RP2040配置VSCODE开发环境——一劳永逸版

前言

前面的环境搭建基本上介绍了所有的开发环境配置，也对比了各个环境的优劣。个人认为，非商业开发的话，可能Segger Embedded Studio是最好的，但是我不太习惯他的使用方式。这而通吃的方案则是使用VSCODE+ARMGCC+GCC，PlatformIO 也是不错的，但是我个人不太习惯，而且经常出错。

由于重装了系统，所以决定找一个尽量不依赖环境变量的方法，比如有些软件可以不安装运行，重装后不需要重新安装，可以直接使用，也懒得追加环境变量。另外尽可能减少手工操作。而是下次崩坏还能迅速重建。
其实官方的工程生成器已经尽可能减少手工操作了，但是那个是针对Linux的，本身安装也十分简单。所以这里要做一点修改，这里的教程也是按照Windows来做。
再一个就是官方的方法有些方法并不好，这里也做出一些改进。

安装软件和免安装软件配置

这一步是我尽量避免的，但是之前有些软件装在C盘，重装后直接没了，所以还是得走这一步，所以这一步我也没有验证如果直接使用之前下载好的可执行文件需要添加什么。

gcc arm none eabi

点安装包直接安装就好了。
我的安装目录改到 `D:\MyPrograms\InstalledProgram\10 2021.10`
这里就安装的时候先把环境变量加进去，毕竟只需要打个钩。
然后这个就不用管了，同时他有个选项是注册这个编译器，我不知道如果不用安装包的话这一步的影响是什么。

CMake

这个不用自己加路径，所以直接重装。

pyocd

由于是python安装 的所以重新安装一次就行。

```
pip config set global.index-url https://pypi.tuna.tsinghua.edu.cn/simple
pip install -U pyocd
```

其实这个和openocd 工具二选一即可，推荐用这个。

```
这个工具配置安装极为简单，一个软件支持rp2040的所有仿真器，而且连接稍微快一点点，（同一个工程openocd 15s ,pyocd 14s，真就一点点），自带gdb配置，不知道为啥网上很少再rp2040的教程
```

免安装软件配置

mingw-w64

之前下好的，直接找到之前部署的路径，我这里是：
`D:\MyProgramData\Toolchain\mingw-w64\mingw64\bin`
对应的 `mingw32-make` 路径位于：
`D:\MyProgramData\Toolchain\mingw-w64\mingw64\bin\mingw32-make.exe`

openocd （可选）

直接添加到环境里面。虽然可以配置到cortex-debug的路径里面，但是这个工具比较有用（不如pyocd），可能有时候会在命令行里面用一用。

工程部署

利用树莓派官方提供的生成工程脚本还需要提供 PICO_SDK_PATH 这个环境变量用于生成cmakelists.txt, 这里需要在环境变量中添加这个名称即可；
同样，用于编译生成工具的make工具链也需要增加到环境变量。
这里为了方便操作，直接做了一个BAT批处理来执行这个事情。

```
实际上完全可以在启动器的python脚本内部完成这个事情，但是我不太想改那个庞大的代码加上参数什么的，于是就把这些简单的东西分离出来。
```

由于这个生成器本身就要加上–gui参数来启动图形界面，所以一起集成到了BAT文件里面：

```
@REM "设置文字编码UTF8(不然这里中文是UFT8,在命令行中会默认不是UTF8会出现斤拷棍,但是还是可能有问题建议不要用)"
@echo off
chcp 65001>nul

@REM "下面是要自己指定的部分(=左右不要空格)，后面都是set来设定临时环境变量，如果希望直接设置好，改成setx就行，写了检测，不会重复"
set picosdkpath_temp=D:\MyProgramData\MCU_SDK\pico-sdk
@REM mingw32-make.exe
set "make_bin_path=D:\MyProgramData\Toolchain\mingw-w64\mingw64\bin"
set make_exe_name=mingw32-make.exe
@REM 初始化一下环境变量,获取系统的环境变量
set "env_path=%PATH%"

@REM "起手就问工程名称，免得还要写一次,不要写空格"
set /p projectname=your projectname:
echo your projectname is %projectname%

@REM 检测PICO_SDK，如果不存在就添加
@echo off
if not defined PICO_SDK_PATH (
    echo Environment variable PICO_SDK_PATH is not set
    echo now, set it to %picosdkpath_temp%
    set PICO_SDK_PATH=%picosdkpath_temp%
) else (
```

```
    echo Environment variable PICO_SDK_PATH is set to %PICO_SDK_PATH%
)

@REM "检测PICO_SDK，如果不存在就添加（用if not exist mingw32-make.exe会一直显示不存在，只能查找字符串。或者findstr /C:%make_bin_path%"

echo "%path%" | find /i "%make_bin_path%" >nul
if %errorlevel% equ 0 (
    @REM "存在就显示一下好了"
    echo mingw toolchain has been set to %make_bin_path%
) else (
    @REM "不存在就添加一下"
    echo mingw toolchain is not set

    @REM "追加变量"
    @set "path=%env_path%;%make_bin_path%"
    echo mingw toolchain is set to %make_bin_path%
)

@REM "调用tsv文件，虽然默认也是这个，设置名称，默认打开vscode"
python pico_project.py  %projectname%  -t pico_configs.tsv -p vscode --gui

@REM "完事打开code"
code ../%projectname%
```

这里没有配置用户环境变量，因为担心出错，但是写成文件足够记住步骤了。如果要一次配置好，就把里面的 `set` 命令改成 `setx` ，就可以把环境变量添加到用户变量了(系统变量需要提权

其中，调用**pico_project.py** 文件的时候，添加了几个参数，一个是在启动BAT文件的时候要求输入工程名称，因为我觉得这种方式最方便，没必要像原来那样：打开之后，点击名称位置自己的。
另一个是添加默认勾选配置vscode，使用配置文件从tsv文件加载预设值(用excel打开)，虽然默认也是这个，但是记下来方便修改。
我尝试给他添加默认特性，例如PIO,但是似乎在图形界面里面不生效，没注意看原因。

```
这个--feature 我实在是没看明白,试了一下它没啥用，写进去什么参数都不报错。。
```

然后就可以启动生成器了，启动之后接着用这个环境启动vscode打开工程文件夹。但是这还不够，下面是工程生成器的修改。

生成器修改点

先说槽点：
1、LOGO图片太大
2、工程生成路径和python文件在同一层。
3、daplink只配置了自己的Link
4、vscode配置文件只配置了linux下自己的Link的配置。
下面一个一个修改。

去掉LOGO和修改显示设置

不记得在哪里了，反正就是找那个图案的路径。然后是勾选python.exe，兼容性->勾选高DPI->改成应用程序，不然打开很模糊。

```
显示设置每次换电脑或者更换python版本都要做一次，这个没办法，Windows的问题。
```

1、工程路径修改到当前生成器的上一层

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

2、添加daplink debugger选项

默认是第一个，所以直接加到第一个，对应配置文件是cmsis-dap.cfg

```
debugger_list = ["DapLink","SWD", "PicoProbe"]

debugger_config_list = ["cmsis-dap.cfg","raspberrypi-swd.cfg", "picoprobe.cfg"]
```

3、vscode生成.vscode文件部分

```
def generateProjectFiles(projectPath, projectName, sdkPath, projects, debugger):

```

它这里是直接一行一行写在python文件里面每一行都带个单引号，修改起来真是脑溢血，但是我当时改的时候脑抽了，所以最后都是按照他那个格式写的。

3.1 生成启动文件部分

- 启动文件为 `launch.json`
  首先，修改第一个配置为Pyocd的配置，因为pyocd兼容性最好，配置也最简单。
  然后，添加多个Openocd的选项，分别为：
  指定pico-debug固件作为debugger的仿真Core0;
  另外一个据说不会过时的指定core0的命令方法;
  以及理论上是同时连接双核的调试方法，但是实际上cortex-debug的双核调试还在路上。
  其中Pyocd的配置如下：

```
    {
      "name": "rp2040_core0 Debug with PyOcd",
      "cwd": "${workspaceFolder}",
      "executable": "${command:cmake.launchTargetPath}",
      "request": "launch",
      "type": "cortex-debug",
```

```
        "servertype": "pyocd",
         //可以修改为rp2040_core1
        "targetId": "rp2040_core0",
        "showDevDebugOutput": "none",
        "svdFile": "${env:PICO_SDK_PATH}/src/rp2040/hardware_regs/rp2040.svd",
        "runToEntryPoint": "main",
        // Give restart the same functionality as runToEntryPoint - main
        "postRestartCommands": [
            "break main",
            "continue"
        ]
    },
```

具体Pyocd的配置和cortex-debug的联合使用见我另外写的PyOCD配置。（我还没写）

TODO: ARM也出了一个调试插件，但是要装好几个插件，而且对应rp2040还有点问题，文档太少了，我就没折腾出来，不知道这个会不会更好，以后没问题了再加上去。

- 同时，写入文件添加指定编码utf-8,以便使用vscode默认utf-8编辑之后输出支持中文。

```
吐槽,他这个东西写的好挫.
```

- 添加自动判断对应系统应该用什么gdb server的名字

```
    if isWindows:
        gdb_path = 'arm-none-eabi-gdb'
    else:
        gdb_path = 'gdb-multiarch'
```

- 修改设置,默认显示build按钮
- 推荐扩展文件添加一个 `twxs.cmake`

最后，c_cpp_properites.json 中它写的也不对，只有linux的配置，而且如果是Windows写入的路径也没有转换。但是这个配置打开vscode的时候点配置会被替换掉，也不影响使用，所以

- 预计其他增加（以后有精力再说）
  task.json 来实现只烧录 参考arm的Arm Embedded Debugger的方式，当然也可以用pyocd来做。
  使用PICO TOOL，就是前面说过的PICO USB那个东西，可以实现烧录和简单的启动停止。
  推荐扩展增加Arm Embedded Debugger以及Arm Device Manager，如果可用的话。

完成这些之后，直接点击前面的启动脚本，就可以直接生成可用的工程并打开vscode了，总算有点IDE的样子了。

vscode配置点（踩坑点）

如果是从头配置起来的基本上没下面这些问题。

cmake tools

插件如果是之前备份的到新电脑的，需要重新安装，否则插件加载不出来。

Cmake编译的时候会报错：

```
CMake Error: CMake was unable to find a build program corresponding to "MinGW Makefiles".  CMAKE_MAKE_PROGRAM is not set.  You probably need to select a different
```

似乎可以通过添加 CMAKE_MAKE_PROGRAM 来解决，但是由于没有环境变量，这还需要指定编译器位置，虽然可以通过cmake tools的脚本解决，但是那还需要额外添加一个文件。设
似乎并不能给命令用。

所以最简单的办法就是**添加mingw32的目录到环境变量**。

如果追求极致不要修改环境变量，那么就在启动vscode的时候添加临时环境变量，这个文件放在工程文件夹下面。

```
...  前面就和启动生成器里面添加环境变量的脚本一样
code ../
```

关于Makefile：找不到CMAKE_MAKE_PROGRAM | 码农家园 (codenong.com)
上面这个老哥说了，很多方法都不如加个环境变量。
但是这个mingw32-make确实很常用，就加到系统里面去吧。

cortex-debugger

之前备份的会启动不了。。。重新安装也不行。只好全部移除了vsc配置文件。

Arm Embedded Debugger

vscode上除了cortex-debugger
Arm Embedded Debugger 似乎也可以：
ARM-software/vscode-embedded-debug: Extension support for VS Code Embedded Debug Extension (github.com)
但是官方文档还不全，给的方法也有问题。。。。我试了一下还是连不上, 报错RDDI-ERROR 13 但是工具本身应该没问题。

```
${command:device-manager.getSerialNumber}
```

上面这句命令要安装一个arm device manager扩展才行，但是还是会报错没有连接的设备，指定串号之后可以连接上（串号可以用keil中的配置看到或者使用"pyocd list"命令找到设备的uic
件不行，不知道是不是被复位了。但是即使连接上了会进入断点进不了主程序，不知道缺什么。

所以暂时不搞了，估计效果不会超过pyocd。

由于armgcc工具链移动路径导致的原来cmakelist失效的问题需要迁移原工程

直接编译或者选择对应的kit都会会报错找不到正确的工具链吧啦吧啦。
所以我的建议是不要移动工具链的位置，保持原来的路径。。。。。这样可以套用上次的设置。
不过迁移很简单：
在cmake插件的选择kit的按钮上，点选unspecified ，然后编译一下，再选对应的arm gcc就可以了。当然，不选回去也行，能用就算成功。

总结

通过这些配置，就可以做到同步到另一台电脑上也可以直接使用，只需要修改脚本中的环境变量的路径，如果工具本身被同步了，不用修改也可以直接使用。
同时ARM GCC是被自动查找的，所以路径不影响，只要之前安装过就行。