SMART HEALTHCARE DEVICE WITH IOT AND CLOUD COMPUTING

A Thesis

by

CHENJIE LUO

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Tie Liu
Committee Members,	Yang Shen
	Chao Tian
	Anxiao Jiang
Head of Department,	Miroslav M. Begovic

December 2019

Major Subject: Electrical Engineering

ABSTRACT

Internet of Things (IoT) is a system of interrelated digital devices such as home appliances which could connect, communicate and exchange data with each other. In the past ten years IoT has become an extensively studied topic since it significantly contributes to decision-making and brings convenience and efficiency into people's lives and homes. Meanwhile, as the growing attention to personal health, healthcare services have become more and more popular in public. As the size of healthcare device turns smaller, it has become possible for users to receive healthcare service at home. However, due to high sensitivity as well as complexity of healthcare devices, those devices are usually operated by well-trained medical personnel. Therefore, one of the biggest difficulties most users currently face is how to configure the complicated parameters for multiple purposes such as rehabilitation. In this project, we are trying to build up a system which could ease users' life by helping users configure the healthcare device with machine learning. Through this carefully designed system with System on a Chip (SoC) which is specially developed for home devices, we aim to not only provide sustainable remote control for users from an application on smart phones but also let users decide whether to use his or her own choice or the cloud suggestion based on both big data and his or her previous treatment history as the input. In the end, we want our system to realize three main functions: (1) Users will easily tune the healthcare device at home; (2) Data is able to be uploaded to the cloud; (3) Collected data will be able to be learned on the cloud in order to generate suggestions.

ACKNOWLEDGEMENTS

I would like to thank my committee chair, Dr. Liu, and my committee members, Dr. Shen, Dr. Tian and Dr. Jiang as well as Dr. Qian, for their guidance and support throughout the course of this research.

I would also like to thank faculty and staff from Department of Electrical and Computer Engineering as well as my colleagues for making my time at Texas A&M University a great experience.

Finally, I would like to thank my parents for their support and love.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supervised by a thesis committee consisting of Professor Tie Liu, Yang Shen and Chao Tian of the Department of Electrical and Computer Engineering and Professor Anxiao Jiang of the Department of Computer Science and Computer Engineering.

All work conducted for the thesis was completed by the student independently.

Funding Sources

This work is not supported by any funding.

NOMENCLATURE

SoC	System on Chip
GCP	Google Cloud Platform
AWS	Amazon Web Service
AVS	Amazon Voice Service
SQL	Structured Query Language
NoSQL	Not Only Structured Query Language
URL	Uniform Resource Locator
LED	Light-emitting Diode

TABLE OF CONTENTS

LIST OF FIGURES

Page

1. INTRODUCTION

Healthcare service has been a big issue around the world for a long time. Limited resources, qualified facilities as well as huge expenses usually hinder patient care and significantly decrease efficiency. It is generally accepted by American that healthcare service in the United States is expensive and could be unaffordable for some families. Meanwhile, patients from rural areas sometimes have no choice but to travel to cities in order to receive appropriate healthcare treatment. Consequently, people started thinking about deploying healthcare devices at home so that people could receive healthcare service in time and conveniently. However, due to complexity of healthcare equipment, it is generally not only unsafe but also impossible to let users manipulate the device themselves without clear instructions and adequate training. At the same time with the help of booming technologies such as 5G network, larger bandwidth has become possible and hence a larger quantity of data is able to be collected all over the world every single second. It turns out clearly that the data could be utilized and studied for our better decision making.

In the Internet of Things field, several giant technology enterprises like Amazon and Google have all established their own ecosystems. Among all the products Amazon Alexa ecosystem is one of the most famous representatives. Amazon Alexa ecosystem is composed of several components including Alexa-enabled devices such as Amazon Echo or Amazon Echo Dot, Alexa cloud deployed on Amazon Web Service as well as client

endpoints like amazon Alexa application on Android, iOS or Fire OS platform. Every Alexa-enabled device is able to be managed by clients through Alexa cloud. Besides, every Alexa-enabled device is connected to Amazon Voice Service (AVS), which is a smart cloud-based voice recognition service. Their most common functionalities include vocal alarm clock setting, voice reminder, touch-free music playing and vocal dialing through Alexa-enabled devices such as Amazon Echo Dot. Apart from these features, Alexa ecosystem is also compatible with third-party applications. For example, a food delivery application may be able to place order by Alexa-enable devices with the help of AVS. However, most of applications right now in IoT field concentrate on smart furniture or entertainment equipment instead of healthcare service.

Therefore, we propose our solution to this problem: We would like to build up a system which could help users easily configure the healthcare device and potentially provide the optimal suggestion for users. With the help of the SoCs which are specially developed for IoT device, we would like to not only provide a sustainable remote tuning system for users from the application on smart phones but also give users freedom to determine whether the cloud suggestion based on a majority of users' choices or his or her previous treatment history as the input. With the collected data, we will further develop statistical learning methods to provide suggestions for different categories of users. Those suggestions will be able to be downloaded to the smart phone application and will be implemented directly without users' interactions. In the end, users have three modes to configure the healthcare device.

2. ARCHITECTURE

Our proposed system consists of three sections: SoC inside the hardware endpoint, iOS application on the smartphone and back-end server deployed on the cloud. The whole system is depicted in Figure 1.

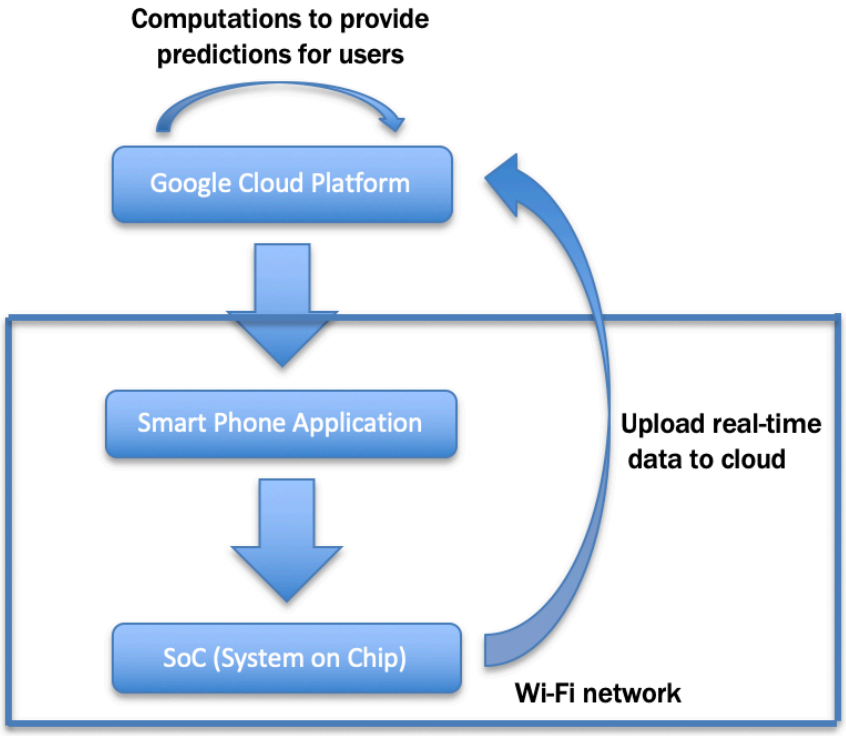


Figure 1. Smart Healthcare System Layout

Users could authorize our system to upload data for analysis or decline the service. They are still able to configure the system manually even if they decline to provide

authorization on data analysis. With users' authorization, data packets are generated the same time as the healthcare device responds to the command. Server on the cloud will listen to the events published from every device. When data packets arrive, our server will accept all the packets and route them into the database. Computations and learning are implemented on the cloud server as well. Output will then be generated and pushed to the smart phone application as suggestions for users. Both our SoC in the hardware and Smart Phone will be covered under Wi-Fi network.

3. IMPLEMENTATION DETAILS

3.1. Hardware

Hardware of our design comprises a LED illumination source along with an embedded chip. When choosing embedded chips, I tried multiple types of chips which are: Gizwits Go Kit, NodeMCU and Particle Photon. All these three chips were able to connect to Internet through Wi-Fi connection. Gizwits Go Kit and NodeMCU both contain a Wi-Fi module named ESP8266 while Particle Photon contains another Wi-Fi module named BCM43362. In compare with another two chips, Gizwits Go Kit is much larger and more complicated than the other two chips in terms of physical size and hence it is less convenient to embed the chip into our LED illumination source. Moreover, due to complicated functionalities and built-in sensors, Gizwits Go Kit is less efficient than the other two chips in power consumption. NodeMCU and Particle Photon turn out to work appropriately and efficiently but our finalized design decided to use Particle Photon as the embedded SoC chip inside the healthcare device since Particle Photon provided more comprehensive documentation. Particle Photon is empowered by a STM32 ARM Cortex M3 microcontroller along with a Cypress Wi-Fi chip named BCM43362. It consists of 18 mixed-signal GPIO and advanced peripherals while it can be programmed in a very similar format as Arduino. Particle Photon can be coded in C and C++ because it inherits all C and C++ structures.

The LED illumination source is powered by 110 VAC voltage while the embedded chip is powered by 3.3 VDC. For the connections between the embedded chip and LED

illumination source, five analog outputs on the chip are connected to the five LED options which differ in wavelengths. The wavelengths ranges are 630nm, 660nm, 830nm, 850nm and 940nm. Apart from wavelength, there is another channel of LED which also have five options: GW, WW, Red, Green and Blue. Besides, there are five illumination rate options in both channels: 0 Hz, 1 Hz, 10Hz, 20Hz, 40 Hz. When the LED illumination source works, users could choose to let it work in either option or in combination. The five options in each channel as well as illumination rate for LED illumination are adjustable simultaneously.

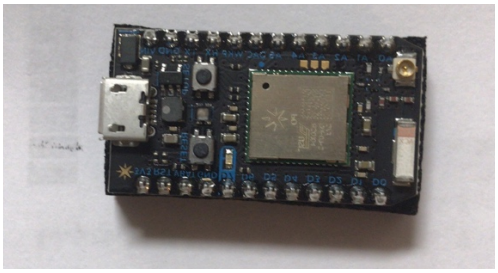


Figure 2. Particle Photon

3.2. Firmware

Since our design is to make embedded chips connect to Internet and communicate with cloud server directly, firmware on the chip should not only handle incoming requests from iOS application but also upload data entity upon authorization to the cloud.

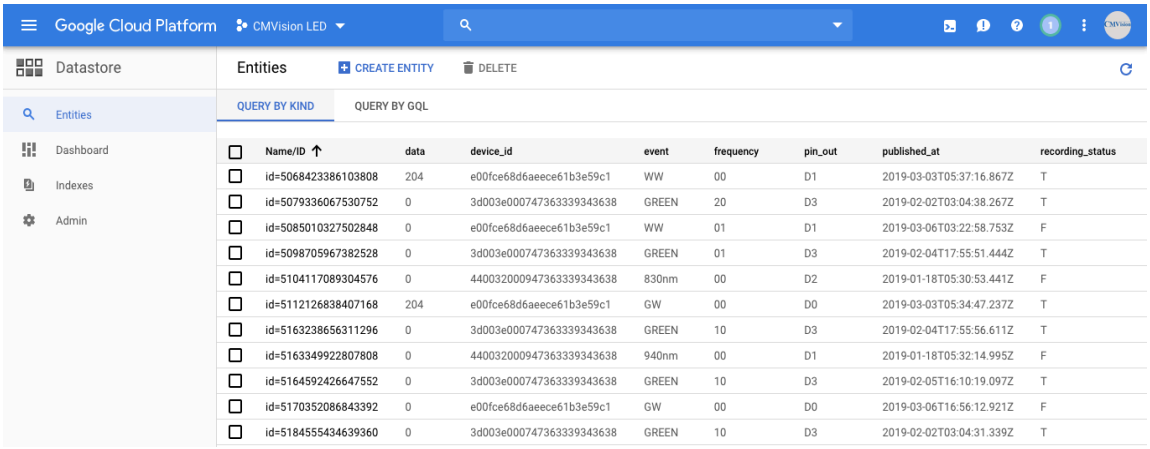
Our firmware consists of firmware of the application and firmware of the SoC chip. For the firmware of the application, Particle Device API is included in our project

so as to communicate with our embedded chip smoothly. Once the firmware is flashed into the chip, it calls a setup function to initialize global variables, declare API enabled functions and declare the working mode of each pin. Afterwards, an infinite loop is called to listen to incoming requests as well as respond to the request continuously. When a new request is received, API enabled function named ledToggle will be called to interpret the request to certain command and a function named Response is called to operate according to the interpreted command. Every Particle Photon has a unique device_id and Particle Device API provides a framework for developers to access the SoC chip using device_id and access_token. Access_token is distributed through Particle Device API as well. AFNetworking framework is used here to properly encode the POST and GET requests. AFNetworking is a widely used networking library for MacOS and iOS platforms and it is built on Foundation URL Loading System. When the chip receives the request, it immediately decodes the request into appropriate commands and start responding accordingly. In order to achieve this function, I wrote our firmware and flashed it into the chip. When the power is on, the chip keeps listening requests from iOS application. Given by the fact that we initially want the healthcare device connected to the Internet, data is uploaded through chip in JSON object directly and we also add timestamps to each JSON object for further analysis.

As for the backend, a NoSQL database was set up for users to record his or her operations when using the app. As is known to all, SQL database, or equivalently called relational database, is composed of a set of related database tables. The relationship between tables is established with the help of the primary key. In every database table, it

is supposed to have a unique primary key for each record. This primary key is also included in other tables as foreign keys. This primary key to foreign key mapping ensures the relationship between each database table. However, it is not difficult to realize that the biggest advantage for SQL database over NoSQL database is the closed relationship between each table. This feature comes with a price which is new columns will be difficult to insert when the database needs to be expanded horizontally yet. The reason lies in when adding new columns, equivalently we are adding new tables for the database. And when a new database table is added, we will need to maintain and update foreign keys in every other database table. This operation will be both complicated and time consuming. NoSQL database, which is also called non-relational database, is more powerful than traditional SQL database in terms of horizontal scaling. Traditional SQL database is easier to be scaled vertically, which means engineers try to increase hardware computation power when the load of the server increases. On the other hand, horizontal scalability means engineers could deploy more servers horizontally instead of keeping pushing the border of hardware. Since our system will expand by including more features, scalability is a vital component which we need to take into consideration. However, as its name implies, since each entity in NoSQL doesn't have strong correlation with other entities like relational databases, queries in a non-relational database is more expensive within sight of time complexity. Since our analysis won't query frequently but a stupendous quantity of entities each time, it is still reasonable to choose non-relational databases other than relational databases. In our design every piece of data uploaded from hardware device is saved as an entity in our non-relational database.

Besides, I wrote the server in Node.js so as to receive data entity and route it into our backend database. Node.js is an open-source, cross-platform, JavaScript run-time environment that executes JavaScript code outside of a browser. It was deployed on the Google Cloud Compute Engine and should keep running all the time. In compare with HTML and CSS, it helps web pages become interactive and could be embedded on the server.



Name/ID	data	device_id	event	frequency	pin_out	published_at	recording_status
id=5068423386103808	204	e00fce68d6aeece61b3e59c1	WW	00	D1	2019-03-03T05:37:16.867Z	T
id=5079336067530752	0	3d003e000747363339343638	GREEN	20	D3	2019-02-02T03:04:38.267Z	T
id=5085010327502848	0	e00fce68d6aeece61b3e59c1	WW	01	D1	2019-03-06T03:22:58.753Z	F
id=5098705967382528	0	3d003e000747363339343638	GREEN	01	D3	2019-02-04T17:55:51.444Z	T
id=5104117089304576	0	4400320009473633393436 38	830nm	00	D2	2019-01-18T05:30:53.441Z	F
id=5112126838407168	204	e00fce68d6aeece61b3e59c1	GW	00	D0	2019-03-03T05:34:47.237Z	T
id=5163238656311296	0	3d003e000747363339343638	GREEN	10	D3	2019-02-04T17:55:56.611Z	T
id=5163349922807808	0	4400320009473633393436 38	940nm	00	D1	2019-01-18T05:32:14.995Z	F
id=5164592426647552	0	3d003e000747363339343638	GREEN	10	D3	2019-02-05T16:10:19.097Z	T
id=5170352086843392	0	e00fce68d6aeece61b3e59c1	GW	00	D0	2019-03-06T16:56:12.921Z	F
id=5184555434639360	0	3d003e000747363339343638	GREEN	10	D3	2019-02-02T03:04:31.339Z	T

Figure 3. Back-end Database

3.3 Software

Our software consists of application on the iOS platform and deployment of the cloud server. For the iOS application, a simple but effective user interface is created which could dynamically adjust the illumination intensity of each option. The layout of the application is in Figure 4 and 5. The application is composed of two main pages. The first page is used for users to log in or sign up their new accounts. Only logged-in users are

authorized to control the device remotely. Google Firebase was applied to store users' login credentials including usernames as well as passwords. The second page is the control panel for hardware device. The functionalities include device_id acquisition, control mode selection and control console. The application is supposed to firstly connect to the chip and obtain device_id from devices directly. With device_id as well as Particle Device API, remote control could be resolved with the help of Uniform Resource Locator, which is abbreviated as url (https://api.particle.io/v1/devices/device_id/led?access_token=token). Data packets will only be uploaded when the start treatment switch is turned on. When it is off users will still be able to adjust the intensity of five options as well as other functionalities and the data will not be uploaded.

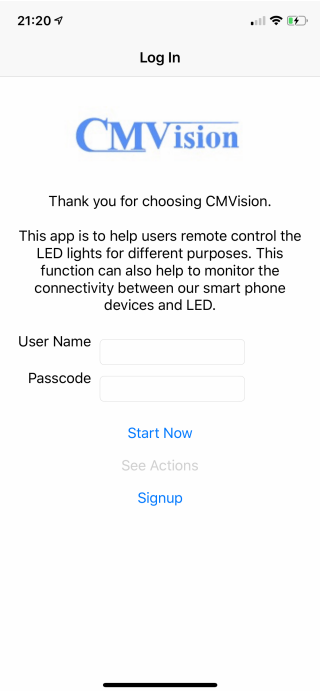


Figure 4. Layout of Login Page

For the data packet, I designed a protocol to encode and decode the request. The data packet includes Channel_Label, Option_Label, Light_intensity, Recording_Now. I designed to use 1 byte to represent the Channel_Label, 4 bytes to represent Light_Intensity and 1 byte to check whether it is recording now or not. Recoding_Now byte is used to track users' authorization of uploading data from cloud or not. If users turn on the start treatment switch, Recoding_Now is set to 'T' otherwise 'F' and the data will be shared to the cloud server and save into the database. The label is also used to check the starting point and ending point of a treatment for future analysis.

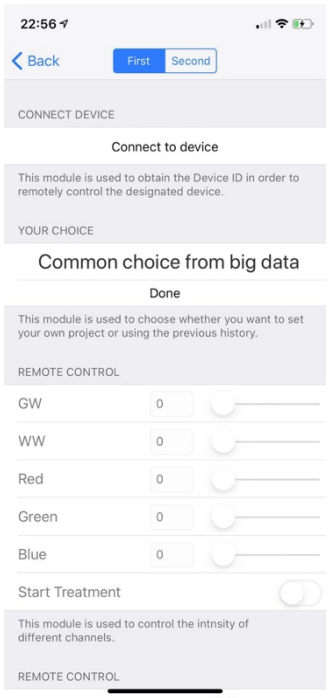


Figure 5. Layout of Control Panel for Suggestion Mode

For the cloud server, we wanted our server able to collect packets all the time once our server was on. However, there are two most common scenarios in practice we need to resolve which are congestion and server breaking down. In the case of our server not working correctly, uploaded packets should not get lost and should still be able to be recovered after the server is rebooted. My solution to this problem was adding a "buffer" as an intermediate layer between hardware endpoints and our backend database in order to temporarily store uploaded packets. When implementing this ideology, I used Google Pub/Sub service as the intermediate layer. Google Pub/Sub service is an asynchronized messaging service which provides highly reliable and low-latency communications. When a packet is uploaded to the cloud from our hardware device, it is firstly published on a topic in Google Pub/Sub service and the packet is stored in one of Google data centers. Generally, the data center closest to the publisher will be chosen. However, due to load balancing algorithms from Google, a single topic could potentially have packets in more than one data center. When a subscriber requests packets under this topic, the subscriber is connected to the closest data center and all data under this topic in different data centers will be aggregated together for the subscriber. Each topic should have at least one subscription and the packet will be delivered to every subscriber once. In our project, I made our scripts as one of subscribers to each topic. All the scripts are running on our cloud server. When the subscribers captured the packet, it should route the packet to our database. If subscribers fail to capture the packet, the packet will be stored on the Pub/Sub server up to 7 days. This period of time should be able to handle most of cases when server

is down accidently. Similarly, when multiple packets are sent to our server simultaneously, all packets are published to the topic first and they are queued in a “First Come First Serve” order and saved in Pub/Sub service before our server collects them.

Besides, in the case of Wi-Fi network does not function properly, the system is supposed to be able to track if a packet has been sent to the cloud successfully. In order to achieve this feature, I wrote a function on the server side named publishACK so as to send acknowledgement from the server indicating the packet had been received successfully. The communication is also through Pub/Sub service. Another topic named “ack” was created on the Pub/Sub service so that PublishACK function will be able to publish an acknowledgement under this topic. PublicACK function will only be called when the packet is saved to the database successfully as well as recording status is flipped. The smart phone application right now worked as a subscriber and was able to capture the acknowledgement once the publishing process is completed.

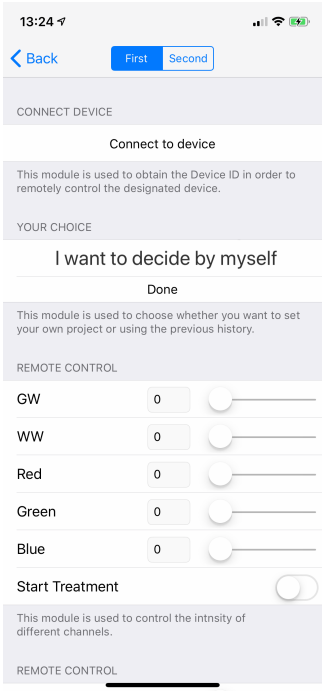


Figure 6. Layout of Control Panel for Manual Mode

4. CONCLUSIONS

In this project the research is mainly concentrated on the implementation of the complete IoT system. Our finalized system is depicted as follows in Figure 7. The final result reveals that hardware devices are able to work adequately and send data to cloud simultaneously.

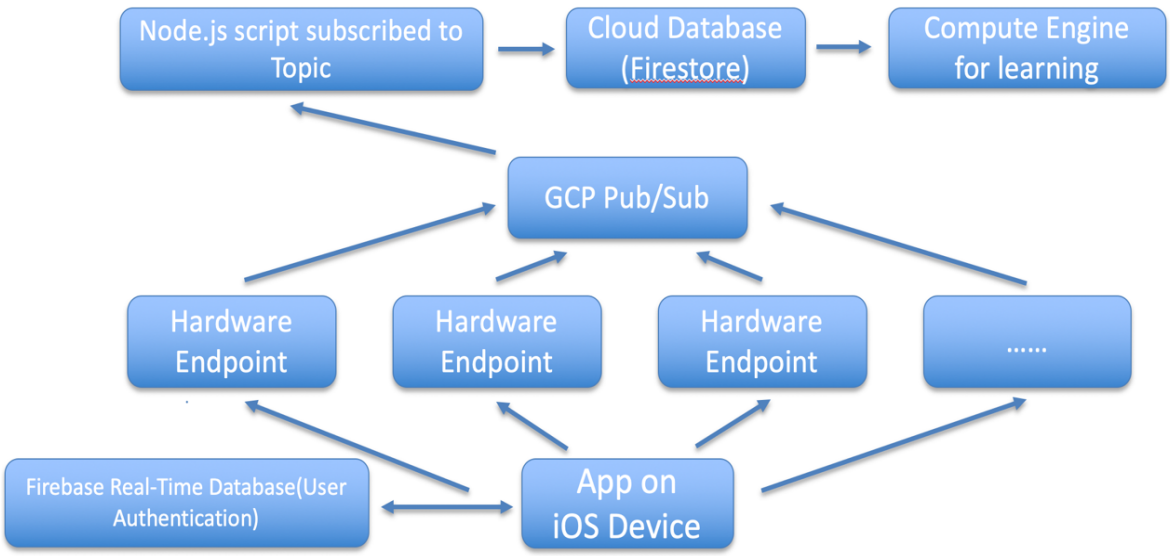


Figure 7. Final System Layout

4.1. Output for hardware

Our test reveals all our hardware components work correctly. When the power is on, it can be set to connect to a new Wi-Fi network with the help of the iOS application.

When controlling with application, the LED light source can respond adequately and data is sent to cloud server correctly as well.

4.2. Output for firmware

Our test indicates all the transmission firmware between hardware, iOS application and cloud server works correctly. When the packet arrives at SoC chip, it can correctly decode it into command. Data can also be uploaded to cloud server correctly.

4.3. Output for software

The test reveals users could log in and sign up through the application. User interface is easy for users to understand and users can operate the LED source after logging in successfully.

5. SUMMARY AND FUTURE WORK

Since our finalized system has realized the functionalities we desired, we are able to start collecting data for learning. When our database grows large enough, we would be able to take advantage of collected data to generate suggestions for users. Since our suggestions will be harnessed by users in the field of healthcare, it is necessary to guarantee no side effect will be triggered even if inaccurate suggestions are pushed to users. Therefore, fuzzy C-means algorithm could be used to classify each data point to clusters. And the centroid of the cluster where user's data point belong to should be generated as suggestions for users. In compare with K-means algorithm, fuzzy C-means attempted to minimize similar objective function but by the addition of fuzzier term m and weight w_{ij}. For the purpose of safety, the number of clusters and initial centroids of clusters should be provided by professional healthcare physicians. The whole algorithm is supposed to be designed as follows.

Step 1: Professional healthcare physicians decide the number of treatment options, which is equivalently the number of clusters N in our algorithm, and initial centroids for each cluster $C = \{c_1, c_2, c_3, \ldots, c_N\}$. At the same time, we randomly query total number of M data points from back-end database for learning.

Step 2: We define a fuzzy term m with a partition matrix W. $W = \{\boldsymbol{w_{11}}, \boldsymbol{w_{12}}, \ldots, \boldsymbol{w_{MN}}\}$ where $i \in [1, M]$, $j \in [1, N]$ and $w_{ij} \in [0,1]$. However, we can soften w_{ij} by importing another parameter b. Parameter m is used to control fuzzy degree for cluster boundaries. Our objective is to minimize the following function:

$$\underset{C}{\boldsymbol{argmin}} \sum_{i=1}^{M} \sum_{j=1}^{N} \boldsymbol{w_{ij}^{k}} ||\boldsymbol{x_i} - \boldsymbol{c_j}||_2$$

where

$$\boldsymbol{w_{ij}} = \frac{\mathbf{1}}{\sum_{k=1}^{c} \left(\frac{||\boldsymbol{x_i} - \boldsymbol{c_j}||}{||\boldsymbol{x_i} - \boldsymbol{c_k}||} \right)^{\frac{2}{b-1}}}$$

And

$$\boldsymbol{c_j} = \frac{\sum_x \boldsymbol{w_k}(\boldsymbol{x})^{\boldsymbol{b}} \boldsymbol{x}}{\sum_x \boldsymbol{w_k}(\boldsymbol{x})^{\boldsymbol{b}}}$$

Step 3: Update centroid of each cluster by calculating the average of all data points within this cluster. Repeat Step 2 to reclassify each data points using updated centroids. The iteration terminates and jumps to Step 4 when change of data points over the total number of data points is less than threshold ε.

Step 4: The output is the generated centroid of each cluster. The cluster with largest number of data points should be defined as the output for the users and pushed to users' application.

The output will be saved as JSON objects in our backend database. Google Pub/Sub service would still be used as the intermediate layer to download to phone application. Differently in this case our server will turn to the publisher and the result will be published to a new topic in Pub/Sub service while our phone application works as the subscriber to capture the result packet. Different from our server we discussed before, phone application does not necessarily need to listen to the Pub/Sub service all the time. The listening process could start only when users select to use suggestions from our

collected data and could terminate once the packet has been received from the Pub/Sub service.

In the longer term, it is also possible to merge our healthcare system into Amazon Alexa Ecosystem or Google Home Ecosystem as a third-party application to enable voice recognition services as well as remote configurations through Alexa-enabled devices or Google Home devices.

REFERENCES

1) "National Health Accounts Historical." CMS.gov Centers for Medicare & Medicaid Services, 11 Dec. 2018. https://www.cms.gov/Research-Statistics-Data-and-Systems/Statistics-Trends-and-Reports/NationalHealthExpendData/NationalHealthAccountsHistorical.html.
2) Young, Aaron, et al. "A Census of Actively Licensed Physicians in the United States, 2016." Journal of Medical Regulation, vol. 103, no. 2, 2017, pp. 7–21., doi:10.30770/2572-1852-103.2.7.
3) OECD (2019), Hospital beds (indicator). doi: 10.1787/0191328e-en (Accessed on 24 September 2019).
4) Li, Zheng, et al. "Early Observations on Performance of Google Compute Engine for Scientific Computing." 2013 IEEE 5th International Conference on Cloud Computing Technology and Science, 2013, doi:10.1109/cloudcom.2013.7.
5) Wahid, Abdul, and M. Tariq Banday. "Machine Type Comparative of Leading Cloud Players Based on Performance & Pricing." 2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI), 2018, doi:10.1109/icacci.2018.8554366.
6) Google Inc. "Googleapis/Google-Cloud-Node." GitHub, 28 June 2019, https://github.com/googleapis/google-cloud-node.
7) Particle.io. "Particle Reference Documentation", https://docs.particle.io/reference/device-cloud/api/.

8) Chung, Hyunji, Park, Jungheum & Lee, Sangjin. “Digital Forensic Approaches for Amazon Alexa Ecosystem.” *Digital Investigation*, vol. 22, 2017, doi:10.1016/j.diin.2017.06.010.

9) Google Inc. “Firebase Realtime Database.”, https://firebase.google.com/docs/database.

APPENDIX A

SOURCE CODE FOR FIRMWARE ON HARDWARE DEVICE

```
//  FIVE OPTIONS TO CONTROL FIVE PINS ON BOARD IN TWO CHANNELS

int option1 = D0; //  option1 is pinned to D0 on board
int option2 = D1; //  option2 is pinned to D1 on board
int option3 = D2; //  option3 is pinned to D2 on board
int option4 = D3; //  option4 is pinned to D3 on board
int option5 = A7; //  option5 is pinned to A7 on board

//  ILLUMINATION INTENSITY FOR EACH OPTION
int option1Value = 0;
int option2Value = 0;
int option3Value = 0;
int option4Value = 0;
int option5Value = 0;

//  ILLUMINATION INTENSITY IN ARRAY FOR EACH OPTION IN ORDER TO
UPLOAD
char option1ValueinChar[4];
char option2ValueinChar[4];
char option3ValueinChar[4];
char option4ValueinChar[4];
char option5ValueinChar[4];

//  FREQUENCY CONTROL FOR ALL OPTIONS
int frequencyValue = 0;
char frequencyinChar[4];

//  INDICATOR FOR ON BOARD LED(PIN D7) TO AVOID
int onBoardLED = D7;

//  INDICATOR FOR NEW INCOMING REQUEST
bool changeOccurred = false;

//  INDICATOR OF WHICH OPTION IS REQUESTED TO CHANGE
int switchCase = 0;

//  INDICATOR OF WHICH CHANNEL IS USING
int currentMode = 0;
```

```
//  INDICATOR OF STARTING POINT AND ENDING POINT OF A TREATMENT
String recordingNow = "F";

//  INDICATOR OF WHICH INDEX OF FREQUENCY FROM APP IS USING,
RANGING FROM 0 TO 5
int blinkingFrequency = 0;
void setup()
{
  pinMode(option1,OUTPUT);
  pinMode(option2,OUTPUT);
  pinMode(option3,OUTPUT);
  pinMode(option4,OUTPUT);
  pinMode(option5,OUTPUT);

  Particle.function("led",ledToggle);

  sprintf(frequencyinChar, "%02d",frequencyValue);
}

//  INFINITE LOOP LISTENING NEW REQUESTS AND REACTING. THE
OPERATION WILL KEEP UNLESS NEW REQUEST ARRIVES
void loop()
{
   Response(blinkingFrequency);
   Communications();
}

//  READ AND INTERPRET REQUEST INTO OPERATIONS
int ledToggle(String command) {
   changeOccurred = true;
   String brightness = command.substring(6,10);
   recordingNow = command.substring(10,11);
   if (command.substring(1,11) == "STARTRECI" || command.substring(1,11) ==
"TERMINATEF"){
      currentMode = (command.substring(0,1)).toInt();
      switchCase = 7;
      return 1;
   }
   String Temp = command.substring(1,10);
   if (Temp == "FREUPDATE"){
      currentMode = (command.substring(0,1)).toInt();
      String temp;
      temp = command.substring(11,12);
```

```
    blinkingFrequency = temp.toInt();
    if(blinkingFrequency == 1)
       frequencyValue = 0;
    else if(blinkingFrequency == 2)
       frequencyValue = 1;
    else if(blinkingFrequency == 3)
       frequencyValue = 10;
    else if(blinkingFrequency == 4)
       frequencyValue = 20;
    else if(blinkingFrequency == 5)
       frequencyValue = 40;
    switchCase = 6;
    return 1;
}
if (command.substring(0,1) == "0")
    currentMode = 0;
else if (command.substring(0,1) == "1")
    currentMode = 1;

if (command.substring(1,6) == "850nm"){
    option1Value = brightness.toInt();
    switchCase = 1;
    return 1;
}
else if (command.substring(1,6) == "940nm"){
    option2Value = brightness.toInt();
    switchCase = 2;

    return 1;
}
else if (command.substring(1,6) == "830nm"){
    option3Value = brightness.toInt();
    switchCase = 3;
    return 1;
}
else if (command.substring(1,6) == "660nm"){
    option4Value = brightness.toInt();
    switchCase = 4;
    return 1;
}
else if (command.substring(1,6) == "630nm"){
    option5Value = brightness.toInt();
    switchCase = 5;
    return 1;
```

```
    }
    else if (command.substring(1,6) == "WGWGW"){
        option1Value = brightness.toInt();
        switchCase = 1;
        return 1;
    }
    else if (command.substring(1,6) == "WWWWW"){
        option2Value = brightness.toInt();
        switchCase = 2;
        return 1;
    }
    else if (command.substring(1,6) == "RRRRR"){
        option3Value = brightness.toInt();
        switchCase = 3;
        return 1;
    }
    else if (command.substring(1,6) == "GGGGG"){
        option4Value = brightness.toInt();
        switchCase = 4;
        return 1;
    }
    else if (command.substring(1,6) == "BBBBB"){
        option5Value = brightness.toInt();
        switchCase = 5;
        return 1;
    }
    else {
         return -1;
    }
}

//  UPLOAD AUTHORIZED OPERATIONS TO THE CLOUD FOR ANALYSIS
void Communications(){
    if(changeOccurred == true && recordingNow == "T" || changeOccurred == true &&
switchCase == 7){
    switch(switchCase){
        case 1:{
            sprintf(option1ValueinChar,"%d", option1Value);
            if (currentMode == 1)

Particle.publish("D0_850nm",frequencyinChar+recordingNow+option1ValueinChar,60,
PRIVATE);
            else if (currentMode == 0)
```

```
Particle.publish("D0_GW",frequencyinChar+recordingNow+option1ValueinChar,60,
PRIVATE);
          break;
      }
      case 2:{
          sprintf(option2ValueinChar,"%d", option2Value);
          if (currentMode == 1)
            Particle.publish("D1_940nm",recordingNow+option2ValueinChar,60,
PRIVATE);
          else if (currentMode == 0)

Particle.publish("D1_WW",frequencyinChar+recordingNow+option2ValueinChar,60,
PRIVATE);
          break;
      }
      case 3:{
          sprintf(option3ValueinChar,"%d", option3Value);
          if (currentMode == 1)

Particle.publish("D2_830nm",frequencyinChar+recordingNow+option3ValueinChar,60,
PRIVATE);
          else if (currentMode == 0)

Particle.publish("D2_RED",frequencyinChar+recordingNow+option3ValueinChar,60,
PRIVATE);
          break;
      }
      case 4:{
          sprintf(option4ValueinChar,"%d", option4Value);
          if (currentMode == 1)

Particle.publish("D3_660nm",frequencyinChar+recordingNow+option4ValueinChar,60,
PRIVATE);
          else if (currentMode == 0)

Particle.publish("D3_GREEN",frequencyinChar+recordingNow+option4ValueinChar,6
0, PRIVATE);
          break;
      }
      case 5:{
          sprintf(option5ValueinChar,"%d", option5Value);
          if (currentMode == 1)
```

```
Particle.publish("A7_630nm",frequencyinChar+recordingNow+option5ValueinChar,60,
PRIVATE);
        else if (currentMode == 0)

Particle.publish("A7_BLUE",frequencyinChar+recordingNow+option5ValueinChar,60,
PRIVATE);
        break;
      }
      case 6:{
        sprintf(option1ValueinChar,"%d", option1Value);
        sprintf(option2ValueinChar,"%d", option2Value);
        sprintf(option3ValueinChar,"%d", option3Value);
        sprintf(option4ValueinChar,"%d", option4Value);
        sprintf(option5ValueinChar,"%d", option5Value);
        sprintf(frequencyinChar,"%02d", frequencyValue);
        if (currentMode == 1){

Particle.publish("D0_850nm",frequencyinChar+recordingNow+option1ValueinChar,60,
PRIVATE);
            delay(250);

Particle.publish("D1_940nm",frequencyinChar+recordingNow+option2ValueinChar,60,
PRIVATE);
            delay(250);

Particle.publish("D2_830nm",frequencyinChar+recordingNow+option3ValueinChar,60,
PRIVATE);
            delay(250);

Particle.publish("D3_660nm",frequencyinChar+recordingNow+option4ValueinChar,60,
PRIVATE);
            delay(250);

Particle.publish("A7_630nm",frequencyinChar+recordingNow+option5ValueinChar,60,
PRIVATE);
        }
        else if (currentMode == 0){

Particle.publish("D0_GW",frequencyinChar+recordingNow+option1ValueinChar,60,
PRIVATE);
            delay(250);
```

```
Particle.publish("D1_WW",frequencyinChar+recordingNow+option2ValueinChar,60,
PRIVATE);
            delay(250);

Particle.publish("D2_RED",frequencyinChar+recordingNow+option3ValueinChar,60,
PRIVATE);
            delay(250);

Particle.publish("D3_GREEN",frequencyinChar+recordingNow+option4ValueinChar,6
0, PRIVATE);
            delay(250);

Particle.publish("A7_BLUE",frequencyinChar+recordingNow+option5ValueinChar,60,
PRIVATE);
          }
          switchCase = 0;
          break;
      }
      default:{
          sprintf(option1ValueinChar,"%d", option1Value);
          sprintf(option2ValueinChar,"%d", option2Value);
          sprintf(option3ValueinChar,"%d", option3Value);
          sprintf(option4ValueinChar,"%d", option4Value);
          sprintf(option5ValueinChar,"%d", option5Value);
          sprintf(frequencyinChar,"%02d", frequencyValue);
          if (currentMode == 1){

Particle.publish("D0_850nm",frequencyinChar+recordingNow+option1ValueinChar,60,
PRIVATE);
            delay(250);

Particle.publish("D1_940nm",frequencyinChar+recordingNow+option2ValueinChar,60,
PRIVATE);
            delay(250);

Particle.publish("D2_830nm",frequencyinChar+recordingNow+option3ValueinChar,60,
PRIVATE);
            delay(250);

Particle.publish("D3_660nm",frequencyinChar+recordingNow+option4ValueinChar,60,
PRIVATE);
            delay(250);
```

```
Particle.publish("A7_630nm",frequencyinChar+recordingNow+option5ValueinChar,60,
PRIVATE);
        }
        else if (currentMode == 0){

Particle.publish("D0_GW",frequencyinChar+recordingNow+option1ValueinChar,60,
PRIVATE);
            delay(250);

Particle.publish("D1_WW",frequencyinChar+recordingNow+option2ValueinChar,60,
PRIVATE);
            delay(250);

Particle.publish("D2_RED",frequencyinChar+recordingNow+option3ValueinChar,60,
PRIVATE);
            delay(250);

Particle.publish("D3_GREEN",frequencyinChar+recordingNow+option4ValueinChar,6
0, PRIVATE);
            delay(250);

Particle.publish("A7_BLUE",frequencyinChar+recordingNow+option5ValueinChar,60,
PRIVATE);
        }
        switchCase = 0;
      }
      }
   changeOccurred = false;
   }
}

//  RESPONDING TO THE REQUEST ACCORDINGLY
void Response(int blinkingMode){
   switch (blinkingMode){
      case 1:{
         analogWrite(option1,option1Value);
         analogWrite(option2,option2Value);
         analogWrite(option3,option3Value);
         analogWrite(option4,option4Value);
         analogWrite(option5,option5Value);
         break;
      }
      case 2:{
```

```
    analogWrite(option1,option1Value);
    analogWrite(option2,option2Value);
    analogWrite(option3,option3Value);
    analogWrite(option4,option4Value);
    analogWrite(option5,option5Value);
    delay(1000);
    analogWrite(option1,0);
    analogWrite(option2,0);
    analogWrite(option3,0);
    analogWrite(option4,0);
    analogWrite(option5,0);
    delay(1000);
    break;
  }
  case 3:{
    analogWrite(option1,option1Value);
    analogWrite(option2,option2Value);
    analogWrite(option3,option3Value);
    analogWrite(option4,option4Value);
    analogWrite(option5,option5Value);
    delay(100);
    analogWrite(option1,0);
    analogWrite(option2,0);
    analogWrite(option3,0);
    analogWrite(option4,0);
    analogWrite(option5,0);
    delay(100);
    break;
  }
  case 4:{
    analogWrite(option1,option1Value);
    analogWrite(option2,option2Value);
    analogWrite(option3,option3Value);
    analogWrite(option4,option4Value);
    analogWrite(option5,option5Value);
    delay(50);
    analogWrite(option1,0);
    analogWrite(option2,0);
    analogWrite(option3,0);
    analogWrite(option4,0);
    analogWrite(option5,0);
    delay(50);
    break;
  }
```

```
        case 5:{
            analogWrite(option1,option1Value);
            analogWrite(option2,option2Value);
            analogWrite(option3,option3Value);
            analogWrite(option4,option4Value);
            analogWrite(option5,option5Value);
            delay(25);
            analogWrite(option1,0);
            analogWrite(option2,0);
            analogWrite(option3,0);
            analogWrite(option4,0);
            analogWrite(option5,0);
            delay(25);
            break;
        }
        default:
            analogWrite(option1,option1Value);
            analogWrite(option2,option2Value);
            analogWrite(option3,option3Value);
            analogWrite(option4,option4Value);
            analogWrite(option5,option5Value);
            break;
    }
}
```

APPENDIX B

SOURCE CODE FOR CLOUD SERVER IN GW OPTION

```
var colors = require('colors');

var util = require('util');
var needACK = 'F';

var config = {
        ProjectId: 'cmvision-led',
        SubSubscriptionName: 'projects/cmvision-led/subscriptions/WWopt',
        KeyFilePath: './gcp_private_key.json'
}
check_config();
console.log(colors.grey('Connecting to Google Cloud...'))
var gcloud = require('google-cloud')({
   projectId: config.ProjectId,
        keyFilename: config.KeyFilePath,
});
console.log(colors.grey('Connection is established...'))
var datastore = gcloud.datastore();
var pubsub = gcloud.pubsub();

var subscription = pubsub.subscription(config.SubSubscriptionName);

subscription.on('message', function(message) {
        console.log(colors.grey('Packet received from Pub/Sub service!\r\n'),
create_record_for_datastore(message, true));
        RouteData(message);
        message.ack();
        if (message.data.substring(2,3) != needACK){
                needACK = message.data.substring(2,3);
                publishACK();
        }
});

function publishACK() {

        var ack_config = {
        ProjectId: 'cmvision-led',
        SubSubscriptionName: 'projects/cmvision-led/subscriptions/acksubscription',
```

```
        KeyFilePath: './gcp_private_key.json'
}
        console.log(colors.black('Connecting to Google Cloud in ACK...'))

        var ack_gcloud = require('google-cloud')({
        projectId: ack_config.gcpProjectId,
                keyFilename: ack_config.gcpServiceAccountKeyFilePath,
                });
                var ack_pubsub = ack_gcloud.pubsub();

                var topic = ack_pubsub.topic('ack');

                topic.publish({ data: 'ack' }, function(err, messageId) {
                  console.log('Acknowledgement published...', messageId);
        });
};

function RouteData(message) {
   var key = datastore.key('CMVision');
   datastore.save({
      key: key,
      data: create_record_for_datastore(message)},
            function(err) {
                        if(err) {
                                console.log(colors.red('There was an error occurred when
routing to database...'), err);
                        }
                        console.log(colors.grey('Data record stored into Datastore!\r\n'),
create_record_for_datastore(message, true))
            });
};

function create_record_for_datastore(message, log) {
        var obj = {
                pin_out: message.attributes.event.substring(0,2),
      device_id: message.attributes.device_id,
      event: message.attributes.event.substring(3),
      recording_status:message.data.substring(2,3),
      data: message.data.substring(3),
      frequency: message.data.substring(0,2),
      published_at: message.attributes.published_at
        }

        if(log) {
```

```
                return colors.grey(util.inspect(obj));
        } else {
                return obj;
        }
};

function check_config() {
        if(config.ProjectId === ''  || !config.ProjectId) {
                console.log(colors.red('The projectID is not valid...'));
                process.exit(1);
        }
        if(config.SubSubscriptionName === '' || !config.SubSubscriptionName) {
                console.log(colors.red('The Pub/Sub subscription name is not correct...'));
                process.exit(1);
        }
};
```