μT-Kernel3.0
RX231 IoT-Engine 向け
構築手順書書

Version.01.00.00

2020.03.13

目次

1. 概要 3
1.1 目的 3
1.2 対象 OS およびハードウェア 3
1.3 対象開発環境 3
2. C コンパイラ 4
2.1 GCC バージョン 4
2.2 動作検証時のオプション 4
2.3 インクルードパス 4
2.4 標準ライブラリ 4
3. e² studio を用いたビルド手順（例） 5
3.1 開発ソフトの準備 5
3.2 e² studio の設定 5
3.3 プロジェクトの作成 5
3.4 プロジェクトのビルド 7
4. 実機でのプログラム実行 8
4.1 E² studio によるプログラムの実行 8
5. アプリケーションプログラムの作成 10

1. 概要

1.1 目的

本書は、TRON フォーラムからソースコードが公開されてる RX231 IoT-Engine 向けμT-Kernel3.0 の開発環境の構築手順を記す。
以降、本ソフトとは前述のμT-Kernel3.0 のソースコードを示す。

1.2 対象 OS およびハードウェア

本書は以下を対象とする。

分類	名称	備考
OS	μT-Kernel3.00.01	TRON フォーラム
実機	RX231 IoT-Engine	UC テクノロジー製
搭載マイコン	RX200 シリーズ RX231 グループ R5F52318ADFL	ルネサス エレクトロニクス製

1.3 対象開発環境

本ソフトは C 言語コンパイラとして、GCC(GNU Compiler)を前提とする。
ただし、本ソフトはハードウェア依存部を除けば、標準の C 言語で記述されており、他の C 言語コンパイラへの移植も可能で可能である。

2. C コンパイラ

2.1 GCC バージョン

本ソフトの検証に用いた GCC のツールチェーンを以下に記す。

ツールチェーン　　**GCC for Renesas 8.3.0.201904-GNURX Toolchain**

（ https://gcc-renesas.com/ja/rx-download-toolchains ）

2.2 動作検証時のオプション

本ソフト動作検証時のコンパイラの最適化オプションは、-O2 を設定している。

リンクタイム最適化-flto(Link-time optimizer)については動作を保証しない。

2.3 インクルードパス

μT-Kernel3.0 のソースディレクトリの以下のディレクトリを指定する必要がある。

ディレクトリパス	内容
¥config	コンフィギュレーションファイル
¥include	共通ヘッダファイル
¥kernel¥knlinc	カーネル内共通ヘッダファイル

¥kermel¥knlinc は OS 内部でのみ使用するヘッダファイルである。ユーザプログラムについては、¥config と¥include のみを使用する。

2.4 標準ライブラリ

本ソフトは基本的にはコンパイラの標準ライブラリを使用しない。よって、リンク時のオプションで-nostdlib を指定している。

ただし、ユーザのアプリケーションや使用するライブラリなどによっては、標準ライブラリが必要となる場合がある。

3. e² studio を用いたビルド手順（例）

本ソフトをビルドするための開発環境の構築と、それを用いたビルド手順を説明する。
本ソフトは極力、特定の開発環境に依存しないように作られている。ここでは、ルネサス エレクトロニクスの統合開発環境 e² studio を使用した例を説明する。
なお、各種の設定は一例であり、開発するアプリケーションなどに応じて適切な値を設定する必要がある。

3.1 開発ソフトの準備

(1) 統合開発環境 e² studio

e² studio は、オープンソースの"Eclipse"をベースとした、ルネサス製マイコン用の統合開発環境である。

本ソフトの動作検証には e² studio の以下のバージョンを使用した。

e² studio Version 7.6.0

e² studio は以下の e² studio のホームページからインストーラが入手可能である。なお、ダウンロードにはルネサスへの登録が必要である。

https://www.renesas.com/jp/ja/products/software-tools/tools/ide/e2studio.html

インストーラによる e² studio のインストールの際には、対象デバイスとして RX マイコン、C コンパイラは GCC を選択する。
RX 用の C コンパイラは GCC の他に RX-CC を e² studio はサポートしている。ただし、本ソフトのソースコードは GCC 用であるため、RX-CC ではそのままでは使用できない。

e² studio のインストールや操作については、上記のホームページを参照のこと。

3.2 e² studio の設定

(1) ワークスペースの作成

e² studio の初回起動時、指示に従いワークスペースを作成する。ワークスペースは、e² studio の各種設定などが保存される可能的な作業場である。

3.3 プロジェクトの作成

E² studio にて以下の手順で本ソフトのプロジェクトを作成する。

(1) メニュー「新規」→「C/C++ プロジェクト」を選択する。
開いた新規 C/C++プロジェクトのテンプレート画面で「GCC for Renesas RX C/C++ Executable Project」を選択する。
次の「GCC for Renesas RX」画面で以下を設定する。
・プロジェクト名：任意
・ロケーション：任意

次画面で以下を設定する。
・言語： C
・ツールチェーン：GCC for Rebesas RX
・ツールチェーン・バージョン：任意※
・ターゲット・デバイス：R5F52318AxFL

※本ソフトの検証にはバージョン 8.3.0.201904 を使用。

(2) メニュー「ファイル」→「インポート…」を選択する。
開いた選択画面で「一般」→「ファイルシステム」を選択し、ファイルシステム画面で本ソフトのソースコードのディレクトリを入力する。

(3) メニュー「プロジェクト」→「プロパティ」を選択する。
項目「C/C++ビルド」→「設定」→「ツール設定」タブ選択し、以下を設定する。
「Optinmization」
-ffunction-sections と-fdata-sections のみ選択

「Compiler」→「Includes」
「Include file directories」にインクルードパスの追加
「2.3 インクルードパス」を参照
「Macro Defines」にターゲット名を定義。
_IOTE_RX231_

「Assembler」→「Source」
「User defined option」にターゲット名を定義
-D_IOTE_RX231_

「Assembler」→「Includes」

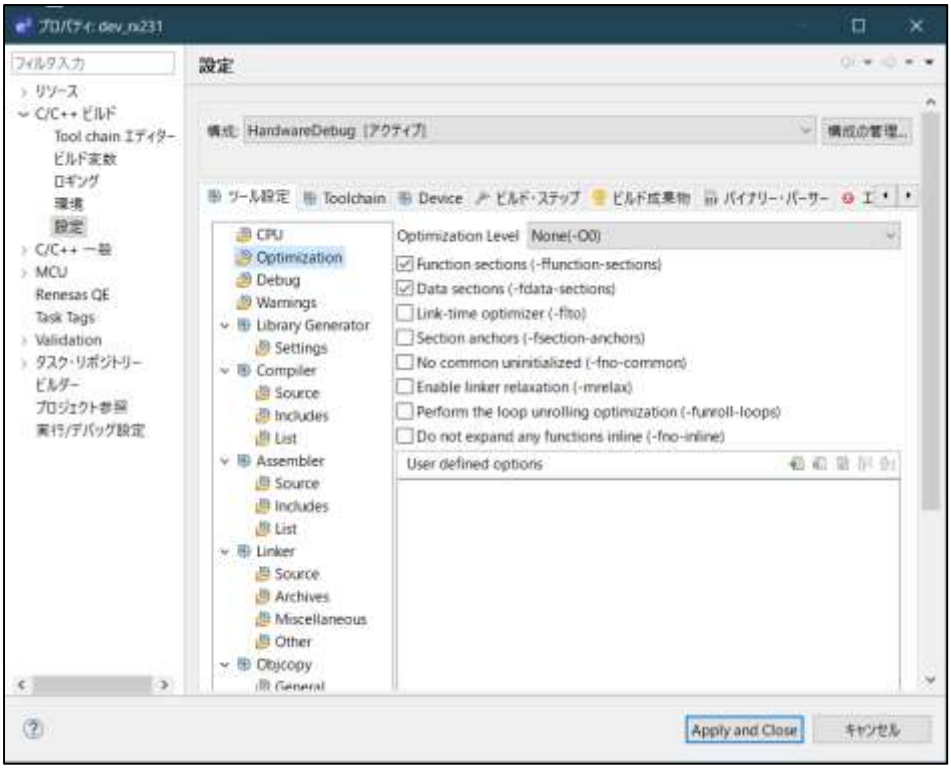
「Include file directories」にインクルードパスの追加

「2.3 インクルードパス」を参照

「Linker」→「Source」

「Linker script」に、以下のファイルのパスを設定する。

etc¥linker¥iote_rx231¥tkernel_map.ld



3.4 プロジェクトのビルド

メニュー「プロジェクト」→「プロジェクトのビルド」を選択すると、本ソフトのソースコードがコンパイル、リンクされ、実行コードの ELF ファイルが生成される。

また、メニュー「プロジェクト」→「ビルド構成」→「アクティブにする」でワーキング・セットを選択することができる。

4. 実機でのプログラム実行

ビルドしたプログラムを実機上で実行する方法を記す。

実機としては、RX231 IoT-Engine Arduino Evaluation Kit を前提に説明する。
エミュレータにはルネサス エレクトロニクス製の E1 を使用し、前章で説明した E² studio の開発環境から実機に実行コードを転送し、実行、デバッグを行う方法を記す。

4.1 E² studio によるプログラムの実行

(1) E² studio のメニューからメニュー「実行」→「デバッグの構成」を選択し、開いたダイアログから項目「Renesas GDB Hardware Debugging」を選択する。

(2) 「新規構成」ボタンを押し、「Renesas GDB Hardware Debugging」に構成を追加する。

(3) 追加した構成を選択し、「構成の作成、管理、実行」画面にて以下の設定を行う。

「メイン」タブ

名前：(任意) を入力

C/C++アプリケーション：ビルドした ELF ファイル

「Debugger」タブ

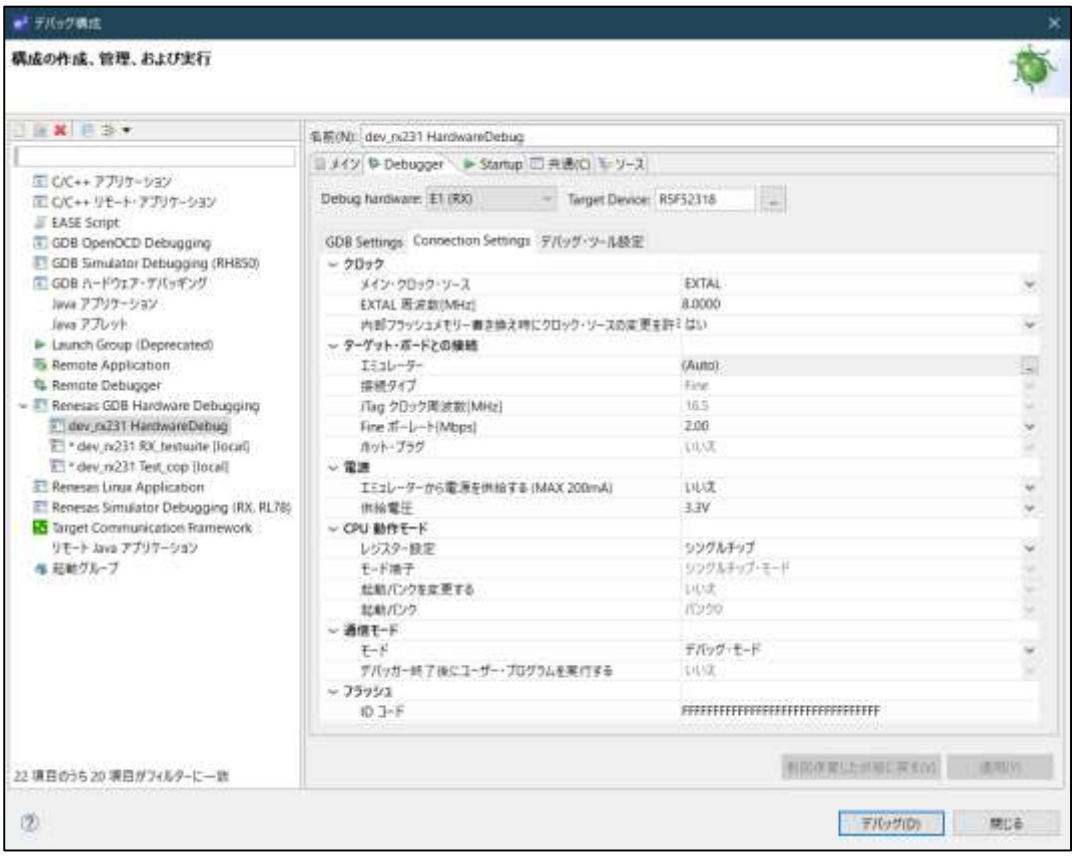
Debug Hardware：「EX (R1)」を選択

Target Deveice：「R5F52318」を選択

「Connection Setting」タブを選択

「クロック」→「EXTAL 周波数」8.0000 を入力

「エミュレータから電源を供給する」いいえを選択



(4) デバッグ開始

「デバッグ」ボタンを押すとプログラムが実機に転送され、ROM に書き込まれたのち、実行される。

本ソフトでは、OS 起動後にサンプルのアプリケーションが実行される。サンプルのアプリケーションは、初期タスクから二つのタスクを生成、実行し、T-Monitor 互換ライブラリを使用してメッセージを出力する簡単なプログラムである。
ソースコードは以下のファイルに記述されている。

/app_sample/usermain.c

5. アプリケーションプログラムの作成

アプリケーションプログラムは、OS とは別にアプリ用のディレクトリを作成して、そこにソースコードを置き、OS と一括でコンパイル、リンクを行う。
e² studio を使用する場合は、OS を構築するプロジェクトに、アプリのソースコードも追加する。

公開しているμT-Kernel3.0 のソースコードには、サンプルのアプリケーションが含まれている。/app_sample ディレクトリが、サンプルのアプリケーションのディレクトリなので、これをユーザのアプリケーションのディレクトリに置き換えれば良い。

アプリケーションには、usermain 関数を定義する。OS は起動後に初期タスクから usermain 関数を実行する。詳細はμT-Kernel3.0 共通実装仕様書「5.2.3 ユーザ定義メイン関数 usermain」を参照のこと。

アプリケーションから OS の機能を使用する場合は、以下のようにヘッダファイルのインクルードを行う。

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

T-Monitor 互換ライブラリを使用する場合は、さらに以下のインクルードが必要である。

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

μT-Kernel3.0 の機能については、μT-Kernel3.0 仕様書を参照のこと。

以上