

# Assignment 1: PWM and Human Perception

**Name:** Genaro Salazar Ruiz

**Course:** WES 237

**Date:** January 19, 2026

**Github:** [WES-237-Assignment-1](#)

---

## 1. Introduction & Methodology

The objective of this assignment was to implement a functional Pulse Width Modulation (PWM) scheme on the PYNQ-Z2 board to control an RGB LED. My approach followed a top-down design methodology:

1. **Hardware Abstraction:** Writing low-level C++ code for the MicroBlaze to handle GPIO toggling.
2. **Signal Logic:** Implementing a Python-based PWM function to handle duty cycles and frequencies.
3. **User Interface:** Integrating asyncio to handle concurrent tasks: background blinking and real-time button polling.

## 2. Technical Implementation

### 2.1 MicroBlaze C++ (Low-Level)

To ensure the GPIO pins on PMODB were controlled with minimal latency, I utilized the %%microblaze processor. I implemented:

- **int write\_gpio(unsigned int pin, unsigned int val):** Opens the target pin and sets the direction to GPIO\_OUT for high-speed switching.
- **int reset\_all\_pins():** Uses a for loop to drive all 8 PMOD pins to 0V (Off) to ensure a known hardware state

### 2.2 Python PWM & Asyncio (High-Level)

The run\_pwm function handles duty cycle emulation. It includes logic for corner cases:

- **0% Duty Cycle:** Force pin LOW and sleep for the duration.
- **100% Duty Cycle:** Force pin HIGH and sleep for the duration.
- **Standard PWM:** Calculates on\_time and off\_time based on the frequency and period to drive the LED.

For the final task, asyncio managed two concurrent coroutines: flash\_leds() for the 1-second blink rhythm and get\_btns() for non-blocking button polling.

## 3. Results and Analysis

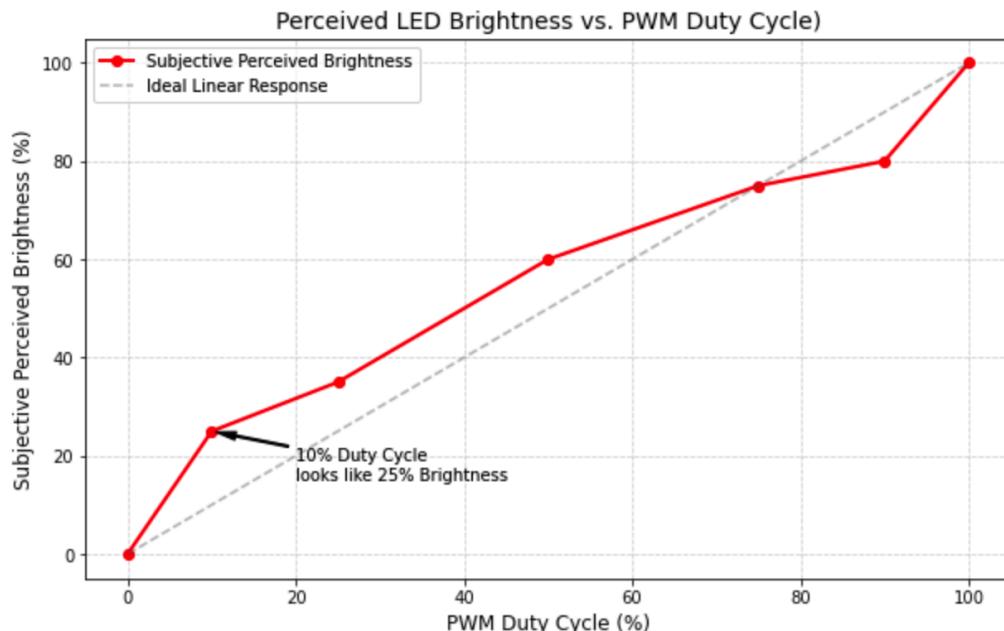
### 3.1 Optimal Frequency Selection

Through testing, I identified **40Hz** as the optimal frequency for this system.

- **Observation:** At 30Hz, a visible flicker was still present for me.
- **Result:** At 40Hz, the "flicker-fusion" threshold was met, where the human eye perceives the pulses as a steady analog voltage.

### 3.2 Brightness vs. Duty Cycle (Weber-Fechner Law)

I mapped my observed brightness to the duty cycle using the following test points: [0, 10, 25, 50, 75, 90, 100].



#### Explanation of Data:

As shown in the plot, the relationship is non-linear. Even at a low 10% duty cycle, the LED appeared to be at 25% brightness. This aligns with the Weber-Fechner Law, which describes how human eyes perceive light on a logarithmic scale. We are significantly more sensitive to light changes at low intensities, which is why the curve bows significantly above the linear line.

## 4. Troubleshooting & Difficulties

- **Asynchronous Task Cleanup:**
  - **Problem:** Encountered a Task That was destroyed, but it is still pending error when exiting the script.

- **Root Cause:** This occurred because the asyncio loop was terminated while the flash\_leds coroutine was suspended during an await asyncio.sleep(1) call.
  - **Solution:** I implemented a more robust exit strategy by ensuring the event loop handled pending tasks before closure and utilized a finally block to execute reset\_all\_pins(), ensuring the hardware was left in a safe, known state.
  - **GPIO State Contamination (Color Mixing):**
    - **Problem:** During color transitions (e.g., switching from Red to Green), the LED would occasionally display unintended secondary colors, such as purple or yellow.
    - **Root Cause:** The system was failing to clear the previous GPIO state before activating the new color channel. This caused multiple pins to remain high simultaneously, resulting in additive color mixing.
    - **Solution:** I updated the logic to include a pull-down sequence. By calling reset\_all\_pins() immediately before writing to the target GPIO, I ensured that only the intended color channel was active at any given time.
  - **Timing Jitter:**
    - **Problem:** Initially, the software-timed PWM exhibited slight inconsistencies and visual lag.
    - **Solution:** By selecting a moderate frequency of 40Hz, I balanced the need for a smooth visual experience (surpassing the human flicker-fusion threshold) with the limitations of Python's time.sleep precision. This reduced the impact of operating system jitter on the duty cycle accuracy.
- 

## 5. Video Demonstration Link

Link:  [IMG\\_1271.mov](#)

The video demonstrates: BTN0 (Blue), BTN1 (Green), BTN2 (Red), and BTN3 (Exit).

---

## 6. Appendix: Jupyter Notebook

```
In [1]: from pynq.overlay.base import BaseOverlay
import time

from datetime import datetime
base = BaseOverlay("base.bit")
bt�s = base.bt�s_gpio
```

```
In [2]: %%microblaze base.PMODB

#include "gpio.h"

// write function, sed by the Python PWM loop
int write_gpio(unsigned int pin, unsigned int val){
    gpio pin_out = gpio_open(pin);
    gpio_set_direction(pin_out, GPIO_OUT);
    gpio_write(pin_out, val);
    return 0;
}

//reset all pins on the PMOD to 0V (Off)
int reset_all_pins() {
    for(int i = 0; i < 8; i++) {
        gpio p = gpio_open(i);
        gpio_set_direction(p, GPIO_OUT);
        gpio_write(p, 0);
    }
    return 0;
}
```

```
In [3]: def run_pwm(pin_num, frequency, duty_cycle, duration):
    # corner cases 0% and 100%
    if duty_cycle <= 0:
        write_gpio(pin_num, 0)
        time.sleep(duration)
        return
    if duty_cycle >= 100:
        write_gpio(pin_num, 1)
        time.sleep(duration)
        return

    period = 1.0 / frequency
    on_time = period * (duty_cycle / 100.0)
    off_time = period - on_time

    start_time = time.time()
    while (time.time() - start_time) < duration:
        write_gpio(pin_num, 1)
        time.sleep(on_time)
        write_gpio(pin_num, 0)
        time.sleep(off_time)

    # mapping
BLUE = 1
GREEN = 2
RED = 3
GND = 4
```

In [4]:

```
# find flicker-fusion threshold
#30Hz (will flicker), 60Hz (better), 100Hz (smooth)

optimal_freq = 40
test_cycles = [0, 10, 25, 50, 75, 90, 100]

for dc in test_cycles:
    print(f"Testing {dc}% Duty Cycle...")
    run_pwm(GREEN, optimal_freq, dc, 3)

    write_gpio(GREEN, 0)
    time.sleep(1)
    #first go
    #10 looked 1/4 to me
    #50 looked half to me
    #100 was definitely the brightest

import matplotlib.pyplot as plt

duty_cycles = [0, 10, 25, 50, 75, 90, 100]
perceived_brightness = [0, 25, 35, 60, 75, 80, 100] #my full observations

plt.figure(figsize=(10, 6))

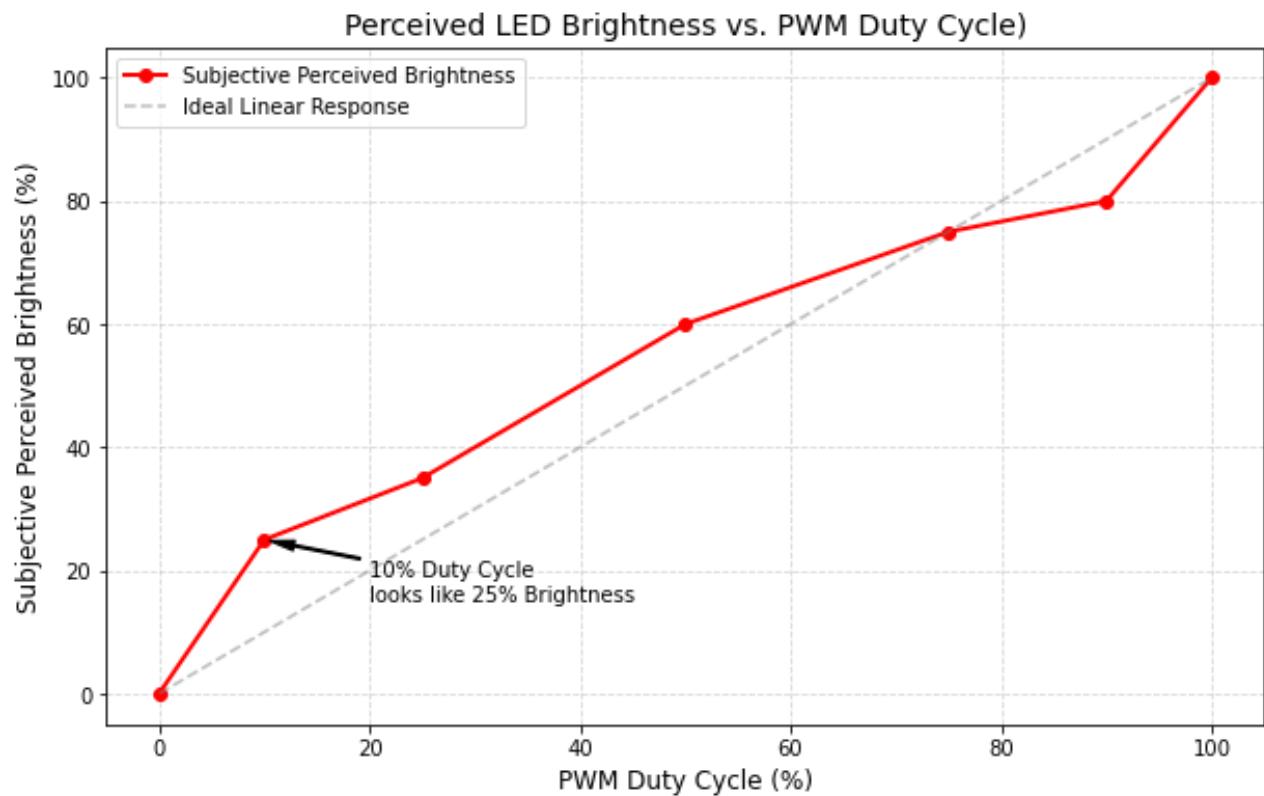
plt.plot(duty_cycles, perceived_brightness, 'ro-', linewidth=2, label='Subjective Perceived Brightness (%)')
plt.plot([0, 100], [0, 100], color='gray', linestyle='--', alpha=0.5, label='Ideal Brightness')

plt.title('Perceived LED Brightness vs. PWM Duty Cycle', fontsize=14)
plt.xlabel('PWM Duty Cycle (%)', fontsize=12)
plt.ylabel('Subjective Perceived Brightness (%)', fontsize=12)
plt.grid(True, which='both', linestyle='--', alpha=0.5)
plt.legend()

plt.annotate('10% Duty Cycle\nlooks like 25% Brightness', xy=(10, 25), xytext=(25, 60),
             arrowprops=dict(facecolor='black', shrink=0.05, width=1, headwidth=5))

plt.show()
```

```
Testing 0% Duty Cycle...
Testing 10% Duty Cycle...
Testing 25% Duty Cycle...
Testing 50% Duty Cycle...
Testing 75% Duty Cycle...
Testing 90% Duty Cycle...
Testing 100% Duty Cycle...
```



In [5]:

```
import asyncio

freq = 10      # my optimal
dim_dc = 25    # weak/dim duty cycle for the 25% brightness task
BLUE = 1
GREEN = 2
RED = 3
cond = True

current_color = RED

reset_all_pins()

async def flash_leds():
    global cond, current_color
    print("Blinking Started. BTN0:R, BTN1:G, BTN2:B, BTN3:Exit")
    while cond:

        # 1 Second ON
        write_gpio(current_color, 1)
        await asyncio.sleep(1)

        # 1 Second OFF
        reset_all_pins()
        await asyncio.sleep(1)

async def get_btns(_loop):
    global cond, current_color
    while cond:
        await asyncio.sleep(0.01)
        if btns[0].read():  #click blue
            current_color = BLUE
        if btns[1].read():  #click green
            current_color = GREEN
        if btns[2].read():  #click red
            current_color = RED
        if btns[3].read():  #click to stop
            print("stopping")
            _loop.stop()
            cond = False

loop = asyncio.new_event_loop()
loop.create_task(flash_leds())
loop.create_task(get_btns(loop))
loop.run_forever()
loop.close()

reset_all_pins()
print("Done.")
```

```
Blinking Started. BTN0:R, BTN1:G, BTN2:B, BTN3:Exit  
stopping  
Done.
```

In [ ]:

In [ ]: