Household Outlet Monitoring and Control System

By:

Michael Young

Computer Engineering Department,
California Polytechnic State University,
San Luis Obispo
Spring 2010
Table of Contents

ACKNOWLEDGEMENTS ........................................................................................................ 5

INTRODUCTION .................................................................................................................. 6

SYSTEM REQUIREMENTS ................................................................................................. 8

SYSTEM SPECIFICATIONS ............................................................................................ 9
  User Interface .................................................................................................................... 9
  Keyboard ........................................................................................................................ 9
  Automation System ......................................................................................................... 9
  Controlling Power .......................................................................................................... 10
  Digital Clock .................................................................................................................. 10

SYSTEM ARCHITECTURE .............................................................................................. 11

COMPONENT DESIGN ................................................................................................. 13
  Hardware ......................................................................................................................... 13
    Nexys Development Board ............................................................................................ 13
    LCD Screen .................................................................................................................. 14
    Keyboard ...................................................................................................................... 14
    Device Loads ................................................................................................................ 15
    16-Channel Analog Multiplexer ................................................................................... 15
    Current Sensors ............................................................................................................ 16
    Analog Mux Control Bits .............................................................................................. 17
    Analog-Digital Converter ............................................................................................. 17
  Software .......................................................................................................................... 17
    Timer/Counter ............................................................................................................... 17
    Digital Clock ................................................................................................................ 18
    LCD Software .............................................................................................................. 19
    Keyboard Software .................................................................................................... 20
    Menu Navigation .......................................................................................................... 21
    Automation System ...................................................................................................... 21
    Switch Components .................................................................................................... 22
    Power Measurements ................................................................................................. 23

TESTING .............................................................................................................................. 25

ROADBLOCKS AND PROBLEMS / SOLUTIONS ............................................................ 28

RESULTS .......................................................................................................................... 30
List of Figures

Figure 1 System Architecture ................................................................. 11
Figure 2 Load Schematic ........................................................................ 12
Figure 3 Nexys Development Board ......................................................... 13
Figure 4 User Interface of System ............................................................. 14
Figure 5 System Setup with One Load Integration .................................. 15
Figure 6 CD74HC4067 Analog Multiplexer ............................................. 15
Figure 7 ACS712 Current Sensor .............................................................. 16
Figure 8 Linear Relationship Between $I_P$ and $V_{OUT}$ .......................... 16
Figure 9 PmodAD1 Analog to Digital Converter ...................................... 17
Figure 10 Flow of Operation of Digital Clock ......................................... 19
Figure 11 Flow of Operation of LCD ......................................................... 20
Figure 12 Flow of Operation for Menu Navigation .................................. 22
Figure 13 Pinout of CD74HC4067 Analog Multiplexer ............................ 65
Figure 14 Pinout of ACS712 Current Sensor .......................................... 65
Acknowledgements

I would like to give my thanks to my friend, Kevin Ersoy, who helped develop the idea for this project. Also my friends and colleagues who helped me with any problems and difficulties I had along the way, especially Manny Tarog. Finally, I would like to thank my advisor, Professor Chris Lupo, for the guidance he has given during the period of work for this project.
Introduction

Energy is being used in every household without the owner’s knowing. Many major appliances are designed with user convenience in mind by creating a standby mode. The main purpose of a standby mode is to allow the electrical appliance to receive a signal from an external source. For example, a television will have standby mode to receive a signal from a remote control. The same idea goes for similar systems such as a DVD player and stereo receiver. Another reason for standby mode is to retain the user customized settings such as picture settings of a television. This kind of energy is sometimes referred to as “vampire energy” and can mean significant amounts of wasted energy. Not only does this waste money for many household owners, but it also accounts for 1% of CO2 emissions (Meier).

To prevent wasted energy caused by standby mode, it is obvious that users of the device should simply turn off the device from the wall when it is not in use. However, this could be difficult with a significant amount of family members, especially having younger kids in the house that might not be as responsible with all the gadgets they use today. Another way of saving power is to purchase appliances with low power consumptions ratings, which is usually about 1 watt (“Home Energy Usage Tips #1 – Standby Power”).

To give an idea of how standby power affects the world, here are some interesting facts provided by the Student Sustainability Education Coordinators at Berkeley:

- 6% of the entire residential electricity consumption is from standby power.
- California would not experience rolling blackouts if “vampire energy” were eliminated.
• Americans consume 26% of the world’s energy despite being 5% of the world’s population.

It would be ideal if the user were able to control the power usage of these common electronic devices. Although there are government compliances for companies to follow, this project will explore a different approach and allow household owners to monitor and control their energy usage. This report explains a simple simulation of this system and explores a theoretical view of how this system can be used in a household.
System Requirements

The user will be welcomed to the system by a start screen, which will include the current time and instructions to press enter to activate the menu. There will be a series of submenus for the user to perform desired tasks such as controlling power outlets, setting up the automation system for certain devices at user defined on and off times, and viewing current power consumption. Since this is a simulation of turning off power outlets in a household, we will have two rooms to control two devices in each room. The user should be able to control this system by keyboard and the system should respond accordingly. To visually show the user that devices are being turned on/off, four LEDs will be used to represent each device.
System Specifications

User Interface

The user is presented with a menu on the LCD component of the Nexys board. The user is presented with a start screen, which shows the current time of day and instructs them to press enter to continue system operation. The menu should allow them to control power of two rooms, each with two components, setup an automation system to turn off/on devices at user inputted designs, and display the current power being used by each room and each component. To visually display each device, an LED will be used to show if the device is on or off.

Keyboard

A keyboard will be used for the user to interact with the menu and is very simple. This includes navigation by the up/down arrows, selecting submenus by numbers, setting up the automation system, and turning on/off devices. Incorrect input should be ignored. For example, if the user selects a submenu that does not exist, then the system will ignore the input and do nothing.

Automation System

The automation system should give the ability to set up on and off times for the devices. The user will input in a “12:00:00 am (or pm)” format. The seconds should not be specified since most timing devices do not request it from the user. If an invalid time is inputted, the user will be presented with a “Time invalid” message and the on and off times will not be saved for the device. At this point, the user must redo this operation for
their desired on and off times. When the on time of a device occurs, the representative LED of the device will switch on. Likewise, when the off time of a device occurs, the representative LED of the device will switch off. If the device is already on, and the on time occurs, the device remains on. This same operation happens if the off time occurs when the device is already off.

**Controlling Power**

The user should be able to turn on and off each device. Each room should indicate the power being used when they are presently in that submenu. If the user selects a room, each device should indicate the power being used. At this point the user can select which device to turn on/off by selecting ‘ON (Y)’ or ‘OFF (N)’ via keyboard input. The appropriate LED should be turned on/off.

**Digital Clock**

The digital clock is displayed on the start screen. The automation system utilizes the digital clock to compare on/off times of devices after each minute.
System Architecture

The hardware includes the MicroBlaze soft-core processor and all the bus routing. The BRAM (Block RAM) is where all the instructions are stored and accessed. The ILMB (instruction local memory bus) and the DLMB (data local memory bus) control the accessing of data and instructions located in the BRAM. The GPIO (general-purpose I/O) modules are used in conjunction with the OPB (on chip peripheral bus) to control the peripheral devices. The following components utilize GPIOs to operate:

- LCD Control and Data
- Nexys on-board LEDs
- Device loads
- Analog Multiplexer control bits

Figure 1 System Architecture
The timer/counter is another main component of the system. The timer/counter requires a clock input and will generate an interrupt output. The timer/counter output is connected to the MicroBlaze interrupt and should generate an interrupt every second. To communicate with the keyboard, the UART Lite IP core will be used to transfer data from the PS/2 component to the Nexys board. Similarly, an SPI IP core will be used to communicate with the Digilent PmodAD1 to obtain digital signals of voltage readings.

Figure 2 Load Schematic
Component Design

**Hardware**

*Nexys Development Board*

The Nexys development board’s Spartan 3-200 microcontroller will be used as the primary hardware component of the system. The LCD screen will be connected to the 16-pin header (J8 port) on the Nexys board. The on-board LEDs will be used for testing purposes, more specifically, confirming software design and correct device selection. The 6-pin headers (JA-JD) will all be utilized in this design. The keyboard will be used for user input and will be connected to the JA header via PS/2 connector. The JB header will support voltages being supplied to the load circuitry, turning on/off LEDs to represent devices. To show power consumption readings by each load, an analog multiplexer will be used to select which loads to measure by control bits out of the JC header. The readings of each load will be received by the Digilent PmodAD1, which is connected to the JD header.

*Figure 3* Nexys Development Board
**LCD Screen**

The user interface comprises of the LCD and a keyboard. The LCD interfaces with the Nexys board by using two general-purpose I/Os (GPIO). The LCD requires one GPIO for transmitting commands to the LCD such as clearing the screen, moving the cursor, delays, etc. The other GPIO is used for transmitting data to be displayed on the LCD screen.

**Keyboard**

The main interaction with the system is through the keyboard. The PS/2 connector is used to connect the keyboard to the Nexys board. The UART Lite IP core is used to communicate between the PS/2 connector and the Nexys board so input. The UART Lite stands for Universal Asynchronous Receiver/Transmitter and is used to transfer bits from a receive FIFO or to a transmit FIFO. The UART Lite transfers 5-8 data bits per transfer. Considering scan code for keys are 8 bits in length, the UART Lite served as the perfect IP core for communication with the keyboard.

![Figure 4 User Interface of System](image)
Device Loads

The LEDs are the “loads” of the system to create a visual representation of devices of the system. This was implemented using a GPIO with a 2-bit data bus width, one bit for each load. The loads are connected to the JB 6-pin header of the Nexys board where two data pins are used for the two loads as well as the ground pin.

![Diagram of system setup with one load integration](image)

**Figure 5** System Setup with One Load Integration

16-Channel Analog Multiplexer

A 16-Channel Analog Multiplexer is used to receive the two current sensor output voltages and outputs the selected channel via control bits. Proper operation requires 5V. Although this is a 16-channel analog multiplexer, only two channels are used for the two loads.

![CD74HC4067 Analog Multiplexer](image)

**Figure 6** CD74HC4067 Analog Multiplexer
**Current Sensors**

The Hall effect based ACS712 current sensors are used for determining current going through the load and requires $5 \text{ V}_{\text{CC}}$. The sensor outputs an analog voltage that varies linearly with the current. Each load has a current sensor and outputs the voltage to an analog multiplexer. Since we are interested in power consumption, the voltages are translated into current measurements.

![ACS712 Current Sensor](image)

**Figure 7 ACS712 Current Sensor**

![Output Voltage versus Sensed Current](image)

**Figure 8 Linear Relationship Between $I_P$ and $V_{OUT}$**
**Analog Mux Control Bits**

Similar to the loads, the multiplexer control bits are implemented using a GPIO IP core. However, only a 2-bit data bus width is required. The bits are used as data selectors to read voltages of each load.

**Analog-Digital Converter**

The signal is transmitted to the PmodAD1 analog-digital converter to be used by the software. To communicate with the Nexys board, a Serial Peripheral Interface (SPI) IP core is used to for fast data transfer from the analog-digital converter.

![Figure 9 PmodAD1 Analog to Digital Converter](image)

**Software**

**Timer/Counter**

The 32-bit timer/counter was initialized in order to generate an interrupt every half second for the digital clock function. With the microprocessor running at 50MHz and requiring an interrupt every half second, it is determined that the load register of the timer/counter should be loaded with half the processing speed (25,000,000). The counter is set to count down from the load register value. The compiler needs to determine what function to jump to when an interrupt occurs. The syntax that designates the interrupt service routine (ISR) is `my_isr(__attribute__((interrupt_handler)))`. When the load register value is zero, the ISR first checks the interrupt bit of the timer/counter’s
status register to check if an interrupt has occurred. The ISR then runs and at the end, the interrupt bit is cleared in order for another interrupt to occur.

*Digital Clock*

When an interrupt occurs, the proper action is to call the function `void update_time()` to handle updating the time display when an interrupt occurs every other half second. Manually updating the time is a separate function called `void manual_update()`. The firmware determines if the proper buttons have been pushed in order to see if control should be handed off to this function. If not, then the clock will operate normally.
LCD Software

The LCD software is comprised of initialization and common LCD functions such as clearing the screen and shifting the cursor. Correct initialization of the LCD ensures that characters are displayed correctly. To initialize the LCD, a 20mS delay is required after power on before sending a function set command. The function set determines the data length, number of lines, and display font. After this command, a 37uS delay is needed before turning the display on. The next command is clearing the display after waiting
another 37uS. Finally, a 1.52mS delay is required before going into entry mode. From here, we can now write characters to the LCD. A delay is required before writing each character.

Figure 11 Flow of Operation of LCD

Keyboard Software

The keyboard is the main component for the user to interact with the system. The software is quite simple for the keyboard. The keyboard uses the UART Lite to communicate with the Nexys board. First, the software checks if there is valid data received in the receive FIFO. If so, the data receive register is checked and the data is held there until it is used to take care of debouncing. Finally, the scan code for the key
pressed is then saved in a variable for use. To translate the scan code into a character, a look up table is used.

*Menu Navigation*

Navigating through the menu is done by user input. The program should continuously wait for the user to provide a valid keystroke to navigate through the menu. At power on, the start screen consists of the digital clock and instructing the user to press enter to continue. They are then presented with the main menu with three options: Control Power, Setup, and Information. The Control Power submenu allows the user to select which room to view, Room 1 or Room 2. In each room, there are 2 devices the user is can switch on/off. The Setup submenu brings the user to the automation system where the user can designate on and off times of a device in one of the rooms. If the user selects Information, the user will be informed of the current total power consumption of the devices.

*Automation System*

The user is allowed to set on and off times for devices. When the automation system is accessed, the user selects the room the device is located in followed by the desired device to set times for. Afterwards, the user will enter a turn off time, followed by a turn on time. There is no need for the user to worry about entering ‘:’ characters for the time since they will automatically be inserted. Also, on and off times are limited by the hour and the minute values as well as the am/pm characters. Therefore seconds will always be ‘:00’. After both times are entered, the next step is testing if the times entered are valid. If one of the entered times is not valid, then the time will not be saved for the selected
device and the user must re-enter valid times. As the digital clock is running, on and off times of each device is checked at after every minute and the correct devices should be turned on/off according to device on and off times.

**Figure 12** Flow of Operation for Menu Navigation

*Switch Components*

When the user navigates through the menu and is presented with the choice to turn on/off a device, a global variable ‘device’ is updated with the ID number of the device selected. This global variable is then passed into the function `switchComp(int device, int onOff)` where ‘device’ is the load to switch off and ‘onOff’ determines whether the user requested the device to turn on (onOff = 1) or off (onOff = 0). When a device is requested to be turned on, the device ID, which is a hexadecimal number, is ORed with a global variable ‘leds’ to turn on the load.

```c
leds |= device;
```
Turning a device off involves more complicated bitwise operations. First, we must ensure that the device is not turned off already. If it is not, then we can update the global variable ‘leds.’ To do so, we take flip the bits of the device ID and XOR it with ‘leds’.

\[
\text{leds} = (\text{leds} \ ^ \ ^ \sim \text{device});
\]

After the appropriate bitwise operation, the LCD screen will display a confirmation message of what device was turned on or off. Finally, the user will be taken back to the beginning of the menu.

*Power Measurements*

The power measurements are given in kWh/day. It will appear when the user selects a device to turn on/off through the Control Power menu. When the device is selected, mux control bits are sent to the analog multiplexer to select which device to read from. The analog multiplexer then sends a signal to the analog-digital converter. The digital value will be in a range 0 to 4095 behaving linearly with 0.0V to 3.3V. The digital signal received must be compared with the output voltage of the current sensor referencing 0.0A, which is 2.5V. Converting the 2.5V by a simple calculation indicates that the digital signal value for 2.5V is roughly 3103. This value will be used to scale up the output voltage since the load circuit is being supplied 5.0V. After confirming the rate of voltage change in relation to current of the current sensor, the value scaled can then be converted into amperage. From this point, we can calculate power in watts using the following equation:

\[
P = IV
\]
The voltage (V) in this equation is the supply voltage 5.0V. Now the power will can be converted into kWh/day. To do so, the wattage is divided by 1000 to obtain kilowatts and then multiplied by 24 hours.
Testing

After a component was integrated, it was necessary to test that each component was operating properly by running simple operational tests.

The first component that was integrated was the LCD screen. After setting up the appropriate memory space and GPIO for the component, “Hello World” was sent to the screen. Next, the keyboard and PS/2 connector was implemented to communicate with the UART Lite. Before we can see proper output of the keyboard, it is necessary to obtain the correct baud rate of the keyboard for communication. Baud rates are not a specification that is readily known or listed by keyboard manufacturers. In the case of this project, the baud rate had to be measured with an oscilloscope. Using the oscilloscope, a key is depressed and the waveform is captured. The frequency of the waveform is measured and indicates the baud rate of the keyboard. Since the baud rate of most of the keyboards is not matched to the 9600 or 19200 of the UART parameters, the following equation is used:

\[ \text{OPB Clk Freq.} = (50\text{MHz} \times (9600 / \text{Measured Baud Rate})) \]

Another important component to test was the output of supply voltages to the load circuits. Since any device can be turned on while other devices are already on and vice versa with devices being off, bitwise operations were required to do this. Before supplying 3.3V to the selected pins of the 6-pin header, correct device selection was tested by outputting to the on-board LEDs. After confirming proper bitwise operation by visual inspection, the output was then routed to the 6-pin header (JB port). Device selection was then tested again by confirming 3.3V(on) or 0.0V(off) output after selecting
what device to turn on/off and measuring the appropriate pin via a multimeter. The same procedure was done for the control bits of the analog multiplexer.

As with components, all software additions were tested incrementally to ensure proper navigation, operation, and execution for a desired result. On-board LEDs were used first to test when selected devices were switched on and off. The automation system required significant testing since this allowed the user to input on/off times rather than selecting menu options. First, it was confirmed that the user input matched the output on the screen. After entering on and off times, the software confirms if the times entered are valid. Otherwise, times are not saved and the user must re-enter valid on and off times. After confirming both operated correctly, on and off times of one device was tested. The digital clock on the start screen was observed until the on and off times occurred. Output was rerouted to an on-board LED and turned on and off at the set times. Afterwards, the automation system was scaled up to support all four loads and outputs were rerouted back to the supporting load circuits.

The main concern for building an appropriate circuit for the load was having a significant amount of current for the current sensor to detect. A voltage reading of the current sensor of 0.18V is translated into 1A. This measurement is on a much larger scale than what the original design anticipated. Therefore, a circuit was designed to draw a current of approximately 0.5A. In order to do this, resistor values power ratings had to be heavily considered in order for them to operate within their range and still generate a 0.5A current. It was determined that a number of resistors should be put in parallel in order to reduce the amount of current running through each. When the circuit was implemented successfully running at approximately 0.5A with a power supply, the
current sensor was then measured and a significant reading of 2.57V was outputted. This corresponds with 0.5A running through the circuit.

Another significant component to test was the analog-digital converter (ADC) that will receive select signals from the analog multiplexer. After setting up the SPI to interface with the ADC, a test analog signal was sent from a power supply to the ADC. An analog signal that would be approximate to a value produced by the system was chosen. Therefore, we chose 2.57V, which is the output voltage of the current sensor of the test circuit. Manual calculations of power in kWh/day were done to confirm the correct output value on the LCD. The value appeared to be off by ~10%. This could be due to the rounding during manual calculations or inaccuracy of a power supply that is not calibrated to industry standards.
Roadblocks and Problems / Solutions

The original design allowed the user to add rooms and devices to the system. However, with the needed peripherals and the limited ports of the Nexys board, only one 6-pin header could be used resulting in the system supporting four loads. To power the loads, the IME156 power supply box was used. This provided an easy and available power supply to have at home. However, the accuracy of the power supply output voltage is inexact. To ensure that the correct voltage was being supplied to the supporting loads of the system, the voltage was periodically measured via a portable multimeter.

Another problem encountered was obtaining the correct baud rate for the keyboard to work properly. The first keyboard that was used would display unknown characters when keystrokes occurred. This led to the belief that the UART was functioning correctly but the baud rate was inaccurate. It was found that it is very important to measure the frequency of the keyboard waveform at the smallest transition between a high and a low. Failure to do this will lead to an incorrect baud rate measurement that will result in faulty keyboard operation.

A big challenge of this project was the implementation of the current sensor. The ACS712 current sensor was designed to sense current at a much higher range. As mentioned before, an output of 0.25V translates into 1.0A of current, which is much more than the original design anticipated. Therefore, current could not be sensed since the current being drawn was in the range of 20-25mA. According to this measurement, the voltage output would be a constant ~2.5V, which indicates almost no current or in this case, a very small amount. To obtain a significant reading, a circuit was designed to draw 0.5A.
One of the main problems of the system when implemented with the Nexys board is the voltage drop of the I/O pin supporting the loads. This is due to supporting the load of the system. The I/O pins of the headers of the Nexys board are capable of supporting 100mA of current. Since the load has a significant amount of current going through it, the Nexys board must compensate by dropping the input voltage of 3.3V. This alters the current of the load significantly from 0.5A to about 0.19A. Since this was the case, the LED of the load no longer indicated that the load was on. Therefore the circuit was modified by removing some of the resistances in parallel to provide more current to the LED. Before, when testing the system with 0.5A, the power supply was able to supply more than 100mA of current and therefore was a more ideal power supply versus the Nexys board.

Although the current of the loads have dropped significantly, the current sensor is still able to read the current going through the load. When tested with the power supply to generate 0.5A, the output of the current sensor was 2.57V. When implemented with the Nexys board, the current sensor now generates 2.53V. This is still a readable signal to convert to kWh. However, the signal of the analog multiplexer is slightly noisy on the millivolt scale. This is relatively on the small scale but allows for significant fluctuation of the kilowatt-hours being calculated by several tenths. For example, if the voltage reading of the current sensor of a load ranges from 2.531V to 2.536V, then the kilowatt-hours will range from 1.28kWh to 1.49kWh, which is a significant change.
Results

After testing and determining necessary modifications, the system was able to determine the power used by the loads. The system as a whole proved to be functional for a user to turn on and off loads from a simple to use interface. First, the LCD was successfully implemented to display the menu. Next, the correct baud rate was obtained to successfully connect the keyboard to the system for the user to navigate through the menu and execute operations such as turning on and off loads. After a menu was setup, loads were connected via GPIOs after outputting to LEDs to verify correct bitwise operations. Once successfully implemented, the automation system was applied to the system to turn on and off loads at user-inputted times. Loads were then constructed for carrying 0.5A in order to obtain a significant current sensor reading. However, after connecting it to the Nexys board, it was seen that this was not possible considering the current that the Nexys I/O pins could supply only 100mA. This lowered the supply voltage from the Nexys board, which also lowered the current going through the load. Fortunately, this was readable by the current sensor and was able to be converted to a kWh value and outputted to the LCD.
**Future Work and Impact In Today’s World**

A web application or a computer program could be implemented via Bluetooth to report more comprehensive data such as power consumption throughout the day in graphical form. The website would provide the same information the user is able to view on the LCD display. Graphical representation of power usage throughout the day would give the user more awareness of the peak power consumption times. The user should be more inclined to save power and money after viewing such information. Also, the website will inform the user of each device and their power usage as well.

A redesign could be applied to actual AC outlets and could be tested with actual devices to replace simulation purposes. However, the Nexys Development board will be unable to provide enough current to the supporting circuitry considering the roadblocks mentioned. This redesign would call for a high power system to handle the actual loads of the household devices.

Such a system as this could help household owners lower electricity bills and improve energy consumption considering the heavy concern of energy efficiency and becoming a greener planet by reducing CO₂ emissions. The idea of this system is to help household owners become more aware of their power consumption due to “vampire energy” of devices in idle state and encourage them to lower the energy used by idle devices in their own home. Also, the system allows the user to easily monitor outlets and control them from one unit without physically unplugging devices.

In a recent study by Lawrence Berkeley National Laboratory, the standby power was measured for common household devices. A number of measurements were taken
over a variety of the same kind of type of device. The average standby power was then calculated for that particular device. To gain a sense of the potential savings, we will observe the highest average recorded: 36.5 Watts standby power for a set-top box DVR. This results in 0.0365 kilowatts. Having this type of device in standby mode, we have potential savings of $0.08/day, $2.36/month, and $28.38/year.
Conclusion

Overall, the project was a success as it simulated devices being switched on and off from one unit. This project dealt heavily with digital design and microprocessor based programmable logic as well as fundamentals of electrical engineering. Despite the setbacks, which were explained in “Roadblocks and Problems / Solutions,” the purpose of this project was to propose a household product idea in order for people to be more aware of their energy use. In the end, this was successfully accomplished, as the user was able to turn off different “outlets” or loads, view power consumption, and setup an automated system to help them save energy and money.

This project required design of both hardware and software, strong aspects of computer engineering. The software side required a good amount of interfacing between the Nexys and its peripherals and components such as the UART for translating keystrokes from the keyboard connected via PS/2 connector, and the SPI with the PmodAD1 in order to calculate DC power measurements for each load. With many components involved, it was very important to test each aspect of the project as they were implemented on to the system. This helped isolate problems when they arose.

A great deal of work was put into this project, from research to implementation. It required a vast amount of patience and thinking outside the box if problems hindered progressing forward. Also, completing a project individually cannot be done, as there were many times that assistance was asked from students or professors better skilled in a certain part of the project where their expertise became useful.
Bibliography


"OPB UART Lite (v1.00b)" Xilinx (2005). 2 Dec. 2005


<http://www.xilinx.com/support/documentation/ipembedprocess_peripheralother_opb-spi.htm>

"Phantom Load." Student Sustainability Education. 31 Jan. 2010


Appendix A: Gantt Chart

<table>
<thead>
<tr>
<th>Task &amp; Responsibilities</th>
<th>Start</th>
<th>Finish</th>
<th>Duration</th>
<th>Jan-10</th>
<th>Feb-10</th>
<th>Mar-10</th>
<th>Apr-10</th>
<th>May-10</th>
<th>Jun-10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proposal</td>
<td>1/10</td>
<td>1/17</td>
<td>7d</td>
<td>1/10</td>
<td>1/17</td>
<td>7d</td>
<td>1/10</td>
<td>1/17</td>
<td>7d</td>
</tr>
<tr>
<td>Research</td>
<td>1/17</td>
<td>2/27</td>
<td>42d</td>
<td>1/17</td>
<td>2/27</td>
<td>42d</td>
<td>1/17</td>
<td>2/27</td>
<td>42d</td>
</tr>
<tr>
<td>Vampire Energy</td>
<td>1/17</td>
<td>2/30</td>
<td>13d</td>
<td>1/17</td>
<td>2/30</td>
<td>13d</td>
<td>1/17</td>
<td>2/30</td>
<td>13d</td>
</tr>
<tr>
<td>Household Energy</td>
<td>2/5</td>
<td>2/18</td>
<td>13d</td>
<td>2/5</td>
<td>2/18</td>
<td>13d</td>
<td>2/5</td>
<td>2/18</td>
<td>13d</td>
</tr>
<tr>
<td>Fast Solutions</td>
<td>2/17</td>
<td>2/27</td>
<td>10d</td>
<td>2/17</td>
<td>2/27</td>
<td>10d</td>
<td>2/17</td>
<td>2/27</td>
<td>10d</td>
</tr>
<tr>
<td>Design</td>
<td>2/28</td>
<td>3/12</td>
<td>9d</td>
<td>2/28</td>
<td>3/12</td>
<td>9d</td>
<td>2/28</td>
<td>3/12</td>
<td>9d</td>
</tr>
<tr>
<td>UCD</td>
<td>4/14</td>
<td>5/11</td>
<td>18d</td>
<td>4/14</td>
<td>5/11</td>
<td>18d</td>
<td>4/14</td>
<td>5/11</td>
<td>18d</td>
</tr>
<tr>
<td>Controller - PolyBot</td>
<td>3/18</td>
<td>4/1</td>
<td>17d</td>
<td>3/18</td>
<td>4/1</td>
<td>17d</td>
<td>3/18</td>
<td>4/1</td>
<td>17d</td>
</tr>
<tr>
<td>Implementation</td>
<td>3/31</td>
<td>5/1</td>
<td>33d</td>
<td>3/31</td>
<td>5/1</td>
<td>33d</td>
<td>3/31</td>
<td>5/1</td>
<td>33d</td>
</tr>
<tr>
<td>Nexys Board</td>
<td>3/31</td>
<td>5/1</td>
<td>33d</td>
<td>3/31</td>
<td>5/1</td>
<td>33d</td>
<td>3/31</td>
<td>5/1</td>
<td>33d</td>
</tr>
<tr>
<td>Testing</td>
<td>5/19</td>
<td>6/5</td>
<td>26d</td>
<td>5/19</td>
<td>6/5</td>
<td>26d</td>
<td>5/19</td>
<td>6/5</td>
<td>26d</td>
</tr>
</tbody>
</table>
Appendix B: User Interface Code

houseUnit.c

//HEADER FILES
#include "mainHeader.h"

//GLOBAL VARIABLES
int key_value, key_value1;
int device;
int autoSys;
int menuArea;
int leds;
int inputamt, inputamt2;
int timeOffDone, timeOnDone;
int number[10] = {0x16, 0x1E, 0x26, 0x25, 0x2E, 0x36, 0x3D, 0x3E, 0x46, 0x45};

extern char hrs[];
extern char min[];
extern char sec[];
extern char amPm[];

DEV_STRUCT temp;
DEV_STRUCT TV1, DVD, TV2, REC;

static int dev1_on, dev2_on, dev3_on, dev4_on;
static int outValue;
static char onOffStr[4];

//MAIN
main()
{
    //Set GPIOs for output mode
    XIo_Out32(XPAR_LCD_COMM_BASEADDR + 4, 0);
    XIo_Out32(XPAR_LCD_DATA_BASEADDR + 4, 0);
    XIo_Out32(XPAR_LED_BASEADDR + 4, 0);
    XIo_Out32(XPAR_LOADS_BASEADDR + 4, 0);
    XIo_Out32(XPAR_LOADS_BASEADDR + 4, 0);
    XIo_Out32(XPAR_BTN_BASEADDR + 4, 1);
    XIo_Out32(XPAR_MUX_CNTRL_BITS_BASEADDR + 4, 0);

    //INITIALIZES THE LCD
    init_lcd();

    /* For Digital Clock */
    init_timer();
    init();

    microblaze_enable_interrupts();

    //START SCREEN
write_word("<ENTER> for Menu", 16);
shift(24, RT_SHIFT);

//Display clock
write_time();

while(1)
{
    //Primary check for keystroke
    if(XIo_In32(XPAR_KEYBOARD_BASEADDR + 8) & 0x1)
    {
        //If letter scan code is pressed
        if(XIo_In32(XPAR_KEYBOARD_BASEADDR) == 0xF0)
        {
            //Takes care of debouncing
            while(XIo_In32(XPAR_KEYBOARD_BASEADDR) == 0xF0);
            key_value = XIo_In32(XPAR_KEYBOARD_BASEADDR);
        }
    }

    /************* START SCREEN *************/
    //DETECT IF ENTER IS PUSHED
    if(key_value == ENT_KEY)
    {
        menuArea = 1;   //Goes to first submenu
        ///-1 indicates no scrolling involved in menu
        updateMenu(-1);
        key_value = 0x00;
    }

    //DISPLAYS TIME IF ON START SCREEN
    if(menuArea == 0)
    {
        write_time();
    }

    /********** *** NAVIGATION ***********/
    //BACKSPACE PRESSED - goes back to previous menu
    if(key_value == GO_BACK)
    {
        //PREVENT MENU FROM FAULT
        if(menuArea == 0)
            menuArea = 0;
        if(menuArea == 1)
        {
            menuArea = 0;
            cntrlDataSetup(CLEAR, RTN_DELAY);
            write_word("<ENTER> for Menu", 16);
            shift(24, RT_SHIFT);
        }
        if(menuArea == 2 || menuArea == 10 || menuArea == 11)
        {
            menuArea = 1;
            updateMenu(-1);
        }
    }
if (menuArea == 4 || menuArea == 5) {
    menuArea = 2;
    updateMenu(-1);
}
if (menuArea == 6 || menuArea == 7) {
    menuArea = 4;
    updateMenu(-1);
}
if (menuArea == 8 || menuArea == 9) {
    menuArea = 5;
    updateMenu(-1);
}
key_value = 0x00;
}

if (key_value == UP_KEY) {
//Scroll UP
    updateMenu(1);
    key_value = 0x00;
} else if (key_value == DOWN_KEY) {
//Scroll DOWN
    updateMenu(2);
    key_value = 0x00;
}

/******* OTHER SUB-MENUS **********/
else if(key_value == 0x1E)
{
    menuArea = 5;  //Room 2
    updateMenu(-1);
}
key_value = 0x00;
} else if(menuArea == 4)
{
    if(key_value == 0x16)
    {
        menuArea = 6;  //TV
        device = DEVICE_1;
        updateMenu(-1);
    }
    else if(key_value == 0x1E)
    {
        menuArea = 7;  //DVD
        device = DEVICE_2;
        updateMenu(-1);
    }
    key_value = 0x00;
}
else if(menuArea == 5)
{
    if(key_value == 0x16)
    {
        menuArea = 8;  //TV
        device = DEVICE_3;
        updateMenu(-1);
    }
    else if(key_value == 0x1E)
    {
        menuArea = 9;  //Receiver
        device = DEVICE_4;
        updateMenu(-1);
    }
    key_value = 0x00;
}

/************* TURN ON/OFF *************/
if(key_value == 0x35)  //ON
{
    switchComp(device, 1);
    updateMenu(-1);
    key_value = 0x00;
} else if(key_value == 0x31)  //OFF
{
    switchComp(device, 0);
    updateMenu(-1);
    key_value = 0x00;
}

/************* AUTO SYSTEM *************/
int a;
if(autoSys == 1 && key_value != 0x00)
{
    //Find which number has been pressed
    for(a = 0; a < 10; a++)
    {
        if(number[a] != key_value)
            break;
    }
    //Update input in autoSystem
    autoSystem(device, key_value);
    key_value = 0x00;
}

//Output voltages to physical loads
XIo_Out32(XPAR_LOADS_BASEADDR, leds);

/* Function: init()
* Params: None
* Description: Re-initializes variables */
void init()
{
    device = 0x00;
    menuArea = 0;
    onOffStr[4] = "OFF!";
    leds = 0x0;
    key_value = 0x00;
    inputamt = 0;
    inputamt2 = 0;
    timeOffDone = 0;
    timeOnDone = 0;
}

/* Function: switchComp(int device, int onOff)
* Params: device – ID of device, onOff – indicates
* Description: Responsible for switching loads off/on */
void switchComp(int device, int onOff)
{
    //Clear display
    cntrlDataSetup(CLEAR, RTN_DELAY);

    if(onOff == 0)
    {
        strcpy(onOffStr, "OFF!");
        if((leds != (leds & ~device)))
            XIo_Out32(XPAR_LOADS_BASEADDR, leds);
    }
        //If device is already on, then turn off
        leds = ~(leds ^ ~device);
        XIo_Out32(XPAR_LED_BASEADDR, leds);
    }
    
    else if(onOff == 1)
    {
        strcpy(onOffStr, "ON! ");
        leds |= device;
        XIo_Out32(XPAR_LED_BASEADDR, leds);
    }
    
    switch(device)
    {
        //Determine which device to indicate on/off
        case DEVICE_1:    write_word("TV is ", 6);
                          break;
        case DEVICE_2:    write_word("DVD is ", 7);
                          break;
        case DEVICE_3:    write_word("TV is ", 6);
                          break;
        case DEVICE_4:    write_word("Receiver is ", 12);
                          break;
    }
    write_word(onOffStr, 4);
    
    //PROVIDE TIME TO DISPLAY
    int i;
    for(i = 0; i < DSP_DELAY; i++);
    
    //RETURN TO OPTIONS MENU
    menuArea = 1;
    }
```c
//HEADER FILES
#include "mainHeader.h"

extern int menuArea;
extern int autoSys;
extern int device;
extern int leds;

int power;

/* Function: updateMenu(int scroll_value)
 * Params: scroll_value - determines if menu can scroll
 * Description: Updates the LCD display with the correct
 * submenu.
 */

void updateMenu(int scroll_value)
{
  //Clear display
cntrlDataSetup(CLEAR, RTN_DELAY);

  if(menuArea == 0){}
  else if(menuArea == 1)
  {
    if(scroll_value == 1 || scroll_value == -1)
    {
      write_word("1.Control Power", 15);
      shift(25, RT_SHIFT);
      write_word("2.Auto System", 13);
    }
    else if(scroll_value == 2)
    {
      write_word("2.Auto System", 13);
      shift(27, RT_SHIFT);
      write_word("3.Information", 13);
    }
    autoSys = 0;
  }
  else if(menuArea == 2)
  {
    write_word("1.Room 1", 8);
    shift(32, RT_SHIFT);
    write_word("2.Room 2", 8);
  }
  else if(menuArea == 4)
  {
    write_word("1.TV", 4);
    shift(36, RT_SHIFT);
    write_word("2.DVD", 5);
  }
  else if(menuArea == 5)
  {
```
write_word("1.TV", 4);
shift(36, RT_SHIFT);
write_word("2.Receiver", 10);
}
else if(menuArea == 6)
{
if(autoSys == 0)
{

write_word("TV", 2);
shift(7, RT_SHIFT);

//POWER = (((adc_get() - Vref)/4095) * 5V)/CurSense Rate)* Supply Voltage
if(leds & 0x01)
{
XIo_Out32(XPAR_MUX_CNTRL_BITS_BASEADDR, 0x0);
power = (((adc_get() - 3103)*139000)/4095);
print_power(power);
power = 0;
}
else
{
write_word("0.00kWh", 7);
}
shift(24, RT_SHIFT);
write_word("ON(Y) / OFF(N)", 14);
}
else if(autoSys == 1)
autoSystem(device, -1);
}
else if(menuArea == 7)
{
XIo_Out32(XPAR_MUX_CNTRL_BITS_BASEADDR, 0x1);
if(autoSys == 0)
{
write_word("DVD", 3);
shift(7, RT_SHIFT);

if(leds & 0x02)
{
power = (((adc_get() - 3103)*139000)/4095); 
print_power(power);
}
else
{
write_word("0.00kWh", 7);
}

shift(24, RT_SHIFT);
write_word("ON(Y) / OFF(N)", 14);
}
else if(autoSys == 1)
autoSystem(device, -1);
}
else if(menuArea == 8)
{
if(autoSys == 0)
{
    write_word("TV", 2);
    shift(7, RT_SHIFT);

    if(leds & 0x04)
    {
        write_word("2.12kWh", 7);
    }
    else
    {
        write_word("0.00kWh", 7);
    }

    shift(24, RT_SHIFT);
    write_word("ON(Y) / OFF(N)", 14);
}
else if(autoSys == 1)
    autoSystem(device, -1);

else if(menuArea == 9)
{
    if(autoSys == 0)
    {
        write_word("Receiver", 8);
        shift(1, RT_SHIFT);

        if(leds & 0x08)
        {
            write_word("3.74kWh", 7);
        }
        else
        {
            write_word("0.00kWh", 7);
        }

        shift(24, RT_SHIFT);
        write_word("ON(Y) / OFF(N)", 14);
    }
    else if(autoSys == 1)
        autoSystem(device, -1);
}
else if(menuArea == 10)
{
    autoSys = 1;
    write_word("Select room", 11);
    //PROVIDE TIME TO DISPLAY
    int i;
    for(i = 0; i < 119189189; i++);
    menuArea = 2;
    updateMenu(-1);
}
else if(menuArea == 11)
{
    power = 0;
    write_word("Total DC Power:", 15);
shift(25, RT_SHIFT);

if((leds & 0x1)==0x1)
{
    XIo_Out32(XPAR_MUX_CNTRL_BITS_BASEADDR, 0x0);
    power += (((adc_get() - 3103) * 139000) / 4095);
}
if((leds & 0x2)==0x2)
{
    XIo_Out32(XPAR_MUX_CNTRL_BITS_BASEADDR, 0x1);
    power += (((adc_get() - 3103) * 139000) / 4095);
}
if(leds & 0x4)
    power += 2120;
if(leds & 0x8)
    power += 3740;

print_power(power);

/* Function: print_power(int kwh1)
 * Params: kwh1 - number to convert to chars
 * Description: Converts the power into chars to be printed on the LCD.
 */
void print_power(int kwh1)
{
    write_char(print_digit(kwh1, 1000));
    write_char('.');
    write_char(print_digit(kwh1, 100));
    write_char(print_digit(kwh1, 10));
    write_word("kWh", 3);
}
autoSystem.c

#include "mainHeader.h"

extern char hrs[];
extern char min[];
extern char sec[];
extern char amPm[];

extern int key_value;
extern int key_value1;
extern int device;
extern int autoSys;
extern int menuArea;
extern int inputamt;
extern int inputamt2;
extern int timeOffDone;
extern int timeOnDone;
extern int leds;

extern DEV_STRUCT TV1, DVD, TV2, REC;
extern DEV_STRUCT temp;

int i;

/* Function: autoSystem(int device, int key_value)
* Params: device - device ID, key_value - the key scan code
* Description: Allows user to input times for devices to be turned on/off. Error checking. */

void autoSystem(int device, int key_value)
{
    cntrlDataSetup(CLEAR, RTN_DELAY);
    if(key_value != 0x00 && timeOffDone != 1)
    {//FOR OFF TIME INPUT
        write_word("Turn off at:", 12);
        shift(28, RT_SHIFT);
        if(key_value != -1)
        {//Displays the time as user makes keystrokes
            temp.off_time[inputamt] = KBtable(key_value);
            write_word(temp.off_time, inputamt+1);
            inputamt++;
        }
        if(inputamt == 2)
        {//Automatically enters the first ':' symbol
            temp.off_time[inputamt] = ':';
            write_char(":");
            inputamt++;
        }
    }
}
else if(inputamt == 5)
{//Automatically enters the second ':' symbol and 00
//seconds
    temp.off_time[inputamt] = ':';
    inputamt++;
    temp.off_time[inputamt] = '0';
    inputamt++;
    temp.off_time[inputamt] = '0';
    inputamt++;
    temp.off_time[inputamt] = ' ';
    write_char(" ");
    inputamt++;
}
else if(inputamt == 11)
{//Signals that the input is done for off time
    timeOffDone = 1;
    for(i = 0; i < DSP_DELAY; i++);
    updateMenu(-1);
}
else if(key_value != 0x00 && timeOffDone == 1)
{//FOR ON TIME INPUT
    cntrlDataSetup(CLEAR, RTN_DELAY);
    write_word("Turn on at:", 11);
    shift(29, RT_SHIFT);
    if(key_value != -1)
    {//Displays the time as user makes keystrokes
        temp.on_time[inputamt2] = KBtable(key_value);
        write_word(temp.on_time, inputamt2+1);
        inputamt2++;
    }
    if(inputamt2 == 2)
    {//Automatically enters the first ':' symbol
        temp.on_time[inputamt2] = ':';
        write_char(":");
        inputamt2++;
    }
    else if(inputamt2 == 5)
    {//Automatically enters the second ':' symbol and 00
        //seconds
        temp.on_time[inputamt2] = ':';
        inputamt2++;
        temp.on_time[inputamt2] = '0';
        inputamt2++;
        temp.on_time[inputamt2] = '0';
        inputamt2++;
        temp.on_time[inputamt2] = ' ';
        write_char(" ");
        inputamt2++;
    }
    else if(inputamt2 == 11)
    {
        if(isValidTimeInput(temp))
        {//CHECKS IF BOTH ON AND OFF TIMES ARE VALID AFTER
//FINISHING INPUT

switch(device)
{//Copies input times into appropriate device struct
    case DEVICE_1:
        strcpy(TV1.off_time, temp.off_time);
        strcpy(TV1.on_time, temp.on_time);
        TV1.device = DEVICE_1;
        break;
    case DEVICE_2:
        strcpy(DVD.off_time, temp.off_time);
        strcpy(DVD.on_time, temp.on_time);
        DVD.device = DEVICE_2;
        break;
    case DEVICE_3:
        strcpy(TV2.off_time, temp.off_time);
        strcpy(TV2.on_time, temp.on_time);
        TV2.device = DEVICE_3;
        break;
    case DEVICE_4:
        strcpy(REC.off_time, temp.off_time);
        strcpy(REC.on_time, temp.on_time);
        REC.device = DEVICE_4;
        break;
}
else
{//If invalid time present, prints message
cntlDataSetup(CLEAR, RTN_DELAY);
    write_word("Invalid time(s).", 16);
}

//Re-initialize variables
autoSys = 0;
timeOffDone = 0;
inputamt = 0;
inputamt2 = 0;

//Delay before returning back to menu
for(i = 0; i < DSP_DELAY; i++);
menuArea = 1;
updateMenu(-1);
}

/* Function: isValidTimeInput(DEV_STRUCT temp)
* Params: temp - a struct that holds the times to validate
* Description: Determines if times inputted are valid.
*/

int isValidTimeInput(DEV_STRUCT temp)
{
    int value = 0;

    //CHECKS VALID OFF TIME
    if(temp.off_time[0] > '1')
    {//Checks hours
value = 0;
}
else
{//Checks hours
if((temp.off_time[0] == '0' && temp.off_time[1] > '0' &&
temp.off_time[1] < '10')||(temp.off_time[0] == '1' &&
temp.off_time[1] < '3'))
 {
   {//Checks minutes
        if((temp.off_time[9] == 'a' || temp.off_time[9] ==
            'p') && temp.off_time[10] == 'm')
        {//Checks am/pm
            value = 1;
        }
        else
            value = 0;
    }
    else
        value = 0;
}
else
    value = 0;
}

//CHECKS VALID ON TIME
if(temp.on_time[0] > '1')
{//Check hours
    value = 0;
}
else
{//Check hours
        if((temp.on_time[0] == '0' && temp.on_time[1] > '0' &&
temp.on_time[1] < '10')||(temp.on_time[0] == '1', &&
temp.on_time[1] < '3'))
        {
            {//Check minutes
                if((temp.on_time[9] == 'a' || temp.on_time[9] ==
                    'p') && temp.on_time[10] == 'm')
                    {//Check am/pm
                        value = 1;
                    }
                else
                    value = 0;
            }
            else
                value = 0;
        }
    else
        value = 0;
}
return value;
/* Function:       checkDevTimes() 
* Params:         None 
* Description:   Checks if on/off times for each device 
*                 has occurred 
*/

void checkDevTimes()
{
    int device1 = 0;

    //CHECK OFF TIMES
    {
        if((leds != (leds & ~DEVICE_1)))
            //If device is already on, then turn off
            leds = ~(leds ^ ~DEVICE_1);
    }

    if(DVD.off_time[0] == hrs[0] && DVD.off_time[1] == hrs[1] &&
    {
        if((leds != (leds & ~DEVICE_2)))
            //If device is already on, then turn off
            leds = ~(leds ^ ~DEVICE_2);
    }

    {
        if((leds != (leds & ~DEVICE_3)))
            //If device is already on, then turn off
            leds = ~(leds ^ ~DEVICE_3);
    }

    {
        if((leds != (leds & ~DEVICE_4)))
            //If device is already on, then turn off
            leds = ~(leds ^ ~DEVICE_4);
    }

    //CHECK ON TIMES
    if(TV1.on_time[0] == hrs[0] && TV1.on_time[1] == hrs[1] &&

{
  leds |= DEVICE_1;
}
if(DVD.on_time[0] == hrs[0] && DVD.on_time[1] == hrs[1] &&
{
  leds |= DEVICE_2;
}
if(TV2.on_time[0] == hrs[0] && TV2.on_time[1] == hrs[1] &&
{
  leds |= DEVICE_3;
}
{
  leds |= DEVICE_4;
}

// Output to loads
XIo_Out32(XPAR_LED_BASEADDR, leds);
//HEADER FILES
#include "mainHeader.h"

/* Function: init_lcd()
 * Params: None
 * Description: Initializes LCD by calling commDataSetup() for each required function call.
 */
void init_lcd()
{
    unsigned int i;
    for(i = 0; i<30000000; i++);
    cntrlDataSetup(0x38, 75000); //FUNCTION SET
    cntrlDataSetup(0x0C, 750000); //DISPLAY CNTRL
    cntrlDataSetup(0x01, 2100000); //DISPLAY CLEAR
    cntrlDataSetup(0x06, 75000); //ENTRY MODE
}

/* Function: commDataSetup(int code, int delay)
 * Params: code - the function code
 * delay - the required time delay to do the function
 * Description: Does the required register select, enable, function code and disables. Then does the required delay for the function
 */
void cntrlDataSetup(int code, int delay)
{
    unsigned int i;
    XIo_Out32(XPAR_LCD_COMM_BASEADDR, 0); //register select
    XIo_Out32(XPAR_LCD_COMM_BASEADDR, 1); //enable
    XIo_Out32(XPAR_LCD_DATA_BASEADDR, code); //function code
    XIo_Out32(XPAR_LCD_COMM_BASEADDR, 0); //disable
    for(i=0; i<delay; i++);
}

/* Function: write_letter(char letter)
 * Params: char letter - the character to display on the LCD
 * Description: Does the required register select, enable, passes letter to the lcd_data line and disables. Then delays before the next LCD operation.
 */
void write_char(char letter)
{
    unsigned int i;
    XIo_Out32(XPAR_LCD_COMM_BASEADDR, 4); //register select
    XIo_Out32(XPAR_LCD_COMM_BASEADDR, 5); //enable
    //function to write letter
XIo_Out32(XPAR_LCD_DATA_BASEADDR, letter);
XIo_Out32(XPAR_LCD_COMM_BASEADDR, 4);  //disable
for(i=0; i<75000; i++);  //Delay before next LCD operation
}

/*
 * Function: write_word(char word[])
 * Params: char word[]
 * Description: Called to write character strings to the display.
 * Loops through the parameter, word[], and sends
 * individual character to the function write_letter for
 * displaying on LCD.
 */
void write_word(char word[], int length)
{
    int i;
    for(i = 0; i<length; i++)
        write_char(word[i]);
}

/*
 * Function: shift(int amt, int code)
 * Params: int amt - amount to shift
 * int code - determines shift right or shift left
 * Description: Shifts the cursor 'amt' right or left determined by
 * 'code'
 */
void shift(int amt, int code)
{
    int n;
    for(n = 0; n < amt; n++)
        cntrlDataSetup(code, SHIFT_DELAY);
}

/*
 * Function: print_digit(int value, int scale)
 * Params: value - number to convert. scale - number to
 * divide by
 * Description: Converts the value to a char.
 */
char print_digit(int value, int scale)
{
    return (value/scale)%10 + '0';
}
digclk.c

//HEADER FILES
#include "mainHeader.h"

//INITIALIZED VALUES
char hrs[3] = {'1','2',':'};
char min[3] = {'0','0',':'};
char sec[3] = {'0','0',' '};
char amPm[3] = "am ";

/* Function: update_time() */
/* Params: None */
/* Description: Continuously updates the time display and prints */
/* to the LCD */
/*/ 
void update_time()
{
    checkDevTimes();
    sec[1]++;
    if(sec[1] == ':')
    {
        sec[1] = '0';
        sec[0]++;
        if(sec[0] == '6') //Seconds rollover
        {
            sec[0] = '0';
            min[1]++;
            if(min[1] == ':')
            {
                min[1] = '0';
                min[0]++;
                if(min[0] == '6') //Minutes rollover
                {
                    min[0] = '0';
                    hrs[1]++;
                    if(hrs[1] == ':')
                    {
                        hrs[1] = '0';
                        hrs[0] = '1';
                    }
                }
            }
        }
    }
    //Hours rollover
    if(hrs[0] == '1' && hrs[1] == '3')
    {
        hrs[0] = '0';
        hrs[1] = '1';
    }
    //AMPM switch
    if(hrs[0] == '1' && hrs[1] == '2')
    {
        if(amPm[0] == 'a' & amPm[1] == 'm')
        {
```c
strcpy(amPm,"pm ");
else if(amPm[0] == 'p' & amPm[1] == 'm')
    { 
    strcpy(amPm,"am ");
    }
}
}
}
}

/* Function:         manual_update() 
*  Params:          None
*  
*  Description:     Reads the from the buttons register to determine 
*                   whether to update hrs or min. Also updates calendar 
*                   accordingly.
*/
void manual_update()
{
    //Read from register to get button status
    int BTN_STAT = XIo_In32(XPAR_BTN_BASEADDR);

    //UPDATE MINUTES
    if(BTN_STAT & 0x2)
    {
        min[1]++;

        //MINUTES ROLLOVER
        if(min[1] == ':')
            {
                min[1] = '0';
                min[0]++;
                if(min[0] == '6')
                    {
                        min[0] = '0';
                    }
            }
    }
}

    //UPDATE HOURS
    else if(BTN_STAT & 0x4)
    {
        hrs[1]++;

        //HOURS ROLLOVER TO 10 AND TO 01
        if(hrs[1] == ':')
            {
                hrs[1] = '0';
                hrs[0] = '1';
            }
        if(hrs[0] == '1' & hrs[1] == '3')
            {
                hrs[0] = '0';
                hrs[1] = '1';
            }
    }
```
//AMPM SWITCH
if(hrs[0] == '1' && hrs[1] == '2')
{
    if(amPm[0] == 'a' & amPm[1] == 'm')
    {
        strcpy(amPm,"pm ");
    }
    else if(amPm[0] == 'p' & amPm[1] == 'm')
    {
        strcpy(amPm,"am ");
    }
}

/* Function: write_time()
 * Params: None
 * Description: Writes the time to the screen.
 */
void write_time()
{
    write_word(hrs, 3);
    write_word(min, 3);
    write_word(sec, 3);
    write_word(amPm, 3);

    shift(12, LT_SHIFT);  //RETURN HOME
    cntrlDataSetup(0x06, 75000);  //ENTRY MODE
Appendix C: Support Code

\textit{timer.c}

//HEADER FILES
#include "mainHeader.h"

void my_isr()__attribute__((interrupt_handler));

/* Function: init_timer() 
* Params: None 
* Description: Initializes the timer_counter peripheral by loading 
a starting value in the load register and starting 
the counter. */
void init_timer()
{
    XIo_Out32(XPAR_TIMER_COUNTER_BASEADDR+0x04, 25000000);
    //Load TLR0 with value 50000000
    XIo_Out32(XPAR_TIMER_COUNTER_BASEADDR, 0x172);
    XIo_Out32(XPAR_TIMER_COUNTER_BASEADDR, 0x1D2); //Starting timer.
}

/* Function: my_isr() 
* Params: None 
* Description: Starts the interrupt service routine to update the 
time display 
*/
void my_isr()
{
    //Toggle for manual updating hrs and min
    static int halfsec = 0;
    halfsec = !halfsec;

    //Reads the status register
    int STAT_REG = XIo_In32(XPAR_TIMER_COUNTER_BASEADDR);
    //Determines what buttons are being pushed
    int BTN_STAT = XIo_In32(XPAR_BTN_BASEADDR);
    int LED_STAT = XIo_In32(XPAR_LED_BASEADDR);

    //If interrupt has occured
    if(STAT_REG & (1 << 8))
    {
        if(BTN_STAT & 0x1)
        { //Goes into manual mode
            manual_update();
        }
    }
}


//Seconds increment every half second
if(halfsec == 0)
{
    update_time();
}

//Clears Interrupt Bit
STAT_REG |= (1 << 8);
XIo_Out32(XPAR_TIMER_COUNTER_BASEADDR, STAT_REG);
}
#include "mainHeader.h"

/*
 * Function: char KBtable(int data)
 * Params: int data - scan code from keyboard
 * Description: LUT for the keyboard to return the correct char on the LCD
 */

char KBtable(int data)
{
    switch(data)
    {
    case 0x1C: return 'a';
    case 0x32: return 'b';
    case 0x21: return 'c';
    case 0x23: return 'd';
    case 0x24: return 'e';
    case 0x2B: return 'f';
    case 0x34: return 'g';
    case 0x33: return 'h';
    case 0x43: return 'i';
    case 0x3B: return 'j';
    case 0x42: return 'k';
    case 0x4B: return 'l';
    case 0x3A: return 'm';
    case 0x31: return 'n';
    case 0x44: return 'o';
    case 0x4D: return 'p';
    case 0x15: return 'q';
    case 0x2D: return 'r';
    case 0x1B: return 's';
    case 0x2C: return 't';
    case 0x3C: return 'u';
    case 0x2A: return 'v';
    case 0x1D: return 'w';
    case 0x22: return 'x';
    case 0x35: return 'y';
    case 0x1A: return 'z';
    case 0x75: return 'u';
    case 0x72: return 'd';
    case 0x5A: return ' ';  // Space
    case 0x1E: return '2';
    case 0x26: return '3';
    case 0x25: return '4';
    case 0x2E: return '5';
    case 0x36: return '6';
    case 0x3D: return '7';
    case 0x3E: return '8';
    case 0x46: return '9';
    case 0x45: return '0';
    }
    return 'default char';  // If data is not in the switch statement
}
adcGet.c

#include "mainHeader.h"

int ADC_SPICR = XPAR_SPI_BASEADDR + 0x60;
int ADC_SPIDTR = XPAR_SPI_BASEADDR + 0x68;
int ADC_SPISSR = XPAR_SPI_BASEADDR + 0x70;
int ADC_SPIISR = XPAR_SPI_BASEADDR + 0x64;
int ADC_IPISR = XPAR_SPI_BASEADDR + 0x20;
int ADC_SPIDRR = XPAR_SPI_BASEADDR + 0x6C;

int adc_get() {

    int read_upper_data;
    int read_lower_data;
    int read_data;

    //4. Set mode, inhibit Master, disable SPI
    XIo_Out32(ADC_SPICR, 0x194);  

    //4. Write DUMMY data: Send lower byte to DTR
    XIo_Out8(ADC_SPIDTR, 0xF);    //Send MSByte first...

    //5. Disable SSR: Disconnect ADC
    XIo_Out32(ADC_SPISSR, 0xFFF);

    //6. Set mode, inhibit Master, enable SPI
    XIo_Out32(ADC_SPICR, 0x196);

    //7. Enable SSR: Activate ADC
    XIo_Out32(ADC_SPISSR, 0);

    //8. Set mode, UNinhibit Master (UNFREEZE)
    XIo_Out32(ADC_SPICR, 0x096);

    //9. Poll TxEmpty
    while ( (XIo_In32(ADC_SPIISR) & 0x4) == 0);

    //10. Byte transfer done! Inhibit Master (FREEZE)
    XIo_Out32(ADC_SPICR, 0x196);

    //Get upper byte from ADC (with leading 0s)
    read_upper_data = (XIo_In32(ADC_SPIDRR) & 0x000000FF);

    //shift to correct bit significance position
    //for concat with lower byte
    read_upper_data = read_upper_data << 8;

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

    //Repeat 10. Write new DUMMY data: Send lower byte to DTR
    XIo_Out8(ADC_SPIDTR, 0xF);   //...send LSByte

    //Repeat 10. UNFREEZE
XIo_Out32(ADC_SPICR, 0x096);

//Repeat 10. Poll TxEmpty
while ((XIo_In32(ADC_SPISR) & 0x4) == 0);

//Repeat 10. Byte transfer done! FREEZE
XIo_Out32(ADC_SPICR, 0x196);

//Get lower byte from ADC
read_lower_data = (XIo_In32(ADC_SPIDRR) & 0x000000FF);

//----------------------------------

//Disable SSR: Deactivate ADC
XIo_Out32(ADC_SPISSR, 0xFFF);

//Set mode, inhibit Master, disable SPI
XIo_Out32(ADC_SPICR, 0x194);

read_data = read_upper_data | read_lower_data;
return read_data;
}
mainHeader.h

//HEADER FILES
#include "xparameters.h"
#include "xio.h"
#include "stdlib.h"
#include "string.h"
#include "stdio.h"
#include "time.h"
#include "mb_interface.h"

//Variable Definitions
#define SHIFT_DELAY 53000
#define RTN_DELAY 2100000
#define ENTRY_DELAY 75000
#define DISCON_DELAY 750000
#define DSP_DELAY 119189189
#define LT_SHIFT 0x10
#define RT_SHIFT 0x14
#define ENT_KEY 0x5A
#define RTN_HOME 0x02
#define CLEAR 0x01
#define GO_BACK 0x66
#define UP_KEY 0x75
#define DOWN_KEY 0x72
#define DEVICE_1 0x01
#define DEVICE_2 0x02
#define DEVICE_3 0x04
#define DEVICE_4 0x08

typedef struct dev
{
   int device;
   char off_time[11];
   char on_time[11];
} DEV_STRUCT;

void init();
void my_isr();

//LCD Function Prototypes
void init_lcd();
void cntrlDataSetup(int code, int delay);
void write_char(char letter);
void write_word(char word[], int length);
void shift(int amt, int code);
char print_digit(int value, int scale);

//DIGITAL CLOCK Function Prototypes
void update_time();
void manual_time();
void write_time();
//AUTO SYSTEM Function Prototypes
void autoSystem(int device, int key_value);
void checkDevTimes();
int isValidTimeInput(DEV_STRUCT temp);
void switchComp(int device, int onOff);

//LUT Function Prototype
char KBtable(int data);

//MENU Function Prototypes
void updateMenu();
void print_power(int kwh1);

//ADC Function Prototype
int readDataFromSPI_MSB();
Appendix D: Component Pinouts

**Figure 13** Pinout of CD74HC4067 Analog Multiplexer

**Figure 14** Pinout of ACS712 Current Sensor
## Appendix E: Bill of Materials

<table>
<thead>
<tr>
<th>Item</th>
<th>Quantity</th>
<th>Total Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 10ohm/1W</td>
<td>4</td>
<td>3.00</td>
</tr>
<tr>
<td>2 Current Sensors</td>
<td>2</td>
<td>20.00</td>
</tr>
<tr>
<td>1 Analog Mux</td>
<td>1</td>
<td>5.00</td>
</tr>
<tr>
<td>2 BJTs</td>
<td>2</td>
<td>0.35</td>
</tr>
<tr>
<td>Various Resistors</td>
<td></td>
<td>0.25</td>
</tr>
<tr>
<td>2 LEDs</td>
<td>2</td>
<td>0.30</td>
</tr>
<tr>
<td>Breadboard</td>
<td>1</td>
<td>15.00</td>
</tr>
<tr>
<td>Break away headers</td>
<td>1</td>
<td>1.50</td>
</tr>
<tr>
<td>PS/2 Connector</td>
<td>1</td>
<td>10.00</td>
</tr>
<tr>
<td>PmodAD1</td>
<td>1</td>
<td>25.00</td>
</tr>
<tr>
<td>Nexys Development Board</td>
<td>1</td>
<td>120.00</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td><strong>200.40</strong></td>
</tr>
<tr>
<td>Task</td>
<td>Hours</td>
<td></td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------</td>
<td></td>
</tr>
<tr>
<td>Research</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>Meetings</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Software Design</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>Software Implementation</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>Hardware Design</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>Hardware Implementation</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>Testing</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Report</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>225 Hours</strong></td>
<td></td>
</tr>
</tbody>
</table>
Appendix G: Analysis of Project

This appendix includes applicable information in response to the “Analysis of Senior Project Design” questions.

Summary of Functional Requirements
Describe the overall capabilities of functions of your project or design. Describe what your project does. (Do not describe how you designed it.)

The user will be welcomed to the system by a start screen, which will include the current time and instructions to press enter to activate the menu. There will be a series of submenus for the user to perform desired tasks such as controlling power outlets, setting up the automation system for certain devices at user defined on and off times, and viewing current power consumption. Since this is a simulation of turning off power outlets in a household, we will have two rooms to control two devices in each room. The user should be able to control this system by keyboard and the system should respond accordingly. To visually show the user that devices are being turned on/off, four LEDs will be used to represent each device.

Primary Constraints
Describe significant challenges or difficulties associated with your project or implementation. For example, what were limiting factors or other issues that impacted your approach? What made your project difficult? What parameters or specifications limited your options or directed your approach?

The original design allowed the user to add rooms and devices to the system. However, with the needed peripherals and the limited ports of the Nexys board, only one 6-pin header could be used resulting in the system supporting four loads. To power the loads, the IME156 power supply box was used. This provided an easy and available power supply to have at home. However, the accuracy of the power supply output voltage is inexact. To ensure that the correct voltage was being supplied to the supporting loads of the system, the voltage was periodically measured via a portable multimeter.

Another problem encountered was obtaining the correct baud rate for the keyboard to work properly. The first keyboard that was used would display unknown characters when keystrokes occurred. This led to the belief that the UART was functioning correctly but the baud rate was inaccurate. It was found that it is very important to measure the frequency of the keyboard waveform at the smallest transition between a high and a low. Failure to do this will lead to an incorrect baud rate measurement that will result in faulty keyboard operation.
A big challenge of this project was the implementation of the current sensor. The ACS712 current sensor was designed to sense current at a much higher range. As mentioned before, an output of 0.25V translates into 1.0A of current, which is much more than the original design anticipated. Therefore, current could not be sensed since the current being drawn was in the range of 20-25mA. According to this measurement, the voltage output would be a constant ~2.5V, which indicates almost no current or in this case, a very small amount. To obtain a significant reading, a circuit was designed to draw 0.5A.

One of the main problems of the system when implemented with the Nexys board is the voltage drop of the I/O pin supporting the loads. This is due to supporting the load of the system. The I/O pins of the headers of the Nexys board are capable of supporting 100mA of current. Since the load has a significant amount of current going through it, the Nexys board must compensate by dropping the input voltage of 3.3V. This alters the current of the load significantly from 0.5A to about 0.19A. Since this was the case, the LED of the load no longer indicated that the load was on. Therefore the circuit was modified by removing some of the resistances in parallel to provide more current to the LED. Before, when testing the system with 0.5A, the power supply was able to supply more than 100mA of current and therefore was a more ideal power supply versus the Nexys board.

Although the current of the loads have dropped significantly, the current sensor is still able to read the current going through the load. When tested with the power supply to generate 0.5A, the output of the current sensor was 2.57V. When implemented with the Nexys board, the current sensor now generates 2.53V. This is still a readable signal to convert to kWh. However, the signal of the analog multiplexer is slightly noisy on the millivolt scale. This is relatively on the small scale but allows for significant fluctuation of the kilowatt-hours being calculated by several tenths. For example, if the voltage reading of the current sensor of a load ranges from 2.531V to 2.536V, then the kilowatt-hours will range from 1.28kWh to 1.49kWh, which is a significant change.
Economic

Original estimated cost of component parts (as of the start of your project)

<table>
<thead>
<tr>
<th>Item</th>
<th>Total Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 10ohm/1W</td>
<td>6.00</td>
</tr>
<tr>
<td>4 Current Sensors</td>
<td>40.00</td>
</tr>
<tr>
<td>1 Analog Mux</td>
<td>5.00</td>
</tr>
<tr>
<td>4 BJT's</td>
<td>0.70</td>
</tr>
<tr>
<td>Various Resistors</td>
<td>0.50</td>
</tr>
<tr>
<td>4 LEDs</td>
<td>0.60</td>
</tr>
<tr>
<td>Breadboard</td>
<td>15.00</td>
</tr>
<tr>
<td>Break away headers</td>
<td>3.00</td>
</tr>
<tr>
<td>PS/2 Connector</td>
<td>10.00</td>
</tr>
<tr>
<td>PmodAD1</td>
<td>25.00</td>
</tr>
<tr>
<td>Nexys Development Board</td>
<td>120.00</td>
</tr>
<tr>
<td>Total</td>
<td>225.80</td>
</tr>
</tbody>
</table>

Actual final cost of component parts (at the end of your project)

<table>
<thead>
<tr>
<th>Item</th>
<th>Total Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 10ohm/1W</td>
<td>3.00</td>
</tr>
<tr>
<td>2 Current Sensors</td>
<td>20.00</td>
</tr>
<tr>
<td>1 Analog Mux</td>
<td>5.00</td>
</tr>
<tr>
<td>2 BJT's</td>
<td>0.35</td>
</tr>
<tr>
<td>Various Resistors</td>
<td>0.25</td>
</tr>
<tr>
<td>2 LEDs</td>
<td>0.30</td>
</tr>
<tr>
<td>Breadboard</td>
<td>15.00</td>
</tr>
<tr>
<td>Break away headers</td>
<td>1.50</td>
</tr>
<tr>
<td>PS/2 Connector</td>
<td>10.00</td>
</tr>
<tr>
<td>PmodAD1</td>
<td>25.00</td>
</tr>
<tr>
<td>Nexys Development Board</td>
<td>120.00</td>
</tr>
<tr>
<td>Total</td>
<td>200.40</td>
</tr>
</tbody>
</table>

Original estimated development time (as of the start of your project)

<table>
<thead>
<tr>
<th>Task</th>
<th>Hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Research</td>
<td>50</td>
</tr>
<tr>
<td>Meetings</td>
<td>10</td>
</tr>
<tr>
<td>Software Design</td>
<td>10</td>
</tr>
<tr>
<td>Software Implementation</td>
<td>40</td>
</tr>
<tr>
<td>Hardware Design</td>
<td>20</td>
</tr>
<tr>
<td>Hardware Implementation</td>
<td>50</td>
</tr>
<tr>
<td>Testing</td>
<td>20</td>
</tr>
<tr>
<td>Report</td>
<td>20</td>
</tr>
<tr>
<td>Total</td>
<td>220 Hours</td>
</tr>
</tbody>
</table>
- Actual development time (at the end of your project)

<table>
<thead>
<tr>
<th>Task</th>
<th>Hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Research</td>
<td>30</td>
</tr>
<tr>
<td>Meetings</td>
<td>5</td>
</tr>
<tr>
<td>Software Design</td>
<td>15</td>
</tr>
<tr>
<td>Software Implementation</td>
<td>50</td>
</tr>
<tr>
<td>Hardware Design</td>
<td>25</td>
</tr>
<tr>
<td>Hardware Implementation</td>
<td>60</td>
</tr>
<tr>
<td>Testing</td>
<td>20</td>
</tr>
<tr>
<td>Report</td>
<td>20</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>225 Hours</strong></td>
</tr>
</tbody>
</table>
Environmental
Describe any environmental impact associated with manufacturing or use.

Such a system as this could help household owners lower electricity bills and improve energy consumption considering the heavy concern of energy efficiency and becoming a greener planet by reducing CO\textsubscript{2} emissions. The idea of this system is to help household owners become more aware of their power consumption due to “vampire energy” of devices in idle state and encourage them to lower the energy used by idle devices in their own home. Also, the system allows the user to easily monitor outlets and control them from one unit without physically unplugging devices.

In a recent study by Lawrence Berkeley National Laboratory, the standby power was measured for common household devices. A number of measurements were taken over a variety of the same kind of type of device. The average standby power was then calculated for that particular device. To gain a sense of the potential savings, we will observe the highest average recorded: 36.5 Watts standby power for a set-top box DVR. This results in 0.0365 kilowatts. Having this type of device in standby mode, we have potential savings of $0.08/day, $2.36/month, and $28.38/year.

Sustainability
- Describe how the project impacts the sustainable use of resources.

One of the project’s main purposes is to make the user more aware of the energy their household is consuming, more specifically standby power. Standby power accounts for 1% of CO\textsubscript{2} emissions, which is detrimental to the environment. The use of this project would help reduce energy usage and CO\textsubscript{2} emissions.

- Describe any upgrades that would improve the design of the project.

The design could be applied to actual AC outlets and could be tested with actual devices to replace simulation purposes. A web application or a computer program could be implemented via Bluetooth to report more comprehensive data such as power consumption throughout the day in graphical form. The website would provide the same information the user is able to view on the LCD display. Graphical representation of power usage throughout the day would give the user more awareness of the peak power consumption times. The user should be more inclined to save power and money after viewing such information. Also, the website will inform the user of each device and their power usage as well.
Describe any issues or challenges associated with upgrading the design.

With the upgrades mentioned above, the Nexys Development board will be unable to provide enough current to the supporting circuitry considering the roadblocks mentioned above. This redesign would call for a high power system to handle the actual loads of the household devices.