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

Version.01.00.00

2020.12.09

更新履歴

| 版数(日付) | 内　容 |
|---|---|
| 1.00.00<br>(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-Kernel3.00.03 | 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。

**GNU Arm Embedded Toolchain**

(https://developer.arm.com/open-source/gnu-toolchain/gnu-rm)

**gcc-arm-none-eabi-9-2019-q4-major-win32**

## 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 -mfloat-abi=soft -ffreestanding
-fomit-frame-pointer -std=gnu11
```

**リンクオプション**

```
-mcpu=cortex-m4 -mthumb -mfloat-abi=soft  -ffreestanding
-fomit-frame-pointer -nostartfiles -static
```

## 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。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、オープンソースの統合開発環境 Eclipse を使用する場合と、自動ビルドツール Make 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 共通開発ソフトの準備

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 でも Make でも使用する。

(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ダウンロードする。

**GNU Arm Embedded Toolchain**
https://developer.arm.com/open-source/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。

**GNU Tools for Arm Embedded Processors 9-2019q4-major Release**

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。

**GNU MCU Eclipse Windows Build Tools**
https://github.com/gnu-mcu-eclipse/windows-build-tools/releases

本稿作成時に検証したバージョンは以下の通り。
**GNU MCU Eclipse Windows Build Tools v2.12 20190422**

ダウンロードした zip ファイルを任意の場所に展開する。

## 3.2 Eclipse を使用した構築手順

### 3.2.1 Eclipse の準備

(1) Eclipese のインストール

Eclipse と CDT および Eclipse を実行するための Java 実行環境を準備する。
Eclipse 本体と日本語化プラグイン、その他必要なソフトをまとめたオールインワン・パッケージ Pleiades All in One は以下からダウンロード可能である。

**MergeDoc Project** https://mergedoc.osdn.jp/

本稿作成時に検証したバージョンは以下の通り。

**Pleiades All in One リリース 2020-03 Full Edition**
pleiades-2020-03-cpp-win-64bit-jre_20200322.zip

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) ワークスペースの作成

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

(3) GNU ARM Eclipse Plug-in の追加

Eclipse に ARM マイコン開発のためのプラグインを追加する。
メニュー「ヘルプ」→「新規ソフトウェアのインストール」→「追加」を選択する。
開いたダイアログに以下を入力

| | |
|---|---|
| **Name :** | **GNU ARM Eclipse Plug-ins** |
| **Location:** | **http://gnuarmeclipse.sourceforge.net/updates** |

以降、画面の指示に従って「GNU ARM C/C++ Cross Development Tools」をすべてインストールする。

### 3.2.2 プロジェクトの作成

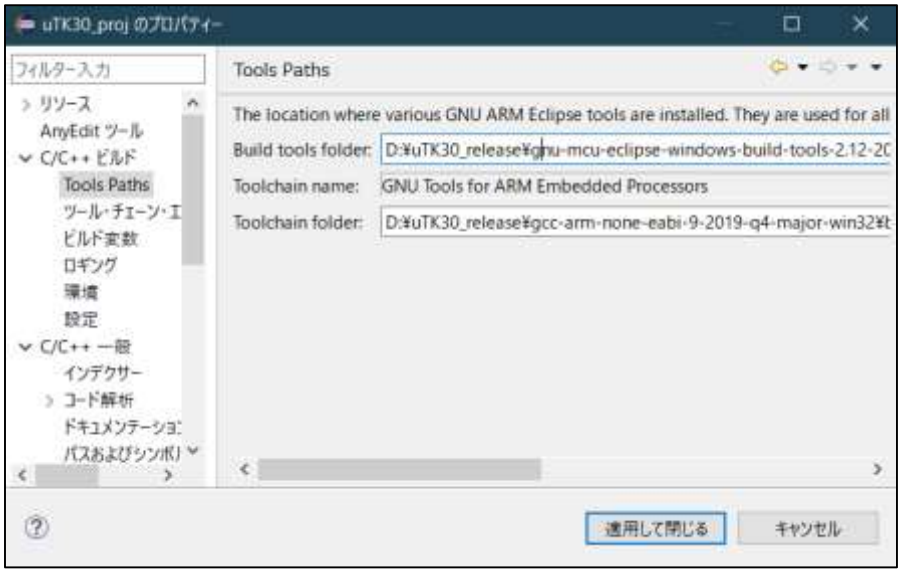
Eclipse にて以下の手順で本ソフトのプロジェクトを作成する。

(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 管理ビルド」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
・プロジェクトタイプ：「空のプルジェクト」選択
・ツールチェーン：「Cross ARM GCC」選択

以降、指示に従って設定を行う。「Cross GNU ARM Toolchain」の「Toolchain path」には、GCC を展開したディレクトリ内の¥bin ディレクトリのパスを設定する。

(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。

(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」→「Tools Paths」を選択し、Build tools folder に GNU MCU Eclipse 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設定する。
Toolchain folder には、 GCC を展開したディレクトリ内の¥bin ディレクトリのパスを設定する。



(4) 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。

「Target Processir」

「ARM family」を「cortex-m4」とする

「Float ABI」を「ライブラリ(soft)」とする

「最適化」

「最適化レベル」は任意

オプションは 「Assume freestanding environment (-ffreestanding)」のみを選択

「Cross ARM GNU Assembler」

「プリプロセッサ」の「定義済みのシンボル(-D)」にターゲット名を定義。

_IOTE_STM32L4_

「インクルード」インクルードパスの追加

「2.3 インクルードパス」を参照

「Cross ARM GNU C Compiler」

「プリプロセッサ」の「定義済みのシンボル(-D)」にターゲット名を定義。

_IOTE_STM32L4_

「インクルード」インクルードパスの追加

「2.3 インクルードパス」を参照

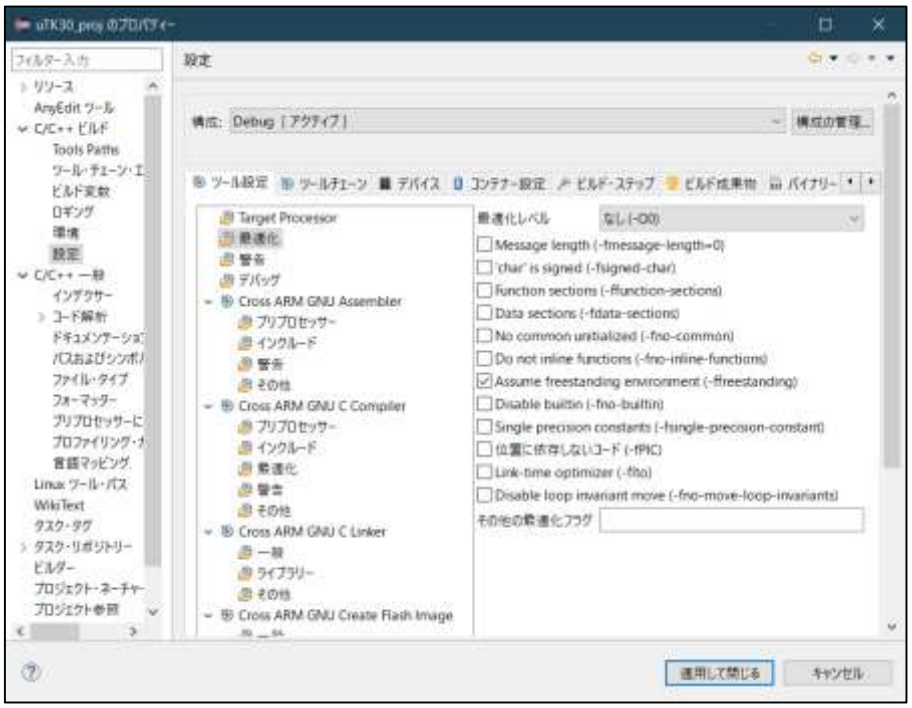
「最適化」の「言語標準」で「GNU ISO C11 (-std=gnu11)」を選択

「Cross ARM GNU C Linker」

「一般」スクリプト・ファイルに、以下のファイルのパスを設定する。

etc¥linker¥iote_stm32l4¥tkernel_map.ld

標準開始ファイルを使用しない(-nostartfile)のみを選択する。




### 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、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。

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。
デフォルトの設定では Release と Debug が選択可能である。

# 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、実機に STM32L4 IoT-Engine Starter Kit を使用する場合を例に説明する。
本 Kit に付属の JTAG エミュレータ J-Link を使用し、前章で説明した Eclipse 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。

## 4.1 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。

(3) Eclipse のメニュー「ウィンドウ」→「設定」を選択し、開いたダイアログの項目「実行/デバッグ」→「SEGGER J-Link」のフォルダーに、(2)でインストールした「SEGGER J-Link Software」のディレクトリを設定する。


## 4.2 Eclipse によるプログラムの実行

(1) Eclipse 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「GDB SEGGER J-Link Debugging」を選択する。

(2) 「新規構成」ボタンを押し、「GDB SEGGER J-Link 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

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

「デバッガー」タブ

デバイス名：「**STM32L486VG**」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。

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。
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

# 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。

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