Assignment 1

The goals of this assignment are

1. Familiarize yourselves with PYNQ board environment including
    - Jupyter notebooks
    - Inline C++ programming of the microblaze
    - Python3 programming language
    - PMOD peripherals as GPIO pins
2. Understand and build a functional Pulse Width Modulation (PWM) scheme to light an LED peripheral.
3. Discover the optimal PWM parameters relating to human perception.

Pulse Width Modulation

Without DAC, the GPIO at the output mode can only set the voltage at the specific pin with 2 states (LOW and HIGH). However, it is possible to 'emulate' an analog voltage with digital pins using a square wave. The electronics connected to the GPIO pin average the digital signal in time resulting in an analog voltage between (0-3.3V). One can modulate the width of square wave resulting in different averaged voltages (see figure below). This technique is called Pulse Width Modulation (PWM). With sufficient high PWM frequency (so that human will not be able to visually perceive the flashing), the brightness of an LED can be finely controlled by the duty cycle (the width of the square wave in percentage).

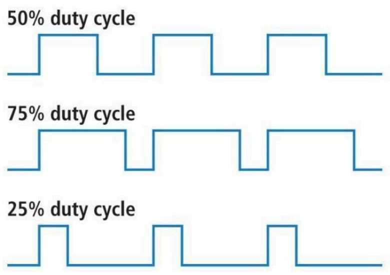


Figure 1: Different duty cycles of PWM signals

In this task, you'll be using the following LED from your sensor kit.

Figure 2: RGB LED from the sensor kit

Each RGB pin should be connected to a PMOD GPIO pin (0-7). The '-' pin is connected to ground.

1. Use the gpio.ipynb from lab as a starting point. Add a C++ function to reset all the GPIO pins on the chosen PMOD.
    - This part was pretty straightforward. I reset all the pins by writing all 8 to 0 using a for loop.

```
void clear_gpios(){
  for(int i = 0; i < 8; ++i){
    write_gpio(i,0);
  }
}
```

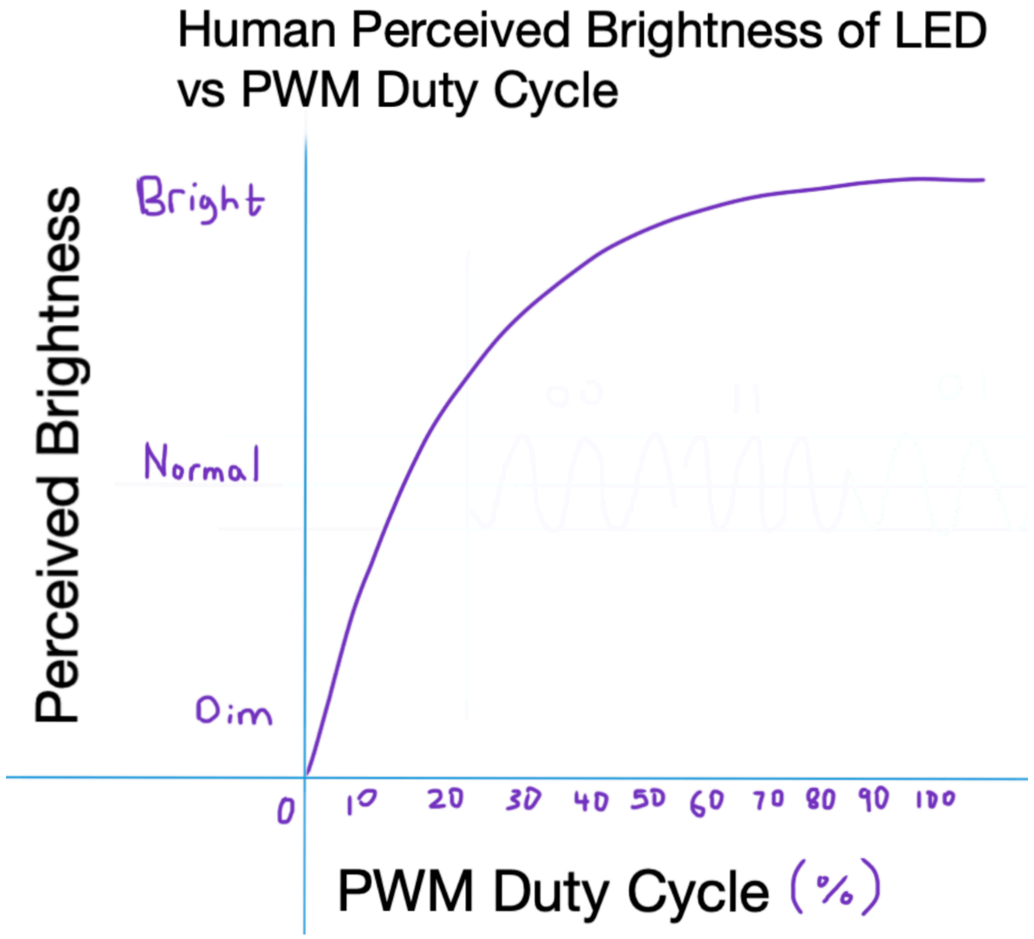
2. Write a Python3 cell that emulates a PWM (for a chosen frequency and duty cycle) on one of the PMOD GPIO pins. For example, PMODB PIN3 needs to turn on for duty_cycle percent of the square wave frequency and off for the rest of the square wave. It may not be possible to achieve exactly 0% or 100% so be sure to add necessary code for those corner cases.
    - Initially, I planned to use time.sleep to implement the PWM, by writing a 1 or 0 and then sleeping for a duration depending on the duty cycle and frequency. Although I did see that we need to use asyncio later, it is easier to debug using time.sleep and isolating the generate pwm function.
    - f = 1/T, so I converted frequency to find the period in seconds, since asyncio.sleep is in units of seconds

```
def generate_pwm_simple(freq_hz, duty):
  period = 1 / freq_hz
  on_time = period * (duty / 100)
  off_time = period * (1 - duty / 100)

  if duty == 0:
    write_gpio(3, 0)
    time.sleep(period)
  elif duty == 100:
    write_gpio(3, 1)
    time.sleep(period)
  else:
    write_gpio(3, 1)
    time.sleep(on_time)
    write_gpio(3, 0)
    time.sleep(off_time)
```

3. Find the optimal PWM frequency, such that the physical flashing phenonmenon will not be perceived visually;
    - I started at 1Hz just to see what the light looks like. In order to test, I implemented an asyncio loop so I could stop the flashing with the buttons and make testing the frequency easier. However, that led to me having to change the function from time.sleep to await asyncio.sleep.
    - Below is the code I used to test the frequency to prevent the human eye from seeing the flashing.
    - The final frequency for the flashing to not be perceivable is around 60Hz

```
import asyncio

cond = True

async def generate_pwm_simple(freq_hz, duty):
    period = 1 / freq_hz
    on_time = period * (duty / 100)
    off_time = period * (1 - duty / 100)

    if duty == 0:
        write_gpio(3, 0)
        await asyncio.sleep(period)
    elif duty == 100:
        write_gpio(3, 1)
        await asyncio.sleep(period)
    else:
        write_gpio(3, 1)
        await asyncio.sleep(on_time)
        write_gpio(3, 0)
        await asyncio.sleep(off_time)

async def flash_leds():
    global cond
    while cond:
        await generate_pwm_simple(1, 50)

async def get_btns():
    global cond
    while cond:
        await asyncio.sleep(0.01)
        if btns.read() != 0:
            cond = False

async def main():
    clear_gpios()
    await asyncio.gather(
        flash_leds(),
        get_btns()
    )
    clear_gpios()
    print("Done.")

asyncio.run(main())
```

4. Achieve the visually perceived 100%, 75%, 50% and 25% of full LED brightness by adjusting the duty cycle;
5. Varying the duty cycles and approximate the corresponding LED brightness (in the unit of %). To the end, plot and explain the approximate relationship of % brightness versus duty cycle. While working on this task, feel free to check out the Weber-Fechner law. For light perception, this law describe the observation that the our eyes perceive light in a non-linear way.



-
- As you increase the duty cycle, the increase in brightness at the same interval of increase becomes less and less noticeable.
- This is because although the duty cycle is linearly increasing the brightness of the light, the eye perceives light in a non-linear way. We instead respond to changes in light in ratios instead of just +10%, which means that a change between 10-20 will be a lot brighter than a change between 80 and 90.

6. After you have experimented with various PWM frequencies and duty cycles, add the following functionalities **for a fixed duty cycle (i.e. 25%)**. You'll want to use asyncio for this part.
    - Start the code and blink the LED's red channel in intervals of 1 second (i.e. 1 second on, 1 second off)
    - When buttons 0, 1, or 2 are pushed, the LED will change color from Red, to Green, to Blue.
    - When button 3 is pushed, the LED will stop blinking.
    - Your video should demonstrate how each button change to each of the (4) colors.

- Based off of the specs, I need a couple of functions:
    - Blink the LED in 1s intervals
        - This function should also be able to change the color of the LED
    - The PWM function to generate the pulse that drives the LED
    - Button read functionality
- To keep these loops going, we will use the cond var similar to the asyncio example from lab1.
- The led variable controls which GPIO to write the signal to, which then translates to which color LED we want on.
- Each button clears the gpio and then changes the led variable to change the gpio being written to which changes the led color.

- While writing the code, I ran into the issue of the prints working but the gpio was stuck green (default). This kept happening despite the earlier test code working. I later realized that it was because I reimported the base overlay at the beginning of the code, which effectively wipes out the microblaze C code, leading to nothing happening because the function definitions were not there.
- Another issue I ran into was the board would randomly get stuck. This was due to how fast we were writing the gpio pins, essentially bit banging the line, which resulted in undesirable behavior. Because of this, I made the frequency as slow as possible to where the led didn't blink to reduce the amount of writes to the pin.
- To debug, I also added prints to each of the buttons to see if my input was going through and if I was changing the variables correctly. However I deleted the debug prints in my final code.

```
import asyncio

"""
cond: runs all of the functions
led: chooses led color. default color is red.
    R=3, G=2, B=1
"""

cond = True
led = 3

async def generate_pwm(freq_hz, duty, gpio):
    # period in s
    period = 1/freq_hz
    on_time = period * (duty/100)
    off_time = period * (1 - (duty/100))

    if duty == 0:
        write_gpio(gpio, 0)
        await asyncio.sleep(period)
    elif duty == 100:
        write_gpio(gpio, 1)
        await asyncio.sleep(period)
    else:
        write_gpio(gpio, 1)
        await asyncio.sleep(on_time)
        write_gpio(gpio, 0)
        await asyncio.sleep(off_time)

async def flash_leds():
    global cond, led
    while cond:
        start = time.time()
        while cond and time.time() - start < 1.0:
            await generate_pwm(55, 50, led)

        start = time.time()
        while cond and time.time() - start < 1.0:
            await generate_pwm(10, 0, led)

async def get_btns():
    global cond, led
    while cond:
        await asyncio.sleep(0.01)
        # turn off led to prepare for transition if any button is pressed
        if btns.read() != 0:
            clear_gpios()
        if btns[0].read() != 0:
            # R
            led = 3
        if btns[1].read() != 0:
            # G
            led = 2
        if btns[2].read() != 0:
            # B
            led = 1
        if btns[3].read() != 0:
            clear_gpios()
            cond = False

async def main():
    clear_gpios()
    print("Running program")
    try:
        await asyncio.gather(
            flash_leds(),
            get_btns()
        )
    finally:
        clear_gpios()
        print("Done.")

asyncio.run(main())
```

Video Link: https://youtu.be/jHgWv3LpY3c
GitHub Repo: https://github.com/cxu4426/WES237A-hw

The following references may be helpful.

1. https://deepbluembedded.com/pwm-pulse-width-modulation-tutorial/
2. https://insights.acuitybrands.com/eldoled-blog/article-why-you-need-dimming-curves

Deliverables

Each student must submit the following **individually**

1. A PDF report detailing your work flow and relative Jupyter notebook cells relating to your progress.
    - Your report should detail your work flow throughout the assignment. Be sure to discuss **any** difficulties or troubles you encountered and your troubleshooting procedure. Also, detail your thought process which led to the design and implementation of the code. For example, describe your top-down design methodology (i.e. how did you split the large task into smaller, more incremental jobs? How were you able to test each of these smaller parts?)
    - Please also include a plot of perceived brightness (relative to 100% duty cycle) vs. duty cycle and an explanation of the data. Be sure to title your plot and label your axes.
2. Your complete Jupyter notebook, downloaded as a PDF **attached at the very end of your report**
    - You can do this by selecting 'File -> Print Preview' then printing to PDF from the browser.
    - Use a PDF stitching tool like pdfjoiner to join your Report and Jupyter Notebook into a single PDF file
3. Video demonstration of your code working on the PYNQ board. Please limit videos to 60 seconds and upload them to a video sharing site and include the link on your PDF report.
4.

Each team should submit the following **one per team**

1. All relevant code (.ipynb, .py, .cpp, .c, etc files) pushed to your team's git repo.

Assignment 1: PWM

Setup

In [3]:
```
from pynq.overlays.base import BaseOverlay
import time
from datetime import datetime
base = BaseOverlay("base.bit")
btns = base.btns_gpio
```

In [4]:
```
%%microblaze base.PMODB

#include "gpio.h"
#include "pyprintf.h"

//Function to turn on/off a selected pin of PMODB
void write_gpio(unsigned int pin, unsigned int val){
    if (val > 1){
        pyprintf("pin value must be 0 or 1");
    }
    gpio pin_out = gpio_open(pin);
    gpio_set_direction(pin_out, GPIO_OUT);
    gpio_write(pin_out, val);
}

//Function to read the value of a selected pin of PMODB
unsigned int read_gpio(unsigned int pin){
    gpio pin_in = gpio_open(pin);
    gpio_set_direction(pin_in, GPIO_IN);
    return gpio_read(pin_in);
}

void clear_gpios(){
    for(int i = 0; i < 8; ++i){
        write_gpio(i,0);
    }
}
```

Test code

In [3]:
```
import asyncio

cond = True

async def generate_pwm_simple(freq_hz, duty):
    period = 1 / freq_hz
    on_time = period * (duty / 100)
    off_time = period * (1 - duty / 100)

    if duty == 0:
        write_gpio(3, 0)
        await asyncio.sleep(period)
    elif duty == 100:
        write_gpio(3, 1)
        await asyncio.sleep(period)
    else:
        write_gpio(3, 1)
        await asyncio.sleep(on_time)
        write_gpio(3, 0)
        await asyncio.sleep(off_time)

async def flash_leds():
    global cond
    while cond:
        await generate_pwm_simple(0.5, 50)

async def get_btns():
    global cond
    while cond:
        await asyncio.sleep(0.01)
        if btns.read() != 0:
            clear_gpios()
            cond = False

async def main():
    clear_gpios()
    try:
        await asyncio.gather(
            flash_leds(),
            get_btns()
        )
    finally:
        clear_gpios()
        print("Done.")

asyncio.run(main())
```

```
Done.
```

RGB code

In [5]:

```
import asyncio

"""
cond: runs all of the functions
led: chooses led color. default color is red.
    R=3, G=2, B=1
"""

cond = True
led = 3

async def generate_pwm(freq_hz, duty, gpio):
    # period in s
    period = 1/freq_hz
    on_time = period * (duty/100)
    off_time = period * (1 - (duty/100))

    if duty == 0:
        write_gpio(gpio, 0)
        await asyncio.sleep(period)
    elif duty == 100:
        write_gpio(gpio, 1)
        await asyncio.sleep(period)
    else:
        write_gpio(gpio, 1)
        await asyncio.sleep(on_time)
        write_gpio(gpio, 0)
        await asyncio.sleep(off_time)

async def flash_leds():
    global cond, led
    while cond:
        start = time.time()
        while cond and time.time() - start < 1.0:
            await generate_pwm(55, 50, led)

        start = time.time()
        while cond and time.time() - start < 1.0:
            await generate_pwm(10, 0, led)

async def get_btns():
    global cond, led
    while cond:
        await asyncio.sleep(0.01)
        # turn off led to prepare for transition if any button is pressed
        if btns.read() != 0:
            clear_gpios()
        if btns[0].read() != 0:
            # R
            led = 3
        if btns[1].read() != 0:
            # G
            led = 2
        if btns[2].read() != 0:
            # B
            led = 1
        if btns[3].read() != 0:
            clear_gpios()
            cond = False

async def main():
    clear_gpios()
    print("Running program")
    try:
        await asyncio.gather(
            flash_leds(),
            get_btns()
        )
    finally:
        clear_gpios()
        print("Done.")

asyncio.run(main())
```

```
Running program
Done.
```