1 Content of the electronics kit

You borrow your kit personally from FlexLab and it must be handed back to FlexLab at the end of the course.

When handing out, please check the contents with this list and let us know if anything is missing. When handing in the kit, we also ask you to check the contents and make us aware if you have lost something or something is defective.

- ☐ ESP32 Development Board
- ☐ 2 Breadboards (830 Point Solderless and Transparent)
- ☐ 2 Joysticks
- ☐ Potentiometer with cap
- ☐ GY-521 Accelerometer
- ☐ Character LCD /w IIC/I2C Serial Interface Adapter
- ☐ 0.91inch OLED LCD Display Module 128x32 I2C
- ☐ Micro USB Cable
- ☐ 3 push buttons
- ☐ Piezo Electronic Buzzer Alarm 95DB
- ☐ Small Toggle Switch Interruptor
- ☐ Cable Bundle
- ☐ various colored LEDs
- ☐ resistor
- ☐ RGB LED

2 Code Examples

2.1 Digital Input

In these first examples, we try to keep things simple and concentrate on controlling a single pin (LED) in the first example and then reading a pin (button) in the next example. In these examples we use digital input / output. Either there is power on the stick or there is not. Later in the next section we try with analog inputs and outputs, such as reading the resistance in a potentiometer or being able to control the brightness of an LED.

2.1.1 Blink example

The Blink example can be described as the ”hello world”-script for microprocessors. Pay attention to the first line where we include arduino, this line of code is only necessary because we use platoformIO as our editor. If you search the net for blink or other arduino examples you will often find examples where this line is missing. That is because the arduino IDE add that by itself.

```
1  #include <Arduino.h>
2
3  const int LEDPIN = 15;
4
5  void setup() { pinMode(LEDPIN, OUTPUT); }
6
7  void loop() {
8    digitalWrite(LEDPIN, HIGH);
9    delay(1000);
10   digitalWrite(LEDPIN, LOW);
11   delay(1000);
12 }
```

2.1.2 Button Example using ’Pullup’

This example demonstration the use og a built in pullup resistor. The reason we use it the pullup resistor is to avoid that the voltage is fluctiating when the button isn’t connected to the ground(gnd)

```
1  #include <Arduino.h>
2
3  const int LEDPIN = 15;
4  const int BUTTON_PIN = 22;
5  int buttonState = 0;
```

```
6
7  void setup() {
8    pinMode(LEDPIN, OUTPUT);
9    pinMode(BUTTON_PIN, INPUT_PULLUP);
10
11   // Init serial
12   Serial.begin(9600);
13 }
14
15 void loop() {
16   buttonState = digitalRead(BUTTON_PIN);
17   Serial.println(buttonState);
18
19   if (buttonState == 1) {
20     // LED OFF
21     digitalWrite(LEDPIN, LOW);
22   } else {
23     // LED ON
24     digitalWrite(LEDPIN, HIGH);
25   }
26 }
```

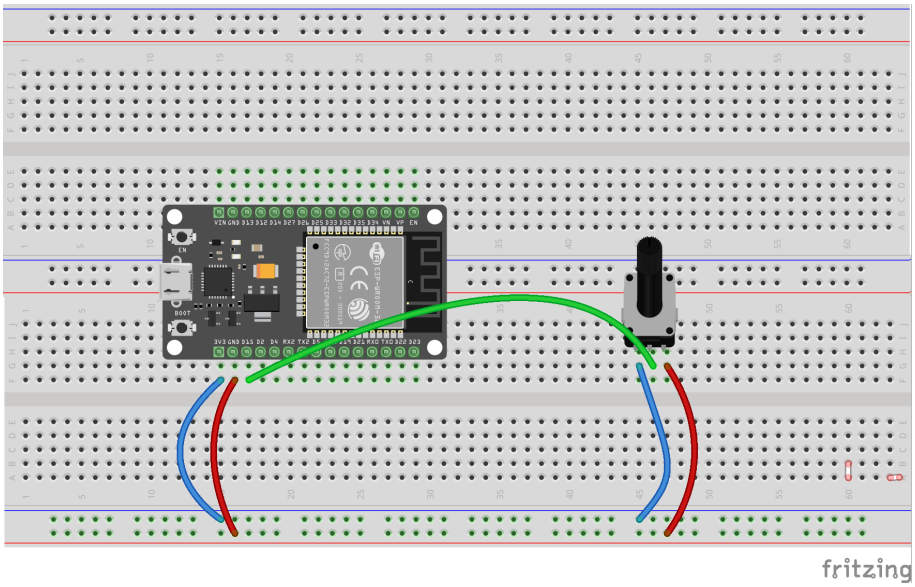
2.2 Analog Input

in previous examples we work with the digitalread function to check if there was power on a pin or not. But many sensors work differently. They are not just on or off but provide a value that can vary between 0 and 4095 (by default on an ESP32)

We will now look at an example where we read a potentiometer. you can change the setting on the potentiometer and read a changing value in our serial monitor

When you run the sample you should be able to see a changing value being printed in your serial monitor. When you turn the potentiometer, the value should change. Even when you are not touching the potentiometer we must expect the value to change a bit because the voltage always fluctuates a little bit.

It's time to set up your electrical circuit and test the example:

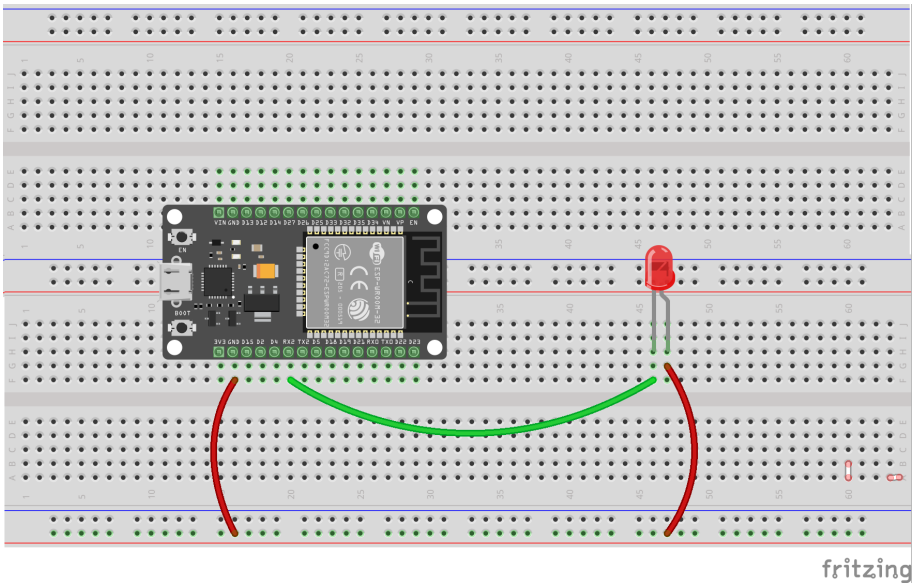



```
1
2  #include <Arduino.h>
3  const int potentiometerPin = 15;
4
5  void setup() {
6    pinMode(potentiometerPin, INPUT);
7    Serial.begin(9600);
8  }
9
10 void loop() {
11   int result = analogRead(potentiometerPin);
12   Serial.println(result);
13 }
```

2.2.1 PWM: Power with modulation

Now we have looked at how to read analog input using the analogRead () function. When we want to control output, a technique called PWM is used. The principle is that we turn the power on and off very quickly. Then the ”voltage” is controlled by how long the power is on in relation to how long it is off. You set a duty cycle that determines how much of the time we turn on the power, and frequency which sets the speed for how often we want to ”turn on and off” the power.

See example below:




```
1  #include <Arduino.h>
2
3  // the number of the LED pin
4  const int ledPin = 16;  // 16 corresponds to GPIO16
5
6  // setting PWM properties
7  const int freq = 5000; // how fast should the power change (
       smaller numbers makes LED flickering)
8  const int ledChannel = 0; // you have 8 channels (0-7) available
       to choose between
9  const int resolution = 8; //8bits (0-255)
10
11 void setup(){
12   // Set up channel with dutycycle and resolution
13   ledcSetup(ledChannel, freq, resolution);
14
15   // attach the channel to the GPIO to be controlled
16   ledcAttachPin(ledPin, ledChannel);
17 }
18
19 void loop(){
20   // increase the LED brightness
21   for(int dutyCycle = 0; dutyCycle <= 255; dutyCycle++){
22     // changing the LED brightness with PWM
23     ledcWrite(ledChannel, dutyCycle);
24     delay(15); // Hello world
25   }
26
27   // decrease the LED brightness
28   for(int dutyCycle = 255; dutyCycle >= 0; dutyCycle--){
```

```
29      // changing the LED brightness with PWM
30      ledcWrite(ledChannel, dutyCycle);
31      delay(15);
32    }
33  }
```