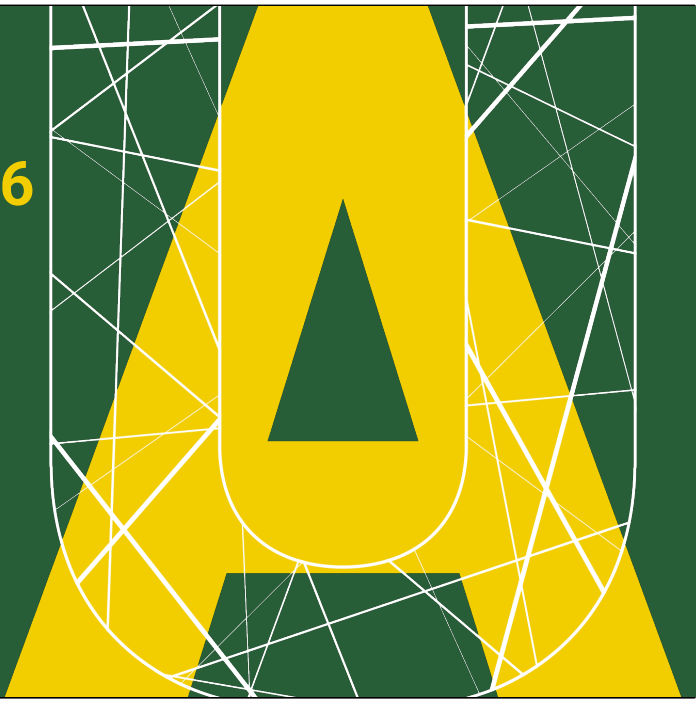
ECE 421 Lecture 16

Embedded Rust on Raspberry Pi Pico

Authors
Steven Knudsen, PhD, PEng
Ron Unrau, PhD

UNIVERSITY OF ALBERTA



This Lecture — Embedded Rust on a Pico

- Set up for development and test on Pico
- Embedded Hello World - ***blink***
- Debugging Rust

Main References

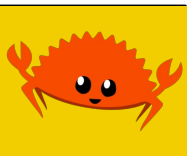
https://datasheets.raspberrypi.com/pico/getting-started-with-pico.pdf

https://github.com/StevenKnudsen

Design Patterns aka Gang of Four book is available for reading online at UofA Library. The others are not.

One can be found

Outline



1. Install VSCode and set up
2. Setup a Pico as debug probe
3. Debug an example project (blink)
4. Create an ADC program and debug

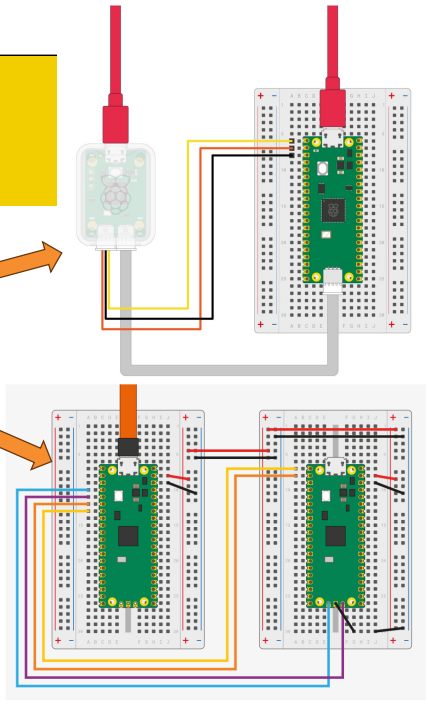
Will only highlight key points from the tutorials and any deviations

How to debug

- Use the Serial Wire Debug (SWD)
- Serial Wire Debug (SWD) is a 2-pin (SWDIO/SWCLK) electrical alternative Joint Test Action Group (JTAG) interface
- JTAG protocol used on top of the physical interface.
- SWD uses an ARM CPU standard bi-directional wire protocol
- Enables the debugger to become another Advanced Microcontroller Bus Architecture (AMBA) bus master for access to system memory and peripheral or debug registers

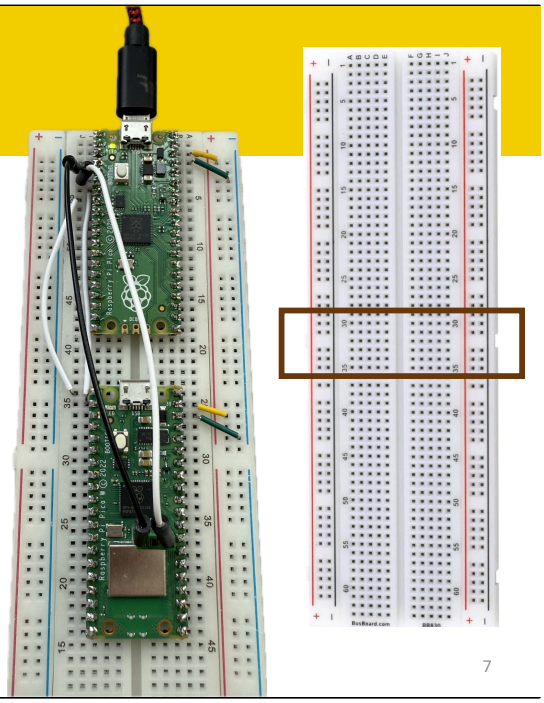
How to debug cont'd

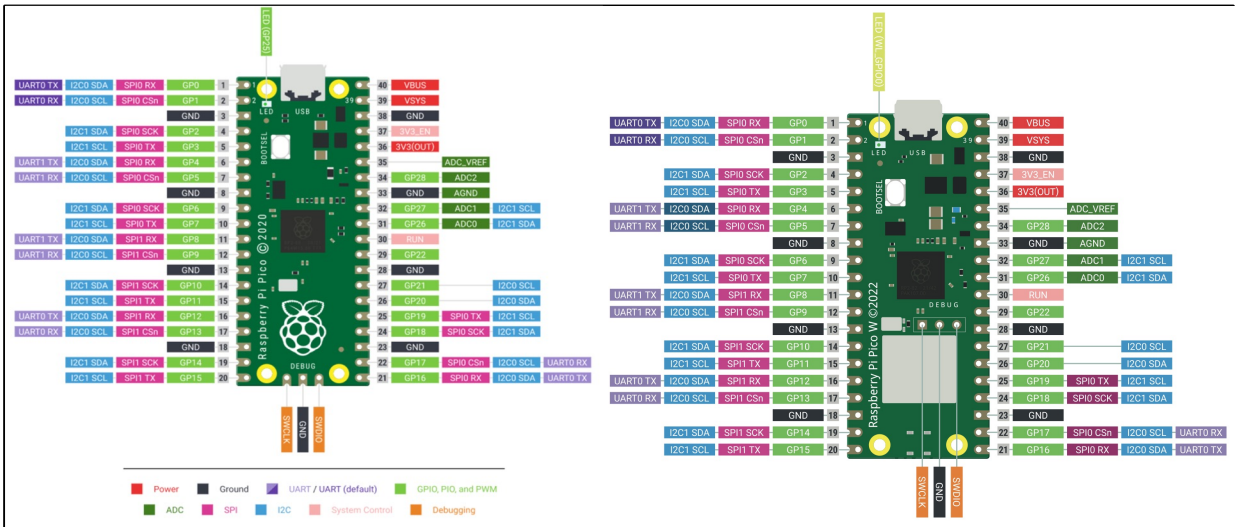
- Need a device to run the SWD. Either
  - (new) Raspberry Pi probe, or
  - Second Pico configured for SWD



How to debug reviewed

- Need a device to run the SWD. Either
  - (new) Raspberry Pi probe, or
  - Second Pico configured for SWD
- Here, target Pico is a Pico W
- B&W wires are SWD




Raspberry Pi Pico ©2020
Raspberry Pi Pico W ©2022
Power
Ground
UART / UART (default)
GPIO, PIO, and PWM
ADC
SPI
I2C
System Control
Debugging

Make sure Rust is up to date

- Visit rust-lang.org/tools/install to install rustup
- Tool that installs and manages Rust
- Rust is developed on a 6-week sprint, good to update often

```
rustup self update
rustup update stable
```

rp-rs

- Github project that supports a Rust HAL (hardware abstraction layer) that enabled embedded programming in Rust on the Raspberry Pi Pico family

```
rustup target add thumbv6m-none-eabi
cargo install flip-link
```

- May have dependency issues. Ubuntu 24.04 needed libudev-dev installed, for example

VSCode

- Install rust-analyzer, Debugger for probe-rs
- Install crates — Keeps crates up to date

- Get `debugprobe_on_pico.u2f` from

  github.com/raspberrypi/debugprobe/releases

  - Drag and drop programming with pico in bootloader mode

Probe-rs — effectively gdb

- Use probe-rs
- Check for pico probe and show output
- Could use `cargo install --locked probe-rs-tools`, but instead

```
$ curl --proto '=https' --tlsv1.2 -LsSf https://github.com/probe-rs/probe-
rs/releases/latest/download/probe-rs-tools-installer.sh | sh

$ probe-rs list
The following debug probes were found:
[0]: Debugprobe on Pico (CMSIS-DAP) – 2e8a:000c:E6616407E33CA42F (CMSIS-DAP)
```

- Be sure to set up `udev` rules in Linux

Blinky Program Example

- Borrow from Murray Todd on Medium

  https://murraytodd.medium.com/learning-rust-with-embedded-programming-on-rp2040-e784389d2d3d

- Use hardware abstraction layer (HAL) from rp-rs project

  https://github.com/rp-rs

-

Blinky

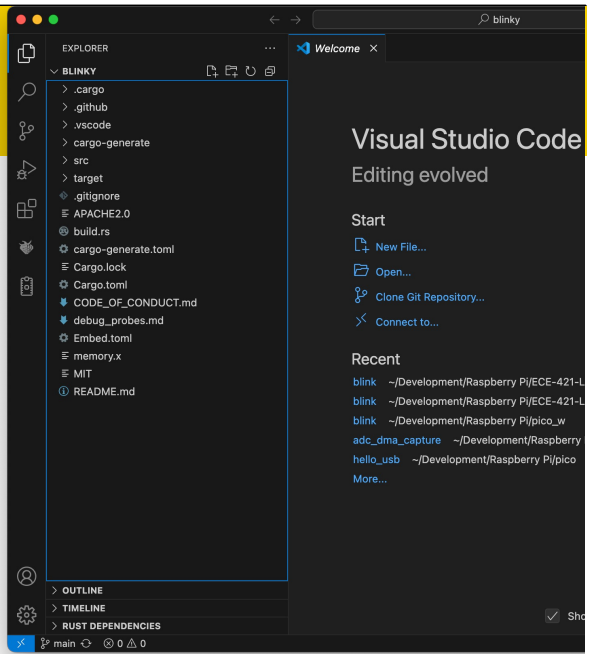
- Go to rp2040-project-template, clone

```
git clone https://github.com/rp-rs/rp2040-project-template.git
```

- Might as well rename the project template folder to `blinky`

VSCode work

- Open the `blinky` folder in VSCode
- We will be working in some of the folders to modify the template defaults



VSCode work

- Check in .cargo/config.toml that

```
runner = "probe-rs run --chip RP2040 --protocol swd”
```

- In Cargo.toml rename the project to `blinky`
- In .vscode/launch.json change “name” to `blinky_pwm` (line 9) and on (line 26)

```
"programBinary": "target/thumbv6m-none-eabi/debug/rp2040-project-template”
```

to

```
"programBinary": "target/thumbv6m-none-eabi/debug/blinky"
```

VSCode work cont'd

- In .vscode/launch.json, uncomment the line

```
"svdFile": "./.vscode/rp2040.svd"
```

- Manually download `rp2040.svd` from

```
curl -O https://raw.githubusercontent.com/raspberrypi/pico-sdk/1.3.1/src/rp2040/hardware_regs/rp2040.svd
```

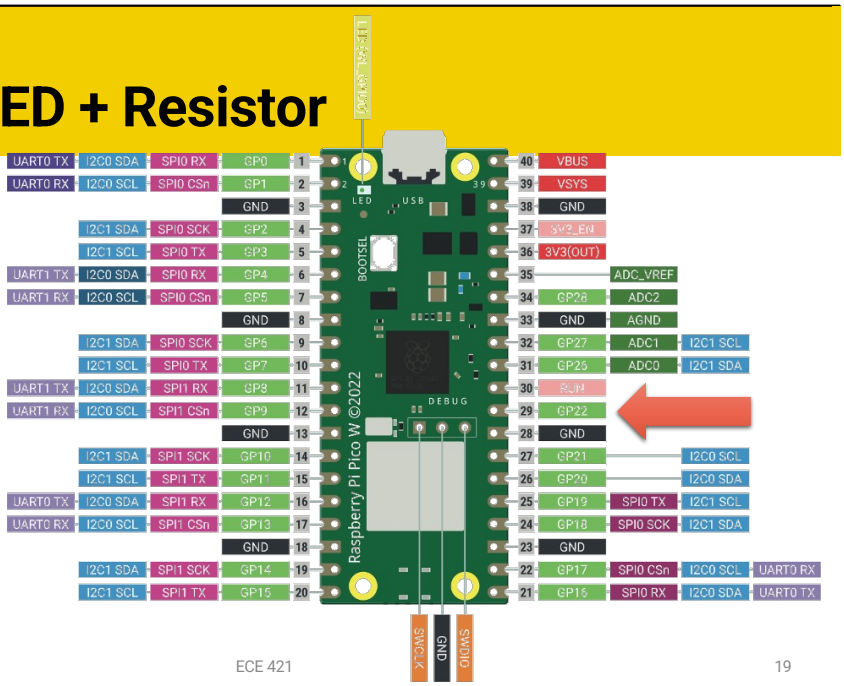
  and place it in the `.vscode` directory so that the debugger can find it

VSCode work cont'd

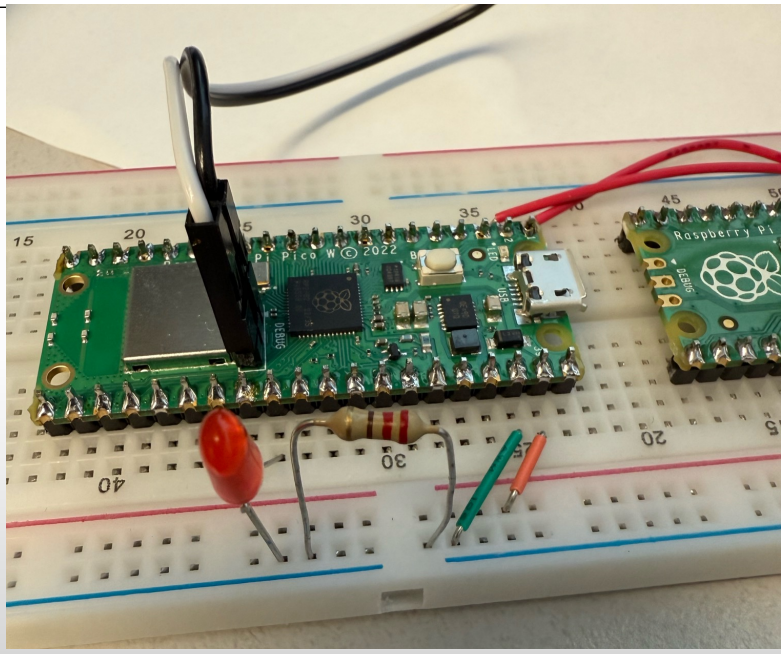
- In VSCode, install the `Debugger for probe-rs` extension
- Time to run the debugger!

Need to add LED + Resistor

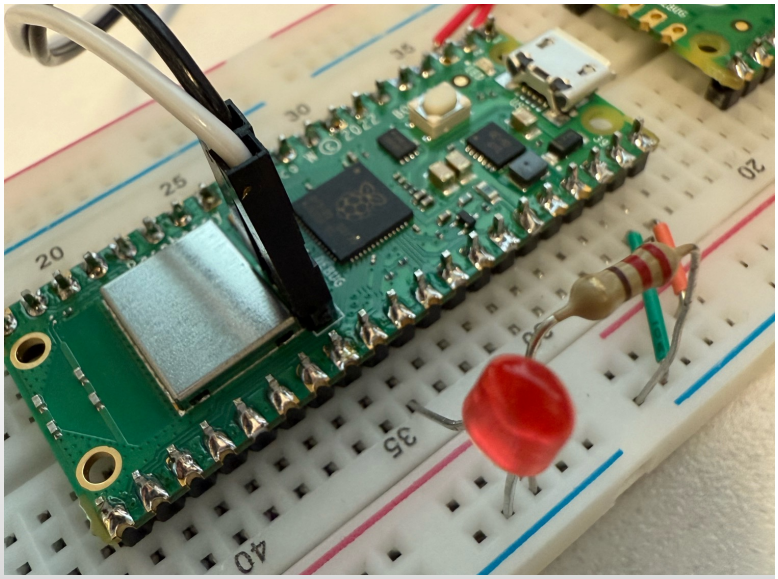
- We need to add an LED and resistor
- The red—blue rails of the breadboard are not connected in the middle.
- Connect the blue rails using the resistor



- The red—blue rails of the breadboard are not connected in the middle.
- Connect the blue rails using the resistor

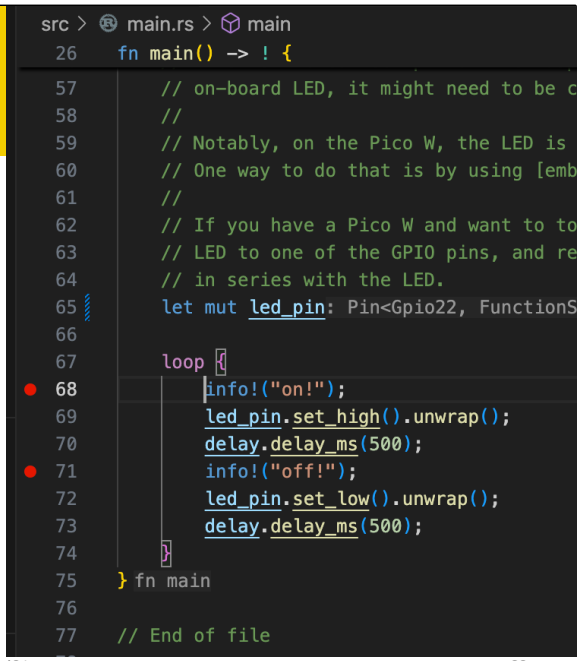


- Put the long lead of the LED in the breadboard at GPIO22 pin's location
- Put the short lead in the blue rail



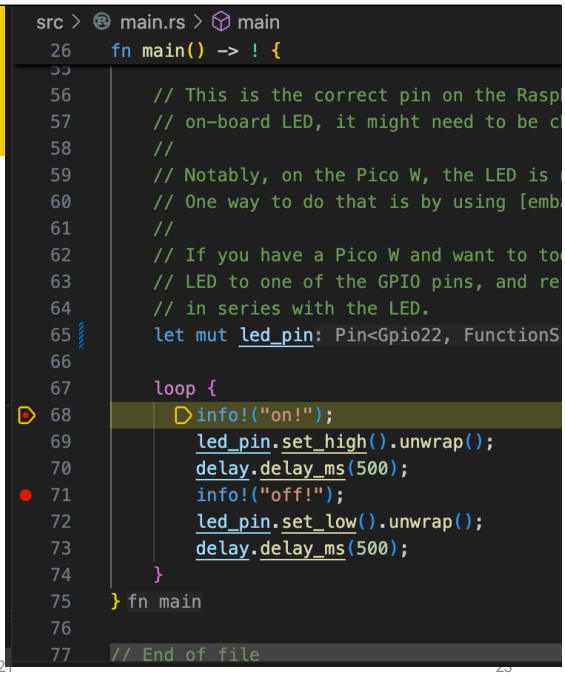
Set breakpoints

- Set breakpoints on the `info!()` statements
- Select Run/Start Debugging



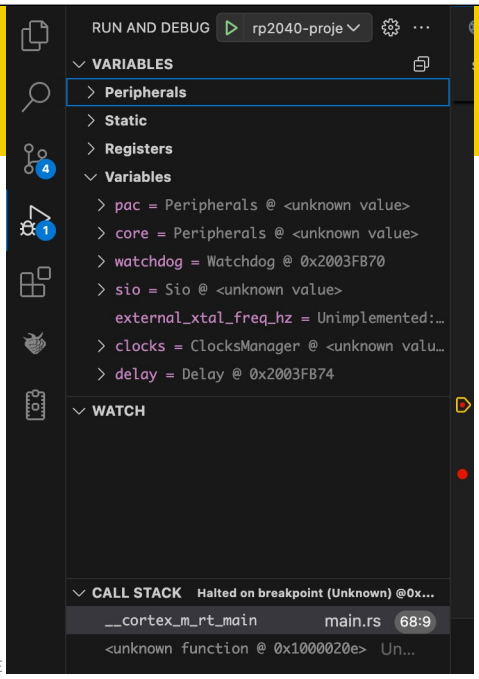
Stepping

- Ta Da
- Can’t set breakpoints on some statements, which is probably a compilation setting I am missing…



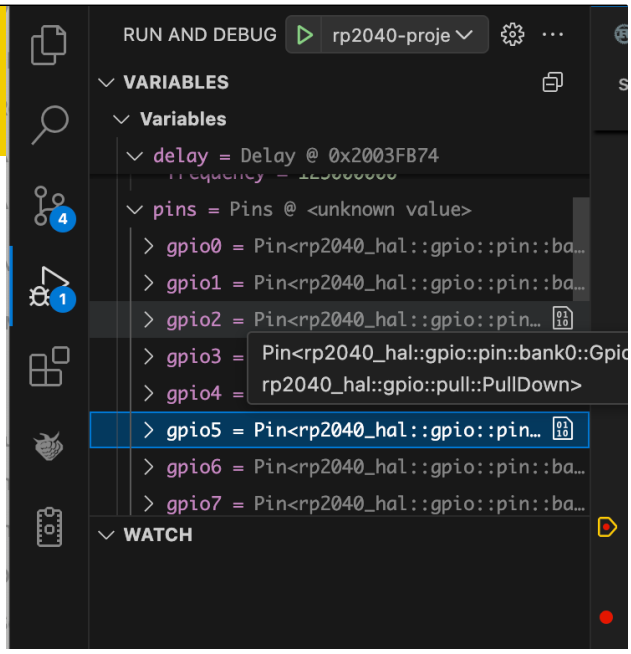
Information

- In debug mode we have access to the Peripherals via the Hardware Abstraction Layer (HAL)
- All kinds of interesting statics
- Scroll down the variables to the pins and pop it open

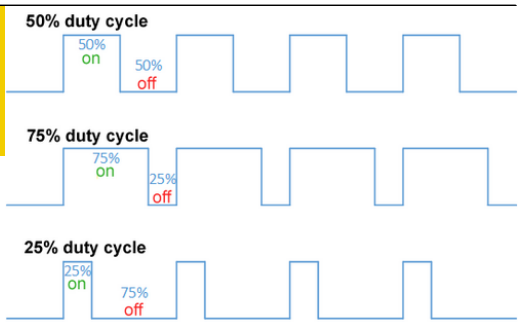


Variables - pins

- All kinds of information is exposed via the HAL



Blinky with PWM



- We can use Pulse Width Modulation to change the intensity of the LED
- Go to rp2040-project-template, clone

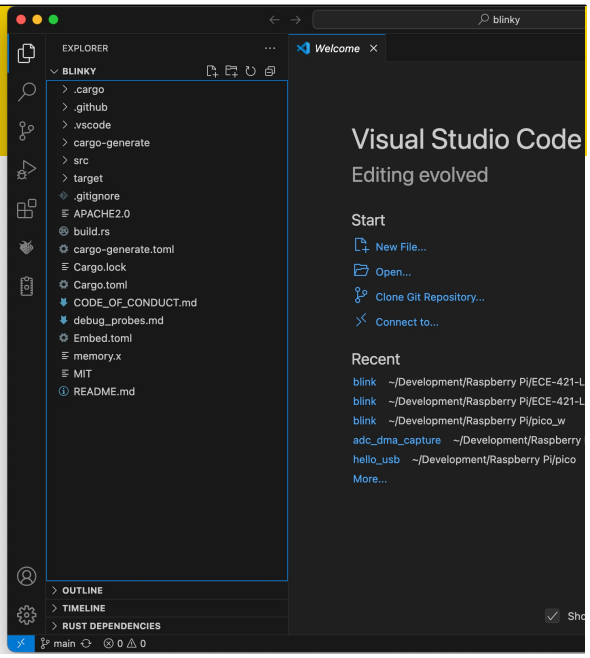
```
git clone https://github.com/rp-rs/rp2040-project-template.git
```

- Rename the project template folder to `blinky_pwm`

Wikipedia contributors. "Pulse-width modulation." *Wikipedia, The Free Encyclopedia*. Wikipedia, The Free Encyclopedia, 24 Mar. 2025. Web. 28 Mar. 2025.

VSCode work

- Open the `blinky` folder in VSCode
- We will be working in some of the folders to modify the template defaults



VSCode work

- Check in .cargo/config.toml that

```
runner = "probe-rs run --chip RP2040 --protocol swd”
```

- In Cargo.toml rename the project to `blinky_pwm`
- In .vscode/launch.json change “name” to `blinky_pwm` (line 9) and on (line 26)

```
"programBinary": "target/thumbv6m-none-eabi/debug/rp2040-project-template”
```

to

```
"programBinary": "target/thumbv6m-none-eabi/debug/blinky_pwm"
```

VSCode work cont'd

- In .vscode/launch.json, uncomment the line

```
"svdFile": "./.vscode/rp2040.svd"
```

- Manually download `rp2040.svd` from

```
curl -O https://raw.githubusercontent.com/raspberrypi/pico-sdk/1.3.1/src/rp2040/hardware_regs/rp2040.svd
```

  and place it in the `.vscode` directory so that the debugger can find it

VSCode work cont'd

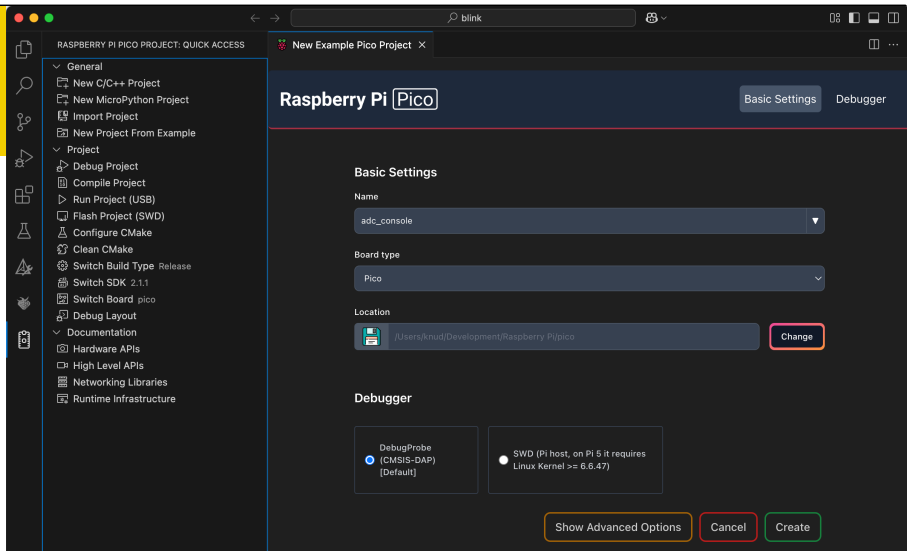
- In VSCode, install the `Debugger for probe-rs` extension (should be done already)
- Make sure we have panic-halt : `cargo add panic-halt`
- Run `main.rs` with downloaded `main.rs`
- Time to run the program!
  - Won't use the debugger, so just select Run/Run Without Debugging

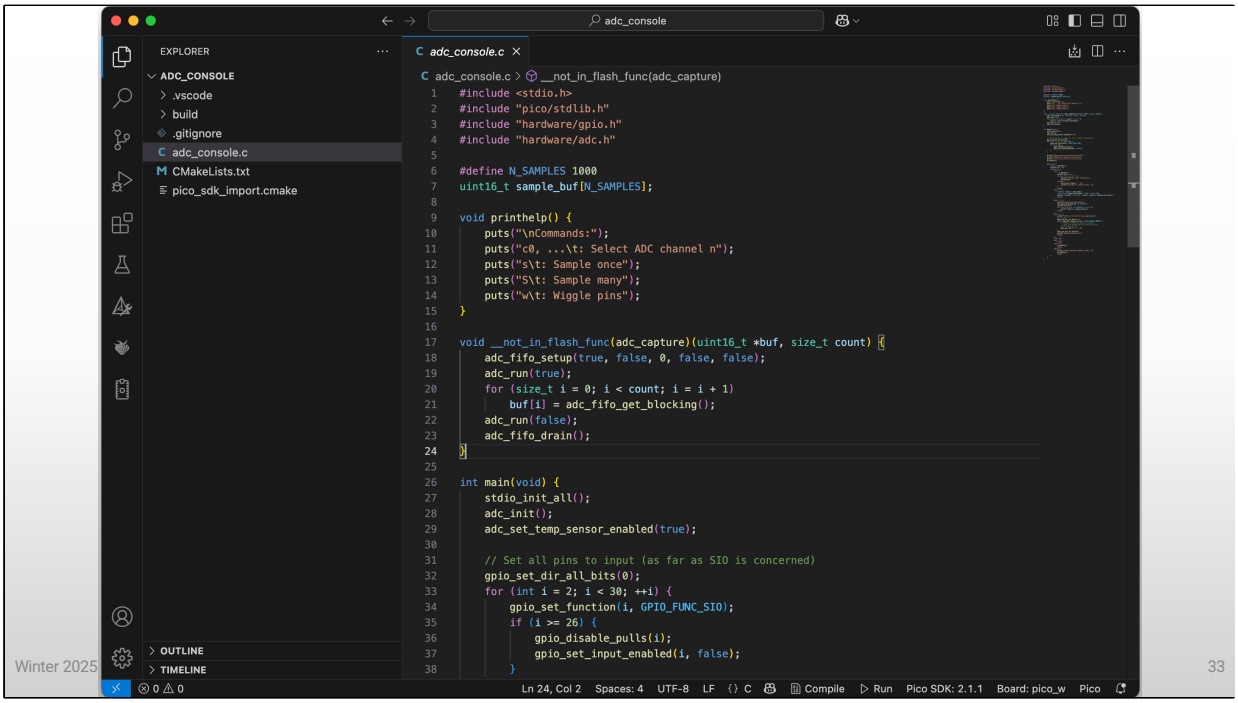
Questions

https://www.monkeyuser.com/2018/focus/?ref=comic

ADC

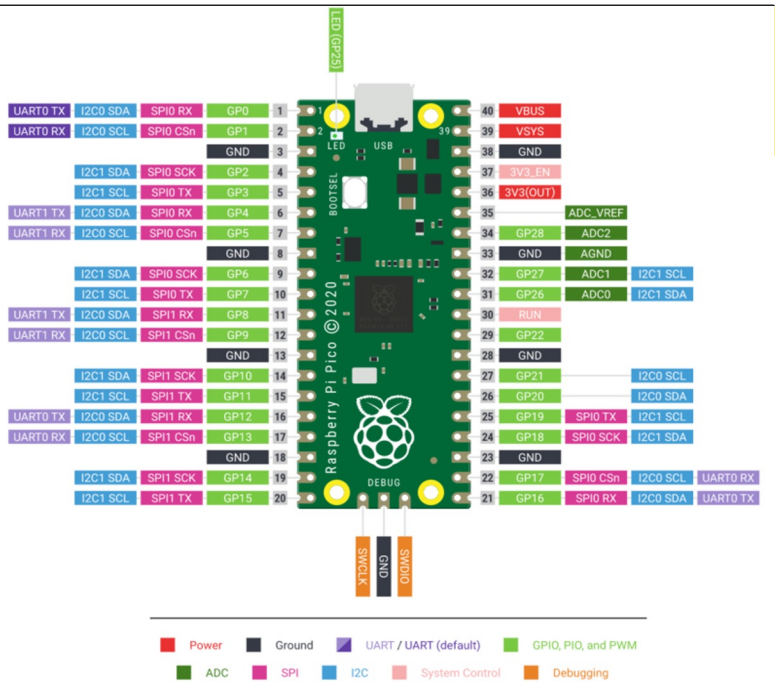
- Let’s try another example.
- Analogue to Digital converter
- Note, back to the Pico, not Pico W




adc_console
EXPLORER
ADC_CONSOLE
.vscode
build
.gitignore
adc_console.c
CMakeLists.txt
pico_sdk_import.cmake
OUTLINE
TIMELINE
adc_console.c
adc_console.c > __not_in_flash_func(adc_capture)
#include <stdio.h>
#include "pico/stdlib.h"
#include "hardware/gpio.h"
#include "hardware/adc.h"
#define N_SAMPLES 1000
uint16_t sample_buf[N_SAMPLES];
void printhelp() {
puts("\nCommands:");
puts("c0, ...\t: Select ADC channel n");
puts("s\t: Sample once");
puts("S\t: Sample many");
puts("w\t: Wiggle pins");
}
void __not_in_flash_func(adc_capture)(uint16_t *buf, size_t count) {
adc_fifo_setup(true, false, 0, false, false);
adc_run(true);
for (size_t i = 0; i < count; i = i + 1)
buf[i] = adc_fifo_get_blocking();
adc_run(false);
adc_fifo_drain();
}
int main(void) {
stdio_init_all();
adc_init();
adc_set_temp_sensor_enabled(true);
// Set all pins to input (as far as SIO is concerned)
gpio_set_dir_all_bits(0);
for (int i = 2; i < 30; ++i) {
gpio_set_function(i, GPIO_FUNC_SIO);
if (i >= 26) {
gpio_disable_pulls(i);
gpio_set_input_enabled(i, false);
}
0 0
Ln 24, Col 2
Spaces: 4
UTF-8
LF
C
Compile
Run
Pico SDK: 2.1.1
Board: pico_w
Pico

Can’t debug

- The program relies on terminal I/O
  - Asks for the ADC pin to use
- Where are ADC channels?
- Pin 31, 32, 34
- Pin 33 is AGND



UNIVERSITY OF ALBERTA

VSCode work cont'd

- In Cargo.toml, uncomment the two lines with `rp2040*`

```
# If you're not going to use a Board Support Package you'll need these:
rp2040-hal = { version="0.10", features=["rt", "critical-section-impl"] }
rp2040-boot2 = "0.3"
```

  so that the HAL is included

- In VSCode, install the `Debugger for probe-rs` extension
- Time to run the debugger!