Smart Hub for Air Conditioning Network

Group Members

Muhammad Jayyed Sabeeh 18I-0813

Haseeb Ahmed 18I-0907

Ahmad Arif 18I-0903

Project Supervisor

Engr. Shehzad Ahmad

Department of Electrical Engineering

National University of Computer and Emerging Sciences, Islamabad

Developers' Submission

"This report is being submitted to the Department of Electrical Engineering of the National University of Computer and Emerging Sciences in partial fulfillment of the requirements for the degree of BS in Electrical Engineering"

Developers' Declaration

"We take full responsibility of the project work conducted during the Final Year Project (FYP) titled "Smart Hub for Air Conditioning Network". We solemnly declare that the project work presented in the FYP report is done solely by us with no significant help from any other person; however, small help wherever taken is duly acknowledged. We have also written the complete FYP report by ourselves. Moreover, we have not presented this FYP (or substantially similar project work) or any part of the thesis previously to any other degree awarding institution within Pakistan or abroad.

We understand that the management of Department of Electrical Engineering of National University of Computer and Emerging Sciences has a zero-tolerance policy towards plagiarism. Therefore, we as an author of the above-mentioned FYP report solemnly declare that no portion of our report has been plagiarized and any material used in the report from other sources is properly referenced. Moreover, the report does not contain any literal citing of more than 70 words (total) even by giving a reference unless we have obtained the written permission of the publisher to do so. Furthermore, the work presented in the report is our own work and we have positively cited the related work of the other projects by clearly differentiating our work from their relevant work.

We further understand that if we are found guilty of any form of plagiarism in our FYP report even after our graduation, the University reserves the right to withdraw our BS degree. Moreover, the University will also have the right to publish our names on its website that keeps a record of the students who committed plagiarism in their FYP reports."

Muhammad Jayyed Sabeeh BS(EE)2018-0813	Haseeb Ahmed BS(EE)2018-0907	Ahmad Arif BS(EE)2018-0903

Certified by Supervisor

Verified by Plagiarism Cell Officer
Dated: _____

Abstract

Most of the ACs are not smart, as in they cannot be controlled remotely or have Wi-Fi enabled access which poses a problem since it is the need of time and the newer models that come equipped with Wi-Fi access are quite costly. Individually controlling these air conditioners also poses a problem in an upscale environment like an office building, university, etc. Our product tackles this issue and provides a cost effective and feasible solution.

Acknowledgements

The final year project “Smart Hub for Air Conditioning Network” was successfully completed in the Final Year Project Lab of National University of Computer and Emerging Sciences (FAST-NUCES), Islamabad Campus under the Pakistan Engineering Council (PEC) Annual Award for Final Year Design Projects (FYDPs) for the year 2022-2023. The project was supervised by Engr. Shehzad Ahmed.

We extend our most profound gratitude to Engr. Shehzad Ahmed, our supervisor, for his critical guidance, scrutiny and suggestions during ourproject. We also extend our profound thanks to Engr. Aamer Munir and Dr. Arshad Hassan for their guidance, critique, and suggestions that helped us steer in a better direction with our project. We would also like to express gratitude to Dr. Farhan Khalid for his guidance, and Engr. Sana Saleh and Engr. Faryal Gula for their comments and queries that helped us get our work done in a better, more efficient manner.

Table of Contents

Chapter 1

Overview

In this chapter, we give an introduction to our project, the motivation and problem statement that we are addressing and the literature review that helped us in our project.

Introduction

Managing different remotes and climate options for different rooms within a home/apartment or a public space like an office or university, having air conditioners of different make, model and variants is a hassle, and an inefficient way to control the climatic options of a home/apartment. An individual would have to physically visit all rooms to ensure if the Acs are turned on or not, and it also leads to greater energy tariffs as ACs may be operating in rooms where they are not required for an instance. A remote, central control can tackle this issue very easily by providing remote access to the controller thereby ensuring efficiency of use and power tariff reduction.

Motivation

Since the advent of automation, the global idea has been to make everything from appliances to security to household, smart. Smart in its functionality, its energy consumption and the required output from the system. With this concept in mind, buildings that have been constructed recently can have smart, automation systems already installed like a centrally controlled, smart HVAC system that provides the user with the required output in an efficient manner. But problems arise because majority of the construction that has been already done does not cater to the smart home requirements, so to solve one aspect of the smart home problem our proposed project is a “Smart Hub for Air Conditioning Network”. By retrofitting our system on the currently used split ACs, the user would be able to make their pre-installed system smart and efficient, along with automated controls without the capital consuming process of having to replace their whole air conditioning system.

Problem Statement

Managing multiple remotes and climate options for various rooms in a building is inconvenient and inefficient. It requires physically checking each room and can result in unnecessary energy usage. A centralized and automated system for controlling air conditioners would streamline the process and save energy costs. Our project has retrofit existing split air conditioning systems with smart technology, allowing users to control and monitor their AC systems remotely. It includes sensors to monitor temperature, humidity, and other factors to optimize the performance of the AC system and minimize energy consumption. A user-friendly mobile app is developed to allow users to control their retrofitted smart AC systems, remotely

Literature Review

A relevant starting point for us was the research on new universal adapter that enables simultaneous control of air conditioners using both a phone app and an infrared remote control. The adapter recognizes commands from the air conditioner's remote control, allowing the phone app to be updated accordingly. This innovation aimed to enhance convenience and flexibility in air conditioner control. [1]

Another useful research was the development of an advanced universal remote controller (URC) that aimed to provide a comprehensive solution for home automation and security. The URC would allow control over various home appliances and can be connected to a PC via the internet. Multiple receivers with wired or wireless communication methods were utilized to connect to the appliances, offering extensive control options. The receivers supported numerous channels and IDs to facilitate simultaneous control of multiple appliances and enabled multi-zone services. Furthermore, a user-friendly PC-based interface enhanced the convenience of operating the URC for end-users. [2]

Another research paper focused on controlling IR-based devices through Wi-Fi to address the inconvenience of managing multiple remotes for various devices such as air conditioners, TVs, set-top boxes, home theaters and DVD players. The paper proposed a smart IR device capable of controlling multiple applications. By utilizing Wi-Fi connectivity, this device eliminates the need for individual remotes and enables centralized control over various IR-based devices. [3]

Chapter 2

Overview

In this chapter, we discuss our design, its implementation, the steps we took to achieve our goal, along with the discussion of our hardware and its specifications.

Solution Design and Implementation

Solution Design:
We firstly analyzed the IR signal that is transmitted via an AC remote to get a better understanding of our work, as the IR signal sent by an AC remote is different than any other IR remote. A regular IR remote simply transmits the data for the respective button that is pressed, but an AC remote creates a packet of data containing information like the mode of the AC, temperature, swing state, fan speed, etc.

To do so, we used the below mentioned circuits to receive the data from the module and encode it to figure out the IR transmission protocol that is being used by the AC. We then transmitted the same data using our above circuit and received it to ensure that it is properly encoded after it has been decoded once. The circuit above uses a TSOP 1738 IR receiver that operates at a frequency of 38 KHz, and demodulates the signal using 38 KHz as the carrier signal frequency.

Circuits for IR Signal Decoding and Analysis:

- **IR Transmitter Circuit**

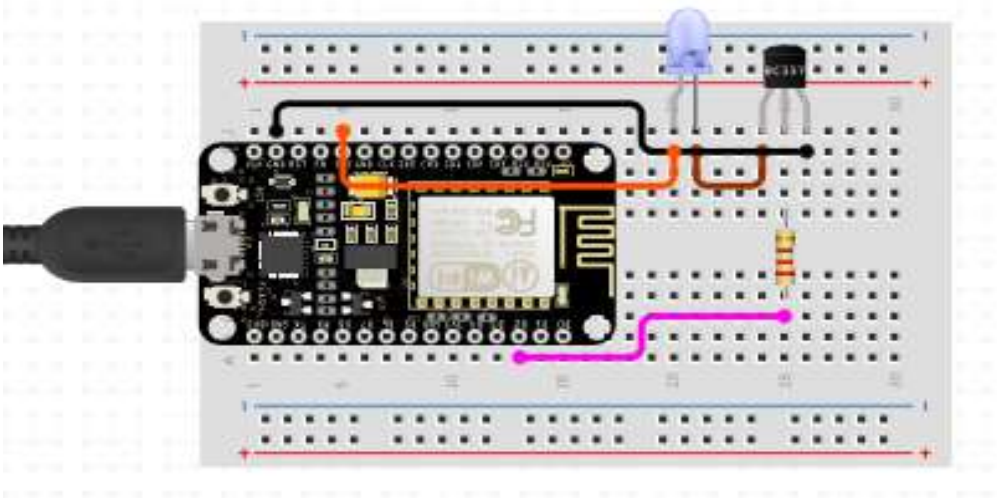


Fig.1

- **IR Receiver Circuit**

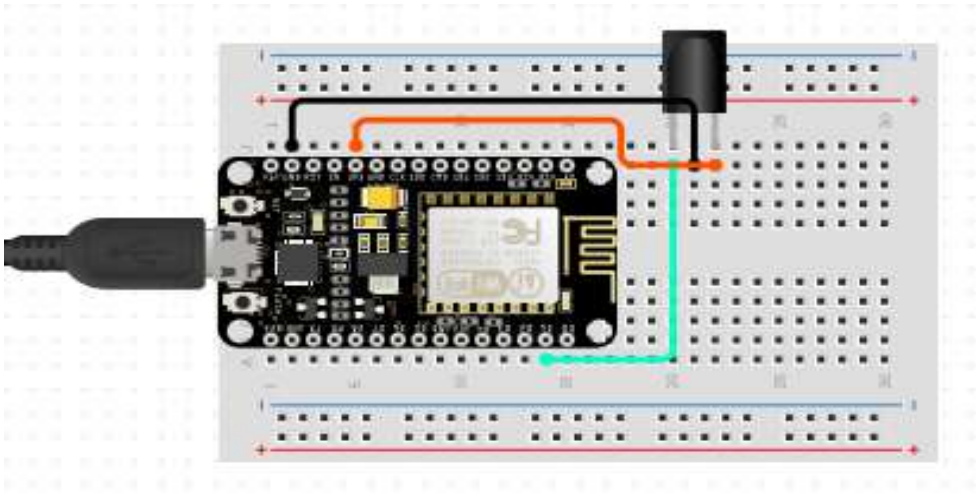


Fig. 2

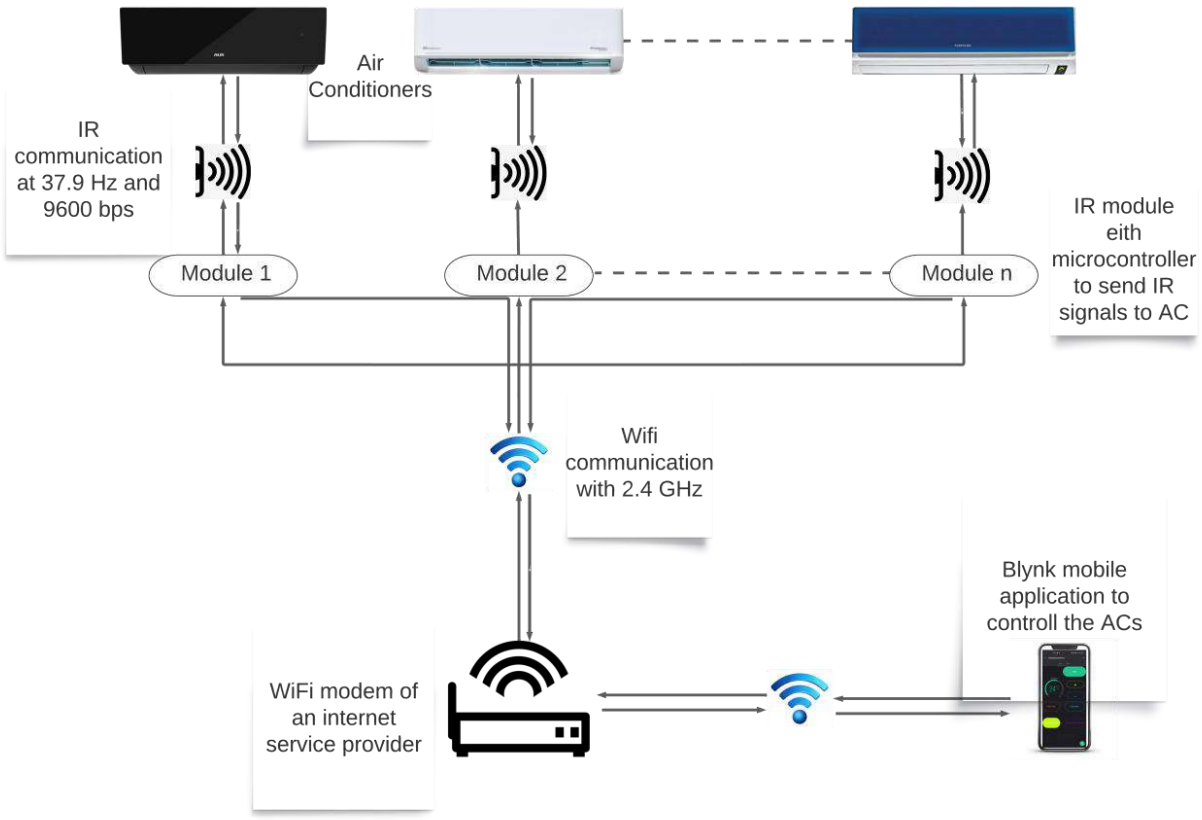
We connected our circuit as shown in figure 1 and 2, and connected output to an oscilloscope to analyze the waveform and thereby decode the signal. The results of the analysis are shown below.

From the above results, we were able to deduce that the remote that we decoded, (Orient AC) uses NEC protocol to decode data. The NEC protocol uses 16-bits to represent data. The first 8 bits are address bits which remain unchanged while the next 8 bits are data bits which change according to the operation being performed. These bits are shown as pulses of varying widths

and various time intervals according to the IR protocol being used. These results are shown below.

- **Bits Changing denoting Temperature Change.**
  - AC ON 25° 1100 0011 1111 0001
  - 25° => 26° 1100 0011 1110 1001
  - 26° => 27° 1100 0011 1111 1001
  - 27° => 28° 1100 0011 1110 0101
- **Bits Changing denoting Mode Change.**
  - AC ON 25° 1100 0011 1111 0001
  - FAN 1100 0011 1110 0000
  - AUTO 1100 0011 1111 1001
  - DRY 1100 0011 1110 0010
  - COOL 1100 0011 1110 0001

Block Diagram:



Block Diagram Description

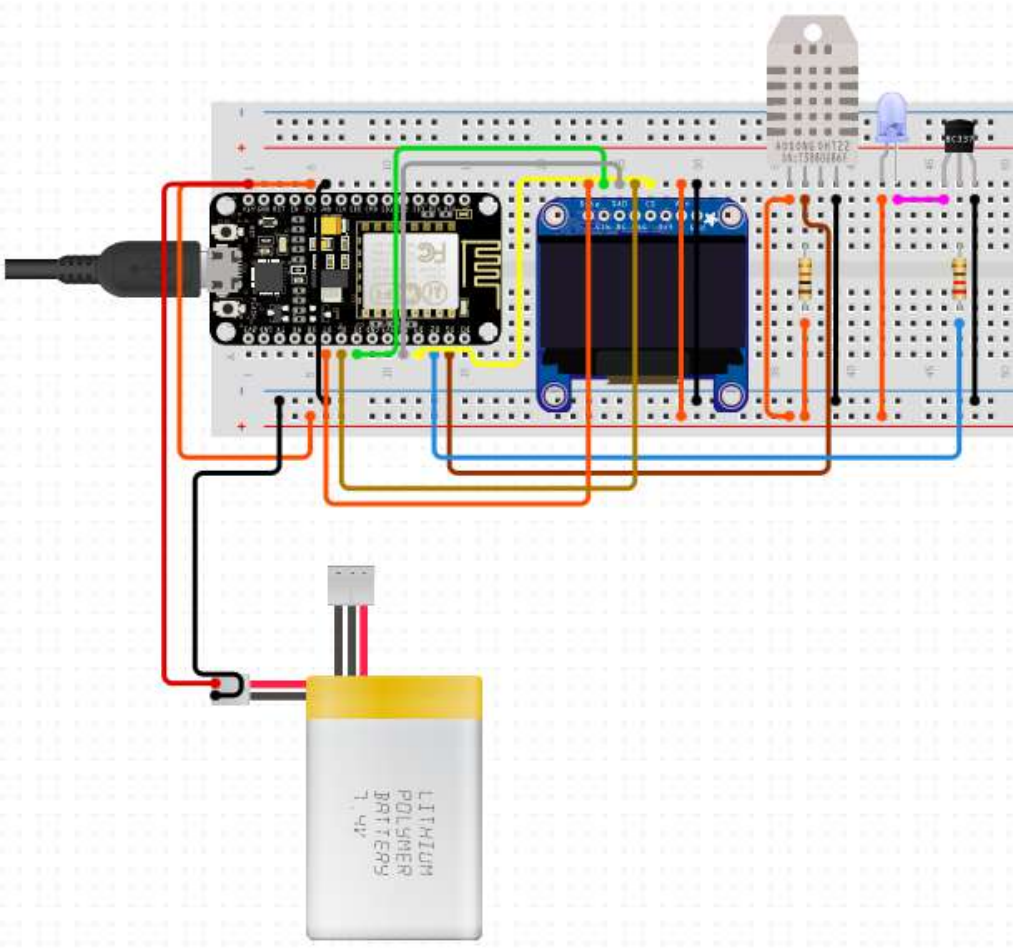
- Mobile application will send a signal to the internet modem.
- Internet modem will then transmit that signal to the individual module.
- The module will transmit the specific task in the form of IR signal to the specific air conditioner.
- The AC will perform the required task and report back.

Component Description:

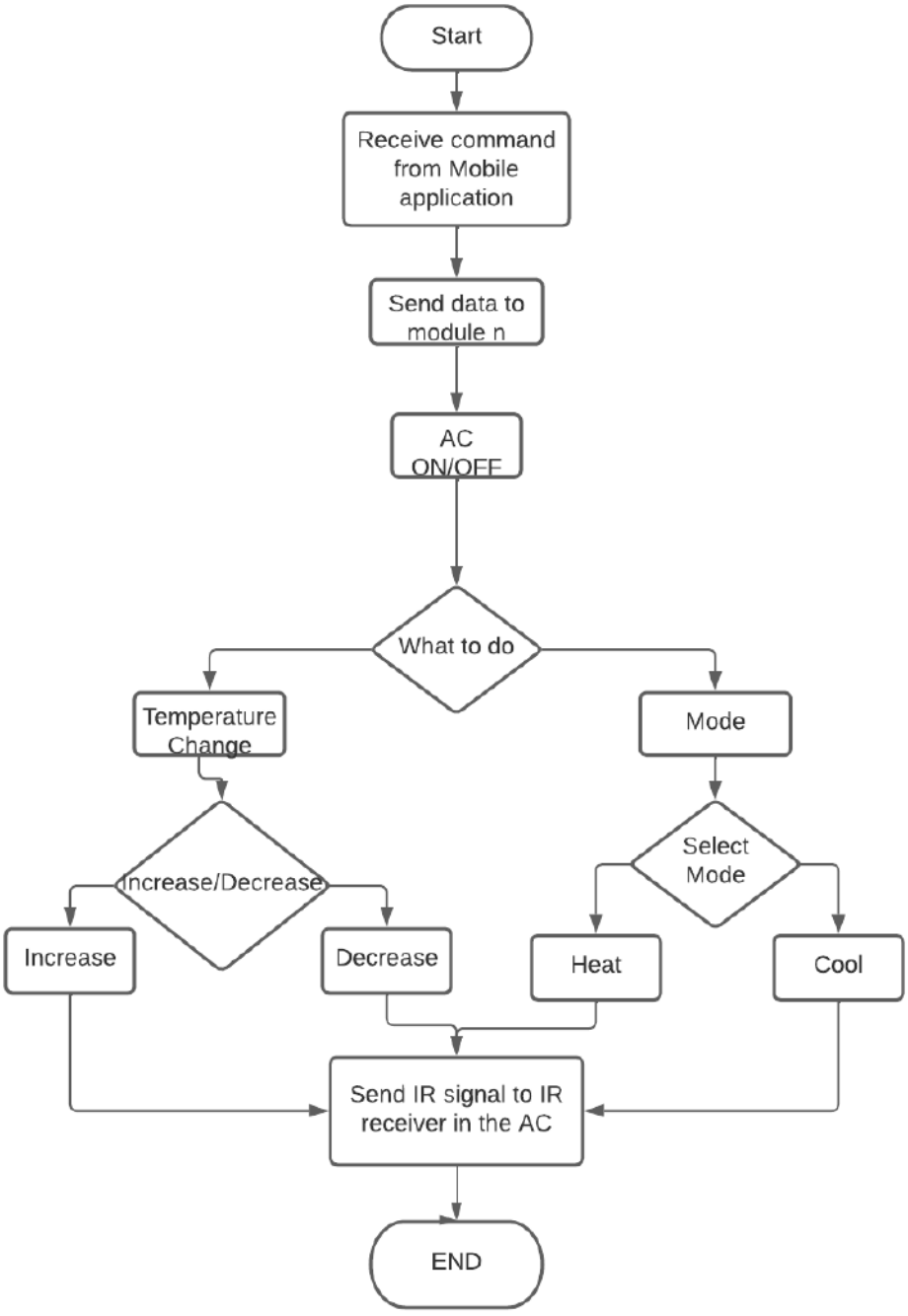
Discussed below are the components that are used in our modules.

- Esp8266 (NodeMCU) used for the design of the module to be placed in front of the AC, with which all components will be interfaced.
- DHT11 interfaced with ESP8266 which will provide real-time information about the temperature and humidity of the room.
- NPN transistor (2N222) used for low-power amplifying and for switching along with resistors.
- IR Led connected with the transistor which will relay the signal to the AC about what operation to perform.
- OLED interfaced to display the temperature and humidity values.
- 9V battery pack to power the module

Optimized Circuit Design for Module:



Flow Chart:


Start
Receive command from Mobile application
Send data to module n
AC ON/OFF
What to do
Temperature Change
Mode
Increase/Decrease
Select Mode
Increase
Decrease
Heat
Cool
Send IR signal to IR receiver in the AC
END

Description:
The mobile app dashboard enables users to control their air conditioners effortlessly. Commands are transmitted from the app to a Wi-Fi-connected module placed in front of each AC unit. Prior to executing a command, the system checks the current status of the AC to determine if it is already on or off. Once verified, the module utilizes infrared (IR) transmission to relay the command to the AC. This centralized control mechanism eliminates the need for physical room visits and promotes efficient energy usage. Users can conveniently manage their ACs across different rooms or spaces, enhancing convenience and reducing costs

Hardware Implementation:

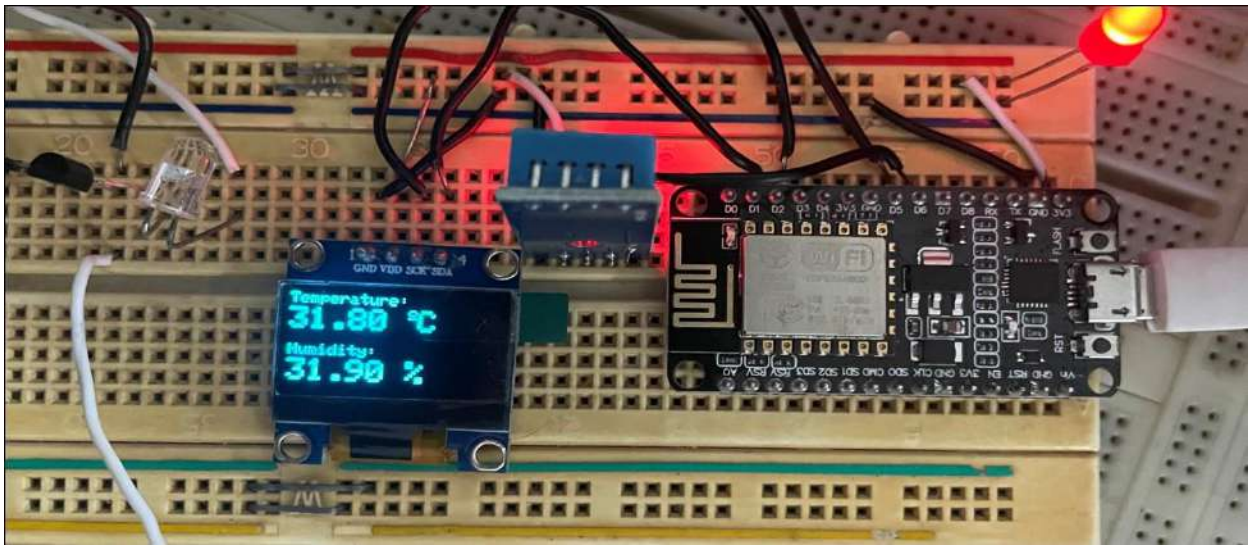


Fig. 3

Figure 3 is the circuit for our individual module. We have a microcontroller, ESP8266 (NodeMCU) connected with a DHT11 that is connected to the OLED for display. The IR LED is connected to the microcontroller via an NPN (2N222) transistor for low power amplification and switching. The circuit requires an operational voltage of 5V to send signal properly.

Hardware Description:

ESP8266 (NodeMCU):
The ESP8266 is a popular Wi-Fi module that enables microcontrollers or other embedded devices to connect to the internet and communicate wirelessly. The key features are:

1. **Wi-Fi connectivity**: The module integrates a Wi-Fi transceiver, allowing devices to connect to wireless networks and communicate over the internet.

2. **Microcontroller functionality:** The ESP8266 includes a built-in microcontroller unit which provides processing power for executing code and controlling connected devices.

3. **GPIO pins:** The module provides GPIO pins for interfacing with external sensors, actuators, and electronic components.

4. **Programming capabilities:** Languages including ArduinoIDE, MicroPython, and LC/C++ can be used to program the ESP8266. We used ArduinoIDE to program our microcontrollers for the project.

5. **Low power consumption:** The low-power design of the module makes it ideal for battery-powered applications such as ours.

DHT11:
The DHT11 is an affordable temperature and humidity sensor commonly used in digital format with microcontrollers. Its primary features include providing accurate temperature and humidity readings at a low cost. Key features include:

1. **Temperature and humidity measurement:** With an accuracy of ±2°C, the DHT11 sensor can measure temperatures ranging from 0°C to 50°C (32°F to 122°F). Additionally, it can accurately measure humidity from 20% to 90% with an accuracy of ±5%, making it well-suited for our project's needs.

2. **Digital output:** The sensor uses a one-wire communication protocol to provide temperature and humidity data in a digital format. This makes it easy to interface with microcontrollers as it requires only one data pin.

3. **Low power consumption:** The low-power design of DHT11 makes it ideal for battery-powered applications such as ours.

4. **Simple interface:** The sensor has only three pins—VCC (power supply), GND (ground), and a data pin for communication. It can be connected directly to microcontrollers without the need for additional components.

5. **Wide availability:** The DHT11 sensor is widely available and can be purchased from various electronics suppliers, making it easily accessible.

OLED:
OLED displays are commonly used in small-scale electronics projects, wearable devices, IoT applications, and embedded systems where a compact and low-power display is required. They are often interfaced with microcontrollers for displaying information, graphical user interfaces (GUIs), or visual feedback. Key features of OLED include:

1. **Fast response time:** OLED displays have a fast response time, which means they can quickly switch between different images or frames.

2. **Power efficiency:** OLED displays are power-efficient because they only consume energy for the pixels that are active

NPN (2N222) Transistor:
The NPN 2N2222 transistor is a widely used bipolar junction transistor (BJT) which is used commonly for switching and amplification purposes in electronic circuits.
Key features of NPN 2N2222 transistor include:

1. **Pinout:** The transistor typically has three leads or pins: the emitter (E), the base (B), and the collector (C). The emitter is typically marked with a protrusion or a dot on the transistor package.

2. **Amplification:** The NPN 2N2222 transistor has a moderate current gain, ranging from 100 to 300. This makes it suitable for small-signal amplification applications.

3. **Switching Speed:** The NPN 2N2222 transistor has a moderate switching speed, which allows it to be used in applications that require switching between on and off states relatively quickly.

4. **Common Uses:** Due to its versatility and availability, the 2N2222 transistor is commonly used in various circuits, including small audio amplifiers, switching applications, motor drivers, and signal amplification stages.

IR LED:
An IR LED emits infrared light rather than visible light, distinguishing it as a specialized type of LED. It is commonly used for applications that involve infrared communication, remote control systems, proximity sensors, and other similar purposes.
Key features of IR LEDs include:

1. **Infrared light emission:** Unlike regular LEDs that emit visible light, IR LEDs emit infrared light, which has a wavelength longer than that of visible light. This light is typically in the infrared spectrum, often in the range of 700 nm to 1 mm.

2. **Near-invisible to human eyes:** Since the emitted light falls outside the range of human vision, IR LEDs appear dim or even invisible to the naked eye. However, certain digital cameras and smart phone cameras can detect infrared light.

3. **Application in infrared communication:** IR LEDs are commonly used in devices that utilize infrared communication protocols, such as remote controls for televisions, DVD players, and other consumer electronics. They emit IR signals that are received and interpreted by corresponding IR receivers.

4. **Commonly available:** IR LEDs are widely available and can be found in various shapes, sizes, and packages. They can be easily integrated into electronic circuits and are often compatible with standard through-hole or surface mount (SMD) soldering techniques.

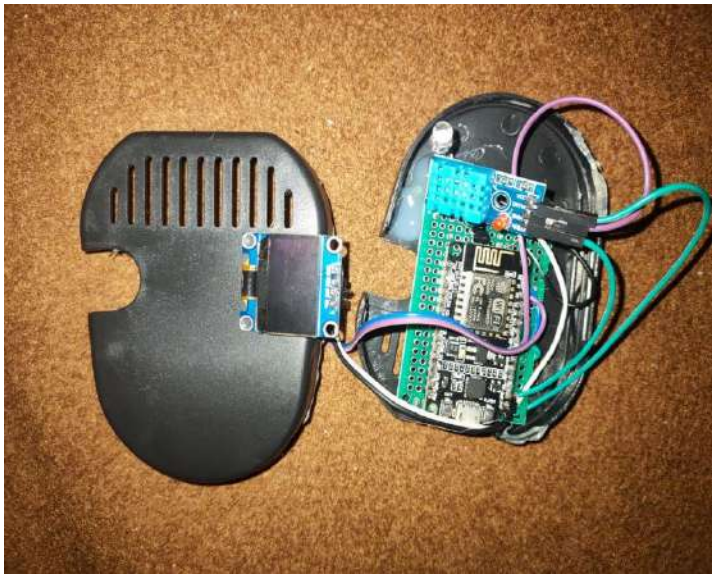
Chapter 3

Overview

In this chapter, we discuss our results, our conclusions and the recommendations for our project.

Results and Recommendations

Finalized Module Design:



The module consists of an ESP8266 interfaced with an IR Led that relays the information to the AC. The DHT11 is also interfaced along with an OLED, where the DHT11 provides real-time information about the temperature and humidity and the OLED is used to display it.

Mobile App Dashboard:

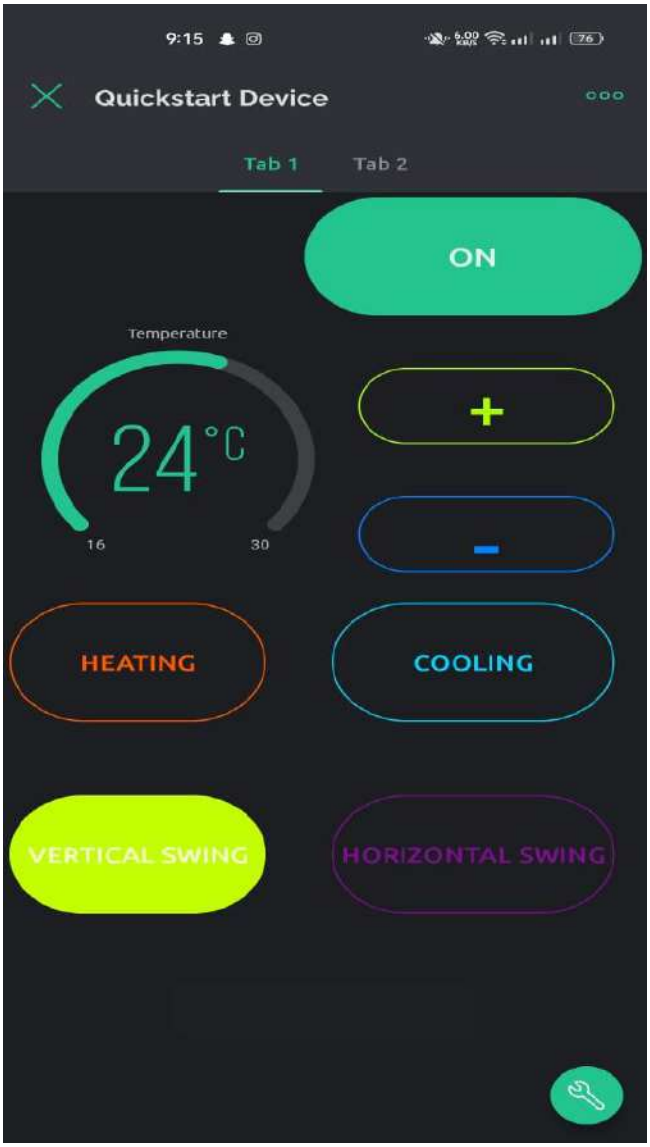
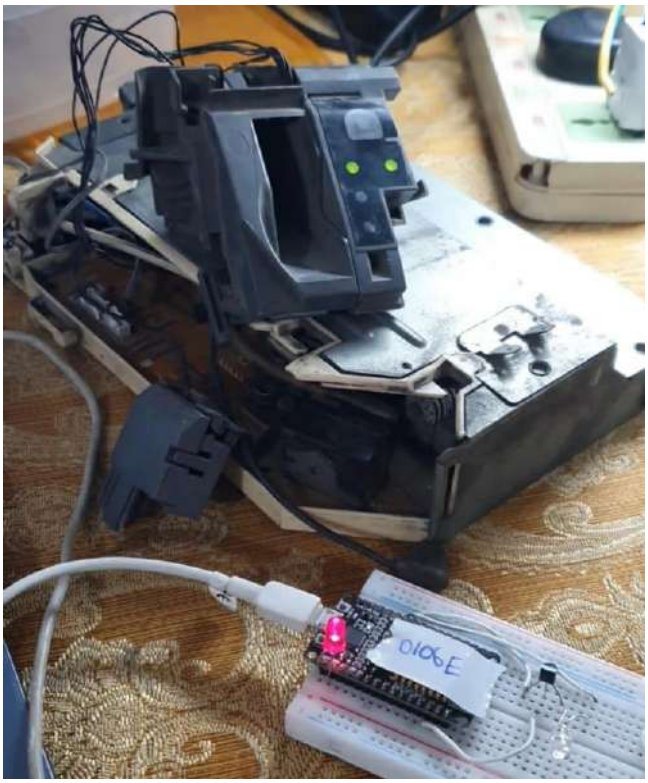


Fig. 4

Figure 4 shows the mobile app dashboard to control the AC. It consists of buttons of all the functions that are used to control the AC and get the required operations from it. The user simply taps the button of the required operation that he wants to perform and the information is then relayed to the module via Wi-Fi. There is also a temperature gauge that shows the temperature at which is the AC is currently running.

Modules and their Respective AC Kits:



Mitsubishi AC Kit with Module

Gree AC Kit with Module

Comments and Conclusion:

The project works perfectly according to the requirements and pre-requisites mentioned. As the command button is pressed from the mobile app dashboard, the signal is instantly relayed to the AC via the individual module, and the AC responds to the command as shown in the demo. Due to unavailability of ACs, we used AC kits to model the AC systems of three brands, namely Mitsubishi, Panasonic, and Gree. All three kits responded to their respective modules and carried out the required functions.

All in all, the project is a success in its working, design and most importantly its novelty and benefit that it would bring to the society in the form of tariff reductions and energy conservation.

Recommendations:

Some tweaks need to be made to the dashboard and the design of the module to turn it into a completely plug-and-play product for the end user. The module design can be made more compact and more cosmetically pleasing for the user.

Appendix: Project Codes

Codes for FYP-1

Code for Sender Circuit:

```
#include <IRremote.h>
IRsendirsend(3);

void setup()
{

}

void loop() {
  int khz = 37.9; // 38kHz carrier frequency for the NEC protocol
    unsigned int irSignal[] = { 499400, 9050, 4400, 650, 1650, 600, 1650, 550, 500, 650, 450, 650, 450,
650, 450, 650, 1650, 600, 1650, 550, 1650, 600, 1650, 600, 1650, 600, 1650, 600, 1650, 550, 550, 600,
450, 650, 1650, 600, 450, 650, 450, 650, 450, 650, 500, 600, 500, 600, 1650, 600, 1650, 600, 1650, 600,
500, 600, 450, 650, 500, 600, 500, 600, 500, 600, 500, 600, 500, 600, 550, 600, 450, 650, 450, 650, 500,
600, 500, 600, 500, 600, 1650, 600, 1650, 600, 500, 600, 500, 600, 500, 600, 500, 600, 500, 600, 500,
650, 450, 650, 450, 650, 500, 600, 500, 600, 500, 600, 500, 600, 500, 600, 500, 650, 450, 650};

irsend.sendRaw(irSignal, sizeof(irSignal) / sizeof(irSignal[0]), khz); //Note the approach used to
automatically calculate the size of the array.

delay(2000); //In this example, the signal will be repeated every 5 seconds, approximately.
}                                                                    s
```

Code for Receiver Circuit:

```
#include <Arduino.h>
#include <IRremote.h>

int buttonstate ;
int buttonpin= 0; // pin D3
int IRled= 5;// pin no D1
int RECV_PIN = 12; // pin D6

IRrecvirrecv(RECV_PIN);

decode_results results;

void setup()
{
Serial.begin(9600);
irrecv.enableIRIn(); // Start the receiver
pinMode(buttonpin , INPUT);
}

 Dumps out the decode_results structure.
```

```
 Call this after IRrecv::decode()
 void * to work around compiler issue
void dump(void *v) {
decode_results *results = (decode_results*)v
void dump(decode_results *results) {
  unsigned int  count = results->rawlen;
   if (results->decode_type == UNKNOWN) {
Serial.print("Unknown encoding: ");
  }
   else if (results->decode_type == NEC) {
Serial.print("Decoded NEC: ");
   }
   else if (results->decode_type == SONY) {
Serial.print("Decoded SONY: ");
   }
   else if (results->decode_type == RC5) {
Serial.print("Decoded RC5: ");
   }
   else if (results->decode_type == RC6) {
Serial.print("Decoded RC6: ");
   }
   else if (results->decode_type == PANASONIC) {
Serial.print("Decoded PANASONIC - Address: ");
     //Serial.print(results->panasonicAddress,HEX);
Serial.print(" Value: ");
   }
   else if (results->decode_type == LG) {
Serial.print("Decoded LG: ");
   }
   else if (results->decode_type == JVC) {
Serial.print("Decoded JVC: ");
   }
Serial.print(results->value, HEX);
Serial.print(" (");
Serial.print(results->bits, DEC);
Serial.println(" bits)");
Serial.print("Raw (");
Serial.print(count, DEC);
Serial.print("): ");

  for (int i = 0; i< count; i++) {
    if ((i % 2) == 1) {
Serial.print(results->rawbuf[i]*USECPERTICK, DEC);
Serial.print(",");
    }
    else {
Serial.print(-(int)results->rawbuf[i]*USECPERTICK, DEC);
Serial.print(",");
```

```
   }
Serial.print(" ");
  }
Serial.println("");
}
void loop() {
  if (irrecv.decode(&results)) {
Serial.println(results.value, HEX);
   dump(&results);
irrecv.resume(); // Receive the next value

  }
}
```

Codes for FYP-2

Code for Gree AC Module:

```
#include <Arduino.h>
#include <IRremoteESP8266.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Adafruit_Sensor.h>
#include <DHT.h>
#include <IRsend.h>
#include <ir_Gree.h>
#include <ESP8266WiFi.h>
#include <BlynkSimpleEsp8266.h>

#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels

#define BLYNK_TEMPLATE_ID           "TMPL6WubH6Esc"
#define BLYNK_TEMPLATE_NAME         "Bed Room"
#define BLYNK_AUTH_TOKEN            "p1rVqyN8gz7vEEHBYKuwl4ilP8CMod2L"

#define BLYNK_PRINT Serial

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);

#define DHTPIN 12     // Digital pin connected to the DHT sensor

#define DHTTYPE    DHT11

DHT dht(DHTPIN, DHTTYPE);


char ssid[] = "Jayyed";
char pass[] = "123456789";
int acMode, temo_up,temp_down, v_swing, h_swing, cool, heat ;
int temp=22;
```

```
BlynkTimer timer;


const uint16_t kIrLed = 14;  // ESP8266 GPIO pin to use. Recommended: 4 (D2).
IRGreeAC ac(kIrLed);  // Set the GPIO to be used for sending messages.

BLYNK_WRITE(V0) // on/off
{
  // Set incoming value from pin V0 to a variable
acMode = param.asInt();
  if (acMode==1) {
Serial.println(" AC on ");
ac.on();

  }
  else {
Serial.println(" AC off ");
ac.off();
  }
#if SEND_GREE
Serial.println("Sending IR command to A/C ...");
ac.send();
#endif  // SEND_GREE
printState();

}
BLYNK_WRITE(V1) //temp up
{
temo_up = param.asInt(); // get the value of the mode button
Serial.println("temp up");
  temp=temp+1;
ac.setTemp(temp);
#if SEND_GREE
Serial.println("Sending IR command to A/C ...");
ac.send();
#endif  // SEND_GREE
printState();
}

BLYNK_WRITE(V2)
{
temp_down = param.asInt(); // get the value of the mode button
Serial.println("temp down");
  temp=temp-1;
ac.setTemp(temp);
#if SEND_GREE
Serial.println("Sending IR command to A/C ...");
ac.send();
#endif  // SEND_GREE
printState();
}
BLYNK_WRITE(V3)
```

```
{
  heat = param.asInt(); // get the value of the mode button
Serial.println("heating");

ac.setMode(kGreeHeat);
#if SEND_GREE
Serial.println("Sending IR command to A/C ...");
ac.send();
#endif  // SEND_GREE
printState();

}
BLYNK_WRITE(V4)
{
  cool = param.asInt(); // get the value of the mode button
Serial.println("cooling");

  //Serial.println(V4);
ac.setMode(kGreeCool);
#if SEND_GREE
Serial.println("Sending IR command to A/C ...");
ac.send();
#endif  // SEND_GREE
printState();

}
BLYNK_WRITE(V5)
{
v_swing = param.asInt(); // get the value of the mode button
Serial.println("vertical swing");

  //Serial.println(V5);
  if (V5==5) {
ac.setSwingVertical(true, kGreeSwingAuto);

  }
  else {
ac.setSwingVertical(false, kGreeSwingAuto);

  }
#if SEND_GREE
Serial.println("Sending IR command to A/C ...");
ac.send();
#endif  // SEND_GREE
printState()
}

void printState() {
Serial.println("GREE A/C remote is in the following state:");
Serial.printf("  %s\n", ac.toString().c_str());
  // Display the encoded IR sequence.
  unsigned char* ir_code = ac.getRaw();
```

```
Serial.print("IR Code: 0x");
  for (uint8_t i = 0; i<kGreeStateLength; i++)
Serial.printf("%02X", ir_code[i]);
Serial.println();
}

BLYNK_CONNECTED()
{
}

void setup() {

   // Debug console
Serial.begin(115200);

dht.begin();

if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
Serial.println(F("SSD1306 allocation failed"));
for(;;);
 }
delay(500);
display.clearDisplay();
display.setTextColor(WHITE);

Blynk.begin(BLYNK_AUTH_TOKEN, ssid, pass);

ac.begin();
  //Serial.begin(115200);
delay(200);


Serial.println("Default state of the remote.");
printState();
Serial.println("Setting desired state for A/C.");
}

void loop() {
delay(500);

  //read temperature and humidity
  float t = dht.readTemperature();
  float h = dht.readHumidity();
  if (isnan(h) || isnan(t))
  {
Serial.println("Failed to read from DHT sensor!");
  }
  // clear display
display.clearDisplay();

  // display temperature
display.setTextSize(1);
```

```
display.setCursor(0,0);
display.print("Temperature: ");
display.setTextSize(2);
display.setCursor(0,10);
display.print(t);
display.print(" ");
display.setTextSize(1);
display.cp437(true);
display.write(167);
display.setTextSize(2);
display.print("C");

  // display humidity
display.setTextSize(1);
display.setCursor(0, 35);
display.print("Humidity: ");
display.setTextSize(2);
display.setCursor(0, 45);
display.print(h);
display.print(" %");

display.display();
Blynk.run();
timer.run();


Blynk.virtualWrite(V7, temp);
}
```

Code for Panasonic AC Module:

```
#include <Arduino.h>
#include <IRremoteESP8266.h>
#include <IRsend.h>
#include <ir_Panasonic.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Adafruit_Sensor.h>
#include <DHT.h>
#include <ESP8266WiFi.h>
#include <BlynkSimpleEsp8266.h>

#define BLYNK_TEMPLATE_ID           "TMPL6WubH6Esc"
#define BLYNK_TEMPLATE_NAME         "Quickstart Device"
#define BLYNK_AUTH_TOKEN            "KYNPuwm4IDkLQFa4Yoc-wmheNHmPQkMo"


/* Comment this out to disable prints and save space */
#define BLYNK_PRINT Serial

#define SCREEN_WIDTH 128 // OLED display width, in pixels
```

```
#define SCREEN_HEIGHT 64 // OLED display height, in pixels

// Declaration for an SSD1306 display connected to I2C (SDA, SCL pins)
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);

#define DHTPIN 12     // Digital pin connected to the DHT sensor

// Uncomment the type of sensor in use:
#define DHTTYPE    DHT11     // DHT 11

DHT dht(DHTPIN, DHTTYPE);


// Your WiFi credentials.
// Set password to "" for open networks.
char ssid[] = "Jayyed";
char pass[] = "123456789";
int acMode, temo_up,temp_down, v_swing, h_swing, cool, heat ;
int temp=22;
BlynkTimer timer;


const uint16_t kIrLed = 14;  // ESP8266 GPIO pin to use. Recommended: 4 (D2).
IRPanasonicAc ac(kIrLed);  // Set the GPIO to be used for sending messages.



BLYNK_WRITE(V0) // on/off
{
  // Set incoming value from pin V0 to a variable
acMode = param.asInt();
  if (acMode==1) {
Serial.println(" AC on ");
ac.on();

  }
  else {
Serial.println(" AC off ");
ac.off();
  }
#if SEND_PANASONIC_AC
Serial.println("Sending IR command to A/C ...");
ac.send();
#endif  // SEND_PANASONIC_AC
printState();

}


BLYNK_WRITE(V1) //temp up
{
temo_up = param.asInt(); // get the value of the mode button
```

```
Serial.println("temp up");
  temp=temp+1;
  //Serial.println(V1);
ac.setTemp(temp);
#if SEND_PANASONIC_AC
Serial.println("Sending IR command to A/C ...");
ac.send();
#endif  // SEND_PANASONIC_AC
printState();
}

BLYNK_WRITE(V2)
{
temp_down = param.asInt(); // get the value of the mode button
Serial.println("temp down");
  temp=temp-1;
  //Serial.println(V2);
ac.setTemp(temp);
#if SEND_PANASONIC_AC
Serial.println("Sending IR command to A/C ...");
ac.send();
#endif  // SEND_PANASONIC_AC
printState();
}
BLYNK_WRITE(V3)
{
  heat = param.asInt(); // get the value of the mode button
Serial.println("heating");

  //Serial.println(V3);
ac.setMode(kPanasonicAcHeat);
#if SEND_PANASONIC_AC
Serial.println("Sending IR command to A/C ...");
ac.send();
#endif  // SEND_PANASONIC_AC
printState();

}
BLYNK_WRITE(V4)
{
  cool = param.asInt(); // get the value of the mode button
Serial.println("cooling");

  //Serial.println(V4);
ac.setMode(kPanasonicAcCool);
#if SEND_PANASONIC_AC
Serial.println("Sending IR command to A/C ...");
ac.send();
#endif  // SEND_PANASONIC_AC
printState();

}
```

```
void printState() {
  // Display the settings.
Serial.println("Panasonic A/C remote is in the following state:");
Serial.printf("  %s\n", ac.toString().c_str());
  // Display the encoded IR sequence.
  unsigned char* ir_code = ac.getRaw();
Serial.print("IR Code: 0x");
  for (uint8_t i = 0; i<kPanasonicAcStateLength; i++)
Serial.printf("%02X", ir_code[i]);
Serial.println();
}

BLYNK_CONNECTED()
{

}

void setup() {

   // Debug console
Serial.begin(115200);

dht.begin();

if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
Serial.println(F("SSD1306 allocation failed"));
for(;;);
  }
delay(500);
display.clearDisplay();
display.setTextColor(WHITE);

Blynk.begin(BLYNK_AUTH_TOKEN, ssid, pass);



ac.begin();
  //Serial.begin(115200);
delay(200);

Serial.println("Default state of the remote.");
printState();
Serial.println("Setting desired state for A/C.");

}

void loop() {

delay(500);

  //read temperature and humidity
```

```
 float t = dht.readTemperature();
 float h = dht.readHumidity();
 if (isnan(h) || isnan(t))
 {
Serial.println("Failed to read from DHT sensor!");
 }
 // clear display
display.clearDisplay();

 // display temperature
display.setTextSize(1);
display.setCursor(0,0);
display.print("Temperature: ");
display.setTextSize(2);
display.setCursor(0,10);
display.print(t);
display.print(" ");
display.setTextSize(1);
display.cp437(true);
display.write(167);
display.setTextSize(2);
display.print("C");

 // display humidity
display.setTextSize(1);
display.setCursor(0, 35);
display.print("Humidity: ");
display.setTextSize(2);
display.setCursor(0, 45);
display.print(h);
display.print(" %");

display.display();


Blynk.run();
timer.run();



Blynk.virtualWrite(V7, temp);

}
```

Code for Mitsubishi AC Module:

```
#ifndef UNIT_TEST
#include <Arduino.h>
#endif
#include <IRremoteESP8266.h>
#include <IRsend.h>
#include <ir_Mitsubishi.h>
```

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Adafruit_Sensor.h>
#include <DHT.h>
#include <ESP8266WiFi.h>
#include <BlynkSimpleEsp8266.h>

#define BLYNK_TEMPLATE_ID           "TMPL6WubH6Esc"
#define BLYNK_TEMPLATE_NAME         "Quickstart Device"
#define BLYNK_AUTH_TOKEN            "aQPfQsm4V9EqOnWYZA9NlsVyooN8sect"

#define BLYNK_PRINT Serial

#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);

#define DHTPIN 12     // Digital pin connected to the DHT sensor

#define DHTTYPE    DHT11     // DHT 11

DHT dht(DHTPIN, DHTTYPE);

char ssid[] = "Jayyed";
char pass[] = "123456789";
int acMode, temo_up,temp_down, v_swing, h_swing, cool, heat ;
int temp=22;
BlynkTimer timer;

const uint16_t kIrLed = 14;  // ESP8266 GPIO pin to use. Recommended: 4 (D2).
IRMitsubishiAC ac(kIrLed);  // Set the GPIO used for sending messages.

BLYNK_WRITE(V0) // on/off
{
  // Set incoming value from pin V0 to a variable
acMode = param.asInt();
  if (acMode==1) {
Serial.println(" AC on ");
ac.on();

  }
  else {
Serial.println(" AC off ");
ac.off();
  }
#if SEND_MITSUBISHI_AC
Serial.println("Sending IR command to A/C ...");
ac.send();
#endif  // SEND_MITSUBISHI_AC
printState();
```

```
}
BLYNK_WRITE(V1) //temp up
{
temo_up = param.asInt(); // get the value of the mode button
Serial.println("temp up");
  temp=temp+1;
  //Serial.println(V1);
ac.setTemp(temp);
#if SEND_MITSUBISHI_AC
Serial.println("Sending IR command to A/C ...");
ac.send();
#endif  // SEND_MITSUBISHI_AC
printState();
}

BLYNK_WRITE(V2)
{
temp_down = param.asInt(); // get the value of the mode button
Serial.println("temp down");
  temp=temp-1;
  //Serial.println(V2);
ac.setTemp(temp);
#if SEND_MITSUBISHI_AC
Serial.println("Sending IR command to A/C ...");
ac.send();
#endif  // SEND_MITSUBISHI_AC
printState();
}
BLYNK_WRITE(V3)
{
  heat = param.asInt(); // get the value of the mode button
Serial.println("heating");

  //Serial.println(V3);
ac.setMode(kMitsubishiAcHeat);
#if SEND_MITSUBISHI_AC
Serial.println("Sending IR command to A/C ...");
ac.send();
#endif  // SEND_MITSUBISHI_AC
printState();

}
BLYNK_WRITE(V4)
{
  cool = param.asInt(); // get the value of the mode button
Serial.println("cooling");

  //Serial.println(V4);
ac.setMode(kMitsubishiAcCool);
#if SEND_MITSUBISHI_AC
Serial.println("Sending IR command to A/C ...");
```

```
ac.send();
#endif  // SEND_MITSUBISHI_AC
printState();

}

void printState() {
  // Display the settings.
Serial.println("Mitsubishi A/C remote is in the following state:");
Serial.printf("  %s\n", ac.toString().c_str());
  // Display the encoded IR sequence.
  unsigned char* ir_code = ac.getRaw();
Serial.print("IR Code: 0x");
  for (uint8_t i = 0; i<kMitsubishiACStateLength; i++)
Serial.printf("%02X", ir_code[i]);
Serial.println();
}

BLYNK_CONNECTED()
{

}

void setup() {

    // Debug console
Serial.begin(115200);

dht.begin();

if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
Serial.println(F("SSD1306 allocation failed"));
for(;;);
  }
delay(500);
display.clearDisplay();
display.setTextColor(WHITE);


Blynk.begin(BLYNK_AUTH_TOKEN, ssid, pass);



ac.begin();
  //Serial.begin(115200);
delay(200);

  // Set up what we want to send. See ir_Gree.cpp for all the options.
  // Most things default to off.
Serial.println("Default state of the remote.");
printState();
Serial.println("Setting desired state for A/C.");
```

```
}

void loop() {

delay(500);

  //read temperature and humidity
  float t = dht.readTemperature();
  float h = dht.readHumidity();
  if (isnan(h) || isnan(t))
  {
Serial.println("Failed to read from DHT sensor!");
  }
  // clear display
display.clearDisplay();

  // display temperature
display.setTextSize(1);
display.setCursor(0,0);
display.print("Temperature: ");
display.setTextSize(2);
display.setCursor(0,10);
display.print(t);
display.print(" ");
display.setTextSize(1);
display.cp437(true);
display.write(167);
display.setTextSize(2);
display.print("C");

  // display humidity
display.setTextSize(1);
display.setCursor(0, 35);
display.print("Humidity: ");
display.setTextSize(2);
display.setCursor(0, 45);
display.print(h);
display.print(" %");
display.display();
Blynk.run();
timer.run();

Blynk.virtualWrite(V7, temp);
}
```

Bibliography

[1] A. Mercier, "Universal infrared adapter for air conditioners," Feb. 2016. [Online]. Available: http://uu.diva-portal.org/smash/get/diva2:910960/FULLTEXT01.pdf. [Accessed Sep. 7, 2022].

[2] N. V. K. Ramesh, S. V. Tejesh Kumar, V. Vamsi, and S. Akarsh, "WI-FI CONTROLLED UNIVERSAL REMOTE USING ESP8266," ARP Journal of Engineering and Applied Sciences, vol. 1217, pp. 6607-6614, Nov. 2017. [Online]. Available: http://www.arpnjournals.org/jeas/research_papers/rp_2017/jeas_1117_6903.pdf. [Accessed Sep. 7, 2022].

[3] T. Kim, H. Lee, and Y. Chung, "Advanced universal remote controller for home automation and security," IEEE Journal on Selected Areas in Communications, vol. 35, no. 12, pp. 2782-2793, Dec. 2017. [Online]. Available: https://ieeexplore.ieee.org/document/8115722/. [Accessed Feb. 21, 2023]