

# $\mu$ T-Kernel 3.0 構築手順書 (EVK-XMC7200)

**Rev 3.00.01**

December, 2023



**$\mu$ 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 .....	6
3.1. ModusToolbox のインストール .....	6
3.2. ModusToolbox の動作確認 .....	6
3.3. μT-Kernel 3.0 のビルド、実行 .....	9
3.4. プログラムの終了手順 .....	15
4. サンプルアプリケーション .....	16
4.1. μT-Kernel 3.0 の標準的なサンプルアプリケーション .....	16
4.1.1. サンプルアプリケーションの実行 .....	16
4.1.2. サンプルアプリケーションの内容 .....	16
4.2. シリアルドライバ用のサンプルアプリケーション .....	17
4.2.1. EVK-XMC7200 のシリアルポート .....	17
4.2.2. シリアルドライバ用のサンプルアプリケーションを実行 .....	17
4.2.3. 実行結果 .....	18
4.2.4. サンプルアプリケーションの内容 .....	19
5. μT-Kernel 3.0 のディレクトリ／ファイル構成 .....	20
5.1. μT-Kernel 3.0 のソースコード .....	21
5.2. その他の μT-Kernel 3.0 関連のディレクトリとファイル .....	21
5.3. ビルド用のファイル .....	21
5.4. その他のファイル .....	22
5.5. アプリケーションプログラムの追加 .....	22
5.5.1. アプリケーションプログラムの追加ディレクトリ .....	22
5.5.2. アプリケーションプログラムの実装例 .....	23

## 1. はじめに

本品は、ユーシーテクノロジー(株)が開発した μ T-Kernel 3.0 である。



本品は、トロント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 KIT\_XMC72\_EVK(以下、EVK-XMC7200)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、EVK-XMC7200 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 EVK-XMC7200 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、μ T-Kernel 3.0 BSP (Board Support Package)の使用方法について説明する。

μ T-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 μ T-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(開発環境は含まない)。

### 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	XMC7200D-E272K8384 (ARM Cortex-M7, Cortex-M0+)	Infineon Technologies AG
OS	μ T-Kernel 3.00.06	トロンフォーラム
実機 (マイコンボード)	KIT_XMC72_EVK (EVK-XMC7200)	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](https://github.com/UCTechnology/mtk3_bsp)

- ❷ μ T-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](https://github.com/tron-forum/mtkernel_3)

- ❸ 対象マイコンボード(EVK-XMC7200)に関しては Infineon Technologies AG のサイトを参照のこと。

[https://www.infineon.com/cms/jp/product/evaluation-boards/kit\\_xmc72\\_evk/](https://www.infineon.com/cms/jp/product/evaluation-boards/kit_xmc72_evk/)

### 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 3.1.0	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	serd	SCB3	P13[0:3]	ARD_P13[0:3]

❗ シリアル通信ドライバ(UART)のサンプルプログラム用のプロジェクト作成方法については「4.2. シリアルドライバ用のサンプル」を参照のこと。

⚠ コネクタ ARD\_P13[0:3]は IO ピン P13[0:3]とは接続されていない。  
このため、コネクタを利用する場合は予め接続(0Ω 抵抗を実装)する必要がある。

## 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	μT-Kernel 3.0 仕様書(Ver. 3.00.00)	トロンフォーラム
	μT-Kernel 3.0 共通実装仕様書(Ver.1.00.8)	トロンフォーラム
	μT-Kernel 3.0 共通実装&構成仕様書(Rev 3.00.06)	ユーシーテクノロジー(株)
T-Monitor	T-Monitor 仕様書(Ver.1.00.01)	トロンフォーラム
デバイスドライバ	μT-Kernel 3.0 デバイスドライバ説明書(Ver.1.00.06)	トロンフォーラム
	μT-Kernel 3.0 共通デバイスドライバ説明書(Rev 3.00.05)	ユーシーテクノロジー(株)
実装仕様書	μT-Kernel 3.0 実装仕様書(EVK-XMC7200)	ユーシーテクノロジー(株)
構築手順書	μT-Kernel 3.0 構築手順書(EVK-XMC7200)	ユーシーテクノロジー(株)
ターゲットボード	取扱説明書 KIT_XMC72_EVK XMC7200 evaluation kit guide	Infineon Technologies AG
	ユーザーガイド KitProg3 user guide	
	回路図 KIT_XMC72_EVK_XMC7200-evaluation-kit_schematic-PCBDesignData	
搭載マイコン	データシート XMC7000 microcontroller	
	テクニカルリファレンスマニュアル XMC7000 MCU family architecture Technical reference manual	

分類	名称	発行
	テクニカルリファレンスマニュアル XMC7200 MCU registers Technical reference manual	
	ModusToolbox ユーザーガイド ModusToolbox user guide	

- 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](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](https://github.com/UCTechnology/mtk3_bsp)

### 3. 開発環境の準備

μ T-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。
- ◆ 2023/11/22 時点での ModusToolbox の最新バージョンは、Version 3.1.0 である。  
本資料では移植作業に使用した Version 3.1.0 を基に説明する。

#### 3.2. ModusToolbox の動作確認

以下では、ModusToolbox を使用して μ T-Kernel をビルドし、動作を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。

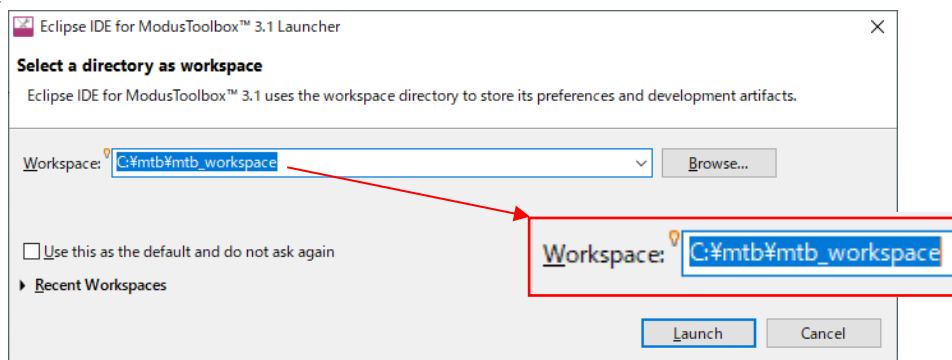


図 1 workspace の指定

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。

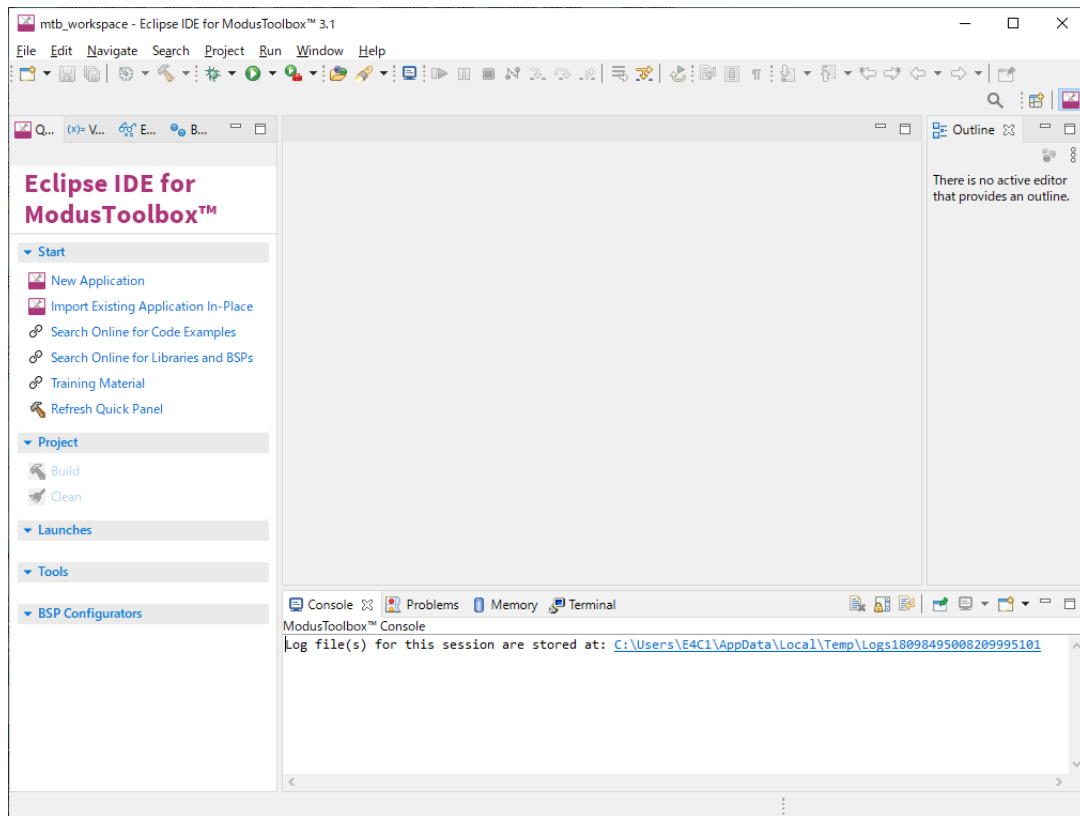


図 2 Eclipse IDE for ModusToolbox 起動

(3) [Quick Panel]の[Start]から[New Application]を選択し、Project Creator を起動する。

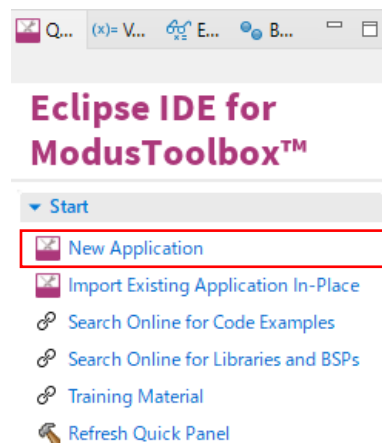


図 3 Quick Panel 画面



## (4) Project Creator が起動する。

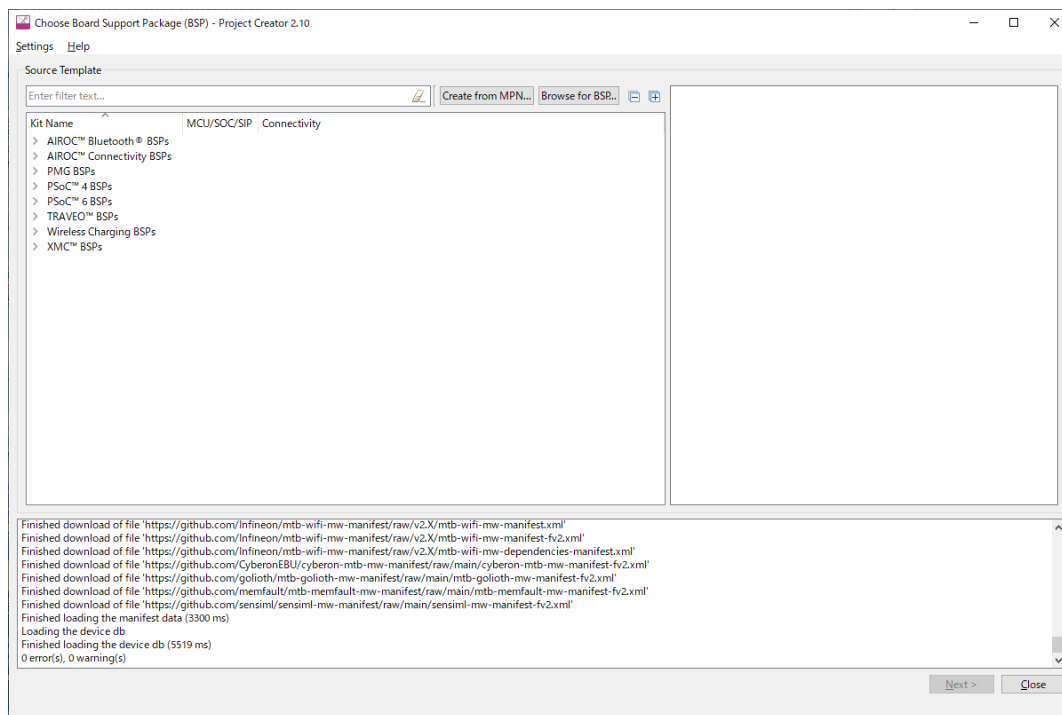


図 4 Project Creator 起動画面

- (5) ボードを選択するため赤枠で囲んだ検索窓に KIT\_XMC72\_EVK と入力して、表示されたターゲットボードを選択し、[Next]をクリックする。

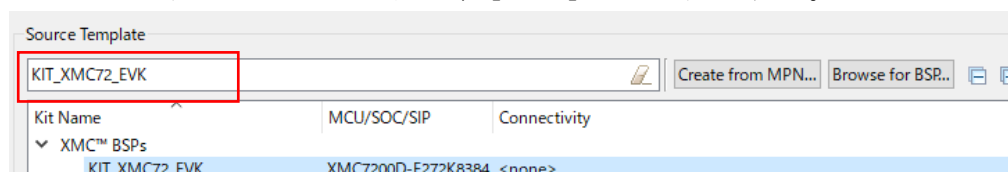


図 5 ターゲットボード選択画面

- (6) 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。

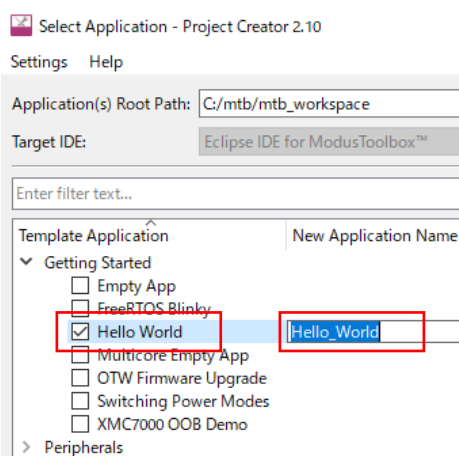


図 6 Template Application 選択画面

(7) プロジェクトの生成に成功すると README.md が表示される。

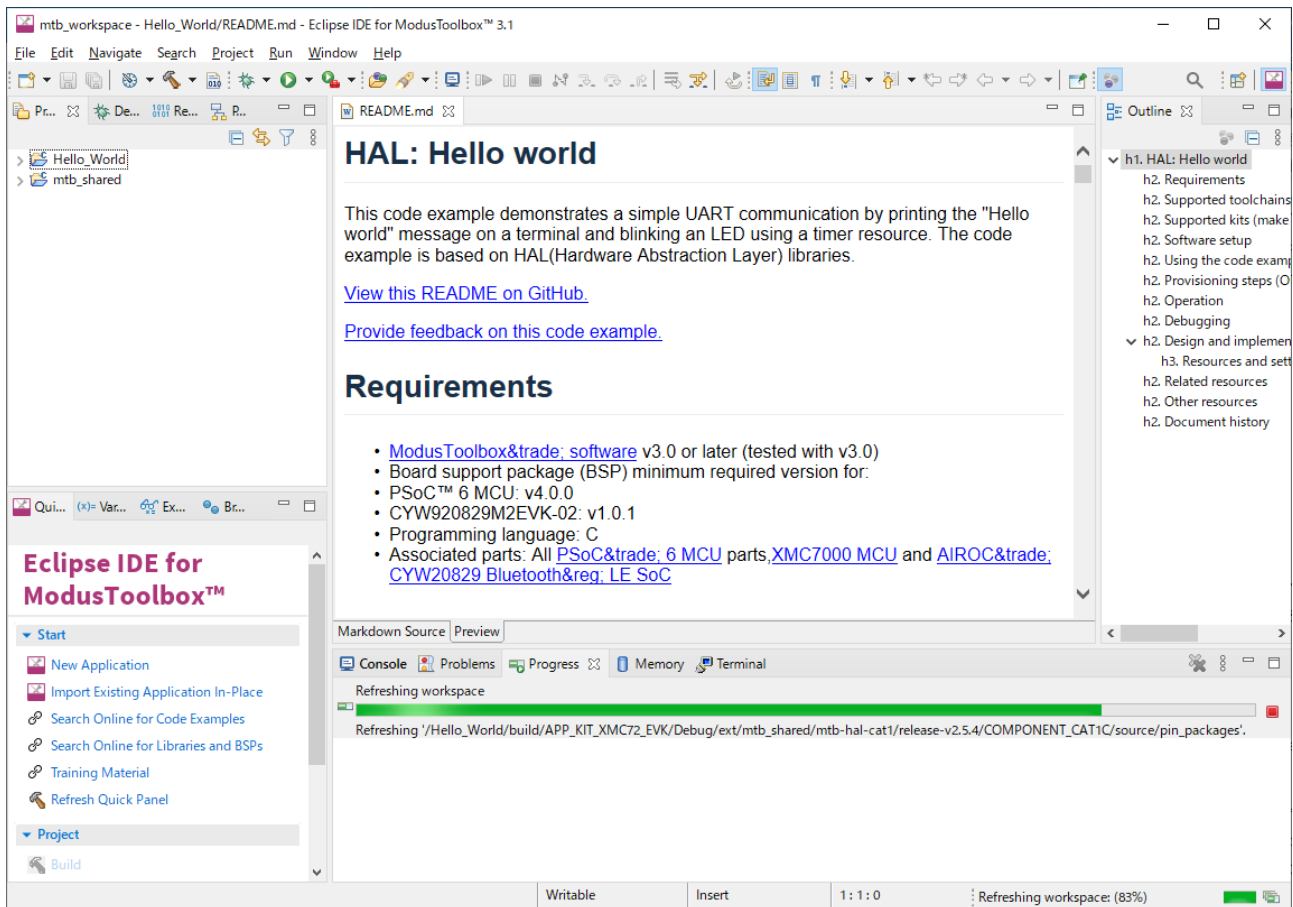


図 7 README.md 表示画面

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\
│   ├── 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

### 3.3. μ T-Kernel 3.0 のビルド、実行

μ T-Kernel のプロジェクトをインポートする。

(1) μ T-Kernel 3.0 のソースコードを、UCT の GitHub(下記)からクローンする。

[https://github.com/UCTechnology/mtk3\\_bsp](https://github.com/UCTechnology/mtk3_bsp)

❗ クローンするローカルのディレクトリは任意で構わない。

(2) クローンしてきたディレクトリにおいて、ブランチを `master` から `evk_xms7200` に切り替える。

- (3) ワークスペースに μ T-Kernel 3.0 用のディレクトリを用意する。

```
C:\mtb¥
├─ mtb_workspace¥
│   ├── mtb_shared¥
│   ├── Hello_World¥
│   ├── mtkernel_3¥
│   └─ 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  
μ T-Kernel 3.0 用のディレクトリを追加

- (4) μ T-Kernel 3.0 のソースコード (EVK-XMC7200 用) を上記の `mtkernel_3` にコピーする。(.git/ はコピーしなくてよい。)

① ディレクトリの生成、ソースコードのコピーは、Windows の通常の操作で実施してよい。もしくは、`mtkernel_3` に直接クローンしてもよい。

- (5) メニューの [File] から [Import] を選択する。

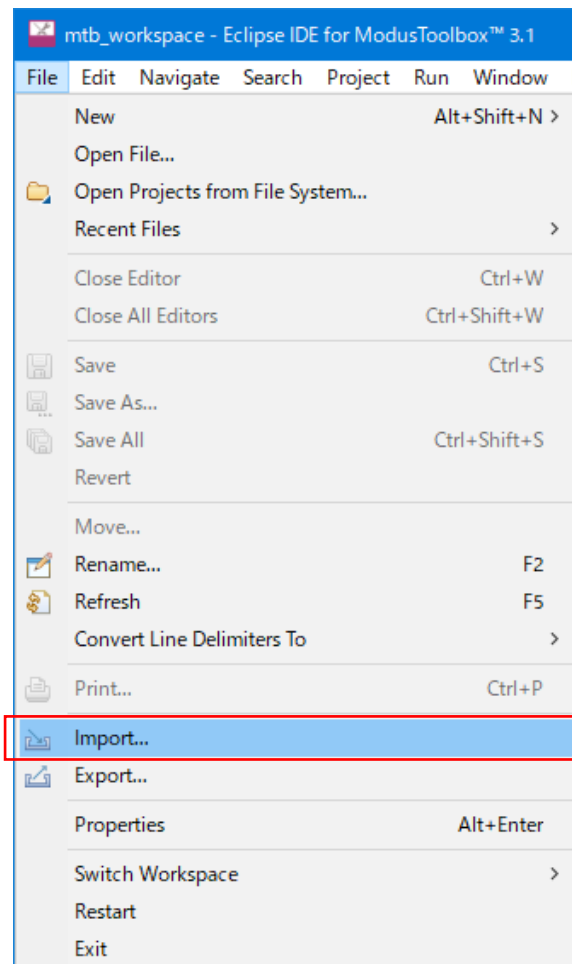


図 8 [Import] を選択

(6) [General] 中の [Existing Projects into Workspace] を選択する。

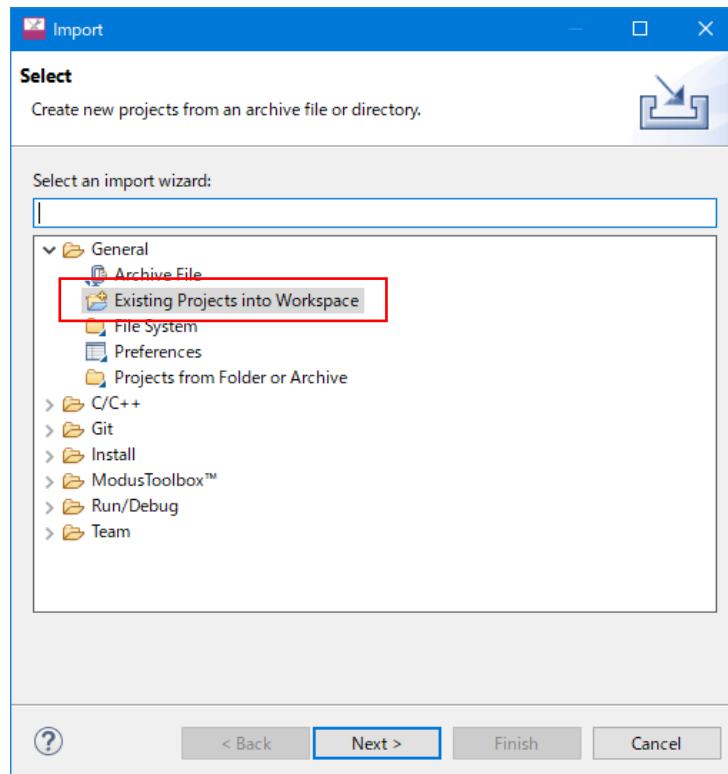


図 9 [Existing Projects into Workspace] を選択

(7) [Select archive file] を選択して、μ T-Kernel のプロジェクトを選択する。

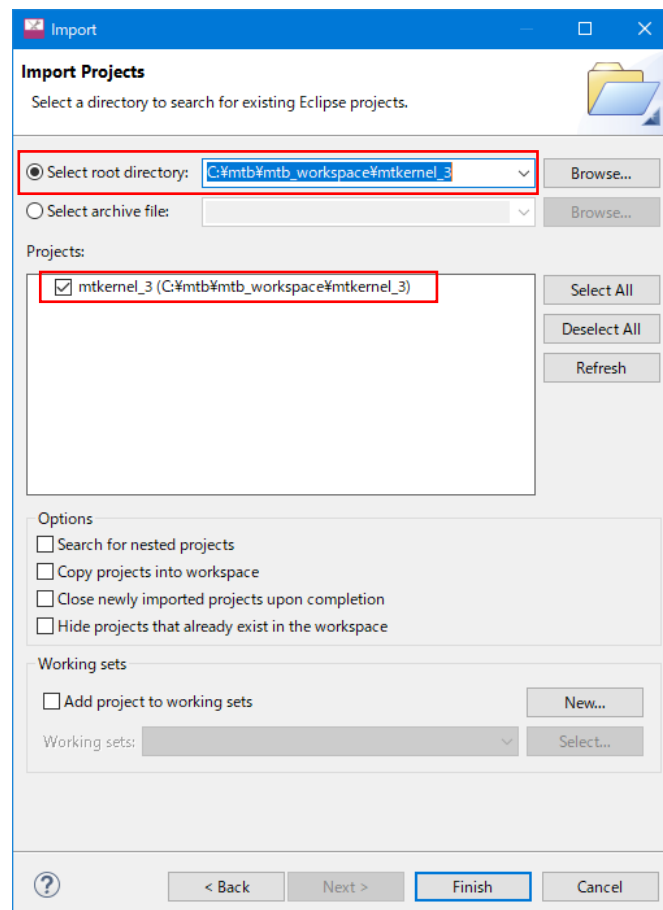


図 10 [Select archive file] を選択

(8) Project Explorer に mtkernel\_3 が追加される。

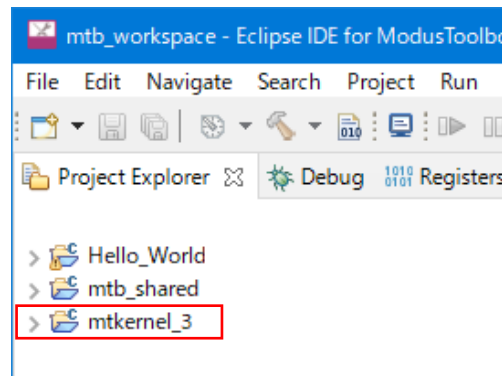


図 11 mtkernel\_3 が追加される。

(9) Project Explorer に表示された mtkernel\_3 を選択する。

(10) [Quick Panel]の[mtkernel\_3 (APP\_KIT\_XMC72\_EVK)]から[Build Application]を選択してビルドを開始する。

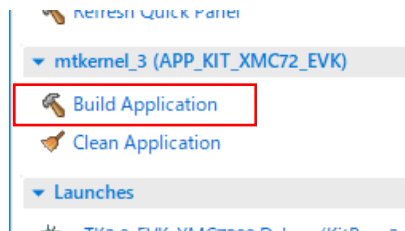


図 12 [Build Application]を選択してビルドを開始

(11) ビルド状況は Console に表示される。

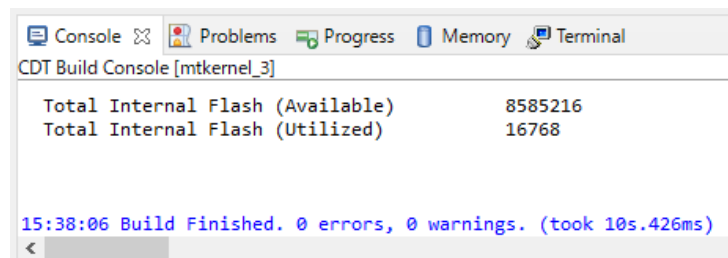


図 13 ビルド完了

(12) ターゲットボードの J7 と PC を付属の USB ケーブルで接続する。



図 14 コンソール用 UART(SCB3)の接続先(J7)

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。

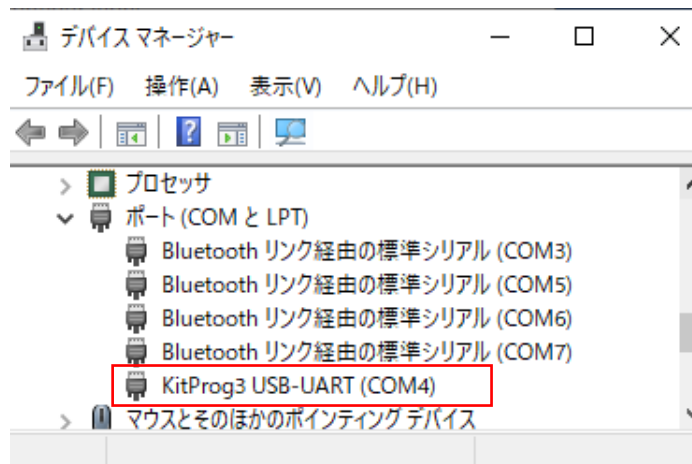


図 15 デバイスマネージャーで KitProg3 USB-UART (COMx) の追加を確認

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)。

(13) [Quick Panel]の[Launches]から[mtkernel\_3 Debug (KitProg3\_MiniProg4)]を選択して評価ボードにプログラムを書き込む。

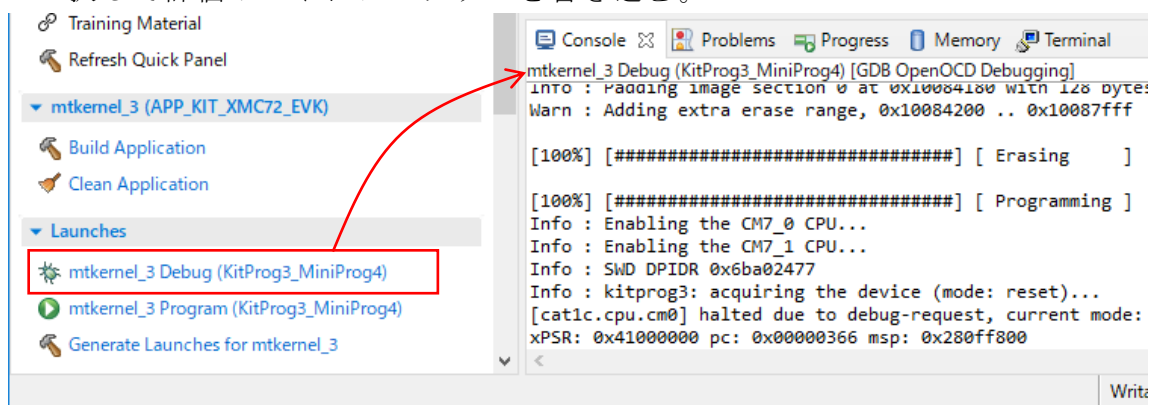


図 16 評価ボードへの書き込み中

① 下図のダイアログが表示されて書き込みが実行されない場合は、[Launches]の[Generate Launches for mtkernel\_3]を選択(実行)する。  
これにより Launches のファイル(.mtbLaunchConfigs/\*.launch)が更新され、正常に書き込めるようになることがある。

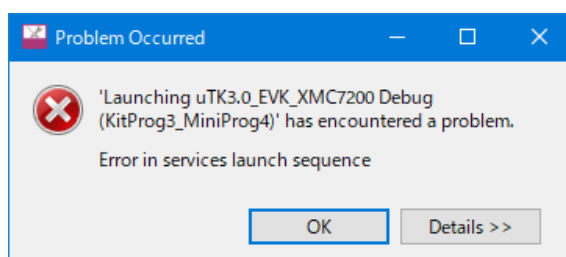


図 17 プログラムの起動時にエラー発生

- (14) 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。

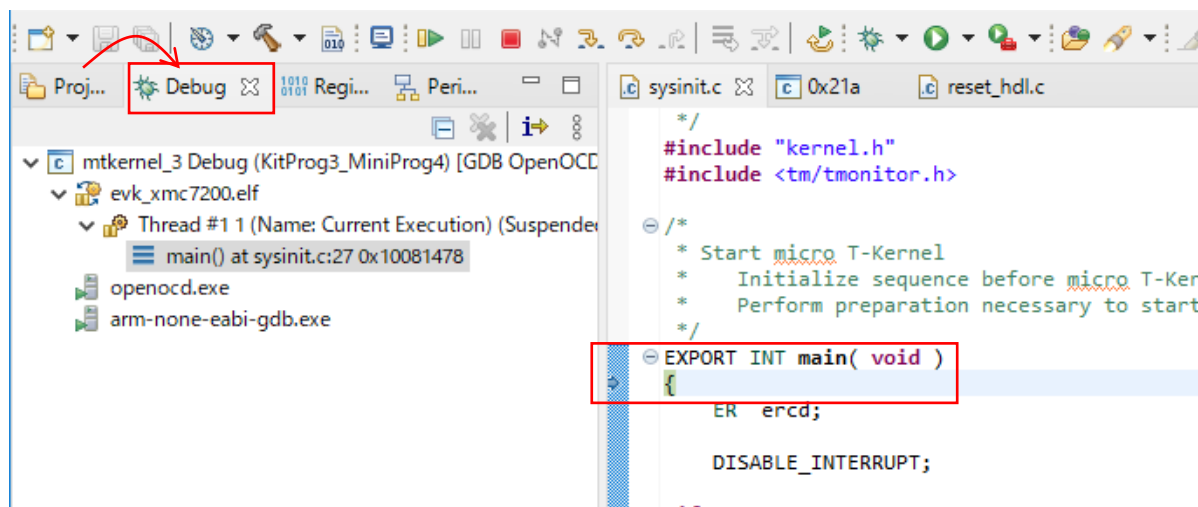


図 18 main 関数の先頭で停止した状態

- (15) 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

- ◆ SCB3 の USB-UART ブリッジを利用して PC と接続する場合、KitProg3 の制限からデータ長／パリティビット／ストップビットはデフォルトの設定(8N1)から変更できない(変更しても正常に動作しない)。詳細については、"KitProg3 user guide"の"2.2.5.1 USB-UART Bridge Feature"を参照のこと。

- (16) [Resume] で実行する。

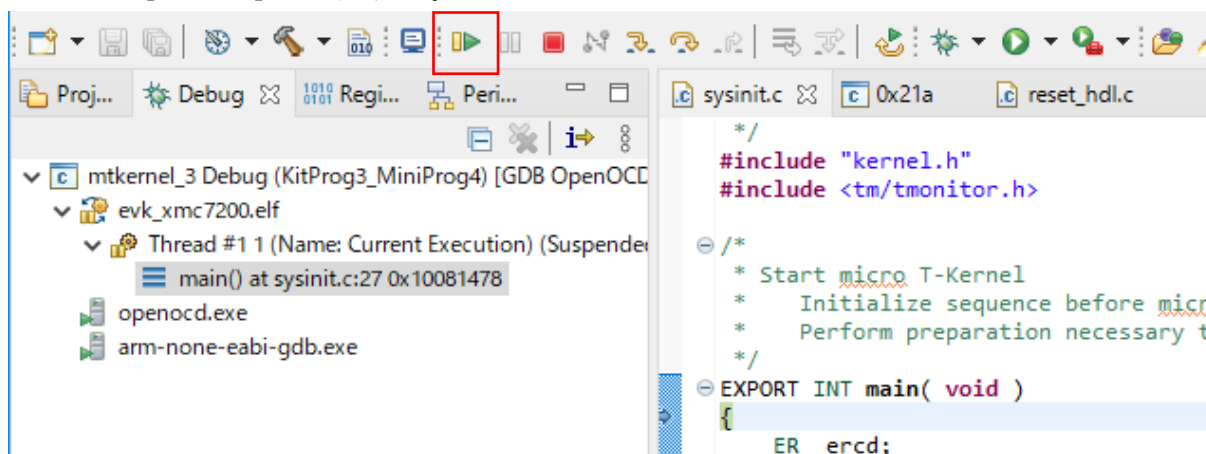


図 19 [Resume] で実行



- (17) 正常に動作すると通信ソフトの画面に以下のメッセージが表示される。

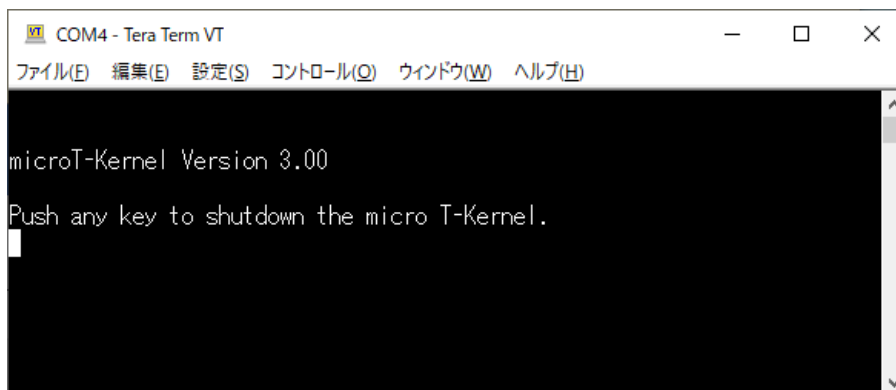



図 20 プロジェクトでのメッセージ表示

この状態で通信ソフトから [Enter] を入力すると、"<< SYSTEM SHUTDOWN >>"と表示される。

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

### 3.4. プログラムの終了手順

実行したプログラムの終了手順は以下の通りである。

- (1) 実行したプログラムは、 [Terminate] で終了する。

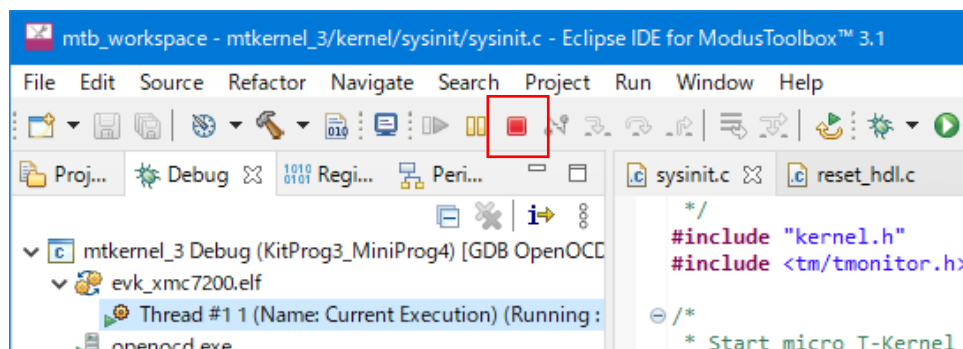


図 21 プログラムを [Terminate] で終了

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

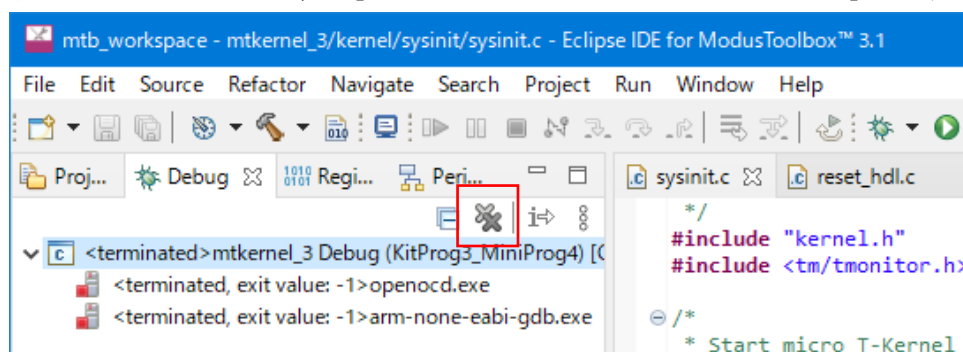


図 22 終了したプログラムは [Remove~] で削除



## 4. サンプルアプリケーション

本章では、「3.3. μ T-Kernel 3.0 のビルド、実行」で動作を確認したサンプルアプリケーションの概要について説明する。

本ソースコードには、以下のサンプルアプリケーションが含まれている。

- ・ `app_program/`      μ T-Kernel 3.0 の標準的なサンプルアプリケーション
- ・ `app_serial/`      シリアルドライバ用のサンプルアプリケーション

以下、各サンプルアプリケーションの内容と実行手順について説明する。

### 4.1. μ T-Kernel 3.0 の標準的なサンプルアプリケーション

標準的なサンプルアプリケーションには、コンソール(通信アプリ)との簡単な通信機能のみを実装してある。

ただし、コンソールとの通信を行っているのはタスクである(後述)ので、この状態で既に μ T-Kernel 3.0 が稼働してタスクが動作する状態になっている。

このアプリケーションを改造する(μ T-Kernel 3.0 の API を用いて拡張していく)ことで、μ T-Kernel 3.0 用のアプリケーションにしていくことも可能である。

#### 4.1.1. サンプルアプリケーションの実行

標準的なサンプルアプリケーションは、「3.3. μ T-Kernel 3.0 のビルド、実行」で説明した手順に従って実行する。

#### 4.1.2. サンプルアプリケーションの内容

このサンプルアプリケーションでは、初期タスク(デフォルトで起動されるタスク)から `app_program.c` に含まれる `usermain()` 関数が呼び出され、コンソール(通信ソフト)にメッセージを表示した後、コンソールからの入力待ちで停止している。コンソールで任意のキーを入力すると、終了メッセージ <<SYSTEM SHUTDOWN>> を表示してシステムが停止する。

実際のプログラムは以下の通りであり、`tm_putstring()` でコンソールに文字列を表示した後は、`tm_getchar()` を呼び出して、その中で入力があるまで待っている。

`app_program/app_program.c` : 初期タスクから呼び出される `usermain()` 関数

---

```
WEAK_FUNC EXPORT INT usermain(void)
{
    tm_putstring((UB*)"Push any key to shutdown the micro T-Kernel.¥n");
    tm_getchar(-1);
    return 0;
}
```

---

## 4.2. シリアルドライバ用のサンプルアプリケーション

app\_serial/には、μ T-Kernel 3.0 のシリアル通信ドライバ用を用いた、簡単なサンプルアプリケーションを実装してある。本節ではこのサンプルアプリケーションの実行方法について説明する。

### 4.2.1. EVK-XMC7200 のシリアルポート

評価ボード EVK-XMC7200 に搭載されている CPU である XMC7200D-E272K8384 には、SCB0～10 の合計 11 本のシリアルポートが内蔵されている。ただし、EVK-XMC7200 ではこのうちの 1 本のみが利用可能な実装になっている。

表 4-1 本実装に含まれるデバイスドライバとピンの割り当て

デバイス名	デバイス	IO ピン	コネクタ	備考
sera	SCB0			
serb	SCB1			
serc	SCB2			
serd	SCB3	P13[0:3]	ARD_P13[0:3]	T-Monitor 共用
sere	SCB4			
:	:			
serk	SCB10			

❗ XMC7200 ではシリアル通信ポート(UART)を SCB(Serial Communication Block)と呼称している。このため、本資料では SCB として説明している。

⚠ コネクタ ARD\_P13[0:3]は IO ピン P13[0:3]とは接続されていない。  
このため、コネクタを利用する場合は予め接続(0Ω 抵抗を実装)する必要がある。

### 4.2.2. シリアルドライバ用のサンプルアプリケーションを実行

シリアルドライバ用のサンプルアプリケーションは app\_serial/app\_program.c に実装してあるが、デフォルトでは ModusToolbox のビルド対象から除外する設定になっている。ModusToolbox のビルド対象にするためには .cyignore ファイルを以下の様に編集して、サンプルアプリケーションのビルド対象を選択する必要がある。

.cyignore でシリアルドライバ用のサンプルアプリケーションを選択

```
## select usermain()
app_program
#app_serial
```

また、シリアルドライバのサンプルプログラムを使用する場合は、上記の処理と共に Makefile を編集して、DEFINES に「SERIAL\_DRIVER」を追加する必要がある。

## Makefile

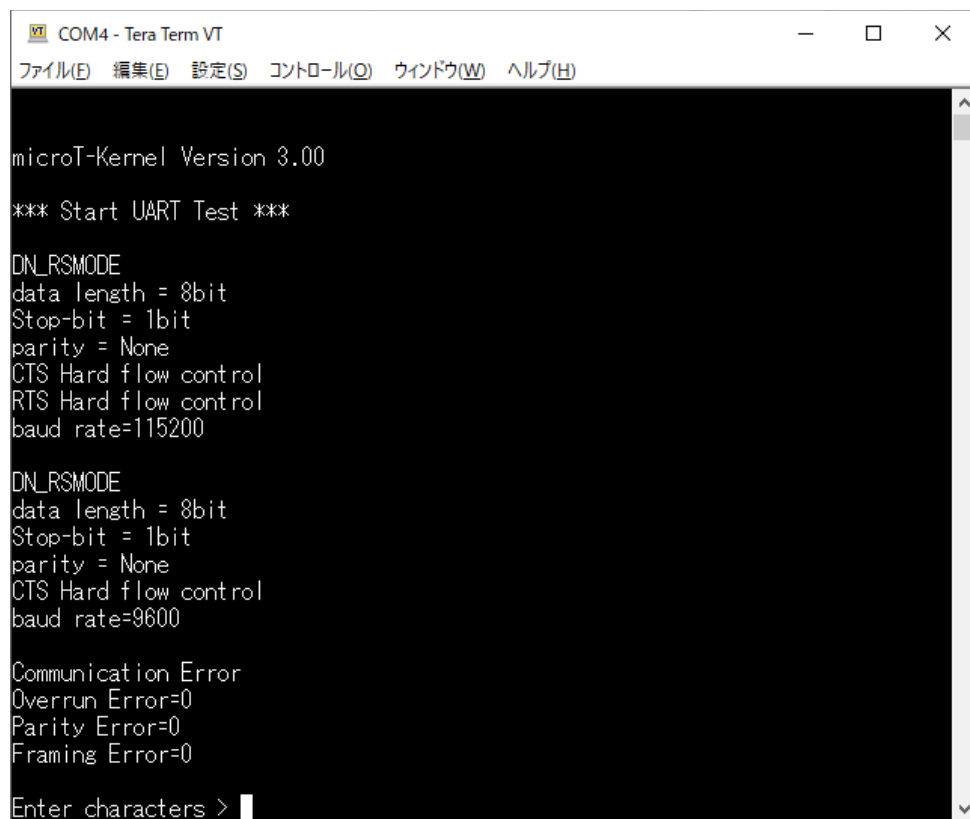
```
# Add additional defines to the build process (without a leading -D).  
DEFINES=_EVK_XMC7200_  
DEFINES+=SERIAL_DRIVER ← 追加
```

- ◆ ModusToolbox に対して無視(ignore)するファイルやディレクトリを指示する場合、`.cyignore` を利用して指定しなければならない。ModusToolbox(Eclipse)の UI の操作によってビルド対象外に設定することはできないので注意が必要である。

`.cyignore` と `Makefile` を修正した後、「3.3. μT-Kernel 3.0 のビルド、実行」の(9)～(16)の手順でビルドして実行する。

### 4.2.3. 実行結果

- (1) [Resume]で実行すると、コンソールに以下の様に表示される。



```
COM4 - Tera Term VT  
ファイル(F) 編集(E) 設定(S) コントロール(C) ウィンドウ(W) ヘルプ(H)  
  
microT-Kernel Version 3.00  
  
*** Start UART Test ***  
  
DNL_RSMODE  
data length = 8bit  
Stop-bit = 1bit  
parity = None  
CTS Hard flow control  
RTS Hard flow control  
baud rate=115200  
  
DNL_RSMODE  
data length = 8bit  
Stop-bit = 1bit  
parity = None  
CTS Hard flow control  
baud rate=9600  
  
Communication Error  
Overrun Error=0  
Parity Error=0  
Framing Error=0  
  
Enter characters > |
```

図 23 シリアルドライバサンプルでのメッセージ表示

- (2) “Enter characters >”と表示されて停止するので、通信ソフトを選択して、キーボードから文字を入力する。

入力した文字は 4 バイト入力する毎にまとめて出力される。



```
Communication Error  
Overrun Error=0  
Parity Error=0  
Framing Error=0  
  
Enter characters > abcdefghijklmnopqrstuvwxyz1234567890-^¥@[:;],./¥
```

図 24 エコーバックのメッセージ表示

#### 4.2.4. サンプルアプリケーションの内容

シリアルドライバのサンプルアプリケーションの主な部分のみを抜き出すと以下の様になっている。

`app_serail/app_program.c` : シリアルドライバサンプルの主な実装(抜粋)

---

```
EXPORT INT usermain(void)
{
    T_CTSK ctsk = { 0, TA_HLNG|TA_RNG1, (FP)sampleApplicationTask, 15, 1*1024 };
    ER      ercd = E_OK;

    ercd = tk_cre_tsk(&ctsk);
    if (ercd < E_OK) goto ERROR;
    ercd = tk_sta_tsk((ID)ercd, 0);
    if (ercd != E_OK) goto ERROR;

    ercd = tk_slp_tsk( TMO_FEVR );
ERROR:
    return 0;
}

#define RS_DEVM "serd"
LOCAL void sampleApplicationTask(INT stacd, void *exinf)
{
    LOCAL B read_buf[READ_SIZE];
        :

    dd = tk_opn_dev( (UB*)RS_DEVM, TD_UPDATE );
    reqid = tk_swri_dev( dd, 0, (void*)ENTER_MESSAGE, SIZE_OF_EM, &asz_com );
    while(1){
        reqid = tk_srea_dev( dd, 0, (void*)read_buf, 4, &asz_com );
        reqid = tk_swri_dev( dd, 0, (void*)read_buf, asz_com, &asz_com );
    }
    (void)tk_cls_dev( dd, 0 );
    tk_exd_tsk();
}

```

---

初期タスクから呼び出される `usermain()` 関数の中でタスクを生成・起動し、起動されたタスクの中で、デバイス管理機能の `tk_opn_dev()` を用いてシリアルポート `serd` をオープンしている。

シリアルポートからの読み込み(データの受信)は `tk_srea_dev()`、書き込み(データの送信)は `tk_swri_dev()` で行っている。ここで読み込みと書き込みの処理が 4 バイト単位になっているため、入力と出力が 4 バイト単位で処理されることになる。

また、本実装では実行されることはないが、シリアルポートの利用を終了する場合は `tk_cls_dev()` を用いてシリアルポートをクローズしている。

## 5. μ T-Kernel 3.0 のディレクトリ／ファイル構成

EVK-XMC7200 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構成にしてある。

また、ModusToolbox のプロジェクトとして取り込むために必要なディレクトリやファイルは、μ T-Kernel 3.0 のディレクトリのルートに追加してある。

表 5-1 プロジェクトのファイル構成

ディレクトリ名、ファイル名	内容
μ T-Kernel 3.0 関連	
app_program/	標準的なサンプルアプリケーション
app_serial/	シリアルドライバ用のサンプルアプリケーション
config/	コンフィギュレーション
device/	サンプルデバイスドライバ
include/	各種定義ファイル
kernel/	μ T-Kernel 3.0 本体
lib/	μ T-Kernel 3.0 用のライブラリ
docs/	ドキュメント
README.md	概要説明
ucode.png	μ T-Kernel 3.0 の ucode
ModusToolbox 関連	
Makefile	ModusToolbox 用の Make ファイル
evk_xmc7200_Flash.ld	EVK-XMC7200 用のリンクスクリプト
.cyignore	ModusToolbox 用のビルド対象外ファイルの指定
.mtbLaunchConfigs/	プログラム起動(書き込み)用コンフィギュレーション
.settings/	Eclipse 用の設定ファイル
.project	Eclipse 用のプロジェクト構成ファイル
.cproject	Eclipse の CDT プラグイン用の構成ファイル
bsps/	ModusToolbox の BSP 関連の構成ファイル
deps/	ModusToolbox のライブラリ関連の構成ファイル
libs/	ModusToolbox のライブラリ関連の構成ファイル
LICENSE	ModusToolbox に関するライセンスの説明
README_mtb.md	ModusToolbox に関する概要説明
.gitignore	Git のトラッキングの対象外にするファイルを指定

以下、μ T-Kernel 3.0 関連のディレクトリやファイルの概要について説明する。

ModusToolbox 関連のディレクトリやファイルに関する詳細については、ModusToolbox の資料を参照のこと。

## 5.1. μ T-Kernel 3.0 のソースコード

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

app\_program/と app\_serial/には、μ T-Kernel 3.0 用のサンプルアプリケーションのプログラムが含まれる。アプリケーションは任意のディレクトリに実装できるが、本実装では、μ T-Kernel 3.0 のリファレンスコードにならって、app\_program/と app\_serail/に配置してある。

config/以下にあるファイルの設定(マクロ定義の設定値)により、μ T-Kernel 3.0 のコンフィグレーション(タスクやセマフォの利用可能な数など)を調整できる。具体的なコンフィギュレーション方法に関しては、以下の資料を参照のこと。

- ・「μ T-Kernel3.0 共通実装&構成仕様書」(ユーシーテクノロジー(株))
- ・「μ T-Kernel 3.0 共通実装仕様書」(トロンフォーラム)

device/には、サンプルのデバイスドライバが含まれる。どのデバイスに対応しているかはターゲットボードや実装に依存する。

kernel/、include/、lib/には、μ T-Kernel 3.0 自体のソースコードが含まれる。

## 5.2. その他の μ T-Kernel 3.0 関連のディレクトリとファイル

docs/には、μ T-Kernel 3.0 関連の資料が含まれる。このディレクトリに含まれている資料の一覧については、「2.4. 関連ドキュメント」を参照のこと。

README.md は、本 μ T-Kernel 3.0 の実装に対する概要説明である。本ソースコードに対するライセンスなどに関する説明が含まれているので、本ソースコードを用いて開発を開始する前に内容を確認しておく必要がある。

ucode.png は、本実装の元となった μ T-Kernel 3.0 の ucode の画像である。

## 5.3. ビルド用のファイル

Makefile、evk\_xmc7200\_Flash.ld、.cyignore は ModusToolbox でのビルドに利用する。

Makefile には EVK-XMC7200 版の μ T-Kernel 3.0 をビルドする際に必要となるマクロ定義(\_EVK\_XMC7200\_)とリンカスクリプト(evk\_xmc7200\_Flash.ld)の指定を追加してある(以下の**太字部分**を追加)。

---

```
:
DEFINES=_EVK_XMC7200_
:
LINKER_SCRIPT=evk_xmc7200_Flash.ld
:
```

---

### リスト 1 Makefile の μ T-Kernel 3.0 用の改変箇所

evk\_xmc7200\_Flash.ld は、gcc 用のリンカスクリプトである。

本実装に含まれる `evk_xmc7200_Flash.ld` は、ModusToolbox Version 3.1.0 が生成するサンプルプログラム Hello World の Cortex-M7 用のリンカスクリプト `linker.ld` をベースに、μ T-Kernel 3.0 の実装に合わせて、セクションの並びやサイズを変更してある。

- ❶ Hello World のリンカスクリプト `linker.ld` は以下のディレクトリに生成される。  
`Hello_World/bsps/TARGET_APP_KIT_XMC72_EVK/COMPONENT_CM7/TOOLCHAIN_GCC_ARM/`
- ❷ ModusToolbox でのサンプルプログラム Hello World の生成手順に関しては、「3.2. ModusToolbox の動作確認」を参照のこと。

`.cyignore` は、ModusToolbox に対してビルドから除外するファイルやディレクトリを指示するファイルである。本実装では、`.cyignore` ファイルを編集することでサンプルアプリケーションのビルド対象を選択している。

`.cyignore` の詳細については、ModusToolbox の資料を参照のこと。

- ◆ ModusToolbox は Eclipse ベースの開発環境であるが、Eclipse の UI 操作によってビルド対象外にする機能は利用できない。(設定しても無視される。)
- ◆ Makefile と `.cyignore` は ModusToolbox 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。

## 5.4. その他のファイル

表 5-1 に掲載していないファイルやディレクトリは ModusToolbox によって生成したファイルやディレクトリである。これらの詳細に関しては、ModusToolbox の資料を参照のこと。

## 5.5. アプリケーションプログラムの追加

### 5.5.1. アプリケーションプログラムの追加ディレクトリ

μ T-Kernel 3.0 用のアプリケーションプログラムは任意の場所に実装することができる。本実装では、既に説明した通り `app_program/` と `app_serial/` の下にサンプルのアプリケーションプログラムを実装してある。

- ❶ `kernel/usermain/usermain.c` にもサンプルアプリケーションのソースコードが含まれているが、こちらは `.cyignore` によってビルド対象から除外してある。

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。`app_program/` 以下にサブディレクトリを作成しても構わない。

他のディレクトリ、例えば `my_application/` などを用意して実装を進めても構わない。アプリケーション用のディレクトリの名称や配置に制限はないが、μ T-Kernel 3.0 本体や

ModusToolbox で利用しているディレクトリやファイルと区別できる名称、配置にすることを推奨する。

なお、ModusToolbox では、新規に追加されたソースコードは ModusToolbox がビルド時に自動的に検出してビルド対象として登録する。

他の開発環境の様に開発者がプロジェクトに明示的に追加する必要はないといった点で便利ではあるが、未完成のソースコードを追加した場合も、ビルドを実行するだけでビルド対象として処理されるのでビルド時にエラーが発生することになる。この様な場合、ModusToolbox では「ビルド対象から除外する」といった逆の処理を行うことになる。

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。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。

### 5.5.2. アプリケーションプログラムの実装例

初期タスクとは別にタスク(subTask)を生成して起動するための実装例を以下に示す。アプリケーションを実装する際の参考にされたい。

---

```
#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 1                                ←ここを0にするとサブタスクは起動しない。
        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 = {
    exinf          = 0,
    tskatr         = TA_HLNG|TA_RNG0,
    task           = (FP)&subTask,
    itskpri        = 1,
    stksz          = 1*1024,
#ifdef USE_OBJECT_NAME
    dsname         = "SubTask",
#endif
    bufptr         = NULL,
};

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 app\_main.c へのタスク生成・起動処理の追加例

上記のサンプルコードは、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するだけの簡単なプログラムである。なお、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 +  $\alpha$  の時間がかかる。

また、タスクの生成と起動の手順を理解しやすくするため、エラー処理などは含めていない。

$\mu$ T-Kernel 3.0

**構築手順書 (EVK-XMC7200)**

**Rev 3.00.01 (December, 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

©2023 Ubiquitous Computing Technology Corporation All Rights Reserved.