WES-237A Final Project - Contactless Health Monitoring

David Rodriguez & Yaqoob Hanona

Group 2

11 March 2023

Part 1: Importing Libraries

In [1]:

```python
import time
import serial
import socket

import subprocess
import multiprocessing
import pynq.lib.rgbled as rgbled
from pynq.overlays.base import BaseOverlay
from pynq.lib import MicroblazeLibrary


from pynq.lib.arduino import Arduino_Analog
from pynq.lib.arduino import ARDUINO_GROVE_A1
from pynq.lib.arduino import arduino_io

base = BaseOverlay("base.bit")
analog1 = Arduino_Analog(base.ARDUINO,ARDUINO_GROVE_A1)

btns = base.buttons[0:3] # Define list containing all built-in Buttons
```

Part 2: Microblaze Functions

In [2]:

```c
%%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);
}


// Reset GPIOS function - Resets all PMODB GPIOs (0-7) to Logic LOW
void resetGPIOS() {
    for (int pin=0; pin < 7; pin++){ // Iterate for all PMODB GPIO Pins
        write_gpio(pin, 0x00); // calling user defined function
    }
}


void setGreen_HIGH(){
    write_gpio(2, 0x01);
}

void setGreen_LOW(){
    write_gpio(2, 0x00);
}

void setRed_HIGH(){
    write_gpio(3, 0x01);
}

void setRed_LOW(){
    write_gpio(3, 0x00);
}
```

Part 3: Reset GPIOs

In [3]:
```
resetGPIOS() # Reset all GPIOs
```

In [4]:
```
setGreen_LOW() # Turn Green LED Off
setRed_LOW() # Turn Red LED Off
```

Part 4: Defining Multiprocessing Processes

In [5]:
```
# WES237A Final Project - Multiprocessing Processes

# Define Multiprocssing Manager List
manager = multiprocessing.Manager()
seq_complete_list = manager.list()

setRed_HIGH() # Turn Red LED On = "Not Connected"

serverName = 'ec2-35-90-59-237.us-west-2.compute.amazonaws.com' # AWS Server Public DNS Domain
testUDP_Port = 1113 # TEST UDP Port

clientPort = 11114 # TCP Client Port
port = 11115 # AWS Server TCP Listening Port


def Presence_Detect(): # Gets Presence Detection status from Radar Sensor GP1 Pin voltage,
    # then reports status to AWS Server via UDP Connection
    serverName = 'ec2-35-90-59-237.us-west-2.compute.amazonaws.com' # AWS Server Public DNS
    UDPclientPort = 1118
    clientSocket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
    while True:
        analog1.read()
        bits = analog1.read('raw')[1]
        if bits >= 1000:
            message = '1'
        else:
            message = '0'
        time.sleep(0.05)
        print('sending message: {}'.format(message))
        # Connect to AWS Server UDP Port and Send Data
        clientSocket.sendto(message.encode(),(serverName, UDPclientPort))
        time.sleep(9)
        if len(seq_complete_list) > 0:
            break
    clientSocket.close()


def TCP_Online_Status(): # Report Online Status to AWS Server via TCP Connection
    # Create a TCP/IP socket
    c_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    print('Connecting PYNQ Client. . .')

    # Connect Client
    c_sock.connect((serverName, clientPort))
    print('Connected to AWS Server!')

    setRed_LOW() # Turn Red LED Off
    setGreen_HIGH() # Turn Green LED On

    upMessage = 'up'
    downMessage = 'down'

    while True:
        setGreen_HIGH()
        time.sleep(8)
        c_sock.send(upMessage.encode())

        if len(seq_complete_list) > 0:
            setGreen_LOW()
            setRed_HIGH()
            # Send AWS TCP Server Offline Message
            c_sock.send(downMessage.encode())
            break
    # Disconnect from Client
    c_sock.close()
    print('Client Connection Socket Closed!')
```

```python
def radar():
    presence_detect = 0
    while True:
        serMsg = serial.Serial('/dev/ttyUSB0', 115200, timeout=2)
        #serMsg.flushInput()

        # Read Radar Data from serial port
        ch = serMsg.read(15).hex() # 15 bytes = max frame size for radar module

        FRAME_HEADER = '5359'
        END_OF_FRAME = '5443'

        pt1 = ch.split(FRAME_HEADER)[1]
        pt2 = pt1.split(END_OF_FRAME)[0]

        control_word = pt2[0:2]
        command_word = pt2[2:4]
        data_len = int(pt2[4:8])
        data = pt2[8:(9+data_len)]

        value_list = []

        # Segment Data based on individual Bytes
        if data_len == 1:
            value = int(data, base=16)
            value_list.append(value)
        else:
            for i in range(0, data_len):
                value_segment = str(data[i]) + str(data[i+1])
                value = int(value_segment, base=16)
                value_list.append(value)

        # Heart Rate Value
        if control_word == '85':
            measurement_type = 'HR' # Heart Rate
            if command_word == '02':
                print('Heart Rate Values: ', value_list)
                # Connect to AWS Server UDP Port and Send Data
                clientSocket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
                clientSocket.sendto(str(value_list[0]).encode(),(serverName, 1115))
                clientSocket.close()
            elif command_word == '05':
                print('Heart Rate Waveform: ', value_list)

        # Respiratory Intensity Value
        if control_word == '81':
            measurement_type = 'RM' # Respiratory Monitoring
            if command_word == '02':
                print('Breathing Waveform: ', value_list)
                # Connect to AWS Server UDP Port and Send Data
                clientSocket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
                clientSocket.sendto(str(value_list[0]).encode(),(serverName, 1120))
                clientSocket.close()

            elif command_word == '01':
                print('Breathing Values: ', value_list)

        # Human Presence Status, Distance, and Movement Info
        if control_word == '80':
            measurement_type = 'HP' # Human Presence Info
            movement = ''

            # Human Presence Status
            # NOTE: Although this is redundant to include since Presence_Detect() function fulfills the same purpose,
            # this UDP transmission is only enacted if the Human Presence parameter is detected in the Raw Radar data stream
            if command_word == '01':
                # Presence Status has changed
                if value == 1:
                    print('Presence Detected!')
                    message = '1'
                else:
                    message = '0'
                    print('Presence Not Detected!')
                clientSocket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
                clientSocket.sendto(message.encode(),(serverName, 1118))
                clientSocket.close()

            # Human Movement Status
            if command_word == '02':
                print('Human Movement Info: ', value_list)
                if (int(value_list[0]) ==  2):
                    movement = 'Active'
                    movement_status = '02'
                elif (int(value_list[0]) ==  1):
                    movement = 'Stationary'
                    movement_status = '01'
                else:
                    movement = 'None'
                    movement_status = '00'
                if (len(value_list) == 0):
                    movement = 'None'
                    movement_status = '00'
                print('Movement Status: ', movement)
                # Connect to AWS Server UDP Port and Send Data
                clientSocket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
                clientSocket.sendto(movement_status.encode(),(serverName, 1119))
                clientSocket.close()

            # Radar Distance Value
            if command_word == '04':
                print('Human Distance Info (cm): ', int(pt2[10:12],base=16))
                # Connect to AWS Server UDP Port and Send Data
                clientSocket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
                clientSocket.sendto(str(int(pt2[10:12],base=16)).encode(),(serverName, 1116))
                clientSocket.close()

        # Send Raw Data from Radar Module
        raw_data = FRAME_HEADER + pt2 + END_OF_FRAME
        # Connect to AWS Server UDP Port and Send Raw Data
        clientSocket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
        clientSocket.sendto(str(raw_data).encode(),(serverName, 1121))
        clientSocket.close()

        serMsg.close()
        time.sleep(0.005)

        if len(seq_complete_list) > 0:
            break
```

Part 5: Health Monitoring Program

In [6]:

```
# Define Main Program

def main():
    setRed_HIGH() # Turn Red LED On

    print('Press Button to initialize Health Monitoring System')
    while True:
        if btns.read() != 0:
            time.sleep(0.5)
            # Initialize Processes
            print('Initializing Processes')
            break

    # Create 3 Processes
    p0 = multiprocessing.Process(target=radar)
    p1 = multiprocessing.Process(target=TCP_Online_Status)
    p2 = multiprocessing.Process(target=Presence_Detect)

    # Start all Processes
    p0.start()
    p1.start()
    p2.start()

    while True:
        if btns.read() != 0:
            time.sleep(0.5)
            seq_complete_list.append(1)
            setRed_LOW() # Red LED off
            setGreen_LOW() # Green LED off
            break

    time.sleep(2) # Allow sufficient time for socket connections to close

    # Join & Close all Processes
    p0.join()
    p0.close()
    p1.join()
    p1.close()
    p2.join()
    p2.close()

    setRed_LOW() # Red LED off
    setGreen_LOW() # Green LED off

    print('Sequence Stopped!')

if __name__ == '__main__':
    main()
```

```
Press Button to initialize Health Monitoring System
Initializing Processes
Connecting PYNQ Client. . .
sending message: 0
Connected to AWS Server!
Heart Rate Values:  [81]
Human Distance Info (cm):  35
Human Movement Info:  [1]
Movement Status:  Stationary
Human Distance Info (cm):  35
Human Movement Info:   [2]
Movement Status: Active
Breathing Waveform:  [9]
Heart Rate Values:  [83]
Human Distance Info (cm):  35
sending message: 0
Heart Rate Values:  [83]
Breathing Waveform:  [9]
Human Distance Info (cm):
 35Heart Rate Values:  [83]
Human Distance Info (cm):  35
Heart Rate Values:  [83]
Breathing Waveform:
 [9]Human Distance Info (cm):
 35sending message: 0
Heart Rate Values:
 [83]Human Distance Info (cm):  35
Client Connection Socket Closed!
Sequence Stopped!
```

In [ ]:

```

```