PSoC™ 4 I2C ブートローダ

About this document

Scope and purpose

AN86526 は、PSoC™ 4 の I²C ベースのブートローダについて説明します。本アプリケーション ノートは、PSoC™ Creator を使用して I²C ベースのブートローダ プロジェクトとブートローダブル プロジェクトを素早く簡単に構築する方法について説明します。また、I²C ベースの組み込みブートローダ ホスト プログラムを構築する方法も示しています。

関連製品ファミリ: PSoC™ 4000、4100、4200、4000S、4100S、4100S Plus、4100PS

ソフトウェア バージョン: PSoC™ Creator 4.2 SP2 以降

関連アプリケーション ノート: こちらをクリックしてください。

Intended audience

このドキュメントは、PSoC™ Creator を使って PSoC™ 4 の I2C ブートローダを作成する方が対象です。

更にサンプルコードをお求めでしょうか？以下の通りご対応いたします。

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

Table of contents

Table of contents

1 はじめに

ブートローダは MCU システム設計の共通部分です。ブートローダにより、製品のファームウェアが現場で更新できます。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) あるいは Arm® Serial Wire Debug (SWD) のインターフェースを介して実行されます。しかし、これらのインターフェースは常に現場でアクセスできません。

ここはブートローディングを活用するところです。ブートローディングは、USB、I²C、UART または SPI などの標準通信インターフェースを経由してシステム ファームウェアをアップグレードすることを可能にするプロセスです。ブートローダはホストと通信して、新しいアプリケーション コードやデータを取得し、デバイスのフラッシュ メモリに書き込みます。

本アプリケーション ノートは次のテーマについて説明しています。

- PSoC™ Creator を使用した I²C ブートローダの作成方法
- ブートローダ ホストの項目
  - ブートローダ ホスト ツールの使用方法
  - ブートローダ ホスト システムの構築用の基本的なブロックと機能
  - PSoC™を使用した組み込み I²C ブートローダ ホストの作成方法

本アプリケーション ノートは PSoC™ 4 および PSoC™ Creator IDE の経験者を対象としています。PSoC™ 4 が初めての場合、**AN79953 - Getting Started with PSoC™ 4** を参照ください。PSoC™ Creator が初めての方は、**PSoC™ Creator ホームページ**をご参照ください。

本アプリケーション ノートは、読者がブートローダの概念を理解していることも想定しています。これらの概念をご存知ない場合、**AN73854 - PSoC™ 3, PSoC™ 4 and PSoC™ 5LP Introduction to Bootloaders** をご参照ください。ブートローディングに関わる他のアプリケーション ノート全ての一覧については、**ここをクリックしてください**。

最後に、本アプリケーション ノートは、読者が I²C プロトコルと PSoC™ Creator の I²C (SCB モード) コンポーネントに慣れていることを想定しています。このコンポーネントに慣れていない場合は、**PSoC™ 4 Serial Communication Block (SCB) コンポーネントのデータシート**を参照してください。PSoC™ Creator 内の I²C (SCB モード) コンポーネントを右クリックすることでもデータシートを取得できます。PSoC™ Creator の詳細情報は以下の節を参照ください。

1.1 用語および定義

Figure 1 はブートローダ システムの主な要素を図にしたものです。これは、製品に組み込まれたファームウェアが 2 つの異なる目的－通常動作とフラッシュ メモリの更新－で通信ポートを使用できなくてはならないことを示します。フラッシュの更新方法を知っている組み込みファームウェアの部分は **bootloader** (ブートローダ) と呼ばれています。

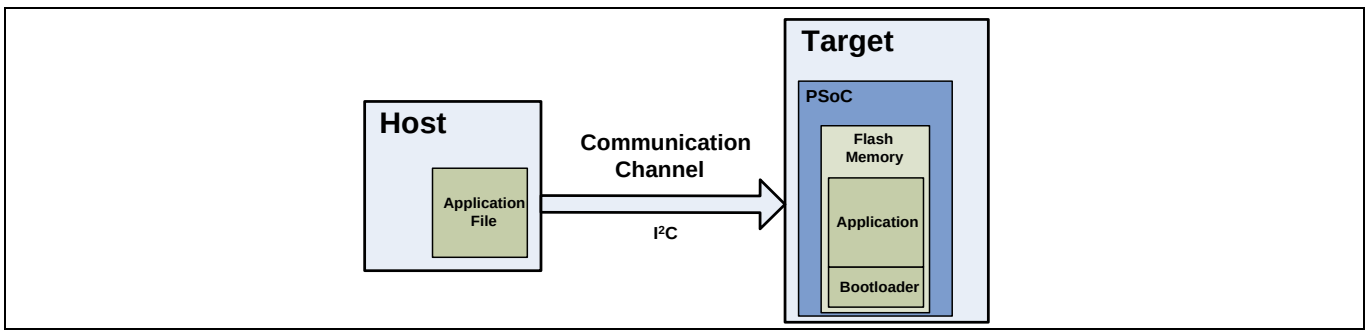


Figure 1 ブートローディングのシステム図

はじめに

フラッシュ メモリを更新するデータを提供するシステムは **Host** (ホスト) 、更新されるシステムは **Target** (ターゲット) と呼ばれます。ホストは外部 PC (PC ホスト) またはその他の MCU (組み込みホスト) 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 **Bootloading** (ブートローディング) 、**Bootload operation** (ブートロード動作) 、または略して **Bootload** (ブートロード) と呼ばれます。フラッシュにあるファームウェアは **Application** (アプリケーション) または **Bootloadable** (ブートローダブル) と呼ばれます。

ブートローディングのもう 1 つの用語はインシステム プログラミング (ISP) です。サイプレスは、似た名前ですが、「In-System Serial Programmer (ISSP)」と呼ばれる異なる機能、および「Host-Sourced Serial Programming (HSSP)」と呼ばれる動作を備えた製品があります。詳細については、**AN84858 - PSoC™ 4 Programming Using an External Microcontroller (HSSP)**をご参照ください。

1.2 ブートローダの使用

ブートローダの通信ポートは、通常、ブートローダと実際の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最初のステップは、アプリケーションではなくブートローダが実行されるようにターゲット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。ブートローダが「OK」応答を送信すると、ブートローディングが開始できます。

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込み、フラッシュ書き込みコマンドに変換し、それらのコマンドをブートローダに送信します。ファイル全体が送信された後、ブートローダは新しいアプリケーションに制御を渡すことができます。

1.3 ブートローダの機能フロー

通常は、デバイスをリセットした時、ブートローダは最初に実行される機能です。その後、次の動作を実行します。

- アプリケーションを実行させる前に、その妥当性をチェックする。
- ホストとの通信を開始するタイミングを管理する。
- ブートロード／フラッシュ更新動作を実行する。
- アプリケーションに制御を渡す。

Figure 2 はこの動作を示すフロー図です。

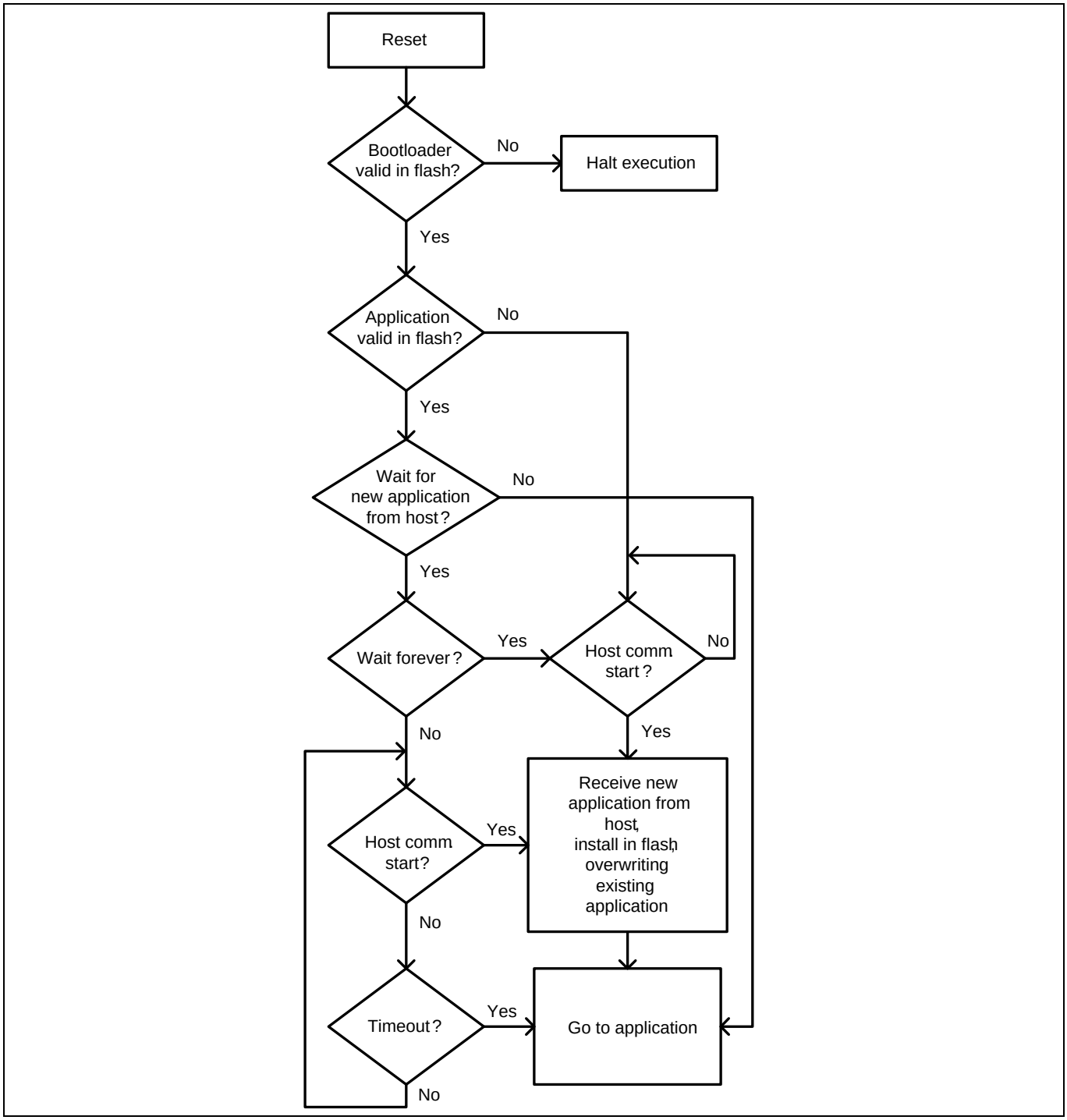


Figure 2　　ブートロード処理のフローチャート

1.4　　ブートローダに移行する技術

前に説明したように、ブートローダはリセット時に最初に実行される機能です。**Figure 2** に示すように、ブートローダ コードはアプリケーションに制御を渡す前に、短期間ホスト (からのコマンド) を待ちます。このことで、ホストはブートロード動作を開始する機会を逃すことがあります。しかし、ブートローディングを開始する別の方法があります。それは、アプリケーションまたはブートローダブルからブートローダに制御をもう一度渡す方法です。

1.4.1 ブートローダブル API

PSoC™ Creator でのブートローダブル コンポーネントは、ブートローダを開始する API 機能、`Bootloadable_Load()` を備えています。これにより、ホストはいつでもブートローダ動作を開始できます。この方法の問題は、アプリケーションの更新を実行するためにアプリケーション コードに依存しなければならないことです。アプリケーションにブートローダへの制御移管を行わない欠陥があると、何が起こるでしょうか？

1.4.2 ブートローダのカスタマイズ

その代わりに、ブートローダはホストのために無限の時間待つほうが良いです。それを行うためには、`Bootloader_Start()` を呼び出してその通常のルーチンを実行する前に、ブートローダ プロジェクトをカスタマイズし、いくつかのユーザー入力を確認します。例えば、ブートローダは `Bootloader_Start()` を呼び出す前に、UART を監視しユーザー コマンドが発生するまで待ちます。詳細については、**AN73854 - PSoC™ 3, PSoC™ 4 and PSoC™ 5LP Introduction to Bootloaders** を参照ください。

2 プロジェクト

本節では、下記の PSoC™ Creator プロジェクトを作成する手順について説明します。

- I²C ブートローダ
- ブートローダブル
- 組み込みブートローダ ホスト

このプロジェクトはサイプレス開発キットと一緒に使用するように設計されています。キットの選択はターゲット デバイスに基づきます; 詳細は**付録 F - キットの選択**を参照ください。キットに基づいてピン接続を変更する必要があります。接続のタイプについては特定のキットに関する資料をご確認ください。しかし、プロジェクトは他のカスタム基板に容易に適用できます。

2.1 I²C ブートローダ

本節では、I²C ベースのブートローダ プロジェクトを作成し構築します。このプロジェクトの特徴の 1 つはブートロード中に、キットの赤色 LED が点滅します。次の例では、PSoC™ 4200 デバイスを使用する I²C ブートローダ プロジェクトを示しますが、処理は他の全ての PSoC™デバイスと全く同じです。

1. 新しい PSoC™ Creator プロジェクトを作成して、「I2C_Bootloader_Red」と名付けます。ターゲット PSoC™デバイスを選択し、プロジェクトの新しいワークスペースを作成します。

*Note: PSoC™ Creator 3.1 以前のバージョンである場合、プロジェクトを作成するときに「Application Type」を指定する必要があります。アプリケーション タイプの指定は、**「Advanced」**タブの隣にある「+」ボタンをクリックし、設定オプションを拡張します。アプリケーション タイプとしてブートローダを選択します。*

2. 先頭の設計回路図に I²C (SCB モード) とブートローダ コンポーネントを追加します。LED を点滅させるために、PWM (TCPWM モード)、クロック、およびデジタル出力ピンのコンポーネントを追加します。**Table 1** に示すように、コンポーネントの名前を変更します。

Table 1 I2C_Bootloader_Red プロジェクト コンポーネント名

コンポーネント	名称
Bootloader_1	Bootloader
I2C_1	I2C
PWM_1	PWM
Clock_1	Clock
Pin_1	Pin_LED

3. ブートローダを設定するために、コンポーネントをダブルクリックします。**Figure 3** に示すように、**Communication component (通信コンポーネント)**として **I2C** を選択します。他のパラメーターは初期設定のままにしてください。これらの設定パラメーターの詳細については、「**Bootloader Component datasheet**」を参照してください。

*Note: パラメーターの「**Wait for command time**」**(コマンド タイムの待ち時間)** は 2 秒に設定します。ブートローダの待機時間の詳細については、ブートローダ コンポーネント データシートの「Wait For Command Time」を参照してください。*

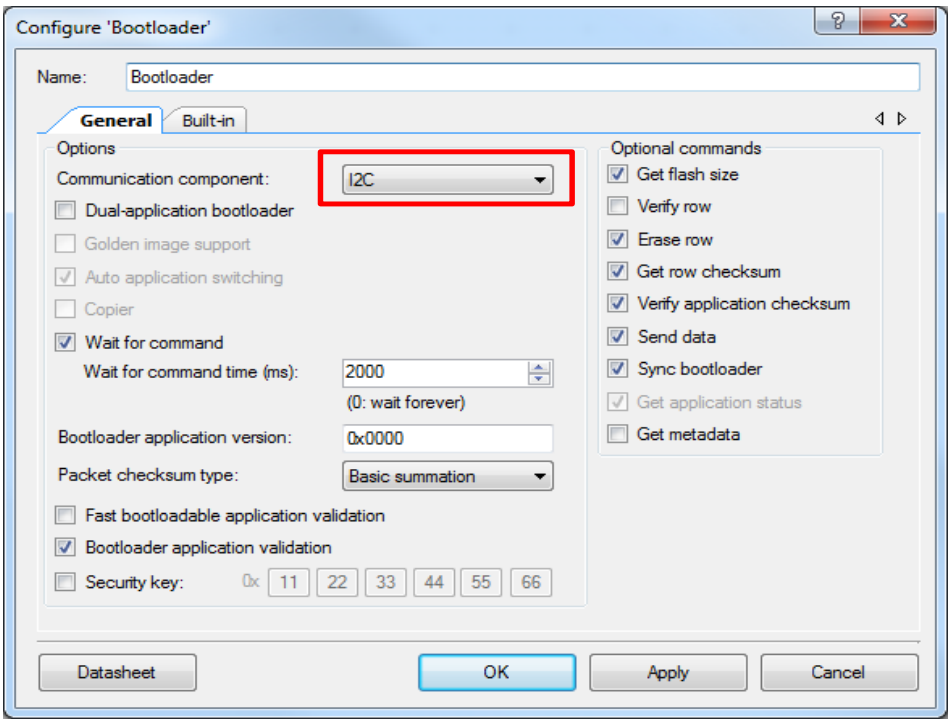


Figure 3　ブートローダの設定

4. I²C (SCB モード) コンポーネントを設定するために、それをダブルクリックします。デフォルトでは、モードを「Slave」に、データレートを 100kbps に、スレーブ アドレスを 8 に設定します。別のアドレスを使用する場合は、**スレーブ アドレス** ボックスに入力します。他のパラメーターは初期設定のままにしてください。**Figure 4** に、I²C コンポーネントの基本設定タブを示します。

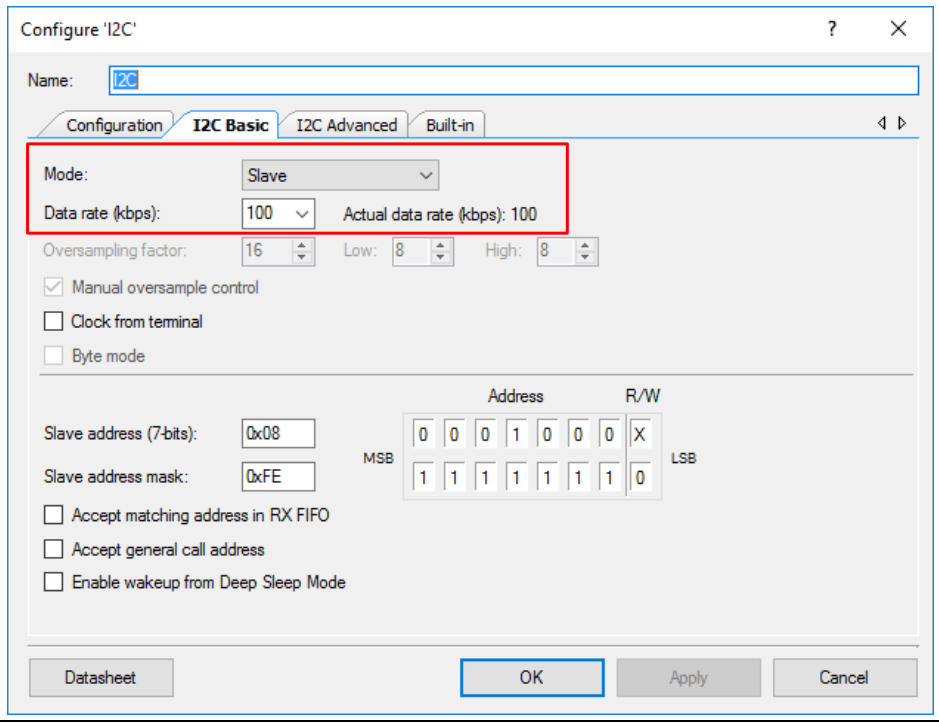


Figure 4　基本 I²C (SCB モード) のコンフィギュレーション

プロジェクト

I²C (SCB モード) コンポーネントは、I²C バス (オープンドレイン、Low 駆動) を使用するために自動的に設定されるピン コンポーネントを含んでいます。10 ページの **Figure 8** に示すように、外部のプルアップ抵抗を追加して、それらを実際の端子に割り当てる (10 ページのステップ 8) 必要があります。

5. PWM (TCPWM モード) コンポーネントを設定するために、ダブルクリックします。「**Period**」を 254 に、「**Compare**」を 127 に設定します。他のパラメーターは初期設定のままにしてください。**Figure 5** に、PWM (TCPWM モード) コンポーネントの基本コンフィギュレーション タブを示します。

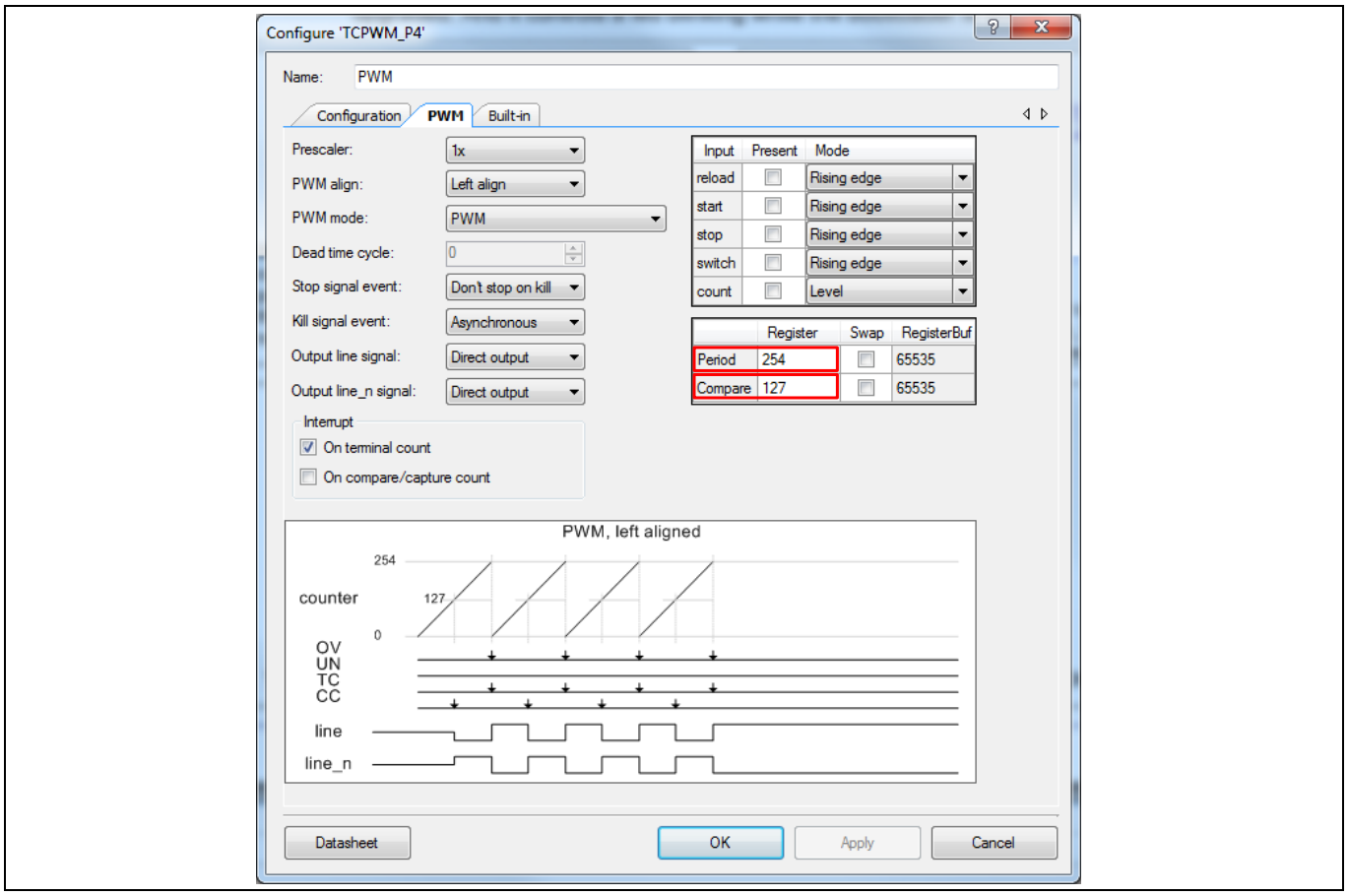


Figure 5　基本 PWM (TCPWM モード) の設定

6. ピン コンポーネントに対しては、それらのパラメーターを初期値のままにします。**Figure 6** に示すように、配線を使用して PWM コンポーネントの「line」端子にピンを接続します。

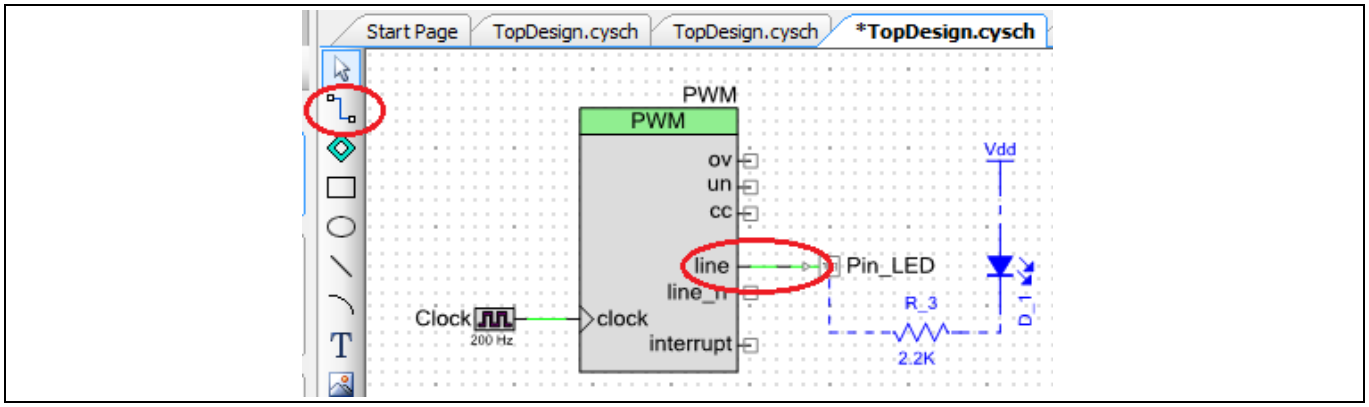


Figure 6　ピン接続

プロジェクト

7. クロック コンポーネントを設定するために、ダブルクリックします。**Figure 7** の通りに、**「Frequency」** を 200Hz に設定します。他のパラメーターは初期設定のままにしてください。

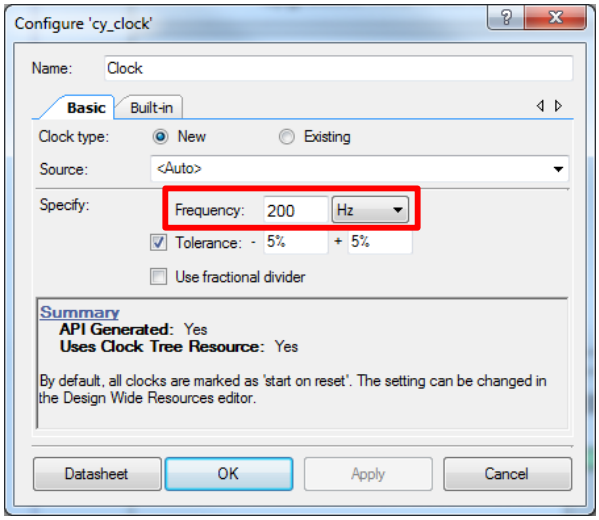


Figure 7　クロックのコンフィギュレーション

抵抗と LED が注釈コンポーネントとして追加された後、プロジェクトのトップ デザインは **Figure 8** のようになります。

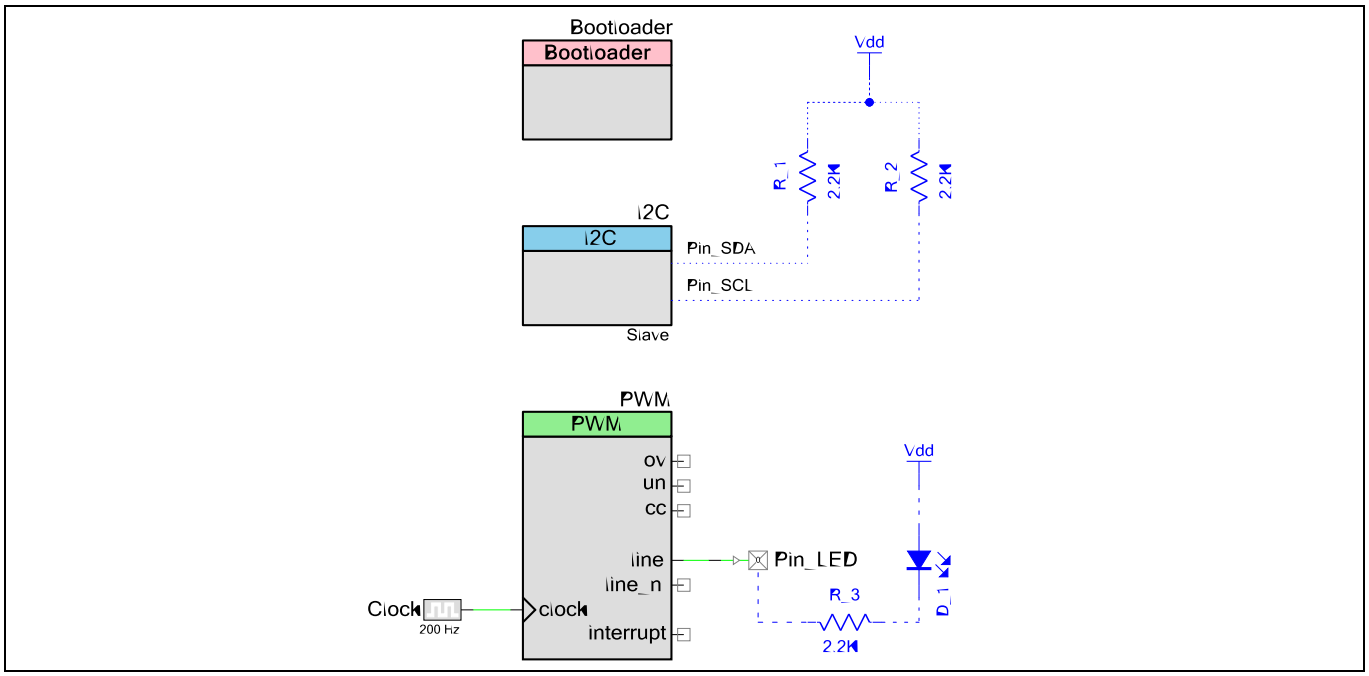


Figure 8　I2C_Bootloader_Red プロジェクトのトップ デザイン

8. I²C ピン コンポーネントを実際の端子に割り当てます。**「Workspace Explorer」** ウィンドウ内で、*「I2C_Bootloader_Red.cydwr」* ファイルをダブルクリックして、**「Pins」** タブをクリックします。ピンの割り当てはデバイスに依存します。詳細はデバイス データシートを参照ください。例えば、**CY8CKIT-042** の **PSoC™ 4200** デバイスでは **Figure 9** のように、そして **CY8CKIT-040** の **PSoC™ 4000** デバイスでは **Figure 10** のようにピンを割り当てます。

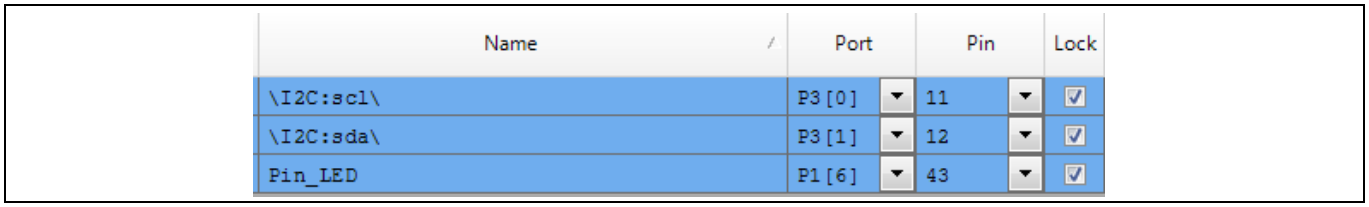


Figure 9　CY8CKIT-042 用の I²C ピンの割り当て

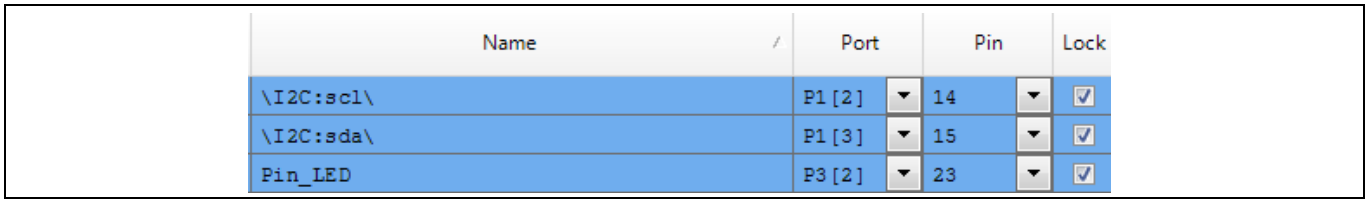


Figure 10　CY8CKIT-040 用の I²C ピンの割り当て

9. `Bootloader_Start()`関数を main()関数に追加します。この API 関数はブートロード動作の全てを行います。これは、リターンしません; ソフトウェアのデバイスリセットで終了します。そのため、この関数を呼び出した後のコードは実行されません。**Code Listing 1** に示すように、2 つの API 関数を `main()`に追加して PWM を初期化します。このコンポーネント API の詳細については、「**TCPWM Component datasheet**」を参照してください。

Code Listing 1　ブートローダ内の PWM の初期化

```
Int main()
{
  /* Initialize PWM */
  PWM_Start();
  PWM_TriggerCommand(PWM_MASK,
                     PWM_CMD_START);

  Bootloader_Start();

  for(;;)
  {
    /* Place your code here. */
  }
}
```

10. プロジェクトをビルドし、ターゲット デバイスの選択に応じた指定するキットにプログラムします。

2.2 ブートローダブル

ここでは、2 つのブートローダブル プロジェクトを作成します。最初のプロジェクトは、PSoC™開発キット上にある緑色 LED を点滅させます。2 番目のプロジェクトは、ブートローダブル プロジェクトからブートローダ プロジェクトに入る方法を説明し、キットの青色 LED を点滅させます。

*Note: 27 ページの **Figure 31** と 27 ページの **Figure 32** に示すように、**CY8CKIT-042** と **CY8CKIT-042** 上には RGB LED があります。ここでは、緑色 LED と青色 LED は、プロジェクトが実行していることを示すために使用されます。*

2.2.1 Bootloadable_Green プロジェクト– 例 1

本節では最初のブートローダブル プロジェクトを作成する手順について説明します。

1. アプリケーション型のブートローダブルの新しい PSoC™ Creator プロジェクトを作成します。**「Bootloadable_Green」** プロジェクトと名付けます。本プロジェクトと **I2C_Bootloader**_Red プロジェクト用のデバイスは同じである必要があります。
2. 本プロジェクトに対しては、ブートローダブル コンポーネントとデジタル出力ピン コンポーネントが必要です。2 つのコンポーネントをトップ デザイン回路図に追加します。コンポーネントを **Table 2** の通りに名付けます。

Table 2 「Bootloadable_Green」 プロジェクトのコンポーネント名

コンポーネント	名称
Bootloadable_1	ブートローダブル
Pin_1	Pin_LED

3. ブートローダブル コンポーネントをダブルクリックして、設定を行います。

ブートローダブル プロジェクトは常にブートローダ プロジェクトの*.hex* ファイルにリンクします。プロジェクトをリンクするためには、**Dependencies** タブに移動して、**Figure 11** に示すように、ブートローダブルを *I2C_Bootloader_Red.hex* ファイルにリンクします。ブートローダブル コンポーネント設定の詳細については、**Bootloadable Component datasheet** を参照してください。

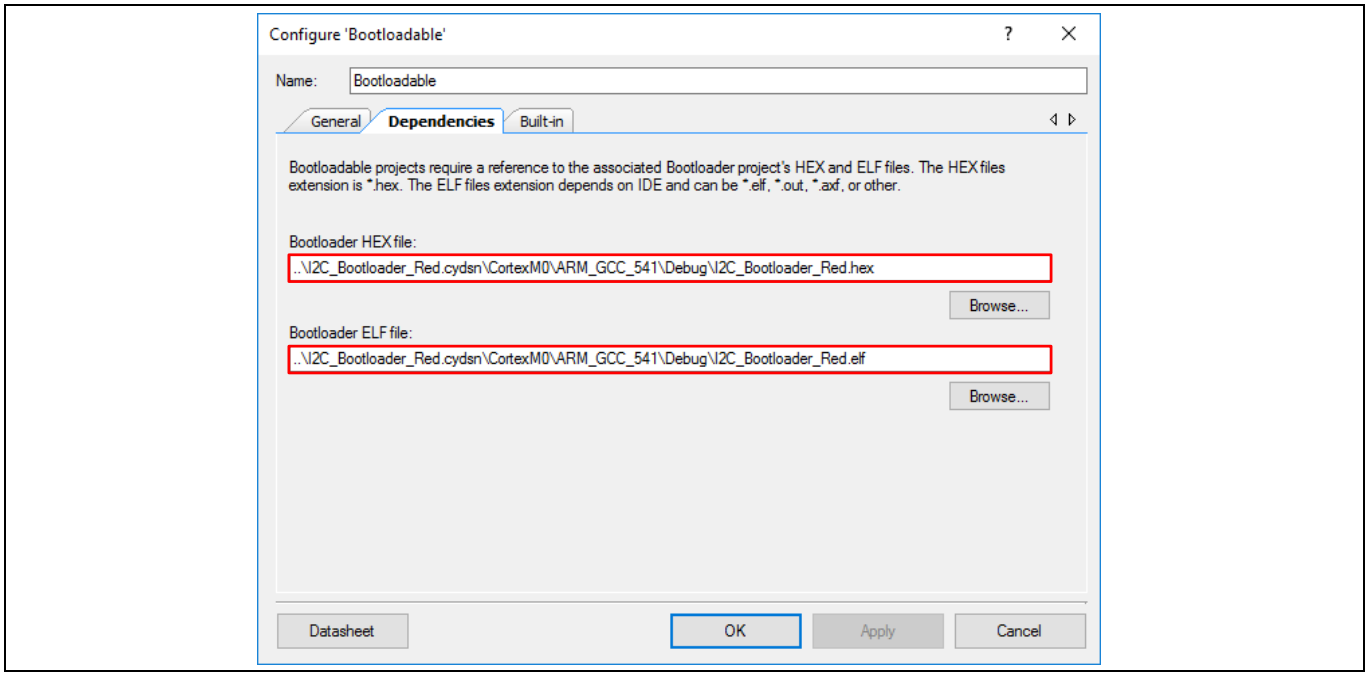


Figure 11　ブートローダブル コンポーネントの設定

I2C_Bootloader_Red.hex ファイルをブートローダ プロジェクトの Debug または Release フォルダで見つけられます。

1. PSoC™ 4000、4100、4200 の場合

   ..\I2C_Bootloader_Red.cydsn\CortexM0\ARM_GCC_541\Debug\I2C_Bootloader_Red.hex

2. PSoC™ 4000S、4100S、4100S Plus、および 4100PS の場合

   ..\I2C_Bootloader_Red.cydsn\CortexM0p\ARM_GCC_541\Debug\I2C_Bootloader_Red.hex

3. ピン コンポーネントを設定するために、それをダブルクリックします。**Figure 12** に示すように、**HW Connection** を無効にします。**Figure 13** に示すように、駆動モードを **Strong Drive** に設定します。

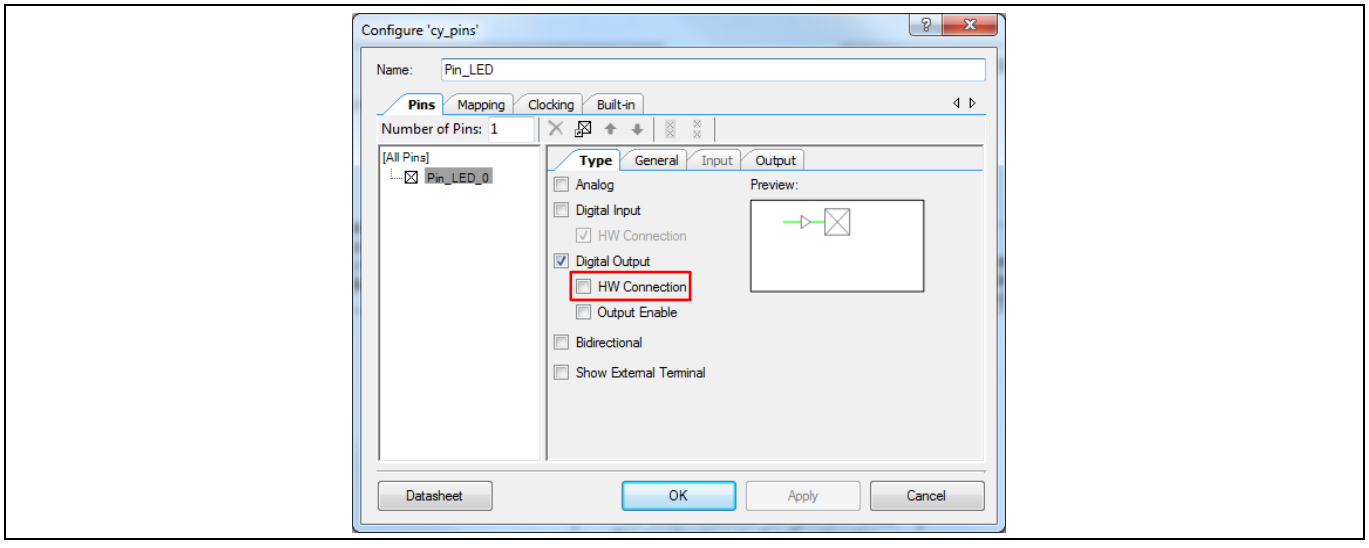


Figure 12　ピン コンポーネントの設定 - 「Type」タブ

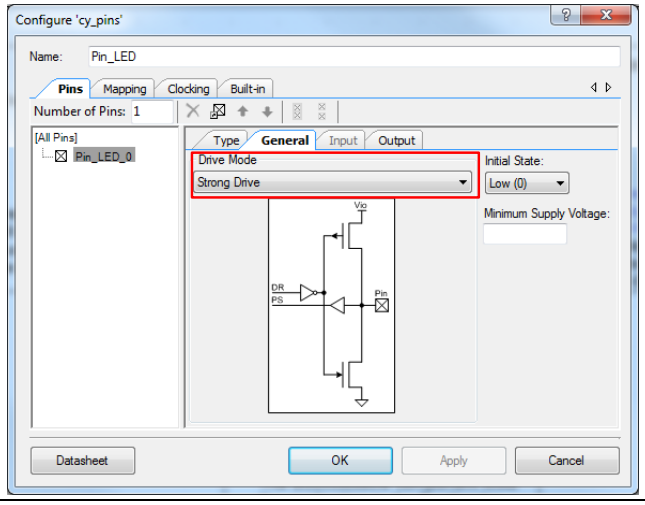


Figure 13　ピンコンポーネントの設定 – 「General」タブ

LED 用の注釈コンポーネントを追加した後、トップ デザインが終了します; **Figure 14** と同じようになります。

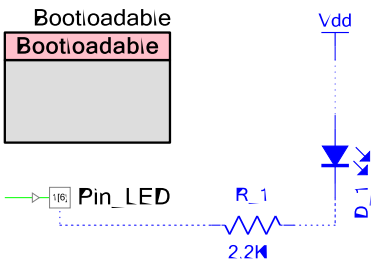


Figure 14　Bootloadable_Green プロジェクトのトップ デザイン

4. 下記のコードを *main.c* に追加して LED を点滅させます。

```
void main()
{
for(;;)
  {
      /* Toggle the LED */
Pin_LED_Write(~Pin_LED_Read());

      /* Delay 1 second */
CyDelay(1000u);
  }
}
```

5. ピン コンポーネントを実際の端子に割り当てます。「**Workspace Explorer**」ウィンドウで、*Bootloadable_Green.cydwr* ファイルをダブルクリックして、ピンを割り当てます。ピンの割り当てに

ついては、キットのユーザー ガイドを参照ください。例えば、**Figure 15** には **CY8CKIT-042** キット基板の、そして **Figure 16** には **CY8CKIT-040** キット基板用のピン割り当てを示します。

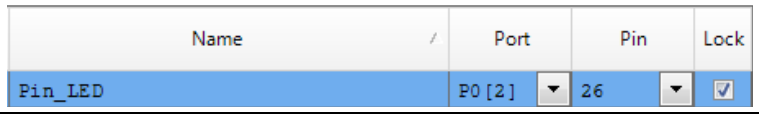


Figure 15　CY8CKIT-042 用に Bootloadable_Green プロジェクトのピンの割り当て

Figure 16　CY8CKIT-040 用の Bootloadable_Green プロジェクトのピンの割り当て

6. プロジェクトを構築します。ブートローダブル プロジェクトが構築されると、PSoC™ Creator は*.cyacd* ファイルを作成します。このファイルはターゲットにブートロードされます。このファイルと内容についての詳細は、**付録 B** を参照してください。

ここで、PSoC™ Creator を使ってこのプロジェクトを PSoC™にブートロードしてみましょう。

2.2.2　PC のホストを使用してブートローディング

ブートローダ ホスト実行可能ファイルは、PC ホストからアプリケーションをブートロードするために PSoC™ Creator に備えられています。PSoC™ 4 および PSoC™アナログ コプロセッサのキットは基板搭載の PSoC™ 5LP を使用する USB-I²C ブリッジを実装します。このブリッジは、**Figure 17** の **CY8CKIT-042** キット用および **Figure 18** の **CY8CKIT-040** キット用のように、PC をブートローダにインターフェースするために使用することができます。

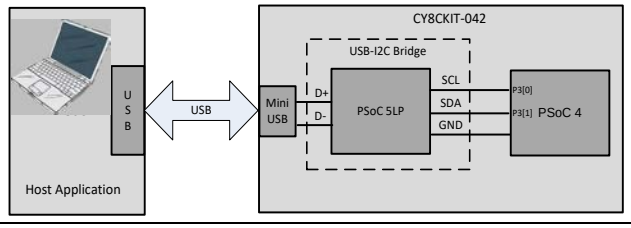


Figure 17　CY8CKIT-042 用の PC ホストを使用してブートローディング

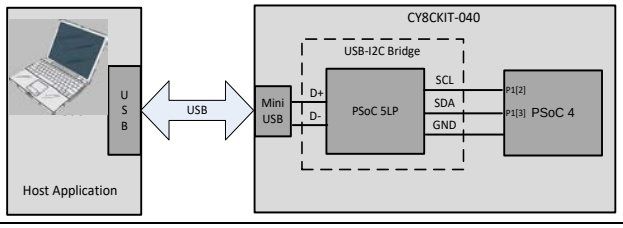


Figure 18　CY8CKIT-040 用の PC のホストを使用してブートローディング

プロジェクト

Note: *すべての PSoC™ 4 キットで、ピンは基板搭載の PSoC™ 5LP SCL に接続し、SDA ピンには内部プルアップ抵抗があります。そのため、外部のプルアップを使用する必要がありません。I²C ピン接続の詳細情報はキットのユーザー ガイドを参照ください。*

Note: *ブートロードのために、**CY8CKIT-002 MiniProg3** を USB-I²C ブリッジとして使用することもできます。詳細については、「**MiniProg3 User Guide**」知識ベース記事「**MiniProg3 Connections for Bootloading Over I2C**」を参照してください。*

PSoC™ 4200 デバイス用のブートローダ ホスト プログラムを使用して、アプリケーションをブートロードするには、以下の手順に従ってください。手順は、正確に他の PSoC™ファミリについても全く同じです。前述した通り、ブートロード動作を開始する前に、PSoC™デバイスにブートローダ プロジェクトをプログラムする必要があります。

1. **Figure 19** に示すように、先ず **CY8CKIT-042** を USB ケーブル (電源を供給) を介して PC に接続します。

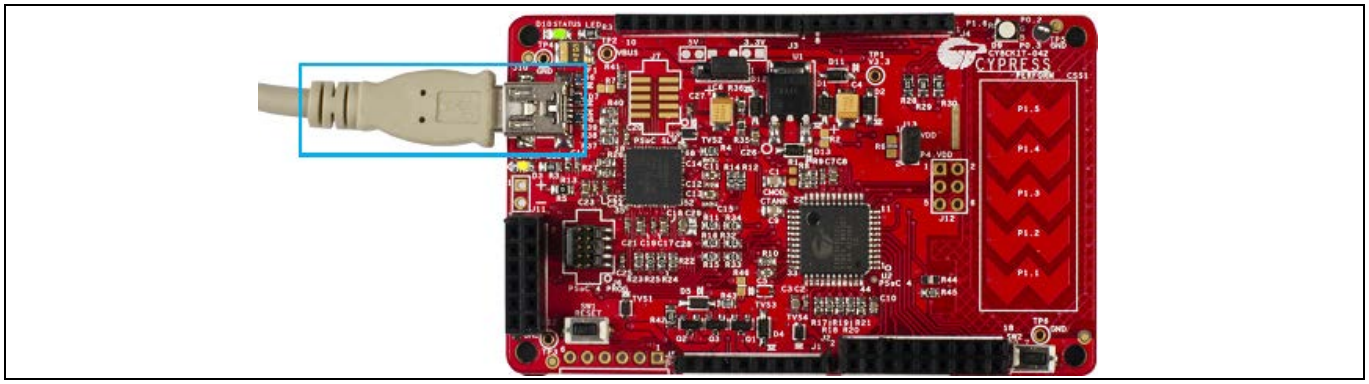


Figure 19　USB ケーブルを CY8CKIT-042 の J10 に接続

2. PSoC™ Creator で「**Tools** > **Bootloader Host**」に移動し、ブートローダ ホスト ツールを開きます。
3. **Figure 20** に示すように、ブートローダ ホスト アプリケーションの I²C の設定は、ブートローダ プロジェクトの I²C (SCB モード) コンポーネント設定 (8 ページの **Figure 4**) と同じであることを確認してください。

Note: *USB ケーブルを介してキットに接続した後、ブートローダ ホスト GUI に KitProg の情報がない場合、**Filters** をクリックして、**Show I2C Devices** が有効になっていることを確認してください。*

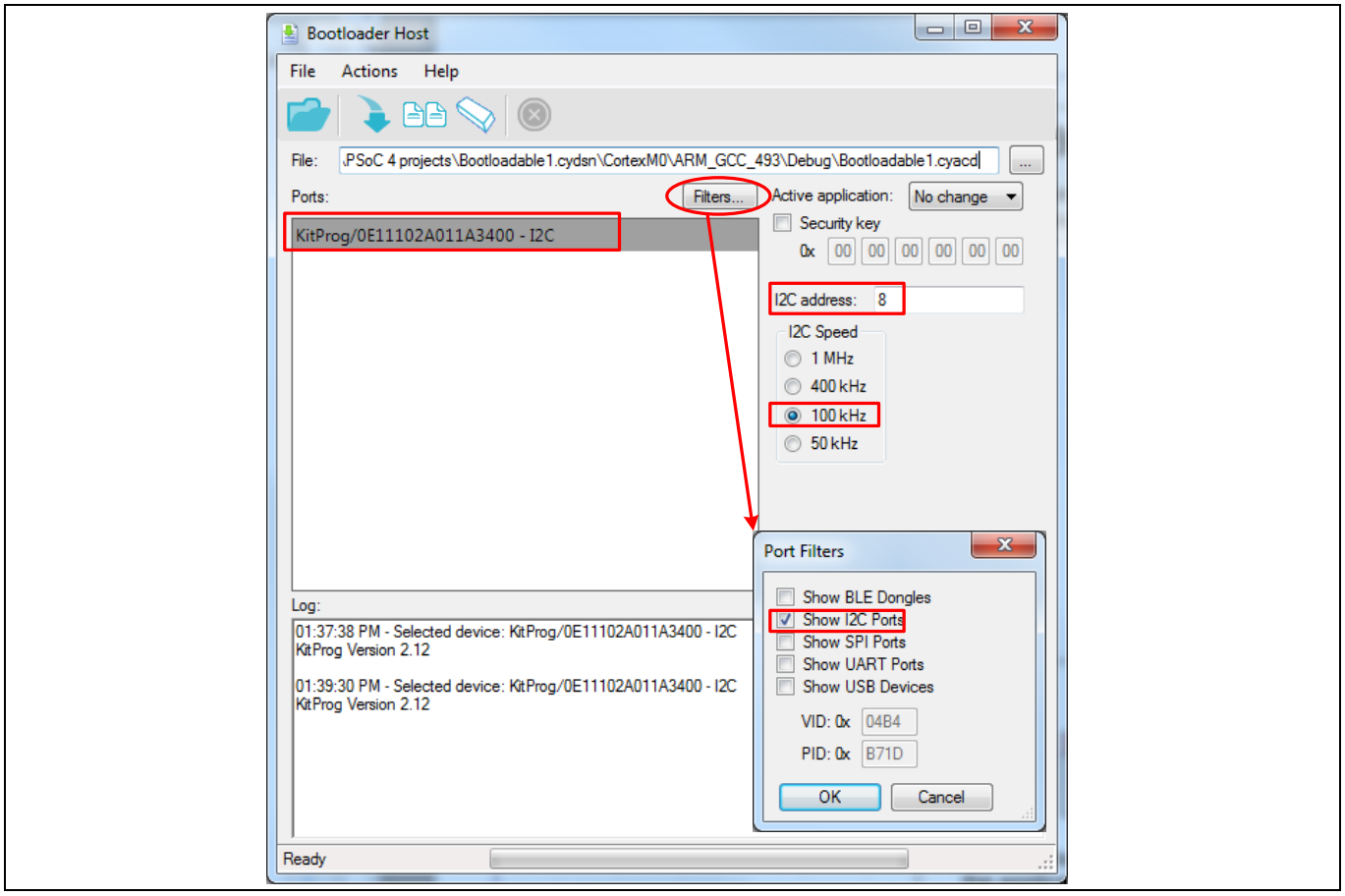


Figure 20　ブートローダのホスト アプリケーション

4. **File** ボタンを押して、ブートローダブル プロジェクトの Debug または Release フォルダ内のブートローダブル ファイル *Bootloadable_Green.cyacd* を選択します。
   - PSoC™ 4000、4100、4200 の場合

     ..\Bootloadable_Green.cydsn\CortexM0\ARM_GCC_541\ Debug\Bootloadable_Green.cyacd

   - PSoC™ 4000S、4100S、4100S Plus、および 4100PS の場合

     ..\Bootloadable_Green.cydsn\CortexM0p\ARM_GCC_541\ Debug\Bootloadable_Green.cyacd

5. デバイスをブートロードするには、**Program** ボタンを押します。**Figure 21** と同じような画面が開きます。

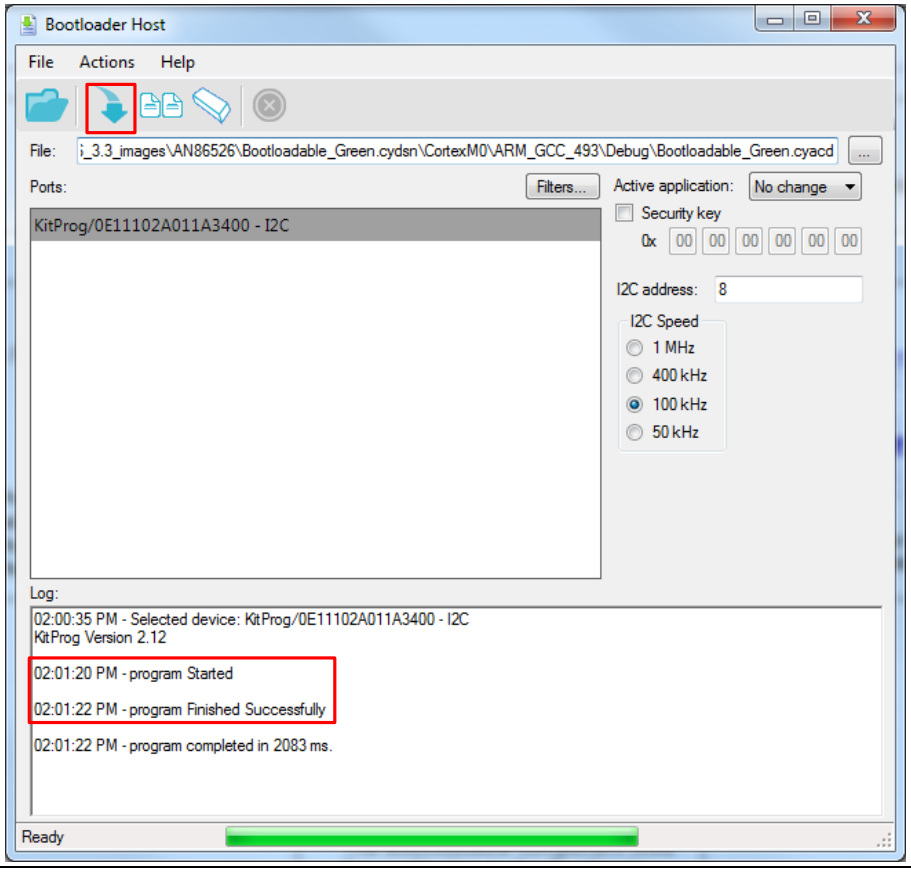


Figure 21　ブートローダブル プロジェクトのダウンロード

6. ブートローダブル プロジェクトがブートロードに成功した後、ソフトウェアのリセットが発生し、デバイスは新しいアプリケーションを実行し始めます。キットの RGB LED が緑色に点滅します。

他のアプリケーションをブートロードするには、デバイスをリセット (PSoC™開発キットのリセットボタンを押して離す) して、ブートローダをアクティブにします。そして、8 ページの **Figure 3** でセットされた「**Wait for command time**」の期間内にブートローダ ホストのプログラム ボタンを押します。ブートローダの待機時間の詳細については、**Bootloader and Bootloadable Component datasheet** 内に記載されている「Wait for Command Time」の項を参照してください。

2.2.3　Bootloadable_Blue プロジェクト – 例 2

現在のブートローダブル プロジェクトが実行中に、新しいプロジェクトをブートロード (アプリケーションのアップグレード) したい場合、ブートローダブル プロジェクトは API 関数 `Bootloadable_Load()` を呼び出すことでブートローダを起動します。

この例では、ボタンを押すと、`Bootloadable_Load()` 関数が呼び出されます。本節ではこのブートローダブル プロジェクトの作成手順を説明します。

1. **例 1** と同じように Bootloadable のアプリケーション タイプの新しい PSoC™ Creator プロジェクトを作成します。「Bootloadable_Blue」プロジェクトと名付けます。本プロジェクトと I2C_Bootloader_Red プロジェクト用の PSoC™デバイスは同じである必要があります。
2. 本プロジェクトでは、1 つのブートローダブル コンポーネント、2 つのピン コンポーネントおよび 1 つの割り込みコンポーネントが必要です。これらのコンポーネントをトップ デザイン回路図に追加し、**Table 3** の通りに名前を付けてください。

プロジェクト

Table 3　　ブートローダブル コンポーネント名

コンポーネント	名称
Bootloadable_1	ブートローダブル
Pin_1 (デジタル入力ピン)	Pin_StartBootloader
Pin_2 (デジタル出力ピン)	Pin_LED
isr_1	isr_EnterBootloader

3. 13 ページの **Figure 11** に示すように、ブートローダブル コンポーネントをブートローダ プロジェクトにリンクします。
4. デジタル入力ピン「Pin_StartBootloader」はアプリケーションからブートローダに切り替えるために使用されます。このピンに接続されたボタンを押すと、アプリケーションは API 関数 `Bootloadable_Load()`を呼び出すことでブートローダを送り込みます。ブートローダは、ホストがブートロード動作を開始するのを無期限に待ちます。

   DVK ボタンを押すとグランドに短絡するので、**Figure 22** に示すようにピンの駆動モードを **Resistive Pull Up** に設定します。また、**Figure 23** に示すように、その**立ち下がりエッジ**で割り込みが生成するように設定します。ボタンを押すと、割り込みが生成されます (ボタンが開放されていると割り込みは生成されません)。

   最後に、**Type** タブをクリックして、**HW Connection** を無効にします。

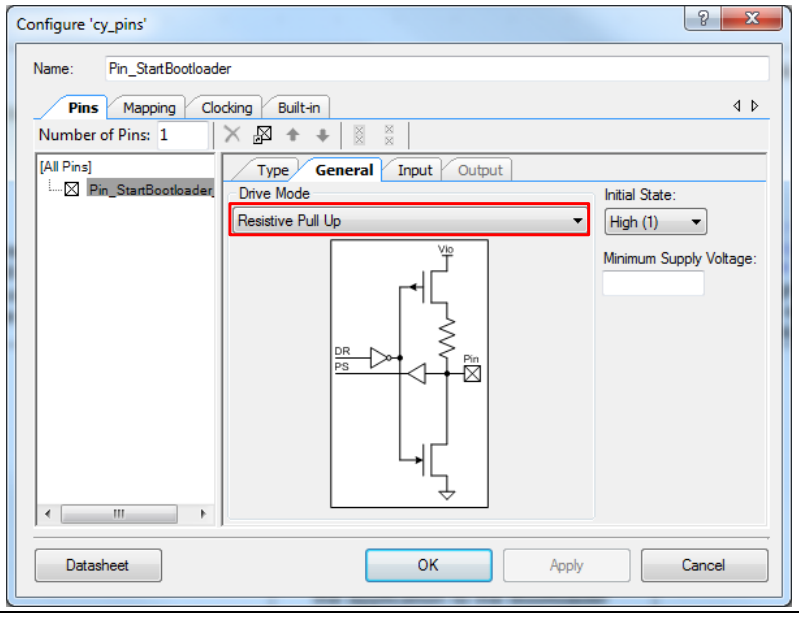


Figure 22　デジタル入力ピンの設定

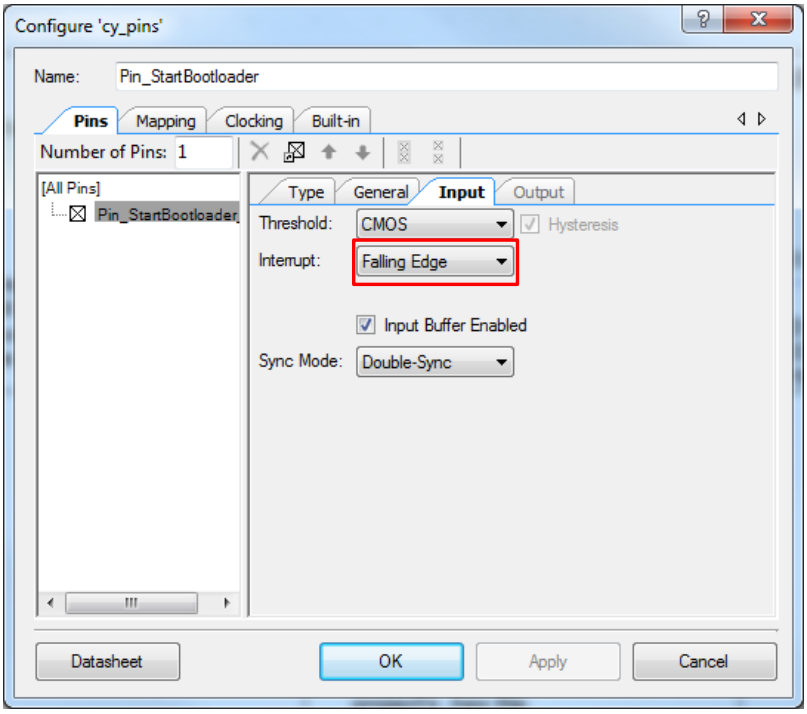


Figure 23　デジタル入力ピンの設定

5. デジタル出力ピン Pin_LED は LED を制御するために使用されます。その設定は**例 1** の Pin_LED の設定と同じです。**Figure 12** と **Figure 13** を参照してください。
6. ISR のコンポーネント「isr_EnterBootloader」を「Pin_StartBootloader」の割り込み端子 (irq) に接続します。ボタンと LED 用の注釈コンポーネントが追加された後、トップ デザインが終了します。同様に **Figure 24** のようになります。

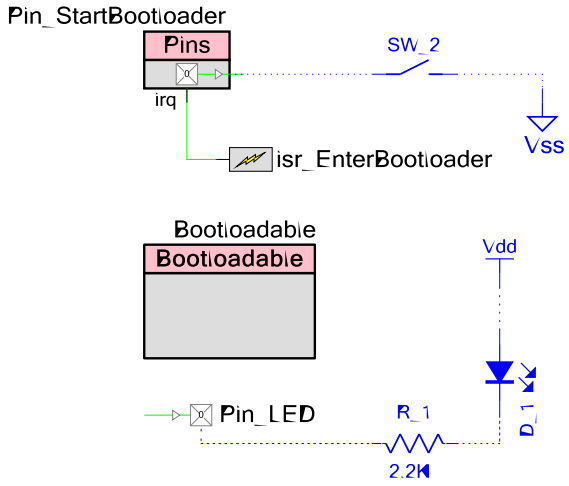


Figure 24　Bootloadable_Blue プロジェクトのトップ デザイン

7. ピンコンポーネントを実際の端子に割り当てます。**Workspace Explorer** ウィンドウで、*Bootloadable_Blue.cydwr* ファイルをダブルクリックして、ピンを割り当てます。ピンの割り当てについてはサイプレス キット用のキット ユーザー ガイドを参照ください。**Figure 15** は **CY8CKIT-042** キット基板用の、そして **Figure 16** は **CY8CKIT-040** キット基板用のピンの割り当てを示します。

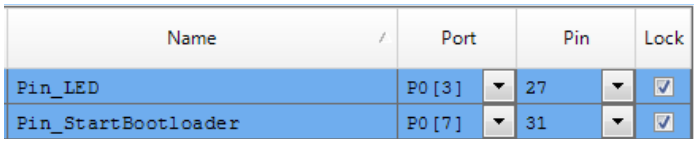


Figure 25　CY8CKIT-042 の Bootloadable_Blue プロジェクトのピンの割り当て

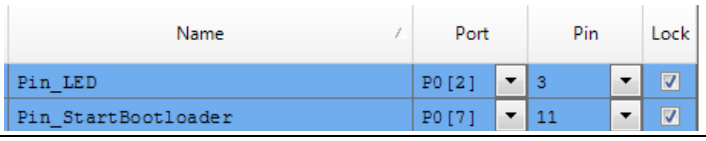


Figure 26　CY8CKIT-040 の Bootloadable_Blue プロジェクトのピンの割り当て

8. プロジェクトを構築します。これで、ISR コンポーネント API ファイルが作成されます。そして、コードを割り込みサービス ルーチンに追加して、変数 `bootload_flag` をセットします。コードは以下のようになります。

```
CY_ISR(isr_EnterBootloader_Interrupt)
{
      /* Place your Interrupt code here.
      */
      /* `#START
      isr_EnterBootloader_Interrupt`
      */
  bootload_flag = 1u;
  Pin_StartBootloader_ClearInterrupt();
        /* `#END` */
}
```

Note:　*変数 `bootload_flag` は、main.c で定義されているので、isr_EnterBootloader.c ファイル内で外部変数として宣言しなければなりません。また、ビルド警告を回避するために、isr_EnterBootloader.c ファイル内に `#include <device.h>` を追加します。*

9. 完成した Bootloadable_Blue プロジェクトは本アプリケーション ノートと関連付けられています。この関連するプロジェクトの *main.c* ファイルから、ユーザー プロジェクトの *main.c* ファイルにリストされたコードを挿入してください。
`main()`関数は継続的に `bootload_flag` 変数をチェックします。変数がセットされると、`main()`関数は、LED を消し、ブートローダを起動するために API 関数 `Bootloadable_Load()`を呼び出します。
10. プロジェクトを再び構築します。**PC のホストを使用してブートローディング**の節に記載された手順の通りに、ブートローダブル プロジェクト *Bootloadable_Blue.cyacd* をダウンロードします。ターゲット上のキットの RGB LED は青色に点滅します。
11. **SW2** スイッチを押してブートローダに入ると、キットの RGB LED が赤色に点滅します。

プロジェクト

12. ブートローダ ホストのプログラムを開始し、*Bootloadable_Green.cyacd* のような異なるブートローダブルファイルを選択して、**Program** ボタンを押します。新しいアプリケーションがブートロードに成功した後、キットの RGB LED は緑色に点滅します。

Note: *再度ブートロードするには、デバイスをリセットし、新しい.cyacd ファイルを素早くダウンロードする必要があります。この理由は、Bootloadable_Green アプリケーションがブートローダを起動するための `Bootloadable_Load()` の呼び出し関数を持っていないためで、ブートローダはリセット時にのみ呼び出すことができます。*

2.3 I²C ブートローダ ホスト

用例のプロジェクトを研究することに加えて、ブートローダ ホスト プログラムの一般的な構造を理解することは、自分のブートローダ ホスト システムを構築するのに役立ちます。

2.3.1 ブートローダ ホスト プログラム

Figure 27 はブートローダ システムのプロトコル レベルでのダイヤグラムを説明します。ブートローダのホストとターゲットはそれぞれ 2 つのブロック – コアと通信層があります。

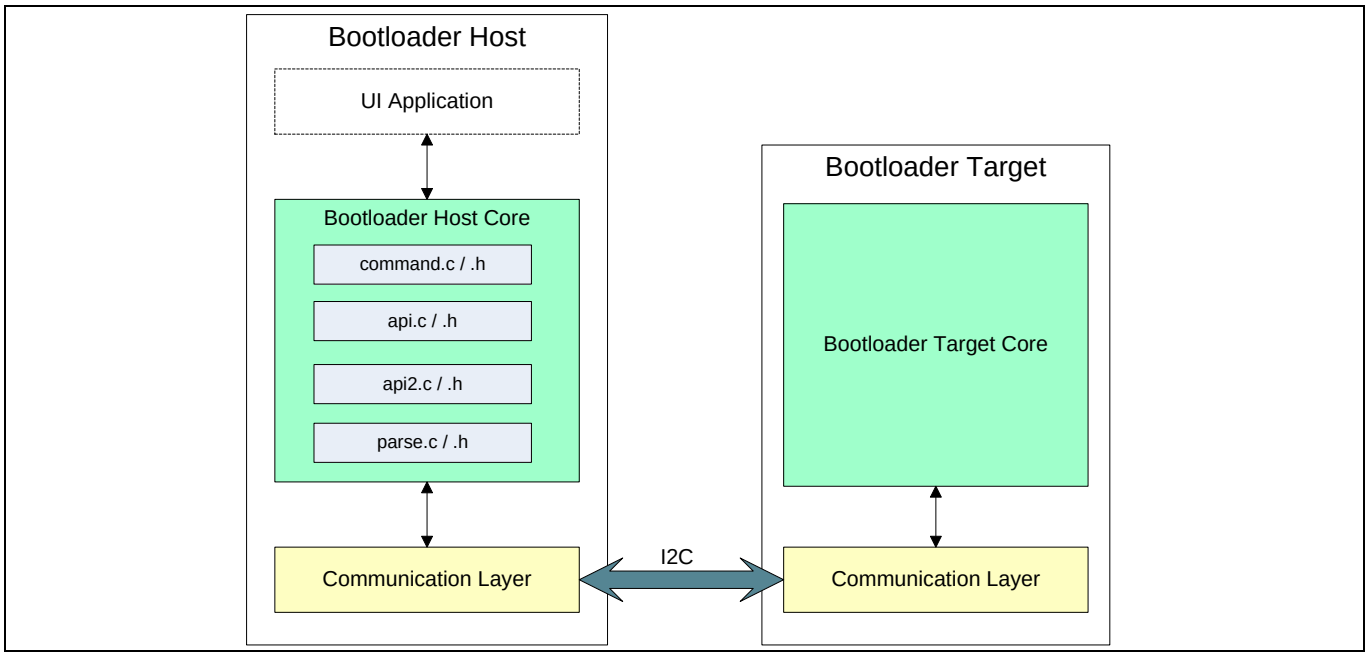


Figure 27 ブートローディング時のプロトコル レベルでのダイヤグラム

ブートローダ ホスト コアは、すべてのブートロード動作を実行します。これは、ターゲットにコマンド パケットとフラッシュ メモリへのデータを送信します。ターゲットからの応答で、ブートロードを実施し続けるかどうか決定します。

ブートローダのターゲット コアはホストからのコマンドを解読し、フラッシュ ルーチン (行の消去、行のプログラム、行の確認など) を呼び出すことでコマンドを実行し、応答パケットを形成します。

ホストとターゲットの両方にある通信層はブートロードのプロトコルをサポートする物理層です。これらは、この関数を実行する通信プロトコル (I²C) 固有の API を含んでいます。この層はホストとターゲット間のプロトコル パケットの送受信を担当します。

2.3.2 ブートローダ システム API

PSoC™ Creator は、ブートローダ プロジェクトを構築する時にブートローダのターゲット コアと通信層用に全ての API を自動的に生成します。

PSoC™ Creator には、この位置で見つけることができるコアに対応したホスト側の API も用意されています。

<インストール フォルダ> \ *PSoC Creator* \ *4.2* \ *PSoC Creator* \ *cybootloaderutils*

これら API ファイルの詳細情報については、**付録 D** を参照してください。

唯一書き込む必要があるコードは、ファイルのペア *communication_api.c* / *.h* 内にある、通信層に対応したホスト側の API 関数です。関数は 4 つあります: `OpenConnection()`、`CloseConnection()`、`ReadData()`、および `WriteData()`。*cybtldr_api.h* にある「CyBtldr_CommunicationsData」のデータ構造において、関数ポインタがこれらの関数にポイントします。

このプロジェクトは、**Bootloadable_Blue プロジェクト – 例 2** を使用して、スイッチの押下でブートローダを起動するのに必要な*.cyacd* ファイルを作成しています。本プロジェクト用の*.cyacd* ファイルを作成して、*Bootloadable_BlueLED.cyacd* のファイル名に変えます。同様に、Pin_LED を P0[2]に割り当てることで *Bootloadable_GreenLED.cyacd* を作成します。これら 2 つのファイルは、クイック リファレンスとして、アプリケーション ノートで提供されます。

2.3.3 I²C ブートローダ ホスト プロジェクトの作成手順

本節では、PSoC™を使用して、他の PSoC™デバイスをブートロードできる、組み込み I²C ブートローダ ホスト プロジェクトの作成方法について説明します。このプロジェクトにより、スイッチを押下するごとに、ホストは 2 つの異なるブートローダブル ファイル (*.cyacd* ファイル) をブートロードできます。

1. 新しい PSoC™ Creator プロジェクトを作成します。PSoC™デバイスを選択し、プロジェクトを **I2C_Bootloader_Host** と命名してプロジェクトの新しいワークスペースを作成します。
2. ブートローダ プロジェクトに I²C スレーブがあるので、ホスト プロジェクトには I² マスターが必要です。そこで、I²C (SCB モード) コンポーネントをトップ デザイン回路図に追加します。また、デジタル入力ピン、割り込み、UART (SCB モード) コンポーネントもトップ デザイン画面に追加します。コンポーネントを **Table 4** の通りに名付けます。

Table 4 I2C_Bootloader_Host プロジェクトのコンポーネントリスト

コンポーネント	名称
I2C_1	I2C
Pin_1	Pin_Switch
isr_1	ISR_Switch
UART_1	UART

3. I2C コンポーネントを設定するために、ダブルクリックします。そして、それをマスター モードに設定します。デフォルトでは、データレートは 100 Kbps です。
4. デジタル入力ピンの Pin_Switch は、ホスト内でのブートロード動作を起動するのに使用されます。キットのボタンを押すと、ピンがグランドに短絡します。そのため、このピンがプルアップ抵抗を持つように設定し、その立下りエッジで割り込みが起きるようにする必要があります。ISR_Switch コンポーネントをこのピンの割り込み出力 (irq) に接続します。
5. UART はブートローディングの状態またはエラーコードを PC に転送するために使用されます。パラメーターはデフォルト設定 (ボーレート: 115200 kbps、データビット: 8 ビット、パリティ: 無し、ストップ ビット: 1 ビット) のままにしておきます。

プロジェクト

ここでは、ボタン用の注釈コンポーネントが追加されて、このプロジェクトのトップ デザインは、**Figure 28** と同じようになります。

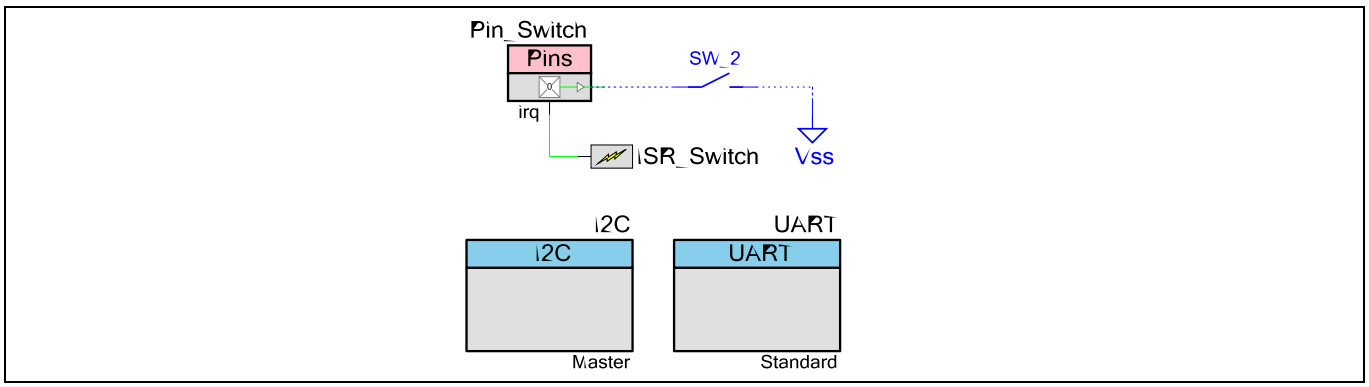
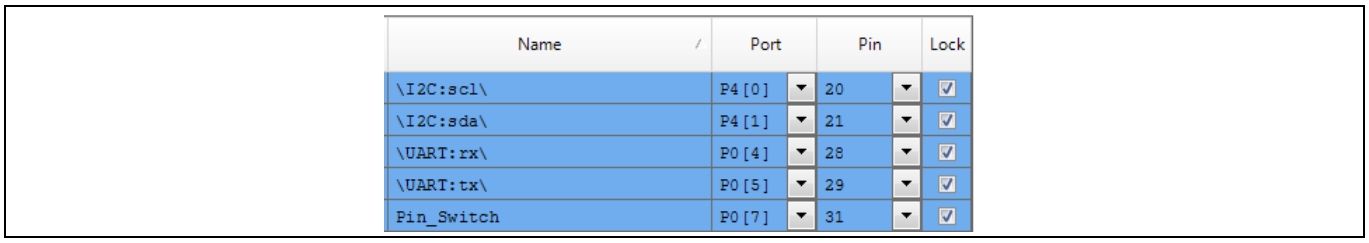


Figure 28　I2C_Bootloader_Host プロジェクトのトップ デザイン

6. 入力と出力のピンを割り当てます。「**Workspace Explorer**」ウィンドウで、*I2C_Bootloader_Host.cydwr* ファイルをダブルクリックし、ピンを割り当てます。ピンの割り当てについては、デバイス データシートまたはサイプレス キット用のキット ユーザー ガイドを参照ください。**Figure 29** は **CY8CKIT-042** 用のピン割り当てを示します。



Name	Port	Pin	Lock
\I2C:scl\	P4[0]	20	☑
\I2C:sda\	P4[1]	21	☑
\UART:rx\	P0[4]	28	☑
\UART:tx\	P0[5]	29	☑
Pin_Switch	P0[7]	31	☑

Figure 29　I2C_Bootloader_Host プロジェクトのピン割り当て

7. ISR コンポーネントの API ファイルを生成するためにプロジェクトを構築します。コードを割り込みサービスルーチンに追加して、変数 `switch_flag` をセットします。コードは以下のようになります。

```
CY_ISR(ISR_Switch_Interrupt)
{
/* Place your Interrupt code here.
 */
/* `#START ISR_Switch_Interrupt` */
  switch_flag = 1u;
Pin_Switch_ClearInterrupt();
/* `#END` */
}
```

Note: *`switch_flag` は、main.c 内で定義されているため、ISR_Switch.c 内で外部変数として宣言する必要があります。また、ビルド警告を回避するために、ISR_Switch.c ファイル内に `#include <device.h>` を追加します。*

プロジェクト

8. ファームウェアをこのプロジェクトに追加します。I2C_Bootloader_Host プロジェクトは本アプリケーション ノートに添付されています。この関連するプロジェクトの *main.c* ファイルから、ユーザー プロジェクトの *main.c* ファイルにリストされたコードを挿入してください。
   main.c にある `main()`関数は連続的に `switch_flag` 変数をチェックします。フラグをセットすると、ブートローディングが起動されます。*main.c* ファイルは *device.h* 内で定義される `BootloadStringImage()`への関数呼び出しを備えています。この関数はブートローダ ホスト API ファイル (ホスト コア、**Figure 27** を参照) を使用して.*cyacd* ファイルをブートロードします。
   `main()` 関数は「`toggle_appcode`」と呼ばれる別の変数があります。ボタンを押すたびに、「`0`」と「`1`」を交互に繰り返します。これにより、ホストはブートローダブル ファイルを交換に選択します。
9. 前に説明した通りに、ブートローダホストコアは 4 つの API ファイルでビルドされています。これらのファイルは全てのホスト ブートロード動作を行います。ユーザーは、ユーザーのプロジェクト内にこれらのファイルを含める必要があります。これらの API ファイルは次の場所に格納されます。
   < install folder > \ PSoC Creator \ 4.2 \ PSoC Creator \ cybootloaderutils
   Figure 30 に示すように、これらのファイルを追加するために、**Workspace Explorer** に移動し、プロジェクト名を右クリックして、**Add > Existing Item** を選択します。PSoC™ Creator が提供する次のファイルを追加します: cybtldr_api.c / .h、cybtldr_command.c / .h、 cybtldr_parse.c / .h、cybtldr_utils.h。

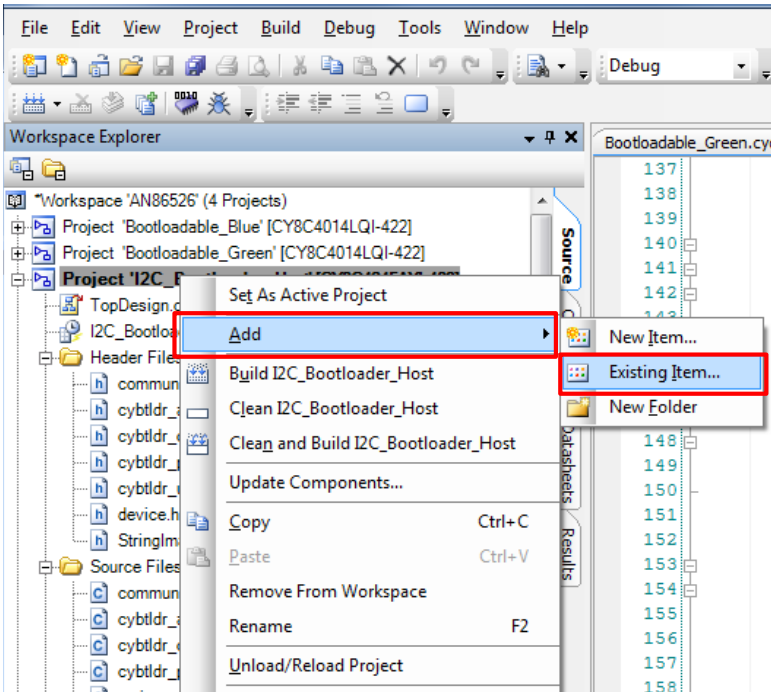


Figure 30　API ファイルの追加

10. API ファイルのブートローディングに加えて、ホストは通信層のサポートも必要です。このサポートは *communication_api.c / .h* ファイルを追加することで提供されます。ユーザーは、本アプリケーション ノートに関連する I2C_Bootloader_Host プロジェクトからこれらのファイルの内容を取り込んで構いません (これらのファイルをプロジェクトに追加するには、前述のステップに従ってください)。本アプリケーション ノートに添付されたプロジェクトからコピーして、これらのファイルを更新します。
11. ブートローダブル ファイルをホストシステムに組み込みます。ブートローダブル ファイルを構築すると、*.cyacd* ファイルが生成されます。ファイルは.*hex* の出力ファイルと似ています。*.cyacd* ファイルの詳細情報については、**付録 B** を参照してください。
    各ラインが配列の要素であるように、このファイルの内容を文字列の配列の形式でコピーします。2

プロジェクト

つのブートローダブル ファイルを持っているので、「StringImage_0」と「StringImage_1」と名付けられた 2 つの配列を定義する必要があります。各配列に対しては、その配列にある行数を格納するマクロを定義します。これらの配列を *StringImage.h* という別のファイルに定義します (文字列を定義する前に、このファイルをプロジェクトに追加する必要があります)。
本アプリケーション ノートに関連する I2C_Bootloader_Host プロジェクト内にある *StringImage.h* ファイルを参照してください。
プロジェクトを構築して、サイプレス キットの PSoC™の中にプログラムします。

3 プロジェクトのテスト

Note: ***Figure 31*** *と* ***Figure 32*** *に示すように、****CY8CKIT-042*** *と* ***CY8CKIT-040*** *キット上の LED は RGB LED です。ここでは、緑色と青色の LED はどちらのブートローダブルが現在実行中であるかを示すために使用されます。*

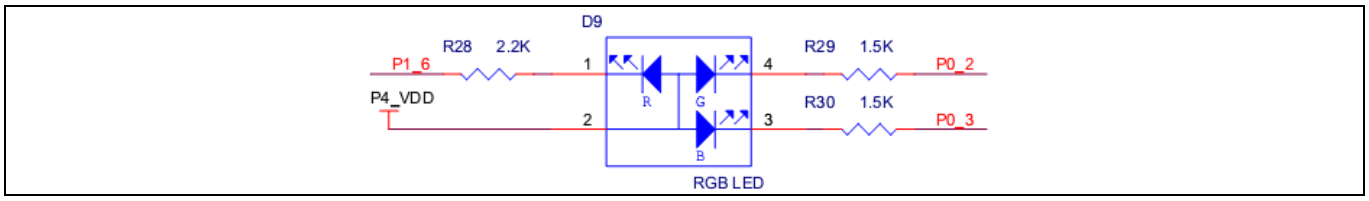


Figure 31 CY8CKIT-042 用の RGB LED の回路図

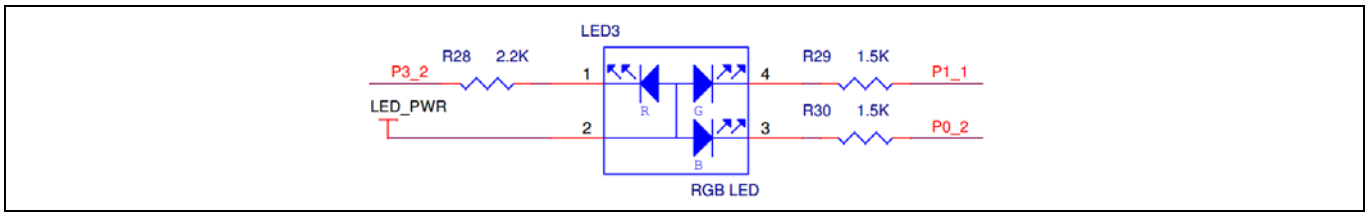


Figure 32 CY8CKIT-040 用の RGB LED の回路図

3.1 キットを設定

プロジェクトをテストするために、次のようにキットを設定します。

ターゲット PSoC™キットの場合、以下の手順に従ってください。

1. ターゲットを USB ケーブル (電源を供給) で PC に接続します。
2. I2C_Bootloader_Red プロジェクトで PSoC™ターゲットをプログラムします。

ホスト PSoC™キットの場合、次の追加ステップを実行してください。

1. ホストを USB ケーブル (電源を供給) で PC に接続します。
2. I2C_Bootloader_Host プロジェクトでホスト PSoC™をプログラムします。
3. PC の Tera Term などの任意のシリアル ポート ビューワを開いて、ブートローディングの情報を表示します。

例えば、ホスト **CY8CKIT-042** キットの場合、PSoC™ 4 内で割り当てられた Rx (P0[4]) と Tx (P0[5]) のピンを拡張ヘッダ J8 にあるピン 10 とピン 9 へ接続します。詳細については、**CY8CKIT-042 PSoC™ 4 Pioneer Kit Guide** を参照してください。I2C_Bootloader_Host プロジェクトでプログラムし、シリアル ポート ビューワを開いてブートローディングの情報を表示します。

ホスト キットとターゲットの接続の場合、以下の手順に従ってください。

1. ホスト キットの SCL と SDA のピンをターゲット キットのそれぞれの SCL と SDA のピンに接続します。
2. キットのグランド ピンを共に短絡します。

例えば、**CY8CKIT-042** ホストおよび **CY8CKIT-042** ターゲットの場合、**Figure 33** に示すように、ホストの P4[0] (SCL) と P4[1] (SDA) のピンをターゲットのそれぞれの P3[0] (SCL) と P3[1] (SDA) のピンに接続しま

プロジェクトのテスト

す。**CY8CKIT-042** ホストおよび **CY8CKIT-040** ターゲットの場合は、**Figure 34** に示すように、ホストの P4[0] (SCL) と P4[1] (SDA) のピンをターゲットのそれぞれの P1[2] (SCL) と P1[3] (SDA) のピンに接続します。キットのグランド ピンを共に短絡します。

Figure 33 と **Figure 34** にこれらの接続を示します。

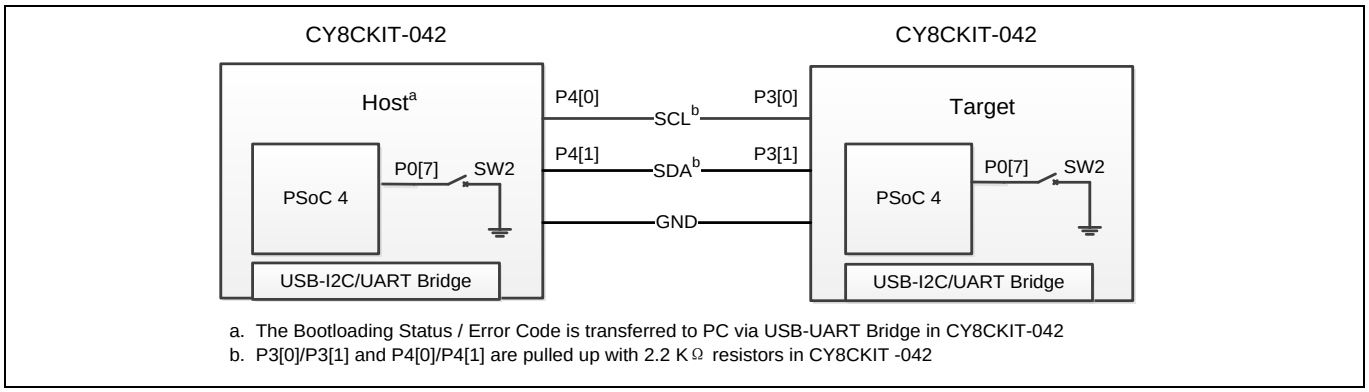


Figure 33　ホスト／ターゲットの接続 (CY8CKIT-042 のターゲット)

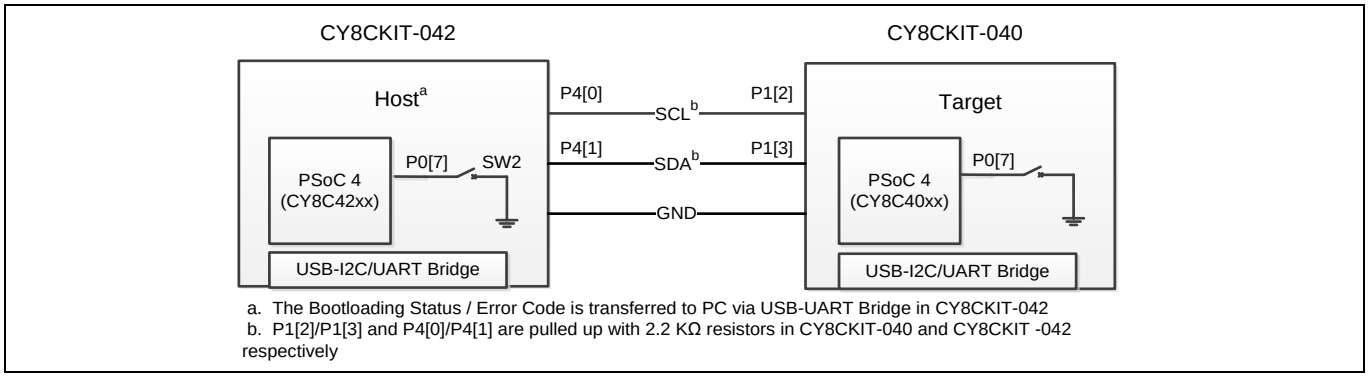


Figure 34　ホスト／ターゲットの接続 (CY8CKIT-040 のターゲット)

3.2　結果の検証

キットが設定された後、用例のプロジェクトを以下のようにテストできます。

- ホスト キット上のボタン (**CY8CKIT-042** 用の P0[7]) を最初に押すと、*Bootloadable_GreenLED.cyacd* ファイルはターゲット PSoC™にブートロードされます。成功して完了すると、「Bootloaded. Led blinks with green color on Target」のメッセージがシリアル ポート ビューワで表示され、ターゲットの緑色 LED が点滅します。
- 次のブートローディング動作のために、ターゲット キットのボタン (**CY8CKIT-042** の場合 P0[7]) を押します。ボタンを押したのを検出した時に PSoC™はブートローダに入ります。PSoC™ 4 はブートローダを実行している時に赤色の LED が点滅します。
- ホスト キット上のボタンを次に押すと、*Bootloadable_BlueLED.cyacd* ファイルはターゲットの PSoC™ にブートロードされます。成功して完了すると、「Blue LED blinks on the target」のメッセージがシリアル ポート ビューワで表示され、ターゲットの青色 LED が点滅します。

4 まとめ

本アプリケーション ノートは I²C を通信インターフェースとして使用して、PSoC™をブートロードする方法について説明しました。また、ブートローダホストの基本的な構成ブロックも紹介し、組み込み I²C ブートローダ ホストの構築方法を示しています。

ブートローダはフィールドのアップグレードを実施するための標準的な方法です。ユーザーにとっては、全ての設定を行う PSoC™ Creator で PSoC™用にブートローダを簡単に作成できます。

詳細情報については、**付録**および **PSoC™ 4** のテクニカル リファレンス マニュアルをご覧ください。

5 関連アプリケーション ノート

ブートローダとフラッシュのプログラミングについてよりよく理解できるように、次のアプリケーション ノートを参照してください。

- **AN73854** – PSoC™ 3, PSoC™ 4, and PSoC™ 5LP Introduction to Bootloaders
- **AN60317** – PSoC™ 3 and PSoC™ 5LP I^2C Bootloader
- **AN68272** – PSoC™ 3, PSoC™ 4, and PSoC™ 5LP UART Bootloader
- **AN84858** – PSoC™ 4 Programming Using an External Microcontroller (HSSP)
- **AN61290** – PSoC™ 3 and PSoC™ 5LP Hardware Design Considerations
- **AN79953** – Getting Started with PSoC™ 4

PSoC™ 4 の他の多くの特長と機能に関わるアプリケーション ノート全ての一覧は、ここをクリックしてください。

6 関連のプロジェクト

このアプリケーション ノートに添付されたプロジェクトは、**Table 5** の通りにまとめられています。

Table 5 本アプリケーション ノートに添付されているプロジェクト

設計プロジェクト名	説明
I2C_Bootloader_Red	このプロジェクトは PSoC™用の I2C ブートローダ プロジェクトを PSoC™ Creator を使用して作成する方法について説明する。このブートローダは CY8CKIT-042*キットの赤色 LED を点滅させる
Bootloadable_Green	このプロジェクトは PSoC™用のブートローダブル プロジェクトを PSoC™ Creator を使用して作成する方法について説明する。このアプリケーションでは CY8CKIT-042*キットの緑色 LED を点滅させる
Bootloadable_Blue	このプロジェクトはブートローダブル プロジェクトの作成方法、およびブートローダブル プロジェクトからブートローダに入る方法について説明する。このアプリケーションでは CY8CKIT-042*キットの青色 LED を点滅させる
I2C_Bootloader_Host	これはブートローダ ホスト プログラム用例。この用例は PSoC™が別の PSoC™をブートロードすることを説明する

* プロジェクトは、簡単に別のキットで動作するように適合させることができます。

7 PSoC™リソース

サイプレスは、**www.cypress.com** に大量のデータを掲載しており、ユーザーがデザインに適切なPSoC™デバイスを選択し、デバイスをデザインに迅速で効果的に統合する手助けをしています。リソースの包括的なリストについては、「**KBA86521, How to Design with PSoC™ 3, PSoC™ 4, and PSoC™ 5LP**」を参照ください。以下は PSoC™ 4 のリソースの要約です。

- **概要: PSoC™ポートフォリオ**、**PSoC™ロードマップ**
- **製品セレクタ: PSoC™ 1**、**PSoC™ 3**、**PSoC™ 4**、または **PSoC™ 5LP**。さらに、**PSoC™ Creator** にはデバイス選択ツールを含んでいます。
- **データシート**では、**PSoC™ 4000**、**PSoC™ 4000S**、**PSoC™ 4100**、**PSoC™ 4100S**、**PSoC™ 4100PS**、**PSoC™ 4100S Plus**、**PSoC™ 4200**、**PSoC™ 4xx7 BLE**、**PSoC™ 4200-M**、および **PSoC™ 4200-L** デバイスファミリ用の電気的仕様をご提供し、説明いたします。
- **CapSense デザイン ガイド**: PSoC™ 4 ファミリのデバイスを使用して静電容量タッチセンス アプリケーションを設計する方法について説明します。
- **アプリケーション ノートおよびサンプルコード**は基本的なレベルから高度なレベルまでの幅広いトピックに触れています。アプリケーション ノートの多くはサンプルコードを含んでいます。PSoC™ Creator はより多くのサンプルコードをご提供します。詳細については「**サンプルコード**」を参照ください。
- **テクニカル リファレンス マニュアル (TRM)**: 各 PSoC™ 4 デバイス ファミリのアーキテクチャとレジスタの詳細な説明をします。
- **開発キット**:
  - **CY8CKIT-040**、**CY8CKIT-041**、**CY8CKIT-042**、**CY8CKIT-042-BLE**、**CY8CKIT-044**、および **CY8CKIT-046** Pioneer Kit は使いやすく、安価な開発プラットフォームです。これらのキットには、Arduino™準拠シールドおよび Digilent® Pmod™ドーターカード用コネクタを搭載しています。
  - **CY8CKIT-001** は、すべての PSoC™ファミリ デバイスの共通開発プラットフォームです。
  - **CY8CKIT-147** と **CY8CKIT-149** は、PSoC™ 4 デバイスをサンプリングするための低コスト プロトタイプ プラットフォームです。
- **MiniProg3** デバイスはフラッシュのプログラムとデバッグ用のインターフェースを提供します。

7.1 PSoC™ Creator

PSoC™ Creator は Windows ベースの統合設計環境 (IDE) です。無料で利用できます。このキットにより、PSoC™ 3、PSoC™ 4 および PSoC™ 5LP ベースのシステムについて、ハードウェアとファームウェアの同時並行の設計が可能です (**Figure 35** を参照ください)。PSoC™ Creator により、以下のことが可能です。

1. コンポーネントをドラッグ アンド ドロップして、メインデザイン ワークスペースでハードウェア システム デザインを構築
2. PSoC™ハードウェアとアプリケーション ファームウェアを同時設計
3. コンフィギュレーション ツールを用いて、コンポーネントを構成
4. 100 以上のコンポーネントを含むライブラリを利用
5. コンポーネント データシートの閲覧

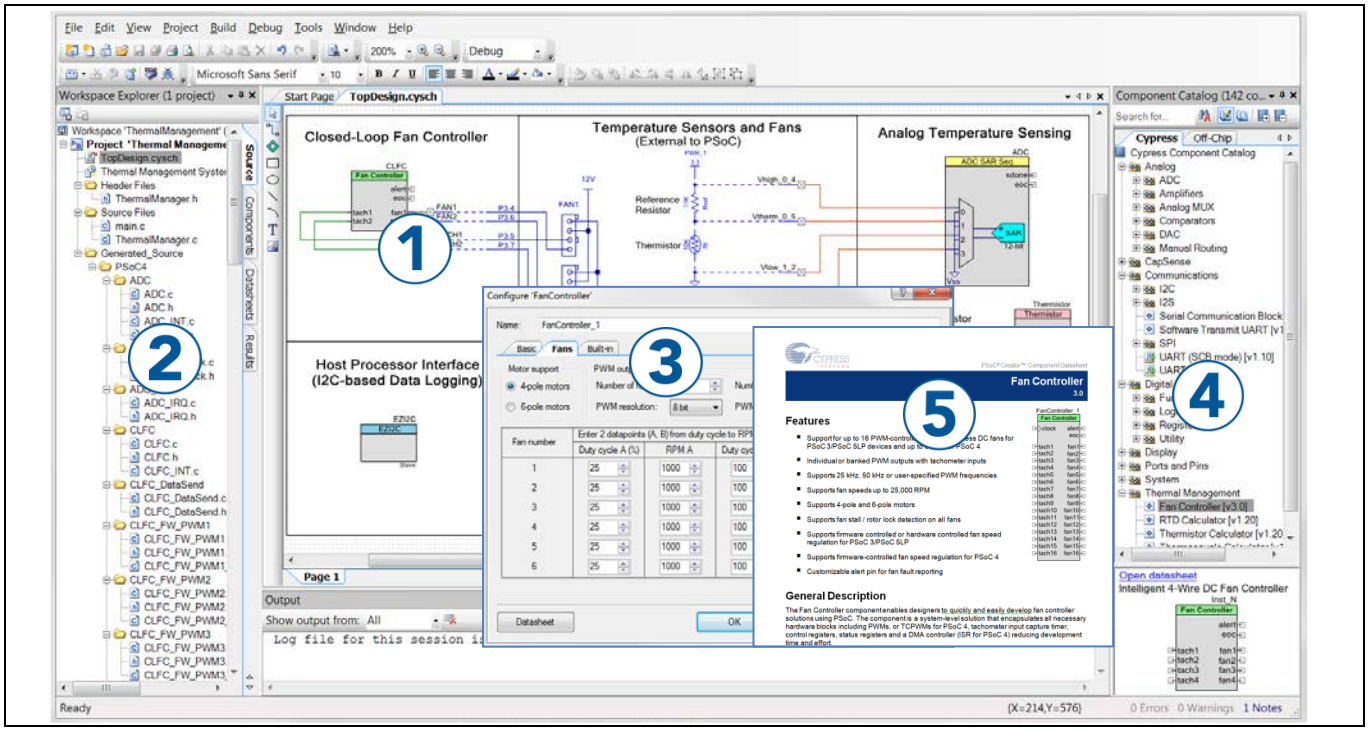


Figure 35 PSoC™ Creator の特長

7.2 サンプルコード

PSoC™ Creator には多数のサンプルコードのプロジェクトが含まれています。これらのプロジェクトは **Figure 36** に示すように、PSoC™ Creator のスタート ページからアクセスできます。

用例のプロジェクトにより、空のページではなく出来上がった設計を元に始めるため、設計時間を短縮することができます。用例のプロジェクトは PSoC™ Creator コンポーネントを様々なアプリケーションに使用する方法も示します。**Figure 36** に示すように、サンプルコードおよびデータシートが含まれています。

Figure 37 に示す「Find Example Project」ダイアログには、いくつかのオプションがあります。

- アーキテクチャまたはデバイス ファミリ (PSoC™ 3、PSoC™ 4、PSoC™ 5LP) またはカテゴリやキーワードに基づいて用例のプロジェクトをフィルターします。
- **Filter Options** に基づいて提供された用例のプロジェクトのメニューから選択します。
- 選択のためにデータシートをレビューします (**Documentation** タブ上で)。
- 選択のためにサンプルコードをレビューします。コード開発時間を短縮させるために、このウィンドウからコードをコピーしプロジェクトに貼り付けることができます。
- 選択したものをベースに新規プロジェクト (また必要な場合は新規ワークスペース) を作成します。完成した基本的な設計から始めることで設計時間を短縮させます。そのあと設計をアプリケーションに適合させることができます。

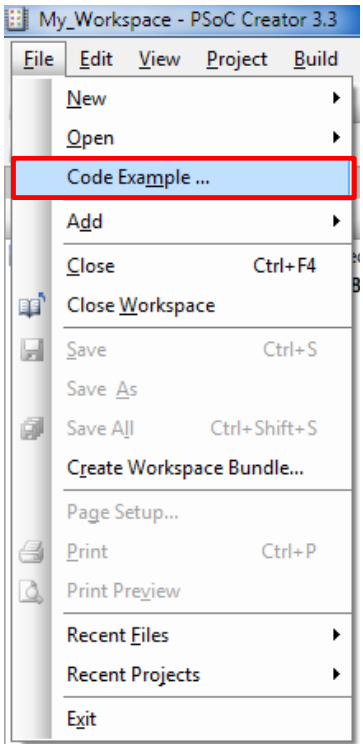


Figure 36 PSoC™ Creator のサンプルコード

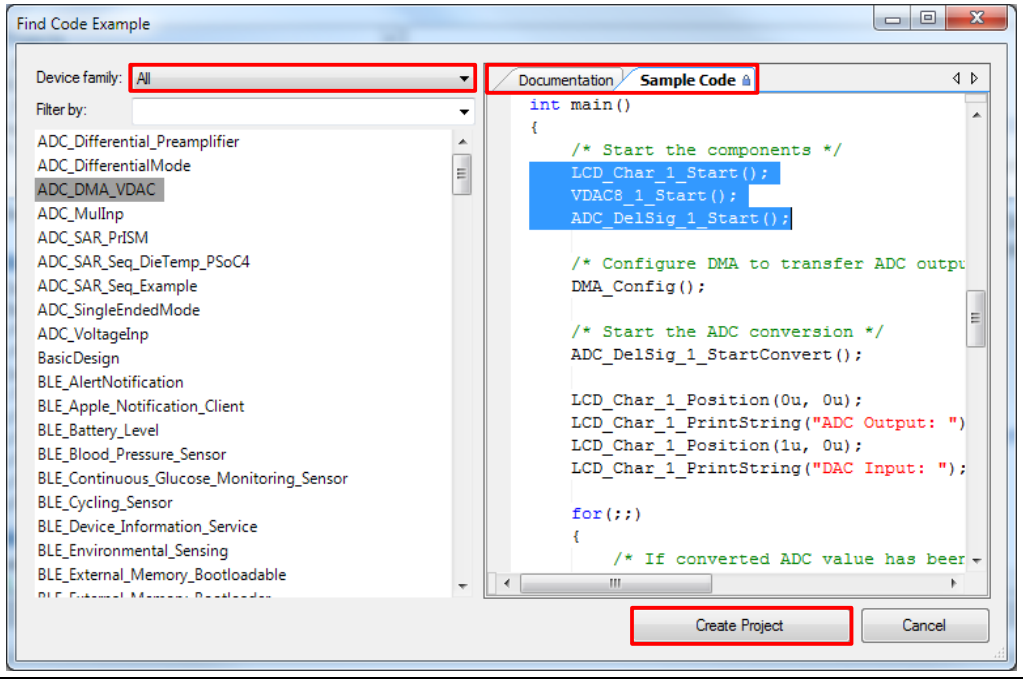


Figure 37 サンプルコードを含む用例のプロジェクト

7.3 PSoC™ Creator ヘルプ

PSoC™ Creator ホームページへアクセスし、PSoC™ Creator の最新版をダウンロードしてください。次に、PSoC™ Creator を起動して以下のアイテムへナビゲートします。

- **クイック スタート ガイド**: **Help** > **Documentation** > **Quick Start Guide** を選択します。このガイドは PSoC™ Creator プロジェクトを開発するための基礎知識を提供します。
- 簡単なコンポーネント用例のプロジェクト: **File** > **Open** > **Example projects** を選択します。これらの用例プロジェクトは、PSoC™ Creator のコンポーネントの設定と使用方法を説明します。
- **Starter designs (初級者向けの設計)**: **File > New** > **Project** > **PSoC 4 Starter Designs** を選択します。これらの初級者向けの設計は、PSoC™ 4 のユニークな機能を説明します。
- **システム リファレンス ガイド**: **Help** > **System Reference** > **System Reference Guide** を選択します。このガイドは PSoC™ Creator により提供されるシステム機能を一覧で説明します。
- **コンポーネント データシート**: コンポーネントを右クリックして「データシートを開く」を選択します。全ての PSoC™ 4 のコンポーネント データシートの一覧を表示するには、**PSoC™ 4 のコンポーネント データシート**ページへアクセスしてください。
- **ドキュメント マネージャー**: PSoC™ Creator が提供するドキュメント マネージャーにより、ドキュメント リソースを容易に検索し、閲覧することができます。ドキュメント マネージャーを開くには、メニューアイテムの **Help** > **Document Manager** を選択します。

7.4 テクニカル サポート

ご質問には弊社のテクニカル サポート チームが対応させて頂きますので、お気軽にご連絡ください。**Cypress Technical Support** ページにアクセスし、サポート リクエストを作成してください。

米国のお客様は、テクニカル サポート チームに連絡する際、以下の電話番号 (通話無料) にお問い合わせください: +1-800-541-4736 プロンプトでオプション「8」を選択してください。

緊急サポートが必要な場合は、以下のサポート リソースをご利用ください。

- **セルフ ヘルプ**
- **所在地の販売代理店**

8 付録 A: メモリ

8.1 フラッシュメモリの詳細

フラッシュ メモリはファームウェア、バルク データ、デバイスの設定データ、工場出荷時の設定データおよびユーザー定義のフラッシュ プロテクション データの記憶領域を提供します。**Figure 38** は PSoC™にあるフラッシュ メモリの物理的な構造を示します。

PSoC™のフラッシュ メモリは「アレイ」と呼ばれるブロックに分けられます。アレイはアレイ ID によって個別に識別されます。それぞれのアレイには、フラッシュ メモリの 128 個または 256 個の列があります。各行には、PSoC™ 4100、4200、4000S と 4100PS ファミリ デバイス用の 128 データ バイト、4100S ファミリ デバイス用の 256 データ バイトおよび 4000 ファミリ デバイス用の 64 データ バイトがあります。 これにより、アレイは命令とデータの記憶領域用に 8KB、16KB 、32KB または 64KB を持つことができます。

PSoC™ 4100、4200、4000S と 4100PS デバイス ファミリは、最大フラッシュ 32KB を備え、これは 1 つだけのアレイであり、唯一の有効なアレイ ID は 0 です。PSoC™ 4100S デバイスは、最大フラッシュ 64KB を備え、これも 1 つだけのアレイであり、唯一の有効なアレイ ID は 0 です。また PSoC™ 4000 デバイスは最大フラッシュ 16KB を備え、これも 1 つだけのアレイで、唯一有効なアレイ ID も 0 です。

フラッシュ メモリは 1 回で 1 列がプログラムされます。1 回で 1 列が消去でき、また同時にフラッシュ メモリ全体を消去できます。列はアレイ ID と列番号のユニークな組み合わせで識別されます。

Figure 38 は、フラッシュ メモリの最初の X 列はブートローダで占められていることも示します。X は、以下の項目に十分な空間があるように設定されます。

- アドレス 0 から始まるブートローダのベクタ テーブル
- ブートローダ プロジェクトの設定バイト
- ブートローダ プロジェクトのコードとデータ
- フラッシュのブートローダ部のチェックサム

PSoC™の場合、ベクタ テーブルは、ブートローダ プロジェクトの初期スタック ポインタ (SP) 値、およびブートローダ プロジェクト コード用の開始アドレスを含みます。また、ブートローダが使用する、例外および割り込み用のベクタも含みます。

ブートローダブル プロジェクトは、ブートローダの次にある最初の 128 バイト境界から始まるフラッシュ領域を占有しています。このフラッシュ メモリ領域は以下の項目を含みます。

- ブートローダブル プロジェクトのベクタ テーブル
- ブートローダブル プロジェクトのコードとデータ

フラッシュ メモリの最上位の 64 バイト ブロックは、2 つのプロジェクトの共有記憶領域として使用されます。このブロックに保存されるパラメーターは以下の通りです。

- ブートローダブル プロジェクトのフラッシュ メモリ内のエントリ アドレス (4 バイトのアドレス)
- ブートローダブル プロジェクトが占有するフラッシュ メモリ量 (フラッシュの列数)
- フラッシュ メモリのブートローダブル ブロックのチェックサム (1 バイト)
- フラッシュ メモリのブートローダブル ブロックのバイト単位のサイズ (4 バイト)。

フラッシュ メモリ内のメタデータの場所にある詳細情報は、**フラッシュ メモリ内のメタデータ レイアウト**を参照してください。

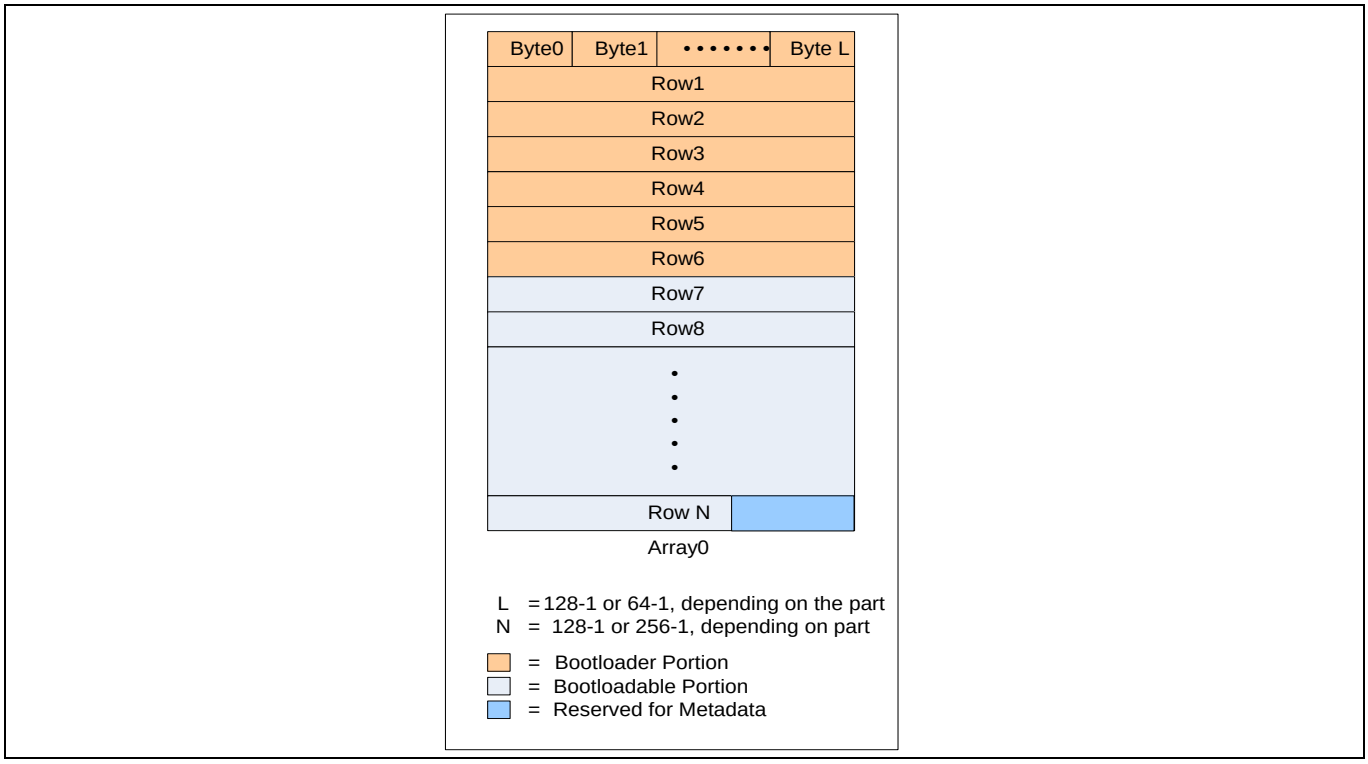


Figure 38 PSoC™内のフラッシュ メモリの物理的な構造

8.2 PSoC™のメモリ使用量

ブートローダ プロジェクトは 2 種類あります。それは標準ブートローダとマルチアプリケーション ブートローダです。マルチアプリケーション ブートローダは、実行可能な有効なアプリケーションが常に存在する保証が必要な設計に役立ちます。しかし、この保証には、各アプリケーションのフラッシュ メモリ領域を半分しか使用できない制限があります。

Figure 39 は PSoC™ Creator プロジェクトの各種類に対するフラッシュ メモリ使用量を示します。

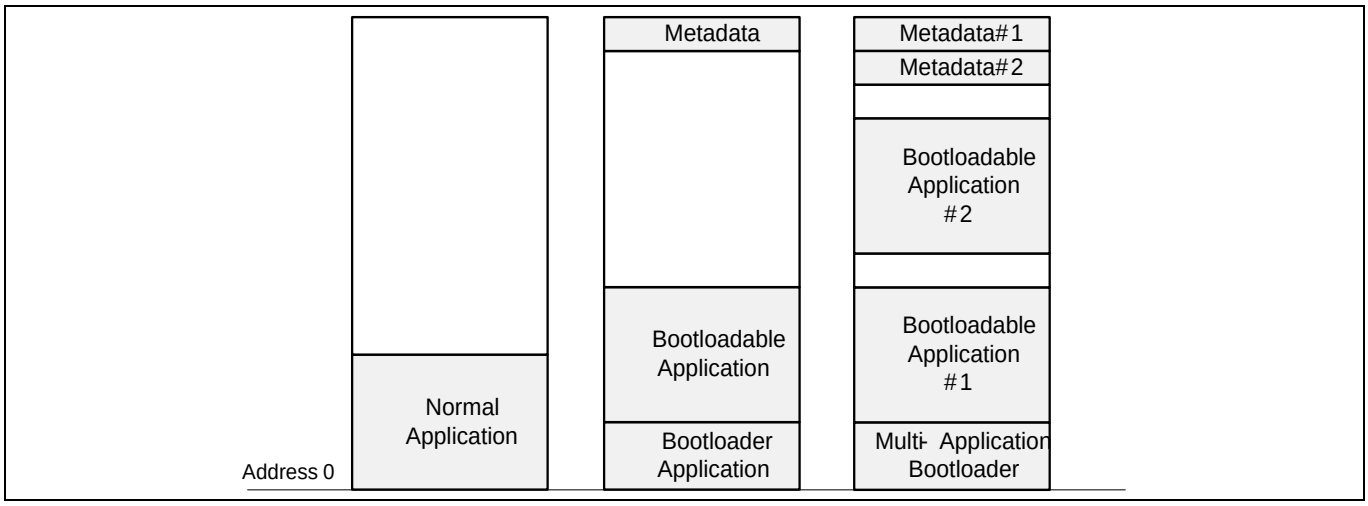


Figure 39 フラッシュ メモリの使用量

8.3 フラッシュ メモリ内のメタデータの配置

Figure 39 に示すように、メタデータの部分は、ブートローダとブートローダブル プロジェクトの共有記憶領域として使用されるフラッシュ メモリ内の最上位 64 バイトのブロックです。Table 6 に示すように、様々なパラメーターがこのブロックに格納されます。マルチアプリケーション ブートローダには 2 セットのメタデータがあります。

Table 6 メタデータの配置

アドレス	PSoC™
0x00	アプリケーション チェックサム
0x01	アプリケーション アドレス
0x02	
0x03	
0x04	
0x05	最後のブートローダ列
0x06	
0x07	
0x08	
0x09	アプリケーションの長さ
0x0A	
0x0B	
0x0C	
0x0D	該当なし
0x0E	該当なし
0x0F	該当なし
0x10	アプリケーション アクティブ
0x11	アプリケーション ベリファイ
0x12	ブートローダ アプリケーションのバージョン
0x13	
0x14	ブートローダブル アプリケーションの ID
0x15	
0x16	ブートローダブル アプリケーションのバージョン
0x17	
0x18	ブートローダブル アプリケーションのカスタム ID
0x19	
0x1A	
0x1B	
0x1C-0x3F	該当なし

Note: マルチアプリケーション ブートローダでは、メタデータ用の最後のブートローダ列(イメージ 2) はブートローダ列ではなく、フラッシュ メモリにあるブートローダブル 1 の最後の列を示します。

8.3.1 フラッシュ メモリの保護

ブートローダ コードが無効になると、製品が使用できなくなります。そのため、フラッシュ メモリのブートローダを不慮の上書きから保護する必要があります。

PSoC™の保護はチップレベルの保護 (オープン、プロテクテッド、キル) およびフラッシュ レベルの保護があります。詳細については、デバイス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) を参照してください。フラッシュ メモリ保護の機能は、ユーザー独自のコードの複製やリバース エンジニアリングを防止するように設計されています。しかし、これはフラッシュ メモリのブートローダ部への予期しない書き込みから守るためにも使用されます。

Table 7 に示すように、フラッシュ レベル保護は 2 つのレベルに分けられます。フラッシュ メモリの各列は PSoC™ Creator (*.cydwr* ファイルのフラッシュ セキュリティ タブ) を使用してセットすることで、それぞれ異なる保護レベルを持つように設定できます。

Table 7 PSoC™のフラッシュ メモリ保護レベル

保護レベル	許可	不可
非保護	外部読み出しおよび書き込み 内部読み出しおよび書き込み	–
完全保護	内部読み出し 外部読み出し	外部書き込み 内部書き込み

フラッシュ メモリのブートローダ部が「完全保護」の保護レベルに設定されると、その設定を変更できません。保護レベルまたはブートローダ コードを変更する方法は、フラッシュ メモリを全て消去して、SWD インターフェースを使って再プログラムすることのみです。

以下はブートローダのフラッシュ メモリ保護の例です。

8.3.2 フラッシュ メモリ保護の例

ブートローダ プロジェクトが構築されると、PSoC™ Creator 出力ウィンドウは使用したフラッシュ メモリ量を示します。例えば、I2C_Bootloader プロジェクトが占めるフラッシュ メモリが 4352 バイトの場合、 (32KB フラッシュの PSoC™の場合) 出力は以下の通りです。

```
Flash used: 4352 of 32768 bytes (13.3%).
```

この例では、ブートローダはフラッシュ メモリの 34 列 (4352 ／128) を占めます。すなわち、フラッシュ メモリ内の場所は、0x0000 ～ 0x10FF です。(PSoC™ Creator にある*.cydwr* ファイルのフラッシュ セキュリティ タブを通して) これらの列のフラッシュ メモリ保護レベルを「完全保護」に設定します。残りの列の保護レベルは Figure 40 に示すように非保護 (デフォルト) にできます。フラッシュ メモリ保護ダイアログの使用方法の詳細については、PSoC™ Creator のヘルプ記事「Flash Security Editor」を参照してください。

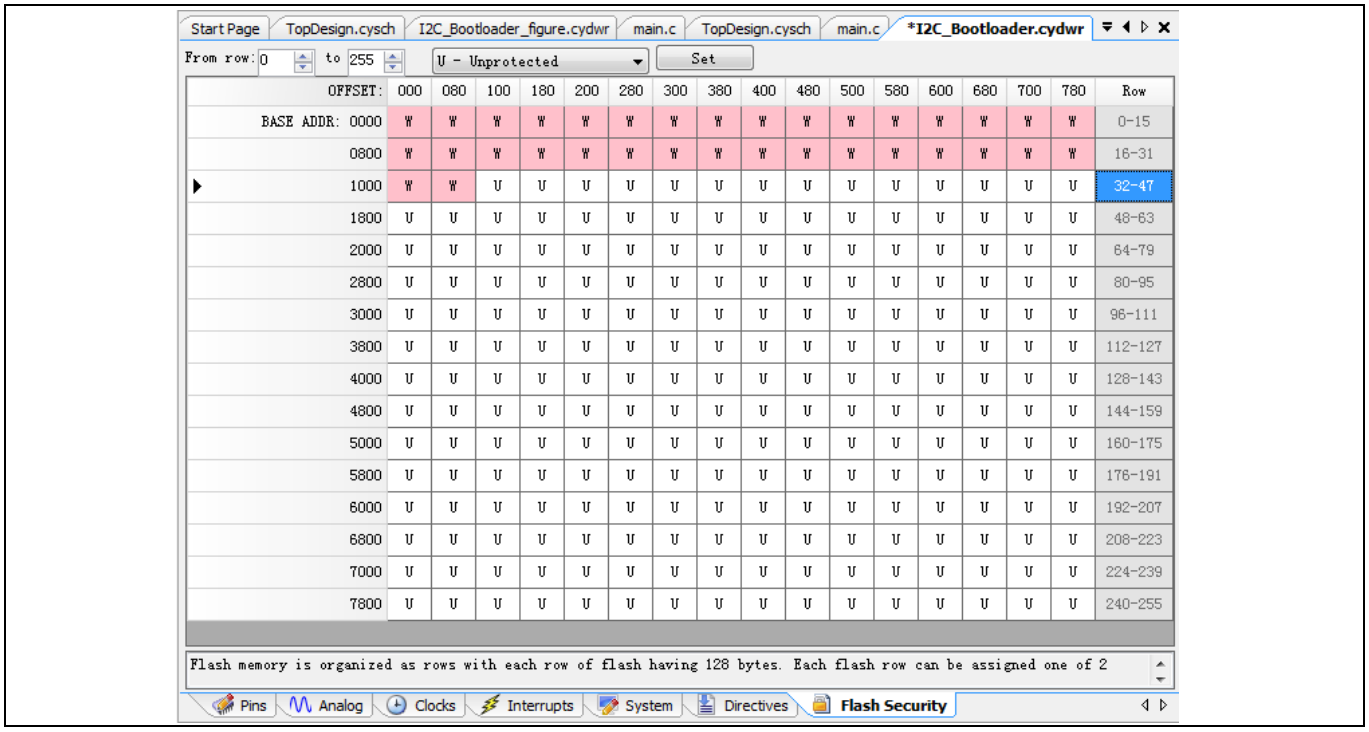


Figure 40 PSoC™ Creator 内のフラッシュ メモリ保護

9 付録 B: プロジェクト ファイル

9.1 ブートローダブルの出力ファイル

PSoC™ Creator プロジェクトが構築されると、*.hex* タイプの出力ファイルが生成されます。このファイルは、SWD インターフェースを使ったプログラミング時に PSoC™へダウンロードされるファイルです。

ブートローダブル プロジェクトの場合、*.hex* ファイルはブートローダブル プロジェクトと関連するブートローダ プロジェクトが結合された*.hex* ファイルです。通常は、プロダクション環境でこのファイルを使用して、両方のプロジェクトを SWD 経由でダウンロードします。

9.1.1 *.cyacd のファイル フォーマット

ブートローダブル プロジェクトが構築されると、*.cyacd* タイプ (アプリケーション コードとデータ) の追加ファイルも生成されます。このファイルには、ヘッダがあり、その後にフラッシュ メモリのデータ行が続きます。ヘッダを除くと、.cyacd ファイルの各行は、それぞれフラッシュ メモリのデータの 1 列分を示します。データは、ビッグエンディアン形式の ASCII データとして保存されます。従って、ブートロードの時、このファイルの内容は構文解析する (ASCII から hex に変換される) 必要があります。構文解析は*.hex* タイプのファイルをプログラムする時には必要ありません。

このファイルのヘッダは次のフォーマットになっています。

```
[4 bytes Silicon ID] [1 byte Silicon rev] [1 byte checksum type]
```

フラッシュ メモリの列は次のフォーマットになっています。

```
[1 byte array ID] [2 bytes row number] [2 bytes data length] [N bytes of
data] [1 byte checksum]
```

ヘッダ内のチェックサムタイプは、ブートロード処理中にブートローダのホストとブートローダとの間で送信するパケット用に使用するチェックサムのタイプを示します。このバイトが 0 の場合、チェックサムは単純な合計値です。このバイトが 1 の場合、チェックサムは CRC-16 です。

10 付録 C: ホスト／ターゲットの通信

10.1 通信フロー

ブートローダ ファンクション フローの節では、PSoC™内のブートローダ動作を説明し、I2C ブートローダのホストの節では、ブートローダ ホストの構成ブロックを紹介しています。これに基づいて、Figure 41 はブートローディング中のホストとターゲット間の通信フローについて説明します。これは、コマンドがターゲットに発行され、応答が受信される順序を示します。ブートロード コマンドの完全なリスト、そのコードと期待される応答については、コマンドとステータス／エラーコードを参照してください。

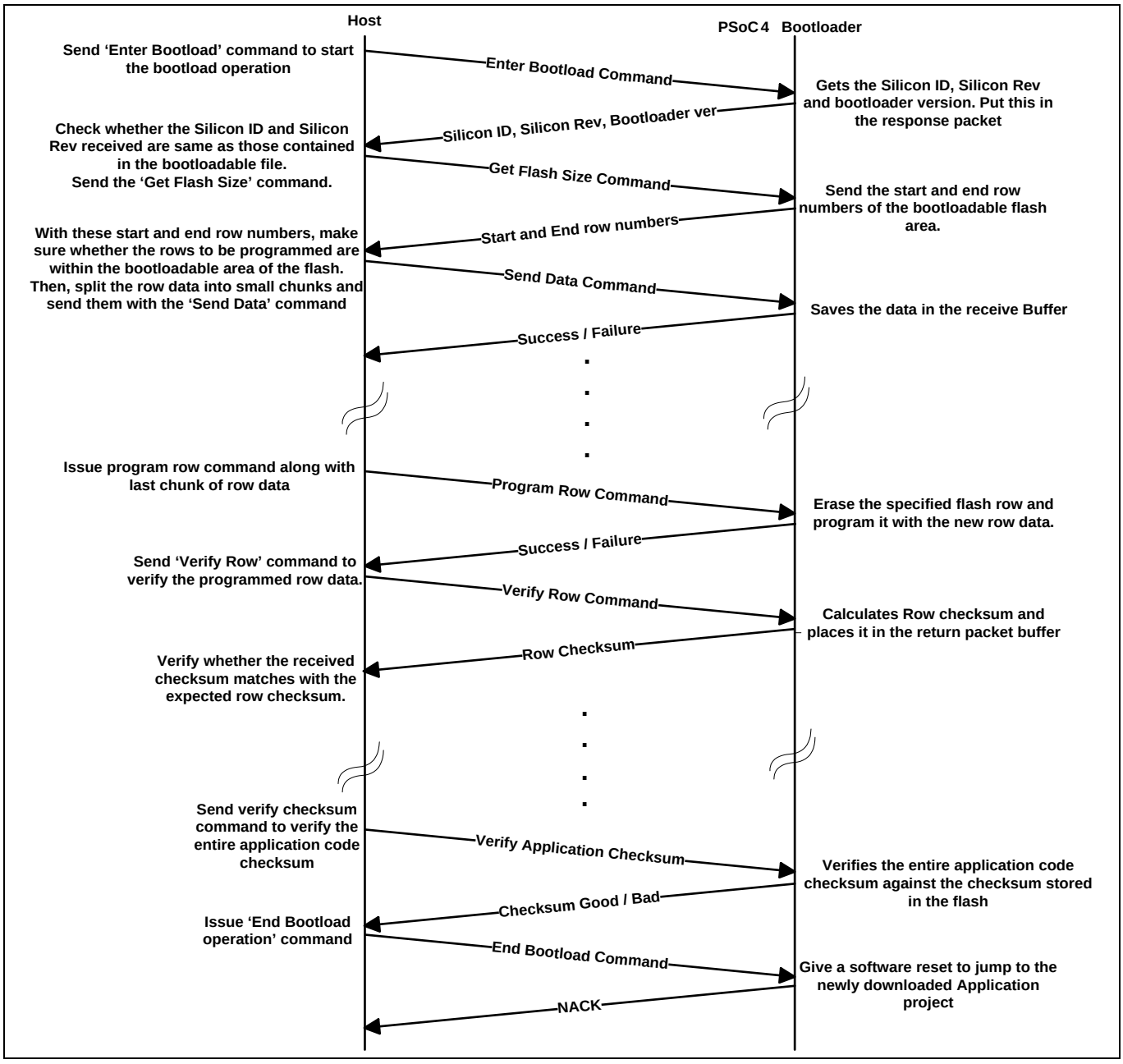


Figure 41 ブートローディング中の通信フロー

10.2 プロトコル パケットのフォーマット

ブートロード動作はホストとターゲット間のコマンドと応答パケットの交換を含みます。これらのパケットは **Figure 42** に示すように、特定のフォーマットを持っています。

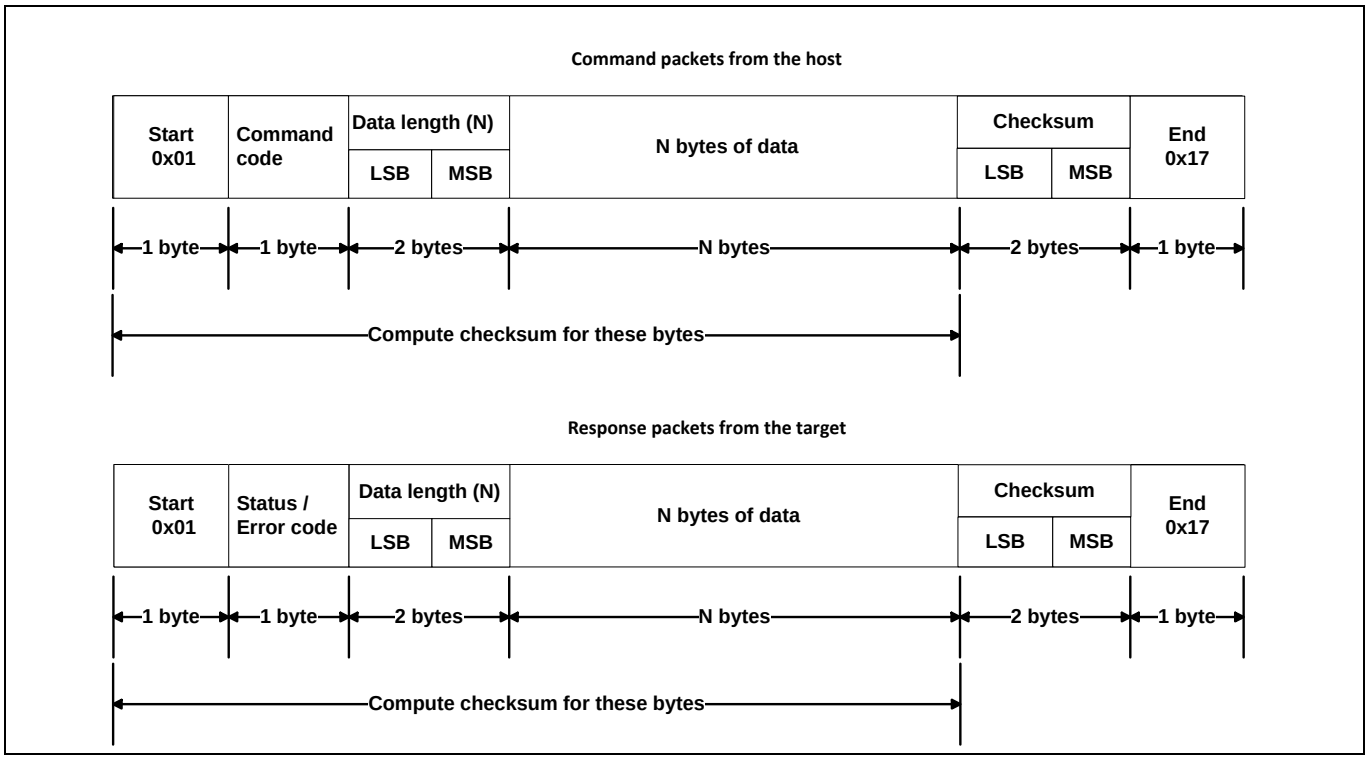


Figure 42 ブートロードパケットのフォーマット

それぞれのパケットはチェックサムバイトを含みます。チェックサムは、ブートローダ プロジェクトの設定によって、単純合計 (2 の補数) または CRC-16 のいずれかにできます。データ長とチェックサムなどのマルチバイト データを送信する時、最下位バイトが先に送信されます。

ブートローダは応答パケットでホストからのコマンドに応答します。応答パケットのフォーマットは、コマンド コードの代わりにステータス／エラーコードがあることを除いて、コマンド パケットのフォーマットと似ています。重要なコマンドとデータバイトおよびブートローダ応答パケット データは 44 ページの **Table 8** に示されています。

10.3 ブートローダ ホスト用の I²C トランザクション情報

I²C トランザクションは、2 つのデバイス間通信のより低い階層です。ホストから PSoC™の読み出しと書き込みフォーマットは、**Figure 43** と **Figure 44** に表示されます。

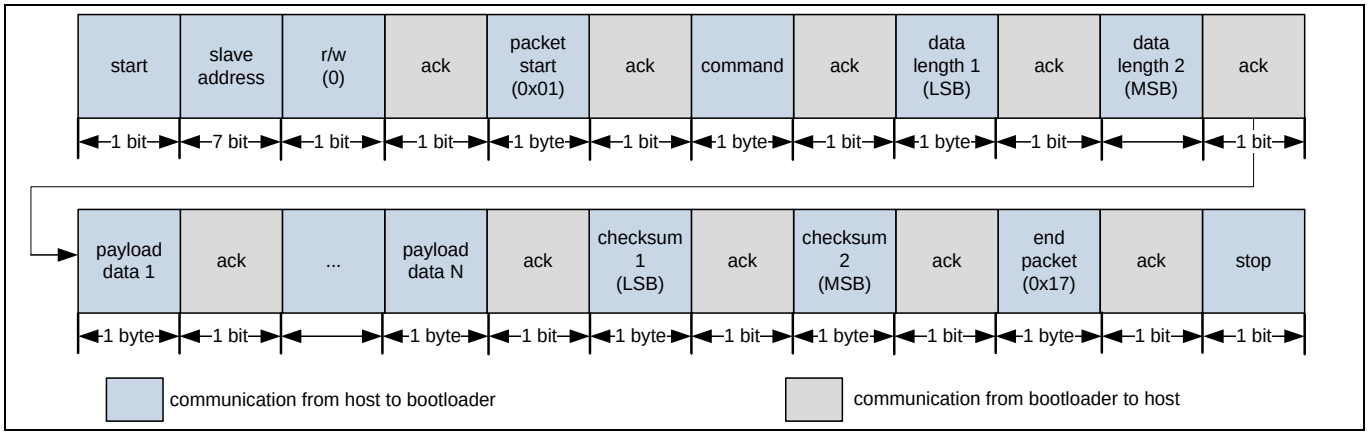


Figure 43 I²C インターフェースを介してホストからブートローダに送信されるトランザクション (書き込み)

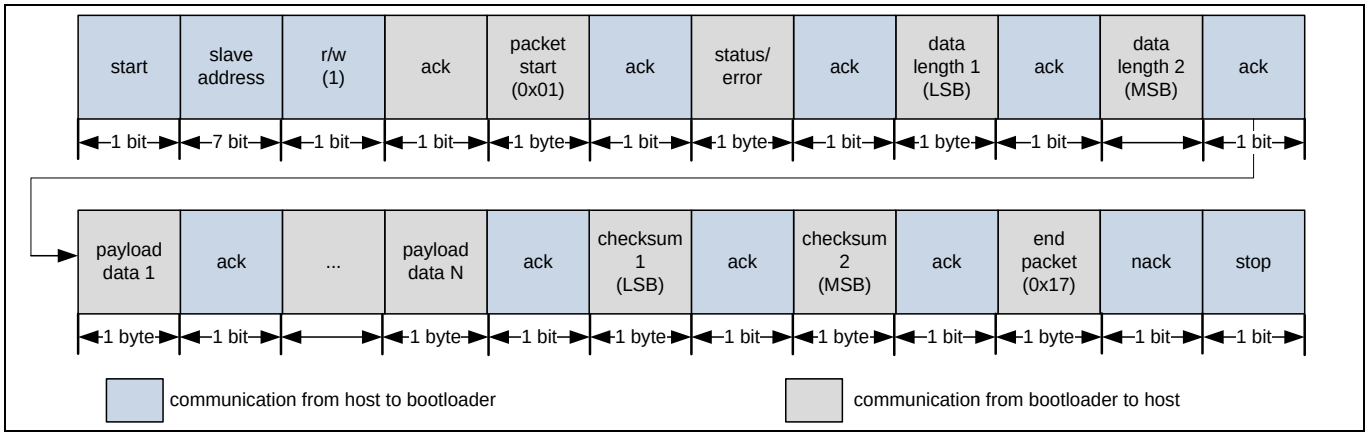


Figure 44 I²C インターフェースを介してブートローダから受信されるトランザクション (読み出し)

10.3.1 コマンドとステータス／エラーコード

前の節で説明した通り、コマンドと応答パケットの構造は同じです。唯一の違いは、第 2 バイトがコマンド コードかステータス／エラー コードを含むかどうかです。

Table 8 は、コマンドのリストとその期待される応答の一覧です。46 ページの **Table 9** はステータスとエラーコードの一覧です。

Table 8 ブートローディング コマンド

コマンド バイト	コマンド	コマンド パケット内のデータ バイト	期待する応答データ バイト
0x31	Verify Checksum (チェックサムの検証)	該当なし	1 バイト:「0」またはゼロ以外 ゼロ以外のバイトの場合、アプリケーション チェックサムが一致し、それは有効なアプリケーション ゼロのバイトの場合、チェックサムが間違い、アプリケーションは無効

付録 C: ホスト／ターゲットの通信

コマンド バイト	コマンド	コマンド パケット内のデータ バイト	期待する応答データ バイト
0x32	Get Flash Size (フラッシュ メモリ サイズの取得)	フラッシュ アレイ ID、1 バイト	ブートローダブル フラッシュ領域の最初の列番号、2 バイト ブートローダブル フラッシュ領域の最終の列番号、2 バイト これらの番号は要求されたアレイ ID 用のものである
0x33	アプリケーションステータスを取得 (マルチアプリケーション ブートローダのみに有効)	アプリケーションの番号、1 バイト	有効なアプリケーションの番号、1 バイト アクティブ アプリケーションの番号、1 バイト 指定されたアプリケーションが有効でアクティブかどうかチェック
0x34	Erase Row (列の削除)	フラッシュ アレイ ID、1 バイト フラッシュ メモリ列の番号、2 バイト	適用なし 指定されたフラッシュ メモリ列の内容を消去
0x35	Sync bootloader (ブートローダの同期)	該当なし	該当なし ブートローダをクリーンな状態にリセットします。バッファ内のデータは全て廃棄される。このコマンドは、ブートローダとホストが互いに同期しなくなる場合にのみ必要である
0x36	Set Active Application (アクティブ アプリケーションを設定) (マルチアプリケーション ブートローダにのみ有効)	アプリケーションの番号、1 バイト	該当なし 指定されたアプリケーションをアクティブとして設定
0x37	Send Data (データの送信)	送信される N データバイト	該当なし 受信されたデータ バイトは、プログラム列コマンドを見越してブートローダによってバッファされる
0x38	Enter Bootloader (ブートローダに入る)	該当なし	シリコン ID、4 バイト シリコン リビジョン、1 バイト ブートローダ バージョン、3 バイト 全てのコマンドは、このコマンドを受信するまで無視される

付録 C: ホスト／ターゲットの通信

コマンド バイト	コマンド	コマンド パケット内のデータ バイト	期待する応答データ バイト
0x39	Program Row (列をプログラム)	フラッシュ アレイ ID、1 バイト フラッシュ メモリ列の番号、2 バイト 送信される N データ バイト	該当なし 複数のデータバイトをデータ送信コマンドでブートローダに送信した後、データの最終チャンクはこのコマンドと共に送信される
0x3A	Verify Row (列の検証)	フラッシュアレイ ID、1 バイト フラッシュ メモリ列の番号、2 バイト	列のチェックサム、1 バイト 指定された列のチェックサムを返す
0x3B	Exit Bootloader (ブートローダの終了)	該当なし	該当なし このコマンドは認識されない

Table 9　　ブートローディング ステータス／エラーコード - コマンドへの可能な応答

ステータス／エラーコード	ラベル	説明
0x00	CYRET_SUCCESS	コマンドは正しく受信され、実行された
0x02	BOOTLOADER_ERR_VERIFY	フラッシュのベリファイに失敗した
0x03	BOOTLOADER_ERR_LENGTH	利用可能なデータ量は予想される範囲外
0x04	BOOTLOADER_ERR_DATA	データが正しい形式ではない
0x05	BOOTLOADER_ERR_CMD	コマンドを認識しなかった
0x06	BOOTLOADER_ERR_DEVICE	期待されるデバイスと検出したデバイスが一致しない
0x07	BOOTLOADER_ERR_VERSION	検出したブートローダのバージョンに対応していない
0x08	BOOTLOADER_ERR_CHECKSUM	チェックサムが想定した値と異なる
0x09	BOOTLOADER_ERR_ARRAY	フラッシュ アレイ ID が有効でない
0x0A	BOOTLOADER_ERR_ROW	フラッシュ行の数は有効でない
0x0C	BOOTLOADER_ERR_APP	アプリケーションは有効ではなく、アクティブに設定できない
0x0D	BOOTLOADER_ERR_ACTIVE	アプリケーションは現在アクティブとしてマークされていない
0x0F	BOOTLOADER_ERR_UNK	未知のエラーが発生した

11 付録 D: ホストコア API

11.1 cybtldr_api2.c / .h

これは、ブートロード動作の全てを処理する、より高レベルの API です。ファイルを開く関数とファイルを閉じる関数があります。これはブートロード動作のために、*cybtldr_api.c / .h*API の関数を呼び出します。この API は GUI ベースのブートローダ ホストを構築する時に使用されます。

11.2 cybtldr_parse.c / .h

このモジュールは、デバイスに送信するブートローダブルのイメージを含む*.cyacd* ファイルの構文解析を行います。ファイルへのアクセス設定、ヘッダの読み出し、列のデータの読み出し、およびファイルを閉じるための関数も持っています。

11.3 cybtldr_api.c / .h

これは 1 回に 1 つのデータ列をブートローダのターゲットに送信する列レベル API ファイルです。ブートロード動作の設定、行の消去、行のプログラミング、行の検証、およびブートロード動作の終了を行う関数が含まれています。**Table 10** は、この API ファイルの関数の詳細を説明します。

Table 10 *cybtldr_api.c /.h* の関数

関数	説明
`CyBtldr_StartBootloadOperation`	通信インターフェースを有効にし、Enter Bootloader コマンドをターゲットに送信する。 受信された応答パケットから、ターゲット デバイスのシリコン ID、シリコン リビジョンおよびブートローダのバージョンを確認する
`CyBtldr_ProgramRow`	最初に列を確認する。つまり、ターゲット フラッシュの特定のアレイ ID のために、Get Flash Size コマンドをターゲットに送信する。これに対する応答で、ターゲットはそのアレイにあるブートロード可能なフラッシュ部の開始と終了の列番号を返す。ホストはこの応答を読み込んで、指定した列がフラッシュのブートローダブルの領域にあるかどうか確認する。 列の確認が成功したら、ホストは列データをより細かい部分に分割して、Send Data コマンドでターゲットに送信する。 列データの最終部と共に、Program Row コマンドをターゲットに送信する
`CyBtldr_VerifyRow`	この関数は最初に特定のアレイ ID と列番号のために列を確認する。 列の確認が成功したら、確認されたフラッシュ列の Verify Row コマンドを送信する。このコマンドの応答で、ターゲットは列のチェックサムを返す。 返されたチェックサムは期待されるチェックサム値と比較される
`CyBtldr_EraseRow`	この関数も最初に特定のアレイ ID と列番号のために列を確認する。 列の確認が成功したら、確認されたフラッシュ列の Erase Row コマンドを送信する

関数	説明
CyBtldr_EndBootloadOperation	Exit Bootload コマンドを送信し、通信インターフェースを無効化する

11.4 cybtldr_command.c / .h

この API は、ターゲットへのコマンド パケットの構成を処理し、ターゲットから受信した応答パケットを解析します。cybtldr_api.c / .h は、この API の関数を呼び出します。例えば、Enter Bootload コマンドを送信するために、CyBtldr_StartBootloadOperation()は、この API の CyBtldr_CreateEnterBootloadCmd()関数を呼び出します。また、ターゲットに送信する前に、コマンド パケットのチェックサムを計算する関数もあります。

12 付録 E: その他のトピック

12.1 HSSP 対ブートローダ

ブートローダは、通信インターフェースを経由してシステム ファームウェアをアップグレードできるようにします。しかし、ブートローダのフラッシュ領域を含む完全なフラッシュのアップグレードを完了させるためには、SWD プログラマ (Host Sourced Serial Programming) を使用する必要があります。PSoC™ 4 の HSSP を作成するには、「**AN84858, PSoC™ 4 Programming Using an External Microcontroller (HSSP)**」を参照してください。

12.2 ブートロード処理中に電源が切れた場合は、どのようになるのか？

ブートロード処理中に電源が切れたら、次のリセットで、ブートローダブル プロジェクトのチェックサムが期待の値 (フラッシュの最終列に格納されるブートローダブル プロジェクトのチェックサム) と一致せず、そのブートローダブル プロジェクトが無効であると見なされます。ブートロードが成功するまでプログラムの実行はブートローダ内に留まります。ブートローダホストは、ブートロード動作を再起動するようにブートロード開始のコマンドを送信する必要があります。

12.3 なぜブートローダとブートローダブル プロジェクト間にジャンプするためにリセットが必要か？

PSoC™は非常にコンフィギュアラブルなデバイスです。ブートローダはファームウェアと内蔵ハードウェア リソースの変更を許可します。その高度なコンフィギュアラブルなアーキテクチャにより、ハードウェアの再設定 (配置、配線、ファンクション) はリセット状態にのみ可能です。従って、ブートローダはブートローダとブートローダブル プロジェクト間にジャンプするリセットが必要になります。

12.4 通常アプリケーション プロジェクトのブートローダブル プロジェクトへの変換

標準 (通常) のアプリケーション プロジェクトを既に作成し、これをブートローダブル プロジェクトに変換したい場合は、13 ページの **Figure 11** に示すようにトップ デザインにブートローダブル コンポーネントを追加し、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してください。

プロジェクトが通常のプロジェクトとして作成された後で、アプリケーション タイプを**「Bootloader」**に変更することでブートローダ プロジェクトに変更させる場合、ブートローダ プロジェクトが期待するように実行されるために、`Bootloader_Start()`呼び出し関数を *main.c* に挿入してください。

Note: *PSoC™ Creator 3.1 の場合、標準 (通常) のプロジェクトをブートローダブル／ブートローダ プロジェクトに変換したい時に、プロジェクトのアプリケーション タイプを「Bootloadable/Bootloader」に変更してください。それを行うには、**「Project」を右クリックして、「Build Setting」**>**「Code Generation」**>**「General」タブ**を選択して、「Application Type」を変更します。そして、トップ デザインにブートローダブル／ブートローダ コンポーネントを追加します。*

12.5 ブートローダブル プロジェクトのデバッグ

PSoC™ Creator ブートローダ システムでは、最初にブートローダ プロジェクトが、そしてブートローダブル 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 プロジェクトへのジャンプはソフトウェアが制御するデバイスリセットによって実施されます。この時、デバッガ インターフェースはリセットされます。つまり、ブートローダブル プロジェクトはデバッガ モードで実行できません。

付録 E: その他のトピック

ブートローダブル プロジェクトをデバッグするために、その「Application Type」を「Normal」に変更して、デバッグを実行します。デバッグが完了した後、それを「Bootloadable」に変えます。

他のオプションは、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にプログラムすることです。そして、ブートローダブルプロジェクトが実行中に、「Attach to running target」オプションを使用してデバッグを実行します。この場合には、デバッガがデバイスに取り付けられた時点からのみ、ブートローダブル プロジェクトのデバッグを実行できます。

12.6 マルチアプリケーション ブートローダ

マルチアプリケーション ブートローダ (MABL) は 2 つのブートローダブルのアプリケーションを同時にフラッシュに入れる時に使用されます。デバイスのフラッシュに 1 つの有効なアプリケーションが常に存在するよう、2 つのアプリケーションを同じものにすることができます。または、ブートローダ コマンドを使用して切り替えることができるように、2 つのアプリケーションを異なるものにすることもできます。確かに、各アプリケーションがフラッシュ領域を半分しか使用できないことはこの機能の制限です。

Figure 39 はフラッシュ メモリ内で MABL 用のプロジェクト配置を示します。

MABL は、標準ブートローダのアプリケーションとは異なる次の手順に従って実施できます。

1. MABL ブートローダ プロジェクトを新規作成します。ブートローダの設定ウィンドウで「Multi-App Bootloader」チェックボックスにチェックを入れます。

Note: PSoC™ Creator 3.1 の場合、アプリケーション タイプを「Multi-App Bootloader」に設定します。

2. 「Project_A」と「Project_B」という 2 つのブートローダブル プロジェクトをワークスペースに追加します。各プロジェクトでは、MABL プロジェクトに依存関係を追加します。各プロジェクトのために生成された 2 つの.*cyacd* ファイルは以下のように、1 つのファイルはフラッシュの上位領域用のもので、残りはフラッシュの下位領域用のものです(**Figure 39** を参照してください)。
   - Project_A_1.cyacd と Project_A_2.cyacd
   - Project_B_1.cyacd と Project_B_2.cyacd
3. 通常は、1 のサフィックスを付ける.*cyacd* ファイルは、フラッシュの前半分を、2 のサフィックスを付ける.*cyacd* ファイルは、フラッシュの後半分を占めます。そのため、.*cyacd* ファイルの特定の結合のみが使用できます。これらの結合は以下の通りです。
   - Project_A_1.cyacd と Project_A_2.cyacd
   - Project_B_1.cyacd と Project_B_2.cyacd
   - Project_A_1.cyacd と Project_B_2.cyacd
   - Project_B_1.cyacd と Project_A_2.cyacd
4. デバイスをマルチアプリケーション ブートローダ プロジェクトでプログラムし、上記のいずれか 1 つの結合に従い、ブートローダのホスト アプリケーションを使用して、アプリケーション (.*cyacd* ファイル) を順番にブートロードします。
5. アプリケーション間の切り替えは、下記の手順に従ってください。
   - USB ケーブルを使って、PSoC™キットを PC に接続します。ブートローダがアクティブであることを確認してください。
   - **Start** > **All Programs** > **Cypress** > **Bridge Control Panel** に移動して、Bridge Control Panel を開きます。I²C プロトコルの KitProg を選択します。
   - 0x38 コマンドを送信して、ブートローダに入ります。

```
w 08 01 38 00 00 C7 FF 17
```

```
r 08 x x x x x x x x x x x x x x x x
```

– application_1 から application_2 に切り替えるために、「set_active_application」コマンド (0x36) を送信します。

```
w 08 01 36 01 00 01 C7 FF 17
r 08 x x x x x x x
```

– application_2 から application_1 に切り替えるために、「set_active_application」コマンド (0x36) を送信します。

```
w 08 01 36 01 00 00 C8 FF 17
r 08 x x x x x x x
```

– 「exit_bootloader」コマンド (0x3B) を送信して、アプリケーションを実施します。

```
w 08 01 3B 00 00 C4 FF 17
r 08 x x x x x x x
```

これらのコマンドは、**Table 11** に「set_active_application」コマンド (application_1 から application_2 へ) の例を使って説明されています。

Table 11　　コマンド バイト

バイト	1	1	1	2	N	2	1
値	08	01	36	01 00	01	C7 FF	17
説明	スレーブ アドレス	パケット 開始バイト	コマンド	後続するバイト数(LSB ファースト)	データ バイト	チェックサム (LSB ファースト)	パケット 終了バイト

12.6.1　ブートローダ用に必要なメモリ量

全てのオプション コマンドを含む標準 I²C ブートローダ プロジェクトは、Arm GCC コンパイラの最適化を「size」に設定した場合、PSoC™フラッシュ メモリの約 4.3KB を占めます。プロジェクトを構築する時、ブートローダ プロジェクトが使用するメモリを出力ウィンドウで見ることができます。ブートローダ プロジェクトが使用する RAM メモリはブートローダブル プロジェクトによって再使用できます。

Figure 45 に示すように、ブートローダブル プロジェクトのメモリ使用量は、ブートローダ コンポーネントがサポートするオプションのコマンドを取り除くことにより、少量に低減できます。

Figure 46 に示すように、*.cydwr* > System タブ内で **Device Configuration Mode** を **Compressed** に設定して、フラッシュ メモリの使用量を最小限に抑えます。

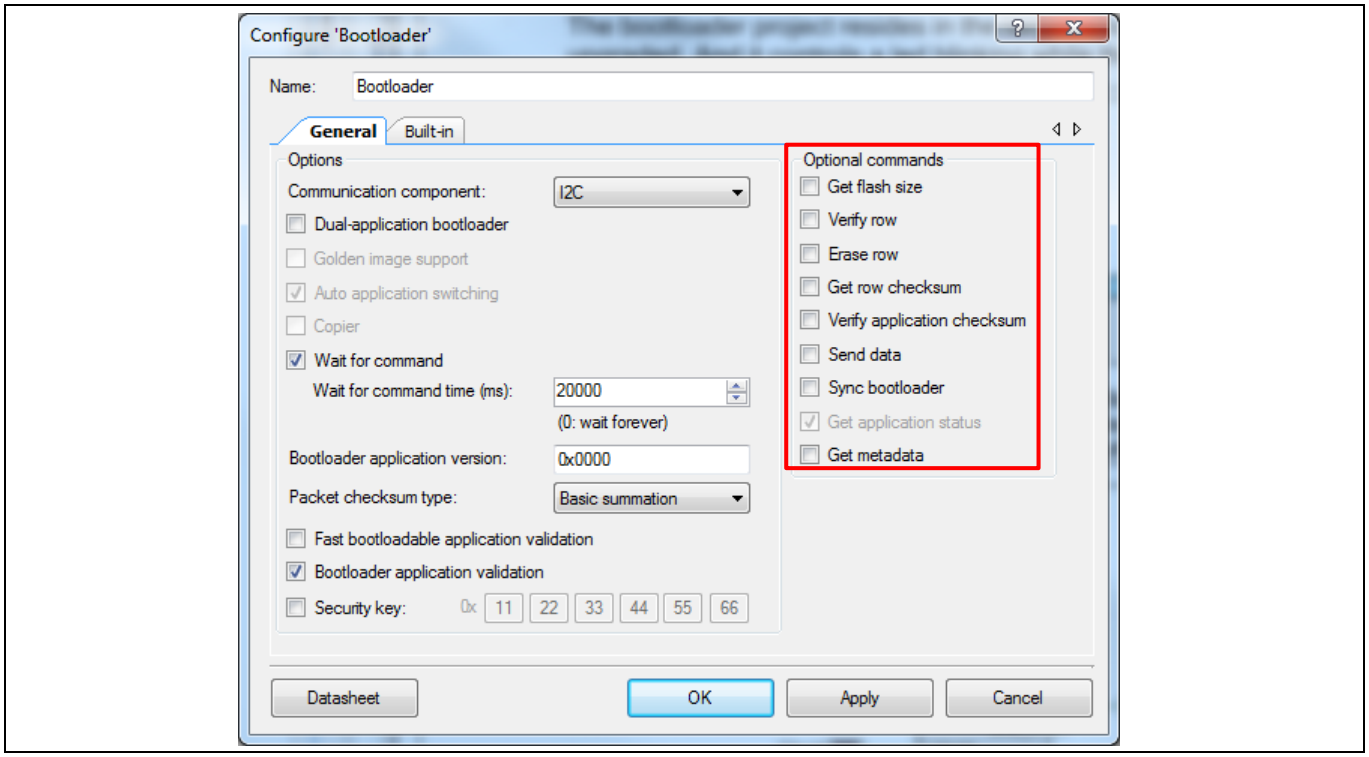


Figure 45　ブートローダ コンポーネント内のオプションのコマンドをチェックしない時

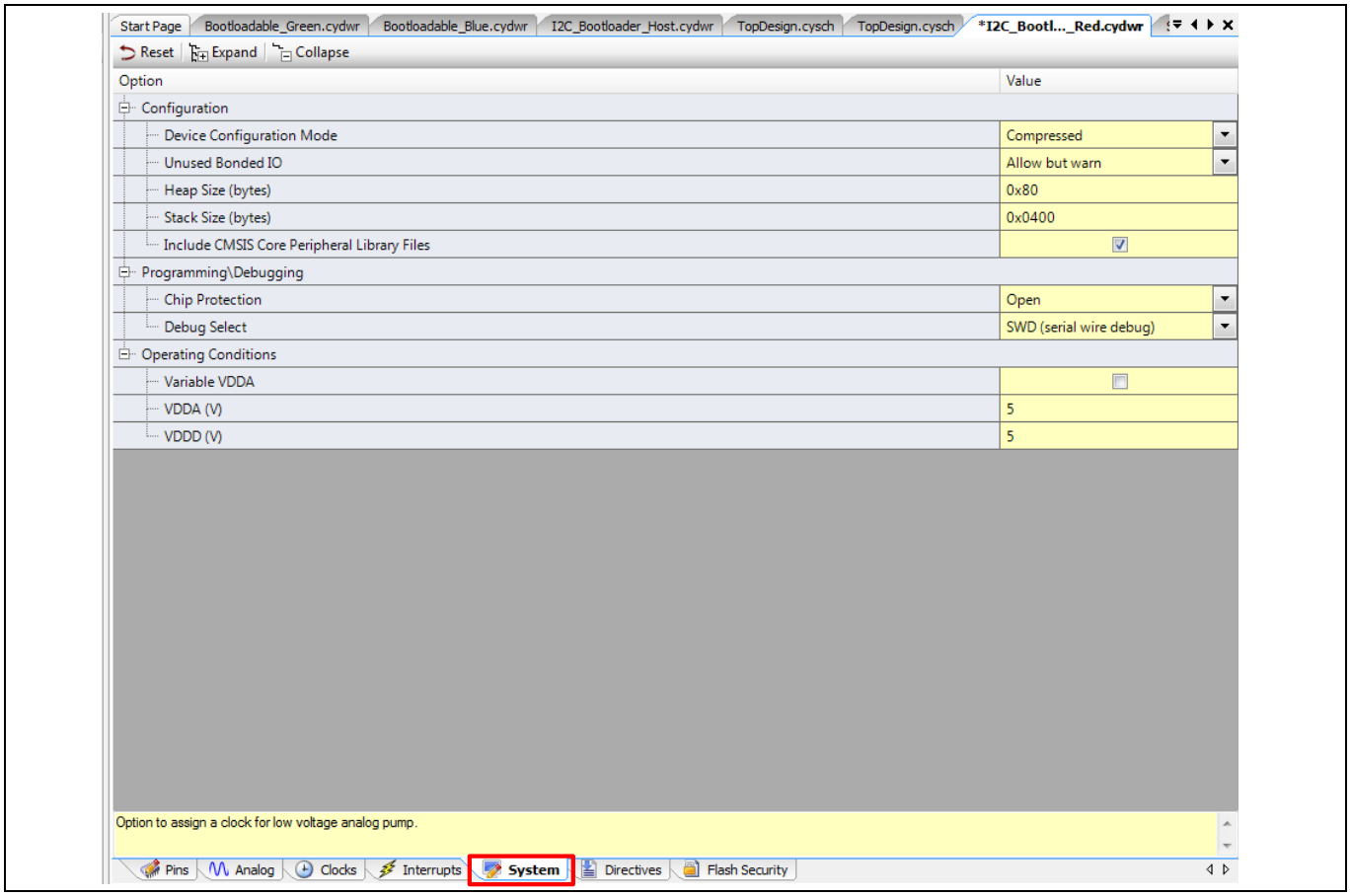


Figure 46　デバイス設定モード

12.6.2 PSoC™キットと PSoC™用 MiniProg3 の比較

PSoC™ 4 キットおよび **MiniProg3** は、PSoC™用にデバッグ／プログラムおよび USB-I^{2}C ブリッジをサポートします。また、PSoC™は、MiniProg3 がまだ対応していない USB-UART ブリッジもサポートします。そのため、PSoC™キットを使用してデバッグ／プログラムおよびブートロードを簡単に実行できます。詳細については、「**CY8CKIT-040 Kit Guide**」と「**CY8CKIT-042 PSoC™ Pioneer Kit Guide**」を参照してください。

13 付録 F - キットの選択

ターゲット デバイス	キット名	ユーザー ガイド
CY8C42xx	CY8CKIT-042 PSoC™ 4 Pioneer Kit	**CY8CKIT-042**
CY8C40xx-S	CY8CKIT-041-40xx PSoC™ 4 S シリーズ Pioneer Kit	**CY8CKIT-041**
CY8C41xx-S	CY8CKIT-041-41xx PSoC™ 4100S CapSense Pioneer Kit	**CY8CKIT-041**
CY8C40xx	CY8CKIT-040 Pioneer Kit	**CY8CKIT-040**
CY8C41xx-PS	CY8CKIT-147 PSoC™ 4100PS Prototyping Kit	**CY8CKIT-147**
CY8C4100S Plus	CY8CKIT-149 PSoC™ 4100S Plus Prototyping Kit	**CY8CKIT-149**

改訂履歴

Document version	Date of release	Description of changes
**	2013-10-03	これは英語版 001-86526 Rev. **を翻訳した日本語版 001-89532 Rev. **です。
*A	2015-04-17	これは英語版 001-86526 Rev. *C を翻訳した日本語版 001-89532 Rev. *A です。
*B	2016-05-10	これは英語版 001-86526 Rev. *D を翻訳した日本語版 001-89532 Rev. *B です。
*C	2017-03-28	最新のテンプレートに更新しました。
*D	2018-11-19	これは英語版 001-86526 Rev. *G を翻訳した日本語版 001-89532 Rev. *D です。
*E	2021-10-25	テンプレートの変更を実施。 これは英語版 001-86526 Rev. *H を翻訳した日本語版 Rev. *E です。

Trademarks
All referenced product or service names and trademarks are the property of their respective owners.

Edition 2021-10-25
Published by
Infineon Technologies AG
81726 Munich, Germany

Do you have a question about this document?
Go to www.cypress.com/support

Document reference
001-89532 Rev. *E

重要事項
本文書に記載された情報は、いかなる場合も、条件または特性の保証とみなされるものではありません（「品質の保証」）。本文に記された一切の事例、手引き、もしくは一般的価値、および／または本製品の用途に関する一切の情報に関し、インフィニオンテクノロジーズ（以下、「インフィニオン」）はここに、第三者の知的所有権の不侵害の保証を含むがこれに限らず、あらゆる種類の一切の保証および責任を否定いたします。

さらに、本文書に記載された一切の情報は、お客様の用途におけるお客様の製品およびインフィニオン製品の一切の使用に関し、本文書に記載された義務ならびに一切の関連する法的要件、規範、および基準をお客様が遵守することを条件としています。

本文書に含まれるデータは、技術的訓練を受けた従業員のみを対象としています。本製品の対象用途への適合性、およびこれら用途に関連して本文書に記載された製品情報の完全性についての評価は、お客様の技術部門の責任にて実施してください。

本製品、技術、納品条件、および価格についての詳しい情報は、インフィニオンの最寄りの営業所までお問い合わせください（**www.infineon.com**）。

警告事項
技術的要件に伴い、製品には危険物質が含まれる可能性があります。当該種別の詳細については、インフィニオンの最寄りの営業所までお問い合わせください。

インフィニオンの正式代表者が署名した書面を通じ、インフィニオンによる明示の承認が存在する場合を除き、インフィニオンの製品は、当該製品の障害またはその使用に関する一切の結果が、合理的に人的傷害を招く恐れのある一切の用途に使用することはできないこと予めご了承ください。