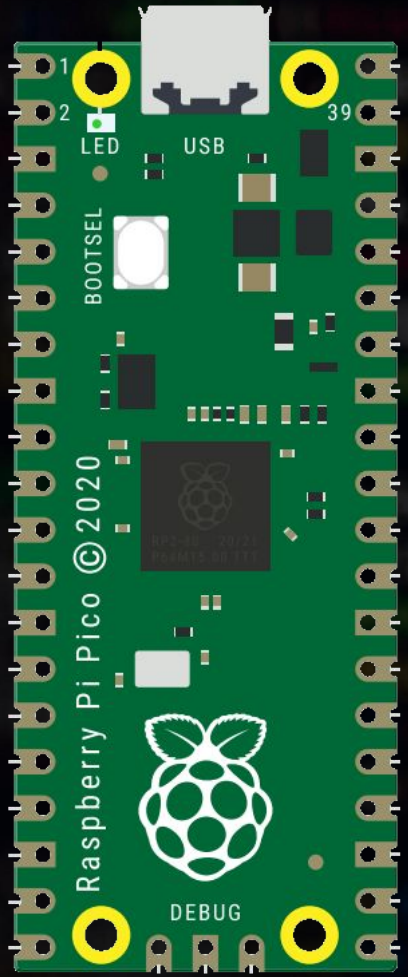
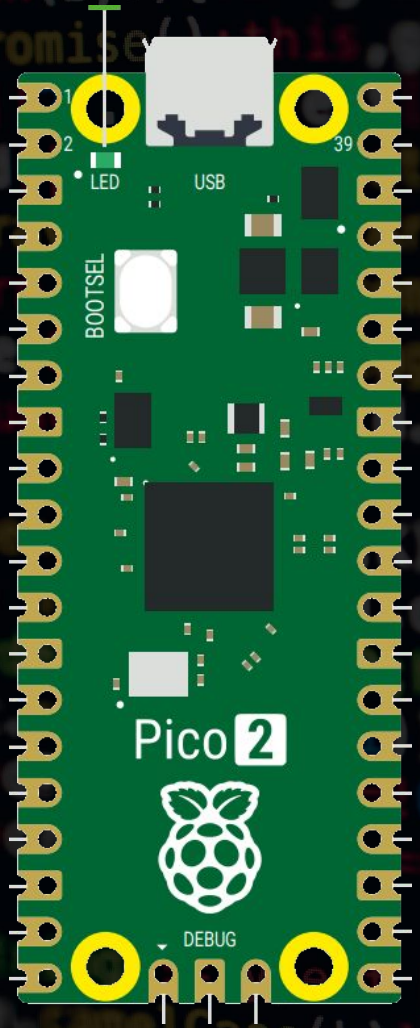
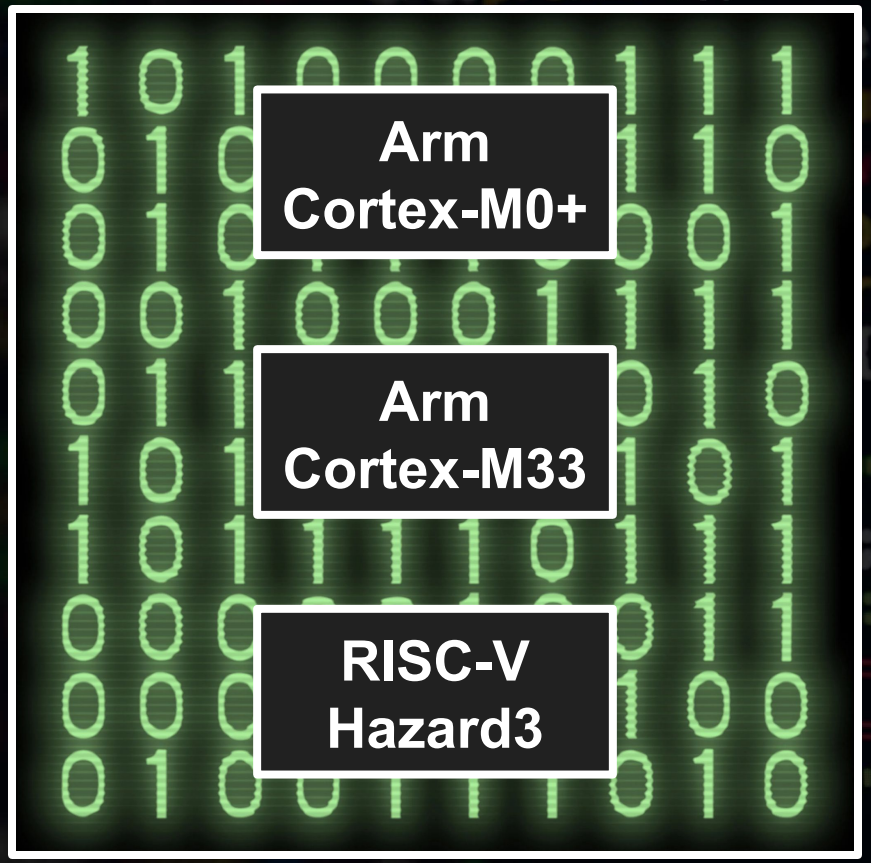
Arm
Cortex-M0+





Arm
Cortex-M33

RISC-V
Hazard3


Arm
Cortex-M0+
Arm
Cortex-M33
RISC-V
Hazard3

RP2350
Fat/Universal Binaries

One Binary to Rule Them All!

RP2040 vs RP2350

	RP2040	RP2350
Clock speed	133MHz	150MHz
SRAM	264K	520K
Arm Cores	2x Cortex-M0+	2x Cortex-M33
RISC-V Cores	N/A	2x Hazard3
Features	Low-level hardware-divider	SHA256 Accelerator, FPU (Arm only)

Universal

- The way a universal build works is it builds separate binaries for each platform.
- Then links them into a single block loop.
- This universal binary will then run on a Pico or Pico 2.

STEP 1

- Get the compiler and SDK for the RP2040 and the RP2350
  - GCC for Arm M0+ and M33
  - GCC for RISC-V Hazard3
- Manually download SDK and manually install compilers
- Let the Pico dev extension in VS Code do it!

Getting started with Raspberry Pi Pico-series

C/C++ development with Raspberry Pi Pico-series and other Raspberry Pi microcontroller-based boards

Appendix C: Manual toolchain setup

Configure your environment via Script

If you are developing for a Pico-series device on the Raspberry Pi 5, the Raspberry Pi 4B, or the Raspberry Pi 400, most of the installation steps in this Getting Started guide can be skipped by running the `pico_setup.sh` script.

The script automates the following setup:

- Creates a directory called `pico` in the folder where you *run* the `pico_setup.sh` script
- Installs required dependencies
- Downloads the `pico-sdk`, `pico-examples`, `pico-extras`, and `pico-playground` repositories
- Defines `PICO_SDK_PATH`, `PICO_EXAMPLES_PATH`, `PICO_EXTRAS_PATH`, and `PICO_PLAYGROUND_PATH` in your `~/.bashrc`
- Builds the `blink` and `hello_world` examples in `pico-examples/build/blink` and `pico-examples/build/hello_world`
- Downloads and builds `picotool` (see Appendix B), and copy it to `/usr/local/bin`.
- Downloads and builds `debugprobe` (see Appendix A).
- Downloads and compiles OpenOCD (for debug support)
- Configures your development Raspberry Pi UART for use with Pico-series devices

TIP

This setup script requires approximately 2.5GB of disk space on your SD card, so make sure you have enough free space before running it. You can check how much free disk space you have with the `df -h` command.

File Edit Selection View Go Run Terminal Help

sha256_rp2040

RASPBERRY PI PICO PROJECT: ...

- General
  - New Project
  - Import Project
  - New Project From Exa...
- Project
  - Debug Project
  - Compile Project
  - Run Project
  - Configure CMake
  - Switch SDK Current: 2.0.0
  - Switch Board
  - Debug Layout
- Documentation
  - Hardware APIs
  - High Level APIs
  - Networking Libraries
  - Runtime Infrastructure

Extension: Raspberry Pi Pico

Raspberry Pi Pico

v0.15.2

Raspberry Pi | 16,212 | ★★★★☆(3)

The official VS Code extension for Raspberry Pi Pico development. It includes several features to simplify project creation and deployme...

Disable Uninstall Auto Update

DETAILS FEATURES CHANGELOG EXTENSION PACK

Raspberry Pi Pico Visual Studio Code extension

> NOTE: The extension is still under development.

This is the official Visual Studio Code extension for Raspberry Pi Pico development. It provides a set of tools to help you get started with development for the Pico boards using Visual Studio Code and the official Pico SDK.

This extension is documented in the Getting started with Raspberry Pi Pico PDF

Download latest Beta RC

Features

- Project generator (targets ninja build-system)
- Automatic CMake configuration on project load
- Easy option to switch between different versions of the Pico-SDK and tools
- No configuration of environment variables required
- Automatic download/management of Toolchain and SDK and some tools (no separate manual download/installation required)
- One-click project compilation per status bar button (with the selected Pico-SDK and tools)
- Offline documentation for the Pico-SDK

Requirements by OS

Categories

Snippets Other

Resources

Marketplace
Issues
Repository
License
Raspberry Pi

More Info

Published	2024-03-27, 18:58:08
Last released	2024-08-23, 19:40:09
Last updated	2024-08-26, 17:40:53
Identifier	raspberry-pi.raspberry-pi-pico

0 0 0

Compile Run Pico SDK: 2.0.0 Board: pico

STEP 1a

```
ls ~/.pico-sdk/sdk/2.0.0/
```

```
bazel          cmake           CONTRIBUTING.md  external  LICENSE.TXT   pico_sdk_init.cmake     README.md  test   WORKSPACE
BUILD.bazel    CMakeLists.txt  docs             lib       MODULE.bazel  pico_sdk_version.cmake  src        tools
```

STEP 1a

```
ls ~/.pico-sdk/sdk/2.0.0/
```

```
bazel        cmake           CONTRIBUTING.md  external  LICENSE.TXT   pico_sdk_init.cmake     README.md  test   WORKSPACE
BUILD.bazel  CMakeLists.txt  docs             lib       MODULE.bazel  pico_sdk_version.cmake  src        tools
```

```
ls ~/.pico-sdk/toolchain/
```

```
13_2_Rel1  RISCV_RPI_2_0_0_2
```

STEP 2 - Set up project

```
$ mkdir universal
$ cd universal
$ code main.c
```

```c
#include <stdio.h>
#include "pico/stdlib.h"
#include "pico/bootrom.h"
#include "boot/picoboot.h"

int main() {
    stdio_init_all();
    int i=0;
    while (true) {
#if PICO_RP2350
        printf("I'm an RP2350 ");
#ifdef __riscv
        printf("running RISC-V\n");
#else
        printf("running ARM\n");
#endif
#else
        printf("I'm an RP2040\n");
#endif

        if(i>10) {
#if PICO_RP2350
    #ifdef __riscv
                rom_reboot(REBOOT2_FLAG_REBOOT_TYPE_NORMAL | REBOOT2_FLAG_REBOOT_TO_ARM, 1000, 0, 0);
    #else
                rom_reboot(REBOOT2_FLAG_REBOOT_TYPE_NORMAL | REBOOT2_FLAG_REBOOT_TO_RISCV, 1000, 0, 0);
    #endif
                printf("Rebooting to other architecture\n");
#endif
            i = 0;
        }
        sleep_ms(1000);
        i++;
    }
}
```

STEP 2a

```
$ code CMakeLists.txt
```

```
cmake_minimum_required(VERSION 3.13)
include(pico_sdk_import.cmake)
project(test_project C CXX ASM)
set(CMAKE_C_STANDARD 11)
set(CMAKE_CXX_STANDARD 17)
pico_sdk_init()
# Ensure a picobin block is present, even on RP2040, so it can be linked into the block loop
target_compile_definitions(pico_crt0 INTERFACE PICO_CRT0_INCLUDE_PICOBIN_BLOCK=1)
add_executable(main main.c)
pico_enable_stdio_usb(main 1)
pico_add_extra_outputs(main)
target_link_libraries(main pico_stdlib)
```

STEP 2b

```
$ cp ~/.pico-sdk/sdk/2.0.0/external/pico_sdk_import.cmake .
```

STEP 2c

```
$ ls
CMakeLists.txt  main.c  pico_sdk_import.cmake
```

STEP 2d

```
$ mkdir buildpico
$ mkdir buildpico2a
$ mkdir buildpico2r
```

STEP 2 - Build for Pico 2 Arm

```
$ cd buildpico2a
$ export PICO_BOARD=pico2
$ export PICO_SDK_PATH=~/.pico-sdk/sdk/2.0.0/
$ export PICO_PLATFORM=rp2350-arm-s
$ export PICO_COMPILER=pico_arm_gcc
$ export PICO_TOOLCHAIN_PATH=~/.pico-sdk/toolchain/13_2_Rel1/
$ cmake ..
$ make
```

STEP 2 - Build finished

```
$ file main.elf

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


$ file main.uf2

main.uf2: UF2 firmware image, family 0xe48bff57,
address 0x10ffff00, 2 total blocks
```

STEP 2 - Build for Pico 1 (Cortex-M0+)

```
$ cd ../buildpico
$ export PICO_BOARD=pico
$ export PICO_SDK_PATH=~/.picofile-sdk/sdk/2.0.0/
$ export PICO_PLATFORM=rp2040
$ export PICO_COMPILER=pico_arm_gcc
$ export PICO_TOOLCHAIN_PATH=~/.pico-sdk/toolchain/13_2_Rel1/
$ cmake ..
$ make
```

STEP 2 - Build finished

```
$ file main.elf

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


$ file main.uf2

main.uf2: UF2 firmware image, family Raspberry Pi
RP2040, address 0x10000000, 94 total blocks
```

STEP 2 - Build for Pico 2 Hazard3

```
$ cd ../buildpico2r
$ export PICO_BOARD=pico2
$ export PICO_SDK_PATH=~/.pico-sdk/sdk/2.0.0/
$ export PICO_PLATFORM=rp2350-riscv
$ export PICO_COMPILER=pico_riscv_gcc
$ export PICO_TOOLCHAIN_PATH=~/.pico-sdk/toolchain/RISCV_RPI_2_0_0_2/
$ cmake ..
$ make
```

STEP 2 - Build finished

```
file main.elf

main.elf: ELF 32-bit LSB executable, UCB RISC-V,
RVC, soft-float ABI, version 1 (SYSV), statically
linked, with debug_info, not stripped


$ file main.uf2

main.uf2: UF2 firmware image, family 0xe48bff57,
address 0x10ffff00, 2 total blocks
```

STEP 3 - Combine

```
$ cd ..
$ ~/.pico-sdk/picotool/2.0.0/picotool/picotool \
link main.bin \
buildpico/main.bin \
buildpico2a/main.bin \
buildpico2r/main.bin \
--pad 0x1000
```

STEP 3 - Make RP2040 .uf2 file

```
$ ~/.pico-sdk/picotool/2.0.0/picotool/picotool \
uf2 convert main.bin rp2040.uf2 \
--family rp2040 --offset 0x10000000
```

STEP 3 - Make RP2350 .uf2 file

```
$ ~/.pico-sdk/picotool/2.0.0/picotool/picotool \
uf2 convert main.bin rp2350.uf2 \
--family absolute --offset 0x10000000
```

STEP 3 - Combined .uf2 file

```
$ cat rp2040.uf2 rp2350.uf2 > main.uf2
```

Running on Pico 2

```
I'm an RP2350 running ARM
I'm an RP2350 running ARM
I'm an RP2350 running ARM
I'm an RP2350 running ARM
I'm an RP2350 running ARM
I'm an RP2350 running ARM
I'm an RP2350 running ARM
I'm an RP2350 running ARM
I'm an RP2350 running ARM
I'm an RP2350 running ARM
Rebooting to other architecture
I'm an RP2350 running RISC-V
I'm an RP2350 running RISC-V
I'm an RP2350 running RISC-V
I'm an RP2350 running RISC-V
I'm an RP2350 running RISC-V
I'm an RP2350 running RISC-V
I'm an RP2350 running RISC-V
I'm an RP2350 running RISC-V
I'm an RP2350 running RISC-V
I'm an RP2350 running RISC-V
Rebooting to other architecture
```

Example

https://github.com/raspberrypi/pico-examples/tree/master/universal

Universal

These are examples of how to build universal binaries which run on RP2040, and RP2350 Arm & RISC-V. These require you to set `PICO_ARM_TOOLCHAIN_PATH` and `PICO_RISCV_TOOLCHAIN_PATH` to appropriate paths, to ensure you have compilers for both architectures.

App	Description
blink	Same as the blink example, but universal.
hello_universal	The obligatory Hello World program for Pico (USB and serial output). On RP2350 it will reboot to the other architecture after every 10 prints.
nuke_universal	Same as the nuke example, but universal. On RP2350 runs as a packaged SRAM binary, so is written to flash and copied to SRAM by the bootloader

Summary

- Universal binaries work because of the new boot loader in the RP2350:
  - On RP2040 the bootrom will just execute the RP2040 binary at the start of flash.
  - On RP2350 the bootrom will search the block loop for the appropriate Arm or RISC-V image and boot from there.


@GaryExplains