2025 TEEP Progress Report Week- 4

Used Wokwi to complete the tasks for Labs 1 through 5

(Before creating the physical project, please use **Wokwi** for online simulation to complete the initial design of the circuit and code. Then, use **Thonny** together with the **ESP32** development board to build the actual project. Please **record a video of the project** in operation, upload it to YouTube, and provide the video link(LAB7) so we can better understand your learning progress.)

Lab1 – Short Answer Questions:

1. When the Duty Cycle is higher, does the LED brightness increase or decrease?

Ans: The LED brightness increases because a higher duty cycle means the LED is ON for a longer portion of each cycle.

2. Does the LED become brighter or dimmer when the voltage is higher?

Ans: The LED becomes brighter since more voltage increases the current through the LED (within safe limits), making it emit more light.

3. When the input value exceeds 1024. How to adjust?

Ans: Limit (saturate) the input value to a maximum of 1023, since analog-to-digital converters (ADC) on microcontrollers like Arduino usually output values in the range 0–1023 (10-bit resolution).

Lab2- On Wokwi and Thonny, complete the following tasks:

Read the voltage value on the ADC and turn it on the LED.The brightness of the LED is changed by PWM output.

1. Connect the floating pin of the variable resistor to ADCXX (GPIO XX) and the LED to GPIOXX

    - **Function 1**: When the program starts, it detects the voltage on ADCXX and converts the voltage into PWM

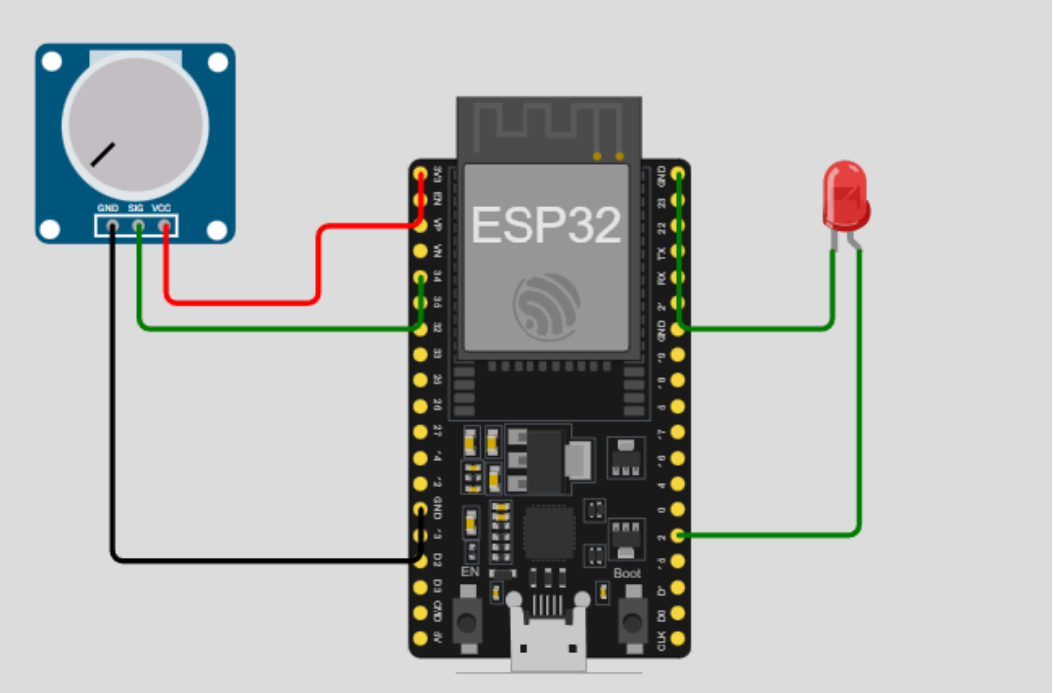
Circuit setup:

Potentiometer-

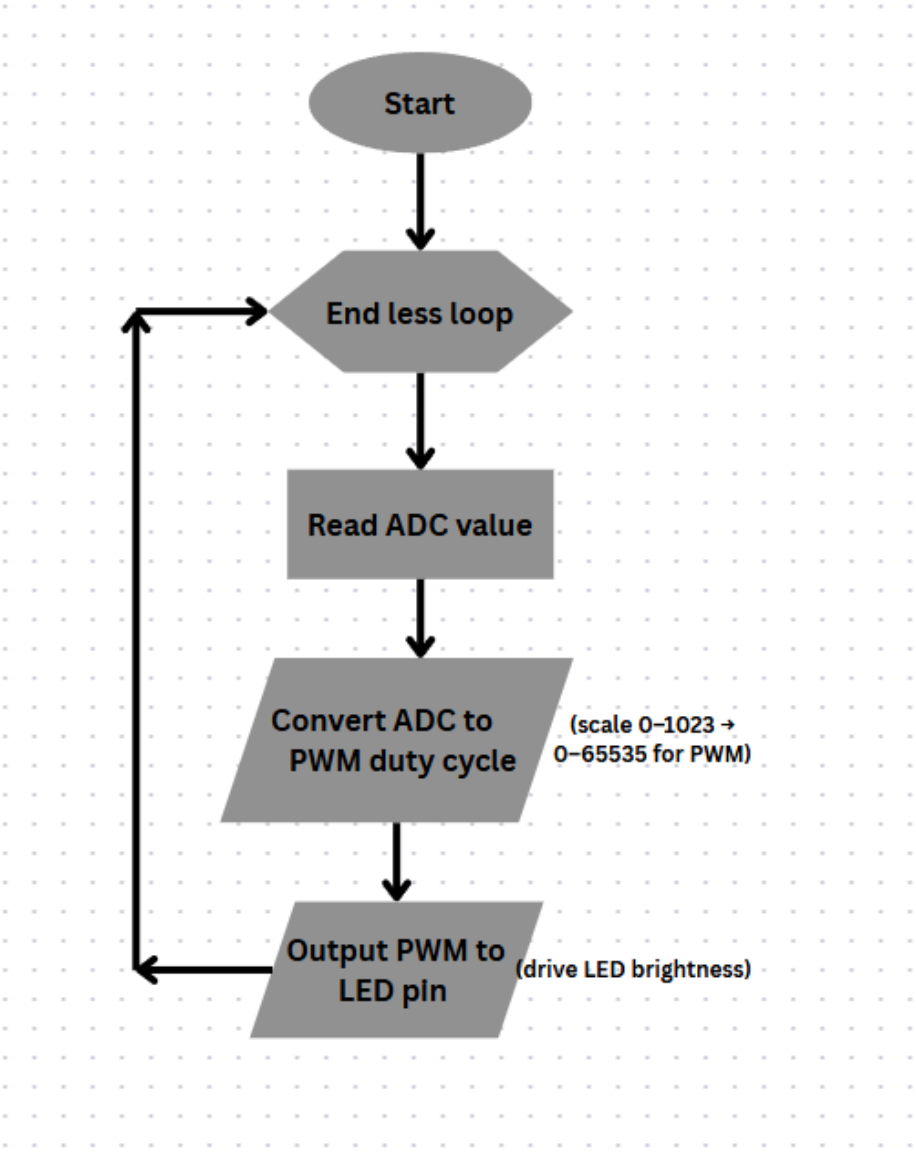
- Middle pin → ADC GPIO34
- One side → 3.3V
- Other side → GND

LED-

- Anode → GPIO2 (PWM output)
- Cathode → GND



Flow Chart :


Start
End less loop
Read ADC value
Convert ADC to PWM duty cycle
(scale 0–1023 → 0–65535 for PWM)
Output PWM to LED pin
(drive LED brightness)

Source Code :

```
# Lab 2 - Control LED brightness using Potentiometer (ADC + PWM)
# Tested in Thonny with ESP32 on Wokwi
from machine import Pin, ADC, PWM
import time
# ADC pin (connect potentiometer middle pin to GPIO34 for example)
pot = ADC(Pin(34))
pot.atten(ADC.ATTN_11DB)     # Full range: 0–3.3V
pot.width(ADC.WIDTH_10BIT)   # 10-bit resolution (0–1023)
# LED pin with PWM output
led = PWM(Pin(2), freq=1000)  # GPIO2 for LED
while True:
    value = pot.read()            # Read ADC value (0–1023)
    duty = int((value / 1023) * 65535)  # Scale to 16-bit PWM duty
    led.duty_u16(duty)            # Update LED brightness
    print("ADC:", value, " PWM:", duty)
    time.sleep(0.05)
```

Test Record :

Shell

```
ADC: 1023  PWM: 65535
ADC: 966  PWM: 61883
ADC: 855  PWM: 54772
ADC: 758  PWM: 48558
ADC: 683  PWM: 43754
ADC: 620  PWM: 39718
ADC: 558  PWM: 35746
ADC: 502  PWM: 32158
ADC: 454  PWM: 29083
ADC: 414  PWM: 26521
ADC: 375  PWM: 24023
ADC: 337  PWM: 21588
ADC: 306  PWM: 19602
ADC: 267  PWM: 17104
ADC: 228  PWM: 14606
ADC: 189  PWM: 12107
ADC: 149  PWM: 9545
ADC: 116  PWM: 7431
ADC: 91  PWM: 5829
ADC: 65  PWM: 4164
ADC: 51  PWM: 3267
ADC: 32  PWM: 2049
ADC: 11  PWM: 704
ADC: 6  PWM: 384
ADC: 0  PWM: 0
ADC: 0  PWM: 0
ADC: 0  PWM: 0
ADC: 0  PWM: 0
ADC: 0  PWM: 0
ADC: 0  PWM: 0
ADC: 0  PWM: 0
ADC: 0  PWM: 0
ADC: 0  PWM: 0
ADC: 6  PWM: 384
ADC: 35  PWM: 2242
ADC: 74  PWM: 4740
ADC: 124  PWM: 7943
ADC: 180  PWM: 11531
ADC: 244  PWM: 15631
ADC: 307  PWM: 19666
ADC: 367  PWM: 23510
ADC: 431  PWM: 27610
ADC: 495  PWM: 31710
ADC: 560  PWM: 35874
ADC: 620  PWM: 39718
ADC: 673  PWM: 43113
ADC: 713  PWM: 45675
ADC: 725  PWM: 46444
ADC: 717  PWM: 45932
ADC: 695  PWM: 44522
```

```
ADC: 0  PWM: 0
ADC: 0  PWM: 0
ADC: 0  PWM: 0
ADC: 0  PWM: 0
ADC: 0  PWM: 0
ADC: 0  PWM: 0
ADC: 0  PWM: 0
ADC: 0  PWM: 0
ADC: 0  PWM: 0
ADC: 0  PWM: 0
ADC: 0  PWM: 0
ADC: 0  PWM: 0
ADC: 0  PWM: 0
ADC: 0  PWM: 0
ADC: 38  PWM: 2434
ADC: 140  PWM: 8968
ADC: 246  PWM: 15759
ADC: 318  PWM: 20371
ADC: 341  PWM: 21845
ADC: 349  PWM: 22357
ADC: 360  PWM: 23062
ADC: 368  PWM: 23574
ADC: 390  PWM: 24984
ADC: 405  PWM: 25944
ADC: 405  PWM: 25944
ADC: 405  PWM: 25944
ADC: 405  PWM: 25944
ADC: 405  PWM: 25944
ADC: 405  PWM: 25944
ADC: 568  PWM: 36386
ADC: 587  PWM: 37604
ADC: 644  PWM: 41255
ADC: 709  PWM: 45419
ADC: 716  PWM: 45868
ADC: 724  PWM: 46380
ADC: 754  PWM: 48302
```

Lab3 – On Wokwi and Thonny, complete the following tasks:

Read the voltage value on the ADC and turn it on the LED.The brightness of the LED is changed by PWM output.

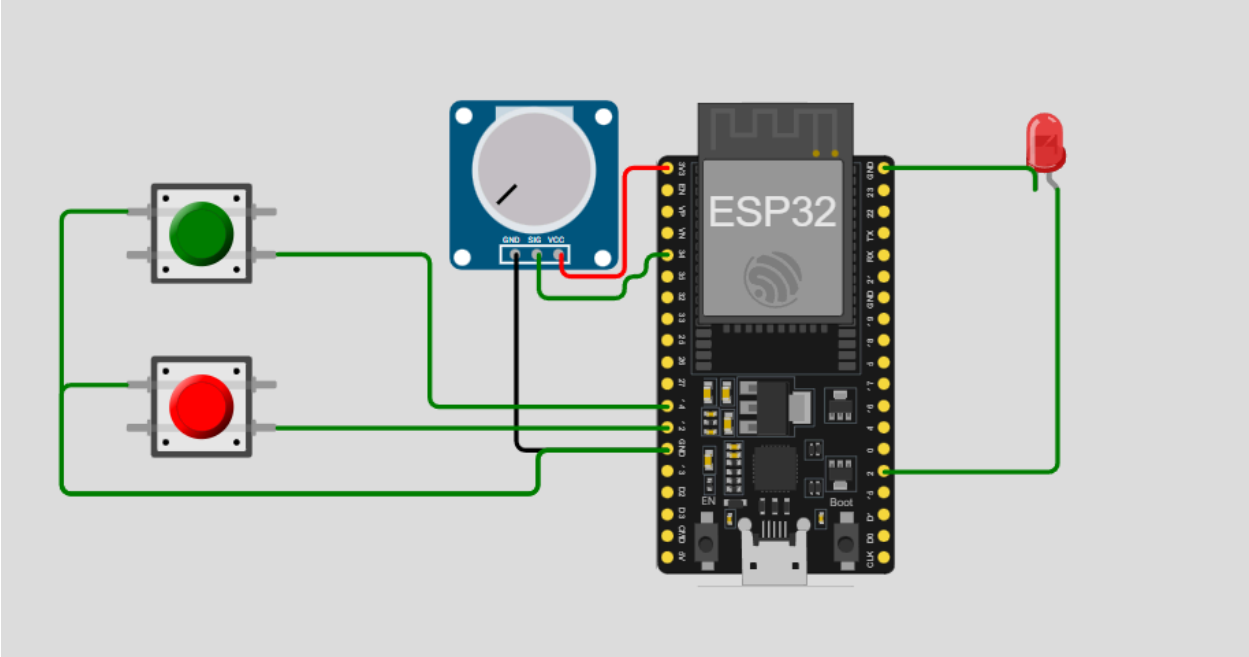
1. Connect SW1 to GPIOXX, SW2 to GPIOXX, and the LED to GPIOXX.

ADC subroutine: Detects voltage and converts it into a PWM duty cycle, outputting it to the LED on GPIO XX. This changes the LED's brightness. The higher the voltage, the lower the brightness.

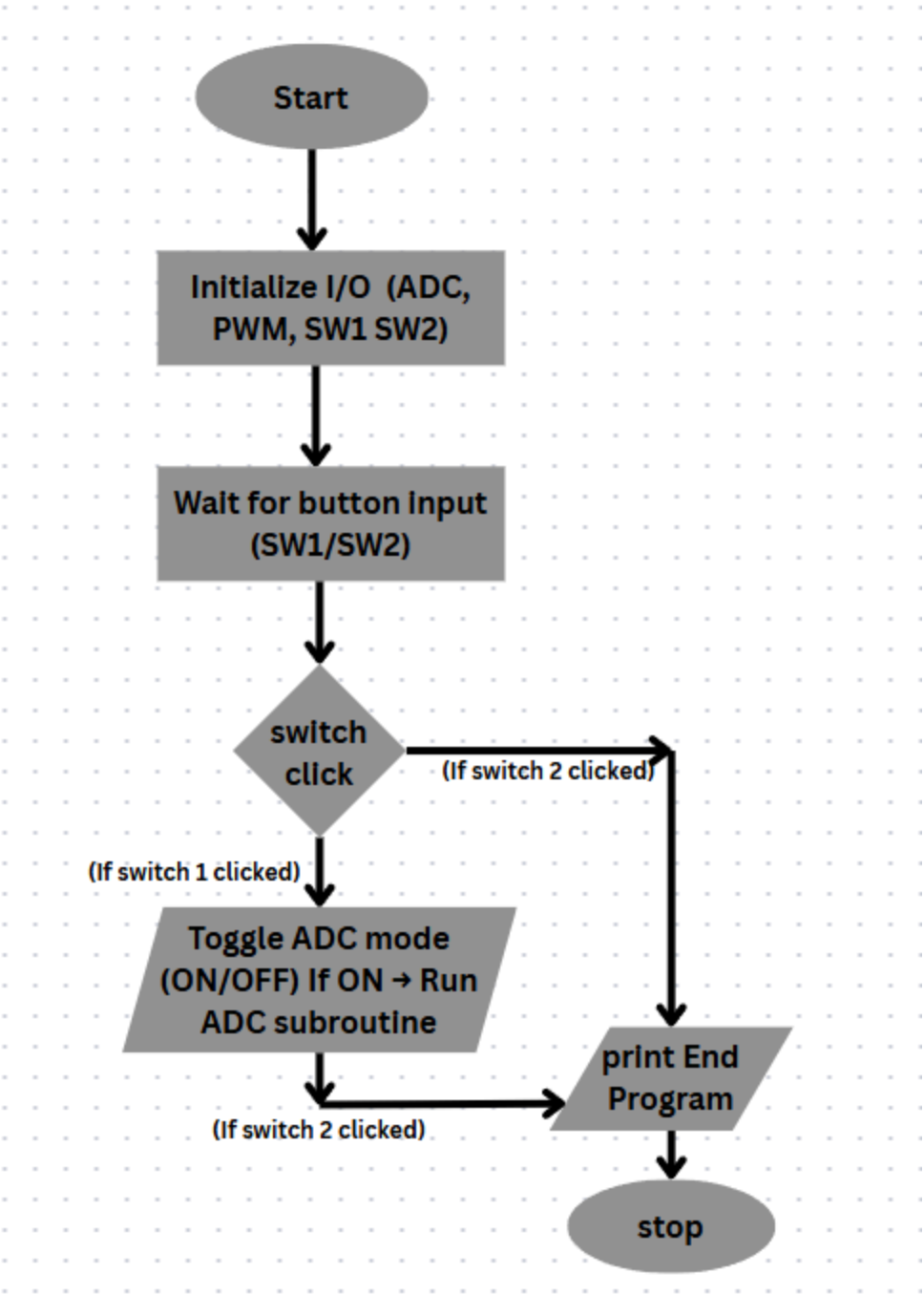
- **Function 1**: Start the ADC program when SW1 is pressed an odd number of times, and stop the ADC program when SW1 is pressed an even number of times.
- **Function 2**: When SW2 is pressed, End Program! is displayed on the Shell and the entire program is closed.

Circuit Setup

- SW1 → GPIO12 (with pull-up resistor).
- SW2 → GPIO14 (with pull-up resistor).
- Potentiometer middle → GPIO34 (ADC input).
- LED → GPIO2 (PWM output).



Flow Chart :


Start
Initialize I/O (ADC, PWM, SW1 SW2)
Wait for button input (SW1/SW2)
switch click
(If switch 2 clicked)
(If switch 1 clicked)
Toggle ADC mode (ON/OFF) If ON → Run ADC subroutine
print End Program
(If switch 2 clicked)
stop

Source Code :

```
from machine import Pin, ADC, PWM
import time
import sys

# Setup ADC (Potentiometer on GPIO34)
pot = ADC(Pin(34))
pot.atten(ADC.ATTN_11DB)       # Full 0–3.3V range
pot.width(ADC.WIDTH_10BIT)     # 10-bit resolution (0–1023)

# Setup LED with PWM (GPIO2)
led = PWM(Pin(2), freq=1000)

# Setup switches
sw1 = Pin(12, Pin.IN, Pin.PULL_UP)   # Switch 1 (toggle ADC mode)
sw2 = Pin(14, Pin.IN, Pin.PULL_UP)   # Switch 2 (end program)
adc_enabled = False       # Start with ADC OFF
sw1_last = 1
sw2_last = 1

print("Program started. Press SW1 to toggle ADC, SW2 to quit.")

while True:
    # --- Check SW1 (toggle ADC mode) ---
    sw1_state = sw1.value()
    if sw1_last == 1 and sw1_state == 0:   # Button pressed (falling edge)
        adc_enabled = not adc_enabled
        print("SW1 pressed → ADC Enabled =", adc_enabled)
        if not adc_enabled:
            led.duty_u16(0)   # Turn LED OFF when ADC is stopped
        time.sleep(0.2)       # Debounce
    sw1_last = sw1_state

    # --- Check SW2 (exit program) ---
    sw2_state = sw2.value()
    if sw2_last == 1 and sw2_state == 0:
        print("End Program!")
        led.deinit()
        sys.exit()
    sw2_last = sw2_state
    # --- If ADC mode enabled, update LED brightness ---
    if adc_enabled:
        value = pot.read()   # Read ADC (0–1023)
        duty = int((1023 - value) / 1023 * 65535)
        # Inverse mapping: higher voltage = dimmer LED
        led.duty_u16(duty)
        print("ADC:", value, " Duty:", duty)

    time.sleep(0.05)   # Small delay
```

Test Record :

Shell

```
MPY: soft reboot
Program started. Press SW1 to toggle ADC, SW2 to quit.
SW1 pressed → ADC Enabled = True
ADC: 383  Duty: 40999
ADC: 383  Duty: 40999
ADC: 383  Duty: 40999
ADC: 383  Duty: 40999
ADC: 383  Duty: 40999
ADC: 383  Duty: 40999
ADC: 382  Duty: 41063
ADC: 381  Duty: 41127
ADC: 383  Duty: 40999
ADC: 383  Duty: 40999
ADC: 383  Duty: 40999
ADC: 383  Duty: 40999
ADC: 383  Duty: 40999
ADC: 383  Duty: 40999
ADC: 383  Duty: 40999
ADC: 383  Duty: 40999
ADC: 383  Duty: 40999
ADC: 383  Duty: 40999
ADC: 383  Duty: 40999
ADC: 383  Duty: 40999
ADC: 382  Duty: 41063
ADC: 383  Duty: 40999
ADC: 383  Duty: 40999
ADC: 383  Duty: 40999
ADC: 383  Duty: 40999
ADC: 383  Duty: 40999
ADC: 383  Duty: 40999
ADC: 383  Duty: 40999
ADC: 383  Duty: 40999
ADC: 383  Duty: 40999
ADC: 383  Duty: 40999
ADC: 383  Duty: 40999
ADC: 383  Duty: 40999
ADC: 383  Duty: 40999
ADC: 383  Duty: 40999
ADC: 383  Duty: 40999
ADC: 383  Duty: 40999
ADC: 383  Duty: 40999
ADC: 383  Duty: 40999
ADC: 383  Duty: 40999
ADC: 382  Duty: 41063
ADC: 380  Duty: 41191
ADC: 383  Duty: 40999
ADC: 383  Duty: 40999
ADC: 393  Duty: 40358
ADC: 403  Duty: 39718
ADC: 427  Duty: 38180
ADC: 452  Duty: 36579
```

```
ets Jul 29 2019 12:21:46

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0030,len:4728
load:0x40078000,len:14888
load:0x40080400,len:3368
entry 0x400805cc
Program started. Press SW1 to toggle ADC, SW2 to quit.
SW1 pressed → ADC Enabled = True
ADC: 0  Duty: 65535
ADC: 0  Duty: 65535
ADC: 0  Duty: 65535
ADC: 0  Duty: 65535
ADC: 0  Duty: 65535
ADC: 0  Duty: 65535
ADC: 0  Duty: 65535
ADC: 0  Duty: 65535
ADC: 0  Duty: 65535
ADC: 0  Duty: 65535
End Program!
MicroPython v1.22.0 on 2023-12-27; Generic ESP32 module with ESP32
Type "help()" for more information.
>>>
```

Lab4 – On Wokwi and Thonny, complete the following tasks:

Read the voltage value on the ADC and turn it on the LED.The brightness of the LED is changed by PWM output.

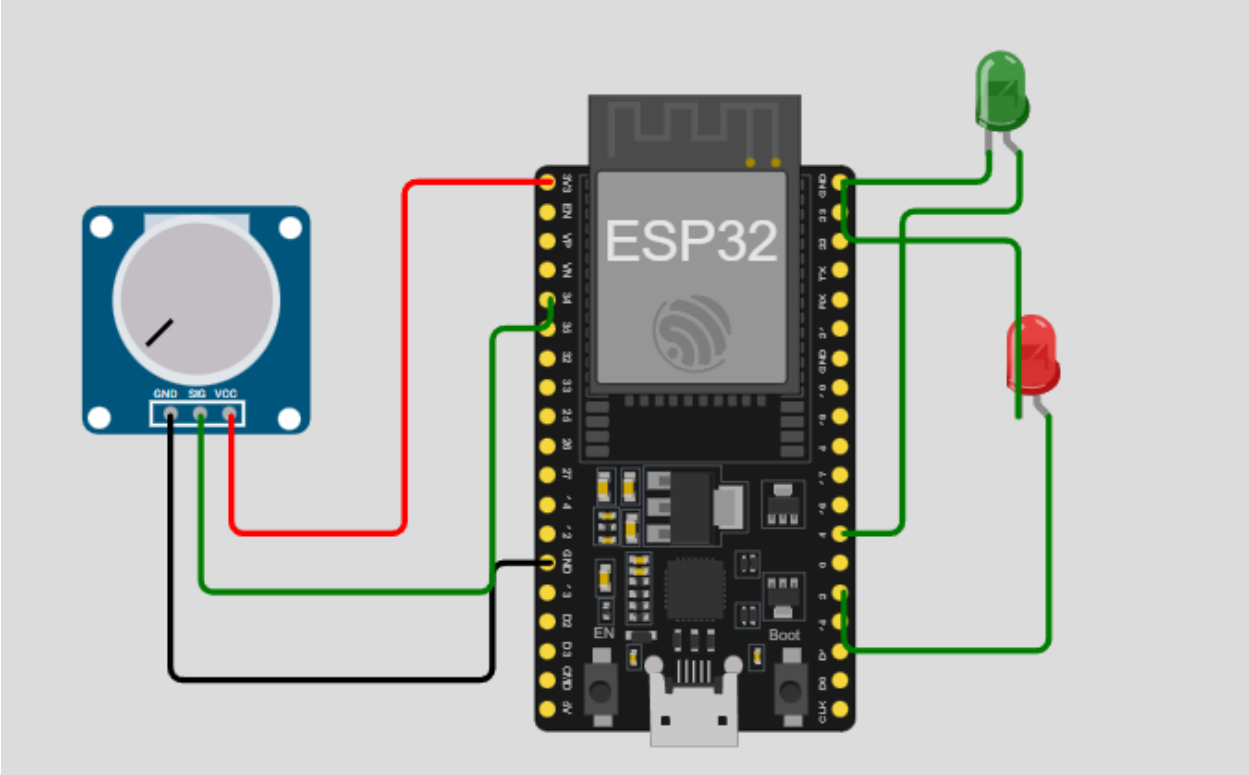
1. Connect Potentiometer to GPIOXX, and the LED1 to GPIOXX,LED2 to GPIOXX.

ADC subroutine: Detects voltage and converts it into a PWM duty cycle, outputting it to the LED on GPIO XX. This changes the LED's brightness. The higher the voltage, the lower the brightness.

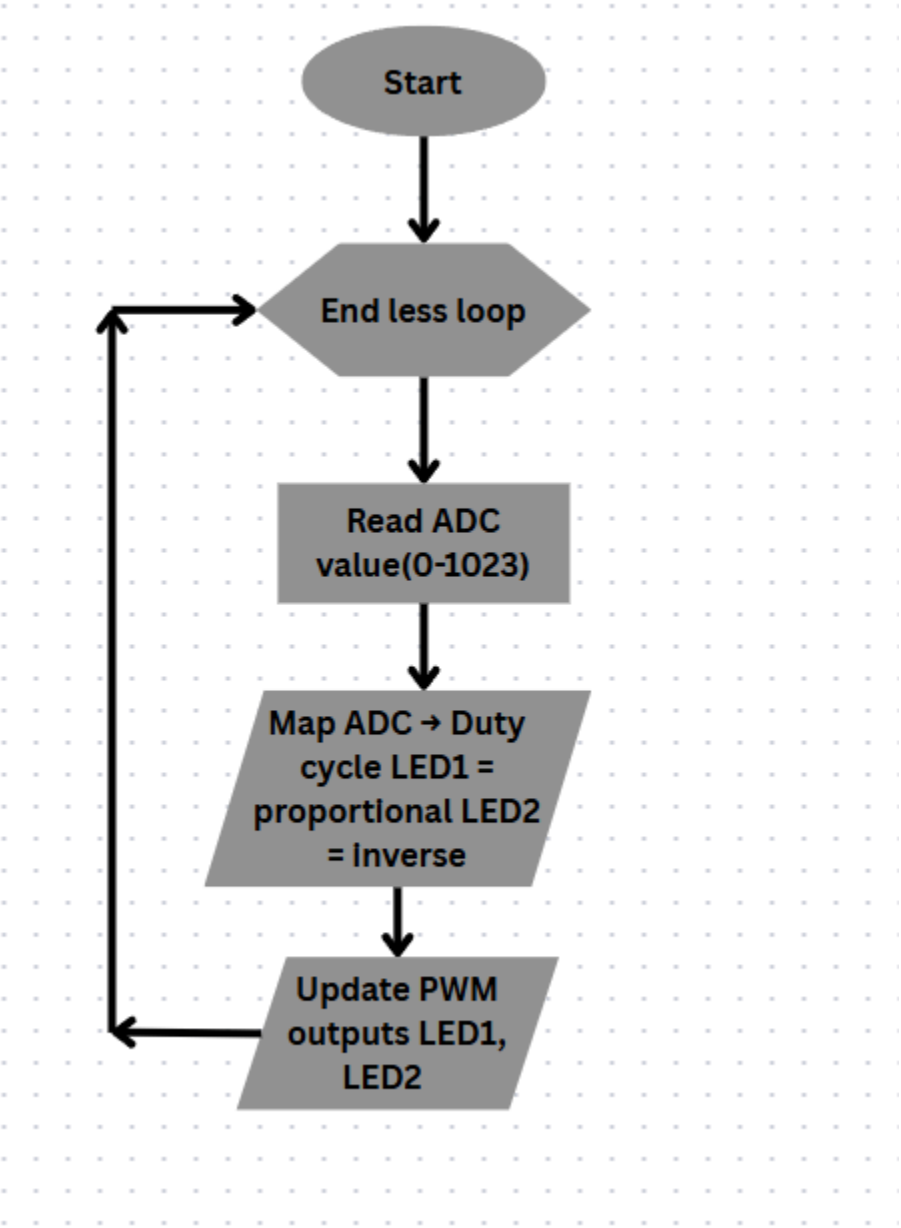
- Function 1: When the potentiometer is turned all the way to the left, LED1 should be completely off while LED2 shines at full brightness. As the potentiometer is rotated toward the right, the brightness of LED1 gradually increases while LED2 gradually dims. Finally, when the potentiometer reaches the far right position, LED1 should be fully bright and LED2 should be completely off.

Circuit setup :

- Potentiometer middle → GPIO34 (ADC), sides → 3.3V & GND.
- LED1 GPIO2 → Resistor → GND.
- LED2 GPIO4 → Resistor → GND.



Flow Chart :


Start
End less loop
Read ADC value(0-1023)
Map ADC → Duty cycle LED1 = proportional LED2 = Inverse
Update PWM outputs LED1, LED2

Source Code :

```
from machine import Pin, ADC, PWM
import time
# Setup ADC (Potentiometer on GPIO34)
pot = ADC(Pin(34))
pot.atten(ADC.ATTN_11DB)       # Full range 0–3.3V
pot.width(ADC.WIDTH_10BIT)     # 10-bit resolution (0–1023)
# Setup LEDs with PWM
led1 = PWM(Pin(2), freq=1000)   # LED1 on GPIO2
led2 = PWM(Pin(4), freq=1000)   # LED2 on GPIO4
while True:
    value = pot.read()   # Read ADC (0–1023)
    # Scale to 16-bit duty cycle (0–65535)
    duty_led1 = int((value / 1023) * 65535)        # Direct mapping
    duty_led2 = int(((1023 - value) / 1023) * 65535)  # Inverse mapping
    # Apply PWM values
    led1.duty_u16(duty_led1)
    led2.duty_u16(duty_led2)
    # Debug print
    print("ADC:", value, " LED1 duty:", duty_led1, " LED2 duty:", duty_led2)
    time.sleep(0.05)
```

Test Records :

Shell

```
ADC: 81  LED1 duty: 5188  LED2 duty: 60346
ADC: 81  LED1 duty: 5188  LED2 duty: 60346
ADC: 80  LED1 duty: 5124  LED2 duty: 60410
ADC: 80  LED1 duty: 5124  LED2 duty: 60410
ADC: 77  LED1 duty: 4932  LED2 duty: 60602
ADC: 81  LED1 duty: 5188  LED2 duty: 60346
ADC: 80  LED1 duty: 5124  LED2 duty: 60410
ADC: 80  LED1 duty: 5124  LED2 duty: 60410
ADC: 80  LED1 duty: 5124  LED2 duty: 60410
ADC: 82  LED1 duty: 5253  LED2 duty: 60281
ADC: 82  LED1 duty: 5253  LED2 duty: 60281
ADC: 80  LED1 duty: 5124  LED2 duty: 60410
ADC: 80  LED1 duty: 5124  LED2 duty: 60410
ADC: 80  LED1 duty: 5124  LED2 duty: 60410
ADC: 79  LED1 duty: 5060  LED2 duty: 60474
ADC: 80  LED1 duty: 5124  LED2 duty: 60410
ADC: 81  LED1 duty: 5188  LED2 duty: 60346
ADC: 80  LED1 duty: 5124  LED2 duty: 60410
ADC: 79  LED1 duty: 5060  LED2 duty: 60474
ADC: 71  LED1 duty: 4548  LED2 duty: 60986
ADC: 55  LED1 duty: 3523  LED2 duty: 62011
ADC: 29  LED1 duty: 1857  LED2 duty: 63677
ADC: 0  LED1 duty: 0  LED2 duty: 65535
ADC: 0  LED1 duty: 0  LED2 duty: 65535
ADC: 0  LED1 duty: 0  LED2 duty: 65535
ADC: 0  LED1 duty: 0  LED2 duty: 65535
ADC: 0  LED1 duty: 0  LED2 duty: 65535
ADC: 0  LED1 duty: 0  LED2 duty: 65535
ADC: 0  LED1 duty: 0  LED2 duty: 65535
ADC: 0  LED1 duty: 0  LED2 duty: 65535
ADC: 0  LED1 duty: 0  LED2 duty: 65535
ADC: 0  LED1 duty: 0  LED2 duty: 65535
ADC: 0  LED1 duty: 0  LED2 duty: 65535
ADC: 0  LED1 duty: 0  LED2 duty: 65535
ADC: 0  LED1 duty: 0  LED2 duty: 65535
ADC: 0  LED1 duty: 0  LED2 duty: 65535
ADC: 0  LED1 duty: 0  LED2 duty: 65535
ADC: 50  LED1 duty: 3203  LED2 duty: 62331
ADC: 104  LED1 duty: 6662  LED2 duty: 58872
ADC: 156  LED1 duty: 9993  LED2 duty: 55541
ADC: 207  LED1 duty: 13260  LED2 duty: 52274
ADC: 253  LED1 duty: 16207  LED2 duty: 49327
ADC: 287  LED1 duty: 18385  LED2 duty: 47149
ADC: 340  LED1 duty: 21780  LED2 duty: 43754
ADC: 371  LED1 duty: 23766  LED2 duty: 41768
ADC: 391  LED1 duty: 25048  LED2 duty: 40486
ADC: 393  LED1 duty: 25176  LED2 duty: 40358
ADC: 383  LED1 duty: 24535  LED2 duty: 40999
ADC: 364  LED1 duty: 23318  LED2 duty: 42216
```

```
mode:DIO, clock div:2
load:0x3fff0030,len:4728
load:0x40078000,len:14888
load:0x40080400,len:3368
entry 0x400805cc
ADC: 379  LED1 duty: 24279  LED2 duty: 41255
ADC: 379  LED1 duty: 24279  LED2 duty: 41255
ADC: 379  LED1 duty: 24279  LED2 duty: 41255
ADC: 371  LED1 duty: 23766  LED2 duty: 41768
ADC: 231  LED1 duty: 14798  LED2 duty: 50736
ADC: 114  LED1 duty: 7303  LED2 duty: 58231
ADC: 80  LED1 duty: 5124  LED2 duty: 60410
ADC: 0  LED1 duty: 0  LED2 duty: 65535
ADC: 0  LED1 duty: 0  LED2 duty: 65535
ADC: 0  LED1 duty: 0  LED2 duty: 65535
ADC: 0  LED1 duty: 0  LED2 duty: 65535
ADC: 0  LED1 duty: 0  LED2 duty: 65535
ADC: 8  LED1 duty: 512  LED2 duty: 65022
ADC: 152  LED1 duty: 9737  LED2 duty: 55797
ADC: 246  LED1 duty: 15759  LED2 duty: 49775
ADC: 314  LED1 duty: 20115  LED2 duty: 45419
ADC: 375  LED1 duty: 24023  LED2 duty: 41511
ADC: 390  LED1 duty: 24984  LED2 duty: 40550
ADC: 390  LED1 duty: 24984  LED2 duty: 40550
ADC: 390  LED1 duty: 24984  LED2 duty: 40550
ADC: 242  LED1 duty: 15502  LED2 duty: 50032
ADC: 42  LED1 duty: 2690  LED2 duty: 62844
ADC: 30  LED1 duty: 1921  LED2 duty: 63613
ADC: 30  LED1 duty: 1921  LED2 duty: 63613
ADC: 30  LED1 duty: 1921  LED2 duty: 63613
ADC: 106  LED1 duty: 6790  LED2 duty: 58744
ADC: 280  LED1 duty: 17937  LED2 duty: 47597
ADC: 341  LED1 duty: 21845  LED2 duty: 43690
ADC: 341  LED1 duty: 21845  LED2 duty: 43690
ADC: 341  LED1 duty: 21845  LED2 duty: 43690
ADC: 341  LED1 duty: 21845  LED2 duty: 43690
ADC: 341  LED1 duty: 21845  LED2 duty: 43690
ADC: 341  LED1 duty: 21845  LED2 duty: 43690
ADC: 341  LED1 duty: 21845  LED2 duty: 43690
```

Lab5 – On Wokwi and Thonny, complete the following tasks:

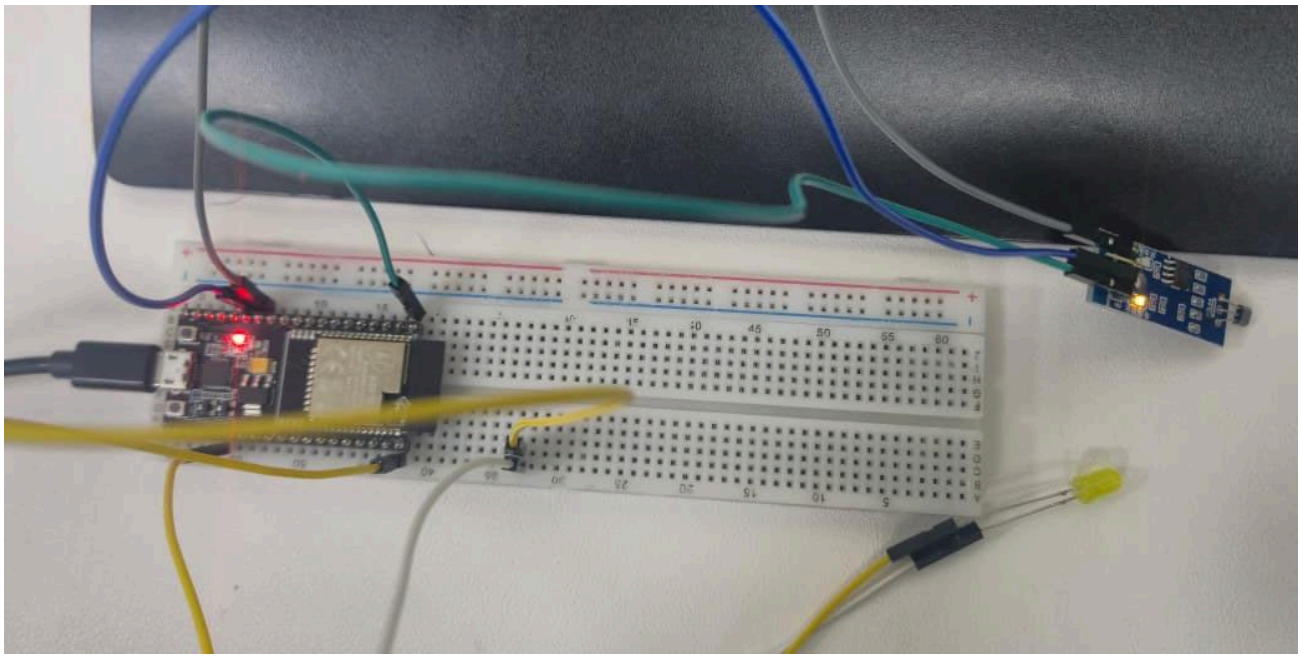
Using the ADC and a Hall sensor module, detect the number of times a magnet passes by and display the counting result.

1. Connect Hall sensor module to GPIOXX and the LED to GPIOXX.

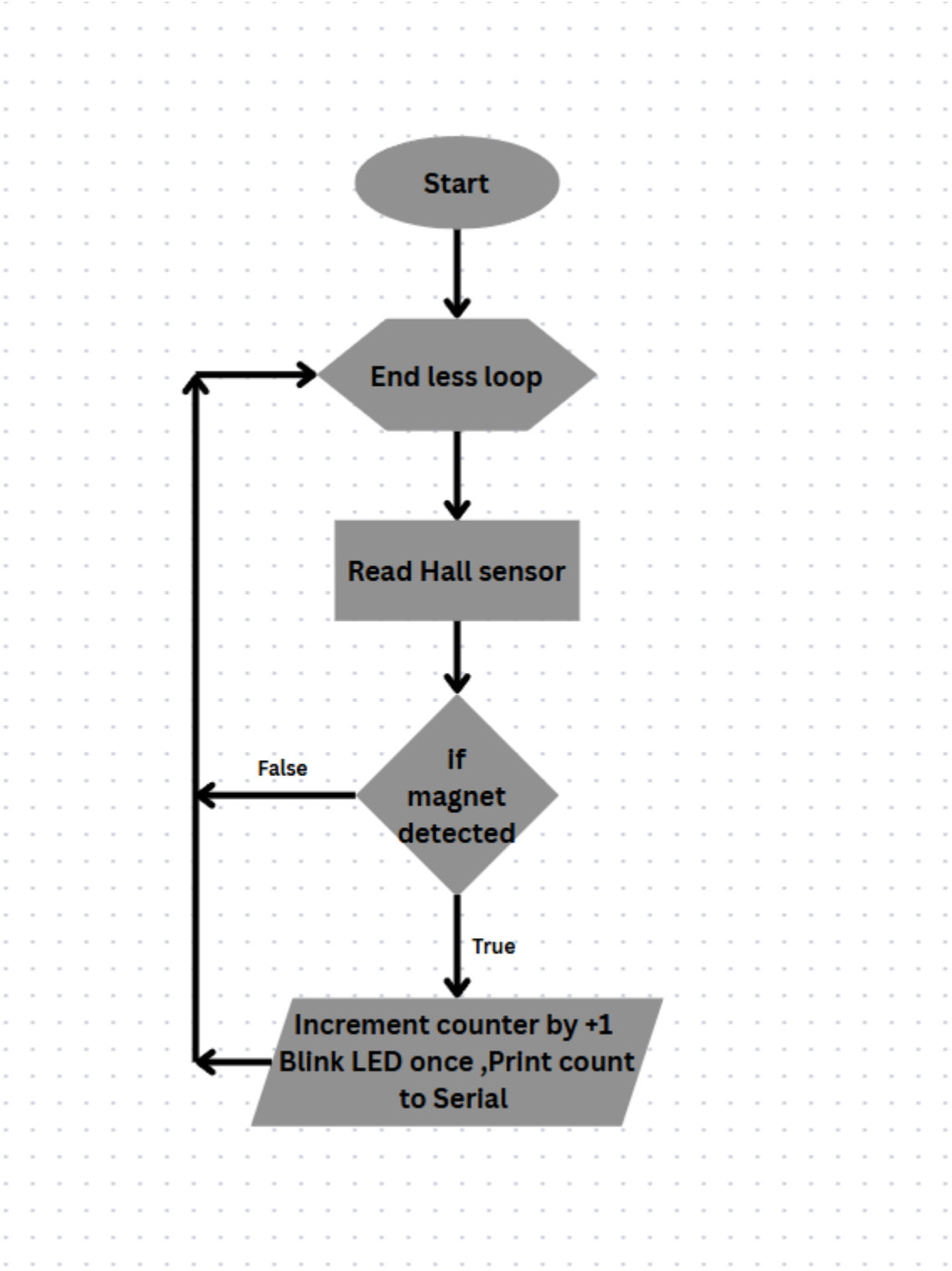
- Function 1: The Hall sensor module is fixed at a position where the magnet will pass. Using a small magnet, each time the magnet passes, the Hall sensor outputs a signal; when detected, the count increases by +1 and the LED turns on, while when not detected the LED remains off. The result is then output to the Serial Monitor.

Circuit setup :

- Hall sensor module OUT → GPIO12
- VCC → 3.3V, GND → GND
- LED GPIO2 → Resistor → GND
- (Small magnet near sensor)



Flow Chart :


Start
End less loop
Read Hall sensor
If magnet detected
False
True
Increment counter by +1
Blink LED once ,Print count to Serial

Source code :

```
from machine import Pin
import time

# Setup Hall sensor input (GPIO 12 for example)
hall_sensor = Pin(12, Pin.IN)

# Setup LED output (GPIO 2 for example)
led = Pin(2, Pin.OUT)

# Counter variable
count = 0
last_state = 1  # Assume Hall sensor idle state is HIGH

print("Hall sensor magnet counter started...")

while True:
    current_state = hall_sensor.value()

    # Detect magnet (falling edge: HIGH → LOW)
    if last_state == 1 and current_state == 0:
        count += 1
        led.value(1)  # Turn LED ON
        print("Magnet detected! Count =", count)
        time.sleep(0.2)  # Debounce delay
    else:
        led.value(0)  # LED OFF if no magnet

    last_state = current_state
    time.sleep(0.05)
```

Test Record:

```
Shell
>>> %Run -c $EDITOR_CONTENT

  MPY: soft reboot
  Hall sensor magnet counter started...
  Magnet detected! Count = 1
  Magnet detected! Count = 2
  Magnet detected! Count = 3
  Magnet detected! Count = 4
  ets Jul 29 2019 12:21:46

  rst:0x1 (POWERON_RESET),boot:0x33 (SPI_FAST_FLASH_BOOT)
  configsip: 0, SPIWP:0xee
  clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
  mode:DIO, clock div:2
  load:0x3fff0030,len:4112
  load:0x40078000,len:15072
  load:0x40080400,len:4
  load:0x40080404,len:3332
  entry 0x400805ac
  MicroPython v1.26.0 on 2025-08-09; Generic ESP32 module with ESP32
  Type "help()" for more information.

>>>
```

Note :

The following Requirements for this task are not available in **WOKWI sim .** The task was implemented directly and run in **Thonny s/w**

Lab6 learning reflection or comments(100 words)

The content of Learning reflection or comments can include your thoughts on the course, such as whether it was too difficult or too easy. You may also share something memorable that happened during the week. Overall, it is similar to a weekly journal, focusing on your learning experiences and personal feelings.

Learning Reflection or Comments :

This week's lab was interesting and helped me understand more about using sensors and LEDs with the ESP32. At first, I thought the tasks might be difficult, but after practicing step by step on Wokwi and Thonny, it became easier. I enjoyed seeing how the potentiometer and Hall sensor could control the brightness or counting in real time.

The most memorable part of this week was the Hall sensor experiment. At first, I wasn't sure if the sensor would detect the magnet properly, but once I tested it, I was excited to see the counter increase each time the magnet passed by. The LED turning on at the same time gave clear feedback, and it made the project feel very practical, like something that could be used in a real application such as speed measurement or counting objects.

Overall, this lab was very engaging and helped me improve my coding, debugging, and problem-solving skills. It also showed me that even simple hardware like LEDs, resistors, and sensors can create useful and fun projects when combined with programming. I feel more motivated to keep learning and trying out new ideas.

Lab7 Video Link

Record a video of approximately 3 minutes to explain your research process. The video should include the following:

- A description of the tools or methods used during your research process
- A presentation of the results and outcomes from LAB1 to LAB7.

Link: https://youtu.be/ZQRjuOVlgWI