ENERGY SMART HOME

ECE CAPSTONE 2014

Advisor: Dr. Hana Godrich

Mark Zoppina, Tom Rizzie, Andrew Demoleas, Irfan Gunes, Ubaid Haroon
ENERGY SMART HOME

ABSTRACT

As energy consumption around the world increases so does the need for a smart grid. This can only be implemented if a consumer uses smart applications and strategically plans to use their appliances during non-peak hours to conserve the amount of strain on the system. We can solve this problem by providing an interface that will inform users of their power consumption and enable them to automate certain processes that require large amounts of energy. This knowledge for energy consumption will ultimately save the user money for wasted and unnecessary energy, save power companies from building extra, undesired infrastructures to handle large peak hour energy use, and will lower the toll on the environment.

In order to create this interface we must start by implementing a device that will monitor the energy flowing from appliances. We used Kill-A-Watt, a device that produces analog readings of voltage and current on the screen of the device. This device is then connected to a switch that can turn the outlet on or off. The power of the Kill-A-Watt is converted from 5V to 3.3V by a voltage divider to power Xbee transmitters. We then created the two Xbee radio transmitters by soldering the appropriate wires in place and then soldered one of the transmitters onto the board of the Kill-A-Watt device. We assigned separate addresses to the receivers so that they could be identified uniquely, and we used XCTU to configure the Xbees.
Each Xbee transmitter was given an LED to indicate whether the device was activated and data was being sent across properly. The casing on the Kill-A-Watt device was adjusted so that the LED on the respective transmitter could protrude out of the device and be seen properly. The other Xbee transmitter we attached to a computer through a USB port so that the transmitter connected to the Kill-A-Watt could send the information to the other Xbee and display the information on a computer. This raw data is taken from the USB by using libusb python library. The data is then taken and organized so that it is easier for a consumer. We use the interface now to turn the switch on or off without physically needing to touch the switch. Finally the switch and Kill-A-Watt data are combined into a single application using python. Specifically WxPython is used to create a GUI that displays to the user the watt usage and allows the toggle of the switch. The graph displayed is continually updated and the live watt usage is displayed.

We plan on taking this small system and using it on every outlet in a home to allow for complete monitoring of the home. The system will turn appliances on or off based off of user scheduling and price fluctuations. The final step is taking our interface and creating a user-friendly mobile application and/or a central home management system to make the consumer’s life easier.
**INTRODUCTION**

Energy plays a fundamental role in shaping the human condition. People's need for energy is essential for survival, so it is not surprising that energy production and consumption are some of the most important activities of human life. Indeed, it has been argued that energy is the key "to the advance of civilization," that the evolution of human societies is dependent on the conversion of energy for human use. Energy conservation has become an issue today as the price of energy continues to increase. Many people wonder how we can conserve energy more efficiently than just remembering to turn off the lights or unplug our laptop charger when the battery is full. These actions, while reducing the amount of energy wasted, are not viable for large-scale conservation. If we do not start an energy conservation trend it will only drive prices up higher and higher. We depend on oil as the main resource for creating energy. This requires us to look to foreign countries and dig up the Earth’s natural resources. The more we use these resources the scarcer they will become causing energy to become low in supply.
The other problem with this continual energy usage is that the energy companies will not be able to continue to sell energy at constant fixed costs per hour. Energy will become more expensive during peak hours because of the extra demand on the generators during these high use hours. The solution to this epidemic can be to implement a smart grid. We can create a grid that uses smart applications and which can power down appliances that have consumed large amounts of power or not allow certain appliances to run if a certain user defined wattage limit was been met, while also helping users strategically plan to use their appliances during non-peak hours to conserve the amount of strain on the system. We can approach these smart applications by providing an interface that will track the wattage used from each outlet in a single home or office building and then inform the users of their power consumption which in turn enables them to automate certain processes that require large amounts of energy. With these new applications we can begin to cut down on the waste of energy and learn to conserve more and more. It will teach people to gather a real understanding of
what is going through their outlets and powering their devices. This knowledge for energy consumption will ultimately save the user money for wasted and unnecessary energy, save power companies from building extra, undesired infrastructures to handle large peak hour energy use, and will lower the toll on the environment.

**Hardware**

1. 3V pin - this is either an input power pin (if 5V is not provided) or an output from the 250mA regulator if 5V is provided

2. DTR - "Data terminal ready" this is a flow control pin used to tell the XBee that the microcontroller or computer host is ready to communicate.

3. RST - this pin can be used to reset the XBee. By default it is pulled high by the 10K resistor under the module. To reset, pull this pin low.'

4. Ground - common ground for power and signal
5. CTS - "Clear to Send" this is a flow control pin that can be used to determine if there is data in the XBee input buffer ready to be read

6. 5V - this is the power input pin into the 3.3V regulator. Provide up to 6V that will be linearly converted into 3.3V

7. RX - This is the XBee's serial receive pin. Serial data is sent on this pin into the XBee to be transmitted wirelessly

8. TX - This is the XBee's serial transmit pin. Serial data is sent on this pin out of the XBee, after it has been transmitted wirelessly from another module

9. RTS - "Ready to Send" this is a flow control pin that can be used to tell the XBee to signal that the computer or microcontroller needs a break from reading serial data.

10. see pin #1

Note: The DTR, RTS, RESET and RX pins (going into the XBee) pass through a level converter chip that brings the levels to 3.3V. You can use pretty much anywhere between 2.7 to 5.5V data to communicate with the XBee. The breakout pins on the bottom of the board are not level shifted and you should try to keep data going directly into the XBee pin sunder 3.3V

XBee comes from the family of form factor compatible radio modules. XBee 802.15.4 OEM RF modules are embedded solutions providing wireless end-point connectivity to devices. These modules use the IEEE 802.15.4 networking protocol
for fast point-to-multipoint or peer-to-peer networking. They are designed for high-throughput applications requiring low latency and predictable communication timing. XBee modules are ideal for low-power, low-cost applications. XBee-PRO modules are power-amplified versions of XBee modules for extended-range applications.” It could be used both as a transmitter and receiver and has many applications. XBee radio can be used at a minimum power of 3.3 V, ground, with bi-directional communication. It also has reset and sleep mode. Reset helps when the firmware crashes during updates or sometimes some settings or configurations fail the device. A reset could be used to help recover the firmware. Also, sleep mode in the device helps conserve energy when the device is not being used. Additionally, most XBee families have some other flow control, I/O, A/D and indicator lines built in. A version of the XBees called the programmable Xbee has an additional onboard processor for user’s code.

The XBees can operate either in a transparent data mode or in a packet-based application programming interface (API) mode. In the transparent mode, data coming into the Data IN (DIN) pin is directly transmitted over-the-air to the intended receiving radios without any modification. Incoming packets can either be directly addressed to one target (point-to-point) or broadcast to multiple targets (star). This mode is primarily used in instances where an existing protocol cannot tolerate changes to the data format. AT commands are used to control the radio’s settings. In API mode the data is wrapped in a packet structure that allows for
addressing, parameter setting and packet delivery feedback, including remote sensing and control of digital I/O and analog input pins.

Encryption is also another option in the XBee. XBee uses 128-bit AES encryption. Even though encryption is helpful to protect against security but it is not recommended because it slows down the throughput for the XBee.

<table>
<thead>
<tr>
<th>Throughput Encrypted</th>
<th>Not Encrypted</th>
</tr>
</thead>
<tbody>
<tr>
<td>47 kbps</td>
<td>80 kbps</td>
</tr>
</tbody>
</table>

Note: The XBee reference guide below was used to set up the transmitter and receiver.
**XBeep S2 Quick Reference Guide**

**Specifications**
- Operating Voltage: 2.1 – 3.6V
- Operating Current: 40mA @ 3.3V
- Indoor range: 40 Metres
- Operating temperature range: -20°C to 60°C
- Max Analog Pin Reading: 1.2V

**Digital I/O pins:**
- 11 pins
- 4 Analog input pins
- Mesh routable

**RF Data Rate:** 250kbps
- Throughput speed: 35kbps
- Frequency: ISM 2.4GHz
- OK Temp: -40 to 85°C

**XBeep Modes**
- **Transparent** – Communication through the XBeep. If data is not data from the XBeep itself then both XBeep should be set to AT.
- **Command** – Communication to the XBeep. If one XBeep is sensing data, that XBeep should be in AT mode while the receiving one should be in API mode.

**XBeep Setup**
- Connect the XBeep to a TTL Serial FTDI adapter – OR – Arduino hack Connect RX to TX, TX to RX, RESET to ground to bypass the Arduino entirely and get serial to XBeep.
- Use the free X-CTU software to configure the XBeep.

**Basic Settings**
- PAN ID – The network to communicate over. If 0, the XBeep will pin any.
- DHDLL – Destination Serial Number. Used to specify a specific XBeep’s Serial. Set to 0 to send to just the Coordinator. Set to 0x0000000000000000 to broadcast.
- Jv – Router/ESP should be set to 1 so it rejoins the network on startup.

**Pin Settings for work to receive XBeep must be in API mode**
- D0 – Set pin 0 to start sensing.
- IR – Collect data on sensing pins every XX millisecond.

**Byte Example Description**

<table>
<thead>
<tr>
<th>Byte</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0e</td>
<td>Start byte – Indicates beginning of data frame</td>
</tr>
<tr>
<td>0x00</td>
<td>Length – Number of bytes (ChecksumBytε - 1 – 2)</td>
</tr>
<tr>
<td>0x17</td>
<td>Frame type - 0x17 means this is a AT command Request</td>
</tr>
<tr>
<td>0x02</td>
<td>Frame ID – Command sequence number</td>
</tr>
<tr>
<td>0x00</td>
<td>64-bit Destination Address (Serial Number)</td>
</tr>
<tr>
<td>0x13</td>
<td>MSB is byte 5, LSB is byte 12</td>
</tr>
<tr>
<td>0x02</td>
<td>AT Command Name (Two ASCII characters)</td>
</tr>
<tr>
<td>0x0A</td>
<td>Command Parameter (queries if not present)</td>
</tr>
<tr>
<td>0x5F</td>
<td>Checksum</td>
</tr>
</tbody>
</table>

**API format for Remote AT Command Request**

- 0x04 (D) – AT Command Name (Two ASCII characters)
- 0x02 (2) – Command Parameter (queries if not present)
- 0x04 – Parameter (queries if not present)

**Arduino Connectivity**
- Arduino TX connects to XBeep RX (Data in)
- Arduino RX connects to XBeep TX (Data out)

**Arduino Integration**
- Data sent to Serial.print() will go to TX port of Arduino which is then connected to the RX port of XBeep. If XBeep is in AT mode it will transmit it wirelessly. Data received from XBeep will be sent to the Serial.

**Arduino Example: Read an analog value using API**

```c
// Remore XBeep: AT, Base XBeep: API
if (Serial.available() >= 21) {  // Make sure the frame is all there
  if (Serial.read() == 0x7e) {  // 7E is the start byte
    for (int i = 1; i < 15; i++) {  // Skip ahead to the analog data
      // discared byte = Serial.read();
      
      if (analogRead == 0x7b) {  // Read the first analog data
        serial.writeInt = (byte)Serial.read();
      } else {  // Read the second byte
        Serial.writeInt += (byte)Serial.read();
      }
    }
  }
}
```

**Arduino Example: Change the pin setting on a remote XBeep**

```c
// Remote XBeep: AT, Base XBeep: API
Serial.write(0x7e);  // Sync up the start byte
Serial.write(0x05);  // Length MSB (always 0)
Serial.write(0x10);  // Length LSB
Serial.write(0x17);  // 0x17 is the frame ID for sending an AT Command
Serial.write(0x0A);  // Frame ID (no reply needed)
Serial.write(0x00);  // Send the 64 bit destination address
Serial.write(0x0A);  // Send the 64 bit destination address
Serial.write(0x0F);  // Send the 64 bit destination address
Serial.write(0x05);  // Send D1 to be 5 (Digital Out HIGH)
```

**Pin I/O Options**

- **Sleep Mode**
  - Endpoints can sleep to save power. An endpoint that only wakes up every 5 minutes to send data may only be awake for 6 seconds a day.
  - SM – Cyclic sleep
  - SP – Sleep time (up to 28 secs)
  - SN – Number of sleep cycles
  - ST – Time awake

- **Pin I/O Options**
  - 0 – Disable
  - 1 – N/A
  - 2 – ADC
  - 3 – Digital IN
  - 4 – Digital OUT, LOW
  - 5 – Digital OUT, HIGH

**Pin I/O Data Sample RX indication**

- First Byte: n/a n/a n/a n/a D12 D11 D10 n/a n/a
- Second Byte: D7 D6 D5 D4 D3 D2 D1 D0
- Example: 0x00 0x13: 0000 0000 0000 1101
- Pins D3, D2 and D0

**API format for I/O Data Sample RX indication**

- First Byte: n/a n/a n/a A3 A2 A1 A0
- Example: 0x06 = 0000 0101 = Pin A2 and A0

**Digital Channel Mask**
- (volt) n/a n/a n/a A3 A2 A1 A0
- Example: 0x06 = 0000 0101 = Pin A2 and A0

**Checksum**
- 0xFF - the 8th bit sum of the bytes from byte 3 to this byte
USB NET POWER 8800

The USB Net Power is an economical device for network power control. With this device power control could not be easier. This piece of equipment is very easy to use. One can simply plug in the device, run the software and can turn on and off their devices after installing the software via PC. Also, with a scheduler function included this device can easily become automated to toggle anything that is connected to this.

Kill A Watt

Kill a watt is an electricity usage monitor which measures the energy used by appliances plugged directly into the meter. It displays voltage, current, real, and reactive power, power factor, energy consumed in kWh and hours connected.

The device that was used in the system is a p440.
Other components that were used to build the adapter board are:

3.3V linear voltage regulator, 250 mA current
5v Compliant buffer chip
LEDs
2 types of resistors were used. 1K and 10k resistors
0.1uF ceramic capacitor
47 uF
Straight male header
10-position 2mm female header, sockets for XBee modem
PCB circuit board

**Cost of the System**

<table>
<thead>
<tr>
<th>Item</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xbee module series 1-</td>
<td>22.95</td>
</tr>
<tr>
<td>Xbee adapter kit-</td>
<td>10.00</td>
</tr>
<tr>
<td>USB Net Power 8800-</td>
<td>25.00</td>
</tr>
<tr>
<td>Kill A Watt-</td>
<td>26.99</td>
</tr>
</tbody>
</table>

---------------------------------------------
Total-                                84.94
The total price would be a one-time installation fee per outlet. Of course, if these 4 technologies were integrated into a single unit the price would surely go down. The consumer would even have the option to only install the system on specific outlets, such as ones that run expensive appliances.

For a cost savings scenario, please consider the following chart:

<table>
<thead>
<tr>
<th>Appliance</th>
<th>Cost per Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cooling</td>
<td>$394</td>
</tr>
<tr>
<td>Water Heater</td>
<td>$317</td>
</tr>
<tr>
<td>Dryer/Washer</td>
<td>$143</td>
</tr>
<tr>
<td>Total</td>
<td>$854</td>
</tr>
</tbody>
</table>

http://www.dailyfinance.com/2010/06/10/10-most-costly-appliances/

Price fluctuations are still to be determined, but even if the price fluctuates by only 5% and the appliances is run at this time instead of peak hours, the annual savings is $42.7. This results in the system paying for itself in only two years! Not, only will the consumer begin experiencing direct financial savings after two years, they will also be provided data of their consumption. This doesn’t include additional savings a consumer might gain due to the knowledge they acquire from the system (that ultimately changes their behavior further).

**Implementation of the System**

The approach of the energy smart home system was to use some sort of wireless communication to extract the data from a measurement device which is
connected to a switch. This led us to use the Xbees, Kill A Watt, and a USB Power net. After further researching it was determined that we could program the USB switch to turn on / off in respect to the data that was being received from the XBee.

**How it was done**

First we built the XBee adapter using a PCB board to connect the receiver and transmitter with specific components. One of the XBee’s is configured as a transmitter and the other as receiver. The transmitter was connected to the PIN1 & PIN2 of the data output of the Kill A Watt device. This extracted data and transferred it to the receiver. On the receiver side, this XBee was connected to using a USB to the PC. Which was later used in form of an array to program in WxPython. Data was organized in Python to be represented in graphs which helped the user analyze the power usage. After analyzing the data we retrieved from the receiver, the USB Net Power was programmed to toggle the switch. The USB Net Power is connected to the PC via USB, and in its outlet, the Kill A Watt is plugged in. By analyzing data in Python, the user or the system could decide when to toggle the switch on or off depending on the desired power usage. By this we can save energy and save money on our monthly bill, and also move towards a greener society.

**SOFTWARE**

The software used for this project was broken into two files. One controls the Kill-A-Watt, and the other controls the usb net power. Both of these programs were
written in python. Python is a very expandable high level programming language. This language was the best choice because Python supports multiple programming paradigms, including object-oriented, imperative and functional programming or procedural styles. The other major factor is the large amount of libraries that can be used with python. These libraries are what makes python the most useful for our project.

**Libraries**

The first library we used was pySerial this module encapsulates the access for the serial port. This module provides us with the ability to open up the FTDI serial port to get data transmitted to xbee. The com/serial port the XBee is connected to is COM4. This library provides us some additional functionality such as setting up the port for binary transmission with no NULL byte stripping, and CR-LF translation which assisted in the Xbee connection process.

The Xbee product library was one of the main libraries utilized in our project. This library was used because it allows for easy access to the advanced features of an XBee device from a Python application. It provides us with a semi-complete implementation of the XBee binary API protocol and allows us to send and receive the information they desire without dealing with the raw communication details.

The WxPython library was important in the creation of a GUI and simplification of data. WxPython is a GUI toolkit for the Python and it allowed us to create a robust project with a highly functional graphical user interface. It is
implemented as a Python extension module that wraps the popular wxWidgets cross platform GUI library, which is written in C++. This was used in order to create and update the dynamic graphs.

The NumPy library was used because it is the fundamental package for scientific computing with Python. Besides the use as a calculator for our project, NumPy was also used as an efficient multi-dimensional container of generic data and was integrated very easily.

The Matplotlib library was our method of plotting the incoming data. Matplotlib is a python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. This in combination with NumPy we were able to produce a dynamic graph for Power, Current, and Voltage.

The Pysub library was used to allow us to connect with the usb from our program. This utilizes C programming to connect with a the USB port, but is built as a cross platform library for python users. This was the main library that is used for all USB connections and data gathering. This allowed us to connect two USB and gather data separately. This was also used to send commands to the device connected to it.

`usbnetpower880.py`

This python program is used to control the usbnetpower switch from the command line. The first thing we do is import sys and usb.core, these are the only
libraries we need for this program. The power class is how we initialize with the USB. We use the vendor id, 0x067b, and the product id, 0x2303, to find the proper usb device. This was easy to set up and has good reliability. The state changes of the switch are commands that are sent through the usb. Using online documentation from usbnetpower we found the proper codes to toggle the switch. The inputed arguments when running the code in the terminal are how the user can toggle the switch.

This program was very concise and reliable. During testing the removal of many complexities and utilization of the libraries made for a program that remained in full control of the device. This program was the easiest way to control the usb without using the software given by the company. In future designs this has good versatility and can be easily moved into a larger GUI.

wattcher-wattgraph.py

Wattcher was a program that was used to draw dynamic graphs from the data gathered through the xbee. This program utilized many libraries, including pySerial, Xbee, wxPython, NumPy, and Matplotlib that were all described above. These made for a simple and reliable design that can easily display all of the necessary information. Fortunately the use of these libraries did not hinder our customizability. Many parameters were taken from tech specs of the hardware we used, including serial ports, baud rate and others. We used the serial port in order
to access the xbee. This allowed us to go through the USB and get the data being transmitted from the kill a watt.

After the data is gathered it is temporarily stored within the program. Using NumPy we are able to do scientific calculations on the incoming data. Within each sample we do calculation for minimum and maximum, voltage and current. This allows for the power to be precisely calculated for each sample. We use a very small sampling rate in order to have the most intuitive dynamic graph.

The graph itself is done using the matplotlib library. Two graphs are created one for voltage and current and another for power. The graphs use the data calculated, described above, and Plots a new graph at each sample. When the graph reaches the end of the X axis or maximum number of samples. It replaces the current Maximum number of samples x axis with a new one, and retains all of the previous data and adds in the new sample. This increases the maximum samples by 10 and this happens for as long as the program is running. The GUI was created using WxPython. This is a library primarily design for designs that require small amounts of display. It made it easy to keep the graph dynamic with a simple GUI page reload.

In conclusion this program worked with great reliability and displayed data the same as the data read directly from the device. The program utilizes the idea that a simple GUI with complex data is the best way to display information. This
software was designed for the use of one device and can easily be expanded to handle more devices.

**Future Work**

After completing our project, all the researches we have done expended our goal to be put on the system by next the generations to make the system the most competitive and best solution to monitor and automate energy consumption in such facilities.

To be more precise and manageable on power consumption the entire system can be integrated into every outlet of the home to offer complete energy monitoring. The system we built has very low cost, and it can easily be impended to all the outlets at home. Once the network starts getting the data from each of the devices, we will have a knowledge of the overall power usage and the system can turn on/off appliances based off user scheduling and price fluctuations.

In the description of hardware, we talked about this device called “USB NET Power” which is used as a switch to turn on and off the outlet that is being used. So this device is connected to the network we built on Python thru a USB cable sending information to manage the outlet on/off conditions. In the future, we can also mount Xbee chips into the switch as we did for kill a watt to send information wirelessly.

As we manage to send commands wirelessly to the switch, imagine having a controller that communicates with a fully integrated device placed on each outlet.
This controller can be an application to be accessed from a smart device and/or a central home management interface using various software programs and cloud technology.

The XBee Platform is fully integrated with the “Device Cloud” by Etherios. This allows developers such as us to create end-to-end solutions even if you are far away from the network you are working with. It also enables developers to create solutions with strong value propositions by bringing them to market in a cost-effective and timely manner.

One other thing can be implemented in our design is the solar inverters to decrease the need of power from suppliers during peak hours. The energy can be drawn from solar panels can be used to keep on the appliances that we need during peak hours, and other times where this energy can be stored in external batteries to be used when needed.
Mentioning about automated systems, one thing can be helpful is sensor. Having light and heat sensors in the network will get ready the house before the user come back from work. Such examples can be turning the lights on at sunset, switching heat and A/C on and off based on the temperatures and user scheduling organizing the blinds to protect the sun coming through, etc.

**CONCLUSION**

In conclusion, energy providers are pushing towards a greener future. Much has been done to improve technologies on the production end. Now it is time improve it on the distribution end. A smart grid is the first step forward. The smart grid relies on the cooperation of the individual consumers to use energy at appropriate times and in appropriate amounts. In order for this to be possible, the consumer most not only be provided financial incentive, but a system in which the can monitor their consumption. This is where the smart home comes into play.
In short, the system consists of sensors at each desired outlet. The outlets then communicate to a central system. Software makes sense of this data and allows the user to monitor their system and schedule loads. Ultimately, energy providers would be able provide energy more efficiently and reliably, individuals can save money, and the toll on the environment is lessened.