# PSOC™ 6 MCU デュアルコア システム設計

## 本書について

### 適用範囲と目的

AN215656 は、Arm® Cortex®-M4, Cortex®-M0+コア, およびプロセッサ間通信 (IPC) モジュールを含む PSOC™ 6 MCU のデュアルコア アーキテクチャについて説明します。デュアルコア 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 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コア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。

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。インフィニオンビデオライブラリについては、ここからご覧ください。

### 対象者

本アプリケーションノートは、PSOC™ 6 MCU のデュアルコア機能をフルに活用したい上級エンジニアを対象としています。

## 目次


**本書について** . . . 1
**目次** . . . 1
**1 はじめに** . . . 3
1.1 ツールとライブラリ . . . 5
1.2 この文書の使い方 . . . 6
**2 一般的なデュアルコアのコンセプト** . . . 7
**3 PSOC™ 6 MCU デュアル CPU アーキテクチャ** . . . 8
**4 PSOC™ 6 MCU デュアルコアでの開発プロセス** . . . 10
4.1 ModusToolbox™ソフトウェアの説明 . . . 10
4.1.1 デュアルコア アプリケーションの作成 . . . 10
4.1.1.1 ModusToolbox™用の Eclipse IDE . . . 10
4.1.1.2 コマンドラインインターフェース (CLI) . . . 11
4.1.1.3 アプリケーション内容の変更 . . . 12
4.1.1.4 ビルドとプログラム . . . 13
4.1.2 リンカースクリプトのカスタマイズ . . . 14
4.1.3 ライブラリと周辺機器の共有 . . . 14
4.2 PSOC™ Creator の手順 . . . 16
4.3 リソース割当ての検討事項 . . . 19
4.4 電力モード遷移の検討事項 . . . 19
4.5 IPC 構成の検討事項 . . . 20
4.6 割込み割当ての検討事項 . . . 20
4.7 デバッグに関する検討事項 . . . 22
4.7.1 ModusToolbox™の手順 . . . 22
4.7.2 PSOC™ Creator の手順 . . . 22

## 目次


4.7.3 他の IDE の手順 . . . . . 23
4.7.3.1 ModusToolbox™アプリケーションをサポート IDE へエクスポート . . . . . 23
4.7.3.2 PSOC™ Creator からのエクスポート . . . . . 23
4.8 接続アプリケーション設計の検討事項 . . . . . 41
4.8.1 スループット分析 . . . . . 41
4.8.2 消費電力の分析 . . . . . 44
4.8.3 コア間のアプリケーションの移植 . . . . . 47
4.8.4 仮想コネクティビティ マネージャー . . . . . 47

**5 まとめ** . . . . . 48

**関連ドキュメント** . . . . . 49

**改訂履歴** . . . . . 51

**商標** . . . . . 53

**免責事項** . . . . . 54

# 1 はじめに

PSOC™ 6 MCU は、IoT 専用に設計された超低消費電力 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。図 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。

## 1 はじめに

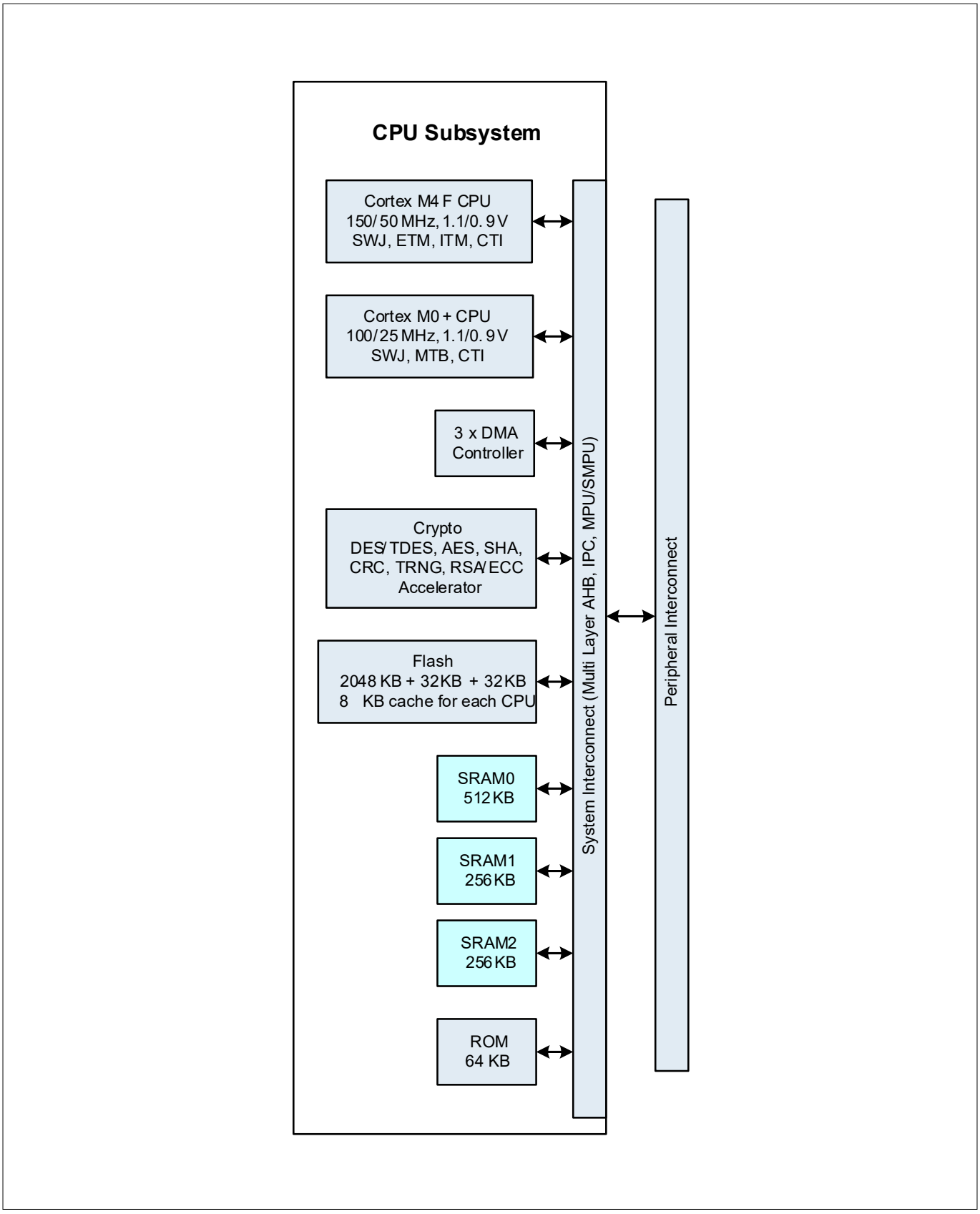


**図 1　　PSOC™ 6 MCU の標準 CPU サブシステム アーキテクチャ**

## 1 はじめに

*注:* *図 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コア 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コア 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 mtb-example-psoc6-dual-cpu-ipc-pipes 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

インフィニオンは、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™クロスプラットフォーム 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 でのみ実行されるインフィニオン独自の 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でアプリケーションをロードして実行するためのメカニズムを提供します。

インフィニオンは、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、mtb-pdl-cat1 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。インフィニオン 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、mtb-hal-cat1 の Web ページにアクセスしてください。HAL は、ModusToolbox™ソフトウェアでのみ使用できます。PSOC™ Creator は HAL をサポートしません。
- **Middleware:** このカテゴリには、一例として静電容量センシングなど、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、インフィニオン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、インフィニオン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。PSOC™ Creator は BSP を提供しませんが、キットソフトウェアをインストールすることで、選択したキットに基づいた新しいプロジェクトを開始できます。

## 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インフィニオンの 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
- AN228571 - Getting Started with PSoC™ 6 MCU on ModusToolbox™ software
- AN221774 - Getting Started with PSoC™ 6 MCU on PSoC™ Creator
- AN210781 - PSOC™ Creator での PSOC™ 6 Bluetooth® LE 入門

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 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 デバイスに基づく設計では、バージョン 3.0 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。。

本アプリケーション ノートの最初の節では、デュアルコア 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 デュアルコア 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 デュアルコアでの開発プロセスを参照してください。

# 2 一般的なデュアルコアのコンセプト

デュアルコア 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コア 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。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コア 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。デュアルコア 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:** デュアルコア 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

Figure 1 に、PSOC™ 6 MCU のデュアルコア 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、インフィニオンのシステム 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 - PSoC™ 6 MCU designing a custom secured 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 イメージを含むインフィニオン 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™ソフトウェアでは、BSP は利用可能な複数の CM0+ビルド済みイメージの中から 1 つを選択し、含めます。
- **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 CPU 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。

一部のサードパーティ IDE は、同期デュアルコア デバッグをサポートしています。デバッグに関する検討事項を参照してください。インフィニオン ModusToolbox™ソフトウェアは、デュアルコア デバッグ (非同期) に対応します。PSOC™ Creator 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 PSOC™ 6 MCU デュアルコアでの開発プロセス

ここでは、PSOC™ 6 MCU デュアルコア 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。

- AN228571 - Getting Started with PSOC™ 6 MCU on ModusToolbox™ software
- AN221774 - Getting Started with PSOC™ 6 MCU on PSOC™ Creator
- AN210781 - PSOC™ Creator での PSOC™ 6 Bluetooth® LE 入門
- 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 3.0 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 デュアルコア 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。

デュアルコアのアプリケーションを作成するためには、以下のいずれかの方法に従ってください。

1. ゼロからデュアルコアアプリケーションの開発を始める場合、ここでホストされている既存のデュアル CPU サンプルコードを基に新しいアプリケーションを作成できます。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。ほとんどの場合、mtb-example-psoc6-dual-cpu-empty-app が適切です。
2. 既存のシングルコア アプリケーションをデュアルコア アプリケーションに変換する場合は、デュアルコア アプリケーションの作成の手順に従ってください。

### 4.1.1 デュアルコア アプリケーションの作成

ここでは、既存のシングルコア アプリケーションとデュアルコア アプリケーションのスターターを使用して、ステップバイステップでデュアルコア アプリケーションを作成する方法を示します。ここで、スターター デュアルコア アプリケーションは、デュアルコア アプリケーションの構造を定義するテンプレートとして機能します。既存のシングルコア アプリケーションの内容は、デュアルコア アプリケーションに簡単にコピーできます。

シングルコアのアプリケーションでは、例として mtb-example-psoc6-hello-world を使用します。デュアルコアのスターター アプリケーションには、mtb-example-psoc6-dual-cpu-empty-app を使用します。両方のアプリケーションに、CY8CPROTO-062S2-43439 BSP を使用します。

アプリケーションは次の方法のいずれかによって作成できます。

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

#### 4.1.1.1 ModusToolbox™用の Eclipse IDE

1. ModusToolbox™ 3.0 以降の Eclipse IDE を起動し、ワークスペースを選択してください。
2. **Quick Panel** から **New Application** をクリックしてください。
3. プロジェクト作成ウィザードが開いたら、**PSOC™ 6 BSPs** を展開し、**CY8CPROTO-062S2-43439** を選択してください。**Next** をクリックしてください。

4. 新しいウィンドウで **Getting Started** を展開し、**Dual-CPU Empty PSOC6 App** と **Hello World** を選択してください。**Create** をクリックしてください。
5. ワークスペースに両方のアプリケーションが作成されたことが図 2 に示されます。

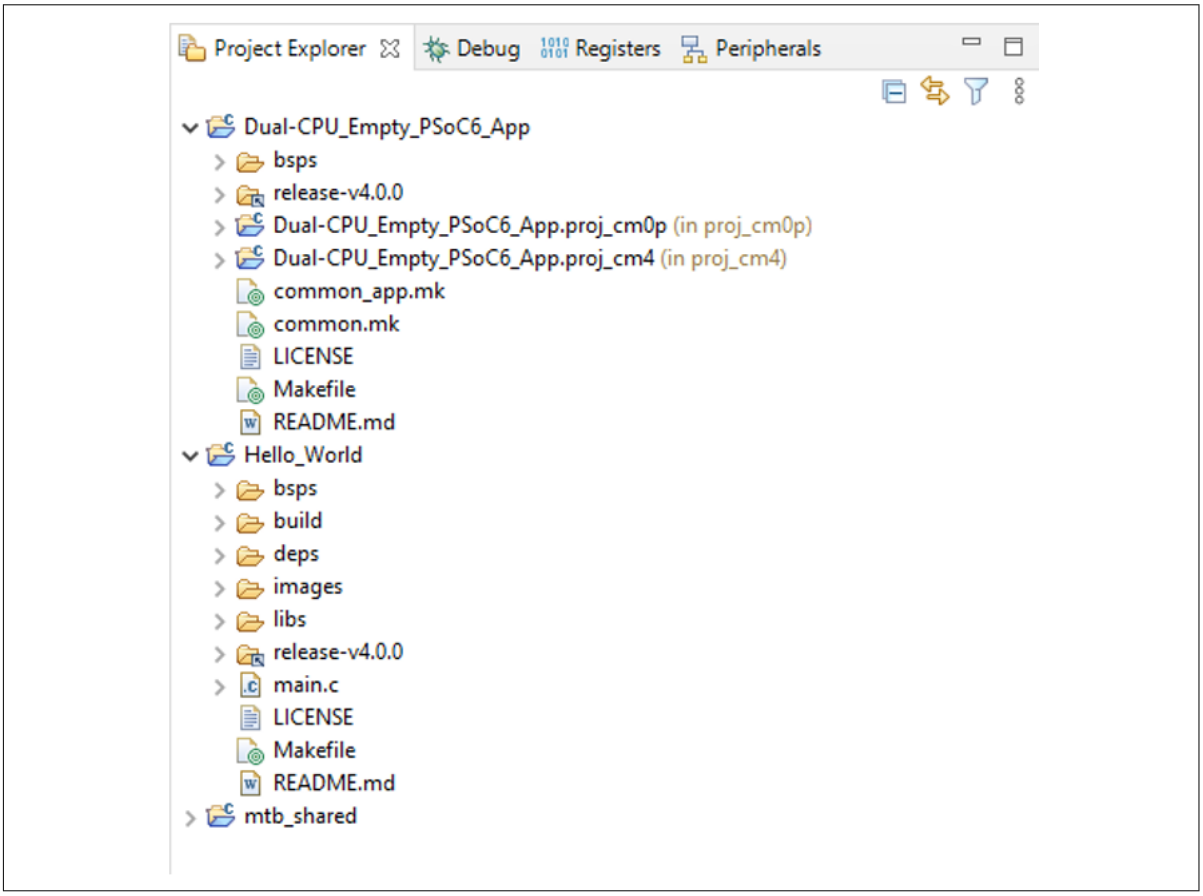


**図 2 シングルコアとデュアルコアの両方のアプリケーションを含む ModusToolbox™ のワークスペース**

### 4.1.1.2 コマンドラインインターフェース (CLI)

1. プロジェクトを作成するために、CLI ターミナルを開き、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、両方のアプリケーションを作成してください。

- ```
  project-creator-cli \
  --board-id CY8CPROTO-062S2-43439 \
  --app-id mtb-example-psoc6-hello-world \
  --user-app-name Hello_World \
  --target-dir "C:/workspace_directory"
  ```

- ```
  project-creator-cli \
  --board-id CY8CPROTO-062S2-43439 \
  --app-id mtb-example-psoc6-dual-cpu-empty-app \
  --user-app-name Dual-CPU_Empty_PSoC6_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 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™ユーザーガイドのセクション 2.3 を参照してください。

#### 4.1.1.3 アプリケーション内容の変更

Dual CPU Empty アプリケーションの CM4 プロジェクト (`proj_cm4`) には、Hello World サンプルの以下の内容を選択的にコピーする必要があります。

1. **Makefile variables** - Hello World アプリケーションと `proj_cm4` アプリケーションの両方の `Makefile` を並べて開きます。Hello World `Makefile` から `proj_cm4` `Makefile` へ、変数値を比較してコピーしてください。Hello World `Makefile` で定義された変数のいくつかが `proj_cm4` `Makefile` に存在しない場合、デュアルコアの空のアプリケーションのルートレベルにある `common.mk` と `common_app.mk` Makefile が利用できるはずです。

   Dual CPU Empty アプリケーションの Makefile にある `MTB_TYPE`、`APPNAME`、`COMPONENTS`、`DISABLE_COMPONENTS` などの変数の現在の値を削除したり変更したりしてはいけません。`COMPONENTS` や `DISABLE_COMPONENTS` などの変数では、新しい値が追加できます。

   Hello World アプリケーションの場合、コピーする必要のある Makefile 変数はありません。
2. **Source files** - `proj_cm4` アプリケーションのすべてのソース ファイルを Hello World アプリケーションに置き換えてください。
3. **Libraries/Middleware** - Hello World アプリケーションで使用されているライブラリやミドルウェアも `proj_cm4` アプリケーションに追加する必要があります。以下の手順に従って追加してください
   a. ModusToolbox™の Eclipse IDE で、クイックパネルから **Library Manager** ツールを起動します。また、ModusToolbox™のインストールディレクトリの tools フォルダから Library Manager を起動することもできます。この場合、Library Manager ウィンドウから Dual CPU Empty アプリケーションへの正しいパスを選択する必要があります。
   b. **Library Manager** が開いたら、**Add Library** をクリックしてください。
   c. **Target Project** プロジェクトを **proj_cm4** として選択してください。
   d. **Peripheral** で **retarget-io** を選択し、**OK** をクリックしてくだてください。
   e. **Library Manager** ウィンドウから **Update** をクリックします。ライブラリが追加されるのを待ちましょう。進捗状況についてはログメッセージを参照してください。
   f. ライブラリが更新されたら、**Close** をクリックしてください。

Dual CPU Empty アプリケーションの CM0+プロジェクト (`proj_cm0p`) は CM4 コアを有効にし、ディープスリープに入ります。それ以外の目的でコアを使用するには、CM0+コアから `cybsp_init()`関数を呼び出す必要があります。この関数は、CM0+コア上でペリフェラルを動作させるために必要なシステムレベルの初期化をすべて行います。Code Listing 1 に示すように、`proj_cm0p` の `main.c` ファイルを修正してください。

**Code Listing 1**

```
#include "cy_pdl.h"
#include "cycfg.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 global interrupts */
  __enable_irq();

  /* Enable CM4. CY_CORTEX_M4_APPL_ADDR must be updated if CM4 memory layout is changed. */
  Cy_SysEnableCM4(CY_CORTEX_M4_APPL_ADDR);

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

インフィニオンの HAL は、CM0+と CM4 コアで同時に動作するように設計されていません。proj_cm4 はすでに HAL を使用しているため、proj_cm0p は PDL API だけを使うべきです。

### 4.1.1.4 ビルドとプログラム

アプリケーションは次の方法のいずれかによって構築およびプログラムできます。

- ModusToolbox™用の Eclipse IDE を使用
  1. プロジェクト エクスプローラから **Dual-CPU_Empty_PSoC6_App** アプリケーションを選択してください。
  2. クイックパネルから、**Dual-CPU_Empty_PSoC6_App Program MultiCore (KitProg3)**を選択してください。これで proj_cm0p と proj_cm4 の 2 つのプロジェクトが順番にビルドされ、プログラムされます。
  3. あるいは、個々のプロジェクト (proj_cm0p または proj_cm4) をプログラムすることもできますが、これは以下に説明する CLI インターフェースを通じてのみ可能です。
- コマンドラインインターフェース (CLI) を使用
  1. modus-shell を開き、**Dual-CPU_Empty_PSoC6_App** アプリケーションのルート ディレクトリに移動してください。
  2. make program コマンドを実行してください。これで proj_cm0p と proj_cm4 の 2 つのプロジェクトが順番にビルドされ、プログラムされます。
  3. あるいは、個別のプロジェクト (proj_cm0p または proj_cm4) をプログラムすることもできます。modus-shell を必要なプロジェクトディレクトリに移動してください。make program_proj コマンドを実行してください。

各プロジェクトの成果物は、build という名前のフォルダの下のプロジェクト ディレクトリに生成されます。アプリケーション レベルの build フォルダには、プロジェクトの hex ファイルのコピーがあることに注意してください。また、個々のプロジェクトの hex ファイルのデータを含む結合 hex ファイルを持っています。

フラッシュ オーバーフロー エラーによって、プロジェクトのビルドに失敗した場合は、リンカースクリプトのカスタマイズの手順に従って、各プロジェクトに必要なメモリを割り当ててください。

### 4.1.2 リンカースクリプトのカスタマイズ

CM0+および CM4 プロジェクトには、BPS から提供される、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+プロジェクト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。フラッシュ サイズの増減は、フラッシュ行サイズ (512 バイト) の倍数でなければならないことに注意してください。次の手順に変更点を示します。

1. CM0+リンカースクリプトは、bsps/TARGET_APP_<BSP_NAME>/COMPONENT_CM0P/ディレクトリにあります。
   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。
2. 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 0x0000**2000**<br>#define FLASH_SIZE 0x0000**2000** |
| GCC_ARM (*.ld) | ram (rwx) : ORIGIN = 0x08000000, LENGTH = 0x**2000**<br>flash (rx) : ORIGIN = 0x10000000, LENGTH = 0x**2000** |
| IAR (*.icf) | define symbol __ICFEDIT_region_IRAM1_end__ = 0x0800**1FFF**;<br>define symbol __ICFEDIT_region_IROM1_end__ = 0x1000**1FFF**; |

3. CM4 リンカースクリプトは、bsps/TARGET_APP_<BSP_NAME>/COMPONENT_CM4/ディレクトリにあります。
4. 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 0x0800**2000**<br>#define RAM_SIZE 0x000**FD800**<br>#define FLASH_CM0P_SIZE 0x**2000** |
| GCC_ARM (*.ld) | FLASH_CM0P_SIZE = 0x**2000**;<br>ram (rwx) : ORIGIN = 0x0800**2000**, LENGTH = 0x**FD800** |
| IAR (*.icf) | define symbol __ICFEDIT_region_IRAM1_start__ = 0x0800**2000**;<br>define symbol __ICFEDIT_region_IRAM1_end__ = 0x080**FF7FF**; |

5. 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
```

### 4.1.3 ライブラリと周辺機器の共有

各プロジェクトの deps フォルダには、そのプロジェクトで使用されるライブラリのリストが含まれます。前述の 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。

## 4 PSOC™ 6 MCU デュアルコアでの開発プロセス

デフォルトで Library manager を通して追加された BSP は、デュアル CPU アプリケーションのルート ディレクトリにある `bsps` という名前のフォルダにダウンロードされます。両方のプロジェクトはアプリケーションの BSP を共有します。

デフォルトでは、Library manager を通して追加されたライブラリは、デュアル CPU アプリケーションのルート ディレクトリの 1 つ上の階層にある `mtb_shared` という名前のフォルダにダウンロードされます。ライブラリは共有でもプロジェクト固有でも可能です。Library manager を通してライブラリを追加する際、そのライブラリが属するプロジェクトを選択するオプションが提供されます。同じライブラリが両方のプロジェクトに追加されている場合、両方のプロジェクトは `mtb_shared` ディレクトリから同じライブラリ コピーにアクセスできます。

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 の手順

PSOC™ 6 MCU デュアルコア 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。図 3 のような **Create Project** ダイアログが表示されます。

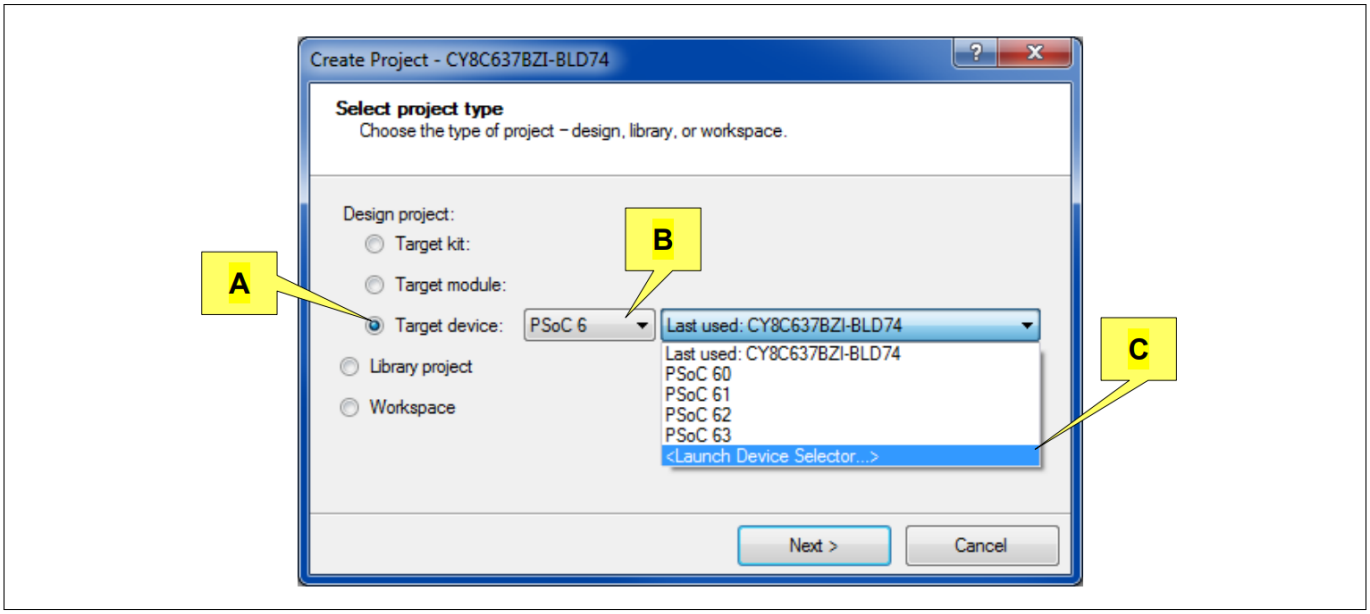


**図 3 PSOC™ Creator の Create Project ダイアログ**

**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。

図 4 に、Device Selector ダイアログを示します。デュアルコア 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 パイオニアキット CY8CKIT -062- BLE では、PSOC™ 6 MCU デュアルコア デバイスの部品番号は CY8C6347BZI-BLD53 です。PSOC™ 6 Wi-Fi Bluetooth® パイオニアキット CY8CKIT -062- WiFi -BT では、PSOC™ 6 MCU デュアルコア デバイスの部品番号は CY8C6247BZI-D54 です。

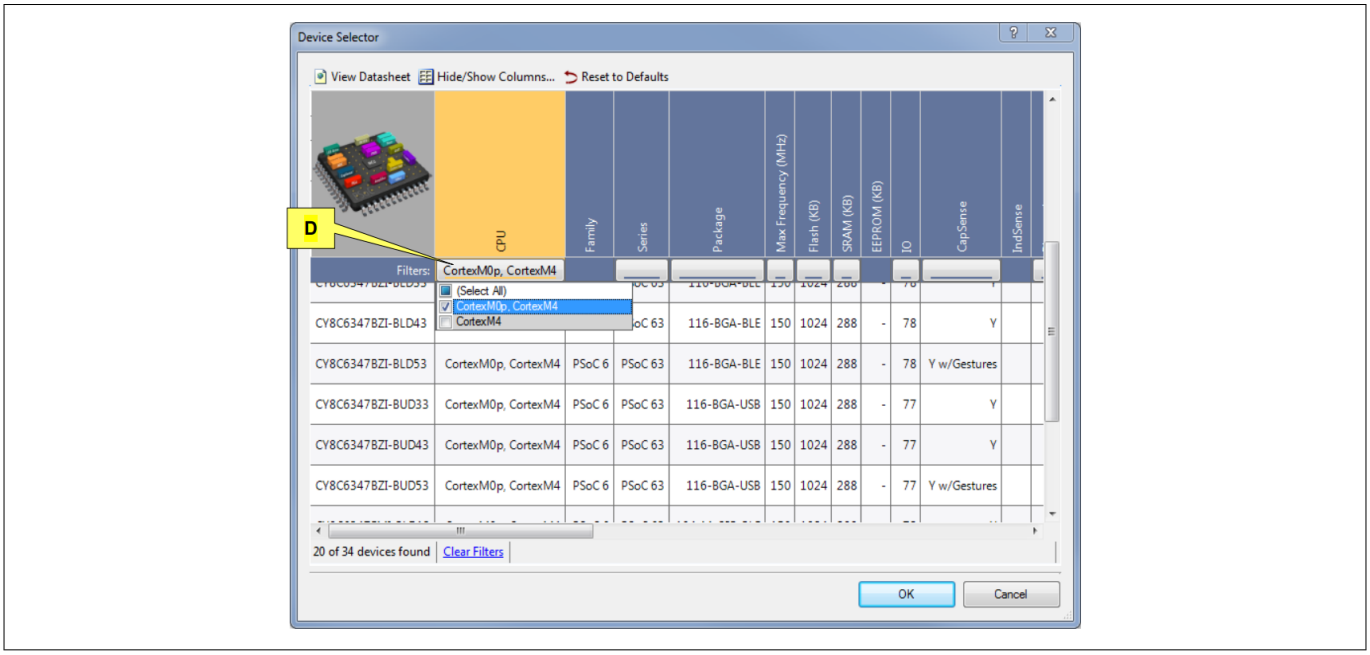


**図 4 PSOC™ Creator の Device Selector ダイアログ**

## 4 PSOC™ 6 MCU デュアルコアでの開発プロセス

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 (図 5) は、デュアルコア 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。

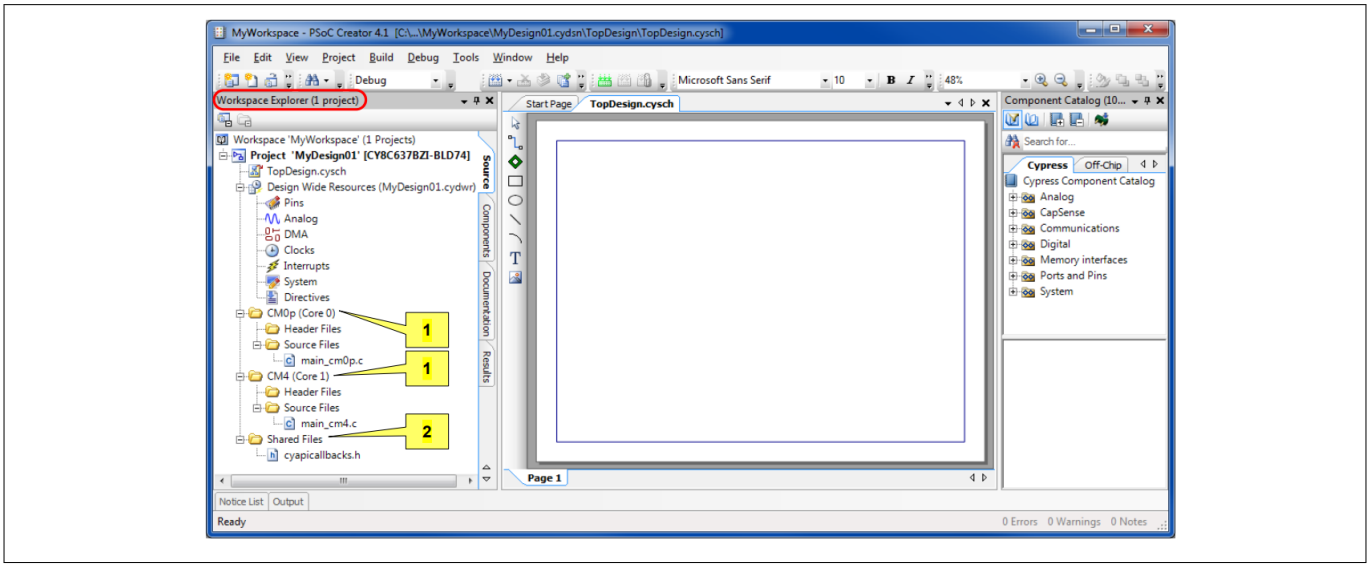


**図 5　　デュアルコア デバイスの PSOC™ Creator 初期プロジェクト レイアウト**

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、図 6 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インフィニオン 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 PSOC™ Creator の **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。

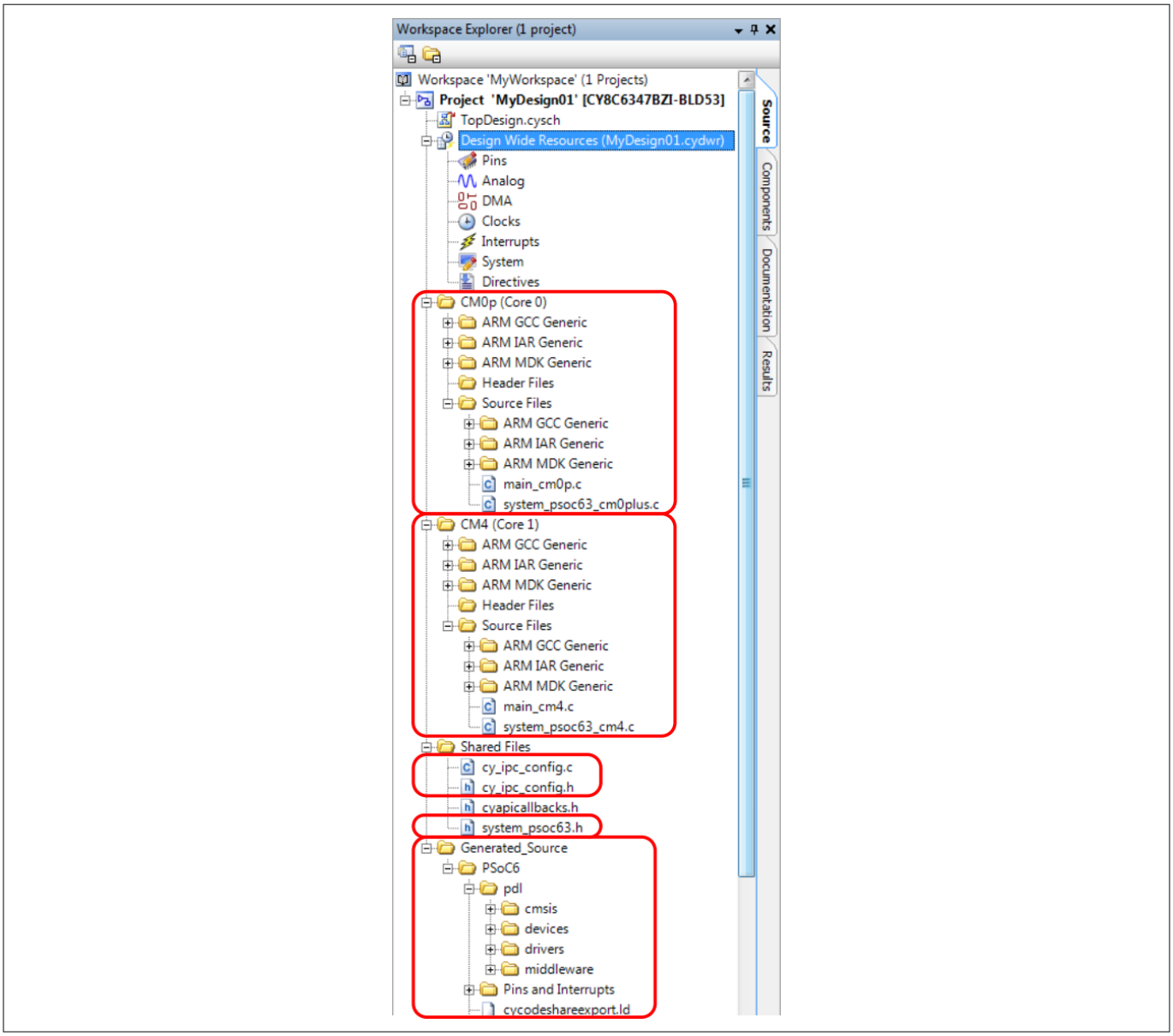


**図 6　　生成されたソースをプロジェクトに追加**

## 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。

*注: PSOC™ Creator をご使用のお客様は、プロジェクト作成ウィザードを使用して必要なサンプルをダウンロードしてください。PSOC™ Creator のサンプルコードへのリンクは、サンプルコードドキュメントへのリンクのみです。*

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。

リソースマネージメントを行うには以下の 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、図 7 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。

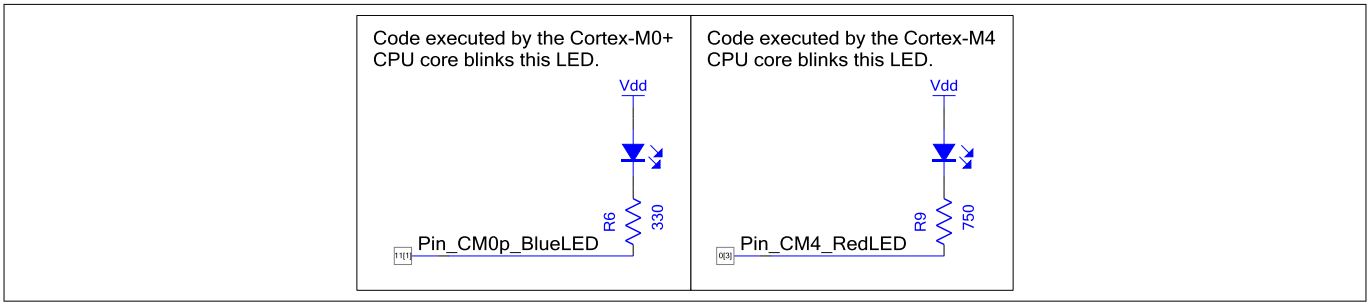


**図 7 個別ピン コンポーネントを制御するデュアル CPU の PSOC™ Creator プロジェクト回路図**

- **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 のリンカー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。

## 4.4 電力モード遷移の検討事項

システムクロックなどのリソースは、常に CPU とすべてのペリフェラル間で共有されます。`cybsp_init()`関数は `cybsp_register_sysclk_pm_callback()`関数を呼び出します。この関数は、クロックシステムをディープスリープモードに準備し、ディープスリープからのウェイクアップ時にクロックを復元する電源管理コールバックを登録します。

デフォルトの `cybsp_register_sysclk_pm_callback()`は、CM0+がディープスリープ状態にあるシングル CPU の例として設計されています。CM4 CPU もディープスリープになった場合、コールバック関数はクロックシステムを無効にします。すでにディープスリープに入っているため、これは CM0+の CPU には何の問題もありません。デュアル CPU の例では異なります。CPU の 1 つがディープスリープに入ると、コールバックは他の CPU の状態をチェックせず、クロックシステムを無効にします。これにより、他のアクティブな CPU が誤動作を起こします。

上記の問題を回避するためには、カスタム電源マネージメント コールバック関数が必要です。このカスタムコールバックでは、システムクロックを無効にするかどうかを決めることができます。また、電源モード移行を行う際に、アプリケーションに必要なその他のカスタム関数を実装できます。カスタム コールバックを正しい形式で記述するために、デフォルトのコールバック関数の実装を参照してください。

以下の手順に従って、カスタム電源マネージメント コールバック関数を作成し、呼び出してください。

1. デュアル CPU 例のルートから、ソースファイルを追加する共有フォルダを作成してください。

   例: `shared/source/`
2. 任意の名前で C ファイルを作成してください。

   例: `shared/source/custom_pm_callback.c`
3. ファイルを開き、以下のように全く同じ定義と名前でカスタム コールバック関数を書き込んでください。

```
cy_rslt_t cybsp_register_custom_sysclk_pm_callback(void)
{
. . .
}
```

4. CM0p と CM4 の両方のプロジェクトの Makefile を開いてください。以下の変更を行ってください。
   a. SOURCES 変数を編集して、新しく作成した C ファイルをビルド プロセスに追加してください。

      `SOURCES= ../shared/source/custom_pm_callback.c`
   b. DEFINES 変数に `CYBSP_CUSTOM_SYSCLK_PM_CALLBACK` マクロを追加してください。

## 4.5 IPC 構成の検討事項

プロセス間通信 (IPC) は複数の CPU が通信を行い、アクティビティを同期させるための機能を提供します。以下は、IPC を使用する際に考慮すべきガイドラインの一部です。

1. IPC 通信に使われる共有メモリは、保護されていないパブリック RAM 領域に置くことができます。この `public_ram` 領域は、ターゲット BSP が提供するデフォルトの `CM0+`リンカースクリプトで定義され、コメントされています。このパブリック RAM 領域を使用するには、`CM0+`リンカースクリプトの `public_ram` 行と`.cy_sharedmem` セクションのコメントを外してください。
2. SRAM を除外してフラッシュ書込み関数を実行するためには、`public_ram` 領域に実行可能な特権を与える必要があります。これは、`CM0+`リンカースクリプトの `public_ram` 行の `rw` パラメータを `rwx` パラメータに変更することで実現できます。
3. `public_ram` 領域が使用されない場合、共有メモリは CM0+ CPU 用に定義された SRAM 領域に割り当てられます。
4. 保護ユニットを使用する場合、ユーザー アプリケーションは共有メモリ領域を保護しないように注意してください。

## 4.6 割込み割当ての検討事項

デュアルコア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 割込み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/HAL 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 Bluetooth® LE 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。インフィニ

オン HAL を使用している場合、HAL ドライバが必要とするすべての割り込みは、HAL 関数を呼び出す CPU に割り当てられます。

*注: CY8C61x6/7、CY8C62x6/7、および CY8C63xx デバイスでは、CM0+ CPU は 32 個のシステム割り込みしか処理できず、そのうちディープスリープ可能なのは 8 個だけです。HAL 関数が、割り込みを必要とする周辺機器の初期化に使用されている場合、すべての割り込みラインが完全に使用されると、関数は "none free error "を返します。*

**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。図 8 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(図 9 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。その割込みは図 8 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。

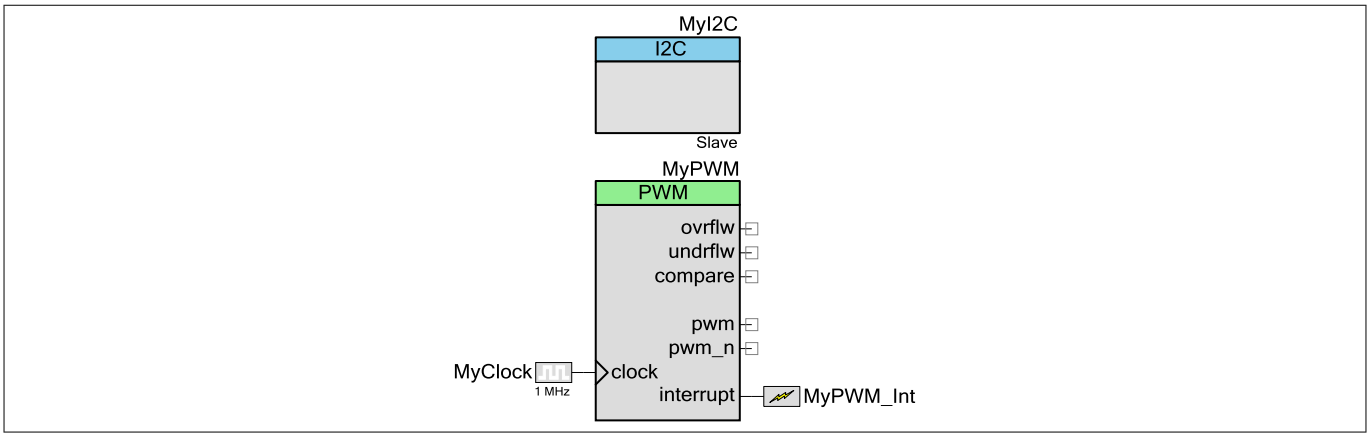


**図 8 2 つの割込みを持つ回路設計の例**

**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。

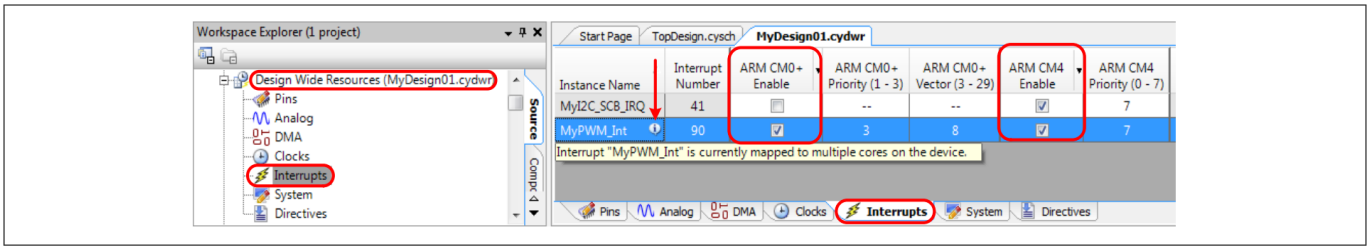


**図 9 CPU への割込み割当て**

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。

*注: 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。*

## 4.7 デバッグに関する検討事項

Keil µVision や IAR Embedded Workbench などのサードパーティ IDE は、デュアルコア デバッグに対応します。ModusToolbox™ソフトウェア用の Eclipse IDE は、デュアルコア デバッグ (非同期) に対応します。PSOC™ Creator 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。

### 4.7.1 ModusToolbox™の手順

ModusToolbox™ソフトウェアの Eclipse IDE は、非同期デュアルコアデバッグをサポートしています。両方のコアのデバッグ セッションを同時に起動することはできますが、一方のコアでのデバッグ アクティビティ (例: 一時停止、停止、ブレークポイント) がもう一方のコアには影響しなくなります。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。

### 4.7.2 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、図 10 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。

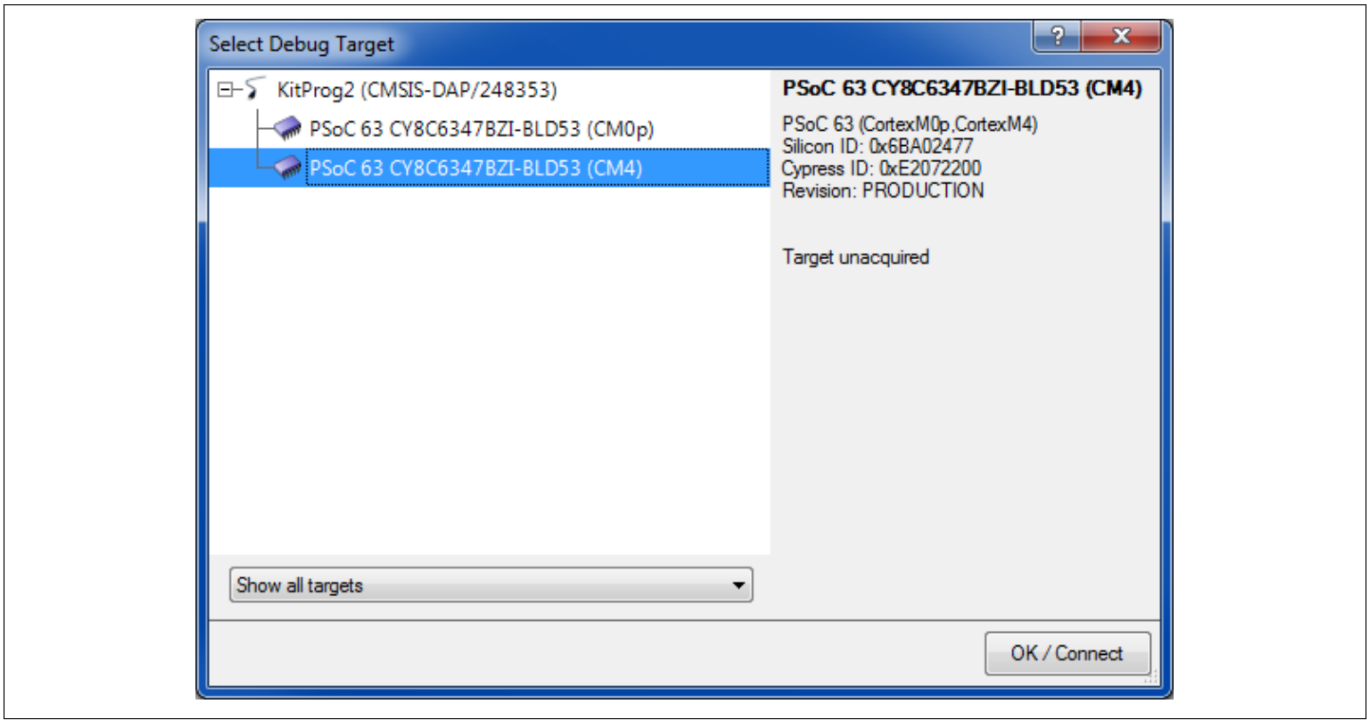


**図 10 PSOC™ Creator のデバッグ用 CPU 選択**

**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。

### 4.7.3 他の IDE の手順

#### 4.7.3.1 ModusToolbox™アプリケーションをサポート IDE へエクスポート

ModusToolbox™ソフトウェアには、次の IDE 上でアプリケーションをロードして実行するためのメカニズムがあります。

- IAR Embedded Workbench
- Keil µVision
- Visual Studio Code

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。

#### 4.7.3.2 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、デュアルコア 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**

図 11 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。

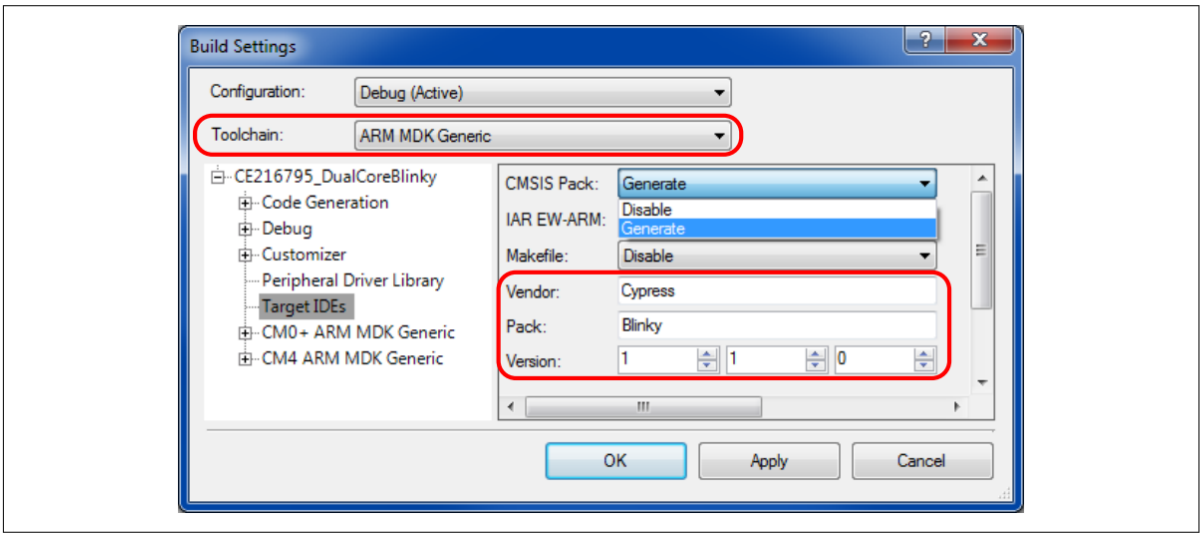


**図 11 ターゲット IDE のビルド設定**

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。図 12 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。

*注: 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。*

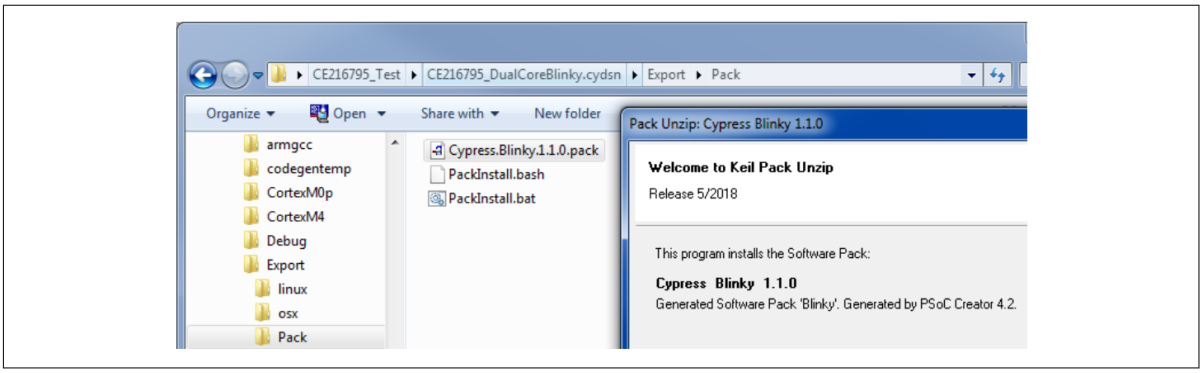


**図 12 PSOC™ Creator プロジェクトから µVision パックのインストール**

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 (図 11 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 - 生成コードの IDE へのインポート (PSOC™ 6 MCU プロジェクト用)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 (例: `µVisionBuild`)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、図 13 に示すように、CM4 オブジェクトファイル (例: `ObjectsM4`) 用の別の新しいフォルダを作成してください。

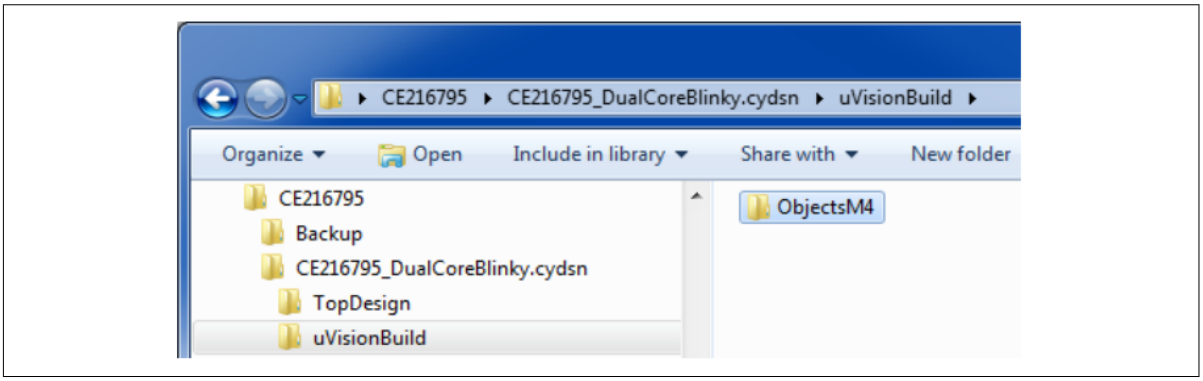


**図 13 µVision プロジェクトの新しいフォルダ**

µVision 5.25 以降を開き、`µVisionBuild` 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:** 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 デュアルコア点滅プロジェクトの場合、図 14 のように、CM0+ CPU 用の µVision プロジェクト `BlinkyM0p` を作成します。

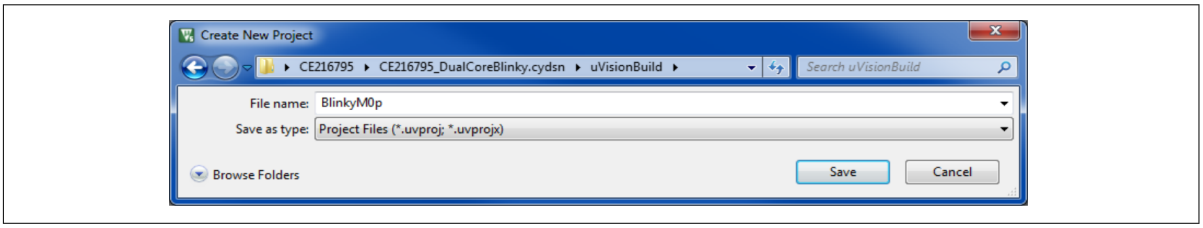


**図 14 CM0+用 µVision プロジェクトの生成**

**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 (図 12) で定義された 2 つの PSOC™ 6 MCU CPU が表示されます。図 15 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。

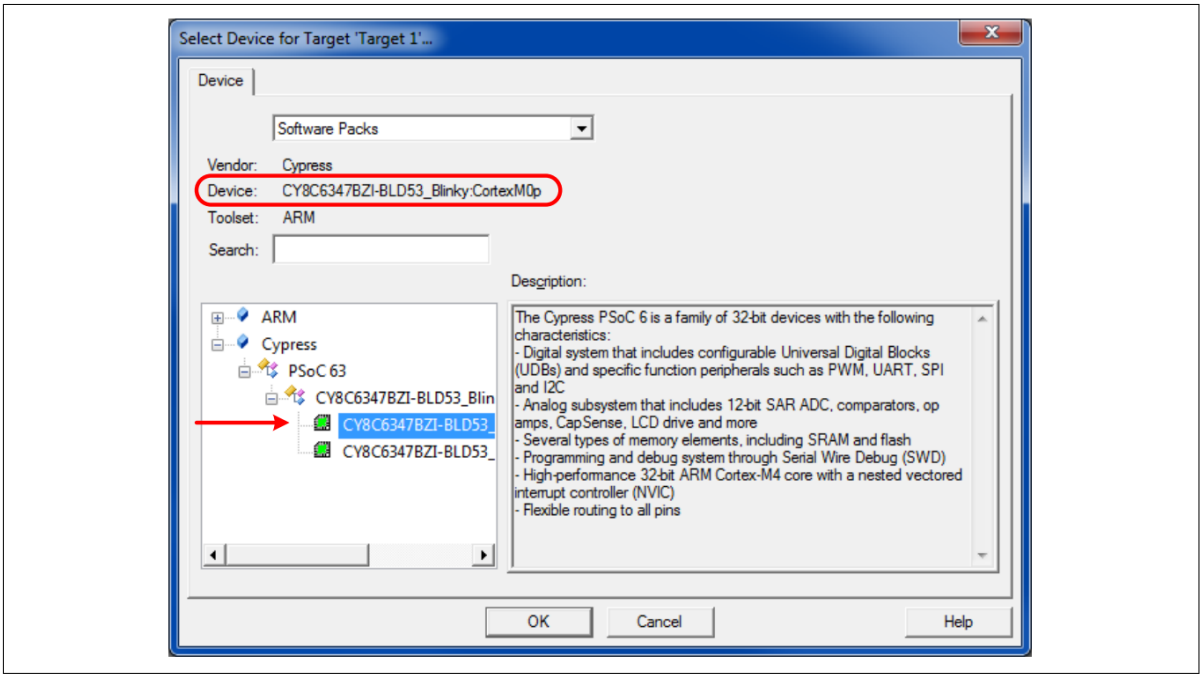


**図 15 プロジェクトデバイスとして CM0+を選択**

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。図 16 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。

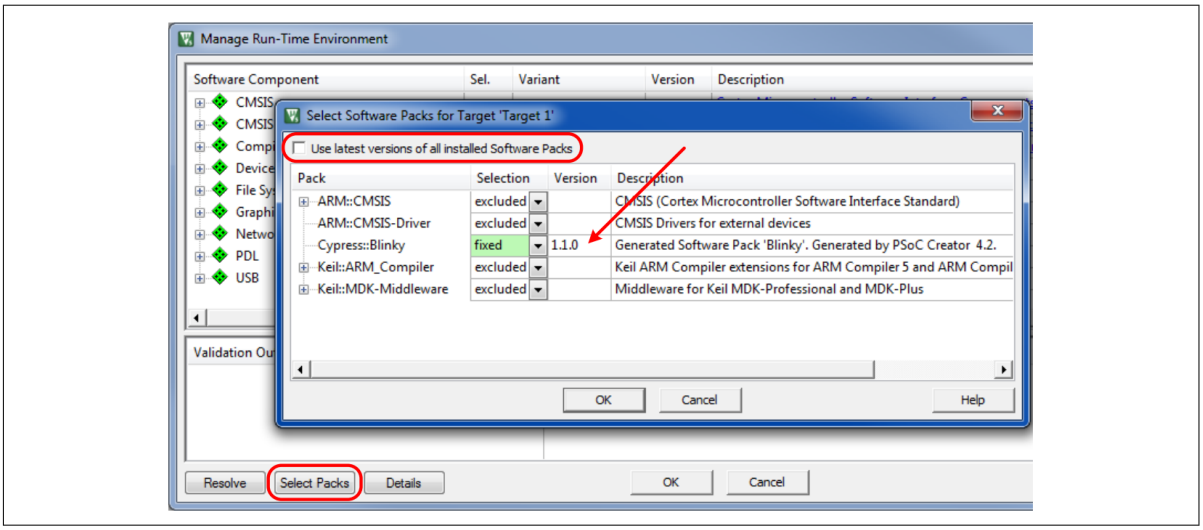


**図 16　　PSOC™ Creator プロジェクトパックの選択**

**OK** をクリックしてください。図 17 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。図 18 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。

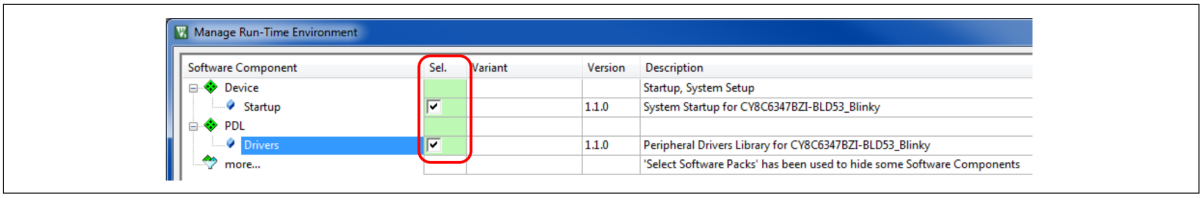


**図 17　　パック開始と PDL ドライバーファイルの選択**

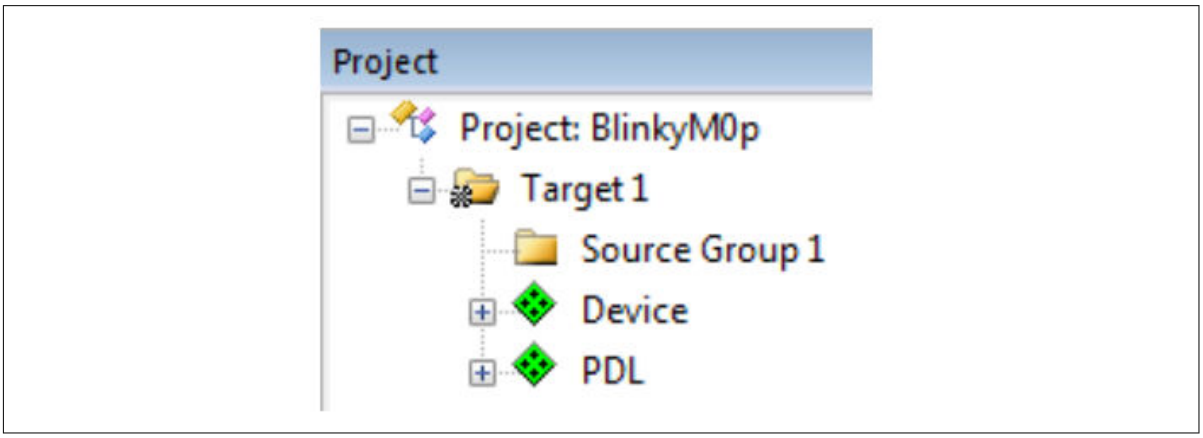


**図 18　　初期プロジェクト生成**

**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、図 19 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。

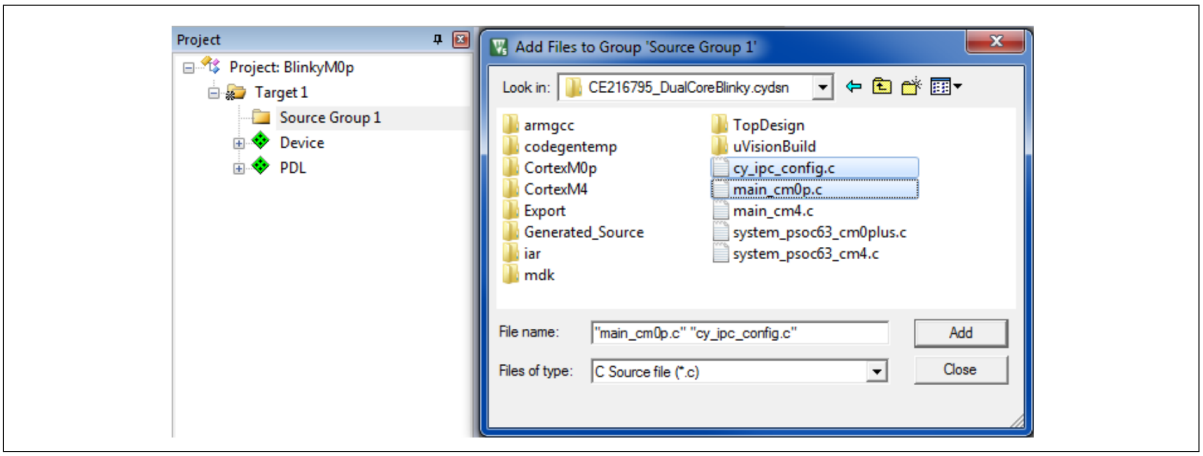


**図 19　　PSOC™ Creator プロジェクト C ソースファイルのソースグループへの追加**

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。図 20 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。

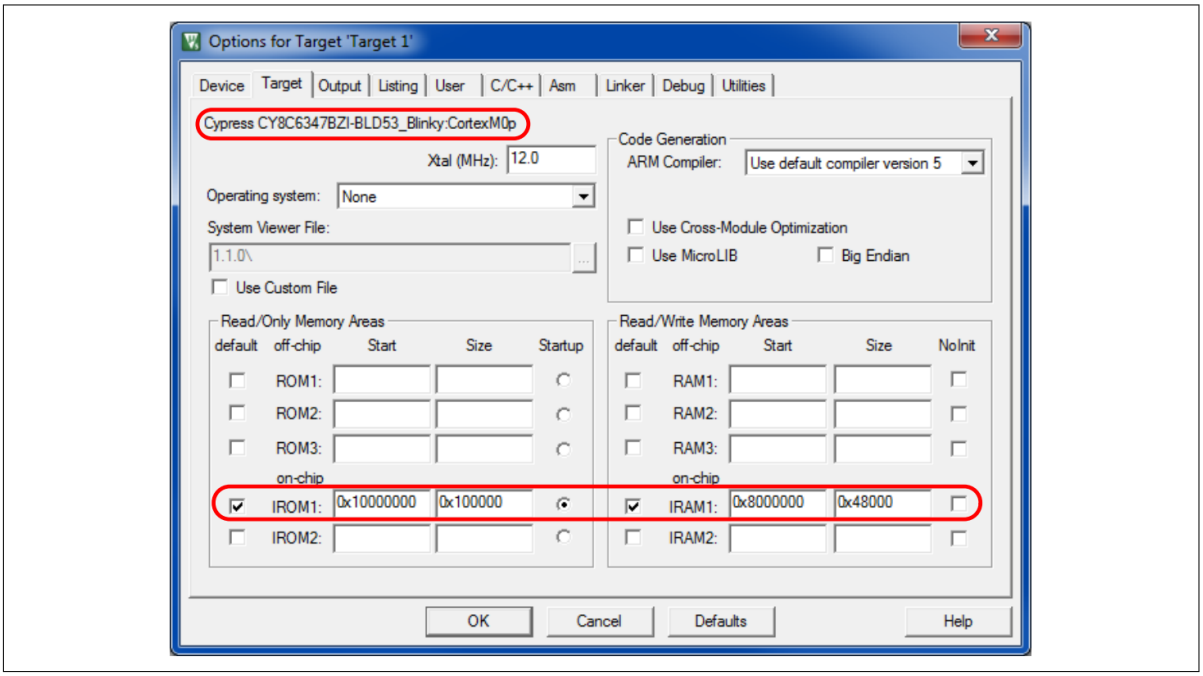


**図 20　　プロジェクトターゲットオプション**

**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。図 21 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。

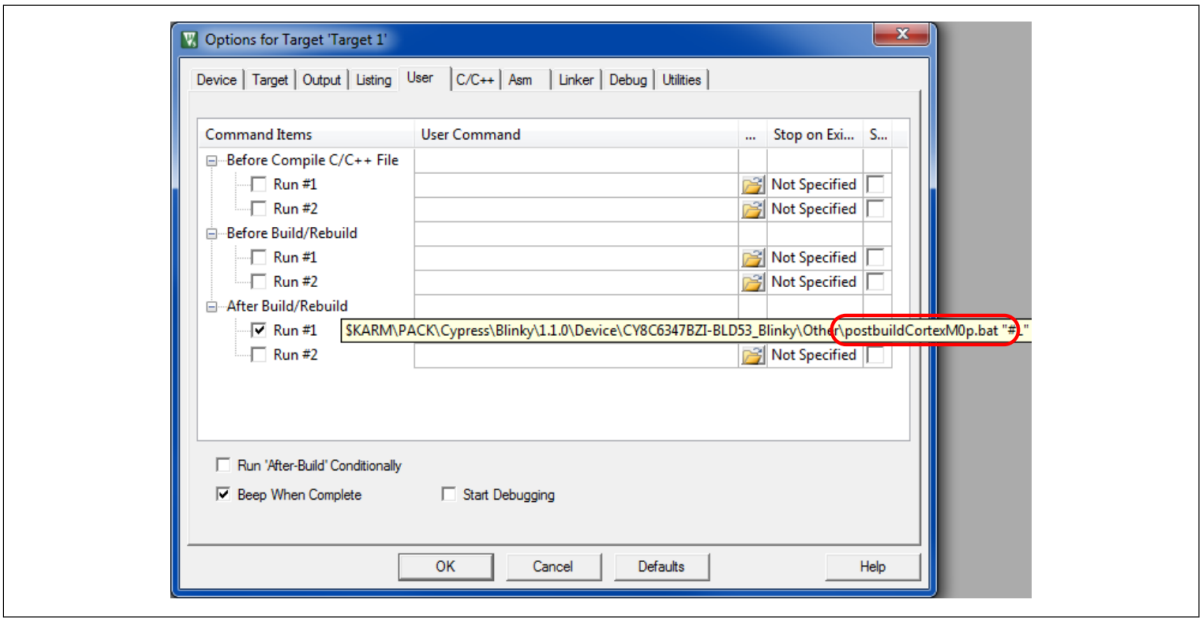


**図 21　　プロジェクトユーザーオプション**

図 22 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。

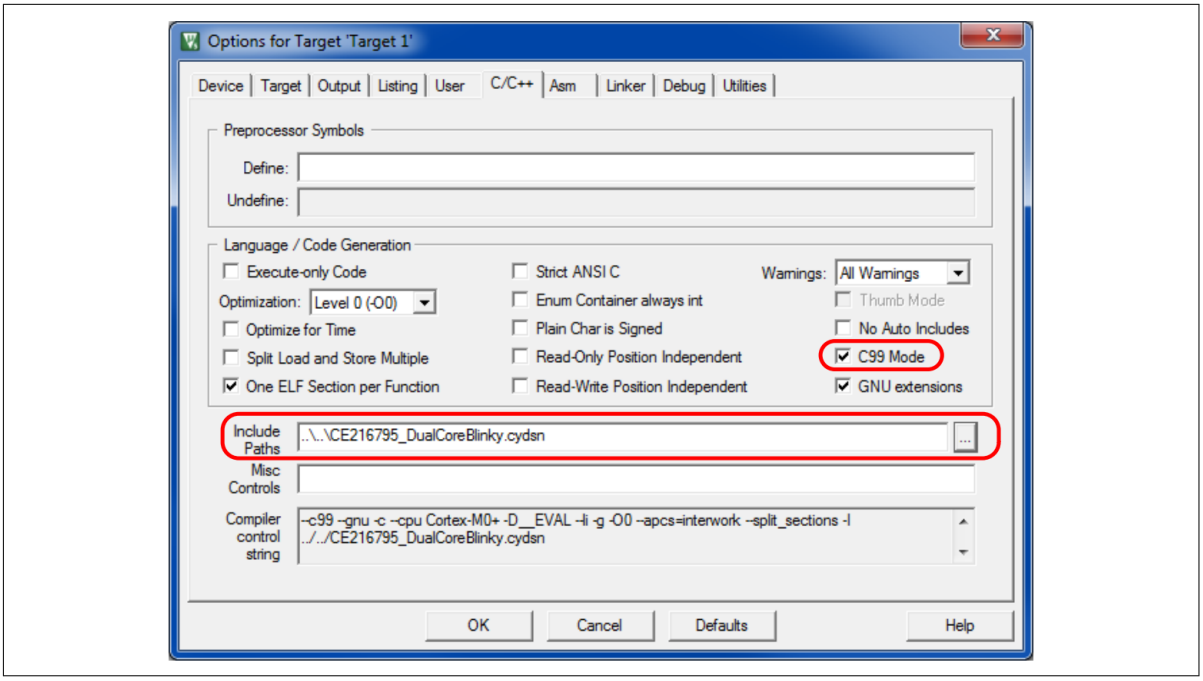


**図 22　　プロジェクト C/C++オプション**

図 23 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。

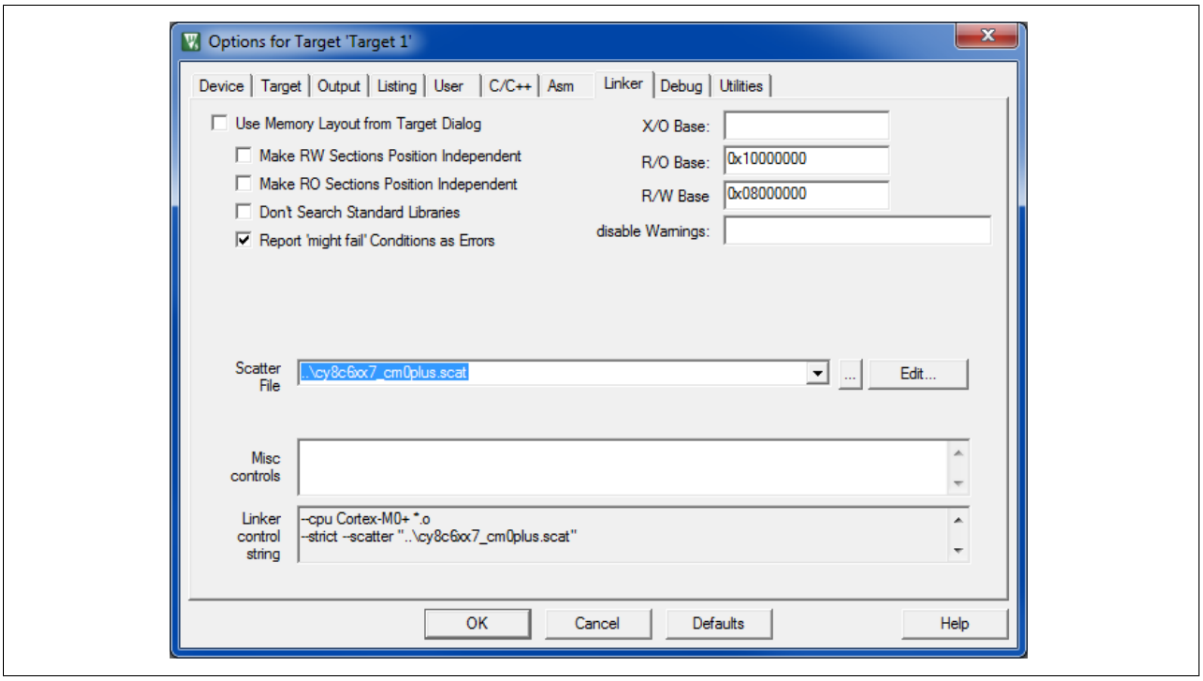


**図 23　　プロジェクトリンカーオプション**

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。

図 24 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。

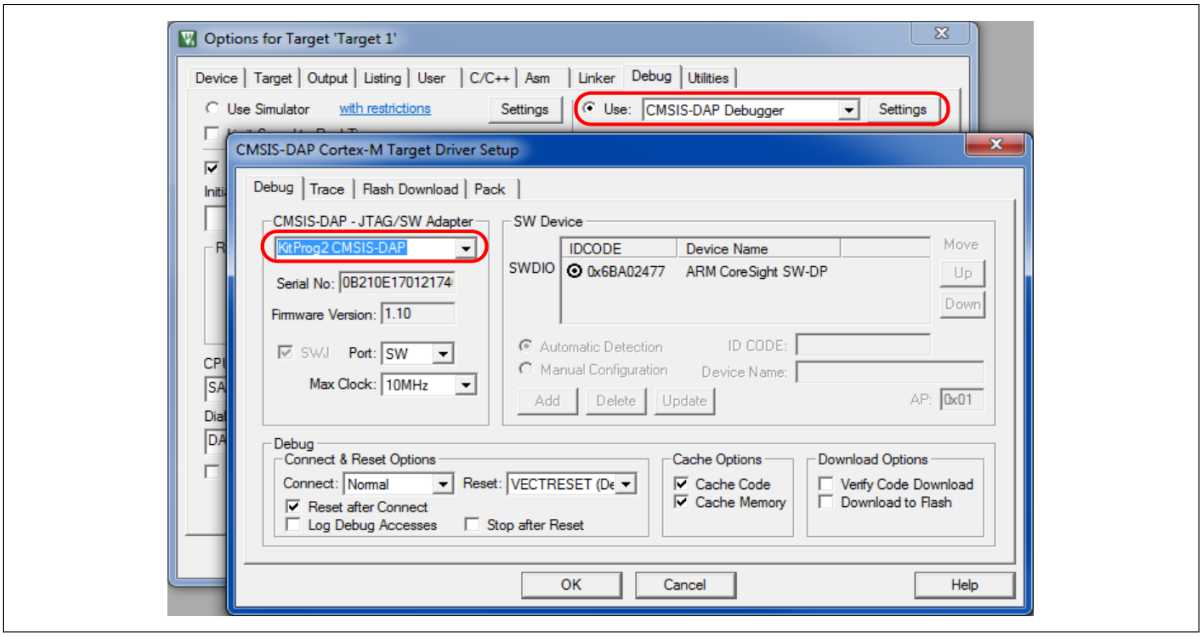


**図 24　　プロジェクトデバッグオプション**

**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。図 25 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。

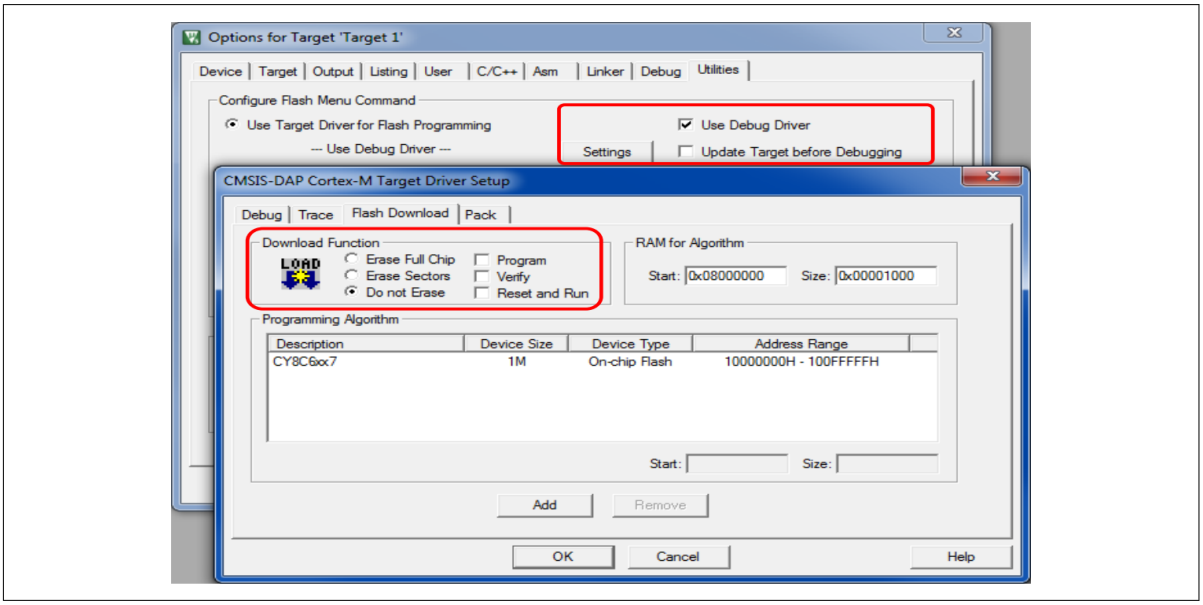


**図 25　　プロジェクトユーティリティオプション**

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 デュアルコア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。図 14 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 `µVisionBuild` です。図 14 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。図 15 を参照してください。
- 自身の PSOC™ Creator プロジェクトフォルダに移動し、図 19 に示すように `main_cm4.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, 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。図 13 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。図 22 を参照してください。
- 図 26 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。

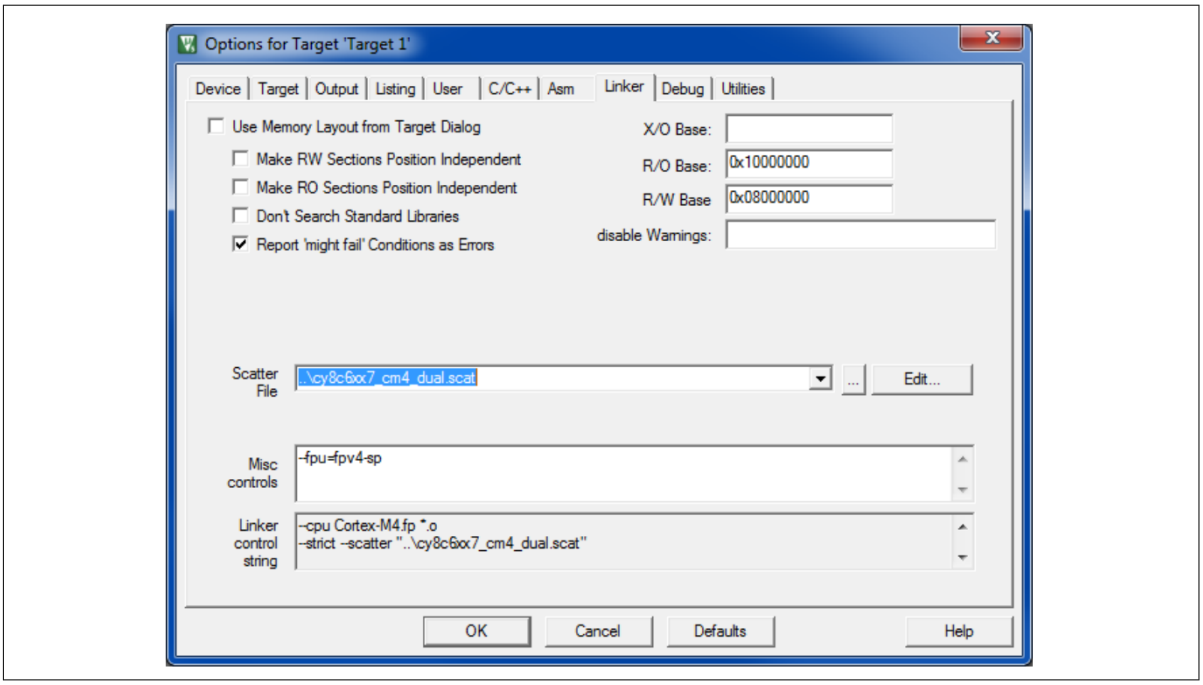


**図 26　　CM4 プロジェクト用リンカーオプション**

- **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。図 24 を参照してください。
- 図 27 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。

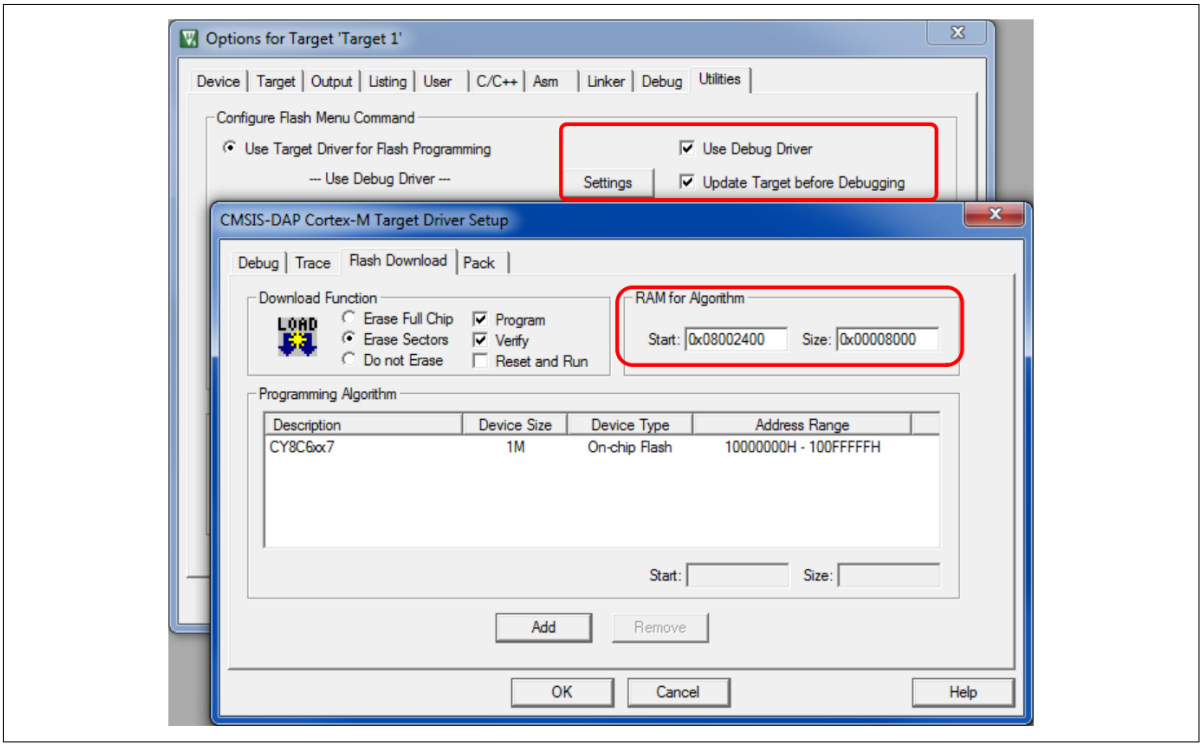


**図 27　　CM4 プロジェクト用ユーティリティオプション**

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、図 28 のようになります。

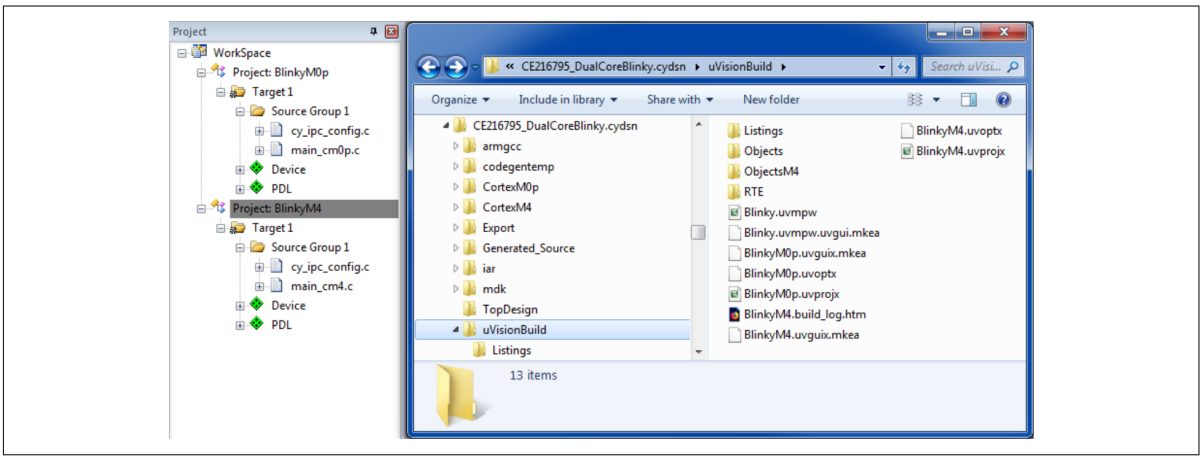


**図 28　　プロジェクトを順番にビルドしてください。**

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 (**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 (図 27 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。

***注**:　　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、図 29 のようになります。

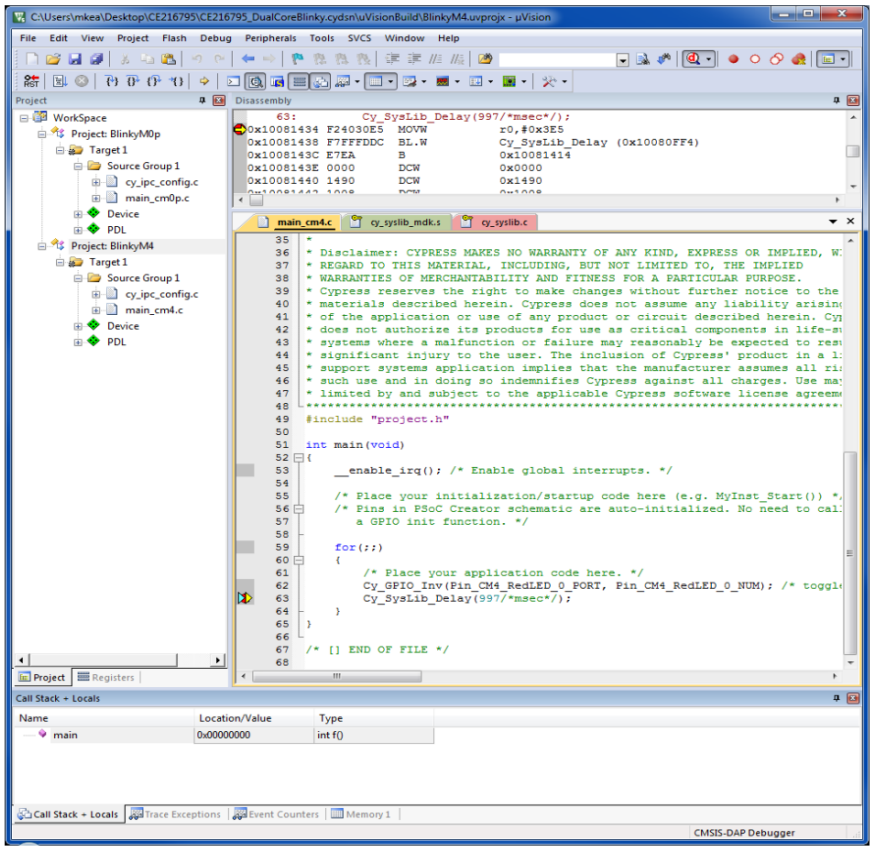


**図 29　　CM4 デバッグウィンドウ**

CE216795 デュアルコア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。

*注:　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、図 30 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。

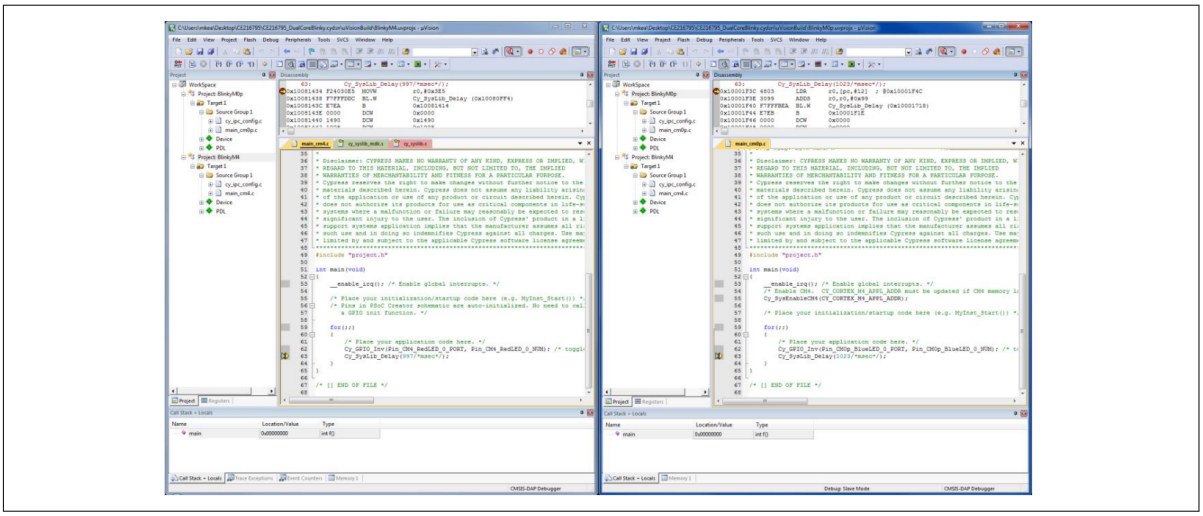

**図 30　　µVision デュアルコア デバッグ**

### 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。

***注**:　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 (図 31) 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。

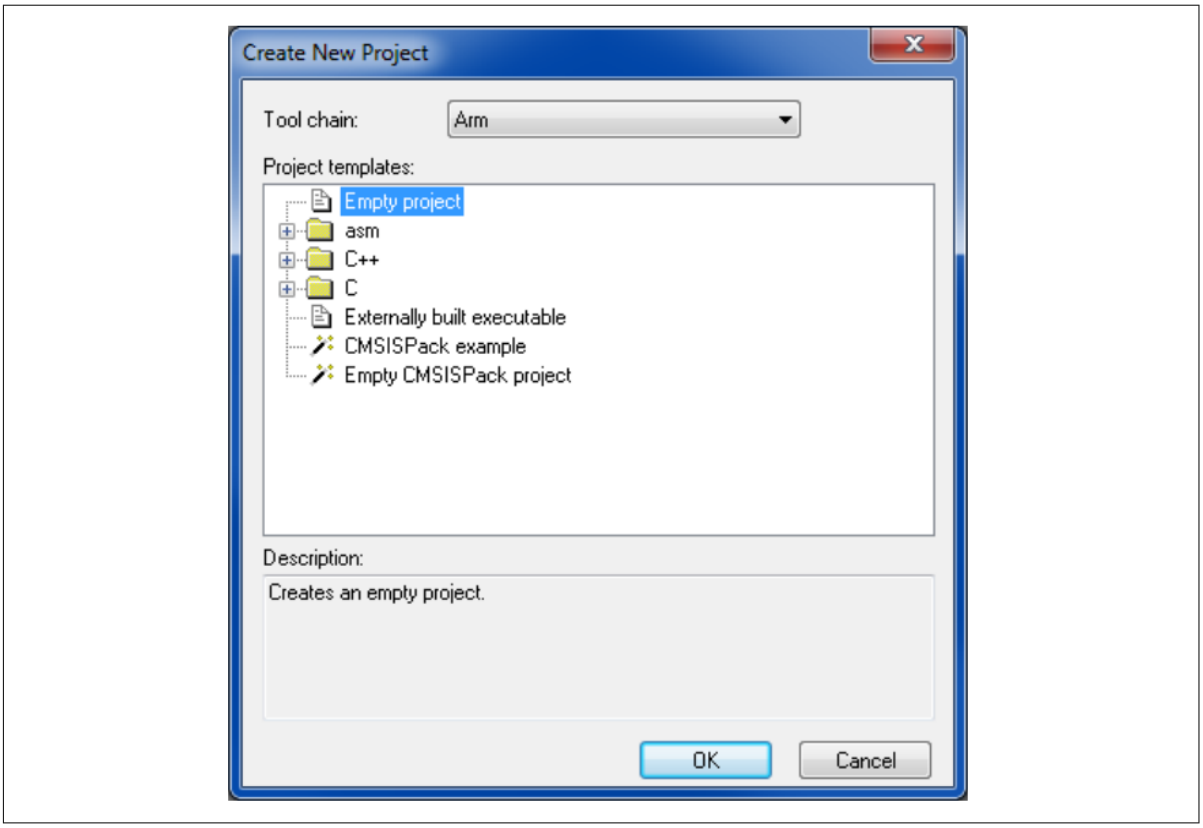


**図 31　　IAR Embedded Workbench の新規プロジェクトの作成ダイアログ**

**推奨:** ダイアログ (図 32) 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 デュアルコア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。

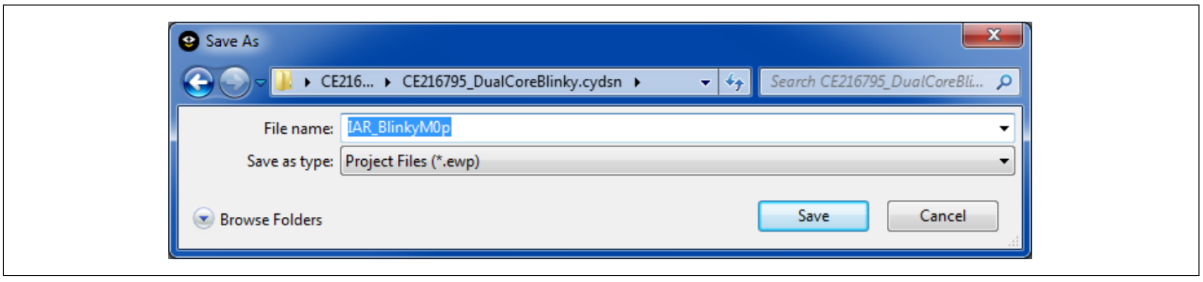


**図 32　　CM0+用 IAR Embedded Workbench プロジェクトの生成**

**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、図 33 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。

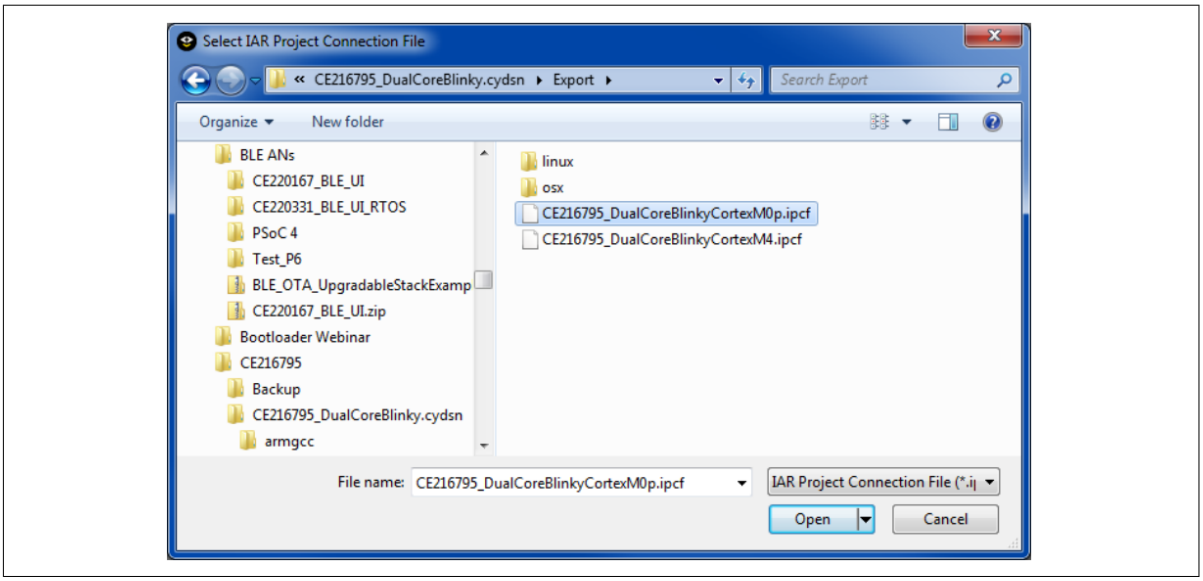


**図 33　　PSOC™ Creator プロジェクトエクスポートフォルダから IAR プロジェクト接続ファイルの選択**

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。図 34 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。

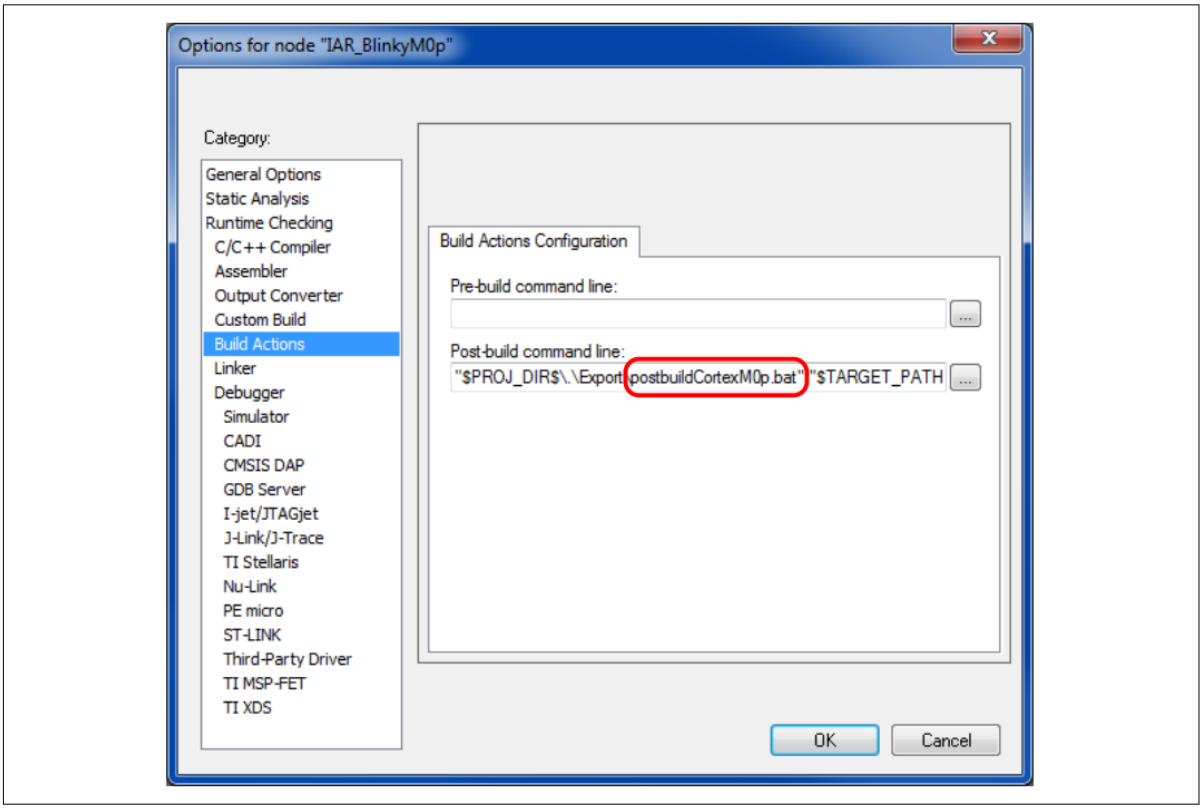


**図 34　　PSOC™ Creator ポストビルドバッチファイルの選択**

**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 デュアルコア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。図 32 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。図 32 を参照してください。
- `...CortexM4.ipcf` ファイルを選択してください。図 33 を参照してください。
- 図 35 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。

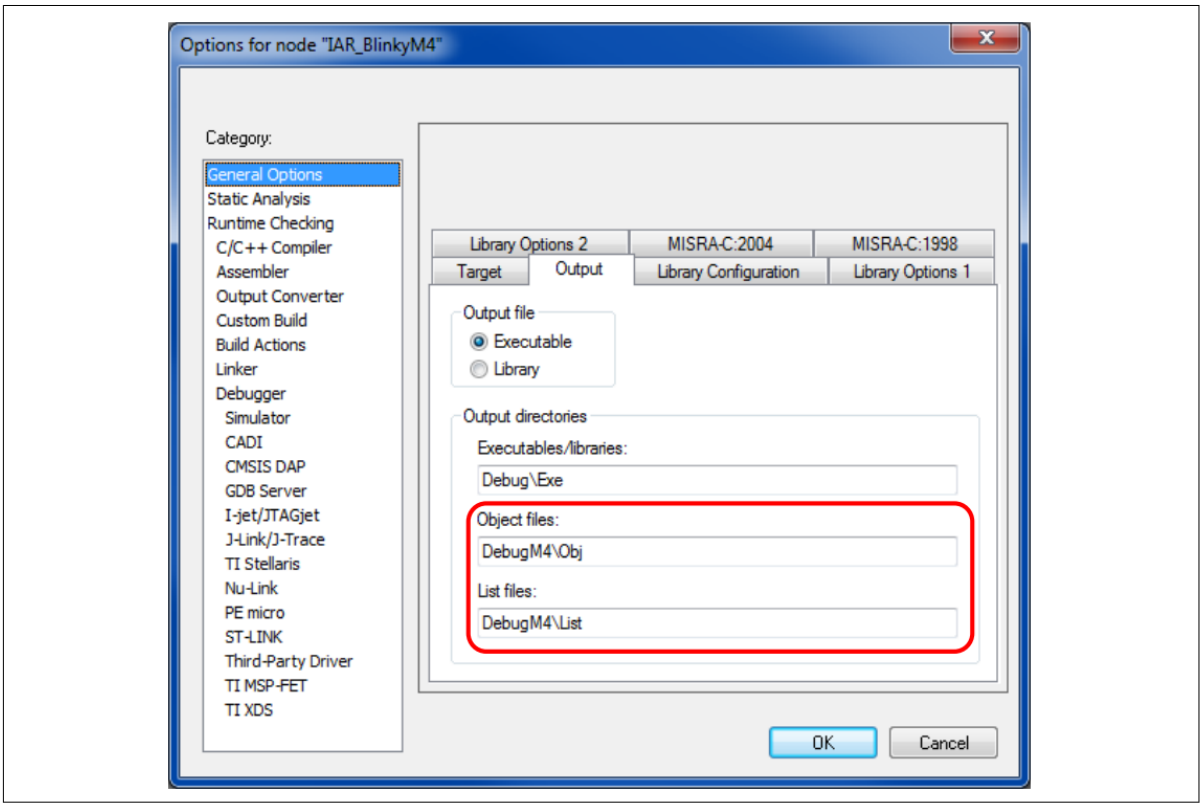


**図 35　　CM4 プロジェクト用固有出力フォルダ**

- **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。図 34 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。

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、図 36 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。

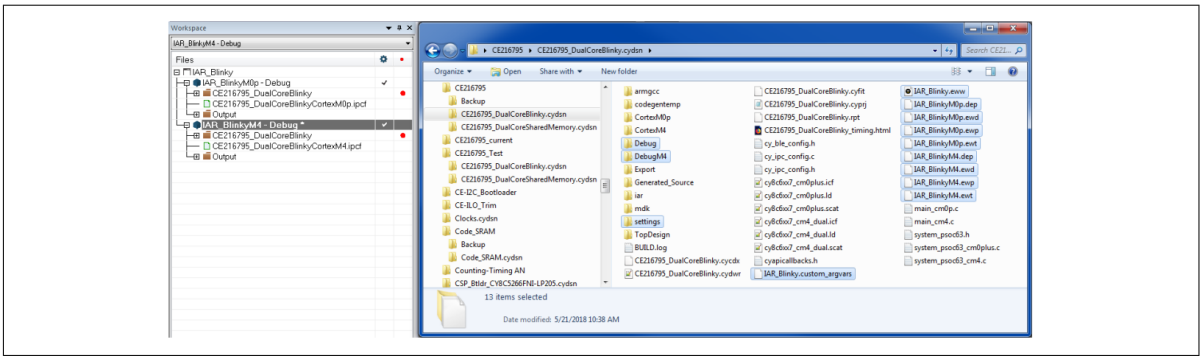

**図 36　　IAR Embedded Workbench プロジェクトウィンドウとプロジェクトファイルの結果**

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。

***注**:　　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 (図 37 を参照) 。*

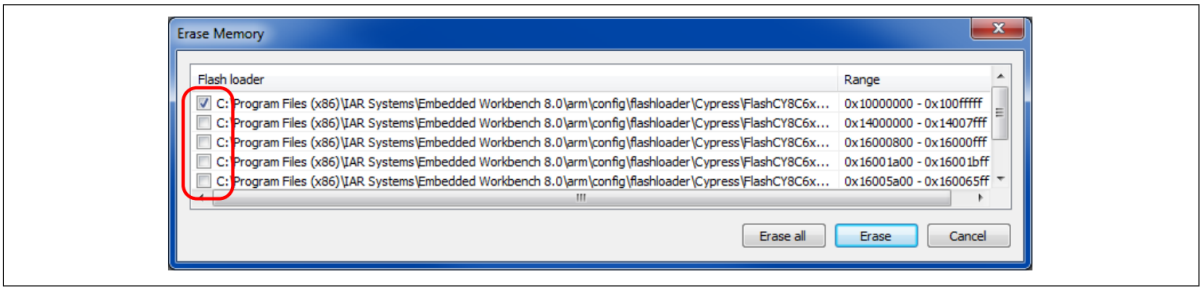


**図 37　　IAR PSOC™ 6 MCU 用 Embedded Workbench 消去メモリダイアログ**

***注**:　　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。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、例えば 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、図 38 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。

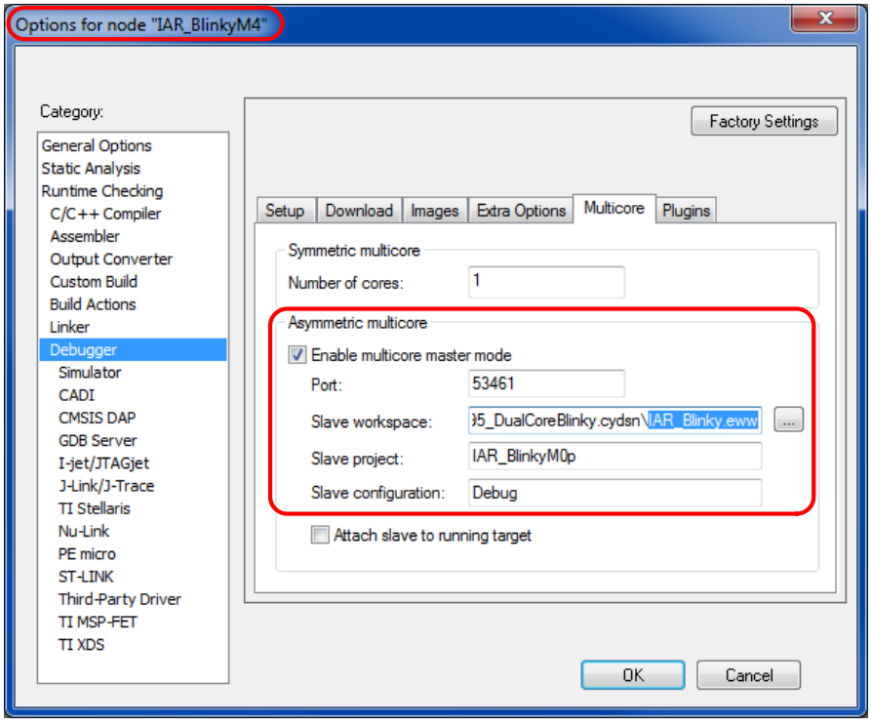


**図 38　　マルチコアデバッグの設定**

**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、図 39 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。

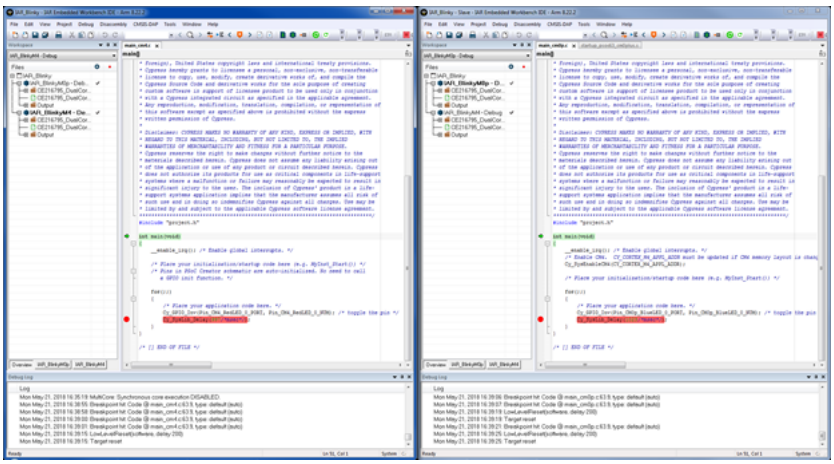

**図 39　　IAR Embedded Workbench デュアル CPU デバッグ**

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。

## 4.8 接続アプリケーション設計の検討事項

ModusToolbox™コネクティビティ、Wi-Fi、および Bluetooth®ミドルウェアライブラリは GitHub から入手できます。これらのライブラリは、PSOC™ 6 MCU とインフィニオンの AIROC™ Wi-Fi および Bluetooth®コンボデバイスを使用したワイヤレス IoT アプリケーションの開発に使用できます。ユーザーは、必要なミドルウェアやドライバ ライブラリとともに接続機能を備えたアプリケーションを実行するために、Arm® CM0+または CM4 コアのいずれかを選択できます。スループット (データ転送速度: ビット/秒) と PSOC™ 6 MCU の消費電力 (単位: ミリアンペア/マイクロアンペア) は、接続アプリケーションのコアを決定する際に考慮すべき重要な要素です。関連する性能要件は、接続機能を備えたアプリケーションに使用する最適なコアを決定する上で重要な役割があります。以下は、その分析結果です。

### 4.8.1 スループット分析

下図は、スループットの測定に使用したテストセットアップを示します。CY8CPROTO-062S2-43439 キットは、TCP クライアント モードで iperf アプリケーションを実行するように設定されています。キットは USB 経由で ModusToolbox™が動作する PC に接続し、プログラミングと iperf ログ表示を行います。KitProg3 USB-UART ブリッジは、ログ表示用の Tera Term ソフトウェアと共に使用します。PC はサーバーモードで iperf アプリケーションも実行します。キットと PC は同じ Wi-Fi ルーターに接続されています。テスト セットアップは、温度/無響室内ではない、通常の環境設定です。

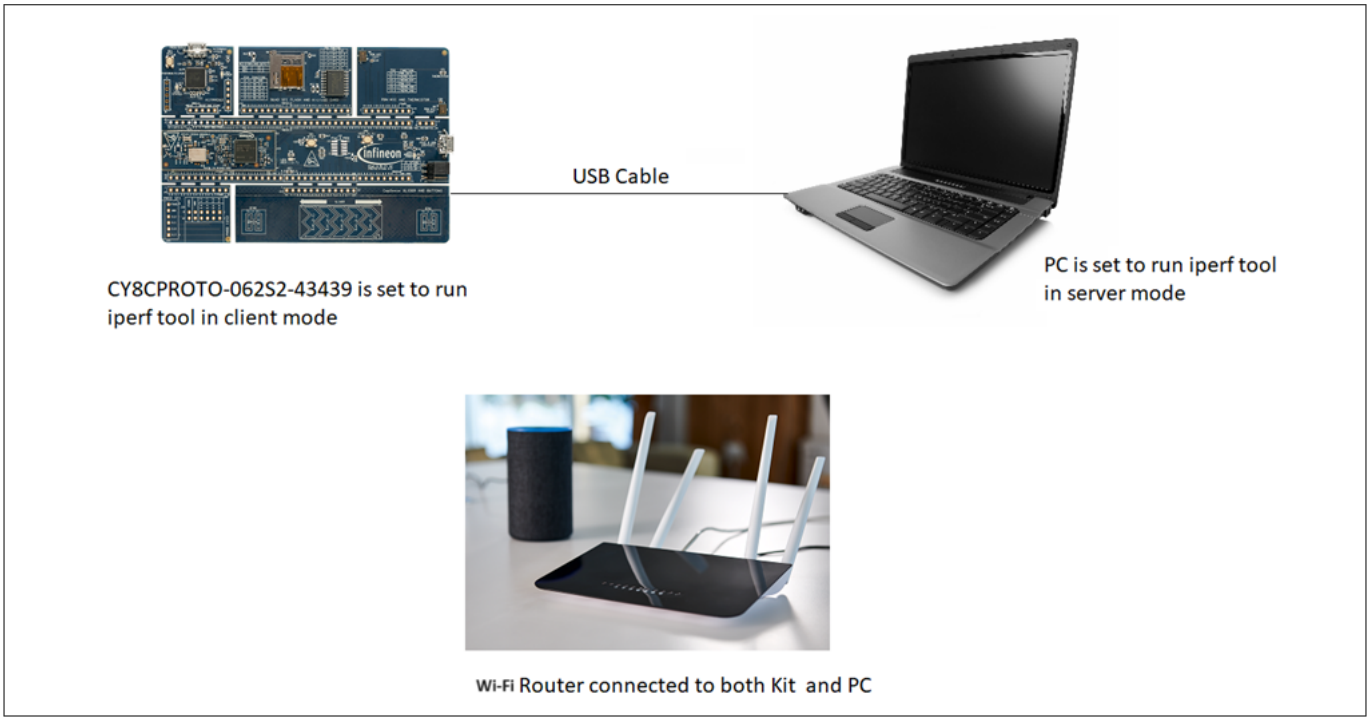


**図 40 スループット測定のセットアップ**

以下の表は、ホーム Wi-Fi ネットワークにおいて、システム アイドル パワーモードをアクティブに設定したデフォルトの BSP 設定で達成されたスループットを示します。

**表 1 スループットの詳細**

| シナリオ | スループット (メガビット/秒, Mbps) |
|---|---|
| CM0+コア スリープ、CM4 コア アクティブ @100 MHz | ピーク= 21.9、標準= 21.2 |
| CM4 コア スリープ、CM0+コア アクティブ @100 MHz | ピーク= 16.2、標準= 15.9 |

以下は、キットと PC の両方から出力された iperf 測定のスクリーンショットです。

### CM4 の実行

```
> iperf -c 192.168.1.11 -p 50007 -t 20 -i 1
> ------------------------------------------------------------
Client connecting to 192.168.1.11, TCP port 50007
TCP window size: 29.3 KByte (default)
------------------------------------------------------------
[ ID] Interval       Transfer     Bandwidth
[  0]  0.0- 1.0 sec  2.58 MBytes  21.7 Mbits/sec
[  0]  1.0- 2.0 sec  2.46 MBytes  20.6 Mbits/sec
[  0]  2.0- 3.0 sec  2.40 MBytes  20.1 Mbits/sec
[  0]  3.0- 4.0 sec  2.56 MBytes  21.5 Mbits/sec
[  0]  4.0- 5.0 sec  2.53 MBytes  21.3 Mbits/sec
[  0]  5.0- 6.0 sec  2.55 MBytes  21.4 Mbits/sec
[  0]  6.0- 7.0 sec  2.48 MBytes  20.8 Mbits/sec
[  0]  7.0- 8.0 sec  2.47 MBytes  20.7 Mbits/sec
[  0]  8.0- 9.0 sec  2.40 MBytes  20.1 Mbits/sec
[  0]  9.0-10.0 sec  2.57 MBytes  21.6 Mbits/sec
[  0] 10.0-11.0 sec  2.54 MBytes  21.3 Mbits/sec
[  0] 11.0-12.0 sec  2.54 MBytes  21.3 Mbits/sec
[  0] 12.0-13.0 sec  2.51 MBytes  21.1 Mbits/sec
[  0] 13.0-14.0 sec  2.56 MBytes  21.5 Mbits/sec
[  0] 14.0-15.0 sec  2.59 MBytes  21.7 Mbits/sec
[  0] 15.0-16.0 sec  2.52 MBytes  21.1 Mbits/sec
[  0] 16.0-17.0 sec  2.61 MBytes  21.9 Mbits/sec
[  0] 17.0-18.0 sec  2.52 MBytes  21.1 Mbits/sec
[  0] 18.0-19.0 sec  2.54 MBytes  21.3 Mbits/sec
[  0] 19.0-20.0 sec  2.58 MBytes  21.6 Mbits/sec
[  0]  0.0-20.0 sec  50.5 MBytes  21.2 Mbits/sec
```

**図 41　　CM4 コア搭載キットで測定されたスループット**

```
PS C:\DATA\Apps\iperf> .\iperf-win-isoch-fix.exe -s -w 64k -p 50007
------------------------------------------------------------
Server listening on TCP port 50007
TCP window size: 62.5 KByte
------------------------------------------------------------
[  1] local 192.168.1.11 port 50007 connected with 192.168.1.7 port 52437
[ ID] Interval       Transfer     Bandwidth
[  1] 0.00-19.90 sec  49.7 MBytes  20.9 Mbits/sec
[  2] local 192.168.1.11 port 50007 connected with 192.168.1.7 port 52438
[ ID] Interval       Transfer     Bandwidth
[  2] 0.00-19.90 sec  50.4 MBytes  21.3 Mbits/sec
[  3] local 192.168.1.11 port 50007 connected with 192.168.1.7 port 52439
[ ID] Interval       Transfer     Bandwidth
[  3] 0.00-19.88 sec  50.6 MBytes  21.3 Mbits/sec
```

**図 42　　CM4 コアでアプリを実行した PC で測定されたスループット**

### CM0+の実行

```
> iperf -c 192.168.1.11 -p 50007 -t 20 -i 1
> ------------------------------------------------------------
Client connecting to 192.168.1.11, TCP port 50007
TCP window size: 29.3 KByte (default)
------------------------------------------------------------
[ ID] Interval       Transfer     Bandwidth
[  0]  0.0- 1.0 sec  1.86 MBytes  15.6 Mbits/sec
[  0]  1.0- 2.0 sec  1.91 MBytes  16.1 Mbits/sec
[  0]  2.0- 3.0 sec  1.93 MBytes  16.2 Mbits/sec
[  0]  3.0- 4.0 sec  1.87 MBytes  15.6 Mbits/sec
[  0]  4.0- 5.0 sec  1.91 MBytes  16.0 Mbits/sec
[  0]  5.0- 6.0 sec  1.92 MBytes  16.1 Mbits/sec
[  0]  6.0- 7.0 sec  1.91 MBytes  16.1 Mbits/sec
[  0]  7.0- 8.0 sec  1.90 MBytes  15.9 Mbits/sec
[  0]  8.0- 9.0 sec  1.91 MBytes  16.1 Mbits/sec
[  0]  9.0-10.0 sec  1.92 MBytes  16.1 Mbits/sec
[  0] 10.0-11.0 sec  1.88 MBytes  15.7 Mbits/sec
[  0] 11.0-12.0 sec  1.90 MBytes  16.0 Mbits/sec
[  0] 12.0-13.0 sec  1.90 MBytes  16.0 Mbits/sec
[  0] 13.0-14.0 sec  1.90 MBytes  15.9 Mbits/sec
[  0] 14.0-15.0 sec  1.88 MBytes  15.8 Mbits/sec
[  0] 15.0-16.0 sec  1.91 MBytes  16.0 Mbits/sec
[  0] 16.0-17.0 sec  1.91 MBytes  16.0 Mbits/sec
[  0] 17.0-18.0 sec  1.92 MBytes  16.1 Mbits/sec
[  0] 18.0-19.0 sec  1.90 MBytes  15.9 Mbits/sec
[  0] 19.0-20.0 sec  1.81 MBytes  15.2 Mbits/sec
[  0]  0.0-20.0 sec  38.0 MBytes  15.9 Mbits/sec
```

**図 43　　CM0+コア搭載キットで測定されたスループット**

```
PS C:\DATA\Apps\iperf> .\iperf-win-isoch-fix.exe -s -w 64k -p 50007
------------------------------------------------------------
Server listening on TCP port 50007
TCP window size: 62.5 KByte
------------------------------------------------------------
[  1] local 192.168.1.11 port 50007 connected with 192.168.1.7 port 52432
[ ID] Interval       Transfer     Bandwidth
[  1] 0.00-19.91 sec  38.0 MBytes  16.0 Mbits/sec
[  2] local 192.168.1.11 port 50007 connected with 192.168.1.7 port 52433
[ ID] Interval       Transfer     Bandwidth
[  2] 0.00-19.91 sec  37.5 MBytes  15.8 Mbits/sec
[  3] local 192.168.1.11 port 50007 connected with 192.168.1.7 port 52434
[ ID] Interval       Transfer     Bandwidth
[  3] 0.00-19.91 sec  37.8 MBytes  15.9 Mbits/sec
```

**図 44　　CM0+コアでアプリを実行した PC で測定されたスループット**

上記の表と画像は、明確に CM4 コアのスループットが CM0+コアより約 5.7 Mbps 高いことを示しています。したがって、CM4 からの実行は、一般的に CM0+と比較してコア アーキテクチャの特長によって、スループットが高くなると結論づけられます。

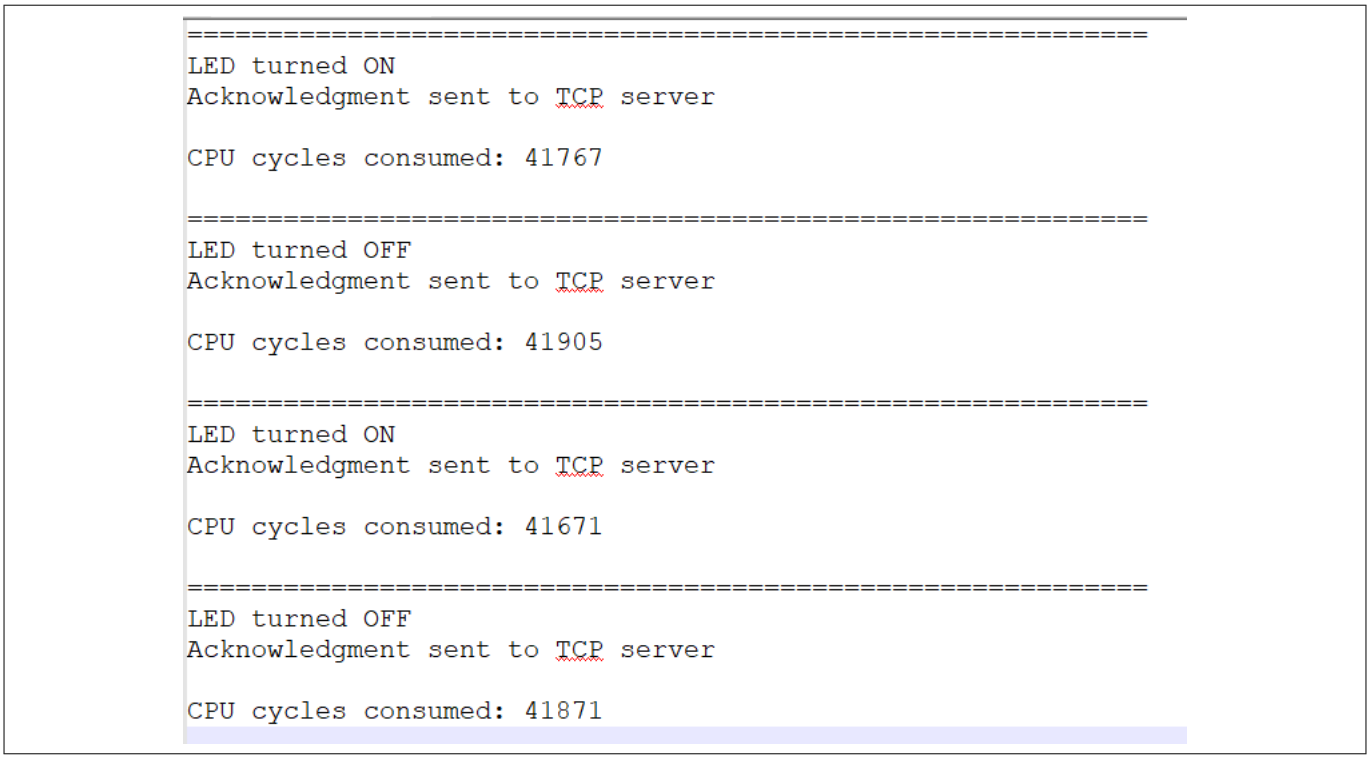


**図 45　　CM コアの Wi-Fi タスクが消費した CPU サイクル**

定期的にデータバイトを受信する TCP クライアント アプリケーションは、上の画像に示すように、CM4 コアで転送ごとに約 42k CPU サイクルを消費します。同じことは、Wi-Fi スタックとアプリケーションコードを CM0+コアに移動させることで節約でき、CM0+コアは他のプロセス負荷の高いタスクに利用できます。

### 4.8.2　　消費電力の分析

下図は、消費電力の測定に使用したテストセットアップを示します。CY8CPROTO-062S2-43439 キットは、低電力モードで mtb-example-wifi-wlan-lowpower アプリケーションを実行するように設定されており、キットのプログラミングのために ModusToolbox™が動作する PC に USB ケーブルで接続されています。キットと PC は同じ Wi-Fi ルーターに接続されています。キットの電流測定ポイントは、消費電流を測定するためにマルチメーターに接続されています。

*注:　　このボードで電流を測定するには、キットの微調整が必要です。抵抗 R65 を外し、マルチメーターのプローブを P6_VDD 信号と VTARG 信号の間に接続してください。*

## 4 PSOC™ 6 MCU デュアルコアでの開発プロセス

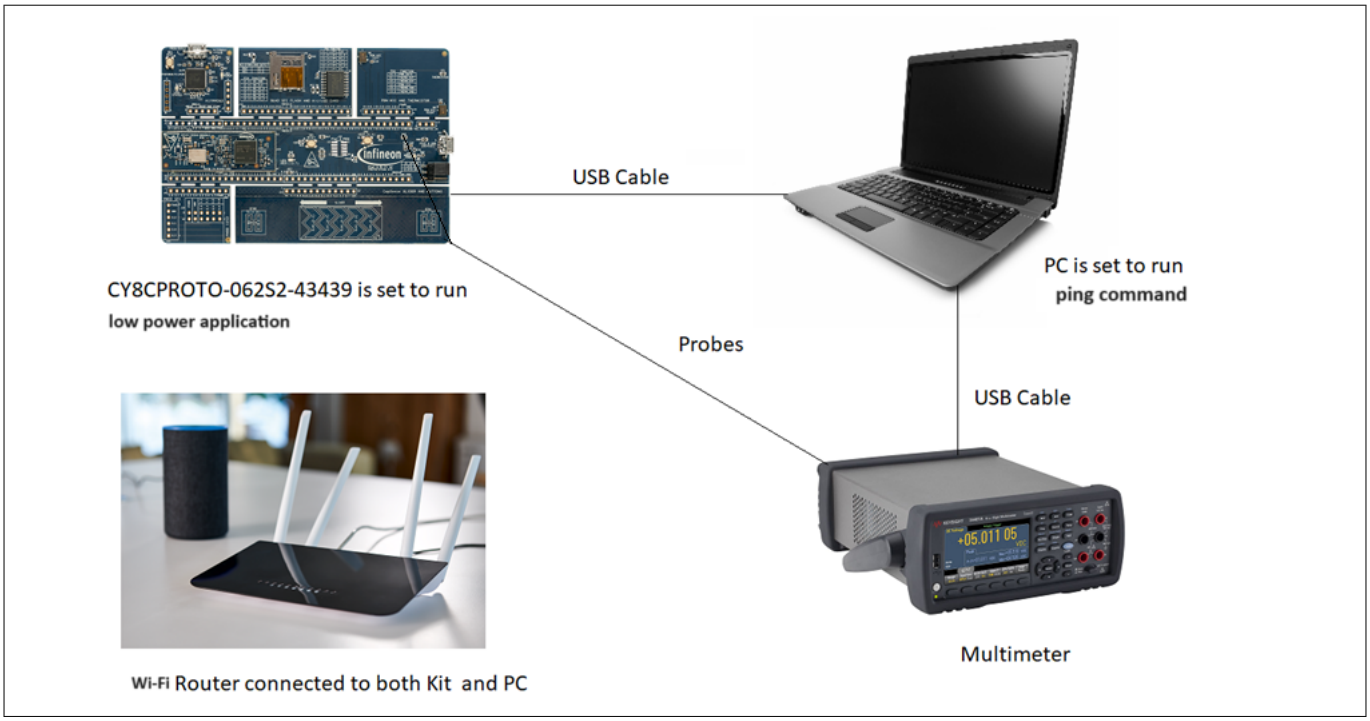


**図 46　　電流測定のセットアップ**

次の表は、デフォルトの BSP 設定で、10 個の ping コマンドを実行した場合に達成されたスループットを示します。

**表 2　　消費電流の詳細**

| シナリオ | 電流 (mA) |
|---|---|
| CM0+コア スリープ、CM4 コア アクティブ @3.3 V | ピーク = 3.42 mA、平均 = 773 µA |
| CM4 コア スリープ、CM0+コア アクティブ @3.3 V | ピーク = 2.58 mA、平均 = 751 µA |

下の図は、パワーアナライザーで得られた電流測定値を示します。

## 4 PSOC™ 6 MCU デュアルコアでの開発プロセス




**図 47　　CM4 でのアプリケーション実行**




**図 48　　CM0+でのアプリケーション実行**

表 2、図 47、および図 48 は、CM0+コアと比較して、CM4 コアでアプリケーションを実行すると、ピーク消費電流が約 0.85 mA 高くなることを示しています。この例で使用されているアプリケーションは、パケットフィルタリングなどのオフロード機能を利用することで、消費電力を最小限に抑えるように設計されていることに注意してください。その結果、コア間の平均消費電流にはわずかな差しかありません。したがって、CM4 からの実行は、一般的に CM0+と比較してコア アーキテクチャの特長によって、消費電流が高くなると結論づけられます。

### 4.8.3 コア間のアプリケーションの移植

インフィニオン開発者コミュニティの記事 KBA236435 のガイドラインに従って、開発した ModusToolbox™接続アプリケーションをあるコアから別のコアへ (例えば CM4 から CM0+へ) 簡単に移植できます。

この移植は、アプリケーションからの要求のトレードオフを評価するために必要な場合があります。

### 4.8.4 仮想コネクティビティ マネージャー

CM4 コアと CM0+コアの両方で接続アプリケーションを同時に実行する必要がある場合 (例えば、2 つの MQTT クライアント)、ModusToolbox™は、コア間で接続ライブラリを共有するための VCM ライブラリを提供します。セカンダリ コネクティビティ コアからプライマリ コネクティビティ コアへ、コネクティビティ関連の API を呼び出すために、このライブラリは IPC メカニズムを使用します。これによって、アプリケーションの設計がシンプルになります。また、両方のコアのバイナリに接続ライブラリをリンクする必要がなくなるため、メモリが節約されます。サンプルコード mtb-example-wifi-dual-core-virtual-mqtt-client は、このライブラリの機能を示します。

# 5 まとめ

本アプリケーション ノートでは、PSOC™ 6 MCU のデュアルコア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。

  ***注**: 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。*

- **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。インフィニオン 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。インフィニオン 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、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 および ModusToolbox™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。

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、すべてのインフィニオン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。

インフィニオンは、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。

# 関連ドキュメント

PSOC™ 6 MCU リソースの包括的なリストについては、インフィニオン サポートの KBA223067 を参照してください。

| | |
|---|---|
| **アプリケーションノート** | |
| AN228571 - Getting started with PSOC™ 6 MCU on ModusToolbox™ software | PSOC™ 6 MCU デバイスと、最初の ModusToolbox™プロジェクトを構築する方法について説明 |
| AN221774 – Getting started with PSOC™ 6 MCU on PSOC™ Creator | 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 – PSOC™ Creator での PSOC™ 6 Bluetooth® LE 入門 | Bluetooth® LE 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 割込み | 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 – PSOC™ 6 MCU - 生成コードの IDE へのインポート (PSOC™ 6 MCU プロジェクト用) | 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-CPU 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** | |
| 割込み | 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 データシート | PSOC™ 6 テクニカル リファレンス マニュアル |
| **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 Wi-Fi Bluetooth® パイオニアキット |
| CY8CKIT-062S2-43012 | PSOC™ 62S2 Wi-Fi Bluetooth® パイオニアキット |

## 関連ドキュメント

| | |
|---|---|
| CY8CPROTO-063-BLE | PSOC™ 6 Bluetooth® LE プロトタイプキット |
| CY8CPROTO-062-4343W | PSOC™ 6 Wi-Fi Bluetooth®プロトタイプキット |
| CY8CPROTO-062S3-4343W | PSOC™ 62S3 Wi-Fi Bluetooth®プロトタイプキット |
| CYW9P62S1-43438EVB-01 | PSOC™ 62S1 Wi-Fi Bluetooth®パイオニアキット |
| CYW9P62S1-43012EVB-01 | PSOC™ 62S1 Wi-Fi Bluetooth®パイオニアキット |
| CY8CPROTO-062S2-43439 | PSOC™ 62S2 Wi-Fi Bluetooth®プロトタイプキット |
| **ツール資料** | |
| ModusToolbox™ソフトウェア | 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。 |
| ペリフェラル ドライバ ライブラリ (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。mtb-pdl-cat1 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。 |
| ハードウェアアブストラクションレイヤ (HAL) | GitHub からのみ利用できます。mtb-hal-cat1 を参照してください。 |

# 改訂履歴

| 版数 | 発行日 | 変更内容 |
|---|---|---|
| 英語版(**) | - | この版は英語版のみです。英語版の改訂内容: New Application Note |
| 英語版(*A) | - | この版は英語版のみです。英語版の改訂内容: Updated Template |
| 英語版(*B) | - | この版は英語版のみです。英語版の改訂内容: Updated text and screen shots for release versions of PSOC™ Creator 4.1 and PDL 3.0.0<br>Other miscellaneous edits |
| 英語版(*C) | - | この版は英語版のみです。英語版の改訂内容: Minor edits<br>Ported to new application note document template<br>Confidential tag removed |
| ** | 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。英語版の改訂内容: Added a new Figure 2<br>Updated Figure 4 and associated kit device part number<br>Updated Figures 6 and 9 for PSOC™ Creator 4.2 beta 2<br>Emphasized using CM0+ as a support CPU for tasks such as Bluetooth® LE and CAPSENSE™<br>Added references to AN221111, Creating a Secure System; AN217666, PSOC™ 6 Interrupts; AN219528, PSOC™ 6 Low Power Modes; and CE216795, PSOC™ 6 Dual-CPU<br>Updated power modes description<br>Miscellaneous minor edits<br>Ported to new application note template<br>Changed the document title to PSOC™ 6 MCU Dual-CPU System Design |
| 英語版(*E) | - | この版は英語版のみです。英語版の改訂内容: Expanded section 4.3 to include dual-CPU debugging with µVision and IAR Embedded Workbench IDEs<br>Updated power mode descriptions in section 3<br>Miscellaneous minor edits<br>Ported to *Y application note template |
| *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。英語版の改訂内容: Added support for ModusToolbox™ |
| 英語版(*G) | - | この版は英語版のみです。英語版の改訂内容: Updated instructions based on ModusToolbox™ 2.1<br>Added HAL support |
| *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。英語版の改訂内容: Updated instructions based on ModusToolbox™ 2.2 |
| 英語版(*I) | 2022-09-19 | この版は英語版のみです。英語版の改訂内容: Migrated to Infineon template<br>Updated instruction based on ModusToolbox™ 3.0 |
| 英語版(*J) | 2023-08-03 | この版は英語版のみです。英語版の改訂内容: Updated link references |
| 英語版(*K) | 2023-12-01 | この版は英語版のみです。英語版の改訂内容: Added the 接続アプリケーション設計の検討事項 section and its subsection<br>Removed the Techniques to improve throughput section |

## 改訂履歴

| 版数 | 発行日 | 変更内容 |
|---|---|---|
| *C | 2024-10-23 | これは英語版 002-15656 Rev. *L を翻訳した日本語版 Rev. *C です。英語版の改訂内容: Replaced CY8CPROTO-062-4343W with CY8CPROTO-062S2-43439 and added reference to CY8CPROTO-062S2-43439 in the Development kit documentation table. |

# 商標

The Bluetooth® word mark and logos are registered trademarks owned by Bluetooth SIG, Inc., and any use of such marks by Infineon is under license.

**Trademarks**

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

**Edition 2024-10-23**

**Published by**

**Infineon Technologies AG**

**81726 Munich, Germany**

**© 2024 Infineon Technologies AG**

**All Rights Reserved.**

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

**Email: erratum@infineon.com**

**Document reference**
**IFX-jii1649671899455**

**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。