UNIVERSITY OF CAPE TOWN
IYUNIVESITHI YASEKAPA • UNIVERSITEIT VAN KAAPSTAD

Faculty of Engineering and the Built Environment

Department of Electrical Engineering

Practical 4: RPI-3B SPI and Interrupt

EEE3096S – Embedded Systems II

Name: Mueez Allie and Ferial Najaar

Student Number: ALLMUE001 and NJRFER001

Date: 4 September 2018

Contents

Question 1a

An SPI is a serial synchronous type of communication which allows 2 devices to synchronise the exchange of data through a common clock signal. In the timing diagram below it can be seen that the clock is an oscillating signal. The clock tells the receiver when to sample the received signal. The received signal is either sampled on the rising edge or falling edge of the clock. In this case, it can be seen that, the signal is sampled on the rising edge of the clock. When this edge is detected by the receiver the data line is read to determine the next bit.

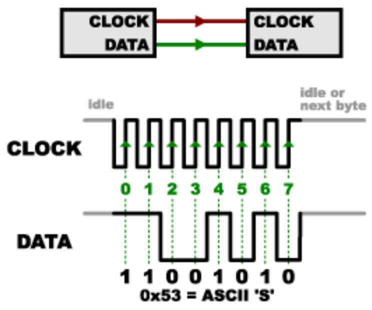

Figure 1: SPI timing diagram

Question 1b

An interrupt is a signal, indicating an event that must be executed immediately, from a program which causes the operating system to stop any running processes in order to handle the event sent by the signal. After the interrupt event has been handled the program will continue running where it left off .
A callback is a function which can be passed as an argument to another function. This can be used to enforce an order of operations in a program. Threading allows the program to execute other code while waiting for another event to complete. A threaded callback occurs when a callback function is run in a second thread while the main function continues to run.

Question 1c

Function that converts a 10-bit ADC reading from a potentiometer to a 3V3 limited voltage output:

```
#Function to convert ADC reading to a voltage
def voltage(adc):
    volt = (adc/float(1023))
    voltage = (volt*3.3)
    voltage = round(voltage, 1)
    return voltage
```

Question 1d

Function that converts a 10-bit ADC reading from the temperature sensor to a reading in degrees Celsius:

```
#Function to convert ADC reading to a temperature in degrees celsius
def temp(adc):
    volt = (adc/float(1023))
    vout = (volt*3.3)
    temp = (vout-0.5)/float(0.01)
    temp = round(temp, 1)
    return temp
```

Question 1e

Function that converts a 10-bit ADC reading from the LDR to a percentage representing the amount of light received by the LDR:

```
#Function to convert ADC reading to a percentage for light
def lightper(adc):
    per = (adc/float(1023))
    perce = (per*100)
    percent = 100-perce
    percent = round(percent, 2)
    return percent
```

Question 1f

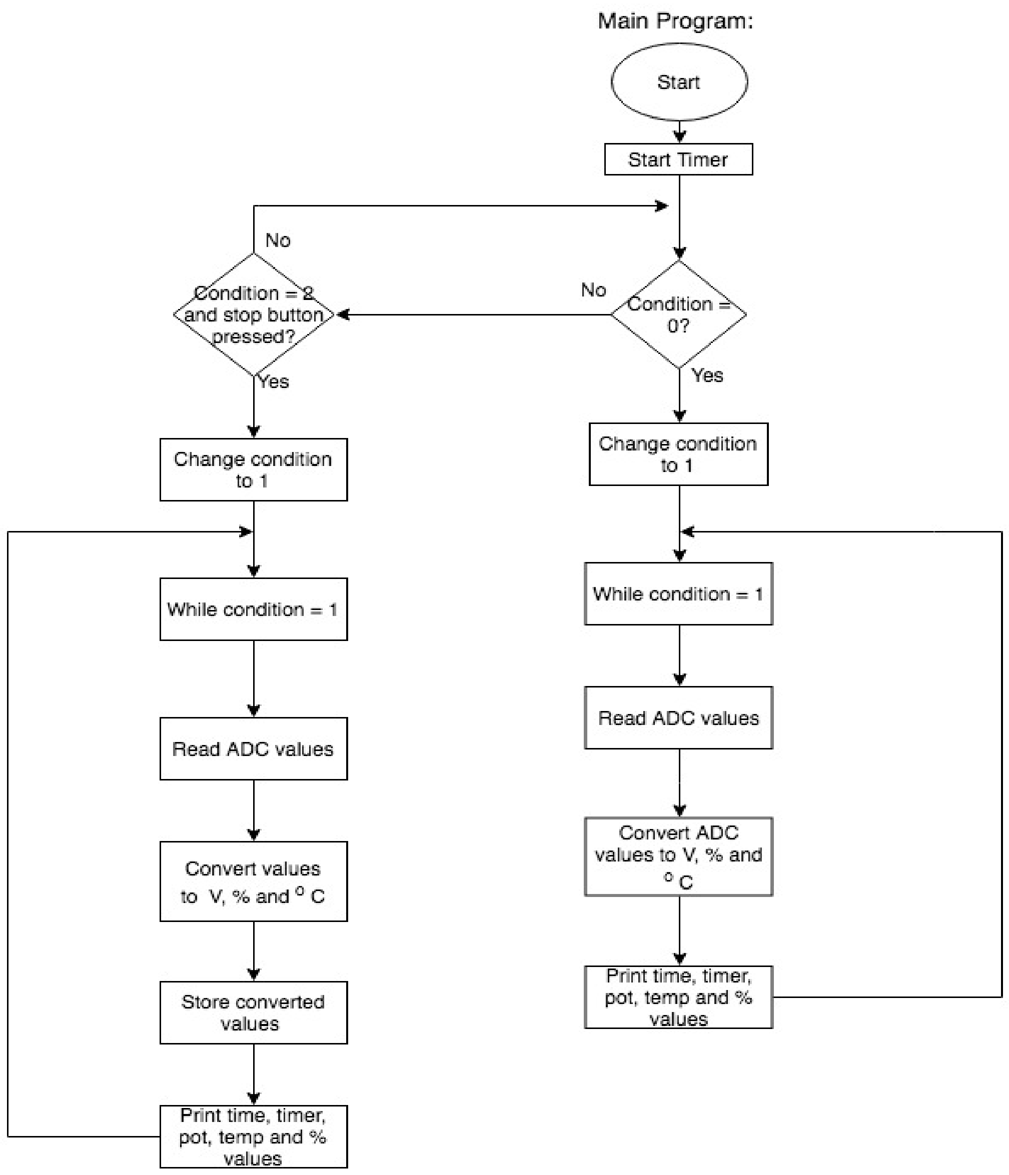

Figure 2: Flow chart showing the main program function.

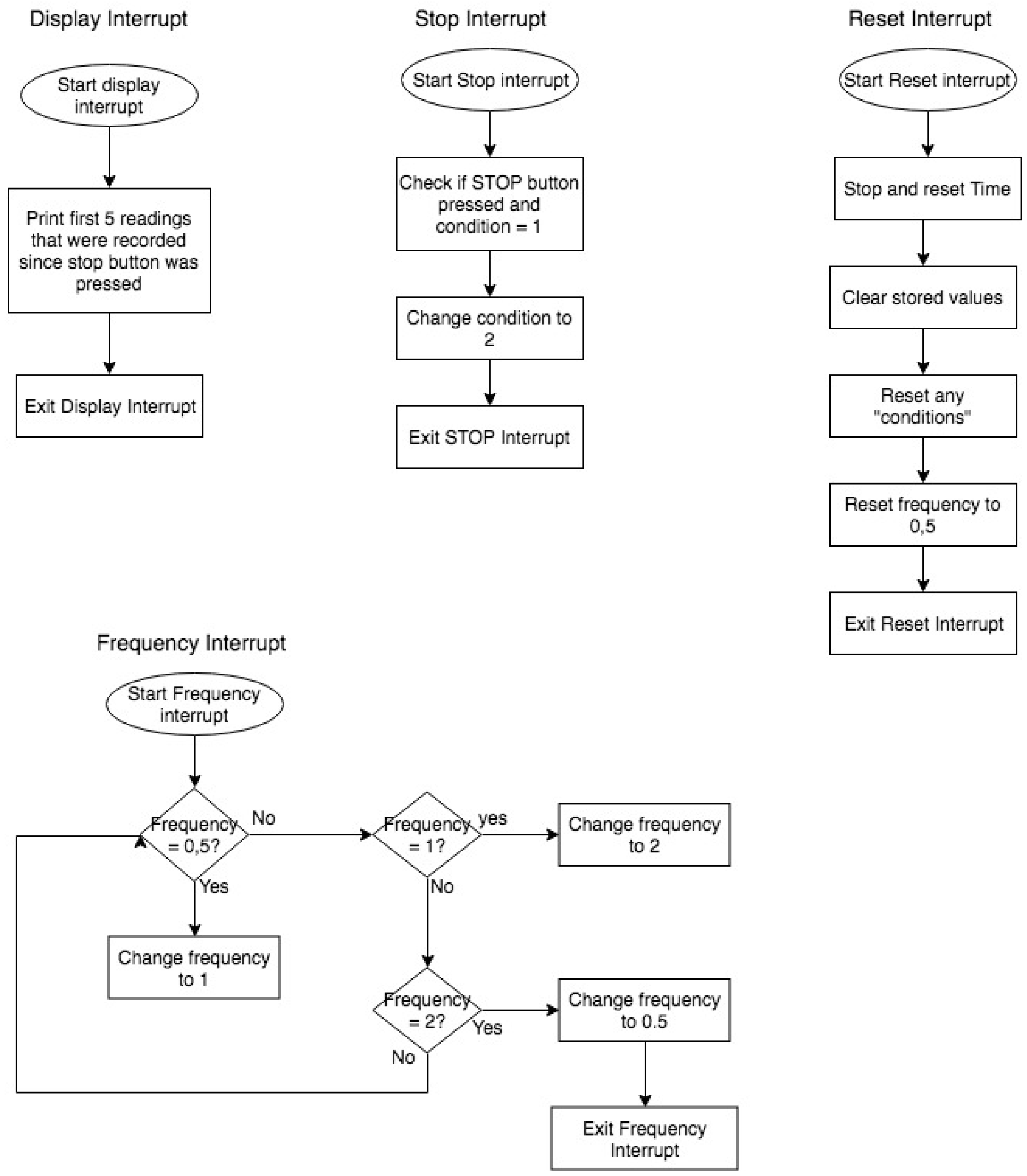

Figure 3: Flow chart representing the interrupt functions.

Appendix A

GPIO pins are configured in the BCM layout.

Circuit diagrams:

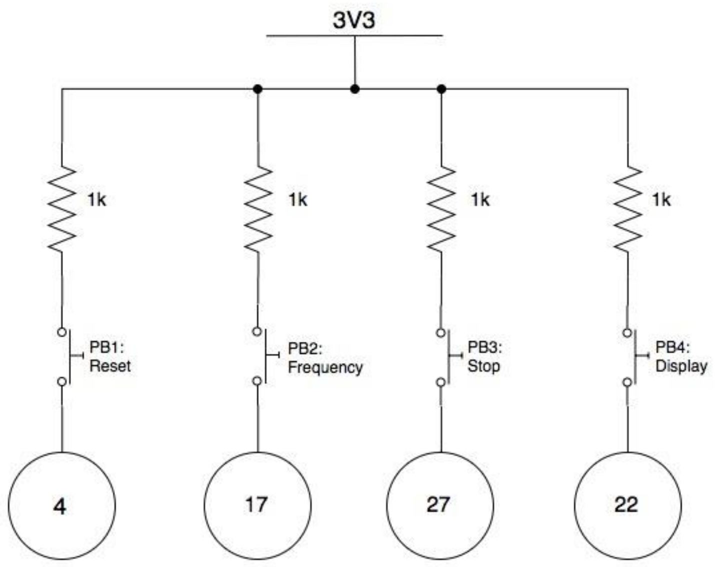

Figure 4: Circuit Diagram showing how each pushbutton in connected to the Raspberry Pi GPIO pins

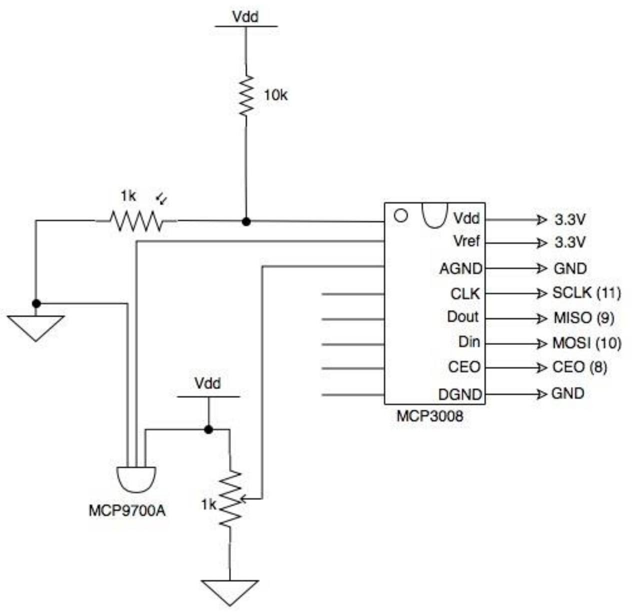

Figure 5: Circuit Diagram showing how the 10-bit ADC was connected to external component circuitry, power, ground as well as the dedicated GPIO pins on the Raspberry Pi.

Appendix B

Code:

```
#Ferial Najaar & Mueez Allie
#NJRFER001     & ALLMUE001

import spidev
import time
import os
import RPi.GPIO as GPIO
import Adafruit_MCP3008
import datetime
#from datetime import timedelta

#Pin Definition
GPIO.setmode(GPIO.BCM)

SPICLK = 11
SPIMISO = 9
SPIMOSI = 10
SPICS = 8

RESET = 4
FREQ = 17
STARTSTOP = 27
DISP = 22

#Making the pushbuttons inputs and setting them to 'pull-down' mode
GPIO.setup(RESET, GPIO.IN, pull_up_down = GPIO.PUD_DOWN)
GPIO.setup(FREQ, GPIO.IN, pull_up_down = GPIO.PUD_DOWN)
GPIO.setup(STARTSTOP, GPIO.IN, pull_up_down = GPIO.PUD_DOWN)
GPIO.setup(DISP, GPIO.IN, pull_up_down = GPIO.PUD_DOWN)

#Setting up the input and output pins to and from the 10-bit ADC chip
GPIO.setup(SPIMOSI, GPIO.OUT)
GPIO.setup(SPIMISO, GPIO.IN)
GPIO.setup(SPICLK, GPIO.OUT)
GPIO.setup(SPICS, GPIO.OUT)


mcp = Adafruit_MCP3008.MCP3008(clk=SPICLK, cs=SPICS, mosi=SPIMOSI,
miso=SPIMISO)

global timing
timing = datetime.timedelta(seconds = 0)


#Function to convert ADC reading to a voltage
def voltage(adc):
    volt = (adc/float(1023))
    voltage = (volt*3.3)
    voltage = round(voltage, 1)
    return voltage

#Function to convert ADC reading to a temperature in degrees celsius
def temp(adc):
    volt = (adc/float(1023))
    vout = (volt*3.3)
    temp = (vout-0.5)/float(0.01)
    temp = round(temp, 1)
    return temp

#Function to convert ADC reading to a percentage for light
def lightper(adc):
    per = (adc/float(1023))
    perce = (per*100)
    percent = 100-perce
    percent = round(percent, 2)
    return percent

#Function to display the first five readings since the stop button was pressed
def Display(x):

    print("Display button has been pressed")
    global val
    global readings

    Time = time.strftime('%X')
    pot = voltage(val[2])
    t = temp(val[1])
    light = lightper(val[0])
    print('-----------------------------------------------------')
    print('Time          Timer          Pot       Temp       Light')
    print('-----------------------------------------------------')

    for i in range(5):
        print(str(readings[i][0]) + '     ' + str(readings[i][1]) + '      ' +
str(readings[i][2])+ 'V      ' + str(readings[i][3])+ 'C      ' +
str(readings[i][4]) + '%')
        print('-----------------------------------------------------')

    time.sleep(0.2)

#Function to change the frequency once the frequency button is pressed
def changeFreq(y):

    print("Frequency will be changed")
    global f
    if f == 0.5:
        f = f+0.5
        print("New Frequency = " + str(f))
        time.sleep(0.2)
    elif f == 1:
        f = f+1
        print("New Frequency = " + str(f))
        time.sleep(0.2)
    else:
        f = 0.5
        print("New Frequency = " + str(f))
        time.sleep(0.2)

#Function that stops the monitoring of the system
def Stop(z):
    global onoff


    if GPIO.input(STARTSTOP) == GPIO.HIGH and onoff == 1:

        print("Monitoring has stopped")
        onoff = onoff + 1


def Timer():
    global f
    global timing
    if f == 0.5:
        timing = timing + datetime.timedelta(seconds = 0.5)
        strtime = str(timing) + '.0000000'
        strtime = strtime[2:10]
        #print(timing)
    elif f == 1:
        timing = timing + datetime.timedelta(seconds = 1)
        strtime = str(timing) + '.0000000'
        strtime = strtime[2:10]
        #print(timing)
    elif f == 2:
        timing = timing + datetime.timedelta(seconds = 2)
        strtime = str(timing) + '.0000000'
        strtime = strtime[2:10]
        #print(timing)

    return strtime




#Function that resets the system
def Reset(g):

    print("Console has been Reset")
    global readings
    readings = []
    global onoff
    onoff = 0
    global f
    f = 0.5
    global timing
    timing = datetime.timedelta(milliseconds = 0)

#Defining the interrupts and their applications
GPIO.add_event_detect(RESET, GPIO.RISING, callback = Reset, bouncetime = 200)

GPIO.add_event_detect(FREQ, GPIO.RISING, callback = changeFreq, bouncetime =
300)

GPIO.add_event_detect(STARTSTOP, GPIO.RISING, callback = Stop, bouncetime =
300)

GPIO.add_event_detect(DISP, GPIO.RISING, callback = Display, bouncetime = 300)

#Global Variables
values = [0]*8

global onoff
onoff = 0

global readings
readings = []
```

```
global f
f = 0.5

global val
val = [0,0,0,0,0,0,0,0]

global temperature


try:
    #Infinite while loop
    while True:

        #Checking for the default system condition
        if onoff == 0:

            print("Default behaviour")

            print('-------------------------------------------------------')
            print('Time         Timer          Pot        Temp        Light')
            print('-------------------------------------------------------')

            timing = datetime.timedelta(seconds = 0)
            readings = []
            time.sleep(0.2)
            onoff = onoff + 1     #Changing the default condition
            loop = 0

            #Monitor the system while the default condition remains unchanged
            while onoff == 1:
                col = 5
                row = 5

                for i in range(8):
                    values[i] = mcp.read_adc(i)

                #Delay for half a second

                global val
                val = values

                volt = voltage(values[2])

                global temperature
                temperature = temp(values[1])

                perlight = lightper(values[0])

                Time = time.strftime('%X')

                Timers = Timer()

                readings.append([Time, Timers, volt, temperature, perlight])

                loop += 1

                print(str(Time) + '     ' + str(Timers) + '      ' + str(volt)+
'V      ' + str(temperature)+ 'C      ' + str(perlight) + '%')
                print('-------------------------------------------------------
')
```

```
                time.sleep(f)

        #Checking is the 'system start' condition has been met
        if GPIO.input(STARTSTOP) == GPIO.HIGH and onoff == 2:

            print("Monitoring has started")

            print('-------------------------------------------------------')
            print('Time         Timer         Pot       Temp       Light')
            print('-------------------------------------------------------')

            readings = []
            time.sleep(0.2)
            onoff = onoff - 1 #changing the system start condition
            loop = 0

            #Monitor the system while the 'system start' condition is met
            while onoff == 1:
                col = 5
                row = 5

                for i in range(8):
                    values[i] = mcp.read_adc(i)

                global val
                val = values

                volt = voltage(values[2])

                global temperature
                temperature = temp(values[1])

                perlight = lightper(values[0])

                Time = time.strftime('%X')

                Timers = Timer()

                readings.append([Time, Timers, volt, temperature, perlight])

                loop += 1



                print(str(Time) + '     ' + str(Timers) + '      ' + str(volt)+
'V      ' + str(temperature)+ 'C      ' + str(perlight) + '%')
                print('-------------------------------------------------------
')

                time.sleep(f)

        Timer()
        time.sleep(f)

#Removing interrupts
#GPIO.remove_event_detect(RESET)
#GPIO.remove_event_detect(FREQ)
#GPIO.remove_event_detect(STOP)
#GPIO.remove_event_detect(DISP)

except KeyboardInterrupt:
    GPIO.cleanup()

#Cleaning up the GPIO pins
GPIO.cleanup()
```

References

Grusin, M. (n.d.). *Serial Peripheral Interface (SPI) - learn.sparkfun.com*. [online] Learn.sparkfun.com. Available at: https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi [Accessed 4 Sep. 2018].

En.wikipedia.org. (2018). *Callback (computer programming)*. [online] Available at: https://en.wikipedia.org/wiki/Callback_(computer_programming)#Python [Accessed 8 Sep. 2018].

En.wikibooks.org. (2018). *Python Programming/Threading - Wikibooks, open books for an open world*. [online] Available at: https://en.wikibooks.org/wiki/Python_Programming/Threading [Accessed 8 Sep. 2018].

Sourceforge.net. (n.d.). *raspberry-gpio-python / Wiki / Inputs*. [online] Available at: https://sourceforge.net/p/raspberry-gpio-python/wiki/Inputs/ [Accessed 8 Sep. 2018].

Tutoring, L. (2017). *What is a callback?*. [video] Available at: https://www.youtube.com/watch?v=xHneyv38Jro [Accessed 8 Sep. 2018].

www.tutorialspoint.com. (n.d.). *Embedded Systems Interrupts*. [online] Available at: https://www.tutorialspoint.com/embedded_systems/es_interrupts.htm [Accessed 8 Sep. 2018].

Murphy, M. (2011). *Interrupts and I/O*. [video] Available at: https://www.youtube.com/watch?v=tn45bY-EA8Y [Accessed 8 Sep. 2018].