Raspberry Pi Pico 開発環境の構築（公開資料） (Raspberry Pi Pico Development environment construction)			作成 2021/11/30 ニューラルソフト有限会社 市来 博記
初版	–	2021/11/30	
B	PDF にメタ データを追加した。	2021/12/25	

概要

本書は、Windows 10 が稼働する PC に Raspberry Pi Pico 開発環境を構築する方法について調査した結果を記したものです。

調査内容

Raspberry Pi Pico SDK のインストールから Visual Studio Code を使用したソース レベルのリモート デバッグ環境の構築までを調査しました。

調査項目を示します。

情報源

Getting started withRaspberry Pi Pico

Qiita @iwatake2222 Raspberry Pi Pico C/C++ SDK 環境構築 on Windows 10

Rakuten BLOG「Raspberry Pi Pico のサンプルがビルドできた」

GitHub raspberrypi / picoprobe：openocd with picoprobe segfaulting on windows #3

前提条件

調査における前提条件を示します。

- 開発用ホスト マシンは、Windows10（64 ビット）がインストールされた PC（i7-3930K，32GB メモリ搭載の PC）を使用します。
- 既にインストール済みのソフトウェアへの影響を避ける為、一括インストール用のスクリプトを使用せずに手動でインストールします。
- Raspberry Pi Pico は Probe 用とターゲット用の 2 台を使用します。
- 目標となる開発環境は、図 1 の通りです。
  （赤線部の SPI による通信ソフトウェアーを開発するための環境）

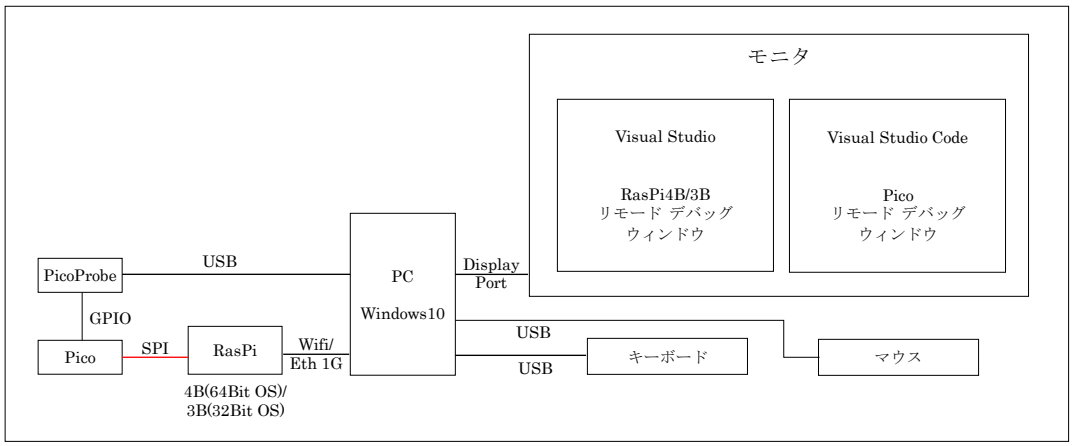

図 1 目標の開発環境

- シリアル通信(USB 仮想 COM ポート)のモニタ用に Tera Term を使用します。
- 本書の記載内容は Microsoft Visusal Studio での C/C++言語プログラムのビルドとデバッグの経験がある方が対象です。

調査結果

〔ツール チェーンのインストール〕

Raspberry Pi Pico 開発環境に必要なツールを以下に示します。

- ARM GCC compiler　10.3-2021.10
- CMake　3.22.0
- Visual Studio 2019　16.11.7
- Python 3.9　3.9.9
- Git　2.34.1

Visual Studio 2019 は既にインストール済み（「C++によるデスクトップ開発」と「C++による Linux 開発」のみ）であった為、インストールを省略しました。Visual Studio 2019 がイストールされていない場合は、

「C++によるデスクトップ開発」のみ選択してインストールすればよいと思います。又、上に示した Visual Studio 2019 以外のツールの最新バージョン(2021/11/30 現在)をインストールして、サンプルをビルドすると失敗する為、以下のバージョンをインストールしました。

- ARM GCC compiler　10-2020-q4-major
- CMake　3.19.4
- Python 3.9　3.9.2
- Git　2.34.1

手順は Getting started withRaspberry Pi Pico の記載通りですが、以下の点に注意しなければなりません。

1　ARM GCC compiler のインストール(完了画面)で、「Add path to environment variable」にチェックを入れる。
2　CMake のインストール(Install Options 設定画面)で、「Add CMake Tto the system PATH for all users」にチェックを入れる。
3　Visual Studio 2019 の「C++によるデスクトップ開発」をインストールする場合、「Windows10 SDK」パッケージのチェックを外してはならない。（SDK の pioasm および elf2uf2 ツールのビルド時に必要）
4　Python 3.9 のインストール(最初の画面)で、「Install launcher for all users」と「Add Python 3.9 to PATH」にチェックを入れる。インストール完了画面で「MAX_PATH length limit」を無効に設定する。
5　Git のインストール
  5.1　Adjusting your PATH enviroment の設定画面で、「Git from the command line and also from 3rd-party software」を選択する。
  5.2　Configuring the line ending conversions の設定画面で、「Checkout as-is,commont as-is」を選択する。
  5.3　Configuring the terminal emulator to use with Git Bash の設定画面で、「Use Windows' default console window」を選択する。
  5.4　Configuring experimental options の設定画面で、「Enable experimental support for pseudo consoles.」にチェックをいれる。

〔Raspberry Pi Pico SDK とサンプル コードの取得〕

1 Git Bash を起動する。
2 cd /ドライブ/～（Raspberry Pi Pico 開発用ルート ディレクトリを作成するディレクトリに移動）
3 mkdir pico（Raspberry Pi Pico 開発用ルート ディレクトリを作成）
4 cd pico
5 git clone -b master https://github.com/raspberrypi/pico-sdk.git
6 cd pico-sdk
7 git submodule update --init
8 cd ..
9 git clone -b master https://github.com/raspberrypi/pico-examples.git

〔コマンド ラインからのサンプル コードのビルド〕

1 Visual Studio 2019 の Developer Command Prompt for VS 2019 を起動する。
2 Raspberry Pi Pico 開発用ルート ディレクトリに移動する。
3 setx PICO_SDK_PATH "..¥..¥pico-sdk"
4 Visual Studio 2019 の Developer Command Prompt for VS 2019 を閉じる。
5 Visual Studio 2019 の Developer Command Prompt for VS 2019 を起動する。
6 Raspberry Pi Pico 開発用ルート ディレクトリに移動する。
7 cd pico-examples
8 mkdir build
9 cd build
10 cmake -G "NMake Makefiles" ..
11 nmake

build ディレクトリに blink、hellow_world 等のサブ ディレクトリが生成され、それらのディレクトリ内に ELF、bin、uf2 形式のモジュールが生成される。

〔LED 点滅プログラムの書き込みと動作確認〕

1 BOOTSEL ボタン押しながら Raspberry Pi Pico と PC を USB で接続する。
2 Pico が USB ストレージとして認識される。エクスプローラなどで Pico ストレージに x:~¥pico¥pico-examples¥build¥blink¥blink.uf2 をドラッグ&ドロップする。
Pico が自動的にリブートして、blink プログラムが実行される。（Pico の USB ストレージとしての接続は解除され、Pico の LED の点滅を確認できる。）

〔“Hellow World”表示プログラム(USB 仮想 COM ポート版)の書き込みと動作確認〕

1　BOOTSEL ボタン押しながら Raspberry Pi Pico と PC を USB で接続する。
2　Pico が USB ストレージとして認識される。エクスプローラなどで Pico ストレージに x:~¥pico¥pico-examples¥build¥hello_world¥usb¥hello_usb.uf2 をドラッグ&ドロップする。
3　Pico が自動的にリブートして、Pico の USB ストレージとしての接続が解除される。
4　Tera Term を起動する。（接続はシリアル：仮想 COM ポートを指定する。）
5　メニューの「設定/シリアルポート」で以下の通信設定を行う。

スピード	: 115200	ストップ ビット	: 1bit
データ	: 8 bit	フロー制御	: none
パリティ	: none		

Tera Term で”Hello, world!”が 1 秒毎に受信されることが確認できる。

〔Visual Studio Code からのサンプル コードのビルド〕

1　マイクロソフトの Visual Studio のページから Visual Studio Code をダウンロードして、インストールする。
2　最初に Visual Studio Code を起動すると、日本語化パッケージのインストールを促すメッセージが表示される。日本語化したい場合はインストールする。
3　Visual Studio Code を終了する。
4　Visual Studio 2019 の Developer Command Prompt for VS 2019 を起動する。
5　起動したコンソールで、”code”+リターン キーを入力する。（Visual Studio 2019 の環境を引き継いだ Visual Studio Code が起動される。）
6　Visual Studio Code の画面の左側のバーにあるブロックのアイコンをクリックして、右側の拡張機能ビューで必要な拡張機能をインストールする。

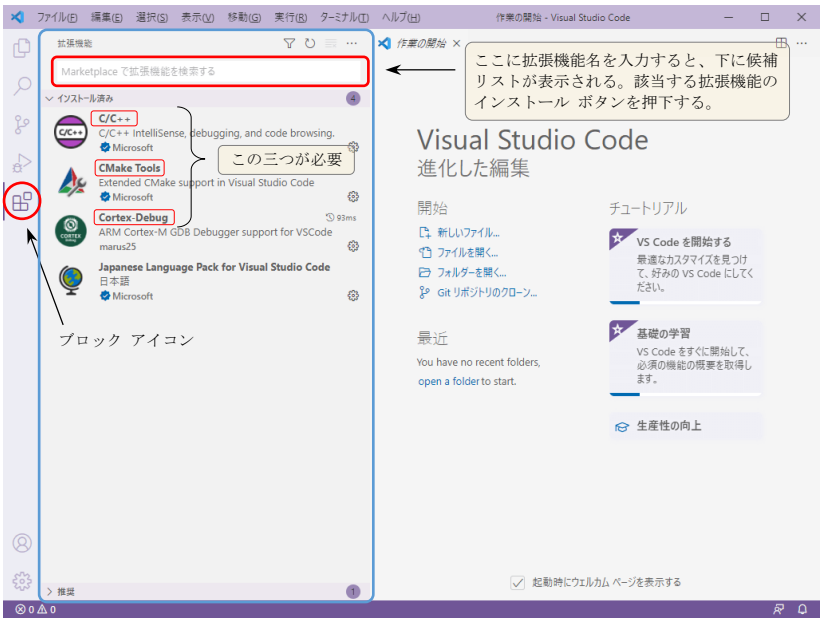

図 2 Visual Studio Code の拡張機能ビュー

CMake Tools に付いてくる VSCode 用 CMake をアンインストールする。

7　Visual Studio Code を再起動する。

8　Visual Studio Code の画面の左側のバーの一番下にある歯車のアイコンをクリックして、メニューの「設定」を選択後、右側の設定ビューで 8.1～8.3 の設定を実施する。

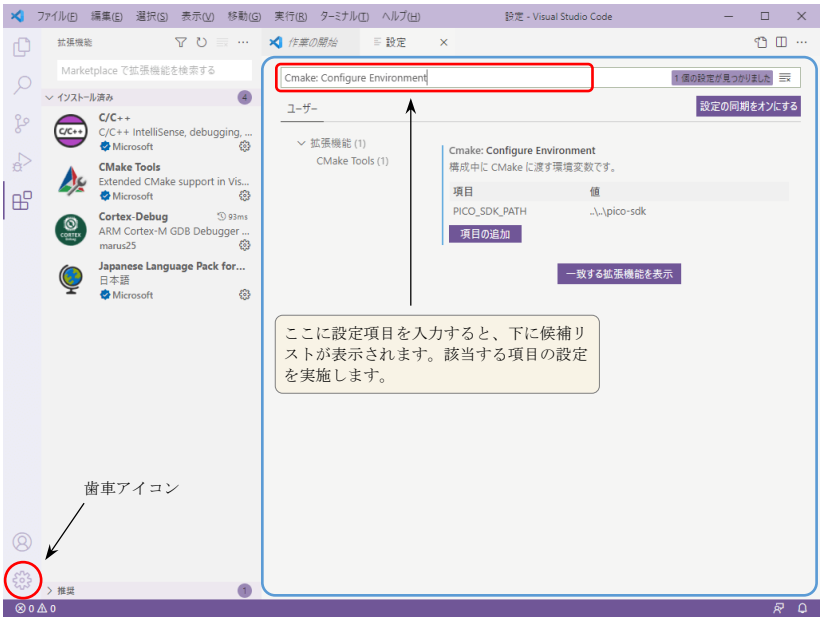

図 3 Visual Studio Code の拡張機能設定ビュー

8.1　Cmake: Configure Environment

項目追加ボタンを押下して、項目名：PICO_SDK_PATH、値：..¥..¥pico-sdk　を追加する。

8.2　Cmake: Generator に「NMake Makefiles」を設定

8.3　Cmake: Output Log Encoding に「utf8」を設定

9　メニューの「ファイル/フォルダーを開く」を選択して、x:~¥pico¥pico-examples を指定する。

10　プロジェクトの構成を行うかどうか（Would you like to configure project 'pico-examples ?'）を確認するメッセージが表示されるので、「はい(Yes)」を選択する。

最後に、使用するコンパイラに"GCC for arm-none-eabi"を指定する。

11　ボトム バーにある歯車アイコンの付いた「Build」ボタンを押下する。（ビルドが開始されます。）

ボトム バーにあるボタンでビルドの構成を変更できる。

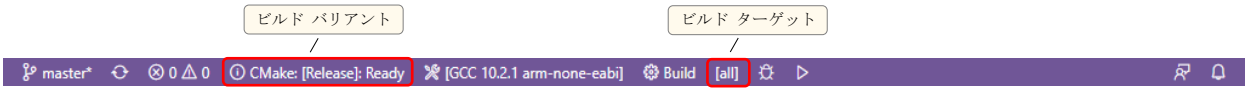

図 4 Visual Studio Code のボトム バー

ビルド バリアント選択ボタン：Release/Debug などの切り替えが可能

ビルド ターゲット選択ボタン：all/blink/hellow_world などの切り替えが可能

〔OpenOCD のビルド〕

1 MSYS2 のサイトから MSYS2 のインストーラをダウンロードして、インストールする。
インストール終了時に「MSYS2 64Bit を起動する」にチェックを入れる。

2 次のコマンドでパッケージとコア システムのデータ ベースを更新する。
pacman -Syu
MSYS2 が閉じた場合は、再起動(64Bit バージョン)する。

3 次のコマンドで更新を完了する。
pacman –Su

4 次のコマンドで必要なソフトウェアーをインストールする。
pacman -S mingw-w64-x86_64-toolchain git make libtool pkg-config autoconf automake texinfo mingw-w64-x86_64-libusb
選択リストが表示されたら、リターン キーのみ入力する。(すべてインストールされる。)
2021/11/30 現在の mingw-w64-x86_64-libusb のバージョンは 1.0.24-4 なのですが、これでビルドした OpenOCD モジュールでホストと PicoProbe を接続すると、セグメンテーション フォールトが発生します。5～7 の手順で旧バージョンの mingw-w64-x86_64-libusb をインストールします。
未確認ですが、4 の手順のコマンドで ”mingw-w64-x86_64-libusb”を除外することもできると思います。その場合は、6 に進みます。

5 次のコマンドで mingw-w64-x86_64-libusb Ver 1.0.24-4 をアンインストールする。
pacman -R mingw-w64-x86_64-libusb

6 https://www.msys2.org/ にアクセス後、「Package Index」を選択して、MSYS2 Packages に移動する。「Repos」を選択して、mingw64 の URL をクリックする。
Index of /mingw/mingw64/ のページで「libusb」を検索して、”mingw-w64-x86_64-libusb-1.0.23-1-any.pkg.tar.xz”をクリックしてダウンロードする。

7 次のコマンドで、mingw-w64-x86_64-libusb Ver 1.0.23-1 をインストールする。
pacman -U mingw-w64-x86_64-libusb-1.0.23-1-any.pkg.tar.xz

8 MSYS2 を閉じて、MSYS2 MinGW x64 を起動して、/ドライブ/~/pico に移動後、次のコマンドを実行する。

8.1 git clone https://github.com/raspberrypi/openocd.git --branch rp2040 --depth=1

8.2 cd openocd

8.3 ./bootstrap

8.4 ./configure --enable-picoprobe --disable-werror

8.5 make -j4

9 次のコマンドで、生成された OpenOCD モジュールが起動できるか確認する。(オプション指定しないので、エラーとなります。)
src/openocd.exe

＜実行例＞

```
Open On-Chip Debugger 0.11.0-gdf76ec7 (xxxx-xx-xx-xx:xx)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
embedded:startup.tcl:26: Error: Can't find openocd.cfg
```

```
in procedure 'script'
at file "embedded:startup.tcl", line 26
Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
Error: Debug Adapter has to be specified, see "adapter driver" command
embedded:startup.tcl:26: Error:
in procedure 'script'
at file "embedded:startup.tcl", line 26
```

〔PicoProbe のビルドと書き込み〕

1 Git Bash を起動して、/ドライブ/~/pico に移動する。
2 git clone https://github.com/raspberrypi/picoprobe.git
3 Visual Studio 2019 の Developer Command Prompt for VS 2019 を起動する。
4 /ドライブ/~/pico に移動後、次のコマンドを実行する。
  4.1 cd picoprobe
  4.2 mkdir build
  4.3 cd build
  4.4 cmake -G "NMake Makefiles" ..
  4.5 nmake
5 BOOTSEL ボタン押しながら Raspberry Pi Pico と PC を USB で接続する。
6 Pico が USB ストレージとして認識される。エクスプローラなどで Pico ストレージに
  x:~¥pico¥picoprobe¥build¥picoprobe.uf2 をドラッグ&ドロップする。
  Pico が自動的にリブートして、picoprobe プログラムが実行される。
7 PC と Raspberry Pi Pico を接続している USB ケーブルを抜く。

以後、この Raspberry Pi Pico は PC と USB ケーブルで繋ぐと、自動的に PicoProbe になります。

〔PicoProbe とデバッグ対象の Pico の接続〕

図 5 に示すように PicoProbe とデバッグ対象の Raspberry Pi Pico を接続する。

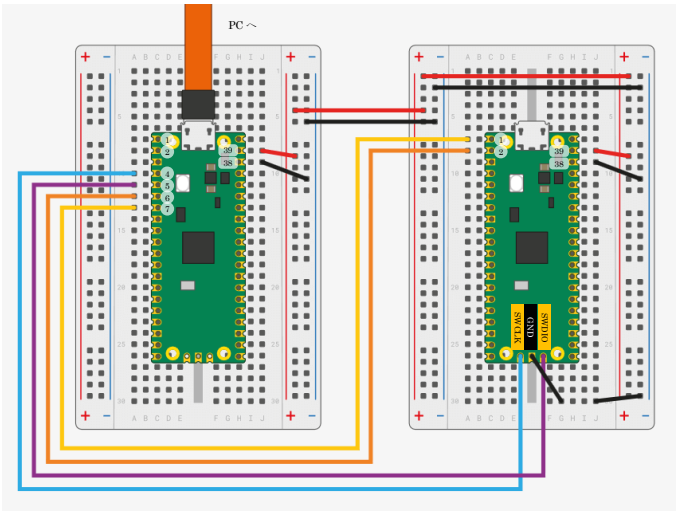

図 5 PicoProbe とデバッグ対象の Raspberry Pi Pico の接続

(Getting started withRaspberry Pi Pico.pdf より引用)

〔PicoProbe 用 USB ドライバーのインストール〕

1　Zadig のサイトにアクセスして、USB Driver インストール ツールをダウンロードする。
（2021/11/30 現在の最新バーションは 2.7）

2　PicoProbe 用の Raspberry Pi と PC を USB ケーブルで接続する。

3　ダウンロードした、zadig-2.7.exe を起動する。
図 6 の左の画面が表示される。（Picoprobe（Interface2）が選ばれた状態になる筈です。）
真ん中のコンボ リストで libusb-win32(V1.2.6.0) を選択する。
「Install Driver」ボタンを押下する。（インストールが開始される。）
暫くすると完了メッセージが表示され、図 6 の右の表示となる。

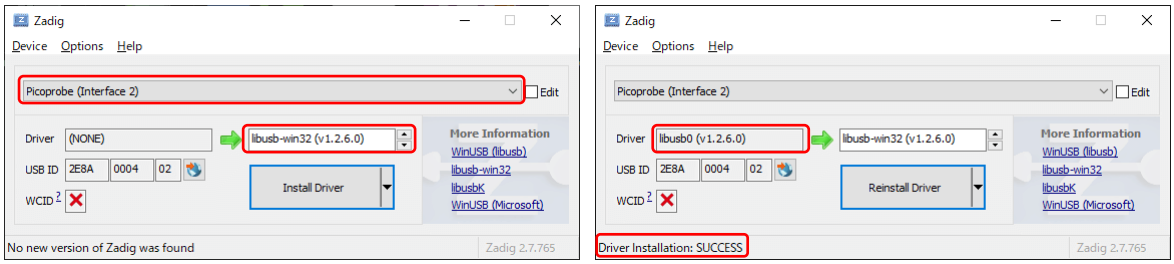

図 6 Zadig USB ドライバ インストール画面

〔LED 点滅プログラムのデバッグ用ビルド〕

1　Visual Studio 2019 の Developer Command Prompt for VS 2019 を起動する。

2　”code” +リターン キーで Visual Studio Code を起動する。（pico-examples は開かれた状態）

3　Visual Studio Code の画面の左側のバーの一番下に表示されている歯車のアイコンをクリックしてメニューの「設定」を選択後、右側の設定ビューで 3.1～3.2 の設定を実施する。

3.1　Cmake: Configure Args に「-DPICO_DEOPTIMIZED_DEBUG:BOOL=on」 を追加

3.2　Cmake: Configure On Open を ON（チェックを入れる）に設定

4　ボトム バーにあるビルド ターゲットで「blink」を選択し、ビルド バリアントで「Debug」を選択して、Visual Studio Code を終了する。

5　/ドライブ/~/pico/pico-examples/build ディレクトリを削除後、Visual Studio Code を起動する。

6　「Build」ボタンを押下する。（x:~¥pico¥pico-examples¥bluild¥blink に blink.elf が生成される。）

図 7 Visual Studio Code のボトム バー

（Debug モードと blink ターゲットを選択）

〔gdb によるデバッグ機能確認〕

1　PicoProbe 用の Raspberry Pi と PC を USB ケーブルで接続する。

2　MSYS2 MinGW x64 を起動する。

3　/ドライブ/~/pico/openocd に移動する。

4　次のコマンドで OpenOCD を起動する。

src/openocd -f interface/picoprobe.cfg -f target/rp2040.cfg -s tcl

＜実行例＞

```
$ src/openocd -f interface/picoprobe.cfg -f target/rp2040.cfg -s tcl
Open On-Chip Debugger 0.11.0-gdf76ec7 (2021-12-03-12:08)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
Info : only one transport option; autoselect 'swd'
adapter speed: 5000 kHz

Info : Hardware thread awareness created
Info : Hardware thread awareness created
Info : RP2040 Flash Bank Command
Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
Info : clock speed 5000 kHz
Info : SWD DPIDR 0x0bc12477
Info : SWD DLPIDR 0x00000001
Info : SWD DPIDR 0x0bc12477
Info : SWD DLPIDR 0x10000001
Info : rp2040.core0: hardware has 4 breakpoints, 2 watchpoints
Info : rp2040.core1: hardware has 4 breakpoints, 2 watchpoints
Info : starting gdb server for rp2040.core0 on 3333
Info : Listening on port 3333 for gdb connections
```

PC と PicoProbe が未接続の場合は、Info : DAP init failed が表示される。

5　別の MSYS2 MinGW x64 を起動する。

6　/ドライブ/~/pico/pico-examples/build/blink に移動する。

7　次のコマンドで gdb を起動する。

gdb-multiarch blink.elf

＜実行例＞

```
GNU gdb (GDB) 11.1
Copyright (C) 2021 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-w64-mingw32".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...

warning: A handler for the OS ABI "Windows" is not built into this configuration
of GDB.  Attempting to continue with the default armv6s-m settings.

Reading symbols from blink.elf...
(gdb)
```

8　次のコマンドで openOCD に接続する。

target remote localhost:3333

＜実行例＞

gdb を起動した MSYS2 MinGW x64 の表示

```
(gdb) target remote localhost:3333
Remote debugging using localhost:3333
warning: A handler for the OS ABI "Windows" is not built into this configuration
of GDB.  Attempting to continue with the default armv6s-m settings.

                .
                .
(gdb)
```

openOCD 起動した MSYS2 MinGW x64 の表示

```
Info : accepting 'gdb' connection on tcp/3333
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x00000178 msp: 0x20041f00
target halted due to debug-request, current mode: Thread
xPSR: 0x61000000 pc: 0x10001670 msp: 0x20041f60
Info : RP2040 B0 Flash Probe: 2097152 bytes @10000000, in 512 sectors

Info : New GDB Connection: 1, Target rp2040.core0, state: halted
Warn : Prefer GDB command "target extended-remote 3333" instead of "target remote 3333"
```

9　次のコマンドで blink.elf をデバッグ対象の Raspberry Pi Pico のフラッシュに書き込む。

load

＜実行例＞

```
(gdb) load
Loading section .boot2, size 0x100 lma 0x10000000
Loading section .text, size 0x3fc8 lma 0x10000100
Loading section .rodata, size 0xe44 lma 0x100040c8
Loading section .binary_info, size 0x20 lma 0x10004f0c
Loading section .data, size 0x1ac lma 0x10004f2c
Start address 0x100001e8, load size 20696
Transfer rate: 7 KB/sec, 3449 bytes/write.
(gdb)
```

10　次の操作で blink.elf の main 関数の先頭にブレーク ポイントを設定した後、ステップ実行する。

monitor reset init

b main

continue

n

n

・

・

quit

‘n’でステップ実行を進めると、デバッグ対象の Raspberry Pi Pico の LED の点滅を確認できる。

〔Visual Studio Code によるリモート デバッグ〕

1 MSYS2 MinGW x64 を起動する。

2 /ドライブ/~/pico/pico-examples に移動して、以下のコマンドを実行する。

2.1 mkdir .vscode

2.2 curl https://raw.githubusercontent.com/raspberrypi/pico-examples/master/ide/vscode/launch-remote-openocd.json -o .vscode/launch.json

2.3 curl https://raw.githubusercontent.com/raspberrypi/pico-examples/master/ide/vscode/settings.json -o .vscode/settings.json

3 MSYS2 MinGW x64 を閉じて、.vscode/launch.json を環境に合わせて修正する。

＜修正例＞

```
{
    "version": "0.2.0",
    "configurations": [
        {
            "name": "Pico Debug",
            "type":"cortex-debug",
            "cwd": "${workspaceRoot}",
            //"executable": "${command:cmake.launchTargetPath}",
            "executable": "${workspaceRoot}/build/blink/blink.elf",
            "request": "launch",
            "servertype": "external",
            // This may need to be arm-none-eabi-gdb depending on your system
            "gdbPath" : "gdb-multiarch.exe のフルパス",
            // Connect to an already running OpenOCD instance
            "gdbTarget": "localhost:3333",
            "svdFile": "${workspaceRoot}/build/${env:PICO_SDK_PATH}/src/rp2040/hardware_regs/rp2040.svd",
            "runToMain": true,
            // Work around for stopping at main on restart
            "postRestartCommands": [
                "break main",
                "continue"
            ]
        }
    ]
}
```

ここまでは、最初にリモート デバッグを実施する時のみの手順です。

4 PicoProbe 用の Raspberry Pi と PC を USB ケーブルで接続する。

5 MSYS2 MinGW x64 を起動する。

6 /ドライブ/~/pico/openocd に移動する。

7 次のコマンドで OpenOCD を起動する。

src/openocd -f interface/picoprobe.cfg -f target/rp2040.cfg -s tcl

8 Visual Studio 2019 の Developer Command Prompt for VS 2019 を起動する。

9 ”code” +リターン キーで Visual Studio Code を起動する。（pico-examples は開かれた状態）

10 F5 キーでデバッグを開始する。

これで、Visual Studio と同じような操作感でデバッグが可能になりました。

所感

今回は、既にインストール済みのソフトウェアへの影響を考え、一括インストール用のスクリプトを使用しなかった為、うんざりするような手順を踏まなければなりませんでした。他のソフトから影響を受けず、他のソフトに影響を与えないカスタマイズ済みのオール・イン・ワン IDE が望まれるのではないかと感じました。（実際、他人が作成したサンプルのビルドが何故失敗するのだろう？と考えることは、時間の無駄です。）