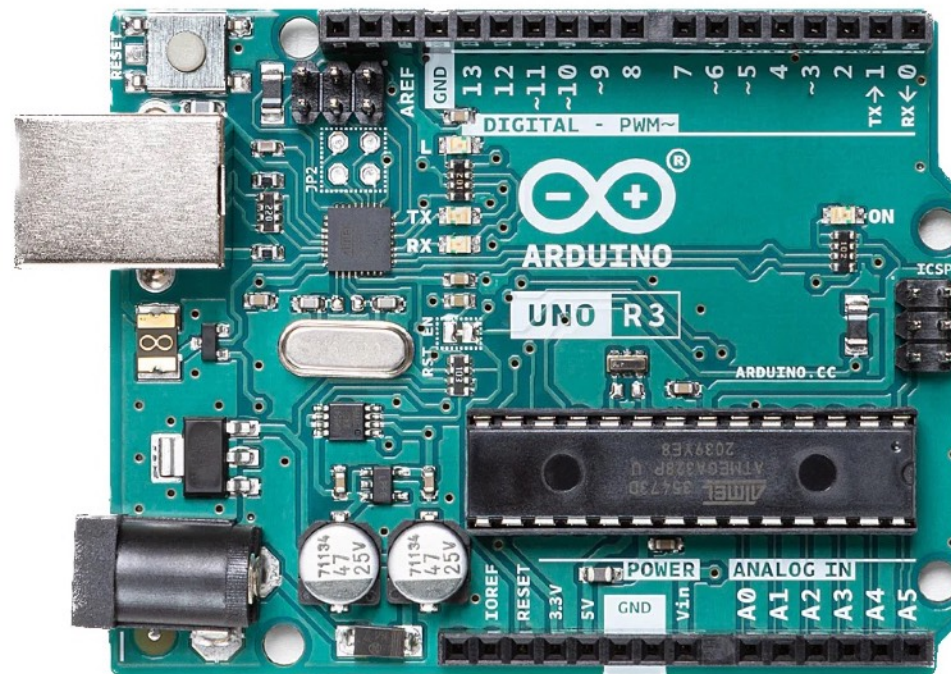
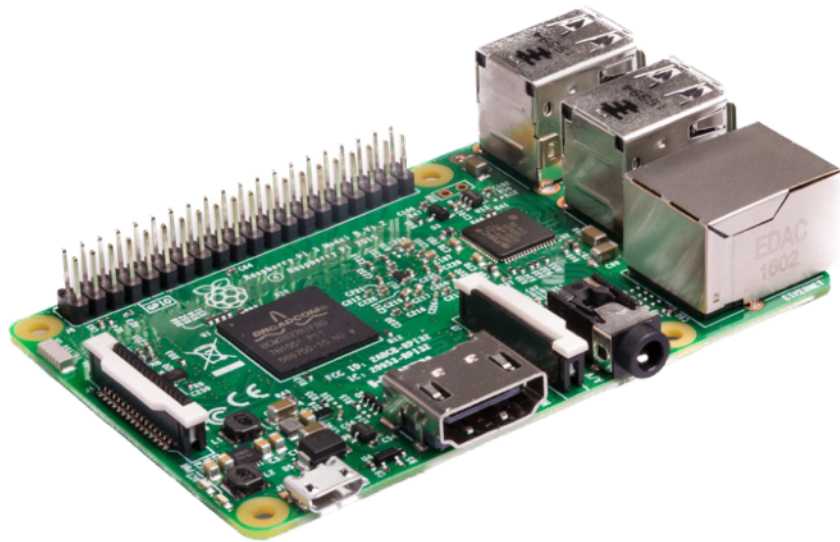


Embedded Controllers

Embedded Controllers

- Embedded control is the concept of using a microcontroller as the brains of your project



Each microcontroller choice has issues

- Processor complexity – number of bits for instruction set
 - Development complexity – language used to program
 - Documentation and Support – time in market and design goals
 - Execution Speed – clock speed of processor
 - Memory for storage and for execution – SRAM and Flash RAM
 - Power consumption – battery issue?
-
- Good news? Price is rarely an issue!

Simple Board Comparison

	Arduino Uno (2010)	Raspberry Pi Pico (2020)
Clock Speed	16MHz	133MHz
Instruction Width	8bit	32bit
SRAM (programs)	64kB	264kB
Flash RAM (data)	3kB	2MB
GPIO	23	26
PWM	6	16
Board Cost	\$28	\$4

So which one do we use?

- What is the purpose?
 - **Control?** – do you want something to blink, move? Does it need to do it quickly?
 - **Data Gathering?** – do you want to measure something over a period of time? How much analysis do you want to do on the microcontroller?
- For **control** - the **Uno** is the easiest to make the fastest, has a simple hardware model, making it easy to interface
- For **data gathering** – the **Pico** has the most memory, has a language which supports a file structure, making it easy to gather data

Blink – the first program

- Blink is invariably the first program as it is a fast way to see your results
- Blink can also indicate speed of execution
- Blink Iterations:
 - On/Off – indicate an operating state
 - Blink Faster – indicate an operating state efficiently
 - Blink Faster at a specified rate – communications protocol such as I2C or SPI
 - Blink Faster with different colors – video protocol such as VGA video
- The faster the processor can process commands, the more the processor can appear to do at once, **multi-tasking**

Activity – Blink!

On each board/development environment do the following:

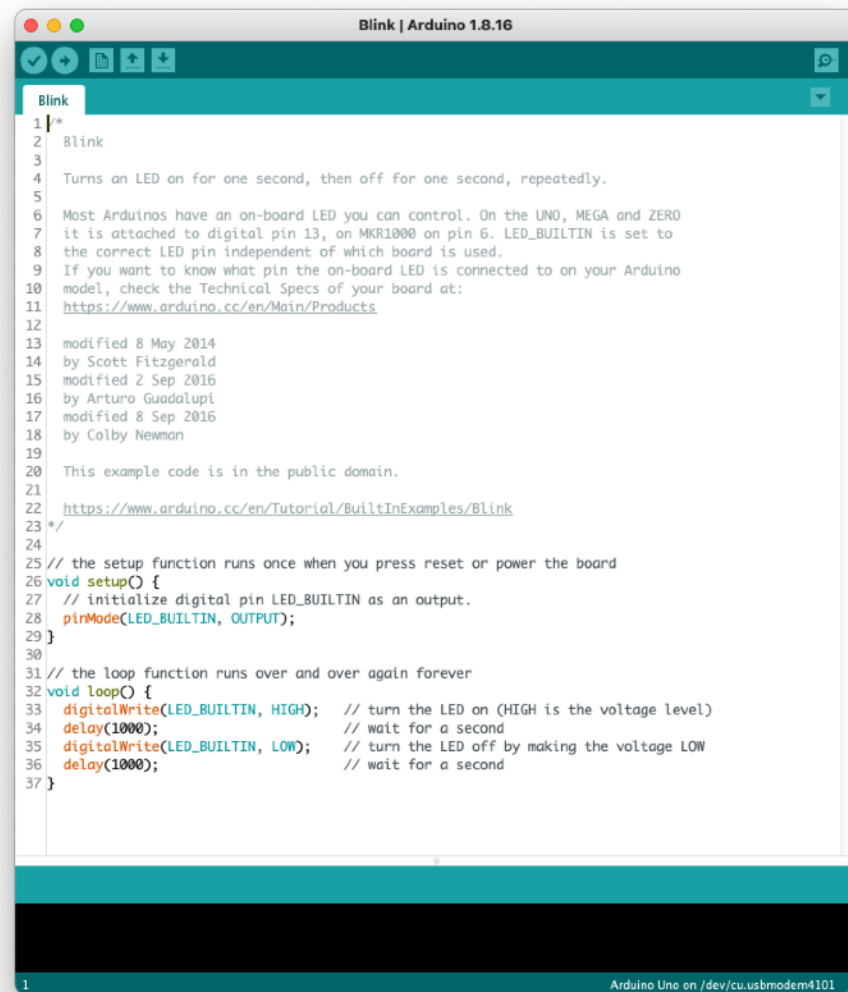
1. Blink an LED
2. Determine the fastest speed the board can blink

Board/Development Environments

1. Arduino Uno using *C++* with the Arduino 1.8 IDE
2. Raspberry Pi Pico using *MicroPython* and Thonny IDE

Integrated Development Environment (IDE)

Arduino 1.8

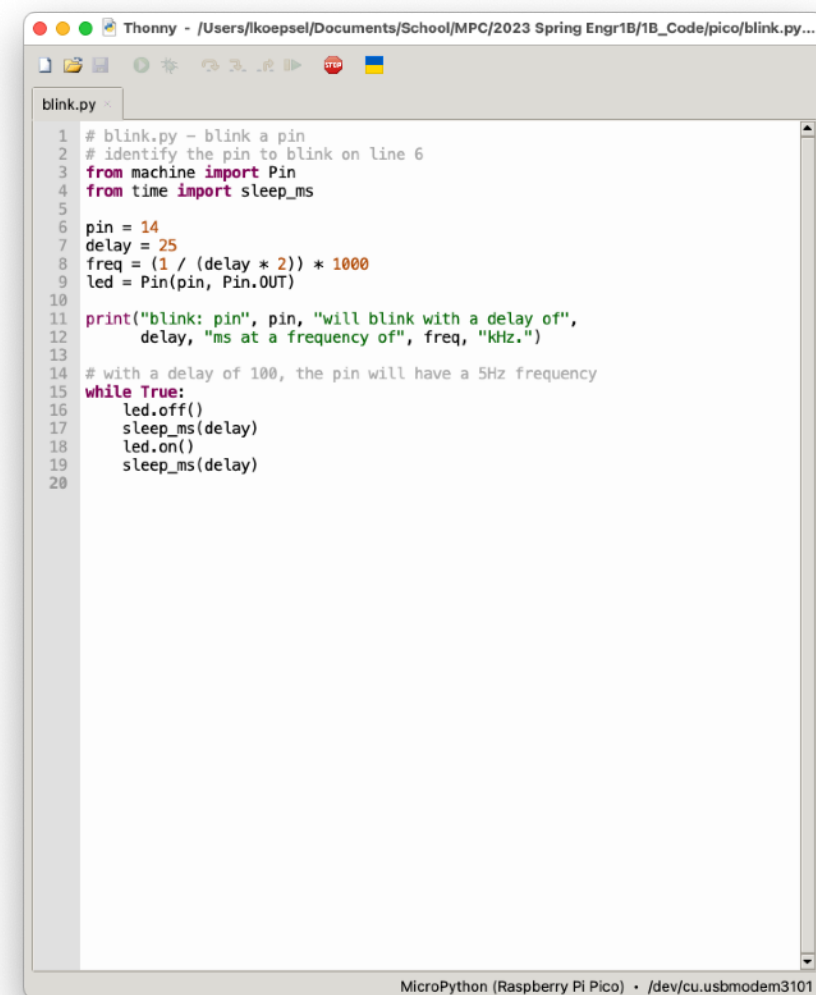


The screenshot shows the Arduino IDE window titled "Blink | Arduino 1.8.16". The code editor displays the standard Arduino Blink example. The code includes a multi-line comment explaining the function, a `void setup()` function that initializes the LED pin as an output, and a `void loop()` function that turns the LED on and off with a 1000ms delay. The status bar at the bottom indicates the board is "Arduino Uno" and the port is "dev/cu.usbmodem4101".

```
1 /*  
2  * Blink  
3  *  
4  * Turns an LED on for one second, then off for one second, repeatedly.  
5  *  
6  * Most Arduinos have an on-board LED you can control. On the UNO, MEGA and ZERO  
7  * it is attached to digital pin 13, on MKR1000 on pin 6. LED_BUILTIN is set to  
8  * the correct LED pin independent of which board is used.  
9  * If you want to know what pin the on-board LED is connected to on your Arduino  
10 * model, check the Technical Specs of your board at:  
11 * https://www.arduino.cc/en/Main/Products  
12 *  
13 * modified 8 May 2014  
14 * by Scott Fitzgerald  
15 * modified 2 Sep 2016  
16 * by Arturo Guadalupi  
17 * modified 8 Sep 2016  
18 * by Colby Newman  
19 *  
20 * This example code is in the public domain.  
21 *  
22 * https://www.arduino.cc/en/Tutorial/BuiltInExamples/Blink  
23 */  
24  
25 // the setup function runs once when you press reset or power the board  
26 void setup() {  
27   // initialize digital pin LED_BUILTIN as an output.  
28   pinMode(LED_BUILTIN, OUTPUT);  
29 }  
30  
31 // the loop function runs over and over again forever  
32 void loop() {  
33   digitalWrite(LED_BUILTIN, HIGH); // turn the LED on (HIGH is the voltage level)  
34   delay(1000); // wait for a second  
35   digitalWrite(LED_BUILTIN, LOW); // turn the LED off by making the voltage LOW  
36   delay(1000); // wait for a second  
37 }
```

Arduino Uno

Thonny

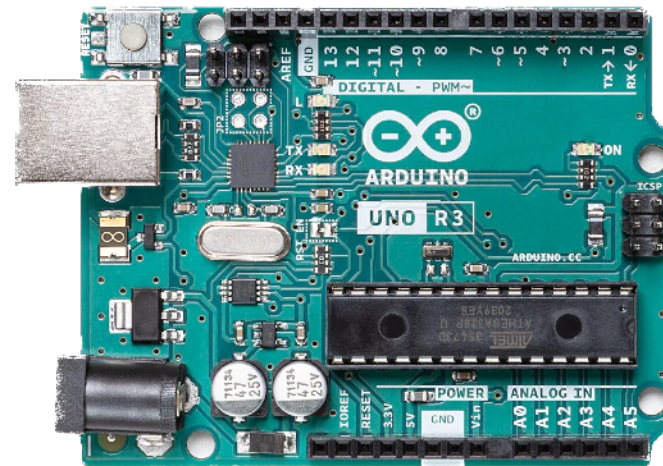


The screenshot shows the Thonny IDE window titled "Thonny - /Users/lkoepsel/Documents/School/MPC/2023 Spring Engr1B/1B_Code/pico/blink.py...". The code editor displays a Python script for blinking an LED on a Raspberry Pi Pico. The script imports `Pin` from `machine` and `sleep_ms` from `time`. It sets `pin = 14` and `delay = 25`. It calculates the frequency as `freq = (1 / (delay * 2)) * 1000`. It creates a `led = Pin(pin, Pin.OUT)` object. It prints a message with the pin number, delay, and frequency. It then enters a `while True:` loop that turns the LED off, sleeps for the delay, turns the LED on, and sleeps for the delay. The status bar at the bottom indicates the board is "MicroPython (Raspberry Pi Pico)" and the port is "dev/cu.usbmodem3101".

```
1 # blink.py - blink a pin  
2 # identify the pin to blink on line 6  
3 from machine import Pin  
4 from time import sleep_ms  
5  
6 pin = 14  
7 delay = 25  
8 freq = (1 / (delay * 2)) * 1000  
9 led = Pin(pin, Pin.OUT)  
10  
11 print("blink: pin", pin, "will blink with a delay of",  
12      delay, "ms at a frequency of", freq, "kHz.")  
13  
14 # with a delay of 100, the pin will have a 5Hz frequency  
15 while True:  
16     led.off()  
17     sleep_ms(delay)  
18     led.on()  
19     sleep_ms(delay)  
20
```

Raspberry Pi Pico

Parts

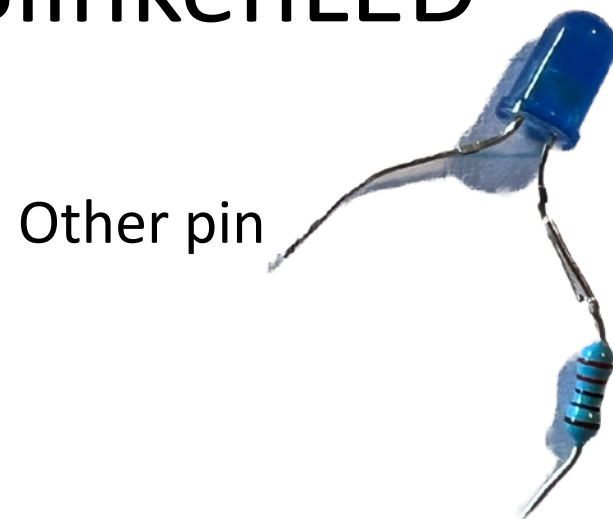


Arduino Uno



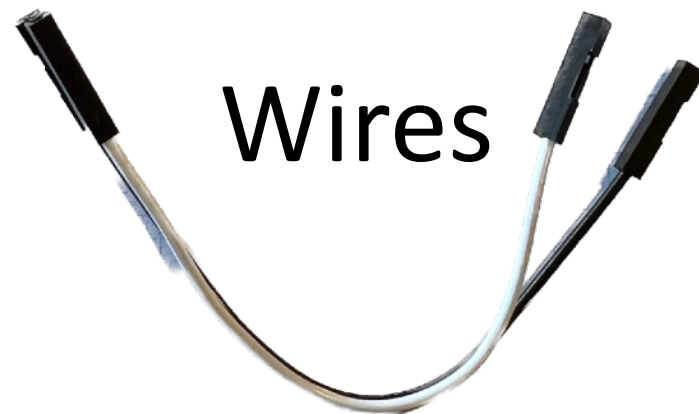
Uno USB cable

BlinkenLED



Other pin

Resistor pin



Wires



Pi Pico

microUSB cable



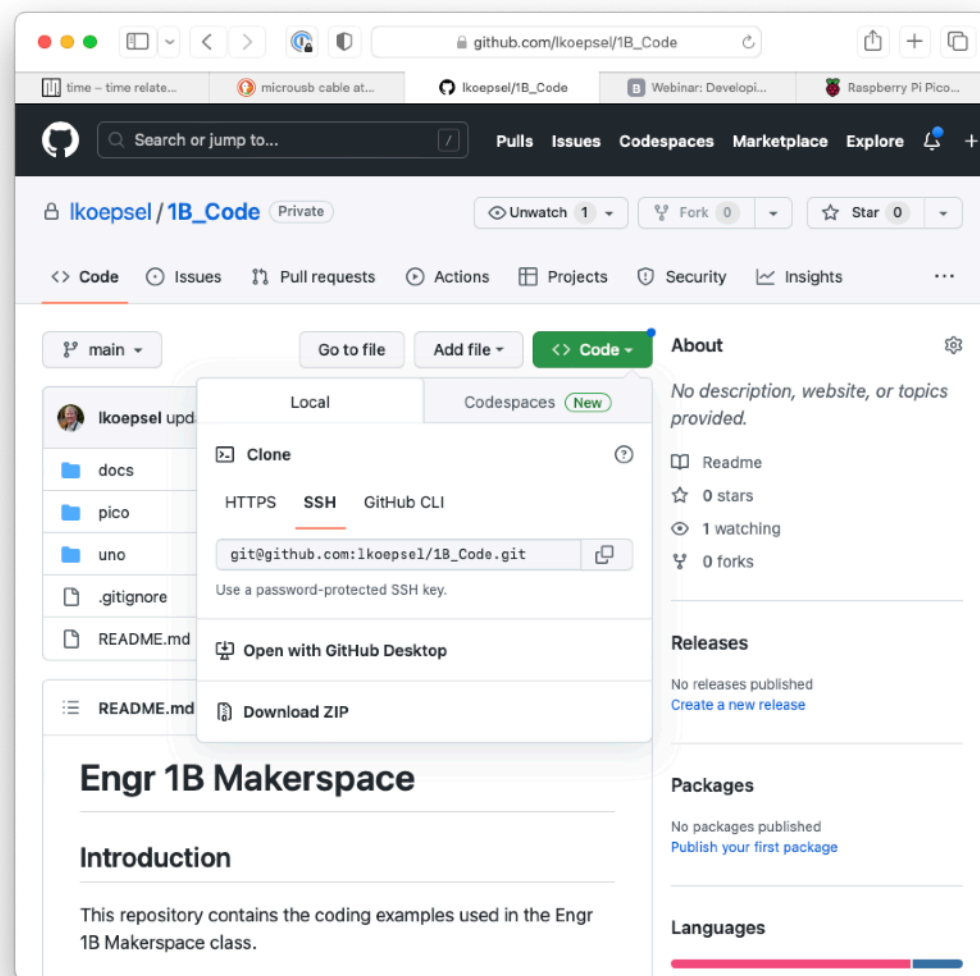
General Instructions for both boards

1. When making connections to the board pins, **DISCONNECT** the board from the computer
2. The **resistor-side** of the blinkenLED always goes to **ground**
3. Confirm connections **TWICE** before plugging board into USB, this reduces the chance of “frying” the board
4. Once plugged in, confirm the IDE can “see” the board before attempting to program
 - Arduino: Tools -> Port -> “board”
 - Thonny: Lower right-hand corner “board * port”



MicroPython (RP2040) • /dev/cu.usbmodem141301

Download from https://github.com/lkoepsel/1B_Code



- Programs are pre-written to accelerate your understanding
- Click on green button then “Download ZIP”
- Extract to your desktop
- Pico files are in pico folder
- Uno files are uno folder

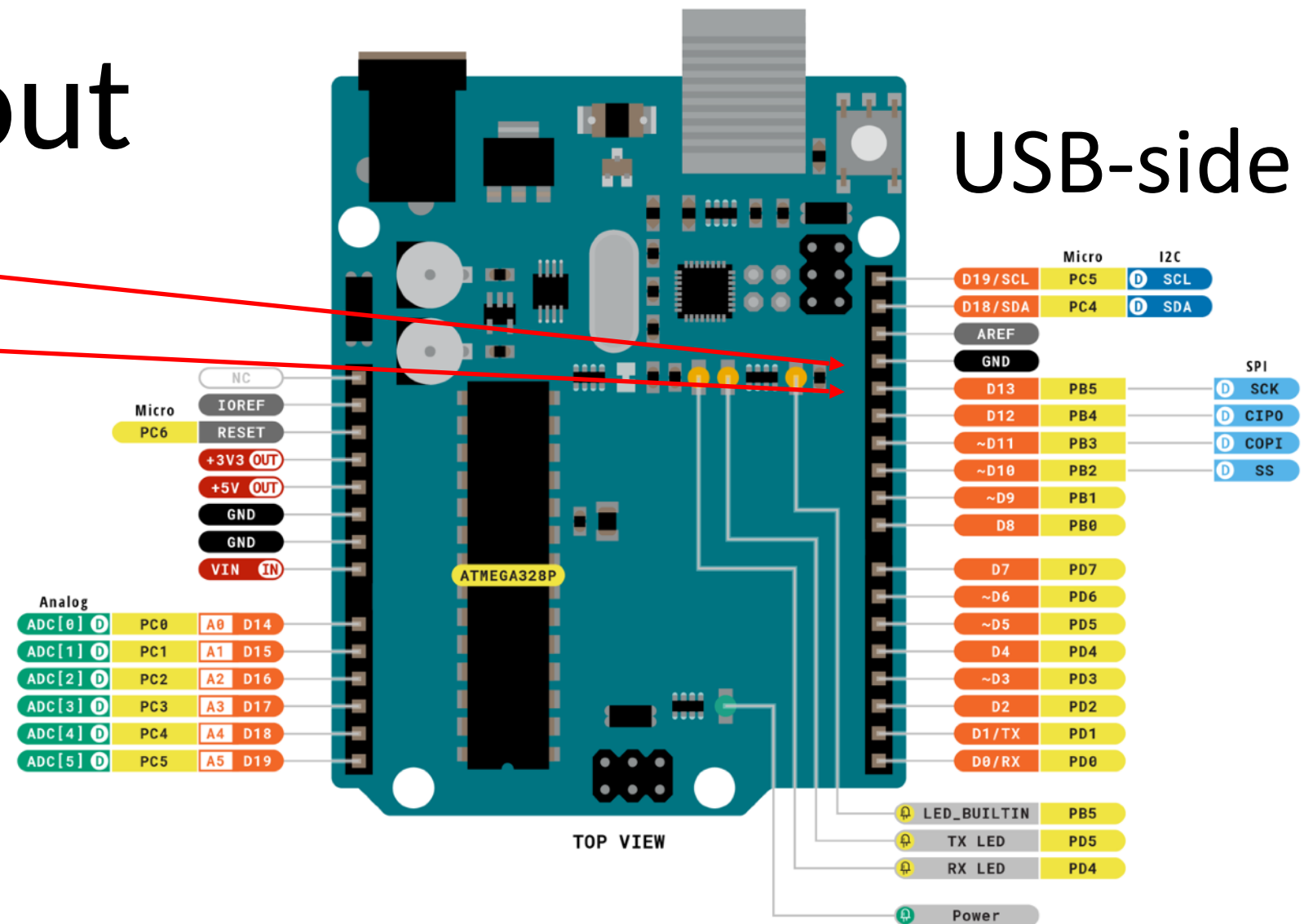
Arduino Uno - Blink

1. Use Arduino to edit your program and send it to the Uno
2. With the Uno **unconnected**:
 - A. Connect , resistor-side pin to GND (4th from top, USB-side)
 - B. Connect other pin to D13 (5th from top, USB-side)
3. Plugin Uno
4. Confirm connection in Arduino
5. Open blink.ino from uno folder and upload to Uno board
6. Is light blinking correctly?
7. Now alter the program to run as fast as possible, still blinking
8. At what freq does it blink? (*Ask for digital multimeter*)

Uno R3 Pinout

GND
D13

USB-side



Legend:	Digital	I2C
Power	Analog	SPI
Ground	Main Part	Analog

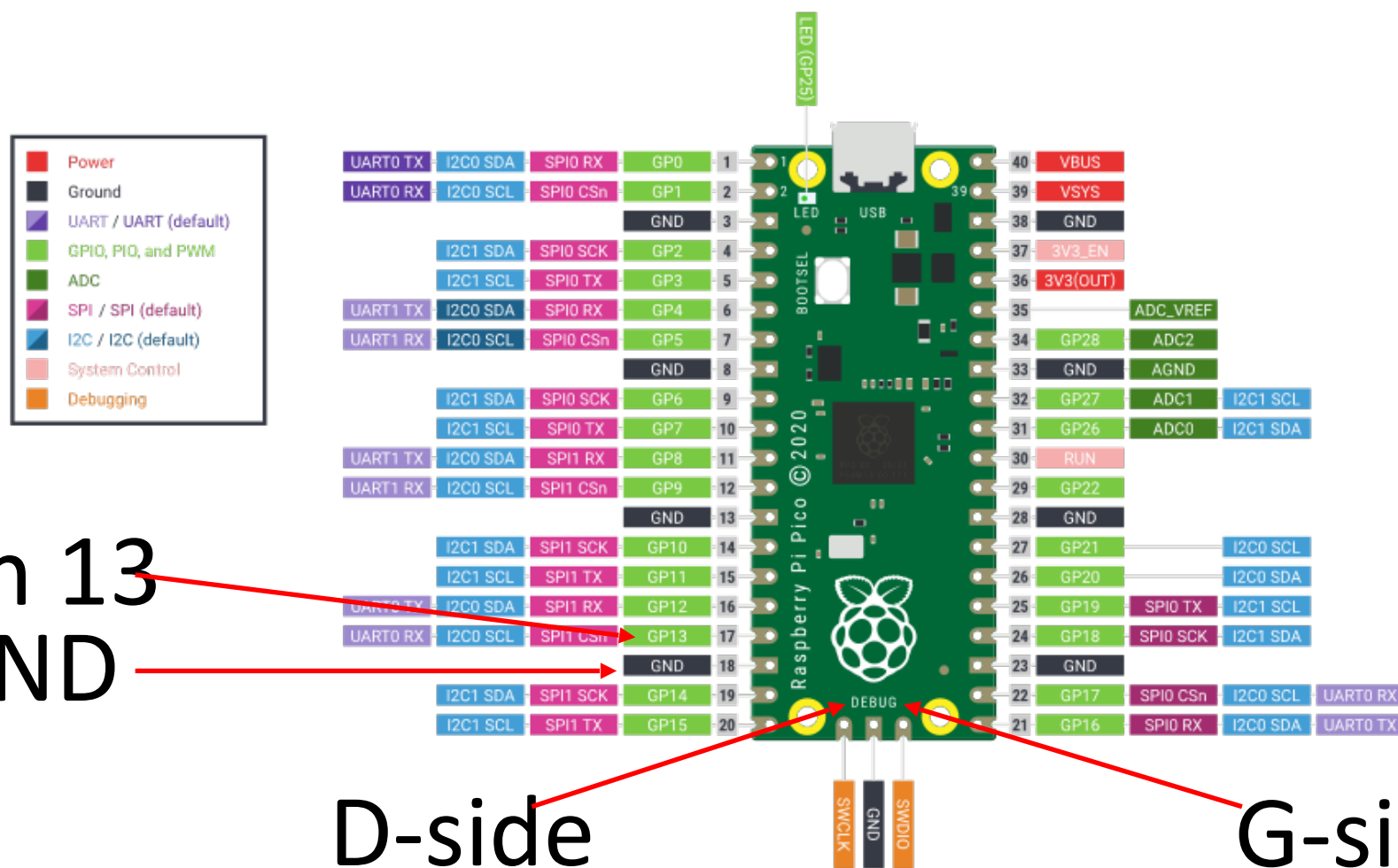


ARDUINO UNO REV3
SKU code: A000066
Pinout
Last update: 6 Oct, 2022

Raspberry Pi Pico - Blink

1. Use Thonny to edit your program and send it to the Pico
2. With the Pico **unconnected**:
 - A. Connect , resistor-side pin to GND (3rd from bottom, D-side)
 - B. Connect other pin to GP13 (4th from bottom, D-side)
3. Plugin Pico
4. Confirm connection in Thonny
5. Open blink.py from pico folder and Save As to pico board
6. Is light blinking correctly?
7. Now alter the program to run as fast as possible, still blinking
8. At what freq does it blink? (*Ask for digital multimeter*)

Raspberry Pi Pico Pinout



Pin 13
GND

D-side

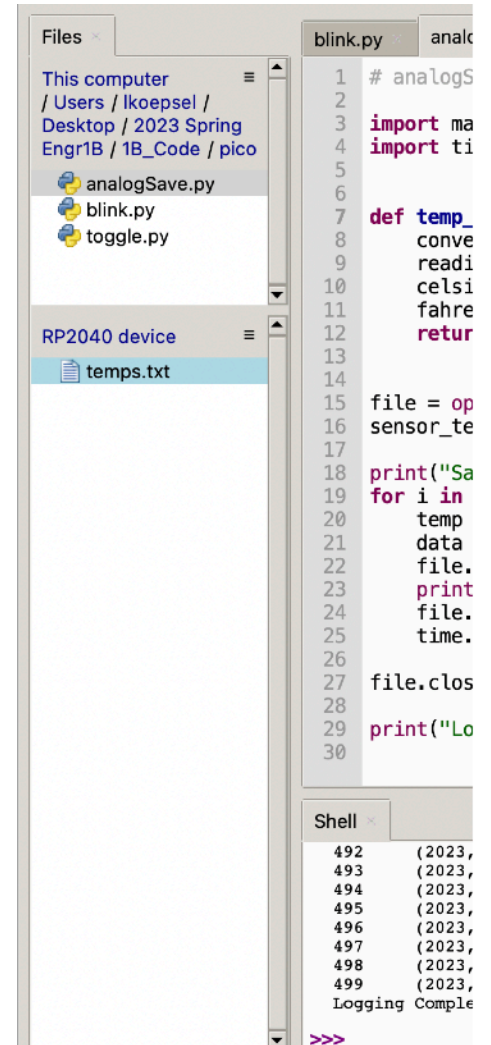
G-side

Arduino Uno - Data Logger

1. Use Arduino to edit your program and send it to the Uno
2. For the data logging, we'll leave the pin A0 unconnected and measure air temperature
3. Plugin Uno
4. Confirm connection in Arduino
5. Open AnalogSave.ino from uno folder and upload to Uno board
6. Open eepromRead from the uno folder and upload to Uno Board
7. Is the data read the same as what you wrote?
8. How many samples are you able to capture?

Raspberry Pi Pico - Data Logger

1. Use Thonny to edit your program and send it to the Pico
2. For the data logging, we'll use the processor core temperature
3. Plugin Pico
4. Confirm connection in Thonny
5. Open AnalogRead.py from pico folder and upload to Pico board
6. Open *temps.txt* from the pico
 - A. Use View -> Files
 - B. Look in the RP2040 window for temps.txt, double click to open
7. Is the data read the same as what you wrote?
8. How many samples are you able to capture?
9. What else do you notice about the data?



The screenshot shows the Thonny IDE interface. On the left, the 'Files' pane displays the project structure: 'This computer' (containing 'Users / lkoepsel / Desktop / 2023 Spring / Engr1B / 1B_Code / pico' with files 'analogSave.py', 'blink.py', and 'toggle.py') and 'RP2040 device' (containing 'temps.txt'). The main editor window shows a Python script named 'analogS.py' with the following code:

```
1 # analogS
2
3 import ma
4 import ti
5
6
7 def temp_
8     conve
9     readi
10    celsi
11    fahre
12    retur
13
14
15 file = op
16 sensor_te
17
18 print("Sa
19 for i in
20     temp
21     data
22     file.
23     print
24     file.
25     time.
26
27 file.clos
28
29 print("Lo
30
```

At the bottom, the 'Shell' pane shows the output of the program, displaying a list of tuples representing temperature readings over time, such as (2023, 492), (2023, 493), (2023, 494), (2023, 495), (2023, 496), (2023, 497), (2023, 498), (2023, 499), and 'Logging Comple'.

Appendix - Extra Slides

Simplifying the tradeoffs

- **Speed of development:**

- How hard is the processor, language and framework to learn?
- How complex is the development environment?
- How quickly can I have a minimum viable project?

- **Ability of execution:**

- What is the speed of the controller, compared to the task?
- How much data can the controller handle, compared to the task?
- How can the internal hardware help?

Why is Speed of Development important?

- You want to accomplish something!
- You want to be efficient.
- You want to enjoy what you are doing.
- You want to search less and develop more.
- You don't want to have to jump through hoops.

Speed of Development Components

- **Language – Python, C Language, Assembly Language**
 - Framework – Libraries, Headers, Functions, Library Management
 - Tool Chain – Compilers, Linkers, Editors
 - GUI vs. CLI – Ease of view vs. Ease of Automation
-
- Ultimately, all decisions come down the **language followed by the framework**

Speed of Execution

- Board and Language Compatibility
 - **Raspberry Pi** – all languages, however, OS has significant overhead
 - **Raspberry Pi Pico** – all languages
 - **Arduino Uno** – C/C++, Assembly Language, best supported processor
 - ESP32 – all languages
 - STM32 – all languages

Language vs. Speed

