AN97060

PSoC® 4 BLE および PRoC™ BLE – 無線 (OTA) のデバイス ファームウェア アップグレード (DFU) ガイド

著者: Deepak John

関連製品ファミリ: すべての PSoC 4 BLE および PRoC BLE

関連サンプル コード: CE95351

関連アプリケーション ノート: 完全な一覧はこちらをクリックしてください。本アプリケーション ノートの最新版または関連プロジェクトファイルは http://www.cypress.com/go/AN97060 をご覧ください。

AN97060 は PSoC® 4 および PSoC™ BLE デバイスに基づくアプリケーション用の無線 (OTA) のファームウェア アップグレード機能を実装するガイドラインを提供します。

目次

1 はじめに 2
2 PSoC および PRoC リソース 2
3 PSoC Creator 4
4 BLE OTA ブートローダ 5
4.1 外部メモリ OTA ブートローダ 6
4.2 固定スタック OTA ブートローダ 7
4.3 アップグレード可能スタック OTA ブートローダ 8
5 ターゲット プロジェクトへのファームウェア OTA ブートローダの追加 10
5.1 基本的なサンプル ターゲット プロジェクトの作成 10
5.2 外部メモリ OTA ブートローダの追加 13
5.3 固定スタック OTA ブートローダの追加 16
5.4 アップグレード可能スタック OTA ブートローダの追加 22
6 OTA アップグレードの実行 30
6.1 Bootloader Host ツールによるアップグレード 31
6.2 CySmart PC ツールによるアップグレード 32
6.3 CySmart モバイル アプリによるアップグレード 33
7 OTA 機能のテスト 33
8 その他の注意事項 34
8.1 ボンディング／ペアリングの情報 34
8.2 デバッグ 35
8.3 データ長拡張 (DLE) 36
8.4 データの永続性 37
9 まとめ 39
10 関連アプリケーション ノート 39
A 付録 A 40
A.1 サンプル プロジェクト ワークスペースの作成 40
A.2 既存のワークスペースへのサンプル プロジェクトの追加 41
A.3 他のデバイス選択 43
A.4 ブートローダ サービスの追加 45
A.5 サイプレスの他の BLE デバイス用の固定スタック OTA プロジェクトの設定 47
改訂履歴 50
ワールドワイドな販売と設計サポート 51

1 はじめに

無線 (OTA) のデバイス ファームウェア アップグレードは基本的に無線リンクを使用してターゲット デバイスのファームウェアを更新するブートロード メカニズムです。OTA は任意の無線リンクで実行できますが、本アプリケーション ノートで言及する OTA は Bluetooth® Low Energy (BLE) リンクで実行されるものを意味します。BLE デバイスの OTA 機能は、デバイスの機能をアップグレードするか、すでに現場で展開されたデバイスのファームウェアの問題を修正する際に役立ちます。

本アプリケーション ノートでは、さまざまな OTA アップグレード オプションおよびユーザーの製品に適切なオプションを選択する方法を簡単に説明します。また、本書は最終製品での機能の統合および展開を支援するテストおよびトラブルシューティングの詳細な情報も提供します。

本アプリケーション ノートをお読みになる前に、ブートローダ理論および技術について簡単に紹介し、PSoC Creator™を使用して PSoC 3、PSoC 4 および PSoC 5LP デバイスでブートローダを迅速かつ簡単に実装する方法を示す AN73854 をお読みください。

本アプリケーション ノートで説明される OTA 機能に加えて、PSoC ブートローダを使用することにより UART、I²C や SPI などの他のインターフェースを介して PSoC 4 BLE および PRoC BLE デバイス (詳細は「PSoC および PRoC リソース」を参照してください) のファームウェアを更新できます。「関連アプリケーション ノート」を参照してください。本アプリケーション ノートでは、BLE による OTA 機能を説明します。

PSoC Creator のメニュー オプション **File** > **Code Example...**を選択し、ブートローダに関するサンプル プロジェクトにアクセスできます。ポップアップ ウィンドウで「bootloader」を検索してください。

2 PSoC および PRoC 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 (プログラマブル システム オン チップ) および PRoC (プログラマブル ラジオ オン チップ) デバイスを選択し、迅速かつ効率的にデバイスをデザインに統合できます。リソースの包括的な一覧は「KBA86521, How to Design with PSoC 3, PSoC 4, and PSoC 5LP」を参照してください。以下は PSoC 4 BLE および PRoC BLE のリソースの要約です。

- **概要:** Bluetooth Low Energy ポートフォリオ、サイプレス ワイヤレス／RF ロードマップ
- **製品セレクタ:** PSoC 4 BLE または PRoC BLE。また、PSoC Creator にはデバイス選択ツールが含まれています。
- **データシート**: PSoC 4 BLE および PRoC BLE デバイス ファミリの電気的仕様を説明します。
- **CapSense®デザイン ガイド:** PSoC 4、PSoC 4 BLE、および PRoC BLE ファミリのデバイスを使用して静電容量タッチセンシング 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**: 基本レベルから上級レベルまでの幅広いトピックに触れています。多くのアプリケーション ノートはサンプル コードを含みます。
- **テクニカル リファレンス マニュアル (TRM)**: 各 PSoC 4 BLE および PRoC BLE デバイス ファミリのアーキテクチャおよびレジスタの詳細な説明を提供します。
- **開発キット:**
  - CY8CKIT-042-BLE および CY8CKIT-042-BLE-A の BLE Pioneer Kit を使用し、PSoC 4 BLE および PRoC BLE デバイスを使って BLE アプリケーションを評価し、開発できます。
  - CY5682 PRoC BLE タッチ マウス リファレンス デザイン キットを使用し、BLE または Bluetooth Smart タッチ マウスの完全な量産品の開発が可能です。
  - CY5672 PRoC BLE リモコン リファレンス デザイン キットを使用し、BLE または Bluetooth Smart リモコンの完全な量産品の開発が可能です。
  - CY8CKIT-042 および CY8CKIT-040 の PSoC 4 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 カードの専用コネクタを搭載します。
  - CY8CKIT-049 は、PSoC 4 デバイスをサンプリングできる低コスト プロトタイプ プラットフォームのシリーズです。
  - CY8CKIT-001 は、すべての PSoC デバイス ファミリの共通開発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ミングとデバッグ用のインターフェースを提供します。

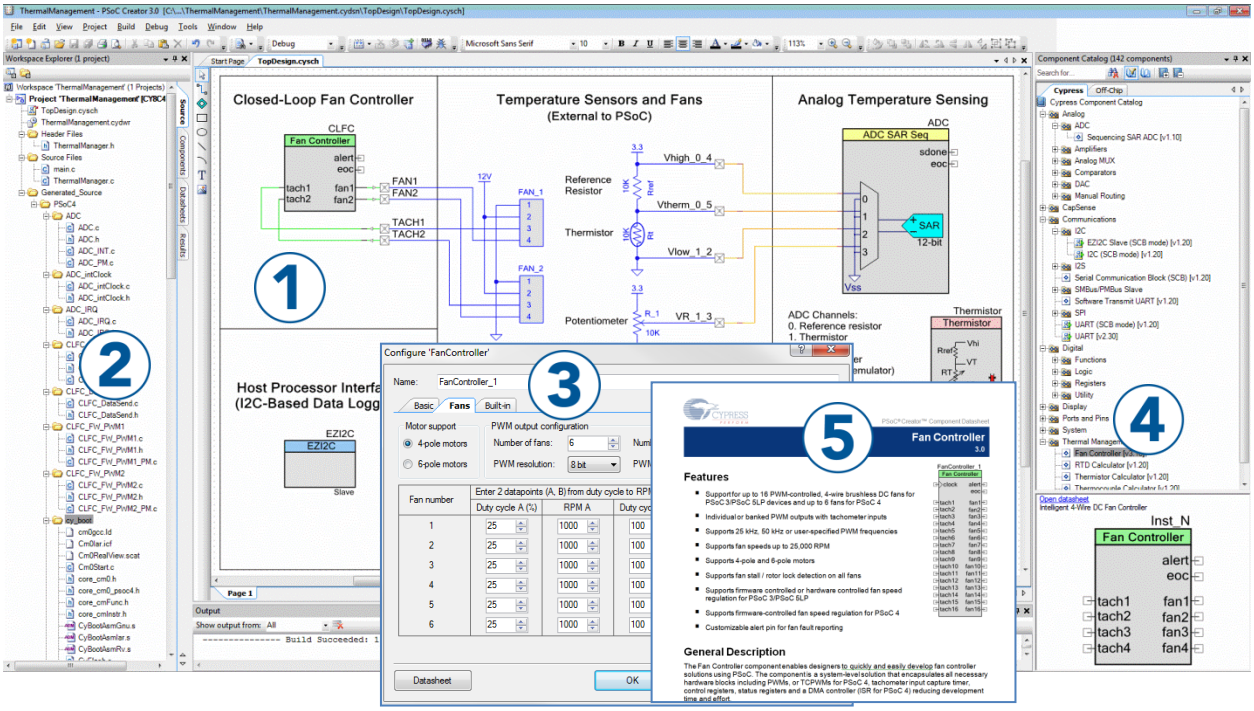
- **CySmart™**は Windows PC 用の BLE ホスト エミュレーション ツールです。このツールは、BLE ペリフェラル アプリケーションをテストおよびデバッグするための使いやすい GUI を提供します。
- **CySmart モバイル アプリ**はサイプレスが開発した BLE または Bluetooth Smart ユーティリティです。CySmart はさまざまな BLE 製品との接続に使用し、また CY8CKIT-042-BLE PSoC 4 BLE Pioneer Kit、CY5672 PRoC BLE リモコン リファレンス デザイン キット、および CY5682 PRoC BLE タッチ マウス リファレンス デザイン キットを含むサイプレスの BLE 開発キットと併用できます。
- **サイプレスのカスタム BLE プロファイルとサービス**: サイプレスの BLE コンポーネント (PSoC Creator の一部として使用される) は Bluetooth Special Interest Group (SIG) によって導入された GATT ベースの BLE プロファイルおよびサービスを含むように定期的に更新されます。Bluetooth SIG によって定義されたプロファイルおよびサービス以外に、サイプレスはいくつかのカスタム BLE プロファイルおよびサービスを定義しました。これにより、Bluetooth SIG によって指定された標準 BLE プロファイルおよびサービスがサポートしていない機能に対してデータを BLE 経由で送信できます。これらのプロファイルおよびサービスは、サイプレス BLE デバイスと通信するデバイス、または互いに通信するサイプレス BLE デバイスによって利用できます。

3 PSoC Creator

PSoC Creator は無償の Windows ベースの統合開発環境 (IDE) です。これは、PSoC 3、PSoC 4、PSoC 4 BLE、PRoC BLE および PSoC 5LP ベース システムのハードウェアとファームウェアの同時設計を可能にします。図 1 を参照してください。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 &ドロップし、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設定
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 データシートを閲覧

図 1. PSoC Creator の特長



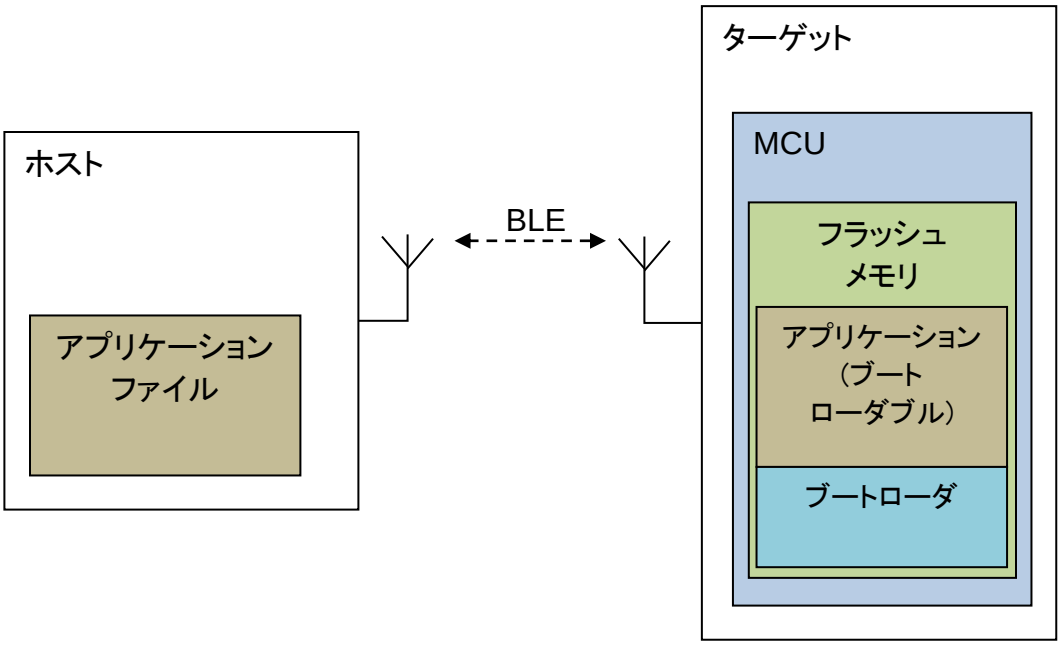
4 BLE OTA ブートローダ

以下の用語は本書で頻繁に使用されています。これらの定義を理解することは、本アプリケーション ノートの理解に重要です。

- **ブートローダ:** フラッシュ メモリの更新方法を知っており、その責任を担うファームウェアの一部です。
- **ブートローダブル:** 無線で受信され、ターゲット デバイスのフラッシュ メモリで更新されるアプリケーションを含むファームウェアの一部です。
- **ランチャー:** 通信コンポーネントなしでブートローダ コンポーネントを組み込むことによって定義されているブートローダです。ランチャーは [ランチャー＋コピアー] として構成することもできます。コピアーは、以前に保存されたスタック アプリケーション (BLE スタックを含む PSoC Creator プロジェクト) イメージを一時的位置からスタック アプリケーション フラッシュ空間にコピーする (旧スタック アプリケーションを上書きする) 追加の機能 (ランチャーに備えられる) です。

ターゲット MCU におけるフラッシュ メモリは、図 2 に示すように、2 つのセクションに分割されています: アプリケーション (ブートローダブル イメージ) とブートローダです。このアーキテクチャの他のバリエーションは特定のニーズを満たすために利用可能です。ブートローダおよびその機能フローの実装の詳細は AN73854 とブートローダおよびブートローダブル コンポーネント データシートを参照してください。

図 2. BLE ブートローダ システム



データ (ブートローダブル部分) をホスト (PC またはスマートフォン) からターゲット デバイスのフラッシュに転送するプロセスは「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」)、「ファームウェア アップグレード」、または「デバイス ファームウェア アップグレード (DFU) 」と呼びます。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、OTA のアップグレードを可能にする任意の BLE プロジェクトに追加できる 3 種類の BLE ブートローダを提供します。

- 外部メモリ OTA ブートローダ
- 固定スタック OTA ブートローダ
- アップグレード可能スタック OTA ブートローダ

それぞれのブートローダは利点と欠点を持っています。ここでは、それらのブートローダのアーキテクチャおよび他の設計上の考慮事項について簡単に説明します。ここで述べられる情報に基づいて、ユーザーは設計に最適なブートローダを選択できます。

4.1 外部メモリ OTA ブートローダ

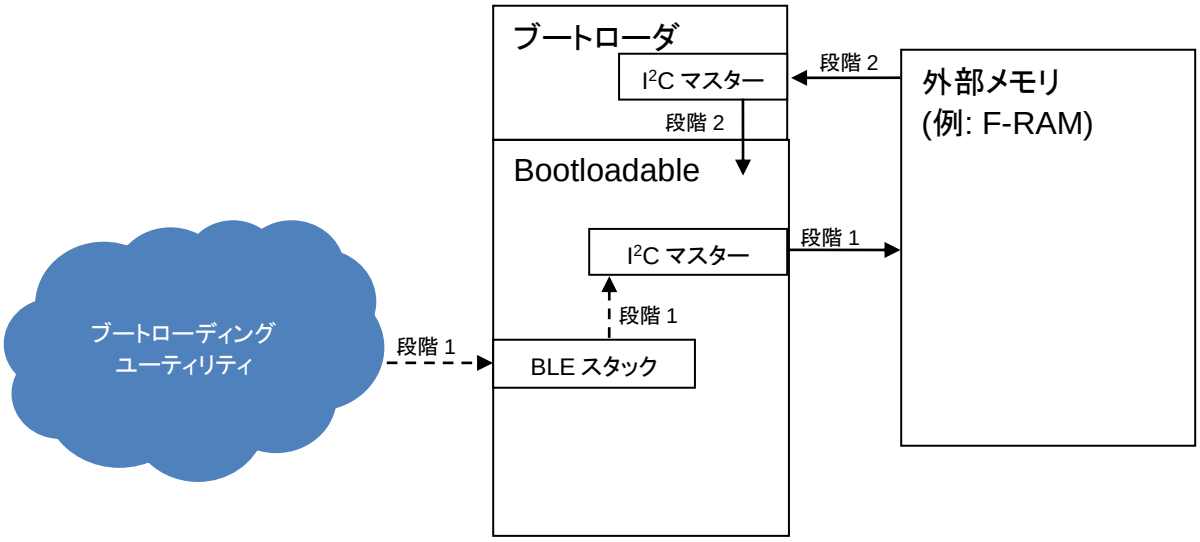
外部メモリ OTA ブートローダは PSoC Creator で提供されているブートローダおよびシングル ブートローダブル イメージ アーキテクチャを使用します。これはブートローダブル イメージを一時的に格納するための外部メモリを使用します。BLE スタックはファームウェアのブートローダブル セクションに位置します。図 3 を参照してください。したがって、ファームウェアのアップグレードによりアプリケーションと BLE スタックの両方がアップグレードされます。

例えば、外部メモリは BLE Pioneer Kit の I²C バスを介して PSoC／PRoC BLE デバイスに接続されます。PSoC／PRoC BLE デバイスは I²C マスターであり、外部メモリ デバイス (F-RAM™) は I²C スレーブです。

外部メモリ OTA ブートローダ実装では、ブートロードは 2 段階で行われます。段階 1 では、デバイスのフラッシュ メモリに存在しているブートローダブル アプリケーションは BLE を介して新しいブートローダブル イメージを受信します。アプリケーション イメージの各チャンクは、受信されると、検証され、I²C を介して外部メモリに書き込まれます。ブートローダブル イメージが正常に受信されると、ファームウェアはブートローダ セクションに制御を渡します。

段階 2 では、ブートローダは新しいアプリケーション イメージ (段階 1 で受信した) でデバイス フラッシュを再プログラムします。プロセス全体は図 3 に示されています。矢印はデータ フローの方向を示します。

図 3. 外部メモリ OTA ブートロード プロセス



利点

- BLE スタックとアプリケーションを一緒にアップグレードできます。
- ブートローダとブートローダブルがメモリを共有しない 2 つの個別プロジェクトであるため、ファームウェアは簡単に実装できます。

欠点

- 外部メモリを必要とします。外部メモリの目的が OTA を有効にすることだけであれば、これは課題 (BOM の増加) です。
- ファームウェアのアップグレードは他の OTA ブートローダに比べてより長い時間がかかります。原因としてイメージは、I²C を使用して外部メモリ デバイスに保存され、そこから取得されるためです。
- I²C に関するコードは、ブートローダとブートローダブル両方のエリアにあるため、フラッシュの使用量が増えます。

外部メモリ OTA ブートローダはサイプレスのプログラマブル BLE 製品と共に使用できます。

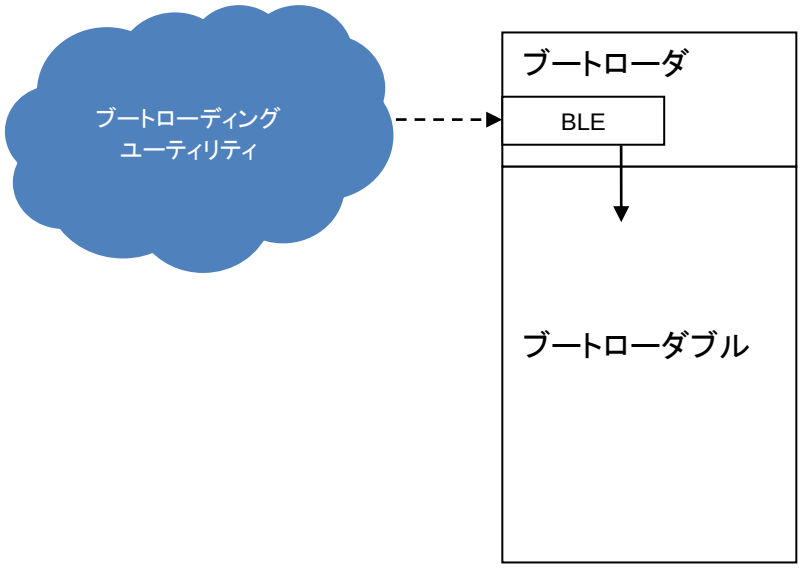
4.2 固定スタック OTA ブートローダ

外部メモリ ブートローダの方法と同様に、固定スタック OTA ブートローダも PSoC Creator のブートローダおよびシングル ブートローダブル イメージ アーキテクチャを使用します。しかし、BLE スタックはブートローダ メモリ内に位置しているため、OTA アップグレードを介してアップグレードできません。ブートローダブル アプリケーションはブートローダ メモリに位置している BLE API (BLE コンポーネント データシートを参照してください) に接続する必要があります。

理想的には、ブートローダとブートローダブル イメージ (2 つの PSoC Creator プロジェクトで実装された) はそれら自身の BLE スタックのコピーを持つ必要がありますが、これはメモリ消費量を増加させます。実用的な方法として、スタックを共有することで、ブートローダとブートローダブル プロジェクトは BLE コンポーネントに関連するコードを再利用できます。これにより、ファームウェアのアプリケーション部分によって使用できるフラッシュのかなりの量を節約できます。このアーキテクチャは次のようにファームウェアのアップグレード プロセスを簡略化します。

固定スタック OTA ブートローダ実装では、ブートロードは 1 段階で行われます。ファームウェアはブートローダ モードに直接入り、アプリケーション イメージが BLE リンクを介して受信されるまで待機します。検証が成功した後、受信されたアプリケーション イメージはブートローダブル エリアに直接書き込まれます。図 4 に、データ フローを示す矢印で、固定スタック OTA ブートロード プロセスを示します。

図 4. 固定スタック OTA ブートロード プロセス



利点

- BLE スタックはブートローダとブートローダブル プロジェクトで再利用されるため、フラッシュ メモリ使用量を節約できます。
- 受信されたアプリケーション イメージがフラッシュ メモリに直接書き込まれるため、アップグレード時間は短くなります。
- 外部メモリまたはストレージは必要ありません。
- 現行のイメージが無効になってもアプリケーション イメージのアップグレードは可能です。

欠点

- BLE スタックはブートローダの一部になり、OTA を介してアップグレードすること (BLE プロファイルを含む) はできません。

BLE スタックは、フラッシュ メモリ内のブートローダ エリアにあります。これは、OTA ファームウェアのアップグレード中は変更されません。したがって、ブートローダは常に回復し、有効なアプリケーション イメージがなくなっても、いつでも OTA アップグレードができる状態になっています。固定スタック OTA ブートローダはサイプレスのプログラマブル BLE 製品と共に使用できます。

4.3 アップグレード可能スタック OTA ブートローダ

アップグレード可能スタック OTA ブートローダはデュアル アプリケーション イメージ アーキテクチャを使用します。このアーキテクチャでは、利用可能なフラッシュはランチャー＋コピアーのイメージ (以下、「ランチャー」という)、スタック アプリケーション イメージ、およびユーザー アプリケーション イメージの 3 つのセクションに分けられています (図 5 を参照してください)。

ランチャーのイメージは、フラグやイメージの有効性に応じて、スタック アプリケーションまたはユーザー アプリケーションのいずれかを開始します。スタック アプリケーションの更新中にランチャーは、最新のスタック アプリケーション イメージ (BLE リンクでダウンロードされ、一時的な[ユーザー アプリケーション] に保存された) をフラッシュのスタック アプリケーション位置にコピーします。

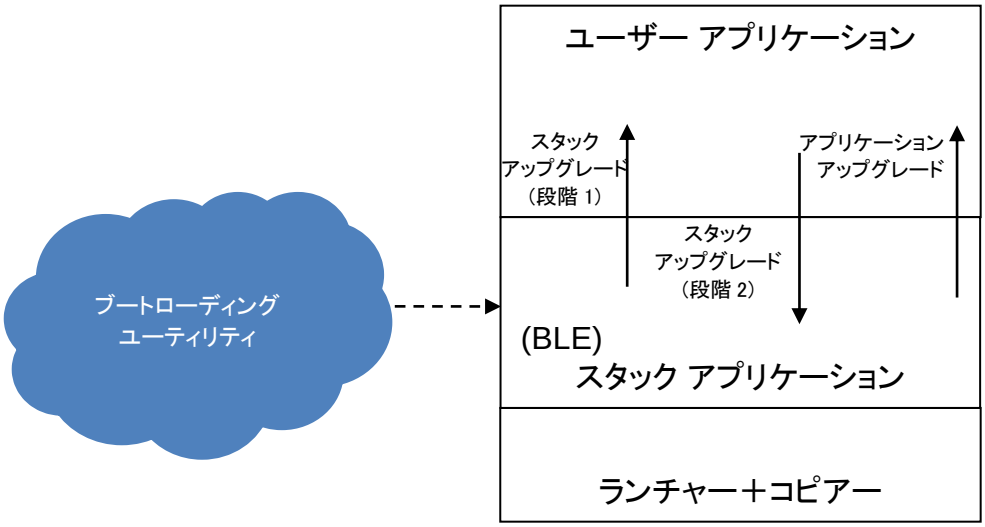
スタック アプリケーション イメージは、スタック アプリケーション イメージとユーザー アプリケーション イメージ間で共有される必要な BLE スタックまたは他のコードを含みます。スタック アプリケーションはユーザー アプリケーション イメージのアップグレードを行います。また、スタック アプリケーション イメージの新しいバージョンをダウンロードしてフラッシュのユーザー アプリケーション領域に一時的に保存します。

ユーザー アプリケーションは心拍数センサー、リモコン、マウスなどの最終アプリケーションの機能を実現します。これはスタック アプリケーション イメージに位置する BLE API (BLE コンポーネント データシートを参照してください) に接続します。これにより、スタック (および／またはこの領域内の他のコード) をスタック アプリケーションとユーザー アプリケーション間で共有できます。

理想的には、スタック アプリケーション イメージとユーザー アプリケーション イメージ (2 つの PSoC Creator プロジェクトで実装された) はそれら自身の BLE スタックのコピーを持つ必要がありますが、これはメモリ消費量を増加させます。スタック (および／または他のコード) を共有することで、スタック アプリケーションとユーザー アプリケーションが BLE スタックに関するコードを再利用できます。これにより、アプリケーション イメージによって使用できるフラッシュのかなりの量を節約します。このアーキテクチャも 2 つのファームウェア アップグレード オプションを提供します。

- **アプリケーション アップグレード:** ユーザー アプリケーション イメージのみがアップグレードされます。アプリケーション アップグレードは 1 段階で行われます。OTA アップグレード モードに入るには、ファームウェアは新しいユーザー アプリケーション イメージを受け取るスタック アプリケーションに制御を渡します。その後、スタック アプリケーションは対応するフラッシュ領域にその新しいユーザー アプリケーション イメージを直接書き込みます (図 5 を参照してください)。
- **スタック アップグレード:** スタック アプリケーションとユーザー アプリケーションは両方ともアップグレードされます。
  - 段階 1: ファームウェアはスタック アプリケーションに制御を渡します。スタック アプリケーションは、新しいスタック アプリケーション イメージを受信してフラッシュ メモリ内の一時的位置 (ユーザー アプリケーション領域) に書き込みます。この段階ではユーザー アプリケーションが破損しています (新しいスタック アプリケーション イメージは既存のユーザー アプリケーション イメージを上書きします)。
  - 階段 2: ダウンロードが完了した後、スタック アプリケーションはソフトウェア リセットを開始し、制御はランチャー イメージに渡されます。これは、一時的位置 (ユーザー アプリケーション領域) にあるイメージを検出し、スタック アプリケーション領域にコピーします (図 5 を参照してください)。ここでユーザー アプリケーション イメージが破損しているため、アプリケーション アップグレードも実行する必要があります。

図 5. アップグレード可能スタック OTA ブートロード プロセス



利点

- BLE スタックとアプリケーション イメージの両方を更新する柔軟性を提供します。
- この方法では BLE スタックが再利用されるため、フラッシュ メモリの消費量を減らせます。
- 受信される BLE スタック／アプリケーション イメージがフラッシュに直接書き込まれるため、アップグレードの時間は短くなります。
- 外部メモリまたはストレージは必要ありません。

- BLE スタックがアップグレードされているとき、BLE スタックの新しいコピーを格納するために大容量 (256KB) のフラッシュ デバイスを使用する必要があります。

欠点

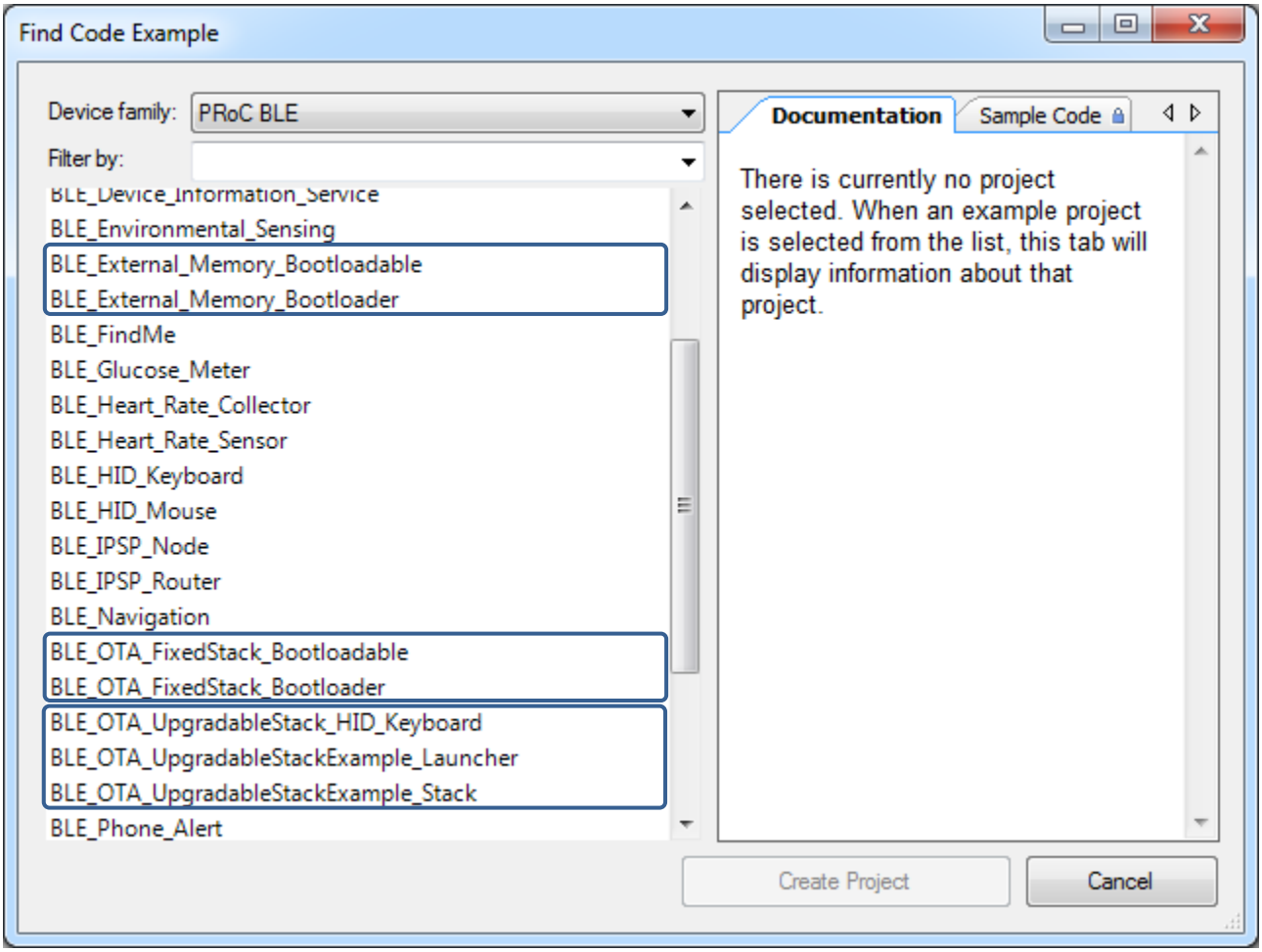
アップグレード可能スタック OTA は BLE スタックとブートローダブル／アプリケーション イメージの両方をアップグレードすることを可能にします。さらに、(ダウンロード中に) BLE スタック イメージを一時的にフラッシュに格納する必要があるため、大容量のフラッシュ デバイスを必要とします。そのため、アップグレード可能スタック OTA ブートローダのオプションは、サイプレスの 256KB の BLE 製品のみで使用できます。

5 ターゲット プロジェクトへのファームウェア OTA ブートローダの追加

ここではターゲット プロジェクトに OTA ブートローダを追加する方法について説明します。これを実現するために、サンプルのターゲット プロジェクトを作成してから、このターゲット プロジェクトに OTA ブートローダを追加します (すべての 3 種類の OTA ブートローダを追加する手順は個別に説明されます)。

PSoC Creator は OTA 対応サンプル プロジェクトを提供しています。図 6 に示すように、**File** > **Example Project…**を選んで、適切な **Device Family** とキーワードによる **Filter by** を選択することで、それらのサンプル プロジェクトにアクセスできます。各サンプル プロジェクトには、それぞれの資料が付属しています。特定の実装情報は資料をお読みください。ここでは、非 OTA アプリケーション ファームウェアに OTA 機能を追加する方法について説明します。ブートローダの異なる種類は「BLE OTA ブートローダ」を参照してください。

図 6. BLE サンプル プロジェクト



5.1 基本的なサンプル ターゲット プロジェクトの作成

OTA ブートローダを追加できる基本のサンプル プロジェクトを用意する必要があります。このために、スタート ポイントとして PWMExample プロジェクトを使用します。このプロジェクトは、PRoC BLE／PSoC 4 BLE 内の PWM コンポーネントを使って LED 輝度調整を実装します。PSoC 4 BLE／PRoC BLE デバイス用の PWMExample プロジェクトを作成するには、以下の手順に従ってください。いずれの OTA ブートローダを追加したいプロジェクトがすでにあった場合、本節を飛ばしてください。

1. PSoC Creator で、**File** > **Code Example …**を選択します。図 7 に示す **Find Example Project** ダイアログが開きます。

2. 図 7 に示すように、**Find Example Project** ダイアログで、**Device Family** フィルターを **PSoC 4200 BLE** に、**Filter by** のキーワードを **PWMExample** に設定します。
3. 一覧から **PWMExample** プロジェクトを選択して **Create New Project** をクリックします。
4. **Workspace** フィールドで **Create New Workspace** オプションを選択します。新しいサンプル プロジェクトのワークスペース用のロケーションを選択して (図 8 を参照してください)、**Finish** をクリックします。

図 7. Find Example Project ダイアログ

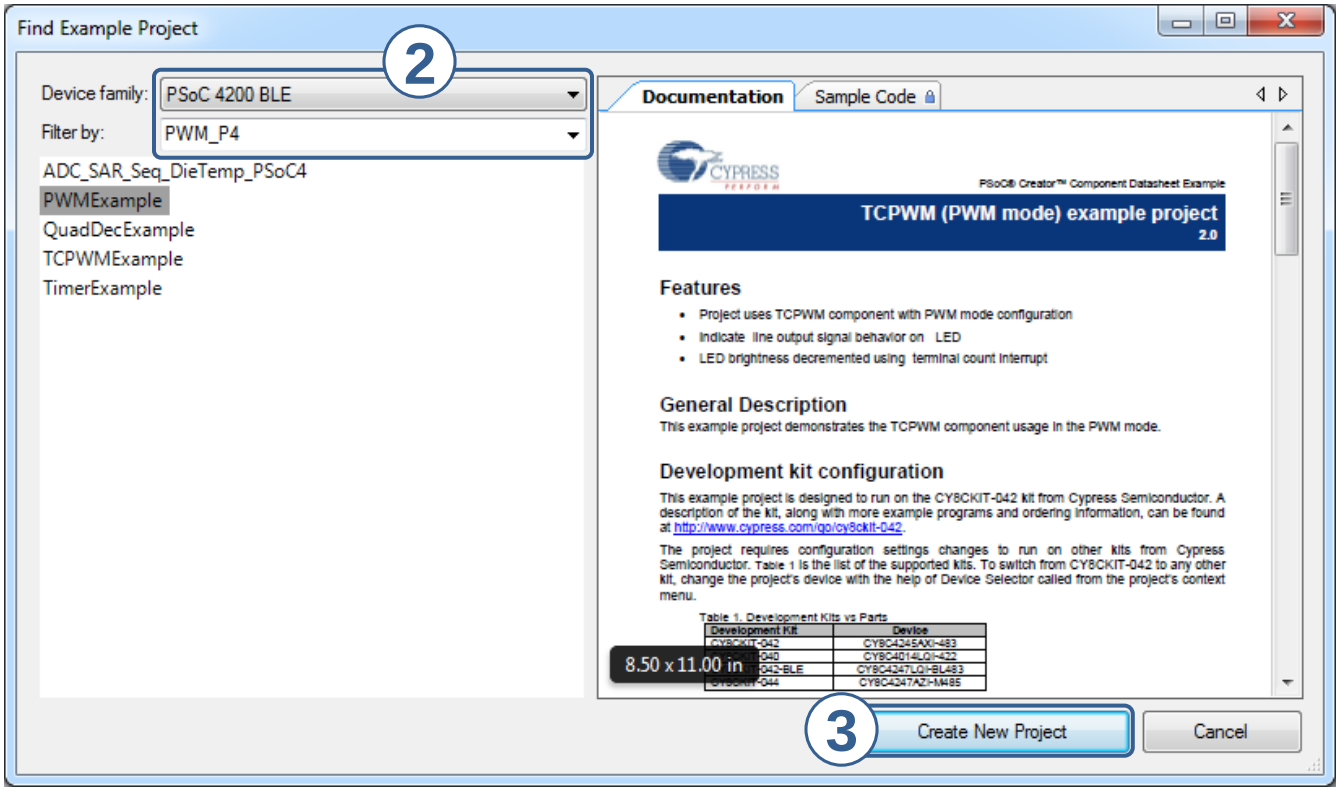
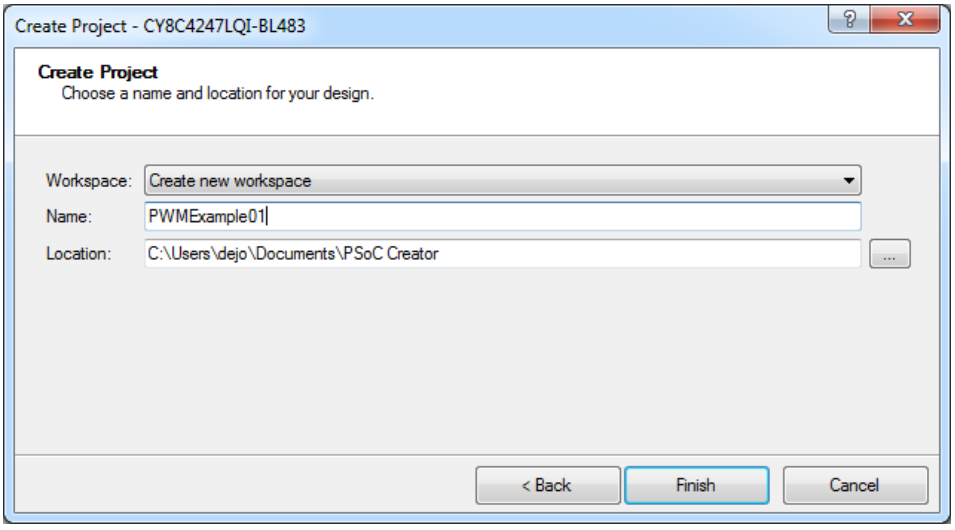


図 8. Create Project ダイアログ



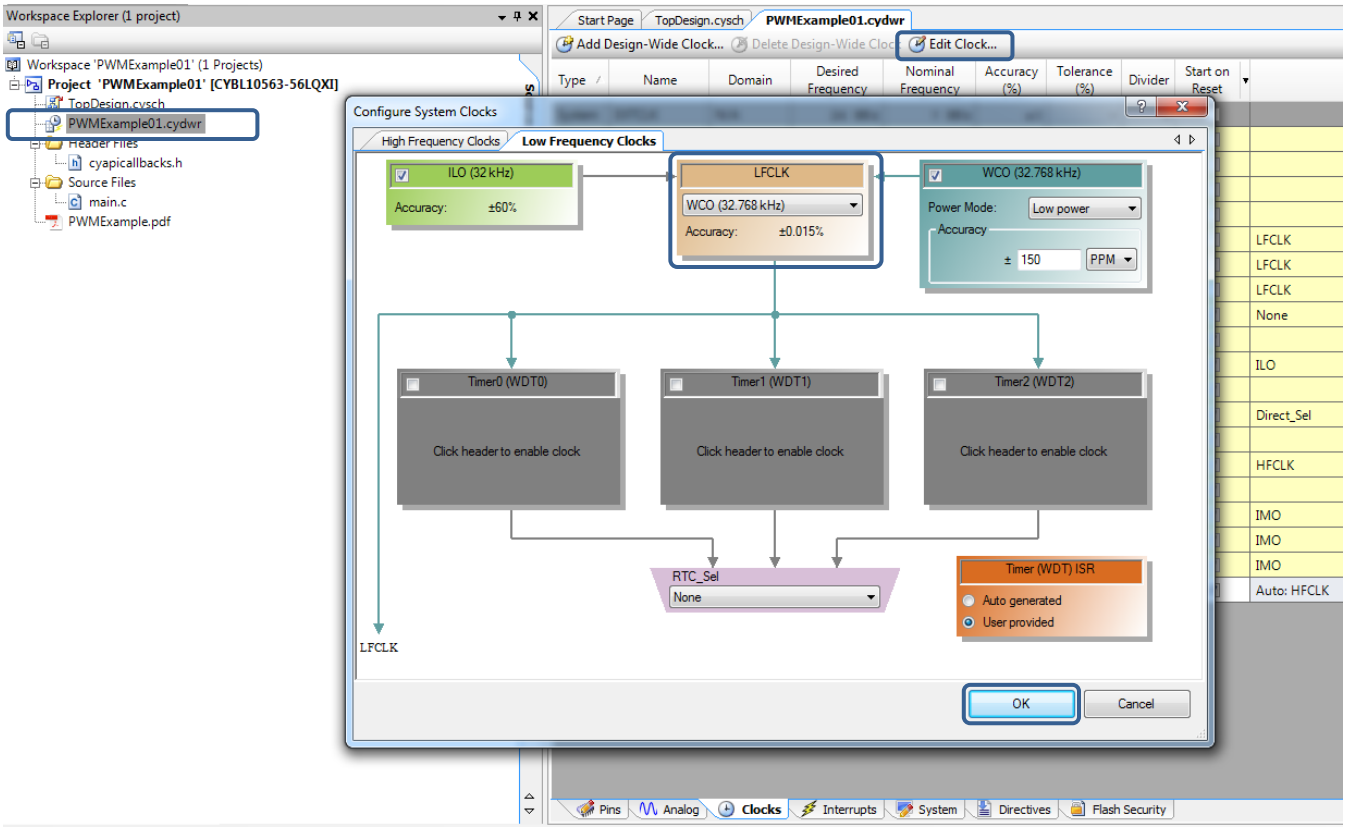
5. **Workspace Explorer** からファイルをダブルクリックして **PWMExample01.cydwr** ウィンドウを開きます。LED_GREEN のデフォルトのポート割り当てを P3[6]に変更します (表 1 を参照してください)。この変更は、プロジェクトを移植して CY8CKIT-042-BLE Pioneer Kit で動作させるために必要です。ポート P3[6]はキット上の緑色 LED に接続されます。

   ターゲット アプリケーションに適切なデバイスを変更／選択するために「他のデバイス選択」の手順に従ってください。
6. **LFCLK** ソースを **WCO (32.768kHz)** に変更します。
   a. **PWMExample01.cydwr** を開いて、**Clocks** タブに移動して、**Edit Clock...**をクリックして、**Configure System Clocks** ダイアログを開きます。
   b. **Configure System Clocks** ダイアログで、**Low Frequency Clocks** タブに移動して、LFCLK ソースを変更します (図 9 を参照してください)。
7. プロジェクトを保存します。

表 1. PWMExample01 用のピン マッピング

ピン名	ポートの割り当て
LED_GREEN	P3[6]

図 9. LFCLK の選択



ここで、ターゲット BLE デバイスで動作できる基本的な PWM サンプル プロジェクトをセットアップできました。サンプル プロジェクトがどのように動作するかを理解するために、ターゲット デバイスをビルドし、プログラムできます (ビルドとプログラムを同時に行うために **Debug** > **Program** を選択します)。PSoC Creator と CY8CKIT-042-BLE Pioneer Kit のプログラミングの詳細は、AN91267 と AN94020 を参照してください。

LED の動作は、*main.c* にある `BRIGHTNESS_DECREASE` マクロの値を変更することによって制御できます。このマクロ値は 0～63000 の間で定義できます。値が小さいほど輝度調光サイクル速度が遅くなり、値が高いほど輝度調光サイクル速度が速くなります。

5.2 外部メモリ OTA ブートローダの追加

「基本的なサンプル ターゲット プロジェクトの作成」で用意した PWMExample プロジェクトに外部メモリ OTA ブートローダを追加する方法について説明します。BLE 外部メモリ ブートローダおよびブートローダブル サンプル プロジェクトのデータシートおよびソースコードを参照することも必要です。

CY8CKIT-042-BLE Pioneer Kit では、I²C ベースの F-RAM が外部メモリとして使用されます。ただし、外部メモリ OTA ブートローダは I²C や SPI などのインターフェースを使用するフラッシュなどの他のメモリ種類でも動作できます。SPI ベースの外部メモリ OTA ブートローダの例はこちらを参照してください。

PSoC 4 BLE Pioneer Kit 用の PWMExample プロジェクトで I²C ベースの外部メモリ ブートローダをセットアップするために以下の手順に従ってください。

1. PSoC Creator で作成した PWMExample01 プロジェクトを開きます (「基本的なサンプル ターゲット プロジェクトの作成」を参照してください)。
2. PWMExample01 ワークスペースに BLE 外部メモリ ブートローダの例を追加します (「既存のワークスペースへのサンプル プロジェクトの追加」を参照してください)。
3. **Workspace Explorer** で BLE_External_Memory_Bootloader01 プロジェクトを右クリックして **Set As Active Project** を選択することにより、このプロジェクトをアクティブなプロジェクトとして設定します。
4. ターゲット アプリケーションに適切なデバイスを変更／選択するために「他のデバイス選択」の手順に従ってください。
5. **Build** > **Build BLE_External_Memory_Bootloader01** を選択して BLE_External_Memory_Bootloader01 プロジェクトをビルドします。プロジェクトはエラーなしでビルドされます。
6. PSoC Creator の他のインスタンスを開き、BLE 外部メモリ ブートローダブル サンプル プロジェクト用に新しいワークスペースを作成します (「サンプル プロジェクト ワークスペースの作成」を参照してください)。外部メモリ ブートローダを有効にするために、このプロジェクトからのファイルとコードが必要です。
7. PWMExample01 プロジェクトをアクティブなプロジェクトとして設定します。
8. 次のコンポーネントを、ステップ 6 で作成した BLE 外部メモリ ブートローダブル プロジェクトの *TopDesign.cysch* から、「基本的なサンプル ターゲット プロジェクトの作成」で作成した PWMExample01 プロジェクトの *TopDesign.cysch* ファイルにコピーします。このステップで追加されるコンポーネントのコンフィギュレーションは BLE 外部メモリ ブートローダブル サンプル プロジェクトのデータシートで説明されています。

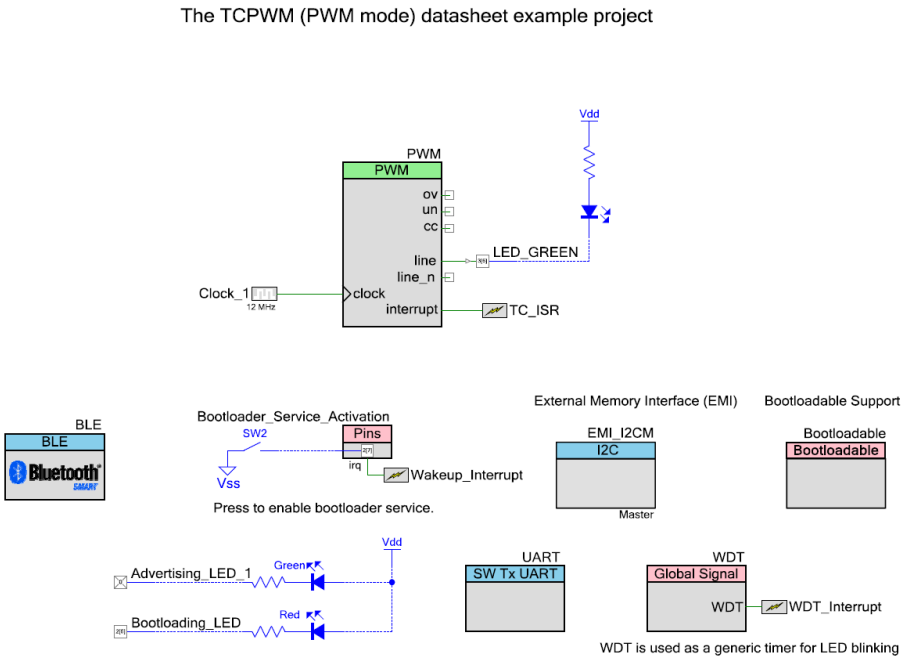
   - BLE
   - Bootloader_Service_Activation
   - Bootloading_LED
   - Advertising_LED_1
   - Bootloadable
   - EMI_I2CM
   - UART
   - WDT
   - WDT_Interrupt
   - Wakeup_Interrupt

   ユーザーのプロジェクトには BLE コンポーネントがすでに含まれている場合、このステップの BLE コンポーネントを省略し、その代わりに、「ブートローダ サービスの追加」で述べた手順に従って、既存の BLE コンポーネントのブートローダ サービスを追加し、設定してください。

   外部メモリ OTA を実装するために BLE、EMI_I2CM、および Bootloadable コンポーネントが必要です。Bootloader_Service_Activation および Wakeup_Interrupt は、ブートローダ モードに入るためのトリガーとして使用されます。ユーザーのプロジェクトにはブートローダ モードに入る他のメカニズムがある場合、Bootloader_Service_Activation および Wakeup_Interrupt コンポーネントをコピーしないでください。UART コンポーネントはデバッグ メッセージを表示するために使用されます。他のすべてのコンポーネントは外部メモリ OTA の実装に不要であるため、このステップを無視してもかまいません。しかし、これらはコンパイル時のエラーを防ぐために必要です。

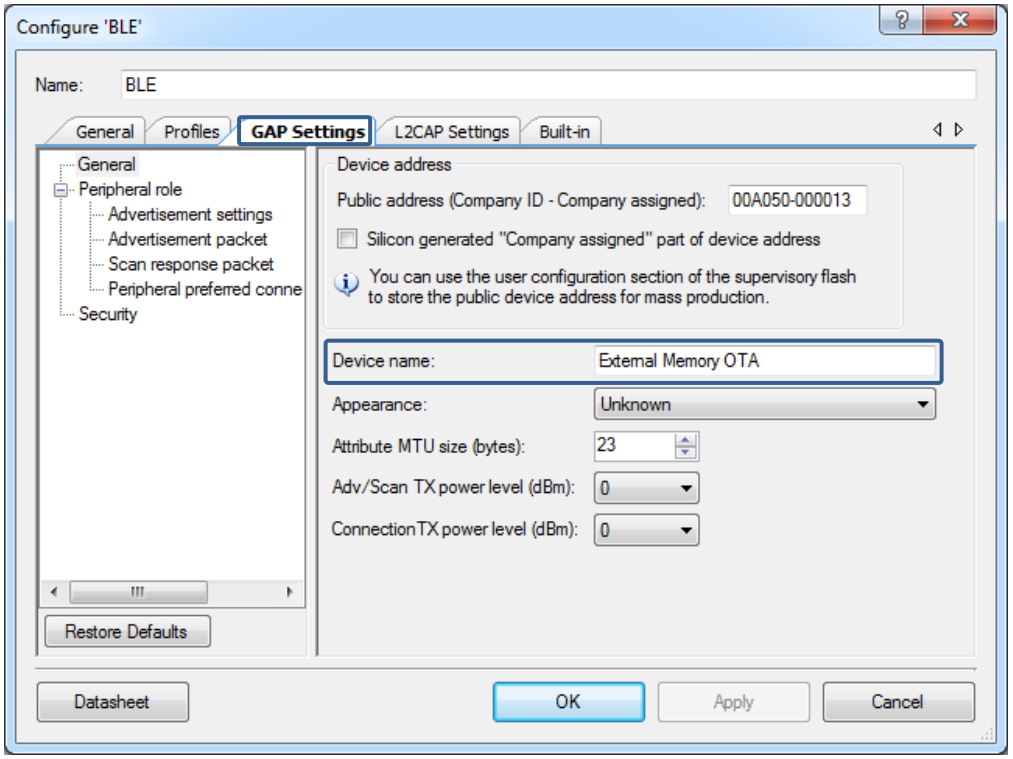
   このステップの終わりにおいて、ユーザーの回路図は図 10 のようになります。

図 10. 必要なコンポーネントを追加した後の TopDesign.cysch ビュー



9. BLE コンポーネント コンフィギュレーション ウィンドウ（BLE コンポーネントをダブルクリックしてください）の **GAP Settings** タブで、**Device Name** を「External Memory OTA」に変更します 。図 11 を参照してください。

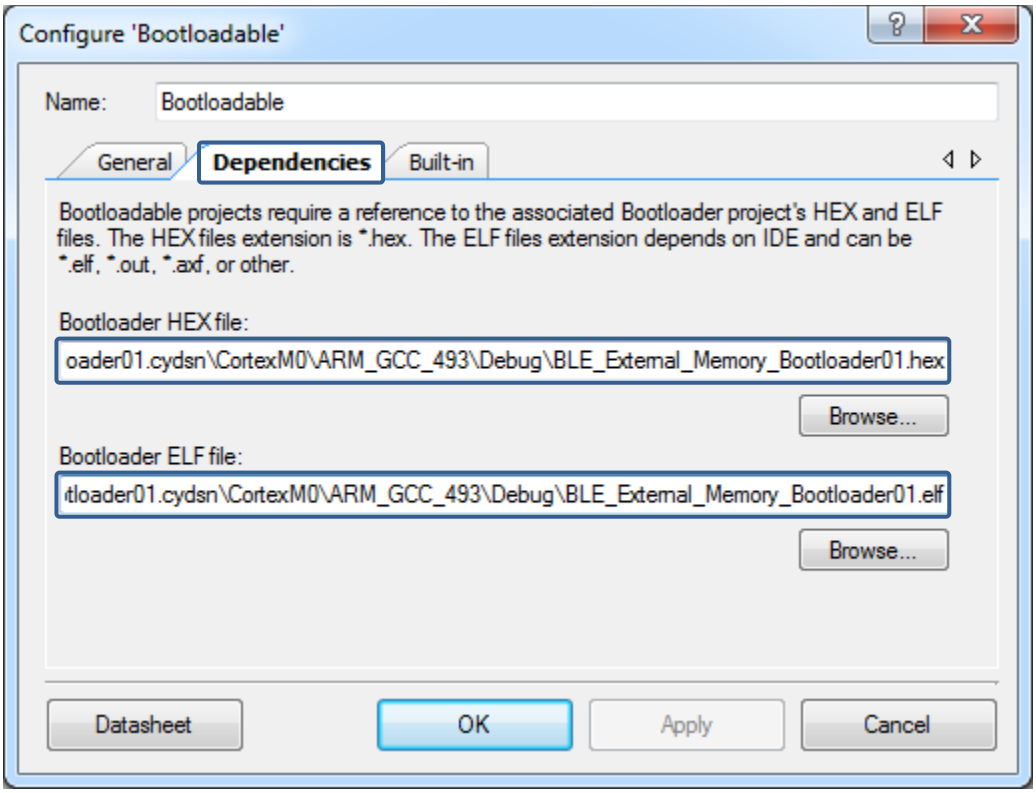
図 11. BLE コンポーネント コンフィギュレーション ウィンドウの GAP 設定



10. ブートローダ プロジェクトの HEX および ELF ファイルへのパスを指定します。

a. Bootloadable コンポーネントをダブルクリックします。
b. 図 12 に示すように、**Dependencies** タブに移動し、**Bootloader HEX file** を *BLE_External_Memory_Bootloader01.hex* ファイル (*…\BLE_External_Memory_Bootloader01.cydsn\CortexM0\<compiler version>\<build configuration>*にある) にリンクさせます。
HEX ファイルを選択すると、対応する ELF ファイルは自動的に選択されます。
c. **OK** をクリックして Bootloadable コンポーネント コンフィギュレーション ダイアログを閉じます。

図 12. Bootloadable コンポーネント コンフィギュレーション



11. ステップ 8 で追加したコンポーネントに正しいピンを割り当てるために、*PWMExample01.cydwr* を開いて **Pins** タブに移動します。表 2 のとおりにピンを設定します。

表 2. PWMExample01 用のピン マッピング

ピン名	ポートの割り当て
EMI_I2CM:scl	P5[1]
EMI_I2CM:sda	P5[0]
UART:tx	P1[5]
Advertising_LED_1	P3[7]
Bootloader_Service_Activation	P2[7]
Bootloading_LED	P2[6]
LED_GREEN	P3[6]

12. 次のファイルをブートローダブル プロジェクト ワークスペースのディレクトリから PWMExample01 プロジェクトのワークスペースのディレクトリにコピーし、これらを PWMExample01 プロジェクトに追加します。これらのファイルは OTA とデバッグ機能の一部を実装します。

a. ヘッダ ファイルを追加するには、**Workspace Explorer** 内の **Header Files** フォルダを右クリックして、**Add** > **Existing Item…**を選択します。必要なファイルを選択して、**Open** をクリックします。

b. ソース ファイルを追加するには、**Workspace Explorer** 内の **Source Files** フォルダを右クリックし、**Add** > **Existing Item…**を選択します。必要なファイルを選択して、**Open** をクリックします。

- *Common.h*
- *debug.h*
- *main.h*
- *Options.h*
- *OTAMandatory.h*
- *OTAOptional.h*
- *Common.c*
- *debug.c*
- *OTAMandatory.c*
- *OTAOptional.c*

OTAMandatory.c／h ファイルは外部メモリ OTA を有効にするために必要な機能をすべて実装します。他のファイルは必須ではありませんが、コンパイル時のエラーを防ぐためにコピーされます。*debug.h／c* ファイルは、UART ベースのデバッグ メッセージの表示を実装します。*Common.c／h* ファイルはウォッチドッグ タイマー (WDT)、LED、デバッグ メッセージ表示およびブートローダ サービス可視化の設定のためのヘルパー関数を実装します。*main.h* ファイルには、LED の状態およびブートローダ サービスの有効化／無効化の定義が含まれています。*Options.h* ファイルには、デバッグおよび暗号化を有効／無効にする定義が含まれています。*OTAOptional.c／h* ファイルは外部メモリに格納されている情報の暗号化／復号を実装します。このために、*Options.h* の `ENCRYPT_ENABLED` マクロを `YES` に設定します。

13. ブートロードまたは OTA 機能を有効にするために、追加のコード (BLE 外部メモリ ブートローダブル プロジェクトからのコード) を PWMExample01 プロジェクトの *main.c* に追加する必要があります。変更点が多く、個別に記載することができません。その代わりに、変更ファイルをダウンロードし、使用できます。PWMExample01 プロジェクトの *main.c* ファイルを*…\Code\External Memory OTA*の *main.c* ファイルに置き換えます。
14. Bootloader／Bootloadable コンポーネントを追加した後、デバッグの対応は PSoC Creator で無効になります。しかし、ファームウェアが実行を開始すると、ファームウェアをデバッグするために、ターゲット デバイスに接続し (**Debug** > **Attach to Running Target…**)、または UART メッセージングを使用します。UART が有効のとき、プロジェクトが正常に実行されるように十分なヒープ (0x400 バイト) とスタック (0×800 バイト) が設定されたことを確認してください。そうしないと、ファームウェアは予測できない動作となる場合があります。詳細は「デバッグ」を参照してください。
15. **Build** > **Build PWMExample01** を選択し、PWMExample01 プロジェクトをビルドします。PWMExample01 プロジェクトはエラーなしでビルドされます。
16. **Debug** > **Program** を選択し、PSoC BLE Pioneer Kit をプログラムします。

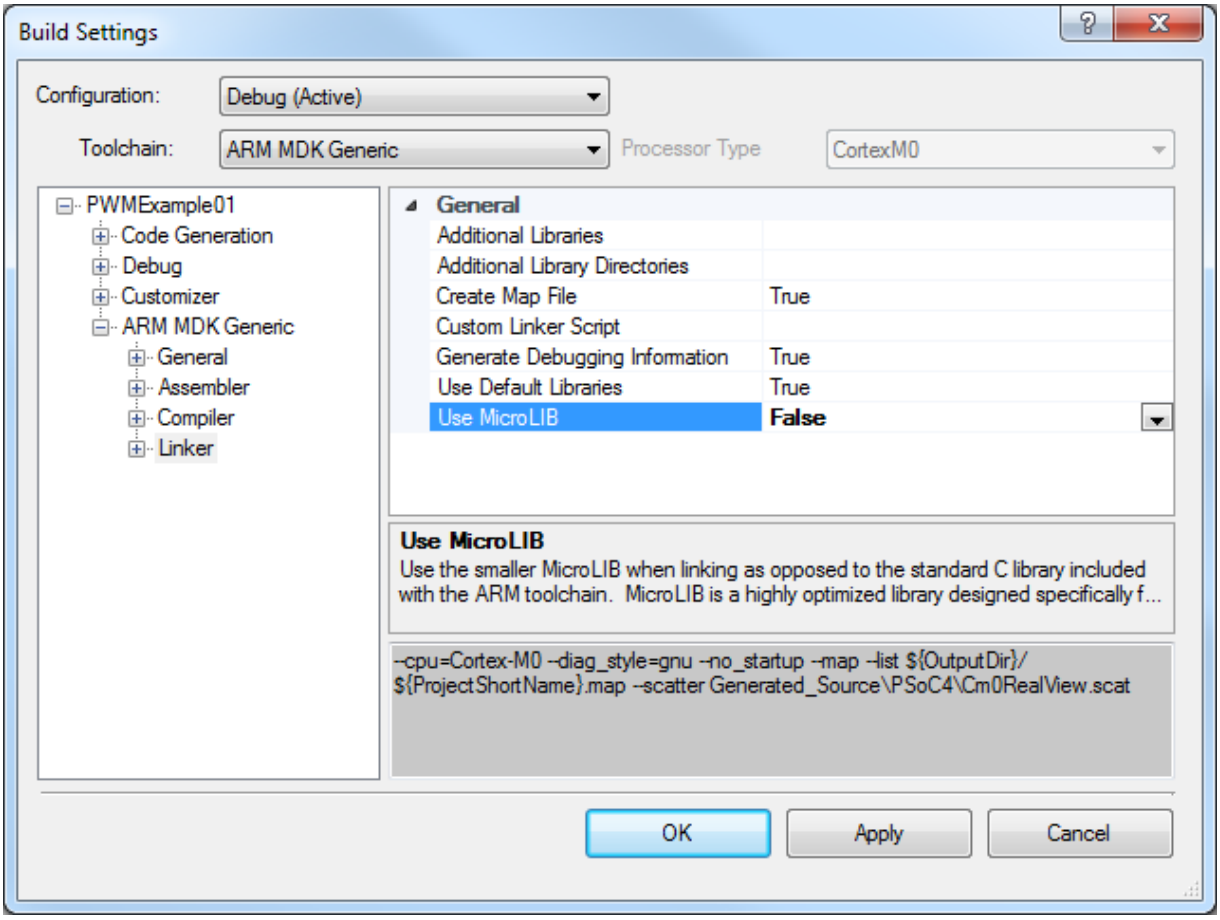
プログラミングが完了した後、緑色 LED は高輝度から低輝度まで周期的に繰り返します。ここで、「OTA アップグレードの実行」で述べた方法の 1 つで、デバイス ファームウェアのアップグレードを実行できます。また、「OTA 機能のテスト」で述べた手順に従って、OTA 機能もテストできます。

5.3 固定スタック OTA ブートローダの追加

「基本的なサンプル ターゲット プロジェクトの作成」で用意した PWMExample プロジェクトに固定スタック OTA ブートローダを追加する方法について説明します。また、BLE 固定スタック ブートローダおよびブートローダブル サンプル プロジェクトのデータシートおよびソースコードも参照する必要があります。PWMExample プロジェクトで固定スタック ブートローダをセットアップするために以下の手順に従ってください。

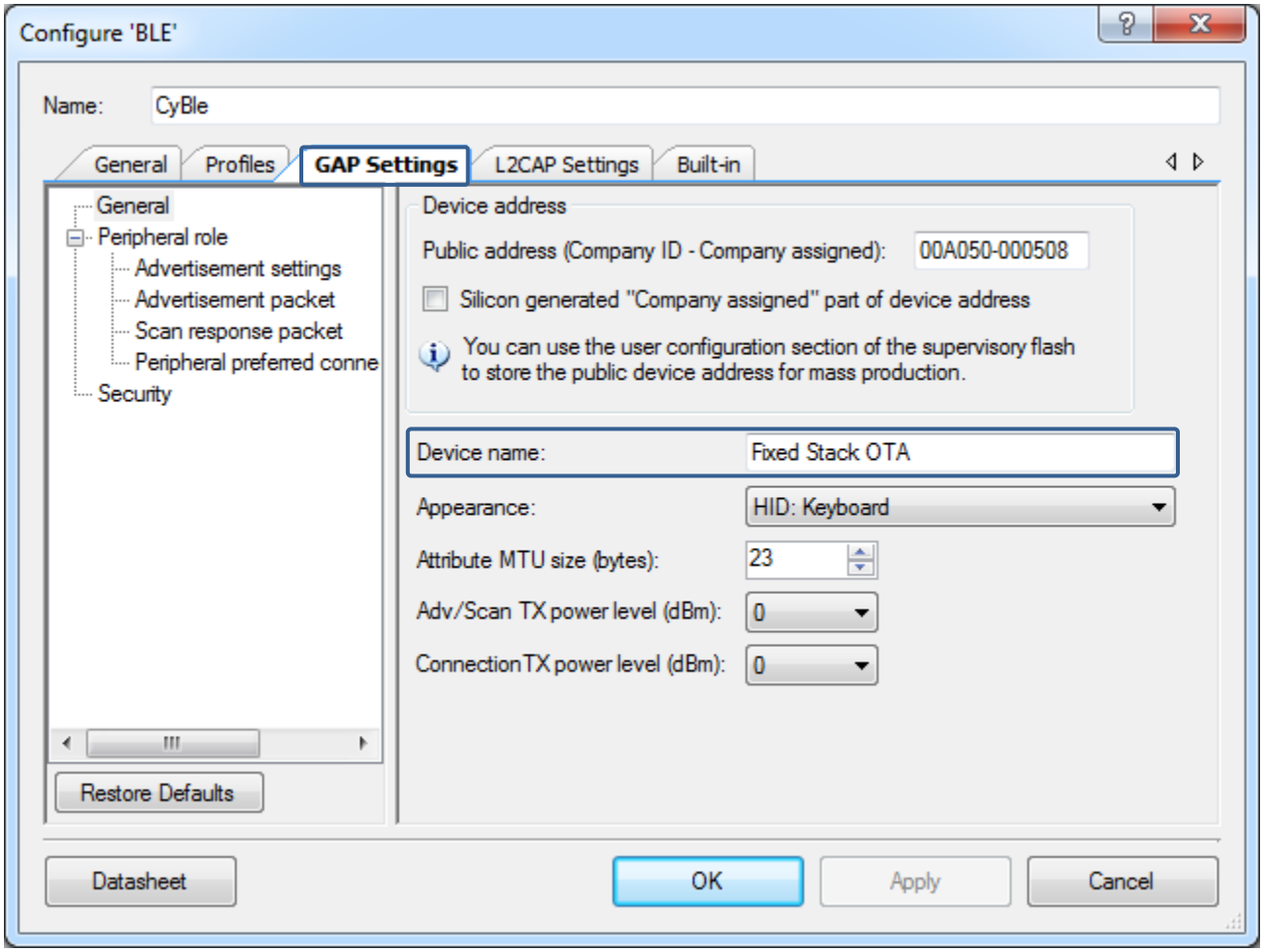
注: 固定スタック OTA ブートローダのコード共有機能は、MDK MicroLIB リンカー オプションが選択された場合にサポートされません。そのため、**Project** > **Build Settings** の **Use MicroLIB** は **False** に設定する必要があります (図 13 を参照してください)。

図 13. MDK リンカーのビルド設定



1. PSoC Creator で作成した PWMExample01 プロジェクトを開きます (「基本的なサンプル ターゲット プロジェクトの作成」を参照してください)。
2. PWMExample01 ワークスペースに BLE_OTA_FixedStack_Bootloader の例を追加します (「既存のワークスペースへのサンプル プロジェクトの追加」を参照してください)。
3. **Workspace Explorer** で BLE_OTA_FixedStack_Bootloader01 プロジェクトを右クリックし、**Set As Active Project** を選択することにより、このプロジェクトをアクティブなプロジェクトとして設定します。
4. ターゲット アプリケーションに適切なデバイスを変更／選択するために「他のデバイス選択」の手順に従ってください。
5. BLE コンポーネント コンフィギュレーション ウィンドウの **GAP Settings** タブで、**Device Name** を「Fixed Stack OTA」に変更します (図 14 を参照してください)。BLE コンポーネント コンフィギュレーション ダイアログを開くには、BLE コンポーネントをダブルクリックします。

図 14. BLE コンポーネント コンフィギュレーション ウィンドウの GAP 設定



ユーザーのブートローダブル プロジェクトには BLE コンポーネントが既にある場合、次のようにしてください。

a. BLE_OTA_FixedStack_Bootloader01 プロジェクトの BLE コンポーネントをユーザーのブートローダブル プロジェクトの BLE コンポーネントに置き換えます。

b. 既存の BLE コンポーネントにブートローダ サービスを追加し、設定します (「ブートローダ サービスの追加」を参照してください)。

c. ブートローダブル プロジェクトの回路図から BLE コンポーネントを削除します。

BLE コンポーネントの置き換えにより、新しい BLE コンポーネント コンフィギュレーションに応じて、コードに関連するサービス／プロファイルの一部は、コンパイル時のエラーを防ぐために削除または追加する必要があります。

6. BLE OTA 固定スタックのサンプル プロジェクトはカスタム リンカー スクリプトを使用し、選択したデバイス用に設定される必要があります。このために、「サイプレスの他の BLE デバイス用の固定スタック OTA プロジェクトの設定」で述べた手順に従ってください。
7. BLE_OTA_FixedStack_Bootloader01 プロジェクトをビルドします。プロジェクトはエラーなしでビルドされます。
8. *PWMExample01.cydsn* プロジェクト フォルダに新しいフォルダを作成し、「LinkerScripts」と名前を付けます。
9. PSoC Creator で、*mk.bat* ファイル (**Workspace Explorer** > **BLE_OTA_FixedStack_Bootloader01** > **Scripts** > **mk.bat**) をダブルクリックして開きます。
10. 表 3 のとおりにファイルを編集し、保存します。`LOADABLE_PRJ_NAME` はブートローダブル アプリケーション名で割り当てる必要があります (この場合は「PWMExample01」です)。

表 3. *mk.bat* の変更

行番号	変数	値
28	LOADER_PRJ_NAME	BLE_OTA_FixedStack_Bootloader01
30	LOADABLE_PRJ_NAME	PWMExample01

11. **Windows Explorer** から *mk.bat* ファイルを実行します。このファイルは *…\PWMExample01\BLE_OTA_FixedStack_Bootloader01.cydsn\Scripts* にあります。このバッチ ファイルはエラーなしで実行されます。バッチ ファイルの実行が完了した後、任意のキーを押してウィンドウを閉じます。このステップでは、*…\PWMExample01\PWMExample01.cydsn\LinkerScripts* の下に *BootloaderSymbolsGcc.ld* ファイルが作成されます。
12. PSoC Creator の他のインスタンスを開いて、BLE_OTA_FixedStack_Bootloadable サンプル プロジェクト用の新しいワークスペースを作成します (「サンプル プロジェクト ワークスペースの作成」を参照してください)。固定スタック ブートローダを有効にするために、このプロジェクトからのファイルとコードが必要です。
13. PWMExample01 プロジェクトをアクティブなプロジェクトとして設定します。
14. PWMExample01 プロジェクト用に PSoC Creator の **Workspace Explorer** で新しいフォルダを作成し、「LinkerScripts」と名前を付けます。新しいフォルダを作成するには、**Workspace Explorer** 内のプロジェクト名を右クリックして、**Add** > **New Folder** を選択します。
15. ステップ 12 で作成したブートローダブル サンプル プロジェクトには LinkerScripts フォルダがあります。このフォルダには、PSoC Creator がサポートするすべての 3 つのコンパイラ用のリンカー スクリプト (*cm0gcc.ld*、*Cm0Iar.icf*、および *Cm0Mdk.scat*) が含まれています。ステップ 8 で PWMExample01 プロジェクトで作成された LinkerScripts フォルダに 3 つのファイルをすべてコピーします。
16. PSoC Creator の PWMExample01 プロジェクトの **Workspace Explorer** 内の (ステップ 14 で作成した) LinkerScripts フォルダに以下のファイルを追加します。ファイル システムでは、これらのファイルはステップ 8 で作成した LinkerScripts フォルダにあります。
    - *BootloaderSymbolsGcc.ld*
    - *cm0gcc.ld*
    - *Cm0Iar.icf*
    - *Cm0Mdk.scat*
17. BLE OTA 固定スタックのサンプル プロジェクトはカスタム リンカー スクリプトを使用し、選択したデバイス用に設定される必要があります。このために、「サイプレスの他の BLE デバイス用の固定スタック OTA プロジェクトの設定」で述べた手順に従ってください。
18. 次のコンポーネントを、ステップ 12 で作成した BLE_OTA_FixedStack_Bootloadable サンプル プロジェクトの *TopDesign.cysch* から、「基本的なサンプル ターゲット プロジェクトの作成」で作成した PWMExample01 プロジェクトの *TopDesign.cysch* ファイルにコピーします。このステップで追加したコンポーネントのコンフィギュレーションは BLE 固定スタック ブートローダブル サンプル プロジェクトのデータシートで説明されています。コンポーネントは複数の回路図ページにわたって広がる可能性があります。
    - Bootloadable
    - UART_DEB
    - WDT
    - WDT_Interrupt

    最終のブートローダブル プロジェクトの回路図には BLE コンポーネントがないことを確認してください。ブートローダブル プロジェクトに BLE コンポーネントが既にある場合、次のようにします。
    a. そのコンポーネントを BLE_OTA_FixedStack_Bootloader01 プロジェクトに移動します。
    b. 既存の BLE コンポーネントにブートローダ サービスを追加し、設定します (「ブートローダ サービスの追加」を参照してください)。

    このステップの後に、ステップ 5 から繰り返して、ブートローダを更新し、新しいリンカー スクリプトを生成します。ブートローダブル コンポーネントは固定スタック OTA を実装するために使用されます。他のすべてのコンポーネントは重要ではないため、このステップで無視してもかまいません。しかし、これらはコンパイル時のエラーを防ぐために必要です。UART コンポーネントはデバッグ メッセージを表示するために使用されます。必要であれば、WDT と WDT_Interrupt はプロジェクトでタイミングを実装します。

このステップの終わりにユーザーの回路図は図 15 のようになります。ブートローダ プロジェクトの HEX ファイルと ELF ファイルへのパスを指定します。

a. Bootloadable コンポーネントをダブルクリックします。

b. 図 16 に示すように、**Dependencies** タブに移動し、**Bootloader HEX file** を *BLE_OTA_FixedStack_Bootloader01.hex* ファイル (*…\BLE_OTA_FixedStack_Bootloader01.cydsn\CortexM0\\<compiler version>\<build configuration>*にある) にリンクさせます。

c. **OK** をクリックして Bootloadable コンポーネント コンフィギュレーション ダイアログを閉じます。

HEX ファイルを選択すると、対応する ELF ファイルは自動的に選択されます。

19. ステップ 17 で追加したコンポーネント用に正しいピンを割り当てるために、*PWMExample01.cydwr* を開き、**Pins** タブに移動します。表 4 に示すようにピンを設定します。

図 15. 必要なコンポーネントを追加した後の TopDesign.cysch ビュー

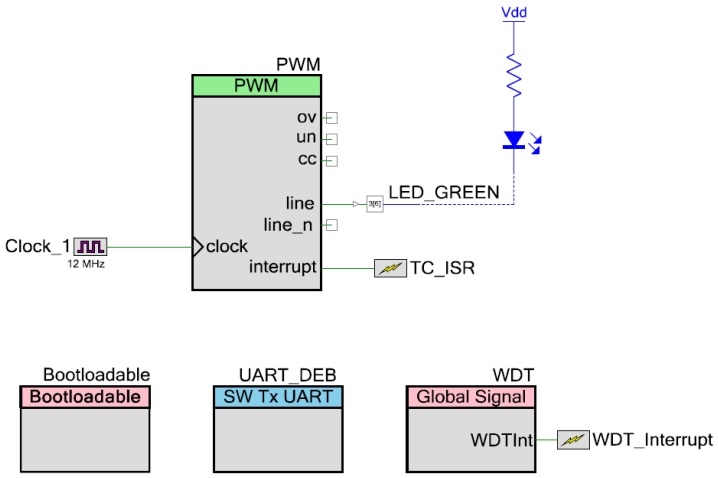


図 16. Bootloadable コンポーネント コンフィギュレーション

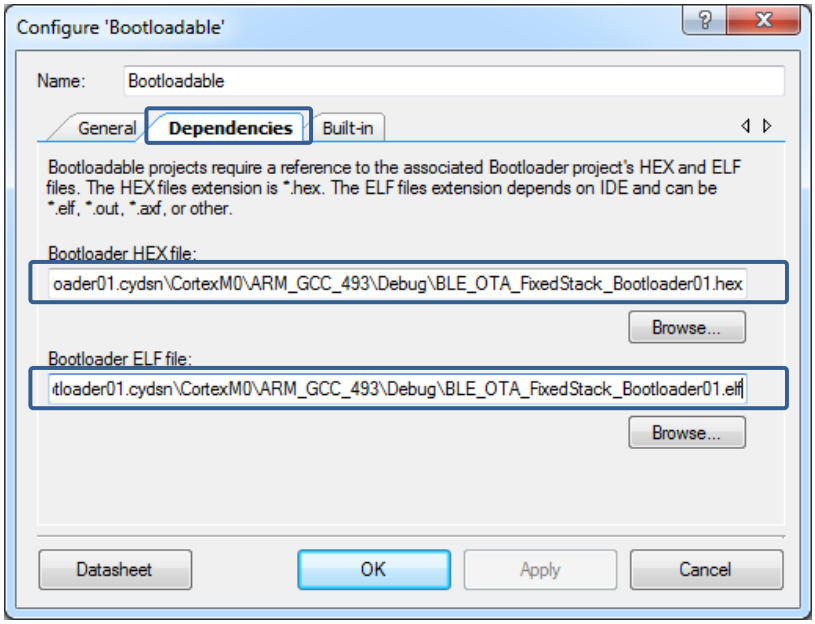


表 4. PWMExample01 用のピン マッピング

ピン名	ポートの割り当て
UART_DEB:tx	P1[5]
LED_GREEN	P3[6]

20. 次のファイルを BLE 固定スタック ブートローダブル サンプル プロジェクトのワークスペース ディレクトリから PWMExample01 プロジェクトのワークスペース ディレクトリにコピーし、それらを PWMExample01 プロジェクトに追加します。これらのファイルは OTA とデバッグ機能の一部を実装します。

    a. ヘッダ ファイルを追加するには、**Workspace Explorer** 内の **Header Files** フォルダを右クリックして、**Add** > **Existing Item…**を選択します。必要なファイルを選択して、**Open** をクリックします。

    b. ソース ファイルを追加するには、**Workspace Explorer** 内の **Source Files** フォルダを右クリックし、**Add** > **Existing Item…**を選択します。必要なファイルを選択して、**Open** をクリックします。

    - *common.h*
    - *main.h*
    - *OTAMandatory.h*
    - *OTAOptional.h*
    - *debug.h*
    - *Options.h*
    - *OTAMandatory.c*
    - *OTAOptional.c*
    - *debug.c*

    OTAMandatory.c／h ファイルは外部メモリ OTA を有効にするために必要な機能をすべて実装します。他のファイルは必須ではありませんが、コンパイル時のエラーを防ぐためにコピーされます。*debug.h／c* ファイルは、UART ベースのデバッグ メッセージの表示を実装します。*common.h* ファイルには LED 状態および WDT オプションの定義が含まれています。*Options.h* ファイルには、デバッグを有効／無効にする定義が含まれています。*OTAOptional.c／h* ファイルは WDT、LED およびデバッグ メッセージ表示のためのヘルパー関数を実装します。

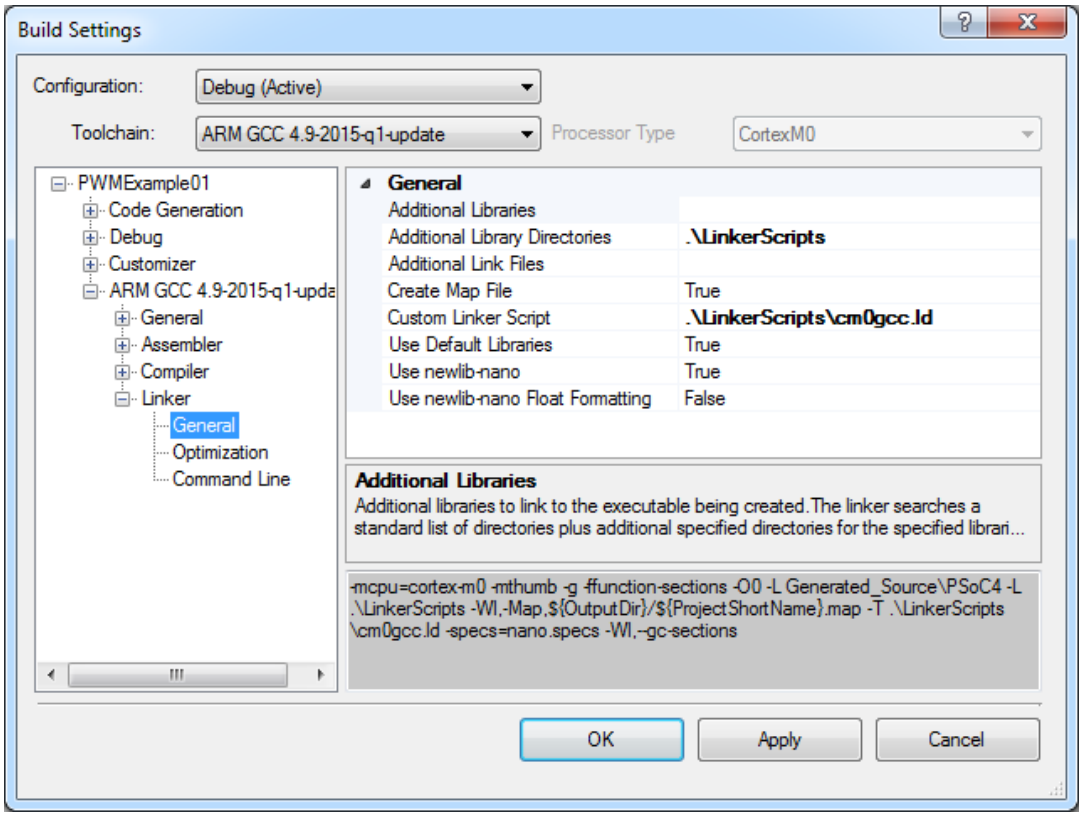
21. ブートロードまたは OTA 機能を有効にするために、追加のコード (BLE_OTA_FixedStack_Bootloadable プロジェクトからのコード) を PWMExample01 プロジェクトの *main.c* に追加する必要があります。変更点が多く、個別に記載することができません。その代わりに、変更ファイルをダウンロードし、使用できます。PWMExample01 プロジェクトの *main.c* ファイルを*…\Code\Fixed Stack OTA*の *main.c* に置き換えます。

22. 表 5 に示す設定が PWMExample01 のビルド設定に適用されたことを確認してください。これらの変更は、新しいカスタム リンカー スクリプトを使用するようにリンカーに指示します。ビルド設定を変更するには、**Project** > **Build Settings…**を選択し、図 17 に示すように、ツリー ビュー ウィンドウで **PWMExample01** > **ARM GCC 4.9-2015-q1-update** > **Linker** > **General** を選択します。

表 5. ビルド設定の変更

フィールド	値
Additional Library Directories	.\LinkerScripts
Custom Linker Script	.\LinkerScripts\cm0gcc.ld

図 17. Build Settings ダイアログのリンカー設定



23. Bootloader／Bootloadable コンポーネントを追加した後、デバッグの対応は PSoC Creator で無効です。しかし、ファームウェアが実行を開始すると、ファームウェアをデバッグするために、ターゲット デバイスに接続し (**Debug** > **Attach to Running Target...**)、または UART メッセージングを使用します。UART デバッグを有効／無効にする方法は「デバッグ」を参照してください。UART が有効のとき、プロジェクトが正常に実行されるように十分なヒープ (0x400 バイト) とスタック (0×800 バイト) が設定されたことを確認してください。そうしないと、ファームウェアは予測できない動作となる場合があります。詳細は「デバッグ」を参照してください。

24. PWMExample01 プロジェクトをビルドします。プロジェクトはエラーなしでビルドされます。

25. **Debug** > **Program** を選択し、PSoC BLE Pioneer Kit をプログラムします。

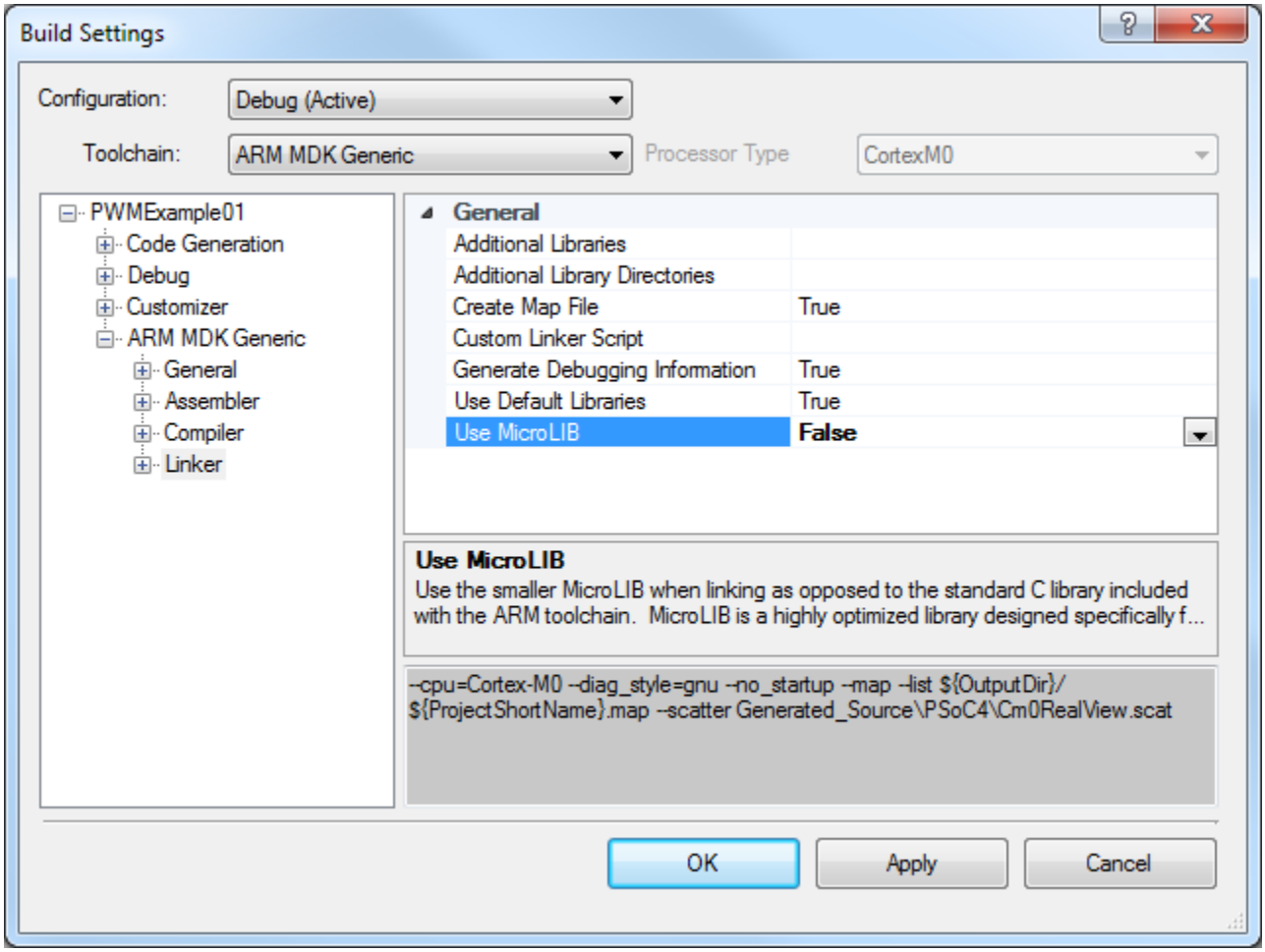
プログラミングが完了した後、緑色 LED は高輝度から低輝度まで周期的に繰り返します。ここで、「OTA アップグレードの実行」で述べた方法の 1 つで、デバイス ファームウェアのアップグレードを実行できます。また、「OTA 機能のテスト」で述べた手順に従って、OTA 機能もテストできます。

5.4 アップグレード可能スタック OTA ブートローダの追加

「基本的なサンプル ターゲット プロジェクトの作成」で用意した PWM サンプル プロジェクトにアップグレード可能スタック OTA ブートローダを追加する方法について説明します。また、BLE OTA アップグレード可能スタック ランチャー、スタックおよびキーボード サンプル プロジェクトのデータシートおよびソースコードを参照することも必要です。PWM サンプル プロジェクトでアップグレード可能スタック メモリ ブートローダをセットアップするために以下の手順に従ってください。

注: アップグレード可能スタック OTA ブートローダのコード共有機能は、MDK MicroLIB リンカー オプションが選択された場合にサポートされません。そのため、**Project** > **Build Settings** の **Use MicroLIB** は **False** に設定する必要があります (図 18 を参照してください)。

図 18. MDK リンカーのビルド設定



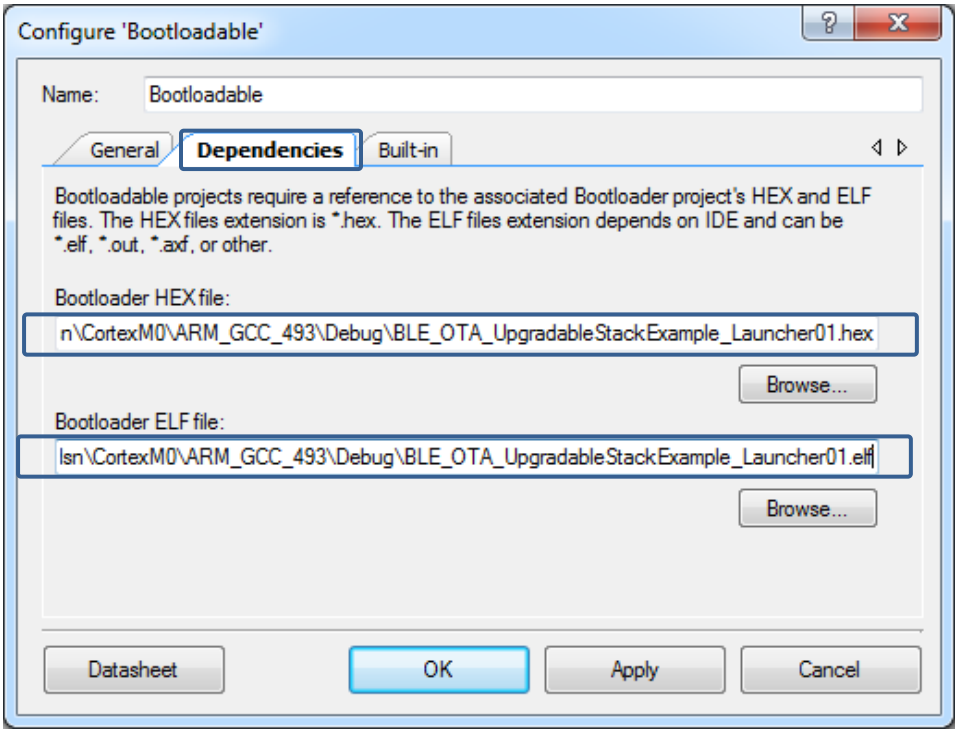
1. PSoC Creator で作成した PWMExample01 プロジェクトを開きます (「基本的なサンプル ターゲット プロジェクトの作成」を参照してください)。
2. PWMExample01 のワークスペースに BLE_OTA_UpgradableStackExample_Launcher の例を追加します (「既存のワークスペースへのサンプル プロジェクトの追加」を参照してください)。
3. BLE_OTA_UpgradableStackExample_Launcher01 プロジェクトをアクティブなプロジェクトとして設定します。このために、**Workspace Explorer** 内のプロジェクトを右クリックして、**Set As Active Project** を選択します。
4. ターゲット アプリケーションに適切なデバイスを変更／選択するために「他のデバイス選択」の手順に従ってください。
5. BLE_OTA_UpgradableStackExample_Launcher01 プロジェクトをビルドします。プロジェクトはエラーなしでビルドされます。
6. PWMExample01 のワークスペースに BLE_OTA_UpgradableStackExample_Stack の例を追加します (「既存のワークスペースへのサンプル プロジェクトの追加」を参照してください)。
7. BLE_OTA_UpgradableStackExample_Stack01 プロジェクトをアクティブなプロジェクトとして設定します。
8. ターゲット アプリケーションに適切なデバイスを変更／選択するために「他のデバイス選択」の手順に従ってください。
9. ランチャー プロジェクトの HEX および ELF ファイルへのパスを指定します。
   a. Bootloadable コンポーネントをダブルクリックします。

b. 図 19 に示すように、**Dependencies** タブに移動し、**Bootloader HEX file** を *BLE_OTA_UpgradableStackExample_Launcher01.hex* ファイル (*…\BLE_OTA_UpgradableStackExample_Launcher01.cydsn\CortexM0\\<compiler version>\<build configuration>*にある) にリンクさせます。

c. **OK** をクリックして Bootloadable コンポーネント コンフィギュレーション ダイアログを閉じます。

HEX ファイルを選択すると、対応する ELF ファイルは自動的に選択されます。

図 19. Bootloadable コンポーネント コンフィギュレーション

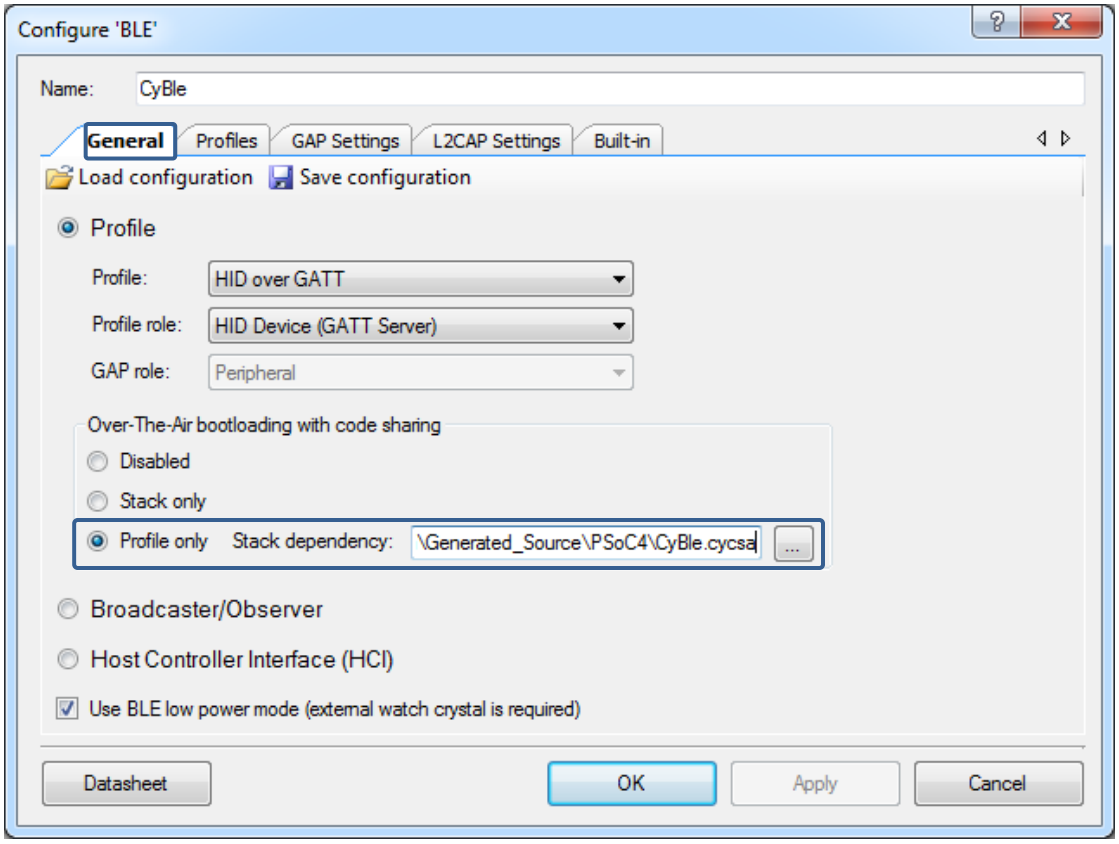


10. BLE_OTA_UpgradableStackExample_Stack01 プロジェクトをビルドします。プロジェクトはエラーなしでビルドされます。
11. PSoC Creator の他のインスタンスを開いて、BLE_OTA_UpgradableStack_HID_Keyboard サンプル プロジェクト用の新しいワークスペースを作成します (「サンプル プロジェクト ワークスペースの作成」を参照してください)。アップグレード可能スタック OTA ブートローダを有効にするために、このプロジェクトからのファイルとコードが必要です。
12. PWMExample01 プロジェクトをアクティブなプロジェクトとして設定します。
13. 次のコンポーネントを、ステップ 24 で作成した BLE_OTA_UpgradableStack_HID_Keyboard サンプル プロジェクトの *TopDesign.cysch* ファイルから、「基本的なサンプル ターゲット プロジェクトの作成」で作成した PWMExample01 プロジェクトの *TopDesign.cysch* ファイルにコピーします。このステップで追加したコンポーネント コンフィギュレーションは BLE_OTA_UpgradableStack_HID_Keyboard サンプル プロジェクトのデータシートで説明されています。

- CyBle
- Bootloadable
- UART
- SW2
- Wakeup_Interrupt

ユーザーのプロジェクトに BLE コンポーネントが既にある場合、BLE コンポーネントが最新バージョンに更新され、**Profile only** モードで設定されていることを確認してください (図 20 を参照してください)。

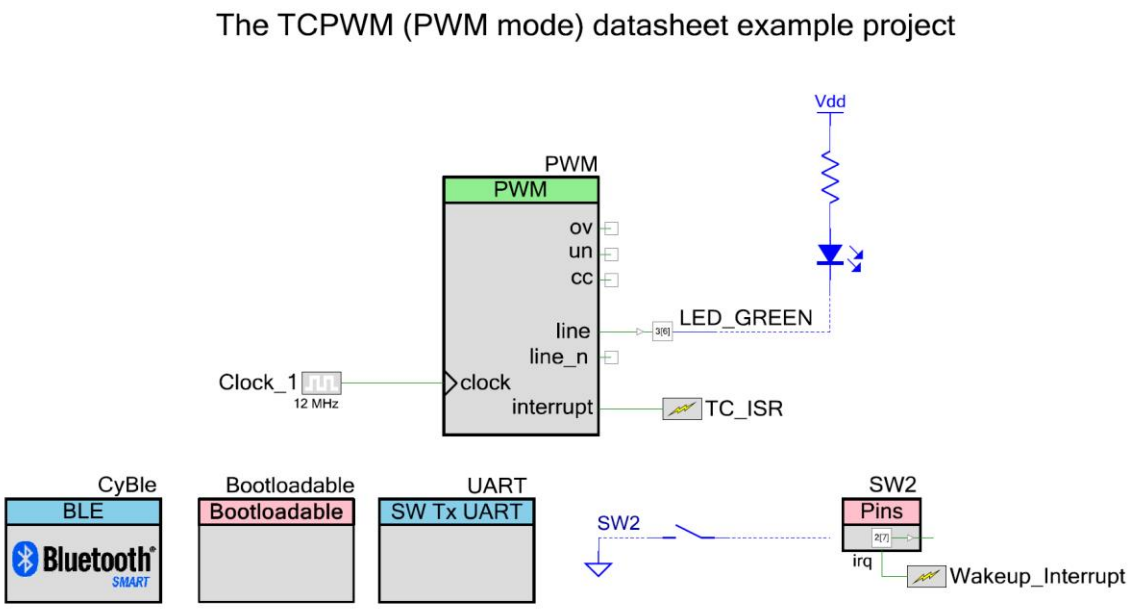
図 20. BLE コンポーネント コンフィギュレーションの Profile Only オプション



BLE およびブートローダブル コンポーネントはアップグレード可能スタック OTA を実装するために使用されます。他のすべてのコンポーネントはアップグレード可能スタック OTA の実装に不要であるため、このステップで無視してもかまいません。しかし、これらはコンパイル時のエラーを防ぐために必要です。UART コンポーネントはデバッグ メッセージを表示するために使用されます。SW2 および Wakeup_Interrupt はブートローダ モードに移行するトリガーとして使用されます。ユーザーのプロジェクトは、ブートローダ モードに移行する他のメカニズムがある場合、SW2 および Wakeup_Interrupt コンポーネントをコピーしないでください。

このステップの終わりにおいて、ユーザーの回路図は図 21 のようになります。

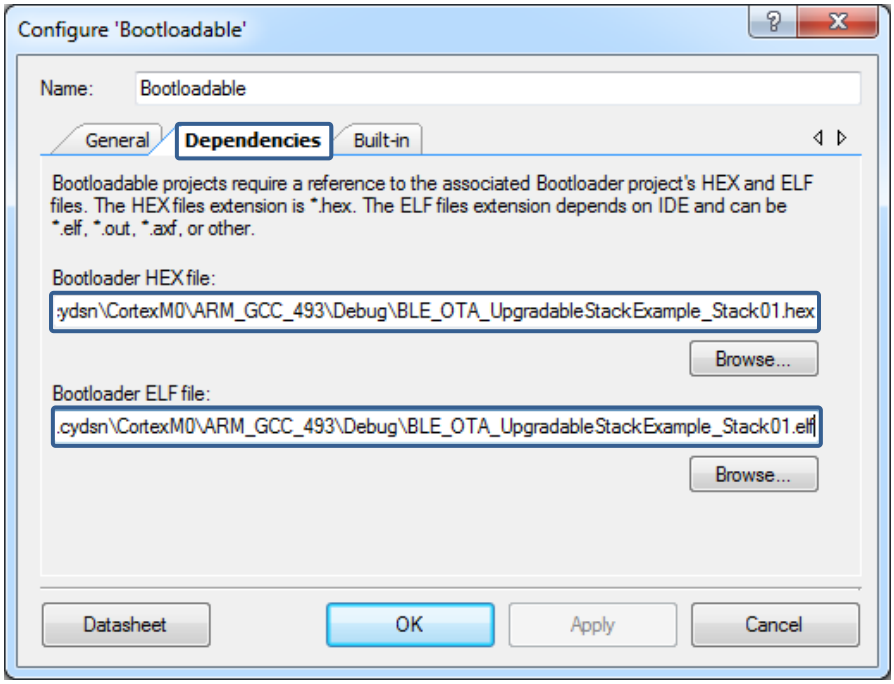
図 21. 必要なコンポーネントを追加した後の TopDesign.cysch ビュー



14. BLE コンポーネントの BLE_OTA_UpgradableStackExample_Stack01 の *CyBle.cycsa* ファイルへのパスを指定します。
    a. BLE コンポーネントをダブルクリックします。
    b. **General** タブに移動します。**Over-The-Air bootloading with code sharing** セクションで **Profile only** オプションを選択します。
    c. 図 20 に示すように、**Stack dependency** フィールドの *CyBle.cycsa* ファイルを選択します。
    d. *CyBle.cycsa* ファイルは BLE_OTA_UpgradableStackExample_Stack01 プロジェクト ディレクトリ下の *…\Generated_Source\PSoC4*ディレクトリにあります。
15. ランチャー プロジェクトの HEX と ELF ファイルへのパスを指定します。
    a. Bootloadable コンポーネントをダブルクリックします。
    b. 図 22 に示すように、**Dependencies** タブに移動し、**Bootloader HEX file** を *BLE_OTA_UpgradableStackExample_Stack01.hex* ファイル (*…\BLE_OTA_UpgradableStackExample_Stack01.cydsn\CortexM0\\<compiler version>\<build configuration>*にある) にリンクさせます。
    c. **OK** をクリックして Bootloadable コンポーネント コンフィギュレーション ダイアログを閉じます。

    HEX ファイルを選択すると、対応する ELF ファイルは自動的に選択されます。

図 22. Bootloadable コンポーネント コンフィギュレーション



16. ステップ 17 で追加したコンポーネント用に正しいピンを割り当てるために、*PWMExample01.cydwr* を開き、**Pins** タブに移動します。表 6 のとおりにピンを設定します。

表 6. PWMExample01 用のピン マッピング

ピン名	ポートの割り当て
UART:tx	P1[5]
LED_GREEN	P3[6]
SW2	P2[7]

17. 次のファイルを BLE_OTA_UpgradableStack_HID_Keyboard サンプル プロジェクトのワークスペース ディレクトリから PWMExample01 プロジェクトのワークスペース ディレクトリにコピーし、それらを PWMExample01 プロジェクトに追加します。これらのファイルは OTA とデバッグ機能の一部を実装します。

    a. ヘッダ ファイルを追加するには、**Workspace Explorer** 内の **Header Files** フォルダを右クリックして、**Add** > **Existing Item…**を選択します。必要なファイルを選択して、**Open** をクリックします。

    b. ソース ファイルを追加するには、**Workspace Explorer** 内の **Source Files** フォルダを右クリックし、**Add** > **Existing Item…**を選択します。必要なファイルを選択して、**Open** をクリックします。

    - *OTAMandatory.h*
    - *debug.h*
    - *common.h*
    - *options.h*
    - *OTAMandatory.c*
    - *debug.c*

    OTAMandatory.c／h のファイルはアップグレード可能スタック OTA を有効にする機能をすべて実装します。他のファイルは必須ではありませんが、コンパイル時のエラーを防ぐためにコピーされます。*debug.h*／*c* ファイルは、UART ベースのデバッグ メッセージの表示を実装します。*common.h* ファイルには LED 状態および WDT オプションの定義が含まれています。*options.h* ファイルには、デバッグを有効／無効にする定義が含まれています。

18. ブートロードまたは OTA 機能を有効にするために、追加のコード (BLE_OTA_UpgradableStack_HID_Keyboard プロジェクトからのコード) を PWMExample01 プロジェクトの *main.c* に追加する必要があります。変更ファイルをダウンロードできます。PWMExample01 プロジェクトの *main.c* ファイルを*…\Code\Upgradable Stack OTA*からの *main.c* に置き換えます。下記は (OTA に) 関連するコード部分の検証です。

   `AfterImageUpdate()` 関数は、アプリケーション イメージが更新され、初めて実行されるかどうかを確認します。アプリケーション イメージが初めて実行され、BLE コンポーネントの「Bonding requirement」オプションが「Bonding」に設定されている場合、この関数はボンディング データを検証し、無効の場合にはデータを消去します。また、未使用のメタデータ領域内で更新検出フラグをセットアップします。

   `InitializeBootloaderSRAM()` 関数は、コード共有に必要とされる BLE スタック SRAM を初期化するために使用されます。この関数は main 関数の初めに呼び出す必要があります。そうしないと、ファームウェアは予測できない動作となる可能性があります。

   アプリケーション イメージをスタック アプリケーションに切り替えるために、まず、入力パラメーター 0 で `Bootloadable_SetActiveApplication()` を呼び出してスタックをアクティブなアプリケーションとして設定します。次に、`Bootloadable_Load()` 関数を呼び出します。その後、`CySoftwareReset()` 使ってソフトウェア リセットを行います。このシーケンスはコード 1 に示されます。このコード例では、SW2 の押下と解放はブートロード プロセスを開始するためのトリガーとして使用されます。

コード 1. アプリケーション レベルのボンディング情報の書き込み

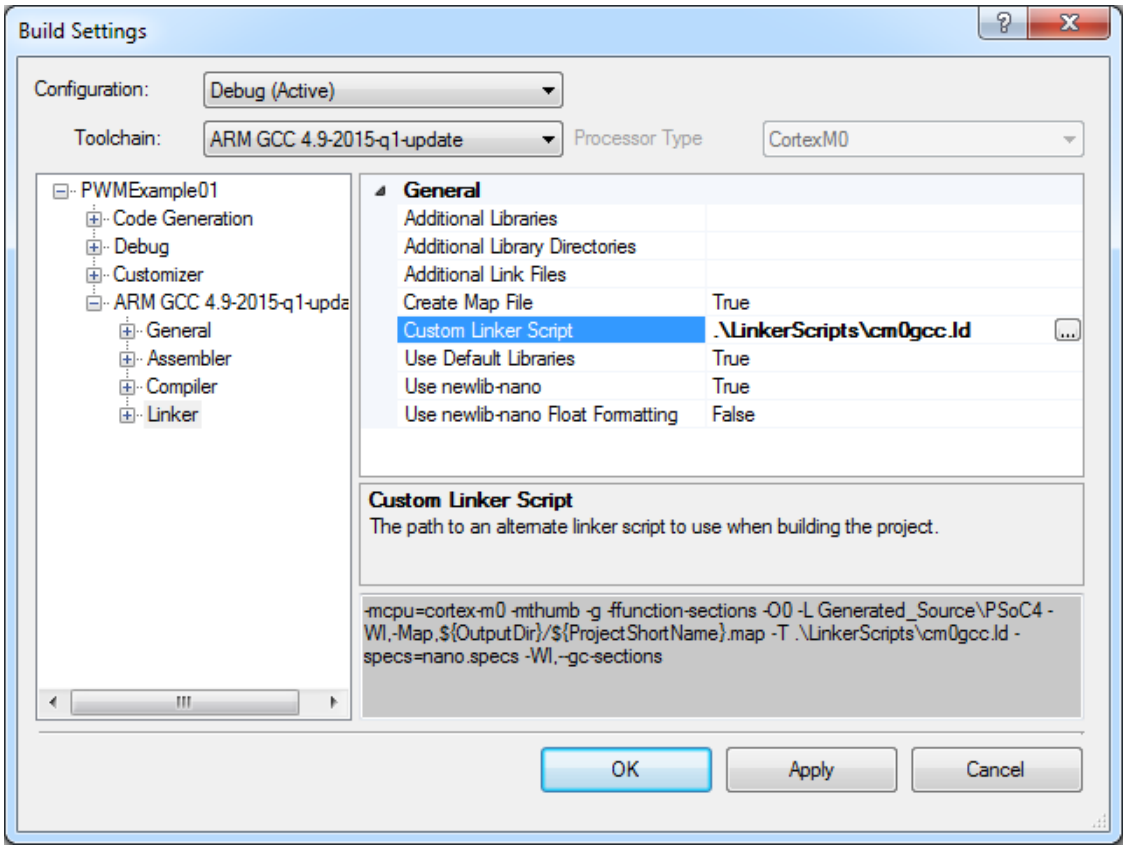
```
/* For GCC compiler use separate API to initialize BLE Stack SRAM.
 * This is needed for code sharing.
 */
#if !defined(__ARMCC_VERSION)
    InitializeBootloaderSRAM();
#endif


/* Checks if Self Project Image is updated and Runs for the First time */
AfterImageUpdate();

/* Start CYBLE component and register generic event handler */
CyBle_Start(AppCallBack);

while (1)
{
    /* If key press event was detected - debounce it and switch to
    bootloader emulator mode */
    if (SW2_Read() == 0u)
    {
        CyDelay(500u);
        if (SW2_Read() == 0u)
        {
            CyDelay(500u);
            while (SW2_Read() == 0u)
            {
                /* Wait for button to be released */
            }

            //Switch to the Stack project, which enables OTA service
            Bootloadable_SetActiveApplication(0);
            Bootloadable_Load();
            CySoftwareReset();
        }
    }

}
```

19. *PWMExample01.cydsn* プロジェクトに新しいフォルダを作成し、「LinkerScripts」と名前を付けます。
20. ステップ 11 で作成したブートローダブル サンプル プロジェクトには LinkerScripts フォルダがあります。このフォルダには、PSoC Creator がサポートするすべての 3 つのコンパイラ用のリンカー スクリプト (*cm0gcc.ld* および *Cm0Mdk.scat*) が含まれています。ステップ 0 で PWMExample01 プロジェクトで作成した LinkerScripts フォルダに 3 つのファイルをすべてコピーします。
21. PWMExample01 プロジェクト用に PSoC Creator の **Workspace Explorer** で新しいフォルダを作成し、「LinkerScripts」と名前を付けます。
22. 次のファイルを PSoC Creator PWMExample01 プロジェクトの **Workspace Explorer** 内の LinkerScripts フォルダ (ステップ 21 で作成した) に追加します。ファイル システムでは、これらのファイルはステップ 0 で作成した LinkerScripts フォルダにあります。
    - *cm0gcc.ld*
    - *Cm0Mdk.scat*
23. 表 7 に示す設定が PWMExample01 のビルド設定に適用されていることを確認します。これらの変更は、新しいカスタム リンカー スクリプトを使用するようにリンカーに指示します。ビルド設定を変更するには、**Project** > **Build Settings...**を選択し、図 23 に示すように、ツリー ビュー ウィンドウで **PWMExample01** > **ARM GCC 4.9-2015-q1-update** > **Linker** > **General** を選択します。

表 7. ビルド設定の変更

フィールド	値
Custom Linker Script	.\LinkerScripts\cm0gcc.ld

図 23. Build Settings ダイアログのリンカー設定



24. Bootloader／Bootloadable コンポーネントを追加した後、デバッグの対応は PSoC Creator で無効になります。しかし、ファームウェアが実行を開始すると、ファームウェアをデバッグするために、ターゲット デバイスに接続し (**Debug** > **Attach to Running Target…**)、または UART メッセージングを使用します。UART デバッグを有効／無効にする方法は「デバッグ」を参照してください。UART が有効のとき、プロジェクトが正常に実行されるように十分なヒープ (0x400 バイト) とスタック (0×800 バイト) が設定されたことを確認してください。そうしないと、ファームウェアは予測できない動作となる場合があります。詳細は「デバッグ」を参照してください。
25. PWMExample01 プロジェクトをビルドします。プロジェクトはエラーなしでビルドされます。
26. メニュー項目の **Debug** > **Program** を選択して PSoC BLE Pioneer Kit をプログラムします。

プログラミングが完了した後、緑色 LED は高輝度から低輝度まで周期的に繰り返します。ここで、「OTA アップグレードの実行」で述べた方法の 1 つで、デバイス ファームウェアのアップグレードを実行できます。また、「OTA 機能のテスト」で述べた手順に従って、OTA 機能のテストもできます。

BLE_OTA_UpgradableStackExample_Stack01.cyacd (スタック イメージ) および *PWMExample01.cyacd* (アプリケーション イメージ) の 2 個のブートローダブル イメージが使用可能です。アプリケーションをアップグレードするには、アプリケーション イメージ ファイルを使用してアップグレードを実行します。スタックをアップグレードするには、スタック イメージ ファイルを使用してアップグレードを実行します。ターゲット デバイスとの接続を再確立してから、アプリケーションのアップグレードを実行します。

6 OTA アップグレードの実行

サイプレスは 3 種類のホスト アプリケーションを提供し、これらのアプリケーションを介してターゲット デバイスのファームウェアをアップグレードできます。ターゲット デバイスで実装した OTA 方法にかかわらず、すべての 3 つのホストは OTA アップグレードを実行するために互換的に使用できます。

OTA アップグレードを実行するために、ターゲット プラットフォームは「外部メモリ OTA ブートローダの追加」、「固定スタック OTA ブートローダの追加」、または「アップグレード可能スタック OTA ブートローダの追加」で説明されたプロセスの終わりに生成される HEX ファイルで事前にプログラムする必要があります。ビルド プロセスは (ブートローダブル プロジェクト出力ディレクトリに、HEX ファイルと共に) ブートローダブル／アプリケーションン イメージである.*cyacd* ファイルも生成します。

特定の種類の OTA ブートローダを使用するプロジェクトから得られた.*cyacd* ファイルは、他の OTA ブートローダがプログラムされたデバイスをアップグレードする目的には使用できません。例えば、外部メモリ OTA ブートローダを使用するプロジェクトからの.*cyacd* ファイルは、固定スタック OTA ブートローダがプログラムされたデバイスでは機能しません。

PC ベースのファームウェア アップグレード方法が正常に機能するために、CySmart USB ドングルが必要です (図 24 を参照してください)。これは PSoC 4 BLE Pioneer Kit に同梱されています。

図 24. CySmart USB ドングル



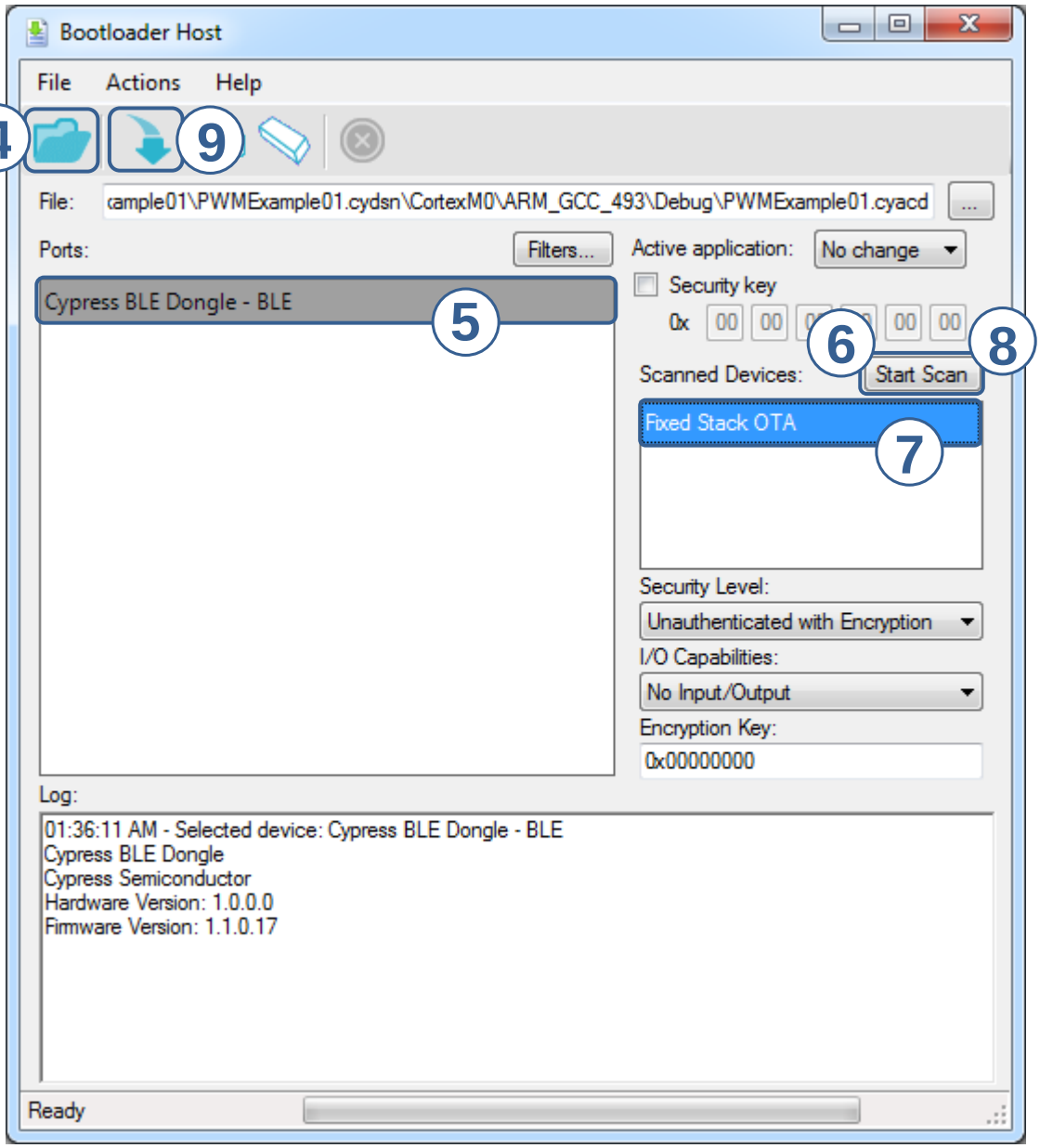
6.1 Bootloader Host ツールによるアップグレード

Bootloader Host ツール (図 25 を参照してください) は PSoC Creator で提供され、BLE OTA アップグレードを含むさまざまなデバイスのファームウェア アップグレード動作のために使用できます。Bootloader Host ツールを使用して OTA ベースのデバイス ファームウェア アップグレードを実行するために、以下の手順に従ってください。進む前に CySmart USB ドングルを PC に差し込んだことを確認してください。

注: CySmart 1.2 ドングル ファームウェアのリリースによって、Bootloader Host ツールはもはや OTA をサポートしていません。

1. CySmart USB ドングル (図 24 に示されるもの) を PC の USB ポートに接続します。

図 25. Bootloader Host ツール



2. PSoC 4 BLE Pioneer Kit 上の **SW2** スイッチを押して、赤色 LED で示されるブートローダ モードにデバイスを移行させます。特定要件に合わせるために、ブートローダ モードに移行する方法を再実装できます (例えば、特定の BLE 特性書き込みコマンドでブートロード プロセスをトリガーできます)。

   アップグレード可能スタック OTA を実装するとき、OTA プロセスがまだ開始していない場合、40 秒後にブートロード モードがタイムアウトします。外部メモリ OTA および固定スタック OTA の実装の場合、タイムアウトはなく、ファームウェアはブートロード モードで無制限に待ちます。

3. PSoC Creator で **Tools** > **Bootloader Host** を選択して Bootloader Host ツールを開きます。

4. **Open** (図 25 を参照してください) をクリックし、**.cyacd* ファイルへのパスを指します。このファイルはプロジェクト フォルダ *([project folder]\CortexM0\[compiler name])* に格納されています。

5. Bootloader Host ツールで、**Ports** 下にリストされる **Cypress BLE Dongle** を選択します。図 25 を参照してください。

6. 図 25 に示すように、**Scanned Devices** フィールドの隣にある **Start Scan** ボタンをクリックします。

7. 期待するデバイス (外部メモリ OTA／固定スタック OTA) が **Scanned Devices** に表示されるまで待ち、それを選択します (図 25 を参照してください)。

8. **Stop Scan** ボタンをクリックします (図 25 を参照してください)。

9. Bootloader Host ツールで **Program** をクリックし (図 25 を参照してください)、新しいアプリケーション イメージのアップロードが完了するまで待ちます。

ファームウェア アップグレードが完了した後、デバイスは自動的にリセットし、緑色 LED が点滅します。詳細は PWMExample プロジェクトのデータシートを参照してください。

6.2 CySmart PC ツールによるアップグレード

CySmart は Windows PC 用の BLE ホスト エミュレーション ツールです。使いやすい GUI を使用することで、ユーザーは BLE ペリフェラル アプリケーションのテストおよびデバッグを実行できます。CySmart PC ツールをダウンロードできます。CySmart の詳細は同じ場所にあるユーザー ガイドを参照してください。

CySmart PC ツールを使って OTA ベースのデバイス ファームウェア アップグレードを実行するために、以下の手順に従ってください。進む前に CySmart USB ドングルを PC に差し込んだことを確認してください。

1. CySmart USB ドングル (図 24 に示されるもの) を PC の USB ポートに接続します。

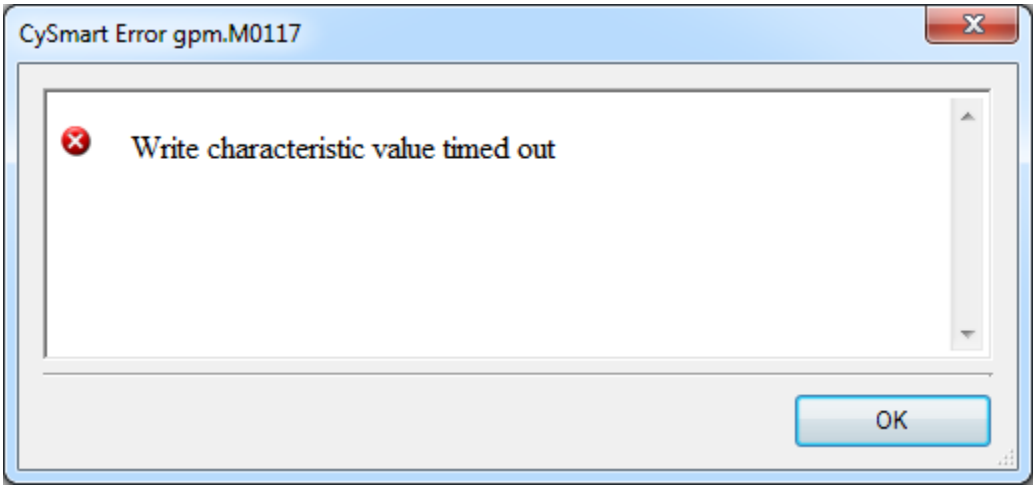
2. PSoC 4 BLE Pioneer Kit 上の **SW2** スイッチを押して、赤色 LED で示されるブートローダ モードにデバイスを移行させます。特定要件に合わせるために、ブートローダ モードに移行する方法を再実装できます (例えば、特定の BLE 特性書き込みコマンドでブートロード プロセスをトリガーできます)。

3. CySmart ツールを開いて、「CySmart User Guide」の「2.7. Updating Peripheral Device Firmware」で述べた手順に従ってイメージをダウンロードします。

ファームウェア アップグレードが完了した後、デバイスは自動的にリセットし、緑色 LED が点滅します。詳細は PWMExample プロジェクトのデータシートを参照してください。

CySmart PC ツールで OTA アップグレードを実行するとき、ブートロード プロセスの終了時に **Write characteristic value timed out** エラーが発生することがあります (図 26 を参照してください)。ここで、ブートロード プロセス自体は失敗でなく、ユーザーがこのエラーを無視できることに注意してください。

図 26. ブートロード プロセス終了時の「Write Characteristic Value Timed Out」エラーのダイアログ



6.3 CySmart モバイル アプリによるアップグレード

CySmart モバイル アプリはサイプレス セミコンダクタ社が開発した BLE または Bluetooth Smart ユーティリティです。CySmart はさまざまな BLE 製品との接続に使用し、また CY8CKIT-042-BLE PSoC 4 BLE Pioneer Kit、CY5672 PRoC BLE リモコン リファレンス デザイン キットおよび CY5682 PRoC BLE タッチマウス リファレンス デザイン キットを含むサイプレスの BLE 開発キットと共に使用できます。CySmart モバイル アプリは iOS®および Android™に対応しています。それらは www.cypress.com/cysmartmobile からダウンロードできます。詳細は同じ場所にあるユーザー ガイドを参照してください。

6.3.1 iOS からのアップグレード

CySmart iOS アプリケーションを使って OTA ベースのデバイス ファームウェア アップグレードを実行するために、以下の手順に従ってください。進む前に、CySmart iOS アプリがユーザーのモバイル／タブレット デバイスにインストールされたことを確認してください。また、新アプリケーション イメージ (*PWMExample01.cyacd*) がモバイル デバイスにあることを確認してください。

1. PSoC 4 BLE Pioneer Kit 上の **SW2** スイッチを押して、赤色 LED で示されるブートローダ モードにデバイスを移行させます。
2. iOS デバイスの CySmart アプリを起動し、「CySmart iOS App User Guide」の「2.1.2.3. Cypress Bootloader Service」で述べた手順に従ってイメージをダウンロードします。

6.3.2 Android からのアップグレード

CySmart Android アプリを使って OTA ベースのデバイス ファームウェア アップグレードを実行するために、以下の手順に従ってください。進む前に、CySmart Android アプリがユーザーのモバイル／タブレット デバイスにインストールされたことを確認してください。また、新アプリケーション イメージ (*PWMExample01.cyacd*) がモバイル デバイスにあることを確認してください。

1. PSoC 4 BLE Pioneer Kit 上の **SW2** スイッチを押して、赤色 LED で示されるブートローダ モードにデバイスを移行させます。
2. Android デバイスの CySmart アプリケーションを起動し、「CySmart Android App User Guide」の「2.1.2.3. Cypress Bootloader Service」で述べた手順に従ってイメージをダウンロードします。

ファームウェア アップグレードが完了した後、デバイスは自動的にリセットし、緑色 LED が点滅します。詳細は PWMExample プロジェクトのデータシートを参照してください。

7 OTA 機能のテスト

OTA 機能をテストするために、以下の手順に従ってください。

1. PWM サンプル プロジェクトの *main.c* では、`BRIGHTNESS_DECREASE` マクロの値を 1000 に変更します。詳細は PWMExample プロジェクトのデータシートを参照してください。
2. PWMExample プロジェクトを再度ビルドします (**Build** > **Build PWMExample01**)。すると、新しい *.cyacd* ファイルが生成されます。
3. 「OTA アップグレードの実行」で記述される OTA アップグレード方法のいずれかに従ってください。ステップ 2 で出力された *PWMExample.cyacd* ファイルの選択を確認してください。

ファームウェア アップグレードが完了した後、デバイスは自動的にリセットし、緑色 LED 輝度調光はより速い速度で繰り返します。

`BRIGHTNESS_DECREASE` マクロを 0～63000 範囲の値に変更し、上記の手順に従って、異なる結果を観察します。値が小さいほど輝度調光サイクル速度が遅くなり、値が大きいほど輝度調光サイクル速度が速くなります。

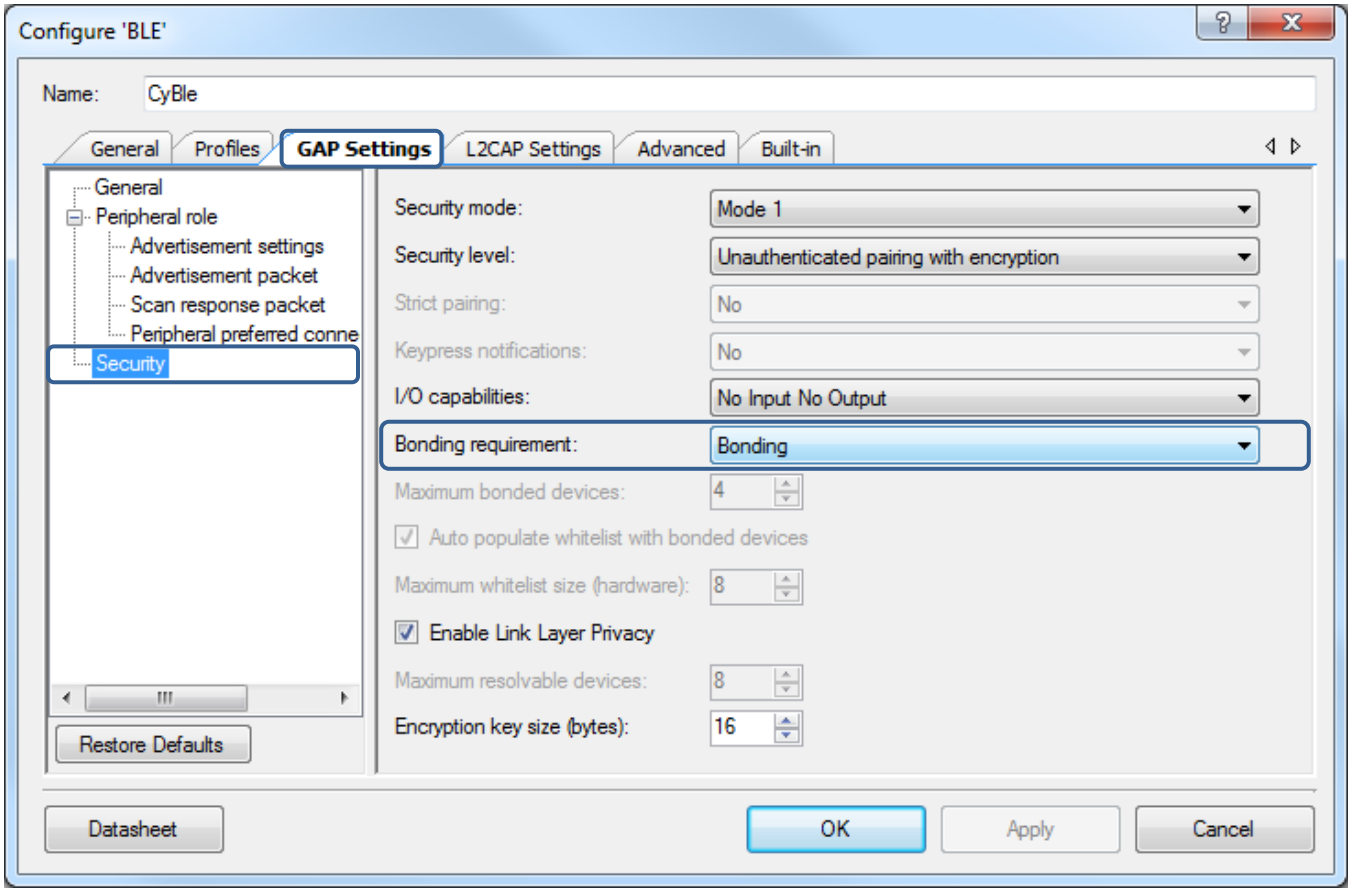
8 その他の注意事項

8.1 ボンディング／ペアリングの情報

BLE コンポーネントでボンディングを有効にする方法を提供します。また、ボンディング／ペアリング情報に対する 3 種類の OTA アップグレードの影響を説明します。

ボンディングを有効にするには、**Configure BLE** ダイアログの **GAP Settings** の下の **Security** で **Bonding requirement** を **Bonding** オプションに選択します (図 27 を参照してください)。

図 27. ボンディング情報保持のための BLE コンポーネントの設定



フラッシュ書き込み中には、割り込みの許可／処理ができないため、フラッシュ書き込みは都合の良いときにアプリケーションによって処理する必要があります。フラッシュ書き込みイベントが保留中になるたびに cyBle_pendingFlashWrite 変数はスタックによって設定されます。アプリケーションは、このフラグの状態を CyBle_StoreBondingData API と共にチェックすることにより、ボンディング情報を書き込めます。コード 2 を参照してください。

コード 2. アプリケーション レベルのボンディング情報の書き込み

```
if((cyBle_pendingFlashWrite != 0u))
{
    #if (DEBUG_UART_ENABLED == YES)
        CYBLE_API_RESULT_T apiResult;
        apiResult = CyBle_StoreBondingData(0u);
        DBG_PRINTF("Store bonding data, status: %x \r\n", apiResult);
    #else
         (void)CyBle_StoreBondingData(0u);
   #endif /* (DEBUG_UART_ENABLED == YES) */
}
```

次の節では、各 OTA ブートローダ種類に特有のボンディング／ペアリング情報を提供します。

8.1.1 外部メモリ OTA ブートローダ

外部メモリ OTA ブートローダでは、ボンディング情報はアプリケーション (ブートローダブル) プロジェクトで処理されます。そのため、アップグレード後にボンディング情報全体が失われます。

8.1.2 固定スタック OTA ブートローダ

固定スタック OTA ブートローダの場合、ボンディング情報はブートローダ プロジェクトの範囲に割り当てられます。したがって、ブートローダブル プロジェクトがアップグレードされた後でも、ボンディング情報は変わらないままですが、デバイスがシリアル ワイヤ デバッグ (SWD) プログラマで再プログラムされる場合にのみ消去されます。

8.1.3 アップグレード可能スタック OTA ブートローダ

アップグレード可能スタック OTA ブートローダの場合、ボンディング情報はスタック (任意) およびアプリケーション イメージの両方に格納されます。デフォルトではボンディング情報はスタックによって格納されません。クライアント特性コンフィギュレーション ディスクリプタ (CCCD) が変更されない限り、アプリケーションのアップグレードはアプリケーションのボンディング情報に影響を与えません。スタックをアップグレードすると、アプリケーションのボンディング情報は失われます。

8.2 デバッグ

プロジェクトに 1 個の OTA ブートローダを追加した後、PSoC Creator からプロジェクトをデバッグするオプションは無効になります。しかし、ファームウェアの実行が開始されると、デバッグのためにターゲット デバイスに接続できます (**Debug** > **Attach to Running Target...**)。この方法を使用する場合、1 回にただ 1 つのプロジェクトだけはデバッグできます。間違ったプロジェクトから行う場合、実行中のターゲット デバイスへの接続はデバイスを再プログラムする可能性があります。また、デバッガをターゲット デバイスに接続するために、SWD インターフェースは有効にする必要があります。プロジェクトの SWD インターフェースは、使用されるプロジェクトの *.cydwr* ファイル (例: *PWMExample01.cydwr*) で **Debug Select** を **SWD** に変更することにより有効にされます (図 28 を参照してください)。*.cydwr* ファイルはプロジェクトの **Workspace Explorer** からアクセスできます。

SWD インターフェースはデバッグのために利用できない場合、GPIO トグル、任意のシリアル通信インターフェースまたは UART などのプロトコルを使用してください。それを実現するために、PSoC Creator の OTA サンプル プロジェクトはソフトウェア UART (TX のみ) コンポーネントを利用します。ユーザーがブートローダブル プロジェクトを作成するために PSoC Creator サンプル プロジェクトから *debug.c* および *debug.h* ファイルを取得したため (「ターゲット プロジェクトへのファームウェア OTA ブートローダの追加」を参照してください)、この機能を使ってこれらのプロジェクトをデバッグできます。

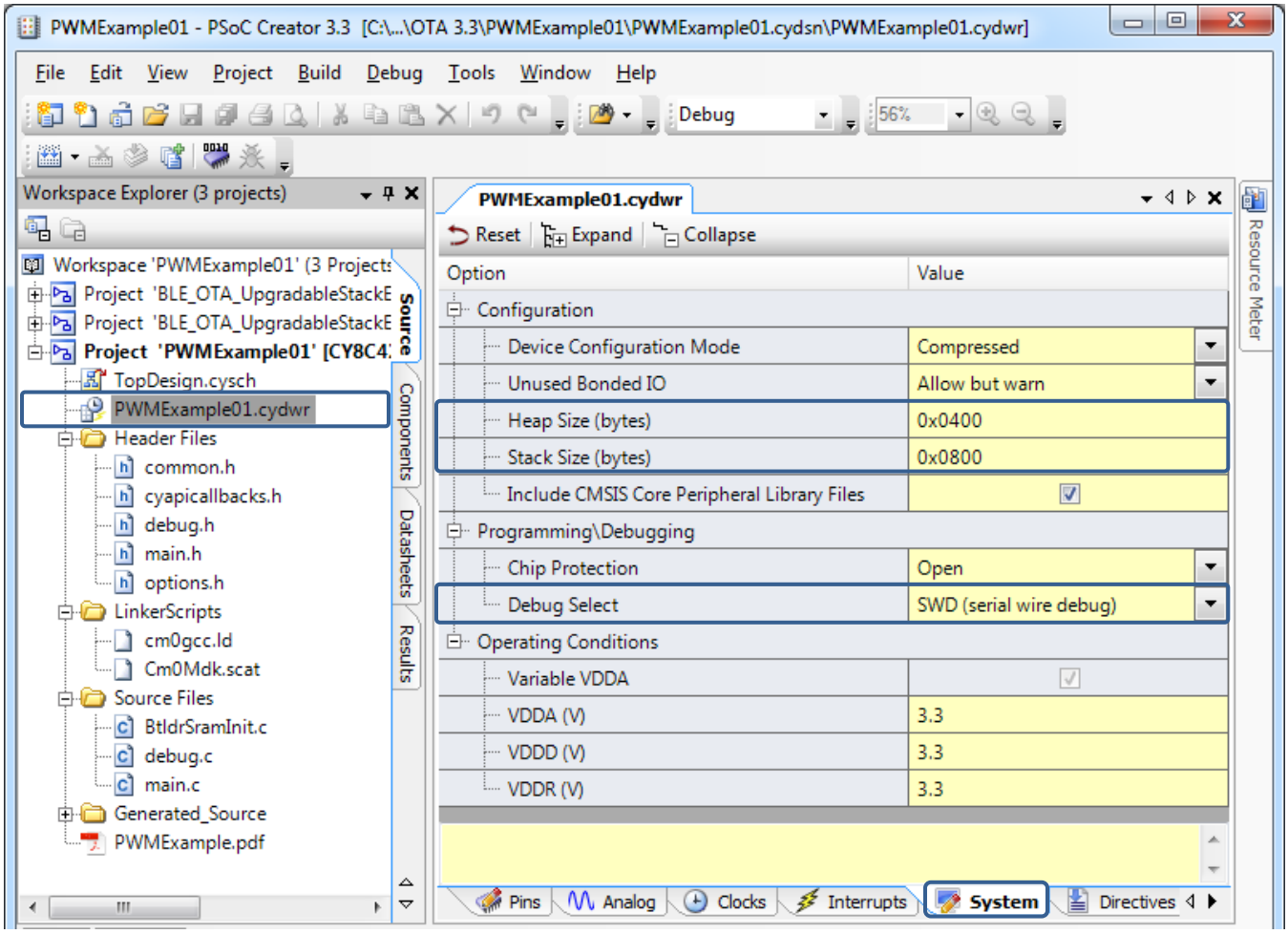
コード 3 に示すように、*Options.h* ヘッダ ファイルにある DEBUG_UART_ENABLED マクロを YES に設定することで、あらゆる OTA プロジェクトで UART デバッグを有効にすることができます。

コード 3. Options.h の DEBUG_UART_ENABLED マクロ

```
#define DEBUG_UART_ENABLED (YES)
```

OTA の例では、UART を介してメッセージを送信するために `printf` 文 (直接またはマクロの定義) を使用します。`printf` は、正常の実行のためにかなりのヒープとスタック空間を必要とします。したがって、`DEBUG_UART_ENABLED` マクロが有効のとき、アプリケーションが正常に機能するために十分なヒープとスタック メモリを割り当てる必要があります。それを実現するために、**Workspace Explorer** からユーザーのプロジェクトの *.cydwr* ファイル (例えば、*PWMExample01.cydwr*) をダブルクリックし、それを開いて、**System** タブに移動します (図 28 を参照してください)。適切な **Heap Size** および **Stack Size** を設定します (**Heap Size** を **0x400** バイトに、**Stack Size** を **0x800** バイトに設定することはサンプル プロジェクトでの良いスタート ポイントです)。

図 28. PSoC Creator の Heap Size および Stack Size の変更



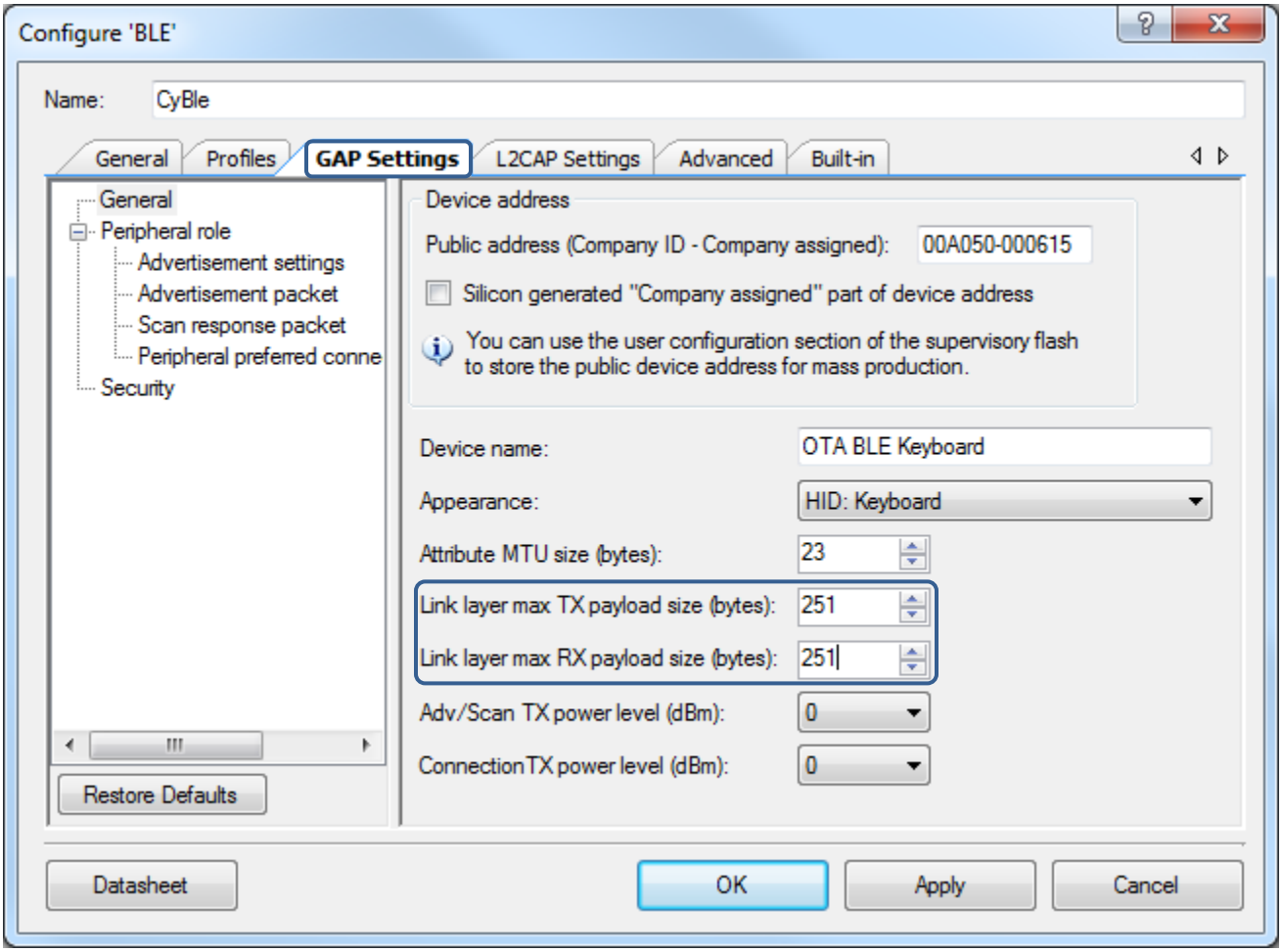
8.3 データ長拡張 (DLE)

Bluetooth SIG は Bluetooth コア仕様 4.2 で LE データ パケット長拡張またはデータ長拡張 (DLE) を提供します。この機能により、リンク層の最大データ チャネル ペイロード長は 27 バイト (Bluetooth コア仕様 4.1 またはそれ以前) から 251 バイトに増えます。したがって、リンク層データ パケットの容量が約 10 倍、リンクのスループットが約 2.6 倍増加します。サイプレスのデバイスがサポートしている DLE および Bluetooth コア仕様 4.2 の他の機能の詳細は AN99209 を参照してください。

DLE 機能は、BLE コンポーネント Ver. 3.0 以降で利用可能であり、Bluetooth コア仕様 4.2 に準拠するすべてのサイプレス デバイスによってサポートされます。以下の手順を行うことで、DLE はいかなる BLE プロジェクトでも (OTA プロジェクトを含む) 有効にできます。

1. BLE コンポーネントをダブルクリックして BLE コンポーネント コンフィギュレーション ダイアログを開きます。
2. **GAP Settings** タブで、**Link layer max TX payload size (bytes)** および **Link layer max RX payload size (bytes)** の値を **27～251** 範囲内の値に変更します (図 29 を参照してください)。

図 29. BLE コンポーネントの GAP Settings タブ



8.4 データの永続性

8.4.1 SFlash の使用

PSoC と PRoC BLE 両方のデバイスはユーザーSFlash 領域があります。SFlash はフラッシュ保護設定、トリム設定などの情報を格納するために使用されます (ユーザーはこれらの情報にアクセスできません)。また SFlash には、Bluetooth または製品固有の情報 (デバイス アドレス、製造／シリアル番号、センサー較正データなど) を格納するために使用できる、ユーザー設定可能な 4 行があります。これらのユーザー設定可能な行はプログラミング サイクル中または OTA のアップグレード中に消去されないため、データの永続性を提供します。さらに、SFlash は、ファームウェア プログラミングから独立した製造サイクル中には書き込めます。SFlash のユーザー設定可能な行はファームウェアまたは SWD プログラミング インターフェースを介してアクセスできます。

ユーザーSFlash の読み出し／書き込みのサンプル プロジェクトはこちらで入手可能です。この機能をユーザーのプロジェクトで実装する方法は、プロジェクトのユーザー ガイドを参照してください。このサンプル プロジェクトは 128KB BLE デバイスで実装されます。表 8 に、すべての BLE デバイスのパラメーター情報を示します。

表 8. SFlash サンプル プロジェクトのデバイス固有パラメーター

パラメーター	128KB デバイス	256KB デバイス
USER_SFLASH_ROW_SIZE	128	256
USER_SFLASH_ROWS	4	4
USER_SFLASH_BASE_ADDRESS	0x0FFFF200u	0x0FFFF400u

8.4.2 チェックサム除外の使用

ブートローダブル アプリケーションのチェックサム計算からフラッシュの一部を除外できます。このフラッシュ領域は、ユーザーまたはコンポーネントのデータを格納するために使用できます (例えば、この方法で BLE はペアリング データを格納します)。チェックサム除外の詳細はブートローダブル コンポーネント データシートを参照してください。この機能を有効にするには、下記のステップを従ってください。

1. ターゲット データを cy_checksum_exclude 領域に移動させます。以下のサンプル コードを参照してください。

コード 4. GCC コンパイラの cy_checksum_exclude の例

```
const uint8 byteArray[10] CY_SECTION(".cy_checksum_exclude") =
            {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
```

コード 5. MDK コンパイラの cy_checksum_exclude の例

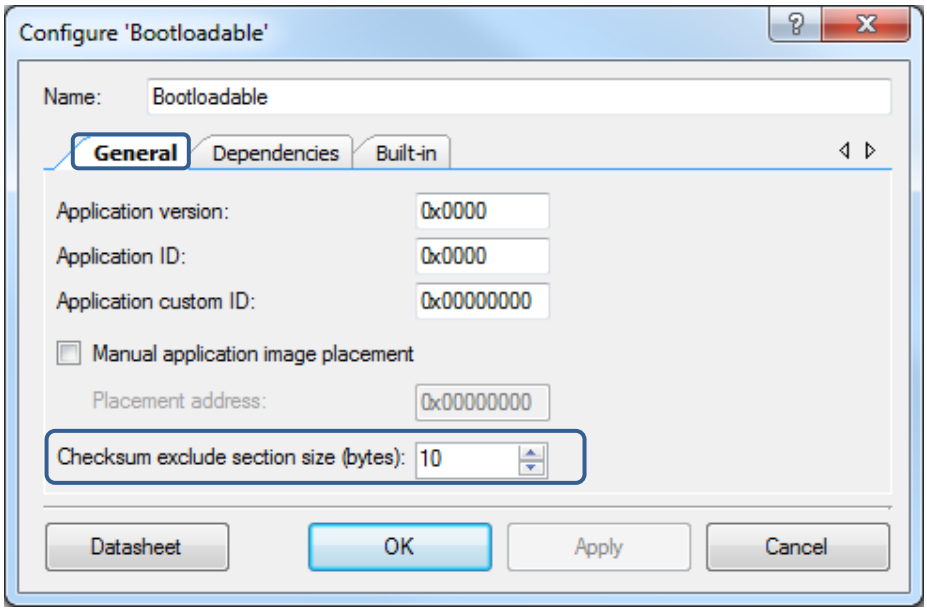
```
const uint8 byteArray[10] CY_SECTION(".cy_checksum_exclude") =
            {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
```

コード 6. IAR コンパイラの cy_checksum_exclude の例

```
#pragma location=".cy_checksum_exclude"
const uint8 byteArray[10] =
            {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
```

2. **General** タブの Bootloadable コンポーネントでは、**Checksum exclude section size (bytes)** を必要な値に設定します (コード 4、5、および 6 では 10 バイト)。図 30 を参照してください。

図 30. Bootloadable コンポーネント ダイアログ



3. カスタム リンカー スクリプト (LinkerScripts フォルダにある *cm0gcc.ld*) が使用されている固定スタック OTA ブートローダに対して、チェックサム除外の情報は手動で追加する必要があります。このために、次のステップを行ってください。

   a. Bootloadable コンポーネントで **Checksum exclude section size (bytes)** が設定されると、プロジェクトをクリーンにして (**Build** > **Clean PWMExample01**)、生成します (**Build** > **Generate Application**)。これは *….\PWMExample01.cydsn\Generated_Source\PSoC4* に新しいリンカー スクリプト ファイルを作成します。また、それらのファイルは PSoC Creator **Workspace Explorer** で **Generated Source** > **PSoC 4** > **cy_boot** からもアクセスできます。

   b. カスタム リンカー スクリプト ファイルを開きます。固定スタック OTA サンプル プロジェクトでは、このファイルは *…\PWMExample01.cydsn\LinkerScripts* にあります。また、PSoC Creator **Workspace Explorer** で **PWMExample01** > **LinkerScripts** からもアクセスできます。

   c. PSoC Creator が生成したリンカー スクリプトで `CY_CHECKSUM_EXCLUDE_SIZE` リンカー スクリプト変数を検索します。カスタム リンカー スクリプトにも同じ変更を加えます。例えばこの場合、以下のとおりに 10 バイトを除外する必要があります。

コード 7. GCC の *cm0gcc.ld* リンカー スクリプトの `CY_CHECKSUM_EXCLUDE_SIZE` 変数

```
CY_CHECKSUM_EXCLUDE_SIZE        = ALIGN(10, CY_FLASH_ROW_SIZE);
```

コード 8. IAR の *Cm0Iar.icf* リンカー スクリプトの `CY_CHECKSUM_EXCLUDE_SIZE` 変数

```
define symbol CY_CHECKSUM_EXCLUDE_SIZE = 10;
```

コード 9. MDK の *Cm0RealView.scat* リンカー スクリプトの `CY_CHECKSUM_EXCLUDE_SIZE` 変数

```
#define CY_CHECKSUM_EXCLUDE_SIZE        AlignExpr(10, CY_FLASH_ROW_SIZE)
```

   d. この変更を行ってから、リンカー スクリプト ファイルを保存し、プロジェクトをビルドします (**Build** > **PWMExample01**)。

9 まとめ

本アプリケーション ノートは、OTA の異なる種類および、それらのアプリケーションでの実装方法について説明しました。また、OTA アップグレード機能およびサイプレスが提供するツールを使ってターゲット デバイスのファームウェアを更新する方法についても説明しました。

10 関連アプリケーション ノート

AN86526 – PSoC 4 I^2C Bootloader

AN73854 – PSoC 3, PSoC 4, and PSoC 5LP Introduction to Bootloaders

AN68272 – PSoC 3, PSoC 4, and PSoC 5LP UART Bootloader

AN91267 – Getting Started with PSoC 4 BLE

AN94020 – Getting Started with PRoC BLE

著者について

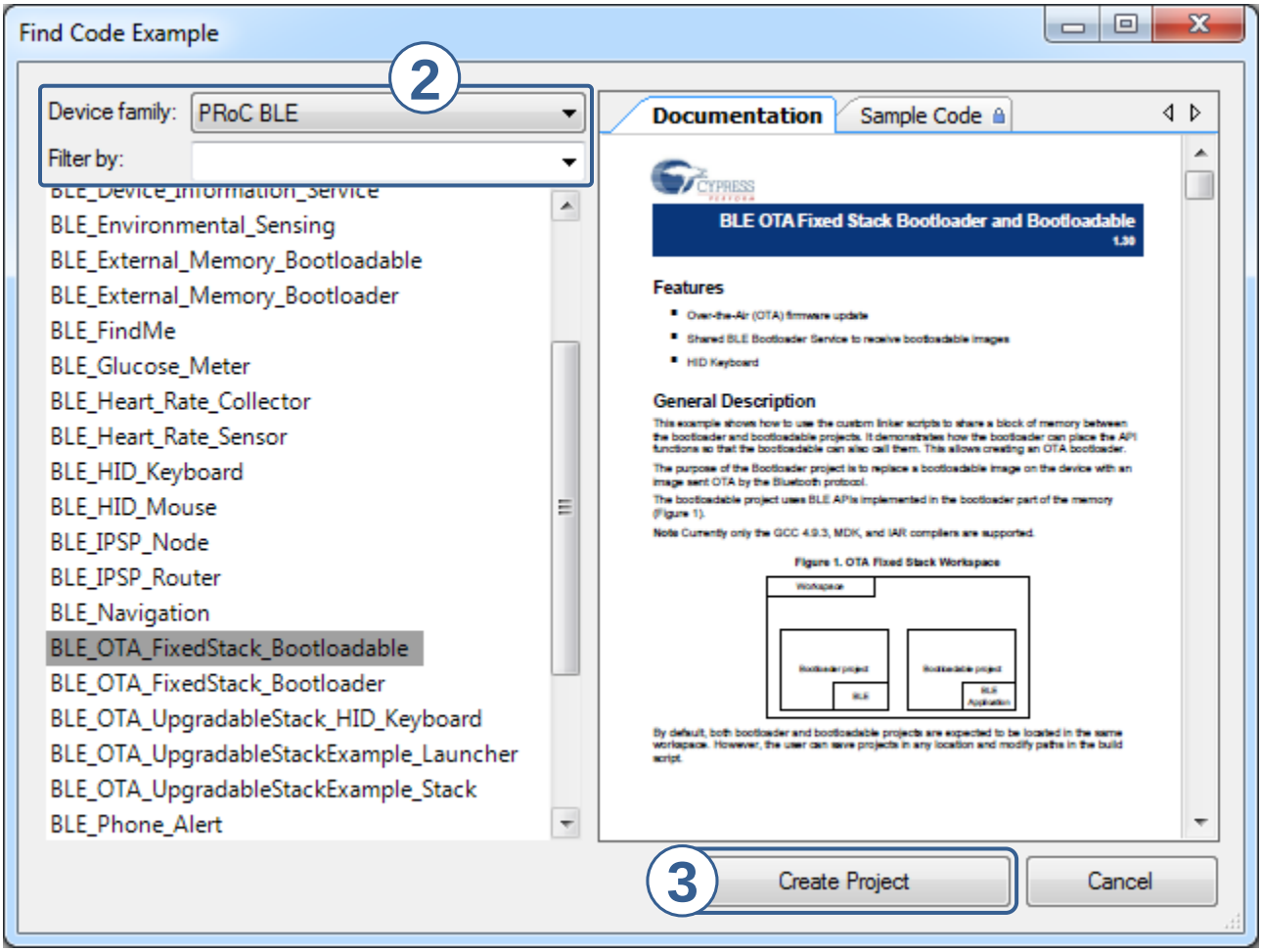
氏名:	Deepak John
役職:	スタッフ アプリケーション エンジニア
経歴:	Deepak John はコーチン科学技術大学の電子通信工学の学士号、およびブリッジポート大学の電気工学理学修士を取得し、組み込みシステムの設計および構築に長く従事しています。

A 付録 A

A.1 サンプル プロジェクト ワークスペースの作成

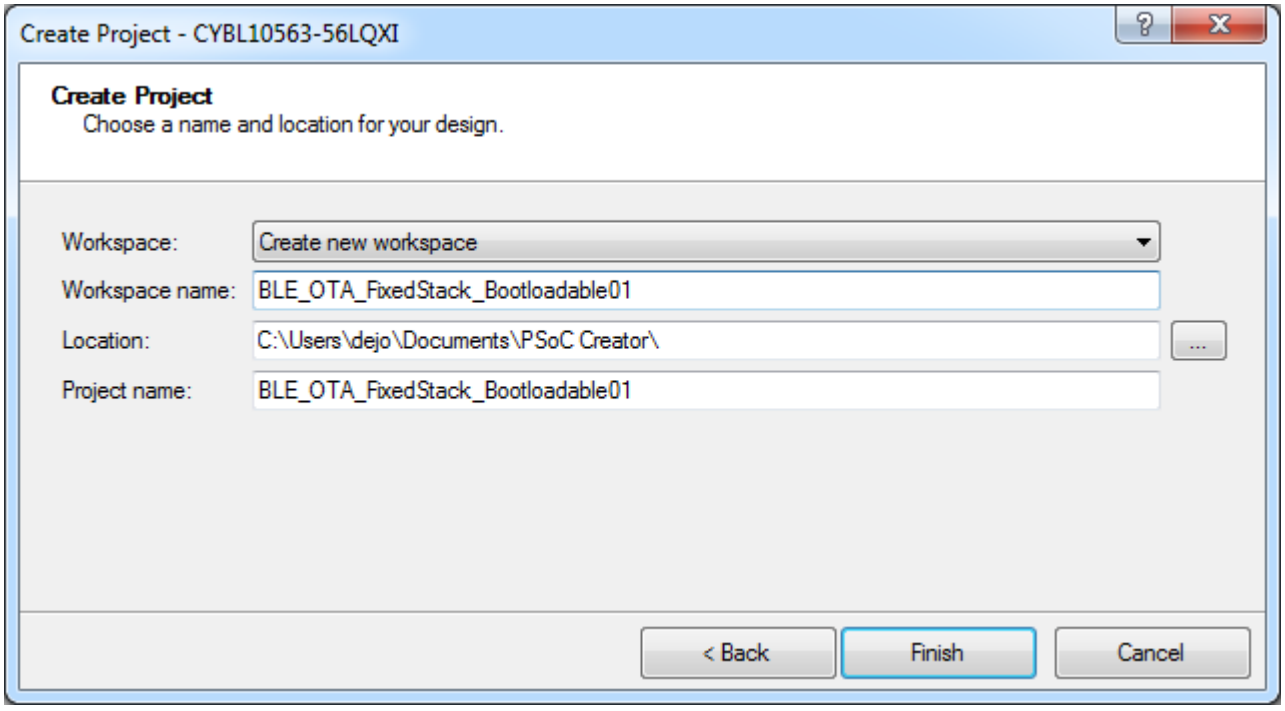
1. 図 31 に示すように、PSoC Creator で **File** > **Code Example...**を選択し、**Find Example Project** ダイアログを開きます。
2. 図 31 に示すように、必要な **Device Family** および **Filter by** キーワードを適用して検索を絞り込みます。
3. 任意のサンプル プロジェクトを選択し、**Create Project** ボタンをクリックします。

図 31. Find Code Example プロジェクト ダイアログ - サンプル プロジェクト ワークスペースの作成



4. **Workspace** フィールドで **Create New Workspace** オプションを選択します。新しいサンプル プロジェクトのワークスペース用に **Location** を選択し (図 32 を参照してください)、**Finish** をクリックします。
5. 必要なパスを選択し、**Finish** をクリックします。

図 32. Create Project ダイアログ - サンプル プロジェクト ワークスペースの作成



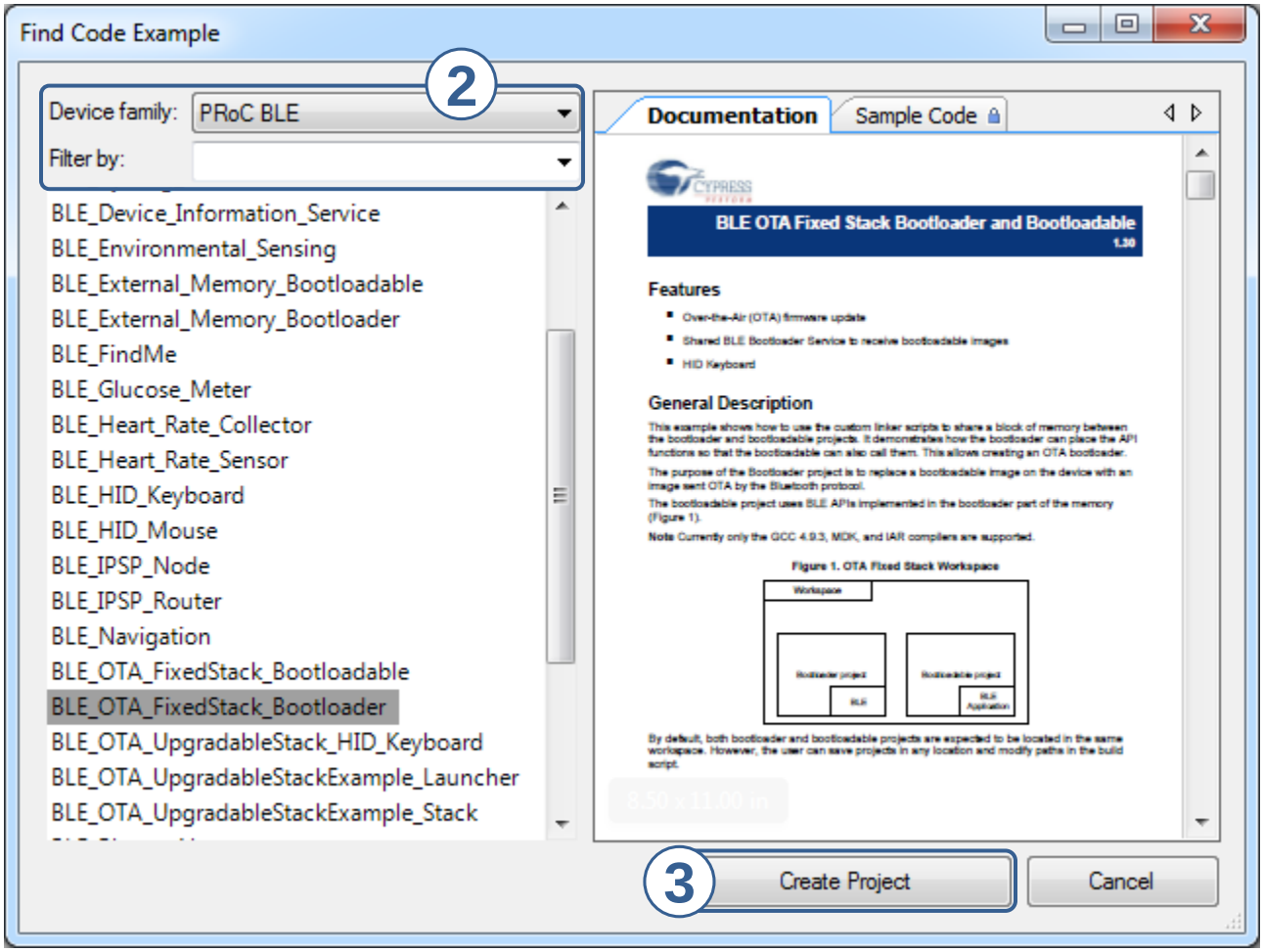
これらのステップが完了したとき、選択した場所にワークスペースが作成されます。選択したサンプル プロジェクトはこのワークスペースに追加されます。データシート *(<project_name>.pdf)* は新しく作成したプロジェクトの下にあります。サンプル プロジェクトの詳細はこの資料を参照してください。

A.2 既存のワークスペースへのサンプル プロジェクトの追加

本節のステップを行う前に、PSoC Creator のインスタンスで Project/Workspace が開いていることを確認してください。既存の Project/Workspace を開くには、**File** > **Open** > **Project/Workspace** を選択します。

1. 図 33 に示すように、**File** > **Code Example…**を選択し、**Find Example Project** ダイアログを開きます。

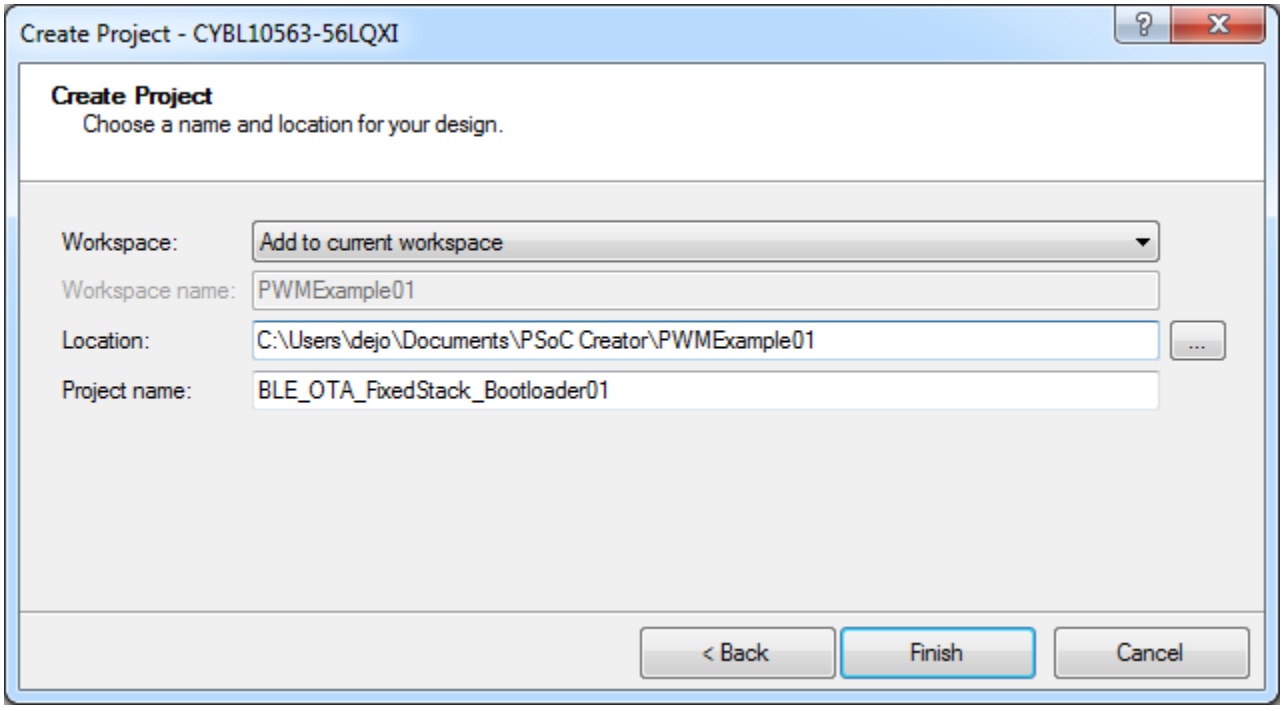
図 33. Find Example Project ダイアログ – 既存のワークスペースへのサンプル プロジェクトの追加



2. 図 33 に示すように、必要な **Device family** および **Filter by** キーワードを適用して検索を絞り込みます。
3. 任意のサンプル プロジェクトを選択し、**Create Project** ボタンをクリックします。
4. **Workspace** フィールドで **Add to current workspace** オプションを選択します。新しいサンプル プロジェクトのワークスペース用に **Location** を選択して(図 34 を参照してください)、**Finish** をクリックします。
5. 必要なパスを選択し、**Finish** をクリックします。

これらのステップが完了すると、選択したサンプル プロジェクトは PSoC Creator のインスタンスで既に開いているワークスペースに追加されます。データシート *(<project_name>.pdf)* は新しく追加したプロジェクトの下にあります。サンプル プロジェクトの詳細はこの資料を参照してください。

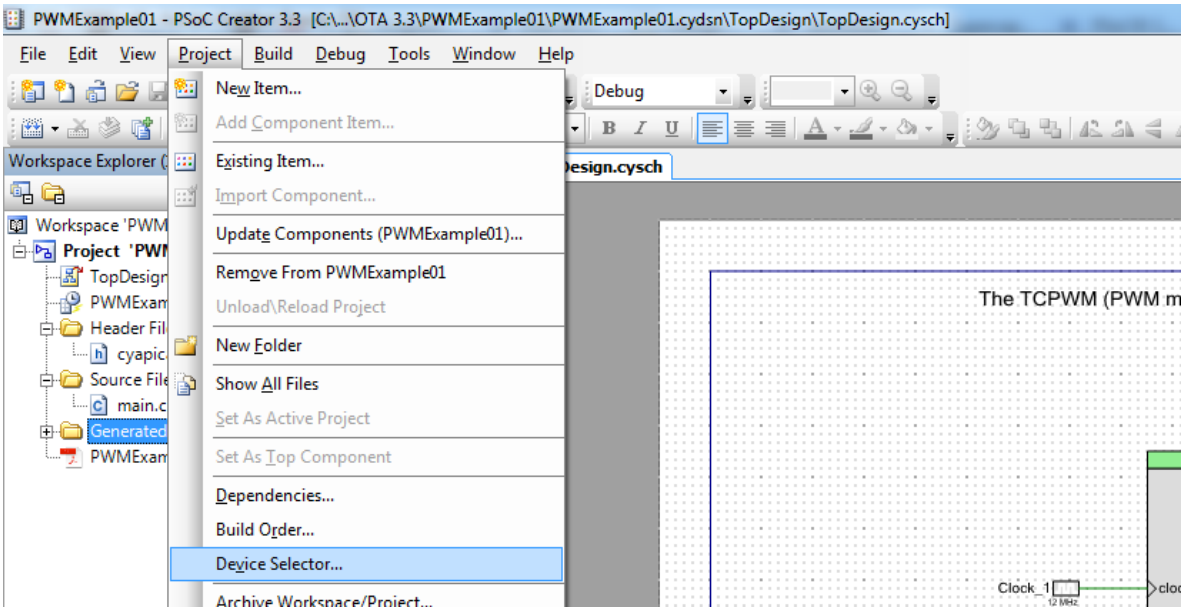
図 34. Create Project ダイアログ – 既存のワークスペースへのサンプル プロジェクトの追加



A.3 他のデバイス選択

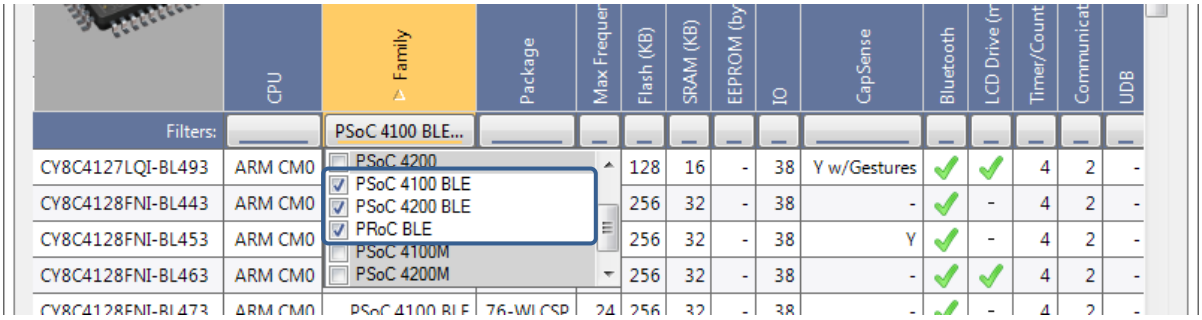
1. PSoC Creator で、**Project** > **Device Selector…**を選択します (図 35 を参照してください)。

図 35. Device Selector の起動



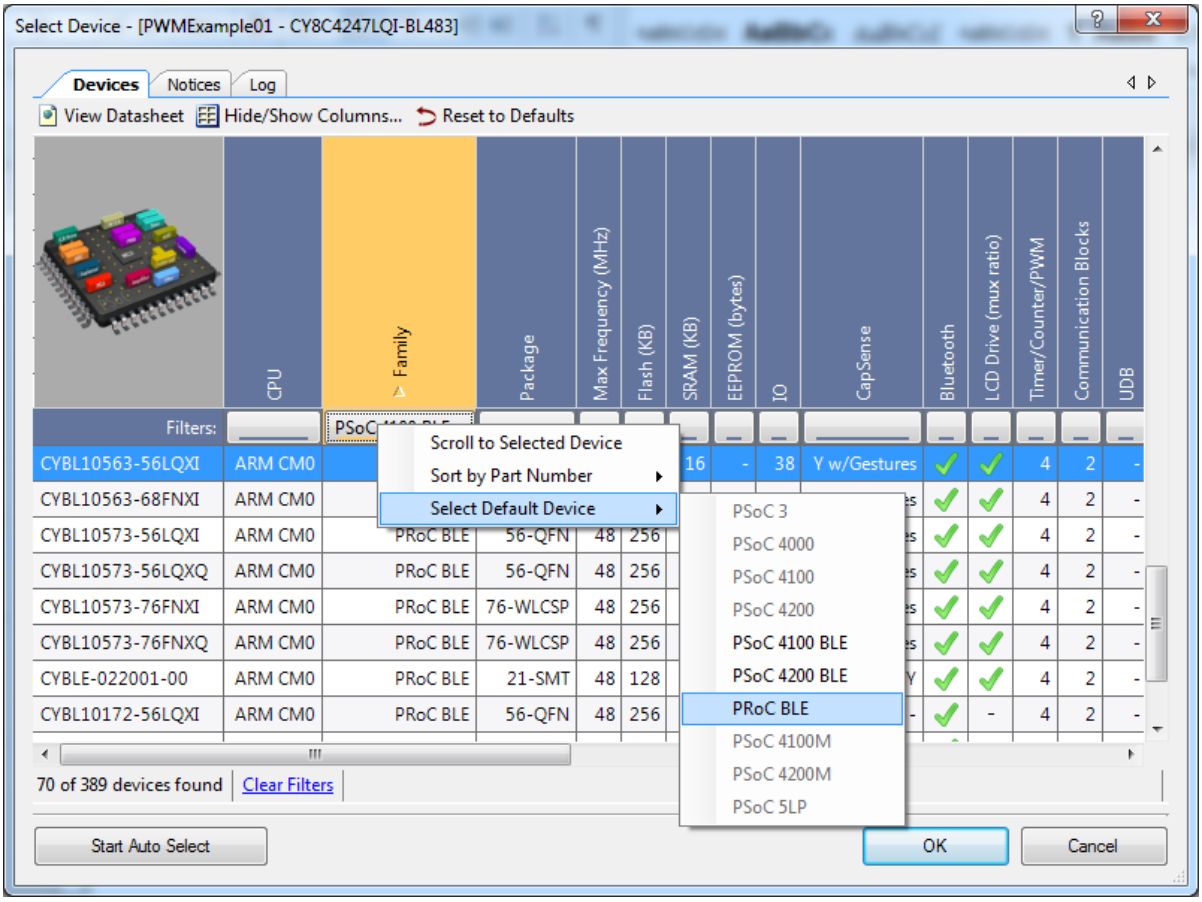
2. 図 36 に示すように、**Family** フィルター カテゴリで **PSoC 4200 BLE**、**PSoC 4100 BLE**、および **PRoC BLE** フィルター項目を有効にします。

図 36. デバイス ファミリの選択



3. リストからユーザーのアプリケーションに適切な BLE デバイスを選択します。ユーザーのターゲット デバイスが CY8CKIT-042-BLE Pioneer Kit に同梱されている PRoC BLE または PSoC 4 BLE モジュールであれば、図 37 に示すように、**Family** フィルターを右クリックして、**Select Default Device** を選択し、適切なデバイス ファミリを選びます。

図 37. CY8CKIT-042-BLE Pioneer Kit のデフォルト PRoC BLE デバイスの選択



4. **OK** をクリックして、Device Selector を閉じます。

A.4 ブートローダ サービスの追加

既存の BLE コンポーネントにブートローダ サービスを追加するには、以下の手順に従ってください。ブートローダ サービスの詳細は BLE コンポーネント データシートを参照してください。

1. BLE コンポーネントをダブルクリックしてコンフィギュレーション ダイアログを開きます (図 38 を参照してください)。
2. **Profiles** タブに移動してサービス／プロファイルを選択します (例えば、図 38 の **HID Device** など)。
3. **Add Service** をクリックし (図 38 を参照してください)、利用可能なサービスを表示して、**Bootloader** を選択します (図 39 を参照してください)。
4. **Bootloader** > **Command** > **Fields** 下の **Data** フィールドを 137 に設定します (図 40 を参照してください)。
5. **Security level** (**GAP Settings** > **Security** タブの下にある) を **Unauthenticated pairing with encryption** に設定します (図 41 を参照してください)。この設定は PSoC Creator 提供のサンプル プロジェクトによって使用され、正常動作のために本アプリケーション ノートの手順に必要です。あるいは、更新の実行中に更新ツール (CySmart) 用にセキュリティ オプション (コンポーネントで選択される) を使用します。このステップは必須ではありません。

図 38. BLE コンポーネント コンフィギュレーション ダイアログ

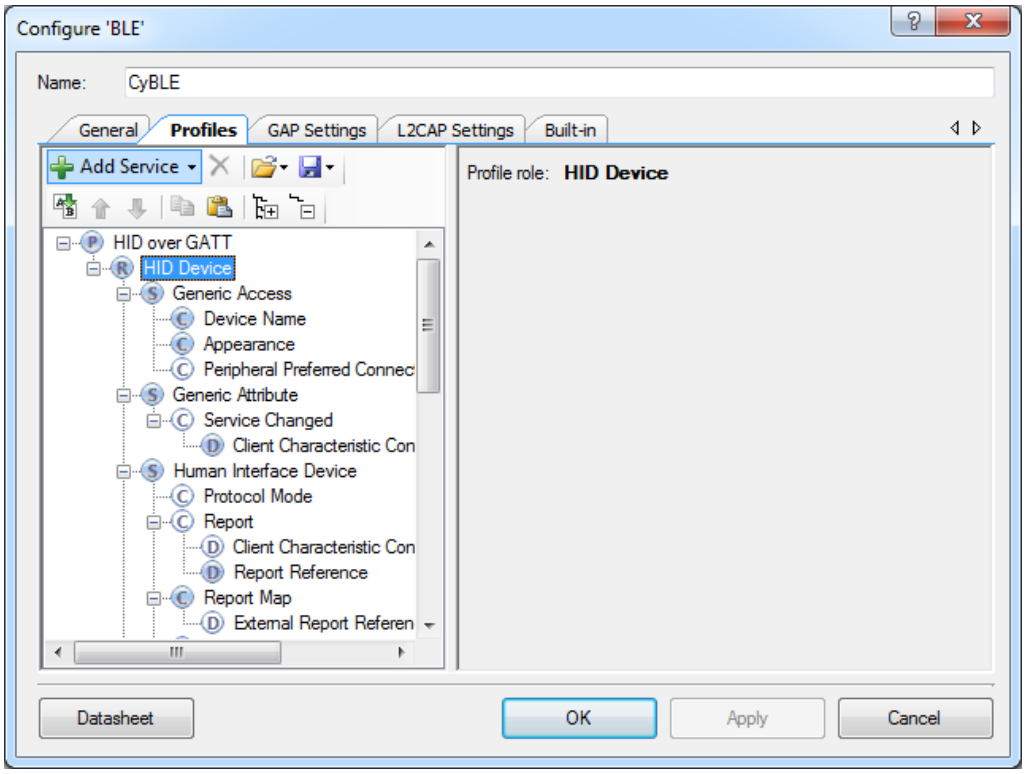


図 39. ブートローダ サービスの選択

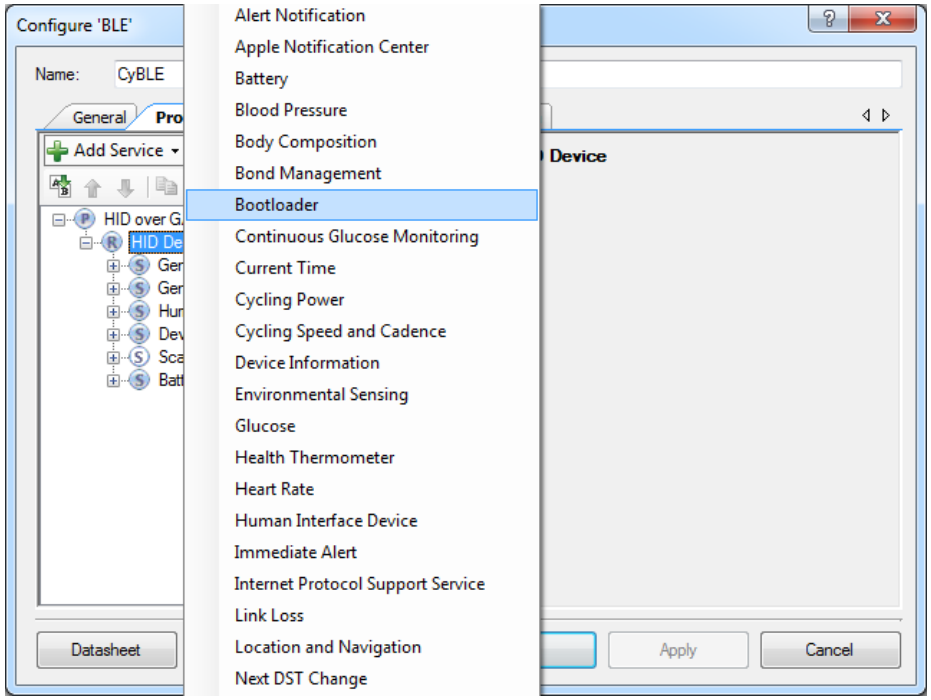


図 40. ブートローダ サービス データ フィールドに 137 の設定

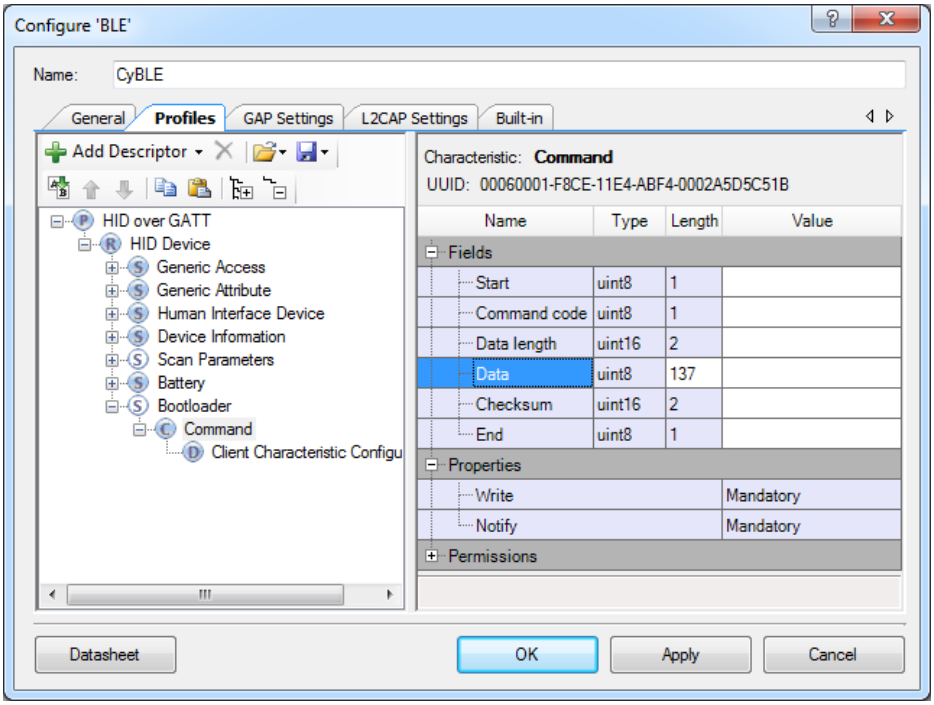
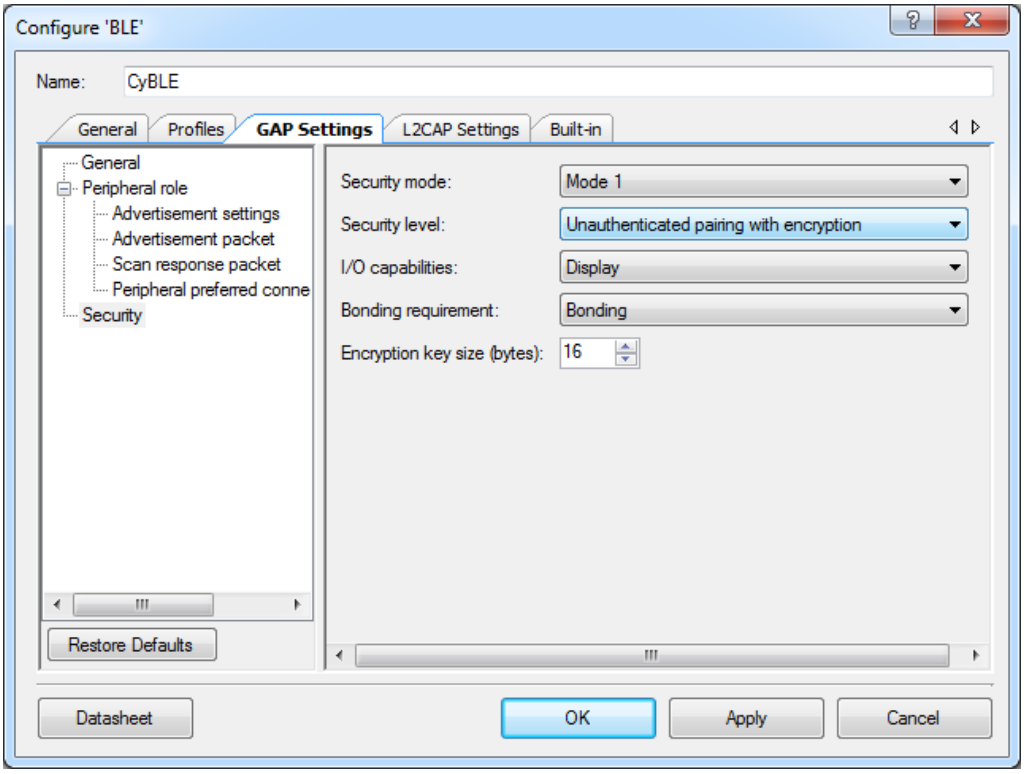


図 41. セキュリティ オプションの変更



A.5 サイプレスの他の BLE デバイス用の固定スタック OTA プロジェクトの設定

BLE OTA 固定スタックのサンプル プロジェクトはカスタムのリンカー スクリプトを使用するため、PSoC Creator は正しいリンクと配置を提供するためにリンカー スクリプトを設定できません。ここでは、CY8C4247LQI-BL483 や CYBL10563-56LQXI デバイスを使用していない、または異なる RAM／ROM メモリ サイズを持つデバイスを使用している場合に従う必要がある手順を説明します。デフォルトで、リンカー スクリプトは 128KB デバイス用の設定があります。

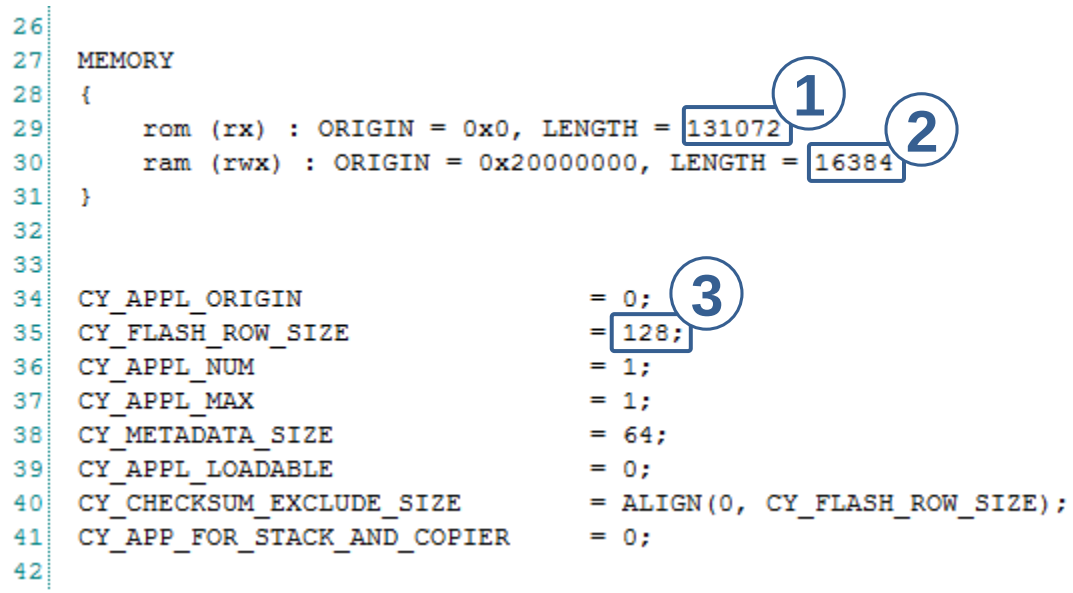
次の節では、PSoC Creator でデバイスを変更した後にリンカー スクリプトを修正することについて説明します。

A.5.1 GCC コンパイラ

GCC コンパイラの場合、以下のようにブートローダおよびブートローダブル リンカー スクリプトの両方に変更が必要です。

1. 256KB デバイスの場合、デバイスのフラッシュ メモリ サイズを 262144 に変更します (図 42 を参照してください)。
2. 256KB デバイスの場合、デバイスの RAM サイズを 32768 に変更します (図 42 を参照してください)。
3. 256KB デバイスの場合、デバイスの行サイズを 256 に変更します (図 42 を参照してください)。

図 42. 128KB デバイス用のコンフィギュレーションを示す cm0gcc.ld ファイルの内容



ブートローダ プロジェクトのリンカー スクリプトは*…\cm0gcc.ld*、ブートローダブル プロジェクトのリンカー スクリプトは *…\LinkerScripts\cm0gcc.ld* にあります。

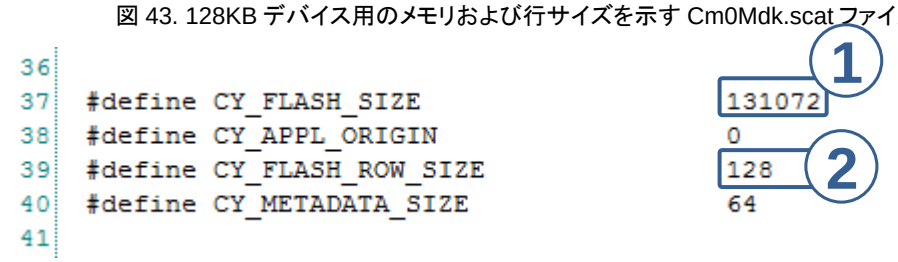
ユーザーのデバイス用に入力する値がわからない場合、PSoC Creator よって生成されるリンカー スクリプト (使用されるかどうかにかかわらず) にある値を参照してください。これは *%PROJECT_DIR%\Generated_Source\PSoC4\cy_boot* フォルダにあり、*cm0gcc.ld* として名付けられます。

A.5.2 MDK コンパイラ

MDK コンパイラの場合、ブートローダブル プロジェクトのリンカー スクリプトのみは変更する必要があります。リンカー スクリプトは*…\LinkerScripts\Cm0Mdk.scat* フォルダにあります。

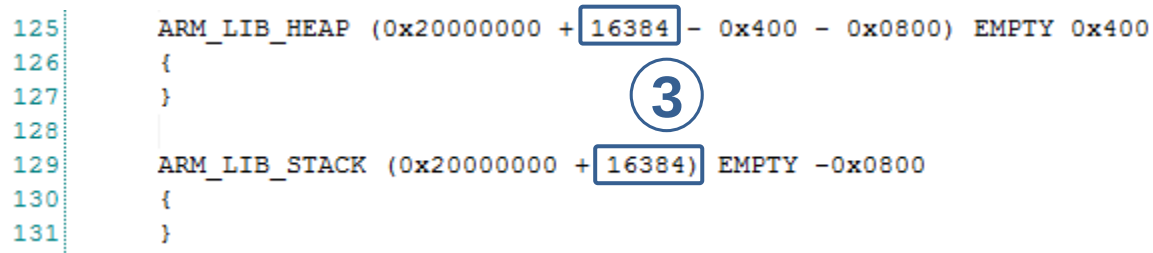
1. 256KB デバイスの場合、デバイスのフラッシュ メモリ サイズを 262144 に変更します (図 43 を参照してください)。
2. 256KB デバイスの場合、デバイスの行サイズを 256 に変更します (図 43 を参照してください)。

図 43. 128KB デバイス用のメモリおよび行サイズを示す Cm0Mdk.scat ファイルの内容



3. 256KB デバイスの場合、デバイスの RAM サイズ (両方の値) を 32768 に変更します (図 44 を参照してください)。

図 44. 128KB デバイス用の RAM サイズを示す Cm0Mdk.scat ファイルの内容



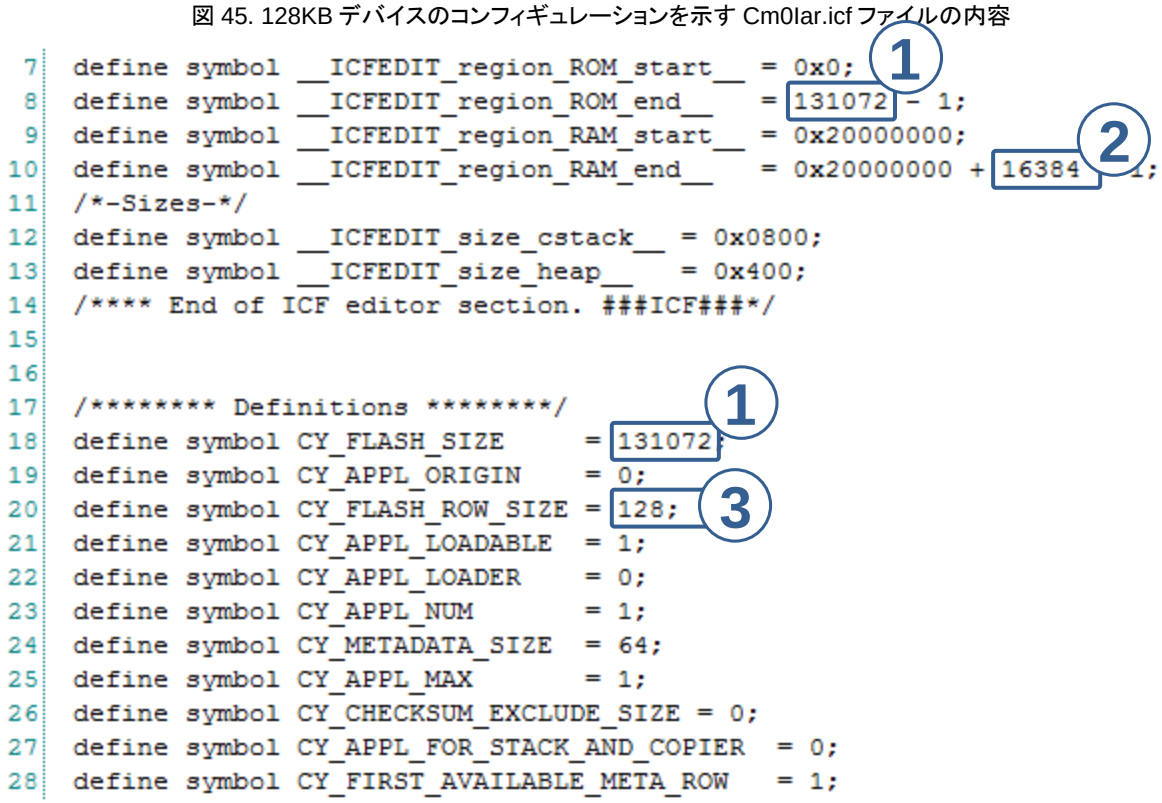
ユーザーのデバイス用に入力される値がわからない場合、PSoC Creator よって生成されるリンカー スクリプト (プロジェクトが再ビルドされた後に使用されるどうかにかかわらず) にある値を参照してください。これは *%PROJECT_DIR%\Generated_Source\PSoC4\cy_boot* フォルダにあり、*Cm0Mdk.scat* として名付けられます。

A.5.3 IAR コンパイラ

IAR を使用する場合、ブートローダ プロジェクトのリンカー スクリプトに対して、PSoC Creator でデバイスが変更された後に PSoC Creator によって生成されるリンカー スクリプトを使用しても問題ありません。しかし、ブートローダブル プロジェクトのリンカー スクリプトを修正する必要があります。これは…*\LinkerScripts\Cm0Iar.icf* フォルダにあります。

1. 256KB デバイスの場合、デバイスのフラッシュ メモリ サイズを 262144 に変更します (図 45 を参照してください)。
2. 256KB デバイスの場合、デバイスの RAM サイズを 32768 に変更します (図 45 を参照してください)。
3. 256KB デバイスの場合、デバイスの行サイズを 256 に変更します (図 45 を参照してください)。

図 45. 128KB デバイスのコンフィギュレーションを示す Cm0Iar.icf ファイルの内容



ユーザーのデバイス用に入力される値がわからない場合、PSoC Creator よって生成されるリンカー スクリプト (プロジェクトが再ビルドされた後に使用されるどうかにかかわらず) にある値を参照してください。これは *%PROJECT_DIR%\Generated_Source\PSoC4\cy_boot* フォルダにあり、*Cm0Iar.icf* として名付けられます。

改訂履歴

文書名: AN97060 - PSoC® 4 BLE および PRoC™ BLE – 無線 (OTA) のデバイス ファームウェア アップグレード (DFU) ガイド

文書番号: 002-15741

版	ECN	発行日	変更内容
**	5403981	08/19/2016	これは英語版 001-97060 Rev. *B を翻訳し た日本語版 002-15741 Rev. ** です。
*A	5827575	07/25/2017	ロゴと著作権を更新しました。
*B	6655977	01/08/2020	これは英語版 001-97060 Rev. *C を翻訳し た日本語版 002-15741 Rev. *B 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 | 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
198 Champion Court
San Jose, CA 95134-1709

© Cypress Semiconductor Corporation, 2015-2020. 本書面は, Cypress Semiconductor Corporation 及び Spansion LLC を含むその子会社 (以下「Cypress」という。) に帰属する財産である。本書面 (本書面に含まれ又は言及されているあらゆるソフトウェア若しくはファームウェア (以下「本ソフトウェア」という。) を含む) は, アメリカ合衆国及び世界のその他の国における知的財産法令及び条約に基づき Cypress が所有する。Cypress はこれらの法令及び条約に基づく全ての権利を留保し, 本段落で特に記載されているものを除き, その特許権, 著作権, 商標権又はその他の知的財産権のライセンスを一切許諾しない。本ソフトウェアにライセンス契約書が伴っておらず, かつ Cypress との間で別途本ソフトウェアの使用方法を定める書面による合意がない場合, Cypress は, (1) 本ソフトウェアの著作権に基づき, (a) ソースコード形式で提供されている本ソフトウェアについて, Cypress ハードウェア製品と共に用いるためにのみ, かつ組織内部でのみ, 本ソフトウェアの修正及び複製を行うこと, 並びに (b) Cypress のハードウェア製品ユニットに用いるためにのみ, (直接又は再販売者及び販売代理店を介して間接のいずれかで) 本ソフトウェアをバイナリーコード形式で外部エンドユーザーに配布すること, 並びに (2) 本ソフトウェア (Cypress により提供され, 修正がなされていないもの) が抵触する Cypress の特許権のクレームに基づき, Cypress ハードウェア製品と共に用いるためにのみ, 本ソフトウェアの作成, 利用, 配布及び輸入を行うことについての非独占的で譲渡不能な一身専属的ライセンス (サブライセンスの権利を除く) を付与する。本ソフトウェアのその他の使用, 複製, 修正, 変換又はコンパイルを禁止する。

適用される法律により許される範囲内で, Cypress は, 本書面又はいかなる本ソフトウェア若しくはこれに伴うハードウェアに関しても, 明示又は黙示をとわず, いかなる保証 (商品性及び特定の目的への適合性の黙示の保証を含むがこれらに限られない) も行わない。いかなるコンピューティングデバイスも絶対に安全ということはない。従って, Cypress のハードウェアまたはソフトウェア製品に講じられたセキュリティ対策にもかかわらず, Cypress は, Cypress 製品への権限のないアクセスまたは使用といったセキュリティ違反から生じる一切の責任を負わない。加えて, 本書面に記載された製品には, エラッタと呼ばれる設計上の欠陥またはエラーが含まれている可能性があり, 公表された仕様とは異なる動作をする場合がある。適用される法律により許される範囲内で, Cypress は, 別途通知することなく, 本書面を変更する権利を留保する。Cypress は, 本書面に記載のある, いかなる製品若しくは回路の適用又は使用から生じる一切の責任を負わない。本書面で提供されたあらゆる情報 (あらゆるサンプルデザイン情報又はプログラムコードを含む) は, 参照目的のためのみに提供されたものである。この情報で構成するあらゆるアプリケーション及びその結果としてのあらゆる製品の機能性及び安全性を適切に設計, プログラム, かつテストすることは, 本書面のユーザーの責任において行われるものとする。Cypress 製品は, 兵器, 兵器システム, 原子力施設, 生命維持装置若しくは生命維持システム, 蘇生用の設備及び外科的移植を含むその他の医療機器若しくは医療システム, 汚染管理若しくは有害物質管理の運用のために設計され若しくは意図されたシステムの重要な構成部分としての使用, 又は装置若しくはシステムの不具合が人身傷害, 死亡若しくは物的損害を生じさせるようなその他の使用 (以下「本目的外使用」という。) のためには設計, 意図又は承認されていない。重要な構成部分とは, それの不具合が装置若しくはシステムの不具合を生じさせるか又はその安全性若しくは実効性に影響すると合理的に予想できるような装置若しくはシステムのあらゆる構成部分をいう。Cypress 製品のあらゆる本目的外使用から生じ, 若しくは本目的外使用に関連するいかなる請求, 損害又はその他の責任についても, Cypress はその全部又は一部をとわず一切の責任を負わず, かつ Cypress はそれら一切から本書により免除される。Cypress は Cypress 製品の本目的外使用から生じ又は本目的外使用に関連するあらゆる請求, 費用, 損害及びその他の責任 (人身傷害又は死亡に基づく請求を含む) から免責補償される。

Cypress, Cypress のロゴ, Spansion, Spansion のロゴ及びこれらの組み合わせ, WICED, PSoC, CapSense, EZ-USB, F-RAM, 及び Traveo は, 米国及びその他の国における Cypress の商標又は登録商標である。Cypress のより完全な商標のリストは, cypress.com を参照すること。その他の名称及びブランドは, それぞれの権利者の財産として権利主張がなされている可能性がある。