

μ T-Kernel 3.0 構築手順書 (CY8CKIT-CY8C6)

Rev 3.00.06

January, 2023



μ T-Kernel 3.0

目次

1. はじめに	2
1.1. 本書について	2
1.2. 表記について	2
2. 概要	3
2.1. 対象とするハードウェアと OS	3
2.2. 対象とする開発環境	3
2.3. デバイスドライバ	4
2.4. 関連ドキュメント	4
3. 開発環境の準備	5
3.1. ModusToolbox のインストール	5
3.2. ModusToolbox の動作確認	5
4. プロジェクトの作成	12
4.1. ModusToolbox でサンプルアプリケーションを生成	12
4.2. μT-Kernel 3.0 を Git のサブモジュールとして追加	12
4.3. サブモジュールのブランチを CY8CKIT-062S2-43012 対応版に変更	14
4.4. プログラムのベースを μT-Kernel 3.0 に変更	16
4.5. Device Configurator でコンソール用に SCB5 を設定	16
4.6. ビルド、実行	22
5. μT-Kernel 3.0 のディレクトリ／ファイル構成	28
5.1. アプリケーションプログラムの追加	28
5.2. ビルド用のファイル	30
5.3. μT-Kernel 3.0 のソースコード	31

1. はじめに

本品はユーシーテクノロジー(株)が開発した CY8CKIT-062S2-43012(CY8CKIT-CY8C6) 対応の μ T-Kernel 3.0 である。



本品は、トロンプフォーラムの配布する μ T-Kernel 3.0 をベースに、Infineon 製の評価ボード CY8CKIT-062S2-43012 で動作させるための機種依存部を追加してある。

1.1. 本書について

本書は CY8CKIT-062S2-43012 向けの μ T-Kernel 3.0 の構築手順について記載した構築手順書である。本書の対象となる実装は、ユーシーテクノロジー(株)が公開している μ T-Kernel 3.0 BSP(Board Support Package)に含まれている。

以降、単に OS や RTOS と称する場合は μ T-Kernel 3.0 を示し、**本実装**と称する場合は CY8CKIT-062S2-43012 向けのソースコードの実装を示すものとする。

1.2. 表記について

表記	説明
[]	[]はソフトウェア画面のボタンやメニューを表す。
「 」	「 」はソフトウェア画面に表示された項目などを表す。
	注意が必要な内容を記述する。
	補足やヒントなどの内容を記述する。

2. 概要

本書では、μT-Kernel 3.0 BSP (Board Support Package)の使用方法について説明する。

μT-Kernel 3.0 BSP は、特定のマイコンボード等のハードウェアに対して移植した μT-Kernel 3.0 の開発および実行環境一式を提供するものである。

2.1. 対象とするハードウェアと OS

開発対象のハードウェアおよび OS は以下である。

表 2-1 開発対象のハードウェアと OS

分類	名称	備考
マイコン	CY8C624ABZI-S2D44A0 (ARM Cortex-M4F, Cortex-M0+)	Infineon Technologies AG
OS	μT-Kernel 3.00.06	トロンフォーラム
実機 (マイコンボード)	CY8CKIT-062S2-43012 (CY8CKIT-CY8C6)	Infineon Technologies AG

- ❶ 本実装の最新版は、ユーシーテクノロジー(株)の GitHub リポジトリにて公開している。

https://github.com/UCTechnology/mtk3_bsp

- ❷ μT-Kernel 3.0 の最新版は以下の GitHub リポジトリにて公開されている。

https://github.com/tron-forum/mtkernel_3

- ❸ 対象マイコンボード(CY8CKIT-062S2-43012)に関しては Infineon Technologies AG のサイトを参照のこと。

<https://www.infineon.com/cms/en/product/evaluation-boards/cy8ckit-062s2-43012/>

2.2. 対象とする開発環境

対象とする開発環境は以下である。

開発を行うホスト PC の OS は Windows とする。動作確認は Windows 10 にて行った。

表 2-2 開発環境

分類	名称	備考
開発環境	ModusToolbox Version 3.0.0	Infineon Technologies AG

- ❶ バージョンは動作確認に使用したバージョンを示している。

2.3. デバイスドライバ

μ T-Kernel 3.0 BSP では、トロンフォーラムが提供する μ T-Kernel 3.0 のサンプル・デバイスドライバを、対象となる実機に移植して実装している。

以下に本実装に含まれるデバイスドライバを示す。

表 2-3 本実装に含まれるデバイスドライバ

種別	デバイス名	デバイス	IO ピン	コネクタ
UART	serb	SCB1	P10[0:3]	ARD_A[0:3]
	serc	SCB2	P9[0:3]	ARD_A[8:11]
	serf	SCB5	P5[0:3]	ARD_D[0:3]

2.4. 関連ドキュメント

表 2-4 関連ドキュメント一覧

分類	名称	発行
OS 仕様	μ T-Kernel 3.0 仕様書 (Ver.3.00.00)	トロンフォーラム TEF020-S004-3.00.00
デバイスドライバ	μ T-Kernel 3.0 デバイスドライバ 説明書 (Ver.1.00.2)	トロンフォーラム TEF033-W007-210331
実装仕様書	μ T-Kernel 3.0 実装仕様書 (CY8CKIT-CY8C6)	ユーシーテクノロジー(株)
構築手順書 (本書)	μ T-Kernel 3.0 構築手順書 (CY8CKIT-CY8C6)	ユーシーテクノロジー(株)

- ① トロンフォーラムが発行するドキュメントは、トロンフォーラムの Web ページ、または GitHub で公開する μ T-Kernel 3.0 のソースコードに含まれている。

<https://www.tron.org/ja/specifications/>
https://github.com/tron-forum/mtkernel_3

- ① ユーシーテクノロジー(株)が発行するドキュメントは、ユーシーテクノロジー(株)の GitHub で公開する μ T-Kernel 3.0 のソースコードに含まれている。

https://github.com/UCTechnology/mtk3_bsp

3. 開発環境の準備

μT-Kernel 3.0 BSP を使用するにあたり、以下の手順で開発環境の準備を行う。

3.1. ModusToolbox のインストール

Infineon の Web サイト(下記)から ModusToolbox をダウンロードする。

<https://www.infineon.com/cms/jp/>

中央の一覧から[設計サポート]→[開発ツール]を選択すると右側に[開発ツール概要]が表示される、この中の[ModusToolbox™ソフトウェア]を選択すると ModusToolbox の日本語ページが表示される。

<https://www.infineon.com/cms/jp/design-support/tools/sdk/modustoolbox-software/>

このページで[Download]を選択し、OS に合わせたインストーラをダウンロードする。
インストーラを実行し、指示に従って ModusToolbox のインストールを進める。

- ◆ 上部にある一覧の[設計サポート]経由で選択していくと英語ページが表示される。
- ◆ 2023/01/23 時点での ModusToolbox の最新バージョンは、Version 3.0.0 である。
本資料では移植作業に使用した Version 3.0.0 を基に説明する。

3.2. ModusToolbox の動作確認

本実装ではハードウェアの初期化処理を ModusToolbox が提供する関数で行っている。このため、新しいデバイスを利用する場合に開発者がデバイスの初期化処理を個別に実装する必要はない。新しいデバイスを利用する場合は、ModusToolbox の Device Configurator を用いてデバイスの構成(入出力端子、クロック、分周器など)を指定し、Device Configurator が生成した初期化関数を呼び出してデバイスを初期化することができる。そこで、まずは ModusToolbox が生成したサンプルコードが正常に動作することを実機で確認する。

- ◆ Device Configurator で指定した情報のうち、クロックなどの共通部分の初期化は cybsp_init()で実行されるが、デバイス毎の初期化処理は個別に初期化関数を呼び出す必要がある。詳細については、Device Configurator や生成されたサンプルコードのドキュメントを参照すること。
- ◆ ModusToolbox や Device Configurator が自動生成する全てのサンプルコードが μT-Kernel 3.0 で利用できることを保証するものではない。自動生成されたサンプルコードを利用する際は十分に検証したうえで利用すること。

以下では、ModusToolbox が生成する基本的なサンプルプロジェクトの 1 つである Hello World を用いて動作確認する手順について説明する。

- (1) Eclipse IDE for ModusToolbox を起動し、ワークスペースを新規に作成する。

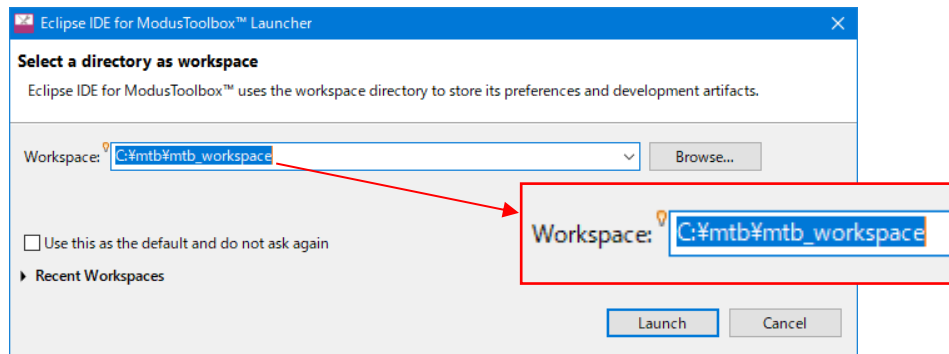


図 1 workspace の指定

上記の例では、C:\mtb\mtb_workspace を指定している。

- (2) Eclipse IDE for ModusToolbox が起動する。

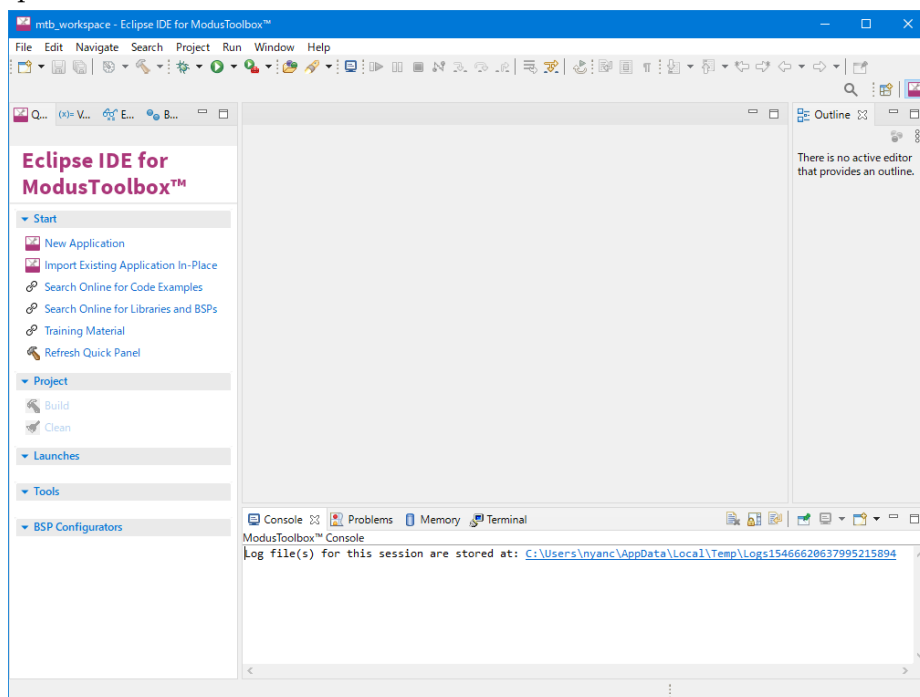


図 2 Eclipse が起動

- (3) [Quick Panel]の[Start]から[New Application]を選択する。

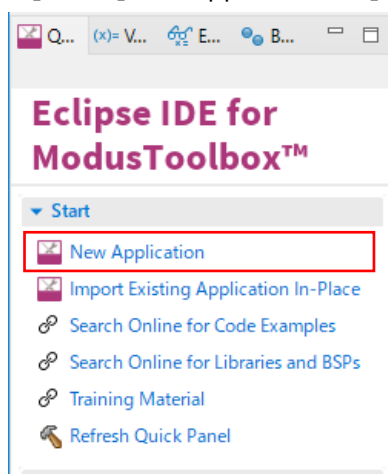


図 3 Project Creator を起動

(4) Project Creator が起動する。

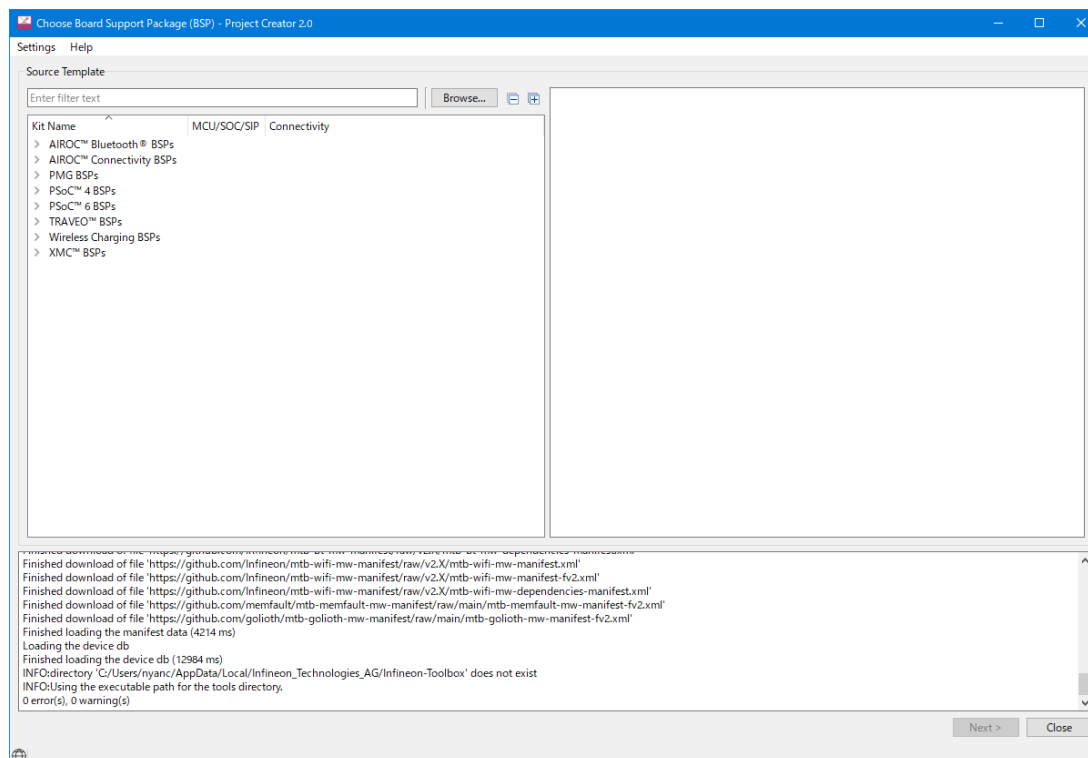


図 4 Project Creator が起動

(5) ボードを選択するため検索窓に CY8CKIT-062S2-43012 と入力して、表示されたターゲットボードを選択し、右下の [Next] をクリックする。

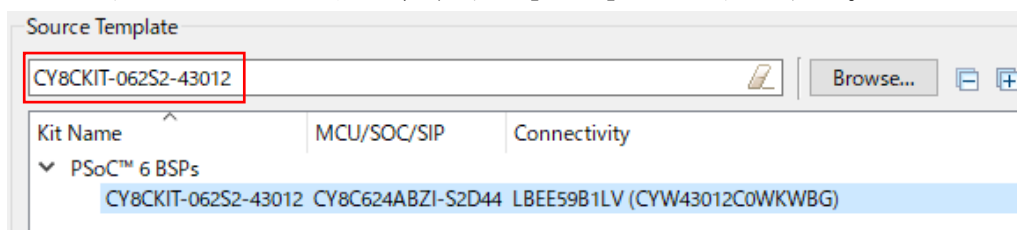


図 5 ターゲットボードを選択

(6) Application(s) Root Path を C:/mtb/uTK3 に変更する。

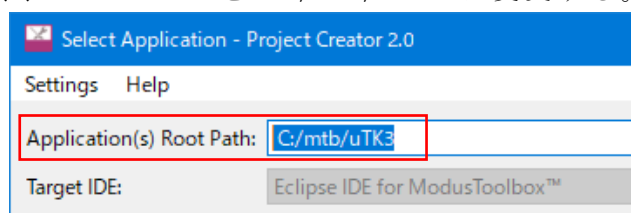


図 6 Application(s) Root Path を変更

アプリケーションプロジェクトを格納するためのディレクトリを指定する。

上図では例として C:/mtb/mtb_workspace を C:/mtb/uTK3 に変更している。他のディレクトリを指定しても構わないが、本書では C:/mtb/uTK3 に μT-Kernel 3.0 用のアプリを集める前提で説明する。

- (7) ベース用のアプリケーションとして、Template Application に含まれている [Getting Started] から [Hello World] を選択し、右下の [Create] をクリックする。

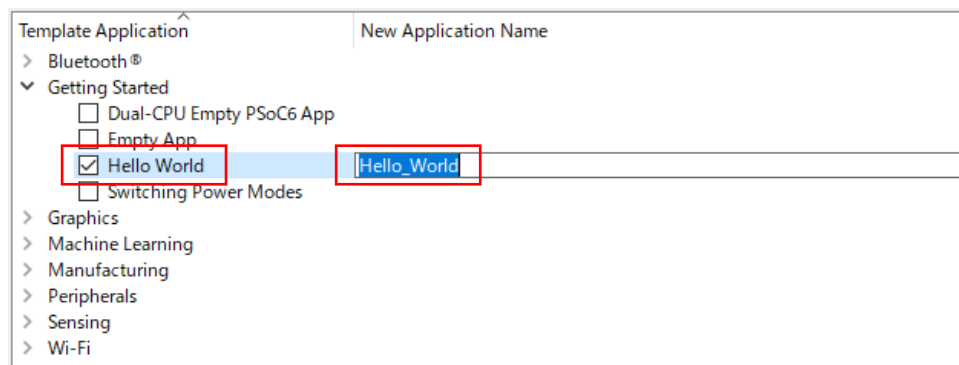


図 7 プロジェクトとして Hello World を選択

必要に応じて、New Application Name に表示されているアプリケーション名を変更する。ここでは Hello_World のまま変更していない。

- (8) 「ディレクトリがないので作るか？」と聞かれるので [Yes] を選択する。

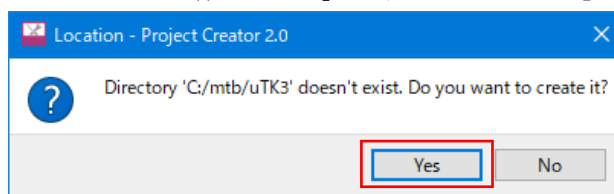


図 8 ディレクトリは未生成なので [Yes] を選択して生成

- (9) プロジェクトの生成に成功すると README.md が表示される。

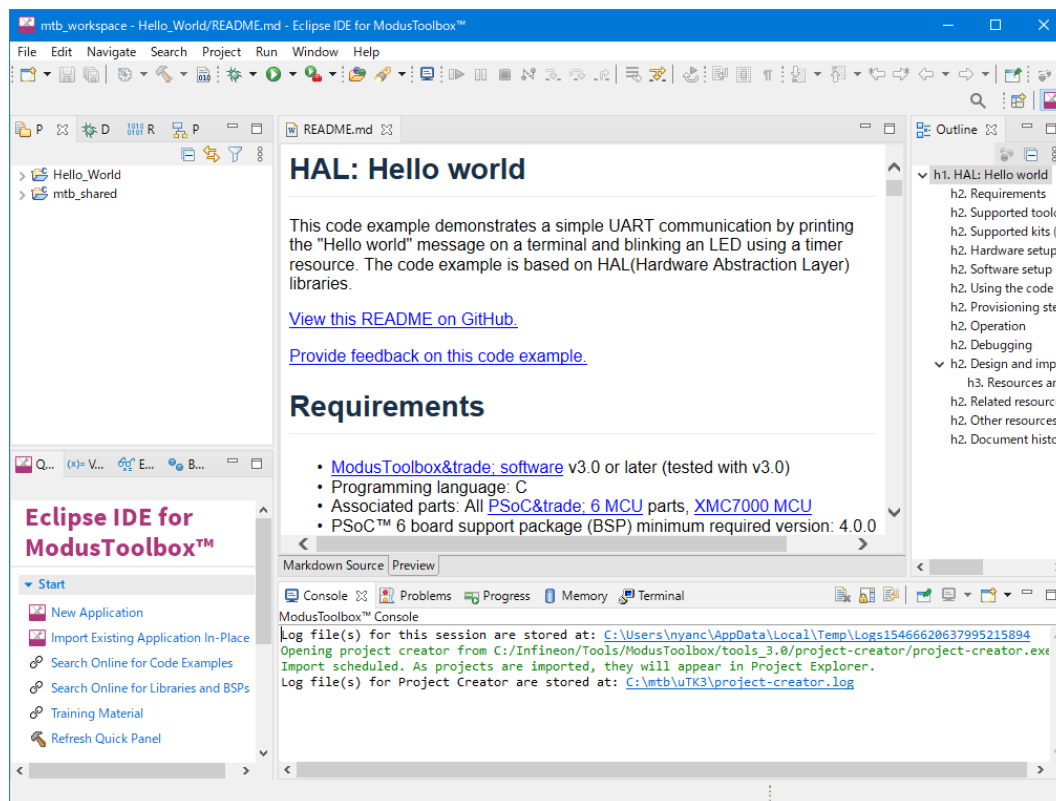


図 9 プロジェクト Hello_World の生成に成功

ここまでの操作によって以下のディレクトリが生成される。

```

C:\mtb¥
├── mtb_workspace¥
├── uTK3¥
│   ├── mtb_shared¥
│   ├── Hello_World¥
│   └── project-creator.log

```

Eclipse 用の設定ファイルなど

ModusToolbox が提供するモジュール
アプリケーションプロジェクト

続いて、サンプルプロジェクトのビルドとターゲットボードでの動作を確認する。

(10) Project Explorer で[Hello_World]を選択する。

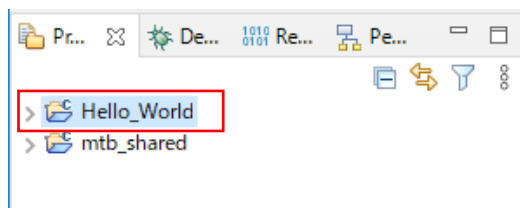


図 10 生成したプロジェクト Hello_World を選択

① [Quick Panel]の表示は選択したプロジェクトに応じて表示が変化する。

(11) [Quick Panel]の[Hello_World(CY8CKIT-062S2-43012)]から[Build Application]を選択してビルドを開始する。

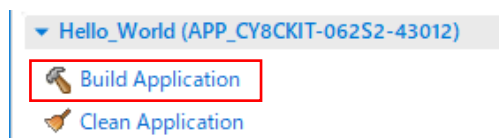


図 11 [Build Application]を選択してビルドを開始

(12) ビルド状況は Console に表示される。

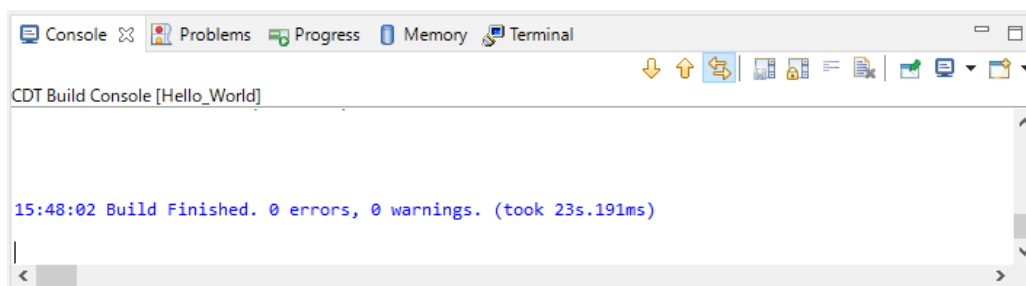


図 12 ビルド完了

(13) ターゲットボードの J6 と PC を付属の USB ケーブルで接続する。



図 13 コンソール用 UART (SCB5) の接続先 (J6)

ターゲットボードを接続した状態でデバイス マネージャーを起動し、[KitProg3 USB-UART (COMx)]が追加されていることを確認する。

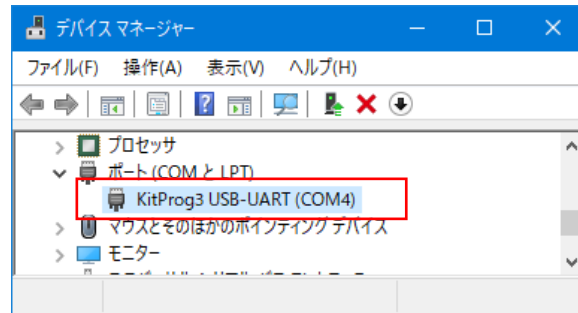


図 14 デバイスマネージャーで KitProg3 USB-UART (COMx) の追加を確認

① COM ポートの番号(x)は PC の状態によって異なる(上図では 4)。

(14) [Quick Panel]の[Launches]から[Hello World Debug (KitProg3_MiniProg4)]を選択して評価ボードにプログラムを書き込む。

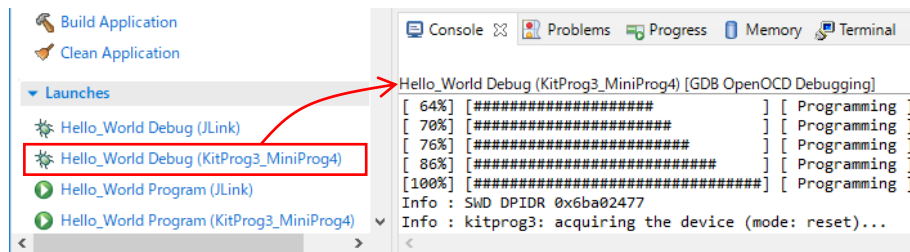


図 15 評価ボードへの書き込み中

(15) 書き込みが完了すると自動的にデバッグモードに切り替わり、プログラムが実行されて main 関数の先頭で停止する。

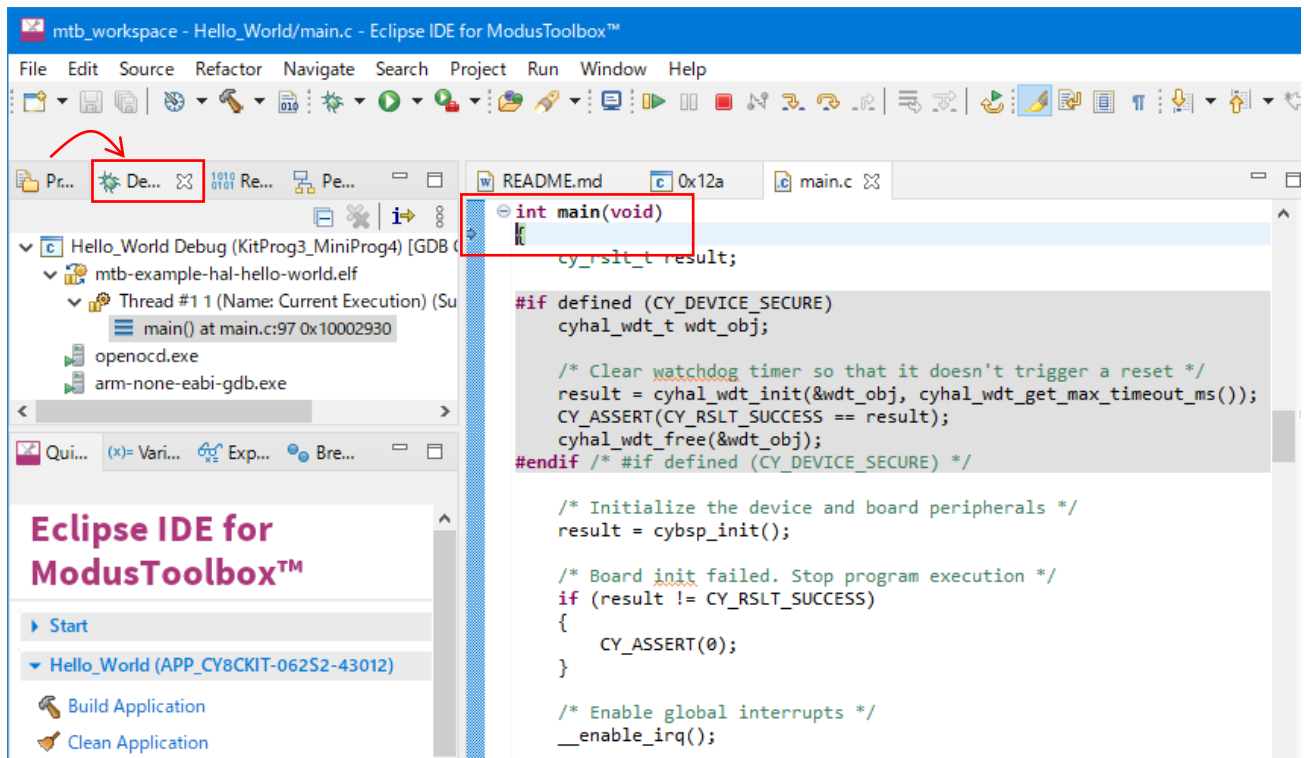


図 16 main 関数の先頭で停止した状態

(16) 通信ソフトを起動する。

本書では Tera Term を用いているが、特に限定はされていない。

通信パラメータは以下のとおりである。

通信速度	115,200 bps
データ長	8 bits
パリティビット	なし
ストップビット長	1 bit

(17) [Resume (F8)] で実行する。

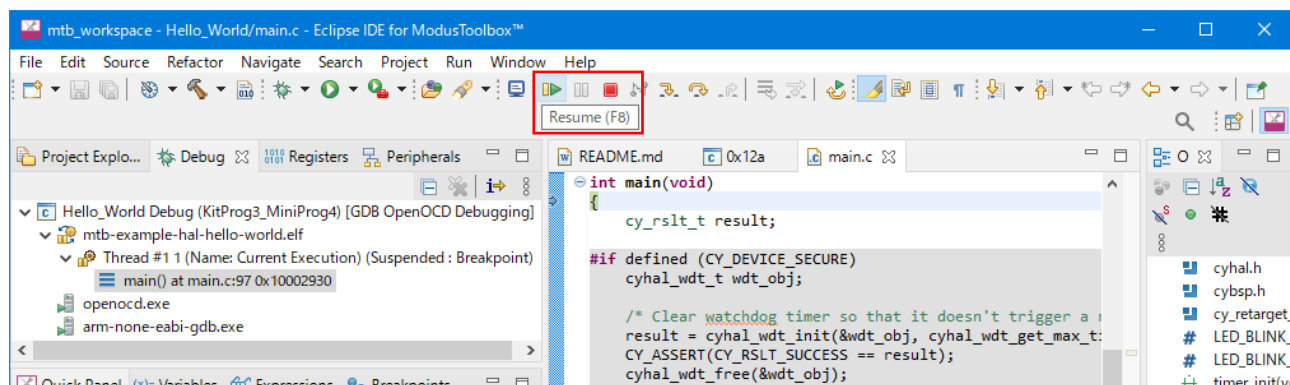


図 17 [Resume (F8)] で実行

(18) 正常に動作すると通信ソフトの画面にメッセージが表示される。

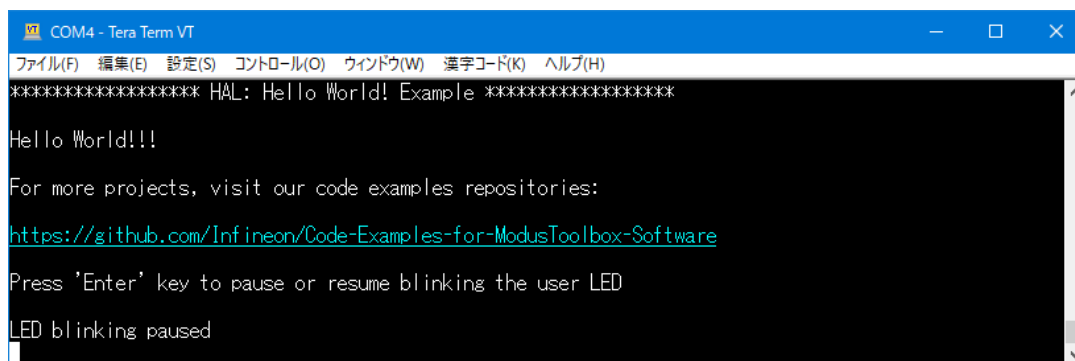


図 18 プロジェクト Hello_World でのメッセージ表示

また、評価ボード上の LED8 が点滅する。

この状態で通信ソフトから [Enter] を入力すると、LED8 の点滅を停止／再開できる。この時、コンソールには LED8 の点滅状態に合わせて "LED blinking paused" または "LED blinking resumed" と表示される。

- ① LED8 の点滅が停止／再開しない場合は、通信ソフトの設定で [改行] の送信が [CR](0x0D) になっているかを確認する。[LF](0x0A) では制御できない。

(19) 実行しているアプリケーションを終了し、ModusToolbox を終了する。

- ① アプリケーションの終了手順に関しては「4.6. ビルド、実行」を参照のこと。

4. プロジェクトの作成

ModusToolbox を利用した μ T-Kernel 3.0 BSP のプロジェクトの構築手順は以下のとおりである。

- (1) ModusToolbox でサンプルアプリケーションを生成
- (2) μ T-Kernel 3.0 を Git のサブモジュールとして追加
- (3) サブモジュールのブランチを CY8CKIT-062S2-43012 対応版に変更
- (4) プログラムのベースを μ T-Kernel 3.0 に変更
- (5) Device Configurator でコンソール用に SCB5 を設定
- (6) ビルド、実行

以下、各作業の詳細について説明する。

4.1. ModusToolbox でサンプルアプリケーションを生成

「3.2. ModusToolbox の動作確認」で説明した手順に従って ModusToolbox のサンプルアプリケーションを生成する。

4.2. μ T-Kernel 3.0 を Git のサブモジュールとして追加

以下の例では、TortoiseGit を用いてサブモジュールを追加する手順を説明する。
利用ツールなどによる差異に関しては適宜読み替えること。

- (1) サンプルアプリケーションを Git に登録する。
① サブモジュールは Git の機能なので、サンプルアプリケーション(3.2.では Hello_World として生成)を一旦 Git に登録する必要がある。
- (2) UCT が配布する μ T-Kernel 3.0 の BSP のサイトにアクセスする。
https://github.com/UCTechnology/mtk3_bsp
- (3) [<> Code▼]を開いて Clone 用の URL をクリップボードにコピーする。

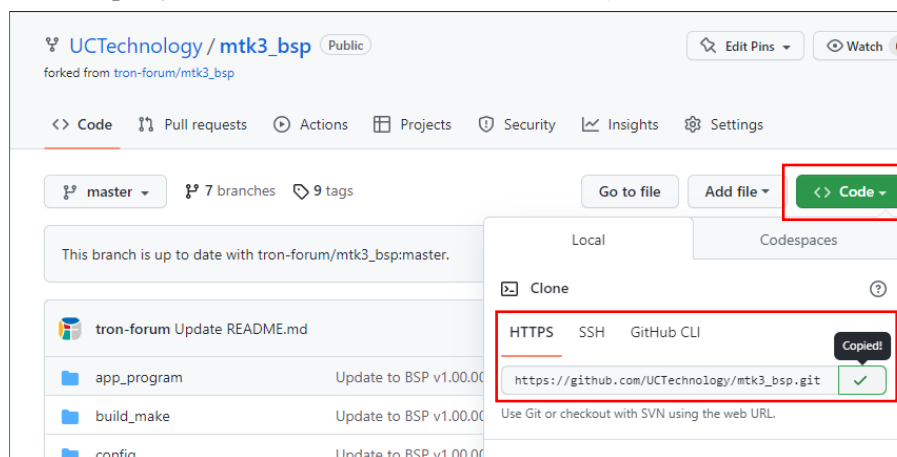


図 19 Clone 用の URL をクリップボードにコピー

- (4) サンプルアプリケーションのフォルダをエクスプローラで開く。
- (5) 右クリックメニューの[TortoiseGit]から[サブモジュールの追加]を選択する。

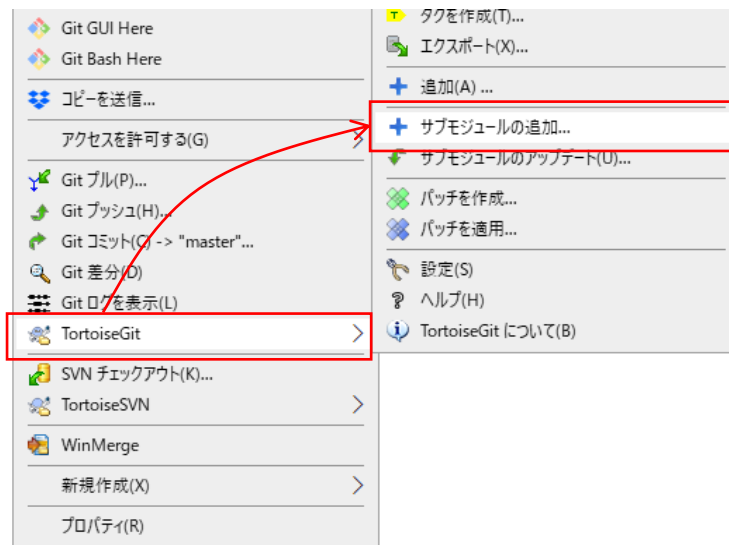


図 20 サブモジュール追加のメニュー操作

- (6) サブモジュールの追加ダイアログが表示される。
パスに `mtkernel_3` と入力し、[OK]をクリックする。

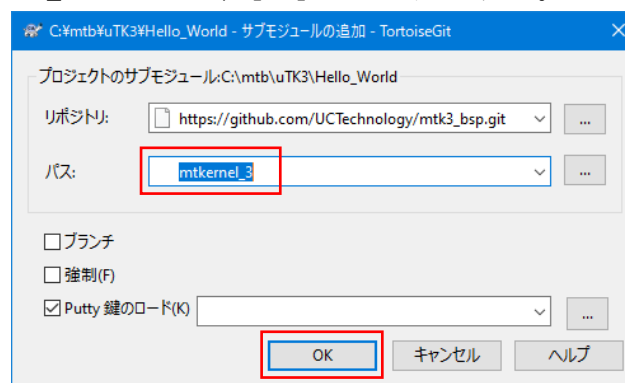
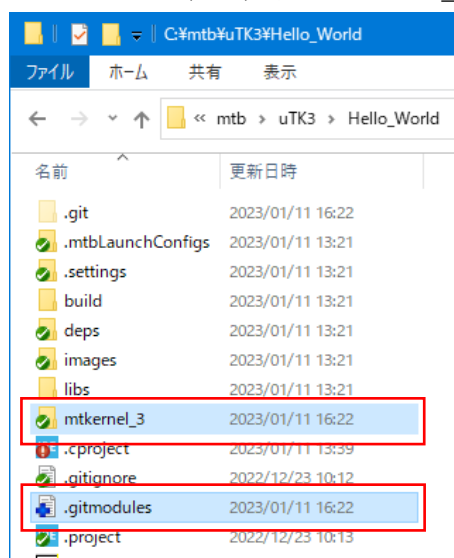


図 21 サブモジュールの追加設定

- (7) サンプルアプリケーションのフォルダに `mtkernel_3` と `.gitmodules` が追加される。

図 22 サブモジュール `mtkernel_3` 追加後の構成

(8) `.gitmodules` の内容を確認すると以下になっている。

```
[submodule "mtkernel_3"]
  path = mtkernel_3
  url = https://github.com/UCTechnology/mtk3_bsp.git
```

リスト 1 `.gitmodules` の内容

(9) この状態で一旦コミットしておく。

4.3. サブモジュールのブランチを CY8CKIT-062S2-43012 対応版に変更

(1) サブモジュールとして追加した `mtkernel_3` のフォルダを開く。

(2) 右クリックメニューの [TortoiseGit] から [切り替え/チェックアウト] を選択する。

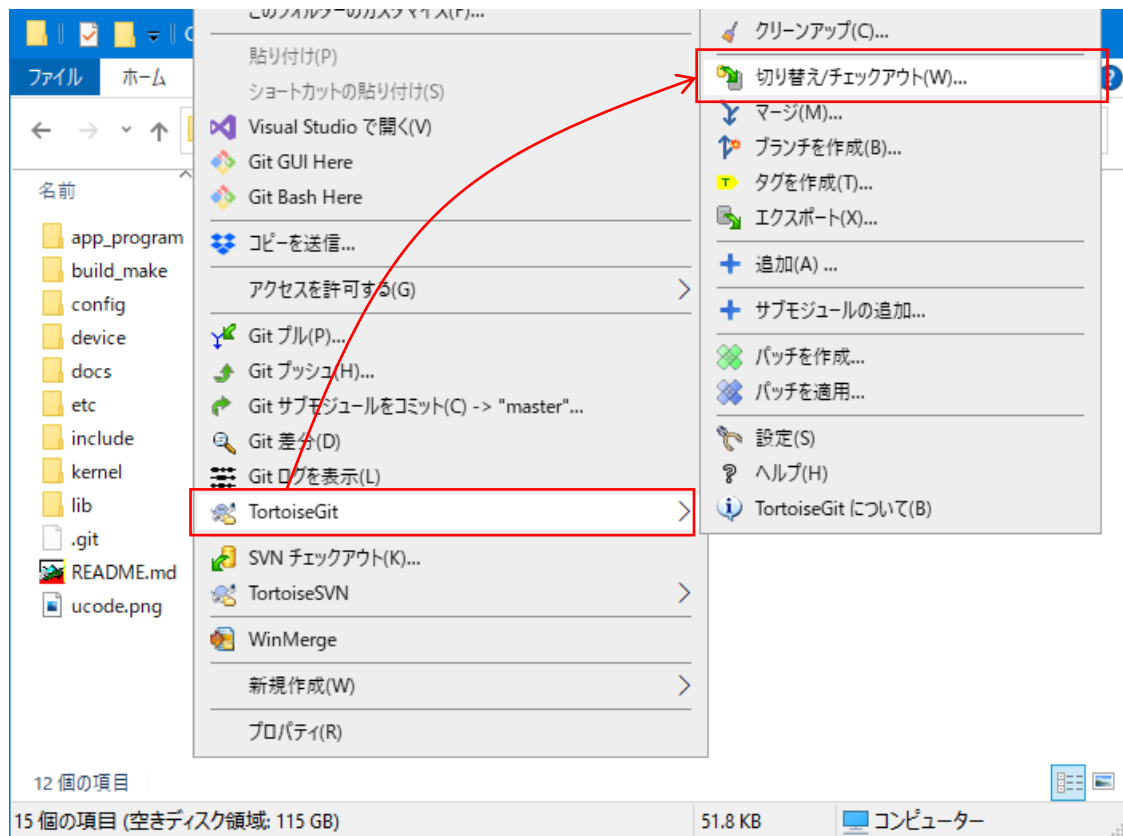


図 23 ブランチ切り替えのメニュー

(3) 切り替え/チェックアウトダイアログが表示される。

ブランチとして `remotes/origin/cy8ckit_cy8c6` を一覧の中から選択し、新しいブランチとして `cy8ckit_cy8c6` を作成することにして、[OK]をクリックする。

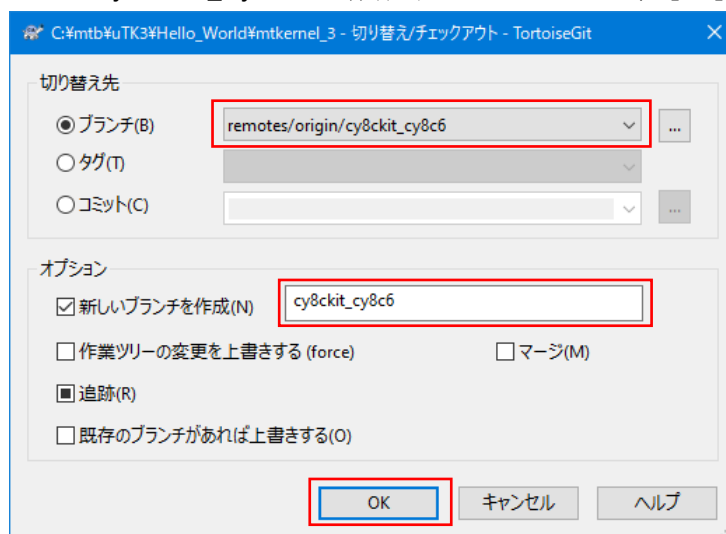


図 24 ブランチの切り替え

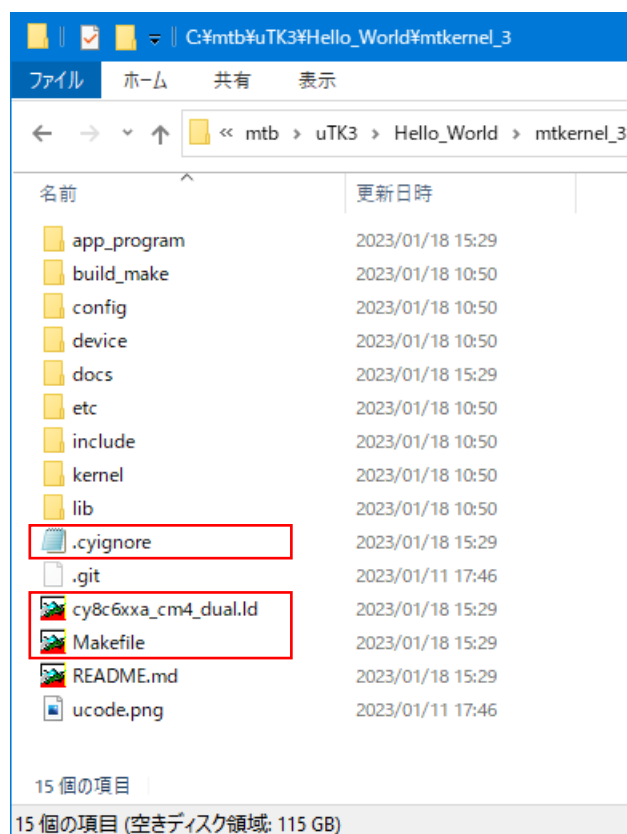


図 25 ブランチの切り替え完了

図 25 の表示を図 23 のフォルダの内容と比較すると、CY8CKIT-CY8C6 用のファイルやディレクトリが追加されていることが分かる。

この他に、`app_program/app_main.c` や `kernel/` 以下のソースコードなども追加・変更されている。

4.4. プログラムのベースを μ T-Kernel 3.0 に変更

μ T-Kernel 3.0 用のビルド情報を上位にコピーする。

具体的には、以下のディレクトリとフォルダを `Hello_World/mtkernel_3/` から上位の `Hello_World/` にコピーする。(コピーは Windows の GUI で操作すればよい。)

- ・ `app_program/` フォルダごとコピーする。
- ・ `.cyignore` 置き換え、またはマージ
- ・ `cy8c6xxa_cm4_dual.ld`
- ・ `Makefile` 置き換え、またはマージ

`app_program/` は、フォルダごとコピーする。

`.cyignore` と `Makefile` は、プロジェクトが `Hello_World` の場合は、既存のファイルに上書きしてよい。他のプロジェクトの場合は内容を確認してマージする必要がある。

- ❶ `.cyignore` には μ T-Kernel 3.0 の中でビルド対象から除外すべきディレクトリやファイルの一覧などが記載されている。

`Makefile` には CY8CKIT-CY8C6 版の μ T-Kernel 3.0 で必要となるマクロ定義とリンカスクリプトの指定が追加されている。

各ファイルの概要については「5. μ T-Kernel 3.0 のディレクトリ／ファイル構成」で説明する。

4.5. Device Configurator でコンソール用に SCB5 を設定

μ T-Kernel 3.0 では、標準でコンソール用に UART を 1 チャンネル使用する。

CY8CKIT-CY8C6 用の μ T-Kernel 3.0 では、SCB5 (Serial Communication Block 5) をコンソールとして使用する実装にしてある。そこで、ビルドする前に Device Configurator で SCB5 を設定する。

- (1) Eclipse IDE for ModusToolbox を起動し、Project Explorer で [Hello_World] を選択する。

この操作によりプロジェクト `Hello_World` 用のデバイスを設定する状態になる。

- ❶ プロジェクトが未選択の状態では Quick Panel のプロジェクト用の項目 ([Project], [Launches], [Tools], [BSP Configurations]) は空白のままで各機能を利用できない(項目が表示されていない)。Project Explorer でプロジェクトを選択することで必要な項目が表示されて利用可能になる。(図 36 参照)

- (2) Quick Panel の [BSP Configurations (APP_CY8CKIT-S62S2-43012)] から [Device Configurator] を選択して起動する。



図 26 Device Configurator を選択

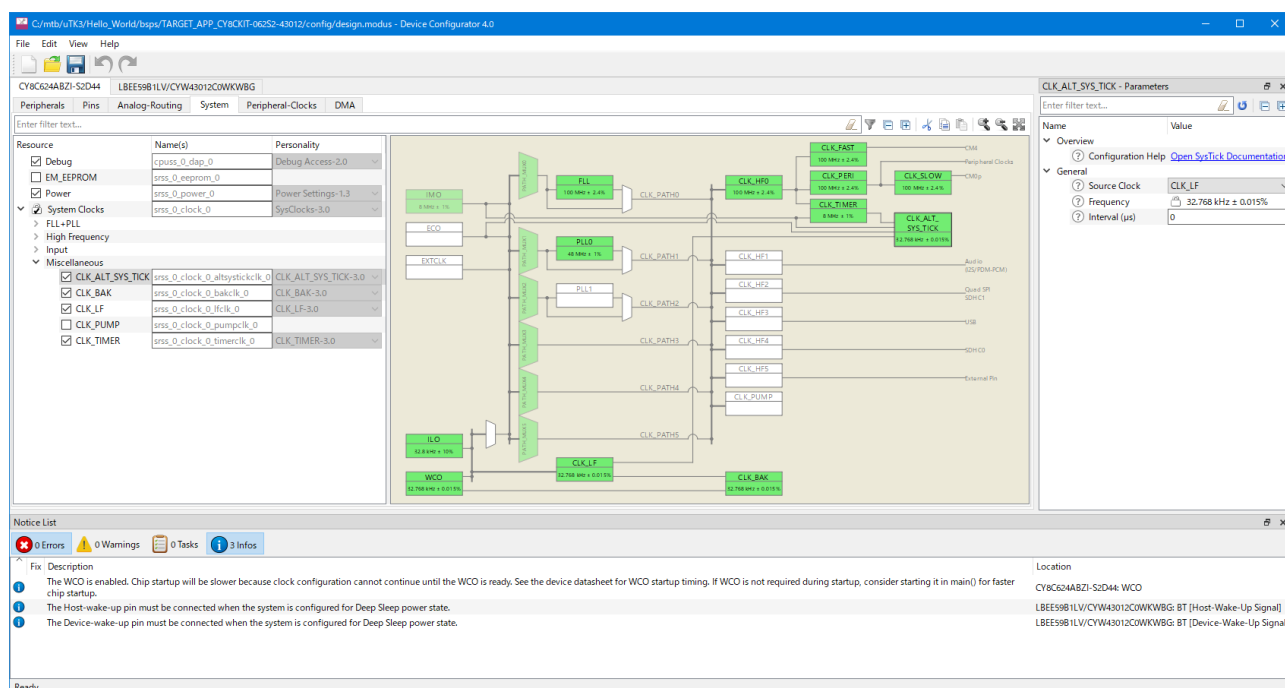


図 27 Hello_World 用の Device Configurator が起動

- (3) [CY8C624ABZI-S2D44]の[Peripherals]を選択する。

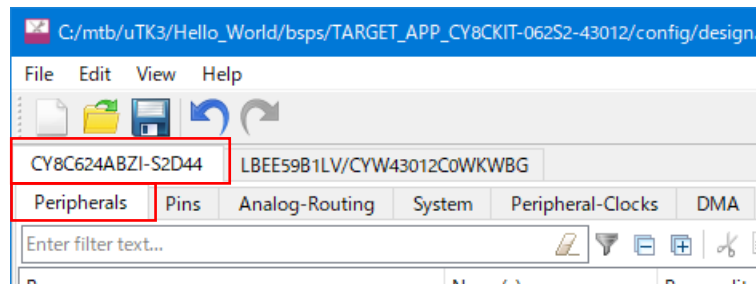


図 28 [Peripherals]を選択

- (4) Resource 中の [Communication] の [Serial Communication Block (SCB) 5] をチェックし、Personality として [UART-3.0] を選択する。

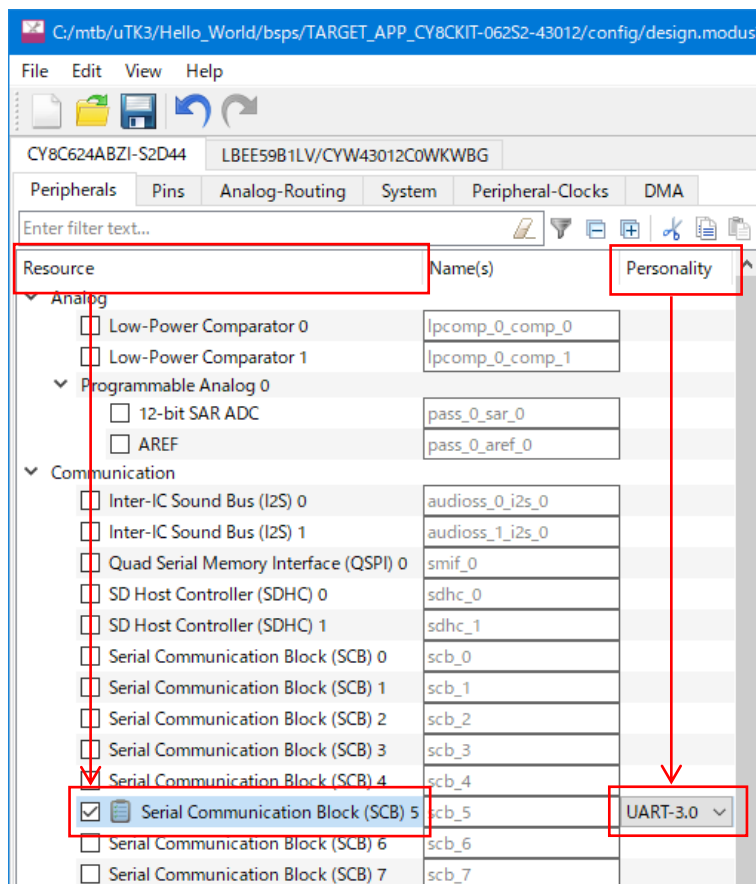


図 29 SCB5 を有効化し、UART の機能を割り当て

(5) 右のペインに **SCB5** 用のパラメータが表示されるので詳細な項目を設定する。

Name	Value
Configuration Help	Open UART (SCB) Documentation
Com Mode	Standard
Baud Rate (bps)	115200
Oversample	8
Bit Order	LSB First
Data Width	8 bits
Parity	None
Stop Bits	1 bit
Enable Digital Filter	<input type="checkbox"/>
Support RS-485	<input type="checkbox"/>
TX-Enable	<input type="checkbox"/>
Enable Flow Control	<input type="checkbox"/>
CTS Polarity	Active Low
RTS Polarity	Active Low
RTS Activation Level	63
Clock	16 bit Divider 0 clk [USED]
RX	P5[0] digital_inout (CYBSP_DEBUG_UART_RX, CYBSP_D0) [USED]
TX	P5[1] digital_inout (CYBSP_DEBUG_UART_TX, CYBSP_D1) [USED]
RX Trigger Output	<unassigned>
TX Trigger Output	<unassigned>
Actual Baud Rate (bps)	114678
Baud Rate Accuracy (%)	0.453
Clock Frequency	917.431 kHz
RX FIFO Level	63
TX FIFO Level	63
Enable Multi Processor Mode	<input type="checkbox"/>
Address	0
Mask	255
Accept Matching Address in RX FIFO	<input type="checkbox"/>
Drop on Frame Error	<input type="checkbox"/>
Drop on Parity Error	<input type="checkbox"/>
Break Signal Bits	11
Store Config in Flash	<input checked="" type="checkbox"/>
API Mode	High Level

図 30 SCB5 用のパラメータ設定

ここでは少なくとも **Clock** のソースと **RX/TX** のピンを選択する必要がある。コンソールはターゲットボードの **USB** コネクタ(J6)を利用して **PC** に接続するので、**RX = P5[0]**と **TX = P5[1]**の設定にする必要がある。

Baud Rate なども設定可能だが、**CY8CKIT-CY8C6** 用の **μT-Kernel 3.0** の実装ではデフォルト設定(上図参照)のままでよい。

- (6) [CY8C624ABZI-S2D44] の [Peripheral-Clocks] を選択する。

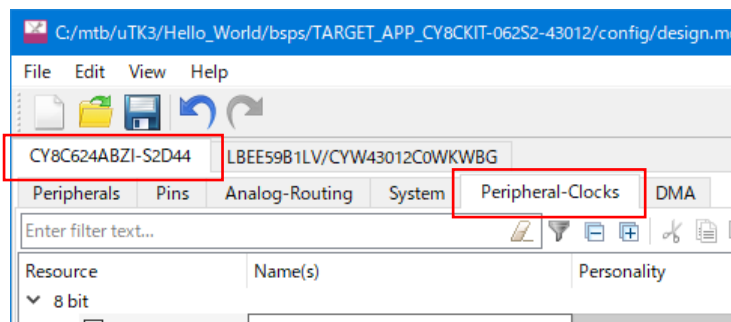


図 31 [Peripheral-Clocks] を選択

- (7) Resource の中の [16 bit] から [16 bit Divider 0] をチェックし、Name に SCB5_CLK_DIV と入力する。

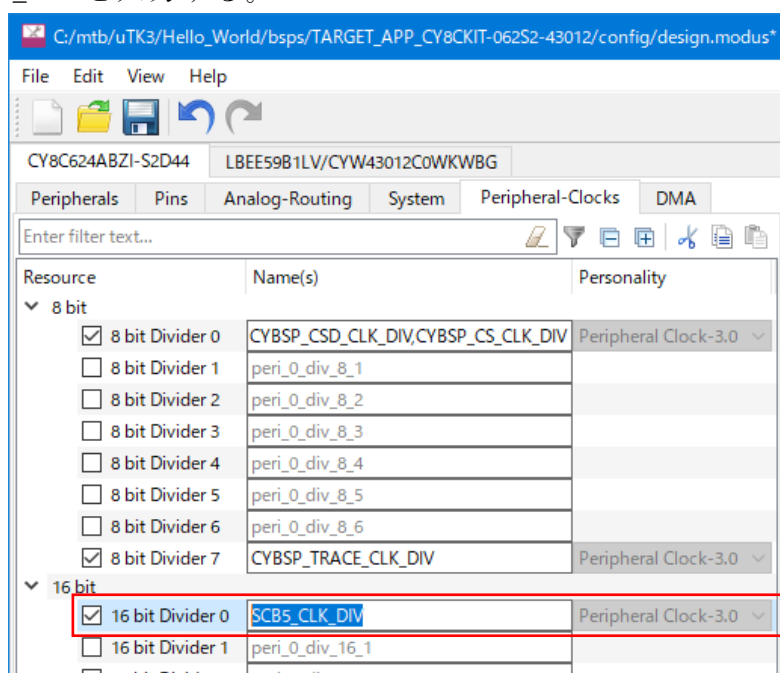


図 32 [16 bit Divider 0] を有効化して名前を付ける

- (8) 右のペインに 16 bit Divider 0 用のパラメータが表示される。

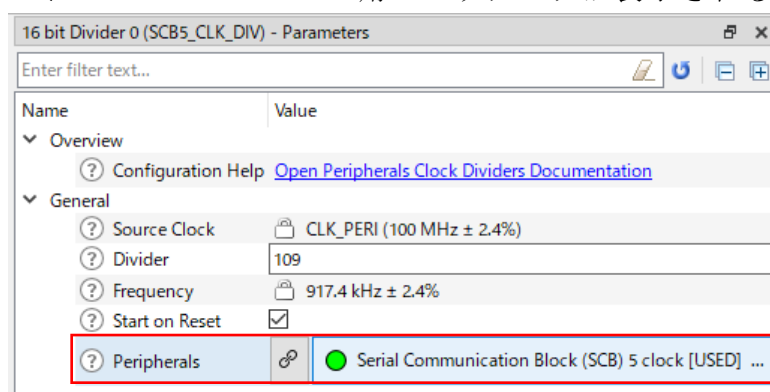


図 33 [Peripherals] で SCB5 を選択

- ① SCB5 の設定が完了していれば上図のように自動的に設定されている。
また、名称(SCB5_CLK_DIV) も自動的に SCB5 の設定に反映される。

(9) [CY8C624ABZI-S2D44] の[System]を選択する。

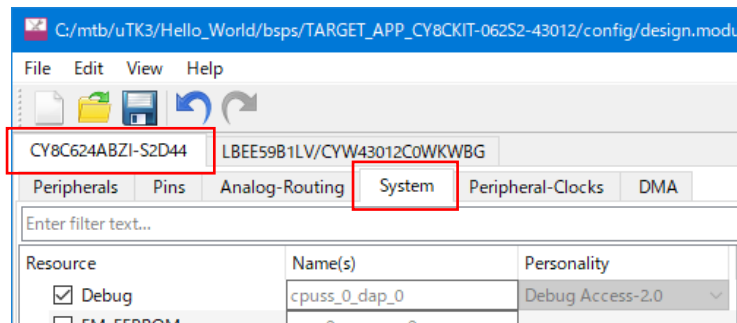


図 34 [System]を選択

(10) Resource の[System Clocks]の下に[Miscellaneous]の[CLK_ALT_SYS_TICK]のチェックを外して無効化する。

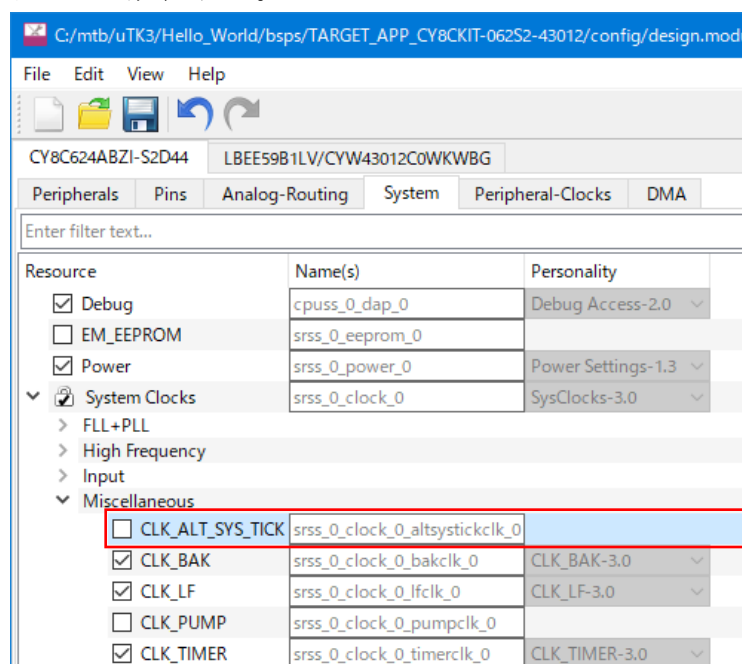


図 35 [CLK_ALT_SYS_TICK]を無効化

- ① [CLK_ALT_SYS_TICK]が有効の設定だと ModusToolbox が用意しているデフォルトのティックタイマが動作し、μT-Kernel 3.0 のティックタイマが停止する。
μT-Kernel 3.0 には専用のティックタイマを実装してあるので ModusToolbox のティックタイマを無効化する必要がある。

(11) 全ての設定が完了したら、設定を保存して Device Configurator を終了する。

Device Configurator は、プロジェクト Hello_World の以下にあるソースコードなどを変更する。

libs/TARGET_CY8CKIT-062S2-43012/COMPONENT_BSP_DESIGN_MODUS/GeneratedSource/

この設定でビルドすることにより、SCB5 が μT-Kernel 3.0 のコンソールとして機能する状態になる。

4.6. ビルド、実行

前節までの操作で必要な設定が完了したので、μT-Kernel 3.0 版の Hello_World プロジェクトをビルドする。

(1) Project Explorer で Hello_World を選択する。

この操作により Quick Panel でのビルド、実行などの操作が可能となる。

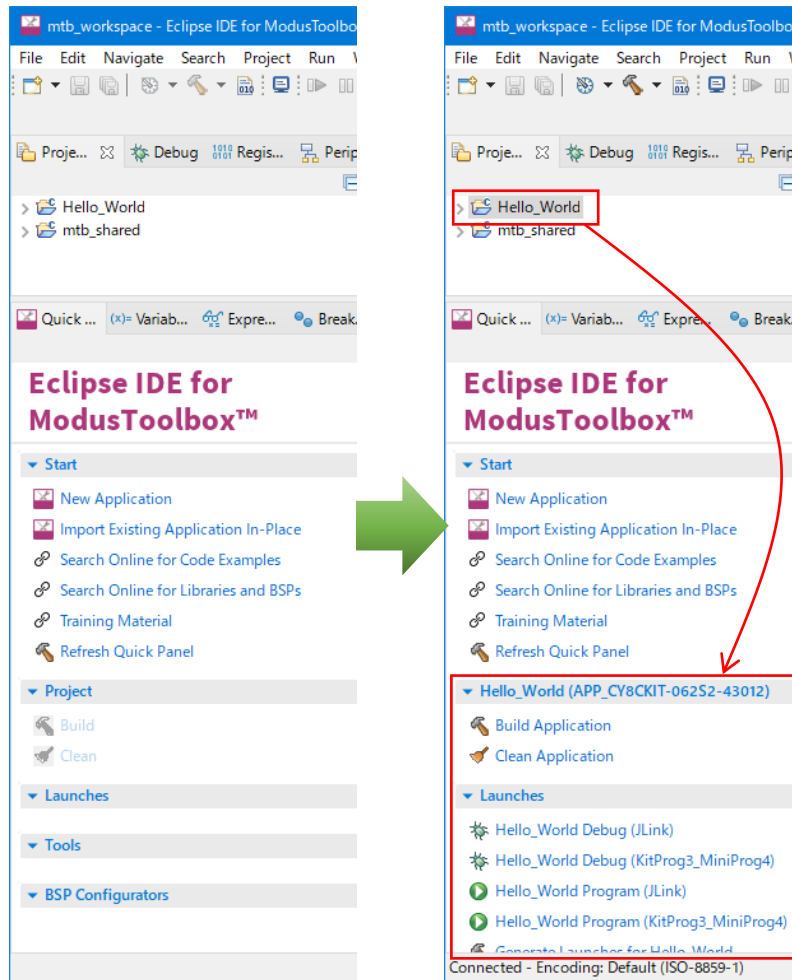


図 36 プロジェクトを選択するとビルドや実行が可能となる。

- (2) Quick Panel の [Build Hello_World Application] を選択するとビルドが開始される。

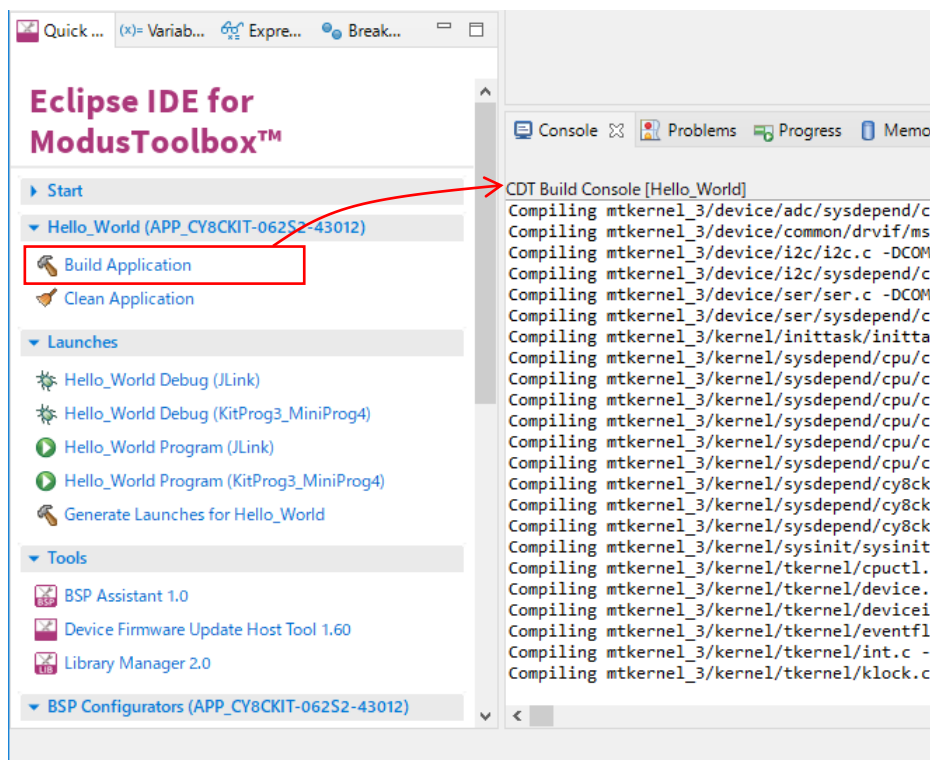


図 37 ビルド開始

- (3) ビルドが完了すると Console に **Build Finished. 0 errors, 0 warnings.** と表示される。

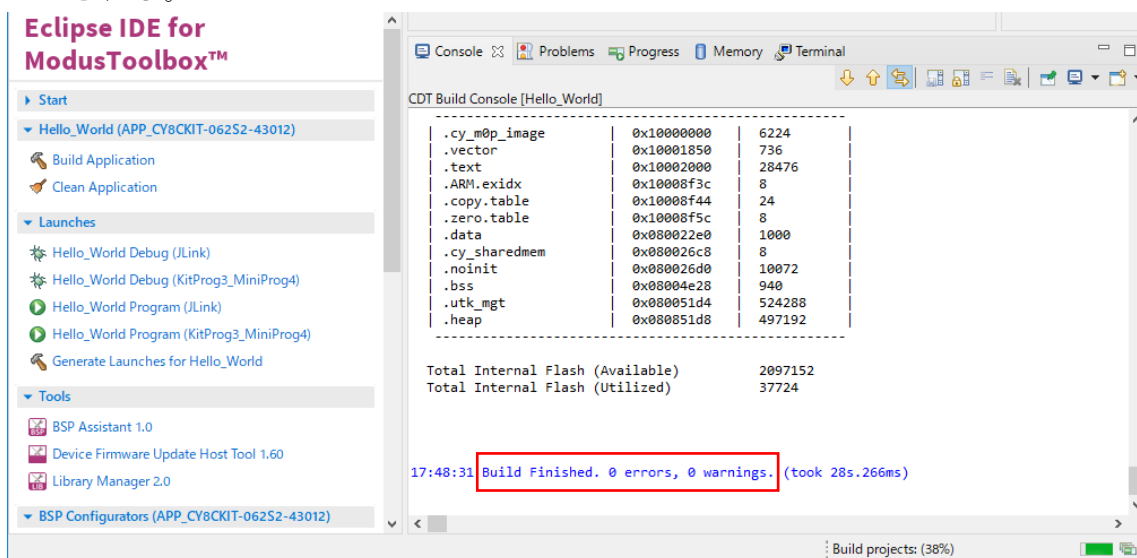


図 38 ビルド完了

- (4) [Hello_World Debug (KitProg3_MiniProg4)]を選択してデバッグ実行する。

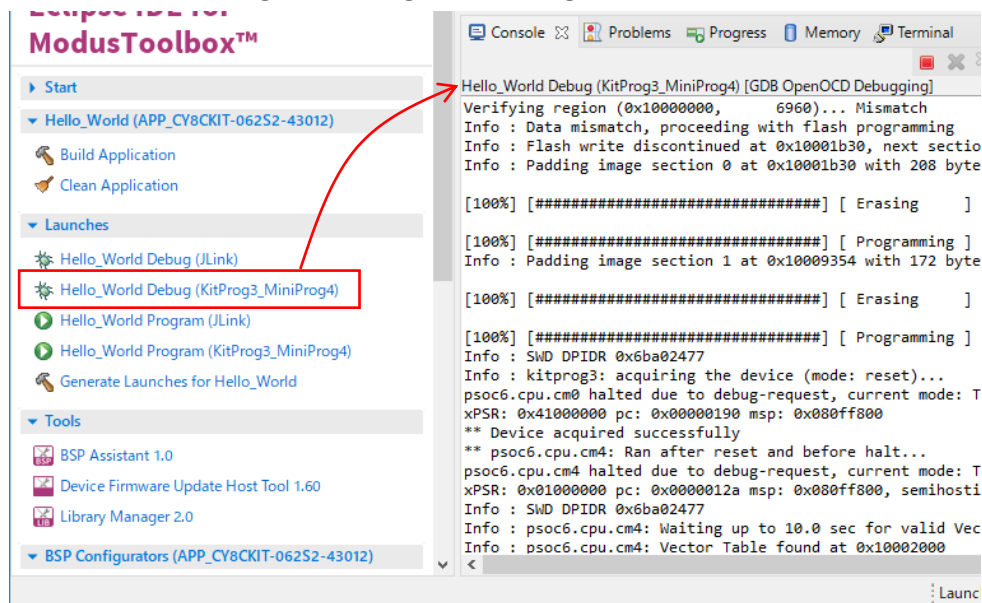


図 39 デバッグ実行

- (5) ターゲットボードへの書き込みが完了すると main 関数の先頭で停止する。

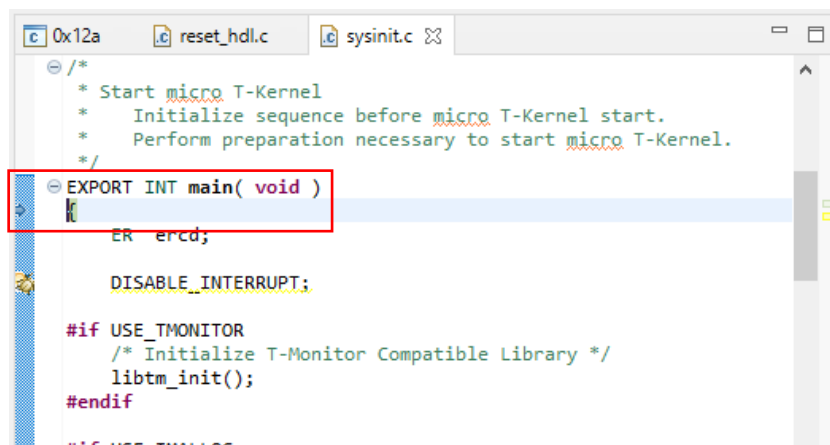


図 40 main 関数の先頭で停止

- ① μT-Kernel 3.0 の main 関数は sysinit.c に含まれている。

- (6) μT-Kernel 3.0 のコンソール用に「3.2. ModusToolbox の動作確認」で使用した通信ソフトを起動する。
- (7) [Resume (F8)]で実行すると、通信ソフトにメッセージが表示される。

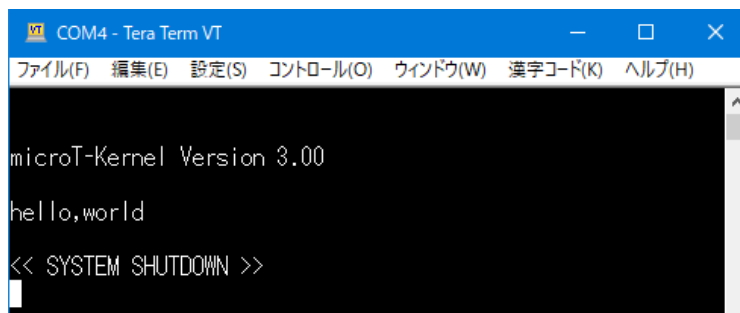


図 41 μT-Kernel 3.0 からコンソールに表示されたメッセージ

(8) 実行したプログラムは、[Terminate]で終了する。

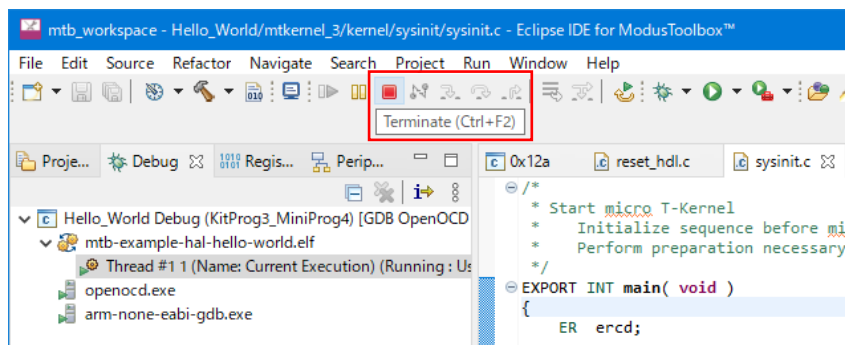


図 42 プログラムを[Terminate]で終了

終了したプログラムは[Remove All Terminated Launches]で削除する。

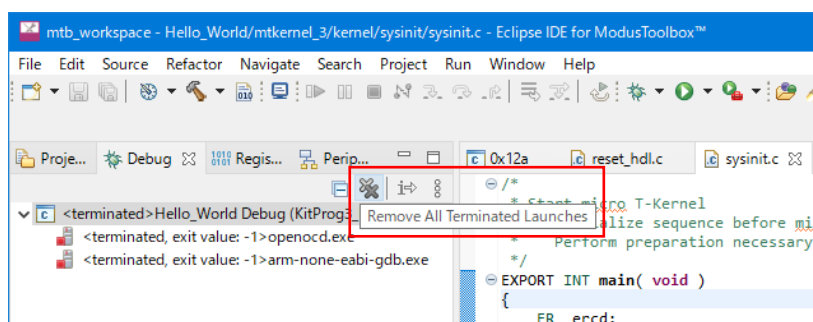


図 43 終了したプログラムは[Remove~]で削除

続いてデバッグのコンフィギュレーションを調整する。

デフォルトでは Non-OS の設定になっているので、main 関数の先頭で停止する(図 40)。これを μT-Kernel 3.0 に合わせて初期タスクから呼び出される usermain 関数の先頭で停止するように変更する。

(9) [Breakpoints]を選択して現在の設定を確認し、[Remove All Breakpoints]で全てのブレークポイントを削除する。

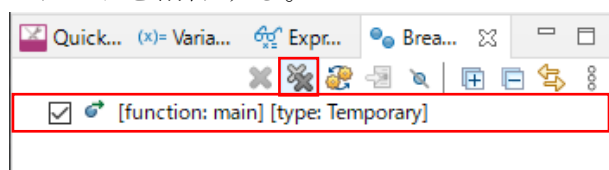


図 44 設定されているブレークポイント

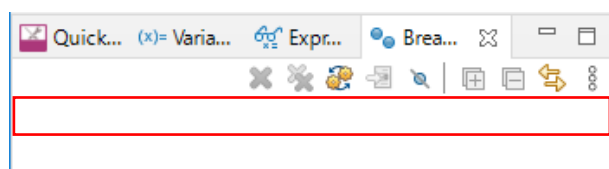


図 45 全てのブレークポイントを削除

① [Breakpoints]のタブが見当たらない場合は、メニューの[Window]の中から[Show View]→[Breakpoints]と選択して表示する。

- (10) メニューの[(虫マークの横の▼)]でプルダウンメニューを表示し、その中から [Debug Configurations]を選択して Debug Configurations を開く。

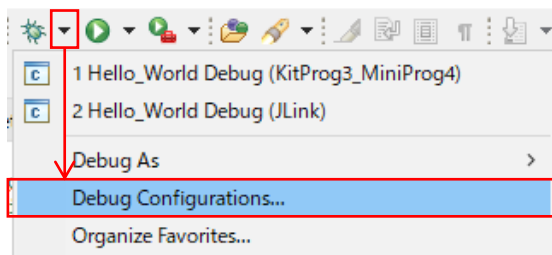


図 46 Debug Configurations を選択

- ① をクリックすると前回使用したデバッグ構成でデバッグが開始される。
上図のメニューを開くためにはその横にある▼をクリックする必要がある。

- (11) 左ペインの中から [GDB OpenOCD Debugging] の中の [Hello_World Debug (KitProg3_MiniProg4)]を選択する。

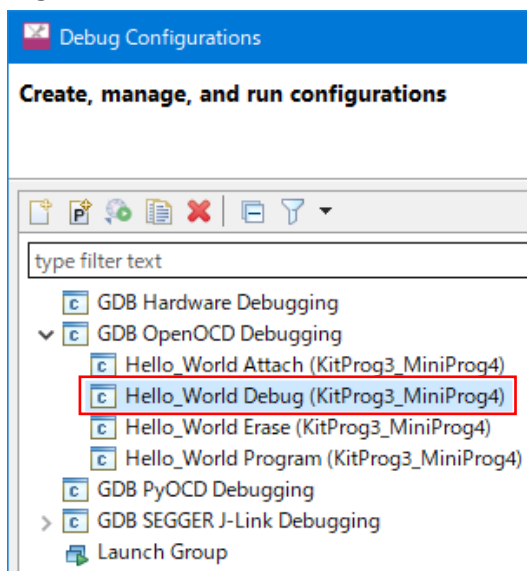


図 47 [Hello_World Debug (KitProg3_MiniProg4)]を選択

- (12) [Startup]タブを選択し、その中の Set breakpoint at:に usermain と入力する。

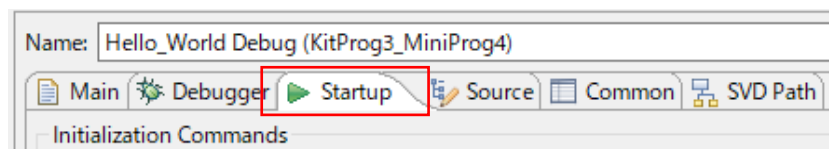


図 48 [Startup]タブを選択

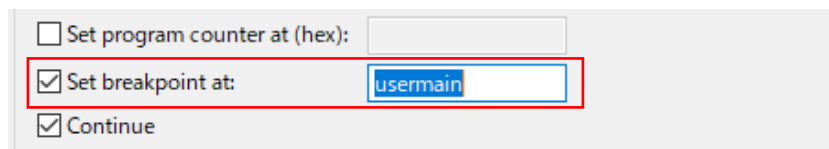


図 49 プログラム実行後のブレークポイントとして usermain 関数を指定

(13) 設定が完了したら右下の [Apply]→[Close] で終了する。

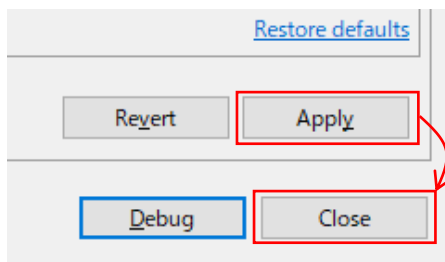


図 50 [Apply]→[Close]で設定を保存して終了

(14) [Hello_World Debug (KitProg3_MiniProg4)] を選択してデバッグ実行すると usermain 関数の先頭で停止する。

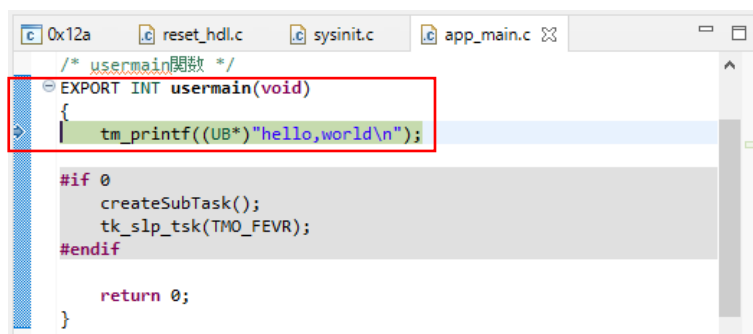


図 51 usermain 関数の先頭で停止

(15) [Resume (F8)] で実行すると、通信ソフトにメッセージが表示される。

(図 41 参照)

以上で μT-Kernel 3.0 の初期タスクから呼び出される usermain 関数の実行まで実行できたことになる。

CY8CKIT-CY8C6用のサンプルコードではサブタスクを起動するサンプルを実装してある。図 51 の #if 0 を #if 1 に変更してビルド、実行するとコンソールに 1 秒毎に経過秒数を表示するサンプルが実行される。

詳細については app_program/app_main.c のコードを参照されたい。(「5.1. アプリケーションプログラムの追加」参照)

- ④ サブタスクが経過秒数を表示するタイミングは正確には 1 秒ではなく 1 秒 + α の時間がかかる。これはサンプルコードの実装と μT-Kernel 3.0 の仕様／実装によるものである。

5. μ T-Kernel 3.0 のディレクトリ／ファイル構成

CY8CKIT-CY8C6 版の μ T-Kernel 3.0 のディレクトリおよびファイルの構成は、μ T-Kernel 3.0 の正式リリース版に準じて以下のように構成してある。

表 5-1 プロジェクトのファイル構成

ディレクトリ名 またはファイル名	内容
app_program/	μ T-Kernel 3.0 用アプリケーションのディレクトリ
config/	コンフィギュレーション
device/	デバイスドライバ
docs/	ドキュメント
include/	各種定義ファイル
kernel/	μ T-Kernel 3.0 本体
lib/	ライブラリ
build_make/	Make 構築ディレクトリ(本実装では使用しない)
etc/	リンカファイル等(本実装では使用しない)
.cyignore	本実装用のビルド対象外ファイルの指定
Makefile	本実装用の Make ファイル
cy8c6xxa_cm4_dual.ld	本実装用のリンクスクリプト
README.md	本実装の ReadMe ファイル
ucode.png	μ T-Kernel 3.0 の ucode
.git	Git 用のファイル
.gitignore	Git 用のファイル

「4.4. プログラムのベースを μ T-Kernel 3.0 に変更」で説明した通り、上記のディレクトリとファイルのうち、下記を上位のディレクトリ(プロジェクトのルートディレクトリ)にコピーして利用する。

- ・ **app_program/** フォルダごとコピーする。
- ・ **.cyignore** 置き換え、またはマージ
- ・ **Makefile** 置き換え、またはマージ
- ・ **cy8c6xxa_cm4_dual.ld**

5.1. アプリケーションプログラムの追加

μ T-Kernel 3.0 用のアプリケーションプログラムはプロジェクトのルートディレクトリにコピーした **app_program/**に追加する。サブモジュールの下にある **app_program/**は変更しない。

つまり、以下の(B)ではなく、(A)を使用することになる。

(A) <プロジェクトのルート>/app_program/ ←アプリケーションはこちらに追加

(B) <プロジェクトのルート>/mtkernel_3/app_program/

❗ (B)は.cyignoreによってビルド対象から除外されている。

初期状態ではサンプルの `usermain` 関数を含む `app_main.c` のみが用意されている。アプリケーション用のコードは `app_main.c` に直接追加するか、または `app_program/` の下にソースコードのファイルを追加することで実装する。`app_program/` 以下にサブディレクトリを作成しても構わない。

追加したファイルは ModusToolbox がビルド時に自動的に検出してビルド対象に加える。

❗ C 言語のソースコード(*.c)をプロジェクトのディレクトリ(サブディレクトリを含む)に追加した場合、ModusToolbox が自動的に(強制的に)にビルド対象として登録するので注意が必要である。

追加したソースコードをビルド対象から除外する場合は、追加したファイル名(プロジェクトのルートからの相対パス)を.cyignore に記載する必要がある。

ModusToolbox は.cyignore に記載されているファイル(ディレクトリの場合はそれ以下の全てのファイル)をビルド対象外として無視(ignore)する。

CY8CKIT-CY8C6 版の μT-Kernel 3.0 では初期タスクとは別のタスク(subTask)を生成して起動するコードを追加してあるので実装時の参考にされたい。

具体的には以下の太字の部分**が追加したサンプルコード**である。

```
#include <tk/tkernel.h>
#include <tm/tmonitor.h>

IMPORT void    createSubTask(void);

/* usermain 関数 */
EXPORT INT usermain(void)
{
    tm_printf((UB*)"hello,world\n");

#if 0                                ←ここを1にするとサブタスクが起動する。
    createSubTask();
    tk_slp_tsk(TMO_FEVR);
#endif

    return 0;
}

/*-----*/
/* Additional program */
/*-----*/
#define MSG(f,...)    do{ tm_printf((UB*)f,##__VA_ARGS__); }while(0)
```

```

LOCAL void subTask(void)
{
    W cnt = 0;
    while(1){
        tk_dly_tsk(1*1000);
        MSG( "¥r%4d 秒経過", cnt++ );
    }
}

const T_CTSK ctsk_subTask = {
    0, /* exinf */
    TA_HLNG|TA_RNG0, /* tskatr */
    (FP)&subTask, /* task */
    1, /* itskpri */
    10*1024, /* stksz */
#ifdef USE_OBJECT_NAME
    "SubTask", /* dsname[8] */
#endif
    NULL, /* bufptr */
};

LOCAL ID tid; /* Sub Task ID */
EXPORT void createSubTask(void)
{
    ER ercd;

    tid = tk_cre_tsk( (CONST T_CTSK*)&ctsk_subTask );
    MSG( "tk_cre_tsk() = %d¥n", tid );
    ercd = tk_sta_tsk( tid, 0 );
    MSG( "tk_sta_tsk() = %d¥n", ercd );
}

```

リスト 2 CY8CKIT-CY8C6 版の app_main.c

- ❶ 上記のサンプルコードでは、タスク subTask が起動されてからの経過秒数をコンソールに表示しているだけである。ただし秒数を表示するタイミングは正確には 1 秒ではなく 1 秒 + α の時間がかかる。
- ❷ 簡便なサンプルとするためにエラー処理などは含めていない。

5.2. ビルド用のファイル

Makefile、cy8c6xxa_cm4_dual.ld、.cyignore は ModusToolbox でのビルドに利用する。このため、プロジェクトのルートディレクトリにコピーした状態で利用することになる。

Makefile には CY8CKIT-CY8C6 版の μT-Kernel 3.0 で必要となるマクロ定義とリンクスクリプトである cy8c6xxa_cm4_dual.ld の指定を追加してある(以下の**太字部分**を追加)。

```

:
DEFINES=_CY8CKIT_CY8C6_
:
LINKER_SCRIPT=cy8c6xxa_cm4_dual.ld
:

```

リスト 3 Makefile の相違点

◆ **Makefile** と **.cyignore** は **ModusToolbox Version 3.0.0** が生成するサンプルプロジェクト **Hello_World** に合わせた実装になっている。他のサンプルプロジェクトを利用する場合は、**ModusToolbox** が生成するファイルとマージする必要がある。

❗ **cy8c6xxa_cm4_dual.ld** は他のサンプルプロジェクトでも原則としてそのまま利用可能だが、必要に応じて調整が必要となる場合がある。

5.3. μ T-Kernel 3.0 のソースコード

config/、**device/**、**docs/**、**include/**、**kernel/**、**lib/** の各ディレクトリには μ T-Kernel 3.0 のソースコードが含まれる。

サンプルプログラム程度であれば特に変更する必要はないが、タスクやセマフォの最大数などの調整が必要な場合は **config/**以下のファイルで調整することになる。

❗ μ T-Kernel 3.0 のコンフィギュレーションについては「μ T-Kernel 3.0 共通実装仕様書」を参照のこと

build_make/、**etc/**は **gcc** を用いて開発する場合に利用する。本実装では **ModusToolbox** を利用するので、これらのディレクトリは利用しない。

μT-Kernel 3.0
構築手順書 (CY8CKIT-CY8C6)
Rev 3.00.06 (January, 2023)

ユーシーテクノロジー株式会社
141-0031 東京都品川区西五反田 2-12-3 第一誠実ビル 9F
©2022-2023 Ubiquitous Computing Technology Corporation All Rights Reserved.