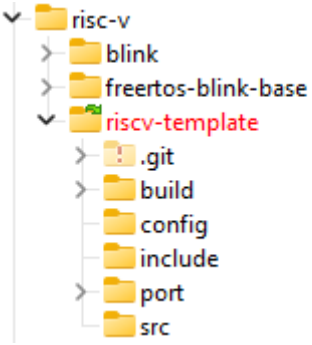
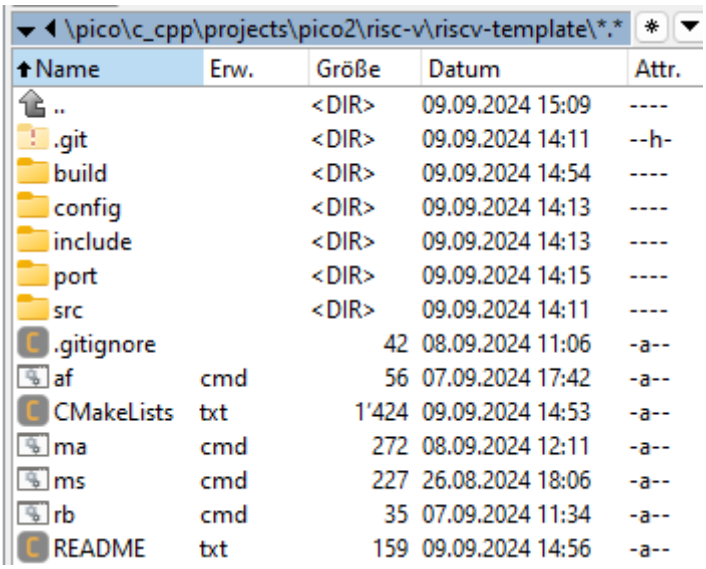
# Template-Project for pico2 RISCV compilation

## 1. Project directory structure




**riscv-template directory**

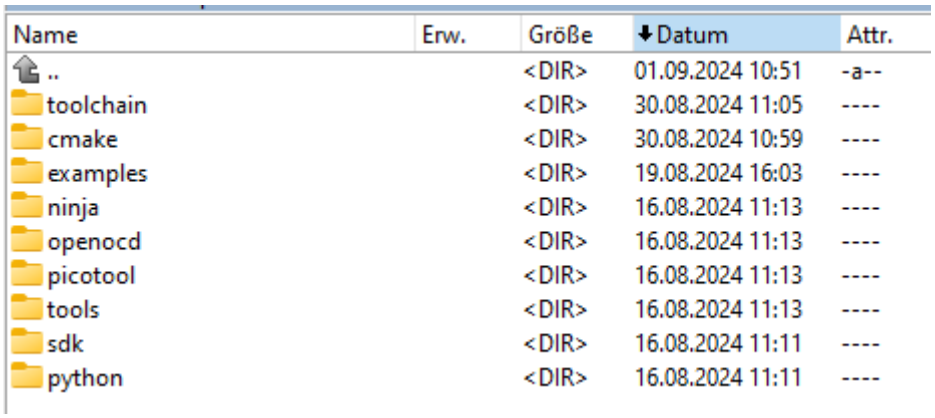



```
config
     config.h.in            see CMakeLists.txt file
port
     logging                my llog-tool, see CMakeLists.txt
          log.h
          logging_levels.h
          logging_stack.h
src
     CMakeLists.txt
     main.cpp
```

## 2. Prequisit

- OS Windows11
- VSCode and the VSCode Raspberry Pico extension are installed, both newest version.
  This installs the SDK, the ARM and RISCV toolchain and nearly all tools you need under %USERPROFILE%\.pico-sdk
- Environment variables
    - PICO_SDK_PATH points to %USERPROFILE%\.pico-sdk\sdk\2.0.0\
    - PICO_RISCV_TOOLCHAIN_PATH points to %USERPROFILE%\.pico-sdk\toolchain\RISCV_COREV_MAY_24\
    - PICOTOOL_FETCH_FROM_GIT_PATH points to %USERPROFILE%\.pico-sdk\picotool\'

The VSCode extension installs under %USERPROFILE%\.pico-sdk the following structure.



| Name | Erw. | Größe | Datum | Attr. |
|---|---|---|---|---|
| .. | | <DIR> | 01.09.2024 10:51 | -a-- |
| toolchain | | <DIR> | 30.08.2024 11:05 | ---- |
| cmake | | <DIR> | 30.08.2024 10:59 | ---- |
| examples | | <DIR> | 19.08.2024 16:03 | ---- |
| ninja | | <DIR> | 16.08.2024 11:13 | ---- |
| openocd | | <DIR> | 16.08.2024 11:13 | ---- |
| picotool | | <DIR> | 16.08.2024 11:13 | ---- |
| tools | | <DIR> | 16.08.2024 11:13 | ---- |
| sdk | | <DIR> | 16.08.2024 11:11 | ---- |
| python | | <DIR> | 16.08.2024 11:11 | ---- |

The RISCV toolchain is in the `toolchain\RISCV….` Directory

The RISCV toolchain is in the `toolchain\13_2_Rel1\` Directory

So we have every thing we need to work.

## 3. The Make files

You can use the same CMakeLists.txt files for both variantes, RISCV or ARM. The difference ist the setting of the variables

`PICO_TOOLCHAIN_PATH` and

`PICO_PLATFORM`

**RISCV**

```
set(PICO_TOOLCHAIN_PATH $ENV{PICO_RISCV_TOOLCHAIN_PATH})
set(PICO_PLATFORM rp2350-riscv CACHE STRING "Pico Platform")
```

**ARM**

```
set(PICO_TOOLCHAIN_PATH $ENV{PICO_ARM_TOOLCHAIN_PATH})
set(PICO_PLATFORM rp2350 CACHE STRING "Pico Platform")
```

You can check the file with cygwins '`file`' command

>file APP.elf

RISCV

```
APP.elf: ELF 32-bit LSB executable, UCB RISC-V, version 1 (SYSV), statically linked, with
debug_info, not stripped
```

ARM

```
APP.elf: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), statically linked, with
debug_info, not stripped
```

## 4. Compile, link and load to pico2

Change to the projects build directory and remove all files.

Open a VSCode console with the command

```
"C:\Program Files (x86)\Microsoft Visual Studio\2019\BuildTools\Common7\Tools\VsDevCmd.bat"
```

Run the commands

```
prj-build>cmake -G "NMake Makefiles" ..
```

```
prj-build>nmake
```

copy the UF2 file from `<prj-build\src>` as usual to the pico2 device.

**Remark to the pico debug Pprobe**

You can load the .elf file with the debug probe from the projects build\src directory with the command:

```
%USERPROFILE%\.pico-sdk\openocd\0.12.0+dev\openocd.exe -s %USERPROFILE%\.pico-
sdk\openocd\0.12.0+dev\scripts -f interface\cmsis-dap.cfg -f target\rp2350-riscv.cfg -c
"adapter speed 5000" -c "program APP.elf verify reset exit"
```

Replace `APP.elf` with your .elf filename if necessary.

If the upload doesn't work, copy first the .uf2 file as usual to the pico2.

The following uploads should work with the probe (my experience…)

## 5. Helpers

| | |
|---|---|
| `af.cmd` | Load ARM compiled .elf file via debug probe (not for use) |
| `afriscv.cmd` | Load ARM compiled .elf file via debug probe (not for use) |
| `make_build.cmd` | make call with VSCode environment |
| `ma.cmd` | make call with MinGW environment |
| `ms.cmd` | make call with MinGW without remove content of build dir |
| `rb.cmd` | remove content of build dir |