデバイスごとのデータシートまたはテクニカルリファレンスマニュアル (TRM) の PSoC フラッシュとセキュリティ機能を参照してください。

表 1. フラッシュ保護レベル

	PSoC 3 および PSoC 5LP		PSoC 4	
保護設定	**許可**	**非許可**	**許可**	**非許可**
非保護	外部読出しおよび書込み 内部読出しおよび書込み	–	外部読出しおよび書込み 内部読出しおよび書込み	–
ファクトリー アップグレード	外部書込み 内部読出しおよび書込み	外部読出し	NA	NA
フィールド アップグレード	内部読出しおよび書込み	外部読出しおよび書込み	NA	NA
完全保護	内部読出し	外部読出しおよび書込み 内部書込み	内部読出し	外部書込み 内部書込み(下記注を参照)

注: PSoC 4 デバイスを外部読出し動作から保護する場合、PSoC Creator の *.cydwr* システム設定およびデバイスをプログラムする PSoC プログラマソフトウェアを使用して、デバイスの保護設定を"Protected"に変更してください。この保護設定を有効にするにはデバイスのプログラミングの前に **Options > Programmer Options** の"Chip Lock"を有効にする必要があります。

フラッシュのブートローダ部分を保護するには、対応する行を「完全保護」に設定します。PSoC Creator は各行の保護設定を簡単に選択できるようにしています。詳細については PSoC Creator のヘルプまたは関連文書にリスト化された高度なブートローダアプリケーションノートを参照してください。

6.8 カスタマイズ

ブートローダは PSoC Creator のプロジェクトの 1 つで、他の同様なプロジェクトと同様に、PSoC をアプリケーションに対して設定できます。これにより、特に高信頼性アプリケーションでのブートローダのカスタマイズが簡単になります。

- **ブートロード中の他のタスク**: コンポーネントをブートロードプロジェクトの回路図に追加できます。多くの場合、これらのコンポーネントは CPU を使用せずに複雑なタスクを実行できます。ブートロード中に CPU を他のタスクに使用したい場合、最も簡単な方法は、定期的な割込みハンドラに組み込まれたステートマシンとしてタスクを構造化することです。この方法でブートローダと 2 番目のタスクを独立したプロセスとして動作できます。
- **ピン状態の保持**: ピンコンポーネントを回路図に配置することができ、デバイスリセットとブートローダ起動の両方のピン状態を設定できます。ピン状態の調整の詳細については、AN61290 PSoC 3 and PSoC 5LP Hardware Design Considerations、または AN88619 PSoC 4 Hardware Design Considerations を参照してください。あわせて付録 A も参照してください。

7 PSoC Creator プロジェクトへのブートローダの追加

ここでは PSoC Creator でブートローダの実装方法をお見せします。具体的な手順を紹介していきます。詳細については、関連文書にリスト化された高度なブートローダアプリケーションノートを参照してください。

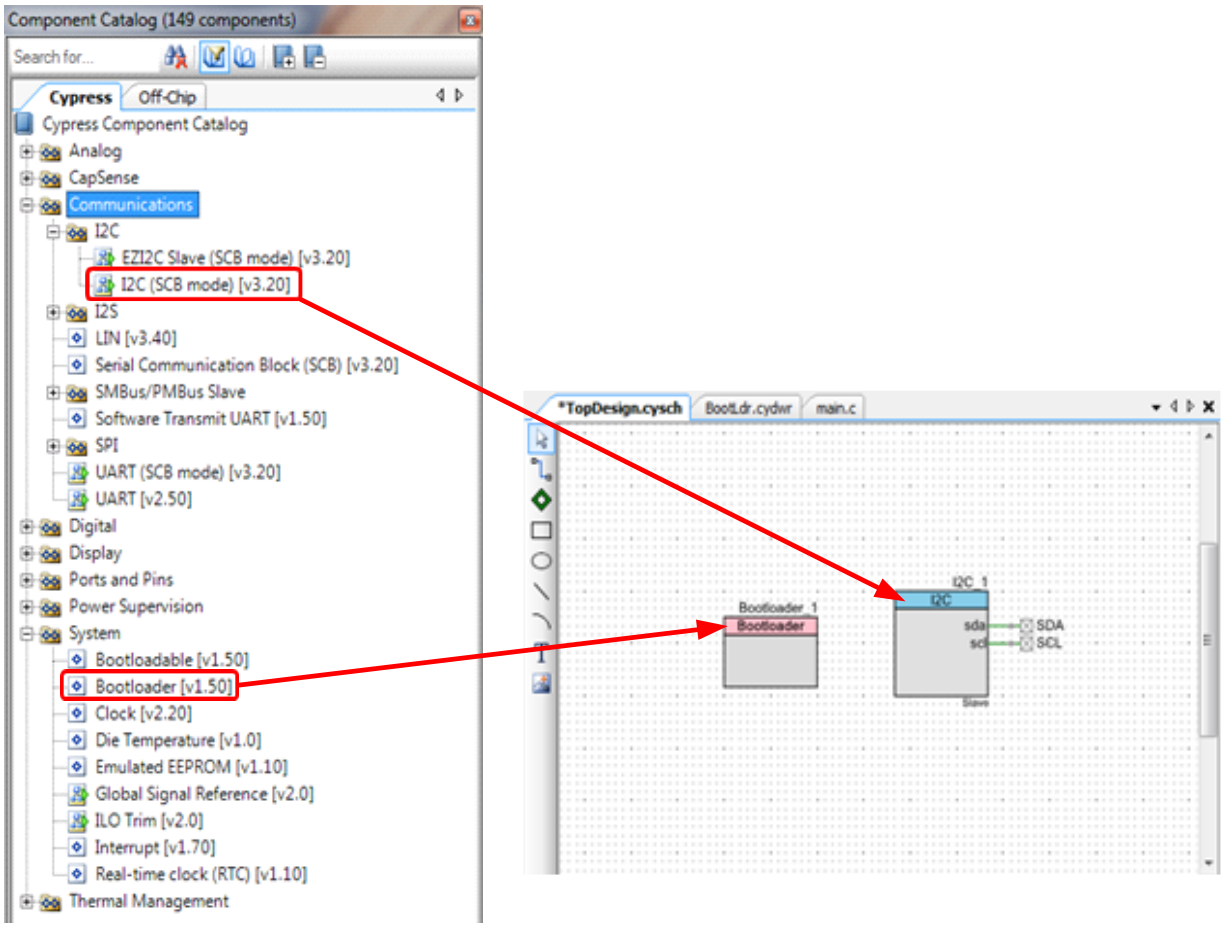
PSoC Creator 3.1 以前を使用している場合には付録 B を参照してください。

7.1 ブートローダのビルド

他の MCU の場合、一般的にアプリケーションにブートローダを追加します。しかし PSoC Creator でのベストプラクティスは逆方向の設計です。最初にブートローダプロジェクトを作成し、その次に 1 つ以上のブートローダブルプロジェクトを作成します。

新規プロジェクトを作成してください。プロジェクトの回路図にブートローダコンポーネントとブートロードで使用する通信コンポーネントをドラッグしてください。詳細については、関連文書にリスト化されている関連する入門アプリケーションノート (Getting Started) を参照してください。図 8 に示すように、ブートローダおよびブートローダブルコンポーネントがコンポーネントカタログウィンドウのシステムタブの下で使用できます。また図では、ブートローダコンポーネントとブートロード時に使用する I2C 通信コンポーネントがブートローダプロジェクトの回路図にあることを示します。

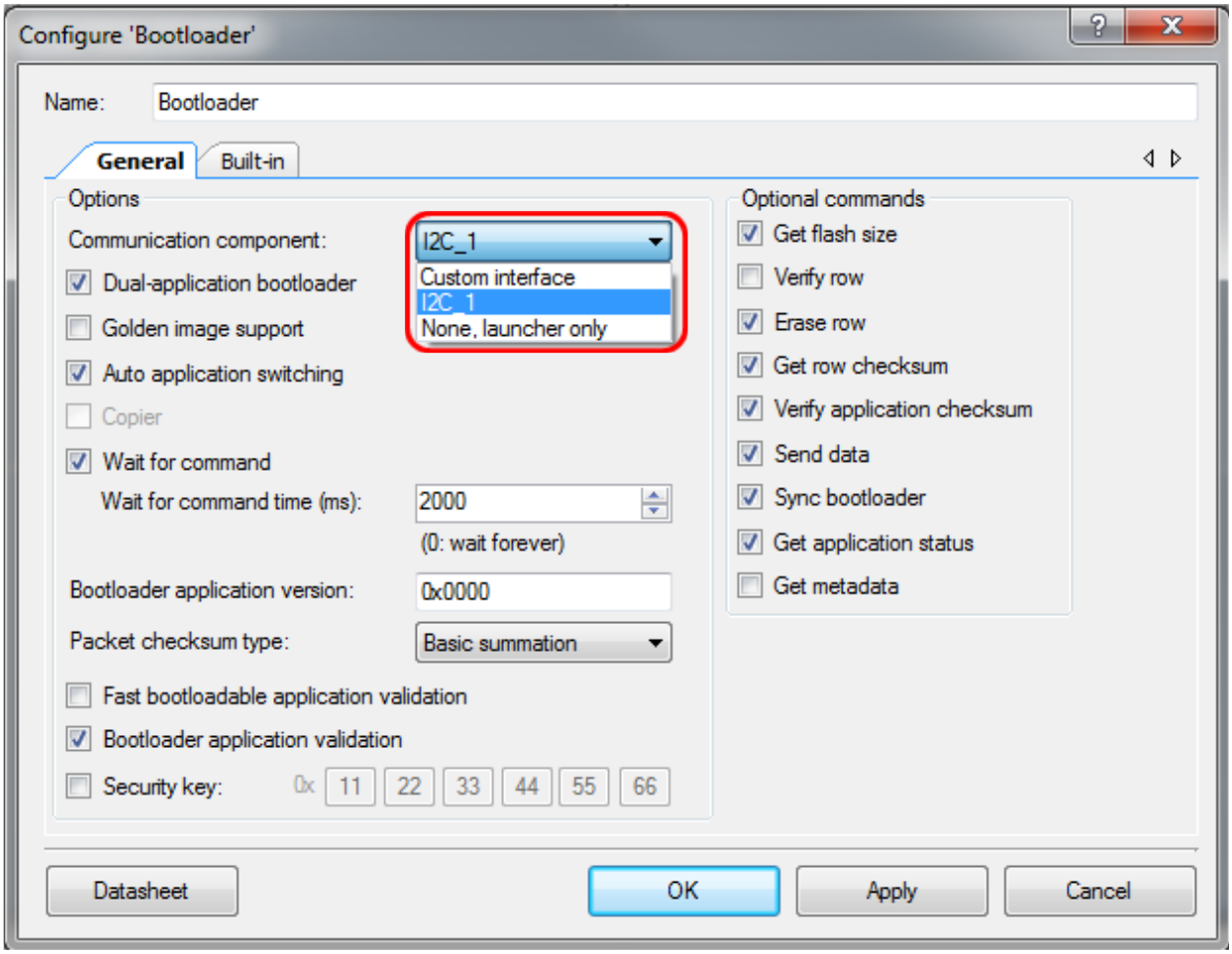
図 8. コンポーネントカタログおよび回路図ウィンドウ



次に図 9 に示すようにブートローダコンポーネントを設定してください。図 9 で通信コンポーネントを選択するメニューは、ホストとの通信で使用するコンポーネントであることに注意してください。ブートローダプロジェクトは、このコンポーネントを定義して選択する必要があります。回路図にあるすべてのブートローダと互換性のある通信コンポーネントを選択するか、または **Custom_Interface** を選択して独自に定義することもできます。また **None, Launcher Only** オプションも選択できます。これは通信コンポーネントが必要ない限定されたブートローダ機能にのみ対応します。

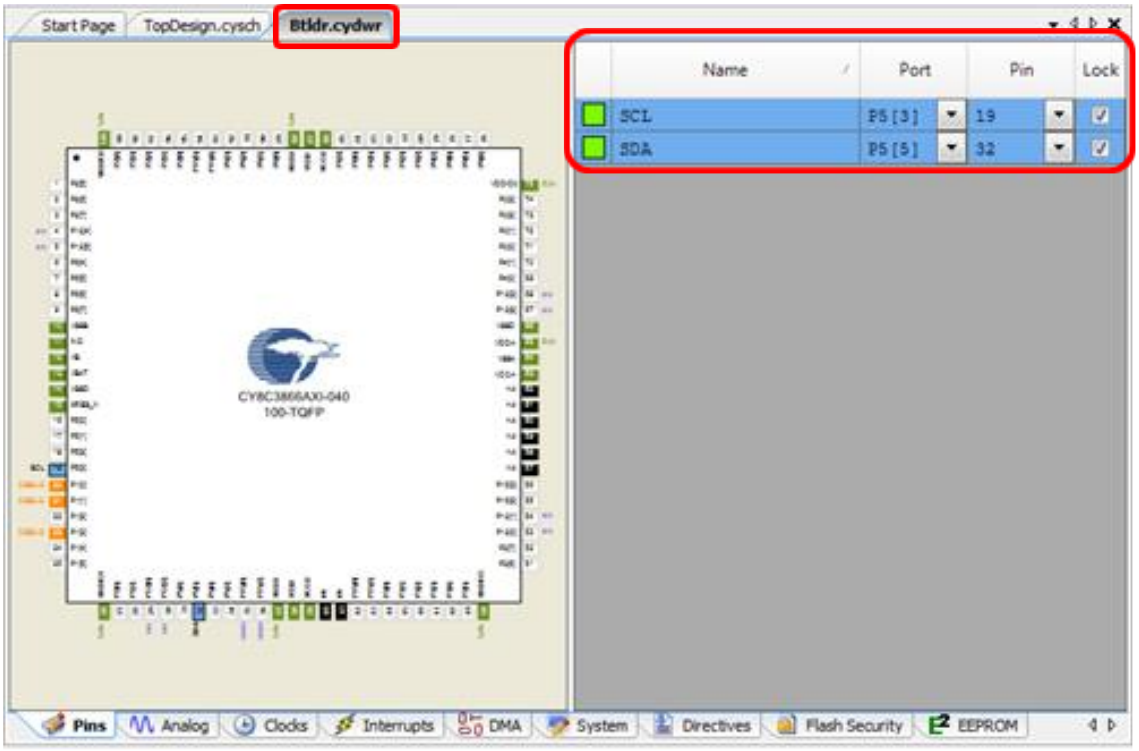
注: PSoC Creator 3.2 では **None, Launcher Only** は使用できません。

図 9.ブートローダコンポーネント設定



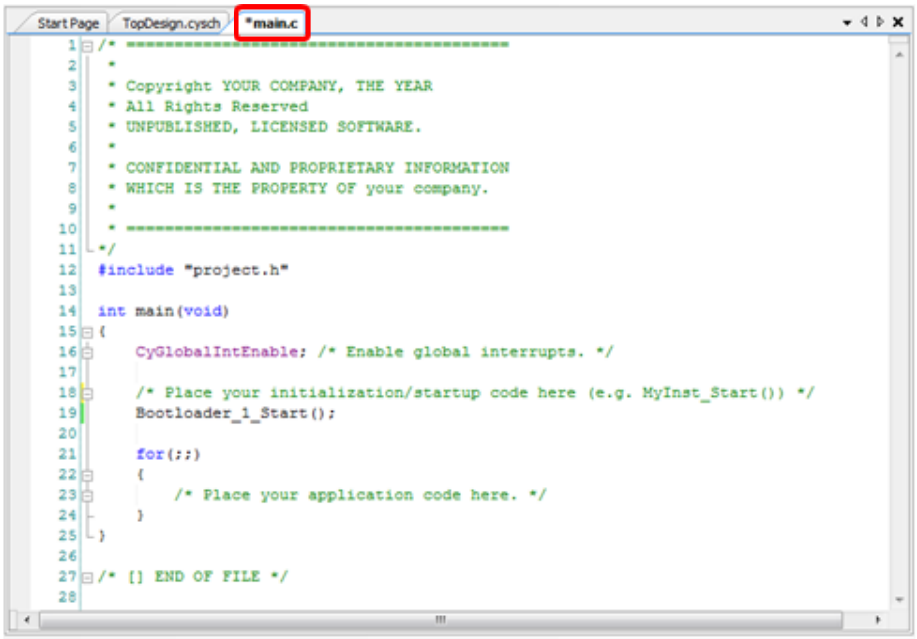
最後に、デザインワイドリソース (DWR) ウィンドウで図 10 に示すように回路図のピンを物理的なピンへ接続してプロジェクトを完了させてください。

図 10.ブートローダプロジェクトピン割り当て



プロジェクトをビルドしてください。他のすべてが行われ、その結果は基本的なブートローダプロジェクトです。*main.c* ファイルには、ブートローダの開始関数を呼び出すコードが 1 行だけ含まれます。図 11 に示すように、この関数の *main.c* への追加は手動で行う必要があります。

図 11. ブートローダ *main.c*

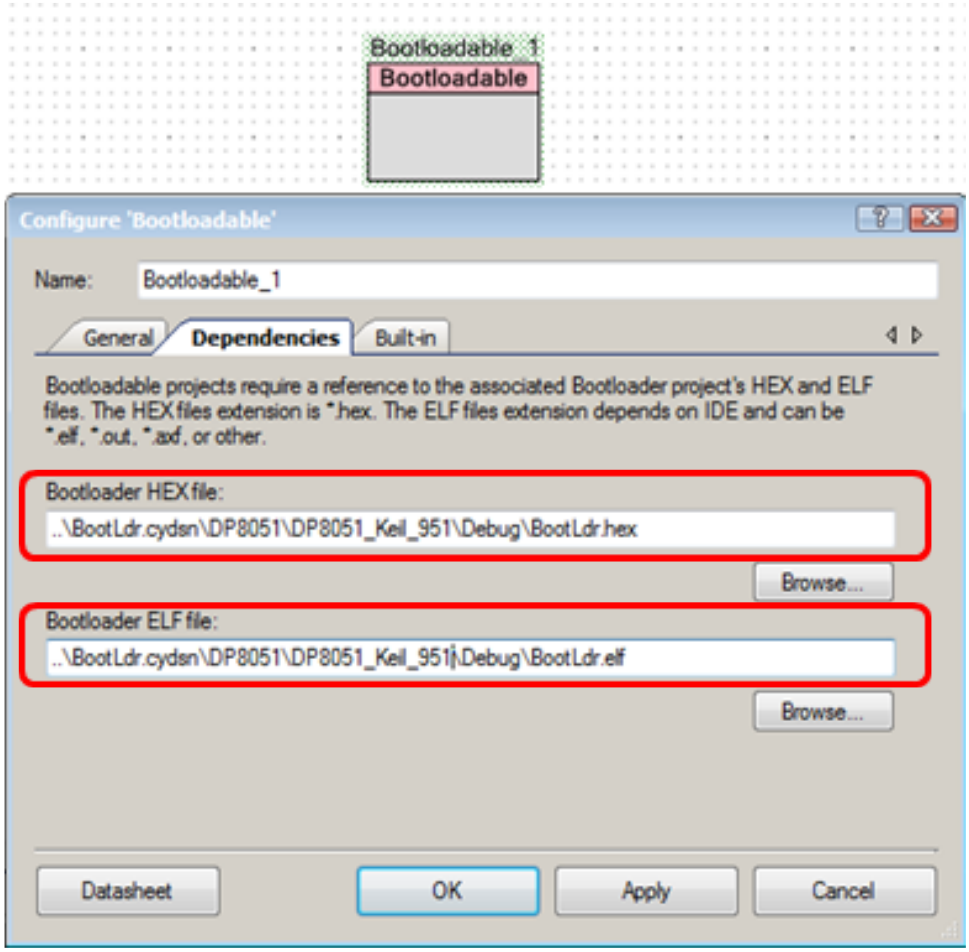


7.2 ブートローダブルアプリケーションの追加

ブートローダの作成後は、多くのブートローダブルアプリケーション、つまりプロジェクトを必要なだけ定義できます。

ブートローダブルプロジェクトは、回路図にそれ自身のブートローダブルコンポーネント (12 ページの図 8 を参照) が必要となります。プロジェクトは、図 12 に示すように、ブートローダプロジェクトにも関連付けられる必要があります。これを行うには、ブートローダブルコンポーネントの設定ダイアログでブートローダの*.hex* および*.elf* ファイルの場所を選択してください。詳細についてはプロジェクトファイルを参照してください。

図 12. ブートローダブル／ブートローダプロジェクトリンク



PSoC Creator のワークスペースは、複数のプロジェクトを持てます。多くの場合、ブートローダプロジェクトは関連付けられたブートローダブルと同じワークスペースにあります。しかし、ブートローダおよびブートローダブルをそれぞれ別のワークスペースや PC 上の別の場所にもできます。PSoC を開始する前に、全体的なシステム開発の要求に合わせてワークスペース／プロジェクトの計画を立てることを推奨します。

注: ブートローダブルプロジェクトに対するフラッシュ保護設定は無視されます。関連付けられたブートローダプロジェクトのフラッシュ保護設定が優先されます。

注: ブートローダブルコンポーネントはチェックサム除外セクションを指定するオプションがあります。このセクションは通常、ブートローダのブートローダブルチェックサム検証に影響を与えずに、変更される可能性のあるフラッシュデータ (例えば、走行距離またはデータログ) を格納するために使用します。詳細については、Bootloadable Component データシートを参照してください。

7.3 ブートローダブルプロジェクトのデバッグ

PSoC Creator のブートローダシステムでは、ブートローダプロジェクトを最初に実行し、それからブートローダブルプロジェクトを実行します。ブートローダからブートローダプロジェクトへジャンプはソフトウェア制御のデバイスリセットを通じて行われます。詳細については付録 A を参照してください。これはデバッガのインターフェースをリセットします。つまり、ブートローダブルプロジェクトはデバッガモードでは実行できません。

ブートローダブルプロジェクトをデバッグするには、ブートローダブルコンポーネントを無効にし、プロジェクトをデバッグし、それからデバッグ完了後にプロジェクトをブートローダブルに戻す変換をしてください。

他のオプションとしては、ブートローダブルプロジェクトの*.hex* ファイルをデバイス上にプログラムし、それからブートローダブルプロジェクトが実行中にデバッグのために **Attach to running target** オプションを使用することです。この場合、デバッガがデバイスにアタッチされた箇所からのみブートローダブルプロジェクトのデバッグが可能となります。

7.4 ブートローダのカスタマイズ

8 ページのカスタマイズで述べたように、ブートローダは、回路図上に追加のコンポーネントをドラッグし、*main.c* にコードを追加することでカスタマイズできます。単純な例として、図 13 と 図 14 に示すように、ブートロード中を表示する LED の点滅させるため PWM、クロック、およびピンコンポーネントを追加できます。API を開始するブートローダコンポーネントは、手動で *main.c* へ配置する必要があります。任意の周波数およびデューティサイクルで LED を点滅させるようにコンポーネントを容易に設定できます。

PSoC のブートローダプロジェクトへの設定はブートローダブルプロジェクトに制御を渡すまでの間のみ有効であることに注意が必要です。PSoC デバイスは、その後ブートローダブルプロジェクトに再設定します。もし両方のプロジェクトで同じ機能としたい場合には同じコンポーネントとコードを両方のプロジェクトへ配置してください。

図 13. ブートローダブルプロジェクトのカスタマイズ

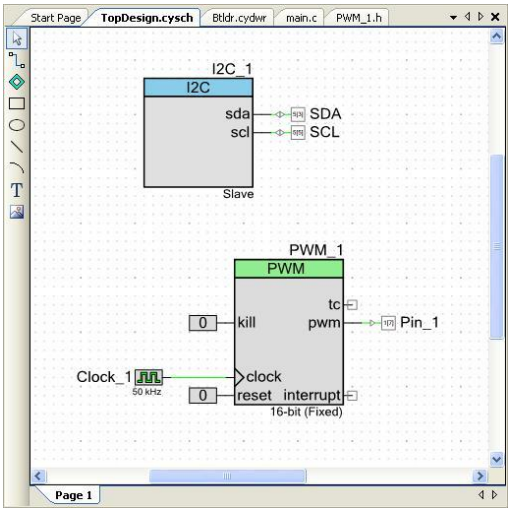
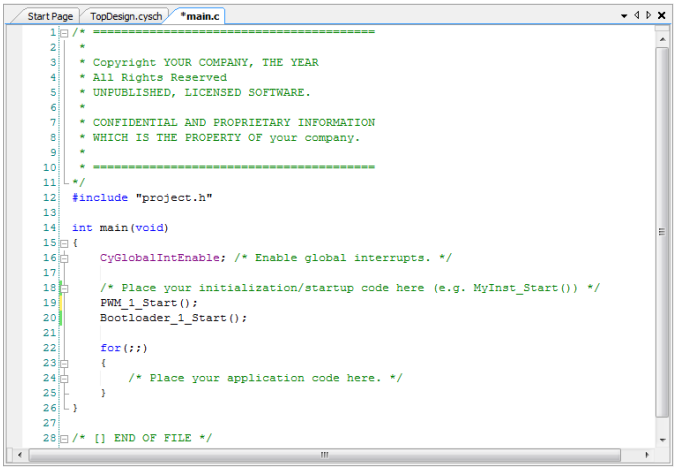


図 14. *main.c* のブートローダカスタマイズ



7.5 ブートローダの呼出し

ブートローダ - ホストタイミングで述べたように、アプリケーションからのブートローダ呼出しによるブートローダの初期化タイミングに関する問題については回避できます。次に、アプリケーションは、ボタン押下やホストからのメッセージなどの外部イベントに応答し、ブートロード動作を開始できます。

ブートローダコンポーネントは、パブリックファンクションである Bootloader_Start()という API を持ちます。ブートローダブルプロジェクトコードから、ブートロード動作を開始するために、この機能を呼び出してください。

Bootloader_Start()はデバイスのソフトウェアリセットを行い、ブートローダは CPU を引き継ぎます。リソースとペリフェラルはブートローダにより再設定され、ブートローダブルの設定は無効化されます。割込みハンドラを含むブートローダブルプロジェクトコードは、実行されなくなります。ブートロード動作が完了すると、CPU が再びリセットされます。詳細については付録 A を参照してください。

8 PSoC へのプロジェクトのロード

標準のプロジェクトと同様に、ブートローダプロジェクトはターゲットの PSoC へ PSoC Creator または PSoC プログラマを使用し、JTAG または SWD 経由のみでインストールされます。ブートローダがインストールされアクティブになった後、ブートローダブルプロジェクトはブートロード動作によって JTAG または SWD を経由せずにインストールされます。

PSoC Creator にはブートローダホストと呼ばれる PC プログラムが含みます。これは PC 側のブートロード動作を行います。CY8CKIT-002 MiniProg3 kit のようなプログラマを使用して、ターゲットの PSoC と USB ポートまたは I²C ポートを経由して直接通信します。ブートローダホストを使用するには、ブートローダおよびブートローダプロジェクト向けに生成される出力ファイルについて知る必要があります。

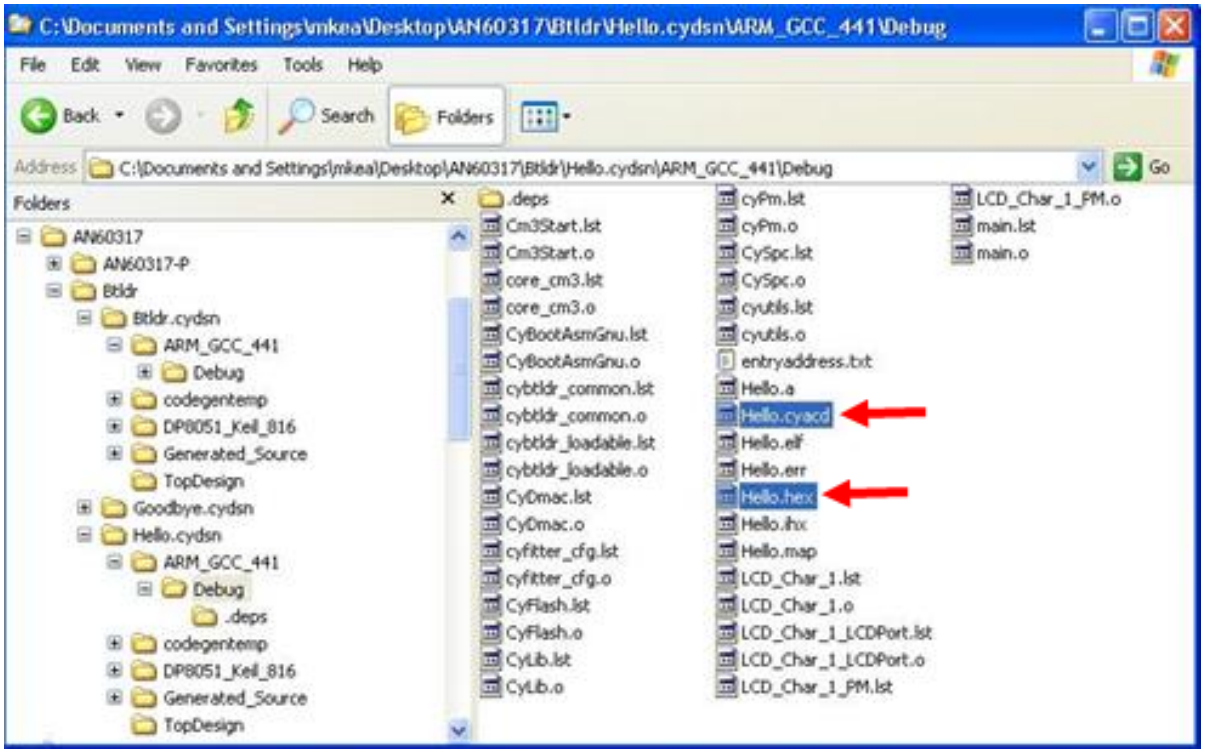
8.1 プロジェクトファイル

ビルドを実行すると、PSoC Creator プロジェクトは出力として.*hex* ファイルを生成します。これらのファイルは MiniProg3 を含むあらゆる HSSP プログラマで使用できます。Hex ファイルには CPU コードとプロジェクト設定の両方のバイトを含みます。

通常およびブートローダプロジェクト向けに、*.hex* ファイルは対象のプロジェクト用のコードとデータバイトを含みます。ブートローダブルの*.hex* ファイルは、ブートローダブル*および*関連付けられたブートローダプロジェクトのコードとデータバイトが含まれているという点で異なります。両方のプロジェクトは同時にプログラムされます。

ブートローダブルプロジェクトは、図 15 に示すように、出力として*.cyacd* タイプのセカンドファイルを生成する点でも通常のプロジェクトとは異なります。*.cyacd* ファイルはブートローダブルプロジェクト用のみのコードとデータを含み、関連付けられたブートローダのものは含みません。これはブートローダホストプログラムによって使われることを意図され、既にインストールされた関連するブートローダプロジェクトを持つターゲットの PSoC にダウンロードされます。

図 15. ブートローダブルプロジェクトファイル



8.2 ユースケース

プロジェクトがビルドされ出力ファイルが作成された後、通常は以下のシナリオの 1 つが使用されます。(5 ページの図 4 も参照)

1. ブートローダプロジェクトを作成してビルドします。MiniProg3 のような HSSP プログラマを使用してターゲットの PSoC へ.*hex* ファイルをプログラムします。
2. ターゲット PSoC をリセットし、ブートローダを開始します。ブートローダはフラッシュのプロジェクトのみであるため、ホストからのブートロードコマンドを待ち続けます。
3. ブートローダブルプロジェクトを作成し、ブートローダブルプロジェクトと関連付け、およびビルドします。ホストプログラムと事前にインストールしたブートローダを使用して.*cyacd* ファイルをターゲットにダウンロードします。

後続のブートロード動作では、フラッシュに有効なブートローダブルが存在するため、ブートローダはホストを短時間待機してから制御をブートローダに渡すことに注意してください。

工場生産のシナリオでは、代わりに以下を行えます。

4. ブートローダプロジェクトを作成してビルドします。
5. ブートローダブルプロジェクトを作成し、ブートローダプロジェクトと関連付け、およびビルドします。 *.hex* ファイル (ブートローダとブートローダブルの両方を含む) をターゲットの PSoC デバイスへ MiniProg3 のような HSSP を使用してプログラムします。
6. ターゲット PSoC デバイスをリセットし、ブートローダを開始します。ブートローダがフラッシュ内に有効なブートローダブルを見つけ、ホストからのブートロードコマンドを待つことが可能なタイムアウトの後、ブートローダブルへ制御を渡します。

注: もしブートローダコンポーネントが高速ブートローダブルアプリケーション検証 (**Fast bootloadable application validation**) のボックスにチェックされている場合、ブートローダが実行される初回の動作で、有効なブートローダブルアプリケーションをチェックします。もしブートローダが有効なブートローダブルアプリケーションを検出した場合、アプリケーションのメタデータを含むフラッシュの行を更新します。これによりフラッシュチェックサム製造試験での不一致が発生する可能性があります。製造試験プロセスに応じてレビューおよび調整を行ってください。

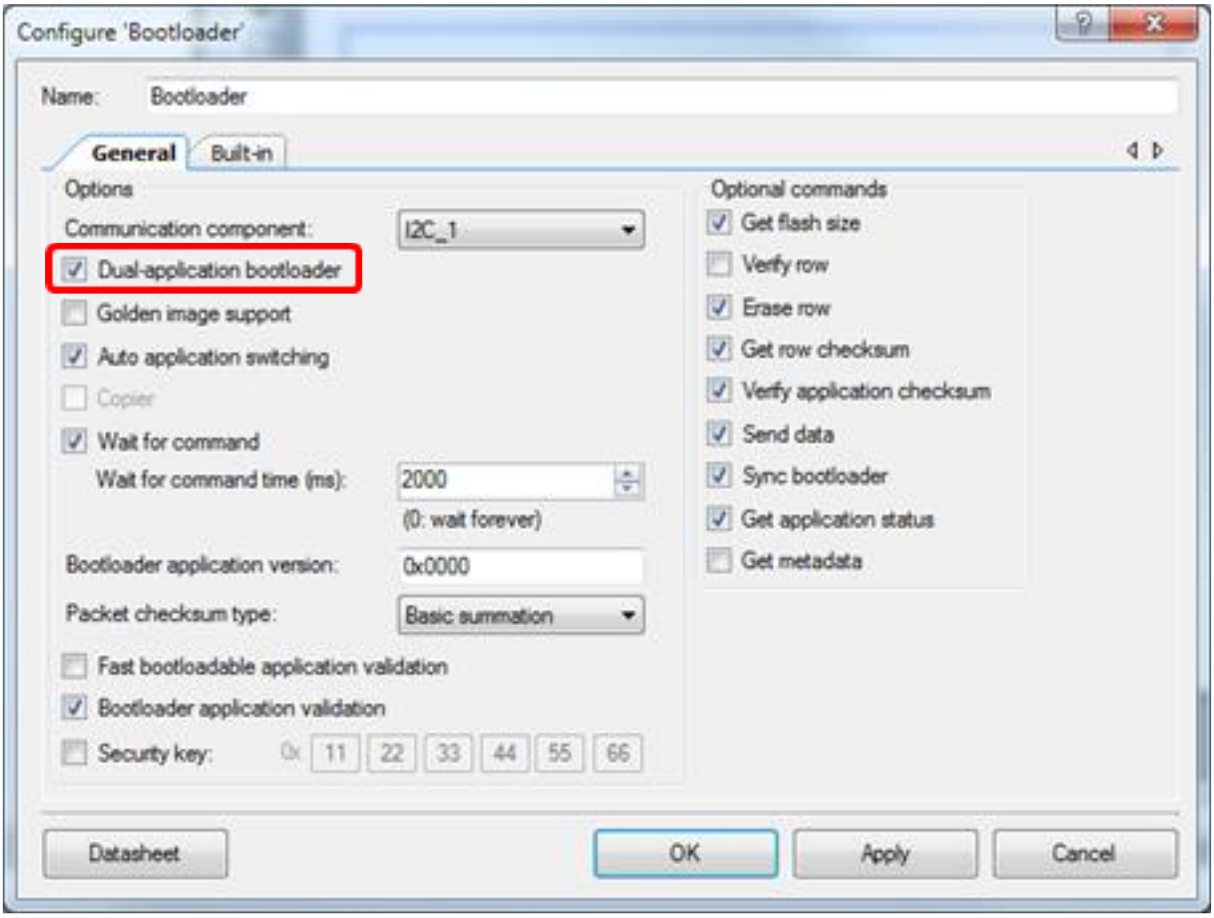
将来ブートローダブルを更新する場合は、ホストプログラムを使用してターゲットへ.*cyacd* ファイルをダウンロードできます。これは以前のバージョンのブートローダブルを上書きします。

9 デュアルアプリケーションブートローダの考慮事項

PSoC Creator のブートローダとブートローダブルコンポーネントは、カスタマイズで説明したように、高信頼性アプリケーション向けのデュアルアプリケーションイメージに対応します。デュアルアプリケーションブートローダ向けの PSoC Creator ビルドプロセスは、シングルアプリケーション向けと同様ですが、一部異なります。

1. 図 16 に示すように、ブートローダコンポーネント設定ダイアログで Dual-application bootloader のボックスをチェックしてください。PSoC Creator 3.2 以前では、このオプションは **Multi-Application bootloader** と名付けられています。

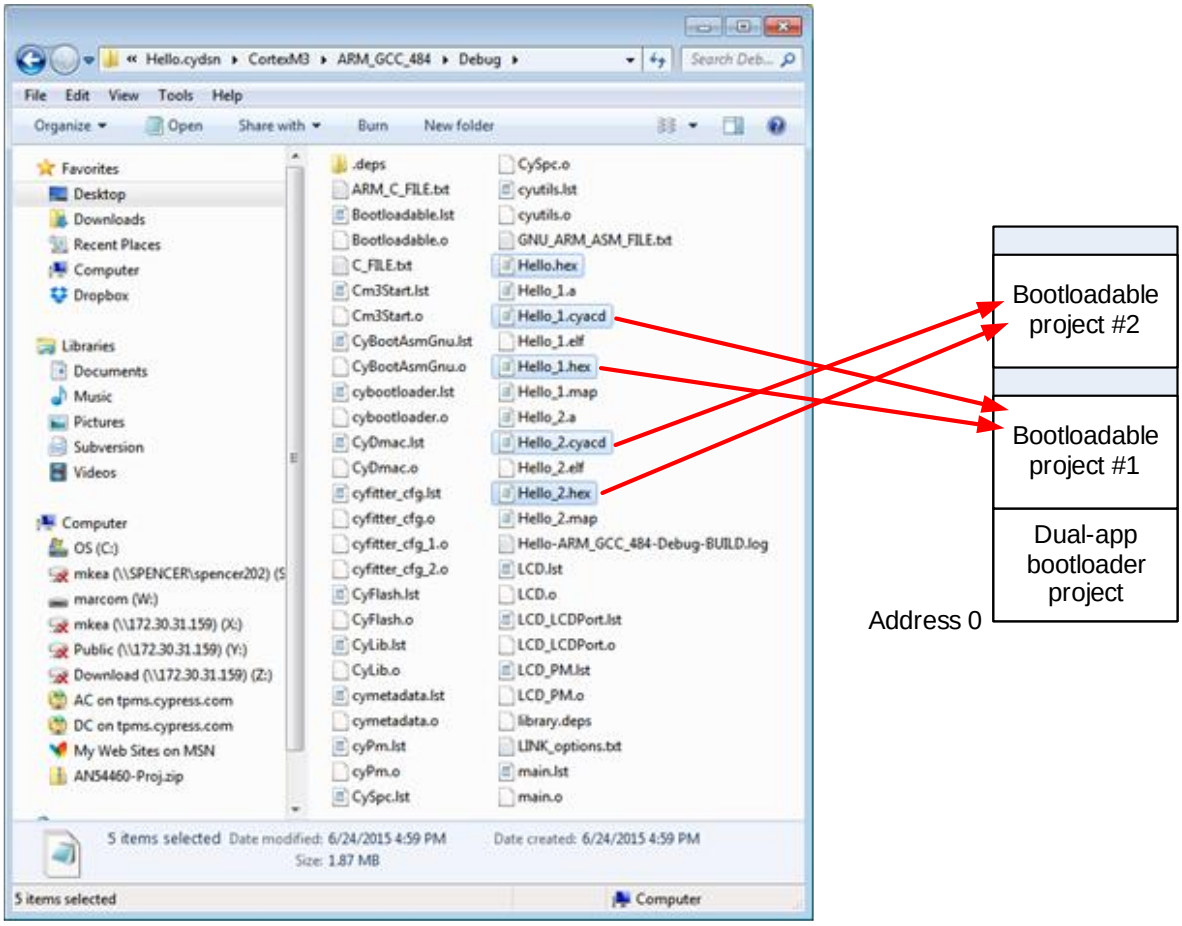
図 16. ブートローダコンポーネント設定



2. プロジェクトファイル： デュアルアプリケーションのブータブルプロジェクトは、図 17 に示すように、2 つのファイルの代わりに 5 つの出力ファイルがあります。これらのファイルは、図 17 に示すように、アプリケーション 1 または 2 のどちらにもブートローダブルプロジェクトの配置ができます。高信頼性アプリケーション向けには、同じ 2 つのブートローダブルプロジェクトのコピーをフラッシュへ配置できます。

   2 つの異なるブータブルプロジェクトも作成できます。1 つ目のアプリケーションをインストールした後、2 つ目のアプリケーションをインストールできます。

図 17.デュアルアプリケーションのブートローダブルプロジェクトファイル



事前の注意点として、通常製品生産時には、*.hex* ファイルは JTAG/SWD ポートを経由してインストールされます。*.hex* ファイルは、ブートローダブルプロジェクトの 1 つまたは両方と同様に、ブートローダプロジェクトを含みます。

フィールドでは.*cyacd* ファイルは、例えばブートローダホストプログラムとともにインストールされます。

9.1 アプリケーション起動プロセス

デュアルアプリケーションブートローダが行う必要がある決定の 1 つは、(存在する場合) どのアプリケーションを起動するか、または制御を転送するかです。各アプリケーションには、この決定を左右する 2 つの特性があります。

- アクティブ (Active)： 前述のように PSoC Creator のブートローダコンポーネントは、アプリケーションのデータ (「メタデータ」としても知られる) をフラッシュの最上位行に保存します。このデータには「アクティブ」ビットを含みます。アプリケーションの 1 つだけにアクティブビットセットがあり、それが起動に推奨されるアプリケーションです。

- 有効 (Valid) : 起動前に、ブートローダはそれぞれのアプリケーションのメタデータ内にあるデータバイトをチェックし、アプリケーションが有効かどうかテストします。テストで不合格となることもあります。例えば、フラッシュメモリの破損、またはフラッシュにアプリケーションがインストールされていなかったなどです。この場合、アプリケーションは「無効」となります。

ブートローダがどのアプリケーションを起動するのか決定する対応表を表 2 に示します。場合により、両方のアプリケーションがアクティブとなることがありますが、それは異常な状態であり、通常状態では発生しません。

表 2. アプリケーション起動対応表

	アプリケーション #1		アプリケーション#2		
ケース	Active	Valid	Active	Valid	ブートローダのアクション
0	0	0	0	0	ブートローダで待機、ホストからの応答待ち
1	0	0	0	1	#0 と同じ
2	0	0	1	0	#0 と同じ
3	0	0	1	1	アプリケーション #2 へ
4	0	1	0	0	#0 と同じ
5	0	1	0	1	#0 と同じ
6	0	1	1	0	#0 と同じ
7	0	1	1	1	アプリケーション #2 へ
8	1	0	0	0	#0 と同じ
9	1	0	0	1	#0 と同じ
10	1	0	1	0	#0 と同じ
11	1	0	1	1	アプリケーション #2 へ
12	1	1	0	0	アプリケーション #1 へ
13	1	1	0	1	アプリケーション #1 へ
14	1	1	1	0	アプリケーション #1 へ
15	1	1	1	1	アプリケーション #1 へ

10 まとめ

このアプリケーションノートはブートローダの基本概要として使用方法と重要な設計上の考慮事項を提供しました。PSoC 3、PSoC 4、および PSoC 5LP デバイスに対する PSoC Creator 設計環境での考慮事項への対応方法についても説明しました。

PSoC Creator を使用してブートローダをすばやく簡単に設計へ追加する方法の基本概要もお伝えしました。これらのトピックについて、より詳細にカバーするアプリケーションノートについては、関連文書を参照してください。

11 関連文書

PSoC 3、PSoC 4、および PSoC 5LP の包括的なリソースのリストはサイプレスコミュニティの KBA86521 を参照してください。

ブートローダアプリケーションノート	
AN60317 – PSoC 3 and PSoC 5LP I²C Bootloader	PSoC 3 および PSoC 5LP 向けの I²C ベースのブートローダの説明
AN86526 – PSoC 4 I²C ブートローダ	PSoC 4 向けの I²C ベースのブートローダの説明
AN73503 – PSoC 3 and PSoC 5LP USB HID Bootloader	PSoC 3 および PSoC 5LP 向け USB HID ベースのブートローダの説明
AN68272 – PSoC 3, PSoC 4 and PSoC 5LP UART Bootloader	PSoC 3, PSoC 4 および PSoC 5LP 向け UART ベースのブートローダの説明
AN84401 – PSoC 3 and PSoC 5LP SPI Bootloader	PSoC 3 および PSoC 5LP 向けの SPI ベースのブートローダの説明
その他の関連アプリケーションノート	
AN213924 – PSoC 6 MCU デバイス ファームウェア アップデート ソフトウェア開発キット ガイド	デバイスファームウェアアップデート (DFU) ソフトウェア開発キット (SDK) での PSoC 6MCU 向けの DFU システム開発に関する包括的な情報の提供
AN73054 – PSoC 3 and PSoC 5LP Programming Using an External Microcontroller (HSSP)	PSoC 3 または PSoC 5LP デバイスへのモジュラーC コードを使用した外部 MCU によるプログラムの実装方法の説明
AN84858 – 外部マイクロコントローラーを使用した PSoC 4 プログラミング (HSSP)	PSoC 4 デバイスへのモジュラーC コードを使用した外部 MCU によるプログラムの実装方法の説明
AN61290 – PSoC 3 and PSoC 5LP Hardware Design Considerations	PSoC 3 または PSoC 5LP デバイスのハードウェアシステムの設計に関するトピックをレビュー
AN88619 – PSoC 4 Hardware Design Considerations	PSoC 4 デバイスのハードウェアシステムの設計に関するトピックをレビュー
AN54181 – PSoC® 3 入門	PSoC 3 デバイスとはじめての PSoC Creator プロジェクトのビルドの説明
AN79953 – PSoC 4 入門	PSoC 4 デバイスとはじめての PSoC Creator プロジェクトのビルドの説明
AN77759 – PSoC 5LP 入門	PSoC 5LP デバイス,はじめての PSoC Creator プロジェクトのビルドの説明
AN2100 – Bootloader: PSoC 1	PSoC 1 向けの UART ベースのブートローダの説明
PSoC Creator コンポーネントデータシート	
Bootloader and Bootloadable	ブートローディング機能の実装
デバイスドキュメント	
PSoC 3 Datasheets	PSoC 3 アーキテクチャ テクニカルリファレンスマニュアル
PSoC 4 Datasheets	PSoC 4 アーキテクチャ テクニカルリファレンスマニュアル
PSoC 5LP Datasheets	PSoC 5LP アーキテクチャ テクニカルリファレンスマニュアル
開発キットドキュメント	
PSoC 3 Kits PSoC 4 Kits PSoC 5LP Kits	
ツールドキュメント	
PSoC Creator	Downloads タブからクイックスタートとユーザーズガイドが参照可能

付録 A. ブートローダおよびデバイスリセット

このアプリケーションノートの他の箇所で述べたように、ブートローダからブートローダブルへ、またはその逆に制御を渡す際には、常にデバイスリセットを通じて行われます。もしシステムがあるプログラムから別のプログラムへ変更しながら極めて重要な機能を継続しなければならない場合には、これを考慮することが必要です。ここでは、リセットを使用しなければならない理由と、アプリケーションでのデバイスパフォーマンスへの影響について詳しく説明します。

A.1 なぜデバイスリセットが必要か？

デバイスリセットがなぜ必要か理解するには、ブートローダとブートローダブルプロジェクトがシステム内でそれぞれ完全に自己完結型の PSoC Creator プロジェクトであるということへの注意が重要です。それぞれのプロジェクトはそれぞれにデバイスコンフィグレーションの設定があります。つまり、あるプロジェクトから別のプロジェクトへ変えるときには、PSoC デバイスのハードウェア機能を完全に再定義できます。

複雑なカスタム機能を実装するにはデバイス設定に数千の PSoC レジスタの設定が行われます。PSoC のデジタルおよびアナログルーティング機能に対しては特に当てはまります。レジスタ設定とルーティングを設定するときは、新しい設定のビット設定に加えて古い設定のビットがリセットされたことの確認が必要となります。そうでない場合、新しい設定が動作しなかったり、デバイスにダメージをおよぼす可能性もあります。

ブートローダとブートローダブルプロジェクトが変更される間にデバイスソフトウェアリセット (SRES) は完了します。すべての PSoC レジスタに影響しデフォルト状態にリセットします。新しいプロジェクト向けの設定が、それから開始できます。すべての PSoC レジスタがデバイスリセットのデフォルト状態に初期化される想定により、設定時間とフラッシュメモリの使用量の両方が削減されることに注意してください

A.2 デバイス I/O ピンへの影響

アプリケーションノート AN61290, PSoC 3 and PSoC 5LP Hardware Design Considerations、および AN60616, PSoC 3 and PSoC 5LP Startup Procedure で説明されているように、リセットおよびスタートアッププロセスの間、PSoC I/O ピンは表 3 に示すように 3 つの個別の駆動モードになります。

表 3.デバイスリセット中の PSoC I/O ピン駆動モード

スタートアップイベント	I/O ピン駆動モード	期間 (Typical)		コメント
		スロー IMO (12MHz)	ファスト IMO (48MHz)	
デバイスリセット (SRES) アクティブ デバイスリセットが完了	HI-Z アナログ	40 µs		リセットがアクティブな場合、I/O は HI-Z アナログモードになります。
不揮発性ラッチ(NVLs) I/O ポートへのコピー コードの実行開始	NVL 設定: HI-Z アナログ、 Pull-up、または Pull-down	~12 ms	~4 ms	期間はコードの実行速度と設定の複雑さに依存します。
I/O ポートとピンの設定	PSoC Creator プロジェクト設定	N/A		8 つの駆動モードの可能性があります。詳細はデータシートを参照。
main()にコードが達したとき	コードは I/O ピン機能を変更	N/A		

PSoC での NVL 使用の詳細については、デバイスのデータシートを参照してください。PSoC Creator プロジェクトでの NVL 設定は 2 つの場所で確立されています。

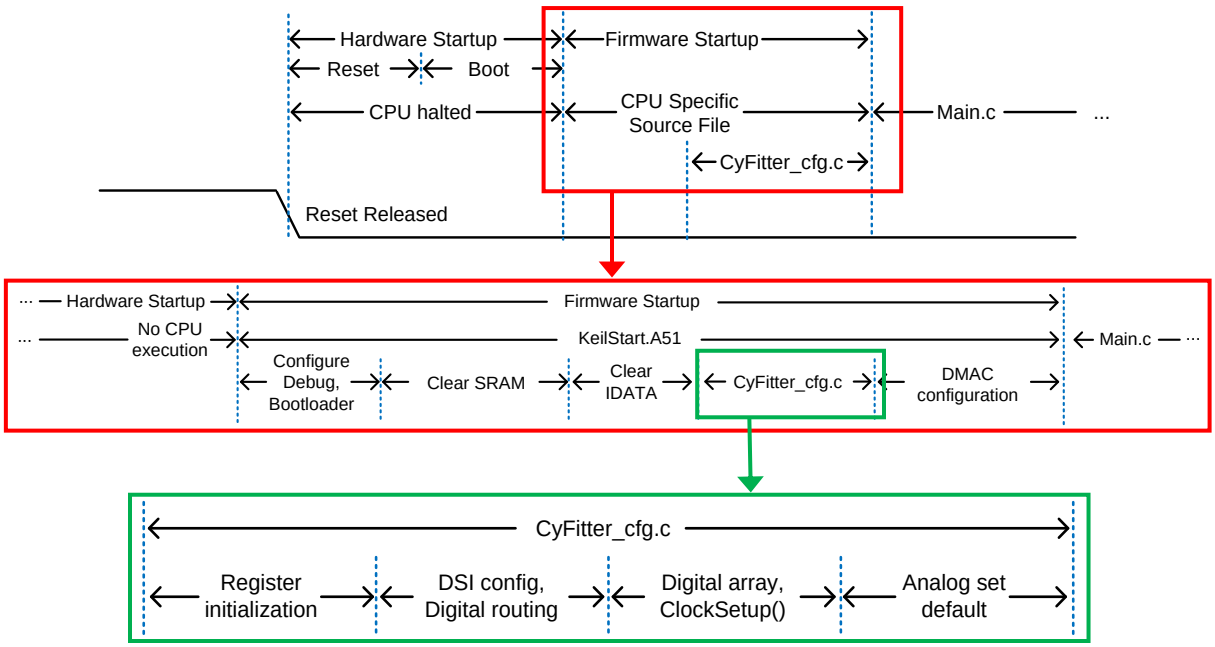
- I/O ポート: 個々のピンコンポーネント設定の **Reset** タブ
- その他すべての NVL: 設計ワイドリソース (DWR) ウィンドウの **System** タブ

NVL はプロジェクトがプログラムされたときに更新されます。ブートローダブルプロジェクトは NVL 設定ができないことに注意してください。DWR 設定は関連付けられたブートローダプロジェクトの NVL と一致している必要があります。

最終的な I/O 駆動モードは、個々のピンコンポーネント設定によって設定されます。

図 18 にデバイススタートアップと設定のタイミング図を示します。図の中段の例は、PSoC 3 のものです。PSoC 4、および PSoC 5LP にも同様のプロセスがあります。詳細については、AN60616 – PSoC 3 and PSoC 5LP Startup Procedure を参照してください。

図 18. デバイススタートアッププロセス図



A.3 他の機能への影響

デバイスリセット時、UDB レジスタはリセットされ、すべての UDB ベースのコンポーネントがなくなり、機能が停止します。設定可能なアナログコンポーネントも同様です。すべての固定されたデジタルおよびアナログのペリフェラルはリセットされ、アイドル状態になります。これは DMA、DFB、タイマ (TCPWM)、I²C、USB、CAN、ADC、DAC、コンパレータ、およびオペアンプを含みます。IMO を除くすべてのクロックは停止します。

すべてのデジタルおよびアナログのルーティング制御レジスタはリセットされます。これによりすべてのデジタルおよびアナログのスイッチはオープンになり、すべてのデバイスとの接続は切断されます。これは NVL を除くすべての I/O への接続を含みます。すべてのハードウェアベースの機能は、設定後にリストアされます (図 18 を参照してください)。すべてのファームウェア機能は、プロジェクトの main()機能の実行が開始されたときにリストアされます。

A.4 例:ファン制御

ブートローダとそれに関連するデバイスリセットをファン制御のような一般的なアプリケーションにどのように統合できるかを紹介します。PSoC Creator はファンコントローラコンポーネントを提供します。これは PWM、タコメータ入力キャプチャタイマー、制御レジスタ、ステータスレジスタ、および DMA チャネルまたは割込みを含む必要なすべてのハードウェアブロックをカプセル化します。詳細については Fan Controller Application page を参照してください。

ファン制御アプリケーションは、ブートローダブルプロジェクトにあります。必要に応じてブートローダは、ブートローディング中にファンを回し続けられるようにカスタマイズできます。表 4 に示すように、ファンはデバイスリセット中も同様に回し続けられ、ブートローダからブートローダブルへ転送する間も回し続けられます。

表 4. ファンコントローラ向けデバイスリセット中の PSoC I/O ピン駆動モード

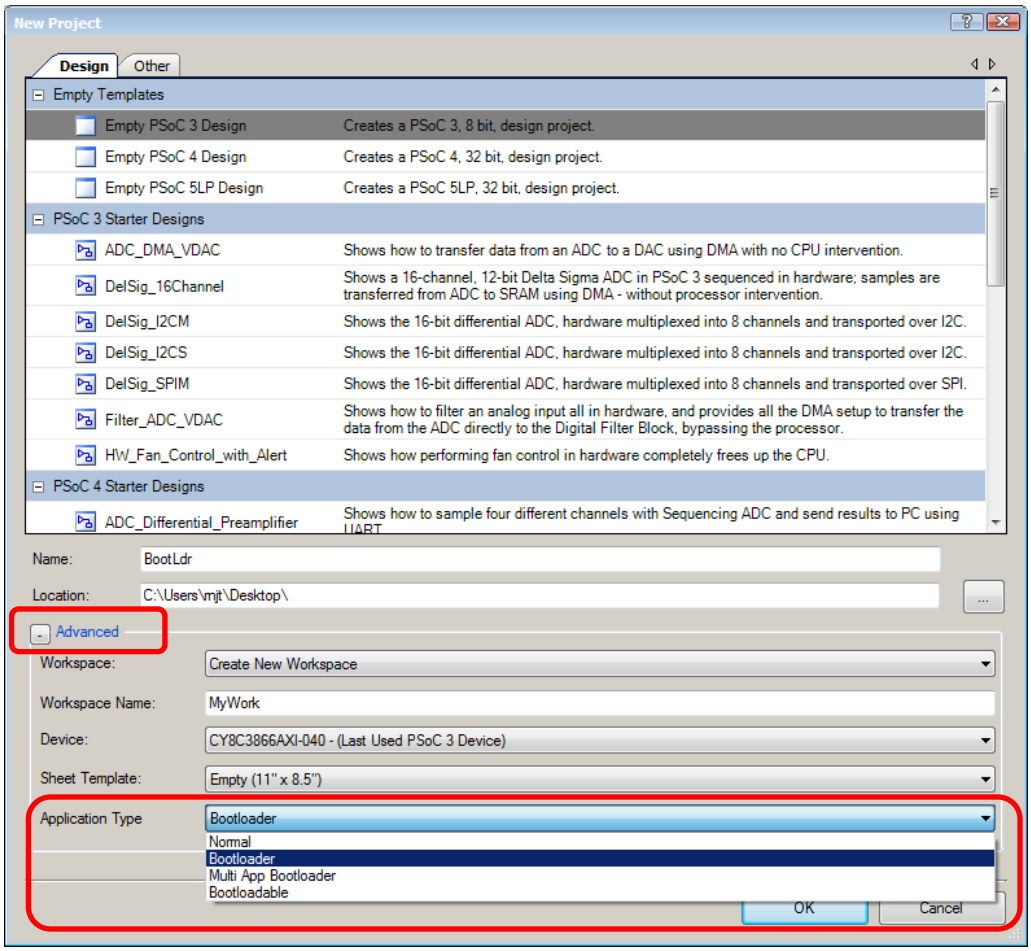
I/O ピン駆動モード	コメント
HI-Z アナログ	オプションとして外部プルアップまたはプルダウン抵抗を PWM へ 100%デューティサイクルとして追加できます。慣性によってファンが回り続けることがあるため必要ない可能性もあります。
NVL 設定: HI-Z アナログ, Pull-up, または Pull-down	オプションとして、PWM ピンコンポーネントのリセット値を 100%デューティサイクルのプルアップまたはプルダウンに設定できます。慣性によってファンが回り続けることがあるため必要ない可能性もあります。
PSoC Creator プロジェクコンフィグレーション	PWM ピンコンポーネントのドライブモードを初期設定の 100%デューティサイクルに設定します。 PWM コンポーネントはアクティブになりますが動作はしません。
main() 処理開始	.PWM_Strat()がコールされると PWM は PWM ピンのコンポーネントのデフォルトのデューティサイクルで駆動します。ファームウェアタコメータのデータを読出しデューティサイクルのアクティブな制御を開始できます。

付録 B. PSoC Creator 3.1 以前のブートローダ

B.1 ブートローダのビルド

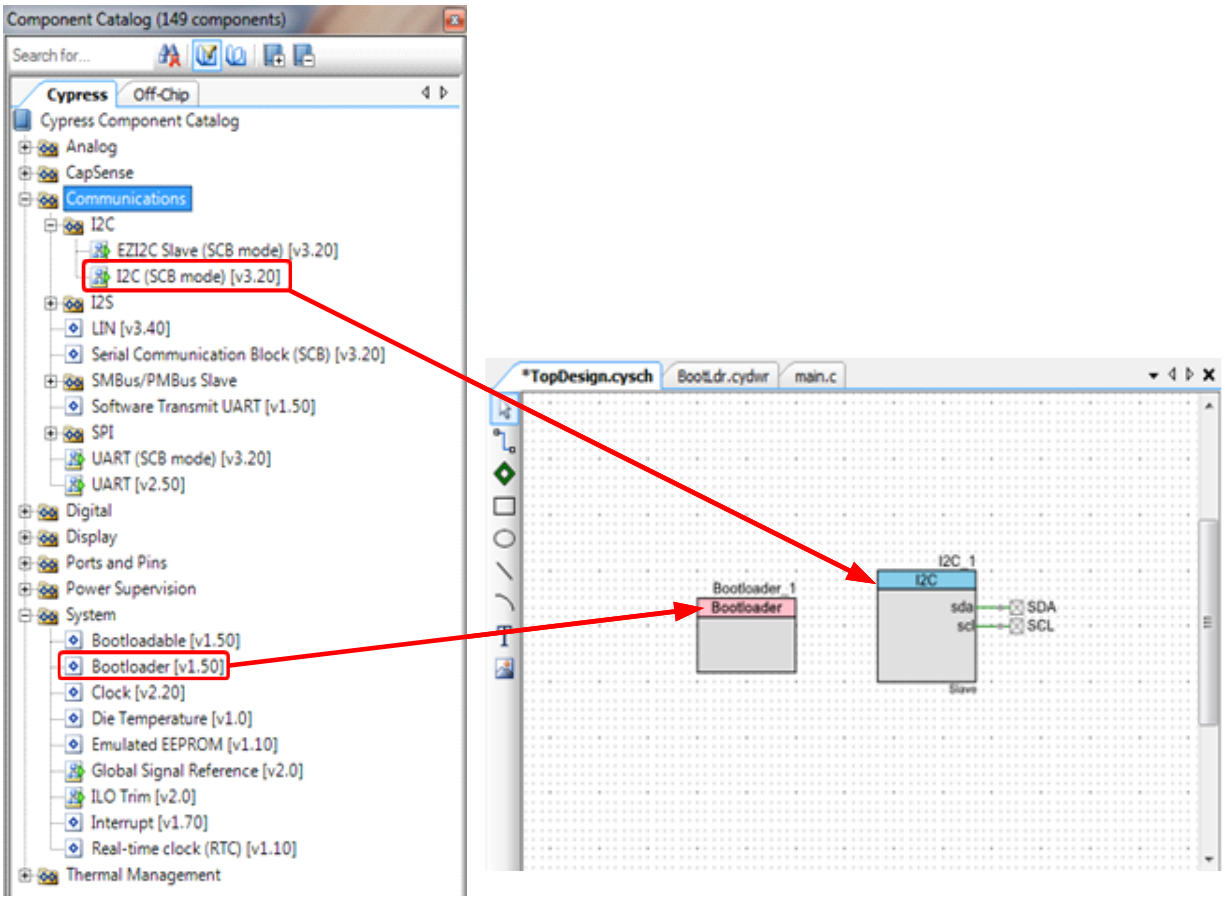
ブートローダプロジェクトを作成するには、図 19 に示すように、タイプが **Bootloader** または **Multi-App Bootloader** のプロジェクトを作成してください。この例では、プロジェクト名“BootLdr” はワークスペース名“MyWork” と異なることに注意してください。PSoC Creator のワークスペースは異なるタイプの複数のプロジェクトを含められます。

図 19. PSoC Creator 3.1 以前でのブートローダプロジェクトの作成



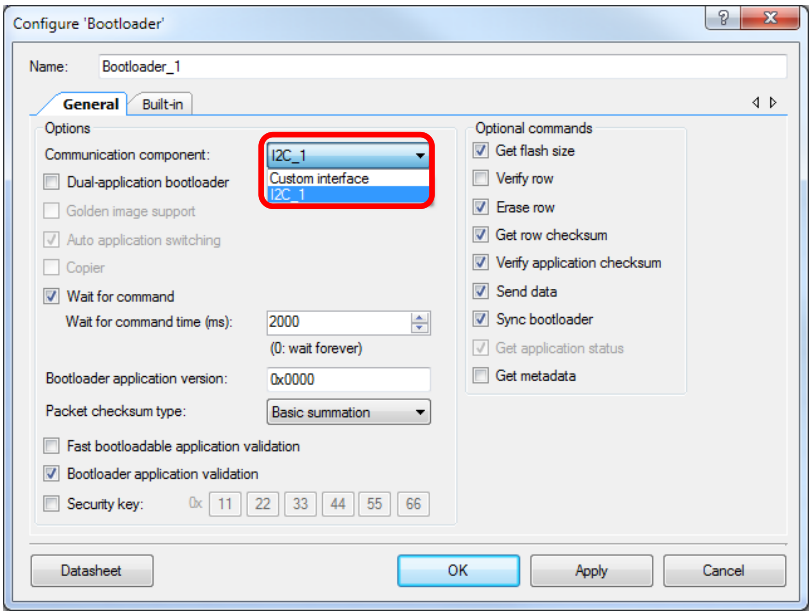
プロジェクトを作成したら、図 20 に示すように、ブートローダコンポーネントとブートロードで使用する通信コンポーネントをプロジェクトの回路図にドラッグしてください。

図 20. コンポーネントカタログと回路図ウィンドウ



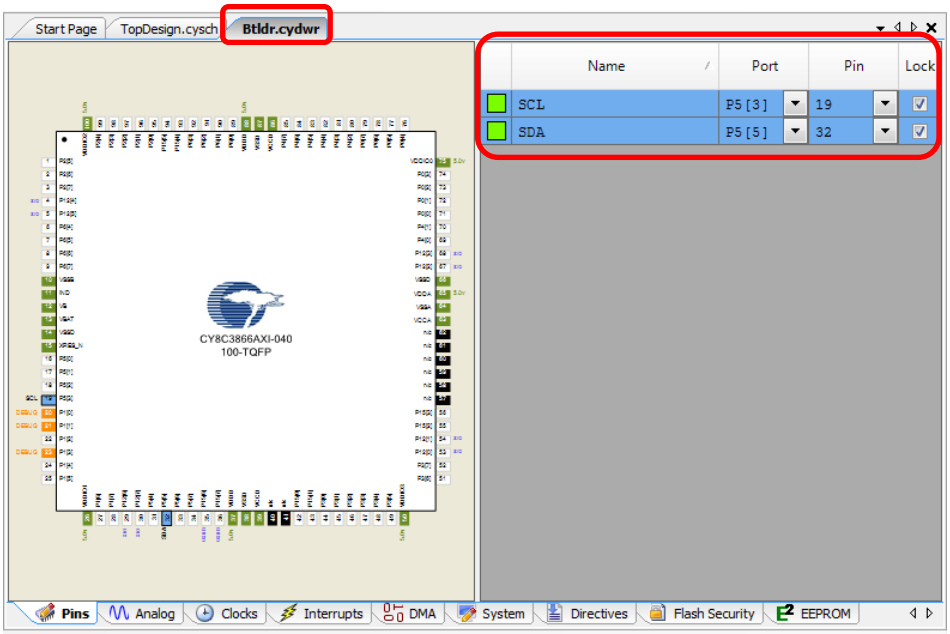
次に、図 21 に示すように、ブートローダコンポーネントを設定してください。図 21 の通信コンポーネントを選択するメニューに注意してください。これは、ホストとの通信に使用するコンポーネントです。ブートローダプロジェクトは、このコンポーネントを定義し、選択する必要があります。回路図にあるすべてのブートローダ互換通信コンポーネントから選択するか、または **Custome_Interface** を選択して自身で定義できます。

図 21. ブートローダコンポーネント設定



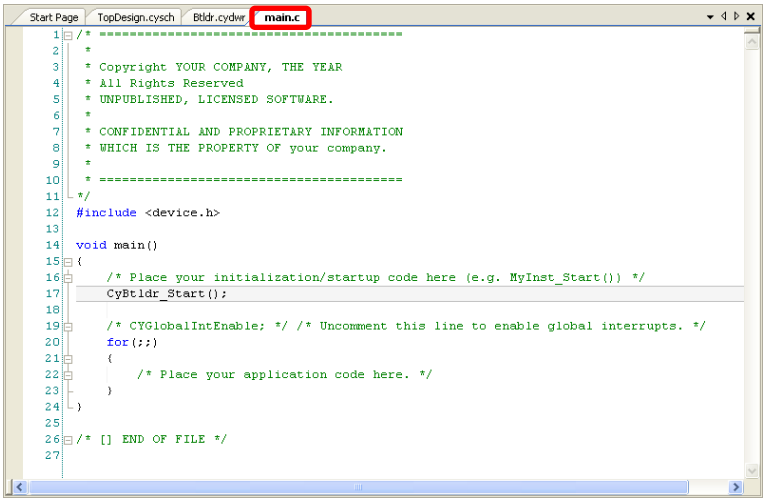
最後に、設計ワイドリソース (DWR) ウィンドウで、図 22 に示すように回路図のピンを物理的なピンへ接続してプロジェクトを完了させてください。

図 22. ブートローダプロジェクトピン割当て



プロジェクトをビルドしてください。他のすべてが行われ、その結果は基本的なブートローダプロジェクトです。*main.c* ファイルには、図 23 に示すように、ブートローダの開始関数を呼び出すコードが 1 行だけ含まれます。

図 23. ブートローダのデフォルト main.c

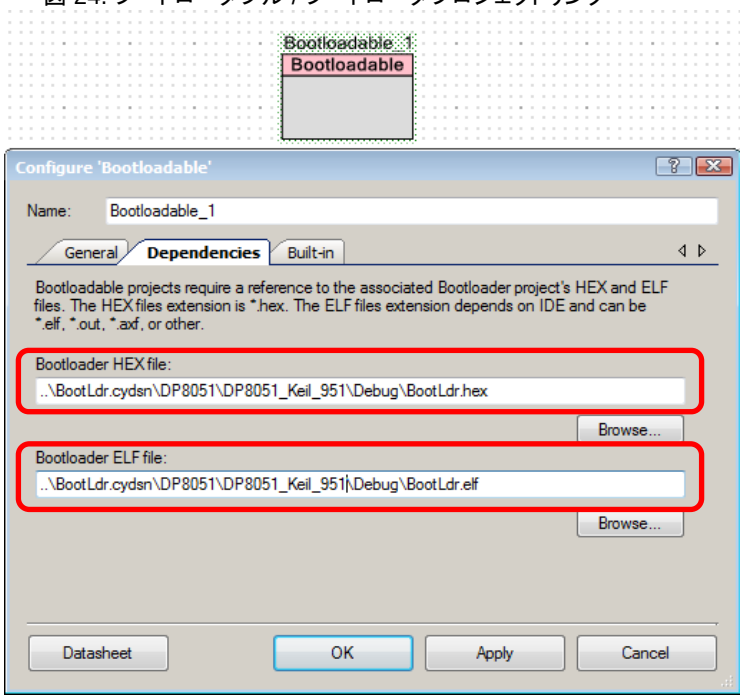


B.2 ブートローダブルアプリケーションの追加

ブートローダの作成後、28 ページの図 19 に示すように **Bootloadable** オプションを使用して、必要なだけ多くのブートローダアプリケーション、つまりプロジェクトを定義できます。すでにある **Normal** プロジェクトもタイプを **Bootloadable** に変更できます。詳細については 31 ページを参照してください。

ブートローダブルプロジェクトは回路図にブートローダブルコンポーネントを持つ必要があります (32 ページの 図 25 参照)。また、図 24 に示すように、プロジェクトはブートローダプロジェクトに関連付ける必要もあります。これを行うため、ブートローダブルコンポーネント設定ダイアログで、ブートローダの*.hex* および*.elf* ファイルの場所を選択します。詳細についてプロジェクトファイルを参照してください。

図 24. ブートローダブル / ブートローダプロジェクトリンク



PSoC Creator のワークスペースは、複数のプロジェクトを持てます。多くの場合、ブートローダプロジェクトは関連付けられたブートローダブルと同じワークスペースにあります。しかし、ブートローダおよびブートローダブルをそれぞれ別のワークスペースや PC 上の別の場所にもできます。PSoC を開始する前に、全体的なシステム開発の要求に合わせてワークスペース／プロジェクトの計画を立てることを推奨します。

注: ブートローダブルプロジェクトに対するフラッシュ保護設定は無視されます。関連付けられたブートローダプロジェクトのフラッシュ保護設定が優先されます。

注: もしブートローダが更新された場合、そのブートローダプロジェクトに依存するすべてのブートローダブルプロジェクトも再ビルドする必要があります。"Clean and Build" オプションを使用してください。

B.3 ブートローダブルプロジェクトのデバッグ

PSoC Creator のブートローダシステムでは、ブートローダプロジェクトは最初に実行され、次にブートローダブルプロジェクトが実行されます。ブートローダからブートローダブルプロジェクトへジャンプは、ソフトウェア制御のデバイスリセットを通して行われます。詳細については付録 A を参照してください。このリセットはデバッガインタフェースであり、ブートローダブルプロジェクトはデバッガモードで実行できないことを意味します。

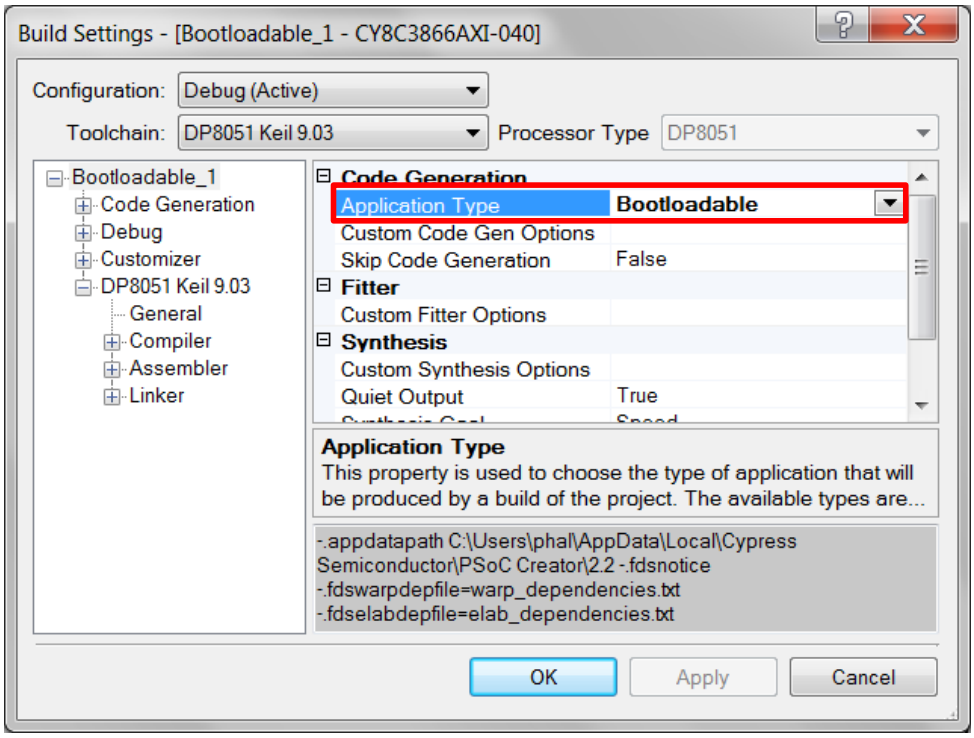
ブートローダブルプロジェクトをデバッグするには、アプリケーションタイプを Normal (図 25) に変換し、デバッグを行い、そしてデバッグが完了したらブートローダブルに戻す変換をしてください。

他のオプションとしては、ブートローダブルプロジェクトの*.hex* ファイルをデバイス上にプログラムし、それからブートローダブルプロジェクトが実行中にデバッグのために **Attach to running target** オプションを使用することです。この場合、デバッガがデバイスにアタッチされた箇所からのみブートローダブルプロジェクトのデバッグが可能となります。

B.3.1 Normal アプリケーションプロジェクトからブートローダブルプロジェクトへの変更

既に作成済みの標準 (Normal) プロジェクトをブートローダブルに変換する場合、図 25 に示すように、プロジェクトの **Application Type** をブートローダブルに変更します。

図 25. ブートローダブルへのアプリケーションタイプの変更



アプリケーションタイプを変更した後、図 24 で示すように、必ずブートローダブルコンポーネントをプロジェクトの回路図に追加し、ブートローダプロジェクトの*.hex* ファイルを依存関係として追加する必要があります。

B.4 ブートローダのカスタマイズ

8 ページのカスタマイズで述べたように、ブートローダは、回路図上に追加のコンポーネントをドラッグし、*main.c* にコードを追加することでカスタマイズできます。単純な例として、図 26 と図 27 に示すようにブートロード中を表示する LED を点滅させるため PWM、クロック、およびピンコンポーネントを追加できます。*main.c* のこの API は自動的に生成されます。任意の周波数およびデューティサイクルで LED を点滅させるようにコンポーネントを容易に設定できます。

PSoC のブートローダプロジェクトへの設定ははブートローダブルプロジェクトに制御を渡すまでの間のみ有効であることに注意が必要です。PSoC デバイスはその後ブートローダブルプロジェクトに再設定します。もし両方のプロジェクトで同じ機能としたい場合には同じコンポーネントとコードを両方のプロジェクトへ配置してください。

図 26. ブーローダプロジェクトのカスタマイズ

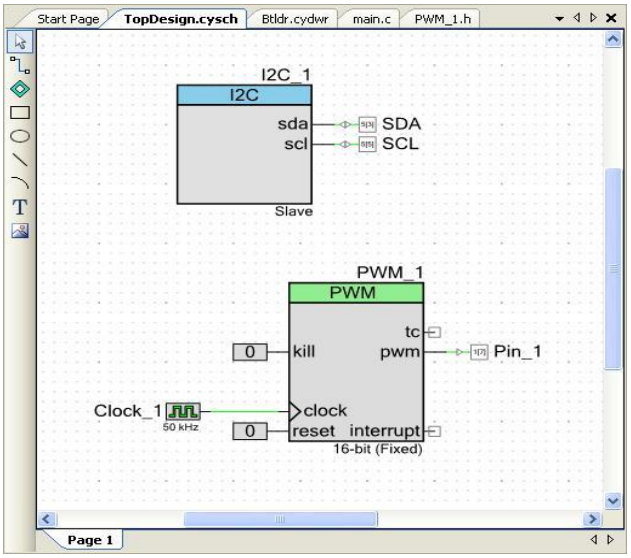


図 27.main.c のブートローダカスタマイズ

```
Start Page | TopDesign.cysch | Btldr.cydwr | main.c
 1 /* ========================================
 2  *
 3  * Copyright YOUR COMPANY, THE YEAR
 4  * All Rights Reserved
 5  * UNPUBLISHED, LICENSED SOFTWARE.
 6  *
 7  * CONFIDENTIAL AND PROPRIETARY INFORMATION
 8  * WHICH IS THE PROPERTY OF your company.
 9  *
10  * ========================================
11 */
12 #include <device.h>
13
14 void main()
15 {
16     /* Place your initialization/startup code here (e.g. MyInst_Start()) */
17     PWM_1_Start();
18     CyBtldr_Start();
19
20     /* CYGlobalIntEnable; */ /* Uncomment this line to enable global interrupts. */
21     for(;;)
22     {
23         /* Place your application code here. */
24     }
25 }
26
27 /* [] END OF FILE */
28
```

ブートローダプロジェクトを作成する詳細な手順については、ブートローダの呼出しセクションに戻ってください。

改版履歴

文書名: AN73854 - PSoC - ブートローダの導入

文書番号:002-XXXXX

版	ECN	発行日	変更内容
**		06/23/2020	本版は英語版 001-73854 Rev. *L について、CYPRESS DEVELOPPER COMMUNITY の参画者によって日本語に翻訳されたドキュメントです。

ワールドワイドな販売と設計サポート

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

製品

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

PSoC®ソリューション

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

サイプレス開発者コミュニティ

コミュニティ | サンプルコード | Projects | ビデオ | ブログ | トレーニング | Components

テクニカルサポート

cypress.com/support

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

Cypress Semiconductor
AN Infineon Technologies Company
198 Champion Court
San Jose, CA 95134-1709