WORCESTER POLYTECHNIC INSTITUTE

Department of Electrical and Computer Engineering

MAJOR QUALIFYING PROJECT

Final Report

SMART HOME SYSTEM

Submitted to the Faculty of

Worcester Polytechnic Institute

In partial fulfillment of the requirements for the

Degree of Bachelor of Science,

Electrical and Computer Engineering

By

_____

Anh Tran

_____

Duc Tran

_____

Thinh Ly

Sequence Number: REL AA4H

Date: 03/02/2017

Professor Reinhold Ludwig, Project Advisor

Abstract

This project involves the design and implementation of a Smart Home system using IoT solutions. Three types of sensors, namely an occupancy sensor, a light sensor and a temperature sensor, along with a security camera are used and incorporated with a microcontroller in a master/slave architecture via Zigbee, a short-range network communication. The data collected from these sensors is transmitted to a cloud-based platform through Wi-Fi for analyzing and downloading to personal smartphones via a designated user interface. The entire system can be controlled both by users' smartphones and by personal computers.

Acknowledgements

We would like to dedicate special thanks to Professor Reinhold Ludwig for the meticulous guidance and innovative instructions throughout the project. We also want to thank Professor James P.O'Rourke and electronic technicians in ECE Shop, Mr. William Appleyard for helping us in the equipment ordering process and for advising the packaging and modeling. We also appreciated the assistance of the ZigBee corporation team in setting up the configuration for the module. Finally, we would like to devote notable thanks to the Department of Electrical and Computer Engineering at WPI for having given us the opportunity to grow and challenging us to solve real-world problems.

Contents

Abstract i
Acknowledgements ii
1. Introduction 8
1.1. Motivation 8
1.2. Applications 9
1.3. Existing Solutions and Market Research 9
1.3.1. Individual Modules 9
1.3.2. Central Hub System 11
1.3.3. Industrial Products 13
1.4. Problem Statement 14
2. Objectives 17
2.1. Customer Requirements 17
2.2. Product Requirements 17
2.3. Project Goals 18
3. Design Approaches and Solutions 19
3.1. Short-range Communication Protocols for Home Automation 19
3.1.1. Bluetooth Low Energy 19
3.1.2. ZigBee Communication 21
3.1.3. Z-Wave Communication 21
3.1.4. IPv6 over Low-power Wireless Personal Area Networks (6LowPAN) 22
3.1.5. Wi-Fi Communication 22
3.1.6. Summary Short-range Communication Protocol 22
3.2. Microcontroller Modules 23
3.2.1. Arduino Mega 2560 Microcontroller Board 23
3.2.2. TI MSP432 Microcontroller Board 25
3.2.3. Raspberry Pi Board Computer 26
3.2.4. Summary Microcontroller Module 27
3.3. Sensor Modules 29
3.3.1. Occupancy Sensor Module 29
3.3.2. Temperature Sensor Module 31
3.4. Camera Module 33
3.4.1. ArduCam Mini Module 2MP 34
3.4.2. Raspberry Pi NoIR Camera Board v2 35

3.4.3. Pixy CMU Cam5 Sensors 36
3.4.4. Summary Camera Approach 38
4. Design Implementation 39
4.1. Sensor Functions 39
4.1.1. Occupancy Sensor 39
4.1.2. Temperature Sensor 40
4.1.3. Camera Module 41
4.2. Local sensor-communication ZigBee Protocol 43
4.2.1. Zigbee Xbee Parameter Configuration 43
4.2.2. Zigbee Xbee – Arduino Connection 47
4.2.3. Zigbee Xbee Implementation with Arduino 48
4.3. Local-server communication WiFi 50
4.3.1. ESP8266 WiFi Module Wiring: 50
4.3.2. ESP8266 WiFi Module Configuration: 53
4.3.3. ESP8266 WiFi Module Implementation: 54
4.4. User Interface 56
4.5. Integration 58
4.5.1. Routers Implementation 58
4.5.2. Coordinator Implementation 59
4.5.3. Web Server Implementation 64
5. Results 65
5.1. Functionality Testing 65
5.2. Case studies 69
5.2.1. Severe temperature Detection and Notification 69
5.2.2. Breakdown of WiFi connection 69
6. Deliverables 71
6.1. Modules Packaging 71
6.2. House Model Prototype 73
7. Discussions and Future Works 76
7.1. Initial Cost 76
7.2. Safety Analysis 76
7.3. Privacy and Security 77
7.4. Scalability 78
7.4.1. Transmission and server robustness 78

7.4.2. Variety of sensing mechanisms 78
7.4.3. Better User Interface and Easier Integration Scheme 79
8. Summary 80
References 81
Appendices 84
Appendix 1: Server/Website User Interface 85
Appendix 2: Arduino Code 106

LIST OF FIGURES

Figure 1: LIFX (Life-X) Smart Bulbs 11
Figure 2: NEST Thermostat. NEST Learning Thermostat is an electronic, programmable and self-learning Wi-Fi-enabled thermostat based on machine learning algorithm. It is designed by NEST LABS to help users optimize cooling and heating of their homes and businesses. [4] 11
Figure 3: Insteon Dual Mesh Network Diagram. [5] 12
Figure 4: Insteon Pool Control System. [6] 13
Figure 5: Proposed System Block Diagram. In Sensor Area Network, Master and Slave modules communicate by the ZigBee protocol. The Master module sends data and receives command from the Remote Control. 15
Figure 6: Individual Module Proposed Design. Each sensor/appliance module consists of ZigBee module, Wi-Fi module, Sensors and different appliances modules. They can interchange the roles of Master and Slave, which makes the system more durable. 16
Figure 7: Bluetooth. Bluetooth Smart Ready device is the center of the system, which connecting both normal Bluetooth devices and Bluetooth Smart devices (sensors, using small bits of data, using little energy). 19
Figure 8: Mesh network. [12] 21
Figure 9: Arduino Mega and Genuino Mega board. [16] 25
Figure 10: TI MSP432 LaunchPad. [17] 26
Figure 11: Raspberry Pi Model B. [18] 27
Figure 12: PIR sensor. [19] 29
Figure 13: HC-SR04 module for UltraSonic Motion Sensor. [20] 30
Figure 14: Mercury thermometer for the measurement of room temperature. [33] 32
Figure 15: Thermistor. [34] 32
Figure 16: TMP36 from Analog Devices. [35] 33
Figure 17: ArduCam Mini Module 2MP. [21] 35
Figure 18: Raspberry Pi NoIR Camera Board V2. [22] 36
Figure 19: Pixy CMU Cam5 Sensors. [23] 37
Figure 20: Ultrasonic Wave Terminology. [38] 40
Figure 21: Implementation of Ultrasonic Sensor HC-SR04. 40
Figure 22: TMP 36 – IC Temperature sensor from Analog Devices to control indoor temperature and detect critical overheating situation. [40] 41
Figure 23: Implementation of temperature sensor TMP36. 41
Figure 24: ArduCam Mini 2MP- Camera Module for Capturing Image Compatible with Arduino Development Board. In this image, there are only 6 pins visible, since the 5V and GND pins are female pin headers, so they are not showed in this image. [36] 42
Figure 25: ArduCam Mini 2MP Connection with Arduino Module. The ArduCam Mini 2MP module is also provided with supporting library for Arduino Development Board. [36] 43
Figure 26: Zigbee Xbee Module. [24] 44
Figure 27: Sparkfun Xbee Explorer USB, which connects the Zigbee Xbee to a computer with the XCTU software. [25] 44
Figure 28: Discover Xbee Module Setting in XCTU user interface. 45
Figure 29: Changing Xbee configuration parameters in XCTU user interface. 46
Figure 30: Changing Xbee destination parameters in XCTU. 47
Figure 31: Addressing Serial Interfacing Parameters in XCTU. 47

Figure 32: Sparkfun Xbee Explorer Regulated, which has a 3.3V regulator to power Xbee from Arduino 5V power supply. [39] 48
Figure 33: Connect Zigbee Xbee to Explorer board. 48
Figure 34: Boot Loader Menu. [41] 49
Figure 35: Example of Xbee Transmit Request Frame format. The example shows how to send a transmission to device with address 0x0013A200 400114011, and message "TxData1B". [42] 50
Figure 36: ESP8266 WiFi Module. [37] 51
Figure 37: ESP8266 Pin Diagram. [43] 51
Figure 38: Logic Level Converter from SparkFun, The HV (High Voltage Pin) is connected to 5V source. The LV (High Voltage Pin) is connected to 3.3V source. [44] 52
Figure 39: The connection diagram between the ESP8266 module, the logic level converter, and the Arduino Mega 2560 microcontroller [45]. 53
Figure 40: AT Command Lists. The AT Command is divided into Layer: Basic set up ESP8266, WiFi Layer to set up Internet Connection, and TCPIP Layer to send and receive HTTP messages [46]. 54
Figure 41: Designed Website for User Interface 57
Figure 42: checkCommand()reads the 4th byte of the command, and turn the LED on/off 58
Figure 43: Code snippet for Router 1 request response. It starts by writing "Sending" to Computer console, then send 5-byte status message to Arduino TX pin (DIN pin on Xbee for transmission). 59
Figure 44: Router 2 command execution code snippet. 59
Figure 45: ZigBee message format and the message that are used. The 14th to 17th bytes are the address of the destination router, and the 5 bytes just before the last byte is the data to be sent. 60
Figure 46: HTTP GET message format for sending JSON data "uri" is the name of the php file that handle receiving data on the web server. "json" is the data to be sent in json format, and "server" is the URL of the web server. 61
Figure 47: JSON data format. Each JSON object has multiple keys and values "sender" is the router that sent the JSON data. The data "time" is a counter value that we use to keep track of sending time; and data "value" is the temperature value from the router to the heater module. 61
Figure 48: HTTP POST message for sending an image file date. This HTTP message indicates the sending of binary data as a file CAM.TXT to the "userfile" folder in the web server. 62
Figure 49: Camera setup code. Due to hardware limitation, we only used image size of 640x480 63
Figure 50: readSonic() function 64
Figure 51: Data Flow in our designed system. After the data is sent to the server by the coordinator (the master Arduino) it is processed and analyzed in the backend before being presented in the front-end website of the user interface. 65
Figure 52: Response message from the Wifi module indicating successful delivery of JSON data. On the left is the response message for successfully sending data to the web server. The first line indicates establishing the TCP connection. The second line indicates the length of the HTTP message. The third line indicates that the message has been sent. Next, the HTTP response indicates successful reception of the message. The last line indicates the closing of the TCP connection. On the right is the sample message for receiving commands from the web server. The first line and second line indicate reception of a command from the webserver to turn on/off the LED in router 1. Next is the status of forwarding the command to router 1. The fourth line indicates the time in milliseconds between sending the command and receiving the acknowledgement message from router 1, and the last line is the acknowledgement message in hexadecimal. 66
Figure 53: Example of sending request message and receiving requested information from router 1. The first two lines, as mentioned in figure 53, indicated the time in milliseconds between sending the request message and receiving the acknowledgement message. The third, fourth and fifth line indicates receiving

of requested information from router 1, with the fifth line is the message in hexadecimals. In the message, “4152EC6B” is the MAC address for the router 1 ZigBee module, and “3130303030” is the requested information in hexadecimals. All time measurements are in milliseconds. 67
Figure 54: An example of a requested message received from the coordinators at router 1. The message is in hexadecimals. The first byte of the message (31, which is character ‘1’ in ASCII) indicates the destination routers. The fourth byte (30, which is character ‘0’ in ASCII) is used to command the LED turning on or off. However, the fourth byte will only be considered if the fifth byte value is 30, which indicates command message. For this example, the fifth byte is 31, which indicates a request message. . 67
Figure 55: An example of a command message received from the coordinators at router 2. The message is in hexadecimals. The first byte of the message (32, which is character ‘2’ in ASCII) indicates the destination routers. The fourth byte (12, which is 18 in decimal) is used to turn the fan on to decrease the temperature to 18^0C. The fourth byte will only be considered if the fifth byte value is 30, which indicates command message. For this example, the fifth byte is 30. 68
Figure 56: Example of severe temperature warning sent from server. 69
Figure 57: Code snippet of WiFi verification with AT command. 70
Figure 58: Example of using AT command to set Xbee configuration parameters. AT command starts with “+++” to turn Xbee into configuration mode. It is followed by destination address. In this case, Router 1 destination address is set to “4152ECD7”, which is the MAC address of Router 2 instead of the Coordinator’s address. We also set the Network ID (discussed in section 4.2) to 123 (original ID) since we notice the ID parameter got scrambled after we set the new destination address. After that, “ATWR” is sent to save all configuration parameters to Xbee. It ends with “ATCN” to get out of configuration mode. 70
Figure 59: Packaging of the camera and ultrasonic sensor modules. 72
Figure 60: Packaging of the PIR occupancy sensor. 73
Figure 61: Front-view of the house prototype. 74
Figure 62: Side-view of the house prototype. 74
Figure 63: Another side-view of the house prototype. 75
Figure 64: Down-view of the house prototype. 75
Figure 65: Email warning of indoor critical temperature detected. 77

LIST OF TABLES

Table 1: Internet of Things Units Installed Base on Category (Millions of Units) 8
Table 2: Designed Applications for Home Automation 9
Table 3: Comparison between different Microcontrollers. 28
Table 4: Comparisons of different Motion Detection Technologies. 31
Table 5: Comparison of different camera modules. 38
Table 6: Dimension of designed house model prototype 73
Table 7: Components List 76

1. Introduction

1.1. Motivation

In recent years, the phrase "Smart Home" has been one of the most frequently searched keywords. Since 2013, with the fast-paced development of multiple technologies, ranging from wireless communication to the Internet and from embedded systems to micro-electromechanical systems (MEMS), the vision of the Internet of Things (IoT) has evolved substantially, leading to the networks of physical devices, vehicles, buildings with electronics, software, sensors and actuators. As the availability of Internet is widely spread, more devices are being implemented with Wi-Fi capabilities, technology costs are going down and smartphone penetration is escalating, these has created an advanced and leading edge environment for IoT. This concept involves mapping of the physical world to a virtual world, interconnecting devices with sensing capabilities, and passing their collective information to the cloud. In other words, the technology enables devices to collect and exchange data, providing real-time notifications and updates for different application purposes. Along with the indefatigable development, the technology is widely used in a plethora of areas, including but not limited to smart grid, intelligent transportation, smart city and home automation, each of which is not only uniquely identifiable through its embedded computing system but also able to interoperate within the existing Internet infrastructure. By estimations of analyst firm Gartner, Inc., IoT will consist of 6.4 billion things in 2016, up 30 percent from 2015 and will reach more than 20 billion objects by 2020[1]. Table 1 represents statistics for IoT units installed based on categories.

Table 1: Internet of Things Units Installed Base on Category (Millions of Units) [1]

Category	2014	2015	2016	2020
Consumer	2,277	3,023	4,024	13,509
Business: Cross-Industry	632	815	1,092	4,408
Business: Vertical-Specific	898	1,065	1,276	2,880
Total (Millions of Units)	**3,807**	**4,902**	**6,392**	**20,797**

Due to the increased need for low-power consumption, green energy and more secure houses, there have been several improvements that can make a house smarter and more efficient. A smart home, in fact, is where household devices and home appliances could be monitored and controlled remotely. When these household devices connect with the Internet using the proper network architecture and standard communication protocols, the entire system is regarded as IoT based Smart Home. For instance, Smart Locks helps you lock or unlock your door just by a simple tap of a button on an application in your smartphone utilizing the built-in GPS sensing in this device. Smart lights can save you a significant amount of money and secure your home by automatically reacting to your daily routines and preferences. You can also keep track of your home energy bill by automatically turning down your thermostat of the A/C unit through a simple app from your phone. A moisture sensor can effectively detect a minor leak in your home and alert you immediately. IoT has opened endless applications in transforming homes to be smarter, safer secur and environmental-friendly. Indeed, the technology covers a wide range of

industries and uses cases that scale from a single constrained device to massive cross-platform deployments of embedded technologies and real-time cloud systems [2]. As the trend keeps rising, a Smart Home will certainly be one of the most innovative IoT platforms.

1.2. Applications

The Smart Home System Prototype gives home-owners the following basic functionalities as summarized in Table 2.

Table 2: Designed Applications for Home Automation

Functionalities	How
Reduce the power consumption	Set up different modes for Smart Home System: occupant, no occupant
Remotely control house from a website user interface	Commands from website will be send to central hub, which talks to the appliance modules
Alarm when worst case scenarios (fire, thief, etc.) occur	When sensors from appliance modules detect extreme conditions, appliance modules send a warning to the server/hub to trigger "Hazard Mode" and give special tasks to all modules (camera on, heat turns off, etc.)

1.3. Existing Solutions and Market Research

To identify our project's scopes and targets, we conducted a thorough research on the smart home system market. We explored some of the most popular emerging products, investigating their technologies and standards. Based on our research, we divide the existing smart home products into two different types:

1.3.1. Individual Modules

"Individual Modules" is what we call IoT solutions that only deal with one specific aspect of a smart home system, for example only lighting, heating or security system. Most of these solutions provide users with a well-designed user interface, which could be installed on the user's PCs or smartphones, permitting them to remotely control the systems through their Internet-connected devices.

In terms of connection structures, these solutions rely on a Master/Slave architecture. When a system is installed, one device is chosen to be the Master. This Master device would be in charge of receiving commands from and reporting data to users via a home internet router. The other devices (Slave devices) in the system will communicate to the Master device using different types of low-power, short-range wireless network protocols, such as Low Power IPv6 802.15.4 Peer-To-Peer (P2P) network, ZigBee Mesh Network, Bluetooth Low Energy (BLE), etc. In order to do that, each device must be embedded with at least two different wireless network protocols, one is Wi-Fi and the other can be any type of low-power, short-range wireless network protocols.

There are several existing solutions that could be classified as an "Individual Module". For lighting systems, increasingly popular solution are the LIFX Smart Bulbs (in Figure 1). For heating, there is Nest Thermostat (in Figure 2). For security system, a good example is the Nest Protect.

We will discuss in depth the LIFX Smart Bulbs system. LIFX (Life-X) is a brand of multi-color, Wi-Fi-enabled LED light bulbs that can be controlled by Wi-Fi-connected devices such as smartphones or PCs. As mentioned above, LIFX Smart Bulb uses a Master/Slave network architecture, with a Wi-Fi microcontroller embedded on each device that allows communication between the Master bulb with router via 802.11a/802.11g/802.11n (Wi-Fi) and between the Master bulb and the Slave bulbs via Low Power IPv6 802.15.4 Peer-To-Peer Network. This type of connection does not require a central communication port, and therefore offers significant amount of flexibility. If the users fail to reach the Master bulb, a Slave bulb would be chosen to be the new Master in order to maintain the connectivity of the system.

For the LIFX Smart Bulb, the configuration is very simple as users only need to power up their LIFX bulb, and download the user interface application to their smartphones or PCs. When powering up, the bulb will automatically create its own local Wi-Fi access point, to which the user interface application will automatically look for and allow user to connect the bulb to their Internet-connected devices. The first bulb to be connected will be chosen to be the Master, and the other bulbs are Slave bulbs.

Aside from remote controlling of the lighting system, LIFX smart bulbs also allow users for high-level tasks such as scheduling, and cooperating operations with solutions from other vendors. For example, users could schedule the light to automatically turn on at 7PM and turn off at 12AM everyday by using the user interface application. LIFX Smart Bulbs could also be integrated with NEST Protect Smoke Detector, such as when NEST Protect detect smoke or CO, the LIFX Smart Bulbs should be flashing red to indicate dangers to users with hearing impair.

Although offering users with many different features and abilities, these "Individual Module" systems have their limitation. The first limitation is that, since each single solution only deals with one aspect of a house, such as lighting, heating, or security, users would need to incorporate many different solutions from different vendors to make their entire home "smart". This would require users to use many different user interfaces for different solutions, make management of their smart home relatively hard and inconvenience. Secondly, power consumption of these systems may become an issue, since they usually incorporate microcontrollers and sensor technologies, which may use a significant amount of power. For example, the LIFX Smart Bulbs in Figure 1 requires 18 Watts of power consumption to produce 1000 lumens (a measure of the total quantity of visible light emitted by a source), which can be produced using only 9-13 Watts by a normal LED light bulb.

Figure 1: LIFX (Life-X) Smart Bulbs

Embedded inside a LIFX Smart Bulbs is a Wi-Fi Microcontroller that uses Master/Slave Concept. [3]

Figure 2: NEST Thermostat. NEST Learning Thermostat is an electronic, programmable and self-learning Wi-Fi-enabled thermostat based on machine learning algorithm. It is designed by NEST LABS to help users optimize cooling and heating of their homes and businesses. [4]

1.3.2. Central Hub System

Along with "Individual Module" solutions, several companies in the Smart Home market also provide complete solutions, which allow users to control their entire Smart Home using one single solution from one single vendor. A typical example of this system is Insteon Home Automation System.

An Insteon Home Automation System consists of different Insteon Smart Devices (light bulbs, wall switches, thermostat, etc.) and an Insteon Central Hub. The connection architecture is peer-to-peer network (each node in the network will connect to several nearby neighbor nodes and connect to the central node), with the Insteon Central Hub as the central node. The network protocol is an integrated dual-mesh network that combines wireless radio frequency (RF) network protocol (such as ZigBee) and building's existing electrical wiring. An example for the dual-mesh network of the Insteon Home Automation System is shown in Figure 3.

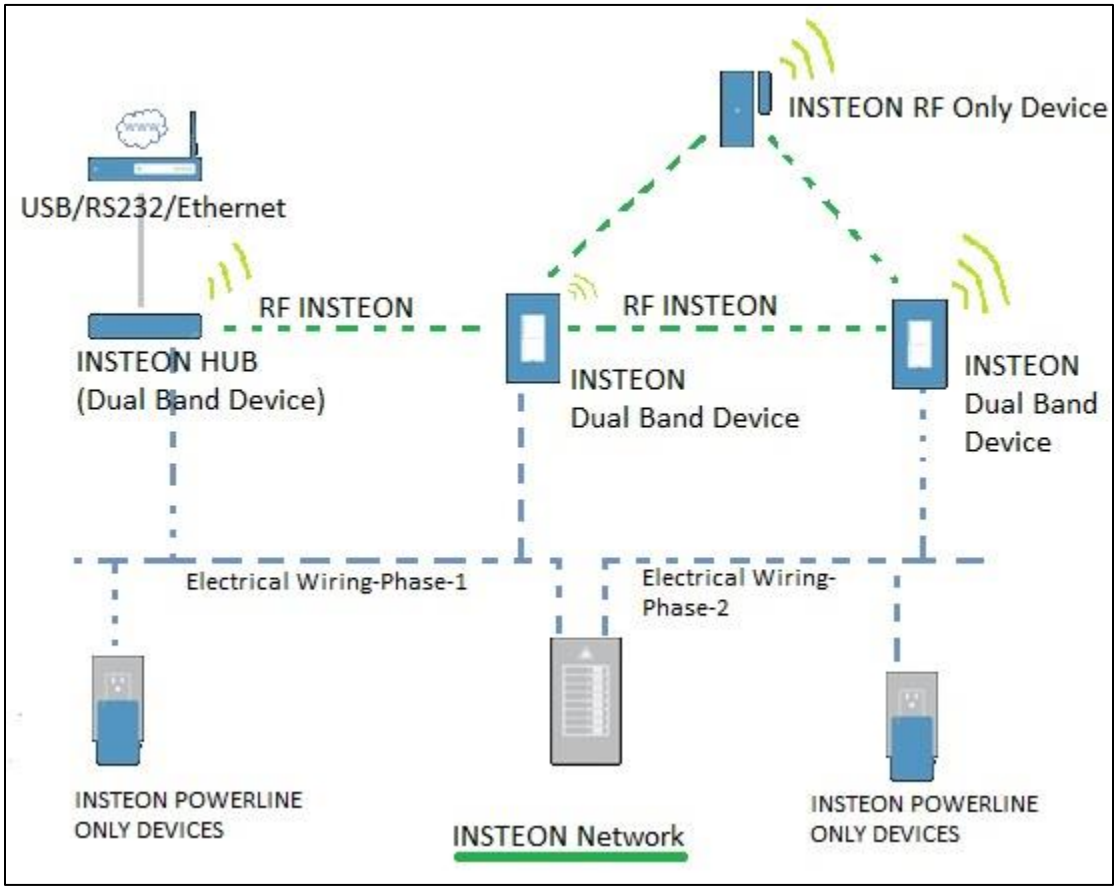

Figure 3: Insteon Dual Mesh Network Diagram. [5]

In this type of network as shown in Figure 3, we can see three different types of networks. The Insteon RF only device will communicate with its nearby neighbors that also use RF or dual-band network. The same process is applied to Insteon Powerline Only Devices. Then these Insteon Dual Band Devices will communicate with the Insteon Central Hub using either RF network or powerline connection. Finally, the Insteon Central Hub will send all the data received from other devices through USB connection or Internet to user's smartphone or PCs. Commands from users also go through the same path in the opposite way, from user's devices to Central Hub, where the commands are distributed to devices via RF communication or a powerline.

Also in Figure 4 is another example of Insteon Network for Pool Control Application. However, here the network protocol between the Central Hub and the On/Off Switch and the two Load Controller is a RF communication (such as ZigBee), and the network protocol between the

Central Hub and the Internet Router is Wi-Fi. There is also a user interface application for smartphones or PCs that allows remote control and monitor of the entire system.

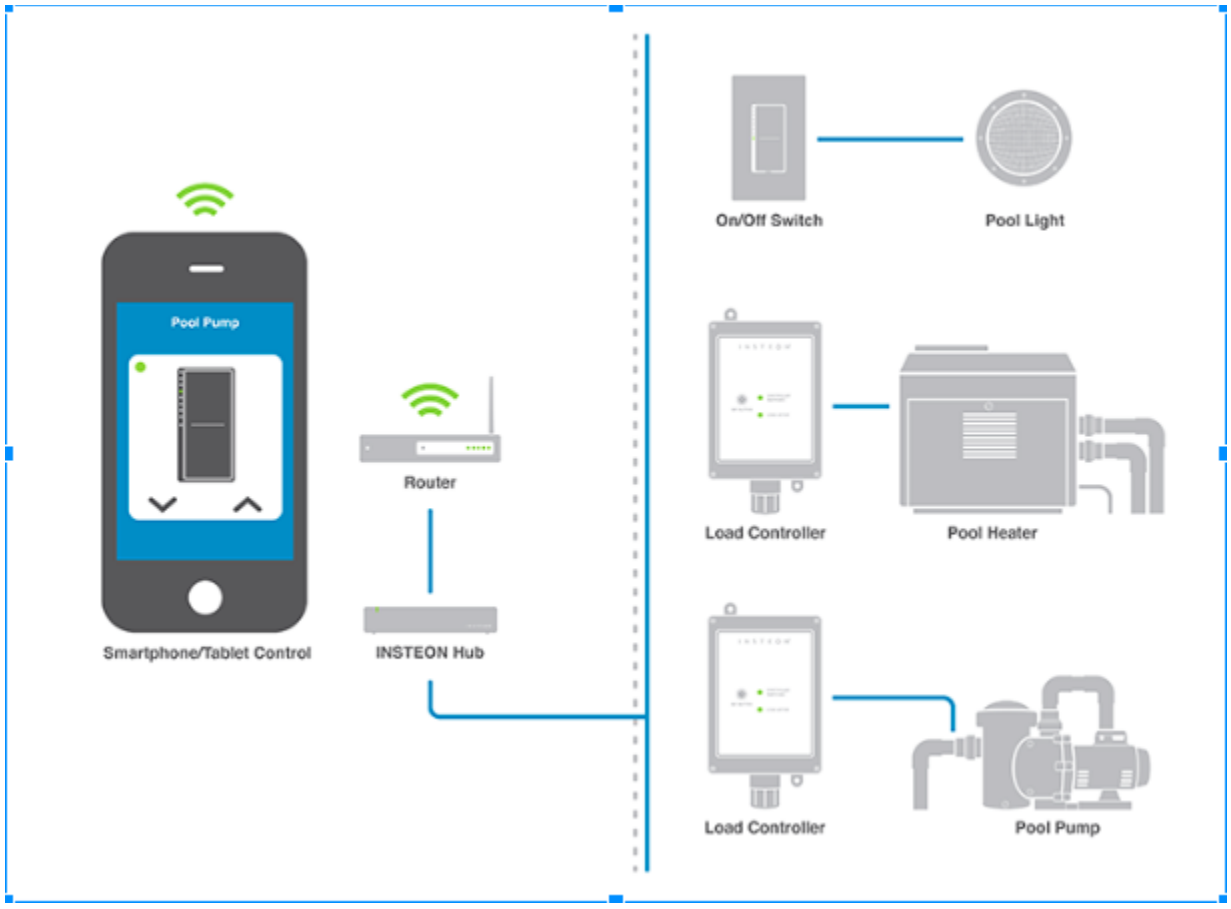

Figure 4: Insteon Pool Control System. [6]

Although allowing more flexibility and convenience when compared with "Individual Module" solutions, the Insteon Home Automation System, as is typical for other Home Automation System in the market, has some drawbacks. The most obvious drawback is that if in a worst-case scenario, the connection between the Insteon Hub and the Internet router is lost (Hardware malfunction), and users would lose control over their entire systems. The second drawback is that these types of systems would often require a much more complex initial setting and configuration, in comparison with the "plug and play" capability of "Individual Modules", which simplifies the configuration process as much as possible for users.

1.3.3. Industrial Products

Through our market research, we identified three basic characteristics for every smart home system:

- **Low Energy Consumption in compare to Traditional Home**: This is one of the top priority standards for a smart home system and the target objective of every smart home system on the market. By incorporating sensors technology and software solution into home appliances, these smart home systems are able to control and optimize their power usage in compare to traditional home, and help users to cut off

energy use when not necessary, such as automatically turn off a light when no people present, which is often forgotten by users in traditional home. One case study, conducted in May 2016 by the Fraunhofer Center for Sustainable Energy Systems CSE, and commissioned by the US Consumer Technology Association (CTA), estimated that by switching to smart homes, users could reduce up to 10 % of their monthly energy consumption. [7]

- **Automation:** Smart home systems are designed to free users from everyday tasks in their houses that are often neglected or too time-consuming. For instance, doors and windows could be set by users to be automatically locked after a specific time; light and heat could be automatically turned off when no people around and turned on when detected people. Main doors would automatically open when detecting family members, instead of having to carry a door key every time. Such operations are every inconvenient for users to do manually every day, but could be easily done automatically by smart home systems.
- **User Control Interface and Wireless Control:** This is a key requirement for every smart home system. Users must be able to control their entire home system remotely using a simple, user-friendly interface on their PCs or smartphones. This interface should present users with useful information of their home (such as power consumption, appliances status), allow users to assign tasks to their appliances and control them wirelessly without being home.

We also identify two inherent problems with existing smart home IOT solutions:

- **Lack of Combined Operations and Combined User Interfaces**: Some of most popular smart home systems in the market currently are limited in their scopes of operation. These systems only deal with one aspect of household appliances (such as lighting, thermostat, or security). This problem limits the user's ability to have a single application to control their smart home and perform complex operations that would require multiple appliances involved.
- **Central Hub**: Most of the smart home systems on the market are currently dependent on a central communication gateway for transferring data between their appliances cloud and the server. In these systems, the appliances cloud would communicate with the central gateway using a low power wireless protocol (ZigBee, Bluetooth, etc.) and the central gateway would transfer the data to a server using Wi-Fi. If for some reason, the central gateway could not be reached (connection dropped, hardware malfunctioned, or system hacked) the users would lose control of their entire smart home system.

1.4. Problem Statement

The problem presented to us is to design and create a low-cost, easy-to-build and maintain, user-friendly smart home system prototype that not only brings home-owners basic functionalities (remotely monitor thermostat, lighting, and security camera) but also includes

sensor fusion and software solutions to alarm and guide users on worst case scenarios (fire, thief, etc.)

Because of the drawbacks associated with Central Hubs and Individual Module architectures (as discussed above), we would like to explore a "System Architecture with Centralized but Subsidiary Controllers". This approach would effectively eliminate the problems of the two earlier architectures, as demonstrated in Figure 5.

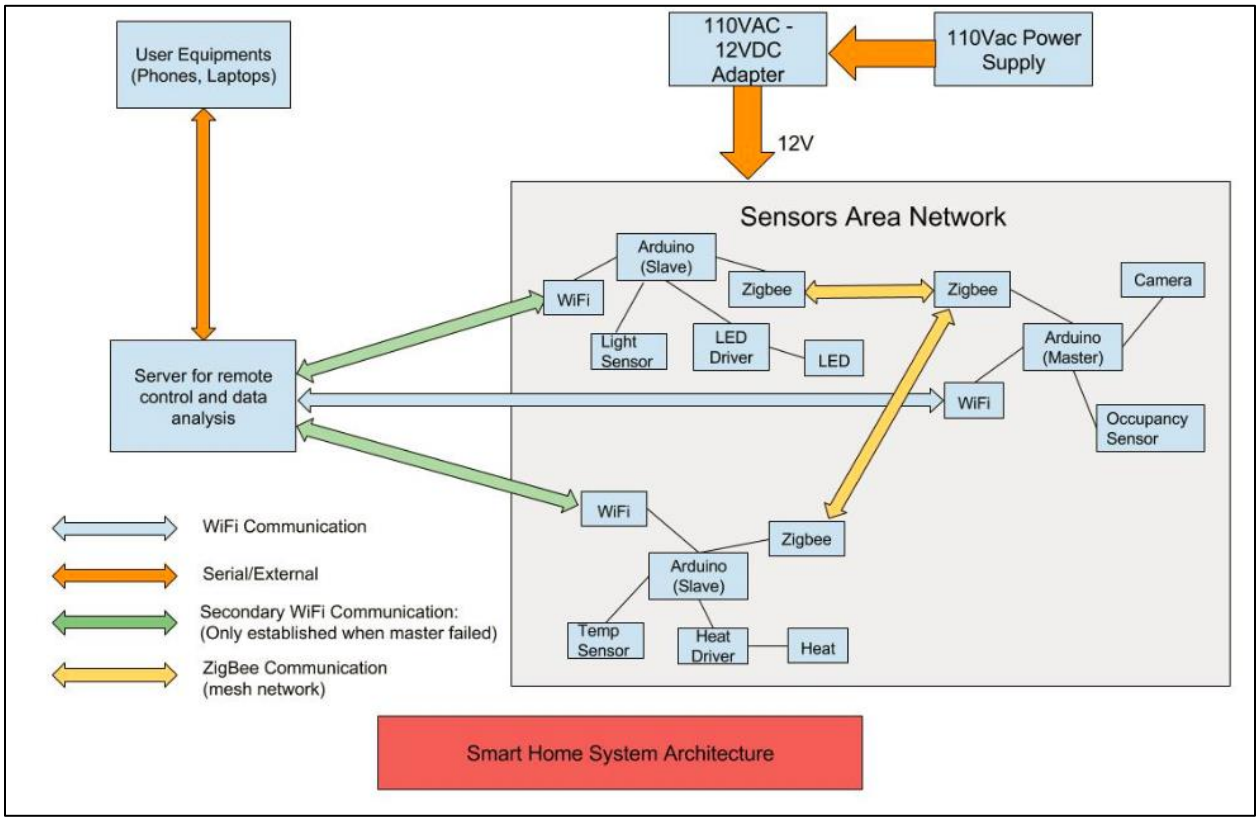

Figure 5: Proposed System Block Diagram. In Sensor Area Network, Master and Slave modules communicate by the ZigBee protocol. The Master module sends data and receives command from the Remote Control.

The proposed system consists of a Sensors Area Network and an External Server. Sensors Area Network contains a set of short-range connected sensors and appliances. The network has a Master Hub, which establishes communication to the external server via a Wi-Fi module and a Home Internet Router (Gateway). The Master Hub can send information from the sensor network to the server, or it can receive commands from the server and send them to the appliances. Users can send commands to the server from their personal laptops or smart phones. This IoT architecture has the following strengths when compared to the two existing predecessors.

The first advantage of the proposed architecture is that the separation of sensor networks short-range communication and the Master Hub – External Server communication improves the allocation of Internet bandwidth because short-range communication in the Sensors Area Network is generated from low-power wireless Machine-to-Machine (M2M) technologies such as Bluetooth, ZigBee, Internet Protocol version 6 (IPv6) over Low-power Wireless Personal Area Network (6LoWPAN), which does not interfere with normal household Internet connection. As a result, home-owners can have a smart home system but do not notice any

different in their Internet data rate since only Master Hub – External Server connection allocates Wi-Fi and a router. This feature resolves the problem proposed from Individual Modules architecture.

Secondly, each sensor or appliance module is designed similarly with identical Microcontroller, Wi-Fi and ZigBee modules, as depicted in Figure 6. As a result, they can interchange the roles of Master and Slave, and thereby effectively lessen the responsibility of the Master Hub. If the Master Hub fails, our algorithm will set a different module as Master Hub, which solves the problem of Central Hub architecture in earlier chapter.

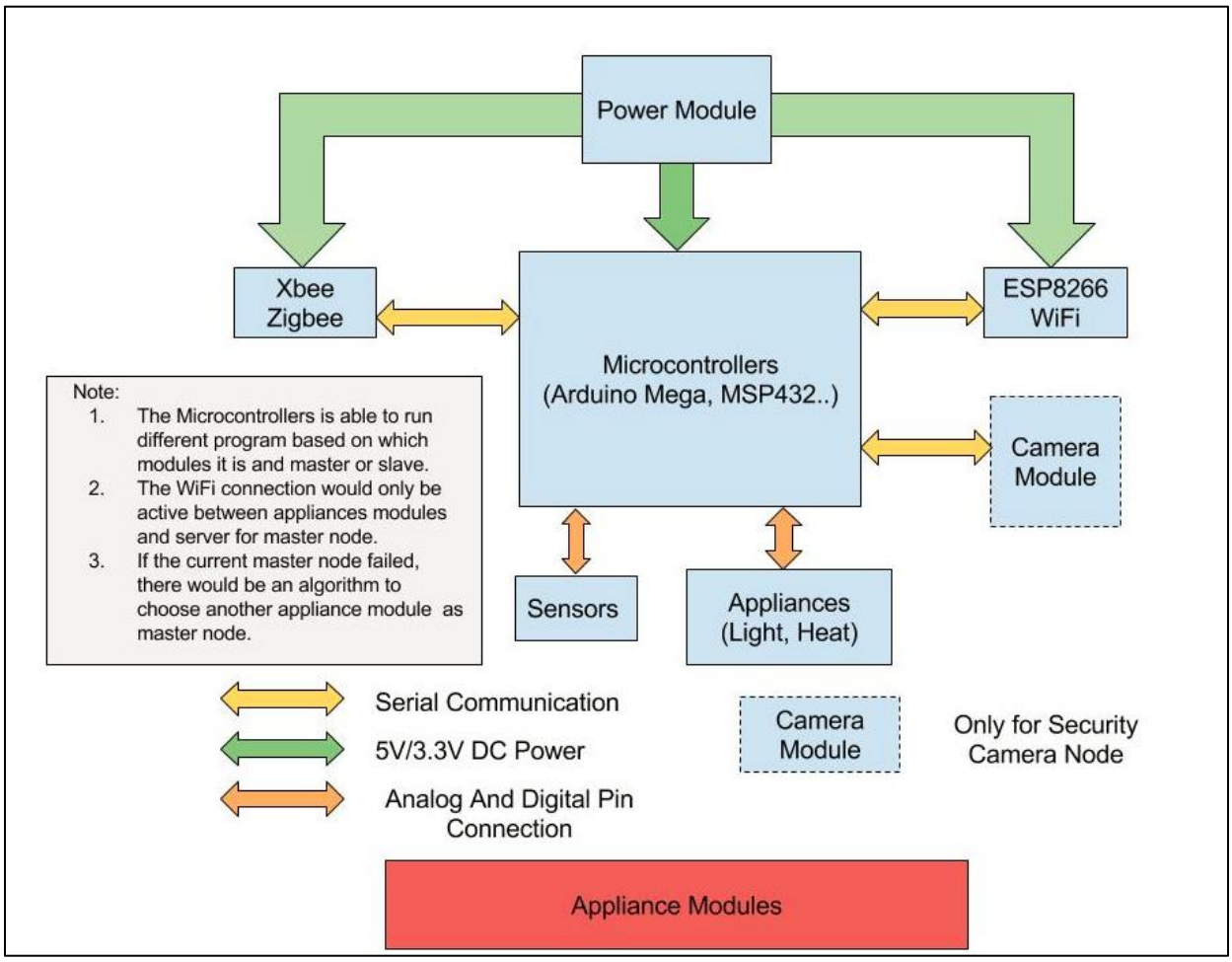

Figure 6: Individual Module Proposed Design. Each sensor/appliance module consists of ZigBee module, Wi-Fi module, Sensors and different appliances modules. They can interchange the roles of Master and Slave, which makes the system more durable.

As a result, the proposed system removes the drawbacks of existing architectures. The proposed architecture directly addresses lack of combined operations and user interfaces from Individual Modules architecture because users only need one user interface (can be web-based/smart-device-based) to fully control every appliance. The separation of Sensor Area Network and Central Hub - Server communication protocols mitigate communication traffic and data rate competitions between modules. On the other hand, the overwhelming responsibilities of Central Hub are reduced with interchangeable Master/Slave modules. Central Hub might fail, but

the system will still operate since a different appliance module will take over the Central Hub responsibilities.

2. Objectives

2.1. Customer Requirements

From our market research, we obtained a number of requirements that the customers would like to see in an IoT product, as well as a few implicit requirements that are not typically mentioned. The explicit requirements are as the following:

- Energy saving
- Durable
- Automatic
- Reliable
- Environment Friendly
- Long life span

Energy saving is required because our system was designed and created to address the energy efficiency issue. Therefore, this must be our first requirement for the project. Also, as our system will be used both in an indoor and outdoor environment, it has to sustain harsh weather conditions, including humidity, rain, storm, snow and hot weather. Therefore, our product must be **durable.** As a smart home system, what the customers would expect is not only energy-efficient system, but it also has to be **automatic** and **reliable** in operation, which would give customers a convenient and easy-to-manage their home as well as ensure the customers that the system always operates. Furthermore, due to the development of technology and the lack of natural resources, it is imperative that products are **environment friendly**. A **long lifespan** of the system would reduce the cost of repair and replacement in the long term.

Besides the requirements from research and survey, there also exist some implicit requirements that the system should have to attract customers:

- Aesthetic
- Versatility
- Low Costs
- Safety for user

Obviously, the design of our system must be appealing to users and, therefore **aesthetic** is one of the most important customer requirements. As it is designed for customers' homes, the system should be designed with a focus not only on its functionalities but also its appearances, since it becomes part of a home. The system must have multiple functions, including but not limited to motion detection, temperature sensing, automatic camera, ambient light sensing. For that reason, **Versatility** is also an important feature for our Smart Home system. Currently, the existing Smart Home products in the market are at a high price that customers can rarely purchase. We would like our system is affordable and reachable to every home and to global customers. Last but not least, the system's **safety** must be guaranteed to the users so that it could be able to withstand the severe conditions, both indoor and outdoor environments, and do not pose a threat to the users.

2.2. Product Requirements

From our market research, discussion of existing solutions, and customer requirements described in section 2.1, we identified some general necessities for the prototype as an entire system and for each component, respectively.

For the entire system, it should meet the following three criteria:

- **Low Power Consumption**: The system must be power-efficient when operates, such that its power use must be comparable to or even lower than existing smart home solution in the market and traditional homes. This requirement is also the criteria that every existing smart home solution system targeting to achieve. Our prototype will be scaled to 1/20 of the real system. Assuming a real-life system consists only of one 10W light bulb, a floor fan, and a 50W security camera. Based on data Ref [8], this system will consume roughly 46kWh per month (assuming camera is always on), which is about 1.5kWh per day. Therefore, the power consumption limit target for our prototype must be equal to or less than 75Wh.
- **User-friendly interface on Wi-Fi connected devices**: This is a very important requirement. Users must find it easy and convenient to monitor and control their smart home system remotely using their PCs or smartphone. The user interface must be straightforward and easy to use for users with no technical skills or knowledge, and must be globally accessible and controllable.
- **Accessibility**: The Smart Home System Prototype must be able to be accessed and controlled remotely anytime and in any conditions. Algorithms must be developed to monitor connectivity of the system and to prevent worst-case scenario (connection dropped, hardware malfunction…) so that users would not lose control of their smart home. Every command issued by the users must be performed in real-time by the targeted devices. Data from the system must be updated frequently to the web server.

In particular, for the convenience and user friendliness, each component in our system has to be straightforward implementation, low power consumption, affordable price, and exemplary high quality, as discussed in more details below.

- **Plug and Play**: Since the time constraint of our projects, we focus on components that would require less set-up time and lengthy customization. We prefer choosing components and technology that would have more available resources, such as example projects, library, academic and technical documents related, so we can have enough information and resources to work on them.
- **Price:** The price of the components should not be too expensive since we have imposed a budget constraint of 150 USD/person for the project.
- **Quality:** The chosen components must come from established brands/vendors on the market, and their features must match what we need and required for our project.

2.3. Project Goals

For our project, we design and implement a fully functional prototype that contains various features of a smart home system. For low power consumption purposes, we need to use low-power devices, including microcontroller, sensors, and communication protocol as well as utilize the energy efficiently. The system has to be automatic and support wireless applications. It is also easy to use, and requires simple but elegant and attractive user interface, which is reliable and provides real time notifications. Each module of the system will be designed and packaged carefully for deliverable purposes. At the end of the project, we will build a home prototype to set up our demo Smart Home system.

3. Design Approaches and Solutions

In this section, we discuss in detail our proposed solutions, including the communication protocols, the microcontrollers, sensors and security cameras.

3.1. Short-range Communication Protocols for Home Automation

There are many options for short-range communication of IoT system. This section will go through some well-known communication technologies and several new emerging networking options. The choice of one or combination of technologies depends on application requirements, such as range, security, data rate, and battery life.

3.1.1. Bluetooth Low Energy

Bluetooth Low Energy (BLE), or Bluetooth SMART, is a version of Bluetooth technology operates at 2.4GHz that focuses on reducing energy consumption rather than increasing data throughput. Classic Bluetooth problems lie in fast battery draining and frequent loss of connection, which requires frequent pairing (requires large power consumption). BLE overcomes these obstacles by sending small chunks of data when necessary, putting the connection into "sleep mode" during idle periods, reducing necessity of re-pairing so it overall reduces power consumption but it still maintains 1 Megabytes per second (Mbps) data rate. Therefore, it has advantage in personal device areas. [10]

However, BLE is not designed for file transfer so data rate is one of its drawbacks to its competitors. On the other hand, BLE devices need custom gateway to translate information to a format that can be transmitted over IP. Thus, adding more devices over time becomes cumbersome. Figure 7 shows the popularity of Bluetooth in various applications.

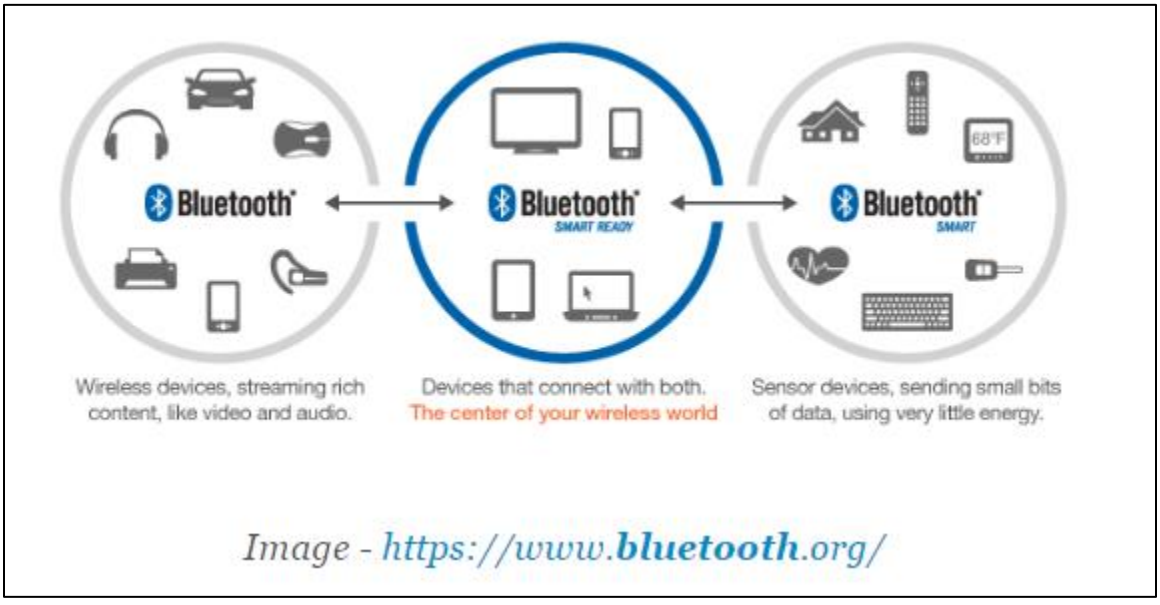

Figure 7: Bluetooth. Bluetooth Smart Ready device is the center of the system, which connecting both normal Bluetooth devices and Bluetooth Smart devices (sensors, using small bits of data, using little energy).

Bluetooth 4.2, the newest version of Bluetooth, enables BLE devices to access the Internet directly via 6LoWPAN connectivity (more about 6LoWPAN will be discussed in latter

section). Bluetooth 4.2 also builds upon government-grade security features and increases data transfer rate to 2.5 times faster than previous version. Mesh networking is also being developed and hope to be release in the second half of 2016, according to Bluetooth Special Interest Group (SIG). Nevertheless, it was release in 2014 so there is limited amount of “plug and play” modules for our integration purpose.

3.1.2. ZigBee Communication

ZigBee is a wireless mesh network protocol operating at 2.4 GHz, which targets low data rates over restricted area application such as in a home. It is based on the IEEE 802.15.4 protocol. ZigBee addresses the needs of device-to-device communication, the foundation of IoT. It runs on a mesh topology network. Nodes are interconnected with other nodes, which enables multiple pathways connect each node in a mesh network (see Fig 8). Connection paths between nodes are dynamically updated and optimized. Therefore, ZigBee system is still stable if nodes leave the network. Furthermore, it is low cost and, essentially, low power consumption, which is suitable for sensors and battery-operated devices. [11]

Some technical specifications of the protocol are mentioned below and Figure 8 demonstrates the Mesh Network architecture of ZigBee:

- Range: 10-100m
- Data Rates: 250kbps

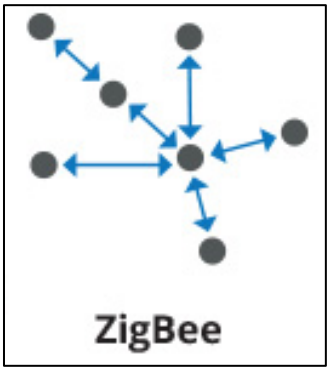

Figure 8: Mesh network. [12]

However, ZigBee devices frequently have difficulty communicating with those made by other manufacturers, which is not the best option for seamless interoperability. ZigBee is local area network (LAN), so it cannot connect devices directly to External Server (eventually to user) without the help of Ethernet or Wi-Fi. On the other hand, there are numerous ZigBee development boards with low price. Development libraries for ZigBee are also well developed, as the technology has existed over 15 years.

3.1.3. Z-Wave Communication

Z-Wave is another wireless networking protocol. Z-Wave operates in sub-GHz band of 908.42 MHz in the US, which improves range, reliability, and provides less interference from Wi-Fi and other wireless technologies in the 2.4 GHz range such as Bluetooth and ZigBee. It supports full mesh networks without necessity of coordinator node and can control up to 232 devices. It also uses simple protocol, which provides faster and simpler development. The technology also consumes extremely low power. [13]

However, Z-Wave has limited data rates of 100kbit/s and Sigma Designs is the only chip maker compared to faster data rates and multiple sources for other wireless technologies. The Source Routing Algorithm makes it difficult to manage mobility because rediscover neighbors is a complicated and power-consuming process. Technical specifications include:

- Range: 30m
- Data Rates: 9.6/40/100kbit/s

3.1.4. IPv6 over Low-power Wireless Personal Area Networks (6LowPAN)

6LowPAN is an Internet-Protocol (IP) technology, which is an acronym for IPv6 over Low-power Wireless Personal Area Networks. The technology is free of frequency bands and physical layers, and as a result it can be used across multiple communication platforms, as well as with devices that use other IP network links (Ethernet or Wi-Fi) with simple bridge device. IPv6 is the latter version of IPv4, which provides 2^{128} IP addresses. It supports both star network and mesh network. IPv4 is the first version of Internet Protocol, which limits the address space to 2^{32} addresses and by 2020, which 50 billion connected devices, we would run out of IPv4 addresses. Therefore, the need to shift to IPv6 is crucial. [14]

6LoWPAN is very attractive since it is IP-based, but support for 6LoWPAN is limited since it is recently developed for IoT. Development boards for 6LoWPAN are also expensive, ranging from $150-$500.

3.1.5. Wi-Fi Communication

Wi-Fi is a wireless technology well known for connecting devices to WLAN (wireless LAN) network, mainly using 2.4GHz and 5GHz bands. It is an obvious choice for IoT because of its pervasiveness of Wi-Fi within the home environments, and fast data transfer rate. It is also suitable for mesh and star network architecture.

However, Wi-Fi has some drawbacks, including interference and bandwidth issues. In a system full of Wi-Fi-connected gadgets, devices using Wi-Fi will need to compete for bandwidth and may be slower to respond. Wi-Fi is also power hungry which makes it inappropriate for smart devices such as sensors. On the other hand, a Wi-Fi connected mesh proves to be power-consuming for many IoT applications. Moreover, the number of devices connected to a Wi-Fi access point is limited. To solve these problems, Wi-Fi Alliance, the organization that dictates Wi-Fi standards, is developing Wi-Fi HaLow, which promises double the range of standard 2.4 GHz Wi-Fi, better penetrates obstacles, and most importantly, consumes less power. Unfortunately, it will take until 2018 for the Wi-Fi Alliance to begin certifying HaLow products. Therefore, implementing only Wi-Fi for our mesh network is not viable. Specifications for Wi-Fi include: [15]

- Frequencies: 2.4GHz and 5GHz bands
- Range: Approximately 50m
- Data Rates:150-200Mbps

3.1.6. Summary Short-range Communication Protocol

As shown in above characteristics of popular short-range communication protocols, Zigbee stands out as a user friendly, low-power, and reasonable range and transmission rate. Zigbee modules are more available and easier to integrate than 6LoWPAN and BLE (more development board options and libraries), since the latter two are being developed in recent year.

On the other hand, Zigbee not only has higher data rate than Z-Wave but it also avoids bandwidth consumption problem of Wi-Fi, so users do not have to experience slow Internet access because of their Smart Home System.

3.2. Microcontroller Modules

Since each appliance module in our system needs to communicate with several sensors and wirelessly send and receive data from the server, a microcontroller is required at each module. Based on our system architecture, we would analyze different requirements for our microcontroller.

Because we plan to use both Wi-Fi and ZigBee protocol for each node, and the camera module may require serial communication, our microcontrollers at each node must have at least 3 serial communication ports and serial communication interfaces.

Since every complex operation in our system would be performed at the web server side, a microcontroller is not required to have very powerful computational powers. Nevertheless, since the master node would be in charge of receiving every command from server and distributed to slave nodes, and receiving data from slave nodes and send back to server, therefore our microcontrollers must be capable of running multi-task or supporting internal/external software interrupts to operate in real-time, so that distributing data at the master node will not affect other operations.

We want to have algorithm that when the master node failed, the system would automatically pick a slave node to be the new master node. In order to do that, the microcontrollers must be able to store and switch between different programs. This would require large enough memory for the microcontroller to store multiple programs. Large memory is also required for security camera node, since this node would need to store and send high definition image.

Each microcontroller must have a large number of I/O pins for complex sensors and appliances system. Moreover, it would be preferred if the microcontrollers have low power consumption. One of the basic requirements of the IOT system is to help reduce power consumption for households.

For our project, and since we are only attempting to produce a prototype, not actual commercial product, we need to have a development microcontroller board available to reduce mechanical tasks so that we could solely focus on the system architecture and application.

From the above requirements, we looked at some different options for microcontrollers. There are several different microcontrollers that are popular in the current smart home industry, as well as common and well-known microcontrollers that are very versatile and can be used in many different applications. We chose to look closely at three development boards that represent 3 most popular brands of microcontrollers that we think could satisfy our requirements.

3.2.1. Arduino Mega 2560 Microcontroller Board

Arduino Mega 2560 is a microcontroller board based on the ATmega2560. This microcontroller board is a new branch of the well-known Arduino family that is designed for more complex projects. With 54 digital I/O pins, 16 analog input pins and 4 UARTs, this board has more-than-enough communication pins to accommodate multiple sensors and wireless board

that are required in our system. Arduino Mega 2560 has a RAM memory of 8KB and with the clock speed of 16MHz, provides us with a decent computational power for the nodes of our system. Flash size of 256 KB also gives us enough memory space to store multiple programs, which is also one of our requirements for the microcontroller.

However, there are three limitations of the Arduino Mega 2560 that we need to consider more carefully. The first problem, which is also an inherent problem of every Arduino development board, is its power consumption. The board required a very high input voltage of 7-12V if powered through I/O pin, and 5V if power through micro-USB connection. The board also draws high input currents, with 25mA of current in normal operation modes. Although the board could be coded to operate in low power mode when not in use, however its power consumption is still enormous in compare with other high performance but low power microcontrollers.

The second problem with the Arduino Mega 2560 is its relatively high price in comparison with other microcontrollers. The price of 45.95 USD is very high for a microcontroller, if we compare with TI MSP series, which have price of lower than 20 USD, or Raspberry Pi series, which are single-board small size computers but have the price of around 40-50USD.

The third issue, which is also the most important one that we need to consider, is that the Arduino Mega 2560 does not support multi-tasking, which mean we could not run 2 different programs at the same time on the Arduino Mega 2560. Although this issue could be resolved by using interrupts and external switch/timer to changing continuously back and forth between programs, when the system get more complex as more node involved, real-time operation could be a really challenging issue.

Despite all of these disadvantages, Arduino Mega 2560 still has incomparable advantages. It has one of the most user-friendly development environments, Arduino IDE, with a very high-level programming language, Arduino language and thousands of open-source libraries that are available to developers. Moreover, almost every sensors and modules are Arduino compatible, with an Arduino library associated, that would help coding for each node much easier for our project and give us more time to concentrate on the software and application side of the project. Figure 9 below is the image of the Arduino Mega 2560 and Genuino Mega 2560 microcontroller board.

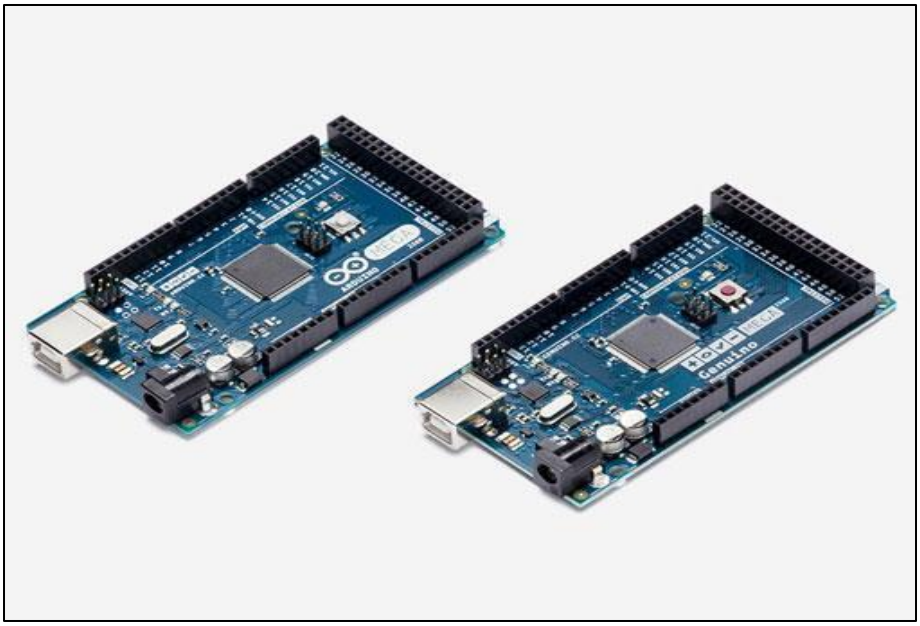

Figure 9: Arduino Mega and Genuino Mega board. [16]

3.2.2. TI MSP432 Microcontroller Board

The MSP432 is a mixed-signal microcontroller family from Texas Instrument, based on 32-bit low power ARM Cortex-M4F CPU chip. With 48 digital I/O pins, 24 analog input pins, and 4 UARTs, the board could accommodate our very complex node system with multiple sensors, appliances and wireless modules. It also offers decent computational power, with clock speed of 48 MHz and 64KB RAM. Flash size of 256 KB also allows storing multiple programs, which is an important requirement for microcontrollers in our system.

The biggest advantage of the TI MSP432 development board is that, it offers ultra-low power consumption for its high performance. Its required power input for micro USB connection, which mean 5V input voltage, but draw ultra-low current with only 95uA/MHz in normal operation mode. This advantage provides us more flexibility in designing our system, since we can power the TI MSP432 easily with batteries instead of 110V- AC main electricity.

The second advantage of the TI MSP432 is that the board is very price-competitive while offering very good computational power. It offers higher clock speed than the Arduino Mega, with 8 times the RAM size, with one-third the price of the Arduino Mega 2560. The TI MSP432 Launchpad is only price at 13 USD, making it a very appealing option for our prototype and also for any actual commercial products.

The problem with the TI MSP432 is that it is difficult to use. Currently, the most common way to configure and programmed a TI MSP432 Launchpad is by using embedded C language in Code Composer Studio IDE. C language is not as high-level as Arduino, and for the TI MSP432, users have to care about ADCs configuration, UARTs configuration… while these things can be done automatically or have well-defined support libraries in Arduino. The TI MSP432 also suffers from lacking of multi-tasking ability, as also an issue in Arduino Mega 2560. Too much I/O communications that required multiple programs running at the same time may cause serious

real-time issues for the TI MSP432 Launchpad. Figure 10 demonstrate the TI MSP432 LaunchPad from Texas Instrument.

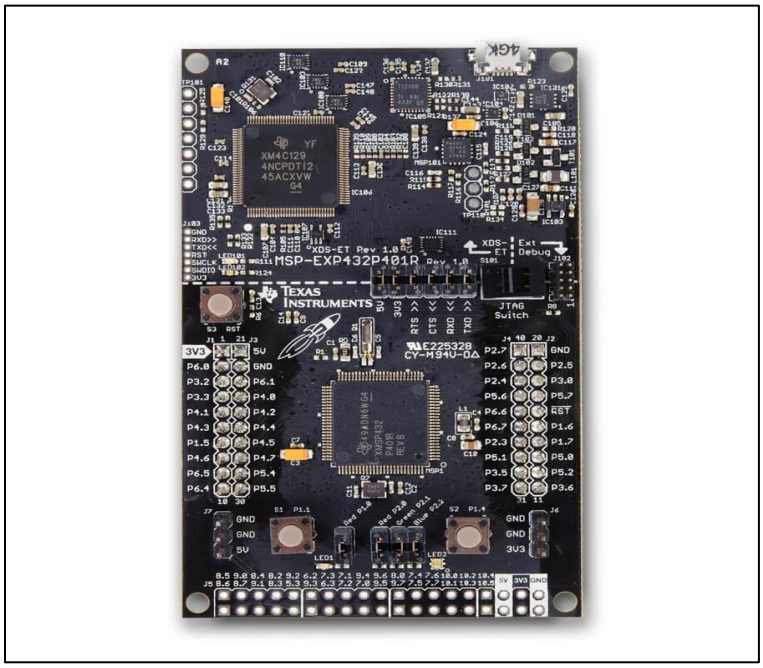

Figure 10: TI MSP432 LaunchPad. [17]

3.2.3. Raspberry Pi Board Computer

The Raspberry Pi is a series of credit card-sized single board computer. All Raspberry Pi model features a Broadcom System on a Chip (SoC), which include an ARM compatible CPU and an on-chip GPU. Since Raspberry Pi is actually not a microcontroller, but a small-size and fully functional computer, it offers very good computational powers. With CPU speed ranges from 700 MHz to 1.2 GHz, and on-board RAM of 1GB, the Raspberry Pi is powerful enough for every complex control operation that would be required at our nodes. Though it doesn't have on-board Flash Memory, an external SD card could be used instead with the capability from 2-16GB, allows us to store multiple programs and even images from security camera module if needed. The Raspberry Pi could also load and run Linux Operating System from microSD card. The Linux OS allows the Raspberry Pi to do multi-tasking, which make it stands out from our 2 previous options and provide a lot of flexibility and easiness when dealing with the real-time requirement of our system. Raspberry Pi also offers 40 general purpose GPIO pins to accommodate multiple sensors and appliances that required in our products. Moreover, the new version of Raspberry Pi 3 Model B provides built-in Wi-Fi, which could help us reduce a lot of works and costs buying and installing a separate Wi-Fi module to our microcontrollers. It is also very price competitive, with only 35 USD. The Raspberry Pi only have one UART port, however this could be worked around since Wi-Fi is built-in for the Raspberry Pi 3 Model B, we only need the UART port for ZigBee module.

Despite all of these advantages, The Raspberry Pi suffered from too high power consumption. With 5V input voltage and at least 2.4A current draw, this is impossible for the

board to be powered by battery but only by main electricity. Since in our system, the microcontrollers may have to run continuously 24/7, the high power consumption of the Raspberry Pi will add up to the electricity cost of the system. Also, too high computational power may be a waste, since our application may never exploit all of the board computational power potentials. Figure 11 shows the newest version of Raspberry Pi, Raspberry Pi 3 Model B.

Figure 11: Raspberry Pi Model B. [18]

3.2.4. Summary Microcontroller Module

Based on these analyses, we developed Table 3 which summarizes key technical features of three different types of microcontroller that we discussed. In the table, several interested criteria are listed and compared, to figure out the best option for our system.

Table 3: Comparison between different Microcontrollers.

	Arduino Mega	**TI MSP432**	**Raspberry Pi 3 B Model**
Price	45.95 USD (from Adafruit websites)	12.99 USD (from Texas Instrument)	35.00 USD (from Mouser)
RAM Memory	8KB	64KB	1GB
Clock Speed	16MHz	48MHz	700MHz
UARTs serial communication ports	4	4	1
Multitasking	None	None	Yes
Internal/External Interrupts	Yes	Yes	Yes
Flash	256KB	256KB	SD card 2-16GB
Input Voltage	5V (from USB)	5V	5V
Power Consumption	25mA (normal mode)	4.5mA (normal mode)	2.4A
On-Board Network	None	None	Built-in Wi-Fi, Ethernet Connection, Bluetooth Low Energy
Integrated Development Environment	Arduino IDE	Code Composer Studio Energia IDE	Scratch, IDLE, anything with LINUX supports

From the Table 4 and discussion, we decided to use the Arduino Mega board for our system. The Arduino Mega has low power consumption advantages compared to the Raspberry Pi 3 B Model, and is more straightforward in configuration and coding in compare to the TI MSP432 board. Also, the Arduino board has proven compatibility with the ESP8266 Wi-Fi module and the Xbee ZigBee module, while the compatibility of the TI MSP432 with those modules needs to be verified and tested.

3.3. Sensor Modules

The system could not be considered as Smart Home without the implementation of sensors. For our projects, we decide to apply occupancy sensor to detect the movement and light sensor to automatically adjust the brightness in the house according to the sunlight and different time of the day.

3.3.1. Occupancy Sensor Module

As mentioned above, an occupancy sensor is used in our Smart Home system to detect the presence of people, vehicles, animals, objects to automatically adjust the appliances based on the data receiving from the sensor. When the occupancy sensor detects no people or vehicle, it will send signal to the central hub to control the system accordingly in order to save energy. We have conducted research of three different types of occupancy sensors.

The first type of occupancy sensor that we looked at is the PIR (Passive Infrared) Motion Sensor. PIR Motion Sensor measures infrared (IR) light radiating from objects in its field of view, to detect motions of people and objects. The sensor is highly reliable and resilient to false triggering. The sensor has a very low price and requires very little power of operation. However, it also has several disadvantages. The sensor is vulnerable to "dead spots", which are the places that the sensor cannot detect motion although still in the detection range. Also, the sensors couldn't detect motions behind obstacles. Figure 12 shows the module of PIR sensor.

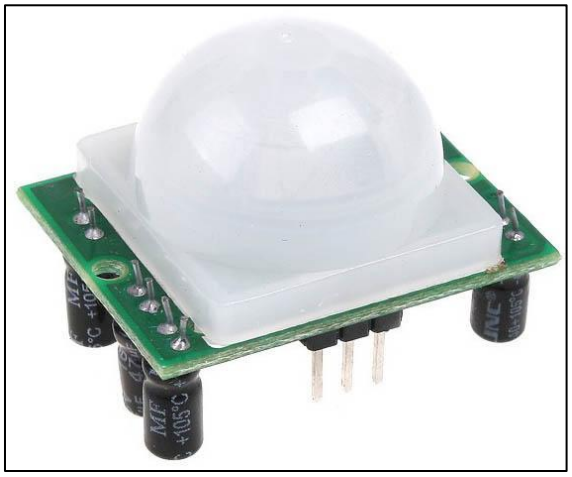

Figure 12: PIR sensor. [19]

The second type of occupancy sensor is the UltraSonic Motion Sensor. UltraSonic Motion Sensor can detect movement of people or objects within a limited area. It senses motion by analyzing sound waves in its environment like the way bats or dolphins do. This could be used in our product to help detect people or vehicle moving around, and send signal to the microcontroller to brighter/dimmer the LED. Typical products for UltraSonic Motion Sensor have a very large detection range, around 6-7m in radius, which we valued most for occupancy sensor. The price of the sensor is relatively cheap, usually less than 4 USD. The sensor can even detect motion through obstacles. However, its main drawback is that it sensitivity could be affected by loud noise, which would affect its sound wave analyzing. Also, changes in environment, such as temperature, humidity, or air particles could lead to false triggering. Ultrasonic cannot distinguish between human and non-human motion and may interfere with

other devices due to high intensity sound waves. Figure 13 demonstrates the module of UltraSonic Motion Sensor.

Figure 13: HC-SR04 module for UltraSonic Motion Sensor. [20]

Lastly, the third type that we have researched is the Microphonics Sensor found by Sensor Switch, an Acuity Brands company, which is located in Wallingford, Connecticut. Microphonics utilizes a microphone inside sensor to hear sounds that indicate occupancy. Microphonics technology is especially useful in obstructions. One of the great benefits of Microphonics technology is its ability to distinguish sounds made by human activity from ambient noise. This technology uses automatic gain control (AGC) to dynamically self-adapt the sensor to the environment by filtering out constant background noise. Moreover, it uses advanced digital acoustic filtering that prevents the prolonged presence of varying noise such as television or radio from keeping the light on unnecessarily. It can also filter out periodic sounds like clock ticking. However, since this advanced technology is new and is owned by Acuity Brands Company, we were not able to find any available module in the market at this time.

Due to the pros and cons of each type of occupancy sensors discussed above, we decided to apply both technologies, PIR and Ultrasonic Motion Sensor. This Dual Technology will provide the more reliable results, preventing from false tripping, reducing interference and saving more energy. Table 4 highlights a comparison between different motion detection technologies.

Table 4: Comparisons of different Motion Detection Technologies.

	PIR	**Ultrasonic**	**Dual-tech (PIR + Ultrasonic)**
Power Supply	5-16V, 50µA	5V DC, 15mA	5V
Range (maximum)	7 meters 120-degree cone	2cm – 400cm 30 degree	2 – 700cm 120 degree
Output	Digital signal output of 3.3V high/ 0V low	Digital signal output	Digital signal output
Dimension	32mm x 24mm	45mm x 20mm x 15mm	
Pros	▪ Sense the difference between heat emitted by moving people and background heat ▪ Suitable for enclosed space	▪ Send out the high frequency ultrasonic waves into a space and measure the reflected pattern ▪ Detect motion behind objects	▪ More reliable system ▪ Better accuracy
Cons	Cannot detect motion behind obstacles ➔ Need line of sight	▪ May interfere with other devices due to high intensity sound waves ▪ Accuracy of results is affected by ambient loud noise, temperature, humidity	More challenging in implementation
Price	$9.95 (Adafruit)	$5.00 (Amazon)	$14.95

According to the comparison above, we decide to use both technologies for our project since we greatly consider the quality of our prototype and both sensors have reasonable prices.

3.3.2. Temperature Sensor Module

The temperature sensor is utilized for the measurement of indoor temperature to support different applications such as control of the temperature in the house, and detection overheating appliances. The temperature data is sent to the server and displayed in real-time. Also, the microcontroller, after receiving data from the temperature sensor, decides either turning on/off the fan and heating systems or switching to critical mode and warn users about overheating situation. There exist many types of temperature sensors available in the market, categorizing into mechanical temperature sensors (thermometer), electrical temperature sensors (thermistor) and integrated circuit (IC) sensors.

The mechanical temperature sensor, the thermometer in particular, is a device containing a sensor made of mercury and some means of converting the physical change into a numerical

value. It is widely used in industry to control and regulate processes, in study of weather, in medicine and in scientific research. This device has existed in the early 17th century and is still being used in today applications. However, due to the relatively large size and the incapability of communicating to microcontroller, we decided not to use the mechanical temperature sensor for our system. Figure 14 depicts a sample of a mercury thermometer.

Figure 14: Mercury thermometer for the measurement of room temperature. [33]

Another type of sensor is the electrical temperature sensor, the thermistor. It is a type of resistor whose resistor is dependent on temperature, much more responsive and sensitive to temperature than the standard resistors. This type of device was first discovered by Michael Faraday in 1883, although commercially useful thermistors were not manufactured until 1930. Although the thermistor provides a high degree of accuracy, it is not easy to configure the thermistor with the microcontroller, whose responsibility was to receive the temperature data for further process and analysis. Figure 15 shows a thermistor.

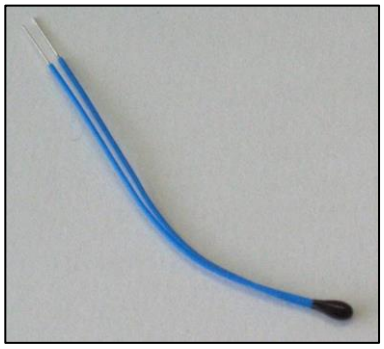

Figure 15: Thermistor. [34]

The temperature sensor that is the most compatible for our project, however, is the IC sensor, which can be easily integrated with the microcontroller in our system. We decide to choose the analog-output temperature sensor from manufacturer, Analog Devices, the TMP36 due to its compatibility with Arduino, wide range, low power consumption and precision centigrade. The sensor provides a voltage output that is linearly proportional to the Celsius

temperature and has accuracies of $\pm 1^{o}C$ at $25^{o}C$ and $\pm 2^{o}C$ over the $-40^{o}C$ to $+125^{o}C$ temperature range. Figure 16 demonstrates the TMP36 sensor and following is its specifications.

Figure 16: TMP36 from Analog Devices. [35]

Specifications of temperature sensor TMP36:

- Voltage input: 2.7 V to 5.5 Vdc
- $10\text{mV}/^{o}C$ scale factor
- $\pm 2^{o}C$ accuracy over temperature range
- $\pm 0.5^{o}C$ linearly
- Operating range: $-40^{o}C$ to $+125^{o}C$

3.4. Camera Module

For the security camera module, we define some criteria to narrow down our options from various existing products on the market. First, we want our camera to have at least 2MP resolution, since high definition security image will be more appealing to users and also provide comfort if we want to apply image processing algorithm (such as face recognition, character recognition, etc.) on the software side of our project.

Secondly, the camera must be compatible with our microcontroller. From the product datasheet, we must determine if our options of camera module could be used along with our microcontroller. The communication interface on the camera module must be available on our microcontroller (not being used by other components). For example, since there is only one serial UARTs port on the Raspberry Pi, which is reserved for the ZigBee module, we omit all camera options that use serial communication.

The next criterion lies in cost-effective. The price of the camera module must be reasonable, while still offers decent quality. And the camera module must also be power-efficient, for the same reason that we discussed with other components. The camera module must be powered by 3.3/5V inputs DC source, so we could use one standardized power module to power our entire system.

The other criterion is that the camera module, since it is intended to use both indoor/outdoor, it must be able to operate in severe weather condition, such as low/high temperature. In New England area, where the temperature could drop to -20 to -30 degree Celsius in the winter, the camera module must be able to sustain that.

These are the basic criteria for the camera module to build our prototype. However, for a better image quality for software-side image processing in any working conditions, we have additional criteria. Night-Vision Capability is very important for security purposes, Automatic Functions (such as automatic exposure control (AEC), automatic white balance (AWB), automatic 50/60 Hz luminance detection, automatic black level calibration (ABLC) are desirable.

From these criteria, we limit our component options into three camera modules. We will discuss about these modules along with their pros and cons, and decide on the most suitable option for our project.

3.4.1. ArduCam Mini Module 2MP

The first camera module that we looked at is the ArduCam Mini Module 2MP. The module features 2MP CMOS image sensor OV2640. This camera module also has a 5MP version using CMOS image sensor OV5642. Both of these image sensors have high sensitivity for low-light operation and including several automatic image control functions (AEC, AWB, and ABLC). The module provides user with active array size of 1600 x 1200 pixels, which is also the resolution for still images that can be captured. Its communication can be through 8 MHz SPI for camera commands and data stream, and through I2C interface for sensor configuration. The module is compatible with many platforms such as Arduino or Raspberry Pi with plenty of support libraries, and can also be connected to the Wi-Fi module ESP8266 without the needs of a microcontroller. The price of the module is also relatively reasonable, with only 25.89 USD each from eBay. Also, the output image format includes JPEG, which would be very convenient since we don't need any conversion on the software end.

However, there are several problems with this module. The first one is its power consumption. In normal operation mode, it required 5V input with 70mA current draws, which is even higher than the power consumption in normal mode of the TI MSP432 and Arduino microcontroller. The second problem is that the module using "rolling shutter" techniques, which could cause image distortion when capture moving objects. The third problem is that, although its embedded camera sensor has high sensitivity for low-light operation, the module does not have night-vision mode, which mean in other to operate in night time, we may have to incorporate an additional flash light to our microcontroller that will illuminate the objects when the camera capturing image. The last problem is the camera temperature ranges from -10 to 55 degree Celsius, which is clearly incompatible with the actual temperature in New England in winter. Figure 17 shows the module of the ArduCam Mini.

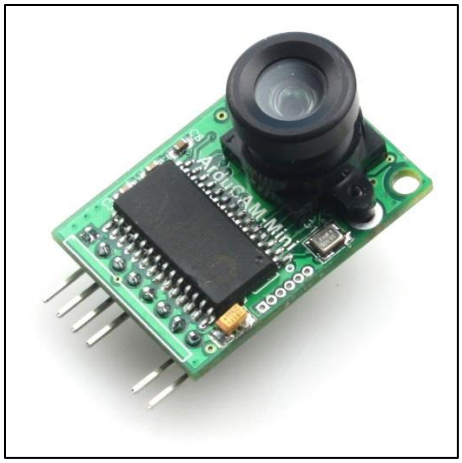

Figure 17: ArduCam Mini Module 2MP. [21]

3.4.2. Raspberry Pi NoIR Camera Board v2

The Raspberry Pi NoIR Camera Board v2 is a new, high definition camera module that is compatible with all Raspberry Pi models. The modules use Sony IMX 219 PQ CMOS Image Sensor in a fixed-focus module. It offers a very high definition of 8MP (megapixel), with still picture resolution of 3280x2464. The module could be connected to the Raspberry Pi through a 15-pin ribbon cable to the dedicated 15-pin MIPI Camera Serial Interface (CSI-2), which provide a 3.3V voltage to the camera module. The camera has high data transfer rate, with 30fps (frame-per-second) for 1080p image and 60fps for 720p image. It also has automatic image control functions, including automatic exposure control, automatic white balance, automatic band filter, automatic 50/60 Hz luminance detection and automatic black level calibration. The camera has built-in LED Flash, and no infrared filter, thus suitable for taking pictures in low light environment. The price is also very competitive, with only 29.95 USD from AdaFruit websites.

The problem with this module is that it also used “rolling shutter” techniques, which may cause image distortion if trying to capture moving objects. Moreover, we could not find any documents about the power consumption of this module, so whether the power consumption of the Raspberry Pi NoIR Camera Board v2 meet our requirements or not is still in question. Figure 18 demonstrates the camera module.

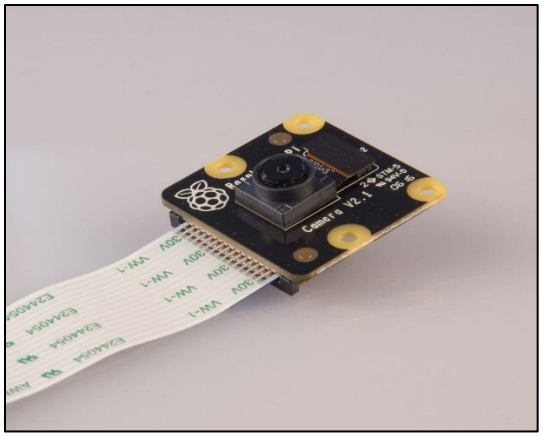

Figure 18: Raspberry Pi NoIR Camera Board V2. [22]

3.4.3. Pixy CMU Cam5 Sensors

The Pixy CMU Cam5 Sensors is a smart vision sensor object tracking camera module developed by Charmed Labs and Carnegie Mellon University. The camera features an OmniVision OV9715 camera sensor with image resolution of 1280x800 pixels. The camera module has an on-board processor with RAM and Flash, which mean we could configure the camera to perform different vision operations such as objects tracking, face recognition without overwhelming the computational power of the microcontroller. The camera communication interface includes UART serial, SPI, I2C or USB, allowed it to communicate with many different microcontrollers such as Arduino or Raspberry Pi, with plenty available support library. The camera sensor also uses an infrared filter technique, which means it can work in the low-light condition with night vision.

The problem with the Pixy CMU Cam5 Sensors module is that, because of processor and memory, its power consumption is enormous. It uses a 5V USB input with 140mA current drawn. Secondly, the only known and supported way to view an image or video from this camera is to use the developer application PixyMon. This PixyMon application also allows the users to "teach" the camera for object tracking, or face recognition. However, if we want to get the image to develop our own application, it would be harder and less documented. The final problem with this module is the price of 75 USD, which is too high. The camera module is shown in Figure 19.

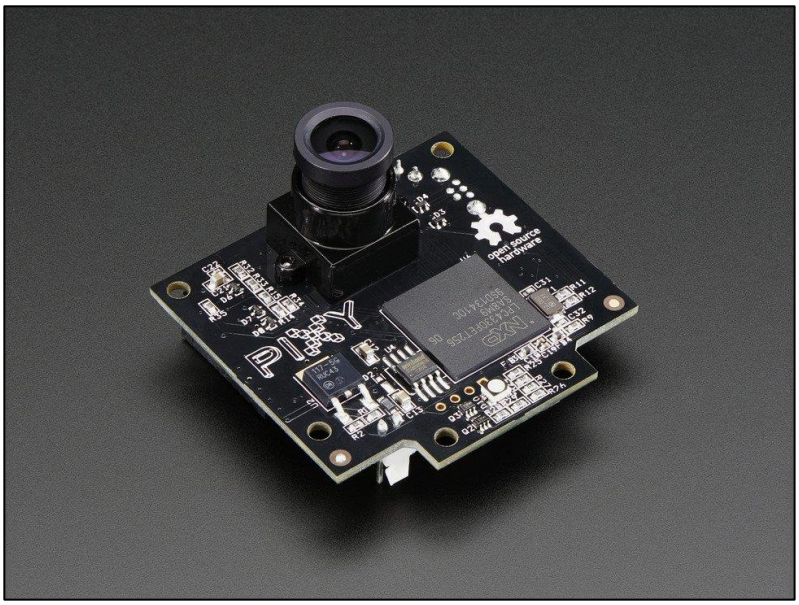

Figure 19: Pixy CMU Cam5 Sensors. [23]

3.4.4. Summary Camera Approach

To summarize the different camera modules discussed above, Table 6 compares these three cameras with respect to various performance criteria.

Table 5: Comparison of different camera modules.

	Arducam Mini Module 2MP/5MP	**Raspberry Pi NoIR Camera Board v2**	**Pixy CMU Cam5 Sensors**
Price	25.89 USD (from eBay websites)	29.95 USD (from Adafruit Websites)	75 USD (from Adafruit Websites)
RAM Memory	None	None	264KB
Power Consumption	5V/70mA	5V/NA	5V/140mA
Communication Interfaces	SPI, I2C	CSI	SPI, I2C, UART, USB
Image Sensors	OV2640/5642	Sony IMX 219 PQ	OV9715
Image Processing Speed	8MHz	1080p:30 fps 720p: 60 fps	720p: 50fps
Image Resolution	2MP/5MP 1600x1200	8MP 3280x2464	2MP 1280x800
Compatibility	Arduino, Raspberry Pi	Raspberry Pi	Arduino, Raspberry Pi
Lens field of view	NA	NA	75-degree horizontal 47-degree vertical

Since we have decided to use the Arduino Mega board for our system, we will eliminate the Raspberry Pi NoIR Camera Board v2 from our camera options, because its compatibility with the Arduino is not verified. Among the remaining 2 options, we decided to choose the Arducam Mini Module because of reasonable price, low power consumption and availability of supported documents and online tutorials.

4. Design Implementation

In this section, we discuss in detail the process of implementing different modules and functions of the system, including sensors, communication protocols, user interface and integration of the entire system. As we used the low-power consumption microcontroller Arduino, the system, except for the user interface, is implemented in Arduino IDE.

4.1. Sensor Functions

As mentioned in section 3, we utilized occupancy sensor, light sensor, temperature sensor and a camera. For each module, we implemented different function that corresponds to the designated purpose of that sensor.

4.1.1. Occupancy Sensor

An occupancy sensor is used in our Smart Home system to detect the presence of people, vehicles, animals, objects in order to automatically adjust the appliances based on the data receiving from the sensor. When the occupancy sensor does not detect people or vehicles, it will send a signal to the central hub to control the system accordingly in order to save energy. As outlined in section 3.3.1, due to the pros and cons of the PIR sensor and the Ultrasonic sensor, we decided to use both technologies for distinct applications. In particular, the PIR sensor, which measures the passive infrared light radiating from objects in its field to view, is implemented for indoor application to turn on and off the light according to the status of movement it detects. As it provides a digital signal output, we read the value from the PIR pin digitally by the following command in Arduino IDE:

```
value = digitalRead(inputPIR);
```

Based on the digital value (0 and 1) of the output pin, we decided to turn off and on the LED light correspondingly. Similar to the PIR sensor, the ultrasonic sensor also provides digital output. As it senses the motion by analyzing the sound waves in its environment like the way bats or dolphins do, we first needed to send a trigger signal via TRIG pin and wait for the ECHO pin to receive back the signal. The described procedure is represented in Figure 20.

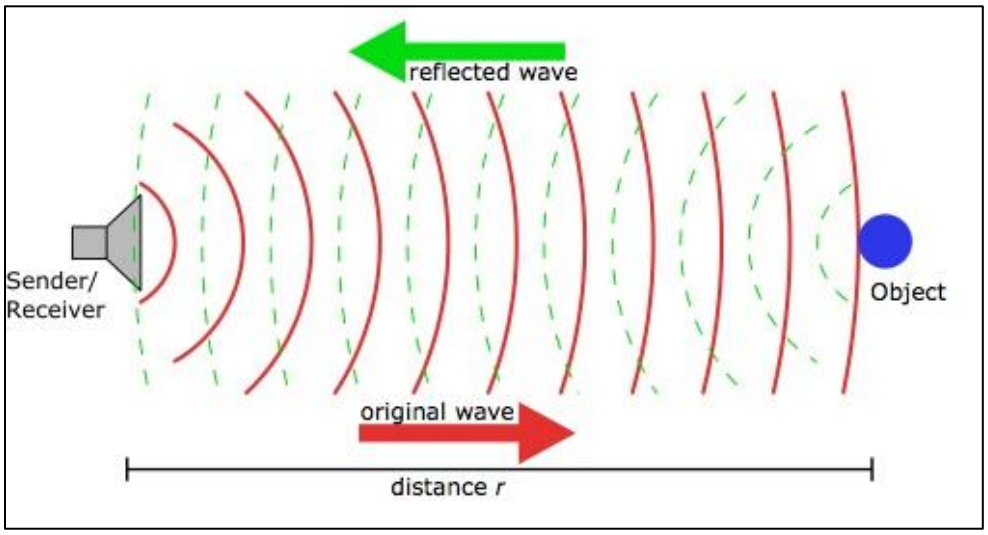

Figure 20: Ultrasonic Wave Terminology. [38]

During the time or the duration from the moment that the trigger signal leaves the sensor until the ECHO pin receives the reflected signal, the wave has travelled twice the distance between the sensor and the object of interest. The following code in Figure 21 is used to measure the duration and then calculate the distance by multiplying half of the duration with the sound velocity.

```
// Determine the time duration and calculate distance
// Duration
duration = pulseIn(echo, HIGH);   // Count the time when echo pin is set HIGH, measured in uS
// Calculate distance in cm
distance = int((duration/2)*0.034);    // d = time*velocity, which is 0.034cm/us
```

Figure 21: Implementation of Ultrasonic Sensor HC-SR04.

4.1.2. Temperature Sensor

As discussed above, the temperature sensor is used for controlling the indoor temperature by adjusting the cooling and heating system accordingly as well as detecting the critical situation of overheating and alerting the users. The IC sensor, TMP36 from Analog Devices, is relatively easy to use and is able to configure well with the Arduino microcontroller. For the sake of ease, the sensor module is shown again in Figure 22.

Figure 22: TMP 36 – IC Temperature sensor from Analog Devices to control indoor temperature and detect critical overheating situation. [40]

The sensor has three pins: the leftmost pin is GROUND, the pin in the middle is Vout and the rightmost pin is connected to Vcc. After pairing and configuring the device with the Arduino microcontroller, the following code demonstrates how to read the voltage output of the TMP36 and then convert the data into corresponding temperatures.

```
// Get the voltage reading from the temperature sensor
reading = analogRead(tempPin);    // return a number from 0 - 1023 (10 bit)

// converting the reading to voltage (for +5V)
voltage = reading*5.0/1024.0;     // 5V-scale
```

Figure 23: Implementation of temperature sensor TMP36.

The output is established in units of degree Celsius. It can also be converted into Fahrenheit degree. Based on the measured temperature, the Arduino will analyze and provide a suitable response to the situation, either uploading the data to the server or powering off appliances if the temperature is critically high and overheat is detected. This will be discussed further in section 5.3 about case studies.

4.1.3. Camera Module

In our system, when the occupancy sensor at the coordinator module is triggered, the camera module will capture an image of the objects that triggered the sensor. The camera module used in our system is the ArduCam Mini 2MP, which is easy to use, provides good quality image, and fully compatible with the Arduino development board. An image of the camera module is shown in Figure 24.

Figure 24: ArduCam Mini 2MP- Camera Module for Capturing Image Compatible with Arduino Development Board. In this image, there are only 6 pins visible, since the 5V and GND pins are female pin headers, so they are not showed in this image. [36]

ArduCam Mini 2MP module provides 2MP resolution images in RAW, YUV, RGB or JPEG format, with a maximum image size of 1600x1200. This module uses an I2C interface for sensor configuration (the embedded 2MP OV2640 image sensor) and an SPI interface for camera commands and data streams. The required power input for the ArduCam Mini 2MP camera module is 5V. For the purpose of our system, the image file format is set to JPEG, and the image file size is set to 640x480.

The ArduCam Mini 2MP has eight pins. The four SPI pins: CS, MOSI, MISO, and SCLK will be connected to the corresponding four pins of the Arduino. The GND and 5V pins of the ArduCam will be connected to the GND and 5V pins of the Arduino. The two pins SDA and SCL (I2C interface) will be connected to the corresponding SDA and SCL pins on the Arduino. A detailed explanation of Arduino code for the camera module will be provided given in section 4.5, and the connection of ArduCam camera module with Arduino is shown in Figure 25.

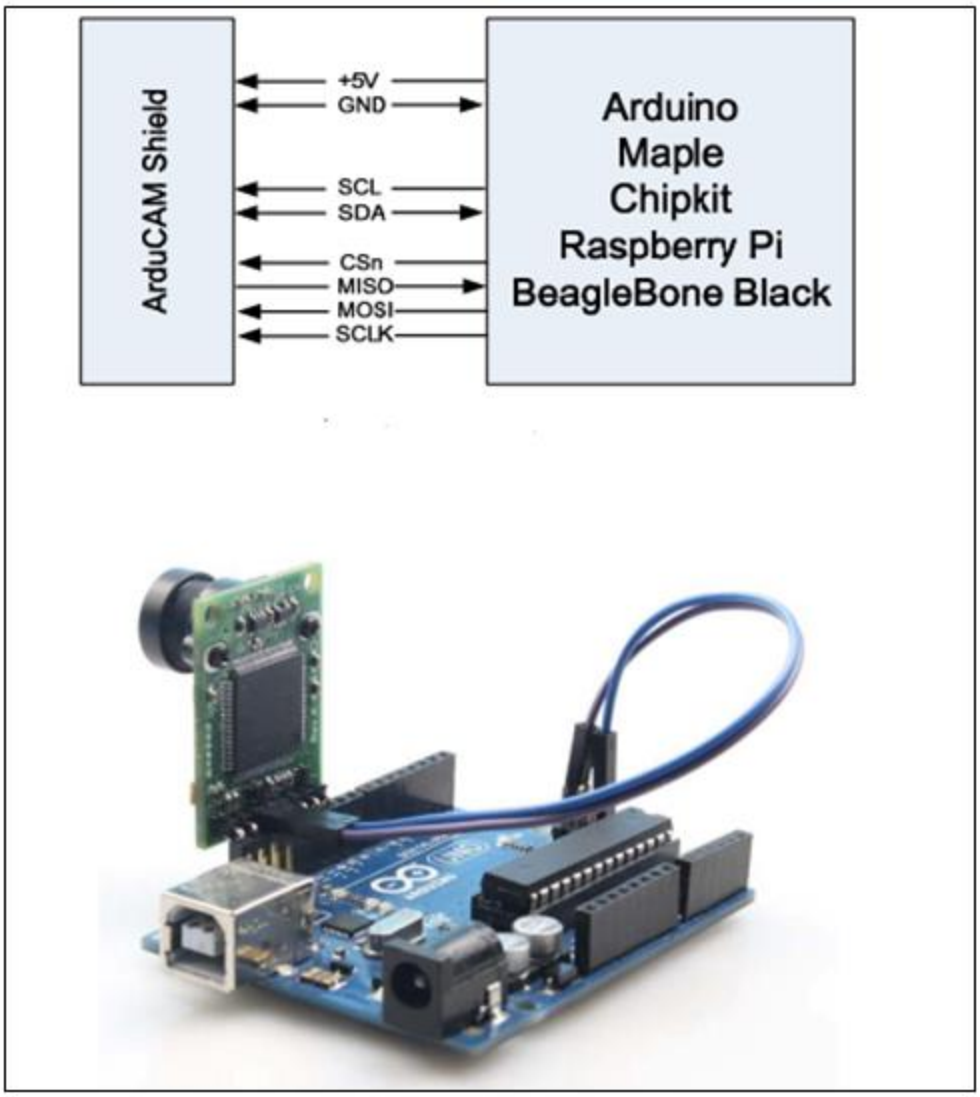

Figure 25: ArduCam Mini 2MP Connection with Arduino Module. The ArduCam Mini 2MP module is also provided with supporting library for Arduino Development Board. [36]

4.2. Local sensor-communication ZigBee Protocol

4.2.1. Zigbee Xbee Parameter Configuration

After careful consideration, we choose Zigbee Xbee (Figure 26), an RF module provide to cost-effective wireless connectivity to devices in a Zigbee mesh networks.

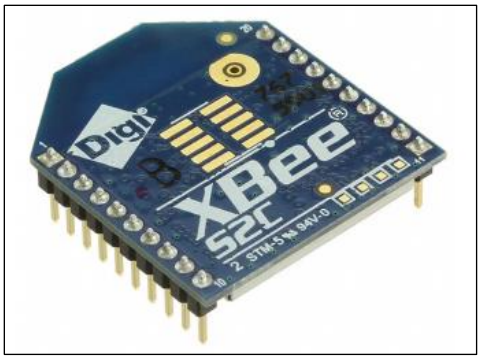

Figure 26: Zigbee Xbee Module. [24]

In order to establish Point-to-Multipoint Zigbee communication, these Xbee modules have to be configured correctly. The central hub module's Xbee has to be configured as Coordinator and operated in API (application programming interface) mode, while the other two Xbees must be configured as Routers and operated in Transparent mode. The Coordinator is the only device type that can start a Zigbee network. It is responsible for selecting channel, network ID, security policy, and stack profile for a network. Routers, on the other hand, must discover and join a valid Zigbee network. When RF data is received, the API mode helps Coordinator to know the origin of the message since it provides ID of the sender. Transparent mode, in contrast, only gives Xbee modules RF messages, but this mode is sufficient for simple operation of routers.

We change the Xbees configuration parameters by using Sparkfun Xbee Explorer USB dongle and Digi's software XCTU. The USB dongle connects Xbee module with a computer, and helps the configuration process through XCTU software. After we open XCTU software, we have to discover our Xbee module (Figure 27) by choosing the right USB Serial Port, Baud Rate, Data Bits, Parity, Stop Bits, Flow Control, and since our Xbee is a programmable module, we have to check "The radio module is programmable".

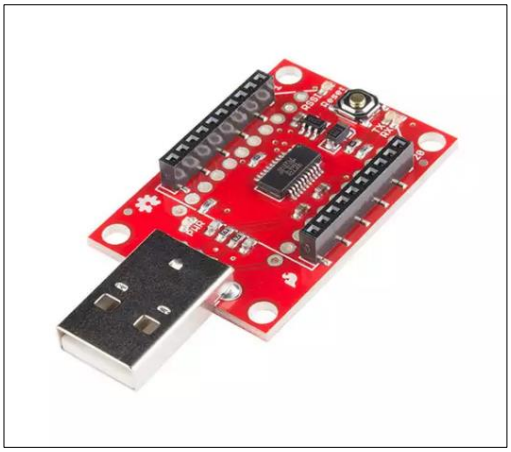

Figure 27: Sparkfun Xbee Explorer USB, which connects the Zigbee Xbee to a computer with the XCTU software. [25]

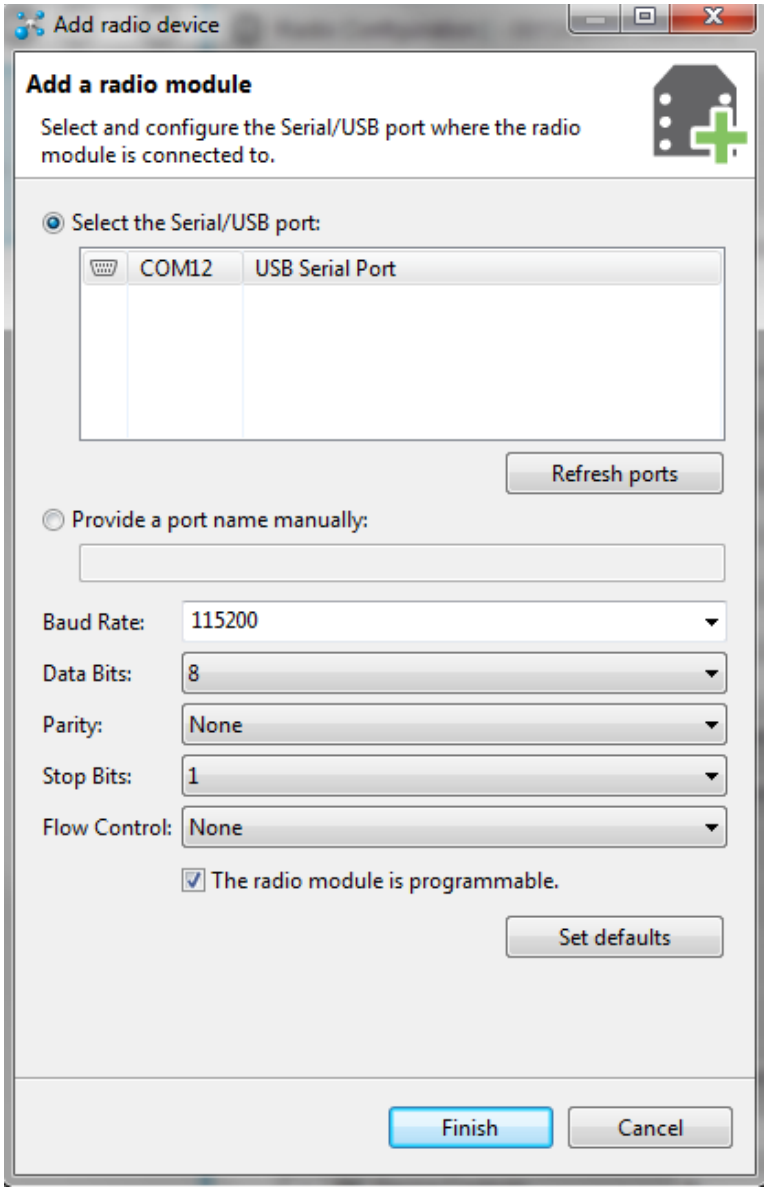

Figure 28: Discover Xbee Module Setting in XCTU user interface.

Next, if the Xbee module is successfully discovered, the software will then load the configuration parameters from the module. All modules have to use the same PAN ID (network ID), so "123" is set for all Xbee PAN ID. For the Coordinator, "CE" (Coordinator Enable) has to be set to "1".

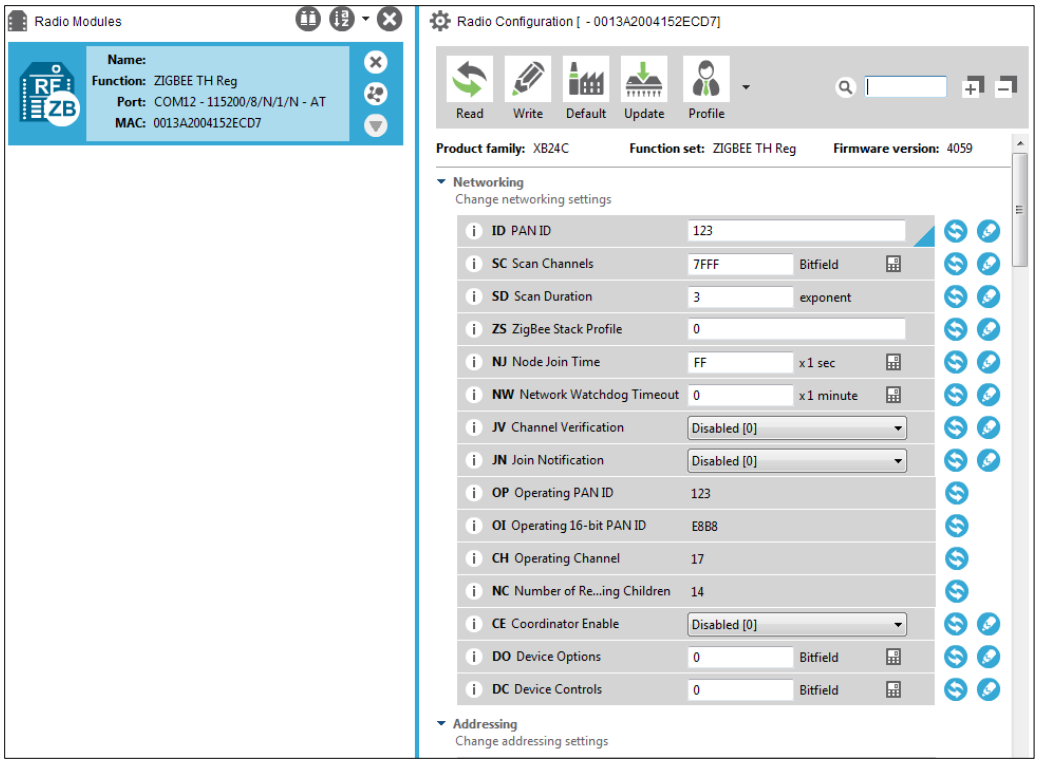

Figure 29: Changing Xbee configuration parameters in XCTU user interface.

The Coordinator Destination Address High and Low are both set to 0, which means the Coordinator module will broadcast its message to all routers in the same network. However, routers do not have broadcast functionality, so they need to send their messages to only the coordinator. The destination configuration is displayed in Figure 30.

As previously mentioned, the Coordinator has to be in API mode, while routers are in Transparent mode. These modes can be change with “AP” (API Enable) parameter (see Figure 29).

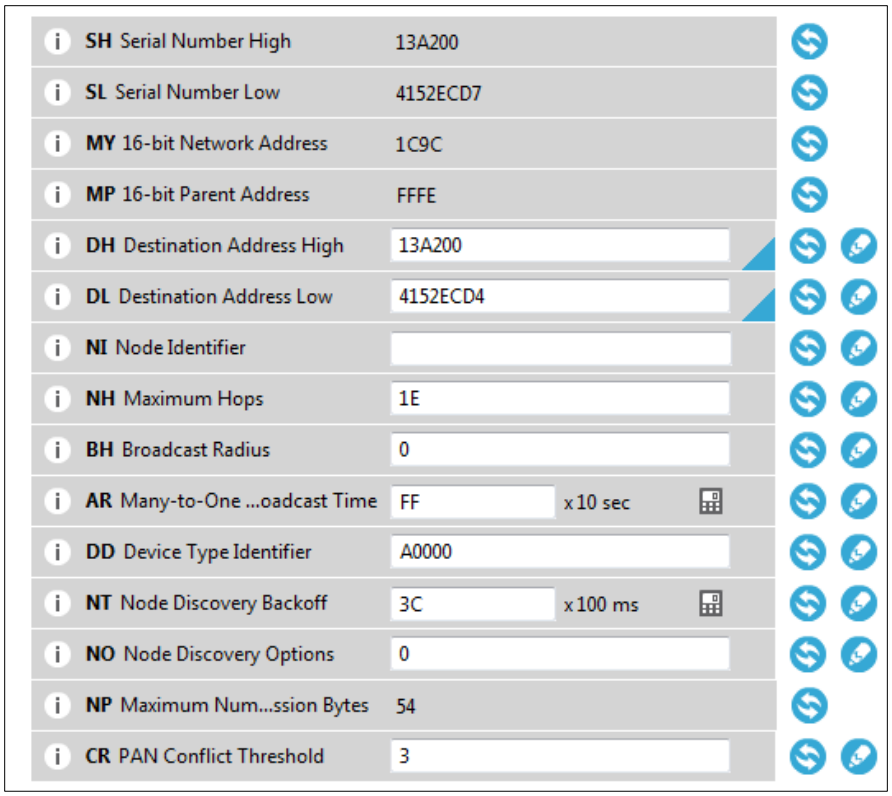

Figure 30: Changing Xbee destination parameters in XCTU.

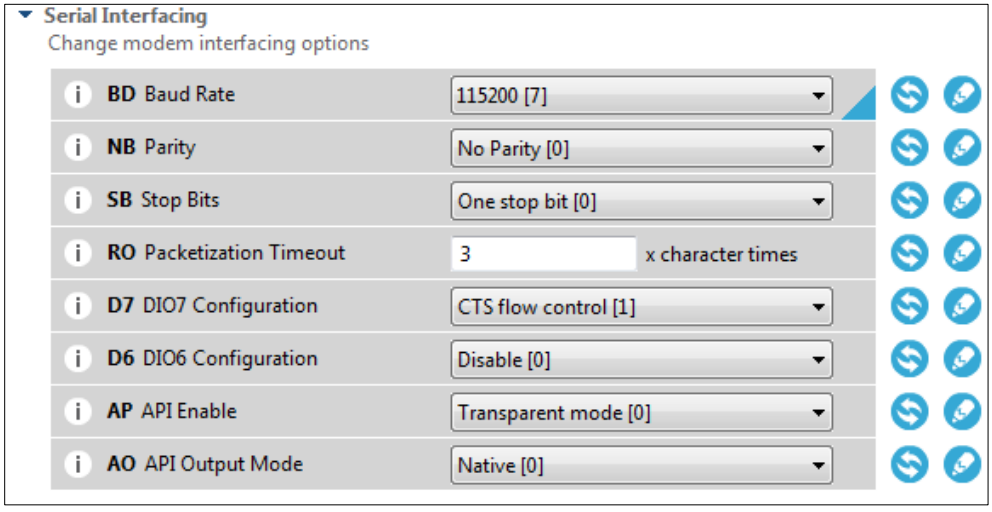

Figure 31: Addressing Serial Interfacing Parameters in XCTU.

4.2.2. Zigbee Xbee – Arduino Connection

After we are done with configuring the Xbee modules for Point-to-Multipoint communication, we have to connect them to the Arduino board and be able to send as well as receive simple messages.

Unfortunately, the Xbee module uses 3.3V DC power supply, which is much lower than 5V DC that Arduino can supply, so Sparkfun Xbee Explorer Regulated (see Figure 32) is

realized. The Explorer board has a 3.3V regulator so we can have direct access to the serial and programming pins on the Xbee unit and be able to power it with the 5V voltage pin from the Arduino. Moreover, the Explorer board has some basic activity indicators: Power, Received Signal Strength Indication (RSSI), Data In (DIN), and Data Out (DOUT) activity LEDs.

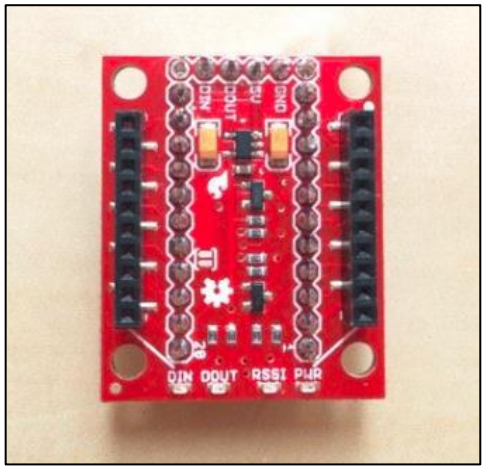

Figure 32: Sparkfun Xbee Explorer Regulated, which has a 3.3V regulator to power Xbee from Arduino 5V power supply. [39]

Figure 33: Connect Zigbee Xbee to Explorer board.

4.2.3. Zigbee Xbee Implementation with Arduino

Since we are using a programmable Xbee module, the RF data received will not transfer directly to the Arduino RX pin, they are blocked by a Freescale microcontroller inside the module itself. Therefore, we have to bypass the Freescale microcontroller in "setup()" function, which is carried out every time the system boots up. During booting, the Arduino writes "\n\r" to Xbee (go to the next line, similar to pressing Enter), and Xbee will respond with Boot Loader Menu (see Figure 34). Next, we will write "B" for Bypass. At this point, the Freescale

microcontroller is successfully bypassed, and Arduino should be able to read and RF data from Xbee.

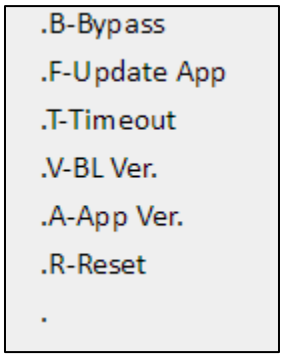

Figure 34: Boot Loader Menu. [41]

In order to transmit a command from Coordinator to Routers, messages written to Coordinator DIN pin have to follow Transmit Request frame format, which is described in Figure 35. This frame type causes Coordinator to send payload data as RF packet to specific Router destination. All fields require Hexadecimal value of ASCII code. Some fields are reserved by the device: Start Delimiter, Frame type, Broadcast Radius, Options. User can use either 64-bit destination address or 16-bit destination network address of the router. In our case, we use 64-bit MAC address on the back of our Router Xbees, and set 16-bit destination network address to 0xFFFE (default for unused). Check sum bit field indicates the difference of 0xFF and the sum of all previous bit fields.

Frame data fields	Offset	Example
Start delimiter	0	0x7E
Length	MSB 1	0x00
	LSB 2	0x16
Frame type	3	0x10
Frame ID	4	0x01
64-bit destination address	MSB 5	0x00
	6	0x13
	7	0xA2
	8	0x00
	9	0x40
	10	0x0A
	11	0x01
	LSB 12	0x27
16-bit destination network address	MSB 13	0xFF
	LSB 14	0xFE
Broadcast radius	15	0x00
Options	16	0x00

RF data	17	0x54
	18	0x78
	19	0x44
	20	0x61
	21	0x74
	22	0x61
	23	0x30
	24	0x41
Checksum	25	0x13

Figure 35: Example of Xbee Transmit Request Frame format. The example shows how to send a transmission to device with address 0x0013A200 400114011, and message "TxData1B". [42]

Messages sent from the Coordinator come in 2 types: routine status request and command. A routine status request tells the Router to send back its sensors information. The command, on the other hand, tells Router to execute a task, which ranges from turn on/off light, to turn on the fan until the desire temperature is reached. Our RF data frame consists of 5 bytes. The first byte indicates the Router's number (1 or 2). The next 3 bytes are reserved for command values (for example, temperature command from the user). The last byte shows 0 for command, 1 for routine status request. Specific messages can be found in the Appendix.

On the Router end, it will receive a purely 5-byte message from the Coordinator without any additional information and carry out a predefined task. For example, when Router 2 receives message "20010", it turns a LED on since byte 0 indicates router number (2), byte 1 and 2 are reserved for future use, byte 3 shows LED state (1 for on, 0 for off), and the last byte shows request mode (0 for command, 1 for status request). Similarly, to send a message back to the Coordinator, we only command the Arduino to write a 5-byte message to Xbee DIN pin.

The Router will send back an acknowledgement message (ACK) if it receives a message from the Coordinator. Therefore, we need to differentiate between an ACK and an actual response. Any response message starts with 0x7E, but the third received byte of an ACK is 0x07 while that of an actual response from Router is 0x11. Bytes 5 to 13 of the message indicate the sender (Router) 64-bit MAC Address and bytes 15 to 20 shows the actual message from the Router.

4.3. Local-server communication WiFi

4.3.1. ESP8266 WiFi Module Wiring:

Based on thorough research, we decided to use the ESP8266 ESP-01 module (Figure 36 and Figure 37) to provide WiFi access to our microcontroller. The ESP8266 module is a low-cost WiFi SoC (System-on-Chip) with full TCP/IP capability and is frequently used in microcontroller project requiring WiFi connection. For communication, the ESP8266 module has both a SPI and an UART interface.

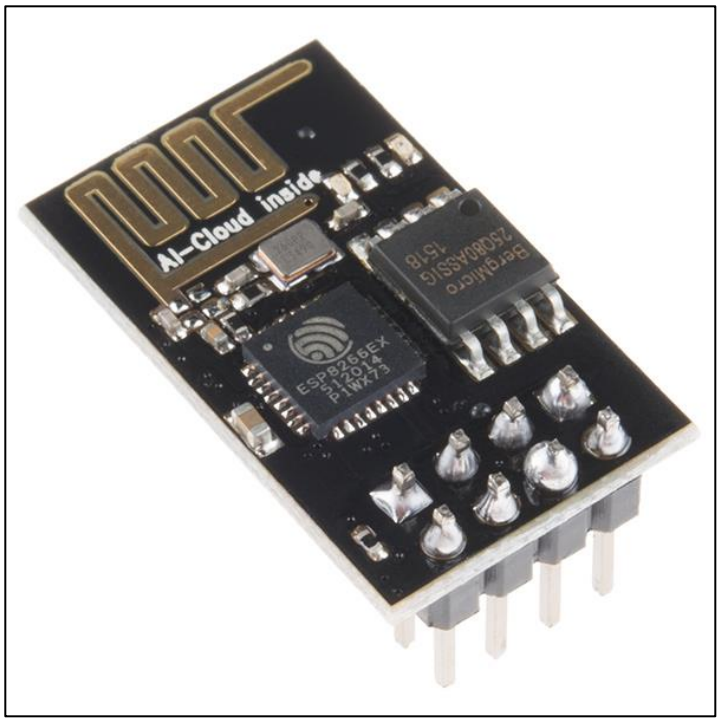

Figure 36: ESP8266 WiFi Module. [37]

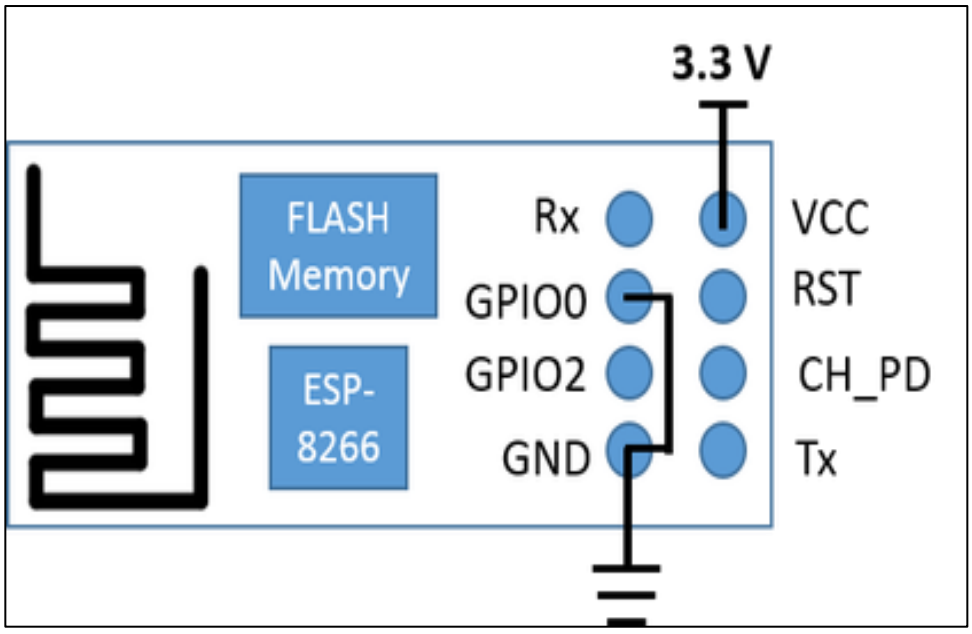

Figure 37: ESP8266 Pin Diagram. [43]

The ESP8266 WiFi Module is a System-on-Chip (SoC) with integrated TCP/IP protocol to grant any microcontroller access to the WiFi network.

In our system, we will connect the ESP8266 module to the coordinator Arduino in order to send information to and to receive commands from the web server. The communication interface UART will be used for connection. The Vcc pin of the WiFi module should be connected to 3.3V pin of the Arduino.The CH_PD (chip_enable) pin and RST (reset) pin of the WiFi module should also be connected to 3.3V for normal operation. The Tx pin of the ESP8266 is connected to the Rx pin of the Arduino, and the Rx pin of the ESP8266 is connected to the Rx pin of the Arduino. Since the logic level of the ESP8266 module is 0V to 3.3V, and the logic level of Arduino Mega 2560 microcontroller is 0V to 5V, we use a logic level converter to connect these two components. The logic level converter, and the connection diagram between the ESP8266 module, the logic level converter, and the Arduino Mega 2560 microcontroller are shown in Figure 38 and Figure 39.

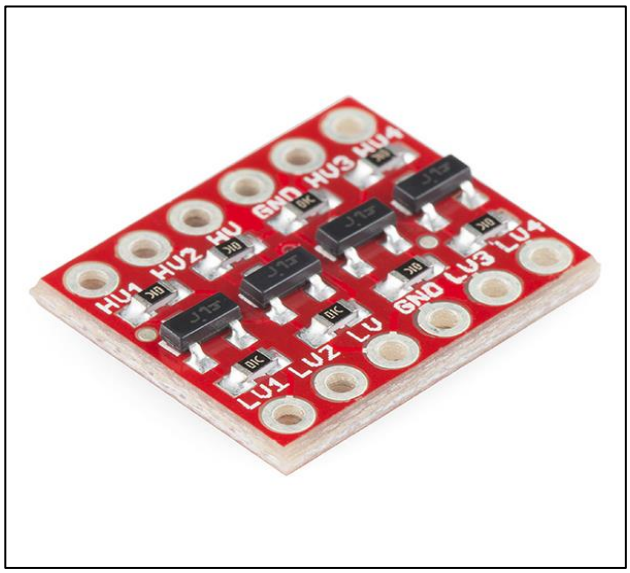

Figure 38: Logic Level Converter from SparkFun, The HV (High Voltage Pin) is connected to 5V source. The LV (High Voltage Pin) is connected to 3.3V source. [44]

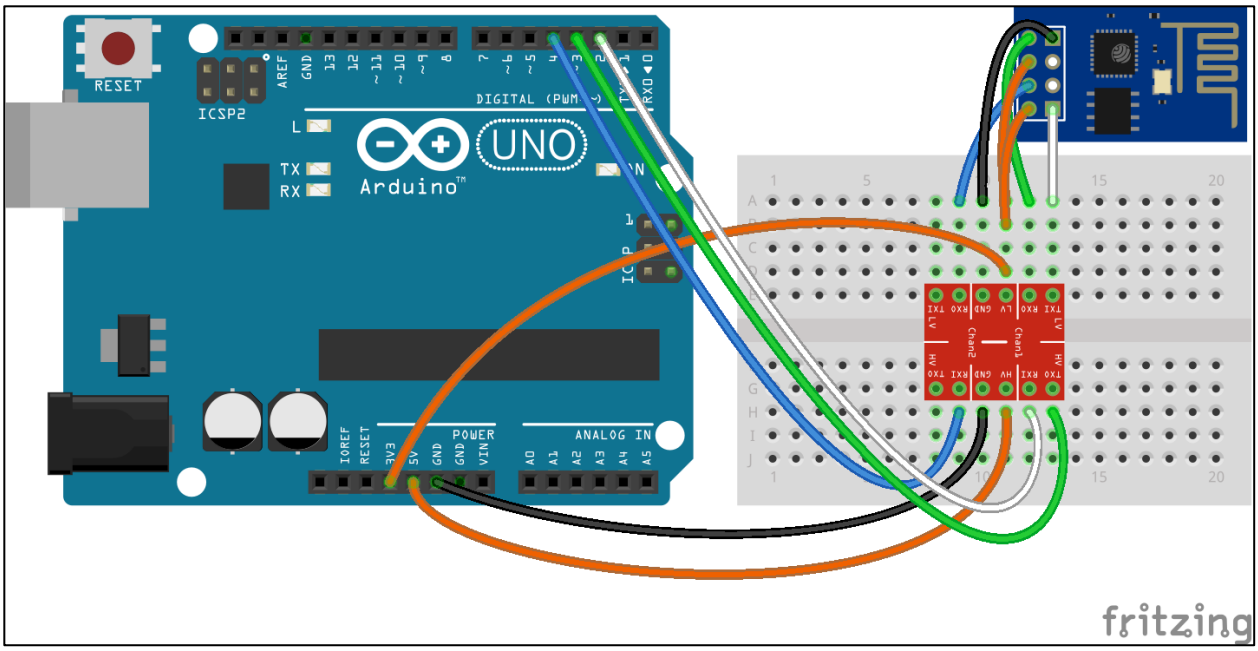

Figure 39: The connection diagram between the ESP8266 module, the logic level converter, and the Arduino Mega 2560 microcontroller [45].

4.3.2. ESP8266 WiFi Module Configuration:

To communicate with the ESP8266 WiFi module, we use the AT (ATtention) command set. The AT commands is a set of instructions used to control a modem. The AT command format, as well as all the commands, are summarized in Figure 40.

Basic	WiFI layer	TCPIP Layer
AT	AT+CWMODE	AT+CIPSTATUS
AT+RST	AT+CWJAP	AT+CIPSTART
AT+GMR	AT+CWLAP	AT+CIPSEND
AT+GSLP	AT+CWQAP	AT+CIPCLOSE
ATE	AT+CWSAP	AT+CIFSR
	AT+CWLIF	AT+CIPMUX
	AT+CWDHCP	AT+CIPSERVER
	AT+CIPSTAMAC	AT+CIPMODE
	AT+CIPAPMAC	AT+CIPSTO
	AT+CIPSTA	AT+CIUPDATE
	AT+CIPAP	+IPD

Figure 40: AT Command Lists. The AT Command is divided into Layer: Basic set up ESP8266, WiFi Layer to set up Internet Connection, and TCPIP Layer to send and receive HTTP messages [46].

After connecting the ESP8266 WiFi module to the serial interface 2 of the Arduino as in Figure 29, we used AT commands to configure the WiFi module from the Arduino. The ESP8266 will return an OK message after each successful command sent. In the setup() function of the Arduino coordinator code, we started by sending AT+RST command to reset the WiFi module. We then used the AT+CWMODE=3 command to set up the ESP8266 to dual AP (access point, or host) mode + station (or client) mode. This setting will configure the ESP8266 to either a WiFi access point, such that it has WiFi name and password, or a WiFi client, so the module can be connected to a local WiFi. For our application purpose, we want to connect the ESP8266 to our local WiFi for sending information and receiving commands. The command that we used is AT+CWJAP = <ssid, pwd>, with ssid is the local WiFi name and pwd is the WiFi password. And finally, after setting up the WiFi connection, we configured the ESP8266 as a server using AT+CIPSERVER=1,80 command to open server at port 80, so it can receive informations sent from our website. The complete list of commands that we used for set up the ESP8266, is shown in Appendix.

4.3.3. ESP8266 WiFi Module Implementation:

After the set up phase, the ESP8266 module will be fully functional to send and receive information through TCP/IP using AT commands. To send data to our web server, the first step is to establish a connection between our ESP8266 open server and our webhost. We accomplished this task using AT+CIPSTART commands. The ESP8266 will return an OK

message if the connection is successful, or return an ERROR message if not. After verifying the connection, we can send data to server using AT+CIPSEND commands. The syntax of this command is to first specify the length of our message in characters. After that, the ESP8266 module will start receiving the message. Since our webhost is constructed using HTML and JavaScript, the message to be sent must also be written in HTML format. The length of the message to be sent, including the HTML format, must correspond to the specified length, otherwise errors may occur.

If the message length is correct to the specified length, and the webhost can receive the message, then the server will return an acknowledgement (ACK) message. This ACK message is also written in HTML format, and provides information whether or not the server can successfully decode the message. If not, there are multiple possible reasons for the problem, but usually this is because the HTML message is not precisely formatted, or the message length is not greater than the specified length, so our web server received incomplete message. Otherwise, if the sending message state is successful, indicated by the returned 200 OK ACK message from the web server, then we can close the connection using AT+CIPCLOSE command. The complete code to send data from ESP8266 to the server is shown in Appendix.

Receiving commands from the server to the ESP8266 is more challenging. Since the ESP8266 is connected to the local WiFi, which has its own local IP address and a local server at this IP address, we need to develop a method to access this local server. We used Ngrok software, which can establish a secure TCP tunnel to our local IP address. When the Ngrok program is running, it provides an alternate URL for our local IP address that can be used to access the local server from anywhere. We used that URL in our webserver code to send HTML commands to our ESP8266.

After establishing the secured tunnel, we set up an interrupt that check for every message our ESP8266 received from the web server. If the message is the command from the web server, we decode the message (in HTML format) to get the command. Then based on which command the server sent, the coordinator Arduino will decide which function to perform. The code for receiving and decoding commands sent from the server is shown in Appendix.

The complicated part of the ESP8266 module implementation is to send the result image from camera module to server. Since the maximum length of each message using AT+CIPSEND is only 2048 bytes, we need to break out image files into several smaller files and send them consecutively to the web server. Because each image file is approximately 14kB to 15kB in size, therefore the ESP8266 need to send 7 or 8 HTTP messages per image. This behavior created certain problems, such as long delay time for other tasks while sending image and if image sending process is interrupted, then the image file the server received could possibly be corrupted. We attempted different ways to reduce the delay time (including, such as further compress the image file before sending), reduce overhead tasks, in an effort to optimize the execution time for sending each message. The complete code for sending images to the web server is shown in Appendix.

4.4. User Interface

An important feature of the project is to design and create a friendly, easy-to-use, responsive and interactive user interface for the system. This is the place where users can view the current status of different criteria in their houses such as temperature, light luminosity, humidity, camera sensor, etc. After the master Arduino coordinator sends data to the server, the data is analyzed in the backend using PHP programming language and is then displayed and represented in the front-end website in a meaningful way. Here, users can view the temperature, light radiance and camera status in real-time over-the-air. The data received from the coordinator is updated continuously and whenever the data is changed in the backend (the server), it will be reflected immediately in a responsive graph at the website. To create and design a front-end website, we use standard markup languages, HTML, CSS and Javascript for the layout of content, style and program for the website. Especially, in order to support a dynamic and interactive website, we use an additional Javascript framework, the AngularJS. It provides the tools for designing reactive, responsive and beautiful charts. [32]

For reference, the link to the sample website is: http://smarthomewpi.host22.com/. This website is different from the original website in that the data of the light luminosity, temperature and figures of camera are hardcoded, which is not real data. In the original website, the entire system has to be powered and the data will be updated in real-time. The current website is designed for the use in laptop or desktop resolution. In the future, the user interface can be significantly improved by designing responsive website specifically for smartphones and tablets. Figure 41 demonstrates our designed user interface.

Figure 41: Designed Website for User Interface.

4.5. Integration

In this section, we will discuss how individual modules are connected together and how they perform as a system is described. Overall, the Coordinator frequently requests data from the Routers' sensors, and in case it receives a command from the Web Server, the command will be send directly to the routers to perform specific task.

4.5.1. Routers Implementation

Router 1 is the Light Module, its peripherals consist of a LED, an XBEE module, and a PIR sensor. Router 1 turns LED on for 5 seconds if the PIR sensor detects any movement. When it receives command from the Coordinator, it turns the LED on or off accordingly. A Helper function "readPIR" (discussed in section 4.1.1) is developed so Router 1 can read data from PIR sensor every loop cycle. When Arduino receives data from RX pin, we read all 5 bytes and identify if the message is a command from the server or a status request by looking at the 5th byte (ASCII "0" for command, ASCII "1" for request).

If a command is identified, "checkCommand" function is called to check the whether the Router 1 should turn light on or off depending on the 4th byte. Figure 42 shows the implementation of the "checkCommand" function.

```
void checkCommand() {
  if (mess3 == 49) {              //mess3 = 0
    LED_status = HIGH;            //LED on
    flag_command = 1;
  }
  else if (mess3 == 48) {         //mess3 = 1
    LED_status = LOW;             //LED off
    flag_command = 0;
  }
  mess0 = 48;
  mode = 2;
}
```

Figure 42: checkCommand()reads the 4th byte of the command, and turn the LED on/off

If a request is identified, Router 1 sends a 5-byte message that starts with its router number (which is ASCII "1"), two zeroes, and ends with LED status ("1" for on, "0" for off) and PIR data ("1" if a motion is detected within 5 seconds, "0" if no motion is detected). Figure 43 depicts the code implementation for the Router 1 request response.

```
else if (mode == 1 && mess0 == 49) {            // for Request message
  if (ack == 1) {
    Serial.println("Sending");                  // Print "Sending" to computer console
    Serial1.write(49);                          // Router number
    Serial1.write(48);
    Serial1.write(48);
    if ((LED_status == HIGH)) {                 // Write LED status
      Serial1.write(49);
    } else Serial1.write(48);
    if ((value == HIGH)) {                      // Write PIR data
      Serial1.write(49);
    } else Serial1.write(48);
    Serial.println("End Data");                 // Print "End Data" to computer console
    ack = 0;                                    // Reset flag and variable for next loop
    mess4 = 52;
    mess0 = 48;
    mode = 2;
  }
```

Figure 43: Code snippet for Router 1 request response. It starts by writing "Sending" to Computer console, then send 5-byte status message to Arduino TX pin (DIN pin on Xbee for transmission).

Router 2 is the Heat Module, whose peripherals consist of a LED, a XBEE module, a temperature sensor, and a small fan. Its core operation is similar to that of Router 1; however, Router 2 can receive and execute two commands (for LED and fan), and a function to collect temperature data. If the requested temperature is less than the current temperature from the temperature sensor, the fan is turn on. Figure 44 demonstrates a Router 2 command execution code snippet.

```
void checkCommand() {
  request_temp = mess3;
  if (mess3>47) {                   //LED command if byte 4th decimal == 48 or 49 ("0" or "1")
    if (mess3 == 49) LED = HIGH;    //LED on
    else if (mess3 == 48) LED = LOW;  //LED off
  }
  else {
    request_temp = mess3;           //read request temperature
    Serial.print("Request temp = ");
    Serial.println(request_temp);
    if (request_temp<temp) fan_on = 1;//turn fan on if request temperature < current temperature from temperature sensor
    else fan_on = 0;
  }
  mess0 = 48;
  mode = 2;
}
```

Figure 44: Router 2 command execution code snippet.

4.5.2. Coordinator Implementation

As mentioned in previous chapters, the coordinator module of our system is in charge of sending commands and receiving information from two routers module with the ZigBee module, communicating with the webserver using the ESP8266 WiFi module, as well as capturing image with the ArduCam camera module. Based on these primary functionalities, the coordinator implementation can be described in three critical steps.

The first critical step is the ZigBee communication with two routers. It includes several steps: receiving messages from two routers module, decoding the messages, and sending messages to a destination router. To send messages from coordinator to routers, since the coordinator is in API mode, the messages must be constructed in the same format as described in section 4.2. There are two types of messages that the coordinator uses: request message and command message. The request message is sent from the coordinator to request information from routers, i.e the LED status at router 1 or the temperature measured at router 2. The command message is to command a router to execute an operation, such as turning on the LED at router 1 or turning on the fan at router 2. Both types of messages contain 5-bytes of desired information, and 4-bytes of destination router's address. With defined message format, sending message is very simple. We used the built-in function Serial1.write() in Arduino to send the message to routers. The message format, as well as the messages that we used, are shown in Figure 45.

```
byte light1_on[23] = {0x7E, 0x00, 0x13, 0x10, 0x01, 0x00, 0x13, 0xA2, 0x00, 0x41, 0x52, 0xEC, 0x6B,
                      0xFF, 0xFE, 0x00, 0x00, 0x31, 0x30, 0x30, 0x31, 0x30, 0x60}; // Command message: turn light 1 on
byte light1_off[23] = {0x7E, 0x00, 0x13, 0x10, 0x01, 0x00, 0x13, 0xA2, 0x00, 0x41, 0x52, 0xEC, 0x6B,
                       0xFF, 0xFE, 0x00, 0x00, 0x31, 0x30, 0x30, 0x30, 0x30, 0x61}; // Command message: turn light 1 off
byte request_light[23] = {0x7E, 0x00, 0x13, 0x10, 0x01, 0x00, 0x13, 0xA2, 0x00, 0x41, 0x52, 0xEC, 0x6B,
                          0xFF, 0xFE, 0x00, 0x00, 0x31, 0x30, 0x30, 0x30, 0x31, 0x60}; // Request message:
                                                                                      // request information of the light module

byte light2_on[23] = {0x7E, 0x00, 0x13, 0x10, 0x01, 0x00, 0x13, 0xA2, 0x00, 0x41, 0x52, 0xEC, 0xD7,
                      0xFF, 0xFE, 0x00, 0x00, 0x32, 0x30, 0x30, 0x31, 0x30, 0xF3}; // Command message: turn light 2 on
byte light2_off[23] = {0x7E, 0x00, 0x13, 0x10, 0x01, 0x00, 0x13, 0xA2, 0x00, 0x41, 0x52, 0xEC, 0xD7,
                       0xFF, 0xFE, 0x00, 0x00, 0x32, 0x30, 0x30, 0x30, 0x30, 0xF4}; // Command message: turn light 2 off
byte request_temp[23] = {0x7E, 0x00, 0x13, 0x10, 0x01, 0x00, 0x13, 0xA2, 0x00, 0x41, 0x52, 0xEC, 0xD7,
                         0xFF, 0xFE, 0x00, 0x00, 0x32, 0x30, 0x30, 0x30, 0x31, 0xF3}; // Request message:
                                                                                      // request information of the heat module
byte command_temp[23] = {0x7E, 0x00, 0x13, 0x10, 0x01, 0x00, 0x13, 0xA2, 0x00, 0x41, 0x52, 0xEC, 0xD7,
                         0xFF, 0xFE, 0x00, 0x00, 0x32, 0x30, 0x30, 0x30, 0x30, 0xF4}; // Command message: turn heat on
                                                                                      // to the desired temperature
```

Figure 45: ZigBee message format and the message that are used. The 14th to 17th bytes are the address of the destination router, and the 5 bytes just before the last byte is the data to be sent.

For receiving a message from the routers, the only type of message that the coordinator can receive is the messages with each router's status. In particular, these messages provide information of the LED (from router 1) or the fan module and the temperature (from router 2). Therefore, for simplicity, instead of dedicating an interrupt to listen for every message from the routers, the coordinator will send a request (as discussed above) to both routers frequently, at a period of 5 seconds. By receiving the request message, the router will then send a response.

The respond message from the router has two different parts. The first one is an acknowledge (ACK) message, generated by the router's ZigBee module to indicate it has received the request message. The second part is the actual message that contains the desired information from router modules. As mentioned in previous chapters, a ZigBee message is padded with several additional bytes. The actual message is only the last 5 bytes of the message, excluding the checksum byte, and the only additional information required is the 4 bytes of router's address to know the origin of the message. In order to extract the exact information, the code first checks for the ACK message. The first three bytes of the ACK message are 0x7E, 0x00 and 0x07. The coordinator checks for these three bytes to ensure receiving the ACK

response for its sent request. The length of an ACK message is always 10 bytes. Then, by indexing byte by byte of the response message, the code will detect the 4 bytes of router address and 5 bytes of information. The complete code for sending ZigBee message and decoding responded ZigBee message are shown in the Appendix.

The next primary task of the coordinator is the WiFi communication with the webserver. Communication includes two parts: send the message to, and receive message from the webserver. There are two functions for sending message in the coordinator's code. The first function is for sending status data from the coordinator to webserver. We implemented this function using the AT commands mentioned in 4.3.3. The message format is HTTP 1.1 and the data that is embedded in the message is in JSON format. The code also checks for responding HTTP message from the web server for each AT command to check the sending status. These responding messages will later on be erased from serial buffer of the serial interface. The HTTP message and the JSON data format are shown in Figure 46 and Figure 47.

```
String postRequest0 =
  "GET /" + uri + "?" + json + " HTTP/1.1"; // first line of the HTTP Get message
String postRequest1 =
  "Host: " + server; // second line of the HTTP Get message
```

Figure 46: HTTP GET message format for sending JSON data "uri" is the name of the php file that handle receiving data on the web server. "json" is the data to be sent in json format, and "server" is the URL of the web server.


```
{"name":"Occupancy2","sender":1,"time":171,"value":18},
{"name":"Occupancy2","sender":1,"time":172,"value":18},
{"name":"Occupancy2","sender":1,"time":173,"value":18},
```


Figure 47: JSON data format. Each JSON object has multiple keys and values "sender" is the router that sent the JSON data. The data "time" is a counter value that we use to keep track of sending time; and data "value" is the temperature value from the router to the heater module.

The second function is used to send images to the web server. The main difference between sending images and sending JSON data is the file size. Image file sizes typically exceed the limit of 2 kB for each AT+CIPSEND command. In our case, with 640x480 in dimensions and 2MP resolution, the result JPEG files from the camera module are typically 14 to 15 kB. Therefore, as mentioned before, for each image the coordinator has to send 7-8 HTTP messages. These messages are formatted to send .txt file, and each .txt file contains a part of the binary data of the JPEG image file. When received by the webserver, these files will be assembled into one JPEG image file and displayed. The HTTP message format for sending image will be shown in Figure 48.

```
start_request = start_request +
                  "\n--AaB03x\n" +
                  "Content-Disposition: form-data; name=\"userfile\"; filename=\"CAM.TXT\"\n" +
                  "Content-Transfer-Encoding: binary\n\n";
String postRequest0 = "POST " + uriCamera + " HTTP/1.1";
String postRequest1 = "Host: " + server;
String postRequest2 = "Content-Type: multipart/form-data; boundary=AaB03x";
String postRequest3 = "Content-Length: "; // The message length is inputed here.

end_request = end_request + "\n--AaB03x--\n";
// in file upload POST method need to specify arbitrary boundary code
```

Figure 48: HTTP POST message for sending an image file date. This HTTP message indicates the sending of binary data as a file CAM.TXT to the "userfile" folder in the web server.

For receiving commands from the web server to the Arduino, we set up an interrupt on serial interface 2 to detect incoming messages. This interrupt, in turn, will call the httpCommand() function, which will decode the received HTTP messages and retrieved the commands. After the commands are retrieved, according to which commands it is, the code will send corresponding commands to the corresponding router. The complete code for receiving and decoding HTTP messages are shown in Appendix.

The final primary task of the Coordinator is to take an image from the ArduCam camera module. The ArduCam Mini 2MP module is provided with supporting library, and it is connected to the Coordinator using the SPI interface. For our implementation, initially the code open the SPI communication between ArduCam module and the coordinator using SPI.begin() function. Then it will attempt to write a sample byte to a register on ArduCam SPI bus, and later read the byte at that register to verify the functionality of the SPI bus. Next, setup code checks for the register CHIPID of the camera sensor used by ArduCam module, to verify it is the correct version OV2640. Finally, the code sets the format of the result image to JPEG, sets JPEG image size to 640x480, initializes the camera module and clears its FIFO flags using functions provided in the ArduCam library. The complete setup code for camera is shown in Figure 49.

```
// Setup for Camera, resolution 640x480, compresssed file type JPEG
pinMode(CS, OUTPUT); // set the CS as an output:
SPI.begin(); // initialize SPI

//Check if the ArduCAM SPI bus is OK
myCAM.write_reg(ARDUCHIP_TEST1, 0x55);
temp = myCAM.read_reg(ARDUCHIP_TEST1);
if (temp != 0x55) {
  Serial.println("SPI1 interface Error!");
  while (1);
}

// Check if the camera module type is OV2640
myCAM.wrSensorReg8_8(0xff, 0x01);
myCAM.rdSensorReg8_8(OV2640_CHIPID_HIGH, &vid);
myCAM.rdSensorReg8_8(OV2640_CHIPID_LOW, &pid);
Serial.println(vid, HEX);
Serial.println(pid, HEX);
if ((vid != 0x26) || (pid != 0x42)) {
  Serial.println("Can't find OV2640 module!");
  while (1);
}
else {
  Serial.println("OV2640 detected.");
}

// Change to JPEG capture mode and initialize the OV2640 module
myCAM.set_format(JPEG);  myCAM.InitCAM();
myCAM.OV2640_set_JPEG_size(OV2640_640x480);
Serial.print("640x480:");
Serial.println(OV2640_640x480);
myCAM.clear_fifo_flag(); myCAM.write_reg(ARDUCHIP_FRAMES, 0x00);
```

Figure 49: Camera setup code. Due to hardware limitation, we only used image size of 640x480

After being setup, taking image from ArduCam camera is straightforward. The capturing process is, first the FIFO flags are cleared and the FIFO bus of the camera module is flushed, to capture the new image. After the image was captured using the library function start capture on the input ArduCam object, the code used SPI.transfer() function to read the image bytes by bytes to the Arduino and store in a buffer. This buffer's size, 1781 bytes, is the maximum size of data that could be sent in each AT+CIPSEND command instead of 2048 due to the HTTP format overhead. For each time the buffer is filled, the code will send this buffer data in .txt format to the web server and clear the buffer. The process is repeated until the entire image has been sent. The complete codes for capturing image, as well as sending image to the web server, are shown in Appendix.

By predefining the primary functionalities, the coordinator's operation becomes simple to implement. The coordinator will send a request message for each router at 5s interval, to request their status data and send them to the web server. The 5s interval is achieved using a Timer object. The sending process can be interrupted if the coordinator receives a command from the web server. Whenever there is a command from a web server is detected, an interrupt will be raised and the coordinator will suspend other tasks to decode the message and send the corresponding commands to the corresponding router, before resuming its current tasks.

When taking an image, whenever motion is detected by checking the distance measure using an ultrasonic sensor, images will be captured and send to the webserver for each 2 seconds

with a defined number (in our code) of total images to be sent. This action also has higher priority when it comes to sending JSON data to the web server and receiving commands from the web server, such as when an image is sent these tasks are suspended and are only resumed when the image sending process is finished. The code for reading distance from ultrasonic sensor is shown in Figure 50. The loop() function code is shown in the Appendix.

```
void readSonic() {
  digitalWrite(trig, LOW);    // trig off
  delayMicroseconds(5);
  digitalWrite(trig, HIGH);   // trig on
  delayMicroseconds(10);
  digitalWrite(trig, LOW);    // trig off
  duration = pulseIn(echo, HIGH);
  distance = int((duration / 2) * 0.034); // calculate distance
}
```

Figure 50: readSonic() function

The output of this function is the distance between the sensors and the detected moving object.

4.5.3. Web Server Implementation

In this section, after receiving sensor data (which includes the occupancy status) the temperature and picture captured by the camera from the coordinator, the server processes the data in the backend, analyzes the results, organizes and manages them into the corresponding categories in the front-end website of the user interface. Specifically, the light luminosity and temperature are graphically presented in a responsive graph in real-time. The image captured by camera is updated frequently and changed whenever a new image has been sent to the server from the coordinator. Figure 51 visually demonstrates the data flow in our Smart Home System.

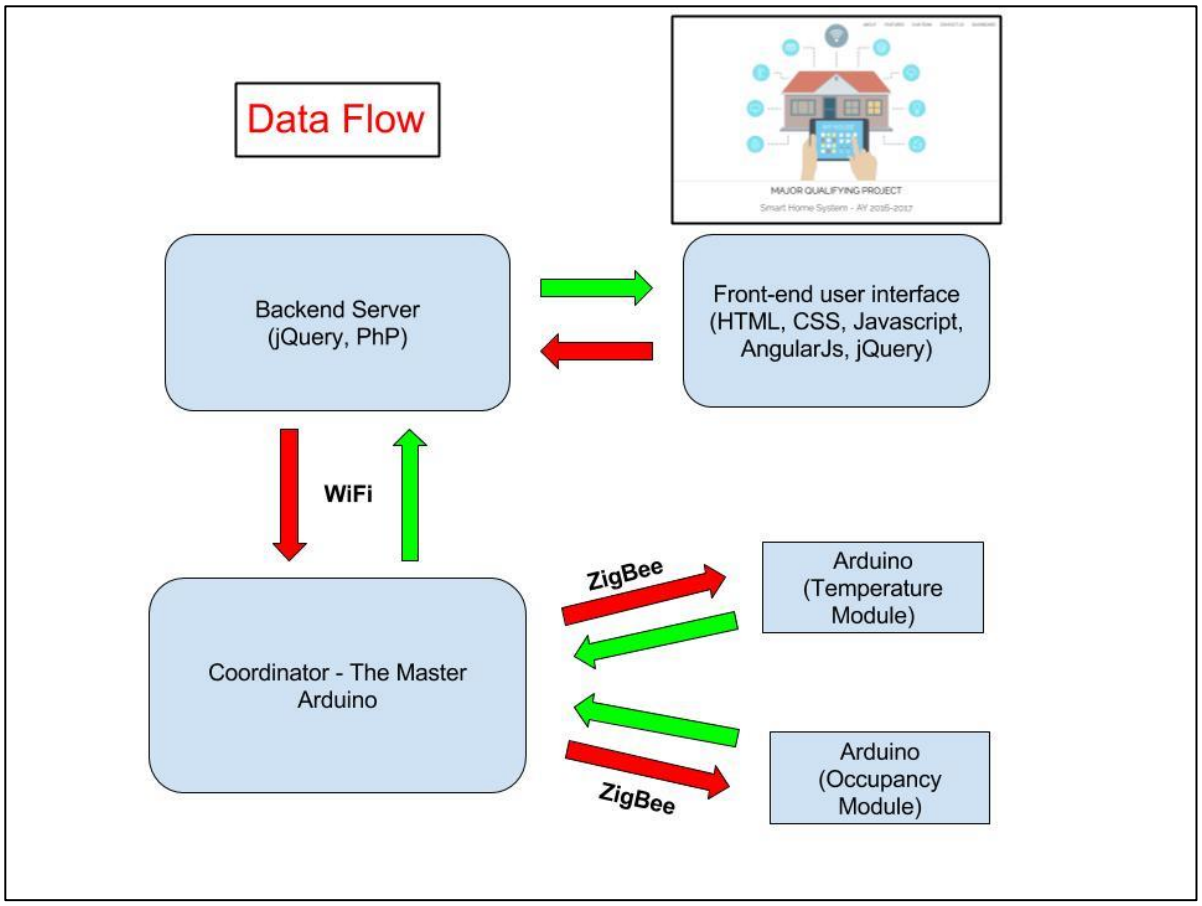

Figure 51: Data Flow in our designed system. After the data is sent to the server by the coordinator (the master Arduino) it is processed and analyzed in the backend before being presented in the front-end website of the user interface.

As discuss in section 4.4, which deals with the design and implementation of the user interface, we utilized the PHP server programming language for the server backend side and the standard markup languages (HTML, CSS, Javascript) and Javascript framework, the AngularJS for the front-end side. These web programming languages are the necessary tools to not only design a fully-functional, user-friendly website, but also to create an interactive and responsive user interface.

5. Results

5.1. Functionality Testing

The final prototype has the four main functionalities. The first one is coordinator's ability to receive commands from the webserver, receive information from routers and router's information and camera images to the webserver. The second functionalities are the web UI's ability to send commands to and receive information and images from the coordinator, as well as display these information and images. The third functionalities are the routers' ability to receive ZigBee commands from the coordinator, and use these commands to control the corresponding peripheral modules.

We conducted various functionality tests to verify the capabilities of our system. For the first functionalities testing, we dedicated one computer as a connection host, where the ngrok program runs to provide a secure TCP connection to the coordinator. One computer was dedicated to open the web UI for controlling and monitoring the system. The two routers with peripherals were connected to 12V power supply. The coordinator module was connected to a laptop to monitor its serial interfaces. To test the communication with the two routers, we monitored the serial interface that connect to the ZigBee module to check for data coming from routers every 5 seconds and verify the format and information correctness. To test the ability to send information to the webserver, we checked for the JSON data file that continuously being updated with data sending from the coordinator for information correctness, as well as the web UI display the correct information/images sending from the coordinator. To test the ability to receive commands from the web server, we monitored the serial interface that connected to the WiFi module to check for commands information received whenever a button is pressed. The monitored information on the coordinator is shown in Figure 52 and Figure 53.

Figure 52: Response message from the Wifi module indicating successful delivery of JSON data. On the left is the response message for successfully sending data to the web server. The first line indicates establishing the TCP connection. The second line indicates the length of the HTTP message. The third line indicates that the message has been sent. Next, the HTTP response indicates successful reception of the message. The last line indicates the closing of the TCP connection. On the right is the sample message for receiving commands from the web server. The first line and second line indicate reception of a command from the webserver to turn on/off the LED in router 1. Next is the status of forwarding the command to router 1. The fourth line indicates the time in milliseconds between sending the command and receiving the acknowledgement message from router 1, and the last line is the acknowledgement message in hexadecimal.

```
Send Command
ACK :27
Receive Command
Get Data
90013A204152EC6B9BF5131303030304ERouter 1
Time for receiving and decoding message:18
Total Time Spent
46
```

Figure 53: Example of sending request message and receiving requested information from router 1. The first two lines, as mentioned in figure 53, indicated the time in milliseconds between sending the request message and receiving the acknowledgement message. The third, fourth and fifth line indicates receiving of requested information from router 1, with the fifth line is the message in hexadecimals. In the message, "4152EC6B" is the MAC address for the router 1 ZigBee module, and "3130303030" is the requested information in hexadecimals. All time measurements are in milliseconds.

For the web UI testing, we retained our testing configuration. We monitored the JSON database files that contains information and images and verify with the information and images sent from the coordinator. We wanted to verify if the files are updated correctly and if the stored information is in the correct format. We also verified the consistency between the data stored in our web database and the data displayed on the webserver.

Lastly, to test the router's functionalities, we sent various commands from the webserver to check if the LED in router 1 was turned on/off, the LED in router 2 was turned on/off, or the fan module was turned on/off. We then monitored the serial interfaces of the routers, to check if they receive the corresponding commands from the coordinator, and if they turned the peripherals on/off. Figure 54 shows the monitored information on router 1. The monitored information on router 2 is shown in Figure 55.

```
Receive Data
3130303031
Sending
```

Figure 54: An example of a requested message received from the coordinators at router 1. The message is in hexadecimals. The first byte of the message (31, which is character '1' in ASCII) indicates the destination routers. The fourth byte (30, which is character '0' in ASCII) is used to command the LED turning on or off. However, the fourth byte will only be considered if the fifth byte value is 30, which indicates command message. For this example, the fifth byte is 31, which indicates a request message.

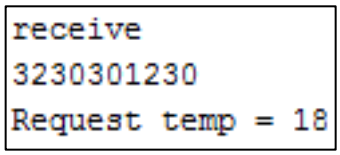

Figure 55: An example of a command message received from the coordinators at router 2. The message is in hexadecimals. The first byte of the message (32, which is character '2' in ASCII) indicates the destination routers. The fourth byte (12, which is 18 in decimal) is used to turn the fan on to decrease the temperature to 18^0C. The fourth byte will only be considered if the fifth byte value is 30, which indicates command message. For this example, the fifth byte is 30.

To test the entire system, we combined the three above tests. We first monitored the routers to check if they received the requested messages from the coordinator each 5 seconds, and if they sent the respond message back to the coordinator. We then monitored the coordinator, checking if the message is received and matched the monitored message on routers, and checking if the information retrieved from the message is packaged into JSON-formatted data and forwarded to the web server. Lastly, we checked the database file on our webhost to find the message sent from the coordinator, and verified if the UI has updated with the new message.

For receiving and executing commands from the webserver, when a button was pressed on when the required temperature is entered on the web UI, we first monitored the coordination for this command. Then, we monitored the corresponding router module to see if the commands are forwarded from the coordinator to the router. Finally, we checked if the light is turn on/off, or the fan module is turned on/off based on the corresponding commands.

The testing result was encouraging and matched our expectation. The lights are correctly turned on/off if the users press the corresponding button on the web UI. The fan module is correctly turned on if the users enter the required temperature if less than the current temperature (measured by the temperature sensors) and turned off if it is higher. The web UI is correctly updated based on the information from routers. A link to the video that showcases the full functionality demonstration of our system is provided in Appendix.

After verifying all functionalities, we next tested for transmission robustness of our system. The ZigBee communication works perfectly, even if we placed each router and coordinator in different rooms to maximize the transmission range. The data sent and received from the ZigBee communication was always consistent, and the latency between sending a request message, and receiving a response message, is less than 200 microseconds most of the time.

The WiFi communication, however, still has certain issues. These issues, unfortunately, are beyond our ability to correct or improve. The first issue is the message length limitation of the AT commands for the ESP8266 module. Since the maximum message length is only 2048 bytes, to send an image to the webserver the image must be divided into multiple smaller files and sent multiple times. For this reason, there is a possibility that the image received by the server can be corrupted if the connection is interrupted between each send. The second issue is the web server. Since we are using a free online webhost for our project, this server is not reliable, as we can't access the online database and our web UI sometimes. However, aside from

these two issues, the overall performance of our system communication is adequately good and meets our expectation.

5.2. Case studies

In this section, we will discuss the performance of the Smart Home System in extreme situations: severe temperature, and WiFi module losing connection.

5.2.1. Severe temperature Detection and Notification

Smart Home System ability to detect severe temperature and notice user is suggested. When the server collects a temperature value (from Router 2) greater than 40 degree Celsius, an email is sent to the user to warn them of extreme temperature at their home. An example of the extreme temperature warning can be found in Figure 56. One possible cause of this temperature value might come from malfunction of the temperature sensor, TMP36. Therefore, users can still go to the website and observe their houses from the security camera.

Figure 56: Example of severe temperature warning sent from server.

5.2.2. Breakdown of WiFi connection

The ability to recover from WiFi connection breakdown at the Coordinator is the main difference between our architecture and Central Hub architecture. By providing a backup WiFi module (ESP8266) and setting up ngrok for Router 1, the system still maintains connection to the server, even if the WiFi module on the Coordinator fails. The code snippet in Figure 57 describes verification of WiFi connection. If the WiFi module returns "No AP" when AT command "AT+CWJAP?" is issued, the WiFi connection is broken and the indicator "wifi_status" is set to 1. Then, Coordinator will stop its normal operation and send hard-coded special messages to Router 1 (message "11000") and Router 2 (message "11111") to indicate this failure mode.

```
void wifi_Status(){
  Serial2.println("AT");
  delay(200);
  if (Serial2.find("OK")){
    wifiStatus = 0;
  }
  else {
    wifiStatus = 1;
    return;
  }
  Serial2.println("AT+CWJAP?");
  delay(200);
  if (Serial2.find("No AP")){
    wifiStatus = 1;
    return;
  }
  else {
    wifiStatus = 0;
  }
}
```

Figure 57: Code snippet of WiFi verification with AT command.

Using the same reading mechanism indicated in section 4.2, when Router 1 and Router 2 detect "WiFi failure" messages, they both changed their Xbees configuration in order to talk to each other (Routers were originally talking to Coordinator only). Figure 58 demonstrates the code snippet for setting new configuration for the two Routers. Router 1's destination has to change to Router 2, and vice versa.

```
Serial1.write("+++");
delay(1000);
while(Serial1.available()) Serial.write(Serial1.read());
Serial1.write("ATDL4152ECD7\n\r");
delay(1000);
while(Serial1.available()) Serial.write(Serial1.read());
Serial1.write("ATID123\n\r");
delay(1000);
while(Serial1.available()) Serial.write(Serial1.read());
Serial1.write("ATWR\n\r");
delay(1000);
while(Serial1.available()) Serial.write(Serial1.read());
Serial1.write("ATCN");
delay(2000);
```

Figure 58: Example of using AT command to set Xbee configuration parameters. AT command starts with "+++" to turn Xbee into configuration mode. It is followed by destination address. In this case, Router 1 destination address is set to "4152ECD7", which is the MAC address of Router 2 instead of the Coordinator's address. We also set the Network ID (discussed in section 4.2) to 123 (original ID) since we notice the ID parameter got scrambled after we set the new destination address. After that, "ATWR" is sent to save all configuration parameters to Xbee. It ends with "ATCN" to get out of configuration mode.

WiFi module start-up operations (reset, connect to household WiFi) and “ngrok” set up on Router 1 are executed in “setup()” phase of the code. Router 1 will effectively be the new Coordinator, but it still operates in transparent mode for the simplicity sake. Therefore, it will send a 5-bytes request of command to Router 2 instead of 23-bytes message as of API mode. All of Router 1 messages to Router 2 are kept identical to Coordinator message (5-byte at the end before the Check sum byte).

Moreover, Router 1 updates data on a different JSON file (“light_data2.json”) on the server. When the server detects an increase in size of this JSON file, it turns into WiFi failure mode, and operate with “ngrok” address of Router to plot data. It also sends an email notification to user indicating there is a problem with the Coordinator WiFi module.

In a nutshell, Router 1 takes over the responsibilities of Coordinator in WiFi failure mode as it sends occupancy data and temperature data to the server, and delivers commands from server.

6. Deliverables

After successfully implementing every proposed functions and features of the Smart Home System, it is necessary to package our system and build the prototype for demonstration. In this section, we discuss packaging of each module as well as building a house prototype.

6.1. Modules Packaging

For the purpose of deliverable, we design and organize modules in separate boxes depending on the functionalities of the modules. The boxes will then be put in an appropriate position in the house prototype model. The first box contains an ultrasonic sensor and a camera, which will be turned on by the microcontroller regarding to the status of the ultrasonic sensor. Since the transmitter and receiver of the ultrasonic sensor are at the two front holes, we need to design the package so that these two holes are apparent and the sensor works most efficiently. Figure 59 demonstrates our designed box for this module.

Figure 59: Packaging of the camera and ultrasonic sensor modules.

Another module contains the PIR sensor and the light that can be turned on by the microcontroller with detection data from our sensor. Similar to the ultrasonic sensor, the most sensitive part of the sensor is at the tip of the PIR. Thus, we needed to make sure our designed package does not block this part and that it has the clearest viewing angle to acquire reliable results. Figure 60 captures our designed for the packaging of the occupancy sensor.

Figure 60: Packaging of the PIR occupancy sensor.

The last box includes the coordinator, which is the coordinator Arduino to receive data from other microcontrollers via ZigBee communication protocol and upload to server via WiFi. Unlike the ultrasonic sensor, the PIR sensor and the camera, which needs clear view for the sensor parts, the coordinator, the ZigBee and the WiFi module are completely inside box.

6.2. House Model Prototype

After having gone through the designed packaging for each module, we had to make a scaled-down house model prototype for our project demonstration. With the help from Mr. William Appleyard and advice from Professor Ludwig, our house model is made of plain wood and then covered by paint and decoration. Table 6 summarizes the dimensions of our designed house model.

Table 6: Dimension of designed house model prototype

Part	Dimension (inch)
Base	28 x 20
Side view	28 x 12
Front view	20 x 12

Figures 61 to 64 depict different viewing angles of the designed home model prototype for our system.

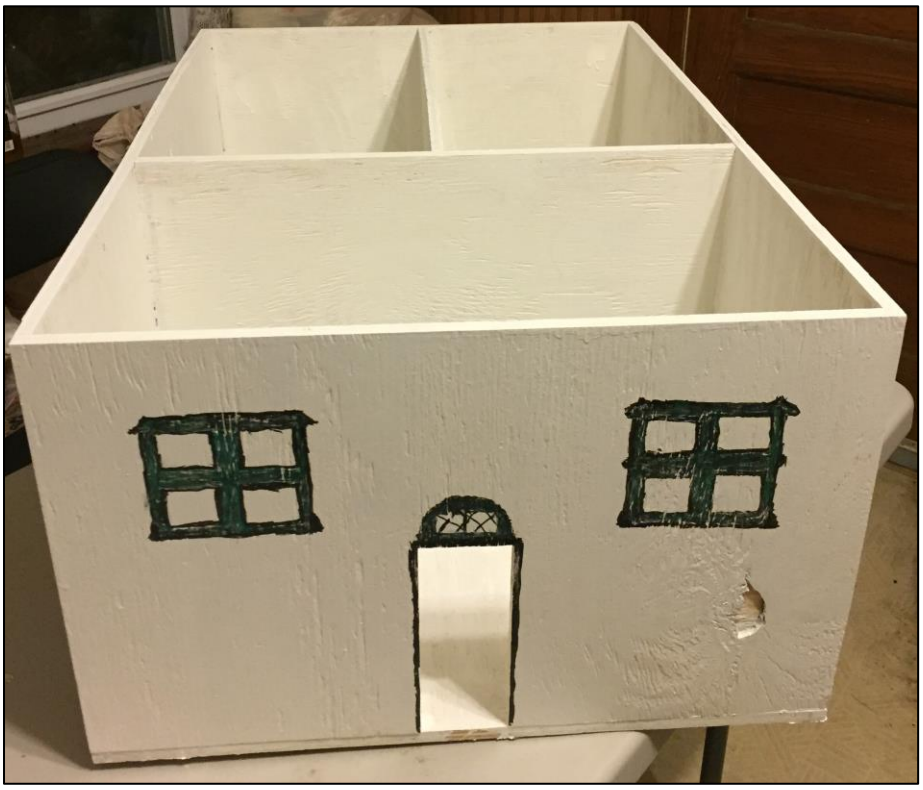

Figure 61: Front-view of the house prototype.

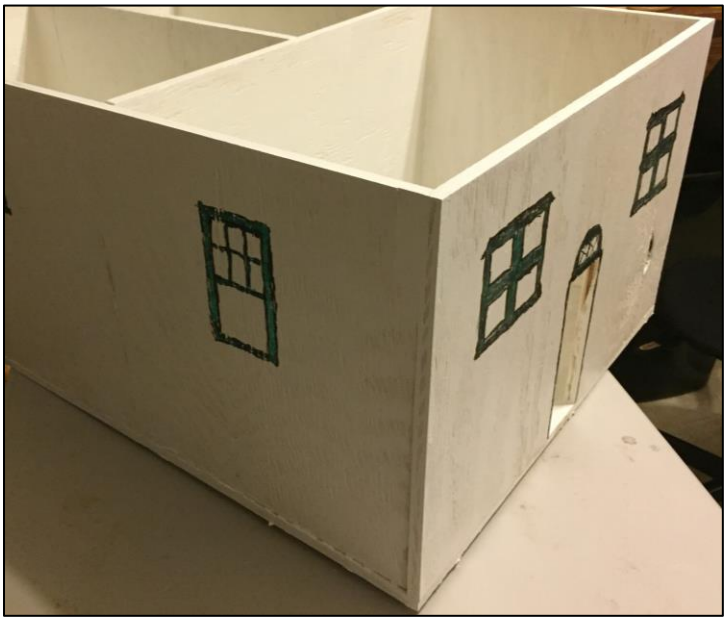

Figure 62: Side-view of the house prototype.

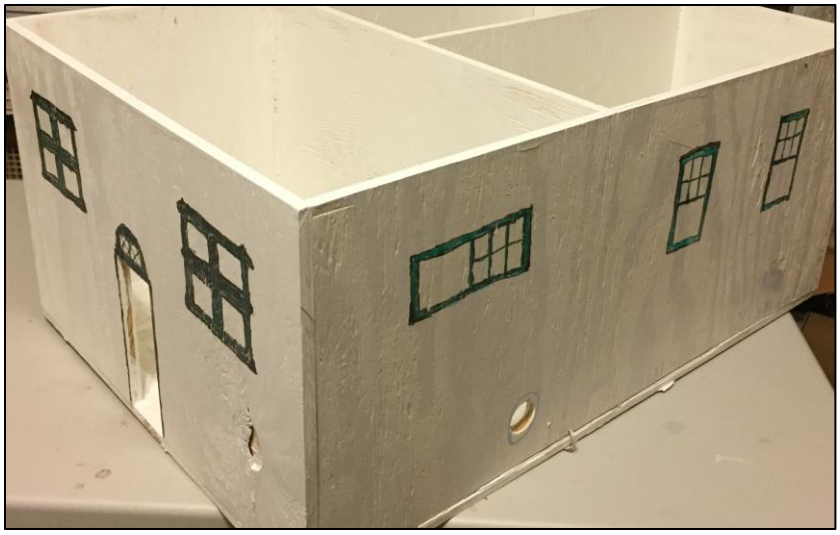

Figure 63: Another side-view of the house prototype.

Figure 64: Down-view of the house prototype.

As seen in Figure 64, the house model is divided into three separate rooms, each of which will contain a box of different modules described previously in section 6.1 to demonstrate the corresponding functionalities of the project.

7. Discussions and Future Works

7.1. Initial Cost

This section describes the components and modules that we decide to purchase for our project, including components, quantities, sources to buy from, and prices. For future work, when the system is ready to be mass-produced, the total cost of the required materials as well as the cost for the entire product is necessary to be re-considered by analyzing more deeply the supply market, and investigating our return on investment and break-even cost point. For the scope of our project, we would like to concentrate more on the technical implementation and, hence, we will not go into details of the cost analysis. Table 7 summarizes the components and modules we purchased.

Table 7: Components List

Components	Quantity	Source	Price ($) per unit	Total Price
ESP8266 Wi-Fi Module	3	Sparkfun	6.95	20.85
Programmable Xbee Zigbee	5	Digikey	20.25	101.25
Arduino Mega 2560	3	Sparkfun	45.95	137.85
Bidirectional Logic Level Converter	3	Sparkfun	2.95	8.85
SparkFun XBee Explorer USB	1	Sparkfun	24.95	24.95
SparkFun XBee Explorer Regulated	3	Sparkfun	9.95	29.85
Break Away Headers - 40-pin Male	3	Sparkfun	0.75	2.25
Adafruit PIR Sensor	1	Adafruit	9.95	9.95
Ultrasonic Sensor HC-SR04	1	Amazon	5.2	5.2
TMP 36 Temperature Sensors	3	Digikey	1.45	4.35
USB Cable A to B - 6 Foot	4	Sparkfun	3.95	15.8
Password Protected Web Server	1	Random Nerd	11.33	11.33
Shipping Charges	1	Sparkfun		29.4
Shipping Charges	2	Digikey	20.99	41.98
Total				**443.86**

7.2. Safety Analysis

As stated earlier, because of time constraint and budget limitations, we are not able to deliver a fully-optimized, commercial version of our proposed smart home system. Instead, as originally proposed, we have successfully designed, tested and built a fully-functional product prototype that will be used to demonstrate our prototype architecture. As a result, in this section, we discuss the safety aspects of our proposed system as well as current functionalities and features that we implemented. One of the main features in our smart home system that makes us

different from other existing products is the low power consumption and environmental friendliness. We chose energy-efficient Arduino microcontrollers and a low-power short-range ZigBee communication protocol. As a result, even though all the modules are put in a plastic cover, they are highly reliable and they are unlikely to overheat or overpower the system. As safety for users and home owners are always our first priority, we also implemented a notification system to alert the users and home owners if a dangerous overheating situation is detected. This is done by sending a critical warning to the users' email address immediately. This could help alert the users to perform certain immediate actions to cease the danger. Moreover, if the situation of abnormal high temperature is identified, the microcontrollers will turn off all the appliances as well as the sensors to prevent them from being damaged. Figure 65 shows a warning email which is sent to the user's email address whenever an overheating situation is detected.

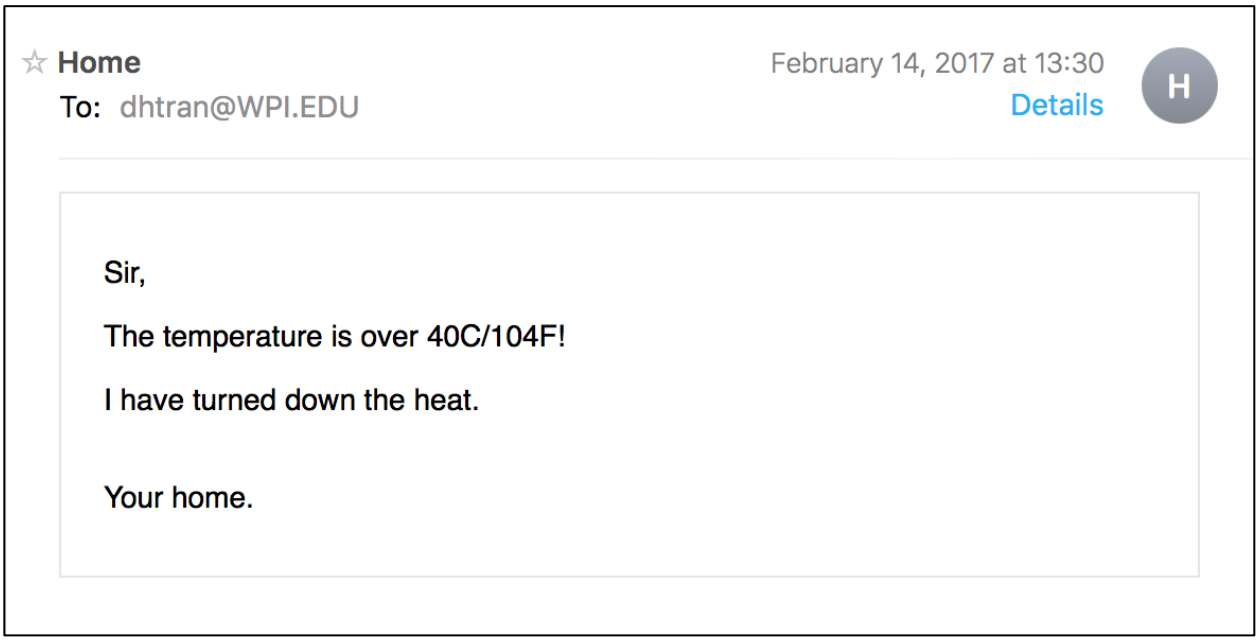

Figure 65: Email warning of indoor critical temperature detected.

Also, for the future work, besides developing and extending on the hardware, we should focus more on the security and provide safety for the user's home information in the server. As the world of Internet of Things is growing exponentially, the server will be able to verify the credentials and privacy of home owners to prevent hackers and intruders. This will be discussed further in the section 7.3.

7.3. Privacy and Security

Privacy and security are important factors in every Internet of Things system. Since there are many inter-communications present at all different levels of an IoT system, there is a high risk that an attacker/eavesdropper can observe/modify/corrupt the messages being sent. The attacker can also retrieve important information, such as WiFi password, or even take control of the entire system by infiltrating to the IoT devices and installing their viruses. These risks are all realistic, even in the most advanced IoT devices from the most reputable corporations. An

example is an experiment where researchers successfully implanted a virus into the Phillip Hue light bulbs system by using a drone flying outside the building [47].

For that reason, privacy and security is also one of the most important concerns for our Smart Home IoT project. Unfortunately, due to the timing constraint and our inexperience in security protocols, we do not implement any specific protection protocol for our system. However, we tried to the best of our capabilities to work around this limitation, by choosing network components that are packaged with secured communication protocols. For the coordinator-server communication we use an ESP8266 WiFi module, which is connected to our local WiFi router and therefore protected since WiFi is encrypted. To receive commands from the web server, we use ngrok, an application that guarantee to provide a secure TCP tunnel to the local IP address of the coordinator module. For the web server, we use 000webhost, a free online webhost provider that also guarantees users with SSL certificate, the standard security technology for establishing an encrypted link between a web server and a browser. And lastly, for coordinator-routers communication, we used Xbee ZigBee from Digi International, which is protected by using the AES encryption standard with a 128-bit key, as the ZigBee protocol is based on IEEE 802.15.4 network standard.

We have to admit that our system is still insecure, as all components listed in the previous paragraph are either free open source (and therefore not very reliable) or have some known security issues (such as the ZigBee protocols). Also, we do not have any encryption for the messages, they are thus vulnerable to attackers if they can exploit the ZigBee or WiFi communications. For future development, we plan to have a more secur and reliable webhost provider. We also need a replacement for ngrok since it is an open source project, which could be vulnerable to hackers since it is freely available from GitHub. Lastly, we need to implement some level of encryption for our messages, as well as for critical information such as the WiFi name, password, or the TCP URL in our coordinator.

7.4. Scalability

This section focuses on the ability to be changed in size or scale of the Smart Home System Prototype. In order to successfully implement the system for a reasonable sized house, the following aspects must be taken into consideration.

7.4.1. Transmission and server robustness

As previously discussed in section 5.1, our web server is unstable since it is built around a free resource. An unstable web server is undesirable because users can possibly lose control of their houses. A better server, which is guaranteed to be alive 100% of the time and has data saving feature, is necessary in order to scale up the Smart Home System.

7.4.2. Variety of sensing mechanisms

Given the scope of our MQP, we only implemented three modes of sensing: occupancy, temperature, and camera. A desire Smart Home must have all data about its state available to its user: water leakage, humidity condition; break-in detection, etc. Thus, one possible future guideline for this MQP is to implement different sensors, which open up many interesting applications. On the other hand, the quantity of single sensing mode also has to increase. For

example, each room needs a temperature sensor, occupancy sensor, or security camera, in order to present full state of its condition to users.

7.4.3. Better User Interface and Easier Integration Scheme

An easier to use, more friendly user interface and integration scheme is desired as a means to implementing new modules. Unfortunately, our software solution currently needs custom script for each module, which scales up the amount of work significantly as the number of modules increases. A semi-automated integration method, which collects user need for the module (set up as coordinator or router, create undefined task, etc.) but automatically gets configuration parameters from Xbee connect them to the entire Smart Home Network, will greatly improve the user experience and help the user smoothly transit into an automated house.

8. Summary

As engineering students who are strongly interested in the development of technology, we are driven by the concept of Internet of Things and their expansion to smart devices that make homes safer, more secure, less power consuming, and more environmentally friendly. In the above sections, we have defined a smart home system. We went through the need for a smart home system, the state of the art of existing products in the market and their limitations. We also identified different technologies and methods that are currently researched and developed, and selected the most feasible and applicable for our system. We also came up with a proposed architecture and design plan, which discuss the advantages over existing products and how they will be better solutions compared to the current state of the art.

For our project, we have first successfully implemented the Sensor Area Network. The system contains three different nodes in the network, with two slave nodes and one master node. Each node has an Arduino microcontroller, a ZigBee module, a Wi-Fi module, a variety of sensors, and an appliance. The microcontroller is responsible for controlling and fetching data from the sensors and appliances. The data is transferred either by ZigBee to the master node, or directly from the master node to our web server by Wi-Fi. Commands from the server is sent to the master node by Wi-Fi, and is then distributed by the master node to the slave nodes, where the commands will be executed by the microcontroller to control the appliances. For the appliances, as discussed in the earlier, we do not use the real appliances for our prototype. Instead, we just mimic the real appliances with scaled products by utilizing small LED diodes instead of real LED light bulbs, an on-board camera module instead of a real security camera and a small fan module as a factor to adjust the temperature sensor status.

Second, we have successfully developed the Web Server. The web server keeps its role as the gateway for communication between users and our system. Every data obtained from our Sensor Area Network is sent directly to the web server. Every command from the users is transferred to the web server before reaching our master node in the Sensor Area Network. Besides acting as data storage, the web server functions as our software back end. Data processing, algorithms for handling worst-case scenario with our system, the command structure, algorithm for different functions that we want to include in our prototype (such as scheduling, hazard mode, etc.), are implemented completely on the web server.

Third, and as mentioned previously, a Web-based module for the PC or Smartphone User Interface becomes necessary for users to easily view the current status and control of their home. The control of our final prototype is done remotely by our self-developed user interface. Because of the time constraint, we have only developed a web-based user interface on a PC (a future user interface for a smartphone could be one of the primary development goals). From the user interface (UI), users should be able to access and view data on the web server in real time and remotely control each appliance module as well as the entire system. The UI is developed and designed in a user-friendly, easy-to-use, interactive way. Useful data from the system is presented in a graphical way for better visualization and interpretation of data. Controlling each module is simplified as much as possible in the UI, with straightforward instructions that non-technical users could easily use in their first try.

References

[1] Gartner, http://www.gartner.com/newsroom/id/3165317

[2] Wikipedia, https://en.wikipedia.org/wiki/Internet_of_things#Early_history

[3] LIFX Smart Bulb embedded components, http://www.arm.com/innovation/products/lifx.php

[4] NEST Thermostat, https://nest.com/thermostat/meet-nest-thermostat/

[5] Insteon Home Automation System, http://www.insteon.com/insteon-hub/

[6] Insteon, *http://www.dirnan.com/controlling-and-automating-pool-attachments-with-insteon/*

[7] Reuter, *http://www.cse.fraunhofer.org/recent-news*

[8] Efficiency Vermont, www.efficiencyvermont.com

[9] 11 Internet of Things Network Protocol, http://www.rs-online.com/designspark/electronics/knowledge-item/eleven-internet-of-things-iot-protocols-you-need-to-know-about

[10] Bluetooth Low Energy in Internet of Things, https://www.linkedin.com/pulse/what-bluetooth-low-energy-means-internet-things-premaratne

[11] ZigBee Wireless Standard, http://www.digi.com/resources/standards-and-technologies/rfmodems/zigbee-wireless-standard

[12] ZigBee Digi, *http://www.digi.com/*

[13] Z-Wave pros and cons https://iotee.wordpress.com/tag/z-wave-pros-and-cons/

[14] ZigBee Vs 6LowPan for sensor networks, https://www.lsr.com/white-papers/zigbee-vs-6lowpan-for-sensor-networks

[15] Low power Wi-Fi (Wi-Fi Halow) for Internet of Things, http://www.wired.com/2016/01/Wi-Fi-halow-internet-of-things/

[16] Arduino, *https://www.arduino.cc/en/Main/ArduinoBoardMega2560*

[17] TI MSP432 Development Board, *http://www.ti.com/ww/en/launchpad/launchpads-msp430-msp-exp432p401r.html*

[18] Raspberry Pi, *https://www.adafruit.com/products/3055?gclid=COfr9YHym88CFcsehgodnTYEMA*

[19] PIR sensor, *https://www.adafruit.com/product/189*

[20] Ultrasonic sensor, *https://www.amazon.com/Arrela%C2%AE-Hc-sr04-Ultrasonic-DistanceMeasuring/dp/B00KKKT7YK*

[21] ArduCam, *http://www.arducam.com/arducam-mini-released/arducam_mini-1/*

[22] Raspberry Pi NoIR, *http://www.geeky-gadgets.com/raspberry-pi-noir-camera-version-1-and-2-compared-06-06-2016/*

[23] Pixy CMU Cam5, *https://www.adafruit.com/products/1906?gclid=COGX9-Hcoc8CFUk8gQodbNsB3Q*

[24] Zigbee Xbee, https://www.digi.com/products/xbee-rf-solutions/embedded-rf-modules-modems/digi-xbee-zigbee

[25] Zigbee Xbee Explorer USB, https://www.sparkfun.com/products/11697

[26] 9 ways a smart home can improve people's life
https://blog.smartthings.com/news/roundups/9-ways-a-smart-home-can-improve-your-life/

[27] "*Automatic protocol configuration for dependable internet of things application*", Felix Jonathan Oppermann, Carlo Alberto Boano, Marco Antonio Zýýiga

[28] "*A survey based on Smart Homes system using Internet-of-Things*", Pranay P. Gaikwad, Jyotsna P. Gabhane, Snehal S. Golait

[29] ZigBee vs Z-Wave vs Insteon Home Automation Protocols Comparison
http://www.digitaltrends.com/home/zigbee-vs-zwave-vs-insteon-home-automation-protocols-explained/

[30] Smart Home Challenges and Approaches to Solve Them,
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.455.4469&rep=rep1&type=pdf

[31] Electric Usage Chart Tool
https://www.efficiencyvermont.com/tips-tools/tools/electric-usage-chart-tool

[32] Angular Chart
http://jtblin.github.io/angular-chart.js/

[33] Mechanical Temperature Sensor - Thermometer
https://en.wikipedia.org/wiki/Thermometer

[34] Thermistor
https://en.wikipedia.org/wiki/Thermistor#/media/File:NTC_bead.jpg

[35] Analog-output temperature sensor TMP36
https://www.sparkfun.com/products/10988

[36] ArduCam Mini Camera

ArduCAM_Mini_2MP_Camera_Shield_Hardware_Application_Note, www.arducam.com

[37] ESP8266 WiFi Module, www.sparkfun.com

[38] Ultrasonic Wave Terminology, http://sensorwiki.org/lib/exe/fetch.php/sensors/ultrasound_echo_ranging.jpg?w=&h=&cache=cache

[39] Sparkfun Zigbee Explorer Regulated, https://www.sparkfun.com/products/11373

[40] TMP36 Analog Devices, http://www.analog.com/media/en/technical-documentation/data-sheets/TMP35_36_37.pdf

[41] Digi Xbee Knowledge Article, http://knowledge.digi.com/articles/Knowledge_Base_Article/Interact-with-Programmable-XBee-using-XCTU/?q=programmable&l=en_US&fs=Search&pn=1

[42] Digi Xbee User Manual, http://www.digi.com/resources/documentation/digidocs/pdfs/90002002.pdf

[43] ESP 8266 guide, techtutorialsx.wordpress.com/2016/02/28/esp8266-uploading-code-from-arduino-ide/

[44] Sparkfun Logic Level Converter, https://www.sparkfun.com/products/12009

[45] Esphant-Arduino Library, https://libraries.io/github/laCour/esphant-arduino

[46] AT Command Lists, https://room-15.github.io/blog/2015/03/26/esp8266-at-command-reference/

[47] Watch a drone hack a room full of smart lightbulbs from outside the window, http://www.theverge.com/2016/11/3/13507126/iot-drone-hack

Appendices

This section discusses the technical challenges that we have dealt, either solved or not, throughout the project for future review and experiences. At the end, we also provided the entire code for the system, including implementation of each module, communication protocols, integration of the system as well as the server and user interface design.

Real-time data display for user interface:

One of the most important features for user interface in our project is to represent sensors data in a useful way. Users should be able to view current status and control their homes via our real-time applications. Whenever the sensor data change, including the temperature sensor, light sensor and camera, it should be reflected in the front-end website, or the user interface. For data received from light sensor and temperature sensor, which show the current lighting luminosity and indoor temperature, we decide to plot the data in two graphs, respectively that should be updated automatically. For the camera, we show the captured image whenever the server received an updated picture. There exist different JavaScript frameworks and libraries to represent the data in graph such as FusionChart and Angular Chart. Both of the libraries are able to plot the data in a neat, eye-catching graph. However, only Angular Chart provides responsive and reactive charts. Therefore, we chose Angular Chart, an extension of AngularJS, to represent the data for real-time applications, the sensor data is then changed according to the received data from the microcontroller and the image is also updated automatically.

Complete Code:

Appendix 1: Server/Website User Interface

Index.html:

```
<!DOCTYPE html>
<html>
<script async defer
src="https://maps.googleapis.com/maps/api/js?key=AIzaSyCfgDOyQX3Ou8wA0VPCUgZ475uvm54FN2c&callback=initMap"></script>
<head>
    <title>MQP website</title>
    <link rel="stylesheet" type="text/css" href="w3.css">
    <link rel="stylesheet" type="text/css"
href="https://fonts.googleapis.com/css?family=Raleway">
    <link rel="stylesheet" type="text/css" href="main.css">
</head>
<body>
<!-- Navigation bar -->
<div class="w3-top">
    <ul class="w3-navbar" id="myNavBar">
            <li class="w3-right w3-hide-small">
                    <a href="#about">ABOUT</a>
                    <a href="#feature">FEATURES</a>
                    <a href="#team">OUR TEAM</a>
                    <a href="#contact">CONTACT US</a>
                    <a href="live_data.html">DASHBOARD</a>
            </li>
    </ul>
</div>

<!-- Header image -->
<header class="bgimg-1">
</header>

<h1 class="MQP">MAJOR QUALIFYING PROJECT</h1>
<h1 class="w3-opacity">Smart Home System - AY 2016-2017</h1>

<!-- ABOUT section -->
<div>
    <div class="w3-container w3-padding-64" id="about">
    <h2 style="text-align: center; padding-bottom: 3%">FUTURE HOME</h2>
            <div class="quarter w3-margin-bottom">
                    <div class="w3-card-2">
                            <img
src="https://d30y9cdsu7xlg0.cloudfront.net/png/74467-200.png" style="padding-top: 9%" class="about-col">
                            <p style="text-align: center; font-size: 150%;
padding-bottom: 12%">Automatic</p>
```

```
 39                          <!-- <p class="w3-opacity" style="text-align:
center;"></p> -->
 40                  </div>
 41              </div>
 42              <div class="quarter w3-margin-bottom">
 43                  <div class="w3-card-2">
 44                          <img src="houseplant.png" style="padding-top:
9%" class="about-col">
 45                          <p style="text-align: center; font-size: 150%;
padding-bottom: 12%">Environmental Friendly</p>
 46                          <!-- <p class="w3-opacity" style="text-align:
center;">Write some description here</p> -->
 47                  </div>
 48              </div>
 49              <div class="quarter w3-margin-bottom">
 50                  <div class="w3-card-2">
 51                          <img src="lock a.png" style="padding-top: 9%"
class="about-col">
 52                          <p style="text-align: center; font-size: 150%;
padding-bottom: 12%">Reliable Secure</p>
 53                  </div>
 54              </div>
 55              <div class="quarter w3-margin-bottom">
 56                  <div class="w3-card-2">
 57                          <img src="power.png" style="padding-top: 9%"
class="about-col">
 58                          <p style="text-align: center; font-size: 150%;
padding-bottom: 12%">Energy Efficient</p>
 59                  </div>
 60              </div>
 61      </div>
 62 </div>
 63 <!-- ENDING ABOUT SECTION -->
 64
 65 <!-- FUNCTIONALITY -->
 66 <div style="padding-bottom: 4%">
 67      <div class="w3-container w3-padding-64" id="feature">
 68      <h2 style="text-align: center;">CURRENT FEATURES</h2>
 69      <p class="w3-opacity" style="text-align: center; font-size: 150%">More
features are coming!</p>
 70              <div class="third w3-margin-bottom">
 71                  <a href="live_data.html#light">
 72                          <img src="bulb.png" class="about-col">
 73                  </a>
 74                  <p style="text-align: center; font-size:
150%">Lighting</p>
```

```
 75             </div>
 76             <div class="third w3-margin-bottom">
 77                     <a href="live_data.html#temp">
 78                             <img src="temperature.png" class="about-col">
 79                     </a>
 80                     <p style="text-align: center; font-size:
150%">Temperature</p>
 81             </div>
 82             <div class="third w3-margin-bottom">
 83                     <a href="live_data.html#camera">
 84                             <img src="eye.png" class="about-col">
 85                     </a>
 86                     <p style="text-align: center; font-size:
150%">Camera</p>
 87             </div>
 88     </div>
 89 </div>
 90 <!-- ENDING FUNCTIONALITY -->
 91
 92 <!-- TEAM SECTION -->
 93 <div id="team" style="padding-bottom: 10%">
 94     <h3>WE ARE</h3>
 95     <div class="w3-row-padding" style="margin-top: 64px" >
 96             <div class="third w3-margin-bottom">
 97                     <div class="w3-card-2">
 98                             <img
src="https://upload.wikimedia.org/wikipedia/en/1/1b/WPI_logo.png"
style="width: 80%; padding-left: 20%; padding-top: 3%">
 99                             <div>
100                                     <h3>Duc Tran</h3>
101                                     <p class="w3-opacity"">Electrical
Engineer</p>
102                             </div>
103                     </div>
104             </div>
105             <div class="third w3-margin-bottom">
106                     <div class="w3-card-2">
107                             <img
src="https://upload.wikimedia.org/wikipedia/en/1/1b/WPI_logo.png"
style="width: 80%; padding-left: 20%; padding-top: 3%">
108                             <div>
109                                     <h3>Anh Tran</h3>
110                                     <p class="w3-opacity">Electrical
Engineer</p>
111                             </div>
112                     </div>
```

```
113             </div>
114             <div class="third w3-margin-bottom">
115                     <div class="w3-card-2">
116                             <img
src="https://upload.wikimedia.org/wikipedia/en/1/1b/WPI_logo.png"
style="width: 80%; padding-left: 20%; padding-top: 3%">
117                             <div>
118                                     <h3>Thinh Ly</h3>
119                                     <p class="w3-opacity">Electrical
Engineer</p>
120                             </div>
121                     </div>
122             </div>
123     </div>
124 </div>
125 <!-- ENDING TEAM SECTION -->
126
127 <!-- CONTACT SECTION -->
128 <div style="height: 50%; width: auto;" id="contact">
129     <h3>CONTACT US</h3>
130     <!-- <div class="w3-row-padding" style="margin-top: 64px"
id="contact"> -->
131     <div style="height: 80%; margin: 0; padding: 0;">
132             <div id="myMap" style="height: 100%;"></div>
133     </div>
134 </div>
135 <!-- ENDING CONTACT SECTION -->
136
137 <!-- FOOTER -->
138 <footer class="w3-center" style="padding-bottom: 3%;">
139     <div style="text-align: center;">
140             <button class="btn" type="button" style="margin-right: 1%;"><a
href="index.html">Go to top</a></button>
141             <button class="btn" type="button"><a
href="live_data.html">DASHBOARD</a></button>
142     </div>
143 </footer>
144 <!-- ENDING FOOTER -->
145
146 <script>
147     // Add Google Map into Contact Section
148     window.initMap = function() {
149     // function initMap() {
150             var myCenter = new google.maps.LatLng(42.274148,-71.808372);
        // set location to WPI
151             var mapProp = {
```

```
152                 center: myCenter,
153                 zoom: 15,
154                 scrollwheel: false,
155                 draggable: true,
156                 mapTypeId: google.maps.MapTypeId.ROADMAP
157               };
158
159               var map = new
google.maps.Map(document.getElementById("myMap"),mapProp);
160
161               var marker = new google.maps.Marker({
162                 position: myCenter,
163                 map: map
164               });
165
166               marker.setMap(map);
167               // alert("OK");
168     }
169     // google.maps.event.addDomListener(window, 'load', initMap);
170 </script>
171 </body>
172 </html>
```

live_data.html:

```
  1 <!DOCTYPE html>
  2 <html ng-app="chartApp">
  3 <!-- AngularJS Library -->
  4 <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.8/angular.min.js"></
script>
  5
  6 <!-- jQuery Library -->
  7 <script type="text/javascript" src="jquery.min.js"></script>
  8
  9 <!-- Angular Chart Library -->
 10 <script src="Chart.js"></script>
 11 <script src="angular-chart.js"></script>
 12 <!-- <script src="angular-chart.min.js"></script> -->
 13
 14 <head>
 15     <title>DASHBOARD</title>
 16     <link rel="stylesheet" type="text/css" href="w3.css">
 17     <link rel="stylesheet" type="text/css" href="live_data.css">
 18     <link rel="stylesheet" type="text/css"
href="https://fonts.googleapis.com/css?family=Raleway">
 19 </head>
 20 <body>
```

```
 21
 22 <!-- TOP CONTAINER -->
 23 <div class="w3-container w3-top w3-light-grey w3-large w3-padding">
 24     <span class="topContainer">My Dashboard</span>
 25 </div>
 26 <!-- Navigation Bar Menu -->
 27 <div class="w3-top">
 28     <ul class="w3-navbar" id="myNavBar">
 29             <li class="w3-right w3-hide-small">
 30                     <a href="index.html">HOME</a>
 31             </li>
 32     </ul>
 33 </div>
 34 <!-- ENDING TOP CONTAINER -->
 35
 36 <header>
 37     <img src="https://www.soti.net/media/5694/internet-of-things.jpg"
style="width: 100%; height: auto; margin-top: 3%;">
 38 </header>
 39
 40 <!-- SIDE MENU -->
 41 <nav class="w3-sidenav w3-collapse w3-white w3-animate-left"
style="width: 250px; z-index: 3; margin-top: 10px;">
 42     <h3 style="font-size: 200%; padding-left: 80px;">Menu</h3>
 43     <a href="#light" class="w3-hover-black" style="font-size: 120%;
padding-left: 10px;">Lighting</a>
 44     <a href="#temp" class="w3-hover-black" style="font-size: 120%;
padding-left: 10px;">Temperature</a>
 45     <a href="#camera" class="w3-hover-black" style="font-size: 120%;
padding-left: 10px;">Camera</a>
 46 </nav>
 47 <!-- ENDING SIDE MENU -->
 48
 49 <!-- MAIN CONTENT -->
 50 <div class="w3-main" style="margin-left: 250px; margin-top: 1px">
 51     <p ng-controller="chartController" id="ngrokIDTesting"
onclick="ngrokChange()"></p>
 52
 53     <div class="w3-container w3-padding-64" id="light">
 54             <h2 style="text-align: center; font-size: 240%;">Lighting</h2>
 55             <div class="c1-box" ng-controller="chartController">
 56                     <canvas class="chart chart-line" chart-data="lightData"
chart-labels="lightLabels" chart-options="options" chart-
colors="lightColors"></canvas>
 57             </div>
```

```
 58             <div class="c2-box" ng-controller="chartController"
style="padding-top: 10%">
 59                     <h3 style="font-size: 200%">Light Control</h3>
 60                     <button id="1" class="led lightButton brightLi">Light
1</button>
 61                     <button id="2" class="led lightButton brightLi">Light
2</button>
 62             </div>
 63     </div>
 64
 65     <div class="w3-container w3-padding-64" id="temp">
 66             <h2 style="text-align: center;font-size:
240%">Temperature</h2>
 67             <div class="c1-box" ng-controller="chartController">
 68                     <canvas class="chart chart-line" chart-data="tempData"
chart-labels="tempLabels" chart-options="options" chart-
colors="tempColors"></canvas>
 69             </div>
 70             <div class="c2-box" ng-controller="chartController">
 71                     <h3 style="text-align: center; font-size:
200%;">Current Temperature</h3>
 72                     <h1 style="font-size: 300%">{{nowTemp}}</h1>
 73                     <p style="text-align: center; font-size: x-
large;">Celcius Degree</p>
 74                     <form name="tempForm" onsubmit="return validateForm()">
 75                             <input name="tempInput" id="temp" class="temp
tempForm" placeholder="Temperature Control" style="font-size: 150%;text-
align: center;"><br><br>
 76                             <input type="submit" id="submit0"
class="submit0 submitButton">
 77                     </form>
 78             </div>
 79     </div>
 80
 81     <div class="w3-container w3-padding-64" id="camera">
 82             <h2 style="text-align: center;">Camera</h2>
 83             <div class="c1-box" ng-controller="chartController">
 84                     <img ng-src="{{cameraURL}}"/>
 85             </div>
 86             <div class="c2-box" style="padding-top: 10%;">
 87                     <!-- <h3>Camera</h3> -->
 88                     <img id="4" src="camera-icon.png" class="camImg">
 89                     <p class="subInfo">Click on camera to update</p>
 90             </div>
 91     </div>
 92 </div>
```

```
 93 <!-- ENDING MAIN CONTENT -->
 94
 95 <script src="live_data.js"></script>
 96 <script>
 97     document.getElementById("ngrokIDTesting").innerHTML = ngrokID;
 98 </script>
 99 <script>   // Function to send warning email if overheating detected
100    function sendEmail() {
101             emailjs.send("default_service","template_iPWhGaKg",{name:"DT",
notes:"Test Email"}).then(function(response) {
102                     console.log("SUCCESS. status=%d, text=%s",
response.status, response.text);
103             }, function(err) {
104                     console.log("FAILED. error=", err);
105             });
106    };
107 </script>
108 <script>    // Verify temperature input from user and update temp
109     function validateForm() {
110             var desiredTemp =
document.forms["tempForm"]["tempInput"].value;
111             // console.log(isNaN(desiredTemp));
112             if (isNaN(desiredTemp)==true) {
113                     alert("Enter temperature in number");
114             } else if((desiredTemp<5) || (desiredTemp>30)) {
115                     alert("Select temperature between 5 and 30");
116             } else {
117                     $(document).ready(function() {
118                       $(".submit0").unbind().click(function() {
119                       desiredTemp =
document.forms["tempForm"]["tempInput"].value;
120                                 //alert(desiredTemp);
121                           $.get("http://0.tcp.ngrok.io:16499",
{pin:desiredTemp}); // execute get request
122
//document.forms["tempForm"]["tempInput"].value = "";
123                       });
124                     });
125             };
126     };
127 </script>
128
129 <script>// Update camera
130     $(document).ready(function() {
131             $("#4").click(function() {
132                     var p = $(this).attr('id');
```

```
133             $.get("http://0.tcp.ngrok.io:16499", {pin:p});
134             });
135     });
136 </script>
137
138 <script>// Update Mode
139     $(document).ready(function() {
140             $("#5").click(function() {
141                     var p = $(this).attr('id');
142             $.get("http://0.tcp.ngrok.io:16499", {pin:p});
143             });
144     });
145 </script>
146
147 <script>
148     $(document).ready(function() {
149             var isClicked1 = 0;
150             $("#1").click(function() {
151                     isClicked1++;
152                     var p = $(this).attr('id');
153                     $.get("http://0.tcp.ngrok.io:16499", {pin:p});
154                     // console.log((isClicked1%2));
155                     if ((isClicked1%2)==1) {                                    // Lig
156                     $("#1").removeClass("brightLi");
157                     $(this).addClass("brightLiOnClick");
158             } else {
159                     $("#1").removeClass("brightLiOnClick");
160                     $(this).addClass("brightLi");
161                     };
162             });
163     });
164     $(document).ready(function() {
165             var isClicked2 = 0;
166             $("#2").click(function() {
167                     isClicked2++;
168                     var p = $(this).attr('id');
169                     $.get("http://0.tcp.ngrok.io:16499", {pin:p});
170                     // console.log((isClicked1%2));
171                     if ((isClicked2%2)==1) {                                    // Lig
172                     $("#2").removeClass("brightLi");
173                     $(this).addClass("brightLiOnClick");
174             } else {
175                     $("#2").removeClass("brightLiOnClick");
176                     $(this).addClass("brightLi");
177                     };
178             });
```

```
179     });
180 </script>
181 </body>
182 </html>
```

live_data.js:

```
var chartApp = angular.module("chartApp",["chart.js"]);
chartApp.controller('chartController', [
    '$scope',
    '$interval',
    '$http',
    '$timeout',
    function($scope, $interval, $http, $timeout) {
            // This testURL file is using to detect the disconnection of
the master coordinator to WiFi.
            // If so, it will switch to another JSON file uploaded by
other Arduino
        var testURL =
"/Users/dhtran/Desktop/MQP/Website/script/services/test.js" + "?" + new
Date().getTime();
        $scope.getTest = function() {
            $http.get(testURL,{
                cache: false
            }).then(function(response) {      // function handles SUCCESS
                testURL =
"/Users/dhtran/Desktop/MQP/Website/script/services/test.js" + "?" + new
Date().getTime();
                $scope.content = response.data;
                if(response.data.length == 1) {             // WiFi is in
GOOOD condition
                    $scope.ngrokID = "WiFi is good"
                    // console.log(ngrokID);

                    // LIGHT
                    // Access external JSON file for light data
                    var lightURL = "lightData.json" + "?" + new
Date().getTime();
                    $scope.getLight = function() {
                        $http.get(lightURL,{
                            cache: false
                        }).success(function(result, status, headers,
config) {
                            lightURL = "lightData.json" + "?" + new
Date().getTime();
                            val = [];
                            var index = 0;
                            for (var i = result.length - 1; i >= 0; i--) {
                                val[index] = result[i].value;
                                index++;
                            };
                            $scope.lightData = [val];
```

```
 36                          });
 37                      };
 38                      $scope.getLight();
 39                      // Update light data
 40                      $interval(function(){
 41                          var x = $scope.lightData[0].shift();
 42                          $scope.lightData[0].push(x);
 43                      }, 1000);
 44                      // Update JSON light
 45                      $interval(function(){
 46                          $scope.getLight();
 47                      },10000);
 48                      $scope.lightLabels = ["8' ago","7' ago","6' ago","5'
ago","4' ago","3' ago","2' ago","1' ago","Now"];
 49                      $scope.lightColors = [
 50                          "#ff9932"       // orange
 51                      ];
 52
 53                      // TEMPERATURE
 54                      // Access external JSON file for temperature data
 55                      var tempURL = "MotionSensor.json" + "?" + (new
Date()).getTime();
 56                      $scope.getTemp = function() {
 57                          $http.get(tempURL,{
 58                              cache: false
 59                          }).success(function(data, status, headers, config)
{
 60                              tempURL = "MotionSensor.json" + "?" + (new
Date()).getTime();
 61                              val = [];
 62                              var index = 0;
 63                              for (var i = data.length - 1; i>=0; i--) {
 64                                  val[index] = data[i].value;
 65                                  index++;
 66                              }
 67                              $scope.tempData = [val];
 68                          });
 69                      };
 70                      $scope.getTemp();
 71                      // Update data in real-time
 72                      $interval(function() {
 73                          var x = $scope.tempData[0].shift();          //
Remove the FIRST element in array
 74                          $scope.tempData[0].push(x);                  //
Push the element to the END of array
 75                          $scope.nowTemp = x;
```

```
 76                     }, 1000);                    // Update data every 1
second
 77                     // Update JSON file
 78                     $interval(function() {
 79                         $scope.getTemp();
 80                     },5000);                     // Update JSON every 5
seconds
 81                     $scope.tempLabels = ["6' ago","5' ago","4' ago","3'
ago","2' ago","1' ago","Now"];
 82                     $scope.tempColors = [
 83                         "#43db25"       // greeen
 84                     ];
 85
 86                     // CAMERA
 87                     var imageURL = "CameraTemp.jpg";
 88                     $scope.cameraURL = imageURL + "?" + (new
Date()).getTime();
 89                     // console.log($scope.cameraURL);
 90                     $scope.getImage = function() {
 91                         $http.get($scope.cameraURL, {
 92                             cache: false
 93                         }).success(function(data, status, headers, config)
{
 94                             $scope.cameraURL = "CameraTemp.jpg" + "?" +
(new Date()).getTime();
 95                         });
 96                     };
 97                     $scope.intervalFunction = function() {
 98                         $timeout(function() {
 99                             $scope.getImage();
100                             $scope.intervalFunction();
101                             // console.log('Enter Camera');
102                         }, 5000)
103                     };
104                     $scope.intervalFunction();
105
106
107                 } else {                                                  // WiF
108                     console.log("bad. File length: %d",
$scope.content.length);
109                     $scope.ngrokID = "WiFi is bad";
110
111                 // LIGHT
112                 // Access external JSON file for light data
113                 var lightURL = "lightData.json" + "?" + new
Date().getTime();
```

```
114                 $scope.getLight = function() {
115                     $http.get(lightURL,{
116                         cache: false
117                     }).success(function(result, status, headers, config)
{
118                         lightURL = "lightData.json" + "?" + new
Date().getTime();
119                         val = [];
120                         var index = 0;
121                         for (var i = result.length - 1; i >= 0; i--) {
122                             val[index] = result[i].value;
123                             index++;
124                         };
125                         $scope.lightData = [val];
126                     });
127                 };
128                 $scope.getLight();
129                 // Update light data
130                 $interval(function(){
131                     var x = $scope.lightData[0].shift();
132                     $scope.lightData[0].push(x);
133                 }, 1000);
134                 // Update JSON light
135                 $interval(function(){
136                     $scope.getLight();
137                 },10000);
138                 $scope.lightLabels = ["8' ago","7' ago","6' ago","5'
ago","4' ago","3' ago","2' ago","1' ago","Now"];
139                 $scope.lightColors = [
140                     "#ff9932"       // orange
141                 ];
142
143                 // TEMPERATURE
144                 // Access external JSON file for temperature data
145                 var tempURL = "MotionSensor.json" + "?" + (new
Date()).getTime();
146                 $scope.getTemp = function() {
147                     $http.get(tempURL,{
148                         cache: false
149                     }).success(function(data, status, headers, config) {
150                         tempURL = "MotionSensor.json" + "?" + (new
Date()).getTime();
151                         val = [];
152                         var index = 0;
153                         for (var i = data.length - 1; i>=0; i--) {
154                             val[index] = data[i].value;
```

```
155                             index++;
156                         }
157                         $scope.tempData = [val];
158                     });
159                 };
160                 $scope.getTemp();
161                 // Update data in real-time
162                 $interval(function() {
163                     var x = $scope.tempData[0].shift();          // Remove
the FIRST element in array
164                     $scope.tempData[0].push(x);                  // Push
the element to the END of array
165                     $scope.nowTemp = x;
166                 }, 1000);                    // Update data every 1 second
167                 // Update JSON file
168                 $interval(function() {
169                     $scope.getTemp();
170                 },5000);                     // Update JSON every 5
seconds
171                 $scope.tempLabels = ["6' ago","5' ago","4' ago","3'
ago","2' ago","1' ago","Now"];
172                 $scope.tempColors = [
173                     "#43db25"       // greeen
174                 ];
175
176                 // CAMERA
177                 var imageURL = "CameraTemp.jpg";
178                 $scope.cameraURL = imageURL + "?" + (new
Date()).getTime();
179                 // console.log($scope.cameraURL);
180                 $scope.getImage = function() {
181                     $http.get($scope.cameraURL, {
182                         cache: false
183                     }).success(function(data, status, headers, config) {
184                         $scope.cameraURL = "CameraTemp.jpg" + "?" + (new
Date()).getTime();
185                     });
186                 };
187                 $scope.intervalFunction = function() {
188                     $timeout(function() {
189                         $scope.getImage();
190                         $scope.intervalFunction();
191                         // console.log('Enter Camera');
192                     }, 5000)
193                 };
194                 $scope.intervalFunction();
```

```
                }
            });
        };
        $scope.getTest();
        $interval(function(){
            $scope.getTest();
        }, 10000);
}]);

ngrokID = 'Need to be changed';


// Function to access ngrok in controller and use in plain javascript
setInterval(function ngrokChange() {            // Keep checking for WiFi condition every 1s
    var scope = angular.element($("#ngrokIDTesting")).scope();
    console.log(scope.ngrokID);
},1000);

// Function to send critical warning email to user's email address whenever high temperature is detected
setInterval(function alertTemp() {              // Keep checking for temperature every 1s
    var temp = angular.element($("#currentTemp")).scope();
    if (temp.nowTemp > 20) {
        // alert(temp.nowTemp);
        // sendEmail();
    }
},1000);
```

main.css: (CSS file for styling the index.html)

```
body,h1,h2,h3,p {font-family: "Raleway", sans-serif}
body, html {
     height: 100%;
     line-height: 1.6;
}
h1.MQP {text-align: center}
.w3-opacity {text-align: center}
h3 {
     text-align: center;
}
.quarter {
     width: 25%;
     float: left;
     padding: 0 8px;
}
.third {
     width: 33%;
     float: left;
     padding: 0 8px;
}
.half {
     width: 50%;
     float: left;
     padding: 0 8px;
}
.bgimg-1 {
     background-position: center;
     background-size: cover;
     /*background-image: url("http://www.covertec.it/wp-
content/uploads/2015/10/casa-intelligente.jpg");*/
     background-image: url("home-main.jpg");
     min-height: 80%;
}
.w3-navbar li a {
     padding: 16px;
     float: left;
}
.about-col {
     width: 30%;
     height: auto;
     margin-left: 35%;
     margin-top: 10%;
     margin-bottom: 10%;
}
.btn {
```

```
45      text-align: center;
46      font-size: 20px;
47      cursor: pointer;
48      border-radius: 8px;
49      background-color: white;
50      display: inline-block;
51      border: 2px solid #0c0c0c;
52  }
53  .btn:hover {
54      background-color: #0c0c0c;
55      color: white;
56  }
57  footer {
58      padding-top: 16px;
59      padding-bottom: 16px;
60  }
```

live_data.css: (CSS file for styling the live_data.html)

```
body,h1,h2,h3,p {
    font-family: "Raleway", sans-serif;
}
body, html {
    height: 100%;
    /*line-height: 1.6;*/
}
h1 {
    text-align: center;
}
.topContainer {
    text-align: center;
}
.c1-box {
    float: left;
    width: 70%;
}
.c2-box {
    float: left;
    width: 30%;
    text-align: center;
/*  padding-top: 3%;*/
}
.headerDash {
    background-position: center;
    background-size: cover;
    background-image:
url("http://www.eclipse.org/smarthome/img/pipes.png");
    /*min-height: 80%;*/
    margin-top: 3%;
}
.w3-sidenav {
    /*overflow: hidden;*/
}
.w3-main {
    border-left: solid;
    border-left-color: #999a9b;
    border-left-width: 1px;
}
.lightButton {
    font-size: 24px;
    text-align: center;
    display: inline-block;
    -webkit-transition-duration:0.4s;
    transition-duration: 0.4s;
```

```
45      cursor: pointer;
46      border-radius: 8px;
47  }
48  .blueLi {
49      background-color: white;
50      border: 2px solid #008CBA;
51  }
52  .blueLi:hover {
53      background-color: #008CBA;
54      color: white;
55  }
56  .blueLiOnClick {
57      background-color: #008CBA;
58      color: white;
59  }
60  .redLi {
61      background-color: white;
62      border: 2px solid #f44336;
63  }
64  .redLi:hover {
65      background-color: #f44336;
66      color: white;
67  }
68  .redLiOnClick {
69      background-color: #f44336;
70      color: white;
71  }
72  .greenLi {
73      background-color: white;
74      border: 2px solid #4CAF50;
75  }
76  .greenLi:hover {
77      background-color: #4CAF50;
78      color: white;
79  }
80  .greenLiOnClick {
81      background-color: #4CAF50;
82      color: white;
83  }
84  .brightLi {
85      background-color: white;
86      border: 2px solid black;
87  }
88  .brightLi:hover {
89      background-color: #ffff60;
90      color: black;
```

```
}
.brightLiOnClick {
    background-color: #ffff60;
    color: black;
}
.subInfo {
    text-align: center;
    font-style: italic;
}
.camImg {
    text-align: center;
    cursor: pointer;
}
.tempForm {
    width: 90%;
    border: 2px solid black;
    border-radius: 4px;
}
.submitButton {
    cursor: pointer;
    font-size: 20px;
    text-align: center;
    display: inline-block;
    border-radius: 6px;
    background-color: white;
    border: 2px solid black;
    width: auto;
}
.submitButton:hover {
    background-color: black;
    color: white;
}
```

Appendix 2: Arduino Code

Coordinator Code:

```
/*************************************************************************
* FILENAME :        coordinator_with_failure_mode.ino
*
* DESCRIPTION :
*       Complete code for the coordinator Arduino in Smart Home IOT System
* NOTES :
*       This coordinator code using sample code from opensource ArduCam library for ArduCam camera module
*                       The link to the source is provided here: https://github.com/ArduCAM/Arduino/tree/master/ArduCAM/examples/mini
*
* AUTHOR :    THINH LY, ANH TRAN, DUC TRAN
*
**/
#include <SimpleTimer.h>
#include <ArduinoJson.h>
#include <Wire.h>
#include <ArduCAM.h>
#include <SPI.h>
#include "memorysaver.h"

#define DEBUGGING  // Enabe debug tracing to Serial port.
#define MAX_FRAME_LENGTH 64   // maximum framelength of 64 bytes.
#define CALLBACK_FUNCTIONS 1  // Define how many callback functions you have. Default is 1.
const int CS = 53; // chip select of camera
ArduCAM myCAM(OV2640, CS);
static const size_t bufferSize = 1781; // maximum buffer size for sending images.
static uint8_t buffer[bufferSize] = {0xFF};
#define DEBUG true
```

```
byte light1_on[23] = {0x7E, 0x00, 0x13, 0x10, 0x01, 0x00, 0x13, 0xA2, 0x00, 0x41, 0x52, 0xEC, 0x6B, 0xFF, 0xFE, 0x00, 0x00, 0x31, 0x30, 0x30, 0x31, 0x30, 0x60}; // Command message: turn light 1 on

byte light1_off[23] = {0x7E, 0x00, 0x13, 0x10, 0x01, 0x00, 0x13, 0xA2, 0x00, 0x41, 0x52, 0xEC, 0x6B, 0xFF, 0xFE, 0x00, 0x00, 0x31, 0x30, 0x30, 0x30, 0x30, 0x61}; // Command message: turn light 1 off

byte request_light[23] = {0x7E, 0x00, 0x13, 0x10, 0x01, 0x00, 0x13, 0xA2, 0x00, 0x41, 0x52, 0xEC, 0x6B, 0xFF, 0xFE, 0x00, 0x00, 0x31, 0x30, 0x30, 0x30, 0x31, 0x60}; // Request message: request information of the light module

byte light2_on[23] = {0x7E, 0x00, 0x13, 0x10, 0x01, 0x00, 0x13, 0xA2, 0x00, 0x41, 0x52, 0xEC, 0xD7, 0xFF, 0xFE, 0x00, 0x00, 0x32, 0x30, 0x30, 0x31, 0x30, 0xF3}; // Command message: turn light 2 on

byte light2_off[23] = {0x7E, 0x00, 0x13, 0x10, 0x01, 0x00, 0x13, 0xA2, 0x00, 0x41, 0x52, 0xEC, 0xD7, 0xFF, 0xFE, 0x00, 0x00, 0x32, 0x30, 0x30, 0x30, 0x30, 0xF4}; // Command message: turn light 2 off

byte request_tempInfo[23] = {0x7E, 0x00, 0x13, 0x10, 0x01, 0x00, 0x13, 0xA2, 0x00, 0x41, 0x52, 0xEC, 0xD7, 0xFF, 0xFE, 0x00, 0x00, 0x32, 0x30, 0x30, 0x30, 0x31, 0xF3}; // Request message: request information of the heat module

byte command_temp[23] = {0x7E, 0x00, 0x13, 0x10, 0x01, 0x00, 0x13, 0xA2, 0x00, 0x41, 0x52, 0xEC, 0xD7, 0xFF, 0xFE, 0x00, 0x00, 0x32, 0x30, 0x30, 0x30, 0x30, 0xF4}; // Command message: turn heat on to the desired temperature

byte wifiStatusFailed1[23] = {0x7E, 0x00, 0x13, 0x10, 0x01, 0x00, 0x13, 0xA2, 0x00, 0x41, 0x52, 0xEC, 0x6B, 0xFF, 0xFE, 0x00, 0x00, 0x31, 0x31, 0x30, 0x30, 0x30, 0x60};

byte wifiStatusFailed2[23] = {0x7E, 0x00, 0x13, 0x10, 0x01, 0x00, 0x13, 0xA2, 0x00, 0x41, 0x52, 0xEC, 0xD7, 0xFF, 0xFE, 0x00, 0x00, 0x31, 0x31, 0x31, 0x31, 0x31, 0xF1};

// WiFi SSID and Passwords

String ssid = "MySpectrumWiFia4-2G";

String password = "pinksquirrel283";

// Web Server URL

String server = "smarthomewpi.000webhostapp.com";

// PHP files for images and json data

String uriCamera = "/esp8266.php";

String uri = "esp8266b.php";

int inp;

int mess_length = 23;
```

```
int cameraPosting = 0;
int wifiStatus = 0;
int changeCoordinator = 0;
int jsonPosting = 0;

// timer for sending request message and receive feedback from router
SimpleTimer timer;

// Arrays to get router's address and data.
int sender[8];
int data[5];

int fail1 = 0;
int fail2 = 0;
volatile int pinNumber = 0;
int period = 20;
int counter = 0;
int senderID = 0;
int a[2];
int prevStatus1 = 0;
int prevStatus2 = 0;
int router1 = 0;
int router2 = 0;
int data1 = 0;
int data2 = 0;
int post = 0;
int start_index = 0;
int postCamera = 1;
unsigned long distance = 100;
int duration;
```

```
int request_temp = 0;
int temp;
unsigned long counter_camera = 0;
int camera_status = 0;
unsigned long start_time_camera = 0;
int readData_flag = 0;
int prev_camera_flag = 0;
const int trig = 8;     // Set Trig pin for HC-SR04
const int echo = 7;     // Set Echo pin for HC-SR04
int timerID = 0;
int startIndex = 0;

// Structure for storing sent data from router
struct SensorData {
  String name;
  int sendFrom;
  int time;
  int value;
};
int t = 0;
#define SENSORDATA_JSON_SIZE (JSON_OBJECT_SIZE(3))
String sendData(String command, const int timeout, boolean debug);
void postData();
/*
 *  Setup function for ZigBee, Camera and WiFi module
 */
void setup()
{
  uint8_t vid, pid, temp;
  // starts serial communication
```

```
pinMode(trig, OUTPUT); // Set trig pin as output to transmit the ultrasonic wave
pinMode(echo, INPUT); // Set echo pin as input to receive the reflected wave
Wire.begin();
Serial2.begin(115200);
Serial1.begin(115200);
Serial.begin(115200);
// Bypass the ZigBee module
Serial1.write("\n\r");
Serial1.write("B");
Serial1.println();
Serial1.println();
delay(1000);
if (Serial1.find("Bypass")) {
  Serial1.write("B");
}
// Boot up and configure the WiFi module
reset();
connectWifi();
sendData("AT+CWMODE=3\r\n", 1000, DEBUG); // configure as access point
sendData("AT+CIPMUX=1\r\n", 1000, DEBUG); // configure for multiple connection
sendData("AT+CIPSERVER=1,80\r\n", 1000, DEBUG); // turn on server on port 80
sendData("AT+CIFSR\r\n", 1000, DEBUG);
// set timer for calling the function postData each 5 seconds
timerID = timer.setInterval(5000, postData);
a[0] = 0;
a[1] = 0;



// Setup for Camera, resolution 640x480, compresssed file type JPEG
pinMode(CS, OUTPUT); // set the CS as an output:
```

```
SPI.begin(); // initialize SPI

//Check if the ArduCAM SPI bus is OK
myCAM.write_reg(ARDUCHIP_TEST1, 0x55);
temp = myCAM.read_reg(ARDUCHIP_TEST1);
if (temp != 0x55) {
  Serial.println("SPI1 interface Error!");
  while (1);
}

// Check if the camera module type is OV2640
myCAM.wrSensorReg8_8(0xff, 0x01);
myCAM.rdSensorReg8_8(OV2640_CHIPID_HIGH, &vid);
myCAM.rdSensorReg8_8(OV2640_CHIPID_LOW, &pid);
Serial.println(vid, HEX);
Serial.println(pid, HEX);
if ((vid != 0x26) || (pid != 0x42)) {
  Serial.println("Can't find OV2640 module!");
  while (1);
}
else {
  Serial.println("OV2640 detected.");
}

// Change to JPEG capture mode and initialize the OV2640 module
myCAM.set_format(JPEG);  myCAM.InitCAM();
myCAM.OV2640_set_JPEG_size(OV2640_640x480);
Serial.print("640x480:");
Serial.println(OV2640_640x480);
myCAM.clear_fifo_flag(); myCAM.write_reg(ARDUCHIP_FRAMES, 0x00);
```

```
}
/*
   Check the wifi status and set the global variable wifiStatus
   wifiStatus = 1 mean the ESP8266 module is not working
*/
void wifi_Status(){
  Serial2.println("AT");
  delay(200);
  if (Serial2.find("OK")){
    wifiStatus = 0;
  }
  else {
    wifiStatus = 1;
    return;
  }
  Serial2.println("AT+CWJAP?");
  delay(200);
  if (Serial2.find("No AP")){
    wifiStatus = 1;
    return;
  }
  else {
    wifiStatus = 0;
  }
}
/*
 *  Reset the ESP8266 module
 */
void reset() {
  Serial2.println("AT+RST");
```

```
  delay(1000);
  if (Serial2.find("OK") ) {
    Serial.write(Serial2.read());
    Serial.println("Module Reset");
  }
}
/*
 * connect the ESP8266 module to the local WiFi
 */
void connectWifi() {
  String cmd = "AT+CWJAP=\"" + ssid + "\",\"" + password + "\"";
  Serial2.println(cmd);
  delay(2000);
  if (Serial2.find("OK")) {
    Serial.println("Connected!");
  }
  else {
    connectWifi();
    Serial.println("Cannot connect to wifi");
  }
}
int intArraySum(int A[8]) {
  int sum = 0;
  for (int j = 0; j < 8; j++) {
    sum = sum + A[j];
  }
  return sum;
}
void cleanBuffer() {
  for (int i = 0; i < bufferSize; i++) {
```

```
    buffer[i] = 0xFF;
  }
}
/*
 * CAMERA
 */
/*
 * Read  Data From Camera To Buffer
 */
int readToBuffer(int len) {
  cleanBuffer();
  Serial.println("Reading");
  byte temp_last, temp;
  int i = 0;
  if (len >= 393216 || len > 1785) {
    Serial.println("Over size.");
    return 0;
  }
  else if (len == 0) {
    Serial.println("Size is 0.");
    return 0;
  }
  while (len)
  {
    temp_last = temp;
    temp =  SPI.transfer(0x00);//read a byte from spi
    //Serial.write(temp);
    buffer[i] = temp;
    len -= 1;
    i += 1;
```

```
    if ( (temp == 0xD9) && (temp_last == 0xFF) ) { //If find the end ,break while,
      return i;
    }
  }
  return i;
}

/*
 * Post Camera Data To Web Server
 */
void httpPostCamera(int len, bool lastPacket) {
  cameraPosting = 1;
  while (Serial2.available()) Serial2.read();
  unsigned long start_time = millis();
  Serial2.println("AT+CIPSTART=0,\"TCP\",\"" + server + "\",80");//start a TCP connection.
  Serial.println("AT+CIPSTART=0,\"TCP\",\"" + server + "\",80");
  if ( Serial2.find("OK")) { // if connection is successful
    Serial.println("TCP connection ready");
    while (Serial2.available()) Serial2.read();
  }
  else {
    postCamera = 0;
    Serial2.println("AT+CIPCLOSE=0");
    if (Serial2.find("OK")) {
      Serial.println("Closed");
    }
    return;
  }
  String start_request = ""; String end_request = "";
  if (lastPacket == true) {
```

```
  start_request = start_request +
                "\n--AaB03x\n" +
                "Content-Disposition: form-data; name=\"userfile\"; filename=\"CAM1.TXT\"\n" +
                "Content-Transfer-Encoding: binary\n\n";
 }
 else {
  start_request = start_request +
                "\n--AaB03x\n" +
                "Content-Disposition: form-data; name=\"userfile\"; filename=\"CAM.TXT\"\n" +
                "Content-Transfer-Encoding: binary\n\n";
 }
 end_request = end_request + "\n--AaB03x--\n";  // in file upload POST method need to specify arbitrary boundary code


 uint16_t full_length;
 full_length = start_request.length() + len + end_request.length();
 byte temp, temp_last;
 Serial.println("Message Length: ");
 Serial.println(len);
 String sendCmd = "AT+CIPSEND=0,";//determine the number of bytes to be sent.
 String postRequest0 = "POST " + uriCamera + " HTTP/1.1";
 String postRequest1 = "Host: " + server;
 String postRequest2 = "Content-Type: multipart/form-data; boundary=AaB03x";
 String postRequest3 = "Content-Length: ";

 // Length in bytes of the message to be sent
 int postLength = postRequest0.length() + 2 + postRequest1.length() + 4 + postRequest2.length() + 2 + postRequest3.length() + String(full_length).length() + 2 + len + start_request.length() + end_request.length();
 int i = 0;
```

```
Serial2.print(sendCmd);
Serial.print(sendCmd);
Serial2.println(postLength);
Serial.println(postLength);
unsigned long end_time1 = millis();
// Send image file byte by byte
if (Serial2.find(">")) {
  Serial2.println(postRequest0);
  Serial2.println(postRequest1);
  Serial2.println(postRequest2);
  Serial2.print(postRequest3); Serial2.println(full_length);
  Serial2.print(start_request);
  unsigned long end_time2 = millis();
  while (len)
  {
    temp =  buffer[i];//read a byte from spi
    Serial2.write(temp);
    //Serial.write(temp);
    len -= 1;
    i += 1;
  }
  Serial2.println(end_request);
  Serial.println(end_request);
  unsigned long end_time3 = millis();
  // if sending successful
  if ( Serial2.find("SEND OK")) {
    Serial.println("Packet sent");
    while (Serial2.available()) Serial2.read();
  }
  unsigned long end_time4 = millis();
```

```
    // close the connection
    delay(150);
    Serial2.println("AT+CIPCLOSE=0");
    if (Serial2.find("OK")) {

      Serial.println("Closed");
      postCamera = 1;
      cameraPosting = 0;
    }
    // Measure the total time for sending each message
    Serial.print("Start Time: "); Serial.println(end_time1 - start_time);
    Serial.print("Find > Time: "); Serial.println(end_time2 - end_time1);
    Serial.print("Write Message Time: "); Serial.println(end_time3 - end_time2);
    Serial.print("Response Time: "); Serial.println(end_time4 - end_time3);
  }
  else
  {
    postCamera = 0;
    Serial2.println("AT+CIPCLOSE=0");
    if (Serial2.find("OK")) {
      Serial.println("Closed");
      cameraPosting = 0;
    }
    return;
  }

}

// Capture Image, Read Image Pixels To Buffer And Post Image To Server
void Camera(ArduCAM myCAM) {     //reads out pixels from the Arducam mini module
```

```
myCAM.clear_fifo_flag();
myCAM.flush_fifo();
myCAM.start_capture();
while (!myCAM.get_bit(ARDUCHIP_TRIG, CAP_DONE_MASK));
Serial.print("Picture captured. ");

size_t len = myCAM.read_fifo_length();
Serial.println("Capture Length");
Serial.println(len);

if (len >= 393216) {
  Serial.println("Over size.");
  return;
}
else if (len == 0) {
  Serial.println("Size is 0.");
  return;
}
Serial.print("Length in bytes: "); Serial.println(len); Serial.println();
myCAM.CS_LOW(); myCAM.set_fifo_burst(); SPI.transfer(0xFF);
Serial.print("Length: ");
Serial.println(len);
int numOfRound = (len - 1) / bufferSize;
Serial.print("Number Of Round: ");
Serial.println(numOfRound);
int remainder = len - (numOfRound * bufferSize) - 1;
int blockLength = 0;
Serial.println("Remainder: ");
Serial.println(remainder);
int i;
```

```
for (i = 0; i < numOfRound; i++) {
  unsigned long start_time = millis();
  unsigned long end_time1;
  if (postCamera == 1) {
    start_time = millis();
    blockLength = readToBuffer(bufferSize);
    /*Serial.println("Block Length: ");
      Serial.println(blockLength);*/
    end_time1 = millis();
  }
  if (blockLength < bufferSize) {
    httpPostCamera(blockLength, true);
    break;
  }
  else httpPostCamera(blockLength, false);
  unsigned long end_time2 = millis();
  Serial.print("Read Time: "); Serial.println(end_time1 - start_time);
  Serial.print("Post Time: "); Serial.println(end_time2 - end_time1);
  Serial.print("Each Round Time: "); Serial.println(end_time2 - start_time);
}
if (remainder > 0) {
  if (blockLength == bufferSize) {
    Serial.println("Remainder");
    remainder = readToBuffer(remainder);
    httpPostCamera(remainder, true);
  }
}
myCAM.CS_HIGH();
//Clear the capture done flag
prev_camera_flag = 1;
```

```
  Serial.print("prev_camera_flag = ");
  Serial.println(prev_camera_flag);
}

/*
 * Send command/request message to router
 */
void sendCommand(byte message[], int routerNumber, int mode) { //mode = 0 command  mode = 1 request
  int flag = 0;
  int total_start = millis();
  while (Serial1.available()) Serial1.read();
  Serial.println("Send Command");
  Serial1.write(message, mess_length);
  Serial1.println();
  int i = 0;
  unsigned long ack_start = millis();
  // Wait for ACK message
  while (Serial1.available() < 10);
  unsigned long ack_end = millis();
  Serial.print("ACK :");
  Serial.println(ack_end - ack_start);
  // Get the ACK message
  if (Serial1.available() > 10) {
    byte dataByte0 = Serial1.read();
    Serial.println("Receive Command");
    if (dataByte0 == 0x7E) {
      byte dataByte1 = Serial1.read();
      byte dataByte2 = Serial1.read();
      if (dataByte2 == 0x11) {
        for (int i = 0; i < 18; i++) Serial1.read();
```

```
  dataByte0 = Serial1.read();
  dataByte1 = Serial1.read();
  dataByte2 = Serial1.read();
 }
if (dataByte2 == 0x07) {      // decode the ACK message
  for (int a = 0; a < 8; a++) {
    int txstatus = Serial1.read();
    if ((a == 5) && (txstatus != 0)) {
      return;
    }
    else if (a == 7) {
      if (mode == 0) {
        post = 0;
        return;
      }
    }
  }
  // if
  unsigned long start_time = millis();
  while (millis() - start_time < 200) { // wait for the response message if request command
    if (Serial1.available() > 20) {
      dataByte0 = Serial1.read();
      dataByte1 = Serial1.read();
      dataByte2 = Serial1.read();
      flag = 1;
      break;
    }
  }
  unsigned long end_time1 = millis();
  if (flag == 0) {
```

```
  while (Serial1.available()) Serial1.read();
  post = 0;
  return;
 }
 if ((dataByte1 == 0x00) && (dataByte2 == 0x11) && (mode == 1)) { // decode the response message
  Serial.println("Get Data");
  for (int a = 0; a < 18; a++) {
   int mess = Serial1.read();
   Serial.print(mess, HEX);
   if ((a > 0) && (a < 9))   sender[a - 1] = mess;
   else if ((a > 11) && (a < 17)) data[a - 12] = mess;
  }
  // Identify the original router of the response message
  if (intArraySum(sender) == 671) {
   if (counter == 1) {
    Serial.println();
    Serial.println("Fault Packet");
    post = 0;
   }
   else {
    senderID = 0;
    router1 = senderID;
    data1 = data[4];
    Serial.print("Router 1");
    post = 1;
   }
  }
  if (intArraySum(sender) == 779) {
   if (counter == 0) {
    Serial.println();
```

```
          post = 0;
          Serial.println("Fault Packet");
        }
        else {
          senderID = 1;
          router2 = senderID;
          data2 = data[4];
          post = 1;
          Serial.print("Router 2");
        }
      }
      Serial.println();
      unsigned long end_time2 = millis();
      Serial.print("Time for receiving and decoding message:");
      Serial.println(end_time2 - start_time);
    }
    // if message sending/receiving failed.
    else {
      post = 0;
      for (int i = 0; i < 18; i++) Serial1.read();
      return;
    }
  }
  else {
    post = 0;
    for (int i = 0; i < 18; i++) Serial1.read();
    return;
  }
}
else {
```

```
      post = 0;
      while (Serial1.available()) Serial1.read();
      return;
    }
  }
  else {
    post = 0;
    while (Serial1.available()) Serial1.read();
    return;
  }
  int total_end = millis();
  Serial.println("Total Time Spent");
  Serial.println(total_end - total_start);
}
/*
 * Convert an SensorData object to JSON data type
 */
String serialize(SensorData jdata)
{
  String A = /*String("\"") + */String("{") + String("\"") + String("name") + String("\"") + String(":") + String("\"") + String(jdata.name) + String("\"") + String(",");
  String B = String("\"") + String("sender") + String("\"") + String(":") +  String(jdata.sendFrom) + String(",");
  String C = String("\"") + String("time") + String("\"") + String(":") +  String(jdata.time) + String(",");
  String D = String("\"") + String("value") + String("\"") + String(":") +  String(jdata.value) + String("}")  /*String("\"")*/;
  return A + B + C + D;
}
/*
 * Send HTTP message containing JSON data to the web server
 */
```

```
void httpPost(String json) {
  jsonPosting = 1;
  while (Serial2.available()) Serial2.read();
  Serial2.println("AT+CIPSTART=0,\"TCP\",\"" + server + "\",80");//start a TCP connection.
  if (Serial2.find("OK")) { // if connection is successful
    Serial.println("TCP connection ready");
    while (Serial2.available()) Serial2.read();
  }
  else { // if the connection is not successful
    // close connection
    Serial2.println("AT+CIPCLOSE=0");
    if (Serial2.find("OK")) Serial.println("CLOSED 1");
    while (Serial2.available()) Serial2.read();
    jsonPosting = 0;
    return;
  }

  String postRequest0 =
    "GET /" + uri + "?" + json + " HTTP/1.1"; // first line of the HTTP Get message
  String postRequest1 =
    "Host: " + server; // second line of the HTTP Get message

  String sendCmd = "AT+CIPSEND=0,";//determine the number of bytes to be sent.;
  Serial2.print(sendCmd);
  // determine the length of the JSON message to be sent
  Serial.print("Length: "); Serial.println(postRequest0.length() + postRequest1.length() + 8);
  if (senderID == 0) Serial2.println(postRequest0.length() + postRequest1.length() + 8);
  else if (senderID == 1) Serial2.println(postRequest0.length() + postRequest1.length() + 8);
  // Send Json data
  if (Serial2.find(">")) {
```

```
    Serial2.println(postRequest0);
    Serial2.println(postRequest1);
    Serial2.println();
    Serial2.println();
    if ( Serial2.find("SEND OK")) { // if sending is successful
      Serial.println("Packet sent");
      Serial2.println("AT+CIPCLOSE=0");
      if (Serial2.find("OK")) Serial.println("CLOSED");
      while (Serial2.available()) Serial2.read();
      jsonPosting = 0;
    }
    else { // if sending fail
      Serial2.println("AT+CIPCLOSE=0");
      if (Serial2.find("OK")) Serial.println("CLOSED 2");
      while (Serial2.available()) Serial2.read();
      jsonPosting = 0;
      return;
    }
  }
  else { // if sending fail
    Serial2.println("AT+CIPCLOSE=0");
    if (Serial2.find("OK")) Serial.println("CLOSED 1");
    while (Serial2.available()) Serial2.read();
    jsonPosting = 0;
    return;
  }
}
/*
 * Receive Command From The Web Server
 */
```

```
void httpCommand() {
  if (Serial2.available() && jsonPosting == 0 && cameraPosting == 0)
// check if the esp is sending a message
  {
    // Specific string to identify the commands from server
    if (Serial2.find("+IPD,"))
    {
      Serial.println("Get Pin Number");
      delay(50);
      int connectionId = Serial2.read() - 48;
// Get the ID of the TCP connection, subtract 48 because the read() function returns
      // the ASCII decimal value and 0 (the first decimal number) starts at 48
      int test1, test2 = 0;
      if (Serial2.find("pin=")) { // // advance cursor to "pin=", this information is the router that is commanded to turn LED on/off
        test1 = Serial2.read();
        test2 = Serial2.read();
        if (test2 == 32) { //led command
          pinNumber = test1 - 48;  // get the number after "pin="
          Serial.println(pinNumber);
          String closeCommand = "AT+CIPCLOSE=";
          closeCommand += connectionId; // append connection id
          closeCommand += "\r\n";
          sendData(closeCommand, 100, false); // close connection
        }
        else { // advance cursor to "temp=", this information is the request temperature for router 2 to turn the
            // fan module on
          pinNumber = 3;
          temp = (test1 - 48) * 10 + (test2 - 48);
          String closeCommand = "AT+CIPCLOSE=";
```

```
        closeCommand += connectionId; // append connection id
        closeCommand += "\r\n";
        sendData(closeCommand, 100, false); // close connection
       }
      }
     // if there is no command can be find in the serial buffer, or if the command arrive when the critical
     // session of sending image to web server
     else {
       String closeCommand = "AT+CIPCLOSE";
       closeCommand += connectionId; // append connection id
       closeCommand += "\r\n";
       sendData(closeCommand, 100, false); // close connection
     }
    }
  }
}
/*
 * Interrupt On Serial Event 2
 */
void serialEvent2() {
  // Get the commands sent from the web server
  httpCommand();
  // Decode the commands, forward to the corresponding router to execute
  switch (pinNumber) {
    // Turn LED at router 2 on/off
    case 2:
     {
       if (prevStatus2 == 0) {
         if (camera_status == 0)
           sendCommand(light2_on, 2, 0);
```

```
        prevStatus2 = 1;
    }
    else {
      if (camera_status == 0)
        sendCommand(light2_off, 2, 0);
        prevStatus2 = 0;
    }
    pinNumber = 0;
    break;
  }
// Turn LED at router 1 on/off
case 1:
  {
    if (prevStatus1 == 0) {
      if (camera_status == 0)
        sendCommand(light1_on, 1, 0);
        prevStatus1 = 1;
    }
    else {
      if (camera_status == 0)
        sendCommand(light1_off, 1, 0);
        prevStatus1 = 0;
    }
    pinNumber = 0;
    break;
  }
// Turn the fan module at router 2 on/off
case 3:
  {
    if (camera_status == 0) {
```

```
        request_temp = temp;
        command_temp[20] = request_temp;
        command_temp[22] = 0xF4 + 0x30 - request_temp;
        Serial.println(request_temp);
        Serial.println();
        for (int i = 0; i < 23; i++) Serial.print(command_temp[i], HEX);
        Serial.println();
        sendCommand(command_temp, 2, 0);
       }
       pinNumber = 0;
       break;
     }
   default:
     break;
  }
}
/*
 * Send request/command message to routers
 */
String sendData(String command, const int timeout, boolean debug)
{
  String response = "";
  Serial2.print(command); // send the read character to the esp8266
  long int time = millis();
  while ((time + timeout) > millis())
  {
   while (Serial2.available())
   {
     // The esp has data so display its output to the serial window
     char c = Serial2.read(); // read the next character.
```

```
      response += c;
    }
  }
  if (debug)
  {
    Serial.print(response);
  }
  return response;
}
/*
 * read from the ultrasonic motion sensor
 */
void readSonic() {
  digitalWrite(trig, LOW);    // trig off
  delayMicroseconds(5);
  digitalWrite(trig, HIGH);   // trig on
  delayMicroseconds(10);
  digitalWrite(trig, LOW);    // trig off
  duration = pulseIn(echo, HIGH);
  distance = int((duration / 2) * 0.034); // calculate distance
}
/*
 * Send request message to router, serialize received data into JSON format and post to web server periodically (using timer)
 */
void postData() {
  if (counter == 0) { // router 1
    // send request message
    sendCommand(request_light, 1, 1);
    // Serialize JSON data and send to the web server
    if (post == 1) {
```

```
      SensorData jdata = {"Occupancy", router1, t, data1 - 48};
      String json;
      json = serialize(jdata);
      String http = String("jsonString1=") + String(json);
      httpPost(http);
    }
    counter = 1;
  }
  else { // router 2
    // send request message
    sendCommand(request_tempInfo, 2, 1);
    // Serialize JSON data and send to the web server
    if (post == 1) {
      SensorData jdata = {"Occupancy2", router2, t, data2};
      String json;
      json = serialize(jdata);
      String http = String("jsonString1=") + String(json);
      httpPost(http);
      t++;
    }
    counter = 0;
  }
}
/*
 * Main Code
 */
void loop()
{
  wifi_Status();
  readSonic(); // read distance from sensors
```

```
  if ((distance < 5) && (readData_flag == 0)) { // check if the objects is in range
    start_time_camera = millis();
    Serial.println("Motion Detected");
    camera_status = 1;
    readData_flag = 1;
  }
  // start capturing and sending image.
  if ((counter_camera < 1) && (millis() - start_time_camera > 2000) && (camera_status == 1)
&& (changeCoordinator == 0)) {
    Serial.print("Take Image "); Serial.println(counter_camera);
    counter_camera++;
    Camera(myCAM);
    start_time_camera = millis();
  }
  else if (camera_status == 0 && changeCoordinator == 0) {
    // otherwise, run timer to send JSON data each 5s interval
    timer.run();
  }
  // Sending message to routers when the ESP8266 module is disconnected
  if (wifiStatus == 1 && changeCoordinator == 0){
    sendCommand(wifiStatusFailed1, 1, 0);
    sendCommand(wifiStatusFailed2, 2, 0);
    changeCoordinator = 1;
  }
  if (counter_camera == 1) {    // stop capturing image
    counter_camera = 0;
    camera_status = 0;
    readData_flag = 0;
  }
  inp = 0;
  int i = 0;
```

```
  int j;
  if (fail1 == 3) {
    // Router 1 status
    Serial.println("Router 1 fail");
    fail1 = 0;
  }
  if (fail2 == 3) {
    // Router 2 status
    Serial.println("Router 2 fail");
    fail2 = 0;
  }
}
```

Router 1 Code:

```
/*************************************************************************
* FILENAME :        router_1_0219.ino
*
* DESCRIPTION :
*       Complete code for the router 1 Arduino in Smart Home IOT System
* NOTES :
*       The router 1 is designated to be the alternatives coordinator when the coordinator fail.
*       In normal working condition, router 1 is in charge on controlling the LED peripheral.
*
* AUTHOR :    THINH LY, ANH TRAN, DUC TRAN
*
**/

#include <SimpleTimer.h>
#include <ArduinoJson.h>
#include <Wire.h>
#include <SPI.h>
```

```
#define DEBUGGING // Enabe debug tracing to Serial port.
#define DEBUG true
// WiFi SSID and Passwords
String ssid = "MySpectrumWiFia4-2G";
String password = "pinksquirrel283";
// Web Server URL
String server = "smarthomewpi.000webhostapp.com";
// PHP files for json data
String uri = "esp8266c.php";
// timer for sending request message and receive feedback from router
int jsonPosting = 0;
SimpleTimer timer;
int timerID = 0;
int temperature = 0;
int senderID = 0;
int strlenght = 5;
int message[5];
int data[5];
int ack = 0;
int mode = 0; //0 for command, 1 for request
int LED = 0;
int inputPIR = 2;
int PIR_delay = 0;
int outputLED = 3;
int LED_status = 0;
int value;
int request_temp = 0;
int temp;
```

```
volatile int pinNumber = 0;
int prevStatus1 = 0;
int prevStatus2 = 0;
unsigned long duration;
int distance;
int mess3, mess1, mess2 = 0;
int mess4 = 0;
int flag_command = 0;
int counter_command = 0;
int mess0 = 0;
int post = 0;
int mess_length = 5;
byte light2_on[5] = {0x32, 0x30, 0x30, 0x31, 0x30}; // Command message: turn light 2 on
byte light2_off[5] = {0x32, 0x30, 0x30, 0x30, 0x30}; // Command message: turn light 2 off
byte request_tempInfo[5] = {0x32, 0x30, 0x30, 0x30, 0x31};
// Request message: request information of the heat module
byte command_temp[5] = {0x32, 0x30, 0x30, 0x30, 0x30};
// Command message: turn fan on to the desired temperature
byte wifi_fail[5] = {0x31, 0x31, 0x31, 0x31, 0x31};
int timer_id = 0;
void postData();
int wifi_on = 0;
int counter = 0;
int t = 0;
// Structure for storing sent data from router
struct SensorData {
  String name;
  int sendFrom;
  int time;
  int value;
```

```
};
/*
 *  Setup function for ZigBee and WiFi module
 */
void setup() {
  Wire.begin();
  Serial2.begin(115200);
  Serial1.begin(115200);
  pinMode(inputPIR, INPUT);       //Set up sensor input pin, LED output pin
  pinMode(outputLED, OUTPUT);
  Serial1.begin(115200);
  Serial.begin(115200);
  Serial1.write("\n\r");          //Bypass Xbee microcontroller
  Serial1.write("B");
  Serial1.println();
  delay(1000);
  if (Serial1.find("Bypass")) {
    Serial1.write("B");
  }
  // Configure the ZigBee module by entering AT mode
  Serial1.write("+++");
  delay(1000);
  while(Serial1.available()) Serial.write(Serial1.read());
  Serial1.write("ATDL4152ECD4\n\r");
  delay(1000);
  while(Serial1.available()) Serial.write(Serial1.read());
  Serial1.write("ATID123\n\r");
  delay(1000);
  while(Serial1.available()) Serial.write(Serial1.read());
  Serial1.write("ATWR\n\r");
```

```
  delay(1000);
  while(Serial1.available()) Serial.write(Serial1.read());
  Serial1.write("ATCN");
  // Boot up and configure the WiFi module
  delay(1000);
  reset();
  connectWifi();
  sendData("AT+CWMODE=3\r\n", 1000, DEBUG); // configure as access point
  sendData("AT+CIPMUX=1\r\n", 1000, DEBUG); // configure for multiple connection
  sendData("AT+CIPSERVER=1,80\r\n", 1000, DEBUG); // turn on server on port 80
  sendData("AT+CIFSR\r\n", 1000, DEBUG);
  // set timer for calling the function postData each 5 seconds
  timerID = timer.setInterval(5000, postData);
}
/*
 *  Reset the ESP8266 module
 */
void reset() {
  Serial2.println("AT+RST");
  delay(1000);
  if (Serial2.find("OK") ) {
    Serial.write(Serial2.read());
    Serial.println("Module Reset");
  }
}
/*
 *  Connect the ESP8266 module to local WiFi.
 */
void connectWifi() {
  String cmd = "AT+CWJAP=\"" + ssid + "\",\"" + password + "\"";
```

```
  Serial2.println(cmd);
  delay(2000);
  if (Serial2.find("OK")) {
    Serial.println("Connected!");
  }
  else {
    connectWifi();
    Serial.println("Cannot connect to wifi");
  }
}
/*
 * Send request/command message to router 2 (only when router 1 become coordinator)
 */
String sendData(String command, const int timeout, boolean debug)
{
  String response = "";
  Serial2.print(command); // send the read character to the esp8266
  long int time = millis();
  while ((time + timeout) > millis())
  {
    while (Serial2.available())
    {
      // The esp has data so display its output to the serial window
      char c = Serial2.read(); // read the next character.
      response += c;
    }
  }
  if (debug)
  {
    Serial.print(response);
```

```
  }
  return response;
}
/*
 * read from the PIR motion sensor
 */
void readPIR() {                //read data from PIR
  if (value == 0) {
    if (PIR_delay == 1) {
      value = digitalRead(inputPIR);
      PIR_delay = 0;
    }
    else PIR_delay++;
  }
  else {
    if (PIR_delay == 5000) {
      value = digitalRead(inputPIR);
      PIR_delay = 0;
    }
    else PIR_delay++;
  }
  if (value == 1) {
    LED_status = 1;
  }
  else {
    LED_status = 0;
  }
}
/*
 * Check if the incoming message is command/request message
```

```
 */
void checkMode() {
  if (mess4 == 48) mode = 0;      //command
  else if (mess4 == 49) mode = 1; //request
}
/*
 * Check of the incoming message is to turn the LED on/off
 */
void checkCommand() {
  if (mess1 == 48) {
    if (mess3 == 49) {             //mess3 = 1
      LED_status = HIGH;           //LED on
      flag_command = 1;
    }
    else if (mess3 == 48) {        //mess3 = 0
      LED_status = LOW;            //LED off
      flag_command = 0;
    }
    mess0 = 48;
    mode = 2;
  }
  // if receive critical messsage informing coordinator fail
  // configured to be the new coordinator by changing ZigBee configuration
  else {
      Serial1.write("+++");
      delay(1000);
      while(Serial1.available()) Serial.write(Serial1.read());
      Serial1.write("ATDL4152ECD7\n\r");
      delay(1000);
      while(Serial1.available()) Serial.write(Serial1.read());
```

```
    Serial1.write("ATID123\n\r");
    delay(1000);
    while(Serial1.available()) Serial.write(Serial1.read());
    Serial1.write("ATWR\n\r");
    delay(1000);
    while(Serial1.available()) Serial.write(Serial1.read());
    Serial1.write("ATCN");
    delay(2000);
    Serial1.write(wifi_fail, mess_length);
    delay(1000);
    wifi_on = 1;
  }
}
/*
 * Send request/command message to routers
 */
void sendCommand(byte message[], int mode) { //0 command 1 request
  int flag = 0;
  while (Serial1.available()) Serial1.read();
  Serial.println("Send Command");
  Serial1.write(message, mess_length);
}
}
/*
 * Convert an SensorData object to JSON data type
 */
String serialize(SensorData jdata)
{
  String A = String("{") + String("\"") + String("name") + String("\"") + String(":") + String("\"")
+ String(jdata.name) + String("\"") + String(",");
```

```
  String B = String("\"") + String("sender") + String("\"") + String(":") +  String(jdata.sendFrom) + String(",");
  String C = String("\"") + String("time") + String("\"") + String(":") +  String(jdata.time) + String(",");
  String D = String("\"") + String("value") + String("\"") + String(":") +  String(jdata.value) + String("}");
  return A + B + C + D;
}
/*
 * Send HTTP message containing JSON data to the web server
 */
void httpPost(String json) {
  jsonPosting = 1;
  while (Serial2.available()) Serial2.read();
  Serial2.println("AT+CIPSTART=0,\"TCP\",\"" + server + "\",80");//start a TCP connection.
  if (Serial2.find("OK")) { // if connection is successful
    Serial.println("TCP connection ready");
    while (Serial2.available()) Serial2.read();
  }
  else { // if the connection is not successful
    // close connection
    Serial2.println("AT+CIPCLOSE=0");
    if (Serial2.find("OK")) Serial.println("CLOSED 1");
    while (Serial2.available()) Serial2.read();
    jsonPosting = 0;
    return;
  }

  String postRequest0 =
    "GET /" + uri + "?" + json + " HTTP/1.1"; // first line of the HTTP Get message
  String postRequest1 =
```

```
  "Host: " + server; // second line of the HTTP Get message

 String sendCmd = "AT+CIPSEND=0,";//determine the number of bytes to be sent.;
 Serial2.print(sendCmd);
// determine the length of the JSON message to be sent
 Serial.print("Length: "); Serial.println(postRequest0.length() + postRequest1.length() + 8);
 if (senderID == 0) Serial2.println(postRequest0.length() + postRequest1.length() + 8);
 else if (senderID == 1) Serial2.println(postRequest0.length() + postRequest1.length() + 8);
  // Send Json data
 if (Serial2.find(">")) {
  Serial2.println(postRequest0);
  Serial2.println(postRequest1);
  Serial2.println();
  Serial2.println();
  if ( Serial2.find("SEND OK")) { // if sending is successful
    Serial.println("Packet sent");
    Serial2.println("AT+CIPCLOSE=0");
    if (Serial2.find("OK")) Serial.println("CLOSED");
    //while (Serial2.available()) Serial2.read();
    jsonPosting = 0;
  }
  else { // if sending fail
    Serial2.println("AT+CIPCLOSE=0");
    if (Serial2.find("OK")) Serial.println("CLOSED 2");
    while (Serial2.available()) Serial2.read();
    jsonPosting = 0;
    return;
  }
 }
 else { // if sending fail
```

```
    Serial2.println("AT+CIPCLOSE=0");
    if (Serial2.find("OK")) Serial.println("CLOSED 1");
    while (Serial2.available()) Serial2.read();
    jsonPosting = 0;
    return;
  }
}
/*
 * Send request message to router 2, serialize received data into JSON format and
 * post to web server periodically (using timer)
 * (only when router 1 become coordinator)
 */
void postData() {
  if (counter == 0) {
      SensorData jdata = {"Occupancy", 1, t, LED_status};
      String json;
      json = serialize(jdata);
      String http = String("jsonString1=") + String(json);
      senderID = 0;
      httpPost(http);
      counter = 1;
  }
  else {
      sendCommand(request_tempInfo, 1);
      SensorData jdata = {"Occupancy2", 2, t, temperature};
      String json;
      Serial.println(post);
    //  while (post != 1);
     //if (post == 1) {
      json = serialize(jdata);
```

```
    Serial.println(json);
    String http = String("jsonString1=") + String(json);
    senderID = 1;
    httpPost(http);
    t++;
   // post = 0;
  // }
    counter = 0;
 }
}
/*
 * Receive Command From The Web Server
 */
void httpCommand() {
 //Serial.println("Command");
 if (Serial2.available() && jsonPosting == 0) // check if the esp is sending a message
 {
  Serial.println("httpcommand");
  if (Serial2.find("+IPD,"))
  {
    Serial.println("Get Pin Number");
    delay(50);
    int connectionId = Serial2.read() - 48; // subtract 48 because the read() function returns
    // the ASCII decimal value and 0 (the first decimal number) starts at 48
    int test1, test2 = 0;
    if (Serial2.find("pin=")) { // advance cursor to "pin=", this information is the router that is commanded to turn LED on/off
      test1 = Serial2.read();
      test2 = Serial2.read();
      if (test2 == 32) { //led command
        pinNumber = test1 - 48;  // get the number after "pin="
```

```
        Serial.println(pinNumber);
        String closeCommand = "AT+CIPCLOSE=";
        closeCommand += connectionId; // append connection id
        closeCommand += "\r\n";
        sendData(closeCommand, 100, false); // close connection
       }
      else { // advance cursor to "temp=", this information is the request temperature for router 2 to turn the
           // fan module on
        pinNumber = 3;
        temp = (test1 - 48) * 10 + (test2 - 48);
        Serial.print("temperature =");
        Serial.println(temp, DEC);
        String closeCommand = "AT+CIPCLOSE=";
        closeCommand += connectionId; // append connection id
        closeCommand += "\r\n";
        sendData(closeCommand, 100, false); // close connection
       }
     }
    // if there is no command can be find in the serial buffer
     else {
       String closeCommand = "AT+CIPCLOSE";
       closeCommand += connectionId; // append connection id
       closeCommand += "\r\n";
       sendData(closeCommand, 100, false); // close connection
     }
   }
  }
}
/*
 * Interrupt On Serial Event 2
```

```
 */
void serialEvent2() {
  if (wifi_on == 1) { // only executed when router 1 become coordinator
  // Get the commands sent from the web server
  httpCommand();
  // Decode the commands, forward to the corresponding router to execute
  switch (pinNumber) {
    // Turn LED at router 1 on/off
    case 1:
    {
      if (prevStatus1 == 0) {
        flag_command = 1;
        // set LED_status to turn the LED on/off
        LED_status = 1;
        prevStatus1 = 1;
      }
      else {
        flag_command = 0;
        LED_status = 0;
        prevStatus1 = 0;
      }
      break;
    }
    // Turn LED at router 2 on/off
    case 2:
      {
        if (prevStatus2 == 0) {
          sendCommand(light2_on, 2);
          prevStatus2 = 1;
        }
```

```
        else {
          sendCommand(light2_off, 2);
          prevStatus2 = 0;
        }
        break;
      }
   // Turn the fan module at router 2 on/off
    case 3:
      {
         request_temp = temp;
         command_temp[3] = request_temp;
         for (int i = 0; i < 5; i++) Serial.print(command_temp[i], HEX);
         Serial.println();
         sendCommand(command_temp, 2);
        break;
      }
    default:
      break;
  }
       pinNumber = 0;
  }
}
/*
 * Main Code
 */
void loop() {
  if (flag_command == 0) {
    readPIR();
  }
  // Normal operation (when the coordinator is working)
```

```
if (wifi_on == 0) {
while (Serial1.available()) {
  Serial.println("Receive Data");
  delay(10);
  mess0 = Serial1.read();

  if (mess0 == 49 || mess0 == 50) {
    mess1 = Serial1.read();            //read message from RX buffer
    mess2 = Serial1.read();
    mess3 = Serial1.read();
    mess4 = Serial1.read();
    ack = 1;
    checkMode(); // Check whether message is request/command
    Serial.print(mess0,HEX);           //display received message on console for debugging
    Serial.print(mess1,HEX);
    Serial.print(mess2,HEX);
    Serial.print(mess3,HEX);
    Serial.print(mess4,HEX);
    Serial.println();
    break;
  }
  else { // if not receive the message, or the message is not in correct format
    // clear the ZigBee buffer
    Serial1.read();
    Serial1.read();
    Serial1.read();
    Serial1.read();
    mess0 = 48;
    break;
  }
```

```
  }
  if (mode == 0 && mess0 == 49) checkCommand();   // for Command message
  else if (mode == 1 && mess0 == 49) {            // for Request message
   if (ack == 1) {
    Serial.println("Sending");                 // Print "Sending" to computer console
    Serial1.write(49);                        // Router number = 1
    Serial1.write(48);
    Serial1.write(48);
    if ((LED_status == HIGH)) {               // Write LED status
     Serial1.write(49);
    } else Serial1.write(48);
    if ((value == HIGH)) {                    // Write PIR data
     Serial1.write(49);
    } else Serial1.write(48);
    Serial.println("End Data");               // Print "End Data" to computer console
    ack = 0;                                  // Reset flag and variable for next loop
    mess4 = 52;
    mess0 = 48;
    mode = 2;
   }
  }

}
  // Coordinator mode
  else {
   timer.run(); //Run the timer that periodically post JSON data to the web server every 5 seconds
   while (Serial1.available()) {
   Serial.println("Receive Data");
   delay(10);
   mess0 = Serial1.read();
```

```
  if (mess0 == 49 || mess0 == 50) {
    mess1 = Serial1.read();            //read message from RX buffer
    mess2 = Serial1.read();
    mess3 = Serial1.read();
    mess4 = Serial1.read();
    ack = 1;
    checkMode();
    Serial.print(mess0,HEX);           //display received message on console for debugging
    Serial.print(mess1,HEX);
    Serial.print(mess2,HEX);
    Serial.print(mess3,HEX);
    Serial.print(mess4,HEX);
    Serial.println();
    break;
  }
  // if not receive any message
  else {
    // clear the ZigBee buffer
    Serial1.read();
    Serial1.read();
    Serial1.read();
    Serial1.read();
    mess0 = 48;
    break;
  }
 }
 }
 // turn the LED on/off based on command from web server
 digitalWrite(outputLED, LED_status);
}
```

Router 2 Code:

```
/*************************************************************************
* FILENAME :        router_2_0219.ino
*
* DESCRIPTION :
*       Complete code for the router 2 Arduino in Smart Home IOT System
* NOTES :
*       In normal working condition, router 2 is in charge on controlling the LED and the fan peripheral.
*       When the coordinator failed, router 2 will be notified and change its default message's destination
*       to router 1
*
* AUTHOR :    THINH LY, ANH TRAN, DUC TRAN
*
**/
int strlenght = 5;
int message[5];
int data[5];
int ack = 0;
int mode = 0; //0 for command, 1 for request
int LED = 0;
int value;
unsigned long duration;
int mess4 = 0;
int mess3 = 0;
int mess2 = 0;
int mess1 = 0;
int mess0 = 0;
int fan_on = 0;
int outputLED = 3;
```

```
const int tempPin = 0;  // Analog pin A0
float deltaR = 0;
float deltaT = 0;
int temp = 0;
int request_temp = 0;
int start_time = 0;
int end_time2 = 0;
int end_time1 = 0;
int end_time3 = 0;
// variable to check if the coordinator fail and router 1 become the new coordinator
int wifi_fail = 0;
/*
 *  Setup function for ZigBee module
 */
void setup() {
  pinMode(outputLED, OUTPUT);
  Serial1.begin(115200);
  Serial.begin(115200);
  Serial1.write("\n\r");          //Bypass Xbee microcontroller
  Serial1.write("B");
  Serial1.println();
  delay(1000);
  if (Serial1.find("Bypass")) {
    Serial1.write("B");
  }
  // Configure the ZigBee module by entering AT mode
  Serial1.write("+++");
  delay(1000);
  while(Serial1.available()) Serial.write(Serial1.read());
  Serial1.write("ATDL4152ECD4\n\r");
```

```
  //Serial1.write("ATDL4152EC6B\n\r");
  delay(1000);
  while(Serial1.available()) Serial.write(Serial1.read());
  Serial1.write("ATID123\n\r");
  delay(1000);
  while(Serial1.available()) Serial.write(Serial1.read());
  Serial1.write("ATWR\n\r");
  delay(1000);
  while(Serial1.available()) Serial.write(Serial1.read());
  Serial1.write("ATCN");
  delay(1000);
  pinMode(LED_BUILTIN, OUTPUT);
}
/*
 * Check if the incoming message is command/request message
 */
void checkMode() {
  if (mess4 == 48) mode = 0;      //command
  else if (mess4 == 49) mode = 1; //request
}
/*
 * Check of the incoming message is to turn the LED on/off
 */
void checkCommand() {
  request_temp = mess3;
  if (mess3>47) {                  //LED command if byte 4th decimal == 48 or 49 ("0" or "1")
    if (mess3 == 49) LED = HIGH;      //LED on
    else if (mess3 == 48) LED = LOW;  //LED off
  }
  else {
```

```
    request_temp = mess3;             //read request temperature
    Serial.print("Request temp = ");
    Serial.println(request_temp);
    if (request_temp<temp) fan_on = 1;//turn fan on if request temperature < current temperature from temperature sensor
    else fan_on = 0;
  }
  mess0 = 48;
  mode = 2;
}
 /*
  * read temperature in Celsius from the TMP36 temperature sensor
  */
void readTempC() {
  int reading;
  reading = analogRead(tempPin);
  deltaR = 51000*reading/1024/(1-reading/1024)-50000;
  deltaT = deltaR/-4800;
  temp = deltaT+25; //celsius
}
/*
 * Main Code
 */
void loop() {
  // read temperature
  readTempC();
  // get message from either coordinator, or router 1
  // (if the coordinator fail)
  while (Serial1.available()) {
    delay(50);
    Serial.println("receive");
```

```
    mess0 = Serial1.read();
    if (mess0 == 49 || mess0 == 50) {
      mess1 = Serial1.read();        //read message from RX buffer
      mess2 = Serial1.read();
      mess3 = Serial1.read();
      mess4 = Serial1.read();
      ack = 1;
      end_time1 = millis();
      checkMode();   // Check whether message is request/command
      Serial.print(mess0,HEX);      //display received message on console for debugging
      Serial.print(mess1,HEX);
      Serial.print(mess2,HEX);
      Serial.print(mess3,HEX);
      Serial.print(mess4,HEX);
      Serial.println();
      break;
    }
    else { // if not receive the message, or the message is not in correct format
      // clear the ZigBee buffer
      Serial1.read();
      Serial1.read();
      Serial1.read();
      Serial1.read();
      mess0 = 48;
      break;
    }
  }
  // if receive critical message informing the coordinator has failed
  if ((mess0 == 49) && (mess1 == 49) && (mess2 == 49) && (mess3 == 49) && (mess4 == 49)
&& (wifi_fail == 0)) {
    // Change the default message's destination to router 1 by changing ZigBee configuration
```

```
  Serial1.write("+++");
    delay(1000);
    while(Serial1.available()) Serial.write(Serial1.read());
    Serial1.write("ATDL4152EC6B\n\r");
    delay(1000);
    Serial.println(1);
    while(Serial1.available()) Serial.write(Serial1.read());
    Serial1.write("ATID123\n\r");
    delay(1000);
    Serial.println(2);
    while(Serial1.available()) Serial.write(Serial1.read());
    Serial1.write("ATWR\n\r");
    delay(1000);
    Serial.println(3);
    while(Serial1.available()) Serial.write(Serial1.read());
    Serial1.write("ATCN");
    delay(2000);
    Serial.println(4);
    wifi_fail = 1;
    mess0 = 0;
    mess1 = 0;
    mess2 = 0;
    mess3 = 0;
    mess4 = 0;
}
if (mode == 0 && mess0 == 50) checkCommand();     // for Command message
else if (mode == 1 && mess0 == 50) {              // for Request message
  if (ack == 1) {
    Serial1.write(50);                            // Router number = 2
    Serial1.write(48);
```

```
    Serial1.write(48);
    Serial1.write(48);
    Serial1.write(temp);
    Serial.println("End Data");              // Print "End Data" to computer console
    ack = 0;                                 // Reset flag and variable for next loop
    mess4 = 52;
    mess0 = 48;
    mode = 2;
    end_time3 = millis();
   }
  }
  digitalWrite(outputLED, LED);
}
```