ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Smart Home Remote Control With Android Support

by
Vardalakis Nikolaos

Thesis
Submitted in partial fulfillment of the
Requirements for the acquisition of diploma degree of
Electrical & Computer Engineering

Examination Committee

Professor Kalaitzakis Kostas (*Supervisor)*
Associate Professor Deligiannakis Antonios
Associate Professor Kolokotsa Dionysia

Chania 2017

Abstract

Overuse of electricity consumption has caused a number of environmental and economic issues over the past decades. Typical household appliances are known to require a lot of power for everyday use and the average household contains a large number of such devices. Users that do not manage their appliances properly are responsible for significantly increasing the energy footprint of the average home. This paper has designed and implemented a smart home management and control system based on the concept of the Internet of Things, that will help reduce the wasted electricity of the average household, by utilizing the BeagleBone Black, Android, and Arduino technologies. The system presents an alternative to well-known building automation systems (BAS), mainly because of its low cost and its features. The system, when in operation, periodically gathers information about the smart home from the various sensors and is subsequently making smart decisions regarding the energy conservation of the smart home, while also providing basic security and alarm features. The design is non-invasive and can be implemented easily into any home. The system is characterized by flexibility, as it can be modified to interface with any commercial sensor, such as for example humidity sensors, or more accurate temperature sensors. The capabilities of the Android platform allow for providing an easy-to-use interface to the user, who have full control over their household appliances by controlling them remotely through their smartphones. The system requires minimum user supervision and helps the users make more conscious choices during their everyday lives, significantly reducing the energy impact of houses to the grid and eliminating wasted energy.

Keywords: Smart Home; Automation; BeagleBone Black; Android; Arduino; IoT; Energy Conservation;

Acknowledgements

Firstly, I must thank my supervising professor, Mr. K. Kalaitzakis, for his contribution to the work. The door to his office was always open and he greeted me every time I required his input. His guidance and support have been extremely helpful during the past few months, as well as during my time as a student. Furthermore, I must thank professor A. Bletsas and his post-graduate students for their assistance during my work. It was their work that allowed me to fully complete my thesis as they agreed to sacrifice their precious time in order to print in the fab-lab some of the PCBs I designed. They have been extremely helpful and their work was exceptional. I wish to also thank Mrs. A. Sergaki of the Electric Circuits and Renewable Energy Sources Laboratory for letting me use the facilities and laboratory equipment in my work. Special thanks to my committee members for their time and contribution. Finally, I wish to thank my family and friends for their support during my studies.

Table of Contents

Chapter 1.

Introduction....1
1.1. Problem Description....1
1.2. Thesis Goals....2
1.3. Chapter Contents....3

Chapter 2.

Presentation and Comparison of Technical Equipment....6
2.1. BeagleBone Black....8
2.2. Arduino Pro Mini....12
2.3. RFM22B-S2....18

Chapter 3.

Preparations....31
3.1. Preparing the BeagleBone Black....32
3.2. Preparing the Arduinos....36
3.3. Preparing the RFM22B....37

Chapter 4.

Module Types....40
4.1. Reed....41
4.2. Relay....43
4.3. Sensor....45

Chapter 5.

Wireless Communications and Networking....52

Chapter 6.

MySQL Database....63

Chapter 7.

System Workings....68
7.1. Module Synchronization....68
7.2. Normal Mode of Operation....72

Chapter 8.

Android Application....83

Chapter 9.

System Security....96

Chapter 10.

Implementation.....100

Chapter 11.

Conclusions.....105

List of Figures

Figure 2.1. Designed IoT System Diagram....7
Figure 2.2: BeagleBone Black Block Diagram....10
Figure 2.3: Full System Description Table....11
Figure 2.4: Arduino Pro Mini Pinout Diagram....14
Figure 2.5: SPI Timing During Read Mode....19
Figure 2.6: SPI Timing During Burst-Write Mode....19
Figure 2.7: RFM22B Interrupt Registers....21
Figure 2.8: Sensitivity at 1% PER vs. Carrier Frequency Offset....22
Figure 2.9: FSK vs GFSK Spectrum Comparison....23
Figure 2.10: General Packet Structure....23
Figure 2.11: RSSI vs Input Power Graph....27
Figure 2.12: 2.4 GHz WiFi Channels....28
Figure 2.13: 2.4 GHz ZigBee Channels....29
Figure 3.1: Approximate Occupied Bandwidth of a Digital Frequency Modulated Signal....37
Figure 3.2: RFM22B Breakout Board EAGLE Schematic....38
Figure 3.3: RFM22B Breakout Board EAGLE PCB Layout....39
Figure 4.1: Normally Closed Switch Operation....41
Figure 4.2: Reed PCB Schematic....42
Figure 4.3: Reed PCB Board Layout....43
Figure 4.4: Relay PCB Schematic....44
Figure 4.5: Relay PCB Board Layout....45
Figure 4.6: Photocell Circuit Connection....46
Figure 4.7: Photocell Resistance - Output Voltage Figure....47
Figure 4.8: Photo Sensor PCB Schematic....48
Figure 4.9: Photo Sensor PCB Board Layout....49
Figure 4.10: Outside Sensor PCB Schematic....50
Figure 4.11: Outside Sensor PCB Board Layout....51
Figure 5.1: General RF Packet Structure....54
Figure 5.2: SYNC Packet Structure....55
Figure 5.3: General RF ACK Packet Structure....56
Figure 5.4: Communication Diagram w/ Failures During REED Packet Transmission And ACK Reply....57
Figure 5.5: Communication Diagram w/ Failures During RELAY Packet Transmission And ACK Reply....58
Figure 5.6: Communication Diagram w/ Failures During PING Packet Transmission And MEAS Reply....59
Figure 6.1: MySQL Database UML Class Diagram....63
Figure 7.1: Communication Diagram During Synchronization....70
Figure 7.2: Collision Resolution Between Two SCM Devices During Synchronization....71
Figure 8.1: Android Application Login Screen....85
Figure 8.2: Android Application Main Menu Screen....86
Figure 8.3: Android Application Rooms Overview Screen....87
Figure 8.4: Android Application Room Details Screen....88
Figure 8.5: Android Application View Modules Screen....90
Figure 8.6: Android Application Reed Module Details Screen – Window Sensor....91

Figure 8.7: Android Application Sensor Module Details Screen - Inside Sensor.....92
Figure 8.8: Android Application Relay Module Details Screen - Lights Control.....94
Figure 8.9: Android Application Relay Module Details - Air Condition Control.....95
Figure 10.1: Implementation - Top-down View.....100
Figure 10.2: Implementation - Side View.....101
Figure 10.3: Main Program During Startup.....102
Figure 10.4: Main Program While Handling A User Request.....102
Figure 10.5: Main Program After Capturing Termination Signal - Normal Exit.....104

List of Tables

Table 2.1: Arduino Pro Mini Specs....13
Table 2.2: Arduino Pro Mini Digital Pin Special Functions....15
Table 2.3: Arduino Pro Mini Analog Pin Special Functions....15
Table 2.4: Arduino and Alternatives Comparison....17
Table 2.5: Minimum RX Settling Time - Preamble Length....25
Table 2.6: Temperature Sensor Range....26
Table 3.1: BeagleBone Black SPI to RFM22B Pin Connections....34
Table 3.2: Arduino Pro Mini SPI to RFM22B Pin Connections....36
Table 4.1:SCM Device Basic Cost 40
Table 4.2: Reed SCM Device Cost....41
Table 4.3: Relay SCM Device Cost....44
Table 4.4: Sensor SCM Device Cost....47
Table 5.1: RF Packet Types....60
Table 7.1: Temperature Discrete Values....78

List of Acronyms

AP	Access Point
APM	Arduino Pro Mini
BBB	BeagleBone Black
DRAM	Dynamic Random-Access Memory
DTO	Device Tree Overlay
eMMC	embedded Multi-Media Card
GPIO	General Purpose Input-Output
IoT	Internet of Things
MCU	Microcontroller Unit
SCM	Sensor/Control Module
SPI	Serial Peripheral Interface
SSH	Secure Shell

Glossary

CRC	A cyclic redundancy check (CRC) is an error-detecting code commonly used in digital networks and storage devices to detect accidental changes to raw data.
micro-SD	A micro-SD card is a format for removable flash memory cards. Commonly used in mobile devices, they are found in almost every modern cellphone, media player and other handheld devices.
SPI	The Serial Peripheral Interface bus is a synchronous serial communication interface specification used for short distance communication, primarily in embedded systems. The SPI bus specifies four logic signals: • SCLK : Serial Clock (master output) • MOSI : Master Output, Slave Input • MISO : Master Input, Slave Output • SS : Slave Select
Sysfs	*sysfs* is a pseudo file system provided by the Linux kernel that exports information about various kernel subsystems, hardware devices, and associated device drivers from the kernel's device model to user space through virtual files.
Star Network	Star networks are among the most common computer network topologies. A star network consists of a central node, which acts as a conduit to transmit messages. Every other node is connected and communicates directly with the central node.
Syslog	Most systems have a facility called “Syslog” that allows programs to submit messages of interest to system administrators and can be configured to pass these messages on in various ways, such as printing on the console, mailing to a particular person, or recording in a log file for future reference.

Chapter 1.

Introduction

1.1. Problem Description

The overall power consumption of the world has been steadily increasing over the past decades with most of the energy being produced from burning oil and coal[11]. At the same time, the percentage of the energy produced by renewable energy sources has been slightly increased, yet it is not nearly enough to cover our global energy needs. Greece, for instance, is ranked 47th in the world in electricity consumption as of 2014[40], while most of the electricity produced in Greece comes from factories burning coal and oil. It is also worth noting that the average household electricity consumption in first-world countries has significantly increased as well, with Chinese and US homes consuming the most electricity in 2016[16]. From these facts we can deduce that as a direct result the carbon footprint of the world has been also steadily increasing, as more and more energy is required in order for humans to cover our needs while at the same time, most of the produced energy comes from non-renewable energy sources.

Recent advances in technology have led to the evolution of the vision of the IoT. A convergence of multiple technologies, such as embedded systems, automation, and wireless sensor networks, has been a catalyst in creating more complex IoT systems. Some of these systems, such as smart homes and building automation, hold much promise in limiting the negative effects of excessive power consumption by automating various aspects of the household for both optimal power consumption and pleasant user experience.

However, most of the currently commercially available smart home solutions suffer from quite a few flaws. For example, commercially available solutions have a high cost of installation and setup, as prior schematics and architectural design is required in order to fully integrate the system into a home. At the same time the hardware used in order to implement the smart home control system is also very expensive, such as the required remote controls, the sensors and the gateways. Furthermore, the maintenance and upgrading costs, combined

with the cost of the replacement of individual parts of the system due to hardware failure are not to be taken lightly. As a result, the extreme costs deter users from investing in smart home technologies in general, as they do not treat it as a profitable investment.

One other issue that has arisen from the rapid evolution in technology is the increased complexity of the IoT. Most users are not familiarized with newer technologies, especially those above a certain age, hindering the expansion of IoT solutions, such as smart homes. At the same time, most commercially available solutions aim to increase the features of their provided systems in order to make them more competitive in a competitive and growing market, with little regard to user-friendliness. Such practices have led to users being afraid of installing and using smart devices in their homes because of the difficulties they would face while operating them.

Another possible problem is the high energy consumption of most commercial smart home solutions that can, in some cases. As most companies aim to provide as many features as possible, their main goal being to increase the quality of life of the user, do not take into account the electricity required for their system's operation. As such, energy-hungry devices, such as industrial motors and actuators, installed in the home will further encumber the household instead of helping the user reduce their current electricity consumption. In practice, this only serves in defeating the environmental purpose of installing smart home control systems and does not help the user reduce their household electricity consumption nor their electricity bills.

Finally, every commercial solution is protected by law and does not allow access to neither their source code nor their hardware. Therefore, any modifications are made only by licensed professionals. These types of systems are characterized by little to no potential for expansion, a lack of flexibility, and many prerequisites in order to be utilized to their fullest.

1.2. Thesis Goals

The main purpose of the thesis was to develop a smart home control system to address the rising household electricity consumption, while keeping in mind the cost of the implementation and developing the system in the most user-friendly way possible. The goal

was creating a system that will help users make more conscious decisions regarding their electricity consumption and also prevent many of the common electricity wasting actions, such as forgetting to turn off the lights after leaving the home, or turning on the air conditioning system in a room with the windows open and is, therefore, improperly insulated.

In more detail, the developed system must above all be open-source and open-hardware. Users must be able to freely distribute the developed software, while the hardware schematics must be also freely available for modification and study. At the same time, the developed system must be easy to install and it should be available for installation into any household layout with limited to no restrictions. The system also ought to be flexible and easily expandable, so it can interface with any sensor the user requires in a fairly quick and easy manner.

However, the most important aspect of the developed system was its cost and its security. The hardware must be as low-cost as possible, with little impact to its capabilities. Furthermore, installation and maintenance costs should be minimized, as well as the costs of possible future expansions, if the user wishes so. In addition to these costs, the system should not require a complex hardware remote control but instead it should allow the users to control their home from their smartphone devices and their computers. The system also must offer adequate security to the user, in the form of both an physical intruder-detection alarm and a secure wireless sensor network with controlled access and strong security credentials.

Finally, the user's interaction with the developed system ought to be simplified and intuitive, rather than complex and tiresome. Therefore, in order to allow for a pleasant experience to inexperienced new users, the remote control application was designed to be as user-friendly as possible, using images to display information to the user and simple vocabulary.

1.3. Chapter Contents

In the second chapter, the specifications of the developed system are presented in detail, based on the previously mentioned objectives. The overall organization of the system and a relevant diagram is also presented and explained. Furthermore, research is carried out in

order to select the best hardware solutions (such as sensors, MCUs, platforms) for the developed system. Each hardware selection for the developed smart home system is presented in its own subsection with a complete disclosure of its features and specifications. Finally, a detailed comparison is made between the selected hardware component and its possible alternatives, in terms of their total cost, their provided features, and their overall capabilities. Thorough explanation of the reasons why the selected hardware component is ideal for the current work are also presented.

Having selected the ideal hardware solutions for the proposed system, the third chapter describes the preparations required before these parts were to be connected into creating a fully working home automation system that provides security, allows extensive home management and energy saving. Each subsection of the third chapter covers all the necessary steps taken in preparing the corresponding hardware subsystem for integration with the others, both in terms of programming and hardware preparations.

In the fourth chapter, the various control and sensor devices developed are presented with details regarding the hardware, the way each device operates and the corresponding PCB that was designed for each device to achieve its intended purpose. All of the miscellaneous electronics and sensors for each device are presented and their purpose is discussed in detail. Furthermore, the cost of each device is calculated and presented, including the price of the MCU.

The fifth chapter discusses in detail the networking aspect of the developed system. The wireless RF sensor network functionality is presented with each of the transmitted packets and its intended purpose. Also, the server's features are listed and the way the wireless communications work. Furthermore, the way that the user can control the system remotely from the android application is explained in detail, along with the server's infrastructure.

In the sixth chapter, the MySQL database is discussed. The UML diagram of the implemented database is shown and each of the tables in the database is presented in detail. The purpose of the MySQL database is also discussed, as well as the information it stores and the way these are presented to the user.

The seventh chapter is the most important chapter in the thesis, as it focuses on the actual

implementation of the smart home system. During this chapter, the various stages during the operation of the system are explained, such as the way the miscellaneous devices are registered on the gateway and how the main program functions. The control algorithm that enables the smart home to regulate the environment inside the smart home is discussed, with its various stages and procedures. Furthermore, the way the main program responds to various events is presented in detail. Finally, in this chapter the reader will gain complete understanding of why the system performs exceptionally in controlling every connected device inside the smart home, as well as the way the user requests are processed at the core of the system.

The eighth chapter focuses on the development of the Android application and the user interfaces. The information contained in the chapter is accompanied by examples of the various user interface screens the user is presented with during interaction with the system. Most of the methods of the software developed are presented, while the key points are highlighted. This chapter also focuses on the ease-of-use aspect of the developed application, as within it, detailed explanations are given for each of the allowed actions.

The final chapter of the thesis focuses on the security aspect of the developed system. Security as a whole is discussed, whether it involves the alarm features of the smart home, the security credentials, or the way access to the system is secured. This chapter is extremely important, as it highlights the main security points of the developed smart home control system and discusses whether the security is adequate against various network hacking attacks and system intrusions.

Finally, the conclusions drawn from the thesis are presented. In this section, the main points that arose during the implementation of the smart home control system are presented. An overview is made of the whole work and the conclusions drawn during the thesis. Furthermore, reference to future work and possible expansions of the system are made, as proposals that add to the capabilities and features of the developed smart home control system are presented.

The final section of the thesis contains a complete list of the bibliography sources that were used during the preparation of this thesis.

Chapter 2.

Presentation and Comparison of Technical Equipment

Connecting physical objects and devices to the virtual world and allowing them to transmit or receive data and be controlled remotely by the user is the basis of the IoT[30]. There are currently many BAS or smart home solutions available, but they have many drawbacks when it comes to installation, service and overall cost. One such solution is the KNX Building Control System[26], a solution providing safety and security with an integrated alarm system, flexibility in controlling many different devices and comfort, while being energy-efficient. This is a complete industrial, but high-end solution, as it is costly and requires prior knowledge on a number of variables in order to be installed or integrated into a home. BAS systems are mostly wired, therefore they require prior knowledge of the electrical and architectural design plans of the building, limiting the options in the installation of each sensor and device significantly. Also, these automation solutions require specific remote controls in order to interface with the various system variables, which can cost up to €5,000 depending on the application.

The system described in the proposed work is designed to serve as a low-cost alternative to the BAS, countering the flaws described above, without sacrifices in usability and user comfort. It can be used as an alarm system, it can control many different electrical devices and can help the user make better choices in order to minimize the energy consumption of the smart home, a feature which is missing from the KNX Building Control System. Operating in the 868 MHz ISM band, the proposed system allows wireless communication between the hub and the devices, eliminating the prior knowledge requirement on the electric plans. It can be installed easily and integrated in most homes without requiring specialized and trained personnel. The user can communicate with the hub and control the devices from their smartphone or personal computer, thus eliminating the need of specialized and expensive remote control devices. The designed system uses a BeagleBone Black (BBB) platform as a hub. The user installs the sensors and control devices inside the house according to their needs, and synchronizes them to the hub from designed android application. Synchronization is done separately for each room inside the house, with the user inserting a name for the

room in the android application and all the devices that are installed in that specific room are entered into the database. This way the user has complete control over the installation procedure and can create a home automation specific to their needs. After the system is synchronized, the user can turn on or off the controllable devices, review the current status of each sensor and is notified via email of suggestions to maximize the smart home energy saving. The whole system is controllable from the android application and the web pages hosted by the BBB. The main devices the system can control are air conditioning units and lights, but any device can be controlled with appropriate modifications to allow full control over the whole home. The general layout of the IoT system is presented in **Figure 2.1**.

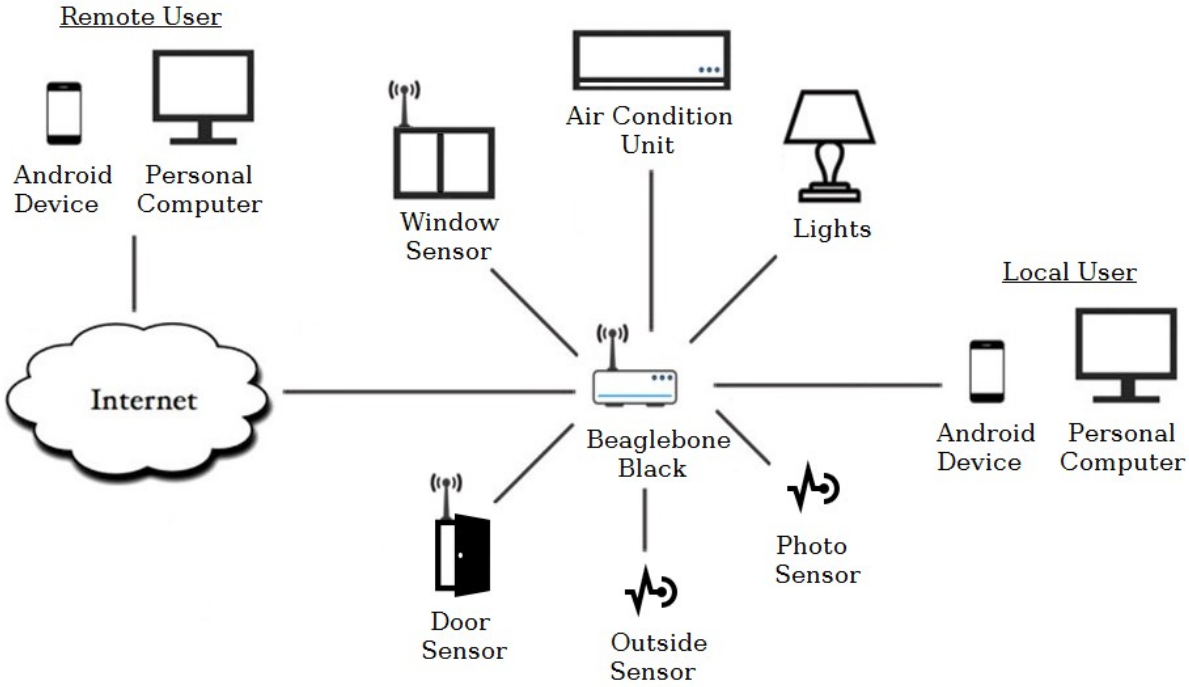

Figure 2.1. Designed IoT System Diagram

One of the major challenges in any engineering work is making the developed system as user-friendly as possible, while keeping the total cost as low as possible. With those limitations in mind, the hardware chosen is mainly comprised of low-cost embedded platforms and hardware, without any compromise in capabilities or features. After extensive research and comparison of many well-known embedded systems, it became evidently clear what platforms should be included in each stage and why. The following sub-sections contain a detailed description of each component and how it compares to the available alternatives.

2.1. BeagleBone Black

The BBB is a development platform, used mainly by amateur developers and hobbyists alike, in a wide range of electronics projects. It is a credit-card sized, low-cost, open-hardware and open-source computer, highly expandable and extremely capable in running many different Linux distributions without any performance hindrance. That is mainly due to the **Sitara XAM3359AZCZ100 Cortex A8 ARM**, the main processor of the unit. As it is evident, the processor is based on the ARM architecture and is manufactured by **Texas Instruments**. These processors are enhanced with image, graphics processing, peripherals and industrial interface options. The processing speed of the unit is 1 GHz, which is considered fast for an embedded platform, compared to other options with approximately the same cost.

The BBB supports three different types of memory. Those are a RAM memory chip, an internal flash memory and micro-SD memory cards. Those are listed below.

- The supported RAM memory is a 512 MB and 800 MHz DRAM integrated circuit, using the DDR3L technology.
- The supported flash memory is a 4 GB eMMC storage.
- External micro-SD memory cards of any storage capacity are supported.

One major advantage of the BBB is the ability to expand its capabilities through the provided cape expansion headers, labeled P8 and P9 respectively. They number 92 female header pins in total, through which the BBB can perform various tasks, such as powering other hardware through the VDD slots with DC both 5 V and 3.3 V, reading analog voltages via the 7 analog inputs provided, which are connected to an ADC chip with 12 bits of resolution, or miscellaneous digital input-out, through the 69 available GPIO pins. Some allow for SPI communication with other embedded devices, such as sensors. All the digital GPIOs are accessible and fully controllable through the sysfs pseudo file system Linux. Finally, many expansions can be connected, in the form of cape plug-in boards, making it easy to augment the capabilities of the board with features such as motor control, LCD screens, battery power. It even allows users to create and prototype their own circuits, with capes that function as a breadboard, greatly simplifying the development process.

Arguably one of the most user-friendly platforms, connecting and interacting with the BBB

presents no problems, even for the inexperienced users. Connecting the BBB with a personal computer is done via the on-board mini USB port. Users can subsequently connect and control the BBB through the SSH protocol. Alternatively, the user can connect a display to the onboard HDMI slot and all the necessary computer peripherals, using the classic USB slot provided, and use the BBB like any other PC. Powering the platform is achieved via the USB cable whenever it acts as a USB peripheral on a host PC, requiring no additional power supply, or via an external 5 Volt power supply, which is connected to the on-board DC barrel connector. The DC connection option is preferable whenever the platform is required to be autonomous, instead of depending on a PC host.

Given the versatility of the BBB, networking can be a trivial matter, one that can be approached by a number of ways, depending on the preferences of the user and the interior layout of the smart home. Networking is provided by using a common Ethernet cable, connecting the BBB directly to the Internet router inside the home, by connecting the BBB with an external WiFi USB dongle, allowing for wireless communication over WiFi protocol with the router, or by expanding the BBB through a cape that handles the Internet networking, such as the Wireless Connectivity Cape. Since both the overall cost and ease of installation in any type of home are two of the most important limitations, using a simple Ethernet cable while placing the BBB in a physically accessible spot is encouraged.

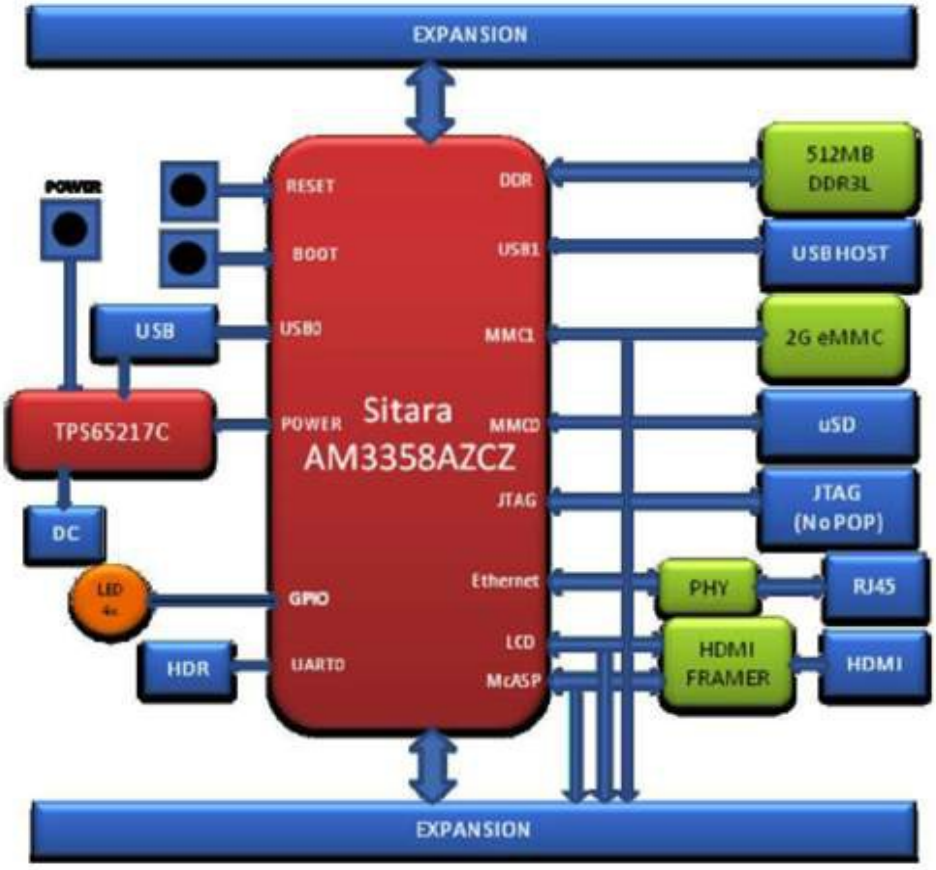

Figure 2.2: BeagleBone Black Block Diagram

The BBB is one of the most user-friendly and flexible development platforms currently in the market, which is also evident by the wide range of supported operating systems. It comes with a lightweight version of Linux called **Angstrom**, which is mainly oriented towards a variety of embedded systems. On the BBB official site[8], it is stated that users can choose from many well-known Linux distributions, such as Arch and Debian, among others. Operating systems can either run from the micro-SD card directly, meaning that booting in that specific OS can only be done while the micro-SD card is inserted, or by flashing the on-board eMMC flash memory, overwriting the previous OS. The most common alternative to the Angstrom distribution is the Debian distribution, which can be found from the BBB official site and is largely supported.

Energy consumption during the normal mode of operation was the final and most important limitation we had to consider in our design. The BBB consumes a moderate amount of electricity and requires little power, even under computational stress. More

specifically, the current drain ranges from 210 mA to 460 mA, when operating at the nominal voltage of 5 V, consuming from 1.05 to 2.3 Watts of power, which is more than reasonable.

	Feature	
Processor	Sitara AM3358BZCZ100 1GHz, 2000 MIPS	
Graphics Engine	SGX530 3D, 20M Polygons/S	
SDRAM Memory	512MB DDR3L 800MHZ	
Onboard Flash	4GB, 8bit Embedded MMC	
PMIC	TPS65217C PMIC regulator and one additional LDO.	
Debug Support	Optional Onboard 20-pin CTI JTAG, Serial Header	
Power Source	miniUSB USB or DC Jack	5VDC External Via Expansion Header
PCB	3.4" x 2.1"	6 layers
Indicators	1-Power, 2-Ethernet, 4-User Controllable LEDs	
HS USB 2.0 Client Port	Access to USB0, Client mode via miniUSB	
HS USB 2.0 Host Port	Access to USB1, Type A Socket, 500mA LS/FS/HS	
Serial Port	UART0 access via 6 pin 3.3V TTL Header. Header is populated	
Ethernet	10/100, RJ45	
SD/MMC Connector	microSD , 3.3V	
User Input	Reset Button Boot Button Power Button	
Video Out	16b HDMI, 1280x1024 (MAX) 1024x768,1280x720,1440x900 ,1920x1080@24Hz w/EDID Support	
Audio	Via HDMI Interface, Stereo	
Expansion Connectors	Power 5V, 3.3V , VDD_ADC(1.8V) 3.3V I/O on all signals McASP0, SPI1, I2C, GPIO(69 max), LCD, GPMC, MMC1, MMC2, 7 AIN(1.8V MAX), 4 Timers, 4 Serial Ports, CAN0, EHRPWM(0,2),XDMA Interrupt, Power button, Expansion Board ID (Up to 4 can be stacked)	
Weight	1.4 oz (39.68 grams)	
Power	Refer to Section 6.1.7	

Figure 2.3: Full System Description Table

There are quite a few alternatives to the BBB which provide similar capabilities, yet were found lacking in major fields of importance, violating some or all of the set restrictions of energy consumption, ease of usage and low cost. The alternatives were also lacking in some of the features they provided, given their prices. One of those alternatives was the Raspberry Pi 3 Model B board[33], which is available at lower basic cost, offers two USB slots instead of

the single USB port on the BBB, supports Bluetooth connectivity, offers native 802.11n Wireless LAN support and outperforms the BBB in the graphics and image processing fields. In contrast, the Raspberry is extremely energy-hungry and inefficient, as it draws 2.5 A during the normal mode of operation. Also, it provides no internal storage, therefore users are required to also obtain at least one micro-SD storage card, increasing the overall cost. Finally, the number of female pin-outs available on the Raspberry are significantly fewer, reducing the capabilities of the board and the system flexibility. In comparison, the BBB was the better choice, since it draws less current, has internal storage, making the use of a micro-SD card situational and offers more expansions. Also, the great advantages of the Raspberry over the BBB were in the fields of image processing and graphics, a field of little importance in our research.

Other potential choices included platforms such as the ODROID-C2[3], PINE A64[31] and UDOO[41]. The BBB was preferred over the ODRIOD-C2 because, while both boards were in the same price range of 40-50$, the ODRIOD-C2 had a reduced amount of GPIO pins and consumed more energy during normal mode of operation. Also the BBB has greater integrated storage, which was necessary in the later stages of design. Next, while the PINE A64 was only a fraction of the cost compared to the BBB as it costs around 15$ compared to the 45$ of the BBB, the main problems faced were the difficulty in finding a proper power supply unit for the board to work, while at the same time the lack of a open community and proper support, making it less user-friendly. The UDOO platform costs twice as much as the BBB with a cost around 90$, depending on the platform version. Also, the official OS images are not full distributions, making it less user-friendly than expected.

As proven, the BBB is the preferable platform for development, given the research limitations. It consumes little power, it is user-friendly, it is expandable and flexible and comes at a low cost. The current version of the BBB and available for purchase is revision C and it can be purchased online at 45$.

2.2. Arduino Pro Mini

Arduino is an open source software and hardware project, aiming to simplify the process of creating devices for interaction with the environment, both for professionals and amateurs.

Based on microcontroller board designs, they provide a simple way to handle analog and digital signal inputs and outputs. Inputs provide information coming from the environment, another board, or the user, while outputs can control other devices. All boards feature serial communication interfaces, simplifying the communication between users and boards. Programming is made easy through the official Arduino IDE, which also supports writing programs in C/C++. The Arduino platform is widely established among engineers, developers and hobbyists alike, as it is one of the most cost-effective solutions, while being extremely easy to use and handle.

While the Arduino project supports a wide array of boards, operating at different voltages and via many different interfaces, it is recommended for permanent and semi-permanent designs to use the Arduino Pro Mini (APM)[5]. This board is extremely small and was developed for applications where available space is limited. Its size offers little as a prototyping board, while at the same time, makes it perfect for permanent and semi-permanent installations. In **Figure 2.3** the visual datasheet of the APM is presented, with the complete pinout diagram and each available mode and function of every pin on the side. The full specifications of the board are shown below in **Table 2.1**:

Arduino Pro Mini Specs	
MCU	ATmega328
Digital I/O pins	14
Analog Input pins	6
PWM pins	6
UART	1
SPI	1
I2C	1

Table 2.1: Arduino Pro Mini Specs

The APM board also has three different memory slots. These are:

- 32 kB of flash memory for storing code, of which 512 B are used for the bootloader.
- 2 kB of SRAM.
- 1 kB of EEPROM, which can be read and written from the software via the EEPROM

library.

There are two available versions of the APM, with small differences. The first version runs at 3.3 V and the clock speed is 8 MHz, while the other runs at 5 V and the clock speed is doubled, reaching 16 MHz. In the smart home system design, the preferred version is the 3.3 V, 8 MHz board. The reason being the RF module, which requires 3.3 V for both power and communication. More details about the RF module in the next section of the current chapter. The design of the APM board originally comes from SparkFun Electronics, yet the schematics are open to the public, because of the open-source hardware policy of the Arduino project, effectively reducing the cost of the board to the sum of the PCB printing and the electronic parts, making it an extremely low cost option.

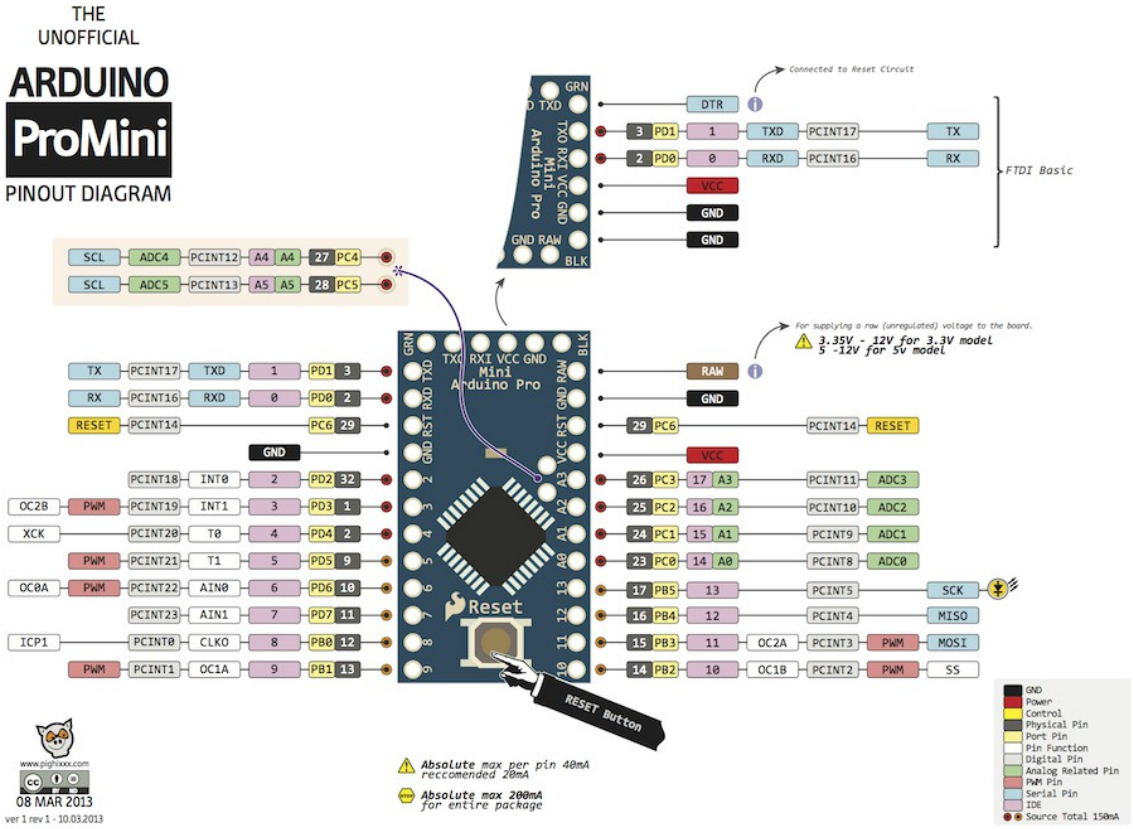

Figure 2.4: Arduino Pro Mini Pinout Diagram

Each digital pin on the APB can be configured both as an input or an output. Depending on the board model, the logic level of those input and output pins can be either the standard TTL logic level of 5 V, or the 3.3 V CMOS Logic Levels, with a maximum of 40 mA of both

input and output current per pin. Internal pull-up resistors are also available. Some digital pins have specialized functions, as shown below in **Table 2.2**:

Digital Pin Specialized Functions		
Function	**Pins**	**Description**
Serial	0 (RX), 1 (TX)	Used to receive (RX) and transmit (TX) TTL serial data.
External Interrupts	2, 3	Can be configured to trigger an interrupt on high and low values and rising and falling edges.
PWM	3, 5, 6, 9, 10, 11	These pins provide 8-bit Pulse Width Modulation output.
SPI	10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK)	Provide for SPI communication. Software library required by default.
LED	13	The built-in LED.

Table 2.2: Arduino Pro Mini Digital Pin Special Functions

Except the Digital Pins, the APM also supports 6 Analog Input pins that can read voltages in the range of GND to VCC. The internal ADC of the ATmega328[6] has a 10-bit resolution. Some of the analog pins have specialized functions, as shown below in **Table 2.3**:

Analog Pin Specialized Functions		
Function	**Pins**	**Description**
I2C	A4 (SDA), A5 (SCL)	I2C communication support. Wire library included
Reset	RST	When this line goes LOW, the board software-resets.

Table 2.3: Arduino Pro Mini Analog Pin Special Functions

A great advantage of the Arduino Project is the ability to add expansion boards that augment the capabilities of the boards, named shields. While no shields were used in the

proposed smart home automation system, the option to expand the abilities of the board is crucial in any desired future work. For example, in larger installations there may be a need to topologically label each module, which is an ability offered by the GPS Arduino shield. If audio input was required for inside the home voice control, implementing a baby monitor or even an intercom module, the spectrum shield would simplify the process greatly. The implementation of motor-based control over the home for locking and unlocking doors and windows, or even raising and lowering window covers can also be achieved by using a plethora of motor driver shields. Finally, there is a camera shield, named CMUcam, which also adds vision to the Arduinos, a feature that proves useful for distance home monitoring or even capturing snapshots after the occurrence of certain events. All the above are in no way connected to the energy monitoring scope of the system under development, therefore they would simply add to the cost, with no real benefits. Nonetheless, they can be implemented in the future, in order to build a fully automated home.

The APM was the best choice for creating sensor and actuator modules around the smart home that are barely visible to the user and do not interfere with the home layout. It allowed for great flexibility at installation and simplified the necessary maintenance greatly, while at the same time, it does not interfere with the user's visual perception. The current draw for normal operation of the 3.3 V board is 4.74 mA, 15.64 mW of power consumption in the active mode, but with certain modifications the current draw can go even lower. According to a guide[7] posted in the Home Automation Community website, an APM can be modified in order to achieve a minimum of current draw and lengthen the battery life. More specifically, by disabling the on-board LED, the current draw drops to 3.9 mA and the power consumption drops to 12.87 mW when the board is in active mode. For even further power saving, the on-board regulator can be bypassed completely, by powering the board through the VCC pin instead of the RAW voltage input, reducing the current draw to 3.58 mA and the power consumption is reduced to 11.81 mW. The on-board linear regulator can also be replaced with a non-linear regulator, reducing even further the power consumption of the board in the active mode while powering the board through the RAW voltage input. These notes are of great importance for running the APM on batteries, achieving many years of operation on a single coin battery. In the proposed home automation system, the power is provided through power outlets, since the batteries cannot provide security and stability in a day to day basis,

plus the power consumption of the devices is very low compared to the house appliances.

The board comes without built-in USB circuitry, therefore an off-board USB-to-TTL serial converter is required for uploading sketches to the APM. The available connection options are either through an FTDI serial cable, or through a programmer chip. In the proposed home automation system, the Arduinos were programmed using the FTDI chipset **FT232RL**[20] USB to Serial breakout board, yet the choice is up to the user. Programming can be done via the programming headers on the board. Connecting the Arduino with the FT232RL breakout board was done as shown in the **Figure 2:3**.

The alternatives to the APM are the Teensy 3.2 and the Teensy LC, two boards that share many similarities with the APM. The detailed comparison follows in **Table 2.4.**

	Teensy 3.2	**Teensy LC**	**Arduino Pro Mini**
CPU	32 bit ARM Cortex-M4 72 MHz	32 bit ARM Cortex-M0+ 48 MHz	ATmega328 8MHz (external resonator)
Flash Memory	256 kB	62 kB	32 kB
RAM	64 kB	8 kB	2 kB
EEPROM	2 kB	1/8(emu) kB	1 kB
Analog Inputs	21 High Resolution	13 High Resolution	6
Digital I/O Pins	34	27	14
PWM outputs	12	10	6
Timers	7	7	3
USB	Yes	Yes	Off Board
UARTs	3	3	1
Real Time Clock	Yes		NO
Operating Voltage	5 V	3.3 V	3.3 V
Miscellaneous	SPI, I2C, I2S,CAN Bus, IR modulator, Touch Sensor Inputs	SPI, I2C, I2S, Touch Sensor Inputs	SPI, I2C, Low-voltage board

Table 2.4: Arduino and Alternatives Comparison

As shown above, the Teensy boards are more capable than the APM. They are also more costly, with their prices being higher in SparkFun. More specifically, the price of a simple APM board on SparkFun Electronics is 9.95$, while the Teensy 3.2 is priced at 19.95$ and the

Teensy LC at 12.95$. The requirements of the intended work are met with all of the listed boards, therefore the selection criteria is based on the pricing and the expandability of each board. The APM was selected because if offers a vast amount of expansions (Arduino shields), providing a sound basis for future work and flexibility during development, while at the same time is offered at the lowest pricing. All boards are open-source hardware and can be manufactured freely, as their schematics are available for acquisition online.

2.3. RFM22B-S2

The RFM22B[35] is a highly integrated, low cost and low power ISM-band transceiver module from HopeRF microelectronics, intended for home automation, remote control and home security/alarm applications. The supply voltage can range from 1.8 V to 3.6 V, with the typical voltage being 3 V. The current draw during RX mode is 18.5 mA, while during TX mode we have the maximum current draw at 85 mA for transmissions at +20 dBm. The transceiver module can transmit and receive in the frequency range of 433/470/868/915 MHz ISM bands, supporting FSK, GFSK and OOK modulation. Low receive sensitivity (-121 dBm) coupled with a maximum of +20 dBm power output allows for improved link performance and stable communication over extended distances, a feature that is enhanced even further by supporting built-in antenna diversity and frequency hopping. The communication data rate can be set anywhere from 123 bps to 256 kbps, while the module also allows for automatic packet handling, sync word and preamble detection, and has two (2) separate 64-byte FIFOs, one for incoming (RX) data and one for outgoing (TX) data. These features can simplify the process of programming and using the module, while allowing the flexibility of choosing to enable or disable them to create a custom communication protocol and data handling process. Other notable features include the on-board general purpose ADC, an integrated temperature sensor, a low battery detector and the three (3) separate GPIO pins.

Programming the module can be done by modifying the data of the available registers. In order to do so, the module communicates over a SPI interface with the host MCU. A SPI transaction is a 16-bit sequence, starting with the R/W bit, followed by the register address and the data. During a write operation, the R/W bit is set to 1 and the data field contains the byte to be written in the register specified by the address field, while during a read operation,

the R/W bit is set to 0, followed by the address of the register to be read from and the data. During a read, the data bits are ignored, therefore they can be either 0 or 1 without distinction. The SDO signal is the output from the RFM22B which during a read operation contains the data bits from the register read. During a write operation however, the SDO signal is held low and ignored. The Read mode SPI transaction is presented in **Figure 2:5**.

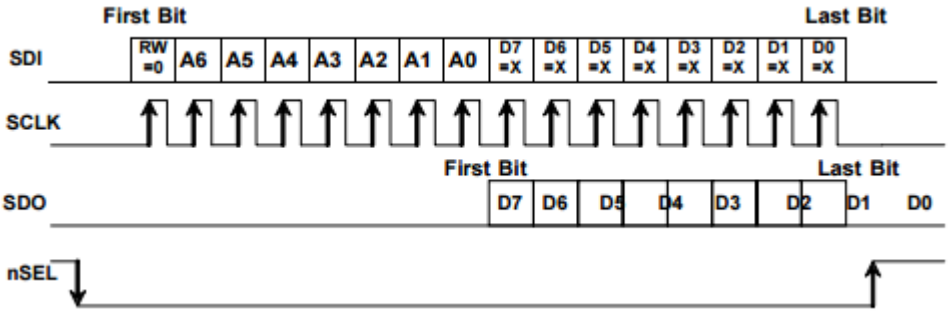

Figure 2.5: SPI Timing During Read Mode

The SPI interface also contains a burst read/write mode as well, allowing for sequential register writes or reads without having to resend the register address. If the SS (nSEL) pin is held low and continuous SCLK pulses are sent, the SPI interface automatically increments the register address and continue reading/writing as already specified. The bits are latched into the RFM22B every eight (8) clock cycles, while the clock rate is flexible, with a maximum rate of 10 MHz. During a burst-read operation the SDO signal outputs the data read from the sequence of registers starting from the address specified, until the nSEL signal goes high and the reading is ceased. During a burst-write operation, the SDO signal is held low and ignored, just as in a simple write operation. A burst-write operation is presented in **Figure 2:6**.

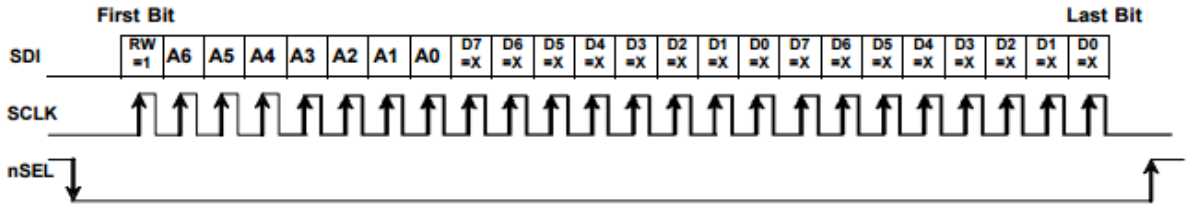

Figure 2.6: SPI Timing During Burst-Write Mode

There are four primary states in the radio state machine, each with different current consumption and features, providing optimal functionality depending on the application. The

module can switch between states by modifying the contents of the 0x07 register, named Operating Mode and Function Control 1. The available states are:

- SHUTDOWN state offers the lowest current consumption. It can be entered by driving the SDN pin of the chip high. During this state, the contents of the registers are lost and there is no SPI access.

- IDLE state can be selected from the 0x07 register and has five different modes, all of which have a tradeoff between current consumption and response time to TX/RX modes. Those are STANDBY, SLEEP, SENSOR, READY and TUNE. STANDBY mode has the lowest current consumption at 450 nA and a response time of 800 µs to each of the TX/RX modes while register values are preserved and SPI access is allowed. SLEEP mode has a current consumption of 1 µA and the same response time as STANDBY mode. The wake-up timer can be set to wake the module. SENSOR mode draws the same current and shares the same response time as SLEEP mode, while the embedded sensors can be enabled as well. READY mode draws 800 µA and is designed mainly for fast transition to TX/RX, therefore the response time is faster, at 200 µs. In TUNE mode the PLL remains enabled, resulting in the most current draw at 8.5 mA. This mode is designed for frequency hopping spread spectrum systems (FHSS) and is characterized by the fastest response to TX mode, as the PLL remains locked, and 200 µs time to RX mode.

- TX state may be entered from any of the IDLE modes. The built-in sequencer goes through all the required actions for transitioning to this state and the packet transmission.

- RX state may be also entered from any of the IDLE modes. The built-in sequencer goes through all the required actions for transitioning to this state, enabling receiver circuits and enabling the receive mode in the digital modem. The modem can be set to automatically handle packets, perform sync word, header and CRC checks, update status registers, AGC, AFC and bit synchronization.

The module is able to generate interrupts on certain events, by driving the nIRQ output signal to low (0 V or GND signal). When an event happens, the Interrupt Status 1 and 2

registers change by setting the appropriate bit. When one or more interrupt events occur, the nIRQ pin will go low and remain low until the contents of the Interrupt Status registers are read. The corresponding interrupt enable bit must be enabled in the registers Interrupt Enable 1 and 2 for the nIRQ pin to go low, but the contents of the Interrupt Status registers can be read anytime. The event interrupts can be used to inform the MCU when a new packet is received. The registers responsible for enabling and handling interrupt signals are presented in **Figure 2:7**, with each bit showing the corresponding interrupt it represents.

Add	R/W	Function/Description	D7	D6	D5	D4	D3	D2	D1	D0	POR Def.
03	R	Interrupt Status 1	ifferr	itxffafull	itxffaem	irxffafull	iext	ipksent	ipkvalid	icrcerror	—
04	R	Interrupt Status 2	iswdet	ipreaval	ipreainval	irssi	iwut	ilbd	ichiprdy	ipor	—
05	R/W	Interrupt Enable 1	enfferr	entxffafull	entxffaem	enrxffafull	enext	enpksent	enpkvalid	encrcerror	00h
06	R/W	Interrupt Enable 2	enswdet	enpreaval	enpreainval	enrssi	enwut	enlbd	enchiprdy	enpor	01h

Figure 2.7: RFM22B Interrupt Registers

In order to receive and transmit data, the carrier frequency must be programmed into the module. Adjusting the frequency settings can be done from the registers 0x73 to 0x77. The frequency range can be partitioned into two separate bands, the low-band, for frequencies from 240 MHz and up to 479.9 MHz, and the high-band, for frequencies above 480 MHz and up to 960 MHz. The frequency deviation and frequency offset settings can be also adjusted according to the requirements of the application. There is an automatic frequency control (ACF) setting also available, which, when enabled, results in optimal sensitivity and selectivity over a wide range of frequency offsets. The trade-off is that for frequency offsets above 80 kHz, the sensitivity of the receiver gets exponentially worse. The tradeoff is shown in **Figure 2:8**, with the data rate set to 100 kbps and the signal bandwidth at 208.4 kHz. The TX data rate is also configurable from the registers TX Data Rate 1/0 with supported data rates ranging from 123 bps to 256 kbps. These settings are best calculated from the calculator sheet provided by the manufacturer for optimal performance.

The RFM22B supports FSK, GFSK and OOK modulation options. GFSK provides the best performance and cleanest spectrum and as such is the recommended modulation type, while unmodulated carrier signals are also supported. A comparison between the FSK and the GFSK spectrums is shown in **Figure 2:9**, with the data rate set to 64 kbps. The type of modulation

can be set in the Modulation Mode Control 2 register. Modulation data can be obtained from three different sources, the FIFO mode, the Direct mode and the PN9 mode, with the FIFO mode being the most prominent. Direct mode is mainly aimed towards legacy devices that perform packet handling within an MCU or other baseband chip and bypasses the FIFOs entirely, a feature that was not used in the current work. In FIFO mode, the TX/RX data are stored in the integrated FIFO register memory, which can be accessed by burst-reading or burst-writing to the register 0x7F, named FIFO Access. The TX FIFO is used for pushing bytes to be transmitted, while the RX FIFO is used for reading bytes that have been received. Their size is identical and equal to 64 Bytes and the module supports three programmable threshold values that generate interrupts upon FIFO events. These are the TX FIFO Almost Full, TX FIFO Almost Empty and RX FIFO Almost Full thresholds and are measured in bytes. Both FIFOs can be cleared or reset from the corresponding bits in the Operating & Function Control 2 register.

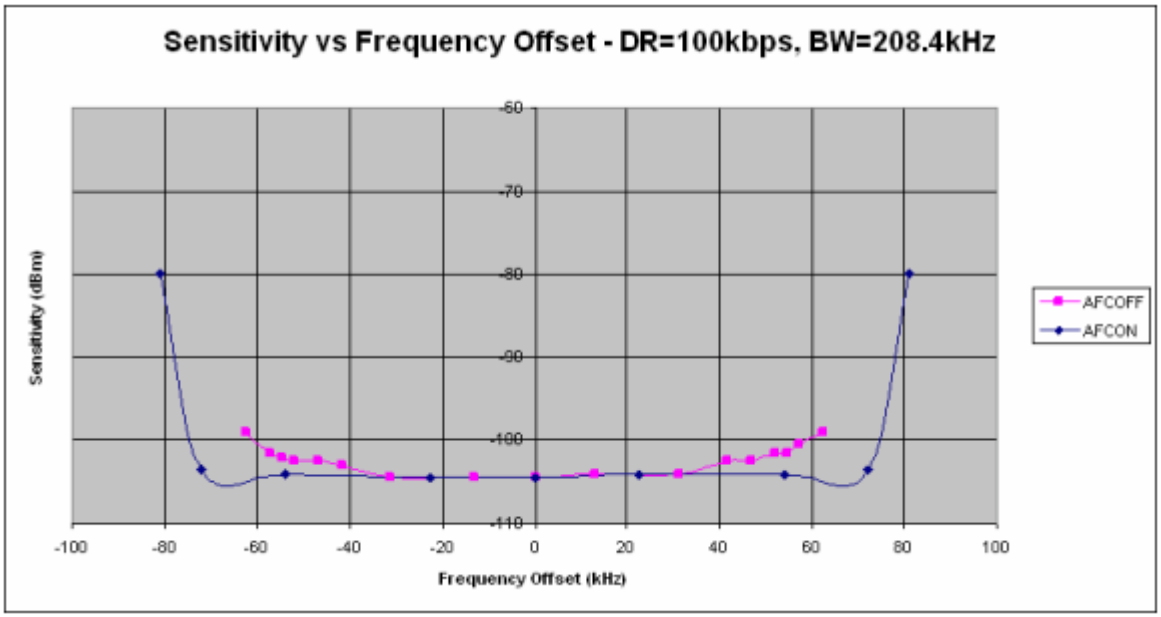

Figure 2.8: Sensitivity at 1% PER vs. Carrier Frequency Offset

When using the FIFOS, automatic packet handling may be enabled. During TX mode, the data bytes in the FIFO are packaged with other bytes of information to create the final packet to be transmitted. The information fields are Preamble (1-512 B), Sync words (1-4 B), Header (0-4 B) and the CRC checksum (either 0 B or 2 B) for error detection, all of which can be setup from the corresponding registers. If the Automatic Packet Handler (APH) is enabled, these

fields are required and filled automatically from the contents of the Packet Handler Registers. Also, when the packet length is not fixed, the APH will fill in the packet length field during a packet TX. Else, when the APH is disabled, the entire packet to be transmitted should be loaded into the TX FIFO. Similarly, if the APH is enabled during RX, only the data bytes are stored in the RX FIFO, while all the fields in the packet structure are required. Else, the modem still requires the preamble and sync word registers to be set, with all bits after the sync words being treated as raw data, allowing for creating custom packet handlers. The packet structure is shown in **Figure 2:10**.

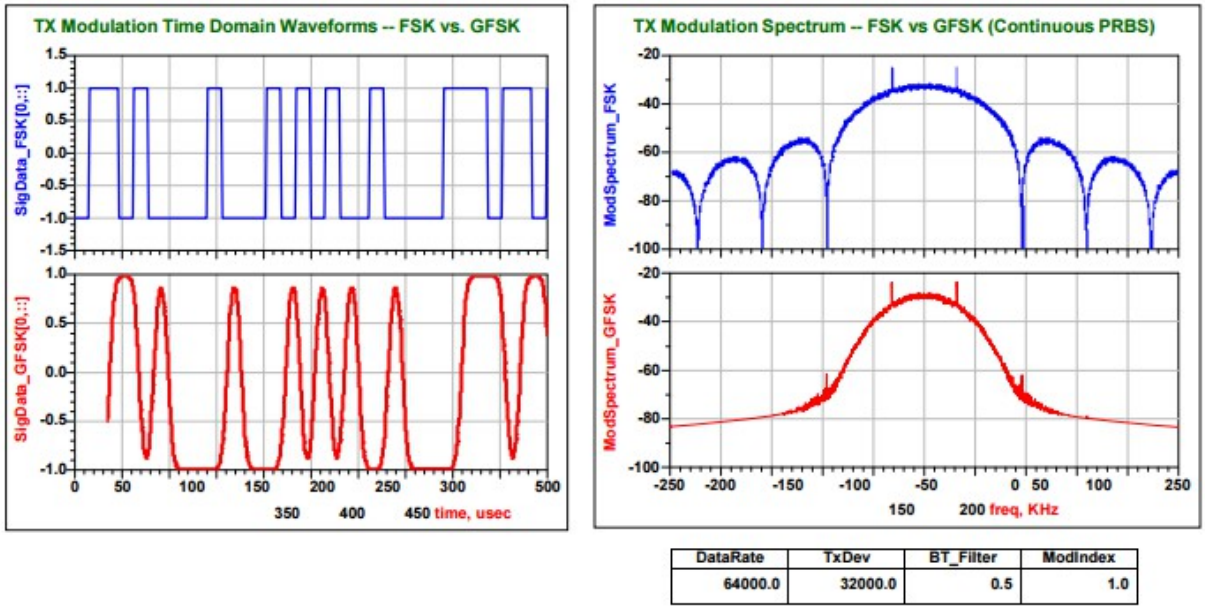

Figure 2.9: FSK vs GFSK Spectrum Comparison

Figure 2.10: General Packet Structure

The RFM22B also supports data whitening, Manchester encoding and CRC coding. Data whitening is used to avoid extended sequences of 0s and 1s, resulting in a more uniform

spectrum. Manchester encoding can be used to achieve better synchronization properties and dc-free transmission. Due to the nature of the encoding, while the effective datarate remains unchanged, the actual data rate is effectively doubled. CRC can be applied only to the data portion of the transmitted packet, or it can be set to include all of the header, packet length and data fields. A set of the most common CRC polynomials can be selected, such as CCIT, CRC16, IEC16 and BIACHEVA.

The module has integrated automatic preamble detection. As noted above, the preamble length can vary from 1-512 B and can be set from the registers 0x33 Header Control 2 and 0x34 Preamble Length. The preamble detection threshold can be set in the register 0x35 Preamble Detection Control 1 and is in units of 4 bits, also called nibbles. The preamble detector searches for valid preamble patterns with the set length. If a preamble is detected, valid or false, then the module will start searching for sync. In case the sync is not found, then the module will go back to preamble detection, after a small timeout. In the current work, the modulation used was GFSK with AFC enabled and the preamble length was set to 16 nibbles, or equivalently 8 B, while the preamble threshold was set to 6 nibbles, or 3 B, well above the specified limits stated in the RFM22B manual. The complete specifications with the recommended preamble length settings are shown in **Table 2.5**. Also, all four synchronization words (each of 1 B in length) were enabled to be transmitted during TX or expected during RX and hard-coded to each of the modules. As such, optimal performance was achieved.

Mode	Approximate Receiver Settling Time	Recommended Preamble Length, 8-bit Detection Threshold	Recommended Preamble Length, 20-bit Detection Threshold
(G)FSK AFC Disabled	1 B	20 bits	32 bits
(G)FSK AFC Enabled	2 B	28 bits	40 bits
OOK	2 B	3 B	4 B
(G)FSK AFC Disabled + Antenna Diversity Enabled	1 B	–	64 bits
(G)FSK AFC Enabled + Antenna Diversity Enabled	2 B	–	8 B
OOK + Antenna Diversity Enabled	8 B	–	8 B

Table 2.5: Minimum RX Settling Time - Preamble Length

The modem settings were calculated using the spreadsheet calculator provided by HopeRF. It is encouraged to use the calculator instead of calculating and setting each value manually in any application.

Finally, the module provides several auxiliary functions and secondary options. It comes with an integrated SMART RESET (or Power On Reset – POR) circuit, which was designed to produce a reliable reset signal under any circumstances. When powered on, the device is initialized with the default values for each register, a function that also is enabled whenever the input voltage drops below the reset limit threshold. There is also a crystal oscillator, which provides a 30 MHz signal that is internally divided and can be outputted to the MCU through the GPIO2 pin. This way, the cost of the module is reduced, since a single oscillator is used to produce a number of frequencies. Next, there is an integrated General Purpose ADC chip embedded, with 8-bit resolution. The inputs on the ADC can be any one of the GPIO signals or the temperature sensor, selected from the Input MUX, while the reference voltage can be obtained from either the VDD/3 signal, the VDD/2 signal or the VBG signal (1.2V) through the Ref MUX. The default setting is to read the temperature sensor voltage using the bandgap voltage (VBG), providing an LSB resolution of 4 mV. The resolution can be changed by changing the ADC reference voltage accordingly. The module can measure temperature through the on-board integrated analog temperature sensor. The sensor is enabled

automatically whenever the input of the ADC is set to the sensor, or the analog temp voltage is selected on the appropriate bus. It is digitized and read over the SPI through the register 0x11 ADC Value, with the value ranging according to the table 16 in the manual, which is shown in **Table 2.6**.

entoff bit	Tsrange[1:0]	Temp. range	Unit	Slope	ADC8 LSB
1	00	–64 … 64	°C	8 mV/°C	°C
1	01	–64 … 192	°C	4 mV/°C	°C
1	10	0 … 128	°C	8 mV/°C	0.5 °C
1	11	–40 … 216	°F	4 mV/°F	1 °F
0*	10	–64 … 64	°K	3 mV/°K	1.333 °K
***Note:** Absolute temperature mode, no temperature shift. This mode is only for test purposes.					

Table 2.6: Temperature Sensor Range

There is also a Low Battery Detector circuit integrated into the chip with a programmable Low Battery Detector Threshold. The chip is capable of generating interrupts when the digitized battery voltage drops below the threshold specified, while the battery voltage can also be read from the register 0x1B Battery Voltage Level at any time, when the LBD is enabled. Other available features are the wake-up timer, used to wake the device up from SLEEP mode and the Low Duty Cycle Mode, which is used to automatically wake-up the receiver from SLEEP to check the availability of a valid signal. Detection of a valid preamble or sync means extension of the period, so that the whole packet is received. There are also the three available GPIOs, which can be configured to perform a variety of tasks, receive signals from other MCUs, send signals to MCUs and even output clock periods. The module also supports antenna diversity through the GPIOs, to counteract the problem of frequency-selective fading, due to multi-path propagation of signals. Finally, the module also supports RSSI and clear channel assessment. Once a packet is received, the RSSI value of the link can be read from register 0x26 Received Signal Strength Indicator, while there is a programmable threshold for clear channel indication. After the RSSI value is evaluated in the preamble, a decision is made based on the level of the RSSI and the threshold set in the register above. The RSSI status bit will be set, if the RSSI value is above the threshold value. The RSSI is

plotted against the input power on the antenna of the receiver in **Figure** below.

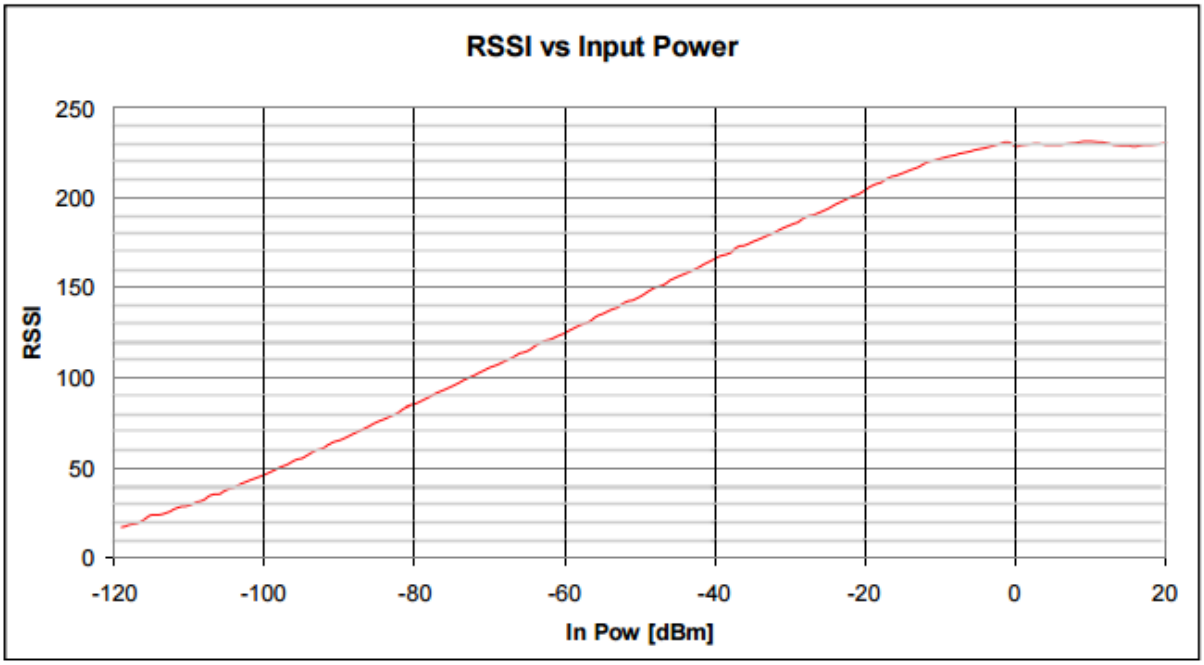

Figure 2.11: RSSI vs Input Power Graph

The above mentioned auxilliary functions are all important features that allow for designing custom complex communication networks and systems with little to no limitations and great adaptability.

There is a number of alternative devices and protocols when it comes to creating sensor networks and communicating with sensors and other devices in a wireless fashion. The main alternatives to the RFM22B are the devices in the XBee device family. The XBee modules are a family of embedded systems, providing reliable wireless end-point connectivity to devices. They use the IEEE 802.15.4 networking protocol, which enables peer-to-peer and fast point-to-multipoint communication. They use the ZigBee standard[42] of communication for the IoT and are mainly oriented towards devices and applications which require high-throughput, low-latency, and predictable communication timing. They are simple, easy to use, come in many different types and shapes, there are modules available for almost every frequency spectrum and have specialized features that allow the creation of complex automation systems. They are manufactured and provided mainly by the American company Digi International[42], which also standardized the mesh networking protocols that the XBee

modules implement. The main differences between the regular and pro versions of the XBee module family are that the pro versions cost more, use more power and offer longer range, and are slightly bulkier than their regular versions. Pro and regular versions can be used together and matched to create complex networks, allowing for connecting together localized networks, created using the regular versions, over long distances, using the pro versions to cover the extra distance.

One main problem with the XBee radios is their cost, which is substantially greater than the cost of the RFM22B module. One of the most popular XBee modules, the 2.4GHz XBee Series 1 (802.15.4) costs around 25$ from SparkFun's online website, while the pro version of the same module costs around 38$. Both operate at 3.3 V and draw 215 mA of current during normal mode of operation, which is substantially more than the current draw of the RFM22B. Their data rates are 250 kbps maximum, as with the RFM22B. The regular XBee has a 1 mW output (+0 dBm), while the pro has a 60 mW output (+18 dBm), both lower than the maximum power output of the RFM22B (+20 dBm). Finally, the XBee radios operate in the 2.4 GHz band, therefore they are susceptible to WiFi interference. The WiFi non-overlapping channels are plotted in the **Figure 2.4**, while the ZigBee protocol channels 11-22 are plotted in the **Figure 2.5**. It is evident that the ZigBee channels 11-22 use the exact same frequencies as the three non-overlapping WiFi channels 1, 6, and 11, while even the ZigBee channels 23-25 are susceptible to interference from the sideband lobes of the WiFi channel 11. While ZigBee channel 26 remains unaffected, many radios and devices do not support it.

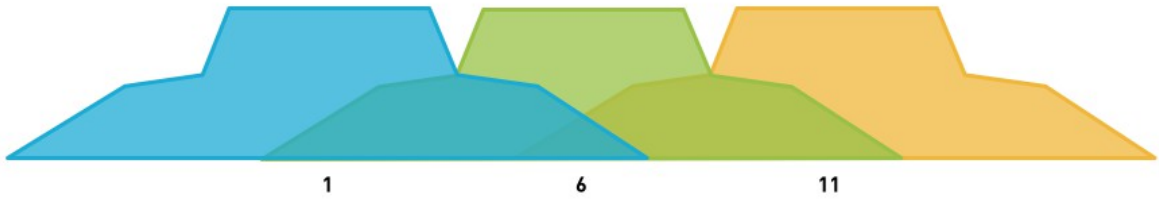

Figure 2.12: 2.4 GHz WiFi Channels

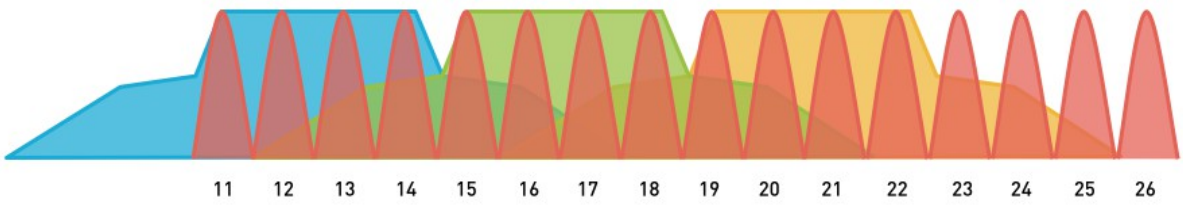

Figure 2.13: 2.4 GHz ZigBee Channels

Dealing with the problem of interference requires planning ahead which of the channels will be used for WiFi and which for ZigBee communication, a principle which is to be followed both for the Access Points and the ZigBee radios. Also, the radios are to be placed some distance from any actively transmitting WiFi device, such as routers, laptops, smartphones, or even TVs, in order to avoid the sideband lobe interference as much as possible, since sideband lobes are only visible very close to the actively transmitting devices. Finally, in apartment buildings that are densely populated, the existence of many different wireless networks complicates this even further, as the installation of the network will have to take in account all the visible networks around the apartment and all the channels used, making it extremely hard to set up a system that exists in a completely separate channel and is immune to interference, a problem which becomes more complex in the event of multiple smart home installations in nearby apartments of the same building.

When designing a new system that communicates wirelessly, it is imperative to secure the transmitted information, so that it is not visible to any unauthorized user. In that note, the ZigBee protocol uses the IEEE 802.15.4 standard, which is easy to view and capture from any smartphone or computer with a wireless network interface. Specifically, using software such as Wireshark, any user can capture the data sent between the access points and the XBee radios, dissect them and extract useful information, such as the presence of people in the home, and exploit it accordingly. At the same time, it is easy to locate the channel of communication used by the radio modules and create artificial noise to disrupt their communication, something that can be used to avoid triggering the alarm during a robbery. Finally, an experienced system hacker could possibly create artificial packets that simulate the packets used by the XBee modules and trigger false alarms, take control of devices and potentially harm the owners.

The RFM22B was the best solution to all the problems mentioned above. It operates outside the frequency range of the WiFi signal, making it immune to interference from household devices. It is not as easy to capture the communication packets, as the developer of the system has full control over the carrier frequency and its modulation, can enable the data whitening and Manchester encoding features of the radio and can even encrypt the information transmitted through the host MCU. Even if a malicious third party could view the raw data packets transmitted between the RFM22B modules, it would be almost impossible to decode the information, therefore making it a much safer option. The module has a higher output power maximum, providing for extended range, compared to the XBee radios and also has an on-board temperature sensor, eliminating the need of separate sensors and lowering the total cost. Finally, the RFM22B costs 5.95$, which is only a fraction of the XBee radio cost.

Chapter 3.

Preparations

This chapter revolves around the specific preparations that were made during the process of creating a fully working home automation system that provides security, allows extensive home management and energy saving. Having presented each of the hardware parts included in the present design, as well as their strong points and their conformity with the selection criteria, it is necessary to go into more detail, regarding their roles and the steps taken towards achieving these goals. A detailed analysis of the individual sub-systems and their specific role in the final system will follow.

The BBB serves primarily as the central Access Point (AP) of the system, providing an easy and simple way for the user to communicate with the home automation system. It also provides a secure storage for all the information regarding the present status of each connected sensor, module, and device throughout the controlled smart home. It also serves as the main web server of the system and it is where the necessary web pages are hosted. All of the developed websites have been designed while keeping in mind the maximum convenience to the user, therefore they provide easy access and user-friendly presentation of the current state of the system. The BBB is also responsible for receiving and handling all the HTML and POST requests from, but not limited to, any android device running the developed home control application program. The websites and the android application back-end are developed using a combination of HTML, JavaScript, and PHP scripts. The BBB is also where all the choices regarding the regulation of the inner house temperature and lightning are made, according to the user-set preferences. This is done from the developed software that is responsible for making smart decisions in order to minimize energy consumption from all the connected device, while regulating the living environment of the users inside the home. Finally, the BBB is running a MySQL server that stores information regarding the home layout, the current status of each door or window, the current power status of the lights and air condition, as well as the ambient light and temperature inside and outside the smart home and the motion in each of the connected rooms. All of the information is available to the user via the web pages, if the user prefers to use a computer, or via the developed android

application, which also allows for more control over the system variables, allowing the user to manually disable devices, not allowing the algorithm to turn them on or off, or to manually request from the system to turn on devices in certain areas of the smart home.

The AP is connected to a plethora of miscellaneous devices, the sensor/control modules (SCM). The developed SCMs are able to perform simple tasks, such as controlling devices inside the smart home using relays and sensing their environment. The APM is the board used to create all the SCMs responsible for controlling devices, sensing their environment and reporting events in the smart home. In order to allow the APM boards to perform their intended tasks, it was necessary to design specialized hardware PCB boards to connect all the electronic parts and components, as well as power the APM, the RFM22B and every other electronic component used. The schematics and the board layout files were created in CadSoft EAGLE using SparkFun's libraries[38]. The AP and the SCMs are connected using a star network topology and they communicate in a wireless fashion by utilizing the RFM22B radios. Each SCM is directly controlled by and communicates only with the AP, while SCM-to-SCM communication is not allowed. The different types of the SCM devices are described in more detail in **Chapter 4**.

3.1. Preparing the BeagleBone Black

On the BBB used during the implementation of the current work, the default installed distribution did not come with a working NTP installation and the time was not set automatically. This presented a major problem, because the temperature measurements in the MySQL database are identified by a timestamp field and every time the BBB would go offline, the time would reset to January 1st in 2000, creating problems with the consistency of the data stored in the database. In order to overcome this issue, the NTP software needed to be installed on the BBB. The first step was finding a NTP server close to the physical location of the BBB and update the "*/etc/ntp.conf*" file accordingly. In recreating the current work, it is important to make sure the "server" and "fudge" lines in the "*ntp.conf*" file are commented out, in order to prevent the server from synchronizing back to itself. Use of the root NTP servers is discouraged, as they are heavily loaded. Connecting to an NTP pool server in close proximity to the physical location of the installation helps the NTP servers with load

balancing, while also providing better response times and stability. Next, it was required to set the "*/etc/localtime*" file according to the timezone of the chosen location. This is done by creating a symbolic link of the selected timezone file named "*localtime*" from the directory "*/usr/share/zoneinfo*" to the "*/etc/*" directory. The final step is to start the NTP services and perform a reboot. It is also recommended to fix the hardware clock of the BBB by modifying the "*ntpdate.service*" file accordingly, because in some cases the RTC Time may still be off. Further information regarding the clock and NTP services on the BBB can be found on Derek Molloy's website[14].

The Device Tree (DT) and the Device Tree Overlay (DTO) are a way to describe hardware in a system and are used extensively in the BBB. One main issue of the DT was that it was not designed for embedded systems, which required to modify the system during run-time. A collaboration of scientific personnel led to the development of a system that would allow the modification of the DT from user-space during run-time. There is a collection of existing overlays available under the directory "*/lib/firmware*" on the BBB. For SPI communication and enabling the SPIDEV, which is the SPI kernel driver appearing as a device in user-space, it was necessary to export and enable the SPI0 and SPI1 DTOs. It is important to note that the HDMI interface has to be disabled for the SPI1 interface to be used, since the HDMI overlay is using the same GPIOs as the SPI1. Disabling the HDMI interface is recommended, as the board will have lower power consumption and allow for more GPIOs to be used, but with doing so, the user will have no video output from the BBB and will be limited to interaction through the SSH protocol.

Having enabled the SPI device and having loaded the SPI0, the RFM22B radio can communicate with the BBB. The pin connections between the BBB and the RFM22B are presented in **Table 3.1.** The RFM22B nIRQ pin is the interrupt signal output pin and can be connected to any (*) available GPIO pin configured as a digital input on the RFM22B.

BBB pins		RFM22B-S2 pins
SPI0_CS0	P9_17	NSEL
SPI0_D1	P9_18	SDI
SPI0_D0	P9_21	SDO
SPI0_SCLK	P9_22	SCK
*	*	nIRQ

Table 3.1: BeagleBone Black SPI to RFM22B Pin Connections

Storing and presenting information to the user is achieved by turning the BBB into a web server[10], using MySQL to store the information that is later offered to the users. MySQL is a very popular and widely used open-source relational database management system that is robust and easy to use. In the developed home automation system it stores all the information provided by each SCM device. This information can later be retrieved by a web page and presented to the user. The Angstrom distribution does not come with MySQL installed by default, therefore it was necessary to download and install the MySQL5 package through "*opkg*", the default Angstrom package management software. After the installation, there is a series of steps to be taken in order to make MySQL fully working without errors. The first step is to comment the line 3 in the following script "*/etc/init.d/mysqld*". This is done to eliminate an error that arises from the "*/etc/default/rcS*" file, which is missing from the Angstrom file system, preventing the MySQL service from starting. At this point, while MySQL can start and the user can log in successfully, it is very important to configure MySQL properly to prevent a system error when the BBB restarts and the user attempts to log in again. This error, if allowed to occur, requires the re-installation of a new image of the OS and having to setup the BBB from the start. The error is easily preventable by deleting some unnecessary links in the "*/etc/*" system folder, effectively preventing MySQL from starting at boot. Setting up the MySQL service to start at boot requires creating a new service file. Once the "*/lib/systemd/system/mysql.service*" is created, the MySQL service can be enabled to start at boot and then it can be successfully started. The final step is to download the MySQL C programming Client Library. The library allows the creation of C programs that can directly communicate with the MySQL database.

After the MySQL server is configured, the next step involves installing the actual web server

software on the BBB. The technologies used in the proposed system are a lightweight open-source web server, named Lighttpd, and the PHP scripting language. Lighttpd is preferred over the widely used Apache web server software mainly because of the limitations of the available hardware. Firstly, it is required to install the Lighttpd package using the "*opkg*" packet manager, as before. This will report the job failing, which is because port 80 is already in use from the preloaded services on the BBB, responsible for the Cloud 9 IDE and the unique bonescript scripting language. After this is done and the BBB reboots, the web server service should start automatically. The next step is the installation of PHP, then Lighttpd must be configured to use PHP by editing the Lighttph *config* file, located in the "*/etc/*" directory. Specifically, the "*/etc/lighttpd.conf*" file must be edited so that fast-cgi mode is enabled and the bin-path is pointing to the installed php-cgi location. To complete the configuration, the Lighttpd service needs to be restarted, so that the configuration file is re-read. The location of the web pages the web server is serving is under the "*/www/pages*" directory.

Users can connect to the BBB over SSH. By default, the username field for connecting to the BBB is "root" and the password field is left blank, making it easy for malicious users to gain unauthorized access to the system. Therefore an equally important step was to secure the BBB with different user accounts without administration privileges and secure them with strong passwords. Unauthorized access to the system was therefore made harder, while even if an intruder managed to gain system access, the privileges of the user would safeguard any important system features, creating a second layer of security.

Finally, the BBB required the necessary software for controlling the SCM devices remotely and handling all the user requests. In order to allow the BBB to perform these tasks, custom libraries had to be written to create an appropriate interface and simplify the coding of the programs. Apart from the libraries, three main programs were written. The first is a program that allows synchronization of SCM devices and serves as the means of synchronization of the system. It is responsible for creating a general layout of all the rooms and SCM devices installed in the smart home, separated by the appropriate room identifier, which is set by the user at run-time. Next is the main control process, which is run when the synchronization is over and the home layout is complete. It is the process which makes all the smart decisions in the home and informs the user over email on certain events. Finally, a socket server was

created as a separate thread in the main control process, which is used to listen and process all the requests coming from the user, more specifically the requests that regard the turning on or off a certain relay. All software libraries and programs were coded using the C programming language. More details regarding the software development process can be found in **Chapter 7**.

3.2. Preparing the Arduinos

The APM serves as the master MCU in each of the SCMs. It allows for SPI communication over the pins 10 to 13, as presented in **Figure 2.3**. To communicate with the RFM22B radio, the APM required a C++ library to allow controlling the radio from the APM. There is an SPI software library available in the current Arduino version, which served as the basis of the produced RFM22B library, which is the software required for fully controlling and programming the RFM22B radio from the APM. The pin connections between the APM and the RFM22B are presented in **Table 3.2.** The RFM22B nIRQ pin is the interrupt signal output pin and can be connected to *any* (shown using the "*" character) available Digital pins, configured as an Input, on the APM.

APM pins		**RFM22B-S2 pins**
10	SS	NSEL
11	MOSI	SDI
12	MISO	SDO
13	SCK	SCK
*	*	nIRQ

Table 3.2: Arduino Pro Mini SPI to RFM22B Pin Connections

The Arduino IDE is the software used to program all Arduino devices. In order to connect the APM with a PC, an FTDI chip is required, because it lacks an on-board USB port. In the current work, a FTDI to USB board with the FT232R chip was used. Programming was done in C++ language, which is native to the Arduino IDE. The main libraries written for the BBB, as described in the previous section, were used and altered accordingly, in order to accommodate the specific needs of the Arduino and the SPI communication. The developed software for each SCM device is presented in greater detail in **Chapter 7**.

3.3. Preparing the RFM22B

In order to select the appropriate values for the registers[37] of the RFM22B-S2, the provided calculator sheet was used. The radios are tuned to the 868 MHz band, which is primarily used by burglar alarm systems, thermostats and other household security/control devices. Low-power radios in the 868 MHz band are exempt from licensing, as specified by the EU regulations[25], but there exists a limitation that requires all applications in the 868 MHz to 868.6 MHz frequency band to meet a duty cycle of 1%. This roughly translates to a 36 second transmission time for each transmitter per hour, with the average transmission time of the RFM22B ranging between 25-35 ms and is more than enough for the needs of the system. One alternative is the 869.7 MHz to 870 MHz frequency band, where the only limitation is the +7 dBm absolute maximum transmit power in terms of the effective radiated power, without any limitations in the duty cycle or channel bandwidth. This frequency band is mostly suited for small installations that have little interference and less signal attenuation, whereas the 868 MHz to 868.6 MHz frequency band is mostly suited for larger installations, with a greater number of installed devices and greater distances in between the AP and the SCM devices, as it has a +14 dBm absolute maximum transmit power in terms of the effective radiated power and no channel bandwidth limitations.

The RFM22B chip supports OOK, FSK and GFSK modulation types. The cleanest modulation spectrum and best performance is achieved through GFSK, which is the recommended modulation type according to the datasheet[35]. The data rate setting in the RFM22B radio was set to 64 kbps. According to occupied bandwidth formula shown below, given the GFSK modulation with a 64 kbps data rate and the default setting of 50 kHz deviation, the occupied bandwidth of the transmitted signal becomes 164 kHz. This is well above the 25 kHz channel bandwidth limitations of the EU regulations[25] for the 869.3 MHz to 869.65 MHz frequency bands, therefore these are not suitable for the current application. Dropping the data rate far below the 64 kbps setting while reducing the default deviation setting can be done, if the need to operate in the frequency bands, as mentioned above, is presented.

$$OBW = symbol\ rate + 2 \times (outer)\ deviation$$

Figure 3.1: Approximate Occupied Bandwidth of a Digital Frequency Modulated Signal

The RFM22B is capable of producing an interrupt signal on certain events, by driving the nIRQ pin LOW. The interrupts must be enabled in the Interrupt Enable 1 and 2 registers in order for this to happen. When the nIRQ signal goes LOW, the MCU must read the Interrupt Status register containing the active Interrupt Status bit for the signal to be reset. Any interrupt status bit can be read individually at any time from the Interrupt Status 1 and 2 registers, without requiring the according interrupt signals to be enabled. In the current work, the nIRQ pin produces an interrupt whenever a new, valid packet is detected. Therefore, the *"enpkvalid"* bit was set in the Interrupt Enable 1 register.

Finally, in order to have access to the pins of the RFM22B radio chipset, a breakout board was designed using CadSoft Eagle. The breakout board aims to allow for easier access to the pins of the radio and make it compatible with 2.54 mm headers. A breakout board template was available in SparkFun's website, which was adjusted slightly. The final EAGLE schematic and the corresponding PCB layout are presented in **Figure 3.2** and **Figure 3.3** respectively.

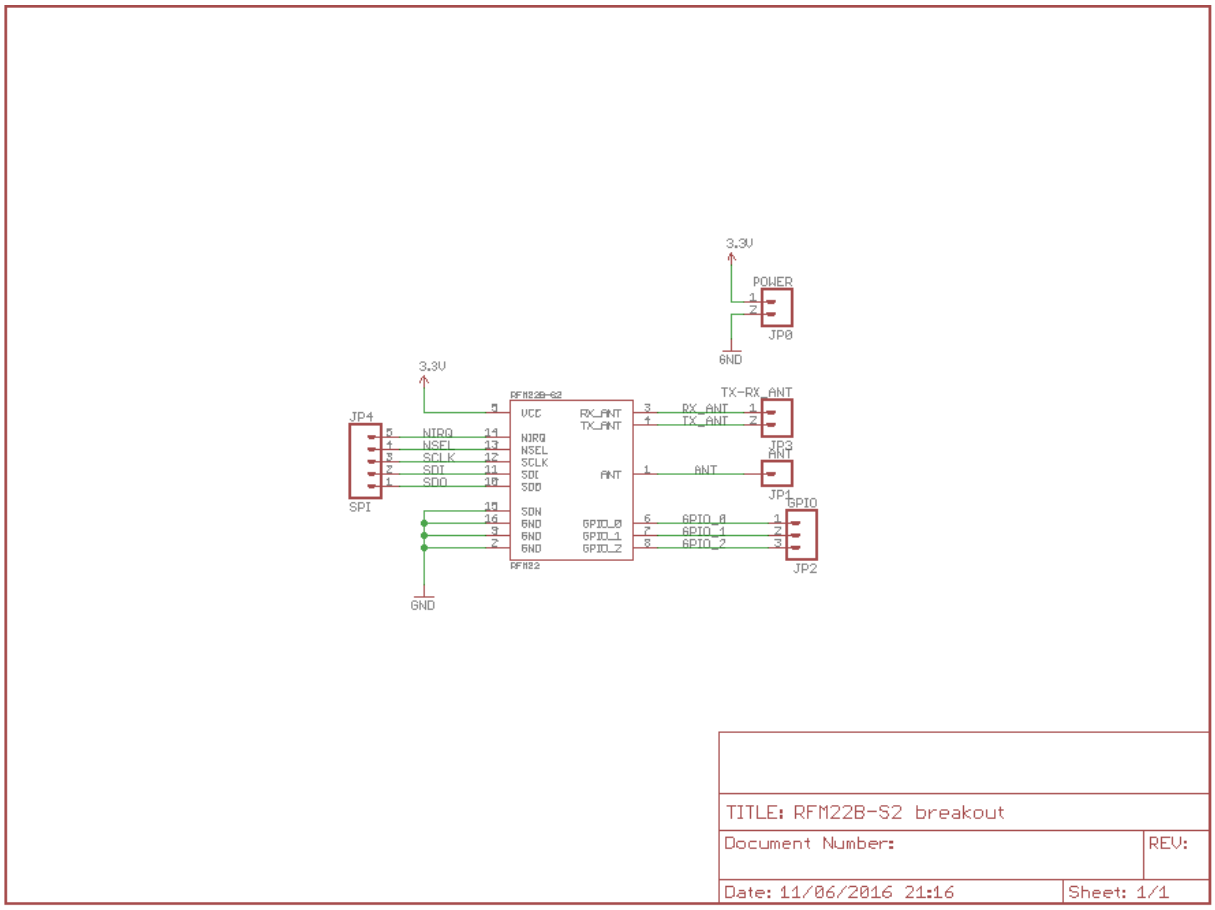

Figure 3.2: RFM22B Breakout Board EAGLE Schematic

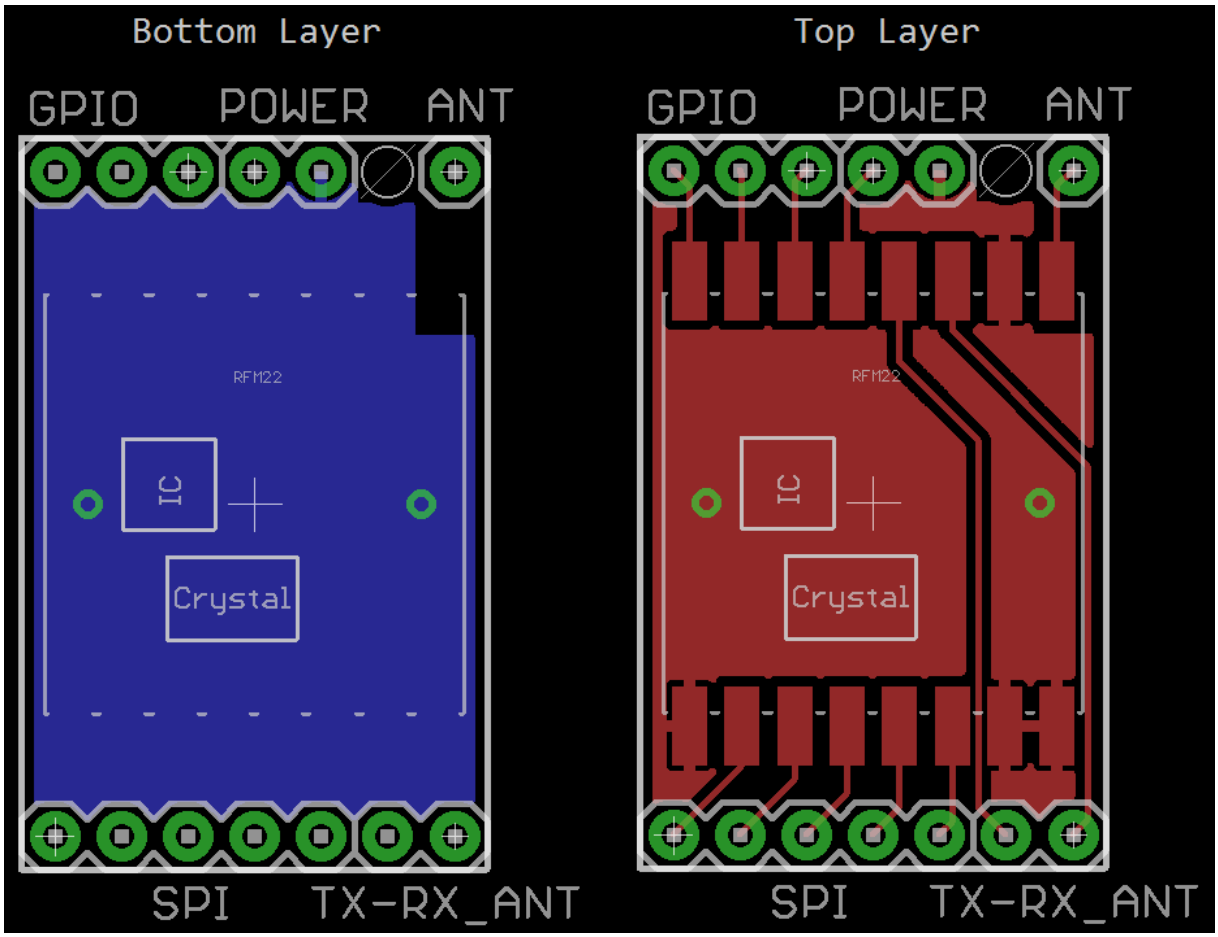

Figure 3.3: RFM22B Breakout Board EAGLE PCB Layout

Chapter 4.

Module Types

There are three main SCM types. Every SCM is composed of an APM board and a RFM22B radio, and separate electronics and boards to achieve their specific purpose. The power is provided from a 5 V wall outlet and regulated down to 3.3 V through a low-dropout linear regulator. The regulator used is the LD1117AV33[27], which has a low dropout voltage of 1.15 V typically and output current up to 1 A, more than enough to power the APM, the RFM22B and the miscellaneous electronics and sensors for each SCM type.

The cost of SCM was the most important factor in the implementation of the system. Each SCM device adds to the total cost, which can be increased considerably when talking about big houses. The basic cost of each SCM consists of the basis of electronics, such as the RFM22B radio, the APM and the circuit of the voltage regulator. The APM were bought online at a fraction of the price stated in **Chapter 2**, whereas the voltage regulator circuit consisted from two capacitors, a 100 nF ceramic input capacitor and an 10 μF electrolytic output decoupling capacitor, and the regulator. The RFM22B were procured in the regular price, stated in **Chapter 2**. A breakdown of the necessary materials cost is shown in **Table 4.1**. The cost of all the modules is increased further, based on the cost of the parts used in the PCB design. This cost is shown separately for each device type in a separate table under each subsection of the current chapter.

Part Name	Price
Enhancement V2 Pro Mini 328 16Mhz 3.3V	€2.50
RFM22B 868Mhz	€5.80
LD1117AV33 Voltage Regulator	€0.77
100 nF Ceramic Capacitor (Ebay pricing)	< €0.01
10 μF Electrolytic Capacitor (Ebay pricing)	€0.21
TOTAL	**€9.28**

Table 4.1:SCM Device Basic Cost

4.1. Reed

Reed SCMs have a Reed magnetic switch[15], which is a proximity switch magnetic sensor. They are further sub-categorized into door and window devices, a necessary categorization in order to achieve better control and distinguish information accordingly. These magnetic sensors are used to detect changes in the current state of doors and windows. The Reed switch used, is a Normally-Closed type switch, meaning that the switch is closed whenever the reed is away from the magnet. When the magnet, which is usually installed in the moving part, is moved further than approximately 20 mm from the magnetic sensor, the contacts of the reed switch are closed and current is flowing, informing the SCM of the change in the state.

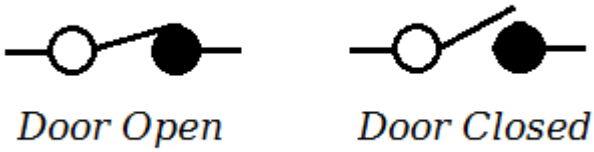

Figure 4.1: Normally Closed Switch Operation

The magnetic sensor is powered from a 3.3 V power source and is also connected to a pull-down resistor with a value of 1 kΩ, in order to form a voltage divider circuit. This way, the output of the voltage divider is a a digital 0 (also 0 V, GND plane) when the contacts of the sensor are open, meaning a magnetic field is in close proximity, and a digital 1 (also 3.3 V, POWER plane) when the contacts of the sensor are closed with the absence of a magnetic field. The APM senses these changes by constantly reading the output voltage of the voltage divider and comparing the previous value with the current. The complete circuit schematic for the designed PCB is shown in **Figure 4.2** and the corresponding PCB layout in **Figure 4.3.** The pricing of each part used is shown in **Table 4.2**.

Part Name	Price
Magnetic Reed Sensor (Ebay pricing)	€1.30
TOTAL	**€10.58**

Table 4.2: Reed SCM Device Cost

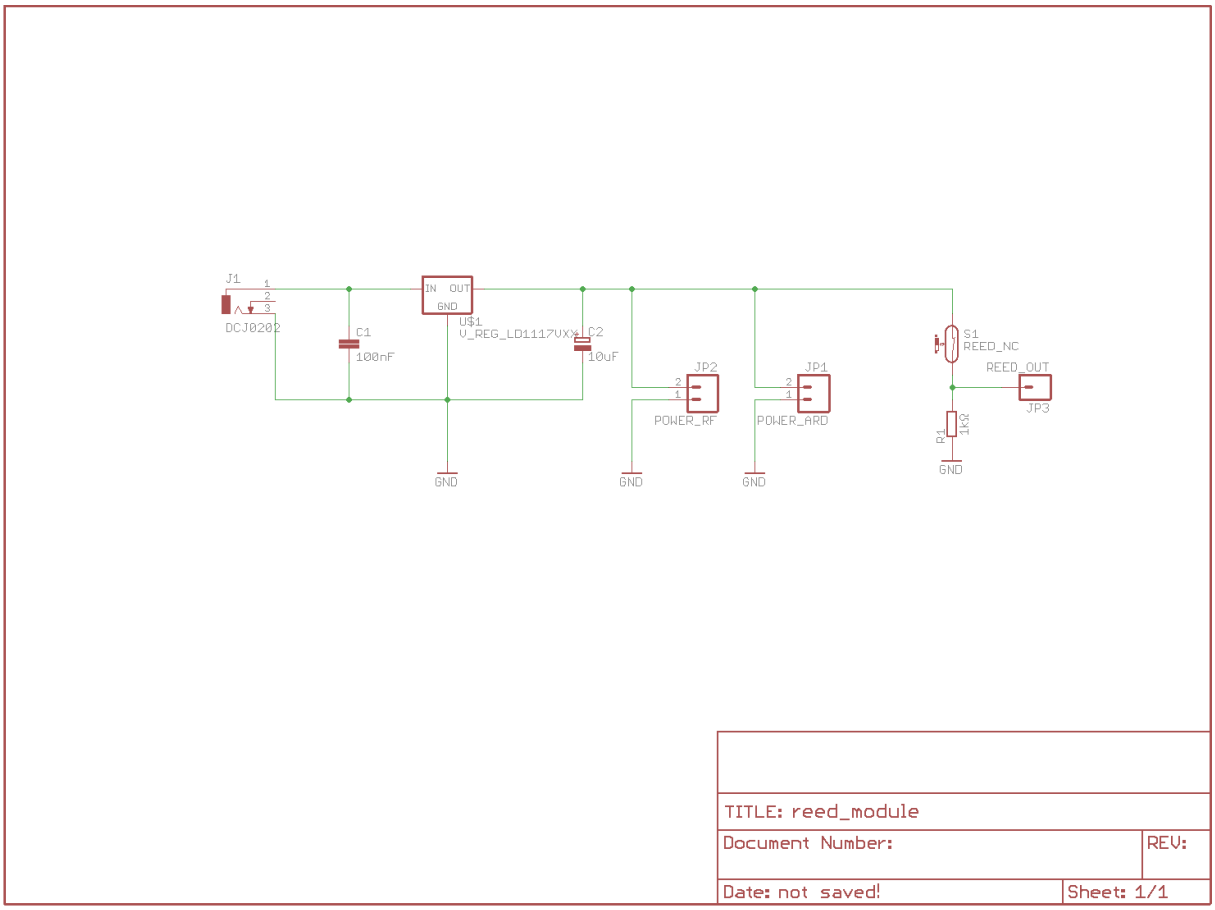

Figure 4.2: Reed PCB Schematic

The board layout of the PCB designed for the Reed SCM device is shown in **Figure 4.3**.

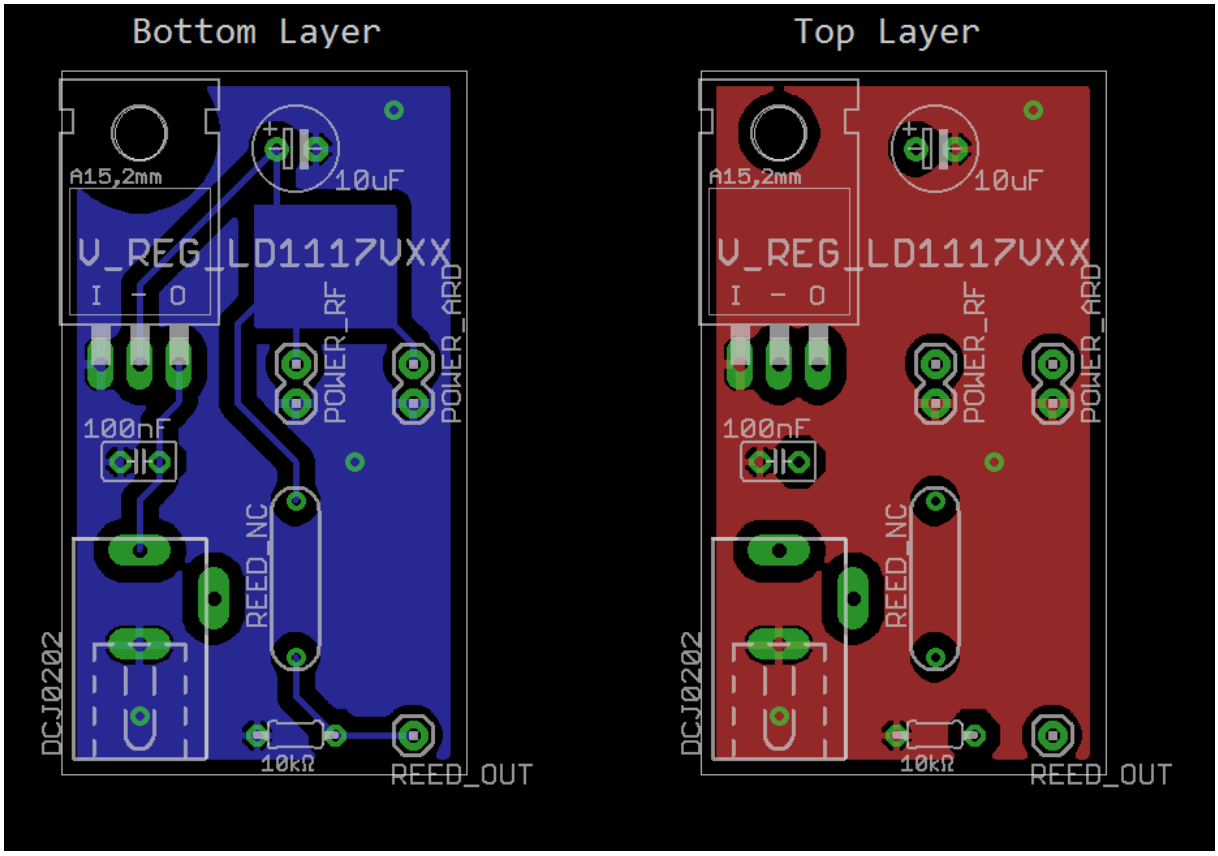

Figure 4.3: Reed PCB Board Layout

4.2. Relay

Relay SCMs are control type devices. They receive requests from the BBB AP and turn on/off their relays, controlling AC devices. The difference is that the relay modules used in the current work were solid-state relays that operate in the range from 5 V to 7.5 V, which was a problem for the APM board because it is operating at 3.3 V. In order to overcome this, a bi-level converter chip was used, to translate the 3.3 V output signals of the APM to 5 V input signals for the relay module. The relay SCMs are further sub-categorized into lights and air conditioning devices. These are necessary for providing the most basic home automation services, but many more types can be created using the same technology to allow for controlling motors, creating many opportunities for a completely automated home. The main difference between the lights and the air conditioning devices is that the solid state relay used for the lights is a single-channel isolated relay module, whereas in the air conditioning device, the relay used is a double-channel isolated relay module. The reason for this

difference is that in the lights scenario, the relay is used to turn them on/off, whereas in the air conditioning scenario, the first channel allows to control the power to the device and the second channel allows to control the hot/cold setting. The complete circuit schematic for the designed relay PCB is shown in the **Figure 4.4** and the pricing in **Table 4.3**.

Part Name	Price
2 Channel Relay Module, 5 V	€3.12
Single Channel Relay Module, 5 V	€1.60
TOTAL	**€12.4 / €10.88**

Table 4.3: Relay SCM Device Cost

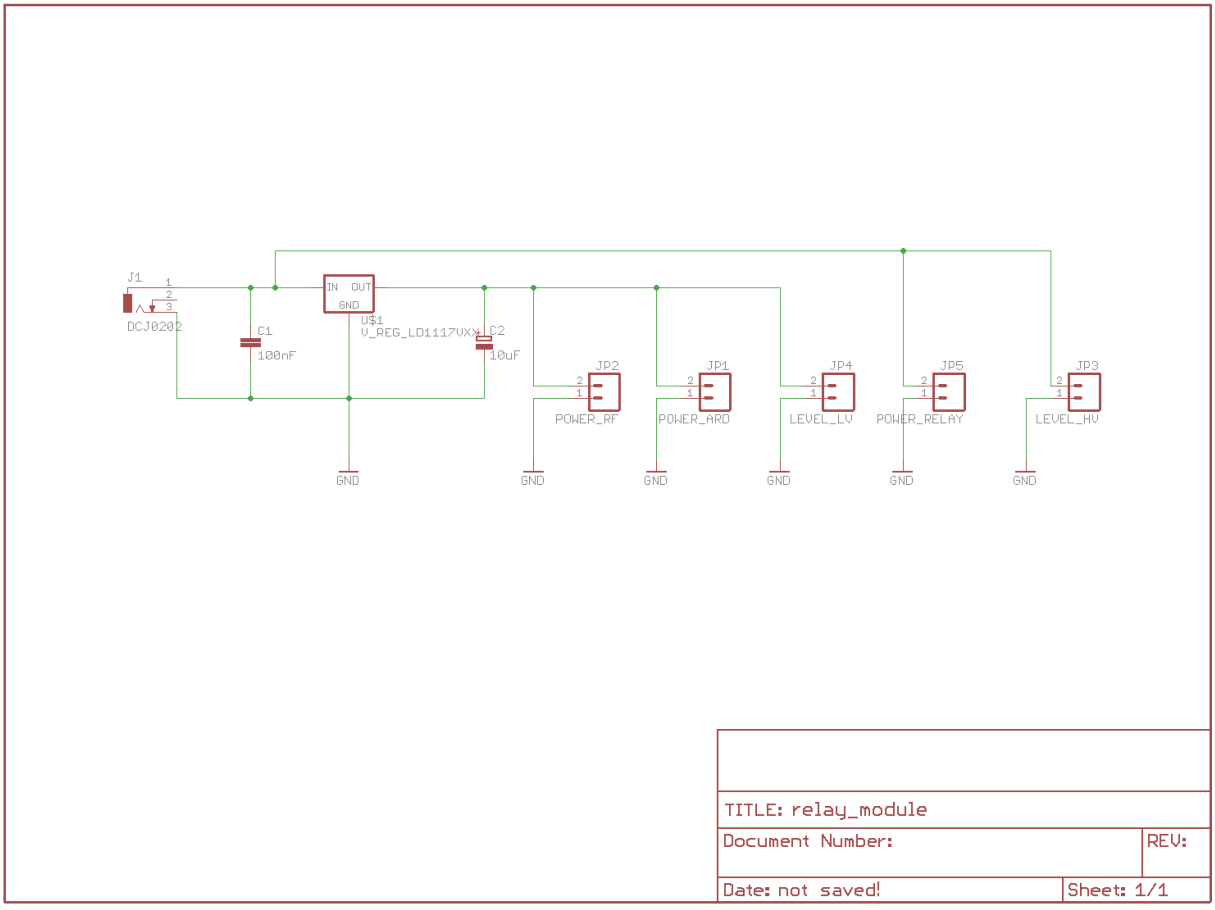

Figure 4.4: Relay PCB Schematic

The board layout of the PCB designed for the Relay SCM device is shown in **Figure 4.5**.

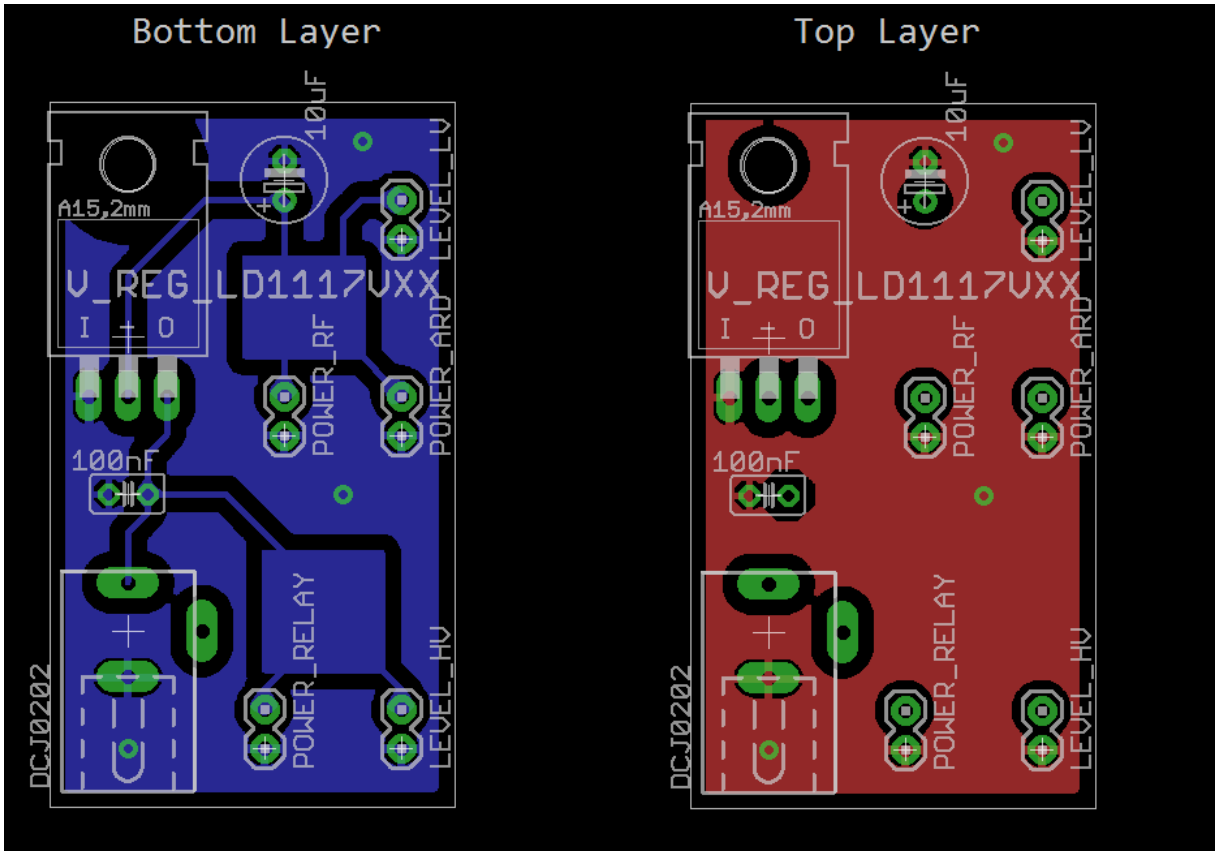

Figure 4.5: Relay PCB Board Layout

4.3. Sensor

The final type of SCM is the sensor device. These are the devices that sense their environment and provide the AP with information, allowing it to help the user make better choices and minimize the overall power consumption of the smart home. They are further sub-categorized in two categories, the photo SCM and the outside SCM. Their main difference lies in the sensing capabilities of each device, with the photo device being able to measure temperature, ambient lightning and also detect motion inside the room using a low-cost PIR motion sensor[32], while the outside device cannot detect motion and it is to be placed outside the home, providing information about the exterior conditions. Word of warning, the outside device should be protected from environmental hazards, such as rain.

These devices have separate PCB designs. This is due to the necessity of a 5 V power supply required by the PIR motion sensor installed on the photo device. Otherwise, both schematics are similar and are presented in the following figures. The PIR sensor is an

electronic device that measures infrared light, which is radiating from every object above absolute zero temperature (0 K), in its field of view[36]. They are low-cost solutions, being used for decades in motion detection and alarm systems. For measuring ambient light , a simple photocell (light-dependent resistor) was installed in both devices. Reading analog voltage from a photocell is achieved by connecting one end of the cell to the power node and the other to a pull-down resistor to ground, as shown in **Figure 4.4**. The connection point is where the analog voltage is measured.

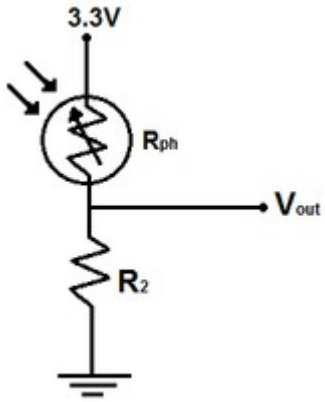

Figure 4.6: Photocell Circuit Connection

The resistance of the photocell against the output analog voltage is plotted in **Figure 4.4** using MATLAB. The simulation was performed for three common pull-down resistor values, those of 10 kΩ, 20 kΩ and 40 kΩ, while the input voltage was set to 3.3 V. Smaller pull-down resistor values increased the sensitivity of the voltage divider circuit, as smaller changes in the resistance of the photocell resulted in greater changes in the output voltage. The final selected value used on the sensor-type SCM devices was the medium value of 20 kΩ. By selecting this value, the photocell circuit is not extremely sensitive, while it remains sensitive enough to detect the dimming of the ambient lightning in a way similar to the human eye, resulting in more accurate indications of whether artificial lightning is required or not.

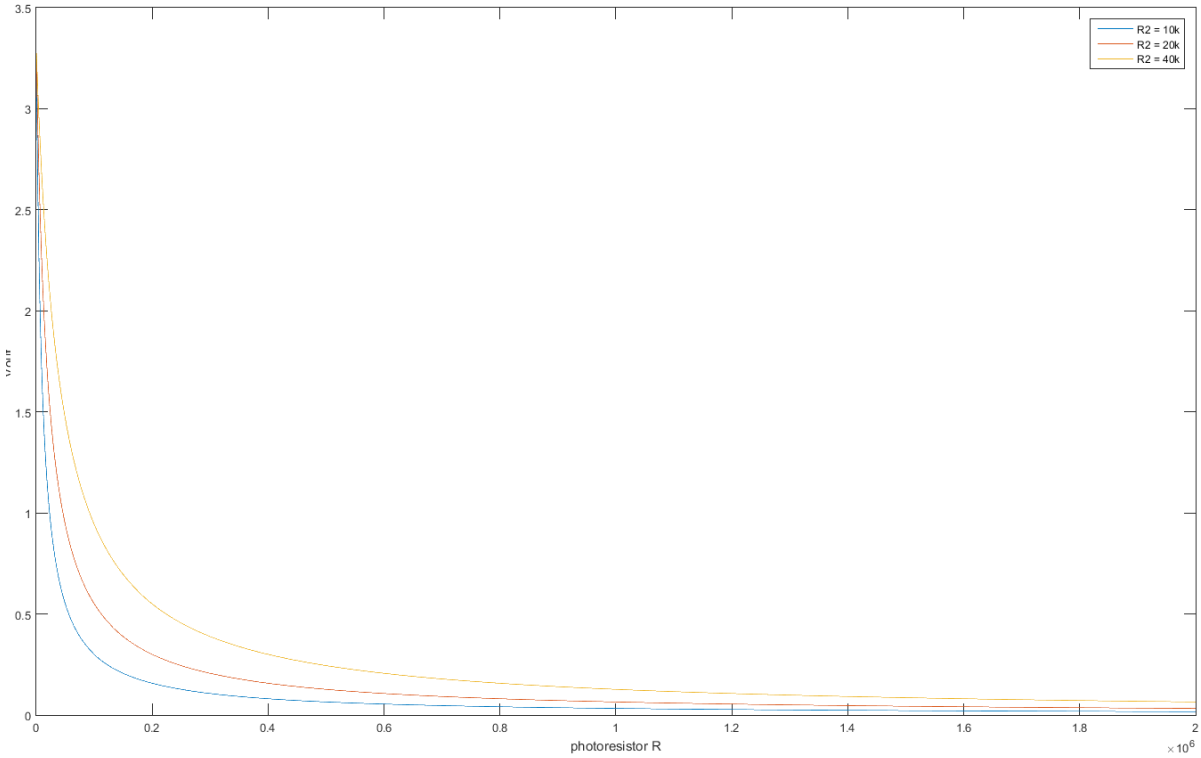

Figure 4.7: Photocell Resistance - Output Voltage Figure

The complete circuit schematic of the SCM Photo device is shown in **Figure 4.6**, while the circuit schematic of the SCM Outside device is shown in **Figure 4.7**. The Photo PCB layout is shown in **Figure 4.8**, while the Outside PCB layout is shown in **Figure 4.9**. The pricing details for the extra electronic parts used for both the Photo and the Outside devices is shown in **Table 4.**.

Part Name	Price
Photo Resistor LDR 5mm	€0.32
20 kΩ Resistor	< €0.01
PIR Sensor Module	€2.40
TOTAL	**€9.60 / €12.00**

Table 4.4: Sensor SCM Device Cost

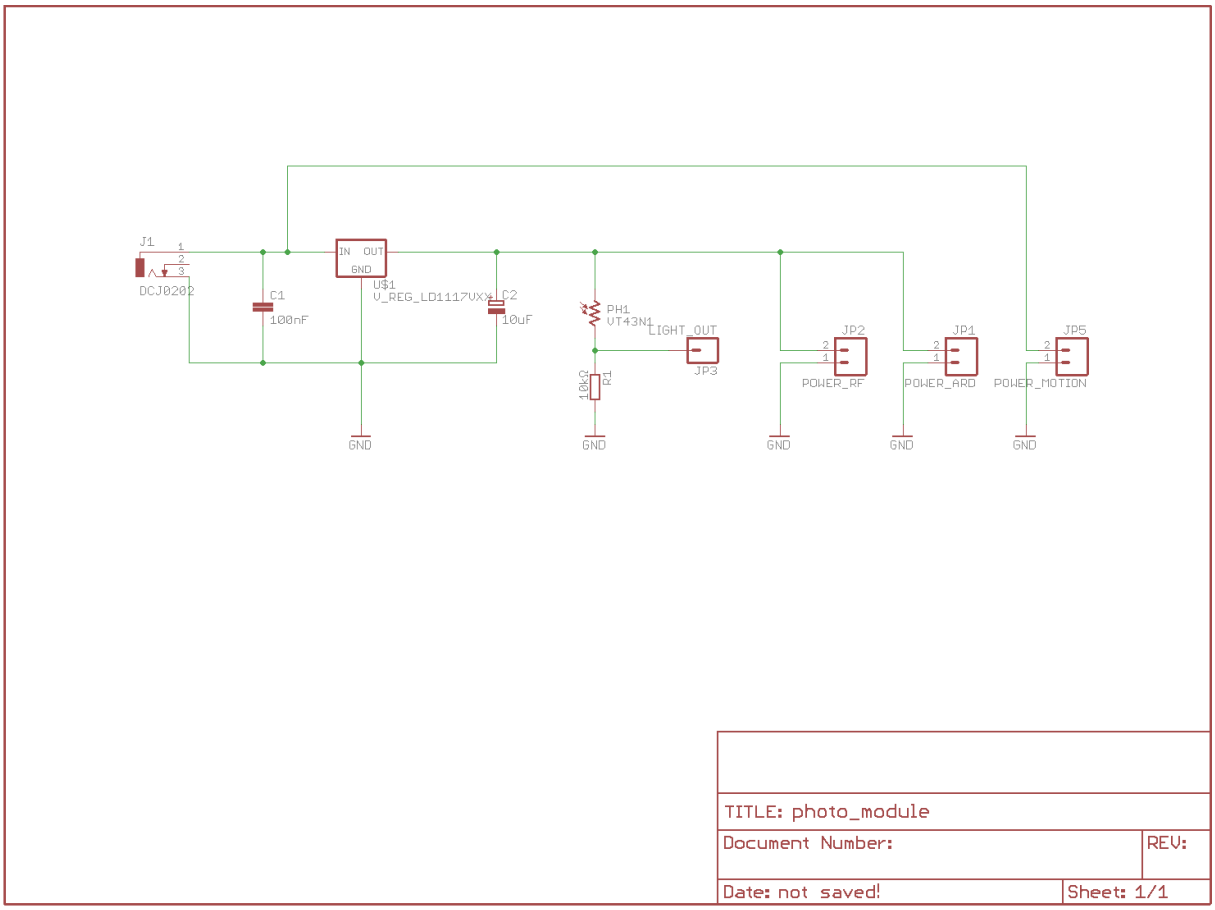

Figure 4.8: Photo Sensor PCB Schematic

The board layouts for both SCM devices, designed using CadSoft EAGLE, are shown in **Figure 4.7** and in **Figure4.8** respectively.

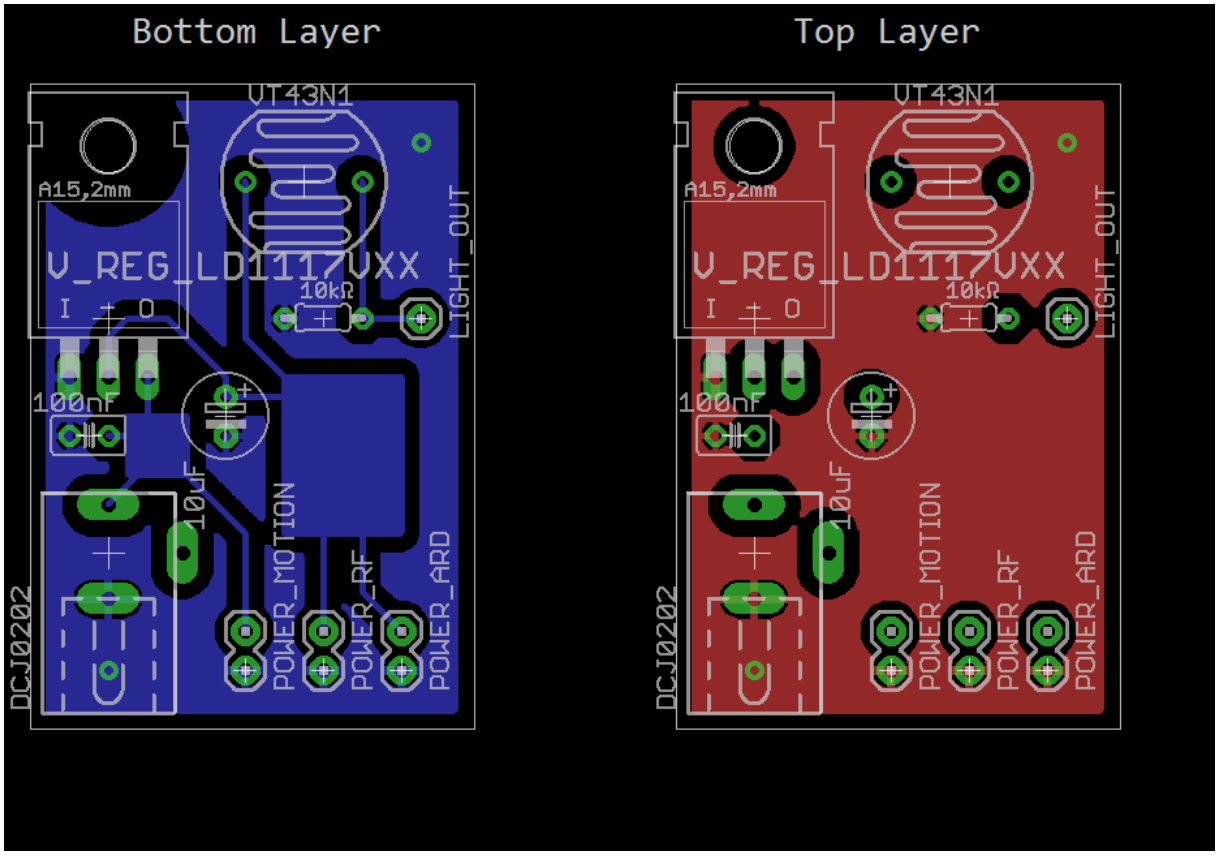

Figure 4.9: Photo Sensor PCB Board Layout

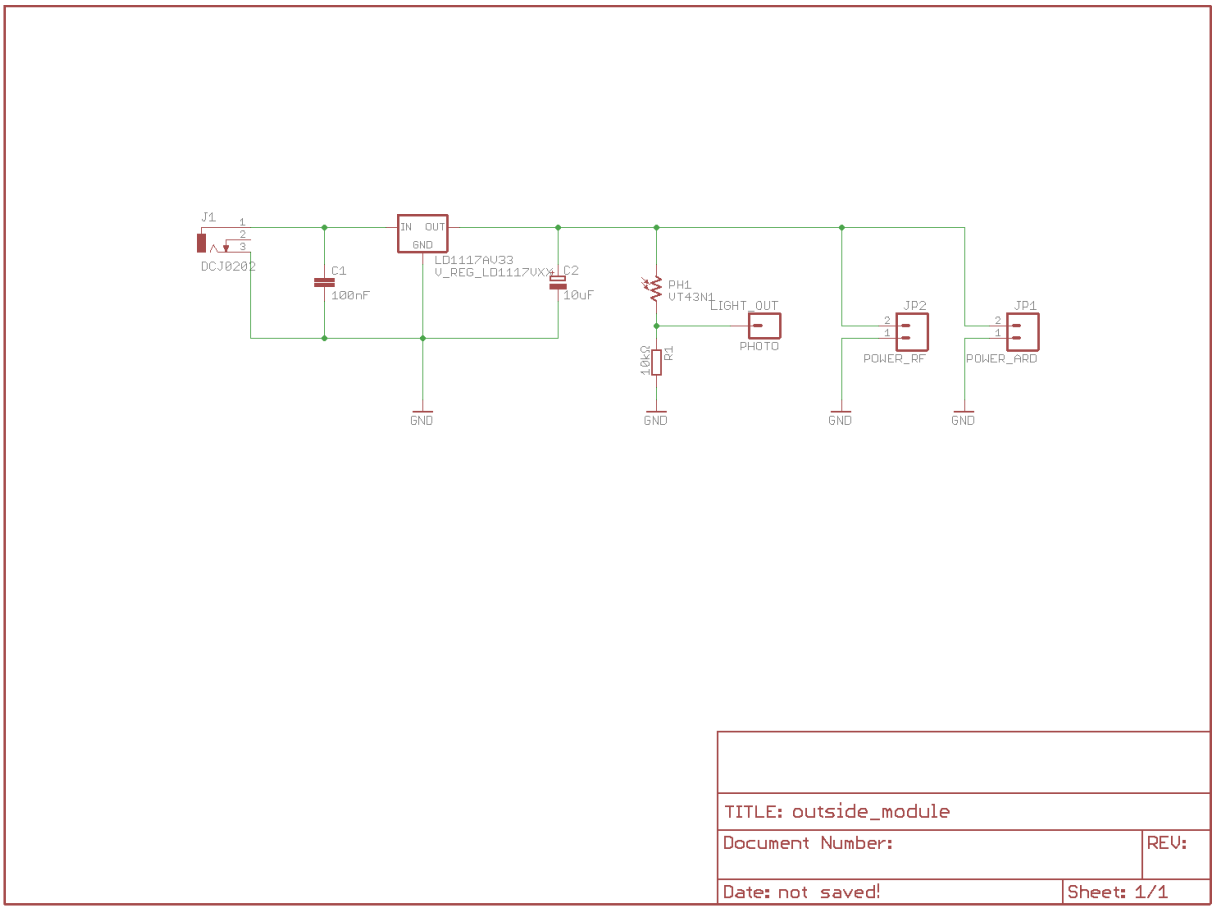

Figure 4.10: Outside Sensor PCB Schematic

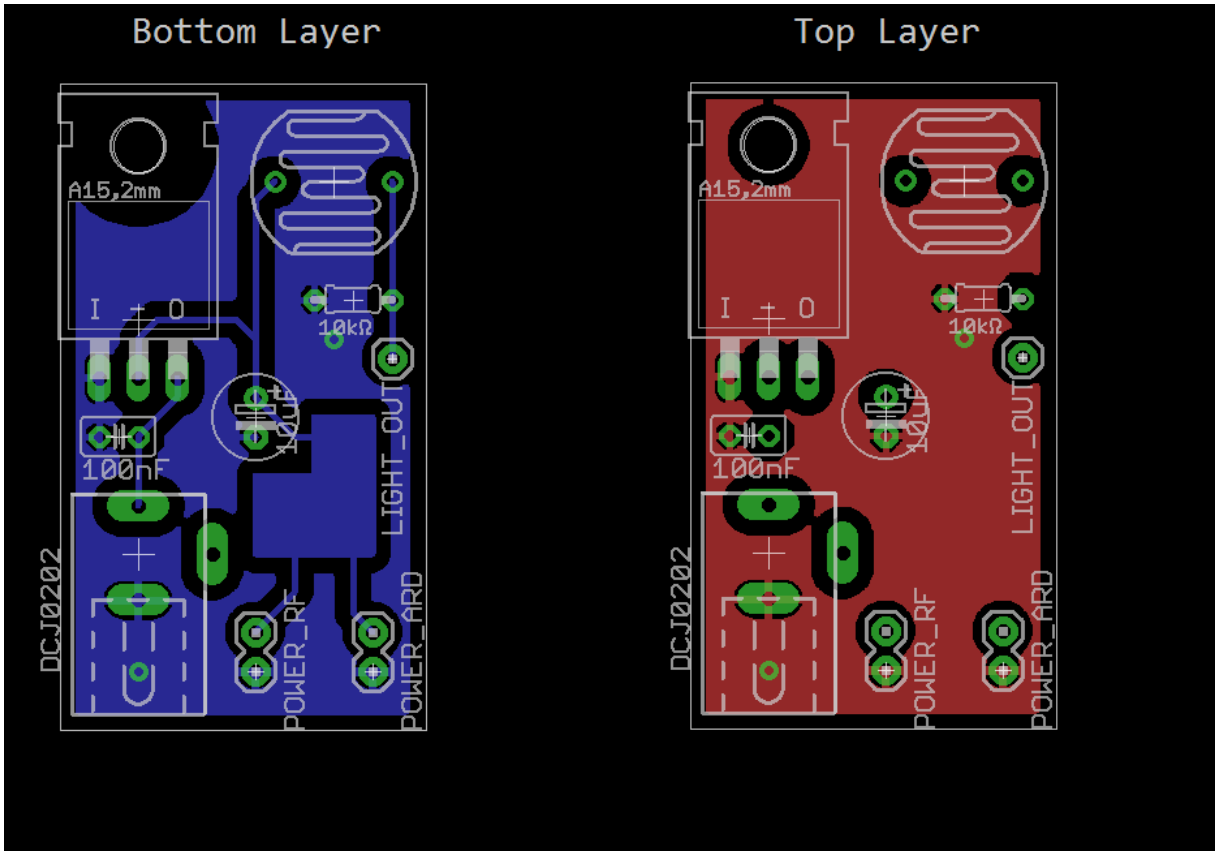

Figure 4.11: Outside Sensor PCB Board Layout

Chapter 5.

Wireless Communications and Networking

The devices communicate wirelessly with the AP, which is a BBB development platform that controls an RFM22B radio over SPI and is also connected to the internet over Ethernet cable. This type of network topology is defined as "star network topology", since the BBB acts as the central unit that controls all other devices wirelessly. In a star network, devices can be removed or added easily and without disturbing the rest of the network[2], allowing for a more flexible approach to automating a home. Also, the maintenance tasks of the devices connected to the AP are simple and can be carried out without having to shut down the network. One of the major disadvantages of this network topology is the physical networking requirements, since a large amount of connections is required, which, if wired, can increase the cost substantially. This, however, does not affect the proposed system, since all the communications are performed wirelessly by using the RFM22B radios. The only potential problem stems from the reliability of the network nodes to the AP. The AP is called the single point of failure for the network, since if it is damaged, the whole network has been rendered unusable. This can be countered by supplying the AP with backup power and securing it with strong usernames and passwords, since it is directly connected to the Internet. These should be adequate measures to achieve optimal uptime and performance.

The SCM devices communicate with the AP by transmitting and receiving RF messages, therefore no cables and wired connectors are required. Installing the AP in the center of the house is the recommended practice, in order for the radios to be able to communicate with as few problems as possible. By using small coil antennae, the output power of the RFM22B radios is increased, providing much better wall penetration inside the home and extended range. As such, the devices are ideally positioned spherically around the AP so that all the radios can reach the AP with no problems.

The RFM22B radios are programmed to operate in the 868 MHz frequency band. This means that whenever a packet is transmitted by a radio, it can be received by all the other radios that are listening and are located within range, creating a number of problems. For

example, if the AP transmitted an RF packet in order to turn on or off a relay, it could be received by many different SCM devices simultaneously and processed. In order to avoid this, each RFM22B radio is also programmed with a certain "identification word", which consists of four bytes and it is unique to each SCM device. More specifically, each APM has a four byte word hard-coded into their program, which is used to identify uniquely each SCM device. These bytes are transmitted to the AP during the synchronization phase and are stored in the MySQL Database as the unique identifier of each SCM device. This means that whenever the AP wishes to send a packet to an SCM device, it must recall their specific identification word from the database, set the Transmit Header registers of the RFM22B radio to the identification word and transmit the packet. The RFM22B radios on the SCM devices are programmed so that the Check Header registers are set to the identification word. According to the RFM22B manual, whenever a packet is received by a radio, the received packet's header is checked against the Check Header bytes. If they are a match, then the packet received is valid. If they are not, then the packet is simply discarded. The Check Header bytes of the AP are set to the word "BASE", which logically means that the TX Header bytes in every radio of an SCM device are set to this specific word. It is worth mentioning that a manual identification mechanism was also implemented, because, in some cases, packets were processed by devices that were not supposed to, which indicated that the Check Header and TX Header registers were insufficient on their own. The manual identification mechanism was implemented only as a precaution and required the transmission of the receiver's identification word within the data field of the RFM22B packet, which was later processed by the software.

As stated above, RF communications are packet based. This, however has nothing to do with the way the RFM22B radio handles internally their packets, which was presented in an earlier chapter. All of the information transmitted and all of the packet types described in the present chapter is stored in the "data" field of the internal RFM22B TX FIFO. That is, packets of data are created and labeled manually for use in the system, which, of course, are transmitted in the "formal" way of the RFM22B. Depending on the information transmitted, a different packet is used. However, all packets follow a specific format. Every packet is fragmented into data fields, which are separated by a "***:***" character, as shown in **Figure 5.1**. The TYPE field is used by the receiver, in order to identify the information carried by each of the data bytes. In

the proposed work, the star topology of the network prevents SCM devices from communicating with each other, as they are programmed to be controlled and report directly to the gateway. Therefore, routing is performed manually and in order for packets to be targeted to specific SCM devices the FROM and TO fields are required. These are filled by the transmitter. The four unique identification characters of the transmitter are contained in the FROM field and the four unique identification characters of the receiver are contained in the TO field. Both fields are 4 B in length. Whenever a device receives a packet it checks these fields and accepts only packets intended for the specific device and discards the rest.

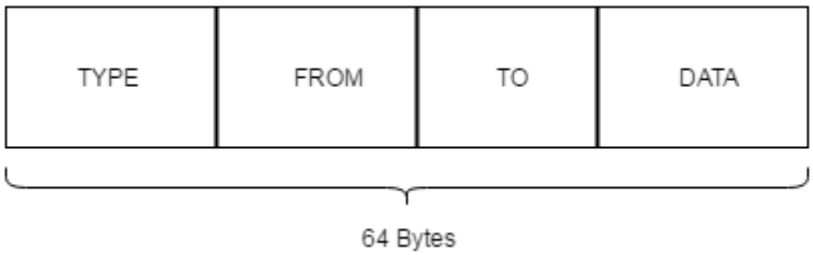

Figure 5.1: General RF Packet Structure

The packet types are separated into categories, according to their intended purpose. A complete list of the packets used in the proposed system is shown in **Table 5.1**. Not all packets follow the general RF packet structure shown in **Figure 5.1**, as the system has been designed in a way that avoids transmitting many unnecessary bytes for power consumption. This means that packets that can only go "one-way", such as packets that can only be sent from the AP to the SCM devices, do not include the "FROM" field, as it is redundant. These are the RELAY-type packets. This principle also holds true for packets that are sent from the SCM devices and can only go to the AP and they do not include the "TO" field, as it also leads to transmitting redundant bytes that hold no real information. Such an example is the SYNC-type packets.

The SYNC-type packets differ in some other aspects from the general packet structure, as shown above. They are created with two extra fields, which help the gateway identify the role of each SCM device and properly insert the relevant information into the MySQL database. In more detail, the SYNC packets follow the general RF packet structure, as shown in **Figure 5.1**, without a "TO" field, since these are only aimed towards the gateway. Instead, the type of the

device as a string of characters is transmitted, followed by a single byte, the identification mask, and the DATA field, which contains the current measurements from the sensors installed on the device, or, if the device has a relay, the current relay status information. A schematic of the SYNC packet structure is shown in **Figure 5.2**. Because the SYNC packets are created in such a way, the user can program and create any device they need, according to their specific needs, and the gateway will handle them properly without requiring further modifications. These packets are only transmitted when the devices are powered on for the first time and have not already transmitted successfully their information to the gateway in a prior time. The synchronization status information is stored in the EEPROM memory of the APM. Upon successful synchronization the status byte is updated and is read upon any subsequent power on, skipping the SYNC procedure.

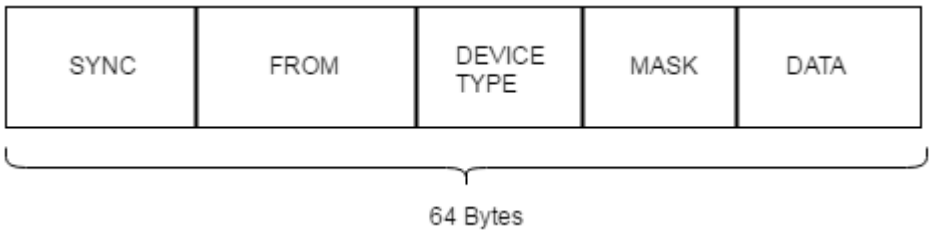

Figure 5.2: SYNC Packet Structure

SCM devices that have Reed magnetic switches are capable of notifying the gateway of immediate changes of the sensor's status. This is accomplished by transmitting a REED-type packet directly to the gateway. Such packets are only meant for the gateway, therefore the TO field is omitted, while the TYPE, FROM and DATA fields are present, according to **Figure 5.1**. In more detail, the TYPE field holds the bytes "REED", the FROM field holds the ID bytes of the SCM device and the DATA field holds the new status of the magnetic switch. Once this packet is received from the gateway, it will transmit a packet to acknowledge that the packet is received and processed successfully. This packet is aimed towards the SCM device that provided the new REED information to the gateway, but carries no data in itself, therefore the DATA field is unnecessary. Instead, it follows the ACK packet structure, as shown in **Figure 5.3**.

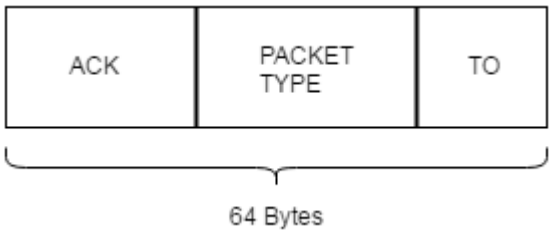

Figure 5.3: General RF ACK Packet Structure

In more detail, the three first Bytes, the "ACK" section, helps identify this packet as an acknowledgement packet, signaling the successful reception of a packet, and is directed towards the SCM device with the ID bytes specified in the "TO" field. In the REED case, the PACKET TYPE field will contain the bytes "REED", identifying the packet as the acknowledgement for the successful reception of a REED packet from the SCM device with the specific ID bytes. This way, the SCM device that sent the REED notification will know that the packet has been successfully received by the gateway and will not resend duplicate packets, avoiding unnecessary subsequent retransmissions. On the other hand, if the ACK packet does not arrive within a specified time frame, then a timeout event will occur on the APM and the packet will be resent. This will continue, until the ACK packet arrives successfully from the gateway to the SCM device. This ensures that no information packets are lost due to packet collisions or a busy gateway and the system monitoring the smart home remains always up-to-date with the current status of the various sensors. The communication between the SCM device that transmits a REED packet and the reply from the gateway is shown in **Figure 5.4**, in all possible cases. The first case shows the failure of the initial REED packet to reach the gateway. The first case shows the failure for the ACK packet to reach the SCM device. The third case shows the successful communication between the SCM device and the gateway, after an arbitrary amount of communication failures. Upon a timeout occurs on the SCM device, a counter is incremented and the packet is retransmitted. The counter is used in order to increase the time in between retransmissions and give time to the gateway to exit from any "busy" state and handle the packet properly. The packets are sent in multiples of the waiting time, T1, of 150 ms with a maximum of 750 ms between retransmissions. The REED-type packet will continue being transmitted, with each retransmission occurring after increasing time intervals, until the proper ACK-type packet is

received from the gateway.

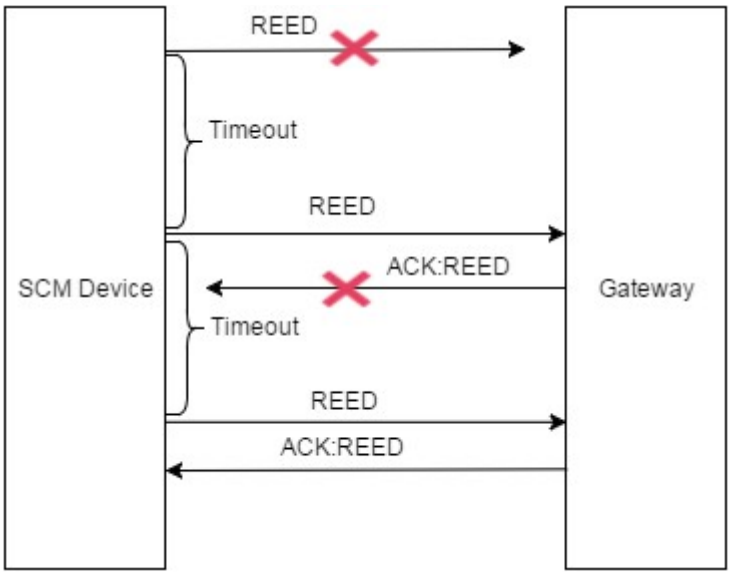

Figure 5.4: Communication Diagram w/ Failures During REED Packet Transmission And ACK Reply

The RELAY-type packets are transmitted from the gateway (AP) to SCM devices that control relays. They carry information in order to change the current state of the controllable relays and are used to power on or off the devices that are attached to the relays. These packets are only sent from the gateway, therefore the "FROM" field is not required. Furthermore, the ACK requirement is also present, as the gateway will require an acknowledgement of the successful reception of the RELAY packet from the intended SCM device, in order to update the new status to the MySQL database. This process is presented in the communication diagram w/ failures of **Figure 5.5**, which shares a lot of similarities with the REED communication diagram. One major difference though, is what happens upon communication failure. The gateway will send the RELAY-type packet and if no acknowledgement is received within the waiting time window, then a counter will be incremented and the packet will be retransmitted. The counter works as a means to implement a scalable timeout, just as described during the REED-type transmissions. The waiting time between gateway retransmissions will be incremented in multiples of the T1 time, the 150 ms default value, until it reaches the upper limit of 750 ms. Then, the gateway will not resend the packet. Instead, an email message will be generated and sent to the user, using the registered email address, notifying them about this failure and requesting servicing

the network node that failed. This option is more suitable than resending until an acknowledgement is received, because the gateway is the critical point of the network and it should never hang. It has many other things to handle and it also communicates with every other SCM device and the user simultaneously, therefore it cannot afford hanging the whole network on a potentially damaged node.

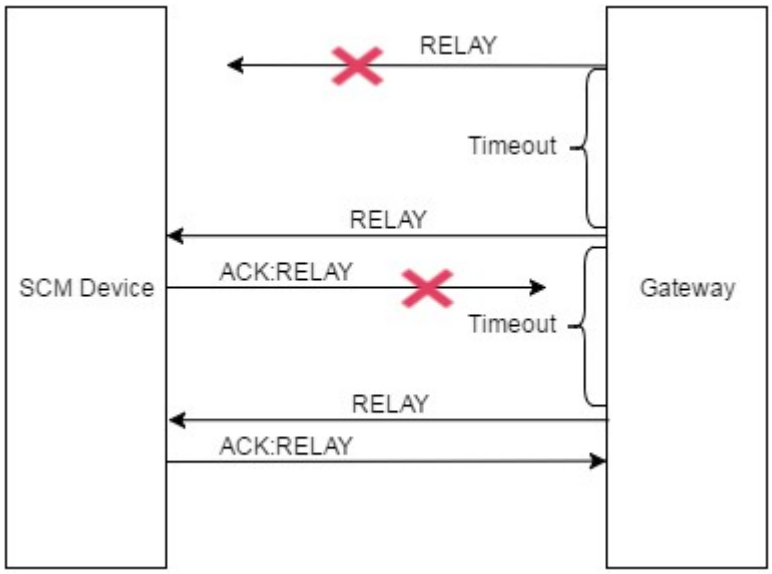

Figure 5.5: Communication Diagram w/ Failures During RELAY Packet Transmission And ACK Reply

The PING-type and MEAS-type packets are used in order for the gateway (AP) to get new, updated measurements from the sensor-type SCM devices. More specifically, the gateway implements a software timer, responsible for counting the time in between sensory measurements, and periodically checks if the timer has expired. The default expiration value is 1800 seconds, or equivalently 30 minutes. Upon the expiration of the timer, a certain event is triggered which can be captured by the main loop code within the program. If this happens, then the gateway will have to take updated measurements from the sensor-type SCM devices. More specifically, the gateway will first gather the information from outside the smart home by sending a PING-type packet to the "outside" sensor-type SCM device. Upon successful reception of the PING-type packet, the "outside" device will collect data regarding the ambient temperature and the ambient lightning and transmit the information back to the gateway in the form of a MEAS-type packet. No acknowledgements are required, since if a communication problem arises, then the gateway will try again to communicate with the

device. Once the gateway collects the data from the "outside" SCM device, it will move on to the "photo" SCM devices in a similar fashion. It is worth noting that in case no "outside" SCM device is logged in the database and, therefore, it is not installed, an email message is generated and sent to the user, informing them of the problem. The program is automatically terminated because the user must immediately synchronize the missing device by using a different program. The complete communication diagram of the measurement procedure is presented in **Figure 5.6**.

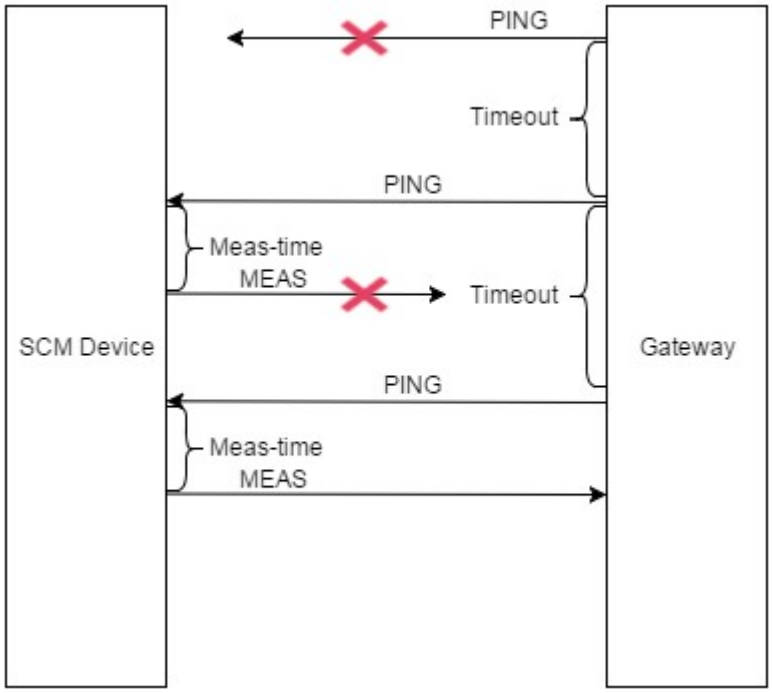

Figure 5.6: Communication Diagram w/ Failures During PING Packet Transmission And MEAS Reply

It is worth mentioning that during the PING-MEAS transaction, the need of an ACK-type packet is eliminated. This is happening only because the gateway is the master device and it is responsible for requesting data from the other devices in the network. The sensor-type SCM devices will not transmit any information on their own, as they require a PING-type packet from the gateway in order to reply with a MEAS-type packet. This ensures the flexibility of the system, as the gateway controls the arrival of the measurement packets and the user is able to alter the timeout parameter only once, in the gateway device, to modify the frequency of the measurements. It also ensures that no excess transmissions will be made, as would happen if the sensor-type SCM devices were running their own timers. The

gateway will never be busy when it requests a MEAS-type packet, therefore unnecessary transmissions are avoided.

Type	Data	Description
SYNC	Current measurements from the interfacing sensors	Transmitted by an SCM device that has not already been registered with the system. It holds data about the device's type and the sensors it can interface with.
REED	Reed state	Transmitted by an SCM device that interfaces with a Reed magnetic switch whenever the magnetic switch detects a change.
RELAY	ON/OFF	Transmitted by the gateway (AP). Intended to power on/off a relay.
PING	–	Transmitted by the gateway (AP) to the sensor-type SCM devices whenever new measurements are required.
MEAS	Miscellaneous data and measurements from sensors	Transmitted by the sensor-type SCM devices as a response to the PING-type packet. Contains all the measurements from the relevant sensors in order.
ALARM	ON/OFF/NONE	If transmitted by the gateway (AP) it notifies a sensor-type SCM device to enable its alarm functionality. If transmitted by an SCM device, it notifies the AP that the alarm has been triggered.

Table 5.1: RF Packet Types

User interaction and control of the system is performed by passing requests over TCP to the main program running on the gateway (AP). More specifically, the main program is running a socket server in a separate thread, responsible for receiving and handling the user requests. The server part of the program, is based on C sockets and is configured to listen

and accept a small number of connections from the local host. The small number of queued listening connections is mainly chosen because of the hardware limitations of the BBB. Also, the user should be the only one who connects and uses the system, therefore there is no need to handle many different connections simultaneously. The listening server socket is created when the main control program is executed and it uses the port 3434 for accepting new connections over the IPV4 protocol.

The code is using the select system call for implementing a non-blocking server with a 5 second timeout parameter that is capable of handling different socket connections simultaneously. The main control program of the gateway creates a new thread upon starting which is responsible for setting up the server socket on port 3434, listening and accepting new connections from the clients. The maximum queue length of the pending connections for the listening socket is set to 3, in order to be able to handle more than one users in the house who are sending requests to the gateway. A certain flag is initialized as well, that monitors the state of the server. If the server unexpectedly stops, then the user is informed of the error via a generated email and the program terminates, awaiting inspection or maintenance. Also, when the main program is terminated by the user, for any reason, the server thread is also shut down and no more user requests are received and stored to be processed at a later time.

The implemented request mechanism in the system is based on a data queue. Each node holds the unique identification bytes for the relay the user wishes to change, the new power state and any extra information required by the relay, such as the hot/cold state of the air conditioning, which is controlled by a separate channel on the relay controlled by the appropriate SCM device. Upon execution of the main control program, the queue is initialized and memory is initialized to store a maximum of 10 nodes, which is more than enough for a small home, but can change from within the source code.

The process through which the user passes requests to the gateway is fairly complex and is composed of a few different steps. Since the user should control the system from their Android devices, the components that formulate the request are sent to the gateway, using the POST HTTP method. More specifically, the POST parameters are parsed by the PHP script on the server's side, the data is checked for consistency and the client program is

subsequently executed, using the PHP function *shell_exec()*. The output of the program's execution, the *stdout* pipe, is then parsed as a JSON message and is sent as a reply back to the Android device for the user to see. On the server's side, after the client program is executed, the socket server thread of the main program will accept the connection and read the required data from the socket, such as the SCM device's identification word (which is 4 B in length), the power status and the extra function byte, in the form of a string. The server replies to the client program and processes the string while the client disconnects. The string is split into components and stored in a new node, which is inserted to the end of the queue, or equivalently, it is enqueued. The main program periodically checks the queue to see if requests are pending, or equivalently nodes that have not been dequeued yet. If that is the case, then an event is triggered and the program will process the head of the queue, then it will form a RELAY-type packet, directed towards the device the user wishes to change, and, finally, will transmit the packet over RF and wait for an acknowledgement from the relay SCM device, before it updates the status in the MySQL database. This process is already described above and the RELAY-type communication diagram is shown in **Figure 5.5**.

The system is also capable of providing security in the form of an alarm function. When the user wishes to leave the house, the alarm function can be enabled via the Android application by sending a special request to the socket server of the main program. Instead of the relay-type device's identification bytes, the message "ALRM" must be sent. Therefore no relay-type device can be identified by these four characters, as it will never be accessible by the user. When an alarm is triggered, the "*photo*" sensor-type SCM devices will transmit an ALARM packet to the AP and wait for an ACK packet, following the same procedure as the REED-type packet case described earlier. The way the alarm code works will be explained in more detail in a following chapter.

Chapter 6.

MySQL Database

All information regarding the smart home is stored in a MySQL database. It can be easily retrieved and presented, or even altered, from the designed web page, the Android application and the BBB service controlling the smart home. The UML class diagram of the created database is shown in **Figure 6.1**.

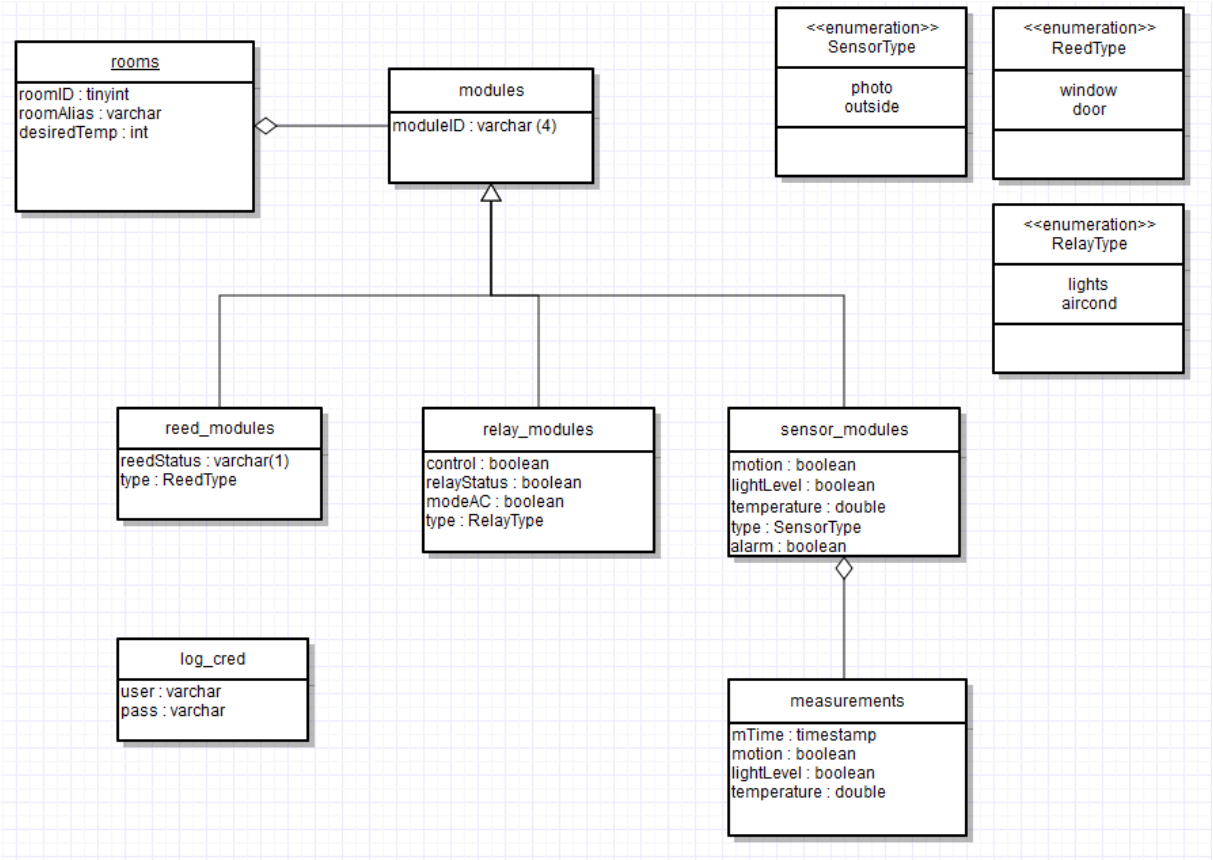

Figure 6.1: MySQL Database UML Class Diagram

The "*rooms*" table is used to store all information regarding the registered rooms inside the smart home. In more detail, it stores a unique identification number, an alias for the room, which is provided by the user during the synchronization process, and the desired temperature inside the room. The desired temperature is an integer number and directly affects the actions taken by the control algorithm, regarding the regulation of the interior environment.

Each room the user wishes to regulate or control must have at least one installed SCM device. The devices are stored in the table labeled as "*modules*". They are characterized by their unique identification code, an alphanumeric four-letter word, which is hard-coded into each of the Arduinos during the programming phase. This is to ensure there are no conflicts between the IDs of the installed SCM devices and to simplify the installation procedure, making the whole system flexible and allowing for each module to be moved to any other room easily, if needed. The primary key of the table is the combination of the "*moduleID*" and the "*roomID*" columns.

The modules are split into three major categories. The ones with installed Reed switches are stored in the table named "*reed_modules*". This table contains information about the current status of the Reed switch (open or closed) and the type of the module, classified into "door" and " window", according to the intended way of installation. The ones able to switch relays and control other devices are stored in the "*relay_modules*" table and are classified into "lights" and "aircond", for lightning control devices and air conditioning control devices. These types can be expanded as seen fit by the user and add more device categories, such as coffee machines or the home TV. The table lists a number of attributes, which allow the complete control over the devices they interface with. The "control" attribute can be set by the user in order to grant control over this specific device to the control algorithm. If the value is not set, then the control algorithm will ignore the device during the decision stage. This can prevent a light switch to be turned off until reset, or prevent the algorithm turning on the air conditioning, therefore conserving energy. The "modeAC" attribute is specific to the air conditioning control modules. It is *NULL* in the modules that control miscellaneous devices, whereas in an air conditioning devices it describes the air setting, either hot or cold. The information regarding the final module category is stored in the "*sensor_modules*" table and it contains all the modules that are used in sensing the environment and providing the data necessary for the algorithm to make choices. They are classified as "outside" and "photo" and they differ in the fact that the former cannot provide info about motion and are intended for external use, while the latter are intended for internal use and provide information about movement in the room they monitor. The internal devices use a PIR sensor and are to be placed higher, if the owner has house pets. The table labeled "*sensor_modules*" also stores information about the motion setting, which is *NULL* for the "outside" module, the ambient

lightning level, the alarm flag setting, which is used to store whether the alarm setting is activated on which device, and the current measured temperature. The temperature reading is provided by the temperature sensor installed on the RFM22B radio, a feature that reduces the overall cost of the device with a small tradeoff in the accuracy of the temperature reading.

Each of the sensor-type SCM devices can take measurements. These are stored in the database for future reference and can offer insight into the system status overnight, or when the owner is away. The readings are stored in the table labeled "*measurements*", where the motion (if applicable), ambient lightning level and temperature reading of each SCM are stored, along with a unique time-stamp identifier for easier presentation. The system is set by default to receive measurements from the SCMs around the smart home every 30 minutes.

The table labeled "*log_cred*" is where the credentials of the user are stored. The table has two attributes, named "user" and "pass" and represent the chosen username and hashed password respectively. These can be altered only from the website, under the "*House Management*" tab. The password is hashed, using a PHP script, to provide more security and decrease the chances of unauthorized access to the system, preventing information leakages.

The available MySQL version for the Angstrom distribution is the MySQL 5.1.40 version. Given the nature of the BBB and the application it is intended for, InnoDB is not the best choice. InnoDB is performing better than other engines, especially as the simultaneous traffic increases on the server, but this, however, comes at a cost; the high usage of the available RAM on the BBB. When using the InnoDB engine, the RAM usage of MySQL can be over 250 MB, nearly half of the available system memory. Since the system is designed for home and personalized use, the BBB is not expected to handle a large amount of simultaneous connections, therefore the main advantage of the InnoDB engine is gone. By disabling the InnoDB engine and using the MyISAM engine instead, the memory usage of the system dropped substantially, solving the high memory usage problem stated above. However, the use of the MyISAM engine comes with several restrictions, such as missing implementation of transactions, foreign key and relationship constraints. These issues can only be addressed by writing trigger functions to perform the necessary actions, such as implementing the "*ON UPDATE CASCADE*" or "*ON DELETE CASCADE*" constraints. The use of small trigger functions to perform most of the cascaded update and delete actions simply requires some extra

coding.

Having described the tables in the database, it is also necessary to describe the triggers associated with each table. Firstly, the column "*desiredTemp*" of the "*rooms*" table has a limitation: it is required to be in a sensible range for a human living environment, therefore it is regulated against incorrect temperatures upon any update or insertion on the table "*rooms"* with the appropriate trigger. The allowed range for the "*desiredTemp*" column values is regulated from 15 °C to 30 °C, while it defaults to the minimum of accepted room temperature, which is 18 °C. These two triggers are activated before an update action and before an insertion. They checks the provided new value of the "*desiredTemp*" column and it if is out of limits, it is replaced with a *NULL* value and discarded. Similarly, in the "*measurements*" table, it is required for the sensor taking the measurement to be inserted in the database beforehand and specifically under the "*relay_modules*" table. This trigger is fired before any insertions performed on the "*measurements*" table and preserves the data integrity of the database, as it prevents any data injection attacks from unauthorized devices.

In the MyISAM Storage Engine, foreign key constraints are not supported by default. This means that the functionality of the cascaded updates and deletions on any foreign key in the tables needed to be handled separately by triggers. In the event of a manual update on any room's identification number, the "*roomID*" column in the "*rooms*" table, the change had to be also reflected on the "*modules*" table in the respective foreign key, otherwise the modules installed in that room would be inaccessible from the main control program. In order to achieve this, a trigger was created that cascades the changes of any successful update of the "*rooms*" table to the "*modules*" table as well, updating the respective foreign key constraint and guaranteeing the integrity of the inserted data. This trigger is fired after any update action on the "*rooms*" table.

The same procedure should be also followed when the data of any module is altered from the table "*modules*", as the changes of the "*moduleID*" column in that table should be reflected on the respective entries in the tables containing the information about the specific module. This, however, was not implemented, because the modules come with a hard-coded identification word and was therefore unnecessary to change it manually from the database. Such action should never take place, therefore such triggers were not implemented.

The limited memory and storage on the BBB prevent the system from storing measurements over an extended time period. Since the system is set by default to update the status of the sensor-type SCM devices every 30 minutes, this means that each device will send two measurements per hour. A smart home can have a large amount of sensor-type SCM devices, which means that the measurements stored per hour will be twice as much as the number of the installed devices overall, plus one measurement from the device installed outside of the smart home. Therefore, in order to reduce the number of measurements stored in the system, an event is scheduled on the MySQL event scheduler responsible for periodically checking and clearing out all the measurements with a timestamp older than 12 hours. The event is scheduled to run every two hours and clears out all the measurements that have been recorded over twelve hours ago. The time limit is set to 12 hours because we do not want to delete any measurements without the user having reviewed them at least once. A shorter time limit, for example at 8 hours, would mean that there could be cases where measurements are deleted without the user having seen them at least once, such as overnight measurements.

Chapter 7.

System Workings

Having described in detail the parts of the system, it is also necessary to describe their interactions with one another during each phase of the algorithms. This chapter will mainly focus on how all the parts of the system work together and allow the user to control the smart home. The idea and the concept regarding the synchronization of the SCM-devices with the gateway will be presented in detail, as well as the developed software. Furthermore, in this chapter, the control algorithm responsible for regulating the temperature in each room of the smart home and the lightning will be presented and explained. Finally, the way the system operates in an everyday basis will be examined.

7.1. Module Synchronization

After the installation of the SCM devices around the home has been completed, the user has to synchronize them with the AP. The SCM device synchronization is performed in steps. The first step is to power on all the devices that are located in the specific room the user wants to synchronize. This prevents any unwanted SCM devices, which are located in different rooms around the house, from being inserted into the database and synchronized with the system. Secondly, the user must open the android app on their device and select …. . The user then inserts the name for the room they want and the app relays the name specified by the user to the BBB, which starts the synchronization algorithm. Finally, when the process is done, the user is informed of the number of sensors found in the room via email. This process must then be repeated for each room in the house with installed devices.

The process is designed to allow synchronizing and categorizing a number of SCM devices at the same time automatically in order to avoid requiring from the user to insert each device into the database manually. The SCM devices are programmed to check whether they have already been synchronized with the system and if not, they start transmitting their information when they are powered on intermittently. In more detail, once a device is powered on and it has not been successfully synchronized with the BBB earlier, it will create a

SYNC packet and transmit it to the BBB. If no reply is received within the timeout limit, it will enter sleep mode for a random time interval and transmit the information again upon waking up. The SYNC packet consists of the SYNC bytes, the ID bytes of the SCM device, the measurement mask, which specifies what type of device is the one sending this packet, and the data bytes, which provide information about the current status of the communicating device. The SYNC packet format is described in more detail in **Chapter 5**. It is also worth noting that the RFM22B radio can also enter SLEEP mode when the synchronization fails, in order to minimize the power consumption of the whole device.

While the SCM devices are transmitting their information and trying to get registered into the database, the AP is running a separate program for that purpose. Once the user requests for a room to be synchronized in the database, the Android application sends the alias of the new room to a PHP script, hosted in the BBB. There, after a brief verification process, the synchronization program is executed from within the script, with the alias provided as input. This program is responsible for inserting a new room with the provided alias into the MySQL database, if it does not exist already, listen for RF connection requests until the timeout between two packet arrivals expires and process the data received. This timeout is set to 25 seconds since the maximum sleeping time of the SCM devices is 20 seconds long. Therefore, no device is skipped during the synchronization, since each device wakes up at least once every 25 seconds. Once the RFM22B radio detects a valid packet, the nIRQ pin will go LOW, informing the BBB that an SCM device is trying to get synchronized. The BBB checks the structure of the packet to verify it is valid and if it is, an ACK packet is sent from the BBB to the device and the data inside the packet are split and handled. Finally, all the information received about the SCM device are inserted into the MySQL database and the BBB resumes listening for any other device trying to get synchronized.

This synchronization procedure ensures the registration of all the SCM devices without any problems. Each device transmits the identification data as programmed upon waking up and waits for a response from the AP in the form of an ACK packet. In case of failure, then the device tries again, after a random time delay. This delay ensures the minimum number of overall packet collisions and optimal packet routing within the 1% duty cycle limit of the 868 MHz frequency band, in case the specific band is preferred. The communications diagram

of a successful synchronization is presented in **Figure 7.1**.

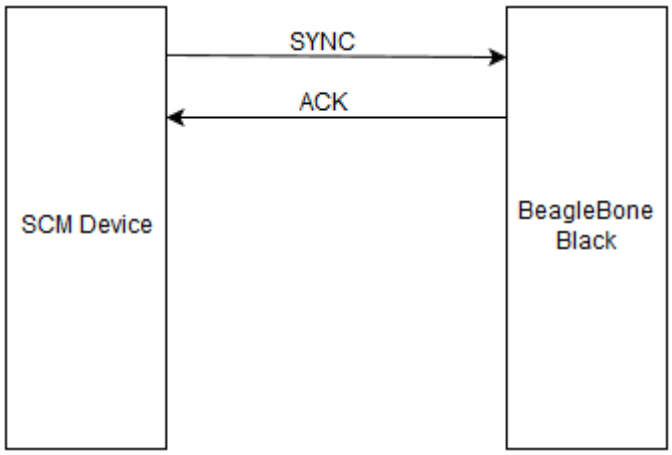

Figure 7.1: Communication Diagram During Synchronization

Since all of the SCM devices are tuned to the same carrier frequency for transmitting and receiving packets, the packet transmitted from a device can interfere with other packets being transmitted at the same time. If this occurs, the packets can be discarded. This is also known as packet collision and they lead to unnecessary packet retransmissions. A single room can have many different SCM devices installed which can lead to packet collisions during the synchronization phase, resulting in higher power consumption and RF pollution. This problem was countered by having each of the devices wait for a random amount of time before retransmitting their SYNC packet. More specifically, the APM of each device uses the RFM22B to send a SYNC packet and waits up to 150 ms for a reply. The value of 150 ms was chosen after measuring the time it takes for a packet to be transmitted and the reply to be received. The values were in the interval [53, 70] with the most common value being that of 64 ms, when the devices were close and above 85 ms when they were separated by a wall. So, a value of 150 ms was chosen as the timeout, since a few milliseconds of delay do not impact the design and 150 ms are almost two times the worst-case scenario, offering reliable communication between all devices. If the SYNC packet did not arrive correctly due to a collision or other noise, then the AP will send no ACK, forcing the APM to enter sleep mode. The APM uses the random() function call to generate a number in the interval [1,000, 20,000], which is the number of milliseconds the APM will delay the program execution for. After waking up, both devices will have waited for a different amount of time and will not interfere

any longer. A full diagram showing the resolution of such a collision is shown in **Figure 7.2**, where two devices are both transmitting their data in the same time slot, labeled T1. Since there is a collision, none of them will receive a reply, so they will both generate a random millisecond waiting time and enter sleep mode. In the example shown in the figure, the SCM device 1 will wait for two time slots, whereas the SCM device 2 will wait for only one time slot before retransmitting. Since the second device will wake up sooner, the next transmission will also happen sooner, in a clear channel with no interference, as the first device is still sleeping and effectively the collision has been resolved. In case only one packet got discarded from the interference and the other went through, the collision has been resolved, since the AP will reply to the packet it did get, whereas the second device will sleep and when it wakes up, the transmission will be completed.

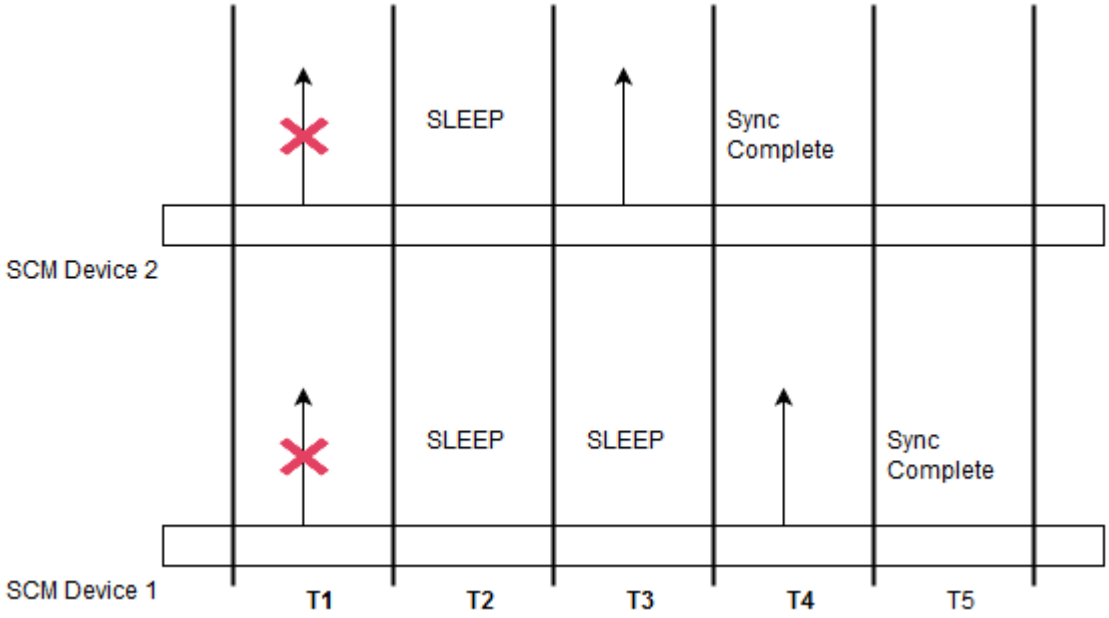

Figure 7.2: Collision Resolution Between Two SCM Devices During Synchronization

In order to use the random() function call and get different random waiting times on each APM, the seed of the pseudo-random generator must be initialized. The recommended way is to read the analog voltage of an unconnected analog pin and use the value read as the seed of the sequence. By doing this, the APM will use as seed value the random electric noise generated by other devices operating in close proximity. While this is not a very accurate random number generator, it serves adequately for randomizing all the APM devices in the

current work, as the randomness of the environment and the analog input voltage is also combined by the randomness of the starting time for each device. Since each device is installed and running for different amounts of time, even if two APMs have the same initial seeds, they will not run for the exact same time, thus making the waiting times truly random. This can be also augmented further through software, since there are many truly random libraries for the Arduino available online, while another option is connecting the analog pin to a hardware piece that provides better random voltages. During the testing phase in an apartment room with one SCM device of each type, no collisions were noticed.

7.2. Normal Mode of Operation

Once the installation has been completed, the developed system is ready to control the devices of the smart home and regulate the inner home's environment. This is achieved by receiving data from the various sensors on the SCM devices and subsequently using the data in order to make "smart" choices. It is logical that all the programmable devices should perform certain actions upon the occurrence of certain events. Examples of such events are a switch opening or closing, or the temperature inside a monitored room rising above a threshold. Since the whole system can be modeled as a set of event-driven finite-state machines (**FSM**),operating in unison, programming all devices using automata-based programming is the most viable solution.

Each SCM device is used to monitor different events and provides different inputs. During the normal mode of operation, the Reed type devices will inform the AP whenever the monitored switch changes states. This keeps the AP always up-to-date with the current status of each monitored window or door. The APM of the reed-type SCM device is programmed as an event-driven finite-state machine (**FSM**), consisting of a total of five different states. It is capable of transitioning between these states after one of four distinct events occur. The device enters the neutral state (**ST_NEUTRAL**) right after the setup and synchronization phases are over. In the neutral state, the APM loops and continuously checks for events. It remains in this state while waiting for an event to be triggered. The available events are triggered by the state change of the magnetic switch, the receiving of an ACK packet from the AP, the occurrence of a receive timeout, or none of the above. All events are returned

from the getEvent() function, which is responsible for checking whether the conditions of each event are met and returning the appropriate enumerated event_t value. When the reed changes state, the "*EV_REED_CHANGE*" event is returned to the neutral state, where it is caught and the program transitions to the Reed packet creation state (**ST_REED**), where the package to be transmitted to the AP is formatted before it is actually transmitted. When the packet has been created, the program transitions to the TX state (**ST_TX_REED**), where the packet is loaded to the TX FIFO of the RFM22B radio module and transmitted to the AP. In this state, a special counter is initialized. The counter is responsible for counting the number of times the timeout for a valid ACK pack has expired. Once the packet has finished being transmitted, the program transitions to the RX state (**ST_RX**), where it waits for a valid ACK pack from the AP, informing the APM that the REED packet has been successfully received. In case the timeout expires, the program transitions to the RTX state (**ST_RTX**), which is also called the retransmission state, where the packet loaded in the FIFO is retransmitted before the device enters the RX state again. In order to avoid overloading the wireless network with RF packets and conserving power, since the RFM22B radio draws the most current during a packet transmission, the waiting time for the ACK pack increases as the timeout expiration counter increases. This mechanism was implemented in order to allow for more time in between retransmissions and increase the chances of the AP to have returned to its neutral state and serve the packet the APM tries to relay.

The relay SCM devices work in a similar fashion, with the exception that these devices do not provide information to the AP, instead they are waiting for directions from the AP. The finite-state machine of the relay program consists of three main states and the transitions between them are caused by three or four different events, depending on the sub-category of the device. As with the reed devices, the relay devices also enter the neutral state right after the synchronization process is over, where it loops continuously until an valid event is generated and returned from the implemented "*getEvent()*" function. When the AP requires a relay to be powered on, it will send a RELAY packet to the SCM device responsible for controlling the relay. When this type of packet is received, it triggers an event on the SCM device (**EV_RELAY_ON**) and the APM turns on the controlled device through the relay. Controlling a 5 V relay from a 3.3 V APM requires a voltage level shifter, so the lower voltage can be translated into a higher one and vice versa. In the specific case of the "*aircond*" device,

the AP also informs the device of the mode it should operate, either heating or cooling. This is controlled by a separate channel on the 2-channel solid state relay the "*aircond*" relay uses. A similar procedure is followed whenever a relay must be switched off. A different packet is sent, that informs the SCM to turn off the controlled device by opening the relay. After the packet is received and the relay is switched, the SCM also transmits an ACK packet to inform the AP of the successful reception and the subsequent relay switch. If this packet is not received by the AP during the timeout period, then the AP will resend the RELAY packet and the process is repeated. The AP will wait between retransmissions according to how many times the ACK failed to be received, in a similar way to the previous case, to avoid consuming excess power and overloading the RF network.

Sensor SCM devices follow the same guidelines and are programmed in a similar fashion. Not all types of sensor SCM devices have the same number of states, as the "*photo*" devices are also programmed to function as an motion detector alarm. Transitions between the states are triggered by different events. All sensor-type SCM devices act as passive receivers, as they are constantly receiving input from their environment, but the AP must request from them to transmit their data. After the synchronization process, they enter the neutral state, which they will only escape when a PING packet from the AP arrives. When this happens, the "*getEvent()*" function returns the appropriate event (**EV_PING**) and the program makes a transition to the measurements state (**ST_MEAS**), which is where the measurements packet is created. After the packet has been formatted, the program transitions to the TX state (**ST_TX_MEAS**) and the packet is transmitted to the AP. In this case, the wait for the ACK pack is unnecessary, as the AP is waiting for a MEAS packet and if it is not received until the timeout expires, the AP will send a new PING packet and wait again for the respective MEAS packet. This process will continue either until the communication is successful and the AP receives the packet with the measurement data, or the number of failures excedes the threshold. In the second case, the AP will stop trying to communicate with this SCM device and will alert the user of the failure via email message.

In the case of the "*photo*" sensor-type SCM device, there is also the alarm capability that adds to the complexity of the code. More specifically, when the user wishes to leave the smart home, or before bedtime, the alarm function can be enabled. This means that the main

control program will send an ALARM packet to the installed "*photo*" SCM devices once the proper request is received from the C socket server. When this packet is received from a "*photo*" device, it will enter the alarm state. This means that the device will notify the AP with an ALARM TRIGGERED packet once it detects motion and subsequently wait for an ACK packet from the AP. In more detail, once motion has been detected, the APM also checks if the alarm flag has been also enabled. If both of the prerequisites are met, then the "*alarm_triggered*" flag is set. In the neutral state, if this flag is set, then the device must send an ALARM packet and transitions to the proper state (**ST_TX_ALARM**). In this state, the same principles as before are followed. The device will transmit the packet and enter the RX state (**ST_RX**), waiting for the ACK packet from the AP. If the ACK is not received and the timeout period ends, then the APM enters the retransmission state (**ST_RTX**) and the packet is resent. This procedure is repeated until the ACK packet is received. There is no upper limit of times tried to communicate with the AP, as this packet is very urgent and it must be received. Otherwise, there would be a potential security threat.

The AP is the core of the system and has the most complex FSM, which consists of a total of forty one (41) states and ten (10) different events for transitions between them to occur. The main software took around 2,500 lines of code to be perfectly operational, while a lot of effort was put into its optimization, such as reusing states and lowering the CPU and memory requirements to a minimum. Miscellaneous libraries that were coded required an overall of another 2,000 lines of code. The main program is to be executed once the system has been initialized and all devices have been synchronized successfully. If the main program runs on the AP and the user selects to synchronize SCM devices, then this results in undefined behavior. The initial state is the initialization state (**ST_INIT**), where the user request queue is initialized to a maximum of ten (10) requests, the socket server thread is started, the corresponding flags as well as all the arrays used in the program are initialized, the total number of rooms is calculated from the MySQL database records and the timer responsible for taking measurements is started. In more detail, once the server thread has been executed, a special flag is also initialized. The flag is shared between the main thread and the server thread and it is set to "*true*" while the server thread is still running. If the server thread fails and terminates by an error, then this flag is switched to "*false*" and an event is triggered. When the main thread receives this event, it will terminate with an error. One other flag is the

one responsible for constantly iterating the main program loops in both the server, allowing it to continuously accept and handle new connections, and the main program. Once all of the above actions have been completed, the program enters the neutral state (**ST_NEUTRAL**), and waits for an event. The neutral state has no output. Instead, the program loops continuously, with a small delay of 500 ms in order to avoid hogging all of the available resources and CPU cycles. During the continuous loops, the program will check for events and once an event is triggered, it will transition to the corresponding state. The priority of the events has been fully specified and they will be checked in sequence, as their priority matters during event conflicts. User requests are checked first, as these are the rarest. Secondly, the program checks for errors or interrupted system calls on the socket server and the polling method. If such an event occurs, the program exits the loop and terminates after notifying the user via email. Next in sequence are alarm and reed packet arrivals from a "*photo*" device or reed-type SCM device. Finally, the program checks whether the measurements timer has expired, which is the least important of the events.

In case a REED packet is received, the reed packet event will occur (**EV_PACK_REED**) and the program will transition to the state responsible for handling the packet and updating the status of the SCM device in the MySQL database (which are the **ST_HANDLE_REED** and **ST_REED_COMMIT_MYSQL** respectively). In the handling state, the ID of the SCM device is extracted from the packet, along with the current status, if the device has been registered with the database. If a unregistered device tries to send data to the AP, then the user is notified of the illegal action via email. The alarm flag is also checked and if it is set, then the REED packet triggers the alarm and the user is informed of this via email. In the case of the alarm flag being set when a REED packet arrives, the user is notified of the triggering, even if there is an error. This measure is taken to ensure that proper security standards are held and maximum security is achieved. Next, the new status of the magnetic switch is committed to the MySQL database and an ACK packet is sent to the device from the ACK state (**ST_TX_ACK_REED**), before returning to the NEUTRAL state once more.

The most complex phase of the program is the algorithm used to take measurements from the sensors around the house and using the information received in order to make "smart" decisions about the environment of the smart home. Once the measurements timer expires,

which is set to 30 minutes by default, the AP must take measurements from all the registered sensor SCM devices sequentially, starting with the outside device and moving on to the "*photo*" devices installed in each of the rooms. This happens in the measurements state (**ST_TAKE_MEASUREMENTS**), which is entered after the timeout event (**EV_MEAS_TIMEOUT**) is triggered in the NEUTRAL state. The AP will transmit a PING packet to the outside device first and wait for the reply with the measurements of ambient light and outside temperature. The received data are used to update the current status of the sensor in the "sensor_modules" table and are also stored in the "measurements" table of the MySQL database for future reference and display on the website. The AP notifies the SCM device by transmitting a PING packet, which happens in the transmit ping state (**ST_TX_PING**) for both types of sensor devices, and subsequently waits for a reply in the form of a MEAS packet, in the measurement RX state (**ST_RX_MEAS**). Once a packet with measurements arrives, its validity is checked and the packet is handled in one of the handling states. Since the system works with only one "*photo*" type device per room, after the measurement data of each "*photo*" device has been processed, the program transitions to the decision states (**ST_DECIDE_AC** and **ST_DECIDE_LIGHTS**) where the main decision algorithm is executed. The temperature decisions state is only entered when motion has been detected, as regulating the temperature in unused rooms is sub-optimal and expensive. If no motion is detected, then the program simply finds all of the air condition and the lights inside the room that are switched on and turns them off.

The purpose of the decision states is to make "smart" decisions in order to regulate the temperature of the room and bring it close to the desired temperature setting, set by the user. The main control algorithm is also responsible for adjusting the lightning inside the room when the ambient lightning is insufficient. As the ambient lightning is measured, it is checked against the environmental lightning outside the smart home and either the lights are switched on or off. In order to regulate the inner temperature, the program quantizes both the outer and inner temperature measurements to five (5) levels, ranging from cold to hot. Temperatures are quantized into discrete values as shown in **Table 7.1**. The program checks the discrete values of the current temperature inside the house against the discrete value of the desired temperature setting, which is provided by the user, by calling the "*quantTemp()*" function. If the inner temperature is close to the desired temperature, no action is required. If

these two values are different, then actions must be taken. More specifically, if the inner temperature is cooler than the desired value and the outer temperature is higher, then by opening the windows the room temperature can be adequately adjusted towards the user's liking. Similarly, if the outer temperature cannot equalize the inner temperature, then the windows should be closed, if opened, or if they are already closed, the air conditioning unit is powered on with a RELAY ON packet and is set to heat mode. The user is informed via email whenever the windows should be opened or closed, which can also be automated by installing motors to the windows. When the inside temperature is higher than the desired value, the system follows the same principles as above, in the opposite way.

Temperature Range	Quantized Temperature
< 15 °C	COLD
15 °C to 18 °C	COOL
18 °C to 26 °C	NORMAL
26 °C to 32 °C	HEAT
> 32 °C	HOT

Table 7.1: Temperature Discrete Values

The user can send requests to the AP from the android application in order to manually control the installed devices around the house. The android application sends the request to the BBB, where it is processed from a PHP script. The script will execute a client-program, which connects to the localhost, to pass the request to the main program using C sockets. Using this method, the C socket server, running in a separate thread inside the main program, will not be directly accessible from outside of the home network, increasing the system security. Once the request has been successfully received, the server will enqueue the request to the queue responsible for handling all the requests coming from the user. Periodically, the main program checks for available data in the queue. If the head of the queue has data available, the user request event (**EV_USER_RQ**) is triggered. The program will transition to the state responsible for interpreting the request information and creating a valid packet (**ST_HANDLE_REQUEST**), where the information of the queue's head node will be retrieved and interpreted. The head is subsequently dequeued. During this state, the program checks whether the request is a simple request or an "*ALRM*" request. In the first case, then the

program will assess the type of the SCM device the user wishes to power on. This is necessary in order to ensure that the user will not waste power by accidentally turning on the air conditioning while there are windows still open in the room. So, if the type of the relay-type SCM device is an "*aircond*" device, the program transitions to the state where the AC request is handled and the room conditions are assessed. Finding open windows prevents the air conditioning from being turned on and the user is informed of the situation via email message. On the other hand, if the device is not controlling an air conditioning unit, or if there are no open windows, then the request is processed as normal with the RELAY packet being transmitted (**ST_TX_RELAY**). After the packet has been successfully transmitted, the program will wait for the appropriate ACK packet from the SCM device in the RX state (**ST_RX_RELAY_ACK**), before updating the status of the device in the MySQL database (**ST_RELAY_COMMIT_MYSQL**). In case the ACK packet does not arrive, a maximum of five (5) retransmissions will occur. If all are unsuccessful, the program notifies the user via email. Afterwards, the program returns to the NEUTRAL state and waits for more events. In the second case, where the user's request is an "*ALRM*" request, the alarm sequence must be started. The "*ALRM*" request is received and handled in the same way as any other. The "*power*" byte is used to select whether the alarm will be enabled or disabled, while the "*extra*" byte is discarded. The program transitions to the alarm request handling state (**ST_ALARM_RQ_CONTROL_1**), where all the "*photo*" type SCM devices are retrieved from the MySQL database and the result is stored. Subsequently, the program transitions to the next state (**ST_ALARM_RQ_CONTROL_2**), where the results are iterated in a sequence, row by row. If the current row is *NULL*, this means that there are no more "*photo*" modules to handle and the program returns to the neutral state. Otherwise, it stores the identification word of the current SCM device and transitions to the next state (**ST_ALARM_RQ_CONTROL_3**). During this state, the ALARM packet is created and the program transitions to the corresponding TX state (**ST_TX_ALARM_RQ**). Once the packet is transmitted successfully, the program enters the RX state (**ST_RX_ALARM_RQ_ACK**) and waits for the ACK from the SCM device. If it does not arrive within the time limit, the same process as before is followed. If it does, it is checked against the template in the alarm RQ handling state (**ST_HANDLE_ALARM_RQ_ACK**) before the alarm flag of the current device is updated in the MySQL database (**ST_ALARM_RQ_COMMIT_MYSQL**) and the program goes back and retrieves the next row

from the result set.

The user is informed of any errors during the main program's execution via email messages. Emails were preferred during development, since they can be read from both a smartphone and a computer, which means that if the user's smartphone has no battery, they can still be notified if the alarm is triggered or any errors occur. The email messages are stored in a separate character buffer of the main control program and are formatted as character strings. Sending an email message is achieved by creating a worker thread, which is responsible for executing the "*sendmail*" program installed on the BBB. In more detail, a function was coded that opens a pipe to the "*sendmail*" program by using the "*popen()*" function and subsequently writes the email message information to the pipe. The email address of the recipient is hard-coded in the main control code, since allowing the user to change it from outside the program would lead to a security vulnerability. Before compiling the main program, the email must be changed to a valid and frequently used address of the user. All emails are labeled as "*BeagleBone – Home Control*" in order to be easier for the user to create a filter based solely on the subject of the emails. When the message has been written successfully, the pipe is closed and the "*sendmail*" program sends the email.

The code responsible for writing and sending an email message could sometimes take up to a few seconds until it was complete. This could significantly hinder the performance of the control algorithm and lead to network overflow problems, since the AP would stall for a few seconds every time an email is to be sent. Therefore, it was deemed necessary to create a new thread for the function responsible for writing and sending the email messages. These threads were detached upon creation by invoking the "*pthread_detach()*" function. Detached threads are threads that their resources are automatically released back to the system upon termination, eliminating the need of the main program thread, or any other thread, to perform a "*pthread_join()*" with them. Since the email threads were to be simplified worker threads with a single purpose that is completed in a deterministic amount of time, detaching them was the most logical solution. Finally, in order for these threads to be able to synchronize and avoid sending overlapping messages, or one thread overwriting the message of the other, mutual exclusion locks (mutex) were required. The mailing threads are performing a "*pthread_mutex_lock()*" on their common mutex lock, ensuring that the email

messages will not be sent simultaneously. Instead, every new mailing thread will try to lock the shared mutex and if it is not free at that time, it will block until it becomes available.

A number of different errors can occur during run-time. Therefore, a safe and reliable mechanism for logging these errors for maintenance purposes was required. The SYSLOG protocol was used in order to address this matter. SYSLOG is supported by the GNU C Library, which provides functions to submit messages to the system logger. Specifically, the program must first open a connection to the system logger. This is achieved by invoking the "*openlog()*" function with the option parameter set to "*LOG_CONS*" and the facility parameter set to "*LOG_USR*", which is the default value. The first parameter is to ensure that any messages that failed to be submitted to the system logger up is written to the system console instead. The second parameter is to set the default facility of the connection as a miscellaneous user process.

Submitting messages to the system logger is performed by either the "*syslog()*" or the "*vsyslog()*" functions. These functions require as the first parameter the logical OR between the selected facility and priority codes, provided in the GNU C Library, followed by the formatted text to be logged in the system logger. A full list of the SYSLOG codes can be found in the official GNU website[21] under the manual section. In the developed code, the errors originating from the MySQL database API are logged using the "*LOG_ERR*" code. The security issues are logged using the "*LOG_AUTH*" and "*LOG_WARNING*" codes. Whenever an alarm is triggered, the event is logged using the "*LOG_NOTICE*" code. Missed packets or unresponsive SCM devices are logged using the "*LOG_ALERT*" code, as these devices might be damaged and they have to be replaced, or they are in need of service. Purely informational logs that are kept for debugging purposes and notifications are logged using the "*LOG_INFO*" code to denote their purely informational nature. Finally, errors that result in the termination of the program, or are of increased severity, are logged using the "*LOG_EMERG*" code, to note that the system has become unusable. Viewing the system logs on the BBB is done via the "*journalctl*" command. Capturing and viewing events in real time is performed by invoking the same command with the "*-f*" flag, to denote the "follow" mode.

When the main control program is terminated, a few things are performed. Firstly, a message to the system logger is submitted and then the connection to the SYSLOG is closed

by invoking the "*closelog()*" function. Secondly, the queue is cleared of any stored requests and the allocated memory of the queue's nodes is freed. After the queue has been cleared, the program will unexport the GPIO pin connected to the RFM22B radio's nIRQ pin, which is monitored via the "*epoll*" mechanism and notifies the program upon a valid packet reception. Any other pins that have been exported are also cleared. Then, the SPI bus will be closed and subsequently an email will be sent to the user to notify them of the termination of the program. This email is sent from a separate thread, which, however, is not detached. Instead, this thread is joined with the main thread before the program successfully terminates as a way to ensure that the mail will be sent before the program's threads are killed by the system. Finally, the main program will join with the C socket server's thread and retrieve a return code from the server. If the return code is not zero (0), then the program will exit with an error, otherwise the main thread is terminated gracefully and the process is killed.

Chapter 8.

Android Application

Creating a user-friendly interface that is intuitive and easy to use by anyone was a challenging objective in the current work. Smartphones have become very common and the Android operating system, developed by Google, is open-source with a very large community of developers and users. Therefore, the easiest way to design such an interface, without requiring separate devices or remote controls, was to create an Android application that allows the user to control their smart home remotely and view the current status of each sensor and installed device from their own smartphones. Doing so provides a solution to the problem with no increases to the overall cost of the system.

The purpose of the developed Android application is to present the users with a simple, yet powerful, and intuitive interface to allow them to control their smart home remotely. The *ligghtpd* server that runs and hosts the web content on the BBB is responsible for providing a medium between the user and the system using PHP scripts. The Android application is capable of connecting to the server, retrieving the requested data in the form of a JavaScript Object Notation (JSON) text, parsing them, and displaying them to the user. It is also capable of sending data via POST to the server and update the MySQL data, or passing requests to the main control program as discussed in earlier chapters. It is also responsible for logging in to the system and retrieving the generated credentials, which are stored internally, so the users do not have to log in every time they run the application or send a new HTTP request to the server. The capabilities of the application are presented more thoroughly later in the current chapter.

The application required extensive networking in order to communicate with the server in a safe and reliable way. A separate singleton class was created that handles all of the networking processes required by the rest of the application. The class is implemented as a singleton, which means that an instance of the class is created once the instance getter method is called for the first time, while any subsequent calls to the instance getter method return the already initialized instance of the class. The singleton implementation was chosen

in order to avoid many instances of the class being created, since sensitive information were stored within it. All the information that is relayed between the gateway and the application is handled by the methods implemented by this singleton class. The class also stores the session cookie, which is generated by the gateway after the user logs in to the system and sent to the application, the server's IP address/URL provided by the user, and the various methods used for completing HTTP transactions, such as some method implementations responsible for retrieving and subsequently parsing JSON data from the server.

The first screen the user is presented with when the application is launched is a splash screen with a custom-made home automation logo. The splash screen stays visible only for a brief period of time, during which the application tries to communicate in the background with the server, assessing the connectivity of the device and the network. If this is the first time the application is running and there is no stored data with the server's URL or IP, then a alert window opens up and requests from the user to input the address of the server. The information inserted by the user are stored in a special file of the Android, called Shared Preferences in the form of a key-value pair. When this is done, the splash screen is dismissed and the user is either presented with the login page, if they have not been logged in previously or if they have logged out, or presented with the main screen of the application.

The login screen is composed of two EditText fields, where the user provides their login credentials (a username and a password), and two buttons, one for logging in and one for changing the server's address. This screen is shown in **Figure 8.1**. Since the "*login*" button stores the contents of the username and password fields before sending them to a PHP script on the server, it is required that they are escaped beforehand, in order to avoid SQL injection attacks. For security reasons, the username and password fields can only accept alphanumeric characters on the application. When they are sent to the server as POST parameters they are escaped again, securing the system even further. The "*change server*" button, when pressed, will popup a new window that contains another EditText field and prompts the user to input the server's IP address. Note that the "*http(s)://*" prefix is required when inputting an address. If it is missing from the inputted text by the user when the IP address is set, an error message will notify the user to correct this.

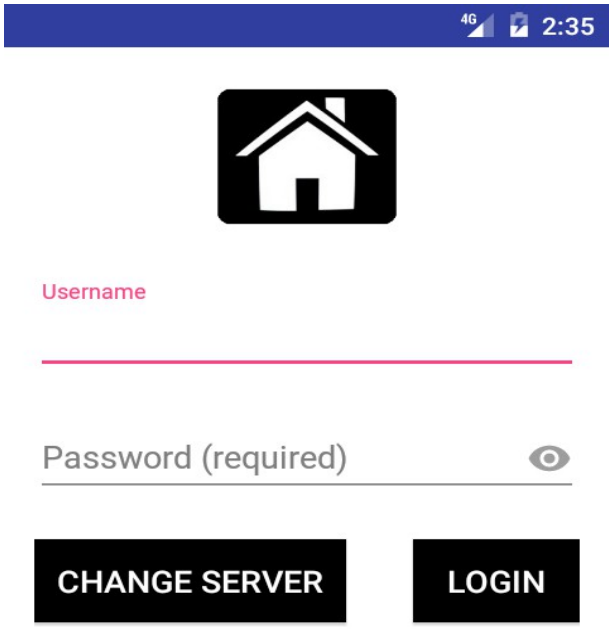

Figure 8.1: Android Application Login Screen

After the user logs in, they are presented with the main screen. In the main screen there are a few buttons available. The first one is labeled as "*VIEW ROOMS*" and it allows the user to navigate to a separate screen, which contains information about the rooms listed in the database. This screen is shown in **Figure 8.2**. As already mentioned, the user can insert new rooms to the database and create a "virtual map" of the smart home layout in the database. The second button is labeled as "*VIEW MODULES*" and it is used to transition to a new screen, which contains all the information about the installed SCM devices in the smart home. Next, right below this button, there are also two buttons side-by-side that enable the user to transmit requests to the main program of the gateway and enable or disable the alarm function. These buttons simply transmit the request to the gateway and inform the user whether the operation was successful or not with the appropriate popup message on the bottom of the screen. Finally, the last button is used to logout from the system. When

pressed, the application will delete the session cookie from the Network Manager singleton class, as well as from the Shared Preferences. The server will be notified of the logout action and the local session file will be cleared as well.

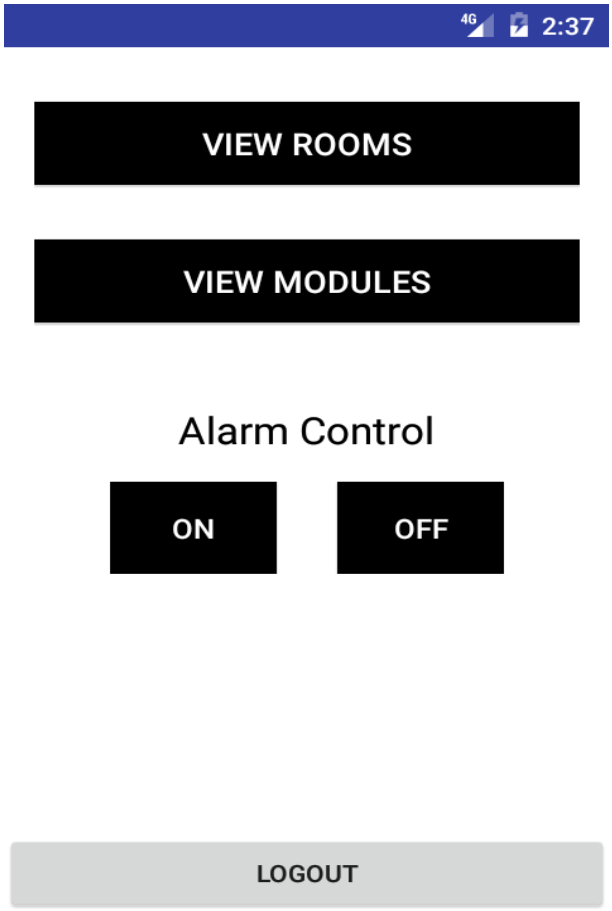

Figure 8.2: Android Application Main Menu Screen

By pressing the button labeled as "*VIEW ROOMS*", the user will be transitioned to a new screen that lists all of the rooms inserted in the database. The new screen is named "*Rooms Overview*", as seen on the topmost of the screen. The relevant screen is shown in **Figure 8.3**. The information regarding all of the rooms in the smart home is presented in a ListView, which allows for presenting information in a vertical scrollable list. It is preferable to allow for a flexible user interface, as a smart home can contain any number of rooms and sensors. Therefore, the user should be presented with an intuitive and easy-to-use interface that makes finding information fairly simple. Each entry of the scrollable list is characterized by the alias of the room, as stored in the database. Right below the room alias entry, a separate line

informs the user of the number of the installed SCM devices in the specific room. Both lines are grouped together as a list item and each list item is separated by a straight line. Each entry of the scrollable list is selectable and when pressed the user is transitioned to a new screen containing information about the selected room, such as the desired temperature setting, used in the temperature regulation process, and the installed SCM devices. At the bottom of the former screen, outside of the scrollable list, a button labeled "*INSERT ROOM*" allows the user to insert a new room into the database. When pressed, a popup alert box is opened, which contains a text field that accepts input from the user and prompts them to insert a room alias for the new room. The alias must be alphanumeric only. In case any other characters are used, the application informs the user that they need to change their selected name and try again. Upon insertion of an acceptable name, the information is relayed to the server and the a new room entry is created in the MySQL database.

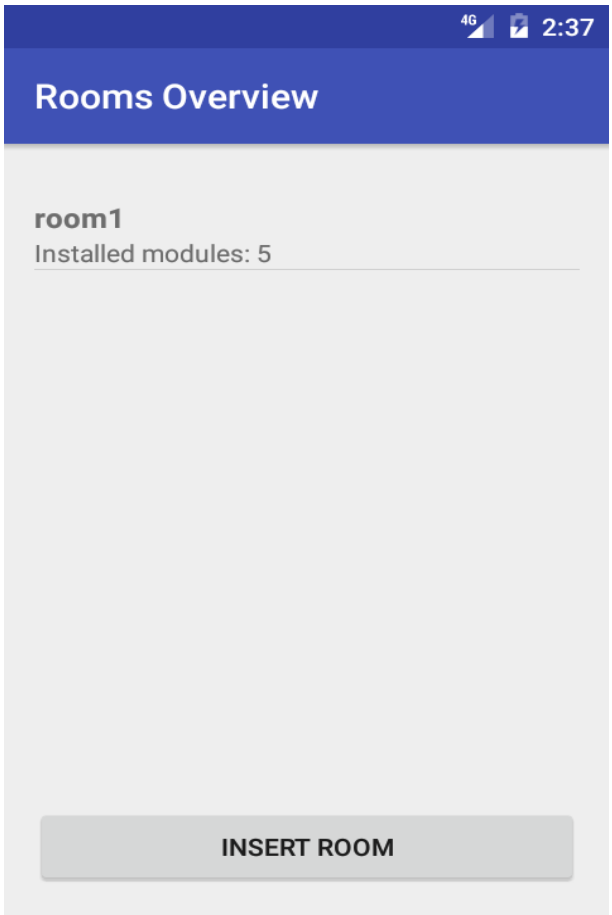

Figure 8.3: Android Application Rooms Overview Screen

Selecting a room entry from the previously mentioned scrollable list, will transition the user to an information screen about the selected room. The new screen is labeled "*Room Details*", as seen on the topmost of the screen. This screen is shown in **Figure 8.4**.

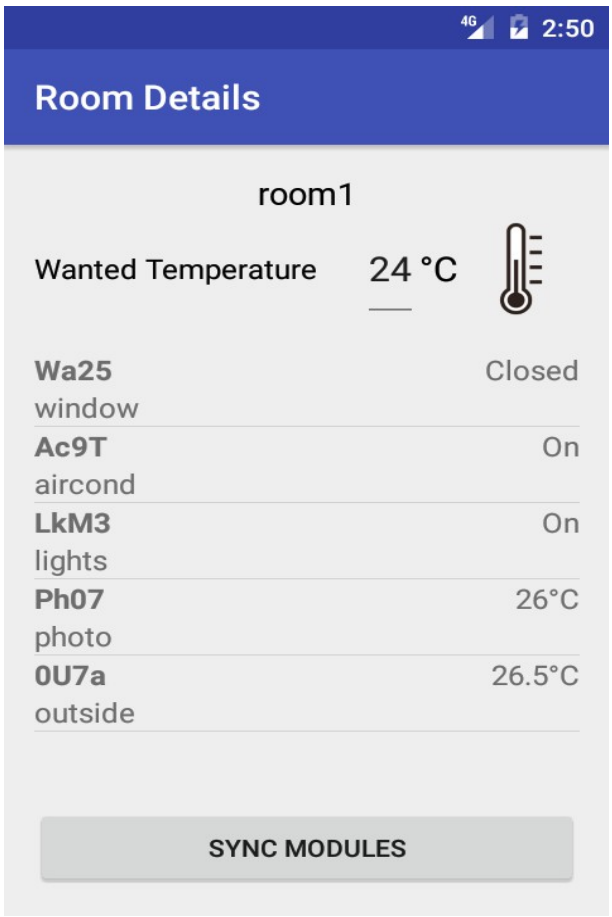

Figure 8.4: Android Application Room Details Screen

Firstly, a text field displays the room's alias on top of the screen, below the screen's title. It is followed by an entry labeled "*Wanted Temperature*" and a temperature number in degrees Celsius, which reflects the information of the "*desiredTemp*" column from the MySQL database. This temperature is used by the main control algorithm in order to regulate the internal temperature of each room. By clicking on the temperature, the user can insert a new number in the specified range of 15-30 °C, which is also the accepted range of the respective MySQL column. Below the temperature setting, a ListView is used to generate a vertical scrollable list, as before, which contains information about every installed SCM device in the current room. Each entry of the scrollable list is characterized by the unique identification

word of each SCM device, which is relayed to the gateway during the synchronization phase of the system and stored in the MySQL database. Below the identification word, in a separate line, the type of the SCM device is also displayed, while on the far right of each entry, a relevant measurement is displayed. For reed-type devices, the status of the monitored door or window is shown as either "*Open*" or "*Closed*". For relay-type devices, their "*On/Off*" status is shown. Finally, for sensor-type devices, a temperature measurement is displayed. Selecting an entry from the scrollable list will transition the user to another screen that displays information and a control interface for the selected SCM device. At the bottom of the former screen, a button labeled as "*SYNC MODULES*" allows for the initial synchronization of the devices in the specific room. This is to be used only during the initial phase, or whenever the user wishes to rearrange the layout of the devices anew, or wishes to add more devices in the future. It must be noted that synchronization and normal control operation is completely distinguished. Synchronization must only happen when the main control program is not running, as both programs use the same radio and SPI bus. Therefore, if both programs run at the same time on the same device it would result in undefined behavior.

At this point it is preferable to go back to the main menu and discuss the function of the "*VIEW MODULES*" button, as it is very similar to the scrollable list presented above. From the main menu, if the user presses the above mentioned button, they will be transitioned to a new screen that displays the information about all of the SCM devices installed in the system, from any room. The new screen is titled as "*Modules Overview*" and displays the sensor information in a vertical scrollable list using a ListView. An example is presented in **Figure 8.5**.The entries are displaying the same information in the same way as with the scrollable list presented in the "*Room Details*" screen. The only difference is that, in this case, every installed SCM device will be shown. By selecting an entry from the list, a new screen opens that displays information about the selected SCM device and a control interface, if the device can be controlled, as with the relay-type SCM devices.

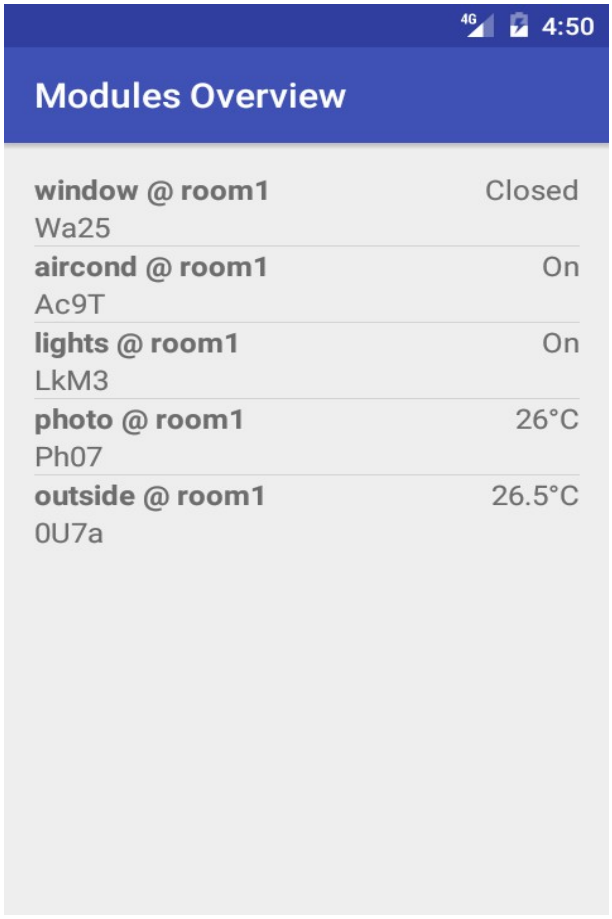

Figure 8.5: Android Application View Modules Screen

After selecting an entry from any of the SCM devices scrollable lists, the user is redirected to the "*Module Details*" screen, which contains information regarding the current status of the device and provides a simple user interface that allows controlling relay-type devices remotely. Depending on the type of the device selected, a different screen will be shown to the user. Selecting a reed-type device will take the user to the "*Reed Module Details*" screen, with only a banner image and some text, notifying the user if the door or window is open or closed. The banner image changes dynamically based on whether the reed device is monitoring a window or a door. An example is seen in **Figure 8.6**.

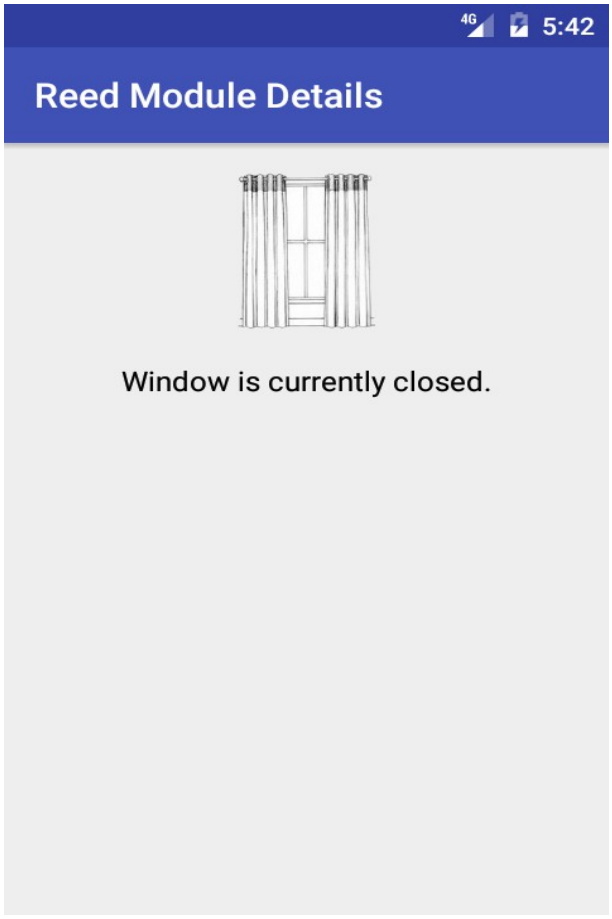

Figure 8.6: Android Application Reed Module Details Screen – Window Sensor

On the other hand, if the user selects a sensor-type SCM device, the new screen will be labeled as "*Sensor Module Details*" and will display the most current temperature reading and a set of three images to the user. The first image shows a thermometer, which is colored depending on the quantized temperature measured by the sensor. Light blue means a temperature reading below 15 °C. Dark blue means a temperature reading between 15 °C and 18 °C. Green stands for normal room temperature readings and represents temperatures between 18 °C and 26 °C. Finally, orange represents a temperature reading that is associated with heat and temperatures between 26 °C and 32 °C, while red represents any temperature above 32 °C. The ambient lightning that is measured by the device is displayed with either a black image with the word "*Dark*" in white letters, whenever the lightning is not sufficient, or a white image with the word "*Light*" written in black letters otherwise. Finally, the third image stands for the motion detection feature of the inner home devices. This image shows an eye

which is either open, in case the sensor has detected motion, or closed with a single crossed line, in case the sensor detected no motion. In case the sensor displayed is the sensor placed outside of the smart home and is incapable of detecting motion, then this image will display a closed eye with two crossed lines on top of it. This way, the information is displayed in the form of an image, making the user interface more intuitive and friendly to the user. A complete example of the aforementioned screen is shown in **Figure 8.7** for a "*photo*" type device.

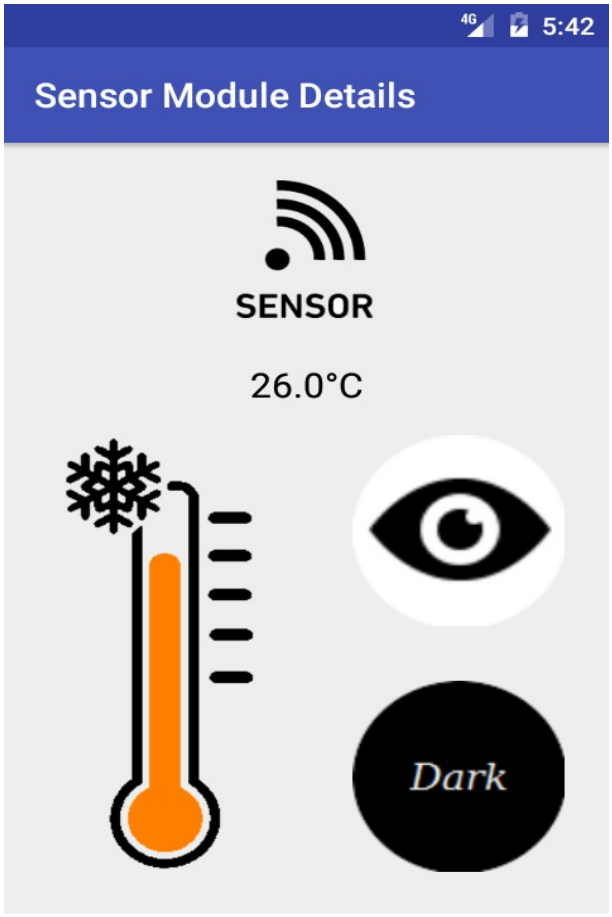

Figure 8.7: Android Application Sensor Module Details Screen - Inside Sensor

When the user selects a relay-type SCM device, they are presented with a different menu from what has been previously described. In order to allow for compact control while at the same time displaying the current status of the device, as well as the relevant information from the MySQL database, a different approach was used. The screen is labeled as "*Relay Module Details*" and contains a text field, which informs the user whether the relay is currently on or

off, followed by a set of toggle buttons. These buttons are either toggled to positive or negative, depending on the current status of the corresponding device. For example, if the SCM device that controls the lights in a room has currently turned on the relay it controls, the "*Relay*" button would be switched to positive. In the same way, if the main control program is allowed to switch the state of the relay during the decision phase, then the "*Control*" button would be also enabled, reflecting the control flag being set in the MySQL database. The last button is used to select the heat or cool states of the air conditioning and therefore is unavailable and greyed out in the "simple" relays, such as the light controls. Whenever the "*Relay*" button is pressed, it will change state and go to the negative, if it was positive, and vice versa. Upon a state change, a request will be sent to the gateway, which will be served, as discussed previously, and the corresponding relay will be turned on or off. It is worth mentioning that if a user tries to turn on the air conditioning in a room that has any open windows, then the main control algorithm will prevent this from happening and inform the user via email of the problem it has encountered. Other than that, the air conditioning will be switched on in either heat or cold, according to the "*Mode*" toggle button, and will be regulated normally from the main control program the next time any new measurements are requested. The user may prevent the main control program to alter the state of any relay by disabling the "*Control*" button which, when pressed, will prompt the application to change the control flag status of the corresponding relay in the MySQL database. This way, the control program will not retrieve any information about the SCM devices that it is not allowed to control during "*SELECT*" queries. An example of the aforementioned screen is shown in **Figure 8.8** for the "*lights*" type control SCM device, while an example of the "*aircond*" type SCM device is shown in **Figure 8.9**.

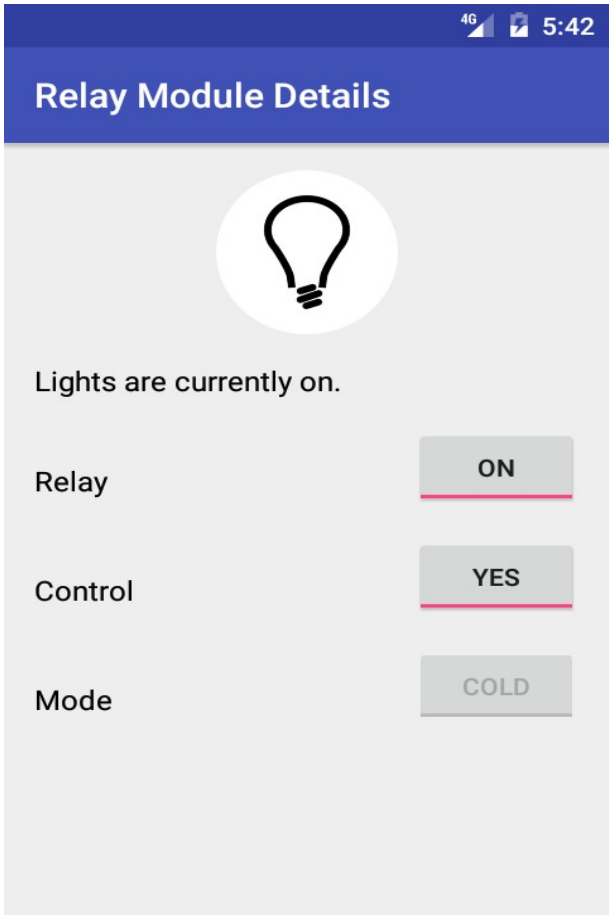

Figure 8.8: Android Application Relay Module Details Screen - Lights Control

The SCM device control and data exchange is performed by several pieces of software that include the Android application, which is written in JAVA, the PHP scripts running on the server, which are responsible for receiving information via POST from the application and serving content to the user, and the C socket server program that is running on the gateway. The Android application uses the "*HttpCookie*" class to store the session cookie generated by the server upon a successful login. It also uses the "*HttpURLConnection*" class and its more secure equivalent, the "*HttpsURLConnection*" class, for communicating with the PHP scripts on the server and sending POST parameters, as well as the "*URL*" class that is responsible for "understanding" IP addresses and parsing the URLs that point to the scripts on the server. The server replies with data formatted as a JSON array object, by using the PHP function "*json_encode()*", therefore the classes "*JSONArray*", "*JSONObject*" and "*JSONException*" were also used to declare JSON objects in the JAVA code and parsing the JSON text. On the

server's side, the JSON response contains two main fields, one that contains an error flag and one that contains a message from the server. When the application receives the response, it initially checks the error flag. If it is set, then it simply prints the received message from the JSON to the user, using the "*Toast*" mechanism of the Android. If the error flag is not set, then the message printed is "*Success*" and the rest of the JSON is parsed as planned. After posting an HTTP request to the server, the application will wait for a maximum of 5 seconds before it terminates with a connectivity error message. This mechanism was implemented in order to prevent the application from waiting indefinitely for a reply from a server or gateway that is down, or from a wrong IP address, and hanging the user interface.

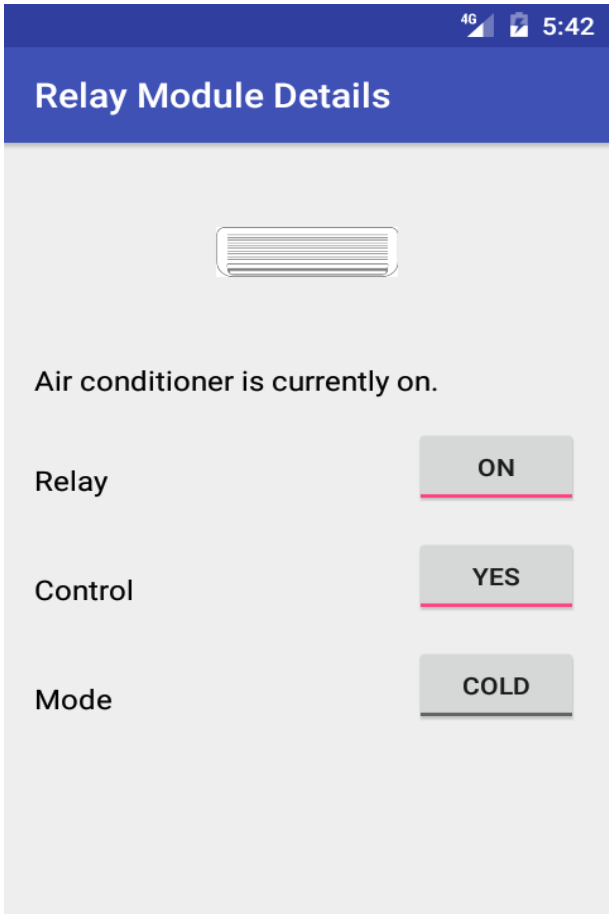

Figure 8.9: Android Application Relay Module Details - Air Condition Control

Chapter 9.

System Security

Security is one of the major aspects of any sensor network, especially those of the wireless type. A large number of attacks has been developed lately that is specifically aimed towards exploiting the weaknesses of wireless networks, such as radio jamming, packet injection or packet capture. In this chapter the possible wireless attacks are explained and their proposed solutions are presented.

Creating a secure way of wireless communication with low packet loss probability is challenging and has been studied for years. The use of symmetric or asymmetric cryptography can be thought as a solution, but the problem of key distribution requires an already secure channel of communication, which cannot be guaranteed at system start up. Asymmetric cryptography eliminates the need of a common secret to be transmitted for key agreement in the system, but this requires more expensive hardware and it is susceptible to man-in-the-middle attacks, threatening the confidentiality and integrity of communications[2].

The wireless nature of the proposed system does reduce the overall cost of installation and increases the flexibility in servicing and adding new SCM devices to the network, but it also requires a high degree of security in order to guarantee the privacy of the users. Limitations, such as the available CPU and memory of the used MCUs, prevented the implementation of complex encryption algorithms, or the storage of long ciphers. One possible workaround would be to generate the ciphers online, or from a PC, and then use them to encrypt all RF traffic. This, however, cannot provide for much security, as the maximum data size in bytes that can be transmitted for each packet by the RFM22B is 64 B. This means that the encryption would be weak, or would require a series of packets to be sent, each carrying only fragments of the data, which would increase the power consumption and the transmissions, so it generally is not worth the extra effort.

Packet capture means that an unauthorized party can capture a network packet through the air and use it to gain intelligence about the system, compromising the security of the whole network. Attacks like this one are extremely dangerous, since the attacker could

potentially gain knowledge of credentials, such as passwords and usernames, or even deduce information about whether there are users in the smart home currently or not. In the proposed system, the first case is dealt with by not transmitting any sensitive data over RF. This means that, even without encrypting the RF data, any leaks would not provide any critical information to unauthorized parties that could be used in order to tamper with the rest of the system. Secondly, all packets that come from sensor-type SCM devices and provide information regarding the motion, the lightning status and the temperature, transmit this information in "raw" bytes. Therefore, the intruder who manages to capture a packet and can read it, cannot draw any conclusions about the nature of information each of the bytes represents, or how it is used by the gateway (AP), In addition to the above, the system communicates over RF in the 868 MHz band and not in the IEEE 802.11 2.4 GHz band. One major issue of the 2.4 GHz band is that the antennae of every-day devices, such as smartphones and laptop computers are able to communicate in that band. This means that an attacker could easily capture the network's packets by setting the antenna of their device to passive and eavesdrop the network traffic, a process called "sniffing". On the contrary, the 868 MHz band requires other equipment, such as transceiver devices with antennae tuned in that specific frequency band. Therefore, the RF communications are safe from eavesdropping.

One other threat comes from injecting rogue packets in the network, or "simulating" a device in the network by stealing its identity and providing false data to the rest of the connected devices. Protection against packet injections is not simple, since the attacker could potentially capture a wireless packet and retransmit it at another time, potentially damaging the network. In the proposed system, such attacks cannot damage the system in any way, since the only packets that are transmitted by RF and can affect the system are RELAY-type packets. The only issue from a rogue packet, injected in the rest of the traffic, would be a relay changing its state, then transmitting an acknowledgement, which the gateway would see as invalid action and inform the user via email.

The only potentially dangerous attack would be the signal jamming method. During an attack like this, the attacker floods the air with RF noise that prevents the packets from being communicated successfully, as they are altered by noise. Such an attack would make the network unusable and would isolate all of the SCM devices and the gateway. This attack is the

hardest to predict and counter, since it virtually disables the communications between the network's nodes. In the proposed system, such attack would disable communications between the gateway and the SCM devices and as soon as the jamming signal was removed, then communications would return to normal, without further disruption to the system. Another possible way to counter such an attack would be to change the frequencies of the sensors in a predictable manner and inform the user. However, this method requires more advanced hardware and was not used in the present work.

The other aspect of the security of the system involves the internet security, since the users communicate with the gateway over IPV4. The BBB is connected via an Ethernet cable to the user's router, which provides access to anyone who knows the SSH credentials of the BBB. Unauthorized access to the BBB would mean that the system's control would be compromised, as the attacker would be able to turn off the alarm by terminating the main control process. The SSH protocol is secure enough, therefore the only precaution taken against such attacks, was to change the default usernames and passwords of all the accounts on the BBB with more secure ones.

Apart from the connection to the gateway over SSH, there is another way of connecting to the system, which is implemented on the Android program and the host website. Access to either one of them by an unauthorized user would mean a breach of confidentiality and privacy, as the system's data, the measurements and the current status of each sensor would be visible to the attacker. In order to avoid that, the proposed work uses PHP functions to create and store hashed passwords to the MySQL database. In more detail, the username and password, which can be changed by the user through the website version of the control interface, hosted on the BBB, are stored in the MySQL database, with the password hashed by using the BLOWFISH algorithm. Then, whenever a user wishes to log in to the system, they provide a username and a password. The password is hashed and then subsequently checked against the hash stored in the database. If they are a match, then the user logs in successfully. This provides an extra layer of security in the credentials of the system. Furthermore, the android application that allows the users to connect and control the system by transmitting HTTP data, requires from the users to know the IP of the gateway as well as the system's credentials.

Finally, the system uses PHP sessions in order to identify already logged in users, eliminating the need for constant exchange of credentials over possibly insecure networks. After a user logs in to the control interface from either their smartphone or their PC, a unique session file is created. This file will expire after 30 minutes of inactivity, as a precaution against session hijacking attacks. During an attack like this, an unauthorized user can"hijack" the session cookie, which is exchanged between the client and the server during an HTTP transaction and can use it to trick the server into leaking information, by providing the same credentials as the original user. By expiring the session cookies, this risk is minimized. The session timeout mechanism is implemented using a PHP script and uses a timer to check the inactivity time of the user. When this time exceeds the 1800 seconds, or equivalently, 30 minutes, then the session is terminated and the user is logged out of the system and returned back to the index-login page. This ensures the security of the system, as it makes it more difficult for an attacker to eavesdrop the session cookie and use it to gain access to the system. The session timeout mechanism is only implemented when the user is logged in via the website, which is accessible via any web-browser, and not implemented when the Android application requests or sends information, as it would be inconvenient for the users having to log in after 30 minutes of inactivity. A compromise between security and convenience was made in order to provide a user-friendly control interface to the user.

Chapter 10.

Implementation

The proposed system has not only been designed, but it has also been implemented. A fully-operational remote control smart home system was created, able to control automatically any connected devices and regulate the inner temperature of the home as expected. Two photos of the implementation follow.

Figure 10.1: Implementation - Top-down View

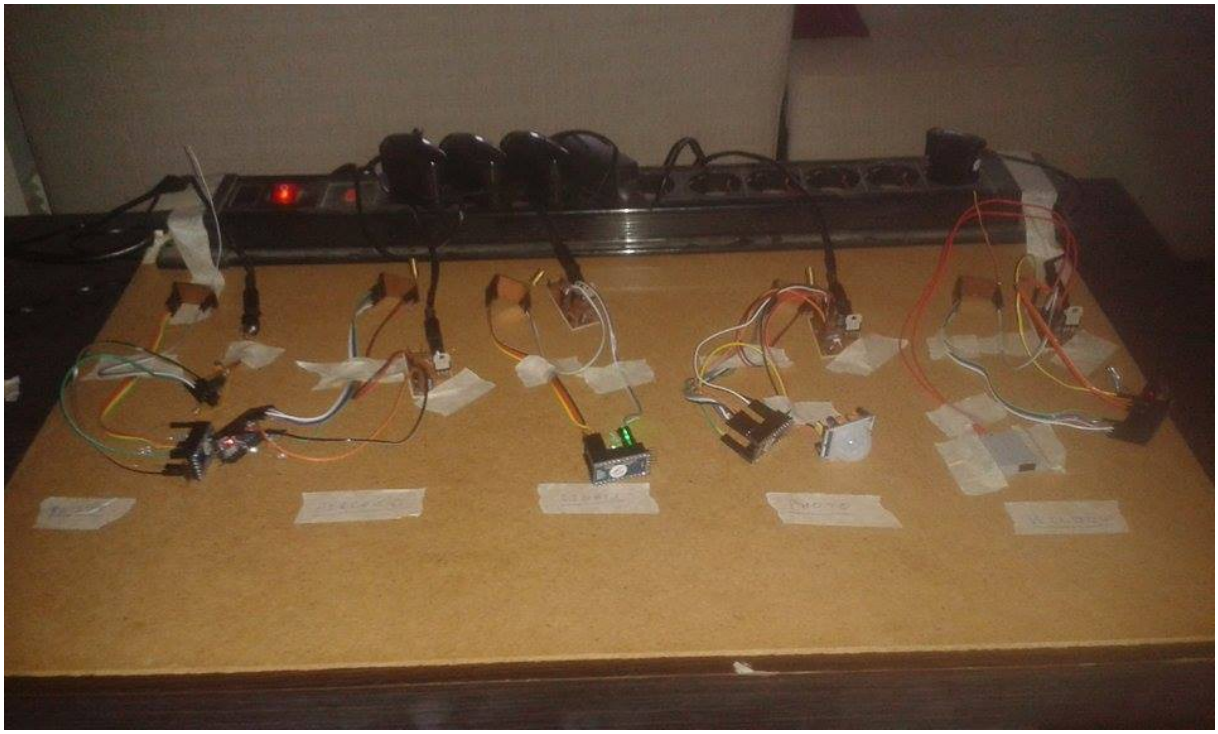

Figure 10.2: Implementation - Side View

In more detail, during the testing phase the BBB serving as the gateway was connected to an internet router via Ethernet cable, while one of each of the SCM devices was powered by a 5 V power supply. Initially, the SCM devices were not synchronized with the gateway and they were placed in the same room as the gateway. The synchronization algorithm worked flawlessly and the MySQL database was quickly populated with the correct data from the SCM devices. No problems or errors were detected and any collisions were solved without problems. As soon as the synchronization process was over, the main program was manually initialized. Two more shell windows were also open during the main program's execution, each running a separate tool. One showed in real-time the available resources of the BBB using the "*top*" resource monitor and the other was used to check whether the system journal was being populated by SYSLOG. The second shell window was running the command "*journalctl -f*" in order to view the system logs in real-time.

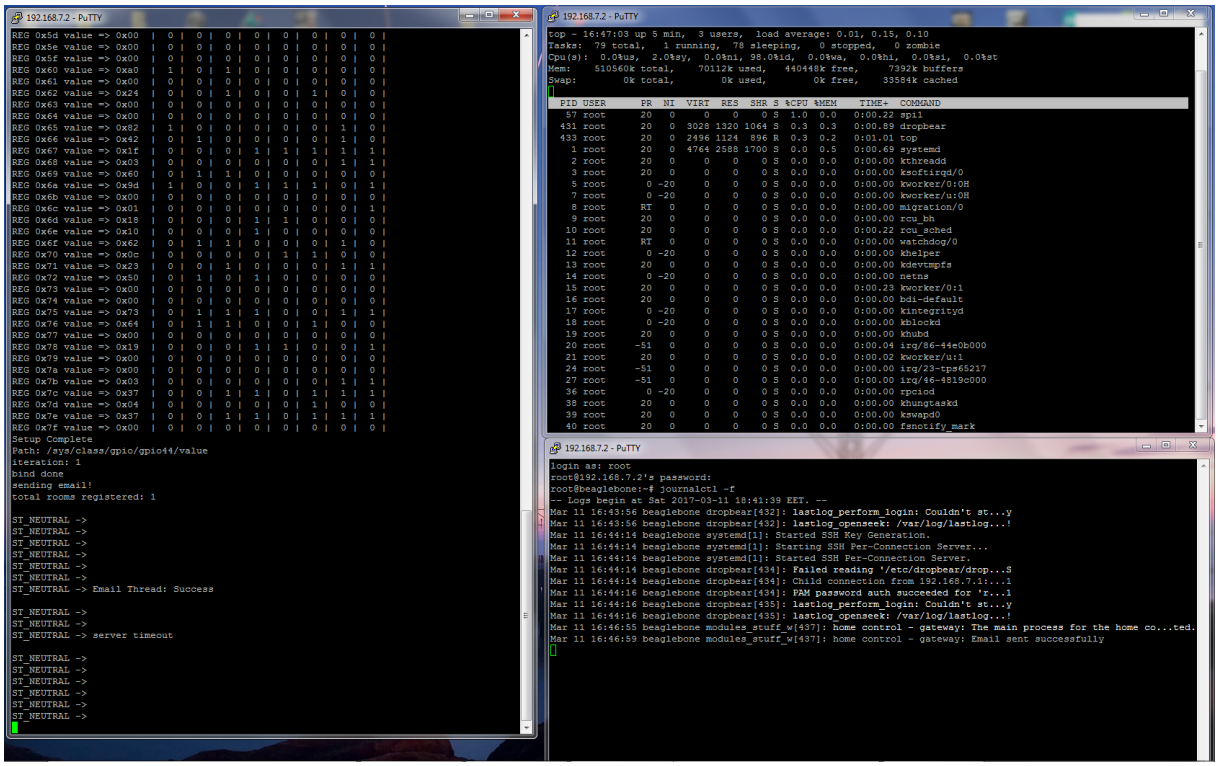

Figure 10.3: Main Program During Startup

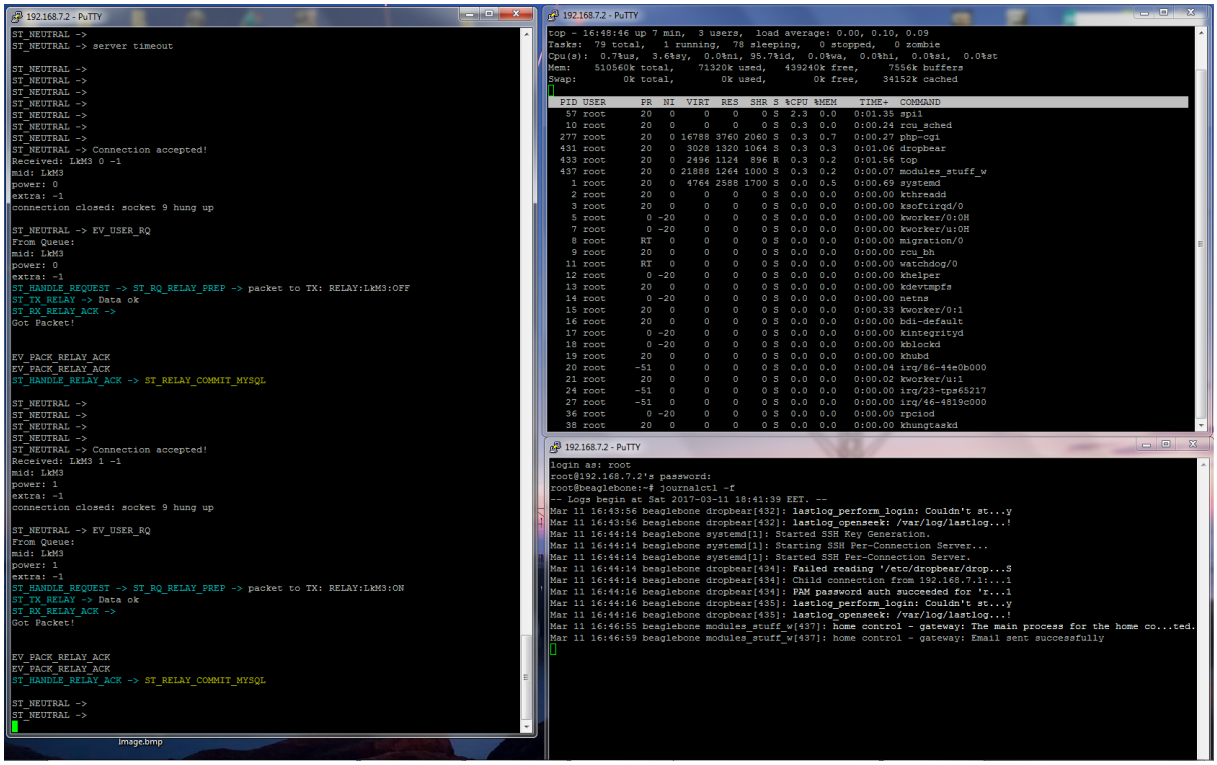

Figure 10.4: Main Program While Handling A User Request

The main process was run from a shell window in order to view its output to the two standard output streams, the "*stdout*" and the "*stderr*". The output of the program is the FSM state changes, as well as any errors that might occur during run-time. The measurements timeout was also set to 180 seconds, instead of 1800 seconds, in order to skip the long waiting times for the timer to expire and test the measurements procedure efficiently. During the testing process, no errors were noted. The FSM transitioned between states as expected, all events triggered as planned and the MySQL database accesses were executed successfully and the relevant information from the program was stored successfully to the system logs. All threads of the main process were executed successfully and the server thread did push the user requests to the queue for the main thread to handle. Measurements were also received as planned and the alarm functionality did not fail to detect intrusions. There were no packets dropped during this test. The Android application was able to control the system efficiently, turning devices on/off and updating the data in the MySQL database. Finally, the temperature and lightning regulation algorithms functioned as expected, making the optimal decision during each measurements iteration.

Another test was performed with the SCM devices scattered throughout the rooms of the home. There was no direct line-of-sight with the gateway, as they were separated by cement walls. During this test, the SCM devices were left to synchronize again with the gateway and as soon as the synchronization process was over, the main program was initialized manually, as before. Again, no errors were found as the synchronization was performed correctly and the MySQL database was quickly populated with the correct data from the SCM devices. Afterwards, the main process was initialized manually and the same results were drawn from its execution. The state transitions of the FSM were correct and fast, events triggered accordingly and the MySQL database accesses were successful. Measurements were received and handled again with no errors and the decision algorithms functioned correctly, reaching the optimal conclusion each time. System logs were stored as well and all the threads of the main program worked as planned. The Android application was able to control the system efficiently, turning devices on/off and updating the data in the MySQL database. The separating walls did not affect the overall performance of the system in any way, mainly due to the small coil antennae used by the RFM22B radios. The antennae provided extended range and reliability, allowing the system to function efficiently even when the SCM devices

were scattered around the house with no direct line-of-sight to the gateway.

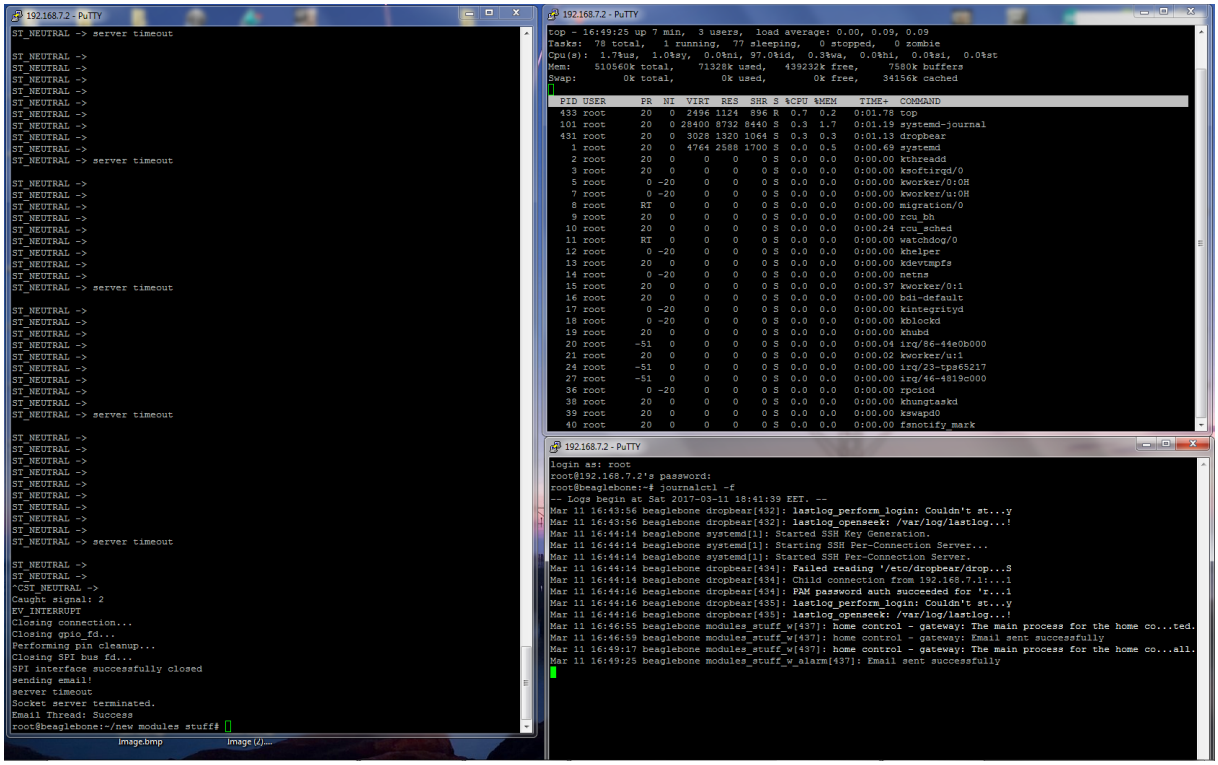

Figure 10.5: Main Program After Capturing Termination Signal - Normal Exit

Chapter 11.

Conclusions

Combining a number of different hardware components and developing software to drive the hardware was the main implementation difficulty of the thesis. The hardware chosen was sufficiently low-cost for the provided features and potential, while the developed software succeeds in providing reliable RF communication between the hardware components and devices, as well as autonomously controlling the connected devices and reducing any wasted energy from their operation.

A powerful development platform, such as the BBB, paired with a reliable integrated RF radio chipset, such as the RFM22B, was the optimal solution in creating an AP with optimal uptime and multiple user interfaces. Communication over SPI is not easy on the BBB without the proper software drivers and therefore needed to be simplified by developing the driver software. The result produced was successfully turning the BBB into the master SPI device which controls the RFM22B as a slave chip. The BBB is also capable of functioning as a server with little impact on the consumed resources, using the "*lighttpd*" and PHP software. Hosting a number of different websites and PHP scripts allowed for a lot of flexibility when presenting the system's information to the user. As a result, the rest of the developed software is not hindered performance-wise by the server software. Speed and reliability in both RF and IP communications, combined with adequate security and adequate data storage capabilities are the main benefits of such an AP in a wireless network of sensors.

Fully utilizing the Arduino platform helped significantly during the design and creation of the miscellaneous SCM devices used inside the smart home. Low-cost, open-source and open-hardware solution, combined with its small size and its low operating voltage at 3.3 V made the APM the perfect MCU choice for the developed system. Each of the designed devices serves a specific and unique purpose in the smart home. From monitoring the status of doors and windows to controlling relays wirelessly and sensing the environment, the devices fulfill their purposes far better than expected. The developed software was based on a few preexisting libraries, while others were designed specifically for the purpose of this

thesis to allow for more selectivity in each device's functionality. Controlling the RFM22B radio chipset over SPI was straightforward and simple. Only a few Arduino functions were used, while the specific driver for the RFM22B needed to be coded, just as with the BBB, making the APM the master device that controls the RFM22B as a slave device.

Every SCM device created was a combination of an APM device and other pieces of hardware, such as sensors and relays. These devices had different power requirements and operated in voltages that ranged from lower than 3.3 V to over 5 V. Therefore, it was necessary to design proper power adapters for the SCM devices, as well as design and print specialized PCBs for each of them. The design of the PCBs was performed in CadSoft Eagle. Each PCB is required to be powered by at least 5 V. The input voltage is split and used to power separately all the hardware components, from the APM to the RFM22B radio and the miscellaneous relays and sensors. Since the relays were solid-state devices that operated at 5 V, in order for the APM to interface with them a voltage level shifter was also used. The resulting SCM devices are able to reliably control relays connected to AC devices and provide measurements to the AP from their environment regarding motion, ambient lightning and temperature while keeping their power consumption remarkably low.

Networking is also a critical part in the proposed system. Each of the SCM devices communicates directly with the AP, as intra-device communication is not allowed. The AP controls directly all of the SCM devices by transmitting special RF packets that inform the receiver device of what the AP requires of them. This is called a star network topology and the AP is referred to as the critical point of the network as any malfunctions on the AP will result in the whole network being compromised. On the positive side, this topology allows for flexibility during installation and ease of use for the users, as adding new SCM devices or removing others is extremely easy. Therefore, the user can create their own layout topology of sensors, organizing them as they see fit. This is one of the most important features of the system because, as a result, it can be installed in any home and transform it easily into a smart home.

All information regarding the smart home is stored in a MySQL database. This allows for easy access and presentation from both the designed web site, as well as the Android application and the main control program of the smart home. MySQL is freely available and

open source, making it ideal for a system that strives to be as low-cost as possible. It is also accompanied by libraries for both the C programming language as well as the PHP programming language. The developed database consists of a number of tables that store information about the smart home's available rooms, sensors, relays and the measurements over the past 12 hours. Users can create and name their own rooms from the Android application, tailoring the system to their own personal preference.

The Android application serves as the remote control for the whole smart home. Through the developed application, the user has full control over how the house handles the installed devices and can enable or disable control for any of them, allowing or preventing the main control program from interfacing with them automatically. Users can also send requests via the application to the AP in order to power on or off the connected household appliances, lights and the air conditioning system. In order for the application to be able to connect and communicate with the AP, only the gateway's IP address is required. The network IP address can be used while the user remains connected to the smart home's local network, while the public IP is required for access from outside the smart home's local network.

Guaranteeing the security of the developed system was also of utmost importance and as such, security measures were implemented. Connecting to the system can only be achieved by inserting the correct username and password credentials from both the developed website as well as the Android application. These credentials can be changed anytime but only from a special menu on the website. Furthermore, secure communications between the RFM22B radios is achieved, while not necessary, by transmitting raw bytes over the ISM band. As a result, the wireless communications are not performed in the WiFi band and intruders cannot sniff information from their smartphone and laptop devices, which are two of the most common hacking methods. Finally, a security system for the smart home itself was implemented. An alarm function was developed that, when enabled, will inform the user if motion is detected or a door or window opens. It can be enabled or disabled from the Android application manually by the user.

Notifications are sent to the user in the form of email messages. This method is preferred because an email message can be read from any device. Smartphones are also using applications to notify the user of new email messages, which means that notifications are

read instantaneously by the user. However, if the user's smartphone has ran out of battery or is damaged or generally inaccessible, then the notifications can be read from a computer. If an alarm is triggered, or a system problem occurs, the user will still be able to get informed on time and take any appropriate actions.

Regulation of the inner temperature of the house is automated by the main control program developed for the BBB. Every 30 minutes the AP receives measurements from the sensors around the smart home regarding the inner temperature, as well as the outer temperature from a separate sensor, the lightning levels and the motion detector sensors. Based on the received data, the control program makes smart decisions in order to regulate the inner temperature of each of the rooms and control the lightning conditions. The control algorithm is performing exceptionally well in reducing the power consumption in each of the rooms. This is achieved by turning off the lights and the air condition that are powered on in every room that is empty. It also prevents the air condition from being turned on by the user or the control algorithm if any windows are open in the room.

During the current thesis, a low-cost, multi-operational, autonomous control system was created that is capable of being installed in every type of home in a non-invasive way and monitor the home, effectively transforming it to an IoT smart home. At the same time, the developed system monitors the interior of the house, as well as the connected household appliances, and also sends feedback to the user such as proposed actions to further decrease the unnecessary power consumption. It also allows for reliable control of physically any device in the house from the user's smartphone by making full use of the Android platform. Furthermore, it includes a security and alarm feature for the safety of the user. All the above are included in a very cost-effective package, as the developed system has no initial installation costs and its overall cost is only a fraction of similar BAS solutions currently in the market. In addition, the wireless nature of the developed system allows it to be installed easily by anyone. The many available features combined with its low-cost make the developed system superior to the current alternatives.

In the future, actuators and motors can be also included, with a proportional increase of the system's cost. The actuators and the motors would be installed in the windows of the smart home and allow the control algorithm to not only inform the user via email, but to act

autonomously by opening and closing the windows, lowering the shades, and locking the doors with no user intervention. The alarm system can be also upgraded to allow for making calls to law enforcement or the user when a break-in is detected. A sound alarm can be also installed to deter possible intruders. Finally, the Android application can be improved by adding voice control, which sadly was outside the scope of the current work.

Bibliography

[1] adafruit, Adafruit PIR Sensor Tutorial, found at https://learn.adafruit.com/pir-passive-infrared-proximity-motion-sensor/

[2] Algorithms for Sensor and Ad Hoc Networks, by D. Wagner, R. Wattenhofer, 2007

[3] Amlogic, Amlogic S905 ODROID-C2 Online Overview, found at http://www.hardkernel.com/main/products/prdt_info.php

[4] Google, Android Development Community

[5] Arduino, Arduino Pro Mini Board Information, found at https://www.arduino.cc/en/Main/arduinoBoardProMini

[6] Atmel, ATmega328 Datasheet

[7] Home Automation Community, ATmega328 Low Power Modifications, found at http://www.home-automation-community.com/arduino-low-power-how-to-run-atmega328p-for-a-year-on-coin-cell-battery/

[8] BeagleBoard, BeagleBone Black Official Website & Community, found at http://beagleboard.org/bone

[9] G. Coley, BeagleBone Black System Reference Manual

[10] element14, BeagleBone Web Server Using Lighttpd & MySQL, found at https://www.element14.com/community/community/designcenter/single-board-computers/next-gen_beaglebone/blog/2013/11/23/beaglebone-web-server--setup

[11] British Petroleum, BP Statistical Review of World Energy, 2016

[12] Robert Ingalls, C Sockets Server and Client, found at http://www.cs.rpi.edu/~moorthy/Courses/os98/Pgms/socket.html

[13] Database Management Systems, Third Edition, by Ramakrishnan, Gehrke, 2002

[14] Derek Molloy, Derek Molloy's BeagleBone Online Resources - Blog, found at http://derekmolloy.ie/beaglebone/

[15] Electric Measurements and Sensors - Operation Principles and Electronic Measurement Systems Design, by K. Kalaitzakis, E. Koutroulis, 2010

[16] Global Energy StatisticalYearbook 2016, Electricity domestic consumption, found at https://yearbook.enerdata.net/electricity-domestic-consumption-data-by-region.html

[17] Electronic Measurement Systems - Practical Implementation of Analogue and Digital Techniques, by T. T. Lang, S. Lang, 1987

[18] Wiki, Embedded Linux Wiki, found at http://elinux.org

[19] Exploring BeagleBone: Tools and Techniques for Building with Embedded Linux, by Derek Molloy, 2014

[20] Future Technology Devices International Ltd, FTDI Chip FT232R To USB UART IC, found at http://www.ftdichip.com/Products/ICs/FT232R.htm

[21] Free Software Foundation, GNU Operating System, found at https://www.gnu.org/

[22] Google, Google Charts Javascript Libraries, found at https://developers.google.com/chart/

[23] Brian Hall, Guide to Network Programming Using Internet Sockets, found at http://beej.us/guide/bgnet/output/html/singlepage/bgnet.html

[24] Instrumentation and Control, by C. Nachtigal, 1990

[25] Texas Instruments, ISM-Band and Short Range Device Regulatory Compliance Overview, found at http://www.ti.com/lit/an/swra048/swra048.pdf

[26] KNX, KNX Building Control System, found at http://www2.schneider-electric.com/sites/corporate/en/products-services/product-launch/knx/knx.page

[27] STMicroelectronics, Low Drop Fixed and Adjustable Positive Voltage Regulators Datasheet

[28] Microelectronic Circuits, by K. C. Smith, A. Sedra, 1982

[29] Network Security Essentials: Applications & Standards, 5th Edition, by W. Stallings, 2013

[30] The ITU-T, Overview of the Internet of Things, found at https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-Y.2060-201206-I!!PDF-E&type=items

[31] PINE64, PINE64 Online Resources, found at https://www.pine64.org/

[32] Parallax, Inc., PIR Sensor Datasheet

[33] Raspberry, Raspberry Pi 3 Model B Online Overview, found at https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

[34] SparkFun, RFM22B Breakout Board Schematics and Layout, found at https://www.sparkfun.com/products/retired/10154

[35] Hope RF Microelectronics CO.,LTD, RFM22B/23B ISM Transceiver Module Datasheet

[36] Sensors for Measurement and Control, by P. Elgar, 2000

[37] Silicon Labs, Si4430/31/32 Register Descriptions Datasheet

[38] SparkFun, SparkFun Libraries for CadSoft EAGLE, found at https://learn.sparkfun.com/tutorials/how-to-install-and-setup-eagle/using-the-sparkfun-libraries

[39] Arduino, The Arduino Forum, found at https://forum.arduino.cc/

[40] Central Intelligence Agency , The World Factbook, 2017

[41] UDOO, UDOO Online Resources, found at http://www.udoo.org/

[42] Digi International, ZIGBEE Wireless Standard & Protocol, found at https://www.digi.com/resources/standards-and-technologies/rfmodems/zigbee-wireless-standard