第1章 EclipseからPico-sdkやTinyUSBを使って ビルド&デバッグ！FreeRTOSプログラムの作成方法も

Pico用USBプログラミング環境の構築

関本 健太郎

この章では，第2部 第2章以降の活用例として紹介するプログラムを作成するための環境の構築方法を説明していきます．開発のフレームワークとしてEclipseを，OSはWindows 10をメインで使用します．

ステップ1：Pico関連のライブラリの準備

Raspberry Pi財団が提供するラズベリー・パイPico（以降，Pico）関連のソフトウェアには，主なところ でpico-sdk，pico-examples，pico-extras，pico-playgroundがあります．

● pico-sdk…アプリケーション作成

pico-sdkは，RP2040向けのアプリケーション作成のためのヘッダ・ファイル，ライブラリ，CMakeを利用したビルド・システムのフレームワークを提供しています．この中でTinyUSBライブラリがサブモジュールとして含まれています．

● pico-examples…各種周辺機能のサンプル

pico-examplesには，pico-sdkを利用したRP2040の各種周辺機能に関連するサンプル・プログラムが多数含まれています．

● pico-extras…追加ライブラリ

pico-extrasには，pico-sdkに含める前の段階の追加ライブラリ［オーディオ（I²S）サンプル，VGA/DPIなどのサンプル］が含まれています．

● pico-playground…pico-examples同様のサンプル

pico-playgroundは，pico-examplesと同様のサンプル・プログラム集ですが，追加ライブラリとしてpico-extrasを利用するプログラムを含んでいます．

＊

これらはCMakeによるビルドが前提になっています．本記事では，pico-sdkのみ利用するので，適当なフォルダを作成し，pico-sdkリポジトリをクローンしておきます（**リスト1**）．

ステップ2：TinyUSBライブラリの設定

pico-sdkには，gitサブモジュールとして，libフォルダ下にTinyUSBライブラリがインストールされます．この設定は前節のPico関連のライブラリを中心に開発していく場合には便利ですが，基本的にTinyUSBライブラリのgitリポジトリの特定のコミットを前提としているため，pico-sdkで使われるTinyUSBのコミットよりも新しいコミットを利用したり，TinyUSBのバグを修正したりする場合には管理が複雑になります．そこで，独立してTinyUSBのgitリポジトリをクローンし，TinyUSBからpico-sdkのライブラリを参照するように設定します．

ここでは，pico_projectというgitリポジトリを作成し，そのサブモジュールとしてTinyUSBライブラリを設定します（**リスト2**）．

TinyUSBライブラリには多くのマイコンのライブラリがgitサブモジュールとして登録されています．これらをまとめてgitサブモジュールとして初期化すると，余分な時間とディスク領域を確保することになるので，後述のビルド・スクリプトでは必要なサブモジュールのみ初期化しています．

リスト1　pico-sdkリポジトリをクローンするコマンド

```
mkdir c:\pico
cd pico
git clone https://github.com/raspberrypi/pico-sdk.
                                                  git
git clone https://github.com/raspberrypi/pico-
                                            examples.git
git clone https://github.com/raspberrypi/pico-
                                                extras.git
cd pico-sdk
git submodule update --init
// tinyusbサブモジュールを初期化する
```

リスト2　プロジェクトのgitリポジトリを作成するコマンド

```
cd c:¥pico
git init pico_projects
cd pico_projects
git submodule add https://github.com/hathach/
                                tinyusb.git lib/tinyusb
```

リスト3　Windows MSYS2-Mingw64環境でPicoのUSBプログラミングに必要なパッケージをインストールするコマンド

```
pacman -Syu
pacman -S base-devel mingw-w64-x86_64-toolchain
mingw-w64-x86_64-arm-none-eabi-toolchain autoconf
automake gcc libtool zlib git mingw-w64-x86_64-
cmake mingw-w64-x86_64-doxygen mingw-w64-x86_64-
ninja
```

ステップ3：Eclipse環境の構築

RP2040の開発環境としては，Visual Studio CodeあるはArduino IDEなども利用できますが，Eclipse環境は複数のプロジェクトをProject Explorerで一望できるので，筆者の個人的な感想としては，本特集のように複数のプロジェクトを利用した開発環境には便利ではないかと思っています．

Windows 10のMSYS2（Mingw64またはMingw32）環境で，Eclipse CDTを利用します．以下，Mingw64環境を前提としています．Mingw32環境の場合は，インストールするパッケージを変更してください．Eclipseにはマイコンのレジスタを表示するEmbSysRegViewというプラグインがあります．RP2040マイコン用のsvdファイルが用意されているので，Eclipse Embedded C/C++の他に，そのプラグインも追加します．

● Windows 10環境

以下の手順でEclipse環境を構築します．

1. MSYS2-Mingw64のインストーラをダウンロードし，実行します．https://www.msys2.org/にアクセスし，インストーラ（例：msys2-x86_64-20220319.exe）をダウンロードし，実行します．MSYS2-Mingw64 shellを起動し，パッケージをインストールします（**リスト3**）．
2. Eclipse CDT（ZIPファイル）をダウンロードし，適当なフォルダにインストールします．https://www.eclipse.org/cdt/downloads.phpにアクセスし，最新のパッケージをダウンロードします．
3. Eclipseを起動し，マーケット・プレイスより，幾つかのプラグインをインストールします．［help］→［Eclipse Marketplace］メニューを起動し，Eclipse Embedded C/C++およびEmbSysRegViewをインストールします．
4. pico-sdkリポジトリよりrp2040.svdファイルをEclipseのpluginフォルダにコピーします．
   コピー元：pico-sdk¥src¥rp2040¥hardware_regs¥rp2040.svd
   コピー先：¥eclipse¥plugins¥org.eclipse.cdt.embsysregview.data_0.2.6.r191¥data¥SVD（CMSIS）¥raspberrypi¥rp2040_svd.xml　（rp2040.svdのファイル名をrp2040_svd.xmlに変更してコピー）
5. Eclipseの［Window］-［Preference］メニューから［C/C++］-［Debug］-［EmbSys Register View］ダイアログを開き，RP2040を設定しておきます（**図1**）．

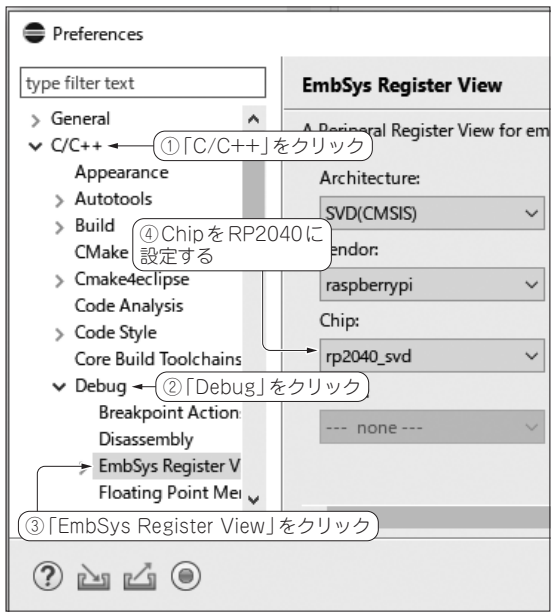

図1　EmbSys Register Viewのダイアログで RP2040と設定する

この設定によってPicoprobe + OpenOCDでデバッグしたときに，RP2040のレジスタ値を参照できるようになります（**図2**）．

● Ubuntu 22.04環境

本特集では，Linux環境のみしか利用できない活用例がありますので，筆者が利用したLinuxの開発環境も説明します．Linuxディストリビューションは，Ubuntu 22.04を利用しました．

1. ビルド・ツールをインストールします（**リスト4**）．
2. Eclipse CDT（tar.gzファイル）をダウンロードし，適当なディレクトリにインストールします．以降はWindows 10環境と同じ手順で設定します．

ステップ4：プロジェクト・フォルダの設定

Windows 10のEclipse環境を前提とします．プロジェクト・フォルダは，Eclipseで用意されている

(x)= Variables　Breakpoints　Expressions　Modules　EmbSys Registers　Disassembly　Peripherals
Arch: SVD(CMSIS) Vendor: raspberrypi Chip: rp2040_svd Board: --- none ---

Register	Hex	Bin	Reset	Access	Address	Description
USBCTRL_REGS						USB FS/LS controller device registers
USBCTRL_REGS						USB FS/LS controller device registers
ADDR_ENDP	0x0000001C	00000000000000000000000000...	0x00000000	RO	0x50110000	Device address and endpoint control
ENDPOINT (bits 19-16)	0x0	0000				Device endpoint to send data to. Only valid for HOST ...
ADDRESS (bits 6-0)	0x1C	0011100				In device mode, the address that the device should re...
ADDR_ENDP1	0x00000000	00000000000000000000000000...	0x00000000	RO	0x50110004	Interrupt endpoint 1. Only valid for HOST mode.
ADDR_ENDP2	0x00000000	00000000000000000000000000...	0x00000000	RO	0x50110008	Interrupt endpoint 2. Only valid for HOST mode.
ADDR_ENDP3	0x00000000	00000000000000000000000000...	0x00000000	RO	0x5011000c	Interrupt endpoint 3. Only valid for HOST mode.
ADDR_ENDP4	0x00000000	00000000000000000000000000...	0x00000000	RO	0x50110010	Interrupt endpoint 4. Only valid for HOST mode.
ADDR_ENDP5	0x00000000	00000000000000000000000000...	0x00000000	RO	0x50110014	Interrupt endpoint 5. Only valid for HOST mode.

図2　RP2040用のsvdファイルを設定することでEclipse上でRP2040のレジスタ値を参照できる

CMake向けのプロジェクト・テンプレートであるEmpty or Existing Cmake Projectまたは Makefile Project with Existing Codeを利用します．

ただし，このテンプレートの利用はプロジェクト登録だけにとどめます．実際のプロジェクトのビルドには，gitのサブモジュールの構成，環境変数の設定など，追加パラメータ設定も必要になることから，CMakeの事前処理を行うシェル・スクリプトを利用することにしました．

● CMakeプロジェクト登録

Eclipseのメニュー・バーの［File］-［New］-［C/C++ Project］で開いたダイアログで，「CMake」→「Empty or Existing CMake Project」を選択します（**図3**）．「Use default location」のチェックを外し，［Browse］ボタンを押し，プロジェクト・フォルダを選択します．Project name欄に適当なプロジェクト名（例えばpico_projects）を設定し，［Finish］ボタンを押します．

同じようにpico-sdkフォルダもCMakeプロジェクトとして登録しておきます（**図4**）．作成したプロジェクト名がProject Explorerのウィンドウに表示されます．その後，PICO_SDK_PATH環境変数（仮にc:¥pico¥pico-sdkに設定）をWindows 10環境に登録します．

リスト4　Ubuntu 22.04環境でPicoのUSBプログラミングに必要なパッケージをインストールするコマンド

```
sudo apt update
sudo apt upgrade
sudo apt install build-essential texinfo libtool
automake autoconf git pkg-config cmake gcc-arm-
                                        none-eabi curl
sudo apt install gdb-multiarch
```

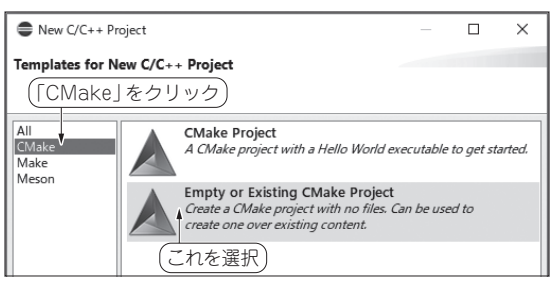

図3　Eclipseから「Empty or Existing CMake Project」を選択する

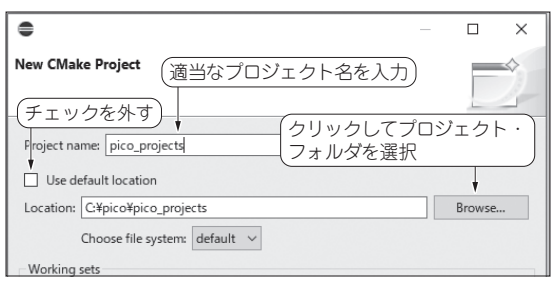

図4　CMake Projectの設定項目

リスト5　プロジェクトのビルド・スクリプト例（build.sh）

```
#!/bin/bash
set -eu -o pipefail
# git submodule update --init lib/FreeRTOS-Kernel
# git submodule update --init lib/tinyusb
cd ./lib/tinyusb
# git submodule update --init hw/mcu/raspberry_pi/
                                              Pico-PIO-USB
# git submodule update --init lib/lwip
cd ../..
export PICO_SDK_PATH="../pico-sdk"
# pico sdkのパスを適切に設定する
# 下記2行は、Mingw64環境でpico-sdkのツールをビルドするときの修正
cp -f ./pico-sdk/tools/elf2uf2/CMakeLists.txt ${PICO_SDK_
                                        PATH}/tools/elf2uf2/CMakeLists.txt
cp -f ./pico-sdk/tools/pioasm/CMakeLists.txt ${PICO_SDK_
                                        PATH}/tools/pioasm/CMakeLists.txt
export BUILD="build"
export BOARD="PICO"
export LOG_PATH="./_log"
mkdir -p ${LOG_PATH}
DT=`date +%Y%m%d%H%M`
make V=1 DEBUG=1 BOARD=${BOARD} clean 2>&1 | tee ${LOG_
                                        PATH}/${BOARD}_build_${DT}.log
make V=1 DEBUG=1 BOARD=${BOARD}       2>&1 | tee -a
                                ${LOG_PATH}/${BOARD}_build_${DT}.log
```

リスト6　プロジェクト・ルート・フォルダ上のCMakeLists.txt

```
cmake_minimum_required(VERSION 3.12)

set(FAMILIY RP2040 CACHE INTERNAL "")
set(FAMILIY_MCU RP2040 CACHE INTERNAL "")
set(CMAKE_BUILD_TYPE Debug)

include(pico_sdk_import.cmake)
include(pico_extras_import.cmake)

set(PROJECT_NAME "pico_projects" C CXX ASM)
set(CMAKE_C_STANDARD 11)
set(CMAKE_CXX_STANDARD 17)

project(${PROJECT_NAME})

pico_sdk_init()

add_subdirectory(projects/template)

set(TINYUSB_DIR ${PICO_SDK_PATH}/lib/tinyusb)
set(FREERTOS_DIR ${CMAKE_SOURCE_DIR}/freertos/
                                    FreeRTOS-Kernel)

add_subdirectory(projects/template)
```

リスト8　テンプレート・プロジェクトのmain関数例

```
#include "pico/stdlib.h"

int main() {
#ifndef PICO_DEFAULT_LED_PIN
#warning blink example requires a board with a
                                        regular LED
#else
    const uint LED_PIN = PICO_DEFAULT_LED_PIN;
    gpio_init(LED_PIN);
    gpio_set_dir(LED_PIN, GPIO_OUT);
    while (true) {
        gpio_put(LED_PIN, 1);
        sleep_ms(250);
        gpio_put(LED_PIN, 0);
        sleep_ms(250);
    }
#endif
}
```

● ビルド

build.shというビルド用のシェル・スクリプトを作成しました(**リスト5**)．Mingw64のコンソールを開き，Eclipseプロジェクトのルート・ディレクトリで./build.shと入力してシェル・スクリプトを実行します．スクリプト中のgitサブモジュールの初期化は必要に応じコメント・アウトを外して行ってください．

▶Eclipseのビルド・メニューによるビルド

Eclipseのビルド・メニューによるCMakeプロジェクトのビルドも可能です．その場合には，[Project]-[Build Project]メニューを実行することで開始します．ただし，このメニューを2度選択する必要があります．1度目の選択では，CMakeの設定ファイル群がbuildフォルダに作成されます．自動的にファイルの更新がされないようなので，buildフォルダを選択し，F5キーなどでフォルダ情報をリフレッシュし，もう一度Build Projectメニューを実行します．EclipseからMSYS-Mingw64環境のCMakeが呼び出され，ビルドが開始されます．

▶補足…エラー対応

Mingw64環境では，ビルド中に作成されるelf2uf2.exeやpioasm.exeで，elfファイルからuf2ファイルを変換する際にエラーが発生することがありました．その場合にはpico-sdk/tools/elf2uf2(またはpioasm)/CMakeLists.txtファイルに，set(CMAKE_EXE_LINKER_FLAGS "-static")を追加してみてください．

Build not configured correctlyが表示されてビルドできない場合には，プロジェクトを作成し直すことでビルドできるようになる場合があります．

リスト7　テンプレート・プロジェクトのフォルダ(template)上のCMakeLists.txt

```
set(APP_NAME "template")

add_executable(${APP_NAME})

target_sources(${APP_NAME} PUBLIC
        ${CMAKE_CURRENT_LIST_DIR}/main.c
)

target_include_directories(${APP_NAME} PUBLIC
        ${CMAKE_CURRENT_LIST_DIR}
)

target_link_libraries(${APP_NAME} pico_stdlib)

pico_add_extra_outputs(${APP_NAME})
```

ステップ5：Picoプログラムの新規作成

● CMakeLists.txtとmain.cを作成

Pico向けにpico-sdkを利用したプログラムを新規に作成する手順を説明します．まずは，適当なプロジェクト名でCMakeプロジェクトを作成します．フォルダ構造は任意で構いません．ここではまず，pico_projectsフォルダ下にCMakeLists.txt(**リスト6**)を作成します(**図5**)．同様にpico_projectsフォルダの下にprojectsフォルダを作成します．pico-sdkフォルダから，pico_sdk_import.cmakeをコピーし，pico_projectsフォルダの下にテンプレート・プロジェクトのフォルダ(template)を作成し，CMakeLists.txt(**リスト7**)ファイルとmain.c(**リスト8**)を作成します．

● Eclipseでビルドできるようになる

プロジェクト・ルート・フォルダ中のCMakeLists.

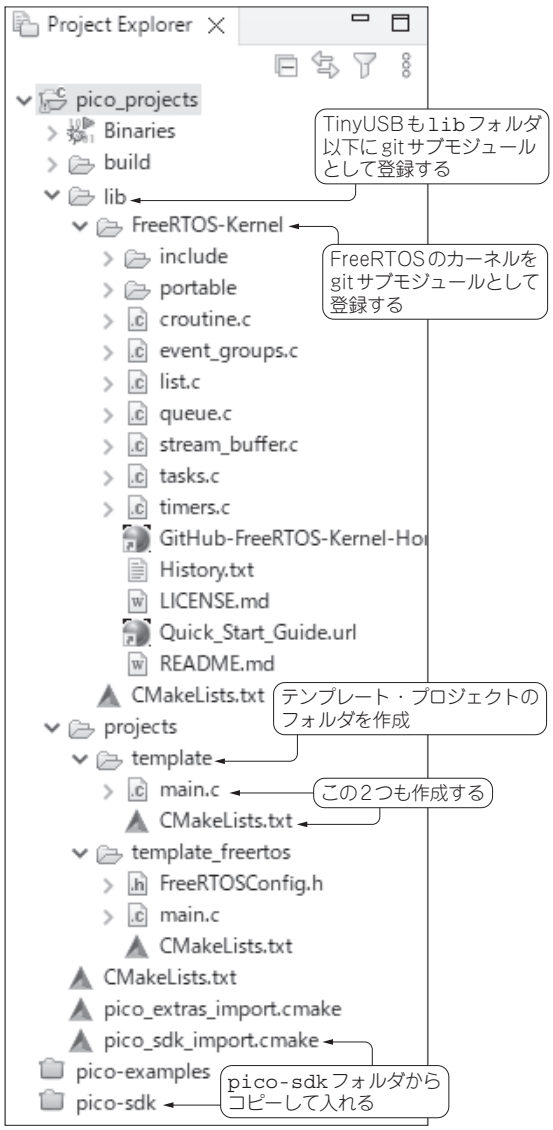

図5　テンプレート・プロジェクトのフォルダ中にCMakeLists.txtとmain.cを作成する

txtファイル(**リスト6**)中に，add_subdirectory(projects/template)と記述することで，pico-sdkを利用した最小限のプログラムを，Eclipseの[Project]-[Build Project]メニューでビルドできるようになります．

本章以降のサンプル・プログラムは，projectsフォルダ以下に追加していきます．

テンプレート・プロジェクトのフォルダには，アプリケーションのソース・ファイル(main関数を含むmain.c)，アプリケーションのインクルード・ディレクトリ・パスの定義，およびアプリケーション作成に必要なライブラリ情報などを含むCMakeLists.txtがあります(**リスト7**)．

リスト9　FreeRTOSカーネル・ライブラリのCMakeLists.txt

```
set(PICO_SDK_FREERTOS_SOURCE FreeRTOS-Kernel)

add_library(freertos
    ${PICO_SDK_FREERTOS_SOURCE}/event_groups.c
    ${PICO_SDK_FREERTOS_SOURCE}/list.c
    ${PICO_SDK_FREERTOS_SOURCE}/queue.c
    ${PICO_SDK_FREERTOS_SOURCE}/stream_buffer.c
    ${PICO_SDK_FREERTOS_SOURCE}/tasks.c
    ${PICO_SDK_FREERTOS_SOURCE}/timers.c
    ${PICO_SDK_FREERTOS_SOURCE}/portable/MemMang/
                                          heap_3.c
    ${PICO_SDK_FREERTOS_SOURCE}/portable/GCC/
                                       ARM_CM0/port.c
)

target_include_directories(freertos PUBLIC
    .
    ${PICO_SDK_FREERTOS_SOURCE}/include
    ${PICO_SDK_FREERTOS_SOURCE}/portable/GCC/ARM_CM0
)
```

このテンプレート・プロジェクトのmain関数(**リスト8**)は，RP2040のLED_PIN(GP25)をGPIO機能として初期化し，LEDを250msごとにON/OFFを繰り返すプログラムです．

ステップ6：FreeRTOSプログラム作成

● FreeRTOSを利用する理由

第1部 第4章で説明した通り，一般的にOSなし環境では，各種TinyUSBライブラリ処理，アプリケーション独自の処理などは，main関数中の無限ループの中で処理します．これは，USBのデバイス，ホスト通信で発生した割り込みで作成された処理のイベントがイベント・キューに格納され，無限ループの中でなるべく遅延を最小限にして，それらのイベントを処理する必要があるからです．特にUSBデバイスの場合には，イベント処理が遅延すると，TinyUSBライブラリ中で致命的なエラーが発生します．これを避けるための手段として，リアルタイムOS(RTOS)を利用するのが効率的です．TinyUSBではFreeRTOSのサポートは限定的で，RP2040についてはサンプル・アプリケーションでサポートされていません．ただし，利用することは可能です．

● FreeRTOSの環境構築

FreeRTOSの最小限のプログラムを作成する環境を構築します．FreeRTOS-Kernelリポジトリをgitサブモジュール(lib/FreeRTOS-Kernel)として登録します．同時にCMakeLists.txtを作成します(**リスト9**)．前節と同じように，テンプレート・プログラムとしてprojectsフォルダ下にtemplate_

リスト10　FreeRTOSカーネル・ライブラリを参照するTemplateプロジェクトの**CMakeLists.txt**

```
set(APP_NAME "template_freertos")

add_executable(${APP_NAME})

target_sources(${APP_NAME} PUBLIC
    ${CMAKE_CURRENT_LIST_DIR}/main.c
    ${FREERTOS_DIR}/event_groups.c
    ${FREERTOS_DIR}/list.c
    ${FREERTOS_DIR}/queue.c
    ${FREERTOS_DIR}/stream_buffer.c
    ${FREERTOS_DIR}/tasks.c
    ${FREERTOS_DIR}/timers.c
    ${FREERTOS_DIR}/portable/MemMang/heap_3.c
    ${FREERTOS_DIR}/portable/GCC/ARM_CM0/port.c
)

target_include_directories(${APP_NAME} PUBLIC
    ${CMAKE_CURRENT_LIST_DIR}
    ${FREERTOS_DIR}/include
    ${FREERTOS_DIR}/portable/GCC/ARM_CM0
)

target_link_libraries(${APP_NAME} pico_stdlib)

pico_add_extra_outputs(${APP_NAME})
```

リスト11　FreeRTOS向けテンプレート・プロジェクトの**main**関数例

```
#include <FreeRTOS.h>
#include <task.h>
#include <stdio.h>
#include "pico/stdlib.h"

void led_task()
{
    const uint LED_PIN = PICO_DEFAULT_LED_PIN;
    gpio_init(LED_PIN);
    gpio_set_dir(LED_PIN, GPIO_OUT);
    while (true) {
        gpio_put(LED_PIN, 1);
        vTaskDelay(100);
        gpio_put(LED_PIN, 0);
        vTaskDelay(100);
    }
}

int main()
{
    stdio_init_all();

    xTaskCreate(led_task, "LED_Task", 256, NULL, 1,
                                                NULL);
    vTaskStartScheduler();

    while(1){};
}
```

リスト12　Windows 10環境でのOpenOCDのビルド手順

```
git clone https://github.com/raspberrypi/openocd
                                    (rp2040 branch)
cd openocd
pacman -S mingw-w64-x86_64-libusb
      mingw-w64-x86_64-libftdi mingw-w64-x86_64-hidapi
./bootstrap
./configure --enable-ftdi --enable-stlink
⋮
libjaylink configuration summary:
 - Package version ..... 0.2.0
 - Library version ..... 1:0:1
 - Installation prefix ..... /mingw64
 - Building on ..... x86_64-w64-mingw3
 - Building for ..... x86_64-w64-mingw32

Enabled transports:
 - USB ..... yes
 - TCP ..... yes



OpenOCD configuration summary
--------------------------------------------------
MPSSE mode of FTDI based devices        yes
Raspberry Pi Pico Probe                 yes (auto)
⋮
SEGGER J-Link Programmer                yes (auto)
Use Capstone disassembly framework      no

make
⋮
make install
```

freertosフォルダを作成し，CMakeLists.txt（**リスト10**），FreeRTOSConfig.hおよびmain.c（**リスト11**）を作成します．ルート・フォルダ中のCMakeLists.txtファイルの最後にadd_subdirectory（projects/template_freertos）を追加します．

リスト11の例はTinyUSBライブラリ実装前のFreeRTOSの最小限のサンプルです．FreeRTOSのタスク（led_task）としてLEDの点灯を行います．TinyUSBライブラリ実装時には，led_taskがOSなし環境の無限ループ処理に置き換わります．

ステップ7：デバッグ環境の構築

● Pico向けOpenOCDのビルド

▶ Windows 10環境

Picoが2個あれば，その中の1個をデバッガとして利用できます．デバッガとして利用するPicoと接続するPC環境で使うOpenOCDは**リスト12**の手順でビルドします．

▶ Linux 22.04環境

Linux 22.04環境でも，Windows 10環境とほぼ同じ手順でビルドします（**リスト13**）．

リスト13　Ubuntu 22.04環境でのOpenOCDのビルド手順

```
sudo apt install automake autoconf build-essential
       texinfo libtool libftdi-dev libusb-1.0-0-dev
                                          libhidapi-dev
git clone https://github.com/raspberrypi/openocd
cd openocd
./bootstrap
./configure --enable-ftdi --enable-stlink
⋮
make
```

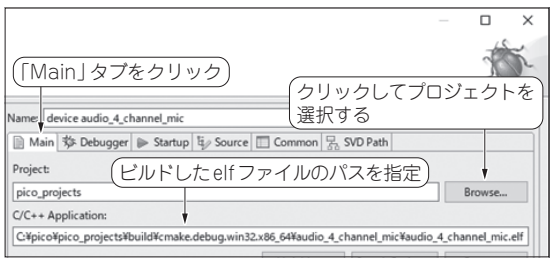

図6　Windows 10環境のOpenOCDでのデバッグ設定1…Mainタブ

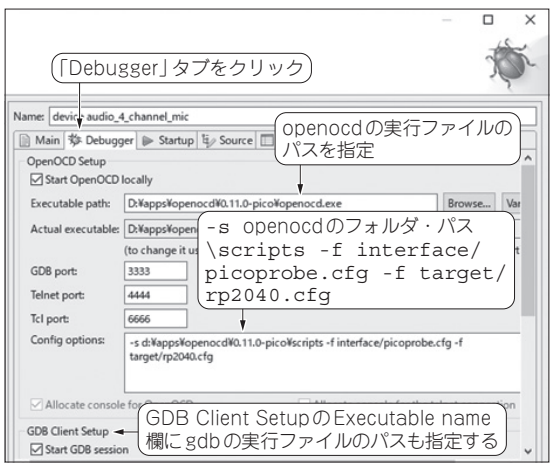

図7　Windows 10環境のOpenOCDでのデバッグ設定2…Debuggerタブ

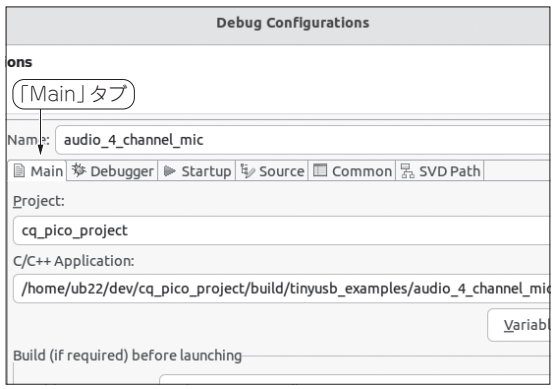

図8　Ubuntu 22.04環境のOpenOCDでのデバッグ設定1…mainタブ

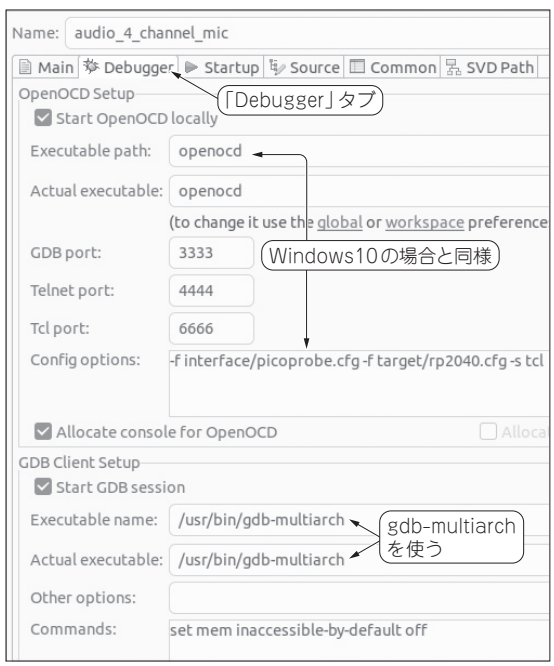

図9　Ubuntu 22.04環境のOpenOCDでのデバッグ設定2…Debuggerタブ

● デバッグ設定

▶Windows 10環境

Eclipseを起動し，ターゲットのプロジェクトを開きます．「実行(Run)」メニューから「Debug Configurations...」を選択し，「Debug Configurations」ダイアログ(**図6**)を開きます．「Main」タブでProject欄の［Browse...］ボタンをクリックしてプロジェクトを選択し，C/C++ Application欄にビルドしたelfファイルのパス名を指定します．パス名はプロジェクトのCMakeLists.txtファイル中の設定に依存します．

「Debugger」タブで，Executable path欄にopenocdの実行ファイルのパス名を入力し，Config options欄にパラメータを指定します(**図7**)．GDB Client SetupのExecutable name欄にgdbの実行ファイルのパス名も指定します．

▶Ubuntu 22.04環境

Ubuntu 22.04環境もWindows 10環境と同様です．**図8**と**図9**に設定例を示します．gdbは，gdb-multiarchを使っています．

せきもと・けんたろう