ModusToolbox™ ソフトウェアを使用する PSOC™ 6 MCU 入門

本書について

適用範囲と目的

このアプリケーション ノートでは、デュアル CPU Arm® Cortex-M4®および Cortex-M0+®プロセッサを搭載した低消費電力で安全な MCU である PSOC™ 6 マイクロコントローラ (MCU) を紹介します。このアプリケーション ノートでは、PSOC™ 6 MCU アーキテクチャと開発ツールについて、ModusToolbox™ ソフトウェアを使用して最初のアプリケーションを作成する方法を説明します。また、このアプリケーションノートでは、PSOC™ 6 MCU についての理解を深めるために、オンラインで利用可能なリソースについても説明します。

対象者

このドキュメントは、PSOC™ 6 MCU および ModusToolbox™ソフトウェアを初めて使用するユーザーを対象とします。

関連製品ファミリー

すべて PSOC™ 6 MCU デバイス

ソフトウェア バージョン

ModusToolbox™ 3.2 以降

さらにサンプル コードをお求めでしょうか？以下のとおり対応いたします。

ModusToolbox™を使用して増え続ける PSOC™ PSOC™ 6 サンプル コードの一覧にアクセスするには、GitHub サイトにアクセスしてください。

目次

目次

1 はじめに

PSOC™ 6 MCU は、デュアルコアアーキテクチャとバッテリ駆動アプリケーション向けの低消費電力設計技術を備えた超低消費電力 PSOC™ デバイスです。デュアルコアの Arm® Cortex-M4®および Cortex-M0+®アーキテクチャにより、設計者は消費電力と性能を同時に最適化できます。オンチップフラッシュ/SRAM メモリ、暗号化された外部フラッシュメモリ拡張、設定可能なメモリおよびペリフェラル保護ユニット、暗号アクセラレータを内蔵した PSOC™ 6 MCU は、セキュアブート、セキュアプロビジョニング、セキュアキーストレージ、ランタイムセキュリティ、セキュアファームウェアアップデートなどの標準的な組み込みセキュリティ機能をサポートしています。

設計者は、MCU の豊富なアナログおよびデジタル ペリフェラルを使用して、MEMS センサーや電子ペーパー ディスプレイなどの革新的なシステム コンポーネント用のカスタム アナログ フロントエンド (AFE) やデジタル インターフェースを作成できます。PSOC™ 6 MCU は、業界をリードする第 4 世代の CAPSENSE™ 静電容量式センシング テクノロジを搭載し、堅牢で信頼性の高い最新のタッチおよびジェスチャーベースのインターフェイスを実現します。PSOC™ 6 MCU とインフィニオンの AIROC™ Wi-Fi、AIROC™ Bluetooth®、または AIROC™ コンボ無線モジュールの組み合わせは、セキュアで低消費電力、機能豊富な IoT 製品に最適なソリューションです。

PSOC™ 6 MCU の多彩な機能により、民生用電子機器、産業用アプリケーション、スマートホーム、IoT、汎用組み込みアプリケーションなど、幅広いアプリケーションで使用できます。

1.1 PSOC™ 6 のアーキテクチャとポートフォリオの概要

下図に示す PSOC™ 6 アーキテクチャのブロック図は、PSOC™ 6 MCU ポートフォリオ全体をカバーする統一ブロック図です。PSOC™ 6 ポートフォリオの特定のデバイスで提供される機能については、各製品ファミリーのデータシートを参照してください。たとえば、フラッシュメモリ/SRAM メモリ、IO 数、ペリフェラル機能/数、およびサポートされるパッケージは、PSOC™ 6 ポートフォリオで選択された製品ファミリーによって異なります。

1 はじめに

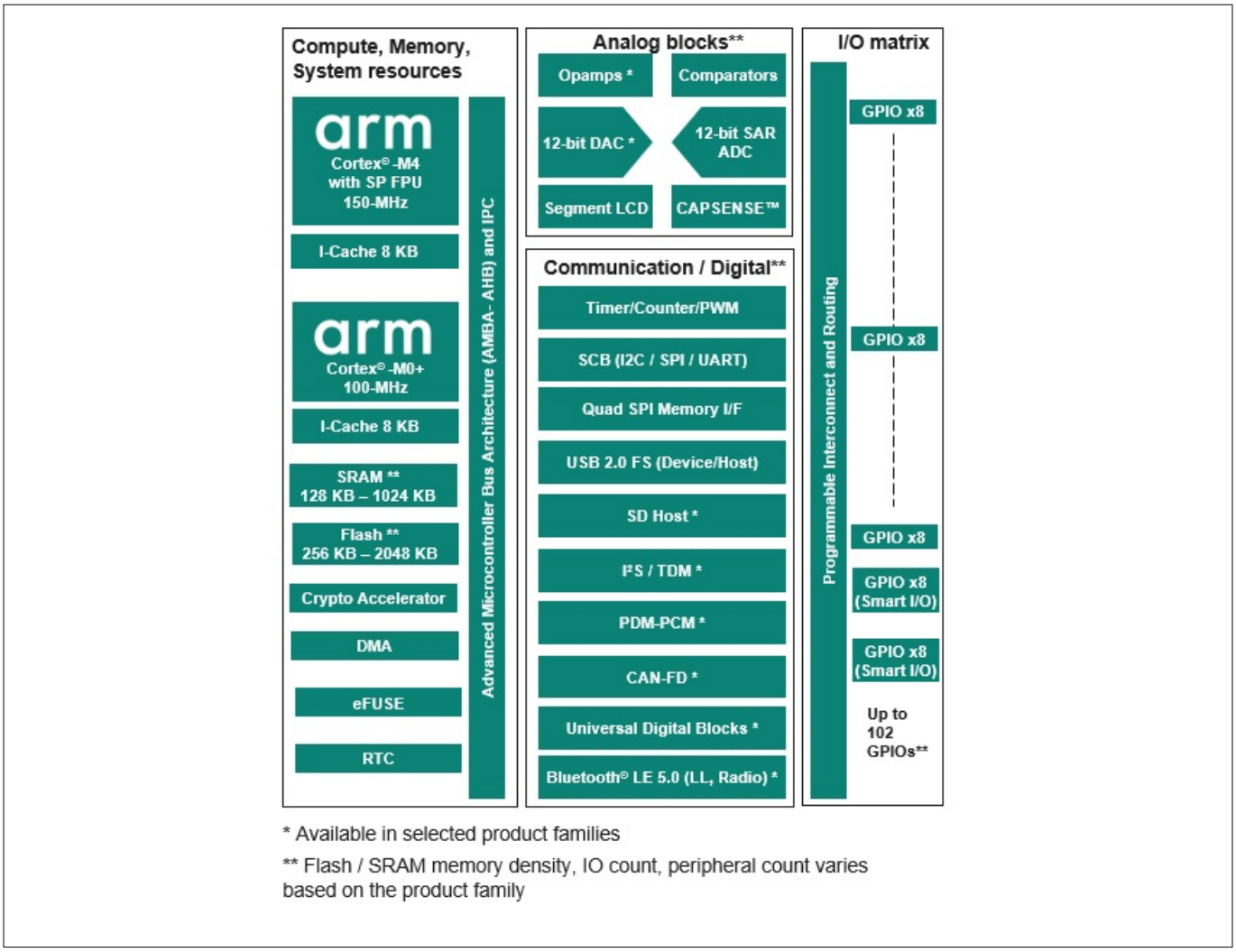

図 1　　PSOC™ 6 アーキテクチャ

PSOC™ 6 MCU ポートフォリオ セグメンテーション

PSOC™ 6 MCU ポートフォリオは、4 つの異なる製品ラインに分類される複数の製品ファミリーで構成されます。

1. **PSOC™ 61 (プログラマブル ライン)**は、Arm® Cortex® M4 のみがユーザー アプリケーションで使用可能な製品ファミリーで構成されます。PSOC™ 61 ラインの注文部品番号 (OPN) は、「CY8C61」から始まります。
2. **PSOC™ 62 (パフォーマンスライン)**は、ユーザーアプリケーション向けに Arm® Cortex® M4 と Arm® Cortex-M0+の®両方が利用可能な製品ファミリーで構成されています (デュアル CPU アーキテクチャ)。PSOC™ 62 ラインの注文部品番号 (OPN) は、「CY8C62」から始まります。
3. **PSOC™ 63 (コネクティビティライン)**は、Bluetooth® LE 無線 PHY と MAC を MCU の一部として統合した製品ファミリーで構成されます。PSOC™ 63 ラインの注文部品番号 (OPN) は、「CY8C63」から始まります。
4. **PSOC™ 64 (セキュアライン)**は、Arm® Cortex® M0+がセキュアな処理環境 (SPE) として構成され、Arm® Cortex® M4 が非セキュアな処理環境 (NSPE) として動作する製品ファミリーで構成されます。PSOC™ 64 ラインの注文部品番号 (OPN) は、「CYB064」または「CYS064」から始まります。この製品ファミリーは、主に、セキュリティ キーと証明書を含む、ファームウェアの安全な工場でのプロビジョニングを必要とするアプリケーションを対象とします。

1.2 デバイスの特長

PSOC™ 6 MCU は、図 1 に示すように、広範な機能を備えています。以下は主な機能の一覧です。詳細はデバイスのデータシート、テクニカル リファレンス マニュアル (TRM) および「関連資料」を参照してください。

高性能、低消費電力の計算システム

- デュアルコアアーキテクチャ: 150MHz Arm® Cortex®-M4 および 100MHz Arm® Cortex®-M0+
- オンチップ メモリ: 最大 1024KB SRAM、最大 2048KB フラッシュ
- 超低消費電力 (0.9V) および低消費電力 (1.1V) 動作モード
- 複数のデバイスの低消費電力モード: ハイバネート、ディープスリープ、スリープ、アクティブ。低電力アナログ動作

堅牢なセキュリティ機能

- 高度な暗号アクセラレータと真性乱数生成器
- セキュア キー ストレージ用のワンタイムプログラマブル eFUSE5
- セキュアブート、セキュアプロビジョニング、イメージ認証
- ファームウェア アップデートのための Read-While-Write フラッシュ技術による安全な無線 (OTA) ファームウェア アップデート

統合

- 堅牢なタッチ ユーザー インターフェースを単一の MCU に統合する第 4 世代 CAPSENSE™
- セグメント LCD ドライブ、シリアルインターフェースディスプレイドライバ
- メモリ拡張用クアッド SPI メモリ I/F
- 外部デジタル グルーロジックを MCU に統合するスマート I/O
- MCU に CPLD とミニ FPGA ロジックを実装するためのユニバーサル デジタル ブロック (UDB)

豊富なアナログ ペリフェラル

- 12 ビット SAR ADC、12 ビット DAC、オペアンプ、低消費電力コンパレータ。低電力アナログ動作

デジタル ブロックと通信インターフェース

- 高度な設定が可能な 16 ビットおよび 32 ビットのタイマ、カウンタおよび PWM
- デジタルセンサー/ホスト MCU インターフェース用シリアル通信ブロック (I2C/SPI/UART)
- オーディオ アプリケーション用 I2S/TDM、PDM-PCM コンバータ
- SD ホストコントローラ (SDHC)、USB 2.0 フルスピード (ホストおよびデバイス)
- 産業アプリケーション用 CAN-FD

豊富な I/O とパッケージオプション

- 124-BGA, 100-WLCSP, 80-WLCSP, 80-M-CSP, 49-WLCSP, 68-QFN, 128-TQFP, 100-TQFP, 80-TQFP, 64-TQFP
- 最大 102 の GPIO
- -40～85℃動作、一部のデバイスでは 105℃の拡張温度動作に対応

1.3 対象アプリケーション

PSOC™ 6 MCU は、汎用性、安全性、低消費電力、豊富な機能を備えており、さまざまなエンドアプリケーションに最適なマイクロコントローラです。アプリケーションの一部を以下に示します。

- スマートウォッチやフィットネストラッカーなどのウェアラブル機器
- スマートロック、サーモスタット、統合アラームシステムなどのスマートホームデバイス
- 洗濯機、冷蔵庫、調理台などのスマート家電
- システム管理機能のためのデータセンターおよびコンピューティング アプリケーション
- 電動工具、e-bike、その他のモーター制御アプリケーション

1 はじめに

- 産業用 IoT アプリケーション
- バッテリー駆動デバイス

以下の図に、PSOC™ 6 MCU を使用した実際の使用例のアプリケーションレベル ブロックダイヤグラムを示します。

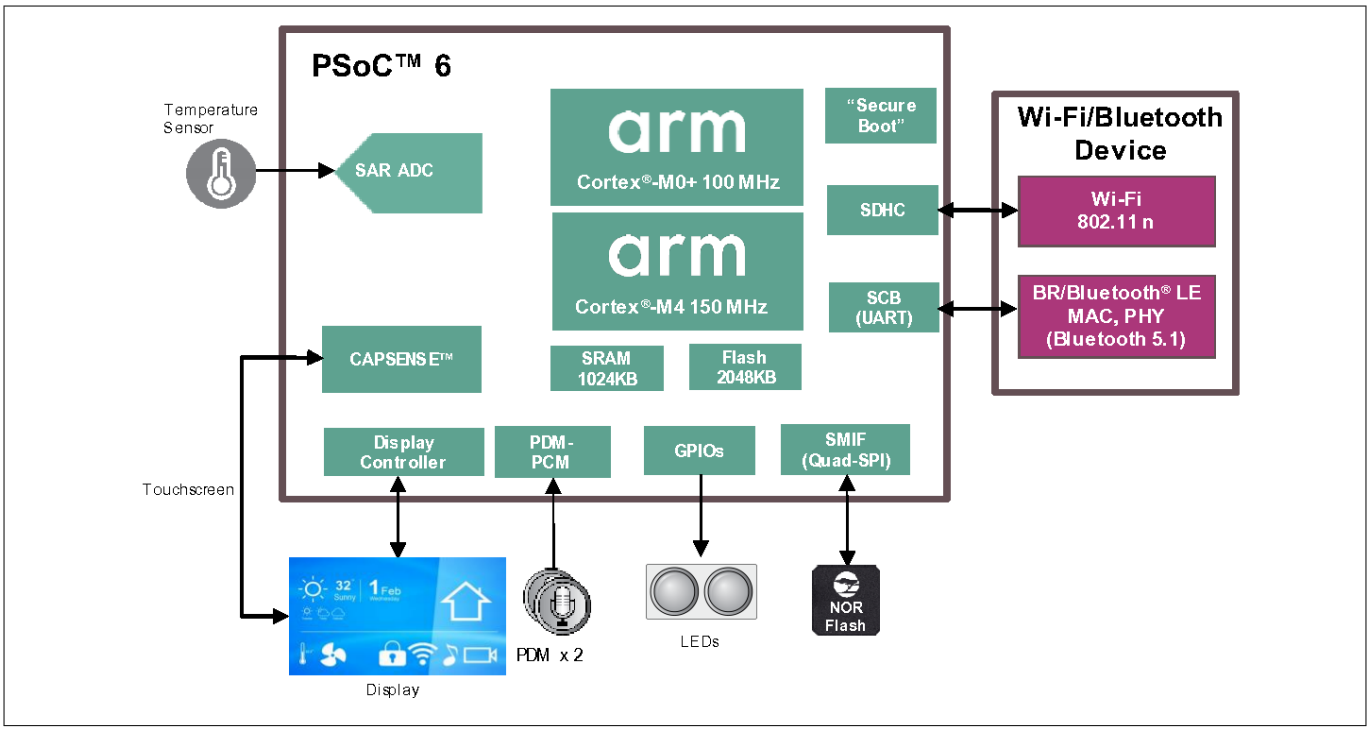

図 2　　PSOC™ 6 MCU を使用するアプリケーションレベル ブロックダイヤグラム

最終的なアプリケーションのユースケースに応じて、PSOC™ 6 MCU は以下に示すような多種多様なシステム機能を実行できます。

- メイン システム アプリケーション MCU
- アナログ フロントエンド (AFE)、CAPSENSE™ (タッチ)、ディスプレイ、オーディオ機能を統合した Wi-Fi MCU
- Arm® A クラス ベースの SoC および MPU 向け低消費電力センサー コプロセッサ
- CAPSENSE™ (タッチ) およびその他のユーザー インターフェース機能を統合したモーター制御 MCU
- ファン制御、電源シーケンス、サーバー バックプレーン管理などのシステム管理コントローラー

2 PSOC™ 6 リソース

PSOC™ 6 MCU4 でアプリケーションを開発するための豊富な技術リソースがあります。これらのリソースを以下に記載します。

- PSOC™ 6 ビデオライブラリ
- PSOC™ 6 MCU ウェブページ
- 製品セレクタ: PSOC™ 6 MCU
- データシート:各デバイス ファミリーの電気的仕様を説明します。
- アプリケーション ノートおよびサンプル コードは、基本的なレベルから上級レベルまで、幅広いトピックを提供します。サンプル コードのコレクションも参照できます。
- テクニカル リファレンス マニュアル (TRM): 各デバイス ファミリーのアーキテクチャとレジスタの詳細な説明をします。
- PSOC™ 6 MCU プログラミング仕様書: PSOC™ 6 MCU デバイスの不揮発性メモリをプログラムするために必要な情報を提供します。
- CAPSENSE™デザイン ガイド: PSOC™デバイスを使用して静電容量タッチ センシング アプリケーションを設計する方法について説明します。
- 開発ツール: PSOC™ 6 MCU を使用したさまざまなアプリケーションの評価、設計、および開発のために、多くの低コストのキットとシールドボードを提供します。
- トレーニング ビデオ: PSOC™ 6 MCU の専用シリーズを含む製品とツールに関するビデオトレーニングを提供します。
- 技術サポート: PSOC™ 6 community forum、Knowledge base articles.

3 PSOC™ 6 MCU 開発キット

インフィニオンは、PSOC™ 6 ベースのアプリケーションを簡単かつ迅速に評価およびプロトタイピングできるよう、さまざまなフォームファクタのハードウェア開発キットを幅広く提供しています。プロトタイピング キットは、ハードウェア シールドを追加接続する必要のないプロトタイピング アプリケーション向けの、低価格で小型の評価キットです。Pioneer/Evaluation キットは、Arduino シールドやワイヤレス無線インターフェース用の M.2 インターフェースなど、さまざまなハードウェア拡張機能を提供する、よりフル機能のキットです。

次の表は、さまざまな製品ファミリーに対応する PSOC™ 6 キットの一覧です。PSOC™ 6 MCU ポートフォリオの一般的な評価には、PSOC™ 6 プロトタイピングキット (CY8CPROTO-062S2-43439) または PSOC™ 6 パイオニアキット (CY8CKIT-062S2-43012) を使用することを推奨します。ModusToolbox™ ソフトウェアは、開発キットを使用して組込みアプリケーションを作成するためのソフトウェア開発プラットフォームです。

表 1 PSOC™ 6 キット

Kit MPN	適用可能な PSOC™ 6 製品ファミリー	Kit type	ボードサポートパッケージ GitHub リポジトリ
CY8CPROTO-062S2-43439 **(Default recommended Proto kit)**	CY8C61x8, CY8C61xA, CY8C62x8, CY8C62xA	prototyping	BSP
CY8CKIT-062S2-43012 **(Default recommended Pioneer kit)**	CY8C61x8, CY8C61xA, CY8C62x8, CY8C62xA	Pioneer	BSP
CY8CEVAL-062S2	CY8C61x8, CY8C61xA, CY8C62x8, CY8C62xA	Evaluation	BSP
CY8CKIT-062S4	CY8C61x4, CY8C62x4	Pioneer	BSP
CY8CPROTO-062S3-4343W	CY8C61x5, CY8C62x5	prototyping	BSP
CY8CKIT-062-WIFI-BT	CY8C61x6, CY8C61x7, CY8C62x6, CY8C62x7	Pioneer	BSP
CY8CPROTO-064B0S3	CYB064x5	prototyping	BSP
CY8CPROTO-064S1-SB	CYB064x7	prototyping	BSP
CY8CKIT-064B0S2-4343W	CYB064xA	Pioneer	BSP

シールドモジュールとともに PSOC™ 6 MCU のキットの完全なリストについては、Microcontroller (MCU) kits のページを参照してください。

4 PSOC™ 6 ソフトウェア エコシステムとファームウェア/アプリケーション開発

インフィニオンは、PSOC™ 6 MCU をベースとしたファームウェア/アプリケーション開発用の ModusToolbox™ ソフトウェアを提供しています。ModusToolbox™ ソフトウェアは、以下を含む幅広いインフィニオン マイクロコントローラ デバイスをサポートする新しく、拡張性の高い開発エコシステムです: PSOC™ Arm® Cortex® マイクロコントローラー、XMC™ 産業用マイクロコントローラー、AIROC™ Wi-Fi デバイス、AIROC™ Bluetooth®デバイス、および USB-C パワーデリバリー (給電) マイクロコントローラーこのソフトウェアには、コンフィギュレーション ツール、低レベル ドライバ、ミドルウェアライブラリと、MCU およびワイヤレス 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。すべてのツールは Windows、macOS、Linux 上で動作します。また ModusToolbox™には、すべての ModusToolbox™ツールと統合されたフローを提供する Eclipse IDE も含まれます。Visual Studio Code、IAR Embedded Workbench、Arm® MDK (µVision) などの他の IDE もサポートしています。

ModusToolbox™ ソフトウェアは、スタンドアロンのデバイスとミドルウェアのコンフィギュレーターをサポートしています。コンフィギュレーターを使用して、デバイスのさまざまなブロックのコンフィギュレーションを設定し、ファームウェア開発で使用できるコードを生成します。

GitHub サイトで、ライブラリと利用可能なソフトウェアを提供します。

ModusToolbox™のツールとリソースは、コマンド ラインでも使用できます。詳細なドキュメントについては、ModusToolbox™ tools package user guide の build system の章を参照してください。

4.1 ModusToolbox™ ツールパッケージのインストール

詳細は ModusToolbox™ tools package installation guide を 参照してください。

4.2 IDE の選択

最新世代のツールセットである ModusToolbox™ソフトウェアは、Windows、Linux、および macOS プラットフォームに対応します。ModusToolbox™は、Eclipse IDE、Visual Studio Code、Arm® MDK (µVision)、および IAR Embedded Workbench などのサードパーティ IDE にも対応します。ツール パッケージには、サポートするすべての IDE 用の実装が含まれています。ツールは、すべての PSOC™ 6 MCU をサポートしています。また、関連する BSP とライブラリ コンフィギュレーターは、3 つのホスト OS すべてで動作します。

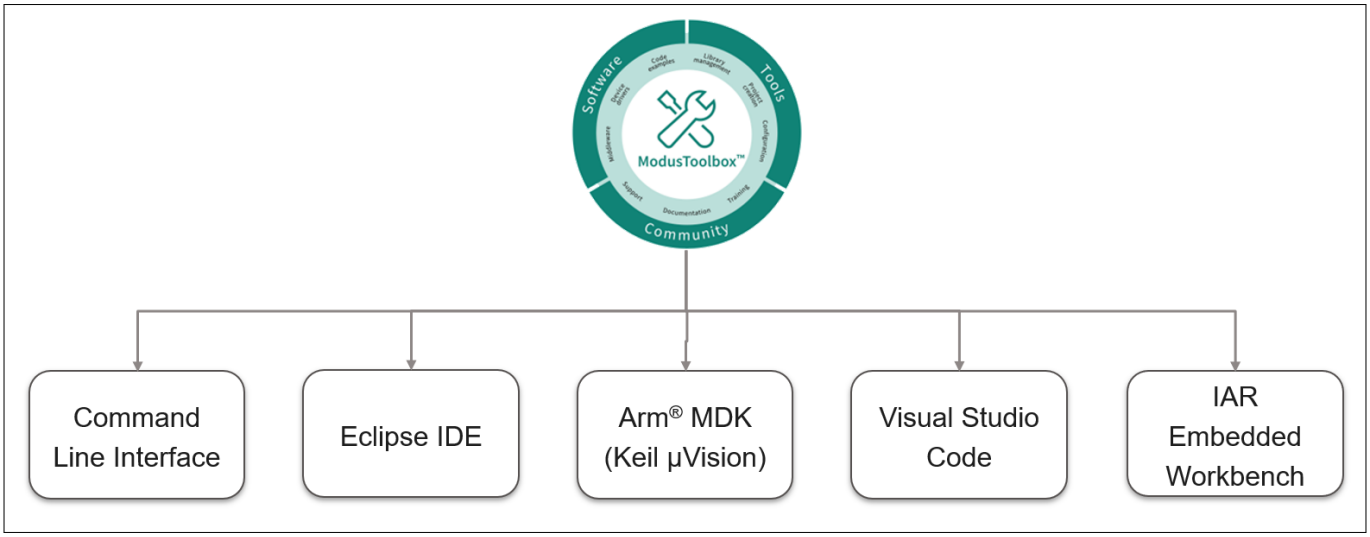

図 3 ModusToolbox™ 環境

5 PSOC™ 6 MCU 設計のスタートガイド

ここでは、以下を説明します。

- シンプルな PSOC™ 6 MCU ベースの設計をビルドし、開発キットにプログラムする方法を説明します。
- PSOC™ 6 MCU デザインテクニックや ModusToolbox™ ソフトウェアをさまざまな IDE で使用する方法を簡単に学べます。

5.1 前提条件

開始する前に、PSOC™ 6 MCU 製品ラインに適切な開発キットがあり、必要なソフトウェアがインストールされていることを確認してください。プロジェクトの作成中に、GitHub リポジトリへのインターネットアクセスも必要です。

5.1.1 ハードウェア

以下に示すように、この例では PSOC™ 62S2 Wi-Fi Bluetooth® prototyping kit (CY8CPROTO-062S2-43439) 用に開発されています。ただし、他の開発キットのアプリケーションをビルドできます。PSOC™ 6 MCU を始めるために使用できるキットのリストについては、PSOC™ 6 MCU 開発キット のセクションを参照してください。

5.1.2 ソフトウェア

ModusToolbox™ 3.2 以降

ソフトウェアをインストールした後、ModusToolbox™ tools package user guide を参照し、本ソフトウェアの概要を確認してください。

5.2 これらの手順の使用

これらの説明はいくつかのセクションにグループ化されています。各セクションは、アプリケーション開発作業の流れを段階ごとに割り当てられています。主な内容は以下のとおりです。

1. 新しいアプリケーションの作成
2. 設計コンフィギュレーションの閲覧と更新
3. ファームウェアの記述
4. アプリケーションのビルド
5. デバイスのプログラミング
6. 設計のテスト

この設計は PSOC™ 62S2 Wi-Fi Bluetooth® prototyping kit (CY8CPROTO-062S2-43439) 用に開発されています。アプリケーションの作成に適切なキットを選択することにより、他のサポートされているキットを使用しても、このサンプルをテストできます。

5.3 設計について

この設計は、PSOC™ 6 MCU の CM4 コアを使用して、UART 通信と LED 制御の 2 つのタスクを実行します。

デバイスのリセット時、インフィニオン提供のビルド済み CM0+アプリケーション イメージが CM4 コアを有効にし、CM0+コアがスリープ状態になるように構成します。CM4 コアは、UART を使用して「Hello World」メッセージをシリアルポートストリームに出力し、キット上のユーザー LED を点滅させます。ユーザーがシリアル コンソールの Enter キーを押すと、点滅が一時停止または再開されます。

5.4 新しいアプリケーションの作成

ここでは新しいアプリケーションに関するプロセスについて順に説明します。**Empty App** スターター アプリケーションを使用し、**Hello World** スターター アプリケーションの機能を手動で追加します。

IDE の選択セクションで述べたように、ModusToolbox™ ソフトウェアは以下のサードパーティ IDE をサポートしています。

1. Eclipse IDE
2. Visual Studio Code (VS Code)
3. IAR Embedded Workbench
4. Keil µvision

以下のセクションでは、さまざまな IDE で新しいアプリケーションを作成する方法について詳しく説明します。

5.4.1 Eclipse IDE for ModusToolbox™

ModusToolbox™を使用したプロジェクト開発に精通している場合は、「**Hello World**」スターター アプリケーションを直接使用できます。すべてのファームウェアが対応しているキット用に書かれた完全な設計です。サンプル コードで、この説明がどのように実装されているかを確認できます。

ゼロから始め、本アプリケーション ノートに記載されているすべての説明に従う場合は、**Hello World** のサンプル コードを参考にしてください。

Dashboard 3.2 アプリケーションを起動して開始してください。**Dashboard 3.2** アプリケーションがスターター アプリケーションをコンピュータに正常に複製するためにインターネットにアクセスする必要があることに注意してください。

Dashboard 3.2 アプリケーションは、ドキュメントやトレーニング資料への簡単なアクセス、アプリケーションの作成や BSP の作成 編集のためのシンプルなパスにより、様々なツールの使用を開始するのに役立ちます。

1. **Dashboard 3.2** アプリケーションを開いてください。

   Dashboard 3.2 アプリケーションを開くには、`[ModusToolbox installation path]/ModusToolbox folder/dashboard 3.2.0` をクリックしてください。
2. **Dashboard 3.2** ウィンドウの右側で、**Target IDE** ドロップダウン リストから **Eclipse IDE for ModusToolbox™**を選択し、**Launch Eclipse IDE for ModusToolbox™**をクリックしてください。

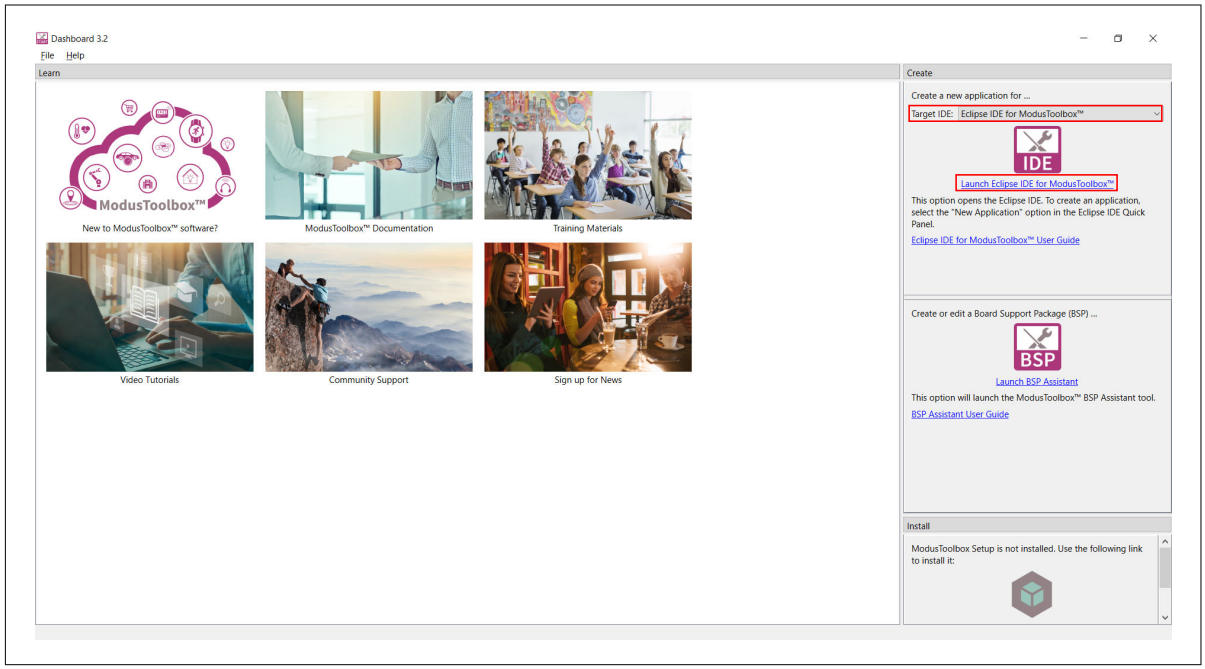

図 4 ダッシュボード 3.2 アプリケーション

3. **新しいワークスペースを選択してください。**

   ModusToolbox™の Eclipse IDE の起動時、ワークスペース ディレクトリとして使用するディレクトリを選択するダイアログが表示されます。ワークスペース ディレクトリは、ワークスペースの設定と開発結果の資産を格納するために使用されます。以下の図に示す、ように、**Browse** ボタンをクリックして、既存の空のディ

レクトリを選択できます。または、ワークスペース ディレクトリとして使用するディレクトリ名および完全パスを入力すると、IDE がそのディレクトリを作成します。

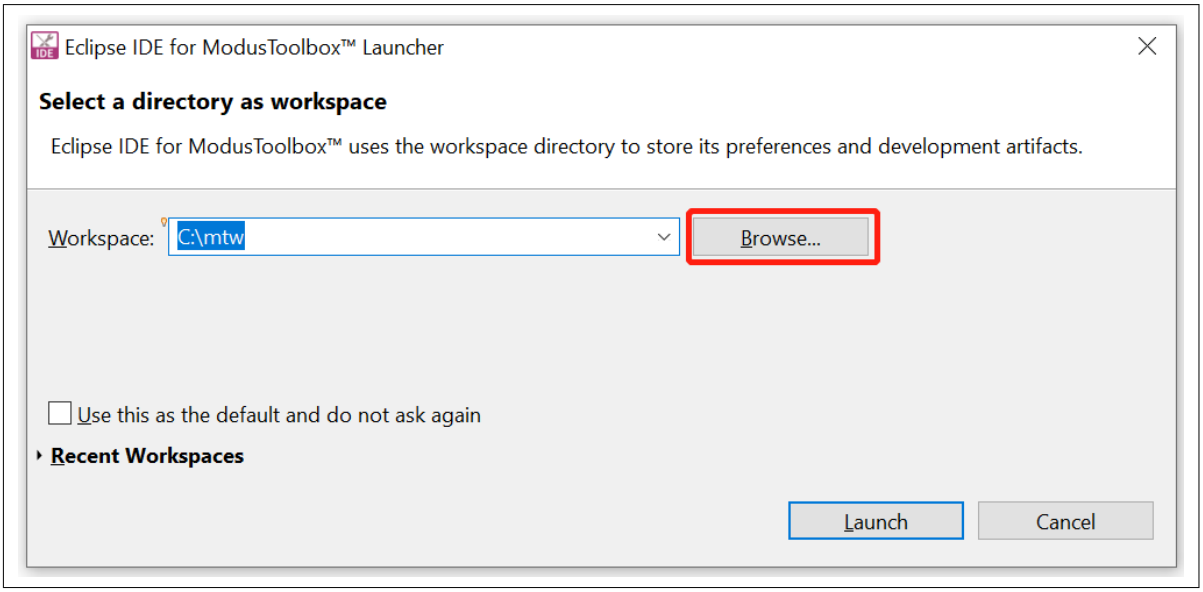

図 5　　ワークスペースとしてのディレクトリの選択

4. **ModusToolbox™ IDE で新しいアプリケーションを作成してください。**

   a. Quick Panel の Start グループで **New Application** をクリックします。

   b. または、下図のように **File** > **New** > **ModusToolbox™ Application** を選択することもできます。Project Creator が開きます。

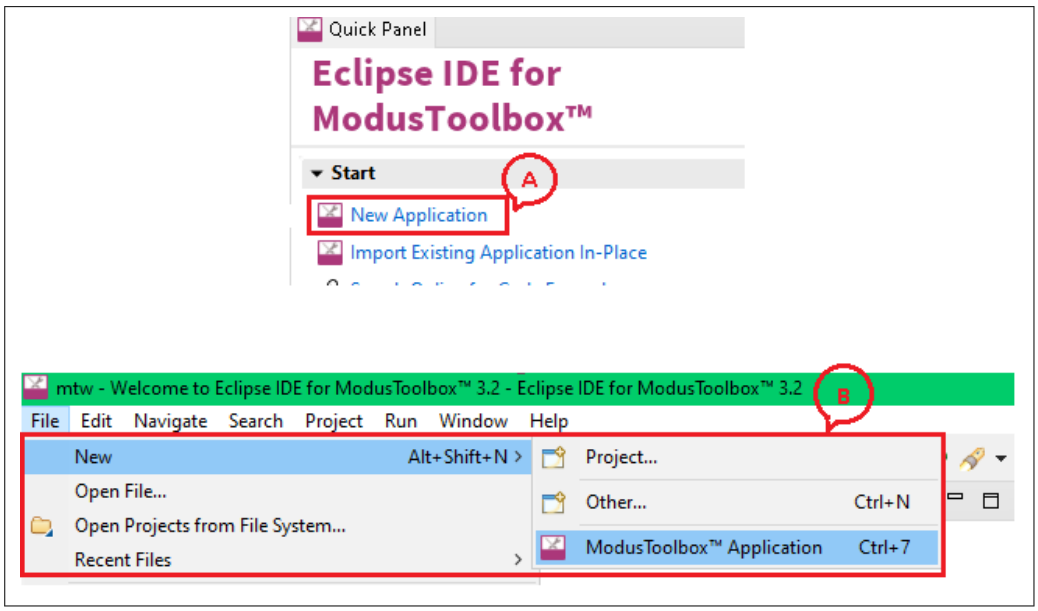

図 6　　ModusToolbox™ IDE で新しいアプリケーションの作成

5. **ターゲット PSOC™ 6 MCU 開発キットを選択してください。**

   ModusToolbox™は、新しいアプリケーション ダイアログで指定された開発キットのさまざまなワークスペース/プロジェクト オプションを設定する BSP を提供することにより、開発プロセスをスピードアップします。

   a. **Choose Board Support Package (BSP)** ダイアログで、使用する **Kit Name** を選択してください。以下の手順では **CY8CPROTO-062S2-43439** の使用を想定しています。この手順については図 7 を参照してください。

   b. **Next** をクリックしてください。

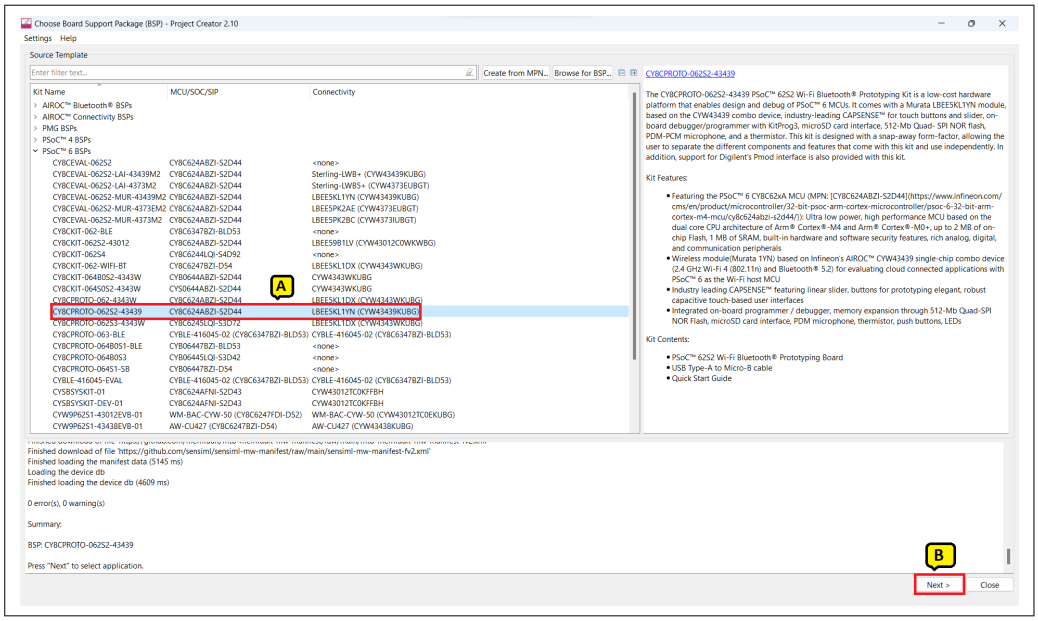

図 7　　ターゲット ハードウェアの選択

c. 下図のように、**Select Application** ダイアログで、**Empty App** starter アプリケーションを選択してください。

d. **Name** フィールドには、**Hello_World** などのアプリケーションの名前を入力してください。必要に応じて、デフォルト名のままにすることもできます。

e. 以下の図に示すように **Create** をクリックしてアプリケーションを作成してください。プロジェクトが正常に作成されたら、Project Creator が自動的に閉じるのを待ってください。

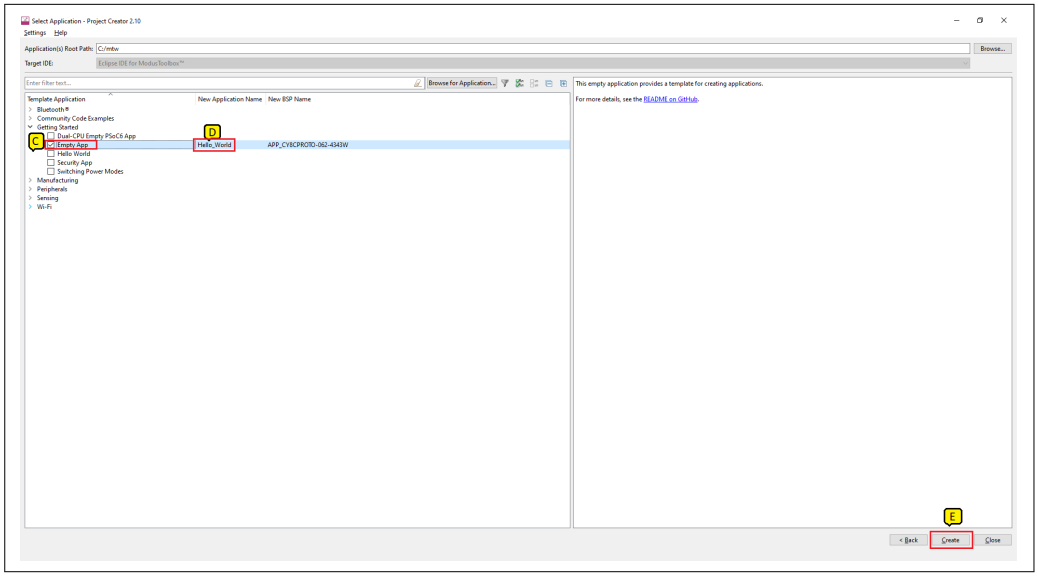

図 8　　新しいアプリケーションの作成

PSOC™ 6 MCU の新しい ModusToolbox™アプリケーションが正常に作成されました。

BSP は、CYW43439KUBG Wi-Fi/Bluetooth®無線とともに、PSOC™ 62S2 Wi-Fi-Bluetooth® prototyping kit (CY8CPROTO-062S2-43439) along with the CYW43439KUBG Wi-Fi/Bluetooth® radio に搭載されたデフォルト デバイスとして CY8C624ABZI-D54 を使用します。

PSOC™ 6 MCU または異なる PSOC™ 6 MCU 部品番号に基づくカスタム ハードウェアを使用する場合は、Custom BSP App Note または BSP Assistant user guide を参照してください。

詳細については Eclipse IDE for ModusToolbox™ user guide を参照してください。

5.4.1.1 設計コンフィギュレーションの閲覧と更新

図 9 に、アプリケーション プロジェクトの構造を示す Eclipse IDE プロジェクト エクスプローラー インターフェースを示します。

PSOC™ 6 MCU アプリケーションは、CM4 コアのコードを開発するプロジェクトで構成されますプロジェクト フォルダは、さまざまなサブフォルダで構成され、それぞれのサブフォルダがプロジェクトの特定の側面を示します。

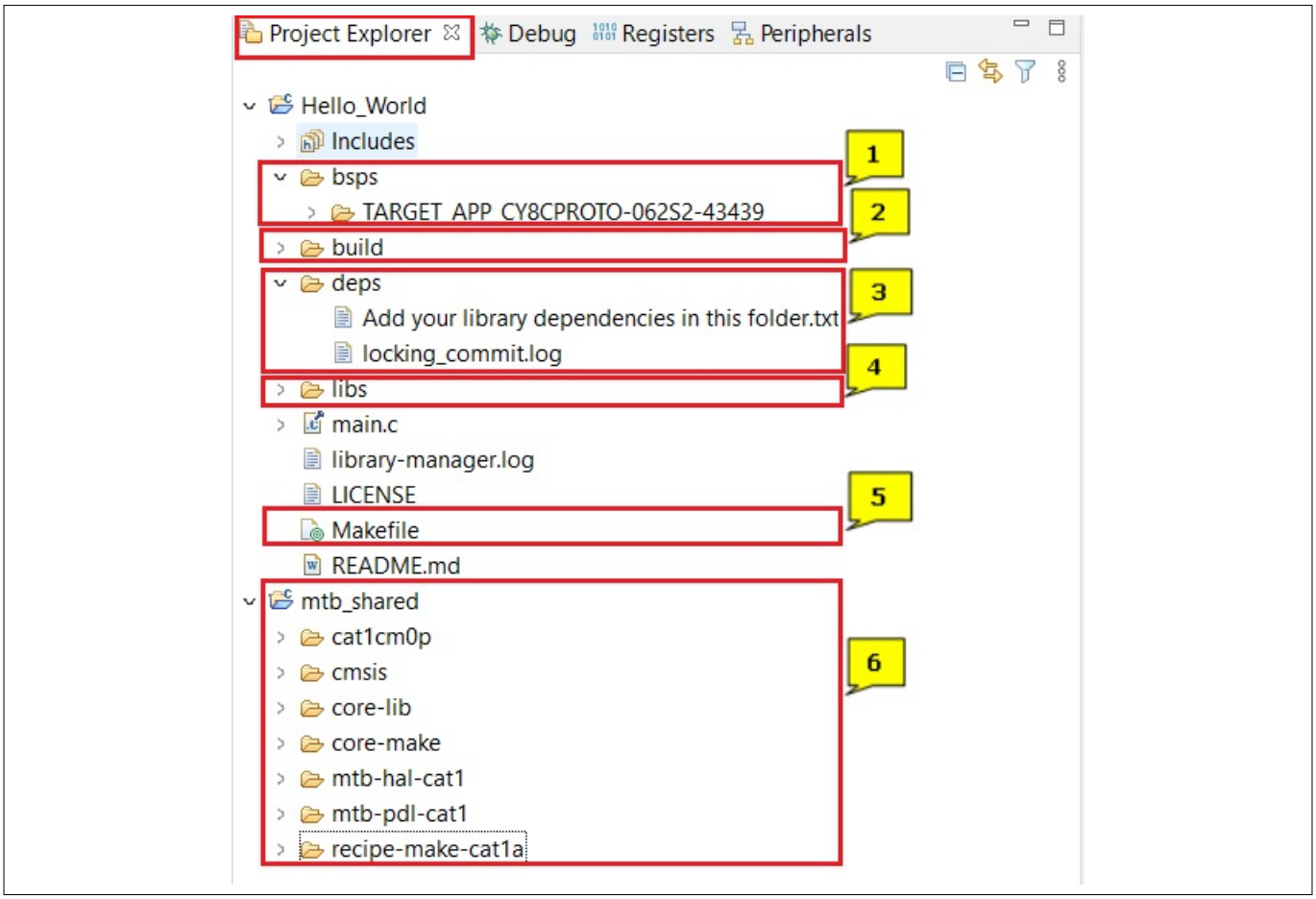

図 9 Project Explorer ビュー

1. BSP が提供するファイルは `bsps` フォルダにあり、`TARGET_<bsp name>` サブフォルダの下にリストされています。Device Configurator と周辺機器コンフィギュレーターのすべての入力ファイルは、BSP 内の `config` フォルダーにあります。BSP の `GeneratedSource` フォルダには、コンフィギュレーターによって生成されたファイルが格納されており、`cycfg_`というプレフィックスが付いています。これらのファイルには、BSP で定義されている設計コンフィギュレーションが含まれます。ModusToolbox™ 3.x 以降では、BSP は完全にアプリケーションの所有物であるため、デフォルトのデザイン コンフィギュレーター ファイルをカスタム デザイン コンフィギュレーター ファイルで上書きするのではなく、アプリケーション用に BSP のコンフィギュレーター ファイルを直接カスタマイズすることができます。BSP フォルダには、リンカ スクリプトやボード上で使用される PSOC™ 6 MCU のスタートアップ コードも含まれています。
2. build フォルダには、プロジェクトのビルドから生じるすべての結果の資産が含まれます。出力ファイルは、ターゲット BSP ごとに編成されます。
3. `deps` フォルダには `.mtb` ファイルが含まれており、ModusToolbox™がアプリケーションによって直接参照される library をプルする場所を提供します。これらのファイルには通常、ライブラリの各 GitHub の場所が含まれます。`..mtb` ファイルには、フェッチするライブラリのバージョンと、ライブラリを保存する場所に関するパスを示す git Commit Hash または Tag も含まれます。

例えば、retarget-io.mtb は `mtb://retarget-io#latest-v1.X#$$ASSET_REPO$$/retarget-io/latest-v1.X` を指します。変数$$ASSET_REPO$$は、デフォルトが `mtb_shared` の共有ロケーションルートを指します。共有ライブラリではなく、アプリケーションのローカル ライブラリでなければならない場合は、$$ASSET_REPO$$の代わりに$$LOCAL$$を使用してください。

4. `libs` フォルダには `.mtb` ファイルも含まれています。この場合、これらのファイルは、BSP または別のライブラリの依存関係として間接的に含まれるライブラリを指します。間接的な依存関係ごとに、ライブラリマネージャーはこのフォルダに `.mtb` ファイルを配置します。これらのファイルは、`deps` フォルダで使用可能なターゲットに基づいて入力されます。

   `libs` フォルダには、アプリケーションに必要なすべてのライブラリの相対パスを格納する `mtb.mk` ファイルが含まれます。ビルドシステムはこのファイルを使用して、アプリケーションに必要なすべてのライブラリを検索します。

   `libs` フォルダにあるものはすべてライブラリ マネージャによって生成されるので、そのフォルダにあるものを手動で編集するべきではありません。

5. アプリケーションには、ルートフォルダにある Makefile が含まれます。このファイルには、make ツールがアプリケーションプロジェクトをコンパイルおよびリンクするために使用する一連の指示が含まれます。1 つのアプリケーションに複数のプロジェクトが存在することもあります。その場合、アプリケーション レベルの Makefile と各プロジェクト内の Makefile があります。マルチプロジェクトアプリケーションの詳細については、AN215656 - PSOC™ 6 MCU デュアルコア システム設計を参照してください。

6. デフォルトでは、新しいアプリケーションを作成するか、既存のアプリケーションにライブラリを追加して共有として指定すると、すべてのライブラリはアプリケーション ディレクトリに隣接する `mtb_shared` ディレクトリに配置されます。

   `mtb_shared` フォルダは、ワークスペース内の異なるアプリケーション間で共有されます。必要に応じて、アプリケーションごとに異なるバージョンの共有ライブラリを使用することもできます。

5.4.1.1.1 Device Configurator を開く

BSP コンフィギュレーター ファイルは `bsps/TARGET_<BSP-name>/config` フォルダにあります。例えば **Project Explorer** から**<Application-name>**をクリックし、**Quick Panel** の **Device Configurator** リンクをクリックすると、下図のように **Device Configurator** で `design.modus` ファイルが開きます。また、他のコンフィギュレーションファイルをそれぞれのコンフィギュレーターで開くか、または **Quick Panel** で対応するリンクをクリックすることもできます。

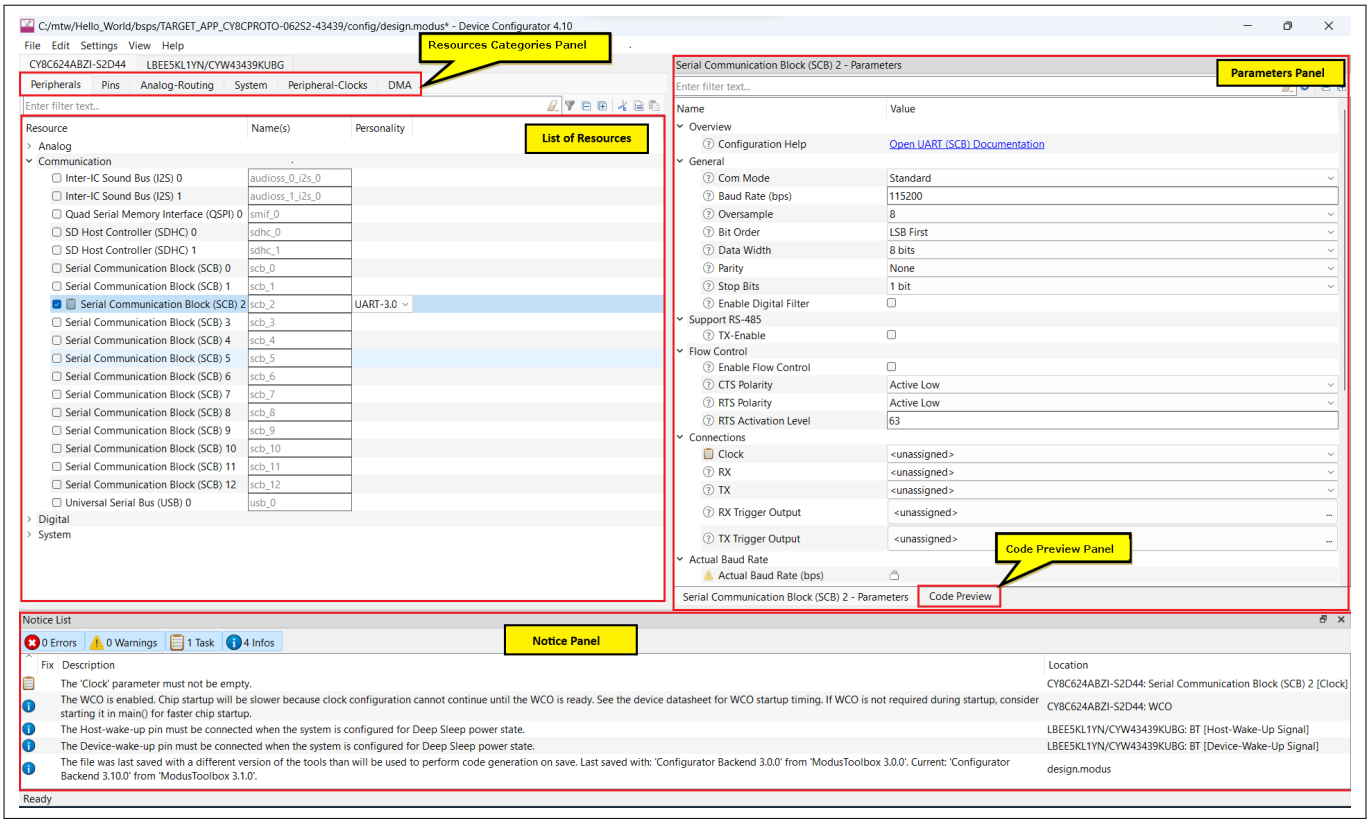

図 10　　Device Configurator

Device Configurator には、一連の **Resources Categories** タブがあります。ここでは、ペリフェラル、端子、クロックなどのデバイスで利用可能なさまざまなリソースを**リソース リスト**から選択できます。

リソースの **Personality** を選択することにより、リソースの動作を選択できます。たとえば、**Serial Communication Block (SCB)** リソースは、**EZI2C**、**I2C**、**SPI** または **UART** の特長を備えています。**Alias** は、ファームウェア開発で使用されるリソースの名前です。1 つ以上の別名は、カンマを使用してそれらを区切って指定できます (スペースなし)。

Parameters ペインでは、有効な各リソースと選択したパーソナリティのコンフィギュレーションパラメーターを入力します。**Code Preview** ペインには、選択したコンフィギュレーションパラメーターごとに生成されたコンフィギュレーション コードが表示されます。このコードは、`GeneratedSource` フォルダの `cycfg_`ファイルに生成されます。パラメータ ペインとコード プレビュー ペインは、別々のウィンドウではなくタブとして表示されることがありますが、内容は同じです。

コンフィギュレーションから生じるエラー、警告や情報メッセージは、**Notices** ペインに表示されます。

現在、Device Configurator は PDL ソースを使用した設定をサポートしています。アプリケーションで HAL ライブラリを使用する場合は、ここでデバイス設定を変更する必要はありません。アプリケーション プロジェクトには、CM4 コアのアプリケーションの作成に役立つソースファイル (`main.c`) が含まれます CM0+アプリケーションは `c` ファイル (`CY8C624ABZI-D44` デバイス用の `psoc6_02_cm0p_sleep.c`) として提供されます。cat1cm0p ライブラリを参照してください。。この `c` ファイルは、通常ビルド プロセスの一部としてコンパイルされ、CM4 イメージとリンクされます。

開発プロセスのこの時点で、必要なミドルウェアを設計に追加する準備が整った。Hello World アプリケーションに必要なミドルウェアは、retarget-io ライブラリだけです。

5.4.1.1.2　　retarget-io ミドルウェアの追加

このステップでは、retarget-io ミドルウェアを追加し、標準入力と出力ストリームを BSP により構成された UART にリダイレクトします。ミドルウェアの初期化は `main.c` コードで実行されます。

1. **Quick Panel** で、**Library Manager** リンクをクリックしてください。
2. 続くダイアログで、**Add Libraries** をクリックしてください。
3. **Peripherals** で、retarget-io を選択し、有効にしてください。
4. **OK** を クリックし、**Update** をクリックしてください。

に示すように、retarget-io ミドルウェアの使用に必要なファイルは `mtb_shared > retarget_io` フォルダに追加され、`.mtb` ファイルも `deps` フォルダに追加されます。

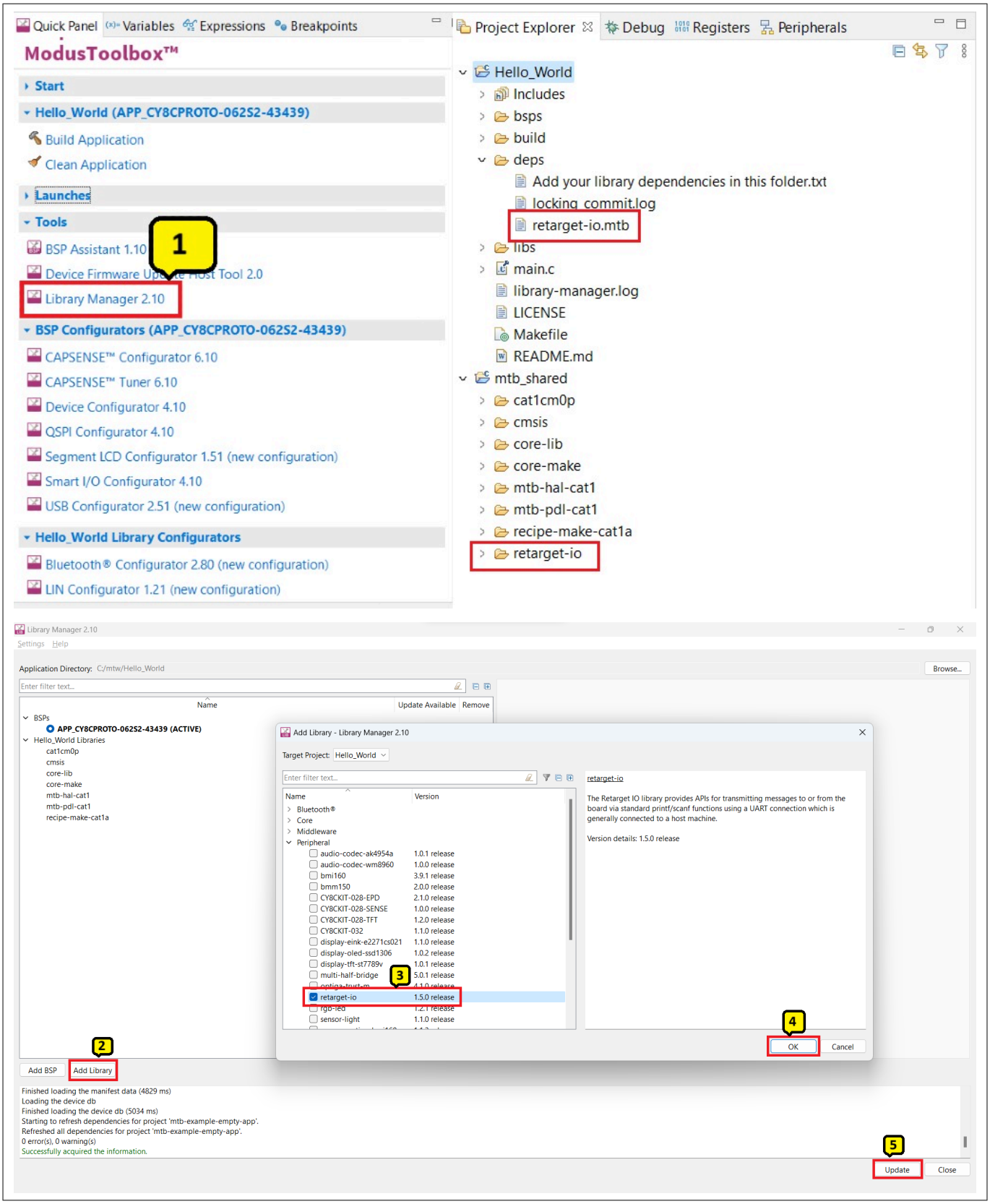

図 11　　retarget-io ミドルウェアの追加

5.4.1.1.3 UART、タイマー ペリフェラル、端子およびシステム クロックのコンフィギュレーション

デバッグ UART ペリフェラル、タイマー ペリフェラル、端子、およびシステム クロックのコンフィギュレーションは、BSP および HAL が提供する関数 API を使用してコードで直接実行できます。したがって、Device Configurator で設定する必要はありません。詳細については、ファームウェアの記述のセクションを参照してください。

5.4.1.2 ファームウェアの記述

開発プロセスのこの時点で、アプリケーション テンプレートを使用してアプリケーションを作成し、retarget-io ミドルウェアを追加する修正をしました。ここでは、設計機能を実装するファームウェアを書きます。

空の PSOC™ 6 スターター アプリケーションを使用してゼロから作業されている場合、本節で提供されるプログラム コードから対応ソース コードをアプリケーション プロジェクトの *main.c* にコピーできます。Hello World サンプル コードを使用している場合、すべての必要なファイルはアプリケーション内に既にあります。

ファームウェア フロー

ここでは、アプリケーションの `main.c` ファイルのコードを調べます。図 12 に、ファームウェアのフローチャートを示します。

CM0+コアはリセットから復帰して、CM4 コアを有効にします。次に、CM0+コアは提供される CM0+アプリケーションにより、スリープに入るように構成されます。このサンプルのリソース初期化は、CM4 コアによって実行されます。これはシステム クロック、端子、ペリフェラル接続のクロック、および他のプラットフォーム リソースを構成します。

CM4 コアが有効になると、クロックとシステム リソースは BSP 初期化機能により初期化されます。retarget-io ミドルウェアはデバッグ UART を使用するように構成され、ユーザー LED は初期化されます。デバッグ UART は端末エミュレータに「Hello World!」メッセージを出力し、オンボード KitProg3 は USB-UART ブリッジとして動作し、仮想 COM ポートを作成します。1000 ミリ秒ごとに割込みを生成するようにタイマー オブジェクトが構成されます。各タイマー割込みで、CM4 コアはキットの LED 状態を切り替えます。

ファームウェアは「Enter」キーを入力として受け付け、「Enter」キーが押されるたびにファームウェアが LED の点滅を開始または停止するように設計されています。

アプリケーション コードは、BSP/HAL/ミドルウェアの関数を使用して目的の機能を実行することに注意してください。

`cybsp_init()`この BSP 関数は、HAL ハードウェア マネージャをセットアップし、デバイスのすべてのシステム リソースを初期化しますが、システム クロックと電源レギュレータに限定されません。

`cy_retarget_io_init()`retarget-io ミドルウェアに含まれるこの関数は、デバッグ UART 端子用に設定されたエイリアスを使用して標準ボーレート 115200 でデバッグ UART を構成し、入力/出力ストリームをデバッグ UART にリダイレクトします。

***注**: デザイン コンフィギュレーターを開くと、BSP に設定されているエイリアスを表示できます。*

`cyhal_gpio_init()`GPIO HAL に含まれるこの関数は、物理端子を初期化して LED を駆動します。使用される LED は、BSP で設定された端子のエイリアスに基づきます。

`timer_init()`この関数は、ハードウェア タイマーをインスタンス化および構成する一連のタイマー HAL 関数呼び出しを含みます。また、タイマー割込みのコールバックも設定します。

以下のプログラムコードをアプリケーション プロジェクトの `main.c` ファイルにコピーしてください。

Code listing 1:main.c ファイル

```
#include "cyhal.h" #include "cybsp.h" #include "cy_retarget_io.h" /
******************************************************************************** * Macros
*******************************************************************************/ /* LED blink
timer clock value in Hz */ #define LED_BLINK_TIMER_CLOCK_HZ (10000) /* LED blink timer period
value */ #define LED_BLINK_TIMER_PERIOD (9999) /
******************************************************************************** * Function
Prototypes *******************************************************************************/
void timer_init(void); static void isr_timer(void *callback_arg, cyhal_timer_event_t event); /
******************************************************************************** * Global
Variables *******************************************************************************/ bool
timer_interrupt_flag = false; bool led_blink_active_flag = true; /* Variable for storing
character read from terminal */ uint8_t uart_read_value; /* Timer object used for blinking the
LED */ cyhal_timer_t led_blink_timer; /
******************************************************************************** * Function
Name: main ******************************************************************************** *
Summary: * This is the main functionfor CM4 core. It sets up a timer to trigger a * periodic
interrupt. The main while loop checks for the status of a flag set * by the interrupt and
toggles an LED at 1Hz to create an LED blinky. The * while loop also checks whether the 'Enter'
key was pressed and * stops/restarts LED blinking. * * Parameters: * none * * Return: * int *
*******************************************************************************/ int main(void)
{ cy_rslt_t result; /* Initialize the device and board peripherals */ result = cybsp_init(); /*
Board init failed. Stop program execution */ if (result != CY_RSLT_SUCCESS)
{ CY_ASSERT(0); } /* Enable global interrupts */ __enable_irq(); /* Initialize retarget-io to
use the debug UART port */ result = cy_retarget_io_init(CYBSP_DEBUG_UART_TX,
CYBSP_DEBUG_UART_RX, CY_RETARGET_IO_BAUDRATE); /* retarget-io init failed. Stop program
execution */ if (result != CY_RSLT_SUCCESS) { CY_ASSERT(0); } /* Initialize the User LED */
result = cyhal_gpio_init(CYBSP_USER_LED, CYHAL_GPIO_DIR_OUTPUT, CYHAL_GPIO_DRIVE_STRONG,
CYBSP_LED_STATE_OFF); /* GPIO init failed. Stop program execution */ if (result !=
CY_RSLT_SUCCESS) { CY_ASSERT(0); } /* \x1b[2J\x1b[;H - ANSI ESC sequence for clear screen */
printf("\x1b[2J\x1b[;H"); printf("****************** " "Hello World! Example
" "****************** \r\n\n"); printf("Hello World!!!\r\n\n"); printf("For more projects,
" "visit our code examples repositories:\r\n\n"); printf("https://github.com/Infineon/" "Code-
Examples-for-ModusToolbox-Software\r\n\n"); /* Initialize timer to toggle the LED */
timer_init(); printf("Press 'Enter' key to pause or " "resume blinking the user LED \r\n\r\n");
for (;;) { /* Check if 'Enter' key was pressed */ if (cyhal_uart_getc(&cy_retarget_io_uart_obj,
&uart_read_value, 1) == CY_RSLT_SUCCESS) { if (uart_read_value == '\r') { /* Pause LED blinking
by stopping the timer */ if (led_blink_active_flag) { cyhal_timer_stop(&led_blink_timer);
printf("LED blinking paused \r\n"); } else /* Resume LED blinking by starting the timer */
{ cyhal_timer_start(&led_blink_timer); printf("LED blinking resumed\r\n"); } /* Move cursor to
previous line */ printf("\x1b[1F"); led_blink_active_flag ^= 1; } } /* Check if timer elapsed
(interrupt fired) and toggle the LED */ if (timer_interrupt_flag) { /* Clear the flag */
timer_interrupt_flag = false; /* Invert the USER LED state */
cyhal_gpio_toggle(CYBSP_USER_LED); } } } /
******************************************************************************** * Function
Name: timer_init
******************************************************************************** * Summary: *
This function creates and configures a Timer object. The timer ticks * continuously and
produces a periodic interrupt on every terminal count * event. The period is defined by the
'period' and 'compare_value' of the * timer configuration structure 'led_blink_timer_cfg'.
Without any changes, * this application is designed to produce an interrupt every 1 second. * *
Parameters: * none *
```

```
*******************************************************************************/ void
timer_init(void) { cy_rslt_t result; const cyhal_timer_cfg_t led_blink_timer_cfg =
{ .compare_value = 0, /* Timer compare value, not used */ .period = LED_BLINK_TIMER_PERIOD, /*
Defines the timer period */ .direction = CYHAL_TIMER_DIR_UP, /* Timer counts up */ .is_compare
= false, /* Don't use compare mode */ .is_continuous = true, /* Run timer indefinitely
*/ .value = 0 /* Initial value of counter */ }; /* Initialize the timer object. Does not use
input pin ('pin' is NC) and * does not use a pre-configured clock source ('clk' is NULL). */
result = cyhal_timer_init(&led_blink_timer, NC, NULL); /* timer init failed. Stop program
execution */ if (result != CY_RSLT_SUCCESS) { CY_ASSERT(0); } /* Configure timer period and
operation mode such as count direction, duration */ cyhal_timer_configure(&led_blink_timer,
&led_blink_timer_cfg); /* Set the frequency of timer's clock source */
cyhal_timer_set_frequency(&led_blink_timer, LED_BLINK_TIMER_CLOCK_HZ); /* Assign the ISR to
execute on timer interrupt */ cyhal_timer_register_callback(&led_blink_timer, isr_timer,
NULL); /* Set the event on which timer interrupt occurs and enable it */
cyhal_timer_enable_event(&led_blink_timer, CYHAL_TIMER_IRQ_TERMINAL_COUNT, 7, true); /* Start
the timer with the configured settings */ cyhal_timer_start(&led_blink_timer); } /
******************************************************************************* * Function
Name: isr_timer
******************************************************************************** * Summary: *
This is the interrupt handler function for the timer interrupt. * * Parameters: * callback_arg
Arguments passed to the interrupt callback * event Timer/counter interrupt triggers *
*******************************************************************************/ static void
isr_timer(void *callback_arg, cyhal_timer_event_t event) { (void) callback_arg; (void)
event; /* Set the interrupt flag and process it from the main while(1) loop */
timer_interrupt_flag = true; }
```

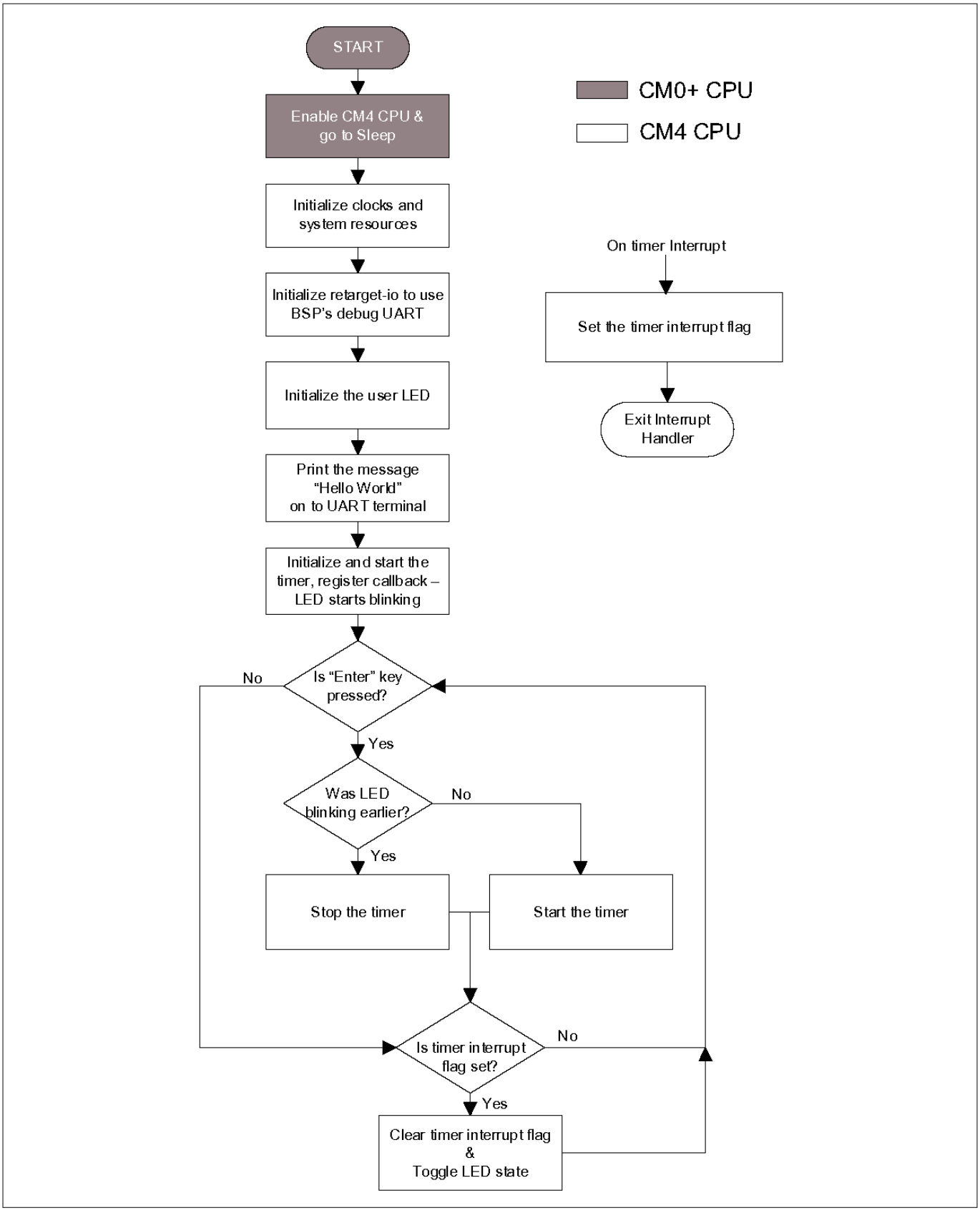

図 12　　ファームウェア フローチャート

以上、サンプル コードを用いてファームウェアの動作概要を説明しました。理解を深めるために、ご自由にソース ファイルを活用してください。

5.4.1.3 アプリケーションのビルド

ここでは、アプリケーションのビルド方法を説明します。

1. **Project Explorer** ビューでアプリケーション プロジェクトを選択してください。
2. **Quick Panel** の **<name>**グループにある **Build Application** ショートカットをクリックします。
   これにより、Makefile からビルドコンフィギュレーションが選択され、アプリケーションを構成するすべてのプロジェクトがコンパイル/リンクされます。デフォルトでは、デバッグコンフィギュレーションが選択されています。
3. 図 13 に示すように、**Console view** にはビルド動作の結果が一覧表示されます。

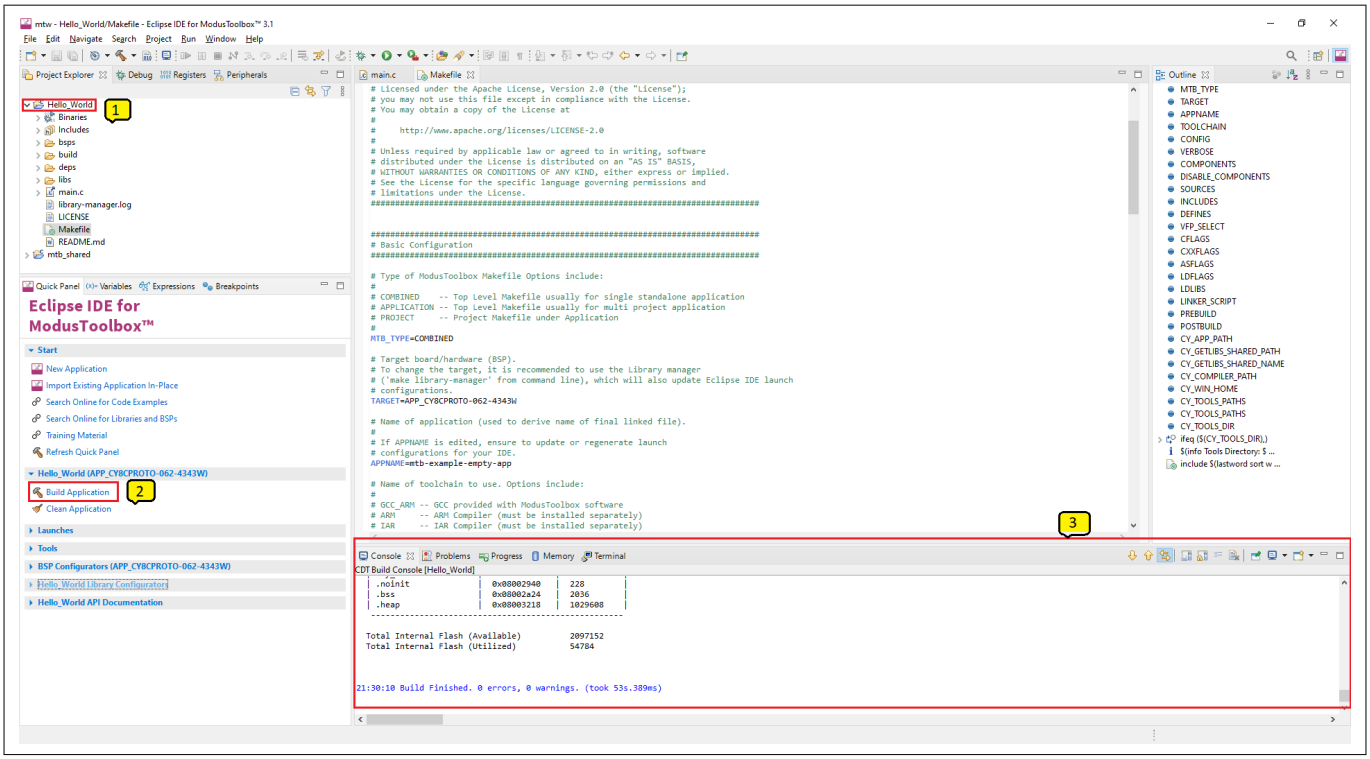

図 13 アプリケーションのビルド

エラーが発生した場合、前のステップに戻って必要な作業がすべて完了したことを確認してください。

*注: アプリケーションをビルドするために、コマンド ライン インターフェース (CLI) も使用できます。詳細なドキュメントについては、ModusToolbox™ tools package user guide の **Build system** の章を参照してください。この資料は、ModusToolbox™をインストールした `/docs_<version>/`フォルダにあります。*

5.4.1.4 デバイスのプログラミング

ここでは、PSOC™ 6 MCU のプログラム方法を説明します。

ModusToolbox™は、OpenOCD プロトコルを使用して PSOC™ 6 MCU 上のアプリケーションをプログラムおよびデバッグします。.キットは KitProg3 が動作している必要があります。いくつかのキットには、KitProg3 ではなく KitProg2 ファームウェアが同梱されます。詳細は KitProg3/MiniProg4 を使用したアプリケーションのデバッグを参照してください。ModusToolbox™には、KitProg ファームウェアを KitProg2 から KitProg3 に切り替える `fw-loader` コマンドライン ツールが含まれます。詳細は Eclipse IDE for ModusToolbox™ user guide の PSOC™ 6 MCU KitProg Firmware Loader セクションを参照してください。

内蔵プログラマを備えた開発キットを使用する場合、USB ケーブルを使用してボードをコンピュータに接続してください。

ご自身でハードウェア開発をされる場合、、https://www.segger.com/products/debug-probes/j-link/、またはhttps://www2.keil.com/mdk5/ulink/ulinkpro/などのハードウェア プログラマ/デバッガが必要な場合があります。

図 14 に示すように、アプリケーション プロジェクトを選択し、**Quick Panel** の **Launches** グループの下にある **<application name> Program (KitProg3_MiniProg4)**のショートカットをクリックしてください。IDE は適切な実行コンフィギュレーションを選択して実行します。

注: 直前のビルド以降にファイルが変更されている場合、このステップでもビルドが実行されます。

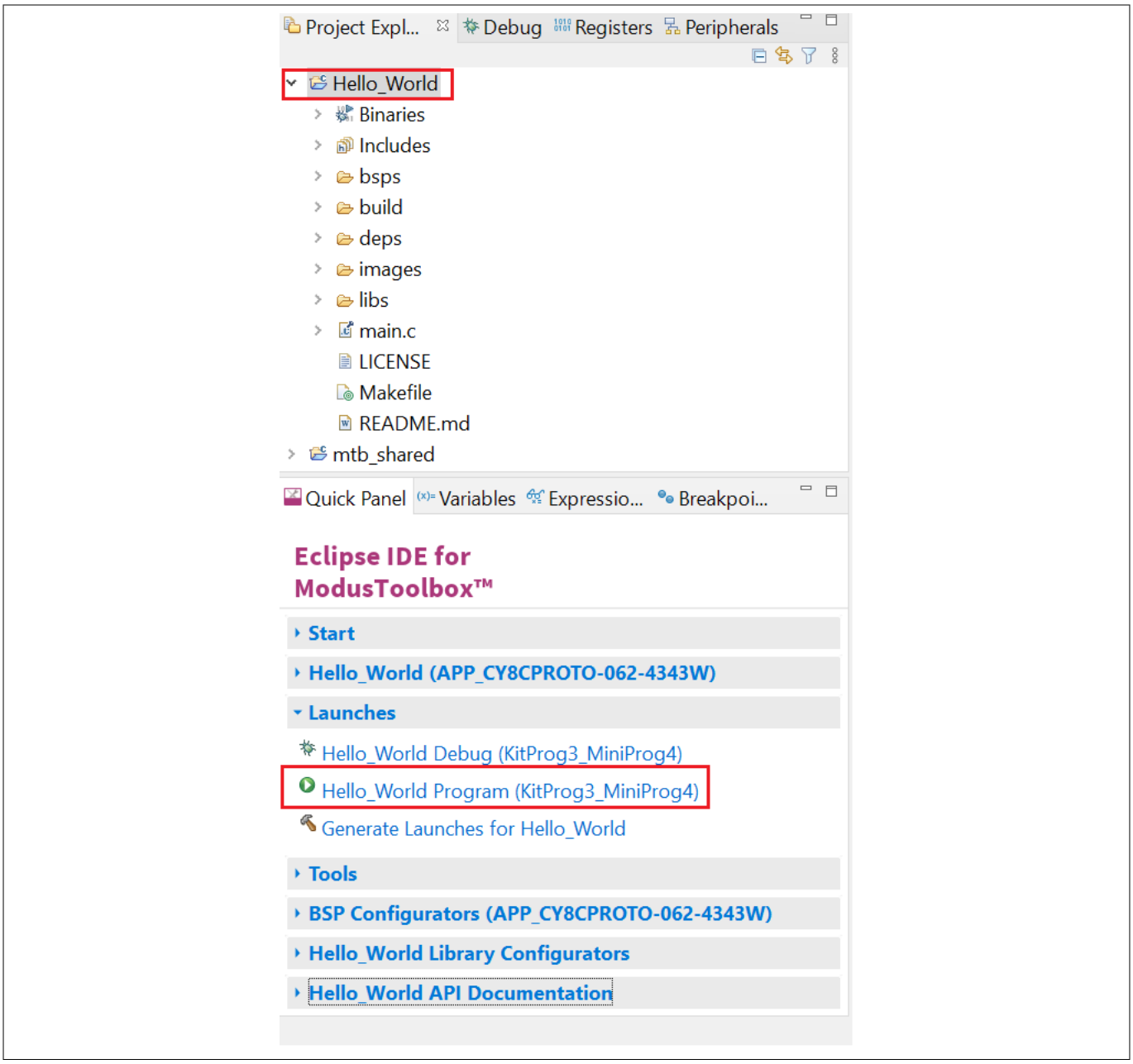

図 14 デバイスへのアプリケーション プログラミング

下図に示すように、**Console** ビューにはプログラミング動作の結果が一覧表示されます。

図 15　　デバイスへのアプリケーション プログラミング

5.4.1.5　設計のテスト

ここでは、設計のテスト方法について説明します。

これらの手順に従って、設計の出力を観察します。このノートでは UART ターミナル エミュレーターとして Tera Term を使用しているが、出力結果を見るには好きなターミナルを使うことができます。

1.　シリアル ポートの選択

図 16 に示すように、Tera Term を起動し、USB-UART COM ポートを選択してください。COM ポート番号は異なる場合があることに注意してください。

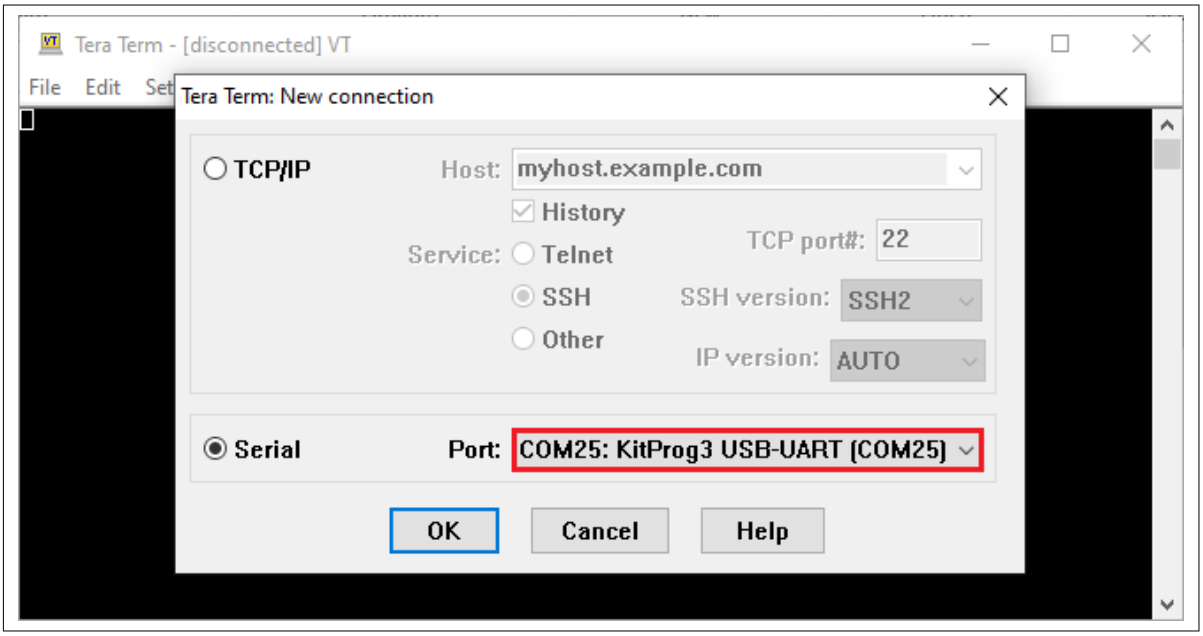

図 16　　Tera Term で KitProg3 COM ポートの選択

2.　ボーレートの設定

図 17 に示すように、**Setup > Serial port** でボーレートを 115200 に設定してください。

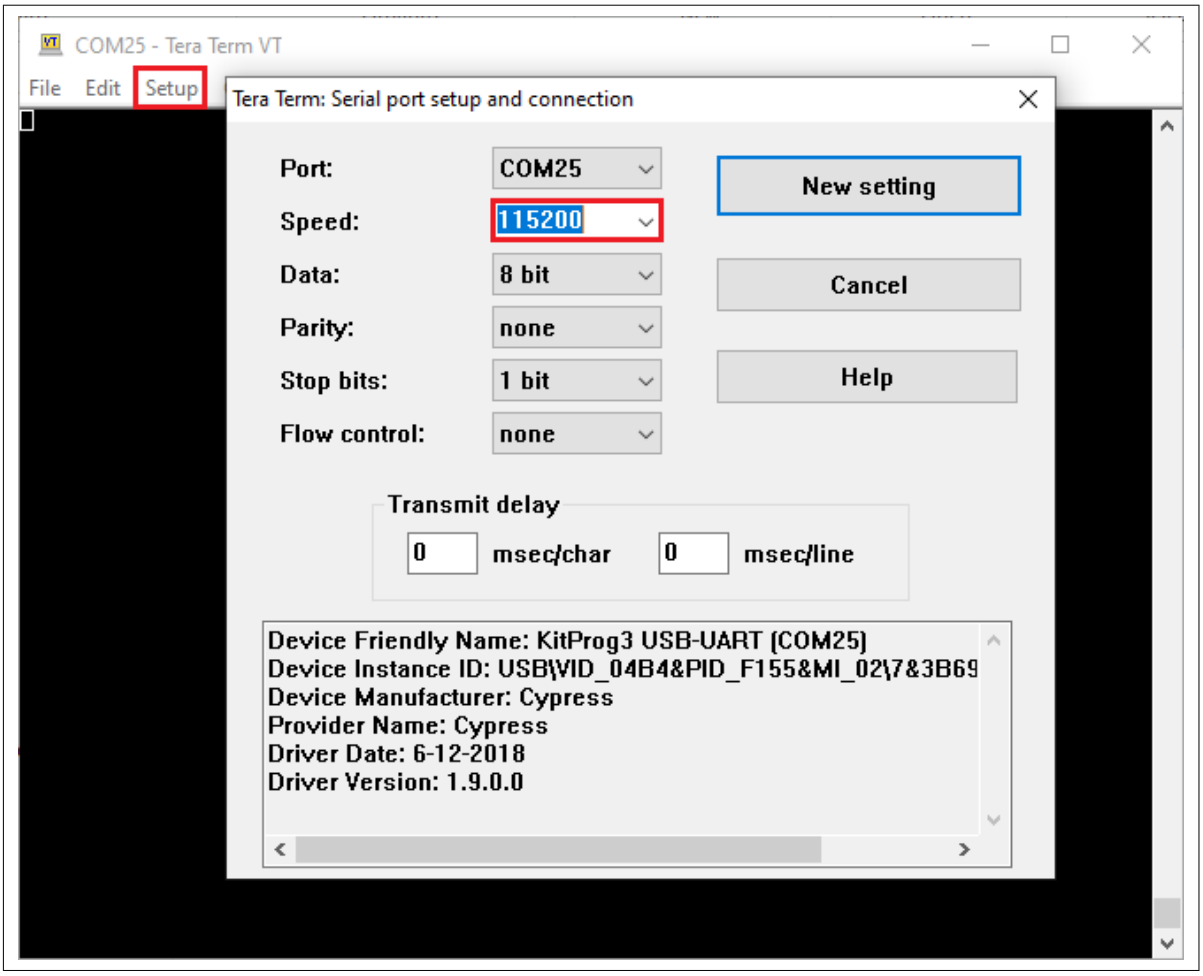

図 17　　Tera Term でボーレートの設定

3.　デバイスのリセット

キットのリセット スイッチ (**SW1**) を押してください。図 18 に示すように、端末にメッセージが表示されます。キットのユーザー LED が点滅を始めます。

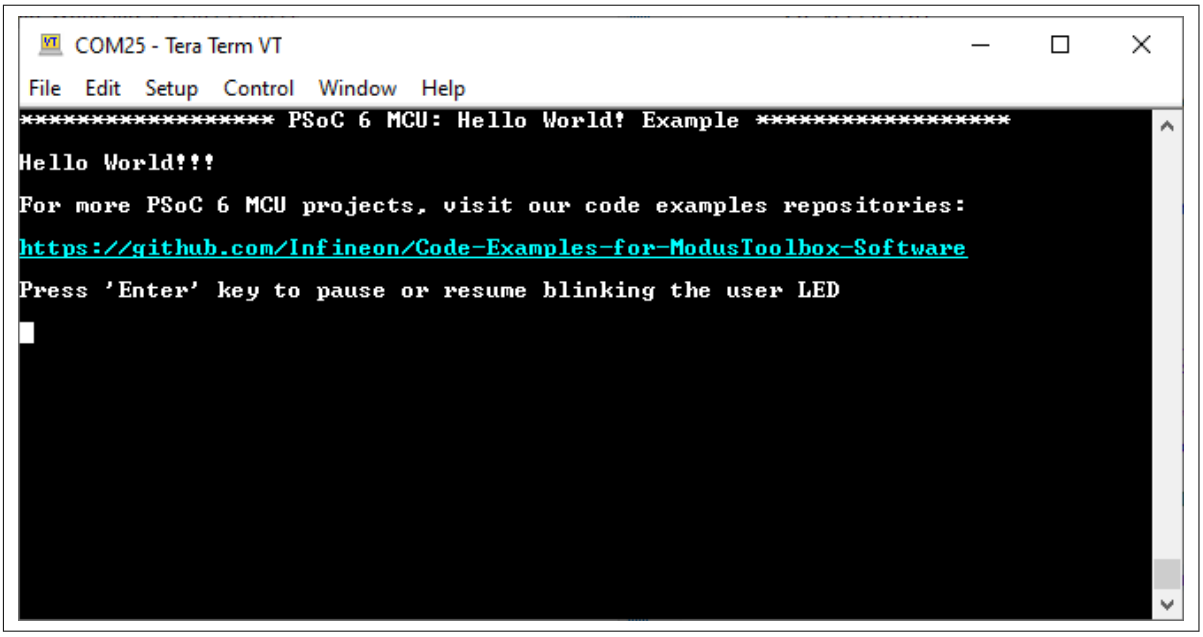

図 18　　表示された UART メッセージ

4.　LED 点滅機能の一時停止/再開

Enter キーを押して、LED 点滅を一時停止/再開してください。図 19 に示すように、LED 点滅が一時停止すると、対応するメッセージが端末に表示されます。

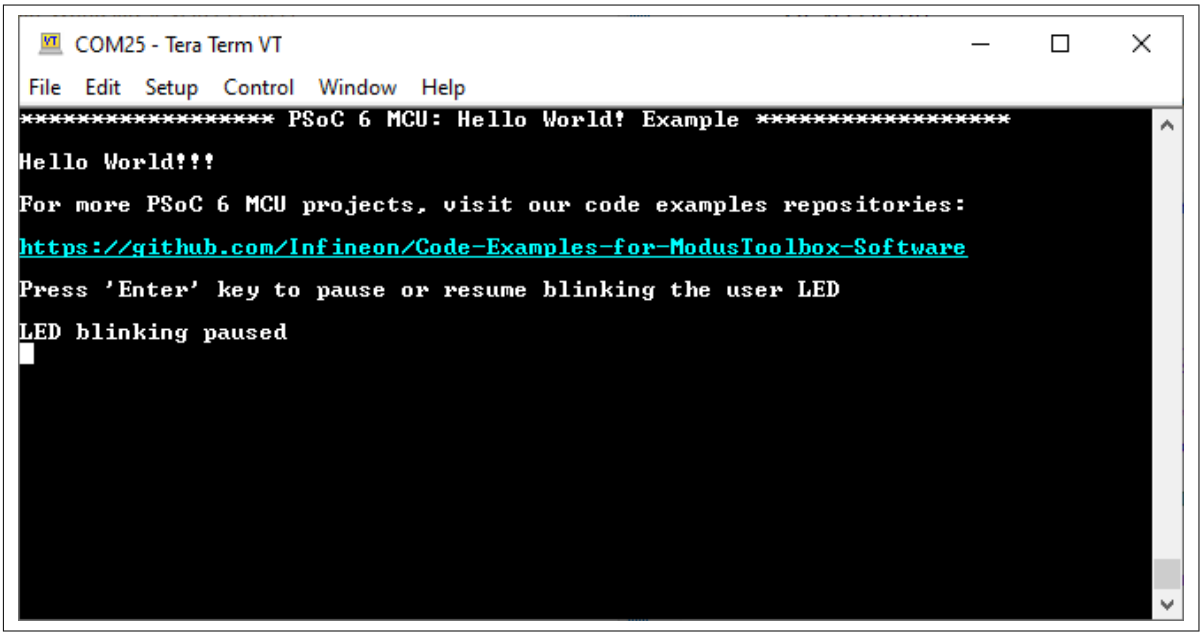

図 19　　表示された UART メッセージ

5.4.1.6　KitProg3/MiniProg4 を使用したアプリケーションのデバッグ

すべての PSOC™ 6 キットは、KitProg3 オンボード プログラマ/デバッガを備えます。これは Cortex® Microcontroller Software Interface Standard のデバッグ アクセス ポート (CMSIS-DAP) をサポートします。詳細は KitProg3 user guide を参照してください。Eclipse IDE は KitProg3 を必要とし、PSOC™ 6 MCU アプリケーションのデバッグに OpenOCD プロトコルを使用します。

注: *PSOC™ 6 Wi-Fi-Bluetooth® pioneer kit (CY8CKIT-062-WiFi-BT) および PSOC™ 6 Bluetooth® LE pioneer kit (CY8CKIT-062-BLE) にはファームウェアにプリインストールされた KitProg2 オンボードプログラマ/デバッガがあります。ModusToolbox™ で動作させるには、ModusToolbox™ ソフトウェアに含まれる fw-loader コマンドラインツールを使用してファームウェアを KitProg3 にアップグレードしてください。詳細は Eclipse IDE for ModusToolbox™ user guide の PSOC™ 6 Programming/Debugging - KitProg Firmware Loader セクションを参照してください。*

Eclipse IDE には、デバイスのプログラミングやデバッガーの起動に関するさまざまな設定を制御する、いくつかの起動コンフィギュレーションが含まれています。キットや使用するアプリケーションの種類によって、さまざまな起動コンフィギュレーションが可能です。このようなコンフィギュレーションが KitProg3/MiniProg4 の起動コンフィギュレーションです。起動コンフィギュレーションの詳細については、Eclipse IDE for ModusToolbox™ user guide の「PSOC™ MCU プログラミング/デバッグ」セクションを参照してください。

アプリケーションが作成されると、下図のように **Quick Panel** の **Launches** の下に、ツールは KitProg3_MiniProg4 用の起動コンフィギュレーションを生成します。

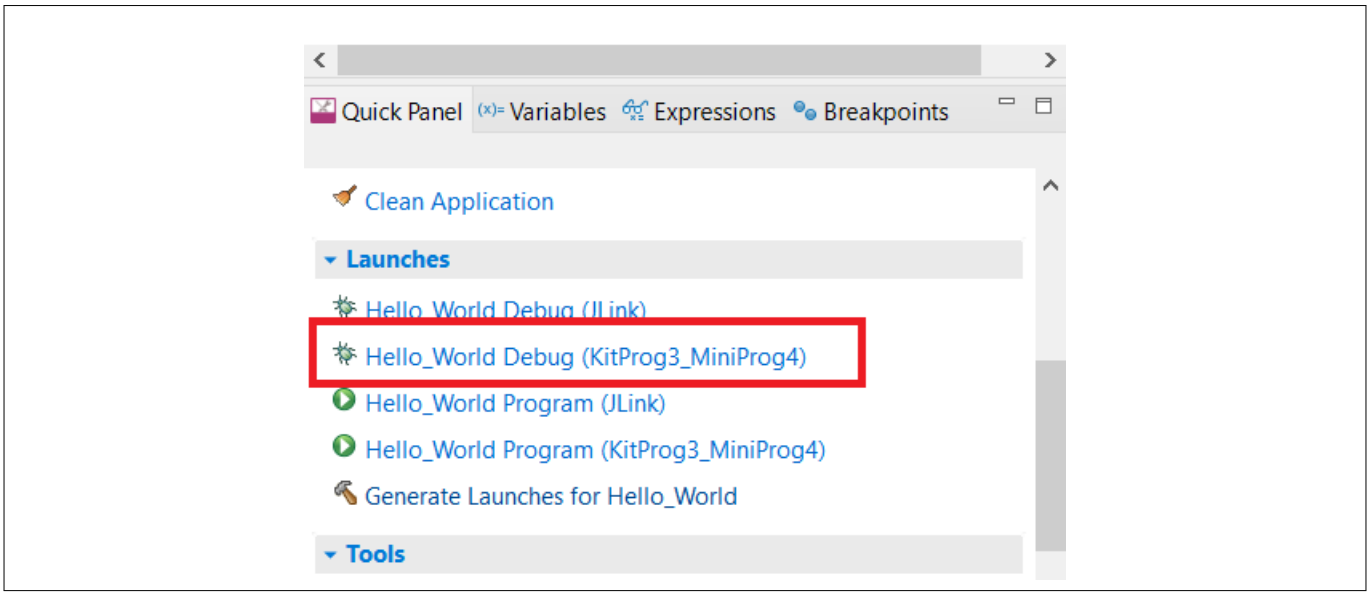

図 20 KitProg3/MiniProg4 起動コンフィギュレーション

デバイスをホスト マシンに接続し、図 20 に示すように、**Hello_World Debug (KitProg3_MiniProg4)** launch をクリックしてデバッグを開始してください。デバッグが開始されると、実行は main()関数で停止し、ユーザーは下図のように main()の先頭からデバッグを開始できます。

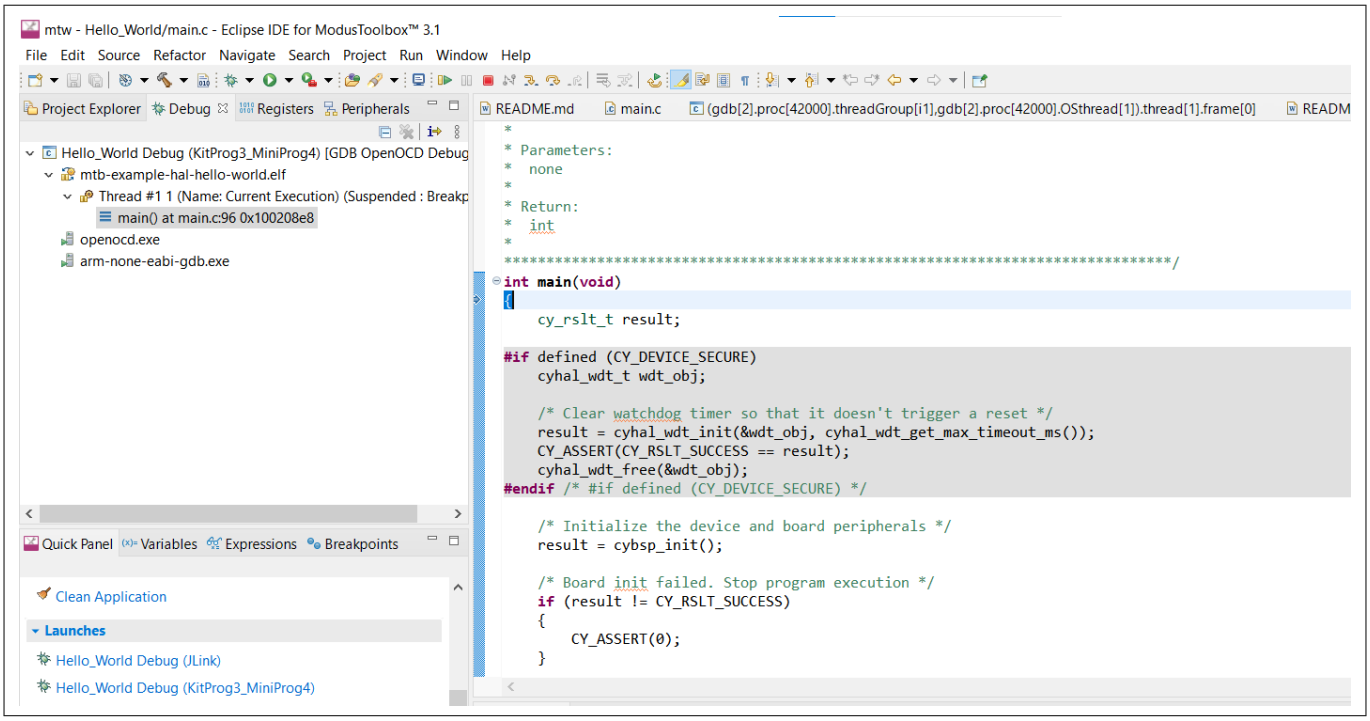

図 21　　main()デバッグ

5.4.2　ModusToolbox™用 Visual Studio Code (VS コード)

VS コードで新しいアプリケーションを作成するには、Visual Studio Code for ModusToolbox™ user guide を 参照してください。

5.4.3　ModusToolbox™用 IAR Embedded Workbench

IAR で新しいアプリケーションを作成するには、IAR Embedded Workbench for ModusToolbox™ user guide 照してください。

5.4.4　ModusToolbox™用 Keil µVision

Keil uVision で新しいアプリケーションを作成するには、Keil µVision for ModusToolbox™ user guide を 参照してください。

6 まとめ

本アプリケーション ノートは PSOC™ 6 MCU デバイス アーキテクチャおよび関連する開発ツールを説明しました。PSOC™ 6 MCU は、構成可能なアナログおよびデジタルペリフェラル機能、メモリ、およびデュアルコア システムを単一のチップに統合した真のプログラム可能な組込みシステムオンチップです。統合された機能、組込みセキュリティ、および低消費電力モードにより、PSOC™ 6 MCU はスマート家電製品、IoT ゲートウェイ、およびその他の関連アプリケーションに理想的なデバイスです。

関連資料

PSOC™ 6 MCU のサンプル コードの完全で更新されたリストについては、当社の GitHub を参照してください。その他の PSOC™ 6 MCU 関連文書については、PSOC™ 6 MCU Digital Documentation Portal を参照してください。

表 2 に、PSOC™ 6 MCU と ModusToolbox™の学習の次段階に推奨されるシステム レベルおよび一般的なアプリケーションノートを示します。

表 2　一般的なシステム レベルのアプリケーション ノート

文書番号	文書名
AN218241	PSOC™ 6 MCU ハードウェア設計上の注意事項

表 3 に特定のペリフェラルとアプリケーション用のアプリケーション ノート (AN) を示します。

表 3　PSOC™ 6 MCU 機能に関するドキュメント

文書番号	文書名
システム リソース、CPU、割込み	
AN215656	PSOC™ 6 MCU デュアルコア システム設計
AN217666	PSOC™ 6 MCU 割込み
AN235279	Performing ETM and ITM Trace on PSOC™ 6 MCU
CAPSENSE™	
AN92239	CAPSENSE™による近接センシング
AN85951	PSOC™ 4 および PSOC™ 6 MCU CAPSENSE™デザインガイド
デバイス ファームウェア更新	
AN213924	PSOC™ 6 MCU デバイス ファームウェア アップデート ソフトウェア開発キット ガイド
Low-power	
AN230938	PSOC™ 6 MCU 低電力アナログ
AN219528	PSOC™ 6 MCU low-power modes and power reduction techniques
セキュリティ	
AN221111	PSOC™ 6 MCU MCU designing a custom secured system
AN227860	PSOC™ 64 Secure MCU Secure Boot SDK User Guide
AN239061	PSOC™ 64 security getting started guide
ModusToolbox™	
ModusToolbox™ tools package installation	
ModusToolbox™ tools package release notes	
ModusToolbox™ tools package quick start guide	
ModusToolbox™ tools package user guide	
Eclipse IDE for ModusToolbox™ user guide	
Visual Studio Code for ModusToolbox™ user guide	
Keil µVision for ModusToolbox™ user guide	
IAR Embedded Workbench for ModusToolbox™ user guide	

用語集

ここでは、PSOC™ ファミリーで操作する際によく出てくる用語について説明します。

- ボード サポート パッケージ (BSP): BSP は、ボード固有のドライバおよびその他の機能を含むファームウェアのレイヤです。ボード サポート パッケージは、ボードを初期化し、ボード レベルのペリフェラルにアクセスできるファームウェア API のライブラリ セットです。
- インフィニオン プログラマ: インフィニオン プログラマは、インフィニオンのデバイスをプログラムする柔軟なクロスプラットフォームアプリケーションです。これはターゲット デバイスのプログラム、消去、検証およびフラッシュの読み出しができます。
- ハードウェアアブストラクションレイヤ (HAL): HAL は、低レベル ドライバ (MTB-PDL-CAT1 など) を包含し、MCU に高レベル インターフェースを提供します。インターフェースは、任意の MCU で動作するように抽象化されています。
- KitProg: KitProg は、USB-I2C および USB-UART ブリッジ機能を備えたオンボードのプログラマ/デバッガです。KitProg はほとんどの PSOC™開発キットに統合されています。
- MiniProg3/MiniProg4: 開発用のプログラミング ハードウェアであり、内蔵のプログラマがサポートされていないカスタムボードまたは PSOC™開発キットの上に PSOC™デバイスをプログラムするために使用されます。
- Personality (パーソナリティ): これは、機能に対してリソースが構成可能かを表します。例えば、SCB リソースは UART、SPI または I2C パーソナリティに構成できます。
- PSOC™: 32 ビット Arm® Cortex®-M0 のような CPU、プログラム可能なアナログとデジタル ブロックを内蔵する、プログラム可能な組込み設計プラットフォームです。タッチ センシングなどの信頼性が高く、使いやすいソリューションにより組込みシステムの設計を加速させ、低消費電力の設計を可能にします。
- ミドルウェア: ミドルウェアは、アプリケーションに特定の機能を提供するファームウェア モジュールのセットです。いくつかのミドルウェアはネットワーク プロトコル (例:MQTT) を提供し、また、その他にはデバイス機能 (例:USB、オーディオ) への高レベル ソフトウェア インターフェースを提供するものがあります。
- ModusToolbox™: IoT 設計者向けの Eclipse ベースの組込み設計プラットフォームであり、業界で最も展開されている Wi-Fi および Bluetooth®テクノロジーと、クラス最高のセンシングを備えた最も低消費電力で柔軟性の高い MCU を組み合わせ、シングルで一貫性があり、他の製品に近い設計体験を提供します。
- ペリフェラル ドライバ ライブラリ (PDL): ペリフェラル ドライバ ライブラリ (PDL) は、PSOC™ 6 MCU アーキテクチャのソフトウェア開発を簡素化します。PDL により、レジスタの使用法とビット構造を理解する必要性が減るため、利用できる広範なペリフェラルセットのソフトウェア開発が容易になります。

改訂履歴

版数	発行日	変更内容
**	2020-02-07	これは英語版 002-28571 Rev. **を翻訳した日本語版 Rev. **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 screenshots with latest release of ModusToolbox™ Added new supported PSOC™ 6 MCU devices Added AnyCloud description under ModusToolbox™ software
英語版(*B)	-	この版は英語版のみです。英語版の改訂内容: Added new supported PSOC™ 6 MCU device – PSOC™ 62S4 Added information on PSOC™ 6 product lines and development kits available for each product line
*A	2020-12-10	これは英語版 002-28571 Rev. *C を翻訳した日本語版 Rev. *A です。英語版の改訂内容: Updated 図 2 Updated Screenshots with MTB v2.2 Added mtb_shared folder description, updated application creation process with MTB flow
*B	2021-05-13	これは英語版 002-28571 Rev. *D を翻訳した日本語版 Rev. *B です。英語版の改訂内容: Updated to Infineon template
*C	2021-08-20	これは英語版 002-28571 Rev. *E を翻訳した日本語版 Rev. *C です。英語版の改訂内容: Updated the GitHub links Added reference to new PSOC™ 6 MCU low-power analog Updated 図 16 to 図 19 Firmware updated to the latest version
英語版(*F)	-	この版は英語版のみです。英語版の改訂内容: Template update
英語版(*G)	-	この版は英語版のみです。英語版の改訂内容: Updated the development flow as per MTB v3.0 software Updated link references Updated configurator info Added new Figure 2 and Figure 7 Updated Figure 8 and Figure 14 Added reference to new AN235279 Added a new section for ModusToolbox™ Applications Removed reference to deprecated AN225588 Updated Figure 17 to Figure 19
英語版(*H)	-	この版は英語版のみです。英語版の改訂内容: Updated content with latest release of c™ 3.1
英語版(*I)	-	この版は英語版のみです。英語版の改訂内容: Updated with kit CY8CPROTO-062S2-43439
*D	2024-10-30	これは英語版 002-28571 Rev. *J を翻訳した日本語版 Rev. *D です。英語版の改訂内容: Added references to all the supporting IDEs on ModusToolbox™ and restructured the content as per the MKTG need.

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 2024-10-30

Published by

Infineon Technologies AG

81726 Munich, Germany

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

Email: erratum@infineon.com

Document reference
IFX-ofg1649405242329

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。