EEE3096S

Embedded Systems II

Mini Project A: Environment Logger

Presented By: Junior Makgoe (MKGPUL005) and
Mark Njoroge (NJRMAR003)

For Keegan Crankshaw

Introduction

This mini project simulates an environmental logger in a greenhouse. The logger is expected to take in a light reading, humidity reading and a temperature reading. If the environmental conditions go outside a specific desired range, the system sounds an alarm.

The components used to simulate this were a Raspberry Pi Model 3b+, an MCP3008 Analogue to digital converter, an MCP7940M real time clock, a potentiometer to model the humidity, an LDR, a temperature sensor, a buzzer and input buttons. Each Input device is fed into the ADC and the output of the ADC is sent to the pi for processing. The reason for choosing these components was mostly fueled by price and accessibility, the focus being to simulate the environment logger at an affordable price.

In order to accurately design the system, the first step was to outline the uses of the system, the inputs and the outputs. This step told us the specific requirements of the system and how we expect the user to interact with the system. Once this was done the next step was to outline the different states of the system as well as how to implement it. The system was then built and implemented. This involves building the circuitry and writing the code to perform the processing of the data. This report outlines each of the steps and shows the different design specifications of the system.

Requirements

Since we are dealing with embedded systems, there are separate requirements for the hardware and the software.

Hardware requirements

The hardware requirements are as follows:

- The components should have low tolerances so that accurate results can be given to the students
- Working design to demonstrate

Software requirements

The software requirements are therefore to:

- Display the time to user
- Display elapsed system time to the user
- Read light, temperature and potentiometer readings from the ADC and convert these into data that can be understood by the users
- Cause a buzzer to sound when the output voltage falls outside the range 0.65 <V< 2.65 and notify the user.
- Use Blynk to monitor the conditions within the greenhouse remotely
- Stop the alarm when it is ringing by pressing a button
- Start and stop logging information from the sensors and ADC by the press of a second button
- Reset the system time
- Change the intervals for the timing of the system time between 1,2 and 5 seconds

The use case diagram for the environment logger is shown in figure 1:

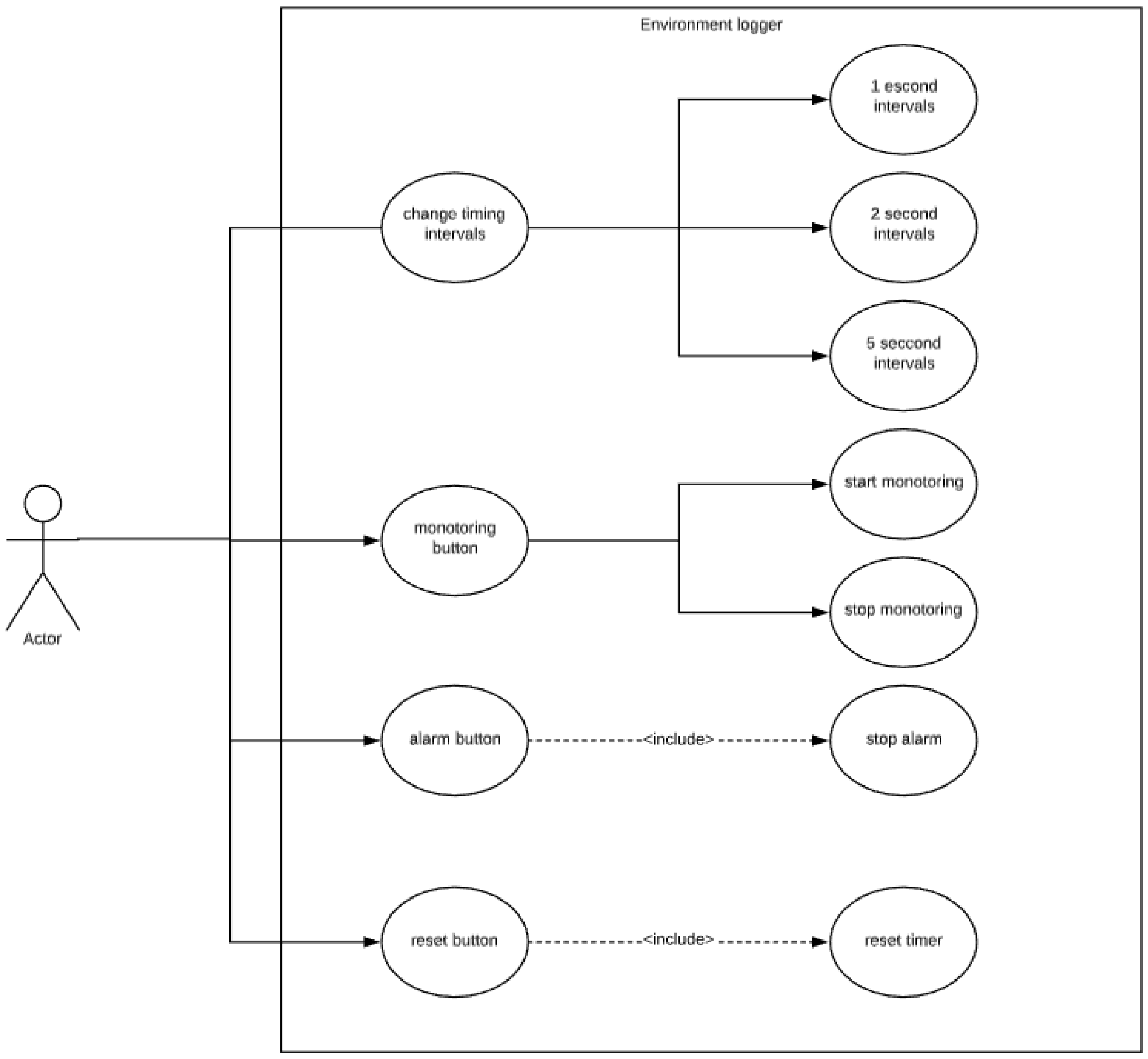


Figure 1: Use case diagram for environment logger

Specification and Design

The system has 5 states and interchanges between each state given the inputs into the system. The state chart below shows this:

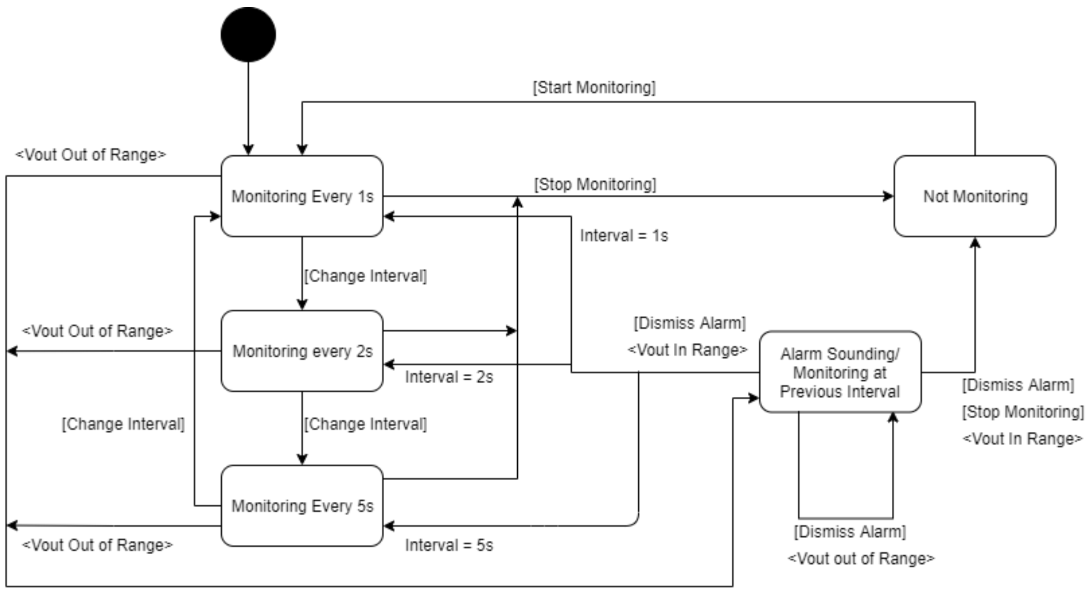


Figure 2: State Chart

It can be seen that once the system starts, it continues indefinitely and has no end. It only ends when the user manually switches it off. It is either monitoring at 1s, 2s or 5s intervals or not monitoring at all. Additionally, the alarm can be sounding while it monitors.

Coding this system was done sequentially, with one main function calling various functions that interact with the RTC and the ADC, as well as outputting them on the display. The diagram below shows this:

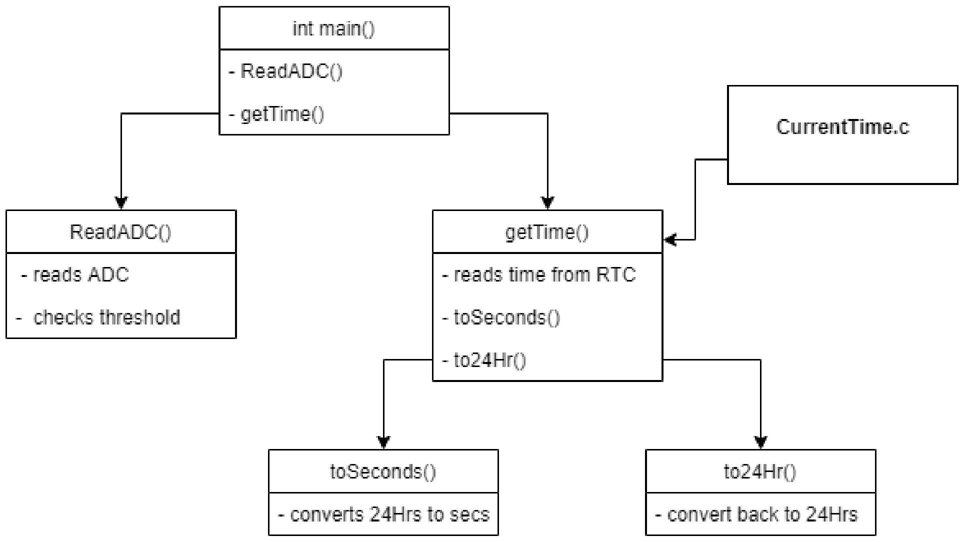


Figure 3: Functional Diagram

The circuit diagram of the system is also shown below:

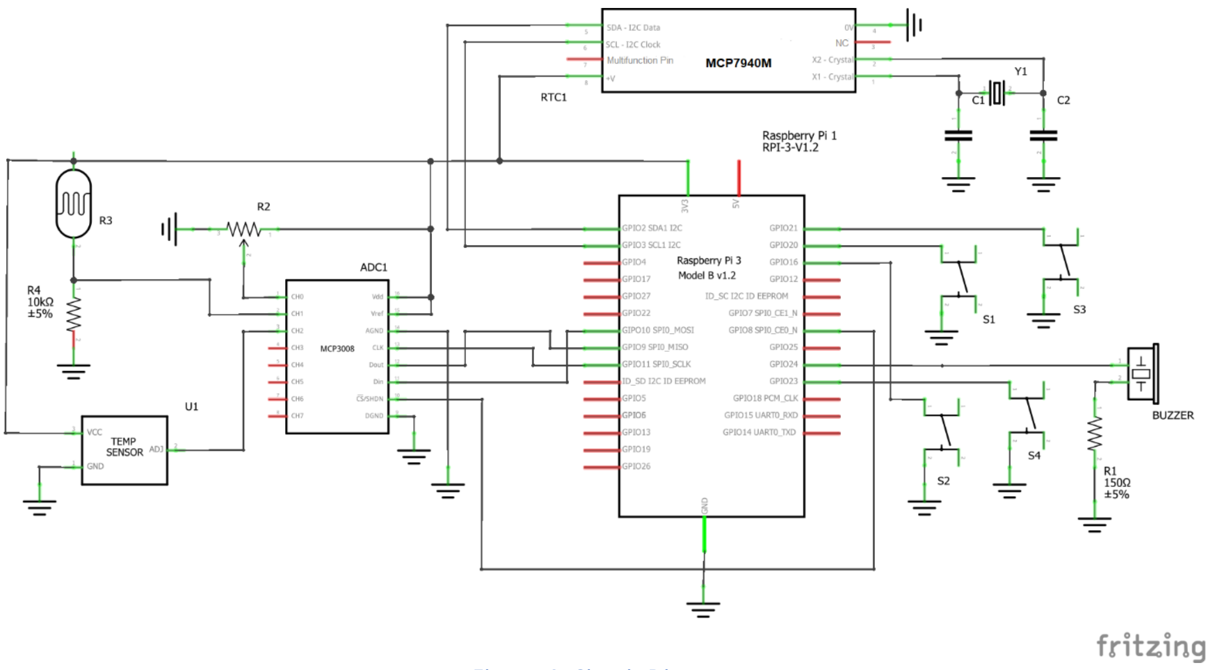


Figure 4: Circuit Diagram

Implementation

As seen from the state diagram, the system starts and goes straight into monitoring. When it is monitoring, it is reading input from the ADC and outputting into the screen. It is also constantly reading the time from the RTC. This is done in an infinite loop in the main function.

Analogue to Digital Converter

The code snippet below shows how the system obtains readings and checks if the threshold ha been crossed:

```
//read from ADC
void ReadADC(void) {
        // reads values from the ADC
        humidity = analogRead(PIN);
        light = analogRead(PIN+1);
        temperature = analogRead(PIN+2);

        //formats the values into desired thing
        humidity = humidity*(3.3/1023);
        temperature = temperature*(3.3/1023);
        temperature = ((temperature-0.05)/0.01);


        output = (light/1023)*humidity;
        //checks threshold for the alarm
        if ((output<0.65) || (output>2.65)) {
                alarmsound = true;
                alarmSecs = toSeconds(getSecs(),getMins(),getHours());

        }
}
```

Figure 5: Code to read from ADC

The function updates the global variables for the temperature, humidity and light. This makes it easier for the main function to output the readings.

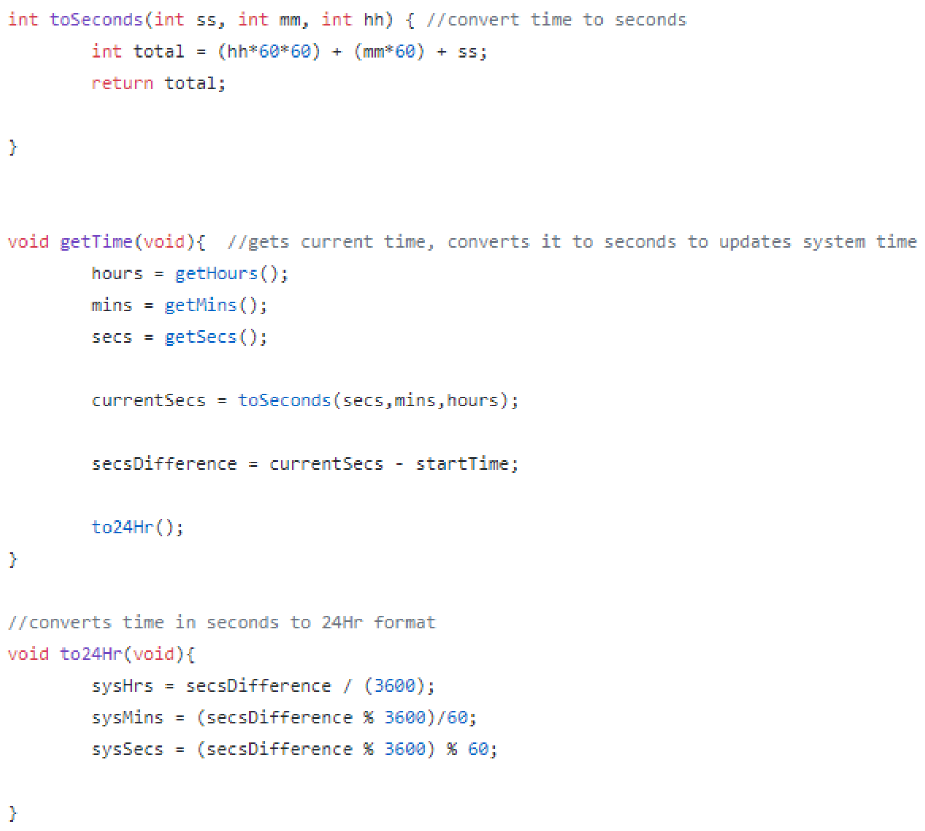
The GPIO interface library WiringPi is used in this system, and this has a set of functions specific for the ADC used in this project. This made it easier to handle the reading from the ADC and the function used is analogRead(). In order to set up the device, the function mcp3004Setup() is called. These two functions handle setting up the SPI interface between the devices as well as communication, including the control bits sent to the ADC telling it which pins to be read from.

Real Time Clock

The RTC was first set up using the kernel driver, and as such the pi's operating system handles connection between the two devices. This made it easy to obtain the time from the RTC because the code only needed to obtain the time from the system. This was done using CurrentTime.c provided in prac 3.

The system clock is calculated by first setting the time at which the system starts or when the reset button is pressed. The system clock will therefore be the difference from that set point to the current time. This difference in seconds is then calculated back to a 24Hr format to be displayed to on screen.

The following snippet shows how the time is obtained from the RTC and manipulated in order to set the system clock.



```
int toSeconds(int ss, int mm, int hh) { //convert time to seconds
        int total = (hh*60*60) + (mm*60) + ss;
        return total;

}


void getTime(void){  //gets current time, converts it to seconds to updates system time
        hours = getHours();
        mins = getMins();
        secs = getSecs();

        currentSecs = toSeconds(secs,mins,hours);

        secsDifference = currentSecs - startTime;

        to24Hr();
}

//converts time in seconds to 24Hr format
void to24Hr(void){
        sysHrs = secsDifference / (3600);
        sysMins = (secsDifference % 3600)/60;
        sysSecs = (secsDifference % 3600) % 60;

}
```

Figure 6: Interacting with the time

Buttons and Interrupts

Each of the buttons were set up using interrupts and debouncing. Each button’s interrupt service routine changed the global variables that the main function and other subfunctions dealt with. The main function does not call any of the interrupt functions which is why they are not connected in the class diagram. The buttons were used to change the monitoring states of the system. The code snippets below show two of these buttons implementations.

```
//stops the alarm from going off
void alarm_stop(void){
        long interrupttime = millis();
        if (interrupttime - lastInterruptTime > 200){
                printf("Stopping Alarm\n");
                alarmsound = false;
                digitalWrite(BUZZER,LOW);
                interval = 1000;
                }
        lastInterruptTime = interrupttime;
}
```

Figure 7: Stopping the Alarm

```
//resets the time
void reset(void){
        long interrupttime = millis();
        if (interrupttime - lastInterruptTime > 200){
                system("clear");
                printf("resetting\n");
                startTime = toSeconds(getSecs()+1,getMins(),getHours());
                }
        lastInterruptTime = interrupttime;
}
```

Figure 8: Resetting the system timer

```
//changes the interval at which data is measured
void changeInterval(void){
        long interrupttime = millis();
        if (interrupttime - lastInterruptTime > 200){
                if (interval == 5000){
                printf("reading every 1 second\n");
                interval = 1000;
                sysInterval = 1;
                }
                else if (interval == 1000){
                printf("reading every 2 second\n");
                interval = 2000;
                sysInterval = 2;
                }

                else {
                printf("reading every 5 second\n");
                interval = 5000;
                sysInterval = 5;
                }
        }
        lastInterruptTime = interrupttime;
}
```

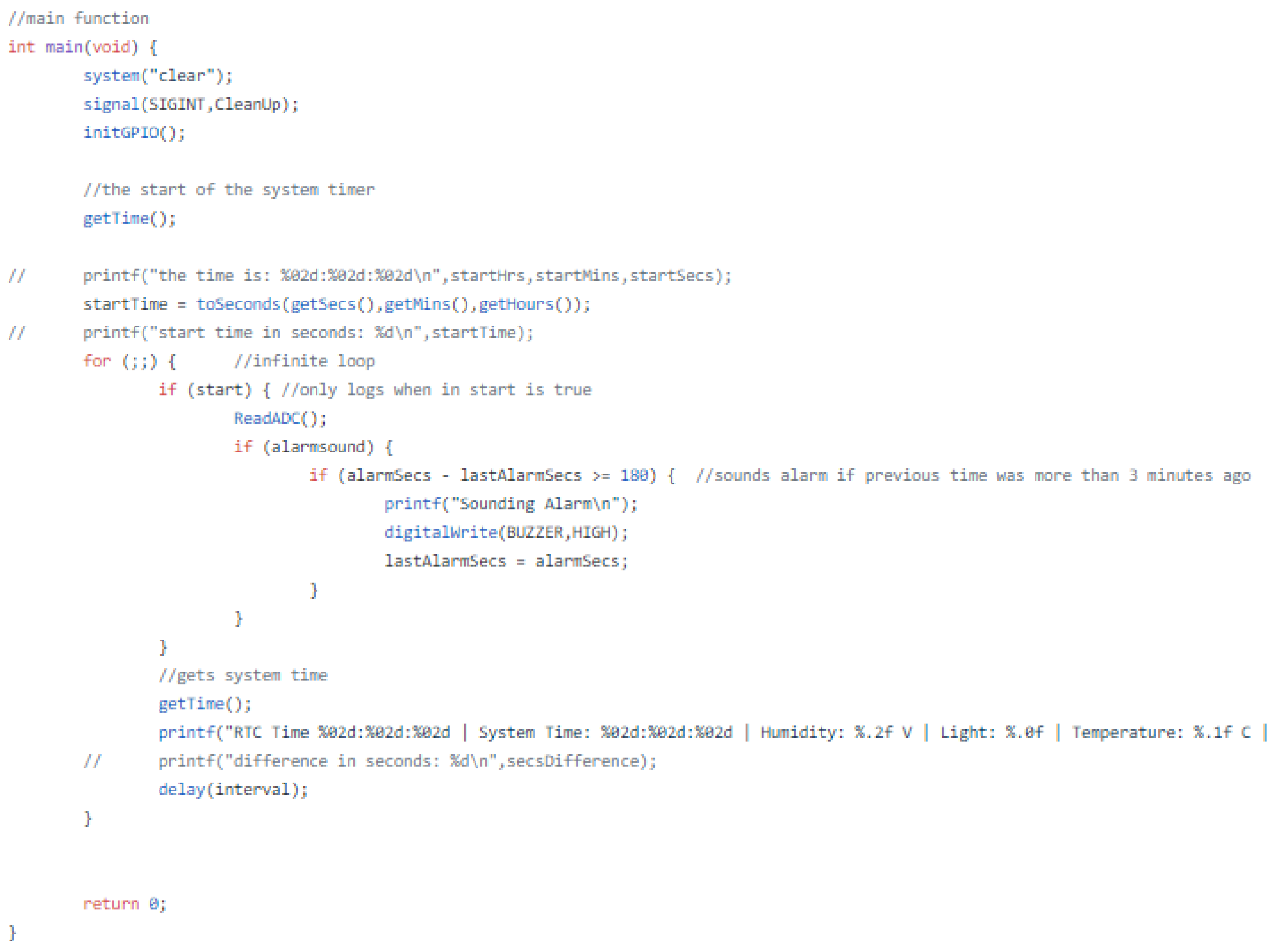
Figure 9: Changing Monitoring Intervals

```
//toggles monitoring
void start_stop_isr(void){
        long interrupttime = millis();
        if (interrupttime - lastInterruptTime > 200){
                if(start == true){
                        printf("Monitoring Stopped\n");
                        start = false;
                }
                else{
                        printf("Monitoring Started\n");
                        start = true;
                }
        }
        lastInterruptTime = interrupttime;
}
```

Figure 10: Toggling System Monitoring

Main Function

The main function is the backbone of the program, and outputs all the data. It has an infinite loop to keep monitoring the system. The code is shown below:



```
//main function
int main(void) {
        system("clear");
        signal(SIGINT,CleanUp);
        initGPIO();

        //the start of the system timer
        getTime();

//      printf("the time is: %02d:%02d:%02d\n",startHrs,startMins,startSecs);
        startTime = toSeconds(getSecs(),getMins(),getHours());
//      printf("start time in seconds: %d\n",startTime);
        for (;;) {      //infinite loop
                if (start) { //only logs when in start is true
                        ReadADC();
                        if (alarmsound) {
                                if (alarmSecs - lastAlarmSecs >= 180) {  //sounds alarm if previous time was more than 3 minutes ago
                                        printf("Sounding Alarm\n");
                                        digitalWrite(BUZZER,HIGH);
                                        lastAlarmSecs = alarmSecs;
                                }
                        }
                }
                //gets system time
                getTime();
                printf("RTC Time %02d:%02d:%02d | System Time: %02d:%02d:%02d | Humidity: %.2f V | Light: %.0f | Temperature: %.1f C |
        //      printf("difference in seconds: %d\n",secsDifference);
                delay(interval);
        }


        return 0;
}
```

Figure 11: Main Function

Note: The line showing the logging output is long and overflows and is not shown

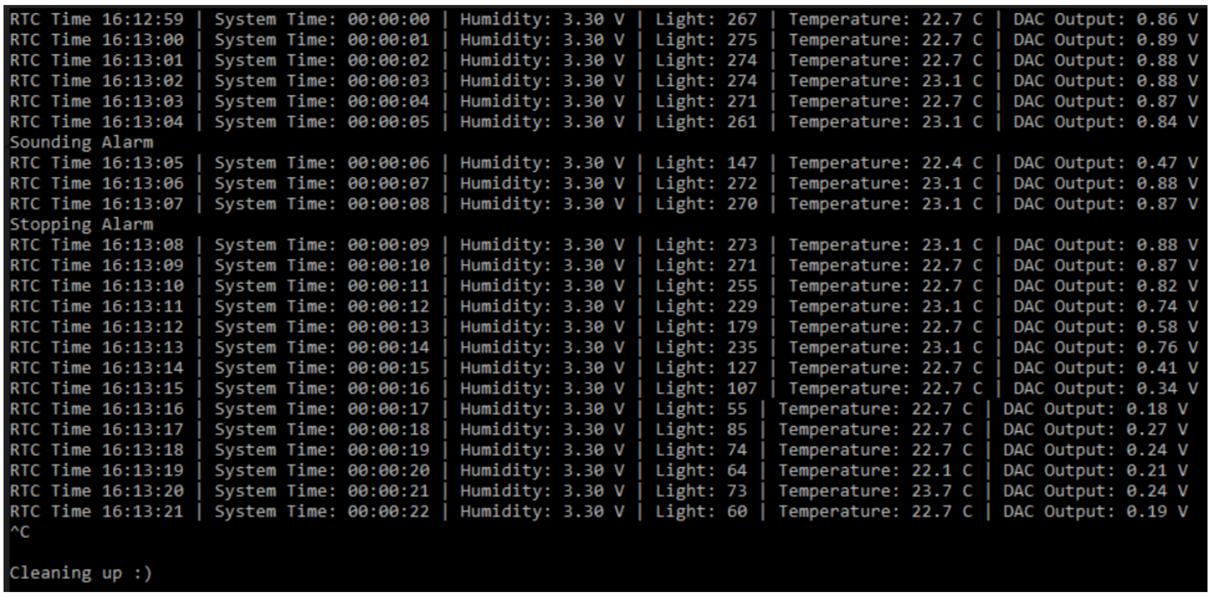
The main function also sounds the alarm if the flag set by the threshold being crossed is true and if the previous alarm rang more than 3 minutes ago.

Validation and performance

To test the validity of the logger, we tested the four use cases mentioned in figure 1. The test cases follow below but to ensure the validity of the experiment, only the light parameter was changed.

Alarm

A snippet of the alarm functionality is shown in figure 12.



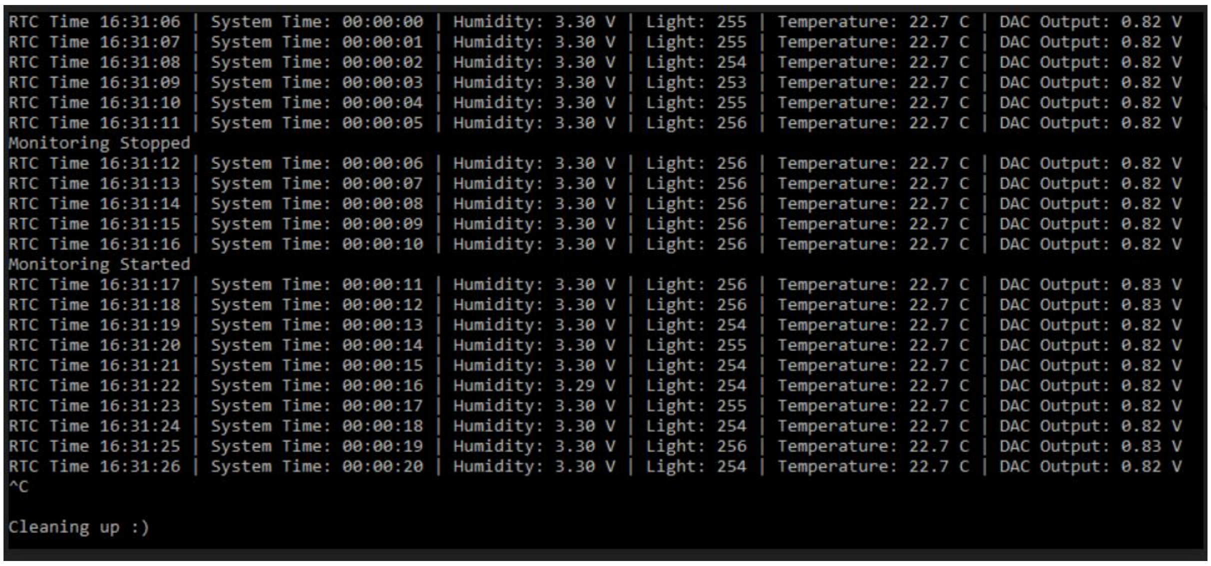
```
RTC Time 16:12:59 | System Time: 00:00:00 | Humidity: 3.30 V | Light: 267 | Temperature: 22.7 C | DAC Output: 0.86 V
RTC Time 16:13:00 | System Time: 00:00:01 | Humidity: 3.30 V | Light: 275 | Temperature: 22.7 C | DAC Output: 0.89 V
RTC Time 16:13:01 | System Time: 00:00:02 | Humidity: 3.30 V | Light: 274 | Temperature: 22.7 C | DAC Output: 0.88 V
RTC Time 16:13:02 | System Time: 00:00:03 | Humidity: 3.30 V | Light: 274 | Temperature: 23.1 C | DAC Output: 0.88 V
RTC Time 16:13:03 | System Time: 00:00:04 | Humidity: 3.30 V | Light: 271 | Temperature: 22.7 C | DAC Output: 0.87 V
RTC Time 16:13:04 | System Time: 00:00:05 | Humidity: 3.30 V | Light: 261 | Temperature: 23.1 C | DAC Output: 0.84 V
Sounding Alarm
RTC Time 16:13:05 | System Time: 00:00:06 | Humidity: 3.30 V | Light: 147 | Temperature: 22.4 C | DAC Output: 0.47 V
RTC Time 16:13:06 | System Time: 00:00:07 | Humidity: 3.30 V | Light: 272 | Temperature: 23.1 C | DAC Output: 0.88 V
RTC Time 16:13:07 | System Time: 00:00:08 | Humidity: 3.30 V | Light: 270 | Temperature: 23.1 C | DAC Output: 0.87 V
Stopping Alarm
RTC Time 16:13:08 | System Time: 00:00:09 | Humidity: 3.30 V | Light: 273 | Temperature: 23.1 C | DAC Output: 0.88 V
RTC Time 16:13:09 | System Time: 00:00:10 | Humidity: 3.30 V | Light: 271 | Temperature: 22.7 C | DAC Output: 0.87 V
RTC Time 16:13:10 | System Time: 00:00:11 | Humidity: 3.30 V | Light: 255 | Temperature: 22.7 C | DAC Output: 0.82 V
RTC Time 16:13:11 | System Time: 00:00:12 | Humidity: 3.30 V | Light: 229 | Temperature: 23.1 C | DAC Output: 0.74 V
RTC Time 16:13:12 | System Time: 00:00:13 | Humidity: 3.30 V | Light: 179 | Temperature: 22.7 C | DAC Output: 0.58 V
RTC Time 16:13:13 | System Time: 00:00:14 | Humidity: 3.30 V | Light: 235 | Temperature: 23.1 C | DAC Output: 0.76 V
RTC Time 16:13:14 | System Time: 00:00:15 | Humidity: 3.30 V | Light: 127 | Temperature: 22.7 C | DAC Output: 0.41 V
RTC Time 16:13:15 | System Time: 00:00:16 | Humidity: 3.30 V | Light: 107 | Temperature: 22.7 C | DAC Output: 0.34 V
RTC Time 16:13:16 | System Time: 00:00:17 | Humidity: 3.30 V | Light: 55 | Temperature: 22.7 C | DAC Output: 0.18 V
RTC Time 16:13:17 | System Time: 00:00:18 | Humidity: 3.30 V | Light: 85 | Temperature: 22.7 C | DAC Output: 0.27 V
RTC Time 16:13:18 | System Time: 00:00:19 | Humidity: 3.30 V | Light: 74 | Temperature: 22.7 C | DAC Output: 0.24 V
RTC Time 16:13:19 | System Time: 00:00:20 | Humidity: 3.30 V | Light: 64 | Temperature: 22.1 C | DAC Output: 0.21 V
RTC Time 16:13:20 | System Time: 00:00:21 | Humidity: 3.30 V | Light: 73 | Temperature: 23.7 C | DAC Output: 0.24 V
RTC Time 16:13:21 | System Time: 00:00:22 | Humidity: 3.30 V | Light: 60 | Temperature: 22.7 C | DAC Output: 0.19 V
^C

Cleaning up :)
```

Figure 12: Alarm validity

It can be seen that as the light conditions become poorer, the DAC output voltage decreases to below the threshold of 0.65 V and the alarm sounds. The button to dismiss the alarm is then pressed and the buzzer stops sounding. Thereafter, even after the output voltage drops below the threshold the alarm does not sound. This fulfils the specification that the alarm should only ring 3 minutes after an alarm has already sounded.

Monitoring

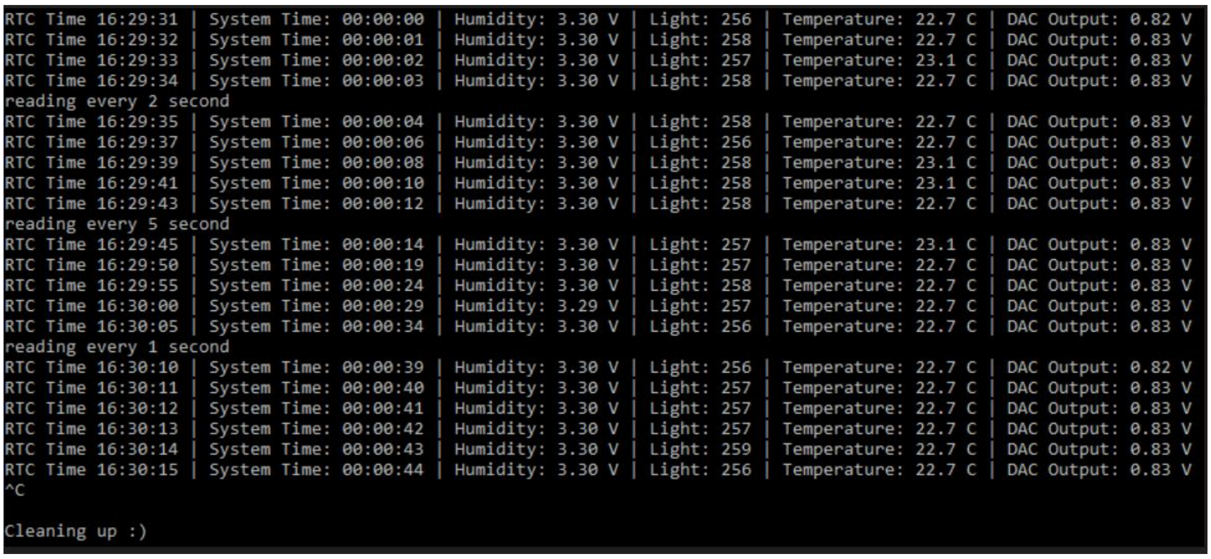
From figure 13: before the monitoring is stopped, the value of the light is changing slightly. However, after button is pressed the value of light that was most recently recorded remains constant until the nest button press. When the monitoring button is pressed again, the values change yet again.



```
RTC Time 16:31:06 | System Time: 00:00:00 | Humidity: 3.30 V | Light: 255 | Temperature: 22.7 C | DAC Output: 0.82 V
RTC Time 16:31:07 | System Time: 00:00:01 | Humidity: 3.30 V | Light: 255 | Temperature: 22.7 C | DAC Output: 0.82 V
RTC Time 16:31:08 | System Time: 00:00:02 | Humidity: 3.30 V | Light: 254 | Temperature: 22.7 C | DAC Output: 0.82 V
RTC Time 16:31:09 | System Time: 00:00:03 | Humidity: 3.30 V | Light: 253 | Temperature: 22.7 C | DAC Output: 0.82 V
RTC Time 16:31:10 | System Time: 00:00:04 | Humidity: 3.30 V | Light: 255 | Temperature: 22.7 C | DAC Output: 0.82 V
RTC Time 16:31:11 | System Time: 00:00:05 | Humidity: 3.30 V | Light: 256 | Temperature: 22.7 C | DAC Output: 0.82 V
Monitoring Stopped
RTC Time 16:31:12 | System Time: 00:00:06 | Humidity: 3.30 V | Light: 256 | Temperature: 22.7 C | DAC Output: 0.82 V
RTC Time 16:31:13 | System Time: 00:00:07 | Humidity: 3.30 V | Light: 256 | Temperature: 22.7 C | DAC Output: 0.82 V
RTC Time 16:31:14 | System Time: 00:00:08 | Humidity: 3.30 V | Light: 256 | Temperature: 22.7 C | DAC Output: 0.82 V
RTC Time 16:31:15 | System Time: 00:00:09 | Humidity: 3.30 V | Light: 256 | Temperature: 22.7 C | DAC Output: 0.82 V
RTC Time 16:31:16 | System Time: 00:00:10 | Humidity: 3.30 V | Light: 256 | Temperature: 22.7 C | DAC Output: 0.82 V
Monitoring Started
RTC Time 16:31:17 | System Time: 00:00:11 | Humidity: 3.30 V | Light: 256 | Temperature: 22.7 C | DAC Output: 0.83 V
RTC Time 16:31:18 | System Time: 00:00:12 | Humidity: 3.30 V | Light: 256 | Temperature: 22.7 C | DAC Output: 0.83 V
RTC Time 16:31:19 | System Time: 00:00:13 | Humidity: 3.30 V | Light: 254 | Temperature: 22.7 C | DAC Output: 0.82 V
RTC Time 16:31:20 | System Time: 00:00:14 | Humidity: 3.30 V | Light: 255 | Temperature: 22.7 C | DAC Output: 0.82 V
RTC Time 16:31:21 | System Time: 00:00:15 | Humidity: 3.30 V | Light: 254 | Temperature: 22.7 C | DAC Output: 0.82 V
RTC Time 16:31:22 | System Time: 00:00:16 | Humidity: 3.29 V | Light: 254 | Temperature: 22.7 C | DAC Output: 0.82 V
RTC Time 16:31:23 | System Time: 00:00:17 | Humidity: 3.30 V | Light: 255 | Temperature: 22.7 C | DAC Output: 0.82 V
RTC Time 16:31:24 | System Time: 00:00:18 | Humidity: 3.30 V | Light: 254 | Temperature: 22.7 C | DAC Output: 0.82 V
RTC Time 16:31:25 | System Time: 00:00:19 | Humidity: 3.30 V | Light: 256 | Temperature: 22.7 C | DAC Output: 0.83 V
RTC Time 16:31:26 | System Time: 00:00:20 | Humidity: 3.30 V | Light: 254 | Temperature: 22.7 C | DAC Output: 0.82 V
^C

Cleaning up :)
```

Figure 13: Monitoring validity

Time intervals



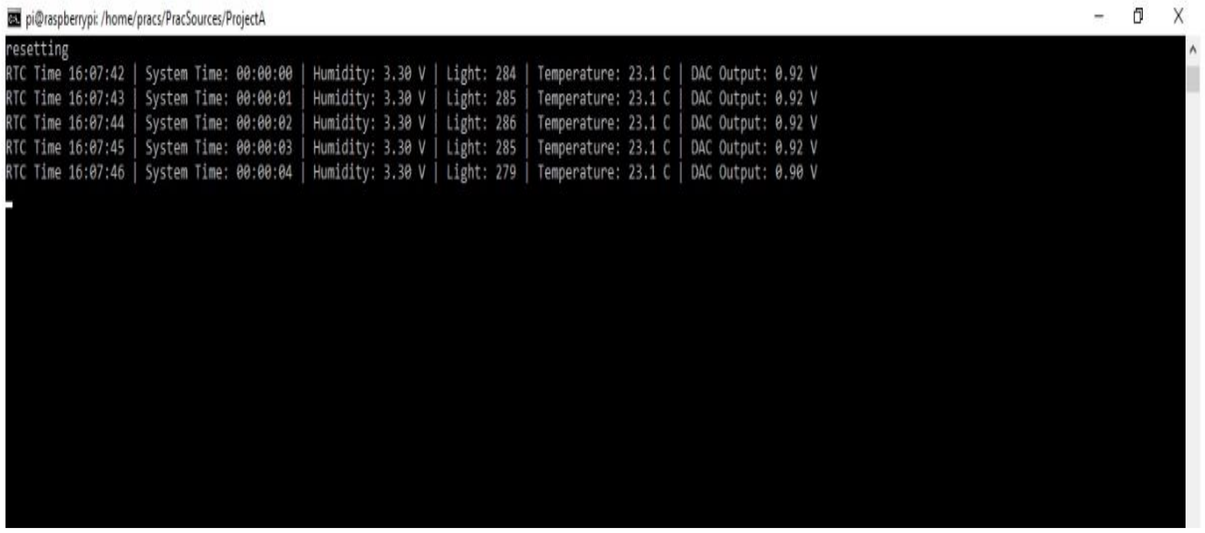
```
RTC Time 16:29:31 | System Time: 00:00:00 | Humidity: 3.30 V | Light: 256 | Temperature: 22.7 C | DAC Output: 0.82 V
RTC Time 16:29:32 | System Time: 00:00:01 | Humidity: 3.30 V | Light: 258 | Temperature: 22.7 C | DAC Output: 0.83 V
RTC Time 16:29:33 | System Time: 00:00:02 | Humidity: 3.30 V | Light: 257 | Temperature: 23.1 C | DAC Output: 0.83 V
RTC Time 16:29:34 | System Time: 00:00:03 | Humidity: 3.30 V | Light: 258 | Temperature: 22.7 C | DAC Output: 0.83 V
reading every 2 second
RTC Time 16:29:35 | System Time: 00:00:04 | Humidity: 3.30 V | Light: 258 | Temperature: 22.7 C | DAC Output: 0.83 V
RTC Time 16:29:37 | System Time: 00:00:06 | Humidity: 3.30 V | Light: 256 | Temperature: 22.7 C | DAC Output: 0.83 V
RTC Time 16:29:39 | System Time: 00:00:08 | Humidity: 3.30 V | Light: 258 | Temperature: 23.1 C | DAC Output: 0.83 V
RTC Time 16:29:41 | System Time: 00:00:10 | Humidity: 3.30 V | Light: 258 | Temperature: 23.1 C | DAC Output: 0.83 V
RTC Time 16:29:43 | System Time: 00:00:12 | Humidity: 3.30 V | Light: 258 | Temperature: 22.7 C | DAC Output: 0.83 V
reading every 5 second
RTC Time 16:29:45 | System Time: 00:00:14 | Humidity: 3.30 V | Light: 257 | Temperature: 23.1 C | DAC Output: 0.83 V
RTC Time 16:29:50 | System Time: 00:00:19 | Humidity: 3.30 V | Light: 257 | Temperature: 22.7 C | DAC Output: 0.83 V
RTC Time 16:29:55 | System Time: 00:00:24 | Humidity: 3.30 V | Light: 258 | Temperature: 22.7 C | DAC Output: 0.83 V
RTC Time 16:30:00 | System Time: 00:00:29 | Humidity: 3.29 V | Light: 257 | Temperature: 22.7 C | DAC Output: 0.83 V
RTC Time 16:30:05 | System Time: 00:00:34 | Humidity: 3.30 V | Light: 256 | Temperature: 22.7 C | DAC Output: 0.83 V
reading every 1 second
RTC Time 16:30:10 | System Time: 00:00:39 | Humidity: 3.30 V | Light: 256 | Temperature: 22.7 C | DAC Output: 0.82 V
RTC Time 16:30:11 | System Time: 00:00:40 | Humidity: 3.30 V | Light: 257 | Temperature: 22.7 C | DAC Output: 0.83 V
RTC Time 16:30:12 | System Time: 00:00:41 | Humidity: 3.30 V | Light: 257 | Temperature: 22.7 C | DAC Output: 0.83 V
RTC Time 16:30:13 | System Time: 00:00:42 | Humidity: 3.30 V | Light: 257 | Temperature: 22.7 C | DAC Output: 0.83 V
RTC Time 16:30:14 | System Time: 00:00:43 | Humidity: 3.30 V | Light: 259 | Temperature: 22.7 C | DAC Output: 0.83 V
RTC Time 16:30:15 | System Time: 00:00:44 | Humidity: 3.30 V | Light: 256 | Temperature: 22.7 C | DAC Output: 0.83 V
^C

Cleaning up :)
```

Figure 14: Interval validity

Figure 14: shows the working of the frequency switching buttons. It can be seen that as the intervals change, the system time changes intervals change with the specified frequency.

Reset

From figure 15: it can clearly be seen that when the reset button is pressed that the timer resets to zero and the console is cleared as well.



```
pi@raspberrypi: /home/pracs/PracSources/ProjectA
resetting
RTC Time 16:07:42 | System Time: 00:00:00 | Humidity: 3.30 V | Light: 284 | Temperature: 23.1 C | DAC Output: 0.92 V
RTC Time 16:07:43 | System Time: 00:00:01 | Humidity: 3.30 V | Light: 285 | Temperature: 23.1 C | DAC Output: 0.92 V
RTC Time 16:07:44 | System Time: 00:00:02 | Humidity: 3.30 V | Light: 286 | Temperature: 23.1 C | DAC Output: 0.92 V
RTC Time 16:07:45 | System Time: 00:00:03 | Humidity: 3.30 V | Light: 285 | Temperature: 23.1 C | DAC Output: 0.92 V
RTC Time 16:07:46 | System Time: 00:00:04 | Humidity: 3.30 V | Light: 279 | Temperature: 23.1 C | DAC Output: 0.90 V
```

Figure 15: Reset Validity

Conclusion

As can be seen from the above, our model of the environment logger works well in accordance to the hardware and software specifications. The environment logger can detect light, humidity and temperature changes and display them to the user in real time. Furthermore, the user can easily interact with the device and select different functioning by the press of the buttons. This includes starting and stopping monitoring of the logger, changing of the time intervals for data collection and the system time and finally resetting the device's system time.

In addition, it indicates an error by ringing a buzzer when the output voltage falls outside of a specified range and the user is notified via the IOT platform Blynk. This alarm can also be switched off by a button but will only ring in 3-minute intervals.

This project may be potentially useful because it is easily implemented, and it is not very expensive. Due to this reason, many people in the respective field can implement this into their projects and modify it to their own liking or to accomplish even better tasks and perform better.

With this, further improvements to this device could be in using better components such as using a more accurate temperature sensor, an ADC with a higher resolution and an actual humidity sensor. This will result in even more accurate results. Furthermore, other sensors such as wind and pressure sensors can be added to the device to make it a full weather analyser.