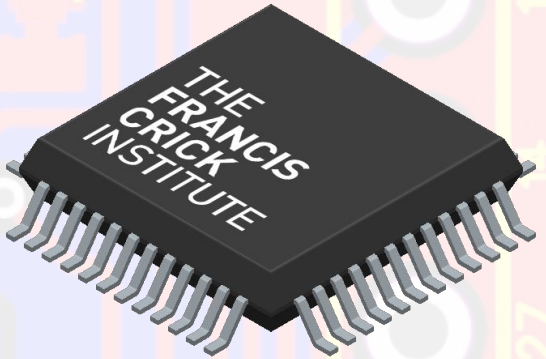
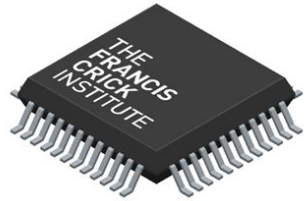
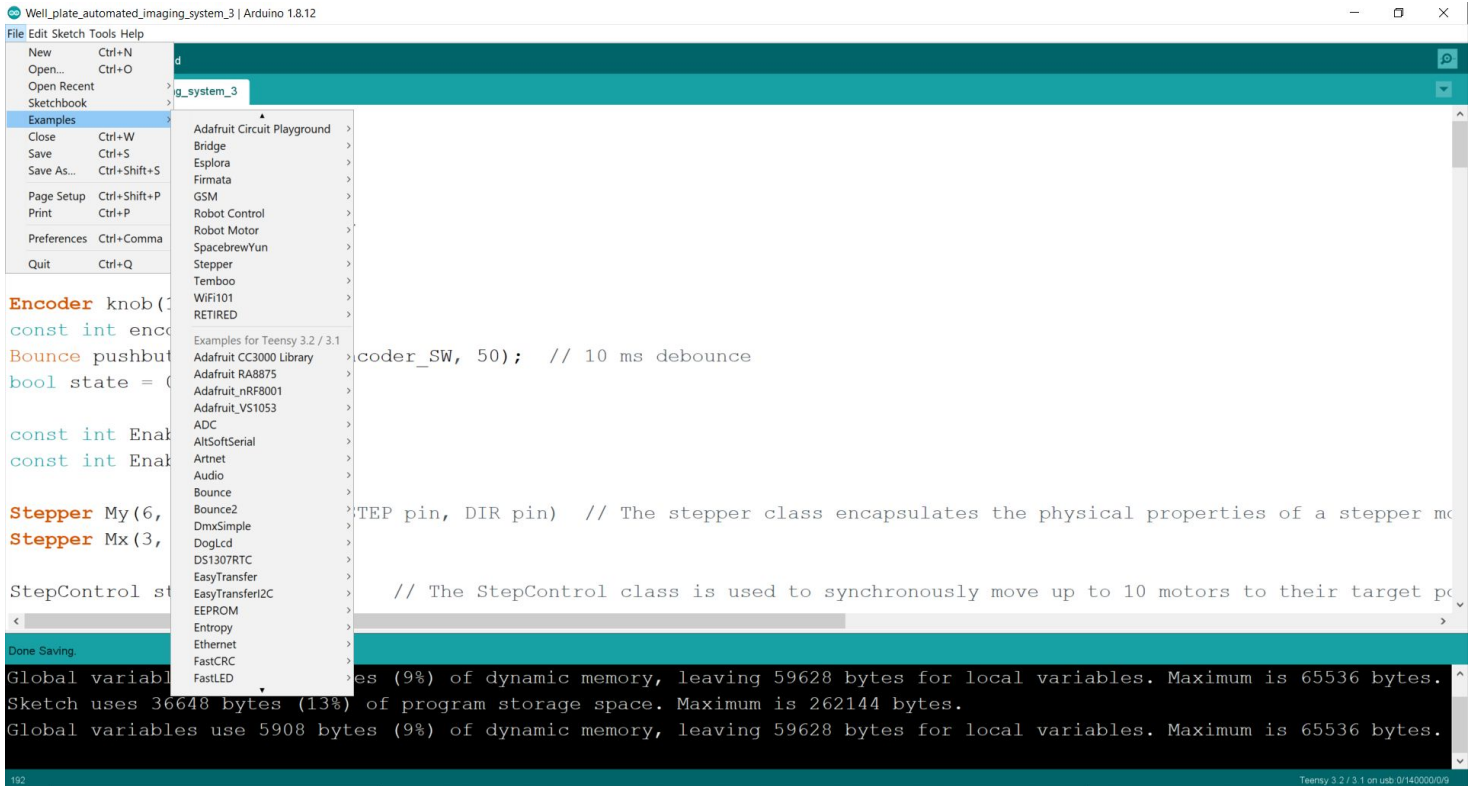


Microcontrollers (2022)

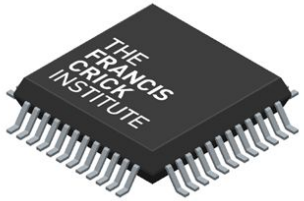
Session 2

Where to find examples from installed libraries





Arduino Sketch typical structure



#include libraries

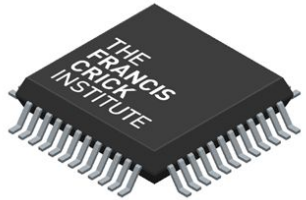
Declare Objects

Global variables

setup() function (It is executed one time)

loop() function (It loops indefinitely)

Main data types in C/C++ (Arduino language)

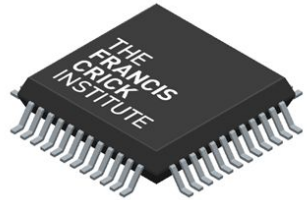


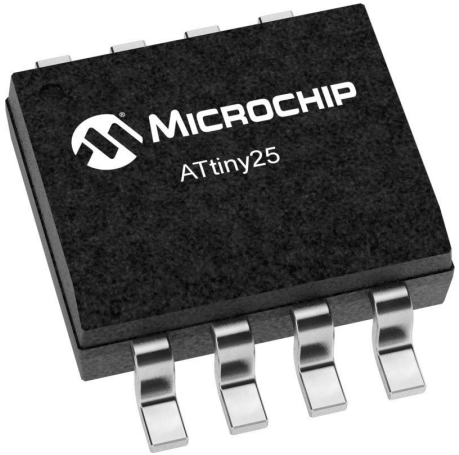
In the Arduino programing language you **always** have to **declare** the **data type** of your **variable/constant**.

- **boolean** (8 bit) - simple **logical true/false**. *In other languages it's 1 bit.*
- **byte** (8 bit) - unsigned number from **0-255**
- **int** (16 bit) - **signed number** from -32768 to 32767.
- **unsigned int unsigned number** from 0-65535 (16 bit).
- **long** (32 bit) - **signed number from** -2,147,483,648 to 2,147,483,647
- **unsigned long** (32 bit) - **unsigned number** from 0-4,294,967,295. The most common usage of this is to store the result of the `millis()`.
- **float** (32 bit) - **signed number** from -3.4028235E38 to 3.4028235E38. **For real values.**

https://learn.sparkfun.com/tutorials/data-types-in-arduino/defining-data-types

Memory is a limiting factor is Microcontrollers!



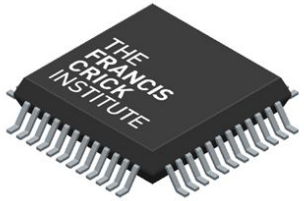


The ATtiny25 from Atmel has 128 Bytes of SRAM!

That only gives us space for:

128 **byte** variables
64 **int** variables
32 **float** variables

How we declare a variable or constant in C/C++ ?




```
int pin = 13;
```

integer

Name we assign to the variable

Value we assign to the variable

or…

```
int pin;
```

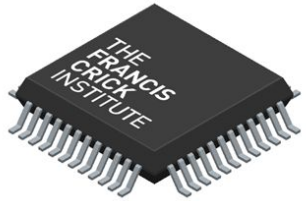
```
pin = 13;
```

```
const float R = 3.14;
```

It cannot change

They are case sensitive, R ≠ r !!

Conditional structures in C/C++

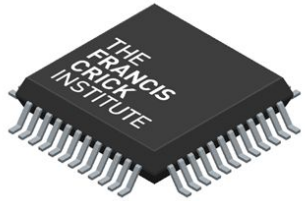



```
if(someVariable > 50)
{
  // do something here
}

else
{
  // do else something here
}
```

x == y (x is equal to y)
x != y (x is not equal to y)
x < y (x is less than y)
x > y (x is greater than y)
x <= y (x is less than or equal to y)
x >= y (x is greater than or equal to y)

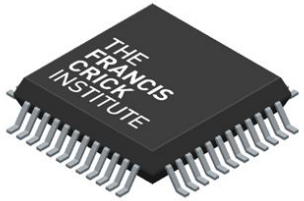
Iterative structures in C/C++




```
byte i = 0;
while (i < 255)
{
    analogWrite(pin5, i);
    i++;
}
for (byte i = 0; i <= 255; i++)
{
    analogWrite(pin5, i);
}
```

i++ ⇔ i=i+1

Functions in C/C++



The function returns an integer

```
int multiply (int x, int y)
{
   int result;
   result = x * y;
   return result;
}
```

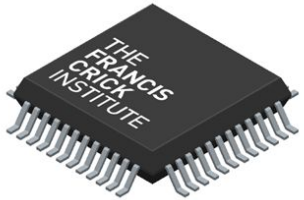
The function receives two integers x and y

Curly brackets required

```
k = mutliply(x, y);
```

Function call

Functions in C/C++

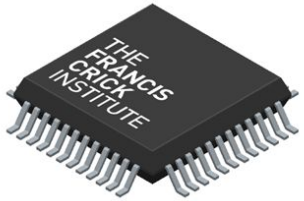


“void” if the function does not return anything

```
void manage_pins ()
{
   digitalWrite(3,HIGH);
}
```

In this case the function does not receive anything (Nothing between parenthesis)

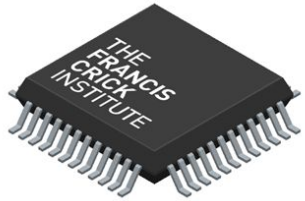
Global vs local variables




```
int result;
int multiply (int x, int y)
{
   result = x * y;
   return result;
}
```

It will work,
but the variable can be changed
in other places in the program

Global vs local variables

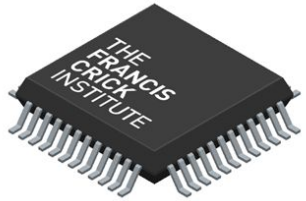



```
void setup()
{
   int var;
}
var = 12;
```

comiling_error:8: error: 'var' does not name a type

var has been declared as a **local variable** inside the setup() function

Time, delay() vs millis()/micros()



delay()

Pauses the program for the amount of time (in milliseconds) specified as parameter. **It won't do anything else for the duration of the delay.**

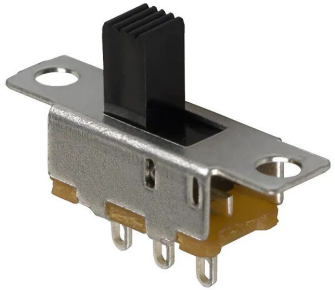
delay(ms) / delayMicroseconds(us)

data types: unsigned long / unsigned int

millis()/micros()

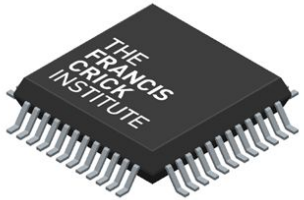
Returns the number of milliseconds passed since the Arduino board began running the current program. **We can use it to do other tasks and keep track of time.**

Contact bounce or “Chatter”

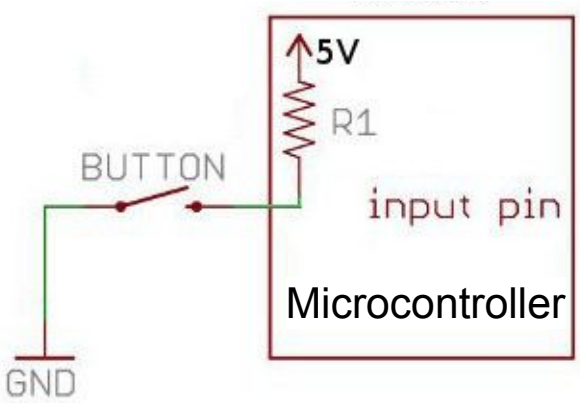


https://www.pjrc.com/teensy/td_libs_Bounce.html

How to avoid “chatter” or contact bounce: debouncing



Internal pull-up resistor



Some examples:

Only check when the state change and hope for the best

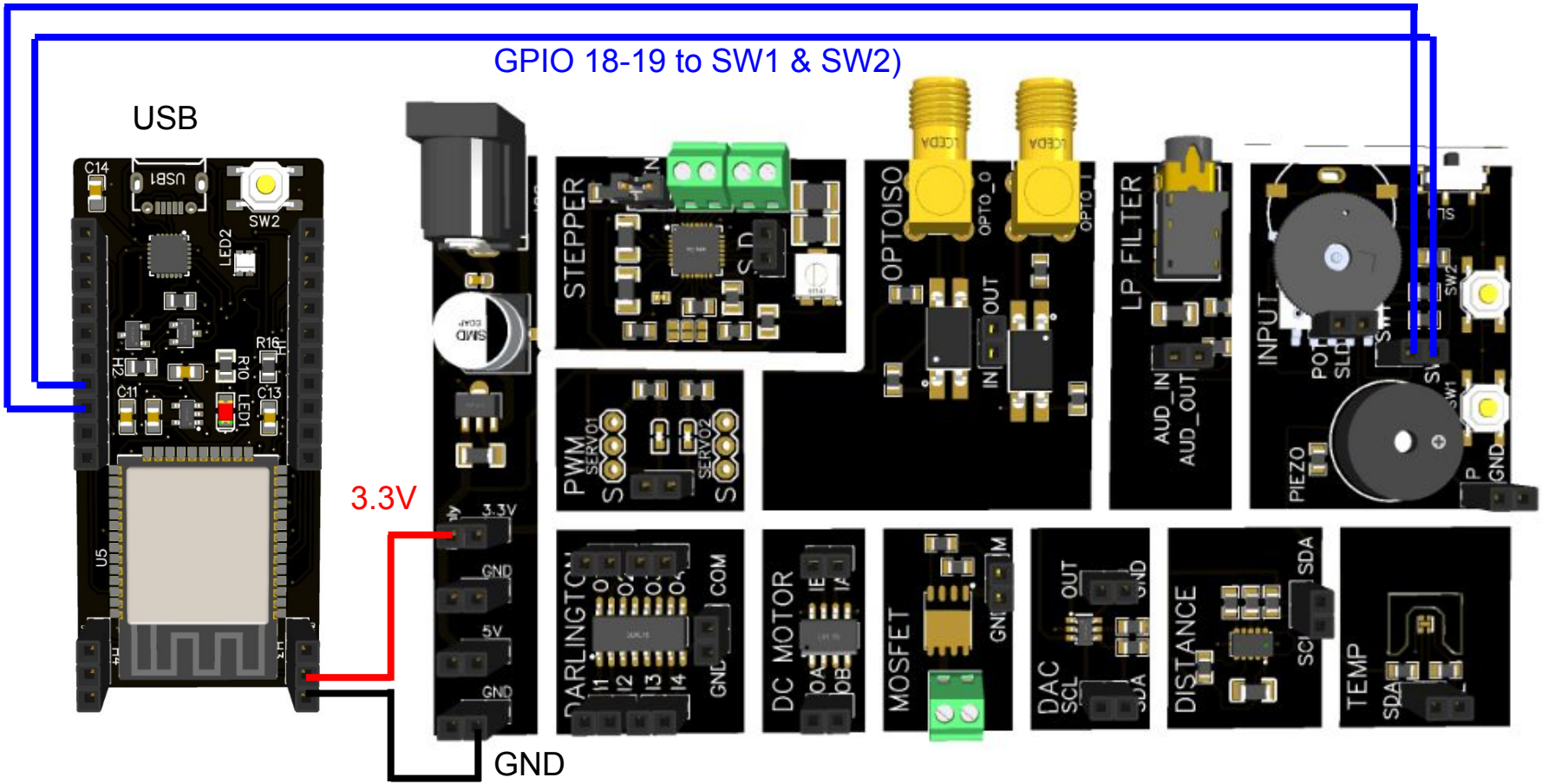
Using a delay(ms) to avoid it

Use a millis() strategy

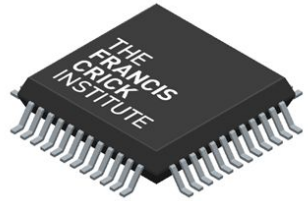
Using the bounce 2 library

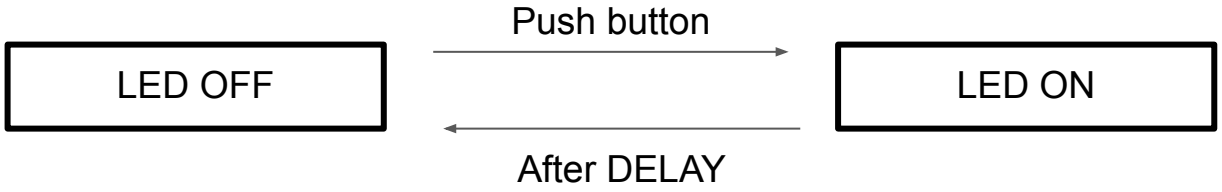
https://github.com/thomasfredericks/Bounce2

Examples 201 - 203

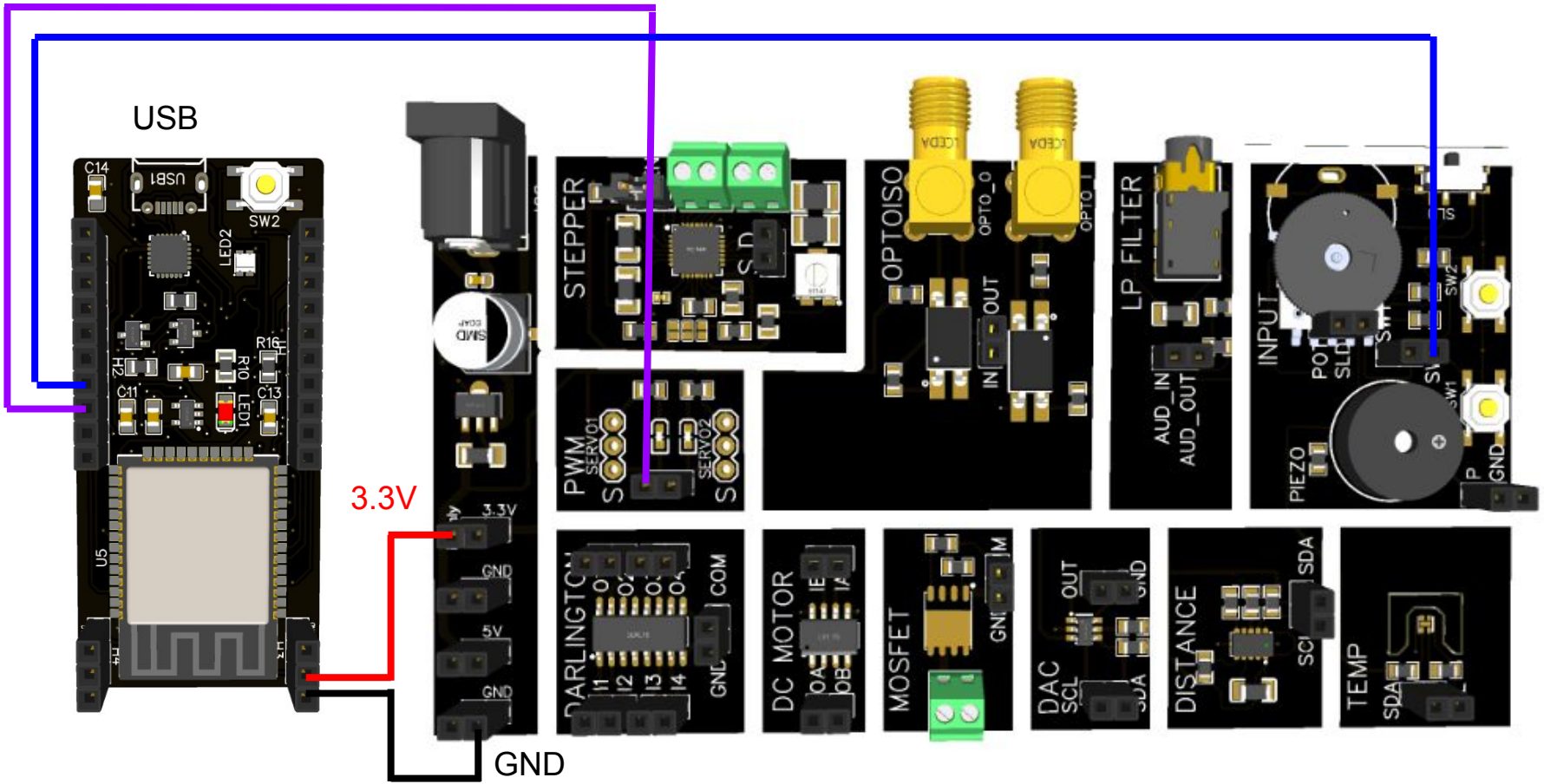


Finite state machines (FSM)

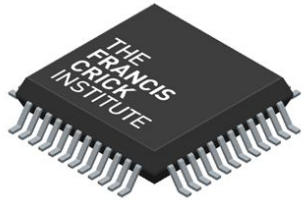




Examples 204

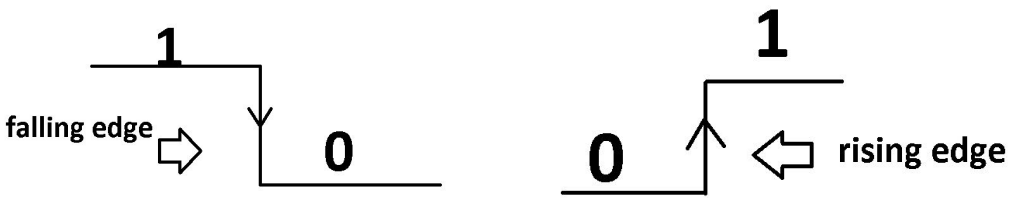


Interrupt Service Routines (ISR) or just Interrupts



We can see as interrupts as pins we configure to be externally triggered in a specific condition:

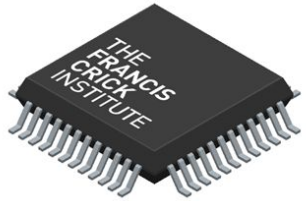
- **LOW** to trigger the interrupt whenever the pin is low,
- **CHANGE** to trigger the interrupt whenever the pin changes value
- **RISING** to trigger when the pin goes from low to high,
- **FALLING** for when the pin goes from high to low.



When an interrupt occurs, the microcontroller runs the interrupt service routine.

https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/

Interrupts syntax



Inside the **setup()** function we specify the essentials of the Interrupt using the **attachInterrupt()** function:

```
void setup()
{
  attachInterrupt(digitalPinToInterrupt(interruptPin), function,  RISING);
}
```

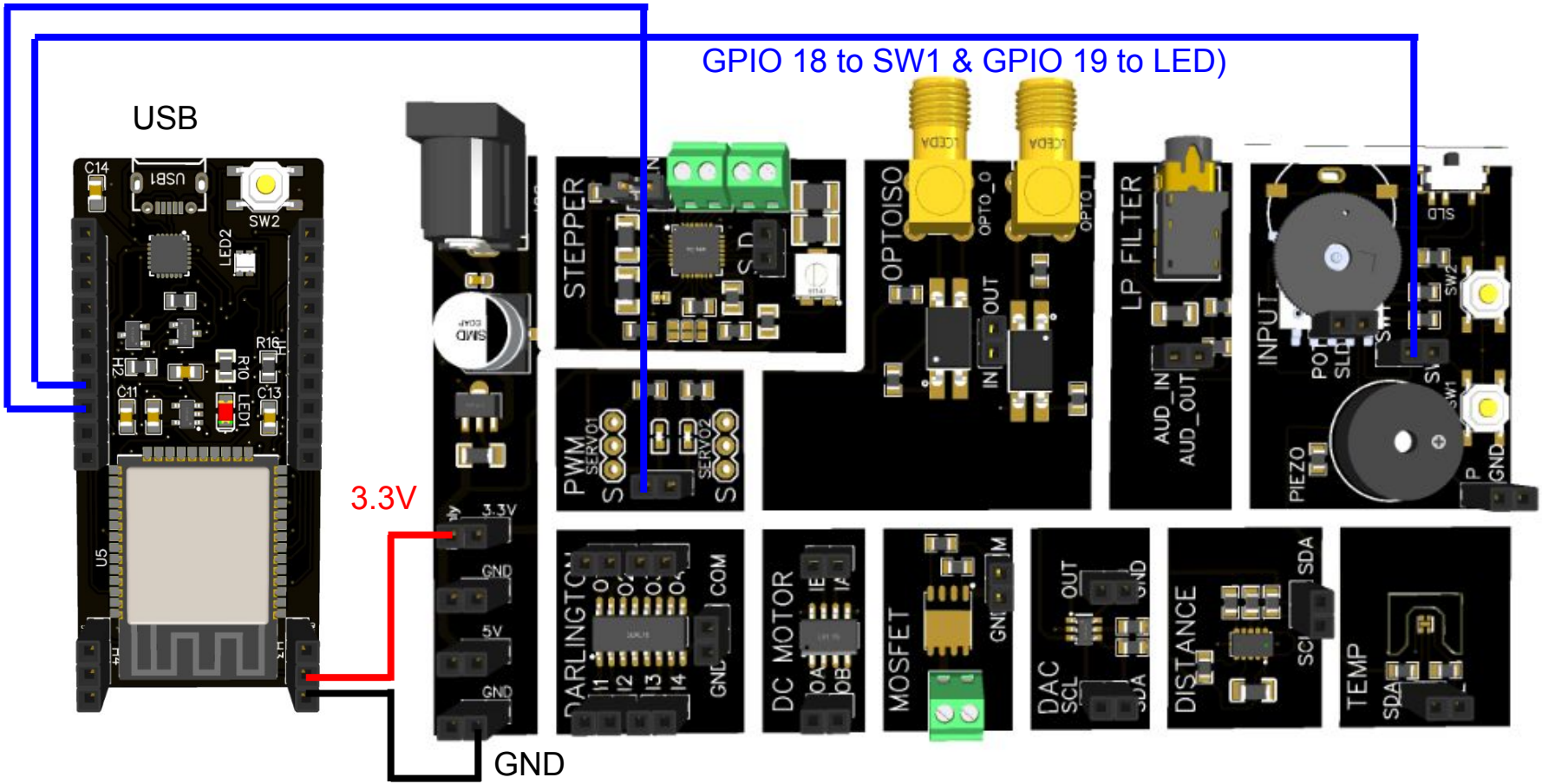
The condition

The pin that causes the interrupt

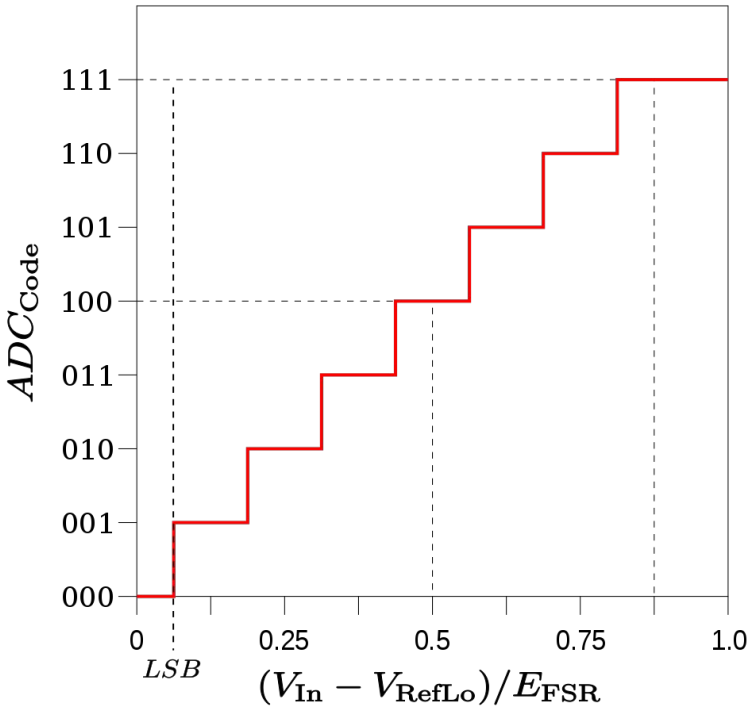
The function that is called after the interrupt is triggered

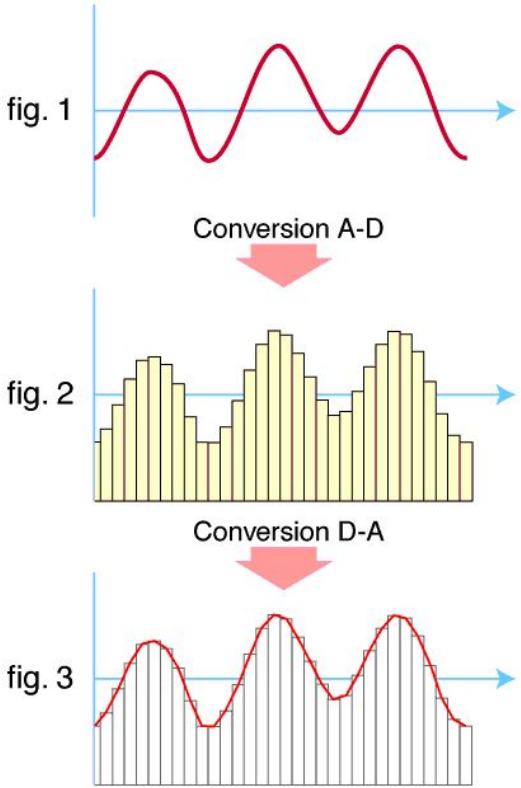
https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/

Examples 204-205

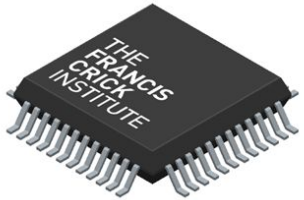


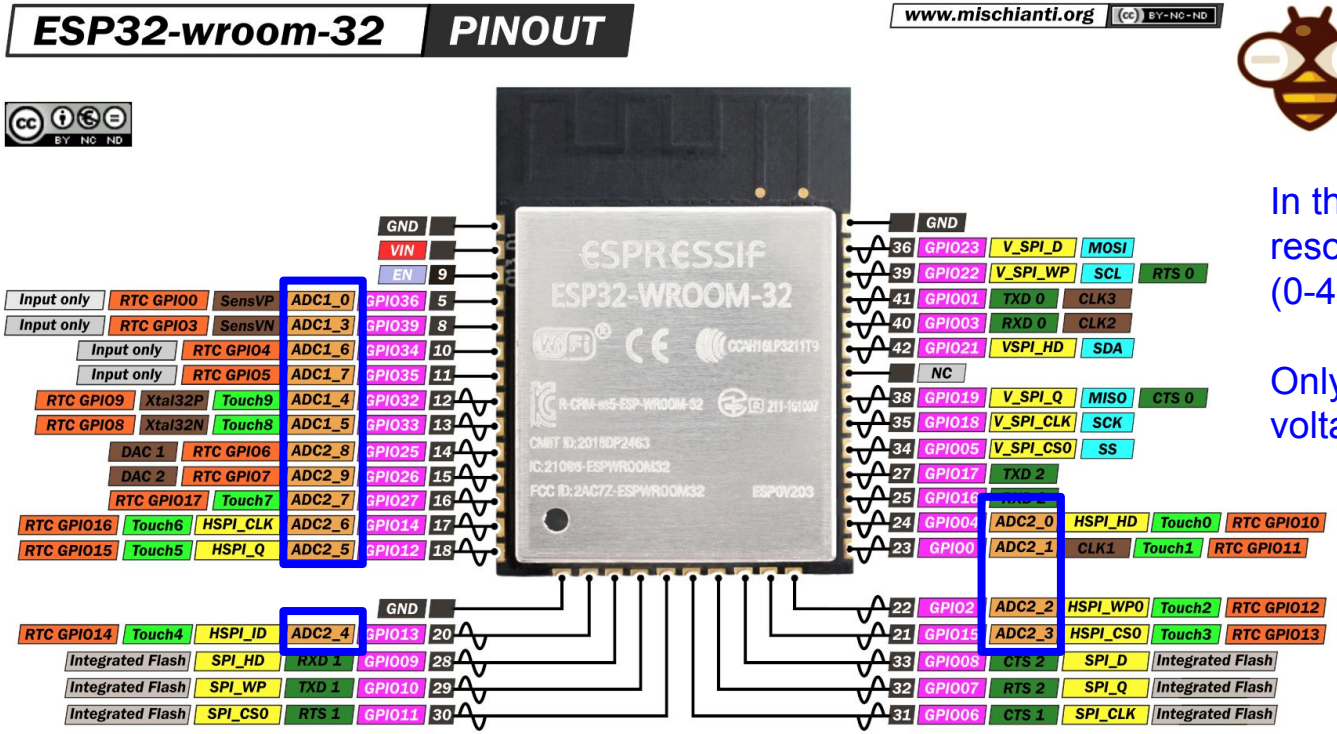
How Microcontrollers can deal with analogue signals ? Analogue to digital converters (ADC)





Usually Microcontrollers have a number of “built in” ADC

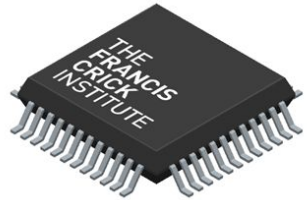


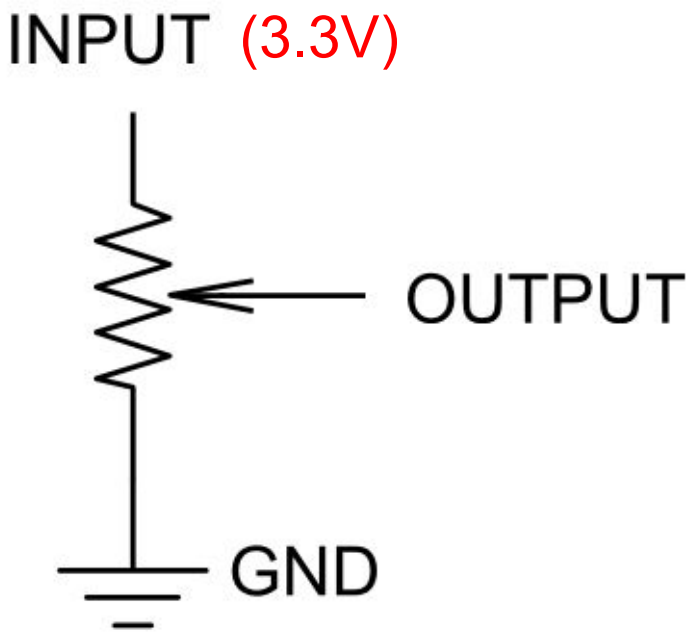


In the ESP32 the ADC resolution is 12 bits (0-4096 values)

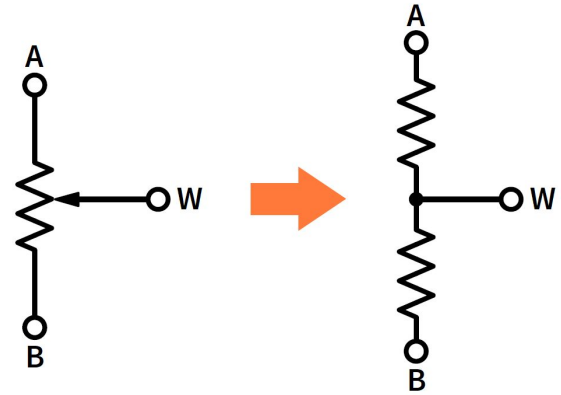
Only can take positive voltage values

A quick exercise to check how we can read an analogue signal with a variable resistor





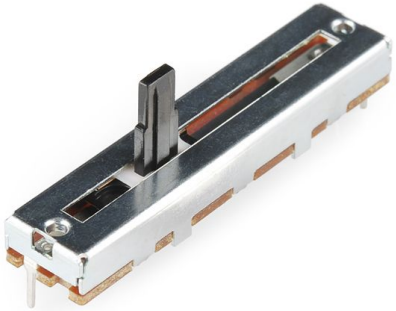
To GPIO pin with ADC

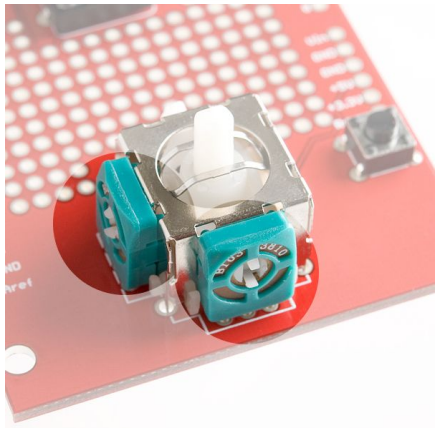


$$V_{out} = V_{in} \cdot \frac{R_2}{R_1 + R_2}$$

https://www.circuitbread.com/tutorials/basics-of-potentiometers

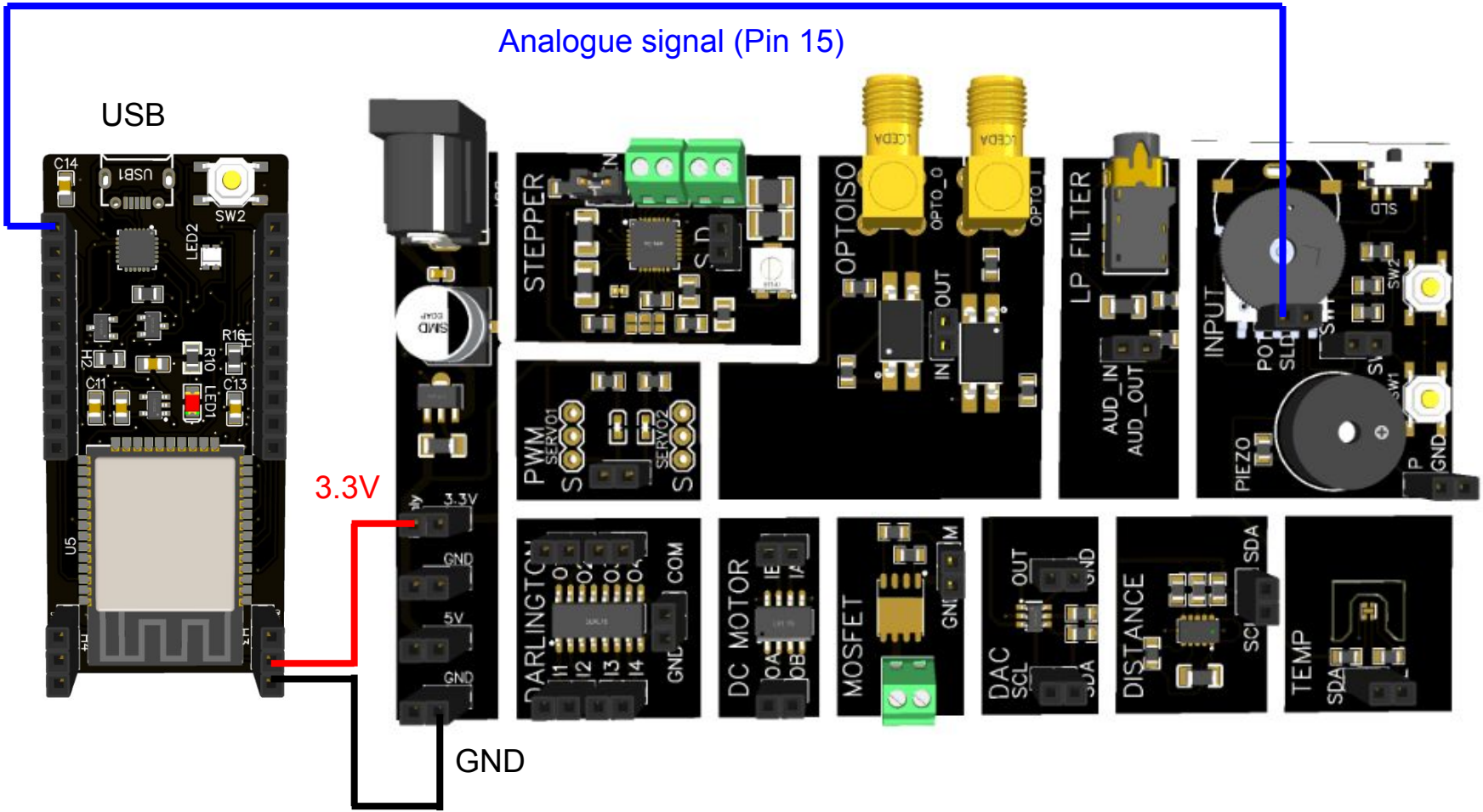
We find potentiometers everywhere



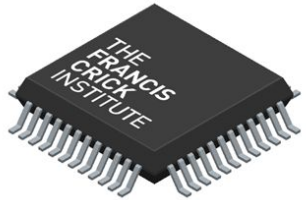


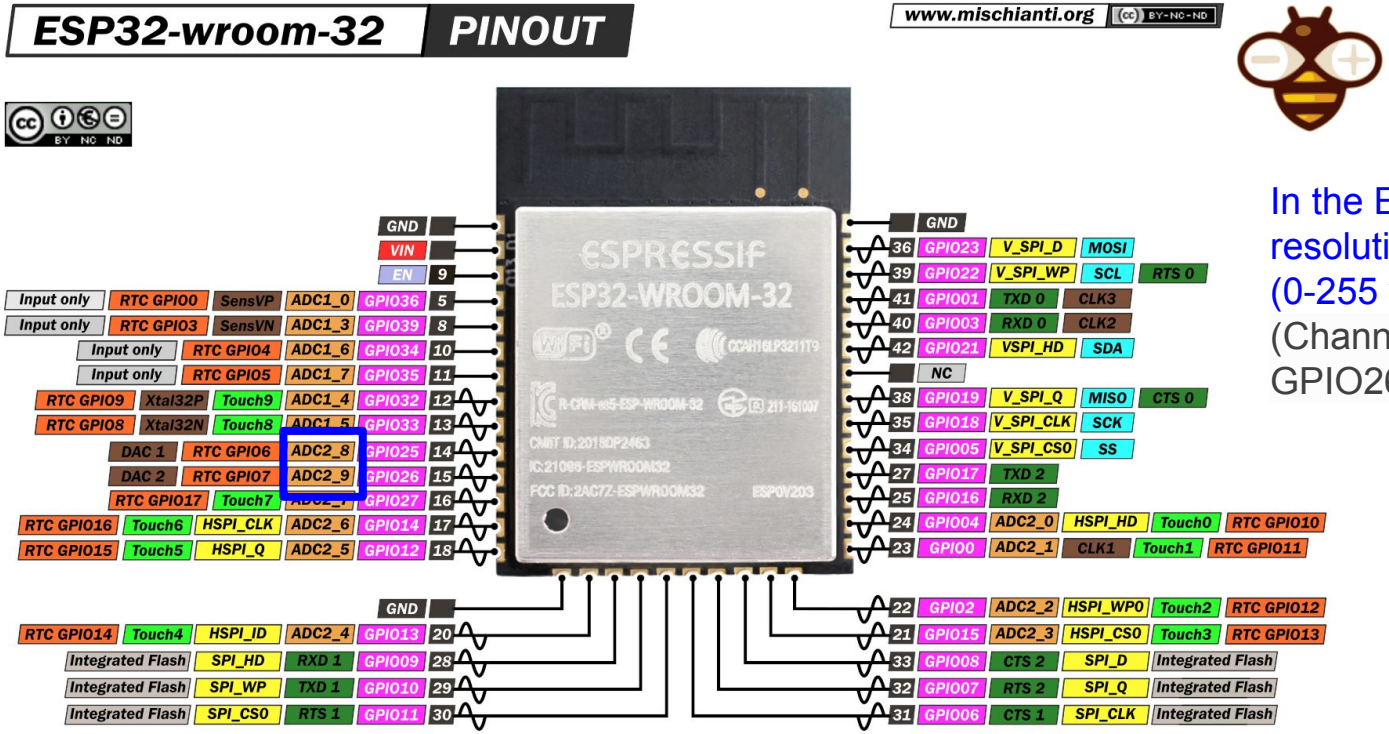
https://learn.sparkfun.com/tutorials/rotary-switch-potentiometer-hookup-guide/background-and-theory

Analogue to digital converters (ADC) AnalogReadSerial.ino (Example 206)



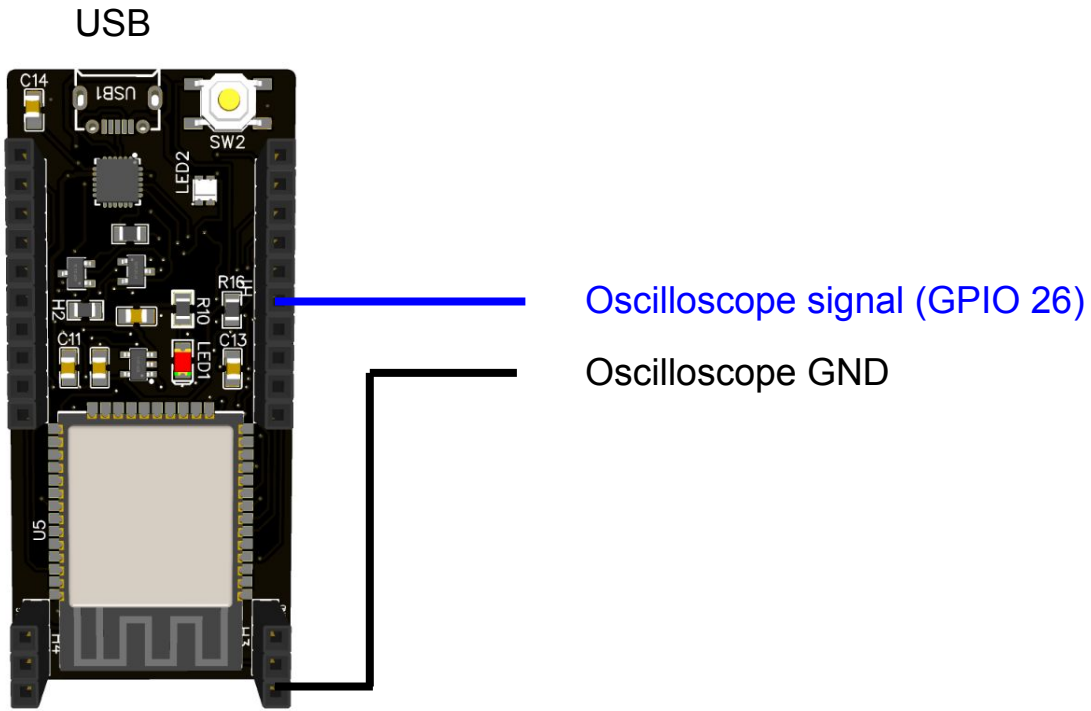
32-Bit microcontrollers such as ESP32 have built-in Digital to Analogue Converters (DAC)



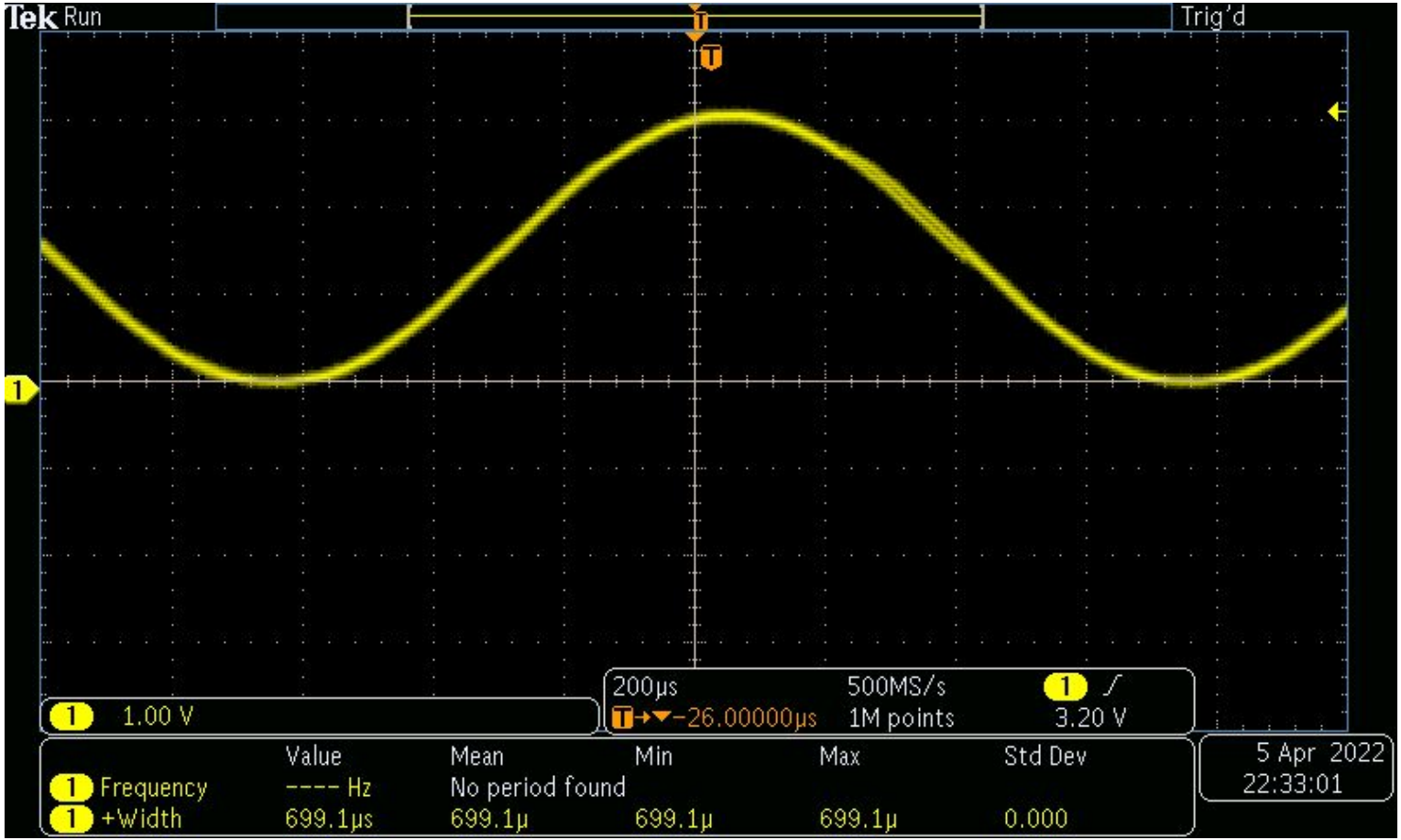


In the ESP32 the DAC resolution is 8 bits (0-255 values) GPIO25 (Channel 1) and GPIO26 (Channel 2)

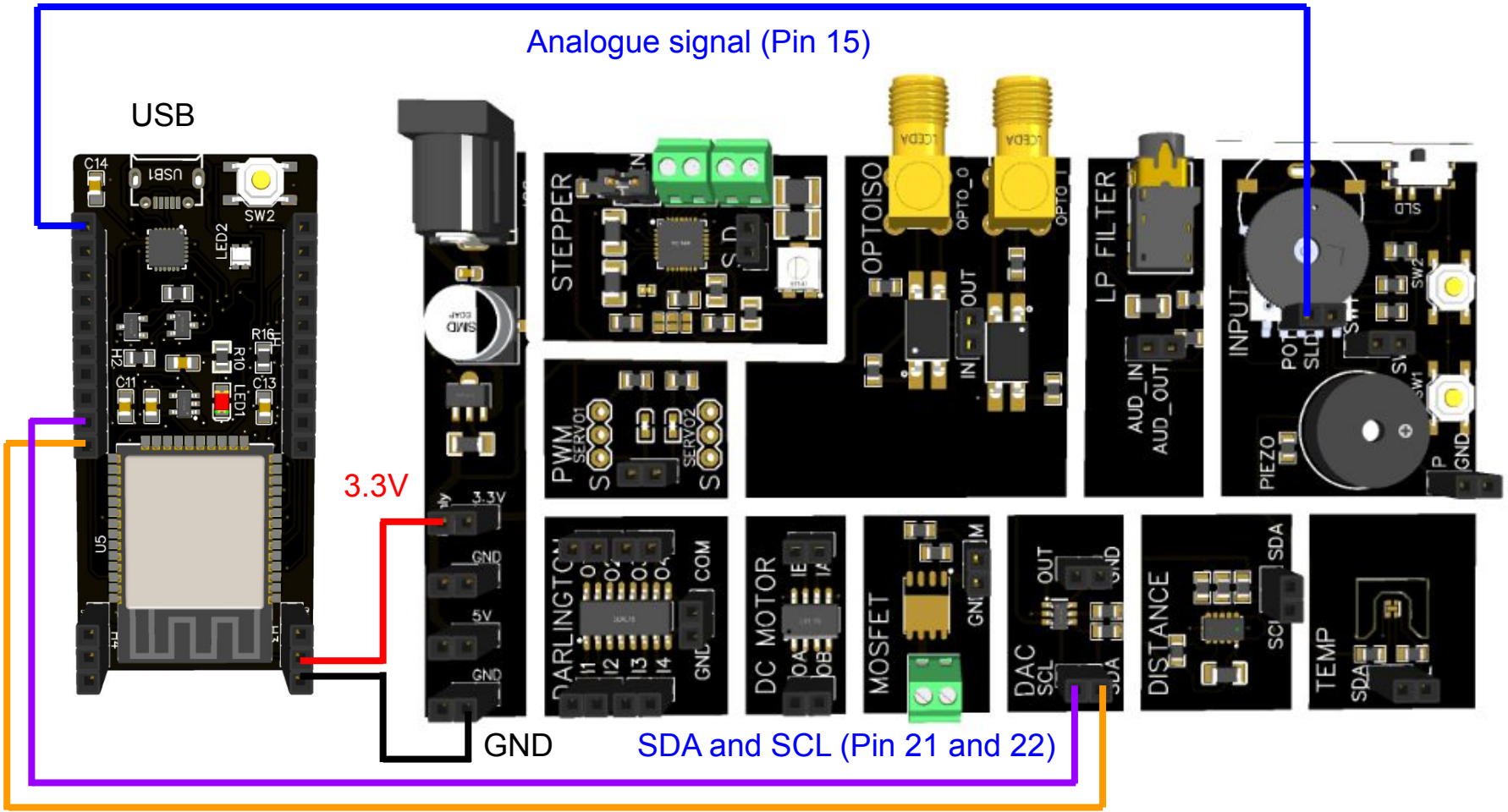
Digital to analogue converters (DAC) Basic_DAC_built_in.ino (Example 207)



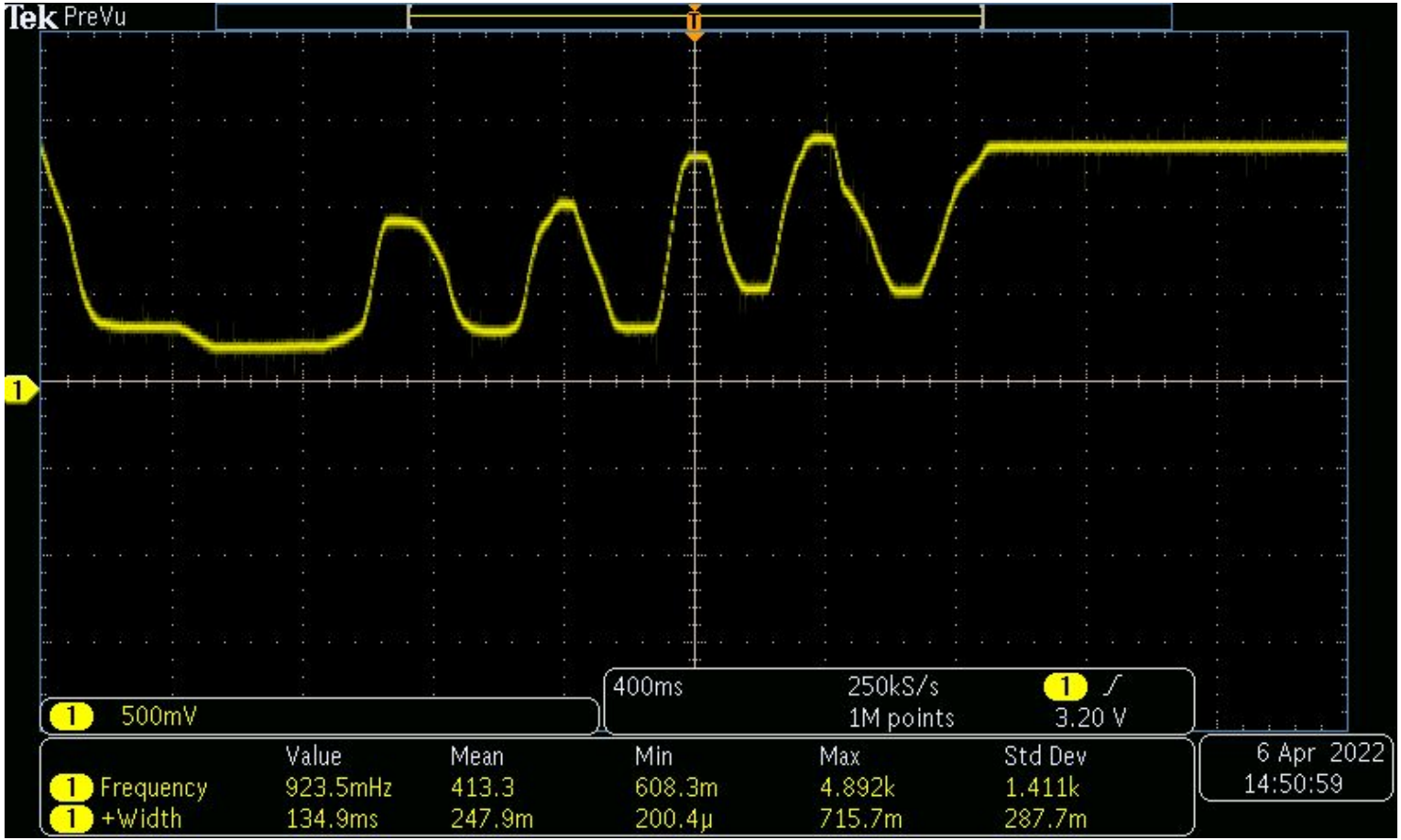
Digital to analogue converters (DAC) Basic_DAC_built_in.ino



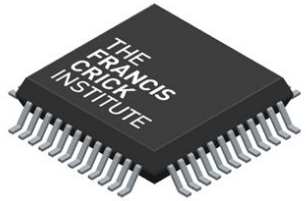
Digital to analogue converters (DAC) Basic_DAC_MCP4725.ino (Example 207)



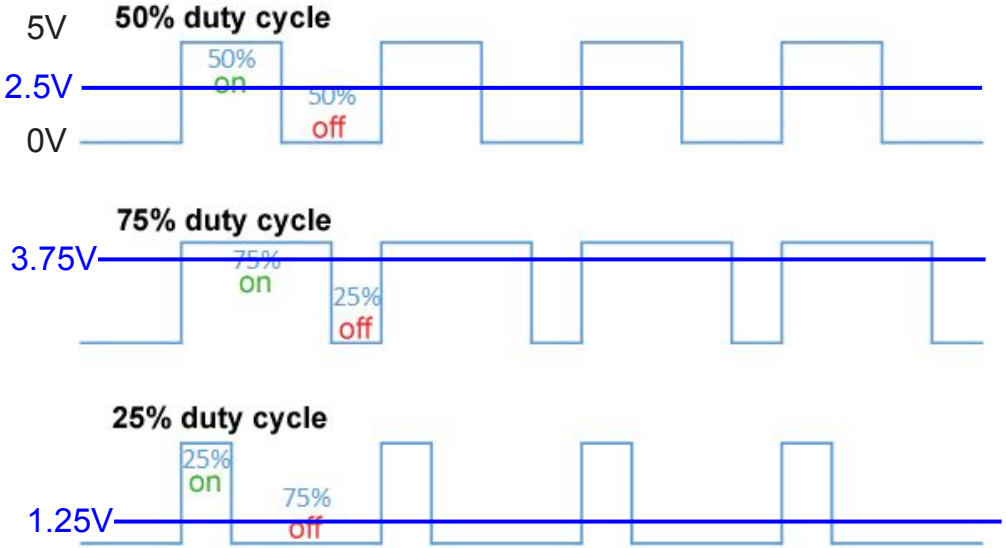
Digital to analogue converters (DAC) Basic_DAC_MCP4725.ino



Pulse Width Modulation (PWM)

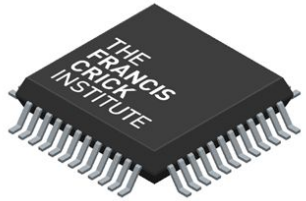


The average value of **voltage** fed to the **load** is controlled by turning the switch between supply and load on and off at a **fast rate**.



That is what we use to perform gradual control of actuators such as DC motors, Servo motors, Brightness levels on LEDs …

Pulse Width Modulation (PWM)



There are only certain pins that can do PWM

In the Arduino environment, the function to control PWM pins is called `analogWrite(Pin_number, dutyCycle)` (duty cycle has 8 bits and ranges 0-255 values)

In ESP32…

The `ledcWrite(Pin_number, dutyCycle)` is usually the function used in the ESP32 context.

Or downloading the ESP32_anlaogWrite library

https://github.com/erropix/ESP32_AnalogWrite

PWM: LED fade (Example 301)