

**American University of Sharjah**

**College of Engineering**

**Department of Computer Engineering**

**Embedded Systems (COE 410L)**

**Lab 1,2,3 Report**

**Amir Mohideen Basheer Khan - b00074559**

**Syed Raza - b00073832**

**Introduction**

Raspberry Pi (RPi) is a cheap and very small computer device that is widely used in embedded systems. We can connect external devices such as switches, Wi-Fi dongle, keyboard and mouse to receive user input and connect devices to Raspberry Pi such as monitor, LED, and buzzer to display output just like a usual computer. We can connect many devices due to the various ports available in the Raspberry Pi such as ethernet, HDMI, SD card, CSI, etc. In this lab, we will be studying the Raspberry pi 3 model B and coding programs that interact with the pi. Raspberry pi 3 model B which has

* Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
* 1GB RAM
* BCM43438 wireless LAN and Bluetooth Low Energy (BLE) on board
* 100 Base Ethernet
* 40-pin extended GPIO
* 4 USB 2 ports
* 4 Pole stereo output and composite video port
* Full size HDMI

**GPIO**

Raspberry Pi has 26 GPIO pins and 14 power or ground pins

There are two modes: BCM and Board mode

Graphical user interface, application

Description automatically generated

**Figure 1: GPIO of RPi 3 model B**

BCM board is preferred because the ground and source is not labelled as numbers like in Board mode.

In order to access the GPIO of Pi we enter the command “import RPi.GPIO as GPIO”

Certain GPIO pins can be used as UART (Pins 14,15), I2C (Pins 2,3) and SPI bus (Pins 7,8,9,10,11) when they are free.

The pi does not have a built-in real-time clock. It retrieves time through an internet connection.

**PWM**

The GPIO pins in the RBPi can generate pulse width modulation or PWM as output to make square waves of required frequencies and duty cycles to simulate analog output.

**Passive Buzzer**

Passive Buzzer is an output device that produces sound according to the AC transmitted to it

**Ultrasonic sensor**

**A picture containing electronics, projector

Description automatically generated**

**Figure 2: Ultrasonic Sensor**

An ultrasonic sensor is an input device that measures the distance to an object using ultrasonic soundwaves. It uses a transducer to send and receive ultrasonic pulses which is used to calculate the distance of objects by making use of the fact that the speed of sound is constant. It is based on how bats use echoes to determine its surroundings.

The ultrasonic sensor used in this lab has a VCC, ground trigger and echo. The trigger pin of the sensor should be set to high for 10 us for it to send an 8-cycle ultrasonic burst. The time that the sound wave travelled is recorded and sent as output to the echo pin.

Distance is usually found using this formula:

**Distance** (cm) = **time** (us) \* (**1000000/58**)

**PIR Motion sensor**



**Figure 3: PIR Motion Sensor**

PIR sensor is a digital motion sensor used to detect motion using infrared waves. The sensor sends infrared waves which hits a surface and reflects back to the sensor made of special set of lenses. By judging the deference in the signal read back, it could detect if an object moved in its range. This sensor has a power, ground and an output pin. When a motion is detected the alarm pin (output pin) of the sensor is triggered and it is set to LOW. Since, the alarm pin is an open collector, it needs to be connected to a pull up resistor. PIRs are used in cars

Graphical user interface, application

Description automatically generated**RFID**

**Figure 4: RFID Reader**

The radio frequency identification reader (RFID) is capable of receiving and transmitting information to RFID tags. It consists of a transceiver which transmits and receives data and a decoder which decodes received signals, and an antenna which transmits a signal continuously to be received by a tag within the range of the reader. When the tag receives the signal, the tag transmits its identification data in a stream of bits. The reader receives this stream of bits, verifies if it is correct and establishes communication between the reader and a tag through modulation of electromagnetic waves. RFID operates on a frequency of 125KHz and can read tags within a distance of 4 inches. Some of its applications are:

Access control and identification

Graphical user interface, application

Description automatically generated

**Figure 5: Types of RFID Tags**

**System Interface Diagram**

SOUT

Shape

Description automatically generated with medium confidenceShape

Description automatically generated with medium confidence

Raspberry Pi

5V vcc

3.3V

33k ohm

22K ohm

RFID Reader

Dr SW

Reverse

SW

Park SW

Ultrasonic

Sensor

PIR

Sensor

PassiveBuzzer

Output signal

PWM Signal

Echo

Trigger

RLED

PLED

DLED

**Code**

**#Lab 1**

import RPi.GPIO as GPIO

import time

#setting mode to BCM

GPIO.setmode(GPIO.BCM)

#setting up the switches, LED pins

ParkSwitch = 14

ParkLED = 15

ReverseSwitch = 23

ReverseLED = 24

DriveSwitch = 25

DriveLED = 8

GPIO.setup(ParkSwitch,GPIO.IN)

GPIO.setup(ParkLED ,GPIO.OUT)

GPIO.setup(ReverseSwitch,GPIO.IN)

GPIO.setup(ReverseLED,GPIO.OUT)

GPIO.setup(DriveSwitch ,GPIO.IN)

GPIO.setup(DriveLED ,GPIO.OUT)

GPIO.output(ParkLED,GPIO.LOW)

GPIO.output(ReverseLED,GPIO.LOW)

GPIO.output(DriveLED,GPIO.LOW)

# function for flashing the LEDs

def flashLED(LED, umode,tmode):

GPIO.output(LED, GPIO.LOW)

time.sleep(1)

GPIO.output(LED, GPIO.HIGH)

time.sleep(1)

print(" mode is changed to : {} on {} through {}".format(tmode,time.ctime(), umode))

while 1:

global terminalInput # make global variable terminal input

userInput = input("Enter SW or T") #take in the user input as SW or T

#set all LEDS to LOW

GPIO.output(ParkLED,GPIO.LOW)

GPIO.output(ReverseLED,GPIO.LOW)

GPIO.output(DriveLED,GPIO.LOW)

if userInput == "T": #check if userinput is Terminal

terminalInput = input("Enter mode") #if user wants to use terminal, ask which mode user wants to use

if terminalInput == "P": # check if terminal input is parking

flashLED(ParkLED,userInput,terminalInput) #flash parking LED

elif terminalInput == "R": #check if terminal input is Reverse

flashLED(ReverseLED,userInput,terminalInput) #call the reverse function

else: # if terminal input is Drive

flashLED(DriveLED,userInput,terminalInput) #call the drive function

else:

if GPIO.input(ParkSwitch)==1: # check if parking switch is high

terminalInput = "P"

flashLED(ParkLED,userInput,terminalInput) #flash parking LED

elif GPIO.input(ReverseSwitch) == 1: # check if reverse switch is high

terminalInput = "R"

flashLED(ReverseLED,userInput,terminalInput) #flash reverse LED

elif GPIO.input(DriveSwitch) == 1: # check if drive switch is high

terminalInput = "D"

flashLED(DriveLED,userInput,terminalInput) #flash drive LED

else:

pass

Graphical user interface, application, table

Description automatically generated

Figure 1.1:Drive mode using switch

Graphical user interface, application, table, Teams

Description automatically generated

Figure 1.2:Park mode using switch

Graphical user interface, application, table

Description automatically generated

Figure 1.3:Reverse mode using switch

**Lab 2**

#lab2

import RPi.GPIO as GPIO

import time

#setting mode to BCM

GPIO.setmode(GPIO.BCM)

#setting up the switches, LEDS, buzzer and trig/echo pins

ParkSwitch = 14

ParkLED = 15

ReverseSwitch = 23

ReverseLED = 24

DriveSwitch = 25

DriveLED = 8

buzzPin = 17

GPIO.setup(buzzPin, GPIO.OUT)

buzz = GPIO.PWM(buzzPin, 10)

buzz.start(1)

TRIG = 19

GPIO.setup(TRIG,GPIO.OUT)

ECHO = 20

GPIO.setup(ECHO,GPIO.IN)

GPIO.setup(ParkSwitch,GPIO.IN)

GPIO.setup(ParkLED ,GPIO.OUT)

GPIO.setup(ReverseSwitch,GPIO.IN)

GPIO.setup(ReverseLED,GPIO.OUT)

GPIO.setup(DriveSwitch ,GPIO.IN)

GPIO.setup(DriveLED ,GPIO.OUT)

GPIO.output(ParkLED,GPIO.LOW)

GPIO.output(ReverseLED,GPIO.LOW)

GPIO.output(DriveLED,GPIO.LOW)

# function for flashing the LEDs

def flashLED(LED, umode,tmode):

GPIO.output(LED, GPIO.LOW)

time.sleep(1)

GPIO.output(LED, GPIO.HIGH)

time.sleep(1)

print(" mode is changed to : {} on {} through {}".format(tmode,time.ctime(), umode))

# function for parking the car

def park(led,ui,ti): #takes in the led,the userinput and terminalInput as parameters

flashLED(led,ui,ti) #flash the specific led

# function for reversing the car

def reverse():

while True:

# function to calculate the distance

def distance():

GPIO.output(TRIG, 0) # set trigger to low

time.sleep(0.000002) # time delay of 2 micro seconds

GPIO.output(TRIG, 1) # Generate Pulse of 10µs

time.sleep(0.00001) # time delay of 2 micro seconds

GPIO.output(TRIG,0) # set trigger of 0

while GPIO.input(ECHO) == 0: # check if echo pin is low

a = 0

time1 = time.time() #capture time

while GPIO.input(ECHO) == 1: # check if echo pin is high

a = 0

time2 = time.time() #capture time

duration = time2-time1 # find pulse width

return (duration\*1000000)/58

dis = distance()/100

if dis < 30: # check if distance is less than 30

buzz.ChangeDutyCycle(50) # setting duty cycle to 50%

print("warning too close")

print ("distance = {} cm".format(dis))

GPIO.output(ReverseLED,GPIO.HIGH) # setting Reverse LED to High

time.sleep(1)

GPIO.output(ReverseLED,GPIO.LOW) # setting Reverse LED to Low

time.sleep(1)

else:

buzz.ChangeDutyCycle(1) # setting duty cycle to 1%

print ("distance = {} cm".format(dis))

# function for drive the car

def drive(led,ui,ti):

flashLED(led,ui,ti)

while 1:

global terminalInput # make global variable terminal input

userInput = input("Enter SW or T") #take in the user input as SW or T

#set all LEDS to LOW

GPIO.output(ParkLED,GPIO.LOW)

GPIO.output(ReverseLED,GPIO.LOW)

GPIO.output(DriveLED,GPIO.LOW)

if userInput == "T": #check if userinput is Terminal

terminalInput = input("Enter mode") #if user wants to use terminal, ask which mode user wants to use

if terminalInput == "P": # check if terminal input is parking

park(ParkLED,userInput,terminalInput) #flash parking LED

elif terminalInput == "R": #check if terminal input is Reverse

reverse() #call the reverse function

else: # if terminal input is Drive

drive(DriveLED,userInput,terminalInput) #call the drive function

else:

if GPIO.input(ParkSwitch)==1: # check if parking switch is high

terminalInput = "P"

flashLED(ParkLED,userInput,terminalInput) #flash parking LED

elif GPIO.input(ReverseSwitch) == 1: # check if reverse switch is high

terminalInput = "R"

flashLED(ReverseLED,userInput,terminalInput) #flash reverse LED

elif GPIO.input(DriveSwitch) == 1: # check if drive switch is high

terminalInput = "D"

flashLED(DriveLED,userInput,terminalInput) #flash drive LED

else:

pass

Table

Description automatically generated

Figure 1.4:Normal Reverse mode

Table

Description automatically generated

Figure 1.5: Updated Reverse mode

**Lab3**

#lab 3

import RPi.GPIO as GPIO

import time

#setting mode to BCM

GPIO.setmode(GPIO.BCM)

#setting pin 18 as pull up

GPIO.setup(18,GPIO.IN,pull\_up\_down=GPIO.PUD\_UP)

#setting up the switches, LEDS, buzzer and trig/echo pins

ENABLE\_PIN = 27

SERIAL\_PORT = '/dev/ttyS0'

ParkSwitch = 14

ParkLED = 15

ReverseSwitch = 23

ReverseLED = 24

DriveSwitch = 25

DriveLED = 8

buzzPin = 17

GPIO.setup(buzzPin, GPIO.OUT)

buzz = GPIO.PWM(buzzPin, 10)

buzz.start(1)

TRIG = 19

GPIO.setup(TRIG,GPIO.OUT)

ECHO = 20

GPIO.setup(ECHO,GPIO.IN)

GPIO.setup(ParkSwitch,GPIO.IN)

GPIO.setup(ParkLED ,GPIO.OUT)

GPIO.setup(ReverseSwitch,GPIO.IN)

GPIO.setup(ReverseLED,GPIO.OUT)

GPIO.setup(DriveSwitch ,GPIO.IN)

GPIO.setup(DriveLED ,GPIO.OUT)

GPIO.setup(ENABLE\_PIN, GPIO.OUT)

GPIO.output(ParkLED,GPIO.LOW)

GPIO.output(ReverseLED,GPIO.LOW)

GPIO.output(DriveLED,GPIO.LOW)

GPIO.output(ENABLE\_PIN, GPIO.LOW)

#function to validate the RFID code

def validate\_rfid(code):

# A valid code will be 12 characters (bytes) long with the first char being

# a line feed and the last char being a carriage return.

s = code.decode("ascii")

if (len(s) == 12) and (s[0] == "\n") and (s[11] == "\r"):

# We matched a valid code. Strip off the "\n" and "\r" and just

# return the RFID code which 10 bits. Linefeed= 0A an CR=0D

return s[1:-1]

else:

# We didn't match a valid code, so return False.

return False

# function for flashing the LEDs

def flashLED(LED, umode,tmode):

GPIO.output(LED, GPIO.LOW)

time.sleep(1)

GPIO.output(LED, GPIO.HIGH)

time.sleep(1)

print(" mode is changed to : {} on {} through {}".format(tmode,time.ctime(), umode))

flag = 0 #set a global flag so that action function gets called only in the parking mode

def action(self): #Call back function to perform an action whenever a falling edge is detected

if flag:

print('motion detected')

GPIO.output(ParkLED, GPIO.LOW) # set Parking led to low

time.sleep(1)

GPIO.output(ParkLED, GPIO.HIGH) # set Parking led to low

time.sleep(1)

#set

GPIO.add\_event\_detect(18,GPIO.FALLING,callback=action, bouncetime=2000)

# function for parking the car

def park():

flag = 1

# function for reversing the car

def reverse():

flag = 0

while True:

# function to calculate the distance

def distance():

GPIO.output(TRIG, 0) # set trigger to low

time.sleep(0.000002) # time delay of 2 micro seconds

GPIO.output(TRIG, 1) # Generate Pulse of 10µs

time.sleep(0.00001) # time delay of 2 micro seconds

GPIO.output(TRIG,0) # set trigger of 0

while GPIO.input(ECHO) == 0: # check if echo pin is low

a = 0

time1 = time.time() #capture time

while GPIO.input(ECHO) == 1: # check if echo pin is high

a = 0

time2 = time.time() #capture time

duration = time2-time1 # find pulse width

return (duration\*1000000)/58

dis = distance()/100

if dis < 30: # check if distance is less than 30

buzz.ChangeDutyCycle(50) # setting duty cycle to 50%

print("warning too close")

print ("distance = {} cm".format(dis))

GPIO.output(ReverseLED,GPIO.HIGH) # setting Reverse LED to High

time.sleep(1)

GPIO.output(ReverseLED,GPIO.LOW) # setting Reverse LED to Low

time.sleep(1)

else:

buzz.ChangeDutyCycle(1) # setting duty cycle to 1%

print ("distance = {} cm".format(dis))

# function for drive the car

def drive(led,ui,ti):

flag = 0

flashLED(led,ui,ti)

# Set up the serial port as per the Parallax reader's datasheet.

ser = serial.Serial(baudrate = 2400,

bytesize = serial.EIGHTBITS,

parity = serial.PARITY\_NONE,

port = SERIAL\_PORT,

stopbits = serial.STOPBITS\_ONE,

timeout = 1)

while 1:

print("please swipe your RFID card")

# Read in 12 bytes from the serial port.

data = ser.read(12)

# Attempt to validate the data we just read.

code = validate\_rfid(data)

# If validate\_rfid() returned a code, display it.

if code:

print("Read RFID code: " + code)

if (code == “XXXXXXXXX”): # If correct tag is used open the door.

while true:

time.sleep(2)

terminalInput

userInput = input("Enter SW or T") #take in the user input as SW or T

#set all LEDS to LOW

GPIO.output(ParkLED,GPIO.LOW)

GPIO.output(ReverseLED,GPIO.LOW)

GPIO.output(DriveLED,GPIO.LOW)

if userInput == "T": #check if userinput is Terminal

terminalInput = input("Enter mode") #if user wants to use terminal, ask which mode user wants to use

if terminalInput == "P": # check if terminal input is parking

park() #call the parking function

elif terminalInput == "R": #check if terminal input is Reverse

reverse() #call the reverse function

else: # if terminal input is Drive

drive(DriveLED,userInput,terminalInput) #call the drive function

else:

if GPIO.input(ParkSwitch)==1: # check if parking switch is high

terminalInput = "P"

flashLED(ParkLED,userInput,terminalInput) #flash parking LED

elif GPIO.input(ReverseSwitch) == 1: # check if reverse switch is high

terminalInput = "R"

flashLED(ReverseLED,userInput,terminalInput) #flash reverse LED

elif GPIO.input(DriveSwitch) == 1: # check if drive switch is high

terminalInput = "D"

flashLED(DriveLED,userInput,terminalInput) #flash drive LED

else:

pass

else:

print("please try again")

Graphical user interface, text

Description automatically generated

Graphical user interface, text, application, chat or text message

Description automatically generated

**Conclusion**

This lab helped us in familiarizing with embedded systems by exploring the functionalities of Raspberry Pi and the various sensors that can be connected and utilized by coding Python programs that provide instruction to Raspberry Pi to interact with them by receiving input and outputs.

As a result, we built a simple car control system. The user can select between the terminal or a set of switches to change to three gear modes namely parking, drive, or reverse mode. When user is in reverse mode, ultrasonic sensor calculates the distance of any object that is behind the car and alerts the user if object is very close to the car by flashing led lights and buzzer. When user is in parking mode, motion sensor detects motion within the car and alerts the user. We also implemented a RFID system so that a user can open the car only if they have the correct tag.

To conclude, this lab enhanced our knowledge about embedded systems by guiding us to code programs that allow Raspberry Pi to interact with the sensors in a specific way for it to be able to be useful for human users in the real world.