WES 237A – Assignment 1, Pulse Width Modulation

Ricardo Lizarraga, rlizarraga0@gmail.com , *619.252.4157* | **PID: A69028483** | https://github.com/RiLizarraga

Demo videos files at: https://drive.google.com/drive/folders/1jYQ4IE-Q7_aljBgwKIEFN_wpze6EYsMB?usp=sharing
"Assigm1_solution4.mp4",
"Assigm1_solution5.mp4"
"Assigm1_solution6.mp4"

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).

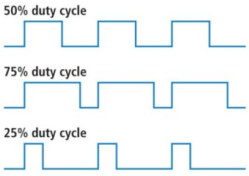
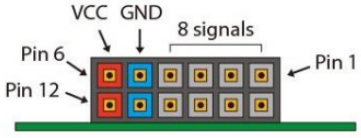


Figure 1: Different duty cycles of PWM signals

Figure 2: RGB LED from the sensor kit

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

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



The VCC and Ground pins can deliver up to 1A of current.

PMODB		RGB Led Module
Pin#1 GPIO 0		
Pin#2 **GPIO 1**	-	**B**
Pin#3 **GPIO 2**	-	**G**
Pin#4 **GPIO 3**	-	**R**
Pin#5 **GND**	-	(-)
Pin#6 **Vcc**	-	(+)

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.

Assignment 1

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.

In [1]:
```
from pynq.overlays.base import BaseOverlay
#import time
from time import sleep
from datetime import datetime
base = BaseOverlay("base.bit")
```

In [3]:
```
%%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);        //instance object
    gpio_set_direction(pin_in, GPIO_IN); //configure to be Input
    return gpio_read(pin_in);
}
//Function to reset all GPIOs of PMODB
void reset_gpio_PMODB_all(){
    pyprintf("Reset all GPIOs of PMODB");
    for (int pin = 0; pin < 8; pin++) {
        gpio pin_out = gpio_open(pin);
        gpio_set_direction(pin_out, GPIO_OUT);
        gpio_write(pin_out, 0);
    }
}
```

In [4]:
```
reset_gpio_PMODB_all()
```

Comments/ Observations

Implementation/ design:
Since lab1 we did a C coding for %%microblaze, it was just adding a new function “void reset_gpio_PMOD_all()”, so this function will be public the Python application being executed in Jupyter Notebook, so it can be called directly from Python code with the corresponding argument

Difficulties:
none

Testing tactic:
This function was tested by programming zero value to the 3 pins we will be using (2,3 &5), after the LED’s were lighted up, I exercised “void reset_gpio_PMOD_all()”, and visually noticed that the LED were OFF.

Conclusions:
Amazing potential to be able to work on C code & Python on the same application

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.

pynq.lib.pmod.pmod_pwm Module

class pynq.lib.pmod.pmod_pwm.Pmod_PWM(*mb_info*, *index*) [source]

Bases: object

This class uses the PWM of the IOP.

microblaze

Microblaze processor instance used by this module.

Type: Pmod

generate(*period*, *duty_cycle*) [source]

Generate pwm signal with desired period and percent duty cycle.

Parameters:
- **period** (*int*) – The period of the tone (us), between 1 and 65536.
- **duty_cycle** (*int*) – The duty cycle in percentage.

Returns:

Return type: None

stop() [source]

Stops PWM generation.

Returns:

Return type: None

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.

In [11]:

```
# Single PWM for any period and duty cycle 1 to 99%
from pynq.overlays.base import BaseOverlay
base = BaseOverlay("base.bit")
from pynq.lib import Pmod_PWM
import time
# globals
PIN_R = 3;PIN_G = 2;PIN_B = 1;PIN_0 = 0

def pmw_gen(period,duty):
    period = min(max(period,1),65535)
    duty = min(max(duty,1),99)
    pwm.generate(period,duty)

pwm = Pmod_PWM(base.PMODB,PIN_0)
pmw_gen(10,50)
time.sleep(20)
pwm.stop()
```

Comments/ Observations

Implementation/ design/ blocks:

My first step was to check PYNQ-Z2 documentation to see if there's any arrangements already implemented for PWM generation and then found "Pmod_PWM" library

Difficulties:

None

Testing tactic:

For testing I used my Digital Oscilloscope to inspect and measure the function generated on one of the pins, and to avoid any run-time error is important apply limit in this case I clipped any value outside of the expected band:

```
period = min(max(period,1),65535)
duty = min(max(duty,1),99)
```

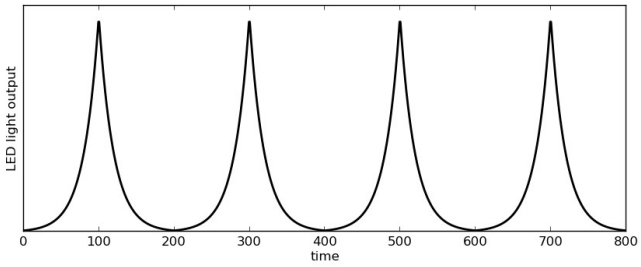
Conclusions:

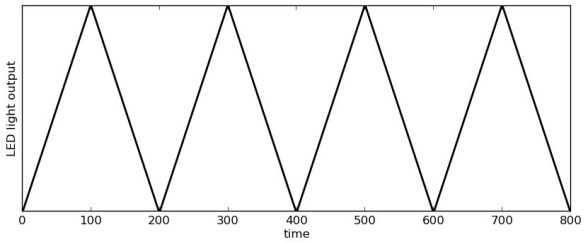
Pmod_PWM library make's very simple to implement PWM on any pin, but unfortunately only supports (1-65535) uSeconds, the problem is that if I want to do a period greater than 65.535 mSec, this will not support it

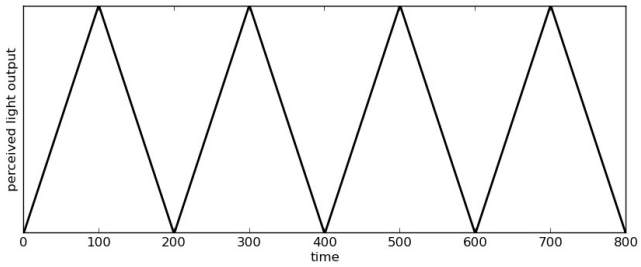
3. Find the optimal PWM frequency, such that the physical flashing phenomenon will not be perceived visually;

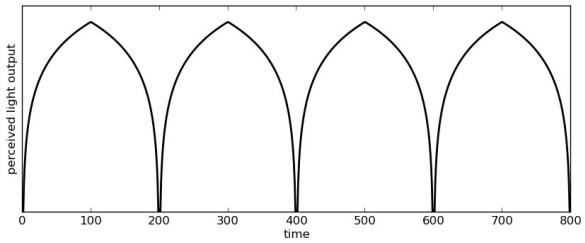
There are a lot of webpages describing the actual process of dimming a LED light using Pulse-with Modulation (PWM). The general idea is to switch the LED completely on and of at a high frequency (in the range of **kHz**) so that you eyes cannot see the actual on or off state anymore. By varying the so called duty cycle (the percentage of on-time within one cycle of the dimming signal), we can create the perception to our eyes, that the LED is less bright than full on but also not completely off – depending on the duty cycle. Clearly a duty cycle of 100% means the LED is completely on where as 0% mean the LED is off.

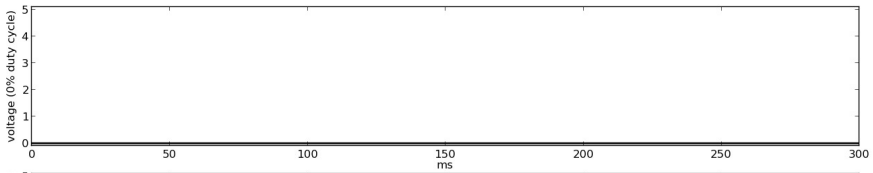
Now say you want the LED to dim from 0% to 100% and back down to 0% linearly – i.e .the increase in brightness should be linearly increasing (and then decreasing) with the time passed. To make this happen we can simply increase (and then decrease) the duty cycle linearly.

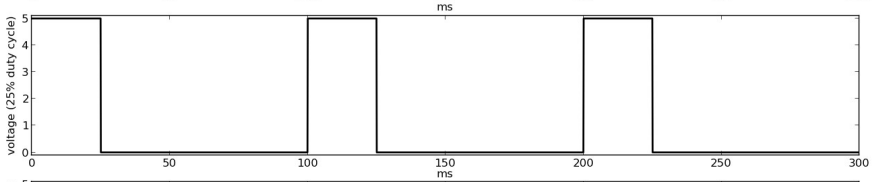


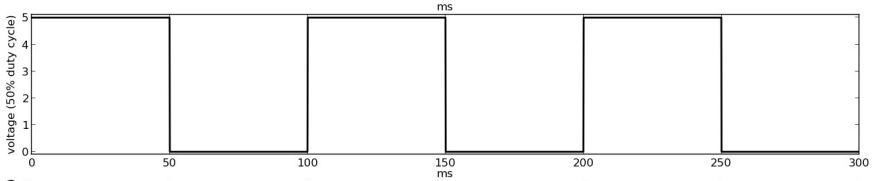


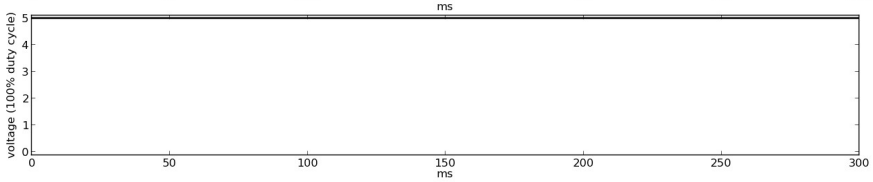












Now say you want the LED to dim from 0% to 100% and back down to 0% linearly – i.e .the increase in brightness should be linearly increasing (and then decreasing) with the time passed. To make this happen we can simply increase (and then decrease) the duty cycle linearly.
This will give us a linear increase/decrease in light output BUT this will not result in a perceived linear increase/decrease of the light. This is due to the Weber-Fechner law. For light perception, this law describes the observation that the our eyes perceive light in a logarithmic way. So this is approximately what we are observing:
The Weber-Fechner law thus implies, that if we have to increase the light output of the LED exponentially (and not linearly) to achieve the perception, that the light is increased linearly

3. Find the optimal PWM frequency, such that the physical flashing phenonmenon will not be perceived visually;

In [1]:

```python
from pynq.overlays.base import BaseOverlay
import time
from datetime import datetime
base = BaseOverlay("base.bit")
```

In [2]:

```
%%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);        //instance object
    gpio_set_direction(pin_in, GPIO_IN); //configure to be Input
    return gpio_read(pin_in);
}
//Function to reset all GPIOs of PMODB
void reset_gpio_PMODB_all(){
    pyprintf("Reset all GPIOs of PMODB");
    for (int pin = 0; pin < 8; pin++) {
        gpio pin_out = gpio_open(pin);
        gpio_set_direction(pin_out, GPIO_OUT);
        gpio_write(pin_out, 0);
    }
}
```

In [*]:

```
import asyncio
threads_flag = True    # "cond" will keep threads live or kill them all
# Initial states
pmwR_flag = False
pmwG_flag = False
pmwB_flag = False
delay = 1;period = 1;duty = 25
PIN_R = 3;PIN_G = 2;PIN_B = 1;PIN_0 = 0
period=10;duty=90;on=0.5;off=0.5;blinks=0;blink_stop=100000
def user_input():
    globals()['write_gpio']=write_gpio
    global threads_flag, blinks, blink_stop, on, off, pmwR_flag, pmwG_flag, pmwB_flag, pmw_flag, period, duty, PIN_R, PI
    ret = read_gpio(PIN_R);
    ret = read_gpio(PIN_G)
    ret = read_gpio(PIN_B)
    write_gpio(PIN_R, 0);write_gpio(PIN_G, 0);write_gpio(PIN_B, 0); #reset_gpio_PMODB_all()
    default = 100;val1 = input("  Enter Frequency(Hz): {default}") or default
    freq = min(max(int(val1),1),65535)
    period = 1/freq# T=1/f
    print("period = "+ str(period))
    default = 25;val2 =  input("Enter Duty percentage: {default}") or default
    duty = min(max(int(val2),1),99)
    print("duty = "+ str(duty))
    on = (duty/100)*period
    off = period-on
    print("on="+str(on)+", off="+str(off))
async def RGB_leds():
    globals()['write_gpio']=write_gpio
    global threads_flag, blinks, blink_stop, on, off, pmwR_flag, pmwG_flag, pmwB_flag, pmw_flag, period, duty, PIN_R, PI
    while threads_flag:
        if blinks >= blink_stop:
            pmwR_flag = pmwG_flag = pmwB_flag = False
            blinks = 0
            write_gpio(PIN_R, 0);write_gpio(PIN_G, 0);write_gpio(PIN_B, 0); #reset_gpio_PMODB_all()
        await asyncio.sleep(off)
        if (pmwR_flag == True):
            write_gpio(PIN_R, 1)
            await asyncio.sleep(on)
            ret = read_gpio(PIN_R)
            write_gpio(PIN_R, 0)
            blinks += 1
        elif (pmwG_flag == True):
            write_gpio(PIN_G, 1)
            await asyncio.sleep(on)
            ret = read_gpio(PIN_R)
            write_gpio(PIN_G, 0)
            blinks += 1
        elif (pmwB_flag == True):
            write_gpio(PIN_B, 1)
            await asyncio.sleep(on)
            ret = read_gpio(PIN_R)
            write_gpio(PIN_B, 0)
            blinks += 1
        else:
            await asyncio.sleep(1)
            print('.', end='')
async def get_btns(_loop):
    btns = base.btns_gpio
    globals()['write_gpio']=write_gpio
    global threads_flag, blinks, blink_stop, delay, pmwR_flag, pmwG_flag, pmwB_flag, pmw_flag, period, duty, PIN_R, PIN_
    while threads_flag:
        await asyncio.sleep(0.1)
        if btns.read() != 0:
            if (btns[0].read()==1):
                await asyncio.sleep(0.3)
                print("button 0 pressed, Enter params for RED")
                pmwR_flag = True;pmwG_flag = False;pmwB_flag = False
                user_input()
            elif (btns[1].read()==1):
                await asyncio.sleep(0.3)
                print("button 1 pressed, Enter params for GREEN")
                pmwG_flag = True;pmwR_flag = False;pmwB_flag = False
                user_input()
            elif (btns[2].read()==1):
                await asyncio.sleep(0.3)
                print("button 2 pressed, Enter params for BLUE")
                pmwB_flag = True;pmwR_flag = False;pmwG_flag = False
                user_input()
            else:
                await asyncio.sleep(0.3)
                print("button 3 pressed, Exit program")
                threads_flag = False
                _loop.stop()
### main sequence ##
write_gpio(PIN_R, 0);write_gpio(PIN_G, 0);write_gpio(PIN_B, 0);
time.sleep(0.2)
loop = asyncio.new_event_loop();  # Instance event_loop object
loop.create_task(RGB_leds());   #
loop.create_task(get_btns(loop)); # take user input buttons
print("Press buttons 0-3\n   0 - RED,\n   1 - GREEN\n   2 - BLUE\n   3 - Exit program")
loop.run_forever()
loop.close()
print("Done.")
```

```
Press buttons 0-3
   0 - RED,
   1 - GREEN
   2 - BLUE
   3 - Exit program
..button 0 pressed, Enter params for RED
  Enter Frequency(Hz): {default}10
period = 0.1
Enter Duty percentage: {default}
duty = 25
on=0.025, off=0.07500000000000001
.button 0 pressed, Enter params for RED
  Enter Frequency(Hz): {default}30
period = 0.03333333333333333
Enter Duty percentage: {default}
duty = 25
on=0.008333333333333333, off=0.025
button 0 pressed, Enter params for RED
  Enter Frequency(Hz): {default}50
period = 0.02
Enter Duty percentage: {default}
duty = 25
on=0.005, off=0.015
button 0 pressed, Enter params for RED
  Enter Frequency(Hz): {default}60
period = 0.016666666666666666
Enter Duty percentage: {default}
duty = 25
on=0.004166666666666667, off=0.0125
button 0 pressed, Enter params for RED
  Enter Frequency(Hz): {default}90
period = 0.011111111111111112
Enter Duty percentage: {default}
duty = 25
on=0.002777777777777778, off=0.008333333333333333
button 0 pressed, Enter params for RED
  Enter Frequency(Hz): {default}110
period = 0.00909090909090909
Enter Duty percentage: {default}
duty = 25
on=0.0022727272727272726, off=0.006818181818181818
```

Comments/ Observations

Implementation/ design/ blocks:

To full fill this application on #3, the library “Pmod_PWM” that I used on #2 will not work here, because we need very low frequencies (bigger periods), so I had to create Implement Multi-threading capabilities (**async**):

Thread1: User_input(): Arbitrary parameters values (period & Duty cycle %)

Thread2: get_btns(): User is allowed to change parameter values per LED color by choosing the corresponding button.

Thread3: RGB_leds(): Infinitum task loop to execute based on the parameters’ states

Difficulties:

I had one problem: After blinking cycle started, after a while, the other 2 LEDs starts to show small current leaks, so I tried to reprogram zero before and after LED transition, and the behavior was fixed by reading the input pin then setting to zero, see code:

```
ret = read_gpio(PIN_R)
write_gpio(PIN_G, 0)
```

I checked the code several time, and looks good, it could be in one of the abstraction levels of PYNQ, or maybe Jupyter Python behavior, I will keep in an eye on that

Testing tactic:

For testing, the user is prompted to enter parameters (PERIOD & DUTY CYCLE) per LED color, then visually inspect the behavior.

Conclusions:

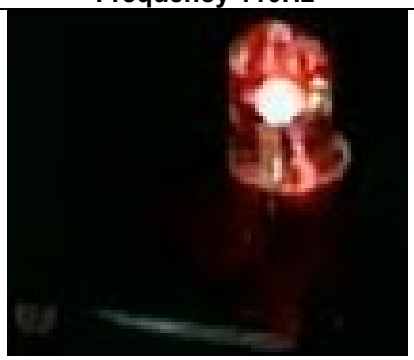
To measure LED perceived brightness intensity is very subjective to each individual person and depends on the environment lighting at the moment of the test.

In this test in particular With a Duty cycle of 25% and Frequency of 110 Hz, my eye could not detect the fluctuation anymore.

See DEMO video “Assigm1_solution3.mp4”

https://drive.google.com/drive/folders/1jYQ4IE-Q7_aIjBgwKIEFN_wpze6EYsMB?usp=sharing

4. Achieve the visually perceived 100%, 75%, 50% and 25% of full LED brightness by adjusting the duty cycle;

Duty cycle	Frequency 110Hz
25%	
50%	
75%	
100%	

With a Frequency of 110 Hz, Changed dynamically Duty Cycle 25, 50, 75 & 100% {in the picture is hard to distinguish but as duty cycle increases, the light intensity increaded too..

```
4. Achieve the visually perceived 100%, 75%, 50% and 25% of full LED brightness by adjusting the duty cycle
```

In [1]:

```
from pynq.overlays.base import BaseOverlay
import time
from datetime import datetime
base = BaseOverlay("base.bit")
```

In [2]:

```
%%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);        //instance object
    gpio_set_direction(pin_in, GPIO_IN); //configure to be Input
    return gpio_read(pin_in);
}
//Function to reset all GPIOs of PMODB
void reset_gpio_PMODB_all(){
    pyprintf("Reset all GPIOs of PMODB");
    for (int pin = 0; pin < 8; pin++) {
        gpio pin_out = gpio_open(pin);
        gpio_set_direction(pin_out, GPIO_OUT);
        gpio_write(pin_out, 0);
    }
}
```

In [ ]:

```
import asyncio
threads_flag = True    # "cond" will keep threads live or kill them all
# Initial states
pmwR_flag = False
pmwG_flag = False
pmwB_flag = False
delay = 1;period = 1;duty = 25
PIN_R = 3;PIN_G = 2;PIN_B = 1;PIN_0 = 0
period=110;duty=90;on=0.5;off=0.5;blinks=0;blink_stop=100000
def user_input():
    globals()['write_gpio']=write_gpio
    global threads_flag, blinks, blink_stop, on, off, pmwR_flag, pmwG_flag, pmwB_flag, pmw_flag, period, duty, PIN_R, PI
    ret = read_gpio(PIN_R);
    ret = read_gpio(PIN_G)
    ret = read_gpio(PIN_B)
    write_gpio(PIN_R, 0);write_gpio(PIN_G, 0);write_gpio(PIN_B, 0); #reset_gpio_PMODB_all()
    default = 110;val1 = input("  Enter Frequency(Hz): {default}") or default
    freq = min(max(int(val1),1),65535)
    period = 1/freq# T=1/f
    print("period = "+ str(period))
    default = 25;val2 =  input("Enter Duty percentage: {default}") or default
    duty = min(max(int(val2),1),99)
    print("duty = "+ str(duty))
    on = (duty/100)*period
    off = period-on
    print("on="+str(on)+", off="+str(off))
async def RGB_leds():
    globals()['write_gpio']=write_gpio
    global threads_flag, blinks, blink_stop, on, off, pmwR_flag, pmwG_flag, pmwB_flag, pmw_flag, period, duty, PIN_R, PI
    while threads_flag:
        if blinks >= blink_stop:
            pmwR_flag = pmwG_flag = pmwB_flag = False
            blinks = 0
            write_gpio(PIN_R, 0);write_gpio(PIN_G, 0);write_gpio(PIN_B, 0); #reset_gpio_PMODB_all()
        await asyncio.sleep(off)
        if (pmwR_flag == True):
            write_gpio(PIN_R, 1)
            await asyncio.sleep(on)
            ret = read_gpio(PIN_R)
            write_gpio(PIN_R, 0)
            blinks += 1
        elif (pmwG_flag == True):
            write_gpio(PIN_G, 1)
            await asyncio.sleep(on)
            ret = read_gpio(PIN_R)
            write_gpio(PIN_G, 0)
            blinks += 1
        elif (pmwB_flag == True):
            write_gpio(PIN_B, 1)
            await asyncio.sleep(on)
            ret = read_gpio(PIN_R)
            write_gpio(PIN_B, 0)
            blinks += 1
        else:
            await asyncio.sleep(1)
            print('.', end='')
async def get_btns(_loop):
    btns = base.btns_gpio
    globals()['write_gpio']=write_gpio
    global threads_flag, blinks, blink_stop, delay, pmwR_flag, pmwG_flag, pmwB_flag, pmw_flag, period, duty, PIN_R, PIN_
    while threads_flag:
        await asyncio.sleep(0.1)
        if btns.read() != 0:
            if (btns[0].read()==1):
                await asyncio.sleep(0.3)
                print("button 0 pressed, Enter params for RED")
                pmwR_flag = True;pmwG_flag = False;pmwB_flag = False
                user_input()
            elif (btns[1].read()==1):
                await asyncio.sleep(0.3)
                print("button 1 pressed, Enter params for GREEN")
                pmwG_flag = True;pmwR_flag = False;pmwB_flag = False
                user_input()
            elif (btns[2].read()==1):
                await asyncio.sleep(0.3)
                print("button 2 pressed, Enter params for BLUE")
                pmwB_flag = True;pmwR_flag = False;pmwG_flag = False
                user_input()
            else:
                await asyncio.sleep(0.3)
                print("button 3 pressed, Exit program")
                threads_flag = False
                _loop.stop()
### main sequence ##
write_gpio(PIN_R, 0);write_gpio(PIN_G, 0);write_gpio(PIN_B, 0);
time.sleep(0.2)
loop = asyncio.new_event_loop();  # Instance event_loop object
loop.create_task(RGB_leds());   #
loop.create_task(get_btns(loop)); # take user input buttons
print("Press buttons 0-3\n   0 - RED,\n   1 - GREEN\n   2 - BLUE\n   3 - Exit program")
loop.run_forever()
loop.close()
print("Done.")
```

```
Press buttons 0-3
   0 - RED,
   1 - GREEN
   2 - BLUE
   3 - Exit program
....button 0 pressed, Enter params for RED
  Enter Frequency(Hz): {default}110
period = 0.00909090909090909
Enter Duty percentage: {default}25
duty = 25
on=0.0022727272727272726, off=0.006818181818181818
.button 0 pressed, Enter params for RED
  Enter Frequency(Hz): {default}110
period = 0.00909090909090909
Enter Duty percentage: {default}50
duty = 50
on=0.0045454545454454545, off=0.0045454545454454545
button 0 pressed, Enter params for RED
  Enter Frequency(Hz): {default}110
period = 0.00909090909090909
Enter Duty percentage: {default}75
duty = 75
on=0.006818181818181818, off=0.0022727272727272726
button 0 pressed, Enter params for RED
  Enter Frequency(Hz): {default}110
period = 0.00909090909090909
Enter Duty percentage: {default}100
duty = 99
on=0.009, off=9.090909090909115e-05
```

See DEMO video "Assigm1_solution4.mp4"

https://drive.google.com/drive/folders/1jYQ4IE-Q7_aIjBgwKIEFN_wpze6EYsMB?usp=sharing

Comments/ Observations

Implementation/ design/ blocks:

This application has the same architectural skeleton has #3, Multi-threading capabilities (**async**):

Thread1: User_input(): Arbitrary parameters values (period & Duty cycle %)

Thread2: get_btns(): User is allowed to change parameter values per LED color by choosing the corresponding button.

Thread3: RGB_leds(): Infinitum task loop to execute based on the parameters' states

Difficulties:

I had one problem: same reported on #3

Testing tactic:

Same Frequency of 110 Hz was entered during user input except DUTY CYCLE values were changed (25, 50, 75 & 100%), then from continuous video, the LED picture was taken by using the snipping Tool

Conclusions:

To measure LED perceived brightness intensity is very subjective to each individual person and depends on the environment lighting at the moment of the test.

In this test in particular Same frequency was programmed (110Hz), Just varying Duty Cycle of 25, 50, 75 & 100%, the perceived brightness normally remains the same approximately after 55% then start to flatten the perception

See DEMO video "Assigm1_solution3.mp4"

https://drive.google.com/drive/folders/1jYQ4IE-Q7_aIjBgwKIEFN_wpze6EYsMB?usp=sharing

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.

5. E. 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.

In [1]:

```
from pynq.overlays.base import BaseOverlay
import time
from datetime import datetime
base = BaseOverlay("base.bit")
```

In [2]:

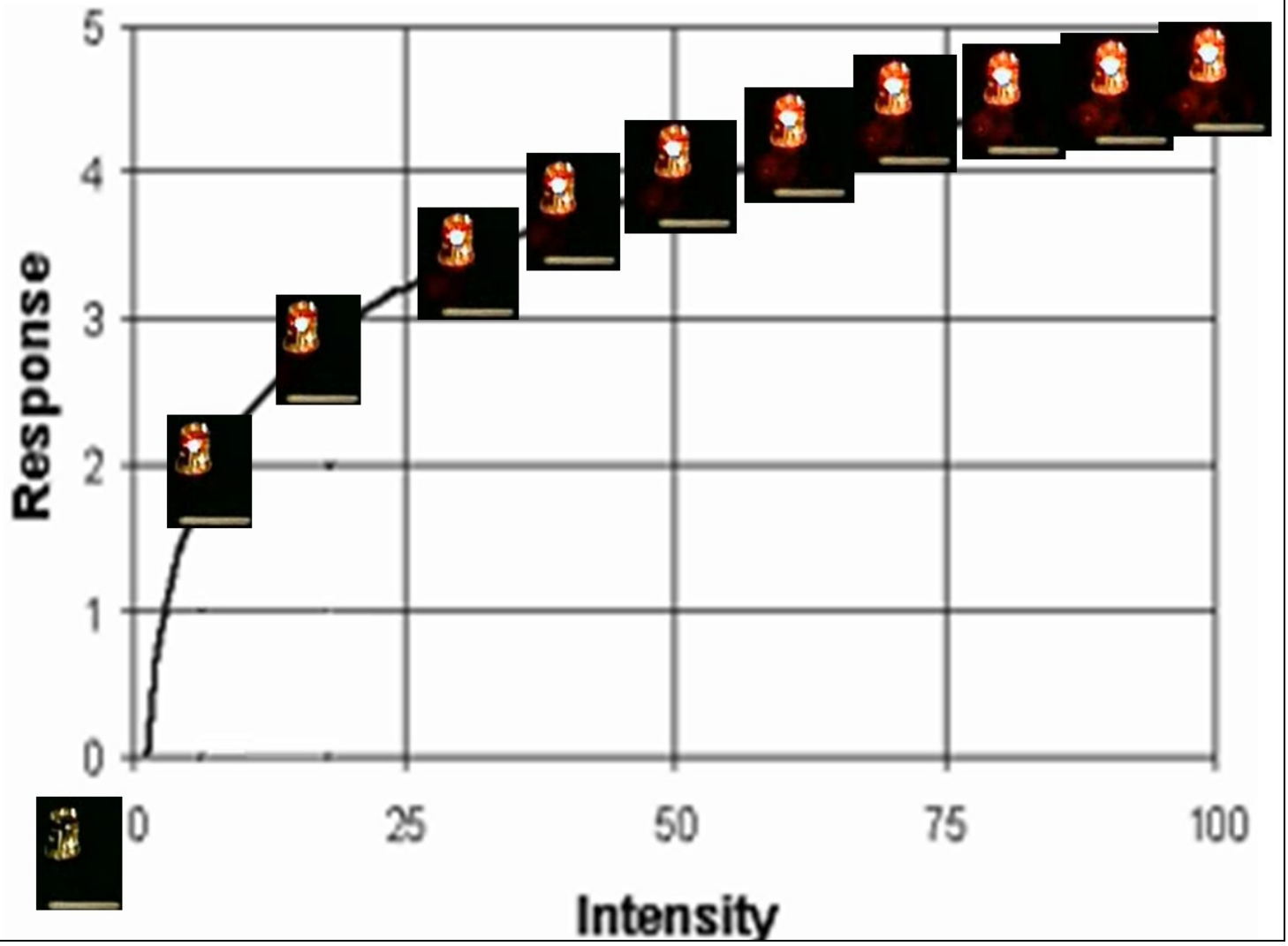
```
%%microblaze base.PMODB
#include "gpio.h"
#include "pyprintf.h"
//Function to turn on/off a selected pin of PMODB
void write_gpio(unsigned int pin, unsigned int val){
    if (val > 1){
        pyprintf("pin value must be 0 or 1");
    }
    gpio pin_out = gpio_open(pin);
    gpio_set_direction(pin_out, 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);        //instance object
    gpio_set_direction(pin_in, GPIO_IN); //configure to be Input
    return gpio_read(pin_in);
}
//Function to reset all GPIOs of PMODB
void reset_gpio_PMODB_all(){
    pyprintf("Reset all GPIOs of PMODB");
    for (int pin = 0; pin < 8; pin++) {
        gpio pin_out = gpio_open(pin);
        gpio_set_direction(pin_out, GPIO_OUT);
        gpio_write(pin_out, 0);
    }
}
```

In [ ]:

```
import asyncio
threads_flag = True    # "cond" will keep threads live or kill them all
# Initial states
pmwR_flag = False
pmwG_flag = False
pmwB_flag = False
delay = 1;period = 1;duty = 0
PIN_R = 3;PIN_G = 2;PIN_B = 1;PIN_0 = 0
period=110;duty=90;on=0.5;off=0.5;blinks=0;blink_stop=100000
def user_input():
    globals()['write_gpio']=write_gpio
    global threads_flag, blinks, blink_stop, on, off, pmwR_flag, pmwG_flag, pmwB_flag, pmw_flag, period, duty, PIN_R, PI
    ret = read_gpio(PIN_R);ret = read_gpio(PIN_G);ret = read_gpio(PIN_B)
    write_gpio(PIN_R, 0);write_gpio(PIN_G, 0);write_gpio(PIN_B, 0); #reset_gpio_PMODB_all()
    default = 110;val1 = input("  Enter Frequency(Hz): {default}") or default
    freq = min(max(int(val1),1),65535)
    period = 1/freq# T=1/f
    print("period = "+ str(period))
    duty = 0
    print("Start Duty Cycle = "+ str(duty))

async def RGB_leds():
    globals()['write_gpio']=write_gpio
    global threads_flag, blinks, blink_stop, on, off, pmwR_flag, pmwG_flag, pmwB_flag, pmw_flag, period, duty, PIN_R, PI
    while threads_flag:
        if blinks >= blink_stop:
            pmwR_flag = pmwG_flag = pmwB_flag = False
            blinks = 0
            write_gpio(PIN_R, 0);write_gpio(PIN_G, 0);write_gpio(PIN_B, 0); #reset_gpio_PMODB_all()
        await asyncio.sleep(off)
        if (pmwR_flag == True):
            write_gpio(PIN_R, 1)
            await asyncio.sleep(on)
            ret = read_gpio(PIN_R)
            write_gpio(PIN_R, 0)
            blinks += 1
        elif (pmwG_flag == True):
            write_gpio(PIN_G, 1)
            await asyncio.sleep(on)
            ret = read_gpio(PIN_R)
            write_gpio(PIN_G, 0)
            blinks += 1
        elif (pmwB_flag == True):
            write_gpio(PIN_B, 1)
            await asyncio.sleep(on)
            ret = read_gpio(PIN_R)
            write_gpio(PIN_B, 0)
            blinks += 1
        else:
            await asyncio.sleep(1)
            print('.', end='')
async def DutyCycleIncreasing():
    globals()['write_gpio']=write_gpio
    global threads_flag, blinks, blink_stop, on, off, pmwR_flag, pmwG_flag, pmwB_flag, pmw_flag, period, duty, PIN_R, PI
    while threads_flag:
        if pmwR_flag == True or pmwG_flag == True or pmwB_flag == True:
            if duty > 100:
                duty = 0
                pmwR_flag = False;pmwG_flag = False;pmwB_flag = False
            else:
                duty += 5
                await asyncio.sleep(0.5)
                duty = min(max(int(duty),0),100)
                on = (duty/100)*period
                off = period-on
                print(str(duty)+" ", end='')
        else:
            await asyncio.sleep(0.5)

async def get_btns(_loop):
    btns = base.btns_gpio
    globals()['write_gpio']=write_gpio
    global threads_flag, blinks, blink_stop, delay, pmwR_flag, pmwG_flag, pmwB_flag, pmw_flag, period, duty, PIN_R, PIN_
    while threads_flag:
        await asyncio.sleep(0.1)
        if btns.read() != 0:
            if (btns[0].read()==1):
                await asyncio.sleep(0.3)
                print("button 0 pressed, Enter params for RED")
                pmwR_flag = True;pmwG_flag = False;pmwB_flag = False
                user_input()
            elif (btns[1].read()==1):
                await asyncio.sleep(0.3)
                print("button 1 pressed, Enter params for GREEN")
                pmwG_flag = True;pmwR_flag = False;pmwB_flag = False
                user_input()
            elif (btns[2].read()==1):
                await asyncio.sleep(0.3)
                print("button 2 pressed, Enter params for BLUE")
                pmwB_flag = True;pmwR_flag = False;pmwG_flag = False
                user_input()
            else:
                await asyncio.sleep(0.3)
                print("button 3 pressed, Exit program")
                threads_flag = False
                _loop.stop()
### main sequence ##
write_gpio(PIN_R, 0);write_gpio(PIN_G, 0);write_gpio(PIN_B, 0);
time.sleep(0.2)
loop = asyncio.new_event_loop();  # Instance event_loop object
loop.create_task(RGB_leds());   #
loop.create_task(DutyCycleIncreasing());   #
loop.create_task(get_btns(loop)); # take user input buttons

print("Press buttons 0-3\n   0 - RED,\n   1 - GREEN\n   2 - BLUE\n   3 - Exit program")
loop.run_forever()
loop.close()
print("Done.")
```

```
Press buttons 0-3
   0 - RED,
   1 - GREEN
   2 - BLUE
   3 - Exit program
..button 0 pressed, Enter params for RED
  Enter Frequency(Hz): {default}110
period = 0.009090909090909091
Start Duty Cycle = 0
.5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100
```

Perceived Brightness (relative to 100% duty cycle)
VS
Duty Cycle Intensity



Comments/ Observations

Implementation/ design/ blocks:

This application has the same architectural skeleton has #3, Multi-threading capabilities (**async**):

Thread1: User_input(): Arbitrary parameters values (period & Duty cycle %)

Thread2: get_btns(): User is allowed to change parameter values per LED color by choosing the corresponding button.

Thread3: RGB_leds(): Infinitum task loop to execute based on the parameters' states

Difficulties:

I had one problem: same reported on #3

Testing tactic:

Same Frequency of 110 Hz was entered during user input except DUTY CYCLE values were changed (10, 20, 30, 40, 50, 60, 70, 80, 90 & 100%), then from continuous video, the LED picture was taken by using the snipping Tool

Conclusions:

To measure LED perceived brightness intensity is very subjective to each individual person and depends on the environment lighting at the moment of the test.

In this test in particular Same frequency was programmed (110Hz), Just varying Duty Cycle of 10, 20, 30, 40, 50, 60, 70, 80, 90 & 100%, the perceived brightness normally remains the same approximately after 55% then start to flatten the perception, really following Weber's law non-linear brightness perception

See DEMO video "Assigm1_solution5.mp4"

https://drive.google.com/drive/folders/1jYQ4IE-Q7_aljBgwKlEFN_wpze6EYsMB?usp=sharing

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.

Weber's Law and Fechner's Law

Introduction

This handout describes Weber's law and Fechner's law. You are not responsible for the mathematics presented here, but you should know what the terms Weber's law and Fechner's law refer to.

Discrimination Thresholds

Weber's law expresses a general relationship between a quantity or intensity of something and how much more needs to be added for us to be able to tell that something has been added. Experiments designed to find out such things are called *discrimination threshold* experiments, because the observer is asked to tell apart, or discriminate, two things that differ by only a slight increment. The discrimination threshold, then, is the smallest detectable increment above whatever the initial intensity was. You might imagine that this increment is not the same for the entire range of initial intensities that are possible—generally speaking, you're right. The smallest detectable increment typically changes depending on what the starting intensity is.

For instance, if we are trying to discriminate which of two flashes of light lasts longer, we may be able to tell the difference between a short flash and a slightly longer one. But if the same slight difference in duration is added to a very long flash, we are less able to tell the difference between the slightly-longer long flash from the long flash by itself. Similarly, for a particular combination of target and masker sounds, our ability to detect a tone in the presence of a masker depends on how intense the masker is. If the masker is not very intense, we don't have to increase the intensity of the tone very much to be able to detect it. If the masker is more intense, the same small increase in intensity will not make the tone audible anymore. We won't be able to detect it unless we make it considerably more intense.

Weber's law

This general relationship between the initial intensity of something and the smallest detectable increment is exactly what Weber noticed and formalized into "Weber's Law". The discrimination threshold, or the threshold for detecting an increment in the quantity or intensity of something, changes depending on how much there is before we add the increment. Weber's law is a hypothesis about how this threshold change happens. Let's call the initial intensity (the intensity before adding the increment) I, and let's call the amount needed to detect a difference ΔI. (Often, the specific letter that researchers choose bears some relation to the stimulus dimension being discriminated—if we are interested in duration discrimination, we might have D and ΔD.)

In a discrimination experiment, then, we are interested in measuring ΔI as a function of I. That is, we want to find the discrimination threshold ΔI such that a stimulus with an intensity $I+\Delta I$ is just discriminable from a stimulus of intensity I. Weber's law characterizes how the ΔI we measure depends on I. It states that

$$\Delta I = K_w I$$

for some constant K_W. The constant K_W is called the Weber Fraction. You may notice the similarity between this equation and the equation for a straight line, $y = m * x + b$. Weber's law expresses the equation for a straight line, with the slope m being the Weber Fraction, K_W and the y-intercept b being zero. Likewise, I plays the part of x and ΔI plays the part of y. Thus, if Weber's law holds, we expect the data to graph as a straight line through the origin, as shown in Figure 1.

Figure 1

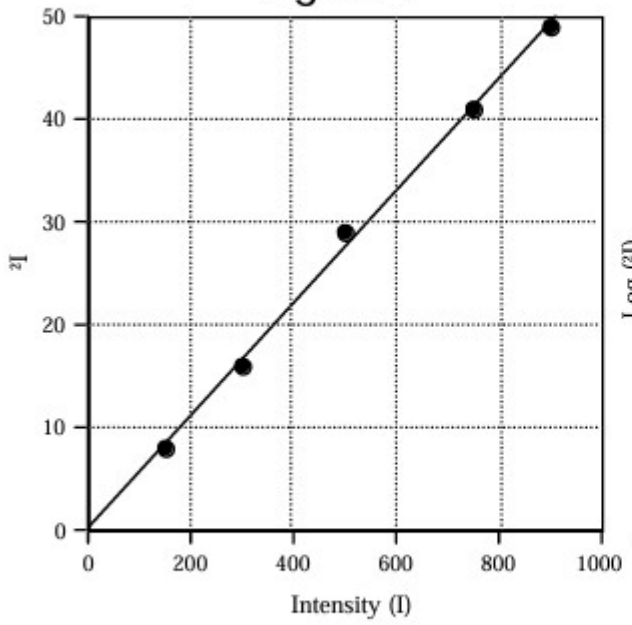
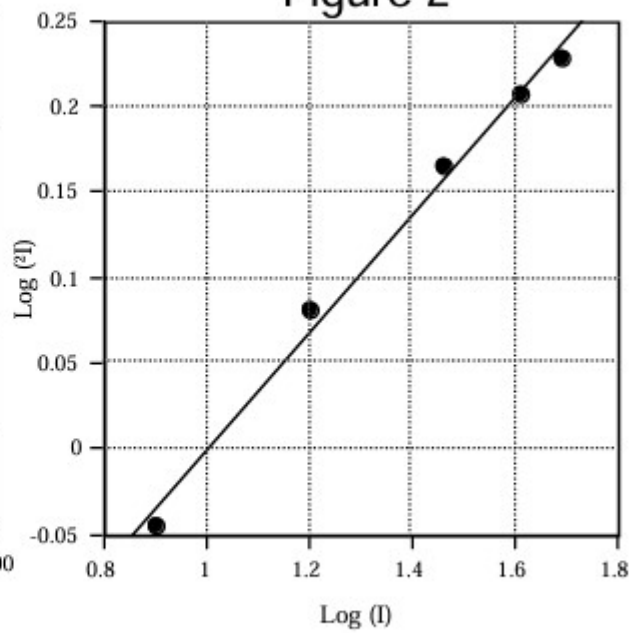


Figure 2



Other ways of expressing Weber's law

There are other ways of expressing Weber's law which might allow a more convenient visual analysis of whether or not Weber's law holds for a set of data. The first one described below requires you to divide the measured difference thresholds ΔI by the intensity I. The second one requires you to convert the discrimination thresholds and the base intensity into logarithms:

$$\frac{\Delta I}{I} = K_w .$$

In this formulation of Weber's law, if $\Delta I/I$ is plotted on the y-axis and I is plotted on the x-axis, you should see a horizontal straight line with intercept K_W if Weber's law holds. Another way of saying this is that no matter what intensity I we present to the observer, if Weber's law holds, we should measure a discrimination threshold ΔI that always results in the same number when we divide it by I. Thus, the fraction $\Delta I/I$ is a *constant* (K_W, the Weber Fraction) which does not change for different values of I. Thus,

$$\log(\Delta I) = \log(I) + \log(K_w) .$$

This formula is useful when I can take on a large range of values. If Weber's law holds, $\log(\Delta I)$ should plot as a straight-line function of $\log(I)$, with slope 1 and intercept K_W. In Figure 2 above, the data plotted in Figure 1 were transformed to logs and replotted. You can see that the data still fall along a straight line.

Why Weber's law is useful

As we have already seen, Weber's law is a useful way to summarize the relation between the discrimination threshold ΔI and the base intensity I. In general, this relation holds true for many

different physical dimensions (but not always—see below). If we know that Weber's law holds for two dimensions, we can compare our sensitivity to changes along those dimensions by comparing the Weber fractions. The Weber fraction is often expressed as a percentage. A Weber fraction of 1% indicates a fairly high sensitivity to increments, while a Weber fraction of 15% is rather poor and indicates lower sensitivity to increments.

When performing discrimination threshold experiments, the first analysis is often aimed at finding out whether or not Weber's law holds for that set of discriminations. If it does, we then want to find out what the Weber fraction is. By now, you can probably see that if we know Weber's law holds and we know what the Weber fraction is, we can *predict* what the discrimination threshold should be—it's just the Weber fraction times the base intensity.

How general is Weber's law?

You may have already suspected that Weber's law might not hold for absolutely all values of I. In fact, it's often the case that for very small values of I, Weber's law starts to break down. Here is a generalized formulation of Weber's law that can accommodate the most common deviation from the simple form of the law:

$$\Delta I = K_w(I + I_0).$$

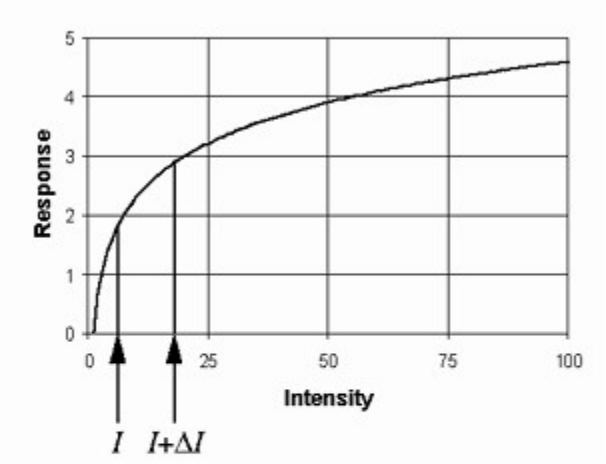
$K_W I_O$ is the absolute threshold—the smallest intensity of a given physical dimension we can reliably detect. You can see that when I is large, the difference threshold ΔI approaches $K_W * I$, which is the usual formulation for Weber's law. When I is near threshold, however, ΔI approaches $K_W * I_O$. This means that when I takes on values near threshold, the difference threshold ΔI is more influenced by the absolute threshold (which is approximately constant, ignoring a bit of noise) than it is by the changing value of I.

Fechner's law

Fechner's law provides an explanation for Weber's law. Fechner's explanation has two parts. The first part is that two stimuli will be discriminable if they generate a visual response that exceeds some threshold. The second part is that the visual response R to an intensity I is given by the equation

$$R = \log(I).$$

This relation is plotted in the next graph. Suppose we call the change in response necessary for discrimination "one". Then by Fechner's law the two intensities indicated by arrows will be just discriminable. The discrimination threshold ΔI is given by the distance on the x-axis between I and $I+\Delta I$.



If you use Fechner's law and the graph to find the discrimination threshold for a stimulus more intense than the I shown, you will you will discover that the discrimination threshold is larger. This is consistent with Weber's law, which predicts that the discrimination threshold grows as we increase the base intensity I.

Mathematical relation between Fechner's law and Weber's law

(You are not responsible for this math. I provide it for the curious.)

Fechner's law predicts that two lights will be just discriminable if the responses they generate differ by a constant amount. Let us call this amount ΔR. So Fechner's law says that when ΔI represents discrimination threshold,

$$\Delta R = \log(I + \Delta I) - \log(I).$$

Using the rules for logarithms, we can re-write this as

$$\Delta R = \log\big((I + \Delta I)/I\big).$$

Now raise the constant e to both sides of this equation:

$$e^{\Delta R} = e^{\log((I+\Delta I)/I)}.$$

Since ΔR is a constant, we can simply define a new constant $K = e^{\Delta R}$. Since $e^{\log(x)} = x$ for any x (by the definition of a logarithm), we can re-write our equation as

$$K = (I + \Delta I)/I.$$

Now multiply both sides by I:

$$KI = I + \Delta I.$$

Now subtract I from both sides:

$$KI - I = \Delta I.$$

Now regroup the terms on the left:

$$(K - 1)I = \Delta I.$$

Now define

$$K_w = K - 1 = e^{\Delta R} - 1.$$

Then we have Weber's law:

$$\Delta I = K_w I,$$

for some constant K_W. (And we have even shown how K_W depends on ΔR.) The result of this math is that if Fechner's law holds, then we can predict Weber's law.

Final notes for the brave of heart

(You are not responsible for this next paragraph.)

The visual response that plays a role in Fechner's law is closely related to the perceptual response that played a role in our discussion of the theory of signal detection. If you are curious, stop by during office hours and we can chat about the relation in more detail. I also note that there are conditions under which Weber's law does not hold. Again, if you are curious please come ask me to tell you more.

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.

---

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 (3) colors.

In [1]:

```python
from pynq.overlays.base import BaseOverlay
import time
from datetime import datetime
base = BaseOverlay("base.bit")
```

In [2]:

```
%%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);        //instance object
    gpio_set_direction(pin_in, GPIO_IN); //configure to be Input
    return gpio_read(pin_in);
}
//Function to reset all GPIOs of PMODB
void reset_gpio_PMODB_all(){
    pyprintf("Reset all GPIOs of PMODB");
    for (int pin = 0; pin < 8; pin++) {
        gpio pin_out = gpio_open(pin);
        gpio_set_direction(pin_out, GPIO_OUT);
        gpio_write(pin_out, 0);
    }
}
```

In [3]:

```
import asyncio
threads_flag = True    # "cond" will keep threads live or kill them all
# Initial states
pmwR_flag = True
pmwG_flag = False
pmwB_flag = False
delay = 1;period = 0.5;duty = 50
PIN_R = 3;PIN_G = 2;PIN_B = 1;PIN_0 = 0
on=1;off=1;blinks=0;blink_stop=100000
async def RGB_leds():
    globals()['write_gpio']=write_gpio
    global threads_flag, blinks, blink_stop, on, off, pmwR_flag, pmwG_flag, pmwB_flag, pmw_flag, period, duty, PIN_R, PI
    while threads_flag:
        if blinks >= blink_stop:
            pmwR_flag = pmwG_flag = pmwB_flag = False
            blinks = 0
            write_gpio(PIN_R, 0);write_gpio(PIN_G, 0);write_gpio(PIN_B, 0); #reset_gpio_PMODB_all()
        await asyncio.sleep(off)
        if (pmwR_flag == True):
            write_gpio(PIN_R, 1)
            await asyncio.sleep(on)
            ret = read_gpio(PIN_R)
            write_gpio(PIN_R, 0);write_gpio(PIN_G, 0);write_gpio(PIN_B, 0);
            blinks += 1
        elif (pmwG_flag == True):
            write_gpio(PIN_G, 1)
            await asyncio.sleep(on)
            ret = read_gpio(PIN_R)
            write_gpio(PIN_R, 0);write_gpio(PIN_G, 0);write_gpio(PIN_B, 0);
            blinks += 1
        elif (pmwB_flag == True):
            write_gpio(PIN_B, 1)
            await asyncio.sleep(on)
            ret = read_gpio(PIN_R)
            write_gpio(PIN_R, 0);write_gpio(PIN_G, 0);write_gpio(PIN_B, 0);
            blinks += 1
        else:
            await asyncio.sleep(1)
            print('.', end='')
async def get_btns(_loop):
    btns = base.btns_gpio
    globals()['write_gpio']=write_gpio
    global threads_flag, blinks, blink_stop, delay, pmwR_flag, pmwG_flag, pmwB_flag, pmw_flag, period, duty, PIN_R, PIN_
    while threads_flag:
        await asyncio.sleep(0.1)
        if btns.read() != 0:
            if (btns[0].read()==1):
                await asyncio.sleep(0.3)
                print("button 0 pressed, Enter params for RED")
                ret = read_gpio(PIN_R);ret = read_gpio(PIN_G);ret = read_gpio(PIN_B)
                write_gpio(PIN_R, 0);write_gpio(PIN_G, 0);write_gpio(PIN_B, 0);
                pmwR_flag = True;pmwG_flag = False;pmwB_flag = False
            elif (btns[1].read()==1):
                await asyncio.sleep(0.3)
                print("button 1 pressed, Enter params for GREEN")
                ret = read_gpio(PIN_R);ret = read_gpio(PIN_G);ret = read_gpio(PIN_B)
                write_gpio(PIN_R, 0);write_gpio(PIN_G, 0);write_gpio(PIN_B, 0);
                pmwG_flag = True;pmwR_flag = False;pmwB_flag = False
            elif (btns[2].read()==1):
                await asyncio.sleep(0.3)
                print("button 2 pressed, Enter params for BLUE")
                ret = read_gpio(PIN_R);ret = read_gpio(PIN_G);ret = read_gpio(PIN_B)
                write_gpio(PIN_R, 0);write_gpio(PIN_G, 0);write_gpio(PIN_B, 0);
                pmwB_flag = True;pmwR_flag = False;pmwG_flag = False
            else:
                await asyncio.sleep(0.3)
                print("button 3 pressed, Exit program")
                threads_flag = False
                await asyncio.sleep(1)
                ret = read_gpio(PIN_R);ret = read_gpio(PIN_G);ret = read_gpio(PIN_B)
                write_gpio(PIN_R, 0);write_gpio(PIN_G, 0);write_gpio(PIN_B, 0);
                _loop.stop()
### main sequence ##
ret = read_gpio(PIN_R);ret = read_gpio(PIN_G);ret = read_gpio(PIN_B)
write_gpio(PIN_R, 0);write_gpio(PIN_G, 0);write_gpio(PIN_B, 0);
freq = 0.5
period = 1/freq# T=1/f
print("period = "+ str(period))
print("duty = "+ str(duty))
on = (duty/100)*period
off = period-on
print("on="+str(on)+", off="+str(off))
loop = asyncio.new_event_loop();  # Instance event_loop object
loop.create_task(RGB_leds());   #
loop.create_task(get_btns(loop)); # take user input buttons
print("Press buttons 0-3\n   0 - RED,\n   1 - GREEN\n   2 - BLUE\n   3 - Exit program")
loop.run_forever()
loop.close()
print("Done.")
```

```
period = 2.0
duty = 50
on=1.0, off=1.0
Press buttons 0-3
   0 - RED,
   1 - GREEN
   2 - BLUE
   3 - Exit program
button 0 pressed, Enter params for RED
button 1 pressed, Enter params for GREEN
button 1 pressed, Enter params for GREEN
button 2 pressed, Enter params for BLUE
button 1 pressed, Enter params for GREEN
button 0 pressed, Enter params for RED
button 3 pressed, Exit program
Done.
```

Comments/ Observations

Implementation/ design/ blocks:

This application has the same architectural skeleton has #3, Multi-threading capabilities (**async**):

Thread1: User_input(): Arbitrary parameters values (period & Duty cycle %)

Thread2: get_btns(): User is allowed to change parameter values per LED color by choosing the corresponding button.

Thread3: RGB_leds(): Infinitum task loop to execute based on the parameters' states

Difficulties:

I had one problem: same reported on #3

Testing tactic:

Same Frequency of 110 Hz was entered during user input except DUTY CYCLE values were changed (10, 20, 30, 40, 50, 60, 70, 80, 90 & 100%), then from continuous video, the LED picture was taken by using the snipping Tool

Conclusions:

To measure LED perceived brightness intensity is very subjective to each individual person and depends on the environment lighting at the moment of the test.

In this test in particular Same frequency was programmed (110Hz), Just varying Duty Cycle of 10, 20, 30, 40, 50, 60, 70, 80, 90 & 100%, the perceived brightness normally remains the same approximately after 55% then start to flatten the perception, really following Weber's law non-linear brightness perception

See DEMO video "Assigm1_solution6.mp4"

https://drive.google.com/drive/folders/1jYQ4IE-Q7_aljBgwKlEFN_wpze6EYsMB?usp=sharing

Assignment 1

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.

In [ ]:
```
from pynq.overlays.base import BaseOverlay
#import time
from time import sleep
from datetime import datetime
base = BaseOverlay("base.bit")
```

In [ ]:
```
%%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);        //instance object
    gpio_set_direction(pin_in, GPIO_IN); //configure to be Input
    return gpio_read(pin_in);
}
//Function to reset all GPIOs of PMODB
void reset_gpio_PMODB_all(){
    pyprintf("Reset all GPIOs of PMODB");
    for (int pin = 0; pin < 8; pin++) {
        gpio pin_out = gpio_open(pin);
        gpio_set_direction(pin_out, GPIO_OUT);
        gpio_write(pin_out, 0);
    }
}
```

In [ ]:
```
reset_gpio_PMODB_all()
```

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.

In [ ]:
```
# Single PWM for any period and duty cycle 1 to 99%
from pynq.overlays.base import BaseOverlay
base = BaseOverlay("base.bit")
from pynq.lib import Pmod_PWM
import time
# globals
```

```
PIN_R = 3;PIN_G = 2;PIN_B = 1;PIN_0 = 0

def pmw_gen(period,duty):
    period = min(max(period,1),65535)
    duty = min(max(duty,1),99)
    pwm.generate(period,duty)

pwm = Pmod_PWM(base.PMODB,PIN_0)
pmw_gen(10,50)
time.sleep(20)
pwm.stop()
```

In [ ]:
```
# Reset RED & BLUE leds, so we can vary the BLUE one
reset_gpio_PMODB_all()
```

In [ ]:
```

```

In [ ]:
```

```

1. Find the optimal PWM frequency, such that the physical flashing phenonmenon will not be perceived visually;

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

In [ ]:
```
%%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);        //instance object
    gpio_set_direction(pin_in, GPIO_IN); //configure to be Input
    return gpio_read(pin_in);
}
//Function to reset all GPIOs of PMODB
void reset_gpio_PMODB_all(){
    pyprintf("Reset all GPIOs of PMODB");
    for (int pin = 0; pin < 8; pin++) {
        gpio pin_out = gpio_open(pin);
        gpio_set_direction(pin_out, GPIO_OUT);
        gpio_write(pin_out, 0);
    }
}
```

In [ ]:
```
import asyncio
threads_flag = True    # "cond" will keep threads live or kill them all
# Initial states
pmwR_flag = False
pmwG_flag = False
pmwB_flag = False
delay = 1;period = 1;duty = 25
PIN_R = 3;PIN_G = 2;PIN_B = 1;PIN_0 = 0
period=10;duty=90;on=0.5;off=0.5;blinks=0;blink_stop=100000
def user_input():
    globals()['write_gpio']=write_gpio
    global threads_flag, blinks, blink_stop, on, off, pmwR_flag, pmwG_flag, pmwB_flag,
    ret = read_gpio(PIN_R);
    ret = read_gpio(PIN_G)
    ret = read_gpio(PIN_B)
    write_gpio(PIN_R, 0);write_gpio(PIN_G, 0);write_gpio(PIN_B, 0); #reset_gpio_PMODB_
    default = 100;val1 = input("  Enter Frequency(Hz): {default}") or default
    freq = min(max(int(val1),1),65535)
    period = 1/freq# T=1/f
    print("period = "+ str(period))
    default = 25;val2 =  input("Enter Duty percentage: {default}") or default
    duty = min(max(int(val2),1),99)
    print("duty = "+ str(duty))
    on = (duty/100)*period
    off = period-on
    print("on="+str(on)+", off="+str(off))
async def RGB_leds():
    globals()['write_gpio']=write_gpio
    global threads_flag, blinks, blink_stop, on, off, pmwR_flag, pmwG_flag, pmwB_flag,
    while threads_flag:
        if blinks >= blink_stop:
            pmwR_flag = pmwG_flag = pmwB_flag = False
            blinks = 0
            write_gpio(PIN_R, 0);write_gpio(PIN_G, 0);write_gpio(PIN_B, 0); #reset_gpi
        await asyncio.sleep(off)
        if (pmwR_flag == True):
            write_gpio(PIN_R, 1)
            await asyncio.sleep(on)
            ret = read_gpio(PIN_R)
            write_gpio(PIN_R, 0)
            blinks += 1
        elif (pmwG_flag == True):
            write_gpio(PIN_G, 1)
            await asyncio.sleep(on)
            ret = read_gpio(PIN_R)
            write_gpio(PIN_G, 0)
            blinks += 1
        elif (pmwB_flag == True):
            write_gpio(PIN_B, 1)
            await asyncio.sleep(on)
            ret = read_gpio(PIN_R)
            write_gpio(PIN_B, 0)
            blinks += 1
        else:
            await asyncio.sleep(1)
            print('.', end='')
async def get_btns(_loop):
    btns = base.btns_gpio
    globals()['write_gpio']=write_gpio
```

```
    global threads_flag, blinks, blink_stop, delay, pmwR_flag, pmwG_flag, pmwB_flag, p
    while threads_flag:
        await asyncio.sleep(0.1)
        if btns.read() != 0:
            if (btns[0].read()==1):
                await asyncio.sleep(0.3)
                print("button 0 pressed, Enter params for RED")
                pmwR_flag = True;pmwG_flag = False;pmwB_flag = False
                user_input()
            elif (btns[1].read()==1):
                await asyncio.sleep(0.3)
                print("button 1 pressed, Enter params for GREEN")
                pmwG_flag = True;pmwR_flag = False;pmwB_flag = False
                user_input()
            elif (btns[2].read()==1):
                await asyncio.sleep(0.3)
                print("button 2 pressed, Enter params for BLUE")
                pmwB_flag = True;pmwR_flag = False;pmwG_flag = False
                user_input()
            else:
                await asyncio.sleep(0.3)
                print("button 3 pressed, Exit program")
                threads_flag = False
                _loop.stop()
### main sequence ##
write_gpio(PIN_R, 0);write_gpio(PIN_G, 0);write_gpio(PIN_B, 0);
time.sleep(0.2)
loop = asyncio.new_event_loop();  # Instance event_loop object
loop.create_task(RGB_leds());   #
loop.create_task(get_btns(loop)); # take user input buttons
print("Press buttons 0-3\n   0 - RED,\n   1 - GREEN\n   2 - BLUE\n   3 - Exit program"
loop.run_forever()
loop.close()
print("Done.")
```

In [ ]:

1. Achieve the visually perceived 100%, 75%, 50% and 25% of full LED brightness by adjusting the duty cycle

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

In [ ]:
```
%%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);        //instance object
    gpio_set_direction(pin_in, GPIO_IN); //configure to be Input
    return gpio_read(pin_in);
}
//Function to reset all GPIOs of PMODB
void reset_gpio_PMODB_all(){
    pyprintf("Reset all GPIOs of PMODB");
    for (int pin = 0; pin < 8; pin++) {
        gpio pin_out = gpio_open(pin);
        gpio_set_direction(pin_out, GPIO_OUT);
        gpio_write(pin_out, 0);
    }
}
```

In [ ]:

```
import asyncio
threads_flag = True    # "cond" will keep threads live or kill them all
# Initial states
pmwR_flag = False
pmwG_flag = False
pmwB_flag = False
delay = 1;period = 1;duty = 25
PIN_R = 3;PIN_G = 2;PIN_B = 1;PIN_0 = 0
period=110;duty=90;on=0.5;off=0.5;blinks=0;blink_stop=100000
def user_input():
    globals()['write_gpio']=write_gpio
    global threads_flag, blinks, blink_stop, on, off, pmwR_flag, pmwG_flag, pmwB_flag,
    ret = read_gpio(PIN_R);
    ret = read_gpio(PIN_G)
    ret = read_gpio(PIN_B)
    write_gpio(PIN_R, 0);write_gpio(PIN_G, 0);write_gpio(PIN_B, 0); #reset_gpio_PMODB_
    default = 110;val1 = input("  Enter Frequency(Hz): {default}") or default
    freq = min(max(int(val1),1),65535)
    period = 1/freq# T=1/f
    print("period = "+ str(period))
    default = 25;val2 =  input("Enter Duty percentage: {default}") or default
    duty = min(max(int(val2),1),99)
    print("duty = "+ str(duty))
    on = (duty/100)*period
    off = period-on
    print("on="+str(on)+", off="+str(off))
async def RGB_leds():
    globals()['write_gpio']=write_gpio
    global threads_flag, blinks, blink_stop, on, off, pmwR_flag, pmwG_flag, pmwB_flag,
    while threads_flag:
        if blinks >= blink_stop:
            pmwR_flag = pmwG_flag = pmwB_flag = False
            blinks = 0
            write_gpio(PIN_R, 0);write_gpio(PIN_G, 0);write_gpio(PIN_B, 0); #reset_gpi
        await asyncio.sleep(off)
        if (pmwR_flag == True):
            write_gpio(PIN_R, 1)
            await asyncio.sleep(on)
            ret = read_gpio(PIN_R)
            write_gpio(PIN_R, 0)
            blinks += 1
        elif (pmwG_flag == True):
            write_gpio(PIN_G, 1)
```

```
            await asyncio.sleep(on)
            ret = read_gpio(PIN_R)
            write_gpio(PIN_G, 0)
            blinks += 1
        elif (pmwB_flag == True):
            write_gpio(PIN_B, 1)
            await asyncio.sleep(on)
            ret = read_gpio(PIN_R)
            write_gpio(PIN_B, 0)
            blinks += 1
        else:
            await asyncio.sleep(1)
            print('.', end='')
async def get_btns(_loop):
    btns = base.btns_gpio
    globals()['write_gpio']=write_gpio
    global threads_flag, blinks, blink_stop, delay, pmwR_flag, pmwG_flag, pmwB_flag, p
    while threads_flag:
        await asyncio.sleep(0.1)
        if btns.read() != 0:
            if (btns[0].read()==1):
                await asyncio.sleep(0.3)
                print("button 0 pressed, Enter params for RED")
                pmwR_flag = True;pmwG_flag = False;pmwB_flag = False
                user_input()
            elif (btns[1].read()==1):
                await asyncio.sleep(0.3)
                print("button 1 pressed, Enter params for GREEN")
                pmwG_flag = True;pmwR_flag = False;pmwB_flag = False
                user_input()
            elif (btns[2].read()==1):
                await asyncio.sleep(0.3)
                print("button 2 pressed, Enter params for BLUE")
                pmwB_flag = True;pmwR_flag = False;pmwG_flag = False
                user_input()
            else:
                await asyncio.sleep(0.3)
                print("button 3 pressed, Exit program")
                threads_flag = False
                _loop.stop()
### main sequence ##
write_gpio(PIN_R, 0);write_gpio(PIN_G, 0);write_gpio(PIN_B, 0);
time.sleep(0.2)
loop = asyncio.new_event_loop();  # Instance event_loop object
loop.create_task(RGB_leds());   #
loop.create_task(get_btns(loop)); # take user input buttons
print("Press buttons 0-3\n   0 - RED,\n   1 - GREEN\n   2 - BLUE\n   3 - Exit program"
loop.run_forever()
loop.close()
print("Done.")
```

1. A. 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.

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

```
from datetime import datetime
base = BaseOverlay("base.bit")
```

In [ ]:

```
%%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);        //instance object
    gpio_set_direction(pin_in, GPIO_IN); //configure to be Input
    return gpio_read(pin_in);
}
//Function to reset all GPIOs of PMODB
void reset_gpio_PMODB_all(){
    pyprintf("Reset all GPIOs of PMODB");
    for (int pin = 0; pin < 8; pin++) {
        gpio pin_out = gpio_open(pin);
        gpio_set_direction(pin_out, GPIO_OUT);
        gpio_write(pin_out, 0);
    }
}
```

In [ ]:

```
import asyncio
threads_flag = True    # "cond" will keep threads live or kill them all
# Initial states
pmwR_flag = False
pmwG_flag = False
pmwB_flag = False
delay = 1;period = 1;duty = 0
PIN_R = 3;PIN_G = 2;PIN_B = 1;PIN_0 = 0
period=110;duty=90;on=0.5;off=0.5;blinks=0;blink_stop=100000
def user_input():
    globals()['write_gpio']=write_gpio
    global threads_flag, blinks, blink_stop, on, off, pmwR_flag, pmwG_flag, pmwB_flag,
    ret = read_gpio(PIN_R);ret = read_gpio(PIN_G);ret = read_gpio(PIN_B)
    write_gpio(PIN_R, 0);write_gpio(PIN_G, 0);write_gpio(PIN_B, 0); #reset_gpio_PMODB_
    default = 110;val1 = input("  Enter Frequency(Hz): {default}") or default
    freq = min(max(int(val1),1),65535)
    period = 1/freq# T=1/f
    print("period = "+ str(period))
    duty = 0
    print("Start Duty Cycle = "+ str(duty))

async def RGB_leds():
    globals()['write_gpio']=write_gpio
    global threads_flag, blinks, blink_stop, on, off, pmwR_flag, pmwG_flag, pmwB_flag,
    while threads_flag:
        if blinks >= blink_stop:
            pmwR_flag = pmwG_flag = pmwB_flag = False
            blinks = 0
```

```
            write_gpio(PIN_R, 0);write_gpio(PIN_G, 0);write_gpio(PIN_B, 0); #reset_gpi
        await asyncio.sleep(off)
        if (pmwR_flag == True):
            write_gpio(PIN_R, 1)
            await asyncio.sleep(on)
            ret = read_gpio(PIN_R)
            write_gpio(PIN_R, 0)
            blinks += 1
        elif (pmwG_flag == True):
            write_gpio(PIN_G, 1)
            await asyncio.sleep(on)
            ret = read_gpio(PIN_R)
            write_gpio(PIN_G, 0)
            blinks += 1
        elif (pmwB_flag == True):
            write_gpio(PIN_B, 1)
            await asyncio.sleep(on)
            ret = read_gpio(PIN_R)
            write_gpio(PIN_B, 0)
            blinks += 1
        else:
            await asyncio.sleep(1)
            print('.', end='')
async def DutyCycleIncreasing():
    globals()['write_gpio']=write_gpio
    global threads_flag, blinks, blink_stop, on, off, pmwR_flag, pmwG_flag, pmwB_flag,
    while threads_flag:
        if pmwR_flag == True or pmwG_flag == True or pmwB_flag == True:
            if duty > 100:
                duty = 0
                pmwR_flag = False;pmwG_flag = False;pmwB_flag = False
            else:
                duty += 5
                await asyncio.sleep(0.5)
                duty = min(max(int(duty),0),100)
                on = (duty/100)*period
                off = period-on
                print(str(duty)+" ", end='')
        else:
            await asyncio.sleep(0.5)

async def get_btns(_loop):
    btns = base.btns_gpio
    globals()['write_gpio']=write_gpio
    global threads_flag, blinks, blink_stop, delay, pmwR_flag, pmwG_flag, pmwB_flag, p
    while threads_flag:
        await asyncio.sleep(0.1)
        if btns.read() != 0:
            if (btns[0].read()==1):
                await asyncio.sleep(0.3)
                print("button 0 pressed, Enter params for RED")
                pmwR_flag = True;pmwG_flag = False;pmwB_flag = False
                user_input()
            elif (btns[1].read()==1):
                await asyncio.sleep(0.3)
                print("button 1 pressed, Enter params for GREEN")
                pmwG_flag = True;pmwR_flag = False;pmwB_flag = False
                user_input()
            elif (btns[2].read()==1):
                await asyncio.sleep(0.3)
```

```
                print("button 2 pressed, Enter params for BLUE")
                pmwB_flag = True;pmwR_flag = False;pmwG_flag = False
                user_input()
            else:
                await asyncio.sleep(0.3)
                print("button 3 pressed, Exit program")
                threads_flag = False
                _loop.stop()
### main sequence ##
write_gpio(PIN_R, 0);write_gpio(PIN_G, 0);write_gpio(PIN_B, 0);
time.sleep(0.2)
loop = asyncio.new_event_loop();  # Instance event_loop object
loop.create_task(RGB_leds());   #
loop.create_task(DutyCycleIncreasing());   #
loop.create_task(get_btns(loop)); # take user input buttons

print("Press buttons 0-3\n   0 - RED,\n   1 - GREEN\n   2 - BLUE\n   3 - Exit program"
loop.run_forever()
loop.close()
print("Done.")
```

In [ ]:
```
# Start here
from pynq.overlays.base import BaseOverlay
import time
from datetime import datetime
base = BaseOverlay("base.bit")
```

In [ ]:
```
%%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);        //intance object
    gpio_set_direction(pin_in, GPIO_IN); //configure to be Input
    return gpio_read(pin_in);
}
```

In [ ]:
```
from pynq.overlays.base import BaseOverlay
import pynq.lib.rgbled as rgbled
import time
# globals
base = BaseOverlay("base.bit")
import asyncio
led4 = rgbled.RGBLED(4)
led5 = rgbled.RGBLED(5)
threads_flag = True    # "cond" will keep threads live or kill them all
# Initial states
pmwR_flag = False
pmwG_flag = False
pmwB_flag = False
```

```
duty = 25
# PMW
from pynq.lib import Pmod_PWM
PIN_R = 3;PIN_G = 2;PIN_B = 1;PIN_0 = 0
period=10;duty=90
# Coroutines:
async def RGB_leds():
    global threads_flag, pwmR, pwmG, pwmB, pmwR_flag, pmwG_flag, pmwB_flag, pmw_flag,

    while threads_flag:
        if (pmwR_flag == True):
            pwmR = Pmod_PWM(base.PMODB,PIN_R)
            pwmR.generate(period,duty)
            write_gpio(PIN_G, 0);write_gpio(PIN_B, 0)
            pmwR_flag = False
            await asyncio.sleep(0.1)
        elif (pmwG_flag == True):
            pwmG = Pmod_PWM(base.PMODB,PIN_G)
            pwmG.generate(period,duty)
            write_gpio(PIN_R, 0);write_gpio(PIN_B, 0)
            pmwG_flag = False
            await asyncio.sleep(0.1)
        elif (pmwB_flag == True):
            pwmB = Pmod_PWM(base.PMODB,PIN_B)
            pwmB.generate(period,duty)
            write_gpio(PIN_R, 0);write_gpio(PIN_G, 0)
            pmwB_flag = False
            await asyncio.sleep(0.1)
        else:
            await asyncio.sleep(0.1)

async def get_btns(_loop):
    btns = base.btns_gpio
    global threads_flag, pwmR, pwmG, pwmB, pmwR_flag, pmwG_flag, pmwB_flag, pmw_flag,
    while threads_flag:
        await asyncio.sleep(0.01)
        if btns.read() != 0:
            if (btns[0].read()==1):
                await asyncio.sleep(0.3)
                print("buton 0 pressed, RED")
                pmwR_flag = True
            elif (btns[1].read()==1):
                await asyncio.sleep(0.3)
                print("buton 1 pressed, GREEN")
                pmwG_flag = True

            elif (btns[2].read()==1):
                await asyncio.sleep(0.3)
                print("buton 2 pressed, BLUE")
                pmwB_flag = True

            else:
                await asyncio.sleep(0.3)
                print("buton 3 pressed, Exit program")
                threads_flag = False
                pwmR.stop()
                pwmG.stop()
                pwmB.stop()
                _loop.stop()
### main sequence ##
```

```
pwmR = Pmod_PWM(base.PMODB,PIN_R)
pwmG = Pmod_PWM(base.PMODB,PIN_G)
pwmB = Pmod_PWM(base.PMODB,PIN_B)
time.sleep(0.5)
pwmR.stop()
pwmG.stop()
pwmB.stop()
write_gpio(PIN_R, 0);write_gpio(PIN_G, 0);write_gpio(PIN_B, 0);
loop = asyncio.new_event_loop();  # Instance event_loop object
loop.create_task(RGB_leds());   #
loop.create_task(get_btns(loop)); # take user input
print("Press buttons 0-3\n   0 - Start binking,\n   1 - Stop\n   2 - No action\n   3 -
loop.run_forever()
loop.close()
led4.write(0x0)
led5.write(0x0)
print("Done.")
```

In [ ]:

1. 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 (3) colors.

In [1]:
```
from pynq.overlays.base import BaseOverlay
import time
from datetime import datetime
base = BaseOverlay("base.bit")
```

In [2]:
```
%%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);        //instance object
```

```
    gpio_set_direction(pin_in, GPIO_IN); //configure to be Input
    return gpio_read(pin_in);
}
//Function to reset all GPIOs of PMODB
void reset_gpio_PMODB_all(){
    pyprintf("Reset all GPIOs of PMODB");
    for (int pin = 0; pin < 8; pin++) {
        gpio pin_out = gpio_open(pin);
        gpio_set_direction(pin_out, GPIO_OUT);
        gpio_write(pin_out, 0);
    }
}
```

In [3]:

```
import asyncio
threads_flag = True    # "cond" will keep threads live or kill them all
# Initial states
pmwR_flag = True
pmwG_flag = False
pmwB_flag = False
delay = 1;period = 0.5;duty = 50
PIN_R = 3;PIN_G = 2;PIN_B = 1;PIN_0 = 0
on=1;off=1;blinks=0;blink_stop=100000
async def RGB_leds():
    globals()['write_gpio']=write_gpio
    global threads_flag, blinks, blink_stop, on, off, pmwR_flag, pmwG_flag, pmwB_flag,
    while threads_flag:
        if blinks >= blink_stop:
            pmwR_flag = pmwG_flag = pmwB_flag = False
            blinks = 0
            write_gpio(PIN_R, 0);write_gpio(PIN_G, 0);write_gpio(PIN_B, 0); #reset_gpi
        await asyncio.sleep(off)
        if (pmwR_flag == True):
            write_gpio(PIN_R, 1)
            await asyncio.sleep(on)
            ret = read_gpio(PIN_R)
            write_gpio(PIN_R, 0);write_gpio(PIN_G, 0);write_gpio(PIN_B, 0);
            blinks += 1
        elif (pmwG_flag == True):
            write_gpio(PIN_G, 1)
            await asyncio.sleep(on)
            ret = read_gpio(PIN_R)
            write_gpio(PIN_R, 0);write_gpio(PIN_G, 0);write_gpio(PIN_B, 0);
            blinks += 1
        elif (pmwB_flag == True):
            write_gpio(PIN_B, 1)
            await asyncio.sleep(on)
            ret = read_gpio(PIN_R)
            write_gpio(PIN_R, 0);write_gpio(PIN_G, 0);write_gpio(PIN_B, 0);
            blinks += 1
        else:
            await asyncio.sleep(1)
            print('.', end='')
async def get_btns(_loop):
    btns = base.btns_gpio
    globals()['write_gpio']=write_gpio
    global threads_flag, blinks, blink_stop, delay, pmwR_flag, pmwG_flag, pmwB_flag, p
    while threads_flag:
        await asyncio.sleep(0.1)
        if btns.read() != 0:
```

```
            if (btns[0].read()==1):
                await asyncio.sleep(0.3)
                print("button 0 pressed, Enter params for RED")
                ret = read_gpio(PIN_R);ret = read_gpio(PIN_G);ret = read_gpio(PIN_B)
                write_gpio(PIN_R, 0);write_gpio(PIN_G, 0);write_gpio(PIN_B, 0);
                pmwR_flag = True;pmwG_flag = False;pmwB_flag = False
            elif (btns[1].read()==1):
                await asyncio.sleep(0.3)
                print("button 1 pressed, Enter params for GREEN")
                ret = read_gpio(PIN_R);ret = read_gpio(PIN_G);ret = read_gpio(PIN_B)
                write_gpio(PIN_R, 0);write_gpio(PIN_G, 0);write_gpio(PIN_B, 0);
                pmwG_flag = True;pmwR_flag = False;pmwB_flag = False
            elif (btns[2].read()==1):
                await asyncio.sleep(0.3)
                print("button 2 pressed, Enter params for BLUE")
                ret = read_gpio(PIN_R);ret = read_gpio(PIN_G);ret = read_gpio(PIN_B)
                write_gpio(PIN_R, 0);write_gpio(PIN_G, 0);write_gpio(PIN_B, 0);
                pmwB_flag = True;pmwR_flag = False;pmwG_flag = False
            else:
                await asyncio.sleep(0.3)
                print("button 3 pressed, Exit program")
                threads_flag = False
                await asyncio.sleep(1)
                ret = read_gpio(PIN_R);ret = read_gpio(PIN_G);ret = read_gpio(PIN_B)
                write_gpio(PIN_R, 0);write_gpio(PIN_G, 0);write_gpio(PIN_B, 0);
                _loop.stop()
### main sequence ##
ret = read_gpio(PIN_R);ret = read_gpio(PIN_G);ret = read_gpio(PIN_B)
write_gpio(PIN_R, 0);write_gpio(PIN_G, 0);write_gpio(PIN_B, 0);
freq = 0.5
period = 1/freq# T=1/f
print("period = "+ str(period))
print("duty = "+ str(duty))
on = (duty/100)*period
off = period-on
print("on="+str(on)+", off="+str(off))
loop = asyncio.new_event_loop();  # Instance event_loop object
loop.create_task(RGB_leds());   #
loop.create_task(get_btns(loop)); # take user input buttons
print("Press buttons 0-3\n   0 - RED,\n   1 - GREEN\n   2 - BLUE\n   3 - Exit program"
loop.run_forever()
loop.close()
print("Done.")
```

```
period = 2.0
duty = 50
on=1.0, off=1.0
Press buttons 0-3
   0 - RED,
   1 - GREEN
   2 - BLUE
   3 - Exit program
button 0 pressed, Enter params for RED
button 1 pressed, Enter params for GREEN
button 1 pressed, Enter params for GREEN
button 2 pressed, Enter params for BLUE
button 1 pressed, Enter params for GREEN
button 0 pressed, Enter params for RED
button 3 pressed, Exit program
Done.
```

In [ ]: