

ECE 421 Lecture 15

Embedded C/C++ on Raspberry Pi Pico

Authors

Steven Knudsen, PhD, PEng

Ron Unrau, PhD



UNIVERSITY
OF ALBERTA



This Lecture – Embedded C/C++ on a Pico

- What is the Pico?
- Why do I care?
- Set up for development and test on Pico
- Embedded Hello World - *blink*
- Debugging C/C++



UNIVERSITY
OF ALBERTA

Winter 2025

ECE 421

2

Main References

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

<https://github.com/StevenKnudsen>



UNIVERSITY
OF ALBERTA

Winter 2025

ECE 421

3

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

Raspberry Pi Pico and Pico W

Raspberry Pi Pico is a low-cost, high-performance microcontroller board with flexible digital interfaces. Key features include:

- RP2040 microcontroller chip designed by Raspberry Pi in the United Kingdom
- Dual-core Arm Cortex M0+ processor, flexible clock running up to 133 MHz
- 264 kB of SRAM, and 2MB of on-board flash memory
- USB 1.1 with device and host support
- Low-power sleep and dormant modes
- Drag-and-drop programming using mass storage over USB
- 26 × multi-function GPIO pins
- 2 × SPI, 2 × I2C, 2 × UART, 3 × 12-bit ADC, 16 × controllable PWM channels
- Accurate clock and timer on-chip
- Temperature sensor
- Accelerated floating-point libraries on-chip
- 8 × Programmable I/O (PIO) state machines for custom peripheral support

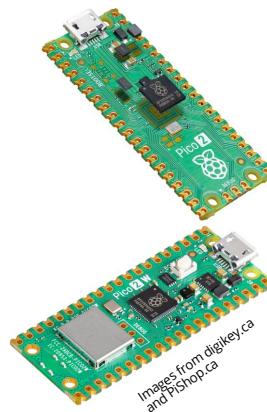


Images from digikey.ca

Raspberry Pi Pico 2 and Pico 2 W

Raspberry Pi Pico 2 updates the Pico with:

- Dual Arm Cortex-M33 or dual Hazard3 RISC-V processors @ 150MHz
- 520 KB on-chip SRAM
- 2.4GHz 802.11n wireless LAN and Bluetooth 5.2 (Raspberry Pi Pico 2 W only)
- Software- and hardware-compatible with Raspberry Pi Pico 1
- Robust and fully documented security features:
 - ARM TrustZone for Cortex-M
 - Optional boot signing, enforced by on-chip mask ROM, with key fingerprint in OTP
 - Protected OTP storage for optional boot decryption key
 - Global bus filtering based on Arm or RISC-V security/privilege levels
 - Peripherals, GPIOs, and DMA channels individually assignable to security domains
 - Hardware mitigations for fault injection attacks
 - Hardware SHA-256 accelerator
- 2 × SPI, 2 × I2C, 2 × UART, 24 × PWM, 3 × 12-bit ADC
- 1 × USB 1.1 controller and PHY, with host and device support
- 12 × PIO state machines
- Open-source C/C++ SDK, MicroPython support



Images from digikey.ca
and PiShop.ca

Pico & Pico W

Pico 2 & Pico 2 W

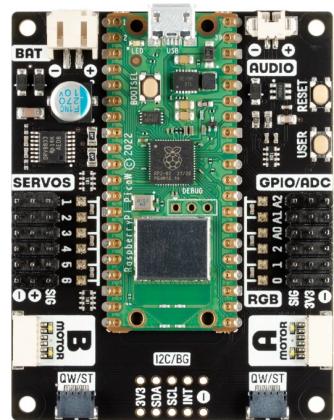


\$CA6.10 & \$CA9.15

at mouser.ca

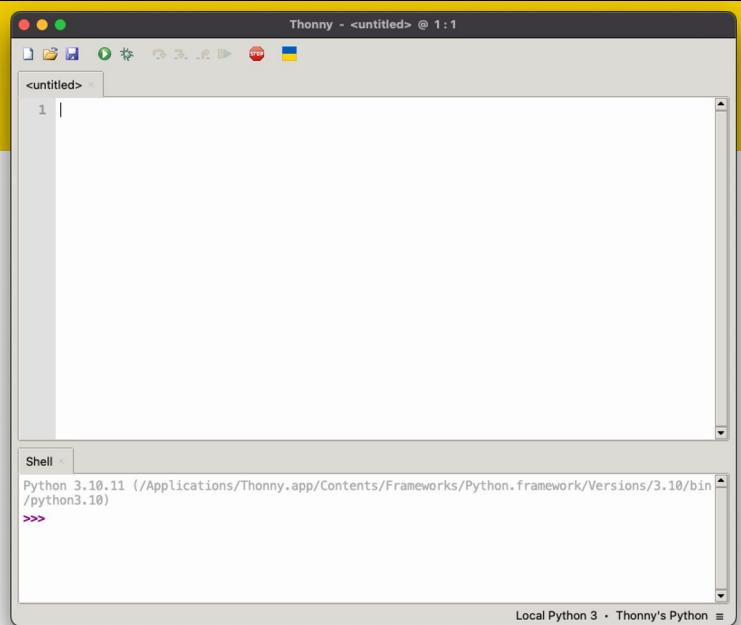
\$CA7.63 & \$CA10.68

Pico will be core for Mechatronics



MicroPython

- Usual starting point is to use MicroPython
- Dead easy to use and program
- Not that interesting for us



UNIVERSITY
OF ALBERTA

Winter 2025

ECE 421

9

I have a different tutorial on my github in a couple of HackED workshop repos

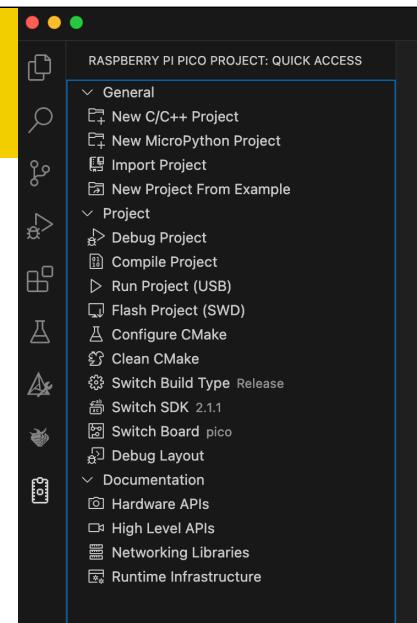
VSCode Setup

- Assume you know and use VSCode
- Install *Raspberry Pi Pico* extension
- Environment dependencies;
 - Windows – none
 - macOS – `xcode-select --install`
 - Linux – `sudo apt install python3 git tar build-essential`



New Project from Example

- Click Pico icon on the left to get project quick access pane
- Click *New Project From Example*



The screenshot shows the Raspberry Pi Pico Project Manager interface. On the left, there's a yellow sidebar with the word "blink". The main area has a dark theme with a sidebar on the left containing various project-related icons and options like "New C/C++ Project", "New MicroPython Project", "Import Project", "New Project From Example", "Project", "Debug Project", "Run Project (USB)", "Flash Project (SWD)", "Configure CMake", "Clean CMake", "Switch Build Type Release", "Switch SDK 2.1.1", "Switch Board pico", "Debug Layout", "Documentation", "Hardware APIs", "High Level APIs", "Networking Libraries", and "Runtime Infrastructure". The central part of the screen shows a "Raspberry Pi Pico" configuration window. It has two tabs: "Basic Settings" and "Debugger". In "Basic Settings", the "Name" field is set to "blink", "Board type" is set to "Pico", and the "Location" field shows the path "/Users/knud/Development/Raspberry Pi/pico". A "Change" button is highlighted with a red box. In the "Debugger" tab, there are two options: "BlinkProbe (CMSIS-DAP) [Default]" and "SWD (Pi host, on Pi 5 it requires Linux Kernel >= 6.6.47)". The "Create" button at the bottom right of this window is also highlighted with a red box. Below the configuration windows, there's a terminal window titled "TERMINAL" showing command-line output related to the project build and flash process. At the bottom of the screen, there are buttons for "Compile", "Run", "Pico SDK: 2.1.1", and "Board: pico".

• Choose the example
 • Set board type
 • Pico
 • Set location
 • Leave Debugger default
 • Click Create

Raspberry Pi Pico

Basic Settings

Name: blink

Board type: Pico

Location: /Users/knud/Development/Raspberry Pi/pico

Debugger

BlinkProbe (CMSIS-DAP) [Default] SWD (Pi host, on Pi 5 it requires Linux Kernel >= 6.6.47)

Show Advanced Options Cancel Create

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS MEMORY XRTOS SERIAL MONITOR + v ... x

Executing task: /Users/knud/.pico-sdk/picotool/2.1.1/picotool picotool load /Users/knud/Development/Raspberry Pi/pico/blink/build/blink.elf -fx

Loading into Flash: [=====] 100%

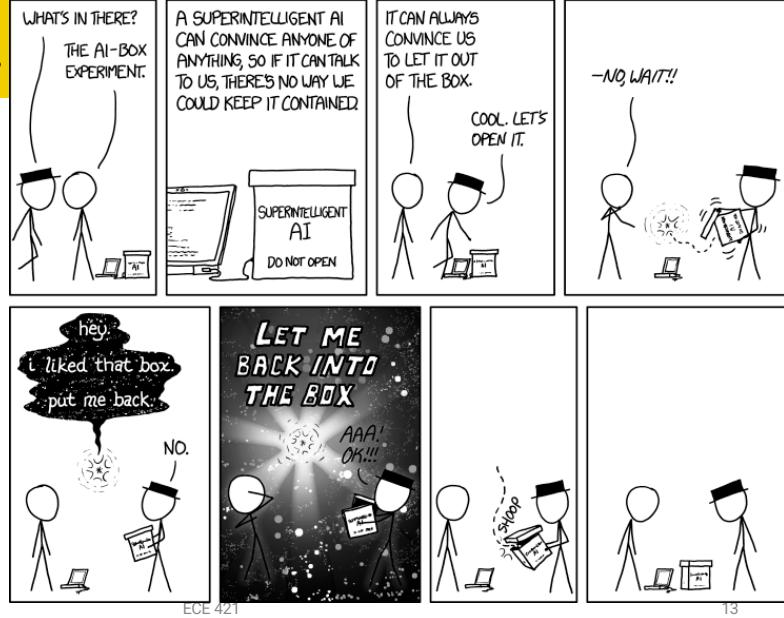
The device was rebooted to start the application.

The terminal will be reused by tasks, press any key to close it.

ECE 421

12

While you wait...



- Notice the convenience items at the bottom

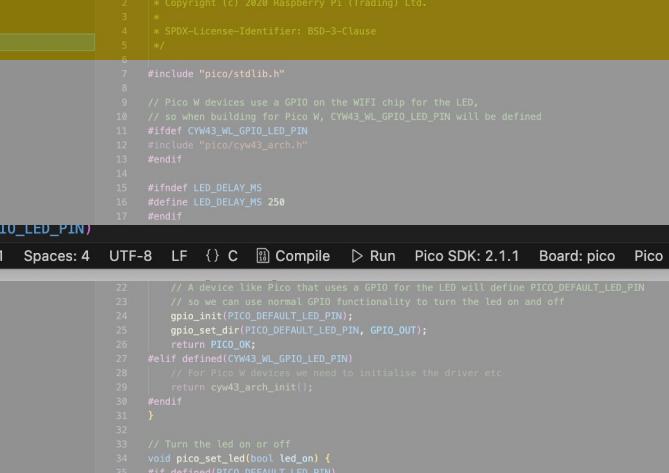
```

C blink.c > ...
1  /**
2   * Copyright (c) 2020 Raspberry Pi (Trading) Ltd.
3   *
4   * SPDX-License-Identifier: BSD-3-Clause
5   */
6
7  #include "pico/stl.h"
8
9  // Pico W devices use a GPIO on the WIFI chip for the LED,
10 // so when building for Pico W, CYW43_WL_GPIO_LED_PIN will be defined
11 #ifdef CYW43_WL_GPIO_LED_PIN
12 #include "pico/cyw43_arch.h"
13 #endif
14
15 #ifndef LED_DELAY_MS
16 #define LED_DELAY_MS 250
17 #endif
18
19 // Perform initialisation
20 int pico_led_init(void) {
21 #if defined(PICO_DEFAULT_LED_PIN)
22     // A device like Pico that uses a GPIO for the LED will define PICO_DEFAULT_LED_PIN
23     // so we can use normal GPIO functionality to turn the led on and off
24     gpio_init(PICO_DEFAULT_LED_PIN);
25     gpio_set_dir(PICO_DEFAULT_LED_PIN, GPIO_OUT);
26     return PICO_OK;
27 #elif defined(CYW43_WL_GPIO_LED_PIN)
28     // For Pico W devices we need to initialise the driver etc
29     return cyw43_arch_init();
30 #endif
31 }
32
33 // Turn the led on or off
34 void pico_set_ledbool(bool led_on) {
35 #if defined(PICO_DEFAULT_LED_PIN)
36     // Just set the GPIO on or off
37     gpio_put(PICO_DEFAULT_LED_PIN, led_on);
38 #elif defined(CYW43_WL_GPIO_LED_PIN)

```

Ln 1, Col 1 Spaces: 4 UTF-8 ⓘ C ⚒ Compile ▶ Run Pico SDK: 2.1.1 Board: pico Pico

- Notice the convenience items at the bottom



```
C blink.c x
C blink.c > ...
1  /*
2   * Copyright (c) 2020 Raspberry Pi (Trading) Ltd.
3   *
4   * SPDX-License-Identifier: BSD-3-Clause
5   */
6
7 #include "pico/stdlib.h"
8
9 // Pico W devices use a GPIO on the WIFI chip for the LED,
10 // so when building for Pico W, CYW43_WL_GPIO_LED_PIN will be defined
11 #ifndef CYW43_WL_GPIO_LED_PIN
12 #include "pico/cyw43_arch.h"
13 #endif
14
15 #ifndef LED_DELAY_MS
16 #define LED_DELAY_MS 250
17 #endif

W43_WL_GPIO_LED_PIN)
Ln 1, Col 1  Spaces: 4  UTF-8  LF  {}  C  Compile  ▶ Run  Pico SDK: 2.1.1  Board: pico  Pico  📡

22 // A device like Pico that uses a GPIO for the LED will define PICO_DEFAULT_LED_PIN
23 // so we can use normal GPIO functionality to turn the led on and off
24 gpio_init(PICO_DEFAULT_LED_PIN);
25 gpio_set_dir(PICO_DEFAULT_LED_PIN, GPIO_OUT);
26 return PICO_OK;
27 #elif defined(CYW43_WL_GPIO_LED_PIN)
28 // For Pico W devices we need to initialise the driver etc
29 return cyw43_arch_init();
30 #endif
31 }

32

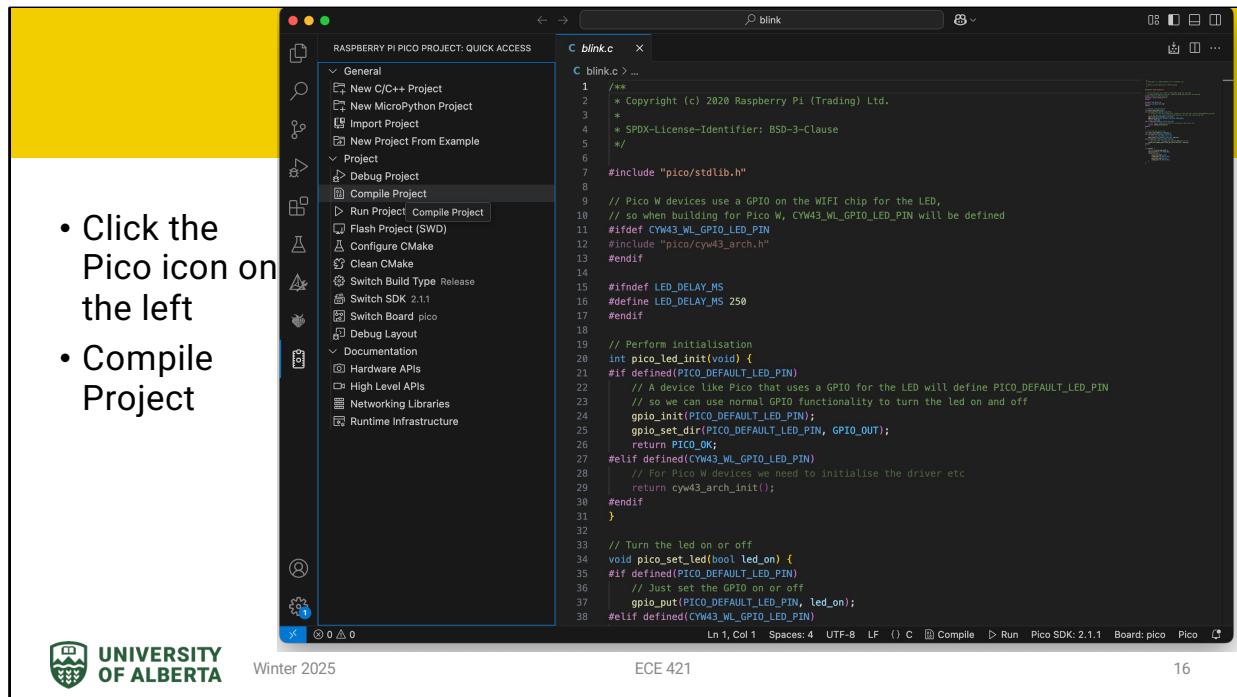
33 // Turn the led on or off
34 void pico_set_led(bool led_on) {
35 #if defined(PICO_DEFAULT_LED_PIN)
36     // Just set the GPIO on or off
37     gpio_put(PICO_DEFAULT_LED_PIN, led_on);
38 #elif defined(CYW43_WL_GPIO_LED_PIN)
```



Winter 2025

ECE 421

15



Pico in boot select (bootsel) mode

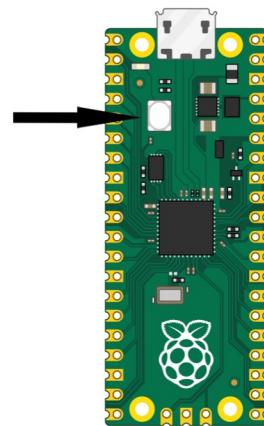
Unplug Pico from USB

Press and hold `bootsel` button

Plug Pico into USB and wait a few seconds

Release `bootsel` button

Pico appears as storage device

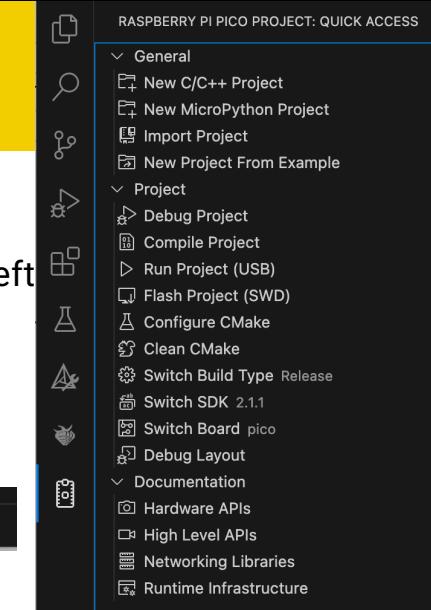


<https://projects.raspberrypi.org/en/projects/getting-started-with-the-pico/3>

Run it!

- With Pico in `BOOTSEL` mode,
- Click Run at bottom or Run Project (USB) on left
- Yay, a blinking LED

{ } C  Compile  Run Pico SDK: 2.1.1 Board: pico Pico

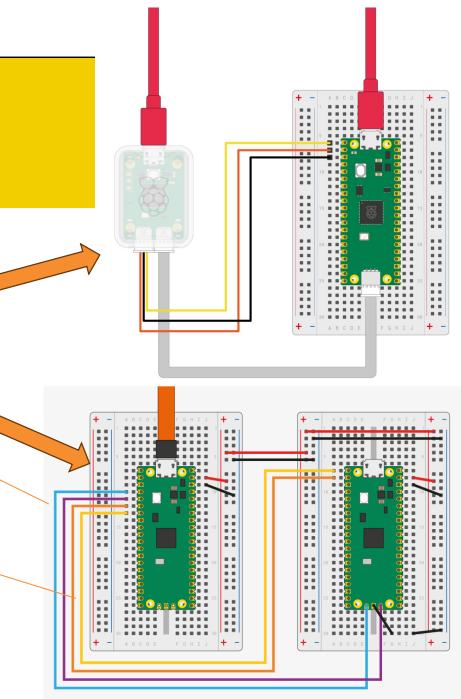


How to debug

- 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

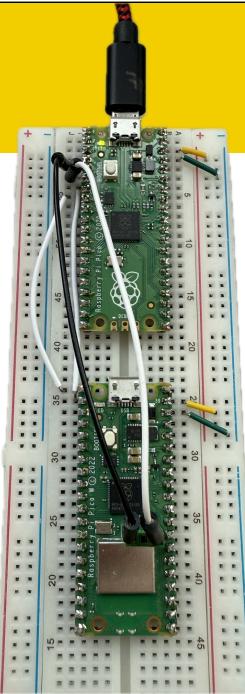
How to debug cont'd

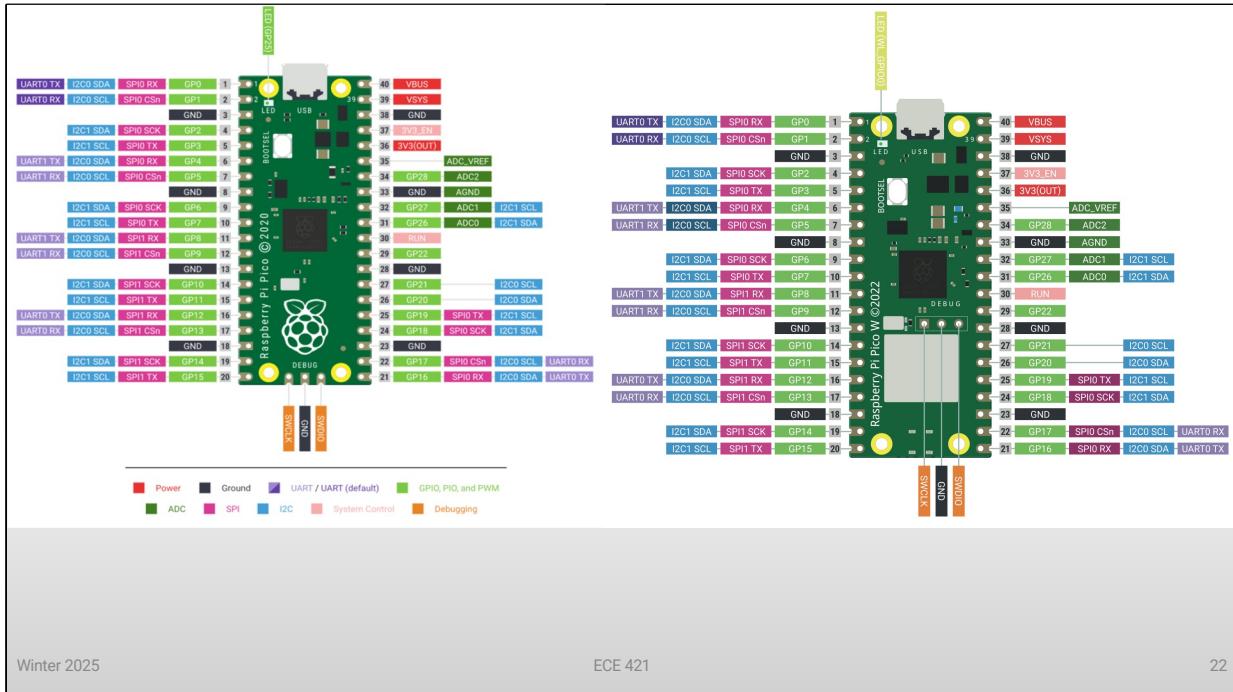
- Need a device to run the SWD. Either
 - (new) Raspberry Pi probe, or
 - Second Pico configured for SWD



How to debug cont'd

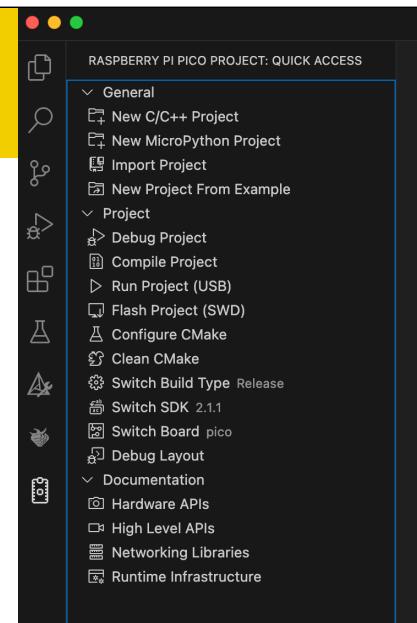
- Need a device to run the SWD. Either
 - (new) Raspberry Pi probe, or
 - Second Pico configured for SWD
- Here, target is Pico W
- B&W wires are SWD





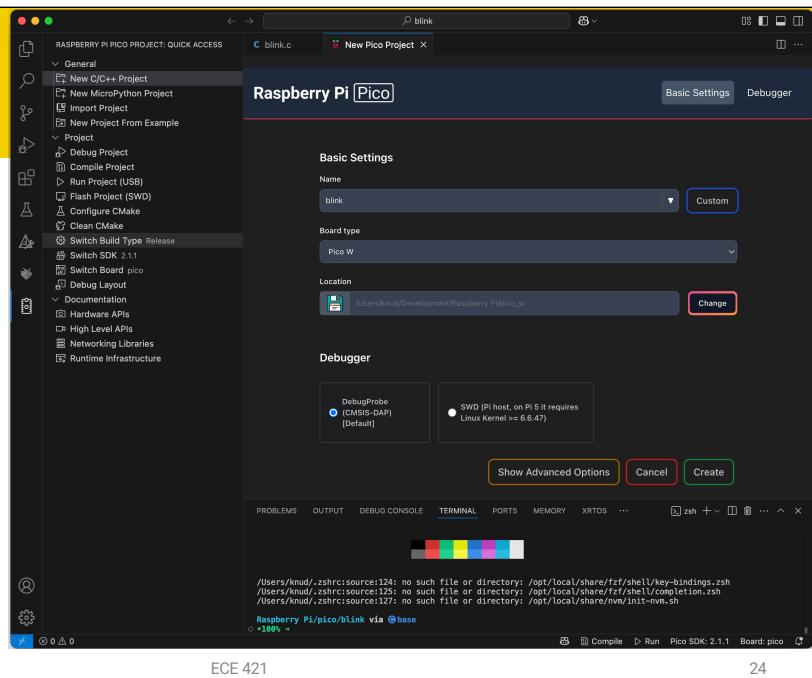
New Project from Example

- Need to select the Pico W as target
- So need new example
- Click Pico icon on the left to get project quick access pane
- Click *New Project From Example*

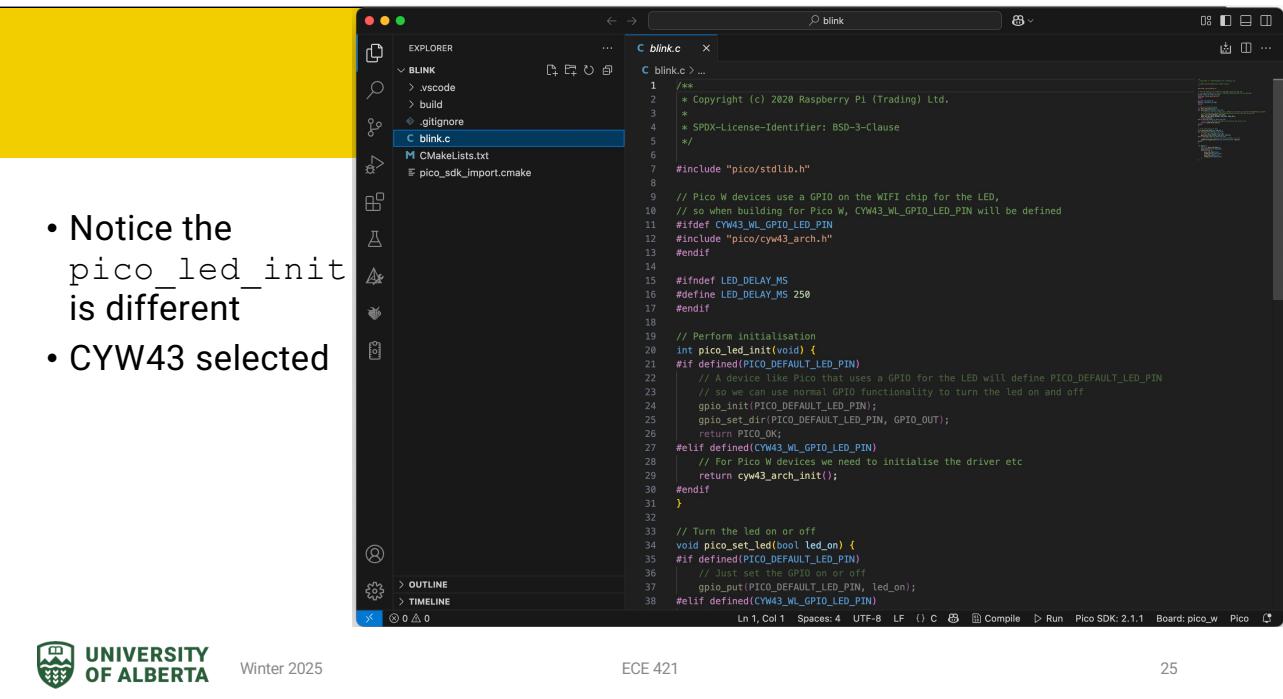


blink

- Choose the example
- Set board type
 - Pico W
- Set location
- Leave Debugger default
- Click Create



- Notice the `pico_led_init` is different
- CYW43 selected



```

C blink.c > ...
1  /*
2  * Copyright (c) 2020 Raspberry Pi (Trading) Ltd.
3  *
4  * SPDX-License-Identifier: BSD-3-Clause
5  */
6
7 #include "pico/stdlib.h"
8
9 // Pico W devices use a GPIO on the WIFI chip for the LED,
10 // so when building for Pico W, CYW43_WL_GPIO_LED_PIN will be defined
11 #ifndef CYW43_WL_GPIO_LED_PIN
12 #include "pico/cyw43_arch.h"
13 #endif
14
15 #ifndef LED_DELAY_MS
16 #define LED_DELAY_MS 250
17 #endif
18
19 // Perform initialisation
20 int pico_led_init(void) {
21 #if defined(PICO_DEFAULT_LED_PIN)
22     // A device like Pico that uses a GPIO for the LED will define PICO_DEFAULT_LED_PIN
23     // so we can use normal GPIO functionality to turn the led on and off
24     gpio.init(PICO_DEFAULT_LED_PIN);
25     gpio.set_dir(PICO_DEFAULT_LED_PIN, GPIO_OUT);
26     return PICO_OK;
27 #elif defined(CYW43_WL_GPIO_LED_PIN)
28     // For Pico W devices we need to initialise the driver etc
29     return cyw43_arch_init();
30 #endif
31 }
32
33 // Turn the led on or off
34 void pico_set_led(bool led_on) {
35 #if defined(PICO_DEFAULT_LED_PIN)
36     // Just set the GPIO on or off
37     gpio.put(PICO_DEFAULT_LED_PIN, led_on);
38 #elif defined(CYW43_WL_GPIO_LED_PIN)

```

Ln 1 Col 1 Spaces: 4 UTF-8 LF ⌂ C ⌂ Compile ⌂ Run Pico SDK: 2.1.1 Board: pico_w Pico ⌂

Configure Pico debugprobe

- <https://github.com/raspberrypi/debugprobe/releases/tag/debugprobe-v2.2.1>
- [debugprobe_on_pico.uf2](#)
- Put Pico into storage mode (hold BOOTSEL while applying power)
- Drag and drop the `uf2` file to program it
- For good measure, power it off/on

Time to debug...

- Select Debug Project or press F5.
- If prompted to select a debugger, choose Pico Debug (Cortex-Debug)
- The program will start, connected to the debugger, and stop



UNIVERSITY
OF ALBERTA

Winter 2025

ECE 421

27

The screenshot shows a debugger interface with the following details:

- File:** blink.c
- Function:** pico_led_init(void)
- Breakpoint:** Set at line 42 (pico_set_dir).
- Call Stack:** rp2040.core0 (Paused on Breakpoint)
- Variables:** Local, Global, Registers
- Watch:** No items listed.

```
int pico_led_init(void) {
    #if defined(PICO_DEFAULT_LED_PIN)
        // A device like Pico that uses a GPIO for the LED will define PICO_DEFAULT_LED_PIN
        // so we can use normal GPIO functionality to turn the led on and off
        gpio_init(PICO_DEFAULT_LED_PIN);
        gpio_set_dir(PICO_DEFAULT_LED_PIN, GPIO_OUT);
        return PICO_OK;
    #elif defined(CYW43_WL_GPIO_LED_PIN)
        // For Pico W devices we need to initialise the driver etc
        return cyw43_arch_init();
    #endif
}

// Turn the led on or off
void pico_set_led(bool led_on) {
    #if defined(PICO_DEFAULT_LED_PIN)
        // Just set the GPIO on or off
        gpio_put(PICO_DEFAULT_LED_PIN, led_on);
    #elif defined(CYW43_WL_GPIO_LED_PIN)
        // Ask the wifi "driver" to set the GPIO on or off
        cyw43_arch_gpio_put(CYW43_WL_GPIO_LED_PIN, led_on);
    #endif
}

int main() {
    int rc = pico_led_init();
    hard_assert(rc == PICO_OK);
    while (true) {
        pico_set_led(true);
        sleep_ms(LED_DELAY_MS);
        pico_set_led(false);
        sleep_ms(LED_DELAY_MS);
    }
}
```

Winter 2025

ECE 421

28

Notice the controls at the top and that we have stopped

The screenshot shows the Pico Debug interface with several controls highlighted by orange boxes:

- Reset device**: Located in the Variables panel.
- Step Over**: Located in the code editor toolbar.
- Step Out**: Located in the code editor toolbar.
- Stop**: Located in the code editor toolbar.
- Continue**: Located in the code editor toolbar.
- Step Into**: Located in the code editor toolbar.
- Restart**: Located in the code editor toolbar.

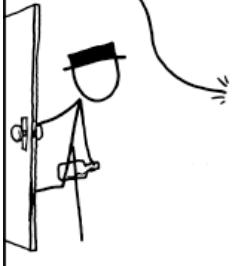
The code editor displays the file `blink.c` with the following content:

```
int main() {
    int rc = pico_led_init();
    hard_assert(rc == PICO_OK);
    while (true) {
        pico_set_led(true);
        sleep_ms(LED_DELAY_MS);
        pico_set_led(false);
        sleep_ms(LED_DELAY_MS);
    }
}
```

The status bar at the bottom indicates "Winter 2025", "ECE 421", and "29".

Notice the controls at the top and that we have stopped

SERIOUSLY? THIS THING RUNS JAVA?
IT'S SINGLE-PURPOSE HARDWARE!



I BET THEY ACTUALLY HIRED SOMEONE
TO SPEND SIX MONTHS PORTING THIS
JVM SO THEY COULD WRITE THEIR 20
LINES OF CODE IN A FAMILIAR SETTING.



WELL, YOU KNOW WHAT THEY SAY—
WHEN ALL YOU HAVE IS A PAIR OF
BOLT CUTTERS AND A BOTTLE OF VODKA,
EVERYTHING LOOKS LIKE THE LOCK ON
THE DOOR OF WOLF BLITZER'S BOATHOUSE.



<https://xkcd.com/801/>

Questions

FOCUS



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