ESSE 2220 – Lab 2 Report

Part 1

Course: ESSE 2220
Lab Title: Software Debouncing with GPIO **Date Performed:** 2025-09-19
Date Submitted: 2025-09-26

---

1. Names & Group Info

Group Number: GROUP 5
Members: Yathharthha Kaushal · Owen Oliver

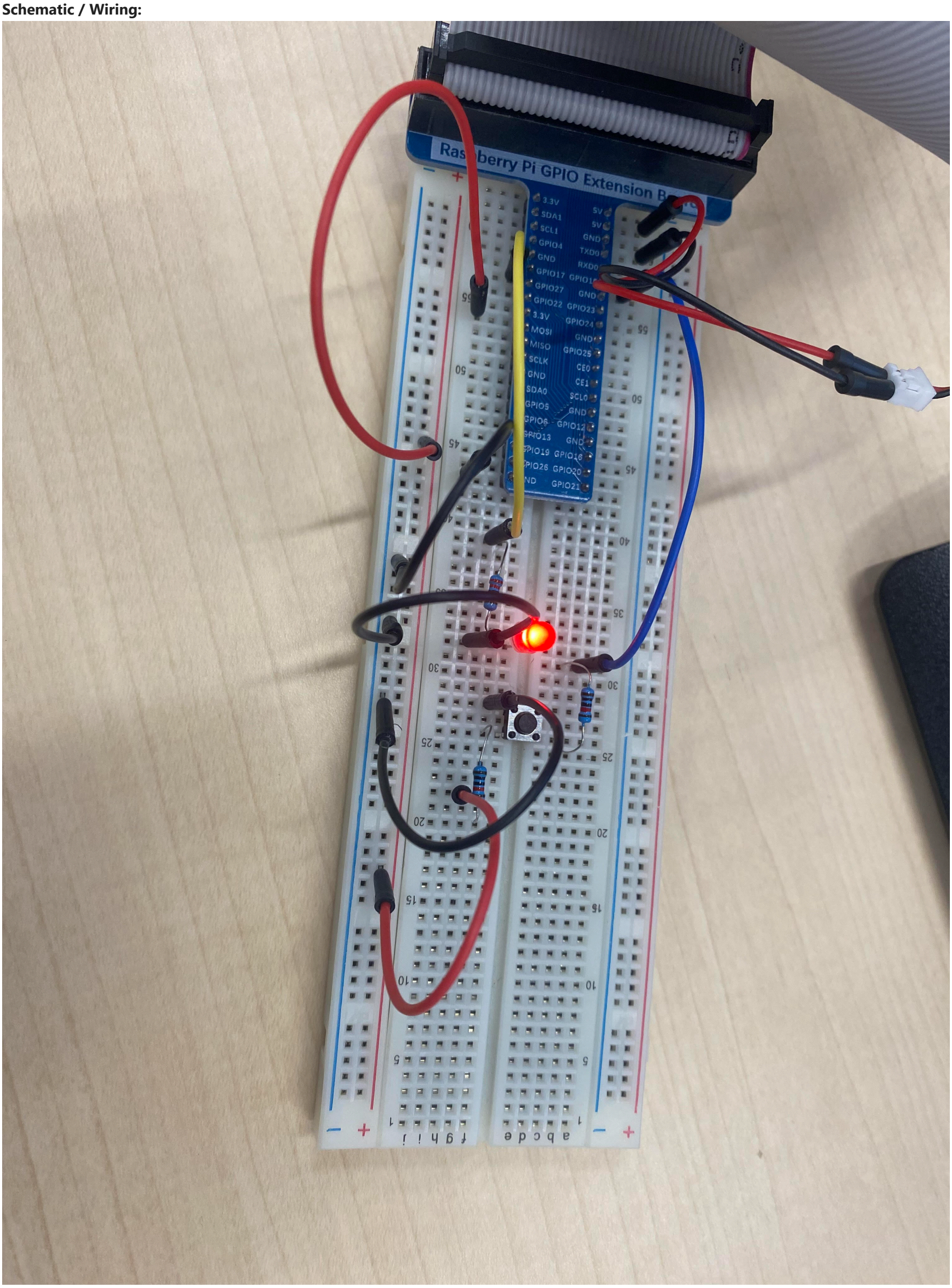
---

2. Circuit Setup

GPIO Pin Used for LED: `GPIO 17` **GPIO Pin Used for Push Button:** `GPIO 18`

We connected the LED (with resistor) to GPIO 17 and ground, and we connected the pushbutton to GPIO 18.

Schematic / Wiring:



Program:

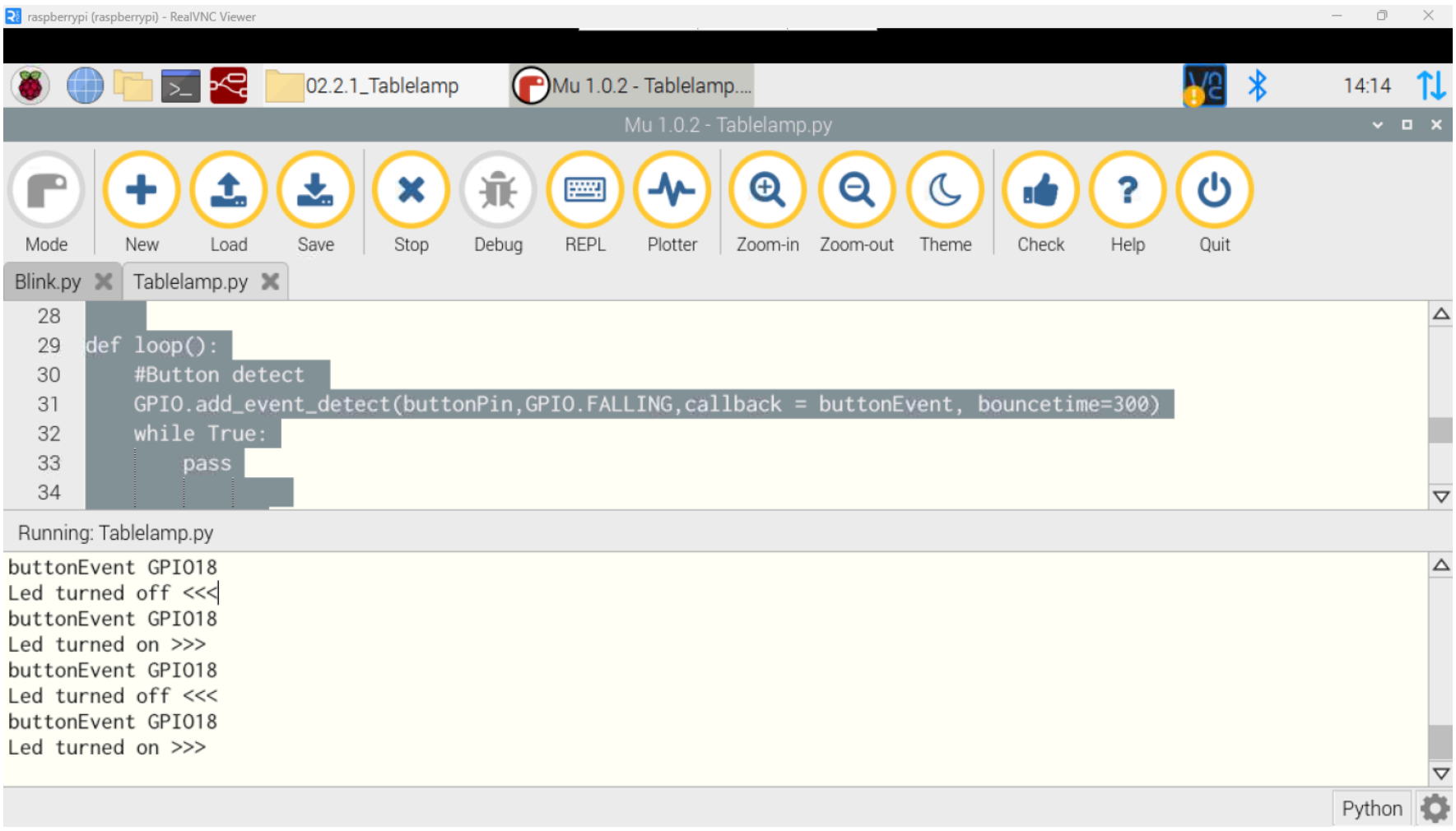
```python
#!/usr/bin/env python3
########################################################################
# Filename    : Tablelamp.py
# Description : a DIY MINI table lamp
# auther      : www.freenove.com
# modification: 2019/12/28
########################################################################
import RPi.GPIO as GPIO

ledPin = 17      # define ledPin
buttonPin = 18    # define buttonPin
```

```
ledState = False

def setup():
    GPIO.setmode(GPIO.BCM)          # use PHYSICAL GPIO Numbering
    GPIO.setup(ledPin, GPIO.OUT)     # set ledPin to OUTPUT mode
    GPIO.setup(buttonPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)     # set buttonPin to PULL UP INPUT mode

def buttonEvent(channel): # When button is pressed, this function will be executed
    global ledState
    print ('buttonEvent GPIO%d' %channel)
    ledState = not ledState
    if ledState :
        print ('Led turned on >>>')
    else :
        print ('Led turned off <<<')
    GPIO.output(ledPin,ledState)

def loop():
    #Button detect
    nMilliSeconds = 300
    GPIO.add_event_detect(buttonPin,GPIO.FALLING,callback = buttonEvent, bouncetime=nMilliSeconds)
    while True:
        pass

def destroy():
    GPIO.cleanup()                     # Release GPIO resource
```

```
if __name__ == '__main__':     # Program entrance
    print ('Program is starting...')
    setup()
    try:
        loop()
    except KeyboardInterrupt:  # Press ctrl-c to end the program.
        destroy()
```

---

3. Output:



---

4. Observations:

bouncetime (ms)	What happens when I press once?	What happens when I press quickly?
0	It flickers on and off randomly	Same thing but more flickering
50	Works normally	Sometimes it flickers randomly, but works most of the time.
100	Works perfectly	Occasional double presses
200	Works as expected	Doesn't recognize consecutive presses within 200ms of the first press
300	Works as expected	Doesn't recognize consecutive presses within 300ms of the first press

bouncetime (ms)	What happens when I press once?	What happens when I press quickly?
500	Works as expected	Doesn't recognize consecutive presses within 500ms of the first press
1000	Works as expected	Doesn't recognize consecutive presses within 1000ms of the first press

5. Analysis

Which debounce value worked best for your button? Why?

Based on our observations, **100ms** worked best for our button setup. This value provided the optimal balance between:

- **Reliability**: It eliminated the random flickering that occurred with lower debounce times (0ms and 50ms)
- **Responsiveness**: Unlike higher values (200ms, 300ms, 500ms, 1000ms), it still allowed for reasonably quick consecutive button presses
- **Practical usability**: While 200ms+ values worked perfectly for single presses, they became too inefficient for faster presses.

At 100ms, we observed "occasional double presses - which isn't something we can eliminate (without re-introducing random flicker if we decrease the bouncetime)" during rapid pressing, which indicates we're right at the threshold where the debouncing is effective but not overly conservative.

In your own words, what is "button bounce" and why do we need debouncing?

Button bounce occurs when we press/depress a mechanical button/switch. Instead of making a clean, instantaneous electrical connection, the metal contacts inside the button physically "bounce" against each other multiple times in rapid succession (typically within a few milliseconds).

It's essentially like dropping a ball - it doesn't just hit the ground once and stop - it bounces several times before settling.

Why we need debouncing:

Without debouncing, our microcontroller reads these rapid on/off signals as multiple separate button presses instead of just one. This causes unwanted behavior like:

- LEDs flickering randomly when it's pressed once
- Button press triggering a mechanism too many times in quick succession
- Unreliable input detection from the buttons

Software debouncing solves this by ignoring any additional button state changes for a specified time period after the first detected change, ensuring that one physical button press registers as exactly one logical button press in our code.