PSoC 6 MCU デュアル CPU システム設計

About this document

Scope and purpose

AN215656 は、Arm® Cortex®-M4, Cortex-M0+ CPU, およびプロセッサ間通信 (IPC) モジュールを含む PSoC 6 MCU のデュアル CPU アーキテクチャについて説明します。デュアル CPU アーキテクチャは、システム性能を向上させ効果的に動作し、消費電力も削減します。また、本アプリケーションノートではサイプレスの ModusToolbox®ソフトウェア、Command-line Interface (CLI)、および PSoC Creator IDE を使用してシンプルなデュアル CPU 設計のビルド方法と、様々な IDE を用いた設計のデバッグ方法も説明します。

関連製品ファミリ

デュアル CPU を備えるすべての **PSoC® 6 MCU** デバイス

関連サンプルコード

CE216795

関連アプリケーションノート

関連資料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？以下の通り対応いたします。

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。
サイプレス ビデオ トレーニング ライブラリについては**ここ**からご覧ください。

Table of contents

Table of contents

1 はじめに

PSoC 6 MCU は、IoT 専用に設計されたサイプレスの超低消費電力 32 ビット PSoC です。これには低消費電力フラッシュと SRAM テクノロジ、プログラマブル デジタル ロジック、プログラマブル アナログ、高性能アナログデジタル変換、低消費電力コンパレータおよび標準の通信機能とタイミング ペリフェラルを内蔵します。

PSoC 6 MCU の主な部分は CPU サブシステムです。**Figure 1** に示すように、このアーキテクチャは複数のバス マスタ (2 つの CPU、DMA コントローラーや暗号化ブロック (Crypto)) が組み込まれます。

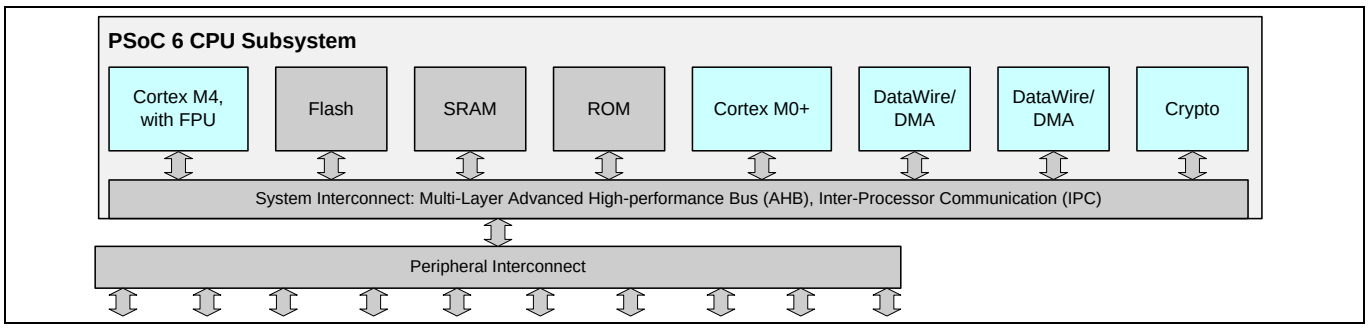

Figure 1 PSoC 6 MCU の標準 CPU サブシステム アーキテクチャ

Note: *Figure 1 のブロック図の内容は、デバイスによって異なります。PSoC 6 MCU の一部の製品にはアプリケーションで使用できる CPU が 1 つしかありません。詳細についてはデバイス データシートをご覧ください。このアプリケーション ノートはシングル CPU PSoC 6 MCU デバイスには適用されません。*

一般的に、すべてのメモリとペリフェラルはすべてのバス マスタによって共有されます。共有リソースは、標準 Arm マルチレイヤ バス アービトレーションを介してアクセスされます。排他的アクセスは、ハードウェアにセマフォと相互排除 (mutex) を実装するプロセッサ間通信 (IPC) ブロックによってサポートされます。

DMA および暗号化 (Crypto) バス マスタを含むデュアル CPU アーキテクチャは、シングル MCU でシステム レベルの設計と性能最適化に比較すると比類がない能力を提供します。2 つの CPU は以下を与えます。

- 複数タスク同時実行の CPU タスク割当て
- CPU にリソースを割当て、CPU がこれらのリソース管理に専念して効率を向上させる
- 消費電力を最小限にするために CPU を有効／無効にする
- IPC ブロックを使用し、CPU 間でデータ送信する詳細はサンプル コード「**CE216795**、PSoC 6 MCU Dual-CPU Basics」を参照してください。

アプリケーションの 1 例では、Cortex-M0+ CPU (CM0+) は、すべての通信チャネルを「所有」し管理します。Cortex-M4 CPU (CM4) は、CM0+経由でチャネルからメッセージを送受信できます。これにより、CM0+が通信を管理している間、CM4 は開放され他のタスクが実行できます。

1.1 ツールとライブラリ

サイプレスは、PSoC 6 MCU を使用したアプリケーション開発に使用できる 2 つの開発プラットフォームを提供します。

- **ModusToolbox**: ModusToolbox ソフトウェアには、構成ツール、低レベルドライバー、ミドルウェアライブラリ、オペレーティングシステムのサポート、および MCU とワイヤレスアプリケーションの作成を可能にするその他のパッケージが含まれます。また、ModusToolbox 用のオプションの Eclipse IDE も含まれます。
- **PSoC Creator**: Windows でのみ実行されるサイプレス独自の IDE。これは、PSoC 6 MCU デバイスのサブセットと、PSoC 3、PSoC 4、PSoC 5LP などの他の PSoC デバイスファミリをサポートします。

どちらのプラットフォームも、Keil µVision、IAR Embedded Workbench、Visual Studio Code などのサードパーティツールにプロジェクトをエクスポートするメカニズムを提供します。

サイプレスは、PSoC 6 MCU を使用したアプリケーション開発を容易にする一連のライブラリも提供します。

- **Peripheral Driver Library (PDL)**: PDL ドライバは、ハードウェア関数を使いやすい API のセットに抽象化します。PDL の詳細については、**https://github.com/cypresssemiconductorco/psoc6pdl** の Web ページにアクセスしてください。このライブラリは、PSoC Creator および ModusToolbox ソフトウェアで利用できます。
- **Hardware Abstraction Layer (HAL)**: HAL は PDL の上に構築されます。サイプレス MCU でハードウェアブロックを構成および使用するための高レベルのインターフェースを提供します。これは、複数の製品ファミリで使用できる汎用インターフェースです。HAL の詳細については、**https://github.com/cypresssemiconductorco/psoc6hal** の Web ページにアクセスしてください。HAL は ModusToolbox ソフトウェアで利用可能であり、CM4 CPU でのみ機能します。PSoC Creator は HAL をサポートしません。
- **Middleware**: このカテゴリには、静電容量センシングや HTTP サーバーなど、さまざまなドメインの API を実装する複数のライブラリが含まれます。ミドルウェアライブラリは、サイプレスによって作成されるか、サードパーティから提供される場合があります。PSoC Creator では、ミドルウェアは通常、PSoC Creator コンポーネントとして利用できます。ModusToolbox ソフトウェアでは、ライブラリとして利用できます。
- **Board Support Package (BSP)**: BSP は、ボードの機能への標準インターフェースを提供します。API は、サイプレスキット全体で一貫しています。他のソフトウェア (ミドルウェアやアプリケーションなど) は、BSP を使用してハードウェアを設定および制御できます。BSP は、ModusToolbox ソフトウェアでのみ使用できます。

1.2 この文書の使い方

本文書では、PSoC 6 MCU アーキテクチャおよびサイプレスの ModusToolbox ソフトウェアまたは PSoC Creator IDE を使用した PSoC デバイス用アプリケーション開発に精通されていることを前提とします。PSoC 6 MCU の説明は、下記を参照してください。

- PSoC 6 MCU **デバイスデータシート**
- **AN221774**, Getting Started with PSoC 6 MCU
- **AN210781**, Getting Started with PSoC 6 MCU with Bluetooth Low Energy (Bluetooth LE) Connectivity

ModusToolbox ソフトウェアを初めて使用する場合は、**ModusToolbox ソフトウェアのホームページ**を参照してください。 一部の PSoC 6 MCU デバイスは PSoC Creator でサポートされません。この場合、ModusToolbox ソフトウェアを使用する必要があります。ModusToolbox ソフトウェアを使用する場合は、デュアル CPU を搭載した PSoC 6 MCU デバイスに基づく設計では、バージョン 2.2 以降であることを確認してください。

PSoC Creator を初めて使用する場合は、**PSoC Creator のホームページ**を参照してください。 PSoC 6 MCU ベースの設計には PSoC Creator バージョン 4.3 以降を使用してください。

はじめに

本アプリケーション ノートの最初の節では、デュアル CPU MCU の一般的なコンセプトと、それらを PSoC 6 MCU に実装方法について説明します。PSoC 6 デュアル CPU MCU のための ModusToolbox ソフトウェアまたは PSoC Creator プロジェクトの作成概要にスキップするためには、**PSoC 6 MCU デュアル CPU での開発プロセス**を参照してください。

2 一般的なデュアル CPU のコンセプト

デュアル CPU MCU のファームウェア開発プロセスは、1 つではなく 2 つの CPU のコードを記述する点を除いてシングル CPU MCU の場合と同じです。また、プロセッサ間通信の必要性も考慮する必要があります。

- **パフォーマンス**: 2 つの CPU を持つ主な利点は、本質的に CPU 性能と帯域幅を増やすことです。PSoC 6 MCU では、その増加した帯域幅はシングル CPU MCU と同程度の価格で提供されます。増加した帯域幅の使用方法については、アプリケーションが実行するタスクに依存します。
- **シングル タスクの場合**: アプリケーションが大規模で複雑な場合を除いてシングル タスク アプリケーションはデュアル CPU MCU には適しません。ただし、PSoC 6 MCU では 1 つの CPU でタスクを実行し、もう片方の CPU をスリープさせて消費電力を削減できます。
- **デュアル タスクの場合**: 最も有用で、タスクごとに CPU を割り当てます。より大きなコンピューティング要件のタスクを高性能 CPU に割当てます (例えば PSoC 6 MCU の Cortex-M4 (CM4) CPU)。
- **マルチ タスクの場合**: 各タスクを CPU に割り当てます。各 CPU では、タイムリーに各タスクを実行するためのメソッドを含む必要があります。
- **RTOS**: 複雑なマルチタスク システムは、リアルタイム オペレーティング システム (RTOS) によって管理できます。基本的に RTOS は、タスク優先順位やタスクがイベントを待つかどうかによって、各タスクにいくつかの CPU サイクルを割り当てます。CPU にタスクを割り当てることで、効果的に自身を実行できます。デュアル CPU RTOS アーキテクチャの例は以下のとおりです。
  - 各 CPU は所有の RTOS とタスクの独自セットを持ちます。各 RTOS には、他の CPU との通信を管理するタスクを持ちます。
  - 1 つの CPU だけが RTOS と複数のタスクを持ちます。他の CPU は、指定するタスクを実行するよう知らされるまでアイドル状態になります。それからウェイクアップし、タスクを実行してその結果を最初の CPU に戻します。例として、PSoC 6 MCU の低性能 CPU CM0+は、PSoC 6 MCU の高性能 CPU CM4 を使用して、必要なときに高い計算要件のタスクを実行できます。
- **消費電力**: デュアル CPU システムでは、ファームウェアは CPU の起動と停止を行い、消費電力を調整します。前の例では、消費電力を減らすために、高性能 CPU は、負荷が高い計算動作が必要となるまでスリープ状態になります。
- **デバッグ**: 同時に 2 つのコード分をデバッグするのは複雑な作業です。通常は一方の CPU コードをデバッグしてから、もう一方の CPU コードをデバッグします。さらにオシロスコープやロジック アナライザなどのデバイスは、CPU 間通信を監視するのに使用できます。

3 PSoC 6 MCU デュアル CPU アーキテクチャ

ページ 2 の **Figure 1** に、PSoC 6 MCU のデュアル CPU アーキテクチャの概要を示します。(PSoC 6 MCU の詳細ブロック図については、デバイスのデータシートまたは **AN221774** を参照してください。) ここではデュアル CPU に関連する特定の機能やその他の詳細について説明します。詳細は **Arm documentation sets for Cortex-M4** と **Cortex-M0+**および PSoC デバイスの**テクニカル リファレンス マニュアル (TRM)**を参照してください。

- **CPU:** 両方の CPU (Cortex-M4 と Cortex-M0+) は 32 ビットです。CM4 は最大 150MHz で動作し、浮動小数点ユニット (FPU) を備えます。CM0+は最大 100MHz で動作します。
  CM4 はメイン CPU です。割込み応答時間が短く、コード密度が高く、そしてスループットが高く設計されています。CM0+ CPU はセカンダリです。PSoC 6 MCU でシステム コールとデバイス レベルのセキュリティ、安全性、および保護機能を実装するために使用されます。また、CM0+は Bluetooth LE 通信や CapSense などの機能にも推奨されます。
- **パフォーマンス:** 通常、CM0+は CM4 よりも低速なクロック速度動作です。CM0+命令セットは、CM4 命令セットよりも制限されます。したがって、CM0+に機能を実装するためにより多くのサイクルが必要となり、サイクル時間が長くなります。どの CPU にタスクを割当てるかを決めるときは、これを考慮してください。
- **セキュリティ**: PSoC 6 MCU は、いくつかのセキュリティ機能を備えます。詳細は TRM を参照してください。セキュリティ要件を満たすために、CM0+は「セキュア CPU」として使用されます。これには、信頼できるエンティティとみなされ、サイプレスのシステム コードとアプリケーション コードを実行します。システムおよびセキュリティ タスクに CM0+の使用では、ユーザー アプリケーションへのその可用性が制限される恐れがあります。セキュア システムの詳細は **AN221111 - Creating a Secure System** を参照してください。
  デバイス システム コールはいずれかの CPU によって起動されますが、常に CM0+によって実行されます。
- **起動 シーケンス:** デバイスがリセットされた後、CM0+のみが動作し、CM4 はリセット状態に保持されます。CM0+は、SROM コード、FlashBoot およびセキュア イメージを含むサイプレス システムとセキュリティ コードを最初に実行します。これらのコード モジュールの詳細は、アーキテクチャ **TRM** の Boot Code 章を参照してください。
  CM0+がシステム コードとセキュリティ コードを実行した後、アプリケーション コードを実行します。その時、CM0+は CM4 のリセットを解除し、CM4 はそのアプリケーション コードの実行を開始します。PSoC Creator は、CM4 のリセットを解除するために **CM0+の main() にコードを自動生成**します。ModusToolbox ソフトウェアは、CM0+のデフォルトで空のプロジェクトを作成します。
- **プロセッサ間通信 (IPC):** IPC は CPU 間の通信と動作同期のために機能します。IPC ハードウェアは、IPC チャネル機能および IPC 割込み用レジスタ構造を含みます。IPC チャネル レジスタは、相互排除 (mutex) ロックおよび解放メカニズムおよび CPU 間のメッセージングを実装します。IPC 割込みレジスタは、メッセージング イベント、ロック イベントおよびリリース イベントで両方の CPU に割込みを生成します
- **割込み:** 各 SPU はそれぞれ独自の割込みセットを持ちます。すべてのペリフェラル割込みラインは、特定の CM4 割込み入力に配線されています。ペリフェラル割込みは CM0+の 8 個の割込み入力の限定されたセット (CY8C61x7、CY8C62x7、および CY8C63x7 の 32 の割込み入力) にも多重化されます。**割込み割当ての検討事項**を参照してください。
- **消費電力モード:** PSoC 6 MCU には、システム全体または単一の CPU に影響を与える可能性があるいくつかの電力モードがあります。 Arm の定義に従って、CPU の電力モードはアクティブ, スリープ, およびディープスリープです。デバイスシステムの電力モードは、LP, ULP, ディープスリープ, およびハイバネートです。

- システム低電力(LP)モードは、リセット後のデバイスのデフォルトの動作モードです。最高のパフォーマンスを発揮します。システム LP モードでは、CPU は Arm 定義のモードのいずれでも動作します。
- システムの超低消費電力(ULP)モードは LP モードと同じで、より低いシステム電流を達成するために性能のトレードオフがあります。このトレードオフはコア動作電圧を下げるため、動作クロック周波数を下げ、高周波クロックソースを制限する必要があります。 システム ULP モードでは、CPU は Arm 定義モードのいずれでも動作します。
- システムディープスリープモードでは、すべての高速クロックソースはオフです。 これにより、両方の CPU が停止し、高速周辺機器が使用できなくなります。 ただし、ファームウェアによって設定され有効になっている場合、低速クロックソースとペリフェラルは引き続き動作します。 これらの周辺機器からの割込みにより、デバイスはシステム LP または ULP モードに戻り、1 つ以上の CPU がアクティブモードにウェイクアップします。 各 CPU には、CPU をウェイクアップするためのウェイクアップ割込みコントローラー(WIC)があります。
- システムハイバネートモードは、デバイスの最低電力モードです。休止状態になる可能性のあるアプリケーションを対象とします。デバイスは、ハイバネートからのウェイクアップ時にリセットを受けます。**起動 シーケンス**を参照してください。
- CPU アクティブモードでは、CPU がコードを実行し、すべてのロジックとメモリに電力が供給されます。 デバイスはシステム LP または ULP モードである必要があります。
- CPU スリープモードでは、CPU クロックはオフになり、CPU はコードの実行を停止します。デバイスはシステム LP または ULP モードである必要があります。
- CPU ディープスリープモードでは、CPU はデバイスにシステムディープスリープモードへの移行を要求します。デバイスの準備が整うと、システムディープスリープモードに入ります。PSoC 6 MCU では、システムがディープスリープに移行する前に両方の CPU が CPU ディープスリープに入る必要があります。1 つの CPU だけが CPU ディープスリープモードに入った場合、システムは LP または ULP モードのままです。

PSoC 6 MCU の消費電力モード詳細は **AN219528 - PSoC 6 MCU Low-Power Modes and Power Reduction Techniques** を参照してください。

- **デバッグ**: PSoC 6 MCU は、デバイス プログラミングおよびデバッグのインターフェースとして動作するデバッグ アクセス ポート (DAP) を備えます。外部プログラマまたはデバッガ (「ホスト」) は、デバイスのシリアル ワイヤ デバッグ (SWD) またはジョイント テスト アクション グループ (JTAG) インターフェースを介して DAP と通信します。DAP の使用で (デバイスのセキュリティ制限に従う)、ホストはデバイスのメモリとペリフェラルおよび両方の CPU のレジスタにアクセスできます。
  各 CPU は、次のようないくつかのデバッグ機能とトレース機能を提供します。
  - CM4 はシングル ワイヤ出力 (SWO) ピン経由で、6 つのハードウェア ブレークポイント、4 つのウォッチ ポイント、4 ビット組込みトレース マクロセル (ETM)、シリアル ワイヤ ビューア (SWV) および printf()スタイルのデバッグをサポートします。
  - CM0+は 4 つのハードウェア ブレークポイントと 2 つのウォッチ ポイントをサポートし、4KB 専用 RAM を備えたマイクロ トレース バッファ (MTB) をサポートします。

PSoC 6 MCU は、両方の CPU の同期デバッグとトレースのための **Embedded Cross Trigger** もあります。

ModusToolbox ソフトウェアといくつかのサードパーティ製 IDE はデュアル CPU デバッグをサポートします。デバッグに関する考慮事項を参照してください。PSoC Creator は一度に 1 つの CPU (CM4 または CM0+) のデバッグに対応します。

4 PSoC 6 MCU デュアル CPU での開発プロセス

ここでは、PSoC 6 MCU デュアル CPU デバイスに固有の開発側面についてのみ説明します。PSoC 6 MCU, ModusToolbox ソフトウェア, または PSoC Creator の詳細については、以下の 1 つないしそれ以上を参照ください。

- **AN221774**, Getting Started with PSoC 6 MCU
- **AN210781,** Getting Started with PSoC 6 MCU with Bluetooth Low Energy (Bluetooth LE) Connectivity
- **ModusToolbox ソフトウェアホームページ**。 一部の PSoC 6 MCU デバイスは PSoC Creator によってサポートされません。この場合、ModusToolbox ソフトウェアを使用する必要があります。ModusToolbox ソフトウェアを使用する場合は、デュアル CPU を搭載した PSoC 6 MCU デバイスベースの設計で、バージョン 2.2 以降であることを確認してください。
- **PSoC Creator ホームページ**。 PSoC 6 MCU ベースの設計には PSoC Creator バージョン 4.3 以降を使用してください。

4.1 ModusToolbox ソフトウェアの説明

PSoC 6 MCU デュアル CPU デバイス用の ModusToolbox ソフトウェアアプリケーション開発は、ModusToolbox ソフトウェアでサポートされる他のデバイス用のそれと似ています。デフォルトでは、ModusToolbox ソフトウェアに付属するボードサポートパッケージ (BSP) は、いくつかのビルド済み CM0+アプリケーションイメージを提供します。ここでは、ビルド済みのイメージの代わりに独自の CM0+アプリケーションを作成する方法について説明します。

デュアル CPU ワークスペースを作成するには、CM0+ CPU 用と CM4 CPU 用の 2 つの新しいアプリケーションを作成する必要があります。または、デュアル CPU サンプルコードに基づいて新しいプロジェクトも作成できます。このサンプルコードでは、両方のアプリケーションが既にセットアップされています。開始点として「dual-cpu」を含むリポジトリ名を持つ任意のサンプルコードを使用できます。**こちら**のリストを参照してください。

次のセクションでは、デュアル CPU ワークスペースを作成する方法を段階的に示します。

各セクションでは、ModusToolbox ソフトウェアで異なるツールを使用する 2 つの方法について説明します。

- ModusToolbox 用の Eclipse IDE
- コマンドラインインターフェース (CLI)

4.1.1 CM0+ CPU アプリケーションの作成

4.1.1.1 ModusToolbox 用の Eclipse IDE の場合

1. クイックパネルで **New Application** をクリックしてください。次に、**Figure 2** に示すように、PSoC 6 BSP の 1 つを選択してください。

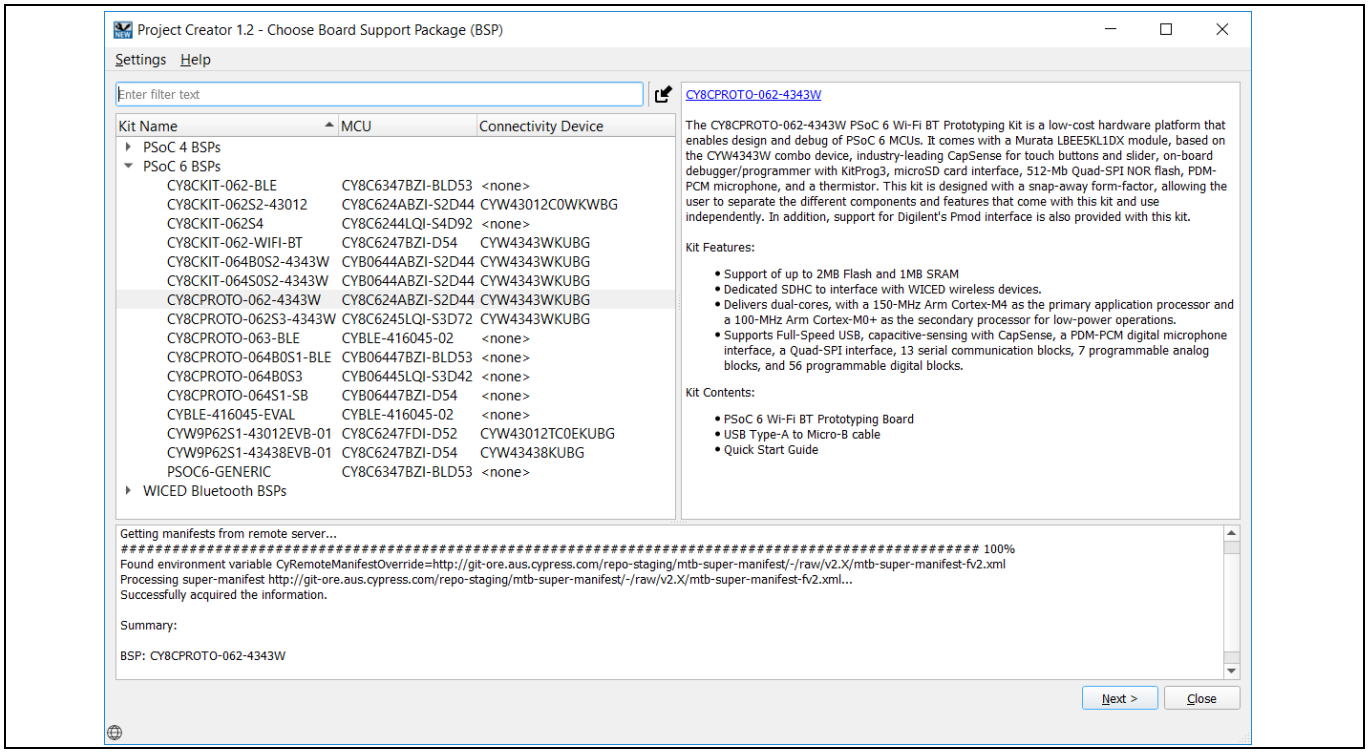

Figure 2　　ModusToolbox ソフトウェアでのボードサポートパッケージの選択

2. **Next** をクリックして、**Empty PSoC6 App** を選択してください。**Figure 3** に示すように、**New Application Name** で「cm0p_app」に設定してください。New Application ダイアログに入力すると、選択したワークスペースに新しいアプリケーションが作成されます。

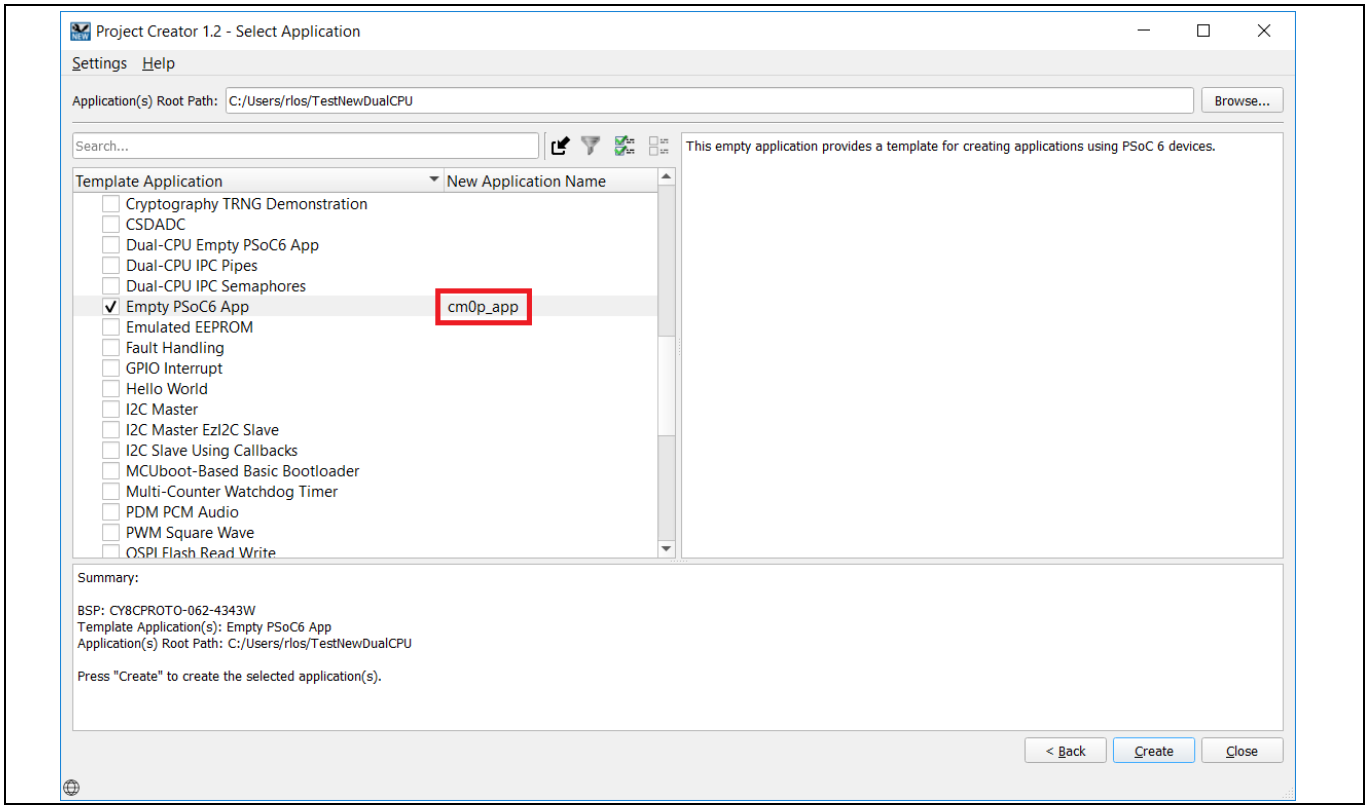

Figure 3 CM0+ CPU アプリケーションの作成

4.1.1.2 CLI の場合

1. CLI ターミナルを開き、プロジェクト作成用の ModusToolbox ソフトウェアインストールフォルダ (*<install_path>ModusToolbox/tools_<version>/project-creator/*) に移動してください。
2. `project-creator-cli` コマンドを実行して、新しいプロジェクトを作成してください。

例:

```
project-creator-cli \
        --board-id CY8CPROTO-062-4343W \
        --app-id mtb-example-psoc6-empty-app \
        --user-app-name cm0p_app \
        --target-dir “C:/workspace_directory”
```

または、同じフォルダで project-creator ツールを実行し、**ModusToolbox 用の Eclipse IDE の場合**で説明したのと同じ手順でも実行できます。

詳細については、**ModusToolbox User Guide** (セクション 2.3) を参照してください。

4.1.1.3 最終変更

アプリケーションが作成されたら (使用する方法に関係なく)、Makefile ファイルを開き、次の変更を加えてください。

- 新しい行を作成して、CORE を CM0+に設定してください。デフォルトでは、make ファイルはアプリケーションターゲットが CM4 であると想定します。この行を追加すると、アプリケーションターゲットは強制的に CM0+になります。
  `CORE=CM0P`
- 必要に応じて、APPNAME 変数を **cy_m0p_image** に変更してください。これは、リンカースクリプトで使用されるデフォルトの名前です。
  `APPNAME=cy_m0p_image`

CM0+ CPU のコードを作成する場合、サイプレス HAL を使用できないため、PDL API のみを使用してください。 main.c ファイルを変更して、HAL への参照をすべて削除してください。必要に応じて、BSP への参照もすべて削除してください。

例:

```
#include "cy_pdl.h"
#include "cyhal.h"
#include "cybsp.h"

int main(void)
{
    cy_rslt_t result;

    /* Initialize the device and board peripherals */
    result = cybsp_init() ;
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }

    enable_irq();

    Cy_SysEnableCM4(CY_CORTEX_M4_APPL_ADDR);

    for (;;)
    {
        Cy_SysPm_DeepSleep(CY_SYSPM_WAIT_FOR_INTERRUPT);
    }
}
```

サイプレス HAL は CM0+で実行するように設計されていないため、正しくコンパイルされません。また、1 つの CPU のみ、優先的には CM4 CPU で BSP を初期化することを推奨します。

また CM0+アプリケーションは、デフォルトで design.modus で生成されたファイルを参照しません。これらのファイルを CM0+アプリケーションの一部として含める場合は、CM0+ Makefile の COMPONENTS 変数に次を追加してください。

```
COMPONENTS=BSP_DESIGN_MODUS
```

または、カスタム design.modus を使用している場合は以下です。

```
COMPONENTS=CUSTOM_DESIGN_MODUS
```

4.1.2 CM4 CPU アプリケーションの作成

4.1.2.1 ModusToolbox 用の Eclipse IDE の場合

CM0+ CPU 用に選択したものと同じ BSP に基づいて新しいアプリケーションを作成してください。任意のアプリケーションテンプレートを選択できます。アプリケーションに「cm4_app」という名前を付けて、アプリケーションの作成を完了してください。

4.1.2.2 CLI の場合

「cm4_app」アプリケーションを作成するために、project-creator-cli を再度実行してください。以下は例です。

```
project-creator-cli \
        --board-id CY8CPROTO-062-4343W \
        --app-id mtb-example-psoc6-hello-world \
        --user-app-name cm4_app \
        --target-dir “C:/workspace_directory”
```

-app-id で任意のテンプレートアプリケーションを使用できることに注意してください。完全なリストを表示するには、**Cypress GitHub** リポジトリにアクセスしてください。

4.1.2.3 最終変更

アプリケーションが作成されたら、Makefile ファイルを開き、次の変更を加えてください。

- DISABLE_COMPONENT リストに「CM0P_SLEEP」を追加してください。別のビルド済みライブラリを使用している場合は、それに応じてこれを変更してください。
  `DISABLE_COMPONENTS=CM0P_SLEEP`
- 新しい行を作成して、cm0p_app への相対パスを依存関係として追加してください。
  `DEPENDENT_APP_PATHS=../cm0p_app`

手順を実行した後、**Figure 4** に示すように、Project Explorer ウィンドウ (ModusToolbox の Eclipse IDE のみ) でアプリケーションを表示できます。どちらのアプリケーションにも、独自のライブラリ、構成、および make ファイルのセットがあります。

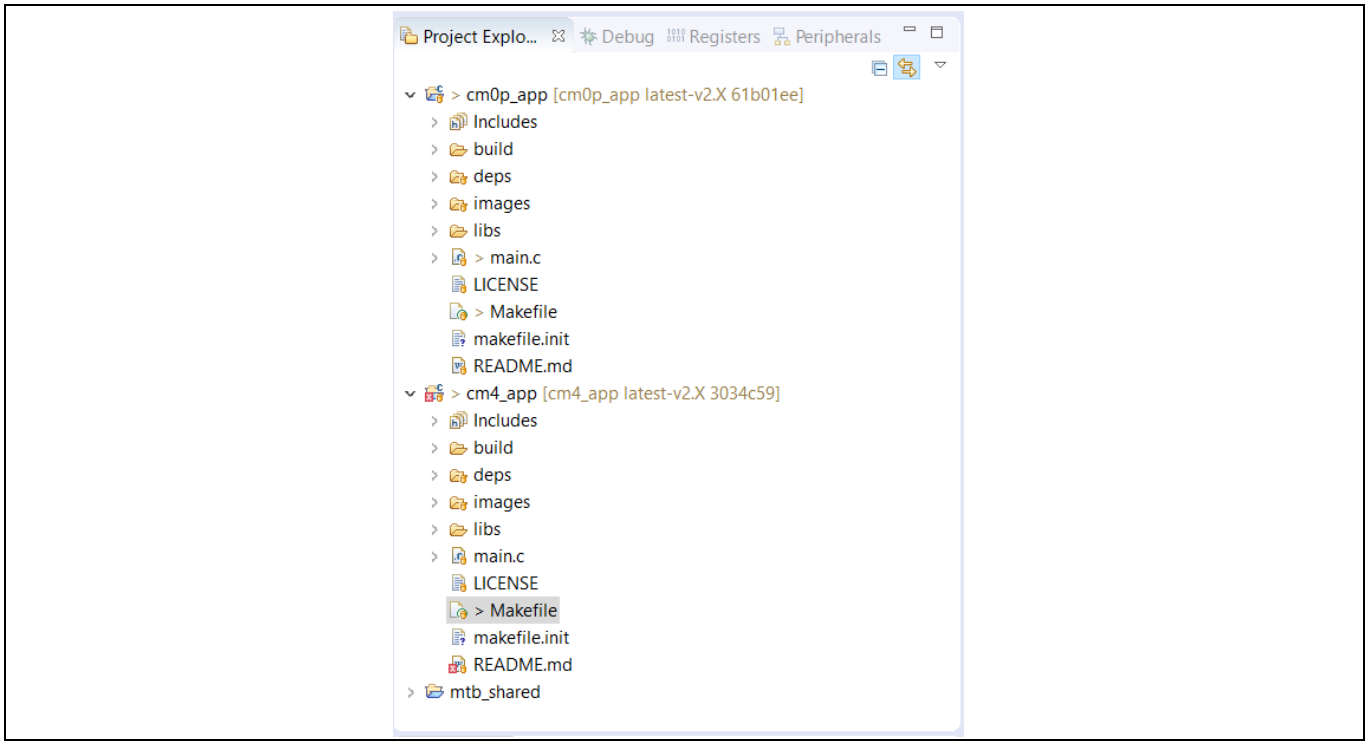

Figure 4 Project Explorer

CM4 アプリケーションは、依存する CM0+アプリケーションを自動的に構築します。アプリケーションを 2 回リンクします。1 回は CM0+イメージなしで、もう 1 回は CM0+イメージありです。デフォルトのイメージには、ビルド済みのイメージの 1 つを使用して CM4 アプリケーションをビルドする場合と同じように、CM0+イメージが含まれます。

ModusToolbox 用の Eclipse IDE では、アプリケーションをビルドまたはプログラミングするときに、アプリケーションを選択する必要があります。Project Explorer で、アプリケーションに属する任意のファイルまたはフォルダをクリックできます。Quick Panel は、選択したアプリケーションに基づく起動スクリプトを含むリンクを更新します。

CPU の 1 つでのみハードウェアを初期化することを推奨します。したがって、design.modus ファイルの 1 つだけを編集する必要があります。

mtb_shared フォルダには、両方のアプリケーションで共有される可能性のあるライブラリと BSP ファイルが含まれます。各アプリケーションには、独自の deps フォルダがあります。アプリケーションが使用する可能性のある依存関係ライブラリをこのフォルダに追加します。同じライブラリが両方のアプリケーションで表示される場合があることに注意してください。

4.1.3 リンカースクリプトのカスタマイズ

CM0+および CM4 アプリケーションには、それぞれ独自のリンカースクリプトがあります。デフォルトでは、CM0+ CPU は 8192 [0x2000]バイトのフラッシュと 8192 [0x2000]バイトの SRAM のみを消費します。CM0+アプリケーションがより多くのメモリを必要とする場合は、両方のリンカースクリプトを変更する必要があります。次の手順に変更点を示します。

1. プロジェクトのルートディレクトリにカスタムリンカースクリプトを保存するための共有フォルダを作成してください。以下は例です。

shared / linker_script / COMPONENT_CM0P
shared / linker_script / COMPONENT_CM4

2. CM0+リンカースクリプト ファイルを以下のフォルダから「*shared / linker_script / COMPONENT_CM0P*」フォルダにコピーしてください。
   mtb_shared / TARGET_<BSP_NAME> / COMPONENT_CM0P / TOOLCHAIN_<TOOL_NAME> /

デフォルトでは、BSP は ARM (*.sct)、GCC_ARM (*.ld)、および IAR (*.icf) ツールチェーンをサポートします。 各ツールチェーンには、独自のリンカースクリプトがあります。

3. 必要に応じて FLASH と SRAM のサイズを編集してください。

ツールチェーン	変更する場所
ARM (*.sct)	`#define RAM_SIZE 0x00002000` `#define FLASH_SIZE 0x00002000`
GCC_ARM (*.ld)	`ram (rwx) : ORIGIN = 0x08000000, LENGTH = 0x2000` `flash (rx) : ORIGIN = 0x10000000, LENGTH = 0x2000`
IAR (*.icf)	`define symbol __ICFEDIT_region_IRAM1_end__ = 0x08001FFF;` `define symbol __ICFEDIT_region_IROM1_end__ = 0x10001FFF;`

4. CM4 リンカースクリプト ファイルを以下のフォルダから「shared / linker_script / COMPONENT_CM4」フォルダにコピーしてください。
   mtb_shared / TARGET_<BSP_NAME> / COMPONENT_CM4 / TOOLCHAIN_<TOOL_NAME> /
5. CM0+メモリサイズに基づいて値を編集してください。

ツールチェーン	変更する場所

ARM (*.sct)	`#define RAM_START 0x08002000`
	`#define RAM_SIZE 0x000FD800`
	`#define FLASH_CM0P_SIZE 0x2000`

GCC_ARM (*.ld)	`FLASH_CM0P_SIZE = 0x2000;` `ram (rwx) : ORIGIN = 0x08002000, LENGTH = 0xFD800`
IAR (*.icf)	`define symbol __ICFEDIT_region_IRAM1_start__ = 0x08002000;` `define symbol __ICFEDIT_region_IRAM1_end__ = 0x080FF7FF;`

6. CM0+アプリケーションの make ファイルで APPNAME に「cy_m0p_image」とは異なる名前を付けた場合、リンカースクリプト内のすべての参照を新しい名前に置き換える必要があります。
7. CM0+および CM4 Makefile で、カスタムリンカースクリプトを参照するために、以下の行を追加してください。
   LINKER_SCRIPT=../shared/linker_script/COMPONENT_$(CORE)/<FILENAME>.<EXTENSION>

手順 (1) と (2) で作成したリンカースクリプト フォルダにコピーしたファイルに基づいて、FILENAME と EXTENSION を置き換える必要があります。

8. CM0+ Makefile で、CM4 アプリケーションアドレス (**CY_CORTEX_M4_APPL_ADDR**) を DEFINES に追加してください。例えば、CM0+イメージのサイズが 0x8000 の場合は、次のように設定してください。
   DEFINES=CY_CORTEX_M4_APPL_ADDR=0x10008000

4.1.4 ライブラリと周辺機器の共有

前に説明したように、CM4 と CM0+は、アプリケーションで使用されるライブラリの一部を共有することも、共有しないこともできます。各アプリケーションの deps フォルダには、アプリケーションで使用されるライブラリのリストが含まれます。ModusToolbox は、Library Manager と呼ばれるライブラリ

と BSP を管理するためのツールを提供します。このツールは、ModusToolbox の Eclipse IDE、**Quick Panel** > **Tools** > **Library Manager** から起動できます。または、このフォルダ (<install_path>ModusToolbox/tools_<version>/library-manager/) にある **Library-manager** ツールも実行できます。

Figure 5 に示すように、BSP またはライブラリを選択すると、ツールはそれを共有するオプション (Shared 列) を提供します。

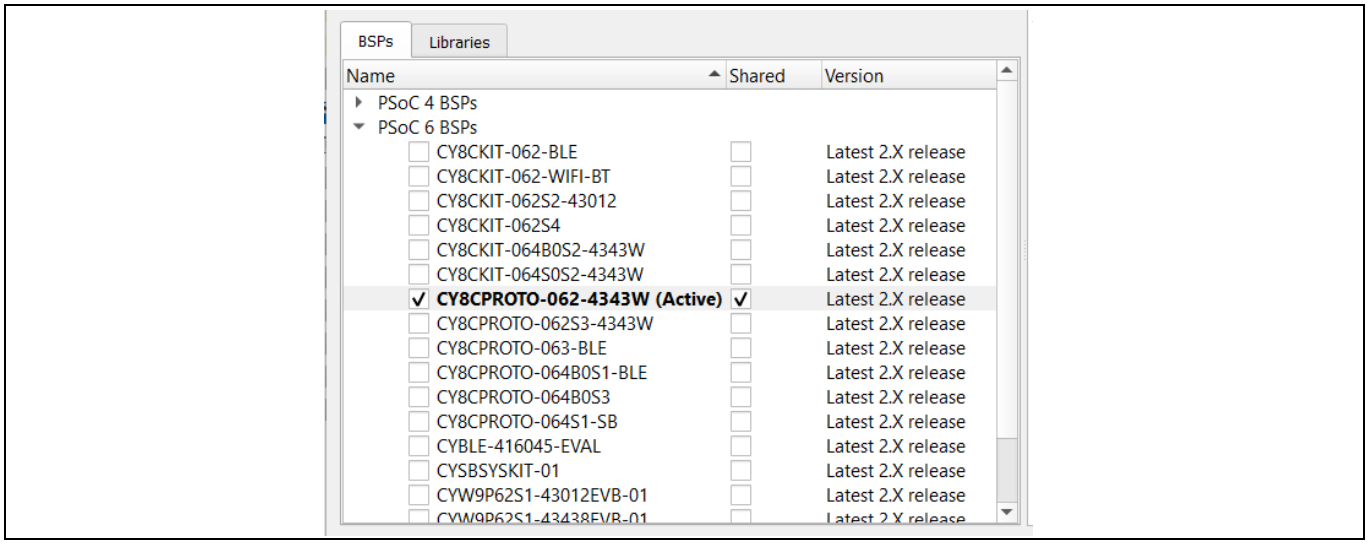

Figure 5 Library Manager

デュアル CPU アプリケーションでは、すべてのライブラリと BSP を共有することを推奨します。

周辺機器の構成に関しては、デュアル CPU アプリケーションで design.modus を 1 つだけにし、`init_cycfg_all()`または `cybsp_init()`、優先的には CM4 を呼び出す CPU を 1 つだけにすることを推奨します。周辺機器が CPU の 1 つだけで使用される場合は、単一の CPU アプリケーションで使用するのと同じように使用できます。

周辺機器が共有される場合は、いずれかの CPU でのみ初期化し、何らかの mutex または同期メカニズムを使用して、両方の CPU が同時に使用しないようにする必要があります。CM0+は CM4 によって作成された HAL オブジェクトにアクセスできないため、共有ペリフェラルに HAL を使用することは推奨しません。

4.2 PSoC Creator の説明

PSoC 6 MCU デュアル CPU デバイス用の PSoC Creator プロジェクト開発は、PSoC Creator でサポートされる他のデバイス用のそれと似ています。新しいプロジェクトを作成するには、**File** > **New** > **Project** を選択します。**Figure 6** のような **Create Project** ダイアログが表示されます。

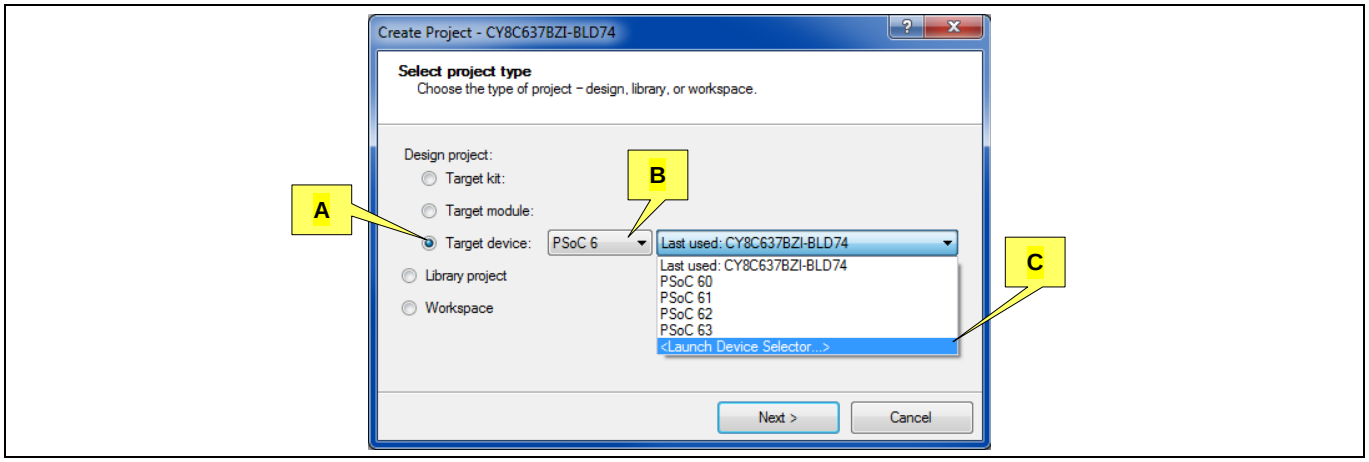

Figure 6　　PSoC Creator の Create Project ダイアログ

Target Device (A)、**PSoC 6** (B) を選択してください。プルダウン リスト (C) で、**<Launch Device Selector ...>**を選択し、PSoC 6 デバイスのリストを表示してください。

Figure 7 に、Device Selector ダイアログを示します。デュアル CPU デバイスのリストを表示するためには、**CPU** カテゴリ (D) をクリックし、**CortexM0p, CortexM4** を選択してください。

PSoC 6 Bluetooth LE Pioneer Kit **CY8CKIT-062-BLE** では、PSoC 6 MCU デュアル CPU デバイスの製品番号は CY8C6347BZI-BLD53 です。PSoC 6 WiFi-BT パイオニアキット **CY8CKIT-062-WiFi-BT** では、PSoC 6 MCU デュアル CPU デバイスの部品番号は CY8C6247BZI-D54 です。

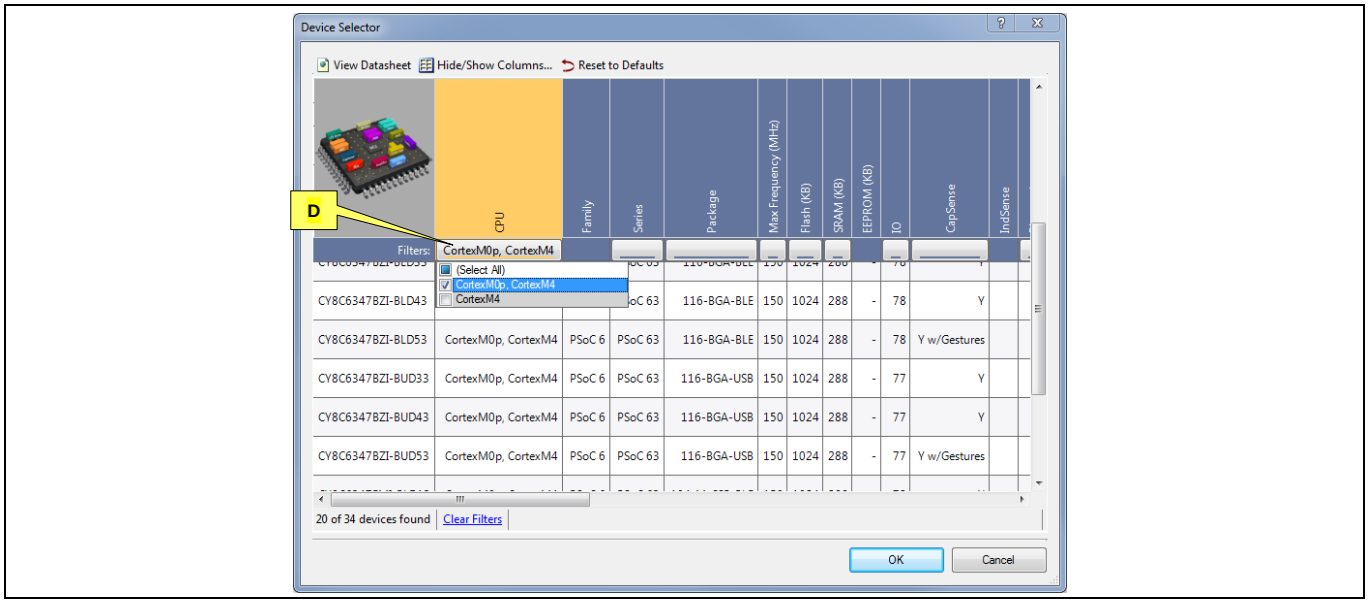

Figure 7　　PSoC Creator の Device Selector ダイアログ

PSoC 6 MCU 製品を選択した後、プロジェクト作成プロセスの残りの部分は他のデバイスと同じです。残りの Create Project ダイアログをクリックしてください。PSoC Creator がプロジェクトを作成します。

初期プロジェクト ウィンドウ レイアウト (**Figure 8**) は、デュアル CPU デバイス用の以下の機能を備えた **Workspace Explorer** ウィンドウが含まれます。

- 各 CPU に個別の main.c ファイル (main_cm0p.c と main_cm4.c)。フォルダ CM0p (Core 0) と CM4 (Core 1) のソースコード ファイルは、それぞれの CPU の別々のバイナリにコンパイルされます。
- Shared Files フォルダ。このフォルダ内のソースコード ファイルは、両方のバイナリにコンパイルされます。

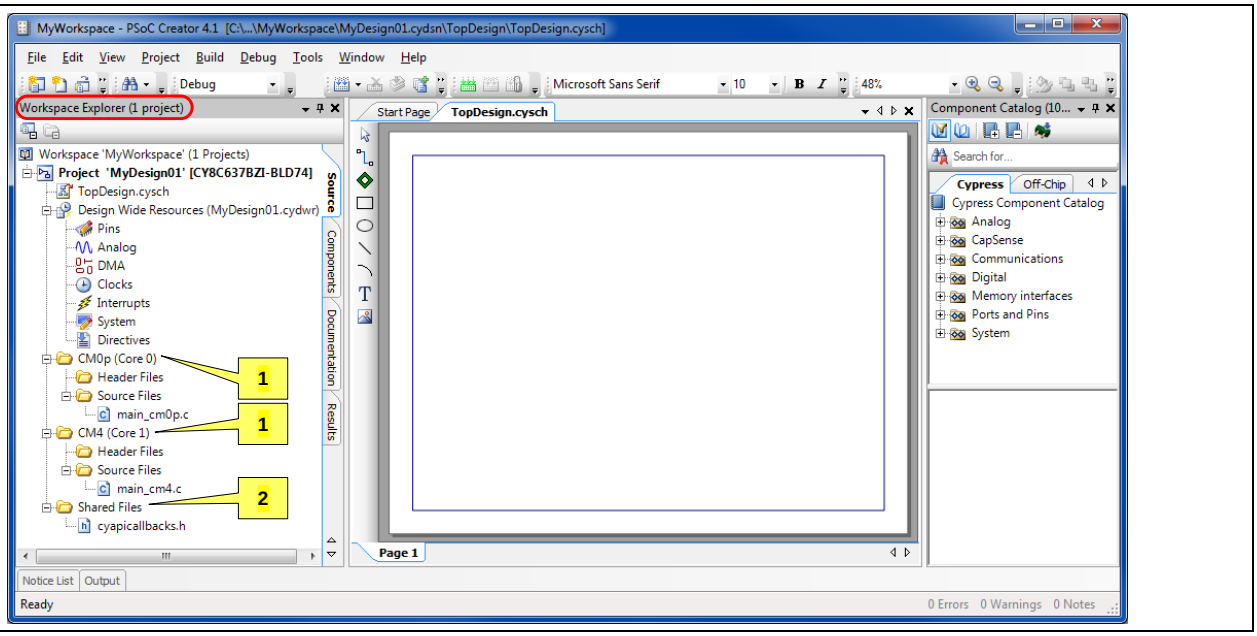

Figure 8　　デュアル CPU デバイスの PSoC Creator 初期プロジェクト レイアウト

初期プロジェクト レイアウトには、TopDesign ハードウェア回路図と関連するコンポーネント カタログ ウィンドウも含まれます。

プロジェクトを作成した後、コンポーネントを回路図にドラッグして設定し、配線して、ハードウェア設計を実装してください。

回路図デザインが完了すると、**Build** > **Generate Application** を選択します。これにより、**Figure 9** に示すように、既存フォルダだけでなく、新しいフォルダ Generated Source にいくつかのシステム ソース コード ファイルとフォルダが作成されます。

生成されたソースには、回路図上の各コンポーネントのドライバとサイプレス ペリフェラル ドライバ ライブラリ (PDL) が含まれます。PDL は、デバイス ヘッダ ファイル、スタートアップ コード、ペリフェラルドライバを統合するソフトウェア開発キット (SDK) です。ペリフェラル ドライバは、ハードウェア機能を使いやすい一連の API に抽象化します。

PDL の詳細情報は **Help** > **Documentation** > **Peripheral Driver Library** を選択します。また、各コンポーネントはそのドライバ API を記述したデータシートがあります。コンポーネントを右クリックし、**Open Datasheet ...**を選択します。

PSoC Creator は、いくつかの他のファイルとフォルダを作成し、既存のフォルダ CM0p (Core 0), CM4 (Core 1) および Shared Files に配置します。これらのファイルは、一般に PSoC Creator のほか、他の IDE に設定、スタートアップおよびリンクのオプションを提供します。これらのファイルの詳細は、PSoC Creator のヘルプ記事 Generated Files (PSoC 6) を参照してください。

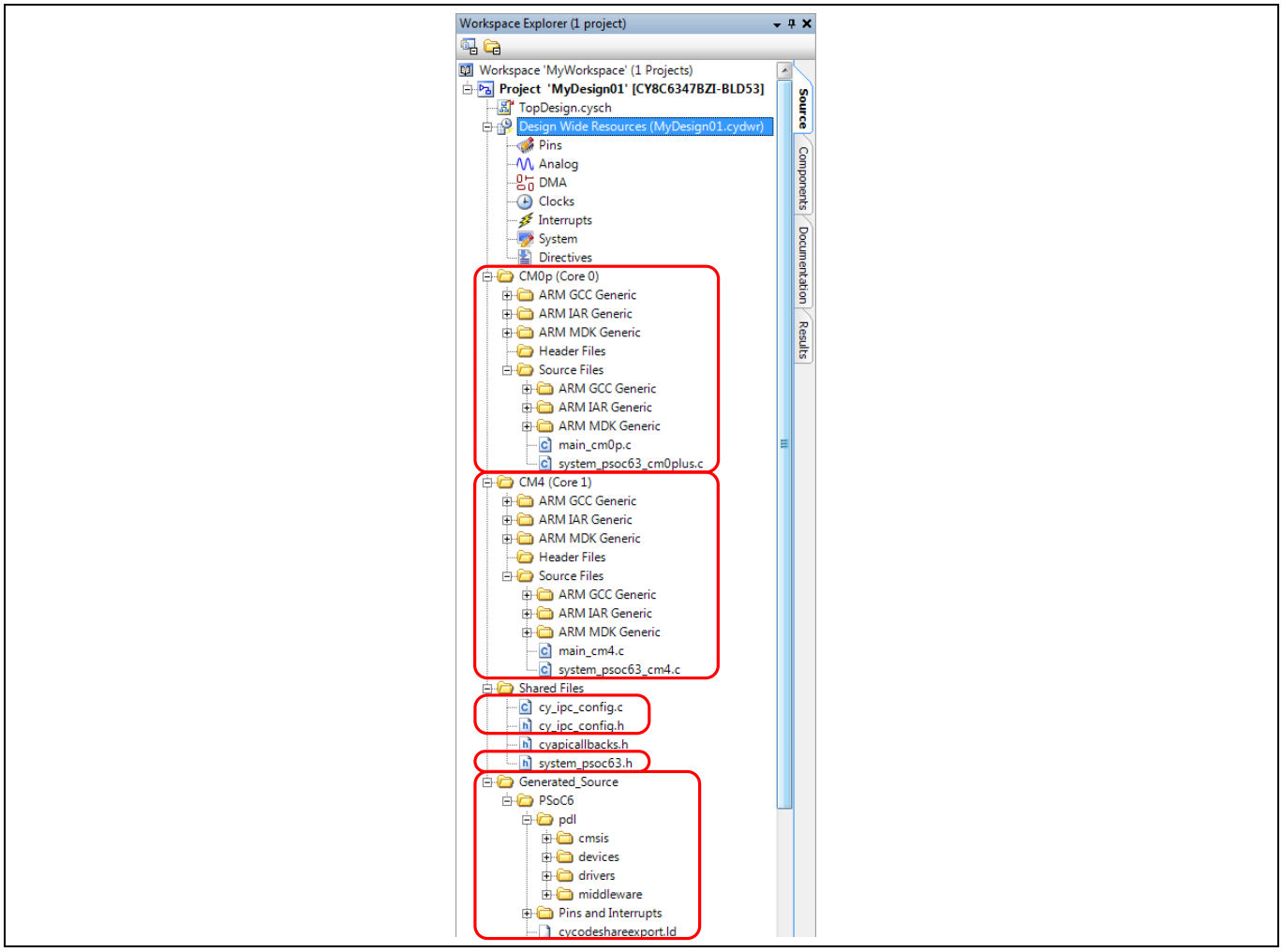

Figure 9　　生成されたソースをプロジェクトに追加

4.3　　リソース割当ての検討事項

すべての PDL ドライバソースと他の API ファイルは両方の CPU で利用可能です。CPU のコードが生成されたソースファイルの API 要素を参照する場合、そのファイルはその CPU のバイナリにコンパイルされます。同じファイルを両方のバイナリにコンパイルできます。詳細はサンプルコードの **CE216795 - PSoC 6 MCU Dual-CPU Basics** を参照してください。

同じソースファイルが両方のバイナリにコンパイルする場合、そのファイル内の関数が両方のバイナリに複製されます。 コピーが別々のビルドとバイナリに入っても、場合によっては、両方の CPU によって同時に実行される関数を検討すると便利です。

前述のように、ペリフェラルが両方の CPU からアクセスされる可能性があります。例えば、両方の CPU が同じ UART を介してデータを送信することがあります。一般に、PDL ドライバ関数は「CPU-safe」であり、つまり、これらの機能は両方の CPU で同時に実行できます。ただし、各 CPU にリソースを割当てることについて設計上の決定をする必要があります。これを行うには以下の 2 つの方法があります。

- **リソースを 1 つの CPU 専用にします。**目的の CPU のファームウェアでのみリソースを使用するためのコードを含めます。 また、PSoC Creator を使用する場合、**Figure 10** に示すように、リソースを「所有する」CPU をプロジェクトの回路図で指定することを推奨します。

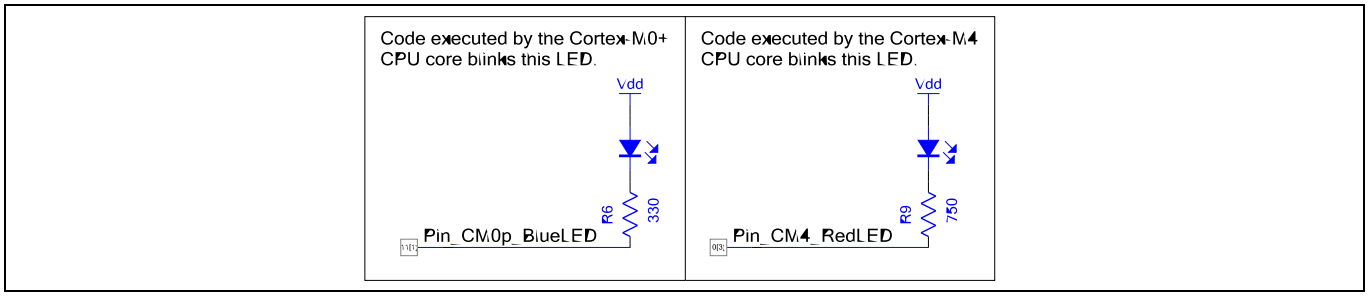

Figure 10　　個別ピン コンポーネントを制御するデュアル CPU の PSoC Creator プロジェクト回路図

- **CPU 間でリソースを共有します。**サンプルコード **CE216795** は、CPU 間でメモリ共有するために、PSoC 6 MCU の IPC ブロックを使用し、相互排除を実装する方法を示します。同じテクニックを使用して、UART などのペリフェラル リソースを共有します。

CPU のバイナリに割り当てられているフラッシュと SRAM は、通常、他の CPU のものとは別のものです。カスタム セクションとセクション配置が CPU のリンカ スクリプトで定義される場合、セクションが重複しないようにする必要があります。メモリを共有する別の方法は、各 CPU に同じアドレスを持つカスタム セクションを定義することです。

Note: *両方の CPU が同じ GPIO ポートのピンを制御する場合は、ピン出力を変更するために次の API 関数のみを使用してください。`GPIO_Write()`、`GPIO_Set()`、`GPIO_Clr()`、および `GPIO_Inv()`。詳細については、PDL のドキュメントの GPIO API リファレンスを参照してください。*

4.4　割込み割当ての検討事項

デュアル CPU 設計の重要な検討事項は、割込みの割当てとその処理です。前述に示したように、すべてのデバイス割込みは CM4 で利用可能であり、割込みサブセットはマルチプレクサを経由し、CM0+にルーティングされます。設計では、どの CPU が各割込みを処理するかを決める必要があります。

詳細については、アプリケーションノート **AN217666、PSoC 6 MCU Interrupts** を参照してください。

ModusToolbox ソフトウェア： ModusToolbox ソフトウェアでは、割込みはプログラムで割り当てられます。現時点では、GUI は対応しません。必要なコードの例は AN217666 にあります。また、PDL のドキュメントの「Cypress Bluetooth LE ミドルウェアライブラリ」セクションの「Configure BLESS Interrupt」サブセクションにあります。このサブセクションのコードは、PSoC 6 MCU BLE サブシステム (BLESS) 割込みを CM0+または CM4 に割り当てる方法を示します。コードは他の割込みソースに対応するように簡単に修正できます。サイプレス HAL は CM4 でのみサポートされるため、HAL ドライバに必要なすべての割込みは CM4 に割り当てられます。

PSoC Creator： PSoC Creator では、サンプルデザインで割込みを割り当てます。**Figure 11** は、2 つの割込みを伴う設計を示します。1 つは割込みコンポーネント MyPWM_Int に接続された PWM コンポーネントからのもの。もう 1 つは I2C コンポーネントからのものです。

Design Wide Resources ウィンドウ (ファイルタイプ .cydwr) で、Interrupts タブを選択して、デザイン内のすべての割込みを確認します(**Figure 12** を参照)。

この例では、I2C コンポーネントに割込みが埋め込まれます。その割込みは **Figure 11** の回路図には示されません。デザイン全体のリソースウィンドウに MyI2C_SCB_IRQ として表示されます。

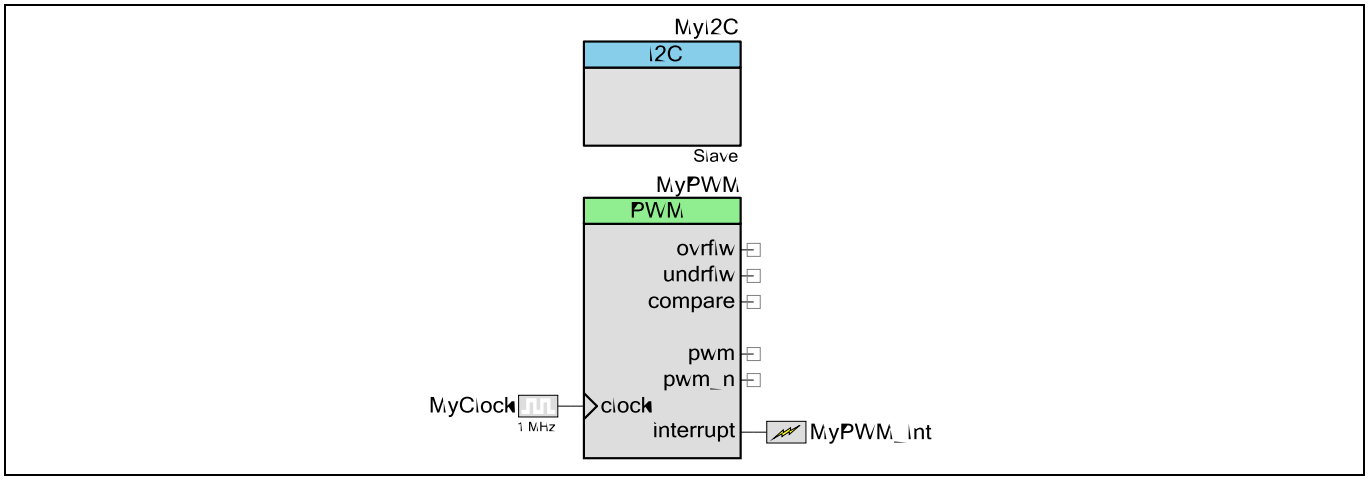

Figure 11　　2 つの割込みを持つ回路設計の例

ARM CM0+ Enable および **ARM CM4 Enable** の列のチェック ボックスをオンまたはオフにして、それぞれの CPU に割込みを割当てます。

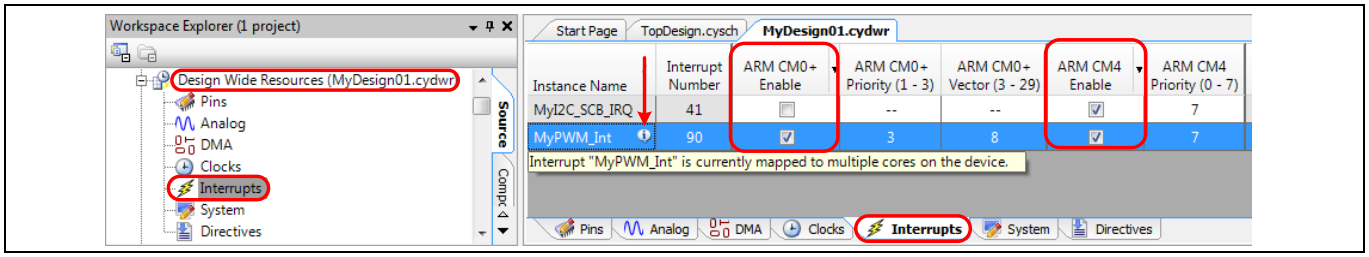

Figure 12　　CPU への割込み割当て

各ペリフェラル割込みは、CM4 に配線接続されるため、プロジェクトをビルドする時に、**Interrupt Number** が PSoC Creator によって自動で割り当てられます。割込みはマルチプレクサ経由で CM0+にルーティングされるため、割込みごとに **ARM CM0+ Vector** を選択できます。

Note: *両方の CPU に 1 つの割込みが割り当てられる場合、警告シンボルとツールチップが表示されます。これは一般的に推奨しませんが、スリープ モードから一方または両方の CPU を復帰させるために 1 つの割込みを使用できます。*

4.5　デバッグに関する考慮事項

Keil µVision や IAR Embedded Workbench などのサードパーティ IDE は、デュアル CPU デバッグに対応します。ModusToolbox および PSoC Creator 用の IDE は、一度に 1 つの CPU (CM4 または CM0+のいずれか) のデバッグに対応します。

4.5.1　ModusToolbox の手順

ModusToolbox ソフトウェア用の Eclipse IDE は、一度に 1 つの CPU (CM4 または CM0+のいずれか) のデバッグに対応します。詳細な手順については、**ModusToolbox User Guide** を開き、サブセクション **PSoC 6 MCU Programming/Debugging** を参照してください。

4.5.2 PSoC Creator の手順

PSoC Creator は一度に 1 つの CPU のみのデバッグに対応します。PSoC 6 MCU でデバッグセッションを開始する前に、**Figure 13** に示すように、希望するデバッグターゲットを選択してください (**Debug** > **Select Debug Target**...)。希望の CPU を選択し、**OK** / **Connect** をクリックしてください。他の CPU をデバッグするには、デバッガを終了してから、その CPU に接続して再入力する必要があります。

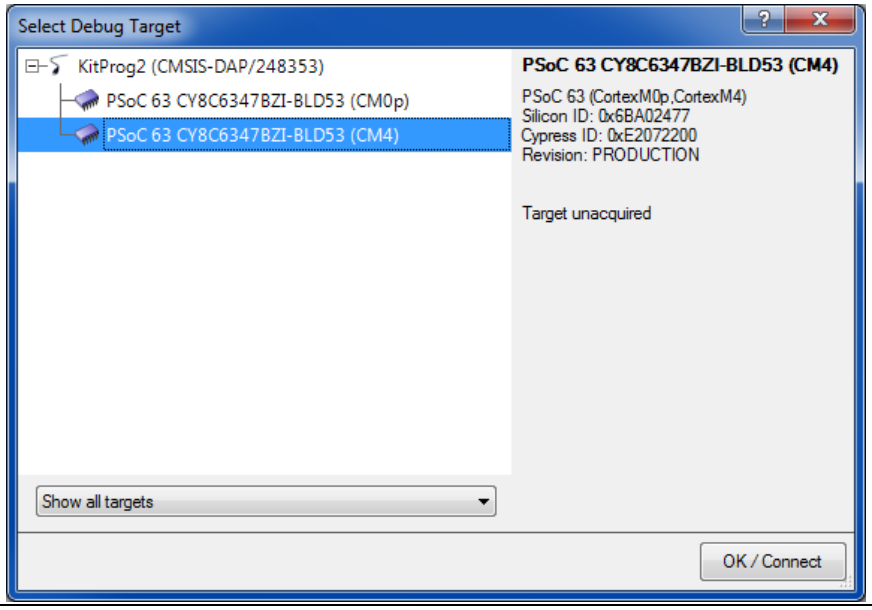

Figure 13 PSoC Creator のデバッグ用 CPU 選択

推奨: 最初に、CPU が互いに通信するコード部分を開発してデバッグします。その後、個々の CPU で実行されたコードを個別にデバッグできます。たとえば、**CE216795** の共有メモリプロジェクトが開発されたとき、CM0+が CM4 に初期メッセージを送信する部分が開発され、その後のコード部分が開発される前にデバッグされます。

µVision や IAR などの他の IDE を使用して、両方の CPU を同時にデバッグできます。そうするには、PSoC Creator プロジェクトを他の IDE にエクスポートする必要があります。PSoC Creator は、このトピックをヘルプ記事 Integrating into 3rd Party IDEs, PSoC 6 Designs に記載しています。ヘルプ記事の説明を確認してください。一般的な手順は次のセクションにまとめられています。

4.5.3 他の IDE の手順

4.5.3.1 ModusToolbox ソフトウェアからのエクスポート

ModusToolbox ソフトウェアには、make ファイルを介した次の IDE のエクスポートメカニズムがあります。

1. IAR Embedded Workbench: `make ewarm8`
2. Keil µVision: `make uvision5`
3. Visual Studio Code: `make vscode`

これらのツールを使用してデバッグをエクスポートおよび設定する方法の詳細については、**ModusToolbox User Guide** の「IDE へのエクスポート」の章を参照してください。

これらの IDE はすべて、デュアル CPU デバッグに対応します。ただし、ModusToolbox を介してエクスポートされたプロジェクトで機能するのは Keil µVision デュアル CPU デバッグのみです。

ModusToolbox ユーザーガイドの説明は、CM4 をプログラム/デバッグするように設計されます。同じ手順が CM0+にも適用されますが、いくつかの変更があります。使用している IDE に応じて、次の追加手順を実行します。

1. **CM0 +を使用した VisualStudio Code**

CM0+プロジェクトに対して `make vscode` コマンドを実行した後、".Vscode"という新しいフォルダが作成されます。このフォルダで、launch.json ファイルを開き、次の変更を加えます。

a) CM4 のすべてのインスタンス名を CM0+に変更

b) すべての `targetProcessor` インスタンスを (1 ではなく) 0 に設定

2. **CM0+を使用した Keil µVision**

CM4 のプロジェクトを作成するときと同じ手順に従います。デュアル CPU デバッグを実行するには、Keil µVision の 2 つのインスタンスを開きます。1 つは CM0+プロジェクトで、もう 1 つは CM4 プロジェクトで開きます。どちらの場合も、**Debug > Start/Stop Debug Session** に移動します。

3. **IAR Embedded Workbench**

ModusToolbox の IAR エクスポートメカニズムは、CM0+プロジェクトに対応しません。CM4 プロジェクトのみをエクスポートできます。

4.5.3.2 PSoC Creator からのエクスポート

PSoC Creator を使用する場合は、プロジェクトを Keil µVision または IAR Embedded Workbench にエクスポートして、デュアル CPU デバッグを行えます。そのための手順は次のとおりです。

1. PSoC Creator プロジェクトの設定

2. µVision プロジェクトの生成

3. µVision プロジェクトのデバッグ

4. IAR-EW プロジェクトの生成

5. IAR-EW プロジェクトのデバッグ

1. **PSoC Creator プロジェクトの設定**

Figure 14 に示すように、プロジェクトの Build Settings で **Target IDEs** 設定を更新します。

µVision の場合は、**CMSIS Pack: > Generate** を選択します。**Vendor**、**Pack**、および **Version** の各フィールドに、CMSIS パックの適切な識別テキストを入力します。

推奨: **Toolchain: > ARM MDK Generic** を選択してください。

IAR の場合は、**IAR EW-ARM: > Generate** を選択するだけです。 (高度なオプションである **Generate without copying PDL files** も利用できます。) IAR には独自のコンパイラがあるため (PSoC Creator では対応しません)、Toolchain の選択は関係ありません。

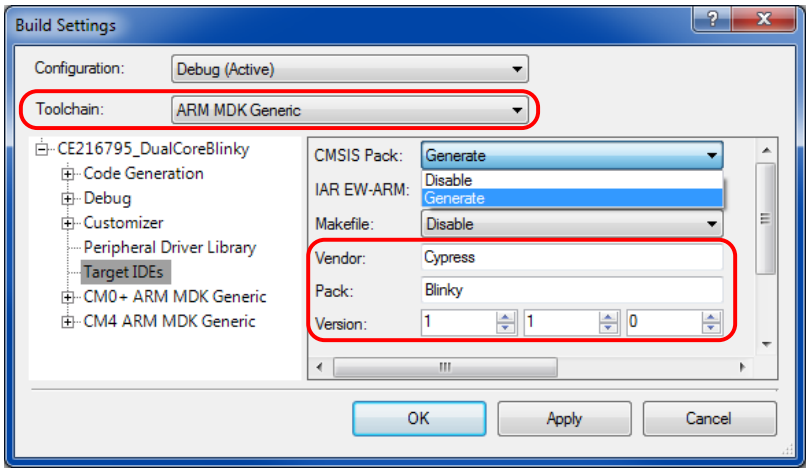

Figure 14　　ターゲット IDE のビルド設定

その後 PSoC Creator プロジェクトを通常の方法でビルドします。フォルダ Export が<project>.cydsn フォルダに作成されます。このフォルダは、選択した 1 つまたは複数の IDE にエクスポートするための関連ファイルを含みます。

µVision の場合、PSoC Creator プロジェクトがビルドされた後に、Export \ Pack フォルダで対応する.pack ファイルを見つけてください。**Figure 15** に示すように、ファイルをダブルクリックして µVision パックとしてインストールしてください。

Note: *このパックのインストールのために µVision Pack Installer Wizard File Import 機能を使用しないでください。*

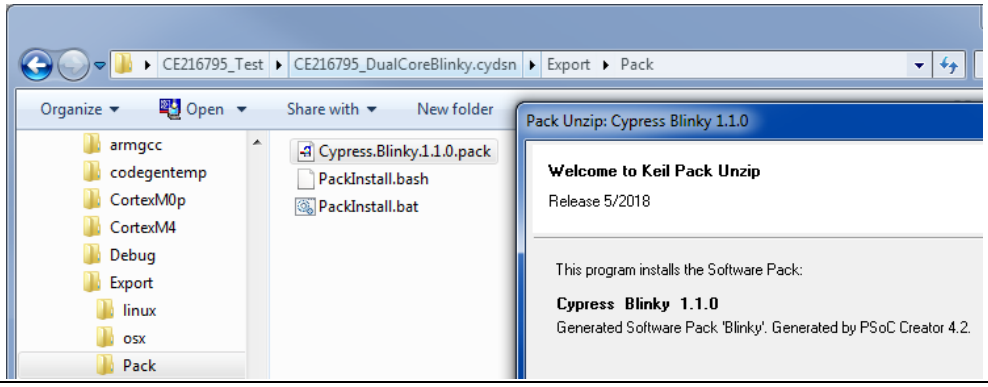

Figure 15　　PSoC Creator プロジェクトから µVision パックのインストール

Note: *PSoC Creator プロジェクトを更新する場合は、µVision パックのバージョン番号を変更し (**Figure 14** を参照)、新しいパックをインストールすることを検討してください。*

詳細については、**AN219434 - PSoC 6 MCU Importing Generated Code into an IDE** を参照してください。

2. µVision プロジェクトの生成

µVision の場合、2 つのプロジェクトを作成する必要があります。各 PSoC 6 MCU CPU (CM0+と CM4) に 1 つずつ。 以下を実施してください。

推奨: PSoC Creator の<project>.cydsn フォルダ内に新しいフォルダ (uVisionBuild など)を作成して、すべての µVision プロジェクトファイルを PSoC Creator ファイルとは別に保存します (これは **IAR の手順**とは異なります)。 そのフォルダ内に、**Figure 16** に示すように、CM4 オブジェクトファイル (ObjectsM4 など) 用の別の新しいフォルダを作成してください。

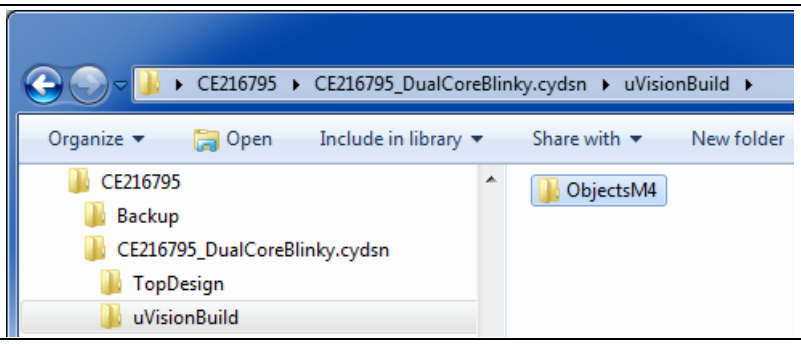

Figure 16　　µVision プロジェクトの新しいフォルダ

µVision 5.25 以降を開き、uVisionBuild フォルダに新しいプロジェクトを作成します (**Project > New µVision Project...**) 。

推奨: 元の PSoC Creator プロジェクト名とターゲット CPU に基づいてプロジェクトに名前を付けます。例えば、**CE216795** デュアル CPU 点滅プロジェクトの場合、**Figure 17** のように、CM0+ CPU 用の µVision プロジェクト BlinkyM0p を作成します。

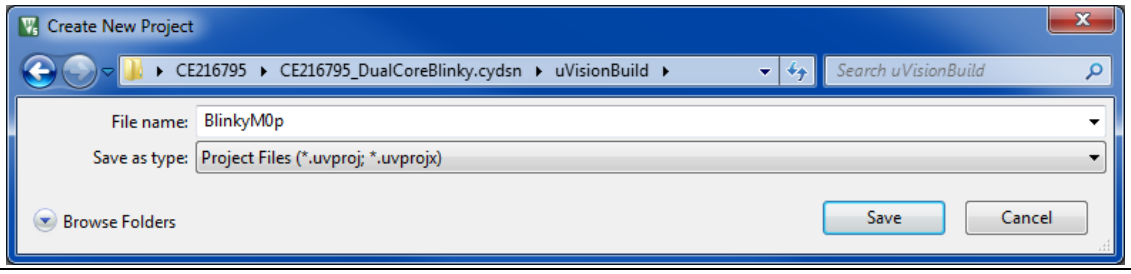

Figure 17　　CM0+用 µVision プロジェクトの生成

Save をクリックすると、**Select Device for Target 'Target 1'...**ダイアログボックスが表示されます。以前にインストールされたパック (**Figure 15**) で定義された 2 つの PSoC 6 MCU CPU が表示されます。 **Figure 18** に示すように、CM0+ CPU を選択してください。 **OK** をクリックしてください。

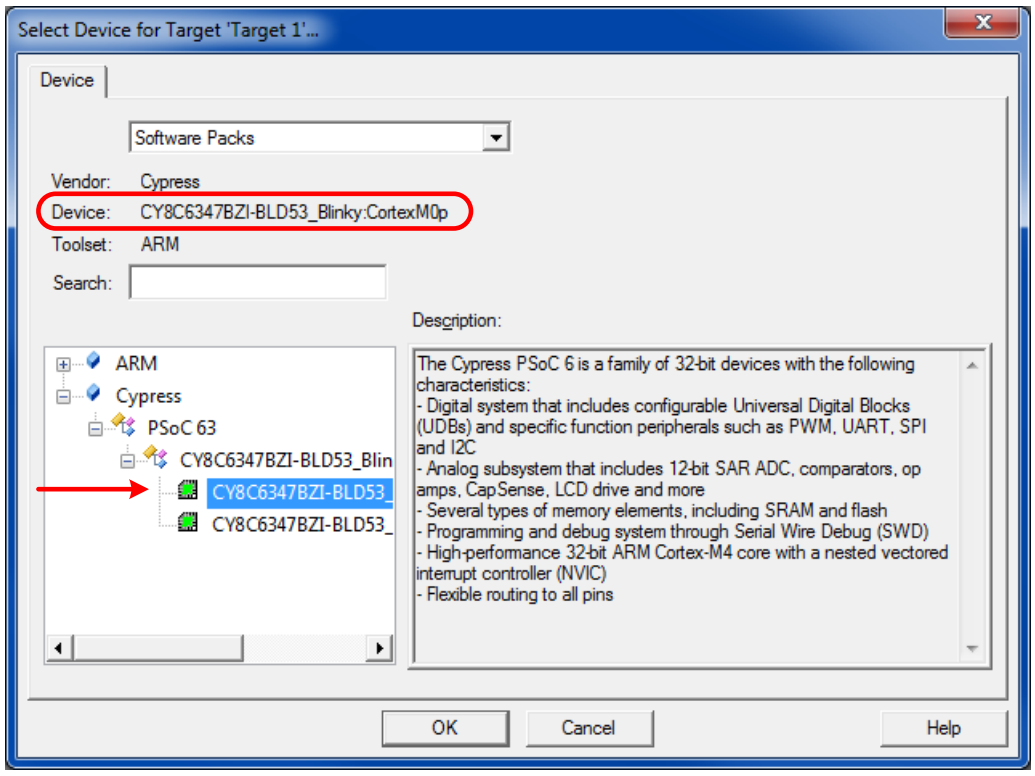

Figure 18　　プロジェクトデバイスとして CM0+を選択

次に、**Manage Run-Time Environment** ダイアログボックスが表示されます。**Select Packs** をクリックし、**Use latest versions of all installed Software Packs** のチェックを外します。**Figure 19** に示すように、PSoC Creator プロジェクトからパックを選択します。

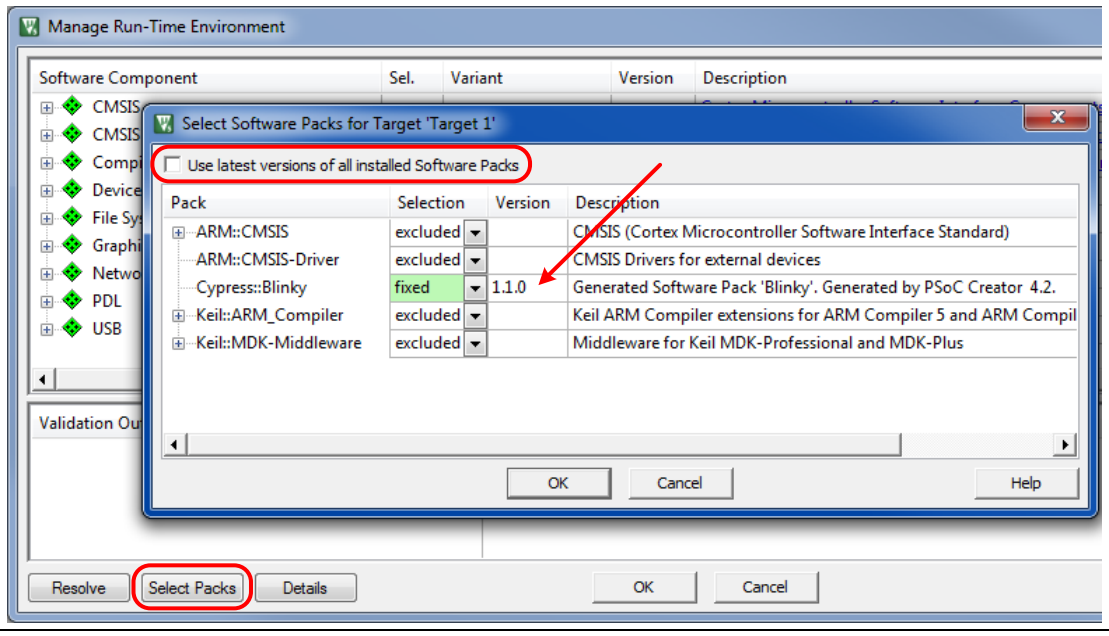

Figure 19　　PSoC Creator プロジェクトパックの選択

OK をクリックしてください。**Figure 20** に示すように、Manage Run-Time Environment ダイアログが変わります。 Device Startup と PDL Drivers を選択し、**OK** をクリックしてください。**Figure 21** に示すように、Target 1、Source Group 1、および Device startup と PDL ファイルでプロジェクトが作成されます。

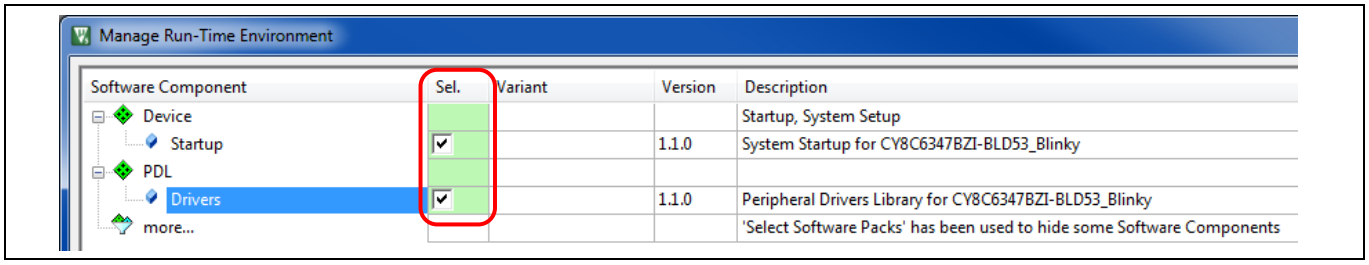

Figure 20　　パック開始と PDL ドライバーファイルの選択

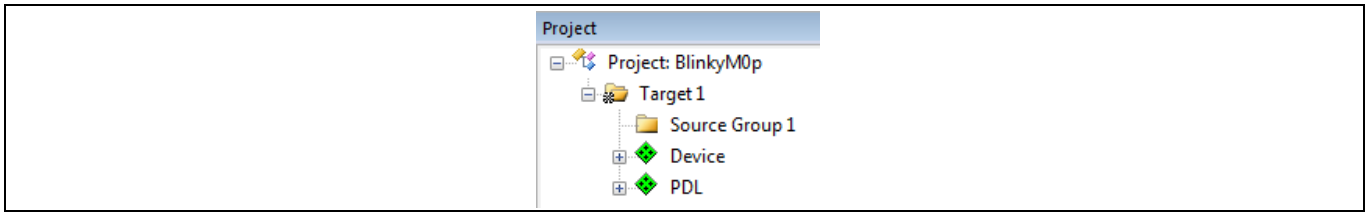

Figure 21　　初期プロジェクト生成

Source Group 1 を右クリックし、**Add Existing Files to Group 'Source Group 1'...**を選択してください。自身の PSoC Creator プロジェクトフォルダに移動し、**Figure 22** に示すように main_cm0p.c, cy_ipc_config.c, および他のシステム以外のすべての.c とプロジェクトに必要なアセンブラファイルを選択してください。.h ファイル, スタートアップファイル, または system .c ファイル, またはアセンブラファイルを追加する必要はありません。**Add** をクリックしてください。ファイルは µVision プロジェクトのソースグループに追加されます。 **Close** をクリックしてください。

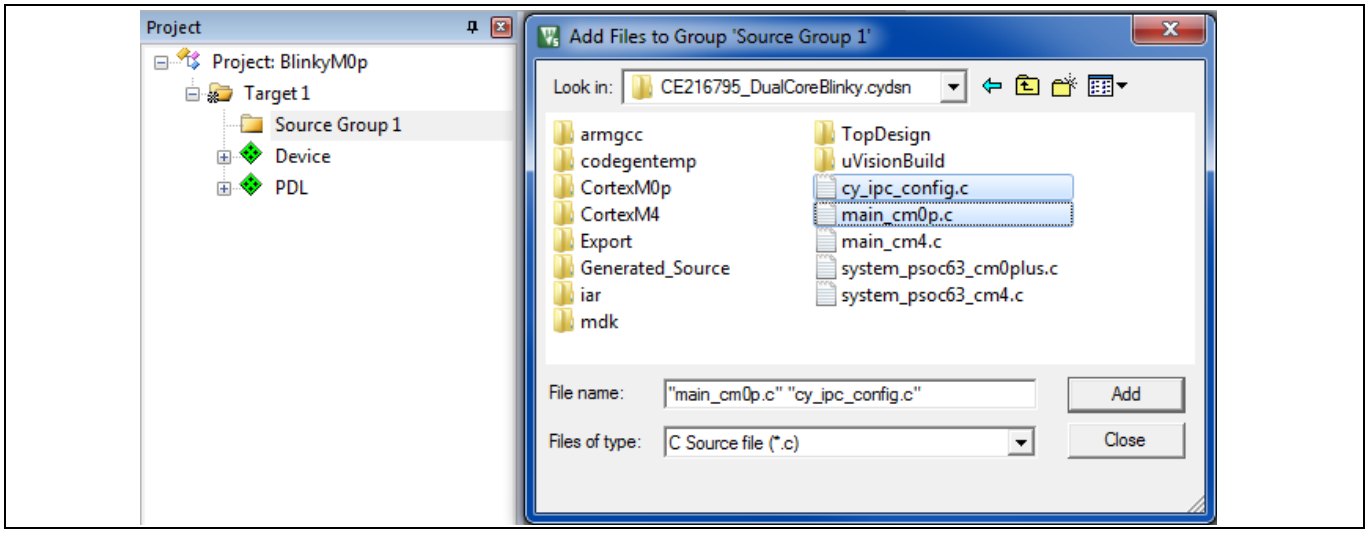

Figure 22　　PSoC Creator プロジェクト C ソースファイルのソースグループへの追加

プロジェクトが作成されたので、次にそのオプションを設定する必要があります。**Target 1** を右クリックし、**Options for Target 'Target 1'...**を選択してください。**Figure 23** に示すように、**Target** タブで、デバイス, CPU, IROM1, および IRAM1 が PSoC 6 MCU デバイスに対して正しいことを確認してください。Xtal (MHz) やオペレーティングシステムなどの他のフィールドの更新はオプションです。

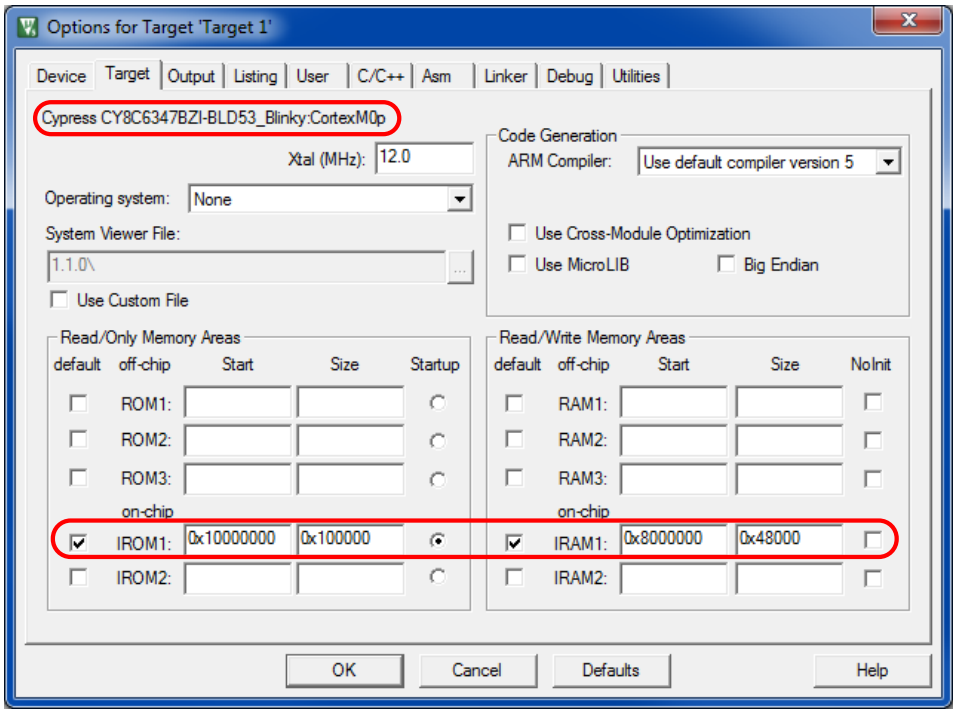

Figure 23　　プロジェクトターゲットオプション

User タブで、パックからビルド後の正しいバッチファイルが呼び出されていることを確認してください。**Figure 24** に示すように、**User Command** フィールドの上にカーソルを置き、postbuildCortexm0p.bat が呼び出されることを確認してください。 必要に応じて、他のビルド前およびビルド後のバッチファイルを追加し、他のオプションを選択してください。

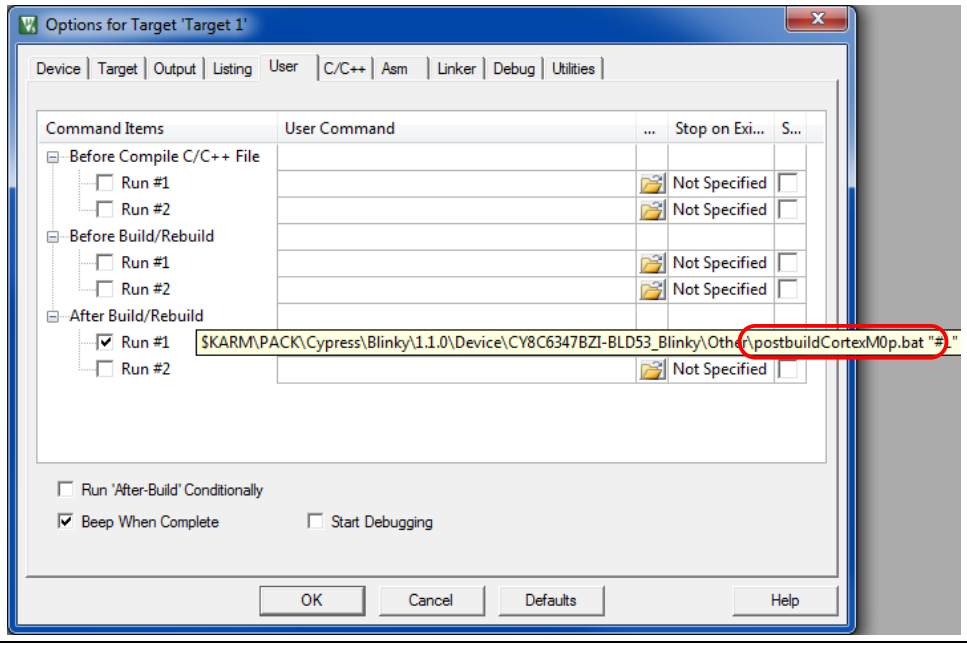

Figure 24　　プロジェクトユーザーオプション

Figure 25 に示すように、**C/C++**タブで **C99 mode** オプションがチェックされていることを確認してください。(PDL は C99 に基づいて開発されています。) PSoC Creator <project>.cydsn フォルダを **Include Paths** に追加します。これは PSoC Creator プロジェクト内の.h ファイルへのリンクを提供します。必要に応じて他のオプションとフィールドを更新してください。

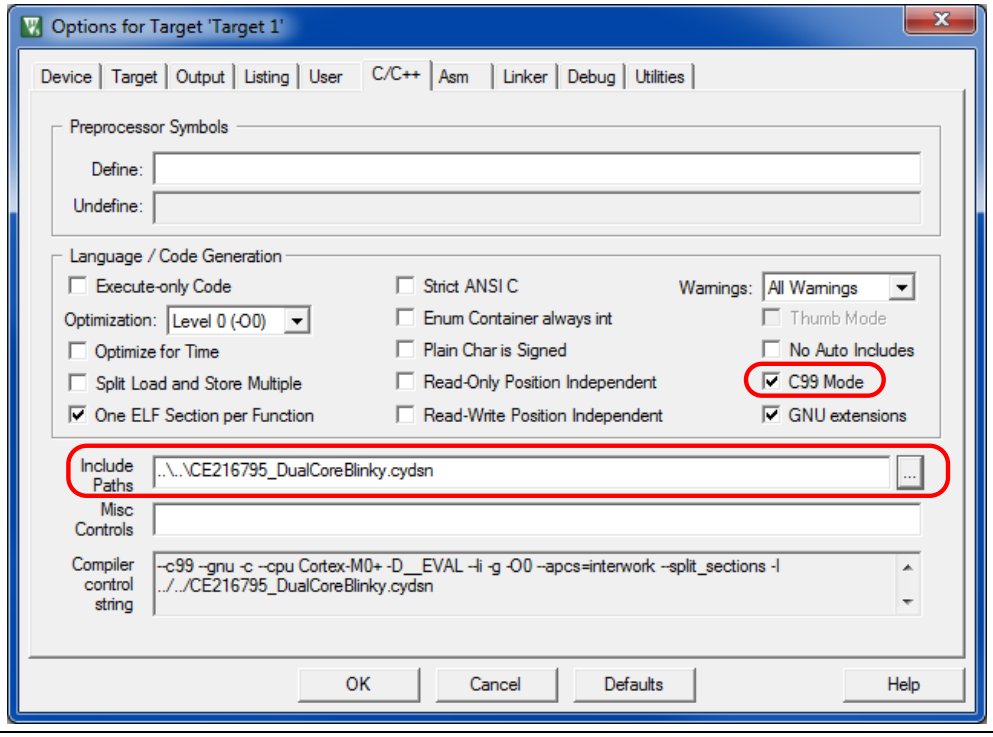

Figure 25　　プロジェクト C/C++オプション

Figure 26 に示すように、**Linker** タブで、**R/O Base** と **R/W Base** フィールドが PSoC 6 MCU デバイスに対して正しいことを確認してください。PSoC Creator プロジェクトフォルダから適切な **Scatter File** を選択してください。

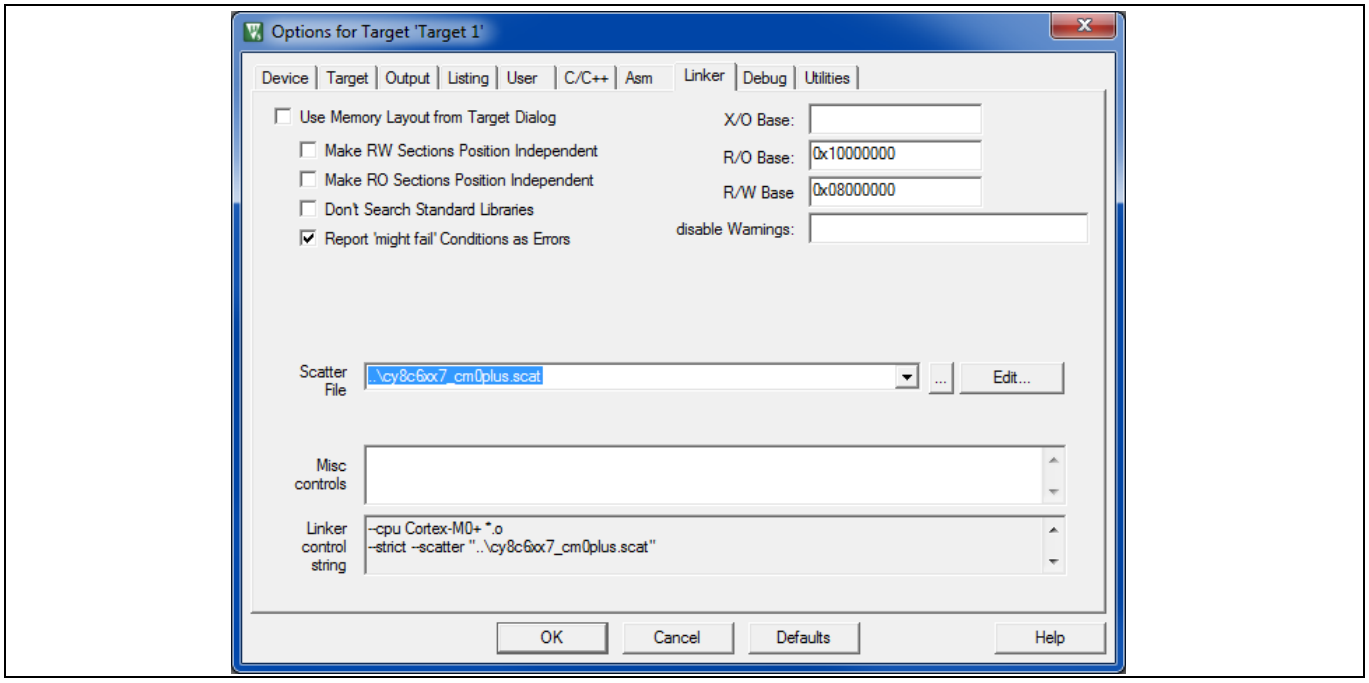

Figure 26　　プロジェクトリンカーオプション

CY8CKIT-062-BLE の USB ポートをコンピュータに接続してください。キットボタン SW3 を押して KitProg2 を CMSIS-DAP モードにします。詳細はキットガイドを参照してください。これにより、外部プローブを使用せずにデバッグできます。

Figure 27 に示すように、**Debug** タブで、**Use CMSIS-DAP Debugger** を選択してください。**Settings** をクリックし、**KitProg2 CMSIS-DAP** を選択して、他のすべての設定が表示されているデフォルトになっていることを確認してください。**OK** をクリックしてオプションダイアログに戻ってください。

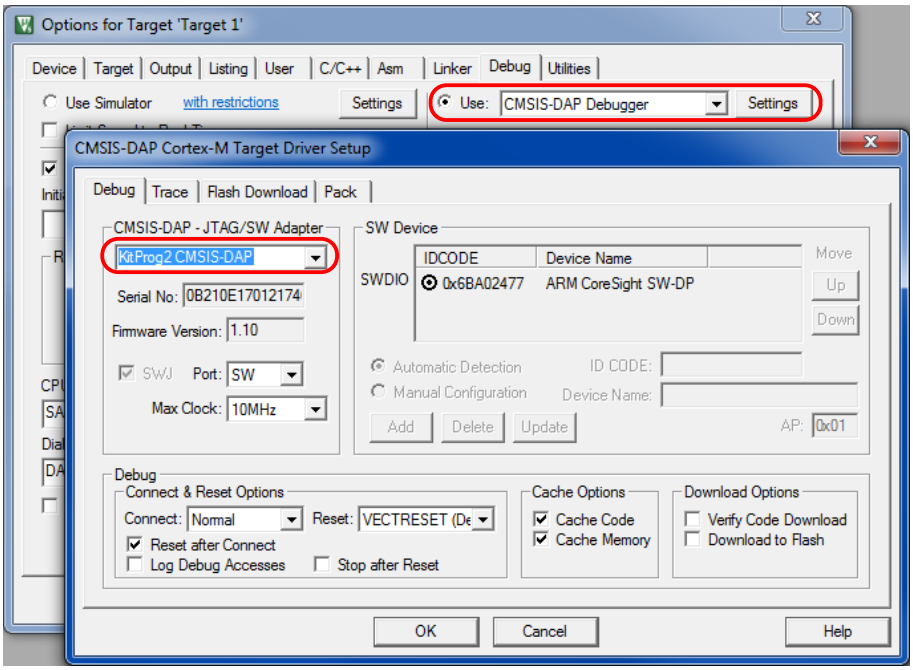

Figure 27　　プロジェクトデバッグオプション

Utilities タブで、Use Debug Driver がチェックされていることを確認し、**Update Target before Debugging** のチェックを外してください。**Figure 28** に示すように、**Settings** をクリックし、**Download Function** ボックスのチェックをすべて外してください。**Do not Erase** をクリックしてください。**OK** をクリックして **Options** ダイアログに戻ってください。 警告 “Nothing to do ... ” が表示されます。**OK** をクリックしてください。 アプリケーションは CM4 プロジェクトによってロードされます。**OK** をクリックしてオプション設定を保存して閉じてください。

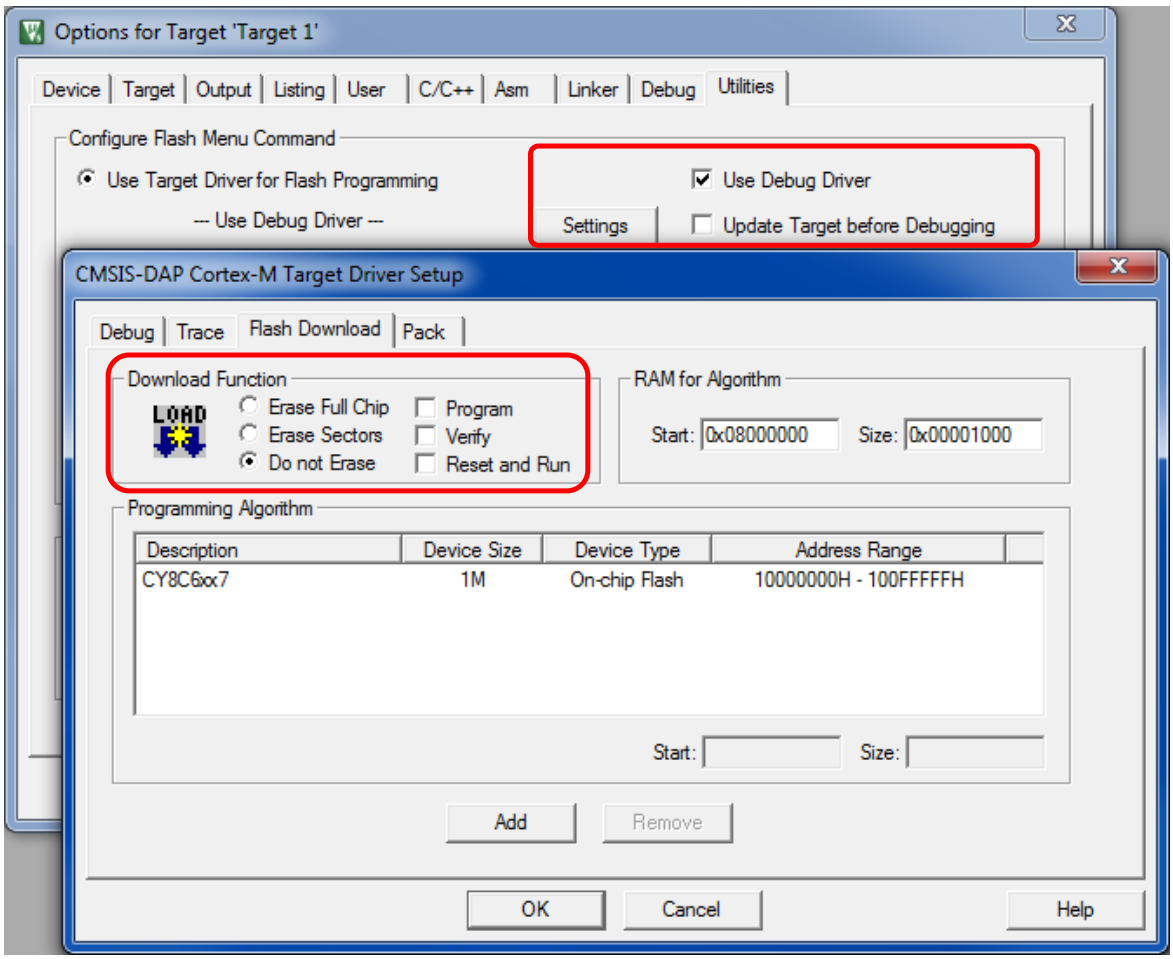

Figure 28　　プロジェクトユーティリティオプション

前のステップを繰り返して、CM4 用の 2 番目のプロジェクトを作成してください。

推奨: オリジナルの PSoC Creator プロジェクト名とターゲット CPU に基づいてプロジェクトに名前を付けてください。例えば、**CE216795** デュアル CPU 点滅プロジェクトの場合は、µVision プロジェクト BlinkyM4p を作成してください。**Figure 17** を参照してください。以下の違いを除いて、CM0+プロジェクトと同じ方法でプロジェクトを構成してください。

- CM4 プロジェクトは CM0+プロジェクトと同じフォルダになければなりません。この場合は uVisionBuild です。 **Figure 17** を参照してください。
- 以前にインストールしたパックから CM4 CPU を選択してください。**Figure 18** を参照してください。
- **Figure 22** に示すように、PSoC Creator プロジェクトフォルダに移動し、main_cm4.c、cy_ipc_config.c、およびプロジェクトに必要な他のすべてのシステム以外の.c ファイルとアセンブラファイルを選択してください。.h ファイル, スタートアップファイル, system .c ファイル, またはアセンブラファイルを追加する必要はありません。
- **Options** ダイアログの **Output** タブで、**Select Folder for Objects...**をクリックして、作成した ObjectsM4 フォルダを選択してください。**Figure 16** を参照してください。
- **Options** ダイアログの **C/C++**タブで、**Misc Controls** に`--fpu=fpv4-sp` を追加してください。**Figure 25** を参照してください。

- **Figure 29** に示すように、**Options** ダイアログの **Linker** タブで、"cm4_dual" スキャッタファイルを選択してください。CM4 プロジェクトは両方の CPU のコードを含みます。**Misc Controls** に `--fpu=fpv4-sp` を追加してください。

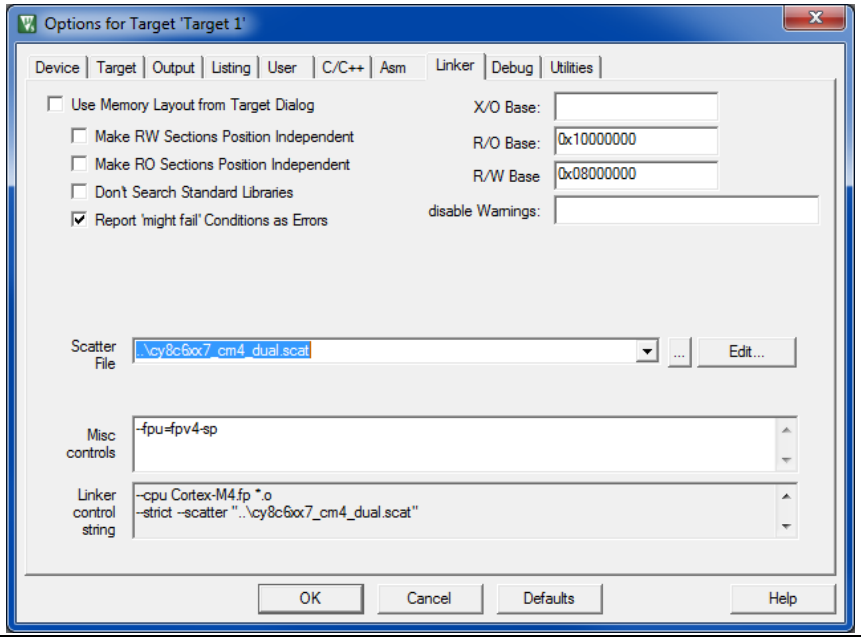

Figure 29　　CM4 プロジェクト用リンカーオプション

- **Options** ダイアログの **Debug** タブの Target Driver Setup で、**Reset** オプションに VECTRESET を選択してください。**Figure 27** を参照してください。
- **Figure 30** に示すように、**Options** ダイアログの **Utilities** タブで、**Update Target before Debugging** がチェックされていることを確認してください。示されるようにアルゴリズム値の RAM を設定してください。**Reset and Run** の確認はオプションですが便利です。

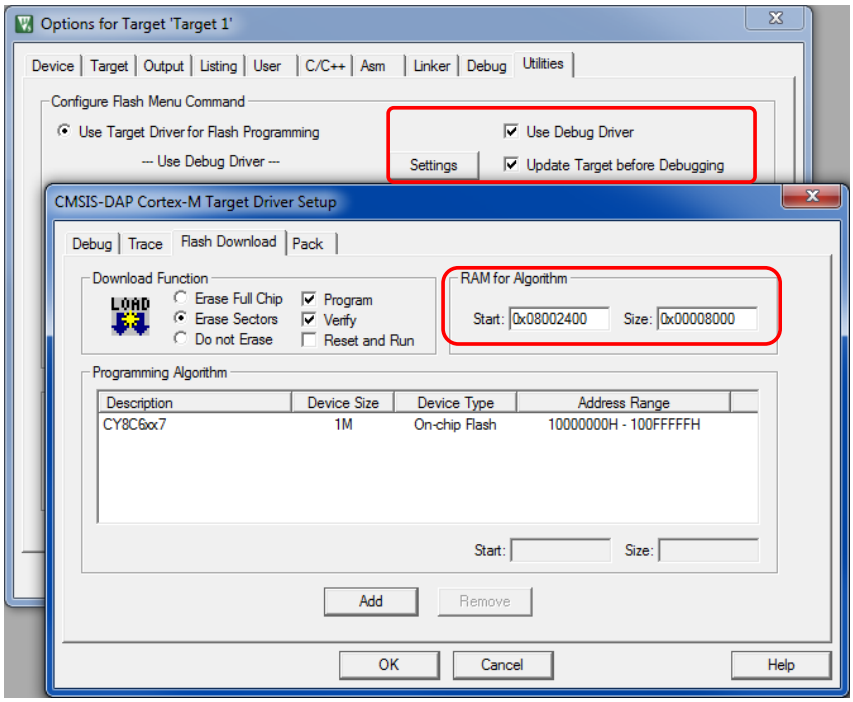

Figure 30　　CM4 プロジェクト用ユーティリティオプション

最後に、uVisionBuild フォルダに、例えば Blinky という名前の µVision ワークスペース (**Project > New Multi-Project Workspace...)** を作成してください。作成した 2 つのプロジェクトをそのワークスペースに追加してください。作成したワークスペースとプロジェクト, および対応するファイルは、**Figure 31** のようになります。

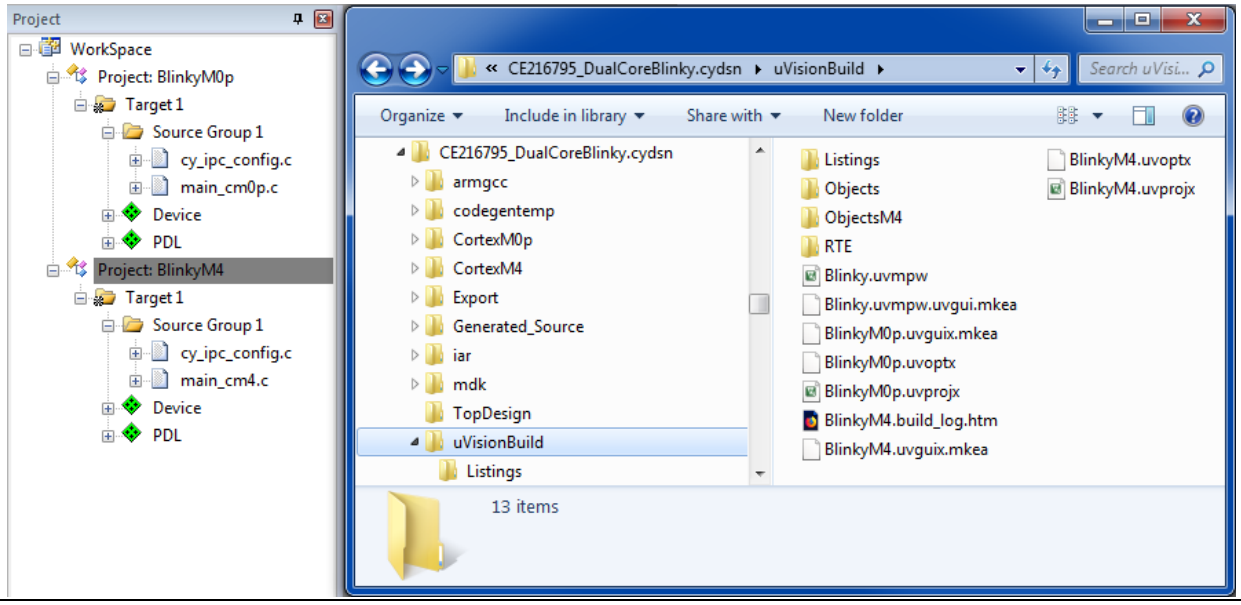

Figure 31　　結果の µVision プロジェクトウィンドウとプロジェクトファイル

プロジェクトを順番にビルドしてください。最初に CM0+プロジェクトをビルドしてください。µVision には、プロセスを自動化するためのバッチビルド機能があります。ビルドが正常に完了したら、

BlinkyM4 プロジェクトを右クリックして、それをアクティブプロジェクトとして設定してください。次に、(1)フラッシュを消去 (**Flash** > **Erase)** し、(2)プロジェクトをダウンロード (**Flash** > **Download)** して正しい動作を確認して、ビルドオプションをテストしてください。Reset and Run を選択しなかった場合 (**Figure 30** 参照)、操作を開始するにはキットのリセットボタン (RST / SW1) を押す必要があります。

Note: *CM0+ プロジェクト内のコードを変更した場合は、両方のプロジェクトを再ビルドする必要があります。µVision には、プロセスを自動化するためのバッチビルド機能があることに注意してください。*

3. µVision プロジェクトのデバッグ

CM4 プロジェクトでデバッグを開始してください。CM4 プロジェクトをダウンロードすると、両方の CPU のコードがインストールされます。CM4 プロジェクトをアクティブプロジェクトとして設定し、必要に応じてそれをダウンロードし、**Debug > Start/Stop Debug Session** をクリックしてデバッグを開始してください。µVision ウィンドウは、**Figure 32** のようになります。

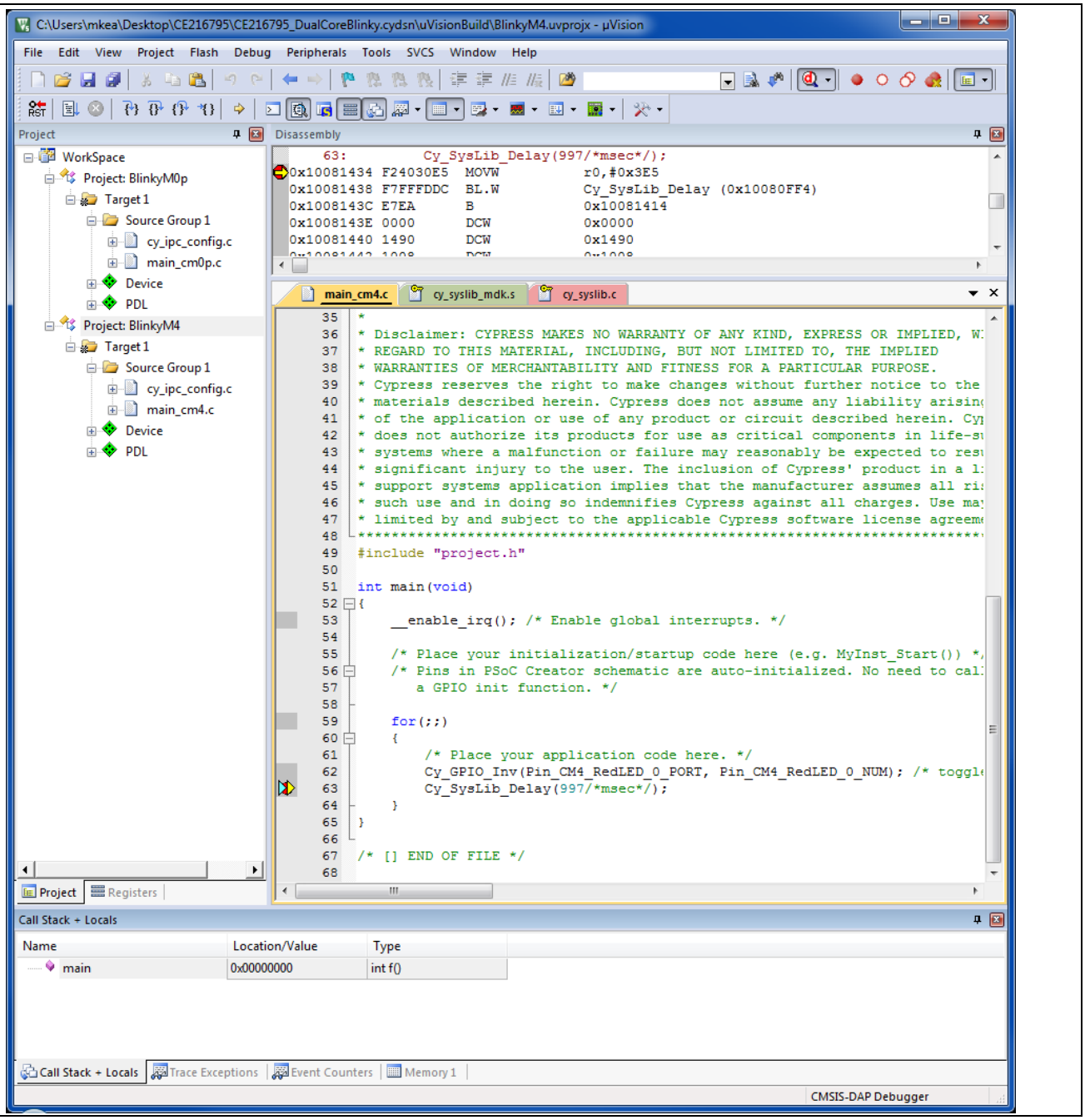

Figure 32 CM4 デバッグウィンドウ

CE216795 デュアル CPU 点滅プロジェクトを実行している場合は、63 行目に `Cy_Syslib_Delay()`にブレークポイントを設定してください。次に **Debug > Run** の順にクリックすると、赤い LED がブレークポイントで停止するたびに切り替わります。

そして µVision の 2 番目のインスタンスを開き、同じワークスペースをロードしてください。両方のインスタンスがキット接続と PSoC 6 MCU デバッグアクセスポート (DAP) を共有します。CM0+プロジェクトをアクティブにして、デバッグセッションを開始してください。63 行目の `Cy_Syslib_Delay()`にブレークポイントを設定してください。次に **Debug > Run** の順にクリックすると、青い LED がブレークポイントで停止するたびに切り替わります。

Note: *55 行目で Cy_SysEnableCM4( ) 関数呼び出しを実行すると、CM4 の実行が再開されます。CM4 ウィンドウに行き、**Debug > Stop** の順にクリックし、次に **Debug > Run** の順にクリックしてください。CM4 は再びブレークポイントまで実行されます。*

インスタンスウィンドウをデスクトップに並べて配置すると便利です。ウィンドウは、**Figure 33** のように表示されます。適切なウィンドウをクリックして、目的の CPU でデバッグ操作を実行してください。ブレークポイントは CPU ごとに個別に設定できることに注意してください。どちらのウィンドウからも同じメモリアドレスを読み出して更新できます。

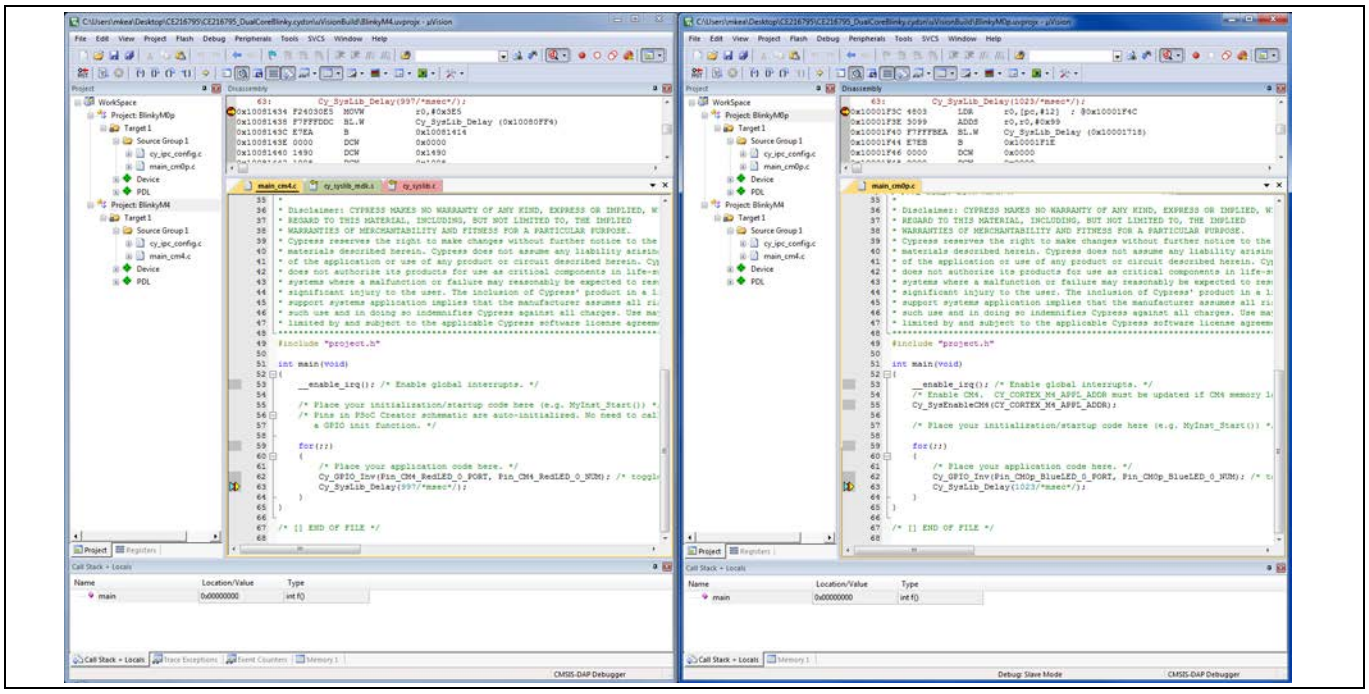

Figure 33 µVision デュアル CPU デバッグ

4. IAR-EW プロジェクトの生成

IAR Embedded Workbench (IAR-EW) の場合、2 つのプロジェクトを作成する必要があります。CM0+と CM4 の各 PSoC 6 MCU CPU に 1 つずつです。以下を実施してください。

Note: *IAR-EW プロジェクトファイルは PSoC Creator の<project>.cydsn フォルダに作成する必要があります。PSoC Creator の<project>.cydsn フォルダ内に別のフォルダを作成しないでください(これは **µVision の手順**とは異なります)。*

推奨: 同じフォルダ内の PSoC Creator ファイルと IAR-EW ファイルを区別するために、各プロジェクトとワークスペースのファイル名に“IAR_”などのタグを追加してください。

Arm 8.22 以降用の IAR Embedded Workbench を開き、新しいプロジェクトを作成してください (**Project > Create New Project...**) 。Create New Project ダイアログ (**Figure 34**) で、ツールチェーンが **Arm** であることを確認し、**Empty project** テンプレートを選択して **OK** をクリックしてください。

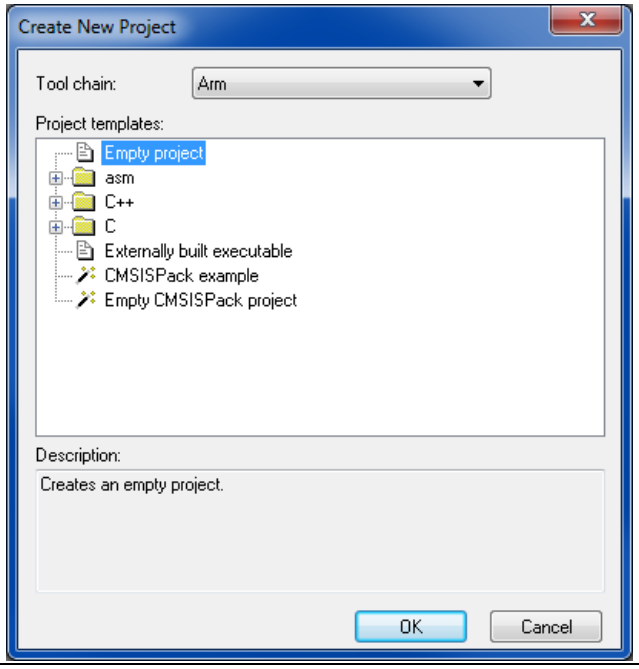

Figure 34 IAR Embedded Workbench の新規プロジェクトの作成ダイアログ

推奨: ダイアログ (**Figure 35**) の保存で、元の PSoC Creator プロジェクト名とターゲット CPU に基づいてプロジェクトに名前を付けてください。例えば、**CE216795** デュアル CPU 点滅プロジェクトの場合は、CM0+ CPU 用の µVision プロジェクト IAR_BlinkyM0p を作成してください。

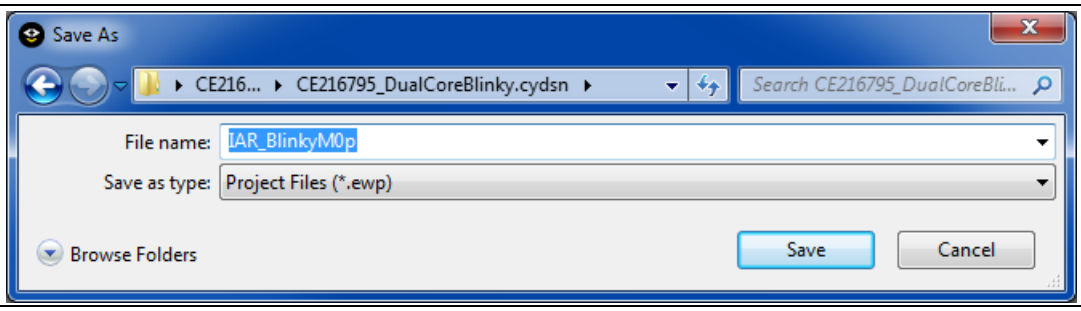

Figure 35 CM0+用 IAR Embedded Workbench プロジェクトの生成

Tools > Options を選択し、**Enable project connections** がチェックされていることを確認してください。**OK** をクリックしてください。次に、**Project > Add Project Connection...**を選択してください。次のダイアログで、**IAR Project Connection** を使用して接続を選択し、**OK** をクリックしてください。次に、**Figure 36** に示すように...CortexM0p.ipcf ファイルを選択してください。**OK** をクリックすると、いくつかのフォルダとファイルが Workspace ウィンドウのプロジェクトに追加されます。

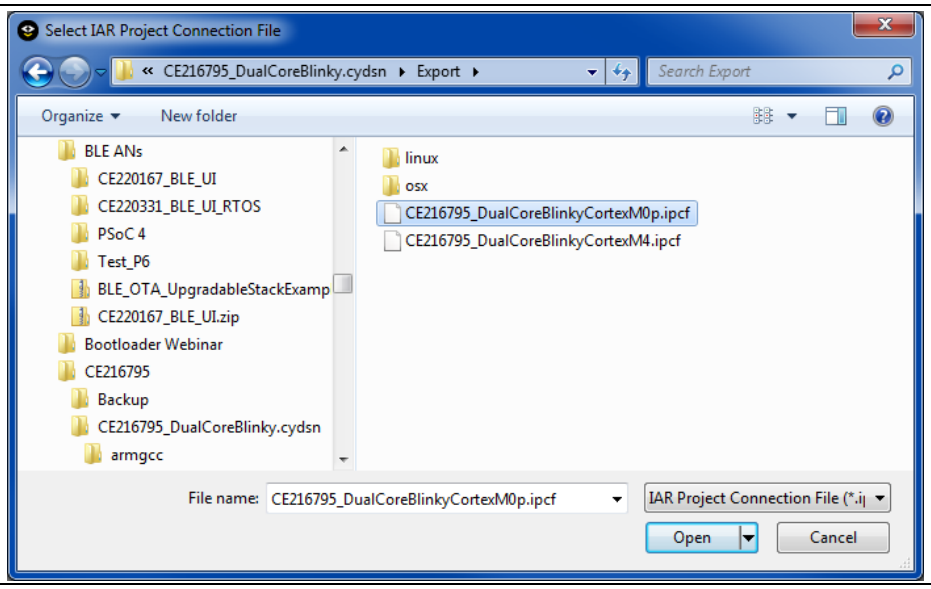

Figure 36 PSoC Creator プロジェクトエクスポートフォルダから IAR プロジェクト接続ファイルの選択

プロジェクトが作成されたので、次にそのオプションを設定する必要があります。プロジェクトを右クリックして、**Options...**を選択してください。**Figure 37** に示すように、Options ダイアログの **Build Actions** セクションで postbuildCortexM0p.bat が呼び出されることを確認してください。

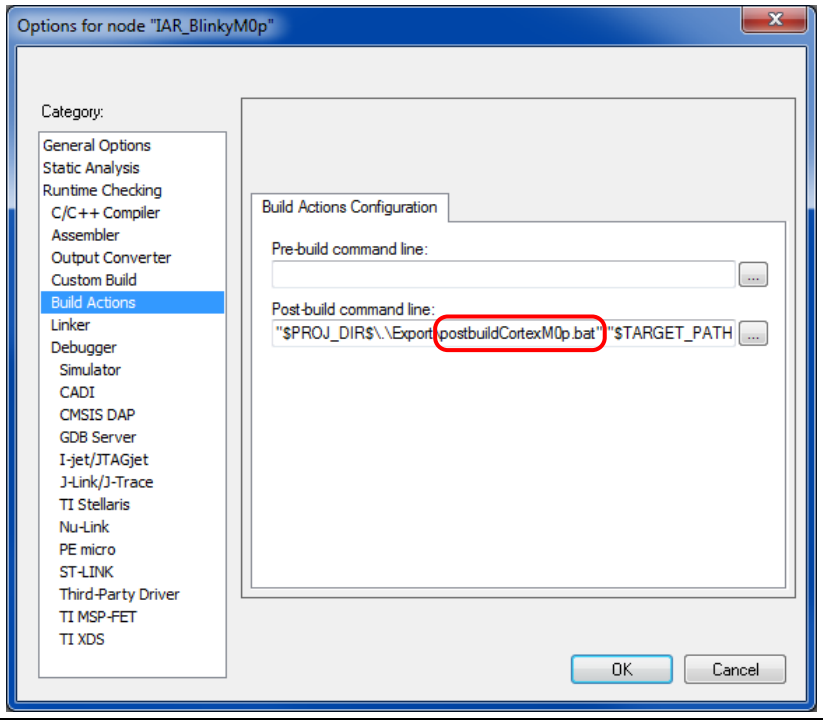

Figure 37 PSoC Creator ポストビルドバッチファイルの選択

Debugger セクションの **Setup** タブで、**CMSIS DAP** ドライバを選択してください。 **Download** タブで、**Suppress download** をチェックしてください。**CMSIS DAP** セクションの **Setup** タブで、**Reset** を **Disabled (no reset)** に設定してください。アプリケーションは CM4 プロジェクトによってロードされます。**Interface** タブで、**SWD** を選択してください。**OK** をクリックしてください。

前のステップを繰り返して、CM4 用の 2 番目のプロジェクトを作成してください。**推奨:** オリジナルの PSoC Creator プロジェクト名とターゲット CPU に基づいてプロジェクトに名前を付けてください。例えば、**CE216795** デュアル CPU 点滅プロジェクトの場合は、IAR-EW プロジェクト IAR_BlinkyM4 を作成してください。**Figure 35** を参照してください。プロジェクトを CM0+プロジェクトと同様に構成してください。以下の違いがあります。

- CM4 プロジェクトは CM0+プロジェクトと同じフォルダになければなりません。この場合、自身の PSoC Creator <project>.cydsn フォルダです。**Figure 35** を参照してください。
- ...CortexM4.ipcf ファイルを選択してください。**Figure 36** を参照してください。
- **Figure 38** に示すように、**Options** ダイアログの **General Options** セクションの **Output** タブで、オブジェクトファイルとリストファイルの出力ディレクトリを変更してください。実行可能ファイル/ライブラリの出力フォルダを変更しないでください。

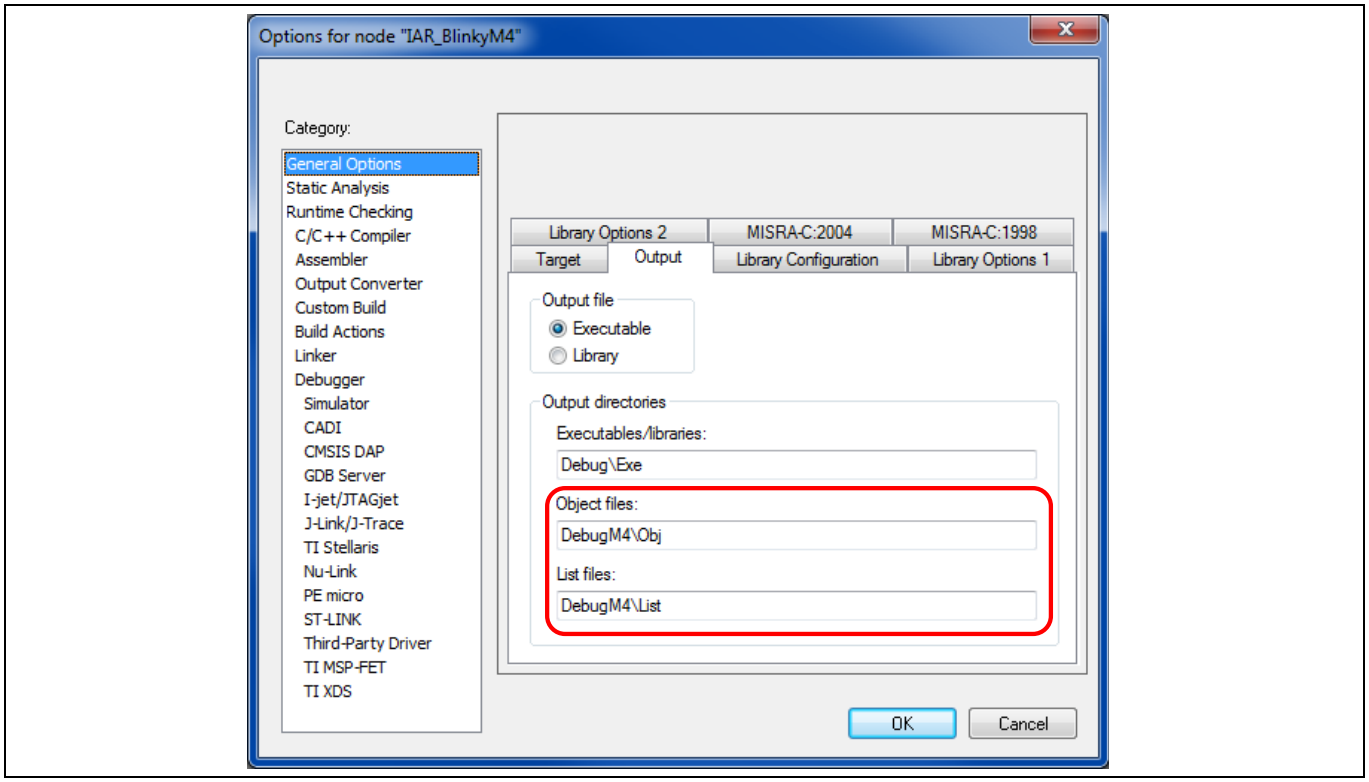

Figure 38　　CM4 プロジェクト用固有出力フォルダ

- **Build Actions** セクションで、postbuildCortexM4.bat が呼び出されることを確認してください。**Figure 37** を参照してください。
- **Debugger** セクションの **Setup** タブで、**CMSIS DAP** ドライバを選択してください。**CMSIS DAP** セクションの **Setup** タブで、**Reset** が **System (default)** に設定されていることを確認してください。**Interface** タブで、**SWD** を選択してください。**OK** をクリックしてください。

File > Save All を選択してください。両方のプロジェクトのすべてのファイルが保存され、ワークスペースファイルが自動的に生成されます。**Save Workspace As** ダイアログで、PSoC Creator の<project>.cydsn

フォルダに、例えば IAR_Blinky という名前の IAR-EW ワークスペースを作成してください。作成されたワークスペースとプロジェクト、および対応するファイルは、**Figure 39** のようになります。IAR-EW によって生成されたファイルとフォルダが強調表示されます。

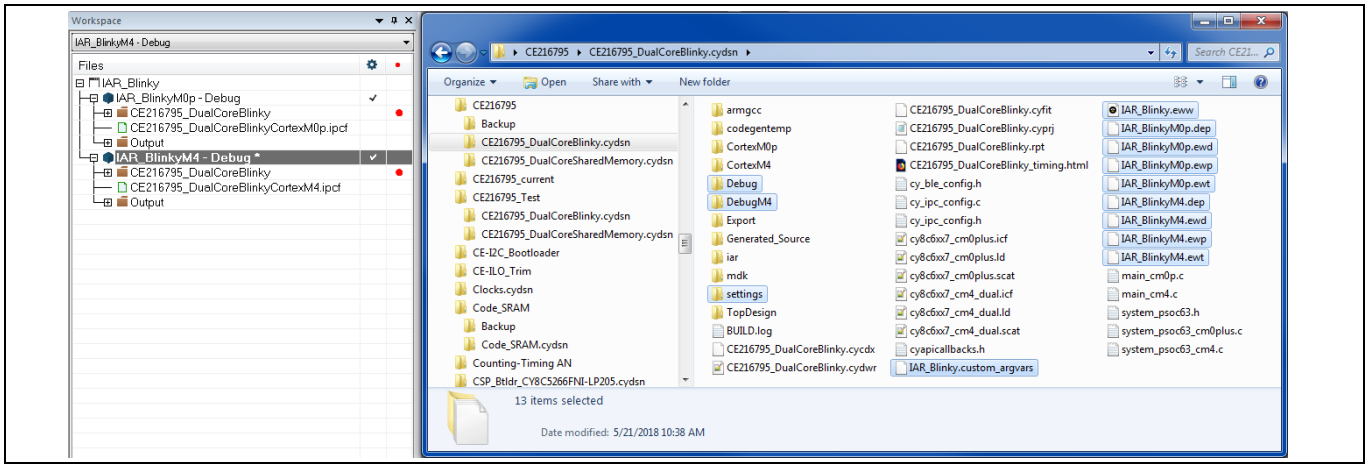

Figure 39 IAR Embedded Workbench プロジェクトウィンドウとプロジェクトファイルの結果

CY8CKIT-062-BLE の USB ポートをコンピュータに接続してください。キットボタン SW3 を押して KitProg2 を CMSIS-DAP モードにしてください。詳細はキットガイドを参照してください。これにより、外部のデバッグプローブを使用せずにデバッグできます。

プロジェクトを順番にビルドしてください。最初に CM0+プロジェクトをビルドしてください。IAR-EW には、プロセスを自動化するためのバッチビルド機能があることに注意してください。ビルドが正常に完了したら、BlinkyM4 プロジェクトを右クリックしてそれをアクティブプロジェクトとして設定してください。次に、(1)フラッシュを消去し (**Project > Download > Erase memory**) 、(2)プロジェクトをダウンロードし (**Project > Download > Download active application**) 、正しい操作を確認して、ビルドオプションが正しいことを確認してください。ダウンロード後、キットリセットボタン (RST / SW1) を押して動作を開始してください。

Note: *フラッシュを消去する場合、通常は PSoC 6 MCU アプリケーションフラッシュ (0x1000 0000 – 0x100F FFFF) のみを消去する必要があります (**Figure 40** を参照) 。*

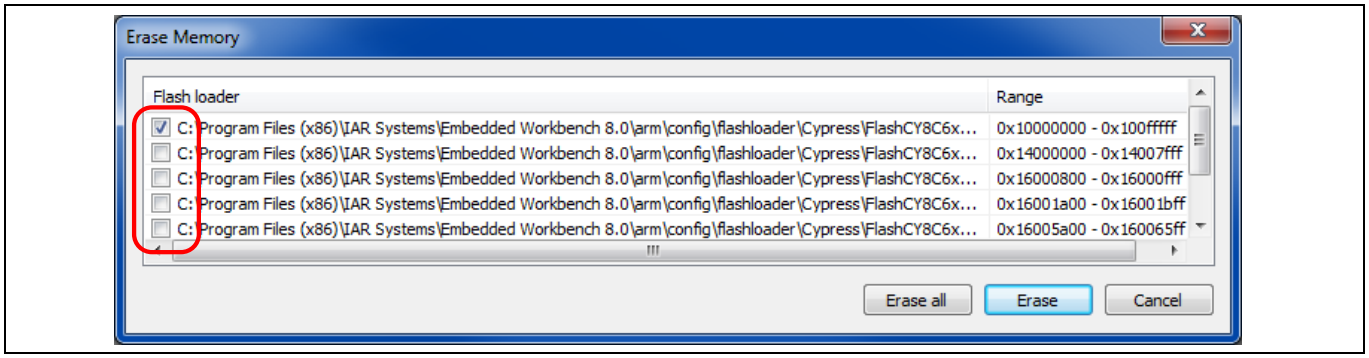

Figure 40 IAR PSoC 6 MCU 用 Embedded Workbench 消去メモリダイアログ

Note: *CM0+ プロジェクトのコードを変更する場合は、両方のプロジェクトを再ビルドする必要があります。IAR-EW には、プロセスを自動化するためのバッチビルド機能があることに注意してください。*

5. IAR-EW プロジェクトのデバッグ

CM4 プロジェクトのオプションを再度開き、**Debugger** セクションの **Multicore** フォルダに移動してください。PSoC 6 MCU には異なるコア、つまり CM0+と CM4 があり、これらは「非対称マルチコア」と呼ばれます。したがって、**Figure 41** に示すように、**Asymmetric multicore** セクションのフィールドに入力してください。**Enable multicore master mode** にチェックを入れると、CM4 CPU はダウンロードおよびデバッグ用のマスタになります。**Port** を変更しないでください。

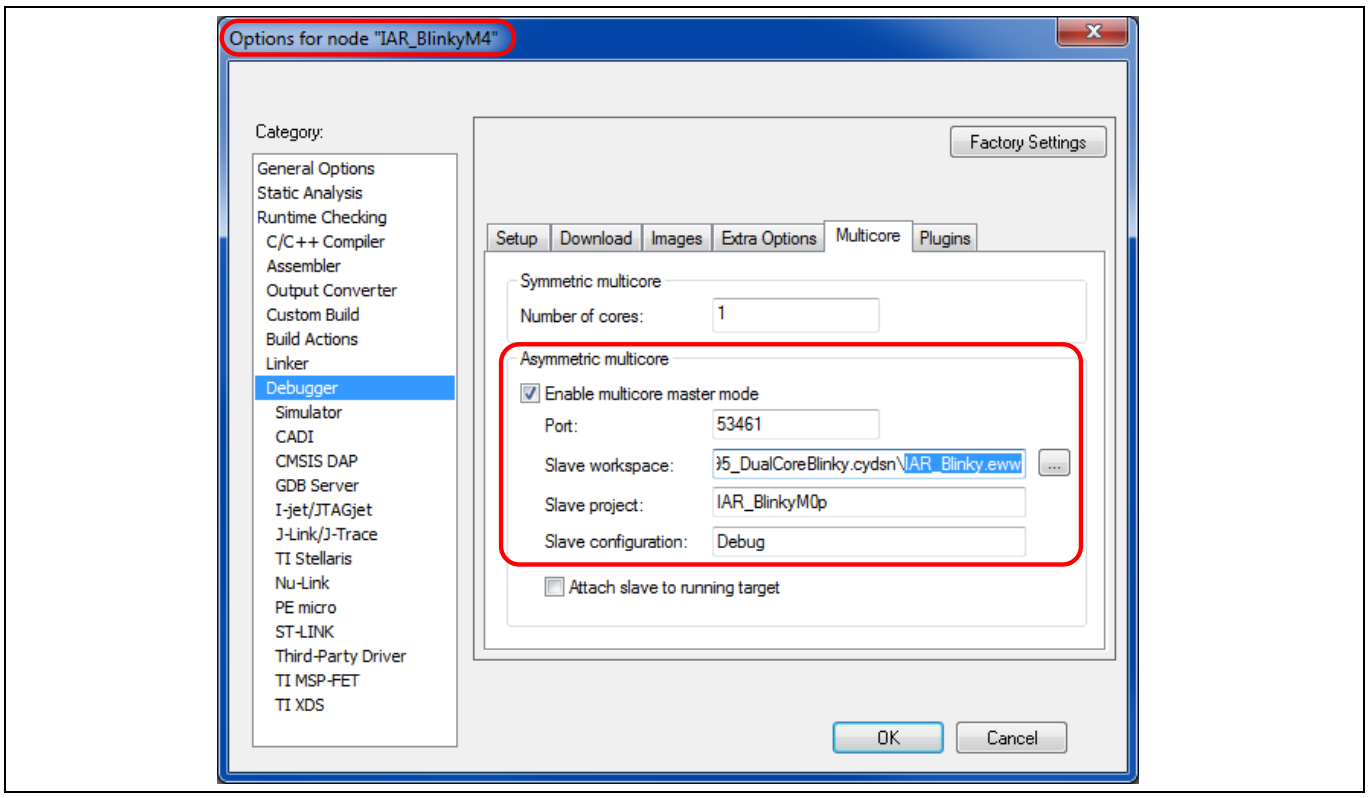

Figure 41 マルチコアデバッグの設定

File > Save All を選択してプロジェクトオプションの変更を保存してください。次に、**Project > Download and Debug** または **Project > Debug without Downloading** を選択してデバッグを開始してください。IAR Embedded Workbench の 2 番目(スレーブ) のインスタンスは、CM0+プロジェクト用に自動的に開かれます。両方のインスタンスがキット接続と PSoC 6 MCU デバッグアクセスポート (DAP) を共有します。

スレーブインスタンスで、63 行目に `Cy_Syslib_Delay()`にブレークポイントを設定してください。次に、**Debug > Go** を繰り返しクリックすると、青い LED がブレークポイントで停止するたびに切り替わります。

CM4 インスタンスウィンドウ内の任意の場所をクリックして、手順を繰り返してください。赤い LED はブレークポイントで停止ごとに切り替わります。

インスタンスウィンドウをデスクトップに並べて配置すると便利です。ウィンドウは、Figure 42 のように表示されます。適切なウィンドウをクリックして、目的の CPU でデバッグ操作を実行してください。ブレークポイントは CPU ごとに個別に設定できることに注意してください。どちらのウィンドウからも同じメモリアドレスを読み出して更新できます。

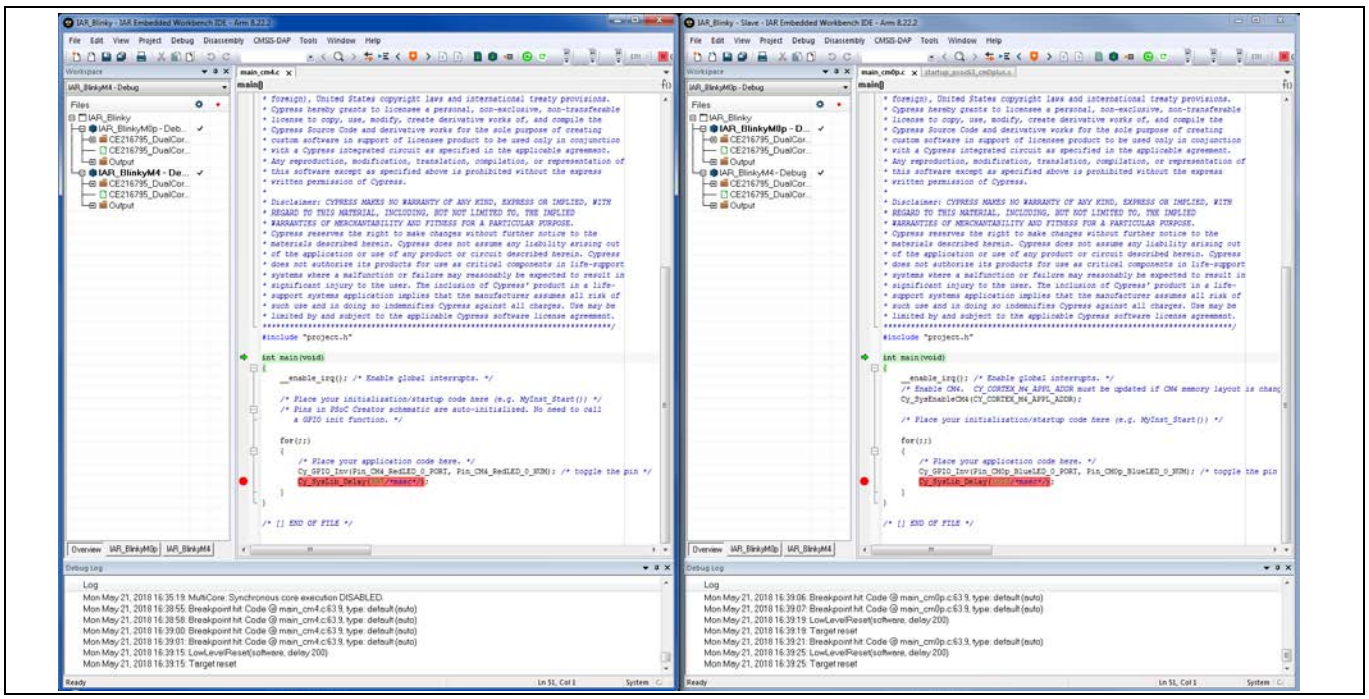

Figure 42　　IAR Embedded Workbench デュアル CPU デバッグ

どちらのウィンドウでもデバッグを中止できます。両方の CPU のデバッグは終了します。キットのリセットボタン (RST / SW1) を押してキットの操作を再開してください。

5 まとめ

本アプリケーション ノートでは、PSoC 6 MCU のデュアル CPU 機能のファームウェアおよびハードウェア設計を使用および最適化する方法を示しました。

PSoC 6 MCU 設計を最適化するもう 1 つの方法は、PSoC がプログラマブル アナログおよびデジタル ブロックでカスタム機能を構築できる柔軟なデバイスとして設計されるということに基づきます。例えば、PSoC 6 MCU には「コプロセッサ」として動作する以下のペリフェラルがあります。

- **DMA コントローラー。** C コンパイラによって出力される最も一般的な CPU アセンブラ命令は MOV, LDR, および STR であることに注意してください。これは、CPU がバイトを移動するだけで多くのサイクルを費やすことを意味します。かわりに DMA コントローラーがそれを実行させます。

Note: *PSoC 6 MCU DMA コントローラーは、CPU とは独立した複雑なデータ転送および制御システムを構築するための様々な機能を持ちます。 これらの機能のソフトウェアサポートは、ModusToolbox ソフトウェア Device Configurator、PSoC Creator DMA コンポーネント, および PDL の API で提供されています。詳細については、Device Configurator のヘルプガイド, DMA コンポーネントのデータシート, または PDL のマニュアルを参照してください。サイプレス HAL は CM4 CPU のみをサポートするため、DMA HAL ドライバは CM4 アプリケーションでのみ機能します。*

- **暗号化ブロック。** このブロックは、対称および非対称暗号方式 (AES、3DES、RSA、および ECC) のハードウェア アクセラレーションおよびハッシュ関数 (SHA-512、SHA-256) を提供します。また、真の乱数発生器 (TRNG) 機能も備えます。これらの機能のソフトウェア サポートは、PDL の API によって提供されます。詳細は PDL 文書を参照してください。サイプレス HAL には現在暗号ドライバがありません。
- **汎用デジタルブロック (UDB)。** 12 個の UDB があり、各 UDB には 8 ビットのデータパスがあり、これを加算、減算、ビット単位演算、シフト、巡回冗長検査 (CRC) に使用できます。データパスは、ワード幅計算のために連鎖させることができます。CPU 計算をデータパスにオフロードすることを検討してください。現時点では、PSoC Creator のみが UDB に対応します。ModusToolbox ソフトウェアは UDB に対応しません。
- UDB は、ステートマシンの構築に使用できるプログラマブル ロジック デバイス (PLD) も備えます。サンプルは、ルックアップ テーブル (LUT) コンポーネント データシートを参照してください。LUT は C 言語のスイッチ文／ケース文などを使用し、CPU のプログラミング ステート マシンの代わりに効果的なハードウェアベースの代替デバイスにできます。
  さらに、2 つの GPIO ポートには、GPIO ピンに入出力する信号を直接ブール演算するために使用できるスマート I/O™が含まれます。サイプレス HAL はスマート I/O に対応しません。
- その他のスマート ペリフェラルには、シリアル通信ブロック (SCB)、カウンター／タイマー／PWM ブロック (TCPWM)、Bluetooth Low Energy (Bluetooth LE)、I2S／PDM オーディオ、プログラマブル アナログ、CapSense®を含みます。これらのペリフェラルを使用して、CPU からの処理をさらにオフロードします。

PSoC Creator は、ペリフェラルの機能をサポートするために、多くのコンポーネントと PDL の広範な API を提供します。これにより、効果的なマルチプロセッシング システムをシングル チップに開発し、CPU から多くの処理をオフロードできます。これは、コード サイズを縮小するだけでなく、CPU が実行しなければならないタスクの数を減らすことによって、CPU 速度と消費電力を削減する機会を提供します。

例えば、CPU の使用なしで多重化 ADC 入力を制御し、DMA とインターフェースして SRAM にデータを保存し、高度なアナログ データ収集システムを作成するデジタル システムを実装します。

まとめ

ModusToolbox ソフトウェアは、周辺機器をセットアップするための一連のツール、すべてのサイプレスキット用の事前定義された BSP、CapSense や emWin などの一般的な機能用のライブラリ、および開始するための包括的な一連のサンプルアプリケーションを提供します。

サイプレスは、PSoC ペリフェラルに関する広範なアプリケーション ノートとサンプル コードだけでなく、デバイス データシート、PDL 文書、HAL 文書、およびテクニカル リファレンス マニュアル (TRM) の詳細情報を提供しています。詳細は関連文書を参照してください。

6 関連資料

PSoC 6 MCU リソースの包括的なリストについては、サイプレスコミュニティの **KBA223067** を参照してください。

アプリケーションノート	
AN221774 – Getting Started with PSoC 6 MCU	PSoC 6 MCU デバイスと、最初の ModusToolbox または PSoC Creator プロジェクトを構築する方法について説明
AN210781 – Getting Started with PSoC 6 MCU with Bluetooth Low Energy (BLE) Connectivity	BLE Connectivity デバイスを搭載した PSoC 6 MCU と、はじめて PSoC Creator プロジェクトを構築する方法について説明
AN217666 – PSoC 6 MCU Interrupts	PSoC 6 MCU の割込みアーキテクチャと割込みの設定方法について説明
AN219434 – Importing PSoC Creator Code into an IDE for a PSoC 6 MCU Project	PSoC Creator で生成されたコードを好みの IDE にインポートする方法について説明
サンプル コード (PSoC Creator)	
CE216795 – PSoC 6 MCU Dual-Core Basics	PSoC 6 MCU の 2 つの CPU コアで独立したタスクを実行し、共有メモリとプロセッサ間通信 (IPC) ブロックを使用して相互通信する方法について説明
CE223820 – PSoC 6 MCU IPC Pipes	この例は、プロセッサ間通信 (IPC) ドライバを使用して PSoC 6 MCU にメッセージパイプを実装する方法を示します。パイプは、CPU 間でメッセージを送信するために使用されます。
CE223549 – PSoC 6 MCU IPC Semaphore	この例は、プロセス間通信 (IPC) ドライバを使用して PSoC 6 MCU にセマフォを実装する方法を示します。セマフォは、CPU によって共有されるリソースへのアクセスを制御するためのロックとして使用されます。
CE226306 – PSoC 6 MCU Power Measurements	この例は、PSoC 6 MCU データシートにリストされている電力測定を実現する方法を示します。
サンプル コード (ModusToolbox)	
mtb-example-psoc6-dual-cpu-empty-app	これは、PSoC 6 MCU デバイス用の最小限のスターターデュアル CPU アプリケーションテンプレートです。
mtb-example-psoc6-dual-cpu-ipc-sema	この例は、プロセッサ間通信 (IPC) ドライバを使用して PSoC 6 MCU にセマフォを実装する方法を示します。セマフォは、CPU によって共有されるリソースへのアクセスを制御し、初期化命令を同期するためにロックするために使用されます。
mtb-example-psoc6-dual-cpu-ipc-pipes	この例は、プロセッサ間通信 (IPC) ドライバを使用して PSoC 6 MCU にメッセージパイプを実装する方法を示します。パイプは、CPU 間でメッセージを送信するために使用されます。
PSoC Creator コンポーネント データシート	
Interrupt	ハードウェア信号から CPU 割込み生成をサポート
デバイス資料	
PSoC 6 MCU Datasheet	**PSoC 6 Technical Reference Manual**

関連資料

開発キット資料	
CY8CKIT-062-BLE	PSoC 6 Bluetooth LE パイオニアキット
CY8CKIT-062-WiFi-BT	PSoC 6 WiFi-BT パイオニアキット
CY8CKIT-062S2-43012	PSoC 62S2 Wi-Fi BT パイオニアキット
CY8CPROTO-063-BLE	PSoC 6 Bluetooth LE プロトタイプキット
CY8CPROTO-062-4343W	PSoC 6 Wi-Fi プロトタイプキット
CY8CPROTO-062S3-4343W	PSoC 62S3 Wi-Fi BT プロトタイプキット
CYW9P62S1-43438EVB-01	PSoC 62S1 Wi-Fi BT パイオニアキット
CYW9P62S1-43012EVB-01	PSoC 62S1 Wi-Fi BT パイオニアキット
ツール資料	
ModusToolbox IDE	ModusToolbox ソフトウェアは IoT デザイナーのための開発を簡単にします。使いやすいツールと、Windows, macOS, および Linux 用の使い慣れたマイクロコントローラ (MCU) 統合開発環境 (IDE) を提供します。
PSoC Creator	PSoC Creator は PSoC デバイスのハードウェアとファームウェアの同時編集、コンパイルおよびデバッグを可能にします。アプリケーションは、回路図キャプチャと 150 以上の事前検証済みのプロダクション対応ペリフェラル コンポーネントを使用して作成されます。クイックスタートガイドとユーザーガイドのダウンロードタブを参照してください。
Peripheral Driver Library (PDL)	PSoC Creator 4.3 によってインストールされ、GitHub から利用できます。**https://github.com/cypresssemiconductorco/psoc6pdl** にアクセスしてください。PSoC Creator の場合は、<PDL install folder>\doc でユーザーガイドと API リファレンスを確認してください。
Hardware Abstraction Layer (HAL)	GitHub からのみ利用できます。**https://github.com/cypresssemiconductorco/psoc6hal** にアクセスしてください。

著者について

氏名: Mark Ainsworth

役職: シニア アプリケーション エンジニア主任

経歴: Mark Ainsworth は、シラキュース大学のコンピュータ工学の学士号およびワシントン大学の修士号を取得しました。また、組込みシステムの設計と構築に長く従事しています。

改訂履歴

Document version	Date of release	Description of changes
**	2018-06-26	これは英語版 002-15656 Rev. *D を翻訳した日本語版 Rev. **です。
*A	2019-07-26	これは英語版 002-15656 Rev. *F を翻訳した日本語版 Rev. *A です。
*B	2021-05-17	これは英語版 002-15656 Rev. *H を翻訳した日本語版 Rev. *B です。

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

Edition 2021-05-17
Published by
Infineon Technologies AG
81726 Munich, Germany

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

Document reference
002-23479 Rev. *B

重要事項

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

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

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

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

警告事項

技術的要件に伴い、製品には危険物質が含まれる可能性があります。当該種別の詳細については、インフィニオンの最寄りの営業所までお問い合わせください。
インフィニオンの正式代表者が署名した書面を通じ、インフィニオンによる明示の承認が存在する場合を除き、インフィニオンの製品は、当該製品の障害またはその使用に関する一切の結果が、合理的に人的傷害を招く恐れのある一切の用途に使用することはできないこと予めご了承ください。