Web based Remote Monitor and Control System for Smart Home

Xiaolong Li[*1], Tim Morgan[2], William Clyburn[3]

Department of Electronics and Computer Engineering Technology, Indiana State University, Terre Haute Indiana

[*1]Xiaolong.li@indstate.edu

Abstract

Smart home is a house that uses information technology to control the electronic appliances, monitor the home environment and communicate with outer world. A sample house electric appliance monitor and control system that is one brand of the Smart home is addressed in this paper. This system integrates the AC power socket, low-power microcontroller and wireless communication into a wireless power socket that can be switched ON/OFF remotely through Internet. The system consists of three modules, that is, the Web Server Module, the Control Device Module, and the End Device Module which together provide an indoor wireless, and an outdoor remote control and monitor of home electric appliances. This paper presents the hardware and software implementation. The test results of the system have shown that it can be easily used for the smart home applications.

Keywords

Smart Home, Remote Control, Home Automation, Microcontroller

Introduction

Smart home is a house that uses information technology to connect all type of devices to communicate each other through Internet. With the ever expanding range of consumer devices gaining Internet capability, smart home will become a common luxury that nearly everyone will soon enjoy. This includes being able to know what is currently happening at the home as well as being able to control what is happening at the home from over the Internet.

The smart home involves the control and automation of lighting, heating, ventilation, air conditioning (HVAC), security and home appliances such as refrigerators/freezers, washer/dryer, microwave, etc [1]. In [2], the authors presented a way of monitoring and controlling energy consumption of electronics and devices using smart sockets and smartphone. The smart power socket can measure power consumption of electrical devices, and transmit the collected data to a host server. The authors in [3] provided a similar idea as [2] to monitor, control and manage electric loads at home. In this paper, we also develop our own wireless power socket that integrates the AC power socket, low-power microcontroller and wireless communication. The wireless power socket can be switched ON/OFF remotely through Internet. The system consists of three modules, that is, the Web Server Module, the Control Device Module, and the End Device Module which together provide an indoor wireless, and an outdoor remote control and monitor of home electric appliances. The user/home owner connects to a web site hosted on a web server located inside of the home. This web site, called the Control Dashboard, is the interface to control the devices in their home. On the webpage it lists whether the power socket is currently energized or inactive. The ability to turn this power socket ON or OFF is also available from this Dashboard interface. A central unit, called the Control Device, acts as the central hub of the wireless network as well as maintain the communication between the End Devices and the web server. Compared to the power socket presented in [2] and [3], our power socket can be controlled from Internet through any computer or smartphone. Instead of using the power line communication as in [4], we are using 434MHz transmitter and receiver units. A very simple wireless protocol was created to allow communication between devices. With very minimal modification to the code, additional units of varying types can be implemented in this system. Although only power sockets were created, this project is designed and engineered with the intention of only being the backbone to a larger network of possible devices with the ability to

2324-7037/17/02 001-12
© 2017 DEStech Publications, Inc.
doi: 10.12783/ijcsa.2017.0602.01

add features with relative ease.

This paper has been organized as following: in Section 2, the detailed system design will be presented. The implementation is described in Section 3. At the end of this paper, the conclusions will be provided.

System Design

System Overview

The overall system will allow a user to monitor and control an electric appliance through the wireless power socket from a remote location via an Internet connection. The system can be divided into three logical modules as shown in Figure 1. These modules include the Web Server module, the Control Device module and the End Device module. Because the system can be divided into these distinct functional modules, each module can be designed and tested independently from the others.

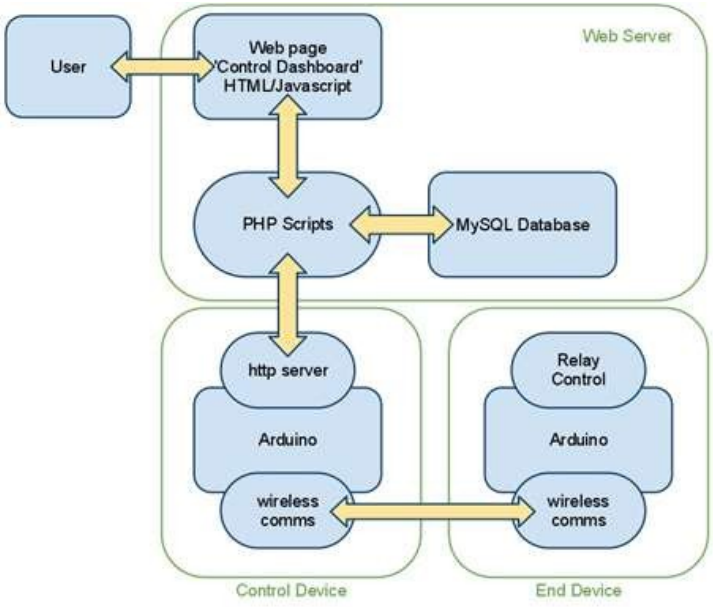

FIGURE 1. FUNCTIONAL BLOCK DIAGRAM

Web Server

The web server is physically located on an Asus Eee PC netbook computer. The Asus netbook computer has a small profile and can be conveniently placed anywhere in the home due to built-in Wi-Fi capability. This Asus Eee PC netbook computer, model number 900HA, is equipped with an Intel 1.6GHz processor, 1GB of RAM and 160GB of hard drive space. With an expected light work load, the Eee PC has more resources than what is required, but due to its size and mobility it was deemed viable for this project.

A LAMP architecture is used to implement the web server structure. It is built upon a Linux, Apache, MySQL and PHP framework. Linux is the underlying operating system that everything on the computer runs on. Apache is the actual web server software used. MySQL is the type of database that is utilized in this project while PHP is a scripting language that allows code to be run on the backend of the web server. This type of setup is very common and its ease of configuration was an obvious choice for this project.

1) *Control Dashboard*

The web server contains several aspects of the system. The most apparent to the user is the Control Dashboard. This is the web interface that the user interacts with. The Control Dashboard has three separate pages: the login page, the main page and the settings page. The login screen shown in the Figure 2 is the initial screen that the user sees when connecting to the system. This screen uses a very simplistic layout with only a username field, a password field and a submit button.

FIGURE 2. CONTROL DASHBOARD – LOGIN SCREEN

It should also be noted that the Control Dashboard is accessed through an SSL connection. This provides a certain level of security to prevent log in information from being viewed over an internet connection. In the pictures provided, it shows the IP address of the Dashboard as a local address. The Dashboard can be setup to be accessible from the Internet by simply opening port 443 on the home router.

The page that is utilized the most is the main page of the Control Dashboard (Figure 3). It displays a module for each end device connected to the system. Figure 3 shows an example with two modules. One of these two modules has the user defined name of the device in the top corner.

This name can be changed by clicking on the edit button next to the name and typing in a new name (Figure 4). The type of device is listed under the name. In our project, only a power socket is being implemented, but additional types of devices are possible with very little change to the system.

FIGURE 3. CONTROL DASHBOARD –MAIN SCREEN

FIGURE 4. EDITING THE DEVICE NAME FROM THE MODULE

The final two pieces of information listed within a module is the current state of the device, in this case either ON or OFF, and buttons to turn this device either ON or OFF. Out of the two buttons, only one will be active at a time. At the very top right of the main screen are two options for Settings and Logout. By clicking on Settings, a user will be taken to the settings page (Figure 5). The User Settings page allows certain user attributes to be changed. This page lists the username, first name, last name and email address of the user. The username cannot be modified because it is used the unique identifier in the database for each user. Most of these fields are not currently used, but future implementations could use these settings for additional features. An example of a feature that could be added would be an alert that is sent to a user's email address if an event is triggered or if an event lasts for a certain amount of time. This page also allows the user to change the password that is used to access the Control Dashboard. The password requires a minimum of eight characters and must be typed twice to avoid misspellings. If either of these conditions are not met, an error appears describing the cause of the error and the Save button becomes disabled. The passwords are stored in the database using an SHA-1 has function for an extra layer of security.

FIGURE 5. CONTROL DASHBOARD – USER SETTINGS

2) *PHP Scripts*

The Control Dashboard web page calls various PHP scripts to do things such as querying the database, updating the database, sending commands to the Control Device as well as calling PHP scripts to update the information being displayed on the Dashboard itself.

The scripts are separated into individual files based on function. For example, to open a connection and then to close a connection to the database, the files called opendb.php and closedb.php would be called. Certain files, such as get_modules.php and update_dev_state.php, communicate with the database by either reading the database or updating fields within the database. A full listing of the PHP files is shown in Table 1.

TABLE 1. LIST OF PHP FILES

change_user_settings.php	edit_device_name.php	update_dev_state.php
checkDB.php	get_info.php	update_module.php
closedb.php	get_modules.php	update_state.php
dbconfig.php	login.php	updated.php
dbconfig_user_info.php	logout.php	verify_user.php
display_module.php	opendb.php	verify_user_root.php

3) *MySQL Database*

The database used is MySQL. This is the standard database in the LAMP architecture. Like the rest of the software included, MySQL is free software and is easy to setup, configure and to use.

The database in this system is called home_automation and includes three tables: device_info, device_properties and user_info. The user_info table (Figure 6) stores information that is editable from the User Settings page of the Control Dashboard. Included in this table is the username, password, first name, last name and email address. As shown in the Figure 6, the password is encrypted into a 160-bit (40 hexadecimal) field.

```
mysql> select * from user_info;
+---------+------------+------------------------------------------+---------+----------+--------------------------+
| user_id | user_name  | password                                 | f_name  | l_name   | email_address            |
+---------+------------+------------------------------------------+---------+----------+--------------------------+
|       1 | admin2011  | 54e52c22c76df2d2132a11b991e783e4b1a2be7d | Admin   | User     | NULL                     |
|       2 | timmy      | b0dc3c34eaff1fd9419dce1ae71101b4bb3f668f | Tim     | Morgan   | timmy.w.morgan@gmail.com |
|       3 | jason      | 16bfc71f9e5ef30075a96f2853906736e051fbb3 | Jason   | King     | NULL                     |
|       4 | aeldridge1 | 7f3311167e863f108e8ec37a09bc0f1d28535b95 | Andrick | Eldridge | aeldridge1@indstate.edu  |
|       5 | aeldridge  | c657116de99973d2314c90fa37c8d80da1a0987f | Andrick | Eldridge | aeldridge1@indstate.edu  |
+---------+------------+------------------------------------------+---------+----------+--------------------------+
5 rows in set (0.00 sec)
```

FIGURE 6. USER INFORMATION TABLE

The other two tables, device_info and device_properties, store information about the End Devices. The device_info table stores information that primarily deals with what information is displayed in the modules on the Control Dashboard. As shown in Figure 7, the fields in this table are the device id, device name and the device type. The last two fields, last active and is active, are not yet implemented. These two fields are intended

to display modules for devices that are currently connected to the system and not to display modules for devices that are inactive. The device_name field is editable by the user from the Dashboard.

```
mysql> select * from device_info;
+-----------+-------------------+-------------+-------------+-----------+
| device_id | device_name       | device_type | last_active | is_active |
+-----------+-------------------+-------------+-------------+-----------+
| A1        | Living Room Lamp  | Power Strip | 21:40:20    |         1 |
| A2        | Office Lamp       | Power Strip | 22:07:05    |         1 |
+-----------+-------------------+-------------+-------------+-----------+
2 rows in set (0.00 sec)
```

Figure 7. Device Information Table

The last table, device_properties, holds the status of the end device. This table has fields for device id, device property id, description, state and update state. An example of this table is shown in Figure 8. When a user clicks a button from the Control Dashboard to modify the state of a device, the update_state field changes. Following the change, another PHP script is ran and checks for discrepancies between the state field and the update_state field. If the two fields are different, the PHP script sends a command to the Control Device to update the End Device. After the End Device has changed its status, it reports back and updates the state field.

```
mysql> select * from device_properties;
+-----------+----------------+-------------+-------+--------------+
| device_id | device_prop_id | description | state | update_state |
+-----------+----------------+-------------+-------+--------------+
| A1        | 65             | NULL        |     1 |            1 |
| A2        | 65             | NULL        |     0 |            0 |
+-----------+----------------+-------------+-------+--------------+
2 rows in set (0.00 sec)
```

FIGURE 8. DEVICE PROPERTIES TABLE

Control Device

The Control Device acts as a connection point within the system. The block diagram in Figure 1 shows the Control Device connecting the web server to the End Device(s). All of the End Devices must be able to communicate with the Control Device to send message to and from the Dashboard.

The processing power behind the Control Device is an ATmega328 microcontroller. The microcontroller can be used from and is programmed using an Arduino board and software, which is why 'Arduino' is listed on the functional block diagram. The ATmega328 was chosen for the sole purpose of the Arduino software/hardware functionality. By using the Arduino, designing, prototyping and programming are significantly simplified compared to other products.

The Arduino, pictured in Figure 9, allows prototyping and programming directly from a single board. It can be powered from a USB cable, an AC-to-DC power adapter or by batteries. It has 14 digital I/O pins, comes equipped with 32KB of flash memory and operates from a 16MHz clock speed.

Another advantage of using the Arduino platform is the abundance of resources. There are several devices called shields that are compatible with Arduino boards. A shield plugs directly on top of the board and normally only requires including additional libraries in the code. This project took advantage of the Ethernet Shield. By installing

FIGURE 9. ARDUINO BOARD (LEFT) AND AN ETHERNET SHIELD TACKED ON TOP OF AN ARDUINO BOARD (RIGHT)

the shield on top of the Arduino board (Figure 9) and including the Ethernet library in the code, Ethernet capability is nearly as simple as plug-and- play. This Ethernet capability appears in the block diagram (Figure 1) as the http server. The language used to program the Arduino is closely related to C++. This was another reason the Arduino was chosen over other microcontroller units.

Another important aspect of the Control Device is to communicate to the appropriate End Device. This is done wirelessly with RF Link 434MHz transmitters and receivers. Another option that was considered was to use XBee transceivers that use the ZigBee wireless standard. This was decided against due to cost considerations. An XBee transceiver can cost between $20 and $25 while a pair of the RF Link transmitter/receiver costs less than $10. While the XBee modules come with a built in wireless protocol, using the RF Link devices require additional programming to create a simple protocol. It was deemed that writing this code initially will overshadow the high price of the other option. The transmitter and receivers are simple communication devices that transmit and receive whatever appears on their data pins. These ASK modules operates on voltages between 2 and 12 volts. This system uses 5 volts to power the devices and is capable of 2400 or 4800 bps transfer rates. The transmitter (Figure 10) is a four pin device while the receiver is quite a bit longer and has eight pins.

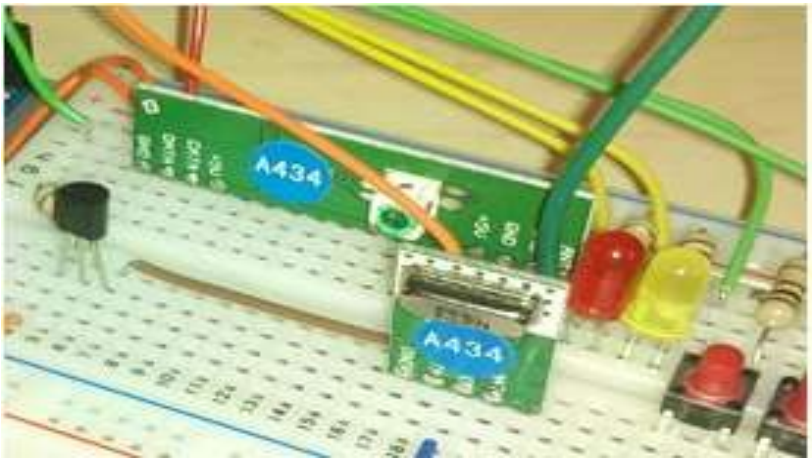

FIGURE 10. WIRELESS RECEIVER (BACK) AND WIRELESS TRANSMITTER

End Device

The actual device that is being controlled remotely is called the End Device. For this project, the end device is the electric light. The end device consists of the wireless communications portion, the microcontroller and the relay portion that controls the power provided to the electric light.The End Device must monitor the state of the device (ON or OFF) and report that state back to the Control Device. This End Device must also listen for commands and act accordingly. It is not only necessary for this device to be controlled remote, but also needs a physical means to toggle the unit. This will allow the user to operate the device without the need to log on with an internet connection.

Implementation

Control Dashboard

The Control Dashboard is an important aspect of the system because it is the interface between the user and the rest of the system. The Dashboard, since it is a web interface, is programmed using a combination of HTML, JavaScript, CSS and PHP. The accumulation of these languages provides the user a dynamic feel to the system.

4) Login Page

The login page has a very basic layout. As seen in Figure 2, the screen only has a username and password fields with a submit button. The submit button passes the username and password to login.php. The PHP script protects against MySQL injection by stripping away any slashes and escape characters from the input before going to the database (Figure 11, lines 12-15). This method is used anytime input is sent to the database.

```
7  // username and password sent from form
8  $user_name=$_POST['user_name'];
9  $password=$_POST['password'];
10
11 // To protect MySQL injection
12 $user_name = stripslashes($user_name);
13 $password = stripslashes($password);
14 $user_name = mysql_real_escape_string($user_name);
15 $password = mysql_real_escape_string($password);
16
17 $sql="SELECT * FROM $tbl_name WHERE user_name='$user_name' and
18      password=SHA1('$password')";
19 $result=mysql_query($sql);
```

FIGURE 11. PARTIAL CODE FOR LOGIN.PHP

The user's input is then verified against the database. If found to be correct, a PHP session is created (Figure 12, lines 30-33). A session allows information to be passed between different pages of the website. The username is saved into the session and as long as the session is valid, the user is logged in. Once the session is created, the user is directed to main.php (line 35), which is the main screen for the Dashboard.

```
30 // Register $username, $password and redirect to file "main.php"
31 session_register("user_name");
32 session_register("password");
33 $_SESSION['user_name'] = $user_name; // stores user in the session variable.
34
35 header("location: /main.php");
```

FIGURE 12. PARTIAL CODE FOR LOGIN.PHP

5) Main Screen

This screen is the most widely used page of the Dashboard interface. It is required to not only display the modules for the End Devices, but it also shows the state of the device and the buttons to control that device. The main.php file is rather small, but calls several PHP and JavaScript files. Displaying a module on the main page happens in two steps. First, the get_modules.php script is called to query the database and reads information about the devices connected to the system. Secondly, display_module.php is called for each device and creates a module on the screen. The display_module.php script, as shown in Figure 13, is called once for every module.

```
20 echo "  <div class='modName' id='$device_id'>\n";  // Customizable name of the
21 echo "          $device_name <a class='modEdit' id='$device_id'
22                  onClick=\"editDeviceName('$device_id')\">edit</a>\n";
23 echo "  </div>\n";
24 echo "  <div class='modType'>\n"; // Hard coded type of device
25 echo "          $device_type\n";
26 echo "  </div>\n";
27 echo "              <div class='modControl'>\n";
28 echo "                  <button class='offBtn' type='button'
29                          onClick=\"turnOff('$device_id')\" $off_btn_disable>
30                          Off</button>\n";
31 echo "                  <button class='onBtn' type='button'
32                          onClick=\"turnOn('$device_id')\" $on_btn_disable>
33                          On</button>\n";
34 echo "              </div>\n";
35 echo "  <div class='modInfo'>\n"; // Can contain current state of device
36 echo "          The current state of this device is ";
37 echo " <span class='modState'>$device_state</span>\n";
38 echo "  </div>\n";
```

FIGURE 13 DISPLAY_MODULE.PHP

This PHP script, lines 20-38, generates the HTML code that will display the module on the Control Dashboard. Line 20 creates each module with a unique div id value. The editable device name, which is shown in Figure 4, is listed in the HTML code in lines 21-22. When the edit button is clicked, it calls the editDeviceName() JavaScript function. Once the user has updated the name, updateDeviceName() is called to store the new name into the database.

User Settings Screen: The User Settings Screen allows the user to view and update settings specific to them. A picture of the User Settings can be seen on Figure 6. The things that can be viewed on this page are the username, password, first name, last name and email address. All of these options can be changed except for the username. There are two requirements for the password fields. First, the password has to be at least eight characters long. The code for validateLength() can be seen in Figure 14. If the password is less than eight characters, an error message is displayed and the submit button is disabled (lines 44-45). The rest of the code shown in Figure 14 makes the error makes disappear when appropriate.

The second requirement for the password is that it must be typed in twice and both times must match. This is checked in the function called validatePass(). This function checks both fields and if they do not match, an error message is displayed and the submit button is disabled.

Besides checking the password fields before the database is updated, every field is protected against MySQL injection. This code is found in the change_user_setings.php file which is similar to the code in Figure 11. By protecting against MySQL injection, this prevents a user from executing code from within the database and keeps the system secure.

```
33  /* This checks that the password is a certain length. It will display
34      an error message and the submit button becomes disabled         */
35  function validateLength()
36  {
37      // checks if pw block is not blank
38      if (document.user_settings.password.value != "")
39      {
40          // check is pw block is below min length
41          if (document.user_settings.password.value.length < 8)
42          {
43              // display error and disable submit
44              document.getElementById('pass_length_error').style.display="inline";
45              document.user_settings.submit.disabled=true;
46          }
47          else
48          {
49              // removes error msg and enables submit
50              document.getElementById('pass_length_error').style.display="none";
51              document.user_settings.submit.disabled=false;
52          }
53      }
54      else
55      {
56              // removes error msg and enables submit
57              document.getElementById('pass_length_error').style.display="none";
58              document.user_settings.submit.disabled=false;
59      }
60  }
```

FIGURE 14. VALIDATE_FORM.JS

Implementation – Control Device

The primary purpose of the Control Device is to facilitate the communication between the Dashboard and the End Devices. By doing this, it needs to be able to use the Ethernet protocol as well as a wireless protocol to communicate with the End Device. The Control Device's main program can be divided into three main functions.

The first thing that the Control Device does is to read from the Ethernet connection (Figure 15, line 80). The checkEthernet() function returns true if it receives information that it is able to pull variables from. The device will then pass along those variables to the appropriate End Device by sending a wireless signal (line 85). It also stores these variables in the struct called lastSent in case it needs to be sent again for any reason. The second main function that is in the main program is checkStatus(). This portion of the code checks the wireless communications for any signal sent from the End Devices and passes it along to the web server.

```
80   if (checkEthernet())
81   {
82     /* if a HTTP request was successfully read then it will
83        forward that command to the specified end device */
84     digitalWrite(LED_TXRX, HIGH);
85     wireless.send(controlSignal.toAddress, controlSignal.command, controlSignal.data1, c
86     lastSent.toAddress = controlSignal.toAddress;
87     lastSent.command = controlSignal.command;
88     lastSent.data1 = controlSignal.data1;
89     lastSent.data2 = controlSignal.data2;
90     delay(50);
91     digitalWrite(LED_TXRX, LOW);
92   }
93
94   /* checks the wireless comms to see if any end devices have sent data */
95   checkStatus();
```

FIGURE 15. PARTIAL PROGRAM CODE FOR CONTROL DEVICE

The last thing that the main function does, as shown in Figure 16, is to send a GET request to the web server. This portion of the code is ran once about every ten seconds. This runs a PHP script that checks the database for any updates. If there are updates, the server sends a request back to the Control Device which is read in through the checkEthernet() function.

```
99   if (checkDBcounter == 10000)
100  {
101    if (client.connect())
102    {
103      client.println("GET /php/checkDB.php HTTP/1.0");
104      client.println();
105      delay(500);
106      client.stop();
107    }
108    checkDBcounter = 0;
109  }
110  checkDBcounter++;
111  delay(1);
```

FIGURE 16. PARTIAL PROGRAM CODE FOR CONTROL DEVICE

Implementation – End Device

The End Device has two primary functions, to communicate with the Control Device and to control the relay that controls the power socket. The main loop in the programming for the End Device is relatively short. The code appears in Figure 17. The checkWireless() function reads the wireless signal sent from the Control Device and does any appropriate action based on it. The checkButton() function monitors the physical on/off toggle button on the device. Lines 47 through 51 sends the current state of the device to the Control Device on a regular basis, based on the sendStateDelayCounter. This counter is initially set to 15000 which causes it to send a signal every 15 seconds.

```
44   checkWireless();   // checks for wireless communications
45   checkButton();     // checks the toggle on/off button
46
47   if ((sendStateDelayCounter == maxSendDelayCounter) && (SEND_DELAY_ENABLED))
48   {
49     sendState();
50     sendStateDelayCounter = 0;
51   }
52   sendStateDelayCounter++;
53   delay(1);
```

FIGURE 17. PARTIAL PROGRAM CODE FOR END DEVICE

Implementation – Ethernet Communication

The Ethernet connectivity is implemented in the Control Device by using the Arduino Ethernet Shield. Using this

device requires the Ethernet library to be included in control_device.cpp. By using this library, programming the communications to the web server is made relatively simple. Figure 18 shows the variables needed to set the MAC, the IP address and the address of the Dashboard webserver. The MAC is found on the Ethernet Shield and the IP address is determined by the programmer.

```
12  /* MAC and IP of Device */
13  byte mac[] = { 0x90, 0xA2, 0xDA, 0x00, 0x38, 0x16 };
14  byte ip[] = { 192,168,1,45 };
15  /* address of web server */
16  byte webServer[] = { 192,168,1,40 };
```

FIGURE 18. MAC ADDRESS AND IP ADDRESS FOR WEBSERVER

Inside of the function checkEthernet(), the information is read into readstring[] one character at a time so it can be processed. When readstring[] reaches a newline character (line 189), it sends an 'OK' response back to the web server (lines 193-195) as shown in Figure 19.

```
184     readstring[n] = client.read();
185
186     // if you've gotten to the end of the line (received a newline
187     // character) and the line is blank, the http request has ended,
188     // so you can send a reply
189     if (readstring[n] == '\n')
190     {
191       readstring[n+1] = '\0';
192       // send a standard http response header
193       client.println("HTTP/1.1 200 OK");
194       client.println("Content-Type: text/html");
195       client.println();
```

FIGURE 19. READING INFORMATION FROM ETHERNET

Once all of the information is read in, the variables are extracted from it. The actual information that is received is in the format of "GET /?id=A1&cm=C1&d1=65&d2=B1" and the variables are converted to toAddress, command, data1 and data2. This happens in the portion of the code listed in Figure 20. This is the information that gets sent to the appropriate End Device.

```
197     /* extracts variables from the http GET request */
198     controlSignal.toAddress = extractVariableValue(readstring, OPT_TOADDRESS);
199     controlSignal.command = extractVariableValue(readstring, OPT_COMMAND);
203       controlSignal.data1 = extractVariableValue(readstring, OPT_DATA1);
206         controlSignal.data2 = extractVariableValue(readstring, OPT_DATA2);
```

FIGURE 20. EXTRACTING VARIABLES FROM THE ETHERNET INFORMATION

Implementation – Wireless Communications

The wireless communication between the devices is packet based. Each packet is eight bytes long. An example of the packet structure is shown in Figure 21. The first byte is the preamble byte. The purpose for this is to tell the receiver to start receiving the packet. The second byte is the SYNC byte. Each receiver waits until it reads a SYNC byte. After a SYNC byte is read, it tells the receiver that it has a legitimate packet it begins to read all of the other bytes into appropriate variables.

The toAddress and fromAddress bytes indicate what device the message is intended for and which device it is being sent from. All devices are hardcoded with a network address. The Control Device is always assigned address 0xA0 while all subsequent devices are addressed 0xA1, 0xA1 and so on.

1	2	3	4	5	6	7	8
Preamble	SYNC	To Address	From Address	Command	Data1	Data2	Checksum
0xF0	0x8F	0xA1	0xA0	0xC3	0xB5	0xB0	0xC6

FIGURE 21. WIRELESS PACKET STRUCTURE

The Command field indicates what type of information is being passed in the packet. The list of commands can be found in Figure 22. For example, if 0xC1 is in the command field, the receiving device will resend the last packet that it sent. A command of 0xC2 tells the device to perform some sort of action. A byte of 0xC4 requests that the device sends back information about a specific attribute while 0xC3 indicates that the data bytes hold information about an attribute.

```
/* Address List */
const byte CONTROL_DEVICE = 0xA0;   // address of control device is always 0xA0

/* Command List */
const byte CMD_RESEND_LAST = 0xC1; // command to retransmit last packet
const byte CMD_INSTRUCT = 0xC2; // instruction command
const byte CMD_INFO = 0xC3; // command that states this packet includes device info
const byte CMD_REQUEST_INFO = 0xC4; // request for device to send info

/* Device Info */
const byte STATE_OFF = 0xB0; // state of device is off
const byte STATE_ON = 0xB1;  // state of device is on
```

FIGURE 22. CONSTANT VARIABLES FOR PACKET STRUCTURE

Bytes 6 and 7 are data fields. These fields are determined by the command byte. A command signal of 0xC2, instruction signal, would list Data1 as the device property to be changed and Data2 as the state to change it to. Otherwise, a command signal of 0xC3 would have Data1 as the device property and Data2 as the current state of the device.

The last byte of the packet, the checksum, is XOR'ed between data1, data2 and the command byte. Data1 and data2 are XOR'ed. Then the result of that is XOR'ed with the command byte. This is then sent as the checksum in the wireless packet. On the receiving end, the exact same is done and the two are compared to verify that the packet has held integrity. The calculation of the checksum is shown in Figure 23.

```
byte checksum = data_1 ^ data_2;    // calculates checksum
checksum = checksum ^ command;      // by using XOR
```

FIGURE 23. CALCULATING THE CHECKSUM

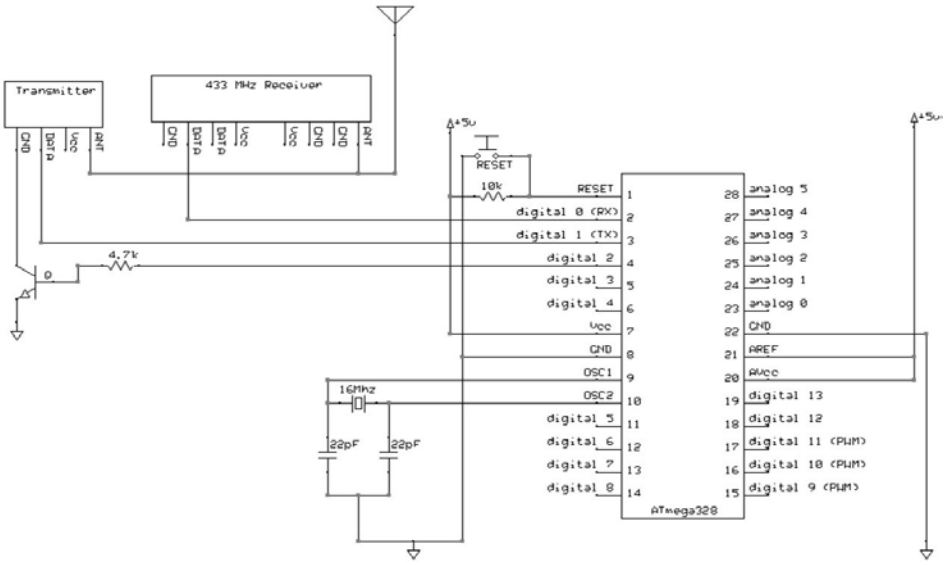

FIGURE 24. SCHEMATIC FOR THE MICROCONTROLLER AND THE WIRELESS RECEIVER AND TRANSMITTER

One downfall to having the transmitter and receiver as two separate units is that they cannot be active at the same time. As long as the transmitter is active, the receiver will not pick up any new wireless transmissions. This was resolved by keeping the transmitter powered down until it is needed to send a packet. This is accomplished by using a transistor, controlled from Pin 4 of the microcontroller, as a switch that interrupts the power to the device (Figure 24).

The code inside of the Transceiver library that enables and disables the transmitter is shown in Figure 25. The delay functions allow time for the receiver to finish if it was getting a signal and allows the transmitter to turn off before the receiver turns on again.

```
154  void Transceiver::turnOffTransmit()
155  {
156      /* includes delays to make sure that the
159      delay(75);
160      digitalWrite(txSwitch, LOW);
161      delay(75);
162  }
```

FIGURE 25. CODE FROM TRANSCEIVER.CPP TO DISABLE THE TRANSMITTER.

Conclusions

In this paper, a web based remote monitor and control system for power socket has been designed and developed. The test results show the system works very well. With very minimal modification to the code, additional units of varying types can be implemented in this system. Although only power sockets were created, this project is designed and engineered with the intention of only being the backbone to a larger network of possible devices with the ability to add features with relative ease.

REFERENCES

[1] Home automation (2017), https://en.wikipedia.org/wiki/Home_automation.

[2] Jaeseok Yun, Sang-Shin Lee, Il-Yeop Ahn, Min-Hwan Song, Min-Woo Ryu, Monitoring and Control of Energy Consumption Using Smart Sockets and Smartphones, Computer Applications for Security, Control and System Engineering. Communication in Computer and Information Science, Vol 339. Springer, Berlin, Heidelberg.

[3] Morales, R., Badesa, F., Garcia-Aracil, N., Perez-idal, C., Sabater, J.M., Distributed Smart Device for Monitoring, Control and Management of Electric Loads in Domotic Environments, Sensors 2012, 12, 5212-5224.

[4] Lien, C.H., Chen, H.C., Bai, Y. W., and Lin, M.B., Power Monitoring and Control for Electric Home Appliances based on Power Line Communication, 2008 Conference Proceedings of Instrumentation and Measurement Technology, British Columbia, Canada, May 12-15, 2008.