ALPACA

(Adaptive Lighting Power and Control Applications)

REPLACE WITH PDF COVER PAGE

Jeremy Gamache
Hugo Garcia
Roy Mullins
Anton Nathanson
Ryan Rosales
Gilberto Valenzuela
Omar Vega

Sponsored by:

San Diego Gas & Electric

Submitted to: John Kennedy and Lal Tummala

Department of Electrical and Computer Engineering

San Diego State University

December 2013

Table of Contents

ABSTRACT 1

INTRODUCTION 2

HARDWARE SOLUTION 4

RELAYS AND POWER SUPPLY 5
CIRRUS CS5480 ENERGY MEASUREMENT IC 5
MSP430F5529 6
PCB DESIGN 7
ANALOG FRONT END CIRCUITRY 8
EWO INTERFACE DESIGN DESCRIPTION 9
EWO EMBEDDED FRAMEWORK DESIGN 10
LIGHTING DISPLAY 16

SOFTWARE SOLUTION 17

WEB INTERFACE 17
OSI-PI SDK 19
TIMING 19
SWITCHING 20
FINAL PRODUCT 20
OSI-PI DATABASE 21
CONNECTING TO OSI-PI 21
CREATING POINTS 22
VIEWING POINTS 23
MANAGING DATA 24
CLIENT APPLICATION 26
COMMUNICATION 26
SWITCHING 27
PARSING 28

CONCLUSION 29

REFERENCES 31

APPENDICES 32

PCB SCHEMATIC 32
EWO EMBEDDED CODE 32
CLIENT APPLICATION CODE 34
PROJECT SCHEDULE 37
BILL OF MATERIALS 39

Abstract

The Energy Wise Outlet (EWO) by ALPACA provides a comprehensive energy monitoring, automating, and educational solution. The EWO allows users to not only monitor their energy use but learn about their power consumption and ways to reduce it. The EWO also has the ability to provide visual feedback in the form of LEDs to inform the user of their current level of energy use. This system will also allow for remote switching of their selected loads from any Wi-Fi-enabled device such as a cell phone, tablet, or computer. The outlet is rated for up to two sixteen amp loads and accommodates a wide range of household appliances.

As previously mentioned, a web-based application is available to the consumer to track their power consumption. The Web application will function by querying the SDG&E OSI PI server and displaying real-time or collected data. The SDG&E tier system is taken into account when calculating cost along with forecasting future costs as well. In designing this web application we are taking into consideration the bigger picture of creating a software infrastructure for a complete home monitoring system. The Client application serves to connect the consumer's PC with the EWO as well as to display real-time data. Data will be automatically uploaded to the PI server by means of a background C# application.

Introduction

Due to the steady rise of consumer electronics and appliances within each passing year, it has become increasingly difficult for the average household to determine the extent of their energy use. This recent trend has become a key issue during peak energy use times, causing even more stress on an already overloaded San Diego power grid. Our solution to this problem is the EWO, a smart outlet that combines an intuitive web interface with a simple client application to provide the user with an intelligent monitoring and automating system. The result provides detailed energy use statistics as well as usage cost analysis. This application could also provide data to SDG&E along with allowing the utility to shed non-essential loads.

The EWO provides power use and cost statistics in addition to detailed energy use statistics such as the voltage, top and bottom current of the outlets. Users have access to this information via a user friendly graphical web interface or web-based application. This information is in the form of easy-to-read charts, graphs and tables allowing for a variety of personalized reports and settings. The home automation aspect of the EWO also has several benefits over traditional switching receptacles by allowing users to wirelessly switch each socket on the outlet independently. This allows users to separate essential and non-essential loads within close proximity. Switching of the EWO is made for convenient thanks to web interface, which is accessible from any Wi-Fi-enabled device.

The consumer can access the website to view data on real-time, as well as past energy usage. The user interface also has the option of plotting a graph of the usage and cost of a particular outlet. Advanced features in the EWO consist of forecasting monthly bills and the effects of weather on energy use. The forecasting feature warns the customer if their energy usage indicates that they might be crossing over to a higher billing tier which will cost them

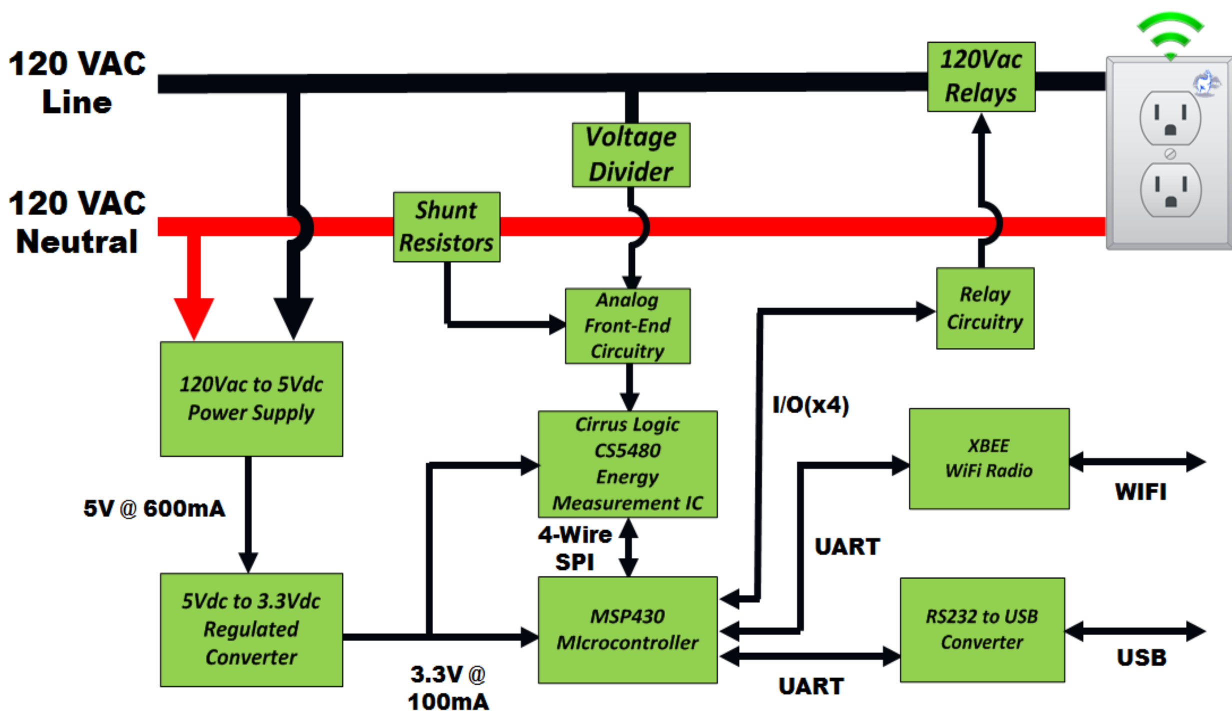
more money. It displays their estimated monthly bill and gives suggestions as to how they can reduce their energy use and climb back down to a lower billing tier.

The C# GUI Client application is used to connect the user's PC with the smart-outlets as well as handle data transmissions from the parser. The client application displays energy usage in real-time while giving the user control over the status of the outlets. The C# application also serves to relay data to the OSI PI software which in turn will update the servers.

Many products on the market today offer similar features to the EWO but with the trade-off of being an external plug-in. This takes up wall space and often requires visible placement in order to obtain readings. The EWO resolves this issue by fitting in an existing electrical receptacle. By being fitted within a wall not only will the EWO not be visible to a consumer, but because of the wireless measurement the placement of the device becomes irrelevant.

HARDWARE Solution

The hardware solution for the EWO was developed to meet and exceed the requirements set forth by the client. The EWO consists of two Printed Circuit Boards (or PCB): one is handling the high voltages and currents, and the latter contains the measurement and control circuitry; or low-voltage components. The figure below contains the hardware block diagram of the EWO:



The high-voltage board accepts the Line and Neutral feed from the wall, and splits them into a two separate circuits that allow for individual control of each socket in the outlet. We placed a relay on each circuit, thus allowing the user to switch the sockets on and off separately. There is also a power supply which takes 120VAC input from the branch circuitry and outputs 5VDC to power the relays and the components contained on the low voltage board.

The low-voltage board contains the more sensitive electronics that provide energy measurement and control, as well as external communication to server via a wireless module.

The measurement is done using the Cirrus Logic 5480 measurement IC, which is perfectly suited for this application. The microcontroller we chose for the design is the MSP430F5529 from Texas Instruments. This microcontroller has low power consumption and plenty of onboard storage space and peripherals. The communication via Wi-Fi is done using the Digi XBee S6B Wi-Fi module. Each of these components was chosen for their specific advantages that met our design constrains. Below is an outline and description of the chosen components:

Relays and Power Supply

The relays on the high-voltage board are made by Schrack, model number RT314F05. This relay is a dual latching single-pole dual-throw (SPDT) relay. This allows us to send a single command to the relay to switch in one direction; holding this state, until a second command is sent to switch in the opposite direction. The continuous current rating is 16 Amps, meaning we can run large loads on our outlet without the risk of overworking the relays. These relays are a great example of our choice to have quality components that give us a robust design, and are designed to prevent failures under normal operating conditions.

The power supply on the high-voltage board is the RAC03-05SC/277, made by Recom. This power supply takes the 120VAC from the branch circuitry and outputs the 5VDC necessary to supply our relays and ICs. This model can output a maximum current of 600mA, which is well over the total current draw of all of our components combined.

Cirrus Logic CS5480 Energy Measurement IC

This specialty device is a high-accuracy energy measurement analog front end that incorporates three 4th order Σ-Δ analog-to-digital converters for low noise and a 4000:1

measurement dynamic range.

The CS5480 will provide the following measurements to the MSP430 via SPI:

- Line Voltage
- Line Current
- Line Frequency
- Power Factor
- Real Power (kW)
- Apparent Power (kVA)
- Reactive Power (kVAR)

The EWO will provide one voltage input; using a voltage divider network, and 2 current inputs; using two shunt resistor networks on the 120VAC neutral, to the CS5480. A 4-wire SPI interface will be used for sending data and receiving commands from the MSP430.

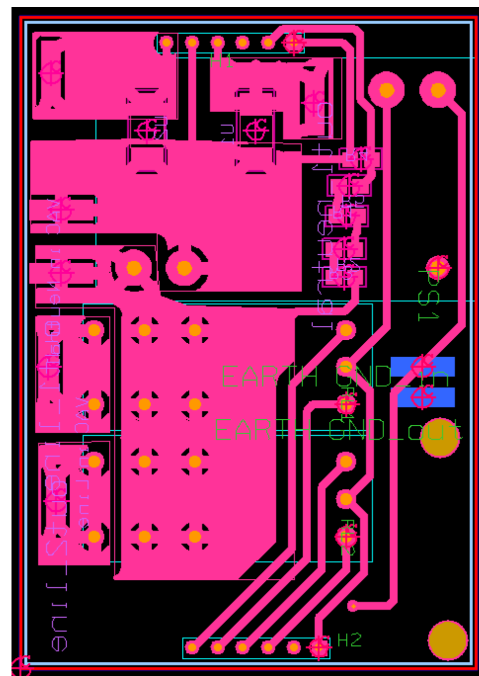
MSP430F5529

The microcontroller selection process was defined based on four main categories: power consumption, form factor, programming difficulty and resource availability. In the initial design stages we had two microcontrollers that met our needs: the 32-bit STM32F0 and the 16-bit MSP430F5529. The STM32F0 is comprised of a 32-bit Cortex-M4F ARM core, 48MHz clock, 2 UARTS, 2 SPI/I2C and a 100-pin. The MSP430F5529 is a 16-bit RISC, 25MHz clock, 2 UARTs, 2 SPI/I2C and an 80-pin package. Both microcontrollers have enough processing power and peripheral interfaces that met our needs. We decided for the MSP430 due to its ample resource availability, the ECLIPSE development environment, and its low power consumption. In addition, since our project did not require the additional computational power the STM32F0 provided, the MSP430 with its 16bit architecture and 25MHz clock was the best decision. The STM32F0 was a potential option, but its 100-pin package made it challenging to integrate into our custom design.

PCB Design

One of the most fundamental elements in our final product was the design of our PCB. For our project we opted for a two-board design separating the high voltages and currents from the sensitive IC's for the measurement and control circuitry. The software we used to design the PCBs is Mentor Graphics. This powerful software package allowed us to create our custom components and symbols, which were required for our design. Although some components were already included in the library, other components, such as the power supply and relays, needed to be manually entered into the program. The outer dimensions of the boards were a very important design factor. We designed our boards to fit inside a single gang box behind an electrical outlet. With this in mind we chose to have a board with dimensions of 2.6" tall by 1.8" wide.

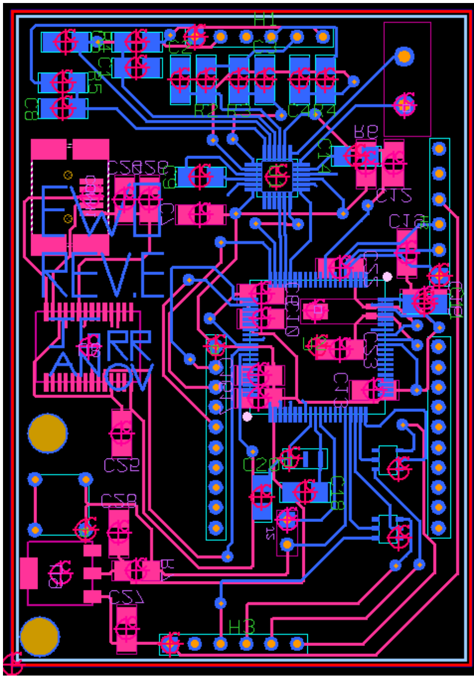
After the necessary parts and symbols were made on Mentor Graphics Design Capture, we began to design the high-voltage board. This design required special considerations, in that it needed to handle 120VAC and up to 16 Amps. To make the board robust enough to handle these requirements, we researched the thickness of the traces required to fulfill the goals. The two main aspects of PCB traces are the width and clearance. The clearance of the traces (or space between them) is determined by the amount of potential voltage between any two traces. Since the highest voltage difference anywhere on the board is 120VAC, we found that fifty-thousandths of an inch (th) would be enough to prevent arcing. The width of the traces is determined by the maximum amount of current the traces could potentially be carrying. Since



we were designing towards 16 Amps peak current, this meant we would need to have widths of

400 th for each of the high current traces. However, this requirement exceeded the allowed space on our constrained PCB dimensions. In order to solve this we chose to use copper pours; or areas of copper, instead of traces, so that the current has enough copper to flow through an area to dissipate heat. These copper pours were designed to be as small as possible so that the currents had a short distance to flow through, reducing the possibility of failure.

The low-voltage board did not need the same considerations as the high voltage PCB. The low voltage board contains the 24-pin Cirrus Logic measurement IC and the 80-pin microcontroller, along with all of the supporting capacitors and resistors needed to make them function properly. The high-voltage PCB required trace considerations with little complexity, whereas the low-voltage PCB did not need special trace requirements yet was a lot more complex to design. Mentor Graphics did a lot to help with complex circuitry by using the connectivity from the schematic and displaying “hairlines,” showing which pins needed to be connected. The placement of the components was extremely important so that laying out the traces did not become too complex to debug and fix.



Both PCBs were designed with their own considerations to ensure a robust design that performs as needed. Upon testing the high voltage board there was no problems such as burning traces (due to too much current) or arcing (due to not enough clearance between traces). The low-voltage board was refined through multiple iterations to allow for simple debugging and ease of assembly.

Analog Front End Circuitry

The analog front end circuitry consists of a voltage divider network, bypass capacitors, and two shunt resistors on the neutral side. The voltage divider network will step down line

voltage to a more reasonable voltage that the CS5480 can handle. The resistor selection will set the scale factor the CS5480 compute engine will use to provide the most accurate measurements. The two shunt resistors will be very low resistance power resistors that will provide a differential current measurement for the CS5480 compute engine. This circuitry will follow the recommendations suggested by Cirrus Logic, in order to provide the most accurate energy measurements.

Relay Circuitry

In order to energize the relay coil at the higher 50mA rating, an inductive load driver manufactured by ON Semiconductor will be used to interface the 3.3V/6mA MSP430 circuitry with the 5V/50mA dual coil relays. This MOSFET based inductive load driver provides all the required circuitry and protection to safely drive the dual coil 120VAC rated relays using the output pins of the MSP430. With its small form factor, this inductive load driver will save precious PCB real estate, by eliminating the need for: 4-2N222 NPN transistors, 4 resistors and 4- 1n4004 diodes.

EWO Interface Design Description

The communication and control layout of the EWO consist of 4 main interfaces:

- ***Microcontroller to FTDI Chip Interface:***

  This interface consists of a Universal Asynchronous Receiver Transmitter (UART) between the MSP430 and the FT232RL IC. This interface has a link speed of 115200 baud. No hardware handshaking is implemented. A 64-byte circular buffer stores data from FTDI chip without sacrificing microcontroller memory. Several fail-safe checks were added to ensure messages received are of the correct format.

- ***Microcontroller to XBee S6B Wi-Fi Interface:***

This interface consists of a Universal Asynchronous Receiver Transmitter (UART), between the MSP430 and the XBee S6B Wi-Fi module. The link speed of this interface is 115200 baud. No hardware handshake is implemented. A 64-byte circular buffer stores data received wirelessly; through the XBee module, without sacrificing microcontroller memory. Several fail-safe checks were added to ensure messages received are of the correct format.

- ***Microcontroller to CS5480 Interface:***

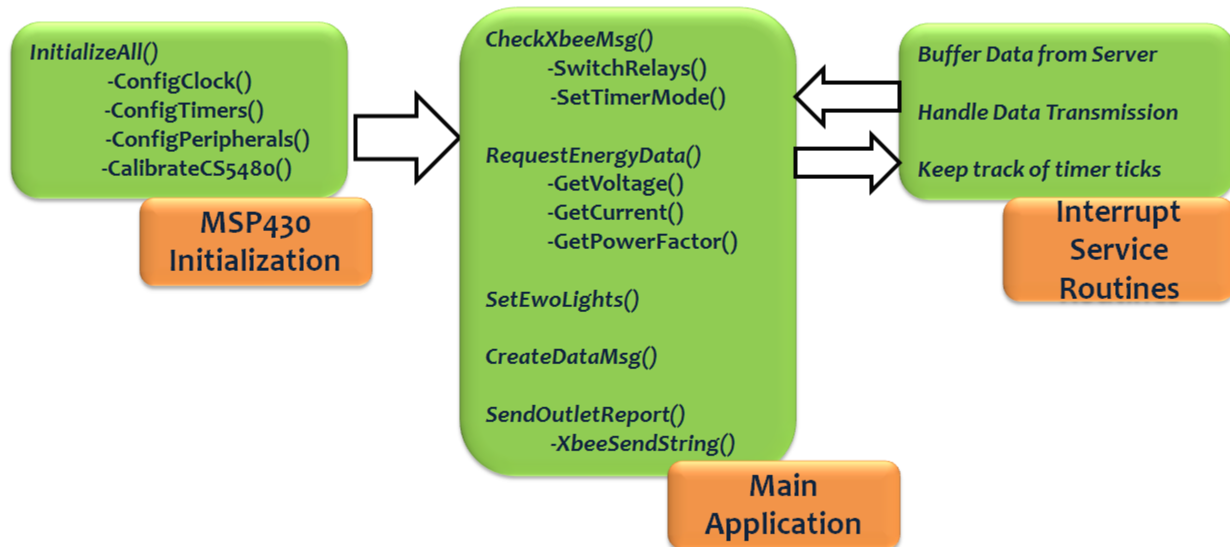
This interface consists of a 4-wire Serial Peripheral Interface (SPI) link between the MSP430 and the Cirrus Logic CS5480. The link speed of this interface is 2MHz. A separate GPIO pin is used to drive the reset circuitry of the CS5480. The microcontroller requests data from the CS5480, by sending the Page and Register Address of interest. The CS5480 responds with a 3-byte message; i.e. 0x112233, where '11' is the most significant byte of the 24-bit word. Communication between microcontroller and CS5480 is on-demand and is controlled via a chip select signal; which initiates and terminates communications.

- ***Microcontroller to Relay Interface:***

The interface between microcontroller and relays consists of 4 GPIO pins; where each relay utilized 2 GPIO pins to switch on/off electrical power on a socket. Two relay drivers are used to provide an interface from the 3.3V circuitry of the microcontroller to the 5V circuitry of the relays. In addition, the relay drivers provide protection from current surges; in the form of free-wheeling diodes, that protect the sensitive electronics and the microcontroller.

EWO Embedded Framework Design:

The figure below describes the top-level implementation of the embedded framework of the EWO:



The EWO was coded using ANSI C principles and the built-in driverlib API from Texas Instruments. The design rationale was to develop functional code that is readable, robust and scalable. An example of this is the usage of high-level function calls to modify register value, as opposed to writing the abstract name for the register and the hex value to be written. The following code snippet displays the high-level function calls that were used through the project:

```
USCI_A_UART_initAdvance(USCI_A1_BASE,
                USCI_A_UART_CLOCKSOURCE_SMCLK,
                6, //UCBR0  = 6 See MSP430F5529 User's Guide Page 910
                8, //UCBRSx = 8 See MSP430F5529 User's Guide Page 910
                0, //UCBRFx = 0
                USCI_A_UART_NO_PARITY,
                USCI_A_UART_LSB_FIRST,
                USCI_A_UART_ONE_STOP_BIT,
                                USCI_A_UART_MODE,
                                USCI_A_UART_OVERSAMPLING_BAUDRATE_GENERATION ));
```

The above function call is used to configure a UART peripheral on the MSP430 at 115200 baud using 12MHz clock speed. As seen above the function is fairly easy to comprehend, as most of the parameters are descriptive strings as opposed to hexadecimal values.

```
UCA0CTL0 = 010000000; //No Parity, LSB First, 1 Stop bit, UART Mode
UCA0CTL1 = 100000000; //Use SMCLK as clock source
```

As seen above, the high-level function call is cleaner and easier to comprehend than setting individual bits in a register. This design approach proved to be easier to debug and develop throughout this course.

Another design approach was to create custom functions that were tailored for each peripheral that was used in the EWO. This was performed by creating separate ".c" and header files for each peripheral. An outline of these functions is seen below:

- Main.c
- ConfigMCU.c
    - ConfigMCU.h
- CS5480.c
    - CS5480.h
- LED.c
    - LED.h
- MainFunctions.c
    - MainFunctions.h
- XBEE.c
    - XBEE.h
- Relay.c
- ISR.c
- FTDI.h

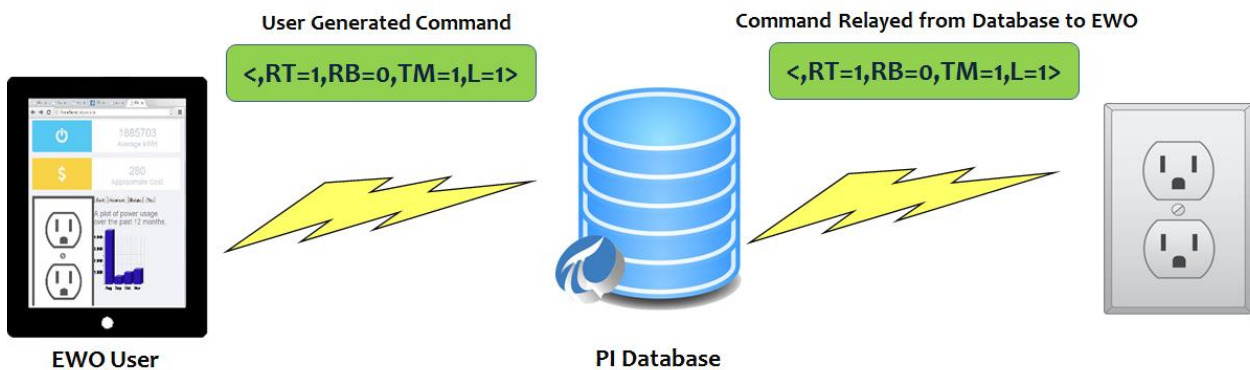
The type of project structure allows for easier content management if used on a CVS system, and also for global variable usage across files.

The main challenge faced during development was getting the SPI communication between MSP430 and Cirrus Logic CS5480 IC. This was a challenge because of the limited knowledge about SPI and the lack of description on the datasheet for the CS5480. The datasheet for the CS5480 was not descriptive enough to allow the user to determine some of the more obscure register settings required for SPI to operate correctly. These parameters were found through countless iterations of experimentation on the bench using the MSP430, the CS5480 development board and a logic analyzer. Another challenge that was encountered was getting the

XBee Wi-Fi module to communicate reliably with the MSP430. This was matter was resolved in about 2 days and worked consistently for the rest of the semester.

In order for the microcontroller to manage all tasks that were required several flags were implemented to act as schedulers for certain events. For instance, when the EWO received a message from the EWO server, the microcontroller did not process that command until the message received flag was set. This was useful because it ensured that the correct and complete message was received by EWO and acted upon every single time. These types of flags were set for incoming messages from server to EWO, and for managing when the EWO requested energy data from the CS5480. The flags for energy data requests were managed by a timer that was user configurable. All communication that was either incoming or outgoing was handled by interrupt service routines. These routines were kept as short as possible to prevent any unwanted side-effects and interrupt priorities were determined for the same reason. The highest priority interrupt, which was UCSIA0, was assigned to the XBee Wi-Fi peripheral.

The figures below describe how the EWO receives and process a command from the Server:



First, a command is sent by the server in the following format: **<,RT=1,RB=0,TM=1,L=1>**, where “RT” and “RB” correspond to Relay Top and Relay Bottom commands. The values that the server sends for relay commands are Boolean, where “1” instructs the EWO to allow either top/bottom socket of an outlet to deliver electrical power and “0” instructs the EWO to disable

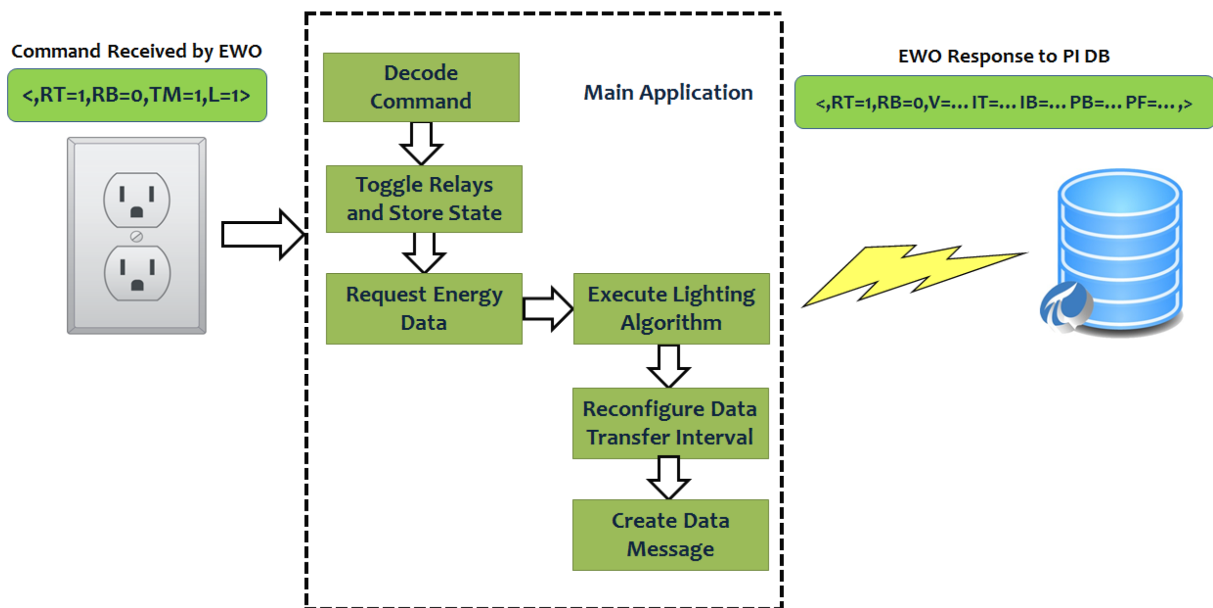
electrical power for that socket. The next command is “TM”, “TM” stands for Timer Mode.

The parameters that are sent to EWO are:

- “0” for no data transmission
- “1” for 3 second data transmission (Streaming Mode)
- “2” for 5 minute data transmission (Normal Mode)

Lastly, the “L” command refers to the EWO lighting display. This command is also a Boolean, and if a “1” is sent the EWO lighting display is enabled, otherwise it is disabled. Once the command is received by the EWO, it is then stored in a buffer that is processed on the main application once the message complete flag is raised.

Now that the command is stored in the EWO’s flash memory and the message complete flag is set, a function called “**CheckMSG(xbee_rx_fifo);**” is invoked. This function checks the contents of the messages and reacts upon the command received. The figure in the following page describes the processing of a command message received by the EWO:



As seen in the figure above, priority is given to relay switching and data acquisition from the CS5480. The execution of the lighting algorithm has less priority and could be disabled if user desires. Lastly after all commands have been processed and executed, the EWO gathers all those parameters, packages them and sends wirelessly back to the PI database.

One problem that was observed while testing the EWO with light loads was the abrupt interruption of data when a high-amperage load was switched on or off. To overcome this problem, a message compensation algorithm was developed. This algorithm ensured that data did not get corrupted any time a high-amperage load was switched on/off, or if for some reason a glitch in the microcontroller corrupted the data. In addition this algorithm allowed the EWO to convert the 24-bit values received from the CS5480 into ASCII characters that allowed the client application to perform effectively.

Below is an example of this algorithm:

```
/************************************************************************
*
*           FUNCTION:                   MsgCompensator
*
*           DESCRIPTION: Function is used to compensate for any glitches that may
*                                 occur when a high current load is connected to the EWO.
*
*           ARGUMENTS:                  const char* dataBuffer, int startIndex
*
*           RETURN:                                     None
*
*           NOTES:   This algorithm had to be developed because it was found
*            that a high current load was corrupting the messages being
*                       retrieved from cirrus chip
************************************************************************/
void MsgCompensator(const char* dataBuffer, int startIndex)
{
volatile int i=0,j=0,n=0,y=0; //temp variables
volatile char temp[8]; //temp buffer

for(i = 0 ;i < 8; i++) //loop through an array of 8 elements
{
            temp[i] = '0'; //Fill each index of temp buffer as 0
            //If data is an ASCII value between 0-9
            if(dataBuffer[i] >= 0x30 && dataBuffer[i]<=0x39)
            {
                        temp[i] = dataBuffer[i]; //store ASCII value in temp buffer
                        n++; //increment counter
                        y=n; //set y equal to number of legit ASCII numbers found
            }
            else
            {
            //Otherwise if value is not an ASCII number
            msgOut[(startIndex+i)-n] = 0x30;
    //simply make it an ASCII zero
            }

}
            j = startIndex + (i-n);
//make j equal to number of values that were real ASCII numbers
```

```
            for(i=0;i< y;i++)
            {
               msgOut[j+i] = temp[i];
            //start filling msgOut buffer after the zeros padded values.
            }
}
```

Basically, the MsgCompensation() algorithm takes a pointer to char buffer array which converts the long integer into an array of ASCII characters. If the ASCII character is not a decimal number from 0-9, then it makes that invalid character a decimal 0, represented by ASCII value 0x30. After it checks all the values on the original array it takes the values that were converted to zeroes and inserts them, starting on index 0, and continues until all indexes of the array contain a valid ASCII value in the range of 0-9 (decimal).

Lighting Display

One of the unique features of the EWO is the adaptive lighting display surrounding the outlet. The EWO utilizes RGB LEDs from Rohm Semiconductor, product number MSL0201RGBW1. These visual indicators are specifically calibrated to emit a colored aura dependent on the consumer's current power usage, covering the spectrum of green, lime green, yellow, orange, orange red, and red. When the EWO begins to glow red (indicating excessive energy use) this will trigger an alert on the web application, which will be detailed later in this report. The LEDs themselves are monitored and controlled by the MSP430-F5529 through a pulse-width modulator algorithm.

In order to showcase the lighting display of the EWO we also included a 3.08" by 4.902"

inch of acrylic with a thickness of an eighth of an inch. The program used to design this piece was InkScape, which allowed us to save the file as a .dxf in order to cut the piece of acrylic. The RGB LEDs were specially designed to face the inner part of the acrylic within the EWO to emit the real-time color display.

SOFTWARE Solution

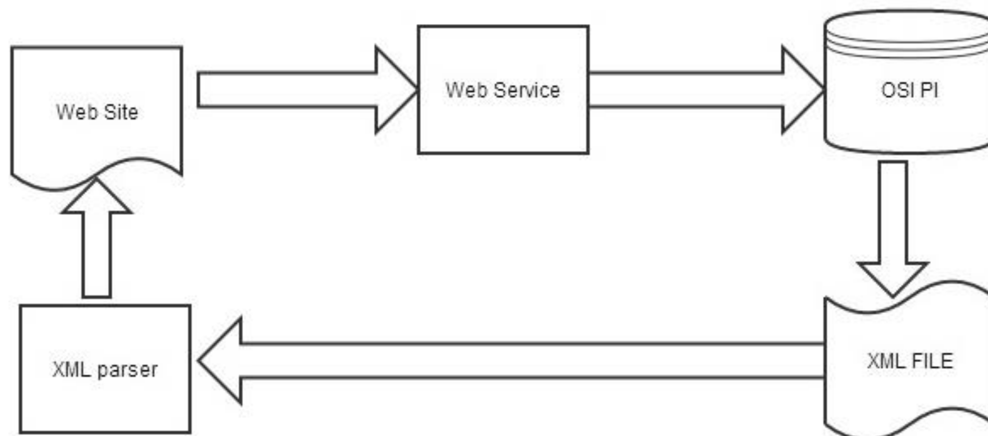
The software component of the EWO system is comprised of three parts: The GUI Web Interface, the OSI-PI Database, and the client application. The web interface serves as a graphical representation of budgeting options as well as a display of consumed power. The OSI-PI Database stores the large amount of data-points needed to make budgeting and power consumption calculations, and the client application's function is to wirelessly connect the EWOs with the web interface and database. The web interface is located in the "Web_CFG" solution while the code pertaining to database management and the client application is in the "AlpacaFinal" solution.

Web Interface

It was our goal to design a user interface that was easy to use and adaptive to a customer's need. We opted to use a responsive website that would work across multiple platforms and multiple devices. This eliminated the need for mobile app installation and it gave us the opportunity to work on one design instead of creating separate designs for different browsers. Furthermore, it created a common interface that customer's would feel comfortable using on their different devices. In other words, our user interface has the same functionality, capability, and appearance regardless of the platform or device on which you are viewing it.

We displayed relevant information on the user interface that could give the user a glimpse of the instantaneous data that is being transferred from the EWO to our client application. Although our display of voltages and currents offer an interesting visual representation of the outlet's power usage, a customer would be mostly interested in the associated cost with their energy use. We displayed the customer's monthly bill over a span of several months so that they can be conscious of what their energy expenses have been over time. Additionally, we used a live chart to display a selected current as well as HTML to show the updated data from each EWO. Although our cost for individual outlets is a forecast of what the customer's cost could be at the end of the month, it still gives them useful information which could lead to conscientious decisions that may lead to reduced energy consumption.

The purpose of our website was to display the data being transmitted from the EWO device as well as giving the customer useful information about their energy use. Our goals for the website consisted of successfully pulling data form OSI PI database, updating data on the website in a timely manner, switching outlets ON/OFF from the web interface, and the completion of the website with some added extra features. The following is a flow chart of how our website works:



OSI-PI SDK

In order to successfully pull data from OSI PI, we obtained access to the OSI PI SDK which was provided to us by our sponsors through the OSI Soft website. The SDK for Microsoft Visual Studio gave us the tools we needed in order to update and make requests to the database. Here is a sample code segments which illustrates how the SDK works:

```
public static PISDK.Server PI_Server;  // Creates instance of Pi Server
PISDK.PISDK SDK = new PISDK.PISDK();  //Creates new instace of PI SDK
        PI_Server = SDK.Servers[PiServerName]; // Assign PI server to local machine

        string username = "hpm";//hugo userName
        PI_Server.Open("username"); // Open connection through default user

        PIPoint currentPoint = PI_Server.PIPoints[currentTagName];//create a point for the current
        PIPoint voltagePoint = PI_Server.PIPoints[voltageTagName];//create a point for the voltage

        PIValues voltageValues = voltagePoint.Data.RecordedValues(startTime, endTime); //get the voltage values at that
      point
        PIValues currentValues = currentPoint.Data.RecordedValues(startTime, endTime);  //get the current values at that
      point
```

Timing

In order to display data in a timely manner on the website, we used two 500ms timers that queried data every time they went off. We had one timer on the client side which queried the XML data as well as called update functions. On the server side, we had one timer which queried OSI PI in order to store updated results onto our XML file. Here is a sample code segments which show how we used timers in our system:

client side

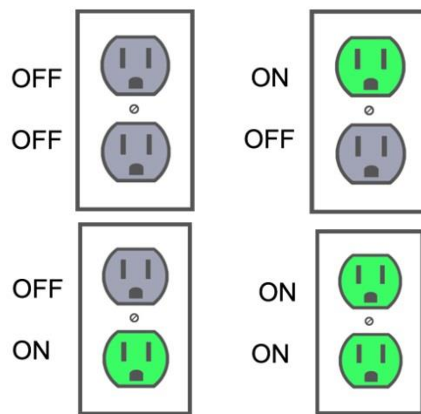
```
var myVar = setInterval(function () { myTimer() }, 500); //Timer set to go off every second
function myTimer() {
   var d = new Date();
   var t = d.toLocaleTimeString(); //t holds a printable string containing the current time
   $.get("pi3.xml", {}, function (xml) {
      $('piValue', xml).each(function (i) {
.
.
.
}
```

Server Side

```
System.Timers.Timer aTimer =  new System.Timers.Timer(500);
public static void OnTimedEvent(object source, System.Timers.ElapsedEventArgs e)
```

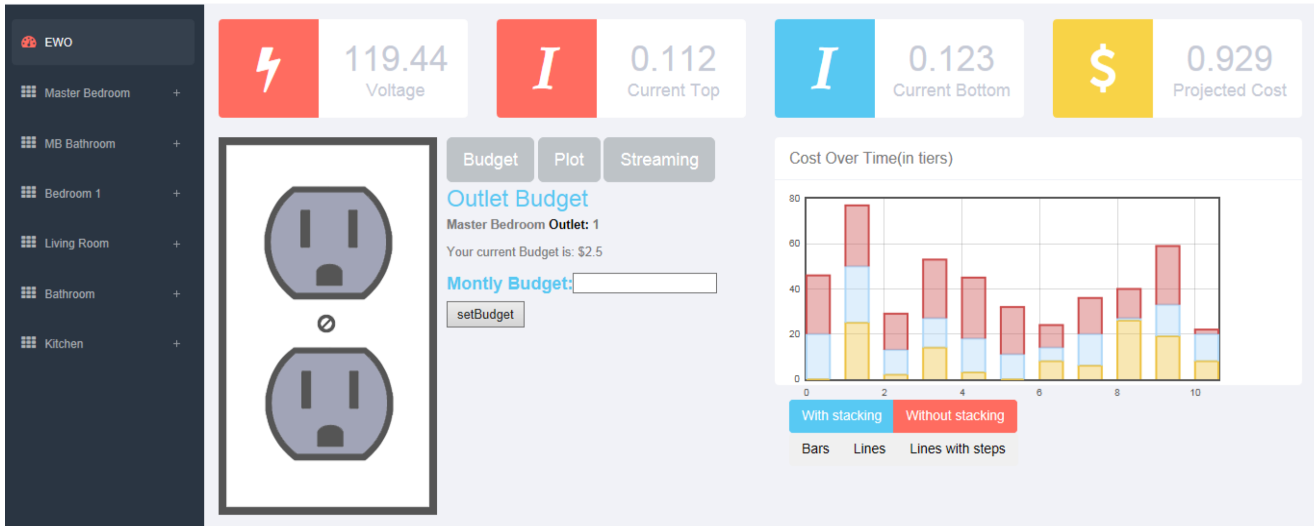
Switching

The switching capability of our website consisted of designing something visually indicative of the state of each plug as well as programming the functionality. For the visual portion of the design, we decided to use the color gray to indicate the OFF position and green to indicate ON. The selection of the outlet of choice is done by clicking the Room on the left sidebar and then choosing the outlet you want. Once you have the outlet selected you can then switch them ON or OFF. The following are the images which are displayed on our website when the user clicks on an outlet:



Final Product

Our completed website displayed currents (top and bottom) and the voltage drop on each outlet. Additionally, the projected cost of the outlet was calculated and displayed as well. The bar charts displayed the customer's electrical cost of their overall home usage over six months. We generated this data by using sample data that was provided to us from our sponsors. We also included a live chart which displayed the Bottom current of the selected outlet which changed as soon as the data was sent from the EWO. Provided in the following page is a screenshot of the PC, Tablet, and phone versions of our completed website:



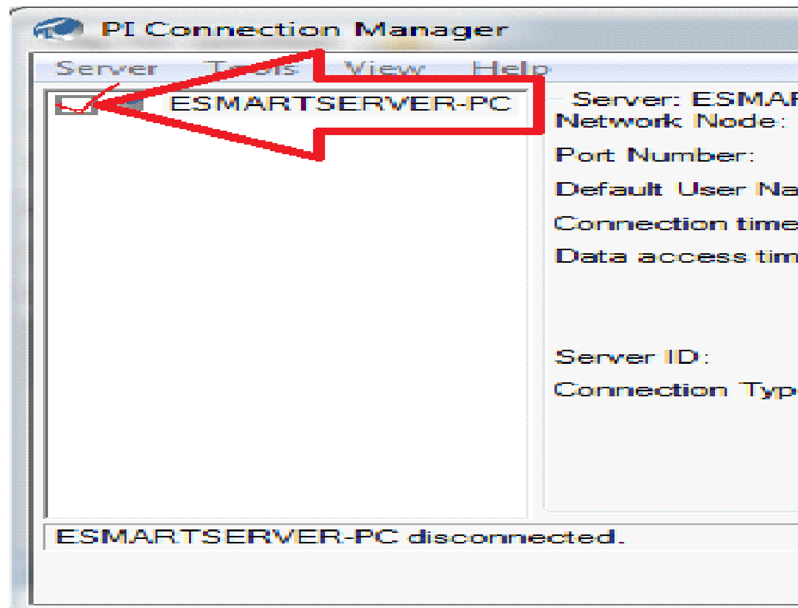
OSI-PI Database

Connecting to OSI-PI

The OSI PI server resides on the Lenovo PC provided by SDG&E. There we use SMT in order to manage the database. By going to the Windows start up menu and selecting the PI

System Management Tools Software. This will open up the software that will allow you to connect and manage the data server.

Once the program called PI System Management Tools is opened you can just check the "ESMARTSERVER-PC" and make sure you are connected to the internet. This will give you full access to the data base in order to store, analyze, and manage data. This program can be downloaded via the Vcampus website (http://vcampus.osisoft.com/). A valid login is required to be able to download any of the OSIsoft programs. When the program loads you will need to type in the connection information. If connecting for the first time, click File > Connections to be able to type in the connection information. The following is an image of the connection information.

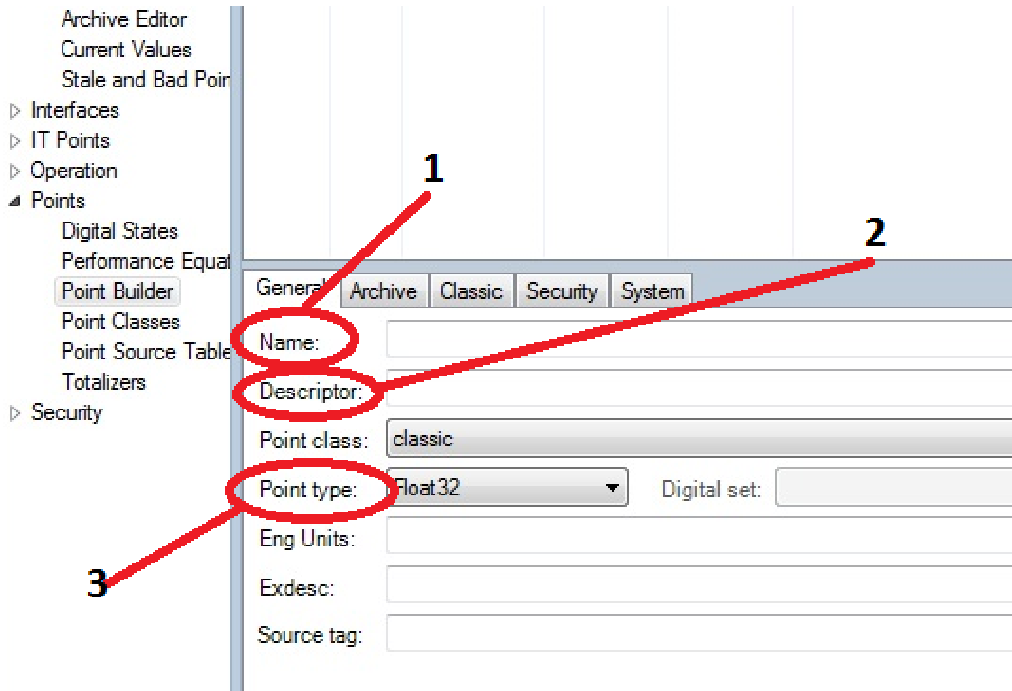


Creating Points

Important key concepts to understand are how to create tags or "point" to store data then learn how to manage it. The first thing that we need to know when we have the SMT software open is how to create a point or "tag" that will represent a measurement that will be displayed in the web interface. Once the SMT software is open we can start to create a tag by selected the Point Builder feature.

Once you have selected the point builder feature you can start by

1. Adding a tag name to represent a measurement (Voltage, current, etc.)
2. Add a Description
3. Setting the point type (Integer, float, string)

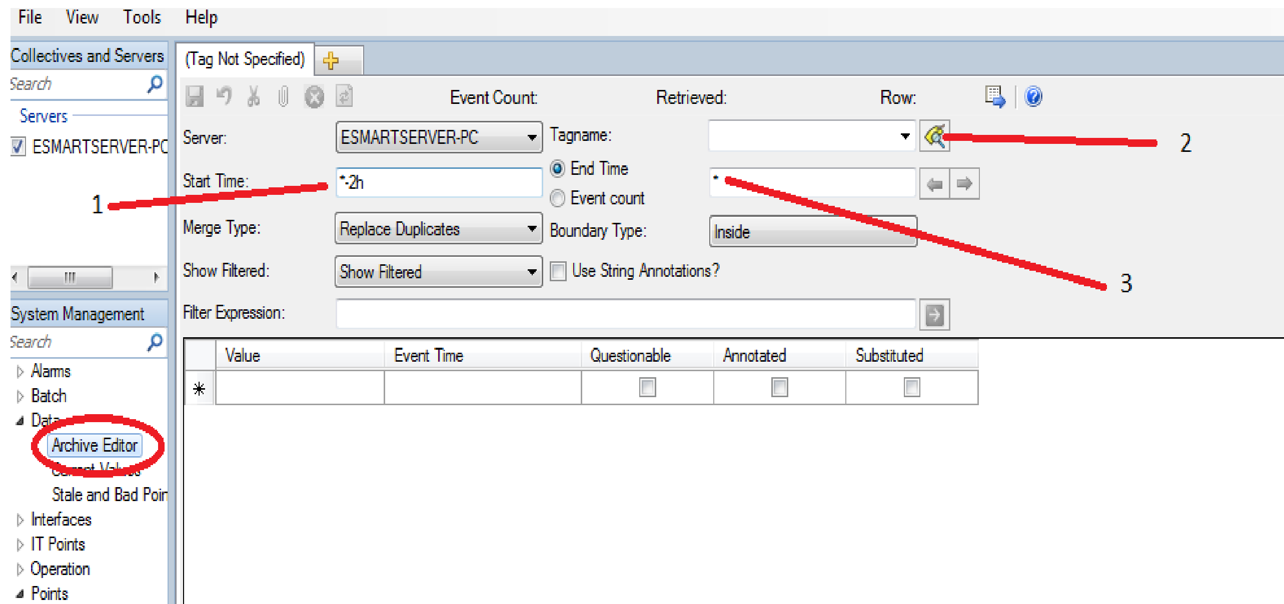


Viewing Points

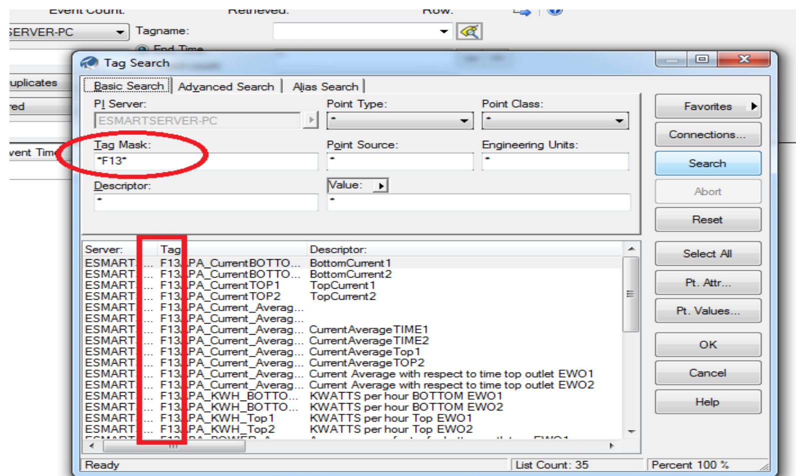
Using the Archive Editor feature in the SMT we are able to look at the history of any tag that exist and manage how far into time we want to view data for.

1. We are able to set the time in which we want to begin out history search. For example, in the image below we see that *-2h is that it is inserted. This will look for every value in the data base in the last 2 days.
2. The search label will allow you to search for tags by name

The "end time" will specify the end time you want to look at data for a certain tag.



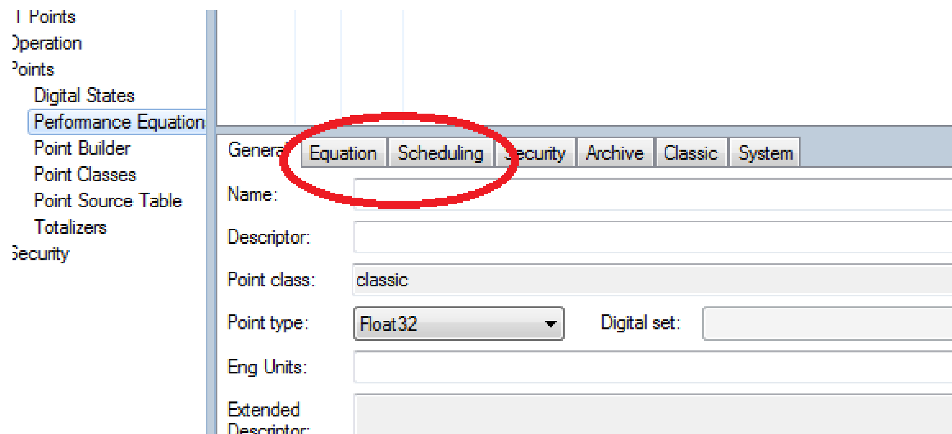
The search feature facilitates your search for tags. By simply inserting "*" at the beginning at end of key words of a tag you should get a window with all the tags with those characters. For example when inserting *F13* in the search box we were given the following window:



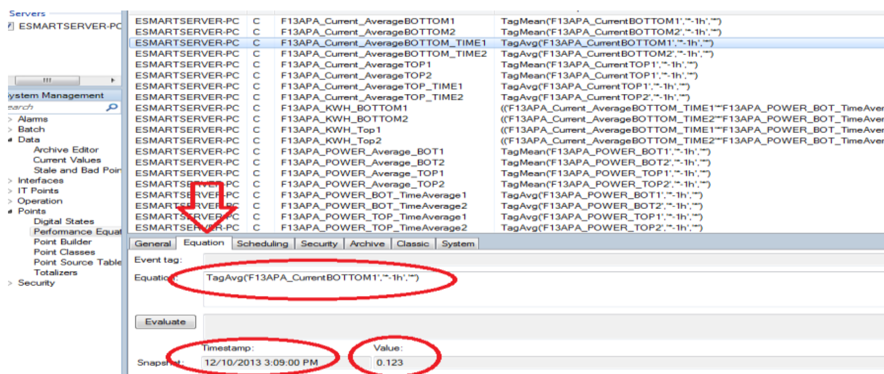
Managing Data

Next we will go over how we can manage and manipulate the data that we have collected and stored in the OSI PI server. We will be using a feature in the SMT called "Performance Equations" that allows us to create a new tag that will be populated from the outcome values of a calculation made with at least one

other tag.

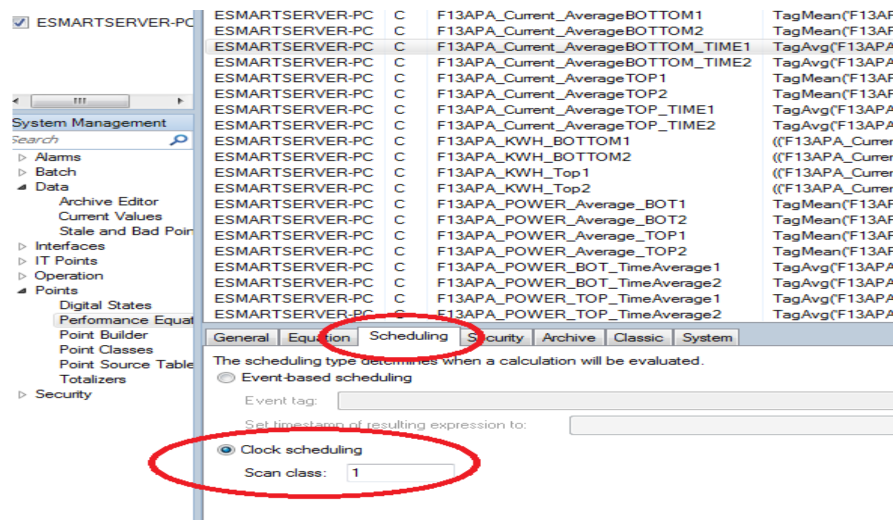


Once we have at least one tag or point that we want to manage and calculate an equation for (average, last value, repeated values, etc.) we can use this feature to manipulate the data. This procedure is fairly simple. It starts out by creating a tag just the same way we learned before, however, this tag will have an “Equation” and “ Scheduling” sections to it.



Here can analyze how the “Equation” feature works. Once a tag is create all we need to add is the equation we want this tag to perform. The result of this equation will then be stored as a value in this point. We can also use this feature to view the result of a calculation to verify that the calculations are correct. We can also see how the value that is being calculated is recorded at a specific time, time which

is displayed in the timestamp section.



Lastly we can see that we can manipulate how often these calculations are taking place in order to update the database with a new value every so often. In the "scheduling" section we can set it to have this feature update every time that a tag changes its value or more conveniently for our project, we set it so that it would make calculations every minute and then spit back value into the database.

Client Application

The client application enables wireless connectivity between the EWOs, the Web Interface, and the OSI PI Database. In doing so, the client application serves three main purposes: receiving data from the EWOs, relaying commands from the web interface, and updating the database with relevant, converted values.

Communication

The primary function of the client application is receiving data from the EWOs. In order to simultaneously send and receive data to and from multiple sources an asynchronous server was deemed necessary for the project. Simple TCP read/write functions had the potential to lock

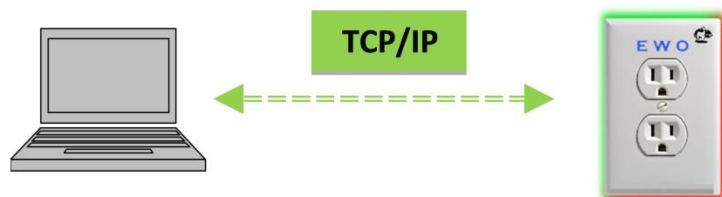
thread at inconvenient times in the client application which caused incoming connections to be ignored and massive amounts of retransmissions. Asynchronous servers are based on the principle of Event Handlers which act as interrupts when there is data to be sent or received. This enables the client application to continuously listen to the network for incoming connections while processing data and sending commands. Code fragments for the send and receive commands are shown below.

```
// Create the state object.
            StateObject state = new StateObject();
            state.workSocket = handler;
            handler.BeginReceive(state.buffer, 0, StateObject.BufferSize, 0, new
    AsyncCallback(ReadCallback), state);

// Begin sending the data to the remote device.
            handler.BeginSend(byteData, 0, byteData.Length, 0, new
    AsyncCallback(SendCallback), handler);
```

The server code is based on the MSDN Asynchronous Server which can be found in the MSDN repository.

Switching

The second task of the client application is to relay the switching commands from the web interface in a timely manner. Because the time between EWO data transmissions can be quite long (5 minutes+), storing socket data for each outlet is not an option as the sockets are likely to automatically close during an extended downtime. Thus the asynchronous server is unable to relay commands from the web interface to the EWOs without creating a new socket to a static IP for each request. To circumvent potentially locking the server during these transmissions, a simple one way TCP send class is called to send a single packet to a EWO based on its IP when a switching command from the web interface is received. The code for the TCP send class is shown below as well as the data



packet for switching outlets. Note: if the EWOs themselves are on a network connected to the internet, switching can be performed directly from the web interface.

```
public void sendTCP(string IP, int PORT, string message)
        {
            try
            {
                TcpClient connection = new TcpClient(IP, PORT);     //Create new
connection to IP and PORT
                NetworkStream stream = connection.GetStream();      //Create a stream to
pass the message
                StreamWriter sw = new StreamWriter(stream);         //Create a steam
writer to write to

                sw.Write(message);      //Write the message to the writer
                Thread.Sleep(200);
                sw.Close();             //Close the stream writer
                stream.Close();         //Close the stream
                connection.Close();     //Close the connection
            }
```

Parsing

The third primary function of the client application involves the parsing, converting, and uploading of received data. Data received in packet form from the EWOs needs to be parsed and converted before being uploaded to the OSI-PI Database. Values received are still in the 24 bit representation given by the energy measuring hardware and must be converted into Voltage, Current, or Power Factor depending on the data header.

The first step in processing incoming data is parsing. The data is split via string splitting operations shown below, first into individual values and then into header/datum representation.

```
string[] com = message.Split(',');      //split string on ',' chars into a string array

            foreach (string s in com)                   //for each string in array split
again on '='
            {
                string[] eq = s.Split('=');
                if (eq.Length > 1)                     //if successful split
                    try
                    {
                        parseVal(eq[0], eq[1]);        //call parse function
                        //Console.WriteLine(" In Parser {0}",eq[0]);
                        Validate();
                    }
```

The header and datum are then stored in a char array and analyzed by a switch statement.

Depending on the header, the proper converting formula is applied to the data value, the value is converted into proper representation for the database and then uploaded. The process for uploading a "Voltage" value is shown below. "F13_APA_Voltage" is the data array in the OSI-PI database where voltage values for a particular outlet are stored.

```
case "V":                         //if voltage
                    string vtemp = voltage + outlet.ToString();
                    try
                    {
                        myPI.connect_Server("ESMARTSERVER-PC");

                        if (myPI.check_connection())                        {
                            //Setting the value of the point
                            myPI.setPiPointValue(vtemp,
((((float.Parse(value))*120)/Math.Pow(2,23))/.6));
                        }
                    }
```

Conclusion

Team ALPACA was tasked with developing the building blocks of a home energy monitoring system, through the development of smart outlets and a responsive user interface. Through hard work and determination, the design came together and from it came the EWO. The graphical user interface is well designed with neatly organized graphs showing appropriate data and easy-to-use controls. The data is pulled from a database that is secure and powerful enough to handle the large amounts of data and calculations. And the data originates from a hardware product that is built with a robust design able to handle any standard load connected to it and is able to send the data with the use of a Wi-Fi network. The hardware product was designed to be retrofittable so that it can be adapted to any type of socket layout.

Even with the final product we have designed, there is always room for improvements. What we have developed is a great baseline product that can be upgraded with a few refinements and add-ons. One aspect that could be refined is the Wi-Fi module. Our design uses an XBee through-hole module that takes up a lot of space over a surface-mounted one. It would be nice to have a superior on-board Wi-Fi module to conserve space and have a more original design. With more time spent on making the PCB layout more efficient would allow for the overall size of the product to reduce allowing the produce to be retrofitted into more areas. Also the PCBs should have been sent off to be professionally made so that the product has the copper covered to prevent and arcing and possible injury. The professional PCBs would also have had thicker copper layers to further prevent any high currents from damaging the boards and a silkscreen layer to make assembly easier.

Finally, after researching into standards and certifications, it would have been nice to have the EWO certified under UL listing so that it can be deemed safe for the end user. With all of these aspects put together, the end result is a product that is very intuitive and easy to use while being robust and accurate.

References

1. *Asynchronous Server Socket Example*. Retrieved from
http://msdn.microsoft.com/en-us/library/fx6588te(v=vs.110).aspx

2. Cirrus Logic, "Three Channel Energy Measurement IC," CS5480 datasheet, Apr. 2012 [Revised Mar. 2013].

3. Digi International, "XBee Wi-Fi RF Module," XB2B-WFST-001 datasheet, Aug. 2013.

4. Microsoft Developer Network. C# Reference (n.d.) Retrieved December 17, 2013, from
http://msdn.microsoft.com/en-us/library/618ayhy6.aspx

5. PHP Manual.(n.d.) Retrieved December 17, 2013, from http://www.php.net/manual/en/index.php

6.. Recom. "3 Watt Single Output," RAC03-05SC/277 datasheet, Jan. 2013.

7. Texas Instruments, "MSP43055xx Family: User's Guide," MSP430F5529 datasheet, May. 2013
[Revised Jul. 2013].

8. Schrack, "Power PCB Relay RT1 Bistable," General Purpose Relays PCB Relays, Jun. 2013.

9. "Sourceforge.net". Apps.sourceforge.net

10. Tizag. Javascript Tutorial. (n.d.) Retrieved December 17, 2013, from
http://www.tizag.com/javascriptT/

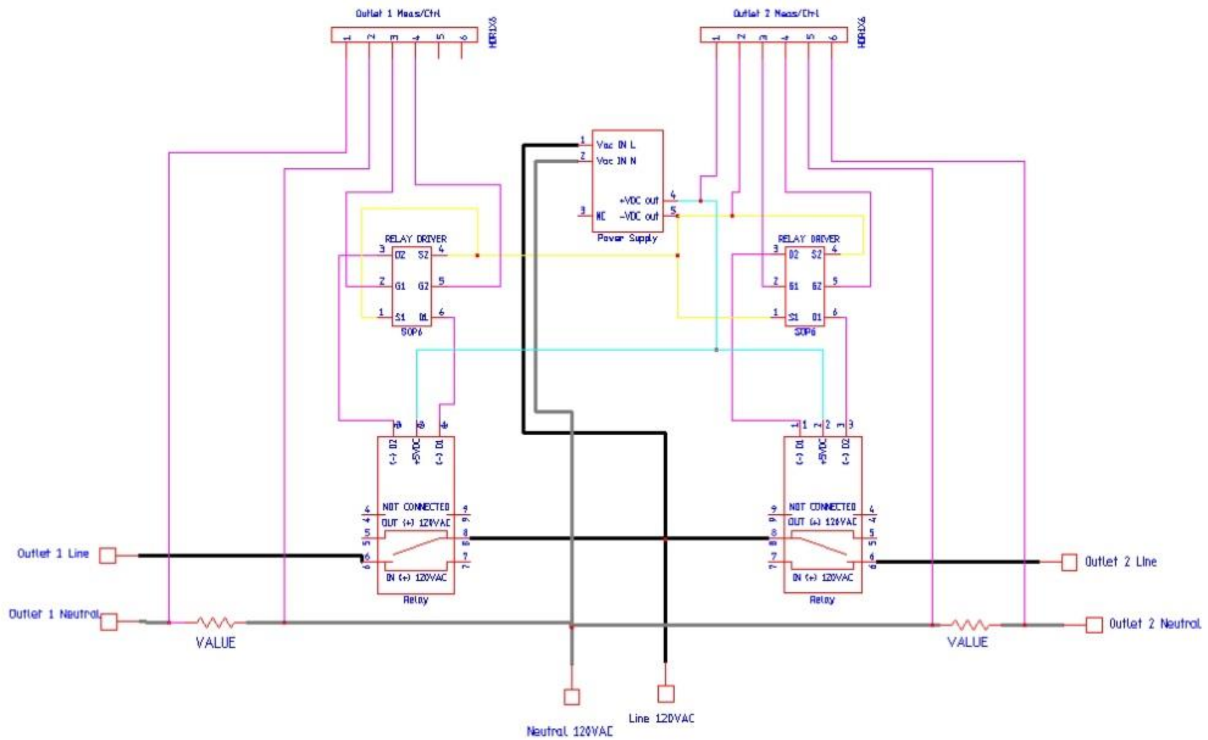
11. Tizag. Javascript Tutorial. (n.d.) Retrieved December 17, 2013, from
http://www.tizag.com/xmlTutorial/

12. W3Schools. HTML 5 Introduction. (n.d.) Retrieved December 17, 2013, from
http://www.w3schools.com/html/html5_intro.asp

13. W3Schools. CSS Tutorial (n.d.) Retrieved December 17, 2013, from http://www.w3schools.com/css/

Appendices

PCB Schematic



High Voltage

EWO Embedded Code

Below is the function that is a generic function that reads register values in the CS5480 that are signed values:

```
/**************************************************************************
 *
 *          FUNCTION:                   CS5480_Read_Signed
 *
 *          DESCRIPTION:    This function sends 2 bytes to CS5480 and reads three
 *                                      signed bytes returned by CS5480
 *
 *          ARGUMENTS:                  uint8_t regPage, uint8_t regAdd
 *
 *          RETURN:                                 int32_t
 *
 *          NOTES:                  regPage is the register page to access and regAdd is the
 *                                   register address to read from CS5480
 *
 **************************************************************************/
int32_t CS5480_Read_Signed(uint8_t regPage, uint8_t regAdd)
{
```

```
	volatile int8_t hi=0,mid=0,lo=0,dummy=0; // declare temp variables
	CS5480_CS_LOW(); //Start SPI transmission
	CS5480_SendByte(regPage); //Send register page
	CS5480_SendByte(regAdd);  //Send register address
	hi	= CS5480_GetRegValue_Signed(regAdd);//read hi byte
	mid	= CS5480_GetRegValue_Signed(NULL); //read mid byte
	lo	= CS5480_GetRegValue_Signed(NULL); //read lo byte
	dummy	= CS5480_GetRegValue_Signed(NULL); //dummy byte do nothing
	CS5480_CS_HIGH();						//stop SPI transmission
	return Make32(hi,mid,lo); //return composite 32bit integer
```

This function is used to read the voltage register of the CS5480 and stores the values in the MSP430:

```
/*********************************************************************
 *
 *	FUNCTION:		CS5480_GetVoltage
 *
 *	DESCRIPTION:	This function read voltage register in CS5480
 *				and stores the value in a structure
 *	ARGUMENTS:		None
 *
 *	RETURN:			None
 *
 *	NOTES:			This is used for the MCU to point to the most recent
 *				conversion and use that in the message sent to server
 *
 *********************************************************************/
void CS5480_GetVoltage(void)
{
	volatile uint32_t vOut=0;
	CS5480_CS_LOW();				//BEGIN SPI STREAM
							//READ VRMS REGISTER, PAGE 16
	vOut = CS5480_Read_Unsigned(SW_REG_PAGE16,VRMS_1);
	CS5480_CS_HIGH();				//END CURRENT SPI STREAM
	pvRmsOut->hiByte  = (vOut>>16);		//STORE HI BYTE IN VOLT STRUCT
	pvRmsOut->midByte = (vOut>>8);		//STORE MID BYTE IN VOLT STRUCT
	pvRmsOut->loByte  = (vOut);			//STORE LO BYTE IN VOLT STRUCT
	vOutTemp = vOut;			//TEMP VARIABLE FOR CALIBRATION CHECK

}
```

This function is used to send a pulse to toggle a relay off:

```
/*********************************************************************
 *
 *	FUNCTION:		TopRelayToggleOff
 *
 *	DESCRIPTION:	This function sends a pulse to toggle top socket relay off
 *
 *	ARGUMENTS:		None
 *
 *	RETURN:			None
 *
 *	NOTES:			A half second delay is inserted between pulses to prevent
 *				current sinking issues that trip the MCU.
 *
 *********************************************************************/
void TopRelayToggleOff(void)
{
	 //FUNCTION SENDS A QUICK PULSE TO
	 //ENERGIZE ONE OF THE TWO COILS
```

```
            //IN THE LATCHING RELAYS

        //SET P 6.0 OUTPUT HIGH
         GPIO_setOutputHighOnPin(
                GPIO_PORT_P6,
                GPIO_PIN0
                );
                    __delay_cycles(600000);
        //SET P 6.0 OUTPUT LOW
                    GPIO_setOutputLowOnPin(
                GPIO_PORT_P6,
                GPIO_PIN0
                );
}
```

Client Application Code

Data management code:

```
namespace AlpacaFinal
{
    public partial class DataMGT : Form
    {
        public class PointVal                    //class to store date/time for updating PI DB
        {
            public string date;
            public string value;
        }
        public PythonComm PythonComm;            //instantiate TCP basic send class
        TcpClient connect = new TcpClient();     //instantiate asynchronous server

        OSIPI myPI = new OSIPI();                //new PI connection
        string get = "teststring";
        //string get2 = "change";
        int count = 0;
        int oldLength;
        int outlet = 1; //store outlet # as well as text for updating PI
        int stream = 0;
        string EWO2 = "192.168.144.139";
        string EWO1 = "192.168.144.147";

        public DataMGT()
        {
            PythonComm = new PythonComm();    //initialize tcp/ip control
            InitializeComponent();            //load form
        }

        public void gotString(string message)
        {

            string[] com = message.Split(',');      //split string on ',' chars into a string array

            foreach (string s in com)                 //for each string in array split again on '='
            {
                string[] eq = s.Split('=');
                if (eq.Length > 1)                    //if successful split
                    try
                    {
                        parseVal(eq[0], eq[1]);       //call parse function
                        //Console.WriteLine(" In Parser {0}",eq[0]);
                        Validate();
```

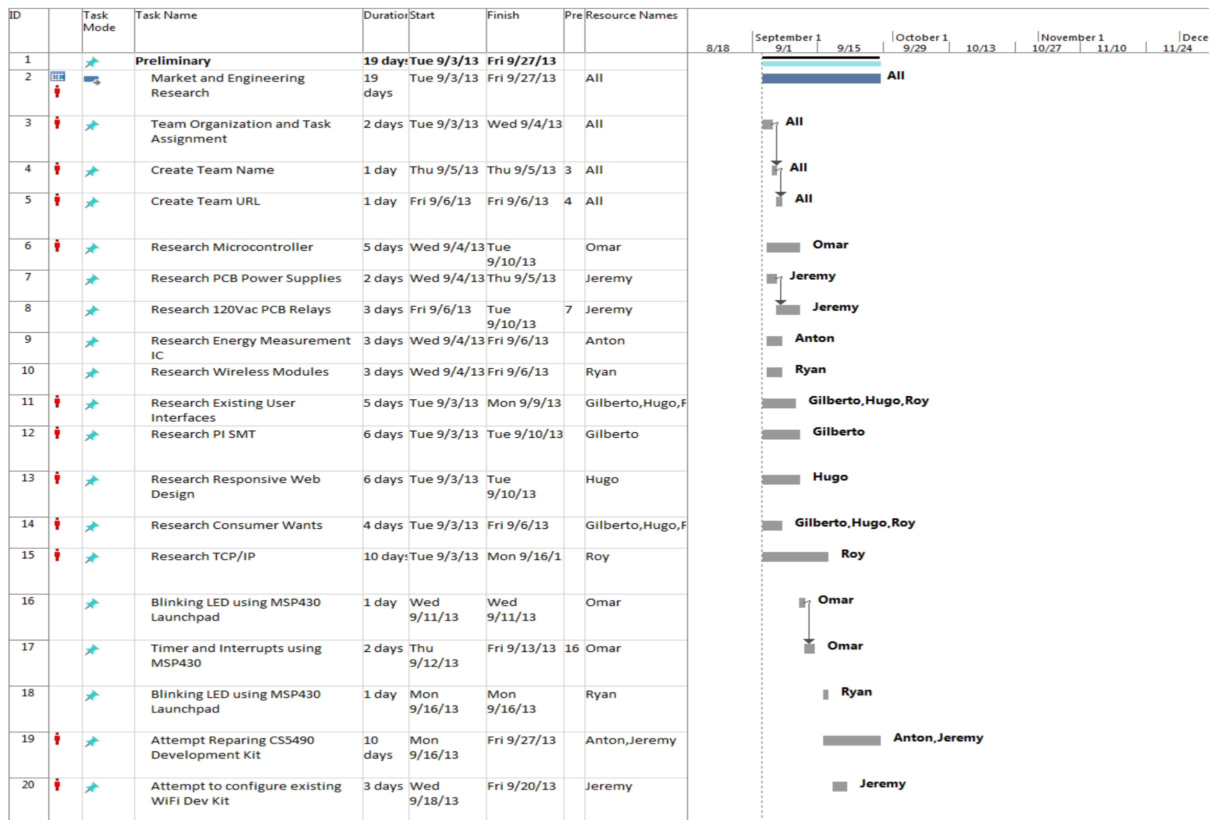
```
            }
            catch (Exception e)
            {
                Console.WriteLine(" {0} Can not parse value", e);
            }

        }
    }
```

Full Parsing Function:

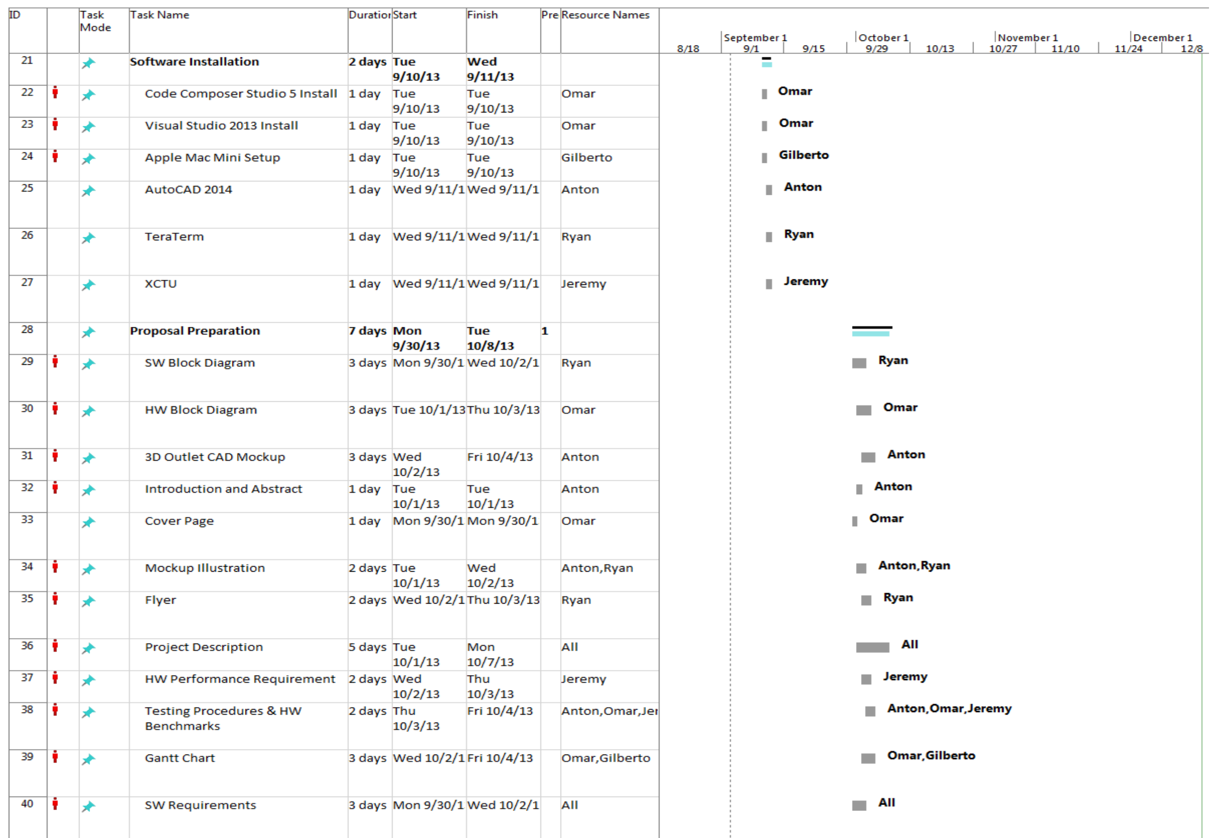
```
public void parseVal(string letter, string value)
    {


        string voltage = "F13APA_Voltage";
        string currenttop = "F13APA_CurrentTOP";
        string currentbot = "F13APA_CurrentBOTTOM";
        string powertop = "F13APA_Power_Top";
        string powerbot = "F13APA_Power_Bot";
        switch (letter)                    //checks first value in string for command
        {
            case "V":                      //if voltage
                string vtemp = voltage + outlet.ToString();
                try
                {
                    myPI.connect_Server("ESMARTSERVER-PC");

                    if (myPI.check_connection())
                    {
                        //Setting the value of the point
                        myPI.setPiPointValue(vtemp, ((((float.Parse(value))*120)/Math.Pow(2,23))/.6));
                    }
                }
                catch (Exception ex)
                {
                    Console.WriteLine(ex.Message);
                }
                break;
            case "IT":                     //if current TOP outlet
                string currenttoptemp = currenttop + outlet.ToString();
                try
                {
                    myPI.connect_Server("ESMARTSERVER-PC");

                    if (myPI.check_connection())
                    {
                        //Setting the value of the point
                        myPI.setPiPointValue(currenttoptemp, ((((float.Parse(value))*15)/(Math.Pow(2,24)-1))/.6));
                    }

                }
                catch (Exception ex)
                {
                    Console.WriteLine(ex.Message);
                }
                break;
            case "IB":                     //if current BOT outlet
                string currentbottemp = currentbot + outlet.ToString();
                try
                {
                    myPI.connect_Server("ESMARTSERVER-PC");
```

```
        if (myPI.check_connection())
        {
            //Setting the value of the point
            myPI.setPiPointValue(currentbottemp, ((((float.Parse(value)) * 15) / (Math.Pow(2, 24) - 1)) / .6));
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
    break;
case "PT":                          //POWER TOP
    string powertoptemp = powertop + outlet.ToString();
    try
    {
        myPI.connect_Server("ESMARTSERVER-PC");

        if (myPI.check_connection())
        {
            //Setting the value of the point
            myPI.setPiPointValue(powertoptemp, 1-((float.Parse(value))/(Math.Pow(2,23)-1)));
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
    break;
case "PB":                              //POWER BOT
    string powerbottemp = powerbot + outlet.ToString();
    try
    {
        myPI.connect_Server("ESMARTSERVER-PC");

        if (myPI.check_connection())
        {
            //Setting the value of the point
            myPI.setPiPointValue(powerbottemp, 1-((float.Parse(value)) / (Math.Pow(2, 23) - 1)));
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
    break;
```

Project Schedule

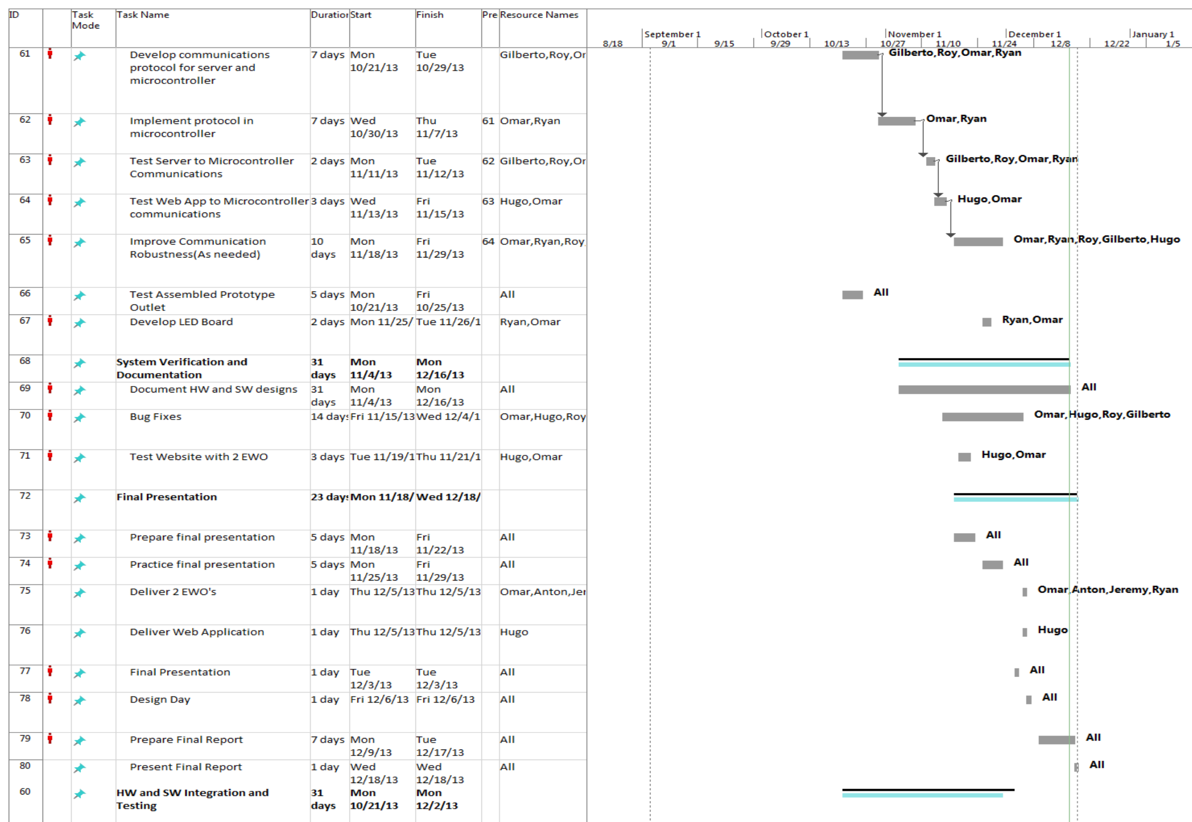


ID	Task Mode	Task Name	Duration	Start	Finish	Pre	Resource Names
1		**Preliminary**	**19 days**	**Tue 9/3/13**	**Fri 9/27/13**		
2		Market and Engineering Research	19 days	Tue 9/3/13	Fri 9/27/13		All
3		Team Organization and Task Assignment	2 days	Tue 9/3/13	Wed 9/4/13		All
4		Create Team Name	1 day	Thu 9/5/13	Thu 9/5/13	3	All
5		Create Team URL	1 day	Fri 9/6/13	Fri 9/6/13	4	All
6		Research Microcontroller	5 days	Wed 9/4/13	Tue 9/10/13		Omar
7		Research PCB Power Supplies	2 days	Wed 9/4/13	Thu 9/5/13		Jeremy
8		Research 120Vac PCB Relays	3 days	Fri 9/6/13	Tue 9/10/13	7	Jeremy
9		Research Energy Measurement IC	3 days	Wed 9/4/13	Fri 9/6/13		Anton
10		Research Wireless Modules	3 days	Wed 9/4/13	Fri 9/6/13		Ryan
11		Research Existing User Interfaces	5 days	Tue 9/3/13	Mon 9/9/13		Gilberto,Hugo,F
12		Research PI SMT	6 days	Tue 9/3/13	Tue 9/10/13		Gilberto
13		Research Responsive Web Design	6 days	Tue 9/3/13	Tue 9/10/13		Hugo
14		Research Consumer Wants	4 days	Tue 9/3/13	Fri 9/6/13		Gilberto,Hugo,F
15		Research TCP/IP	10 days	Tue 9/3/13	Mon 9/16/1		Roy
16		Blinking LED using MSP430 Launchpad	1 day	Wed 9/11/13	Wed 9/11/13		Omar
17		Timer and Interrupts using MSP430	2 days	Thu 9/12/13	Fri 9/13/13	16	Omar
18		Blinking LED using MSP430 Launchpad	1 day	Mon 9/16/13	Mon 9/16/13		Ryan
19		Attempt Reparing CS5490 Development Kit	10 days	Mon 9/16/13	Fri 9/27/13		Anton,Jeremy
20		Attempt to configure existing WiFi Dev Kit	3 days	Wed 9/18/13	Fri 9/20/13		Jeremy



ID	Task Mode	Task Name	Duration	Start	Finish	Pre	Resource Names
21		**Software Installation**	**2 days**	**Tue 9/10/13**	**Wed 9/11/13**		
22		Code Composer Studio 5 Install	1 day	Tue 9/10/13	Tue 9/10/13		Omar
23		Visual Studio 2013 Install	1 day	Tue 9/10/13	Tue 9/10/13		Omar
24		Apple Mac Mini Setup	1 day	Tue 9/10/13	Tue 9/10/13		Gilberto
25		AutoCAD 2014	1 day	Wed 9/11/1	Wed 9/11/1		Anton
26		TeraTerm	1 day	Wed 9/11/1	Wed 9/11/1		Ryan
27		XCTU	1 day	Wed 9/11/1	Wed 9/11/1		Jeremy
28		**Proposal Preparation**	**7 days**	**Mon 9/30/13**	**Tue 10/8/13**	**1**	
29		SW Block Diagram	3 days	Mon 9/30/1	Wed 10/2/1		Ryan
30		HW Block Diagram	3 days	Tue 10/1/13	Thu 10/3/13		Omar
31		3D Outlet CAD Mockup	3 days	Wed 10/2/13	Fri 10/4/13		Anton
32		Introduction and Abstract	1 day	Tue 10/1/13	Tue 10/1/13		Anton
33		Cover Page	1 day	Mon 9/30/1	Mon 9/30/1		Omar
34		Mockup Illustration	2 days	Tue 10/1/13	Wed 10/2/13		Anton,Ryan
35		Flyer	2 days	Wed 10/2/1	Thu 10/3/13		Ryan
36		Project Description	5 days	Tue 10/1/13	Mon 10/7/13		All
37		HW Performance Requirement	2 days	Wed 10/2/13	Thu 10/3/13		Jeremy
38		Testing Procedures & HW Benchmarks	2 days	Thu 10/3/13	Fri 10/4/13		Anton,Omar,Jer
39		Gantt Chart	3 days	Wed 10/2/1	Fri 10/4/13		Omar,Gilberto
40		SW Requirements	3 days	Mon 9/30/1	Wed 10/2/1		All

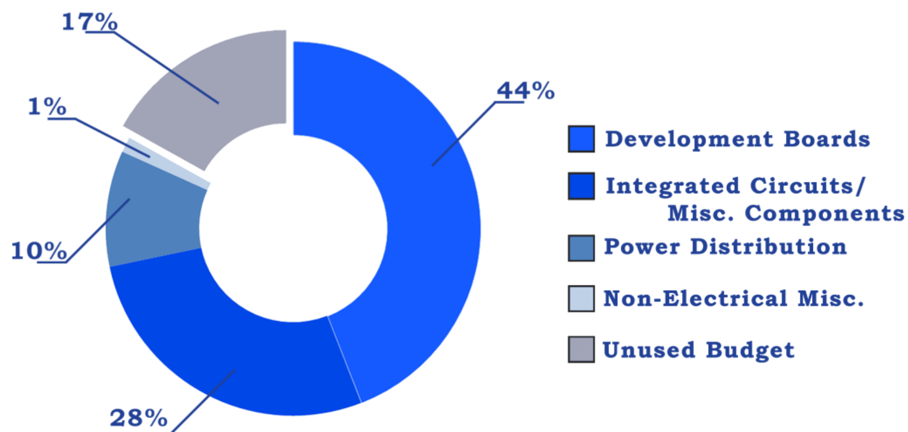
ID	Task Mode	Task Name	Duration	Start	Finish	Pre	Resource Names
41		Editing	2 days	Fri 10/4/13	Mon 10/7/1		Roy,Ryan
42		Cost Analysis	2 days	Tue 10/1/13	Wed 10/2/1		Jeremy
43		**Hardware Design**	**31 day:**	**Tue 10/8/13**	**Tue 11/19/1**	**28**	
44		Breadboard relay circuitry	3 days	Tue 10/8/13	Thu 10/10/13		Anton,Jeremy
45		Design software simulator to test relays using MSP430	2 days	Tue 10/8/13	Wed 10/9/13		Omar
46		Design Wifi module to microcontroller interface	12 days	Thu 10/10/13	Fri 10/25/13		Ryan,Omar
47		Interface with CS5480 IC via SPI	14 days	Mon 10/7/13	Thu 10/24/13		Omar
48		Design High Voltage Board PCB layout	10 days	Fri 10/11/13	Thu 10/24/13		Anton,Jeremy
49		Design Low Voltage Board PCB Layout	10 days	Mon 10/14/13	Fri 10/25/13		Anton,Jeremy,F
50		Print High/Low Voltage	5 days	Mon 10/28/13	Fri 11/1/13	49,	Anton,Jeremy
51		Integrate Components and Assemble prototype outlet	3 days	Mon 11/4/13	Wed 11/6/13	50	Anton,Jeremy
52		Test/Debug Assembled EWO	5 days	Thu 11/7/13	Wed 11/13/	51	Anton,Jeremy,C
53		**Software Design**	**31 day:**	**Tue 10/8/13**	**Tue 11/19/1**		
54		Design Server Framework	31 days	Tue 10/8/13	Tue 11/19/13		Gilberto
55		Design Server-Outlet Communications Protocol	21 days	Tue 10/8/13	Tue 11/5/13		Roy
56		Create C# Classes to Connect to PI SMT	5 days	Tue 10/8/13	Mon 10/14/13		Gilberto
57		Create Attributes for Website	11 days	Tue 10/8/13	Tue 10/22/13		Hugo
58		Create Website Template	6 days	Tue 10/8/13	Tue 10/15/13		Hugo,Roy
59		Develop Javascript code for web:	12 day:	Mon 11/4/1	Tue 11/19/1		Hugo
60		**HW and SW Integration and Testing**	**31 days**	**Mon 10/21/13**	**Mon 12/2/13**		



ID	Task Mode	Task Name	Duration	Start	Finish	Pre	Resource Names
61		Develop communications protocol for server and microcontroller	7 days	Mon 10/21/13	Tue 10/29/13		Gilberto,Roy,Or
62		Implement protocol in microcontroller	7 days	Wed 10/30/13	Thu 11/7/13	61	Omar,Ryan
63		Test Server to Microcontroller Communications	2 days	Mon 11/11/13	Tue 11/12/13	62	Gilberto,Roy,Or
64		Test Web App to Microcontroller communications	3 days	Wed 11/13/13	Fri 11/15/13	63	Hugo,Omar
65		Improve Communication Robustness(As needed)	10 days	Mon 11/18/13	Fri 11/29/13	64	Omar,Ryan,Roy
66		Test Assembled Prototype Outlet	5 days	Mon 10/21/13	Fri 10/25/13		All
67		Develop LED Board	2 days	Mon 11/25/	Tue 11/26/1		Ryan,Omar
68		**System Verification and Documentation**	**31 days**	**Mon 11/4/13**	**Mon 12/16/13**		
69		Document HW and SW designs	31 days	Mon 11/4/13	Mon 12/16/13		All
70		Bug Fixes	14 day:	Fri 11/15/13	Wed 12/4/1		Omar,Hugo,Roy
71		Test Website with 2 EWO	3 days	Tue 11/19/1	Thu 11/21/1		Hugo,Omar
72		**Final Presentation**	**23 day:**	**Mon 11/18/**	**Wed 12/18/**		
73		Prepare final presentation	5 days	Mon 11/18/13	Fri 11/22/13		All
74		Practice final presentation	5 days	Mon 11/25/13	Fri 11/29/13		All
75		Deliver 2 EWO's	1 day	Thu 12/5/13	Thu 12/5/13		Omar,Anton,Jer
76		Deliver Web Application	1 day	Thu 12/5/13	Thu 12/5/13		Hugo
77		Final Presentation	1 day	Tue 12/3/13	Tue 12/3/13		All
78		Design Day	1 day	Fri 12/6/13	Fri 12/6/13		All
79		Prepare Final Report	7 days	Mon 12/9/13	Tue 12/17/13		All
80		Present Final Report	1 day	Wed 12/18/13	Wed 12/18/13		All
60		**HW and SW Integration and Testing**	**31 days**	**Mon 10/21/13**	**Mon 12/2/13**		

Bill of Materials

Parts for One Complete Product		
High Voltage Board		
Part Number	Description	Quantity
RT315F05	RELAY GEN PURPOSE SPDT 16A 5V	2
RAC03-SC_277	AC/DC CONV 3W 85-305VIN 5VOUT	1
CRF2512-FX-R010ELFCT-ND	RES 0.01 OHM 2W 1% 2512 SMD	2
P422KFCT-ND	RES 422K OHM 1/4W 1% 1206 SMD	4
	6pin Male Header 0.1" spacing	2
	RES 1K OHM 1206 SMD	1
Low Voltage Board		
Part Number	Description	Quantity
296-25427-ND	IC MCU 16BIT 128KB FLASH 80LQFP	1
598-1929-5-ND	IC ENERGY METER 3-CH 24QFN	1
NUD3105DMT1GOSCT-ND	IC INDCT LOAD DRVR DUAL SC74-6	2
PRT-08272	2mm 10pin Xbee Socket	2
1903CK-ND	HEX STANDOFF 6-32 NYLON 1/2"	2
602-1377-ND	XBEE WI-FI WIRE ANTENNA TH	1
GS1086CST	5VDC to 3.3VDC regulator	1
X1089-ND	CRYSTAL 4.096MHZ 20PF THRU	1
CSTCR4M00G15L99-R0	Resonators 4MHZ	1
AB26TRQ-32.768kHz-T	Crystals 32.768kHz +/-20ppm 12.5pF -40C +85C	1
S7004-ND	CONN HEADER FEMALE 6POS .1" TIN	2
	CAP 0.1 uF 1206 SMD	6
	CAP 27 nF 1206 SMD	6
	CAP 10 uF 1206 SMD	4
	CAP 1 uF 1206 SMD	2
	CAP 220nF 1206 SMD	2
	CAP 12 pF 1206 SMD	2
	CAP 470 nF 1206 SMD	1
	CAP 8.2 pF 1206 SMD	1
	RES 1K OHM 1206 SMD	5
	RES 10K OHM 1206 SMD	1
	RES 47K OHM 1206 SMD	1



BUDGET: $1200