Open Sesame
Final Report

EE 41440: Senior Design II
Dr. R. Michael Schafer
7 May, 2014

Denise Garcia
Ka Hin Lee
Veronica Martinez
Jane McGuinness
Angela Savela

Table of Contents

1 Introduction

Many homeowners across the world are plagued with the common problem of forgetfulness or absent-mindedness. One of the largest issues that forgetful homeowners have to deal with is whether or not they remembered to close the garage door. This concern, however, is not limited to the chronically preoccupied. It can affect anyone who is simply not paying attention or is distracted momentarily thus leading to a wide-open garage door. Forgetting to close the garage door is a mistake that can easily be made when leaving or even entering the home. An unattended, open garage door can lead to many dangers such as burglary, trespassing, or a small child escaping. There are even minor annoyances that can come from an opened garage door such as a pet running away or unwanted weather damage to cars or homes. It is almost impossible to determine whether or not the door was opened or closed once one has left their residence. Lack of control from a distance not only means the door may be left open, but also that it cannot be opened to allow entry to permitted visitors—such as maintenance workers.

Although many people fall victim to a gaping garage door, it does not mean the problem cannot be remedied. The headache of the many will be answered and that answer is "Open Sesame." The Open Sesame system is designed to interface with an existing garage door. All it requires is a garage door opener, tools to mount a set of switches, a stable Wi-Fi connection, and a smart phone, although the commands can also be run from an internet interface. The design retains all of the original capabilities of the garage door opener system and acts as an additional remote that is not limited by range. This unlimited "remote" has several features that provide knowledge about and control over the door. The user can now check the status of the door while anywhere from work to their bedroom so no matter where they are they never have to wonder if they remembered to close it. They can also choose to take a picture of the door to see if all is well. This is particularly useful should the door become stuck in transition. The user will be able to see if something has fallen in the way of the door or if someone has entered to cause this error. Not only can the user stay informed, they also have control. Should they forget to close the door, Open Sesame allows them to close it, even if they are already miles from home. If the door needs to be opened for a repairman or a child returning from school, a button can be pressed from wherever the owner is and access will be granted. Open Sesame even goes beyond fixing the status after the fact. It also incorporates proactive features that close and open the door automatically when the user (and therefore their phone) leave or approach the home and leave or enter the home Wi-Fi. This feature can be turned on and off so that it does not activate the motor when the user is just mowing the lawn or going for a walk. With this addition, it becomes unlikely that the door will be forgotten in an unwanted state to begin with, further enhancing the benefits of the Open Sesame system.

In order to function, the product requires communication from the Android application or internet all the way to the module's microcontroller via a Wi-Fi enabling device called the Electric Imp. All of the commands begin at the user's cell phone, and are sent by accessing URLs set to match the specific Electric Imp. The Imp then communicates with the microcontroller mounted on a circuit board designed for this project. This communication is achieved through SPI commands that cause the microcontroller to access the corresponding case of a switch statement. The microcontroller has been programmed to send commands to the garage door opener in the same fashion as someone using the wall remote, allowing for flexibility and retention of all original safety features. It also controls the light and the camera and can determine the status of the door via a switch system.

Overall, the Open Sesame system design meets most of the expectations originally envisioned when this project first began. Of the four primary features (door status checks, motor control, automatic open and close and picture taking), most are fully operational. Open Sesame was able to control a garage door opener motor from an Android application—or a web browser on a computer or any smartphone—and check whether the garage door was open, closed, or in between. In addition, a command to take a picture can be executed from the cell phone application—or web browser—to the camera connected to the main circuit board through the Electric Imp. The camera can then send the picture through the microcontroller to the Electric Imp device which in turn sends it to its online agent. The cell phone application can successfully grab an image from the server where pictures for the system are to be placed. However, for this prototype, the agent to server communication link is not functional. The ability to identify the home Wi-Fi network and to issue commands based on changes in connectivity with it is also demonstrated by this version of the prototype. It requires time for the commands to send because of the delay caused by the phone disconnecting from one Wi-Fi and connecting to another or the mobile network, but the feature is still functional.

In addition to these primary requirements being mostly met, a variety of additional features were incorporated to increase safety and increase the owner's ease of use. All of the control interfacing was completed without overriding any of the motor's existing safety features, such as the optic eyes. For the user's benefit, the Open Sesame system has the built in the intelligence of detecting when an open/close command has been issued redundantly, unlike the garage motor wall panel, which just toggles between open and close, often resulting in the user accidentally opening/closing the door unintentionally. For extra safety, when a command is executed remotely—be it from the cell phone application or a web browser—the garage motor flashes the light off and on to warn anyone in the vicinity of the garage door that it will be opening or closing. Also, much like the existing motor system, when the door is stopped and detected as in transition, the next action of the motor will be to open the door only, in case whatever was blocking the door is still there.

The Open Sesame team was able to test the functionality of the project through the construction of a mini-garage door model. Access to a full garage door motor and system allowed for confirmation that the product could successfully communicate with and control an existing garage motor and light. By attaching a fake "door" to the motor and installing the switches as they would be in a real garage, the accuracy of the switches used to determine the door status was also confirmed. The automatic open/close feature was tested by disconnecting and connecting the phone on which the application was running to the ND-secure Wi-Fi which stood in for a home Wi-Fi network. This enabled the team to confirm that the motor would eventually respond appropriately to these changes. Unfortunately, the camera subsystem was not able to be fully tested in this setting because of the issues in saving the image to the server. Previous work with the USBee, the logs of the Electric Imp agent, a dummy picture on the server and the application, showed that all of the links in the communication chain were functional except for the server-Imp one.

The main unresolved issue is sending an image from the Electric Imp cloud to the Notre Dame server made for this project, which is most likely due to the network permissions of the server. The Electric Imp can receive the image from the camera through the microcontroller, and the cell phone application can get an image from a server, but there is a disconnect in actually saving the image from the camera to the server. This is hypothesized to be a permissions issue because the Notre Dame servers are very secure and are unlikely to accept uploads from unrecognized and unapproved sources such as the Electric Imp cloud. Other minor things such as the delays in the automatic open/close commands and the time needed to transmit the picture are not ideal, but they do not prevent the functioning of the product as a whole. The length of some of these delays seems to be dependent on the model of phone used, which is unexpected. There are also issues that can arise should the user grow impatient and begin pushing buttons without waiting for the previous command to complete. With additional time and resources, various upgrades to the project could be made before taking it to market and these are discussed in the To-Market Improvements portion of this report.

2 Detailed System Requirements

Enhanced Intelligence

The microcontroller must be able to interface with an existing garage door system, so that signals may be sent from the microcontroller to the main circuit of the garage door opener system. This connection allows for the microcontroller to transmit open/close signals for the door and on/off signals for the light. This is done through a hardwired circuit. It communicates like the wall panel does. There are existing screws on the opener system to allow for the attaching of such wires. This allows the main features of the existing garage door system to remain unchanged. It is also user-friendly because the set up is not more complex or invasive than the installation of the already included wall panel. It could also help the product be more universal and able to work with more garage door opening systems. The microcontroller is also connected to two magnetic switches, which relay the status of the door. With this information obtained, the user can then proceed to change the status of the door if desired.

Another aspect of the additional intelligence is the ability of the user to request and receive photos of the garage door. These need not be high quality as their main purpose is to provide an image that proves the current state of the door and that would allow the user to identify any potential problems or security threats. For example, it would let the user see why the door may have failed to close full or see if someone is entering, should the door open unexpectedly.

Wireless and Hands-free Interface

A wireless router with WPA2 security is used to provide the system with protected access Wi-Fi, which is the typical wireless network protocol for most modern homes. The only device that needs to be supported from the system is the Electric Imp, which is what allows the user to communicate with the system using the Internet. Although the Electric Imp must be within the range of the wireless router providing Wi-Fi, there is no required range for the user to be able to remotely access or control the entire system from the Android application on their phones. In addition to allowing for access to door information and door control from a distance, the ability to recognize the home Wi-Fi network allows for a feature that automatically opens the garage door as the home owner (and their phone) pull within range of the home Wi-Fi.

User Interfacing

There are three main user interfaces: the cell phone application for Androids (or the website), the existing button panel, and the existing remote control devices. The cell phone application is the primary focus, as the latter two interfaces do not need to be changed. The cell phone interface allows the user to communicate with the system, by receiving images of the garage door area and the status of the door, as well as sending the commands to perform these functions and the commands to open or close the door as desired. The user also has the ability to make portions of the interface hands-free with the Wi-Fi detection-enabled automatic open/close feature.

System Installation, Use, and Safety

The system is installed by connecting the microprocessor circuit board to the existing garage door system circuit, by hardwiring it in a manner mimicking the wall panel. The new system should not interfere with existing functions of the original garage door opener system, as it is designed to be as noninvasive as possible by connecting to screws that the owner is already meant to work with. The new system uses external sensors to detect the status of the garage door, so the connection between the new system and the previous system is one way with the microcontroller sending open/close and light on/light off signals to the garage door opener circuit. A primary motivation for making the system as noninvasive as possible is to avoid conflicting with the security and safety features and protocols of the existing garage door opener system.

Power System

The microprocessor circuit board and the Electric Imp connecting circuit board must be powered by some kind of continuous power source. Because the device is installed near the existing motor box on the ceiling of the garage, it can plug into the outlet that typically already exists there for the garage door motor. This voltage must be divided down to provide the 3.3 V and 5 V expected by the electronic components. All of the additional components are connected up to the central board that is connected to this power source. Battery power is not required. Protocol upon an unexpected loss of power or reset must be established.

Table 2.1. Summary of System Requirements

Requirement	Description	Status/Result
Primary Features		
Check Door Status	User can check status of door from app or website; code can make decisions based off this information	Achieved
Motor Control	User can specify open or close door from app or website	Achieved
Light Control	Ability to turn on and off the motor light to take pictures and warn users	Achieved
Picture Capture	User can request and receive picture on app or website	Partial – Imp to server link unachievable
Wi-Fi Recognition	App can identify when connected to home Wi-Fi	Achieved
Approach/Exit Response	When user is in Wi-Fi detection mode, door opens or closes automatically when phone connects or disconnects from home Wi-Fi	Achieved
Existing User Interfaces Retained	Wall panel and remote still function as expected	Achieved
Power		
Power	Ability to be powered from standard outlet; battery power not required	Achieved
System Outage Recovery	Ability for device to recover from power outage	Partial - requires a reset
Safety Features		
Retention of Safety	Optic eyes, force sensors, etc. built into motor still stop door in	Achieved

Features	required situations	
Additional Safety Features	Light flashes to warn of impending door status changes commanded remotely	Achieved
Existing User Interfaces Retained	Wall panel and remote still function as expected	Achieved
General Requirements		
Reasonable Price	Components chosen to minimize cost to the user and stay under the project budget of $500	Achieved
Packaging	Housing to protect from garage elements and allow camera angling; reasonably small; user access to required features; nonflammable	Achieved
Noninvasive Installation	User can install without accessing the internal components of the motor	Achieved
Minimal Installation Requirements	User should only need to install two switches, the module and connect to motor through accessible screws	Achieved
Security	Only user should be able to control their door through the app	Partial – would be achieved through the existing unique Imp URLs

3 Detailed Project Description

3.1 System Theory of Operation

The Open Sesame system is an Android or web interface for your garage door opener. It has been designed to bring an outdated, but functional, product into the age of instant information. Unlike other systems which require replacing the entire garage door opener, the Open Sesame is a means to update and increase an opener's functionality. Additionally, because it is non-invasive all of the original safety features are retained along with the ability to use the wall panel and remotes. The center of the system is the Electric Imp, a product developed by Electric Imp, Inc. Through this chip, the circuit board is interfaced with the internet, making all of the commands possible.

By making minimal adjustments to the existing system, in this case adding switches and mounting a small box, the capabilities are expanded immensely. The system has four main functions: check the status, change the status (open and close), take a picture, and proximity sensing. These commands can all be sent from the user's Android phone, and all but proximity sensing can be performed using an internet interface. For the first three, the user makes a selection by pressing a button on the app. This button activates a URL which sends a message to the Electric Imp. The Electric Imp then sends a signal via SPI to the microcontroller located on the circuit board. Depending on the character received, the microcontroller moves between the cases of a switch statement contained in an infinite while loop. It either sends a signal to the motor or to the camera or checks the switch status, and then passes the relevant information to the Imp. The Imp also provides information back to the user by changing messages at the accessed URL. This text is then displayed on the Android application. The proximity sensor is not so much a sensor as a code to check which wireless network the phone is currently connected to. After the user has assigned a home network, by selecting Wi-Fi Detection mode they can automate the commands sent for leaving and arriving at their home. In order to prevent searching for a remote, or forgetting to close the garage door when you leave, when this mode is turned on the entire open/close process is automated. This does assume that when leaving (disconnecting from home Wi-Fi) the garage door should close automatically, and when arriving at home (reconnecting to home Wi-Fi) it should open automatically. This option can easily be turned off for any reason.

The interactions between the various subsystems and modules that implement this theory of operation are summarized in Figure 3.2.1 below. Almost all of the communication is two-way. The interaction with the motor is not as no information is obtained from the motor system itself and commands are only sent to it. The central circuit board to camera communication is handled via UART and it can determine the status by checking the state of two I/O pins. The two-way communication between the Imp and the microcontroller is SPI with the Imp as the master. Communication between the Imp and the Android app is achieved both through accessing URLs and posting text to them as well as exchanging data over a server.

3.2 System Block Diagram

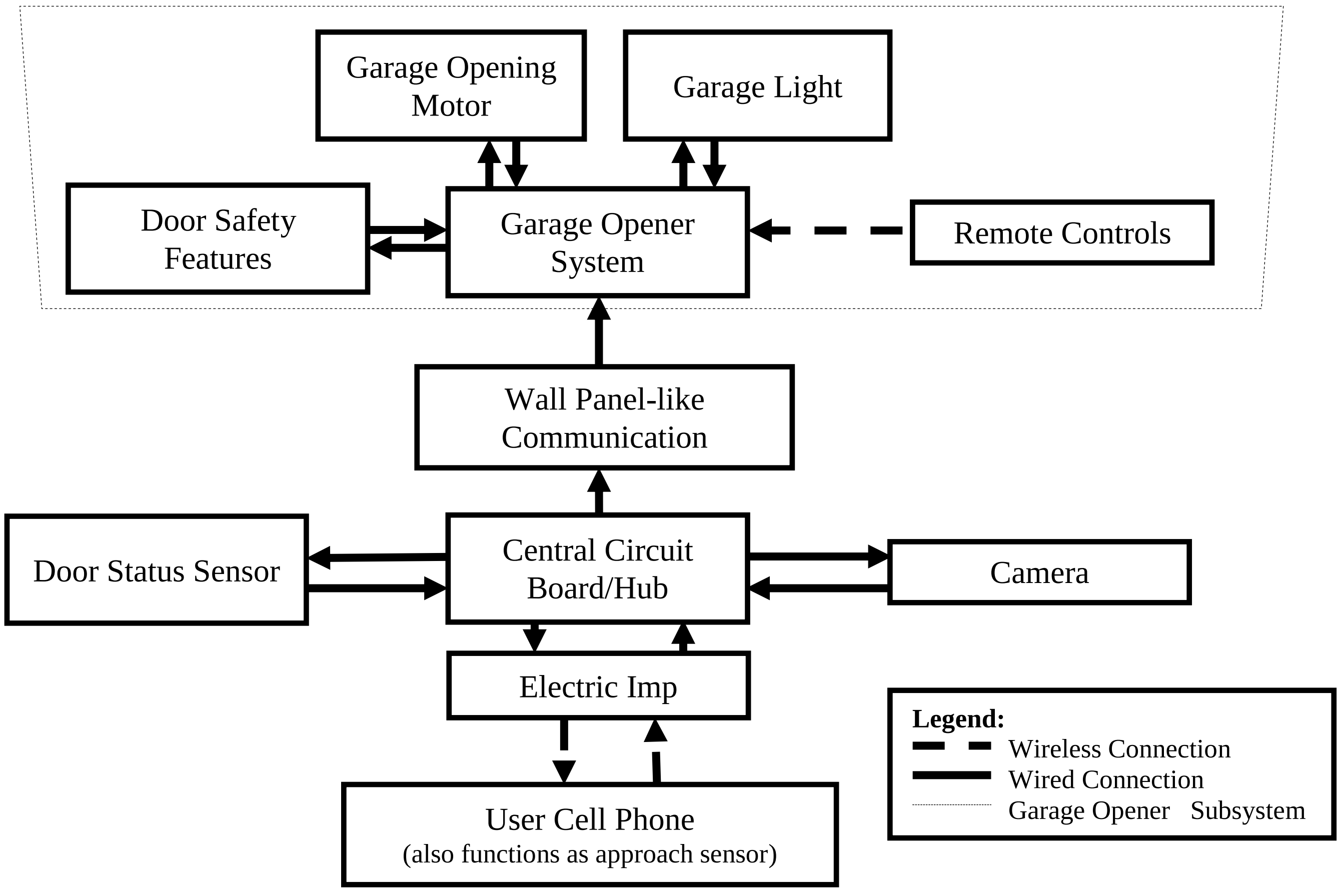


Figure 3.2.1. Proposed Block Diagram for the Open Sesame System Using Hardwire Interface

3.3 Door Status Subsystem

Requirements

The purpose of this subsystem is to allow the device and the user to access the status of the door at any time. The subsystem reports to the microcontroller the current state of the door: open, closed, in transition or error. Upon request, this information can then be sent via the Electric Imp to the Android App and therefore the user. The current status of the door will also dictate the choices that the app presents to the user. For example, if the door is already closed, should the user attempt to close the door, the motor will not run and the user will be reminded that the door is already closed.

The subsystem is required to identify a variety of possible states. It is important that it recognize when the door is fully opened or fully closed. It also recognizes when neither switch is closed and the door is therefore in transition. The error state is trigger in the case when both switches are closed which should only result from incorrect user installation.

Subsystem Description

The subsystem consists of both hardware and software. Two magnetic switches are the central components of the hardware. These switches operate on the simple basis that when the two halves are within one inch of each other (direct contact is not necessary), a short circuit is created. The two halves of the switch must be well-aligned as the tops and bottom edges must line up for the switch to be properly engaged. Figure 3.3.1 shows an image of the selected switches.

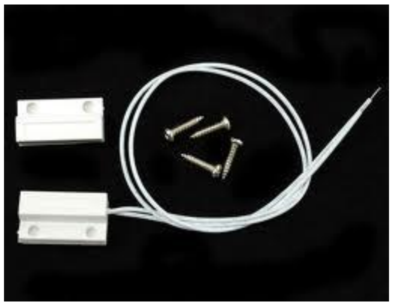


Figure 3.3.1. Magnetic Contact Switch (https://www.adafruit.com/products/375)

These switches were chosen for their simple operation and the durability of the pieces. They were also chosen in efforts to keep the final price of the product down. The functions that they must complete are fairly basic and straightforward and therefore a simple switch is all that is needed. The halves of the switches without the wires attached are screwed to the garage door itself: one to the bottom corner about one foot from the ground and one at the top corner about one foot from the end of the door. The halves with the wires are affixed to the garage door track. The set at the top of the door are aligned when the door is open and the set at the bottom are aligned when the door is closed. The spacing between the two halves must be less than one inch and the tops and bottoms must align for the switch to be closed. The planned placement of the switches can be seen in Figure 3.3.2.

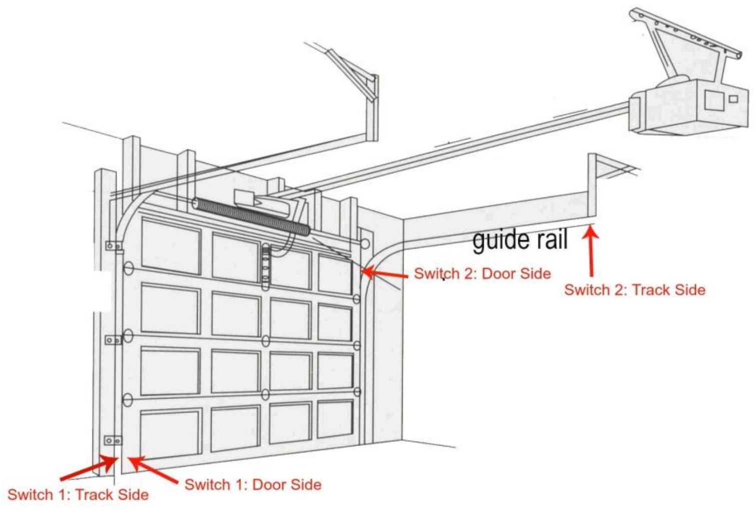


Figure 3.3.2 Positioning of Magnetic Switches

User installation of the switches would also require the four loose ends of the switch wires to be installed into the appropriate screw terminals so that their status can be determined by the microcontroller. These wires and terminals are color-coded. For example, the switch at the top of the door must go into two specific terminals in order for the door to make the decision that the door is open. However, between the two wires attached to that switch, there is no distinction that has to be made as there is no polarity to the simple switch. The portions of the board dedicated to the door status system can be seen in Figure 3.3.3.

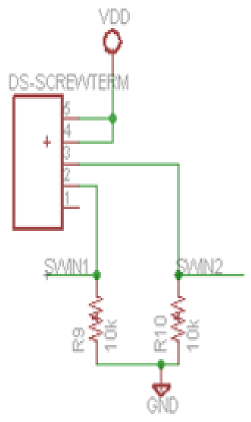


Figure 3.3.3. Board Schematic for Door Status Switches Portion

The operation of the switches requires the use of pull-down resistors chosen to be 10 kΩ. The switches are provided input at 3.3 V by being connected to V_{dd}. from an output I/O pin on the microcontroller. D0 was chosen for this purpose. Between the switch and the resistor, the voltage is read by an input pin on the microcontroller. D0 and D4 are used, one for each switch. If the value is read as high, the switch is closed and vice versa. The circuit schematics, including the different switch positions leading to different state classifications can be seen in Figure 3.3.4.

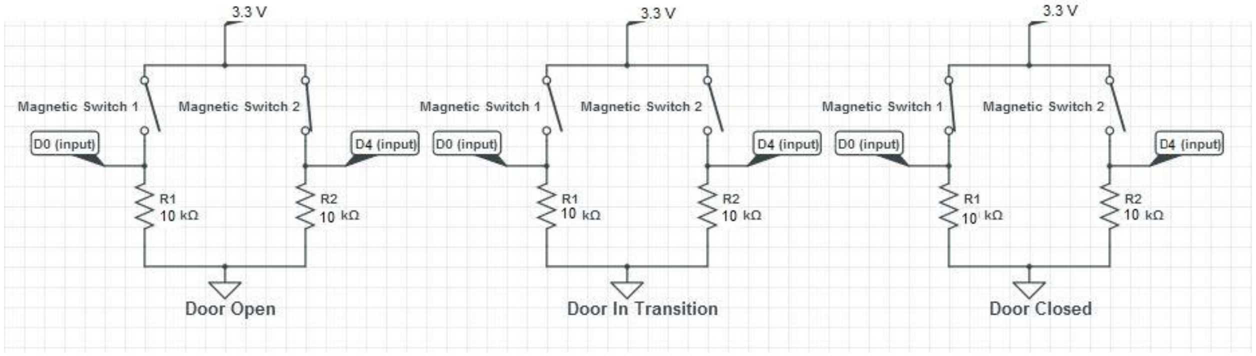


Figure 3.3.4. Circuit Diagrams of the Door Status Switch System

The software for the system consists of code written in MPLAB for the PIC32MX795F512H microcontroller that controls and monitors the values on the two I/O pins: D0 and D4. All of the pins are first set to digital. Pins D0 and D4 are set as inputs using the TRISD command. When the SPI command 0xD5 is sent from the Electric Imp, the door status case of the switch statement in the code is entered. Inside the case is an if statement that can identify four different cases. If both values are high, the variable "send" is set to 0x04 which the Electric Imp interprets as an error since this is the impossible state where both switches are closed. If D0 is high and D4 is low, the door is closed and the "send" variable is set to send 0x00 on the next read, which the Imp identifies as closed. In the opposite case, the door is open and "send" is set to 0x01. When both values are low, "send" is set to 0x05 which the Imp knows as the transition state. If the user checks back after attempting to either open or close the door and the status has not changed, the user should then choose to take a picture to see what is obstructing the door. The error state of both switches being closed generates a 0x04 to send. These code segments can be seen in the main microcontroller code in the Appendix and the flow of this logic can be seen in Figure 3.3.5.

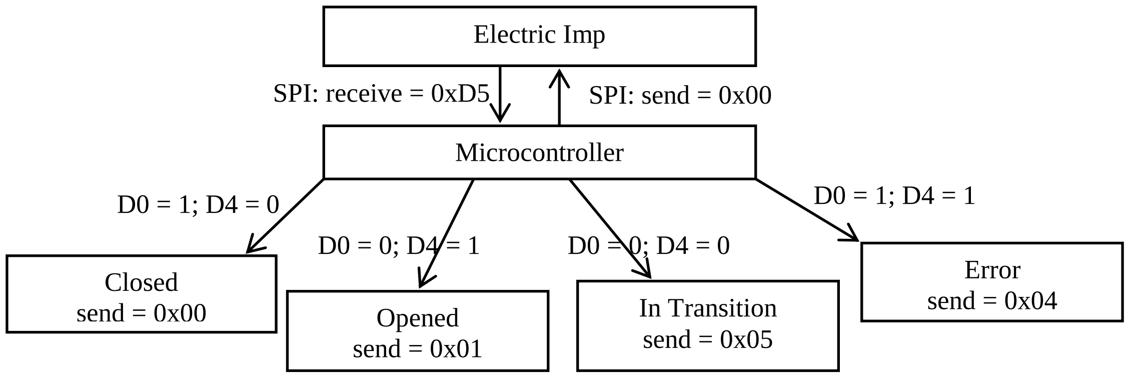


Figure 3.3.5. Flow Diagram of Door Status Code

Subsystem Testing

Before interfacing the subsystem to the rest of the project, it was tested using the LED bank on the kit boards. The switches were taped to two pieces of paper with the wired halves about eleven inches apart and the other halves about 4 inches apart. The switches were then breadboarded to the necessary components and then connected to the kit board. The two pieces of paper were then moved past each other to simulate the movement of the door between the stationary switch halves attached to the tracks. Instead of setting a variable within the code to indicate the current state of the door, the LEDs on the kit board were lit up in different combinations to indicate different states. The allowed for dynamic, obvious and instantaneous results as the switches moved positions. One half of the LEDs were lit up to indicate one switch closed, and the other half to indicate the other switch closed. All LEDs were off when neither switch was closed. All the LEDs were lit up in the case of the both switches being closed, because this is an (impossible) error state.

In order to test the subsystem after it was integrated into the rest of the project, a physical demo was built. The switches were attached to the "door" in a manner reflecting what the actual installation would look like. The door was placed in the three most likely positions in turn: open, closed and in transition. The app was used to check the state in each of these positions and the results were determined to be accurate. The subsystem was also tested by selecting the open option when the door was already open and vice versa to ensure that the subsystem would warn the user that the door was already in the desired status and that the motor would not run. It was also tested to ensure that the correct status is displayed on website when those buttons are used to check the door position.

3.4 Electric Imp Subsystem

The Electric Imp, developed by Electric Imp, Inc., exists to bring Wi-Fi capabilities to everyday devices. In the case of Open Sesame, the Electric Imp provides an interface between the internet to the circuit board and connects directly to the microcontroller. Aside from connecting the microcontroller to the internet, the chip also runs two sets of code both written in Squirrel, a coding language used for some video games and this application. The first code is considered the "Agent" and runs on the Electric Imp Cloud. Changes to this code must be made on the Electric Imp website. This code responds to http requests made by the user accessing an Imp specific URL. Each Electric Imp come preloaded with their own URL, and modifications to this web address allow for different commands. For Open Sesame this includes adding "?status=1", "?open=1", "?close=1", or "?pic=1" to the end of the web address to access the different possible commands. Depending on the URL accessed, the Agent will send commands to the other set of code located on the physical Imp called the "Device" code. The device code must be sent to the Imp over a stable Wi-Fi connection. This connection is established using BlinkUp, a protocol developed by Electric Imp to send wireless network login information through a series of blinking lights. On the device side, the code is written to communicate with the microcontroller via SPI with the Electric Imp as the master. This is the only way to use the Electric Imp in SPI. The final versions of both the agent code and the device code used for the prototype can be found in Appendix 9.4.

On startup the Electric Imp automatically checks the status of the door. This allows for status changes to happen instantaneously as opposed to only after checking the status of the door. This is also the only command that is automatically run by the Imp, all other commands are made only after the agent URL is accessed. After the agent URL is accessed, decisions are made based on the current status and command by the Imp to run certain commands. The agent URL commands trigger different functions on the device side. Once a decision is made, the command is sent to the microcontroller, and the Imp is put to sleep until the command can go through. This prevents the Imp from asking for data from the microcontroller sooner than it is ready. For example, if the command to take a picture is made, the Imp will send the SPI command “0xCA” to the microcontroller it will then wait before reading back the picture in the process described in the Camera subsystem section. If the command to open the door is made, first the Imp requests the door status from the microcontroller with the command “0xD5”. It then decides from there whether the door needs to be opened and if it does, it sends the command “0x0D” to the microcontroller and if not, it updates the website text to inform the user that the door is already open. The current version of the Imp code also has the beginnings of a future improvement that could inform the user of the time of the last door status change.

The Imp has a vital security flaw, that anyone with access to the URLs may command it. As the team has taken steps to prevent the URLs from being distributed, this should not be of immediate concern. If the product is mass produced, each owner would still have an Electric Imp with a custom URL and therefore each user’s application can be customized to access these specific URLs only which addresses the security issue. The greater concern will be users sharing their passwords too willingly, leaving their garage doors open to the world.

In addition to the software required for the Electric Imp to function properly, certain hardware needs to be incorporated on to the board. As can be seen in the schematic in Figure 3.4.1, the Imp is connected to the board through a SD card mount.

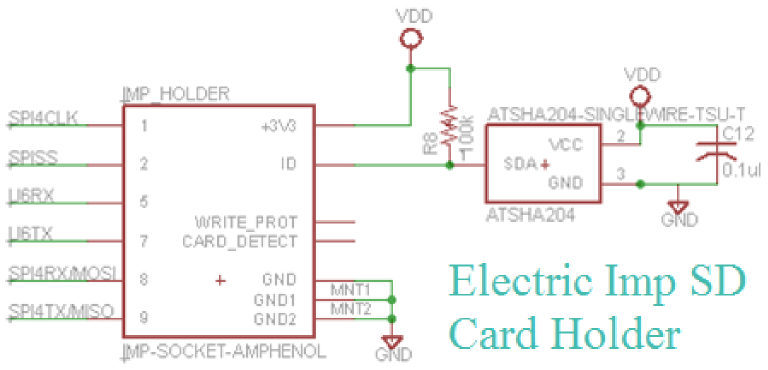


Figure 3.4.1 Schematic for Electric Imp SD Card Mount

The mount needs to be connected to ground at three separate points so that the whole metal casing is grounded. Three pins are used to establish the SPI 4 communication that was used for all microcontroller-Imp signals. This requires the clock, transmit, receive and slave select pins to be connected to the microcontroller in a way that allows the Imp to be the master. The exact pin connections can be seen in the full schematic in Figure 3.8.8. The two pins that would be required to allow for UART 6 communication between the Imp and the microcontroller are also connected. This was done in case these pins would be needed for some unforeseen application, which they were not. The V_{dd} of the Imp is 3.3 V which matches that of the rest of the board. The creators of the Electric Imp require the use of the ATSHA204 IC chip to connect to the ID pin. A link to the data sheet can be found in Appendix 9.2. This chip allows for secure authentication and validation of the chip to allow it to be recognized by the code on the Electric Imp cloud. The information for the values of the circuit components surrounding this chip can be obtained from the Electric Imp development website.

3.5 Garage Door Motor Subsystem

Requirements

- The system must have communication between the microprocessor circuit and the garage door motor. The user must be able to issue commands through the cell phone application, which will be processed through the Electric Imp cloud and passed on from the Electric Imp to the microprocessor, and finally to the motor circuit.
- The system must be able to send a signal to the motor such that the motor will reverse the previous state whenever a button is pressed. If the door is closed, pressing a certain button on the app sends a signal to the Imp that will then cause the motor to act so that the door is opened. Likewise, if the door is open, pressing the close button on the app will close the garage door.
- The system cannot interfere with existing safety features of the garage door opener system, only work in parallel. This means that the system cannot send commands that would override safety features, such as detection of an obstruction to the door's path. The system should only send commands that are processed the same way the existing wall panel and remote control signals are processed.

Subsystem Description

The function of this subsystem is to provide a bridge between the garage door motor and the additional features the entire system is providing, as there must be a way for the user's input via the cell phone application to reach the motor. In other words, this subsystem is the communication between the user and the garage door opener itself, as well as the interface between the physical motor and the microcontroller.

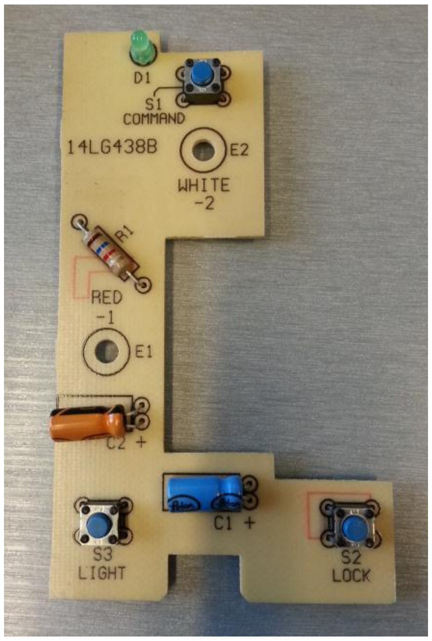


Figure 3.5.1. Wall Panel Circuit

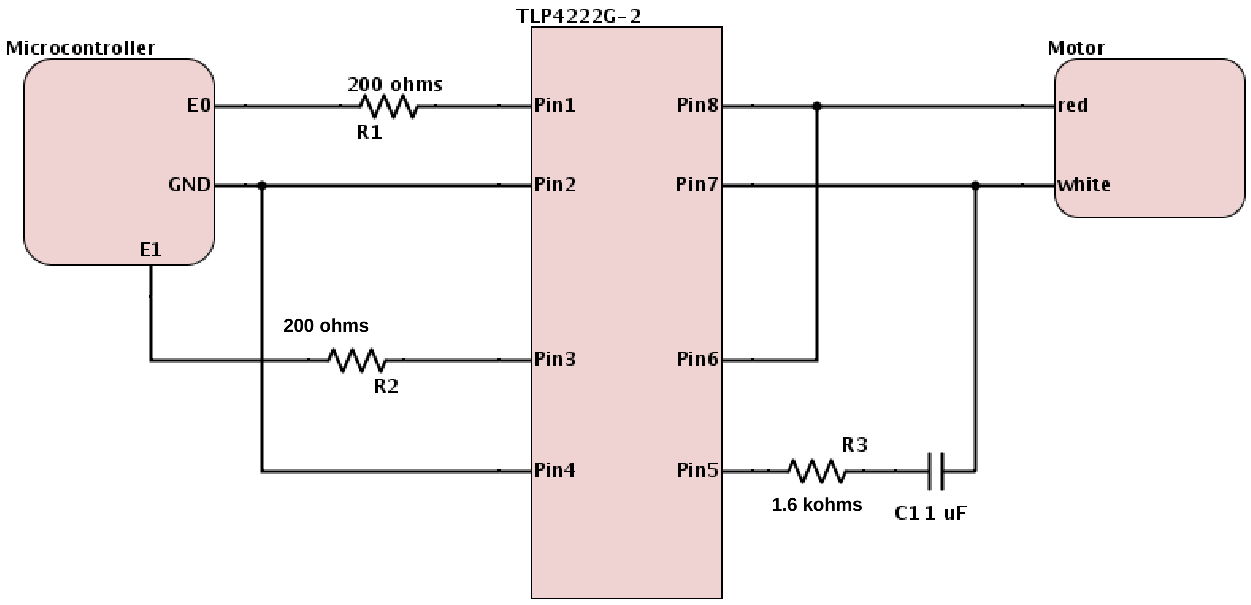


Figure 3.5.2. Hardwire schematic

As most of the hardwiring components of this subsystem are circuit elements, the only programming aspect is setting the appropriate microcontroller output. As a result the microcontroller code is written in Eagle and exists in the length of code for the entire system. The code needed to specifically send the relay a signal from the microcontroller is very simple, as it is just setting an output pin from the microcontroller to high (logical 1), thereby sending the relay a logical high that closes the relay. When the relay receives a high signal, the motor circuit is sent a signal to reverse the previous condition of the door. The following code is used to set up motor control from the microcontroller:

```
TRISBbits.TRISB3 &= 0;// B3 out (controls door)
TRISBbits.TRISB4 &= 0;// B4 out (controls light)
LATBbits.LATB4 = 0;
LATBbits.LATB3 = 0;
```

To run the motor the output pin B3 is set to one and as a result the motor will reverse its previous state. The output pin B4 controls the light. As can be seen in the schematic diagram, both output pins on the microcontroller are connected to the same terminal screws on the actual motor, therefore the difference in the motor's resulting action occurs only because there are capacitors at the output of the relay and before the terminal of the motor. This RC circuit delays the voltage high and the motor is built to recognize such a delay and then reverses the status of the light, not the motor.

Once this subsystem was integrated into the entire project, some additional precautions were necessary. Since the time it takes for the code to run is much shorter than what is practical to send a 1 and then 0 so that the state does not remain 1 forever, this setting of motor back to 0 is done whenever the microcontroller checks the status of the motor. Similarly, there is a statement included for the light to be set back to 0, otherwise the light would not turn off after four and a half minutes of inactivity. As an additional feature, the microcontroller is programmed to send the motor a 1 and 0 three times when an open/close command is issued remotely, effectively causing the motor lights to flash on/off three times to warn anyone within the physical vicinity of the garage door that a remote command was issued.

In the circuit the two solid state relays are 4-pin from Panasonic, chosen for a few features. A link to the data sheet for these relays can be found in Appendix 9.2. (Note: The relay parts are through-hole, which required a special Eagle library file for the board design.) It has a 100 mA load current and can support the voltage seen at the output of the motor (around 17 V); in addition, the LED operates at around 5 - 25 mA and the current coming from the output pin on the microcontroller (and to the LED) is within this range. Figure 3.5.3 shows the schematic of the relay, with terminals 3 and 4 being the output seen at the terminals of the motor. Additionally, this relay is normally open, which is necessary because in the original wall panel the circuit is open until the user presses the button to close the circuit. Therefore the circuit implemented needed to also be open until closed by the user (by wire signals now instead of physically pressing a button).

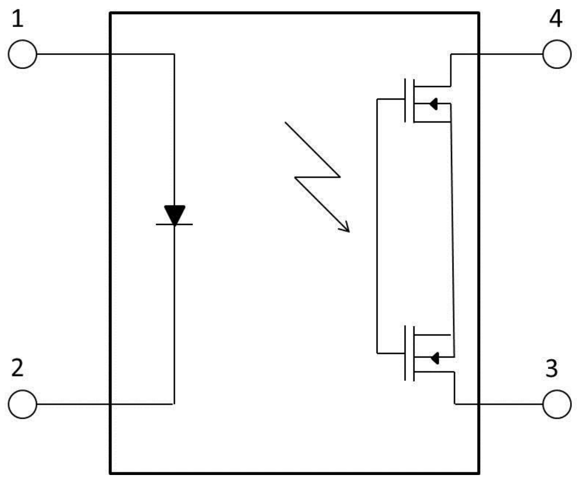


Figure 3.5.3. Relay Diagram

Subsystem Testing

In order to test the subsystem the code was downloaded to the PicKit, which was hardwired to the relay on the breadboard that was wired to the screw terminals on the motor. With the motor plugged in, and the program downloaded to the PicKit with a value of 1 (a digital high) passed on an output pin to the (input pin 1) relay, the motor then began running (as long as the optical eyes were aligned with no obstruction). When a 0 (digital low) was passed through the PicKit the motor did not run. This test proved that once the interface between Imp and microcontroller is introduced, the Imp should send a digital high when the door is to be opened/closed and this will be received and carried out by the motor (via the relay), otherwise the Imp will send a digital low and the motor will not move.

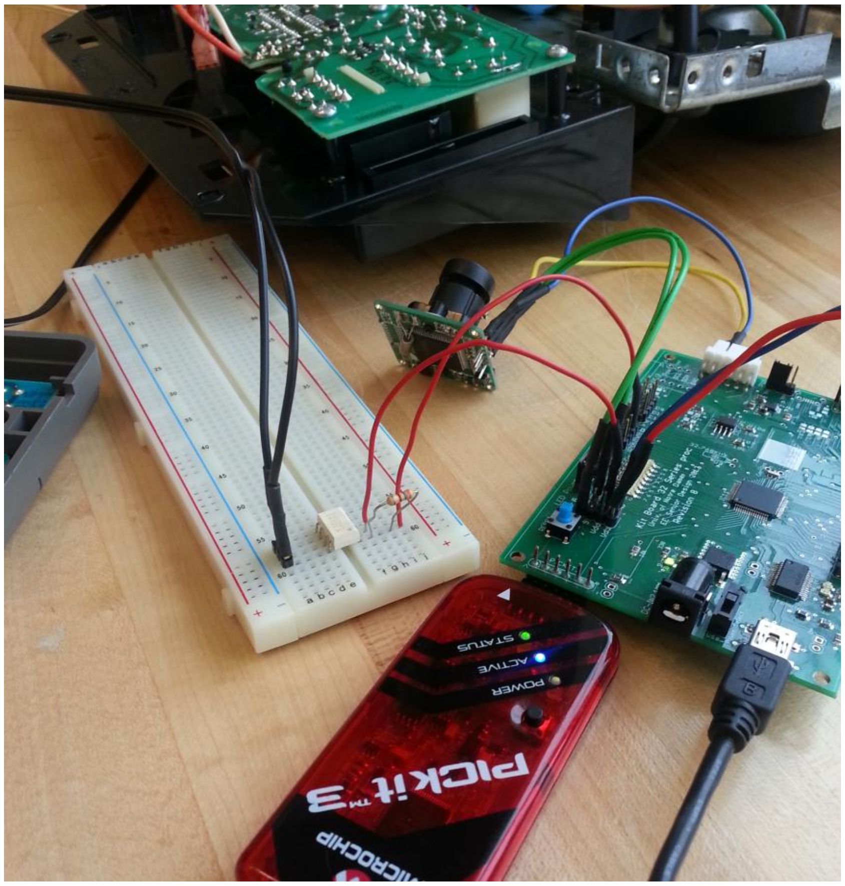


Figure 3.5.4. Subsystem Testing Setup

Another Approach

The method of communication mimics the existing wall panel circuit seen in Figure 3.5.1 by directly hardwiring Open Sesame's main intelligence circuit to the garage door motor circuit. This hardwiring is done through the use of a solid state relay, which is used in place of the buttons that would send commands to the motor circuit when pressed. When the user presses a button in the cell phone application, the microcontroller receives a corresponding signal through the Electric Imp and Electric Imp cloud, which then sends the appropriate signal to the microcontroller and the relay switch, closing the (normally open) relay so the signal is sent to the motor; the schematic for the circuit created can be seen in Figure 3.5.2. Closing the relay completes the circuit across the terminals of the motor, allowing the signal to be sent. Though the hardwire option is what was actually implemented for this project, there was another avenue explored for how to interface the main circuit board with the microcontroller to the garage motor circuitry. Rather than recreate the wall panel circuitry, Open Sesame attempted to mimic the structure and function of existing garage opener remote and keypads. The aforementioned apparatuses use radio frequency (RF) as the method of communication with the garage motor circuitry.

In order to accomplish this, research on the radio frequency communication process was done, specifically with the use of encoding, as the motor circuit decodes the signals it is sent from the RF devices. As a security feature, the encoder must be able to "code hop," meaning that the code viewed externally to the system appears to change unpredictably each time it is transmitted. There is also an encryption key used in the encryption algorithm to scramble data, and a decryption key used in the decryption algorithm to unscramble data. In a symmetrical block cipher, the encryption and decryption keys are equal and thus referred to as a crypt key. To generate these unique crypt keys, the encoders are programmed as a function of what is called the manufacturer's code, and the decoders are programmed with the manufacturer code itself.

Since the motor comes with a predetermined decoder, the next step Open Sesame took was to find a compatible encoder to be connected to the main circuit board. The KEELOQ Code Hopping Encoder HCS362 seemed to fit most of the requirements for the RF circuit to be designed in terms of operating conditions and function—the specifications sheet, for which a link can be found in Appendix 9.2, for the product even listed a typical application for the HCS362 to be gate and garage door openers. The problem lies in the mystery of the motor decoder.

While it does not seem necessary to have the encoder and decoder have the same manufacturer, information is needed from both sides. It is important to know that the decoder will be able to decrypt the data transmitted and the length of code that the decoder is expecting to receive. Figure 3.5.5, taken from the HCS262 specifications sheet, illustrates a basic idea of what would be transmitted from the encoder to the decoder. Looking for this information proved to be fruitless, as the company that manufactured the motor used for this project, along with all other companies that manufacture garage door opener motors, keep that information tightly guarded. Following the advice of a couple of professors, research was done on universal remotes and how those manage to be compatible with most appliances. Unfortunately, the manufacturers of universal remotes are just as quiet about their methods as garage door opener manufacturers are of their special codes.

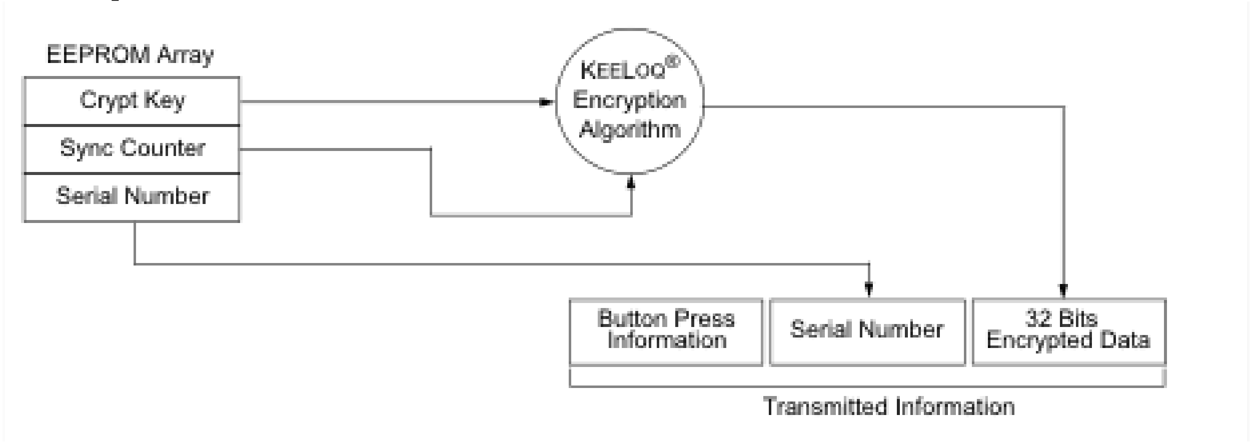


Figure 3.5.5. Building the Transmitted Code Word (Encoder)

It is possible that this problem of encoder/decoder compatibility could be bypassed if the motor decoder was able to learn the HCS362, however there was not enough time to devote to this without a guarantee that it would at least somewhat work. Figure 3.5.6. is also a block diagram taken from the HCS263 specifications sheet, illustrating the decoder's basic operation. In that diagram, the KEELOQ decryption algorithm is listed, although this would not be used by the motor as its decoder already has a separate decryption algorithm.

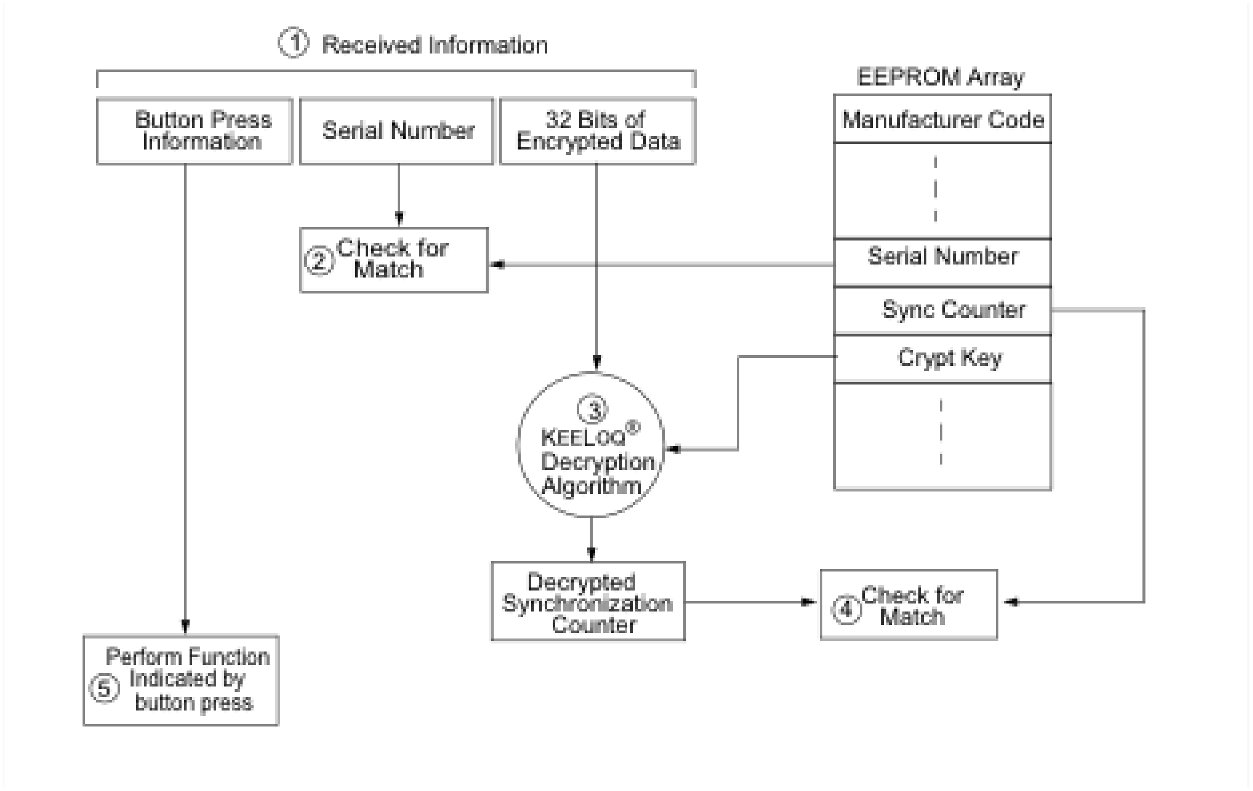


Figure 3.5.6. Basic Operation of Receiver (Decoder)

3.6 Android Application Subsystem

The purpose of the Android Application subsystem is to allow the user of Open Sesame to access their garage door through their mobile phones. The wireless application will give the user another method of control over their garage door. Through this application, the user will be connected to their garage door even after the user leaves his or her home. Note that all of the files named in this section can be found in Appendix 9.5, at the end of this report.

The features of the Open Sesame application include: status update, door controls, Wi-Fi detection mode, and a camera option. The status update will let the user know the current status of the garage door, which has four different statuses or states. These states are: opened, closed, capturing picture, or in transition. The transition state of the garage door signifies that the door is either in the process of closing or opening but has not fully completed the action yet. This way the user will never be in left in the dark about whether the door has been left opened. The door controls act like a remote control for the door with the open and close buttons. The Wi-Fi detection mode is a user convenience feature that acts as a proximity sensor based on the user's home Wi-Fi connection. If the Wi-Fi detection mode is turned on, the mobile application will automatically open the garage door if it recognizes the user's home Wi-Fi. The idea behind this feature is to allow a hands-free option for the user as they enter and leave their homes through their garage. Finally, the camera option will allow the user to access the camera that is integrated with the system and take a picture confirmation of the door status. The Open Sesame application will improve the convenience of the user by giving the user wireless communication with the garage door opener and reassurance with the door status and camera confirmations. These are all the criteria set for the Open Sesame mobile application.

The application was developed using the Eclipse IDE with the Android Development Tools (ADT) plug-in. This plug-in provides an integrated environment for Android app development which makes creating a new app easier. In other words, the ADT provides all the necessary library used in Android programming. The Java language is the primary programming language that Android apps are developed with, which is the language used for developing the Open Sesame app. The user interface (UI) in Android programming is described by the Extensible Markup Language (XML). The XML files in the Android project creates the view and UI that the user sees while the Java code in the project provides all the functionality of the app. More specifically, the Java code in the project inflates the XML files that describe the UI view and also provide functionality to the app.

The AndroidManifest.xml file is an important file that is present in every Android application. The manifest file presents essential information about the app to the Android system. This is essential information that the system needs before it runs any of the app's code. In addition, the manifest describes the components of the application such as the activities, permissions that the app requires from the system, and action-filters among other things. For the Open Sesame app, there is only one activity that needs permission to access the internet, the Wi-Fi connection state, and a change in Wi-Fi state action-filter. By including the proper permission, the app will be able to use the internet connection on the phone and by enabling the action-filter; the app will be able to detect when there is a change in the Wi-Fi state. This last point will be important when discussing the Wi-Fi detection mode.

There is one main activity that describes the main page that the user sees when he or she opens the Open Sesame app. The main activity inflates the tab layout view to create a tabhost, from the tabs.xml file, with three tabs labeled, "STATUS", "CONTROLS", and "CAMERA" from left to right. This entire view is described in the tabs.xml file which sets the buttons, the text views, and the switches that the user sees.

In the "STATUS" tab, there is a "STATUS" button and a switch that activates Wi-Fi detection mode. Again, the button and the switch views are implemented in the tabs.xml file. If the "STATUS" button is pressed, the app will communicate with the Electric Imp (if it is online) through the Imp cloud to ask for the current status of the garage door. The status will be returned to the app and displayed in the text view just above the "STATUS" button. On the other hand, if the Wi-Fi detection mode is switch on, then the app will start to be aware of the Wi-Fi state and the Wi-Fi that it is connected to. The way the app is able to become conscience of the Wi-Fi state is through a broadcast receiver. A broadcast receiver in Android allows the application to listen in the background for actions that occur. The Wi-Fi state change action-filter was enabled in this app as described in the AndroidManifest.xml. Once this action-filter is registered for a broadcast receiver, the app will listen on that receiver for any changes in the Wi-Fi. This means that when the Wi-Fi detection mode is turned on, a broadcast receiver with that Wi-Fi action-filter is registered and starts to listen. If the mobile phone connects to a new Wi-Fi connection, the broadcast receiver will check if the connection is recognized as a home network. If that connection is recognized, then the phone will automatically send an open command to the Imp. If the phone loses connection with the home Wi-Fi network, then the app will automatically send a close command to the Imp. The range of the home Wi-Fi network acts as a proximity sensor for the app. If the user is at home, he or she can turn off the Wi-Fi detection mode, in which case the app will unregister the broadcast receiver and stop listening for the Wi-Fi network.

The Electric Imp was used as an in-between component that establishes communication between the app and the microcontroller. We chose the Imp because it is readily able to efficiently communicate with the microcontroller and also connect to the internet. When the Check Status button is pressed, the application will send an HTTP GET request to a certain Uniform Resource Locator (URL) that is established by the Imp. The HTTP request will GET from that URL the current status of the garage door. And in the process of getting the status, the URL will send a response to the Imp indicating that the status was checked. Using the IMP and the HTTP request establishes a two-way communication. This method of communication between the app and the Imp is consistent for all the buttons on the Open Sesame app. The difference between the buttons in the app is the URL that they access. Since each button needs to send a different command, each button is hard-coded to send a GET request to a specific URL that corresponds to that command.

When the “CONTROLS” tab is pressed on the tabhost, the view switches to the “CONTROLS” tab view and the user is presented with an Open and a Close button. These are the buttons that the user can press to open or close the door. Sending a signal to the Imp is the same as checking the status of the door. By sending an HTTP GET request, the Imp knows that the app is trying to communicate with the microcontroller. Each button that requires communication with the Imp is associated with a state. When the app communicates with the Imp, the Imp receives the command of the app as a state. For example, when the Open button is pressed a state of 1 is passed and the Imp knows to set the door state to 1, which is open.

When the user switches to the “CAMERA” tab, he or she is presented with just the “CAPTURE” button. When this button is pressed, it will send the usual HTTP GET request to the Imp indicating that the user wants to take a picture. At the same time this is happening, the button will try to pull an image from a Notre Dame Virtual Private Server (VPS). The VPS is where all the images taken from the camera will be stored. When the Imp gets the command to take the picture, it will send the picture to the VPS, which the app will then download that picture and present it to the user. At current version of the Open Sesame app is able to pull a picture from the VPS, however, the Imp is not able to post a picture. This could be due to security clearances that the Imp requires in order to upload onto a Notre Dame server. Please refer to the Imp section of this report for more details.

Figure 3.6.1 below gives a diagram of the hierarchy of the Open Sesame app. The top level is the MainActivity.java file which sets up the components below it. The main activity inflates the tabs.xml file which in turn includes a tabhost, as described above. The tabhost hosts three different tabs each with its own specific layout. The three screen captures at the bottom of figure 1 are the actual images of each tab in the latest version of the Open Sesame app.

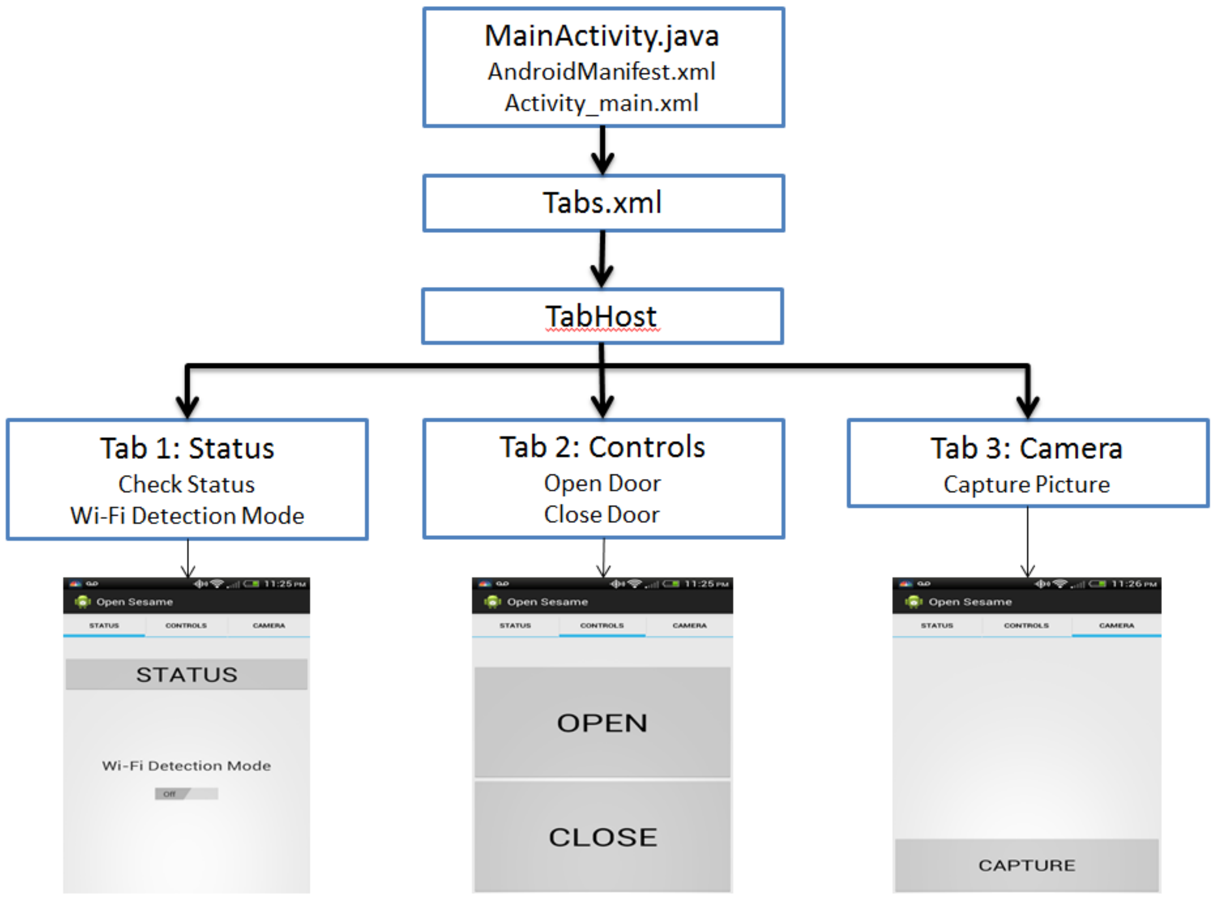


Figure 3.6.1. Application Hierarchy

3.7 Camera Subsystem

Requirements

This subsystem must successfully place an image in the Electric Imp cloud after a command is sent over the internet via the app. Once the image is on the agent, it can be moved to a server where it can be viewed. One of the central features of the system that makes it modern and enhances security is the ability to view the door remotely. This subsystem allows the user to accomplish this through their cell phone. For example, if the user is told by the door status feature that the door is stuck in transition, this feature will allow them to view the door and see what the problem is and what is potentially blocking the door. This subsystem also uses the ability to control the light on the existing garage motor to help improve the visibility and quality of the image.

When designing this subsystem, customer specifications were considered. The subsystem must be easy to mount in a location where the garage door is visible. The cost of the camera was kept low because the overall cost of the product is more important than a high quality image from a fancier camera with many features that would remain unused in this application. In addition, the delay from when the app requests a picture to when the image is fully transmitted to the phone was minimized when possible by using SPI communication and compressed images.

Subsystem Description
Hardware

There are three main components of hardware for this subsystem. They are the Electric Imp, the camera and the microcontroller. The PIC32MX795F512H microcontroller is once again used as the central hub for this subsystem as it is with many others. The two UART 3 pins and three SPI pins were required to connect it to both the camera and the Imp. The microcontroller is used as an intermediate step between the camera and the Imp for several reasons. One is that the camera and the Imp prefer different signal protocols and the microcontroller can accommodate both. Secondly, connecting the Imp directly to the camera would take up three of only six pins available on the Imp and they would only be able to be used by this one system. This seemed like a waste of limited pins. Also, the microcontroller serves as the central point of communication for most of the subsystems, so it seemed logical to use it in a similar way for this one as well to better control the command priorities.

The camera chosen is the Adafruit TTL Serial JPEG Camera. Figure 3.7.1 shows a picture of the camera and its board.

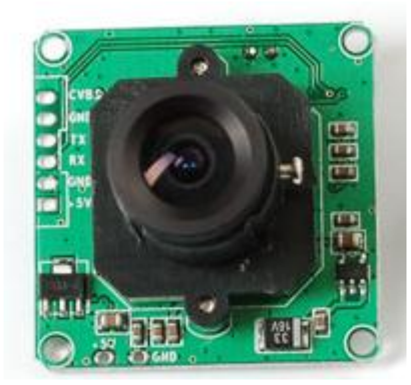


Figure 3.7.1. Adafruit TTL Serial JPEG Camera (https://www.adafruit.com/products/397)

The Adafruit camera was chosen because of its compact design as well as its image processing capabilities. Before taking a picture the camera automatically adjusts the white balance and contrast to appropriate levels, and after the image is taken it is compressed into the .jpeg format which is much more manageable to transmit in reasonable amounts of time. This also removes the need for any further image processing, which is outside the scope of this project. When compared to other cameras, the price of the Adafruit camera was appropriate based on the expected functions. Other more expensive cameras may have more capabilities or be more customizable, however the customer requirements force the cost of the project to be as low as possible. Other possible cameras were more simple and it was determined that a camera with a library of commands was necessary over a camera that would require programming before continuing. A sample picture taken by the camera can be seen in Figure 3.7.2. This is an appropriate level of quality for the applications and requirements of the product.

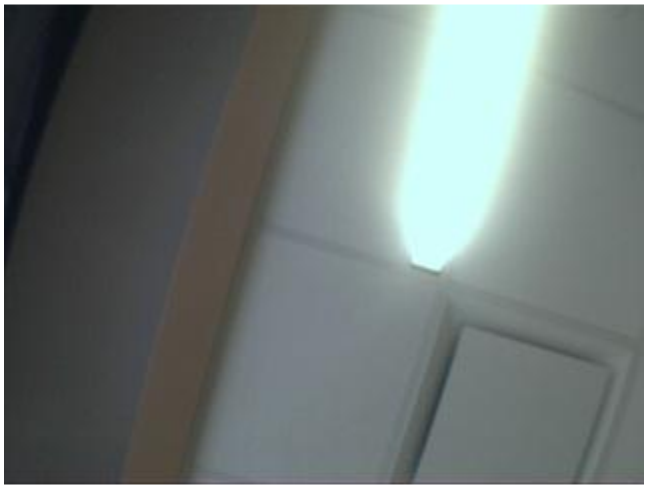


Figure 3.7.2. Sample Image from Camera to Demonstrate Quality

The camera requires 5 V as its V_{dd}. This voltage was provided using a MCP1252 low noise, positive-regulated charge pump. The data sheet link can be found in Appendix 9.2. The board schematic for this portion of the circuitry can be found in Figure 3.7.3.

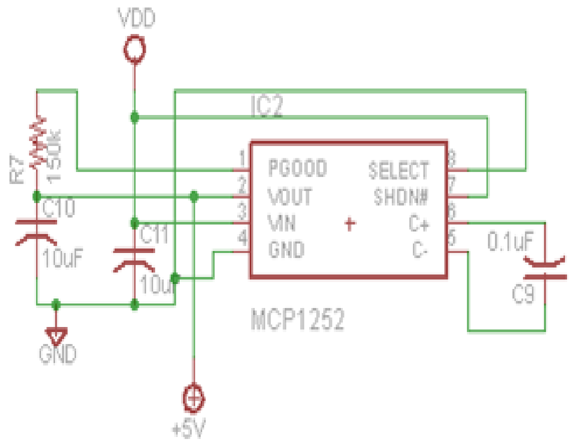


Figure 3.7.3. Schematic for Circuit Providing 5 V to Camera

Here the desired output is on pin 2, the V_{out} pin. The board V_{dd} of 3.3 V is connected to pins 3 (V_{in}) and 7 (SHDN’). The GND pin and the SELECT pin are connected to ground. By connecting the SELECT pin to ground, the fixed output voltage of 5 V is selected. The output current of the device is between 120 mA and 150 mA which are within the range that the camera can handle. For the C_{in} and C_{out} capacitors, a value of 10 µF is used to minimize noise and ripple. The flying capacitor connected between the C+ and C- has a value of 0.1 µF because it controls the strength of the charge pump. This value results in less output ripple, but it also reduces the maximum output current which was not an issue in this instance. The purpose of the P_{GOOD} pin is to act as an open-drain output to sink excess current when the regulator output voltage becomes too low. This pin requires a resistor of value 150 kΩ between it and the V_{out} pin to perform this function correctly.

The third component of the hardware is the Electric Imp through its breakout board. The Electric Imp was chosen because of its ability to interface with the internet in a general user-friendly fashion, with the capabilities allowing for sending information to a server. It also had enough memory and speed to transmit a full image.

The Electric Imp component is connected to the microcontroller via the SPI pins discussed in the Imp section of this report. The camera is connected to the microcontroller through two UART pins. UART 3 on the microcontroller requires pins 5 (G7, U3RX) and 6 (G8, U3TX) and these are connected to the appropriate pins on the MOLEX connector that connects to the camera. The MOLEX connector also provides the connections to ground and 5 V. The use of the MOLEX connectors simplified the process of joining the non-standard pin spacing of the camera board to the main board. It also allowed the camera to be connected by a cable that allows the user to adjust the position and angle of the camera to best capture the door.

Software

On the Imp side of this system, there are two parts: the agent code, and the device code. The agent code exists in the cloud and handles the http request made by a user accessing the website assigned to each Imp. The agent will also be responsible for sending any desired information to the server including any images. The device code exists on the Imp and has access to the individual pins. These pins are connected to pins on the microcontroller and are wired for SPI communication. SPI communication is used because it only requires three pins and continues to keep the UART pins on the Imp free if needed. The challenge with SPI is the Imp must be set as the master forcing the microcontroller to be set as the slave. All code described here can be found in Appendix 9.3 and 9.4.

When the user presses the capture button on the app, the device code on the Electric Imp runs the function for taking a picture. This command originates only after an http request is made by opening the web page associated with the agent for the Imp, which is what pressing the app button does. As part of this function, the Imp sends an SPI message to the microcontroller, 0xCA, initializing the steps required for the camera to take a picture. Once this is sent, the Imp goes into a temporary sleep cycle that acts as a 4.5 second delay. This delay allows the microcontroller to take the picture and put the data into the buffer before the Imp attempts to access the information. After it has waited a sufficient amount of time, the Imp will ask the microcontroller for the length of the file. As the microcontroller is the slave, it is waiting for the Imp to make a request, and the file length is waiting in the buffer. Once the length is read, the Imp calculates how many blobs of information it needs to read based on the length. Based on discussion with the developers, the maximum size of blob the Imp can read at one time is 8192 bytes. Therefore the length is divided by 8192, and the number of iterations is calculated. In the agent code, in addition to the command for the device, functions are included that have been written for the agent to handle the image before it is sent to the server. These include “jpeg_start”, “jpeg_chunk”, and “jpeg_end”. In order to use the functions, they are first called by the device code. For instance, “jpeg_start” is called immediately after the the length is read, and prepares the agent to handle a jpeg of this specific length by creating a buffer of this length. Once the number of iterations is known, the Imp can begin reading from the microcontroller. After the first blob is read, the agent function “jpeg_chunk” is called to process it. Because the entire length cannot be read at once, “jpeg_chunk” places the blobs in the correct order as they are read. Usually the length of the image only requires two iterations, further simplifying this task, however the code will run for the number of iterations regardless, creating a level of robustness to the system. After the blobs have been read, “jpeg_end” is called in the agent. “jpeg_end” performs the finals tasks associated with taking a picture including sending the .jpeg file to the web server. Additionally it outputs to the server log the size of the image in bytes.

The PIC32MX795F512H microcontroller communicates between the Electric Imp and the camera. The camera used is set up for UART communication. This is the recommended setting for successful transmission of the commands and the image data. The baud rate for this transmission is 34800 Hz. A function called “serial_init3” initializes this to occur using the microcontroller’s UART 3 pins. Two more functions complete the transferring of the signals. The first is “readu3” which retrieves a byte of data from the camera when it becomes available and the other is “writeu3” which sends a byte input into the register to the camera. However, the microcontroller interfaces with the Electric Imp through SPI, as the slave. SPI communication with the Imp is also used for other subsystems so it seemed logical to use this already setup interface. It also allows for the faster transmission of the images. This was set up in the “SPIinitslave” function on SPI 4. A function called “do_SPI_slave” handles the input of data to be written into the buffer and the writing of the data when it is received from the Imp. There are also commands to set the various pins used to digital inputs and outputs to enable their proper functionality.

Upon receiving a ‘0xCA’ command from the Electric Imp through SPI, the microcontroller first uses the LATB command to set pin B4 high to turn on the light. It then initiates the series of UART commands that the camera requires in order to take the picture. First a sequence is sent that informs the camera to take a new picture and the camera returns five bytes to confirm the message was received correctly. Then the frame length must be obtained in order to ensure that the proper number of bytes are transmitted and stored when the picture is sent. This causes the camera to return an eight byte sequence, the last four bytes of which represent the image length in bytes. The number is typically in the range of 12,000 to 13,000 bytes. The microcontroller then initiations the transmitting of the picture data, including a request for the proper number of bytes. After a sequence of five bytes confirming the correct message was received and a slight delay, the camera begins transmitting the image over UART, one byte at a time. This is read and then stored by the microcontroller in an array named “picture” by a for loop of the length of the data stream. However, because of the limitations of the switch statement form, the array “Picture” in which the picture data is stored, must have a preallocated length. “Picture” is initialized to be 16384 characters long, which is longer than any picture the camera captures. This does not cause any complications because the Imp is given the correct length of the picture so decisions can be made in that code to determine which bytes are meaningful. After the data has been transmitted, there is a slight delay then a five byte code is sent to note the end of the data. Lastly, a resume command sequence is sent to the camera that sets it up and enables it to take another picture.

After the UART communication with the camera concludes, the microcontroller waits for the Imp, as SPI master, to send it two bytes to which it replies with the length of the data. The Imp then sends 0x00 bytes for the whole length of the data and the microcontroller responses with the next element of the “Picture” array until a for loop of the full length has been run. The Imp continues to request data due to the block nature of the blobs and the microcontroller sends 0x00 bytes until the for loop is completed. The code for both the microcontroller and Imp that enable this subsystem to function can be found in the Appendix.

Therefore the process of taking the picture in the code can be summarized in the following steps:

- User selects Capture button on app, making an http request specific to taking a picture
- The Imp receives the request and runs the function for taking a picture
- The SPI command “0xCA” is sent to the microcontroller which enters the corresponding case
- The light on the motor is turned on
- The microcontroller sends the command via UART to the camera to take a picture
- It then sends the command to get the frame length
- The picture is then read and a resume command is sent to the camera
- The Imp requests the length of the image and then the image
- The Imp transfers the data to the agent and from there to the server

Subsystem Testing

Before the subsystem could be constructed, it was necessary to first test if the camera was functioning properly. By following a demo provided with the Adafruit camera, the camera was connected to a PC through an Arduino UNO. Using the VC0Z06CommTool software the functionality of the camera was confirmed by taking and saving pictures to the hard drive.

In order to effectively test this subsystem, the individual signals traveling between the devices must be monitored. This can be done using the USBee software by measuring the signals at connections between components. Using two USBee probes it is possible to measure the UART signal between the camera and the microcontroller, and if the baud rate is set properly to 38400 the signals should be decoded, simplifying the testing. In order to properly watch the SPI signals, three probes are used and in the software the proper channel must be set as the clock. As long as the clock is set properly, the signals should be decoded accordingly. After these five probes are assigned, and the software adjusted, set the USBee to trigger on the clock of SPI signal because this will be the first signal to go through the system and adjust the length of the capture for at least seven seconds. This will provide ample time for execution of the commands. Finally, capture once, and wait for the signal.

After the hardware is set up with the USBee probes, the agent website is ready to be accessed. Once this URL is opened, the agent will begin processing the http request and send a command to the device to run the camera function. This is all internal to the Imp and the Imp cloud. It could be monitored by placing server log commands in the code, however that will also slow down the system. At this point the Imp should send a 0xCA command to the microcontroller to take a picture and the USBee will trigger. After it has triggered and decoded the values, the USBee should display the camera functions. After the SPI 0xCA command, the UART communication should begin between the microcontroller and the camera while the SPI sleeps for a few seconds. After the picture has been returned to the microcontroller and the interactions between the Imp and the microcontroller have taken place, it is possible to compare the two sets of bytes for the image to check for accuracy. Sequences at both the beginning and end of the data were compared. Finally the server log displays the size of the file in bytes, which can be compared to the size of the file passed between the camera and microcontroller.

Ultimately, in the final Implementation, the picture was never actually able to be placed on the server. Confirmation of the above steps occurring accurately was obtained in the methods described. The agent code also returned a message saying that the data had been transmitted to the server and the page would be opened showing that the agent could access the correct location. However, upon checking the server, the picture file size always remained at 0 bytes. Many attempts were made to understand why and to correct this problem. The conclusion reached was that the server provided by and connected to the secure Notre Dame network, did not allow for the upload of files of unknown origins, which is what this picture file would be viewed as.

3.8 Housing & PCB Design

Housing

As this product would be placed in a garage, and effort was made to create a housing that would prevent dust from accumulating on the circuit board, while still allowing access to the reset button and the Electric Imp. Because of available materials, the decision was made to create a prototype on a 3D printer. A casing was created for the main circuit board along with a smaller housing for the camera. The decision to make them separate was deliberate as it is difficult to predict the specific angle that is allowable based on the garage door opener, and making them two separate parts retains some flexibility. Both cases were designed with ports for wires to come out of, and the cover for the camera housing has a hole to allow the lens to come out slightly, if desired. The models were created in Google Sketchup and can be seen in Figures 3.8.1 – 3.8.4. Photographs of the printed cases can be seen in figures 3.8.5 – 3.8.7.

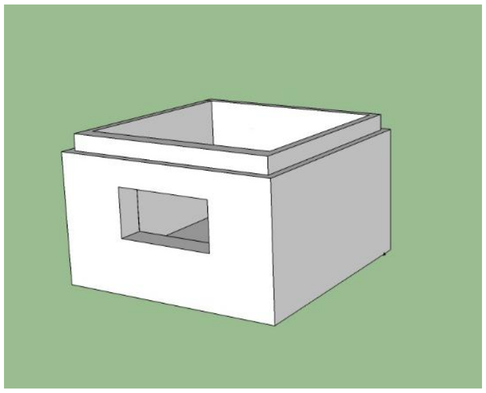


Figure 3.8.1 Camera Housing Base

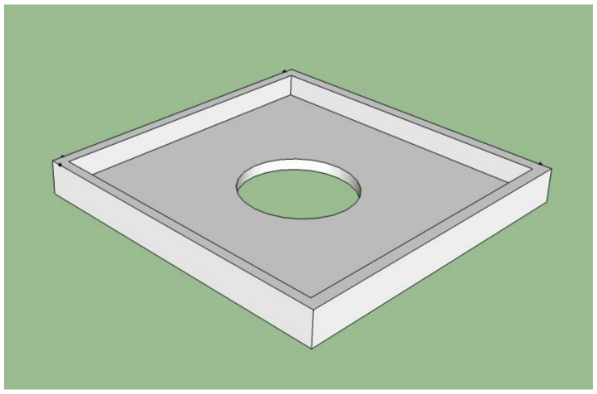


Figure 3.8.2 Camera Housing Cover

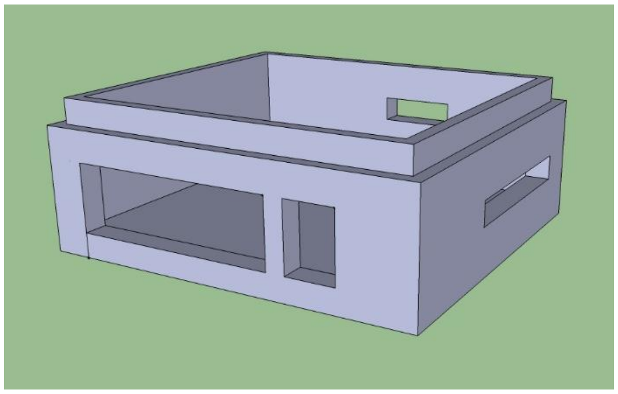


Figure 3.8.3. Main Board Housing

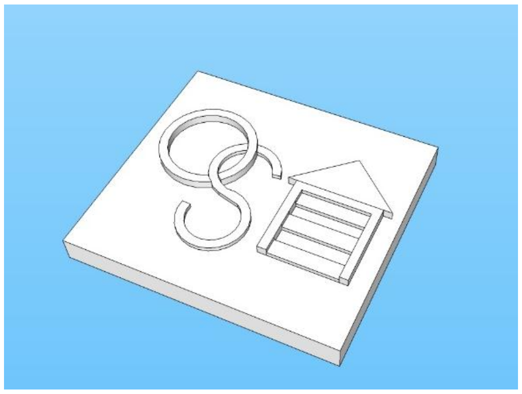


Figure 3.8.4 Main Board Cover

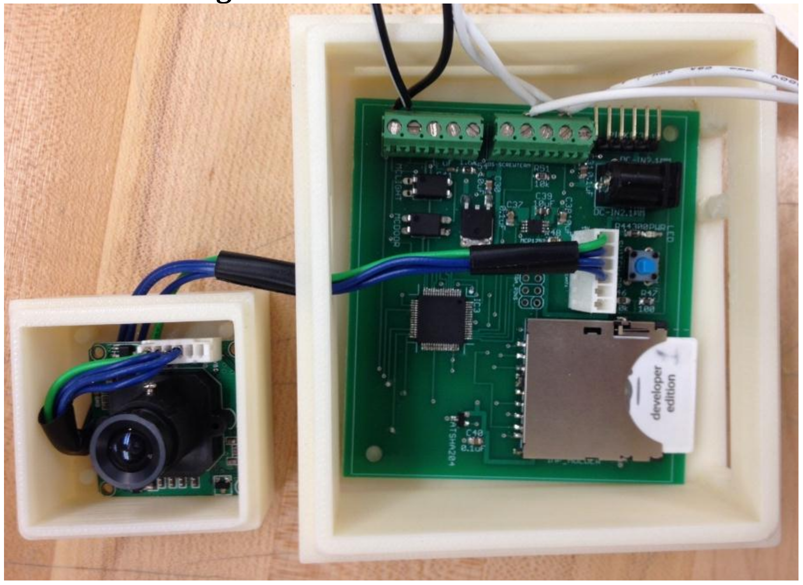


Figure 3.8.5. Boards inside of Housing

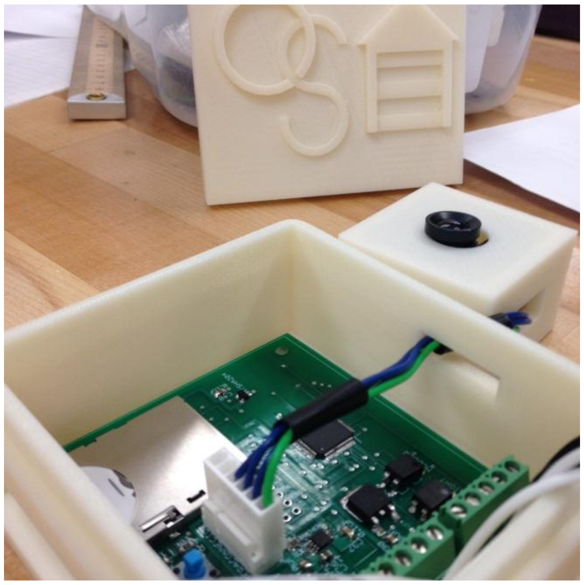


Figure 3.8.6. Boards inside housing with camera lid on

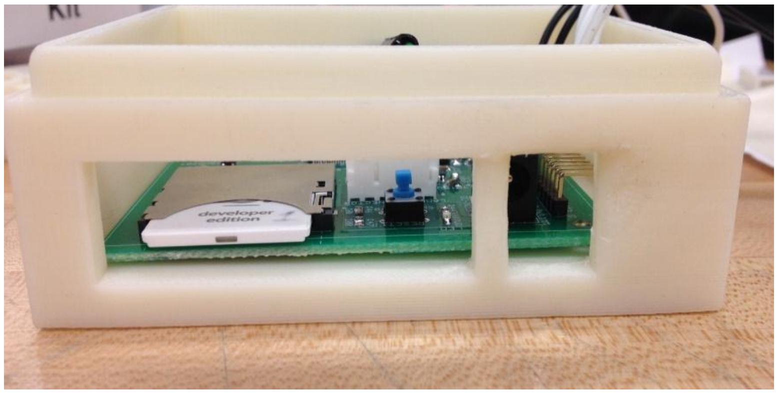


Figure 3.8.7. Main board housing from the side showing access to Imp, reset button, and power

Printed Circuit Board Design

An essential part of the project that allows the subsystems to become fully integrated is the custom printed circuit board. Figure 3.8.8 shows the full Eagle schematic for the Open Sesame board with each of the major sections boxed and labeled. Many of the key parts have been described as part of the hardware for the respective subsystem. In addition to these subsystem-specific circuits, portions are essential for general functionality and those are described here.

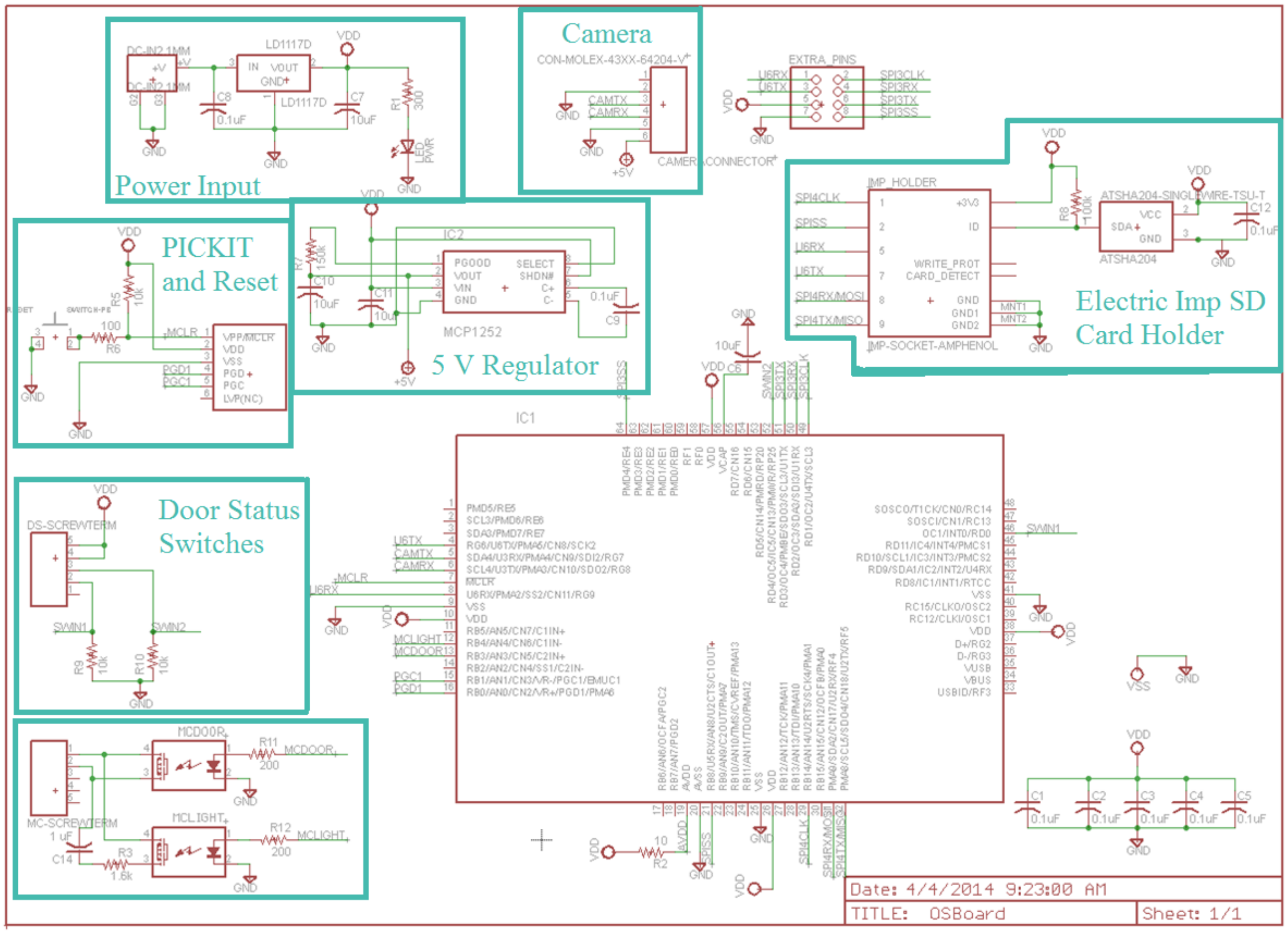


Figure 3.8.8. Full Eagle Schematic for Custom Open Sesame Board with Circuits Labeled

One of the requirements that the board helps the Open Sesame product meet is the ability to power easily. The goal is for the user to be able to power the device from a standard wall outlet because the layout of a garage will typically have two plugs on the ceiling, one for the motor itself and the other that this product can plug into. This eliminates the need for any batteries that would have to be charged or changed. Any standard wall wart capable of outputting between 3 to 5 V and with the proper jack can be used to power the device. The circuitry that allows for this can be seen in Figure 1 in the area labeled "Power Input". This features the jack itself as well as the LD1117D voltage regulator with its C_{in} and C_{out} capacitors valued as suggested in the data sheet referenced in the Appendix 9.2. However, as can be seen in the board diagram in Figure 3.8.9., this voltage regulator is not supplied with a heat sink as recommended due to an oversight on the part of the designer. As a result, the regulator chip tends to heat up during prolonged use or during particularly strenuous activities such as send a picture. Planes are included around the MCP1252 charge pump as recommended by the data sheet in order to sink heat and minimize RF effects. An LED is also incorporated with a current limiting resistor to indicate when the board is supplied with power.

Other important features of the board are the PICKIT connector and circuitry and the attached reset button. The PICKIT connector allows the microcontroller to be programed with the necessary code before being sold to a user. The reset button can be used should the device ever begin malfunctioning due to an unanticipated failure or incorrect usage of the app.

Underneath the microcontroller, one can see the various decoupling capacitors required for correct functionality of the chip located in close proximity to be most effective. The schematic and board also show the inclusion of a set of eight pins that provide access to 3.3 V, ground, SPI 3 and UART 6. These were included in case the team decided to attempt different options for motor control. In final revisions to the board, this bank of pins could be left out along with the UART connections between the Electric Imp and microcontroller which were added as a precautionary measure, but were not necessary.

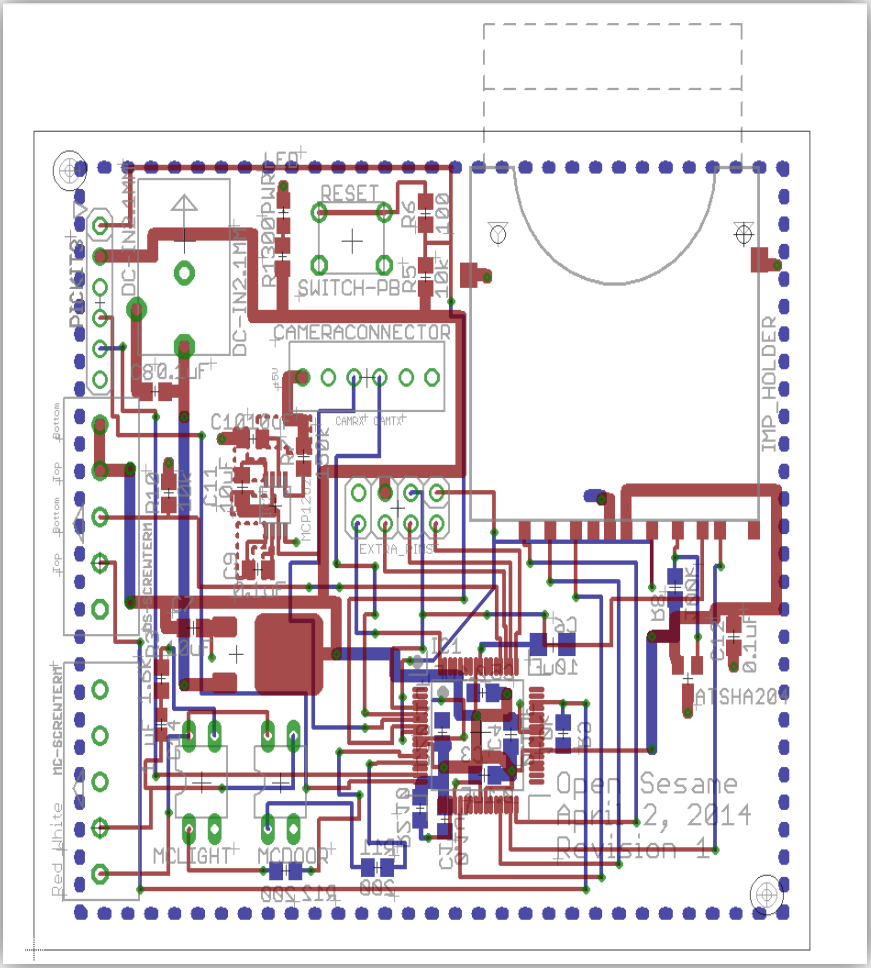


Figure 3.8.9. Full Eagle Board Image for Custom Open Sesame Board

3.9 Interfaces

Integrating the Android System

The Open Sesame app is the front-end of the Open Sesame product. It is the product that the user sees, which is why it is important to interface the app with the rest of the system for total functionality. Figure 3.9.1 shows all the components in Open Sesame and how they interact with each other. All communications with the garage door opener is done through the Imp cloud and to the Imp, except for downloading the picture. Since the Imp cloud cannot store images, a VPS provided by Notre Dame had to be used. The VPS has the server-side PHP language enable, so it is capable of storing images sending images. When the user pushes the capture button in the "CAMERA" tab, the app gets the image from the VPS. As seen in figure 3.9.1, there is two-way communication between all components except the VPS. As stated above, there were issues with posting an image onto the VPS. And when the app downloads the image, there is only a one-way communication where the VPS does not know the app is downloading an image. Two-way communication with the VPS will definitely be implemented on the server in order to create some sort of authentication and security. It is important for a server that is storing the user's data to know who is trying to download an image; therefore, if the server that is able to recognize a user's app is important.

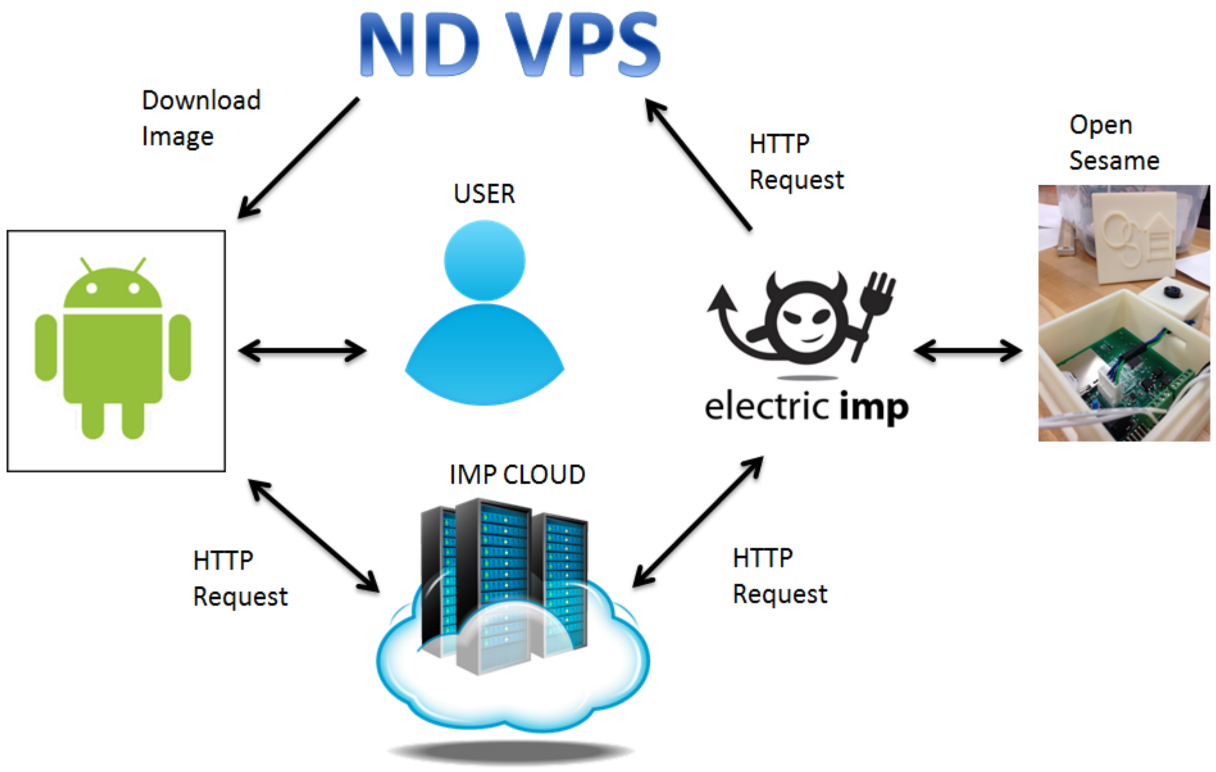


Figure 3.9.1. OPEN SESAME FLOW CHART

An Alternative Approach

While implementing the Open Sesame subsystems, an alternative approach to how we should implement the back-end of the product was suggested. The garage door and Imp side of the product remained the same. What could have been done differently was the web server and mobile app.

Figure 3.9.2 below diagrams the system of the alternative approach. Compared to Figure 3.9.1 above, there is only one point of communication or one node between the Imp/garage door and the mobile app. This webserver would be Node.js. Node.js is a server platform that is built on Chrome's JavaScript runtime that allows people to build fast, scalable network applications. Basically, Node.js would act as the main server that handles requests between Imp and the mobile app. On this Node.js webserver, there would be room to store images from the camera and it would also have a SQLite database that keeps a log of all the users that use Open Sesame. By keeping a log and saving data, the web server would be able to differentiate between different users that have the Open Sesame system installed in their home. This means the Open Sesame product is scalable as more customers buy the product and as more garage doors and mobile apps need to be handled. Node.js would handle all requests and keep track of all customers.

The mobile app in this design would be developed of JQuery Mobile, which is an HTML5-based user interface system designed to create website and mobile apps. JQuery Mobile would allow all types of smart devices to access the Open Sesame app. This means the Open Sesame app does not need to be developed separately for Android and iPhones. By creating the mobile app through JQuery, any device and access the app through their browser. The web browser would have all the capabilities that an Android app would have.

The interface between the three different subsystems would be simple. The Open Sesame garage door side would use HTTP requests to communicate with the Node.js server and the JQuery mobile app would use web sockets to communicate with the server.

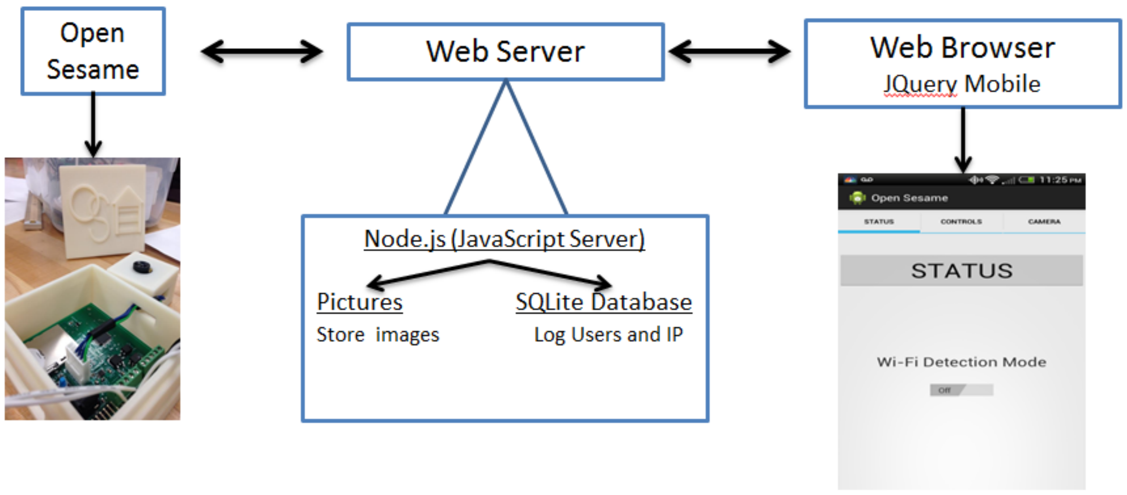


Figure 3.9.2. Node.js and JQuery Mobile

The main advantages of implementing the Open Sesame in this manner would be scalability and availability. Creating a centralized web server would allow easy management of multiple customers with the product. As opposed to the system in figure 3.9.1, there is only one server to manage. To reiterate the point, Node.js would be the brains of the system since it handles all communication. JQuery mobile allows any person with a smart device to use the Open Sesame app. The current Open Sesame app is available only to Androids.

The main reason Open Sesame did not take this approach was due to time constraints. The idea in figure 3.9.2 was suggested to the team more than half way through the project. Most of the subsystems described in figure 3.9.1 were already complete. There would have been no time to change the system to that in figure 3.9.2. Node.js and JQuery mobile are definitely options that will be explored in the future.

Imp & Server

The Electric Imp and the server created for the Open Sesame project communicated using PHP commands in a file called *upload.php* located on the Open Sesame server. The code within that file, which can be seen in Appendix 9.6, uses the *php://input* wrapper from PHP protocol to make a read-only stream that allows for the raw data to be read from the request body—the Electric Imp in this case. To get the contents from the Electric Imp, the *upload.php* uses the *file_get_contents()* function and opens a file with *fopen()* where the data can be placed. If the file already exists, it will modify it, and if the file does not already exist, a new one will be created. This is followed up by *file_put_contents* to put the data into the selected file and *fclose()* to close the modification of data.

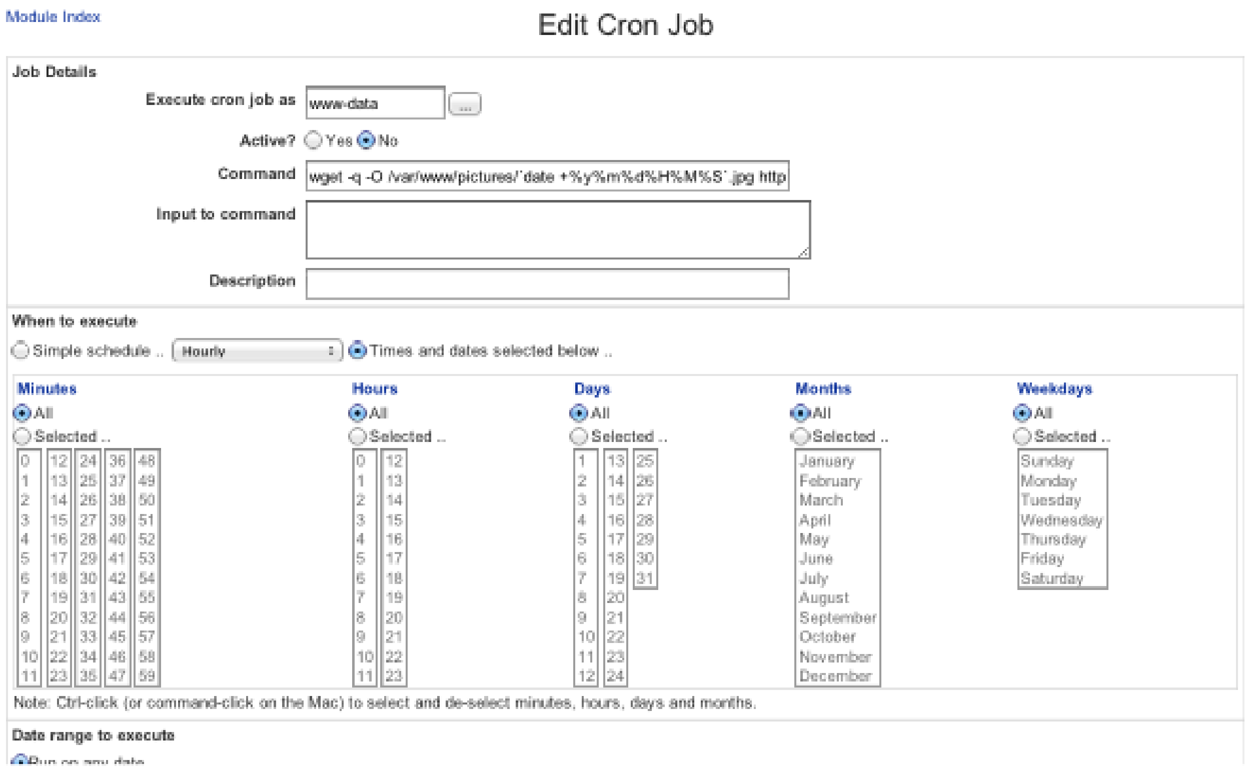
On the Electric Imp side of the server-Imp-interface, when the Electric Imp receives a command from the microcontroller, it accesses the URL where the *upload.php* file is located. From this file, it recognizes the request for data—an image—and uses *http.post()* to return an *httprequest* object primed to perform an HTTP/1.1 POST request to the given URL, which in this case has been set to http://10.36.251.234/pictures/testpic1.jpg. In order for the request to start, *sendsync()* must be called on the returned object to execute a synchronous HTTP request. There is the option to use *httprequest.sendasync()*, however it was desirable for this project to ensure that no other Squirrel code in the agent would be executed until the server returned, or until an error occurred.

The main unresolved problem of the Open Sesame project lies in the server to Electric Imp interface: the Electric Imp cloud is currently not able to save a file on Open Sesame's Notre Dame server. From the system testing, it has been determined that the Electric Imp can send images through the Imp cloud, and the images from the server can be given to the user from the mobile application. Therefore, it has been concluded that the problem is most likely an issue of network permissions and security. Since the server used for this project is a Notre Dame server, it is very specific in receiving inputs from secure sources, which the Imp cloud may not be. As Open Sesame did not have access to any servers that do not belong to Notre Dame, this theory was not able to be confirmed nor denied.

Prior to settling on using PHP protocol for the Electric Imp to server communication, Open Sesame attempted to use *wget()* as a means to get the images from the Imp onto the server. This was done by creating a Cron Job through Webmin, which can be seen in Figure 3.9.3. The actual command run is as follows:

```
wget -q -O /var/www/pictures/`date +%y%m%d%H%M%S`.jpg http://wx.nd.edu/images/wx-header.png
```

The concept of setting up a Cron Job to periodically check for new images was desirable because it would allow the user to access previous photos, which would be stored on the server for a certain amount of time and sorted according to the time stamps the Cron Job assigns to each image. The problem with using Cron Jobs to run the *wget* command is that it assumes the Electric Imp can output the images to a link separate from the server. In reality, the Electric Imp needs to find a request for data within the server, which it then writes directly onto the server. Once the image is on the server, the need for using *wget* no longer exists, except perhaps to rewrite the image names to include a time stamp. The PHP requests are still necessary for the Electric Imp to transfer the image in the first place.



Module Index

Edit Cron Job

Job Details

Execute cron job as: www-data

Active? Yes [✓ No]

Command: wget -q -O /var/www/pictures/`date +%y%m%d%H%M%S`.jpg http

Input to command:

Description:

When to execute

Simple schedule .. Hourly [✓ Times and dates selected below ..]

Minutes	Hours	Days	Months	Weekdays
[✓ All] Selected ..	[✓ All] Selected ..	[✓ All] Selected ..	[✓ All] Selected ..	[✓ All] Selected ..
0–59	0–23	1–31	January, February, March, April, May, June, July, August, September, October, November, December	Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday

Note: Ctrl-click (or command-click on the Mac) to select and de-select minutes, hours, days and months.

Date range to execute

Run on any date

Figure 3.9.3. Cron Job Configurations

4 System Integration Testing

4.1 Testing of Integrated Set of Subsystems

To test the integrated set of subsystems the designed circuit was hardwired to the screw terminals on the motor, as well as attached to the Wall Wart power supply, with the Electric Imp card programmed and placed on the board. Once the microcontroller had been programmed using the PicKit3, the app was then utilized to test if the entire system had been successfully integrated. So the user pressed open/close on the app and the motor would turn on and reverse its current state if the system was successful.

To further test communication with the Electric Imp, the app was used to test the camera by pressing the capture button on the app. The Imp agent was used to detect if the camera had taken a picture and sent the signal to the Imp. If the picture had sent the image to the Imp then the agent IDE would reflect that and the data would be sent on to the microcontroller.

Another test included the Wi-Fi mode on the app. To test the success of the design the Wi-Fi mode on the app was switched on and if working properly, the phone would recognize a connection to the “home” Wi-Fi (ND-secure) and inform the user that the garage door would be opening, and the motor would turn on and open the door. To test the user leaving their home the Wi-Fi on the phone was turned off (then out of ND-secure) and if successful the app would alert the user that they had exited the home Wi-Fi, and the garage door motor would turn on to close the door.

To test the functionality of the included safety design, the optical eyes were tested to make sure that the safety feature had been preserved. An obstruction was placed between the two optic eyes (typically a hand or foot) and the system was sent a signal (through the app) to close the already open garage door. The system successfully retained the feature when the motor began to run but stopped before the chain started to pull the door closed, and rather blinked on and off before stopping the attempt to close the door.

Finally, both the wall control and the remote were tested. Both were able to open and close the door with via communication to the motor. The wall panel was also able to use the vacation mode (by pressing a button on the panel) and to turn the light on and off (again by pressing a button).

4.2 Meeting the Design Requirements In Testing

The design requirements were: to retain the original garage door functions, create a functional Android app based on the app requirements, establish communication between the Imp and the microcontroller, and establish communication between the Imp and the Android app. These requirement would allow the user to experience the product without actually seeing what happens in the background. Having the Android app communicate with the Imp and the Imp to the garage door demonstrates full integration of all the subsystems.

In order for the system to meet the requirements of retaining the original garage door functions, the wall control, the remote control, and the eye sensors must all be functional. During the tests, the remote control used to open and close the garage door showed full functionality. The button on the remote is set to toggle, so by pushing the button, the garage door would move in the opposite direction. We were able to retain this functionality. The wall panel is the stationary version of the remote control and it was able to toggle the garage door when the toggle button was pressed. In addition, the button on the wall control that controlled the light bulb was fully functional. Finally, the eye sensors were able to detect when someone was in the line of sight during the tests. When testing the original safety feature, the garage door did not open when the eye sensors were triggered, which demonstrated that the Open Sesame system was able to retain original motor functions.

Communication between the Android app and the Imp were mainly tested by observing the Imp agent, which outputted to a console based on the signal sent from the Android app. The functions on the Android app were tested by seeing if the HTTP requests were received by the Imp agent when a button was pushed. For example, if the “STATUS” button was pushed on the app, the Imp agent would display a response that corresponded to that specific button. And once that signal from the app was recognized, the Imp agent would send the appropriate signal to the Imp itself. During our tests, the Imp agent displayed the correct output for each corresponding button that was pressed on the Android app. The Imp was then able to relay that command to the microcontroller, which in turn sent that signal to the garage door opener. The system would be fully functional when each button on the Android app was returned the proper message. For instance, if the “OPEN” button was pushed, the corresponding output message for open would be displayed on the IMP agent and the garage door would open. Then when the status is checked, the app would display to the user that the door is in fact opened. In cases when the garage door did not respond to the app, the Imp Agent would reveal that it did not receive any signals from the app.

As stated above, the app was tested by observing the functionality of each individual button. The Imp agent showed that it was in fact receiving signals from the buttons which means communication was established between the app and the Imp. The Wi-Fi detection mode test demonstrated that when the Wi-Fi detection mode was turned on, the app automatically sent the "OPEN" signal to the Imp and the garage door opened. And when the Wi-Fi network changed from the home network, the app sent the "CLOSE" signal to the Imp and the garage door closed. When the detection mode was turned on, the app did not respond to any Wi-Fi connection changes.

The only feature in the overall system that did not meet the design requirements was the camera feature. The app was able to pull an image from a web server, but the Imp was not able to post the image. While testing the camera subsystem, we were able to verify that the camera was able to take an image since the image file was not empty. However, when the Imp tried to post the image on the VPS, the newly created image file was empty. This meant the app would only be pulling an empty file and therefore, a blank image.

In summary, the tests demonstrated that the Open Sesame system met all but one system requirement, which was the camera feature. The app was able to successfully communicate with the Imp cloud and the VPS and the Imp was able to communicate with the microcontroller. During the tests, the app demonstrated the system requirements of being able to display the door status, open and close the door, and enable Wi-Fi detection mode. From the user's perspective, there is seamless integration between the systems with the exception of the camera feature.

5 User Manual/Installation Manual

5.1 Assumptions for Marketed Project

The user manual as described assumes that the supplier of the Open Sesame system is the first to obtain the Electric Imp. It will be this supplier that "Blinks Up" the Imp for the first time, loads the necessary code, and obtains the agent URL necessary for using the Open Sesame Android Application before shipping the product to the user. Before BlinkUp, the username and password will be assigned by the supplier. The password can later be changed, but the username is permanent. BlinkUp would then be performed, and the code loaded onto the Electric Imp. When the code is being loaded, the agent URL assigned to the individual Imp would also be recorded. This information is necessary for a separate log in on the Android App. It would be necessary, as part of the security features described later, to include a login for the Android App. This would also include a pre-assigned username, to match the URL for the Imp. Once the user is in control of Imp, they will be responsible for changing the password to retain security. At present, the same login information would be used for all users of a signal garage door.

5.2 How to Setup the Open Sesame

Here is a description of the setup for the Open Sesame system. Before you begin, please collect your home Wi-Fi network login information and cell phone. The first steps will require setting up your system to connect to the internet. Before you begin, check that your garage is within your wireless network's range by checking your connection with your cell phone. The initial setup can be performed in your home, and are recommended to be done with your internet router nearby.

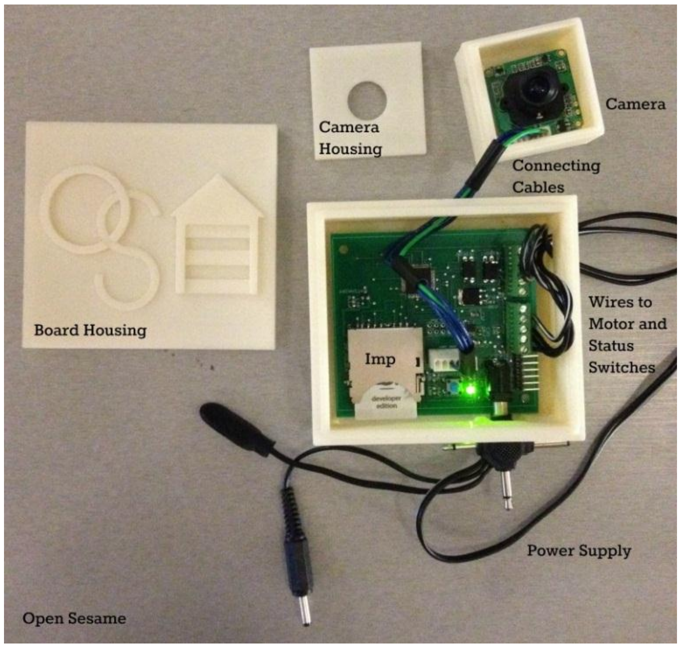
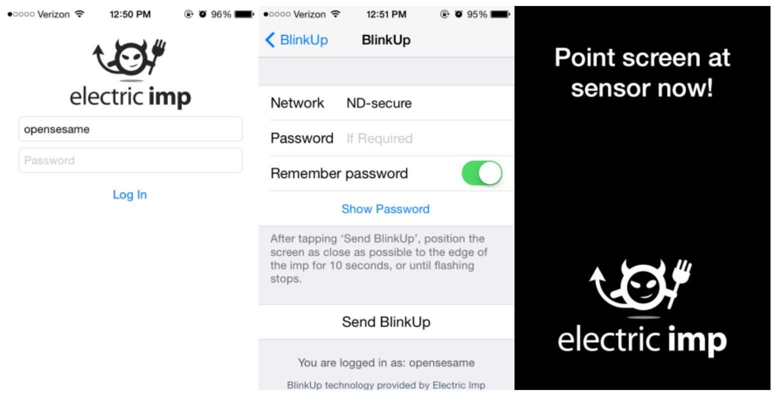


Figure 5.2.1. Components included in your Open Sesame

1. Open Sesame Cell Phone App
    a. The Open Sesame control application can be found in the Google Play store for Android. Eventually this system will be expanded for iPhone, and will also be available through iTunes. If you have an iPhone please visit the Open Sesame website where you will log in as you would on a cell phone app. This can also be done on your computer.
    b. Log in using login username and password supplied with the rest of your Open Sesame.
    c. After logging in, please change your password to improve security.
    d. When you first log in, the system will ask for your network name. This is for the Wi-Fi detection mode and will serve as your proximity sensor. Please use proper capitalization and type your network name as you would see it listed on your cell phone. Note that Wi-Fi detection is not available using the website interface.
    e. Once you have entered the necessary information you should be greeted with a welcome screen.
2. BlinkUp
    a. These next steps will describe how to connect your Open Sesame to the internet.
    b. Download the Electric Imp app from the Google Play Store or iTunes. This is separate from the Open Sesame app.
    c. Connect your circuit board to power. It does not have to be in the housing yet.
    d. Insert the Electric Imp into the SD card slot on the board. If power is connected it should immediately begin blinking red and orange.
    e. Open the Electric Imp app and log in with the information given to you. This will not be the same login you used for the Open Sesame app.

f. Once logged in you will see a screen asking for a network name. Choose “Other Network”.
g. You will be prompted to provide your network name and password. Type in the information exactly.
h. Prepare for BlinkUp by covering the top of the Electric Imp where the light comes through.
i. Select “Send BlinkUp” on the open app and press the edge of the chip up to the phone screen.
j. After BlinkUp is complete the Electric Imp should blink green. If it does not, continue to troubleshooting. Once BlinkUp is completed, unless you change your router name or password, the Imp will always automatically connect to your router without the need to BlinkUp. If you have changed your router name or password, follow the instructions to BlinkUp over again with the new information.
k. At this point the buttons on the Open Sesame should be functioning and should be tested. Go into the app and select “Check Status”. If the word “refresh” appears you may continue setup, otherwise continue to troubleshooting.



opensesame

Network ND-secure

After tapping ‘Send BlinkUp’, position the screen as close as possible to the edge of the imp for 10 seconds, or until flashing stops.

You are logged in as: opensesame

BlinkUp technology provided by Electric Imp

Figure 5.2.2. Screenshots from the Electric Imp iPhone App to assist in BlinkUp

3. Connecting Switches
   a. Before moving into the garage, check that the switches are functioning.
   b. Remove power to the board.
   c. Thread the wires for the switches through the opening on the housing. The board does not need to sit inside the housing yet, but threading the wires through now will prevent redoing this step later.
   d. Match the colors on the wires to the colored ports on the circuit board. Using the included screwdriver secure the connections.
   e. Connect power to the board.
   f. Open the Open Sesame app.
   g. Place the magnet near one of the switches.

    h. Check the status of the door. It should read open or closed. If it does not, continue to troubleshooting. If it does, move the magnet to the other switch and check the status again. It should now read the other status. Keep track of which color is open and which is closed.

4. Connect Camera
    a. Remove power to the board, thread the connecting cables through the ports on the camera and board housing.
    b. Connect power again and allow Electric Imp to connect.
    c. Open the Open Sesame app and select the capture mode. Take a picture and wait for confirmation that the command was sent. In just over a minute you should be taken to a page where your image can be seen.
    d. If the image does not show up, proceed to troubleshooting.
5. Setup switches
    a. Making note of which switch was open and which was closed, move into your garage.
    b. The wires should be long enough to fit on a standard garage door, but first check if this is true. It may be necessary to extend the wires if they cannot reach the door.
    c. Beginning with your garage door open, take the switch marked as open and find a position on the garage door where a switch can be attached most easily. This may vary depending on your garage door. While it is in this position, attach the switch to the rail side of the door, and the magnet to the door.
    d. While it is in this same position, check using the Open Sesame app what the status of the garage door is. It may be necessary to adjust the switch slightly depending on the distance. It is important to have a solid connection between the magnet and the switch otherwise the status will never be read properly. Once this connection is made, and the status can be read properly using your phone, secure the wire connecting the switch, and change the status of your door using your wall panel.
    e. Repeat this process for the closed position. The Switch can be placed anywhere on the door, but it may be useful to use the same magnet as before to create the closed door connection. After the door status reads closed from your phone, you may continue on to connecting the motor.
6. Connect to and test motor
    a. Remove power from your circuit board.
    b. By tracing the wires coming off of the wall panel, there should be two screws coming out of the garage door opener.
    c. On the board there is a white and a red port. The wire from the white port goes to the white screw and the wire from the red port goes to the red screw. Taking care to thread the wires through the opening on the housing, connect these wires as

before, using the screwdriver, and wrapping the other end of the wire around the screw being careful not to short the two screws.

    d. After the connections are made, apply power to the board and through the phone app attempt to change the status of the door. If it works, you're done with setup! If not, proceed to troubleshooting.

7. Finalizing Setup
    a. Place the board inside the housing; this will require removing power temporarily. Adjust the wires to allow the board to fit securely. Do the same with the camera housing. Once everything is in the case, use the Velcro strips provided to connect the housing to your garage door opener in a way that is convenient for accessing the reset button, as well as allowing the power cord to reach the board without stretching. After the board is in place, attach the camera to the housing or the opener so it is facing your garage door. You may need to take some test shots to ensure it is pointing in the proper direction. After everything is placed, provide power to the board, and enjoy your peace of mind.

5.3 Using the Open Sesame Android Application

1. Startup
    a. As is described before, the Android App must be configured for your specific Imp. This is done partially by the supplier, however you, the user, must identify the home network after the initial log in.
2. Options
    a. Check Status
        i. Check the status by going to the status tab. It will be necessary to press the button twice for every status read. This is the nature of the status. The first time the command is sent to check the status, and the second reads back the status to the user's phone.
    b. Change Status
        i. While in this tab you may choose to open or close your garage door. If your door is already closed, it will give you an error message when you try to close it. Likewise if your door is open and you try to open it.
    c. Take a Picture
        i. In this tab you can choose to take a picture. Ensure that the camera is properly plugged into the Open Sesame board before taking a picture; otherwise an error will be returned. Once the button is pressed, it can take up to a minute for the information to be sent back to your phone. A picture should then appear on your phone screen.
    d. Wi-Fi Detection Mode
        i. By selecting Wi-Fi Detection mode, under the status tab, the user's cell phone will send instructions based on connecting to and

disconnecting from the user's home network. This mode is design for driving to and from work, the store, or anywhere. Once enabled, as you leave your home network you phone will automatically send the command to close your garage door. Likewise as you enter the network range your phone will automatically send the command to open the door for you. This prevents fumbling for remotes as you enter your driveway. It is completely optional, and can be turned off for doing yard work or going for a walk where you may go in and out of range.

5.4 Troubleshooting the Open Sesame

1. BlinkUp
    a. The Imp is flashing a series of lights I don't understand
        i. Check out the documentation provided by Electric Imp on this subject
        ii. http://electricimp.com/docs/troubleshooting/blinkup/
    b. Errors setting up Wi-Fi
        i. Check that your range is extending far enough to pick up in your garage
        ii. Test with multiple devices (your phone, tablet, laptop) the signal doesn't have to be strong, but a stronger signal will respond more quickly
2. Status Switches
    a. Transition
        i. Chances are one of your switches is not setup properly. While in the open or closed state, check with your Open Sesame app to see what the status is reading. If it's "in transition" place another magnet near the switch and check again. If the status is correct, your switch is too far from the magnet and must be moved. Alternatively a more powerful magnet (not included) could be used to expand this system.
    b. Always open/closed
        i. Check your connections to the board. If one of the wires is loose, or if a wire is crossing, the logic will be incorrect and cause false readings.
3. Error Changing the Status
    a. Motor does not respond
        i. Before assuming there is a problem with the board, check that your wall panel and remote are still functioning. There is a chance that vacation mode was turned on without your knowledge.
        ii. Otherwise, check that the connections between the circuit board and the motor are solid. If they are at all loose, tighten them
4. Phone App
    a. Unknown Status

   i. If after connecting everything the app returns “unknown status” go out and reset the circuit board by hitting the button next to the Electric Imp.

b. Camera Failed to Take a Picture
   i. Reset your board by hitting the button next to the Electric Imp and try again.
   ii. Check to see that all of the wires connecting the camera to the board are still connected.

5.5 Security

If you feel that your Open Sesame is no longer secure and someone else has access to your garage door controls, immediately unplug your Open Sesame system. It is likely that the Electric Imp has been compromised. Contact Open Sesame and we will provide you with new log in information, and we will adjust the URLs to operate your phone app. You will be required to BlinkUp the Imp using a new username and password, and the login for your phone app will also change. This is all for your safety. If you would feel more comfortable you can send back your Open Sesame for a replacement (feel free to keep the switches in place, they do not pose a security threat).

6 To-Market Design Changes

Estimated Cost

The costs for the prototype created are relatively low, especially considering the few parts necessary to the cost. The manufactured circuit board cost at least $100 for a set of only five with a one-week wait. The Electric Imp SD card is another $30, and the box that encases the entire circuit has an estimated cost of around $12 (this is the average cost of a Raspberry Pi case, which is roughly the size to encase the circuit board designed). The bell wire used to hardwire the circuit to the motor costs roughly $10 for 25 feet. Therefore the total cost for the prototype is approximately $152.

If the product were to be mass-produced, however, the total cost could be significantly reduced, particularly in terms of the designed circuit board. Increasing the quantity to 150 reduces the cost to $7.46 for a one-week wait. Assuming that for mass production this process would most likely be done more quickly the price increases slightly to $20.97 for same day processing. Of course more bell wire would need to be purchased (an approximated 18 inches per product), and a case for each would also need to be purchased (but may be at a slightly lower cost if ordered in bulk). This more than halves the cost per unit produced.

Future Improvements for the Electric Imp/Cell Phone Interface:

- Add time of last change to the door status, along with who made the change (user or door panel).
- Assigning a time frame when the door should be kept closed, such as when the user has gone to work.
- Specifying an amount of time the door may be left open after which the user is asked if they wish to close the door.

- Whenever the user attempts to close the door, but the door is backed up because of a break between the optic eyes, a notification is sent to the user complete with a picture to show the door failed to close.
- Including the option to “view last picture taken” along with a time stamp instead of taking a new picture.
- Notifying the user whenever a status change is made, not by them.

7 Conclusions

The Open Sesame product found a niche in an existing market to bring outdated but functioning products into this century, particularly in solving the problem of unsecured, unintelligent garage doors. By allowing the owner to remotely access key information about their garage door such as its current open or closed status and pictures of the door, users can feel more confident about the safety of their home and family. With the ability to control the door from anywhere with internet, even the most forgetful person can still close the garage while away from home or open it to allow in the repairman. Open Sesame also eliminates the need to go searching for the remote when one leaves or returns home with the Wi-Fi detection mode that can perform these actions automatically for the owner. This further adds to security because it decreases the likelihood of the door being left open when the owner is not home.

This unique combination of features allows Open Sesame to fill the need of a particular, neglected part of the garage door market. The systems on the market that serve to add intelligence to the garage door are complete packages that require total replacement of the existing motor and controls. This greatly adds to their expense and the installation process. It is also mostly unnecessary as the motors themselves have long lifetimes and would otherwise not need to be replaced. From speaking with potential customers, it is clear that these are hindrances to upgrading garages. On the contrary, Open Sesame is designed to be used with an existing garage door motor system. This greatly reduces the cost and installation making it attractive to potential buyers who just want the add intelligence and security and do not need a new motor. A purchaser can have the desired features without any extraneous changes. For these reasons, Open Sesame would be a competitive product even though the idea is not wholly original.

The final prototype developed by the team succeeded in meeting the vast majority of the requirements that were determined necessary to address the problem of unsecured, unintelligent garages. Besides for the exceptions explained in this report such as the nonfunctional Electric Imp to server communication, the prototype version is capable of performing the other functions and these were tested and demonstrated. In addition to ideas for how these issues could be addressed, the Open Sesame team has thoughts for improvements and add-ons that could be incorporated before bringing the product to market and these could further enhance the user experience.

8 Acknowledgements

The Open Sesame team would like to thank the many people who served as references throughout the course of this project and who thus contributed to the success of the prototype. Thanks are extended to Professor R. Michael Schafer for helping in the development of the project and for the advice he gave on a wide-variety of topics from board design to Wi-Fi enabling options to good design processes. He was also instrumental in provide essential parts not only for the board, but also for the demonstration, including a complete, fully-functional garage motor and system. Dr. Robert Stevenson assisted the project during both the exploration of RF options and the attempted implementation of the php code on the server. Yaakov Sloman provided information about servers that was useful during the development of the prototype and when considering future improvements. He also helped set up the server that was used to try uploading and downloading the camera pictures. The team would also like to thank the developers of the Electric Imp for their aid in Squirrel syntax and coding via the Electric Imp development forums. Lastly, Open Sesame would like to extend gratitude to Mrs. Katie Schafer for providing the inspiration behind the problem and project idea; the existence of this product is due largely to you.

9 Appendices

9.1 Appendix of Parts Used

Table 9.1.1. Parts Needed for PCB Design

Part Type	**Part Value**	**Package**
Power Circuit		
LED	PWR	P 0805
R1	300 Ω	R 0805
C7	10 uF	C 0805
LD1117D	(3.3 V)	DPAK
C8	0.1 uF	C 0805
DC IN 2.1 MM		DC-POWERJACK
PICKIT		
R5	10kΩ	R 0805
R6	100 Ω	R 0805
Push Button	Reset	B3F-10XX
PICKIT3 Connector		PICKIT3
Voltage Divider		
MCP1252	(Positive-Regulated Charge Pump (3.3x5.0))	MSOP8
C9	0.1 uF	C0805
C10	10 uF	C0805
C11	10 uF	C0805
R7	150k	R0805
Door Status		
R9	10 k	R0805
R10	10 k	R0805
DS-SCREWTERM	SCREW-TERM-1X5-3.81 (SCREW-TERM-1X5	TYCO-3.81MM-5
Motor Control		
R11	200	R 0805
R12	200	R 0805
R3	1.6 k	R 0805
C14	1 uF	C 0603
MC-SCREWTERM	SCREW-TERM-1X5-3.81 (SCREW-TERM-1X5	TYCO-3.81MM-5
Relay 1	MCDOOR	DIL04
Relay 2	MCLIGHT	DIL04
Camera		
Molex Connector 6 pin	CON-MOLEX-43XX-64204-V (CON-MOLEX-43XX-6)	WM-4204
Extra Pins		

Pin Connector	PINHD-2x4	2X04
Electric Imp		
IMP_HOLDER	IMP-SOCKET-AMPHENOL (IMP-SOCKET)	SDCARD-AMPHENOL1010031368
C12	0.1uF	C 0805
R8	100k	R 0805
ATSHA204	TSHA204-SINGLEWIRE-TSU-T (ATSHA204-SINGLEWIRE)	SOT23
Capacitor Bank		
C1	0.1uF	C 0805
C2	0.1uF	C 0805
C3	0.1uF	C 0805
C4	0.1uF	C 0805
C5	0.1uF	C 0805
Microcontroller		
R2	10 Ω	R 0805
C6	10 uF	C 1206
Microcontroller	PIC32MX795F512H	

9.2 Appendix of Links for Data Sheets

ATSHA204: http://www.atmel.com/Images/Atmel-8740-CryptoAuth-ATSHA204-Datasheet.pdf

AQY21 Relay: http://pewa.panasonic.com/assets/pcsd/catalog/aqy-dip-form-a-reinforced-isolation-catalog.pdf

HCS362: http://ww1.microchip.com/downloads/en/DeviceDoc/40189d.pdf

LD1117 D: http://pdf.datasheetcatalog.com/datasheet/stmicroelectronics/2572.pdf

MCP1252: http://ww1.microchip.com/downloads/en/DeviceDoc/21752B.pdf

9.3 Appendix of Microcontroller Code Files

configbitsrev8internal.h

```
/*
 * File:   configbits.h
 * Author: Mike
 *
 * Created on October 9, 2012, 1:50 PM
 */

#ifndef CONFIGBITS_H
#defineCONFIGBITS_H

/*
 * REv 8 boards.
 * resonator is 8 MHz
 * Will switch to internal if external not present or fails
 *  internal (FRC) clock

 peripher clock = at 10 MHz (80 MHz/8)

 */

#pragma config FNOSC = FRCPLL // Oscillator selection
#pragma config POSCMOD = HS // Primary oscillator mode
#pragma config FPLLIDIV = DIV_2 // PLL input divider (8 -> 4)
#pragma config FPLLMUL = MUL_20 // PLL multiplier  ( 4x20 = 80)
#pragma config FPLLODIV = DIV_1 // PLL output divider
#pragma config FPBDIV = DIV_1 // Peripheral bus clock divider 80 mhz
#pragma config FSOSCEN = OFF // Secondary oscillator enable
/* Clock control settings
*/
#pragma config IESO = ON // Internal/external clock switchover
#pragma config FCKSM = CSECME // Clock switching (CSx)/Clock monitor (CMx)
#pragma config OSCIOFNC = OFF // Clock output on OSCO pin enable
/* USB Settings
*/
#pragma config UPLLEN = OFF // USB PLL enable
#pragma config UPLLIDIV = DIV_2 // USB PLL input divider
#pragma config FVBUSONIO = OFF // VBUS pin control
#pragma config FUSBIDIO = OFF // USBID pin control
/* Other Peripheral Device settings
*/
#pragma config FWDTEN = OFF // Watchdog timer enable
#pragma config WDTPS = PS1024 // Watchdog timer post-scaler
```

```
#pragma config FSRSSEL = PRIORITY_7 // SRS interrupt priority
#pragma config DEBUG = ON


#pragma config          ICESEL          = ICS_PGx1    // ICE pin selection
#endif  /* CONFIGBITS_H */
```

Delaylib.h

```
#ifndef _DELAYLIB_H_
#define _DELAYLIB_H_
#include <xc.h>

/**********************************************************
header file for the  Delay routines
*/


/* delay routines*/
//void set_sys_clock(unsigned long val);
//unsigned long get_sys_clock(void);
//void set_pb_clock(unsigned long val);
//unsigned long get_pb_clock(void);
void delay_ms(unsigned long val);
void delay_us(unsigned long val);

#endif //DELAYLIB
```

kit32r7lib.h

```
#ifndef _KITSUPPORT_H_
#define _KITSUPPORT_H_
#include <xc.h>
#include <stdint.h>
#include <plib.h>
//#define _XTAL_FREQ 20000000UL
/**********************************************************
header file for the support routines for the kit board
* At present, included are:
* Delay routines
* serial (spi) LCD
* serial port routines
* switcher to allow stdout to go to LCD or serial port
```

```
*/

// LCD Function prototypes

/* specific to spi display */

void LCD_init(unsigned long rate);
void LCD_char(char val);
void LCD_display_on(void);
void LCD_display_off(void);
void LCD_clear(void);
void LCD_backlight(char val);
void LCD_contrast(char val);
void LCD_setpos(char row, char col);


/* serial I/O via usart 6 prototypes*/

unsigned char getu(void);  // get char
void putu(unsigned char val); // put char

void serial_init(unsigned long rate);


void set_output_device(unsigned char device);

/* delay routines*/
void set_sys_clock(uint64_t val);
uint64_t get_sys_clock(void);
void set_pb_clk(uint64_t val);
uint64_t get_pb_clock(void);
void delay_ms(unsigned long val);
void delay_us(unsigned long val);

#endif //KITSUPPORT_H
```

OpenSesameR1.c

```
/*
 * File:   OpenSesameR1.c
 * Author: asavela
 *
 * Created on April 3, 2014, 9:24 AM
 */
```

```
#include <stdio.h>
#include <stdlib.h>
#include <xc.h>
#include "configbitsrev8internal.h"
#include "kit32r7lib.h"
#include <plib.h>
#include <sys/attribs.h>
#include "Delaylib.h"
/*
 *
 */

//SPI #defines
#define SPI_INT_R IFS1bits.SPI4RXIF
#define SPI_INT_T IFS1bits.SPI4TXIF
#define ChipSelect PORTBbits.RB8
//SPI Functions
void SPIinitslave(void); //initializes the SPI
unsigned char do_SPI_slave(unsigned char VAL);

// Set Up Information for Camera, #defines and variables
#define VC0706_RESET  0x26
#define VC0706_READ_DATA 0x30
#define CAMERABUFFSIZ 100
#define VC0706_GEN_VERSION 0x11
#define VC0706_READ_DATA 0x30
#define VC0706_STOPCURRENTFRAME 0x0
#define VC0706_FBUF_CTRL 0x36
#define CAMERADELAY 10
#define VC0706_GET_FBUF_LEN 0x34
#define VC0706_READ_FBUF 0x32

typedef unsigned char boolean;
#define true    1
#define false   0

unsigned char serialNum = 0;
unsigned char bufferLen = 0;
unsigned char camerabuff[100];
int frameptr = 0;

// Camera UART Function Declarations
void serial_init3(unsigned long baudrate);
unsigned char readu3(void);
void writeu3(unsigned char input);
```

```
// Door Status #defines
#define Enable_Timer T2CONbits.ON
#define Timer_Flag IFS0bits.T3IF


int main(int argc, char** argv) {
    //Set up for switches
    int count = 0;
    int x = 0; //for light flash
    AD1PCFG=0xFFFF; //digital ports, (high voltage is 3.3V)
    TRISDbits.TRISD0 &= 1;// D0 input (top switch)
    TRISDbits.TRISD4 &= 1;// D4 input (bottom switch)
    //TRISE &= 0x0000; //set LED to output
    //LATE = 0xFFFF; //turn LEDs off

    //Timer set up
    //Enable_Timer = 1; //*enabling timer 2
    //T2CONbits.T32 = 1;
    //T2CONbits.TCKPS = 0b111; //* 1:256 prescale value
    //PR2 = 65535;
    //PR3 = 180; //*we want 45 seconds
    //Enable_Timer = 0;
    //Timer_Flag = 0;

    //Motor Control Set up
    TRISBbits.TRISB3 &= 0;// B3 out (controls door)
    TRISBbits.TRISB4 &= 0;// B4 out (controls light)
    LATBbits.LATB4 = 0;
    LATBbits.LATB3 = 0;

    // Old Motor Control pins
    //TRISDbits.TRISD1 &= 0;// d1 out (controls door)
    //TRISDbits.TRISD2 &= 0;// d2 out (controls light)
    //LATDbits.LATD1 = 0;
    //LATDbits.LATD2 = 0;

    //Camera Variables
    boolean resetresult;
    unsigned char getresult;
    unsigned char result[100];
    unsigned char version[16];
    unsigned char ColorStatus[8];
    unsigned char ImageSize[6];
    unsigned char TakePicture[6];
    unsigned char FrameLength[9];
    unsigned char Length[4];
    unsigned char ReadStart[5];
    unsigned char ReadEnd[5];
    unsigned char ResumeReturn[5];
    unsigned char Picture[16384];
    int i = 0;

    //SPI Set up
    unsigned char send = 0; // initializing send to 0, the hex data to send
    unsigned char read = 0; // the variable that hex reads will be assigned to
    SPIinitslave(); // initialization function to set up as slave
    SPI4STATbits.SPIROV = 0; // clearing the bit

    //Camera Set up
    serial_init3(38400);
    TRISDbits.TRISD1 &= 0;
    LATDbits.LATD1 = 1;
    LATDbits.LATD1 = 0;
    LATDbits.LATD1 = 1;

    //LATDbits.LATD1=1; //sets B3 High
    //LATDbits.LATD2=1; //sets B3 High
    while(1){
       send = 0x00;
       read = do_SPI_slave(send);

       switch (read){
          case 0xCA:
             //send = 0x01;
             //read = do_SPI_slave(send);
             //Turn light on
             LATBbits.LATB4=1; //sets B4 High, turns light on
             LATBbits.LATB3=0; //sets B3 Low

             //Take Picture
             frameptr = 0;
             writeu3(0x56);
             writeu3(serialNum);
             writeu3(VC0706_FBUF_CTRL);
             writeu3(0x01);
             writeu3(VC0706_STOPCURRENTFRAME);
             for (i=0; i<5; i++){
                TakePicture[i] = readu3();
             }

             //Get Frame Length
             writeu3(0x56);
```

```
writeu3(serialNum);
writeu3(VC0706_GET_FBUF_LEN);
writeu3(0x01);
writeu3(0x00);
for (i=0; i<9; i++){
    FrameLength[i] = readu3();
}
int k = 0;
for (k=0; k < 4; k++){
    Length[0] = FrameLength[5];
    Length[1] = FrameLength[6];
    Length[2] = FrameLength[7];
    Length[3] = FrameLength[8];
}

//Reading picture
writeu3(0x56);
writeu3(serialNum);
writeu3(VC0706_READ_FBUF);
writeu3(0x0C);
writeu3(0x00);
writeu3(0x0A);
writeu3(0x00);
writeu3(0x00);
writeu3(0x00);
writeu3(0x00);
writeu3(Length[0]);
writeu3(Length[1]);
writeu3(Length[2]);
writeu3(Length[3]);
writeu3(0x0B);
writeu3(0xB8);
for (i=0; i<5; i++){
    ReadStart[i] = readu3();
}
unsigned long Len;
Len = Length[0]*256^3 + Length[1]*256^2 + Length[2]*256 + Length[3];
//unsigned char Picture[Len];
for (i=0; i<Len-1; i++){
    Picture[i] = readu3();
}

for (i=0; i<5; i++){
    ReadEnd[i] = readu3();
}
```

```
        // Resume: required after every take and read
        writeu3(0x56);
        writeu3(serialNum);
        writeu3(VC0706_FBUF_CTRL);
        writeu3(0x01);
        writeu3(0x03);
        for (i=0; i<5; i++){
            ResumeReturn[i] = readu3();
        }

        // Wait for SPI and send length
        send = Length[2];
        read = do_SPI_slave(send); //Send 01 to get back length part 1
        send = Length[3];
        read = do_SPI_slave(send); //Send 02 to get back length part 2

        for (i=0; i<Len-1; i++){
            send = Picture[i];
            read = do_SPI_slave(send);
        }
        for (i=0; i<(16384-Len); i++){
            send = 0x00;
            read = do_SPI_slave(send);
        }
        // Ending signal
        send=0x00;
        read = do_SPI_slave(send);
        //LATBbits.LATB4=0; //sets B4 low, stops trying to turn light on
        break;
    case 0x0D:
        for (x = 0; x<4; x++){
            LATBbits.LATB4=1; //sets B3 High
            delay_ms(1000);
            LATBbits.LATB4=0;
            delay_ms(1000);
        }
        LATBbits.LATB3 = 1;
        send = 0x02;
        read = do_SPI_slave(send);
        break;
    case 0xCD:
        for (x = 0; x<4; x++){
            LATBbits.LATB4=1; //sets B3 High
            delay_ms(1000);
            LATBbits.LATB4=0;
            delay_ms(1000);
        }
        LATBbits.LATB3=1; //sets B3 High
        send = 0x03;
        read = do_SPI_slave(send);
        break;
    case 0xD5:
        //send = 0x04;
        //read = do_SPI_slave(send);
        LATBbits.LATB3=0; //sets B3 Low
        LATBbits.LATB4=0;
        if (PORTDbits.RD0!=1 & PORTDbits.RD4!=1){ //door in limbo
            //Timer_Flag = 0;
            //Enable_Timer = 1;
            //while(PORTDbits.RD0!=1 & PORTDbits.RD4!=1){
                //if (!Timer_Flag){
                    //LATE = 0xFFFF; // all off, transition
                    send = 0x05;
                    read = do_SPI_slave(send);//}
               // else if (Timer_Flag){ // error state
                   // LATE = 0x0000; //all on!
                   // send = 0x03;
                   // read = do_SPI_slave(send);}
            //}
            //Enable_Timer = 0;
            //Timer_Flag = 0;
        }
        else if (PORTDbits.RD0!=0 & PORTDbits.RD4!=1){
            LATE = 0x00F0; // first half on, door closed
            send = 0x00;
            read = do_SPI_slave(send);}
        else if (PORTDbits.RD0!=1 & PORTDbits.RD4!=0){
            LATE = 0x000F; // second half off, door open
            send = 0x01;
            read = do_SPI_slave(send);}
        else if (PORTDbits.RD0!=0 & PORTDbits.RD4!=0){
            LATE = 0x0000; //all on!, error state
            send = 0x04;
            read = do_SPI_slave(send);}
        break;
    default:
         LATBbits.LATB3=0; //sets B3 Low
         LATBbits.LATB4=0;
        //send = 0x06;
        //read = do_SPI_slave(send);

        break;
```

```
            }
        //LATBbits.LATB3=0; //sets B3 Low
        LATBbits.LATB4=0;
        }

    return (EXIT_SUCCESS);
}

// UART Functions
void serial_init3(unsigned long baudrate){
    U3MODEbits.ON = 1;
    U3MODEbits.BRGH = 1;
    U3STAbits.URXEN = 1;
    U3STAbits.UTXEN = 1;
    U3BRG = (80000000/4/baudrate) - 1;
}

unsigned char readu3(void){
    while(1){
        if(U3STAbits.URXDA == 1){
            unsigned char received = U3RXREG;
            return(received);
        }
    }
}

void writeu3(unsigned char input){
    while(1){
        if(U3STAbits.UTXBF == 0){
            U3TXREG = input;
            return;
        }
    }
}

// SPI Functions
void SPIinitslave(void){
    int rData;
    //IEC1bits.SPI4EIE = 0; // disable interrupts
    SPI4CONbits.ON = 0;
    rData = SPI4BUF; //clearing the receive buffer
    //SPI4CONbits.ENHBUF = 0; // don't run enhanced mode (defaulted to 0 already)
    SPI4CONbits.CKP = 0; //must match imp
    SPI4CONbits.CKE = 1; //must match imp
    SPI4CONbits.MSTEN = 0; //making slave
    //IPC8bits.SPI4IP = 00;
    //IEC1bits.SPI4EIE = 1; // enable interrupts
    //IEC1bits.SPI4RXIE = 1;
    //IEC1bits.SPI4TXIE = 1;
    //SPI4CONbits.SSEN = 1; //enabling slave enable
    SPI4CONbits.SRXISEL = 0b01;
    SPI4STATbits.SPIROV = 0; // clearing the bit
    SPI4CONbits.MODE32 = 0; //set to 8 bit data
    SPI4CONbits.MODE16 = 0; //set to 8 bit data
    //SPI4CONbits.SSEN = 1;
    SPI4CONbits.ON = 1;
    }


unsigned char do_SPI_slave(unsigned char VAL){
    unsigned char read;
    SPI_INT_R = 0;
    //SPI_INT_T = 0;
    SPI4BUF = VAL;
    //while(!SPI4STATbits.SPITBE){}
    while(!SPI_INT_R){}; //checking if the receive buffer is full
    //while(!SPI4STATbits.SPIRBF){}
    do{
        read = SPI4BUF;
        SPI_INT_R = 0;}
    while(SPI_INT_R);
    //while(SPI4STATbits.SPIRBF){};
    return(read);
}
```

Delaylib.c

```
#include "Delaylib.h"
#include <xc.h>


/*Default clock values */
unsigned long FCY = 80000000UL;
unsigned long PBCLK = 10000000UL;

/* delay routines */


void set_sys_clock(unsigned long val)
{
  FCY = val;
}
unsigned long get_sys_clock(void)
{
  return FCY;
}

void set_pb_clock(unsigned long val)
{
  PBCLK = val;
}

unsigned long get_pb_clock(void)
{
  unsigned long temp;
  temp = OSCCON >> 19;
  temp &= 0X03;
  return (FCY >> temp);
}

void delay_ms(unsigned long delayms)
{
  unsigned int tWait, tStart;

  tWait = (FCY / 2000) * delayms;
  tStart = ReadCoreTimer();
  do
  {
    // do something
    _nop();
  } while ((ReadCoreTimer() - tStart) < tWait); // wait auto negotiation start

}

void delay_us(unsigned long delayus)
{
  unsigned int tWait, tStart;
  tWait = (FCY / 2000000) * delayus;
  tStart = ReadCoreTimer();
  do
  {
    // do something
    _nop();
  } while ((ReadCoreTimer() - tStart) < tWait); // wait auto negotiation start

}
```

9.4 Appendix of Electric Imp Code Files

Electric Imp Agent Code

```
const MY_SERVER_URL = "http://10.36.251.234/pictures/upload.php";

requests <- []
jpeg_buffer <- null

local DoorStatus = 9;
local TimerCount = "0";
local lastChanged = 0;
local Datatrue = 0;

http.onrequest(function(request, response) {
   if ("pic" in request.query)
   {
    device.send("spy",request.query["pic"].tointeger()); //commented out to
take out time
    response.send(200, "Taking Picture go to:
http://10.36.251.234/pictures/testpic1.jpg");
    requests.push(response);
    response.header("Location",
"http://10.36.251.234/pictures/testpic2.jpg");
   }

   else if ("door" in request.query)
   {
        if (request.query["door"] == "1"){
            if (DoorStatus == 1){
                response.send(200,"Door is already open.");
            }
            else if (DoorStatus == 0){
                response.send(200,"Door status will be updated.");
                device.send("open",1);
            }
            else if (DoorStatus == 5){
                response.send(200,"Door is in transition, attempting to
open.")
                device.send("open",1);
            }
            else {
                response.send(200,"Error, check status before trying again.")
            }
        }
        else if (request.query["door"]=="0"){
            if (DoorStatus == 0){
                response.send(200,"Door is already closed.");
            }
            else if (DoorStatus == 1){
                response.send(200,"Door status will be updated");
                device.send("close",1);
            }
            else if (DoorStatus == 5){
                response.send(200,"Door is in transition, you may only 
attempt to open.")
            }
            else {
                response.send(200,"Error, check status before trying again.")
            }
        }
        else {
            response.send(200,"error in updating status");
        }
   }

   else if ("status" in request.query) {
       if (Datatrue==1){
            if (DoorStatus ==0){
                    response.send(200, "Door is closed, your family and home 
are safe.");
                }
            else if (DoorStatus == 1){
                    response.send(200, "Door is open! CLOSE IT NOW!!!");
            }
            else if (DoorStatus == 2){
               response.send(200, "Say cheese we're taking your picture.");
            }
            else if (DoorStatus == 5){
                response.send(200, "In transition, you may only attempt to 
open the door. \nCheck again in a few seconds, or take a picture to see if it 
is stuck.");
               }
            else if ((DoorStatus == 3)||(DoorStatus == 4)){
               response.send(200, "In transition\ncheck again in a few 
seconds, or take a picture to see if it is stuck.");}
            Datatrue = 0;
       }
       else {
           response.send(200, "refresh");
           Datatrue = 1;
       }
       device.send("doorstat",1);
       }
    else {
        response.send(200, "for picture add '?pic=1', to open door add 
'?door=1', to close door add '?door=0', and to check status of door add 
'?status'");
       }


    });


device.on("jpeg_start", function(size) {
    jpeg_buffer = blob(size);
});

device.on("jpeg_chunk", function(v) {
    local offset = v[0];
    local b = v[1];
    for(local i = offset; i < (offset+b.len()); i++) {
        if(i < jpeg_buffer.len()) {
            jpeg_buffer[i] = b[i-offset];
        }
    }
});

device.on("jpeg_end", function(v) {
    local s = "";
    foreach(chr in jpeg_buffer) {
        s += format("%c", chr);
    }
    //server.log(s);
    local request = http.post(MY_SERVER_URL, {}, s);
    local response = request.sendsync();

    foreach(response in requests) {
        response.header("Location",
"http://10.36.251.234/pictures/testpic2.jpg");
        response.send(302, "Found\n");
    }
    server.log(format("Agent: JPEG Sent (%d bytes)"s.len()));
});

device.on("status", function(position) {
    DoorStatus = position;
    server.log("after refreshing status is " + DoorStatus); //n "last change
was at" + time /n "when the door " + lastChange);

});

device.on("timeOfchange", function(timer) {
    TimerCount = timer;
})

device.on("lastChange", function(lastChange) {
    lastChanged = lastChange;
})
```

Electric Imp Device Code

```
hardware.configure(SPI_189);
hardware.spi189.configure(CLOCK_IDLE_LOW , 117);
hardware.pin2.configure(DIGITAL_OUT);
const CHUNK_SIZE = 8192;
local timeCount = 0;
local status1 = 0;
local status2 = 0;
local statgius = 0;
local loop = 0;
local a = "FF";
local runDoorStatus = 1; //variable to run door status: 1 = run, 0 = pause
local hold = 1;
local d1 = date();
local d2 = date();
local running = 1;
local datestring = "undetermined";
local lastChange = 2;


agent.on("doorstat",function(hold){
    sendandset();
    a = hardware.spi189.writeread(format("%c",0xD5));
    imp.sleep(0.1);
    status1 = hardware.spi189.readblob(1);
    //server.log("status is " + status1);
    agent.send("status",status1[0].tointeger());
});


agent.on("spy", function(hold){
    server.log("Take a pretty picture please");
    a = hardware.spi189.writeread(format("%c",0xCA)); //CAmera

    imp.sleep(4.5);
    local L0 = hardware.spi189.readblob(1);
    local L1 = hardware.spi189.readblob(1);
    local size = L0[0].tointeger() * 256 +  L1[0].tointeger() +1;

    local num_chunks = math.ceil(size.tofloat()/CHUNK_SIZE).tointeger();
    agent.send("jpeg_start",size);

    for(local i = 0; i < num_chunks; i++) {
       local startingAddress = i*CHUNK_SIZE;
       local buf = hardware.spi189.readblob(CHUNK_SIZE);
       agent.send("jpeg_chunk", [startingAddress, buf]);
    }
    agent.send("jpeg_end",1);

});

agent.on("open",function(hold){
    server.log("opening");
    a = hardware.spi189.writeread(format("%c",0xD5));
    imp.sleep(0.1);
    status1 = hardware.spi189.readblob(1);
    if (status1[0].tointeger()==0) {
         local open = hardware.spi189.writeread(format("%c",0x0D));  // open
doo
         local check = hardware.spi189.readblob(1); //check for command
         server.log(check[0].tointeger());
    }
     else {
         server.log("door open, cannot open");
    }
    imp.sleep(10);
    timesincechange();
    lastChange = 0;
    a= hardware.spi189.writeread(format("%c",0xD5)); //D5 for Door 5tatus
        imp.sleep(0.1);
    status1 = hardware.spi189.readblob(1);
    agent.send("status", status1[0].tointeger());
});

agent.on("close", function(hold){
    server.log("closing");
    a = hardware.spi189.writeread(format("%c",0xD5));
    imp.sleep(0.1);
    status1 = hardware.spi189.readblob(1);
    if (status1[0].tointeger()==1) {
        local close = hardware.spi189.writeread(format("%c",0xCD));
        local check = hardware.spi189.readblob(1); //check for command
        server.log(check[0].tointeger());
    }
    else {
        server.log("door closed, cannot close");
    }
    imp.sleep(10);
    timesincechange();
    lastChange = 0;

    a= hardware.spi189.writeread(format("%c",0xD5)); //D5 for Door 5tatus
        imp.sleep(0.1);
    status1 = hardware.spi189.readblob(1);
    agent.send("status", status1[0].tointeger());

});


function timesincechange(){
    d1 = date();
    datestring = format("%04d%02d%02d-%02d:%02d:%02d", d1.year, d1.month+1,
d1.day, d1.hour-4, d1.min, d1.sec);
    server.log(datestring);
}

function sendandset(){
    agent.send("timeOfchange", datestring);
    agent.send("lastChange", lastChange);
}
```

```
a = hardware.spi189.writeread(format("%c",0xD5));
imp.sleep(0.1);
status1 = hardware.spi189.readblob(1);
agent.send("status",status1[0].tointeger());
```

9.5 Appendix of Android Application Code Files

activity_main.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/
android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:paddingBottom="@dimen/activity_vertical_margin"
android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
tools:context=".MainActivity" >
</RelativeLayout>
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.example.osesame_tab"
android:versionCode="1"
android:versionName="1.0" >
<uses-sdk
android:minSdkVersion="14" android:targetSdkVersion="18" />
<!-- Gives permission to access the internet -->
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"
/>
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE"
/>
<uses-permission
android:name="android.permission.CHANGE_NETWORK_STATE" />
<application
android:allowBackup="true" android:icon="@drawable/ic_launcher"
android:label="@string/app_name" android:theme="@style/AppTheme"
android:debuggable="true" > <!-- Should take out
debuggable for final product! -->
<activity android:name="com.example.osesame_tab.MainActivity"
android:label="@string/app_name" >
<intent-filter>
<action android:name="android.intent.action.MAIN" />
<category android:name="android.intent.category.LAUNCHER" />
</intent-filter> </activity>
</application>
</manifest>
```

MainActivity.java

```
package com.example.osesame_tab; import java.io.BufferedInputStream;
public class MainActivity extends Activity implements OnClickListener
{
```

```
private static final String IMP_URL1 = "https://
agent.electricimp.com/0hHlDZd2PRqR?door=1";
private static final String IMP_URL2 = "https://
agent.electricimp.com/0hHlDZd2PRqR?door=0 ";
private static final String IMP_URL3 = "https://
agent.electricimp.com/0hHlDZd2PRqR?pic=1";
private static final String IMP_URL4 = "https://
agent.electricimp.com/0hHlDZd2PRqR?status=1";
private static final String
private static final String
private static final String
private TextView textView;
private TextView textView2;
private BroadcastReceiver receiver;
DEBUG_TAG = "HttpExample"; homeWifi = "University Edge WiFi";
homeWifi2 = "ND-secure";
@Override
protected void onCreate(Bundle savedInstanceState) { //TODO Auto-
generate method stub super.onCreate(savedInstanceState);
setContentView(R.layout.tabs);
TabHost th = (TabHost)findViewById(R.id.tabhost); th.setup();
Button bOpen = (Button)findViewById(R.id.open_button); Button bClose =
(Button)findViewById(R.id.close_button); Button bCapture =
(Button)findViewById(R.id.capture_button); Button bStatus =
(Button)findViewById(R.id.status_button); Switch sDetection =
(Switch)findViewById(R.id.switch1); textView =
(TextView)findViewById(R.id.status);
textView2 = (TextView)findViewById(R.id.status1);
bOpen.setOnClickListener(this); bClose.setOnClickListener(this);
bCapture.setOnClickListener(this); bStatus.setOnClickListener(this);
/**Set up each tab using TabSpec */
TabSpec specs = th.newTabSpec("tag1"); specs.setContent(R.id.tab1);
specs.setIndicator("Status"); th.addTab(specs);
// Controls Tab
specs = th.newTabSpec("tag2"); specs.setContent(R.id.tab2);
specs.setIndicator("Controls"); th.addTab(specs);
// Camera Tab
specs = th.newTabSpec("tag3"); specs.setContent(R.id.tab3);
specs.setIndicator("Camera"); th.addTab(specs);
//create BroadcastReceiver
receiver = new BroadcastReceiver(){
@Override
public void onReceive(Context context, Intent intent) { boolean
connected = false;
final String action = intent.getAction();
if
(WifiManager.SUPPLICANT_STATE_CHANGED_ACTION.equals(action)){
SupplicantState state =
intent.getParcelableExtra(WifiManager.EXTRA_NEW_STATE);
if (SupplicantState.isValidState(state) && state
== SupplicantState.COMPLETED){
connected = checkConnectedToDesiredWifi();
}
if (connected){
new DownloadWebpageTask().execute(IMP_URL1);
Toast.makeText(context, "Welcome Home...Opening Door.",
Toast.LENGTH_LONG).show();
}
else{
new DownloadWebpageTask().execute(IMP_URL2); Toast.makeText(context,
"Leaving
Home...Closing Door.", Toast.LENGTH_LONG).show(); }
} }
//Detect if on the right wifi connection
private boolean checkConnectedToDesiredWifi(){ boolean connected =
false;
WifiManager wifiManager =
(WifiManager)getSystemService(WIFI_SERVICE);
WifiInfo wifi = wifiManager.getConnectionInfo();
if(wifi != null){ //get wif name
homeWifi2.equals(ssid); }
};
String ssid = wifi.getSSID(); connected = homeWifi.equals(ssid) ||
return connected; }
sDetection.setOnCheckedChangeListener(new
CompoundButton.OnCheckedChangeListener() {
@Override
public void onCheckedChanged(CompoundButton buttonView, boolean
isChecked) {
if(isChecked){
IntentFilter filter = new IntentFilter();
filter.addAction(WifiManager.SUPPLICANT_STATE_CHANGED_ACTION);
//register BroadcastReceiver
registerReceiver(receiver, filter); }
else{ unregisterReceiver(receiver);
} }
}); }
/**
* Create onClick method to handle the button clicks */
@Override
public void onClick(View arg0) {
// Switch between which button was pressed switch(arg0.getId()){
case R.id.open_button:
// Gets the URL from the UI's text field.
ConnectivityManager connMgr = (ConnectivityManager)
getSystemService(Context.CONNECTIVITY_SERVICE);
NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();
if (networkInfo != null && networkInfo.isConnected())
{
/*I changed the parameter to IMP_URL instead of stringURL from the
text field*/
```

```
available."); available.");
new DownloadWebpageTask().execute(IMP_URL1); } else{
textView.setText("No network connectivity
textView2.setText("No network connectivity
} break;
case R.id.close_button:
// Gets the URL from the UI's text field.
ConnectivityManager connMgr1 = (ConnectivityManager)
getSystemService(Context.CONNECTIVITY_SERVICE);
NetworkInfo networkInfo1 = connMgr1.getActiveNetworkInfo();
if (networkInfo1 != null && networkInfo1.isConnected()) {
/*I changed the parameter to IMP_URL instead of stringURL from the
text field*/
available."); available.");
new DownloadWebpageTask().execute(IMP_URL2); } else{
textView.setText("no network connectivity
textView2.setText("No network connectivity
} break;
case R.id.capture_button:
// Gets the URL from the UI's text field.
ConnectivityManager connMgr2 = (ConnectivityManager)
getSystemService(Context.CONNECTIVITY_SERVICE);
NetworkInfo networkInfo2 = connMgr2.getActiveNetworkInfo();
if (networkInfo2 != null && networkInfo2.isConnected()) {
/*I changed the parameter to IMP_URL instead of stringURL from the
text field*/
new DownloadWebpageTask().execute(IMP_URL3);
new
DownloadImageTask((ImageView)findViewById(R.id.imageView1))
.execute("http://10.36.251.234/pictures/
testpic1.jpg");
} else{
textView.setText("no network connectivity
textView2.setText("No network connectivity
} break;
case R.id.status_button:
// Gets the URL from the UI's text field.
ConnectivityManager connMgr3 = (ConnectivityManager)
getSystemService(Context.CONNECTIVITY_SERVICE);
NetworkInfo networkInfo3 = connMgr3.getActiveNetworkInfo();
if (networkInfo3 != null && networkInfo3.isConnected()) {
/*I changed the parameter to IMP_URL instead of
stringURL from the text field*/
available."); available.");
new DownloadWebpageTask().execute(IMP_URL4); } else{
textView.setText("no network connectivity
textView2.setText("No network connectivity
} break;
} }
//Create AsyncTask to get image from URL
```

```
private class DownloadImageTask extends AsyncTask<String, Void,
Bitmap>{
ImageView bmImage;
public DownloadImageTask(ImageView bmImage){ this.bmImage = bmImage;
}
protected Bitmap doInBackground(String... urls) { String urldisplay =
urls[0];
Bitmap mIcon11 = null;
try {
InputStream in = new java.net.URL(urldisplay).openStream();
mIcon11 = BitmapFactory.decodeStream(in); } catch (Exception e) {
Log.e("Error", e.getMessage());
e.printStackTrace(); }
return mIcon11; }
protected void onPostExecute(Bitmap result){
bmImage.setImageBitmap(result);
} }
// Uses AsyncTask to create a task away from the main UI thread. This
task takes a
// URL string and uses it to create an HttpUrlConnection. Once the
connection
// has been established, the AsyncTask downloads the contents of the
webpage as
// an InputStream. Finally, the InputStream is converted into a
string, which is
// displayed in the UI by the AsyncTask's onPostExecute method.
private class DownloadWebpageTask extends AsyncTask<String, Void,
String> {
@Override
protected String doInBackground(String... urls) {
url.
// params comes from the execute() call: params[0] is the
try {
return downloadUrl(urls[0]);
} catch (IOException e) {
return "Unable to retrieve web page. URL may be
invalid.";
}
}
// onPostExecute displays the results of the AsyncTask.
@Override
protected void onPostExecute(String result) {
textView.setText(result); textView2.setText(result);
} }
Given a URL, establishes an HttpUrlConnection and retrieves the web
page content as a InputStream, which it returns as a string.
//
//
//
private String downloadUrl(String myurl) throws IOException {
```

```
InputStream is = null;
// Only display the first 500 characters of the retrieved // web page
content.
int len = 500;
try {
URL url = new URL(myurl);
HttpURLConnection conn = (HttpURLConnection)
url.openConnection();
conn.setReadTimeout(10000 /* milliseconds */);
conn.setConnectTimeout(15000 /* milliseconds */);
conn.setRequestMethod("GET");
is
conn.setDoInput(true);
// Starts the query
conn.connect();
int response = conn.getResponseCode(); Log.d(DEBUG_TAG, "The response
is: " + response); is = conn.getInputStream();
// Convert the InputStream into a string
String contentAsString = readIt(is, len); return contentAsString;
// Makes sure that the InputStream is closed after the app
// finished using it.
} finally {
if (is != null) {
is.close();
}
} }
// Reads an InputStream and converts it to a String.
public String readIt(InputStream stream, int len) throws
IOException,
}
UnsupportedEncodingException {
Reader reader = null;
reader = new InputStreamReader(stream, "UTF-8"); char[] buffer = new
char[len]; reader.read(buffer);
return new String(buffer);
}

tabs.xml <?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/
android"
android:layout_width="match_parent"
android:layout_height="match_parent" android:orientation="vertical" >
<TabHost
android:id="@+id/tabhost" android:layout_width="match_parent"
android:layout_height="match_parent" >
<LinearLayout android:layout_width="match_parent"
android:layout_height="match_parent" android:orientation="vertical" >
<TabWidget
android:id="@android:id/tabs" android:layout_width="match_parent"
android:layout_height="wrap_content" >
</TabWidget>
```

```
<FrameLayout android:id="@android:id/tabcontent"
android:layout_width="match_parent"
android:layout_height="match_parent">
<LinearLayout
android:id="@+id/tab1" android:orientation="vertical"
android:layout_width="match_parent"
android:layout_height="match_parent">
<RelativeLayout android:layout_width="match_parent"
android:layout_height="match_parent">
<TextView android:id="@+id/status1"
android:layout_width="match_parent"
android:layout_height="wrap_content" android:gravity="center"
android:textSize="20sp" android:maxHeight="50dp"
android:paddingTop="20dp" />
<Button
android:id="@+id/status_button" android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_below="@id/status1" android:text="STATUS"
android:textSize="40sp" />
<TextView
android:id="@+id/detection" android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_centerHorizontal="true"
android:layout_centerVertical="true" android:text="Wi-Fi Detection
Mode" android:textSize="25sp" />
<Switch
android:id="@+id/switch1" android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_below="@+id/detection"
android:layout_centerHorizontal="true" android:layout_marginTop="26dp"
android:textOff="Off" android:textOn="On" />
</RelativeLayout> </LinearLayout>
<LinearLayout android:id="@+id/tab2" android:orientation="vertical"
android:layout_width="match_parent"
android:layout_height="match_parent">
<TextView
android:id="@+id/status" android:layout_width="wrap_content"
android:layout_height="wrap_content" android:gravity="center"
android:textSize="20sp" android:layout_weight="1"/>
<Button
android:id="@+id/open_button" android:layout_width="match_parent"
android:layout_height="wrap_content" android:layout_weight="4"
android:text="OPEN" android:textSize="50sp" />
<Button
android:id="@+id/close_button" android:layout_width="match_parent"
android:layout_height="wrap_content" android:layout_weight="4"
android:text="CLOSE" android:textSize="50sp" />
</LinearLayout>
<LinearLayout
android:id="@+id/tab3" android:orientation="vertical"
```

```
android:layout_width="match_parent"
android:layout_height="match_parent">
<ImageView
android:id="@+id/imageView1" android:layout_width="match_parent"
android:layout_height="wrap_content" android:layout_weight="7"/>
<Button
android:id="@+id/capture_button" android:layout_width="match_parent"
android:layout_height="wrap_content" android:textSize="30sp"
android:layout_weight="1" android:text="CAPTURE" />
</LinearLayout> </FrameLayout>
</LinearLayout> </TabHost>
</LinearLayout>
```

9.6 Appendix of Server Code Files

upload.php
```
<?php
$s = file_get_contents("php://input");
$fp = fopen("/var/www/pictures/testpic1.jpg","w");
fwrite($fp, $s);
//file_put_contents("/var/www/pictures/testpic1.jpg", $s);
fclose($fp);
?>
```

testupload.php
```
<html>
 <head>
  <title>PHP Test</title>
 </head>
 <body>
 <? echo '<p>Hello World</p>';
$content = file_get_contents("http://evbdn.eventbrite.com/s3-
s3/eventlogos/1832816/google.png");
$fp = fopen("/var/www/pictures/googlelogo.jpg","w");
fwrite($fp, $content);
fclose($fp);
//header('Content-Type: image/jpeg');
//file_put_contents($patrick.jpg,$content);
//echo '<img src="patrick.jpg">';
?>
</body>
</html>
```

check_php.php
```
<html>
<title>Check PHP</title>
<body>
<?php
print "<h1>I'm Alive!</h1>";
phpinfo( );
exit();
?>
<h1>Oops, PHP is not enabled on this server.</h1><br /><br />
</body>
</html>
```

patrickpic.php
```
<html>
   <head>
```

```
    <meta name="viewport" content="width=device-width, minimum-scale=0.1">
    <title>Patrick Says Open Sesame (350×263)</title>
    <style type="text/css"></style>
  </head>
  <body style="margin: 0px;">
    <img style="-webkit-user-select: none"
src="http://medias.gifboom.com/medias/t_a4ea053fc45d4361bcf891599e3491ab.jpg">
  </body>
</html>
```

stringimage.php

```
<?
//$url = 'http://10.36.251.234/pictures/patrickpic.php';
//$img = '/var/www/pictures/patrick.jpg';
//file_put_contents($img, file_get_contents($url));
copy("http://10.36.251.234/pictures/patrickpic.php", "/var/www/pictures/patrick.jpg");
?>
```

9.7 Appendix of Website Code Files

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!-- Open Sesame Documents page -->
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Open Sesame Documents</title>
<meta name="keywords" content="" />
<meta name="description" content="" />
<link
href="http://fonts.googleapis.com/css?family=Source+Sans+Pro:200,300,400,600,
700,900" rel="stylesheet" />
<link href="stylesheet.css" rel="stylesheet" type="text/css" media="all" />
<link href="fonts.css" rel="stylesheet" type="text/css" media="all" />
</head>

<body>
<div id="header-wrapper">
        <div id="header" class="container">
                <div id="logo">
                        <h1><a href="#">Demo</a></h1>
                </div>
                <div id="menu">
                        <ul>
                                <li><a href="index.html" accesskey="1"
title="">Home</a></li>
                                <li><a href="Documents.html" accesskey="2"
title="">Documents</a></li>
                                <li><a href="AboutUs.html" accesskey="3"
title="">About Us</a></li>
                                <li class="current_page_item"><a href="#"
accesskey="4" title="">Demo</a></li>
                        </ul>
                </div>
        </div>
</div>

<div id="page-wrapper">
        <div id="page" class="container">
                <a id ="switch" href="#" class="button" onclick
="ControlDoor()" style ="text-decoration: none">Close Door</a>
                <a id = "doorstatus" href="#" class="button" onclick =
"CheckStatus()" style = "text-decoration: none">Check Door Status</a>
                <a id = "picture" href="#" class="button" onclick =
"TakePicture()" style = "text-decoration: none">Take Picture</a>
        </div>
</div>

<div id="copyright" class="container">
        <p>Copyright (c) Open Sesame 2013-2014</p>
</div>
<script src="MyScript.js"></script>
</body>
</html>
```