Environment Monitoring

Brendon Sehlako[†] [‡]

Prepared for University of Cape Town
South Africa
†

***Abstract*—The use of a raspberry pi for a monitoring devices, is a very elegant way of showcasing what could be with relevant knowledge of the operating system.**

I. Introduction

Our design for the terrarium logger included both a circuit and a piece of software that runs on the Raspberry Pi connected to our circuit. The software will be discussed later in the report. What follows is a description of the design choices we made for the circuit. As per the specification, our circuit includes two buttons – one for adjusting the intervals at which the temperature is monitored, and one enabling/disabling logging of the temperature and ADC values. The circuit also includes a thermistor for detecting temperature, and a buzzer which goes off at every interval. A circuit diagram will be provided later in the report.

This report will detail the requirements of the system – this will be shown by a UML Case Diagram. The report will also include a Specification and Design section which will indicate the structuring of our software implementation. The Specification and Design section will include a UML State Chart as well as a UML class diagram. Also included in this report is an Implementation section which will provide descriptions of sections of our code for further clarification. After the Implementation section comes the Validation and Performance section which will evaluate the performance of the system. Finally, a conclusion will follow, which will describe the success/failure of the system and discuss whether we believe the terrarium logger is a viable product.

II. Requirements

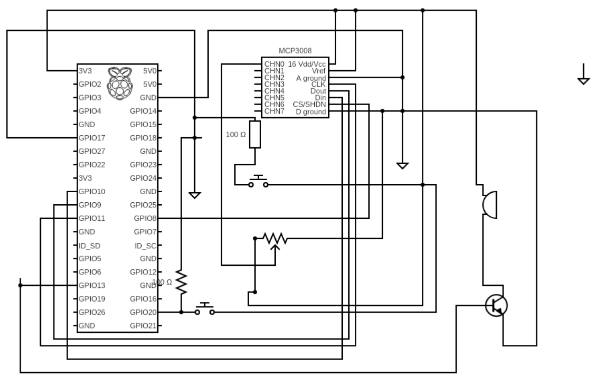


Fig. 1: Circuit

The circuit above shows the system to be implemented, All the relevant components as well as their connections are shown accordingly. For the buttons resistors were used as the code specified pull up resistors. The circuit shows a potentiometer, however this is an ADC.

A. Hardware

- EEPROM
- buzzer
- 2xResistors
- pnp 2222a
- ADC - MCP3008
- Temperature sensor - MCP9700A(In the circuit shown as a potentiometer)
- Raspberry pi

III. Design And Specifications

The design choices and specifications are as follows, in the class diagram chart as well as the flow chart of the system.

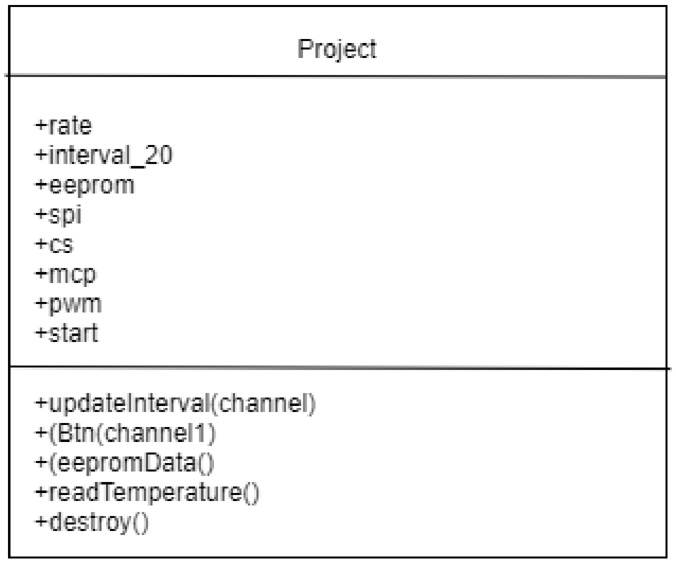


Fig. 2: Class Diagram

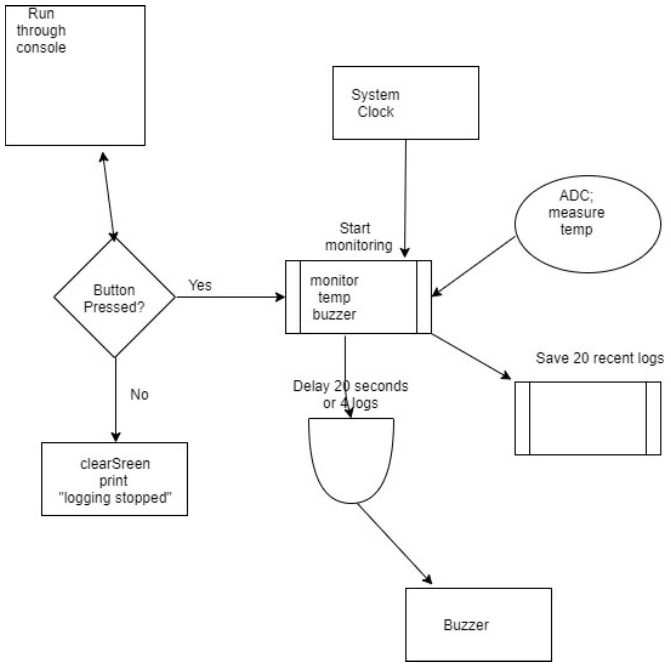


Fig. 3: Eeprom

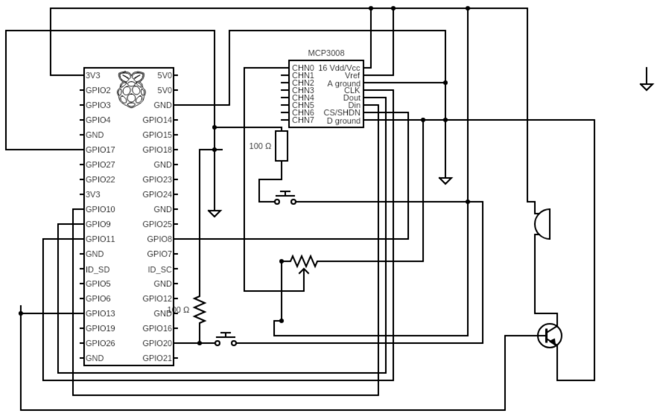


Fig. 4: Eeprom

IV. Implementation

Here we have the setup segments of the code, this was done in this manner in order to minimise debugging errors. Our board is setup, as well as the GPIO parts which are responsible for the functionality of our button, the buttons will be in the bits of code to follow.At the top we have variable which will be responsible for our Buzzer at relevant intervals. Lastly we have the button detect event for changing. The frequency of our monitoring intervals.

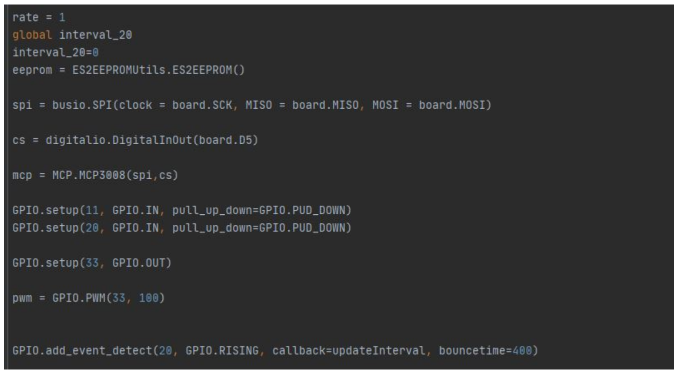

```
rate = 1
global interval_20
interval_20=0
eeprom = ES2EEPROMUtils.ES2EEPROM()

spi = busio.SPI(clock = board.SCK, MISO = board.MISO, MOSI = board.MOSI)

cs = digitalio.DigitalInOut(board.D5)

mcp = MCP.MCP3008(spi,cs)

GPIO.setup(11, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
GPIO.setup(20, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)

GPIO.setup(33, GPIO.OUT)

pwm = GPIO.PWM(33, 100)


GPIO.add_event_detect(20, GPIO.RISING, callback=updateInterval, bouncetime=400)
```

Fig. 5: Setup

Below are the buttons, we have two buttons. One button is for the frequency change of the monitoring times, and the second is for starting and stopping the monitoring process. These do prove very useful for remote monitoring and for times when more frequent monitoring is required, the buttons are very useful feature.

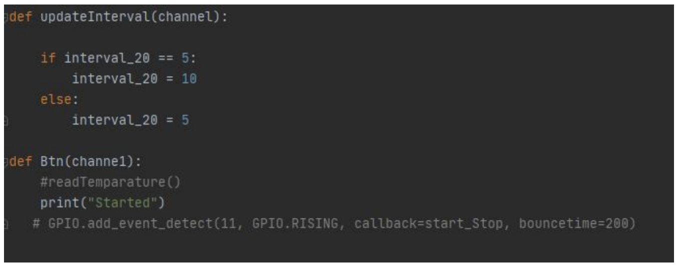

```
def updateInterval(channel):

    if interval_20 == 5:
        interval_20 = 10
    else:
        interval_20 = 5

def Btn(channel):
    #readTemparature()
    print("Started")
   # GPIO.add_event_detect(11, GPIO.RISING, callback=start_Stop, bouncetime=200)
```

Fig. 6: Buttons

Buttons can have added functionalities however, for the implementation two button were sufficient

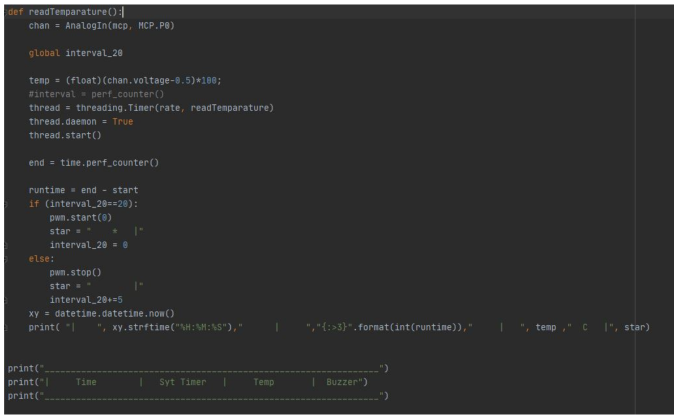
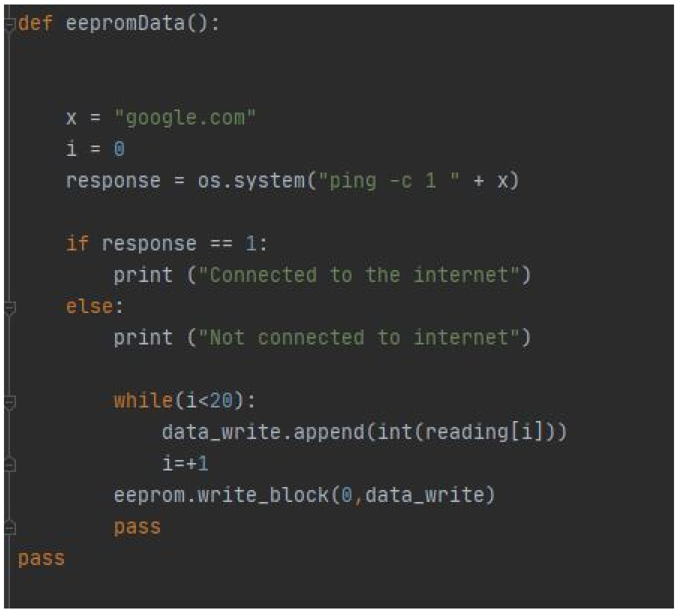

```
def readTemparature():
    chan = AnalogIn(mcp, MCP.P0)

    global interval_20

    temp = (float)(chan.voltage-0.5)*100;
    #interval = perf_counter()
    thread = threading.Timer(rate, readTemparature)
    thread.daemon = True
    thread.start()

    end = time.perf_counter()

    runtime = end - start
    if (interval_20==20):
        pwm.start(0)
        star = "    *   |"
        interval_20 = 0
    else:
        pwm.stop()
        star = "         |"
        interval_20+=5
    xy = datetime.datetime.now()
    print( "|    ", xy.strftime("%H:%M:%S"),"      |      ","{:>3}".format(int(runtime)),"     |   ", temp ," C   |", star)


print("_______________________________________________________________")
print("|     Time        |   Syt Timer   |     Temp      |  Buzzer")
print("_______________________________________________________________")
```

Fig. 7: Temperature

The bit of code above does the processing of the acquired data, and subsequently extracting relevant information from the sensors.It also does the threading as well as reading the real time and system time.

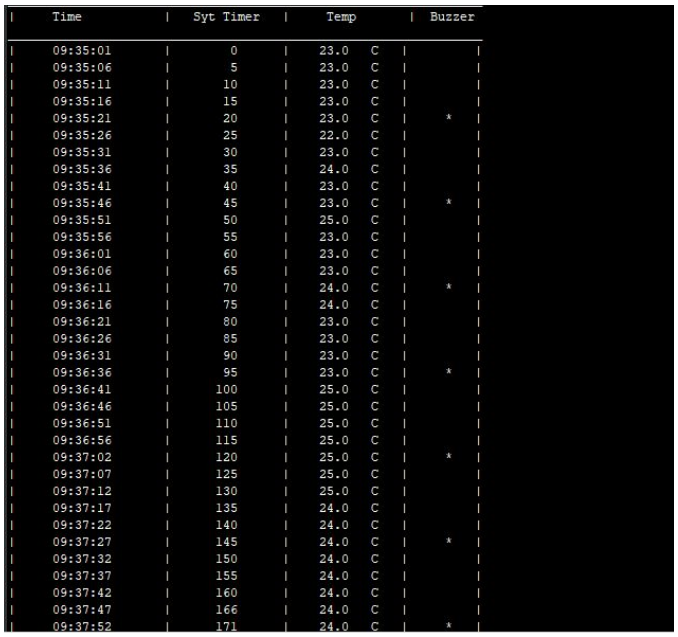
With regards to the eeprom there was a consideration to have another button, however that would have proved to have been a more chunky system than my uncle would want some we took designed liberty and had a piece of code that would check for internet connection. And in cases there isn't the eeprom would start saving readings up to a maximum of 20 readings to later retrieved when necessary.



```
def eepromData():


    x = "google.com"
    i = 0
    response = os.system("ping -c 1 " + x)

    if response == 1:
        print ("Connected to the internet")
    else:
        print ("Not connected to internet")

        while(i<20):
            data_write.append(int(reading[i]))
            i=+1
        eeprom.write_block(0,data_write)
        pass
pass
```

Fig. 8: Eeprom

V. Validation And Performance



```
|      Time          |  Syt Timer  |     Temp       |  Buzzer
---------------------------------------------------------------
|      09:35:01      |        0    |   23.0    C   |          |
|      09:35:06      |        5    |   23.0    C   |          |
|      09:35:11      |       10    |   23.0    C   |          |
|      09:35:16      |       15    |   23.0    C   |          |
|      09:35:21      |       20    |   23.0    C   |     *    |
|      09:35:26      |       25    |   22.0    C   |          |
|      09:35:31      |       30    |   23.0    C   |          |
|      09:35:36      |       35    |   24.0    C   |          |
|      09:35:41      |       40    |   23.0    C   |          |
|      09:35:46      |       45    |   23.0    C   |     *    |
|      09:35:51      |       50    |   25.0    C   |          |
|      09:35:56      |       55    |   23.0    C   |          |
|      09:36:01      |       60    |   23.0    C   |          |
|      09:36:06      |       65    |   23.0    C   |          |
|      09:36:11      |       70    |   24.0    C   |     *    |
|      09:36:16      |       75    |   24.0    C   |          |
|      09:36:21      |       80    |   23.0    C   |          |
|      09:36:26      |       85    |   23.0    C   |          |
|      09:36:31      |       90    |   23.0    C   |          |
|      09:36:36      |       95    |   23.0    C   |     *    |
|      09:36:41      |      100    |   25.0    C   |          |
|      09:36:46      |      105    |   25.0    C   |          |
|      09:36:51      |      110    |   25.0    C   |          |
|      09:36:56      |      115    |   25.0    C   |          |
|      09:37:02      |      120    |   25.0    C   |     *    |
|      09:37:07      |      125    |   25.0    C   |          |
|      09:37:12      |      130    |   25.0    C   |          |
|      09:37:17      |      135    |   24.0    C   |          |
|      09:37:22      |      140    |   24.0    C   |          |
|      09:37:27      |      145    |   24.0    C   |     *    |
|      09:37:32      |      150    |   24.0    C   |          |
|      09:37:37      |      155    |   24.0    C   |          |
|      09:37:42      |      160    |   24.0    C   |          |
|      09:37:47      |      166    |   24.0    C   |          |
|      09:37:52      |      171    |   24.0    C   |     *    |
```

Fig. 9: Results

In the figure about are the values, that our system yielded over a period of time. It was a could morning hence the low temperatures. What the values show us is the that is works, the system time and the local time works as well. The system was left to running for extended periods of time to ensure that it is a reliable system. There could have been editions we could have made to the system but the the amount of time on our hands it wasn't feasible.

VI. Conclusion

We managed to satisfy all of the requirements for this project. The temperature value was outputted every 5 seconds and the buzzer went off every 20 seconds. The buttons performed the correct functions as well. This project was therefore a success. We believe that even though this project was a success, changes would need to be made to make this a viable product. Our version of the circuit was rather large and in order for the user to see the results they had to log into a terminal. We think that the system could be improved by making the system send the results to an app on the users phone, and making the circuit smaller and more compact.

References