PSoC 6 MCU - 生成コードの IDE へのインポート

本書について

適用範囲と目的

本アプリケーションノート (AN219434) は、PSoC™ 6 MCU アーキテクチャ用に PSoC™ Creator™によって生成されたコードを、所望の統合開発環境にインポートする方法を説明します。この知識があれば、自動生成されたコードの利点を望む IDE と組み合わせることができます。高レベルのオプションおよびそれぞれのオプションの詳細な手順を説明します。

関連製品ファミリ

PSoC™ 6 MCU

関連サンプル コード

CE212736

関連アプリケーションノート

AN210781

その他のサンプル コードが必要な場合は、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ご覧ください。ビデオライブラリについては、ここからご覧ください。

目次

本書について . . . 1
目次 . . . 2
1 はじめに . . . 4
2 生成コードの統合 . . . 5
2.1 生成コードの利点 . . . 7
2.2 生成コードの欠点 . . . 7
2.3 ツールの互換性 . . . 8
3 生成コードのエクスポートおよびインポート . . . 9
3.1 ターゲット IDE におけるエクスポートの有効化 . . . 10
3.2 エクスポート パッケージの生成 . . . 11
3.3 パッケージのインポート . . . 11
3.4 反復的開発のサポート . . . 12
3.5 エクスポート/インポートのレビュー . . . 12
4 生成コードの手動によるインポート . . . 13
4.1 プロジェクト ファイルの作成 . . . 13
4.2 ソース ファイルの特定 . . . 13
4.2.1 ユーザー ファイル . . . 14
4.2.2 デザイン ファイル . . . 16
4.2.3 ライブラリ ファイル . . . 17
4.3 プロジェクトへのファイルの追加 . . . 18
4.3.1 PDL ユーザー ファイルの場所 . . . 19
4.3.2 PDL ライブラリ ファイルの場所 . . . 19
4.4 反復的開発のサポート . . . 19
4.5 手動によるインポートのレビュー . . . 20
5 生成コードの手動によるインポートの 1 例 . . . 21
5.1 設計の完了およびアプリケーションファイルの生成 . . . 21
5.2 IDE プロジェクト ファイルのセットアップ . . . 22
5.3 プロジェクトへのファイルの追加 . . . 24
5.3.1 ファームウェア ファイルの追加 . . . 25
5.3.2 ユーザー ファイルの追加 . . . 25
5.3.3 デザイン ファイルの追加 . . . 26
5.3.4 PDL ライブラリ ファイルの追加 . . . 27
5.3.5 Bluetooth® LE ライブラリ ファイルの追加 . . . 29
5.4 インクルード パスの設定 . . . 31
5.4.1 ユーザー ファイルへのパスの設定 . . . 31
5.4.2 デザイン ファイルへのパスの設定 . . . 32
5.4.3 peripheral フォルダ (ドライバー ファイル) へのパスの設定 . . . 33

目次

5.4.4 middleware フォルダ (Bluetooth® LE スタック) へのパスの設定 34
5.4.5 その他の必要なパスの設定 35
5.5 IDE でのプロジェクトのビルド 36
5.6 例のレビュー 37
6 まとめ 38
関連資料 39
A 付録 A -IDE プロジェクト ファイルの設定 40
B 付録 B -学習目的での生成コードの使用 41
改訂履歴 42
免責事項 43

1 はじめに

非常に多くのお客様が、ハードウェア設計プラットフォームとして PSoC™ Creator ツールを使用しますが、異なる環境でアプリケーション ソフトウェアを書きます。理由はいくつかあります。お客様の組織が、製品を開発するツール一式を確立したとか、別の IDE のビルド済みのレガシー コードを再利用したいとか、望ましい開発環境があります。本アプリケーションノートは、そのような IDE で PSoC™ 6 MCU 用に PSoC™ Creator によって生成されたコードを使用する方法を説明します。

PSoC™ 6 MCU は、Arm® Cortex®-M4 (CM4) と Cortex-M0+ (CM0+) のデュアル CPU プログラマブル システムオンチップをベースとしています。

PSoC™ 6 MCU デバイスのファームウェア開発のために、2 つの異なる開発環境(PSoC™ Creator および ModusToolbox™ v2.x ソフトウェアベース)を提供しています。表 1 に各ワークフローの主要な機能を示します。ModusToolbox™ v2.x で利用できる機能の概要とその仕組みについては、ModusToolbox™ユーザーガイドの ModusToolbox™ Software Overview を参照してください。

表 1 PSoC™ Creator と ModusToolbox™ソフトウェアの比較

機能	PSoC™ Creator	Eclipse IDE for ModusToolbox™
IDE フレームワーク	独自仕様、拡張不可	Eclipse ベース、拡張可能
ドライバ ライブラリ	ペリフェラルドライバーライブラリ(PDL)	psoc6pdl, GitHub 上の psoc6hal ライブラリ
コード生成	コンポーネント (Bluetooth® LE および CAPSENSE™を含む)	コンフィギュレーター: デバイス, CAPSENSE™, Bluetooth®, QSPI, SmartIO, USBDev
ホスト OS	Windows	Windows, macOS, Linux
ミドルウェアソフト	Bluetooth® LE, dfu, em_eeprom, emWin, usb_dev, FreeRTOS	複数のエコシステム用の広範なライブラリのコレクション。GitHub の PSoC™ 6 Middleware を参照してください。
後処理ツール	cymcuelftool (Windows, macOS, Linux)	ほとんどのユースケースでは必要ありません: cymcuelftool は引き続きサポートされます。

このアプリケーションノートでは、PSoC™ Creator 環境とワークフローについて説明します。AN225588 Using ModusToolbox™ Software with a Third-Party IDE は環境とワークフローについて説明します。

ペリフェラル ドライバー ライブラリ (PDL) は PSoC™ 6 MCU デバイス用のソフトウェア開発キット (SDK) です。ファームウェアは PDL API 関数の呼び出しを使用してペリフェラル ドライバーを設定、初期化、有効化、および使用します。PDL はまた、Bluetooth® Low Energy (LE) やリアルタイム オペレーティングシステム (RTOS) などのミドルウェアにも対応しています。設計的には、PDL は IDE に依存しません。

しかし、PSoC™ Creator は PDL を完全に組み込んでいます。設計環境として、PSoC™ Creator は使いやすい UI を使用してクロック、割込み、ピン割当ておよびドライバーの設定を手助けします。ドライバーをカスタマイズするために、ペリフェラルに対応する PSoC™ Creator 設計にコンポーネントを追加します。コンフィギュレーション コードを書く代わりに、コンポーネントのオプションを修正します。その後、PSoC™ Creator はすべてのコンフィギュレーション コードを生成して、設計に基づいてクロック、割込み、ピンおよびカスタム ドライバーをセットアップします。PSoC™ 6 MCU デバイスを目標とする時、PSoC™ Creator は PDL を使用してコードを生成します。PSoC™ Creator を使用せずに PDL のみを使用する場合、設計用のすべてのコンフィギュレーション コードを書き込みます。

生成コードは、設計用のファームウェア セットアップに関与する複雑さを大いに減少させる価値の高いリソースです。本アプリケーションノートは、どのように生成コードをすべての IDE に使用できるかを説明します。このための方法は一つではありません。本書は、利用可能なオプションを詳しく説明します。また、一例により使用方法を示します。本書を終わりまで読んだ後、利用可能なオプションを確定でき、望ましい IDE に生成コードを統合する方法が分かります。

生成コードの直接使用を妨げる状況であっても、本書は生成コードを使用して PDL を理解することを示します。

PDL または PSoC™ Creator の詳細情報については、関連資料を参照して利用可能なリソースへのリンクをご覧ください。

2 生成コードの統合

高レベルでは、IDE のプロジェクト ファイルに生成コードを統合する方法は 2 つあります:

- PSoC™ Creator からコードをエクスポートした後、IDE にそのコードをインポート
- IDE のプロジェクトに生成ソース ファイルを手動で追加

図 1 のフロー図は、それぞれのパスで行うタスクを示します。本アプリケーションノートは、フロー図に示したワンタイム タスクについて説明します。状況によって、どちらか一方のパスが適切です。

エクスポートのパスは、いくつかの IDE のみに対応しています。このパスの詳細については、生成コードのエクスポートおよびインポートを参照してください。以下はサポートされる IDE です。

- IAR Embedded Workbench
- Keil µVision
- Eclipse ベース IDE

手動によるコード インポートのパスは、サポートされる IDE を含むすべての IDE に対応しています。サポートされる IDE では、手動のアプローチを容易にする追加のリソースが用意されています。これには、完全に設定されたプロジェクト ファイル、IDE 固有のスタートアップ コード、フラッシュ コンフィギュレーション ファイル、リンカー スクリプトが含まれます。このパスの詳細については、生成コードの手動によるインポートを参照してください。

2 生成コードの統合

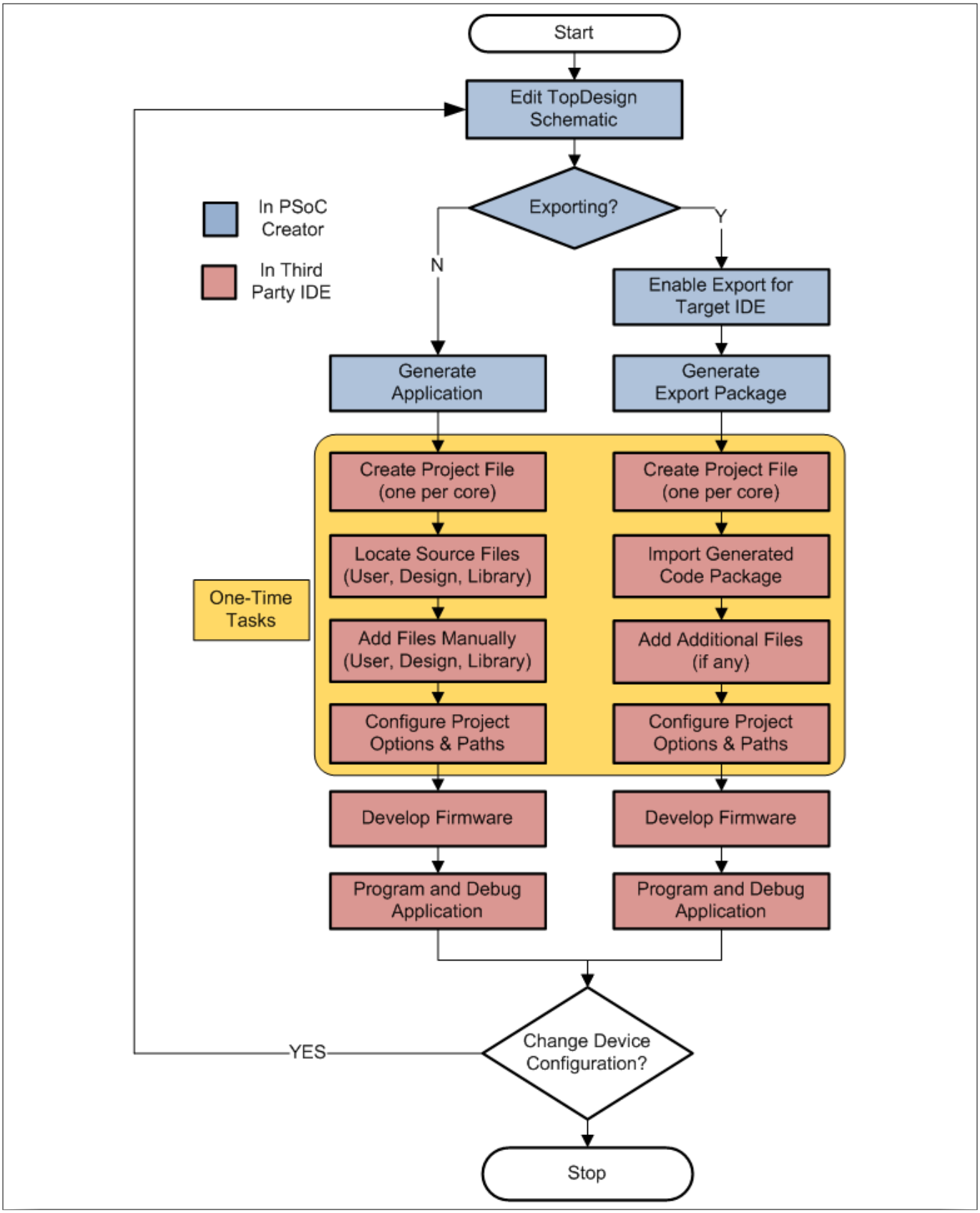


図 1　　生成コードの統合

2 生成コードの統合

2.1 生成コードの利点

システムの設計および設定に PSoC™ Creator を使用することで、生成コードから大きな利益を得られます。すべての必要なコンフィギュレーション コードはユーザーのために作成されます。このコードはかなり複雑であることがあります。例えば、`cyfitter_cfg.c` 内の生成コードはすべてのクロックを初期化し、各クロックのソースと分周器を設定し、設計ですべてのクロックを有効にします。PDL の API 関数呼び出しがそれを行うようにします。このコードは PSoC™ Creator 設計のクロック ツリー コンフィギュレーションをベースとします。

その他の良い例は Bluetooth® LE コンポーネントです。Bluetooth® LE コンポーネントを設定するコードは通常、数百行の長さになり、さまざまなコンフィギュレーション構造体を宣言し、100 項目以上の値を定義します。PSoC™ Creator は、設計内の Bluetooth® LE コンポーネント コンフィギュレーションに基づいて完全なコードを自動的に生成します。

コンフィギュレーション コードに加えて、PSoC™ Creator は各コンポーネント用に簡素化した関数 API を生成します。コンポーネント API は PDL API をベースとし、使用します。すなわち、PDL 関数を直接呼び出す代わりにコンポーネント API を使用できます。これらの API は一貫性と互換性があるので、組み合わせることが可能です。

例えば、PDL API を使用して PWM を初期化、有効化、および起動するために、コードは以下のようになります (エラーが無視される)。

```
Cy_TCPMW_PWM_Init (TCPWM1, counterNumber, &PWM_config);
Cy_TCPWM_Enable_Multiple (TCPWM1, whichCounters);
Cy_TCPWM_TriggerStart (TCPWM1, whichCounters);
```

一方、コンポーネント API を使用する場合、コードは以下のようになります (MyPWM と名付けられた TCPWM コンポーネントの例)。

```
MyPWM_Start ();
```

コード ジェネレータは PWM を有効にするために関数呼び出しの適切なシーケンスを実装します。また、PSoC™ Creator で設計および設定に基づいて、ハードウェア関連のすべてのパラメーターを提供します。`MyPWM_Start()` 関数の生成コードを調べると、3 つの PDL API 呼び出しが実装されており、正しい順序で呼び出されます。

生成コードは主に C ソース ファイルとヘッダ ファイルにありますが、アセンブラ ファイルとコンパイル済みバイナリが含まれる場合もあります。本アプリケーションノートで説明するようにこれらの生成ファイルを IDE プロジェクトに正しく追加したら、反復的開発が完全にサポートされます。PSoC™ Creator で設計を修正し、アプリケーションを再生成すると、更なる作業の必要無しで IDE はそれらのファイルを認識します。

2.2 生成コードの欠点

PSoC™ Creator は C99 規格と互換性のあるコードを生成します。ご使用のコンパイラはその規格をサポートしなければなりません。さらに、コンパイルは微妙な実装詳細によって異なります。特定のコンパイラがいくつかの構文を非標準的な方法で取り扱う可能性がありますが、これは普通ではありません。

最も大きな欠点は、ほとんどの場合に生成コードを修正できないことです。生成ソース ファイルでコードを修正した場合、コードを再生成すると変更が失われます。生成コードに対して行った変更は、変更前の PSoC™ Creator 設計に反映されません。開発サイクル中、設計の修正が必要となったら、PSoC™ Creator で変更を行ってからアプリケーションを再生成します。

ただし、PSoC™ Creator によって生成された特定のファイルはユーザー ファイルとして取り扱われ、変更が失われずに修正できます。ユーザー ファイルを参照してください。

2.3 ツールの互換性

PSoC™ Creator および PSoC™ Programmer は、Windows OS 用に構築された独自のツールです。他のプロバイダーのツールを使用する場合、互換性の問題が発生する場合があります。例えば、PSoC™ Creator は独自の情報を最終的に hex ファイルに追加します。PSoC™ Programmer には、この情報が必要です。他の IDE はこの情報を生成しないため、PSoC™ Programmer ではそのような hex ファイルは使用できません。

ModusToolbox™ソフトウェアおよび新しいインフィニオン製の Programmer には、これらの問題はありません。

3 生成コードのエクスポートおよびインポート

エクスポート/インポートのパスはいくつかの IDE のみに対応しています。サポートされる IDE に対して、PSoC™ Creator はエクスポート パッケージを提供します。表 2 は、エクスポート パッケージの一覧およびそれぞれがサポートする IDE を示します。

表 2 利用可能なエクスポート パッケージ

パッケージ	サポートされる IDE	主要なファイル	パス
CMSIS パック	Keil µVision v5 以降 IAR Embedded Workbench v8.1 以降 CMSIS パックをインポート可能な Eclipse ベース IDE	`.pack` ファイル	`Export/Pack`
プロジェクト間接続ファイル (IPCF)	IAR Embedded Workbench	`.ipcf` ファイル (CPU ごとに 1 ファイル)	`Export`
メイクファイル	GNU コマンド ライン ツール	`makefile`	`<project>.cydsn`

エクスポート/インポートのプロセスはすべてのサポートされる IDE で共通です。設計開発後、以下の手順を行ってください。

- PSoC™ Creator
  - エクスポート パッケージの作成を可能にしてください。
  - コードをビルド (すると、エクスポート パッケージが生成される) してください。
- サードパーティの IDE
  - 空のプロジェクトを作成 (マルチ CPU デバイスの場合は CPU ごとに 1 プロジェクト) してください。
  - 空のプロジェクトにパッケージをインポートしてください。
  - 必要に応じて、エクスポート ファイルに含まれないファイルを追加してください。
  - プロジェクトのオプションを設定してください。

エクスポート/インポート プロセスの詳細は、サポートされる IDE によって大幅に異なります。ここでは、プロセスがどのように行われるかを説明します。詳細な手順は PSoC™ Creator Help および PSoC™ Creator User Guide に記述されています。本アプリケーションノートでは、重複した情報を記載しません。IDE にパッケージを正常にエクスポートおよびインポートするために、その資料を参照する必要があります。

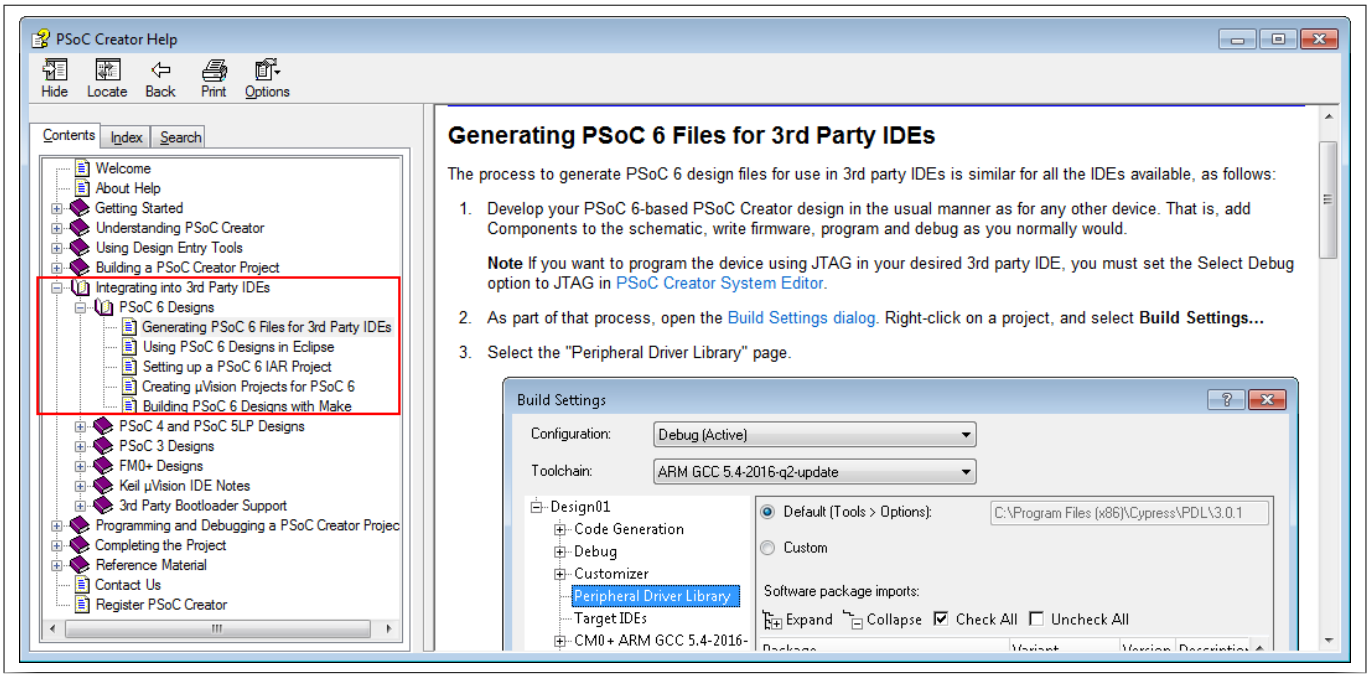


図 2　　サードパーティ IDE への統合に関する PSoC™ Creator Help トピック

3.1　　ターゲット IDE におけるエクスポートの有効化

PSoC™ Creator で、**Project > Build Settings** メニュー コマンドを使用してプロジェクト設定を開きます。次に、**Target IDEs** パネルを開きます。作成したいエクスポート パッケージに対して **Generate** を選択します。図 3 に、有効にされた CMSIS Pack オプションを示します。CMSIS Pack は、ソフトウェア コンポーネント向けの、IDE に依存しないデリバリー パッケージです。

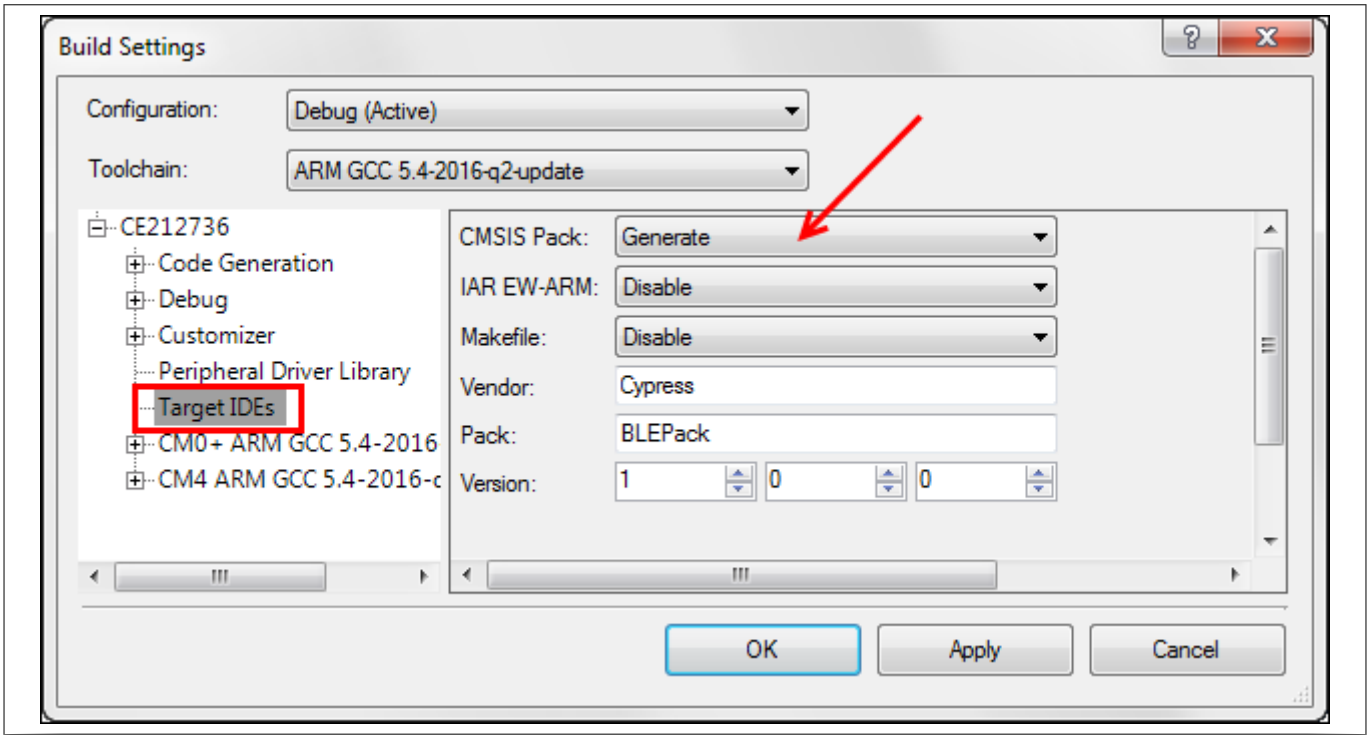


図 3　　CMSIS Pack エクスポート パッケージを有効にする

エクスポートのばらつきの例として、図に示した **Vendor**、**Pack** および **Version** の情報は CMSIS Pack にのみ必要なものです。IAR EW-ARM には、異なるオプションが必要です。

詳細は、PSoC™ Creator の資料を参照してください。

3.2 エクスポート パッケージの生成

パッケージの作成を有効にした後、コードをビルドします。PSoC™ Creator で、**Build > Build <project>**コマンドを使用します。**Build > Generate Application** コマンドはエクスポート パッケージを生成しないことに注意してください。

`Export` フォルダの内容はパッケージによって異なります。パッケージはファイルの集まりです。PSoC™ Creator は、表 2 にリストアップした場所にそれぞれのパッケージを配置します。パッケージをインポートする際、その場所に移動して必要なファイルを探します。

3.3 パッケージのインポート

インポートのステップは、IDE およびパッケージによって大きなばらつきがあります。本節は、パッケージをインポートするために実行しなければならない一般的なタスクを紹介します。詳細な手順については、PSoC™ Creator の付随資料を参照してください。図 2 に示したように、サポートされる IDE のそれぞれにヘルプ トピックがあります。

一般的には、以下のタスクを実行します。

- 空のプロジェクト ファイルを作成 (マルチ CPU デバイスの場合は CPU ごとに 1 ファイル)
- それぞれのプロジェクト ファイルに生成コードをインポート図 4 に、各パッケージが配置される場所を示します。
  - CMSIS Pack の場合、コンピュータにパックをインストールしてから、パックをインポートします。
  - IAR ツールでは、`.ipcf` ファイルを使用してプロジェクト接続を確立します。

GNU ツールの場合は、生成メイクファイルを使用します。

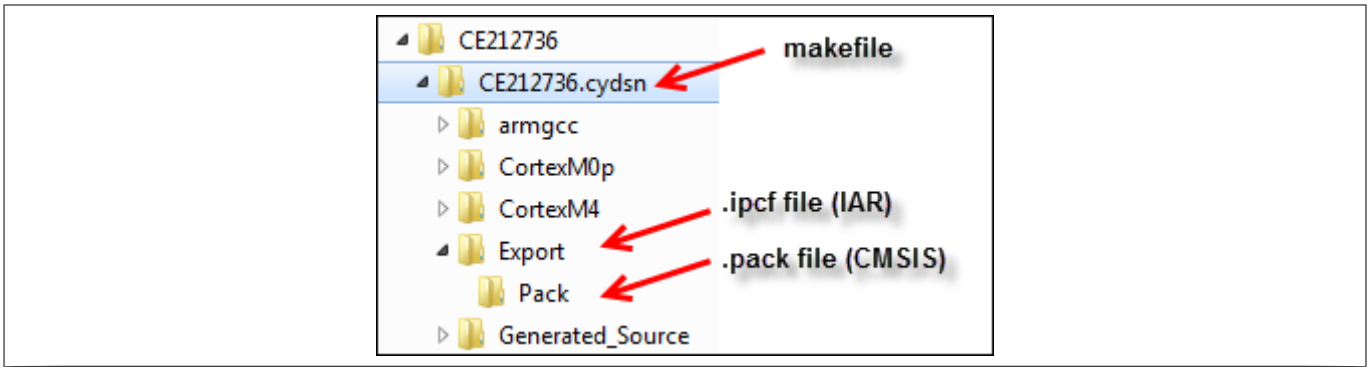


図 4 エクスポート パッケージの場所

- 必要に応じて、各プロジェクト ファイルに他のファイルを追加します。
  - CMSIS Pack では、テンプレートから CPU 固有の `main.c` ファイルおよび `linker` ファイルを作成できます。
  - IAR ツールの場合、これらのファイルは自動的にプロジェクトに含まれます。
- コンパイラ、リンカーやデバッガ設定などを含むプロジェクト オプションをセットします。

プロジェクト オプションはパッケージによって異なります。例えば IAR ツール用の IPCF 接続は、CM0+プロジェクトと衝突しないように CM4 プロジェクトの出力フォルダ名を変更する必要があります。CMSIS Pack では、ヘッダ ファイルのインクルード パスを指定し、リンカー コマンド ファイルへのパスを提供するなどです。PSoC™ Creator の付随資料はプロジェクト オプションの詳細をすべて含みます。生成されたコードをエクスポートおよびインポート

する前に、その資料に精通していることを確認してください。プロジェクト オプションのより多くの情報については、付録 A を参照してください。

プロジェクト内のリンカー ファイル、スタートアップ コード、メイクファイル、およびその他のファイルは CPU 固有または IDE 固有のものです。各 CPU および各 IDE の正しいファイルはパッケージに同梱されます。パッケージをインポートすると、正しいファイルも含まれます。

注: 多くの CPU 固有のファイルは最初に PSoC™ Creator によって生成され、ユーザーによって編集可能なファイルです。それらのファイルは修正した後、PSoC™ Creator プロジェクトを再ビルドしても変更点が失われません。詳細については、ユーザー ファイルを参照してください。

3.4 反復的開発のサポート

生成コードのインポート後、設計を変更することが必要となる可能性があります。PSoC™ Creator プロジェクトで必要に応じて変更してから、プロジェクトを再ビルドします。`Generated_Source` フォルダ内のファイルを修正しないでください。もし修正すると、コードを再生成した時にその修正が失われます。

PSoC™ Creator でコードを再ビルドすると、Export フォルダに更新されたパッケージが現れます。コード生成プロセスは、修正されたファイル、追加された新規ファイルや削除したファイルがあります。

CMSIS Pack のエクスポートのために、パックを再インストールします。`pack` ファイルを `Export` フォルダに保存し、ダブルクリックしてインストールします。解凍したダイアログには、パックがすでにインストールされたことを示すアラートがあります。それを置き換えます。IDE は追加または削除されたファイルを含む変更点を自動的に認識します。IDE でコードを再ビルドして実行ファイルを更新します。

IAR のプロジェクト間接続ファイルは PSoC™ Creator の `Generated_Source` フォルダを指します。IDE は追加または削除されたファイルを含む変更点を自動的に認識します。`.ipcf` に再度接続する必要がありません。単に IDE でコードを再ビルドして実行ファイルを更新します。

メイクファイルに対しては、ビルド プロセスが `Export` フォルダ内のファイルを使用することを確認してください。そうすると、シンプルなクリーンとビルドが新しいコードで実行ファイルを更新します。

3.5 エクスポート/インポートのレビュー

PSoC™ Creator の付随資料では、すべての詳細を説明します。インポート プロセスは IDE の新規で空のプロジェクトから始まります。

CMSIS Pack では、IDE のプロジェクト セットアップは複数のステップを含みます。パックがさまざまな IDE にインポートされる可能性があり、ビルド オプションの取り扱い方が IDE によって異なるため、いくつかのオプションをセットする必要があります。これはワンタイム タスクです。生成コードのすべての変更 (PSoC™ Creator での設計変更から) は透過的に対応されます。パックを再インストールするだけで十分です。

プロジェクト間接続は IAR ツール独自のプロトコルです。1 つの IDE 向けに設計されたため、プロジェクト接続はほとんどのオプションを自動でセットします。これは非常にユーザーフレンドリーなプロセスです。

4 生成コードの手動によるインポート

手動によるインポートは、エクスポート/インポート プロセスを持っているサポート IDE を含む、すべての IDE に対応しています。非サポート IDE に対しては、手動によるインポートは唯一のオプションです。

インポートを始める前に、生成コートを入手しなければなりません。PSoC™ Creator で、**Build > Generate** Application コマンドを使用するだけで十分ですコードをコンパイルする必要がありません。生成コードを手動でインポートするために、以下の手順を行ってください。

- IDE でプロジェクトを作成 (マルチ CPU デバイスの場合は CPU ごとに 1 プロジェクト) してください。
- 必要な生成ファイルを特定してください。
- プロジェクトにファイルを追加してください。

ここでは、PSoC™ Creator が何のファイルを生成するか、どこに保存するか、およびプロジェクトにどのファイルを追加する必要があるかを理解するための基礎知識を提供します。生成コードの手動によるインポートの 1 例は、プロセスを詳しく説明します。

4.1 プロジェクト ファイルの作成

本説明は、ユーザーが好む IDE に精通していること、プロジェクト ファイルの作成および設定方法を知っていること、およびマルチ CPU PSoC™ 6 MCU デバイスの各 CPU に対応するよう設定されたプロジェクトを持っていることを前提とします。設定する必要があるオプションの種類については、付録 A を参照してください。

新規で空のプロジェクト ファイルで始められます。ただし、インフィニオンはそれぞれのサポート IDE および PSoC™ 6 MCU CPU に対して事前設定されたテンプレート プロジェクトを容易しています。µVision IDE、IAR IDE、およびその他のサポート IDE 向けの CM4 と CM0+プロジェクトがあります。PDL テンプレート プロジェクトは、PDL v3.1 ユーザー ガイドの「Using PDL Template Projects」節で十分に説明されています。テンプレート プロジェクトは次のパスにあります: `<PDL Install Folder>/devices/psoc6/templates`。

テンプレート プロジェクトのビルド オプションのほとんどは正しくセットされています。生成コードのインポートの際に必要となる変更は本アプリケーションノートで説明されます。また、テンプレート内のインクルード パスはプロジェクトに関連します。テンプレート プロジェクト ファイルを異なる場所に移動したら、パスを修正しなければなりません。PDL ライブラリ ファイルの場所を参照してください。

4.2 ソース ファイルの特定

PSoC™ Creator は 3 種類のソース ファイルを生成します。

- ユーザー ファイル: 編集可能なファイル
- デザイン ファイル: PSoC™ Creator 設計およびコンポーネント固有のファイル
- ライブラリ ファイル: PDL からのファイル

以下の節で、それぞれのソース ファイルの種類を詳しく説明します。

PSoC™ Creator は、これらのファイルを`<project>.cydsn` フォルダ内のさまざまなフォルダに保存します。本アプリケーションノートでは、この場所を単に cydsn フォルダと呼びます。図 5 に、生成ソース ファイルのフォルダ ツリーおよび場所を示します。

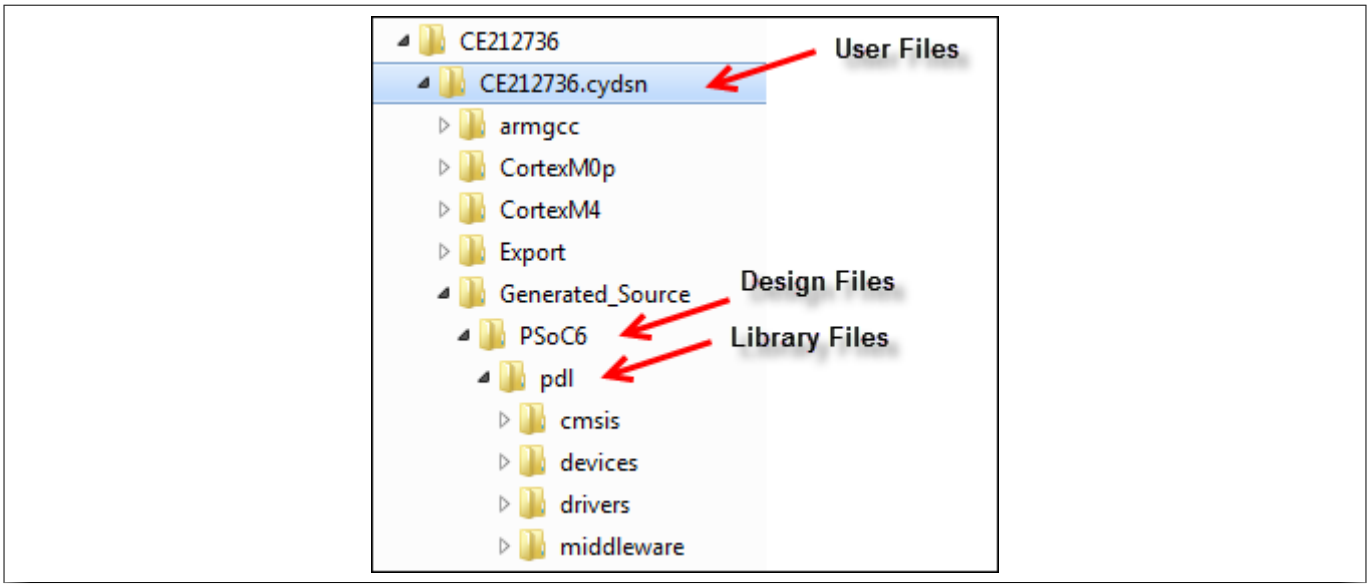


図 5　　生成ファイルの場所

4.2.1　　ユーザー ファイル

ユーザー ファイルは、ユーザーが設計の要件に応じて変更できるファイルです。PSoC™ Creator は一旦このファイル (機能的には空である場合がある) を生成したら、変更しません。ユーザー ファイルを理解するためのいくつかの主なコンセプトがあります:

- ユーザー ファイルは、ユーザーが初めて PSoC™ Creator プロジェクトを作成し、アプリケーションを生成した時に自動で作成されます。
- PSoC™ Creator は、一旦作成されたユーザー ファイルに対して変更や置き換えを行いません。したがって、ユーザーは設計の必要に応じてユーザー ファイルを修正することができます。
- すべてのユーザー ファイルはローカルのコピーであるため、それらへの変更は他のプロジェクトやインストールされた PDL ライブラリには影響をしません。
- マルチ CPU デバイスに対しては、IDE での正しいプロジェクトに CPU 固有のファイルを追加し、それぞれの CPU のプロジェクトに共有ファイルを追加します。

すべてのファイルが必要であるわけではありません。ファイルの一部は特定のツール チェーンに特有です。

表 3 に、共通のユーザー ファイルをリストアップします。ユーザー ファイル (この表に記載したように、スタートアップ コードを除く) は cydsn フォルダのトップ レベルです。デバイスによって変わるファイル名があります。この表では、psoc63 シリーズを例として使用します。ファイルの一部は CPU 固有のものです。また、スタートアップ コードおよびリンカーのファイルは IDE に特有です。ユーザーのプロジェクトに適切なファイルを使用し、その他を無視してください。非サポート IDE を使用する場合、IDE と互換性がある同等のファイルを提供します。

設計によって、追加のユーザー ファイルがある場合があります。cydsn フォルダのトップ レベルのソース ファイルはすべてユーザー ファイルであり、ファイルが CPU 固有のものであるかどうかに応じて 1 つまたは両方のプロジェクトに追加する必要があります。

表 3　　PSoC™ Creator が生成するユーザー ファイル

ファイル	アクション	ソース	使用法	備考
`main_cm0p.c main_cm4.c`	プロジェクトに追加する [1)]、またはユーザー独自のものを提供	生成される	必須	PSoC™ Creator はユーザー ファイルを cydsn フォルダのトップ レベルに保存

(続く)

表 3 (続き) PSoC™ Creator が生成するユーザー ファイル

ファイル	アクション	ソース	使用法	備考
cyapicallbacks.h	ヘッダへのパスを提供	生成される	必須	ユーザーがマクロ コールバックを実行するための空のファイル (PSoC™ Creator のヘルプ トピック「Writing Code」を参照してください)
system_psoc6.h	ヘッダへのパスを提供	PDL からコピーされる	必須	ユーザー定義可能なクロック コンフィギュレーション マクロを含む
system_psoc6_cm4.c system_psoc6_cm0plus.c	プロジェクトに追加。	PDL からコピーされる	必須	CPU に依存。
startup_psoc6_01_cm4.s startup_psoc6_01_cm0plus.s (IAR)	プロジェクトに追加。非サポート IDE では、スタートアップ ファイルを提供	PDL からコピーされる	IDE にファイルを使用	シリーズ、CPU、および IDE に依存。スタートアップ コードのパス: .cydsn/<IDE>/startup
startup_psoc6_01_cm4.s startup_psoc6_01_cm0plus.s (MDK)				
startup_psoc6_01_cm4.S startup_psoc6_01_cm0plus.S (GCC)				
cy8c6xx7_cm4_dual.icf cy8c6xx7_cm0plus.icf	CPU 固有のリンカー ファイルを使用するためのプロジェクト オプションをセット。非サポート IDE では、リンカー ファイルを提供	PDL からコピーされる	IDE にファイルを使用	シリーズ、CPU、および IDE に依存。リンカー コマンド ファイル
cy8c6xx7_cm4_dual.scat cy8c6xx7_cm0plus.scat				
cy8c6xx7_cm4_dual.ld cy8c6xx7_cm0plus.ld				

1) CPU 固有のファイルを対応する CPU のプロジェクトに追加します。

注: *表 3 にリストアップしたすべてのヘッダ ファイルは同じフォルダにあるので、単一のパスですべてのファイルへのアクセスを提供できます。*

注: *デフォルトの cyapicallbacks.h ファイルが空ですが、それをプロジェクトから削除する簡単な方法はありません。それは、他の自動生成されたヘッダ ファイルに含まれます。このヘッダを含む#include ステートメントを削除しても、次回コードを生成した時にそのステートメントが再び現れます。*

注: *PDL テンプル プロジェクトは、cydsn フォルダ内の生成ファイル (コピー) ではなく、インストールされた PDL の場所からのユーザー ファイルとパスを持ちます。コードを手動でインポートする時、ファイルを cydsn フォルダ内のローカル コピーに置き換えます。PDL ユーザー ファイルの場所を参照してください。*

いくつかのファイルは CPU に特有なものであり、それ以外はマルチ CPU デバイスでそれぞれの CPU によって使用されます。system_psoc6.h ファイルが例として挙げられます。各 CPU は同じファイルを使用してユーザー定義可能なクロック構成マクロに基づいて CPU クロックをセットアップします。IDE に共有ファイルを手動でインポートする時、このファイルをそれぞれの CPU のプロジェクトに追加しなければなりません。不確かな場合は、PSoC™

Creator の Workspace Explorer ペインでファイルをクリックしてそのプロパティを確認することで、どの CPU やプロセッサ、ツールチェーンがこのファイルを使用するかが分かります。

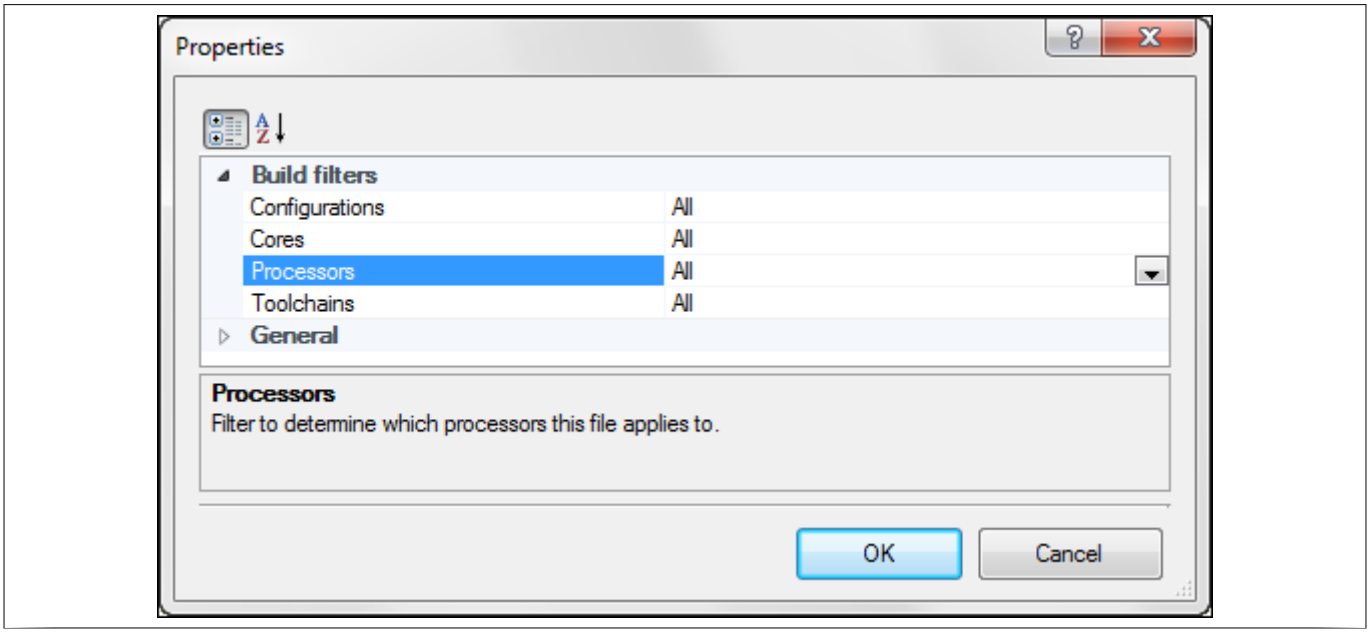


図 6　　system_psoc6.h ファイルのプロパティ

4.2.2　デザイン ファイル

PSoC™ Creator は、設計に特有ないくつかのファイルを自動で生成します。場合によって、これらのファイルはサポート IDE にも特有であることがあります。すべてのデザイン ファイルは次のフォルダにあります。 `cydsn/Generated_Source/PSoC6`

デザイン ファイルは、PSoC™ Creator Help の `Building a Creator Project > Generated Files (PSoC 6)`トピックで説明されています。デザイン ファイルは、設計の各コンポーネント用のソースおよびヘッダ ファイルを含みます。汎用デジタル ブロック (UDB) ベースのコンポーネントの場合、コンポーネントの設定および使用に必要なすべてのファイルはデザイン ファイルに含まれます。

デザイン ファイルを理解するためのいくつかのキーコンセプトがあります。

- アプリケーションを生成した時、いつでも PSoC™ Creator はデザイン ファイルを変更する可能性があります。
- デザイン ファイルを修正しないでください。行った変更は、アプリケーションを生成した時に失われます。
- コンポーネント固有のファイルは<Component Name>.h および<Component Name>.c と名付けられます。
- デザイン ファイルの一部は任意であるか、または IDE に特有のものです (表 4 を参照してください)。

ほとんどのデザイン ファイルは共有ファイルです。マルチ CPU デバイスに対しては、それぞれの CPU のプロジェクトに共有ファイルを追加します。いかなる CPU 固有のファイルも、ファイル名で CPU を識別します。

表 4 に、デザイン ファイルおよびプロジェクトでのファイルの使用に必要なアクションをリストアップします。

表 4　　PSoC™ Creator が生成するデザイン ファイル

ファイル	アクション	使用法	備考
`project.h`	ヘッダへのパスを提供 [1)]	必須	この表にリストアップされたすべてのヘッダを含む
`<Component Name>.c/h`	プロジェクトにソース ファイルを追加。ヘッダへのパスを提供	必須	コンポーネント固有 API のコード `project.h` に含まれるヘッダ ファイル

(続く)

表 4 (続き) PSoC™ Creator が生成するデザイン ファイル

ファイル	アクション	使用法	備考
cyfitter.h	ヘッダへのパスを提供	必須	設計固有のアドレスと値の#define
cyfitter_cfg.c/h	プロジェクトにソース ファイルを追加。ヘッダへのパスを提供	必須	main() に至る前にデバイスを設定するコード
cyfitter_gpio.h	ヘッダへのパスを提供	必須	ピン コンフィギュレーション用の設計固有のアドレスと値の#define
cyfitter_sysint.h	ヘッダへのパスを提供	必須	設計固有の割込みの#define
cyfitter_sysint_cfg.c/h	プロジェクトにソース ファイルを追加。ヘッダへのパスを提供	必須	システム割込みを設定するコード
cymetadata.c	プロジェクトに追加	必須	含む必要がある追加のメモリ空間をすべて定義
cydevice_trm.h	ヘッダへのパスを提供	必須	デバイスのコンフィギュレーション空間ですべてのアドレスを定義
cydisabledsheets.h	ヘッダへのパスを提供	必須	無効になった回路図ページの#define。無効になった回路図ページがない場合は空
cydevicegnu_trm.inc	必要に応じてアセンブラコードに含む	任意	IDE に依存。アセンブラ用にデバイスのコンフィギュレーション空間ですべてのアドレスを定義
cydeviceiar_trm.inc			
cydevicerv_trm.inc			
cycodeshareexport.ld	必要に応じてスクリプトを使用するためにプロジェクト オプションをセット	任意	IDE に依存。異なるアプリケーションへエクスポート/からインポートするためのリンカー スクリプト。通常は空
cycodeshareimport.ld			
cycodeshareimport.scat			

1) この表にリストアップしたすべてのヘッダ ファイルは同じフォルダにあるので、単一のパスですべてのファイルへのアクセスを提供できます。

デフォルトの main_cm0p.c と main_cm4.c ファイルは project.h を含みます。project.h ファイルは、デザイン ファイルに含まれる他のすべての必要なヘッダを含みます。ユーザー独自の main.c ファイルを提供したら、必ず project.h を含めてください。

通常、cydisabledsheets.h ファイルは空ですが、それをプロジェクトから削除する簡単な方法はありません。それは project.h に含まれています。#include ステートメントを削除しても、次回コードを生成した時にそのステートメントが再び現れます。

4.2.3 ライブラリ ファイル

ライブラリ ファイルは PDL インストールからのファイルのコピーです。PSoC™ Creator は、必要なファイルを cydsn/Generated_Source/PSoC6/pdl フォルダのサブフォルダに自動でコピーします。PDL については、PDL v3.1 ユーザー ガイドを参照してください。

ライブラリ ファイルを理解するためのいくつかのキーコンセプトがあります。

- PDL のすべてではなく、必要なファイルのみがコピーされます。
- ライブラリ ファイルの一部はアセンブラ ソース ファイルまたは事前コンパイルされたバイナリです。
- PSoC™ Creator はライブラリ ファイルを修正しません。

- デザイン ファイルを修正しないでください。行った変更は、アプリケーションを生成した時に失われます。
- ほとんどのライブラリ ファイルは共有ファイルです。マルチ CPU デバイスに対しては、それぞれの CPU のプロジェクトに共有ファイルを追加します。いかなる CPU 固有のファイルも、ファイル名で CPU を識別します。

どのライブラリが pdl フォルダ内に現れるかは、設計によります。表 5 に、ツリー内の主な場所、フォルダ内のファイルの一覧、およびプロジェクトでファイルを使用する方法を示します。

表 5 PSoC™ Creator が生成するライブラリ ファイル

パス/フォルダ	アクション	備考
pdl/cmsis/include	フォルダへのパスを提供	CMSIS ヘッダ ファイル
pdl/devices/psoc6/include/ip	フォルダへのパスを提供	デバイスの IP ブロック用のヘッダ ファイル
pdl/devices/psoc6/include	フォルダへのパスを提供	デバイス固有のヘッダ ファイル
pdl/drivers/peripheral	プロジェクトにソース ファイルを追加。フォルダとすべてのサブフォルダへのパスを提供	設計で使用されるペリフェラル ドライバー用のソース ファイルとヘッダ ファイル。
pdl/middleware	プロジェクトにソース ファイルを追加。フォルダへのパスを提供	プロジェクトで使用されるミドルウェア用のソース ファイルとヘッダ ファイル。サブフォルダへのパスを追加しないでください。

注: システム ライブラリ (syslib) ペリフェラル ドライバーは、その実装の一部としてアセンブラ ファイルを持っています。これは、C ソース ファイルと共にプロジェクトに追加する必要があります。生成コードの手動によるインポートの 1 例を参照してください。

注: pdl/middleware フォルダにはサブフォルダがあることがあります。各サブフォルダへのパスを追加しないでください。PDL ソース コードは#include ステートメントでパスを提供します。

注: PDL テンプレート プロジェクトは、cydsn フォルダ内の生成ファイル (コピー) ではなく、インストールされた PDL の場所を指すパスを持っています。ファイルを手動でインポートする時、cydsn フォルダを指すようこれらのパスをリセットします。PDL ライブラリ ファイルの場所を参照してください。

4.3 プロジェクトへのファイルの追加

必要なファイルを特定した後、それらを IDE のプロジェクト ファイルに追加します。それぞれの IDE は、プロジェクトにファイルを追加する独自の方法を持ちます。例えば IDE は、ファイルをドラッグアンドドロップすることや、IDE ユーザー インターフェースを介してファイルのバッチを追加することをサポートします。

ファイルを追加する際に、以下の 3 つの原則に留意してください。

一番目は、各ファイルを PSoC™ Creator の cydsn フォルド内の場所から直接追加することです。これは、反復的開発をサポートするために重要です。PSoC™ Creator で設計を変更すると、いくつかの生成ファイルが変化します。cydsn フォルダ内のファイルで IDE プロジェクトを指すことで、ファイルへのすべての変更がプロジェクトに反映されることを保証します。

二番目は、いくつかのファイルは CPU 固有のものであることです。CPU 固有のファイルを対応するプロジェクトに追加します。マルチ CPU デバイスに対しては、両プロジェクトに共有ファイルを追加します。CPU 固有のファイルのファイル名は CPU を識別します。

三番目は、プリプロセッサが必要なヘッダ ファイルを見つけられるようインクルード パスを追加しなければならないことです。幸いなことに、ほとんどのヘッダは単一のフォルダにグループ化されています。ファイルへの容易な

アクセスのために、プロジェクトにヘッダ ファイルを追加することを好む開発者がいます。そうしても、ほとんどの IDE はプリプロセッサのためにインクルード パスを追加することを必要とします。

4.3.1 PDL ユーザー ファイルの場所

PSoC™ Creator 生成ユーザー ファイルの一部は最初に、デフォルトの PDL ファイルのコピーです。オリジナルの PDL ファイルを使用せず、cydsn フォルダ内の生成コピーを使用してください。

ユーザー ファイルはユーザーによって変更されることがあります。オリジナルの PDL ファイルを変更したら、そのファイルを使用した他のすべてのプロジェクトが影響されます。ベスト プラクティスとして、cydsn フォルダ内のコピーを使用します。ユーザー ファイルを参照してください。

PDL テンプレート プロジェクトで始まる場合、以下のことを行う必要があります。

- プロジェクトから ユーザー ファイル (PDL のインストールからのオリジナル ファイル) を削除してください。
- cydsn フォルダから同じファイルをプロジェクトに追加してください。
- /include フォルダへのプロジェクト相対パスを削除してください。IDE では通常、../../../include のようである。
- PSoC™ Creator がユーザー ファイルを保存する cydsn フォルダへのパスを追加してください。

空のプロジェクトから始まる場合、cydsn フォルダからのファイルを追加し、このフォルダへのパスをセットします。

4.3.2 PDL ライブラリ ファイルの場所

ライブラリ ファイルは PDL ファイルのコピーであり、ユーザーによって変更してはいけません。これらのファイルを cydsn/Generated_Source/PSoC6/pdl フォルダから追加します。PSoC™ Creator は、ユーザーが必要とするライブラリ ファイルのみをコピーします。

PDL テンプレート プロジェクトは、生成コードではなく、PDL インストール内のオリジナル ファイルを指す PDL ヘッダ ファイルへのプリセットされたパスを持っています。PDL テンプレート プロジェクトで始まる場合、以下のことを行う必要があります。

- PDL インストールへのプロジェクト相対パスを削除してください。
- cydsn/Generated_Source/PSoC6/pdl フォルダ内の同じ場所へのパスで置き換えてください。

プリセット パスには、/cmsis/include,/ip,/drivers/peripheral などがあります。

4.4 反復的開発のサポート

IDE でプロジェクトをビルドするために生成コードをインポートし、プロジェクト オプションを設定した後、設計を変更することが必要となります。IDE へのファイル インポートに手間がかかりますが、これはワンタイム タスクです。

反復的開発をサポートするための主要な要件は、単に cydsn フォルダからのファイルをインポートすることです。PSoC™ Creator プロジェクトで設計変更を行った後、コードをビルドします。変更に応じて、コード生成プロセスはファイルを修正、追加または削除します。

修正ファイル

PSoC™ Creator は設計ファイルのみを修正します。cydsn フォルダからの生成コードをプロジェクト ファイルに追加したら、次回のビルドはファイルの最新バージョンを使用します。IDE でコードを再ビルドして実行ファイルを更新します。更なる作業が必要ありません。

追加または削除ファイル

プロジェクトに新規ファイル (デザイン ファイルまたはライブラリ ファイル) を追加します。同様に、ファイルがもはや必要でなくなったら、プロジェクト ファイルから削除することができます。そして、IDE でコードを再ビルドして実行ファイルを更新します。

4.5 手動によるインポートのレビュー

本節は PSoC™ Creator の生成コードの重要な詳細を提供しましたが、IDE プロジェクト ファイルに生成コードを手動でインポートするプロセスは非常に簡単です。

1. 必要な生成ファイル (ユーザーファイル、デザインファイル、およびライブラリファイル) を特定してください。
2. (cydsn フォルダ内の場所から) プロジェクト ファイルにファイルを追加してください。
3. ヘッダ ファイルへのインクルード パスをセットしてください。

5 生成コードの手動によるインポートの 1 例

ここでは、比較的複雑なサンプル コードにより、生成コードのインポート プロセスを説明します。プロジェクトおよび IDE の間でばらつきがあるので、この例は正確な手順および指示ではありません。

このウォークスルーから最大限の価値を得るには、CE21736 サンプル コードをダウンロード、インストールおよびビルドする (PSoC™ Creator で) 必要があります。IDE で空のプロジェクトを新規成します。その後、コードのインポート時に PSoC™ Creator の cydsn フォルダの内容を調べます。

一例は、すべての起こり得る事態を取り上げられません。変数には、プロジェクト ファイル (テンプレートまたは空のプロジェクト) の選択、使用する IDE (多くの可能性)、ファームウェアの特質、ビルド環境の詳細などがあります。

生成コードの手動によるインポートにより、状況に応じて十分な情報を得た上で意思決定ができます。それぞれが異なる例となる幅広い選択肢を考えると、このウォークスルーは以下のオプションに基づきます。

- 空のプロジェクト ファイル: このアプローチはいかなる IDE にも対応します。
- Keil µVision ツール: これはサポート IDE ですが、このオプションは、ユーザーの状況に最適なパス (CMSIS Pack または手動によるインポート) を決断できるように対照を成します。
- サンプル プロジェクトの CE212736: コードのインポート時に直面する選択肢を検討する際に豊富なドメインを提供します。
- CE212736「PSoC™ 6 MCU with Bluetooth® Low Energy Connectivity」は Bluetooth® LE スタック、アセンブラ ソース コード、事前コンパイルされたバイナリ ファイル、およびいくつかのペリフェラル ドライバーを使用します。このウォークスルーは、ハードウェアでサンプル コードも実行せず、その機能も説明しません。

このウォークスルーは CM4 プロジェクトにコードをインポートします。マルチ CPU デバイスでは、CM0+と CM4 両方のプロジェクトをビルドします。

設計を完了し、PSoC™ Creator プロジェクトをビルドした後、それぞれの CPU のプロセスは同じです。

1. IDE プロジェクト ファイルをセットアップしてください。
2. プロジェクトにファイルを追加してください。
3. インクルード パスをセットしてください。
4. IDE でプロジェクトをビルドしてください。

5.1 設計の完了およびアプリケーションファイルの生成

IDE へのインポートの前に、PSoC™ Creator でアプリケーションを正常に生成できることが必要です。図 7 に、CE212736 プロジェクトのアプリケーションファイルを生成した後の PSoC™ Creator Workspace Explorer を示します。

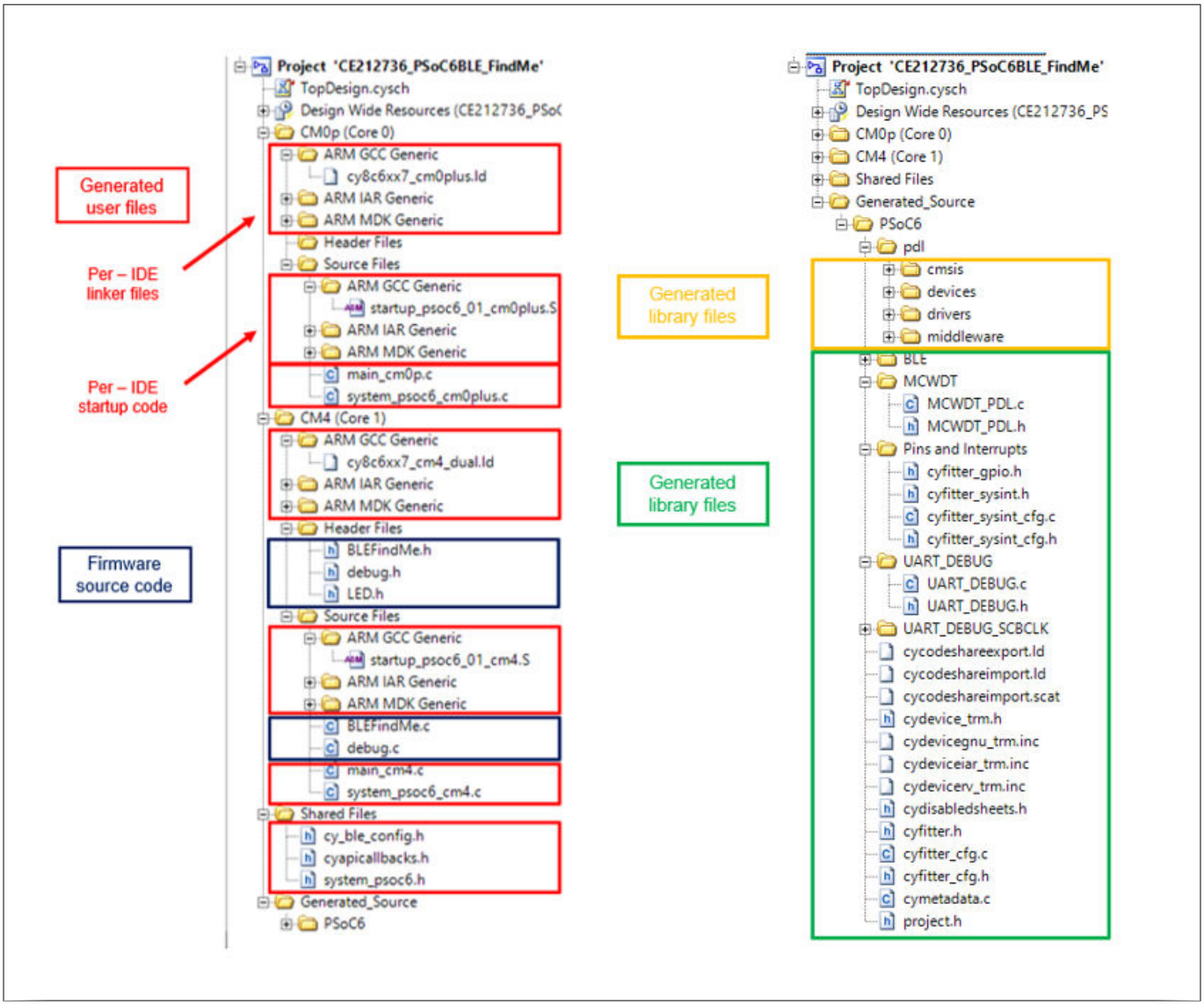


図 7　　PSoC™ Creator の Workspace Explorer

Workspace Explorer ペインの `Generated_Source` 項目は、プロジェクトの `cydsn/Generated_Source` フォルダ内のファイルをすべて含みます。この設計は Bluetooth® LE コンポーネント、マルチカウンター ウォッチドッグ タイマー (MCWDT) 、およびターミナル ウィンドウにデバッグ メッセージをプリントするための UART を使用します。また、LED を制御するためにピンを持っており、システム割込みをセットアップします。

完全なサンプル コードであるため、このプロジェクトは例を実装するファームウェア ソース コードも含みます。別の IDE でのコード開発を好む場合、通常は PSoC™ Creator の代わりにその IDE でこれらのファイルを作成します。

5.2　　IDE プロジェクト ファイルのセットアップ

新規プロジェクトを作成します。通常、IDE はターゲット デバイスを指定することを必要とします。IDE のプロジェクト作成プロセスには追加のステップがある可能性があります。

例えば、Keil µVision ツールでは、**Project > New µVision Project** を選択します。Infineon GitHub repository から PSoC™6_DFP パックをダウンロードして、パックをインストールできます。このパックは、Keil µVision および CMSIS パックをサポートするその他の IDE で PSoC™ 6 デバイスをサポートします。適切なデバイスを使用すると、メモリアドレス範囲、プログラミングアルゴリズムなどが自動的に設定されます。デバイスが利用できない場合、

5 生成コードの手動によるインポートの 1 例

または CMSIS パックをサポートしていない IDE を使用している場合は、プロジェクトに適切な Arm® CPU を選択して開始し、デバイス固有の必要な詳細を手動で提供できます。

デバイスを指定した後、ソフトウェア パックを選択します。図 8 に例を示します。このウォークスルーでは、ソフトウェア パックは必要ありません。

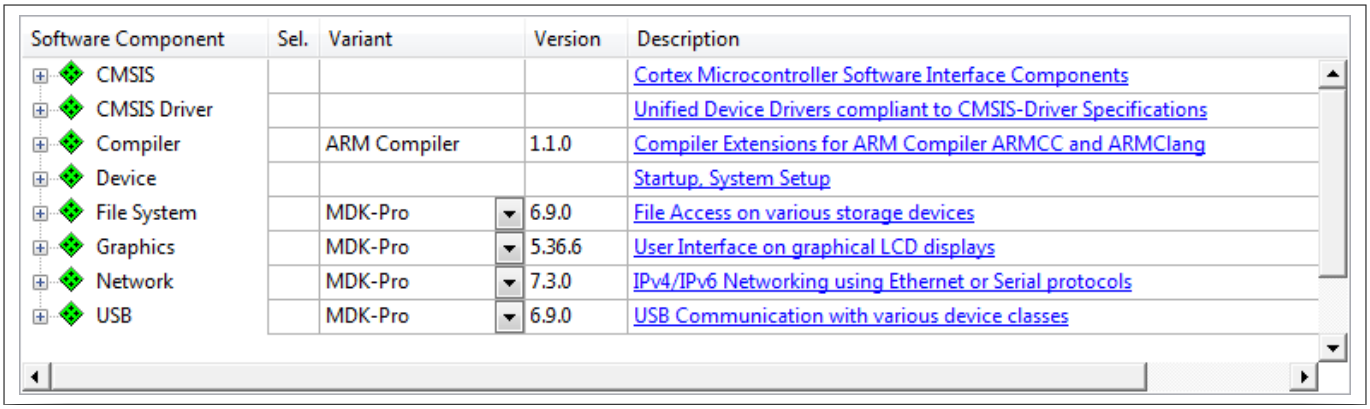


図 8　　µVision のソフトウェア パック選択

完了すると、図 9 に示すように空のプロジェクトが現れます。本例では、プロジェクトを「CM4 Project」と名付けます。マルチ CPU デバイスに対してコードをインポートする際、CPU ごとに 1 プロジェクトが必要です。

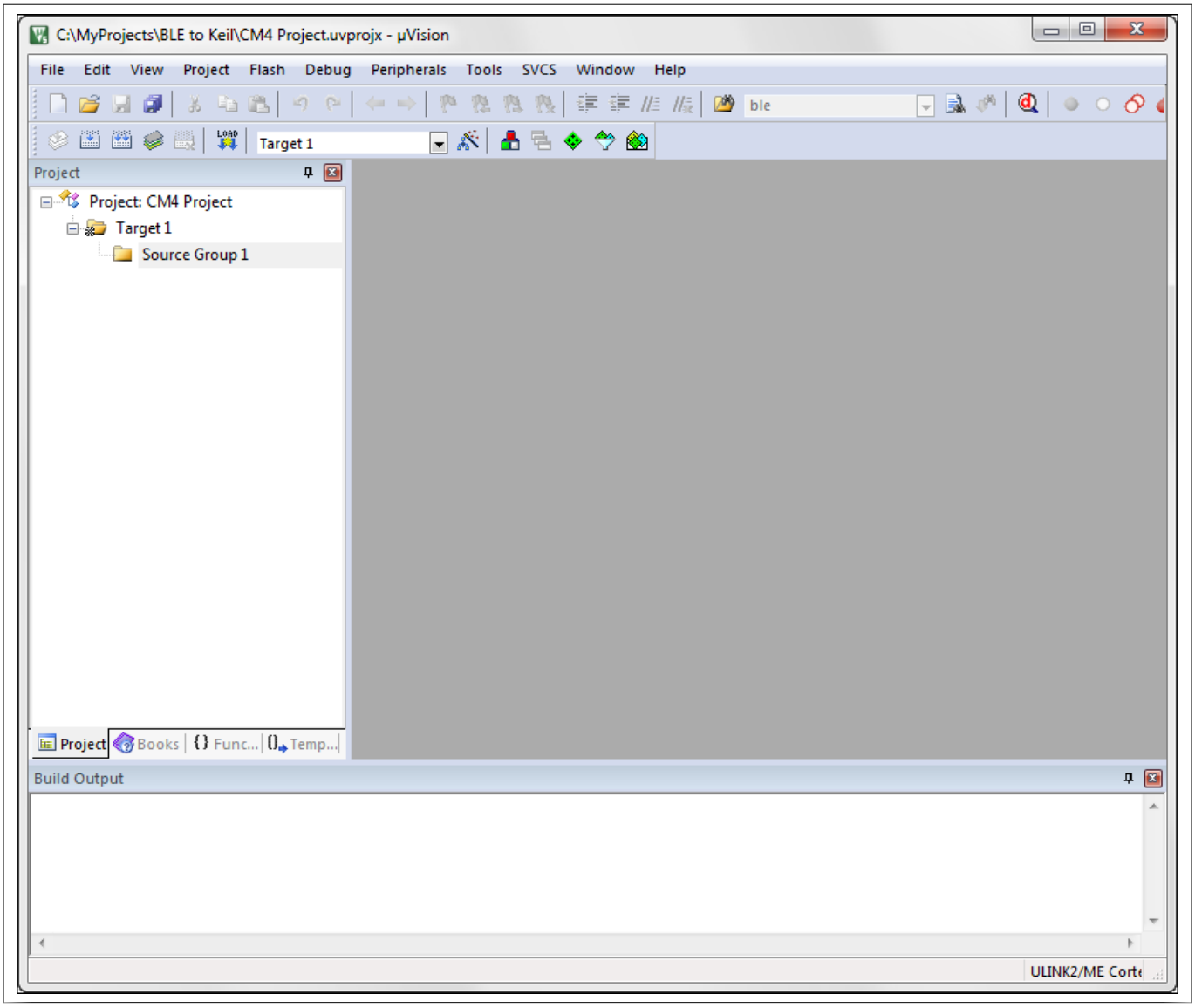


図 9　　µVision IDE での空のプロジェクト

5.3　プロジェクトへのファイルの追加

必要なファイルを特定し、それらをプロジェクトに追加します。それぞれの IDE は、ファイルを追加する独自の方法を持っています。

IDE は通常、何らかのファイル エクスプローラ ペインを持っています。適切にファイルを整理するようにプロジェクト内のグループを追加したり除去したりすることができます。本節でのガイドはガイドに過ぎません。その他の構成スキームを好む場合があります。図 10 に、このウォークスルーで使用するグループ構造を示します。これは 1 つの例外を除き、PSoC™ Creator が生成するファイルの種類と一致します。Bluetooth® LE スタックが大きなファイル収集であるため、専用グループに分けられます。

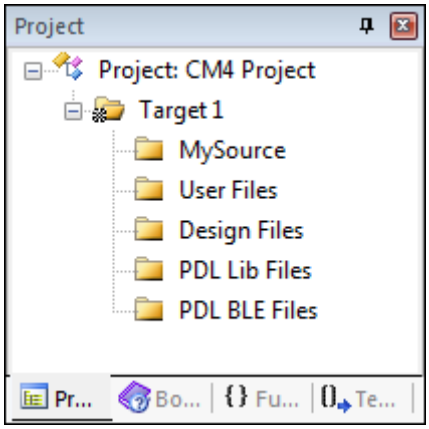


図 10　　µVision IDE での任意のグループ構造

5.3.1　ファームウェア ファイルの追加

本ウォークスルーはフル機能のサンプル コードをインポートするため、図 11 に示すように MySource グループにファームウェア ファイルをインポートします。PSoC™ Creator プロジェクトでは、すべてのファームウェア ファイルは cydsn フォルダにあります。ユーザー独自のプロジェクトでは、独自のファームウェア ファイルを作成し、プロジェクト ファイル内で適切に整理します。

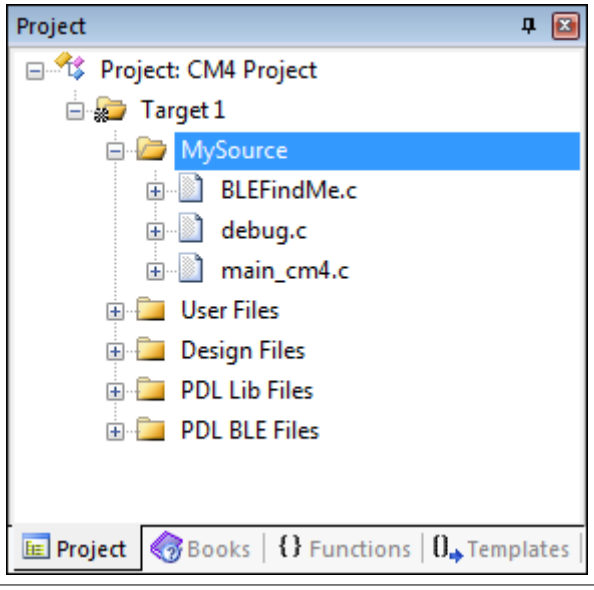


図 11　　プロジェクトに追加したファームウェア ファイル

注:　　CM0+ プロジェクトに対してこのプロセスを繰り返す際に、その CPU に特有のファイルを追加します。

5.3.2　ユーザー ファイルの追加

ユーザー ファイルは PSoC™ Creator によって生成されますが、ユーザーによって変更できます。ユーザー ファイルは cydsn フォルダのトップ レベルに配置されます。ステートアップ コードは IDE 固有のフォルダにあります。ユーザー ファイルを参照してください。この場合、ユーザー ファイルは以下のとおりです。

- システム コンフィギュレーション ファイル (CPU 固有)
- スタートアップ ファイル (IDE 固有かつ CPU 固有)

図 12 に、プロジェクト エクスプローラでのユーザー ファイルを示します。

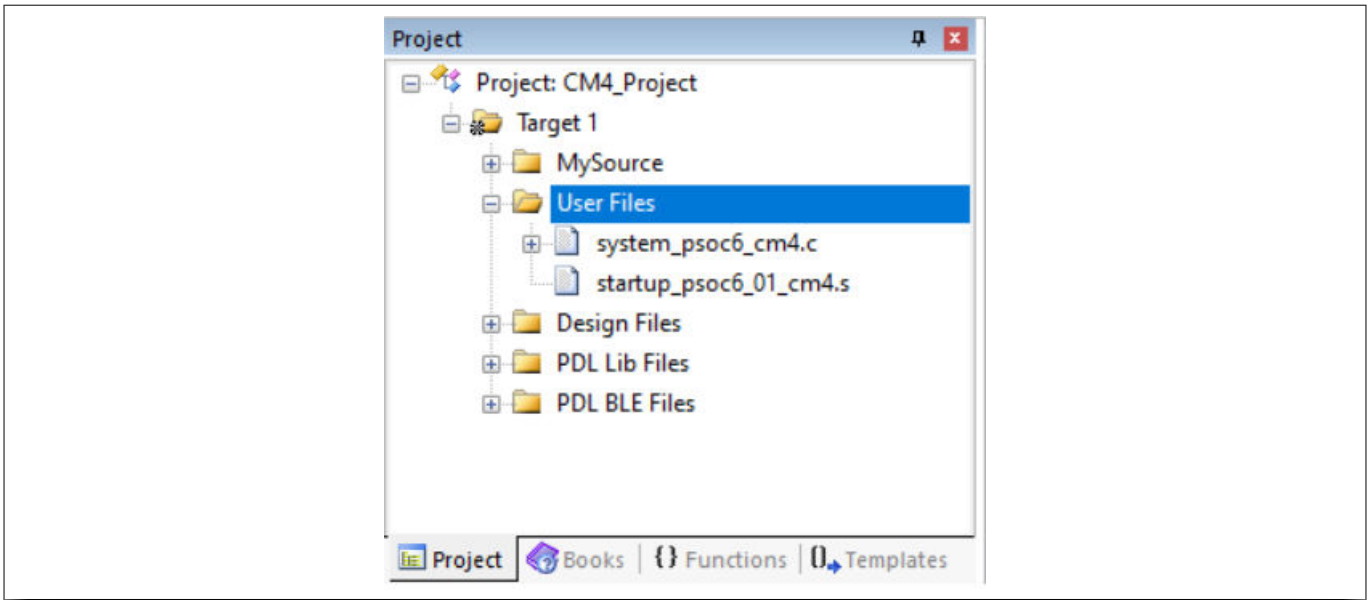


図 12　　プロジェクトに追加したユーザー ファイル

注:　　スタートアップ ファイルは IDE 固有のアセンブラ ソース コードです。生成コードはすべての IDE 用のスタートアップ コードを含みます。ユーザーに必要なファイルを使用します。非サポートの IDE を使用する場合、ユーザー独自のスタートアップ コードを使用します。

5.3.3　　デザイン ファイルの追加

デザイン ファイルは、ユーザーのシステム設計およびそのコンポーネントに特有の、PSoC™ Creator によって生成されるソース ファイルとヘッダ ファイルです。デザイン ファイルを参照してください。デザイン ファイルは `cydsn/Generated_Source/PSoC6` フォルダにあります。

図 13 に、µVision プロジェクトに追加されたこのプロジェクトのデザイン ファイルを示します。これらには、`cyfitter` と `cymetadata` ファイル、および Bluetooth® LE、MCWDT、UART コンポーネントに特有のソース ファイルがあります。

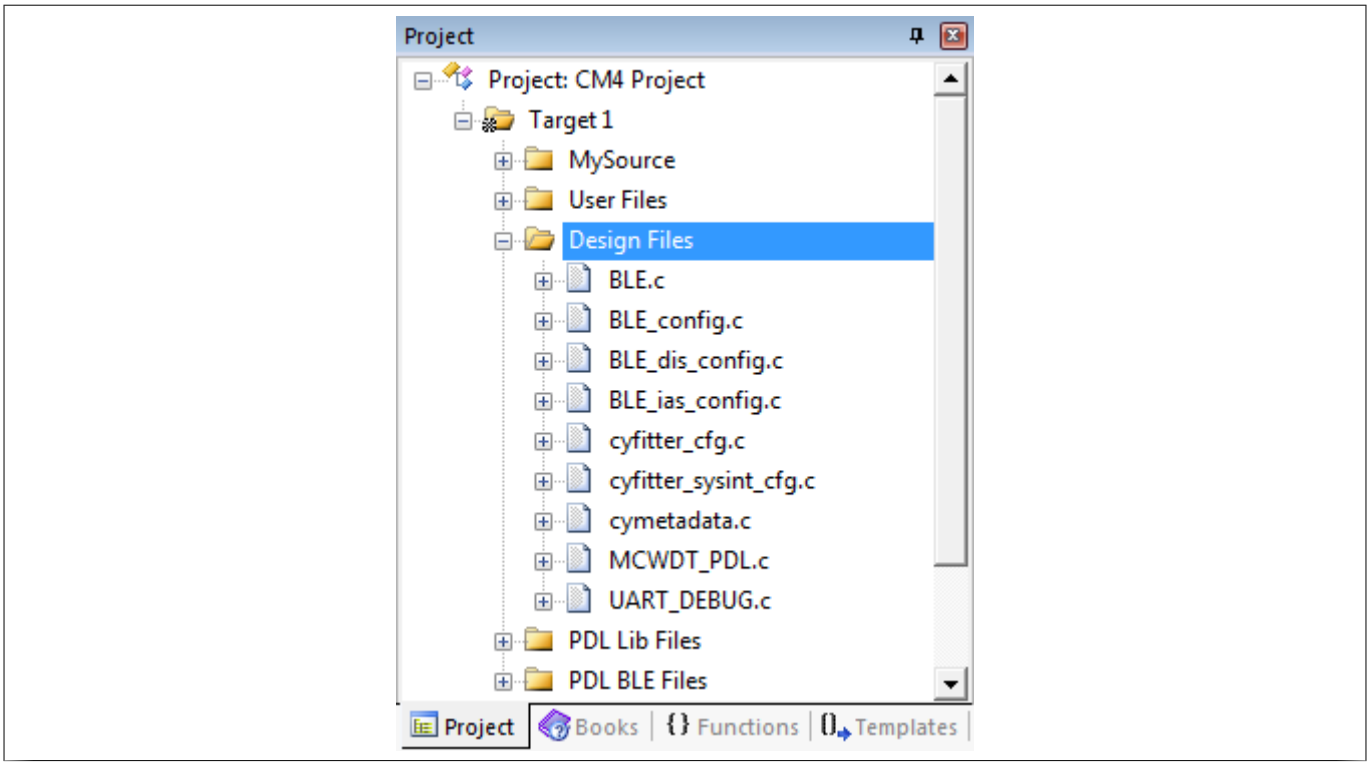


図 13　　プロジェクトに追加したデザイン ファイル

5.3.4　　PDL ライブラリ ファイルの追加

ライブラリ ファイルは PDL インストールからコピーされます。ライブラリ ファイルを参照してください。設計に基づいて、PSoC™ Creator は必要なファイルを `cydsn/Generated_Source/PSoC6/pdl/drivers/peripheral` フォルダにコピーします。図 13 に、IDE に追加されたこのプロジェクトのペリフェラル ドライバー ファイルを示します。

5 生成コードの手動によるインポートの 1 例

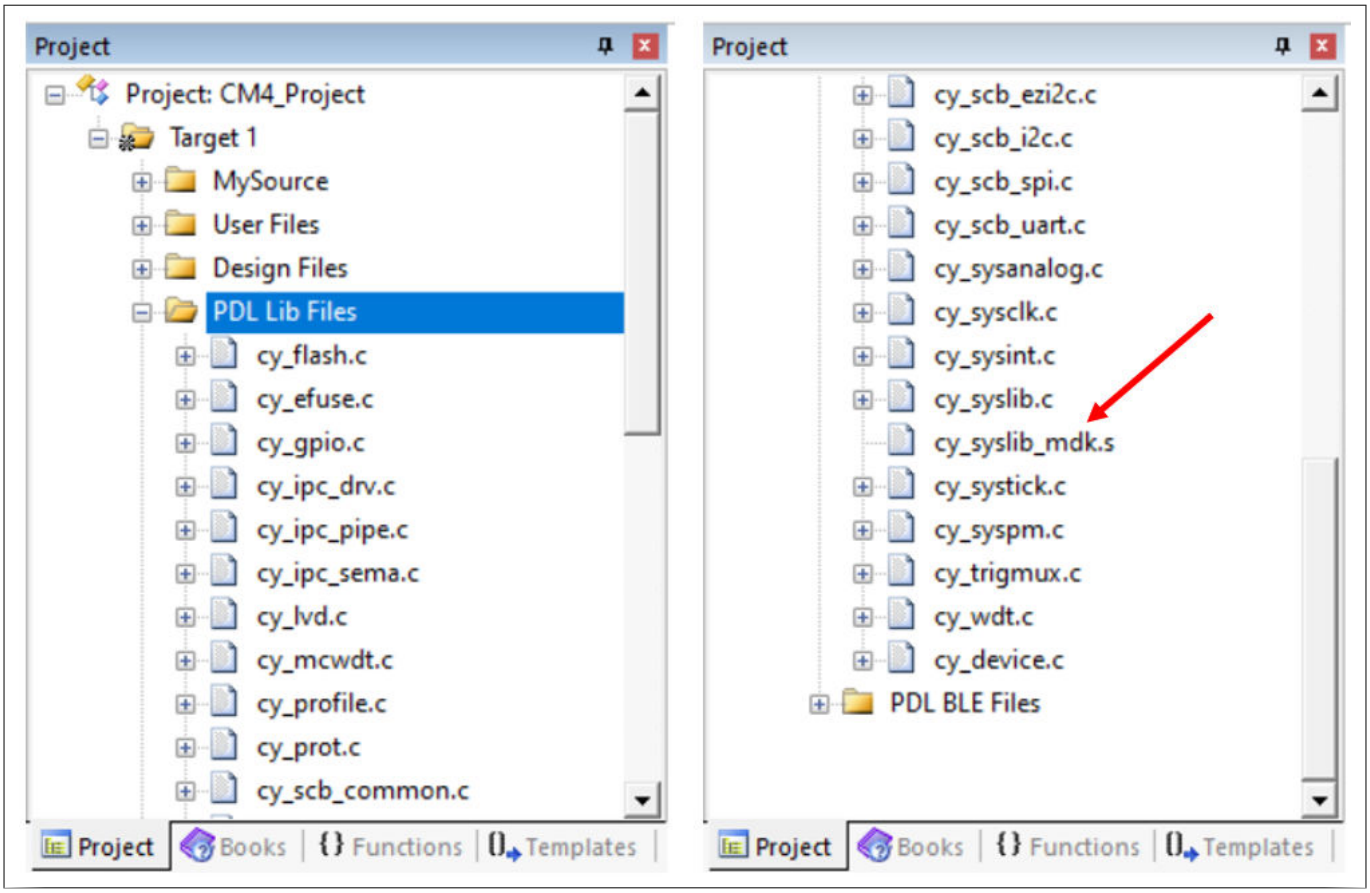


図 14　　プロジェクトに追加したライブラリ ファイル

注:　システム ライブラリ (syslib) の実装は、図 14 に示した IDE 固有のアセンブラ ファイルを含みます。もしこのファイルがプロジェクトに追加されなかったら、ビルドは失敗します。

注:　peripheral フォルダ内には、それぞれのドライバー独自のサブフォルダがあります。フォルダ内で上下に移動したら、ソース ファイルを追加するのは面倒になります。ドラッグアンドドロップによるファイル追加をサポートする IDE では、タスクを容易にするために Windows Explorer を使用することができます。peripheral フォルダを開き、.c を検索します。すべての .c ファイルは図 15 のように現れます。そして、これらのファイルを IDE にドラッグできます。(残念ながら、このヒントは µVision IDE に適用できません。) アセンブリ ソース ファイルも追加することを忘れないでください。*

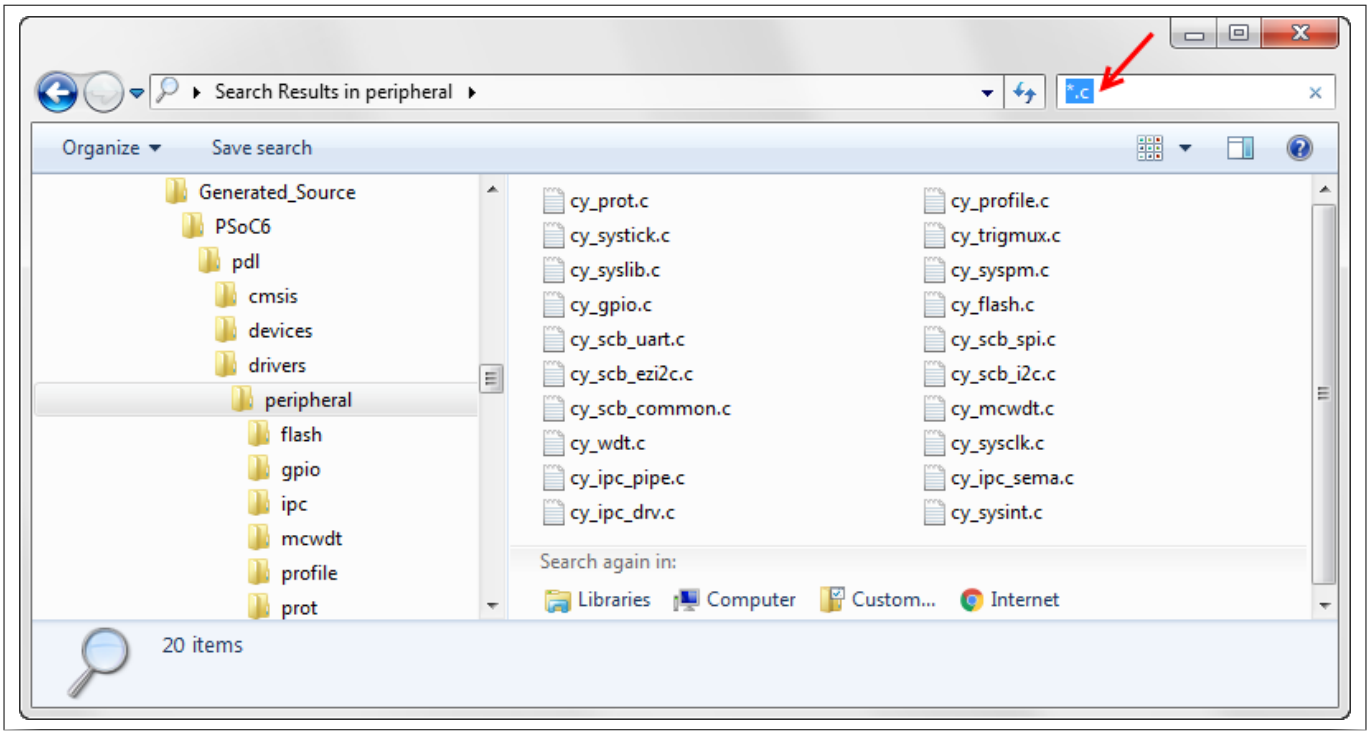


図 15 .c ファイルの検索

5.3.5 Bluetooth® LE ライブラリ ファイルの追加

この設計で使用する最後のファイル収集は Bluetooth® LE スタックです。生成コードは `cydsn/Generated_Source/PSoC6/pdl/ middleware/ble` フォルダに配置されています。Bluetooth® LE スタックの一部は、ソース コードではなく、CPU 特有のバイナリ ライブラリとして提供されています。必要なソース ファイルおよびバイナリ ライブラリはすべてプロジェクトに追加しなければなりません。

5 生成コードの手動によるインポートの 1 例

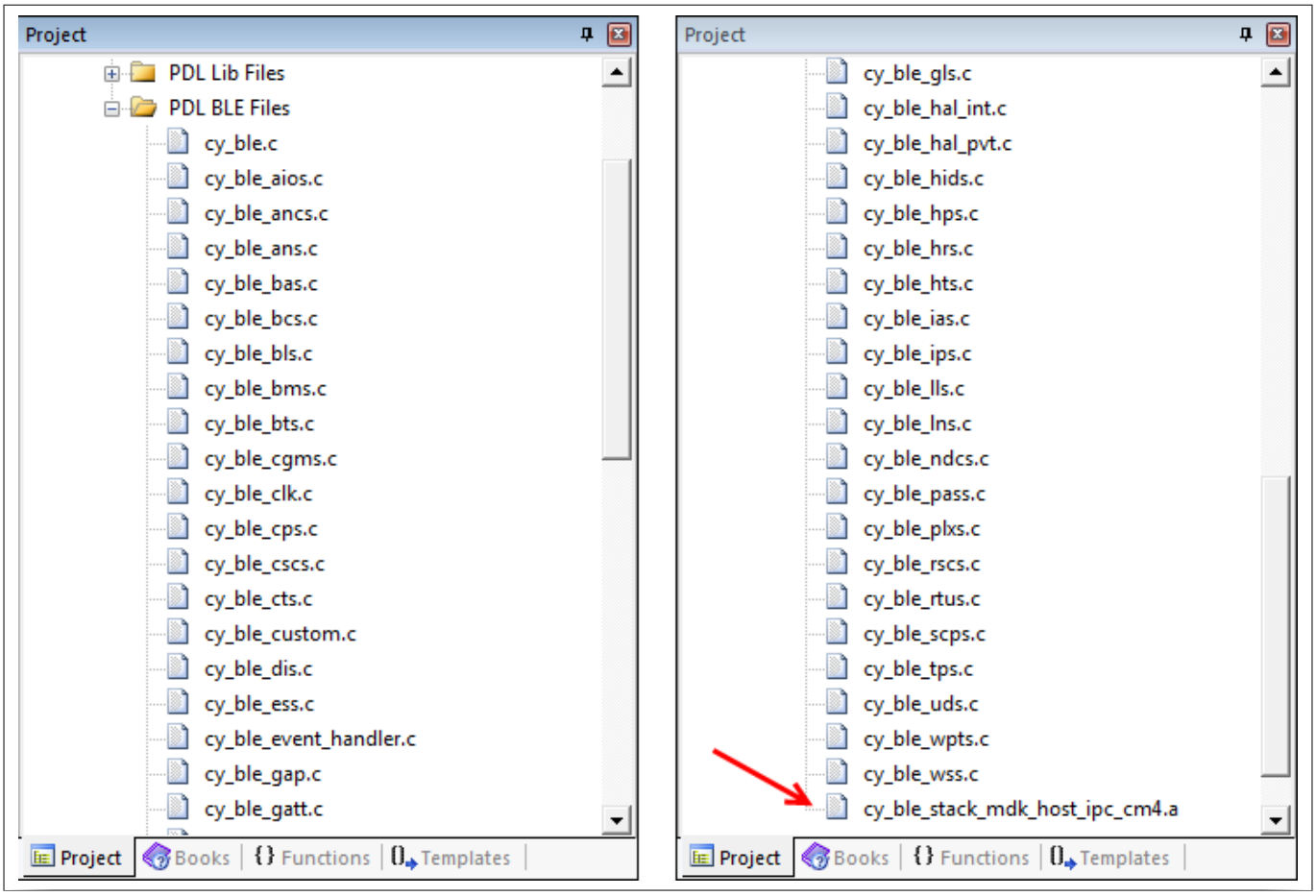


図 16　　プロジェクトに追加した Bluetooth® LE スタック ファイル

注:　CM0+プロジェクトに対してこのプロセスを繰り返す際に、その CPU に特有のライブラリを追加します。

注:　ライブラリ ファイルをライブラリよりもアセンブラ ソース ファイルとして取り扱い、ビルド エラーを引き起こす IDE があります。その IDE は、この問題を制御するプロパティを持っていることがあります。例えば、図 17 に µVision IDE のファイル プロパティ ダイアログを示します。

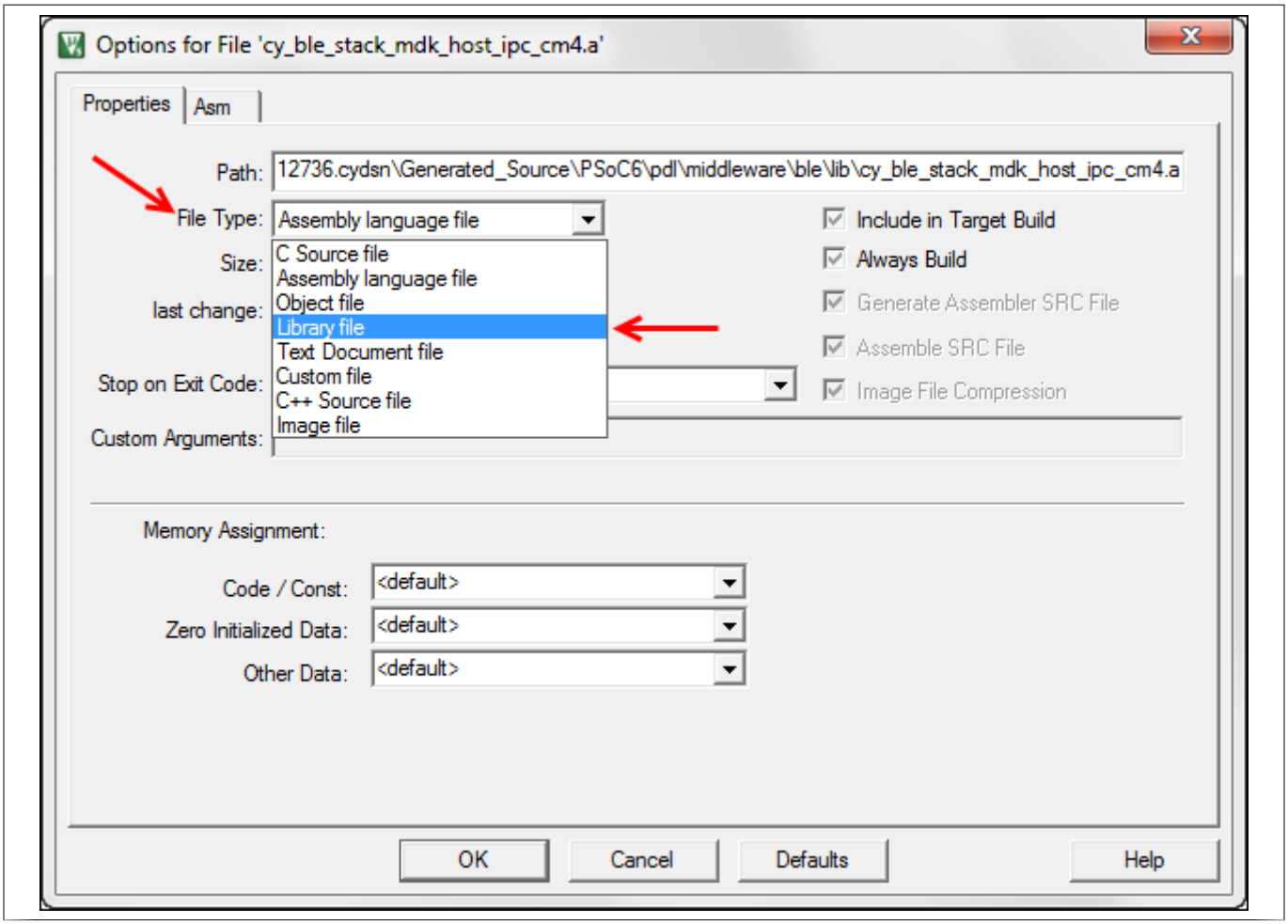


図 17　　µVision IDE におけるファイル タイプ設定

5.4　インクルード パスの設定

IDE のビルド システムは必要なヘッダ ファイルを配置する必要があります。ファイルへの容易なアクセスのために、プロジェクト エクスプローラにヘッダ ファイルを直接追加することを好む開発者がいます。しかし一般的には、これは IDE のインクルード パスをセットしません。

それぞれの IDE はインクルード パスをセットする独自の方法を持っています。本例は、好む IDE にインクルード パスを追加し方が分かることを前提とします。本節の目的は、パスの追加方法ではなく、何のパスを追加する必要があるかを説明します。

µVision IDE では、**Include Paths** 項目でのターゲットの C/C++オプションのパネルでパスを追加します。

5.4.1　ユーザー ファイルへのパスの設定

PSoC™ Creator が生成するユーザー ファイルのすべてのヘッダ ファイルは `cydsn` フォルダにあります。確かに、パスは PSoC™ Creator プロジェクトを配置する場所に依存します。本例では、CE212736 プロジェクトのパスは以下のようになります。

```
C:/MyProjects/CE212736/CE212736.cydsn
```

図 18 に、µVision IDE でのパスの設定を示します。

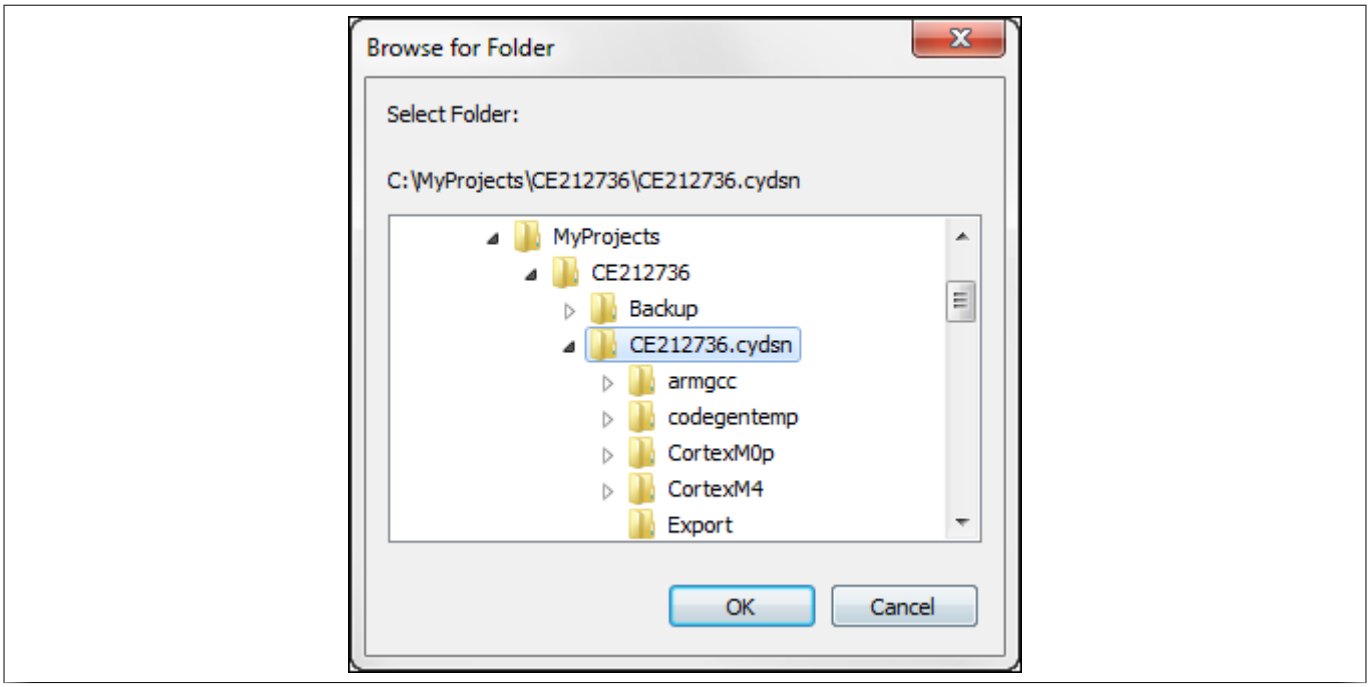


図 18　　cydsn フォルダへのパスの設定

注:　　本例では、ファームウェア ヘッダ ファイルもこのフォルダにあるため、それら用の個別のパスは必要ありません。

5.4.2　　デザイン ファイルへのパスの設定

PSoC™ Creator が生成するデザイン ファイルのすべてのヘッダ ファイルは `cydsn/Generated_Source/PSoC6` フォルダにあります。本例では、CE212736 プロジェクトのパスは以下のようになります。

```
C:/MyProjects/CE212736/CE212736.cydsn/Generated_Source/PSoC6
```

図 19 に、µVision IDE でのパスの設定を示します。

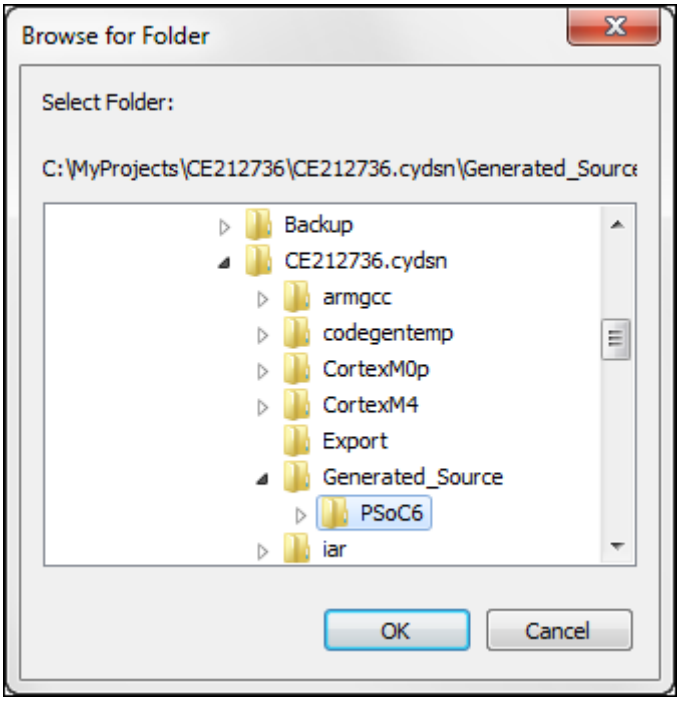


図 19　　デザイン ファイルへのパスの設定

5.4.3　　`peripheral` フォルダ (ドライバー ファイル) へのパスの設定

各ペリフェラル ドライバーはそれ自身のサブフォルダを持っており、各 Peripheral フォルダにはプロジェクトに含める必要があるヘッダ ファイルがあります。そのため、各ドライバー サブフォルダへのパスをインクルードパスの一部として追加する必要があります。たとえば、このプロジェクトにフラッシュ ドライバー ヘッダ ファイルを含めるために追加するパスは次のようになります。

```
C:/MyProjects/CE212736/CE212736.cydsn/Generated_Source/PSoC6/pdl/drivers/peripheral/flash
```

これに加えて、すべてのドライバー サブフォルダを含む Peripheral フォルダへのパスも追加する必要があります。CE212736 プロジェクトの Peripheral フォルダ パスは次のようになります。

```
C:/MyProjects/CE212736/CE212736.cydsn/Generated_Source/PSoC6/pdl/drivers/peripheral
```

図 20 に、µVision IDE での Peripheral フォルダ パスの設定を示します。

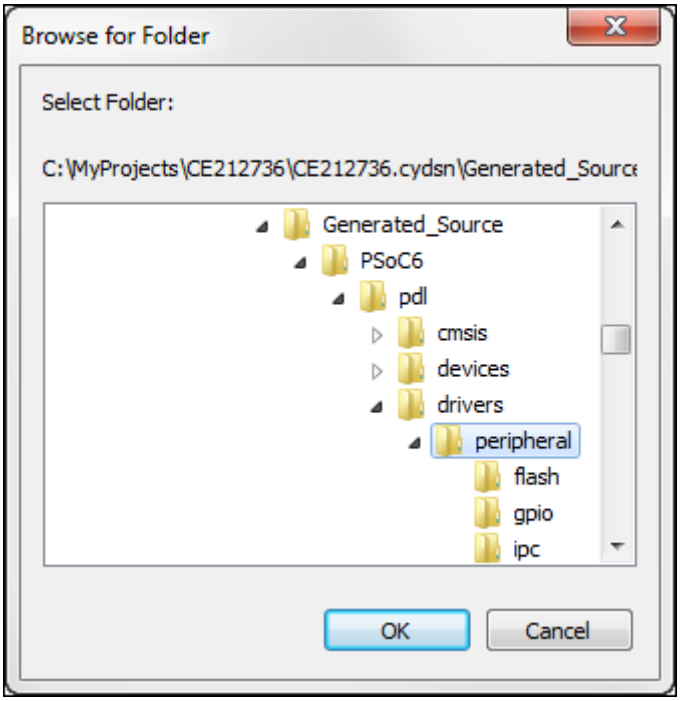


図 20　　ペリフェラル フォルダへのパスの設定

5.4.4　　`middleware` フォルダ (Bluetooth® LE スタック) へのパスの設定

Bluetooth® LE スタックがサブフォルダ内にありますが、ソース コードは middleware フォルダ内のすべてのヘッダ ファイルを含むパスを提供します。その結果、単一のパスはこのフォルダ ツリーのすべての場所に対応しています。本例では、CE212736 プロジェクトのパスは以下のようになります。

```
C:/MyProjects/CE212736/CE212736.cydsn/Generated_Source/PSoC6/pdl/middleware
```

図 21 に、µVision IDE でのパスの設定を示します。

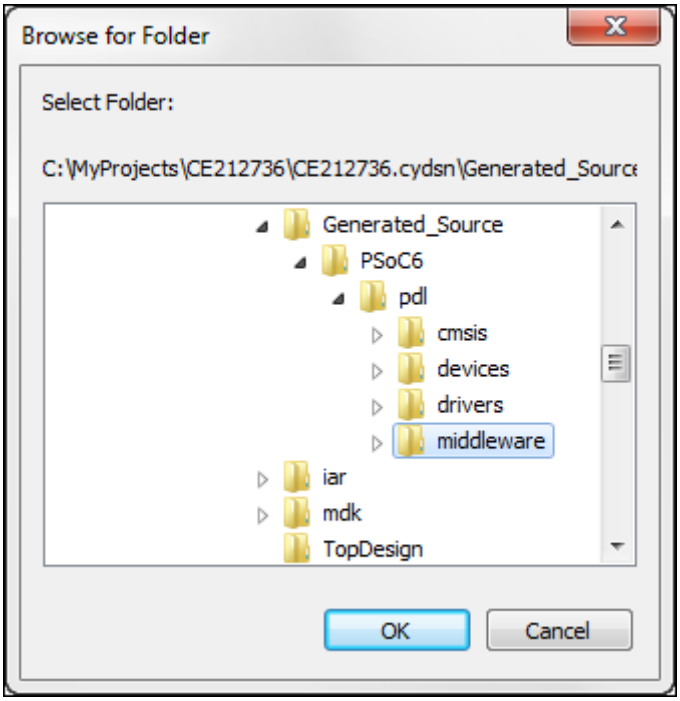


図 21　　Bluetooth® LE スタックへのパスの設定

5.4.5　その他の必要なパスの設定

正常なビルドのために含む必要があるその他のヘッダ ファイルがあります。PSoC™ Creator はそれらのヘッダ ファイルのコピーを `cydsn/Generated_Source/PSoC6/pdl` フォルダ内に提供します。

- CMSIS ヘッダ ファイル: `cydsn/Generated_Source/PSoC6/pdl/cmsis/include`
- IP ヘッダ ファイル: `cydsn/Generated_Source/PSoC6/pdl/devices/psoc6/include/ip`

シリーズ固有のヘッダ ファイル: `cydsn/Generated_Source/PSoC6/pdl/devices/psoc6/include`

図 22 に、µVision IDE でセットされた、生成コードを指すこれらの 3 つのパスを示します。

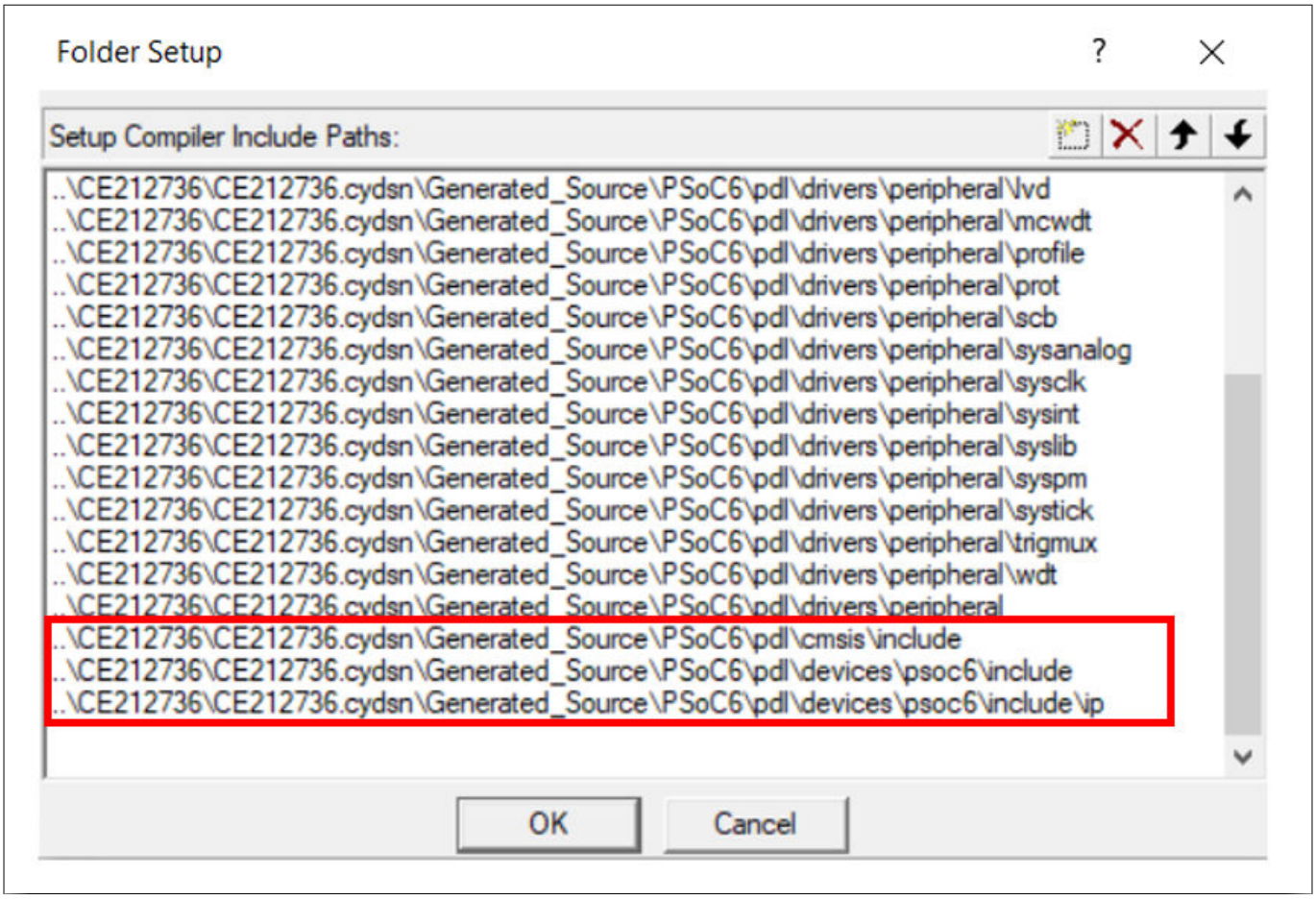


図 22　　その他の必要なヘッダ ファイルへのパス

注:　PDL テンプレート プロジェクトで始まるなら、これらの 3 つのパスは既にセットされている場合があります。しかし、生成コードの代わりに、PDL インストール内のオリジナル ファイルを指します。cydsn フォルダ内の対応する場所を指すようこれらのパスを修正する必要があります。PDL ライブラリ ファイルの場所を参照してください。

5.5　IDE でのプロジェクトのビルド

アプリケーションを正常にビルドするために、コードのコンパイルの前にすべての必要なオプションを設定しなければなりません。本アプリケーションノートは、特定の IDE でのプロジェクト設定ではなく、IDE への生成コードのインポート方法について説明します。ただし、付録 A は追加の基礎知識を提供します。

すべての必要なファイルを追加し、すべてのパスを正しくセットした後、プロジェクトをビルドします。ビルドは正常に成功する場合もあり、警告やエラーが発生する場合もあります。各 IDE は、独自のインターフェースと独自のオプション、デフォルト設定および警告/エラー メッセージを持っています。IDE のばらつきにより、本アプリケーションノートはエラー処理の詳細な指示を提供できません。ご使用の IDE に熟知していることは、発生するすべての問題を解決するのに大いに役立ちます。

ただし、コードのインポートについては通常、発生し得る 3 つのエラー タイプがあります。

ヘッダ ファイルに対して「File Not Found」(ファイルが見つからない) エラーが発生した場合、ご使用のファイル システム内の実際のファイルを探します。そのファイルのインクルード パスがあることを確かめてください。

関数やシンボルが未定義の場合、すべてのユーザー ファイル、デザイン ファイル、ライブラリ ファイルがプロジェクトに追加されたことと、シンボルを宣言するすべてのヘッダ ファイルが見つかったことを保証します。例えば、バイナリ ライブラリを含むことに失敗した場合、このエラー タイプが発生します。

すべてのファイルおよびパスが正しくても、一連のコンパイルやリンカー エラーが発生する場合があります。プロジェクトで正しくない設定があるかもしれません。例えば、IDE のコンパイラがデフォルトで C99 規格をサポートしていない場合は、そのサポートが有効になるようにしてください。

5.6 例のレビュー

ここでは、IDE にコードを手動でインポートする 1 例を提供しました。単一の例はすべての状況に対応できませんが、タスクは簡単です。

1. IDE でプロジェクト ファイルを作成および設定してください (マルチ CPU デバイスの場合は CPU ごとに 1 プロジェクト ファイル)。あるいは、PDL テンプレート プロジェクトを使用して始まる。
2. プロジェクト ファイルに必要な PSoC™ Creator 生成ファイルを追加してください (マルチ CPU デバイスの場合はそれぞれの CPU に対して行う)。
3. ファイルのインクルード パスを設定してください。

PDL テンプレート プロジェクトはフラッシュ コンフィギュレーション、リカーおよびスタートアップ ファイルを提供します。非サポート IDE を好むなら、それらのファイルを参考にします。IDE 固有のファイルは<PDL Install Folder>/devices/psoc6 にあります。

最後に、本アプリケーションノートは、ファイルのタイプ、場所および使用法などの基本原理を説明します。状況に応じてこの知識を適用する必要があります。

6 まとめ

PSoC™ 6 MCU のようなデュアル CPU 組込みシステム向けのソフトウェアを書くことは、大変なタスクになることがあります。PSoC™ Creator はそのプロセスを簡素化します。ユーザーは使いやすい UI により設計を作成および設定します。文字通りボタンのクリックで、PSoC™ Creator は設計の実装に必要なコードをすべて生成します。ユーザーは実の価値に焦点を置き、その生成コードに基づいてファームウェアを作成します。

PSoC™ Creator でファームウェア全体を開発することができますが、多くの開発者および組織が好む開発システムを持っています。

本アプリケーションノートは、サポートされる IDE (エクスポートとインポート) またはすべての IDE (必要なファイルの手動によるインポート) のサードパーティ IDE で PSoC™ Creator 生成コードを使用する方法を示しました。生成ファイルの異なるタイプ、ファイルが配置される場所、および IDE プロジェクトにファイルを追加し方を説明しました。

本アプリケーションノートが提供する知識は、開発時間を短縮させる高品質の生成コードと好む IDE の組合せを可能にします。

関連資料

アプリケーションノート	
AN210781 – PSoC™ Creator での PSoC™ 6 Bluetooth® LE 入門	Bluetooth® LE 接続を備えた PSoC™ 6 MCU デバイスおよび初めての PSoC™ Creator プロジェクトのビルド方法について説明
AN215656 – PSoC™ 6 MCU: Dual-Core CPU System Design	PSoC™ 6 MCU のデュアルコア CPU アーキテクチャおよびシンプルなデュアル CPU 設計のビルド方法について説明
AN225588 – PSoC™ 6 MCU Importing Generated Code into an IDE	PDL および PSoC™ Creator ではなく、ModusToolbox™ソフトウェアおよびコンフィギュレーターでの類似プロセスについて説明
PSoC™ 6 MCU	
PSoC™ 6 MCU ホーム ページ	PSoC™ 6 MCU のすべてのリソースへのアクセス
PSoC™ 6 MCU コミュニティ	PSoC™ 6 MCU の問題を相談
ペリフェラル ドライバ ライブラリ (PDL と共にインストールされたドキュメント)	
Peripheral Driver Library v3.0 ユーザー ガイド	ライブラリの概要および使用法
ペリフェラル ドライバ ライブラリ API リファレンス	API の詳細なテクニカル リファレンス
PSoC™ Creator	
PSoC™ Creator 製品ページ	ダウンロード、トレーニング、コンポーネントなどへのアクセス
PSoC™ Creator クイック スタート ガイド	迅速に準備および実行ができる
PSoC™ Creator ユーザー ガイド	包括的なマニュアル
開発キット資料	
CY8CKIT-062-BLE PSoC™ 6 BLE Pioneer Kit	

A 付録 A -IDE プロジェクト ファイルの設定

コードを正常にビルドするために、IDE プロジェクト ファイルに対してさまざまなオプションを持つ設定を行う必要があります。各 IDE は独自の UI およびデフォルト設定を持っているので、特定の IDE で空のプロジェクトを設定する方法は、本アプリケーションノートの範囲外です。ただし、設定する必要のあるものの一般的なコンセプトは非常に有益です。この知識を持って、IDE で適切なオプションを特定および設定することが可能です。

表 6 は、IDE プロジェクト ファイルの設定に必要なオプションのいくつかをリストアップします。特定の IDE では、追加のオプションがある場合があります。すべてのオプションのデフォルト設定はすでに正しくセットされていることがあります。

表 6 IDE オプション

タイプ	オプション	説明
デバイス	ターゲット	IDE が特定のデバイスをサポートしたら、そのデバイスを選択。これは通常、その他のデバイス固有のオプションを自動でセット。IDE にデバイスがリストアップされない場合、汎用の Cortex® M4 または M0+デバイスを選択し、すべてのデバイス固有のオプションが正しくセットされるようにする
コンパイラ	C99 サポート	C99 規格のサポートを有効にする
	浮動小数点	CM4 CPU の浮動小数点サポートを有効にする
	最適化	ビルドに適切なコンパイラ最適化レベルをセット
	デバッギング	デバッグ ビルド用のデバッグ情報を生成
	インクルード パス	インクルード パスをセット。本アプリケーションノートは生成コードに関連するすべてのパスを説明
	デバッグ シンボル	条件付きでコンパイルされた PDL コードには、デバッグ シンボルを定義することが必要。デバッグ ビルドでは、DEBUG を定義。非デバッグ ビルドでは、NDEBUG を定義
	デバイス シンボル	条件付きでコンパイルされたデバイス固有 PDL コードには、正しいシンボルを定義することが必要。このシンボルは、どのデバイス固有のヘッダ ファイルをビルドに使用するかを制御。cy_device_headers.h を参照してください
リンカー	コマンド ファイル	リンカー スクリプトへのパスを指定。PDL はサポート IDE 用のリンカー ファイルを提供。IDE がデバイス固有のサポートを持っていれば、これは自動的にセットされることがある
	その他の設定	ビルドにその他のリンカー オプションをセット。例えば、リンカー マップ ファイルの生成、デバッグ情報の出力、ログ ファイルの生成など
デバッガ	デバッグ接続	J-Link や CMSIS-DAP などのサポートされるデバッグ接続 (プローブ) を指定
	接続設定	デバッグ接続に基づき、リセット オプション、接続速度、キャッシュ オプション、ダウンロード オプションなどのさまざまな接続設定を指定
	メモリ構成	デバイスのメモリ領域を指定。IDE がデバイス固有のサポートを持っていれば、これは自動的にセットされることがある
	レジスタ記述	PDL はレジスタ レベルのデバッグ情報用のシステム ビュー記述 (SVD) ファイルを提供。IDE がデバイス固有のサポートを持っていれば、これは自動的にセットされることがある
	フラッシュローダ	デバイスに実行ファイルをダウンロードするために使用するフラッシュローダを指定。IDE がデバイス固有のサポートを持っていれば、これは自動的にセットされることがある

B 付録 B -学習目的での生成コードの使用

PSoC™ Creator 生成コードのインポートが不可の場合にも、コードはファームウェア開発を支援する価値のあるリソースです。

この場合、PSoC™ Creator 生成コードの主な役割は PDL の使用法の学習リソースです。これが大いに助けるいくつかの領域は以下のとおりです。

- クロック コンフィギュレーション: system_psoc6_cm0plus.c、system_psoc6_cm4.c、および cyfitter_cfg.c を参照してください。
- 割込みコンフィギュレーション: cyfitter_sysint_cfg.c を参照してください。
- ピン コンフィギュレーション: cyfitter_gpio.h and cyfitter_cfg.c を参照してください。
- ペリフェラル コンフィギュレーション: コンポーネント固有の.c と.h ファイルを参照してください。
- PDL 関数 API: コンポーネント固有の.c と.h ファイルを参照してください。

それぞれの場合、生成コードの内からコード スニペット、特定の関数やアルゴリズム、または完全なファイルを抽出してユーザー独自のコードに使用できます。例えば、ペリフェラル用のコンフィギュレーション構造をコピーするか、ユーザー独自のコードを書く際にそれらの構造を参考にできます。

PSoC™ Creator は PDL API に基づいてコンポーネント固有の API を生成します。コンポーネント API は通常、PDL API に 1:1 マッピングされた定義済みの関数を持っています。コンポーネント API は、設計に基づいて必要なハードウェア パラメーターを提供します。また一般的には、特定のペリフェラルの初期化、有効化、終了に必要な PDL API 呼び出しを (正しい順番で) 行う `Start()`と `Stop()`関数を含みます。PDL API をどのように使用するかについては、コンポーネント API を検討できます。

さらに、PDL のほとんどがソース コードとして提供されているので、機能と動作の制御用にどのレジスタを使用するかについては、PDL ソース ファイルを検討できます。

さまざまな生成コード ファイルの間に依存関係があります。コード生成プロセスはシンボルを定義し、独自の命名規則を使用します。それぞれのコンポーネント用に完全な API を作成します。どの部分を直接使用するか、どの部分をユーザーのファームウェア開発プロセスに適合させるかは、ユーザーによって決まります。

改訂履歴

版数	発行日	変更内容
英文版*A	-	この版は英語版のみです。英語版の改訂内容: First public release.
**	2018-06-26	これは英語版 002-19434 Rev. *B を翻訳した日本語版 Rev. **です。英語版の改訂内容: Updated to new AN template Updated for PSoC™ Creator UI change for IDE-specific files Updated to allow for single-CPU devices
*A	2019-09-18	これは英語版 002-19434 Rev. *C を翻訳した日本語版 Rev. *A です。英語版の改訂内容: Added discussion of ModusToolbox™ software and analogous process Added discussion of tool compatibility with PSoC™ Creator and PSoC™ Programmer Update to latest AN template
*B	2020-09-03	これは英語版 002-19434 Rev. *D を翻訳した日本語版 Rev. *B です。英語版の改訂内容: Updated ModusToolbox™ information for v2.x, with links to collateral
*C	2021-04-23	これは英語版 002-19434 Rev. *E を翻訳した日本語版 Rev. *C です。英語版の改訂内容: Updated to Infineon Template
*D	2023-08-16	これは英語版 002-19434 Rev. *F を翻訳した日本語版 Rev. *D です。英語版の改訂内容: Template update

Trademarks

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

Edition 2023-08-16

Published by

Infineon Technologies AG

81726 Munich, Germany

© 2023 Infineon Technologies AG

All Rights Reserved.

Do you have a question about any aspect of this document?

Email: erratum@infineon.com

Document reference
IFX-kfw1649847777619

重要事項

本手引書に記載された情報は、本製品の使用に関する手引きとして提供されるものであり、いかなる場合も、本製品における特定の機能性能や品質について保証するものではありません。本製品の使用の前に、当該手引書の受領者は実際の使用環境の下であらゆる本製品の機能及びその他本手引書に記された一切の技術的情報について確認する義務が有ります。インフィニオンテクノロジーズはここに当該手引書内で記される情報につき、第三者の知的所有権の不侵害の保証を含むがこれに限らず、あらゆる種類の一切の保証および責任を否定いたします。

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

警告事項

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

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