Interacting with GPIO from MicroBlaze

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

Lab work

Use the code from the second cell as a template and write a code to use two pins (0 and 1) for send and two pins (2 and 3) for receive. You should be able to send 2bits (0~3) over GPIO. You'll need to hardwire from the send pins to the receive pins.

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

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

//Function to turn on/off a selected pin of PMODB
int 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);
    return 1;
}

//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);
}

//Function to clear PMOD
int clear_gpio(){
    for (int i = 0; i < 8; i++)
    {
        write_gpio(i,0);
    }
    pyprintf("GPIO Cleared!\n");
    return 0;
}

//Multitasking the microblaze for a simple function
int add(int a, int b){
    return a + b;
}
```

In [3]:
```
write_gpio(1, 1) ## Blue
write_gpio(2, 0) ## Green
write_gpio(3, 1) ## Red
```

Out[3]:
```
1
```

In [4]:
```
clear_gpio() ## Testing the basics of PWM. Using 100 Hz refresh rate.
for x in range(100):
    write_gpio(1, 1)
    time.sleep(0.0025)
    write_gpio(1, 0)
    time.sleep(0.0075)

clear_gpio()
for x in range(100):
    write_gpio(2, 1)
    time.sleep(0.0025)
    write_gpio(2, 0)
    time.sleep(0.0075)

for x in range(100):
    write_gpio(3, 1)
    time.sleep(0.0025)
    write_gpio(3, 0)
    time.sleep(0.0075)

for x in range(100):
    write_gpio(1, 1)
    time.sleep(0.005)
    write_gpio(1, 0)
    time.sleep(0.005)

for x in range(100):
    write_gpio(1, 1)
    time.sleep(0.0075)
    write_gpio(1, 0)
    time.sleep(0.0025)

for x in range(100):
    write_gpio(1, 1)
    time.sleep(0.0025)
    write_gpio(1, 0)
    time.sleep(0.0075)
```

```
GPIO Cleared!
GPIO Cleared!
```

In [5]:
```
def setPWM(_duty): ## this assumes 100 Hz refresh rate
    if _duty > 100 or _duty < 0:
        print("Error! Out of range! Value not within 0-100")
    elif _duty == 0:
        duty = 1/100000
        offDuty = 999/100000
        return duty,offDuty
    elif _duty == 100:
        duty = 999/100000
        offDuty = 1/100000
        return duty,offDuty
    else:
        duty = _duty/10000
        offDuty = (100-_duty)/10000
        return duty,offDuty

a = setPWM(3) ## 25% perceived brightness
print("25% brightness...")
print(a[0])
print(a[1])

clear_gpio()

for x in range(100):
    write_gpio(1, 1)
    time.sleep(a[0])
    write_gpio(1, 0)
    time.sleep(a[1])

b = setPWM(20) ## 50% perceived brightness
print("50% brightness...")

for x in range(100):
    write_gpio(1, 1)
    time.sleep(b[0])
    write_gpio(1, 0)
    time.sleep(b[1])

c = setPWM(45) ## 75% perceived brightness
print("75% brightness...")

for x in range(100):
    write_gpio(1, 1)
    time.sleep(c[0])
    write_gpio(1, 0)
    time.sleep(c[1])

d = setPWM(100) ## 100% perceived brightness
print("100% brightness...")

for x in range(100):
    write_gpio(1, 1)
    time.sleep(d[0])
    write_gpio(1, 0)
    time.sleep(d[1])
```

```
25% brightness...
0.0003
0.0097
GPIO Cleared!
50% brightness...
75% brightness...
100% brightness...
```

In [7]:

```
import asyncio
clear_gpio()
cond = True
flag = True
flag1 = True
flag2 = False
flag3 = False

l = setPWM(80)
off = setPWM(0)
r = [3,1]
g = [2,1]
b = [1,1]
color = r


async def flash_leds():
    global cond, flag
    while cond:
        if flag:
            for x in range(75):
                write_gpio(color[0],color[1])
                time.sleep(l[0])
                write_gpio(color[0],0)
                time.sleep(l[1])
                await asyncio.sleep(0.01)
            for x in range(75):
                write_gpio(color[0],0)
                time.sleep(off[0])
                write_gpio(color[0],0)
                time.sleep(off[1])
                await asyncio.sleep(0.01)
        else:
            write_gpio(color[0],0)
            await asyncio.sleep(0.01)

async def get_btns(_loop):
    global cond, flag, color
    while cond:
        await asyncio.sleep(0.01)
        if btns[3].read() != 0:
            _loop.stop()
            flag = False
        elif btns[0].read() != 0:
            color = r
        elif btns[1].read() != 0:
            color = g
        elif btns[2].read() != 0:
            color = b

loop = asyncio.new_event_loop()
loop.create_task(flash_leds())
loop.create_task(get_btns(loop))
loop.run_forever()
loop.close()
clear_gpio()
print("Done.")
```

```
GPIO Cleared!
GPIO Cleared!
Done.
```

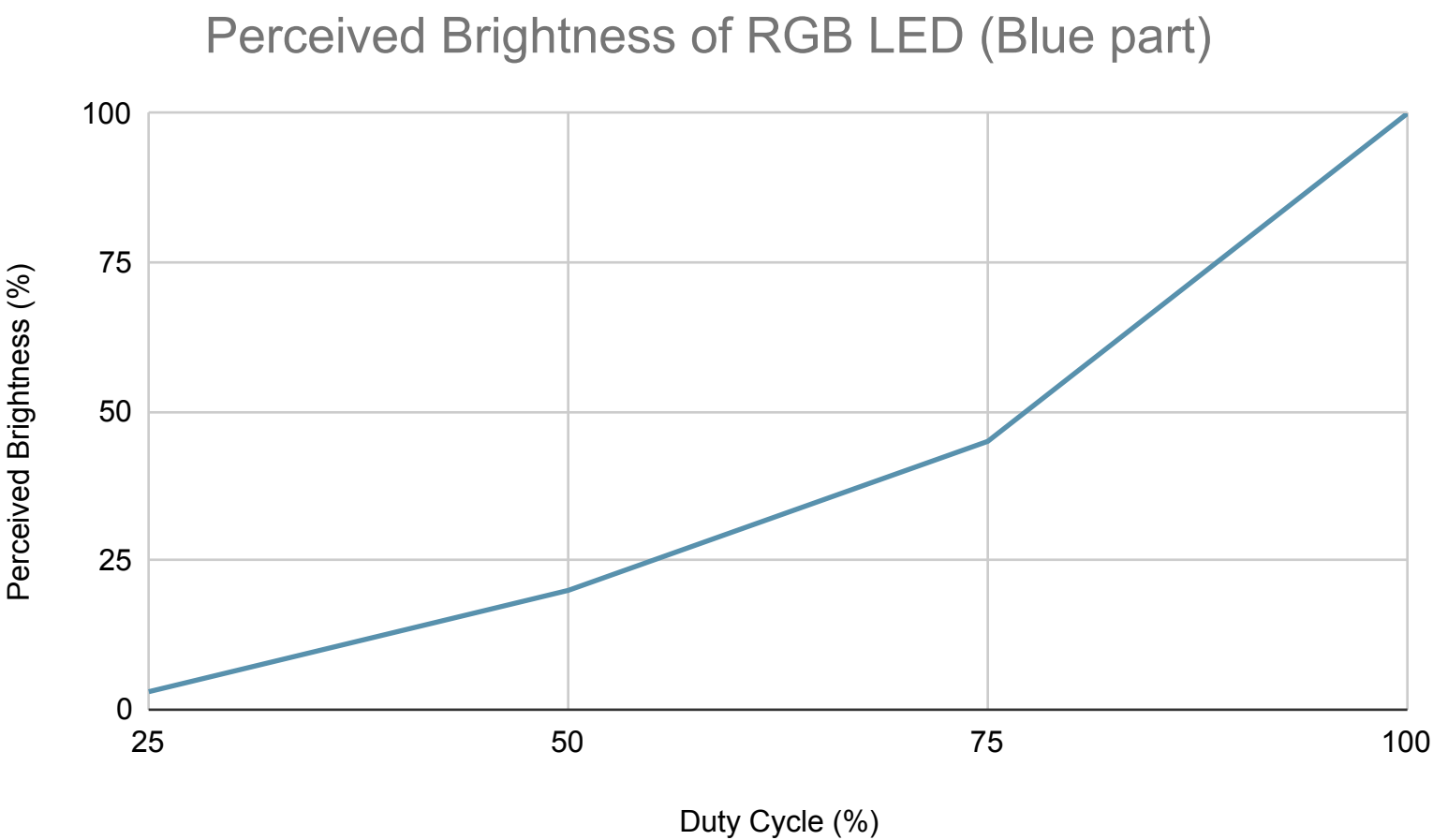
In [ ]:

Assignment 1 Write Up

When it came to designing this assignment, I started out by reading the provided references in the assignment instructions. However, I was initially getting hung up on finding the clock frequency and finding the proper registers and whatnot that the reference from deepbluembedded discussed. Since I figured that I was probably overthinking this project, I went ahead and implemented a basic hardcoded idea of PWM, using 100 Hz as the refresh rate. It ended up working, however the Pynq board was consistently crashing after a few hundred loops of code. After a bit of digging, it turned out that the void functions in the C code cells were causing problems, and that turning them into non-void functions (such as having them return 1 at the end) solved this problem.

From here, I made a setPWM function that passed in an int for duty cycle, and returned an array with the appropriate timings for the high and low parts of a wave period for a 100 Hz refresh rate. These timings were then passed into the time.sleep function. To achieve the 25% - 100% brightness variations, I consulted the other reference from the assignment document, and played around with the values until it looked right to me. I based my numbers off the logarithmic curve from the reference.

For the final part of the assignment, I started off with the asyncio example from the first lab as a base. I then replaced the existing LED logic from that example with the appropriate LED logic for this assignment. For switching colors, I initially planned to have 4 different flags to switch colors. However, I realized that it would be easier to use a global ‘color’ variable that would just be assigned a certain color based on the button press. The color variables would be made as arrays to make accessing their values easier with the write_gpio function. As a result, the numbered ‘flag’ variables went unused here.


Perceived Brightness of RGB LED (Blue part)
100
75
50
25
0
Perceived Brightness (%)
25
50
75
100
Duty Cycle (%)