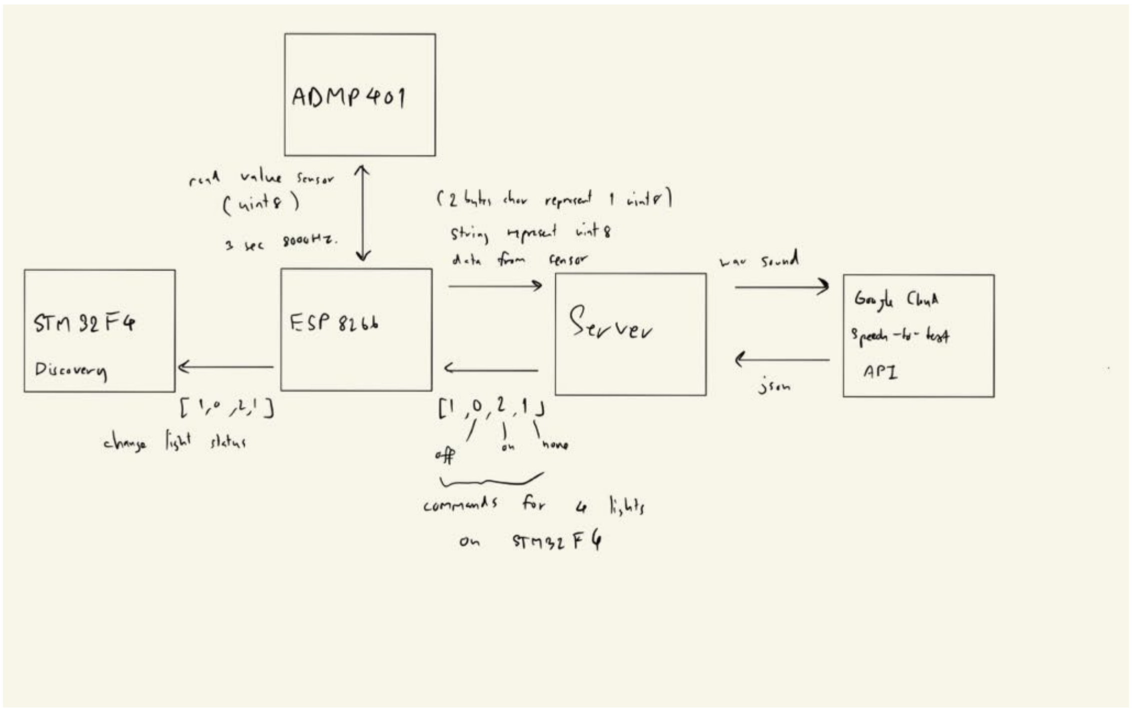
SmartHome101 (Smart Home from scratch)

1. System architecture (Draft)


ADMP401
read value sensor (uint8)
3 sec 8000Hz.
STM32F4 Discovery
ESP8266
Server
Google Cloud Speech-to-text API
(2 bytes char represent 1 uint8)
String represent uint8 data from sensor
wav sound
json
[1,0,2,1]
change light status
[1,0,2,1]
off
on
none
commands for 4 lights on STM32F4

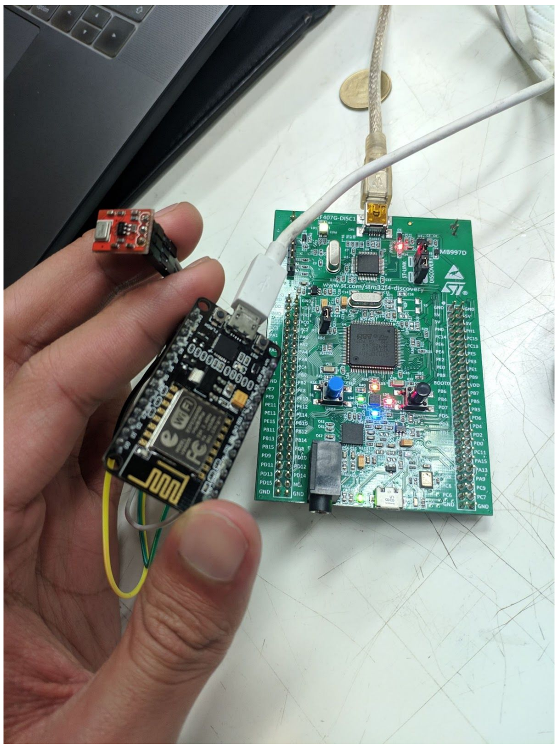
esp8266 record sounds from ADMP401 sensor, for 3 seconds with sampling rate of 8 khz, then those data are sent to server as string with 2 bytes represent 1 actual data from the sensor. After the server receive those data, the server will send these data to Google Cloud Speech-to-text API and get json as a result, which will then be turned into 4-digit String that represent turning on(2), turning off(0) and none(1) for 4 lights, then send to ESP8266, then ESP8266 send those data to STM32F4 so that STM32F4 act according to the command.

Protocol

between ESP8266 and STM32F4Discovery : small data(4 bytes String) is being sent between esp8266 and stm32f4 discovery, therefore using uart is enough, high speed is not necessary.

integration the parts

I connect the electric circuit between ESP8266, STM32F4Discovery and ADMP401 (sound sensor)



Circuit Connection

Between ADMP401 and ESP8266

VCC	---------------------------	3V3 3.3v
GND	---------------------------	GND GND
AUD	---------------------------	A0 ADC0

Between ESP8266 and STM32F4Discovery

D1 STx	---------------------------	PA3 USART2 RX
GND GND	---------------------------	GND GND

3. Embedded system development

- **ADMP401**

  ADMP401 signal output is a varying voltage, when there is no sound the AUD output float at half of the power supply voltage, and when there is sound, it will oscillate between 0V and power supply voltage, which will be read into value 0 to 1024, as it oscillates around 512.

- **ESP8266**

  There are 2 parts in ESP8266 which are getting data from the sensor and sending data to backend.

  Getting data from ADMP401 (sound sensor)

  Reading analog value from ADMP401, which results in 0-1023, then map them into 0-255 to keep in uint8_t, which these steps are repeated on the period that is set so that sampling rate is at 8 khz by using micros command to check if time has passed more than or equal to set period.

  Sending data to backend

  After getting the data, we divide it into 8 parts. For each part, we will encode them and send to backend by POST request. After 8 requests are done, it will send the data, received from backend, to STM32F4Discovery

Note: encoding/decoding

Due to the limitation of ESP8266 memory, we can't convert the whole data at the same time. And due to the limitation of ESP8266 http request, we can't send more than 6000 byte/request. So we decided to divide the data and encode it. There are 24,000 integers for 1 wave file. Each integer is between 0 and 255. So we encode the data by converting two first digits (0-25) into an alphabet (a-y) and remain the last digit.

For example, 6 will be encoded as 'a6'. 25 will be encoded as 'c5'. 240 will be encoded as 'y0'.

Speech Recognition part

- input : sound file (.wav file, 8 bit)
- algorithm : convert to .wav file 16 bit then send to Google Cloud Speech-to-Text API and convert the result to command
- output : the transcriptions of the input audio and the associated command code such as for the transcription "turn on the green one" will be converted to the command "HAL_GPIO_WRITE(GPIO_PIN_12HIGH)".

- **STM32F4Discovery**

input : 4 bytes string, with each byte represent command that lights will act according to data in those bytes. commands are mapped to STM32F4 on-board LED as shown below.

1.GPIO_PIN_12 LD4 green
2.GPIO_PIN_13 LD3 orange
3.GPIO_PIN_14 LD5 red
4.GPIO_PIN_15 LD6 blue

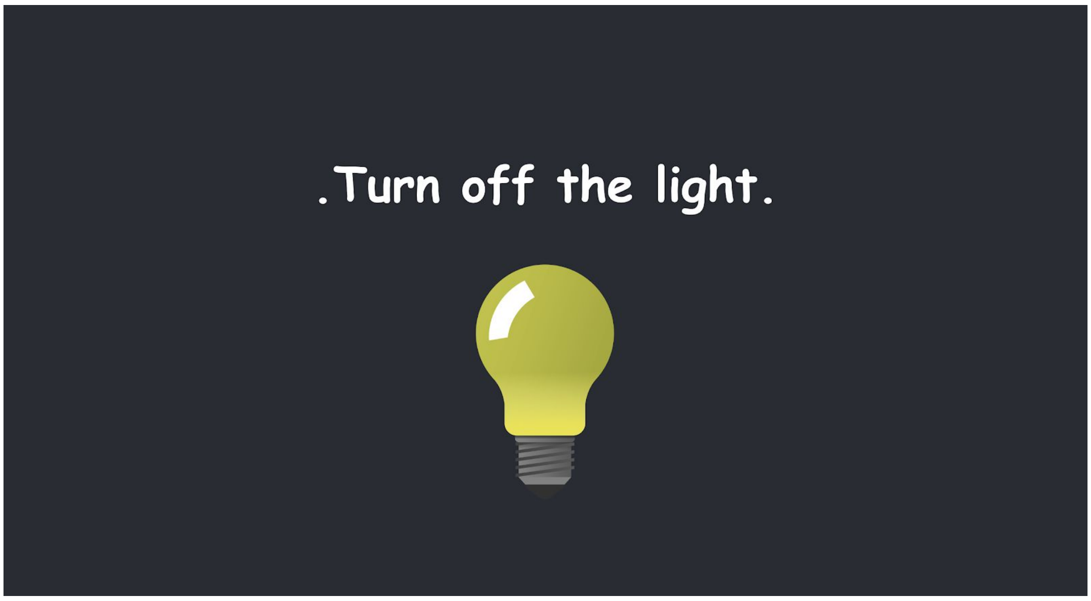
output :

0 : turn off
1 : do nothing
2 : turn on

4. UI designer and development

- **Website (frontend)**

In this project, I used React framework for building the frontend interface. Since the website is only used as a monitor, so it is a single page website. The whole screen is divided into 2 parts which are text part and picture part. The text part is containing the latest message that sensor detected. The picture part is containing 2 pictures of bulbs, which are bright bulb and dark bulb, shown only 1 picture at the time. The picture part show whether the light is on/off. The website use Axios library in order to fetch the data from backend. It will fetch every 3 seconds to refresh the data.



- **Backend**

In this project, I used Flask to build the backend because speech recognize function is written in Python. The backend is containing 2 endpoints which are GET request and POST request.

The GET request is used by frontend to fetch data. It returns data in JSON format containing 2 values which are the latest word and light status (on/off).

The POST request is used by ESP8266 to send the sound data received from sensor. For 1 records, it required to be called 8 times (8 POST requests). Each POST request must have a body in JSON format contain 1 value named “f”. The value of “f” must contain the string with maximum length is 6000. (Because ESP8266 can’t send the request bigger) For each request, we will decode the data and store it until the 8th request and return only the word “success” with the number of requests. For the 8th request, the program will call the speech recognize function with the stored data as an argument. The function will return the status of the light. Then, the program will return the status of the light in string format.