Student Number: _____**SLYTAR001 & EBRAZH001**___

U N I V E R S I T Y O F C A P E T O W N
Department of Electrical Engineering

EEE3096S – Embedded Systems 2
Project 2019 Report

Tareeq Saley & Azhar Ebrahim

SLYTAR001 & EBRAZH001

12/10/2019

Introduction

The purpose of this report is to design and analyse an environment logger that was created as part of the EEE3096S project task. The environment logger is to be used in a private greenhouse to monitor the time of the day, temperature, light levels and humidity.

For monitoring the time of day, an RTC clock was used as the Pi does not have its own clock. The i2c library was used to configure the device. Thereafter, a MCP9700A temperature sensor was used to track the ambient temperature. Next, a potentiometer was used to mimic the humidity levels. Finally, a light dependent resistor was used to monitor the light intensity. The digital outputs of the temperature sensor, light dependent resistor and potentiometer were obtained using an MCP3008 analogue-to-digital converter. These outputs, together with control functionality related to them, were displayed using the Blynk app as an IoT.

This report starts by listing the requirements of the project. It then goes on to provide details about the specifications and design. This is followed by snippets of the code related to its implementataion in C. Snapshots of the results that were obtained are then for validation and to test the performance. Lastly, the report is concluded with a brief summary that includes an analysis of how successful the project was.

Requirements

UML Use Case Diagram

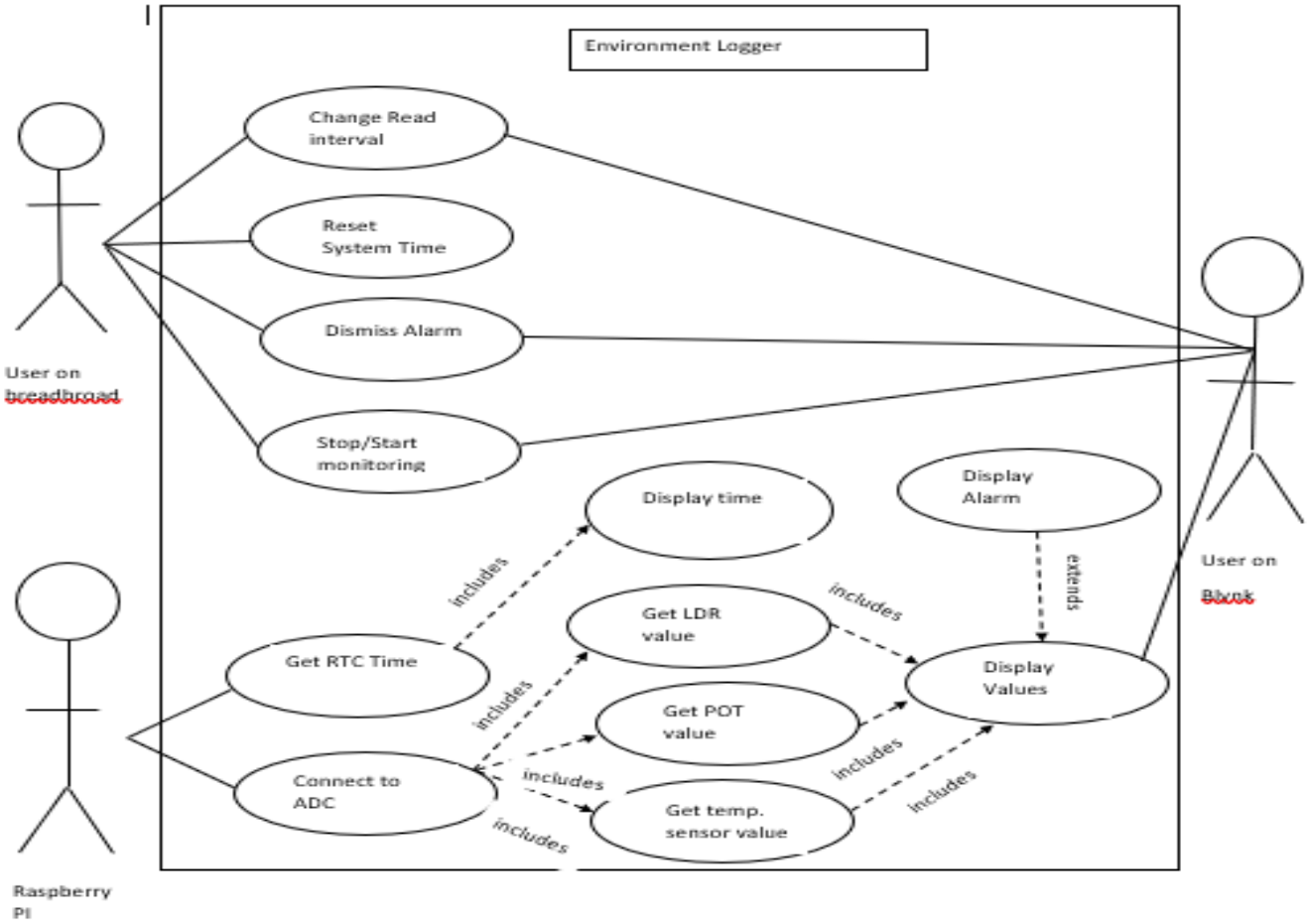

Figure 1: UML Use Case Diagram

Specification and Design

State Diagram

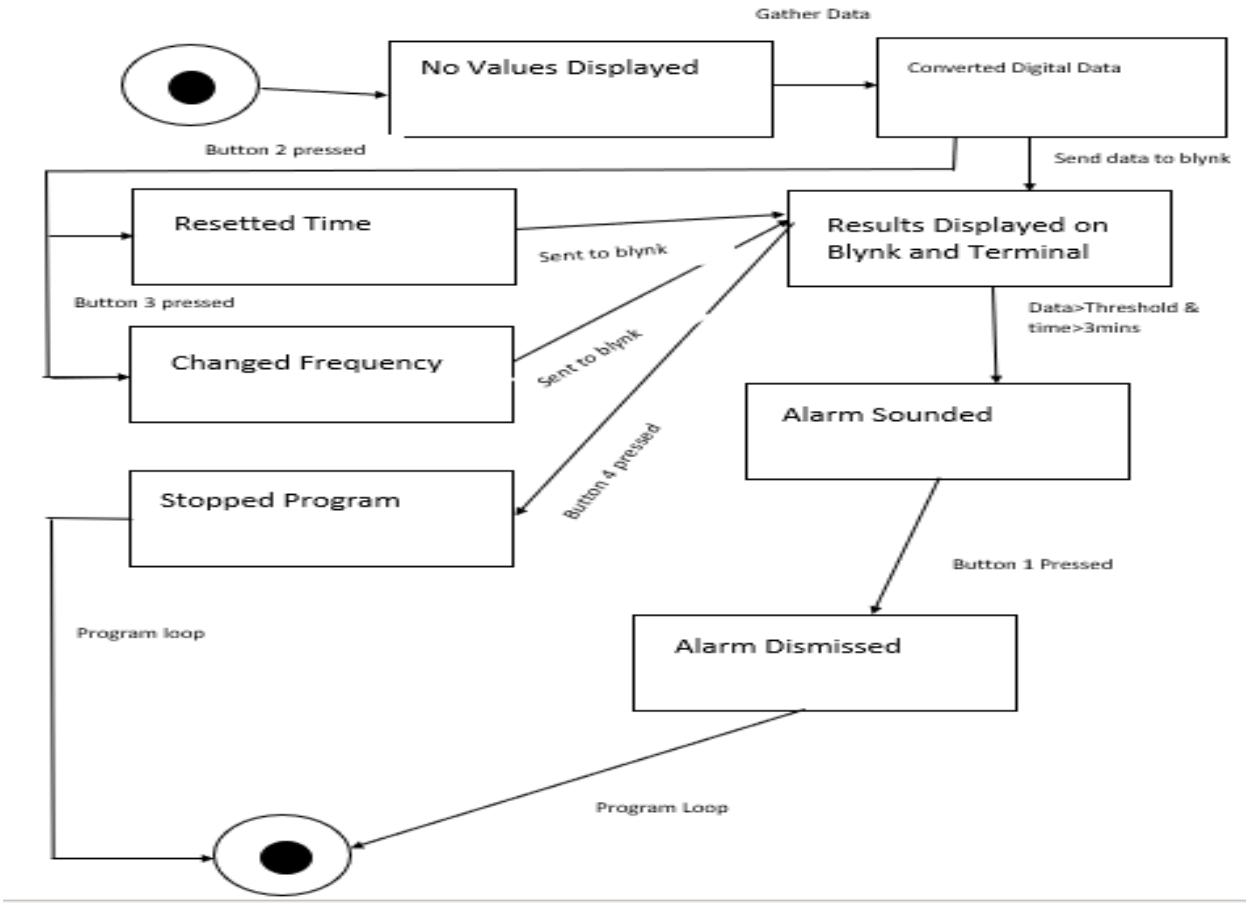

Figure 2: State diagram for design

UML Class Diagram

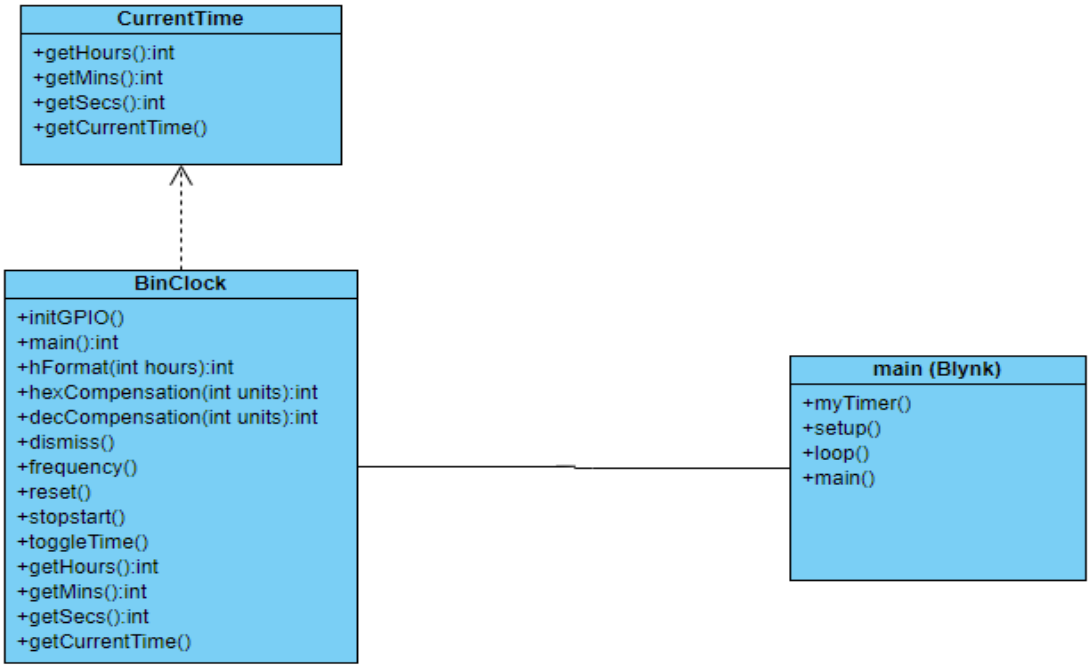

Figure 3: UML Class Diagram

Circuit Diagram

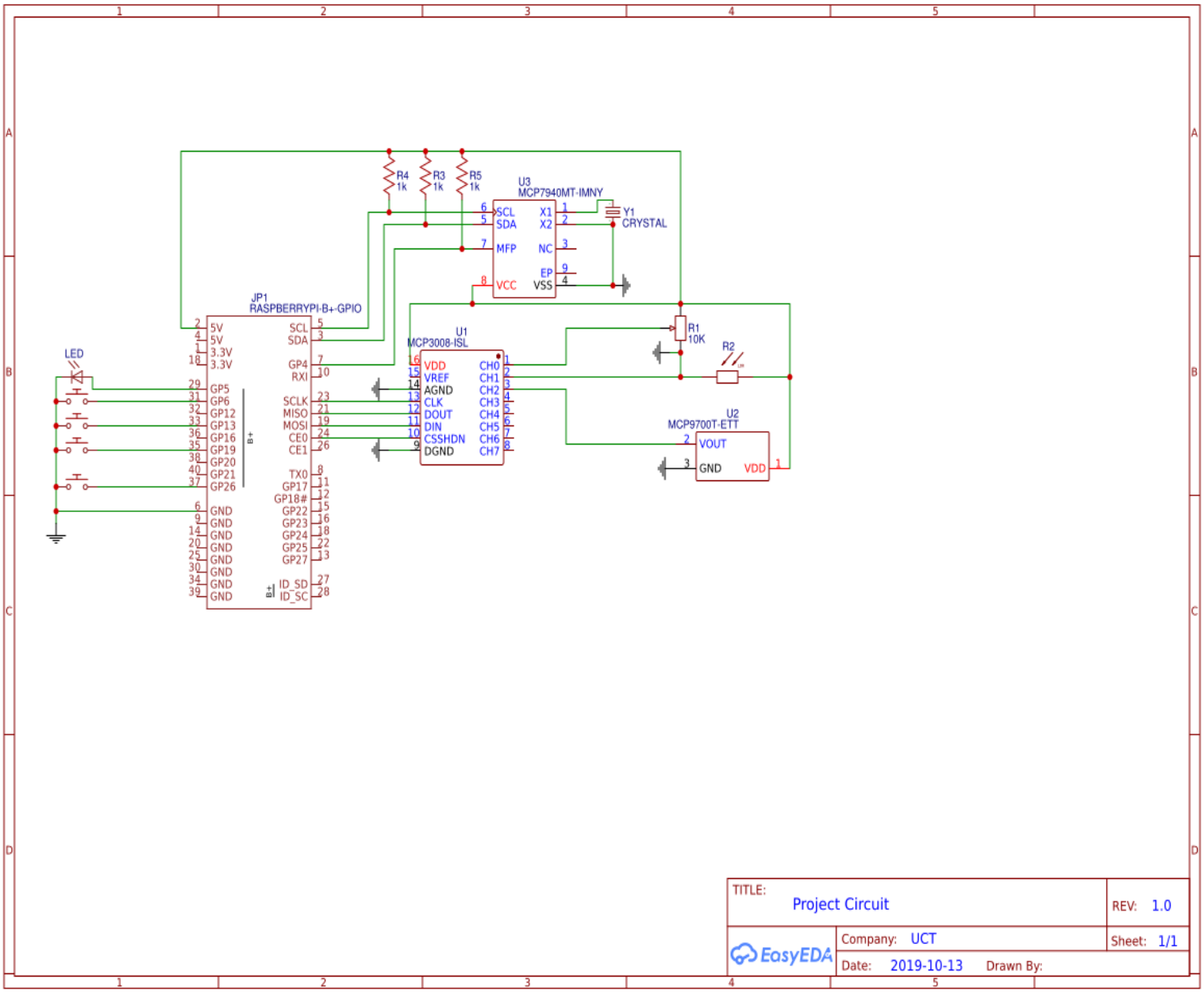

Figure 4: Circuit schematic for design

Implementation

The code in figure 1 describes how the digital outputs of the light dependent resistor, temperature sensor and potentiometer were obtained using the ADC and then displayed. It also shows how the the humidity value was calculated to give an output between 0 and 3.3V as well as the calculation related to the DAC output which is used to trigger the alarm. The readings were only obtained from the ADC if the environment logger was not in its paused state. The alarm was only triggered if the DAC output was between 0.65V and 2.65V, this alarm was indicated by flashing an LED which was configured to GPIO21. The formula used to calculate the DAC output is given below,

$$DAC = \frac{LightReading \times HumidityVoltageReading}{1023}$$

The alarm could only be triggered again 3 minutes after it had previously been dismissed. The frequency at which the readings were taken was defined by the 'freq' variable, which could be changed to 1s,2s or 5s.

```
if (pause == 0)
{
humidity = analogRead(MCP3008_BASE);
float hum = humidity*3.3/1024;
ldr = analogRead(MCP3008_BASE +1);
float dac = hum*ldr/1024;
temperature = analogRead(MCP3008_BASE+2)*0.18;
if (x>0.65 && x<2.65)
{

        if  (count>= 180)
        {
        digitalWrite(21,1);
        }
}

printf("The current time is: %x:%x:%x\n", HH, MM, SS);
printf("Humidity: %fV Intensity: %d DACOut: %f Temp: %dC\n" , hum, ldr, dac, temperature);
}
//using a delay to make our program "less CPU hungry"
delay(freq*1000); //milliseconds

}
```

Figure 5: Code to obtain and display digital outputs

Figure 2 describes the code used to dismiss the alarm once it has been triggered. Figure 3 allows the user to change how often the readings are logged from the ADC. Figure 4 resets the system time and clears the console completely. Figure 5 pauses or restarts the logging if triggered.

```
void dismiss(void){
        long interruptTime = millis();
        if (interruptTime - lastInterruptTime>200)
        {
                printf("Button is Pressed");
                if (count>=180)
                {
                digitalWrite(21,0);
                count = 0;
                }
        }
```

Figure 6: Code to dismiss alarm using switch button

```
void frequency(void){
       long interruptTime = millis();
       if (interruptTime - lastInterruptTime>200)
       {
              printf("Button 3 is Pressed");
              if (freq==1)
              {
                  freq = 2;
              }
              else if (freq==2)
              {
                  freq = 5;
              }
              else if (freq==5)
              {
                  freq = 1;
              }
              printf("freq, %x\n", freq);
       }
       lastInterruptTime = interruptTime;
}
```

Figure 7: Code to control frequency of logging using switch button

```
void reset(void){
        long interruptTime = millis();
        if (interruptTime - lastInterruptTime>200)
        {
                printf("Button 2 is Pressed");
                systim = 0;
                system("clear");
        }
        lastInterruptTime = interruptTime;
}
```

Figure 8: Code to reset system time and clears the console using switch button

```
void stopstart(void){
        long interruptTime = millis();
        if (interruptTime - lastInterruptTime>200)
        {
                if (pause == 1)
                {
                        pause = 0;
                }
                else
                {
                        pause = 1;
                }
                printf("paused?, %x\n", pause);
        }
        lastInterruptTime = interruptTime;
}
```

Figure 9: Code to stop/start logging using switch button

Lastly, the code in figure 6 describes the environment loggers configuration to the Blynk app and its respective virtual GPIO pins.

```
void myTimer()
{
        int humidity = analogRead(MCP3008_BASE);
        float hum = humidity*3.3/1024;
        Blynk.virtualWrite(V4, hum);
//      printf("humidity :%f\n",hum);
        int ldr = analogRead(MCP3008_BASE+1);
        Blynk.virtualWrite(V6, ldr);
        int temp = analogRead(MCP3008_BASE+2) *0.18;
        Blynk.virtualWrite(V8, temp);
        float dac = hum*ldr/1024;
        Blynk.virtualWrite(V1, dac);
```

Figure 10: Code used to configure Blynk

Validation and Performance

The environment logger performs as expected. The debouncing functionality ensures that the buttons don't register multiple signals when pressed. The respective environment logger values are updated instantaneously and its respective ADC values are read depending on the frequency selected by the user. The potentiometer registers output linearly increasing/decreasing between 0V and 3.3V when adjusted. The voltage divider of the potentiometer ensures that there is minimal fluctuation in its analogue output value. The light dependent resistor gauges the light intensity rapidly when the light conditions are altered. This was confirmed by using artificial light from a mobile device. The alarm is triggered whenever the DAC value ranges between 0.65V and 2.65V, and once it has been dismissed, it can only be set off again after 3 minutes. Various resistors and capacitors were used to limit the fluctutation of the sensor components and for robustness. The following snapshots serve to illustrate

that the values displayed on the Blynk app is consistent with what the environment logger values are expected to be based on the conditions that were changed .

Under ambient conditions, before manipulating the sensor values for testing purposes, these are the values that were recorded.

```
Humidity: 1.260059V Intensity: 862 DACOut: 1.060713 Temp: 20C
```

Figure 11: Ambient condition values for environment logger

The following figures will be used to suggest how changing the respective sensor conditions affected its output.

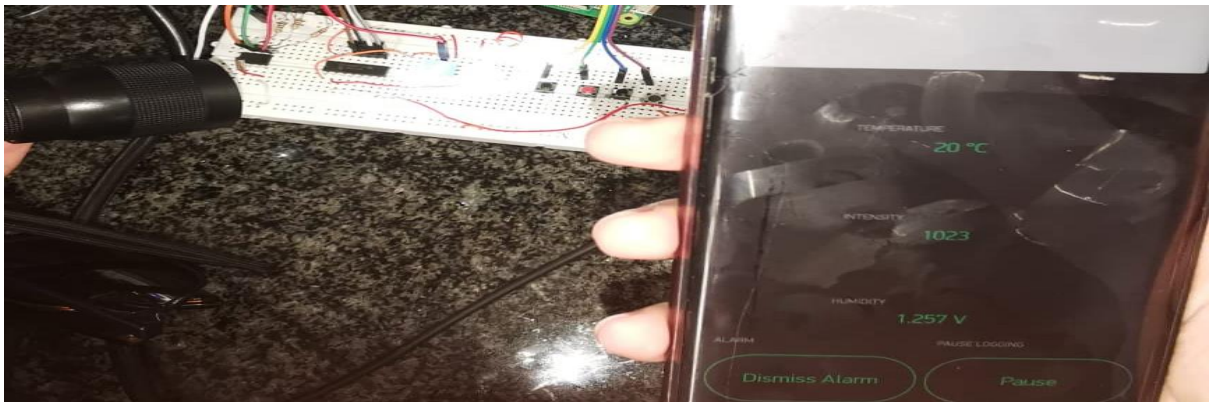

Figure 12: 1023 Light intensity when a torch is shone over the ldr

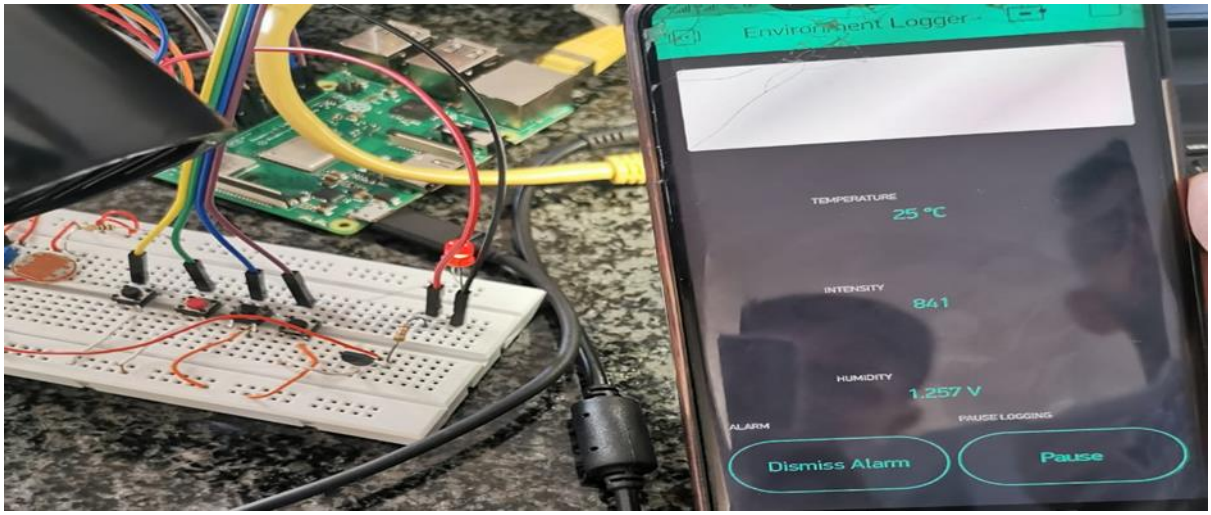

Figure 13: Temperature increases to 25 degrees when the heat of a blow-dryer is placed over the temperature sensor

Figure 14: Temperature decreases to 16 degrees when an ice cube is placed over the temperature sensor

Figure 15: Dismissing the alarm on Blynk turns LED off

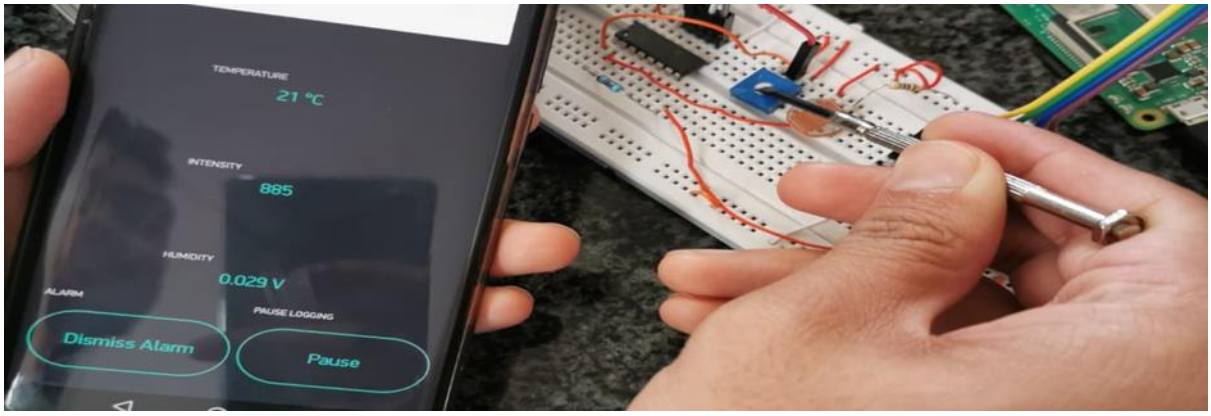

Figure 16: Humidity value at 0V when put turned anti-clockwise

Figure 17: Humidity at 3.3V when pot is turned fully clcokwise

Conclusion

The system was found to be successful as all the functions worked as expected and the deliverables of the project were met. There were no major issues found with the system after it was tested. The system runs smoothly without any unwanted delays.

This system can be considered as being a useful product with adjustments for its specific purposes. The design is simple to implement and does not require too many resources. Furthermore, the system is easy to use. A system working in this way, by having an application that's able to remotely control some of its functionality and remotely track its sensor values, can be useful for its intended purpose in a greenhouse. With the current functionality and through some added functionality, it can allow the greenhouse to maximize its harvest by finding the best location for it to placed depending on the light intensity, and managing the internal climate of the greenhouse by tracking the temperature and humidity. By being able to start or stop data remotely, unwanted alarm triggers can be dismissed easily. The ability to pause the logging of data can save a lot of energy because depending on the consistency of the external temperature and internal greenhouse climate, the need for the data to be logged can be weighed up accordingly.

The project could be improved by using more advanced and less sensitive components and by using more widgets in the Blynk app to maximize its usability and linkage to pi.

Link to Github Code

https://github.com/tareeqsaley/project/blob/master/BinClock.c