μT-Kernel3.0
RZ/A2M IoT-Engine 向け
構築手順書書

Version.01.00.01

2021.11.15

更新履歴

版数(日付)	内　容
1.00.01 (2021.11.15)	● 誤記修正「1.2 対象 OS およびハードウェア」 (誤)STM32L4 IoT-Engine (正)RZ/A2M IoT-Engine
1.00.00 (2021.8.27)	● 初版

目次

1. 概要 4
1.1 目的 4
1.2 対象 OS およびハードウェア 4
1.3 対象開発環境 4
2. C コンパイラ 5
2.1 GCC バージョン 5
2.2 動作検証時のオプション 5
2.3 インクルードパス 5
2.4 標準ライブラリ 6
3. 開発環境と構築手順 7
3.1 Make を使用したビルド方法 7
3.1.1 ビルド環境の準備 7
3.1.2 プロジェクトのビルド 9
3.2 e² studio を使用した構築手順 10
3.2.1 e² studio の準備 10
3.2.2 プロジェクトの作成 10
3.2.3 プロジェクトのビルド 13
4. アプリケーションプログラムの作成 14
5. 実機でのプログラム実行 15
5.1 SEGGER J-Link Software のインストール 15
5.2 E² studio によるプログラムの実行 15
5.3 外部 FLASH ROM からのプログラムのブート 16

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いる RZ/A2M 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-Kernel3.00.05	TRON フォーラム
実機	RZ/A2M IoT-Engine	UC テクノロジー製
搭載マイコン	RZ/A2M (R7S921053VCBG)	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 のバージョンを以下に記す。

GNU Arm Embedded Toolchain 10-2020-q4-major

2.2 動作検証時のオプション

本ソフトの動作検証時のコンパイラ及びリンカのオプションを示す。なお、オプションは、開発するアプリケーションに応じて適したものを指定する必要がある。

FPU を使用しない場合は-mfloat-abi=soft、FPU を使用する場合は-mfloat-abi=soft を指定する。

最適化オプションは、検証時には-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。

その他の主なオプションを以下に示す。

コンパイルオプション

```
-mcpu=cortex-a9 -mthumb -ffreestanding -std=gnu11
```

リンクオプション

```
-mcpu=cortex-a9 -mthumb -ffreestanding -nostartfiles
```

2.3 インクルードパス

μT-Kernel3.0 のソースディレクトリ中の以下のディレクトリを、ビルド時のインクルードパスに指定する。

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。ユーザプログラムでは、¥config と¥include のヘッダファイルのみを使用する。

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。ただし、演算に際してライブラリが使用される場合がある。本ソフトではデバッグサポート機能の中の演算で使用されている（td_get_otm および td_get_tim の処理内で__aeabi_idivmod 関数が使用されている）。

デバッグサポート機能を使用しない場合は、標準ライブラリは不要である。リンカオプションで-nostdlib が指定可能となる。ただし、アプリケーションで使用している場合はこの限りではない。

3. 開発環境と構築手順

本ソフトをビルドするための開発環境の準備と構築手順を説明する。

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。ここでは例として、Windows の PC において、自動ビルドツール Make を使用する場合と、ルネサス エレクトロニクスの統合開発環境 e² studio を使用する場合を説明する。
なお、ここに示す開発環境や構築手順はあくまで例であり、ユーザそれぞれの環境などによって差異がある場合がある

3.1 Make を使用したビルド方法

3.1.1 ビルド環境の準備

(1) C コンパイラのインストール
GCC コンパイラー式を以下からダウンロードする。

GNU Arm Embedded Toolchain
https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-rm

本稿作成時に検証したバージョンは以下の通り。

gcc-arm-none-eabi-10-2020-q4-major

ダウンロードした zip ファイルを任意の場所に展開する。

(2) 開発ツールのインストール
GCC toolchain を使用するためのツール一式（make など）を以下からダウンロードする。

xPack Windows Build Tools
https://github.com/xpack-dev-tools/windows-build-tools-xpack/releases

本稿作成時に検証したバージョンは以下の通り。
xPack Windows Build Tools v4.2.1-2

ダウンロードした zip ファイルを任意の場所に展開する。

(3) 実行パスの設定

Windows のコマンドシェル（PowerShell またはコマンドプロンプト）から、GCC および Make が実行可能となるように、環境変数 path に GCC を展開したディレクトリ内の¥bin ディレクトリのパスおよび、xPack Windows Build Tools を展開したディレクトリ内の¥bin ディレクトリのパスを追加設定する。

コマンドシェルから GCC（arm-none-eabi-gcc）および make コマンドが実行可能であることを確認する。

(4) makefile の設定

本ソフトのソースコード中の Make 用ビルドディレクトリ(build_make)に makefile が格納されている。

ディレクトリ(build_make)の内容を以下に示す。

名称	説明
makefile	μT-Kernel 3.0 のビルド規則（ルート）
iote_rza2m.mk	RZ/A2M IoT-Engine 用のビルド規則
iote_****.mk	その他の IoT-Engine 用のビルド規則 RZ/A2M 版μT-Kernel では使用しない
/mtkernel_3	Make 作業用ディレクトリ

makefile ファイルの先頭の以下の定義を変更する。

定義名	初期値	説明
EXE_FILE	mtkernel_3	ビルドする実行ファイル名
TARGET	_IOTE_M367_	対象とするハードウェア RZ/A2M IoT-Engine の場合は「**_IOTE_RZA2M_**」に変更する

また、iote_rza2m.mk の先頭の以下の定義を必要に応じて変更する。

定義名	初期値	説明
GCC	arm-none-eabi-gcc	C コンパイラのコマンド名
AS	arm-none-eabi-gcc	アセンブラのコマンド名
LINK	arm-none-eabi-gcc	リンカのコマンド名
CFLAGS	省略(※)	C コンパイラのオプション
ASFLAGS	省略(※)	アセンブラのオプション
LFLAGS	省略(※)	リンカのオプション
LINKFILE	省略(※)	リンク定義ファイル

※ iote_rza2m.mk ファイルの記述を参照

他のファイルについては OS のソースコードの変更が無い限り、変更する必要はない。ただし、ユーザプログラムの追加等については、それぞれ対応するビルド規則を記述する必要がある。

また app_sample ディレクトリ下のアプリケーションについては以下のファイルでビルド規則が記述されている。

build_make¥mtkernel_3¥app_sample¥subdir.mk

app_sample ディレクトリにソースファイルを追加しても対応可能なビルド規則となっているが、サブディレクトリには対応してない。サブディレクトリを作成する場合はビルド規則の記述を変更する必要がある。

3.1.2 プロジェクトのビルド

Windows のシェル（PowerShell またはコマンドプロンプト）上で、build_make ディレクトリをカレントディレクトリとし、以下のコマンドを実行する。

```
make all
```

ビルドが成功すると、build_make ディレクトリ下に、実行コードの ELF ファイルが生成される。ELF ファイルの名称は EXE_FILE で指定した名称である(初期値では mtkernel_3.elf が生成される)。

また、以下のコマンドを実行すると、ELF ファイルおよびその他の中間生成ファイルが削除される。

```
make clean
```

3.2 e² studio を使用した構築手順

3.2.1 e² studio の準備

(1) e² studio のインストール

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 2021-04 Windows

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、ダウンロードにはユーザ登録が必要である。

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 RZ マイコンを選択する。
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。

(2) 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。

3.2.2 プロジェクトの作成

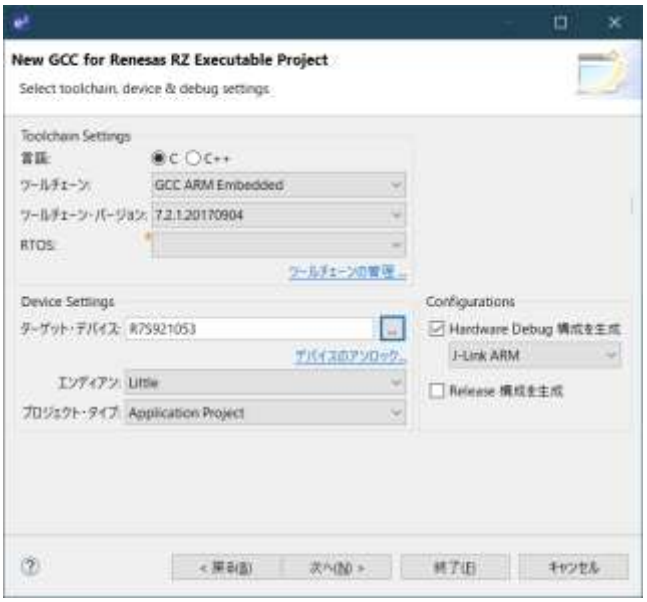
E² studio にて以下の手順で本ソフトのプロジェクトを作成する。

(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「Renesas RZ」から「GCC for Renesas RZ C/C++ Executable Project」を選択する。

次の「New GCC for Renesas RZ Executable Project」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

次の「Select toolchain, device & debug settings」画面で以下を設定する。
・ツールチェーン：「GCC ARM Embedded」

・ターゲット・デバイス：「R7S921053」



「Select Additional CPU Options」画面までは何も選択せず進み、ここで以下を設定する。

・Floating Point ABI:「Soft」(FPU を使用しない場合)

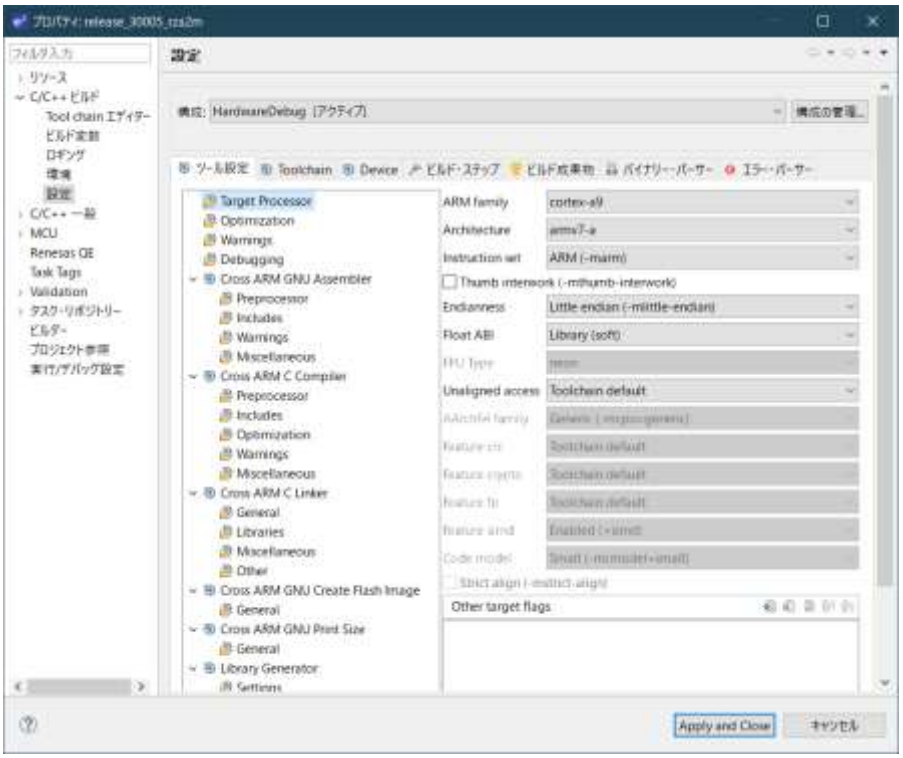
・Floating Point ABI:「Softfp」(FPU を使用する場合)

プロジェクトの作成を終了する。ここで自動生成されたディレクトリ（generate および src）は不要なので削除すること。

(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。
なお、(1)でプロジェクトのロケーションに、既にソースコードのディレクトリが存在するディレクトリを指定した場合は、インポートは不要である。

(3) メニュー「プロジェクト」→「プロパティ」を選択する。
以降、プロパティのダイアログにて各項目を設定していく。なお、本書の設定は一例であり、必要に応じて変更すること。

(4) ダイアログの項目「C/C++ビルド」→「設定」を選択し、「ツール設定」タブを開くと以下のように表示されるので、以降の手順に従って設定を行う。



「Optinmization」

「Optimaization Level」は任意

オプションは「-ffreestanding」のみ選択

「Cross ARM GNU Assembler」

「Preprocessor」の「Defined symboles(-D)」にターゲット名を定義。

_IOTE_RZA2M_

「Include」にμT-Kernel3.0のインクルードパスを設定する。

「Cross ARM C Compiler」

「Preprocessor」の「Defined symbols(-D)」にターゲット名を定義。

_IOTE_RZA2M_

「Includes」にμT-Kernel3.0のインクルードパスを設定する。

「Optimization」の「Language standard」で「GNU ISO C11(-std=gnu11)」を選択する

「Cross ARM C Linker」

「General」の「Script files」に、μT-Kernel3.0の以下のスクリプト・ファイルのパスを設定する。

etc¥linker¥iote_rza2m¥tkernel_map.ld

「-nostartfiles」のみを選択する。

「Entry Point」に「-e_Reset_Handler」を設定する。

(5) ダイアログの項目「C/C++ビルド」→「設定」を選択し、「Toolchain」タブを開いて以下の設定を行う。

「ツールチェーン」:「GCC ARM Embedded」を選択する。

「バージョン」: 任意

3.2.3 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。

4. 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。

公開されているμT-Kernel3.0 のソースコードには、/app_sample ディレクトリにサンプルのアプリケーションのソースコードが含まれている。
ソースコードは以下のファイルに記述されている。

```
/app_sample/app_main.c
```

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シリアル出力にメッセージを出力する簡単なプログラムである。これをユーザの作成したアプリケーションプログラムに置き換えればよい。

アプリケーションプログラムには、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。

5. 実機でのプログラム実行

プログラムを実機上で実行する方法を、E² studio と JTAG エミュレータ J-Link（Segger Microcontroller Systems 製）を使用した例で説明する。
E² studio の開発環境から J-Link を使用し、実機に実行コードを転送しデバッグを行う。
実機には J-Link と接続するための JTAG インタフェースが必要となる。

5.1 SEGGER J-Link Software のインストール

(1) SEGGER J-Link Software を次の Web サイトからダウンロードする。

SEGGER https://www.segger.com/

サイトの「Download」→「J-Link/J-Trace」を選択し、 J-Link Software and Documentation Pack をダウンロードする。以下のインストーラがダウンロードされる（バージョンは変更される可能性がある）。

JLink_Windows_V656a.exe

(2) ダウンロードしたインストーラを実行し、SEGGER J-Link Software をインストールする。

5.2 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。すでにデバッグ構成が生成されている場合も以降の設定を行う。

(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
プロジェクト：前項で作成したプロジェクトを指定
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」に「J-Link ARM」を選択

「Target Device」に「R7S921053」を選択

「Startup」タブ

「ブレークポイント設定先」に「usermain」を入力

(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。

プログラムは実行すると、OS 起動後にユーザのアプリケーションプログラムを実行し、usermain 関数にてブレークする。

5.3 外部 FLASH ROM からのプログラムのブート

プログラムは最終的には外部メモリ等からブートされる。RZ/A2M 用 IoT-Engine ではブート用に外部 FLASH ROM を搭載している。

外部 FLASH ROM へのプログラムの書込みは、前述の、E² studio と J-Link で同様に行うことができる。

外部 FLASH ROM からのブートプログラムは以下である。

kernel¥sysdepend¥cpu¥rza2m¥sf_boot.S

また、外部 FLASH ROM からのブートの際に使用するリンカファイルは以下である。

etc¥linker¥iote_rza2m¥tkernel_rom_map.ld

以上