树莓派Raspberry Pico RP2040 开发环境配置完全缝合终极版C-SDK

树莓派 Raspberry Pico RP2040 开发环境配置骨灰版

总览目录

前置说明

```
RP2040作为一款树莓派的单片机，一开始我以为会有很成熟的开发方案，结果发现网上的配置方法大多数是照着官方给的方法一抄，完全没有解释里面各个部分需要做什么的原因以及有什么过程需要了解

这里总结了方法，先记录下来，当然还有一些东西没来得及琢磨 ---- 之前我都是喜欢开箱即用的懒人，然而pico这个官方给的配置方法（C-SDK）实属不当人。
```

先在这里解释一下公共的部分：

0.1 关于调试使用的Debugger

```
    官方做了一个特殊的daplink，叫做picoprobe，就是直接用pico的板子加载一个dap的固件。在之前的说明中，包括现在的文档都指出这个picoprobe所使用的openOCD都需要raspberry分支下的o

    另外也可以使用pico-debug这个固件加载自调试，后面调试章节会说到。

    在Windows下如果用picoprobe需要使用Zadig为其安装驱动为libusb-win32。（不知道现在还是不是这样，网上有的教程还提到了需要复制一个dll文件，但是我看现在的openocd.exe目录下已经

    而调试服务器在这里一般是使用openocd来完成这个的，不过用pyocd也可以。openocd除了能直接搜到的Github地址之外，也可以从这下载：[Download OpenOCD for Windows (gnutoolchains
```

0.2 UF2文件的下载方法

在rp2040运行前按住boot按键然后运行就会让rp2040进入Boot模式，详细看官方手册。

最简单的方法是按住按键然后上电就进入boot模式了，然后电脑上就会出现一个叫做RP2-PI的硬盘驱动器，右键UF2文件，点发送到树莓派对应的驱动器（btw.如果是win11　强烈建议改

复杂一点，如果不想重新上电，可以拉低Run引脚后按Boot，然后释放Run引脚，效果一样，还有软件进入的方法，比如arduino　框架的固件配置了软件进入Boot模式，会让下载比较方

0.3 环境配置的解释

下面的解释都是默认我们配置了环境变量再来说的，如果有的地方没有提到什么程序加入了环境变量就用了，那就是忘了说，除了说了可以不加。当然这个也很好理解，遇到找不到程序

0.4 参考文档

[1] getting-started-with-pico.pdf.官网文档（目前还在更新）

0.5 解释一下下面说了什么

为了试试哪个环境更顺手，所以我找了好几个方案全都试了试。主要包含MicroPython、PlatformIO的方法、基于第三方IDE的方法和基于官方C-SDk给出的cmake的配置方法和后续的导入

如果想直接用官方的方法可以直接按目录跳到第四章节。

1 MicroPython环境

这个虽然简单，但是用python我不是很认可，而且micropython不能支持多核操作。

简单说一下

1、Raspberry Pi Documentation - Microcontrollers，选择**Getting Start with MicroPython**，下载uf2固件后烧录到rp2040上，打开设备管理器会发现多了一个COM端口（就是串口）。

2、下载Thonny IDE, 打开Thonny，右上角点击切换至一般模式，然后点击视图，勾选上这三个：Shell、变量、文件，当然还有别的，这几个最常用。

3、Thonny：工具-设置-解释器-选择Raspberry Pi Pico的解释器，然后选择串口端口，这个端口可以看刚刚设备管理器多出来的那个端口就是的。也可另外用别的工具直接连板子上的串口

4、进入shell直接依次执行python语句测试

```
from machine import Pin
led = Pin(25, Pin.OUT)
led.value(1)
```

5、写python文件

在Thonny里面编辑文本后保存的时候选择保存到pico就行了，开机默认自动运行main.py .

6、资料：Raspberry Pi Pico Python SDK Raspberry Pi Documentation - MicroPython

8、至于开发，随便找个帖子吧：目前全网最详细的树莓派 Pico入门指南！ - 知乎 (zhihu.com)

开发相对容易，导入包很简单，也支持pio操作，所以说效率上如果用了pio也是可以接受的。

2 PlatformIO环境

如果不喜欢用PlatformIO的话另外说，比如它经常不显示出来主页，不知道为啥给我的感觉就是有点卡。

2.1 Arduino框架环境

个人认为，不在乎那个arduino框架的占用，其实还是一个很好的开发平台，对仿真器的支持也还可以。

安装方式：

板子里面搜rp2040或者rasp，安装对应的平台。

文档：

Using the Raspberry Pi Pico SDK (PICO-SDK) — Arduino-Pico 2.6.2 documentation

功能：

支持arudino框架，配置有arduino风格的函数。

也支持在arduino框架下直接使用pico-sdk的函数，看文档应该是支持pio，支持多核，因为本身就是基于C-SDK的。

下载调试：

这个框架一般会自动选择调试工具，如果要更改就在工程下的ini文件里面添加两行，例如：

```
upload_protocol = picotool
debug_tool = cmsis-dap
```

上面的上传和debug参数可以分别改成

- cmsis-dap, 这个就是一般的cmsis-dap调试器（现在应该叫做daplink）
- jlink，
- picotool，就是UF2文件的上传工具
- raspberrypi-swd，就是用picoprobe作为调试器。

- [ ] 如果手上没有调试器或者不在意使用调试功能，那么点编译后在工程目录下.pio/build下有uf2文件，首次需要手动上传arduino框架的uf2固件，之后后面就可以直接用platformio的上传
- [ ] 如果用openocd调试，点击侧边的调试，有一个pio debug的选项。如上所说支持三种外部debugger，不过我用dap v1的时候也不好使，自调试固件也不好使，别人说用官方的picopr

□□□[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-HMbUQVWF-1671073759503)(file:///D:/MyProjects\4_MyOutputs\学习内容的输出\单片机开发环境的配

参考链接：Cannot debug Pi Pico with CMSIS-DAP - “CMSIS-DAP command CMD_DAP_SWJ_CLOCK failed” - PlatformIO IDE - PlatformIO Community

2.2 基于pico-sdk的裸环境Wiz-IO

Wiz-IO/wizio-pico: Raspberry Pi Pico development platform for PlatformIO (github.com)

个人认为，如果这个方法不是存在一点调试的bug的话，应该是最快最完整的C-SDK的方法。目前调试只支持Windows。

安装方式

- PIO Home > Platforms > Advanced Installation
- 粘贴： https://github.com/Wiz-IO/wizio-pico
- 点击INSTALL

功能

支持原生C-SDK编写方式，所以功能都能支持，没有arduino框架当然也就不能支持UF2文件自动上传了。

写着下载支持cmsisdap picoprobe uf2，目前只支持Windows。

下载调试：

下载默认是uf2下载，需要rp2040先手工进入BOOT模式，稍微有点麻烦，可以参考后面的调试方法里面的一键进入BOOT.

如果要调试，在 `platformio.ini` 里面添加对应的方式，比如

```
upload_protocol = picoprobe
debug_tool = picoprobe
```

上面的值可以改成：cmsis-dap，picoprobe, uf2

不过我测试的时候感觉我的wch-link就没被调用过，，可能实际上这个框架里面估计也就picoprobe好使，不清楚是不是我的问题。

这个框架附带了picoprobe的固件， `picoprobe.uf2` have in `tool-pico-openocd` folder，看起来作者可能也就写了这么多。

3 基于第三方IDE配置

下面这几个都很方便，因为他们都提供了集成的SDK下载和模板工程的创建，不需要官方那一套配置。

3.1 基于VisualGDB的Visual Studio创建工程

先安装visualGDB,这是一个给VS提供的跨平台包，关键词：VisualGDB的基本使用。

然后按照下面的方法，新建工程选模板，下sdk就好。

Developing Raspberry Pi Pico Projects with Visual Studio – VisualGDB Tutorials

但是VisualGDB是收费的，告辞，看起来很简单，我没试过。

3.2 基于Embedded Studio

下载链接：SEGGER - The Embedded Experts - Downloads - Embedded Studio

社区版本免费，支持Jlink, GDB server，其中gdb server可以选 `jlink openocd pyocd st-link` 这几种。

自带仿真器，自带rp2040例程，不过他的例程似乎都是干寄存器，似乎没有引用官方的pico-sdk，但是pico-sdk的文件是被包含在内的，只是有一些路径没有被包含，示例程序还是没用上

编译较快，可以使用printf的特性直接输出到窗口。

而且软件支持多平台，工程配置很方便。

对于指定的工程，打开之后按F1会给出一些手册和提示。下面是具体的步骤。

安装SDK

点Tools - Package manager,搜pico，点一下，然后点next，就可以安装好sdk了。

新建工程

点Project-Add project，选树莓派的第一个pico是官方的板子的，程序里面会给个bsp，不带pico就是裸片的工程，区别不大。

然后是工程配置：

> 目标处理器选*0*或者*1*，对应核心，
> 编译器可以选*gcc*或者*segger*，
> 剩下一些配置默认就行，根据需要选

然后编译，debug。

默认没有生成uf2文件。

工程配置可以修改C/C++的包含路径，工程本身包含了部分pico-sdk的include路径，但是没有完全包含----不知道具体原因是为啥----没有被默认配置到包含路径的的包括hardware_xxx下的

```
....\SEGGER Embedded Studio\v3\packagesRP2040_Pico_BSP\pico-sdk\src\rp2_common
```

而且这个pico-sdk是不包含第三方库的。应该是可以自己加上去的，没有仔细试过。

仿真下载调试

J-link

不支持V10以下的Jlink。。。也有的说能支持，但是反正我的Jlink Lite V9显示不支持multi-drop，也有人说需要Jlink v11才行。

参考链接：[SOLVED] RP2040 Support - J-Link/Flasher related - SEGGER - Forum

官方文档指出至少需要v11:

- J-Link HW revision 11 or later (J-Link EDU Mini and J-Trace PRO V1 or later also supported)
- Raspberry Pi RP2040 - SEGGER Wiki

合着来卖配件了，5块钱的单片机配不上这个尊贵的JLink.

Daplink（Openocd）

1、在Shell或命令行里面开一个openocd server等待连接，如：(这一步实际上可选)

```
openocd -f board/pico-debug.cfg
```

这里这个配置文件是openocd自带的，其配置实际上调试的是rp2040的core0，也就是调试的是单个核心。

2、配置工程调试选项

工程上右键 - option - debug - debugger- 选gdb server，然后点gdb server选项，选openocd。

如果自己开命令执行的话就这样配置就可以了，点Debug就会启动gdb连接到刚刚开的openocd这个gdb server。

实际上第一步手动开一个openocd等待的过程可以省掉：

可以勾选工程选项里面debug下面的 `GDB Server-auto start gdb server` 为yes 。

然后填写 `Debug > GDB Server > GDB Server Command Line`：

```
openocd.exe -f board/pico-debug.cfg
```

这个openocd要加入环境变量，或者使用绝对路径。

实测中，单核使用自调试固件的单核配置（也就是pico-debug.cfg这个配置），多核使用WCH-link使用标准的配置，都可以正常调试、查看变量寄存器。

如果想调试多核，标准多核的配置为：

```
openocd -f interface/cmsis-dap.cfg -f target/rp2040.cfg
```

只是如果使用单核的配置的情况下使用外部daplink的时候就不要让它自己启动openocd,不然连接不上(不知道为啥)。

参考

在烧入配置中选择OpenOCD，用DAP仿真器进行下载。 - Embedded IDE Forum (em-ide.com)

这有个网上找的工程模板：pico-demos/pico-ses at master · majbthrd/pico-demos (github.com) 不过似乎没必要用，这个模板很老了，估计是还不支持rp2040 sdk的时候的时候做的，也是

3.3 基于RT-Thread Studio

国产软件，评价很高，社区版免费。下载链接

如果不介意用rt-thread的话，那么可以去使用rt-thread studio（因为目前他支持rtt的工程使用官方给sdk模板）

支持jlink daplink，不过好像都不怎么好使。。

使用要登陆。。。有点蠢，bug也还不少。。安装的时候会出现超长路径导致不能放到比较长的路径里面（我已经关了路径长度限制）

使用方法

先下载SDK, 随便开个窗口或者新建工程点进下载SDK的界面。

对于Pico,只能创建rtt工程，显示三种调试工具可选，点下载程序的小箭头可以改调试下载方式。

openocd调用的是pyocd，实测我手上的这个daplink v1不支持，已经测过ok的自dap是v2的也不好使，所以不知道好不好使,

jlink也是需要jlink支持，我的版本不够。

stlink不可能支持的。。只是显示出来了。

要先点一下调试配置，改一下可执行程序的位置和名称，默认的位置不对。。。点一下搜索项目按钮，选elf那个文件。（目前的版本是这样）

有人使用picoprobe替换驱动为libusb后用来调试（没看见他发调试成功的图啊）我也没成功过，所以不知道调试行不行。

参考链接：RT-Thread-在 RT-Thread Studio 中使用树莓派 Pico 开发板RT-Thread问答社区 - RT-Thread

3.4 Keil MDK

GorgonMeducer/Pico_Template: An MDK template for Raspberry Pi Pico (github.com)

用这个模板，看起来还行，我就不测试了，不喜欢Keil。

参考链接：

MDK震惊！树莓派Pico的调试还能“单体自助”的？ - 腾讯云开发者社区-腾讯云 (tencent.com)

使用方法：

```
git clone https://github.com/GorgonMeducer/Pico_Template .

git submodule update --init
```

3.5 其实还有很多方法

基本上就是使用CortexM0+的对应文件并正确包含pico-sdk库的文件就可以了，其实后面CMakelist同理。

我看还有人用CrossWorks for ARM,很贵，不多说了。上面介绍的方法都是基于工程模板的，后面介绍的就是对应pico官方给出的基于Cmakelist的构建方法。

4 基于官方C语言环境搭建

关于CMake和CMakelist这里不再赘述，网上内容很多，先了解一下比较好。

4.1 官方文档的方法总结

网上找的方法绝大多数都是基于getting-started-with-pico.pdf这个手册上的方法，最开始这个手册只更新了基于树莓派系统的配置，直接执行setup脚本就行了。

官方的配置里面很多配置文件问题一堆，比如上面的setup脚本，很可能自己的环境用不了。另外，配置中如果遇到问题，重新git pull或者直接覆盖更新自己之前下载的文件是最有可能能

总的来说，工程包含下载pico-sdk和pico-example–也就是例程–还有工程模板生成器来避免自己来配置一个工程重复的部分。然后需要CMake构建生成对应的makefile并构建，这其中需要

- **安装需要的开发工具**

  见下面的内容。

- **下载例程、SDK、模板生成工具**

```
git clone https://github.com/raspberrypi/pico-project-generator.git
git clone -b master https://github.com/raspberrypi/pico-sdk.git
cd pico-sdk/
git submodule update --init
cd ..
git clone -b master https://github.com/raspberrypi/pico-examples.git
```

这里配置更新子模块的时候如果报错，就更新到最新的SDK，大概是之前写的有问题。找不到子模块什么的。

- **编译**

后面的步骤里面详细说，大致就是：

cmake …

make

- **下载调试**

需要安装openocd，后面细说。

调试参考getting-started-with-pico.pdf 5.3章节 第六节

swd的使用，调试也就是基于GDB的调试。

而且手册上的第七节讲了vscode的使用，后面具体说。

4.2 Windows下的RP2040环境配置

目前我的环境是Windows11,不过Win10同理了。

4.2.1 安装各类工具链

cmake：

官方生成器是基于cmake构建的，安装的时候要勾选添加环境变量for all users。实际上就是系统环境变量。

gcc-arm-none-eabi:

Arm GNU Toolchain ,这里没啥特殊要求，可以下最新的。安装完了之后手动勾选添加到路径，还要检查一下能不能直接在cmd里面使用arm-none-eabi-gcc命令，否则说明他添加的路径有

python:

后面用他的生成器要用。勾选环境变量，解除长路径限制。

git:

不用说。

上面的直接到官网下载最新版安装就行了，不用纠结版本要不要旧一点。

Windows下的编译器:

这里有两种选择，现在这里说一下区分，一个是安装visual studio，一个是安装其他的比如MinGW。其中，前者的工具为NMake，使用的时候需要先启动vs的工作环境，得到配置的环境变

a. visual studio

需要调用vs的nmake，要用他的命令环境。从微软安装vs的社区版就好了，如果本来就有vs没有mingw又懒得装可以用这个方法，不然还是建议用MinGW。因为后面都需要从Visual Studi

b. 使用MinGW替代visual studio的方法

实际上vs是可选的，只是官方手册里面提供的方法是VS,实际上启动vs的环境非常不方便。

可以用mingw的minge32-make来替代vs的Nmake。

下载MinGW,把其bin目录加入环境变量。

目前mingw新的是mingw-w64,下载网址： MinGW-w64 - for 32 and 64 bit Windows - Browse Files at SourceForge.net

点下面的MinGW-W64 GCC-8.1.0标题下面的包，因为在线安装包压根连不上。

- x86_64-win32-seh这个表示64位pc windows系统 seh错误处理的包（具体怎么选可以搜一下mingw-w64的安装)

（或者mingw32下载链接：https://osdn.net/projects/mingw/# 上面是旧版mingw32，2017年就停更了，可能也能用，没试过）

p.s.

使用旧版的mingw64可能编译报错，编译的时候可能会出现类似于 `'C:/Program' 不是内部或外部命令，也不是可运行的程序` 这样的问题。这里最终用刚刚下载的mingw -w64，测试OK.

c. 其实别的工具也可以ninja或者其他的gcc替代品，不复杂介绍了，

4.2.2 配置路径SDK

上一步中，下好了pico-sdk,examples,还有generator，把pico-sdk放到自己指定的位置，然后把其目录加到环境变量里面，名称是PICO_SDK_PATH. 这个环境变量的路径是给配置出来的(

a.可以使用使用powershell

```
setx PICO_SDK_PATH "youpath\pico-sdk"
```

b.或者Windows下面直接搜path，就能找到编辑系统环境变量的选项。

(其实这里没加也没关系，有别的方法，但是最好加上，因为generator默认用的是环境变量配置)。

c.当然也可以临时配置环境变量，但是没必要这样做, 不然每次都要重新导入环境变量：

在powershell内执行

```
$env:PICO_SDK_PATH=pico-sdk的路径
```

4.2.3 编译例程测试

其实这个步骤不是必须的，但是最好测试一次，同时也演示了不配置IDE怎么编译。

a. 如配置的是visual studio的编译器

也就是官方手册指出的方法，那么打开Visual Studio 2022 Developer Command Prompt：

使用快捷方式或者WindowsTerminal打开（Windows Terminal点加号旁边的下拉，里面有选项），打开之后会提示

```
**********************************************************************
** Visual Studio 2019 Developer Command Prompt v16.7.2
** Copyright (c) 2020 Microsoft Corporation
**********************************************************************
[vcvarsall.bat] Environment initialized for: 'x86_x64'
```

类似于上面这样表示加载了vs的环境，实际上也可以直接打开命令行之后通过执行 `vcvars64.bat` 这个文件来加载环境，也会给出一样的提示。

从刚刚打开的命令行或者powershell，进入 `pico-example` 目录：

（上一步可以选使用*cmd*还是*powershell*，如果用*cmd*，工程切换到别的盘先要换盘符，然后*cd*进去；

*powershell*可以直接进去）

然后，在build目录里面执行 `cmake ..` ，表示取上层目录的cmakelist.txt，这里需要添加参数 `-G "NMake Makefiles"` ,表示生成的是Nmake对应的makefile。

```
mkdir build
cd build
cmake .. -G "NMake Makefiles"
```

如果这里能正常找到对应sdk、toolchain的路径，就不会报错。(就是前面都配置好PICO-SDK和arm-none-eabi-gcc.exe的环境变量了)

继续编译一个例程，比如blink这个程序，可以继续cd到pico-examples/build目录下的blink(随着上面的步骤做完了，目前shell下的未知已经在build文件夹里面了)，

```
cd blink
nmake
```

这时候看编译信息，可以看到正常生成的信息。

进入build/blink，右键xxx.uf2, 选择发送到RPI-RP2这个驱动器，发送完会自动运行。

对于pico-example，也可以进入example下的build目录后执行make（对应nmake或者mingw32-make）后直接编译全部例程，但是这样耗时比较长，没有必要这样做。

b. 如果配置的是MinGW

直接在build目录打开shell，指定生成格式的makefile（不需要从那个VS的环境进）

```
cmake .. -G "MinGW32 Makefiles"
```

然后执行 `mingw32-make` 就可以编译了,其他的步骤和上面一样。只需要替换nmake命令为现在使用的mingw的mingw32-make。

p.s.

如果在工程中先使用nmake的参数生成了makefile，再想要换成mingw32-make，这时候直接使用 `cmake -G "MinGW32 Makefiles"` 生成makefile的话，会报错之前使用的不符合现在的配置

解释一下上面的语句

这里cmake是生成make工具对应的makefile。-G后面的参数用来指定工具，也可以用来生成IDE对应的工程文件。

使用 `cmake --help` 可以看到很多指令参数，里面有各种generator的名称，注意大小写,例如：

```
  Borland Makefiles            = Generates Borland makefiles.
  NMake Makefiles              = Generates NMake makefiles.
  NMake Makefiles JOM          = Generates JOM makefiles.
  MSYS Makefiles               = Generates MSYS makefiles.
  MinGW Makefiles              = Generates a make file for use with
                                 mingw32-make.
  Green Hills MULTI            = Generates Green Hills MULTI files
                                 (experimental, work-in-progress).
  Unix Makefiles               = Generates standard UNIX makefiles.
  Ninja                        = Generates build.ninja files.
  Ninja Multi-Config           = Generates build-<Config>.ninja files.
  Watcom WMake                 = Generates Watcom WMake makefiles.
```

比如也有人用Ninja，但是那需要ninja的构建文件，同mingw步骤。

以上就是标准的例程配置方法。

4.2.4 生成自己的工程并编译烧录

这一步就是使用pico-project-generator生成工程，下文中把pico-project-generator称作生成器。

p.s.（这里提前说一下，如果生成器启动有问题，首先更新pico-project-generator到最新的python文件；与有的早期教程不同，现在生成器在windows下面没有问题，能正常使用，不需要

进入pico-project-generator目录下，右键打开shell，运行：

```
python ./pico_project.py --gui
#或者
python ./pico_project.py -g
```

这里建议找到pico_project.py 647行的self.locationName.set(os.path.abspath(os.getcwd()),改成：

```
self.locationName.set(os.path.abspath(os.path.join(os.getcwd(), "..")))
```

这样默认路径是pico_project.py的上层目录而不是与生成器的python文件在同级目录。

然后建议去掉开头那个logo，，，因为屏幕太小了话不合适的缩放会导致最后点不到确定按钮。，就是下面这两行。

```
self.logo = tk.PhotoImage(file=GetFilePath("logo_alpha.gif"))
logowidget = ttk.Label(mainFrame, image=self.logo, borderwidth=0, relief="solid").grid(row=0,column=0, columnspan=5, pady=10)
```

这样就打开了生成器，然后按照需要勾选内容, 如下：

□□□[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-9AEpIMeN-1671073759511)(file:///./0_RP2040开发环境搭建\pico-project-generator1.png?msec=1671

这样生成器就会根据指定的参数和默认的参数来生成工程需要的cmakelist和其他文件，其中指定参数至少需要指定PICO_SDK_PATH和把armgcc的bin目录加入环境变量，否则生成器会报

生成器的作用是复制需要的文件和生成对应的cmakelist，同时自动执行 `cmake -G 'xxxmake Makefile' .`（个人认为其实这个没有必要，后面使用ide的时候也会配置cmake的生成参数的，

具体可以看raspberrypi/pico-project-generator: Tool to automatically generate a Pico C SDK Project (github.com)项目的readme，或者在命令下加上 --help来查看帮助。生成器本身也可以

p.s.这里得注意，默认情况下，mingw的优先级高于vs，所以配置了两者都有的话，生成器优先生成基于mingw的工程，当然了，只是cmake -G的参数不同，可以改也可以自己切换。但是

在前面配置的编译器是vs的编译器的情况下，

这里如果勾选Run build afer generation，那么需要先进入Visual Studio 2022 Developer Command Prompt的环境，然后打开在命令行里面这个生成器，不然缺环境来执行build，但是一般

生成之后，进入工程目录下，只有一个 工程名.c 文件，main函数就在里面，编写这个文件就可以了。

如果只有这一个.c文件，也就是cmakelist不需要改动，那么直接从Visual Studio 2022 Developer Command Prompt进入工程下build目录，执行nmake就行了。

如果添加了其他的文件，就修改cmakelist.txt , 然后重新执行cmake -G “NMake Makefiles”，然后nmake就好.

在前面配置的编译器是MinGW的情况下：

直接打开生成器生成工程，然后直接在build目录下执行 `mingw32-make` 就可以编译了，更新makefile的话就是， `cmake -G "MinGW32 Makefiles"`

这时候，uf2，elf文件都在工程下build里面。

以上，就是标准的工程创建和配置流程。

至于cmakelist.txt怎么用，参考文档xxxxx

4.2.5、官方方法工程配置总结

其实前面说过了，姑且当他首尾呼应吧。

1、配置好环境，下好官方sdk和例程还有生成器，配置好环境变量。

2、使用生成器生成工程（其实自己写cmakelist或者复制例程里面的cmakelist改一改然后复制必要的文件，自己做成模板是也可以）

3、使用nmake或者make（mingw32-make)编译

4、编译得到的uf2可以直接从usb启动，得到的elf可以使用swd烧录或调试（调试需要之前就编译参数指定为debug），bin文件可以烧录到flash

5、注意工程路径最好不要有中文。

4.3 配置自己的IDE

这一步基于上一步来完成，而不是前面说的已经官方集成模板的IDE。上一步中，直接打开文件夹里面文件编辑和编译，配置文件也不方便，编译烧录调试更不方便，这些都需要在命令行

1. **首先选的可以是使用CMakelist作为配置的ide**，这样直接建立工程就能用而不需要修改配置。比如：

- QtCreator
- Netbeans
- KDevelop Kdevelop 使用CMake作为其主要的项目配置方式。除此之外还支持qmake、makefile
- codelite
- Clion

然而这里先说结论，除了CLion，其他的似乎都没有对嵌入式编程的调试有任何优化，编译很容易，但是调试很难配置，调试起来也麻烦，下面说。

2. **其他IDE**

CMake能够生成一些IDE的工程配置文件（如Visual Studio、CodeBlocks、XCode，Kate，CodeLite），比如Eclipse就有MCU的插件（这也是官方文档给出的方法之一），具体生成参数

3. VSCODE这种编辑器

后面说。

一般配置方法

无论哪种方法，如果前面没有配置系统环境变量，那么需要在IDE内配置环境变量，但是如果之前配置过了系统环境变量，或者比如有的IDE集成了armgcc可选，那么这一步就忽略就好了

PICO_SDK_PATH=D:\MyProgramData\MCU_SDK\pico-sdk

PICO_TOOLCHAIN_PATH=C:\Program Files (x86)\GNU Arm Embedded Toolchain\10 2021.10\bin\arm-none-eabi-gcc.exe

（其实这个arm gcc电脑上各个软件内的都下了好几个了，从platformio共用一个都行）

4.3.1 CLion

使用CMakelist来构建的IDE其中最有名的可能是CLion了，官方start文档也给出了怎么配置基于CLion来配置.

如果需要配置环境，打开Clion后点设置，选 构建、执行、部署 ，CMake，点环境：增加用户环境变量。

在Clion中文件的路径，来源，函数来源都显示很清楚，引用的sdk也能直接显示出来，使用是比较方便的。

他确实方便，直接打开前面步骤中配置好的文件夹为Clion工程，配置一下openocd就可以用了。文件编辑什么的都很方便。

不过似乎不支持printf定向到IDE的debug输出。

编译比vscode快不少。

具体方法：我的文档【CLion配置rp2040工程】

参考：

官方文档：getting-started-with-pico.pdf 10.2. Using CLion

CLion 中开发 RaspberryPi Pico | St.Lee的个人站 (stlee.tech)

4.3.2 VScode作为IDE

通过插件支持完成对嵌入式设备的调试和对Cmake的支持。需要安装CMakeTool cortex-debug，实际上配置了cmake或者cortex-debug之后就不用自己写task.json了，还算方便。下面是步

a. 安装CMake插件

打开设置，搜索 CMake Tools Configuration

找到Configure Environment, 添加上面最开始说的两个环境变量，如果已经有配置好的系统环境变量就可以不用配置。

找到CMake Tools Configuration - Generator，设置为NMake Makefiles或者MinGW Makefiles. 前者是对应使用VS的编译器，后者是对应使用mingw的编译器。

打开项目文件夹，第一次打开的时候会提示选择工具包，点一下选择有arm-none-eabi的这一项。选错过了也没关系，就在最下面的底边栏。

□□□□□□[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-vcnxCboy-1671073759512)(file:///./0_RP2040开发环境搭建/底边栏选择工具箱.png?msec=16701

b. 安装插件：cortex-Debug

打开文件夹之后，按F7是Cmake的build，F5是cortex-debug的调试，当然具体快捷键是什么得看自己vscode的配置，这两种都能编译，后者会编译后启动openocd进行调试。

c. 创建工程

前面使用生成器的时候，勾选IDE Options上的创建vscode project，会自动生成vscode的配置.vscode。不过目前这个生成器的参数反正是有问题的，后面自己改一下保存成模板，以后自

可以直接从Windows terminal里面加号打开Visual Studio 2022 Developer Command Prompt

使用vs的编译器：

首先，使用visual studio的命令行Developer Command Prompt for VS 2022，或者powershell也行，前面说了怎么打开了。

输入code打开vscode(vscode安装的时候默认添加到环境变量里面了)，这个实际上是%comspec% /k "C:\Program Files\Microsoft Visual Studio\2022\Community\Common7\Tools\VsDevC

说个简单点的启动方法：

找到电脑上的C:\Program Files\Microsoft Visual Studio\2022\Community\VC\Auxiliary\Build\vcvars64.bat 如果是32位机器就选后缀32的（现在应该不会了吧）然后打开，然后执行code。

可以写成脚本：

```
call "C:\Program Files\Microsoft Visual Studio\2022\Community\VC\Auxiliary\Build\vcvars64.bat"
code
```

然后直接点击运行就行了。

developer Command Prompt for VS 2022

%comspec% /k “C:\Program Files (x86)\Microsoft Visual Studio\2019\BuildTools\Common7\Tools\VsDevCmd.bat”

Pico CDC/ACM USB 串口在Windows10、linux、Mac上是自动支持的，在win7上可能需要抓换usb驱动，下载Zadig <https://zadig.akeo.ie/> 安装相应的驱动程序。选择 USB ID 2E8A 并使

d. 配置Debug

这里关键是配置.vscode下的配置json文件,下面分每个文件说一下。

launch.json：

自动生成的首先是gdb的路径没有适配系统，原来默认gdbPath是gdb multiarch, 在Windows下要改成 arm-none-eabi-gdb

然后configFiles的内容根据自己用的仿真器修改。

setting.json：

全局设置已经配置好的话，这里可以不修改，也可以把之前的插件设置的全局设置挪到这里来变成工程设置，比如：

```
    //  "cortex-debug.armToolchainPath": "",

    //  "cortex-debug.openocdPath.windows": "D:\\MyPrograms\\1_DevelopTools\\MCU_Tools\\openocd-04887d3b6-i686-w64-mingw32\\bin\\openocd.exe",

    //  "cortex-debug.openocdPath": ""

    // "cmake.generator": "MinGW Makefiles"

    // "cmake.generator": "NMake Makefiles"
```

同时cmaketool的几个按钮它默认候隐藏了，实际上确实没啥用，可以单独把Build按钮显示出来方便单独编译，就是改成default。

extensions.json：

就是推荐的插件，不用改。

上面几个文件配置好了之后自己备份一下，用来复制用。

另外，打开工程后，会提示使用Cmake的智能感知，这会在build下添加一个json用来给c_cpp的智能感知添加一个包含路径的配置，这样就能正确识别SDK的文件路径。

总结：

使用vscode的缺点是启动相当慢,差不多要两分钟，后面每次大概半分钟，应该可以改编译配置吧，没仔细看。

启动调试快捷键是ctrl+shift+D或者F5 ,F7只能编译。

同样不具备定向printf到debug窗口的功能。

参考链接：

Raspberry-Pi-PICO系列–第四篇 开发环境搭建-Windows系统 + Visual Studio Code开发_coder.mark的博客-CSDN博客

树莓派学习笔记4-pico的vscode c/c++工程配置 - 知乎 (zhihu.com)

参考：(104条消息) Raspberry-Pi-PICO系列–第一篇 初识RP2040_coder.mark的博客-CSDN博客_rp2040

4.3.3 其他IDE配置

下面的配置都不太好用我觉得。。

Eclipse

```
cmake -G "Eclipse CDT4 - MinGW Makefiles"
```

参考官方教程，很详细。如果不想单独下个eclipse的话，看看电脑上哪个基于eclipse的IDE不是阉割的可以直接拿来用,不过我没找到。。

Kdevelop:

Download | KDevelop

更新有点慢，上次更新还是两年前。。怎么用Windows编译器的文档上次更新还是六年前。。。

然而，这个IDE没啥用，添加文件还是要自己写cmakelist。。。。并没有方便，虽然cmakelist可以图形管理。。甚至编译后生成的文件都要重新加载才能显示。

所以，打开仍然需要从vs命令行打开，当然如果用mingw的话就不需要了。

debug配置方法：

运行-配置启动-然后更改gdb选项就可以用gdb调试了。

for the target set it to executable and the path to `build/your-project-name.elf`

Click on the Debug part of the launch configuration and set the debugger executable to `/usr/bin/arm-none-eabi-gdb`

Set the Run gdb script to the script created in the previous step. Click apply.

脚本内容：

```
target remote localhost:3333

# monitor arm semihosting enable
mon reset halt
# openocd must be running in the correct dir for the relative paths to work

# Flash
mon flash write_image erase unlock build/your-project-name.bin 0x08000000
mon verify_image build/your-project-name.bin 0x08000000
mon reset halt


# Not sure why but this is needed for KDevelop
file build/your-project-name.elf
load
mon reset halt
disconnect
target remote localhost:3333
```

然后：

```
openocd -f interface/stlink.cfg -f board/stm32h7x3i_eval.cfg
```

下面是参考资料：

KDE/kdev-embedded: Plugin for KDevelop to support the development of embedded systems (github.com)

Use KDevelop with OpenOCD and mbed-os - CodeLV

放弃，我这里会提示elf不是启动文件，上面的教程是kdevlop3.

至于那个插件，是七年前的了。

安装，工程-打开/导入-选择刚刚生成的工程文件夹，点构建就可以构建了。使用yay -S gdb-multiarch 安装gdb-multiarch

linux下可以直接打开工程并构建，linux下的版本也更新，5.9到了。

Kate

这个编辑器倒是算是正常。不过仍然是linux下比较正常，windows下有bug，而且没有默认的gdb前端插件，所以没法调试，可能可以写插件来支持。。。

在linux下使用yay -S gdb-multiarch 安装gdb-multiarch https://aur.archlinux.org/packages/gdb-multiarch如果报错patch的话就显pamac install patch

点下面的构建，然后构建插件编写为：cd ./build && make添加一个cmake构建命令cmake -G “Kate - Unix Makefiles”//或者Kate - MinGW Makefiles添加一个构建cmake并编译命令cmake

这样就可以完成编译功能了。

调试功能：（只有linux下支持这个插件。。）

配置GDB点下面的调试GDB 设置高级设置改成远程TCP自定义初始化脚本：不知道咋搞，应该和上面kdevelop的差不多。

如果不需要这个gdb前端的功能只需要烧录的话，可以使用外部工具的功能执行命令来调用swd烧录elf文件。

Codelite

这个IDE也是可以基于Cmake的，可以直接用cmake -G "CodeLite - MinGW Makefiles"来转一个工程出来

实测能用，效果还不错，但是配置调试比较麻烦，而且不管调试器连接没连上，都会提示连上了。。。而且似乎压根没法调试。但是确实能连上openocd。

利用外部工具执行烧录还是可以的。

同理：

CodeBlocks - MinGW Makefiles

参考：下面这是个stm32的配置https://blog.csdn.net/u012750409/article/details/52516653

Embedded IDE on vscode

这个插件很强大，同时支持导入eclipse gcc的项目，所以可以通过

cmake -G "Eclipse CDT4 - MinGW Makefiles"来生成eclipse的GCC项目导入到vscode 的embedded IDE

就是这样有点麻烦。

由于 Eclipse CDT 和 EIDE 之间的设计差异，EIDE 暂时不能兼容 Eclipse 项目中的一些项目属性和构建设置

导入完成后，将生成一个 `.warning.txt` 文件，其中记录了所有不兼容的 Eclipse 项目属性

您需要根据这些属性的 `名称` 和 `值` 的含义修改 EIDE 项目的 `构建器选项`，直到可以正确编译

但是，它同样需要cortex-debug插件来支持调试。所以。。。没啥意义似乎。

4.4 Linux环境的配置

由于各种原因，Linux下配置简单一点。但是其实差不多。

4.3.1 基于WSL安装PICO C-SDK环境

如果是Windows，就安装WSL,参考笔记wsl安装

https://www.cnblogs.com/tcjiaan/archive/2021/09/19/15307912.html

https://forums.raspberrypi.com/viewtopic.php?t=303209

RP2040 Development with WSL | Rusty Electrons

假设这里已经安装好了前面对应的pico-sdk，example，generator，

还有需要的cmake make git armgcc：

sudo apt install cmake gcc-arm-none-eabi libnewlib-arm-none-eabi libstdc+±arm-none-eabi-newlib make

然后配置环境变量

```
cd pico-sdk
export PICO_SDK_PATH=`pwd`
```

下载例程编译例程

和windows的步骤类似，简单点

下面是废弃步骤做个记录：

编译的时候提示cmake版本不够，直接编译新版的cmake安装。。。

sudo chmod 777 -R cmake-3.25.0-rc1/

1. ./configure
2. make
3. sudo make instal

sudo ln -sf /usr/local/bin/cmake /usr/bin 如果有老版本的cmake

然后编译cmake的时候发现没有g++。。。。。这个环境怎么什么都没有。。。

然后发现是国内源的问题，先换回官方源了。。。

正确步骤：

找到正确的源。。。

编译例程

```
git clone https://github.com/raspberrypi/pico-examples.git
cd pico-examples
mkdir -p build
cd build
cmake ..
make
```

到相应目录下找到 `*.uf2` 文件，将其复制到 RPI-RP2 的移动硬盘 即可完成下载。

使用pico-project-generator生成自己的工程

和Windows下类似，只不过如果是WSL下需要先需要手工安装tkinter（一般不需要）

sudo apt-get install python3-tk

使用python启动生成器。

```
python3 pico_project.py --gui
```

生成的工程进入build

cmake …

就可以编译了，直接make。

全局总结

不是我说，官方方法是真他娘的复杂。

免费且最省事儿方法：

1、RTT studio -缺点，要登陆。。。

2、segger embedded studio -免费，知道怎么写程序的话可能是最省事儿的。

免费省事儿且可商用的方法：

1、platformIO

(arduino就是个框，引用函数都可以写一样)，而且有别人做的包（https://github.com/Wiz-IO/wizio-pico），可以使用裸环境。

缺点，platformIO容易崩溃，能用就不要动配置。

不省事，免费，能用，原生C SDK：

1、VScode 凑合用，配置完了也能调试

2、kdevelop 主要是这个软件不争气，起码作为ide管理文件还是方便的，基本上没法调试还不如kate

不免费但是好用：

Clion。

swd下载方法

openocd -f interface/raspberrypi-swd.cfg -f target/rp2040.cfg -c "program blink.elf verify reset exit"openocd -f interface/cmsis-dap.cfg -f target/rp2040.cfg -c “program blink.elf verify reset

开发方法总结：

1、如果不需要调试功能

（起码目前SWD连接体验很差，jlink有版本要求，daplink也不一定能用，picoprobe要单独做一个或者买一个）

先用arduino框架开发个大概，因为arduino框架不需要掉电重启进入boot模式，然后切换到wio的包用裸模式开发，这应该是体验最好而且免费开源的方法。缺点是编译比较慢。

至于其他的IDE即使不调试，也不一定好用。

一键下载uf2

上图所示，接线非常简单，只需要将GND和RUN引脚连接在一起，在需要更新固件时，我们只要按住我们刚添加的RESET按钮，BOOTSEL按钮；释放RESET按钮，然后释放BOOTSEL

步骤：

按住我们刚添加的RESET按钮和BOOTSEL按钮释放RESET按钮释放BOOTSEL按钮拖动文件更新固件

参考：https://blog.csdn.net/qq_35181236/article/details/115297790

2、需要调试的开发方法

2.1 自调试

单核开发，不需要调试USB的情况下可以使用。

优点：不用说，只需要一根线就能调试了。

代价是占用16k内存（

- GIMMECACHE only: SRAM 0x2003C000 to 0x2003FFFF must not be used by user code）
- ，占用USB,占用一个核心。不过实际上，如果USB的功能最后再加进去，也可以下载进去，开发另外一个核心同理，只是不能调试了。如果用另外一个模式就会占用flash cache。
- user code cannot reconfigure the PLL_USB, as the USB peripheral needs this（本来PLL_USB也就是固定48Mhz的吧）

基本使用方法：

下载 pico-debug-gimmecache.uf2 固件到rp2040，然后命令行

```
openocd -f board/pico-debug.cfg
```

回显会提示：

```
Warn : rp2040-core0.cfg configuration file is deprecated and will be removed in the next release. Use following parameters instead: -c 'set USE_CORE 0' -f target/
```

这个pico-debug.cfg实际上就是

```
source [find interface/cmsis-dap.cfg]

```

```
adapter speed 4000

set CHIPNAME rp2040

source [find target/rp2040-core0.cfg]
```

等效于

```
openocd -f interface/cmsis-dap.cfg -f target/rp2040-core0.cfg -c  'adapter speed 4000' -c 'set CHIPNAME rp2040'
```

按照提示,下面的方法一样（`set USE_CORE 0`要放中间）：

```
openocd -f interface/cmsis-dap.cfg -c  'set USE_CORE 0'-f target/rp2040.cfg  -c 'adapter speed 4000' -c 'set CHIPNAME rp2040'
```

GDB调试

上面那个会保持等待连接，然后另外开一个terminal， attach gdb to openocd

```
# linux下输入：
gdb-multiarch picoboard_blinky.elf
# 在Windows下输入：
arm-none-eabi-gdb.exe picoboard_blinky.elf
```

然后连接gdb上去：

```
(gdb) target remote localhost:3333 或者
target remote :3333 或者
target extended-remote :3333
同理，也可以远程调试，比如
target remote 192.168.31.16:3333
```

当然，远程连接的时候要先检测对应端口有没有被防火墙拦。and load picoboard_blinky.elf into flash:

```
(gdb) load
```

然后运行：

```
(gdb) monitor reset init
(gdb) continue
```

如果需要下断点，就提前下好然后continue，也可以在编译之前时候就下好断点：

```
(gdb) monitor reset init
(gdb) b main
(gdb) continue
```

也可以中途使用 `file filename.elf` 来更换执行文件，更换后用 `load` 加载。

ctrl+c来暂停正在运行的程序。

windows下输入 `exit` 或者ctrl-d退出gdb调试。

注意自调试下不要调试带usb的程序，会出bug，因为daplink本身就用了USB,会导致daplink失效，不过这样依然能把程序通过SWD下进去。

基于VSCode的自调试

前面打开工程的方法还是一样：

Raspberry-Pi-PICO系列–第四篇 开发环境搭建-Windows系统 + Visual Studio Code开发_coder.mark的博客-CSDN博客_树莓派pico开发环境

前面的配置步骤还是一样，就是哪些使用picoprobe的程序配置，后面只需要更改launch.json文件里面的：

```
 configuration:

                "configFiles": [
                    "/interface/picoprobe.cfg",
                    "/target/rp2040.cfg"
                ],
```

替换为

```
                "configFiles": [
                    "interface/cmsis-dap.cfg",
                    "target/rp2040-core0.cfg"
                ],
                "openOCDLaunchCommands": [
                    "transport select swd",
                    "adapter speed 4000"
                ],
```

或者替换为：

```
                "configFiles": [
                    "interface/cmsis-dap.cfg",
                    "target/rp2040.cfg"
                ],
```

```
                "openOCDLaunchCommands": [
                   "set USE_CORE 0"
                    "transport select swd",
                    "adapter speed 4000"
                ],
```

然后，把configurations下面的花括号以及其内容复制一份，加个逗号后加在后面做一些修改，用来增加一份配置方便切换。目的是方便在单核自调试和外部仿真器调试多核中切换。

同样的方法可以添加一份linux的配置。用来方便跨平台。

参考：

pico-debug/openocd.md at master · majbthrd/pico-debug (github.com)

majbthrd/pico-debug：RP2040“Raspberry Pi Pico”的虚拟调试盒，无需添加硬件 (github.com)

2.2 使用外部debug工具

2.2.1 买一个rp2040做一个picoprobe.

这绝对是体验最好的方法。既然用了picoprobe也可以就去配置vscode，也能调试，但

然后用platformIO的wio框架开发裸环境，这是最快的方法。是比较麻烦。

专用的openocd直接从platformio的platformio/tool-openocd-raspberrypi下面扣一个。或者下载别人编译好的。（btw，现在好像不需要专用的openocd了）

2.2.2 其他的daplink或者jlink

首先需要保证手上的cmsis-dap能用，这是大前提，有的dap在openocd里面效果有问题。尽管十块钱的wch-link就能调试了，但是很玄学，在win10上不行，在win11上又行。

使用IDE来调试

见前面的IDE配置文档。

使用pico-generator生成工程

Clion体验组好，支持openocd

然后是vscode，配置很麻烦，cmakelist也要自己写，基本不考虑。

kdevelop是个five，不如vscode。

经过尝试，同样的openocd语句，板载的dap可以连上，我自己的就不行。

调试方法

probe的驱动需要安装为libusb-win32

关于openocd，除了picoprobe似乎没必要下载专用的openocd了，新的已经支持了，不过还没试过。

实测：

jlink需要v10以上，v9可认但是提示不支持。如果用openocd来的话需要替换驱动为winusb（zadig），当然我手上这个jlink lite v9还是不支持连接。Zadig - USB driver installation made ea

使用Jlink-SWD下载与调试Raspberry Pi Pico | CBC Notes (imcbc.github.io)

普通daplink目前是连不上的，和openocd本身的问题可能有关吧。

如果启用了使用USB打印，确实运行的时候会初始化一个串口设备，第一次也会弹出安装“Pico”成功，这个串口设备也能连接，但是似乎串口助手没有收到东西。但同时有个启动失败的叫

svd file就是CMSIS SVD(System View Description)，相当于给编译器看的数据手册。

很奇怪，被指定50000 50001 50002端口的时候，wchlink就会连接不上，就像v1版本的daplink那样的报错。可能是因为指定端口的时候它阻塞了多核的按顺序分配端口。只有单核的时候

[rp2040.cpu] Could not find MEM-AP to control the core

cortex-debug/debug_attributes.md at master · Marus/cortex-debug (github.com)

正常的时候多核和单核的连接：

```
PS C:\Users\AlwaysTS> openocd -f interface/cmsis-dap.cfg  -f target/rp2040.cfg  -c 'adapter speed 4000' -c 'set CHIPNAME rp2040'
Open On-Chip Debugger 0.12.0-rc2+dev-00988-g04887d3b6 (2022-12-03-09:30)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
adapter speed: 4000 kHz

rp2040
Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
Info : CMSIS-DAP: SWD supported
Info : CMSIS-DAP: FW Version = 2.0.0
Info : CMSIS-DAP: Interface Initialised (SWD)
Info : SWCLK/TCK = 1 SWDIO/TMS = 1 TDI = 0 TDO = 0 nTRST = 0 nRESET = 1
Info : CMSIS-DAP: Interface ready
Info : clock speed 4000 kHz
Info : SWD DPIDR 0x0bc12477, DLPIDR 0x00000001
Info : SWD DPIDR 0x0bc12477, DLPIDR 0x10000001
Info : [rp2040.core0] Cortex-M0+ r0p1 processor detected
Info : [rp2040.core0] target has 4 breakpoints, 2 watchpoints
Info : [rp2040.core1] Cortex-M0+ r0p1 processor detected
Info : [rp2040.core1] target has 4 breakpoints, 2 watchpoints
Info : starting gdb server for rp2040.core0 on 3333
```

```
Info : Listening on port 3333 for gdb connections
Info : starting gdb server for rp2040.core1 on 3334
Info : Listening on port 3334 for gdb connections
shutdown command invoked
PS C:\Users\AlwaysTS> openocd -f interface/cmsis-dap.cfg -c  'set USE_CORE 0' -f target/rp2040.cfg  -c 'adapter speed 4000' -c 'set CHIPNAME rp2040'
Open On-Chip Debugger 0.12.0-rc2+dev-00988-g04887d3b6 (2022-12-03-09:30)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
0
adapter speed: 4000 kHz

rp2040
Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
Info : CMSIS-DAP: SWD supported
Info : CMSIS-DAP: FW Version = 2.0.0
Info : CMSIS-DAP: Interface Initialised (SWD)
Info : SWCLK/TCK = 1 SWDIO/TMS = 1 TDI = 0 TDO = 0 nTRST = 0 nRESET = 1
Info : CMSIS-DAP: Interface ready
Info : clock speed 4000 kHz
Info : SWD DPIDR 0x0bc12477, DLPIDR 0x00000001
Info : [rp2040.core0] Cortex-M0+ r0p1 processor detected
Info : [rp2040.core0] target has 4 breakpoints, 2 watchpoints
Info : starting gdb server for rp2040.core0 on 3333
Info : Listening on port 3333 for gdb connections
```

Cortex-Debug for Visual Studio Code | Alone on a Mountaintop (lonesometraveler.github.io)