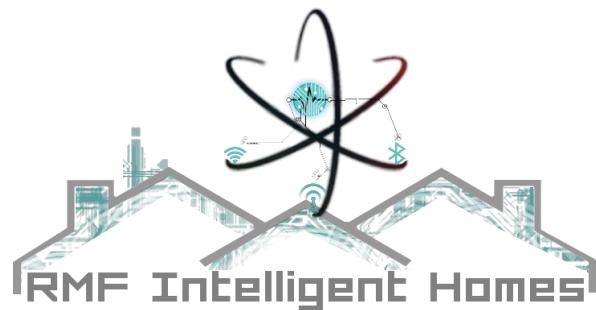


Polytechnic University of Puerto Rico
Electrical & Computer Engineering and Computer Science Department
SPRING 2015 – EE 4002 Capstone Design Course I
FALL 2015 – EE 4022 Capstone Design Course II



Francisco L. Burgos Collazo	#75483	Sec. 22
Manuel E Santiago Laboy	#82577	Sec. 22
Rafael Ruiz Villalobos	#82756	Sec. 26
Bryan Maldonado Rodriguez	#73331	Sec. 49
Paolo Jimenez Soto	#63880	Sec. 26

Contents

Contents	2
Chapter 1: Abstract.....	14
Chapter 2 About RMF Intelligent Home	16
Introduction	17
2.1 What is RMF Intelligent Home?.....	17
2.2 Vision.....	17
2.3 Mission	18
2.4 Work Philosophy.....	18
2.5 Goals	18
2.6 Methods.....	19
2.7 Services.....	20
Chapter 3: Realistic Constraints	22
3.1 Resource Based Constraints	22
3.2 Time Based Constraints	23
3.3 Activity Based Constraints	23
3.4 Final Product Based Constraints	24
3.5 Environmental Based Constraints	24
3.6 Social Based Constraints	24

3.7 Reliability Based Constraints	25
Chapter 4: Multidisciplinary Aspects	26
4.1 Software Development	26
4.2 Electronics, Power Distribution, Controls & Communications.....	26
4.3 Mechanical	27
Chapter 5: Sensors and Microprocessors.....	28
5.1 System Overview.....	28
5.2 Infrared transmitter Diode	30
5.2.1 Infrared Emitting Diode (IR333-A) Specifications	30
5.3 Infrared Receiver Diode.....	32
5.3.1 Infrared Receiver Diode (Tsopp4p) Specifications	32
5.3.2 Description	33
5.3.3 Application.....	33
5.4 Relay Shield V2	34
5.4.1 Relay Shield (RB-see-254) Specifications.....	35
5.4.2 Description	37
5.5 Magnetic Switch	37
5.5.1 Magnetic Switch Sensor Specifications.....	38
5.5.2 Description	38
5.5.3 Application.....	39

5.6 Infra-Red Night Vision	39
5.6.1 Infrared Night Vision Specifications.....	40
5.6.2 Description	40
5.7 PIR Motion Detector Sensor (HC-SR501)	41
5.7.1 PIR Sensor (HC-SR501) Specifications	41
5.7.2 Description	43
5.7.3 Adjustments	43
5.8 Shock Sensor Module	44
5.8.1 Shock Sensor Module (801S) Specifications	45
5.8.2 Applications.....	46
5.9 Current Sensor ACS712 - Invasive.....	48
5.9.1 Current Sensor Invasive (ACS712) Specifications	48
5.9.2 Description	50
5.9.3 Sensitivity and output of ACS712	52
5.10 Raspberry Pi Camera Module	53
5.10.1 Raspberry Pi 5MP Camera Board Module	53
5.10.2 Description	54
5.11 Microcontroller Analysis.....	55
5.11.1 Microcontroller Specifications.....	55
5.11.2 Arduino Platform.....	57

5.11.3 BeagleBone Black Platform.....	60
5.11.4 Raspberry Pi 2 Platform	63
Chapter 6: Lighting System	66
6.1 Alternatives Considered	66
6.2 System Specifications	67
6.3 Economic Analysis	71
6.4 Results	72
6.4.1 Lighting System Code for Arduino.....	72
Chapter 7: IR System	82
7.1 System Description.....	82
7.1.1 NEC Protocol	82
7.1.2 TV Controller Introduction	85
7.1.3 IR TV Controller Diagram	87
7.1.4 Observing the IR TV Transmitter Code with a Serial Monitor.....	89
7.2 System Specifications	90
7.3 Economic Analysis	91
7.4 Results	92
7.4.1 Receiving Code for Arduino	93
7.4.2 System Code for Arduino	96
Chapter 8: Energy Monitor.....	165

8.1 Alternatives Considered	165
8.2 Mathematical Justification	165
8.3 System Specifications	166
8.4 Economic Analysis	172
8.5 Results	173
8.5.3. Energy Monitor Code for Arduino	175
Chapter 9: Surveillance	185
9.1 Alternatives Considered	185
9.1.1 Raspberry Pi	185
9.1.2 Raspberry Pi NOIR camera module	186
9.1.3 Arduino uno	187
9.1.4 Ardu cam	188
9.1.5 Logitech Webcam	189
9.2 System Specifications	190
9.2.1 Hardware	190
9.3 Economic Analysis	193
9.4 Results	195
9.4.1 Software and Configuration	195
9.4.2 Materials	197
9.4.3 Assembly	198

Chapter 10: Alarm.....	200
10.1 Alternatives Considered	200
10.2 System Specifications	203
10.2.1 Alarm System Functions and Components	203
10.3 Economic Analysis	206
10.4 Results	207
10.4.1 Alarm System Wiring.....	207
10.4.2 Alarm System Code	208
10.4.3 Testing the socket	216
Chapter 11: Verification System	1
11.1 Alternatives Considered	1
11.1.1 RFID (Radio Frequency Identification)	1
11.1.2 NFC (Near Field Communication)	2
11.1.3 NFC vs RFID	8
11.2 System Specifications	9
11.2.1 Verification System Components	9
11.3 Economic Analysis	16
11.4 Results	18
11.4.1 Verification System Implementation	18
11.4.2 NFC Verification System Connections	20

11.4.3 NFC System Write to Mifare Classic Memory	22
11.4.4 NFC Verification System	50
Chapter 12: Network.....	83
12.1 Alternatives Considered	83
12.1.1 Internet of Things (IoT).....	83
12.1.2 OSI Model	85
12.1.3 TCP/IP.....	89
12.1.3 UDP.....	90
12.1.4 Static IP Address.....	91
12.1.5 Dynamic IP Address.....	91
12.1.6 MAC Address	92
12.1.7 Network Range.....	93
12.1.8 Mesh Network Topology.....	95
12.1.9 Star Network Topology.....	98
12.1.10 Ring Network Topology	99
12.1.11 Tree Network Topology	101
12.1.12 Bus Network Topology	103
12.1.13 Wi-Fi Connectivity	105
12.1.14 Bluetooth Connectivity.....	107
12.1.15 ZigBee Connectivity	109

12.1.16 LowPAN Connectivity	110
12.1.17 Sub-1GHz Connectivity	112
12.1.18 Ethernet Connectivity	113
12.1.19 Arduino Ethernet Shields.....	115
12.1.20 nRF24L01 Module.....	116
12.1.21 HC-05 Bluetooth Transceiver	119
12.1.22 XBee RF Module	120
12.1.23 Wireless Modules Comparison.....	124
12.2 System Specifications	126
12.3 Economic Analysis	127
12.4 Results	128
Chapter 13: Database.....	129
13.1 Alternatives Considered	129
13.1.1 Relational Database	129
13.1.2 Non-Relational Database	132
13.1.3 MySQL	132
13.2 System Specifications	133
13.2.1 DBMS Components.....	133
13.2.2 MySQL & Python.....	134
13.3 Economic Analysis	135

13.4 Results	135
13.4.1 Drive Formatting & Partitioning.....	135
13.4.2 Database Implementation	136
13.4.3 Database Management System	139
13.4.4 Database Management System Code.....	147
Chapter 14: Phone App	173
14.1 Alternatives Considered	173
14.1.1 Operating Systems.....	173
14.1.2 Android Operating System	174
14.1.3 IPhone/IPad Operating System (iOS).....	174
14.1.3 Programming Languages	175
14.1.4 Integrated Development Environment	176
14.2 System Specifications	177
14.2.1 System Requirements	177
14.2.2 The Application and the Android Operating System.....	178
14.2.3 Java.....	178
14.2.4 Android Studio.....	179
14.2.5 System Interfacing.....	179
14.3 Economic Analysis	180
14.4 Results	180

14.4.1 Application Interfaces	181
14.4.2 Living Room Example	181
14.4.3 Security System Interface	185
14.4.4 Power Consumption Interface	185
14.4.5 Application Class Logic	186
Chapter 15 Backup System	283
15.1 Battery Back-Up System	283
15.1.1 Introduction to the System.....	283
15.2 System Specifications	284
15.2.1 Battery Bank Size Calculation	284
15.2.2 Calculations for the Battery Back-Up System.....	286
15.2.3 Electric Design	289
15.3 Results	294
15.3.1 Backup System Tests.....	294
Chapter 16: RMF Intelligent Home System	296
16.1 Market Evaluation.....	296
16.1.1 ADT Security Services	296
16.1.2 Leviton Home Automation	297
16.1.3 RMF Intelligent Home System.....	298
16.2 Results	298

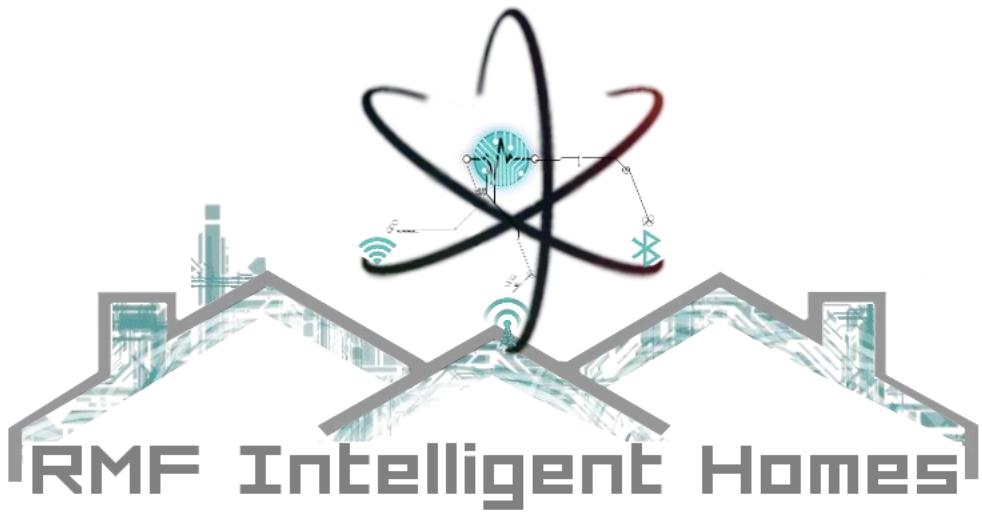
16.2.1 RMF Intelligent Home System Implementation	298
16.2.2 RMF Intelligent Home System Testing	299
Chapter 17: Presentation Concept Design	305
17.1 Showroom Design	305
17.1.1 Introduction to the Concept	305
17.1.2 Organization and Distribution	305
17.1.3 Construction Materials and Component Enclosures.....	310
Chapter 18: References.....	312
18.1 Internet of Things References	312
18.2 Phone Applications References.....	314
18.3 Databases & Cloud Computing References	316
18.4 Microcontroller Platforms References.....	317
18.5 Sensors References	317
18.6 Near Field Communications & RFID References	319
18.7 LED Illumination References	320
18.8 Direct Current Systems References	322
18.9 IR Controller Design References	324
18.10 Network Design References	324
Chapter 19 Budget.....	325
19.1 Sub-Systems Cost.....	326

19.2 RMF Intelligent Home Total Cost.....	330
Chapter 20 Schedule	332
Chapter 21 IEEE Article.....	333
Chapter 22 Meeting Minutes.....	334
Chapter 23 User Manual.....	335
Chapter 23: RMF Operation Manual.....	336
23.1 Phone App.....	336
23.1.1 Phone App User Verification	336
23.2 Alarm System	337
23.2.1 Verification System Operation.....	337
23.2.2 Alarm System Operation Through Phone App	339
23.2.3 Surveillance System.....	340
23.3 IR System	342
23.3.1 IR Control System Operation	342
23.4 Lighting System.....	343
23.4.1 Lighting System Operation	343
Chapter 24 Weekly Report.....	344

Chapter 1: Abstract

On this project the team will apply a series of innovative concepts, all of which are focused on the Internet of things and Home Automation. These concepts are implemented with the goal to create an intelligent house. Our goal is achieved with the help of microcontrollers and computers that maintain a strict communication between user and the house. This system will communicate such relevant data that is also able to provide security to the house with the security system that keeps the user informed at all times of the property status. All the user-house communication will be achieved by means of an Android phone app and an NFC verification system. The phone application presents features like remote control of common lighting systems and camera surveillance of the house. Other home automation technologies like TV and air conditioner control, energy monitor, and alarm system are integrated to provide more control of the house to the homeowner. The main purpose is to simplify all of these tasks to the owner, as well as to provide security to the house. All of these systems are integrated by means of the Internet, so it is an essential part of the project since it is the way for the house to communicate with everything else.

En nuestro Proyecto el equipo aplicara una serie de conceptos innovativos que se veran enfocados en lo que conocemos como “Internet of Things” y “Home Automation” o Automatizacion del Hogar. Como equipo nuestro objetivo es construir y establecer una comunicación estricta entre el Usuario y el Hogar con el uso de Computadoras y Micro-Controladores. Esta comunicación entre el Usuario y el Hogar se lograra a través de una aplicación diseñada para los sistemas Android y el Sistema de verificacion NFC. Esta aplicación le permite al Usuario la posibilidad de observar y controlar los sistemas de Iluminación y los sistemas de Seguridad del Hogar. En adición a estos sistemas previamente mencionados, el Usuario tendra control de las aplicaciones del Hogar como lo son la television y los aires acondicionados e incluso tendra acceso a la informacion de Consumo de Energía utilizando el “Energy Monitor” o Monitor de Energía. Nuestra meta principal es simplificarle las tareas del Hogar al Dueño a la misma vez que se brinda seguridad a la misma. Todos los sistemas se comunican utilizando el Internet, por lo que es esencial para el Proyecto que el mismo tenga acceso al Internet en todo momento.



Chapter 2 About RMF Intelligent Home

Introduction

2.1 What is RMF Intelligent Home?

RMF Intelligent Home was founded in February 2015 in San Juan, Puerto Rico to provide customers with maximum comfort and serenity at their homes. Our Intelligent home systems offer reliability and simplicity to the customers, which are looking for comfortable homes where they can enjoy with their family in a pleasant atmosphere.

Furthermore, homeowners are in the necessity of delivering their families a safe place to live. RMF Intelligent Home presents a way for our customers to live without insecurities, bringing them the peace of knowing their families are in good hands.

2.2 Vision

Provide a safe, comfortable and intelligent environment in all homes, enabling houses to connect with the world.

2.3 Mission

The mission of RMF Intelligent Homes is to connect people around the world with the gift of intelligence by giving each of their homes the capacity to create a world of possibilities. Carrying this responsibility with respect, quality, diligence, and ethical values.

2.4 Work Philosophy

At RMF Intelligent Home we are proud and responsible engineering designers that think out of the box. The innovative services we provide are carried out with diligence and thus focus on quality. Our engineers are committed to achieve our goals and ensure maximum customer satisfaction.

2.5 Goals

The IoT and Automation System will be able to:

- Allow the customers to enjoy in an intelligent house by using microcontrollers with the fundamental concepts of cloud computing, sensor reading and internet-connected microcontrollers.
- Provide houses with innovative network infrastructure that link physical and virtual objects using cloud computing, data capture and network communications.
- Create customizable profiles for each person living at the house with their preferred house atmosphere features.

- Provide customers with a touch screen having a graphic interface on which they can manually control the house, as well as read the data from all kind of sensors.
- Customers will be able to access online house information from remote areas just by connecting to the network using their phones.
- Create safe home environments with a security system that alerts about any burglary intents at the house, as well as providing online camera surveillance to the house.
- Regulate household energy consumption with a remote energy monitor that can inform about the misuse of electrical devices at home.
- Prevent that the system stops working because of electrical blackouts by using a battery backup system, and in case that the blackout lasts too long, the system is capable to shift to a complete manual operation.

2.6 Methods

In order to achieve the project goals it is necessary to develop methods to make the system work. These methods present an overview of how the system will interact with its parts, and how these parts will contribute to achieve the system's main goal. The following methods:

- Microcontrollers will be in charge of every aspect of the control system. The chosen platforms are the Raspberry Pi and Arduino.
- An alarm system will provide security to the house, preventing burglary and/or malicious threats to the homeowners.
- An NFC verification system will be used for the user verification whenever he desires to arm or disarm the alarm.

- Infrared LED's in combination with a microcontroller will control everything on the house that works with infrared settings like TVs, DVD players, and air conditioning consoles.
- A lighting control system will provide an interface for the user to control the lighting fixtures in the house.
- Night vision security cameras will give house-monitoring capabilities to the homeowner with a full live stream via Internet.
- Different programming languages will be used to manipulate the microcontrollers or as part of the interface, these include Java, Python, MySQL, PHP and C.
- The use of Internet in this project will enable the designer and the user to have an easy to use user-friendly environment.
- An Android Smartphone application will provide communication and operation of the house to each of the members.
- A database will store all the information from the house systems to make easier data management.
- A battery bank will be used to provide charge to the whole system when there are energy blackouts at the house.

2.7 Services

At RMF Intelligent Home we are proud, responsible engineering designers that think out of the box. The innovative services we provide are carried out with diligence and thus focus on quality. Our engineers are committed to achieve our goals and ensure maximum customer satisfaction.

RMF Intelligent Home takes pride on the quality of the services it offers. The value of our work comes from the dedication and diligence that is presented while performing our services. Among the services that RMF Intelligent Home offers are:

HOUSE CONTROL	HOME SECURITY
Individual Profiles	Burglary Detection
Create a profile for each person that lives at the house which are used for credentials verification.	Prevent Burglary with an alarm system that recognizes whenever an intruder is trying to enter the house without permit.
Remote Access	24/7 Surveillance
Control certain house appliances from remote areas without the necessity of being inside.	Provide 24/7 camera surveillance to the house, which activates recording when an intruder is detected.

USER IDENTIFICATION	ENERGY CONSUMPTION
Homeowner Recognition	Energy Consumption Monitor
With the use of an NFC tag, the house will be able to recognize the homeowner and change the alarm status accordingly.	Allow homeowners to view the energy consumption of their houses from remote locations, preventing the misuse of appliances at home.

Chapter 3: Realistic Constraints

Several limitations are taken in account on this project. The team decided that these limitations would be divided by categories. These categories will help cover each aspect that can cause deficiency on the project. Elements that will affect the scope of this project will include:

3.1 Resource Based Constraints

- GUI (Graphical User Interface) will only be implemented on the Android platform, so it will not work with any other OS.
- The only microcontroller platforms available for the team are Arduino, BeagleBone Black, and Raspberry Pi.
- The selected development platforms are good, but have a limited range of compatible devices.
- To focus on the project's concept, which is IoT (Internet of Things) applications, the team will use Integrated Circuit such as infrared receivers and transmitters relays, vibration sensors and other easy-to-interface devices.
- Money is one of the biggest constraints the RMF Intelligent Homes has and, with this in mind, many of the equipment to be used will not be most high quality product on the market. The team will make the most out of these devices and make them act as a high end device but please note that most of this equipment may not be the most efficient or best in the market.

3.2 Time Based Constraints

- Since the deadline of this project is approximately October 31, 2015, the team is using microcontrollers that one or more of the members have previous experience on.
- Because this project's nature is a concept, different clients will have different needs. Therefore only certain topics regarding comfort, and security problems will be addressed
- The team must apply various disciplines into this project. Therefore, for each team member to have the fullest experience, each member will work on a sub-system that relates to their field of study.
- Lack of knowledge in certain topics will be tended by doing only what has been given in previous courses in each member's field of study.

3.3 Activity Based Constraints

- The capstone course requires a prototype for the end of the first period of the course. A proper model must be used so the team is limited to a showroom scale model or a scale model of the prototype.
- There are some members of the team that live far away from the meeting places. The team will tend to this problem to maximize the work done by the team.
- Some team members were out of the country during the summer, which impacted workload production.

3.4 Final Product Based Constraints

- The final product design must be an efficient, affordable, and reliable product. Thus, applications that involve many WI-FI modems are not an option.
- The communication between devices will be implemented using the low consumption modules to avoid excessive energy consumption.

3.5 Environmental Based Constraints

- Carbon emissions released into the atmosphere because of the power consumption.
- This new concept of Internet of Things may replace many old devices, meaning that a lot of waste can result from the job.

3.6 Social Based Constraints

- There is a human resource cost to this project and it comes with automated services. With each new operator-less system that is implemented, there is a rise on the unemployment percentage. This rise can lead to serious economic failure since there will be less operator full jobs.
- Difficulties between team members lead to arguments and must be dealt by supporting each other.

3.7 Reliability Based Constraints

- The team also reviewed the possibility that this product can be hacked. Data leakage poses a major security threat and can even be used to harm the person from which the data is being stolen.
- A major reliability constraint will also be the quality of the product; many products that cover certain security risks such as data encryption or secure signal transferring. These options may not be in the scope of this project since the budget for the team is not very big.

Chapter 4: Multidisciplinary Aspects

4.1 Software Development

Software development is the base of the project. Almost all of the components require instructions to interact with the other components. These instructions are written in code into the microcontrollers and computers that compose the whole system. Every component has its own set of instructions that is coded using different programming languages. The main languages used for this purpose are C, Python, Java, PHP, and MySQL.

4.2 Electronics, Power Distribution, Controls & Communications

The main components of the system are microcontrollers and computers that are used to interface between the users and the end-point systems. These microcontrollers interact with other electrical components, which work as actuators or sensors. All of these components are based on digital electronics, which is the language that the microcontrollers can understand.

Some of these components are used to control other systems like the lighting system, and other components are used for communication like the NFC verification system.

All of these sub-systems need power to work, and there is where the distribution system is needed to supply enough energy for the whole system to work, even without direct power from the grid.

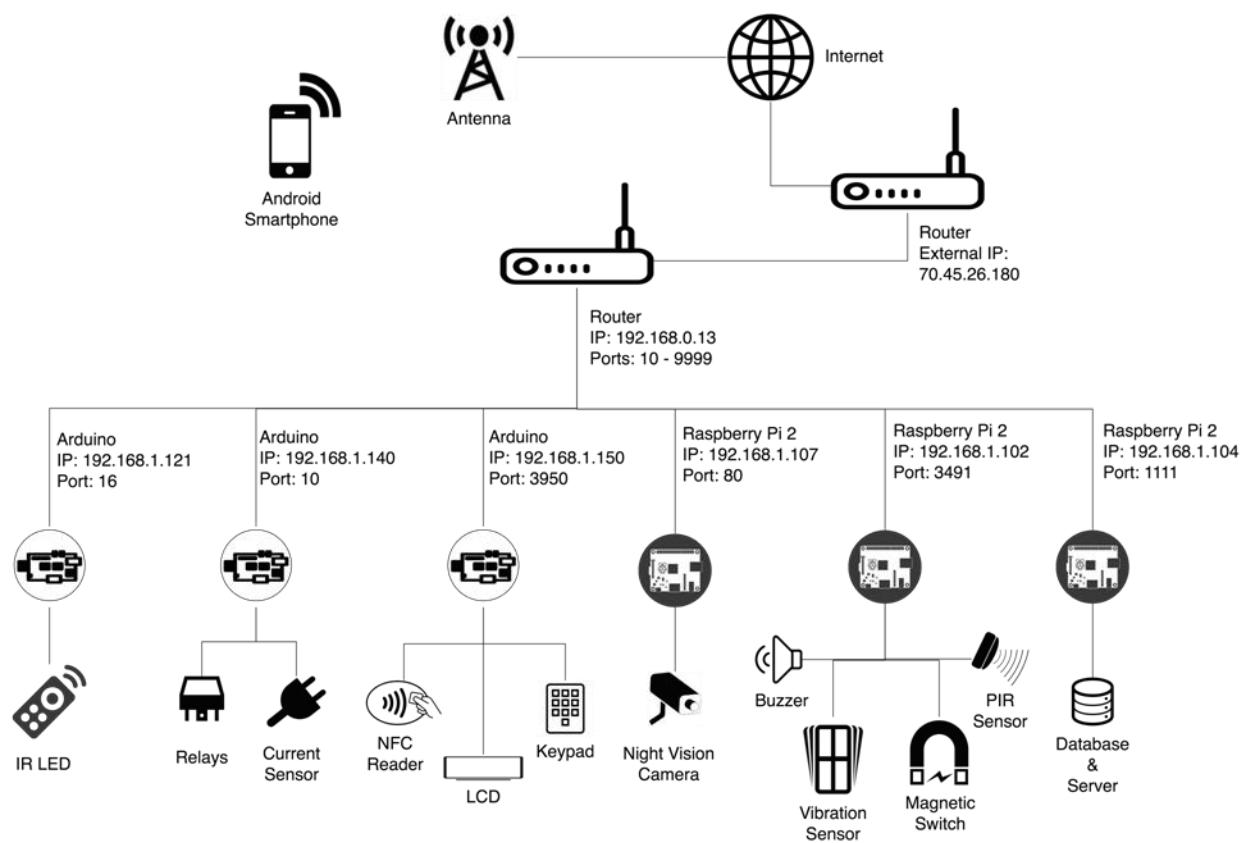
4.3 Mechanical

Mechanical considerations were taken when the systems are implemented. The main purpose was to protect the electrical systems from any mechanical threats. All of the electrical systems were protected inside enclosures that prevented these from mechanical damage.

Chapter 5: Sensors and Microprocessors

5.1 System Overview

The system that RMF Intelligent Home brings will be able to do multiple tasks like energy monitoring, video surveillance, household lights control, infrared control, and an alarm with an integrated user verification system. It has a database which holds the data for the energy monitor and the credentials for each user along with tables for each system. The house will prove the new growing trend the Internet of things, which states that every electronic appliance on the house or at least most of them will communicate with each other without human intervention. As can be seen on the next figure, the complete system behaves as a network of devices connected each by a router and being identified each by an IP address. There are two routers to complete the system, one enables local area connection and the second forwards each internet port in order for the system to have remote access. A mobile application works directly with the system and serves as an user interface between house member and household.



5.2 Infrared transmitter Diode



Figure 5.1.1- Infrared Emitting Diode (IR333-A)

5.2.1 Infrared Emitting Diode (IR333-A) Specifications

Specifications	<ul style="list-style-type: none">• High Reliability• High Radiant Intensity• Peak Wavelength λ p=940nm• 2.54mm Lead Spacing• Low Forward Voltage• Pb free• The Product itself will remain within RoHS compliant version.
Applications	<ul style="list-style-type: none">• Free Air Transmission System• Infrared Remote Control units with high

	power requirement
	<ul style="list-style-type: none">• Smoke Detector• Infrared Applied System

5.1.2 Description

- EVERLIGHT'S Infrared Emitting Diode (IR333-A) is a high intensity diode, molded in a blue transparent plastic package.
- The device is spectrally matched with phototransistor, photodiode and infrared receiver module.

5.3 Infrared Receiver Diode

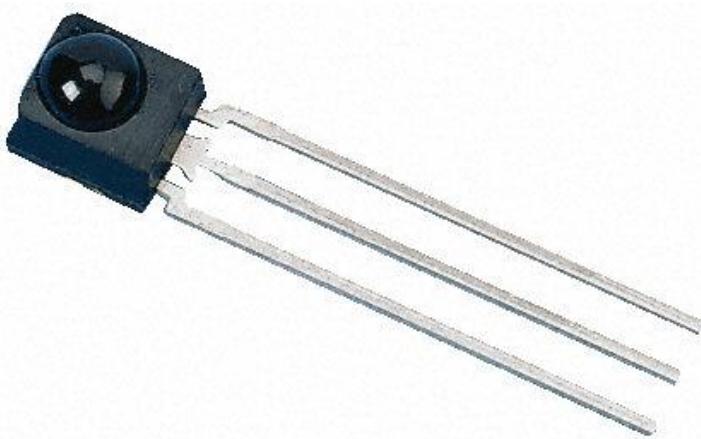


Figure 5.2.1- Infrared Receiver Diode (Tsop4p)

5.3.1 Infrared Receiver Diode (Tsopp4p) Specifications

Specifications	<ul style="list-style-type: none">• Supply Voltage: 2.7~5.5V• Frequency: 38kHz• Receiver Angle: 45°• Operate Distance: 18~20m• Total Length Approx. 30mm• Low Supply Current• Photo detector and preamplifier in one package• Internal filter for burst frequency• Improved shielding against EMI• Improved immunity against ambient
-----------------------	---

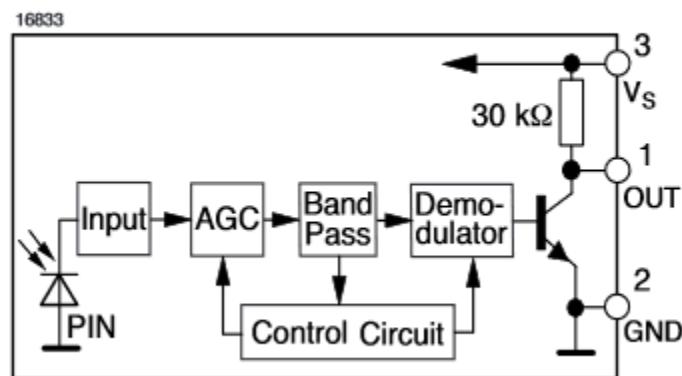
	<p>light</p> <ul style="list-style-type: none"> • Insensitive to supply Voltage Ripple and Noise
--	---

5.3.2 Description

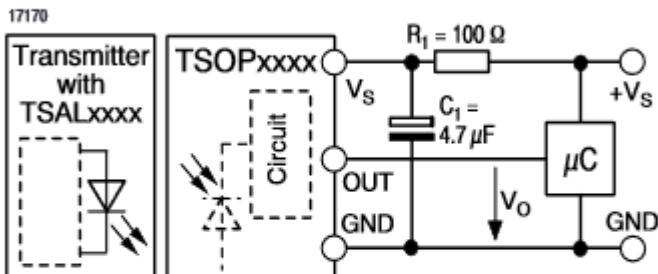
The Tsop4p series are miniaturized receivers for mid-range proximity sensor systems. A PIN diode and a preamplifier are assembled on a lead frame, the epoxy package act as an IR filter. The output pulse width of the tsop4p has an almost linear relationship to the distance of the emitter or the distance of a reflecting object. The tsop4p is optimized to suppress almost all spurious pulses from energy saving fluorescent lamps. This component has not been qualified according to automotive specifications.

5.3.3 Application

Block Diagram



Application Circuit



$R_1 + C_1$ recommended to suppress power supply disturbances.

The output voltage should not be held continuously at a voltage below $V_O = 2.0$ V by the external circuit.

5.4 Relay Shield V2

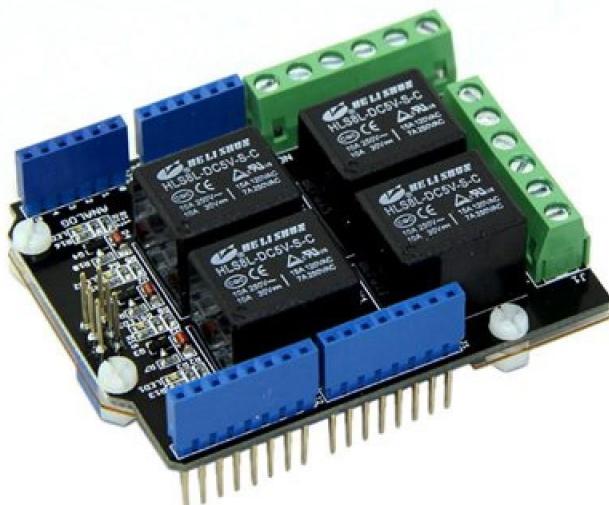


Figure 5.3.1- Relay Shield (RB-see-254)

5.4.1 Relay Shield (RB-see-254) Specifications

Features	<ul style="list-style-type: none"> • Arduino/Seeeduino compatible • Standardized shape design • Working status indicators for each relay • High quality relays • Provides Normally Open/Normally Closed Interfaces
Specifications	<ul style="list-style-type: none"> • Voltage: 4.75 to 5.25VDC • Current: 8 to 250mA • Switching Voltage: 35VDC • Switching Current: 8A • Frequency: 1Hz • Relay Life: 100,000 Cycles • ESD contact discharge: ± 4 KV • ESD air discharge: ± 8 • Dimension 68.7mm X 53.5mm X 30.8mm • Net Weight: 55\pm2g
Cautions	<ul style="list-style-type: none"> • Place Two layers of electrical tape on the top of the Arduino's USB connector to prevent the relay shield from making contact.

- Do not operate with voltages higher than 35VDC.

5.4.2 Description

The Relay Shield utilizes four high quality relays and provides NO/NC interfaces that control the load of high current. Which means it could be a nice solution for controlling devices that couldn't be directly controlled by Arduino's Digital I/Os. Standardized shield form factor enables smoothly connection with the Arduino. The shield also has four dynamic indicators show the on/off state of each relay.

5.5 Magnetic Switch



Figure 5.5.1- Magnetic Switch Sensor

5.5.1 Magnetic Switch Sensor Specifications

Specifications	<ul style="list-style-type: none">• Normally-Closed Dry-Contact Sensor• Holes for Screw-Mounting• Insulation may be required if mounting to metal doors• Cable Type: 2-conductor• Warranty: 1 year (extended warranties available)• Used to detect whether a door is open or closed• Connects to the Goose via analog input or CCAT• Receive alerts via email or SNMP
----------------	--

5.5.2 Description

This sensor offers a low-cost method to monitor access doors or cabinets that should remain closed. The noncontact magnetic switch mounts to the door of a server rack or equipment room and connects to a Goose climate monitor. Remotely check the door status through the web interface or use alarms to notify you if a door is open. The sensor consists of two main parts, a magnet and a switch with terminals to connect the signal wires. The magnet mounts to a door or access panel, while the switch mounts to the frame. When the door is closed the two pieces should be within 1/2" of each other. When the door opens this distance increases, eventually toggling the switch. This allows the Goose to know when the door is open. There are two options for connecting the door switch to the Goose.

You can connect the sensor directly to the digital Input terminals. If the inputs are all in use or the climate monitor doesn't have analog inputs, then you can use a CCAT interface to connect the sensor to a digital sensor port. Multiple door switches can be wired in series to a single Analog Input or CCAT. This is a good way to free up inputs when you don't need to know the specific door that is open, just that at least one door in the group is.

5.5.3 Application

You can configure alarms using the Goose's web interface. With the appropriate alarm thresholds the Goose will send you an alert via email or SNMP trap if a door is detected open. Some units can also trigger an alarm buzzer or energize an output relay in response to an alarm.

5.6 Infra-Red Night Vision



Figure 5.6.1- Infra-Red Night Vision

5.6.1 Infrared Night Vision Specifications

Features	<ul style="list-style-type: none">• This is a IR 30 LED Illumination board plate for CCTV security cameras• High Quality• Can be replaceable if broken or worn out• Low Energy Consumption• Size: 5.2 x 1.7cm /2.0 x 0.7in (Dia. x T)
Details	<ul style="list-style-type: none">• Accurate Design• Bright Infrared LED Illuminator with 30 lights• Built in Auto Protection Circuit in case of high voltage.

5.6.2 Description

Imagine being able to see in total darkness, it's easy with any standard CCTV security camera illuminator board plate and this really cool Infra-Red illuminator. IR infrared 30 LED illuminator board plate for CCTV security camera. Replace such IR LED board for your CCTV security camera. It performs great without much energy consume.

5.7 PIR Motion Detector Sensor (HC-SR501)



Figure 5.6.1- PIR sensor (HC-SR501)

5.7.1 PIR Sensor (HC-SR501) Specifications

Specifications	<ul style="list-style-type: none">• Use BISS0001 signal processing IC, Sanyo Genius Regulator• Voltage: 5V-20V• Power Consumption: 65mA• TTL Output: 3.3V, 0V• Delay Time: Adjustable (0.3 seconds-10 minutes)• Lock Time: 0.2 seconds• Trigger Method: L-disable trigger, H enable repeat trigger• Sensing range: less than 120 degree, within 7 meters
----------------	---

	<ul style="list-style-type: none"> • Temperature: -15 ~ +70 • Dimension: 32*24 mm, distance between screw 28mm, M2, Lens dimension in diameter: 23mm
Features	<ul style="list-style-type: none"> • Automatic detecting the output will be high when objects enter the sensing range and automatically turn to low when object leaves • Photosensitive kcontrol (optional not factory set yet) can be set. • Temperature compensation (optional, factory reset). In the summer when the ambient temperature rises to 30 degrees Celsius to 32 degrees Celsius. • Micro-amp power level consumption, static current <50microamps, particularly suitable for battery powered automatic control products.

5.7.2 Description

HC-SR501 is based on infrared technology, automatic control module, using Germany imported LHI778 probe design, high sensitivity, high reliability, ultra-low-voltage operating mode, widely used in various auto-sensing electrical equipment, especially for battery-powered automatic controlled products.

5.7.3 Adjustments

- **Adjust the distance:** turn the potentiometer clockwise rotation, increased sensing distance (about 7 meters), on the contrary, the sensing distance decreases (about 3 meters).
- **Adjust the delay:** turn the potentiometer clockwise rotation sensor the delay lengthened (600S, 10 minutes), on the contrary, shorten the induction delay (0.3 second).

5.8 Shock Sensor Module



Figure 5.8.1- Shock Sensor Module (801S)

5.8.1 Shock Sensor Module (801S) Specifications

Specifications	<ul style="list-style-type: none">• A very wide range of vibration detection.• No direction limitation• 60,000,000 strokes assurance (especially gold-plated pin surface)• Low loss, sensitivity adjustment circuit can be• Application: Security, Electronic Lock• Size: 10mm X 45mm X 15mm• the main chip: LM393, vibrating probe• working voltage: DC 3-5V• having a signal output instruction• with a TTL level signal, and the analog output signal• the output valid signal is high, the light goes out• the sensitivity is adjustable (fine-tuning)• wide detection range of vibration, no direction <p>with mounting holes, firmware installation flexible and convenient</p>
-----------------------	---

5.8.2 Applications

Can be used in anti-theft devices, electronic locks, mechanical equipment vibration detection, and detection range bull's-eye counts vibration testing occasions.

Shock sensor 801S specification

(Patent's)

- **Features:**

- Micro Shock detecting.
- Non direction limited.
- 60,000,000 times shock guarantee (special gold alloy plated)
- Low cost circuit can adjust Sensitivity.

- **Dimension:**

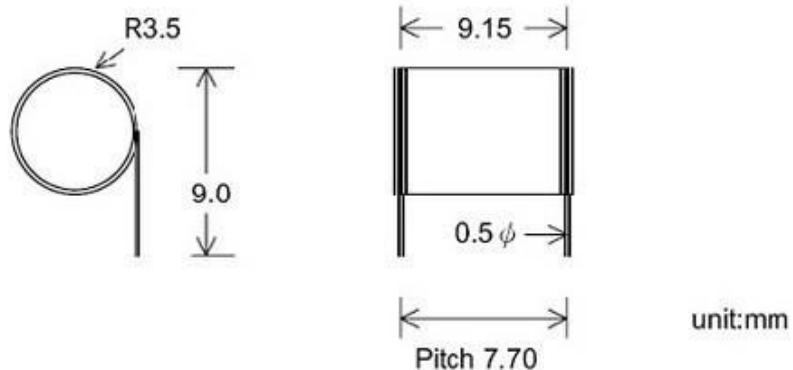
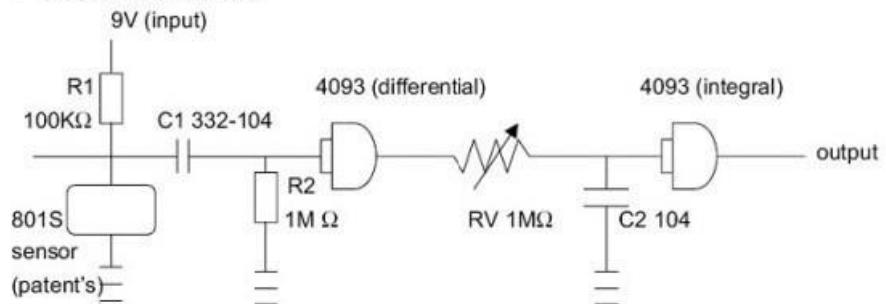


Figure 5.8.2- Sensor Physical Measurements

- Reference circuit:



- Attention:

1. The point of utput must become **square wave**. (*Can not use as shock switch!*)
2. Take low or high of output square wave into your alarm system and adjust sensitivity from the delay time length.

Figure 5.8.3- Sensor Reference circuit

5.9 Current Sensor ACS712 - Invasive

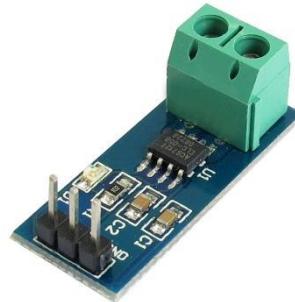


Figure 5.9.1- Current sensor invasive (ACS712)

5.9.1 Current Sensor Invasive (ACS712) Specifications

Specifications	<ul style="list-style-type: none">• Accuracy:$\pm 1.5\%$• Current - Sensing: 5A ~ 30A• Current - Supply (Max): 13mA• For Measuring: AC/DC• Frequency: DC ~ 80kHz• Linearity: $\pm 1.5\%$• Mounting Type: Surface Mount• Number of Channels: 1• Operating Temperature: -40°C ~ 85°C• Output: Ratiometric, Voltage• Package / Case: 8-SOIC (0.154", 3.90mm Width)• Packaging: Tape & Reel (TR)
-----------------------	---

	<ul style="list-style-type: none"> • Polarization: Bidirectional • Response Time: 5µs • Sensitivity: 66mV/A ~ 185mV/A • Sensor Type: Hall Effect, Open Loop • Voltage - Supply: 5V • Total output error 1.5% at $T_A = 25^\circ\text{C}$
Applications	<ul style="list-style-type: none"> • Motor control • Load detection • Management • Switched-mode power supplies • Overcurrent fault protection
Features	<ul style="list-style-type: none"> • Device bandwidth is set via the filter pin • 1.2 mΩ internal conductor resistance • 2.1kVRMS minimum isolation voltage from pins 1-4 to pins 5-8 • Output voltage proportional to AC or DC currents • Low-noise analog signal path • Factory-trimmed for accuracy • Extremely stable output offset voltage • Nearly zero magnetic hysteresis • Ratiometric output from supply voltage

5.9.2 Description

Allegro MicroSystems ACS712 provides economical and precise solutions for AC or DC current sensing in industrial, commercial, and communications systems. The device package allows for easy implementation by the customer. The device is not intended for automotive applications. The device consists of a precise, low-offset, linear Hall sensor circuit with a copper conduction path located near the surface of the die. Applied current flowing through this copper conduction path generates a magnetic field that is sensed by the integrated Hall IC and converted into a proportional voltage. Device accuracy is optimized through the close proximity of the magnetic signal to the Hall transducer. A precise, proportional voltage is provided by the low-offset, chopper-stabilized BiCMOS Hall IC, which is programmed for accuracy after packaging. The ACS712 is provided in a small surface-mount SOIC8 package. The leadframe is plated with 100 percent matte tin, which is compatible with standard lead (Pb) free printed circuit board assembly processes. Internally, the device is Pb-free, except for flip-chip high-temperature Pb-based solder balls, currently exempt from RoHS. The device is fully calibrated prior to shipment from the factory.

The Allergo ACS712 current sensor is based on the principle of Hall effect, which was discovered by Dr. Edwin Hall in 1879. According to this principle, when a current carrying conductor is placed into a magnetic field, a voltage is generated across its edges perpendicular to the directions of both the current and the magnetic field. It is illustrated in the figure shown below. A thin sheet of semiconductor material (called Hall element) is carrying a current (I) and is placed into a magnetic field (B), which is perpendicular to the direction of current flow. Due to the presence of Lorentz force, the distribution of current is no more uniform across the Hall element and therefore a potential difference is created across its edges perpendicular to the directions of both the current and the field. This voltage is known Hall voltage and its typical value is in the order of few microvolts. The Hall voltage is directly proportional to the magnitudes of I and B . So if one of them (I and B) is known, then the observed Hall voltage can be used to estimate the other.

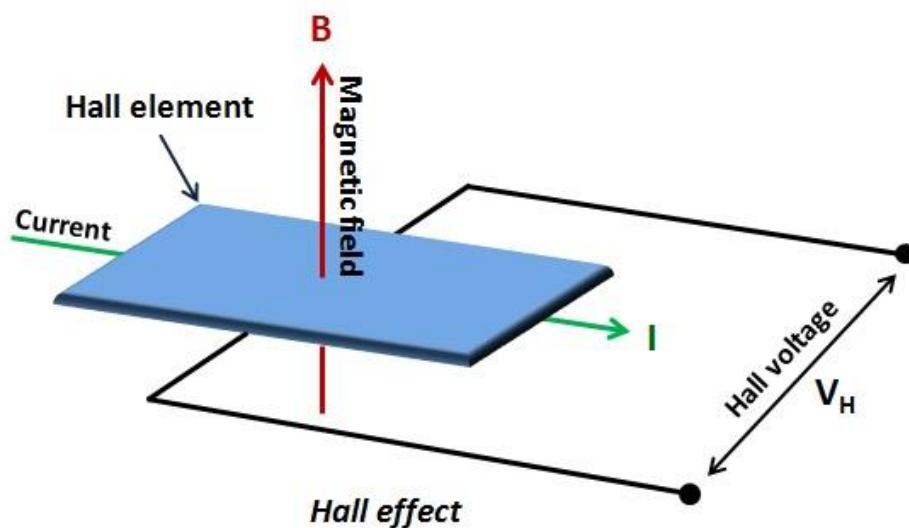


Figure 5.8.2-Hall effect explaining diagram

5.9.3 Sensitivity and output of ACS712

The output of the device has positive slope when an increasing current flows through the copper conduction path (from pins 1 and 2, to pins 3 and 4). The ACS712 device comes in three variants, providing current range of $\pm 5A$ (ACS712-05B), $\pm 20A$ (ACS712-20B), and $\pm 30A$ (ACS712-30A). The ACS712-05B can measure current up to $\pm 5A$ and provides output sensitivity of $185mV/A$ (at $+5V$ power supply), which means for every $1A$ increase in the current through the conduction terminals in positive direction, the output voltage also rises by $185 mV$. The sensitivities of $20A$ and $30A$ versions are $100 mV/A$ and $66 mV/A$, respectively.

5.10 Raspberry Pi Camera Module

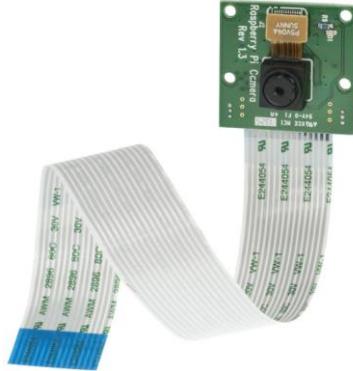


Figure 5.10.1- Raspberry Pi 5MP Camera Board Module

5.10.1 Raspberry Pi 5MP Camera Board Module

Specifications	<ul style="list-style-type: none">• Image Sensor: Omnivision 5647 CMOS image sensor in a fixed-focus module with Integral IR filter.• Resolution: 5 megapixels• Still picture resolution: 2592 x 1944• Max image transfer rate:<ul style="list-style-type: none">○ 1080p: 30fps (encode and decode)○ 960p: 45fps○ 720p: 60fps• Input clock frequency: 6~27 MHz• Connection to Raspberry Pi: 15 Pin ribbon cable, to the dedicated 15-pin
-----------------------	---

	<p>MIPI Camera Serial Interface (CSI-2)</p> <ul style="list-style-type: none"> • Image control functions: <ul style="list-style-type: none"> ◦ Automatic exposure control ◦ Automatic white balance ◦ Automatic band filter ◦ Automatic 50/60 Hz luminance detection ◦ Automatic black level calibration • Temperature range: <ul style="list-style-type: none"> ◦ Operating: -30° to 70° ◦ Stable image: 0° to 50° • Lens size: 1/4" • Dimensions: 20 x 25 x 10mm • Weight: 3g
--	---

5.10.2 Description

This high definition camera module is compatible with the Raspberry Pi. It provides high sensitivity, low crosstalk and low noise image capture in an ultra-small and lightweight design. The camera module connects to the Raspberry Pi board via the CSI connector designed specifically for interfacing to cameras. The CSI bus is capable of extremely high data rates, and it exclusively carries pixel data to the BCM processor.

5.11 Microcontroller Analysis

5.11.1 Microcontroller Specifications

	Arduino Mega:	Beaglebone Black:	Raspberry Pi 2:
Processor:	ATMega 2560	ARM- Cortex A8	ARM-Cortex A7
Clock Speed:	16 MHz	1 GHz	900 MHz
EEPROM:	4 kb	4 kb	
Flash Memory:	256 kb of which 8 kb are used by Bootloader	4 GB	SD Card
SRAM:	8 kb	512 MB	1 GHz
Digital I/O:	54	66	30
PWM I/O:	14	8	2
Analog I/O:	16	7	No
UART I/O:	1	5	1
TWI/I2C I/O:	1	2	1
SPI I/O:	1	1	1
Ethernet:	No	10/100	10/100
USB Master:	No	1 USB 2.0	4 USB
Video Out:	No	HDMI	HDMI and RCA video connector
Audio Out:	No	No	Yes

Operating Power:	500mA@5V	500mA@5V	<u>1.8A@5V</u>
I/O Overall Currents:	200 mA	100 mA	100 mA
Price:	\$37.22	\$45	\$35

5.11.2 Arduino Platform

This platform focuses on hardware control applications. This development board is designed to ease the programming process of a project and enable the user to focus on hardware implementations. It provides multiple input/output pins that have multiple functions such as PWM, analog readings, UART and digital logic control. Arduino incorporates a variety of boards; these boards have different focus or more features. Some of these boards include the Arduino Mega, Due, Leonardo, Lilly pad, Uno, Yun, among others. It has been very well accepted by the “maker” movement and has been enhanced to meet almost any type of control and sometimes processing applications. Documentation is very vast on the Internet about the board and many forums have chats on which you can get help. Due to its extensive community Arduino has a lot of libraries for almost every type of applications, varying from sensor or modules to web applications.



Figure 5.11.1- Arduino UNO

Arduino is programmed in an object-oriented language very similar to C. This platform has its own site at www.arduino.cc and there is well-documented software and photos of all the models available and their respective price and application. As for how to program it, an IDE was designed to make things simpler this very user-friendly integrated development environment is called Arduino IDE. It has many libraries that simplify the programming in a way that easy-to-call functions and libraries cover the assembly part of the logic. The user does not need to know assembly code to program the peripherals of the microprocessor. This allows for rapid out of the box implementation, enabling the user to focus on how the hardware is going to work rather than the code itself.

As for the price, it will vary depending on which of the fore mentioned models, for this case the only microcontroller to be analyzed will be the Arduino Mega 2652. The Arduino Mega 2652 is the most suitable option for this project because it has many I/O pins. The other models are for specific applications like clothing decoration or gaming applications, among others. This model is a general purpose controller with increased I/O capability and given that our project is a new kind of embedded system this model is the perfect choice of all the others offered at the Arduino website. This model contains 54 digital pins of which 14 can PWM, 16 analog pins, and 1 UART connection header. The board is a versatile control platform and can be expanded through shields (add-ons) that give the board many other functions like Ethernet connection or motor controllers.

Advantages	Disadvantages
Many I/O pins.	Little memory space makes it impossible for industrial or big applications.
Many Shields (add-ons) to focus on different applications.	Little processing capability disqualifies the Arduino for media or real time applications.
Many libraries help implementation process.	Lots of included unused libraries.
Very large community.	
Different types of boards with different I/O configurations and different board sizes for diversity of applications.	
Easy to use IDE for out of the box developing.	
No setup required making the board a plug and play console.	

5.11.3 BeagleBone Black Platform

This newly introduced platform combines control and process power into the same board. The credit-card sized board also possesses many input/output pins to enable control applications. For web-based applications the board also comes with a pre-installed Ethernet (RJ45) port. The microprocessor that drives the beagle is enhanced for processing purposes and real-time applications. The microprocessor that handles this board is the Sitara AM3358BZCZ100 developed by Texas Instruments.



Figure 5.11.2- Beaglebone Black

Some of the features that this platform offers are 7 ADCs and 66 digital pins of which 8 can be used for pulse width modulation and 5 can be used as UART. This platform costs \$45 and does not require any extra devices to work, though some features like memory can be expanded up to 32 GB with an external SD card. The platform supports many languages, which include Python, JavaScript, C, C++, Ruby, Perl, java or even a shell script. There is also another super script used to program the beagle bone and it is called BoneScript, it contains many functions to simplify programming. This Linux based minicomputer allows the user to develop in an easy-to-use, widely customizable environment. This board, however, is not as easy to setup as the

Arduino and has various methods to initialize. The most easy and safe method is via USB like the Arduino, this method requires a tether client program like PuTTy to operate. Another setup process is by connecting through an SSH (Secure Shell) connection via the local network of your browser, this method also requires an on-line program to communicate with the board. Another procedure of communicating with the BBB is by connecting a monitor, key board, and mouse to the respective port on the board. This last process will communicate with the operative system of the board and will run like a Linux operative system on the monitor.

The micro-controller was originally designed to facilitate the development of robotic and real-time applications. The beagle board platform is Texas Instrument's contribution to the Makers movement; their team designed a board capable of performing very fast calculations and processes and gave it the ability to do it in Real Time. The PRU's on the BBB can be enabled through libraries and commands and can be used to perform many tasks without interrupting ongoing tasks. A PRU is a Programmable Real-Time Unit, and its job is to enable the embedded Linux system to perform certain tasks without affecting the main portion of the code. This feature has made the beagle bone so popular for robotic projects or for real-time data acquisition.

Advantages	Disadvantages
Balanced processing and control power.	Since it is a new platform it has little community support.
Many programmable I/O pins.	Does not include a high-resolution graphics card disqualifying it for high definition graphic signal processing.
Easy setup for out of the box developing.	Add-ons for the board are limited compared to Arduino and Raspberry Pi.
Rapid processing makes it ideal for real-time applications.	Does not have an exclusive IDE.
12-bit ADC sampling.	
Built in RJ45 Ethernet port for quick internet connection.	
Supports a variety of programming languages, making it available for a lot of software preferences.	
Multi-core processor enables parallel processing applications.	
Many software for programming the board is available online and offline.	

5.11.4 Raspberry Pi 2 Platform

The Raspberry Pi 2 is the upgraded version of the Raspberry Pi B+. This new version has a widely enhanced input/output pin configuration as well as a microprocessor upgrade. Digital controlled applications are possible with the raspberry Pi and processing problems are quite easily managed with the board. Because of the graphics card it contains, the raspberry pi is the best platform for media content applications such as video or sound processing. Similar to the Beaglebone it comes with built-in RJ45 port that gives the ability for Internet connection. It can be programmed using a series of languages such as Python, C, C++, Scratch, Ruby, and Java.



Figure 5.11.3- Raspberry Pi 2

The credit card-sized board was originally developed to make a minicomputer able to perform high quality media applications. On its beginnings it was focused on educational purposes on the UK (United Kingdom) through programming, bringing computer like applications at little cost. This little computer is managed by the powerful quad-core ARM Cortex A7 which runs at 900 (MHz). It has evolved a lot since its first release on February 2012 going from a little single-core computer to a highly compact quad-core computer. The GPU on the raspberry pi can

give 24 GFLOPS and can process videos in 1080p resolution which makes it the perfect machine to process video media files.

The Raspberry Pi 2 does not come loaded with the Linux Raspbian image that it requires to operate. This image is available freely online for most of the leading operating systems on the market which are Linux, Windows and Mac OS. This image will run a complete Linux OS and, if connected to a keyboard, mouse, and monitor can be used as a normal computer. The catch is you require additional storage space for the image, thus need an additional SD card to use it making the board cost go from \$35 to approximately \$60 on the board.

The Raspberry Pi is manufactured by The Raspberry Pi foundation and has its own web page at www.raspberrypi.org. On the page there is documentation and forums to help you get started or complete your project. Of the three platforms chosen this is the one with most documentation on the Internet on how to use it and how to elaborate different projects. There are various models of the raspberry pi from which you can pick and the page also includes direct links to their distributors where there are even more add-ons available to suit different project demands. The community support is very wide on this platform and even schools have been known to use it in their classroom.

Advantages	Disadvantages
Very good GPU enables the board with very high definition graphic processing applications.	Microprocessor does not have an analog-to-digital converter, which means analog data cannot be measured on this device without interfacing it to another device.
Built in RJ45 port for rapid Ethernet connection.	Can only provide 3.3 volts through each I/O pin.
Since it is a relatively old platform there is a lot of information available on the internet for many types of projects.	Requires first time setup so it is not an out of the box plug and play device.
Expanded digital I/O ports for more peripheral interfacing.	Additional storage space required to mount Raspbian image.
Linux based minicomputer.	Not many I/O pins.
Also supports a variety of programming languages which ease programing and troubleshooting	
Quad-core processor enables parallel processing applications.	
1 GB of RAM	

Chapter 6: Lighting System

6.1 Alternatives Considered

One of the main components of the lighting system is the microcontroller. RMF Intelligent Home considered various alternatives when designing the lighting system. There are different options, but we only evaluated the best microcontrollers that are within our price range. Our top choices are Raspberry Pi 2 and the Arduino Uno, and these are the two that we invested more time evaluating. One very important aspect to consider is that the Raspberry Pi 2 GPIO (General Purpose Input Output) pins do not have an analog pins, but one advantage on this board is that it has built in an Ethernet port. The Arduino Uno have analog pins but do not have Ethernet ports built in. So, in our case that everything needs a connection to the network, we need to consider including the Arduino Ethernet shield.

6.2 System Specifications

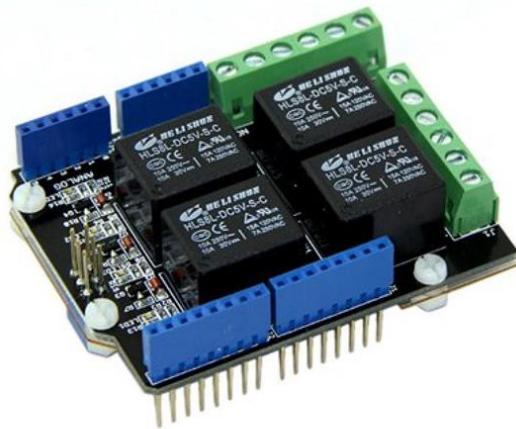


Figure 6.2.1 – 4 Channel Relay Board Shield (RB-See-254)

For this system the Relay Board is utilized for testing the implementation. We use this relay to control the light with a digital bit. We design an electric circuit in order to be able to control the light and the appliance power remotely.



Figure 6.2.2 – Arduino Ethernet Shield Module

To establish a communication with the Arduino, the light control circuit and the android app we need to use a Ethernet shield. With this shield we are able to establish a web server. This server can communicate the Arduino board with the android app. To make possible the control of the light remotely

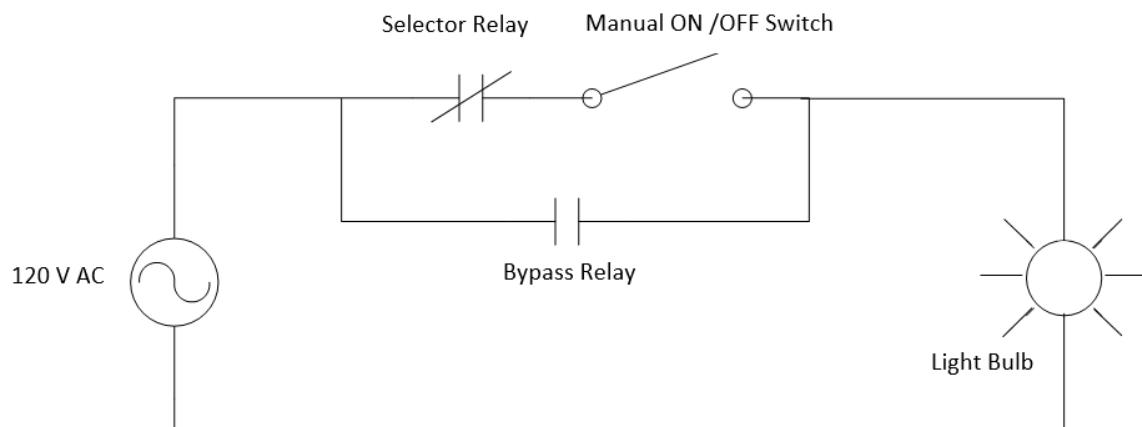


Figure 6.2.3 – Electric Design to Control a Light Remotely

We designed this electric circuit with relays to be able to control de light either remotely or manual. We put a normally closed relay contact that selects whether the circuit is in manual or remote mode. If some problem occurs with the remote circuit the normal state of the circuit becomes manual so that the normal function of the system is not affected.

A	B	C	AND	OR
Manual	Selector	Bypass	Manual*Selector'	Manual*Selector'+ Bypass
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	0	1
1	0	0	1	1
1	0	1	1	1
1	1	0	0	0
1	1	1	0	1

Figure 6.2.4 – Truth Table of Electric Design

This truth table shows all the possible states of the switches and shows in what case the light will turn on. With this table we are able to predict how the circuit is going to work.

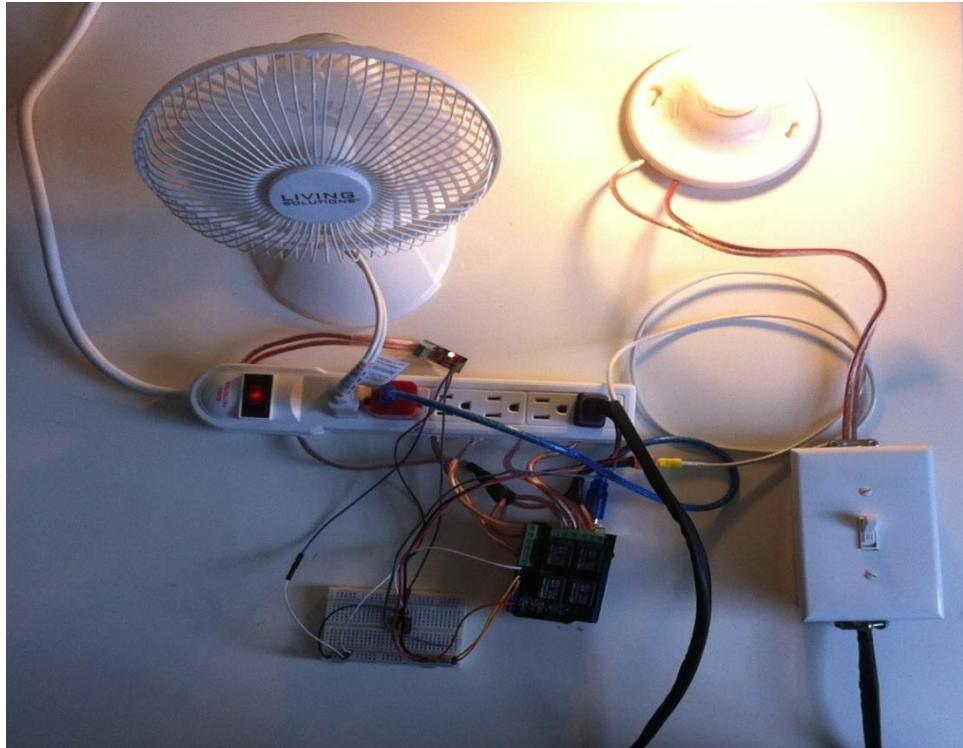


Figure 6.2.5 – Lighting System Implementation

This is an implementation test that we do to prove the concept of communication in order to test the Arduino code. On the last figure, we show the lighting system implemented together with the energy monitor system. We implement the two systems together in the same board make a more efficient system. With this implementation we prove if our circuit was built or designed in order to fulfill our requirements.

6.3 Economic Analysis

For the Lighting system we have 3 main component, an Ethernet shield module an Arduino mega and the Ethernet shield module. We implement this system on a scale sized. To implement this system we modify a multi-outlet to make the socket independent. We put the Ethernet shield in series with the output of the modified multi-outlet to control each one independently. In addition we implement the electric deign showing on the figure 6.2.3. For the purpose of this project, the components chosen were bought on stores, and their respective prices were:

Equipment	Purpose	Price
Arduino uno	The core of the system	1pcs for \$20.99
Ethernet Shield	To cnect the system with the internet	1pcs for \$17.60
4 Channel Relay Shield	To control high voltage with a low voltage logic pin	1pcs for \$ 20.00
Single Channel Relay	To make the bypass circuit	2pcs for \$ 5.95
Multi plug	To put the invasive current sensor	1pcs for \$ 10.00
Light Bulb Soket	To make the electrical conection	2pcs for \$ 5.00
Light Bulb	To prove our circuit	2pcs for \$ 5.00
Light switch with the metal	To implement our system	2pcs for \$ 10.00

housing	an protect it	
Cable	To make the electrical connection	\$ 15.00
Heat Shrink	To protect the cable to make a short circuit	\$ 10.00

6.4 Results

6.4.1 Lighting System Code for Arduino

This program contains the code of the lightning system that are implemented in this code.

This program also has the implementation of the socket connection to communicate with the android app.

```
// Arduino Code Lighting System

#include <SPI.h>      // needed for Arduino versions later than 0018

#include <Ethernet.h>    // Ethernet shield Library

#include <EthernetUdp.h>    // UDP library from: bjoern@cs.stanford.edu

12/30/2008
```

```

// Enter a MAC address and IP address for your controller below.

// The IP address will be dependent on your local network:

byte mac[] = {

    0xEE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };

IPAddress ip(192,168,1,140); // Local ip of the system

unsigned int localPort = 10; // local port to listen on

// buffers for receiving and sending data

char packetBuffer[UDP_TX_PACKET_MAX_SIZE]; //buffer to hold incoming packet,
char ReplyBuffer[] = "acknowledged"; // a string to send back

char replyon1[]="0";// Rely of the state of the Light 1 when on
char replyoff1[]="1";// Rely of the state of the Light 1 when off
char replyon2[]="0";// Rely of the state of the Light 2 when on
char replyoff2[]="1";// Rely of the state of the Light 2 when off
char replyon3[]="0";// Rely of the state of the Switch 1 when on
char replyoff3[]="1";// Rely of the state of the Switch 1 when off
char replyon4[]="0";// Rely of the state of the Switch 2 when on
char replyoff4[]="1";// Rely of the state of the Switch 2 when off

// An EthernetUDP instance to let us send and receive packets over UDP

EthernetUDP Udp;

```

```
void setup() {  
  
// start the Ethernet and UDP:  
  
Ethernet.begin(mac,ip);// intitialize the Ethernet Comunication  
  
Udp.begin(localPort);// initialize the Udp package  
  
  
pinMode(6,OUTPUT);// initialize the pin 6  
  
pinMode(7,OUTPUT);// initialize the pin 7  
  
pinMode(5,OUTPUT);// initialize the pin 5  
  
pinMode(4,OUTPUT);// initialize the pin 4  
  
pinMode(3,OUTPUT);// initialize the pin 3  
  
pinMode(2,OUTPUT);// initialize the pin 2  
  
Serial.begin(9600); // initialize the serial comuniction  
  
}  
  
}
```

```
void loop() {  
  
// if there's data available, read a packet  
  
int packetSize = Udp.parsePacket();  
  
  
if(packetSize)  
{
```

```
Serial.print("Received packet of size ");// Start of the function to print the port and  
the ip adress or the sender of the udp pakage  
  
Serial.println(packetSize);  
  
Serial.print("From ");  
  
IPAddress remote = Udp.remoteIP();  
  
for (int i =0; i < 4; i++)  
{  
    Serial.print(remote[i], DEC);  
    if (i < 3)  
    {  
        Serial.print(".");  
    }  
}  
  
Serial.print(", port ");  
  
Serial.println(Udp.remotePort());  
  
  
// read the packet into packetBufffer  
  
Udp.read(packetBuffer,UDP_TX_PACKET_MAX_SIZE);  
  
Serial.println("Contents:");  
  
Serial.println(packetBuffer);  
  
  
Udp.beginPacket(Udp.remoteIP(), Udp.remotePort()); // End of the function to print  
the port and the ip adress or the sender of the udp pakage
```

```
if(strcmp(packetBuffer,"on1")==0){// if the system receive a udp pakage of on1
activate the pin 2 an deactivate the pin 7

digitalWrite(7, LOW);

digitalWrite(2, HIGH);

Udp.write(ReplyBuffer);

}

else

if(strcmp(packetBuffer,"of1")==0){// if the system receive a udp pakage of of1
activate the pin 7 an deactivate the pin 2

digitalWrite(7, HIGH);

digitalWrite(2, LOW);

Udp.write(ReplyBuffer);

}

else

if(strcmp(packetBuffer,"on2")==0){// if the system receive a udp pakage of on2
activate the pin 3 an deactivate the pin 6

digitalWrite(6,LOW);

digitalWrite(3, HIGH);

Udp.write(ReplyBuffer);

}

else

if(strcmp(packetBuffer,"of2")==0){// if the system receive a udp pakage of of2
```

activate the pin 6 an deactivate the pin 3

```
digitalWrite(6,HIGH);
```

```
digitalWrite(3, LOW);
```

```
Udp.write(ReplyBuffer);
```

```
}
```

```
else
```

```
if(strcmp(packetBuffer,"on3")==0){// if the system receive a udp pakage of on3
```

deactivate the pin 5

```
digitalWrite(5,LOW);
```

```
Udp.write(ReplyBuffer);
```

```
}
```

```
else
```

```
if(strcmp(packetBuffer,"of3")==0){// if the system receive a udp pakage of of3
```

activate the pin 5

```
digitalWrite(5,HIGH);
```

```
Udp.write(ReplyBuffer);
```

```
}
```

```
else
```

```
if(strcmp(packetBuffer,"on4")==0){// if the system receive a udp pakage of on4
```

deactivate the pin 4

```
digitalWrite(4,LOW);
```

```
Udp.write(ReplyBuffer);
```

```
}
```

```
else
    if(strcmp(packetBuffer,"of4")==0){// if the system receive a udp pakage of of4
        activate the pin 4
        digitalWrite(4,HIGH);
        Udp.write(ReplyBuffer);
    }
    else
        if(strcmp(packetBuffer,"kb1")==0){// if the system receive kb1 send the status of
            the pin 7
            if(digitalRead(7))
                Udp.write(replyon1);
            else
                Udp.write(replyoff1);
        }
    else
        if(strcmp(packetBuffer,"kb2")==0){// if the system receive kb2 send the status of
            the pin 3
            if(digitalRead(3))
                Udp.write(replyon2);
            else
                Udp.write(replyoff2);
    }
```

```
else
    if(strcmp(packetBuffer,"kb3")==0){// if the system receive kb3 send the status of
the pin 5
        if(digitalRead(5))
            Udp.write(replyon1);
        else
            Udp.write(replyoff1);
    }

else
    if(strcmp(packetBuffer,"kb4")==0){// if the system receive kb4 send the status of
the pin 4
        if(digitalRead(4))
            Udp.write(replyon1);
        else
            Udp.write(replyoff1);
    }

else
    if(strcmp(packetBuffer,"man")==0){// if the system receive man release all the
holding contact of all the pin
        digitalWrite(7, LOW);
        digitalWrite(2, LOW);
        digitalWrite(6, LOW);
    }
```

```
digitalWrite(3, LOW);
digitalWrite(5, LOW);
digitalWrite(4, LOW);
Udp.write(ReplyBuffer);

}

else

if(strcmp(packetBuffer,"aut")==0){// if the system receive aut put all the pin in the
automatic mode

digitalWrite(7, HIGH);
digitalWrite(2, HIGH);
digitalWrite(6, HIGH);
digitalWrite(3, HIGH);
digitalWrite(5, LOW);
digitalWrite(4, LOW);
Udp.write(ReplyBuffer);

}

memset(packetBuffer,0,sizeof(packetBuffer));

// send a reply, to the IP address and port that sent us the packet we received

Udp.endPacket();

}
```

```
}
```

Figure 6.3.6 – Lighting System Code for Arduino

Chapter 7: IR System

7.1 System Description

7.1.1 NEC Protocol

In order to decode the remote control, lets first look at the coding system that we will use for the remote control. The NEC protocol is a coding system that has the following features:

- 8-bit address spaces, 8-bit command spaces
- Address bits and command bits are transmitted twice for reliability.
- Pulse position modulation.
- Carrier frequency of 38kHz
- Every bit's time is 1.125ms or 2.25ms

To better understand how the coding system works, look at Figure 10.1.1. On the figure we can observe how the system decodes each bit.

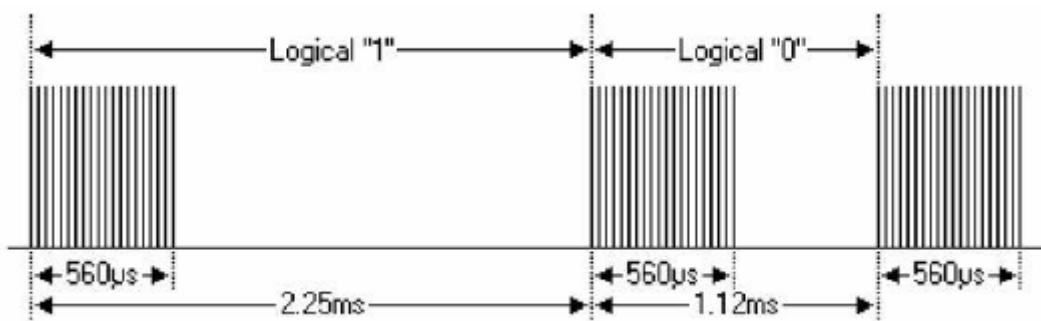


Figure 7.1.1 – TV Model Specifications

Figure 7.1.2 describes the typical NEC protocol pulse sequence. The figure presents the pulse sequence prior sending the LSB (least significant bit) agreement. The message here starts from a 9ms high level, followed by a 4.5ms low level (these two level made boot code) and they by the address code and command code. Address and commands transfer twice. The second time all bits are inverted, can be used for use in the received message recognized. The total transmission time is constant, because the duplication of every point of its length negated.

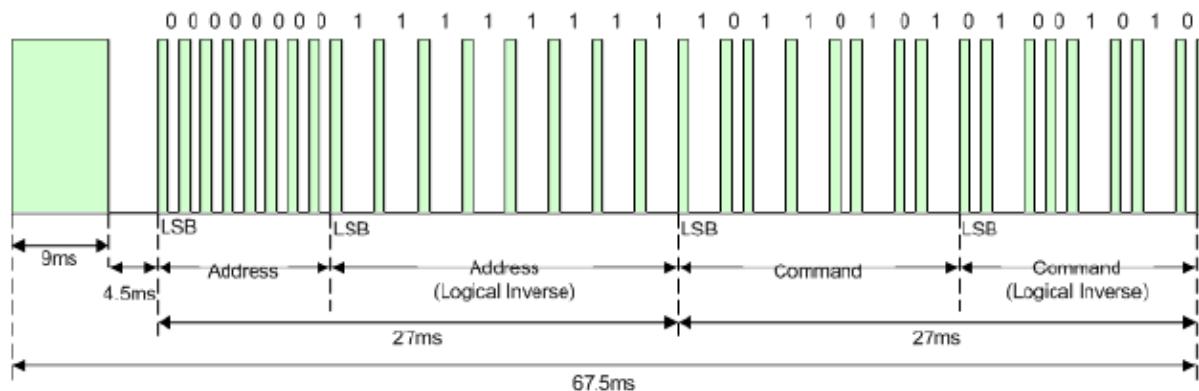


Figure 7.1.2 – TV Model Specifications

According to the characteristics and the receiving end of the waveform of the NEC coding, this experiment will divide receiving end's waveform into five parts: First is the boot code (9ms and 4.5ms pulse), followed by the address code 16(including an 8-bit address and 8-bit address is negated), followed by the command code 16(package including eight command-bit and 8-bit command negated), and finally the end code (560us pulse). It takes 27ms to transmit both the 16 bits for the address and the 16 bits for the command. To fully transmit the message frame, it takes 67.5ms, discounting the final 562.5us that signify the end of the message.

HIGH segments of the received waveform and low section to be measured using the timer, based on the measured time to distinguish a logical “0”, a logical “1”, cited seek pulse, repetitive pulses. Boot code and address code as long as the judge is correct pulse can be, without storage, but the command code must be stored, because each key command codes are different.

7.1.2 TV Controller Introduction

One of the main features of the Intelligent Home project will be the TV controller. In order to design this part of the project, it is required to analyze the infrared signal from the controller of a specific television model. Every television has its own code for each of the control commands, for our purpose we will be using a VIZIO with the model number: VA26LHDTV10T. The selected TV model is presented in Figure 7.2.1, and the specifications of the selected TV are presented in Figure 7.1.2.



Figure 7.1.2 Selected TV Model

Standards are definitely a concept where less is more. Unfortunately, in the world of IR remotes, there are many standards. This makes it more practical to record the signals you wish to send than to try and synthesize them knowing the make and model of the appliance. There are just too many different variations of code.

The part of the team assigned to the communications and controls design focused on analyzing the presented TV model, in order to obtain the necessary commands for the purpose of controlling functions like: increasing or decreasing the volume, increasing or decreasing the channel, and turning ON or OFF the TV.

Infrared remote controls send command as a stream of pulses. These are modulated at a carrier frequency that is around 38kHz. The lengths of the codes vary depending on the brand, but these are usually between 12 and 32 bits. A “1” value indicate the transmitter (sender) LED is ON, and “0” indicate that the transmitter LED is OFF. When a button is pressed, the remote control repeats the message a number of times.

7.1.3 IR TV Controller Diagram

The proposed design for the infrared TV controller design is presented in Figure 7.1.3. As it can be seen, the communication path will be through the server and it will end up in a repeater that will be in charge of amplifying the signal before it reaches the TV IR receiver. The infrared control system is built for the control of household items that involve a conventional infrared remote control. This system simplifies the need of having multiple remotes for multiple devices, making it convenient for users to have all control in one mobile application. Unlike the other systems, this system involves the user having to integrate the IR signals manually into the system's receiver in order to load the action into the system.

The proposed design for the infrared TV controller design is presented in Figure 7.1.3. As it can be seen, the communication path will be through the server and it will end up in a repeater that will be in charge of amplifying the signal before it reaches the TV IR receiver.

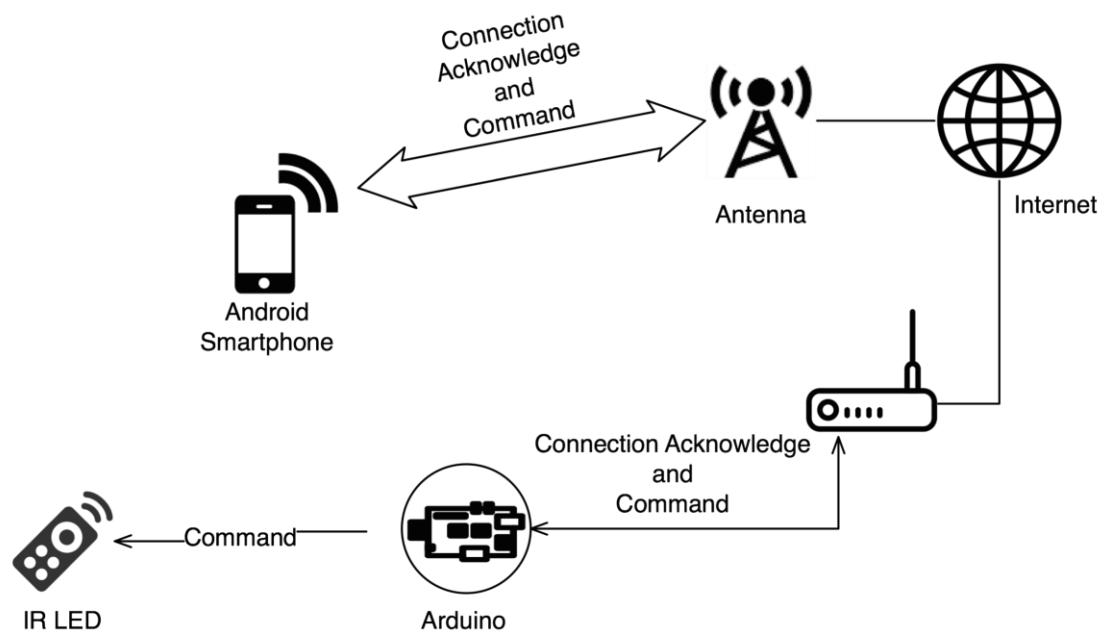


Figure 7.1.3 – TV IR Controller Diagram

The flowchart presented in Figure 7.1.4 explains how the infrared TV controller will make the decisions in order to execute the commands.

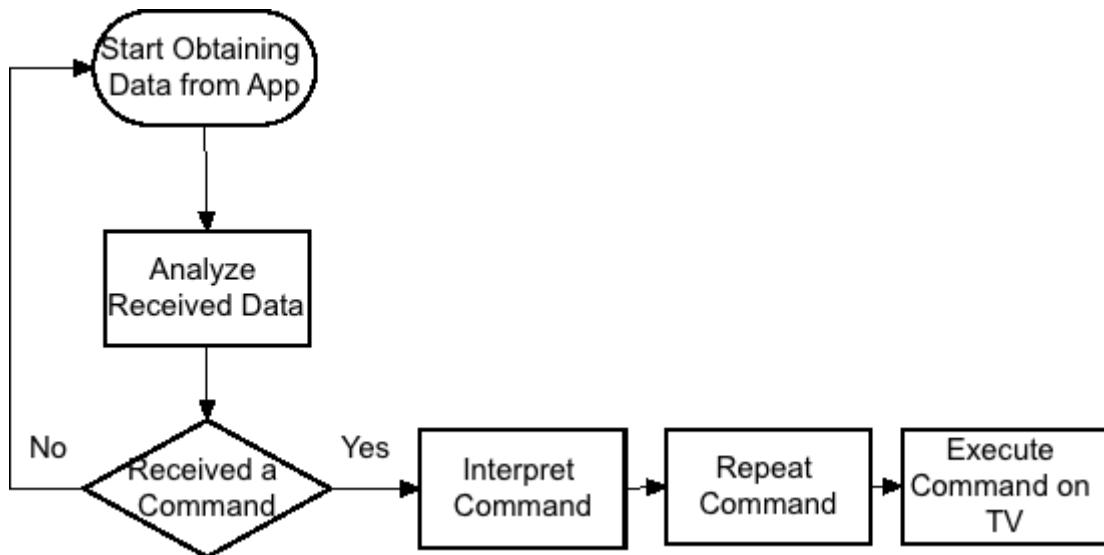


Figure 7.1.4 – TV IR Controller Flowchart

7.1.4 Observing the IR TV Transmitter Code with a Serial Monitor

The first method used to observe the signal was performing reading of the received signal at the receiver phototransistor with a serial monitor. An Arduino board was connected to the signal pin at the phototransistor as is presented at Figure 7.1.5. From the digital pin at the Arduino board we were able to obtain the HIGH and LOW values and after, calculate the times that these occurred.

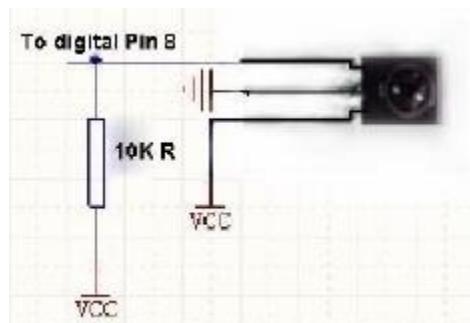


Figure 7.1.5 – Phototransistor Connections

7.2 System Specifications

The components that are used on this system are described on the next table. The prices are included on the economic analysis section of this same system.

Component	Description
Arduino	Microcontroller with the main code of the system and decoder of the signal.
IR LED	Used to transmit the IR sequence depending on the command that the Arduino receives.
Arduino Prototyping Shield	Ease the connection of the system.
IR Receiver Phototransistor	Used to receive the new entry for a command to be put into the system
Arduino Ethernet Shield	Used to send and receive commands via UDP protocol.

7.3 Economic Analysis

After comparing all the available options and figuring out what elements were going to be used, the team decided to keep 5 major components for this system, which are the IR Reciever, IR LED, Arduino microcontroller, the prototyping shield, and the Ethernet shield. The prices for all these elements are shown next on the table of prices.

Device	Use	Price
Arduino UNO Microcontroller	Controller used for modulating the signal required for the television set to change state.	\$21.16 + shipping & handling
Infrared LED	This light emitting diode is capable of sending IR signals. This device will send the modulated signal to the TV.	\$3.49 + shipping and handling
Infrared Receiver	This module will receive the signal that the controller of each input device will send in order to have the sequence for the Arduino to modulate.	10 Pcs. For \$3.87
Prototyping Shield	This shield fill ease the connection to the main	\$13.73 + shipping and handling

	system leaving all the components on the stack of shields on the same Arduino that is in charge of controlling this system.	
Ethernet Shield	This shield gives internet capacity to the controller using the Ethernet connection protocol. This will enable the use of the UDP protocol.	\$33.17 + shipping and handling

7.4 Results

In order to receive the signals from the different IR remotes, the team designed a code that takes this signal and displays it on the serial monitor of the Arduino IDE. This code is shown on figure 7.5.1 and will be used to load all the sequences of each remote control of the household.

7.4.1 Receiving Code for Arduino

```
#define IRpin_PIN PIND  
  
#define IRpin 2  
  
// the maximum pulse time that we will recognize is 65 us  
  
#define MAXPULSE 65000  
  
// For the timing resolution larger is better because it is more precise,  
// but too large won't result in an accurate timing  
  
#define RESOLUTION 20  
  
// The number of pulse pairs to store is of a 100 pulse pairs, more than needed  
  
uint16_t pulses[100][2]; // pair is high and low pulse  
  
uint8_t currentpulse = 0; // index for pulses we're storing  
  
void setup(void) {  
  
    Serial.begin(9600);  
  
    Serial.println("Ready to decode IR!");  
  
}  
  
void loop(void) {  
  
    uint16_t highpulse, lowpulse; // temporary storage timing  
  
    highpulse = lowpulse = 0; // start out with no pulse length  
  
    while (IRpin_PIN & (1 << IRpin)) {  
  
        // pin is still HIGH  
  
        // count off another few microseconds  
  
        highpulse++;  
  
        delayMicroseconds(RESOLUTION);  
  
        // If the pulse is too long, we 'timed out' - either nothing
```

```

// was received or the code is finished, so print
// what we've grabbed so far, and then reset

if ((highpulse >= MAXPULSE) && (currentpulse != 0)) {
    printpulses();
    currentpulse=0;
    return;
}

}

// we didn't time out so lets stash the reading
pulses[currentpulse][0]= highpulse;

//           same as above

while(! (IRpin_PIN & _BV(IRpin))) {

// pin is still LOW

lowpulse++;

delayMicroseconds(RESOLUTION);

if ((lowpulse >= MAXPULSE) && (currentpulse != 0)) {
    printpulses();
    currentpulse=0;
    return;
}

}

pulses[currentpulse][1]= lowpulse;

//we read one high-low pulse successfully, continue!

currentpulse++;

```

```
}
```

```
void printpulses(void) {
```

```
Serial.println("\n\r\n\rReceived:\n\rOFF \tON");
```

```
for (uint8_t i = 0; i < currentpulse; i++) {
```

```
Serial.print("delayMicroseconds(");
```

```
Serial.print(pulses[i][0]* RESOLUTION, DEC);
```

```
Serial.println(");");
```

```
Serial.print("pulseIR(");
```

```
Serial.print(pulses[i][1]* RESOLUTION, DEC);
```

```
Serial.println(");"); }
```

7.4.2 System Code for Arduino

After everything going to be coded in to the system was measured, the system code was written with every function of the TV controller. The system, as explained before receives a UDP packet with the command of the TV which the user wants to input. After the Arduino sends the signal to the TV and serves as the main controller with the press of a button on the app. This makes everything easier because the user can integrate different IR signals on the same system and control them via the app, whereas the normal method the user needs to have every control at hand. The system code is shown on the next figure with all its functions.

```
// Arduino code of the Energy Monitor System

#include <SPI.h>          // needed for Arduino versions later than 0018

#include <Ethernet.h>      // Ethernet shield Library

#include <EthernetUdp.h>    // UDP library from: bjoern@cs.stanford.edu 12/30/2008

int IRledPin = 2;           // LED connected to digital pin 13

// Enter a MAC address and IP address for your controller below.

// The IP address will be dependent on your local network:

byte mac[] = {

  0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0x01

};

IPAddress ip(192, 168, 1, 121); // Local ip of the system
```

```

unsigned int localPort = 16;      // local port to listen on

// buffers for receiving and sending data
char packetBuffer[UDP_TX_PACKET_MAX_SIZE]; //buffer to hold incoming packet,
char ReplyBuffer[] = "acknowledged"; // a string to send back

// An EthernetUDP instance to let us send and receive packets over UDP
EthernetUDP Udp;

void setup() {
    // start the Ethernet and UDP:
    Ethernet.begin(mac, ip); // initialize the Ethernet Communication
    Udp.begin(localPort); // initialize the Udp package

    pinMode(IRledPin, OUTPUT); // initialize the pin 2 as an output
    Serial.begin(9600); // initialize the serial communication
}

void loop() {
    // if there's data available, read a packet
    int packetSize = Udp.parsePacket();

    if (packetSize)
    {
        Serial.print("Received packet of size ");

```

```
// Start of the function to print the port and the ip adress or the sender of the udp pakage

Serial.println(packetSize);

Serial.print("From ");

IPAddress remote = Udp.remoteIP();

for (int i = 0; i < 4; i++)

{

Serial.print(remote[i], DEC);

if (i < 3)

{

Serial.print(".");

}

}

Serial.print(", port ");

Serial.println(Udp.remotePort());


// read the packet into packetBufffer

Udp.read(packetBuffer, UDP_TX_PACKET_MAX_SIZE);

Serial.println("Contents:");

Serial.println(packetBuffer);


// send a reply, to the IP address and port that sent us the packet we received

Udp.beginPacket(Udp.remoteIP(), Udp.remotePort());


// End of the function to print the port and the ip adress or the sender of the udp pakage
```

```

if (strcmp(packetBuffer, "vl+")==0){// if the system receive a udp pakage vl+ Call the function
VolumeUp
VolumeUP();
Udp.write(ReplyBuffer);
}

else
if (strcmp(packetBuffer, "vl-")==0){ // if the system receive a udp pakage vl- Call the function
VolumeDOWN
Serial.println("Sending IR signal");
VolumeDOWN();
Udp.write(ReplyBuffer);
}

else
if (strcmp(packetBuffer, "ch+")==0){// if the system receive a udp pakage ch+ Call the
function ChannelUP
ChannelUP();
Udp.write(ReplyBuffer);
}

else
if (strcmp(packetBuffer, "ch-")==0){// if the system receive a udp pakage ch- Call the function
ChannelDOWN
ChannelDOWN();
Udp.write(ReplyBuffer);
}else

```

```
if (strcmp(packetBuffer,"pwr")==0){// if the system receive a udp pakage pwr Call the
function Power

Power();

Udp.write(ReplyBuffer);

}

else

if (strcmp(packetBuffer,"bt1")==0){// if the system receive a udp pakage bt1 Call the function
Button1

Button1();

Udp.write(ReplyBuffer);

}

else

if (strcmp(packetBuffer,"bt2")==0){// if the system receive a udp pakage bt2 Call the function
Button2

Button2();

Udp.write(ReplyBuffer);

}

else

if (strcmp(packetBuffer,"bt3")==0){ // if the system receive a udp pakage bt3 Call the function
Button3

Button3();

Udp.write(ReplyBuffer);

}

else
```

```
if (strcmp(packetBuffer,"bt4")==0){ // if the system receive a udp pakage bt4 Call the function  
Button4  
    Button4();  
    Udp.write(ReplyBuffer);  
}  
else  
    if (strcmp(packetBuffer,"bt5")==0){ // if the system receive a udp pakage bt5 Call the function  
Button5  
    Button5();  
    Udp.write(ReplyBuffer);  
}  
else  
    if (strcmp(packetBuffer,"bt6")==0){ // if the system receive a udp pakage bt6 Call the function  
Button6  
    Button6();  
    Udp.write(ReplyBuffer);  
}  
else  
    if (strcmp(packetBuffer,"bt7")==0){ // if the system receive a udp pakage bt7 Call the function  
Button7  
    Button7();  
    Udp.write(ReplyBuffer);  
}  
else
```

```
if (strcmp(packetBuffer,"bt8")==0){ // if the system receive a udp pakage bt8 Call the function  
Button8  
    Button8();  
    Udp.write(ReplyBuffer);  
}  
else  
    if (strcmp(packetBuffer,"bt9")==0){ // if the system receive a udp pakage bt9 Call the function  
Button9  
    Button9();  
    Udp.write(ReplyBuffer);  
}  
else  
    if (strcmp(packetBuffer,"bt0")==0){ // if the system receive a udp pakage bt0 Call the function  
Button0  
    Button0();  
    Udp.write(ReplyBuffer);  
}  
else  
    if (strcmp(packetBuffer,"ext")==0){ // if the system receive a udp pakage ext Call the function  
ext  
    Exit();  
    Udp.write(ReplyBuffer);  
}  
else
```

```

if (strcmp(packetBuffer, "inp") == 0) { // if the system receive a udp pakage inp Call the function
Input

Input();

Udp.write(ReplyBuffer);

}

Udp.endPacket();

}

}

// This procedure sends a 38KHz pulse to the IRledPin
// for a certain # of microseconds. We'll use this whenever we need to send codes

void pulseIR(long microsecs) {
// we'll count down from the number of microseconds we are told to wait

cli(); // this turns off any background interrupts

while (microsecs > 0) {
// 38 kHz is about 13 microseconds high and 13 microseconds low
digitalWrite(IRledPin, HIGH); // this takes about 3 microseconds to happen
delayMicroseconds(10); // hang out for 10 microseconds
}
}

```

```
digitalWrite(IRledPin, LOW); // this also takes about 3 microseconds
delayMicroseconds(10); // hang out for 10 microseconds

// so 26 microseconds altogether
microsecs -= 26;

}

sei(); // this turns them back on
}

void VolumeUP(){ // function that have the duty cycle of the pulse of the Infra-Red Signal to
Control the Volume up

delayMicroseconds( 16640 );
pulseIR( 9060 ) ;

delayMicroseconds( 4160 );
pulseIR( 780 ) ;

delayMicroseconds( 300 );
pulseIR( 760 ) ;

delayMicroseconds( 340 );
pulseIR( 720 ) ;

delayMicroseconds( 1460 );
pulseIR( 680 ) ;

delayMicroseconds( 440 );
pulseIR( 640 ) ;
```

```
delayMicroseconds( 460 );
pulseIR( 600 ) ;

delayMicroseconds( 480 );
pulseIR( 600 ) ;

delayMicroseconds( 500 );
pulseIR( 580 ) ;

delayMicroseconds( 500 );
pulseIR( 580 ) ;

delayMicroseconds( 1620 );
pulseIR( 560 ) ;

delayMicroseconds( 1620 );
pulseIR( 580 ) ;

delayMicroseconds( 520 );
pulseIR( 560 ) ;

delayMicroseconds( 1620 );
pulseIR( 580 ) ;

delayMicroseconds( 1600 );
pulseIR( 580 ) ;

delayMicroseconds( 1620 );
pulseIR( 560 ) ;

delayMicroseconds( 1620 );
pulseIR( 580 ) ;

delayMicroseconds( 1620 );
pulseIR( 560 ) ;
```

```
delayMicroseconds( 520 ) ;  
pulseIR( 560 ) ;  
delayMicroseconds( 1620 ) ;  
pulseIR( 580 ) ;  
delayMicroseconds( 520 );  
pulseIR( 580 ) ;  
delayMicroseconds( 500 );  
pulseIR( 580 ) ;  
delayMicroseconds( 520 ) ;  
pulseIR( 580 ) ;  
delayMicroseconds( 520 );  
pulseIR( 580 ) ;  
delayMicroseconds( 500 );  
pulseIR( 580 ) ;  
delayMicroseconds( 520 );  
pulseIR( 560 ) ;  
delayMicroseconds( 1620 );  
pulseIR( 580 ) ;  
delayMicroseconds( 520 ) ;  
pulseIR( 560 ) ;  
delayMicroseconds( 1620 );  
pulseIR( 580 ) ;  
delayMicroseconds( 1600 );  
pulseIR( 580 ) ;
```

```
delayMicroseconds( 1620 ) ;  
pulseIR( 560 ) ;  
delayMicroseconds( 1620 ) ;  
pulseIR( 580 ) ;  
delayMicroseconds( 1620 );  
pulseIR( 560 ) ;  
delayMicroseconds( 1620 ) ;  
pulseIR( 580 ) ;  
delayMicroseconds( 39300 ) ;  
pulseIR( 8780 ) ;  
delayMicroseconds( 2240 ) ;  
pulseIR( 520 ) ;  
delayMicroseconds( 28564 ) ;  
pulseIR( 8780 ) ;  
delayMicroseconds( 2260 ) ;  
pulseIR( 500 ) ;  
}  
  
void VolumeDOWN(){ // function that have the duty cycle of the pulse of the Infra-Red Signal  
to Control the Volume Down  
    delayMicroseconds(61772);  
    pulseIR(8920);  
    delayMicroseconds(4320);  
    pulseIR(620);  
    delayMicroseconds(480);
```

```
pulseIR(600);

delayMicroseconds(480);

pulseIR(580);

delayMicroseconds(1600);

pulseIR(580);

delayMicroseconds(520);

pulseIR(580);

delayMicroseconds(520);

pulseIR(560);

delayMicroseconds(520);

pulseIR(560);

delayMicroseconds(540);

pulseIR(560);

delayMicroseconds(520);

pulseIR(580);

delayMicroseconds(1620);

pulseIR(560);

delayMicroseconds(1620);

pulseIR(560);

delayMicroseconds(520);

pulseIR(580);

delayMicroseconds(1600);

pulseIR(580);

delayMicroseconds(1620);
```

```
pulseIR(560);

delayMicroseconds(1620);

pulseIR(580);

delayMicroseconds(1600);

pulseIR(580);

delayMicroseconds(1600);

pulseIR(580);

delayMicroseconds(1620);

pulseIR(560);

delayMicroseconds(1620);

pulseIR(580);

delayMicroseconds(520);

pulseIR(560);

delayMicroseconds(520);

pulseIR(580);

delayMicroseconds(520);

pulseIR(560);

delayMicroseconds(520);

pulseIR(580);

delayMicroseconds(520);

pulseIR(560);

delayMicroseconds(520);

pulseIR(580);

delayMicroseconds(500);
```

```
pulseIR(580);

delayMicroseconds(520);

pulseIR(580);

delayMicroseconds(1600);

pulseIR(580);

delayMicroseconds(1620);

pulseIR(560);

delayMicroseconds(1620);

pulseIR(580);

delayMicroseconds(1600);

pulseIR(580);

delayMicroseconds(1600);

pulseIR(580);

delayMicroseconds(1620);

pulseIR(560);

delayMicroseconds(39280);

pulseIR(8820);

delayMicroseconds(2180);

pulseIR(580);

}

void ChannelUP(){ // function that have the duty cycle of the pulse of the Infra-Red Signal to
Control the ChannelUP

delayMicroseconds(13932);

pulseIR(8920);
```

```
delayMicroseconds(4320);  
pulseIR(600);  
delayMicroseconds(480);  
pulseIR(600);  
delayMicroseconds(500);  
pulseIR(580);  
delayMicroseconds(1600);  
pulseIR(580);  
delayMicroseconds(520);  
pulseIR(560);  
delayMicroseconds(520);  
pulseIR(580);  
delayMicroseconds(520);  
pulseIR(560);  
delayMicroseconds(520);  
pulseIR(580);  
delayMicroseconds(520);  
pulseIR(560);  
delayMicroseconds(1620);  
pulseIR(580);  
delayMicroseconds(1600);  
pulseIR(580);  
delayMicroseconds(520);  
pulseIR(560);
```

```
delayMicroseconds(1620);
pulseIR(580);
delayMicroseconds(1600);
pulseIR(580);
delayMicroseconds(1620);
pulseIR(560);
delayMicroseconds(1620);
pulseIR(560);
delayMicroseconds(1620);
pulseIR(580);
delayMicroseconds(520);
pulseIR(560);
delayMicroseconds(520);
pulseIR(580);
delayMicroseconds(520);
pulseIR(560);
delayMicroseconds(520);
pulseIR(580);
delayMicroseconds(520);
pulseIR(560);
delayMicroseconds(520);
pulseIR(580);
delayMicroseconds(520);
pulseIR(560);
```

```
delayMicroseconds(520);  
pulseIR(580);  
delayMicroseconds(1620);  
pulseIR(560);  
delayMicroseconds(1620);  
pulseIR(560);  
delayMicroseconds(1620);  
pulseIR(580);  
delayMicroseconds(1600);  
pulseIR(580);  
delayMicroseconds(1620);  
pulseIR(560);  
delayMicroseconds(1620);  
pulseIR(560);  
delayMicroseconds(1620);  
pulseIR(580);  
delayMicroseconds(1600);  
pulseIR(580);  
delayMicroseconds(39280);  
pulseIR(8820);  
delayMicroseconds(2180);  
pulseIR(560);  
delayMicroseconds(28464);  
pulseIR(8820);
```

```
delayMicroseconds(2180);

pulseIR(580);

}

void ChannelDOWN(){// function that have the duty cycle of the pulse of the Infra-Red Signal
to Control the Channel Down

delayMicroseconds(46284);

pulseIR(8920);

delayMicroseconds(4300);

pulseIR(620);

delayMicroseconds(480);

pulseIR(600);

delayMicroseconds(500);

pulseIR(580);

delayMicroseconds(1600);

pulseIR(580);

delayMicroseconds(520);

pulseIR(560);

delayMicroseconds(520);

pulseIR(560);

delayMicroseconds(520);

pulseIR(580);

delayMicroseconds(520);

pulseIR(560);

delayMicroseconds(520);
```

```
pulseIR(580);

delayMicroseconds(1620);

pulseIR(560);

delayMicroseconds(1620);

pulseIR(560);

delayMicroseconds(540);

pulseIR(560);

delayMicroseconds(1620);

pulseIR(560);

delayMicroseconds(1620);

pulseIR(560);

delayMicroseconds(1620);

pulseIR(580);

delayMicroseconds(1620);

pulseIR(560);

delayMicroseconds(1620);

pulseIR(580);

delayMicroseconds(520);

pulseIR(560);

delayMicroseconds(520);

pulseIR(580);

delayMicroseconds(520);
```

```
pulseIR(560);

delayMicroseconds(520);

pulseIR(580);

delayMicroseconds(520);

pulseIR(580);

delayMicroseconds(500);

pulseIR(580);

delayMicroseconds(520);

pulseIR(560);

delayMicroseconds(520);

pulseIR(580);

delayMicroseconds(1620);

pulseIR(560);

delayMicroseconds(1620);

pulseIR(560);

delayMicroseconds(1620);

pulseIR(560);

delayMicroseconds(1620);

pulseIR(580);

delayMicroseconds(1620);

pulseIR(560);

delayMicroseconds(1620);

pulseIR(560);

delayMicroseconds(1620);
```

```
pulseIR(580);

delayMicroseconds(39280);

pulseIR(8800);

delayMicroseconds(2200);

pulseIR(560);

delayMicroseconds(28464);

pulseIR(8800);

delayMicroseconds(2200);

pulseIR(560);

}

void Power()// function that have the duty cycle of the pulse of the Infra-Red Signal to Control
the Power of the device on/off

delayMicroseconds(43304);

pulseIR(8920);

delayMicroseconds(4300);

pulseIR(640);

delayMicroseconds(460);

pulseIR(600);

delayMicroseconds(500);

pulseIR(580);

delayMicroseconds(1600);

pulseIR(580);

delayMicroseconds(520);

pulseIR(560);
```

```
delayMicroseconds(520);  
pulseIR(580);  
delayMicroseconds(520);  
pulseIR(560);  
delayMicroseconds(520);  
pulseIR(580);  
delayMicroseconds(520);  
pulseIR(560);  
delayMicroseconds(1620);  
pulseIR(560);  
delayMicroseconds(1620);  
pulseIR(560);  
delayMicroseconds(540);  
pulseIR(560);  
delayMicroseconds(1620);  
pulseIR(580);  
delayMicroseconds(1600);  
pulseIR(580);  
delayMicroseconds(1600);  
pulseIR(580);  
delayMicroseconds(1620);  
pulseIR(560);  
delayMicroseconds(1620);  
pulseIR(560);
```

```
delayMicroseconds(540);  
pulseIR(560);  
delayMicroseconds(520);  
pulseIR(580);  
delayMicroseconds(520);  
pulseIR(560);  
delayMicroseconds(1620);  
pulseIR(560);  
delayMicroseconds(520);  
pulseIR(580);  
delayMicroseconds(520);  
pulseIR(560);  
delayMicroseconds(520);  
pulseIR(580);  
delayMicroseconds(520);  
pulseIR(580);  
delayMicroseconds(1600);  
pulseIR(560);  
delayMicroseconds(1620);  
pulseIR(580);  
delayMicroseconds(1620);  
pulseIR(560);  
delayMicroseconds(520);  
pulseIR(580);
```

```
delayMicroseconds(1600);
pulseIR(600);
delayMicroseconds(1600);
pulseIR(560);
delayMicroseconds(1620);
pulseIR(560);
delayMicroseconds(1620);
pulseIR(580);
delayMicroseconds(39280);
pulseIR(8820);
delayMicroseconds(2180);
pulseIR(580);
}

void Button1(){// function that have the duty cycle of the pulse of the Infra-Red Signal to
Control the Channel Button 1
delayMicroseconds(52448);
pulseIR(8840);
delayMicroseconds(4340);
pulseIR(620);
delayMicroseconds(480);
pulseIR(600);
delayMicroseconds(480);
pulseIR(620);
delayMicroseconds(1560);
```

```
pulseIR(620);

delayMicroseconds(480);

pulseIR(620);

delayMicroseconds(460);

pulseIR(620);

delayMicroseconds(480);

pulseIR(600);

delayMicroseconds(480);

pulseIR(620);

delayMicroseconds(480);

pulseIR(600);

delayMicroseconds(1580);

pulseIR(620);

delayMicroseconds(1560);

pulseIR(620);

delayMicroseconds(480);

pulseIR(620);

delayMicroseconds(1560);

pulseIR(620);

delayMicroseconds(1580);

pulseIR(600);

delayMicroseconds(1580);

pulseIR(600);

delayMicroseconds(1580);
```

```
pulseIR(620);

delayMicroseconds(1560);

pulseIR(620);

delayMicroseconds(1560);

pulseIR(620);

delayMicroseconds(480);

pulseIR(620);

delayMicroseconds(480);

pulseIR(600);

delayMicroseconds(480);

pulseIR(620);

delayMicroseconds(1560);

pulseIR(620);

delayMicroseconds(480);

pulseIR(600);

delayMicroseconds(500);

pulseIR(600);

delayMicroseconds(480);

pulseIR(620);

delayMicroseconds(480);

pulseIR(600);

delayMicroseconds(1580);

pulseIR(600);

delayMicroseconds(1580);
```

```
pulseIR(620);

delayMicroseconds(1560);

pulseIR(620);

delayMicroseconds(480);

pulseIR(600);

delayMicroseconds(1580);

pulseIR(620);

delayMicroseconds(1580);

pulseIR(600);

delayMicroseconds(1580);

pulseIR(600);

delayMicroseconds(39260);

pulseIR(8840);

delayMicroseconds(2160);

pulseIR(600);

delayMicroseconds(28424);

pulseIR(8860);

delayMicroseconds(2140);

pulseIR(600);

delayMicroseconds(28444);

pulseIR(8840);

delayMicroseconds(2140);

pulseIR(620);

}
```

```
void Button2(){// function that have the duty cycle of the pulse of the Infra-Red Signal to  
Control the Channel Button 2  
  
delayMicroseconds(48936);  
  
pulseIR(8840);  
  
delayMicroseconds(4340);  
  
pulseIR(620);  
  
delayMicroseconds(480);  
  
pulseIR(600);  
  
delayMicroseconds(480);  
  
pulseIR(620);  
  
delayMicroseconds(1580);  
  
pulseIR(600);  
  
delayMicroseconds(480);  
  
pulseIR(620);  
  
delayMicroseconds(460);  
  
pulseIR(620);  
  
delayMicroseconds(480);  
  
pulseIR(620);  
  
delayMicroseconds(460);  
  
pulseIR(620);  
  
delayMicroseconds(480);  
  
pulseIR(620);  
  
delayMicroseconds(1560);
```

```
pulseIR(620);

delayMicroseconds(1560);

pulseIR(620);

delayMicroseconds(480);

pulseIR(620);

delayMicroseconds(1560);

pulseIR(620);

delayMicroseconds(1560);

pulseIR(620);

delayMicroseconds(1580);

pulseIR(600);

delayMicroseconds(1580);

pulseIR(620);

delayMicroseconds(1560);

pulseIR(620);

delayMicroseconds(480);

pulseIR(600);

delayMicroseconds(1580);

pulseIR(620);

delayMicroseconds(480);

pulseIR(600);

delayMicroseconds(480);

pulseIR(620);

delayMicroseconds(1460);
```

```
pulseIR(720);
```

```
delayMicroseconds(480);
```

```
pulseIR(600);
```

```
delayMicroseconds(480);
```

```
pulseIR(620);
```

```
delayMicroseconds(480);
```

```
pulseIR(600);
```

```
delayMicroseconds(1580);
```

```
pulseIR(620);
```

```
delayMicroseconds(480);
```

```
pulseIR(600);
```

```
delayMicroseconds(1580);
```

```
pulseIR(620);
```

```
delayMicroseconds(1560);
```

```
pulseIR(620);
```

```
delayMicroseconds(480);
```

```
pulseIR(600);
```

```
delayMicroseconds(1580);
```

```
pulseIR(600);
```

```
delayMicroseconds(1580);
```

```
pulseIR(620);
```

```
delayMicroseconds(1580);
```

```
pulseIR(600);
```

```
delayMicroseconds(39240);
```

```
pulseIR(8860);

delayMicroseconds(2140);

pulseIR(600);

delayMicroseconds(28424);

pulseIR(8840);

delayMicroseconds(2140);

pulseIR(620);

delayMicroseconds(28424);

pulseIR(8840);

delayMicroseconds(2140);

pulseIR(620);

}

}
```

void Button3()// function that have the duty cycle of the pulse of the Infra-Red Signal to
Control the Channel Button 3

```
delayMicroseconds(43456);

pulseIR(8860);

delayMicroseconds(4320);

pulseIR(620);

delayMicroseconds(480);

pulseIR(620);

delayMicroseconds(480);

pulseIR(600);

delayMicroseconds(1580);

pulseIR(600);
```

```
delayMicroseconds(500);  
pulseIR(600);  
delayMicroseconds(480);  
pulseIR(620);  
delayMicroseconds(480);  
pulseIR(600);  
delayMicroseconds(480);  
pulseIR(620);  
delayMicroseconds(480);  
pulseIR(600);  
delayMicroseconds(1580);  
pulseIR(600);  
delayMicroseconds(1580);  
pulseIR(620);  
delayMicroseconds(460);  
pulseIR(620);  
delayMicroseconds(1580);  
pulseIR(600);  
delayMicroseconds(1580);  
pulseIR(620);  
delayMicroseconds(1560);  
pulseIR(620);  
delayMicroseconds(1560);  
pulseIR(620);
```

```
delayMicroseconds(1580);
pulseIR(600);
delayMicroseconds(1580);
pulseIR(620);
delayMicroseconds(1560);
pulseIR(620);
delayMicroseconds(480);
pulseIR(600);
delayMicroseconds(500);
pulseIR(600);
delayMicroseconds(1580);
pulseIR(600);
delayMicroseconds(480);
pulseIR(620);
delayMicroseconds(480);
pulseIR(600);
delayMicroseconds(480);
pulseIR(620);
delayMicroseconds(480);
pulseIR(600);
delayMicroseconds(480);
pulseIR(620);
delayMicroseconds(1560);
pulseIR(620);
```

```
delayMicroseconds(1580);
pulseIR(600);
delayMicroseconds(480);
pulseIR(620);
delayMicroseconds(1560);
pulseIR(620);
delayMicroseconds(1580);
pulseIR(600);
delayMicroseconds(1580);
pulseIR(600);
delayMicroseconds(39260);
pulseIR(8840);
delayMicroseconds(2140);
pulseIR(620);
delayMicroseconds(28404);
pulseIR(8860);
delayMicroseconds(2140);
pulseIR(620);
delayMicroseconds(28404);
pulseIR(8860);
delayMicroseconds(2140);
pulseIR(620);

}
```

```
void Button4(){// function that have the duty cycle of the pulse of the Infra-Red Signal to  
Control the Channel Button 4  
  
delayMicroseconds(52040);  
  
pulseIR(8840);  
  
delayMicroseconds(4340);  
  
pulseIR(620);  
  
delayMicroseconds(460);  
  
pulseIR(620);  
  
delayMicroseconds(480);  
  
pulseIR(620);  
  
delayMicroseconds(1560);  
  
pulseIR(620);  
  
delayMicroseconds(480);  
  
pulseIR(620);  
  
delayMicroseconds(460);  
  
pulseIR(620);  
  
delayMicroseconds(480);  
  
pulseIR(600);  
  
delayMicroseconds(480);  
  
pulseIR(620);  
  
delayMicroseconds(480);  
  
pulseIR(600);  
  
delayMicroseconds(1580);  
  
pulseIR(620);
```

```
delayMicroseconds(1560);
pulseIR(620);
delayMicroseconds(480);
pulseIR(600);
delayMicroseconds(1580);
pulseIR(620);
delayMicroseconds(1560);
pulseIR(620);
delayMicroseconds(1580);
pulseIR(600);
delayMicroseconds(1580);
pulseIR(600);
delayMicroseconds(1580);
pulseIR(620);
delayMicroseconds(480);
pulseIR(600);
delayMicroseconds(480);
pulseIR(620);
delayMicroseconds(1560);
pulseIR(620);
delayMicroseconds(480);
pulseIR(600);
delayMicroseconds(1580);
pulseIR(620);
```

```
delayMicroseconds(480);  
pulseIR(600);  
delayMicroseconds(480);  
pulseIR(620);  
delayMicroseconds(480);  
pulseIR(600);  
delayMicroseconds(1580);  
pulseIR(620);  
delayMicroseconds(1560);  
pulseIR(620);  
delayMicroseconds(480);  
pulseIR(600);  
delayMicroseconds(1580);  
pulseIR(600);  
delayMicroseconds(500);  
pulseIR(600);  
delayMicroseconds(1580);  
pulseIR(600);  
delayMicroseconds(1580);  
pulseIR(620);  
delayMicroseconds(1560);  
pulseIR(620);  
delayMicroseconds(39240);  
pulseIR(8860);
```

```
delayMicroseconds(2140);
pulseIR(600);
delayMicroseconds(28424);
pulseIR(8860);
delayMicroseconds(2140);
pulseIR(600);
}

void Button5(){// function that have the duty cycle of the pulse of the Infra-Red Signal to
Control the Channel Button 5

delayMicroseconds(46980);
pulseIR(8840);
delayMicroseconds(4340);
pulseIR(600);
delayMicroseconds(500);
pulseIR(600);
delayMicroseconds(480);
pulseIR(620);
delayMicroseconds(1560);
pulseIR(620);
delayMicroseconds(480);
pulseIR(600);
delayMicroseconds(480);
pulseIR(620);
delayMicroseconds(480);
```

```
pulseIR(600);

delayMicroseconds(480);

pulseIR(620);

delayMicroseconds(480);

pulseIR(600);

delayMicroseconds(1580);

pulseIR(600);

delayMicroseconds(1580);

pulseIR(620);

delayMicroseconds(480);

pulseIR(600);

delayMicroseconds(1580);

pulseIR(620);

delayMicroseconds(1560);

pulseIR(620);

delayMicroseconds(1560);

pulseIR(620);

delayMicroseconds(1540);

pulseIR(640);

delayMicroseconds(1580);

pulseIR(620);

delayMicroseconds(1560);

pulseIR(620);

delayMicroseconds(480);
```

```
pulseIR(600);

delayMicroseconds(1580);

pulseIR(600);

delayMicroseconds(500);

pulseIR(600);

delayMicroseconds(1580);

pulseIR(600);

delayMicroseconds(480);

pulseIR(620);

delayMicroseconds(480);

pulseIR(600);

delayMicroseconds(480);

pulseIR(620);

delayMicroseconds(480);

pulseIR(600);

delayMicroseconds(1580);

pulseIR(620);

delayMicroseconds(480);

pulseIR(600);

delayMicroseconds(1580);

pulseIR(600);

delayMicroseconds(480);

pulseIR(620);

delayMicroseconds(1580);
```

```
pulseIR(600);
delayMicroseconds(1580);
pulseIR(600);
delayMicroseconds(1580);
pulseIR(620);
delayMicroseconds(39240);
pulseIR(8840);
delayMicroseconds(2160);
pulseIR(600);
delayMicroseconds(28424);
pulseIR(8840);
delayMicroseconds(2160);
pulseIR(600);

}
```

void Button6(){// function that have the duty cycle of the pulse of the Infra-Red Signal to
Control the Channel Button 6

```
delayMicroseconds(31980);
pulseIR(8840);
delayMicroseconds(4340);
pulseIR(600);
delayMicroseconds(480);
pulseIR(620);
delayMicroseconds(480);
```

```
pulseIR(600);
delayMicroseconds(1580);
pulseIR(620);
delayMicroseconds(480);
pulseIR(600);
delayMicroseconds(480);
pulseIR(620);
delayMicroseconds(480);
pulseIR(600);
delayMicroseconds(480);
pulseIR(620);
delayMicroseconds(480);
pulseIR(600);
delayMicroseconds(1580);
pulseIR(600);
delayMicroseconds(1580);
pulseIR(620);
delayMicroseconds(460);
pulseIR(620);
delayMicroseconds(1580);
pulseIR(620);
delayMicroseconds(1560);
pulseIR(620);
delayMicroseconds(1580);
```

```
pulseIR(600);

delayMicroseconds(1580);

pulseIR(600);

delayMicroseconds(1580);

pulseIR(600);

delayMicroseconds(480);

pulseIR(620);

delayMicroseconds(1580);

pulseIR(600);

delayMicroseconds(1580);

pulseIR(600);

delayMicroseconds(500);

pulseIR(600);

delayMicroseconds(1580);

pulseIR(600);

delayMicroseconds(480);

pulseIR(620);

delayMicroseconds(480);

pulseIR(600);

delayMicroseconds(480);

pulseIR(620);

delayMicroseconds(1560);

pulseIR(620);

delayMicroseconds(480);
```

```
pulseIR(620);

delayMicroseconds(480);

pulseIR(600);

delayMicroseconds(1580);

pulseIR(600);

delayMicroseconds(480);

pulseIR(620);

delayMicroseconds(1580);

pulseIR(600);

delayMicroseconds(1580);

pulseIR(600);

delayMicroseconds(1580);

pulseIR(620);

delayMicroseconds(39240);

pulseIR(8840);

delayMicroseconds(2160);

pulseIR(600);

delayMicroseconds(28424);

pulseIR(8840);

delayMicroseconds(2160);

pulseIR(600);

delayMicroseconds(28424);

pulseIR(8840);

delayMicroseconds(2140);
```

```
pulseIR(620);

delayMicroseconds(28404);

pulseIR(8860);

delayMicroseconds(2140);

pulseIR(620);

}

void Button7(){// function that have the duty cycle of the pulse of the Infra-Red Signal to
Control the Channel Button 7

delayMicroseconds(52088);

pulseIR(8860);

delayMicroseconds(4340);

pulseIR(600);

delayMicroseconds(480);

pulseIR(620);

delayMicroseconds(480);

pulseIR(600);

delayMicroseconds(1580);

pulseIR(620);

delayMicroseconds(480);

pulseIR(600);

delayMicroseconds(480);

pulseIR(620);

delayMicroseconds(480);
```

```
pulseIR(600);

delayMicroseconds(480);

pulseIR(620);

delayMicroseconds(480);

pulseIR(600);

delayMicroseconds(1580);

pulseIR(600);

delayMicroseconds(1580);

pulseIR(620);

delayMicroseconds(480);

pulseIR(600);

delayMicroseconds(1580);

pulseIR(620);

delayMicroseconds(1560);

pulseIR(620);

delayMicroseconds(1560);

pulseIR(620);

delayMicroseconds(1580);

pulseIR(600);

delayMicroseconds(1580);

pulseIR(600);

delayMicroseconds(1580);

pulseIR(620);

delayMicroseconds(1580);
```

```
pulseIR(600);

delayMicroseconds(1580);

pulseIR(600);

delayMicroseconds(480);

pulseIR(620);

delayMicroseconds(1580);

pulseIR(600);

delayMicroseconds(480);

pulseIR(620);

delayMicroseconds(480);

pulseIR(600);

delayMicroseconds(480);

pulseIR(620);

delayMicroseconds(480);

pulseIR(600);

delayMicroseconds(480);

pulseIR(620);

delayMicroseconds(480);

pulseIR(600);

delayMicroseconds(1580);

pulseIR(620);

delayMicroseconds(460);

pulseIR(620);

delayMicroseconds(1580);
```

```
pulseIR(600);

delayMicroseconds(1580);

pulseIR(600);

delayMicroseconds(1580);

pulseIR(620);

delayMicroseconds(39240);

pulseIR(8860);

delayMicroseconds(2140);

pulseIR(600);

delayMicroseconds(28424);

pulseIR(8840);

delayMicroseconds(2140);

pulseIR(620);

}

void Button8(){// function that have the duty cycle of the pulse of the Infra-Red Signal to
Control the Channel Button 8

delayMicroseconds(33036);

pulseIR(8840);

delayMicroseconds(4340);

pulseIR(620);

delayMicroseconds(480);

pulseIR(600);

delayMicroseconds(480);

pulseIR(620);
```

```
delayMicroseconds(1560);
pulseIR(620);
delayMicroseconds(480);
pulseIR(620);
delayMicroseconds(460);
pulseIR(620);
delayMicroseconds(480);
pulseIR(600);
delayMicroseconds(500);
pulseIR(600);
delayMicroseconds(480);
pulseIR(600);
delayMicroseconds(1580);
pulseIR(620);
delayMicroseconds(1560);
pulseIR(620);
delayMicroseconds(480);
pulseIR(620);
delayMicroseconds(1560);
pulseIR(620);
delayMicroseconds(1560);
pulseIR(620);
delayMicroseconds(1580);
pulseIR(600);
```

```
delayMicroseconds(1580);
pulseIR(620);
delayMicroseconds(1560);
pulseIR(620);
delayMicroseconds(480);
pulseIR(600);
delayMicroseconds(480);
pulseIR(620);
delayMicroseconds(480);
pulseIR(600);
delayMicroseconds(1580);
pulseIR(620);
delayMicroseconds(1560);
pulseIR(620);
delayMicroseconds(480);
pulseIR(600);
delayMicroseconds(500);
pulseIR(600);
delayMicroseconds(480);
pulseIR(600);
delayMicroseconds(1580);
pulseIR(620);
delayMicroseconds(1560);
pulseIR(620);
```

```
delayMicroseconds(1560);
pulseIR(620);
delayMicroseconds(480);
pulseIR(620);
delayMicroseconds(480);
pulseIR(600);
delayMicroseconds(1580);
pulseIR(600);
delayMicroseconds(1580);
pulseIR(620);
delayMicroseconds(1560);
pulseIR(620);
delayMicroseconds(39240);
pulseIR(8860);
delayMicroseconds(2140);
pulseIR(620);
delayMicroseconds(28404);
pulseIR(8860);
delayMicroseconds(2140);
pulseIR(600);
}

void Button9(){// function that have the duty cycle of the pulse of the Infra-Red Signal to
Control the Channel Button 9
delayMicroseconds(57508);
```

```
pulseIR(8840);

delayMicroseconds(4340);

pulseIR(600);

delayMicroseconds(500);

pulseIR(600);

delayMicroseconds(480);

pulseIR(600);

delayMicroseconds(1580);

pulseIR(620);

delayMicroseconds(480);

pulseIR(600);

delayMicroseconds(480);

pulseIR(620);

delayMicroseconds(480);

pulseIR(600);

delayMicroseconds(480);

pulseIR(620);

delayMicroseconds(480);

pulseIR(600);

delayMicroseconds(1580);

pulseIR(620);

delayMicroseconds(1560);

pulseIR(620);

delayMicroseconds(480);
```

```
pulseIR(600);
delayMicroseconds(1580);
pulseIR(620);
delayMicroseconds(1560);
pulseIR(620);
delayMicroseconds(1580);
pulseIR(600);
delayMicroseconds(1580);
pulseIR(600);
delayMicroseconds(1580);
pulseIR(620);
delayMicroseconds(1560);
pulseIR(620);
delayMicroseconds(480);
pulseIR(600);
delayMicroseconds(480);
pulseIR(620);
delayMicroseconds(1580);
pulseIR(600);
delayMicroseconds(1580);
pulseIR(600);
delayMicroseconds(480);
pulseIR(620);
delayMicroseconds(480);
```

```
pulseIR(620);

delayMicroseconds(460);

pulseIR(620);

delayMicroseconds(480);

pulseIR(620);

delayMicroseconds(1560);

pulseIR(620);

delayMicroseconds(1560);

pulseIR(620);

delayMicroseconds(480);

pulseIR(600);

delayMicroseconds(500);

pulseIR(600);

delayMicroseconds(1580);

pulseIR(600);

delayMicroseconds(1580);

pulseIR(620);

delayMicroseconds(1560);

pulseIR(620);

delayMicroseconds(39240);

pulseIR(8860);

delayMicroseconds(2140);

pulseIR(600);

delayMicroseconds(28424);
```

```
pulseIR(8860);

delayMicroseconds(2140);

pulseIR(600);

delayMicroseconds(28424);

pulseIR(8860);

delayMicroseconds(2140);

pulseIR(600);

}

void Button0(){// function that have the duty cycle of the pulse of the Infra-Red Signal to
Control the Channel Button 0

delayMicroseconds(4948);

pulseIR(8840);

delayMicroseconds(4340);

pulseIR(620);

delayMicroseconds(480);

pulseIR(600);

delayMicroseconds(480);

pulseIR(620);

delayMicroseconds(1560);

pulseIR(620);

delayMicroseconds(480);

pulseIR(600);

delayMicroseconds(480);

pulseIR(620);
```

```
delayMicroseconds(480);  
pulseIR(620);  
delayMicroseconds(460);  
pulseIR(620);  
delayMicroseconds(480);  
pulseIR(600);  
delayMicroseconds(1580);  
pulseIR(620);  
delayMicroseconds(1560);  
pulseIR(620);  
delayMicroseconds(480);  
pulseIR(600);  
delayMicroseconds(1580);  
pulseIR(620);  
delayMicroseconds(1560);  
pulseIR(620);  
delayMicroseconds(1560);  
pulseIR(620);  
delayMicroseconds(1580);  
pulseIR(600);  
delayMicroseconds(1580);  
pulseIR(620);  
delayMicroseconds(480);  
pulseIR(600);
```

```
delayMicroseconds(480);  
pulseIR(620);  
delayMicroseconds(480);  
pulseIR(620);  
delayMicroseconds(460);  
pulseIR(620);  
delayMicroseconds(1560);  
pulseIR(620);  
delayMicroseconds(480);  
pulseIR(620);  
delayMicroseconds(480);  
pulseIR(600);  
delayMicroseconds(480);  
pulseIR(620);  
delayMicroseconds(1560);  
pulseIR(620);  
delayMicroseconds(1580);  
pulseIR(600);  
delayMicroseconds(1580);  
pulseIR(620);  
delayMicroseconds(1560);  
pulseIR(620);  
delayMicroseconds(480);  
pulseIR(600);
```

```
delayMicroseconds(1580);
pulseIR(600);
delayMicroseconds(1580);
pulseIR(620);
delayMicroseconds(1580);
pulseIR(600);
delayMicroseconds(39240);
pulseIR(8860);
delayMicroseconds(2140);
pulseIR(600);
delayMicroseconds(28424);
pulseIR(8860);
delayMicroseconds(2140);
pulseIR(620);
}

void Input(){// function that have the duty cycle of the pulse of the Infra-Red Signal to Control
the Input video source
delayMicroseconds(5020);
pulseIR(8840);
delayMicroseconds(4340);
pulseIR(620);
delayMicroseconds(480);
pulseIR(600);
delayMicroseconds(480);
```

```
pulseIR(620);

delayMicroseconds(1560);

pulseIR(620);

delayMicroseconds(480);

pulseIR(620);

delayMicroseconds(460);

pulseIR(620);

delayMicroseconds(480);

pulseIR(620);

delayMicroseconds(460);

pulseIR(620);

delayMicroseconds(480);

pulseIR(600);

delayMicroseconds(1580);

pulseIR(620);

delayMicroseconds(1580);

pulseIR(600);

delayMicroseconds(480);

pulseIR(620);

delayMicroseconds(1560);

pulseIR(620);

delayMicroseconds(1560);

pulseIR(620);

delayMicroseconds(1580);
```

```
pulseIR(600);
delayMicroseconds(1580);
pulseIR(620);
delayMicroseconds(1560);
pulseIR(620);
delayMicroseconds(1580);
pulseIR(600);
delayMicroseconds(1580);
pulseIR(620);
delayMicroseconds(1560);
pulseIR(620);
delayMicroseconds(1560);
pulseIR(620);
delayMicroseconds(480);
pulseIR(600);
delayMicroseconds(1580);
pulseIR(620);
delayMicroseconds(480);
pulseIR(600);
delayMicroseconds(480);
pulseIR(620);
delayMicroseconds(480);
pulseIR(600);
delayMicroseconds(480);
```

```
pulseIR(620);

delayMicroseconds(480);

pulseIR(600);

delayMicroseconds(480);

pulseIR(620);

delayMicroseconds(1560);

pulseIR(620);

delayMicroseconds(480);

pulseIR(620);

delayMicroseconds(1560);

pulseIR(620);

delayMicroseconds(1560);

pulseIR(620);

delayMicroseconds(39240);

pulseIR(8860);

delayMicroseconds(2140);

pulseIR(620);

delayMicroseconds(28404);

pulseIR(8860);

delayMicroseconds(2140);

pulseIR(620);

}

void Exit()// function that have the duty cycle of the pulse of the Infra-Red Signal to Control  
the Exit of eny menu
```

```
delayMicroseconds(4672);
pulseIR(8860);
delayMicroseconds(4320);
pulseIR(620);
delayMicroseconds(480);
pulseIR(600);
delayMicroseconds(480);
pulseIR(620);
delayMicroseconds(1580);
pulseIR(600);
delayMicroseconds(480);
pulseIR(620);
delayMicroseconds(480);
pulseIR(600);
delayMicroseconds(480);
pulseIR(620);
delayMicroseconds(480);
pulseIR(600);
delayMicroseconds(480);
pulseIR(620);
delayMicroseconds(1560);
pulseIR(620);
delayMicroseconds(1580);
pulseIR(600);
```

```
delayMicroseconds(480);  
pulseIR(620);  
delayMicroseconds(1580);  
pulseIR(600);  
delayMicroseconds(1580);  
pulseIR(620);  
delayMicroseconds(1560);  
pulseIR(620);  
delayMicroseconds(1560);  
pulseIR(620);  
delayMicroseconds(1580);  
pulseIR(600);  
delayMicroseconds(1580);  
pulseIR(620);  
delayMicroseconds(480);  
pulseIR(600);  
delayMicroseconds(480);  
pulseIR(620);  
delayMicroseconds(1560);  
pulseIR(620);  
delayMicroseconds(480);  
pulseIR(600);  
delayMicroseconds(480);  
pulseIR(620);
```

```
delayMicroseconds(1580);
pulseIR(600);
delayMicroseconds(480);
pulseIR(620);
delayMicroseconds(480);
pulseIR(600);
delayMicroseconds(1580);
pulseIR(600);
delayMicroseconds(1580);
pulseIR(620);
delayMicroseconds(480);
pulseIR(600);
delayMicroseconds(1580);
pulseIR(620);
delayMicroseconds(1560);
pulseIR(620);
delayMicroseconds(480);
pulseIR(600);
delayMicroseconds(1580);
pulseIR(600);
delayMicroseconds(39260);
pulseIR(8860);
delayMicroseconds(2140);
pulseIR(620);
```

```
delayMicroseconds(28404);
pulseIR(8860);
delayMicroseconds(2140);
pulseIR(620);

}

void Menu(){// function that have the duty cycle of the pulse of the Infra-Red Signal to access
the Menu of the tv

delayMicroseconds(44220);
pulseIR(8860);
delayMicroseconds(4340);
pulseIR(620);
delayMicroseconds(480);
pulseIR(600);
delayMicroseconds(480);
pulseIR(620);
delayMicroseconds(1560);
pulseIR(620);
delayMicroseconds(480);
pulseIR(600);
delayMicroseconds(480);
pulseIR(620);
delayMicroseconds(480);
pulseIR(600);
```

```
delayMicroseconds(480);  
pulseIR(620);  
delayMicroseconds(480);  
pulseIR(600);  
delayMicroseconds(1580);  
pulseIR(620);  
delayMicroseconds(1560);  
pulseIR(620);  
delayMicroseconds(480);  
pulseIR(600);  
delayMicroseconds(1580);  
pulseIR(620);  
delayMicroseconds(1560);  
pulseIR(620);  
delayMicroseconds(1580);  
pulseIR(600);  
delayMicroseconds(1580);  
pulseIR(620);  
delayMicroseconds(1560);  
pulseIR(620);  
delayMicroseconds(1560);  
pulseIR(620);  
delayMicroseconds(1580);  
pulseIR(600);
```

```
delayMicroseconds(480);

pulseIR(620);

delayMicroseconds(480);

pulseIR(600);

delayMicroseconds(480);

pulseIR(620);

delayMicroseconds(480);

pulseIR(600);

delayMicroseconds(1580);

pulseIR(620);

delayMicroseconds(480);

pulseIR(600);

delayMicroseconds(480);

pulseIR(620);

delayMicroseconds(480);

pulseIR(600);

delayMicroseconds(1580);

pulseIR(600);

delayMicroseconds(1580);

pulseIR(620);

delayMicroseconds(1560);

pulseIR(620);

delayMicroseconds(1580);

pulseIR(600);
```

```
delayMicroseconds(480);  
pulseIR(620);  
delayMicroseconds(1580);  
pulseIR(600);  
delayMicroseconds(39240);  
pulseIR(8860);  
delayMicroseconds(2140);  
pulseIR(620);  
delayMicroseconds(28424);  
pulseIR(8860);  
delayMicroseconds(2120);  
pulseIR(620);}
```

Chapter 8: Energy Monitor

8.1 Alternatives Considered

For the Energy Monitor system implementation, a very important consideration is that the microcontroller must have analog pins. For this reason, the Raspberry Pi it is not an option for the energy monitor system. Another option is the Beaglebone Black, but to maintain a standard in the system we decided to use the Arduino.

8.2 Mathematical Justification

Used Formulas:

$$Voltage Read_{(volts)} = Value Read_{(data)} \times \left(\frac{5_{(volts)}}{1023_{(data)}} \right)$$

$$Voltage_{(RMS)} = \frac{Voltage_{(pp)}}{2} * \frac{1}{\sqrt{2}}$$

$$Current_{(AmpsRMS)} = Voltage_{(RMS)} \times \left(\frac{1000_{(mV)}}{1_{(V)}} \right) \times \left(\frac{1_{(Amp)}}{260_{(mV)}} \right)$$

$$Power_{(Watt)} = Voltage_{(RMS)} \times Current_{(RMS)} \times P.F.$$

$$Energy_{(KWh)} = \frac{Power_{(Watt)} \times Time_{(hr)}}{1000}$$

$$Monthly Consumption_{(KWh/Month)} = \frac{Power_{(Watt)} \times Time_{(hr)}}{1000} \times 30_{(Day)}$$

Rectangular Snip

$$Cost_{(\$)} = \frac{Power_{(Watt)} \times Time_{(hr)}}{1000} \times 30_{(Day)} \times Cost KWh_{(\$)}$$

This was the formula that we used on the program to obtain the current measurement. With this measurement we are able to calculate the power in KWh and estimate the monthly consumption. To maintain the Homeowner inform in real time of the consumption of the home. This is possible with the UDP communication between the energy monitor system and the android App.

8.3 System Specifications

To monitor the current we use a module ACS712-5B presented in Figure. This sensor is an economical solution to measure the AC current. This sensor read an equivalent voltage proportional to the current.

The device consist of a precise, linear Hall circuit with a copper conduction path located near the surface or the die. The applied current flowing through this copper conduction path generates a magnetic field which the Hall IC converts into a proportional voltage. Device accuracy is optimized through the close proximity of the magnetic signal to the Hall transducer.

The energy monitor reads the current value and converts it into a relevant voltage value. The sensor can measure positive and negative currents, and uses a power supply of 5V.



Figure 8.3.1 - ACS712-05B Current Sensor Module



Figure 8.3.2 –Ethernet Shield Module

To establish a communication with the Energy Monitor System implemented on the Arduino. We need to use the Ethernet shield Module. To establish a communication with the Android app and the data base on the main controller. This server can communicate the app to maintain the user informed of the actual consumption of the house.

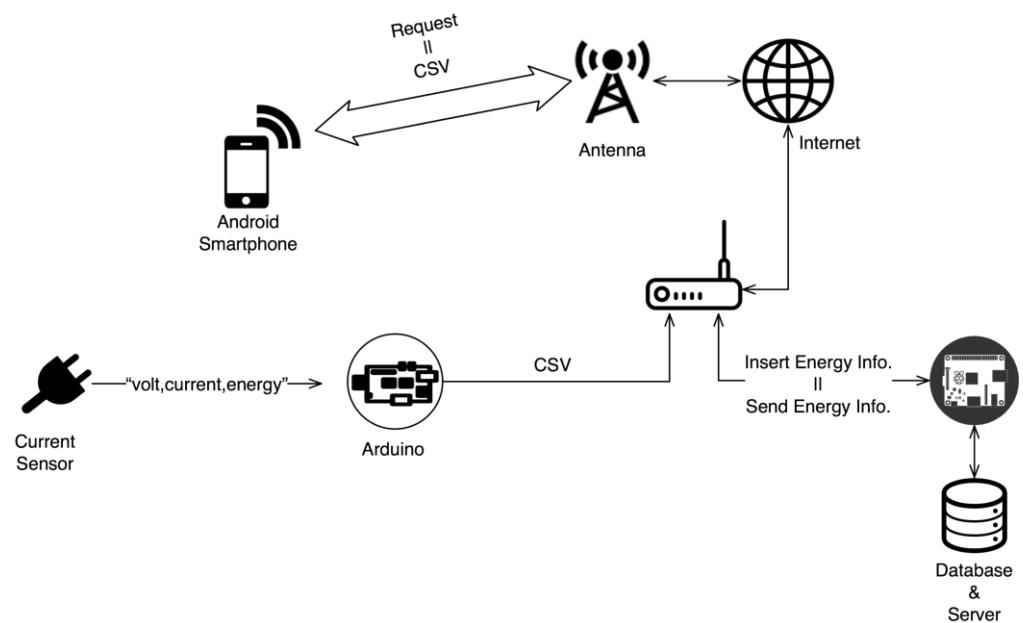


Figure 8.3.3 – Energy Monitor Block Diagram Conceptual Design

Here we have a schematic how we connect our system with the cellphone app. The current sensor read the value of the current. Then go to the Arduino controller and make the calculation of the power and energy consumption. This data is send to the database located in the raspberry pi main processor. When the user request the data the app access the database and collect the data and display on the android app.

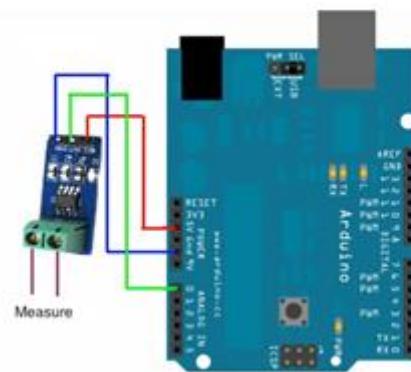


Figure 8.3.4 – Current Sensor Connection to the Arduino

On this image we can see how we connect the current sensor to the Arduino. We need to supply to the sensor 5 volts, ground and a connection to one of the analog pin of the Arduino.

Functional Block Diagram

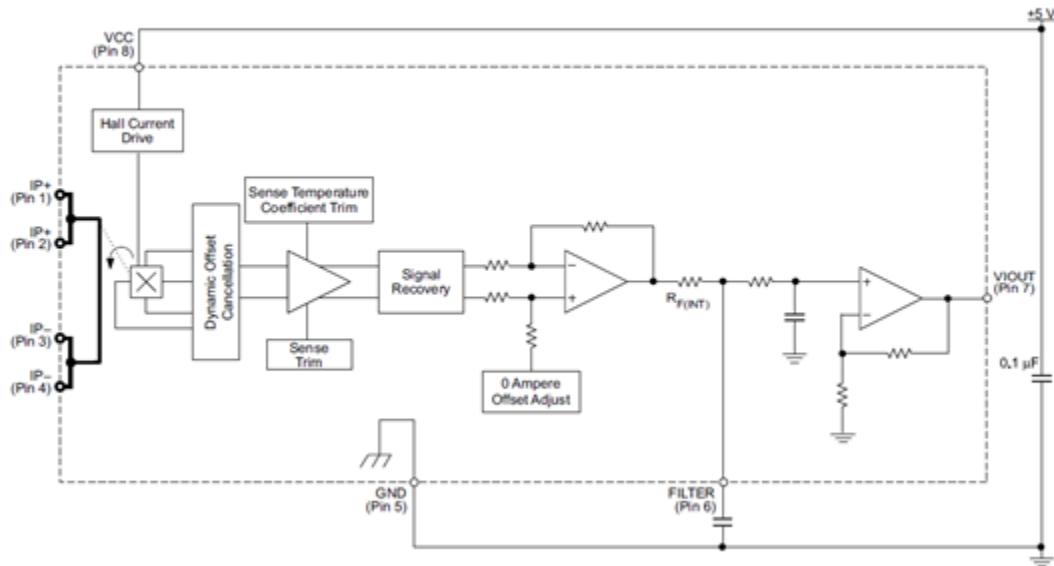


Figure 8.3.5 – Current Sensor Integrated Circuit Internal Design

This diagram shows us the internal design of the integrated circuit current sensor. The pin 1 and 2 (IP+) are the terminals for current being sampled, fused internally. The pin 3 and 4 (IP-) are the terminals for current being sampled, fused internally. The pin 5 is the signal ground terminal. The pin 6 is used to filter. You can connect an external capacitor to set the bandwidth. The pin 7 is the analog signal out pin. The pin 8 is the Device power supply terminal

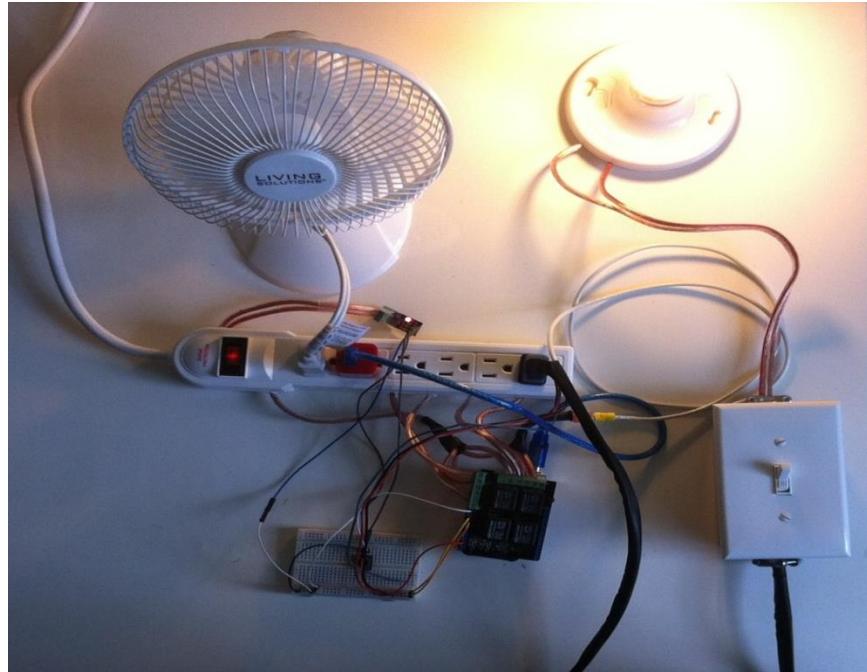


Figure 6.3.5 – Energy monitor System Implementation

This is an implementation test that we do to prove the concept of communication to prove the Arduino code. On this Photo we show the energy monitor system implemented together with the lighting system. We implement the two system together in the same Arduino to save money. With this implementation we prove if our circuit was built/designed in order to fulfill our requirements.

8.4 Economic Analysis

For the energy monitor system we have 3 basic component, a current sensor a Arduino mega and the Ethernet shield module. We implement this system on a scale sized. To implement this system we modify a multiplug and we put the sensor. For the purpose of this project, the components chosen were bought on online stores, and their respective prices were:

Equipment	Purpose	Price
Arduino uno	The core of the system	1 pcs for \$ 20.99
Ethernet shield	To cnect the system with the internet	1pcs for \$ 17.60
Current sensor	To measure the energy compsumption	1pcs for \$ 2.80
Multi plug	To put the invasive current sensor	1pcs for \$10.00
Cable	To make the electrical connection	\$ 8.00
Heat Shrink	To protect the cable to make a short circuit	\$ 5.95

8.5 Results

Current (Amp)	Induced Voltage (mV)
0	56
0.19	75
0.44	112
0.64	283
0.82	177
0.86	179.51
1.19	349
1.43	393
1.93	453

Figure 8.5.1 – Current and Induced Voltage Read by the Sensor

We connect different load to the energy monitor system and measure the current with the help of a multi meter and we compare with the voltage read of the current sensor. With this data we find a linear regression and we obtain the following chart. We obtain that the gain of the sensor to convert from millivolt to the current measurement is the slope of the line that is 259.84 mv per Amps. We obtain a linear collation of the data and the mathematical model of 0.8475 that is good. They have certain error that the manufacturer says that can be temperature fluctuation.

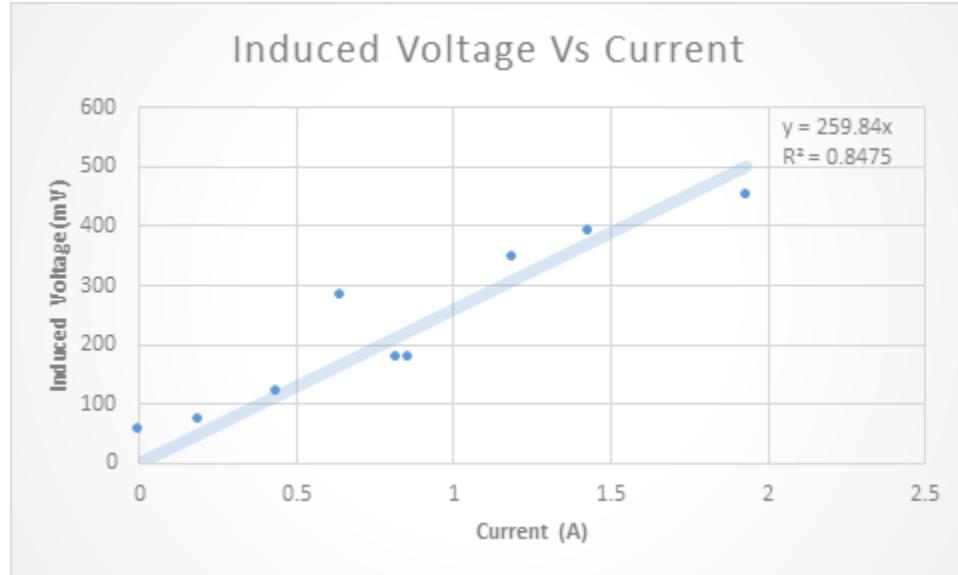


Figure 8.5.2 – Graph of Current Measurement and the Linear Regression

Real value	Measure value	Percent Error
0.11	0.11	0%
0.14	0.14	0%
0.22	0.24	4.545455%
0.87	0.86	1.149425%
0.94	0.86	5.319149%
1.12	1.05	6.25%
1.93	1.86	3.626943%
2.04	1.99	2.45098%

Figure 8.5.3 – Table Of the Percent Error

This table shows the difference between the real value and the measure value. We calculate the percent of Error for each measurement. We observe than the lower percent error is 0% of error. The greater percent error is 5.3% that is the specified percent of error given by the manufacturer. The 5% of error not is very significant because we don't need an extremely accurate measurement to estimate the energy consumption. We conclude that this sensor give accurate measurement.

This program contains the code of the energy monitor system and the code of the lightning system that are implemented in a single code. This program also has the implementation of the socket connection to communicate with the android app.

8.5.3. Energy Monitor Code for Arduino

This program contains the code of the Energy Monitor system that are implemented in this code. This program also has the implementation of the socket connection to communicate with the android app.

```
// Arduino Code Energy Monitor System

#include <SPI.h>      // needed for Arduino versions later than 0018

#include <Ethernet.h>    // Ethernet shield Library

#include <EthernetUdp.h>    // UDP library from: bjoern@cs.stanford.edu 12/30/2008

/*************************************************************************/
*/
```

```

// Variable

const int sensorIn = A3;

double mVperAmp =260; // Sensor Gain to change from mV to Current Value

double pf=.97; // Power Factor measurement Value

double Vcasa=120; // Voltage measurement of the house

double Voltage = 0;// variable to store the induced voltage from the sensor

double VRMS = 0;// variable to store the RMS voltage Value

double AmpsRMS = 0; // variable to store the current value that the sensor reed

double Watt=0;// variable to store the power in watt

double KWh=0;// variable to store the power compsumption in KWH

double TimeHr=0; //time in Hr

double TimeMin=0;//time in Min

double Cost=0;// cost of the power consumption

double time=0;// time in milli second

double KWhMes=0;// montly consumption

/*************************************************************************/
/*************************************************************************/
// Enter a MAC address and IP address for your controller below.

// The IP address will be dependent on your local network:

byte mac[] = {

  0xEE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };

```

```
IPAddress ip(192,168,1,140); // Local ip of the system  
unsigned int localPort = 10; // local port to listen on  
  
// buffers for receiving and sending data  
char packetBuffer[UDP_TX_PACKET_MAX_SIZE]; //buffer to hold incoming packet,  
char ReplyBuffer[] = "acknowledged"; // a string to send back  
  
// An EthernetUDP instance to let us send and receive packets over UDP  
EthernetUDP Udp;  
  
void setup() {  
    // start the Ethernet and UDP:  
    Ethernet.begin(mac,ip); // initialize the Ethernet Communication  
    Udp.begin(localPort); // initialize the Udp package  
  
    pinMode(6,OUTPUT); // initialize the pin 6  
    pinMode(7,OUTPUT); // initialize the pin 7  
    pinMode(5,OUTPUT); // initialize the pin 5  
    pinMode(4,OUTPUT); // initialize the pin 4  
    pinMode(3,OUTPUT); // initialize the pin 3
```

```

pinMode(2,OUTPUT);// initialize the pin 2

Serial.begin(9600); // initialize the serial communication

}

void loop() {

// if there's data available, read a packet

int packetSize = Udp.parsePacket();

/***********************/

***** /



time= millis();// time in measurement store

TimeMin=time/(1000); // time in minute calculation

TimeHr=TimeMin/(60*60);// time in hour calculation

Voltage = getVPP();// store the value of the voltage induced by the current sensor

VRMS = (Voltage/2.0) *0.707; // calculate the Voltage RMS

AmpsRMS = (((VRMS * 1000)/mVperAmp)-0.47); // calculation of the current value

```

```

Watt=(Vcasa*AmpsRMS*pf);// calculate the power in watt

KWh=((Watt*TimeHr)/1000);// calculate the Kwh

KWhMes=((Watt*24)/1000)*30;// estimate the montly comsumption

Cost= (KWhMes*0.1857);// calculate the cost

char ampsch[10];// array to store the value to be send by the network

char wattch[10];// array to store the value to be send by the network

char kwhch[10];// array to store the value to be send by the network

char kwhmesch[10];// array to store the value to be send by the network

char costch[10];// array to store the value to be send by the network

char spacech[10]=", ":"; // array to store the value to be send by the network

dtostrf(AmpsRMS,1,2,ampsch);// function to change from double to string and store in the array

dtostrf(Watt,1,2,wattch);// function to change from double to string and store in the array

dtostrf(KWh,1,2, kwhch);// function to change from double to string and store in the array

dtostrf(KWhMes,1,2, kwhmesch);// function to change from double to string and store in the array

dtostrf(Cost,1,2, costch);// function to change from double to string and store in the array

 ****
 ****
 ****
 ****

if(packetSize)
{

```

```
Serial.print("Received packet of size "); // Start of the function to print the port and the ip adress  
or the sender of the udp pakage  
  
Serial.println(packetSize);  
  
Serial.print("From ");  
  
IPAddress remote = Udp.remoteIP();  
  
for (int i =0; i < 4; i++)  
{  
    Serial.print(remote[i], DEC);  
  
    if (i < 3)  
    {  
        Serial.print(".");  
    }  
}  
  
Serial.print(", port ");  
  
Serial.println(Udp.remotePort());  
  
  
// read the packet into packetBufffer  
  
Udp.read(packetBuffer,UDP_TX_PACKET_MAX_SIZE);  
  
Serial.println("Contents:");  
  
Serial.println(packetBuffer);  
  
  
Udp.beginPacket(Udp.remoteIP(), Udp.remotePort()); // End of the function to print the port and  
the ip adress or the sender of the udp pakage
```

```
if(strcmp(packetBuffer,"pwr")==0){// If the system receive a udp pakage of pwr the system  
send the current the voltage the power consumption in watt, KWH and the cost to the sender  
  
    Udp.write("120");  
  
    Udp.write(spacech);  
  
    Udp.write(ampsch);  
  
    Udp.write(spacech);  
  
    Udp.write(wattch);  
  
    Udp.write(spacech);  
  
    Udp.write(kwhch);  
  
    Udp.write(spacech);  
  
    Udp.write(kwhmesch);  
  
    Udp.write(spacech);  
  
    Udp.write(costch);  
  
}  
  
else  
  
if(strcmp(packetBuffer,"pw2")==0){// If the system receive a udp pakage of pw2 the system  
send the current the voltage the power in watt to the sender  
  
    Udp.write("120");  
  
    Udp.write(spacech);  
  
    Udp.write(ampsch);  
  
    Udp.write(spacech);
```

```

        Udp.write(wattch);

    }

    memset(packetBuffer,0,sizeof(packetBuffer));

    // send a reply, to the IP address and port that sent us the packet we received

    Udp.endPacket();

}

}

*******/

*******/

float getVPP()// Function to convert the analog data reed to a voltage peak to peak
{
    float result;          // value in voltage peak to peak
    int readValue;         // value read from the sensor
    int maxValue = 0;      // store max value here
}

```

```

int minValue = 1024;      // store min value here

uint32_t start_time = millis();

while((millis()-start_time) < 1000) //sample for 1 Sec

{
    readValue = analogRead(sensorIn);

    // see if you have a new maxValue

    if (readValue > maxValue)

    {
        /*record the maximum sensor value*/
        maxValue = readValue;

    }

    if (readValue < minValue)

    {
        /*record the minimum sensor value*/
        minValue = readValue;

    }

}

// Subtract min from max and multiply by the relation to change from data to a voltage value

result = ((maxValue - minValue) * 5.0)/1024.0;

return result;

```

```
}
```

Figure 8.5.3 – Energy Monitor Code for Arduino

Chapter 9: Surveillance

9.1 Alternatives Considered

For the surveillance system, very few options were considered since the intent of this project is to create everything ourselves. Having this mindset, the idea of utilizing already built and functional CCTV systems, IP cameras and others were out of the picture. This left considerations of microcontrollers and the tools that can be used to do live stream. The considerations for this project consisted of two systems that are attachable to the raspberry pi microcontroller and the arduino microcontroller. These two systems include webcam and the arducam, for the arduino, and the raspicam with and without night vision for the raspberry pi. Something to analyze in this project is the requirements for image processing.

9.1.1 Raspberry Pi

The Raspberry Pi microcontroller is a pocket sized computer with a powerful quad-core processor. It is Linux based and has 1 GB of ram. The raspberry pi has a lot of open source resources to program it on the raspberry pi forums website. It can be used as a webserver to transmit the camera livefeed. This consideration is great because of the attachment capabilities in combination with the processing capabilities of this microcontroller.



Figure 9.1.1.1 Raspberry Pi 2

9.1.2 Raspberry Pi NOIR camera module

This camera module is very powerful because it has infrared filters that enable night vision.

An infrared circuit needs to be installed in order for this to be completed. The NOIR camera module allows for full high definition image processing in the desired format. This can be programmed in python by creating scripts or using system commands within php scripts and webpages.



Figure 9.1.2.1 Raspberry Pi NOIR Module

9.1.3 Arduino uno

The arduino uno is very cost effective and has a lot of open source information on the internet which allows for easy references when programming the microcontroller. It has 32 KB of flash memory and 16 MHz single core processor. This board is a great consideration because it is cost effective and can be interfaced with other microcontrollers plus the additional shields that come for it. This microcontroller can be programmed in C.



Figure 9.1.3.1 Arduino Uno

9.1.4 Ardu cam

The ardu cam allows for low resolution image processing. It requires most of the arduino's processing power to process these images alone. Because of this requirement, it makes this consideration not ideal for the RMF Intelligent home system but practical for a quick and simple solution.

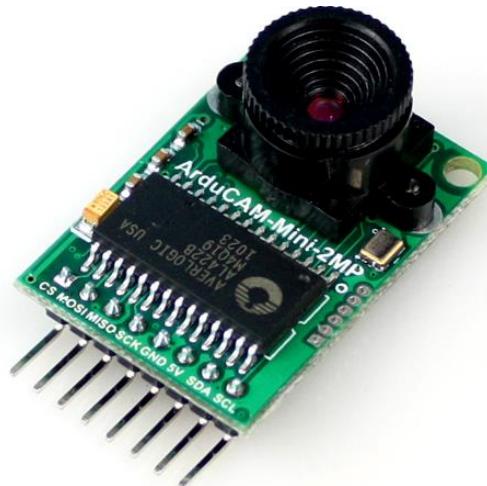


Figure 9.1.4.1 Ardu Cam

9.1.5 Logitech Webcam

The Logitech webcam c615 has the capability of providing high definition transmission up to 1280 by 720 pixels. This webcam can be interfaced with either microcontroller considered. Yet, the same obstacle persists. Most of the arduino uno's processing power is required for the processing of this webcam's images while the raspberry pi can still manage to process these images with ease. This webcam is more costly than the camera modules making it not ideal for the RMF Intelligent home system.



Figure 9.1.4.1 Logitech C615

9.2 System Specifications

The security camera is very complex since it involves image processing. Identifying the constraints versus the tools is the key to success for this part of the system. Using the raspberry pi to overcome the processing constraint is the smart way to go. The raspberry pi has many software tools that make this micro-controller ideal for the system. Protecting the micro-controller against environmental factors is another challenging factor but can easily be solved with a standard CCTV camera housing. It is the owner's responsibility to delete and download the videos from the application in order to free room for the camera to continue recording. It is also the responsibility of the owner to start the recording. All of these features can be manipulated through the RMF Intelligent Homes application. The following sections will describe in depth the implementation of this approach.

9.2.1 Hardware

As previously mentioned, the Raspberry Pi 2 micro-controller is used for the construction of this security camera. Another piece of hardware utilized is the Raspberry Pi NOIR camera module which has night vision filters. In order to have better placement options for the security camera the wi-fi dongle for the raspberry pi camera is used to connect wirelessly to the home network. The last part of hardware needed for this is an infrared circuit to light up the area to be monitored with infrared lights which are not detected by the human eye. The following figures show representations of the hardware.



Figure 9.2.1.1 – Raspberry-Pi 2



Figure 9.2.1.2 – NOIR Camera Module



Figure 9.2.1.3 – Infrared Led Circuit



Figure 9.2.1.4 – Wi-Fi Dongle

9.3 Economic Analysis

Table 9.3.1: Materials cost Raspberry-pi Method

Product	Amount	Cost
Raspberry Pi 2	1	\$40.95
Wi-Fi Dongle	1	\$7.94
NOIR Camera Module	1	\$28.45
Infrared Led Circuit	1	\$9.99
Camera Housing	1	\$9.29
Total Cost		\$96.62

Table 9.3.2: Materials cost Raspberry-pi Method

Product	Amount	Cost
Arduino Uno	1	\$19.95
Ethernet Shield	1	\$29.95
Arducam	1	\$39.99
Infrared Led Circuit	1	\$9.99
Camera Housing	1	\$9.29
Total Cost		\$109.17

The cost of the raspberry-pi method is not only more cost effective but also better for the surveillance camera performance. The materials used for this particular project camera required

only one of each product listed in the tables above. This method is also better for maintenance since it lasts longer than the arduino uno. It also takes less space inside the camera housing.

9.4 Results

9.4.1 Software and Configuration

In order to process the image data through the network, a streaming protocol is established and utilized already programmed within a driver that is specifically designed for the raspberry pis streaming service. This diver is downloaded from linux-projects.org. The driver name is uv4l and works along with a php web interface. The php web interface controls the scripting data based on the user's decision. It allows for a database visualization where the user can download videos, time lapse, and images taken from the system directly to their device. This is also compatible with traditional web browsers to allow the user to download this data directly into their computer while connected to the home network as well. The uv4l uses ffmpeg protocol to stream the images taken at 30 frames per second and finally stores them as video in the camera's mounted sd card. This raspberry pi micro-controller has a static ip address as well and sends the video feed through port forwarding to the user's device. The following code snippet is the camera configuration step by step.

```
"add" deb http://www.linux-project.org/listing/uv4l\_repo/raspbian/ "to the  
/etc/apt/sources.list"
```

```
sudo apt-get update
```

```
sudo apt-get install uv4l uv4l-raspicam
```

```
(search to install uv4l driver if problems continue)
```

```
sudo apt-get dist-upgrade
```

```
sudo rpi-update
```

```
git clone https://github.com/silvanmelchior/RPi_Cam_Web_Interface.git
```

```
cd RPi_Cam_Web_Interface
```

```
chmod u+x RPi_Cam_Web_Interface_Installer.sh
```

```
./RPi_Cam_Web_Interface_Installer.sh install
```

```
./RPi_Cam_Web_Interface_Installer.sh update
```

```
./RPi_Cam_Web_Interface_Installer.sh install
```

```
./RPi_Cam_Web_Interface_Installer.sh "starts and stops"
```

```
sudo chmod 755 /etc/rc.local
```

```
sudo reboot
```

9.4.2 Materials

In order to protect the security camera from environmental factors, a 6 inch CCTV security camera housing is utilized to enclose the hardware. The housing also provides easy installation when mounting on to walls and awkward angles. The following figure shows the housing used. To Ensure the hardware is kept attached safely inside the housing, a 3-D printed base is installed inside the housing as well.



Figure 9.4.2.1 – Camera Housing

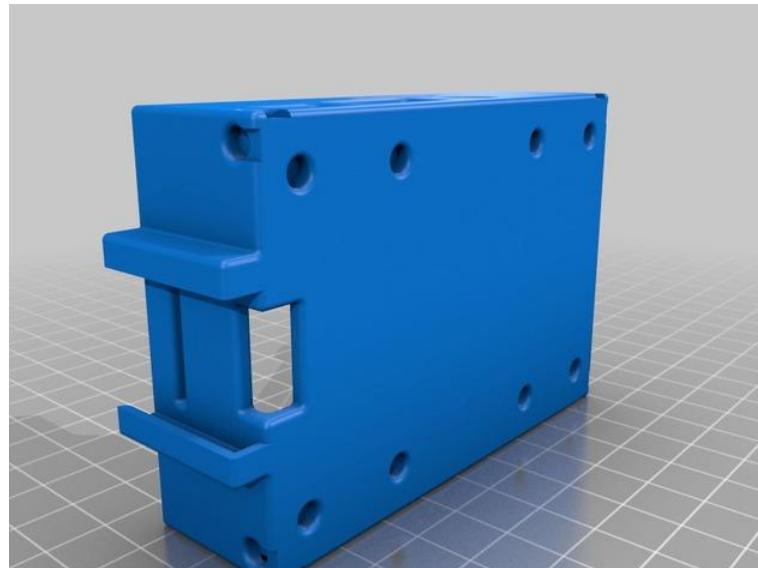


Figure 9.4.2.2 – 3-D Printed Hardware Base

9.4.3 Assembly

Once the camera is verified for its functionality and troubleshoot, it is mounted inside the housing with its base and the hardware connected and mounted to the desire location. The following figures show a brief assembly process of the hardware.

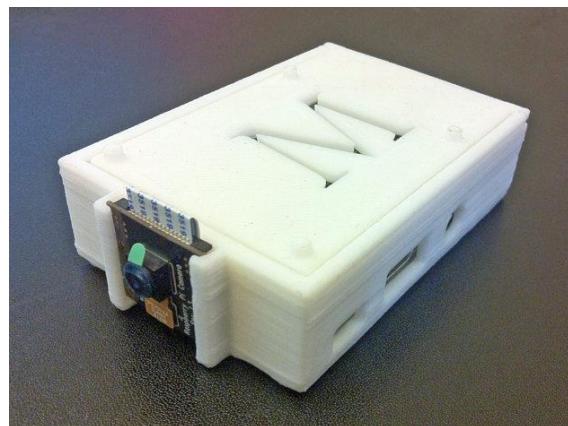


Figure 9.4.3.1 – Hardware On Base



Figure 9.4.3.2 – Security Camera

Chapter 10: Alarm

10.1 Alternatives Considered

For the alarm system construction the alternatives taken in considerations were those of software. On the raspberry pi there is not much languages that can control the GPIO pins on the board. Taking apart that fact of those languages that exist not much is documented about how to manipulate the pins. The languages that exist are python, C++, Java, and a Node Dot js library, these of course are the ones with the most documentation online. There are other methods of controlling the pins but all of the involve tedious work and are not proven to work. All the main languages used for comparison are described on a table below with their respective explanation for the comparison.

The language picked for this system was python because of the proven reviews on the official raspberry pi documentation page. There are various forums suggesting that this is the easiest way to control the board and since the language is not new there are a lot of modules available for its used. Python is a very complete language with a clean syntax that lets you focus on the coding and don't worry about syntax errors, the only catch is that the indentation that the code has is like the brackets in java or C++. All operators are words and that gives the language a very clean and user-friendly appearance.

Programming Language	Comparison	Logo
Python	For python the documentation is vast and there comes a very nice module for the language called RPi.GPIO which is very well documented online and is the one that the books suggest.	
Java	there is a library for java called py4j which is somewhat documented on the internet. The library is often used for application that involve internet pages and web applications.	
Node.js	There is a node.js library called pi-gpio which enables GPIO control on the raspberry pi but needs to be implemented using javascript along with a little server side programming.	

C++

There is a possibility of controlling the GPIO pins on the raspberry pi hardcoding the library using the sys library on c++. This is obviously not directly documented online and is not that easy to find. Some forums online suggest not using this type of control because it can cause problems with the OS on the RPi



10.2 System Specifications

10.2.1 Alarm System Functions and Components

The alarm system designed by RMF Intelligent Homes is an intelligent system which uses both a manual interface and a virtual mobile interface. As any alarm, sensors strategically put around the house will inform the system about changes. If the system detects a change it will be logged in a database and informed to the user via the mobile application. The way this independent system informs the user is by using UDP (User Datagram Protocol), a protocol that is used on socket programming. When the alarm boots the first time a socket is created and the system waits for an incoming connection while asking for the user's input to arm. The user goes through a verification phase which will determine if the alarm is going to be either armed or disarmed. This verification process is the NFC verification system explained on CHAPTER 10 a connection is made the system waits for a packet which will come in form of an array of 3 elements which will be the message, the address and the port that the socket is available on. All processes on the system occur independently, the socket, the alarm, and the arm or disarm processes all are running in each of the cores of the Raspberry Pi 2 processor. A flowchart is shown on Figure 10.2.1, which the parallel logic is included for better understanding.

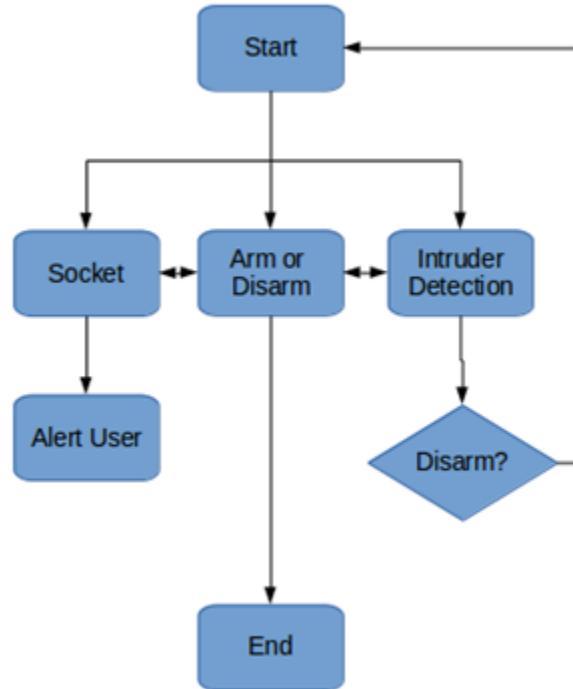


Figure 10.2.1– Alarm System Flow Chart

The sensors that that system has are described on the table below. The magnetic switch is located on doors and windows. The vibration sensor will inform if a window has been broken or tampered with. The passive infrared sensor will monitor for movement once the alarm is armed. Lastly the input panel and buzzer will be the physical interface of the house.

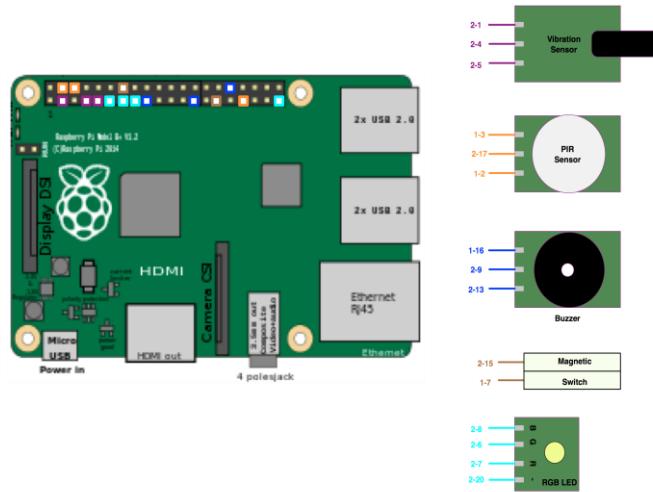
Element	Location
Input Panel	Verification Panel.
Buzzer	Main Alarm Panel.
PIR	On any corner facing the door or window.
Vibration Sensor	Windows.
Magnetic Switch	Doors and Windows.
Camera	Corner near the PIR sensor.

10.3 Economic Analysis

All alarm systems must have at least 4 basic components, a movement sensor, vibration sensors for the windows and fragile structures, magnetic switches for the doors, and an siren of some sort that alerts the user or someone nearby. Since this project is going to be shown on a scale sized environment the siren chosen does not have a loud output. The RMF Intelligent Home proposed alarm system contains these electronic components. For the purpose of this project, the components chosen were bought on online stores, and their respective prices were:

Sensor or Actuator	Purpose	Price
Passive Infrared Sensor (PIR)	Monitor movement and alert the system when motion is detected and alarm is armed.	Box of 10 pcs for \$16.99
Magnetic Switch	Acts as a normally closed button and detects when a door or window is being forced.	\$5.00
Vibration Sensor	Monitor the vibration of sensitive entry points like glass windows, glass doors, etc.	Box of 10 pcs. for \$9.95
Piezo Buzzer	Alerts when any of the sensors is activated by an intrusion or an event that conflicts with the norm occurs.	4 pcs. for \$1.69
Raspberry Pi 2	Core of the System.	\$39.95

10.4 Results



10.4.1 Alarm System Wiring

Figure 10.4.1 Alarm System Connection Diagram

The alarm system's wiring is shown on Figure 10.4.1 on a schematic level. Figure 10.4.2 shows the GPIO pinout of the Raspberry Pi 2 to have a better understanding of the I/O wiring diagram.

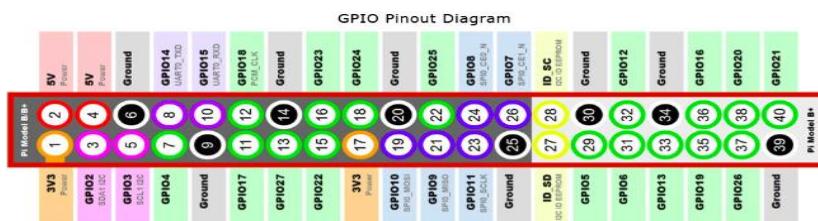


Figure 10.4.1 Raspberry Pi 2 Pinout

10.4.2 Alarm System Code

The system's life is on the firmware is running to control the alarm. This firmware makes the alarm monitor activity, listen to the internet, and function as clearance point between users and system. The code routine that was implemented is shown next and will be explained after.

```
""" RMF Intelligent Homes """
"""
This code was designed as a Capstone
project and is in no way approved by any
regulatory agency. This code is part of
a complete system and will not work other
way. Need at least RMF Mobile App """

"""
 MEMBERS:
Francisco Burgos Collazo id:75483
Manuel Santiago Laboy id.82577
Bryan Maldonado Rodriguez id.73331
Rafael Ruiz Villalobos id.82756
Paolo Jimenez Soto id.63880

"""
#Last Updated: 6/10/15
import RPi.GPIO as GPIO
```

```
import socket
import sys
import multiprocessing as mp
VIBRATION=4
DOOR=5
BUZZER=12
PIR=13
s=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
host='10.42.0.37'
port=3490
s.bind((host,port))
intruder=mp.Value('i',0)#value to inform user of activity
ShutDown=mp.Value('i',1)#value to shut down the system

def setup():
    """
    Sets up the GPIO pins of the Raspberry Pi
    """
    GPIO.setmode(GPIO.BCM)
    GPIO.setwarnings(False)
    GPIO.setup(DOOR,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
    GPIO.setup(PIR,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
    GPIO.setup(VIBRATION,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
    GPIO.setup(BUZZER,GPIO.OUT)
```

```

def alarm():

    """-----"""

    """ function that will activate the buzzer
    when intrusion is detected"""

    """-----"""

    ALARM.start(60)

while 1:

    if ShutDown.value==1:

        ALARM.stop(BUZZER)

        intruder.value=0

        break


def main_loop(intruder,ShutDown):

    """-----"""

    """ Main loop monitored by all multiprocessing variables
    and the responsable for the call of the alarm"""

    """-----"""

while True:

    while ShutDown.value==0:

        intrution=GPIO.input(DOOR)

        movement=GPIO.input(PIR)

        window=GPIO.input(VIBRATION)

```

```

if intruder.value==1 or movement==1 or window==1:

    intruder.value=1

    alarm()

    break


def client_loop(intruder, ShutDown):

    """-----
    """This loop is in charge of informing user and
    database of different events changes on the system"""
    """-----"""

while 1:

    try:

        d = s.recvfrom(1024)

        order = d[0]

        addr = d[1]

        if order=='status':

            if intruder.value==1 and ShutDown.value==0:

                arm="armed"

                detect="detected"

                msg5="insert,alarm,{},{}".format(arm,detect)

                s.sendto(msg5,addr)

            if intruder.value==0 and ShutDown.value==1:

                arm="disarmed"

```

```

detect="notdetected"

msg6="{},".format(arm,detect)

s.sendto(msg6,addr)

if ShutDown.value==0 and intruder.value==0:

    arm="armed"

detect="notdetected"

msg7="insert,alarm,{},{}".format(arm,detect)

s.sendto(msg7,addr)

if intruder.value==1 and ShutDown.value==1:

    arm="disarmed"

detect='detected'

msg8="{},".format(arm,detect)

s.sendto(msg8,addr)

print "entre a status"

if order=='arm':

    ShutDown.value=0

    msg1="SystemArmed"

    s.sendto(msg1,addr)

if order=='disarm':

    ShutDown.value=1

    msg="SystemDisarmed"

```

```
s.sendto(msg,addr)

if intruder.value==1:

    msg="Intruder Detected"

    s.sendto(msg,addr)

    order = None


except socket.error:

    sys.exit()

try:

    setup()

    ALARM=GPIO.PWM(BUZZER,60)

    p1=mp.Process(target=main_loop, args=(intruder,ShutDown))

    p2=mp.Process(target=client_loop, args=(intruder,ShutDown))

    p1.start()

    p2.start()

except KeyboardInterrupt():

    ALARM.stop()

    GPIO.cleanup()

    del ALARM

    del s

    del p1
```

```
del p2
```

```
sys.exit(0)
```

First we declare all the variables that will identify each port of the different sensors and actuators of the alarm. Then the socket is created with its respective port and IP which will be used to identify each system of the house. After this is completed instances of the multiprocessing library are created in order to keep all parallel process without conflict.

Many different functions compose this code as was seen on figure## shown earlier it has a setup function which will initialize the GPIO pins on the Raspberry Pi. The alarm function is where the buzzer activates when on the main loop function an event happens. The client loop also depends on events happening either on the alarm loop or on the main loop functions. This last mentioned function will log status updates to the database, send information to the user via the mobile application system, and receive orders to disarm or arm the alarm. The values of “Shutdown” and “intruder” variables change depending on what happens on each process.

For example, if the user sends the order to disarm the system “Shutdown” goes to 1 meaning that the alarm is disarmed and the main loop must not check anymore for activity. If the alarm was to be armed and an event occurred on the main loop, the “intruder” variable then changes value and client loop informs the user and changes the log to the database. On the moment that the alarm is tripped, a signal sent from the client loop tells the mobile application to change the panel view of the interface to the camera feed. This is of course done for security purposes.

10.4.3 Testing the socket

For testing purposes a dummy program was made to test how the alarm system reacts to different commands via the socket loop. The dummy program is shown on the next figure. This program will only send a packet with information that is provided by the user of the test program. This will be used widely across all systems and will provide most of the tests made through the internet.

```
udp socket test program

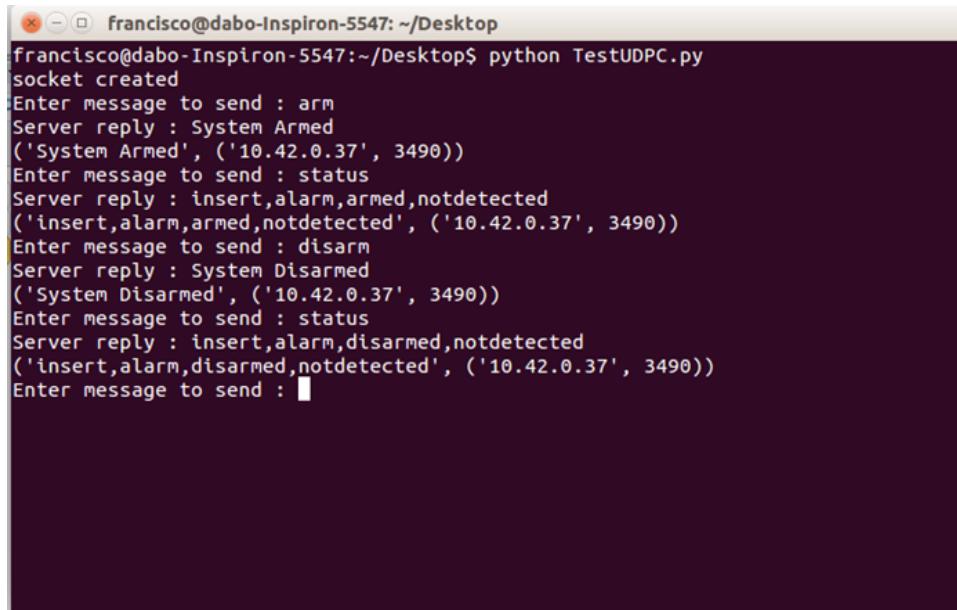
import socket #for sockets
import sys #for exit
# create dgram udp socket
try:
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
except socket.error:
    print 'Failed to create socket'
    sys.exit()
print "socket created"

host = '10.42.0.37'
port = 3490
while(1):
    msg = raw_input('Enter message to send : ')
```

```
try :  
    #Set the whole string  
    s.sendto(msg, (host, port))  
  
    # receive data from client (data, addr)  
    d = s.recvfrom(4096)  
  
    reply = d[0]  
  
    addr = d[1]  
  
    print 'Server reply : ' + reply  
  
    print d  
  
except socket.error, msg:  
    print 'Error Code : ' + str(msg[0]) + ' Message ' + msg[1]  
  
    sys.exit()
```

Steps:

1. The command arm was sent to the Raspberry pi running the alarm code the system replied the expected “System Armed”.
2. The command status was sent to monitor the status of the system. The csv(comma separated value) sent by the alarm system will help the data base store the log for the system.
3. The command disarm was sent to the Raspberry pi running the alarm code the system replied the expected “System Disarmed”.
4. The command status was sent to monitor the status of the system. As expected, the alarm returned disarmed for status of the “Shutdown” variable and “not detected” for the status of intrusion since it is disarmed.



```
francisco@dabo-Inspiron-5547: ~/Desktop
francisco@dabo-Inspiron-5547:~/Desktop$ python TestUDPC.py
socket created
Enter message to send : arm
Server reply : System Armed
('System Armed', ('10.42.0.37', 3490))
Enter message to send : status
Server reply : insert,alarm,armed/notdetected
('insert,alarm,armed/notdetected', ('10.42.0.37', 3490))
Enter message to send : disarm
Server reply : System Disarmed
('System Disarmed', ('10.42.0.37', 3490))
Enter message to send : status
Server reply : insert,alarm,disarmed/notdetected
('insert,alarm,disarmed/notdetected', ('10.42.0.37', 3490))
Enter message to send : █
```

Figure 10.5.3.1 Socket Program Results

Chapter 11: Verification System

11.1 Alternatives Considered

11.1.1 RFID (Radio Frequency Identification)

An RFID exchange involves two actors: a target and an initiator. The initiator, a tag reader or reader/writer device, starts the exchange by generating a radio field and listening for responses from any target in the field. The target, a tag, responds when it picks up a transmission from an initiator. It will respond with an unique identifier number (UID). RFID has two communication modes: active and passive.

Passive RFID exchange involves a reader/writer and a tag that has not power source on board. The tags get their power from the energy of the radio field itself. It is generally a small amount, just enough to send signal back to the reader.

Active RFID exchange involves a target that is an independently powered device. Because the target is powered, its reply to the reader can travel a much greater distance.

RFID tags work at the frequency spectrums. At the lowest common frequency or LF (Low Frequency) , the range is from 58 kHz to 148.5 kHz. The next frequency range spans from 1.75 MHz to 13.56 MHz. This frequency range is called the HF (High Frequency). The next frequency range is UHF (Ultra High Frequency), which spans the 433 MHz, 840-950 MHz, and in the Microwave range that spans from 2.4 GHz to 5.8 GHz, and from 3.1 GHz to 10 GHz ranges.

RFID tags have a small amount of memory on board, usually less than 1 kilobyte. An initiator device can read this data, and if it is a reader/writer device, it can write to the tag as well. This allows you to store small amounts of information associated with the card. Since RFID systems generally are networked to a database, it is more common to store a data record indexed by the tag's UID in a remote database, and store all the information about the tag in that remote database.

There is no single universally interoperable RFID protocol or technology. There are dozens. RFID standards are developed by the International Standard Organization (ISO), in conjunction with major participants in the RFID market. The various RFID standards define the radio frequencies used, the data transfer rates, the data formats, and more. Some of these standards define layers of single interoperable stack, as it is in the case of NFC.

11.1.2 NFC (Near Field Communication)

NFC refers to the set of protocols that enable electronic devices to establish radio communication with each other by bringing these into proximity to a distance of typically 10 cm or less.

Data exchanged between NFC devices and tags is formatted using the NFC Data Exchange Format (NDEF). NDEF is one of the enhancements that NFC adds to the RFID. It is a common data format that operates across all NFC devices, regardless of the underlying tag or device technology.

NDEF is one of the important technical differences between RFID and NFC. Both NFC and many of the RFID protocols operate on 13.56MHz, but RFID tags do not have to format their data in NDEF format. The various RFID protocols do not share a common data format. NDEF messages can be seen as paragraphs and records like sentences.

When considering the implementation of NFC technology in a project, one of the challenges is choosing the type of tag that will be required. We considered four (4) common types of tags: Type 1, Type 2, Type 4, and Mifare Classic. Mainly, the tag features increase as their type increase. Type 1 tags being the basic models, and Type 4 tags have enhanced features that can work for more complicated projects.



Figure 11.1.1 – NFC Mifare Classic 1K Tags

NFC tags typically contain data and are read only, but the data can be changed. Figure 10.1.1 presents some examples for the NFC tags, specifically the Mifare Classic 1K Tag. Some applications of the NFC tags include secure data storage, personal identification numbers (PIN), credit card information, etc.

Type 1:

- Based on ISO-14443 A specification.
- Can be read-only, or read/write capable.
- 96 bytes to 2 kilobytes of memory.
- Communication speed is 106Kb.
- No data collision protection.

Type 2:

- Based on ISO-14443 A specification.
- Can be read-only, or read/write capable.
- 96 bytes to 2 kilobytes of memory.
- Communication speed is 106Kb.
- Anti-collision support.

As we can see, Type 1 and Type 2 do not have too much difference. Until just recently, Type 1 tags are available with higher memory than Type 2 and at the same cost. The reason is that Type 1 tags are so basic that these still have room for additional memory.

Anyways, Type 2 tags have more sophisticated features that make them easier to work with. While Type 2 tags can read 16 bytes at a time and write in four byte blocks, Type 1 tags read and write only one byte at a time. The main reason why Type 2 tags are a better choice would be the speed and compatibility.

Type 4:

- Based on ISO-14443 A specification.
- Configured by factory to be read-only, or read/write capable.
- 2, 4, or 8 kilobytes of memory.
- Variable memory, up to 32 kilobytes per exchange.
- Three communication speeds: 106, 2212, 424 Kbs.
- Anti-collision support.

The main improvement in the Type 4 tag is security. Security does not mean the ability to lock the tag, or prevent someone from overwriting the information. All tags can do that, however only more sophisticated Type 4 tags have true authentication with an abundance of user memory to store application data.

Although some Type 2 tags require password protection as a required feature, a 32-bit password is handy but is really meant for low-value products, such as drink or meal tickets. A password is ultimately transmitted to the tag in unencrypted data that anyone can understand. The authentication makes it possible for a reader to become trusted by the tag and modify the data without any sensitive information ever being transmitted.

Mifare Classic tag:

- Based on ISO-14443 A specification.
- 192,768 or 3,584 bytes of memory.

- Communication speed is 196Kbps.
- Anti-collision support.

Mifare Classic tags are special. It is a chip that is entirely proprietary of NXP and not an NFC Forum defined standard. These can only communicate with a Mifare Classic tag.

These tags can be found at very affordable costs. The reason for the low cost for tags that have so much memory and good features is the lack of security. The encryption is developed by NXP specifically for this product, and it is not reliable. Nonetheless, these tags are still made and sold since for certain application the security is good enough for some applications and there is no requirement for the tag to ever be read by a mobile phone.

The reason that the Mifare Classic chips cannot be read by most phones is that Broadcom was not granted a license to the proprietary security method and as such many new models that contain Broadcom NFC chips cannot read these tags.

The following table presents a summarized comparison between the four tag types:

	Mifare Classic	Type 1	Type 2	Type 4
Pros	<ul style="list-style-type: none"> -Low cost. -Secure enough for some applications. -High memory. -Flexible. 	<ul style="list-style-type: none"> -Low cost. -High memory. 	<ul style="list-style-type: none"> -Widely supported. -Faster than Type 1. 	<ul style="list-style-type: none"> -Highly sophisticated. -High security. -Large memory. -Fast performance.
Cons	<ul style="list-style-type: none"> -Only read by NXP readers. 	<ul style="list-style-type: none"> -Basic capability. -Slow read/write. 	<ul style="list-style-type: none"> -Password, but not true authentication. -Low memory. 	<ul style="list-style-type: none"> -Expensive. -Not as easy to work with. -Need to know ISO 7816.
Typical Applications	<ul style="list-style-type: none"> -Day tickets at events. -Low value ticketing. 		<ul style="list-style-type: none"> -Low value transactions. -Day transit pass. -Events ticket. 	<ul style="list-style-type: none"> -Banking. -Transit. -Government IDs.

			-URL redirects.	
--	--	--	-----------------	--

11.1.3 NFC vs RFID

RFID is a method of identification by radio waves. NFC devices operate at the same frequency (13.56 MHz) as the HF RFID readers and tags. The standards of the NFC format is based on RFID standards outlined in ISO/IEC 14443, FeliCa, and the basis for parts of ISO/IEC 18092.



Figure 10.1.2 – NFC vs RFID

The first major difference in architecture between the RFID and NFC is the peer-to-peer communications mode, which is implemented using the ISO-18092 standard. Peer-to-peer communication is a feature that sets NFC apart from typical RFID devices. An NFC device is able to act both as a reader and as a tag. This unique ability has made NFC a popular choice. There are two protocols,

the logical link protocol (LLCP) and simple NDEF exchange protocol (SNEP) that manage peer-to-peer exchanges.

The second major difference is the NFC Data Exchange Format. NDEF defines the data exchange in messages, which make it possible for the application code to deal with data from NFC tag reading and writing, peer-to-peer exchanges, and card emulation all the same way.

At the end of the day, NFC build upon the standards of HF RFID and turns the limitations of its operating frequency into a unique feature of near-field-communication.

11.2 System Specifications

11.2.1 Verification System Components

The Verification system main component is the Arduino microprocessor. The Arduino microprocessor incorporates the functions of the whole system. It has the capacity to perform the operations needed for the functionality of all the verification system components.



Figure 11.2.1 – Arduino Mega 2560

Another important component is the NFC Shield, which is an interface for the Arduino, built around the NXP PN532 integrated circuit. The NFC technology is regulated by Near Field Communication Forum, which standardizes how the devices pair, share data, and allow a secure transaction to happen.

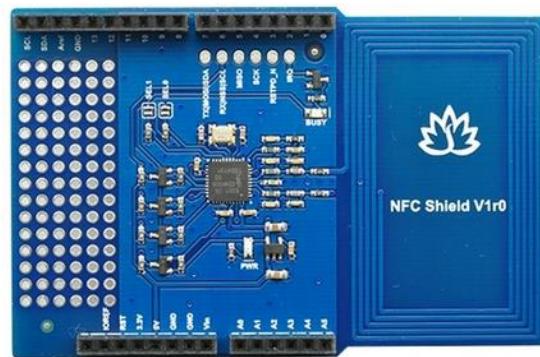


Figure 11.2.2 – NFC Shield V1r0

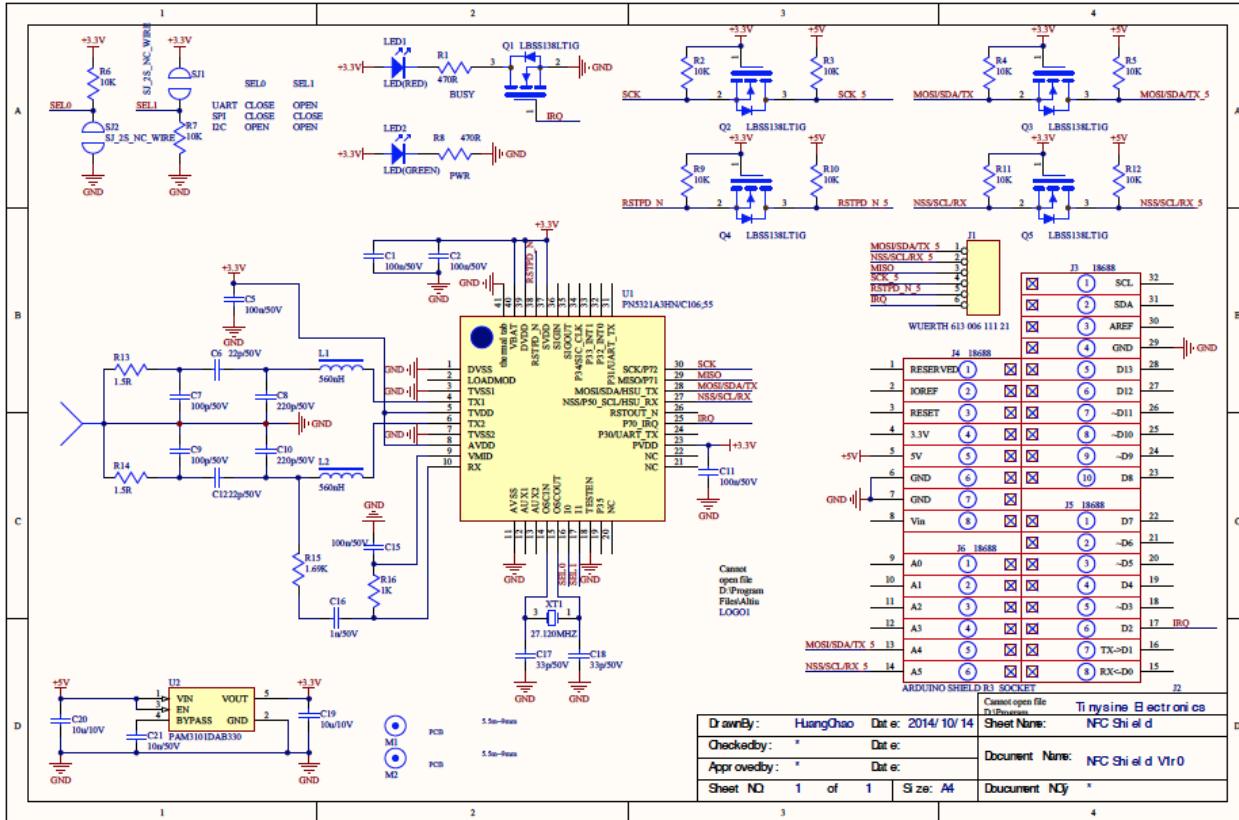


Figure 11.2.3 – NFC Shield V1r0 Schematic

Some important characteristics of the NFC Shield:

- Operate on unlicensed ISM (Industrial Scientific Medical) band of 13.56 MHz
- Although NFC communication range is up to 10 cm, the NFC Shield antenna is designed to work within a range of 1 cm.
- It provides all the necessary circuitry for PN352 like the 27.12MHz crystal, the power supply, and it breaks-out the I/O pins of PN532 for easy access.

Another important component is the Mifare Classic 4K Tag. This Mifare Tag is a memory storage device, where the memory is divided into segments and blocks with simple security mechanisms for access control. The NFC Shield is able to write and read passive id data in the tag.

As explained before, the Mifare Classic 4K tag has 3.5 kB of memory. It has a communication speed of 106 Kbps, and provides anti-collision support.



Figure 11.2.4 – NFC Mifare Classic Tag

The verification system uses the tag to pass the username information. The system uses the username information to verify it against the database information, if the user exists, it compares the password entered by the user to the password stored in the database. If everything matches, the system disarms the alarm system.

The password is obtained by means of a hex keypad that is connected to the microprocessor board. The keypad is an important component of the embedded system. The selected hex keypad is composed of 12 push button switches in a 3x4 matrix form. Figure 10.2.5 presents how the schematic connections inside the keypad. The program in the Arduino identifies the pressed key by a method called column scanning. In this method a particular row is kept low and other rows are held high. When the logic

status of each column line is scanned, the key is found by recognizing the particular column and row that are found low.

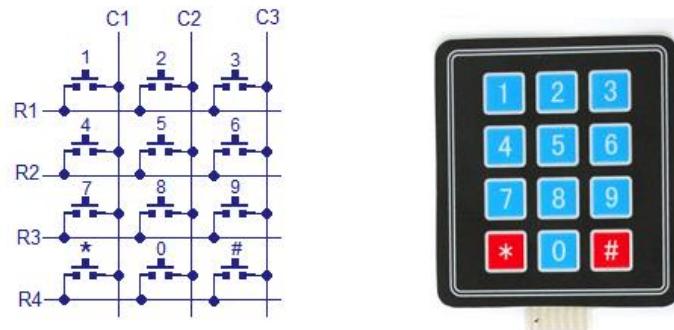


Figure 11.2.5 – Hex Keypad

In order to facilitate the creation of an interface on which the user and the system are able to communicate and understand the status of the process, an LCD is used. The LCD allows the user to know if the tag's username has been identified as well as if the password has been verified.

An LCD is an important part of the verification system because it helps the customer to understand the process. In order to make it work, a variable resistor is used to graduate the intensity of the LCD display.



Figure 11.2.6 – LCD and Variable Resistor

The communication between the system and the database is completely necessary to verify that the user exists and that the password is the one pertaining to the specific user. A process in the database server is in charge of managing the information and returning the verification to the system, but to make it work it is necessary to use an Ethernet Shield that serves as a connector for the Arduino to the system network.

As with the other systems, the verification system uses the User Datagram Protocol (UDP) to transport the information to the database server.

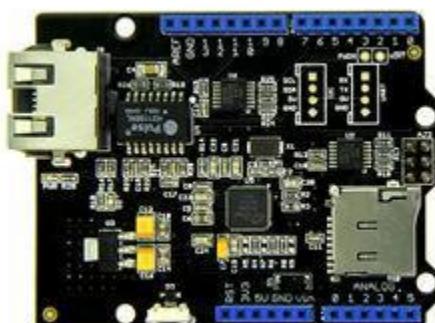


Figure 11.2.7 – Ethernet Shield

Last, but not least, is the project enclosure. The enclosure is constructed with durable plastic, which protects the whole system from water, mechanical damage, and direct heat. It also provides an esthetic part to the system.



Figure 11.2.8 – Project Enclosure (7"x5"x3')

11.3 Economic Analysis

After the system analysis was made and the sensors picked the prices for each sensor was gathered in order to analyze if the components chosen fit our budget. The components for this system involve a numeric keypad, a liquid crystal display (LCD), a potentiometer, an Arduino board, a breadboard, an NFC reader shield, and finally an Ethernet shield. The use and price of each of these devices is detailed on table 11.3.1 shown next.

Component	Use	Price
4 by 4 numeric Keypad	These device will permit the user to input credentials to the system in order to verify the user.	\$3.69 + shipping and handling
Arduino MEGA microcontroller	The board with the main code for this system and is in charge of all the necessary computations the system needs.	\$37.03 + shipping and handling
Ethernet shield	This shield is stacked up on top of the MEGA board and will send data to the data base and the mobile app through Ethernet.	\$33.17 + shipping and handling

NFC shield v1r0	This shield is also stacked up on the board and will read the user's tag which has a programmed id.	\$26.95 + shipping and handling
Potentiometer	This potentiometer will adjust the brightness of the LCD screen.	10 Pcs. For \$4.99
LCD	This will serve as an interface to give instructions to the user about the system.	\$4.39 + shipping and handling
Breadboard	Ease the connections made to and from the MEGA board.	5 Pcs. For \$3.49

11.4 Results

11.4.1 Verification System Implementation

Figure 11.4.1 describe the verification system design in a very simple way. It serves to arm or disarm the alarm system by means of user identification. The verification system process can be explained in the following simple steps:

- The user passes the NFC Mifare Classic tag by the NFC reader box.
- The NFC reader collects the username and passes it through a socket to the database server.
 - The server queries all the usernames in the database and verifies if the username from the tag matches one of the usernames in the database.
 - The username verification result is sent to the Arduino microprocessor, which displays the result in the LCD.
 - If the username is not verified, the verification process ends and it waits for a new tag to start the process.
 - If the username is verified, the LCD asks for the password and it waits for the user to enter the four (4) digit password to be entered in the keypad.
 - After the user enters the password, the Arduino microcontroller sends the password through a socket to the database server.
 - The server queries the password for the specific username and verifies if the entered password matches the password from the database.

- If the password is verified, the system changes the alarm status, otherwise the alarm status stays as it was.
- The password verification result is sent back to the Arduino microprocessor, which displays the result in the LCD.
- After displaying the verification results in the LCD, the system waits for the next NFC tag to start the process again.

Another way to look at the verification process is presented in Figure 11.4.2.

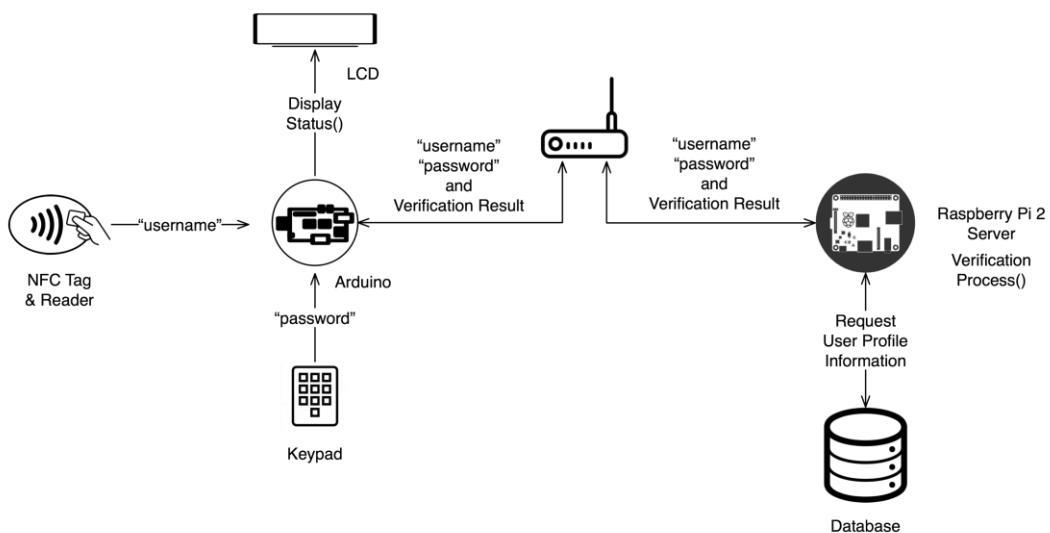


Figure 11.4.1 – Verification System Diagram

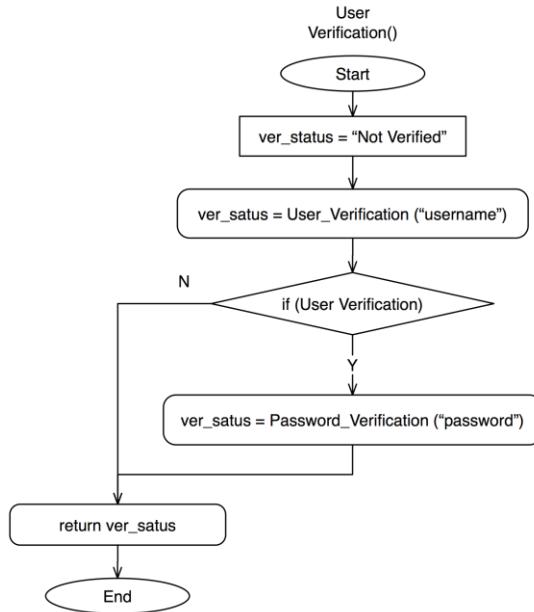


Figure 11.4.2 – Verification Process Flowchart

11.4.2 NFC Verification System Connections

As explained before, all the components were combined together to make the system work. The Arduino works as the controller of the verification system, where all the data input and output is managed, so all of the components are connected to it. Most of the components only use the Digital I/O pins to work, but the NFC Shield is the only component that requires using two Analog I/O pins. The pins used for each of the components are presented in Figure 11.4.3. Those marked in brown correspond to the NFC Shield, blue ones correspond to the Ethernet Shield, purple ones correspond to the 3x4 keypad, orange ones correspond to the LCD, the red ones correspond to the 5V connections, and black ones correspond to the ground.

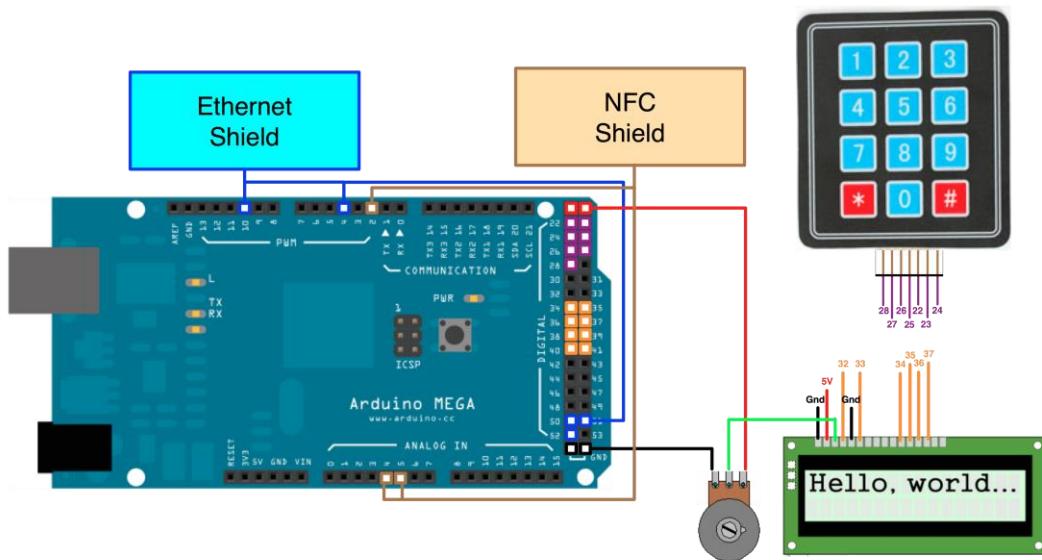


Figure 11.4.3 – NFC Verification System Connections

11.4.3 NFC System Write to Mifare Classic Memory

As explained before, the verification system uses an NFC Mifare Classic tag to identify the user that is trying to activate or deactivate the alarm system. For this purpose, that user information must be written into the Mifare Classic tag. In order to achieve this, RMF Intelligent Home uses the Ethernet Shield to write into the tag. The code to write in the tag is presented in the following figure.

```
//Including the necessary Libraries, the NFC code works for Arduino and with Intel Galileo
#include "Arduino.h"
#include <Wire.h>
#if defined(__AVR__) || defined(__i386__) //compatibility with Intel Galileo
#define WIRE Wire
#else // Arduino Due
#define WIRE Wire1
#endif

// Defining the ports using Hexadecimal
#define IRQ (2) // Interrupt
#define RESET (3) // Not connected by default on the NFC Shield
#define PN532_PREAMBLE (0x00)
#define PN532_STARTCODE2 (0xFF)
#define PN532_POSTAMBLE (0x00)
#define PN532_HOSTTOPN532 (0xD4)
#define PN532_PN532TOHOST (0xD5)

// PN532 Commands
#define PN532_COMMAND_GETFIRMWAREVERSION (0x02)
#define PN532_COMMAND_READGPIO (0x0C)
```

```

#define PN532_COMMAND_WRITEGPIO      (0x0E)

#define PN532_COMMAND_SAMCONFIGURATION (0x14)

#define PN532_COMMAND_RFCONFIGURATION (0x32)

#define PN532_COMMAND_INLISTPASSIVETARGET (0x4A)

#define PN532_COMMAND_INDATAEXCHANGE    (0x40)

#define PN532_RESPONSE_INDATAEXCHANGE   (0x41)

#define PN532_I2C_ADDRESS            (0x48 >> 1)

#define PN532_I2C_BUSY               (0x00)

#define PN532_I2C_READY              (0x01)

#define PN532_MIFARE_ISO14443A        (0x00)

// Mifare Commands

#define MIFARE_CMD_AUTH_A           (0x60)

#define MIFARE_CMD_AUTH_B           (0x61)

#define MIFARE_CMD_READ              (0x30)

#define MIFARE_CMD_WRITE             (0xA0)

// Prefixes for NDEF Records (to identify record type)

// Defining the GPIO (General Purpose Input/Output)

#define PN532_GPIO_VALIDATIONBIT     (0x80)

#define PN532_GPIO_P30                (0)

#define PN532_GPIO_P31                (1)

#define PN532_GPIO_P32                (2)

#define PN532_GPIO_P33                (3)

#define PN532_GPIO_P34                (4)

#define PN532_GPIO_P35                (5)

#define PN532_PACKBUFSIZ 64

byte pn532_packetbuffer[PN532_PACKBUFSIZ];

//Global variables for the NFC

```

```

uint8_t _irq, _reset;

uint8_t _uid[7]; // ISO14443A uid

uint8_t _uidLen; // uid len

uint8_t _key[6]; // Mifare Classic key

uint8_t inListedTag; // Tg number of inlisted tag.

boolean readackframe(void);

uint8_t wirereadstatus(void);

void wirereaddata(uint8_t* buff, uint8_t n);

void wiresendcommand(uint8_t* cmd, uint8_t cmdlen);

boolean waitUntilReady(uint16_t timeout);

byte pn532ack[] = {0x00, 0x00, 0xFF, 0x00, 0xFF, 0x00};

byte pn532response_firmwarevers[] = {0x00, 0xFF, 0x06, 0xFA, 0xD5, 0x03};

/********************* Sends a single byte via I2C ********************/

static inline void wiresend(uint8_t x)

{

#if ARDUINO >= 100

    WIRE.write((uint8_t)x);

#else

    WIRE.send(x);

#endif

}

/********************* Reads a single byte via I2C ********************/

static inline uint8_t wirerecv(void)

{

#if ARDUINO >= 100

    return WIRE.read();

#else

    return WIRE.receive();

}

```

```

#endif

}

/********************************************/

/*
brief -Prints a hexadecimal value in plain characters

param -data    Pointer to the byte data

param -numBytes Data length in bytes

*/
void PrintHex(const byte * data, const uint32_t numBytes)
{
    uint32_t szPos;

    for (szPos=0; szPos < numBytes; szPos++)
    {
        Serial.print(F("0x"));

        // Append leading 0 for small values
        if (data[szPos] <= 0xF)
            Serial.print(F("0"));

        Serial.print(data[szPos]&0xff, HEX);

        if ((numBytes > 1) && (szPos != numBytes - 1))
        {
            Serial.print(F(" "));
        }
    }

    Serial.println();
}

/********************************************/


/*
@brief Prints a hexadecimal value in plain characters, along with
the char equivalents in the following format

```

```

00 00 00 00 00 00 .....  

@param data    Pointer to the byte data  

@param numBytes Data length in bytes  

*/  

/******  

void PrintHexChar(const byte * data, const uint32_t numBytes)  

{  

    uint32_t szPos;  

    for (szPos=0; szPos < numBytes; szPos++)  

    {  

        // Append leading 0 for small values  

        if (data[szPos] <= 0xF)  

            Serial.print(F("0"));  

        Serial.print(data[szPos], HEX);  

        if ((numBytes > 1) && (szPos != numBytes - 1))  

        {  

            Serial.print(F(" "));  

        }  

    }  

    Serial.print(F(" "));  

    for (szPos=0; szPos < numBytes; szPos++)  

    {  

        if (data[szPos] <= 0x1F)  

            Serial.print(F("."));  

        else  

            Serial.print((char)data[szPos]);  

    }  

    Serial.println();  

}  

/******
```

```

//Checks the the PN5xx chip firmware version
//returns the chip's firmware version and ID

/***************************************************************/

uint32_t FirmwareVersion(void) {

    uint32_t response;

    pn532_packetbuffer[0] = PN532_COMMAND_GETFIRMWAREVERSION;

    if (!SendCommandCheckAck(pn532_packetbuffer, 1,1000))
        return 0;

    // read data packet
    wirereaddata(pn532_packetbuffer, 12);

    // check some basic stuff
    if (0 != strncmp((char *)pn532_packetbuffer, (char *)pn532response_firmwarevers, 6)) {
        #ifdef PN532DEBUG
        Serial.println(F("Firmware doesn't match!"));
        #endif
        return 0;
    }

    response = pn532_packetbuffer[7];
    response <= 8;
    response |= pn532_packetbuffer[8];
    response <= 8;
    response |= pn532_packetbuffer[9];
    response <= 8;
    response |= pn532_packetbuffer[10];
}

```

```

return response;
}

/*****************************************/
/*
brief Sends a command and waits a specified period for the ACKNOWLEDGE
param cmd    Pointer to the command buffer
param cmdlen The size of the command in bytes
param timeout timeout before giving up

returns 1 if everything is OK, 0 if timeout occurred before an
ACK was received
*/
/*****************************************/
// default timeout of one second

boolean SendCommandCheckAck(uint8_t *cmd, uint8_t cmdlen , uint16_t timeout) {
    uint16_t timer = 0;

    // write the command
    wiresendcommand(cmd, cmdlen);

    // Wait for chip to say its ready
    while (wirereadstatus() != PN532_I2C_READY) {
        if (timeout != 0) {
            timer+=10;
            if (timer > timeout)
                return false;
        }
        delay(10);
    }
}

```

```

#ifndef PN532DEBUG

Serial.println(F("IRQ received"));

#endif


// read Acknowledgement

if (!readackframe()) {

#ifndef PN532DEBUG

Serial.println(F("No ACK frame received!"));

#endif

return false;

}

return true; // acknowledged command
}

/********************************************/


/*
Reads the state of the PN532's GPIO pins

@returns An 8-bit value containing the pin state where:

pinState[0] = P30
pinState[1] = P31
pinState[2] = P32
pinState[3] = P33
pinState[4] = P34
pinState[5] = P35

*/
/********************************************/


uint8_t readGPIO(void) {

pn532_packetbuffer[0] = PN532_COMMAND_READGPIO;

// Send the READGPIO command (0x0C)

```

```

if (!SendCommandCheckAck(pn532_packetbuffer, 1,1000))
    return 0x0;

// Read response packet (00 00 FF PLEN PLENChecksum D5 CMD+1(0xD) P3 P7 IO1 DATACHECKSUM)
wirereaddata(pn532_packetbuffer, 11);

/* READGPIO response should be in the following format:
byte      Description
-----
b0..6      Frame header and preamble
b7          P3 GPIO Pins
b8          P7 GPIO Pins (not used ... taken by I2C)
b9          Interface Mode Pins (not used ... bus select pins)
b10         checksum */

```

```

#ifndef PN532DEBUG
Serial.print(F("Received: "));
PrintHex(pn532_packetbuffer, 11);
Serial.println();
Serial.print(F("P3 GPIO: 0x")); Serial.println(pn532_packetbuffer[7], HEX);
Serial.print(F("P7 GPIO: 0x")); Serial.println(pn532_packetbuffer[8], HEX);
Serial.print(F("IO GPIO: 0x")); Serial.println(pn532_packetbuffer[9], HEX);
// Note: You can use the IO GPIO value to detect the serial bus being used
switch(pn532_packetbuffer[9])
{
    case 0x00: // Using UART
        Serial.println(F("Using UART (IO = 0x00)"));
        break;
    case 0x01: // Using I2C
        Serial.println(F("Using I2C (IO = 0x01)"));
        break;
}

```

```

        case 0x02: // Using I2C
            Serial.println(F("Using I2C (IO = 0x02)"));

            break;
        }

#endif

return pn532_packetbuffer[6];
}

//*******************************************************************************/

/*
brief Configures the SAM (Secure Access Module)
*/
boolean SAMConfig(void) {

pn532_packetbuffer[0] = PN532_COMMAND_SAMCONFIGURATION;

pn532_packetbuffer[1] = 0x01; // normal mode;

pn532_packetbuffer[2] = 0x14; // timeout 50ms * 20 = 1 second

pn532_packetbuffer[3] = 0x01; // use IRQ pin!

if (!SendCommandCheckAck(pn532_packetbuffer, 4,1000))

    return false;

// read data packet

wirereaddata(pn532_packetbuffer, 8);

return (pn532_packetbuffer[6] == 0x15);

}

//************************************************************************** ISO14443A Commands *****/
/*
Waits for an ISO14443A target to enter the field

```

```

param cardBaudRate Baud rate of the card

param uid      Pointer to the array that will be populated with the card's UID (up to 7 bytes)

param uidLength  Pointer to the variable that will hold the length of the card's UID.

returns 1 if everything executed properly, 0 for an error

*/
boolean readPassiveTargetID(uint8_t cardbaudrate, uint8_t * uid, uint8_t * uidLength) {

pn532_packetbuffer[0] = PN532_COMMAND_INLISTPASSIVETARGET;

pn532_packetbuffer[1] = 1; // max 1 cards at once (we can set this to 2 later)

pn532_packetbuffer[2] = cardbaudrate;

if (!SendCommandCheckAck(pn532_packetbuffer, 3,1000))

{

#ifndef PN532DEBUG

Serial.println(F("No card(s) read"));

#endif

return 0x0; // no cards read

}

// Wait for a card to enter the field

uint8_t status = PN532_I2C_BUSY;

#ifndef PN532DEBUG

Serial.println(F("Waiting for IRQ (indicates card presence)"));

#endif

while (wirereadstatus() != PN532_I2C_READY)

{

    delay(10);

}

#ifndef PN532DEBUG

Serial.println(F("Found a card"));


```

```

#endif

// read data packet

wirereaddata(pn532_packetbuffer, 20);

// check some basic stuff

/* ISO14443A card response should be in the following format:

byte      Description
-----
b0..6     Frame header and preamble
b7        Tags Found
b8        Tag Number (only one used in this example)
b9..10    SENS_RES
b11       SEL_RES
b12       NFCID Length
b13..NFCIDLen NFCID          */

#endif MIFAREDEBUG

Serial.print(F("Found ")); Serial.print(pn532_packetbuffer[7], DEC); Serial.println(F(" tags"));

#endif

if (pn532_packetbuffer[7] != 1)

return 0;

uint16_t sens_res = pn532_packetbuffer[9];

sens_res <<= 8;

sens_res |= pn532_packetbuffer[10];

#endif MIFAREDEBUG

Serial.print(F("ATQA: 0x")); Serial.println(sens_res, HEX);

Serial.print(F("SAK: 0x")); Serial.println(pn532_packetbuffer[11], HEX);

#endif

```

```

/* Card appears to be Mifare Classic */

*uidLength = pn532_packetbuffer[12];

#ifndef MIFAREDEBUG
    Serial.print(F("UID:"));
#endif

for (uint8_t i=0; i < pn532_packetbuffer[12]; i++)
{
    uid[i] = pn532_packetbuffer[13+i];

#ifndef MIFAREDEBUG
    Serial.print(F(" 0x"));Serial.print(uid[i], HEX);
#endif

}

#ifndef MIFAREDEBUG
    Serial.println();
#endif

return 1;
}

/***************************************** Mifare Classic Functions *****/
/*
Tries to authenticate a block of memory on a MIFARE card using the
INDATAEXCHANGE command. See section 7.3.8 of the PN532 User Manual
for more information on sending MIFARE and other commands.

param uid      Pointer to a byte array containing the card UID
param uidLen   The length (in bytes) of the card's UID (Should be 4 for MIFARE Classic)
param blockNumber  The block number to authenticate. (0..63 for 1KB cards, and 0..255 for 4KB cards).
param keyNumber Which key type to use during authentication (0 = MIFARE_CMD_AUTH_A, 1 =
MIFARE_CMD_AUTH_B)

```

```

param keyData    Pointer to a byte array containing the 6 byte key value

returns 1 if everything executed properly, 0 for an error

*/
*****mifareclassic_AuthenticateBlock (uint8_t * uid, uint8_t uidLen, uint32_t blockNumber, uint8_t keyNumber, uint8_t * keyData)

{
    uint8_t len;
    uint8_t i;

    // Hang on to the key and uid data
    memcpy (_key, keyData, 6);
    memcpy (_uid, uid, uidLen);
    _uidLen = uidLen;

#ifdef MIFAREDEBUG
    Serial.print(F("Trying to authenticate card "));
    PrintHex(_uid, _uidLen);
    Serial.print(F("Using authentication KEY "));Serial.print(keyNumber ? 'B' : 'A');Serial.print(F(": "));
    PrintHex(_key, 6);
#endif

    // Prepare the authentication command //
    pn532_packetbuffer[0] = PN532_COMMAND_INDATAEXCHANGE; /* Data Exchange Header */
    pn532_packetbuffer[1] = 1; /* Max card numbers */
    pn532_packetbuffer[2] = (keyNumber) ? MIFARE_CMD_AUTH_B : MIFARE_CMD_AUTH_A;
    pn532_packetbuffer[3] = blockNumber; /* Block Number (1K = 0..63, 4K = 0..255 */
    memcpy (pn532_packetbuffer+4, _key, 6);
    for (i = 0; i < _uidLen; i++)
    {
        pn532_packetbuffer[10+i] = _uid[i]; /* 4 byte card ID */
    }
}

```

```

}

if (!SendCommandCheckAck(pn532_packetbuffer, 10+_uidLen,1000))

return 0;

// Read the response packet

wirereaddata(pn532_packetbuffer, 12);

// Check if the response is valid and we are authenticated???
// for an auth success it should be bytes 5-7: 0xD5 0x41 0x00

// Mifare auth error is technically byte 7: 0x14 but anything other than 0x00 is not good

if (pn532_packetbuffer[7] != 0x00)

{

#define PN532DEBUG

Serial.print(F("Authentication failed: "));

PrintHexChar(pn532_packetbuffer, 12);

#endif

return 0;

}

return 1;
}

***** */

/*
Tries to read an entire 16-byte data block at the specified block address.

param blockNumber The block number to authenticate. (0..63 for 1KB cards, and 0..255 for 4KB cards).

param data Pointer to the byte array that will hold the retrieved data (if any)

returns 1 if everything executed properly, 0 for an error
*/

```

```

uint8_t mifareclassic_ReadDataBlock (uint8_t blockNumber, uint8_t * data)

{

#ifndef MIFAREDEBUG

Serial.print(F("Trying to read 16 bytes from block "));Serial.println(blockNumber);

#endif


/* Prepare the command */

pn532_packetbuffer[0] = PN532_COMMAND_INDATAEXCHANGE;

pn532_packetbuffer[1] = 1;           /* Card number */

pn532_packetbuffer[2] = MIFARE_CMD_READ;    /* Mifare Read command = 0x30 */

pn532_packetbuffer[3] = blockNumber;      /* Block Number (0..63 for 1K, 0..255 for 4K) */


/* Send the command */

if (!SendCommandCheckAck(pn532_packetbuffer, 4,1000))

{

#ifndef MIFAREDEBUG

Serial.println(F("Failed to receive ACK for read command"));

#endif

return 0;

}

/* Read the response packet */

wirereaddata(pn532_packetbuffer, 26);


/* If byte 8 isn't 0x00 we probably have an error */

if (pn532_packetbuffer[7] != 0x00)

{



#ifndef MIFAREDEBUG

Serial.println(F("Unexpected response"));

PrintHexChar(pn532_packetbuffer, 26);


```

```

#endif

return 0;

}

/* Copy the 16 data bytes to the output buffer      */
/* Block content starts at byte 9 of a valid response */

memcpy (data, pn532_packetbuffer+8, 16);

/* Display data for debug if requested */

#ifndef MIFAREDEBUG
Serial.print(F("Block "));
Serial.println(blockNumber);
PrintHexChar(data, 16);
#endif

return 1;
}

/*
Tries to write an entire 16-byte data block at the specified block
address.

@param blockNumber The block number to authenticate. (0..63 for
1KB cards, and 0..255 for 4KB cards).

@param data The byte array that contains the data to write.

@return 1 if everything executed properly, 0 for an error
*/
uint8_t mifareclassic_WriteDataBlock (uint8_t blockNumber, uint8_t * data)
{

```

```

#ifndef MIFAREDEBUG

Serial.print(F("Trying to write 16 bytes to block "));Serial.println(blockNumber);

#endif


/* Prepare the first command */

pn532_packetbuffer[0] = PN532_COMMAND_INDATAEXCHANGE;

pn532_packetbuffer[1] = 1;          /* Card number */

pn532_packetbuffer[2] = MIFARE_CMD_WRITE;    /* Mifare Write command = 0xA0 */

pn532_packetbuffer[3] = blockNumber;      /* Block Number (0..63 for 1K, 0..255 for 4K) */

memcpy (pn532_packetbuffer+4, data, 16);    /* Data Payload */



/* Send the command */

if (!SendCommandCheckAck(pn532_packetbuffer, 20,1000))

{

#ifndef MIFAREDEBUG

Serial.println(F("Failed to receive ACK for write command"));

#endif

return 0;

}

delay(10);



/* Read the response packet */

wirereaddata(pn532_packetbuffer, 26);



return 1;

}






***** Mifare Ultralight Functions *****

*****



/*

```

Tries to read an entire 4-byte page at the specified address.

```
@param page    The page number (0..63 in most cases)
@param buffer   Pointer to the byte array that will hold the
                retrieved data (if any)

*/
//********************************************************************

uint8_t mifareultralight_ReadPage (uint8_t page, uint8_t * buffer)
{
    if (page >= 64)
    {
        #ifdef MIFAREDEBUG
        Serial.println(F("Page value out of range"));
        #endif
        return 0;
    }

    #ifdef MIFAREDEBUG
    Serial.print(F("Reading page "));Serial.println(page);
    #endif

    /* Prepare the command */
    pn532_packetbuffer[0] = PN532_COMMAND_INDATAEXCHANGE;
    pn532_packetbuffer[1] = 1;           /* Card number */
    pn532_packetbuffer[2] = MIFARE_CMD_READ; /* Mifare Read command = 0x30 */
    pn532_packetbuffer[3] = page;       /* Page Number (0..63 in most cases) */

    /* Send the command */
    if (!SendCommandCheckAck(pn532_packetbuffer, 4,1000))
    {
```

```

#ifndef MIFAREDEBUG

Serial.println(F("Failed to receive ACK for write command"));

#endif

return 0;

}

/* Read the response packet */

wirereaddata(pn532_packetbuffer, 26);

#ifndef MIFAREDEBUG

Serial.println(F("Received: "));

PrintHexChar(pn532_packetbuffer, 26);

#endif

/* If byte 8 isn't 0x00 we probably have an error */

if (pn532_packetbuffer[7] == 0x00)

{

/* Copy the 4 data bytes to the output buffer */

/* Block content starts at byte 9 of a valid response */

/* Note that the command actually reads 16 byte or 4 */

/* pages at a time ... we simply discard the last 12 */

/* bytes */

memcpy (buffer, pn532_packetbuffer+8, 4);

}

else

{

#endif MIFAREDEBUG

Serial.println(F("Unexpected response reading block: "));

PrintHexChar(pn532_packetbuffer, 26);

#endif

return 0;

```

```
/* Display data for debug if requested */

#ifndef MIFAREDEBUG

Serial.print(F("Page "));Serial.print(page);Serial.println(F(":"));

PrintHexChar(buffer, 4);

#endif

// Return OK signal

return 1;

}

***** high level I2C *****

*****



/*
@brief Tries to read the PN532 ACK frame (not to be confused with
the I2C ACK signal)

*/
*****



boolean readackframe(void) {

uint8_t ackbuff[6];

wirereaddata(ackbuff, 6);

return (0 == strncmp((char *)ackbuff, (char *)pn532ack, 6));

}

***** mid level I2C *****

/*
@brief Checks the IRQ pin to know if the PN532 is ready

```

```

@returns 0 if the PN532 is busy, 1 if it is free

*/
uint8_t wirereadstatus(void) {

    uint8_t x = digitalRead(IRQ);

    if (x == 1)

        return PN532_I2C_BUSY;

    else

        return PN532_I2C_READY;

}

/********************************************/

/*
@brief Reads n bytes of data from the PN532 via I2C

@param buff  Pointer to the buffer where data will be written
@param n      Number of bytes to be read

*/
/********************************************/

void wirereaddata(uint8_t* buff, uint8_t n) {

    uint16_t timer = 0;

    delay(2);

#ifndef PN532DEBUG
    Serial.print(F("Reading: "));
#endif

    // Start read (n+1 to take into account leading 0x01 with I2C)
    WIRE.requestFrom((uint8_t)PN532_I2C_ADDRESS, (uint8_t)(n+2));

    // Discard the leading 0x01

```

```

wirerecv();

for (uint8_t i=0; i<n; i++) {
    delay(1);
    buff[i] = wirerecv();
}

#ifndef PN532DEBUG
    Serial.print(F(" 0x"));
    Serial.print(buff[i], HEX);
#endif

//endif

}

// Discard trailing 0x00 0x00

// wirerecv();

#ifndef PN532DEBUG
    Serial.println();
#endif

}

//*****************************************************************************
/*
@brief Writes a command to the PN532, automatically inserting the
preamble and required frame details (checksum, len, etc.)

@param cmd Pointer to the command buffer
@param cmdlen Command length in bytes
*/
//*****************************************************************************

void wiresendcommand(uint8_t* cmd, uint8_t cmdlen) {

    uint8_t checksum;

    cmdlen++;

```

```

#ifndef PN532DEBUG

Serial.print(F("\nSending: "));

#endif

delay(2); // or whatever the delay is for waking up the board

// I2C START

WIRE.beginTransmission(PN532_I2C_ADDRESS);

checksum = PN532_PREAMBLE + PN532_PREAMBLE + PN532_STARTCODE2;

wiresend(PN532_PREAMBLE);

wiresend(PN532_PREAMBLE);

wiresend(PN532_STARTCODE2);

wiresend(cmdlen);

wiresend(~cmdlen + 1);

wiresend(PN532_HOSTTOPN532);

checksum += PN532_HOSTTOPN532;

/* *****Debugging for PN532 ***** */

#ifndef PN532DEBUG

Serial.print(F(" 0x")); Serial.print(PN532_PREAMBLE, HEX);

Serial.print(F(" 0x")); Serial.print(PN532_PREAMBLE, HEX);

Serial.print(F(" 0x")); Serial.print(PN532_STARTCODE2, HEX);

Serial.print(F(" 0x")); Serial.print(cmdlen, HEX);

Serial.print(F(" 0x")); Serial.print(~cmdlen + 1, HEX);

Serial.print(F(" 0x")); Serial.print(PN532_HOSTTOPN532, HEX);

#endif

for (uint8_t i=0; i<cmdlen-1; i++) {

    wiresend(cmd[i]);
}

```

```

checksum += cmd[i];

#ifndef PN532DEBUG
    Serial.print(F(" 0x")); Serial.print(cmd[i], HEX);
#endif

wiresend(~checksum);

wiresend(PN532_POSTamble);

// Stops I2C Communication Protocol
WIRE.endTransmission();

#ifndef PN532DEBUG
    Serial.print(F(" 0x")); Serial.print(~checksum, HEX);
    Serial.print(F(" 0x")); Serial.print(PN532_POSTamble, HEX);
    Serial.println();
#endif

void setup(void)
{
/* Local Variables for the Loop */
/* Setting up the Ports of the Arduino */

    uint32_t PN5xxversion;

    pinMode(IRQ, INPUT);

    pinMode(RESET, OUTPUT);

    Serial.begin(115200);

    Serial.println("NFC shield Start!");

    WIRE.begin();
}

```

```

// Reset the PN532

digitalWrite(RESET, HIGH);

digitalWrite(RESET, LOW);

delay(500);

digitalWrite(RESET, HIGH);

PN5xxversion = FirmwareVersion();

if (! PN5xxversion)

{

    Serial.print("Can't find PN53x, Please check the hardware connection and setting");

    while (1); // halt

}

// Got ok data, print it out!

Serial.print("PN5");

Serial.println((PN5xxversion>>24) & 0xFF, HEX);

Serial.print("Firmware version. ");

Serial.print((PN5xxversion>>16) & 0xFF, DEC);

Serial.print('.');

Serial.println((PN5xxversion>>8) & 0xFF, DEC);

// configure board to read RFID tags

SAMConfig();

}

/* Starts the Controller Loop*/

void loop(void)

{

    uint8_t success;

    uint8_t uid[] = { 0, 0, 0, 0, 0, 0, 0 }; // Buffer to store the returned UID

    uint8_t uidLength; // Length of the UID (4 or 7 bytes depending on ISO14443A card type)

    // Wait for an ISO14443A type cards (Mifare, etc.). When one is found

```

```

// 'uid' will be populated with the UID, and uidLength will indicate

// if the uid is 4 bytes (Mifare Classic) or 7 bytes (Mifare Ultralight)

success = readPassiveTargetID(PN532_MIFARE_ISO14443A, uid, &uidLength);

if(success)

{

    // Display some basic information about the card

    Serial.println("Found an ISO14443A card");

    Serial.print("ID Length: ");

    Serial.print(uidLength, DEC);

    Serial.println(" bytes");

    Serial.print("ID: ");

    PrintHex(uid, uidLength);

    Serial.println("");


    if(uidLength == 4)

    {

        // We probably have a Mifare Classic card ...

        Serial.println("Seems to be a Mifare Classic card (4 byte UID)");
    }

    // Now we need to try to authenticate it for read/write access

    // Try with the factory default KeyA: 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF

    Serial.println("Trying to authenticate block 5 with default KEYA value");

    uint8_t keya[6] = { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF };

    success = mifareclassic_AuthenticateBlock(uid, uidLength, 5, 0, keya);

    if (success)

    {

        Serial.println("Sector 1 (Blocks 4..7) has been authenticated");

        Serial.println("Writting Block 5!");

        uint8_t data[16];
    }
}

```

```
// the following 2 lines is to write the username to block 5  
  
memcpy(data, (const uint8_t[]){ 'c', 'a', 'p', 's', '0', '1', 'u', 's', 'e', 'r', 0, 0, 0, 0, 0, 0 }, sizeof data);  
  
success = mifareclassic_WriteDataBlock (5, data);  
  
if(success)  
{  
    Serial.println("Write Successful! Try to read block 5!");  
}  
else  
{  
    Serial.println("Write Failed!");  
}  
delay(2000);  
}  
else  
{  
    Serial.println("Ooops ... authentication failed: Try another key!");  
}  
}  
}  
}
```

Figure 11.4.4 – Write into Tag Code for Arduino

```

NFC shield Start!
PN532
Firmware version. 1.6
Found an ISO14443A card
ID Length: 4 bytes
ID: 0xA1 0x80 0x36 0x32

Seems to be a Mifare Classic card (4 byte UID)
Trying to authenticate block 5 with default KEYA value
Sector 1 (Blocks 4..7) has been authenticated
Writting Block 5!
Write Successful! Try to read block 5!

```

Figure 11.4.5 – Write into Tag Serial Console Result

11.4.4 NFC Verification System

After creating the user, the system is ready to start the verification process. The verification system code includes functions to deal with all the system components, as well as the logic to manage the different verification process variants. The following figure presents the verification process code.

```

/* Including the necessary libaries for the NFC, LCD, Keypad, and Ethernet Shield*/
#include "Arduino.h"

#include <Keypad.h>

#include <LiquidCrystal.h>

#include <SPI.h>

#include <Keypad.h>

#include <EthernetV2_0.h>

#include <EthernetUdpV2_0.h>

#include <Wire.h>

#if defined(__AVR__) || defined(__i386__) //compatibility with Intel Galileo

#define WIRE Wire

```

```

#else // Arduino Due

#define WIRE Wire1

#endif

/* Defining the ports using Hex*/

#define IRQ (2)

#define RESET (3) // Not connected by default on the NFC Shield

#define PN532_PREAMBLE (0x00)

#define PN532_STARTCODE2 (0xFF)

#define PN532_POSTAMBLE (0x00)

#define PN532_HOSTTOPN532 (0xD4)

#define PN532_PN532TOHOST (0xD5)

// PN532 Commands

#define PN532_COMMAND_GETFIRMWAREVERSION (0x02)

#define PN532_COMMAND_READGPIO (0x0C)

#define PN532_COMMAND_WRITEGPIO (0x0E)

#define PN532_COMMAND_SAMCONFIGURATION (0x14)

#define PN532_COMMAND_RFCONFIGURATION (0x32)

#define PN532_COMMAND_INLISTPASSIVETARGET (0x4A)

#define PN532_COMMAND_INDATAEXCHANGE (0x40)

#define PN532_RESPONSE_INDATAEXCHANGE (0x41)

//Defining the I2C Communication Protocol Pins in the PN532

#define PN532_I2C_ADDRESS (0x48 >> 1)

#define PN532_I2C_BUSY (0x00)

#define PN532_I2C_READY (0x01)

#define PN532_MIFARE_ISO14443A (0x00)

// Mifare Commands

#define MIFARE_CMD_AUTH_A (0x60)

```

```

#define MIFARE_CMD_AUTH_B          (0x61)
#define MIFARE_CMD_READ           (0x30)
#define MIFARE_CMD_WRITE          (0xA0)

//Global variables for the NFC

uint8_t _irq, _reset;

uint8_t _uid[7]; // ISO14443A uid

uint8_t _uidLen; // uid len

uint8_t _key[6]; // Mifare Classic key

uint8_t inListedTag; // Tg number of inlisted tag.

boolean readackframe(void);

uint8_t wirereadstatus(void);

void wirereaddata(uint8_t* buff, uint8_t n);

void wiresendcommand(uint8_t* cmd, uint8_t cmdlen);

boolean waitUntilReady(uint16_t timeout);

byte pn532ack[] = {0x00, 0x00, 0xFF, 0x00, 0xFF, 0x00};

byte pn532response_firmwarevers[] = {0x00, 0xFF, 0x06, 0xFA, 0xD5, 0x03};

#define PN532_PACKBUFSIZ 64

byte pn532_packetbuffer[PN532_PACKBUFSIZ];

/***********************/

/*
brief Sends a single byte via I2C
param x The byte to send
*/
/***********************/

static inline void wiresend(uint8_t x)
{
#if ARDUINO >= 100
    WIRE.write((uint8_t)x);
}

```

```

#else
WIRE.send(x);
#endif
}

/*****************************************/
/*
brief Reads a single byte via I2C
*/
static inline uint8_t wirerecv(void)
{
#if ARDUINO >= 100
return WIRE.read();
#else
return WIRE.receive();
#endif
}

/*****************************************/
/*
brief Prints a hexadecimal value in plain characters
param data Pointer to the byte data
param numBytes Data length in bytes
*/
void PrintHex(const byte * data, const uint32_t numBytes)
{
uint32_t szPos;
for(szPos=0; szPos < numBytes; szPos++)
{
Serial.print(F("0x"));
}

```

```

// Append leading 0 for small values

if (data[szPos] <= 0xF)
    Serial.print(F("0"));

Serial.print(data[szPos]&0xff, HEX);

if((numBytes > 1) && (szPos != numBytes - 1))

{
    Serial.print(F(" "));

}

}

Serial.println();

}

/*************************************************************************/
/*
brief Prints a hexadecimal value in plain characters, along with the
char equivalents in the following format 00 00 00 00 00 00 ..... 

param data   Pointer to the byte data
param numBytes Data length in bytes
*/
/*************************************************************************/

void PrintHexChar(const byte * data, const uint32_t numBytes)
{
    uint32_t szPos;

    for(szPos=0; szPos < numBytes; szPos++)
    {
        //Append leading 0 for small values
        if(data[szPos] <= 0xF)
            Serial.print(F("0"));

        Serial.print(data[szPos], HEX);

        if((numBytes > 1) && (szPos != numBytes - 1))

```

```

{
    Serial.print(F(" "));
}

}

Serial.print(F(" "));

for(szPos=0; szPos < numBytes; szPos++)

{
    if (data[szPos] <= 0x1F)
        Serial.print(F("."));

    else
        Serial.print((char)data[szPos]);
}

Serial.println();

}

/********************************************/

//Checks the the PN5xx chip firmware version

//returns the chip's firmware version and ID

uint32_t FirmwareVersion(void) {

    uint32_t response;

    pn532_packetbuffer[0] = PN532_COMMAND_GETFIRMWAREVERSION;

    if (!SendCommandCheckAck(pn532_packetbuffer, 1,1000))

        return 0;

    // read data packet

    wirereaddata(pn532_packetbuffer, 12);

    // check some basic stuff
}

```

```

if (0 != strncmp((char *)pn532_packetbuffer, (char *)pn532response_firmwarevers, 6)) {

#define PN532DEBUG

Serial.println(F("Firmware doesn't match!"));

#endif

return 0;

}

response = pn532_packetbuffer[7];

response <<= 8;

response |= pn532_packetbuffer[8];

response <<= 8;

response |= pn532_packetbuffer[9];

response <<= 8;

response |= pn532_packetbuffer[10];


return response;

}

*****/


/*
brief Sends a command and waits a specified period for the ACK
param cmd    Pointer to the command buffer
param cmdlen  The size of the command in bytes
param timeout timeout before giving up
returns 1 if everything is OK, 0 if timeout occurred before an
        ACKNOWLEDGE was received
*/
*****/


// default timeout of one second

boolean SendCommandCheckAck(uint8_t *cmd, uint8_t cmdlen, uint16_t timeout) {

```

```
uint16_t timer = 0;

// write the command
wiresendcommand(cmd, cmdlen);

// Wait for chip to say its ready!
while (wirereadstatus() != PN532_I2C_READY) {
    if (timeout != 0) {
        timer+=10;
        if (timer > timeout)
            return false;
    }
    delay(10);
}

#ifndef PN532DEBUG
Serial.println(F("IRQ received"));
#endif

// read acknowledgement
if (!readackframe()) {
    #ifndef PN532DEBUG
    Serial.println(F("No ACK frame received!"));
    #endif
    return false;
}

return true; // ack'd command
}
```

```

/*****************/
/*
brief Configures the SAM (Secure Access Module)
*/
boolean SAMConfig(void) {
    pn532_packetbuffer[0] = PN532_COMMAND_SAMCONFIGURATION;
    pn532_packetbuffer[1] = 0x01; // normal mode;
    pn532_packetbuffer[2] = 0x14; // timeout 50ms * 20 = 1 second
    pn532_packetbuffer[3] = 0x01; // use IRQ pin!

    if (!SendCommandCheckAck(pn532_packetbuffer, 4,1000))
        return false;

    // read data packet
    wirereaddata(pn532_packetbuffer, 8);

    return (pn532_packetbuffer[6] == 0x15);
}

/****************** ISO14443A Commands *****************/
/*
Waits for an ISO14443A target to enter the field

param cardBaudRate Baud rate of the card
param uid      Pointer to the array that will be populated with the card's UID (up to 7 bytes)
param uidLength  Pointer to the variable that will hold the length of the card's UID.
returns 1 if everything executed properly, 0 for an error
*/

```

```

boolean readPassiveTargetID(uint8_t cardbaudrate, uint8_t * uid, uint8_t * uidLength) {

    pn532_packetbuffer[0] = PN532_COMMAND_INLISTPASSIVETARGET;

    pn532_packetbuffer[1] = 1; // max 1 cards at once (we can set this to 2 later)

    pn532_packetbuffer[2] = cardbaudrate;

    if (!SendCommandCheckAck(pn532_packetbuffer, 3,1000))

    {

        #ifdef PN532DEBUG

            Serial.println(F("No card(s) read"));

        #endif

        return 0x0; // no cards read

    }

    // Wait for a card to enter the field

    uint8_t status = PN532_I2C_BUSY;

    #ifdef PN532DEBUG

        Serial.println(F("Waiting for IRQ (indicates card presence)"));

    #endif

    while (wirereadstatus() != PN532_I2C_READY)

    {

        delay(10);

    }

    #ifdef PN532DEBUG

        Serial.println(F("Found a card"));

    #endif

    // read data packet

    wirereaddata(pn532_packetbuffer, 20);

```

```

// check some basic stuff

/* ISO14443A card response should be in the following format:

byte      Description
-----
b0..6     Frame header and preamble
b7        Tags Found
b8        Tag Number (only one used in this example)
b9..10    SENS_RES
b11       SEL_RES
b12       NFCID Length
b13..NFCIDLen  NFCID          */

#endif MIFAREDEBUG

Serial.print(F("Found ")); Serial.print(pn532_packetbuffer[7], DEC); Serial.println(F(" tags"));

#endif

if (pn532_packetbuffer[7] != 1)

return 0;

uint16_t sens_res = pn532_packetbuffer[9];

sens_res <<= 8;

sens_res |= pn532_packetbuffer[10];

#endif MIFAREDEBUG

Serial.print(F("ATQA: 0x")); Serial.println(sens_res, HEX);

Serial.print(F("SAK: 0x")); Serial.println(pn532_packetbuffer[11], HEX);

#endif

/* Card appears to be Mifare Classic */

*uidLength = pn532_packetbuffer[12];

#endif MIFAREDEBUG

Serial.print(F("UID:"));

```

```

#endif

for (uint8_t i=0; i < pn532_packetbuffer[12]; i++)
{
    uid[i] = pn532_packetbuffer[13+i];

#ifdef MIFAREDEBUG
    Serial.print(F(" 0x"));Serial.print(uid[i], HEX);
#endif
}

#ifdef MIFAREDEBUG
    Serial.println();
#endif

return 1;
}

/***************************************** Mifare Classic Functions *****/
/*
Tries to authenticate a block of memory on a MIFARE card using the
INDATAEXCHANGE command. See section 7.3.8 of the PN532 User Manual
for more information on sending MIFARE and other commands.

param uid      Pointer to a byte array containing the card UID
param uidLen   The length (in bytes) of the card's UID (Should be 4 for MIFARE Classic)
param blockNumber  The block number to authenticate. (0..63 for 1KB cards, and 0..255 for 4KB cards).
param keyNumber Which key type to use during authentication (0 = MIFARE_CMD_AUTH_A, 1 =
MIFARE_CMD_AUTH_B)
param keyData    Pointer to a byte array containing the 6 byte key value
returns 1 if everything executed properly, 0 for an error
*/
/*****************************************/
uint8_t mifareclassic_AuthenticateBlock (uint8_t * uid, uint8_t uidLen, uint32_t blockNumber, uint8_t keyNumber, uint8_t * keyData)

```

```

{

uint8_t len;
uint8_t i;

// Hang on to the key and uid data

memcpy (_key, keyData, 6);

memcpy (_uid, uid, uidLen);

_uidLen = uidLen;

#endif MIFAREDEBUG

Serial.print(F("Trying to authenticate card "));

PrintHex(_uid, _uidLen);

Serial.print(F("Using authentication KEY "));Serial.print(keyNumber ? 'B' : 'A');Serial.print(F(": "));

PrintHex(_key, 6);

#endif

// Prepare the authentication command //

pn532_packetbuffer[0] = PN532_COMMAND_INDATAEXCHANGE; /* Data Exchange Header */

pn532_packetbuffer[1] = 1; /* Max card numbers */

pn532_packetbuffer[2] = (keyNumber) ? MIFARE_CMD_AUTH_B : MIFARE_CMD_AUTH_A;

pn532_packetbuffer[3] = blockNumber; /* Block Number (1K = 0..63, 4K = 0..255 */

memcpy (pn532_packetbuffer+4, _key, 6);

for (i = 0; i < _uidLen; i++){

pn532_packetbuffer[10+i] = _uid[i]; /* 4 byte card ID */

}

if (!SendCommandCheckAck(pn532_packetbuffer, 10+_uidLen,1000))

return 0;

// Read the response packet

```

```

wirereaddata(pn532_packetbuffer, 12);

// Check if the response is valid and we are authenticated
// for an auth success it should be bytes 5-7: 0xD5 0x41 0x00

// Mifare auth error is technically byte 7: 0x14 but anything other than 0x00 is not good

if (pn532_packetbuffer[7] != 0x00)

{
    #ifdef PN532DEBUG
        Serial.print(F("Authentication failed: "));
        PrintHexChar(pn532_packetbuffer, 12);
    #endif
    return 0;
}

return 1;
}

//*****************************************************************************
/*
Tries to read an entire 16-byte data block at the specified block address.

param blockNumber The block number to authenticate. (0..63 for 1KB cards, and 0..255 for 4KB cards).

param data Pointer to the byte array that will hold the retrieved data (if any)

returns 1 if everything executed properly, 0 for an error
*/
//*****************************************************************************

uint8_t mifareclassic_ReadDataBlock (uint8_t blockNumber, uint8_t * data)

{
    #ifdef MIFAREDEBUG
        Serial.print(F("Trying to read 16 bytes from block "));Serial.println(blockNumber);
    #endif
}

```

```

/* Prepare the command */

pn532_packetbuffer[0] = PN532_COMMAND_INDATAEXCHANGE;

pn532_packetbuffer[1] = 1;           /* Card number */

pn532_packetbuffer[2] = MIFARE_CMD_READ;    /* Mifare Read command = 0x30 */

pn532_packetbuffer[3] = blockNumber;      /* Block Number (0..63 for 1K, 0..255 for 4K) */

/* Send the command */

if (!SendCommandCheckAck(pn532_packetbuffer, 4,1000))

{

#define MIFAREDEBUG

Serial.println(F("Failed to receive ACK for read command"));

#endif

return 0;

}

/* Read the response packet */

wirereaddata(pn532_packetbuffer, 26);

/* If byte 8 isn't 0x00 we probably have an error */

if (pn532_packetbuffer[7] != 0x00)

{

#define MIFAREDEBUG

Serial.println(F("Unexpected response"));

PrintHexChar(pn532_packetbuffer, 26);

#endif

return 0;

}

/* Copy the 16 data bytes to the output buffer */

/* Block content starts at byte 9 of a valid response */

```

```

memcpy (data, pn532_packetbuffer+8, 16);

/* Display data for debug if requested */

#ifndef MIFAREDEBUG
Serial.print(F("Block "));
Serial.println(blockNumber);
PrintHexChar(data, 16);
#endif

return 1;
}

/*****************/
/*
Tries to write an entire 16-byte data block at the specified block
address.

param blockNumber The block number to authenticate. (0..63 for 1KB cards, and 0..255 for 4KB cards).
param data The byte array that contains the data to write.
returns 1 if everything executed properly, 0 for an error
Tries to read an entire 4-byte page at the specified address.

param page The page number (0..63 in most cases)
param buffer Pointer to the byte array that will hold the retrieved data (if any)
*/
/*****************/
uint8_t mifareultralight_ReadPage (uint8_t page, uint8_t * buffer)
{
if (page >= 64)
{
#ifndef MIFAREDEBUG

```

```

Serial.println(F("Page value out of range"));

#endif

return 0;

}

#ifndef MIFAREDEBUG

Serial.print(F("Reading page "));Serial.println(page);

#endif

/* Prepare the command */

pn532_packetbuffer[0] = PN532_COMMAND_INDATAEXCHANGE;

pn532_packetbuffer[1] = 1;           /* Card number */

pn532_packetbuffer[2] = MIFARE_CMD_READ; /* Mifare Read command = 0x30 */

pn532_packetbuffer[3] = page;        /* Page Number (0..63 in most cases) */

/* Send the command */

if (!SendCommandCheckAck(pn532_packetbuffer, 4,1000))

{

#ifndef MIFAREDEBUG

Serial.println(F("Failed to receive ACK for write command"));

#endif

return 0;

}

/* Read the response packet */

wirereaddata(pn532_packetbuffer, 26);

#ifndef MIFAREDEBUG

Serial.println(F("Received: "));

PrintHexChar(pn532_packetbuffer, 26);

#endif

```

```

/* If byte 8 isn't 0x00 we probably have an error */

if (pn532_packetbuffer[7] == 0x00)

{

    // Copy the 4 data bytes to the output buffer

    // Block content starts at byte 9 of a valid response

    // Note that the command actually reads 16 byte or 4 pages at a time ... we simply discard the last 12 bytes

    memcpy (buffer, pn532_packetbuffer+8, 4);

}

else

{

    #ifdef MIFAREDEBUG

        Serial.println(F("Unexpected response reading block: "));

        PrintHexChar(pn532_packetbuffer, 26);

    #endif

    return 0;

}

/* Display data for debug if requested */

#ifdef MIFAREDEBUG

    Serial.print(F("Page "));Serial.print(page);Serial.println(F(":"));

    PrintHexChar(buffer, 4);

#endif

// Return OK signal

return 1;

}

```

```

***** high level I2C *****/
/*
brief Tries to read the PN532 ACK frame (not to be confused with
the I2C ACK signal)

*/
***** */

boolean readackframe(void) {
    uint8_t ackbuff[6];

    wirereaddata(ackbuff, 6);

    return (0 == strncmp((char *)ackbuff, (char *)pn532ack, 6));
}

***** mid level I2C *****/
/*
brief Checks the IRQ pin to know if the PN532 is ready
returns 0 if the PN532 is busy, 1 if it is free

*/
uint8_t wirereadstatus(void) {
    uint8_t x = digitalRead(IRQ);

    if (x == 1)
        return PN532_I2C_BUSY;
    else
        return PN532_I2C_READY;
}

/*
brief Reads n bytes of data from the PN532 via I2C

```

```

param buff  Pointer to the buffer where data will be written
param n      Number of bytes to be read

*/
*****void wirereaddata(uint8_t* buff, uint8_t n) {
    uint16_t timer = 0;

    delay(2);

#define PN532DEBUG
    Serial.print(F("Reading: "));
#endif

    // Start read (n+1 to take into account leading 0x01 with I2C)
    WIRE.requestFrom((uint8_t)PN532_I2C_ADDRESS, (uint8_t)(n+2));

    // Discard the leading 0x01
    wirerecv();

    for (uint8_t i=0; i<n; i++) {
        delay(1);

        buff[i] = wirerecv();

#define PN532DEBUG
        Serial.print(F(" 0x"));
        Serial.print(buff[i], HEX);
#endif

    }

    // Discard trailing 0x00 0x00
    // wirerecv();

#define PN532DEBUG
    Serial.println();
#endif

```

```

}

/*****
*/
/*
brief Writes a command to the PN532, automatically inserting the preamble and required frame details (checksum, len,
etc.)

param cmd    Pointer to the command buffer
param cmdlen Command length in bytes
*/
void wiresendcommand(uint8_t* cmd, uint8_t cmdlen) {
    uint8_t checksum;
    cmdlen++;

#ifndef PN532DEBUG
    Serial.print(F("\nSending: "));
#endif

    delay(2); // or whatever the delay is for waking up the board

    // Start the I2C
    WIRE.beginTransmission(PN532_I2C_ADDRESS);
    checksum = PN532_PREAMBLE + PN532_PREAMBLE + PN532_STARTCODE2;
    wiresend(PN532_PREAMBLE);
    wiresend(PN532_PREAMBLE);
    wiresend(PN532_STARTCODE2);
    wiresend(cmdlen);
    wiresend(~cmdlen + 1);

    wiresend(PN532_HOSTTOPN532);
    checksum += PN532_HOSTTOPN532;
}

```

```

#ifndef PN532DEBUG

Serial.print(F(" 0x")); Serial.print(PN532_PREAMBLE, HEX);

Serial.print(F(" 0x")); Serial.print(PN532_PREAMBLE, HEX);

Serial.print(F(" 0x")); Serial.print(PN532_STARTCODE2, HEX);

Serial.print(F(" 0x")); Serial.print(cmdlen, HEX);

Serial.print(F(" 0x")); Serial.print(~cmdlen + 1, HEX);

Serial.print(F(" 0x")); Serial.print(PN532_HOSTTOPN532, HEX);

#endif

for (uint8_t i=0; i<cmdlen-1; i++) {

wiresend(cmd[i]);

checksum += cmd[i];

#ifndef PN532DEBUG

Serial.print(F(" 0x")); Serial.print(cmd[i], HEX);

#endif

}

wiresend(~checksum);

wiresend(PN532_POSTAMBLE);

// I2C STOP

WIRE.endTransmission();

#endif PN532DEBUG

Serial.print(F(" 0x")); Serial.print(~checksum, HEX);

Serial.print(F(" 0x")); Serial.print(PN532_POSTAMBLE, HEX);

Serial.println();

#endif

}

```

```

#include <Keypad.h>

const byte Rows= 4; //number of rows on the keypad

const byte Cols= 3; //number of columns on the keypad

// Defined the key map as on the key pad:

char keymap[Rows][Cols]=

{
{'1', '2', '3'},
{'4', '5', '6'},
{'7', '8', '9'},
{'*', '0', '#'}
};

// A char array is defined as it can be seen on the above
//keypad connections to the arduino terminals is given as:

byte rPins[Rows]={28,27,26,25}; //Rows 0 to 3

byte cPins[Cols]={22,23,24}; //Columns 0 to 2

int entry = 0;

// initializes an instance of the Keypad class

Keypad kpd= Keypad(makeKeymap(keymap), rPins, cPins, Rows, Cols);

LiquidCrystal lcd(32, 33, 34, 35, 36, 37);

// Enter a MAC address and IP address for your controller below.

// The IP address will be dependent on your local network:

byte mac[] = {0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED};

IPAddress ip(192, 168, 0, 140);

unsigned int localPort = 3940; // local port to listen on

```

```

// buffers for receiving and sending data

char ReplyBuffer[] = "acknowledge"; // a string to send back

char alarmStatus = 'd';

// An EthernetUDP instance to let us send and receive packets over UDP

EthernetUDP Udp;

#define W5200_CS 10

#define SDCARD_CS 4

void setup(void){

    // start the Ethernet and UDP:

    pinMode(SDCARD_CS,OUTPUT);

    digitalWrite(SDCARD_CS,HIGH);//Deselect the SD card

    Ethernet.begin(mac, ip);

    Udp.begin(localPort);

    uint32_t PN5xxversion;

    pinMode(IRQ, INPUT);

    pinMode(RESET, OUTPUT);

    Serial.begin(115200);

    Serial.println("NFC shield Start!");

    WIRE.begin();

    // Reset the PN532

    digitalWrite(RESET, HIGH);

    digitalWrite(RESET, LOW);

    delay(500);

    digitalWrite(RESET, HIGH);

    //LCD 16x2

    lcd.begin(16, 2);

    PN5xxversion = FirmwareVersion();
}

```

```

if(! PN5xxversion)
{
    Serial.print("Can't find PN53x, Please check the hardware connection and setting");

    while (1); // halt
}

// Got ok data, print it out!

Serial.print("PNS");

Serial.println((PN5xxversion>>24) & 0xFF, HEX);

Serial.print("Firmware version. ");

Serial.print((PN5xxversion>>16) & 0xFF, DEC);

Serial.print('.');

Serial.println((PN5xxversion>>8) & 0xFF, DEC);

// configure board to read RFID tags

SAMConfig();

}

void loop(void)
{
    //Clears the LCD and displays new data

    lcd.clear();

    lcd.setCursor(0, 0);

    lcd.print("NFC Tag Reader");

    // Preparing the arrays for the data sent by UDP

    char packetBuffer[UDP_TX_PACKET_MAX_SIZE];

    uint8_t success;

    uint8_t uid[] = { 0, 0, 0, 0, 0, 0, 0 }; // Buffer to store the returned UID

    uint8_t uidLength; // Length of the UID (4 or 7 bytes depending on ISO14443A card type)

    boolean verifyUser = false;
}

```

```

boolean verifyPassword = false;

boolean readNFC;

char username[16];

char password[16];

memset(packetBuffer, 0, sizeof(packetBuffer));

// Wait for an ISO14443A type cards (Mifare, etc.). When one is found

// 'uid' will be populated with the UID, and uidLength will indicate

// if the uid is 4 bytes (Mifare Classic) or 7 bytes (Mifare Ultralight)

success = readPassiveTargetID(PN532_MIFARE_ISO14443A, uid, &uidLength);

while(!success){

    lcd.clear();

    lcd.setCursor(0, 0);

    lcd.print("Did not read");

    delay(1000);

}

if(success){

    lcd.clear();

    // Display some basic information about the card

    Serial.println("Found an ISO14443A card");

    Serial.print("ID Length: ");

    Serial.print(uidLength, DEC);

    Serial.println(" bytes");

    Serial.print("ID: ");

    PrintHex(uid, uidLength);

    Serial.println("");



    if(uidLength == 4){

        // Mifare Classic card ...

        Serial.println("Seems to be a Mifare Classic card (4 byte UID)");

        Serial.println("Trying to authenticate block 5 with default KEYA value");
}

```

```

        uint8_t keya[6] = { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF };

        success = mifareclassic_AuthenticateBlock(uid, uidLength, 5, 0, keya);

        if(success){

            Serial.println("Sector 1 (Blocks 4..7) has been authenticated");

            uint8_t data[16];

            // Try to read the contents of block 5

            success = mifareclassic_ReadDataBlock(5, data);

            for(int i=0; i < 16; i++){

                username[i] = data[i];

                Serial.print(username[i]);

                readNFC =true;

            }

            Serial.println(" ");

            Serial.println("Reading UDP Packet");



// /* *****----- UDP Communication Comparison with NFC -----***** */

        while(readNFC == true){

            Serial.println("ReadNFC");

            memset(packetBuffer, 0, sizeof(packetBuffer));

            int packetSize = Udp.parsePacket();

            IPAddress remote = Udp.remoteIP();

            Serial.println(packetSize);

            if (packetSize){

                // read the packet into packetBufffer

                Udp.read(packetBuffer, UDP_TX_PACKET_MAX_SIZE);

                Serial.print("Connection: ");

```

```

Serial.println(packetBuffer[0]);
}

if(packetBuffer[0] == 'c'){
    Serial.println("Connected");

/*-----Username verification-----*/
    Udp.beginPacket(Udp.remoteIP(), Udp.remotePort());

    Udp.write(username);
    Udp.endPacket();

    delay(355);

    memset(packetBuffer, 0, sizeof(packetBuffer));
    packetSize =0;
    packetSize = Udp.parsePacket();

    if(packetSize){

        Udp.read(packetBuffer,UDP_TX_PACKET_MAX_SIZE);
        Serial.print("Server Username: ");
        Serial.println(packetBuffer[0]);
    }
}

/*----- Verifying user -----*/
if(packetBuffer[0] == 'u'){

    Serial.println("Username verified");
    verifyUser = true;

    memset(packetBuffer, 0, sizeof(packetBuffer));

}else if(packetBuffer[0] == 'n'){

    verifyUser = false;

    memset(packetBuffer, 0, sizeof(packetBuffer));
}

```

```

/*-----LCD Display User verification-----*/
readNFC = false;

delay(10);

// If the user is verified it asks for the password

if(verifyUser == true){

    lcd.clear();

    lcd.setCursor(0, 0);

    lcd.print("Identified!");

    delay(1500);

    lcd.clear();

    lcd.setCursor(0, 0);

    lcd.print("Enter password: ");

    Serial.println("Enter password: ");

}

//*****-----KEYPAD-----*****
while(verifyUser == true){

    char keypressed = kpd.getKey();

    for(int i=0; i<4; i++){

        if (keypressed != NO_KEY){

            if(i == entry){

                password[i] = keypressed;

                lcd.setCursor(i, 1);

                lcd.print("*");

                Serial.print(password[i]);

            }

        }

    }

    if (keypressed != NO_KEY){

        entry++;

    }

}

```

```
}

if(entry == 4){

    readNFC = false;

/*-----Password verification-----*/
    Udp.beginPacket(Udp.remoteIP(), Udp.remotePort());

    Udp.write(password);

    Udp.endPacket();

    delay(350);

    memset(packetBuffer, 0, sizeof(packetBuffer));

    int packetSize =0;

    packetSize = Udp.parsePacket();

    if(packetSize){

        Udp.read(packetBuffer,UDP_TX_PACKET_MAX_SIZE);

        Serial.print("Password Reply: ");

        Serial.println(packetBuffer[0]);

    }

    if(packetBuffer[0] == 'p'){

        verifyPassword = true;

        verifyUser = false;

        readNFC = false;

        lcd.clear();

        lcd.setCursor(0, 0);

        lcd.print("Alarm Status");

        lcd.setCursor(0, 1);

        lcd.print("Changed");

    }

}else if(packetBuffer[0] == '0'){

    verifyPassword = false;

    readNFC = false;
```

```

verifyUser = false;

lcd.clear();

lcd.setCursor(0, 0);

lcd.print("Wrong Password");

}

entry = 0;

}

}

}else if (verifyUser == false && strlen(packetBuffer) != 0){

lcd.clear();

lcd.setCursor(0, 0);

lcd.print("User not found!");

delay(1500);

memset(packetBuffer, 0, sizeof(packetBuffer));

memset(username, 0, sizeof(username));

memset(password, 0, sizeof(password));

lcd.clear();

}

}

}

/****************************************/

if(success)

{

// Data seems to have been read ... spit it out

Serial.println("Reading Block 5:");




Serial.println("");

// Wait a bit before reading the card again

delay(1000);

}

else

```

```
{  
    Serial.println("Ooops ... unable to read the requested block. Try another key!");  
}  
}  
}  
else  
{  
    Serial.println("Ooops ... authentication failed: Try another key!");  
}  
}  
}  
}  
}
```

Figure 11.4.6 – User Verification Code for Arduino

Figure 11.4.7 presents the Verification System after being tested with the users that were created at the database. In the presented case, the user and the password were verified because of the response obtained from the database.

```
Found an ISO14443A card
ID Length: 4 bytes
ID: 0x81 0xD7 0x1B 0x32

Seems to be a Mifare Classic card (4 byte UID)
Trying to authenticate block 5 with default KEYA value
Sector 1 (Blocks 4..7) has been authenticated
caps01user
Reading UDP Packet
ReadNFC
1
Connection: c
Connected
Server Username: u
Username verified
Enter password:
1234
Password Reply: p
Reading Block 5:
```

Figure 11.4.7 – User Verification Serial Console Result

Chapter 12: Network

12.1 Alternatives Considered

12.1.1 Internet of Things (IoT)

The Machine-to-Machine (M2M) communication has been used for a long time, but it was never recognized as the Internet of Things (IoT) until in 1999 Kevin Ashton used the phrase “Internet of Things” while working at MIT’s Media Center. What he was trying to represent was the concept of machines with sensors that were communicating thru a network to report their status and accept control commands.

The IoT communication technologies are advancing very fast. The IoT comprises many uses, ranging from personal devices to industrial equipment. Is because of this huge diversity of applications and environments that there are different approaches to create wireless standards for connectivity.



Figure 12.1.1 – Internet of Things (IoT)

The machines or devices in the IoT are classified as passive, active and autonomous. In order for a device to be classified as working with the IoT it must be capable of at least communicating its status, performance and/or location. Active devices are those that are able to communicate as well as perform actions depending on their status, while passive devices just need to communicate data without the necessity of reacting about it. In order for a system to be qualified as “smart”, the system must be able to do both, communicate data and be capable of making decisions based on the conditions.

Smart phones play a big role in pursuance of creating environments where not just the machines are capable to interact with each other, but that they are also capable of informing people about their status and decision-making. These can be connected to any kind of smart devices as long as they are supported by the same protocol (language). The ability to create powerful, simple and user-friendly interfaces for people to interact with the devices around them is what will allow the IoT concept to be successful among every type of users.

IoT requires complex technologies that allow connectivity like Ethernet, Zigbee, Wi-Fi, or Bluetooth among many others. Choosing the right wireless connectivity technology for an IoT application can be challenging with so many standards in the market, spreading over multiple frequency bands and using different communication protocols. To identify the right technology, key technical tradeoffs must be analyzed in order to identify which is more favorable for a specific project.

There are so many advantages in the IoT that are motivating industries, organizations, and the technologists to explode the concept, that the concept will become a huge market disruption, and those that benefit from it now, will eventually reap an immense advantage and will become leaders.

12.1.2 OSI Model

A communication protocol is a system of rules and standards to format and control data exchange within or between computers. To implement networking protocols, the protocol software modules are interfaced with a framework implemented on the machine's operating system. This framework implements the networking functionality of the operating system. The protocols can be arranged based on functionality groups that are mapped onto layers, each layer solving a distinct class of problem.

The most common model in data communication systems is the Open Systems Interconnection (OSI) model, which breaks the communication into 7 functional layers that allow easier implementation of scalable and interoperable networks. It is a general-purpose template for discussing or describing how computers communicate with one another over a network.

The transmitting computer software gives the data to be transmitted to the application layer, where it is processed and passed from layer to layer down the stack with each layer performing its designated functions. The data is then transmitted over the physical layer of the network until the destination computer or another device receives it. At this point the data is passed up through the layers again, each layer performing its assigned operations until the data is used by the receiving computer's software.

During the transmission, a process called encapsulation occurs where each layer adds a header to the data that directs and identifies the packet. The header and data together form the data packet for the next layer that, in turn, adds its header and so on. The combined encapsulated packet is what is transmitted and received. The receiving computer reverses the process, de-encapsulating the data at each layer with the header information directing the operations.

All of the necessary operations required are grouped together in a logical sequence at each of the layers.

Layer 7 – Application

- This layer works with the application software to provide communication functions as required. It verifies the availability of a communications partner and the resources to support any data transfer. It also works with end applications such as:

- § DNS (Domain Name Service)
- FTP (File Transfer Protocol)
- HTTP (Hypertext Transfer Protocol)
- IMAP (Internet Message Access Protocol)
- POP (Post Office Protocol)
- SMTP (Simple Mail Transfer Protocol)
- Telnet

Layer 6 – Presentation

- This layer checks the data to ensure that it is compatible with the communications resources. It ensures compatibility between the data formats and the applications level and the lower levels. It also handles any needed data formatting or code conversion, as well as data compression and encryption.

Layer 5 – Session

- This layer handles authentication and authorization functions. It also manages the connection between the two communicating devices, establishing a connection, maintaining the connection, and ultimately terminating it. This layer verifies that the data is delivered as well.

Layer 4 – Transport

- This layer provides quality of service (QoS) functions and ensures the complete delivery of the data. The integrity of the data is guaranteed at this layer via error correction and similar functions.

Layer 3 – Network

- The network layer handles packet routing via logical addressing and switching functions.

Layer 2 – Data Link

- This layer handles operating package and unpack the data in frames.

Layer 1 – Physical

- This layer defines the logic levels, data rate, physical media, and data conversion functions that make up the bit stream of packets from one device to another.

The OSI model is the base for most of protocols and systems. It is mainly useful for discussing, describing, and understanding individual network functions.

12.1.3 TCP/IP

The TCP/IP is also a layered protocol but it uses only four layers that are equivalent in operation and function to that on the OSI model.

Layer 4 - Application

- This layer is similar to OSI layers 5,6, and 7 combined.

Layer 3 – TCP (Transport)

- This layer is equivalent to layer 3 in OSI model.

Layer 2 – IP (Internet)

- The layer is comparable to layer 3 in the OSI model.

Layer 1 – Network Interface

- This layer is equivalent to OSI layers 1 and 2.

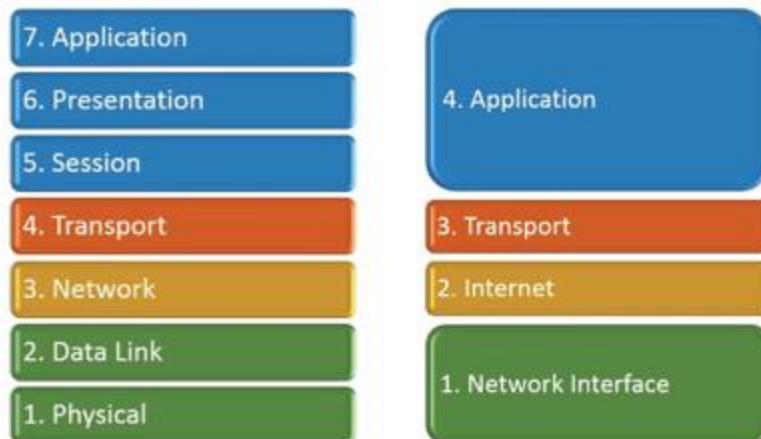


Figure 12.1.2 – OSI and TCP/IP Network Model

12.1.3 UDP

The User Datagram Protocol uses a simple connectionless transmission model with a minimum of protocol mechanism. It has no handshaking dialogues, and thus exposes the user's program to any unreliability of the underlying network protocol. There is no guarantee of delivery, ordering, or duplicate protection. UDP provides checksums for data integrity, and port numbers for addressing different functions at the source and destination of the datagram. With UDP, computer applications can send messages , in this case referred to as datagrams, to other hosts on an internet protocol (IP) network without prior communications to set up special transmission channels or data paths. UDP is suitable for purposes where error checking and correction is either not necessary or is performed in the application, avoiding the overhead of such processing at the network interface level. Time-sensitive applications often use UDP because dropping packets is preferable to waiting for delayed packets, which may not bean option in a real-time system. If error correction facilities are needed at the network interface level, an application may use the Transmission Control Protocol (TCP) or Stream Control Transmission Protocol (SCTP) which are designed for this purpose. This being said, the RMF Intelligent homes application, being a real time system, UDP should not be utilized but, the handshaking dialogues can be managed inside the RMF Intelligent Home Application in order to let the user know the request is being processed. This particular protocol can be utilized within the android environment using UDP sockets. With the combination of UDP sockets and handshake management the user can rely on the systems functionality.

After a limit of tries the socket communication is then closed and the user is notified of system communication failure. This thus creates the effect of a TCP communication but without timing out.

12.1.4 Static IP Address

To get a single IP address, you need to buy a server-hosting package from an Internet Service Provider (ISP). Most of the times, these IP addresses have different sizes, typically separated into classes of 8 bits, 16 bits, and 24 bits.

12.1.5 Dynamic IP Address

It is not always necessary to choose an IP address for every device we connect to a network. Instead, when a device is connected it can request an IP address from the network itself using the Dynamic Host Configuration Protocol (DHCP). When the device tries to connect, instead of checking its internal configuration for its address, it sends a message to the router asking for an address. This is not a static IP address, which belongs to the device indefinitely; rather, it is temporary which is selected dynamically according to which addresses are currently available. If the router is rebooted or the device is switched off, some other device may end up with that IP address.

This means that you can't simply point a DNS entry to a device using a DHCP. In general, you can rely on the IP address probably being the same for a given work session, but you should not hard-code the IP address anywhere that you might try to use it another time, when it might have changed.

12.1.6 MAC Address

Just as every network-connected device has an IP address, they all have MAC addresses, which is like the final address on a physical envelope. It is used to differentiate different machines on the same physical network so that they can exchange packets. This is related to the lowest level “link layer” of the TCP/IP stack. Since MAC addresses are globally unique, they do not typically get used outside of one Ethernet network (for example, beyond a home router). So, when an IP message is routed, it hops from node to node, and when it finally reaches a node which knows where the physical machine is, that node passes the message to the device associated with that MAC address.

MAC stands for Media Access Control. It is a 48-bit number, usually written as six groups of hexadecimal digits, separated by colons.

Most devices, such as your laptop, come with the MAC burned into their Ethernet chips. Some chips, such as the Arduino Ethernet WizNet, do not have a hard-coded MAC address, though. This is for production reasons: if the chips are mass-produced, they are identical. So they can't, physically, contain a distinctive address. The address could be stored in the chip's firmware, but this would then require every chip to be built with custom code compiled in the firmware. Alternatively, one could provide a simple data chip, which stores just the MAC address, and have the WizNet chip read that.

Most consumer devices use some similar process to ensure that the machine always start up with the same unique MAC address. The Arduino board, as a low-cost prototyping platform for developers, does not bother with that nicety, to save time and cost. Yet it does come with a sticker with a MAC address on it. Although this might seem a bit odd, there is a good reason for it: that MAC address is reserved and therefore is guaranteed unique if you want to use it. For development purposes, you can simply choose a MAC address that is known not to exist in your network.

12.1.7 Network Range

Typically networks range is categorized into four classes: Personal Area Network (PAN), Local Area Network (LAN), Metropolitan Area Network (MAN), and Wide Area Network (WAN). These four categories are illustrated in Figure 11.1.3.

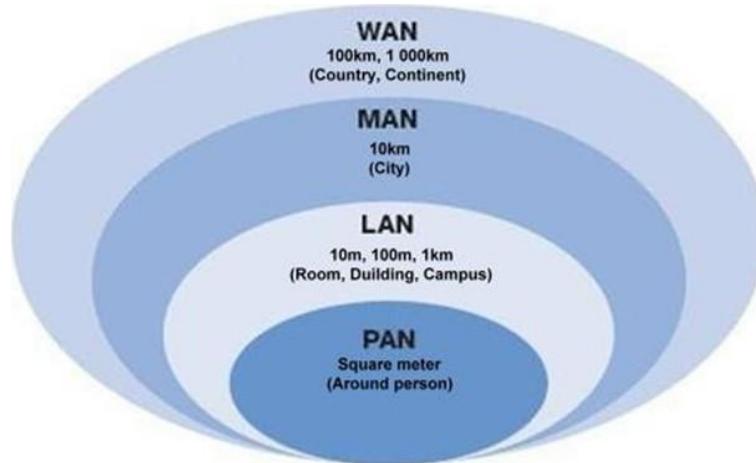


Figure 12.1.3 – Different Area Networks

PANs are computer networks organized around an individual person within a single building. Usually a wireless PAN covers a range of about 10 meters.

A typical PAN includes one or more computers, phones, video game consoles, and other personal devices. Sometimes multiple individuals use the same network within a residence; hence the network is sometimes referred to as Home Area Network (HAN).

In a very typical setup, a residence will have a single wired Internet connection connected to a modem. The modem provides both wired and wireless connections for multiple devices. The network is typically managed from a single computer but can be accessed from any device.

LAN consists of a computer network at a single site, typically an individual office building. These can be built with relatively inexpensive hardware, such as hubs, network adapters, and Ethernet cables. They are very useful for sharing resources, such as data storage and printers.

A small LAN may only use two computers, while larger LANs can accommodate thousands of computers. A typical LAN relies mostly on wired connections for increased speed and security, but wireless connections are also part of the LANs. Wireless LANs (WLANS) usually cover a range up to 100 meters. A predominant example is a home Wi-Fi network providing Internet access to personal computers, TVs, and even to IoT devices such as thermostats and home appliance.

A MAN consists of a computer network across an entire city, college campus, or small region. A MAN is larger than a LAN, which is typically limited to a single building or site. Depending on the configuration, this type of network can cover an area from 10 km to 25 km, and even larger.

An example of a MAN is a smart grid network used to transmit electric meters readings from homes to the utility company using a proprietary protocol over a 900 MHz radio.

A WAN occupies a very large area, such as an entire country or the entire world. A WAN can contain multiple smaller networks, such as LANs or MANs. The Internet is the best-known example of a public WAN that is built of a complex mix of wired and wireless connections. These usually occupy areas over 100 km.

12.1.8 Mesh Network Topology

Another network's categorization is by their topology. The topology of a network refers on the way nodes in the network are arranged and connected to each other. Topologies can be either physical or logical. A physical topology would refer to the physical layout of devices on a network. A logical topology refers to the way that the signals act on the network media, or the way that the data passes through the network from one device to the next.

In a mesh network, every node can connect to multiple other nodes. This provides redundant communication paths between some or all devices. One or more nodes in the network serve as an Internet gateway. The Figure 2.6.1 presents a partial mesh topology where some of the nodes have just one connection, while others have more than one connection.

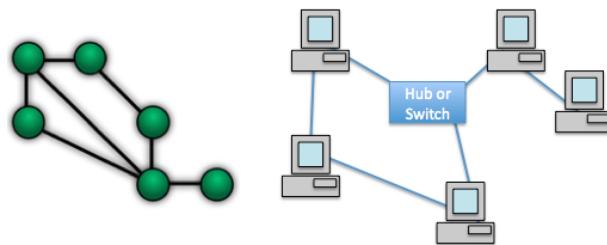


Figure 12.1.4 – Mesh Topology

A popular example of a mesh network is a ZigBee Light Link network where multiple lights from a mesh network to extend the network reach in large buildings. One of the ZigBee nodes is called a coordinator, and it usually serves also as the Internet gateway.

Network size, or the maximum number of simultaneously connected devices, is an important consideration in system design. Some technologies like Bluetooth support up to 20 connections, while other technologies like ZigBee can support thousands of connections.

Advantages:

- A benefit from a mesh topology is that it can extend the range of the network through multiple hops, while maintaining low radio transmission power.
- Can withstand high traffic.
- They can also achieve better reliability by enabling more than one path to relay a message through the network.
- The expansion and modification in topology can be done without disrupting other nodes.

Disadvantages:

- Compared to other topologies like star network, mesh networks are more complex to design and can exhibit longer delay routing a message from a remote node through the mesh.
- Overall cost of this network is way too high as compared to other topologies.
- Set-up and maintenance of this topology is very difficult. Even the administration of the network is tough.

12.1.9 Star Network Topology

In a star network, devices are connected to a central computer called a hub, a router or a switch. Nodes communicate across the network by passing data through the hub. Every workstation is indirectly connected to every other through the central computer.

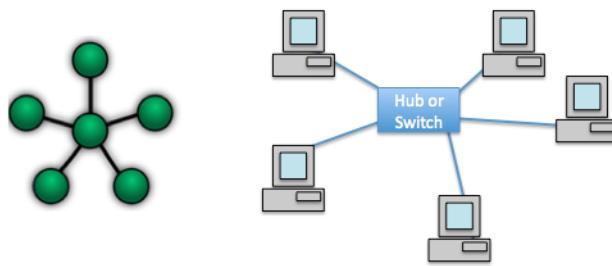


Figure 12.1.5 – Star Topology

All the data on the star topology passes through the central device before reaching the intended destination. The hub acts as a junction to connect different nodes present in the network, and at the same time it manages and controls whole of the network. Depending on which central device is used, the hub can act as a repeater or a signal booster.

Advantages:

- In a star network, one malfunctioning node doesn't affect the rest of the network.
- Compared to bus topology, it gives fam much better performance, signals do not necessarily get transmitted to all the workstations.
- It is easy to connect new nodes or devices, as well as to remove them.
- Centralized management helps to monitor the network.

Disadvantages:

- If the central computer (hub) fails, the entire network becomes unstable.
- The use of hub, a router, or a switch as a central device increases the overall cost of the network.
- Performance as well as number of nodes that can be added in such topology is dependent on the capacity of the central device.

12.1.10 Ring Network Topology

In ring topology, all the nodes are connected to each other in such a way that they make a closed loop. Each workstation is connected to two other components on either side, and it communicates with these two adjacent neighbors. Data travels around the network, in one direction.

Sending and receiving of data takes place by the help of Tokens. The token contains a piece of information that along with data is sent by the source computer. The token passes from computer to computer, until the destination computer recognizes that it is intended to it.

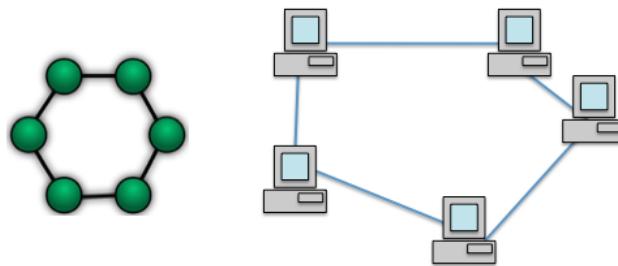


Figure 12.1.6 – Ring Topology

Advantages:

- This network topology is very organized. Each node gets to send the data when it receives an empty token. This helps to reduce the chances of collision. Also the traffic flows in only one direction at very high speed.
- Even when the load on the network increases, its performance is better than that of the bus topology.
- Additional components do not affect the performance of the network.
- Each computer has equal access to resources.

Disadvantages:

- Each packet of data must pass through all the computers between the source and destination, making it very slow compared to star topology.
- If one workstation or port goes down, the entire network gets affected.
- Network is highly dependent on the wire that connects different components.
- The network cards are expensive as compared to Ethernet cards and hubs.

12.1.11 Tree Network Topology

The tree topology integrates the characteristics of star and bus topologies. In this topology, the numbers of star networks are connected using a bus. The bus is the main cable that seems like a main stem of a tree, and other star networks are the branches.

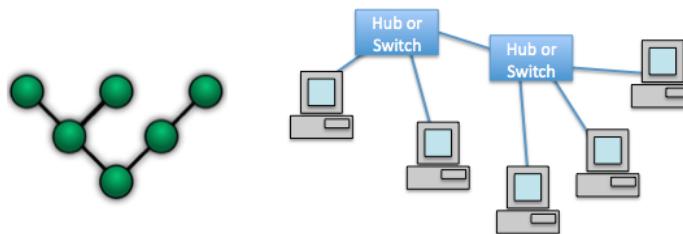


Figure 12.1.7 – Tree Topology

It is also called Expanded Star Topology. Ethernet protocol is commonly used in this type of topology. The diagram in Figure 2.6.4 makes a clear explanation of the configuration.

Advantages:

- It is an extension of star and bus topologies, so in networks where these topologies can not be implemented individually for reasons related to scalability, tree topology is the best alternative.
- Expansion of the network is easy.
- Since the network is divided into segments (star networks), it can be easily managed and maintained.
- Error detection and correction is easy.
- Each segment is provided with dedicated point-to-point wiring to the central hub.
- If one segment is damaged, other segments are not affected.

Disadvantages:

- Because of its basic structure, the tree topology relies heavily on the main bus cable, so if it breaks the whole network is crippled.
- As more and more nodes and segments are added, the maintenance becomes difficult.
- Scalability of the network depends on the type of cable used.

12.1.12 Bus Network Topology

The bus topology is the simplest of the topologies. In this network all the nodes (computers as well as servers) are connected to the single cable (called bus) with the help of interface connectors. This bus is the backbone of the network and every workstation communicates with the other device through this bus. The Figure 2.6.5 presents an example of a simple bus network.

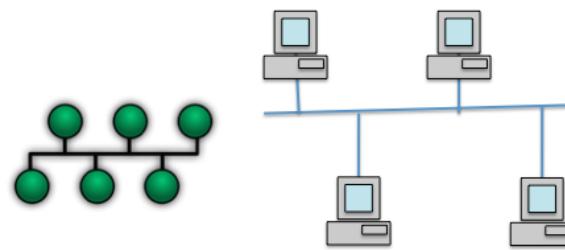


Figure 12.1.8 – Bus Topology

A signal from the source is broadcasted and it travels to all workstations connected to bus cable. Although the message is broadcasted to all the devices, only the intended recipient (which matches the MAC address or IP address) accepts it. If the MAC/IP address of machine does not match with the intended address, the machine discards the signal. A terminator is added at the ends of the central cable to prevent bouncing of signals.

Advantages:

- It is easy to set-up and to extend.
- The cable length required for this topology is the least compared to the other networks.
- It is not expensive.
- Good for LANs.

Disadvantages:

- There is a limit on central cable length and number of nodes that can be connected.
- If the main cable encounters a problem, the whole network breaks down.
- Proper termination is required to dump signals (terminators).
- It is difficult to detect and troubleshoot fault at an individual station.
- Maintenance costs can get high with time.
- Efficiency is reduced as the number of connected devices increase.
- It is not suitable for networks with heavy traffic.
- Security is very low because all the computers receive the signal sent by the source.

12.1.13 Wi-Fi Connectivity

Based on the IEEE 801.11 standard, the Wi-Fi technology was developed as a wireless replacement for the popular wired IEEE 802.3 Ethernet standard. As such, it was created from the first day for Internet connectivity. Although Wi-Fi technology primarily defines the link layer of a local network, it is so natively integrated with the TCP/IP stack, that when people say they are using Wi-Fi they implicitly mean that they are also using a TCP/IP for Internet connectivity.

Wi-Fi Access Points (APs) are deployed today in most homes, as well as in almost all offices, schools, airports, and retail stores. It has become so common that people often refer to it as just “wireless”. The huge success of Wi-Fi is largely due to the remarkable interoperability programs run by the Wi-Fi Alliance and to the increasing demand in the market for easy and cost-effective Internet access.

Wi-Fi networks have a star topology, with the Access Point being the Internet gateway. The output power of Wi-Fi is high enough to allow full in-home coverage in most cases. In enterprise and large buildings, more than one Access Point is often deployed in different locations inside the building to increase the network coverage. To overcome dead signal reception spot in some cases, various Wi-Fi products include two antennas for diversity.

Although most Wi-Fi networks operate in the ISM 2.4 GHz band, some operate in the 5 GHz bands, where more channels exist and higher data rates are available. However, since the range of 5 GHz radios inside buildings is shorter compared to 2.4GHz, 5 GHz is mainly used in enterprise applications along with multiple Access Points to ensure good coverage.

Wi-Fi and TCP/IP software are fairly large and complex. Until recently, adding Wi-Fi connectivity to devices with little processing power such as thermostats and home appliances was not possible or not cost effective. Today, silicon devices and modules coming out on the market embed the Wi-Fi software and the TCP/IP software inside the device. These new devices eliminate most of the overhead from the microprocessors (MPUs) and enable wireless Internet connectivity with the smallest microcontroller (MCU). The increasing level of integration in these Wi-Fi devices also eliminate all required radio design experience and reduces the barriers of Wi-Fi integration.

To enable high data rates (over 100 Mbs in certain cases) and good indoor coverage, Wi-Fi radios have fairly large power consumption. For some IoT devices, which run on batteries and cannot be charged frequently, Wi-Fi can be too power consuming. Although the peak current of Wi-Fi radios cannot be reduced too much, new silicon devices apply advanced sleep protocols and fast on/off time to reduce the average power consumption significantly. Since most IoT products do not need the maximum data rates, Wi-Fi offers clever power management design that can efficiently draw bursts of current from the battery for very short intervals and keep products connected to the Internet for over a year using AA alkaline batteries.

Most Wi-Fi Access Points can support up to 250 simultaneously connected devices. Enterprise-grade Access Points can support even larger number of connections, and some popular consumer Access Points handle no more than 50.

To summarize, Wi-Fi is the most common wireless Internet connectivity technology today. Its high power and complexity has been a major barrier for IoT developers, but new technologies are reducing many of these barriers, enabling Wi-Fi integration into emerging IoT applications and battery-operated devices.

12.1.14 Bluetooth Connectivity

Ericsson developed Bluetooth technology in 1994 as a standard for wireless communication between phones and computers. The Bluetooth link layer operates in the 2.4 GHz ISM band, and it was previously standardized as IEEE 802.15.1, but today the IEEE standard is no longer maintained and Bluetooth standard is controlled by the Bluetooth SIG.

Bluetooth became very successful in mobile phones. The main use case that made Bluetooth popular initially was hands-free phone calls with headsets and car kits. Thereafter, as mobile phones became more capable, more use cases like high fidelity music streaming and data-driven cases such as health and fitness accessories evolved.

Bluetooth is a PAN technology primarily used today as cable replacement for short-range communication. It supports data throughput up to 2Mbs, and although more complex topologies are included in its specifications, Bluetooth is primarily used in a point-to-point or in a star network topology. The technology is fairly low power; devices typically use small rechargeable batteries, or two alkaline batteries.

Bluetooth Low Energy (Bluetooth Smart) is a more recent addition to Bluetooth specification. Designed for lower data throughput, Bluetooth low energy significantly reduces the power consumption of Bluetooth devices and enables years of operation using coin cell batteries. Supported by the new generation of smartphones and tablets, Bluetooth Low Energy has accelerated the Bluetooth market growth and enabled a wide range of new applications and introduced proximity capabilities that opened the door to location-based services like beaconing and geo-fencing applications.

The Bluetooth classic standard can support up to eight devices connected in a star network simultaneously. The Bluetooth Low Energy standard removes this limitation and can theoretically support an unlimited number of devices, but the practical number of simultaneously connected devices is between 10 and 20.

One of the advantages of the Bluetooth standard is that it includes application profiles. These profiles define in great detail how applications exchange information to achieve a specific task. To name one example, the Audio/Video Remote Control Profile (AVRCP) defines how a Bluetooth remote control interfaces with audio and video equipment to relay commands like play, pause, stop, etc. The comprehensive certification programs defined by the Bluetooth SIG cover the entire protocol stack as well as the application profile, helping the Bluetooth achieve excellent interoperability in the market.

12.1.15 ZigBee Connectivity

ZigBee technology is based on the IEEE 802.15.4 link layer standard. It is a low-throughput, low-power, and low-cost technology. Mainly operates in the 2.4 GHz ISM band although the spec also supports the 868 MHz and 915 MHz ISM bands. ZigBee can deliver up to 250 KBps of data throughput, but is typically used at much lower data rates. It also has the capability to maintain very long sleep intervals and low operation duty cycles to be powered by coin cell batteries for a year. This technology has a mesh nature, where data hops from node to node in multiple directions and path throughout large-scale networks.

Although the ZigBee standard has an IP specification, it is separated from the popular smart energy, home automation, and light link profile, and has not gotten much traction in the industry. To connect the IoT, ZigBee networks require an application-level gateway. The gateway participates as one of the nodes in the ZigBee network and in parallel runs a TCP/IP stack and application over Ethernet or Wi-Fi to connect the ZigBee network to the Internet.

12.1.16 LowPAN Connectivity

6LoWPAN is an acronym for IPv6 over Low power Wireless Personal Area Network. The main goal of 6LoWPAN is to apply IP to the smallest, lowest-power and most limited processing power device. 6LoWPAN is the first wireless connectivity standard that was created for the IoT. Although it includes the term “Personal Area Networks” within the 6LoWPAN acronym, it is typically used to form LANs.

The standard was created by the 6LoWPAN working group of the IETF and formalized under RFC 6282 “Compression format for IPv6 datagrams over IEEE 802.15.4-based networks”, in September 2011. As indicated by the RFC title, the 6LoWPAN standard only defines an efficient adaptation layer between the 802.15.4 link layer and a TCP/IP stack.

The term 6LoWPAN is loosely used in the industry to refer to the entire protocol stack that includes the 802.15.4 link layer, the IETF IP header compression layer and a TCP/IP stack. But there is no industry standard for the entire protocol stack, nor is there a standard organization to run certification programs for a 6LoWPAN solution. Since the 802.15.4 link layer has multiple optional modes, different vendors can implement solutions that are not interoperable at the local network level, and still call all of them “6LoWPAN networks”. Although this occurs, devices running on different networks can communicate with each other over the Internet, provided that they are connected to any other IP-based server or device on the Internet, including Wi-Fi and Ethernet devices.

IPv6 was chosen as the only supported IP in 6LoWPAN (excluding IPv4) because it supports a larger addressing space, hence much larger networks, and also because it has built-in support for network auto configuration.

6LoWPAN networks require an Ethernet or Wi-Fi gateway to access the Internet. Similar to Wi-Fi, the gateway is an IP-layer gateway and not an application layer gateway, which allows 6LoWPAN nodes and applications direct access to the Internet. Since most of the deployed Internet today is still using IPv4, a 6LoWPAN gateway typically includes an IPv6 to IPv4 conversion protocol.

6LoWPAN is fairly new to the market. Initial deployments use both the 2.6 GHz and the 868 MHz/915 MHz ISM bands. Building on the 802.15.4 advantages – mesh network topology, large network size, reliable communication, low power consumption, and all the benefits of IP communication. 6LoWPAN is well positioned to fuel the exploding market of Internet-connected sensors and other low data throughput and battery-operated applications.

12.1.17 Sub-1GHz Connectivity

The new IEEE Wi-Fi standard 802.11ah that uses the 900 MHz band could be the solution for low-power wireless data at urban networks, large corporate campuses, hospitals, universities, and other large facilities that need Wi-Fi connectivity all over the place for a large number of sensors, smart meters, and other connected devices.

The IEEE 802.11ah could be able to solve many problems with deploying large-footprint Wi-Fi networks by allowing a significant number of devices, providing power saving services and long distance to the Access Point (AP).

A typical Access Point could associate more than 8,000 devices through a hierarchical ID structure within a range of 1 km, making it ideal for areas with high concentration of sensors and other small devices, such as street lamp controllers and smart parking meters. It can use the 900 MHz band with high reliability because of the limited use of the Sub-1GHz band.

The 802.11ah standard also includes a new PHY and MAC layers, grouping devices into traffic indication maps (TIMs) to accommodate small units (such as sensors) and Machine-to-Machine communications.

The PHY layer will allow devices and Access Points to operate over different Sub-1GHz bands, depending on the country regulations:

- 863-868 MHz in Europe
- 902-928 MHz in USA
- 916.5-927.5 in Japan

The 900 MHz band is currently used in Europe for GSM 2G cellular services (the US uses the 950 MHz and 1,900 MHz bands). Though most carriers have moved to the faster 1,800 MHz and 3G UMTS, many devices are still using the 900 MHz. Unfortunately, that requires the segmentation of devices for different regions, but that is already the case for most M2M communications and small sensors.

12.1.18 Ethernet Connectivity

The wireless connectivity market is growing very fast because of the IoT. Nevertheless, there are many IoT applications that are connected to the Internet with wires. Some examples are the Ethernet connectivity, power line communication (PLC), and Industrial communication standards such as Fieldbus. The scope of this project just approaches the Ethernet connectivity because it is the most commonly used for the project purposes.

The Ethernet is the most widely installed local area network (LAN) technology, and the IEEE standard 802.3 defines it. Typically the Ethernet LANs use coaxial cables or twisted pair wires.

There are various Ethernet ports; the most commonly installed Ethernet systems contain the 10/100 Ethernet ports that support transmission speeds of 10Mbps and 100Mbps. Systems that can provide transmission speeds up to 10 Mbps are called 10BASE-T, while those that provide transmission speeds up to 100 Mbps are called Fast Ethernet or 100BASE-T.

Newer and higher level Ethernet systems are commonly used for backbone support. Gigabit Ethernet systems provide ports 10/100/1000, which are able to support transmission speeds of 1 Gbps. 10-Gigabit Ethernet systems are able to provide up to 10 billion bits per second (10Gbps). When two ports with different transmission speeds are connected, they negotiate with each other to transmit at the highest common speed.

The only two possible topologies for Ethernet are bus and star. The Ethernet transmits variable length frames up to 1500 bytes in length, each of these containing a header with the addresses of the source and destination stations and a trailer that contains error correction data. Higher-level protocols, most notably IP, break apart longer messages into required frame size.

Ethernet uses CSMA/CD (Carrier Sense Multiple Access with Collision Detection). In this method, multiple workstations access a transmission medium (multiple access) by listening until no signals are detected (carrier sense). After this, they transmit and check to see if more than one signal is present (collision detection). If there is a collision, each station attempts to retransmit after a preset delay, which is different for each station.

12.1.19 Arduino Ethernet Shields

Even the simplest computing devices such as the Arduino board can use DHCP. Although the Arduino's Ethernet library allows you to configure a static IP address, you can also request one via DHCP. Using a static address may be fine for development (if you are the only person connected to it with that address), but for working in groups or preparing the device to be distributed to other people on arbitrary networks, you almost certainly want a dynamic IP address.

The Arduino Ethernet shield allows connectivity to the router utilizing the RJ-45 connectors. This provides visibility from anywhere in the network and assigns the microcontroller an IP address utilizing the DHCP. In this case, our demonstration will consist of giving the microcontroller a static IP address to turn on a LED through the network connected locally and receiving acknowledgement from the arduino. This project is based on a UDP server that listens and acknowledges through UDP packets.

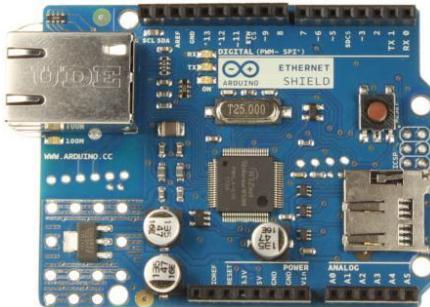


Figure 12.1.9 – Arduino Ethernet Shield

12.1.20 nRF24L01 Module

The nRF24L01 is a single chip transceiver that operates in the 2.4 GHZ ISM band with an embedded baseband protocol engine (Enhanced ShockBurst), suitable for ultra low power wireless applications.



Figure 12.1.10 – nRF24L01 Module

This transceiver is very easy to use, to design a radio system with it; you simply need an MCU (microcontroller) and a few external passive components. It can be easily operated and configured through a Serial Peripheral Interface (SPI). The register map, which is accessible through the SPI, contains all configuration registers and is accessible in all operation modes of the chip.

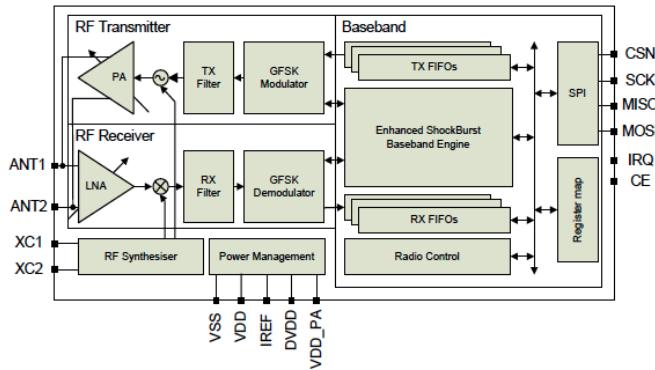


Figure 12.1.11 – nRF24L01 Block Diagram

The RF channel frequency determines the center of the channel used by the nRF24L01. The channel occupies a bandwidth of less than 1 MHz at 250 kbps, and 1Mbps and bandwidth of less than 2 MHz at 2 Mbps. The module can operate on frequencies from 2.4 GHz to 2.525 GHz.

The programming resolution of the RF channel frequency setting is 1 MHz. Since at 2 Mbps the channel occupies a bandwidth wider than the resolution of the RF channel frequency setting, to ensure non-overlapping channels in 2 Mbps mode, the channel spacing must be 2 MHz or more.

The embedded baseband protocol engine (Enhanced ShockBurst) is based on packet communication and supports various modes for manual operation to advanced autonomous protocol operation. Internal FIFO (First In First Out) ensure a smooth data flow between the radio front end and the system's MCU. Enhanced ShockBurst reduces the system cost by handling all the high-speed link layer operations.

The Enhanced ShockBurst features automatic packet transaction handling for the easy implementation of a reliable bi-directional data link. A packet transaction is a packet exchange between two transceivers, one acting as the Primary Receiver (PRX) and the other is always acting as the Primary Transmitter (PTX). An Enhanced ShockBurst packet transaction is always initiated by the packet transmission from the PTX, the transaction is complete when the PTX has received an acknowledgement packet (ACK packet) from the PRX. The PRX can attach user data to the ACK packet enabling a bi-directional data link.

The automatic packet transaction works as follows:

1. The transaction begins by transmitting a data packet from the PTX to the PRX. The Enhanced ShockBurst automatically sets the PTX in receive mode to wait for the ACK packet.
2. If the packet is received by the PRX, it assembles and transmits an acknowledgment packet to the PTX before returning to receive mode.
3. If the PTX does not receive the ACK packet immediately, it automatically retransmits the original data packet after a programmable delay and sets the PTX in receive mode to wait for the ACK packet.

The radio front end uses GFSK (Gaussian Frequency Shift Keying) modulation. It has user configurable parameters like frequency channel, output power and air data rate. nRF24L01 supports an air data rate of 250kbps, 1Mbps, and 2Mbps. The high air data rates combined with two power saving modes make the nRF24L01 very suitable for ultra low power designs.

It contains internal voltage regulators that ensure a high Power Supply Rejection Ratio (PSRR) and wide power supply range.

12.1.21 HC-05 Bluetooth Transceiver

The HC-05 module is an easy to use Bluetooth SPP (Serial Port Protocol) module, designed for transparent wireless serial connection setup. Serial port Bluetooth module is fully qualified Bluetooth V2.0+EDR (Enhanced Data Rate) 3Mbps Modulation with 2.4 GHz radio transceiver and baseband. It uses CSR Bluecore 04-External single chip Bluetooth system with CMOS technology and with AFH (Adaptive Frequency Hopping Feature). The default Baud rate is of 38,400.



Figure 12.1.12 – HC-05 Bluetooth Module

The HC 05 module has two modes: master and slave device, and these can be switched. The master role have no function to remember the last paired slave devices. It can be made paired to any slave device. If you want the HC 05 to remember the last paired slave device address, you need to set the module after paired with the other device.

The master device can not only make pair with the specified Bluetooth address, like cell-phone or a pc, but also can search and make pair with the slave device automatically. One some specific conditions, master device and slave device can make pair with each other automatically (this is the default method).

There is only point-to-point communication for modules, but the adapter can communicate with multi-modules.

It has a low power 1.8 to 3.6 V I/O, with a current consumption during the pairing that is fluctuant in the range of 30-40 mA. The mean current is about 25 mA. After pairing, no matter processing communication or not, the current is 8mA. There is no sleep mode.

12.1.22 XBee RF Module

The XBee RF Module was designed to mount into socket and therefore does not require any soldering when mounting it to a board. The XBee RF Module interface to a host device through a logic-level asynchronous serial port. Through its serial port, the module can communicate with any logic and voltage compatible UART (Universal Asynchronous Receiver/Transmitter), or through a level translator to any serial device (through a Digi proprietary RS-232 or USB interface board).

By default, XBee RF Modules operate in Transparent Mode. When operating in this mode, the modules act as a serial line replacement, all the UART data received through the DI pin is queued up for RF transmission. When RF data is received, the data is sent out the DO pin.



Figure 12.1.13 – XBee Module

XBee RF Modules are configured to operate within Peer-to-Peer network topology and therefore are not dependent upon Master/Slave relationships. This means that modules remain synchronized without use of master/server configurations and each module in the network shares both roles of master and slave. Digi's peer-to-peer architecture features fast synchronization times and fast cold start times. This default configuration accommodates a wide range of RF data applications.

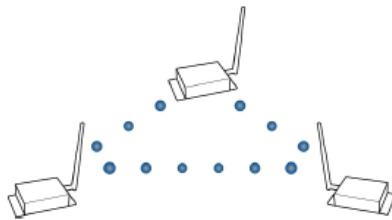


Figure 12.1.14 – XBee Peer-to-Peer Topology

A peer-to-peer network can be established by configuring each module to operate as an End Device ($CE = 0$), disabling End Device Association on all modules ($A1 = 0$) and setting ID and CH parameters to be identical across the network.

Every RF data packet sent contains a Source Address and Destination Address field in its header. The RF module conforms to the 802.15.4 (same as ZigBee) specification and supports both short 16-bit addresses and long 64-bit addresses. A unique 64-bit IEEE source address is assigned at the factory and can be read with the SL (Serial Number Low) and SH (Serial Number High) commands. Short addressing must be configured manually.

By default, the RF module operates in Unicast Mode. Unicast Mode is the only mode that supports retries. While in this mode, receiving modules send an ACK (acknowledgement) of RF packet reception to the transmitter. If the transmitting module does not receive the ACK, it will resend the packet up to three times or until the ACK is received.

XBee modules operate in five modes. When not receiving or transmitting data, the RF module is in Idle Mode. The module shifts into the other modes of operation under the following conditions:

- Transmit Mode – Serial data is received in the DI Buffer.
- Receive Mode – Valid RF data is received through the antenna.
- Command Mode – Command mode sequence is issued.
- Sleep Mode – Enable RF module to enter states of low-power consumption when not in use.

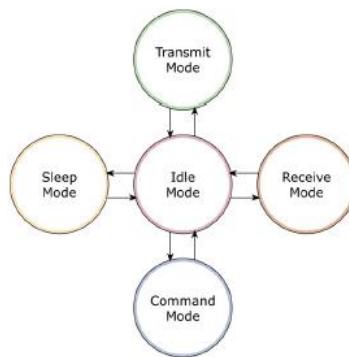


Figure 12.1.15 – XBee Operating Modes

12.1.23 Wireless Modules Comparison

The following table presents a detailed comparison between the specifications of the wireless modules.

	nRF24L01	HC 05 Bluetooth	XBee
Performance			
Outdoor Range	70m-100m @ 250kbps	30ft (10m)	300ft (90m)
RF Data Rate	250kbps, 1Mbps, and 2Mbps	Asynchronous: 2Mbps/160kbps Synchronous: 1Mbps/1Mbps	250kbps
Serial Interface Data Rate	0-10Mbps	1.2kbps – 115.2kbps	1.2kbps – 250kbps (non-standard baud rates also supported)
Transmit Power Output	0dBm	+4dBm	1mW (0dBm)
Receiver Sensitivity	-94dBm	-80dBm	-92dBm (1% packet error rate)

Power Requirements			
Supply Voltage	1.9V – 3.6V	1.8V – 3.6V	2.8V – 3.4V
Transmit Current	11.3mA	30mA	45mA @ 3.3V
Receive Current	13.5mA @ 2Mbps	30mA	50mA @ 3.3V
Idle Current	26uA	8mA	50mA @ 3.3V
Power-down Current	900nA	N/A	< 10uA
General			
Operating Freq.	2.4GHz	2.4GHz	2.4GHz
Operating Temp.	-20° to 70° C	-20° to 75° C	-40° to 85° C
Modulation	GFSK	GFSK	QPSK
Communication Mode	Enhanced ShockBurst	IEEE 802.15.1	IEEE 802.15.4

12.2 System Specifications

For the RMF Intelligent Home system the network solutions consists of a modem/router, router and switch connections. The modem/router is provided by the customer's Internet Service Provider (ISP). The switch allows for more Ethernet/Lan connections for more devices connected to the system. Adding a router to the mode/router of the home allows for the RMF Intelligent Home system to be completely pre-configurable and simply added to the home by mounting the correspondent systems. Instead of assigning static IP's to each subsystem, the router is assigned a static IP and a port range forwarding is established. On the subnet, subsystems connected to this router, the static IP's assigned are based on the subnet router IP address range. In the subnet router, a port forwarding is configured to redirect traffic to the corresponding subsystems.

12.3 Economic Analysis

Table 12.3.1: Materials/Equipment used

Product	Amount	Cost
Lynksys Router E1200	1	\$59.99
Ethernet Cables	10	\$7.99/each
Switch	1	\$39.99
Total		\$179.88

After careful analysis it has been determined that the RMF Intelligent Home system needs the materials listed in the table above for this particular demonstration. At a low cost of \$179.88 for networking purposes this solution is incomparable to any other because a low range router was utilized which implies lower cost and Ethernet cables are bought in quantities that qualify for discounts.

12.4 Results

After connecting all of the systems to the internet via the router and modem, the RMF Intelligent Home system can now be accessed with the use of the android application. This application utilizes the service internet to connect to the customer's unique IP address and into their homes through the modem and then to the system's subnet router. Each system contains a mac address and an IP address. These addresses are unique. The resulting network solution can be observed in the figure below.

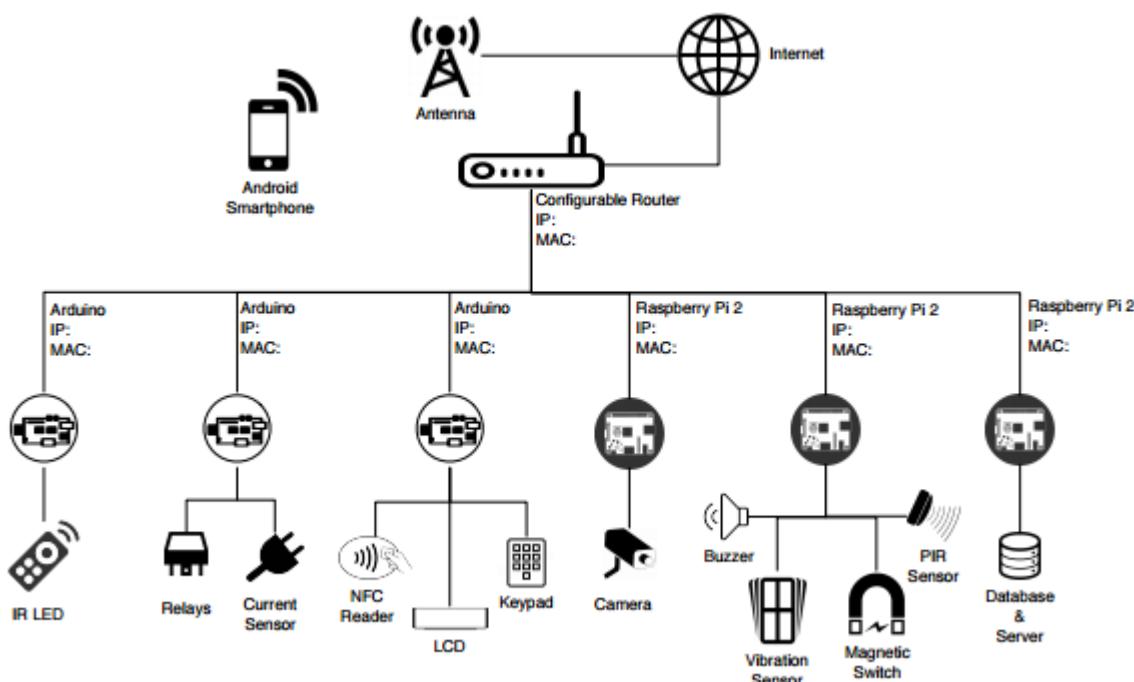


Figure 12.4.1: System Network Diagram

Chapter 13: Database

13.1 Alternatives Considered

13.1.1 Relational Database

A relational database is a collection of data items organized as a set of tables from which data can be accessed or reassembled in many different ways without having to reorganize the tables. Relational databases organize data into tables with unique keys for each row. The relational databases establish a relationship between their tables.

Each table is a representation of an entity or object that is in a tabular format consisting of columns and rows. Columns are the fields of a record or the attributes of an entity. The rows contain the values or data instances; these are also called records or tuples.

Relationships exist among the columns within a table and among the tables. The relationships take three logical forms:

One to One:

- It means that there are two tables that have a relationship, but that relationship only exists in such a way that any given row from Table A can have at most one matching row in Table B.

- A real world example of this could be the relationship between a person and its social security number.

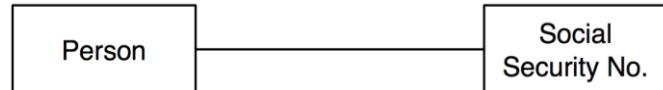


Figure 13.1.1 – One to One Relationship

One to Many:

- This relationship is used when a row in Table A can map different rows in Table B, but not the other way.
- A real world example could be the relationship between a car brand and the car model.

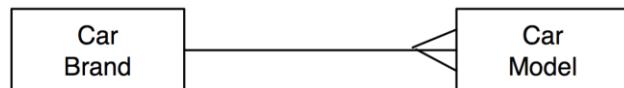


Figure 13.1.2 – One to Many Relationship

Many to Many:

- This relationship is used when the situation requires that a row in Table A can map to different rows in Table B, and those rows in Table B can also map different rows in Table A.
- A real world example of this could be books and authors.



Figure 13.1.3 – Many to Many Relationship

Virtually, all relational database systems use SQL (Structured Query Language) as the language for querying and maintaining the database. There are other relational database query languages, but most of these are variations of the SQL.

Relational databases scale well, but usually only when that scaling happens on a single server (“scale up”). When the capacity of that single server is reached and you need to “scale out” and distribute the capacity across multiple servers, the relational databases start to cause problems and become complex. The characteristics that make relational databases so appealing are the very same that also reduce their viability as platforms for large distributed systems.

Pros:

- Scale well while in a single server.
- Can enforce relationship between data items.
- Ensure and enforce business logic and rules at the database layer.

Cons:

- Each item can only contain one attribute.
- Lack of scalability when dealing with multiple servers (no horizontal scaling).

13.1.2 Non-Relational Database

A non-relational database provides a mechanism for storage and retrieval of data that is modeled in means other than tabular relations used in relational databases. These types of databases do not incorporate the table/key model that relational database management systems promote. These kinds of databases require data manipulation techniques and processes designed to provide solutions to big data problems that big companies face.

Some important aspects of these databases are the simplicity of design, scalability on different servers (horizontal scaling), finer control over availability, cost, and speed. A non-relational implementation can “scale out”, or distribute the database loads across various servers. Although they offer these advantages, they also suffer from being a relatively new and unproven technology, and they cannot provide RDBMS rich reporting and analytical functionality.

Pros:

- Allow the storage of several related items in one row in the same table.
- Easily serve large numbers of user requests.

Cons:

- Key-value stores cannot enforce relationships between data items.
- Although it solves high transaction loads, it is at the cost of data integrity and joins.

13.1.3 MySQL

MySQL is the most popular open source database system. Since it is open source, anyone can use it for a wide variety of tasks for free. If a platform does not include MySQL in the repository, it is easy to download.

It runs as a background process (or as a foreground process if you launch it from the command line) on the system. Like most database systems, it supports Structured Query Language (SQL). You can use SQL to create databases and objects, write or change data, and execute various commands for managing the server.

To issue these commands, you first need to connect to the database server. MySQL provides a client application that enables you to connect to and run commands on the server. The application is named mysql and is known as the mysql client.

The MySQL database system stores data via a mechanism of programmatic isolation called storage engine that is governed by the handler interface. The handler interface permits the use of interchangeable storage components in the MySQL server so that the parser, the optimizer and all manner of components can interact in storing data on disk using a common mechanism.

13.2 System Specifications

13.2.1 DBMS Components

Using microcontrollers like the Arduino to host sensors is an economical way to build the sensor network. But when you need a little more computational power than an Arduino can provide, a computer with more processing power can allow the use of different applications.

The Raspberry Pi is a small and inexpensive personal computer. Although it lacks the capacity for memory expansion and cannot accommodate on-board devices such as DVDs, it has everything a simple personal computer requires.



Figure 13.2.1 – Raspberry Pi 2 Board

Because the Raspberry Pi memory is very limited, an important consideration was to increase the memory of it, or find an external memory source to use for the data storage. With testing purposes, we decided to add a 16 Gb pen drive. The pen drive will serve to collect enough data to prove that the system is fully functional. Our choice of pen drive was a SanDisk Cruzer with 16 Gb of memory, as it is presented in Figure 13.2.2.



Figure 13.2.2 – SanDisk 16Gb Pen drive

13.2.2 MySQL & Python

The Raspberry Pi was designed for Python and most OS images include it. Oracle provides two components that are needed to use MySQL in Python: the MySQL Connector/Python and MySQL Utilities. MySQL Connector/Python enables Python programs to access MySQL databases, using an API that is compliant with the Python Database API Specifications.

First you need the Connector/Python database connector. Connector/Python runs on any platform where Python is installed. This is a pure Python implementation of a common database connector for MySQL. Next, you need to install MySQL Utilities. To use the utilities, you can simply execute them.

13.3 Economic Analysis

Component	Use	Price
Raspberry Pi 2	These device is used for the DBMS.	\$69.99 + shipping and handling
Pendrive	It is used to store the DB data.	\$5.99 + shipping and handling

13.4 Results

13.4.1 Drive Formatting & Partitioning

Before using the pen drive with a file system incompatible with the Raspberry Pi, it is necessary to partition and format the drive. Because the pen drive had a partition on it already, it is necessary to delete it. The Raspberry Pi OS may be able to read the format of the pen drive, but it is necessary to use the ext4 file system for optimal performance. The steps to follow are described below.

1. Open the list of the drives attached to the Raspberry Pi using the fdisk command.
2. Delete the existing partition.
3. Create a new partition.
4. Accept the defaults to use all the free space in the drive.
5. Verify the work with the p command to print the device partition table and metadata.

6. Finally, issue the w command to write the partition table.

External drives in Linux are connected with mount command and disconnected with umount command. Unlike with other operating systems, it is not a good idea to unplug the USB drive without un-mounting it first. Likewise, it is necessary to mount the drive before you can use it.

In order for the pen drive to work as the database without the need of mounting the drive every time, it is necessary to set up the operating system to mount the drive automatically on each boot. You can tell the operating system to mount the drive with a specific UUID to a specific mount point, in our case the mount point is /media/HDD.

If by any chance the pen drive is plugged in to another hub port, the device name might not be the same the next time you boot. So, it is necessary to use the UUID to determine which drive is the data drive, and allows you to having to keep the drive plugged in to a specific port.

To set up automatic drive mapping, you use a feature called static information about the file system (fstab). This consists of a file located in the /etc folder of the system. The only necessary change is to add a line to the file with the UUID of the pen drive, the mount point and the file system format. Now the pen drive is ready to build a MySQL database server.

13.4.2 Database Implementation

Not all the RMF Intelligent Home systems need to save the generated data into a database. Our database contains tables for those systems that generate or need relevant data for the user or their functionality. Specifically, it contains information from those systems that generate data that might be valuable in the future.

Systems like lighting, or infrared control may generate data that does not need to be saved, but other systems like the energy monitor and the alarm might generate data that could be valuable. Other systems that might need to retrieve important data are the verification system and the phone application. The last two require verification of the input data consistency against the records in the database.

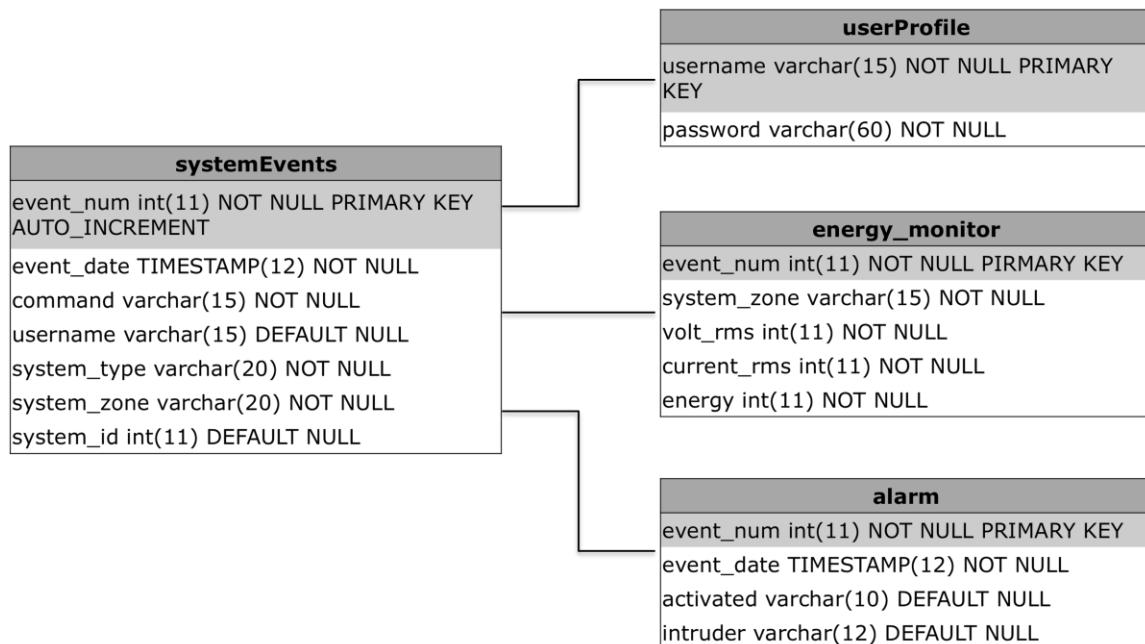


Figure 13.4.1 – Tables Relationship

As it can be seen from Figure 13.4.1, the tables created with a one to one relationship. Each time an event like intruder detection from the alarm or the periodical energy consumption data is stored, the system events table saves the event occurrence. So there is one event number for each of the times the alarm is activated or deactivated, as well as one event number for each of the energy monitor readings.

The same happens when a user (householder) tries to use the verification system to disarm or arm the alarm. The system events table saves the event occurrence, and verifies the user and its password against the records in the user profile table.

The following figures present how these tables described in Figure 13.4.1 are described inside of the database.

```
mysql> DESCRIBE system_events;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| event_num | int(11) | NO | PRI | NULL | auto_increment |
| event_date | date | NO | | NULL |
| command | varchar(15) | NO | | NULL |
| username | varchar(15) | YES | | NULL |
| system_type | varchar(20) | NO | | NULL |
| system_zone | varchar(20) | NO | | NULL |
| system_id | int(11) | YES | | NULL |
+-----+-----+-----+-----+-----+-----+
7 rows in set (0.00 sec)
```

Figure 13.4.2 – System Events Table

```
mysql> Describe userProfile;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| username | varchar(15) | NO | PRI | NULL | |
| password | varchar(60) | NO | | NULL | |
+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

Figure 13.4.3 – User Profile Table

Field	Type	Null	Key	Default	Extra
event_num	int(11)	NO	PRI	NULL	
event_date	timestamp	NO		CURRENT_TIMESTAMP	on update CURRENT_TIMESTAMP
activated	varchar(10)	YES		NULL	
intruder	varchar(12)	YES		NULL	

Figure 13.4.4 – Alarm System Table

Field	Type	Null	Key	Default	Extra
event_num	int(11)	NO	PRI	NULL	
system_zone	varchar(15)	NO		NULL	
volt_rms	int(11)	NO		NULL	
current_rms	int(11)	NO		NULL	
energy	int(11)	NO		NULL	

5 rows in set (0.01 sec)

Figure 13.4.5 – Energy Monitor Table

The alarm system and energy monitor tables will be programmed in the systems so they can insert the data according the code logic. For example, the alarm system will insert data whenever it is triggered or deactivated. This is important, because the user will only be allowed to read the data from these tables, as well as from the events table.

On the other hand, the user profile table will only be used for verification purposes, and the data in this table can only be accessed by the code itself, but will not be presented to anyone. This is done to provide a better level of security.

13.4.3 Database Management System

A database management system is software for creating and managing databases. Our DBMS provides the client with a systematic way to read or insert data to the database. Essentially

is serves as the interface between the database and end users or application programs, and it ensures that the data is consistently organized and remains easily accessible.

The DBMS limit what data the end user sees, as well as how that end user can view the data. It also provides physical and logical data independence, meaning that it can protect users and applications from needing to know where the data is stored or having to be concerned about changes to the physical structure of the data (storage and hardware).

RMF Intelligent Home DBMS uses a series of processes, which we call layers. These layers are presented in Figure 13.4.6.

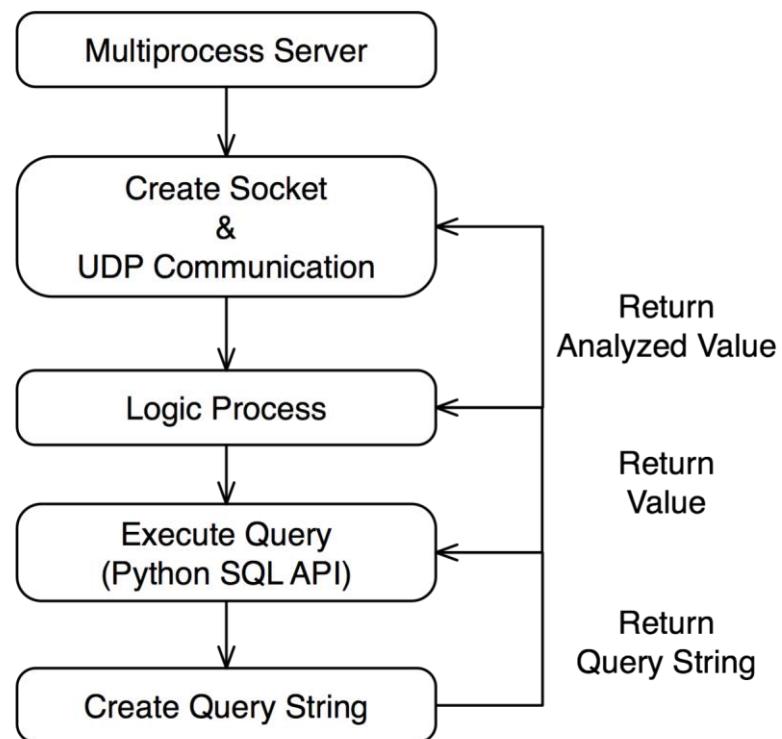


Figure 13.4.6 – DBMS Layers (Processes)

To explain these layers, let's start with the Multiprocess Server layer. This layer is called like that because it uses the Multiprocessing Python library to execute different sub-processes at

the same time. Because Raspberry Pi 2 CPUs is Quad Core, the Multiprocess library divides each of the sub-processes into a specific core.

In our case, these sub-processes are each of the connections that the Raspberry Pi does with the other systems (energy monitor, alarm, phone, and verification system) to obtain the pertinent data. This brings us to the next layer, the Socket and UDP Communication layer. In this layer is where the Raspberry Pi creates the communication socket and receives or sends a request to obtain the information using the User Datagram Protocol explained in the Networks chapter.

Each of the systems will communicate in its way and respond according to different commands, but all follow the same basic UDP process. The following Figures better illustrate the communication of the systems, and how these vary depending on which system is communicating with the DBMS.

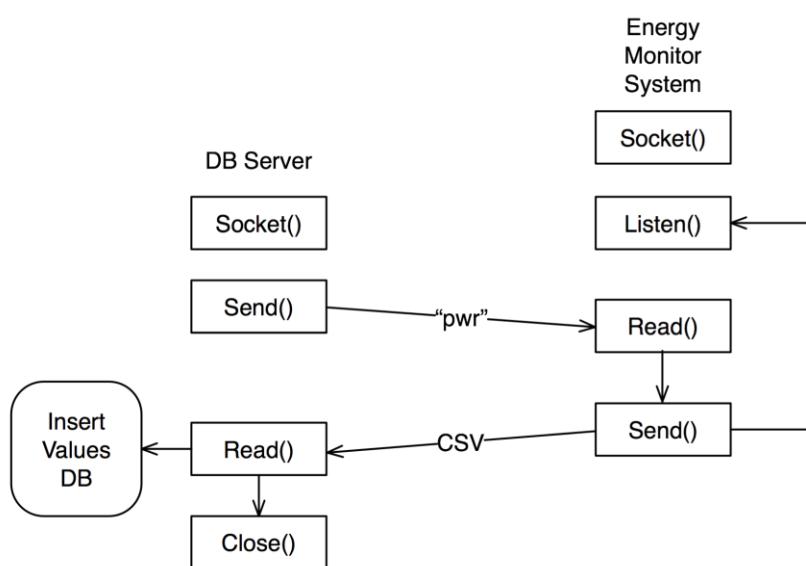


Figure 13.4.7 – DBMS and Energy Monitor Communication

The first one is the Energy Monitor to DBMS communication presented in Figure 13.4.7. As it can be seen, the Energy Monitor system will answer to the “pwr” command and send back a CSV with the data.

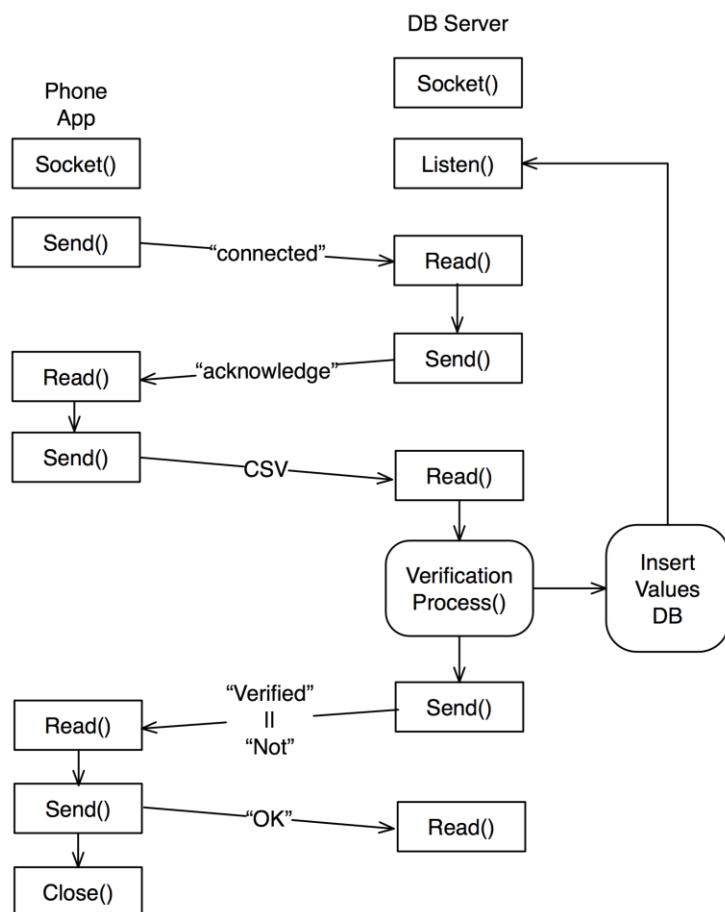


Figure 13.4.8 – DBMS and Phone App Communication (User Verification)

The Phone App Energy to DBMS communication presented in Figure 13.4.8 is the other way around, since the phone is the one who sends the first message. In this case it is the communication to verify the user credentials. After establishing communication, the phone sends

a CSV with the command and values. The DBMS interprets the CSV, verifies the username and password values, and sends back the verification response.

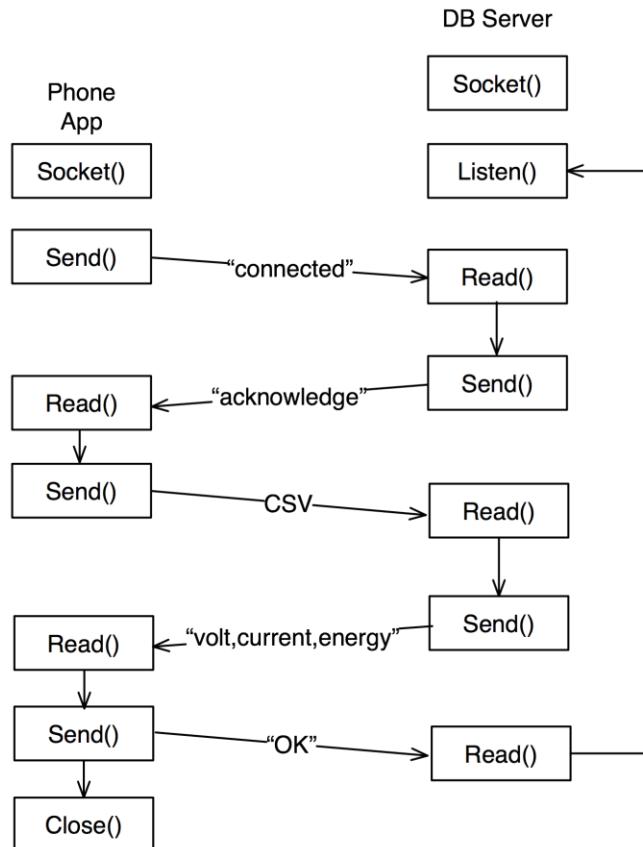


Figure 13.4.9 – DBMS and Phone App Communication (Energy Monitor)

In this case, the Phone App Energy to DBMS communication presented in Figure 13.4.9 is to request the energy monitor data from the database. After establishing communication, the phone sends a CSV with the command and values. The DBMS interprets the CSV and sends back another CSV with the requested information from the database.

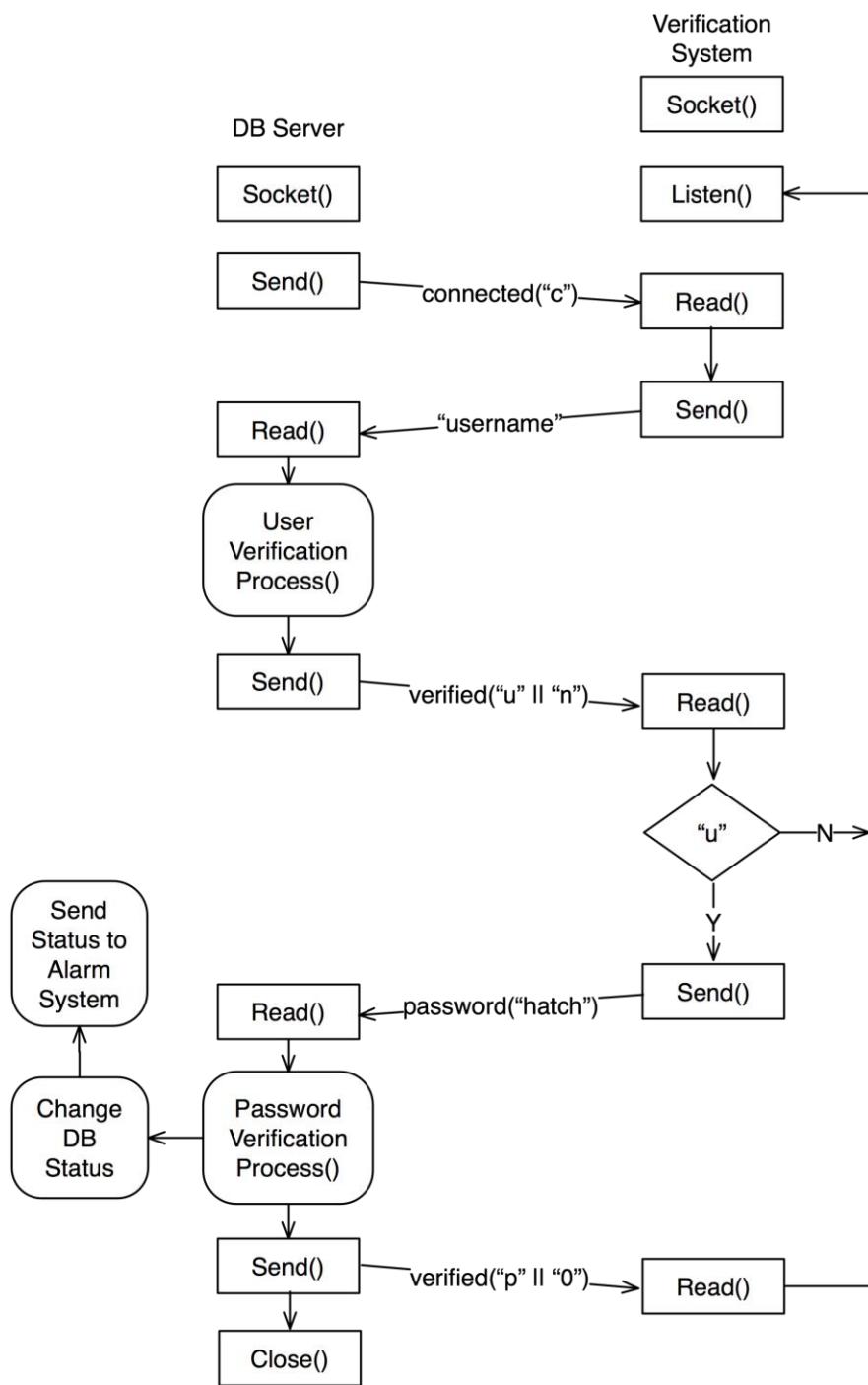


Figure 13.4.10 – DBMS and Verification System Communication

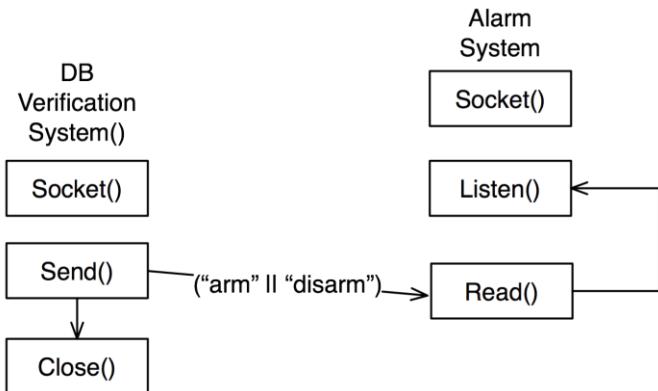


Figure 13.4.11 – DBMS and Alarm System Communication (Change Status)

The Verification system to DBMS communication presented in Figure 13.4.10 is very similar to the phone app user verification communication since the verification system sends the username and password so the DBMS can verify the credentials against the data in the database. The difference is that after the user is verified the DBMS also sends a message to the Alarm system to change the actual status.

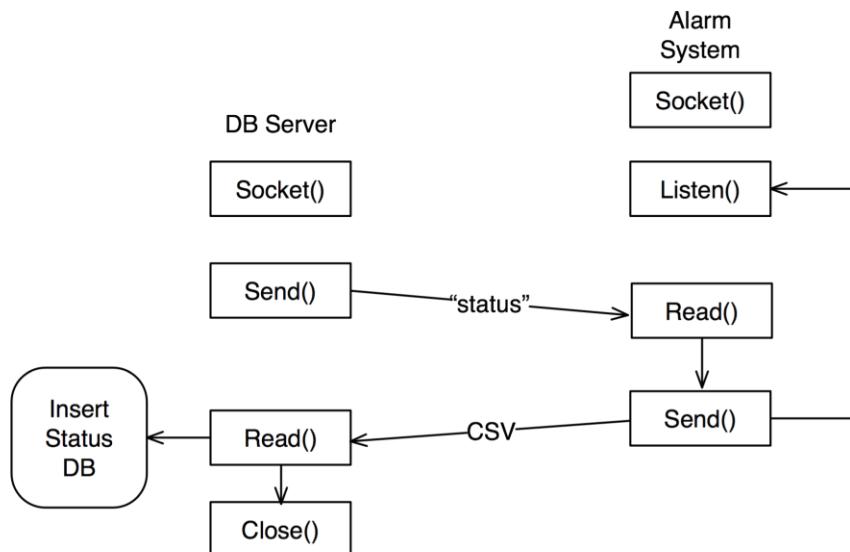


Figure 13.4.12 – DBMS and Alarm System Communication (Read Status)

The last diagram represents the communication between the Alarm system and the DBMS. In this case the communication is establish to know if the alarm status has changed. If any of the alarm variables has changed the DBMS saves the values to the database with the date and time information.

The next layer is the Logic Processes layer. This is the layer that analyses the data received from the systems communication and the data from the databases in order to give an appropriate response, or act accordingly. To visualize this, lets look at Figure 13.4.13, which represents the process to verify the user credentials.

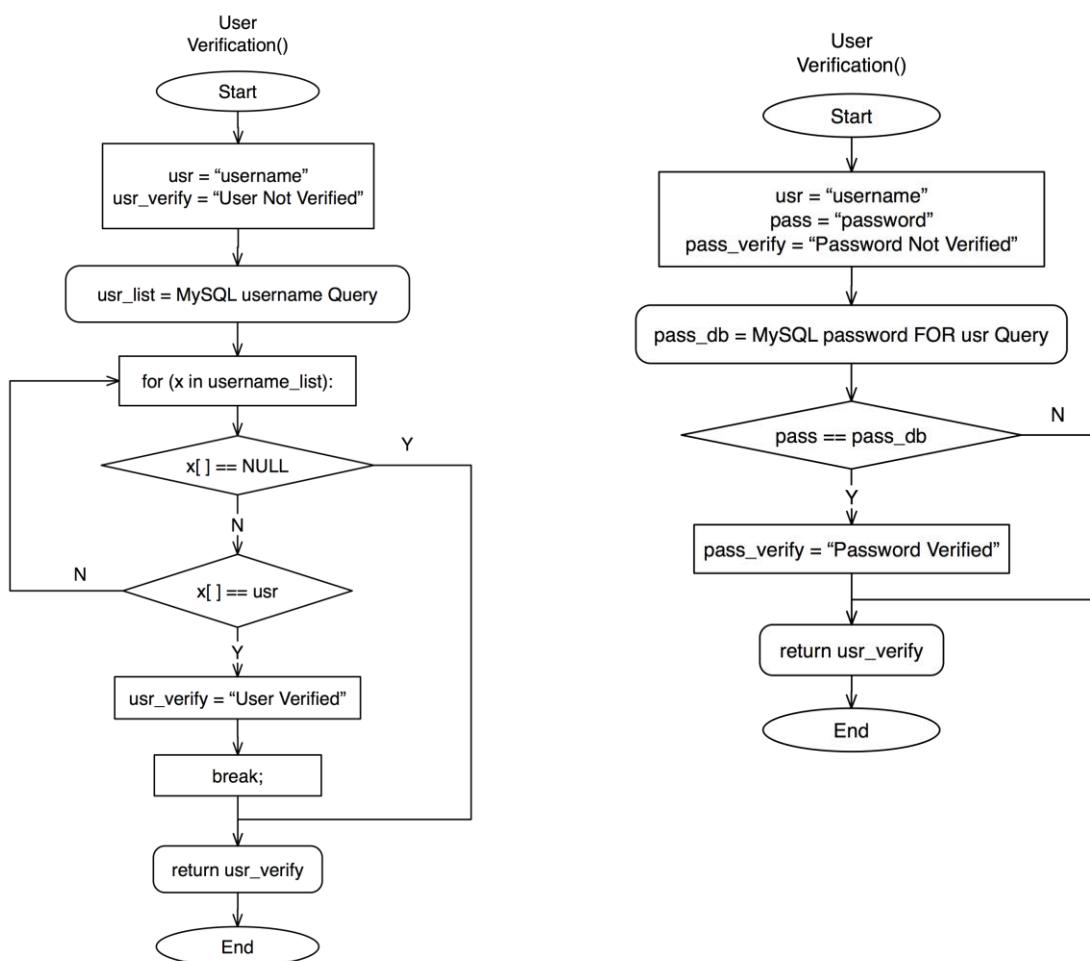


Figure 13.4.13 – User Verification Process

In Figure 13.4.13, first we verify the user against all of the users that exists in the database. If there is a match in the users, we use the username to find its password.

The last two layers are the Execute Query and Create Query String. These two work very close because the Execute Query is the one that receives all the information and uses this information to call the Create Query String. The Execute Query layer uses the query command and system name in order to call the Create Query String.

To better understand this lets imagine that we receive a request from the phone to retrieve the energy monitor data from the database. In this case the query command is a read (or SELECT) and the system is the energy monitor. Because we need to create the query to obtain the information from the database, we can use the known information and call the Create Query String layer to do the work for us. The resulting string returned from Create Query String layer could look like “SELECT * FROM energy_monitor”, which would be used by the Execute Query layer to connect to the database and return the requested values to the phone through the upper layers.

13.4.4 Database Management System Code

All the DBMS layers explained before were implemented in the Raspberry Pi using Python programming language. The layers were represented in classes, and each system was represented with its own class. The purpose of doing it this way was for scalability. If any change is needed to be done, it is easier when the code is divided into classes, rather than changing a whole bunch of script.

```

import time

import socket #for sockets

import sys #for exit

from ProcessEnergyMonitor import ProcessEnergyMonitor

class Comm_EnergySyst:

    def __init__(self, host, port, db_IPAddress, db_Name):

        self.host = host

        self.port = port

        self.db_IPAddress = db_IPAddress

        self.db_Name = db_Name

        self.tries = 0

        self.start = time.time()

    def executeAskStatusCommunication(self):

        socketCreated = False

        while 1:

            end = time.time()

            time_lapse = (end - self.start)

            if(time_lapse >= 5):

                self.start = time.time()

                try:

                    self.tries = self.tries + 1

                    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

                    s.settimeout(3.0)

                    print "socket Energy Monitor created"

                    socketCreated = True

                except socket.error:

                    print 'Failed to create Energy Status socket'

                    socketCreated = False

                try:

                    #-----Ask Energy Monitor PROCESS-----#
                    msg = "pw2"

                    s.sendto(msg, (self.host, self.port))


```

```

d = s.recvfrom(1024)
statusCSV = d[0]
addr = d[1]
#-----SQL CONNECT-----
energy_status = ProcessEnergyMonitor(self.db_IPAddress, self.db_Name,"energySyst", statusCSV)
energy_status.insertStatus()
except socket.error, msg:
    print "Error code Energy Status "

```

Figure 13.4.14 – DB & Energy Monitor Communication Code

```

import socket #for sockets
import sys #for exit
from ProcessAlarmStatus import ProcessAlarmStatus

class Comm_AlarmSyst:
    def __init__(self, host, port, db_IPAddress, db_Name):
        self.host = host
        self.port = port
        self.db_IPAddress = db_IPAddress
        self.db_Name = db_Name
        self.tries = 0
        self.activationStatus = ""
        self.intruderStatus = ""
        self.csv = ""

    def executeAskStatusCommunication(self):
        socketCreated = False
        while 1:
            try:
                self.tries = self.tries + 1
                s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

```

```

s.settimeout(3.0)

print "socket Alarm created"

socketCreated = True

except socket.error:

    print 'Failed to create Alarm Status socket'

    socketCreated = False

try:

    #-----Ask Alarm Status PROCESS-----#
    msg = "status"
    s.sendto(msg, (self.host, self.port))
    d = s.recvfrom(1024)
    statusCSV = d[0]
    addr = d[1]
    activationStatus = self.getActivationStatus(statusCSV)
    intruderStatus = self.getIntruderStatus(statusCSV)
    db_status = self.getDBAlarmStatus()

    if(activationStatus != db_status[0] or intruderStatus != db_status[1]):

        alarm_status = ProcessAlarmStatus(self.db_IPAddress, self.db_Name,"alarmSyst",statusCSV)
        alarm_status.insertStatus()

    except socket.error, msg:

        print "Error code Alarm Status"

def executeChangeStatusCommunication(self):

    socketCreated = False

    while (socketCreated == False and self.tries <= 3):

        try:

            self.tries = self.tries + 1
            s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
            s.settimeout(3.0)
            print "socket Alarm Change created"
            socketCreated = True

```

```

except socket.error:

    print 'Failed to create Alarm Change Status socket'

    socketCreated = False

try:

    #-----Change Alarm Status PROCESS-----#
    msg = "status"

    s.sendto(msg, (self.host, self.port))

    d = s.recvfrom(1024)

    csv = d[0]

    addr = d[1]

    status = self.getActivationStatus(csv)

    if (status == "armed"):

        msg = "disarm"

        s.sendto(msg, (self.host, self.port))

        d = s.recvfrom(1024)

        reply = d[0]

        addr = d[1]

        print "Reply: " + reply

    elif (status == "disarmed"):

        msg = "arm"

        s.sendto(msg, (self.host, self.port))

        d = s.recvfrom(1024)

        reply = d[0]

        addr = d[1]

        print "Reply: " + reply

except socket.error, msg:

    print "Error code Alarm Change Status"

    s.close()

def getDBAlarmStatus(self):

    alarm_status = ProcessAlarmStatus(self.db_IPAddress, self.db_Name,"alarmSyst", " ", "")

    status = alarm_status.readStatus()

```

```

return status

def getActivationStatus(self, csv):
    start = csv.index(",")
    activated = csv[0:start]
    return activated

def getIntruderStatus(self, csv):
    start = csv.index(",")
    values = csv[start+1:]
    return values

```

Figure 13.4.15 – DB & Alarm System Communication Code

```

import socket #for sockets

import sys #for exit

from ProcessVerification import ProcessVerification

from Comm_AlarmSyst import Comm_AlarmSyst


class Comm_VerificationSyst:

    def __init__(self, host, port, db_IPAddress, db_Name, ip_AlarmSystem, port_AlarmSystem):
        self.host = host
        self.port = port
        self.db_IPAddress = db_IPAddress
        self.db_Name = db_Name
        self.ip_AlarmSystem = ip_AlarmSystem
        self.port_AlarmSystem = port_AlarmSystem
        self.tries = 0

    def executeNFCCommunication(self):
        socketCreated = False
        while 1:
            try:

```

```

self.tries = self.tries + 1

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

##         s.settimeout(3.0)

print "socket NFC created"

socketCreated = True

except socket.error:

    print 'Failed to create NFC socket'

    socketCreated = False

try:

#-----VERIFICATION PROCESS-----

msg = "c"

s.sendto(msg, (self.host, self.port))

d = s.recvfrom(1024)

username = d[0]

addr = d[1]

print 'Server reply : ' + username

#-----SQL CONNECT-----

verification = ProcessVerification(self.db_IPAddress, self.db_Name, username)

db_user = verification.getUsername()

print db_user

if(db_user == ""):

    print "User Not Verified"

    msg = "n"

    s.sendto(msg, (self.host, self.port))

elif (db_user == username):

    print "User Verified"

    msg = "u"

    s.sendto(msg, (self.host, self.port))

d = s.recvfrom(1024)

```

```

password = d[0]

addr = d[1]

db_password = verification.getUserPassword()

print password

if(db_password != password):

    print "Password NOT Verified"

    msg = "0"

    s.sendto(msg, (self.host, self.port))

elif (db_password == password):

    print "Password Verified"

    msg = "p"

    s.sendto(msg, (self.host, self.port))

changeAlarm = Comm_AlarmSyst(self.ip_AlarmSystem, self.port_AlarmSystem, self.db_IPAddress,
self.db_Name)

changeAlarm.executeChangeStatusCommunication()

#-----END SQL CONNECT-----#

#-----END VERIFICATION PROCESS-----#


except socket.error, msg:

    print "Error code NFC "

    s.close()

```

Figure 13.4.16 – DB & Verification System Communication Code

```

##import MySQLdb

import socket #for sockets

import sys #for exit

from ProcessPhoneApp import ProcessPhoneApp

class Comm_PhoneApp:

    def __init__(self, localhost, localport, db_IPAddress, db_Name):

```

```

self.localhost = localhost
self.localport = localport
self.db_IPAddress = db_IPAddress
self.db_Name = db_Name
self.tries = 0
self.csv = ""

def executeCommunication(self):
    socketCreated = False
    while (socketCreated == False and self.tries <= 5):
        try:
            self.tries = self.tries + 1
            s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
            s.settimeout(3.0)
            s.bind((self.localhost, self.localport))
            print "socket created"
            socketCreated = True
        except socket.error:
            print 'Failed to create socket'
            socketCreated = False
        try:
#-----Ask Energy Monitor PROCESS-----
            d = s.recvfrom(1024)
            requestCSV = d[0]
            phoneAddrs = d[1]
#-----SQL CONNECT-----
            process = ProcessPhoneApp(phoneAddrs, self.db_IPAddress, self.db_Name, requestCSV)
            returnValue = process.executeProcess()

            s.sendto(returnValue, phoneAddrs)
        except socket.error, msg:

```

```
print "Error code"
```

Figure 13.4.17 – DB & Phone App Communication Code

```
from SQLQuery import SQLQuery

class ProcessEnergyMonitor:

    def __init__(self, ip_addrs, db_name, username, csv):
        self.ip_addrs = ip_addrs
        self.db_name = db_name
        self.username = username
        self.system = "energy_monitor"
        self.csv = csv

    def insertStatus(self):
        query = SQLQuery(self.ip_addrs, self.db_name, "insert", self.username, self.system, "1", "1", self.csv)
        value = query.executeQuery()
        return value

    def readStatus(self):
        query = SQLQuery(self.ip_addrs, self.db_name, "readAll", self.username, self.system, "1", "1", self.csv)
        value = query.executeQuery()
        return value
```

Figure 13.4.18 – Energy Monitor Process Code

```
from SQLQuery import SQLQuery

class ProcessAlarmStatus:

    def __init__(self, ip_addrs, db_name, username, csv):
        self.ip_addrs = ip_addrs
        self.db_name = db_name
```

```

self.username = username
self.system = "alarm"
self.csv = csv

def insertStatus(self):
    query = SQLQuery(self.ip_addrs, self.db_name, "insert", self.username, self.system, "1", "1", self.csv)
    value = query.executeQuery()
    return value

def readStatus(self):
    query = SQLQuery(self.ip_addrs, self.db_name, "readAlarmStatus", self.username, self.system, "1", "1",
self.csv)
    value = query.executeQuery()
    return value

```

Figure 13.4.19 – Alarm Status Process Code

```

from SQLQuery import SQLQuery

class ProcessVerification:
    def __init__(self, ip_addrs, db_name, username):
        self.ip_addrs = ip_addrs
        self.db_name = db_name
        self.username = username
        self.system = "userProfile"

    def getUsername(self):
        query = SQLQuery(self.ip_addrs, self.db_name, "readUsername", self.username, self.system, "1", "1", " ")
        value = query.executeQuery()
        return value

    def getUserPassword(self):

```

```

query = SQLQuery(self.ip_addrs, self.db_name, "readPassword", self.username, self.system, "1", "1", " ")

value = query.executeQuery()

return value


def getUserInfo(self):

    query = SQLQuery(self.ip_addrs, self.db_name, "readFromUser", self.username, self.system, "1", "1", " ")

    value = query.executeQuery()

    print value

    return value

```

Figure 13.4.20 – User Verification Process Code

```

from SQLQuery import SQLQuery

from ProcessVerification import ProcessVerification


class ProcessPhoneApp:

    def __init__(self, phoneAddrs, db_IPAddress, db_Name, csv):

        self.phoneAddrs = phoneAddrs

        self.db_IPAddress = db_IPAddress

        self.db_Name = db_Name

        self.csv = csv

        self.system = self.getSystem()

        self.username = self.getUsername()

    def executeProcess(self):

        status = ""

        if(self.system == "userProfile"):

            status = self.phoneUserVerification()

        elif(self.system == "energy_monitor"):

            status = self.getEnergyConsumption()

```

```

return status

def phoneUserVerification(self):

    password = self.getPassword()
    verificationStatus = ""

    process = ProcessVerification(self.db_IPAddress, self.db_Name, self.username)
    db_user = verification.getUsername()

    if(db_user == ""):
        print "User Not Verified"
        verificationStatus = "User Not Verified"

    elif (db_user == self.username):
        verificationStatus = "User Verified"
        db_password = verification.getUserPassword()

        if(db_password != password):
            print "Password NOT Verified"
            verificationStatus = "Password Not Verified"

        elif (db_password == password):
            print "Password Verified"
            verificationStatus = "Password Verified"

    return verificationStatus

def getEnergyConsumption(self):
    query = SQLQuery(self.db_IPAddress, self.db_name, "energyHistory", self.username, self.system, "1", "1", " ")
    energyConsumption = query.executeQuery()

    return energyConsumption

```

```

def getSystem(self):
    start = self.csv.index(",")
    value = self.csv[0:start]
    return value

def getUsername(self):
    start = self.csv.index(",")
    csv = self.csv[start+1:]
    start = csv.index(",")
    value = csv[0:start]
    return value

def getPassword(self):
    start = self.csv.index(",")
    csv = self.csv[start+1:]
    start = csv.index(",")
    value = csv[start+1:]
    return value

```

Figure 13.4.21 – Phone App Process Code

```

class TableSystemEvents:

    def __init__(self, event_date, command, username, system_type, system_zone, system_id):
        self.event_date = event_date
        self.command = command
        self.username = username
        self.system_type = system_type
        self.system_zone = system_zone
        self.system_id = system_id

    def insertCommand(self):
        insertCommand = "INSERT INTO system_events(event_date, command, username, system_type,

```

```

system_zone, system_id) VALUES(%s,%s,%s,%s,%s,%s)"

    return insertCommand


def insertValues(self):
    insertValues = (self.event_date, self.command, self.username, self.system_type, self.system_zone,
self.system_id)

    return insertValues


def readAllCommand(self):
    readAllCommand = "SELECT * FROM system_events"

    return readAllCommand


def updateCommand(self, update_num):
    updateCommand = "UPDATE system_events SET event_date=%s, command=%s, username=%s,
system_type=%s, system_zone=%s, system_id=%s WHERE event_num=%s"

    return updateCommand


def updateValues(Self, update_num):
    updateValues = (self.event_date, self.command, self.username, self.system_type, self.system_zone,
self.system_id, update_num)

    return updateValues


def getEventNumberCommand(self):
    queryCommand = "SELECT event_num FROM system_events WHERE event_date=%s"

    return queryCommand


def getEventInfoCommand(self):
    queryCommand = "SELECT event_num FROM system_events WHERE event_date=%s"

    return queryCommand


def getEventNumberValues(self):
    queryValues = self.event_date

```

```
return queryValues
```

Figure 13.4.22 – System Events Table Code

```
class TableEnergyMonitor:  
  
    def __init__(self, event_num, system_zone, csv):  
  
        self.csv = csv  
  
        self.event_num = event_num  
  
        self.system_zone = system_zone  
  
        self.volt_rms = self.getVoltRMS()  
  
        self.current_rms = self.getCurrentRMS()  
  
        self.energy = self.getEnergy()  
  
  
    def insertCommand(self):  
  
        energyMonitorInsertCommand = "INSERT INTO energy_monitor(event_num, system_zone, volt_rms,  
current_rms, energy) VALUES(%s,%s,%s,%s,%s)"  
  
        return energyMonitorInsertCommand  
  
  
    def insertValues(self):  
  
        energyMonitorInsertValues = (self.event_num, self.system_zone, self.volt_rms, self.current_rms, self.energy)  
  
        return energyMonitorInsertValues  
  
  
    def readAllCommand(self):  
  
        energyMonitorReadAllCommand = "SELECT * FROM energy_monitor"  
  
        return energyMonitorReadAllCommand  
  
  
    def readHistoryCommand(self):  
  
        energyMonitorReadCommand = "SELECT volt_rms,current_rms,energy FROM energy_monitor ORDER BY  
event_num DESC LIMIT 10"  
  
        return energyMonitorReadCommand  
  
  
    def updateCommand(self, update_num):
```

```

energyMonitorUpdateCommand = "UPDATE energy_monitor SET system_zone=%s, volt_rms=%s
current_rms=%s energy=%s WHERE event_num=%s"
return energyMonitorUpdateCommand

def updateValues(Self, update_num):
    energyMonitorUpdateValues = (self.system_zone, self.volt_rms, self.current_rms, self.energy, update_num)
    return energyMonitorUpdateValues

def getVoltRMS(self):
    start = self.csv.index(",")
    volt_rms = self.csv[0:start]
    self.csv = self.csv[start+1:]
    return volt_rms

def getCurrentRMS(self):
    start = self.csv.index(",")
    current_rms = self.csv[0:start]
    self.csv = self.csv[start+1:]
    return current_rms

def getEnergy(self):
    energy = self.csv[0:]
    return energy

```

Figure 13.4.23 –Energy Monitor Table Code

```

class TableAlarm:

def __init__(self, event_num, event_date, csv):
    self.csv = csv
    self.event_num = event_num
    self.event_date = event_date
    self.activated= self.getActivated()

```

```

self.intruder= self.getIntruder()

def insertCommand(self):
    insertCommand = "INSERT INTO alarm(event_num, event_date, activated, intruder) VALUES(%s,%s,%s,%s)"
    return insertCommand

def insertValues(self):
    insertValues = (self.event_num, self.event_date, self.activated, self.intruder)
    return insertValues

def readAllCommand(self):
    alarmReadAllCommand = "SELECT * FROM alarm"
    return alarmReadAllCommand

def readStatusCommand(self):
    alarmReadStatusCommand = "SELECT activated,intruder FROM alarm"
    return alarmReadStatusCommand

def getActivated(self):
    start = self.csv.index(",")
    activated = self.csv[0:start]
    self.csv = self.csv[start+1:]
    return activated

def getIntruder(self):
    intruder = self.csv[0:]
    return intruder

```

Figure 13.4.24 – Alarm System Table Code

```

class TableUserProfile:
    def __init__(self, csv):

```

```
self.csv = csv

self.username = " "
self.password = " "

def insertCommand(self):
    insertCommand = "INSERT INTO userProfile(username, password) VALUES(%s,%s)"
    return insertCommand

def insertValues(self):
    insertValues = (self.username, self.password)
    return insertValues

def readAllCommand(self):
    readAllCommand = "SELECT * FROM userProfile"
    return readAllCommand

def readAllUsers(self):
    readAllUsersCommand = "SELECT username FROM userProfile"
    return readAllUsersCommand

def readFromUserCommand(self, username):
    readAllCommand = "SELECT username,password FROM userProfile WHERE username=%s"
    return readAllCommand

def readFromUserValues(self, username):
    readUserValues = username
    return readUserValues

def readUserCommand(self, username):
    readUserCommand = "SELECT username FROM userProfile WHERE username=%s"
    return readUserCommand
```

```

def readUserValues(self, username):
    readUserValues = username
    return readUserValues

def readPasswordCommand(self, username):
    readPasswordCommand = "SELECT password FROM userProfile WHERE username=%s"
    return readPasswordCommand

def readPasswordValues(self, username):
    readPasswordValues = username
    return readPasswordValues

def updateCommand(self, username):
    updatePasswordCommand = "UPDATE userProfile SET password=%s WHERE username=%s"
    return updatePasswordCommand

def updateValues(Self, username):
    updateValues = (self.password,username)
    return updateValues

def getUsername(self):
    start = self.csv.index(",")
    username = self.csv[0:start]
    self.csv = self.csv[start+1:]
    return username

def getPassword(self):
    password = self.csv[0:]
    return password

```

Figure 13.4.25 – User Profile Table Code

```

import MySQLdb

import time

import datetime

from time import strftime

"""----Importing DB Tables Classes----"""

from TableSystemEvents import TableSystemEvents

from TableAlarm import TableAlarm

from TableEnergyMonitor import TableEnergyMonitor

from TableUserProfile import TableUserProfile


class SQLQuery:

    def __init__(self, ip_addrs, db_name, command, username, system, system_zone, system_id, values):

        self.user = "caps01user"

        self.passwd = "caps01pass"

        self.ip_addrs = ip_addrs

        self.db_name = db_name

        self.command = command

        self.username = username

        self.system = system

        self.system_zone = system_zone

        self.system_id = system_id

        self.values = values

        self.event_date = (time.strftime("%Y-%m-%d ") + time.strftime("%H:%M:%S"))

    def executeQuery(self):

        if(self.command != "readAlarmStatus"):

            eventTable = TableSystemEvents(self.event_date, self.command, self.username, self.system,

self.system_zone, self.system_id)

            self.insertIntoQuery(eventTable.insertCommand(),eventTable.insertValues())

            event_num = self.readNumberQuery(eventTable.getEventNumberCommand(),

eventTable.getEventNumberValues())

```

```

elif(self.command == "readAlarmStatus"):

    event_num = ""

if(self.system == "userProfile"):

    table = TableUserProfile(self.values)

elif(self.system == "energy_monitor"):

    table = TableEnergyMonitor(event_num, self.system_zone, self.values)

elif(self.system == "alarm"):

    table = TableAlarm(event_num, self.event_date, self.values)

if(self.command == "insert"):

    self.insertIntoQuery(table.insertCommand(), table.insertValues())

elif(self.command == "update"):

    if(self.system == "userProfile"):

        self.updateQuery(table.updatePasswordCommand(self.username),

table.updatePasswordValues(self.username))

elif(self.command == "readAll"):

    if(self.system != "userProfile"):

        self.readValue = self.readQuery(table.readAllCommand())

        return self.readValue

elif(self.command == "readPassword"):

    if(self.system == "userProfile"):

        self.readValue = self.readProfileQuery(table.readPasswordCommand(self.username),

table.readPasswordValues(self.username))

        return self.readValue

```

```

elif(self.command == "readUsername"):

    if(self.system == "userProfile"):

        self.readValue = self.readProfileQuery(table.readUserCommand(self.username),
table.readUserValues(self.username))

    return self.readValue


elif(self.command == "readAlarmStatus"):

    if(self.system == "alarm"):

        self.readValue = self.readQuery(table.readStatusCommand())

    return self.readValue


elif(self.command == "energyHistory"):

    if(self.system == "energy_monitor"):

        self.readValue = self.readAllQuery(table.readHistoryCommand())

    return self.readValue


def insertIntoQuery(self, queryCommand, queryValues):

    db = MySQLdb.connect(host=self.ip_addrs, user=self.user, passwd=self.passwd, db=self.db_name)

    cur = db.cursor()

    sql = (queryCommand,queryValues)

    try:

        print "Writing to database..."

        cur.execute(*sql)

        db.commit()

        print "Write complete"

    except:

        db.rollback()

        print "Failed writing to database"

    cur.close()

    db.close()

```

```

def readNumberQuery(self, queryCommand, queryValues):
    db = MySQLdb.connect(host=self.ip_addrs, user=self.user, passwd=self.passwd, db=self.db_name)
    cur = db.cursor()
    sql = (queryCommand, queryValues)
    try:
        print "Reading Event Num from database..."
        cur.execute(*sql)
        db.commit()

        for (event_num) in cur:
            for x in event_num:
                readValue = x
        print "Reading Event Num complete"
    except:
        db.rollback()
        print "Failed reading Event Num from database"
        readValue=""
    cur.close()
    db.close()
    return readValue

def readQuery(self, queryCommand):
    db = MySQLdb.connect(host=self.ip_addrs, user=self.user, passwd=self.passwd, db=self.db_name)
    cur = db.cursor()
    sql = (queryCommand)
    readValue = ""
    try:
        print "Reading from database..."
        cur.execute(sql)
        db.commit()
        for (value) in cur:
            readValue = value

```

```

print "Reading complete"

except:
    db.rollback()
    print "Failed reading from database"
    cur.close()
    db.close()

return readValue

def readAllQuery(self, queryCommand):
    db = MySQLdb.connect(host=self.ip_addrs, user=self.user, passwd=self.passwd, db=self.db_name)
    cur = db.cursor()
    sql = (queryCommand)
    readValue = ""

    try:
        print "Reading from database..."
        cur.execute(sql)
        db.commit()
        for (value) in cur:
            readValue = value
        print "Reading complete"
    except:
        db.rollback()
        print "Failed reading from database"
        cur.close()
        db.close()

    return readValue

def readProfileQuery(self, queryCommand, queryValues):
    db = MySQLdb.connect(host=self.ip_addrs, user=self.user, passwd=self.passwd, db=self.db_name)
    cur = db.cursor()
    sql = (queryCommand, queryValues)
    readValue = ""

```

```

try:
    print "Reading from database..."
    cur.execute(*sql)
    db.commit()
    for (username) in cur:
        for x in username:
            readValue = x
            print x
    print "Reading complete"
except:
    db.rollback()
    print "Failed reading from database"
    cur.close()
    db.close()
    return readValue

def updateQuery(self, queryCommand, queryValues):
    db = MySQLdb.connect(host=self.ip_addrs, user="caps01user", passwd="caps01pass", db=self.db_name)
    cur = db.cursor()
    sql = (queryCommand, queryValues)
    try:
        print "Updating database..."
        cur.execute(*sql)
        db.commit()
        print "Update completed"
    except:
        db.rollback()
        print "Failed updating database"
    cur.close()
    db.close()

```

Figure 13.4.26 – SQL Query Code

Chapter 14: Phone App

14.1 Alternatives Considered

For the RMF Intelligent Home system to be fully functional at the user's request, an application is designed taking into considerations the availability, reliability and functionality of the operating systems available in the market today, the programming languages that can be used to program the application, the available Integrated development environments for these specific operating systems, and the device hardware specifications to operate the application.

14.1.1 Operating Systems

An operating system supports the computer's basic functions. The operating system recognizes user input through peripherals and displays output in the monitor. Other tasks include managing files and directories. An operating system basically controls the flow of the processes/activities of the ongoing operations of programs preventing for any interference of any. In computers, operating systems can be Multi-user, Multiprocessing, Multitasking, Multithreading, Real time, etc. These different classifications simply allow for a software platform where applications can run.

14.1.2 Android Operating System

Google designs the Android operating system. It is based on Linux, which basically works as the I/O manager and translates data into processing instructions. This is known as a kernel. For Android, the libraries are written in C/C++. This allows for faster processing since the API mainly utilizes the Java programming language. Android is open source, which allows developers to upload their own application to the Google Play Store as long as the developer follows the basic development rules. Other Android users can then download this application. Since Android does not require any acceptance process from the Google Play store then it causes functional and security issues more likely to occur. Android includes many tools for application testing within the Android SDK. Some IDEs include Eclipse, which can be downloaded with the Android development tools package from the official developer website, and the Android Studio which is a new IDE being utilized also serves as testing. For our purposes, the newest version of KitKat, the KitKat 4.4 requires at least 512 MB of RAM.

14.1.3 iPhone/IPad Operating System (iOS)

iOS requires the application review process to take place before the developer uploads their application into the market, which causes the least amount of security and functionality occurrences. Xcode is the main IDE for developing in iOS, which includes its own debugging and testing tools with crash logs which are of very simple use. Installing a new app under testing is more difficult than that in Android. The instructions for this process are more tedious and are

sometimes error-prone. IOS is also not open source like android is which causes limitations to the environments where the developer can utilize. The necessary IDEs exist for the Mac OS to develop for Android and IOS meanwhile Windows only tolerates the development for Android, this is excluding the manipulation of utilizing virtual machines to run Mac OS in the Windows environment. The newest version of iOS, the iOS 8.2 has a minimum requirement of at least 1 GB of RAM.

14.1.3 Programming Languages

Programming languages are instructions utilized to communicate a machine or computer. Some languages are formal but others are not. The differences between the two languages are the capability of having logical processes which formal languages are specifically written for this purpose. Some examples of formal languages are:

- Java
- C++
- C#

Some examples of languages that are not formal are:

- HTML
- CSS

Most languages utilized to create web pages are the non-formal languages. Formal languages are powerful compared to the non-formal languages. Of course each language has their own capabilities but formal languages create processes that include logic manipulation based on events while languages such as Html describe the user interface or front-end visual aspects of a project. For the functionality of this project and the specific platform of Android the language utilized is Java.

14.1.4 Integrated Development Environment

An IDE is nothing more than a programming environment packaged as an application that has code editor, compiler, debugger and tools for graphical user interface building. Visual studio, Eclipse, Netbeans and Android Studio are some of the existing IDEs. Each IDE is based on characteristics and purposes. Some are designed for C/C++, Java, Python, Perl, etc., depending on the intended use. Android studio of course is designed for developing specifically in Java for the Android platforms.

In the Android Studio environment a combination of XML description layouts are utilized for the visual of each user interface within applications. These interfaces are then connected to the program logic of the Java code programmed in class members. This allows for fast interaction between the interfaces.

14.2 System Specifications

After careful considerations, the RMF Intelligent Home system application is based on the Android operating system. This decision is mostly based on the availability this particular has with open source. Android also has an integrated development environment optimized called Android studio. The application provides a solution to all of the system requirements.

14.2.1 System Requirements

The RMFIIntelli App must present the user with options to utilize the full benefits and utilities of the system. Such options include, but are not limited to, the following:

- Profile login
- Manual Control of Automated components
- Arm system
- Video live stream
- System Notifications
- System Status

The RMFIIntelli App meets all of these system requirements by providing the user with a client end hardware interface simple enough for any user to utilize. The application also provides data analysis on power consumption to help the user economize by having a more efficient home.

14.2.2 The Application and the Android Operating System

The application is targeted for android 4.4, and earlier versions of Kit-Kat, but is compatible with android 5.0 and later versions of this Lollipop operating system (5.0-5.x). The application logic is based on menus that provide an ideal graphical user interface for easy control of the home automation system. Each interface has a class, which contains the logical code that includes, but not limited to, socket programming logic, interface manipulation and updates, and various system operations. The Integrated Development Environment (IDE) utilized for this development is called Android Studio. In the following sections, actual interfaces and runtime examples will be shown.

14.2.3 Java

Android studio utilizes the Java programming language and therefore for this particular system this programming language is targeted as a must know. This language is also utilized, not only for Android purposes, based on our programming knowledge at the moment of this system and application realization. Java is a general-purpose computer programming language. This is a formal language and it is object oriented meaning it utilizes classes as the main way of programming. Java allows for WORA, which means write once, run anywhere. This allows for any development to be portable and run anywhere on any device that can support Java. Java appeared in 1995 and it is a multi-platform language. It utilizes an emulator to generate the executable code, which makes it slower than C++ for example but does not mean it's not powerful or reliable. Java began with the JDK 1.0 and has been developed to Java SE 8 today.

Java generates a byte code, which is the intermediate representation of the language code. A Java Runtime Environment is installed on machines for standalone Java applications or web applets. All of these file extension generations require performance and therefore, as previously mentioned, Java programs have the reputation for being slower and requiring more memory than those programs in C++.

14.2.4 Android Studio

The version used to develop the application is 1.0.2. Android Studio is optimized for android application development based on java and XML. The easy installation and configuration of this IDE, compared to the Eclipse IDE, is far less difficult since Android Studio can be downloaded as an installation package executable. It facilitates icon generation, drag and drop method for interface development, intellisense to facilitate programming by bringing up a list of methods the user might be trying to implement and it is also very organized specifically to android development by including great debugging tools and log management to track troubleshooting throughout any trouble shooting or self-generated “breaking points” as code.

14.2.5 System Interfacing

The application controls the corresponding microcontrollers within the RMF Intelligent home system. Each particular system (lighting, camera, etc.) is manipulated with one microcontroller being a raspberry pi or an Arduino. Each one has an IP address and complies

with the Internet of things concept. Using socket programming each one of these controllers is reached and manipulated according to the user's intent in the application.

14.3 Economic Analysis

Android devices are less expensive to the user than apple devices. This cost is relative to the user. The cost for development and time for application publication for apple is much more. This is also why the application was intended for Android users since it is a relative cost for the system.

14.4 Results

The RMF Intelli App is targeted for Android 4.4 and earlier versions of Kit-Kat, but is compatible with Android 5.0 and later versions of the Lollipop operating system. The application logic is based on menus that provide an ideal graphical user interface for easy control of the RMF Intelligent Home system. The sets of interfaces along with the programming logic developed provide the user with the ideal tool to control and monitor their home.

14.4.1 Application Interfaces

Each interface has a specific use but they all serve the same purpose of facilitating the user with a simple and easy way of controlling and monitoring their home from the palm of their hands. The user will navigate through the interfaces as if it were their home and manipulating the states of the home with the help of these interfaces that interact with the actual hardware of the system. The following subsections will show an example of a simulation and how each standard area interface looks. The design and feel of these interfaces were done carefully and a study of tendencies was done to get feedback of the difficulty level, if any, of navigation and comprehension of the application.

14.4.2 Living Room Example

For this specific simulation the living room was chosen. In the living room the system includes a security camera, media, temperature control, and lighting control. The following figures demonstrate the use of the functionalities of the living room interface.

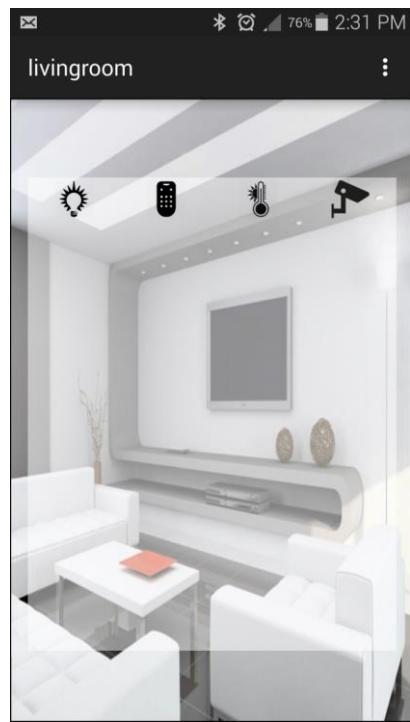


Figure 14.4.1 – Living Room



Figure 14.4.2 – Living Room Lighting

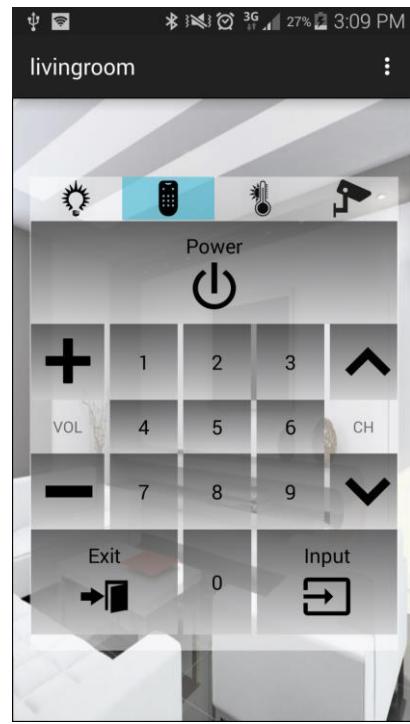


Figure 14.4.3 – Living Room Media

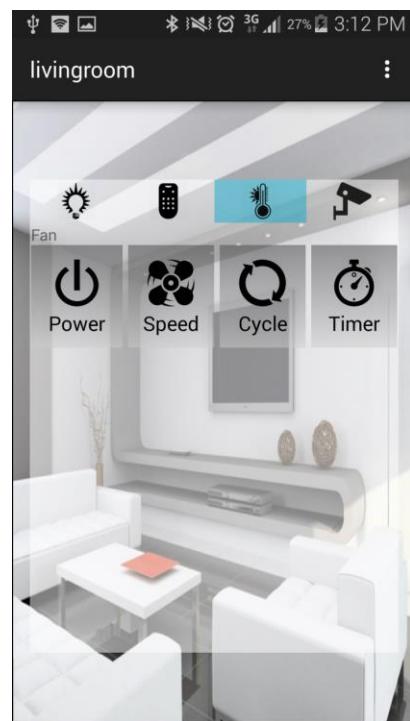


Figure 14.4.4 – Living Room Temperature

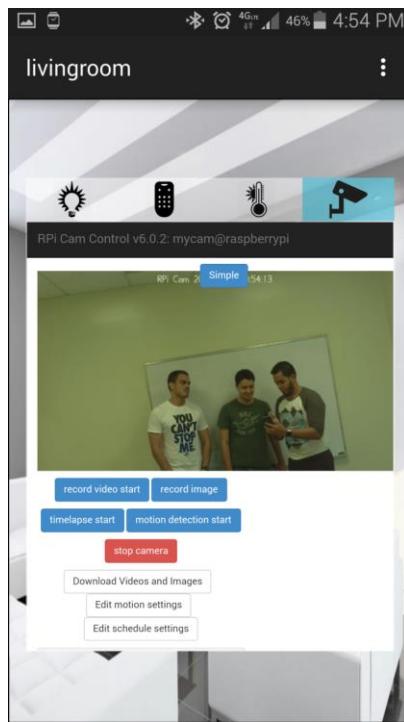


Figure 14.4.5 – Living Room Camera

The living room interface is a great example of the standard visual design of each interface. In other words, the XML code written for each graphical user interface (GUI) is very similar to each standard area specific interface. The logic of each are will vary depending on the user's requests' and configurations. This interface allows the user to monitor the activity happening in the living room by switching to the camera option in the tab menu of the interface. This uses the socket programming logic implemented in each class associated with that specific interface, in this case the living room interface. For the other options the same socket programming logic is implemented except, the message included in the network packet will be based on the attribute the user wishes to alter. When using the tab navigation, the application is

actually sending out a network packet with an update message, encrypted of course, to modify the user interface and give the actual state of the specific area. The media option simply communicates with the corresponding micro-controller by sending the user selection and the actual manipulation is done by the micro-controller logic. This was possible by means of the socket programming capabilities of Android and networking systems. Each interface of the application has the ability to communicate to any subsystem in the network.

14.4.3 Security System Interface

The security interface shows the state of each sensor. This interface allows for monitoring of the different type of sensors that are connected to the system at hardware level. The triggered sensor will be highlighted in red while the armed sensors will be in green. The un-armed sensors will be in a gradient white. This color-coding facilitates the user to quickly understand where the intrusion at the home is happening. This interface is associated with its class logic and executes the logic every time a sensor is armed, disarmed or triggered. Like the other interfaces this logic communicates through packets to the alarm system handler micro-controller.

14.4.4 Power Consumption Interface

The power consumption interfaces shows the user a report of how much power is being used by the home. The class logic of this interface also communicates by packets through the network to gather the current data stored in the power consumption monitor microcontroller. This

class logic has a loop effect to constantly refresh the data on screen as long as the user is still on this particular interface.

14.4.5 Application Class Logic

The application has a standard logic to each of the interfaces. These include, but not limited to, socket programming and system interfacing. In the following subsections some sample code is provided and the standard class logic is explained in more detail. Each micro-controller has a static IP address and is hard coded into the application code. The first few things that are set in the application are the listeners to detect user input. Each listener has other listeners within him or her to allow for a dynamic layout, which provides the user with a tab navigation interface. Once the user selects the desired option from the tab navigation menu, the other listeners begin to work and, along with the listening, update logic executes to show the current state of that selected area. This is then done by using a client class to send the packet to the micro-controller assigned to the particular area selected by the user and returning values of the current states of the area. The current states are then shown to the user via the GUI. The same process occurs when the user decided to alter these states with the difference of simply sending a message of action to the corresponding micro-controller. As this happens the interface continues to change with the new system states. For further algorithmic comprehension/analysis refer to the following subsection. The following code snippet is part of the application code designed to meet all the system requirement specifications for the specific area, in this case the living room.

```
package com.rmfintellihomes.rmfintellihomes;

import android.content.Intent;
import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.KeyEvent;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
```

```
*****  
//This is the login activity. It simply requires the user to enter  
//credentials and then contacts the server for confirmation. It first  
//detects whether the user has placed credentials or admin/local credentials  
//and processes the logic accordingly.  
*****  
  
public class login extends ActionBarActivity {  
  
    private Button login;  
    String message;  
    String received;  
    TextView welcome;  
    EditText usertext;  
    EditText pwd;  
    int tries;  
  
    private static final String TAG = "com.rmfintellihomes.rmfintellihomes";  
  
    @Override
```

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_login);  
  
    Intent i= new Intent(this, MyService.class);  
    startService(i);  
  
    login = (Button) findViewById(R.id.loginbutton);  
  
    welcome = (TextView) findViewById(R.id.notWelcome);  
    usertext = (EditText) findViewById((R.id.username));  
    pwd = (EditText) findViewById(R.id.password);  
    login.setOnClickListener(new View.OnClickListener() {  
        @Override  
        public void onClick(View v) {
```

```
if(usertext.getText().toString().matches("")||pwd.getText().toString().matches("")){  
  
    welcome.setText("Error: Please Enter a Valid Username and Password");  
  
}  
  
else if(usertext.getText().toString().equals("a")&&pwd.getText().toString().equals("a")){  
  
    Intent intent = new Intent(login.this, mainMenuActivity.class);  
  
    intent.putExtra("user", usertext.getText().toString());  
  
    login.this.startActivity(intent);  
  
    finish();  
  
    Log.i(TAG,"else if");  
  
}  
  
else{  
  
    message="userProfile,"+usertext.getText().toString()+","+pwd.getText().toString();  
  
    new Thread(new Client()).start();  
  
}
```

```
    }

});

}

@Override

public boolean onKeyDown(int keyCode, KeyEvent event){

    if((keyCode== KeyEvent.KEYCODE_BACK)){

        finish();

    }

    return super.onKeyDown(keyCode, event);

}

@Override

public boolean onCreateOptionsMenu(android.view.Menu menu) {

    // Inflate the menu; this adds items to the action bar if it is present.

    getMenuInflater().inflate(R.menu.menu_login, menu);

    return true;

}
```

```
@Override  
public boolean onOptionsItemSelected(MenuItem item) {  
    // Handle action bar item clicks here. The action bar will  
    // automatically handle clicks on the Home/Up button, so long  
    // as you specify a parent activity in AndroidManifest.xml.  
  
    int id = item.getItemId();  
  
    //noinspection SimplifiableIfStatement  
    if (id == R.id.action_settings) {  
        return true;  
    }  
  
    return super.onOptionsItemSelected(item);  
}
```

```
public class Client implements Runnable {  
    @Override  
    public void run() {  
        try {  
  
            //*****  
            //Lighting interface communication  
            //*****  
  
            tries=0;  
            do {  
                byte[] receiveData1 = new byte[1024];  
                // send message to Arduino/Pi  
                InetAddress serverAddr = InetAddress.getByName("70.45.26.180");  
                byte[] sendData = new byte[1024];
```



```
//runs when searching for the current status of lights, etc.

runOnUiThread(new Runnable() {

    @Override

    public void run() {

        //updates ui

        if (received.charAt(0) == 'y' && received.charAt(1) == 'y') {

            Intent intent = new Intent(login.this, mainMenuActivity.class);

            intent.putExtra("user", usertext.getText().toString());

            login.this.startActivity(intent);

            finish();

        }

        else if(received.charAt(0) == 'y' && received.charAt(1) == 'n'){

            welcome.setText("Error: Please Enter a Valid Password");

        }

    }

}
```

```
    }

    else

        welcome.setText("Error: Please Enter a Valid Username and Password");

    }

});//end runnable

if(received.charAt(0) == 'y' || received.charAt(1) == 'y'||received.charAt(0) == 'n' ||

received.charAt(1) == 'n')

    break;

tries++;

}while(tries<50);

Log.i(TAG," Login "+received);
```

```
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```

```
}

}

package com.rmfintellihomes.rmfintellihomes;

//*****  
//library importing for necessary classes usage  
//*****  
  
import android.content.Intent;  
import android.support.v7.app.ActionBar;  
import android.support.v7.app.ActionBarActivity;  
import android.os.Bundle;  
import android.view.KeyEvent;  
import android.view.Menu;  
import android.view.MenuItem;  
import android.view.View;  
import android.widget.Button;  
import android.widget.TextView;
```

```
import org.w3c.dom.Text;

public class mainMenuActivity extends ActionBarActivity {

    //*****
    //Button and interface association and ui logic. Each button is
    //assigned to an onlick listener that reacts to the user's command.
    //The user string will hold the user's username in order to identify
    //the user who manipulated the system and stored in the database.
    //*****

    Button kitchen, Master, bathroom, living, dining, garage, terrace, kidsroom,
    alarm,powermonitor;

    String user;
```

```
*****  
//OnCreate method associates each ui element described in the xml code  
//with its corresponding logic.  
*****  
  
@Override  
  
protected void onCreate(Bundle savedInstanceState) {  
  
    super.onCreate(savedInstanceState);  
  
    setContentView(R.layout.activity_main_menu);  
  
  
  
  
    Bundle info = getIntent().getExtras();  
  
    user = info.getString("user");  
  
  
  
  
    getSupportActionBar().setTitle(user);
```

```
kitchen = (Button) findViewById(R.id.kitchen);
kitchen.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(mainMenuActivity.this, KitchenInterface.class);

        mainMenuActivity.this.startActivity(intent);
    }
});

Master = (Button) findViewById(R.id.master);
Master.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(mainMenuActivity.this, MasterBedroom.class);
    }
});
```

```
mainMenuActivity.this.startActivity(intent);

}

});

bathroom = (Button) findViewById(R.id.bath);
bathroom.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

Intent intent = new Intent(mainMenuActivity.this, bathroom.class);

mainMenuActivity.this.startActivity(intent);

}

});
```

```
living = (Button) findViewById(R.id.livingRoom);
living.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(mainMenuActivity.this, livingroom.class);
        mainMenuActivity.this.startActivity(intent);
    }
});
```

```
dining = (Button) findViewById(R.id.diningroom);
dining.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(mainMenuActivity.this, diningroom.class);
        mainMenuActivity.this.startActivity(intent);
    }
});
```

```
garage = (Button) findViewById(R.id.garage);
garage.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(mainMenuActivity.this, garage.class);
        mainMenuActivity.this.startActivity(intent);
    }
});
```

```
terrace = (Button) findViewById(R.id.terrace);
terrace.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(mainMenuActivity.this, terrace.class);
        mainMenuActivity.this.startActivity(intent);
    }
});
```

```
kidsroom = (Button) findViewById(R.id.kidsroom);

kidsroom.setOnClickListener(new View.OnClickListener() {

    @Override

    public void onClick(View v) {

        Intent intent = new Intent(mainMenuActivity.this, kidsroom.class);

        mainMenuActivity.this.startActivity(intent);

    }

});
```



```
alarm = (Button) findViewById(R.id.alarm);

alarm.setOnClickListener(new View.OnClickListener() {

    @Override

    public void onClick(View v) {

        Intent intent = new Intent(mainMenuActivity.this, alarmsystem.class);

        mainMenuActivity.this.startActivity(intent);

    }

});
```

```
powermonitor = (Button) findViewById(R.id.consumption);

powermonitor.setOnClickListener(new View.OnClickListener() {

    @Override

    public void onClick(View v) {

        Intent intent = new Intent(mainMenuActivity.this, powerConsumption.class);

        intent.putExtra("user", user.toString());

        mainMenuActivity.this.startActivity(intent);

    }

});
```

```
}

//*****
//This method sets the username above the interface for easy id of
//who is logged in the account.
//*****



@Override
public boolean onCreateOptionsMenu(android.view.Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.menu_main_menu, menu);
    return true;
}
```

```
@Override  
public boolean onOptionsItemSelected(MenuItem item) {  
    // Handle action bar item clicks here. The action bar will  
    // automatically handle clicks on the Home/Up button, so long  
    // as you specify a parent activity in AndroidManifest.xml.  
  
    int id = item.getItemId();  
  
    //noinspection SimplifiableIfStatement  
    if (id == R.id.action_settings) {  
        return true;  
    }  
  
    return super.onOptionsItemSelected(item);  
}  
  
//*****
```

```
//on back button pressed the activity will be ended in order to prevent  
//the user to access the login screen again for security reasons. This  
//will protect the system application from being accessed by unwanted  
//individuals in case of the device being lost.  
//*****
```

```
@Override
```

```
public boolean onKeyDown(int keyCode, KeyEvent event)
```

```
{
```

```
    if ((keyCode == KeyEvent.KEYCODE_BACK))
```

```
{
```

```
        finish();
```

```
}
```

```
    return super.onKeyDown(keyCode, event);
```

```
}
```

```
}
```

```
package com.rmfintellihomes.rmfintelliomes;
```

```
import android.app.NotificationManager;  
import android.app.PendingIntent;  
import android.app.Service;  
import android.content.Intent;  
import android.media.RingtoneManager;  
import android.net.Uri;  
import android.os.AsyncTask;  
import android.os.IBinder;  
import android.support.v4.app.NotificationCompat;  
import android.util.Log;  
  
import java.net.DatagramPacket;  
import java.net.DatagramSocket;  
import java.net.InetAddress;  
  
//*****  
//This service utilizes client association with database in order  
//to determine the status of the alarm system. The user will not be  
//aware of this event occurring in the background. It uses asynchronounous
```

```
//operations to run on the thread stack in order to process multithreading  
//in which case allows for the user to utilize the application without  
//interruption. The variables declared in this service are used to  
//interact with the system and manipulate the service logic. The service  
//id (uniqueID) is necessary in order to prevent any service duplication  
//or conflict when manipulating push notifications or simply restarting  
//the service by using the "RETURN STICKY" statement.  
//*****
```

```
public class MyService extends Service {  
  
    private static final String TAG = "com.rmfintellihomes.rmfintellihomes";  
  
    boolean running;  
  
    NotificationCompat.Builder notification;  
  
    private static final int uniqueID = 73331;  
  
    String received="";  
  
    InetAddress serverAddr;  
  
    DatagramSocket clientSocket;  
  
    byte[] sendData;  
  
    String message = "intruder";//phony string to access remote alarm  
  
    DatagramPacket sendPacket;  
  
    byte[] receiveData1;
```



```
notification.setTicker("Alarm Triggered!");

notification.setWhen(System.currentTimeMillis());

notification.setContentTitle("Alarm");

notification.setContentText("Access your application to monitor your home now!");

Intent alarmaccess = new Intent(this, alarmsystem.class);

PendingIntent pending = PendingIntent.getActivity(this, 0, alarmaccess,

PendingIntent.FLAG_UPDATE_CURRENT);

notification.setContentIntent(pending);

NotificationManager m = (NotificationManager)

getSystemService(NOTIFICATION_SERVICE);

m.notify(uniqueID, notification.build());

}

@Override

public int onStartCommand(Intent intent, int flags, int startId)
```

```
{  
    super.onStartCommand(intent,flags,startId);  
  
    if(!running) {  
  
        Log.i(TAG,"onStartCommand");  
  
        r = new Runnable() {  
  
            @Override  
            public void run()  
            {  
                running = true;  
                for (; ; )  
                {  
                    futureTime = System.currentTimeMillis() + 5000;  
                    while (System.currentTimeMillis() < futureTime) {  
                        synchronized (this) {  
  
                            try {  
                                wait();  
                            } catch (InterruptedException e) {  
                                e.printStackTrace();  
                            }  
                        }  
                    }  
                }  
            }  
        };  
        r.run();  
    }  
}  
}
```

```
try {  
    wait(futureTime - System.currentTimeMillis());  
    new Thread(new Client()).start();  
  
    //if(b.getState()== Thread.State.RUNNABLE)  
    //Log.i(TAG,"hello thread new");  
    //b.start();  
  
    Log.i(TAG, "Service running");  
  
} catch (InterruptedException e) {  
    e.printStackTrace();  
}  
}  
}  
}  
}  
//running=false;
```

```
        }

    };

    AsyncTask.execute(r);//r.run();

    //a = new Thread(r);

    //a.start();

    //a.yield();

}
```

```
Log.i(TAG,"Returning Sticky");

return START_STICKY;

}

@Override

public void onDestroy()

{

    super.onDestroy();

    running = false;

    Log.i(TAG, "Destroyed Service");

    //System.exit(0);
```

```
}

@Override

public IBinder onBind(Intent intent) {

    // TODO: Return the communication channel to the service.

    throw new UnsupportedOperationException("Not yet implemented");

}

public class Client implements Runnable {

    @Override

    public void run() {

        try {
```

```
Log.i(TAG,"Contacting Server");

tries=0;

do {

    //received="";

    //disarm/arm,detected/notdetected

    //String csv_triggered="IntruderDetected";

    // send message to alarm Raspi

    serverAddr = InetAddress.getByName("70.45.26.180");//70.45.26.180);

    clientSocket = new DatagramSocket();

    sendData = new byte[1024];

    //String sentence = message;

    sendData = message.getBytes();

    sendPacket = new DatagramPacket(sendData, sendData.length, serverAddr,
3491);

    clientSocket.send(sendPacket);
```

```
// get reply back from Arduino/Pi

receiveData1 = new byte[1024];

receivePacket = new DatagramPacket(receiveData1, receiveData1.length);

clientSocket.receive(receivePacket);

received = new String(receivePacket.getData());

//String result;

if (received.charAt(0) == '1') {

    //result = "true";

    notificationAlert();

} else

    //result = "false";

clientSocket.close();

if(received.charAt(0)=='1'||received.charAt(0)=='0')
```

```
        break;

        tries++;

    }while(tries<50);

}

} catch (Exception e) {
    e.printStackTrace();
}

}

}
```

```
package com.rmfintellihomes.rmfintellihomes;

import android.app.AlertDialog;
import android.content.Context;
import android.content.DialogInterface;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Color;
import android.graphics.Point;
import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.Display;
import android.view.KeyEvent;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.webkit.WebView;
import android.webkit.WebViewClient;
import android.widget.Button;
import android.widget.ImageView;
```

```
import android.widget.LinearLayout;

import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;

//*****
//This is the interface that was simulated for the captone presentation
//and holds the complete logic and ui for the application. For UI
//reference please refere to the xml code which is asosciated by ID's
//to the logical options/buttons that the user manipulates.

//*****



public class livingroom extends ActionBarActivity {
```

```
*****  
//The webview is necessary in order to allow video streaming within  
//the android application. It is necessary to allow javascript as  
//part of the configuration options of the webview. The ip address is  
//the external ip address of the home. In this case each system will be  
//forwarded to the corresponding port number.  
*****
```

WebView browser;

```
public static String ARDUINOSERVERIP = "70.45.26.180";//get server ip  
public static int SERVERPORT;  
int tries;
```

```
*****  
//The following variables are logic variables to determine process states  
//logic manipulation and UI association with the xml code just like the  
//main menu code.
```

```
*****
```

```
public String message, received;
```

Button light1, light2,fanpower,fanspeed,fantimer,fancirc, power,
vup,vdown,cup,cdown,one,two,three,four;

Button five, six, seven, eight, nine, zero, exit, input, mode;

ImageView lights, media, temperature, camera;

```
LinearLayout dynamicLayout;
```

```
final Context context = this;
```

boolean processed;

boolean busy;

int selected;

boolean b1selected;

//*****

//bitmaps are used to manipulate the amount of memory being utilized

//by the images of the ui in order to save ram memory usage and pr

//application crash by NOT obtaining and out of memory exce

Bitmap b:

//*****

```
//The oncreate method associates each button to the corresponding xml  
//UI description based on id and usage (please refer to the xml code  
//and project documentation in the app section for visual examples).  
//each option uses an onclick listener which has logical code inside  
//to interact with the system according to the user's request.  
//*****  
  
@Override  
  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_livingroom);  
  
    ImageView back = (ImageView) findViewById(R.id.back);  
  
    b = BitmapFactory.decodeResource(getResources(), R.drawable.livingroominterface);  
    Display screen = getWindowManager().getDefaultDisplay();  
    Point size = new Point();
```

```
screen .getSize(size);

back.setImageBitmap(Bitmap.createScaledBitmap(b,size.x/4,size.y/4,false));

b.recycle();

dynamicLayout = (LinearLayout) findViewById(R.id.dynamic);

lights = (ImageView) findViewById(R.id.livingroomlighting);

media = (ImageView) findViewById(R.id.media);

temperature = (ImageView) findViewById(R.id.temperature);

camera = (ImageView) findViewById(R.id.camera);

camera.setOnClickListener(new View.OnClickListener() {

    @Override

    public void onClick(View v) {
```

```
lights.setBackgroundColor(Color.TRANSPARENT);
media.setBackgroundColor(Color.TRANSPARENT);
temperature.setBackgroundColor(Color.TRANSPARENT);
camera.setBackgroundColor(0x8011aacc);

dynamicLayout.removeAllViews();
dynamicLayout.addView(LayoutInflater().inflate(R.layout.livingroomcamera,null));

//*****
//This is the camera/livestream configuration.

//*****
browser = (WebView) findViewById(R.id.livingroomcamview);
browser.setPadding(0,0,0,0);
browser.setInitialScale(30);//30
```

```
browser.getSettings().setJavaScriptEnabled(true);

browser.getSettings().setJavaScriptCanOpenWindowsAutomatically(true);

browser.getSettings().setAllowFileAccessFromFileURLs(true);

browser.getSettings().setAllowFileAccess(true);

browser.getSettings().setAllowUniversalAccessFromFileURLs(true);

browser.setWebViewClient(new WebViewClient());

browser.loadUrl("http://70.45.26.180/camindex.php");//http://70.45.26.180/camindex.php");//http://70.45.26.180
```

```
        }

});

lights.setOnClickListener(new View.OnClickListener() {

    @Override

    public void onClick(View v) {

        lights.setBackgroundColor(0x8011aacc);

    }
});
```

```
media.setBackgroundColor(Color.TRANSPARENT);
temperature.setBackgroundColor(Color.TRANSPARENT);
camera.setBackgroundColor(Color.TRANSPARENT);

dynamicLayout.removeAllViews();
dynamicLayout.addView(LayoutInflater().inflate(R.layout.livingroomlighting,null));

selected =1;

light1 = (Button) findViewById(R.id.livingroombulb);
b1selected=true;
message = "aut";//getting light1 current state

new Thread(new Client()).start();
```

```
light1.setOnClickListener(new View.OnClickListener() {

    @Override

    public void onClick(View v) {

        b1selected=true;

        SERVERPORT=10;

        mode.setText("Auto");

        ARDUINOSERVERIP = "70.45.26.180";//192.168.1.140

        if(light1.getText().toString().equals("off")){

            light1.setText("on");

            message = "on1";

        }

        else{

            light1.setText("off");

            message = "of1";

        }

        new Thread(new Client()).start();

    }

})
```

```
        Log.i("com.rmfintellihomes.rmfintellihomes","Processed: "+processed);  
    }  
});
```

```
light2 = (Button) findViewById(R.id.livingroombulb2);

//message = "kb2";//getting light1 current state

//new Thread(new Client()).start();

light2.setOnClickListener(new View.OnClickListener() {

    @Override

    public void onClick(View v) {

        mode.setText("Auto");

        //ARDUINOSERVERIP = "192.168.1.140";//70.45.26.180

        if(light2.getText().toString().equals("off")){

            light2.setText("on");

            message = "on2";

        }

        else{

            light2.setText("off");

            message = "of2";

        }

        new Thread(new Client()).start();

    }

});
```

```
    }  
});
```

```
mode= (Button) findViewById(R.id.systemMode);  
mode.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
  
        //ARDUINOSERVERIP = "192.168.1.140";//70.45.26.180
```

```
if(mode.getText().equals("Auto")) {  
    mode.setText("Manual");  
    message = "man";  
}  
  
else{  
    mode.setText("Auto");  
    message = "aut";  
}
```

```
new Thread(new Client()).start();  
  
}  
});  
  
}  
});
```

```
media.setOnClickListener(new View.OnClickListener() {  
  
    @Override  
  
    public void onClick(View v) {  
  
        selected =2;  
  
        lights.setBackgroundColor(Color.TRANSPARENT);  
  
        media.setBackgroundColor(0x8011aacc);  
  
        temperature.setBackgroundColor(Color.TRANSPARENT);  
  
        camera.setBackgroundColor(Color.TRANSPARENT);  
  
  
        dynamicLayout.removeAllViews();  
  
        dynamicLayout.addView(LayoutInflater.inflate(R.layout.livingroommedia,null));  
  
  
        power = (Button)findViewById(R.id.livingroomtv);  
  
        power.setOnClickListener(new View.OnClickListener() {
```

```
@Override  
public void onClick(View v) {  
  
    message="pwr";  
    new Thread(new Client()).start();  
  
}  
});  
  
vup = (Button)findViewById(R.id.volumeup);  
vup.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
  
        message="vl+";
```

```
        new Thread(new Client()).start();  
    }  
});  
  
vdown = (Button)findViewById(R.id.volumedown);  
vdown.setOnClickListener(new View.OnClickListener() {  
  
    @Override  
  
    public void onClick(View v) {  
  
        message="vl-";  
  
        new Thread(new Client()).start();  
    }  
});  
  
cup = (Button)findViewById(R.id.channelup);  
cup.setOnClickListener(new View.OnClickListener() {  
  
    @Override  
  
    public void onClick(View v) {
```

```
        message="ch+";
        new Thread(new Client()).start();
    }
});

cdown = (Button)findViewById(R.id.channeldown);
cdown.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        message="ch-";
        new Thread(new Client()).start();
    }
});

one = (Button)findViewById(R.id.one);
one.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        message="bt1";
    }
});
```

```
        new Thread(new Client()).start();  
    }  
});  
  
two = (Button)findViewById(R.id.two);  
two.setOnClickListener(new View.OnClickListener() {  
  
    @Override  
  
    public void onClick(View v) {  
  
        message="bt2";  
  
        new Thread(new Client()).start();  
    }  
});  
  
three = (Button)findViewById(R.id.three);  
three.setOnClickListener(new View.OnClickListener() {  
  
    @Override  
  
    public void onClick(View v) {
```

```
        message="bt3";
        new Thread(new Client()).start();
    }
});

four = (Button)findViewById(R.id.four);
four.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        message="bt4";
        new Thread(new Client()).start();
    }
});

five = (Button)findViewById(R.id.five);
five.setOnClickListener(new View.OnClickListener() {
```

```
@Override  
public void onClick(View v) {  
  
    message="bt5";  
    new Thread(new Client()).start();  
}  
});  
  
six = (Button)findViewById(R.id.six);  
six.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
  
        message="bt6";  
        new Thread(new Client()).start();  
    }  
});
```

```
seven = (Button)findViewById(R.id.seven);
seven.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        message="bt7";
        new Thread(new Client()).start();
    }
});

eight = (Button)findViewById(R.id.eight);
eight.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
```

```
        message="bt8";
        new Thread(new Client()).start();
    }
});

nine = (Button)findViewById(R.id.nine);
nine.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        message="bt9";
        new Thread(new Client()).start();
    }
});

zero = (Button)findViewById(R.id.zero);
zero.setOnClickListener(new View.OnClickListener() {
```



```
        message="ext";  
        new Thread(new Client()).start();  
    }  
});  
  
input = (Button)findViewById(R.id.input);  
input.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        message="inp";  
        new Thread(new Client()).start();  
    }  
});  
});
```

```
temperature.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        selected =3;  
        lights.setBackgroundColor(Color.TRANSPARENT);  
        media.setBackgroundColor(Color.TRANSPARENT);  
        temperature.setBackgroundColor(0x8011aacc);  
        camera.setBackgroundColor(Color.TRANSPARENT);  
  
        dynamicLayout.removeAllViews();  
  
        dynamicLayout.addView(LayoutInflater.inflate(R.layout.livingroomtemperature,null));  
  
        fanpower = (Button)findViewById(R.id.livingroomfan);  
        //message = "fps";//getting fanpower current state  
        //new Thread(new Client()).start();  
        fanpower.setOnClickListener(new View.OnClickListener() {  
            @Override  
            public void onClick(View v) {
```

```
        message = "fpw";\n\n        new Thread(new Client()).start();\n    }\n}\n\nfanspeed = (Button)findViewById(R.id.fanspeed);\nfanspeed.setOnClickListener(new View.OnClickListener() {\n    @Override\n    public void onClick(View v) {\n\n        message = "spd";\n    }\n}\n\n
```

```
        new Thread(new Client()).start();

    }

});

fancirc = (Button)findViewById(R.id.Cycle);

fancirc.setOnClickListener(new View.OnClickListener() {

    @Override

    public void onClick(View v) {

        message = "rot";

        new Thread(new Client()).start();

    }

});
```

```
fantimer = (Button)findViewById(R.id.Timer);

fantimer.setOnClickListener(new View.OnClickListener() {

    @Override

    public void onClick(View v) {

        message = "tim";

        new Thread(new Client()).start();

    }

});
```

```
 }  
});
```

```
}
```

```
private void notProcessed() {  
  
    AlertDialog.Builder alertDialogBuilder = new AlertDialog.Builder(context);  
  
    // set title
```

```
AlertDialogBuilder.setTitle("RMF Intelligent Homes");

// set dialog message

AlertDialogBuilder
.setMessage("Request Not Processed! Retry?")
.setCancelable(false)

.setPositiveButton("Yes",new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog,int id) {
        // if this button is clicked, close
        // current activity
        //MainActivity.this.finish();
        new Thread(new Client()).start();

        for(int x=0;x<75;x++){
    }
}});
```

```
        if(!processed)

            notProcessed();

    }

})

.setNegativeButton("No", new DialogInterface.OnClickListener() {

    public void onClick(DialogInterface dialog, int id) {

        // if this button is clicked, just close

        // the dialog box and do nothing

        dialog.cancel();

    }

});

// create alert dialog

AlertDialog alertDialog = alertDialogBuilder.create();
```

```
// show it

AlertDialog.show();

}

@Override

public boolean onCreateOptionsMenu(Menu menu) {

    // Inflate the menu; this adds items to the action bar if it is present.

    getMenuInflater().inflate(R.menu.menu_livingroom, menu);

    return true;

}

@Override

public boolean onKeyDown(int keyCode, KeyEvent event){

    if((keyCode== KeyEvent.KEYCODE_BACK))
```

```
message = "man";  
new Thread(new Client()).start();  
b.recycle();  
finish();  
}  
return super.onKeyDown(keyCode, event);  
}  
  
@Override  
public boolean onOptionsItemSelected(MenuItem item) {  
    // Handle action bar item clicks here. The action bar will  
    // automatically handle clicks on the Home/Up button, so long  
    // as you specify a parent activity in AndroidManifest.xml.  
    int id = item.getItemId();
```

```
//noinspection SimplifiableIfStatement  
if (id == R.id.action_settings) {  
    return true;  
}  
  
return super.onOptionsItemSelected(item);  
}
```

```
//divide this class into updateClient and modifyClient
//by having the getstatus part of the system in updateClient
//and the on/off/change/etc in the modifyClient part.

public class Client implements Runnable {

    @Override

    public void run() {

        try {

            //Log.i("com.rmfintellihomes.rmfintellihomes","Entered Client");

            tries=0;

            processed=false;

            busy=true;

            received="";

            do {

                Log.i("com.rmfintellihomes.rmfintellihomes","tries: "+tries);

                //*****
                //Lighting interface communication
```

```
/*
if (selected == 1) {

    byte[] receiveData1 = new byte[1024];

    // send message to Arduino/Pi

    InetAddress serverAddr = InetAddress.getByName("70.45.26.180");

    byte[] sendData = new byte[1024];

    //String sentence = message;

    sendData = message.getBytes();

    DatagramPacket sendPacket = new DatagramPacket(sendData,
sendData.length, serverAddr, 10);

    DatagramSocket clientSocket = new DatagramSocket();

    clientSocket.send(sendPacket);

}

DatagramPacket receivePacket = new DatagramPacket(receiveData1,
receiveData1.length);

// get reply back from Arduino/Pi

clientSocket.receive(receivePacket);
```

```

received = new String(receivePacket.getData());

clientSocket.close();

// Log.w("\n\n\nRMFIntelliHomes", "Entered selected:1 lighting\n\n\n");

}

//*****
//Media interface communication

//*****

if (selected == 2||selected==3) {

byte[] receiveData2 = new byte[1024];

// send message to Arduino/Pi

InetAddress mediaserverAddr = InetAddress.getByName("70.45.26.180");

byte[] mediasendData = new byte[1024];

//String sentence = message;

mediasendData = message.getBytes();

DatagramPacket mediasendPacket = new DatagramPacket(mediasendData,
mediasendData.length, mediaserverAddr, 16);

DatagramSocket mediasocket = new DatagramSocket();

mediasocket.send(mediasendPacket);

```



```
        if (received.charAt(0) == 'a' && received.charAt(1) == 'c' && received.charAt(2)
== 'k') {

            Log.i("com.rmfintellihomes.rmfintellihomes", "Exiting Socket while loop");

            processed = true;

            break;

        }

    }

tries++;

}while(tries<50);

busy = false;
```

```
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

}

package com.rmfintellihomes.rmfintellihomes;

import android.graphics.Color;
import android.graphics.drawable.Drawable;
import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.KeyEvent;
import android.view.Menu;
import android.view.MenuItem;
```

```
import android.widget.TextView;

//*****
//This library is included from an online resource in JRE format. It
//is included in the gradle configurations code. It serves as a tool
//for graphical representations.

//*****

import com.jjoe64.graphview.GraphView;
import com.jjoe64.graphview.LegendRenderer;
import com.jjoe64.graphview.helper.StaticLabelsFormatter;
import com.jjoe64.graphview.series.BarGraphSeries;
import com.jjoe64.graphview.series.DataPoint;
import com.jjoe64.graphview.series.LineGraphSeries;
import java.net DatagramPacket;
import java.net DatagramSocket;
import java.net InetAddress;
```

```
public class powerConsumption extends ActionBarActivity {  
  
    String []values= new String[10];  
  
    public TextView g;  
    String received;  
    String message;  
    String user;  
    int tries;  
  
    //*****  
    //The oncreate method uses the variables above for logical communication  
    //between the server and the user. It associates the graphical data with  
    //the updated data every time the user accesses the power consumption  
    //interface.  
    //*****  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {
```

```
super.onCreate(savedInstanceState);

Bundle info = getIntent().getExtras();

user = info.getString("user");

setContentView(R.layout.activity_power_consumption);

getSupportActionBar().setTitle("Power Consumption");

g = (TextView) findViewById(R.id.powerdata);

//*****
//Using the BarGraphSeries the data is represented accordingly.

//*****
```

```
BarGraphSeries<DataPoint> series = new BarGraphSeries<DataPoint>(new DataPoint[] {

    new DataPoint(0, 233.14),

    new DataPoint(1, 245.12),

    new DataPoint(2, 333.13),

    new DataPoint(3, 168.22),

    new DataPoint(4, 85.36),
```

```
new DataPoint(5, 233.44),  
new DataPoint(6, 245.52),  
new DataPoint(7, 333.16),  
new DataPoint(8, 168.22),  
new DataPoint(9, 85.63)  
});  
  
//new Thread(new Client()).start();  
  
//*****  
//The graph data is associated with a graphView element (refer to xml)  
//and the configured to the specific needs. In this case the  
//x axis is the day and the y axis is the power consumed in watts.
```

```
*****  
GraphView graph = (GraphView) findViewById(R.id.graph);  
  
graph.setViewport().setXAxisBoundsManual(true);  
  
graph.setViewport().setMaxX(9);  
  
graph.setViewport().setMinX(0);  
  
StaticLabelsFormatter staticLabelsFormatter = new StaticLabelsFormatter(graph);  
  
//staticLabelsFormatter.setHorizontalLabels(new String[] {"Day1","Day5","Day10"});  
  
staticLabelsFormatter.setHorizontalLabels(new  
String[]{"1","2","3","4","5","6","7","8","9","10"});  
  
//staticLabelsFormatter.setVerticalLabels(new String[] {"low", "medium", "high"});  
  
graph.getLabelRenderer().setLabelFormatter(staticLabelsFormatter);  
  
//graph.setViewport().setScalable(true);  
  
//graph.setViewport().setScrollable(true);  
  
series.setTitle("Watts per Day");
```

```
series.setValuesOnTopColor(Color.BLUE);
series.setDrawValuesOnTop(true);
series.setValuesOnTopSize(22);

graph.getLegendRenderer().setVisible(true);
graph.getLegendRenderer().setAlign(LegendRenderer.LegendAlign.TOP);
graph.addSeries(series);
```

```
}
```

```
@Override
```

```
public boolean onKeyDown(int keyCode, KeyEvent event)
```

```
{
```

```
    if ((keyCode == KeyEvent.KEYCODE_BACK))
```

```
    {
```

```
        finish();
```

```
    }
```

```
    return super.onKeyDown(keyCode, event);
```

```
}
```

```
@Override
```

```
public boolean onCreateOptionsMenu(Menu menu) {
```

```
    // Inflate the menu; this adds items to the action bar if it is present.
```

```
getMenuInflater().inflate(R.menu.menu_power_consumption, menu);

return true;

}

@Override

public boolean onOptionsItemSelected(MenuItem item) {

    // Handle action bar item clicks here. The action bar will
    // automatically handle clicks on the Home/Up button, so long
    // as you specify a parent activity in AndroidManifest.xml.

    int id = item.getItemId();

    //noinspection SimplifiableIfStatement
    if (id == R.id.action_settings) {
        return true;
    }

    return super.onOptionsItemSelected(item);
}
```

```
public class Client implements Runnable {  
    @Override  
    public void run() {  
        try {  
            tries=0;  
  
            do {  
                byte[] receiveData = new byte[1024];  
  
                // send message to Arduino/Pi  
  
                InetAddress tempserverAddr = InetAddress.getByName("70.45.26.180");  
  
                byte[] tempsendData = new byte[1024];  
  
                message = "energy_monitor," + user + ",energy";  
  
                tempsendData = message.getBytes();  
  
                DatagramPacket tempsendPacket = new DatagramPacket(tempsendData,  
tempsendData.length, tempserverAddr, 1111);  
  
                DatagramSocket tempsocket = new DatagramSocket();  
  
                tempsocket.send(tempsendPacket);  
  
                DatagramPacket receivePacket = new DatagramPacket(receiveData,  
receiveData.length);
```

```
// get reply back from Arduino/Pi  
//lakjsljf  
  
tempsocket.receive(receivePacket);  
  
received = new String(receivePacket.getData());  
tempsocket.close();  
  
if(Character.isDigit(received.charAt(0)))  
    break;  
  
tries++;  
}  
while(tries<50);  
Log.i("RMFIIntelliHomes", "received: " + received);
```

```
runOnUiThread(new Runnable() {  
    @Override  
    public void run() {  
  
        //stuff that updates ui  
  
        int vcount=0,i=0;  
  
        String temp="";  
  
        while(vcount<10){  
  
            if(received.charAt(i)==','){  
  
                values[vcount]= temp;  
  
                temp="";  
  
                vcount++;  
  
            }  
  
            else if (Character.isDigit(received.charAt(i))||received.charAt(i)=='.'){
```

```
temp+=received.charAt(i);

}

else{

    values[vcount]= temp;

    temp="";

    vcount++;

    break;

}

i+=1;

}

double []vs= new double[10];

for(i=0;i<vcount;i++)

    vs[i]= Double.valueOf(values[i]);

BarGraphSeries<DataPoint> pseries = new BarGraphSeries<>(new DataPoint[]{

    new DataPoint(1,vs[0]),

    new DataPoint(2,vs[1]),

    new DataPoint(3,vs[2]),

    new DataPoint(4,vs[3]),

    new DataPoint(5,vs[4]),

    new DataPoint(6,vs[5]),
```



```
graph.getViewport().setXAxisBoundsManual(true);

graph.getViewport().setMaxX(10);

graph.getViewport().setMinX(0);

StaticLabelsFormatter staticLabelsFormatter = new

StaticLabelsFormatter(graph);

    staticLabelsFormatter.setHorizontalLabels(new String[] {"Day1", "Day2",

"Day3", "Day4", "Day5", "Day6", "Day7", "Day8", "Day9", "Day10"});

    staticLabelsFormatter.setVerticalLabels(new String[] {"low", "medium", "high"});

    graph.getLabelRenderer().setLabelFormatter(staticLabelsFormatter);

//graph.getViewport().setScalable(true);

//graph.getViewport().setScrollable(true);

pseries.setTitle("User Energy Consumption");
```

```
graph.getLegendRenderer().setVisible(true);  
graph.getLegendRenderer().setAlign(LegendRenderer.LegendAlign.TOP);  
graph.addSeries(pseries);
```

```
    }

});//end runnable

//}while(received.equals(" ")&&tries<50);

} catch (Exception e) {
    e.printStackTrace();
}
```

}

}

}

Each Java code includes a inner class called Client. This client class utilizes udp communication to interact with the RMF Intelligent home system. It uses port number and Ip address. For exeternal communications, the IP address of the home is included inside the packet configuration. the corresponding port number of the system is the included in the packet in order for the communication to be forwarded to the corresponding system. Finally the packet will the include the message intended for the system. This message is manipulated logically by the java code and the oncreate method according to the user's events and decisiong with the UI. Lastly the UI will be updated with each operation of the user.

Chapter 15 Backup System

15.1 Battery Back-Up System

15.1.1 Introduction to the System

The RMF Intelligent Home System is composed of very sensitive electronic components that make the household smart and unique. Because electronics devices are known to be extremely sensitive to power outages and voltage flow fluctuation, a protection to the system is tremendously necessary. In addition to damaging the systems hardware, both power outages and voltage flow fluctuations can also cause system reboots and programming errors. RMF Intelligent Home took these problems into consideration when designing and constructing its smart system. RMF Intelligent Home designed an Uninterruptable Power Source (UPS) or Battery Back-Up System in charge of supplying power and protection to the system electronics and keeping it up and running at all times.

The Uninterruptable Power Source or UPS will possess a Battery Bank with a number of batteries interconnected in parallel and series connections to supply power to the system. These batteries will keep the system off-grid which in return will protect the system from daily voltage flow fluctuations. Take note that the Battery Bank's size will be determined by how much time the owner of the house wants the system up and running when there is no power. Depending on the choice of time, the owner will know how big or small the UPS will finally be. Although the UPS is only for the system electronics, the size of the system varies on how big the household is and how much control over the house those the owner(s) wants to have. RMF Intelligent Home will provide useful information about Battery Bank's which will explain all the necessary details to take

into consideration when installing an Uninterruptable Power System for the electronic components. The information will help the owner understand how many batteries those the system need and which electrical equipment's to install with the batteries.

15.2 System Specifications

15.2.1 Battery Bank Size Calculation

When designing a battery bank many variables come into consideration and much confusion can come to mind. For these reason RMF Intelligent Home is providing information and a basic step to follow table that will help in the design of the Battery Bank based on the total applications that would run electrically speaking during an emergency. Please note that although this table would help in the design of almost every Battery Bank System, its prime focus is on supplying power to electronic components. That being said, the first and most important part of designing a Battery Bank is to know the Total Load of all the applications that will keep working if a power outage should occur. The Total Load of all the applications will provide how many batteries the system will need and which kind of DC-to-AC Inverter should we use in the design (either 12VDC or 24VDC, and its Watts Rating). Take note that to find the Total Load of the system, the Total Load of electrical equipment must first be calculated. This can be determining by looking at the input electrical nameplate of each appliance or by using some basic electrical formulas.

$$P = V * I \text{ (Watts)}$$

$$I = P/V \text{ (Amperes)}$$

$$Pin = Pout/\eta$$

Where:

P = Power (W)

V = Voltage (V)

I = Current (A)

P_{in} = Input Power (W)

P_{out} = Output Power (W)

η = Efficiency (%)

After acquiring the Total Load of all the components and adding them all together and acquire the Total Load of the system, the next step is to find the right DC-to-AC Inverter and Battery Charger.

Batteries or Battery Banks store energy or electricity in DC or Direct Current form. Because we need to convert this form into AC or Alternate Current, a DC-to-AC Inverter is required. In addition to the DC-to-AC Inverter a Battery Charger is also necessary for two reasons. The Battery Charger will provide current to charge the batteries and keep them fully charged, but also it will supply current to the system to keep it up and running. What is essential when selecting a DC-to-AC Inverter are two very important things. The DC-to-AC Inverter must provide the necessary Watts based on the Total Load previously calculated, but also must have a good Efficiency. In every electrical device, equipment or system power is lost. Manufacturers provide the efficiency of their equipment on the nameplate of the electrical equipment's, but if it's not provided, it can be acquired by dividing the output power by the input power. Most Inverters

have an 80 to an 85 percent efficiency. The Battery Charger follows the same parameters as the DC-to-AC Inverter. This gives a better knowing on the amperes the charger have to provide to both the Battery Bank and the System.

Batteries supply a DC Voltage (usually being 12VDC) and Amperes per Hour. The total Amperes per Hour the Battery Bank has will define how long the system runs based on the Total Load of the System. The question the owner has to answer is simple, How long is the desire time for the Load to run? Based on the time frame given then the Amperes per Hour will then be used. Before calculating the Amperes per Hour it's important to find the Amp or Current Rating from our Load using the current formula previously presented. This Current Rating is then multiplied by the hours the owner want the system to run during a power outage. The result will give the Amperes per Hour you need the batteries to supply in order to maintain the system up and running for that amount of time. To improve the lifetime of the batteries, it is recommended that batteries should not be discharged completely. The Maximum allowable discharge depends on the type of batteries that are used to supply power to the system. The battery manufacturer provides all the necessary information to use with this guide.

15.2.2 Calculations for the Battery Back-Up System

Using the previously presented steps and equations, RMF Intelligent Home is presenting the calculations used for building the Battery Back-Up System or UPS. Before presenting the step by step procedure, let us remember all the electronic components that will be running using the Back-Up System. Three Raspberry Pi 2, three Arduino One, one Router, one Modem, and the camera LEDs.

Raspberry Pi 2: Input Voltage = 5V, Ampere = 2.5A max

Arduino One: Input Voltage 9V, Ampere = 2.5A max

Router: Input Voltage = 12V, Ampere = 1.5A max

Modem: Input Voltage = 15V, Ampere = 1.5A max

Camera LEDs: Input Voltage = 12V, Ampere = 1A max

Now we proceed obtaining each electronic component Total Load and then continue to add them until we acquired the Total Load of the System.

Raspberry Pi 2: $P = 5V * 2.50A$

Raspberry Pi 2: $P = 12.50 \text{ Watts} * 3$

Total Raspberry Pi 2 Load: $P = 37.50 \text{ Watts}$

Arduino One: $P = 9V * 2.50A$

Arduino One: $P = 22.50 \text{ Watts} * 3$

Total Arduino One Load: $P = 67.50 \text{ Watts}$

Router: $P = 12V * 1.50A$

Total Router Load: $P = 18 \text{ Watts}$

Modem: $P = 15V * 1.50A$

Total Modem Load: $P = 22.50 \text{ Watts}$

Camera LEDs: $P = 12V * 1A$

Total Camera LEDs Load: $P = 12 \text{ Watts}$

Total Load of the System: $P = 157.50 \text{ Watts}$

After acquiring the Total Load of the System, the next step is to calculate the Current Rating of the system in order to calculate the Ampere Hour needs to know how many batteries the system will need.

Current Rating: $I = P/V$

Current Rating: $I = 157.50 \text{ W} / 12\text{V}$

Current Rating: $I = 13.125\text{A}$

Ampere per Hour: $A.H = I * \text{Hour}(s)$

Ampere per Hour: $A.H = 13.125\text{A} * 1\text{ Hr}$

Ampere per Hour: $A.H = 13.125\text{A.Hr}$

The results provided by the calculations helped RMF Intelligent Home obtained the data needed to design the UPS or Battery Back-Up System to supply power to the system electronic components for one hour (1 Hr) in case a power outage occurs. The final design will have three 12VDC, 5 Ampere per Hour batteries connected in parallel.

15.2.3 Electric Design

The electric design for the Battery Back-Up System is quite simple and was designed using the AutoCAD design program. For the Battery Back-Up System there are two electric designs to be presented. The first one is the Single Line Diagram, which is the most simple and basic electric design of the system. The Single Line Diagram lets any one understand the components connections and how the system is expected to work. It also establishes all the electrical equipment that is going to be used. The second is the final Electrical Design Blueprint that presents all the electric circuits and connections found in the Battery Back-Up System. The Electrical Design Blueprint provides a layout and explanation on how the power flow of the system will behave. Note that this design is based using the calculations obtained from the power consumption (Watts) of the system electronic components.

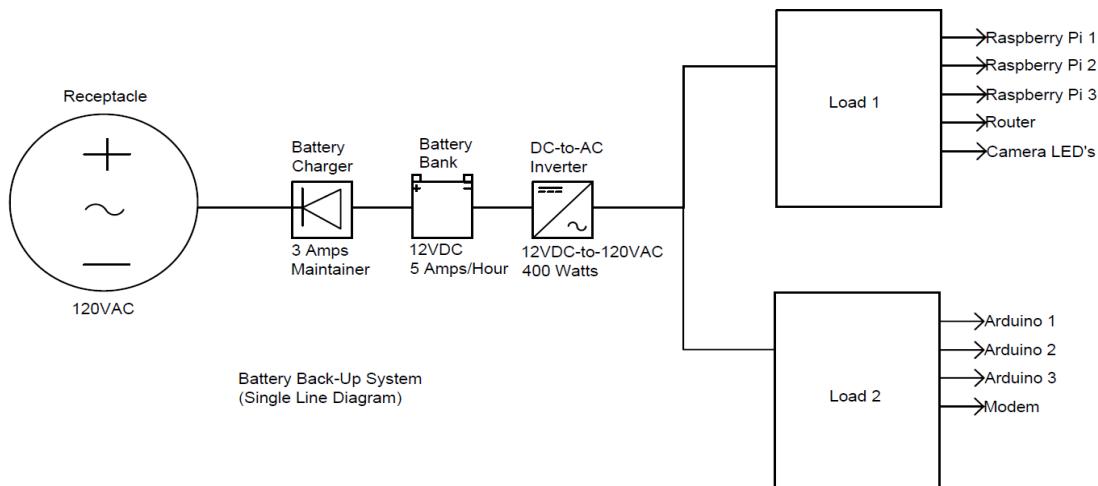


Figure 15.2.1 – Single Line Diagram of the Battery Backup System

The Battery Back-Up System was constructed using a Programmable Battery Charger, a Battery Bank of three batteries (12VDC, 5 Amperes per Hour each), a DC-to-AC Inverter (12VDC, 400 Watts), jumper cables (for our project we used #12 AWG, Cu cables) with connectors to fit the battery terminals (6 AMP Female Disconnects for #12 and #10 AWG wires), and two multi-plugs to connect the independent power supplies for each electronic component. The first connection is to connect the Battery Charger to the 120VAC Receptacle socket. The second connection will be a parallel connection between one of the batteries of the Battery Bank. A parallel connection is simply connecting positive with positive and negative with negative. Since the DC-to-AC Inverter works with 12VDC then the third connection will be a parallel connection between all three batteries. This will maintain a 12VDC and at the same time boost the amperes per hour of the batteries. The final connection will be to the batteries in parallel DC-to-AC Inverter. Now that the Power Inverter has 12VDC to convert to 120VAC, connect the multi-plugs to the electric sockets of the DC-to-AC Inverter with all the electric components connected to the multi-plugs.



Figure 15.2.2 – Enercell Sealed Lead Acid Battery 12V 5A per hour



Figure 15.2.3 – LiPro Balance Charger



Figure 15.2.4 – Independent Charger



Figure 15.2.5 – 400W DC-to-AC Inverter



Figure 15.2.6 – Stranded Wire 12AWG



Figure 15.2.7 – Multi Outlet

Take note that this system was constructed to function at all times. Because the Battery Back-Up System is always supplying power to the electronics, the current that the Battery Charger provides are split between the system and the Battery Bank. In this case the Battery Charger, which is able to maintain three amperes (3 Amps), will approximately supply two amperes (2 Amps) directly to the system and one ampere (1 Amp) to the Battery Bank. This is because the system demands more current because it is being used at all times. The moment a power outage occurs the batteries are in charge of acting as an alternate source of power for the system. This particular system RMF Intelligent Home constructed is particularly build for supplying power for one hour. With more batteries and a better Power Inverter and Battery Charger, a better longer lasting system can be built.

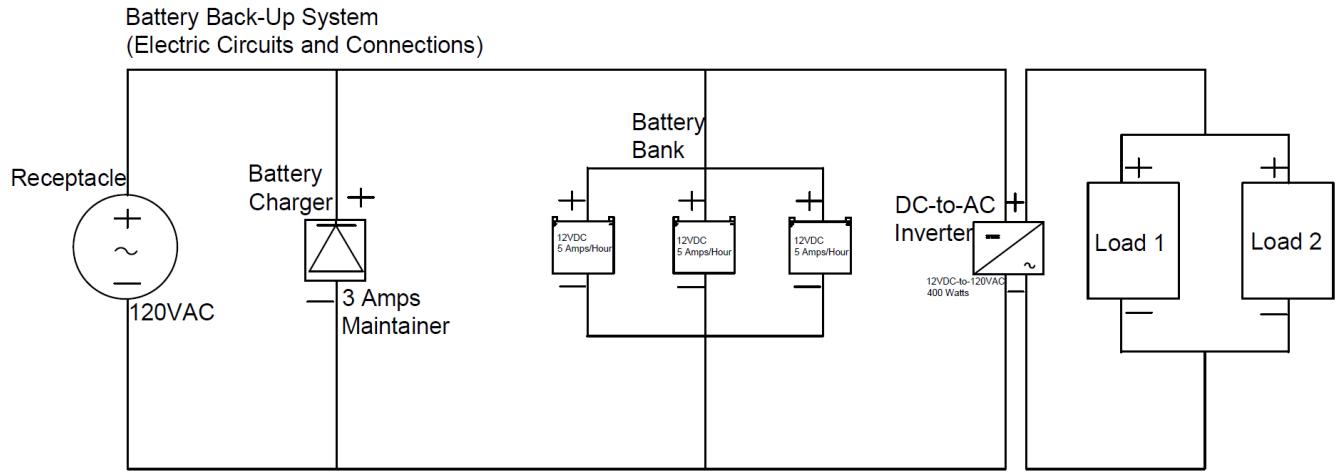


Figure 15.2.8 – Battery Backup Circuit Diagram

15.3 Results

15.3.1 Backup System Tests

After the design phase was completed and the system was implemented, its performance was tested. The first test was to verify the performance of the batteries without the charger connected. This test was necessary to verify that the discharge time is enough for the system to withstand a blackout. The test is presented in Figure 15.3.1 and Figure 15.3.2.

Battery Discharge Time										
Voltage	12.10	12.05	12.00	11.95	11.90	11.85	11.80	11.75	11.70	11.65
Time	0:00	7:45	15:07	24:46	33:33	41:55	50:06	58:32	1:05:59	1:13:03

Figure 15.3.1 – Battery Backup Discharge Time Values

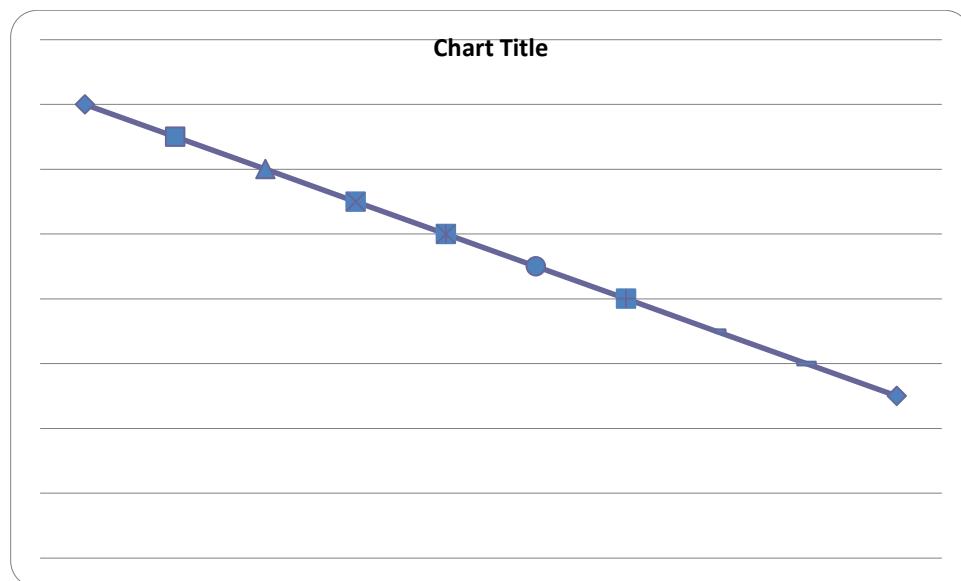


Figure 15.3.2 – Battery Backup Discharge Time

The regulator was programmed to provide 12.00 V and 2.9 A. Because the batteries are Sealed Lead-Acid, they need to be charged at low currents so we provide enough current for the system to operate and there is still approximately 0.09 A, which is divided between the three batteries.

Chapter 16: RMF Intelligent Home System

16.1 Market Evaluation

In order to get a better perspective of how to develop the RMF Intelligent Home system, the team evaluated various security and smart-home systems that are available at the market. This helps us to better visualize the market needs, as well as what aspects could lead us to distinguish RMF Intelligent Home system from others.

16.1.1 ADT Security Services

ADT Security is one of the leaders in the security systems market. Their main focus is on providing security to houses, and recently added some basic aspects to home automation. Their services include:

- Smartphone system connection and alarm control.
- Fire and smoke monitoring.
- Carbon monoxide monitoring.
- Water detection and temperature monitoring.
- Email and text notifications.
- Lighting, garage door and thermostat control.
- Remote secure video.
- Backup Battery for the system.

The ADT Security Services system prices vary from \$36.99 to \$49.99 monthly payments, depending on the system capacities. These prices do not include the installation price, which vary depending on the house size and the systems that will be included in the package.

ADT Security system is a good model to look at because they have been working for over 140 years and still they keep evolving during the years, adding new features as technology keeps advancing.

16.1.2 Leviton Home Automation

Leviton Home Automation is a worldwide company providing home automation services, which also incorporate some basic security services. They provide industrial, government, and home services, using their own brand of controllers. These controllers are adapted to their necessities, which give them an advantage. Some of their services include:

- Energy Management.
- Lighting Control.
- Bluetooth entertainment control.
- Camera surveillance.
- Security alarm system.
- Smartphone system connection and alarm control.

Leviton system is a great model to look at because of the many services that they provide. This variety provides a good guidance for the RMF Intelligent Home system to follow.

16.1.3 RMF Intelligent Home System

RMF Intelligent Home system focus on homes, and its priority is to provide home automation and security to families. RMF Intelligent Home system uses market microcontrollers as prototypes to provide feasible and low cost solutions to homeowners. One of the advantages of the RMF Intelligent Home system is that it uses Internet of Things (IoT) features for each of the systems, so although all of the systems interact with each other, each system is an “individual” system. Some of our services include:

- Energy consumption monitoring.
- Lighting Control.
- Camera Surveillance.
- NFC tag and digital keypad verification system.
- Remote IR appliances control.
- Security alarm system.
- Smartphone system connection and alarm control.
- Backup Battery for the system.

16.2 Results

16.2.1 RMF Intelligent Home System Implementation

RMF Intelligent Home system uses the Internet of Things as the standard to all of its systems, so every system has its own IP address. This individual IP address is what the systems use to communicate between them. Figure 16.2.1 demonstrates how each of the systems has a specific IP address and they are all connected to a local network using a star network topology.

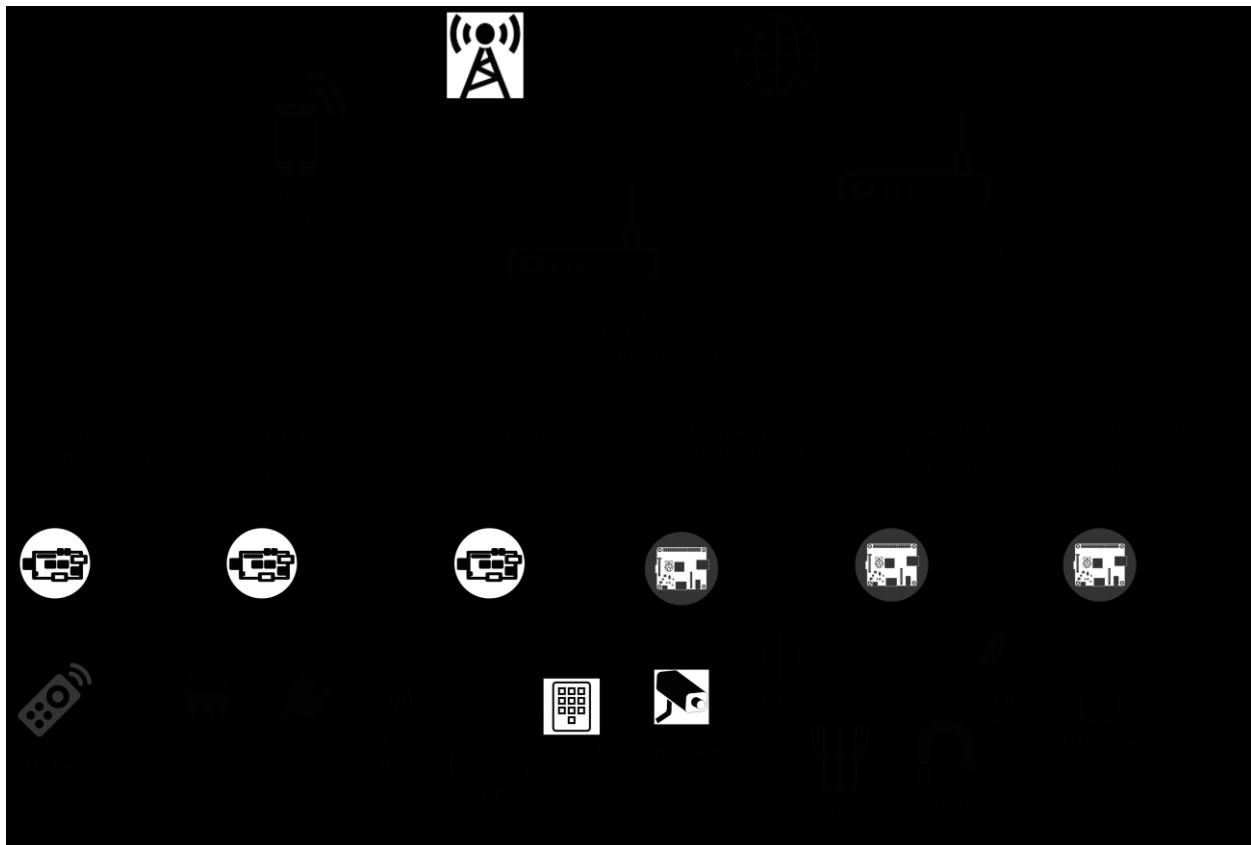


Figure 16.2.1 – RMF Intelligent Home System Network

16.2.2 RMF Intelligent Home System Testing

The RMF Intelligent Home performed several tests to the system to verify that the outcomes are as expected. The tests were focused to uncover errors and bugs in the system. Each test was performed at least ten times, in order to verify that the results are reliable. The tests were established so each of the systems could be tested within the local network and with an external connection. To verify a proper functionality of the systems, the commands were sent from a computer and also from the phone app.

The performed tests are presented in the table below, divided by systems. We marked the results with a Y if these were positive, and marked with an N if these were negative. At the same

time that the tests were performed some of the problems were fixed. The systems were tested again after the new solutions were implemented.

The first tested system was the Energy Monitor system. We performed 4 tests on this system, which gave us a better understanding of the system performance. The performed tests are presented in Figure 16.2.2.

Test	1	2	3	4	5	6	7	8	9	10
Energy Monitor										
1 Compare measurements with those from the kill-a-watt.	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
2 Change loads and read measurements	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
3 Database request measurements from Arduino.	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
4 Phone request measurements from database.	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

Figure 16.2.2 – Energy Monitor System Tests

The next tested system was the Lighting system. We performed 4 tests on this system, which gave us a better understanding of the system performance. The performed tests are presented in Figure 16.2.3.

Test	1	2	3	4	5	6	7	8	9	10
Lighting System										
1 Turn ON/OFF using serial commands	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
2 Turn ON/OFF using the smartphone.	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
3 Turn ON/OFF with phone and try to change status with manual switch.	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
4 Turn ON/OFF manually and try to change status with phone.	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

Figure 16.2.3 – Lighting System Tests

The next tested system was the Alarm system. We performed 9 tests on this system to get a better understanding of the system performance. The performed tests are presented in Figure 16.2.4.

Test		1	2	3	4	5	6	7	8	9	10
Alarm System											
1	Turn ON/OFF using phone application.	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
2	Turn ON/OFF using the verification system.	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
3	Trigger it with the PIR.	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
4	Trigger it with the magnetic switch.	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
5	Trigger it with the vibration sensor.	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
6	Turn OFF the alarm with smartphone after being triggered.	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
7	Turn OFF the alarm with verification system after being triggered.	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
8	Verify the RGB LED color matches the color for each status.	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
9	Verify Phone Notification when activated.	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

Figure 16.2.4 – Alarm System Tests

The next tested system was the Verification system. We performed 4 tests on this system to get a better understanding of the system performance. The performed tests are presented in Figure 16.2.5.

Test		1	2	3	4	5	6	7	8	9	10
Verification System											
1	Use the tag with the correct user and check verification.	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
2	Use the tag with the incorrect user and check verification.	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
3	Enter the right password and check verification.	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
4	Enter the wrong password and check verification.	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

Figure 16.2.5 – Verification System Tests

The next tested system was the Surveillance system. We performed 4 tests on this system to get a better understanding of the system performance. The performed tests are presented in Figure 16.2.6. As it can be seen, the video download to the smartphone was not possible. If the user wants to download the recorded video he needs to download it to a computer.

Test	1	2	3	4	5	6	7	8	9	10
Surveillance System										
1 Verify the live stream of the video.	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
2 Verify the video download to computer.	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
3 Verify the video download to smartphone.	N	N	N	N	N	N	N	N	N	N
4 Verify that the video works under dark conditions.	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
5 Verify the video recording starts when the alarm is activated.	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

Figure 16.2.6 – Surveillance System Tests

The next tested system was the IR system. We performed 5 tests on this system to get a better understanding of the system performance. The performed tests are presented in Figure 16.2.6. The tests to change the channel up and down were not successful because the TV needs an antenna in order to detect that there is an available channel, otherwise it does not allows to change the channel.

Test	1	2	3	4	5	6	7	8	9	10
IR System										
1 Verify the IR control using the serial commands.	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
2 Verify the IR control using the phone.	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
3 Verify all the needed buttons for the TV control.	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
4 Verify all the needed buttons for the fan control.	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

5	Verify the IR system with the repeater.	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
---	---	---	---	---	---	---	---	---	---	---	---

Figure 16.2.7 – IR System Tests

The next tested system was the Database Management system. We performed 6 tests on this system to get a better understanding of the system performance. The performed tests are presented in Figure 16.2.8.

Test		1	2	3	4	5	6	7	8	9	10
Database Management System											
1	Verify the INSERT commands using the terminal.	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
2	Verify the SELECT commands using the terminal.	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
3	Verify the INSERT commands execute automatically.	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
4	Verify the SELECT commands execute automatically.	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
5	Verify the DBMS works with each of the sub-systems individually.	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
6	Verify the DBMS works with multiprocessing.	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

Figure 16.2.8 – DBMS System Tests

The next tested system was the Battery Backup system. We performed 2 tests on this system to get a better understanding of the system performance. The performed tests are presented in Figure 16.2.9.

Test		1	2	3	4	5	6	7	8	9	10
Battery Backup System											
1	Verify the battery charges while the components are working.	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
2	Verify the system works while there is no connection to the grid.	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

Figure 16.2.9 – Battery Backup System Tests

The next tested system was the Phone App. We performed 4 tests on this system to get a better understanding of the system performance. The performed tests are presented in Figure 16.2.10.

Test	1	2	3	4	5	6	7	8	9	10
Phone App										
1 Test the user verification.	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
2 Test the energy monitor graphical presentation.	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
3 Test the surveillance camera live stream connection.	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
4 Test the surveillance camera download.	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

Figure 16.2.10 – Battery Backup System Tests

After the testing phase was completed, 98% of the tests were successful. The only problem that could not be fixed was to download the surveillance videos to the Smartphone, but these could be downloaded to a computer.

The primary purpose of the testing phase is to determine whether the RMF Intelligent Home system is ready for implementation, and RMF Intelligent Home testing phase proved the system's reliability. After all the positive results, the RMF Intelligent Home system was ready to be presented.

Chapter 17: Presentation Concept Design

17.1 Showroom Design

17.1.1 Introduction to the Concept

To demonstrate all the previously discussed applications, RMF Intelligent Home will design, construct and present a basic showroom structure model. The showroom concept will present all the applications created and constructed to make a smart and unique household. The complete structure will measure five feet (5ft) tall, four feet (4ft) wide and one feet (1ft) long and will be divided in three portions. The two portions in the middle will be divided in half to make room for all the systems. Here we will demonstrate all the sensors and system explained up to this point. The showroom concept will give a good idea how the system will work when interconnected completely and how it will bring a smart and unique household that stand above others.

17.1.2 Organization and Distribution

Starting from the bottom, the left division will be dedicated to the Energy Monitor and the Lightning System. The right side will only be dedicated to the Battery Back-Up System. On the second division of the structure the left side will be dedicated to the Data Base and IR sensor, and the right side to the Network (Router and Modem) and the Magnetic Sensor located on the door. Both these divisions will have one feet (1ft) of separation between each floor and six inches (6in) of separation in the middle so both systems have plenty of space to work with.

The third division will have the remaining three feet (3ft) and will be composed of two panels, which are the NFC and Alarm Panels, the Television, the IR sensor, the Camera, two

Light Bulbs, two switches and two receptacles. The NFC Panel will be used to identify the registered User and will be safely sealed in an enclosure for security reasons. The enclosures measurements are six inches (6in) tall, eight inches (8in) wide and one inch (1in) long. The Alarm Panel will have the same measurements as the NFC Panel and the same enclosure for security reasons also. All the components for the alarm will be placed inside the enclosure, but will be made visible in order to demonstrate how this system works and how it is connected. Just like the previous application, the Camera and Light Bulb will also be located on the third division and will be controlled using the smartphone Application.

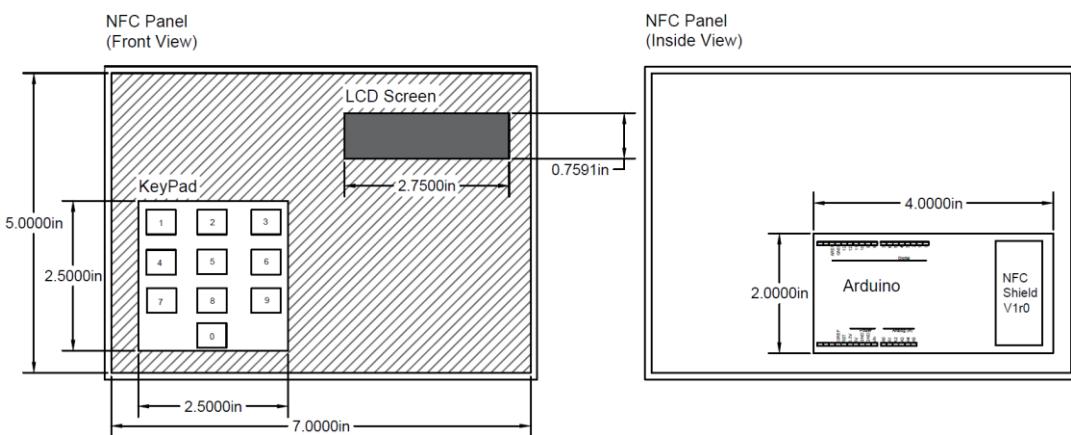


Figure 17.1.1 – Verification System Panel

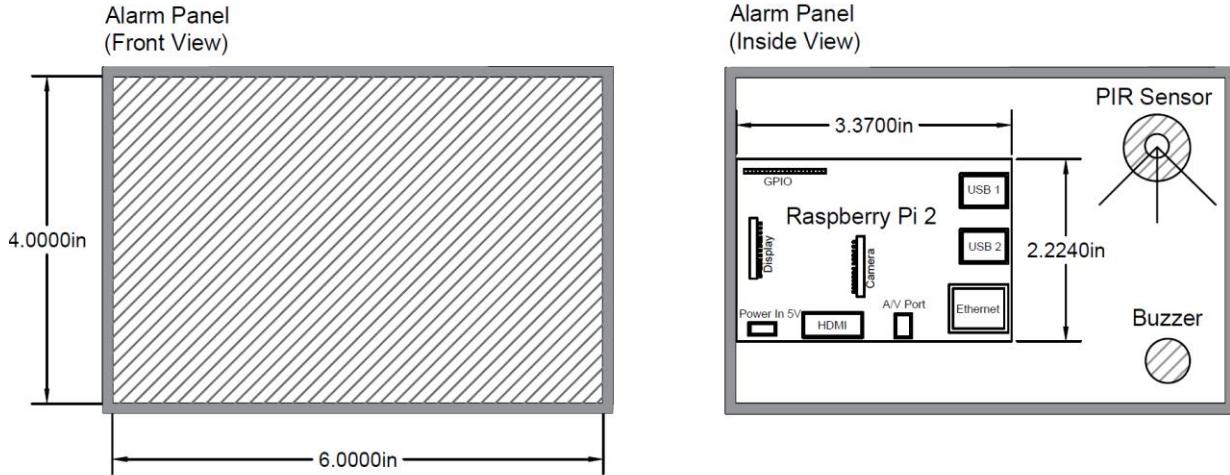


Figure 17.1.2 – Alarm System Panel

Like previously mentioned, the second division will be composed of the Data Base and IR Sensor on the left side and the Network and the Magnetic Sensor on the right side. Cables from the IR Sensor will be distributed along the module to control the Television and the electric fan using the smartphone Application. Since the right side of the second floor is where the Network system is located, a door with a Magnetic Switch was constructed to provide extra security. Almost all connections arrive to the left side of the bottom division where they are connected to the Energy Monitor. On the other side the Battery Back-Up System will be located along with its Battery Bank, Battery Charger and the 400 Watts DC-to-AC Inverter. The Battery Back-Up System will be connected to the Energy Monitor using a multi-plug and all other desired electronic components. Network distribution and connections all around the showroom concept where made

using CAT5E Network Cable. The CAT5E Network Cable will help the different applications communicate with each other.

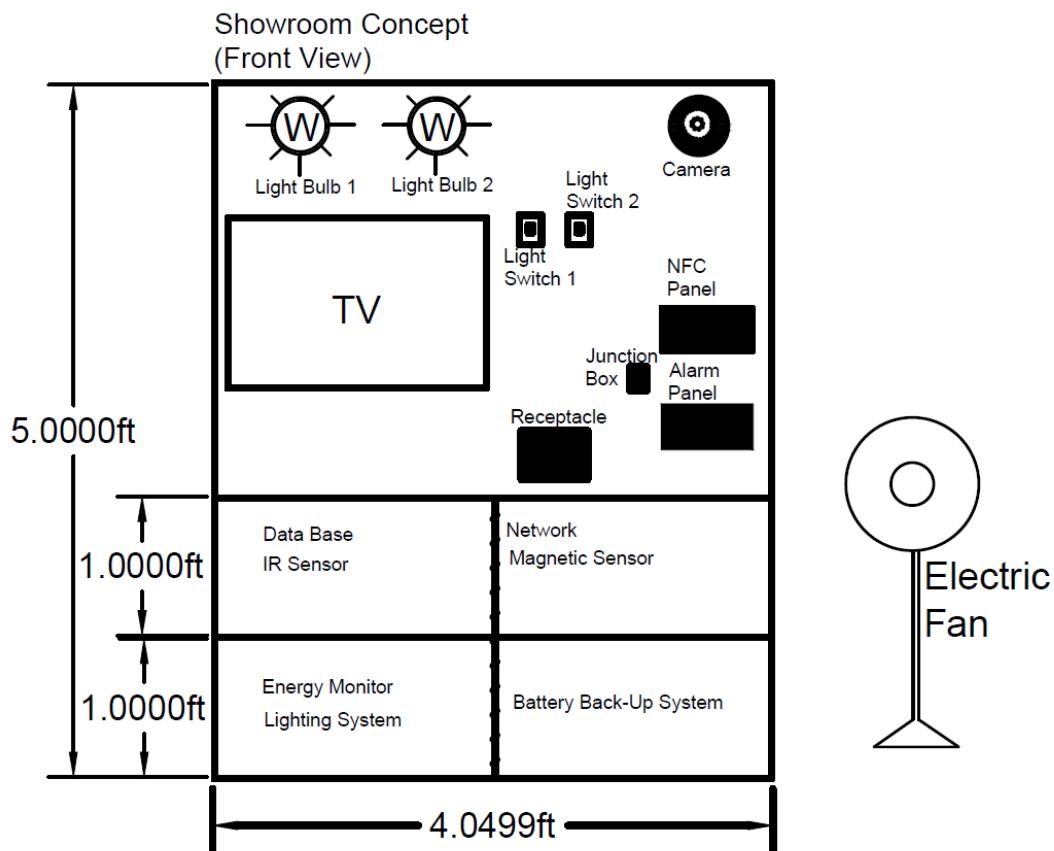


Figure 17.1.3 – Presentation Table Diagram (Front View)

Showroom Concept
(Back View)

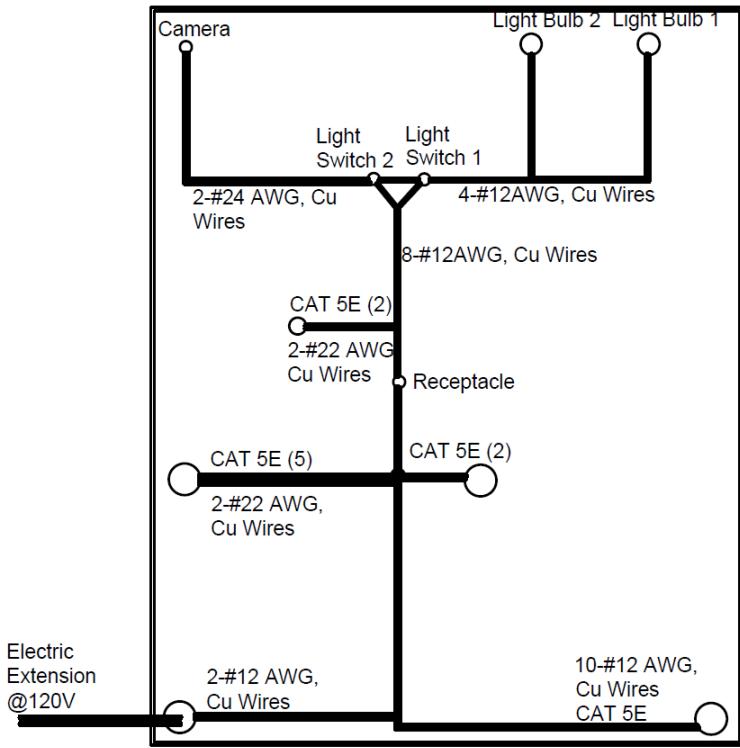


Figure 17.1.4 – Presentation Table Diagram (Back View)

On the third division there will also be two Light Switches available to control de Light Bulbs power manually, and a Receptacle to connect all the electronic devices in that portion. The Light Switches will be located on the middle part of the third floor each one connecting with a Light Bulb separately. This will help demonstrate the Lightning Control System using both manual application with the Light Switches or Remote Control using the smartphone application. On the back of the Showroom Concept we will have all the electric connections to make it as esthetically

as possible. For the distribution of power we will use electric wires of #12 AWG, THHN, Cu (Black, white and Green) covert in Heat Shrink and distributed throughout the showroom concept.

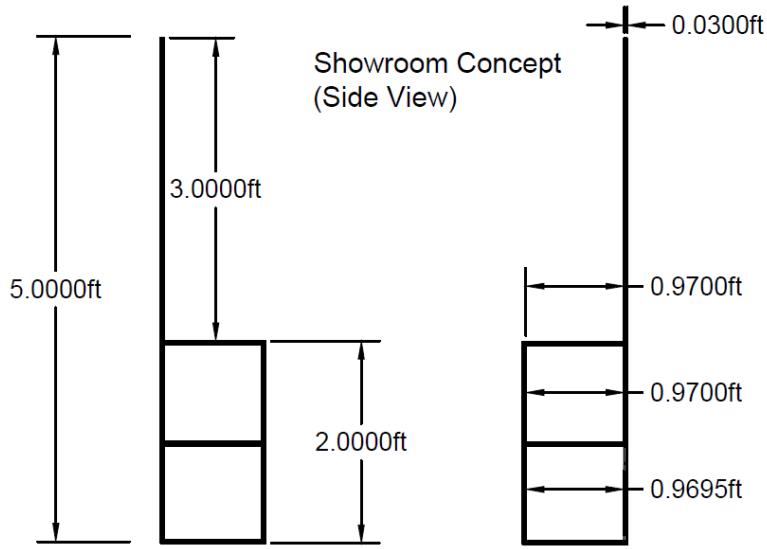


Figure 17.1.5 – Presentation Table Diagram (Side Views)

17.1.3 Construction Materials and Component Enclosures

The construction material used for the showroom model were two Plywood panels. These panels were cut based on the measurements from the AutoCAD model. Each panel was joint together using wood panel bolts and an Electric Drill. Each Plywood panel measured four feet (4ft) wide, eight feet (8ft) tall and three quarters of an inch (3/4 in) long. The IR Sensor enclosure, the NFC and Alarm Panels enclosure where made from plastic. Both the NFC panel and the IR Sensor enclosures are five inches (5in) tall, seven inches (7in) wide, and three inches (3in) long.

The Alarm Panel enclosure is four inches (4in) tall, six inches (6in) wide, and two inches (2in) long. Finally the Energy Monitor enclosure measures six and three quarters of an inch (6 3/4 in) tall, thirteen and three eighths of an inch (13 3/8 in) wide and nine and eleven and sixteen inches (9 11/16 in) long. On like the enclosures of the IR sensor, the NFC and Alarm panel, the Energy Monitor enclosure is made from Plywood.

Chapter 18: References

18.1 Internet of Things References

- daCosta, F. (2013). Rethinking the Internet of Things – A Scalable Approach to Connecting Everything. New York, USA: Apress Open.
- Doukas, C. (2012). *Building Internet of Things with the Arduino*. North Charleston, USA: CreateSpace.
- Igoe, T. (2007). *Making Things Talk* (1st ed.). California, USA: O'Reilly Media.
- Norris, D. (2015). Internet of Things – Do It Yourself at Home Projects for Arduino, Raspberry Pi, and BeagleBone Black. New York: Mc Graw Hill Education.
- Janani. (2013). “*I am still seeing too many UIs being designed and developed by developers and that has to stop...*”. Retrieved from:
<http://internetofthings.electronicsforu.com/2013/09/jayrajugarkar-infosys/>.
- Reiter, G. (2014). Wireless connectivity for the Internet of Things – One size does not fit all. Texas, USA: Texas Instrument Incorporated.
- Frenzel, L. (2015). *What's the Difference Between the OSI Seven Layer Network Model and TCP/IP?*. Retrieved from: <http://electronicdesign.com/what-s-difference-between/what-s-difference-between-osi-seven-layer-network-model-and-tcpip>

- Rowe, C. (2015). *Types of Networks: LAN, WAN, WLAN, MAN, SAN, PAN, EPN & VPN*. Retrieved from: <http://study.com/academy/lesson/types-of-networks-lan-wan-wlan-man-san-pan-epn-vpn.html>
- McEwen, A. & Cassimally, H. (2014). *Designing the Internet of Things*. Chichester, UK: John Wiley & Sons Ltd.
- Beal, V. (2015). *What are Network Topologies? A study guide*. Retrieved from: http://www.webopedia.com/quick_ref/topologies.asp
- Mitchell, B. (2015). *Network Topologies*. Retrieved from: <http://comppnetworking.about.com/od/networkdesign/a/topologies.htm>
- Sparrow, P. (2015). *Network Topology and Types of Network Topologies*. Retrieved from: <http://www.iananswer4u.com/2011/05/network-topology-types-of-network.html>
- Rouse, M. (2015). *Ethernet*. Retrieved from: <http://searchnetworking.techtarget.com/definition/Ethernet>
- Davis, Z. (2015). *Definition of Ethernet*. Retrieved from: <http://www.pcmag.com/encyclopedia/term/42781/Ethernet>
- Valerio, P. (2014). *Sub-1GHz Wireless: The Low-Power WiFi Solution*. Retrieved from: <http://www.networkcomputing.com/wireless-infrastructure/sub-1ghz-wireless-the-low-power-wifi-solution/a/d-id/1315850>

18.2 Phone Applications References

- Kula, P. (2014). Raspberry Pi server essentials transform your Raspberry Pi into a server for hosting websites, games, or even your Bitcoin network. Birmingham, UK: Packt Pub.
- Android Operating system (n.d.). Retrieved March 15, 2015 from:
http://en.wikipedia.org/wiki/Android_%28operating_system%29
- I-phone/I-pad operating system (n.d.). Retrieved March 15, 2015 from:
<http://en.wikipedia.org/wiki/IOS>
- Java Programming language (n.d.). Retrieved March 20, 2015 from:
http://en.wikipedia.org/wiki/Java_%28programming_language%29#Versions
- Java and Android differences (n.d.). Retrieved March 20, 2015 from:
http://en.wikipedia.org/wiki/Comparison_of_Java_and_Android_API
- Integrated Development environment (n.d.). Retrieved March 20, 2015 from:
<http://www.abbreviations.com/IDE>
- Database. (n.d.). Retrieved March 24, 2015, from: <http://en.wikipedia.org/wiki/Database>
- About SQLite. (n.d.). Retrieved March 24, 2015, from: <https://www.sqlite.org/about.html>
- Query language. (n.d.). Retrieved March 26, 2015, from:
http://en.wikipedia.org/wiki/Query_language

- Data Mining. (n.d.). Retrieved March 25, 2015, from:
http://en.wikipedia.org/wiki/Data_mining#Data_mining
- IBM Peterlee Relational Test Vehicle (PRTV). (n.d.). Retrieved March 29, 2015, from:
[http://en.wikipedia.org/wiki/IBM_Peterlee_Relational_Test_Vehicle_\(PRTV\)](http://en.wikipedia.org/wiki/IBM_Peterlee_Relational_Test_Vehicle_(PRTV))
- Cheminformatics. (n.d.). Retrieved March 29, 2015, from:
<http://en.wikipedia.org/wiki/Cheminformatics>
- Database server. (n.d.). Retrieved March 29, 2015, from:
http://www.webopedia.com/TERM/D/database_server.html
- Cloud database. (n.d.). Retrieved March 29, 2015, from:
http://www.webopedia.com/TERM/C/cloud_database.html
- Syncing to the Cloud. (n.d.). Retrieved March 29, 2015, from:
<http://developer.android.com/training/cloudsync/index.html>
- Cloud Database and Android application. (n.d.). Retrieved March 29, 2015, from:
<http://stackoverflow.com/questions/17252239/cloud-database-and-android-application>
- Network. (n.d.). Retrieved March 29, 2015, from: <http://www.webopedia.com/TERM/N/network.html>

18.3 Databases & Cloud Computing References

- Doukas, C. (2012). *Building Internet of Things with the Arduino*. North Charleston, USA: CreateSpace.
- Rouse, M. (2015). Cloud Computing. Retrieved from:
<http://searchcloudcomputing.techtarget.com/definition/cloud-computing>
- Strickland, J. (2015). *How Cloud Computing Works*. Retrieved from:
<http://computer.howstuffworks.com/cloud-computing/cloud-computing.htm>
- Beal, V. (2015). *Cloud Computing (The Cloud)*. Retrieved from:
http://www.webopedia.com/TERM/C/cloud_computing.html
- Wikipedia. (2015). Relational Database. Retrieved from: https://en.wikipedia.org/wiki/Relational_database
- Rouse, M. (2015). Relational Database Definition. Retrieved from:
<http://searchsqlserver.techtarget.com/definition/relational-database>
- Rouse, M. (2015). DBMS Definition. Retrieved from:
<http://searchsqlserver.techtarget.com/definition/database-management-system>
- Serra, J. (2015). Relational Databases vs Non-relational Databases. Retrieved from:
<http://www.jamesserra.com/archive/2015/08/relational-databases-vs-non-relational-databases/>

18.4 Microcontroller Platforms References

- Coley, Gerald (May 22, 2014). *BeagleBone Black System Reference Manual*. Texas-Beagleboard
- Alasdair, Alan (April 15, 2013). *Arduino Uno vs. BeagleBone vs. Raspberry Pi*. Retrieved from: <http://makezine.com/2013/04/15/arduino-uno-vs-beaglebone-vs-raspberry-pi/>
- *Arduino Mega 2560* (n.d.). From p.p. 1-6. Retrieved from:
<http://www.microelectronicos.com/datasheets/ArduinoMega2560.pdf>
- Mosaic Industries (March 29, 2009). *GPIO Electrical Specifications*. Retrieved from: <http://www.mosaic-industries.com/embedded-systems/microcontroller-projects/raspberry-pi/gpio-pin-electrical-specifications>

18.5 Sensors References

- Pololu Corporation (2001-2005). Analog Distance Sensor. Retrieved from:
<https://www.pololu.com/product/136/specs>
- Pololu Corporation (2001-2005). Ultrasonic Sensor. Retrieved from:
<https://www.pololu.com/product/1605>
- Trossen Robotics (2015). Lidar Lite Sensor. Retrieved from:
<http://www.trossenrobotics.com/lidar-lite>
- Electrodragon (2013). PIR Motion Sensor. Retrieved from:
<http://www.electrodragon.com/product/pir/>

- Satistronics Store (2008-2013). Shock Sensor Module. Retrieved from:
http://www.satistronics.com/arduino-shock-sensor-module-801s_p3075.html
- Embedded-Lab (2015). Current Sensor Module. Retrieved from:
<http://embedded-lab.com/blog/?p=4469>
- Elecfreaks (2015). Non-invasive AC current sensor. Retrieved from:
<http://www.elecfreaks.com/store/octopus-noninvasive-ac-current-sensor-ta12100-brick-p-647.html>
- SparkFun Electronics (2009-2015). Infrared Thermometer. Retrieved from:
<https://www.sparkfun.com/products/9570>
- Manualsdir (2012–2015). Precision Centigrade Temperature Sensors. Retrieved from:
<http://www.manualsdir.com/manuals/282103/rainbow-electronics-lm35.html>
- HAOYU Electronics (2009-2015). Water Level Sensor. Retrieved from:
<http://www.hotmcu.com/water-level-sensor-liquid-water-droplet-depth-detection-p-113.html>
- SainSmart (2010-2015). Gas Sensor Module. Retrieved from:
<http://www.sainsmart.com/sainsmart-mq2-gas-sensor-module-for-lpg-propane-hydrogen-arduino-compatible.html>
- Adafruit (2015). Temperature Humidity sensor. Retrieved from:
<http://www.adafruit.com/product/386>

18.6 Near Field Communications & RFID References

- Igoe, T., Coleman, D. & Jepson, B. (2014). Beginning NFC – Near Field Communication with Arduino, Android & Phonegap. California, USA: O'REILLY.
- Coskun, V., Ozdenizci, B. & Ok, K. (2013). *NFC Application Development for Android*. United Kingdom: John Wiley & Sons Ltd.
- Subtil, V. (2014). *Near Field Communication with Android Cookbook*. United Kingdom: Packt Publishing.
- Tiedemann, S. (2014). *NFC Data Exchange Format (NDEF)*. Retrieved from:
<http://nfcpy.readthedocs.org/en/latest/topics/ndef.html>
- Kulkarni, B. (2012). *NFC Data Exchange Format (NDEF)*. Retrieved from:
<http://ibadrinath.blogspot.com/2012/07/nfc-data-exchange-format-ndef.html>

18.7 LED Illumination References

- Zheludev N. (2007). "The Life and Times of the LED-A 100-Year History". *The Optoelectronics Research Center, University of Southampton*. Retrieved from http://holly.orc.soton.ac.uk/fileadmin/downloads/100_years_of_optoelectronics_2.pdf
- Whelan M. (2013). Small Lights with Big Potential: Light Emitting Diodes & Organic Light Emitting Diodes. Retrieved from <http://edisontechcenter.org/LED.html>
- Bausch J. (2011). *The Long History of Light-Emitting Diodes*. Retrieved from http://www.electronicproducts.com/Optoelectronics/LEDs/The_long_history_of_light-emitting_diodes.aspx
- (2008). *LED Thermal Management*. Retrieved from <http://www.lunaraccents.com/educational-LED-thermal-management.html>
- Mills P. M. (2008). *The LED Illumination Revolution*. Retrieved from http://www.forbes.com/2008/02/27/incandescent-led-cfl-pf-guru_in_mm_0227energy_inl.html
- Khan Nisa M. (2014). *Understanding LED Illumination*. Retrieved from <https://books.google.com.pr/books?id=MDrSBQAAQBAJ&pg=PA232&lpg=PA232&dq=understanding+led+illumination+pdf&source=bl&ots=tyk9U2aHBa&sig=KpTgtvEwKETrHjX2U6rEAWQDmSs&hl=en&sa=X&ei=9ZYKVYya4H4jngwTfmYT4Dw&ved=0CDgQ6AEwBA#v=onepage&q=understanding%20led%20illumination%20pdf&f=false>
- (2010). *Adding Intelligence to Lighting Application*. Retrieved from www.microchip.com/lighting
- (2011). *Understanding LED Technology*. Retrieved from greenbuild2011_mccullough_ssl101.pdf

- Taylor A. (2000). *Illumination Fundamentals*. Retrieved from illuminationfund.pdf
- Sanderson Walsh S., Simmons L. K. (2013). Light Emitting Diodes and the Lightning Revolution: The Emergence of a Solid-State Lightning Industry. Retrieved from industryEmergenceSSL.pdf
- (2014). *LED Luminaire Design Guide*. Retrieved from LED_Luminaire_Design_Guide.pdf
- Blitzer F. W. (2003). *Light in Design*. Retrieved from lightindesign.pdf
- (2015). *The History of LED*. Retrieved from http://www.osram.com/osram_com/news-and-knowledge/led-home/professional-knowledge/led-basics/led-history/index.jsp
- Lenk R., Lenk C. (2011). *Practical Lighting Design with LED's*. Retrieved from https://books.google.com.pr/books?id=KGTJKihXvuoC&pg=PT62&lpg=PT62&dq=practical+lighting+design+with+leds+pdf&source=bl&ots=B_SQ7sLwRd&sig=rHVIyUZXuIKMO2eGAuql7q13yWc&hl=en&sa=X&ei=fTsUVdOJG8mbgwTFyoHgBg&ved=0CDQQ6AEwBQ%20-%20v=onepage&q=practical%20lighting%20design%20with%20leds%20pdf&f=false#v=snipet&q=practical%20lighting%20design%20with%20leds%20pdf&f=false

- Khanna Kumar V. (2014). *Fundamentals of Solid-State Lighting*. Retrieved from
<https://books.google.com.pr/books?id=W4PNBQAAQBAJ&pg=PA394&lpg=PA394&dq=practical+lighting+design+with+leds+pdf+download&source=bl&ots=KUnClz21sB&sig=Eki71UpiNZmfhP9yKpmNYkiPL1o&hl=en&sa=X&ei=vHIUVZOGMleqgwSczIHICg&ved=0CDsQ6AEwBQ%20-%20v=onepage&q&f=false#v=onepage&q&f=false>

18.8 Direct Current Systems References

- (2015). *Power-to-Weight Ratio*. Retrieved from http://en.wikipedia.org/wiki/Power-to-weight_ratio
- (2015). *Inrush Current*. Retrieved from http://en.wikipedia.org/wiki/Inrush_current
- (2015). *Lead-Acid Battery*. Retrieved from http://en.wikipedia.org/wiki/Lead%E2%80%93acid_battery
- (2015). *Gaston Planté (1834-1889)*. Retrieved from <http://www.corrosion-doctors.org/Biographies/PlantelBio.htm>
- (2012). *Handbook for Stationary Lead-Acid Batteries*. Retrieved from
<http://www.exide.com/Media/files/Downloads/IndustEuro/Operating%20Instructions/Handbook,%20part%201,%20edition%206,%20Feb,%202012.pdf>
- (2015). *Sealed Lead-Acid Battery Charging Basics*. Retrieved from
<http://www.powerstream.com/SLA.htm#Basics>

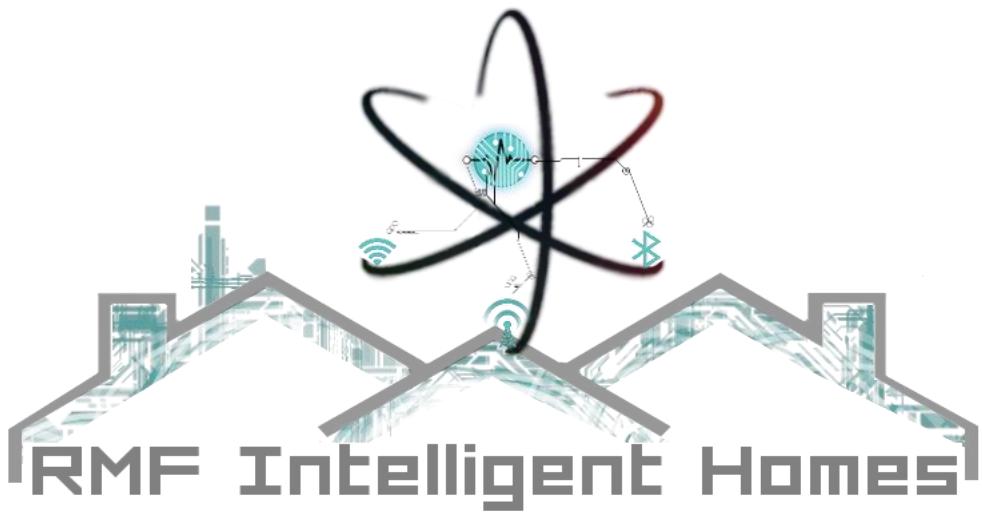
- (2006). *Sealed Lead-Acid Battery Technical Manual*. Retrieved from <http://www.dynamis-batterien.de/pdfs/DYNAMIS-Handling-Lead-Batteries.pdf>
- *History of the Lead Acid Battery*. <http://lead-acid.com/lead-acid-battery-history.shtml>
- U.S. Department of Health and Human Services. (2007). *Toxicological Profile for Lead*. Retrieved from <http://www.atsdr.cdc.gov/toxprofiles/tp13.pdf>
- (2015). *Relative Density*. Retrieved from http://en.wikipedia.org/wiki/Relative_density
- (2015). *Dendrite (Crystal)*. Retrieved from [http://en.wikipedia.org/wiki/Dendrite_\(crystal\)](http://en.wikipedia.org/wiki/Dendrite_(crystal))
- (2015). *Peukert's Law*. Retrieved from http://en.wikipedia.org/wiki/Peukert%27s_law
- (2015). *Battery Pack*. Retrieved from http://en.wikipedia.org/wiki/Battery_pack
- (2015). *Battery Balancer*. Retrieved from http://en.wikipedia.org/wiki/Battery_balancing
- (2015). *Battery Management System*. Retrieved from
http://en.wikipedia.org/wiki/Battery_management_system
- (2015). *Battery Charger*. Retrieved from http://en.wikipedia.org/wiki/Battery_charger
- Brain M. *How Emergency Power Systems Work*. Retrieved from
<http://home.howstuffworks.com/home-improvement/household-safety/security/emergency-power.htm>
- (2015). *Electrochemical Cell*. Retrieved from http://en.wikipedia.org/wiki/Electrochemical_cell
- Bellis M. *History Timeline of the Battery*. Retrieved from
<http://inventors.about.com/od/bstartinventions/a/History-Of-The-Battery.htm>
- Moyal A. *Battery Guidelines*. Retrieved from batteries.pdf
- Hydroelectric Research and Technical Services Group. (1998). *Storage Battery Standards and Techniques*. Retrieved from Storage and Maintanaice.PDF

18.9 IR Controller Design References

- Monk, S. (2012). *Arduino + Android Projects for the Evil Genius*. New York: Mc Graw Hill Education.
- SainSmart. (2013). *Arduino Introduction*. Retrieved from: www.sainsmart.com
- Adafruit. (2012). *Testing an IR Sensor*. Retrieved from: <https://learn.adafruit.com/ir-sensor/testing-an-ir-sensor>

18.10 Network Design References

- Bell, C. (2013). Beginning Sensor Networks with Arduino and Raspberry Pi. New York: Apress.
- Golden, R. (2013). *Raspberry Pi Networking Cookbook*. Birmingham, UK: Packt Publishing.



Chapter 19 Budget

19.1 Sub-Systems Cost

The following tables present detailed budgets for each of the sub-systems.

	Qt	Unit Price	Cost
Lighting System			
Arduino Uno	1	\$20.99	\$20.99
Ethernet Shield	1	\$17.60	\$17.60
Relay Shield	1	\$20.00	\$20.00
Relay (2 Pack)	1	\$5.95	\$5.95
Lighting System Sub-Total			\$64.54

Figure 19.1.1 – Lighting System Budget

	Qt	Unit Price	Cost
Energy Monitor System			
Arduino Uno	1	\$20.99	\$20.99
Ethernet Shield	1	\$17.60	\$17.60
Current Sensor 5A	1	\$2.80	\$0.00
Energy Monitor System Sub-Total			\$38.59

Figure 19.1.2 – Energy Monitor System Budget

	Qt	Unit Price	Cost
IR System			
Arduino Uno	1	\$20.99	\$20.99
Ethernet Shield	1	\$33.17	\$33.17
IR LED	1	\$3.49	\$3.49
Arduino Protoboard	1	\$13.73	\$13.73

IR Repeater	1	\$29.99	\$29.99
Plastic Enclosure 7x5x3	1	\$7.49	\$7.49
IR System Sub-Total			\$108.86

Figure 19.1.3 – Infrared System Budget

	Qt	Unit Price	Cost
Alarm System			
Raspberry Pi 2 Complete Kit	1	\$69.99	\$69.99
Vibration Sensor	1	\$9.95	\$9.95
Passive Infrared Sensor (10 Pack)	1	\$16.99	\$16.99
Buzzer	1	\$1.69	\$1.69
Magnetic Switch		\$5.00	\$0.00
Plastic Enclosure 6x4x2	1	\$6.49	\$6.49
Alarm System Sub-Total			\$105.11

Figure 19.1.4 – Alarm System Budget

	Qt	Unit Price	Cost
Verification System			
Arduino MEGA	1	\$37.03	\$37.03
Ethernet Shield	1	\$33.17	\$33.17
NFC Shield	1	\$26.95	\$26.95
LCD	1	\$4.39	\$4.39
Keypad	1	\$3.69	\$3.69
Mini-breadboard (5 Pack)	1	\$3.49	\$3.49
Potentiometer (10 Pack)	1	\$4.99	\$4.99
Plastic Enclosure 7x5x3	1	\$7.49	\$7.49
Verification System Sub-Total			\$121.20

Figure 19.1.5 – Verification System Budget

	Qt	Unit Price	Cost
Surveillance Camera			
Raspberry Pi 2 Complete Kit	1	\$69.99	\$69.99
Camera 5MP 1080P Night Vision Module	1	\$28.45	\$28.45
IR Night Vision Round 36 LEDs Board	1	\$9.99	\$9.99
Camera Enclosure	1	\$9.29	\$9.29
Surveillance Camera Sub-Total			\$117.72

Figure 19.1.6 – Surveillance System Budget

	Qt	Unit Price	Cost
Database			
Raspberry Pi 2 Complete Kit	1	\$69.99	\$69.99
SanDisk Pen Drive 16 Gb	1	\$5.99	\$5.99
Database Sub-Total			\$75.98

Figure 19.1.7 – Database System Budget

	Qt	Unit Price	Cost
Network System			
Black Box 5 Port 10/100 Ethernet Switch	1	\$29.99	\$29.99
Cat 6 Cable 250' Pull Box	1	\$49.99	\$49.99
RJ-45 Modular Plugs (25 Pack)	1	\$12.19	\$12.19
Network System Sub-Total			\$92.17

Figure 19.1.8 – Network System Budget

	Qt	Unit Price	Cost
Battery Backup System			

Enercell 12V/5AH Lead-Acid Battery	3	\$29.99	\$89.97
Lipro Battery Balance Charger	1	\$35.56	\$35.56
Inverter 400W - 800W Peak	1	\$43.80	\$43.80
Solid Wire UL Hookup 20AWG 15' (3 Pack)	2	\$8.99	\$17.98
Stranded Wire 12AWG	20	\$0.89	\$17.80
Battery Backup System Sub-Total			\$205.11

Figure 19.1.8 – Battery Backup System Budget

Presentation Model	Qt	Unit Price	Cost
Enercell 12V/5AH Lead-Acid Battery	3	\$29.99	\$89.97
Heat Shrinking Tubing 100ft	1	\$18.99	\$18.99
Flat Head Screws #6 x 1/2" (50 Pack)	1	\$5.44	\$5.44
Flat Head Screws #8 x 3/4" (50 Pack)	1	\$5.44	\$5.44
Pan Head Screws #8 x 1/2" (12 Pack)	1	\$1.53	\$1.53
Cables Mounting Base 1x1 (10 Pack)	1	\$2.79	\$2.79
Door Hinges 1-3/4 x 3-1/16	1	\$6.36	\$6.36
Plywood 3/4" 4x8	1	\$33.29	\$33.29
Plywood 3/4" 2x8	1	\$16.65	\$16.65
Paint Kona Brown	2	\$11.12	\$22.24
Plastic Enclosure 5x2.5x2	1	\$5.49	\$5.49
Cable Ties 8" (100 Pack)	1	\$14.43	\$14.43
Multi Outlet	3	\$4.84	\$14.52
Light Bulb	2	\$2.50	\$5.00
Lampholder	2	\$2.50	\$5.00
Light Switch with Metal Housing	2	\$5.00	\$10.00
Presentation Model Sub-Total			\$257.14

Figure 19.1.9 – Presentation Model Budget

19.2 RMF Intelligent Home Total Cost

After detailing the costs of each of the sub-systems, we present the following tables with the budget for the whole system implementation, including the labor costs.

RMF Intelligent Home System	
Lighting System	\$64.54
Energy Monitor System	\$38.59
IR System	\$108.86
Alarm System	\$105.11
Verification System	\$121.20
Surveillance System	\$117.72
Database	\$75.98
Network System	\$92.17
Battery Backup System	\$205.11
Presentation Model	\$257.14
<hr/>	
RMF Intelligent Home System Sub-Total	\$1,186.41

Figure 19.2.1 – RMF Intelligent Home System Budget

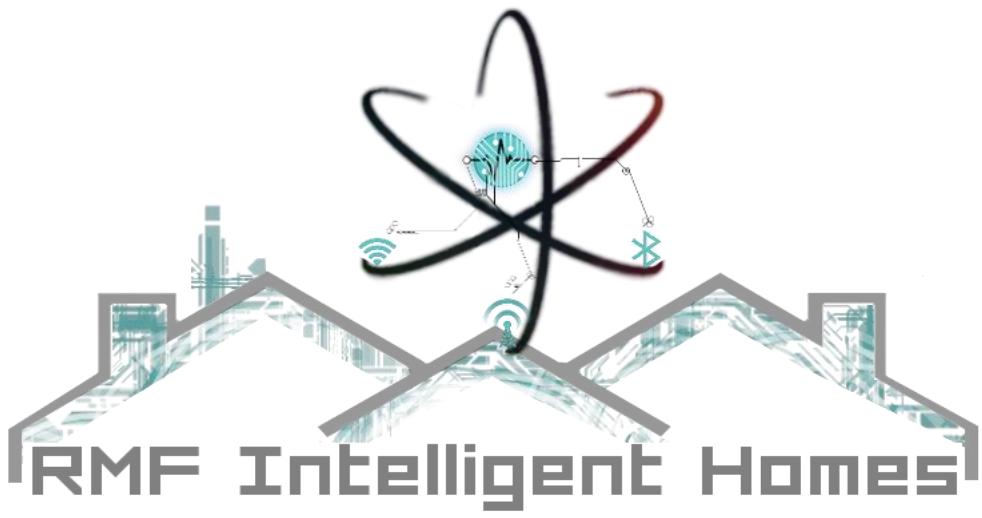
Labor			
	Rate	Hours	Cost
Software Development	\$25.00	968.00	\$24,200.00
System Design	\$25.00	704.00	\$17,600.00

System Wiring	\$25.00	40.00	\$1,000.00
System Installation	\$25.00	176.00	\$4,400.00
RMF Intelligent Home Labor Sub-Total			\$47,200.00

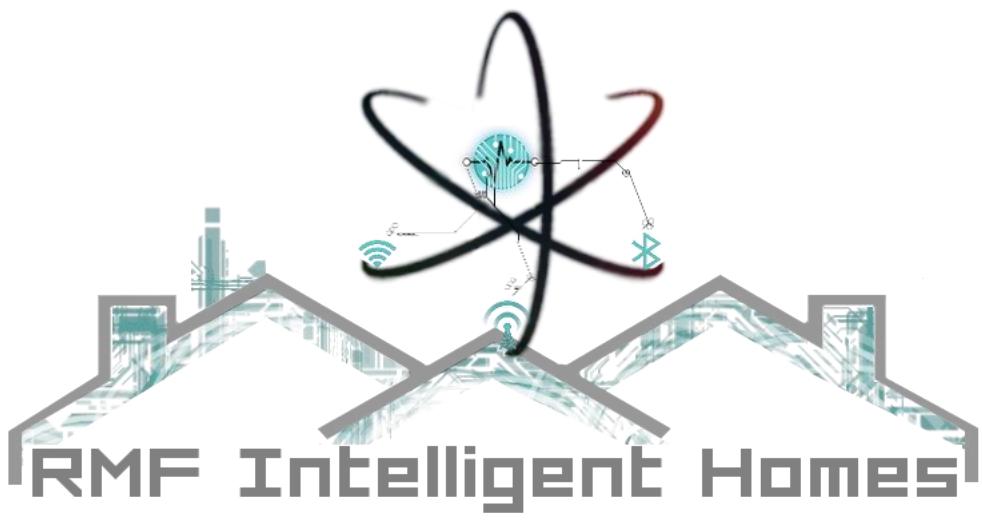
Figure 19.2.2 – RMF Intelligent Home Labor Budget

Total Costs	
System Costs	\$1,186.41
Labor Costs	\$47,200.00
RMF Intelligent Home System Sub-Total	\$48,386.41

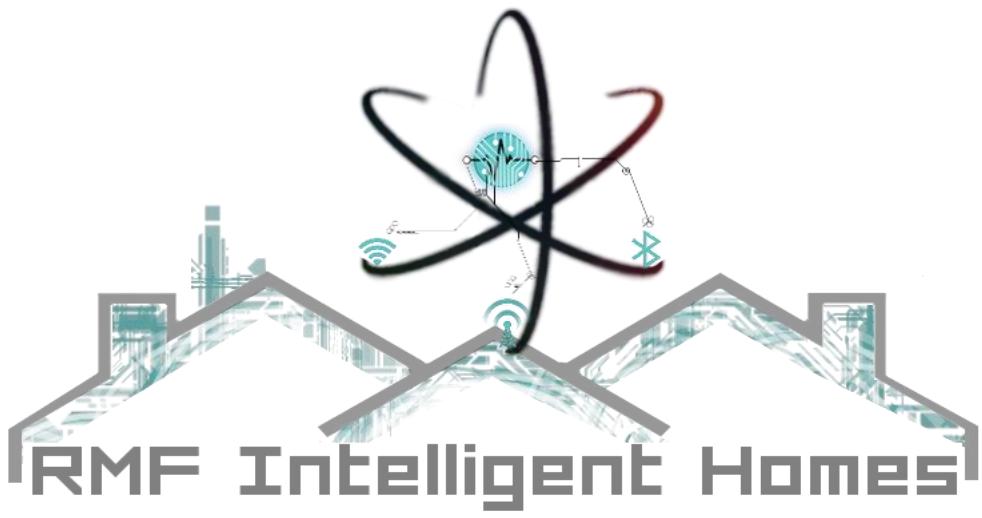
Figure 19.2.3 – RMF Intelligent Home Total Costs



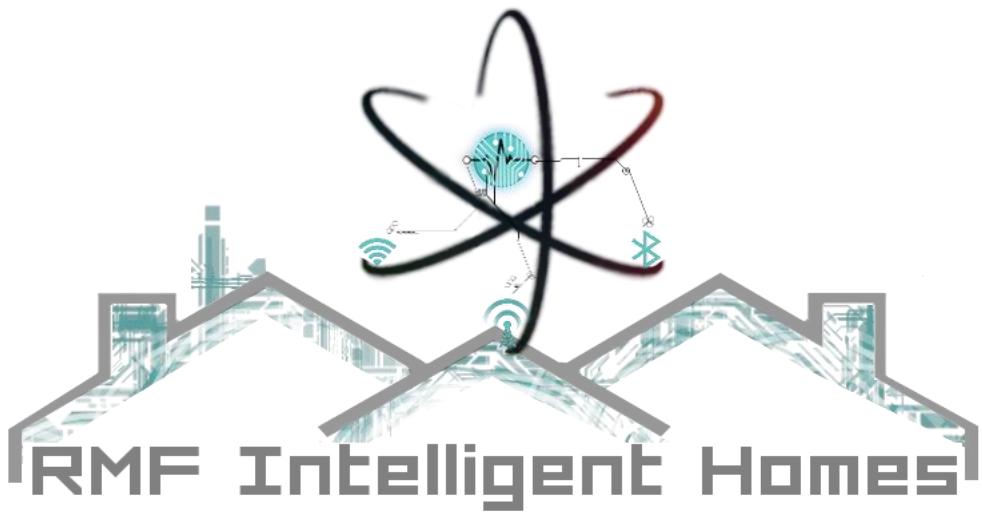
Chapter 20 Schedule



Chapter 21 IEEE Article



Chapter 22 Meeting Minutes



Chapter 23 User Manual

Chapter 23: RMF Operation Manual

23.1 Phone App

23.1.1 Phone App User Verification

The phone app user verification is presented in Figure 23.1.1. It expects the user to enter its username and password in order to verify the credentials. After verifying the credentials, the user is able to access all the other systems and control other appliances at the house.



Figure 23.1.1 – Phone App User Verification Interface

23.2 Alarm System

23.2.1 Verification System Operation

The verification system is composed of two main components, which are presented in Figure 23.2.1 and Figure 23.2.2. The selected tag is a very practical because a keychain holds it, so it makes a great addition to the user's key ring. Holding the tag in a keychain prevents the user from losing it. By all means, the purpose of the verification system is to provide the user with an easy and safe way of deactivating the alarm without the need of the phone application.



Figure 23.2.1 – NFC Verification System Reader



Figure 23.2.2 – NFC Mifare Classic Tag

The procedure for the user verification system is very simple and is explained as follows:

1. Pass the tag by the verification system reader and wait for the system to validate the username, which is held in the tag memory.
2. After the user is validated, the reader will ask for the 4-digit password.
3. The user must enter its password and wait for the system to validate it.
4. After the reader validates the password it will indicate that the alarm status has changed. To verify this, the user can check the alarm LED that must have changed its color.
5. If the username (tag) or the password has not been verified, the system will indicate it and returns to the normal status of waiting for the next tag.
6. After user verification the state of the alarm system will change in one of three ways:
 - a. Green: This means that the system is armed and waiting for an intrusion or any security problem to happen.
 - b. Blue: This means the system is disarmed and the user can safely enter the household or the building.
 - c. Red: The system is telling you that the alarm has been tripped and that an intrusion has been detected. **Note: If the system was just disarmed after an intrusion,**

rapid arming of the alarm will cause the state of the alarm not to change because of the stability period of the sensors.



Figure 23.2.3 – Alarm System

23.2.2 Alarm System Operation Through Phone App

The alarm system can be armed or disarm through the phone app. After the user verifies his credentials, there is an interface in the app that is dedicated to the alarm system. This interface gives the ability to the user of arming and disarming the alarm system. The interface is presented in Figure 23.2.4. The operation procedure for the user alarm interface is explained as follows:

1. After submitting the user credentials on the RMF mobile application, the user has the opportunity to choose between the different systems throughout the house. To interact with the alarm, the user must select the alarm system from the list.

2. A button with the actual state of the alarm system will appear, press button to change state of the system.
3. If the alarm system has been triggered, the user will be prompted for credentials in order to monitor the surveillance system of the house.



Figure 23.2.4 – Alarm System Interface

23.2.3 Surveillance System

The alarm system and the surveillance system are closely related. The alarm system activates the video recording in the surveillance system as soon as it is triggered. The user can also activate the video recording manually through the interface presented in Figure 23.3.5. The working procedure for the surveillance system is explained as follows:

1. If the user wants to monitor the house via the surveillance camera system, he can do so by going through the applications several systems and selecting the button with the camera logo to view the camera on each room.

2. The user will be prompted for credentials for each use of the system to maintain a strict security between systems.
3. After selecting the camera on the system of preference, the user may go through the camera's log or download a video from this log and for storage. The user may also go to "local-ip:80/download.php" to download the video from any web browser.
4. In case of an intrusion on the alarm system, the user will be prompted to re-enter credentials as shown on figure 23.2.2.
5. After credentials are verified, the system will show the corresponding room on which the intrusion was detected.

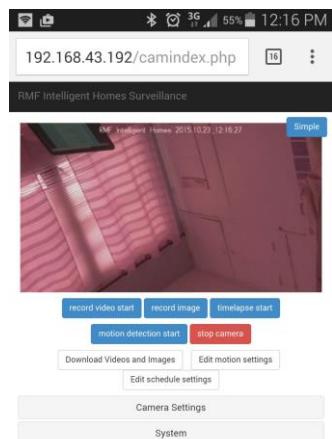


Figure 23.2.5 – Surveillance System Interface

23.3 IR System

23.3.1 IR Control System Operation

The RMF Intelligent Home IR system is prepared to control different devices in the house that send commands using infrared. The system is prepared to control a television using the interface presented in Figure 23.2.1. The interface is very simple and it is easy to understand because it resembles a common TV control.

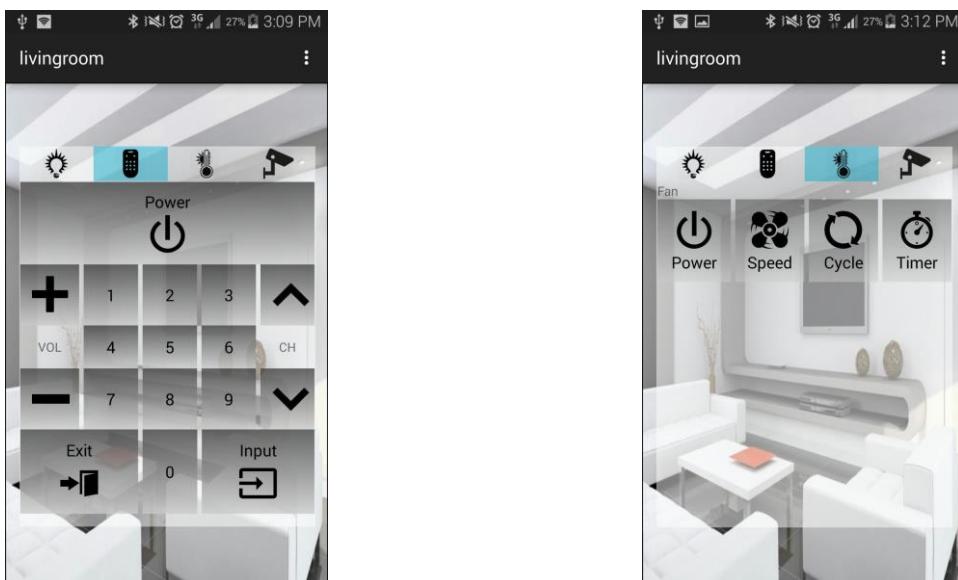


Figure 23.3.1 – TV and Fan IR Control Interfaces

Another devices that can be controlled are the air conditioners and fans that work with infrared controls. The interface to control the IR-controlled fan is also presented in Figure 23.2.1. This interface is very simple and it resembles the same buttons that are in the fan. It has buttons to turn ON and OFF the fan, change the speed, and activate the timer

23.4 Lighting System

23.4.1 Lighting System Operation

The RMF Intelligent Home lighting control system is prepared to control the different lighting fixtures inside the house. The system is very simple since it presents ON and OFF buttons, which are divided by room and locations inside the room. Figure 23.3.1 presents the lighting control interface for the living room in the house with two lighting fixtures.

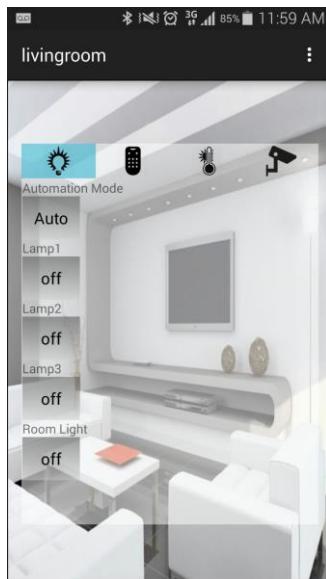
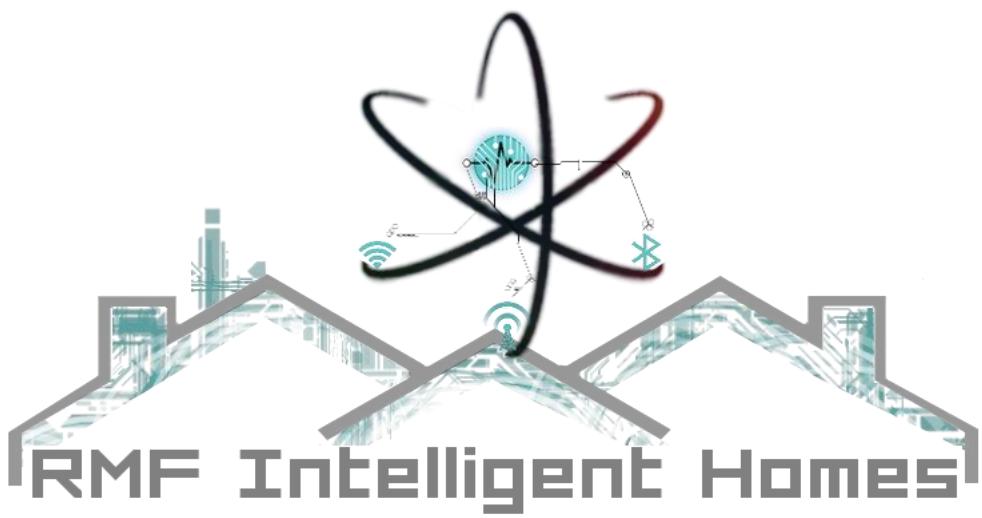


Figure 23.4.1 – Lighting Control Interfaces

The RMF Intelligent Home lighting control system has two operation modes: manual and auto. When the user accesses the phone app, the system turns into auto mode. Auto mode means that it can only be controlled using the phone app, so the house switch will not have any effect. When the user decides to change to manual mode, the phone app stops having an effect on the lighting and it can only be controlled using the house switch.



Chapter 24 Weekly Report