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

Version.01.00.04

2022.06.30

版数(日付)	内容
1.00.04 (2022.06.30)	● 開発環境のバージョンの更新
1.00.03 (2021.05.17)	● 開発環境のバージョンの更新 ● 全体の見直しおよび変更
1.00.02 (2020.10.21)	● 4.1 E² studioによるプログラムの実行 サンプルのファイル名変更 (旧)usermain.c (新)app_main.c
1.00.01 (2020.05.29)	● 3. 構築手順 Eclipseに加えてMakeによる構築手順を記載 ● 開発ツールのバージョン更新 ● その他（文章の直し、誤字修正など）
1.00.00 (2020.03.13)	● 初版

目次

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 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	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。

rx-elf-gcc.exe (GCC_Build_20210528) 8.3.0.202102-GNURX 20190222

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。

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。

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=rx230 -misa=v2 -mlittle-endian-data
-ffunction-sections -fdata-sections
```

リンクオプション

```
-mcpu=rx230 -misa=v2 -mlittle-endian-data
-ffunction-sections -fdata-sections -nostartfiles -nostdlib
```

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. 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とそれを用いた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 開発ソフトのインストール

C コンパイラなど共通の開発ツールをインストールする。これらは Eclipse からも使用される。

(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入手し、Web ページの指示に従いインストールする。

Open Source Tools for RENESAS
https://llvm-gcc-renesas.com/ja/rx-download-toolchains/

本稿作成時に検証したバージョンは以下の通り。
GCC for Renesas 8.3.0.202102-GNURX Toolchain

(2) 開発ツールのインストール

開発ツール一式（make など）を以下から入手し、Web ページの指示に従いインストールする。

xPack Windows Build Tools
https://xpack.github.io/windows-build-tools/

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 v4.3.0-1

(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（rx-elf-gcc）および make コマンドが実行可能であることを確認する。

3.1.2　ビルド環境の準備

(1) 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_m367.mk	M367 IoT-Engine 用のビルド規則(※)
iote_rx231.mk	RX231 IoT-Engine 用のビルド規則
iote_stm32l4.mk	STM32L4 IoT-Engine 用のビルド規則(※)
/mtkernel_3	Make 作業用ディレクトリ

※ 本書の説明では使用しない。

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 M367 IoT-Engine の場合は「**_IOTE_RX231_**」に変更する

また、iote_rx231.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	rx-elf-gcc	C コンパイラのコマンド名
AS	rx-elf-gcc	アセンブラのコマンド名
LINK	rx-elf-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_rx231.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。

3.1.3 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。

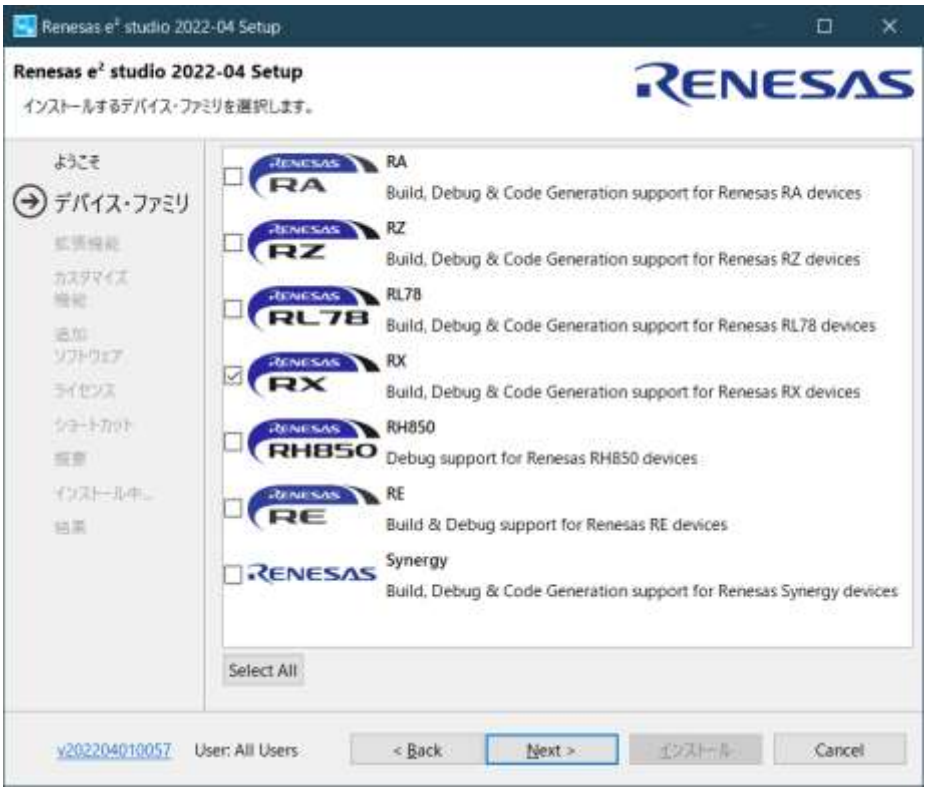
本ソフトの動作検証には e² studio の以下のバージョンを使用した。

e² studio 2022-04 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 RX 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

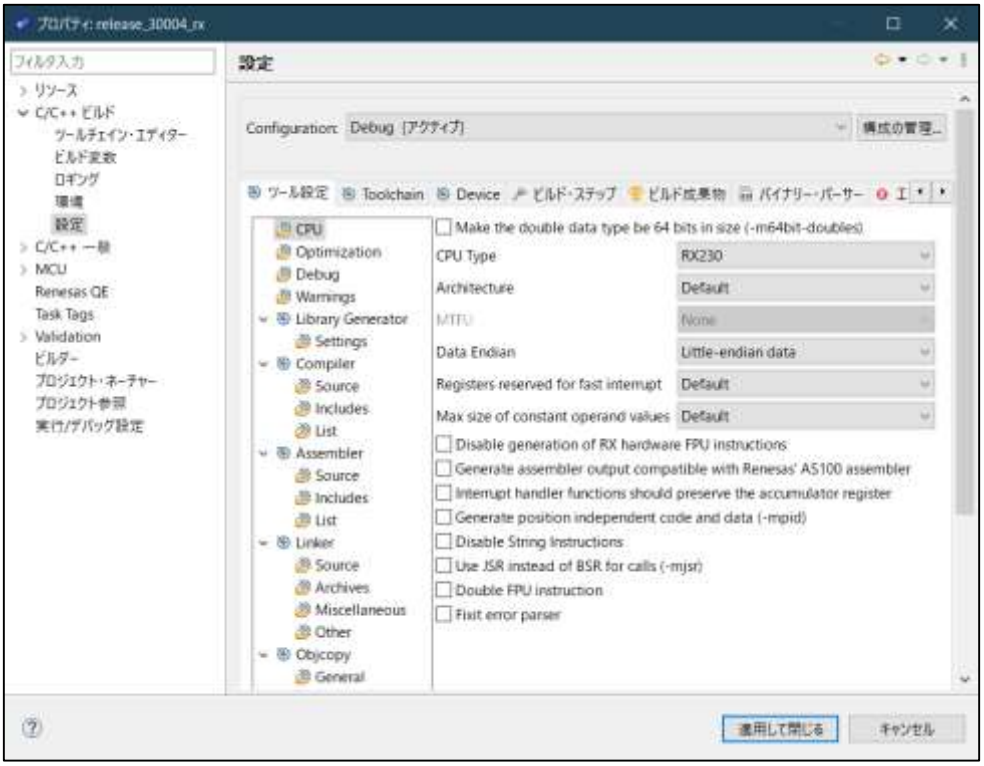
E² studio にて以下の手順で本ソフトのプロジェクトを作成する。

(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「C Managed Build」を選択する。
次の「C プロジェクト」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
・プロジェクトの種類：「実行可能」→「空のプロジェクト」
・ツールチェーン：「GCC for Renesas RX」

(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。



「CPU」

「CPU Type」に「RX230」を選択

「Optinmization」

「Optimaization Level」は任意

オプションは「-ffunction-sections」と「-fdata-sections」のみ選択

「Compiler」→「Source」

「Language standard」に「GNU ISO C11 (-std=gnu11)」を選択

「Compiler」→「Includes」

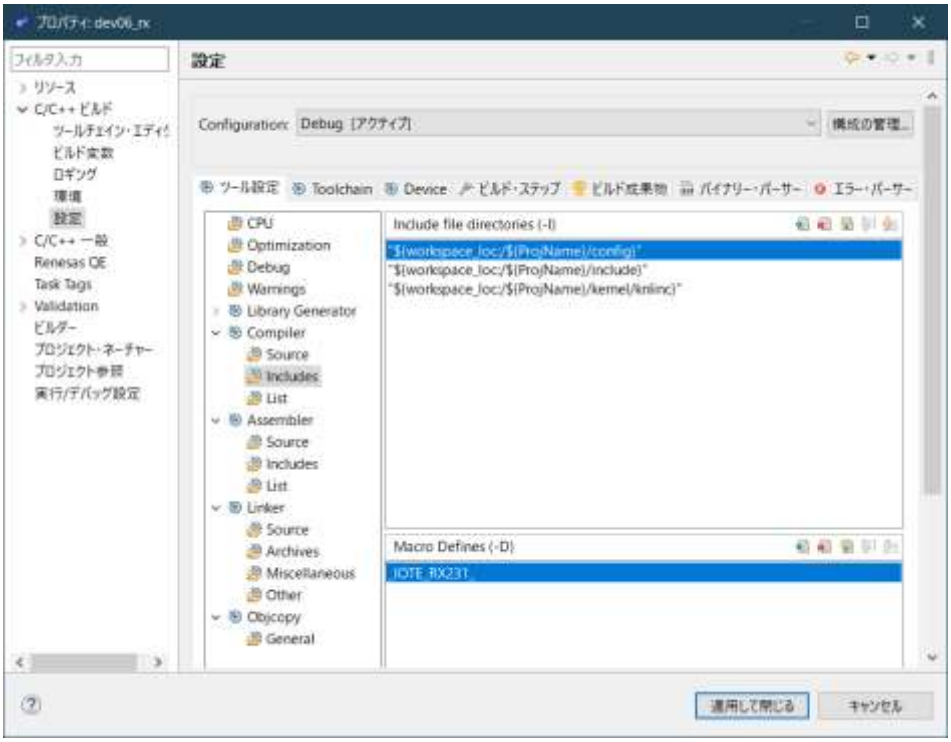
「Include file directories」にインクルードパスの追加

μT-Kernel3.0のインクルードパスを設定する。

「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」

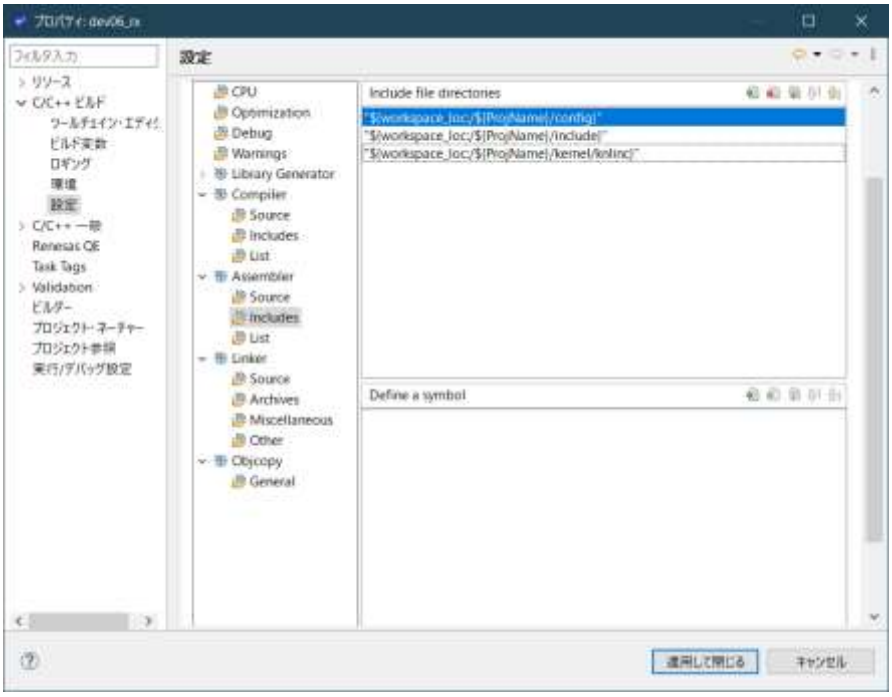
「User defined option」にターゲット名を定義

-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」

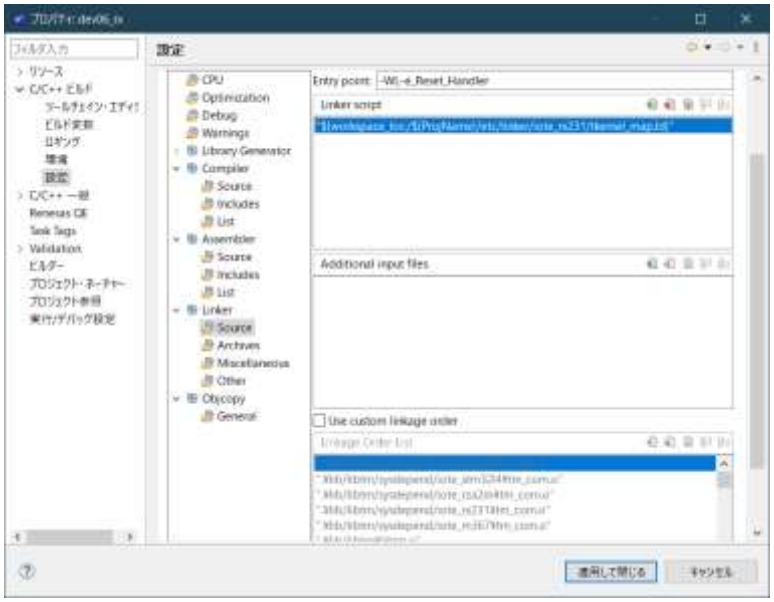
「Entry point」に以下を設定

-WI,-e_Reset_Handler

「Linker script」に、スクリプト・ファイルのパスを設定する。

スクリプト・ファイルは以下にある。

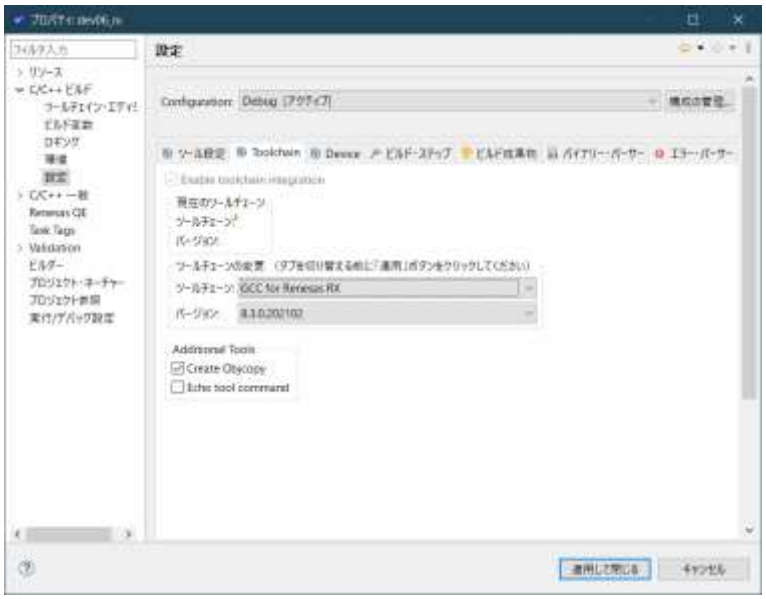
tkermnel_3¥etc¥linker¥iote_rx231¥tkernel_map.ld



(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 for Renesas RX」を選択する。

「バージョン」: 任意



全ての設定が完了したら「適用して閉じる」ボタンを押下する。

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。

5. 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、実機に RX231 IoT-Engine Arduino Evaluation Kit を使用する場合を例に説明する。
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。

5.1 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

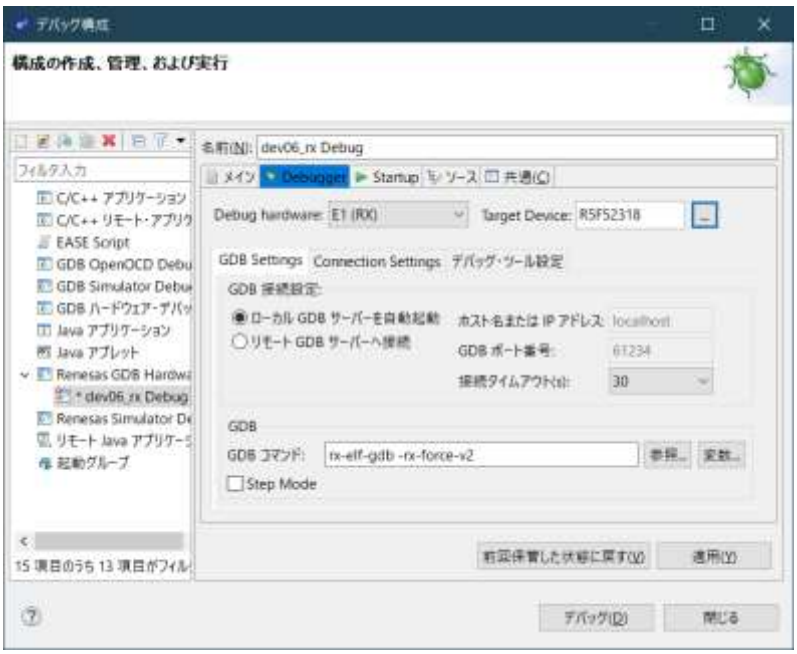
C/C++アプリケーション：ビルドした ELF ファイル

以下、エミュレータに E1 を使用する場合

「Debugger」タブ → 「GDB Settings」

Debug Hardware：「E1 (RX)」を選択 (E1 を使用する場合)

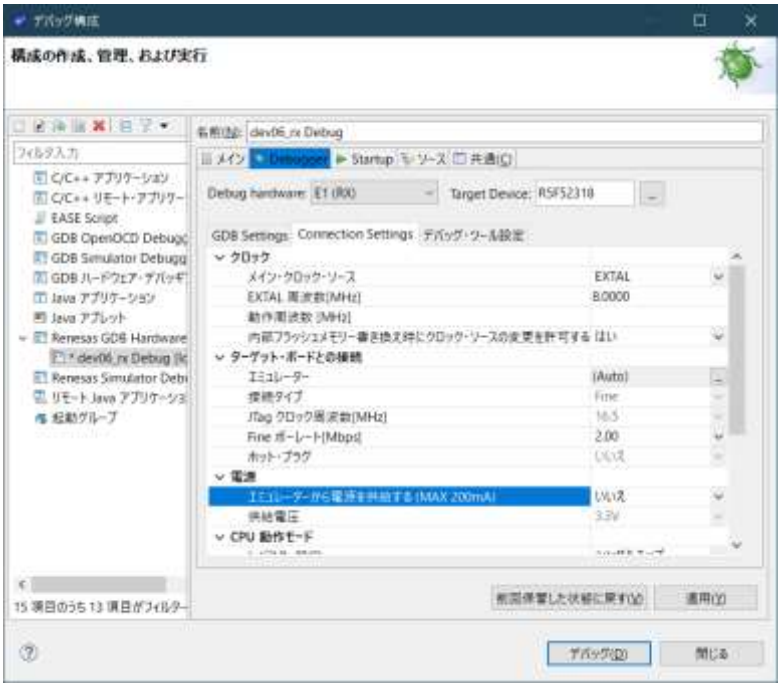
Target Deveice：「**R5F52318**」を選択



「Debbuger」タブ→　「Connection Setting」

「クロック」→「EXTAL 周波数」8.0000 を入力

「電源」→「エミュレータから電源を供給する」いいえを選択



「Startup」タブ

ブレークポイント設定先：「user_main」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、user_main 関数にてブレークする。

以上