TULU-C-IDE for Microcontroller Projects

Project 2

Raspberry Pi Pico SDK Custom project setup with Localvimrc plugin, coc.nvim plugin, and ALE plugin.

Procedure Details to Set Up Raspberry Pi Pico Board with RP2040 MCUs Under MS Windows

A more detailed version of the guide can be found here. Raspberry Pi's instructions are long, and it requires Visual Studio Build Tools which is a huge download and also, VS Build Tools eats up a lot of hard drive space. Alternatively, if you already have MSYS2 installed, then you can take advantage of so many of MSYS2's packages, including MinGW-GCC. You won't have to mess up with another toolchain like Visual Studio Build Tools 2019 that you would probably not use for any purpose other than compiling Pico projects.

There is also an article that demonstrates the setup process with MinGW. It's written by Shawn Hymel from Digi-Key along with a full video series on YouTube on Digi-Key's channel, divided into three parts, Part 1, Part 2, Part 3. Part 3 describes how to use another Pico as a debugger. However, what I want is a more simplified quick-reference card like documentation as a future reference. The Toolchain should have minimum dependencies on a MS-Windows machine. I documented the steps I followed.

Dependencies:

- MSYS2 as the 'base'
- ARM GCC Compiler Toolchain (use the MSYS2 version)
- Cmake (use the MSYS2 package instead)
- MinGW-GCC Compiler Toolchain (probably, you already have it [MSYS2])
- LLVM-Clang to autocomplete codes (probably, you already have it too [MSYS2])
- Python 3 x64 (Install System-Wide. Also, use the MSYS2 version.)
- GIT (use the MSYS2 version)
- TULU-C-IDE

Fire-up MSYS2 **(x64)** Terminal without any procrastination. First, update MSYS2.

```
pacman -Syu
```

If you haven't set up MSYS2 already, install MSYS2 with the following packages. Type one command at a time.

```
pacman -Syu
pacman -Syu
pacman -Su
pacman -S tar
pacman -S --needed base-devel mingw-w64-x86_64-toolchain
pacman -S mingw-w64-x86_64-cmake
pacman -S mingw-w64-x86_64-clang
pacman -S mingw-w64-x86_64-lld
pacman -S mingw-w64-x86_64-lldb
pacman -S mingw-w64-x86_64-clang-tools-extra
pacman -S mingw-w64-x86_64-clang-analyzer
pacman -S mingw-w64-x86_64-compiler-rt
pacman -S mingw-w64-x86_64-cppcheck
pacman -S mingw-w64-x86_64-astyle
pacman -S ssh-pageant-git
pacman -S git
pacman -S git-extras
pacman -S git-flow
pacman -S mingw-w64-x86_64-meld3
pacman -S ctags
pacman -S markdown
pacman -S python python-pip
pacman -S cdecl
```

R-Click Context Menus

Add MSYS2 to Windows Explorer Right-Click Menu Items:

https://github.com/njzhangyifei/msys2-mingw-shortcut-menus

```
cd ~/
git clone https://github.com/njzhangyifei/msys2-mingw-shortcut-menus && \
cd ~/msys2-mingw-shortcut-menus/ && \
./install && \
cd ~/ && \
rm -rf ~/msys2-mingw-shortcut-menus/ \
```

Create a Windows Registry file 'openCMDhere.reg' with the following contents:

```
Windows Registry Editor Version 5.00

; Open CMD Here
; 'Open Terminal Here' MS equivalent

; https://github.com/microsoft/terminal/issues/1060
; https://stackoverflow.com/questions/27632612/comment-in-reg-file
; https://docs.microsoft.com/en-us/previous-
versions/windows/embedded/gg469889(v=winembedded.80)?redirectedfrom=MSDN

[HKEY_CURRENT_USER\Software\Classes\Directory\Background\shell\Open CMD Here\command]
@="C:\\Windows\\system32\\cmd.exe"
```

** Remember that sometimes Windows Explorer can be your best friend.

Now we come to the ARM-GCC part.

Type to search:

```
pacman -Ss arm-none-eabi
```

A long list will be populated, containing the search term 'arm-none-eabi'.

Install the x64 version of 'arm-none-eabi-gcc'.

```
pacman -S mingw-w64-x86_64-arm-none-eabi-gcc
```

The MSYS2 package manager will try to resolve dependencies and show a message quite similar to this one:

```
resolving dependencies...
looking for conflicting packages...

Packages (3) mingw-w64-x86_64-arm-none-eabi-binutils-2.35-1
             mingw-w64-x86_64-arm-none-eabi-newlib-3.3.0-1
             mingw-w64-x86_64-arm-none-eabi-gcc-10.1.0-2

Total Download Size:    33.05 MiB
Total Installed Size:  552.64 MiB

:: Proceed with installation? [Y/n]
```

Type 'y' (without quotes) and hit Enter. Also, install,

```
pacman -S mingw-w64-x86_64-arm-none-eabi-gdb
```

OpenOCD:

```
pacman -S mingw-w64-x86_64-openocd
```

Check for the version number of 'cmake' utilities that has been provided by MSYS2.

```
cmake --version
```

Any newer MSYS2 installation should come with CMake version 3.21.1, so the output should be somewhat like below:

```
cmake version 3.21.1
```

Some registry tweak is needed. It's not tedious.

We will have to enable Long Path in the Windows Registry Settings. Open Windows Registry Editor by holding <[`WINDOWS_KEY`]+`R`> then in the Dialog Box, type `regedit`.

Find `Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\FileSystem` then add an entry for `LongPathsEnabled`. Change `Value data` to `1` and `Base` to `Hexadecimal`, or skip everything if the entry is already there.

Choose a location to keep all your Pico projects. I chose E:\ drive.

Now create a directory structure like /e/VSARM/sdk/pico.

```
cd /e/
mkdir VSARM
cd VSARM/
mkdir sdk
cd sdk/
mkdir pico
cd pico/
```

Now, type the commands below, one by one at a time.

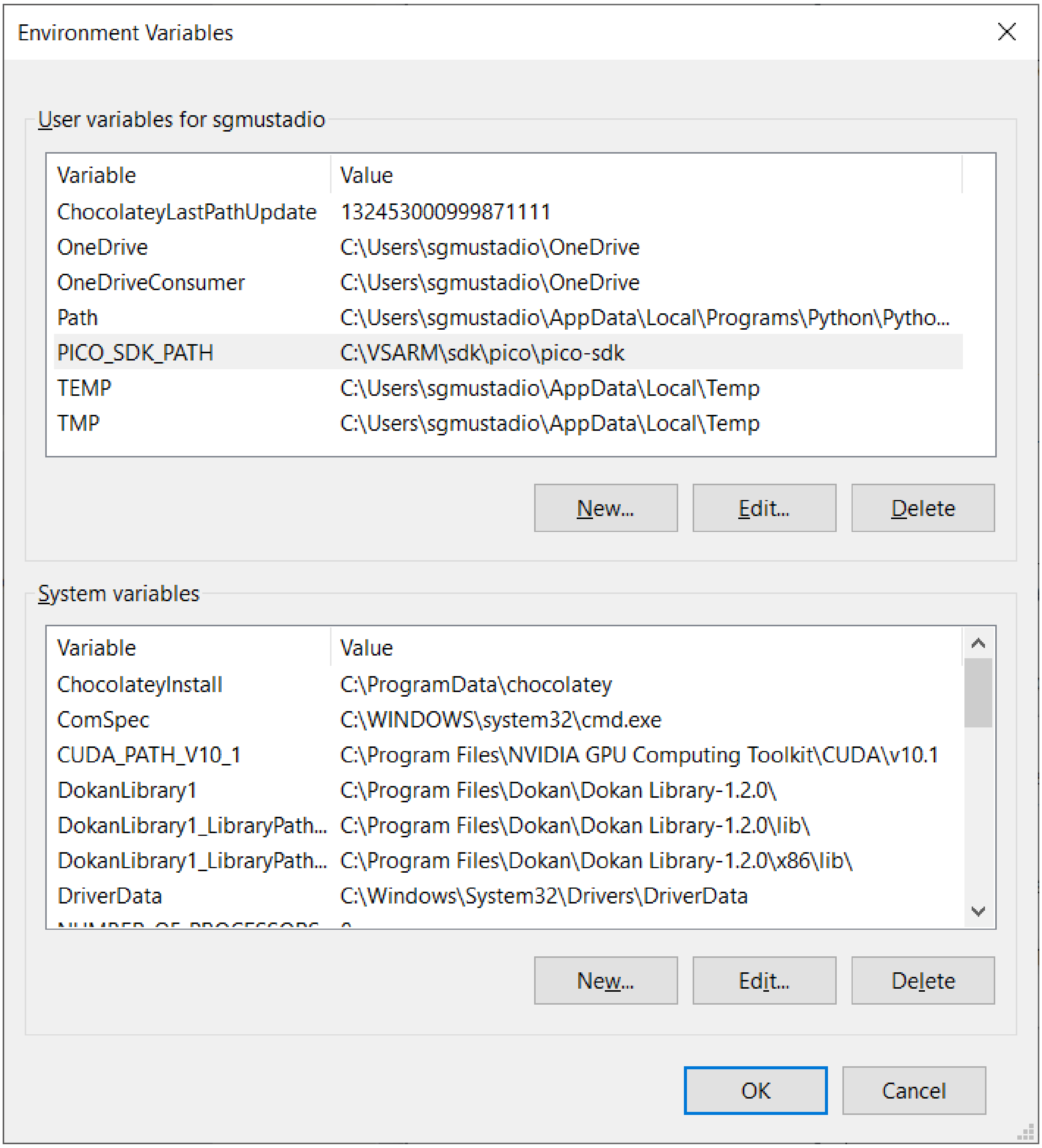
```
cd /e/VSARM/sdk/pico
git clone -b master https://github.com/raspberrypi/pico-sdk.git
cd pico-sdk
git submodule update --init
cd ..
git clone -b master https://github.com/raspberrypi/pico-examples.git
```

Add `pico-sdk` to the system path.

Hold `<[WINDOWS_KEY]+R>`then type `systempropertiesadvanced`in the dialog box. Select the tab 'Advanced'. At the bottom, find 'Environment Variables'. Under 'User variables for YOUR_USERNAME', click 'New'. Add the entries shown below. Fill the entries with:

Variable name: `PICO_SDK_PATH`

Variable value: `E:\VSARM\sdk\pico\pico-sdk`


Environment Variables

User variables for sgmustadio

Variable	Value
ChocolateyLastPathUpdate	132453000999871111
OneDrive	C:\Users\sgmustadio\OneDrive
OneDriveConsumer	C:\Users\sgmustadio\OneDrive
Path	C:\Users\sgmustadio\AppData\Local\Programs\Python\Pytho...
PICO_SDK_PATH	C:\VSARM\sdk\pico\pico-sdk
TEMP	C:\Users\sgmustadio\AppData\Local\Temp
TMP	C:\Users\sgmustadio\AppData\Local\Temp

System variables

Variable	Value
ChocolateyInstall	C:\ProgramData\chocolatey
ComSpec	C:\WINDOWS\system32\cmd.exe
CUDA_PATH_V10_1	C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.1
DokanLibrary1	C:\Program Files\Dokan\Dokan Library-1.2.0\
DokanLibrary1_LibraryPath...	C:\Program Files\Dokan\Dokan Library-1.2.0\lib\
DokanLibrary1_LibraryPath...	C:\Program Files\Dokan\Dokan Library-1.2.0\x86\lib\
DriverData	C:\Windows\System32\Drivers\DriverData

Click 'OK'.

Click 'OK' to finish setting up the Environment Variables as long as there will be one of the settings windows.

In the MSYS2 Terminal (x64) , type `gcc`.

At this point, the output will be:

```
gcc.exe: fatal error: no input files
compilation terminated.
```

Type `make`.

```
make: *** No targets specified and no makefile found.  Stop.
```

Type `mingw32-make`.

```
mingw32-make: *** No targets specified and no makefile found.  Stop.
```

Open Command Prompt ('*WINDOWS+R*', then type '*cmd*').

In the Windows Terminal, which is essentially CMD.EXE on Windows machines, type:

```
echo %PICO_SDK_PATH%
```

The output should come as shown below. If you see something else, then there is something wrong with your setup. It could be that you have made an omission, or maybe something went awry. You might have to repeat the steps described above.

```
echo %PICO_SDK_PATH%
```

Remember, at the command prompt, you need to *click-n-drag* to *select* and hit *enter* to *copy*. Also, *right-click* to *paste*.

```
C:\Users\YOURUSERNAME>echo %PICO_SDK_PATH%
E:\VSARM\sdk\pico\pico-sdk

C:\Users\YOURUSERNAME>
```

We were testing our setup, even if we did not have any files to compile anything. The error messages produced by the compiler toolchains indicate that the necessary compiler executable files are working as expected.

Create a bash alias for 'mingw32-make' with 'make'. From now on, MSYS2 will know that you are calling `mingw32-make` whenever you will be calling `make`.

```
echo "alias make=mingw32-make.exe" >> ~/.bashrc
source ~/.bashrc
```

Fire-up MSYS2 (x64) Terminal. Head over to:

```
cd /e/VSARM/sdk/pico/pico-examples/
```

Follow the steps shown below:

```
mkdir build
```

```
cd build
```

```
cmake -G "MinGW Makefiles" ..
```

NOTE: *Since we are invoking Cmake from the MSYS2 Terminal, we must specify the -G option along with the "MinGW Makefiles" option so that Cmake can generate the necessary makefile for MinGW-GCC.*

The output was:

```
Using PICO_SDK_PATH from environment ('E:\VSARM\sdk\pico\pico-sdk')
PICO_SDK_PATH is E:/VSARM/sdk/pico/pico-sdk
Defaulting PICO_PLATFORM to rp2040 since not specified.
Defaulting PICO platform compiler to pico_arm_gcc since not specified.
-- Defaulting build type to 'Release' since not specified.
PICO compiler is pico_arm_gcc
-- The C compiler identification is GNU 10.1.0
-- The CXX compiler identification is GNU 10.1.0
-- The ASM compiler identification is GNU
-- Found assembler: C:/msys64/mingw64/bin/arm-none-eabi-gcc.exe
Defaulting PICO target board to pico since not specified.
Using board configuration from E:/VSARM/sdk/pico/pico-
sdk/src/boards/include/boards/pico.h
-- Found Python3: C:/Program Files/Python39/python.exe (found version "3.9.0") found
components: Interpreter
TinyUSB available at E:/VSARM/sdk/pico/pico-
sdk/lib/tinyusb/src/portable/raspberrypi/rp2040; adding USB support.
-- Configuring done
-- Generating done
-- Build files have been written to: E:/VSARM/sdk/pico/pico-examples/build
```

The toolchain has created all the directories that you have in '/e/VSARM/sdk/pico/pico-examples/', including the directory 'blink' too.

```
cd blink/
```

(It is: /e/VSARM/sdk/pico/pico-examples/build/blink)

```
make
```

Or,

```
make -j4
```

`-j` option tells the toolchain to use four cores of the processor. Similarly, on a dual-core host, `-j2` will speed up the build process using two of its cores.

The build process took around two minutes on an Intel(R) Core(TM) i5-4570S CPU @ 2.90GHz with 16.0 GB of RAM.

```
[  0%] Creating directories for 'ELF2UF2Build'
[  0%] No download step for 'ELF2UF2Build'
[  0%] No update step for 'ELF2UF2Build'
[  0%] No patch step for 'ELF2UF2Build'
[  0%] Performing configure step for 'ELF2UF2Build'
-- The C compiler identification is GNU 10.3.0
-- The CXX compiler identification is GNU 10.3.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: C:/msys64/mingw64/bin/gcc.exe - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: C:/msys64/mingw64/bin/g++.exe - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: E:/VSARM/sdk/pico/pico-examples/build/elf2uf2
[  0%] Performing build step for 'ELF2UF2Build'
[ 50%] Building CXX object CMakeFiles/elf2uf2.dir/main.cpp.obj
[100%] Linking CXX executable elf2uf2.exe
[100%] Built target elf2uf2
[  0%] No install step for 'ELF2UF2Build'
[  0%] Completed 'ELF2UF2Build'
[  0%] Built target ELF2UF2Build
Scanning dependencies of target bs2_default
[  0%] Building ASM object pico-
sdk/src/rp2_common/boot_stage2/CMakeFiles/bs2_default.dir/compile_time_choice.S.obj
[  0%] Linking ASM executable bs2_default.elf
[  0%] Built target bs2_default
[  0%] Generating bs2_default.bin
[  0%] Generating bs2_default_padded_checksummed.S
[  0%] Built target bs2_default_padded_checksummed_asm
Scanning dependencies of target blink
[  0%] Building C object blink/CMakeFiles/blink.dir/blink.c.obj
[  0%] Building C object blink/CMakeFiles/blink.dir/E_/VSARM/sdk/pico/pico-
sdk/src/rp2_common/pico_stdlib/stdlib.c.obj
[  0%] Building C object blink/CMakeFiles/blink.dir/E_/VSARM/sdk/pico/pico-
sdk/src/rp2_common/hardware_gpio/gpio.c.obj
[  0%] Building C object blink/CMakeFiles/blink.dir/E_/VSARM/sdk/pico/pico-
sdk/src/rp2_common/hardware_claim/claim.c.obj
[  0%] Building C object blink/CMakeFiles/blink.dir/E_/VSARM/sdk/pico/pico-
sdk/src/rp2_common/pico_platform/platform.c.obj
[  0%] Building C object blink/CMakeFiles/blink.dir/E_/VSARM/sdk/pico/pico-
sdk/src/rp2_common/hardware_sync/sync.c.obj
[  0%] Building C object blink/CMakeFiles/blink.dir/E_/VSARM/sdk/pico/pico-
sdk/src/rp2_common/hardware_uart/uart.c.obj
[  0%] Building ASM object blink/CMakeFiles/blink.dir/E_/VSARM/sdk/pico/pico-
```

```
sdk/src/rp2_common/hardware_divider/divider.S.obj
[  0%] Building C object blink/CMakeFiles/blink.dir/E_/VSARM/sdk/pico/pico-
sdk/src/common/pico_time/time.c.obj
[  0%] Building C object blink/CMakeFiles/blink.dir/E_/VSARM/sdk/pico/pico-
sdk/src/common/pico_time/timeout_helper.c.obj
[  0%] Building C object blink/CMakeFiles/blink.dir/E_/VSARM/sdk/pico/pico-
sdk/src/rp2_common/hardware_timer/timer.c.obj
[  0%] Building C object blink/CMakeFiles/blink.dir/E_/VSARM/sdk/pico/pico-
sdk/src/common/pico_sync/sem.c.obj
[  0%] Building C object blink/CMakeFiles/blink.dir/E_/VSARM/sdk/pico/pico-
sdk/src/common/pico_sync/lock_core.c.obj
[  0%] Building C object blink/CMakeFiles/blink.dir/E_/VSARM/sdk/pico/pico-
sdk/src/common/pico_sync/mutex.c.obj
[  0%] Building C object blink/CMakeFiles/blink.dir/E_/VSARM/sdk/pico/pico-
sdk/src/common/pico_sync/critical_section.c.obj
[  0%] Building C object blink/CMakeFiles/blink.dir/E_/VSARM/sdk/pico/pico-
sdk/src/common/pico_util/datetime.c.obj
[  0%] Building C object blink/CMakeFiles/blink.dir/E_/VSARM/sdk/pico/pico-
sdk/src/common/pico_util/pheap.c.obj
[  0%] Building C object blink/CMakeFiles/blink.dir/E_/VSARM/sdk/pico/pico-
sdk/src/common/pico_util/queue.c.obj
[  0%] Building C object blink/CMakeFiles/blink.dir/E_/VSARM/sdk/pico/pico-
sdk/src/rp2_common/pico_runtime/runtime.c.obj
[  0%] Building C object blink/CMakeFiles/blink.dir/E_/VSARM/sdk/pico/pico-
sdk/src/rp2_common/hardware_clocks/clocks.c.obj
[  0%] Building C object blink/CMakeFiles/blink.dir/E_/VSARM/sdk/pico/pico-
sdk/src/rp2_common/hardware_irq/irq.c.obj
[  0%] Building ASM object blink/CMakeFiles/blink.dir/E_/VSARM/sdk/pico/pico-
sdk/src/rp2_common/hardware_irq/irq_handler_chain.S.obj
[  0%] Building C object blink/CMakeFiles/blink.dir/E_/VSARM/sdk/pico/pico-
sdk/src/rp2_common/hardware_pll/pll.c.obj
[  0%] Building C object blink/CMakeFiles/blink.dir/E_/VSARM/sdk/pico/pico-
sdk/src/rp2_common/hardware_vreg/vreg.c.obj
[  0%] Building C object blink/CMakeFiles/blink.dir/E_/VSARM/sdk/pico/pico-
sdk/src/rp2_common/hardware_watchdog/watchdog.c.obj
[  0%] Building C object blink/CMakeFiles/blink.dir/E_/VSARM/sdk/pico/pico-
sdk/src/rp2_common/hardware_xosc/xosc.c.obj
[100%] Building C object blink/CMakeFiles/blink.dir/E_/VSARM/sdk/pico/pico-
sdk/src/rp2_common/pico_printf/printf.c.obj
[100%] Building ASM object blink/CMakeFiles/blink.dir/E_/VSARM/sdk/pico/pico-
sdk/src/rp2_common/pico_bit_ops/bit_ops_aeabi.S.obj
[100%] Building C object blink/CMakeFiles/blink.dir/E_/VSARM/sdk/pico/pico-
sdk/src/rp2_common/pico_bootrom/bootrom.c.obj
[100%] Building ASM object blink/CMakeFiles/blink.dir/E_/VSARM/sdk/pico/pico-
sdk/src/rp2_common/pico_divider/divider.S.obj
[100%] Building ASM object blink/CMakeFiles/blink.dir/E_/VSARM/sdk/pico/pico-
sdk/src/rp2_common/pico_double/double_aeabi.S.obj
[100%] Building C object blink/CMakeFiles/blink.dir/E_/VSARM/sdk/pico/pico-
sdk/src/rp2_common/pico_double/double_init_rom.c.obj
[100%] Building C object blink/CMakeFiles/blink.dir/E_/VSARM/sdk/pico/pico-
sdk/src/rp2_common/pico_double/double_math.c.obj
```

```
[100%] Building ASM object blink/CMakeFiles/blink.dir/E_/VSARM/sdk/pico/pico-
sdk/src/rp2_common/pico_double/double_v1_rom_shim.S.obj
[100%] Building ASM object blink/CMakeFiles/blink.dir/E_/VSARM/sdk/pico/pico-
sdk/src/rp2_common/pico_int64_ops/pico_int64_ops_aeabi.S.obj
[100%] Building ASM object blink/CMakeFiles/blink.dir/E_/VSARM/sdk/pico/pico-
sdk/src/rp2_common/pico_float/float_aeabi.S.obj
[100%] Building C object blink/CMakeFiles/blink.dir/E_/VSARM/sdk/pico/pico-
sdk/src/rp2_common/pico_float/float_init_rom.c.obj
[100%] Building C object blink/CMakeFiles/blink.dir/E_/VSARM/sdk/pico/pico-
sdk/src/rp2_common/pico_float/float_math.c.obj
[100%] Building ASM object blink/CMakeFiles/blink.dir/E_/VSARM/sdk/pico/pico-
sdk/src/rp2_common/pico_float/float_v1_rom_shim.S.obj
[100%] Building C object blink/CMakeFiles/blink.dir/E_/VSARM/sdk/pico/pico-
sdk/src/rp2_common/pico_malloc/pico_malloc.c.obj
[100%] Building ASM object blink/CMakeFiles/blink.dir/E_/VSARM/sdk/pico/pico-
sdk/src/rp2_common/pico_mem_ops/mem_ops_aeabi.S.obj
[100%] Building ASM object blink/CMakeFiles/blink.dir/E_/VSARM/sdk/pico/pico-
sdk/src/rp2_common/pico_standard_link/crt0.S.obj
[100%] Building CXX object blink/CMakeFiles/blink.dir/E_/VSARM/sdk/pico/pico-
sdk/src/rp2_common/pico_standard_link/new_delete.cpp.obj
[100%] Building C object blink/CMakeFiles/blink.dir/E_/VSARM/sdk/pico/pico-
sdk/src/rp2_common/pico_standard_link/binary_info.c.obj
[100%] Building C object blink/CMakeFiles/blink.dir/E_/VSARM/sdk/pico/pico-
sdk/src/rp2_common/pico_stdio/stdio.c.obj
[100%] Building C object blink/CMakeFiles/blink.dir/E_/VSARM/sdk/pico/pico-
sdk/src/rp2_common/pico_stdio_uart/stdio_uart.c.obj
[100%] Linking CXX executable blink.elf
[100%] Built target blink
```

After that, the setup generated one 'blink.uf2' at '/e/VSARM/sdk/pico/pico-examples/build/blink' file that was ready to be uploaded to the flash ROM of any Raspberry Pi Pico RP2040 development board. Some other files were also created along with that 'DOT-UF2' file. The `ls` command revealed those files:

blink.bin

blink.elf

blink.hex

blink.dis

blink.elf.map

blink.uf2

Among them, the 'dot-uf2' file was about 19 KB.

Plug one Pico board. Follow the instructions as described here by Shawn Hymel.

If your machine has loaded the Pico ROM at 'V:', upload that generated UF2 file to the ROM by typing the command:

```
cp blink.uf2 /v/
```

The onboard LED on the Pico board should start blinking right away.

Download the 'Getting Started' guide provided by Raspberry Pi (Trading) Ltd. Documentations from Raspberry Pi are well written. Everything we want to know about the Pico SDK is covered in detail by Raspberry Pi. We have successfully set up our toolchain for Pico. Thanks to Raspberry Pi and Shawn Hymel.

Build your Projects:

```
export PICO_SDK_PATH="/e/VSARM/sdk/pico/pico-sdk" >> ~/.bashrc
export PICO_SDK_PATH="/e/VSARM/sdk/pico/pico-sdk" >> ~/.profile
source ~/.bashrc
source ~/.profile
```

```
cd /e/VSARM/sdk/pico/pico-examples/
mkdir tstprj01
cd /e/VSARM/sdk/pico/pico-examples/tstprj01/
```

```
/e/VSARM/sdk/pico/pico-project-generator/pico_project.py --project cmake test2
cd build/
cmake -G "MinGW Makefiles" ..
make -j4
```

```
/e/VSARM/sdk/pico/pico-project-generator/pico_project.py --feature spi --feature i2c -
-project cmake test
cd build/
cmake -G "MinGW Makefiles" ..
make -j4
```

Pico Project Generator can also generate project folders for VSCode.

```
/e/VSARM/sdk/pico/pico-project-generator/pico_project.py --project vscode test3
cd build/
rm CMakeCache.txt
cmake -G "MinGW Makefiles" ..
```

Then create a Windows BATCH script `vscodestart.BAT` with the contents:

```
@ECHO OFF
code .
exit
```

Double-click on `vscodestart.BAT` to start VSCode.

If needed,

```
rm CMakeCache.txt
```

Generate the `compile_commands.json`:

```
-D CMAKE_EXPORT_COMPILE_COMMANDS=ON
```

Or,

```
mkdir build
(cd build; cmake -D CMAKE_EXPORT_COMPILE_COMMANDS=YES ..)
ln -s build/compile_commands.json
```

The `clangd` server crashed while loading the `compile_commands.json` file when I tried. Do not generate the JSON file if it happens so.

`=================`

`===================`

Now we move on to the autocompletion part.

`===================`

`=================`

Autocompletion with LLVM's clangd LSP

Are you using coc.nvim and ALE?

Copy the folder `LocalVimrc_templates/PLUGIN_CHOICE_ONE` to somewhere on the drive and modify the templates. Mainly, you will need a . `ccls`, a `compile_flags.txt`, a `CMakeLists.txt`, `.lvimrc` and a `pico_sdk_import.cmake` file.

In the MSYS2 (x64) Terminal, type one by one:

```
/e/VSARM/sdk/pico/pico-project-generator/pico_project.py --project cmake test2
cd build/
cmake -G "MinGW Makefiles" ..
make -j4
```

/e/ is the root of the drive where you keep the SDK. You can create projects anywhere you like, not necessarily just in /e/.

Autocompletion and Code Checking are TULU-C-IDE's things.