TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

———————— * ————————

BÁO CÁO
ĐỒ ÁN HỆ NHÚNG

XÂY DỰNG BỘ ĐIỀU KHIỂN ĐA NĂNG SỬ DỤNG TÍN HIỆU HỒNG NGOẠI

Sinh viên thực hiện: **Nguyễn Huy Định**
MSSV 20161042
Lớp CNTT1.01-K61
Giáo viên hướng dẫn: **ThS. Lê Bá Vui**

HÀ NỘI 06-2020

MỤC LỤC

DANH MỤC BẢNG

DANH MỤC HÌNH VẼ

PHẦN I: NHIỆM VỤ ĐỒ ÁN

Xây dựng một thiết bị - bộ điều khiển đa năng sử dụng tín hiệu hồng ngoại, với hai chức năng chính là học tín hiệu hồng ngoại và phát tín hiệu điều khiển hồng ngoại. Học các tín hiệu từ các điều khiển từ xa của tivi, điều hòa, quạt,… sau đó có thể phát lại các tín hiệu này để điều khiển các thiết bị tương ứng. Thiết bị này có thể kết nối với ứng dụng chạy trên smartphone Android để tương tác với người dùng, người dùng có thể cho thiết bị học và lưu lại mã hồng ngoại của các nút điều khiển tương ứng và sử dụng.

Từ đó có thể tạo ra các kịch bản để trở thành bộ điều khiển thông minh có tính ứng dụng cao trong gia đình. Các kịch bản điều khiển gồm:

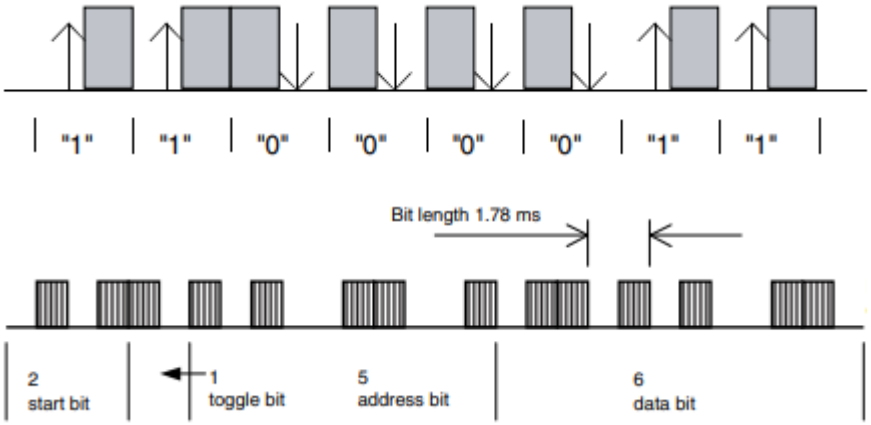
- Tự động bật tắt tivi, điều hòa, quạt theo thời gian hẹn trước.
- Điều khiển từ xa qua môi trường Internet.
- Tích hợp điều khiển bằng giọng nói.

PHẦN II: PHÂN TÍCH VÀ THIẾT KẾ

2.1. Giải pháp thu và lưu trữ tín hiệu điều khiển hồng ngoại

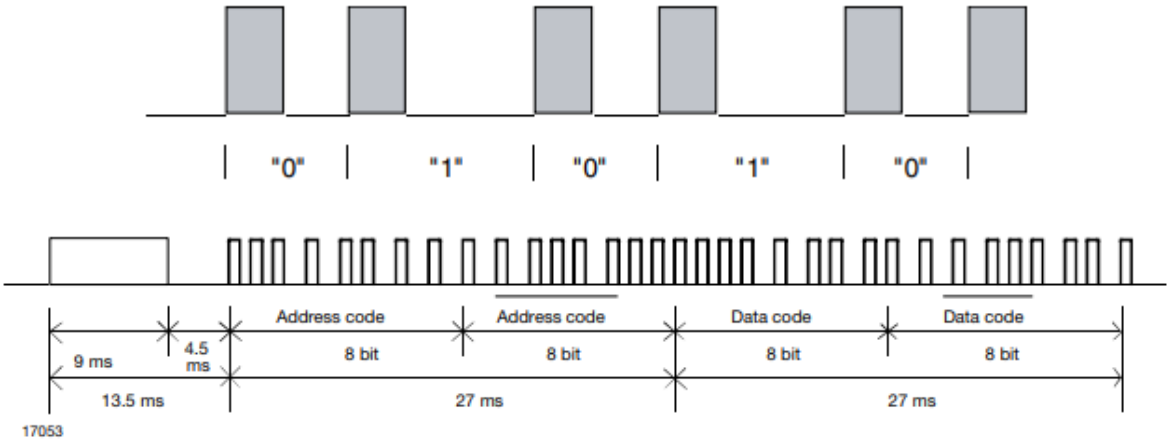
Tín hiệu hồng ngoại là một chùm sóng ánh sáng hồng ngoại mang thông tin dữ liệu cần truyền tải. Việc điều chế tín hiệu hồng ngoại sẽ phụ thuộc vào khoảng thời gian có tín hiệu và không có tín hiệu để quy định giá trị logic 0 và 1. Để phân biệt sóng hồng ngoại giữa các nguồn phát có trong tự nhiên và nguồn phát từ các thiết bị điều khiển, người ta sử dụng một tần số sóng mang trong khoảng từ 30 - 50kHz cho sóng hồng ngoại. Cụ thể, khi phát tín hiệu hồng ngoại ở trạng thái có tín hiệu, thay vì giữ trạng thái này liên tục trong khoảng thời gian đó, nguồn phát sẽ dao động giữa 2 trạng thái có tín hiệu và không có tín hiệu với tần số 30 - 50kHz. Nhờ đó mà bên thu sẽ điều chế và phân biệt được tín hiệu hồng ngoại điều khiển với các sóng hồng ngoại phát ra từ các nguồn khác.

Có 2 dạng mã hóa cơ bản và phổ biến cho các tín hiệu hồng ngoại đó là mã RC5 và mã NEC. Mã RC5 sử dụng tần số sóng mang 36kHz, biểu diễn các bit dữ liệu theo dạng bi-phase. Bit giá trị 0 được biểu diễn bằng sường xuống, bit giá trị 1 được biểu diễn bởi sường lên. Khuôn dạng dữ liệu bao gồm 2 start bits, 1 toggle bit đảo ngược sau mỗi lần nhấn nút mới, 5 bits địa chỉ định danh loại thiết bị và 6 bits dữ liệu.



Hình 1: Mã hóa RC5

Mã NEC sử dụng tần số sóng mang khoảng 38kHz, biểu diễn các bit theo dạng mã hóa độ rộng xung (pulse distance coding). Các bit quy định bởi một tín hiệu ở mức cao, kèm theo một tín hiệu ở mức thấp, tùy thuộc vào độ rộng của xung mức thấp sẽ tương ứng bit 0 - ngắn hay 1 – dài. Khuôn dạng của mã NEC bao gồm phần leadercode có tín hiệu mức cao trong khoảng 9ms và xuống thấp trong khoảng 4.5ms. Theo sau lần lượt là 8 bit địa chỉ, 8 bit đảo của địa chỉ, 8 bit dữ liệu và 8 bit đảo của dữ liệu.



Hình 2: Mã hóa NEC

Mỗi hãng thiết bị thông thường sẽ có một loại mã riêng, có thể tuân theo mã NEC hoặc tùy biến trên cơ sở mã NEC như thay đổi độ rộng xung của leadercode, thay đổi độ dài của data: 7 bits, 8bits, 12bits,… đối với tivi. Đối với điều hòa, do mỗi lần điều khiển thì dữ liệu truyền đi bao gồm toàn bộ trạng thái hoạt động của điều hòa nên độ dài có thể lên tới vài bytes đến chục bytes.

Giải pháp cho việc thu và lưu trữ tín hiệu điều khiển là sử dụng một module thu tín hiệu hồng ngoại, sẽ trả về kết quả 1 hoặc 0 tương ứng với có tín hiệu (mức cao) và không có tín hiệu (mức thấp). Lưu lại khoảng thời gian giữa các lần chuyển đổi trạng thái 0 và 1, thu được một chuỗi số giá trị trong khoảng từ 10 – 50000µs, độ dài chuỗi số tương ứng với số lần chuyển trạng thái. Việc phát lại trở lên dễ dàng bằng cách phát ra tín hiệu mức cao/thấp trong khoảng thời gian tương ứng (tín hiệu mức cao phải dao động cao thấp với tần số tương ứng, thường là 36kHz).

Ví dụ chuỗi “680, 500, 400, 500, 540, 340, 600, 1700, 1900” sẽ mang thông tin trạng thái tín hiệu như sau: mức cao 680µs, mức thấp 500µs, mức cao 400µs, mức thấp 500µs, mức cao 540µs,…

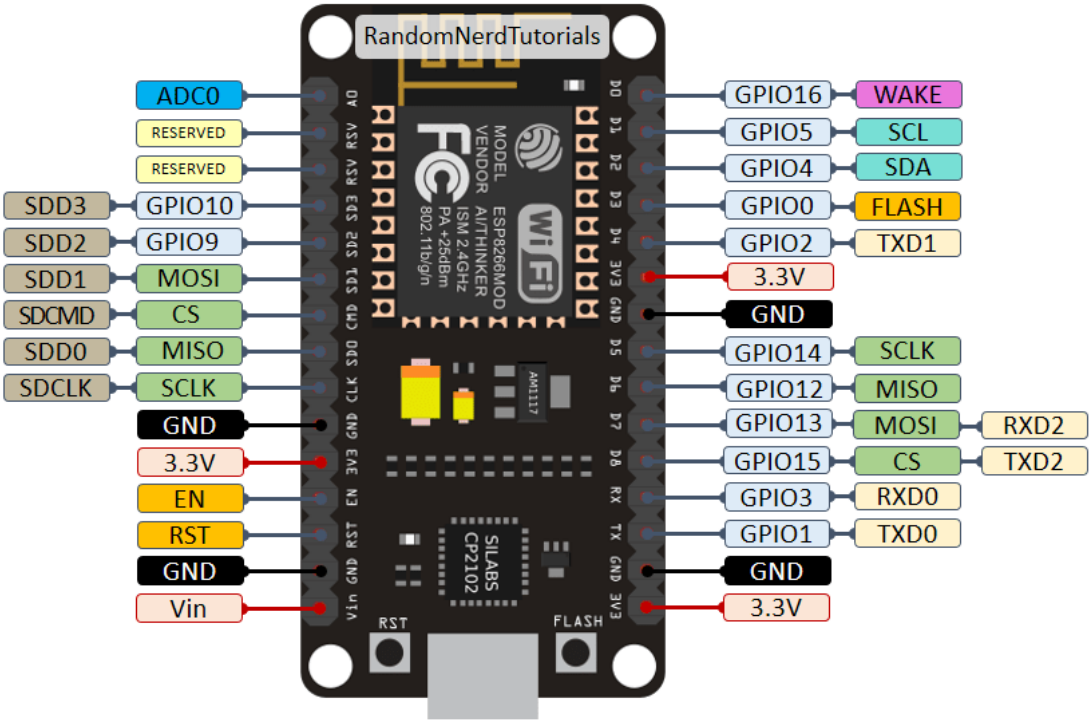
2.2. Phần cứng sử dụng

ESP8266 được sử dụng làm vi điều khiển cho thiết bị. ESP8266 là dòng chip tích hợp WiFi 2.4Ghz có thể lập trình bằng Arduino, với các thư viện kết nối WiFi hỗ trợ TCP, UDP và các giao thức tầng ứng dụng HTTP(s), DNS Servers,… có sẵn thư viện kết nối với Firebase rất tiện lợi cho việc xây dựng các dự án cần trao đổi và lưu trữ dữ liệu thời gian thực.

Thông số phần cứng của ESP8266 và kit NodeMCU ESP-12E:

- 32-bit RISC CPU 80MHz
- Hỗ trợ Flash từ 512KiB đến 4MiB
- 64Kbytes RAM thực thi lệnh
- 80Kbytes RAM dữ liệu người dùng
- 16Kbytes RAM dữ liệu hệ thống

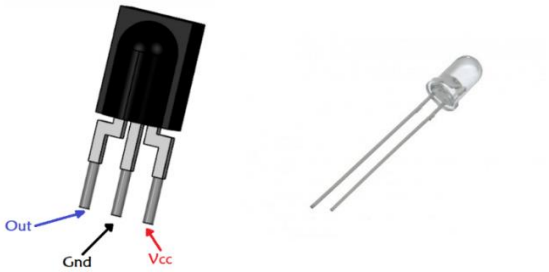
- Chuẩn wifi EEE 802.11 b/g/n, Wi-Fi 2.4GHz. Hỗ trợ WEP, WPA/WPA2
- Tích hợp giao thức TCP/IP, hỗ trợ các chuẩn SPI, I2C, I2S với DMA, UART
- 17 chân GPIO, 1 ADC 10-bit, watchdog timer
- Điện áp hoạt động: 3.3V
- Kit NodeMCU: kit tích hợp ESP8266
    - Tích hợp mạch reset, các mạch để chip hoạt động,… nút RST và FLASH
    - Tích hợp mạch chuyển đổi USB và UART



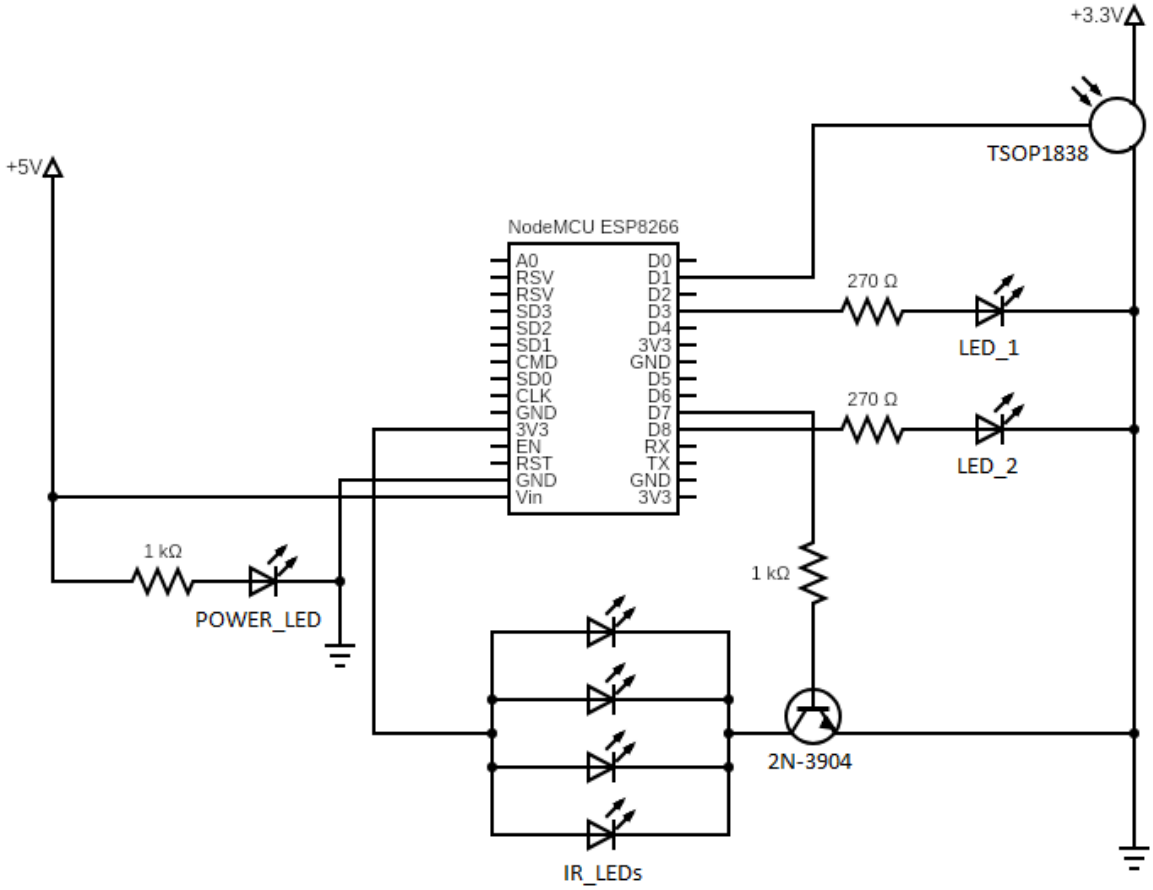
Hình 3: Sơ đồ chân kit NodeMCU ESP-12E sử dụng ESP8266

Module thu và phát tín hiệu hồng ngoại:

- Thu tín hiệu bằng IR Receiver TSOP1838:
    - Điện áp hoạt động: 2.7 - 5.5V
    - Tần số: 38KHz
    - Khoảng cách thu lên đến 18m
    - 1 chân tín hiệu ra digital ứng với trạng thái của tín hiệu thu
- Phát tín hiệu sử dụng IR LED – led hồng ngoại



Hình 4: Sơ đồ chân IR receiver TSOP1838



Hình 5: Sơ đồ ghép nối linh kiện cho thiết bị

Chức năng của các linh kiện:

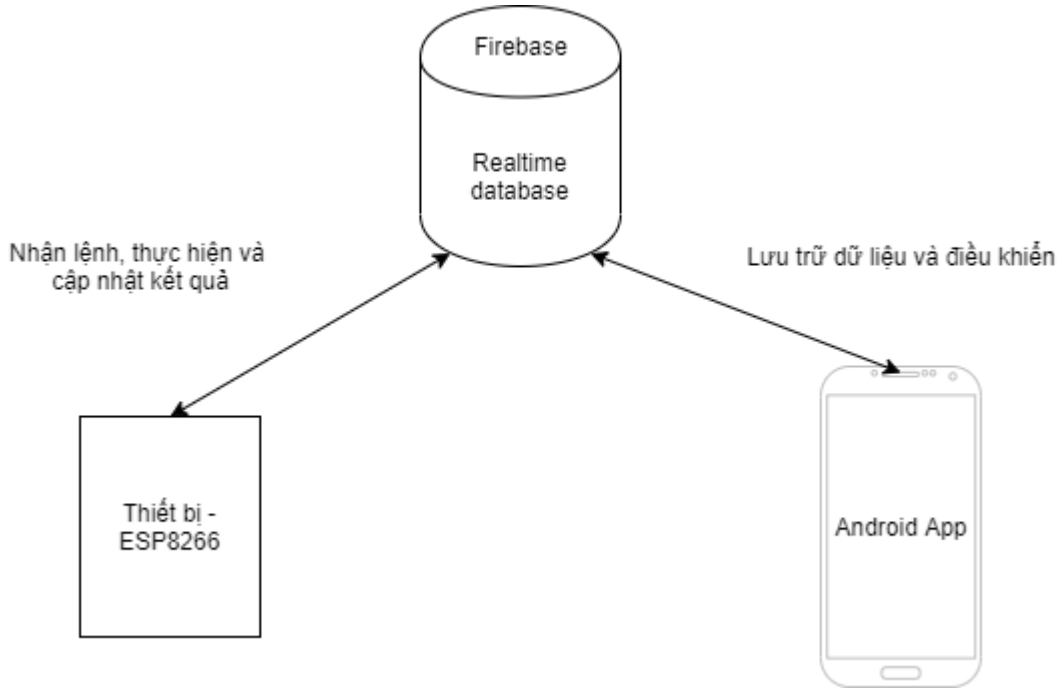
- NodeMCU ESP8266: kit sử dụng ESP8266.
- LED_1: báo trạng thái hoạt động của thiết bị, led sẽ sáng khi chuyển sang chế độ đọc tín hiệu hồng ngoại.
- LED_2: báo trạng thái kết nối WiFi của thiết bị, nhấp nháy khi đang kết nối đến mạng WiFi, sáng liên tục khi đang trong quá trình lấy dữ liệu lần đầu từ Firebase sau khi reset hoặc đang hoạt động ở chế độ webserver để cấu hình thông tin WiFi cho thiết bị, tắt khi hoạt động bình thường.
- POWER_LED: led báo nguồn.
- TSOP1838: module thu hồng ngoại.
- IR_LEDs: 4 leds hồng ngoại phát tín hiệu điều khiển hồng ngoại ra 4 hướng.
- 2N-3904: NPN-transistor để điều khiển 4 led hồng ngoại, chân D7 nối với cực B qua trở 1kΩ, khi D7 ở mức cao thì sẽ có dòng điện chạy từ cực C sang E làm các led hồng ngoại sáng.

2.3. Trao đổi dữ liệu sử dụng Firebase

Firebase là một dịch vụ cơ sở dữ liệu bao gồm realtime database (cơ sở dữ liệu thời gian thực) được cung cấp và phát triển bởi Google. Với realtime database, dữ liệu được lưu trữ theo dạng phi quan hệ, cụ thể là dưới dạng chuỗi JSON (key –

value). Dữ liệu được đồng bộ thời gian thực với tất cả các clients kết nối với nó, mỗi khi có sự cập nhật dữ liệu lên cơ sở dữ liệu, tất cả các clients sẽ đều được thông báo về sự thay đổi này. Firebase cung cấp các SDK và thư viện cho rất nhiều nền tảng khác nhau trong đó có Arduino và Android, do đó việc kết nối và cập nhật dữ liệu trở lên đơn giản bằng cách sử dụng các thư viện có sẵn mà không cần quan tâm đến các vấn đề phát sinh. Dữ liệu được lưu trữ bảo mật và mã hóa an toàn trên đường truyền.

Sử dụng một realtime database để lưu trữ các đoạn mã hồng ngoại đã được đọc bởi người dùng, đồng thời là một server trung gian cho trao đổi dữ liệu giữa ứng dụng và thiết bị. Ứng dụng trên Android sẽ điều khiển thiết bị thông bằng cách cập nhật các trường điều khiển tương ứng trên cơ sở dữ liệu. Thiết bị khi nhận thấy có sự thay đổi dữ liệu trên các trường điều khiển này sẽ thực thi lệnh tương ứng và cập nhật trạng thái, kết quả lên cơ sở dữ liệu để báo cho ứng dụng.



Hình 6: Mô hình trao đổi dữ liệu giữa ứng dụng và thiết bị

irremote-4f614 // Tên database
- **device** // Nhóm các keys điều khiển thiết bị
  - cmd: “OK” // Lệnh điều khiển và trạng thái thiết bị
  - ircode: “” // Mã hồng ngoại đọc được bởi thiết bị (string)
  - scode: “” // Mã hồng ngoại để thiết bị phát (string)
  - schedule // Danh sách các mã hồng ngoại để hẹn giờ
  - sduration // Danh sách thời điểm hẹn giờ
  - sloop // Danh sách chu kỳ lặp của tính năng hẹn giờ
  - sname // Danh sách các tên nút được hẹn giờ
  - sstatus // Trạng thái các timer hẹn giờ

- **irdata** // Nhóm các keys lưu mã hồng ngoại
  - AC: Panasonic // Danh sách các nút đã lưu từ ứng dụng
    - OFF: "" // Key: tên nút, value: mã hồng ngoại
    - ON: ""
  - AC: nec // "AC", "TV": loại thiết bị (điều hòa, tivi)
  - TV: Samsung // "Samsung", "Sony",…: tên hãng
  - TV: Sony // Tên được đặt bởi người dùng qua ứng dụng

2.4. Kịch bản điều khiển thiết bị từ ứng dụng

Khi người dùng chọn chế độ phát một tín hiệu hồng ngoại đã lưu, ứng dụng sẽ lấy mã hồng ngoại trong danh sách ở trường "irdata", cập nhật mã hồng ngoại này vào trường "scode", sau đó cập nhật trường "cmd" với giá trị "SEND". Thiết bị sẽ thực hiện phát tín hiệu và cập nhật trạng thái vào trường "cmd".

Khi người dùng chọn chế độ đọc tín hiệu hồng ngoại, ứng dụng sẽ cập nhật trường "cmd" giá trị là "READ", sau đó thiết bị sẽ chuyển sang chế độ thu, chờ người dùng sử dụng điều khiển để phát tín hiệu cho thiết bị thu. Sau khi thu được, thiết bị sẽ cập nhật trạng thái vào trường "cmd", mã hồng ngoại thu được sẽ được giải mã, nén và cập nhật vào trường "ircode". Ứng dụng Android sẽ cho người dùng nhập tên nút vừa thu được mã hồng ngoại sau đó lưu lại thông tin vào danh sách trong trường "irdata" trên Firebase. Dưới đây là bảng các trạng thái của trường "cmd".

SEND	Điều khiển thiết bị phát tín hiệu hồng ngoại trong "scode"
READ	Điều khiển thiết bị đọc tín hiệu hồng ngoại
CANCEL	Hủy lệnh, chuyển thiết bị về trạng thái chờ
SEND_OK	Báo phát tín hiệu thành công
SEND_ERROR	Báo lỗi khi phát tín hiệu
READ_OK	Báo đọc tín hiệu thành công, tín hiệu đọc được cập nhật vào trường "ircode"
READ_ERROR	Báo lỗi khi đọc tín hiệu
WAIT	Chờ trong quá trình thực thi lệnh
RESET	Thiết bị vừa khởi động lại

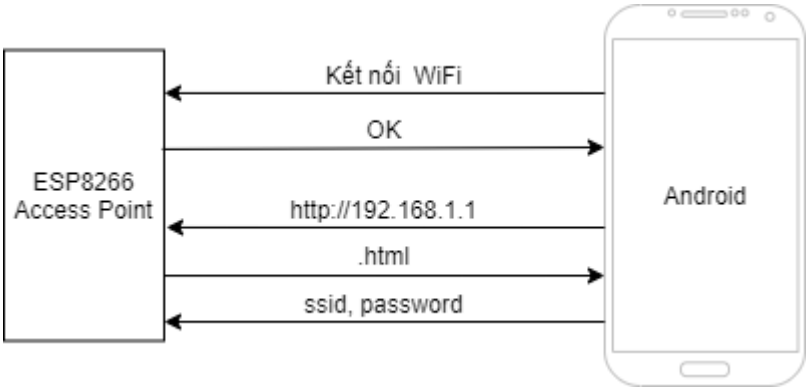
Bảng 1: Bảng các trạng thái của trường "cmd"

Tính năng hẹn giờ điều khiển phát nhiều mã hồng ngoại tại các thời điểm định trước. Ứng dụng sẽ cập nhật mã hồng ngoại vào một trong danh sách các mã ở trường "schedule", cập nhật thời điểm vào một trong danh sách các thời điểm ở trường "sduration", cập nhật chu kỳ lặp vào danh sách các chu kỳ ở trường "sloop", cập nhật tên nút, tên thiết bị điều khiển vào danh sách ở trường "sname", cuối cùng cập nhật vào danh sách trường "sstatus" giá trị "SET" để thiết lập hẹn giờ. Các lỗi và trạng thái trong quá trình cài đặt và hẹn giờ sẽ được cập nhật để báo cho ứng dụng thông qua danh sách ở trường "sstatus".

SET	Lệnh yêu cầu thiết bị thiết lập hẹn giờ từ các thông tin đã cập nhật trong các trường cần thiết
CANCEL hoặc “”	Hủy hẹn giờ
WAIT	Thiết bị đã kích hoạt hẹn giờ
DONE	Thiết bị báo đã thực hiện thành công, đã phát tín hiệu hồng ngoại vào đúng thời điểm
TIME_ERROR	Báo lỗi thời điểm không hợp lệ
INVALID_IRCODE	Báo lỗi mã hồng ngoại không hợp lệ
INVALID_LOOP	Báo lỗi giá trị chu kỳ không hợp lệ

Bảng 2: Bảng các trạng thái của trường “sstatus”

Ngoài việc điều khiển phát và thu tín hiệu hồng ngoại, cũng cần phải có kịch bản cho việc cấu hình thông tin điểm truy cập WiFi cho thiết bị từ smartphone Android, để thiết bị có thể kết nối WiFi và truy cập Internet. Thiết bị sẽ có thêm một chế độ hoạt động đó là chế độ Access Point, chạy Webserver. Ở chế độ này, điện thoại Android sẽ kết nối đến WiFi của thiết bị, truy cập vào trang web của thiết bị và điền thông tin WiFi (ssid, password) để thiết bị lưu trữ lại và sử dụng để kết nối WiFi, truy cập Internet.



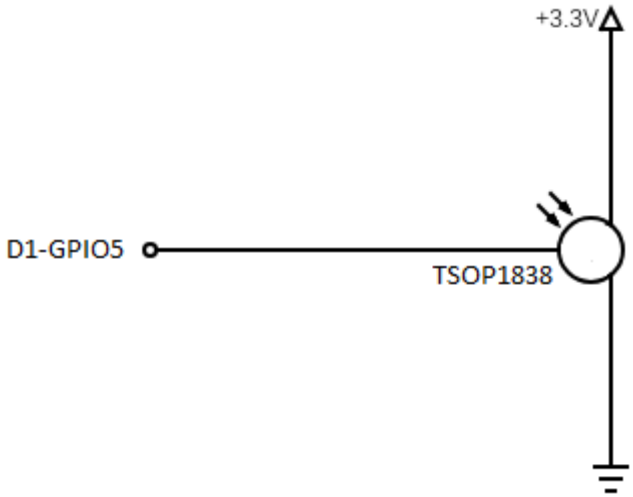
Hình 7: Kịch bản cấu hình thông tin WiFi cho thiết bị

PHẦN III: CÀI ĐẶT VÀ TRIỂN KHAI

Lập trình các tính năng cho thiết bị - bộ điều khiển đa năng sử dụng nền tảng Arduino, ứng dụng tương tác với người dùng và điều khiển thiết bị được xây dựng trên nền tảng Android.

3.1. Đọc tín hiệu hồng ngoại

Module TSOP1838 có chức năng đọc tín hiệu hồng ngoại được chiếu vào, khi có tín hiệu sóng hồng ngoại dao động với chu kỳ 36-38kHz nó sẽ xuất ra chân output điện áp cao (3.3V) và ngược lại nếu không có tín hiệu sóng hồng ngoại, nó sẽ xuất ra chân output điện áp thấp (0V).



Hình 8: Ghép nối TSOP1838 với ESP8266

Sử dụng ngắt trên chân GPIO5, ngắt được kích hoạt mỗi khi có sự thay đổi mức điện áp từ chân output của TSOP1838. Ghi lại các khoảng thời gian mỗi lần chuyển đổi mức điện áp sẽ thu được mã của tín hiệu hồng ngoại, tính theo đơn vị thời gian micro giây (µs). Tín hiệu được đánh dấu là bắt đầu khi chân output được kéo lên mức cao. Ví dụ với đoạn mã sau "5000, 4000, 400, 500, 540, 340, 600, 1700, 1900,…", tương ứng với điện áp trên chân GPIO5 bắt đầu ở mức cao trong 5000µs, mức thấp trong 4000µs, mức cao trong 400µs,…

```
void ICACHE_RAM_ATTR rxIR_Interrupt_Handler(){
      if(len > MAXLEN) return;
      irBuffer[len] = micros();
      if(len > 0)
            irBuffer[len-1] = irBuffer[len] - irBuffer[len-1];
      len++;
}
```

Giải thích:

- Hàm xử lý ngắt này được kích hoạt mỗi khi có sự chuyển đổi giá trị logic 0 và 1 ở chân GPIO5 (nối với chân output của TSOP1838).

- ICACHE_RAM_ATTR: bắt buộc khi dùng hàm xử lý ngắt với ESP8266, hàm có cờ này sẽ được lưu trong RAM để truy cập tốc độ cao
- MAXLEN: giá trị max cho độ dài mã hồng ngoại, MAXLEN = 600.
- irBuffer[]: mảng kiểu int chứa giá trị thời gian (µs) của các trạng thái 0 và 1
- micros(): hàm này trả về thời gian tính theo µs kể từ khi ESP8266 reset, lấy hiệu sẽ thu được khoảng thời gian duy trì trạng thái 0 và 1.

3.2. Phát tín hiệu hồng ngoại

Tín hiệu hồng ngoại được biểu diễn ở dạng chuỗi khoảng thời gian giữa 2 trạng thái cao thấp liên tiếp. Bắt đầu là khoảng thời gian ở mức cao, kế tiếp là mức thấp, cứ xen kẽ như vậy cho đến khi không có tín hiệu mức cao quá khoảng 200ms. Do vậy để phát lại tín hiệu, cần xuất ra đầu ra vào chân led hồng ngoại:

- Tín hiệu mức thấp: tắt led hồng ngoại trong khoảng thời gian đó.
- Tín hiệu mức cao: dao động bật tắt led hồng ngoại liên tục trong khoảng thời gian đó với tần số 36-38kHz. Tức là xuất ra xung PWM với độ rộng 50% và tần số 36-38kHz trong khoảng thời gian đó.

Đoạn mã triển khai theo ý tưởng trên để phát tín hiệu hồng ngoại:

```
analogWriteFreq(38000);
for (int i = 0; i < rawLen; i++){
    if(i & 1){
      digitalWrite(IRLEDPIN, LOW);
    } else{
      analogWrite(IRLEDPIN, 512);
    }
    delayMicroseconds(irBuffer[i]);
}
digitalWrite(IRLEDPIN, LOW);
```

Giải thích:

- analogWriteFreq(38000): thay đổi tần số PWM thành 38kHz.
- rawLen: độ dài đoạn mã hồng ngoại.
- IRLEDPIN = 13: đầu ra điều khiển led hồng ngoại.
- irBuffer[]: mảng chứa giá trị của mã hồng ngoại.
- Với các giá trị khoảng thời gian ở vị trí lẻ thì là tín hiệu mức thấp, đầu ra chân IRLEDPIN là mức thấp: digitalWrite(IRLEDPIN, LOW). Ở mức cao thì xuất ra xung PWM với độ rộng 50%: analogWrite(IRLEDPIN, 512).
- delayMicroseconds(irBuffer[i]): chờ khoảng thời gian duy trì trạng thái tương ứng.

Ngoài ra, để ổn định và chính xác hơn có thể sử dụng thư viện “IRsend.h” để phát tín hiệu hồng ngoại:

```
IRsend irsend(IRLEDPIN);
irsend.enableIROut(38);
for (int i = 0; i < rawLen; i++){
```

```
    if(i & 1){
      irsend.space(irBuffer[i]);
    } else{
      irsend.mark(irBuffer[i]);
    }
}
```

Giải thích các hàm thư viện:

- irsend(IRLEDPIN): khởi tạo đối tượng IRsend với chân led hồng ngoại là chân IRLEDPIN = 13.
- enableIROut(38): bật chế độ phát tín hiệu hồng ngoại với tần số 38kHz.
- space(irBuffer[i]): xuất ra chân IRLEDPIN tín hiệu mức thấp trong khoảng thời gian irBuffer[i].
- mark(irBuffer[i]): xuất ra chân IRLEDPIN tín hiệu mức cao trong khoảng thời gian irBuffer[i].

3.3. Nén dữ liệu tín hiệu hồng ngoại

Dữ liệu hồng ngoại thu được từ module thu hồng ngoại là một chuỗi các giá trị kiểu int, thể hiện cho khoảng thời gian ở mức cao/ thấp của led phát hồng ngoại theo đơn vị micro giây. Ví dụ một chuỗi dữ liệu 71 giá trị thu được có dạng như sau:

"9008 4480 581 573 582 573 582 572 581 573 582 573 581 573 581 573 581 573 581 1673 556 1674 556 1673 556 1674 556 1674 556 1674 556 1673 556 1672 558 573 581 1674 556 1673 556 574 581 1673 557 572 582 573 581 573 582 1673 557 572 581 574 581 1673 557 573 581 1674 557 1673 556 1673 557 39342 9011 2250 557"

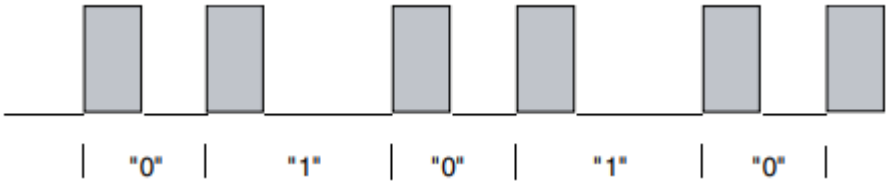
Nếu lưu ở dạng chuỗi rồi gửi lên Firebase thì trung bình mỗi giá trị cần khoảng 4 bytes. Nếu lưu dạng mảng cho mỗi giá trị, phải lưu bằng kiểu int mất 2 bytes/ 1 giá trị. Mà thông thường đối với mã hồng ngoại của điều hòa, số lượng giá trị lên tới 400 - 500, đây là một chuỗi lớn, có thể gây ra lỗi khi cập nhật lên Firebase từ ESP8266. Do đó cần phải nén dữ liệu, loại bỏ các thông tin trùng lặp.

Đối với tín hiệu hồng ngoại, khoảng thời gian các trạng thái mức cao/ thấp thường là bội của 50, do đó ta có thể làm tròn các giá trị thành bội của 50 mà độ chính xác vẫn cao. Chuỗi dữ liệu ví dụ trên sau khi làm tròn thành bội của 50:

"9000 4500 600 550 600 550 600 550 600 550 600 550 600 550 600 550 600 550 600 550 600 1650 550 1650 550 1650 550 1650 550 1650 550 1650 550 1650 550 1650 550 1650 550 550 600 1650 550 1650 550 550 600 1650 550 550 600 550 600 550 600 1650 550 550 600 550 600 1650 550 550 600 1650 550 1650 550 1650 550 39350 9000 2250 550"

Mục đích của việc làm tròn là có thể giảm được dải biểu diễn của dữ liệu từ int sang byte, chuỗi sẽ giảm dung lượng đi còn 1 nửa. Chia toàn bộ giá trị cho 50 sẽ thu được một chuỗi mới có giá trị nhỏ hơn và đa số có thể biểu diễn bằng 1 byte:
"180 90 12 11 12 11 12 11 12 11 12 11 12 11 12 11 12 11 12 33 11 33 11 33 11 33 11 33 11 33 11 33 11 33 11 11 12 33 11 33 11 11 12 33 11 11 12 11 12 11 12 33 11 11 12 11 12 33 11 11 12 33 11 33 11 33 11 787 180 45 11"

Tiếp đến, đối với dữ liệu của tín hiệu hồng ngoại, thông thường sẽ theo chuẩn mã hóa khoảng cách xung (pulse distance). Cụ thể như hình bên dưới:



Hình 9: Mã hóa khoảng cách xung

Các khoảng thời gian tín hiệu ở mức cao là như nhau, do vậy có thể sử dụng kiểu nén độ dài chuỗi: các kí tự liền nhau, giá trị giống nhau sẽ được thay bằng giá trị và số lượng. Cụ thể ở đây sẽ là các giá trị xen kẽ nhau 1 vị trí mà giống nhau liên tiếp sẽ được thay bằng giá trị và số lượng:

- Duyệt các giá trị ở vị trí chẵn, nếu các giá trị liên tiếp giống nhau sẽ được thay bằng giá trị và số lượng, đồng thời chèn thêm 1 kí tự có mã 33 ở trước để báo hiệu.
- Tương tự với các kí tự ở vị trí lẻ.

Xét đoạn đầu của chuỗi ví dụ bên trên: "*180 90 12 11 12 11 12 11 12 11 12 11 12 11 12 11 12 11 12 33,…"*. Các giá trị có số lượng 1 như 180, 90 sẽ được giữ nguyên. Ở vị trí chẵn, ta thấy có 9 giá trị 12 liên tiếp, vậy 9 giá trị này sẽ được thay bằng 33, 9 và 12. Ở vị trí lẻ ta có 8 giá trị 11 liên tiếp, vậy 8 giá trị này sẽ được thay bằng 33, 8 và 11.
Tuy nhiên Firebase chỉ chấp nhận chuỗi là các giá trị ACSII in được, vậy nên phải chuyển các giá trị byte trên (ngoại trừ kí tự báo hiệu 33) thành kí tự ASCII in được, quy ước mã hóa đối với từng giá trị:

- 1 – 63: giá trị được cộng thêm 60 và ép kiểu thành dạng char (1 kí tự in được có mã trong khoảng từ 61 - 123)
- 64 – 4096: chuyển thành 2 kí tự in được. ví dụ 787 = 12 * 64 + 19, sẽ chuyển thành 2 giá trị 12 và 19, cộng thêm 60 để chuyển thành kí tự in được như trên. Chèn thêm 1 kí tự có mã 34 ở đầu để phân biệt với các byte lẻ. Tổng cộng ta cần 3 byte cho giá trị 787 là: 34, 72 và 79. Thông thường các giá trị sẽ tập trung ở khoảng 1 – 63, chỉ có một vài giá trị nằm trong khoảng này.

Định dạng chuỗi sau khi nén: hai kí tự đầu tiên là độ dài chuỗi dữ liệu phía sau (2bytes). Từ kí tự tiếp theo đến kí tự '*#*' là những giá trị ở vị trí chẵn, phần còn lại là ở vị trí lẻ.

Thuật toán sẽ hoạt động hiệu quả khi số lượng giá trị mang thông mã tín hiệu hồng ngoại nhiều và giá trị khoảng thời gian thu được càng chuẩn xác thì số lượng trùng lặp sẽ càng nhiều. Chuỗi ví dụ thu được sau toàn bộ quá trình nén, giảm từ 71 giá trị kiểu int (306 kí tự nếu biểu diễn theo chuỗi) xuống thành chuỗi 65 kí tự:
*=C">p!1H!0GH!*GHG!+HG!*HGH!+G">pG#"=V!0G!0]G!*]G]!+G]!*G]G!+]"Hoi*

3.4. Hẹn giờ phát tín hiệu hồng ngoại

3.4.1. Cập nhật thời gian thực trên ESP8266

Sử dụng thư viện "NTPClient.h" để lấy dữ liệu thời gian thực bằng kết nối UDP đến server "pool.ntp.org", thư viện sẽ tự động đồng bộ thời gian với server trong một khoảng thời gian được định trước. Trong khoảng thời gian đó, thư viện sẽ sử dụng timer trên Esp-8266 để đếm thời gian thông qua 2 hàm milis() và micros(), hai hàm này sẽ trả về khoảng thời gian trôi qua theo micro giây và mili giây kể từ khi chip reset.

Thư viện có các hàm/phương thức hữu ích như:

- Constructor(WiFiUDP wifiUdp, String serverName, long timeOffset, unsigned long updateInterval): khởi tạo đối tượng NTPClient gồm các thuộc tính:
    - WifiUDP: khởi tạo kết nối UDP đến server
    - serverName: địa chỉ server
    - timeOffset: múi giờ tính theo giây, ví dụ Việt Nam có múi giờ là UTC+7, nhanh 7 tiếng so với giờ GMT, timeOffset cộng thêm 7 giờ tương ứng là 25200 giây
    - updateInterval: khoảng thời gian tính theo giây sẽ thực hiện đồng bộ thời gian với server, đang được đặt là 1 giờ (3600 giây).
- update(): khi gọi phương thức này, đối tượng NTPClient sẽ kiểm tra đã vượt quá updateInterval hay chưa, nếu vượt quá sẽ tiến hành đồng bộ thời gian với server. Phương thức này sẽ được đặt trong thân hàm loop() của chương trình.
- getFormattedTime(): trả về thời gian theo định dạng "hh:mm:ss"

- getEpochTime(): trả về thời gian Epoch – là số giây trôi qua kể từ ngày 1/1/1970, phương thức này sẽ được sử dụng để so sánh độ lệch thời gian, từ đó tính ra số giây cần chờ để thực hiện hẹn giờ trên Esp-8266.

3.4.2. Hẹn giờ trên ESP8266

Sử dụng thư viện "Ticker.h" để hẹn giờ, Ticker được cài đặt dựa trên Software timer của ESP8266. Software timer hỗ trợ sẵn hẹn giờ, thực thi nhiệm vụ thông qua hàm callback. ESP8266 có 2 hardware timer là timer0 và timer1, nhưng chỉ có timer1 là có thể sử dụng vì timer0 được dùng cho Wifi. Ngược với hardware timer, từ hardware timer có thể khởi tạo được nhiều software timer, đồng nghĩa với việc ta có thể tạo nhiều đối tượng Ticker để hẹn giờ hoặc lập lịch cho nhiều nhiệm vụ được điều khiển bằng các software timer riêng biệt.

Các hàm/phương thức hữu ích của thư viện:

- attach_ms(uint32_t miliseconds, void* callback, void * arg): phương thức này để thiết lập hẹn giờ với khoảng thời gian miliseconds tính từ lúc bắt đầu gọi hàm, có lặp lại sau mỗi khoảng thời gian đó. Thực thi hàm callback với các tham số arg. Với kiểu uint32_t ta có thể hẹn giờ lên đến gần 50 ngày.
- once_ms(): tương tự attach_ms nhưng chỉ thực thi hàm callback 1 lần.
- detach(): hủy timer và callback
- Ngoài ra còn có các hàm nạp chồng lại hoặc có chức năng tương tự hai hàm attach_ms() và once_ms(), có thể nhận tham số là giây, hàm callback và tham số cho hàm callback được đóng gói thành một lớp,…

Ở chức năng hẹn giờ, thiết bị sẽ nhận dữ liệu từ người dùng trên app chứa thông tin về thời gian, chu kì lặp và mã hồng ngoại, chương trình sẽ thực hiện lên lịch để phát tín hiệu hồng ngoại khi đến đúng thời điểm đã đặt và sẽ phát lại theo chu kì lặp nếu có hoặc chỉ phát một lần.

Tạo một lớp chứa các thông tin để hẹn giờ phát tín hiệu hồng ngoại: khoảng thời gian chờ, mã hồng ngoại, chu kỳ lặp lại và một đối lượng Ticker để hẹn giờ, cùng với đó là phương thức phát tín hiệu hồng ngoại. Khởi tạo một mảng 10 đối tượng (cố định, có thể mở rộng) của lớp đó để có thể lên lịch cho 10 mã riêng biệt đồng thời. Mỗi khi có sự cập nhật từ Firebase thì chương trình sẽ đọc dữ liệu, kiểm tra

trường thời gian sau đó tính toán khoảng thời gian chờ (thời gian sẽ được lưu dạng Epoch, kiểu long), lưu lại trường mã hồng ngoại, trường lặp (là chu kỳ lặp lại trong khoảng từ 0-24 giờ, 0 là không lặp, chỉ thực hiện một lần). Và cuối cùng là lên lịch bằng ticker nếu tất cả dữ liệu là hợp lệ. Dưới đây là mã nguồn của lớp:

```
class SenderSchedule{
  public:
    SenderSchedule(){};
    ~SenderSchedule(){};
    int pos;
    unsigned int duration;
    String data;
    int loop;
    Ticker ticker;

    void set(){
      ticker.once(duration, std::bind(&SenderSchedule::doSend,
this));
    }

    void stop(){
      ticker.detach();
    }

    void doSend(){
      unsigned long _t = timeClient.getEpochTime() + loop * 3600;
      sendIR(this->data);
      if(loop != 0){
        Firebase.setFloat("device/sduration/s" + String(pos), _t);
      } else {
        Firebase.setFloat("device/sduration/s" + String(pos), 0);
      }
    };
} schedule[10];
```

Giải thích:

- Các thuộc tính:
  - int pos: xác định vị trí nằm trong mảng 10 phần tử.
  - unsigned int duration: khoảng thời gian đến thời điểm phát tín hiệu.
  - int loop: chu kỳ lặp tính theo giờ.
  - String data: mã tín hiệu hồng ngoại.
  - Ticker ticker: ticker để lên lịch hẹn giờ.
- Các phương thức:
  - set(): bắt đầu hẹn giờ, hẹn giờ thực hiện hàm doSend() một lần.
  - stop(): hủy hẹn giờ.
  - doSend(): được gọi khi đến thời điểm đã đặt. Sau khi thực hiện phát tín hiệu xong, kiểm tra biến lặp, nếu có lặp (khác 0) thì sẽ cập nhật lại giá trị thời điểm mới lên Firebase và chờ nhận dữ liệu từ Firebase để thiết lập lần lặp kế tiếp.

3.5. Kết nối ESP8266 với Firebase

Các trường dữ liệu trao đổi giữa ESP8266 và Firebase:

- **device** // Nhóm các keys điều khiển thiết bị
  - cmd: “OK” // Lệnh điều khiển và trạng thái thiết bị
  - ircode: “” // Mã hồng ngoại đọc được bởi thiết bị (string)
  - scode: “” // Mã hồng ngoại để thiết bị phát (string)
  - schedule // Danh sách các mã hồng ngoại để hẹn giờ
    - s0: “” // Giá trị các mã hồng ngoại
    - s1: “” // Hỗ trợ 10 mã hẹn giờ đồng thời
    - …
    - s9: “” // s0 – s9
  - sduration // Danh sách thời điểm hẹn giờ
    - s0: “” // s0 – s9
    - s1: “”
    - …
    - s9: “”
  - sloop // Danh sách chu kỳ lặp của tính năng hẹn giờ
    - s0: “” // s0 – s9
    - s1: “”
    - …
    - s9: “”
  - sname // Danh sách các tên nút được hẹn giờ
    - s0: “” // s0 – s9
    - s1: “”
    - …
    - s9: “”
  - sstatus // Trạng thái các timer hẹn giờ
    - s0: “” // s0 – s9
    - s1: “”
    - …
    - s9: “”

Sử dụng thư viện FirebaseArduino để kết nối với Firebase, các hàm sử dụng:

- Firebase.begin(String FIREBASE_HOST, String FIREBASE_AUTH): kết nối đến Firebase thông qua host và xác thực thông qua key, các thông số này được lấy từ bảng điều khiển project firebase. Host: *irremote-4f614.firebaseio.com*.

- Firebase.stream(String path): tạo một stream để bắt các event về sự thay đổi của dữ liệu theo đường dẫn path. Tham số path được truyền cụ thể là:

```
Firebase.stream("/device");
```

- Firebase.available(): trả về true nếu có một sự kiện thay đổi dữ liệu trên Firebase xảy ra. Hàm này sẽ được kiểm tra liên tục trong hàm loop() của chương trình để bắt các sự kiện thay đổi dữ liệu trên Firebase, khi nó trả về true sẽ tiến hành đọc dữ liệu, kiểm tra và thực hiện lệnh.
- Firebase.readEvent(): trả về đối tượng kiểu FirebaseObject chứa thông tin về kiểu event (ví dụ "put"), tên event (là đường dẫn trỏ đến trường vừa thay đổi dữ liệu) và dữ liệu được cập nhật trên Firebase. Dùng để xác nhận xem trường nào vừa được cập nhật.
- Firebase.setFloat(String path, float value), Firebase.setString(String path, String value): cập nhật giá trị trường đường dẫn path trên Firebase bằng value.
- Firebase.getFloat(String path), Firebase.getString(String path), Firebase.getInt(String path): lấy giá trị của dữ liệu trong đường dẫn tương ứng trên Firebase.
- Firebase.failed(): trả về true nếu có lỗi xảy ra trong câu lệnh trước đó.

3.6. Cấu hình Wifi cho ESP8266 bằng Webserver

Mỗi khi thiết bị khởi động lại sẽ đọc giá trị byte đầu tiên trong EEPROM, byte này sẽ cho biết đã có thông tin về mạng Wifi hay chưa. Nếu có sẽ đọc tiếp các byte tiếp theo để lấy ra ssid và password sau đó kết nối đến mạng Wifi và hoạt động ở chế độ chính. Nếu chưa có thông tin về Wifi thì thiết bị sẽ chuyển sang chế độ thứ 2, sử dụng Webserver để lấy thông tin mạng wifi được nhập từ phía người dùng. Lúc này thiết bị sẽ đóng vai trò là một Access point có ssid là "ESP8266", khởi tạo và chạy một local web server hoạt động trong mạng LAN. Người dùng sẽ kết nối vào Access point đó và truy cập từ trình duyệt vào địa chỉ "192.168.1.1" sau đó nhập thông tin và submit, thiết bị sẽ lưu thông tin vào EEPROM.
Chế độ Access Point với địa chỉ IP tự thiết lập, đặt trong hàm setup():

```
WiFi.mode(WIFI_AP); // chế độ Access point
WiFi.softAP("ESP8266","12345678"); // ssid và password
// thiết lập ip
IPAddress ip(192,168,1,1);
IPAddress n_mask(255,255,255,0);
```

```
WiFi.softAPConfig(ip, ip, n_mask); // ip, defaul gateway, netmask
```

Khởi tạo và chạy web server, sử dụng thư viện "ESP8266WebServer.h":

- Phương thức on(String uri, function handler): đăng ký hàm callback mỗi khi có sự kiện từ client trong phạm vi đường dẫn uri.
- Phương thức begin(): khởi chạy đối tượng web server
- Phương thức handleClient(): đặt trong hàm loop để kiểm tra và kích hoạt hàm callback sự kiện tương ứng.
- Phương thức send(int code, String content_type, String content): gửi đến client các chuỗi, văn bản html.
- Phương thức hasArg(String arg) và arg(String arg): kiểm tra và lấy ra các tham số của HTTP request (GET, POST,…) từ phía client. Sử dụng để lấy ssid và password sau khi người dùng submit từ trình duyệt.

```
ESP8266WebServer server(80); // khởi tạo server trên port 80 (HTTP)
void handleWSRoot(){
      // kiểm tra request từ client
      // server.hasArg("arg"), server.arg("arg");
      ...
      // trả về các chuỗi html tương ứng...
      String html = "...";
      Server.send(200, "text/html", html);
}
void setup(){
      ...
      server.on("/", handleWSRoot);
      server.begin();
      ...
}
void loop(){
      ...
      server.handleClient();
      ...
}
```

Hình 10: Giao diện trang web

Lưu trữ, đọc và xóa thông tin trong EEPROM:

- Lưu thông tin wifi vào EEPROM khi nhận được HTTP POST request gồm 2 tham số "name" – ssid và "password" – password từ client. Lưu theo định dạng: "<1><ssid><0><password><0>", trong đó byte đầu tiên bằng 1 để báo hiệu đã có thông tin wifi.

```
EEPROM.begin(512);  // bắt đầu với 512bytes EEPROM
EEPROM.write(0, 1); // ghi tại địa chỉ 0, byte 1
delay(5);           // đợi 5ms sau mỗi lần ghi 1 byte
int e_in = 1;
for(int i = 0; i < wf_name.length(); i++){ // ghi ssid
  EEPROM.write(e_in++, (int)wf_name[i]);
  delay(5);
}
EEPROM.write(e_in++, 0);
delay(5);
for(int i = 0; i < wf_pass.length(); i++){ // ghi password
  EEPROM.write(e_in++, (int)wf_pass[i]);
  delay(5);
}
EEPROM.write(e_in++, 0);
delay(100);
EEPROM.end();
```

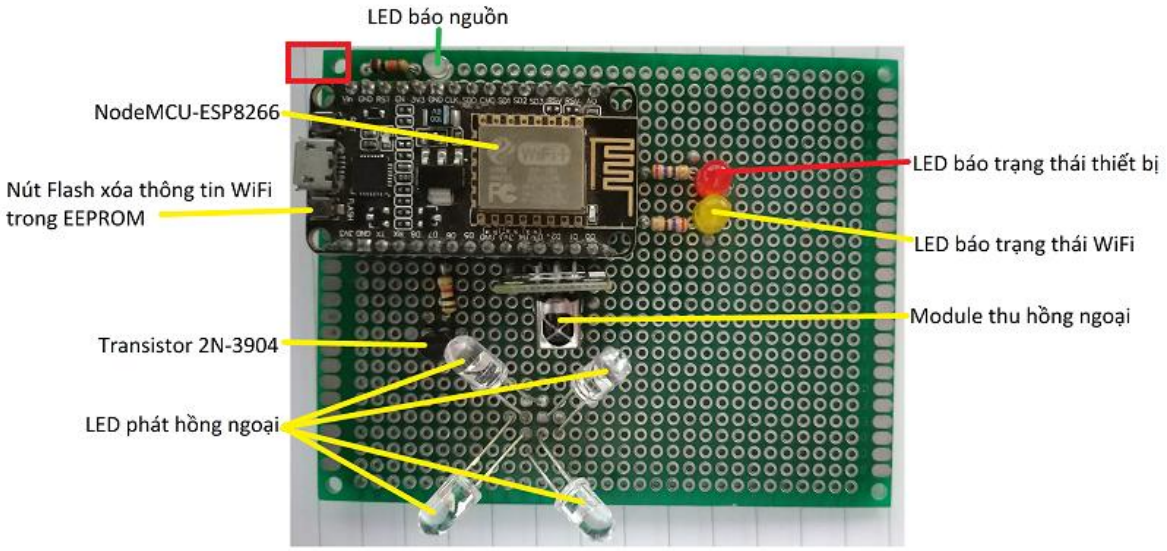
- Đọc thông tin từ EEPROM, kiểm tra byte đầu tiên nếu khác 0 thì đọc các byte tiếp theo để lấy ra ssid và password.

```
EEPROM.begin(512);
int w_a = EEPROM.read(0); // đọc byte địa chỉ 0 – byte đầu tiên
if(w_a){                  // nếu có thông tin (w_a != 0)
  int i = 1;
  int _b = -1;
  WIFI_SSID = "";
  WIFI_PASSWORD = "";
  while(_b != 0 && i < 512){   // đọc ssid
    _b = EEPROM.read(i++);
    if(_b != 0) WIFI_SSID += (char)_b;
  }
  _b = -1;
  while(_b != 0 && i < 512){   // đọc password
    _b = EEPROM.read(i++);
    if(_b != 0) WIFI_PASSWORD += (char)_b;
  }
  EEPROM.end();
}
```

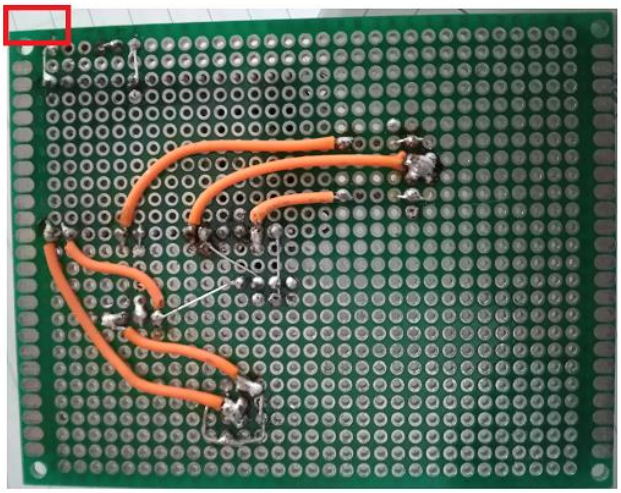
- Xóa thông tin trong EEPROM, mục đích là để cập nhật lại thông tin wifi khi cần thiết. Sử dụng nút Flash trên board, nút này được nối sẵn với chân GPIO-0. Khi được đặt chế độ chân 0 là INPUT_PULLUP thì chân số 0 này sẽ có giá trị mức cao và kéo xuống mức thấp khi nút Flash được nhấn. Bắt sự kiện nút nhấn và thực hiện đoạn mã để xóa byte đầu tiên trong EEPROM về 0.

```
EEPROM.begin(512);
int w_a = EEPROM.read(0); // kiểm tra giá trị byte đầu tiên
delay(200);
if(w_a){
  EEPROM.write(0, 0); // xóa byte đầu tiên về 0
  delay(5);
}
EEPROM.end();
```

3.7. Hàn các linh kiện của thiết bị vào bảng mạch



Hình 11: Mặt trước của bảng mạch sau khi hàn



Hình 12: Mặt sau của bảng mạch sau khi hàn

3.8. Giao diện và các tính năng trên ứng dụng Android

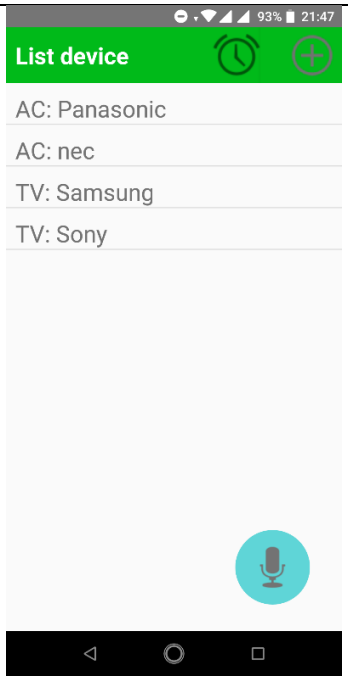
Hình đầu tiên là activity chính của app, listview chứa danh sách các thiết bị đã được lưu trữ trên Firebase, khi khởi động ứng dụng sẽ thực hiện lấy dữ liệu từ Firebase và hiển thị ra listview này. Quy ước:

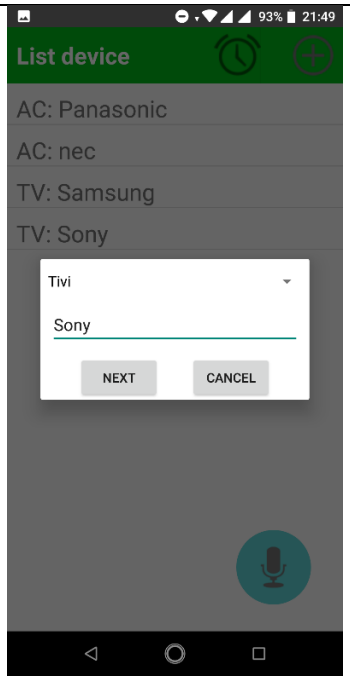
- AC, TV, FAN, OTH: loại thiết bị
- Tên thiết bị: do người dùng nhập

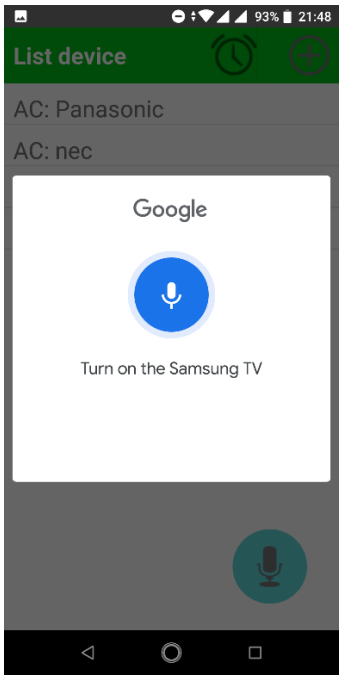
Bấm vào từng thiết bị sẽ chuyển sang activity thứ 2 để hiển thị danh sách các nút bấm đã lưu của thiết bị đó.

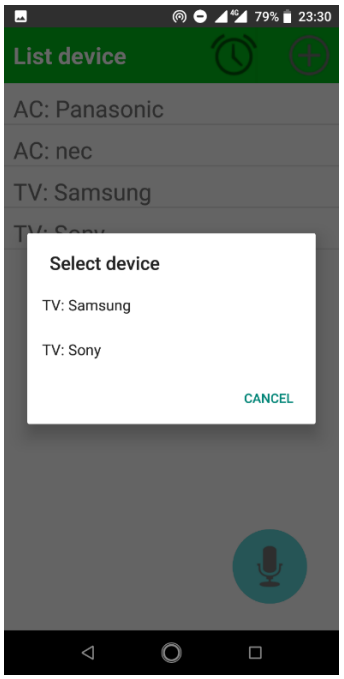
Để tạo thiết bị mới, chọn "Add', ứng dụng hiển thị dialog ở hình thứ 2, người dùng sẽ chọn loại thiết bị và điền tên cho thiết bị. Chọn "Next", ứng dụng sẽ tạo mới thiết bị và người dùng có thể thêm các nút bấm.

Chọn "Micro", ứng dụng sẽ hiển thị dialog như hình thứ 3, cho phép người dùng điều khiển bằng giọng nói. Sử dụng thư viện của Google để chuyển giọng nói thành text. Dùng Regex phân tích chuỗi để xác định các câu lệnh tương ứng. Một số câu lệnh điều khiển và hẹn giờ: "Turn (on/ off) the (tivi/ television/ air conditioner/ fan/ device) (name) (time)".
Nếu câu lệnh không chỉ rõ tên thiết bị ("Turn on the TV") và có nhiều thiết bị cùng loại thì sẽ hiện danh sách trên dialog cho người dùng chọn như hình 4.









Hình đầu tiên là activity hiển thị danh sách các nút của thiết bị AC: Panasonic (điều hòa) đã được đọc bởi người dùng.

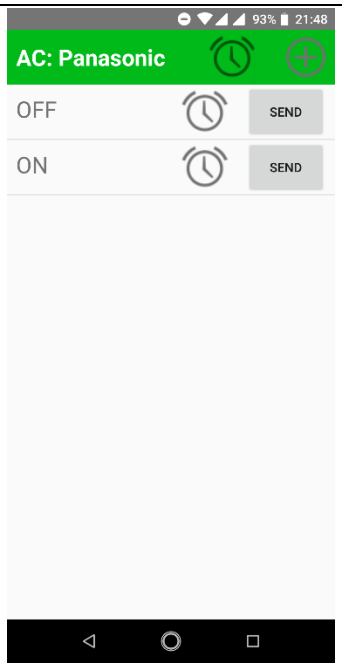
Bấm vào nút "Add" sẽ hiển thị dialog như hình thứ 2, cho phép người dùng đọc mã hồng ngoại của một nút mới, người dùng bấm Read, thiết bị điều khiển sẽ chuyển sang chế độ đọc, chờ đọc tín hiệu từ người dùng.

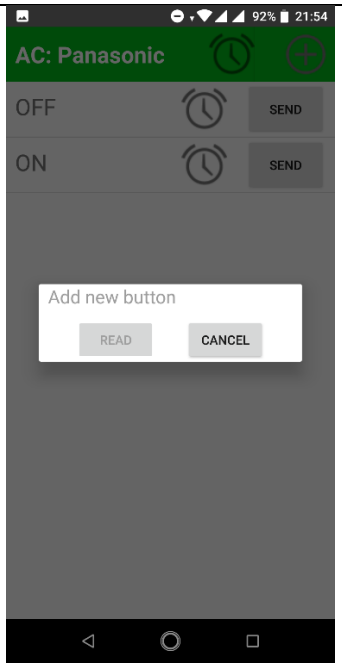
Sau khi đọc thành công sẽ hiển thị dialog như hình thứ 3, người dùng nhập tên và chọn "OK". Ứng dụng sẽ tạo nút mới với tên và mã đọc được và lưu lại trên Firebase.

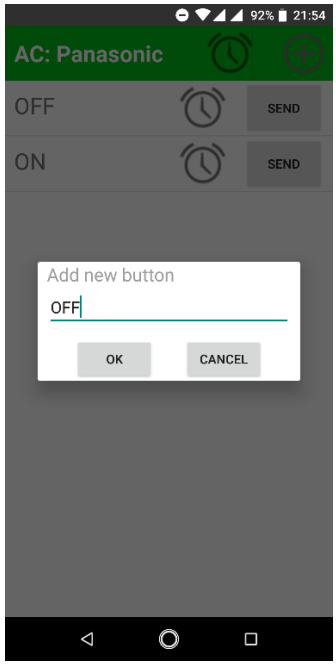
Người dùng cũng có thể xóa các nút bằng các nhấm và dữ vào tên nút tương ứng trên danh sách. Ứng dụng sẽ hiển thị dialog xác nhận xóa nút từ người dùng.

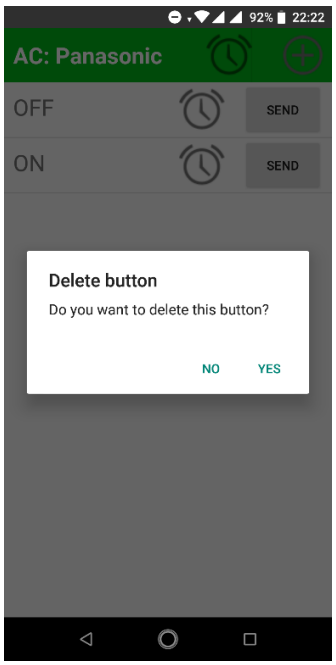
Chọn biểu tượng hình đồng hồ trên mỗi nút, ứng dụng sẽ hiển thị dialog để thiết lập hẹn giờ phát tín hiệu hồng ngoại từ các mã nút tương ứng.

Chọn biểu tượng đồng hồ trên thanh Action Bar, ứng dụng sẽ chuyển sang một activity khác, hiển thị danh sách các thiết lập hẹn giờ trên thiết bị điều khiển.





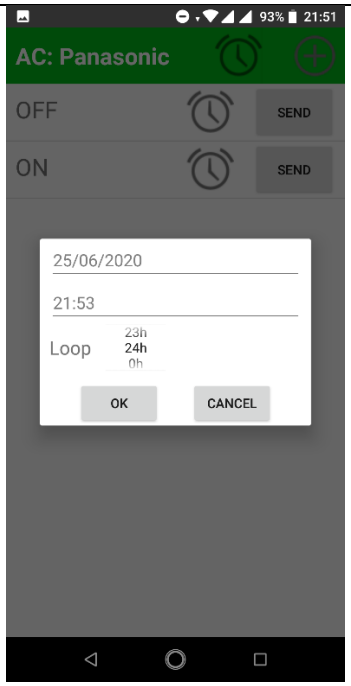


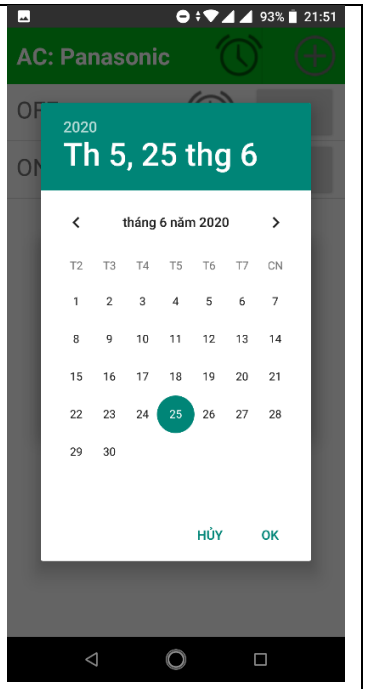


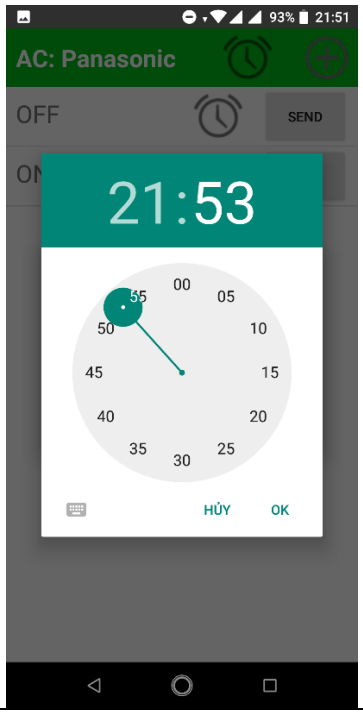
Khi người dùng chọn biểu tượng hình đồng hồ trên mỗi nút, ứng dụng sẽ hiển thị dialog để người dùng thiết lập các thông tin hẹn giờ như hình đầu tiên.

Giao diện chọn ngày và giờ sử dụng DatePickerDialog và TimePickerDialog như hình thứ 2 và thứ 3.

Cuối cùng chọn OK, ứng dụng sẽ kiểm tra nếu còn chỗ trống để thiết lập hẹn giờ thì ứng dụng sẽ cập nhật thông tin lên Firebase để thiết bị điều khiển thiết lập.







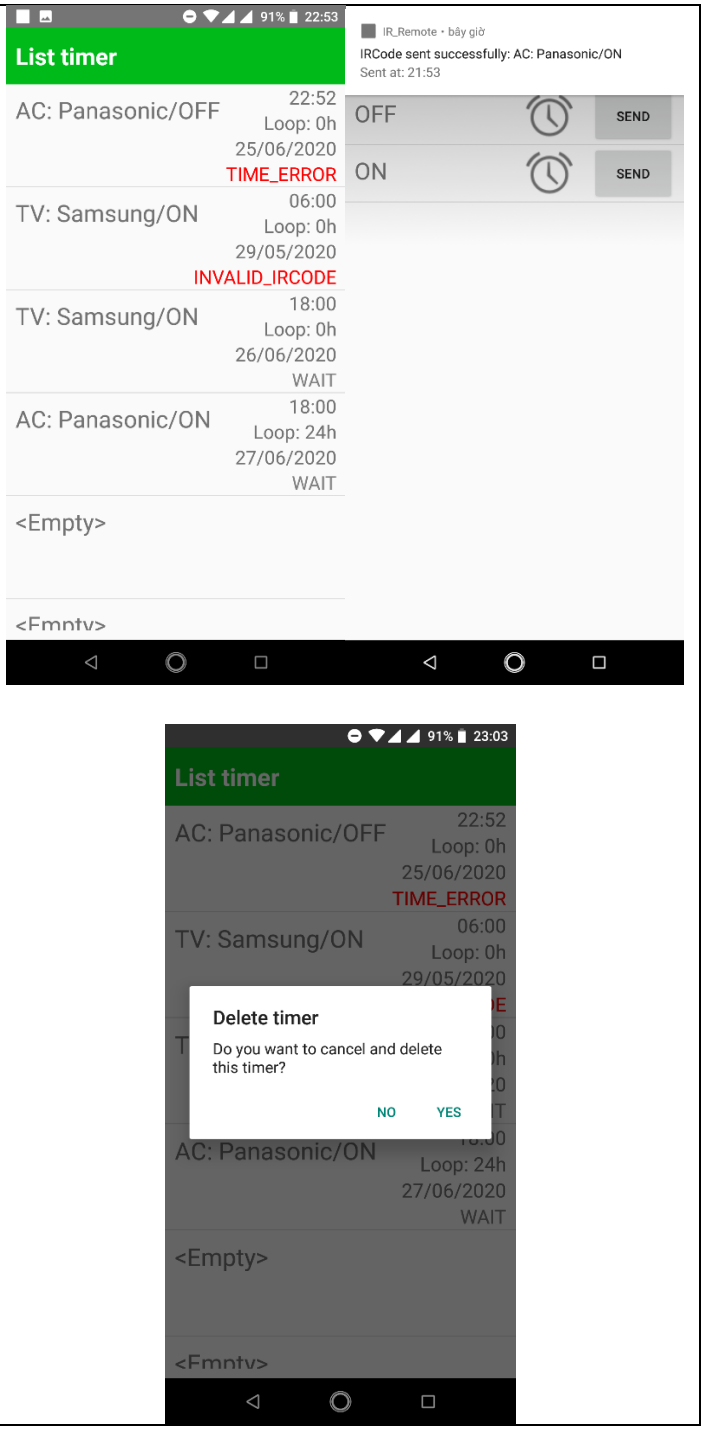
Hình đầu tiên là danh sách các thiết lập hẹn giờ, tên, thời gian và trạng thái:

- Các lỗi TIME_ERROR, INVALID_IRCODE.
- WAIT: đã thiết lập hẹn giờ thành công, chờ phát tín hiệu.
- SET: gửi yêu cầu thiết lập, chờ thiết bị điều khiển phản hồi

Hình thứ 2 là thông báo từ ứng dụng, ứng dụng đã phát tín hiệu được hẹn giờ thành công, đúng thời điểm.

Ngoài ra còn một số thông báo khác như: lỗi trong quá trình hẹn giờ, khi thiết bị điều khiển reset.

Người dùng có thể xóa các thiết lập hẹn giờ bằng cách nhấn vào tên trên danh sách và giữ. Ứng dụng sẽ hiển thị dialog cho người dùng xác nhận như hình thứ 3.



Hình 13: Giao diện và các tính năng trên ứng dụng Android

Bảng 3: Giao diện và các tính năng trên ứng dụng Android

PHẦN IV: KẾT LUẬN

Kết quả đã đạt được đối với thiết bị điều khiển đa năng sử dụng tín hiệu hồng ngoại và ứng dụng trên Android:

- Thiết bị có thể đọc, lưu trữ, nén dữ liệu và phát lại được toàn bộ tín hiệu điều khiển hồng ngoại mà module thu phát hồng ngoại đọc được.
- Thiết bị có thể thiết lập hẹn giờ 10 tín hiệu hồng ngoại cùng lúc (có thể mở rộng tùy thuộc vào bộ nhớ của ESP8266).
- Thiết bị có thể thiết lập lại hẹn giờ khi gặp sự cố phải reset bằng cách đọc từ Firebase và thiết lập.
- Thiết bị có thể được cấu hình thông tin WiFi mà nó kết nối đến để tăng tính linh hoạt khi sử dụng.
- Người dùng có thể sử dụng giọng nói để tương tác với thiết bị thông qua ứng dụng.
- Người dùng có thể lưu trữ các đoạn mã đã đọc và sử dụng để phát ở bất cứ đâu thông qua kết nối Internet.

Một số thử nghiệm thực tế và đánh giá:

- Khoảng cách điều khiển là khoảng 6-7m.
- Vấn đề cản sóng hồng ngoại: cũng như các remote hồng ngoại khác, nếu vật cản gần thì sóng hồng ngoại bị cản toàn bộ và không điều khiển được. Nếu vật cản ở xa, nằm giữa led hồng ngoại và thiết bị, kích thước vật cản nhỏ thì sóng có thể sẽ đến được thiết bị. Ví dụ tương đối: với vật cản rộng bằng tờ giấy A4, khoảng cách nhỏ hơn 2.5m là có thể cản được sóng.
- Thời gian trễ từ lúc thiết bị kết nối thành công đến WiFi đến lúc thiết bị sẵn sàng hoạt động là 5-10s, tùy thuộc vào tốc độ kết nối Internet của WiFi (thiết bị kết nối đến Firebase và lấy các thông tin hẹn giờ để thiết lập mỗi khi khởi động lại).
- Thời gian trễ từ khi người dùng chọn nút trên ứng dụng đến khi thiết bị phát tín hiệu hồng ngoại là 0.5-3s, tùy thuộc vào độ dài của mã hồng ngoại và tốc độ kết nối WiFi.

Những vấn đề còn thiết sót và hạn chế:

- Độ phủ của sóng hồng ngoại ra các hướng xung quanh chưa được tốt.
- Độ trễ còn cao dẫn đến khó có thể điều khiển nhanh các nút.
- Chưa tự động giải mã được toàn bộ các tín hiệu để lấy thông tin dữ liệu trong mã hồng ngoại, cụ thể như mã hồng ngoại của điều hòa.

Nếu như khắc phục được những hạn chế và thiếu sót trên và phát triển các tính năng mới như cho phép sử dụng nhiều người dùng khác nhau trên nhiều thiết bị điều khiển đa năng khác nhau, tự phát triển một Server riêng để trao đổi thông tin, phát triển thành sản phẩm riêng, thì đề tài này sẽ có tính ứng dụng rất cao, có thể tiếp cận và đáp ứng nhu cầu của người sử dụng.

Toàn bộ mã nguồn Arduino của ESP8266, mã nguồn Android của ứng dụng, các bản báo cáo hàng tuần, báo cáo đồ án và tài liệu tham khảo em lưu trữ trên github cá nhân, link github:
https://github.com/huydinh3010/DoAnHeNhung

TÀI LIỆU THAM KHẢO

**** Danh mục internet:***

1. Nguyên tắc điều khiển bằng tín hiệu hồng ngoại: https://www.laser.com/dhouston/ir-rf_fundamentals.html
2. Lập trình ESP8266 Arduino Core's: https://arduino-esp8266.readthedocs.io/en/latest/