Information Technology Institute

Autonomous Greenhouse Farming Control System

Intake 43

Embedded System Track

6/26/23

Abstract

Agriculture sector in Egypt is dominated by small farms which use traditional practices that do not comply with internationally recognized standards. For example, farmers tend to overuse and misuse agricultural chemicals and use outdated technologies and tools for land preparation, irrigation, and harvesting. As a result, farmers experience increased production costs, reduced yields, decreased soil fertility, and limited marketing opportunities. Also raising plants in your home may need a daily caring which may be difficult for some people who doesn't have adequate time, or they may travel away from homes for a time, and they become worried about their plants.

The objective of this project is enhancing the agriculture process by using the technology and more Scientific methods where farmers can monitor their agriculture lands or a person who wants to monitor the plants inside his home without the need to be on their home all the time by using his mobile phone. Smart farming brings out the concept of Internet of Things (IOT) through which he monitors the data obtained from the sensors.

Table of Contents

LIST OF FIGURES

List of Abbreviations

SGH	Smart Green House
GPIO	General-purpose input/output
RX	Receive
TX	Transmit
LSW	Limit Switch
DHT	Temp & HUM Sensor
PVC	Polyvinyl chloride
GND	Ground
PWM	Pulse Width Modulation
PCB	Printed Circuit Board
RST	RESET
ENA	Enable

Final Prototype Implementation

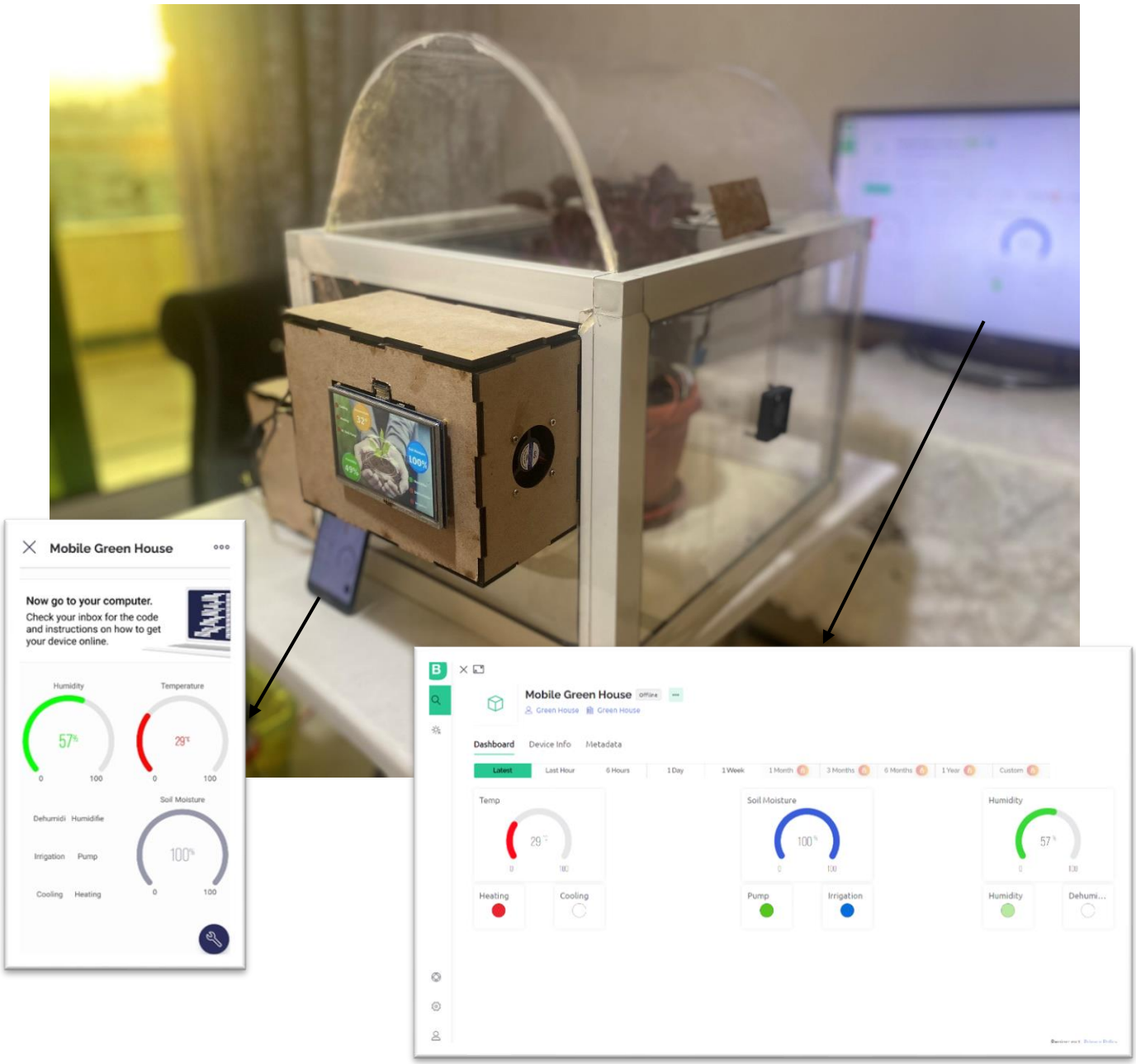


Figure 1 SGH Final Prototype

CHAPTER 1

Introduction

1.1) History of Green Houses

The idea of growing plants in environmentally controlled areas has existed since Roman times. The Roman emperor Tiberius ate a cucumber vegetable daily. The Roman gardeners used artificial methods (like the greenhouse system) of growing to have it available for his table every day of the year.

The concept of greenhouses also appeared in the Netherlands and then England in the 17th century, along with the plants. Some of these early attempts required enormous amounts of work to close at night or to winterize. There were serious problems with providing adequate and balanced heat in these early greenhouses.

Greenhouse structures adapted in the 1960s when wider sheets of polyethylene (polythene) film became widely available, Today, the Netherlands has many of the largest greenhouses in the world, some of them so vast that they can produce millions of vegetables every year.

1.2) Theory of operation

It is a structural building with different types of covering materials, such as a glass or plastic roof and frequently glass or plastic walls. It heats up because incoming visible solar radiation (for which the glass is transparent) from the sun is absorbed by plants ,soil, and other things inside the building In a smart greenhouse, crops are cultivated under controlled conditions to optimize production Technologies such as sensors and devices are used to automate the functions of the greenhouse. Smart greenhouses protect crops from diseases, pests, and harsh weather conditions, while at the same time-consuming low amount of power and allowing remote access to the farmers. A smart greenhouse is equipped with interconnected devices to control sunlight, temperature and humidity depending on the type of crop the

Figure 2 SGH Auto controlled.

control system collects data from various sensors and transmits it for analysis to reduce production losses.

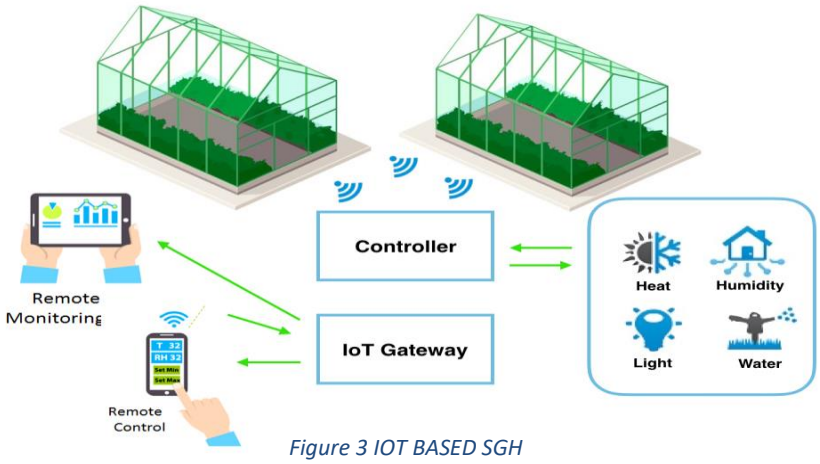


Figure 3 IOT BASED SGH

1.3) Advantages of Smart Green Houses

Smart greenhouses protect crops from diseases, pests, and harsh weather conditions, while at the same time-consuming low amount of power and allowing remote access to the farmers. A smart greenhouse is equipped with interconnected devices to control sunlight, temperature, and humidity depending on the type of crop the control system collects data from various sensors and transmits it for analysis to reduce production losses.

Figure 4 SGH Monitoring

1.4) Traditional Irrigation Method

Surface irrigation

- flooding

-oldest form of irrigation

Figure 5 surface irrigation

Problems of surface irrigation

(I) wasteful use of water.

(II) non-uniform distribution of water.

(III) excessive soil erosion.

(IV) require drainage arrangement

1.5) Modern Methods of Irrigation

I) Sprinkler irrigation

II) Drip irrigation

Sprinkler irrigation

Sprinkler irrigation is a method of applying irrigation water which is like natural rainfall. Water is distributed through a system of pipes usually by pumping. It is then sprayed into the air through sprinklers so that it breaks up into small water drops which fall to the ground.

Figure 6 Modern methods of irrigation

1.6) Benefits of using the modern irrigation methods

I) Water & electricity saving

II) use a clean renewable source of energy for irrigation (electricity)

III) provide a clean environment for the plants & improve the ability of the soil toward the harmful insects and infections

IV) Increase the productivity

Chapter 2
Hardware Design

2.1) Microcontroller ARM-Cortex-M4 (STM32F401CCU6)

The STM32F401CCU6 is a microcontroller designed and manufactured by STMicroelectronics. This microcontroller is based on the ARM Cortex-M4F processor architecture and is intended for use in applications that require high performance and low power consumption.

The STM32F401CCU6 has a clock speed of up to 84 MHz and features 256KB of flash memory, 64KB of SRAM, and a range of peripherals, including ADCs, DACs, timers, and communication interfaces such as SPI, I2C, and UART. It also includes a hardware cryptographic module for secure communication and data storage.

Figure 7 STM32F401CCU6

This microcontroller is well-suited for use in a variety of applications, including IoT devices, wearables, and smart home appliances. Its low power consumption and small form factor make it ideal for use in battery-powered devices.

Power Requirement

The STM32F401CCU6 is designed to operate on a voltage range of 1.7V to 3.6V, making it suitable for use in low-power applications. The microcontroller includes a built-in voltage regulator that can supply up to 150mA of current at a stable voltage of 3.3V.

2.2) Raspberry Pi 3 B+

The Raspberry Pi 3 Model B+ has a Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC running at 1.4GHz, 1GB LPDDR2 SDRAM, microSD card slot, Gigabit Ethernet, dual-band 802.11ac wireless networking, Bluetooth 4.2, 40-pin GPIO header, 4 USB 2.0 ports, HDMI, composite video, 3.5mm audio jack, and HDMI audio output. It can be powered via micro-USB port or GPIO header and has dimensions of 82 x 56 x 19.5 mm, weighing 50 g. It is a versatile and powerful single-board computer suitable for media centers, home automation systems, and robotics projects.

Figure 8 Raspberry Pi 3 B+

2.3) 5" LCD Touch Screen HDMI For Raspberry Pi 800*480

a 5-inch touchscreen display with a resolution of 800 x 480 and resistive touch control. It is compatible with Raspberry Pi and can be used directly in the original Raspbian system. It also supports BB Black with an Angstrom corresponding mirror, and Banana Pi/Banana Pro with Ubuntu and Raspbian corresponding mirrors. The display can be used as a computer monitor with an HDMI interface for display and a USB interface for touch. However, the touchscreen function cannot be used as a computer monitor.

Figure 9 LCD Touch Screen

2.4) Microcontroller ESP8266 (NodeMCU)

The ESP8266 is the name of a micro controller designed by Espressif Systems. It is an open source IoT platform & a self-contained Wi-Fi networking solution offering as a bridge from existing microcontroller to Wi-Fi and is also capable of running self-contained applications.

This module comes with a built in USB connector and a rich assortment of pin-outs. With a micro-USB cable, you can connect NodeMCU devkit to your laptop and flash it without any trouble, it is also breadboard friendly.

Figure 10 NodeMCU

2.5) Water Pump – Ultra-Quite Brushless DC12V – 240L/H

The AD20P-1230C is a brushless DC pump with a DC12V input and power consumption of 3.6W. It has a 3-meter head and can deliver a flow rate of 240 L/H.

Figure 11 Water Pump

2.6) 12V DC Fan Size 5×5 cm2

DC brushless fan that operates on 12V DC and draws a current of 170mA. It measures 5x5x1 cm^3 in size and comes with a long connecting wire.

Figure 12 (12V) DC Fan

2.7) 5V / 5W Heating Pad

This is a 5V DC heating element with an output heating power of 5W. It measures 50 x 100 mm in size and comes with a back adhesive layer, which makes it easy to install and attach to surfaces. It also has a cable length of 260 mm, which provides ample length for connecting it to a power source. This type of heating element is commonly used in various applications such as electronics, medical devices, and laboratory equipment, to provide consistent and controlled heating.

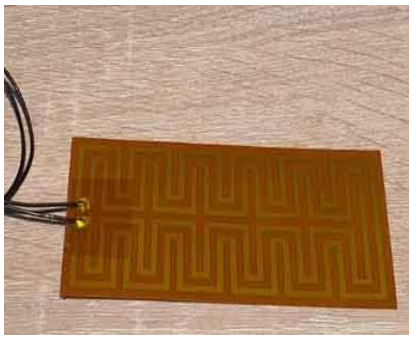


Figure 13 Heating Pad

Sensors

2.8) DHT11 Temperature and Humidity

The DHT11 is a basic, ultra low-cost digital temperature and humidity sensor. It uses a capacitive humidity sensor and a thermistor to measure the surrounding air, and spits out a digital signal on the data pin (no analog input pins needed). It's fairly simple to use, but requires careful timing to grab data.

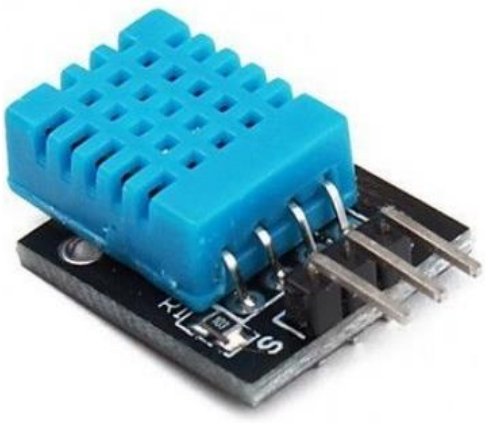


Figure 14 DHT11

The only real downside of this sensor is you can only get new data from it once every 2 Seconds, so when using our Library, sensor readings can be Up to 2 seconds.

Specification:

Low cost
3 to 5V power and I/O
2.5mA max current use during conversion (while requesting data)
Good for 20-80% humidity readings with 5% accuracy
Good for 0-50°C temperature readings ±2°C accuracy
No more than 1 Hz sampling rate (once every second)
Body size 15.5mm x 12mm x 5.5mm
3 pins with 0.1" spacing

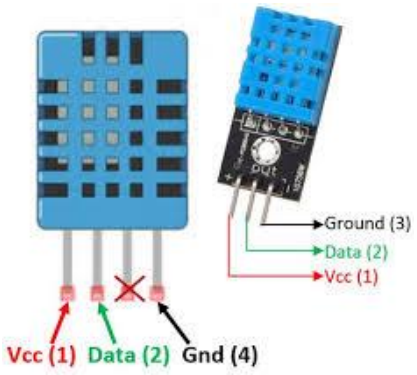


Figure 15 DHT11 pins

2.9) Soil Moisture KG003

A soil moisture sensor can read the amount of moisture present in the soil surrounding it. It is ideal for monitoring garden, or your plant's water level. This is a must have tool for a connected garden.

This sensor uses the two probes to pass current through the soil, and then it reads that resistance to get the moisture level. More water makes the soil conduct electricity more easily (less resistance), while dry soil conducts electricity poorly (more resistance).

It will be helpful to remind you to water your indoor plants or to monitor the soil moisture in your garden. The IO Expansion Shield is the perfect shield to connect this senor to nodemcu

The new soil moisture sensor uses Immersion Gold which protects the nickel from oxidation.

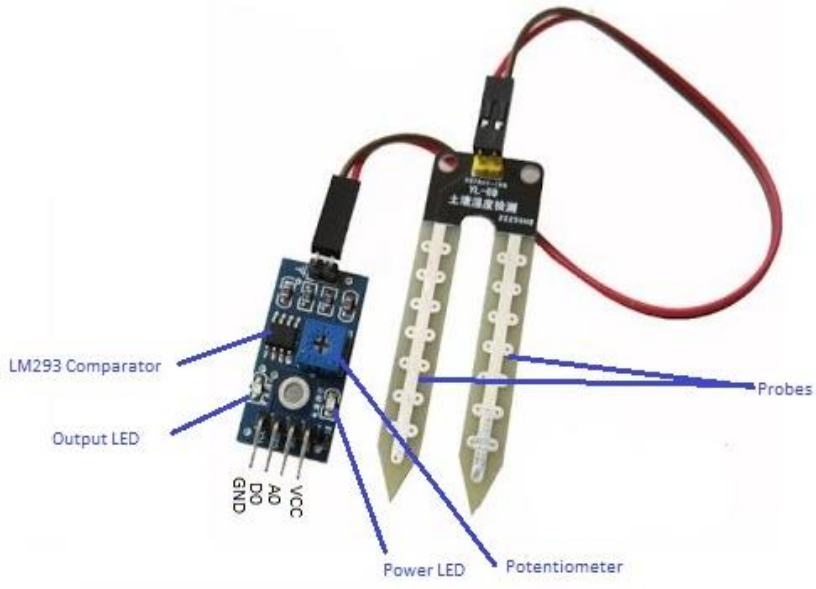


Figure 16 Soil Moist Sensor

Specification

- Working voltage: 5V
- Working Current: <20ma
- Interface: Analog/Digital
- Depth of detection: 37mm
- Working Temperature: 10℃~30℃
- Size: 63×20×8mm
- Low power consumption
- High sensitivity

2.10) Water Level Sensor & Liquid Water Droplet Depth Detection

This is simple and small portable water level/water droplet identification, detection sensor water that have high-cost performance. Complete water yield and analog conversion, the output value apply to your custom function. It is low power consumption and high sensitivity.
Water Sensor water level sensor is an easy-to-use, cost-effective high level/drop recognition sensor, which is obtained by having a series of parallel wires exposed traces measured droplets/water volume in order to determine the water level. Easy to complete water to analog signal conversion and output analog values can be directly read Arduino development board to achieve the level alarm effect.

Specifications

Working Voltage: DC 3-5V
Working Current: <20mA
Sensor Type: Simulation
Detection Area: 40 mm x 16 mm
Manufacturing Process: FR4 double spray tin
Fixed Hole Size: 3.2 mm
Humanized Design: Half-moon sag nonskid treatment
Working Temperature: 10 °C to 30 °C
Work Humidity: 10% to 90% without condensation
Size: 65 mm x 20 mm x 8 mm

Figure 17 Water Level Sensor

PCB Design

2.11) NodeMCU dimensions & schematic

Since in Protues there is no a library for NODEMCU so we had to build a design for

it is using the dimensions available for the NODEMCU as shown.

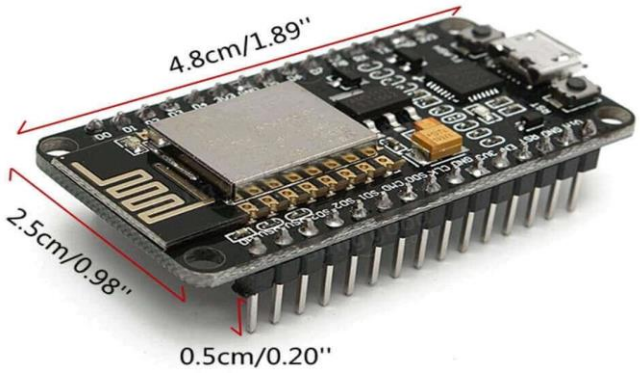


Figure 18 NODEMCU Dimensions

The outer dimensions are 50.8 mm * 25.4mm, we took it 28mm*52 for more tolerance and to guarantee not to face any space problems. After finishing the dimensions and testing their accuracy by printing & compare them practically to the real NodeMCU we had to build the schematic.

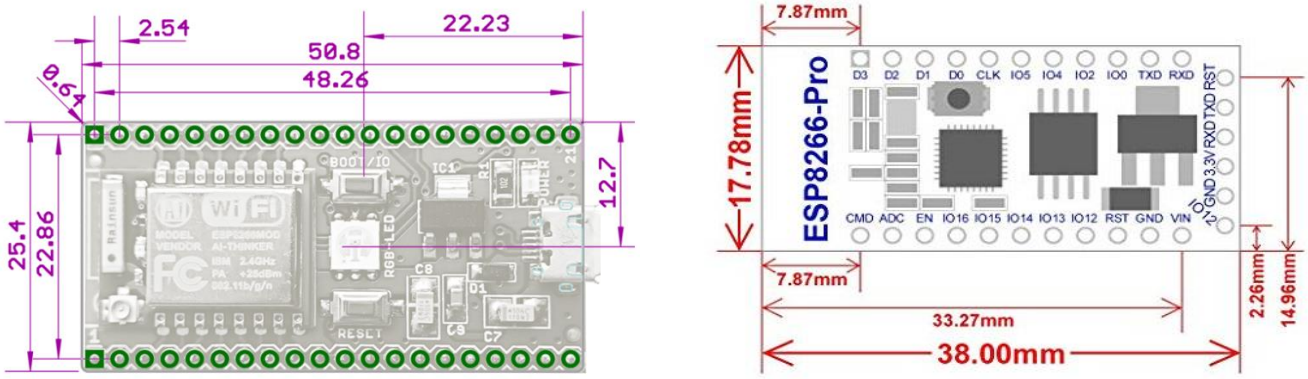


Figure 19 NODEMCU Dimensions

The schematic

It is important and should be with the accurate names and the numbers of the pins as we will use it to connect those pins in the Design of the main smart greenhouse PCB and consequently any fault at connecting this schematic may ruin the whole design.

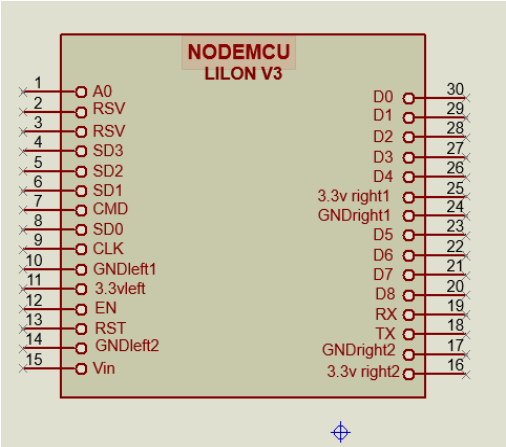


Figure 20 Node MCU Schematic

2.12) 3D Visualization for the PCB

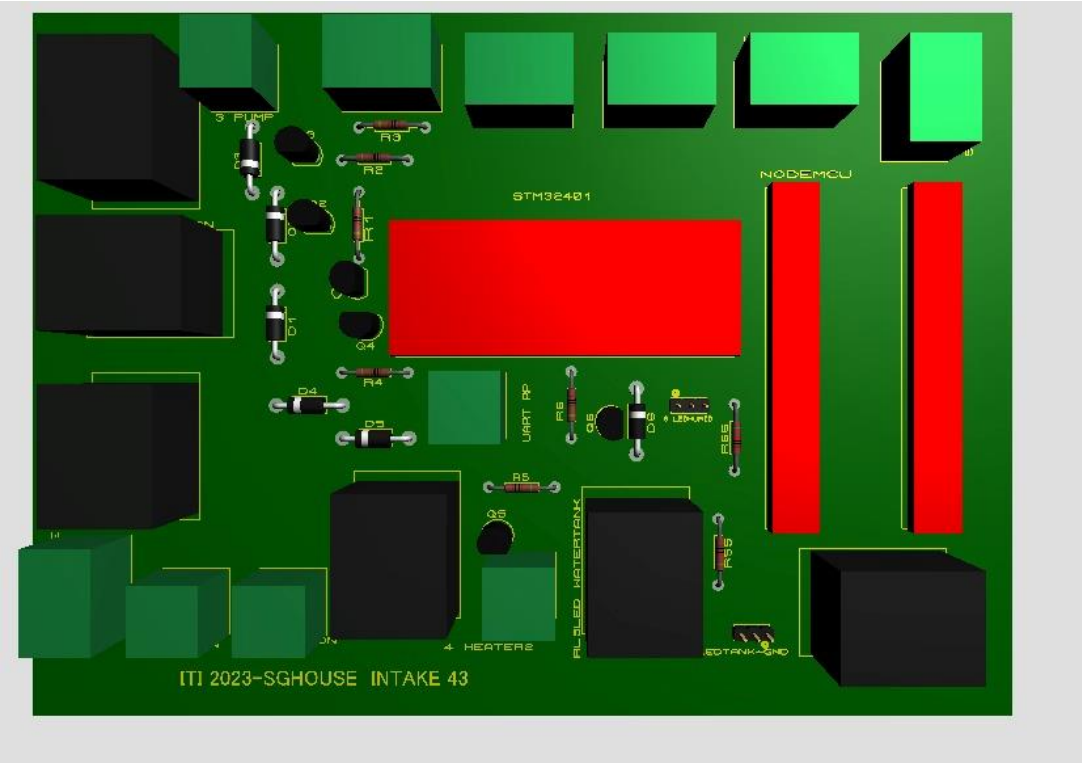


Figure 21 PCB 3D visualization

2.13) PCB Copper Layer:

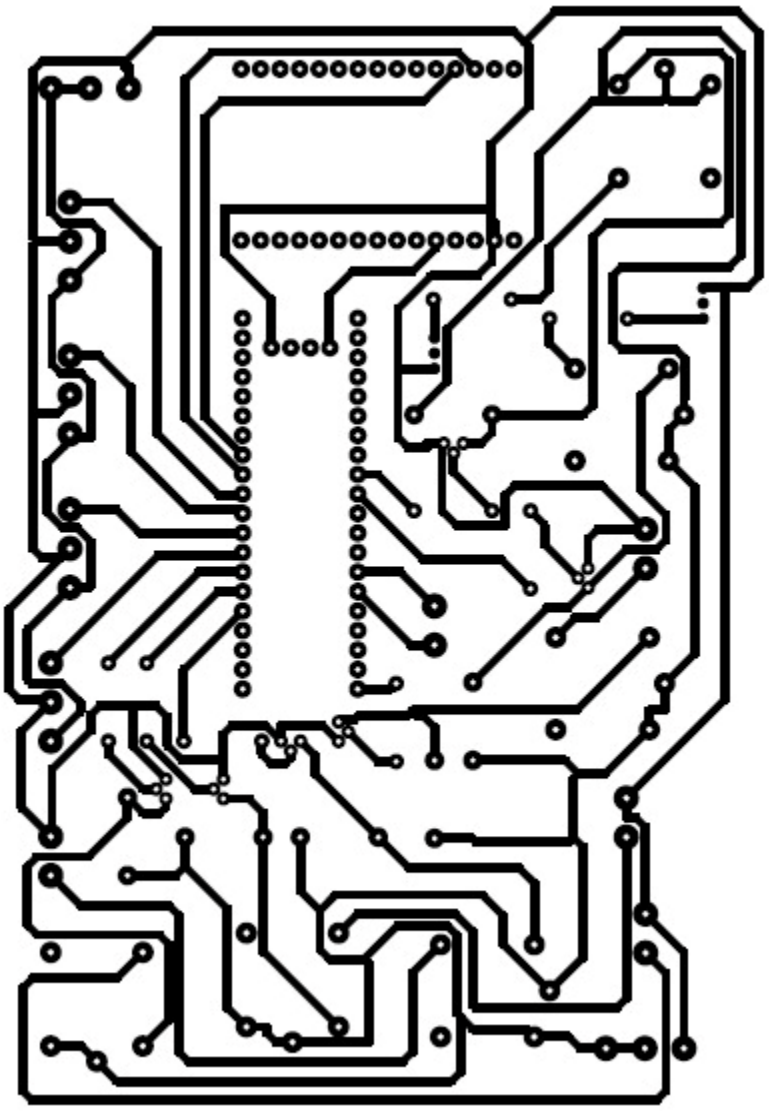


Figure 22 PCB copper layer

2.14) PCB Final Product:

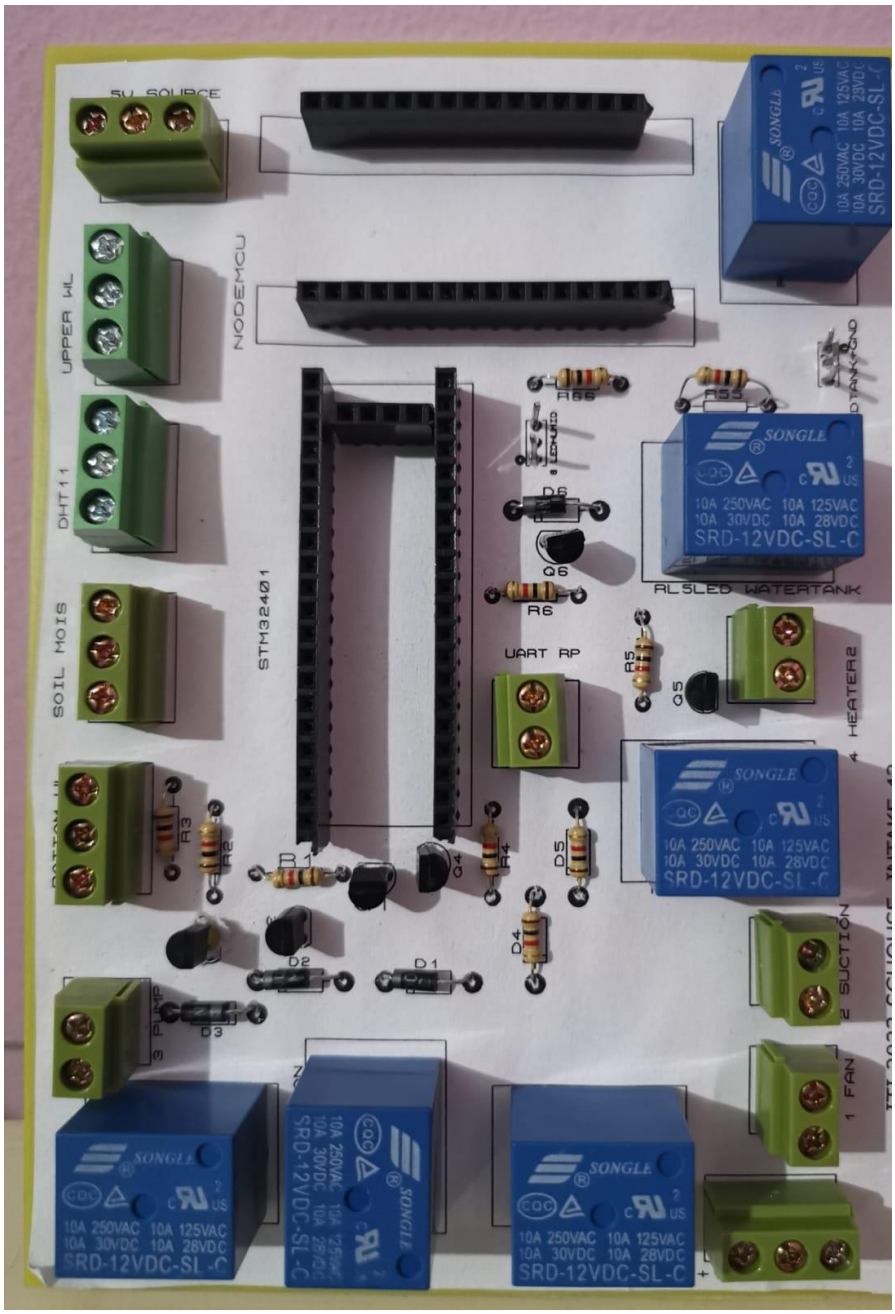


Figure 23 PCB final product

Chapter 3
Programming & Control
3.1) ESP8266 NodeMCU Pinout

The ESP8266 NodeMCU has total 30 pins that interface it to the outside world. The connections are as follows

Power Pins There are four “P” pins. one VIN pin & three 3.3V pins. The VIN pin can be used to directly supply the ESP8266 and its peripherals, if you have a regulated 5V voltage source. The 3.3V pins are the output of an on-board voltage regulator. These pins can be used to supply power to external components.

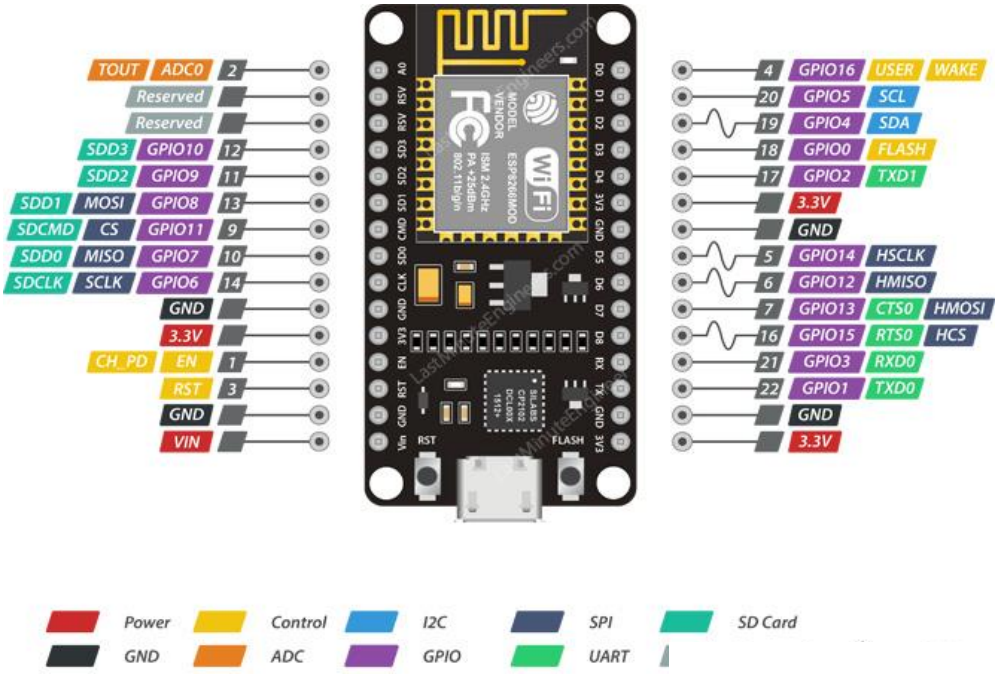


Figure 24 Node MCU pin OUT.

GND is a ground pin of ESP8266 NodeMCU development board.

GPIO Pins ESP8266 NodeMCU has 17 GPIO pins which can be assigned to various functions such as I2C, I2S, UART, PWM, IR Remote Control, LED Light and Button programmatically. Each digital enabled GPIO can be configured to internal pull-up or pull-down, or set to high impedance. When configured as an input, it can also be set to edge-trigger or level-trigger to generate CPU interrupts.

PWM Pins The board has 4 channels of Pulse Width Modulation (PWM). The PWM output can be implemented programmatically and used for driving digital motors and LEDs. PWM frequency range is adjustable from 1000 µs to 10000 µs, i.e., between 100 Hz and 1 kHz. Control Pins are used to control ESP8266. These pins include Chip Enable pin (EN), Reset pin (RST) and WAKE pin.

3.2) Blynk App

Blynk was designed for the Internet of Things. It can control hardware remotely, it can display sensor data, it can store data, visualize it and do many other cool things.

There are three major components in the platform:

Blynk App - allows to you create amazing interfaces for your projects using various widgets we provide.

Blynk Server - responsible for all the communications between the smartphone and hardware.

Blynk Libraries - for all the popular hardware platforms - enable communication with the server and process all the incoming and outcoming commands.

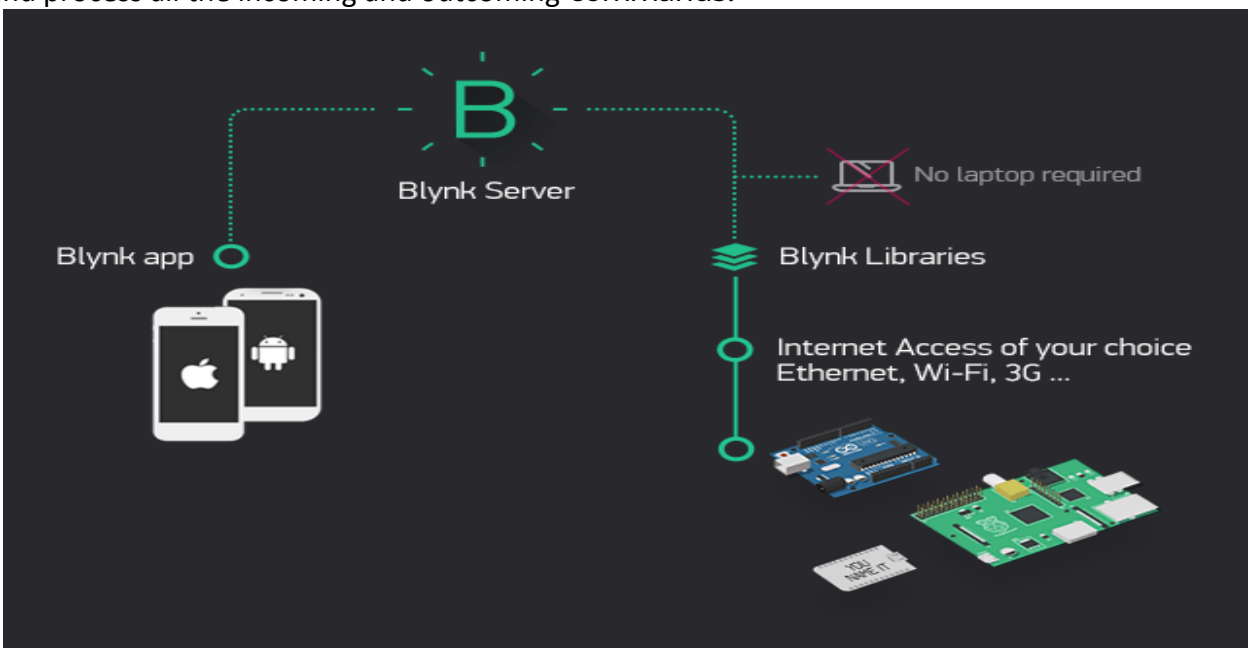


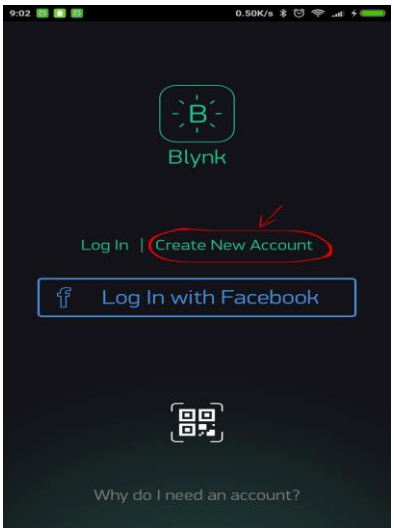
Figure 25 Blynk App Theory of Operation

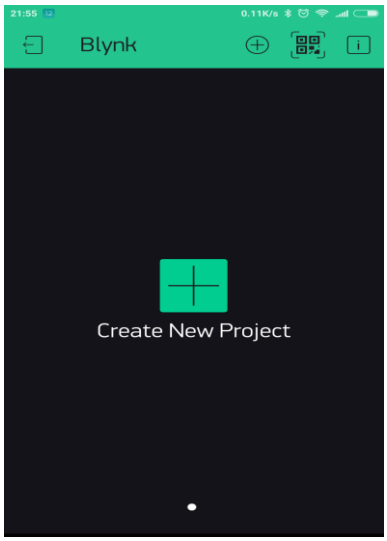
1. Create a Blynk Account

After you download the Blynk App, you'll need to create a New Blynk account. This account is separate from the accounts used for the Blynk Forums, in case you already have one.

2. Create a New Project

After you've successfully logged into your account, start by
creating a new project.



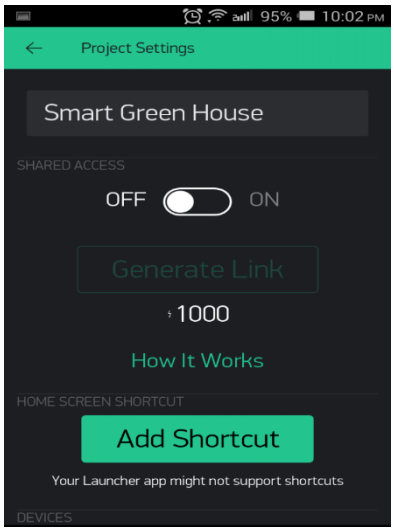


3. Choose Your Hardware

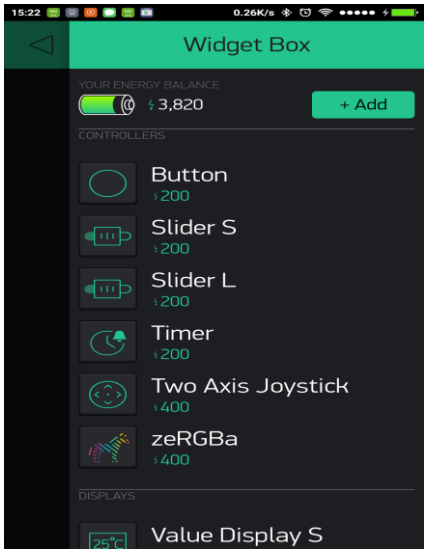
Select the hardware model you will use.

4. Auth Token

Auth Token is a unique identifier which is needed to connect your hardware to your smartphone. Every new project you create will have its own Auth Token. You'll get Auth Token automatically on your email after project creation. You can also copy it manually.

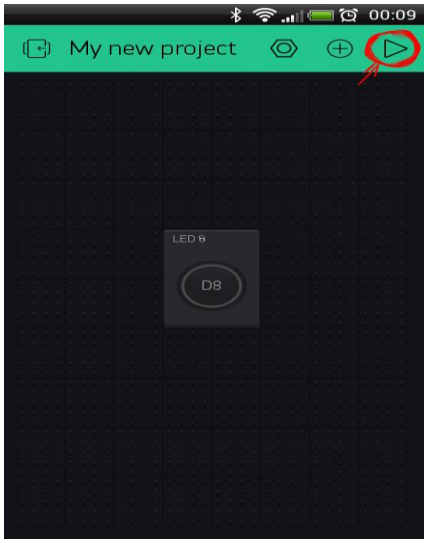


5. Add a Widget



Your project canvas is empty, let's add a button to control our LED. Tap anywhere on the canvas to open the widget box. All the available widgets are located here. Now pick a button.
Widget Box

6. Run The Project



When you are done with the Settings - press the PLAY button. This will switch you from EDIT mode to PLAY mode where you can interact with the hardware. While in PLAY mode, you won't be able to drag or set up new widgets, press STOP and get back to EDIT mode.
You will get a message saying "Arduino UNO is offline". We'll deal with that in the next section.

3.3) Programming Code

3.3.1) Flowchart

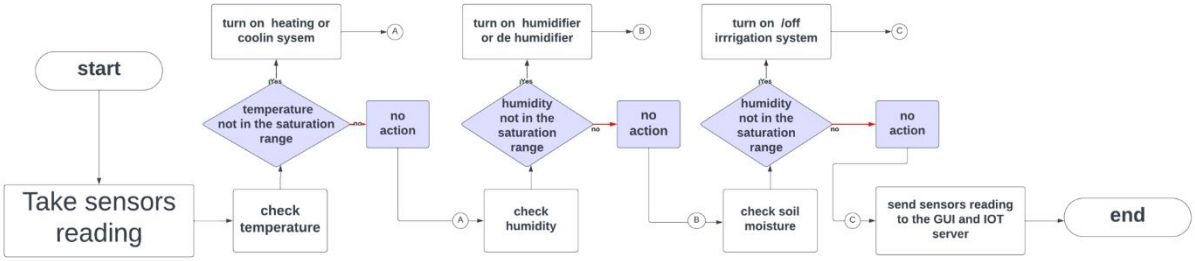


Figure 26 Code flowchart

3.3.2) STM32F401CC Code

```
/* Includes */
#include "main.h"

/**************************Including External Libraries**************************/
#include <stdio.h>
#include <string.h>
#include "FreeRTOSConfig.h"
#include "FreeRTOS.h"
#include "task.h"
#include "MY_DHT22.h"

/* Private variables ---------------------------------------------------------*/
ADC_HandleTypeDef hadc1;
UART_HandleTypeDef huart1;
UART_HandleTypeDef huart2;

/*********************************Task Handlers*********************************/
TaskHandle_t xDHT_Reading_Handler;
TaskHandle_t xSoilMoisture_Reading_Handler;
TaskHandle_t xWaterLevel_Reading_Handler;
TaskHandle_t xHR_Control_System;
TaskHandle_t xTemp_Control_System;
TaskHandle_t xIrrigation_Control_System;
TaskHandle_t xWaterTank_Control_System;

TaskHandle_t xUART_Transmit;
```

```
/* Private function prototypes*******************************/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_ADC1_Init(void);
static void MX_USART1_UART_Init(void);
static void MX_USART2_UART_Init(void);
/*****************************Tasks' Prototypes*******************************/
void vDHT11_Reading(void *pvParameters);
void vSoilMoisture_Reading(void *pvParameters);
void vWaterLevel_Reading(void *pvParameters);
void vHR_Control_System(void *pvParameters);
void vTemp_Control_System(void *pvParameters);
void vIrrigation_Control_System(void *pvParameters);
void vWaterTank_Control_System(void *pvParameters);
void vUART_Transmit(void *pvParameters);

/*****************************Global Variables*******************************/
/* Struct of Type DHT_DataTypedef To Receive the Reading of DHT11 */
DHT_DataTypedef DHT11_Data;

/* 2 Float Global Variables for Holding Temp and Humidity Readings from DHT11 */
uint8_t Temperature, Humidity;

/* Global u8 Variable to Read the ADC Output of the SoilMoisture Sensor */
float SoilMoisture;

/* Global u8 Variable to Hold the Mapped Value of Soil Moisture Sensor (0~100)% */
uint8_t SM_Percentage;

/* Global u8 Variable to Read the Analog Output of the Bottom Water Level Sensor */
uint8_t Water_Tank_Level;

/* Global Flag to Activate the Cooling System */
uint8_t Cooling_System_Flag     = 0;

 /* Global Flag to Activate the Heating System */
uint8_t Heating_System_Flag     = 0;

/* Global Flag to Activate the Irrigation System */
uint8_t Irrigation_System_Flag   = 0;

/* Global Flag to Activate the Humidifier System */
uint8_t Humidifier_System_Flag   = 0;
```

```
/* Global Flag to Activate the DeHumidifier System */
uint8_t DeHumidifier_System_Flag = 0;

/* Global Flag to Activate the Water Tank Filling System */
uint8_t WaterTank_System_Flag    = 0;

/* Global Array to send Data to uart */
char Global_u8Data[50] = {0};

/* Global Variable for Selecting ADC Channel */
ADC_ChannelConfTypeDef sConfig = {0};
int main(void)
{
  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* Configure the system clock */
  SystemClock_Config();

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_ADC1_Init();
  MX_USART1_UART_Init();
  MX_USART2_UART_Init();

  /******************************Creation of Tasks*******************************/
  xTaskCreate(vDHT11_Reading, "DHT11_READING", 64, NULL, 1, &xDHT_Reading_Handler);
  xTaskCreate(vSoilMoisture_Reading, "SoilMoisture_READING", 64, NULL, 1,
&xSoilMoisture_Reading_Handler);
  xTaskCreate(vWaterLevel_Reading, "WaterLevel_READING", 64, NULL, 1,
&xWaterLevel_Reading_Handler);

  xTaskCreate(vHR_Control_System, "HR_Contorl_System", 64, NULL, 1, &xHR_Control_System);
  xTaskCreate(vTemp_Control_System, "Temp_Contorl_System", 64, NULL, 1,
&xTemp_Control_System);
  xTaskCreate(vIrrigation_Control_System, "Irrigation_Contorl_System", 64, NULL, 1,
&xIrrigation_Control_System);
  xTaskCreate(vWaterTank_Control_System, "WaterTank_Control_System", 64, NULL, 1,
&xWaterTank_Control_System);

  xTaskCreate(vUART_Transmit, "UART_Transmit", 64, NULL, 1, &xUART_Transmit);

  /* Start Scheduler */
```

```
 vTaskStartScheduler();

 /* Infinite loop */
 while (1)
 {
 }
}



void SystemClock_Config(void)
{
 RCC_OscInitTypeDef RCC_OscInitStruct = {0};
 RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

 /* Configure the main internal regulator output voltage */
 __HAL_RCC_PWR_CLK_ENABLE();
 __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE2);

 /** Initializes the RCC Oscillators according to the specified parameters
 * in the RCC_OscInitTypeDef structure.
 */
 RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
 RCC_OscInitStruct.HSEState = RCC_HSE_ON;
 RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
 if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
 {
  Error_Handler();
 }

 /** Initializes the CPU, AHB and APB buses clocks
 */
 RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
 RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSE;
 RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
 RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
 RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

 if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
 {
  Error_Handler();
 }
}
```

```
static void MX_ADC1_Init(void)
{
  /* Configure the global features of the ADC (Clock, Resolution, Data Alignment and number of
conversion) */
  hadc1.Instance = ADC1;
  hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV2;
  hadc1.Init.Resolution = ADC_RESOLUTION_8B;
  hadc1.Init.ScanConvMode = DISABLE;
  hadc1.Init.ContinuousConvMode = DISABLE;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
  hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.NbrOfConversion = 1;
  hadc1.Init.DMAContinuousRequests = DISABLE;
  hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;

  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    Error_Handler();
  }

  /* Configure for the selected ADC regular channel its corresponding rank in the sequencer and its
sample time */
  sConfig.Channel = ADC_CHANNEL_4;
  sConfig.Rank = 1;
  sConfig.SamplingTime = ADC_SAMPLETIME_28CYCLES;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
}

static void MX_USART1_UART_Init(void)
{
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 9600;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;
  huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart1.Init.OverSampling = UART_OVERSAMPLING_16;
```

```
  if (HAL_UART_Init(&huart1) != HAL_OK)
  {
    Error_Handler();
  }
}

static void MX_USART2_UART_Init(void)
{
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 9600;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
}

static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_10|GPIO_PIN_12
                    |GPIO_PIN_5|GPIO_PIN_6, GPIO_PIN_RESET);

  /*Configure GPIO pin : PA5 */
  GPIO_InitStruct.Pin = GPIO_PIN_5;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
```

```
  /*Configure GPIO pins : PB0 PB1 PB10 PB12
                        PB5 PB6 */
  GPIO_InitStruct.Pin = GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_10|GPIO_PIN_12
                       |GPIO_PIN_5|GPIO_PIN_6;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;

  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
}

/*****************************Tasks' Implementation*****************************/
void vDHT11_Reading(void *pvParameters)
{
        while(1)
        {
          DHT_GetData(&DHT11_Data);
          Temperature = DHT11_Data.Temperature;
          Humidity = DHT11_Data.Humidity;
          vTaskDelay(15);
        }
}

void vSoilMoisture_Reading(void *pvParameters)
{
        while(1)
        {
   /*Switch To Channel 6*/
                sConfig.Channel = ADC_CHANNEL_6;

   /*Call Channel Configuration Function*/
                if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
             {
                Error_Handler();
             }

   /*Start Conversion sync and Passing the ADC Handler*/
                HAL_ADC_Start(&hadc1);

   /*Busy Wait till end of Conversion*/
                if(HAL_ADC_PollForConversion(&hadc1,20) == HAL_OK);
```

```
/*Read the ADC Value*/
                SoilMoisture = HAL_ADC_GetValue(&hadc1);

                vTaskDelay(16);
        }
}

void vWaterLevel_Reading(void *pvParameters)
{
        while(1)
        {
  /*Switch To Channel 4*/
                sConfig.Channel = ADC_CHANNEL_4;

  /*Call Channel Configuration Function*/
                if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
          {
                Error_Handler();
          }

  /*Start Conversion sync and Passing the ADC Handler*/
                HAL_ADC_Start(&hadc1);

  /*Busy Wait till end of Conversion*/
                if(HAL_ADC_PollForConversion(&hadc1,20) == HAL_OK);

  /*Read the ADC Value*/
                Water_Tank_Level = HAL_ADC_GetValue(&hadc1);

                vTaskDelay(17);
        }
}

void vHR_Control_System(void *pvParameters)
{
        while(1)
        {
                if((Humidifier_System_Flag == 0) && (DeHumidifier_System_Flag == 0))
                {
                        if(Humidity > 70)
                        {
```

```
                /* Activate DeHumidifier System */
                        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, 1);
                        DeHumidifier_System_Flag = 1;
                }
                else if (Humidity < 40)
                {
                        /* Activate Humidifier System */
                        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, 1);
                        Humidifier_System_Flag = 1;
                }
        }
        else if ((Humidity > 50) && (Humidity < 60))
        {
                if (Humidifier_System_Flag)
                {
                        /* Stop Humidifier System */
                        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, 0);
                        Humidifier_System_Flag = 0;
                }
                else if (DeHumidifier_System_Flag)
                {
                        /* Stop DeHumidifier System */
                        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, 0);
                        DeHumidifier_System_Flag = 0;
                }
        }

        vTaskDelay(18);
    }
}

void vTemp_Control_System(void *pvParameters)
{
    while(1)
    {
        if((Heating_System_Flag == 0) && (Cooling_System_Flag == 0))
        {
            if(Temperature > 40)
            {
                /* Activate Cooling System */
                HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, 1); //In take
                HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, 1); //Suction
```

```
                        Cooling_System_Flag = 1;
                }
                else if (Temperature < 30)
                {
                        /* Activate Heating System */
                        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, 1);

                        Heating_System_Flag = 1;
                }
        }
        else if ((Temperature > 34) || (Temperature < 36))
        {
                if (Heating_System_Flag)
                {
                        /* Stop Heating System */
                        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, 0);

                        Heating_System_Flag = 0;
                }
                else if (Cooling_System_Flag)
                {
                        /* Stop Cooling System */
                        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, 0); //In take
                        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, 0); //Suction

                        Cooling_System_Flag = 0;
                }
        }

        vTaskDelay(19);
    }
}
void vIrrigation_Control_System(void *pvParameters)
{
    while(1)
    {
        if (SoilMoisture > 200) {SoilMoisture=200;}
      if (SoilMoisture < 140) {SoilMoisture=140;}

        SM_Percentage = (((200 - SoilMoisture)/60)*100); //Remove Negative

        if (Irrigation_System_Flag == 0)
        {
            if(SM_Percentage < 40)
```

```
                {
                        /* Activate Irrigation System */
                        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, 1);

                        Irrigation_System_Flag = 1;
                }
        }
        else
        {
                if(SM_Percentage > 55)
                {
                        /* Deactivate Irrigation System */
                        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, 0);

                        Irrigation_System_Flag = 0;
                }
        }

        vTaskDelay(20);
    }
}

void vWaterTank_Control_System(void *pvParameters)
{
    while(1)
    {
        if (WaterTank_System_Flag == 0)
        {
                if(Water_Tank_Level < 50)
                {
                        /* Activate Water Tank Fill System */
                        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, 1);

                        WaterTank_System_Flag = 1;
                }
        }
        else
        {
                if(Water_Tank_Level > 200)
                {
                        /* Activate Water Tank Fill System */
                        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, 0);

                        WaterTank_System_Flag = 0;
```

```
                }
            }

            vTaskDelay(21);
        }
}

void vUART_Transmit(void *pvParameters)
{
        while(1)
        {
            sprintf(Global_u8Data, "%d\r\n", Humidity);
            HAL_UART_Transmit(&huart1,(char *) Global_u8Data, strlen(Global_u8Data),3);
            sprintf(Global_u8Data, "%d\r\n", Humidifier_System_Flag);
            HAL_UART_Transmit(&huart1,(char *) Global_u8Data, strlen(Global_u8Data),3);
            sprintf(Global_u8Data, "%d\r\n", DeHumidifier_System_Flag);
            HAL_UART_Transmit(&huart1,(char *) Global_u8Data, strlen(Global_u8Data),3);
            sprintf(Global_u8Data, "%d\r\n", Temperature);
            HAL_UART_Transmit(&huart1,(char *) Global_u8Data, strlen(Global_u8Data),3);
            sprintf(Global_u8Data, "%d\r\n", Heating_System_Flag);
            HAL_UART_Transmit(&huart1,(char *) Global_u8Data, strlen(Global_u8Data),3);
            sprintf(Global_u8Data, "%d\r\n", Cooling_System_Flag);
            HAL_UART_Transmit(&huart1,(char *) Global_u8Data, strlen(Global_u8Data),3);
            sprintf(Global_u8Data, "%d\r\n", SM_Percentage);
            HAL_UART_Transmit(&huart1,(char *) Global_u8Data, strlen(Global_u8Data),3);

            sprintf(Global_u8Data, "%d\r\n", Irrigation_System_Flag);
            HAL_UART_Transmit(&huart1,(char *) Global_u8Data, strlen(Global_u8Data),3);
            sprintf(Global_u8Data, "%d\r\n", WaterTank_System_Flag);
            HAL_UART_Transmit(&huart1,(char *) Global_u8Data, strlen(Global_u8Data),3);
            sprintf(Global_u8Data, "%d\r", Humidity);
            HAL_UART_Transmit(&huart2,(char *) Global_u8Data, strlen(Global_u8Data),3);
            sprintf(Global_u8Data, "%d\r", Humidifier_System_Flag);
            HAL_UART_Transmit(&huart2,(char *) Global_u8Data, strlen(Global_u8Data),3);
            sprintf(Global_u8Data, "%d\r", DeHumidifier_System_Flag);
            HAL_UART_Transmit(&huart2,(char *) Global_u8Data, strlen(Global_u8Data),3);
            sprintf(Global_u8Data, "%d\r", Temperature);
            HAL_UART_Transmit(&huart2,(char *) Global_u8Data, strlen(Global_u8Data),3);
            sprintf(Global_u8Data, "%d\r", Heating_System_Flag);
            HAL_UART_Transmit(&huart2,(char *) Global_u8Data, strlen(Global_u8Data),3);
            sprintf(Global_u8Data, "%d\r", Cooling_System_Flag);
            HAL_UART_Transmit(&huart2,(char *) Global_u8Data, strlen(Global_u8Data),3);
            sprintf(Global_u8Data, "%d\r", SM_Percentage);
            HAL_UART_Transmit(&huart2,(char *) Global_u8Data, strlen(Global_u8Data),3);
```

```
                sprintf(Global_u8Data, "%d\r", Irrigation_System_Flag);
                HAL_UART_Transmit(&huart2,(char *) Global_u8Data, strlen(Global_u8Data),3);
                sprintf(Global_u8Data, "%d\r", WaterTank_System_Flag);
                HAL_UART_Transmit(&huart2,(char *) Global_u8Data, strlen(Global_u8Data),3);
                vTaskDelay(22);
        }
}

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
 if (htim->Instance == TIM1) {
   HAL_IncTick();
 }
}
void Error_Handler(void)
{
 __disable_irq();
 while (1)
 {
 }
}

#ifdef  USE_FULL_ASSERT
void assert_failed(uint8_t *file, uint32_t line)
{
}
#endif /* USE_FULL_ASSERT */
```

3.3.3) IOT Code

```
#define BLYNK_PRINT Serial

#include <ESP8266WiFi.h>
#include <BlynkSimpleEsp8266.h>
#include <SoftwareSerial.h>
#include <Simpletimer.h>

char auth[] = "IBtUnFMyu39jhDvZU1sgRYcjgSijIPSe";

//Set Wifi and password
char ssid[] = "Honor_8C";
char pass[] = "sa3d1234";

SimpleTimer timer;

int Humiditiy;
int Humidifier;
int DeHumidifier;
int Temp;
int Heating;
int Cooling;
int SM_Pre;
int Irrigation;
int WaterPump;

void myTimerEvent()
{
  Blynk.virtualWrite(V1,millis()/1000);
}

void setup() {
```

```
  Serial.begin(9600);
  Blynk.begin(auth,ssid,pass);
  /* Sensor  */
  timer.setInterval(1000L,HumiditiyValue);
  timer.setInterval(1000L,HumidifierVlaue);
  timer.setInterval(1000L,DeHumidifierVlaue);
  timer.setInterval(1000L,TempVlaue);
  timer.setInterval(1000L,HeatingVlaue);
  timer.setInterval(1000L,CoolingVlaue);
  timer.setInterval(1000L,SM_PerVlaue);
  timer.setInterval(1000L,IrrigationVlaue);
  timer.setInterval(1000L,WaterPumpVlaue);
}

void loop() {

  if (Serial.available() == 0)
  {
    Blynk.run();
    timer.run();  //Initiates BlynkTimer
  }
  if(Serial.available()>0){
    String SHumiditiy=Serial.readStringUntil('\r');
     Humiditiy = SHumiditiy.toInt();
    String SHumidifier=Serial.readStringUntil('\r');
     Humidifier = SHumidifier.toInt();
    String SDeHumidifier=Serial.readStringUntil('\r');
     DeHumidifier = SDeHumidifier.toInt();
    String STemp=Serial.readStringUntil('\r');
     Temp = STemp.toInt();
    String SHeating=Serial.readStringUntil('\r');
     Heating = SHeating.toInt();
```

```
    String SCooling=Serial.readStringUntil('\r');
     Cooling = SCooling.toInt();
    String SSM_Pre=Serial.readStringUntil('\r');
     SM_Pre = SSM_Pre.toInt();
    String SIrrigation=Serial.readStringUntil('\r');
     Irrigation = SIrrigation.toInt();
    String SWaterPump=Serial.readStringUntil('\r');
     WaterPump = SWaterPump.toInt();
  }
}

void HumiditiyValue()
{
  Blynk.virtualWrite(V6,Humiditiy);
}

void HumidifierVlaue()
{
  Blynk.virtualWrite(V9,Humidifier);
}

void DeHumidifierVlaue()
{
  Blynk.virtualWrite(V8,DeHumidifier);
}

void TempVlaue()
{
  Blynk.virtualWrite(V0,Temp);
}

void HeatingVlaue()
```

```
{
  Blynk.virtualWrite(V3,Heating);
}

void CoolingVlaue()
{
  Blynk.virtualWrite(V7,Cooling);
}

void SM_PerVlaue()
{
  Blynk.virtualWrite(V2,SM_Pre);
}

void IrrigationVlaue()
{
  Blynk.virtualWrite(V5,Irrigation);
}

void WaterPumpVlaue()
{
  Blynk.virtualWrite(V4,WaterPump);
}
```

3.3.4) GUI Code

```
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.uic import loadUi
import Resource_file_images_rc
import sys
import serial
import time

G_H_Value = 10
G_T_Value = 20
G_Z_Value = 10
GF_Cooling = 1
GF_Heating = 1
GF_WaterTank = 1
GF_Humidifier = 1
GF_DeHumidifier = 1
GF_Irrigation = 1

#Initialization
app = QtWidgets.QApplication(sys.argv)
ser = serial.Serial('/dev/ttyUSB0',9600,timeout=5)
ser.flush()

class UI(QtWidgets.QMainWindow):
    def __init__(self):
        super().__init__()
        loadUi("GP_GUI.ui", self)
        self.update_StyleSheet()
        self.show()
        # Start a timer to periodically read the value from UART
        self.timer = QtCore.QTimer()
        self.timer.timeout.connect(self.read_uart)
```

```
        self.timer.start(1000)  # Read every 1000 milliseconds = 1 Second
    def update_StyleSheet(self):
        global G_H_Value
        global G_T_Value
        global G_Z_Value
        global GF_Cooling
        global GF_Heating
        global GF_WaterTank
        global GF_Humidifier
        global GF_DeHumidifier
        global GF_Irrigation

        #Strings that have the QFrame StyleSheet
        self.stylesheet_HRing = """
        QFrame
        {
        border-radius: 85px;
        background-color: qconicalgradient(cx:0.5, cy:0.5, angle:90, stop:{H_STOP_1}
rgba(0, 0, 0, 0), stop:{H_STOP_2} rgba(27, 130, 0, 255));
        }
        """
        self.stylesheet_TRing = """
        QFrame
        {
        border-radius: 85px;
        background-color: qconicalgradient(cx:0.5, cy:0.5, angle:90, stop:{T_STOP_1}
rgba(0, 0, 0, 0), stop:{T_STOP_2} rgba(255, 94, 0, 255));
        }
        """
        self.stylesheet_ZRing = """
        QFrame
        {
```

```
        border-radius: 85px;
        background-color: qconicalgradient(cx:0.5, cy:0.5, angle:90, stop:{Z_STOP_1}
rgba(0, 0, 0, 0), stop:{Z_STOP_2} rgba(13, 0, 255, 255));
        }
        """
        self.stylesheet_Activated="""
        QFrame{
        border-radius:15px;
        background-color: rgb(0, 140, 0);
        }
        """
        self.stylesheet_Not_Activated="""
        QFrame{
        border-radius:15px;
        background-color: rgb(170, 0, 0);
        }
        """
        #Changing Values to (0~1)string Form
        H_Value_pr1 = (100 - G_H_Value) / 100.00
        H_Value_pr2 = str(H_Value_pr1 - 0.001)
        T_Value_pr1 = (100 - G_T_Value) / 100.00
        T_Value_pr2 = str(T_Value_pr1 - 0.001)
        Z_Value_pr1 = (100 - G_Z_Value) / 100.00
        Z_Value_pr2 = str(Z_Value_pr1 - 0.001)

        #Replace the Original QFrame StyleSheet stop Values with the new Values
        self.New_stylesheet_HRing = self.stylesheet_HRing.replace("{H_STOP_1}",
H_Value_pr2).replace("{H_STOP_2}", str(H_Value_pr1))
        self.New_stylesheet_TRing = self.stylesheet_TRing.replace("{T_STOP_1}",
T_Value_pr2).replace("{T_STOP_2}", str(T_Value_pr1))
        self.New_stylesheet_ZRing = self.stylesheet_ZRing.replace("{Z_STOP_1}",
Z_Value_pr2).replace("{Z_STOP_2}", str(Z_Value_pr1))
```

```
#Update the New StyleSheet
self.H_Ring.setStyleSheet(self.New_stylesheet_HRing)
self.T_Ring.setStyleSheet(self.New_stylesheet_TRing)
self.Z_Ring.setStyleSheet(self.New_stylesheet_ZRing)

#Updating Indications
if GF_Cooling:
    self.Cooling_Indicator.setStyleSheet(self.stylesheet_Activated)
else:
    self.Cooling_Indicator.setStyleSheet(self.stylesheet_Not_Activated)
if GF_Heating:
    self.Heating_Indicator.setStyleSheet(self.stylesheet_Activated)
else:
    self.Heating_Indicator.setStyleSheet(self.stylesheet_Not_Activated)
if GF_WaterTank:
    self.TankPump_Indicator.setStyleSheet(self.stylesheet_Activated)
else:
    self.TankPump_Indicator.setStyleSheet(self.stylesheet_Not_Activated)
if GF_Humidifier:
    self.Humidifier_Indicator.setStyleSheet(self.stylesheet_Activated)
else:
    self.Humidifier_Indicator.setStyleSheet(self.stylesheet_Not_Activated)
if GF_DeHumidifier:
    self.DeHumidifier_Indicator.setStyleSheet(self.stylesheet_Activated)
else:
    self.DeHumidifier_Indicator.setStyleSheet(self.stylesheet_Not_Activated)
if GF_Irrigation:
    self.Irrigation_Inidicator.setStyleSheet(self.stylesheet_Activated)
else:
    self.Irrigation_Inidicator.setStyleSheet(self.stylesheet_Not_Activated)
```

```
        #Update the Labels with the New Values
        self.H_Labe_2.setText(f"{int(G_H_Value)}{'%'}")
        self.T_Labe_2.setText(f"{int(G_T_Value)}{'°'}")
        self.Z_Labe_2.setText(f"{int(G_Z_Value)}{'%'}")

    def read_uart(self):
        global G_H_Value
        global G_T_Value
        global G_Z_Value
        global GF_Cooling
        global GF_Heating
        global GF_WaterTank
        global GF_Humidifier
        global GF_DeHumidifier
        global GF_Irrigation

        if ser.in_waiting:
            # Read the value from UART and update Global Values
            G_H_Value = int(ser.readline().decode('utf-8').strip())
            GF_Humidifier = int(ser.readline().decode('utf-8').strip())
            GF_DeHumidifier = int(ser.readline().decode('utf-8').strip())
            G_T_Value = int(ser.readline().decode('utf-8').strip())
            GF_Heating = int(ser.readline().decode('utf-8').strip())
            GF_Cooling = int(ser.readline().decode('utf-8').strip())
            G_Z_Value = int(ser.readline().decode('utf-8').strip())
            GF_Irrigation = int(ser.readline().decode('utf-8').strip())
            GF_WaterTank = int(ser.readline().decode('utf-8').strip())
            self.update_StyleSheet()
ui=UI()
app.exec_()
```

3.4) Main Code Explanation

The STM32F401CC uses Real time operating system to get the readings of Temperature, Humidity, Soil moisture and Water level sensors. It takes an action controlling Fans, heater and Water pump depending on the readings values. These values are sent from the microcontroller to an IOT server that can be accessed by a mobile application, also the readings values are sent to a Raspberry pi 3 B+ which operates a GUI screen to show the current readings of the sensors.

Chapter 4

4.1) Project's future modifications

- Add manual mode for the user to control fans, water pump and heaters.
- Add mode for the user to change the critical readings values (BOOTLOADER).
- Make these modes available to use through GUI screen and IOT application.

References

https://blynk.io/

https://www.nodemcu.com/index_en.html

https://www.adafruit.com/product/386

https://www.researchgate.net/publication/316448621_IoT_based_smart_greenhouse

http://www.fao.org/3/S8684E/s8684e06.htm

https://en.wikipedia.org/wiki/Greenhouse

https://www.usaid.gov/egypt/agriculture-and-food-security