Assignment 2 Write Up

When it came to designing this assignment, I started out by referring to the previous lab assignments with regards to activating the LEDs. First, I tested out toggling the LEDs on and off across the PYNQ board and on the PMOD for the eventual blinking part of the assignment. However, I got stuck on figuring out how to toggle the PMOD LED and the onboard LEDs in the same function. This is because the PMOD does not have a toggle function for specific bits (at least to my knowledge) in the same manner as the onboard LEDs. I ended up writing separate “toggleOn” and “toggleOff” functions to solve this. These functions work by passing an int from 0-4 inclusive, and if 4 is passed in, they toggle the PMOD LED instead of the onboard ones. After all of this work, I find out that I actually misread the instructions and was meant to use one of the RGB LEDs onboard instead of through the PMOD. I brought this up with the TA, and my revision was approved.

For the main implementation of the assignment, I started off with the non-blocking threading example from the second lab as a base. I then overwrote the existing LED logic from that example with the appropriate LED logic for this assignment. I made sure to incorporate the new toggle functions I made earlier. For the main process function (“philosopher_t” in this case) I made sure to include two non-blocking resources for the “forks” as opposed to one in the original lab code. I’ve also added additional if-else branches that account for only having one or the other fork. In these cases, the process will drop their respective forks in order to prevent a deadlock. In total, I’ve initialized 5 fork variables to emulate the original Dining Philosopher’s problem, and I then manually assigned them to each of the 5 “philosophers.” In addition, to have an easier start, I also manually arranged the order of which the philosophers will pick up their forks to start eating. This was to ensure that there would at least be 2 philosophers eating upon starting the code.

With regards to implementing a button to stop the processes, I initially tried using the asyncio lab as an example for an interrupt. However, I was unsure how to implement asyncio with 5 different threads at once, and so I got stuck for a while. Eventually, I learned that the threading python library also has “events” which could be used instead. This was the solution I ended up going with, as it was a lot easier to understand and implement since I can pass events into the initialization of threads.

The implementation for the random numbers for eating and sleeping was very straightforward since all I had to do was make sure the sleeping time wouldn’t overlap with the eating time. It worked seamlessly with the pre-existing logic.

The Github repository can be found here.

Import Libraries and Initialize LEDS 1-4

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

Initialize PMOD for 5th LED

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;
}
```

Testing LED array

In [3]:
```
for i in range(4):
    base.leds[i].toggle()
    time.sleep(0.5)
    base.leds[i].toggle()
    time.sleep(0.5)
    base.leds[i].on()
    time.sleep(0.5)
    base.leds[i].off()

clear_gpio()
write_gpio(2,1)
time.sleep(0.5)
write_gpio(2,0)
time.sleep(0.5)
write_gpio(2,1)
time.sleep(0.5)
write_gpio(2,0)
```

```
GPIO Cleared!
```

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

In [4]:
```
def toggleOn(n):
    if n >= 0 and n < 4:
        base.leds[n].on()
    elif n == 4:
        write_gpio(2,1)
    else:
        print("Invalid number!")

def toggleOff(n):
    if n >= 0 and n < 4:
        base.leds[n].off()
    elif n == 4:
        write_gpio(2,0)
    else:
        print("Invalid number!")
```

In [8]:
```
clear_gpio()

for i in range(4):
    base.leds[i].off()

while (1):
    toggleOn(0)
    time.sleep(0.1)
    toggleOn(1)
    time.sleep(0.1)
    toggleOn(2)
    time.sleep(0.1)
    toggleOn(3)
    time.sleep(0.1)
    toggleOn(4)
    time.sleep(0.1)

    toggleOff(0)
    time.sleep(0.1)
    toggleOff(1)
    time.sleep(0.1)
    toggleOff(2)
    time.sleep(0.1)
    toggleOff(3)
    time.sleep(0.1)
    toggleOff(4)
    time.sleep(0.1)
```

```
GPIO Cleared!
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Input In [8], in <cell line: 6>()
      6 while (1):
      7     toggleOn(0)
----> 8     time.sleep(0.1)
      9     toggleOn(1)
     10     time.sleep(0.1)

KeyboardInterrupt:
```

In [7]:
```
clear_gpio()

def blink(t, d, n):
    for i in range(t):
        toggleOn(n)
        time.sleep(d)
        toggleOff(n)
        time.sleep(d)
    toggleOff(n)

def philosopher_t(_l1, _l2, num, _event):
    while (not(_event.is_set())):
        res_avail1 = _l1.acquire(False) # this is non-blocking acquire
        res_avail2 = _l2.acquire(False) # this is non-blocking acquire
        if res_avail1 and res_avail2:

            print("Philosopher {} took their forks!".format(num))
            blink(random.randint(10,75), 0.02, num)
            _l1.release()
            _l2.release()
            print("Forks dropped.")
            time.sleep(random.randint(2,3)) # nap time !

        elif res_avail1 and res_avail2 == 0:

            print("Philosopher {} dropped a fork.".format(num))
            blink(random.randint(10,75), 0.02, num)
            _l1.release()
            time.sleep(random.randint(2,3)) # nap time !

        elif res_avail1 == 0 and res_avail2:

            print("Philosopher {} dropped a fork.".format(num))
            blink(random.randint(10,75), 0.02, num)
            _l2.release()
            time.sleep(random.randint(2,3)) # nap time !

        else:
            print("Someone's hungry!.")
            blink(5, 0.1, num)

    print('Philosopher {} is finished.'.format(num))

def buttonStop(_t1, _t2, _t3, _t4, _t5, _event):
    global btns
    while (not(_event.is_set())):
        if btns.read() != 0:
            _event.set()
            print("Stop requested...")
        else:
            time.sleep(0.1)

threads = []
fork1 = threading.Lock()
fork2 = threading.Lock()
fork3 = threading.Lock()
fork4 = threading.Lock()
fork5 = threading.Lock()
```

```
stopButton = threading.Event()

t1 = threading.Thread(target=philosopher_t, args=(fork1, fork2, 0, stopButto
threads.append(t1)
t1.start()

t3 = threading.Thread(target=philosopher_t, args=(fork3, fork4, 2, stopButto
threads.append(t3)
t3.start()

t5 = threading.Thread(target=philosopher_t, args=(fork5, fork1, 4, stopButto
threads.append(t5)
t5.start()

t2 = threading.Thread(target=philosopher_t, args=(fork2, fork3, 1, stopButto
threads.append(t2)
t2.start()

t4 = threading.Thread(target=philosopher_t, args=(fork4, fork5, 3, stopButto
threads.append(t4)
t4.start()

stopFoo = threading.Thread(target=buttonStop, args=(t1, t2, t3, t4, t5, stop
stopFoo.start()
```

```
GPIO Cleared!
Philosopher 0 took their forks!
Philosopher 2 took their forks!
Philosopher 4 dropped a fork.
Someone's hungry!.
Someone's hungry!.
Forks dropped.
Philosopher 1 dropped a fork.
Philosopher 3 dropped a fork.
Forks dropped.
Philosopher 2 took their forks!
Philosopher 0 took their forks!
Philosopher 4 dropped a fork.
Someone's hungry!.
Someone's hungry!.
Forks dropped.
Philosopher 1 dropped a fork.
Philosopher 3 took their forks!
Forks dropped.
Forks dropped.
Philosopher 2 dropped a fork.
Philosopher 4 took their forks!
Someone's hungry!.
Philosopher 0 dropped a fork.
Philosopher 1 dropped a fork.
Philosopher 3 dropped a fork.
Forks dropped.
Philosopher 2 dropped a fork.
Philosopher 1 took their forks!
Philosopher 4 took their forks!
Forks dropped.
Philosopher 2 took their forks!
Philosopher 0 dropped a fork.
Someone's hungry!.
Forks dropped.
```

```
Forks dropped.
Philosopher 3 took their forks!
Philosopher 1 took their forks!
Forks dropped.
Forks dropped.
Philosopher 0 took their forks!
Philosopher 4 dropped a fork.
Philosopher 2 took their forks!
Someone's hungry!.
Forks dropped.
Forks dropped.
Philosopher 3 dropped a fork.
Philosopher 1 took their forks!
Someone's hungry!.
Forks dropped.
Philosopher 0 took their forks!
Philosopher 2 took their forks!
Forks dropped.
Philosopher 1 dropped a fork.
Philosopher 4 took their forks!
Forks dropped.
Philosopher 3 dropped a fork.
Forks dropped.
Philosopher 0 dropped a fork.
Philosopher 2 took their forks!
Forks dropped.
Philosopher 4 took their forks!
Philosopher 3 dropped a fork.
Philosopher 1 took their forks!
Someone's hungry!.
Someone's hungry!.
Forks dropped.
Philosopher 2 took their forks!
Forks dropped.
Philosopher 0 took their forks!
Forks dropped.
Forks dropped.
Philosopher 1 took their forks!
Philosopher 4 took their forks!
Stop requested...
Philosopher 3 is finished.
Forks dropped.
Forks dropped.
Philosopher 2 is finished.
Philosopher 0 is finished.
Philosopher 1 is finished.
Philosopher 4 is finished.
```