WES 237A – Assignment 4.1, Alarm System

Ricardo Lizarraga, rlizarraga0@gmail.com , *619.252.4157* | **PID**: **A69028483**

Assignments Rubric

yes	Report Submitted?	02/23/2024
yes	Video Uploaded?	https://drive.google.com/drive/folders/1Zl5pJz2B-nb9GyvVAljwRYwRe7JzeUzl?usp=sharing Assignm4_1_PYNQA(server)_PYNQB(client).mp4 {***Client #2 connects to Server #1***} Assignm4_1_PYNQB(server)_PYNQA(client).mp4 {***Client #1 connects to Server#2***}
yes	Pushed to Github?	https://github.com/RiLizarraga/WES237A_Assign4.1
ok	Does the video demonstration show correct execution?	***There're 2 videos to demonstrate 2 different remote connection configurations:*** 1. ***Client #1 connects to Server#2*** 2. ***Client #2 connects to Server #1***
ok	Is the submitted code correct?	Source code, documentation and readme, public
ok	How well does the report outlines the design of the code?	See Solution Architecture diagram and algorithm
ok	How well does the report describe the results?	End to end Connectivity is well defined in report
yes	Does the Report detail the student's grasp on the goals/objectives of the assignment?	The learning objectives for this assignment are • Run a server and client on the same machine on separate processes • Connect to and disconnect from a remote server • Communicate button presses between PYNQ boards.

The learning objectives for this assignment are
• Run a server and client on the same machine on separate processes
• Connect to and disconnect from a remote server
• Communicate button presses between PYNQ boards.

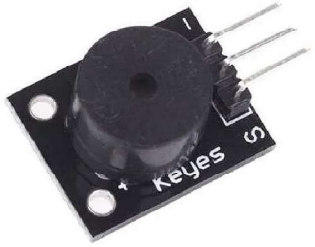
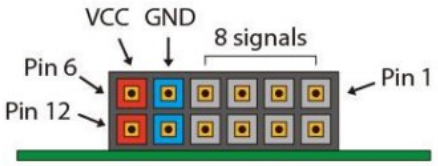


Figure 1: Buzzer module

The buzzer module works as follows

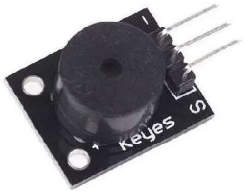
• The **'-'** pin connects to gnd.
• The '+' pin is the signal you want to write.
• The middle pin is the power (3.3V)
• Writing a square wave (1, 0, 1, 0, 1, 0, etc) alternating high/low, will generate a tone at the
given frequency. Psuedocode looks like this
– while we want a tone
    * write gpio value high
    * sleep for 1/(2 * tone_freq)
    * write gpio_value low
    * sleep for 1/(2 * tone_freq)



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

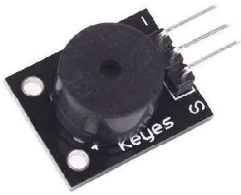
BUZZER A

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



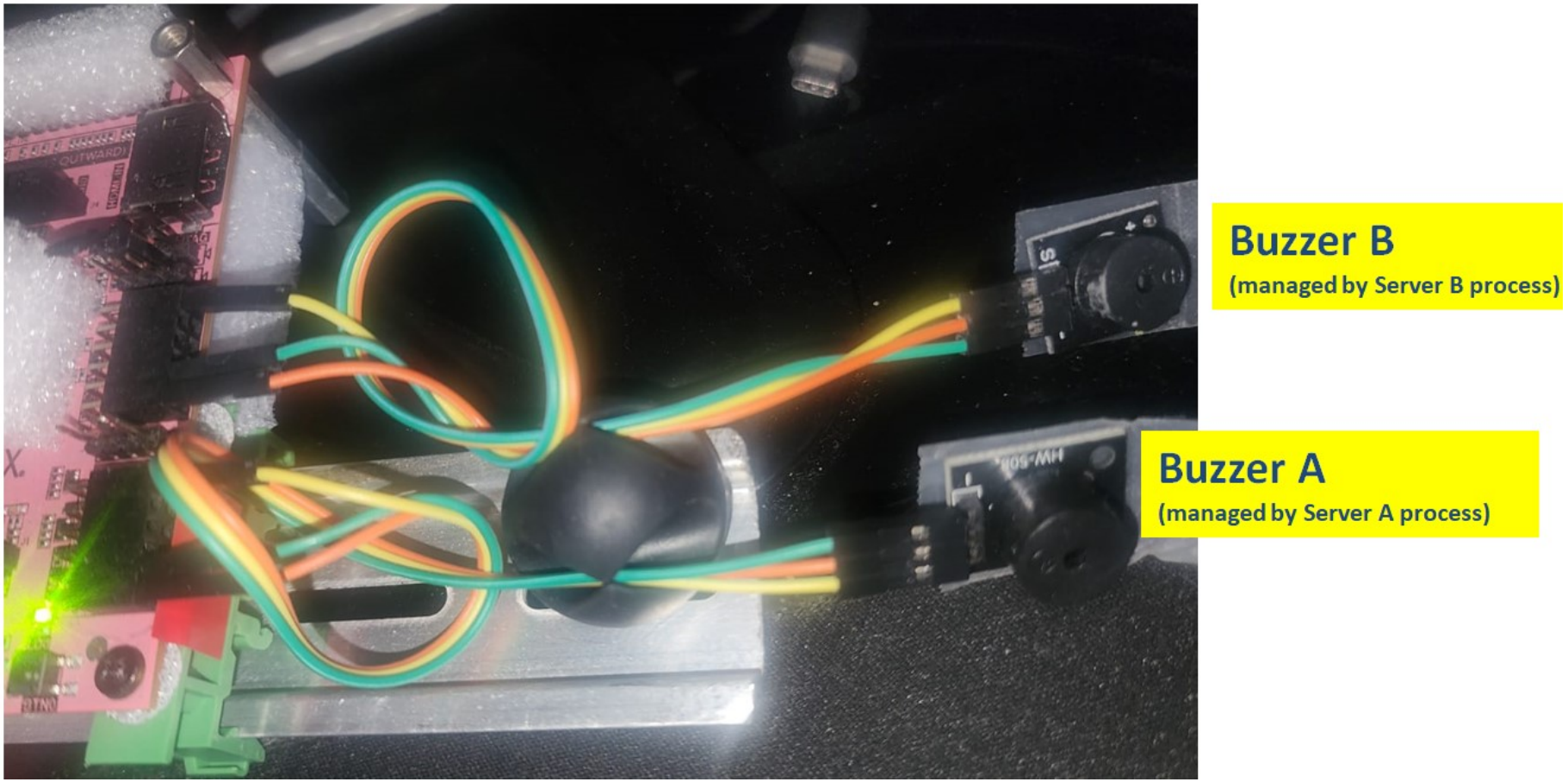
BUZZER B:

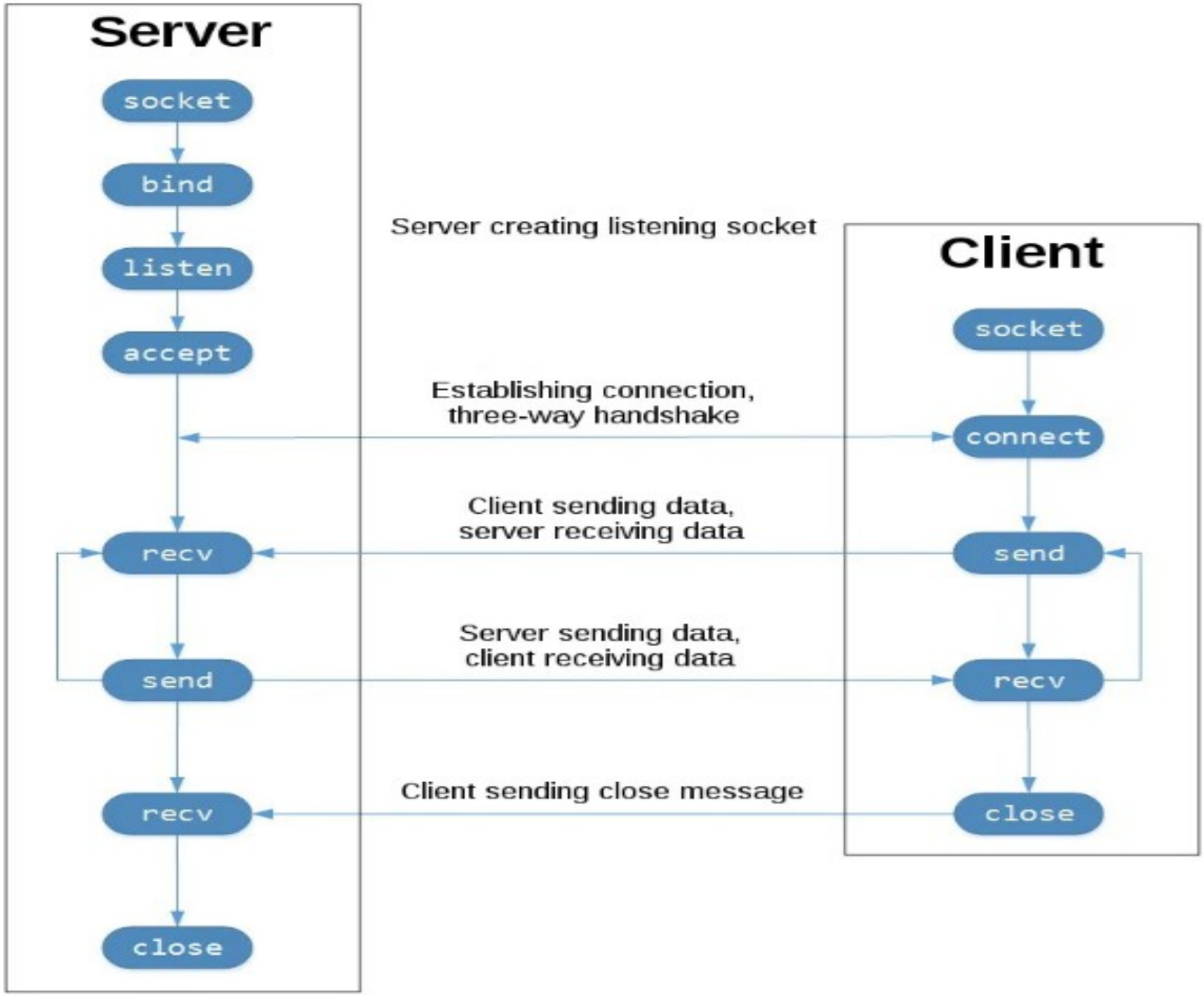
PMODA	RGB Led Module
Pin#1 **GPIO 0**	- Signal
Pin#2 GPIO 1	-
Pin#3 GPIO 2	-
Pin#4 GPIO 3	-
Pin#5 GND	- (-)
Pin#6 Vcc	- (+)



The communication part requires the following:

- Using **multiprocessing** library, create **two processes**: **one process for server** and **one process for client.**
- Each board will need to have the following happening.
  - The server process should always be running in listening mode.
  - By pushing one of the buttons on the PYNQ board, the client has to start and connect to the server board.
  - After client connects, pressing a different button should emit a tone on the other PYNQ board.* Pushing the button should emit a ~0.5 second tone each time it's pressed.
  - By pushing a third button, the client board disconnects from the server. This will end the communication and both the server and client will terminate




Server
socket
bind
listen
accept
recv
send
recv
close
Server creating listening socket
Establishing connection,
three-way handshake
Client sending data,
server receiving data
Server sending data,
client receiving data
Client sending close message
Client
socket
connect
send
recv
close

237A – Assignment 4.1, Alarm System, Block Diagram

Client #1 connects to Server#2
Client #2 connects to Server #1

PYNQ #A

Multiprocessing library

PYNQ #B

{process 1} Server #A

Listening @ *socket:* ***4944***

0.- The server process should always be running in listening mode

Resources

Buzzer A
push button 0

Algorithm

1.- By pushing **push button 0** on the PYNQ board, the client has to start and Connect to Server #A.

3.- After receiving Client #B msg1, emit a tone on **(buzzer #1)**. (a ~0.5 second tone each time it's pressed.)

{process 3} Server #B

Listening @ *socket:* ***4945***

0.- The server process should always be running in listening mode

Resources

Buzzer B
push button 2

Algorithm

1.- By pushing **push button 2** on the PYNQ board, the client has to start and Connect to Server #B.

3.- After receiving Client #A msg1, emit a tone on **(buzzer 2)**. (a ~0.5 second tone each time it's pressed.)

{process 2} Client #A

Resources

push button 1

Algorithm

2.- After client connects, pressing **push button 3** {transmit cmd1 to Server #B}

4.- By toggling ***slide switch 0***, the Client #A board disconnects from the server #B. This will end the **communication** and **both** the **server** and **client** will terminate

{process 4} Client #B

Resources

push button 1

Algorithm

2.- After client connects, pressing **push button 1** {transmit cmd1 to Server #A}

4.- By toggling ***slide switch 1***, the Client #B board disconnects from the server #A. This will end the **communication** and **both** the **server** and **client** will terminate

These steps should be completed for each version of **PYNQ #1** -> **PYNQ #2**. This means pushing a **button 0** on **PYNQ #1** will emit a tone on **PYNQ #2** as well as pushing a button on **PYNQ #2** (**button 1**) will emit a tone on **PYNQ #1**.
– Both directions of communication should be able to operate at the same time.

User interaction

Btn0 – “Server A”, launches a New Process “Client B”

Btn1 – “Client B”, Transmits “cmd1” to “Server A” to emit on “Buzzer A”

Btn2 - “Server B”, launches a New Process “Client A”

Btn3 – “Client A”, Transmits “cmd1” to “Server B” to emit on “Buzzer B”

Switch 0 – Toggle switch for “Client A”, Transmit “exit” to “Server B” to CLOSE All

Switch 1–Toggle switch for “Client B”, Transmit “exit” to “Server A” to CLOSE All

- These steps should be completed for each version of PYNQ1 -> PYNQ2. This means pushing a button on PYNQ1 will emit a tone on PYNQ2 as well as pushing a button on PYNQ2 will emit a tone on PYNQ2.

– Both directions of communication should be able to operate at the same time.

Deliverables

Each student must submit the following **individually**

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

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

In [ ]:
```
#######################  Start here  ###################################
from IPython.display import display, HTML; display(HTML("<style>.container { width:100
import threading;import time;import pynq.lib.rgbled as rgbled;from pynq.overlays.base
import socket;import subprocess;import platform
base = BaseOverlay("base.bit")
```

In [ ]:
```
%%microblaze base.PMODB
#include "gpio.h"
#include "pyprintf.h"
void write_gpioB(unsigned int pin, unsigned int val){//Function to turn on/off a selec
    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
}
unsigned int read_gpioB(unsigned int pin){//Function to read the value of a selected p
    gpio pin_in = gpio_open(pin);gpio_set_direction(pin_in, GPIO_IN);return gpio_read(
}
```

In [ ]:
```
%%microblaze base.PMODA
#include "gpio.h"
#include "pyprintf.h"
void write_gpioA(unsigned int pin, unsigned int val){//Function to turn on/off a selec
    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
}
unsigned int read_gpioA(unsigned int pin){//Function to read the value of a selected p
    gpio pin_in = gpio_open(pin);gpio_set_direction(pin_in, GPIO_IN);return gpio_read(
}
```

In [ ]:
```
## Multiprocessing, sharing information between processes, this is simple form
import multiprocessing, os;from multiprocessing import Process, Lock;
from multiprocessing.sharedctypes import Value, Array
from ctypes import Structure, c_double;import ctypes;from array import *
### CONSTANTS ###
SERVER, LOW, HIGH = 0, 0, 1;TONE_FREQ1, TONE_FREQ2 = 400, 100
DEBUG, OPC = True, 1;ETH_ENDL = "\r\n";Parser_Thr_flag = True
threads_all_flag, Parser_Thr_flag, btn0_flag, btn1_flag, btn2_flag, btn3_flag = True,
threads = [];arg1 = arg2 = ""
procs = []
def buzzer():
    global SERVER
    tone_freq = TONE_FREQ1 if SERVER == 0 else TONE_FREQ2
    start = time.time();print(start)
    for i in range (1000):
        if SERVER == 0:
            write_gpioA(0, LOW)
        else:
            write_gpioB(0, LOW)
        time.sleep(1/(2*tone_freq))
        if SERVER == 0:
            write_gpioA(0, HIGH)
        else:
            write_gpioB(0, HIGH)
        time.sleep(1/(2*tone_freq))
        checktime = time.time()
        if checktime-start > 0.5:break# just 0.5 sec duration
    return 1
def worker_thr_parser():#[_l]: threading lock (resource), [num]: index representing th
```

```
    global cmd,arg1,arg2,DEBUG,OPC, threads_all_flag, Parser_Thr_flag, dac, adc, leds,
    while threads_all_flag:
        try:
            global thr_adcwf,Parser_Thr_flag
            s= socket.socket(socket.AF_INET,socket.SOCK_STREAM)
            s.bind(('',4944)) #bind to all ip in the range
            s.listen(1) #listen for 1 connection
            Parser_Thr_flag = True
            while Parser_Thr_flag:
                try:
                    connection, client_address = s.accept();print ("connection from :'
                except Exception as inst:
                    s.shutdown(socket.SHUT_RDWR)
                    s.close()
                    Parser_Thr_flag = False
                    break
                while Parser_Thr_flag:
                    try:
                        connection.send(bytes(""+ETH_ENDL, "utf-8"))#connection.send(b
                        msg = connection.recv(8192).decode();msg = msg.upper();msg=msg
                    except Exception as inst:
                        s.shutdown(socket.SHUT_RDWR)
                        s.close()
                        Parser_Thr_flag = False
                        break

                    msg = msg.split()
                    cmd = arg1 = arg2 = ''
                    if (len(msg)>0):cmd = msg[0]
                    if (len(msg)>1):arg1=msg[1]
                    if (len(msg)>2):arg2=msg[2]
                    if (DEBUG):print(cmd+','+arg1+','+arg2)
                    if cmd == "CMD1":
                        ret = buzzer()
                        connection.send(bytes(str(cmd+" = "+str(ret))+ETH_ENDL, 'utf-8
                    elif cmd == "EXIT" or Parser_Thr_flag == False or threads_all_flag
                        connection.send(bytes("Adios"+ETH_ENDL, 'utf-8'))
                        time.sleep(1)
                        s.shutdown(socket.SHUT_RDWR)
                        Parser_Thr_flag = False
                        time.sleep(0.3)
                        #s.close()
                        break
                    else:
                        time.sleep(0.3)
        finally:
            s.close()
        time.sleep(2)

def run_client(_btn0,_btn1,_btn2,_btn3,_sw0,_sw1):#Client B     ,sw0_state,sw1_state
    CLIENT = 'B';ETH_ENDL = "\r\n"
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_ip = "127.0.0.1"  # replace with the server's IP address
    server_port = 4944  # replace with the server's port number
    client.connect((server_ip, server_port))
    sw_states = []
    val = _sw0.read();sw_states.append(val)
    val = _sw1.read();sw_states.append(val)
    print("Client: Initial switches states : "+str(sw_states[0])+str(sw_states[1]))
    print("Client " + CLIENT+"is connected")
```

```
    threads_all_flag = True
    while threads_all_flag:#valid commands: "cmd1" "exit"
        time.sleep(0.2)
        if (_btn1.read()==1):
            msg = "cmd1"+ETH_ENDL
            print("Client " + CLIENT+" transmitted: "+msg)
            client.send(msg.encode("utf-8")[:1024])
            time.sleep(0.2)
            response = client.recv(1024)# receive message from the server
            response = response.decode("utf-8")
            print("Response by Server: "+response.rstrip())
            # if server sent us "closed" in the payload, we break out of the loop and
            msg = msg.upper();
            if response.lower() == "closed":
                break
        if (_sw1.read()!=sw_states[1]):
            msg = "exit"+ETH_ENDL
            print("Client " + CLIENT+" transmitted: "+msg)
            client.send(msg.encode("utf-8")[:1024])
            time.sleep(0.2)
            response = client.recv(1024)# receive message from the server
            response = response.decode("utf-8")
            print("Response by Server: "+response.rstrip())
            # if server sent us "closed" in the payload, we break out of the loop and
            msg = msg.upper();
            if response.lower() == "closed" or msg=='EXIT'+ETH_ENDL:
                threads_all_flag = False
                break

    # close client socket (connection to the server)
    client.close()
    print("Connection to server closed")

async def get_btns(_loop):
    btns = base.btns_gpio
    switches = base.switches_gpio
    global DEBUG,threads_all_flag, Parser_Thr_flag, leds, btn0_flag, btn1_flag, btn2_f
    clientLaunched_flag = False
    while threads_all_flag:
        await asyncio.sleep(0.1)
        if (btns[0].read()==1 and clientLaunched_flag==False):#Launch client #A
            clientLaunched_flag = True
            await asyncio.sleep(0.3)
            print("button 0 pressed")
            time.sleep(0.2)
            btn0_flag = True
            print("Launching new process Client B")
            p = multiprocessing.Process(target=run_client, args=(btns[0],btns[1],btns[
            p.start()
            os.system("taskset -p {}".format(p.pid)) # taskset is an os command to pir
            procs.append(p)
            print('Process: {}, PID: {} Started'.format(p.name, p.pid))
            time.sleep(2)

def add(_i, a, b, returnValuePtr):# _i : Index of the process, a, b : Integers to add,
    returnValuePtr[_i] = a+b
##### Start here ######
print("User interaction(Buttons and Switches):\n   \
       0 - Server A, launches a New Process Client B\n   \
       1 - Client B, Transmits “cmd1” to Server A to emit on Buzzer A \n   \
```

```
        2 - Server B, launches a New Process Client A\n   \
        3 - Client A, Transmits “cmd1” to Server B to emit on Buzzer B\n  \
        Switch 0-Toggle switch for  Client A, Transmit exit to Server B to CLOSE All\r
        Switch 1-Toggle switch for  Client B, Transmit exit to Server A to CLOSE All\r
### Create needed Threads (don't start them up yet)
thr1 = threading.Thread(target=worker_thr_parser, args=())
threads.append(thr1)
for thr in threads:### Start Threads
    thr.start()

loop = asyncio.new_event_loop();  # Instance event_loop object
loop.create_task(get_btns(loop)); # take user input buttons

loop.run_forever()
loop.close()
### Stop Live Threads, after  commanded by user buttons
for thr in threads:
    thr.join()
    print('{} joined'.format(t.name))
```

In [ ]:
```
#######################  Start here  ###################################
from IPython.display import display, HTML; display(HTML("<style>.container { width:100
import threading;import time;import pynq.lib.rgbled as rgbled;from pynq.overlays.base
import socket;import subprocess;import platform
base = BaseOverlay("base.bit")
```

In [ ]:
```
%%microblaze base.PMODB
#include "gpio.h"
#include "pyprintf.h"
void write_gpioB(unsigned int pin, unsigned int val){//Function to turn on/off a selec
    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
}
unsigned int read_gpioB(unsigned int pin){//Function to read the value of a selected p
    gpio pin_in = gpio_open(pin);gpio_set_direction(pin_in, GPIO_IN);return gpio_read(
}
```

In [ ]:
```
%%microblaze base.PMODA
#include "gpio.h"
#include "pyprintf.h"
void write_gpioA(unsigned int pin, unsigned int val){//Function to turn on/off a selec
    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
}
unsigned int read_gpioA(unsigned int pin){//Function to read the value of a selected p
    gpio pin_in = gpio_open(pin);gpio_set_direction(pin_in, GPIO_IN);return gpio_read(
}
```

In [ ]:
```
## Multiprocessing, sharing information between processes, this is simple form
import multiprocessing, os;from multiprocessing import Process, Lock;
from multiprocessing.sharedctypes import Value, Array
from ctypes import Structure, c_double;import ctypes;from array import *
SERVER, LOW, HIGH = 1, 0, 1;TONE_FREQ1, TONE_FREQ2 = 400, 100
DEBUG, OPC = True, 1;ETH_ENDL = "\r\n";Parser_Thr_flag = True
threads_all_flag, Parser_Thr_flag, btn0_flag, btn1_flag, btn2_flag, btn3_flag = True,
threads = [];arg1 = arg2 = ""
procs = []
def buzzer():
    global SERVER
    tone_freq = TONE_FREQ1 if SERVER == 0 else TONE_FREQ2
    start = time.time();print(start)
    for i in range (1000):
        if SERVER == 0:
            write_gpioA(0, LOW)
        else:
            write_gpioB(0, LOW)
        time.sleep(1/(2*tone_freq))
        if SERVER == 0:
            write_gpioA(0, HIGH)
        else:
            write_gpioB(0, HIGH)
        time.sleep(1/(2*tone_freq))
        checktime = time.time()
        if checktime-start > 0.5:break# just 0.5 sec duration
    return 1
def worker_thr_parser():#[_l]: threading lock (resource), [num]: index representing th
    global cmd,arg1,arg2,DEBUG,OPC, threads_all_flag, Parser_Thr_flag, dac, adc, leds,
```

```
    while threads_all_flag:
        try:
            global thr_adcwf,Parser_Thr_flag
            s= socket.socket(socket.AF_INET,socket.SOCK_STREAM)
            s.bind(('',4945)) #bind to all ip in the range
            s.listen(1) #listen for 1 connection
            Parser_Thr_flag = True
            while Parser_Thr_flag:
                try:
                    connection, client_address = s.accept();print ("connection from :"
                except Exception as inst:
                    s.shutdown(socket.SHUT_RDWR)
                    s.close()
                    Parser_Thr_flag = False
                    break
                while Parser_Thr_flag:
                    try:
                        connection.send(bytes(""+ETH_ENDL, "utf-8"))#connection.send(b
                        msg = connection.recv(8192).decode();msg = msg.upper();msg=msg
                    except Exception as inst:
                        s.shutdown(socket.SHUT_RDWR)
                        s.close()
                        Parser_Thr_flag = False
                        break

                    msg = msg.split()
                    cmd = arg1 = arg2 = ''
                    if (len(msg)>0):cmd = msg[0]
                    if (len(msg)>1):arg1=msg[1]
                    if (len(msg)>2):arg2=msg[2]
                    if (DEBUG):print(cmd+','+arg1+','+arg2)
                    if cmd == "CMD1":
                        ret = buzzer()
                        connection.send(bytes(str(cmd+" = "+str(ret))+ETH_ENDL, 'utf-8
                    elif cmd == "EXIT" or Parser_Thr_flag == False or threads_all_flag
                        connection.send(bytes("Adios"+ETH_ENDL, 'utf-8'))
                        time.sleep(1)
                        s.shutdown(socket.SHUT_RDWR)
                        Parser_Thr_flag = False
                        time.sleep(0.3)
                        #s.close()
                        break
                    else:
                        time.sleep(0.3)
        finally:
            s.close()
        time.sleep(2)

def run_client(_btn0,_btn1,_btn2,_btn3,_sw0,_sw1):#Client B     ,sw0_state,sw1_state
    CLIENT = 'A';ETH_ENDL = "\r\n"
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_ip = "127.0.0.1"  # replace with the server's IP address
    server_port = 4945  # replace with the server's port number
    client.connect((server_ip, server_port))
    sw_states = []
    val = _sw0.read();sw_states.append(val)
    val = _sw1.read();sw_states.append(val)
    print("Client: Initial switches states : "+str(sw_states[0])+str(sw_states[1]))
    print("Client " + CLIENT+"is connected")
    threads_all_flag = True
```

```
    while threads_all_flag:#valid commands: "cmd1" "exit"
        time.sleep(0.2)
        if (_btn3.read()==1):
            msg = "cmd1"+ETH_ENDL
            print("Client " + CLIENT+" transmitted: "+msg)
            client.send(msg.encode("utf-8")[:1024])
            time.sleep(0.2)
            response = client.recv(1024)# receive message from the server
            response = response.decode("utf-8")
            print("Response by Server: "+response.rstrip())
            # if server sent us "closed" in the payload, we break out of the loop and
            msg = msg.upper();
            if response.lower() == "closed":
                break
        if (_sw0.read()!=sw_states[0]):
            msg = "exit"+ETH_ENDL
            print("Client " + CLIENT+" transmitted: "+msg)
            client.send(msg.encode("utf-8")[:1024])
            time.sleep(0.2)
            response = client.recv(1024)# receive message from the server
            response = response.decode("utf-8")
            print("Response by Server: "+response.rstrip())
            # if server sent us "closed" in the payload, we break out of the loop and
            msg = msg.upper();
            if response.lower() == "closed" or msg=='EXIT'+ETH_ENDL:
                threads_all_flag = False
                break

    # close client socket (connection to the server)
    client.close()
    print("Connection to server closed")

async def get_btns(_loop):
    btns = base.btns_gpio
    switches = base.switches_gpio
    global DEBUG,threads_all_flag, Parser_Thr_flag, leds, btn0_flag, btn1_flag, btn2_f
    clientLaunched_flag = False
    while threads_all_flag:
        await asyncio.sleep(0.1)
        if (btns[2].read()==1 and clientLaunched_flag==False):#Launch client #A
            clientLaunched_flag = True
            await asyncio.sleep(0.3)
            print("button 2 pressed")
            time.sleep(0.2)
            btn0_flag = True
            print("Launching new process Client A")
            p = multiprocessing.Process(target=run_client, args=(btns[0],btns[1],btns[
            p.start()
            os.system("taskset -p {}".format(p.pid)) # taskset is an os command to pir
            procs.append(p)
            print('Process: {}, PID: {} Started'.format(p.name, p.pid))
            time.sleep(2)


def add(_i, a, b, returnValuePtr):# _i : Index of the process, a, b : Integers to add,
    returnValuePtr[_i] = a+b
##### Start here ######
print("User interaction(Buttons and Switches):\n   \
        0 - Server A, launches a New Process Client B\n   \
        1 - Client B, Transmits “cmd1” to Server A to emit on Buzzer A \n   \
```

```
        2 - Server B, launches a New Process Client A\n   \
        3 - Client A, Transmits “cmd1” to Server B to emit on Buzzer B\n  \
        Switch 0-Toggle switch for  Client A, Transmit exit to Server B to CLOSE All\r
        Switch 1-Toggle switch for  Client B, Transmit exit to Server A to CLOSE All\r
### Create needed Threads (don't start them up yet)
thr1 = threading.Thread(target=worker_thr_parser, args=())
threads.append(thr1)
for thr in threads:### Start Threads
    thr.start()

loop = asyncio.new_event_loop();  # Instance event_loop object
loop.create_task(get_btns(loop)); # take user input buttons

loop.run_forever()
loop.close()
### Stop Live Threads, after  commanded by user buttons
for thr in threads:
    thr.join()
    print('{} joined'.format(t.name))
```