DEPARTMENT OF INFORMATION TECHNOLOGY
FACULTY OF ENGINEERING & TECHNOLOGY

IOT PROJECT REPORT

SUBJECT TITLE : INTERNET OF THINGS

SUBJECT CODE: 15IT422E

SUBMITTED TO: Prof Kayalvizhi Jayavel

REMOTE CONTROLLED AC VIA WEB

JUNAID JEHANGIR BHAT RA1611008010165

SUMAIR BAWA RA16110080205

Department of Information Technology

SRM University, SRM Nagar, Kattankulathur-603203
Kanchipuram District, Tamil Nadu

LINKS TO GITHUB AND YOUTUBE:

YouTube:

https://youtu.be/6FulXu9u95Y

Github:

https://github.com/junaidwahab97/iotProject

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to our IoT Professor Mrs Kayalvizhi Jayavel for being out there at every step of our course and guiding us all along the way to be capable of moulding our ideas into smart projects.

We would also thank Mr Jagtheeshwaran Senthilvelan of Fluxgen Technologies who conducted a hands-on , three days workshop on IoT applications which helped us visualize how IoT is being used in the industry .

We would also like to thank our parents, without whom I wouldn't be able to anything in any regard.

ABSTRACT

Coming home in hot summer days and waiting for the AC to cool the room while we sweat makes us really uneasy. To devise a method and turn on the AC from anywhere with our mobile phone would solve this problem easily. All we need is an IoT implementation and solution to the problem.

This can be achieved via two ways either using a relay to control the current to the AC or using an IR setup to control the AC through IR signals.

Every AC has a remote using IR technology to control it. What I did was use the same technology to our disposal. Capturing the IR signal from the remote and then transmitting the same signal via an IR led does the job.

The user just needs to use a software switch via Adafruit to turn on and off the AC. All of the setup uses MQTT protocol and internet to communicate. Google assistant and Adafruit are used to run voice commands while IFTTT app is used to fuse them together.

HARDWARE REQUIRED:

- 1 * Nodemcu ESP8266 module.
- 1 * IR Receiver (TSOP1738).
- 1 * IR Led.
- 1 * NPN Transistor (2N3904).
- 1 * Breadboard
- Jumper Wires
- 1 * 300 Ohm Resistor
- A Computer.
- 1 * Relay Module.
- An Android phone.

SOFTWARE REQUIRED:

- Arduino IDE.
- ESP8266 library.
- IR Remote Library for ESP8266.
- Adafruit MQTT Library.
- Anrdoid OS.

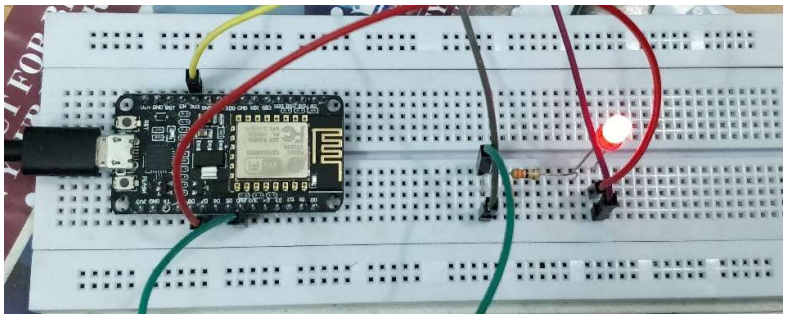
MISCELLANEOUS:

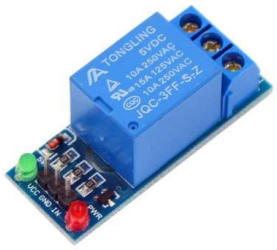
- WiFi Internet connection.
- Wire cutter.
- 1 * Micro USB Cable.

SYSTEM OVERVIEW

The main component of the setup is the Nodemcu ESP8266 module. All the other hardware components are connected to the Nodemcu. The board is programmed in Arduino IDE and uses the ESP8266, Arduino json, adafruit and irremote esp8266 libraries. These libraries have been added to the Arduino IDE.

The neutral wire of the air conditioner is connected to the relay and the relay is connected to the Nodemcu. The IR receiver is also connected to the Nodemcu module. The transistor is connected with IR led to transmit IR signal of the required AC.





CONTROLLING VIA RELAY:

- Connect relay to D7 pin.
- Give power via the 3.3V pin.
- Connect an LED and 300 Ohm resistor also to circuit.
- Setup the Nodemcu on the breadboard as shown.

CODE:

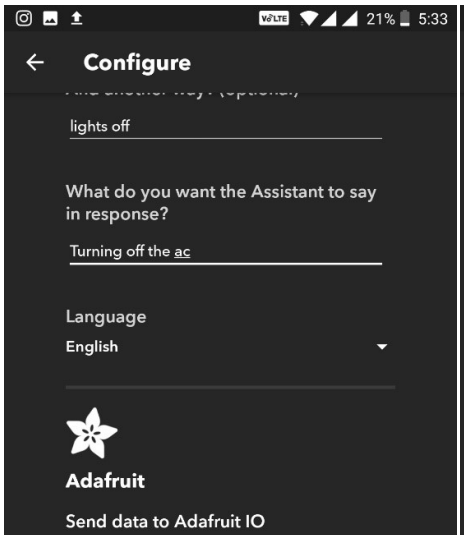
```
#include <ESP8266WiFi.h>

#include "Adafruit_MQTT.h"

#include "Adafruit_MQTT_Client.h"


/************************ WiFi Access Point
*********************************/


#define WLAN_SSID       "VGN 329"

#define WLAN_PASS       "yoyo"


/************************ Adafruit.io Setup
*********************************/


#define AIO_SERVER      "io.adafruit.com"

#define AIO_SERVERPORT  1883                   // use 8883 for SSL

#define AIO_USERNAME    "junaidwahab97"

#define AIO_KEY         "4a8f8c46321b4dc49998eac2bd926d77"


/************ Global State (you don't need to change this!)
******************/


// Create an ESP8266 WiFiClient class to connect to the MQTT server.

WiFiClient client;

// or... use WiFiFlientSecure for SSL
```

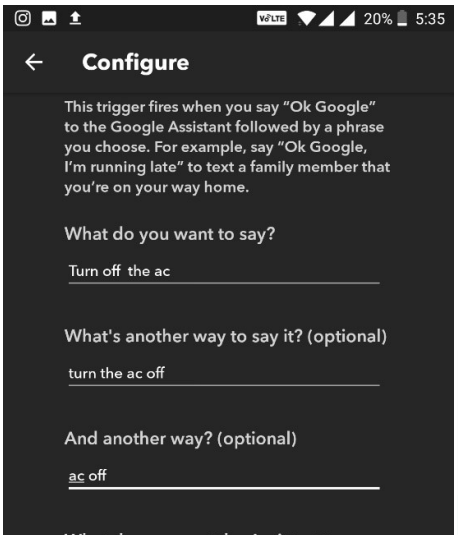
```
//WiFiClientSecure client;


// Setup the MQTT client class by passing in the WiFi client and MQTT
server and login details.

Adafruit_MQTT_Client mqtt(&client, AIO_SERVER, AIO_SERVERPORT,
AIO_USERNAME, AIO_KEY);


/************************** Feeds
***********************************/


// Setup a feed called 'photocell' for publishing.

// Notice MQTT paths for AIO follow the form: <username>/feeds/<feedname>

//Adafruit_MQTT_Publish photocell = Adafruit_MQTT_Publish(&mqtt,
AIO_USERNAME "/feeds/photocell");


// Setup a feed called 'onoff' for subscribing to changes.

Adafruit_MQTT_Subscribe onoffbutton = Adafruit_MQTT_Subscribe(&mqtt,
AIO_USERNAME "/feeds/Led");


/************************* Sketch Code
************************************/


// Bug workaround for Arduino 1.6.6, it seems to need a function
declaration

// for some reason (only affects ESP8266, likely an arduino-builder bug).

void MQTT_connect();


void setup() {

  Serial.begin(115200);

  delay(10);


  Serial.println(F("Adafruit MQTT demo"));


  // Connect to WiFi access point.
```

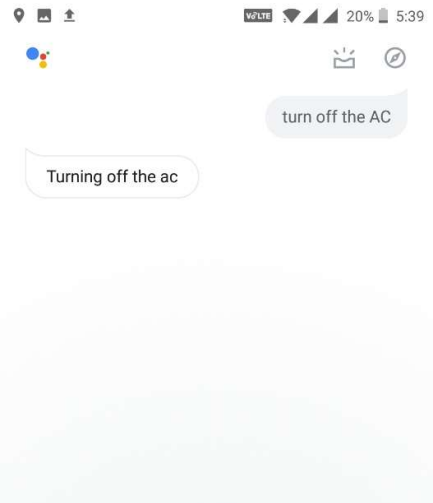
```
  Serial.println(); Serial.println();
  Serial.print("Connecting to ");
  Serial.println(WLAN_SSID);

  WiFi.begin(WLAN_SSID, WLAN_PASS);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println();

  Serial.println("WiFi connected");
  Serial.println("IP address: "); Serial.println(WiFi.localIP());

  // Setup MQTT subscription for onoff feed.
  mqtt.subscribe(&onoffbutton);
  pinMode(D7,OUTPUT);
}

uint32_t x=0;

void loop() {
  // Ensure the connection to the MQTT server is alive (this will make the
first
  // connection and automatically reconnect when disconnected).  See the
MQTT_connect
  // function definition further below.
  MQTT_connect();

  // this is our 'wait for incoming subscription packets' busy subloop
  // try to spend your time here
```

```
  Adafruit_MQTT_Subscribe *subscription;
  while ((subscription = mqtt.readSubscription(5000))) {
    if (subscription == &onoffbutton) {
      Serial.print(F("Got: "));
      Serial.println((char *)onoffbutton.lastread);
      int state = atoi((char *)onoffbutton.lastread);
      digitalWrite(D7,state);
    }
  }

  // Now we can publish stuff!
 /* Serial.print(F("\nSending photocell val "));
  Serial.print(x);
  Serial.print("...");
  if (! photocell.publish(x++)) {
    Serial.println(F("Failed"));
  } else {
    Serial.println(F("OK!"));
  }*/

  // ping the server to keep the mqtt connection alive
  // NOT required if you are publishing once every KEEPALIVE seconds
  /*
  if(! mqtt.ping()) {
    mqtt.disconnect();
  }
  */
}

// Function to connect and reconnect as necessary to the MQTT server.
// Should be called in the loop function and it will take care if
connecting.
```

```
void MQTT_connect() {
  int8_t ret;

  // Stop if already connected.
  if (mqtt.connected()) {
    return;
  }

  Serial.print("Connecting to MQTT... ");

  uint8_t retries = 3;
  while ((ret = mqtt.connect()) != 0) { // connect will return 0 for
connected
       Serial.println(mqtt.connectErrorString(ret));
       Serial.println("Retrying MQTT connection in 5 seconds...");
       mqtt.disconnect();
       delay(5000);  // wait 5 seconds
       retries--;
       if (retries == 0) {
         // basically die and wait for WDT to reset me
         while (1);
       }
  }
  Serial.println("MQTT Connected!");
}
```





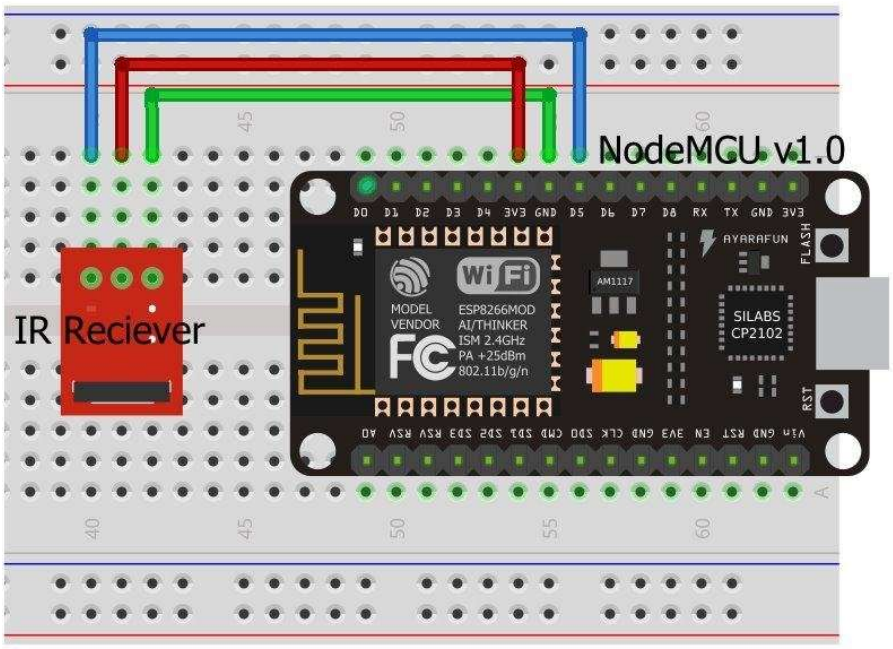


We give our WiFi SSID and password inside the code before compiling and uploading it to the Nodemcu. Also in order to connect to our adafruit database, we give the adafruit ID and key so that it can connect via MQTT.

We declare a variable in our adafruit database which takes either a value 0 or 1, 0 means off and 1 means on.

We then use IFTTT to connect our Google assistant and adafruit cloud so it changes the value of variable to 0 or 1 if we say off or on.

RECEIVING IR SIGNAL FROM AC REMOTE:



We need to first receive the IR signal from our remote in order to replicate it. We set the remote to the required settings then press the power on and off button then capture it with the receiver. Irremoteesp8266 library is used .

This step can be skipped if we already have the code of our AC remote.

CODE:

```
#ifndef UNIT_TEST
#include <Arduino.h>
#endif
#include <IRremoteESP8266.h>
#include <IRrecv.h>
#include <IRutils.h>
// The following are only needed for extended decoding of A/C Messages
#include <ir_Coolix.h>
#include <ir_Daikin.h>
#include <ir_Fujitsu.h>
#include <ir_Gree.h>
#include <ir_Haier.h>
#include <ir_Kelvinator.h>
#include <ir_Mitsubishi.h>
#include <ir_Midea.h>
#include <ir_Panasonic.h>
#include <ir_Samsung.h>
#include <ir_Toshiba.h>

// ==================== start of TUNEABLE PARAMETERS ====================
```

```
// An IR detector/demodulator is connected to GPIO pin 14
// e.g. D5 on a NodeMCU board.
const uint16_t kRecvPin = 14;

// The Serial connection baud rate.
// i.e. Status message will be sent to the PC at this baud rate.
// Try to avoid slow speeds like 9600, as you will miss messages and
// cause other problems. 115200 (or faster) is recommended.
// NOTE: Make sure you set your Serial Monitor to the same speed.
const uint32_t kBaudRate = 115200;

// As this program is a special purpose capture/decoder, let us use a
larger
// than normal buffer so we can handle Air Conditioner remote codes.
const uint16_t kCaptureBufferSize = 1024;

// kTimeout is the Nr. of milli-Seconds of no-more-data before we consider
a
// message ended.
// This parameter is an interesting trade-off. The longer the timeout, the
more
// complex a message it can capture. e.g. Some device protocols will send
// multiple message packets in quick succession, like Air Conditioner
remotes.
// Air Coniditioner protocols often have a considerable gap (20-40+ms)
between
// packets.
// The downside of a large timeout value is a lot of less complex
protocols
// send multiple messages when the remote's button is held down. The gap
between
// them is often also around 20+ms. This can result in the raw data be 2-
3+
// times larger than needed as it has captured 2-3+ messages in a single
// capture. Setting a low timeout value can resolve this.
```

```
// So, choosing the best kTimeout value for your use particular case is
// quite nuanced. Good luck and happy hunting.
// NOTE: Don't exceed kMaxTimeoutMs. Typically 130ms.
#if DECODE_AC
// Some A/C units have gaps in their protocols of ~40ms. e.g. Kelvinator
// A value this large may swallow repeats of some protocols
const uint8_t kTimeout = 50;
#else  // DECODE_AC
// Suits most messages, while not swallowing many repeats.
const uint8_t kTimeout = 15;
#endif  // DECODE_AC
// Alternatives:
// const uint8_t kTimeout = 90;
// Suits messages with big gaps like XMP-1 & some aircon units, but can
// accidentally swallow repeated messages in the rawData[] output.
//
// const uint8_t kTimeout = kMaxTimeoutMs;
// This will set it to our currently allowed maximum.
// Values this high are problematic because it is roughly the typical
boundary
// where most messages repeat.
// e.g. It will stop decoding a message and start sending it to serial at
//      precisely the time when the next message is likely to be
transmitted,
//      and may miss it.


// Set the smallest sized "UNKNOWN" message packets we actually care
about.
// This value helps reduce the false-positive detection rate of IR
background
// noise as real messages. The chances of background IR noise getting
detected
// as a message increases with the length of the kTimeout value. (See
above)
```

```
// The downside of setting this message too large is you can miss some
valid
// short messages for protocols that this library doesn't yet decode.
//
// Set higher if you get lots of random short UNKNOWN messages when
nothing
// should be sending a message.
// Set lower if you are sure your setup is working, but it doesn't see
messages
// from your device. (e.g. Other IR remotes work.)
// NOTE: Set this value very high to effectively turn off UNKNOWN
detection.
const uint16_t kMinUnknownSize = 12;
// ==================== end of TUNEABLE PARAMETERS ====================



// Use turn on the save buffer feature for more complete capture coverage.
IRrecv irrecv(kRecvPin, kCaptureBufferSize, kTimeout, true);


decode_results results;  // Somewhere to store the results


// Display the human readable state of an A/C message if we can.
void dumpACInfo(decode_results *results) {
  String description = "";
#if DECODE_DAIKIN
  if (results->decode_type == DAIKIN) {
    IRDaikinESP ac(0);
    ac.setRaw(results->state);
    description = ac.toString();
  }
#endif  // DECODE_DAIKIN
#if DECODE_FUJITSU_AC
  if (results->decode_type == FUJITSU_AC) {
```

```
    IRFujitsuAC ac(0);
    ac.setRaw(results->state, results->bits / 8);
    description = ac.toString();
  }
#endif  // DECODE_FUJITSU_AC
#if DECODE_KELVINATOR
  if (results->decode_type == KELVINATOR) {
    IRKelvinatorAC ac(0);
    ac.setRaw(results->state);
    description = ac.toString();
  }
#endif  // DECODE_KELVINATOR
#if DECODE_MITSUBISHI_AC
  if (results->decode_type == MITSUBISHI_AC) {
    IRMitsubishiAC ac(0);
    ac.setRaw(results->state);
    description = ac.toString();
  }
#endif  // DECODE_MITSUBISHI_AC
#if DECODE_TOSHIBA_AC
  if (results->decode_type == TOSHIBA_AC) {
    IRToshibaAC ac(0);
    ac.setRaw(results->state);
    description = ac.toString();
  }
#endif  // DECODE_TOSHIBA_AC
#if DECODE_GREE
  if (results->decode_type == GREE) {
    IRGreeAC ac(0);
    ac.setRaw(results->state);
    description = ac.toString();
  }
```

```
#endif  // DECODE_GREE
#if DECODE_MIDEA
  if (results->decode_type == MIDEA) {
    IRMideaAC ac(0);
    ac.setRaw(results->value);  // Midea uses value instead of state.
    description = ac.toString();
  }
#endif  // DECODE_MIDEA
#if DECODE_HAIER_AC
  if (results->decode_type == HAIER_AC) {
    IRHaierAC ac(0);
    ac.setRaw(results->state);
    description = ac.toString();
  }
#endif  // DECODE_HAIER_AC
#if DECODE_HAIER_AC_YRW02
  if (results->decode_type == HAIER_AC_YRW02) {
    IRHaierACYRW02 ac(0);
    ac.setRaw(results->state);
    description = ac.toString();
  }
#endif  // DECODE_HAIER_AC_YRW02
#if DECODE_SAMSUNG_AC
  if (results->decode_type == SAMSUNG_AC) {
    IRSamsungAc ac(0);
    ac.setRaw(results->state);
    description = ac.toString();
  }
#endif  // DECODE_SAMSUNG_AC
#if DECODE_COOLIX
  if (results->decode_type == COOLIX) {
    IRCoolixAC ac(0);
```

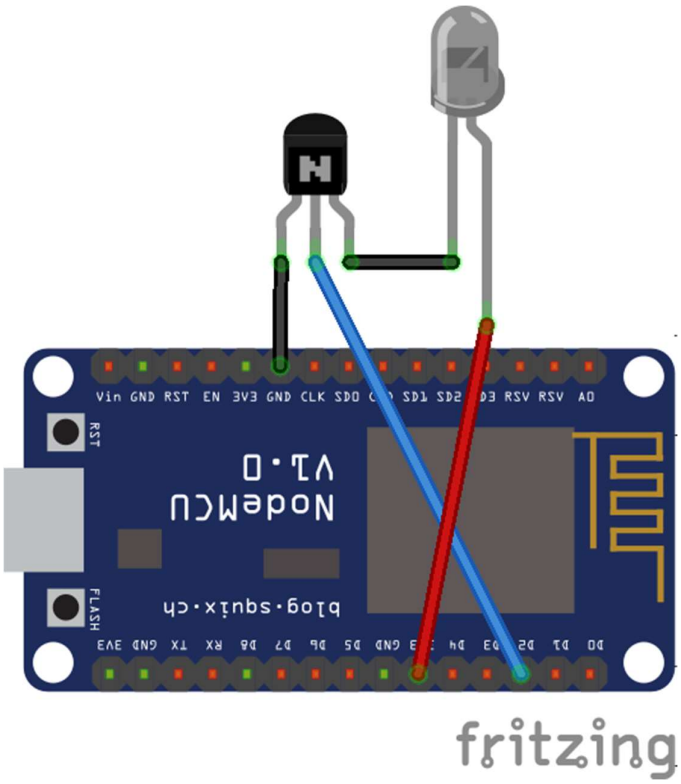
```
    ac.setRaw(results->value);  // Coolix uses value instead of state.
    description = ac.toString();
  }
#endif  // DECODE_COOLIX
#if DECODE_PANASONIC_AC
  if (results->decode_type == PANASONIC_AC &&
      results->bits > kPanasonicAcShortBits) {
    IRPanasonicAc ac(0);
    ac.setRaw(results->state);
    description = ac.toString();
  }
#endif  // DECODE_PANASONIC_AC
  // If we got a human-readable description of the message, display it.
  if (description != "")  Serial.println("Mesg Desc.: " + description);
}

// The section of code run only once at start-up.
void setup() {
  Serial.begin(kBaudRate, SERIAL_8N1, SERIAL_TX_ONLY);
  while (!Serial)  // Wait for the serial connection to be establised.
    delay(50);
  Serial.println();
  Serial.print("IRrecvDumpV2 is now running and waiting for IR input on
Pin ");
  Serial.println(kRecvPin);

#if DECODE_HASH
  // Ignore messages with less than minimum on or off pulses.
  irrecv.setUnknownThreshold(kMinUnknownSize);
#endif  // DECODE_HASH
  irrecv.enableIRIn();  // Start the receiver
}
```

```
// The repeating section of the code
//
void loop() {
  // Check if the IR code has been received.
  if (irrecv.decode(&results)) {
    // Display a crude timestamp.
    uint32_t now = millis();
    Serial.printf("Timestamp : %06u.%03u\n", now / 1000, now % 1000);
    if (results.overflow)
      Serial.printf("WARNING: IR code is too big for buffer (>= %d). "
                    "This result shouldn't be trusted until this is
resolved. "
                    "Edit & increase kCaptureBufferSize.\n",
                    kCaptureBufferSize);
    // Display the basic output of what we found.
    Serial.print(resultToHumanReadableBasic(&results));
    dumpACInfo(&results);  // Display any extra A/C info if we have it.
    yield();  // Feed the WDT as the text output can take a while to
print.

    // Display the library version the message was captured with.
    Serial.print("Library   : v");
    Serial.println(_IRREMOTEESP8266_VERSION_);
    Serial.println();

    // Output RAW timing info of the result.
    Serial.println(resultToTimingInfo(&results));
    yield();  // Feed the WDT (again)

    // Output the results as source code
    Serial.println(resultToSourceCode(&results));
    Serial.println("");  // Blank line between entries
```

```
    yield();  // Feed the WDT (again)
  }
}
```

TRANSMITTING IR SIGNAL:

To transmit the IR signal, we use an IR led and a NPN Transistor. The IR led transmits the IR signal to the AC which turns it on or off.



CODE:

```
/* Copyright 2017 sillyfrog
*
* An IR LED circuit *MUST* be connected to the ESP8266 on a pin
```

```
* as specified by kIrLed below.
*
* TL;DR: The IR LED needs to be driven by a transistor for a good result.
*
* Suggested circuit:
*     https://github.com/markszabo/IRremoteESP8266/wiki#ir-sending
*
* Common mistakes & tips:
*   * Don't just connect the IR LED directly to the pin, it won't
*     have enough current to drive the IR LED effectively.
*   * Make sure you have the IR LED polarity correct.
*     See: https://learn.sparkfun.com/tutorials/polarity/diode-and-led-polarity
*   * Typical digital camera/phones can be used to see if the IR LED is flashed.
*     Replace the IR LED with a normal LED if you don't have a digital camera
*     when debugging.
*   * Avoid using the following pins unless you really know what you are doing:
*     * Pin 0/D3: Can interfere with the boot/program mode & support circuits.
*     * Pin 1/TX/TXD0: Any serial transmissions from the ESP8266 will interfere.
*     * Pin 3/RX/RXD0: Any serial transmissions to the ESP8266 will interfere.
*   * ESP-01 modules are tricky. We suggest you use a module with more GPIOs
*     for your first time. e.g. ESP-12 etc.
*/

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

```
#include <IRsend.h>
#include <ir_Daikin.h>

const uint16_t kIrLed = 4;  // ESP8266 GPIO pin to use. Recommended: 4
(D2).
IRDaikinESP daikinir(kIrLed);  // Set the GPIO to be used to sending the
message

void setup() {
  daikinir.begin();
  Serial.begin(115200);
}



void loop() {
  Serial.println("Sending...");

  // Set up what we want to send. See ir_Daikin.cpp for all the options.
  daikinir.on();
  daikinir.setFan(1);
  daikinir.setMode(kDaikinCool);
  daikinir.setTemp(25);
  daikinir.setSwingVertical(false);
  daikinir.setSwingHorizontal(false);

  // Set the current time to 1:33PM (13:33)
  // Time works in minutes past midnight
  daikinir.setCurrentTime((13*60) + 33);
  // Turn off about 1 hour later at 2:30PM (15:30)
  daikinir.enableOffTimer((14*60) + 30);

  // Display what we are going to send.
  Serial.println(daikinir.toString());
```

```
  // Now send the IR signal.
#if SEND_DAIKIN
  daikinir.send();
#endif  // SEND_DAIKIN

  delay(15000);
}
```

AC TURNED ON

AC TURNED OFF

RESULT:

The AC was turned on and off successfully using both a relay and a IR setup.