μT-Kernel3.0
STM32L4 IoT-Engine 向け
構築手順書書

Version.01.00.02

2022.06.30

更新履歴

版数(日付)	内　容
1.00.02 (2022.06.30)	● 開発環境のバージョンの更新 ● Eclipse ベースの開発環境を Preiades All in One) から STM32CubeIDE に変更
1.00.01 (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
1.00.00 (2020.12.09)	● 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 STM32L4 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	STM32L4 IoT-Engine	UC テクノロジー製
搭載マイコン	STM32L486VG	ST マイクロ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。

arm-none-eabi-gcc (xPack GNU Arm Embedded GCC x86_64) 11.2.1 20220111

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、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-m4 -mthumb -ffreestanding -std=gnu11
```

リンクオプション

```
-mcpu=cortex-m4 -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、統合開発環境 STM32CCudeIDE 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。

The xPack GNU Arm Embedded GCC
https://xpack.github.io/arm-none-eabi-gcc/

本稿作成時に検証したバージョンは以下の通り。

xpack-arm-none-eabi-gcc-11.2.1-1.2

(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（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。

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 STM32L4 IoT-Engine の場合は「**_IOTE_STM32L4_**」に変更する

また、iote_stm32l4.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_stm32l4.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 STM32CubeIDE を使用した構築手順

3.2.1 STM32CudeIDE の準備

STM32CudeIDE は、オープンソースの"Eclipse"をベースとした、ST マイクロエレクトロニクス製マイコン用の統合開発環境である。
以下から使用する PC の OS に合わせて、STM32CudeIDE をダウンロードする。

本ソフトの動作検証には STM32CudeIDE の以下のバージョンを使用した。
STM32CudeIDE v.1.9.0

STM32CudeIDE のダウンロードページ
https://www.st.com/ja/development-tools/stm32cubeide.html

インストーラがダウンロードされるので、それを実行し、指示に従ってインストールを進める。
STM32CudeIDE 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。

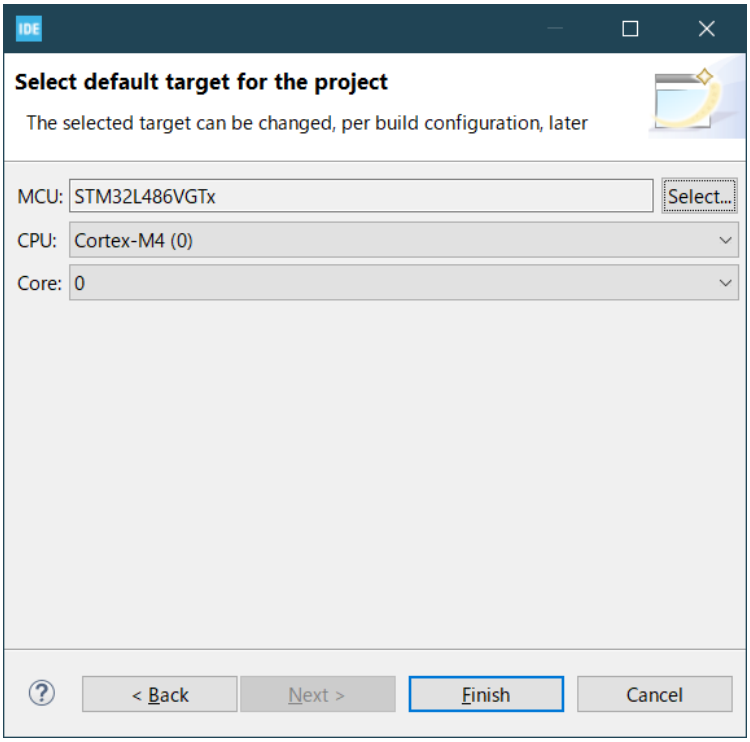
3.2.2 プロジェクトの作成

STM32CudeIDE 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) STM32CudeIDE 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、STM32CudeIDE の各種設定などが保存される可能的な作業場である。

(2) メニュー「New」→「C/C++ Project」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。
・Project name：任意
・Location：任意
・Project type：「Empty Project」選択
・Toolchains：「MCU ARM GCC」選択

「Next」ボタンを押下し進め、MCU には「STM32L486VGTx」を選択し、最後に「Finish」ボタンを押下するとプロジェクトが生成される。

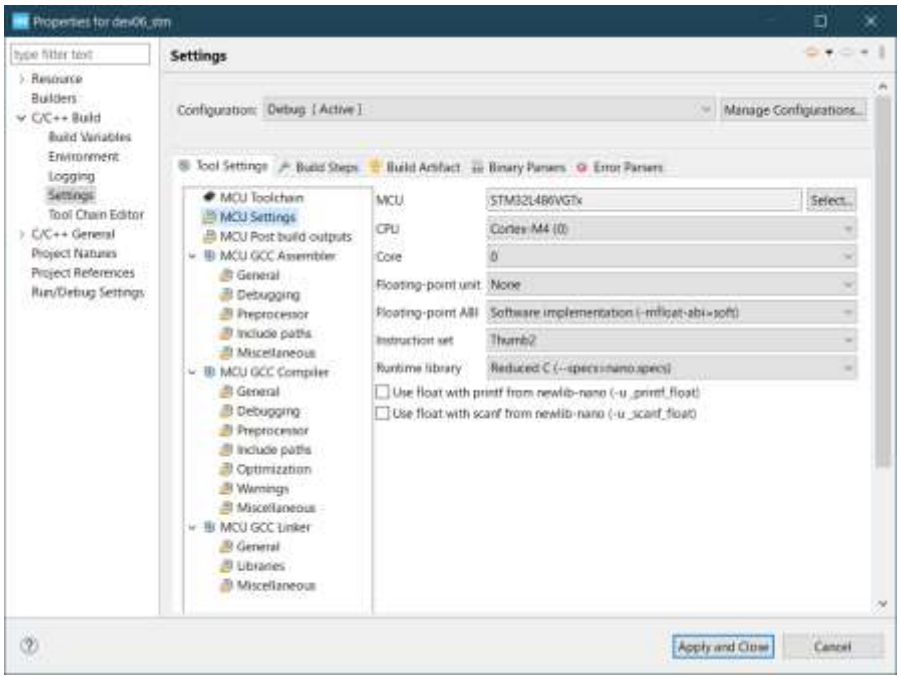


(3) 作成したプロジェクトを選択した状態でメニュー「File」→「Import...」を選択する。
開いた選択画面で「General」→「File System」を選択し、ファイルシステム画面でμT-Kernel3.0のソースコードのディレクトリを入力する。
なお、(1)でプロジェクトのロケーションに、既にソースコードのディレクトリが存在するディレクトリを指定した場合は、インポートは不要である。

(4) 作成したプロジェクトを選択した状態でメニュー「Project」→「Properties」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。

(5) ダイアログの項目「C/C++ Build」→「Settings」を選択し、「Tool Setteings」タブを開き、以降の手順に従って設定を行う。

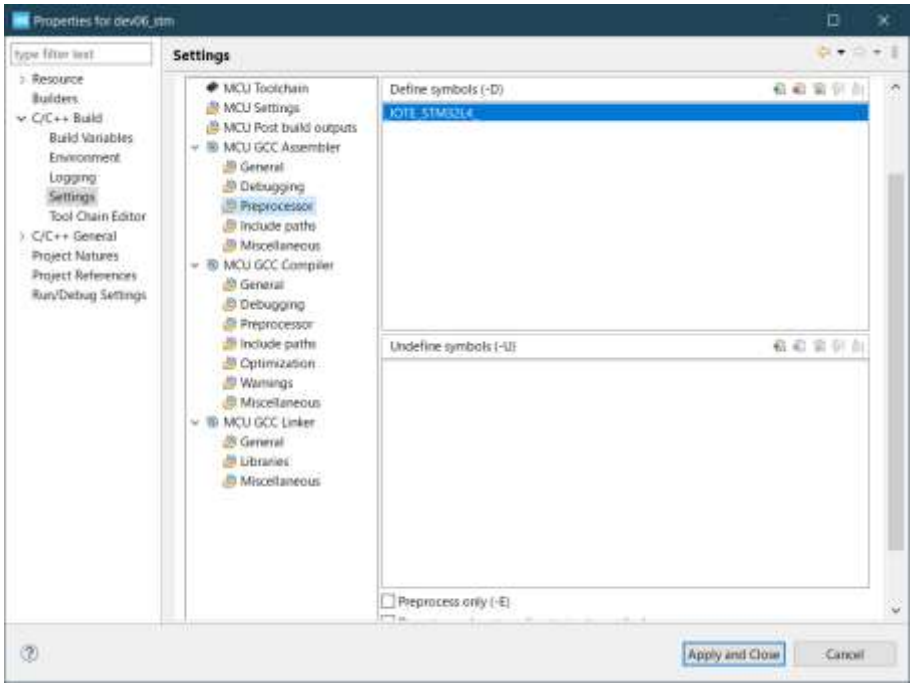
「MCU Settings」:
　　「MCU」:　「STM32L486VGTx」を確認
　　「Floating-point unit」: 任意（使用しない場合は「None」）



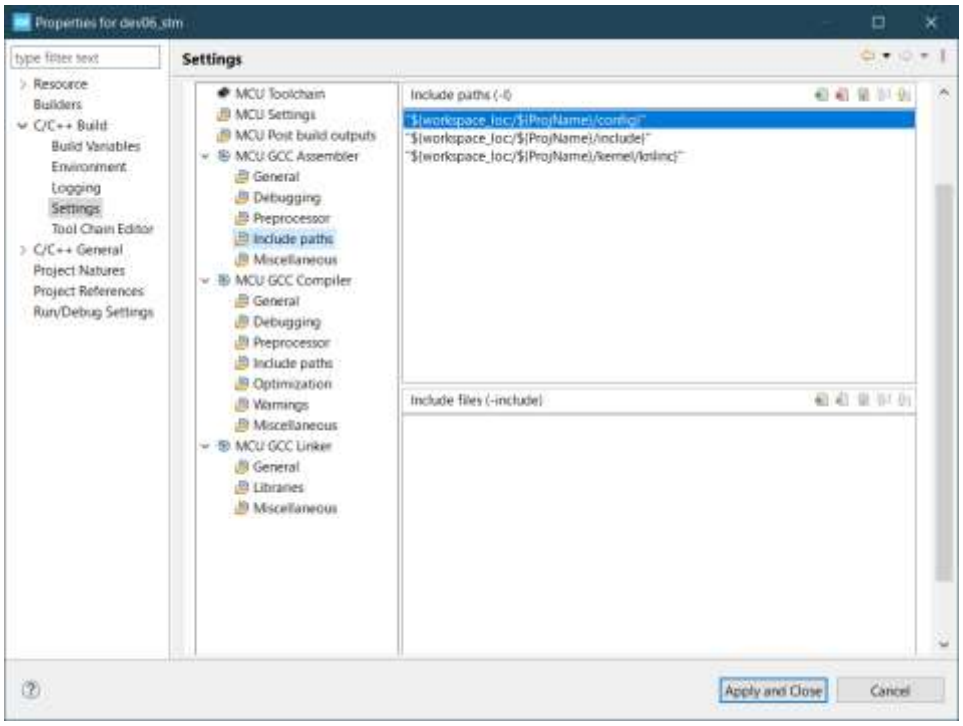
「MCU GCC Assembler」:

「Preprocessor」:

「Dedine symbols (-D)」:「**_IOTE_ST32L4_**」



「Include paths」: μT-Kernel3.0 のインクルードパスを設定



「MCU GCC Compiler」:

「General」:

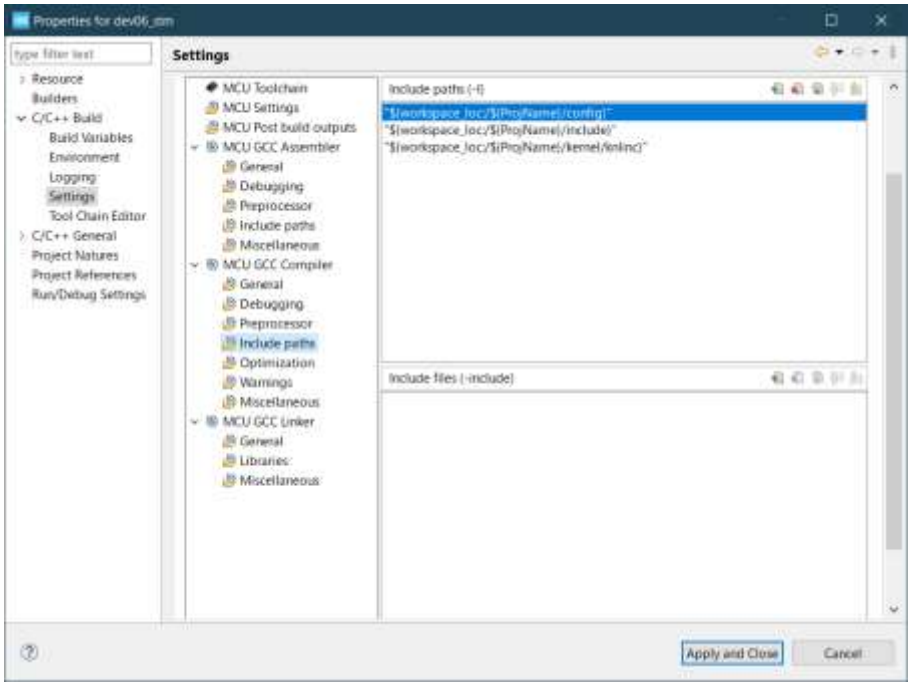
「Language standard」:「GNU11 (-std=gnu11)」

「Preprocessor」:

「Define symbols (-D)」: 「**_IOTE_STM32L4_**」

「Include paths」:

「Include paths」: μT-Kernel3.0 のインクルードパスを設定



「Optimization」:

「Optimaizationlevel」: 任意

他はチェックせず

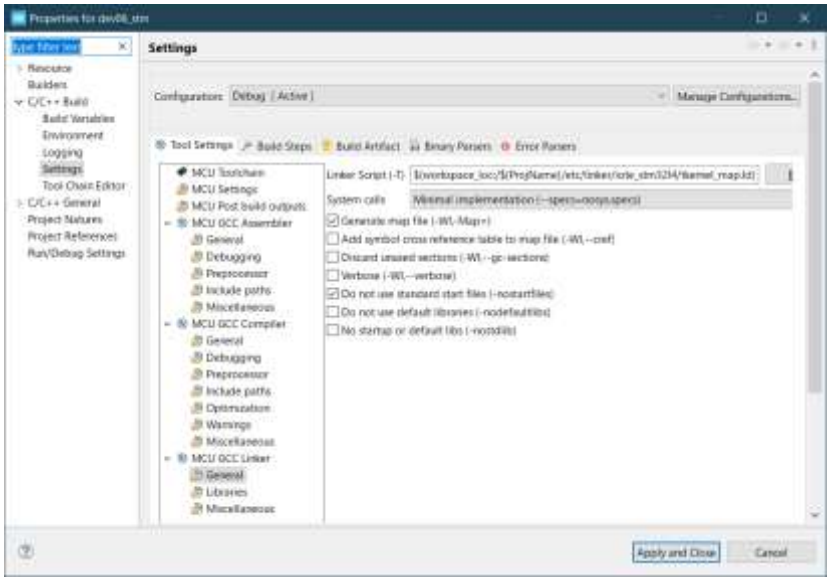
「MCU GCC Linker」:

「General」:

「Linker Script (-T)」: 以下のスクリプト・ファイルを指定

mtkernel_3¥etc¥linker¥iote_stm32l4¥tkernel_map.ld

「Do not use standard start files (-nostartfiles)」をチェック



すべての設定を終えたら「Apply and Close」ボタンを押下し設定を終える。

3.2.3　プロジェクトのビルド

作成したプロジェクトを選択した状態でメニュー「Projects」→「Build Project」を選択すると、μT-Kernel 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、STM32CuleIDE 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。
STM32CuleIDE 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 STM32CuleIDE によるプログラムの実行

(1) STM32CuleIDE のメニューからメニュー「Run」→「Debug configurations」を選択し、開いたダイアログから項目「STM32 Cortex-M C/C++ Application」を選択する。

(2) 「New」ボタンを押し構成を追加する。

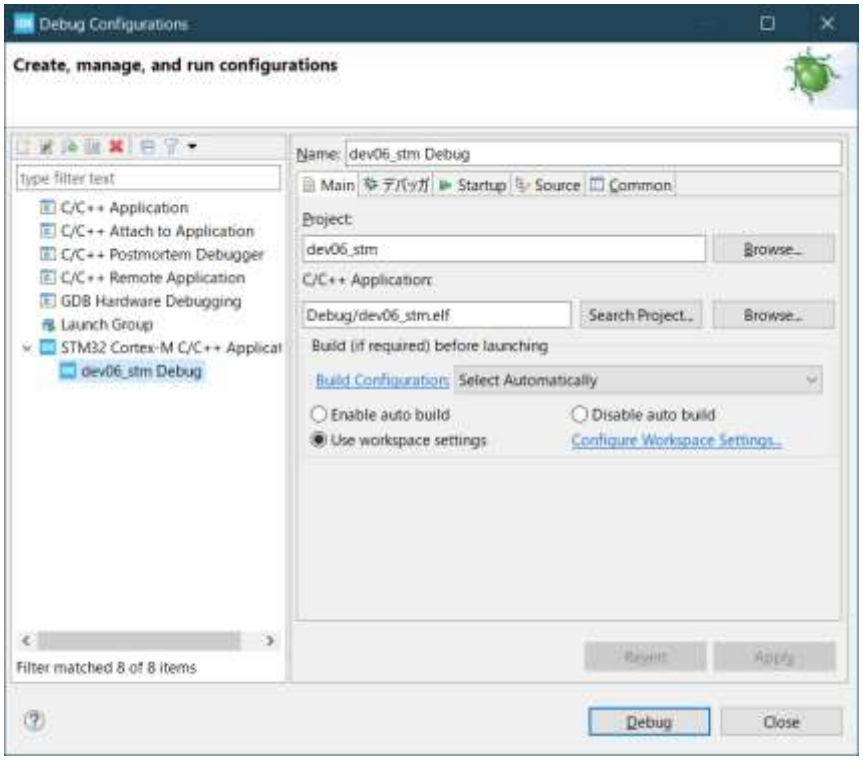
(3) 追加した構成を選択し、「構成の作成、管理、実行」画面にて以下の設定を行う。

「Main」タブ

Name：任意

Project：前項で作成したプロジェクトを指定

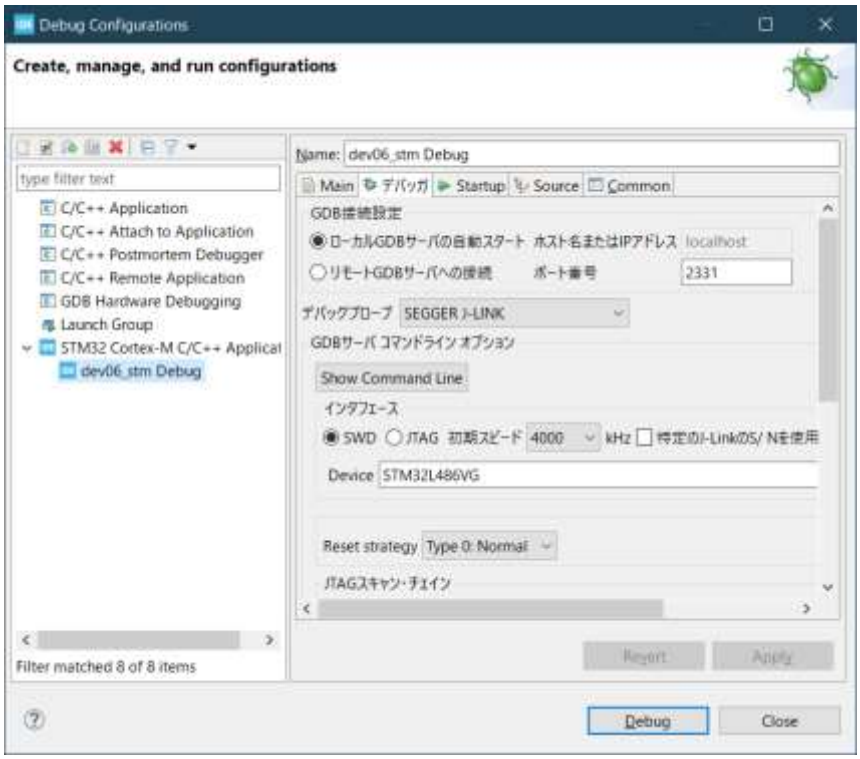
C/C++ Application：ビルドした ELF ファイル



「デバッガー」タブ

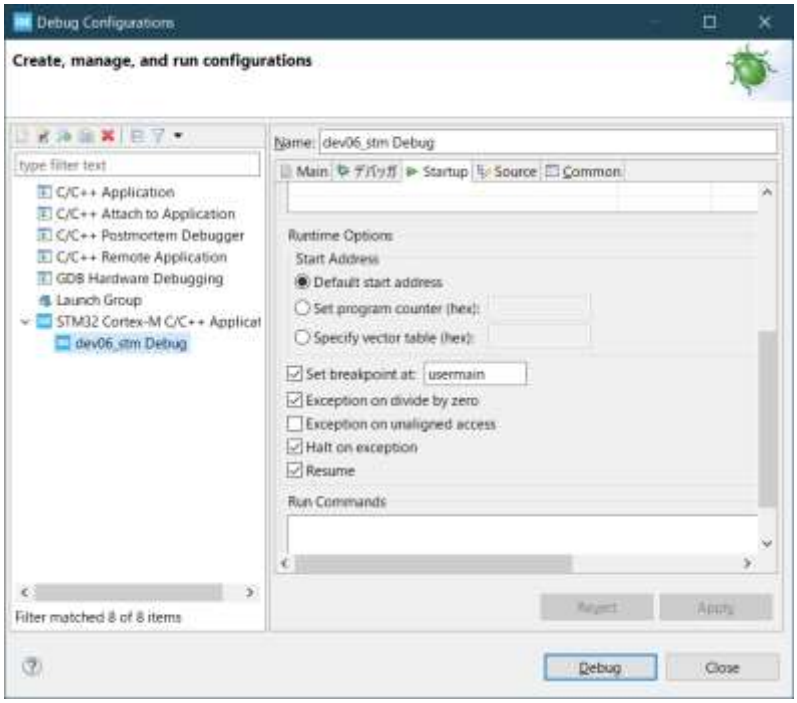
デバッグプローブ：「SEGGER J-LINK」を選択

Device：「**STM32L486VG**」を入力



「Startup」タブ

Set breakpoint at: 「usermain」を入力



(4) デバッグ開始

「Debig」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。

以上