AN91162

BLE カスタム プロファイルの作成について

著者: Rohit Kumar

関連製品ファミリ: CY8C4XXX-BL、CYBL10XXX

関連アプリケーション ノート: 完全な一覧について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 ファイルについては http://www.cypress.com/AN91162 を参照してください。

さらにサンプル コードをお求めでしょうか？以下のとおり対応いたします。

PSoC 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 ページを参照してください。
PSoC ビデオ ライブラリはここからご覧ください。

AN91162 は、カスタム BLE プロファイルを採用して PSoC 4 BLE または PRoC BLE デバイスに基づいて Bluetooth® Low Energy (BLE) アプリケーションを開発する方法について説明します。カスタム プロファイルやサービスの概要を提供し、RGB LED 制御を例として PSoC 4 BLE ベースのアプリケーションを構築する手順について説明します。 本アプリケーション ノートは PRoC BLE デバイスにも適用されます。

目次

1 はじめに 2
2 PSoC リソース 2
2.1 PSoC Creator 3
2.2 PSoC Creator ヘルプ 4
2.3 サンプル コード 5
3 標準サービスとカスタム サービス 6
4 カスタム BLE プロファイルの定義 6
4.1 サービスの定義 6
4.2 キャラクタリスティックの定義 6
4.3 ディスクリプタの定義 7
4.4 カスタム UUID の生成 8
5 PSoC Creator プロジェクト: RGB LED カスタム プロファイル 8
5.1 PSoC Creator プロジェクトの作成 10
5.2 コンポーネントの設定 11
5.3 BLE ペリフェラルの設定 17
5.4 RGB LED 制御 21
5.5 プロジェクトのデザイン ワイド リソースの設定 24
5.6 プロジェクトのビルド 26
5.7 プロジェクトにソース／ヘッダ ファイルの追加 26
5.8 プロジェクト ファイル 26
5.9 ファームウェアの設定 27
5.10 ビルドおよびプログラム 33
5.11 CySmart モバイル アプリによるテスト 34
5.12 CySmart Central Emulation Tool によるテスト 35
6 まとめ 39
7 関連情報 39
A 付録 40
A.1 通知の送信 40
改訂履歴 43
ワールドワイドな販売と設計サポート 44

1 はじめに

Bluetooth Low Energy (BLE) は、Bluetooth Special Interest Group (SIG) によって策定された短距離通信向けの超低消費電力無線規格です。BLE 物理層、プロトコル スタック、プロファイル アーキテクチャは消費電力を最小限にするために設計および最適化されます。BLE は、クラシック Bluetooth と同様に 2.4GHz の ISM 周波数帯を使用しますが、1Mbps の低い帯域幅です。

サイプレスの PSoC 4 BLE デバイスは、BLE、プログラマブル アナログとデジタル ペリフェラル機能、メモリ、および Arm® Cortex®-M0 マイクロコントローラーを 1 つのチップに集積したプログラマブル組み込みシステムオンチップ (SoC) です。

本アプリケーション ノートは、BLE コンポーネント GUI を容易に使用し、カスタム BLE プロファイルを作成する方法について説明します。ユーザーはカスタム プロファイルの構造を定義します。ツールは使用される API とイベント コードを自動生成します。同様の手順に従って、ユーザーのカスタム プロファイルに必要なタイプおよび長さのデータを送受信できます。次に、サイプレスの CY8CKIT-042-BLE Pioneer Kit でカスタム プロファイルをテストします。

本アプリケーション ノートは、読者が BLE アーキテクチャおよび関連技術用語についての基礎知識を持っていることを前提としています。

- BLE または PSoC の初心者である場合、アプリケーション ノート「AN91267 - Getting Started with PSoC 4 BLE」を参照してください。
- PSoC Creator 環境における BLE コンポーネントの構造を理解し、標準 BLE サービスに基づいてアプリケーションを開発する方法を学ぶには、アプリケーション ノート「AN91184 - PSoC 4 BLE Designing BLE Applications」を参照してください。
- BLE 仕様の完全な詳細は BLE 開発者ポータルを参照してください。

最新版の BLE Pioneer Kit ソフトウェアは、BLE アプリケーションの開発とデバッグに関するツールを提供するキットのウェブページからインストールしてください。CY8CKIT-042-BLE (BLE Pioneer Kit) は、PSoC 4 BLE と PRoC BLE 両方のデバイス ファミリに対応するサイプレスの BLE 開発キットです。このキットは、Pioneer ベースボードに接続した取り外し可能な PSoC 4 BLE (および PRoC BLE) モジュールから構成されます。このキットを使用することで、本アプリケーション ノートで提供されるサンプル プロジェクトをデモします。このキットには、ユーザーが独自の BLE アプリケーションを開発できるように、BLE サンプル プロジェクトと資料一式が含まれています。

2 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、迅速かつ効率的にデバイスをデザインに統合するための手助けをしています。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 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が含まれています。
- **データシート:** PSoC 41XX-BL および PSoC 42XX-BL デバイス ファミリの電気的仕様を説明します。
- **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を提供します。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。PSoC Creator は追加のサンプル コードを提供します。詳細は「サンプル コード」を参照してください。

  さらに、PSoC デバイス用のサンプル コードと適切なキットを調べるために、「PSoC 3/4/5 Code Examples」ウェブページを参照してください。BLE のサンプル コードは、CY8CKIT-042-BLE Pioneer Kit のサンプル コードの表をスクロールしてください。
- **テクニカル リファレンス マニュアル (TRM):** 各 PSoC 4 BLE デバイス ファミリのアーキテクチャとレジスタについて詳細に説明します。
- **CapSense デザイン ガイド:** PSoC 4 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を説明します。

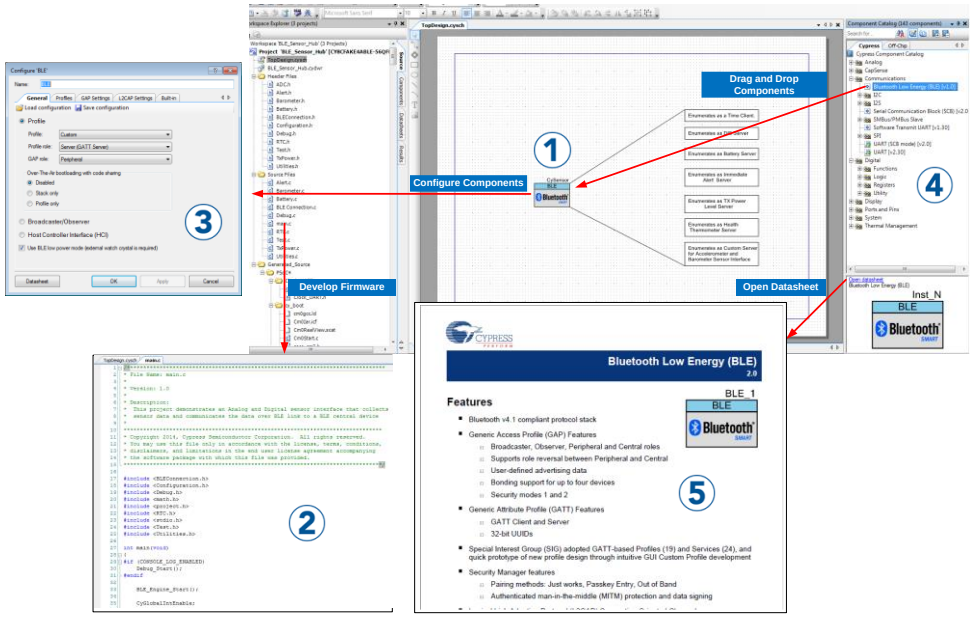
- **開発ツール**
  - CY8CKIT-042-BLE Bluetooth Low Energy (BLE) Pioneer Kit: 使いやすくて安価な BLE 向けのプラットフォームです。Arduino™ 準拠シールドおよび Digilent® Pmod™ドーター カード用コネクタを搭載しています。
  - Windows、iOS および Android 向け CySmart BLE ホスト エミュレーション ツールは、開発される BLE ペリフェラル アプリケーションをテストおよびデバッグするための使い勝手の良い GUI です。
- **テクニカル サポート**
  - よくある質問 (FAQ): サイプレスの BLE エコシステムを詳細に説明します。
  - BLE フォーラム: PSoC 4 BLE と PRoC BLE フォーラムをご覧になることで、ご質問が既に他の開発者によって解答されているか確認していただけます。
  - サイプレス サポート: ご不明な点はございますか？サイプレスのサポート ページを訪れられ、テクニカル サポート ケースを作られるか、またはお近くの代理店までお問い合わせください。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「2」を選択してください。

2.1 PSoC Creator

PSoC Creator は無償の Windows ベースの統合開発環境 (IDE) です。これにより、ユーザーは PSoC 4 BLE と PRoC BLE をベースに、ハードウェアとファームウェア システムを同時に設計できます。図 1 に示すように、PSoC Creator を使用すれば、以下のことを実現できます。

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 データシートをレビュー

図 1. PSoC Creator の回路図入力およびコンポーネント



2.2 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インストールしてください。次に、PSoC Creator を起動して以下の項目を開きます。

- **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を提供します。
- 簡単なコンポーネントサンプル プロジェクト: **File** > **Open** > **Example projects** を選択します。これらのサンプル 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。
- **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 BLE 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「Open Datasheet」を選択します。すべての PSoC 4 BLE 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 BLE コンポーネント データシートのページを参照してください。
- **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、レビューできます。ドキュメント マネージャーを開くには、メニューから **Help** > **Document Manager** を選択します。

2.3 サンプル コード

PSoC Creator は多数のサンプル プロジェクトを含んでいます。これらのプロジェクトは図 2 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。

サンプル プロジェクトにより、空のページではなく完成した設計から始められるため、設計時間を短縮できます。サンプル プロジェクトは PSoC Creator コンポーネントをさまざまなアプリケーションで使用する方法も説明します。図 3 が示すように、サンプル コードおよびデータシートが含まれています。

図 3 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、デバイス ファミリ (PSoC 4、PSoC 4 BLE、PRoC BLE など)、カテゴリまたはキーワードに基づいてサンプル プロジェクトを検索します。
- **Filter Options** に基づいて提供されたサンプル プロジェクトのメニューから選択します。図 3 に示すように、20 以上の BLE サンプル プロジェクトが提供されます。
- 選択したプロジェクトのデータシートをレビューします (**Documentation** タブ)。
- 選択したプロジェクトのサンプル コードをレビューします。このウィンドウからコードをプロジェクトにコピー＆ペーストし、コード開発時間を短縮できます。
- さらに、選択に応じて新規プロジェクト (また、必要な場合は新規ワークスペース) を作成できます。完成した基本的設計から開始することで設計時間を短縮できます。このように、設計を開発されるアプリケーションに適合できます。

図 2. PSoC Creator のサンプル コード

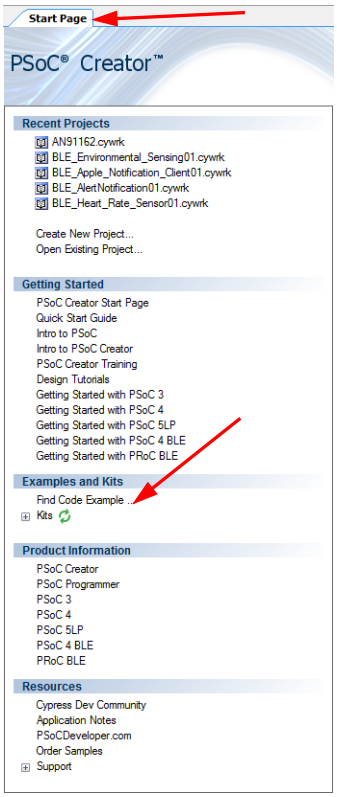
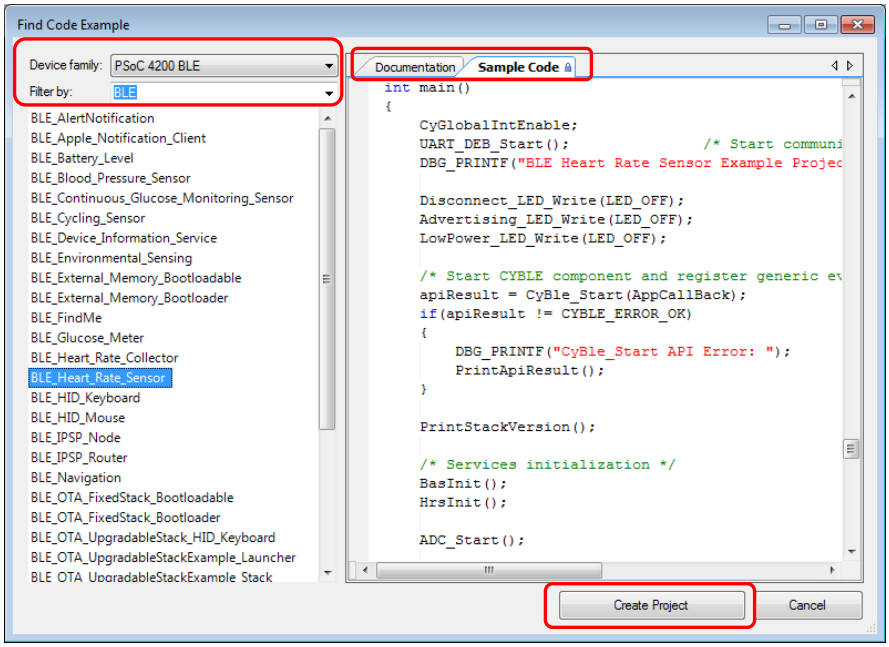


図 3. サンプル プロジェクトおよびサンプル コード



3 標準サービスとカスタム サービス

サービスとは、特定の機能を定義するキャラクタリスティックの集合体です。サービスには 2 種類があります。1 つ目は標準サービスであり、いくつかの一般的な BLE アプリケーション用に Bluetooth SIG によって策定されました。例えば、心拍数、健康温度計、血圧計、アラート通知などです。標準サービスの完全な一覧は Bluetooth 開発者ポータルを参照してください。PSoC 4 BLE に基づいて標準アプリケーションを設計する方法については、アプリケーション ノート「AN91184 - PSoC 4 BLE Designing BLE Applications」を参照してください。

2 つ目のサービスはカスタム サービスです。このサービスでは、名のとおり、カスタム アプリケーション向けに定義され、広く認識されているわけではありません。これらのサービスにより、BLE フレームワークを利用するが BLE SIG によって策定されたサービス セットを超えるカスタム アプリケーションを持つ BLE デバイスを展開できます。カスタム サービスは BLE アプリケーションを開発する誰もが作成できます。本アプリケーション ノートで提供されているサンプル プロジェクトは、BLE Pioneer Kit と BLE 対応携帯電話や PC 間でカスタム RGB LED データを転送できるようにするカスタム サービスについて説明します。

4 カスタム BLE プロファイルの定義

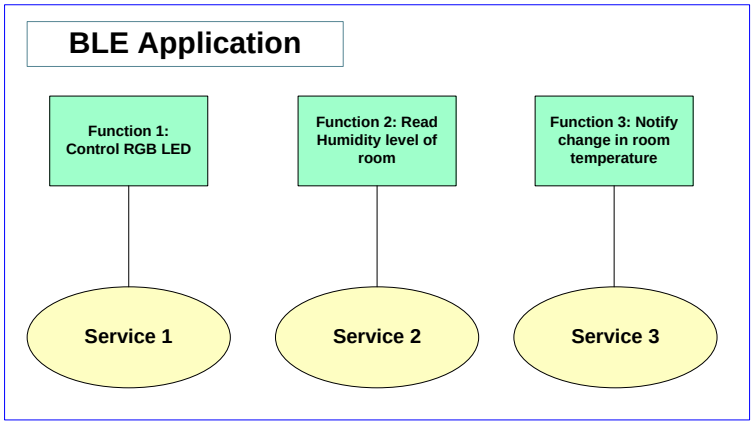
カスタム BLE プロファイルはカスタム サービスとキャラクタリスティックから構成されます。また、標準サービスとキャラクタリスティックを含む場合もあります。

4.1 サービスの定義

カスタム BLE アプリケーションを作成する際、まず分析すべきものはアプリケーションに必要な機能セットです。これらの各機能は、後で必要なデータを取得するカスタム サービスで表します。

例えば、RGB LED の赤色、緑色、青色の輝度を制御する機能です。この機能は、「RGB LED Control」(RGB LED 制御) と名付けるカスタム サービスで表します。その他の例には、室内の湿度や温度を測る機能があります。図 4 に、3 つの機能を実装するカスタム サービスを定義するアプリケーション例を示します。

図 4. カスタム サービスの定義



値だけが異なる機能は、1 つのサービスにまとめられます。RGB LED 制御の例では、4 つの RGB LED 色の値 (赤色、緑色、青色、輝度) を制御する 4 つの異なるカスタム サービスを作る必要がありません。機能として RGB LED の値を制御するだけであるため、1 つのサービスで十分です。サービスを定義した後、各サービスを一意に識別するユニバーサル ユニーク ID (UUID) を割り当てます。カスタム サービスの UUID は 128 ビット値です。

4.2 キャラクタリスティックの定義

次に、各サービスのキャラクタリスティックを定義します。この定義には以下が含まれます。

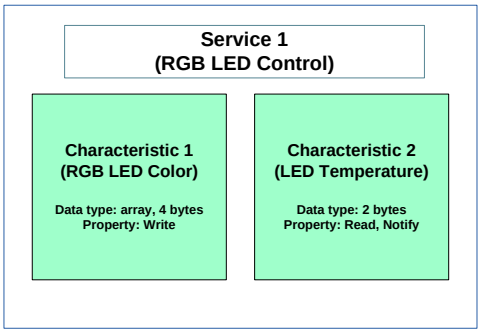
- データ値: データ値は転送されるデータのタイプと長さを記述します。サポートされているデータ タイプには、符号なしバイト、符号付きバイト、ワード、文字列、配列などがあります。

- プロパティ: プロパティはデータにアクセスする方法を記述します。オプションは Broadcast、Read、Write、WriteWithoutResponse、Notify、Indicate、SignedWrite、WritableAuxiliaries です。
- パーミッション: パーミッションはデータのアクセス許可を記述します。パーミッションは暗号化、認証、承認のために設定されます。
- UUID: UUID 値 (128 ビット) はキャラクタリスティックを一意に識別します。

RGB LED 制御の例では、定義したキャラクタリスティックは 4 バイト配列を送信します。その中で 3 バイトは RGB LED の各色の値を定義し、1 バイトは輝度を制御します。キャラクタリスティックの定義は、アプリケーションがデータをどのように解釈するかに依存します。GATT クライアントが新しい RGB LED 値を GATT サーバーに書き込むため、このキャラクタリスティックのプロパティは「Write」となります。

同様に、LED の過熱を監視する基板搭載の温度センサーから 2 バイトの温度情報を提供するもう 1 つのキャラクタリスティックを追加できます。図 5 に、上述のキャラクタリスティックの概要を提供します。

図 5. サービスにおけるキャラクタリスティックの定義

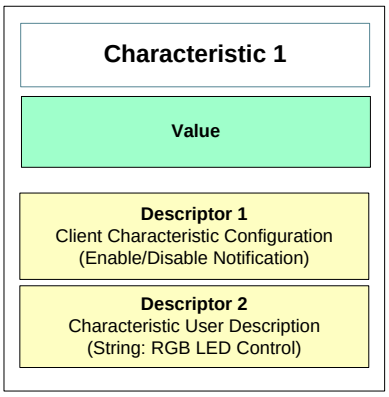


4.3 ディスクリプタの定義

キャラクタリスティックによって、ディスクリプタを追加する場合があります。ディスクリプタはキャラクタリスティック情報をユーザーに提供します。それらは GATT クライアント デバイスによって使用され、通知や表示を有効／無効にすることができます。

図 6 に、カスタムキャラクタリスティックのディスクリプタの一例を示します。この例では、「Client Characteristic Configuration」と呼ばれるディスクリプタは GATT クライアントによって使用されて通知や表示を有効／無効にします。これは通知と表示をサポートしているキャラクタリスティックに属しています。もう 1 つのディスクリプタの例は「Characteristic User Description」です。このディスクリプタは、キャラクタリスティックをユーザー読み出し可能な形式で認識できるようにする文字列を提供します。

図 6. キャラクタリスティックにおけるディスクリプタの定義



4.4 カスタム UUID の生成

すべての BLE カスタム サービスおよびキャラクタリスティックは、識別のために 128 ビット UUID を使用し、ベース UUID は BLE で定義されたベース UUID (00000000-0000-1000-8000-00805F9B34FB) と異なる必要があります 。ベース UUID は 128 ビット値であり、標準 UUID (16 ビットおよび 32 ビット) はこれに基づいて定義されです。

BLE 仕様は BLE サービスとキャラクタリスティックのカスタム UUID を生成する方法を定義しません。ユーザーは BLE で定義されたベース UUID と異なる独自の 128 ビット UUID を生成します。カスタム サービスとキャラクタリスティックの UUID を生成するにはさまざまな方法があります。

サイプレスはカスタム サービスとキャラクタリスティックの UUID を生成するために、以下のメカニズムを使用します。ユーザーは独自の UUID を作成するために、同様の方法を使用できます。

カスタム UUID 値: XXXXYYYY-0000-1000-8000-00805F9B0131

表 1. BLE で定義されたベース UUID からカスタム 128 ビット UUID を生成するサイプレスの方法

UUID の部分	説明
XXXX	デバイス／製品を識別する 16 ビット値
YYYY	特定のサービスまたはキャラクタリスティックの 16 ビット UUID
00805F9B**0131**	すべてのサイプレスのカスタム サービスとキャラクタリスティックのベース UUID。これは、最後の 2 バイトがサイプレスの Bluetooth 割り当て会社 ID (0x0131) に置き換えられた BLE SIG で定義されたベース UUID の下位 6 バイトです。

PSoC 4 BLE 用の XXXX は 0x0003 にセットされています。

例えば、本プロジェクトで RGB LED カスタム サービスを使用します。その 128 ビット カスタム UUID は **0003CBBB-0000-1000-8000-00805F9B0131** にセットされます。ここで、デバイス ID は 0x0003、ベース UUID の下位 6 バイトは 0x00805F9B0131、RGB LED サービス固有の 16 ビット UUID は 0xCBBB にセットされます。

また、独自の 128 ビット UUID を生成するためには http://www.itu.int/en/ITU-T/asn1/Pages/UUID/uuids.aspx ウェブページを参照してください。

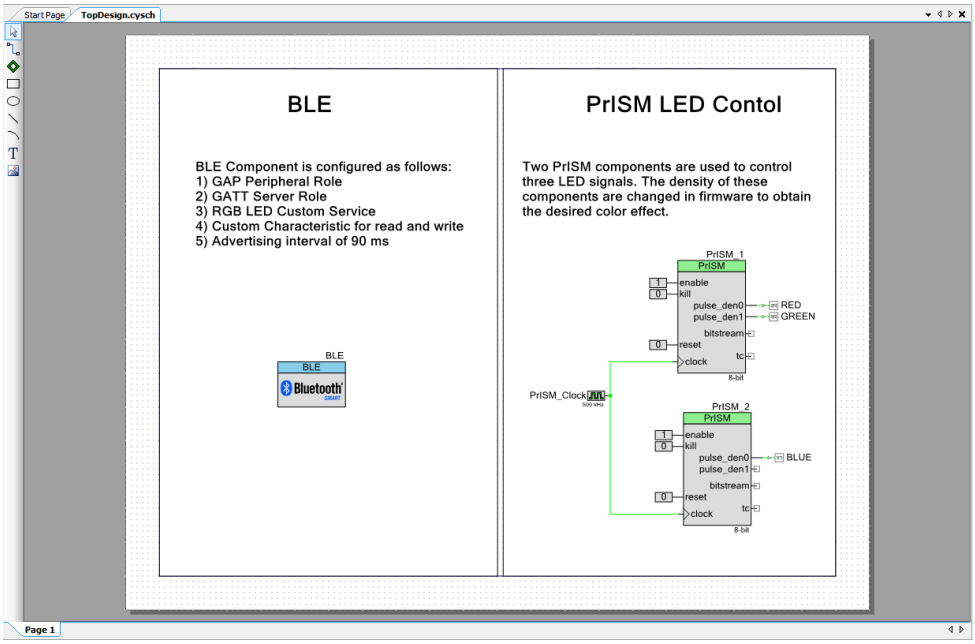
5 PSoC Creator プロジェクト: RGB LED カスタム プロファイル

本プロジェクトを作成および検証するために、以下のものを準備してください。

- PSoC Creator 3.3 SP1 (またはそれ以降) および PSoC Programmer 3.24 (またはそれ以降)
- CySmart™ Central Emulation Tool
- iOS 向け CySmart アプリまたは Android 向け CySmart アプリ
- CY8CKIT-042-BLE Pioneer Kit

本プロジェクトは次の PSoC Creator コンポーネントを使用します: BLE、PrISM、クロック、デジタル出力ピン。図 7 に、PSoC Creator でのプロジェクト回路図を示します。

図 7. PSoC Creator でのプロジェクト回路図



プロジェクトの実装手順は以下のとおりです。

1. PSoC Creator プロジェクトを作成します。
2. PSoC Creator でコンポーネントを設定します。
3. BLE イベントおよびその他のコンポーネントを処理するためにファームウェアを記述します。
4. プロジェクトをビルドし、BLE Pioneer Kit をプログラムします。
5. CySmart ツールまたはアプリでプロジェクトをテストします。

このサンプル プロジェクトは、BLE Pioneer Kit の基板搭載の RGB LED の色と輝度を制御するための RGB LED 制御カスタム サービスを含みます。

RGB LED 制御では、図 8 に示すようにデータ形式を uint8 型の 4 バイト配列として定義します。Write と Read 両方のプロパティがサポートされます。

図 8. RGB LED のデータ形式

RED (0-255)	GREEN 0-255	BLUE 0-255	INTENSITY 0-255

5.1 PSoC Creator プロジェクトの作成

1. **Start** > **All Programs** > **Cypress** > **PSoC Creator 3.3** > **PSoC Creator 3.3** から PSoC Creator を起動します。
2. 新規プロジェクトを作成します (**File** > **New** > **Project**)。PSoC 4100 BLE または PSoC 4200 BLE デザインテンプレートを選択し、CY8C4247LQI-BL483 をデバイスに選択します。プロジェクト名を「AN91162」とし、ワークスペースを所望の場所に保存します。

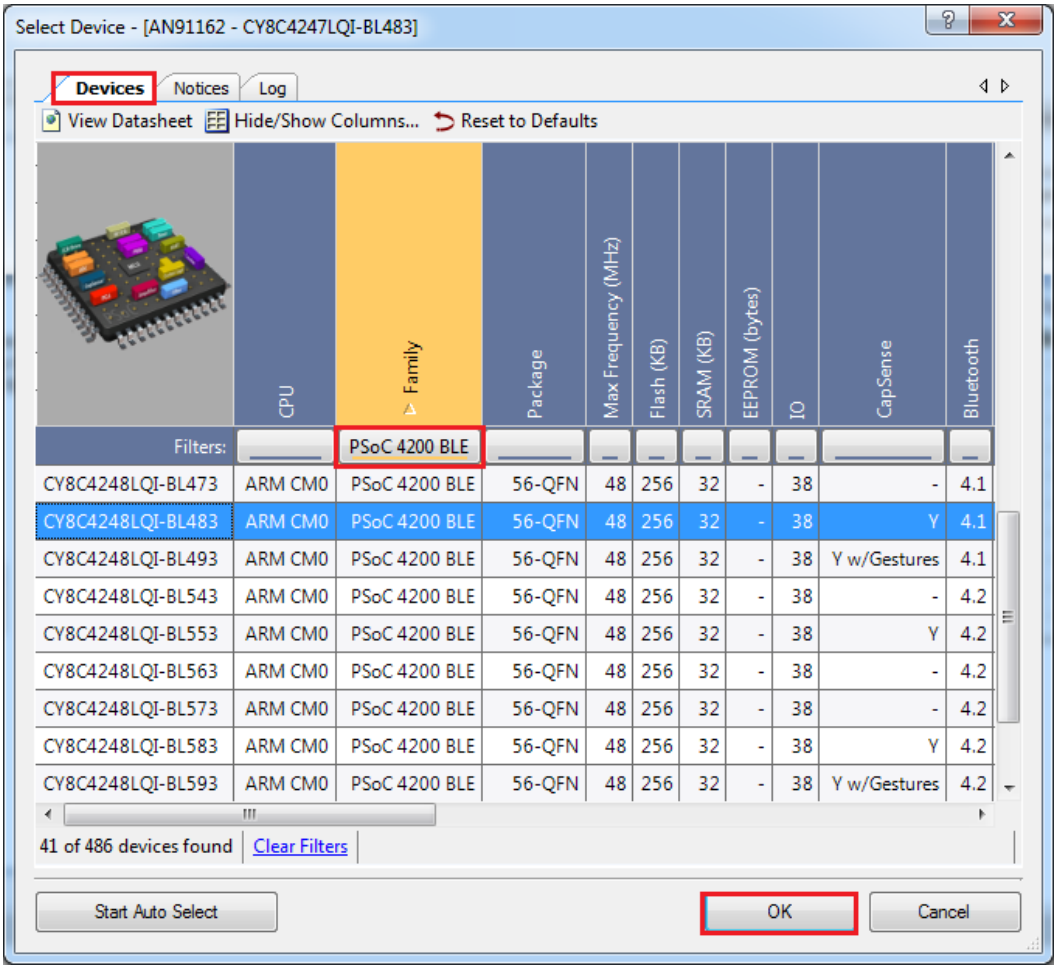
   注: CY8C4XX**7**-BL デバイスは 128K フラッシュと 16K SRAM を内蔵しています。256K フラッシュと 32K SRAM を備えた PSoC 4 BLE を使用する場合、CY8C4XX**8**-BL デバイスを選択してください。これらのデバイスは BLE コンポーネント 2.3 をサポートします。

   CY8C4248LQI-**BL583** は 256K フラッシュと 32K SRAM を備えたデバイスであり、BLE 4.2 をサポートします。BLE 4.2 に対応するデバイスの場合、プロジェクトをデバイス セレクタ設定からこのデバイス番号に変更し、BLE コンポーネントをバージョン 3.0 以上に更新します。

   さまざまな PSoC 4 BLE デバイス間の番号を変更するためには、以下のようにしてください。

   - **Workspace Explorer** でのプロジェクト名を右クリックします。
   - **Device Selector…**を選択します。
   - **Family** を *PSoC 4200 BLE* にセットします。
   - リストからデバイス番号を選択し、**OK** をクリックします (図 9 を参照してください)。

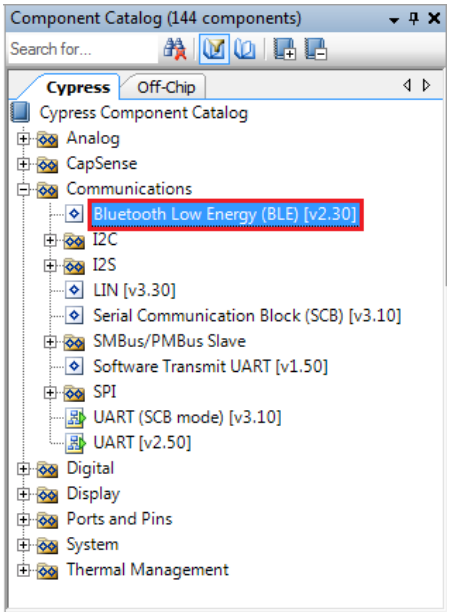
図 9. デバイス セレクタ設定



5.2 コンポーネントの設定

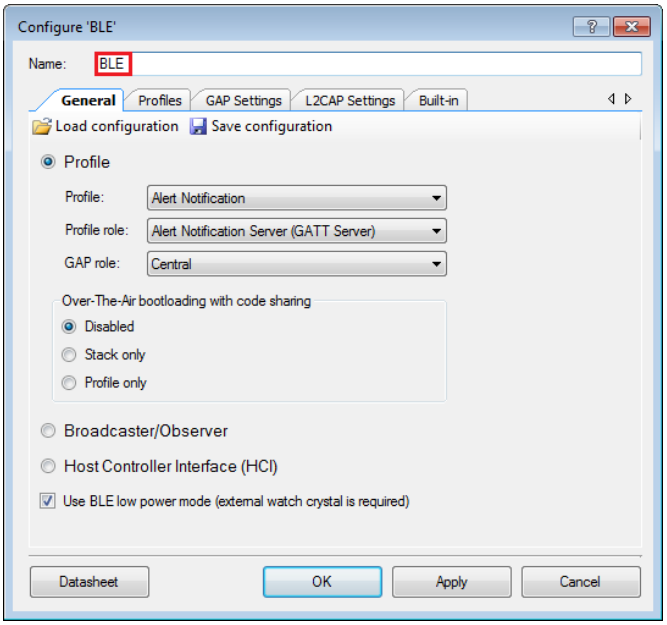
1. 図 10 に示すように、Component Catalog (PSoC Creator IDE の右側) から BLE コンポーネントを Top Design にドラッグ &ドロップします。

図 10. Component Catalog の BLE コンポーネント



2. コンポーネントをダブルクリックし、設定ウィンドウを開きます。図 11 に示すように、コンポーネントのインスタンス名を **BLE** に変更します。

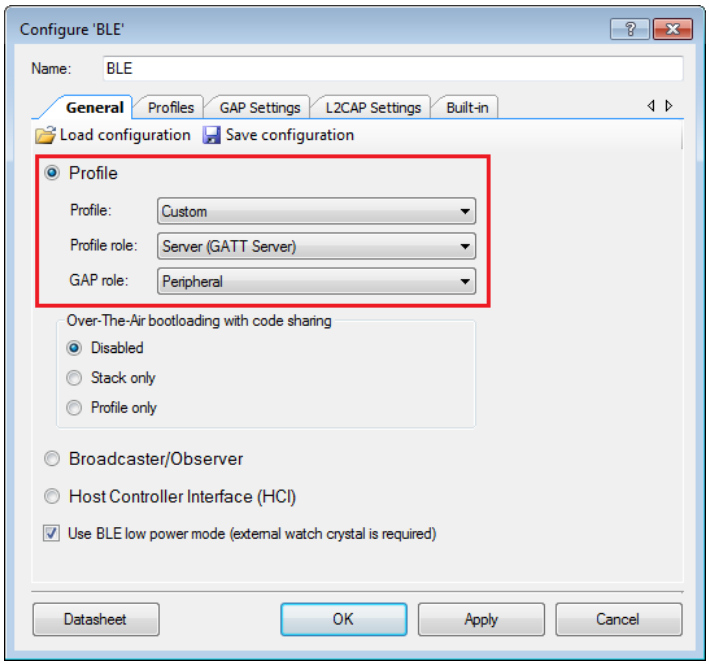
図 11. BLE コンポーネントのインスタンス名



注: 他の PSoC Creator コンポーネントとは異なり、BLE コンポーネントで設定されたインスタンス名は API の命名規則に影響しません。BLE ライブラリを閉じた後、BLE コンポーネントの API はインスタンス名の代わりに常に「*CyBle_*」で始まります。インスタンス名は生成されるファイル名のみを変更します。

3. 図 12 に示すように、**General** タブで **Profile** を選択し、**Profile** オプションを **Custom** にセットします。「Profile role」と「GAP role」の 2 つのオプションはそれぞれ **GATT Server** と **Peripheral** に自動的にセットされます。

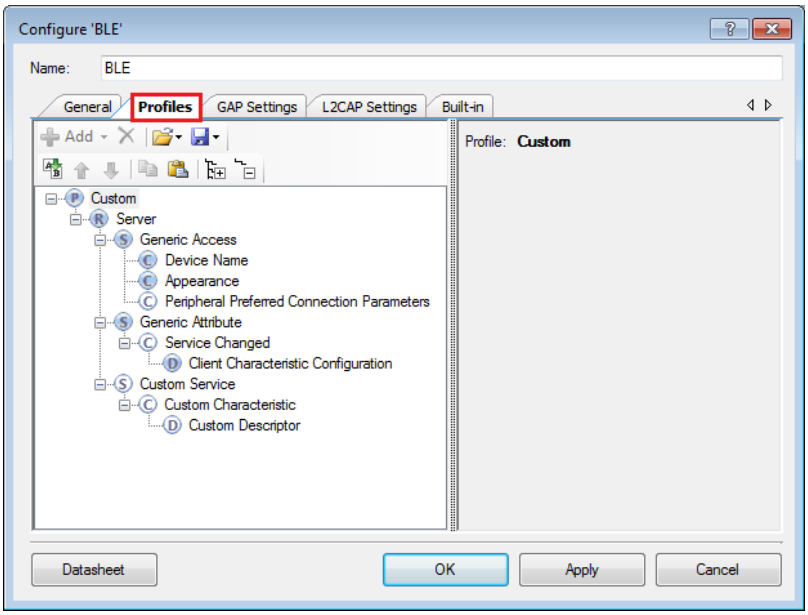
図 12. プロファイル ロールをカスタムに設定



4. **Profiles** タブで、プロファイル固有のパラメーターを設定します。図 13 に示すように、コンポーネントはサービス、キャラクタリスティック、ディスクリプタをプロファイル ツリーの形式で表示します。

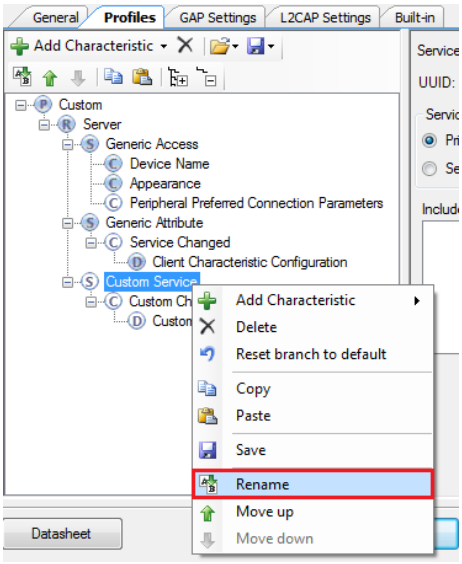
Generic Access と **Generic Attribute** サービスに対する変更は不要です。

図 13. BLE コンポーネントのデフォルト カスタム プロファイル ツリー



5. **Custom Service** を右クリックし、**Rename** を選択します。図 14 に示すように、サービス名を **RGB LED** に変更します。

図 14. カスタム サービス名を変更

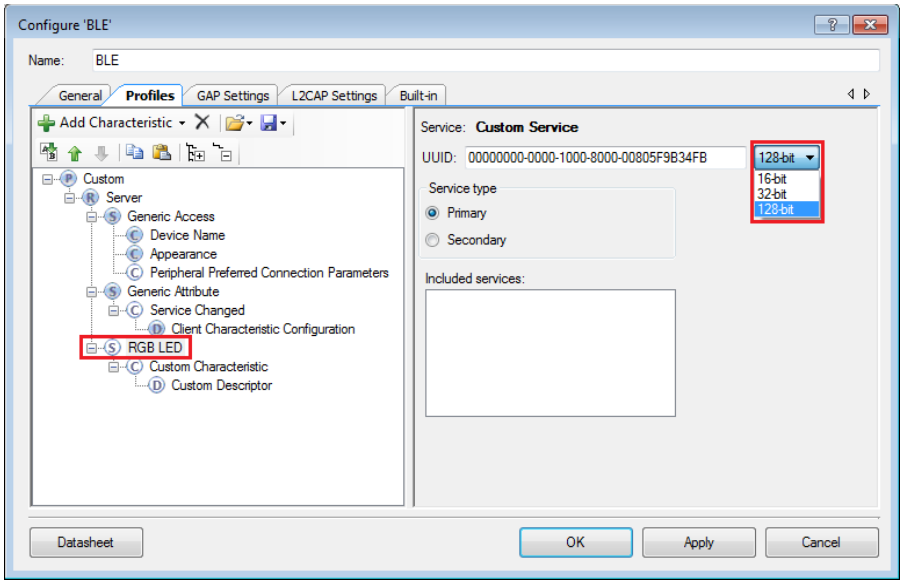


6. 図 15 に示すように、サービス ツリーで **RGB LED** サービスをクリックし、カスタム サービスの UUID 形式を **128 ビット**にセットします。この UUID は GATT サーバー デバイス内の属性を認識するために GATT クライアント デバイスによって使用されます。

   注: コンポーネント内のデフォルトの 128 ビット UUID 値 (00000000-0000-1000-8000-00805F9B34FB) は、BLE SIG によって定義されたベース UUID であり、16 ビットと 32 ビット UUID から完全な 128 ビット UUID を計算するために使用されます。

   標準サービスの既存の UUID との衝突を回避するように、BLE SIG はカスタム属性のためにベース UUID と異なるカスタム 128 ビット UUID を使用することを推奨しています。カスタム UUID を生成する方法は「カスタム UUID の生成」を参照してください。

図 15. 128 ビット UUID 形式を選択

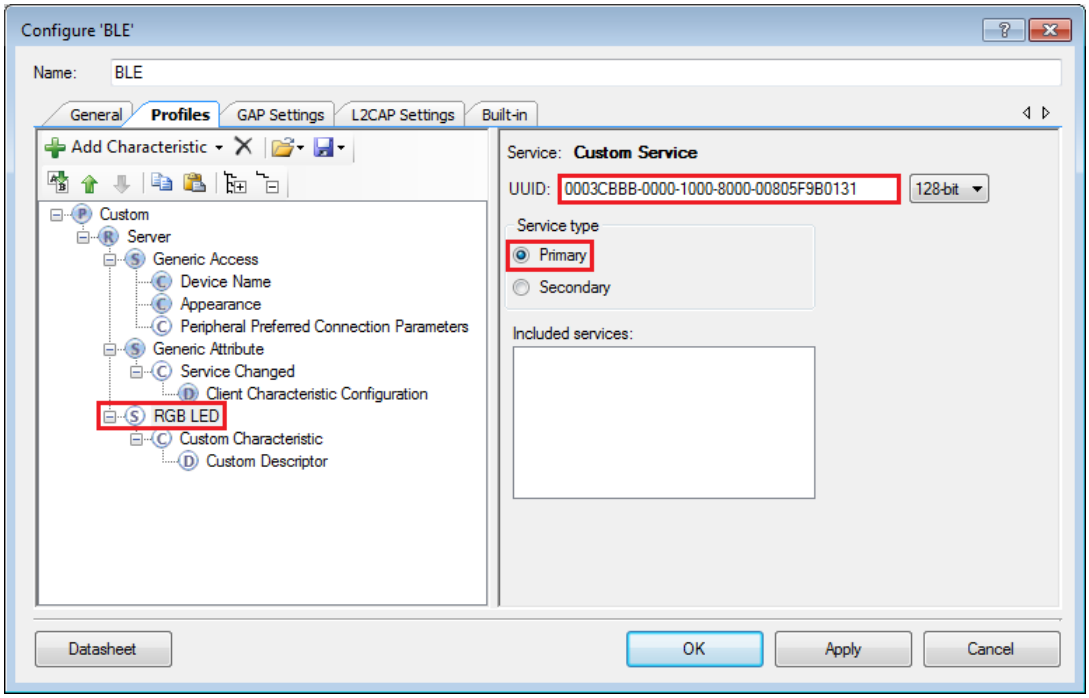


7. 128 ビット UUID 値を 16 進数値 **0003CBBB-0000-1000-8000-00805F9B0131** に変更します。図 16 に示すように、サービス タイプに **Primary** を選択します。

   注: UUID を **0003CBBB-0000-1000-8000-00805F9B0131** に指定する必要があります。サイプレスはこれを RGB LED サービスの UUID として定義します。CySmart アプリでは、この UUID は RGB LED GUI ページを表示するために使用されます。

   他のカスタマ サービス／キャラクタリスティックには、独自の 128 ビット UUID を生成し、このボックスに追加する必要があります。

図 16. RGB LED サービスの UUID をセット

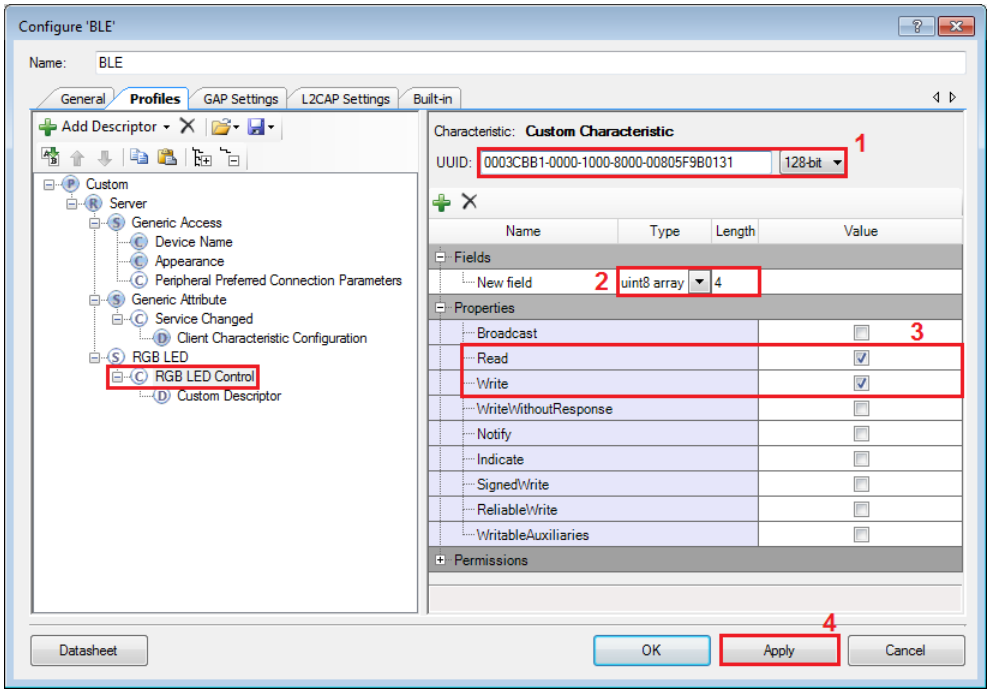


8. **RGB LED** サービス下の **Custom Characteristic** を右クリックし、その名前を **RGB LED Control** に変更し、表 1 のとおりにパラメーターを修正します。これらの変更を図 17 に示します。

表 1. RGB LED キャラクタリスティックパラメーター

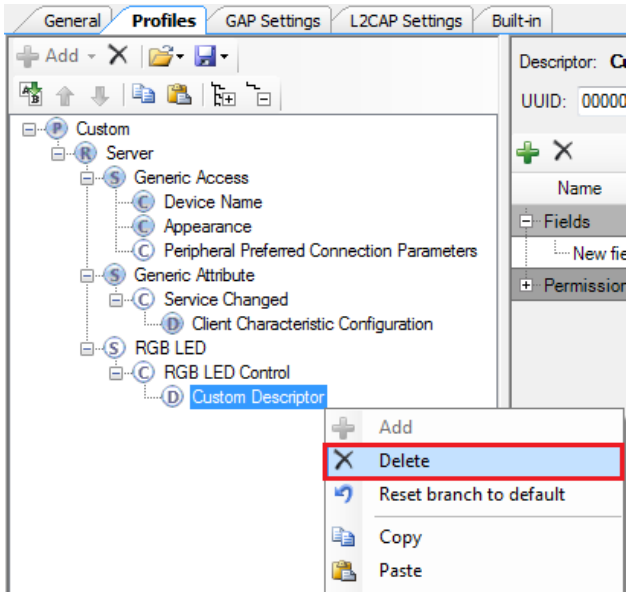
パラメーター	値	説明
UUID	0003CBB1-0000-1000-8000-00805F9B0131	RGB LED キャラクタリスティックの 128 ビット UUID を指定します。モバイル アプリに RGB LED の正しい GUI ページが表示されるためにこの値を UUID として使用します。
Fields	Type: uint8 array Length: 4	転送されるデータのタイプを指定します。
Properties (チェックボックス)	Read、Write	GATT クライアント デバイスがこのキャラクタリスティックに読み書きできることを指定します。

図 17. RGB LED キャラクタリスティック値



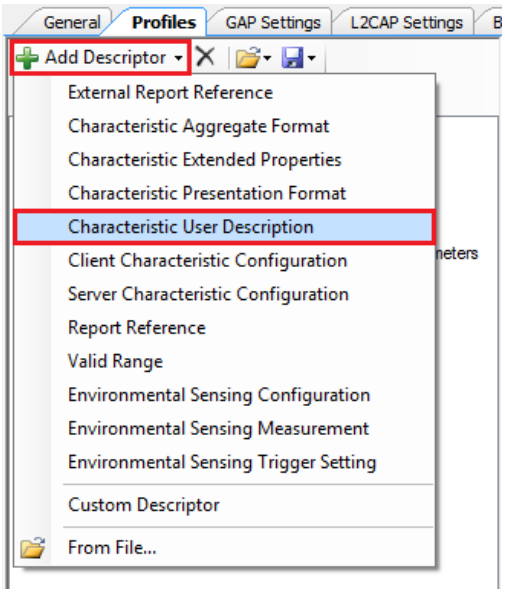
9. このキャラクタリスティックにはカスタム ディスクリプタが必要ないため、図 18 に示すように **Custom Descriptor** を右クリックして **Delete** を選択します。カスタム情報を付加したい場合、カスタム ディスクリプタをキャラクタリスティックに追加できます。

図 18. カスタム ディスクリプタを削除



10. **RGB LED Control** キャラクタリスティックを選択し、ツールバーでの **Add Descriptor** オプションをクリックします。図 19 に示すように、プルダウン リストから **Characteristic User Description** を選択します。

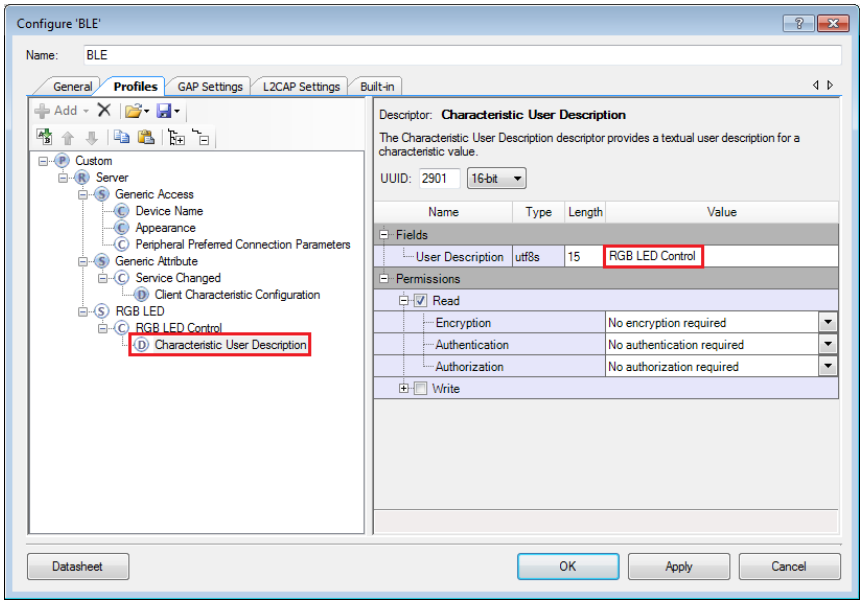
図 19. RGB LED キャラクタリスティックに Characteristic User Description を追加



11. **Characteristic User Description** ディスクリプタをクリックします。図 20 に示すように、画面の右側の **Fields** 下に **Value** フィールドをクリックして **RGB LED Control** の名前をこのキャラクタリスティックのユーザー記述として入力し、**Permissions** を「Read」に選択します。これにより、クライアントは RGB LED Control キャラクタリスティックの名前を読み出せます。

   注: Characteristic User Description ディスクリプタは BLE SIG によって定義された標準ディスクリプタです。BLE 仕様によると、その 16 ビット UUID の値は 0x2901 です。BLE コンポーネントはこのディスクリプタに正しい UUID 値を追加します。ディスクリプタ内の変更は不要です。

図 20. RGB LED キャラクタリスティックの Characteristic User Description



5.3 BLE ペリフェラルの設定

1. BLE コンポーネント設定ウィンドウ内の **GAP Settings** タブでは、**General** 設定下のパラメーターを表 2 のとおりに設定して **Apply** をクリックします。

表 2. ペリフェラル デバイスの一般 GAP 設定

パラメーター	値	説明
Public Address	00A050-XXXXXX	6 バイト Bluetooth デバイス アドレスを指定します。このアドレスはアドバタイズに使用されます。「Silicon generated」オプションが選択された場合、このアドレスの下位 3 バイトはシリコン チップ生成情報となります。BLE SIG の規定によるとこの値は 0 でなくてはいけません。 注: 00A050 はサイプレス セミコンダクタの会社 ID です。
Silicon generated “Company assigned” part of device address	オン	パブリック アドレスの会社割り当て部分がシリコン チップにより生成されることを可能にします。このように設定すると、各デバイスは固有パブリック アドレスを持っています。
Device name	CY Custom BLE	スキャン時に GATT クライアントが見られるデバイス名を指定します。
Appearance	Unknown	デバイスの外観を指定します。このカスタム サービスの場合、「Unknown」を選択します。
MTU size (bytes)	23 (デフォルト)	属性レベルで転送されるプロトコル データ ユニット (PDU) のサイズを指定します。
Adv/Scan TX Power level (dBm)	0 (デフォルト)	データ アドバタイズ時の無線送信の電力レベルを指定します。
Connection TX Power level (dBm)	0 (デフォルト)	接続中のデータ送信時の無線送信電力レベルを指定します。

2. **Peripheral role** 下の **Advertisement settings** をクリックし、表 3 のとおりにパラメーターを設定して **Apply** をクリックします。

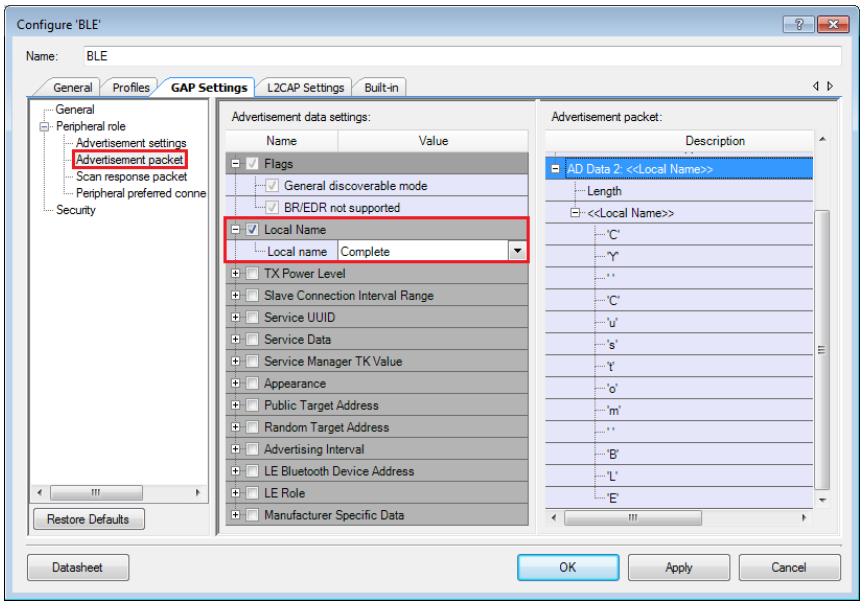
表 3. Advertisement Settings の設定

パラメーター	値	説明
Discovery Mode	General	すべてのセントラル デバイスから認証できるようにデバイスのアドバタイズを全般モードにセットします。
Advertising type	Connectable undirected advertising	ペリフェラルが、特定のセントラル デバイスを優先せずにアドバタイズし、そのアドバタイズをスキャンしたセントラル デバイスからの接続要求を受信するように設定します。
Filter policy	Scan request: Any Connect request: Any	ペリフェラルが特定のデバイスかすべてのセントラル デバイスからスキャンおよび接続要求を受信するかを選択できるように設定します。本プロジェクトでは、すべてのセントラル デバイスからスキャンと接続要求の両方を受信する設定にします。
Advertising channel Map	All channels	ペリフェラルがすべての 3 つのアドバタイズ チャネル (37、38 および 39) を介してアドバタイズするように設定します。
Fast advertising interval: Minimum (ms)	80ms	データ アドバタイズ用の最小の間隔を指定します。実際のアドバタイズ間隔は最小と最大の両方の間隔から計算されます。
Fast advertising interval: Maximum (ms)	100ms	データ アドバタイズ用の最大の間隔を指定します。実際のアドバタイズ間隔は最小と最大の両方の間隔から計算されます。
Fast advertising interval: Timeout (s)	オフ	ペリフェラル デバイスが、時間切れになってさらなるアドバタイズを行わないまでの連続アドバタイズ期間を指定します。 このチェックボックスがオフの場合、アドバタイズはタイムアウトがなく、継続して行われます。

パラメーター	値	説明
Slow advertising interval	オフ	低速アドバタイズ機能を無効にします。 この設定を有効にした場合、ペリフェラル デバイスは高速アドバタイズ タイムアウト後に低速アドバタイズ モードに遷移します。低速アドバタイズ モードでは、アドバタイズ間隔は比較的に長くなりますが、アドバタイズ中の消費電力を節約します。本プロジェクトではこの機能を使用しません。

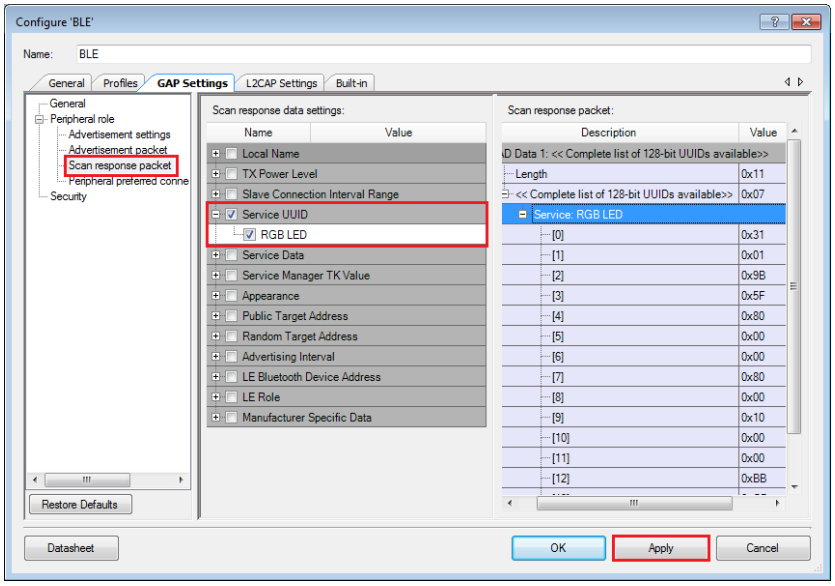
3. **Peripheral role** 下の **Advertisement packet** をクリックし、セントラル デバイスが受信するアドバタイズ パケットの情報を指定します。本プロジェクトでは、図 21 に示すように、「Complete」の **Local name** をアドバタイズ パケットの一部にとして送信するように選択します。

図 21.Advertisement Packet の設定



4. **Peripheral role** 下の **Scan response packet** をクリックし、ペリフェラルがスキャン中に受信したセントラル デバイスからの要求に応答して送信するデータを指定します。本プロジェクトでは、図 22 に示すように **Service UUID** > **RGB LED** サービス データを選択して **Apply** をクリックします。

図 22. ペリフェラルの Scan Response Packet 設定



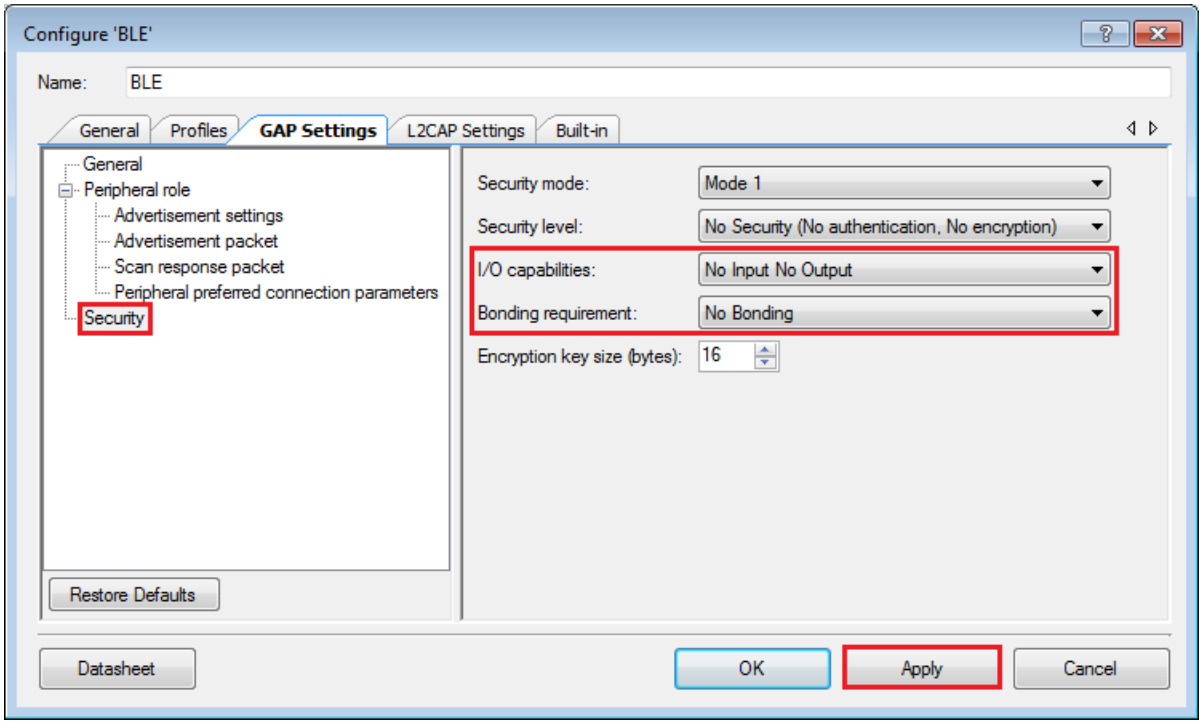
5. 「Peripheral preferred connection parameters」をクリックし、表 4 のとおりにパラメーターを設定します。

表 4. ペリフェラルに必要な接続パラメーター

パラメーター	値	説明
Connection interval: Minimum (ms)	75ms	ペリフェラル デバイスとの接続が確立された後、ペリフェラルとセントラル デバイスが送信モードに遷移してデータのやり取りを行う最小間隔を設定します。最小間隔値が小さいほど、データ転送速度が速くなりますが、消費電力が大きくなります。
Connection interval: Minimum (ms)	80ms	ペリフェラル デバイスが対応できる最大の接続間隔を指定します。セントラルとペリフェラル デバイス間の接続が確立できるためには、接続間隔を一致させる必要があります。実際の接続間隔は接続中にセントラル デバイスとの交渉で決定されます。
Slave latency	0	セントラル デバイスからのデータ要求に対してペリフェラル デバイスが非応答する最大の回数を設定します。より速い速度でデータを送信しながら、送信データがないときに低消費電力モードで維持したいデバイスに対しては特に有用です。 本プロジェクトではスレーブ レイテンシは必要ありません。
Connection supervision timeout (ms)	2000 (2 秒)	直前の接続確立イベント後からの合計時間を設定します。この期間中、ペリフェラルまたはセントラル デバイスは接続がまだ存在していると認識します。この期間中に接続イベントが発生しない場合、接続が切れたと見なされ、デバイスは切断します。

6. **Peripheral role** 下の **Security** をクリックして BLE 通信のセキュリティ レベルを設定します。本プロジェクトはセキュリティの設定を必要としないため、「I/O capabilities」を **No Input No Output** に、「Bonding requirement」を **No Bonding** に設定します。図 23 に示すように、この設定の残り部分を初期設定値のままにし、**Apply** をクリックします。

図 23. Security 設定



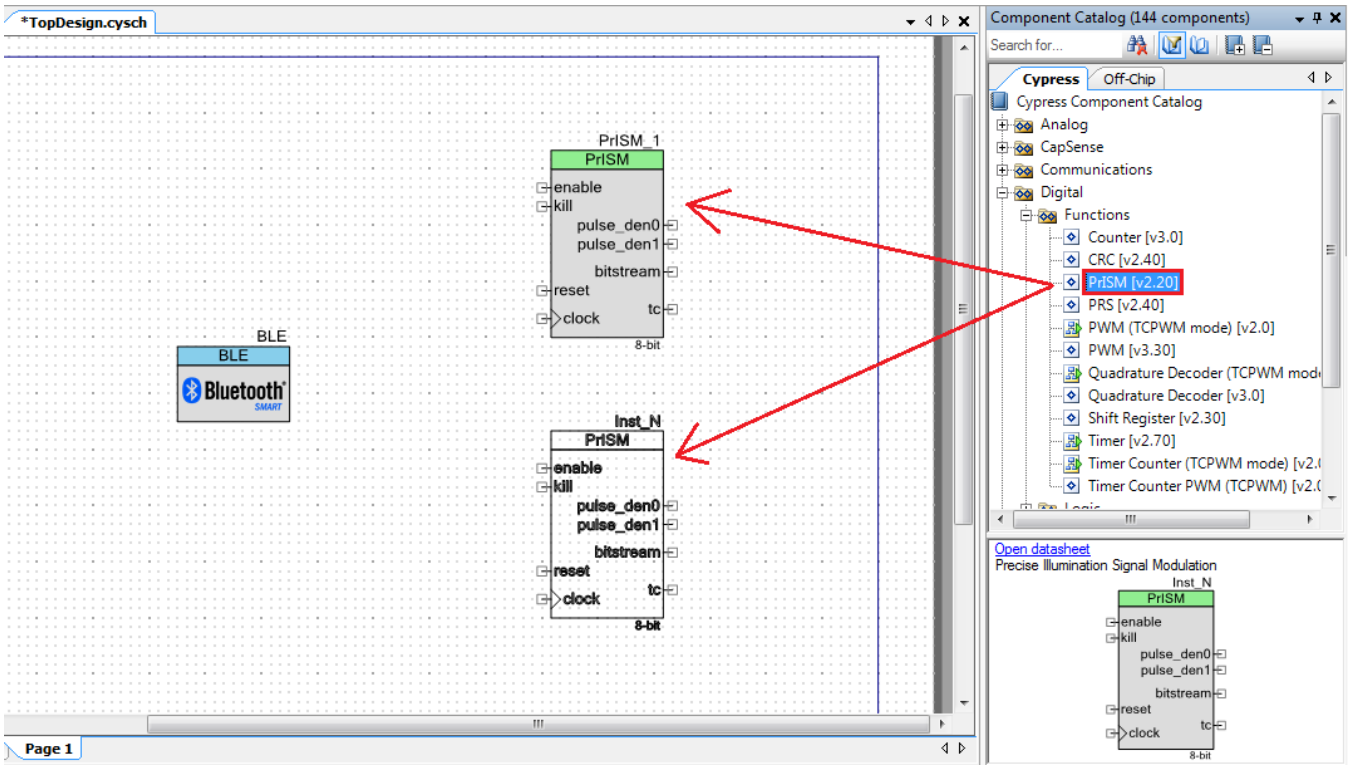
7. **OK** をクリックして変更を保存し、BLE コンポーネントの設定ウィンドウを閉じます。

5.4 RGB LED 制御

RGB LED 制御のために、本プロジェクトではサイプレス独自の LED 輝度調節に関する技術に基づいた PrISM□コンポーネントが使用されます。このコンポーネントは確率信号密度変調を利用して個々の LED の輝度を制御します。複数の LED を組み合わせると、色と輝度の両方を制御できます。詳細は「AN47372 - PrISM™ Technology for LED Dimming」を参照してください。

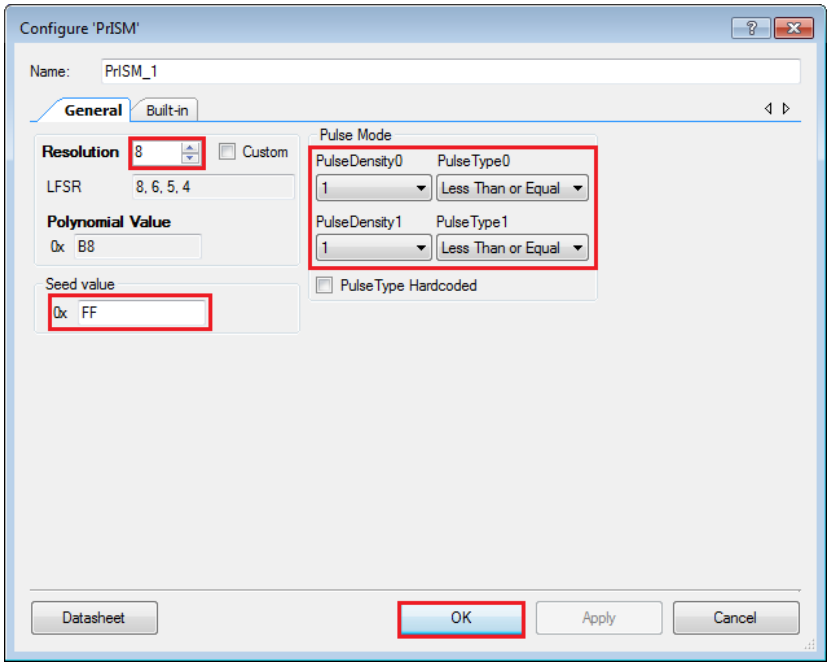
1. 図 24 に示すように、2 個の **PrISM** コンポーネントを Component Catalog (Cypress > Digital > PrISM) からドラッグします。各コンポーネントは 2 つの出力があるため、3 個の LED を制御するには 2 個の PrISM コンポーネントが必要になります。

図 24. PrISM コンポーネントをトップ デザインに配置



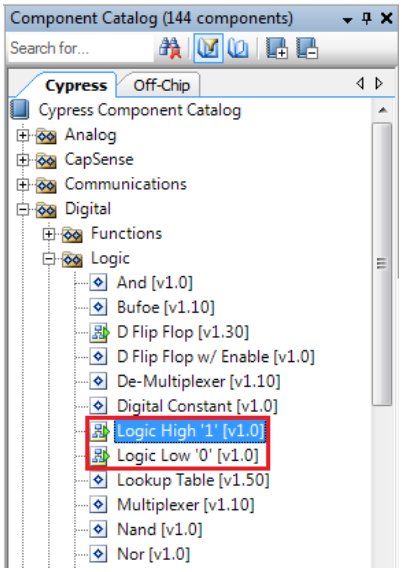
2. 1 つ目の PrISM コンポーネントをダブルクリックします。図 25 に示すように、設定ウィンドウの **General** タブで次のように実行して **OK** をクリックします。
   - **Resolution** を 8 ビットに、**Seed value** を全範囲 (0xFF) にセットします。
   - **Pulse Mode** の下で、**PulseDensity0** と **PulseDensity1** を初期設定値「1」のままにします。生成される乱数はこの値と比較されます。
   - **PulseType0** と **PulseType1** の両方を **Less than or Equal** に選択します。これは、コンポーネントにより生成された乱数が設定されたパルス密度値以下の場合、pulse_den0 と pulse_den1 でのコンポーネント出力は HIGH、そうでない場合、出力は LOW です。

図 25. PrISM コンポーネントの設定



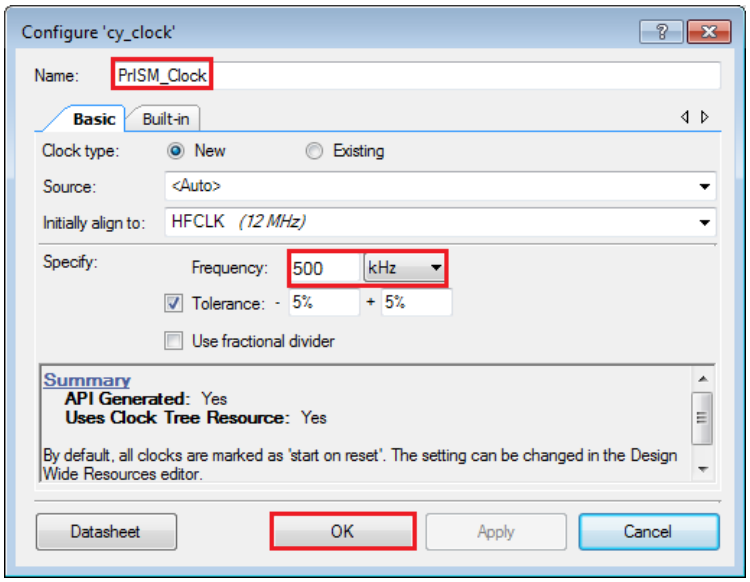
3. **PrISM_2** コンポーネントを同様に設定します。このコンポーネントでは、**pulse_den0** の 1 つの出力のみが 3 番目の LED のために使用されます。残りの出力は未接続のままにします。
4. Component Catalog から次のコンポーネントを両方の PrISM コンポーネントの入力接続に追加します。
   - Logic High (1) コンポーネントを有効な入力ピンに追加し、コンポーネントをデフォルトで有効にします。
   - Logic Low (0) コンポーネントをキルおよびリセット入力ピンに追加し、コンポーネントのハードウェア リセットおよびキル オプションを無効にします。これらの 2 つのオプションは本プロジェクトに不要です。

図 26. Logic High と Logic Low コンポーネント



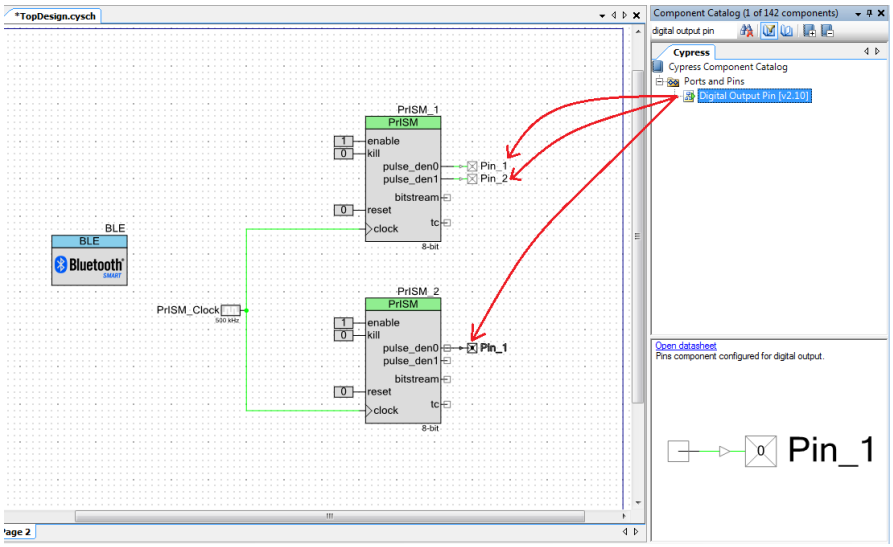
5. Component Catalog の **System** グループから **Clock** コンポーネントをドラッグ &ドロップし、図 27 のとおりに設定して **OK** をクリックします。
   a. インスタンス名を **PrISM_Clock** に変えます。
   b. 「Frequency」を **500kHz** にセットします。

図 27. PrISM クロックの設定



6. ワイヤ ツールを使用してこの **Clock** コンポーネントを両方の PrISM コンポーネントの **clock** 入力に接続します (トップデザインの任意の場所で「w」を押してワイヤ機能を有効にし、接続ポイントをクリックします)。
7. Component Catalog の **Ports and Pins** グループから 3 個の **Digital Output Pin** コンポーネントをドラッグ &ドロップします。図 28 に示すように、それらのコンポーネントを **PrISM_1** の **pulse_den0** と **pulse_den1** ピン、および **PrISM_2** の **pulse_den0** ピンに接続します。これらのピンは PrISM コンポーネントによって駆動され、RGB LED を制御します。

図 28. デジタル出力ピンを PrISM に接続

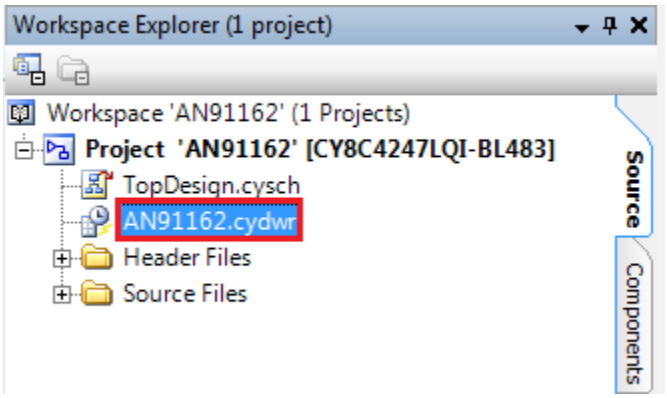


8. Pin_1、Pin_2、および Pin_3 コンポーネントをそれぞれダブルクリックし、**RED**、**GREEN**、**BLUE** に命名します。駆動モードを **High impedance Analog** にセットします。BLE Pioneer Kit の RGB LED がアクティブ LOW のためこれを実行できます。RGB LED ピンの開始時のストロング駆動により RGB LED が短期間で白色に点灯します。

5.5 プロジェクトのデザイン ワイド リソースの設定

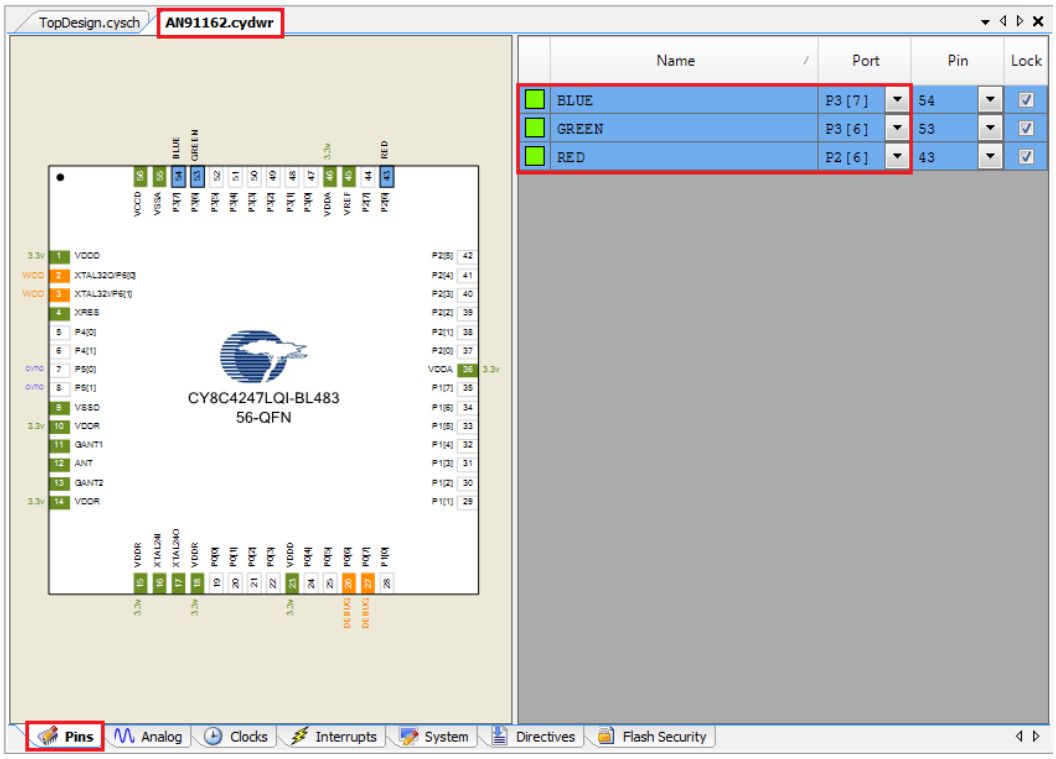
1. コンポーネントにポートを割り当てるためには、図 29 に示すように、Workspace Explorer 内の CYDWR プロジェクトをダブルクリックします。

図 29. プロジェクトの CYDWR を開く



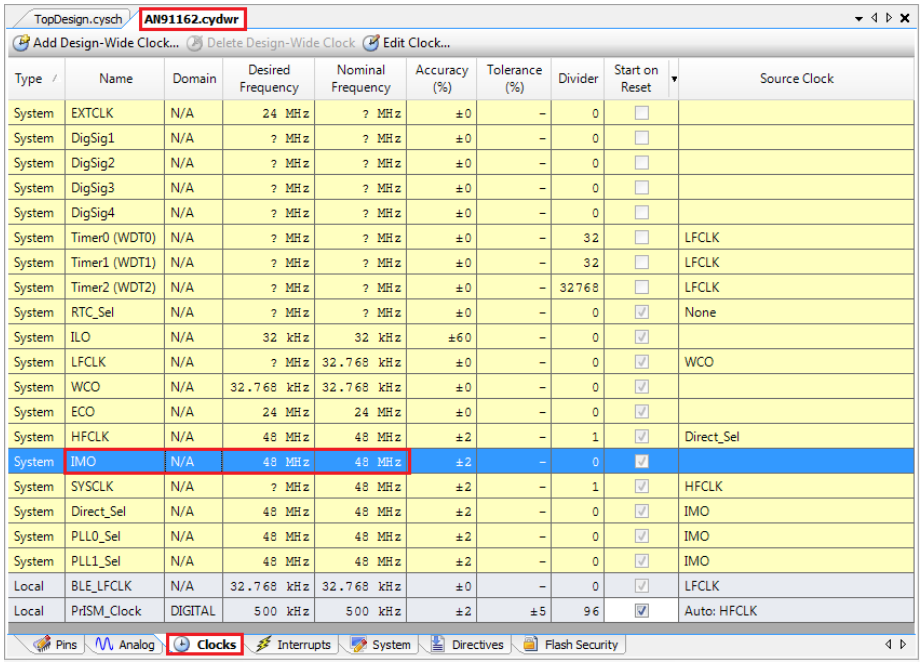
2. 図 30 に示すように、**Pins** タブでは各コンポーネントのピン番号を設定します。ドロップダウン メニューを使用してポート名 (例えば、P3[7]) を入力するか、またはピン名を図内の所望の位置にドラッグすることでポートを割り当てます。

図 30. CYDWR のピン設定



3. 図 31 に示すように、**Clocks** タブでは、**IMO** クロックをダブルクリックしてシステム クロック設定ウィンドウを開きます。

図 31. CYDWR クロック設定

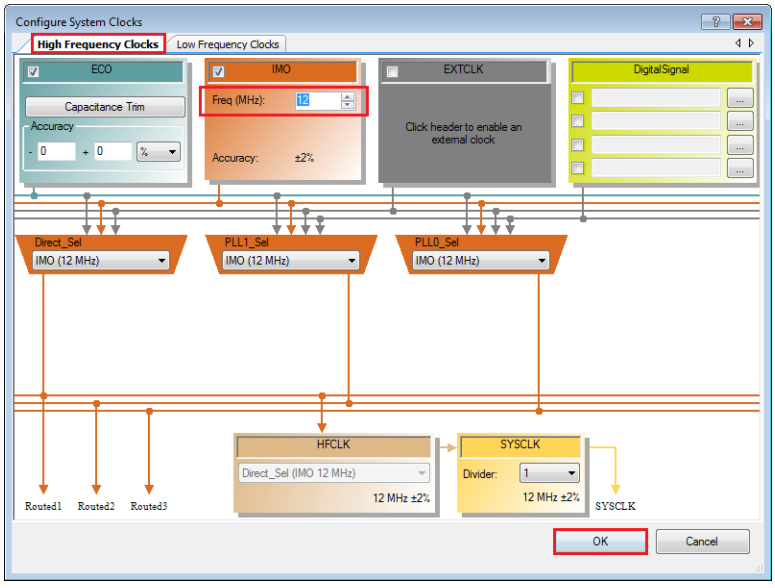


Type	Name	Domain	Desired Frequency	Nominal Frequency	Accuracy (%)	Tolerance (%)	Divider	Start on Reset	Source Clock
System	EXTCLK	N/A	24 MHz	? MHz	±0	–	0	☐	
System	DigSig1	N/A	? MHz	? MHz	±0	–	0	☐	
System	DigSig2	N/A	? MHz	? MHz	±0	–	0	☐	
System	DigSig3	N/A	? MHz	? MHz	±0	–	0	☐	
System	DigSig4	N/A	? MHz	? MHz	±0	–	0	☐	
System	Timer0 (WDT0)	N/A	? MHz	? MHz	±0	–	32	☐	LFCLK
System	Timer1 (WDT1)	N/A	? MHz	? MHz	±0	–	32	☐	LFCLK
System	Timer2 (WDT2)	N/A	? MHz	? MHz	±0	–	32768	☐	LFCLK
System	RTC_Sel	N/A	? MHz	? MHz	±0	–	0	☑	None
System	ILO	N/A	32 kHz	32 kHz	±60	–	0	☑	
System	LFCLK	N/A	? MHz	32.768 kHz	±0	–	0	☑	WCO
System	WCO	N/A	32.768 kHz	32.768 kHz	±0	–	0	☑	
System	ECO	N/A	24 MHz	24 MHz	±0	–	0	☑	
System	HFCLK	N/A	48 MHz	48 MHz	±2	–	1	☑	Direct_Sel
System	IMO	N/A	48 MHz	48 MHz	±2	–	0	☑	
System	SYSCLK	N/A	? MHz	48 MHz	±2	–	1	☑	HFCLK
System	Direct_Sel	N/A	48 MHz	48 MHz	±2	–	0	☑	IMO
System	PLL0_Sel	N/A	48 MHz	48 MHz	±2	–	0	☑	IMO
System	PLL1_Sel	N/A	48 MHz	48 MHz	±2	–	0	☑	IMO
Local	BLE_LFCLK	N/A	32.768 kHz	32.768 kHz	±0	–	0	☑	LFCLK
Local	PrISM_Clock	DIGITAL	500 kHz	500 kHz	±2	±5	96	☑	Auto: HFCLK

4. 本プロジェクトでは、図 32 に示すように、消費電力を低下させるために、IMO 周波数 (高周波数クロック) を初期設定値の 48MHz から 12MHz に下げます。他のクロック設定を初期設定値のままにして、**OK** をクリックします。

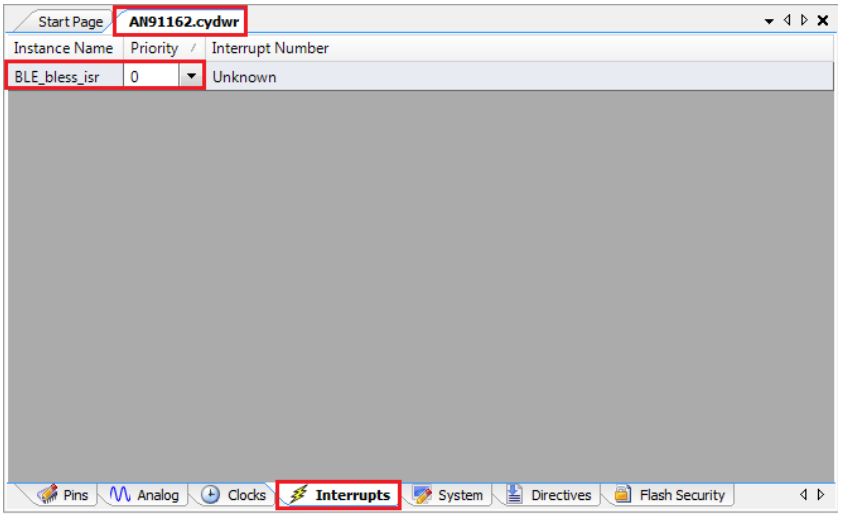
注: ここで設定した CPU クロック周波数はデバイス全体の消費電力に影響を与えます。一方、いくつかのペリフェラルは正常動作のために最小のクロック周波数を必要とします。そのため、消費電力を低く抑えながらプロジェクトの動作を妨げない CPU クロック周波数を選択してください。

図 32. IMO クロック設定



5. 図 33 に示すように、**Interrupts** タブで、BLE 割り込みの優先順位を「0」にセットします。これにより、他の低い優先度割り込みが追加されても、BLE 動作に影響しないように確保されます。

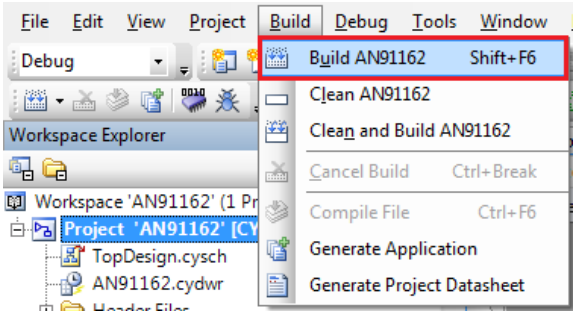
図 33. CYDWR 割り込みの設定



5.6 プロジェクトのビルド

図 34 に示すように、**Build** > **Build AN91162** [Shift+F6] を選択し、完全なプロジェクトをビルドします。

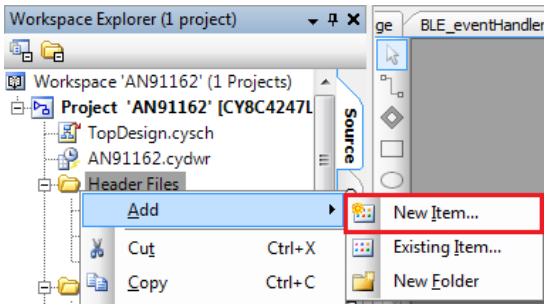
図 34. プロジェクトをビルド



5.7 プロジェクトにソース／ヘッダ ファイルの追加

新しいヘッダ ファイル (H) またはソース ファイル (C) をプロジェクトに追加するために、図 35 に示すように、**Header Files** または **Source Files** を右クリックして、**Add** > **New Item** を選択します。追加するファイル タイプを選択し、所望のファイル名を入力してから、**OK** をクリックします。

図 35. ソース／ヘッダ ファイルを追加



5.8 プロジェクト ファイル

関連プロジェクトは次のファイルがあります。

- **main.c／.h**: これらのファイルはシステムの開始ポイントとして動作する main 関数を含みます。これは BLE を含むシステムを初期化し、BLE イベントの処理のために main 関数を定期的に呼び出します。
- **BLEProcess.c／.h**: これらのファイルは BLE イベントのコールバックを処理し、GATT データベースでの RGB LED キャラクタリスティックの値を更新する関数の定義を含みます。
- **led.c／.h**: これらのファイルは、RGB LED の色と輝度を表示するためのコンポーネントを処理する関数の定義を含みます。

5.9 ファームウェアの設定

本プロジェクトのファームウェアは以下のプロセスを処理します。

- コンポーネントを初期化して、割り込みを有効にします。
- BLE 開始や接続要求、書き込みコマンドなどの BLE スタックにより生成された BLE イベントを処理します。
- GATT クライアントから新しい色のデータを受信すると、RGB LED でその色を表示します。

本プロジェクトは次の BLE API を使用します。

API	説明
CyBle_Start(CYBLE_CALLBACK_T)	BLE コンポーネンを開始し、BLE スタックから生成されたイベントのイベント ハンドラとしての関数を登録します。この関数の引数はイベント ハンドラ関数の名前です。
CyBle_ProcessEvents(void)	BLE スタックとアプリケーションの間の BLE イベントを処理します。これは main 関数で連続的に呼び出す必要があります。この関数は引数がありません。
CyBle_GappStartAdvertisement(uint8)	BLE コンポーネントでセットされた間隔 (表 3 に記述する値) で、BLE ペリフェラル アドバタイズを開始します。引数はアドバタイズを高速／低速／カスタムとして定義します。
CyBle_GattsWriteRsp(CYBLE_CONN_HANDLE_T)	GATT クライアント デバイスが書き込み要求を受信すると、GATT クライアント デバイスに書き込みの応答を送信します。この関数の引数は接続ハンドルです。
CyBle_GattsWriteAttributeValue(CYBLE_GATT_HANDLE_VALUE_PAIR_T *, uint16, CYBLE_CONN_HANDLE_T *, uint8)	データ値が GATT クライアント デバイスによって読み出せるように、属性のデータ値 (キャラクタリスティックなど) を更新します。この関数は、通信されるデータに関わる更新済みデータ、オフセット、接続ハンドル、およびフラグを受信する 4 つの引数があります。

5.9.1 マクロの定義

各ヘッダ ファイルはコードで使用される定数マクロを含みます。各ファイルのマクロを以下に示します。

main.h

```
#define TRUE                                    0x01
#define FALSE                                   0x00
```

BLEProcess.h

```
/* RGB LED Characteristic data length*/
#define RGB_CHAR_DATA_LEN                       4
```

led.h

```
/* LED Color and status related Macros */
#define RGB_LED_MAX_VAL                         0xFF
#define RGB_LED_OFF                             0xFF
#define RGB_LED_ON                              0x00

/* Index values in array where respective color coordinates
* are saved */
#define RED_INDEX                               0x00
```

```
#define GREEN_INDEX                                 0x01
#define BLUE_INDEX                                  0x02
#define INTENSITY_INDEX                             0x03
```

5.9.2 システムの初期化

ファームウェア コンフィギュレーションでの最初のステップはシステムのコンポーネントを初期化することです。次の関数は、*main.c* に入った後、最初に呼び出されます。PSoC Creator ウィンドウの左側にある Workspace Explorer ウィンドウの *main.c* をダブルクリックして開きます。次の関数定義を *main.c* に追加します。

```
void InitializeSystem(void)
{
        /* Enable Global Interrupt Mask */
        CyGlobalIntEnable;

        /* Start BLE stack and register the event callback function. */
        CyBle_Start(GeneralEventHandler);

        /* Start PrISM modules for LED control */
        PrISM_1_Start();
        PrISM_2_Start();

        /* Switch off the RGB LEDs through PrISM modules */
        PrISM_1_WritePulse0(RGB_LED_OFF);
        PrISM_1_WritePulse1(RGB_LED_OFF);
        PrISM_2_WritePulse0(RGB_LED_OFF);

        /* Set Drive modes of the output pins to Strong */
        RED_SetDriveMode(RED_DM_STRONG);
        GREEN_SetDriveMode(GREEN_DM_STRONG);
        BLUE_SetDriveMode(BLUE_DM_STRONG);

}
```

5.9.3 イベント ハンドラの登録

他のコンポーネントの起動と異なり、BLE コンポーネントはコンポーネントを開始するときに**イベント コールバック関数**の登録を必要とします。この関数は BLE イベントを処理するために呼び出されます。BLE イベントはスタック オンなどの一般的なイベント、および接続、切断、書き込みコマンドなどの GAP／GATT 層でのイベントを含みます。一般イベント ハンドラ関数はサンプル プロジェクトの *BLEProcess.c* で定義されます。その関数は個別のファイルまたは *main.c* に格納できます。スイッチ文に含まれるイベントの説明は表 5 を参照してください。以下に示す関数定義では、それぞれのケースは空です。次の節で、各イベントを処理するためにコードを追加します。

```
void GeneralEventHandler(uint32 event, void * eventParam)
{
        /* Structure to store data written by Client */
        CYBLE_GATTS_WRITE_REQ_PARAM_T *wrReqParam;

        /* 'RGBledData[]' is an array to store 4 bytes of RGB LED data*/
        uint8 RGBledData[RGB_CHAR_DATA_LEN];

        switch(event)
{
                case CYBLE_EVT_STACK_ON:
                /* This event is generated when BLE stack is ON */

                break;

                case CYBLE_EVT_GAP_DEVICE_DISCONNECTED:
                /* This event is generated at GAP disconnection. */
```

```
                break;

        case CYBLE_EVT_GATTS_WRITE_REQ:
                /* This event is generated when the connected Central */
                /* device sends a Write request. */
                /* The parameter 'eventParam' contains the data written */

                break;

        default:

        break;
}
}
```

これらのイベントは、BLE 接続を確立できるようにアプリケーションで処理される基本的なイベントです。これらのイベントは 表 5 で説明されます。BLE コンポーネントにより生成できる他のイベントは、BLE_Stack.h *ファイル内の*「CYBLE_EVENT_T」enum で記述されます。

表 5. BLE イベント

イベント名	イベントの説明	イベント処理
CYBLE_EVT_STACK_ON	CyBle_Start()を呼び出した後、BLE スタックが正常に初期化されます。	BLE スタックがオンになると、アドバタイズを開始します。
CYBLE_EVT_GAPP_ADVERTISEMENT_START_STOP	ペリフェラル アドバタイズを開始／停止します。	低消費電力モードに入るか、またはアドバタイズを再起動します。
CYBLE_EVT_GAP_DEVICE_DISCONNECTED	ペリフェラル デバイスとセントラル デバイス間の BLE 接続は切断されます。	低消費電力モードに入るか、またはアドバタイズを再起動します。
CYBLE_EVT_GATT_CONNECT_IND	ペリフェラル デバイスとセントラル デバイス間の接続は確立されました。	接続ハンドルの変数を更新します。 本プロジェクトでは使用しません。
CYBLE_EVT_GATT_DISCONNECT_IND	セントラル デバイスとの接続は切断されました。	GATT データベースの値をリセットします。
CYBLE_EVT_GATTS_WRITE_REQ	GATT クライアント デバイスからの書き込み要求が送信されました。	GATT クライアントから送信されたデータを抽出して、書き込み応答を送信します。

5.9.4 アドバタイズの開始

本プロジェクトは GAP ペリフェラルとして、GAP セントラル デバイスに接続するためにアドバタイズを開始する必要があります。アドバタイズを開始するイベントは 2 つあります。以下のイベントで一般イベント コールバック関数に該当するコードを入れます。

- システムに電源が投入され、BLE スタックがオンのとき (CYBLE_EVT_STACK_ON イベント)

```
case CYBLE_EVT_STACK_ON:
       /* BLE stack is on. Start BLE advertisement */
       CyBle_GappStartAdvertisement(CYBLE_ADVERTISING_FAST);
break;
```

- セントラル デバイスとの接続が切断されたとき (CYBLE_EVT_GAP_DEVICE_DISCONNECTED イベント)

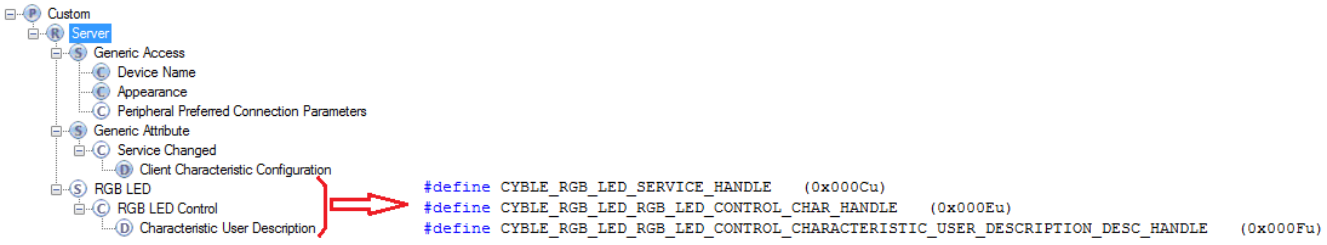
```
case CYBLE_EVT_GAP_DEVICE_DISCONNECTED:
       /* This event is generated at GAP disconnection. */
       /* Restart advertisement */
       CyBle_GappStartAdvertisement(CYBLE_ADVERTISING_FAST);
break;
```

5.9.5 カスタム サービスの属性ハンドル

BLE 通信では、GATT クライアントと GATT サーバーは**属性ハンドル**を使用し、属性のデータ (サービス、キャラクタリスティック、ディスクリプタなど) にアクセスします。この属性ハンドルは、接続が確立された後、属性を一意的に識別する 16 ビットの値です。

BLE コンポーネントに追加されたカスタム サービスおよびキャラクタリスティックに対しては、ハンドルの値はコンポーネントにより生成され、生成されたファイル *BLE_custom.h* 内で#define として見つけられます。本プロジェクトに追加された BLE カスタム サービス (RGB LED) に対しては、生成されたハンドルは図 36 に示されます。

図 36. カスタム サービス用の属性ハンドルのデータ構造



RGB LED サービスは *0x000E* 値の属性ハンドルを使用して、同じキャラクタリスティックに対する読み出しと書き込みの両方をサポートします。

5.9.6 書き込み要求の処理

RGB LED キャラクタリスティックに対しては、GATT クライアントは 4 バイトのデータの書き込み要求を送信します。このデータは、一般イベント コールバック関数の CYBLE_EVT_GATTS_WRITE_REQ イベントの一部として受信されます。受信したデータの属性ハンドルは RGB LED 制御キャラクタリスティックの属性ハンドルと比較されます。これらの値が一致すると、以下の動作が実行されます。

1. 4 バイトのデータが抽出され、配列に格納されます。
2. 基板上の LED 色を更新するために、RGB LED 更新関数 (UpdateRGBLED) が呼び出されます。
3. 内部 GATT データベースの値を更新するために、RGB 制御キャラクタリスティック更新関数 (UpdateRGBcharacteristic) が呼び出されます。
4. 属性ハンドルが RGB LED Control キャラクタリスティックと一致するかどうかを問わず、データが受信されたことをクライアントに通知するために、書き込み応答は、BLE 関数 CyBle_GattsWriteRsp でクライアント デバイスに送信されます。

CYBLE_EVT_GATTS_WRITE_REQ イベントに以下のコードを入れてください。

```
case CYBLE_EVT_GATTS_WRITE_REQ:
       /* Extract the Write data sent by Client */
wrReqParam = (CYBLE_GATTS_WRITE_REQ_PARAM_T *) eventParam;

       /* If the attribute handle of the characteristic written to
       * is equal to that of RGB_LED characteristic, then extract
       * the RGB LED data */
if(CYBLE_RGB_LED_RGB_LED_CONTROL_CHAR_HANDLE ==
wrReqParam->handleValPair.attrHandle)
{
               /* Store RGB LED data in local array */
       RGBledData[RED_INDEX] =
       wrReqParam->handleValPair.value.val[RED_INDEX];
       RGBledData[GREEN_INDEX] =
       wrReqParam->handleValPair.value.val[GREEN_INDEX];
       RGBledData[BLUE_INDEX] =
       wrReqParam->handleValPair.value.val[BLUE_INDEX];
       RGBledData[INTENSITY_INDEX] =
       wrReqParam->handleValPair.value.val[INTENSITY_INDEX];
```

```
                /* Update the PrISM component density value to represent color */
UpdateRGBLED(RGBledData, sizeof(RGBledData));

                /* Update the GATT DB for RGB LED read characteristics*/
                UpdateRGBcharacteristic(RGBledData,
                                        sizeof(RGBledData),
                                        CYBLE_RGB_LED_RGB_LED_CONTROL_CHAR_HANDLE);
}
        /* Send the response to the write request received. */
        CyBle_GattsWriteRsp(cyBle_connHandle);
break;
```

UpdateRGBLED 関数は、受信した 4 バイト値 (赤、緑、青、輝度) を使用して各 RGB LED の輝度を計算します。所望の色を得るために、PrISM コンポーネントの密度値を更新します。以下の (関連プロジェクトの *led.c* ファイルで定義された) 関数をプロジェクトに入れてください。

```
void UpdateRGBLED(uint8* ledData, uint8 len)
{
        /* Local variables to store calculated color components */
        uint8 calc_red;
        uint8 calc_green;
        uint8 calc_blue;

        /* Check if the array has length equal to expected length for
        * RGB LED data */
        if(len == RGB_CHAR_DATA_LEN)
        {
                /* True color to be displayed is calculated on basis of color
                * and intensity value received */
                calc_red = (uint8)
        (((uint16)ledData[RED_INDEX]*ledData[INTENSITY_INDEX])/RGB_LED_MAX_VAL);
                calc_green = (uint8)
        (((uint16)ledData[GREEN_INDEX]*ledData[INTENSITY_INDEX])/RGB_LED_MAX_VAL);
                calc_blue = (uint8)
        (((uint16)ledData[BLUE_INDEX]*ledData[INTENSITY_INDEX])/RGB_LED_MAX_VAL);

                /* Update the density value of the PrISM module */
                PrISM_1_WritePulse0(RGB_LED_MAX_VAL - calc_red);
                PrISM_1_WritePulse1(RGB_LED_MAX_VAL - calc_green);
                PrISM_2_WritePulse0(RGB_LED_MAX_VAL - calc_blue);
        }
}
```

LED 色を設定すると、GATT データベースは RGB LED キャラクタリスティックで更新される必要があります。これは、クライアントが読み出し要求を送信したときに最新の RGB 色を受け取るようにするためです。UpdateRGBcharacteristic 関数は RGB LED 色制御用の属性値を更新します。以下の (関連プロジェクトの *BLEProcess.c* ファイルで定義された) 関数をプロジェクトに入れてください。

```
void UpdateRGBcharacteristic(uint8* ledData, uint8 len, uint16 attrHandle)
{
       /* 'rgbHandle' stores RGB control data parameters */
       CYBLE_GATT_HANDLE_VALUE_PAIR_T            rgbHandle;

       /* Update RGB control handle with new values */
       rgbHandle.attrHandle = attrHandle;
       rgbHandle.value.val = ledData;
       rgbHandle.value.len = len;

       /* Update the RGB LED attribute value. This will allow
       * Client device to read the existing color values over
       * RGB LED characteristic */
CyBle_GattsWriteAttributeValue(&rgbHandle,
                                          FALSE,
                                          &cyBle_connHandle,
                                          CYBLE_GATT_DB_PEER_INITIATED);
}
```

5.9.7 BLE 切断の処理

デバイスはセントラル デバイスから切断されたとき、次の接続が確立される前に RGB LED と GATT データベースはリセットする必要があります。一般イベント コールバック関数の CYBLE_EVT_GATT_DISCONNECT_IND イベントに次のコード スニペットおよびアドバタイズ開始 API 呼び出しを入れてください。

```
case CYBLE_EVT_GAP_DEVICE_DISCONNECTED:
       /* This event is generated at GAP disconnection. */

       /* Reset the color values*/
       RGBledData[RED_INDEX] = FALSE;
       RGBledData[GREEN_INDEX] = FALSE;
       RGBledData[BLUE_INDEX] = FALSE;
       RGBledData[INTENSITY_INDEX] = FALSE;

       /* Switch off LEDs */
       UpdateRGBLED(RGBledData, sizeof(RGBledData));

       /* Register the new color in GATT DB*/
       UpdateRGBcharacteristic(RGBledData,
                                                 sizeof(RGBledData),

       CYBLE_RGB_LED_RGB_LED_CONTROL_CHAR_HANDLE);

       /* Restart advertisement */
       CyBle_GappStartAdvertisement(CYBLE_ADVERTISING_FAST);
break;
```

5.9.8 main 関数

一般イベント コールバック関数が完了した後、プロジェクトのコンポーネントを初期化し、BLE イベントを処理するために main 関数を変更します。以下に示すように *main.c* 内の既存 main 関数を変更します。

```
int main()
{
/* Start the components */
       InitializeSystem();

for(;;)
{
/* Process BLE Events. This generates events in the callback function */
              CyBle_ProcessEvents();
}
}
```

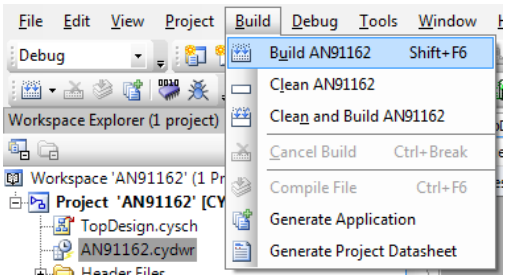
BLE イベントを正常に処理するために、CyBle_ProcessEvents() は各 BLE 接続間隔の間に少なくとも 1 回、周期的に呼び出す必要があります。

完全なファームウェアについては関連プロジェクトを参照してください。

5.10 ビルドおよびプログラム

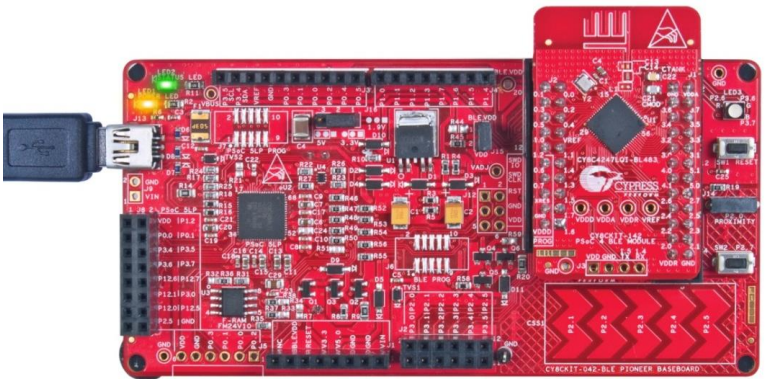
1. 図 37 に示すように、ファームウェアをビルドとコンパイルするために、**Build** > **Build AN91162** を選択します。プロジェクトはエラーや警告なしでビルドされます。

図 37. プロジェクトをビルド



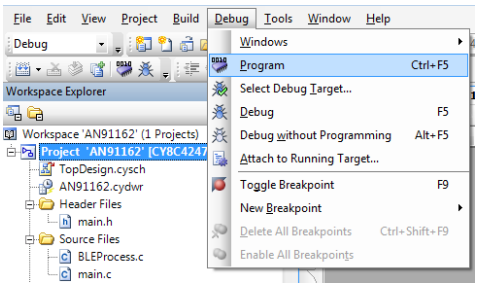
2. BLE Pioneer ベースボードに PSoC 4 BLE モジュール (赤色のモジュール) を差し込み、その後、USB 標準ケーブル A-miniB タイプを使用して PC にキットを接続します (図 38 を参照してください)。PC での USB エニュメレーションを完了させます。

図 38. USB ケーブルで PC に接続



3. **Debug** > **Program** を選択します (図 39 を参照してください)。PC に接続しているのは 1 つのみのキットであれば、プログラミングが自動的に起動します。複数のキットであれば、PSoC Creator はどのキットをプログラムするかという選択を求めます。

図 39. PSoC 4 BLE デバイスをプログラム



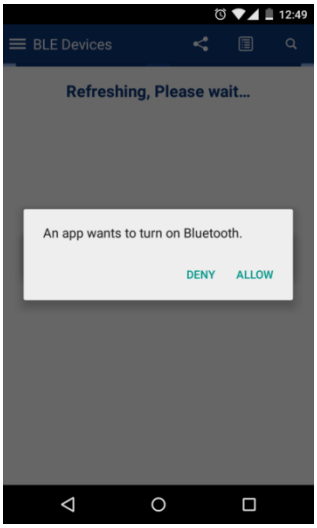
プログラミングが完了した後、BLE Pioneer Kit はアドバタイズを開始します。

5.11 CySmart モバイル アプリによるテスト

1. BLE 対応の携帯電話で CySmart モバイル アプリをダウンロードします。iOS デバイス (iPhone 4S、またはそれ以降) の場合、App Store からアプリをダウンロードします。Android デバイス (Android 4.3、またはそれ以降) の場合、Play Store からアプリをダウンロードします。
2. 携帯電話でアプリを起動します。図 40 に示すように、携帯電話の Bluetooth が有効になっていない場合、アプリはユーザーに Bluetooth の有効化を求めます。

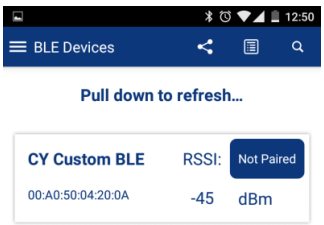
   注: このスクリーンショットは Android 向けの CySmart アプリの画面です。iOS 向けの CySmart アプリには若干の違いがあります。

図 40. 携帯電話で Bluetooth を有効化



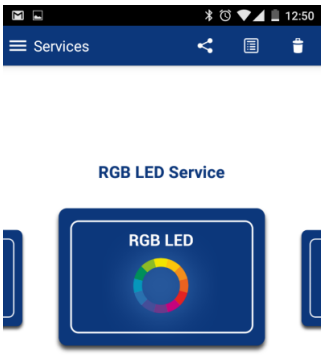
3. Bluetooth が有効化された後、デバイスの画面が表示されます。下方向にスワイプして、PSoC 4 の BLE カスタム サービス プロジェクト「**CY Custom BLE**」と共に、付近の BLE デバイスをリストアップします (図 41 を参照してください)。

図 41. リストされた BLE デバイス



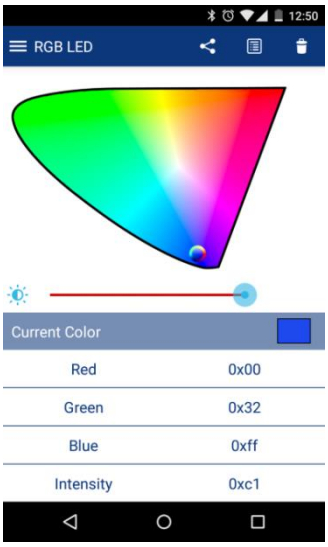
4. **CY Custom BLE** デバイスを選択します。接続の手順が開始し、デバイスが接続されます。接続されたデバイスが対応するプロファイル／サービスのページが画面に表示されます (図 42 を参照してください)。

図 42. サービス ページ



5. RGB LED アイコンが表示されない場合、左また右にスワイプします。RGB LED アイコンが表示されると、そのアイコンを選択します。
6. RGB LED GUI 画面では、色域 (図 43 を参照してください) は赤、緑、青の色成分の値を制御し、リニアスライダーは輝度を制御します。スライダーで輝度を上げて色域をスワイプして、BLE Pioneer Kit の RGB LED にセットされたその色を確認します。

図 43. RGB LED 色の制御



7. デバイスを切断するために、デバイス検索ページが表示されるまで、アプリでの **Back** ボタンをタップします。

5.12 CySmart Central Emulation Tool によるテスト

CySmart Central Emulation Tool は、BLE ドングルと共に、BLE GATT クライアント デバイスをエミュレートします。これにより、ユーザーは任意の BLE デバイスに接続し、デバイスの属性を検出し、属性を通してペリフェラル デバイスとデータを通信できます。www.cypress.com/cysmart から最新版の CySmart Central Emulation Tool をダウンロードし、www.cypress.com/CY8CKIT-042-BLE から BLE ドングルの最新のファームウェア HEX ファイルをダウンロードしてください。

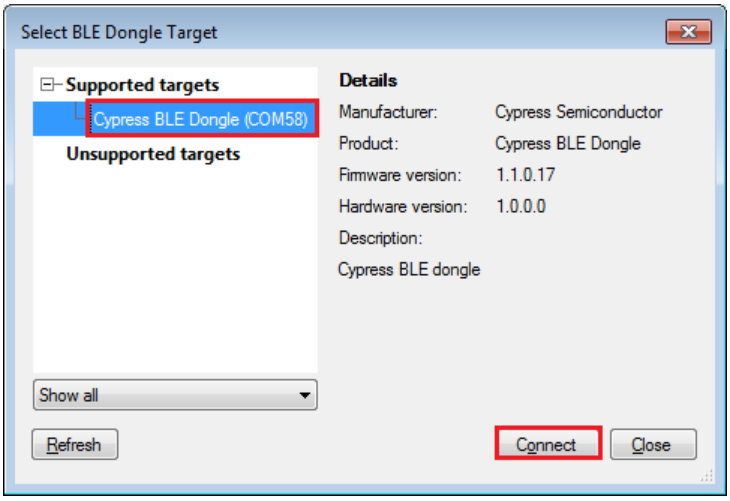
注: CySmart Central Emulation Tool は現在 Windows PC のみに対応しています。

CySmart Central Emulation Tool を使用してプロジェクトをテストするために、以下の手順に従ってください。

1. BLE ドングルを PC に接続します。USB エニュメレーションを完了させます。
2. CySmart ツールを起動するために、**Start** > **All Programs** > **Cypress** > **CySmart <*バージョン*>** > **CySmart <*バージョン*>**をクリックします。
3. 図 44 に示すように、CySmart セントラル エミュレーション ツールで、**Supported targets** リストから **Cypress BLE Dongle** を選択して、**Connect** をクリックします。

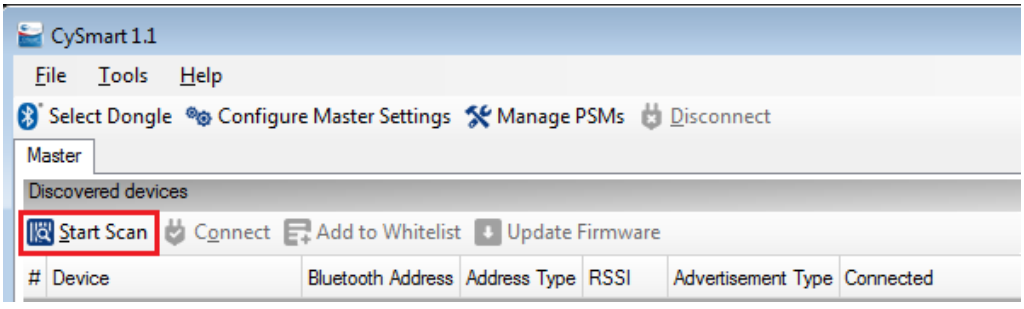
   注: BLE ドングルがリストアップされない場合、BLE ドングルでのリセット ボタンを押して **Refresh をクリックしてください。**

図 44. BLE ドングル ターゲットを選択



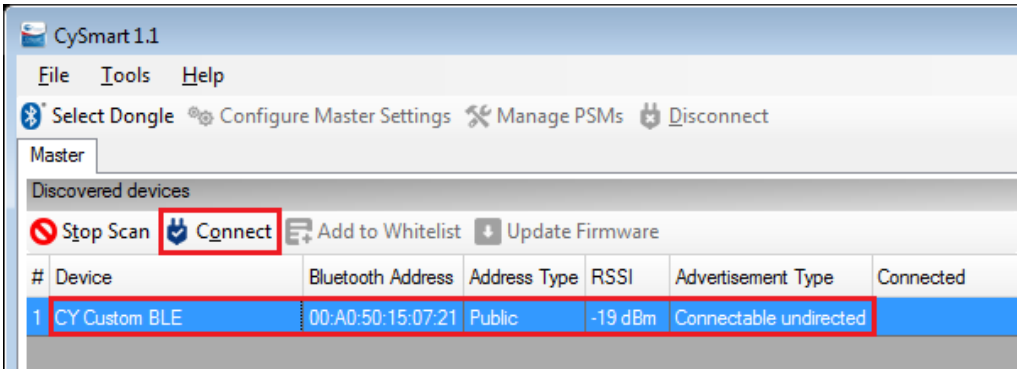
4. BLE ドングルが選択された後、BLE ペリフェラル デバイスのスキャンを開始するために、図 45 に示すように、左上にある **Start Scan** をクリックします。

図 45. CySmart Central Emulation Tool でのスキャンを開始



5. 図 46 に示すように、**CY Custom BLE** デバイスを選択して、**Connect** をクリックします。

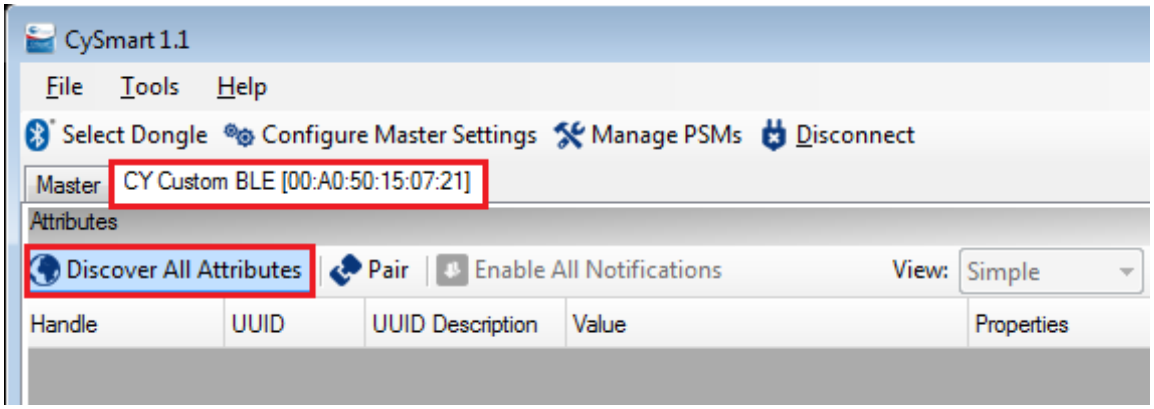
図 46. BLE_Custom デバイスに接続



デバイスが接続された後、図 47 に示すように、CySmart Central Emulation Tool で Master タブの隣に、接続先のデバイスの名前を付けられた新しいタブが開きます。

6. 図 47 に示すように、CySmart ツールで **CY Custom BLE** デバイスがサポートする属性の問い合わせを開始するために、**Discover All Attributes** を選択します。

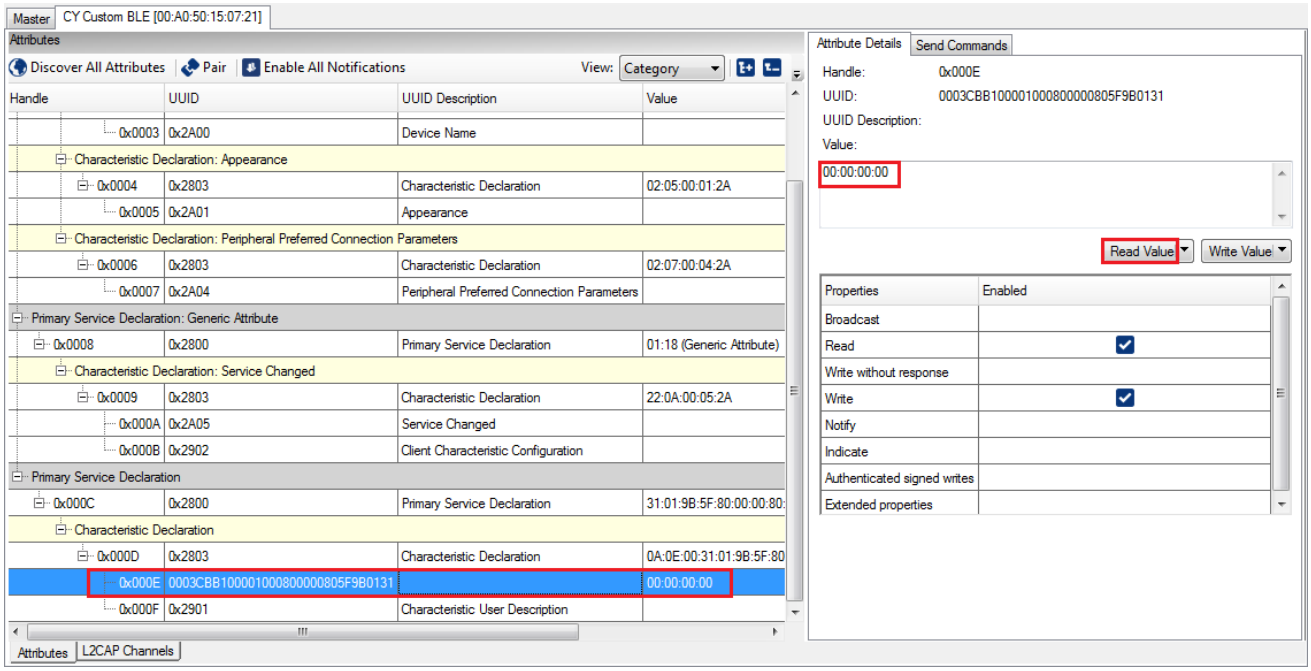
図 47. すべての属性を問い合わせる



属性のリストをスクロールダウンし、RGB LED カスタムキャラクタリスティック (UUID **0003CBB1-0000-1000-8000-00805F9B0131**) をクリックします。CySmart ウィンドウの右側の Attribute Details タブに表示されるように、このキャラクタリスティックは読み出しと書き込みの両方をサポートします。

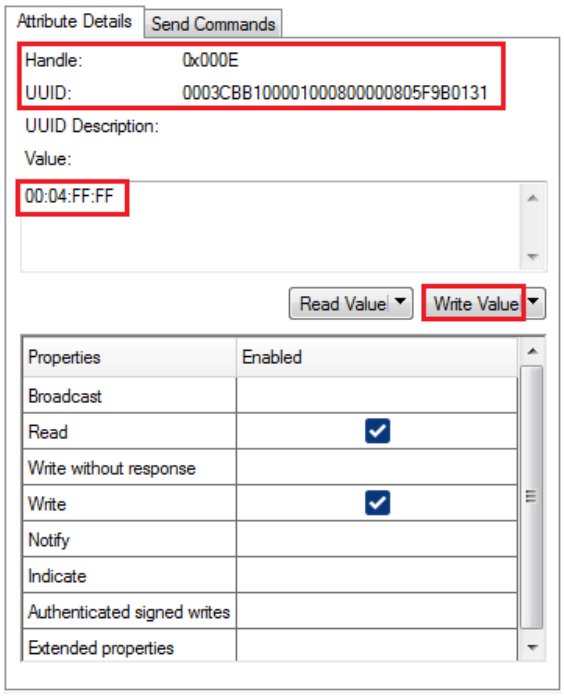
図 48 に示すように、既存の色値を読み出すために、**Read Value** をクリックします。Value フィールドに 4 バイト値が表示されます。

図 48. RGB LED カスタムキャラクタリスティック



7. 図 49 に示すように、**Value** フィールドでの 4 バイト値に 0 以外の値を書き込んで、新しい色値を送信するために **Write Value** をクリックします。4 バイト値のフォーマットは**赤:緑:青:輝度**であり、最小値は「0」で、最大値は「FF」です。

図 49. カスタムキャラクタリスティックに新しい色値を書き込む

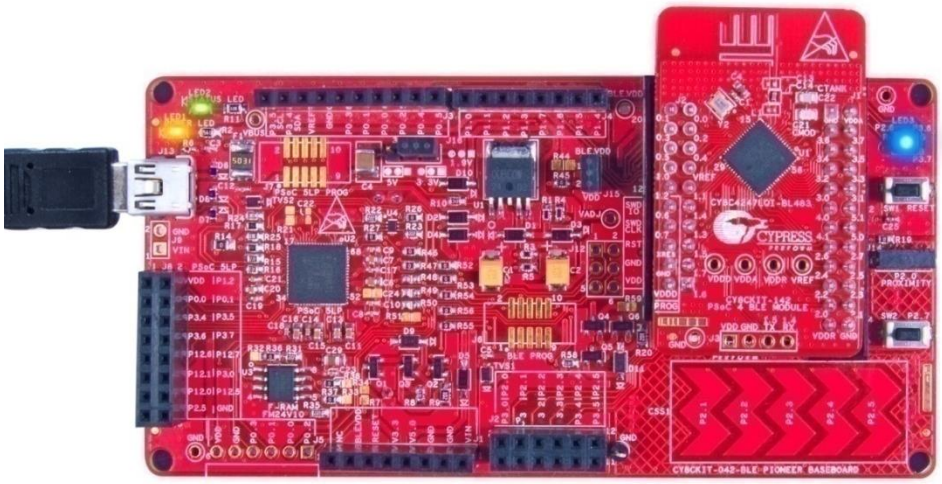


他の 4 バイト データを送信して対応する色を観察してください。

RGB データ	観察された色
00:00:00:00	色なし
FF:00:00:FF	赤色
00:FF:00:FF	緑色
00:00:FF:FF	青色
FF:00:FF:22	紫色、低輝度
FF:FF:00:55	黄色、中輝度

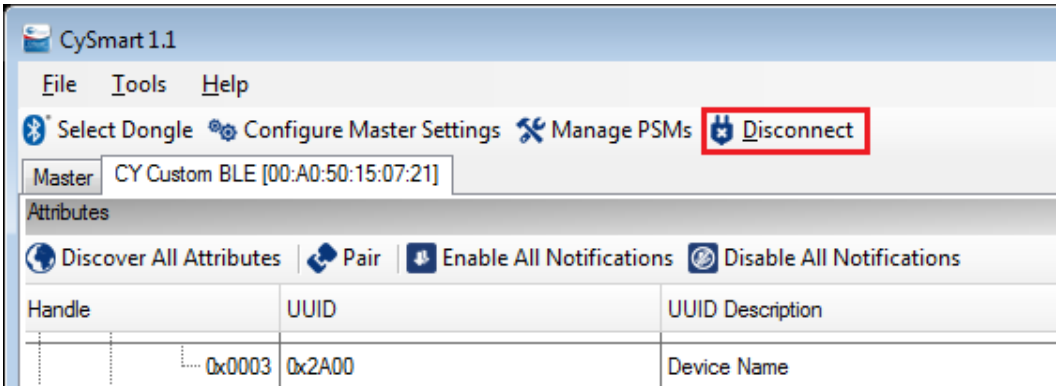
8. 図 50 に示すように、BLE Pioneer Kit の RGB LED の新しい色および輝度を観察します。

図 50. BLE Pioneer Kit での RGB LED 制御



9. 図 51 に示すように、デバイスを切断するために、**Disconnect** をクリックします。

図 51. デバイスを切断



6 まとめ

本アプリケーション ノートは、BLE コンポーネントを使用して PSoc 4 BLE プロジェクトにカスタム BLE サービスを追加し、サービスを設定し、BLE GATT クライアント デバイスとのデータ読み出し／書き込みを実行する手順を説明しました。本書で説明した方法は、開発される PSoc 4 BLE プロジェクト内の BLE カスタム サービスのいかなる種類といかなる数にも簡単に拡張できます。

7 関連情報

- AN91267 - Getting Started with PSoC 4 BLE
- AN91184 - PSoC 4 BLE Designing BLE Applications
- CY8CKIT-042-BLE Pioneer Kit
- BLE 開発者ポータル
- iOS 向け CySmart アプリ
- Android 向け CySmart アプリ

著者について

氏名: Rohit Kumar

役職: シニア アプリケーション エンジニア

A 付録

A.1 通知の送信

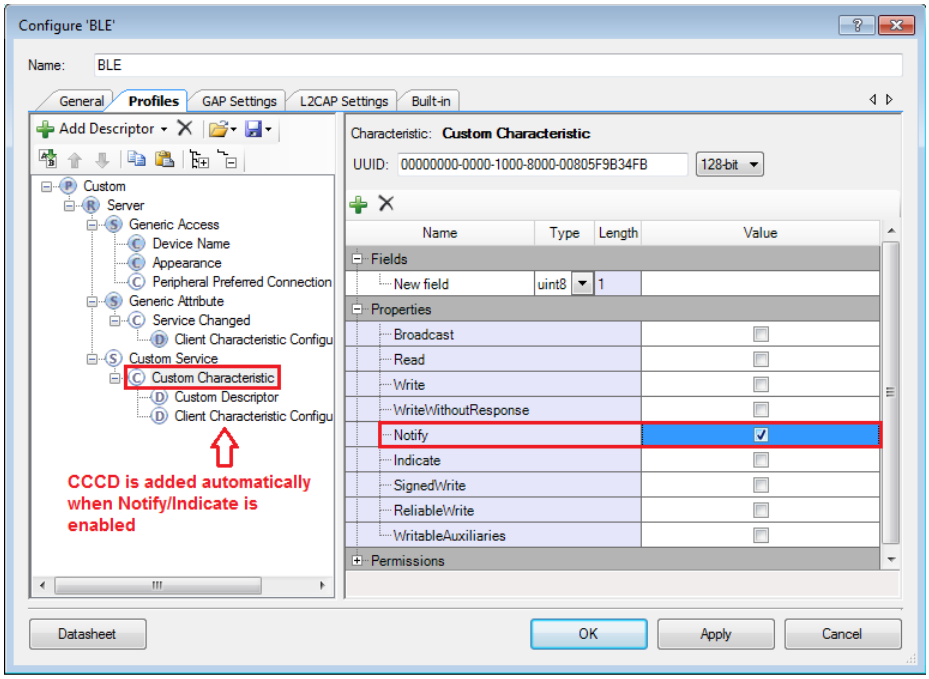
キャラクタリスティックからの書き込みおよび読み出しに加えて、一般に要求される他の重要なアクセスは通知です。通知を使用すると、GATT サーバーは GATT クライアントが継続してポーリングする必要なく、GATT クライアントに新しいデータを送信できます。

通知をサポートする各キャラクタリスティックには、Client Characteristic Configuration Descriptor (CCCD) という関連ディスクリプタがあります。この CCCD に書き込むことで、GATT クライアントは GATT サーバーでの通知を有効または無効にします。GATT クライアントが GATT サーバーでの通知を有効にするまで、GATT サーバーは通知を介してデータを送信することができません。

カスタムキャラクタリスティックへの通知アクセスを許可し、GATT クライアント デバイスにデータを送信するために、これらの手順に従ってください。指示をサポートする場合も同様の手順を適用できます。

1. BLE コンポーネントのコンフィギュレーション ウィンドウで、通知が有効にされるキャラクタリスティックを選択します。図 52 に示すように、**Notify** チェック ボックスを選択します。Client Characteristic Configuration Descriptor は自動的に属性のリスト (キャラクタリスティックの下) に追加されます。**OK** をクリックします。

図 52. コンポーネントでの通知アクセスを選択



2. GATT サーバーでの通知を有効にするために、GATT クライアントは **0x0001** 値を CCCD に書き込みます。「CYBLE_EVT_GATTS_WRITE_REQ」イベントが発生すると、以下を行ってください。
   a. CCCD の属性ハンドルの書き込み要求であるかをチェックします。
   b. そうである場合、送信した値には最下位 2 ビットのどちらか 1 ビットだけがセットされ、他のビットはすべてセットされていないかをチェックします。これらのビットは CCCD での書き込み要求の一部として送信できる唯一の値です。
   c. そうである場合、CCCD 値を GATT サーバーにを登録します。
   d. CCCD の書き込みが正常に完了したかどうかによって、書き込み応答またはエラー応答をクライアントに送信します。

```
case CYBLE_EVT_GATTS_WRITE_REQ:

wrReqParam = (CYBLE_GATTS_WRITE_REQ_PARAM_T *) eventParam;

/* Check if the returned handle is matching to CCCD attribute */
if(CYBLE_CUSTOM_CLIENT_CHARACTERISTIC_CONFIGURATION_DESC_HANDLE ==
                                            wrReqParam->handleValPair.attrHandle)
{
       /* Only the first and second lowest significant bit can be
       * set when writing on CCCD. If any other bit is set, then
       * send error code */
       if(FALSE ==
              (wrReqParam->handleValPair.value.val
       [CYBLE_CUSTOM_CLIENT_CHARACTERISTIC_CONFIGURATION_DESC_INDEX] &
       (~CCCD_VALID_BIT_MASK)))
       {
              /* Set flag for application to know status of notifications.
              * Only one byte is read as it contains the set value. */
              startNotification =
                            wrReqParam->handleValPair.value.val
              [CYBLE_CUSTOM_CLIENT_CHARACTERISTIC_CONFIGURATION_DESC_INDEX];

              /* Update GATT DB with latest CCCD value */
              CyBle_GattsWriteAttributeValue(&wrReqParam->handleValPair,
                                                   FALSE,
                                                   &cyBle_connHandle,
                                                   CYBLE_GATT_DB_PEER_INITIATED);
       }
       else
       {
              /* Send error response for Invalid PDU against Write
              * request */
              CYBLE_GATTS_ERR_PARAM_T err_param;

              err_param.opcode = CYBLE_GATT_WRITE_REQ;
              err_param.attrHandle = wrReqParam->handleValPair.attrHandle;
              err_param.errorCode = ERR_INVALID_PDU;

              /* Send Error Response */
              (void)CyBle_GattsErrorRsp(cyBle_connHandle, &err_param);

              /* Return to main loop */
              return;
       }
}

/* Send response to the Write request */
CyBle_GattsWriteRsp(connectionHandle);
break;
```

「BLE Core specification, Vol 3, Part F, section 3.4.1」によると、エラー コード「ERR_INVALID_PDU」の値は 0x04 です。

アプリケーション コードに以下のように定義します。

```
/*****************GATT Error code******************/
#define ERR_INVALID_PDU                                0x04
#define CCCD_VALID_BIT_MASK                            0x03
#define NOTIFY_BIT_MASK                                0x01
```

3. メイン アプリケーションでは、データが使用可能になり GATT クライアントから通知が有効になっているときはいつでも、通知によりデータを送信できます。

```
/* 'notificationHandle' is handle to store notification data parameters */
CYBLE_GATTS_HANDLE_VALUE_NTF_T      notificationHandle;

/* Check if the notification bit is set or not */
if(startNotification & NOTIFY_BIT_MASK)
{
        /* Update Notification handle with new data*/
        notificationHandle.attrHandle = CYBLE_CUSTOM_CHAR_HANDLE;
        notificationHandle.value.val = &data;
        notificationHandle.value.len = dataLength;

        /* Report data to BLE component for sending data by notifications*/
        CyBle_GattsNotification(connectionHandle, &notificationHandle);
}
```

4. カスタム プロファイルで BLE 通知を実装するサンプル プロジェクトは、CY8CKIT-042-BLE Pioneer Kit の CapSense_Proximity または CapSense_Slider_and_LED のプロジェクトを参照してください。

変更履歴

文書名: AN91162 – BLE カスタム プロファイルの作成について

文書番号: 002-00013

版	ECN	発行日	変更内容
**	4984943	11/02/2015	これは英語版 001-91162 Rev. *A を翻訳した日本語版 002-00013 Rev. **です。
*A	5711741	04/26/2017	ロゴと著作権を更新しました。
*B	6655980	01/08/2020	これは英語版 001-91162 Rev. *D を翻訳した日本語版 002-00013 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。