Energy Mobility Network: System Design, Interfaces, and Future Interactions

by

Natalie Wen Yua Cheung
B.S. Electrical Engineering and Computer Science
Massachusetts Institute of Technology, 2009

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science
at the
Massachusetts Institute of Technology

February 2011

Signature of Author: [illegible]
Department of Electrical Engineering and Computer Science
February 1, 2011

Certified by: [illegible]
Dr. Federico Casalegno
Director of the MIT Mobile Experience Lab
Thesis Supervisor

Accepted by: [illegible]
Dr. Christopher J. Terman
Chairman, Masters of Engineering Thesis Committee

Energy Mobility Network:
System Design, Interfaces, and Future Interactions
by
Natalie Wen Yua Cheung

Submitted to the
Department of Electrical Engineering and Computer Science

February 1, 2011

In Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

The Energy Mobility Network is a mobile, networked energy production, consumption and sharing system that is designed to motivate users to be more aware of their energy consumption. In particular, the system provides a just-in-time message to the user *before* using the device, which allows the user to evaluate his/her needs and the cost of the device. Furthermore, the idea of minimizing electrical costs are extended into the social realm; the system creates a social network among users which allow social energy etiquettes to come into play. With these etiquettes, the system aims to use social means as a way to minimize the use of electricity. In the thesis, I discuss the goals and ideas developed that led to the creation of the network and the technical infrastructure behind the system. I will be going in depth with the prototyping, the pros and cons, as well as the multiple versions of the system that have been prototyped. Finally, I will discuss the future possibilities the Energy Mobility Network will bring when introduced to the general public.

Thesis Supervisor: Dr. Federico Casalegno
Title: Director, MIT Mobile Experience Lab

Acknowledgements

I would like to express my sincere gratitude to Dr. Federico Casalegno, who has given me the opportunity to work at the Mobile Experience Lab. He allowed me to work in an interdisciplinary design team, an opportunity which broadened my perspectives and gave me better insight to architecture and design.

I would also like to express my deepest thanks to Orkan Telhan. It was Orkan who suggested that I should take on this project, and the progress of this project would not have happened without his advice and support. I sincerely appreciate the time he made to sit down and discuss ideas and questions I had no matter how trivial they were. His constant guidance, understanding, and mentorship have allowed me to be more confident with my work.

Many thanks to David Boardman, Gaia Scagnetti, Orkan Telhan, Ivo Wouters, and Huei-Sheng (Carl) Yu for their work on the Energy Mobility Network. Without their ideas, support, and work, I would not have a cohesive prototype or the idea of the Energy Mobility Network to present!

My appreciation goes to our sponsor, Fondazione Bruno Kessler, for collaborating with MEL to create the Green Home Alliance. Without this, the inception of the project would not have happened.

Finally, I would like to express my gratitude to Myoung Lah, Willy Cheung and my parents. Their constant encouragement and support have always been appreciated.

Table of Contents

List of Figures

List of Tables

1. Introduction

In today's society, a common practice is to turn on an appliance and leave it running, with little to no thought about energy consumption. Current energy management methods, which are discussed in Section 2, aim to improve this problem as well as the user's quality of life. However, these practices are too restrictive and end up penalizing users. Users receive a summary of their electricity consumption along with a hefty bill, a bill that does not notify them of what they can do to minimize costs nor provides real-time information about how much electricity each device uses. Consequently, users who are "forced" to save energy often end up sacrificing their convenience for energy conservation -- such behavior is not the goal. Instead, users should be provided with more information about how much electricity is consumed and allowed to act upon this given information. By doing so, we are allowing users to have more freedom as well as giving them the option to save energy when deemed possible by the user. With this in mind, the Energy Mobility Network system was created.

The Energy Mobility Network project was incepted as a smaller project of the Green Home Alliance, an interactive house in Trento, Italy. The system aims to allow users to track their energy production and consumption usages as well as to turn on and off devices remotely. In addition to providing just-in-time prompt to users [7] [8], the network also aspires to provide social interactions among users which indirectly produces a new venue of social etiquettes in our society.

The purpose of creating a new system is two-fold. We realized that the general population's only connection with the amount of electricity consumed was through the monthly gas statement sent to their homes. Receiving information every thirty days is not up to par with today's technology. We wanted to create a system where users would be given data about the electricity usages instantaneously. Even more, we wanted users to be more aware of their electricity consumption with the goal that over time, their energy footprint would attenuate.

Based on the current products in the market, there is an untapped field – there are no products which give users instantaneous feedback about how much energy a product will use before the user uses the device. Moreover, current off-the-shelf products do not encourage social interactions among users. Instead, these products focus more on the individual and how to minimize electricity consumptions as an individual. The Energy Mobility Network strives to address this on an individual and community level by using social interaction as a motivation to improve the electrical footprint. The system solves these problems and produces an easy to use product that will be able to enlighten users about their electricity consumption and production. This is similar to "the house as a personal trainer" paradigm.

My role in the Energy Mobility Network was to develop specific features and to create a working prototype for the project. This entailed creating software for the user interface as well as the technical hardware prototype of the outlet. The motivation for these prototypes are to demonstrate how the user can control devices remotely and utilize this infrastructure to improve their energy consumption behavior. Furthermore, this system provides grounds for a social network that creates social etiquettes among users.

This thesis report will go through background material on what other outlet systems are available and how this project came to fruition. I will then provide a technical overview of the project as well as a thorough description of how I went about fabricating the hardware and software design. I will describe the tests that I made with the hardware to produce a final prototype. The results of the prototype will be explained as well as future recommendations to improve the network.

2. Background Research

The background research was a joint collaboration among David Boardman, Gaia Scagnetti, and myself.

2.1 Timeline of Products Detailing Electricity Consumptions

Throughout the past years, there have been alternative ways to view electricity consumptions as noted in Figure 1.

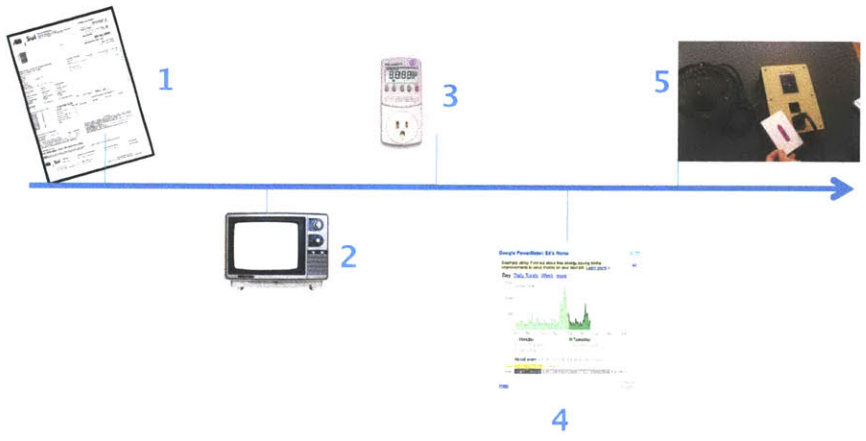


Figure 1. Timeline of Electricity Feedback and Means to Save Electricity[1]

This figure depicts five different ways which users receive data pertaining to their consumptions. The most common way is through the monthly paper electric bill as seen as the first icon. The bill gives monthly feedback on electrical costs in a home. It does not label what specific devices were used or who used those appliances.

The second icon in the figure is a television. Here, the television symbolizes collective awareness. The general public realizes that much of the electricity costs could be minimized as a whole by turning off devices when not in use. For instance, if televisions were not left on and promptly turned off, electricity costs could be decreased. This can be further extended to devices

[1] Figure created by the MEL Energy Mobility Network Design Team

that remain plugged into outlets. While the appliances are off, these vampire devices still consume electricity. Studies have shown that the typical American home has twenty electrical vampire devices that waste electricity and increases the yearly electric bill by $200 [2].

More recently, gadgets have been created that will produce instant feedback about electricity costs after the device has been used. These appliances, seen as the third icon in Figure 1, aim to create awareness about electrical consumptions.

Alongside the hardware appliances, there are software systems which monitor energy expenditures at home. As seen as the fourth icon, these systems work hand in hand with the hardware energy-saving gadgets, to give the user a greater sense of what electricity consumptions have occurred.

Finally, there is the Smart Interface, which is detailed in Section 5. This prototype is the first attempt in creating a system to aid the user with minimizing electrical consumptions. This timeline shows a brief history of what has been done in the past to curb electricity costs.

2.2 Related Work

Many commercial products display electrical consumption data. Each differ in the manner which the information is presented and the number of devices monitored by each product. For instance, Kill A Watt Power Meter, in Figure 2, only monitors one device.

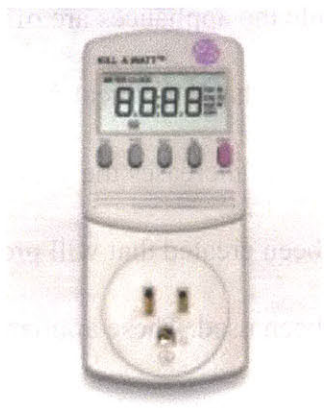


Figure 2. Kill A Watt Power Meter [13]

The LCD display shows metered readings in the form of volts, current, watts, frequency, power factor and VA [14].

A less technical display can be seen in Figure 3, the PowerTab In-Home Display from Energy Aware.

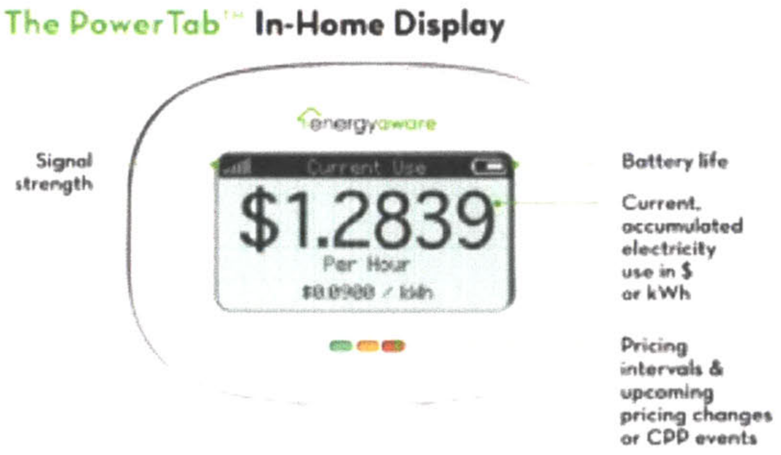

The PowerTab™ In-Home Display
energyaware
Signal strength
Current Use
$1.2839
Per Hour
$0.0900 / kWh
Battery life
Current, accumulated electricity use in $ or kWh
Pricing intervals & upcoming pricing changes or CPP events

Figure 3. User Display of the PowerTab Product [4]

The wireless product updates the user with upcoming pricing changes through passive means -- different colored light-emitting diodes. This product shows the total consumption in the household, instead of focusing on one specific appliance. It specifies the rate in terms of monetary value or in a more technical sense, kilowatts per hour [4].

Another wireless product, Plogg in Figure 4, creates a wireless energy management system for the user.

Figure 4. Plogg Appliance [5]

This product can act as both a stand-alone gadget to detect energy consumptions of a single device as well as an "integrated energy reporting network" [5]. The network can produce wireless information to the user via the Plogg Manager software to present technical graphs pertaining to electrical consumptions.

The Wattson products illuminate the cost, watts, carbon costs, energy use data at the top of the product. The bottom of the Wattson, as seen in Figure 5, glows passive lights which reflect the relative energy usage in the home.

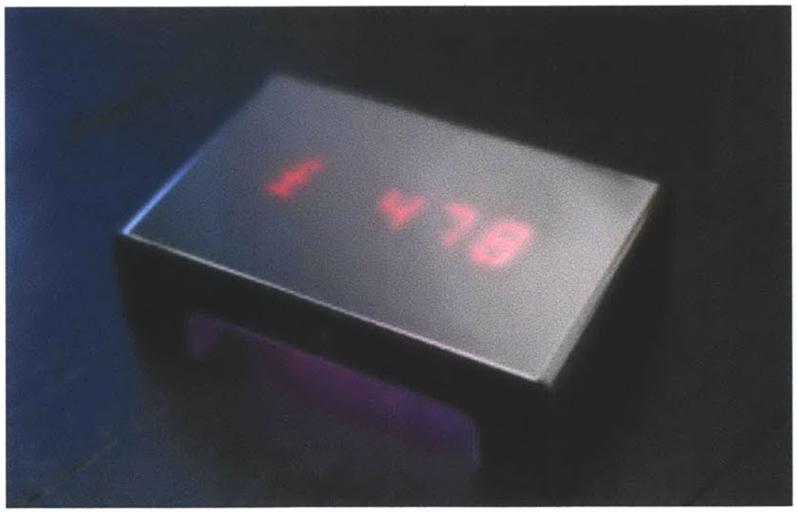


Figure 5. Wattson Product [15]

By downloading the Wattson software, the user is able to log his/her energy consumption for the home. Furthermore, this connects the user to the online Wattson community where the company calculates how many Wattsons have been linked in the community and as a whole, how much energy and money has been consumed.

There are other products available, which display electrical consumptions in graphical form, such as the Google PowerMeter, a software that analyzes your electricity consumptions and customizes the interface to fit the user's needs [10]. Even with these multiple products available, there were a few underlying problems with the appliances researched which spurred the Mobile Experience Lab, in particular: David Boardman, Gaia Scagnetti, Orkan Telhan, Carl Yu, and myself, to create the Energy Mobility Network.

3. Inception of Energy Mobility Network

The system we aimed to create should achieve three main goals. First, before the user turns on a device, the system will present just-in-time information about the current costs of using the device. It will also present a suggested time where the cost of using the device will be cheaper. By doing so, the user can make an informed decision about how much electricity will be expended and if it is the best time to use the device based on his/her needs and costs. Using just-in-time messaging gives the user full control of the devices, but does not force the user to reduce energy consumptions.

The second goal is to shift from a device centric way of thinking to more of a human and system centric method. Instead of focusing on how much electricity each device consumes, we want to focus on the amount of electricity each user consumes. We plan to achieve this by having a gadget that will notify the user of which devices are currently on, the total electricity consumption and production costs, and more.

Our final goal is to change people's perceptions about electricity. Electricity should be a social responsibility instead of an individual responsibility. With these three goals in mind, the system outline and the technical specifications for the project were designed.

4. Design of the Energy Mobility Network

The Energy Mobility Network design is shown in Figure 6. Each user interacts with the system through a gadget we have coined as the Energy ID. The Energy ID is used to determine a profile for the user. The outlet, which powers the device, is connected to the main power supply. Before explaining how the whole system works, each block is described in detail.

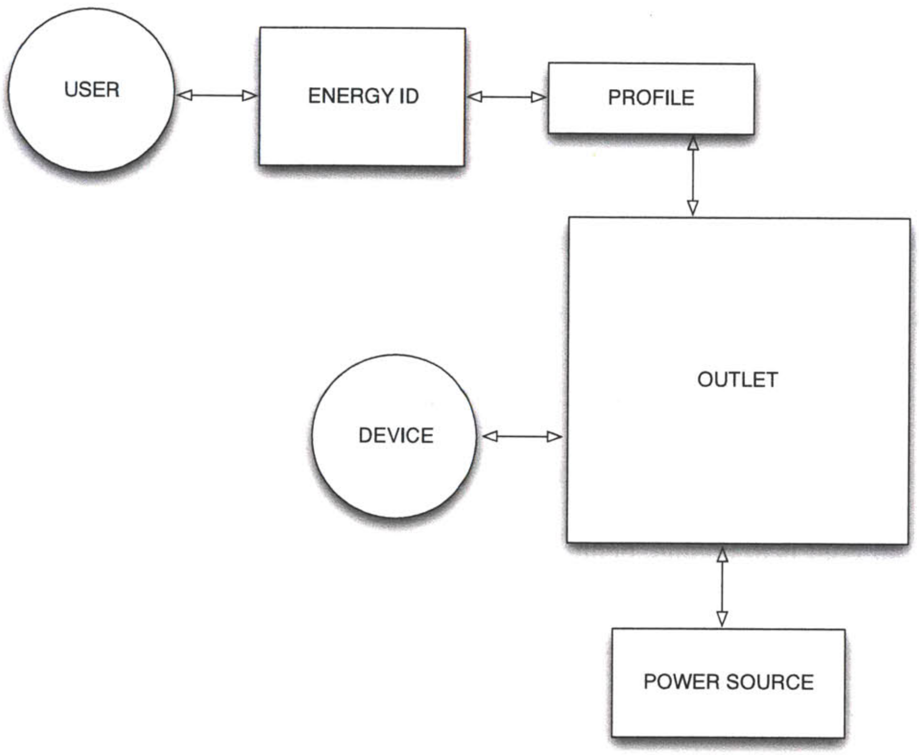


Figure 6. The Energy Mobility Network Block Diagram

4.1 Individual Components

4.1.1 User

The Energy Mobility Network is tailored towards all types of users. All the information is communicated to the user in a "language" that the user can understand. Users who are more tech savvy will see data described in terms of kilowatts. Fiscally invested users will see the amount of electricity consumed in dollars. The units in which we display the quantity will depend on the user -- from a simple progress bar to monetary value to kilowatts. This idea of tailoring the information to the user's interests in an easy to understand method is also abstracted out to the way the interface of the Energy ID and the outlet are presented.

4.1.2 Energy ID

The Energy ID is an integral part of the system. The Energy ID displays the user's information, electricity expenditures, and a list of devices that are currently in use by the user. Additionally, the Energy ID provides the freedom to remotely control devices already plugged into the outlets. It displays all necessary information in one place.

There are two forms of Energy ID: active or passive. Active Energy IDs are geared towards users who want as much information as possible about their energy expenditures, devices, and social network given to them. The active users would have Energy IDs embodied in an electrical device such as an application on a mobile phone. Passive users, users who only want to know how much electricity they have consumed and produced, would have a less tech savvy Energy ID. The Energy ID could be found in a key fob, credit card, watch, and more. The passive Energy ID will show a status bar of how much electricity they have consumed or produced, but not detailed information such as which devices are on or who is in their network -- this information will instead be found on the outlet.

The Energy ID will be embodied in different objects to be more customizable and more socially accepted. For instance, the tech savvy user may prefer the Energy ID information on a mobile device whereas a family planner may prefer the information on a watch. A teenager might find his/her Energy ID device on a key fob. All the Energy ID embodiments can be further customized such that the tech savvy user may use an interface which offers tools to regulate devices or graphs the fluctuations in consumptions and costs. A family planner Energy ID might contain information about current costs and consumptions for the devices in use and just-in-time messages whereas a teenager's Energy ID may contain information about his/her social network and his/her electrical consumptions at that moment.

Every Energy ID will contain an unique identification tag that is linked to the user's name. This tag aids the system with identifying what devices the user has activated.

4.1.3 Profile

Based on the context of the situation, each user carries out different identities. In today's society, identities are fluid; the user has an endless list of identities: MIT student, MIT Mobile Experience Lab research assistant, friend of a friend, guest at a friend's home, Whole Food's client, MBTA commuter, etc. Each fluid identity can be linked to a profile. For instance, if the user is at school, the profile would be "student". If the user is at a friend's home, the profile would be "guest". The profile identifies the fluid identity of the user and is a factor in determining the cost of using a device.

4.1.4 Device

Devices can be consumers or producers. A device is a consumer if it needs electricity to be powered. Such devices could be a cell phone charger, television, or fan. A device is a producer if electricity can be created. Said another way, it does not need electricity from the energy grid to be powered. Such devices might include a treadmill or a hand-operated flashlight.

Each device will contain a RFID tag that will be sent to the system when the device is plugged into an outlet.

4.1.5 Outlet

Devices need to be plugged into the outlet to be used. The outlet is responsible for providing or withholding electricity to the devices that are plugged in. Furthermore, the outlet receives and transfers information to the database detailing which devices are on, the Energy IDs that are connected to the outlet, and more.

For passive users, the outlet will contain a screen that will presents detailed data to the user. Such information may include the ability to turn on or off devices connected to other outlets, viewing the predicted cost patterns for the next two hours, or checking who is in the user's social circle.

Each outlet contains an RFID sensor as well as a means to wirelessly communicate to the server and other outlets.

4.1.6 Power Source

There will be two ways that a device can be powered. The conventional way is to use the electrical grid provided by the local supplier. The other possibility is to source the electricity from the power that the Connected Home has produced. In this thesis, we will focus on the conventional electrical grid.

4.2 Functionalities of the Design

There are a few functionalities that are provided with the system which are detailed below.

4.2.1 Turning On a Device

When the user wants to turn on a device for use, he/she will tap his/her Energy ID into the outlet and plug in the device. The outlet will recognize the user, his/her profile based on the context, and what device was plugged in. The outlet will present a just-in-time projection for using the device for an hour based on the current load rate. It will also present a future time that will have a cheaper cost, if possible. With this information, the user has the choice of turning the device on now or later. If he/she chooses to use the device at the current moment, the outlet will allow electricity to flow into the device and will tell the server that the device is in use. However, if he/she chooses to use it at another time, the outlet will not allow any electricity to flow through to the device.

Remotely turning on a device works in a similar way. The user will select the device that he/she wants to use and the outlet closest to the user will send this information to the server. The server will then send a command to turn on the device connected to the other outlet.

4.2.2 Viewing Information about Electricity Consumptions

When the user connects to the network, either by tapping his/her Energy ID to the outlet or turning on the wireless connection to an outlet, the outlet will grab the latest information about his/her total electricity consumption and production for the day. It will also find what devices are currently in use, who is in the user's network at the moment, and his/her current profile. All this information will be displayed either on the Energy ID or on the outlet's screen, depending on if it is a active or passive user.

4.2.3 Calculating the Cost of a Single User Using the Device

Calculating the cost of the device being used will be different from the method done in today's society. The system will calculate the cost based on a few factors: the amount of time used, the amount of electricity being consumed, as well as the user's profile. Energy charges are dependent upon the context in which someone uses a device. For example, as a student, the user might be charged only 50% of the energy cost (the school would cover the other 50%), where as at home, the user is the owner of the house and is assessed 100% of the energy costs.

4.2.4 Multiple Users using the Same Device

If multiple users want to use one device, the cost of the device will be shared among the users. All the users must tap into the network and be connected to the outlet which powers the device. One person will tap in the device and when selecting to use the device, the outlet will ask if the cost should be divided among the users. If the cost is shared among the users, each user's Energy ID will now show that the device is being used but at the divided cost. If the cost is not shared, the system will calculate the cost as a single user using the device.

4.2.5 Social Gestures Among Users

Social acceptance and rejection is a method of motivation which is integral in today's generation due to the digital world that we live in. People are now more motivated to act a particular way to gain social approval among their friends and family and is a means of survival to some [9]. Consequently, another feature that the Energy ID provides is the ability to participate in social energy gestures. Because the costs of electricity are now on an individual basis, users can treat friends and family to "free" electricity as a social gesture. For instance, a group of friends who watch TV together weekly, may take turns in sharing the cost of electricity. This concept is similar to the way a person buys a round of beer for his/her friends. By doing so, a new realm of etiquettes have been created. Such gestures can enhance relationships among users. These social gestures can gradually establish implicit norms, similar to the way dress codes are accepted, which will regulate individuals' behavior.

4.2.6 Zero Sum Balance

Each user will start off with a zero sum balance. If the user uses a device that consumes electricity, the total amount of electricity will be deducted from the user's balance. However, if the user engages with a production device, the amount will be added to the user's balance. An environmentally friendly user would want to keep a zero or positive balance.

Based on the system design, a very basic first prototype was created to demonstrate the idea of just-in-time projection to users. The next section details the technical specifications and work done to create the first prototype.

5. Smart Interface

The first prototype for the project, then called Smart Interface, was headed by Orkan Telhan. The goal of the Smart Interface was to remind the user that his/her actions could directly impact the environment. This was done by creating a new "outlet" which would provide just-in-time prompts to the user as seen in Figure 7.

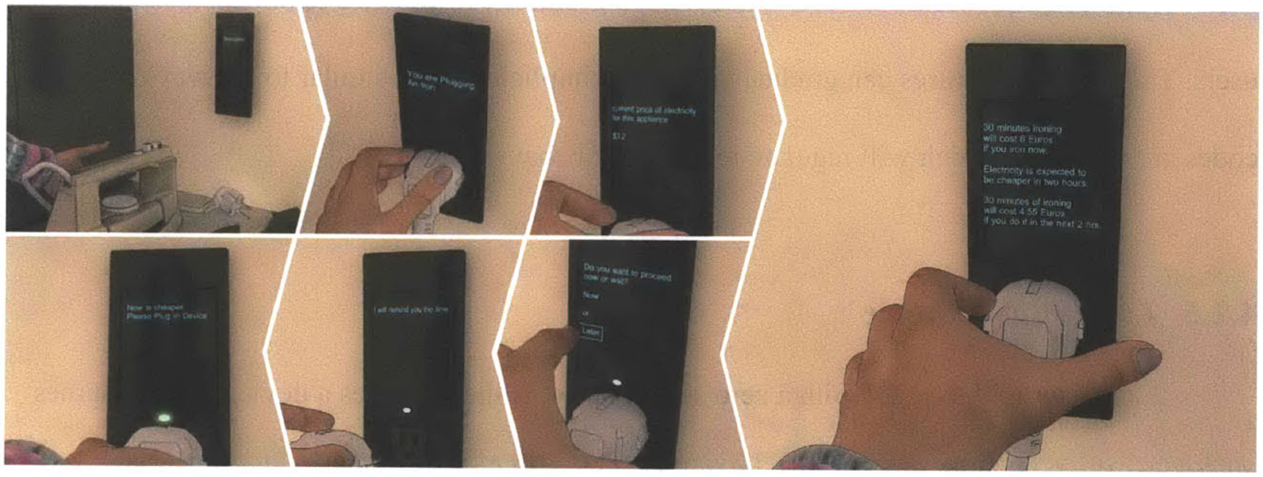


Figure 7. Step-by-Step Process of using a Device[2]

5.1 Sketch of Smart Interface

Instead of just plugging in a device and turning it on for use, the user would have to follow a few steps in order to turn on and to use the device:

1. The user plugs in the device (in this instance, an iron) into the Smart Interface.
2. The Smart Interface recognizes that an iron has been plugged in: "You are plugging in the Iron"

[2] Figure created by the MEL Energy Mobility Network Design Team

3. The Smart Interface displays the current price of electricity for the appliance: "current price of electricity for this appliance: $3.2"
4. The Smart Interface then calculates the price of using the iron now and at a cheaper time: "30 minutes ironing will cost 6 Euros if you iron now. Electricity is expected to be cheaper in two hours. 30 minutes of ironing will cost 4.55 Euros if you do it in the next 2 hrs."
5. After displaying the just-in-time prompt, the Smart Interface asks what you have decided. If the user chooses to use it now, the power to the iron is turned on. However, as is the case in the sketch, the user opts to use it at a later time: "Do you want to proceed now or wait?"
6. The Smart Interface takes note that the user will use it at the later time: "I will remind the time"
7. At the cheaper time, a light turns on to remind the user: "Now is cheaper. Please Plug in Device."

By following these steps to use the device, the user is given more options. Based on his/her needs, he/she can use the device right away and pay the current fee or wait until the peak load has diminished and use the device at a cheaper rate. This is particularly useful for situations where the use of the device is not time sensitive. For instance, if a user needs to wash the dishes before the next meal, the user can just wait to turn on the washing machine at a later time instead of at the peak load. This "cheaper" time could be when the user is asleep at 2 AM and the washing machine can be set to turn on at that time. If the user has a time-sensitive urgency to use a device, the user still has the option to use the device at the more expensive time. Giving the user the option as well as just-in-time prompts about his/her energy expenditures is an idea which helps the user understand that his/her behavior can help him save money and be energy conscious. The Smart Interface would be built into the Connected Home Project and replace the electrical outlets that are found on the wall, similar to Figure 7 shown above.

5.2 Hardware and Software Design

The Smart Interface was built with off the shelf components that are found on Sparkfun. It is comprised of four main components: the Linksprite Powerline Smart Outlet Control, the Linksprite Powerline Smart Outlet, the Active Matrix OLED with Touchscreen attached to the RAI board and a RFID Reader ID-20 sensor. It should be noted that the touchscreen is part of another project, RAI, in the Mobile Experience Lab. The entire RAI board was used within the Smart Interface, but solely for the purpose of using the OLED Touchscreen functionality. As seen in Table 1, the components were powered and connected to the computer through different means:

Component	Powered By	Connected Through
Smart Outlet Controller	Unregulated AC power	USB
Smart Outlet	Unregulated AC power	Power line
Touchscreen	Battery	Bluetooth
RFID sensor	USB	USB

Table 1. How Each Smart Interface Component is Connected and Powered

The integration and hardware design of the Smart Interface is very simplistic. Each component was isolated and tested on the computer so that the functionality of the component could be easily analyzed. Python code was then created to integrate the Smart Outlet and Controller with the RFID sensor. Previous code written from the RAI project was used to create the display of the touchscreen. By isolating the touchscreen code from the rest of the Smart

Interface code, it allowed for a quick prototype to be built and displayed for demonstration as seen in Figure 8.

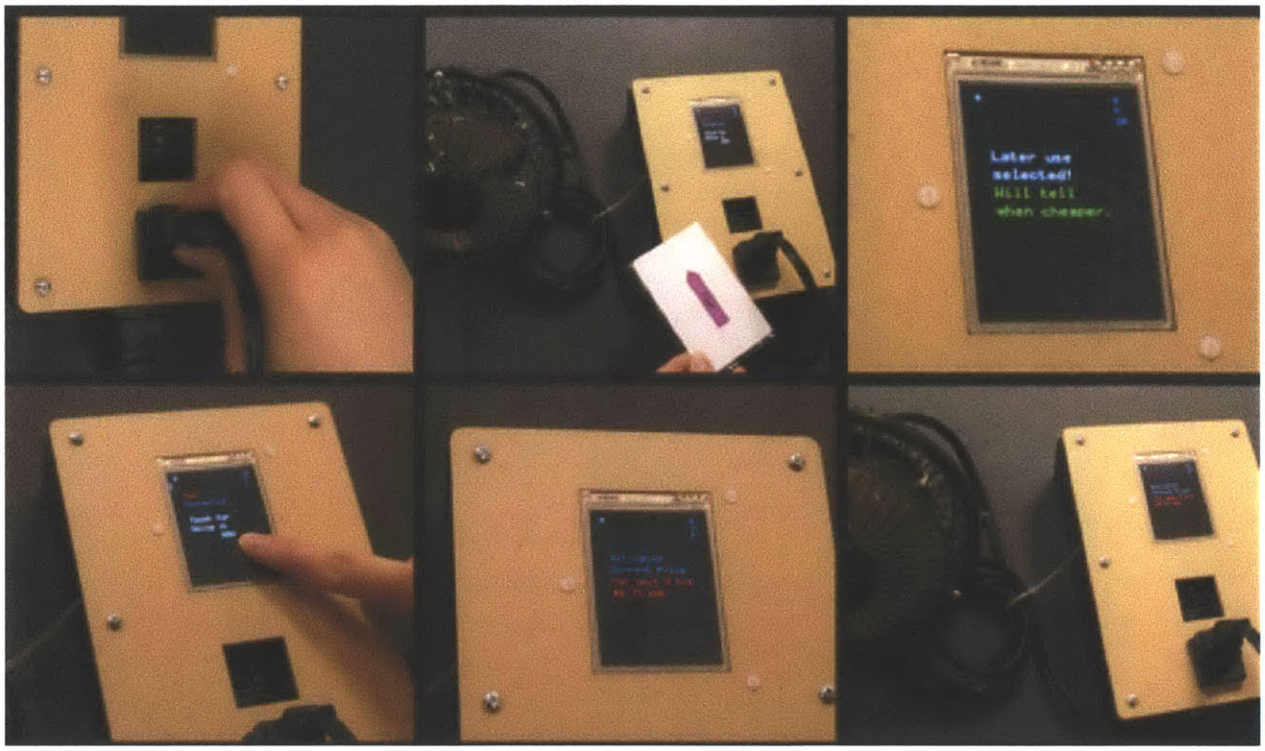
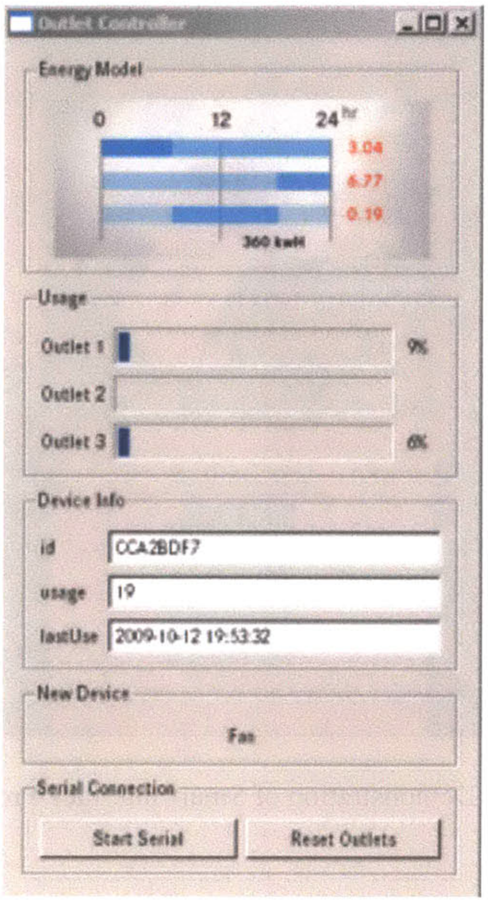

Later use selected!
Will tell when cheaper.

Figure 8. Demonstration of Smart Interface Prototype[3]

The Smart Interface Prototype is very similar to the sketch of the concept seen in Figure 6. After the user plugs in the device, the user taps a RFID tag onto the Smart Interface, which recognizes the device. It asks the user to touch the screen if the device should be turned on at that instant. If the user touches the screen, the device is then powered and the screen shows the activated current price for two hours. However, if the user does not touch the screen within a certain amount of time, the Smart Interface recognizes that the user opted to power the device at

[3] Figure created by the MEL Energy Mobility Network Design Team

a later time. In addition to the working prototype, a GUI, as seen in Figure 9, was created to show the technical details of the project.



id: CCA2BDF7
usage: 19
lastUse: 2009-10-12 19:53:32

Figure 9. GUI for the Smart Interface[4]

The GUI is used throughout the whole demonstration of the Smart Interface. It is comprised of four widgets: Energy Model, Usage, Device Information, and Serial Connection. The Energy Model displays the cost of using each outlet within a 24 hour span. The various colors depict different pricing rates for each outlet. The usage box displays how often each outlet has been used throughout the day. It allows the user to see the frequency with which the outlet

[4] Figure created by the MEL Energy Mobility Network Design Team

has been used. The Device Information presents the specific ID tag of the new device that has been recognized. It also includes when the device was last used and how much it would cost to use the device per hour. Finally, the Serial Connection section is used to activate the Smart Interface. By clicking “Start Serial”, the Smart Interface is activated to sense any devices that are plugged in. By pressing “Reset Outlets”, all the outlets are turned off at once. The Energy Model and Usage functionalities are just presented as a demo and were not calculated in the project. However, the main takeaway is to demonstrate what the user would see and how his/her behaviors would be altered based on this information.

5.3 Discussion of the Smart Interface

The Smart Interface achieved the goal of presenting just-in-time messaging to the user. However, the hardware design relied on multiple connections to the computer. Because of this, it was necessary to have a computer which had multiple USB ports available. Consequently, if three outlets were to work concurrently, six USB ports would be needed. When re-engineering the prototype to a large scale deployment in the Connected Home, it would be necessary to alter the USB port connections to a serial data bus communication system. The only wireless connection to the Smart Interface was the Bluetooth module for the touchscreen. However, based on the Linksprite datasheet, the components purchased could also use XBee to communicate to the Smart Interface. This was noted for future prototypes.

The hardware design also required many different power sources. The touchscreen needed a battery, the Linksprite components needed unregulated AC voltage, and the RFID required power from the USB, which was connected to the computer. Furthermore, because the

Smart Interface was continuously on, the battery for the touchscreen often ran out of power. This would be a problem if used in the Connected Home. For the next prototype, it would be more appropriate if each Smart Interface were powered through one source. Because the Linksprite components requires unregulated AC power, it would be useful to retrieve all the power from the Linksprite location instead of three separate sources.

Based on the hardware components used, the Smart Interface needs to be constantly on, but users only use the system for a small part of the day; only to turn things on and off. Consequently, a lot of power is wasted in the system. It would be useful for future purposes, if a "Sleep Mode" was created so that minimal power would be used in the system.

Based on the assessment as well as the goals that were created, there was room for improvement for the next prototype designed. With this starting point and the system design and goals outlined in the previous section, the next prototype for the Connected Home was constructed.

6. Energy Mobility Network Prototype

The replacement of the Smart Interface is the Energy Mobility Network. The Energy Mobility Network integrated the hardware components and just-in-time concept of the Smart Interface with the new goals of the Energy Mobility Network. The new goals demanded a change in the hardware components as well as a drastic change in the software functionalities. The Energy Mobility Network Block Diagram is seen in Figure 10.

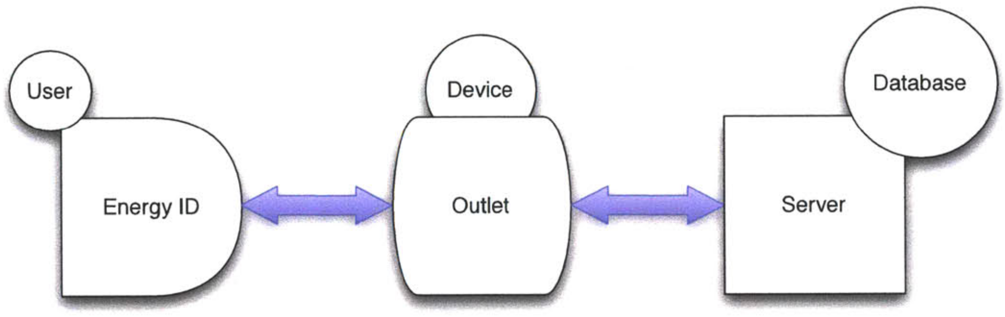


Figure 10. Block Diagram of the Energy Mobility Network

The User goes to the Energy ID for data, the Outlet controls the electricity to the Device, and the Server connects to the Database for information as explained in Section 4. These three modules are connected wirelessly as noted by the arrows. This section will detail the hardware and software ideas for each block as well as the work done to improve the system.

6.1 Energy ID Sketch and Prototype

Based on the requirements for the Energy ID, it was important to create a few Energy ID examples that could be used to represent different personalities of users. Here, the sketch of the Energy ID and the different prototypes of the Energy ID are discussed. This part of the project was led by David Boardman and Carl Yu.

6.1.1 Passive Energy ID Proposed

For active and technologically savvy users, the Energy ID would be implemented in a mobile device as part of an application or as a custom built device. For more passive users, the Energy ID embodiments were more creative. There were two types of embodiments that were focused on: the key fob and bracelet Energy IDs. The goal was to create an embodiment that could be used as a fashion accessory as well as a means to display the energy consumptions of the user. Consequently, the Energy ID sketches were developed as seen in Figure 11.

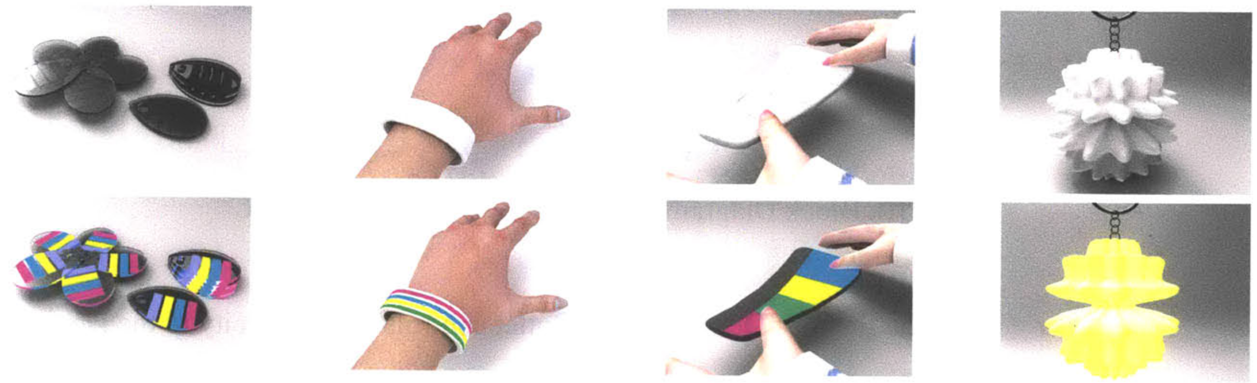


Figure 11. Passive Energy ID Sketch[5]

In Figure 11, there are four different examples of Energy ID embodiments. Each embodiment can be used as a fashion statement -- as a key chain holder, bracelet, ID card, etc. These pieces could be seen in everyday life as normal statements of fashion to the general public. Yet, when the user is in proximity to the outlet, the user can use the Energy ID to tap into the network. The outlet will then display colors onto the Energy ID to signify electricity consumptions as seen in the second row of Figure 11. For instance, the amount of green present in the Energy ID would

[5] Figure created by the MEL Energy Mobility Network Design Team

determine how much electricity has been produced that day and the amount of red present would determine how much electricity has been consumed.

6.1.2 Passive Energy ID Implemented

Based on the proposed passive Energy ID sketches, two passive Energy ID prototypes were created as seen in Figure 12 and Figure 13.

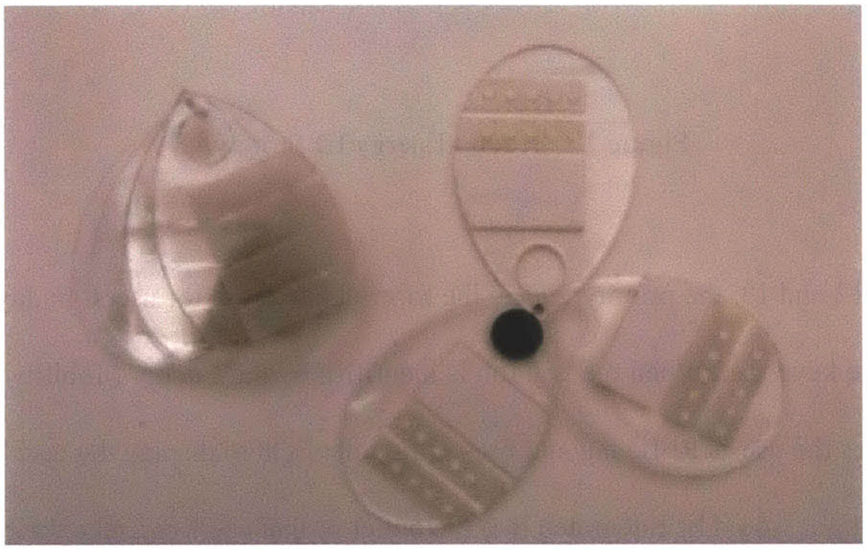


Figure 12. First Energy ID Prototype[6]

[6] Figure created by the MEL Energy Mobility Network Design Team

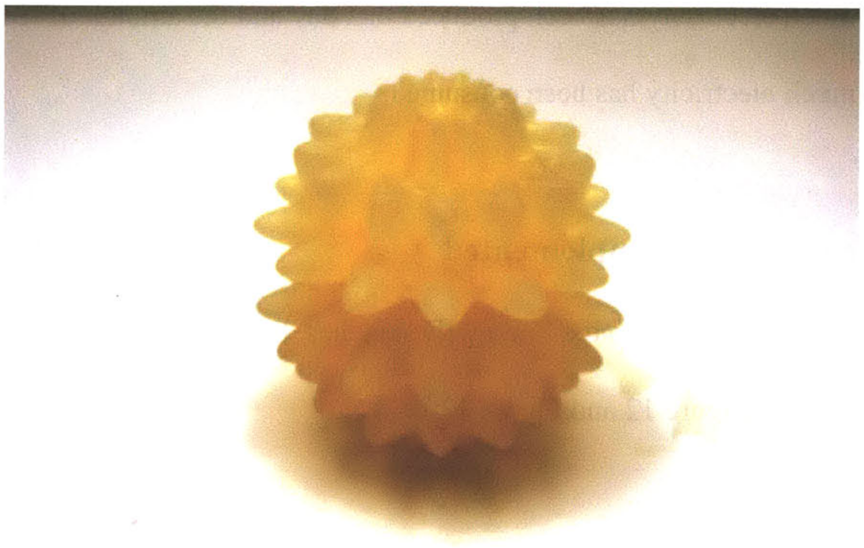


Figure 13. Second Energy ID Prototype[7]

In both Figure 12 and 13, the prototype is in the form of a key fob for users. In daylight, it looks like a decorative keychain. When the key fob is identified by the Energy Mobility Network, the outlet recognizes the user's RFID tag and uses ultraviolet light to display the user's information. The ultraviolet light would be integrated into the outlet so that users can take their passive Energy ID to the outlet to view their current energy data. The amount of information is tailored to the specific passive user. For instance, in Figure 12, the Energy ID prototype is displayed as a cluster on the left, and fanned out as three "petals" on the right side of the figure. The Energy ID can display information on each of the three petals. Each petal has a rectangle of white space as well as two rows; each row having five white dots. In one example, a user could use the white rectangle as his/her profile, the first row as his/her energy consumptions and the bottom row as his/her energy productions for the day, where each dot represents twenty percent of the energy

[7] Figure created by the MEL Energy Mobility Network Design Team

he/she wanted to consume and produce for the day. An example of what this may look like with the ultraviolet light is seen in Figure 14.

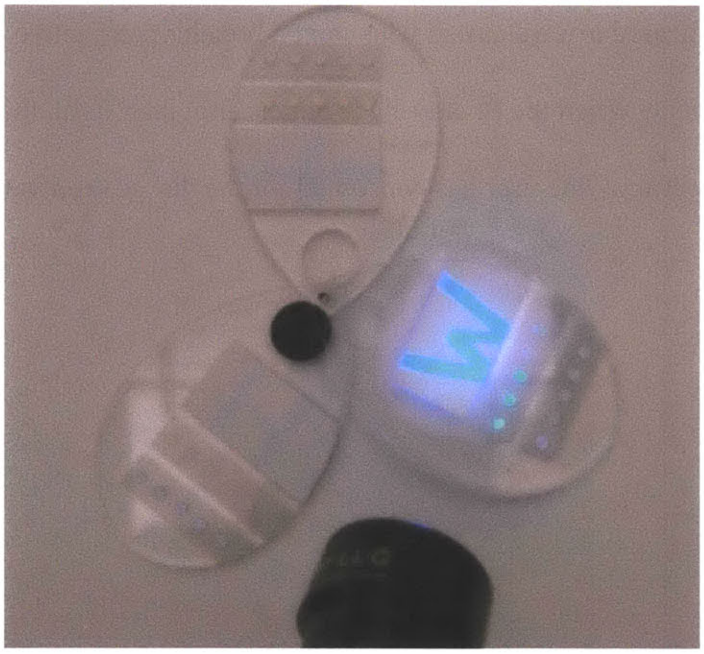


Figure 14. Passive Energy ID under UV Light[8]

In this specific instance, the Energy ID shows that the user has the profile "W", which stands for "Work". He/She has currently consumed 60% of his/her daily electricity (first row of dots) and has not produced any electricity for the day (bottom row of dots). Other users may tailor their Energy ID devices to present information that they feel is important for them. For instance, the UV light could produce a pattern on the rectangular whitespace which shows how much money they've spent on electricity at that moment, the first row of dots symbolizing how many people are in their network, and the bottom row of dots depicting the number of devices that are in use.

[8] Figure created by the MEL Energy Mobility Network Design Team

This specific Energy ID would be useful towards users who want general information about the electricity consumptions, but do not want any detailed scientific data.

A user who wants even less information could have a passive Energy ID as seen in Figure 13. The Energy ID is geared towards users who want a broader perspective on how much electricity has been spent. When the Energy ID is under ultraviolet light, the key fob will light up to show how much electricity the user has consumed for the day as seen in the example in Figure 15.

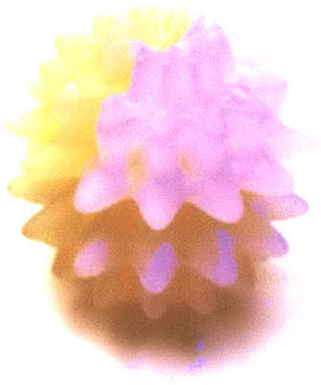


Figure 15. Energy ID under UV Light[9]

Here, the user has consumed fifty percent of his/her given daily electricity consumption. The amount of color produced on the Energy ID will increase as the user continues to use electricity. The user does not know which devices he/she has on or what his/her current profile is. The system retains this information, but displays the information that the user wants to know. In this case, the user only wants very general information about his/her electricity consumptions. This is

[9] Figure created by the MEL Energy Mobility Network Design Team

still beneficial because this type of user may only need minimal prodding to alter his/her behavior to save electricity.

6.1.3 Passive Energy ID and Device Hardware Component

Each passive Energy ID will contain a radio-frequency identification (RFID) tag which will identify the user to the system. Each RFID tag is unique and links the user to his/her personal Energy ID. In this case, using a radio frequency transmission is sufficient. Because the user needs to put his/her Energy ID under the UV light from the outlet to see his/her energy data, close proximity sensors can identify the user to the system. Using a passive RFID tag to distinguish users is advantageous for passive users because there is no power consumption on the user's Energy ID, as opposed to the active users in the system.

The size of the RFID tags is also beneficial for our system. The tags can come in all sizes and forms; it can range from a 16mm diameter button to a 1.93mm diameter cylinder to a credit-card sized RFID tag as seen in Figure 16. In Figure 14, the black button set into the Energy ID is the RFID tag.

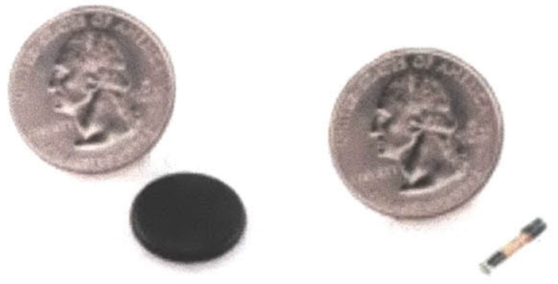


Figure 16. Different Types of RFID Tags[10]

[10] These images were found on Sparkfun's website.

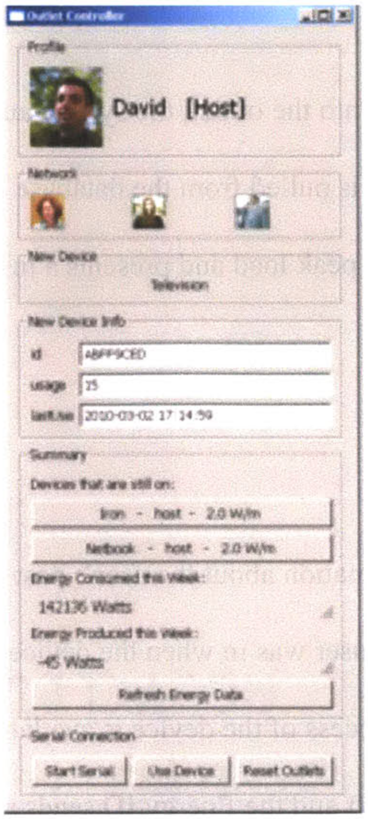
The RFID tags selected were all found on Sparkfun at a relatively inexpensive cost. Each RFID tag contains a 32-bit unique identification number and is sensed by the RFID reader from ID Innovations. The tags run at a 125kHz carrier frequency which aligns with the frequency of the RFID reader.

The devices in the Energy Mobility Network also uses these RFID tags. These tags would ideally be placed by the power cord of the device so that when the user plugs the device into the outlet, the system can automatically sense what device has been plugged in. During the testing phase of the project, the credit card sized RFID tags were used and tapped into the system after the device was plugged in.

When a RFID tag is first linked to a user or device, it must first be recognized to the system as a new RFID tag. The tag must be manually entered into the database along with the user's name or the specific device. This is an easy one-time setup that each user and device must follow in order for the system to correctly identify the object in the future.

6.1.4 Active Energy ID

As explained before, active Energy IDs are geared towards users who want as much information as possible about their electricity expenditures. The physical state for this version of the prototype is assumed to be in the form of a small mobile appliance such as a smart phone. The physical state of the active Energy ID was not explored as much the functionality of it. Figure 16 shows what the user might see on his/her Energy ID interface at any given time.



David [Host]

New Device: Television

New Device Info
id: ABFF9CED
usage: 75
lastUse: 2010-03-02 17:14:59

Summary
Devices that are still on:
Iron - host - 2.0 W/m
Netbook - host - 2.0 W/m
Energy Consumed this Week:
142136 Watts
Energy Produced this Week:
-45 Watts

Serial Connection

Figure 17. User Interface of an Active Energy ID

The user interface seen in Figure 17 presents the user with four different categories: Social, New Device, Summary, and Action Buttons. Each category will be described here, the technical details of it can be found in Section 6.4.2.

Social

This category includes the "Profile" and "Network" sections of the interface. Once the Energy ID is activated, the user receives his/her updated profile. Here, David is at home so he/she receives a "Host" profile. Currently, there are three people logged into his/her network of friends as seen in the "Network" section. These users will be charged accordingly if David chooses to share the cost of using a device with them.

New Device

When a device is plugged into the outlet, the system automatically reads the device's RFID tag. The device information is pulled from the database and presented onto the Energy ID interface. The system analyzes the peak load and presents a just-in-time prompt for the device to the user.

Summary

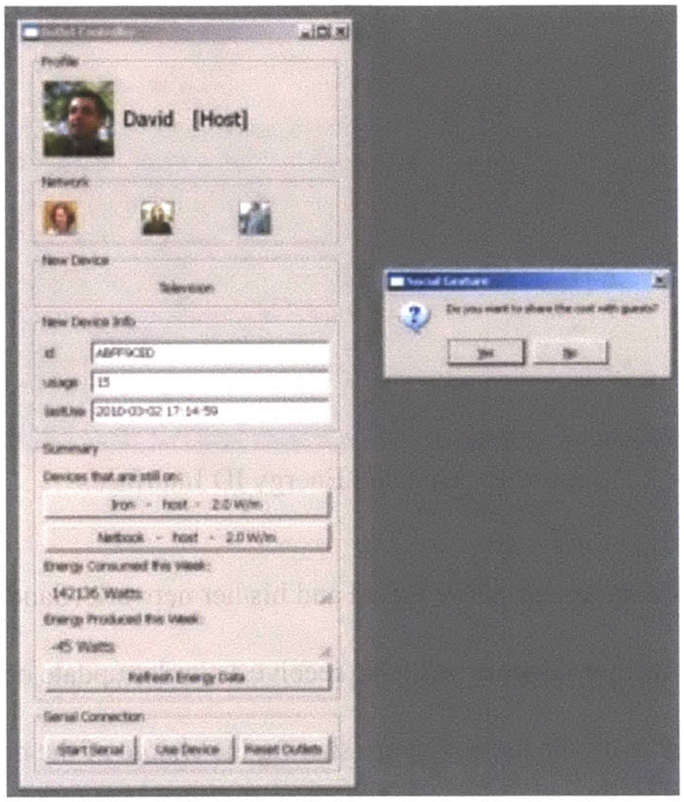
This section displays information about the user's past actions. It lists the devices that are still on as well as what profile the user was in when the device was activated and the cost of the device per minute. Here, remote access of the device is available. The user simply clicks on the device that he/she wishes to turn off and the Energy ID sends a command to the particular outlet to turn off the device. The device disappears from the Summary section and the energy production and consumption costs are updated.

This section also displays the energy consumed and produced for the week. When the user first activates his/her Energy ID, the latest data about his/her energy expenditures are shown. These numbers are updated when the user turns on the Energy ID, the user stops using a device, or if the "Refresh Energy Data" button is pressed.

Action Buttons

The user enables all major actions here. When the user clicks "Start Serial", he/she turns on his/her Energy ID and sends a signal out to the system to update his/her Profile and Summary widget.

Clicking the "Use Device" button tells the system that the user wants to turn on the device listed in the "New Device" section. If there are people in his/her network, another window will pop up and ask if he/she wants to share the cost of the device with his/her network as seen in Figure 18.



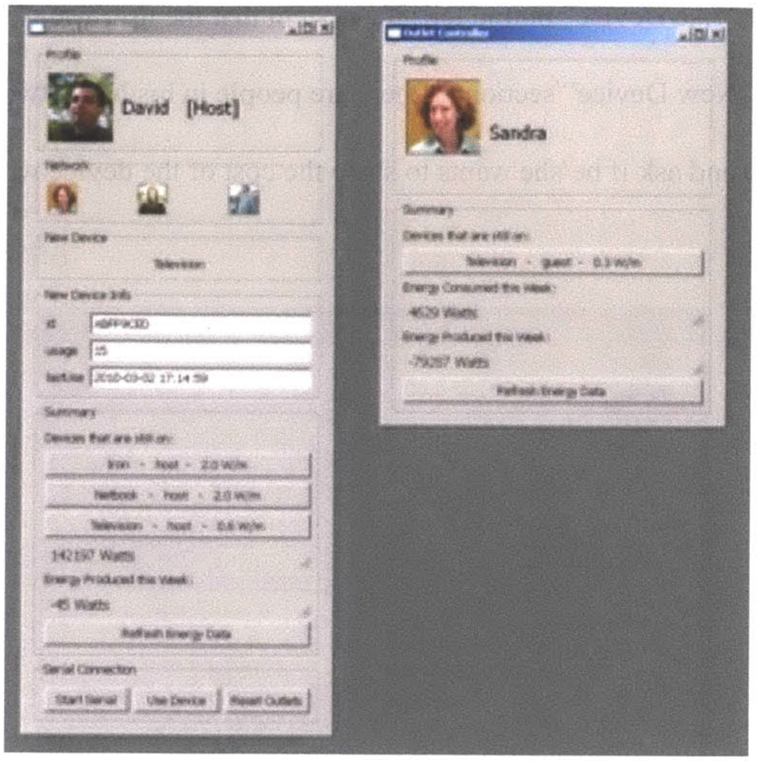
id: ABFF9CED
usage: 15
lastUse: 2010-03-02 17:54:59

Devices that are still on:
Iron - host - 2.0 W/m
Netbook - host - 2.0 W/m
Energy Consumed this Week:
142136 Watts
Energy Produced this Week:
-45 Watts

Do you want to share the cost with guests?

Figure 18. Energy ID Interface with Pop-up Window

The data is then sent to the system so it can turn on the device and start charging the user(s) accordingly. Furthermore, once the device has been turned on, the device will be appended to the list of "Devices that are still on" in the Summary section of the Energy ID as seen in Figure 19.



id: 4BFF9C0D
usage: 15
lastUse: 2010-03-02 17:14:59

Summary
Devices that are still on:
Iron - host - 2.0 W/m
Netbook - host - 2.0 W/m
Television - host - 0.6 W/m
142197 Watts
Energy Produced this Week:
-45 Watts

Summary
Devices that are still on:
Television - guest - 0.3 W/m
Energy Consumed this Week:
4629 Watts
Energy Produced this Week:
-79287 Watts

Figure 19. Updated Energy ID Interfaces

Here, the television is being turned on by David and his/her network (Sandra is only shown here, but the other people in his/her network will also receive a similar update on their Energy ID). This update will happen to all the users that are sharing the charge of the device and will also give them the ability to remotely turn off the device if necessary. It should be noted that because David has the profile of a "Host" he/she is charged .6 Watts per minute where as Sandra, the "Guest", is only charged .3 Watts per minute.

The "Reset Outlets" button allows the user to turn off all the devices listed in the Summary section. As explained before, this action will in turn update the energy consumption and production list.

6.2 Outlet

Based on the user, active or passive, there are two ways that the user communicates with the outlet. An active user will only use the outlet for remote access to devices, but a passive user will have more interactions with the outlet. The passive user will not only use the outlet to control devices, but also to look at the energy consumption summary. The outlet will be designed to meet the requirements of both the active and passive users.

6.2.1 Outlet Hardware Components

The outlet hardware components are similar to the Smart Interface components. Each off the shelf component was tested in isolation first and then integrated together using an Arduino.

Linksprite Powerline Controller and Outlet

The Linksprite Powerline Controller and Outlet allow electricity to flow through the device. The off the shelf component is useful for the remote access functionality of the system. The Linksprite components use the power line for three functions: to source power to the component itself, power the device connected to it, and communicate with the other Linksprite module. Both the Linksprite Controller and Outlet take in unregulated AC voltage as their input voltage.

The Linksprite Controller has a daughter card to give the designer the ability to use the Linksprite with different interfaces such as RS232, RS485, USB, XBee, etc. The communication with Linksprite was done with UART-USB for testing and XBee for the final prototype.

The Linksprite Outlet acts as an electrical relay for the device plugged in. The Linksprite Outlet receives commands directly from the Controller. Each Linksprite Outlet has a unique ID that is nonprogrammable and allows the Controller to identify which Outlet should be switched on or off. Consequently, if the designer wants to control electricity through two devices, there needs to be two Linksprite Outlets connected to the Linksprite Controller.

The Linksprite Powerline Controller and Outlet were bought as a package on Sparkfun. The Controller came with a UART daughter card which allowed for USB access to the package. In order to enter the command mode of the Controller, the designer must type in `+++`. The response `OK` will be received and the Controller will be set in command mode. In order to turn on a device, the designer must type in `ATON <outlet ID>`. Similarly, to turn off a device, the command prompt is `ATOF <outlet ID>`. If the outlet ID is excluded from the command, the controller will turn on or off all the outlets that are connected to it. If the command is received properly, the Controller sends an acknowledgement back to the Outlet.

When purchased, the Linksprite Outlet ID may not be written on the module. Consequently, a Python script was made to find the specific Outlet ID of the component. The script sent an ATON command followed by a number. If the outlet did not turn on, the script would increment the number and try sending the ATON command again. It would continue to iterate until the device turned on. When tested, the Linksprite Outlet was connected to a fan so once the Outlet ID was found, the fan would turn on. The Linksprite was tested at these settings: 9600bps, timeout set to 1, and rts and cts set to 0. The code can be found in Appendix A and is labelled as OutletTester.py.

RFID Sensor

The RFID sensor will identify the passive users and the devices in the system. The specific sensor chosen is the ID-20 by ID Innovations. The RFID reader was selected because it had a carrier frequency of 125kHz, the same frequency as the ID tags purchased on Sparkfun. It has an internal antenna that allows for a 16cm read range, a distance that is appropriate for the outlet. An external antenna and external tuning capacitor were not used.

The RFID USB reader, a breakout board found on Sparkfun, allows the designer to read the ID-20 by sending the data serially via USB. The data can be read with Hyperterminal at 9600bps, 8 data bits, no parity bit, and one stop bit. This was particularly useful for testing the ID-20 in isolation before integrating it into the outlet design.

XBee

XBee, also known as XBee, is used to serially send data to and from the Energy ID, the outlet, and the database. The specific XBee PRO chip used is the Series 1 802.15.4 Chip Antenna. It is a digital radio that is low power, small in terms of footprint on the board, and allows the designer to create a robust multipoint network among the XBee modules -- the key point in selecting this particular connection. Another selling point for the Series 1 chip is that there is a Python library available for Series 1 XBees. Each active Energy ID will have a XBee module that will communicate with the XBee inside the outlet as well as the XBee connected to the server. The outlet and Energy ID XBees are End Device/Router XBees that mainly talk to the XBee Coordinator which is the only means of communication to the server.

The XBee Development Board provided by Digi creates direct access from the PC to the serial pins of the XBee chip through USB. The easiest way to program the XBee was to use the development board and to download the X-CTU program. The Port settings selected were 9600 Baud rate, no flow control, 8 data bits, no parity bit, and one stop bit. After querying the XBee chip to make sure the PC connects to the XBee, the chip is programmed to have a modem of XB24 and a function set of "802.15.4". The PAN ID is set to a unique number, in this instance "3222" and the Destination Address Low is set to "5678", to communicate to the XBee PRO chip set to that address. In order for others to talk to the Coordinate XBee, their Destination Address Low, the 16-bit Source Address Low is set to "5000." Finally, the API must be enabled (AP = 1). This is necessary for the Python library that was used.

In order to test if two XBees are communicating with one another, the other XBee chip should be queried and programmed to have a modem of XB24 with the function set of "802.15.4". The PAN ID must be set to the same PAN ID as the Coordinator, in this case "3222" and the Destination Address Low is set to the Coordinator's 16-bit Source Address Low. The API should not be enabled in the End Device XBee. The easiest way to make sure that the XBee End Device chip will connect to the Coordinator is to make the XBee Coordinator settings, save the profile settings, and load the profile to the XBee End Device chip and only change the Destination Address Low parameter, MY 16-bit Source Address Low, and API parameter.

To verify that two XBees are communicating, two X-CTU programs must be opened. By using the Terminal tab on each X-CTU program, the designer can assemble a packet from one XBee and see if the other XBee receives it on the second X-CTU Terminal window.

The AT Function Set was chosen for the XBee router so that the data can be sent serially from the XBee into the Outlet Serial Port. The XBee Coordinator receives it in a packet structured in a frame, the API setting, where the Coordinator can sift through the information to access the database. The configuration of a coordinator and router/end device is useful because the server is the coordinator, each outlet acts as a router, and each active Energy ID is an end device. This topology serves the best purpose for the Energy Mobility Network.

Kent Display ChLCD

The Kent Display will be useful for passive users who plan to use the outlet to access their electricity expenditure summary as well as to remotely turn on/off devices. This particular display was chosen because of its ability to retain its image on the display even when there is no power to the board. Consequently, power is saved when the screen is on sleep mode.

The 240x160x2.9 cholesteric liquid crystal (ChLCD) display uses Serial Peripheral Interface to communicate with the outlet. In order to produce an image on the screen, it is necessary to purchase the breakout board from Sparkfun -- the breakout board connects to the 16 pin FPC connector and gives the designer a simpler header to work with. By connecting the header pins to an Arduino and using the example code provided by Sparkfun, a striped image is produced on the screen. It should be noted that the Sparkfun example code should be altered from:

```
SPCR = (1<<SPE) | (1<<MSTR) | (1<<SPR0) | (1<<CPHA);
```

to

```
SPCR = (1<<SPE) | (1<<MSTR) | (1<<SPR0) | (1<<SPR1) | (1<<CPHA);
```

Without changing the SPI control register as seen above, the SPI speed is not set correctly. The speed should be set to 250kHz.

It should be noted that the Kent Display is not a dynamic changing screen. Instead, in order to change the screen, each row of the ChLCD screen must be updated. The Kent Display has 32KB of image memory, which is roughly six images. Each image is loaded as a hexadecimal array into the image memory. The Arduino must use a command to read the RAM to update the screen. The update process for a full screen update is slow (about 1.5 seconds), but the display allows partial updates which can work in the designer's favor.

To render text or a custom made display, the image must be saved as a monochrome bitmap file with one bit per pixel. The software, LCD Assistant, helps translate the image into a hex array. It is useful to note that the byte orientation setting in the software should be vertical, the size width should be 240 and the size height is set to 160 in order to create the right hex array for the screen. After pasting the hex array into the PROGMEM in the code, the Kent Display should accurately receive the image [15].

Arduino Uno

The Arduino was used to build the Outlet for the Results section of the report. The ideal situation was to use the microcontroller as the main component of the Outlet. However, the Arduino Uno was used as the replacement for the microcontroller.

The Arduino Uno is composed of an ATMega328 and has headers which connect to the pins of the microcontroller. It contains a USB port as well as a power source. The USB port is used to send the firmware to the ATMega328 for testing purposes. The final product uses neither as the power is sourced from the Linksprite Components and the firmware is precoded into the ATMega328. Consequently, in this report, it is possible to say that the ATMega644P can be replaced with the Arduino Uno and vice versa.

Microcontroller

The ATMega644P allows the outlet to be an isolated unit from the PC. The microcontroller will be the master component of the outlet and will coordinate commands among the outlet components, Energy ID, and device. This particular chip was selected because it has two serial USART and one SPI Serial Interface -- useful for communicating with XBee, Linksprite, ID-20, and the Kent Display. Furthermore, the amount of flash available was maximized to allow the designer to have extra space for future code.

The chip was programmed using the AVRISP mkII programmer. Instead of writing the code in AVR Studio, the hex file from the Arduino code was directly flash programmed into the microcontroller. To access the Arduino hex file from the Arduino code, it is necessary to press the "Shift" button while clicking the "Compile" action in Arduino. This gives you the direct path where you can find the hex file.

6.2.2 Connecting the Individual Components

After selecting the individual components necessary to create the specifications for the outlet, the individual components were then integrated together as seen in Figure 20. Here, the Outlet is described with the ATMega644P as well as the Arduino Uno.

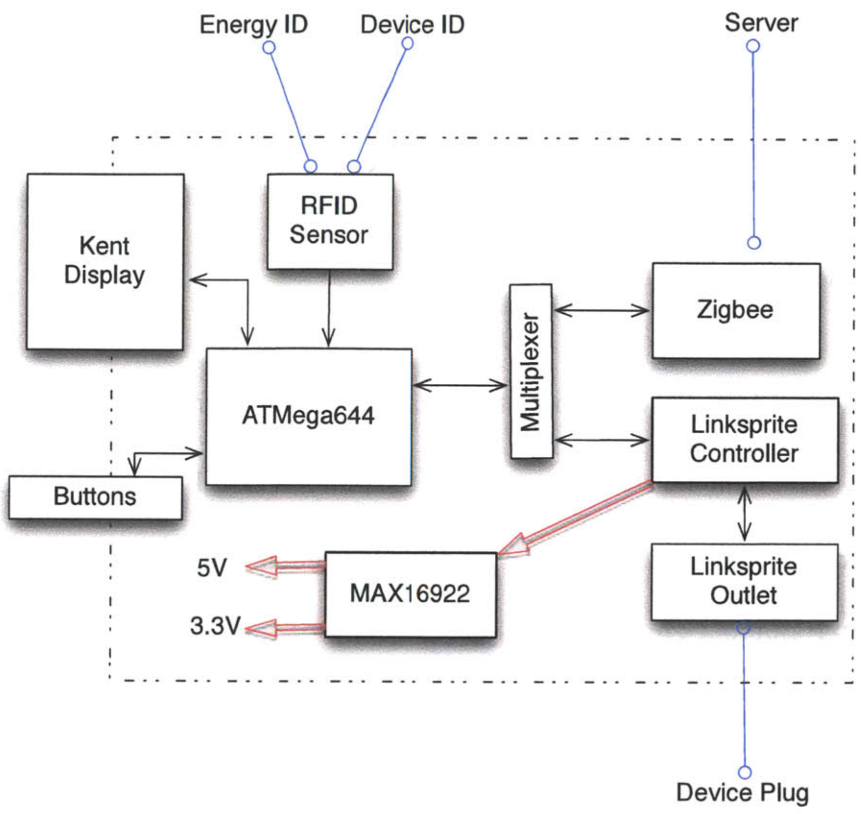


Figure 20. Block Diagram of the Outlet

The ATMega644P is the fundamental component of the outlet as seen in Figure 20. It receives information from the RFID sensor, processes the RFID tag, and sends the data to the Kent Display and XBee for wireless communication to the server. If a passive user wants to turns on a device, the user presses a button, which the microcontroller processes before sending a command out to the Linksprite Controller. The Linksprite Controller directly communicates with the Linksprite Outlet, which turns on the device.

Looking at the ATMega644 datasheet, there are only two USART (Universal Synchronous Asynchronous Receiver Transmitter) connections available. The RFID reader, Linksprite Controller, and XBee each require USART connections to the microcontroller. To fix this, a multiplexer for the Linksprite Controller and XBee to one of the UART connections on the

ATMega644 is needed. The other USART connection will be directly connected to the RFID reader. The multiplexer chosen is the On-Semi 74VHC4052 Multiplexer. The ability to multiplex four separate serial lines was useful for the XBee and Linksprite transmissions as well as for any future transmissions that may be incorporated in the design.

To minimize the number of power sources, the outlet design takes in the unregulated 120VAC via the Linksprite module. On the Linksprite board, the AC220S12DC-6W chip takes the unregulated AC voltage and outputs 12VDC with an output current of 500mA. By soldering a Molex Jumper Wire to the output of the AC220S12DC-6W chip, it is possible to take the 12VDC and use it to supply power to the other components in the outlet. In order to do so, it is necessary to use a step down DC to DC converter. After some research, the MAX16922 was selected because it could output 5V and 3.3V, voltages necessary to power the other parts in the outlet design and for the small footprint.

As said before, the AC220S12DC-6W chip on the Linksprite board outputs a current of 500mA. Because the board will be sourcing all its power from this chip, the maximum amount of current in the system must be less than 500mA. Based on the datasheets of each module, the estimated amount of current would be 235.4mA as seen in Table 2.

Component	Maximum Current
Atmega644P	6 mA
Kent Display	19.4 mA
ID-20	65 mA
XBee	45 mA
LinkSprite	100 mA

Table 2. Maximum Current of Each Chip

Of course, there are a few assumptions made for the calculations. The estimated amount assumes all of the components are at peak current, which is not the case for the outlet. The microcontroller will communicate with all these components throughout the usage of the outlet, but never all at the same time. Furthermore, some minor components have not been calculated into the total because an estimate about the total current was needed. Consequently, the current measurement will not be 235.4mA, but an amount of that spectrum.

In order to receive input from the passive user, buttons are needed. The Omron push buttons directly communicate with the microcontroller and correlate to what the Kent Display presents to the user. Also seen in Figure 20 are inputs and outputs to the outlet. These connections show what information is sent into the outlet (Device ID, Energy ID, Server) and what information is sent out of the outlet (Server, Device Plug). With the block diagram created, the schematic was then created in Eagle, which is found in Section 6.2.3.

In the Results section, the ATMega644P is replaced with an Arduino Uno. Connecting the individual components to the Arduino Uno is quite similar to connecting to the ATMega644P. There are a few minor differences. First, in the Arduino Uno, there is only one USART connection. Consequently, instead of using a multiplexer as seen in the ATMega644P case, software was used to fix this problem. The software details can be found in Section 6.4.3.

The Arduino Uno takes in power from three possible sources: USB, a 9V wall wart, or the *Vin* pin on the Arduino. Given the fact that the Linksprite component is able to source 12V to the Outlet and the Arduino accepts voltages up to 20V, it was logical to use the Linksprite component to power the Arduino. This also allowed minimal changes to occur when shifting from the Arduino Uno to the ATMega644P. Consequently, the MAX16922 chip was not used in the Arduino-mode of the Outlet.

6.2.3 Creating the Outlet Schematic with the ATMega644P

When creating the schematic for the outlet, I opted to use headers whenever possible. Headers are useful because it gives the designer the ability to replace chips when needed and to test the chip in isolation then place it back onto the outlet board. The only consequence of headers is that it uses more vertical space on the board than just placing the chip directly onto the board. Headers were used for the LinkSprite Controller, Kent Display, and XBee as seen in Figure 21.

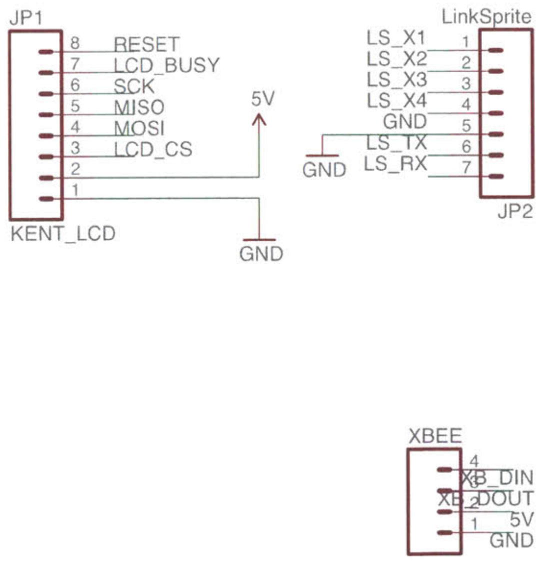


Figure 21. Headers for the Kent Display, Linksprite, and XBee Components

After some testing with the Linksprite Controller and XBee, it came to my attention that directly placing the XBee on the daughter card of the Linksprite Controller was not possible due to a possible short with the VCC pin on the Linksprite board. Consequently, the XBee chip became a separate component on the outlet board. The connections to the microcontroller, ATMega644P are shown in Figure 22.

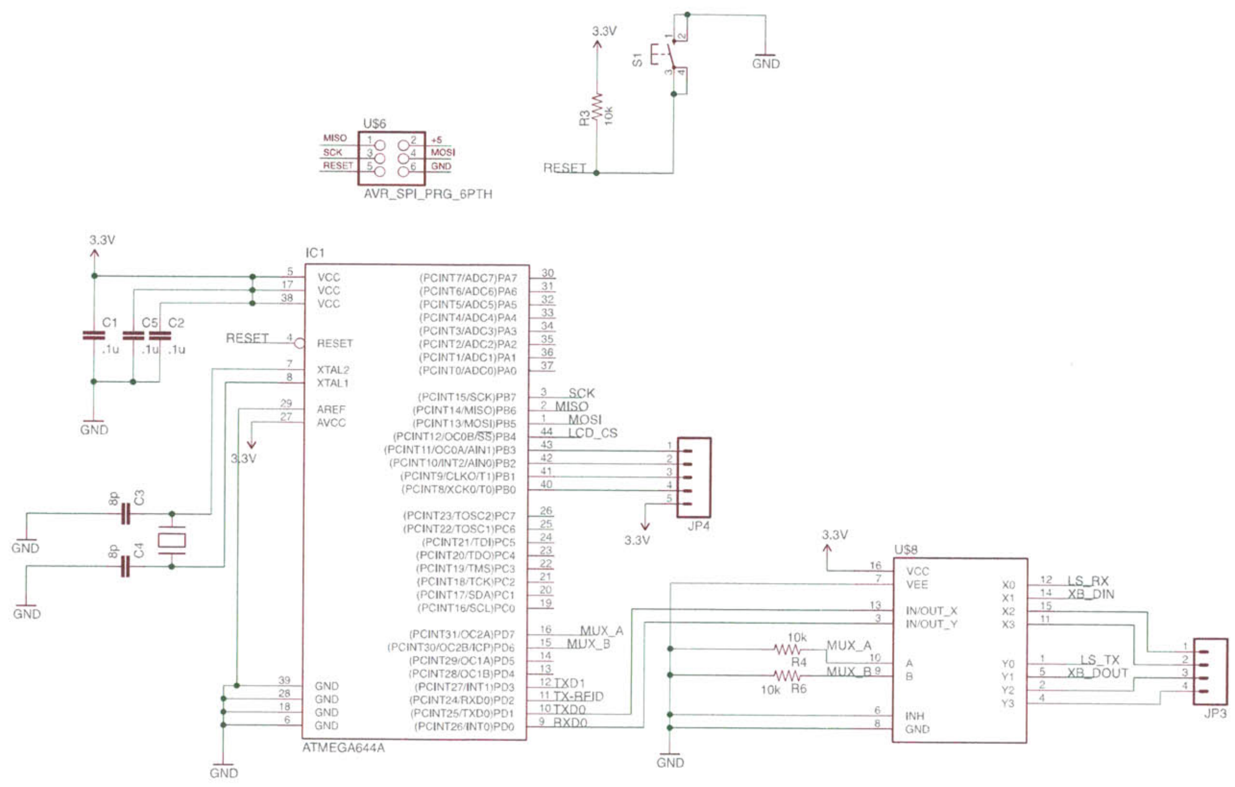


Figure 22. Microcontroller and Multiplexer Schematic

As seen in the microcontroller schematic, decoupling capacitors are used for the power supply and ground. Connections such as setting AREF to ground and putting the RESET pin to active low were based on the ATMega644P datasheet. The AVR_SPI_PRG_6PTH was created in order to easily reprogram the ATMega644P chip while testing and debugging. Pins 40 to 43 are connected to the buttons, where JP4 symbolizes the headers for the buttons. The buttons and Kent Display are headers because they will be sitting on top of the outlet board, as an additional board to the outlet.

As explained in the previous section, the multiplexer is connected to pins 9 and 10 of the ATMega644P. VEE of the multiplexer, U$8, is connected to ground because the lowest voltage received on the multiplexer control pins is ground. The unused multiplexer output pins are connected to a header for future use in the outlet.

JP4 of Figure 22 correlates to the button outputs, S1_OUT and S2_OUT, of Figure 23 as seen below. These two outputs can be placed onto any pin of JP4 -- the software code will signify which pin becomes connected to the button output.

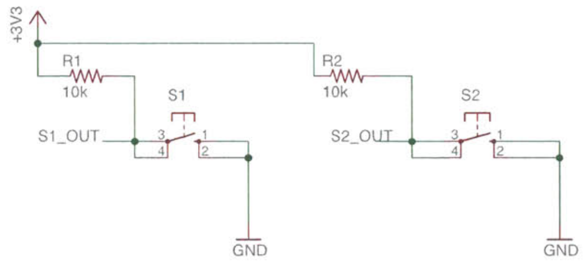


Figure 23. User Interface Button Schematic

A board was not created for the buttons. Instead, a leftover protoboard was cut and used to solder on the buttons and resistors due to the cost of fabricating a board. The 3.3V wire was connected to the JP4 supply voltage and the ground was soldered onto the ground plane. For future purposes, JP4 would have an additional pin for the ground wire coming from the protoboard.

Pin 11 of the ATMega644P is tied to the transmit pin of the ID-20 as seen in Figure 22 and Figure 24.

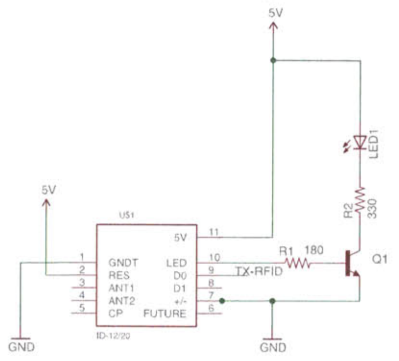


Figure 24. RFID Schematic

Instead of using the RFID breakout board which connects to the PC, the outlet is designed so that it directly connects to the microcontroller. Because the buzzer from the breakout board was not added to the outlet to identify that a RFID tag had been sensed, an LED was used instead. External antennas were not used as seen in Figure 24.

Finally, the MAX16922 schematic was created as seen in Figure 25 below:

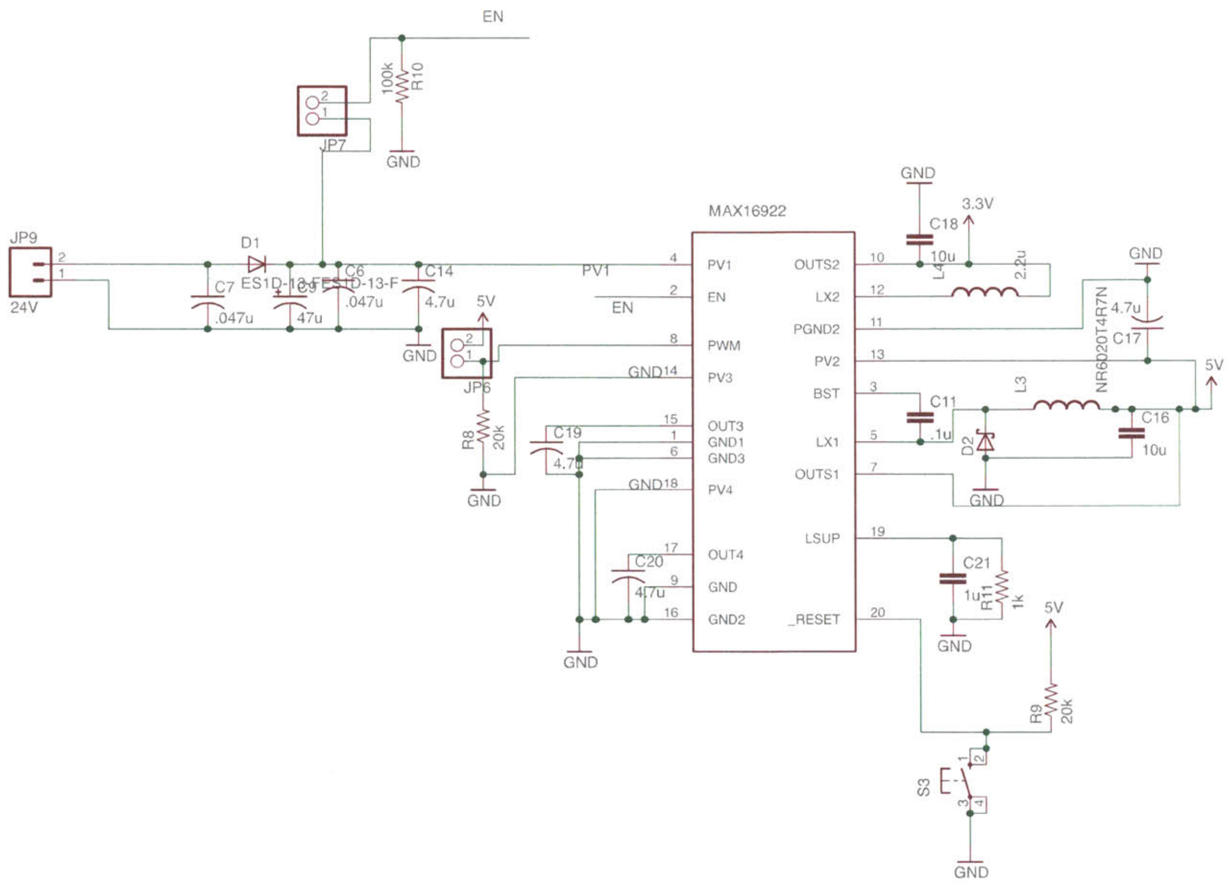


Figure 25. MAX16922 Schematic

The design of the chip is based on the MAX16922 evaluation kit datasheet found on the Maxim website. By using this particular design, the pins OUTS1 and OUTS2 can produce 5V and 3.3V respectively. Because OUT3 and OUT4 were not used, they are connected through a capacitor to ground as requested by the datasheet. Shunts were used in order to check each subsection of the module for testing purposes. In the end, JP6 and JP7 will have shunts on each to produce the right output.

6.2.4 Creating the Outlet Board with the ATMega644P

With the schematics created, the board layout was designed as seen in Figure 26.

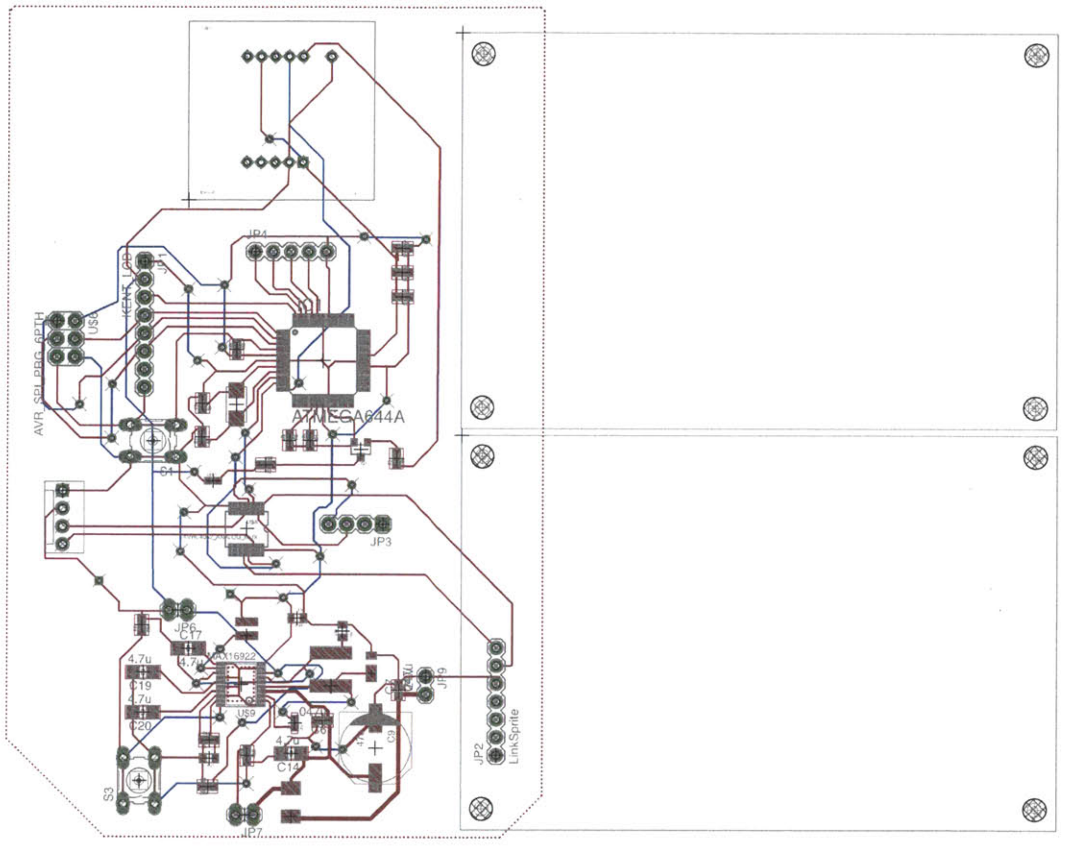


Figure 26. Outlet Board Layout

Components were placed logistically, but also to minimize space. The bottom left section of the board was designed as the power management area, where the MAX16922 chip would output the 5V and 3.3V. The DC-DC converter component placement was done similar to the Evaluation Kit MAX16922 layout from Maxim. The XBee component and RFID sensor were placed on the outside of the outlet board so that data transmission would be easily accessible. Furthermore, the AVR_SPI connections were placed on the outside, so that the AVR kit could easily plug into the outlet board for quick reprogramming of the ATMega644P chip. Figure 26 does not show the

Kent Display and buttons board. Because the users will interface with that directly, the Kent Display and buttons will sit on top of the outlet in a box similar to Figure 8. Finally, the Linksprite Controller and Outlet were placed such that the AC load was on the outside of the outlet.

Because some components were not found on the Eagle library, they were created manually and placed in the Eagle library. The component pads and size were found on the datasheet and then integrated onto the board. The Sparkfun Library was downloaded and used for many of the components on the board.

6.2.5 Outlet Assembly and Testing with the ATMega644P

When assembling the board, the power section (bottom left of Figure 25) was first built and tested. A 12V input from a regulated DC Power Supply and the test points, the square vias, were probed to check if it outputted 5V and 3.3V. The multiplexer control pins needed to be soldered to the multiplexer chip because the board did not connect the pins together. After assembling all the chips and passive components, evaluation of the board was completed.

To test the RFID module of the board, a ID-20 was placed onto the board to see if the LED would light up when a ID was tapped by the sensor. The ATMega644P was evaluated by checking if the ATMega644P key was correctly written and read from. It was useful to write and read the ATMega644P chip at a rate of 125kHz.

The hex file sent to the AVR programmer was code written and compiled in the Arduino program. The majority of the code was evaluated in Arduino because the programming was done while the board was being manufactured. Consequently, an Arduino Uno was used to test out isolated components and integrated components together. The Arduino Uno was chosen due to the size of the ATMega328 chip -- the Arduino Duemilanove uses an ATMega168 chip, which could not fit the entire outlet code. The outlet code will be discussed in Section 6.4.3.

6.2.5 Outlet Wiring with the Arduino Uno

In order to wire the Arduino Uno to the other components, a breadboard was used. As with the ATMega644P, the digital pins were sufficient enough. The XBee Explorer or the XBee Regulated Boards were not used as the XBee Breakout board was sufficient to connect the RX and TX pins to the Arduino. The transmission lines of the XBee and Linksprite were connected together and communicated to the RX/TX lines of the Arduino. The Linksprite TX line was left open because it disrupted the connection between the XBee Router and XBee Controller. Finally, the RFID TX line was connected to the serial port created through software. All of the pin connections can be found in Appendix C.

6.3 Server

The server is responsible for logging all the information about the users, devices, and their interactions. It receives the data wirelessly through the XBee. In particular, a XBee XBIB-U_DEV, the development board for XBee, is directly connected to the server with a XBee chip set as a Coordinator. The server saves all the data on a database. The server code will be discussed in Section 6.4.1.

6.3.1 Database

Before discussing the database, a few terms must be cleared:

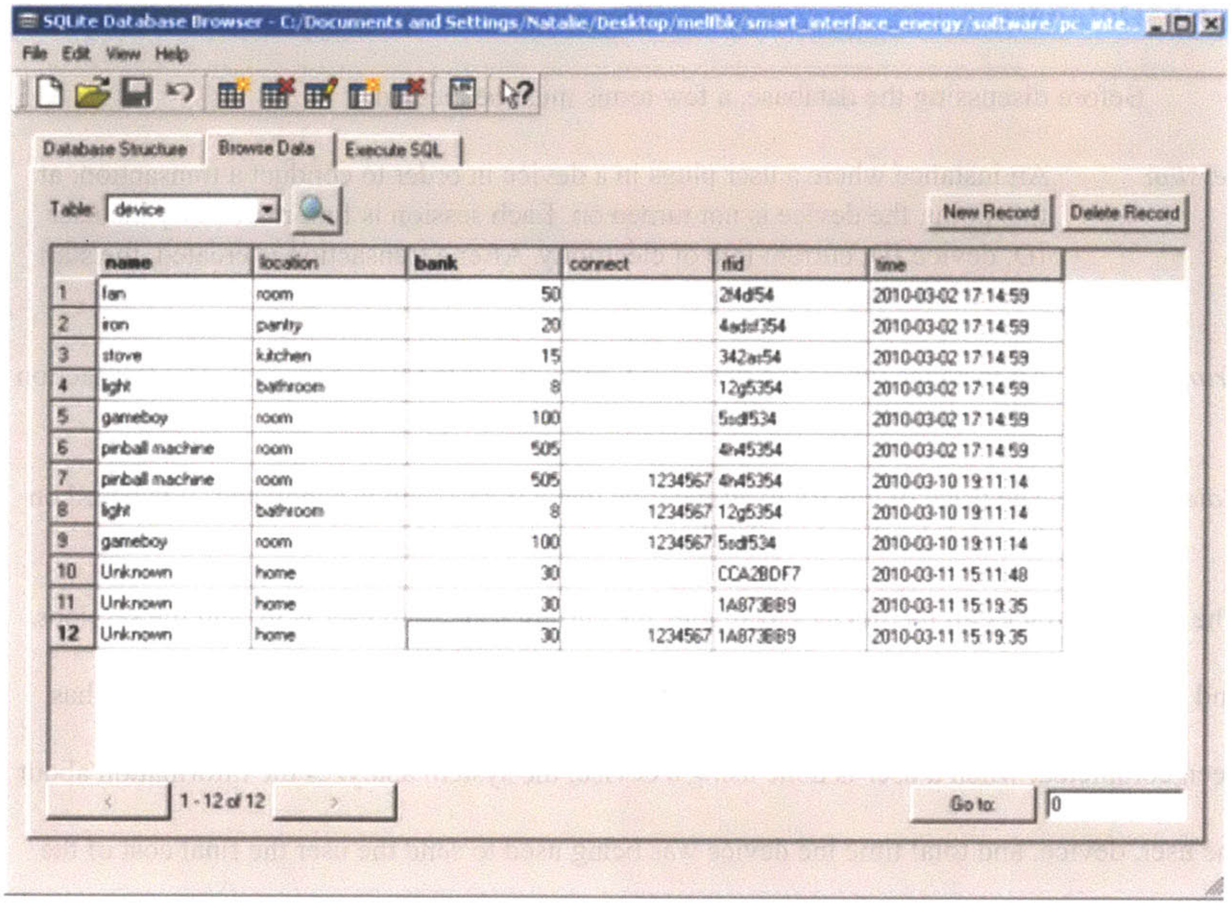
Session	An instance where a user plugs in a device in order to conduct a transaction; at this point, the device is not turned on. Each session is first marked by the user ID, device ID, current rate of electricity. After a transaction is created, the start time of the transaction is noted.
Transaction	An instance where the device is turned on per the user's request. The transaction ends when the device is turned off.
Rate	The price of electricity at the time that a transaction is conducted. It is based on the peak load and availability of energy.

The database tracks all the devices available, the current state each user is in, and the sessions and transactions that are or have occurred between users and devices. Once a transaction has been completed, when a user is done using a device, the system analyzes the information about the user, device, and total time the device was being used to send the user the final cost of the transaction. To insert and receive information from the database, SQLite for Python was used.

The database contains four different tables – Device, User, Outlet and Session:

Device Table

The Device Table contains information about every device logged into the system and if the device is in use. A screenshot of the Device Table is seen in Figure 27; the explanation of the parameters is shown in Table 3.


SQLite Database Browser - C:/Documents and Settings/Natalie/Desktop/melfbk/smart_interface_energy/software/pc_inte...
File Edit View Help
Database Structure
Browse Data
Execute SQL
Table: device
New Record
Delete Record
1 - 12 of 12
Go to:
0

	name	location	bank	connect	rfid	time
1	fan	room	50		2f4df54	2010-03-02 17:14:59
2	iron	pantry	20		4adsf354	2010-03-02 17:14:59
3	stove	kitchen	15		342as54	2010-03-02 17:14:59
4	light	bathroom	8		12g5354	2010-03-02 17:14:59
5	gameboy	room	100		5sdf534	2010-03-02 17:14:59
6	pinball machine	room	505		4h45354	2010-03-02 17:14:59
7	pinball machine	room	505	1234567	4h45354	2010-03-10 19:11:14
8	light	bathroom	8	1234567	12g5354	2010-03-10 19:11:14
9	gameboy	room	100	1234567	5sdf534	2010-03-10 19:11:14
10	Unknown	home	30		CCA2BDF7	2010-03-11 15:11:48
11	Unknown	home	30		1A873BB9	2010-03-11 15:19:35
12	Unknown	home	30	1234567	1A873BB9	2010-03-11 15:19:35

Figure 27. Screenshot of Device Table in the Database

Parameters	**Definition**
name	name of the device
location	location of the device
bank	amount of watts used
connect	who the device is connected to
RFID	RFID tag of the device
time	time of last use

Table 3. Device Table Parameters and Definitions

An example of a Device Table record can be seen in row 1 of Figure 27. The record shows that the device is a "fan" with a RFID tag of "2f4df54". The fan is located in a "room" and costs "50" Watts per hour. The time listed is the first time the fan was introduced to the system.

There can be more than one instance of a device. Each device when initialized for the first time will create a permanent record in the database, as seen in rows 1-6 of Figure 27. If a user turns on a device to create a transaction, a record of the transaction is created as seen in row 7-9 of Figure 27. These rows show current transactions in progress; devices that are still on. Once the transaction is completed and the devices are turned off, the record is deleted from the Device Table. Consequently, the Device Table records what devices have been recognized in the system and current transactions between a user and a device.

User Table

The User Table is similar to the Device Table. It identifies all the users in the system as well as users that have a device in use. The User Table parameters are seen in Table 4.

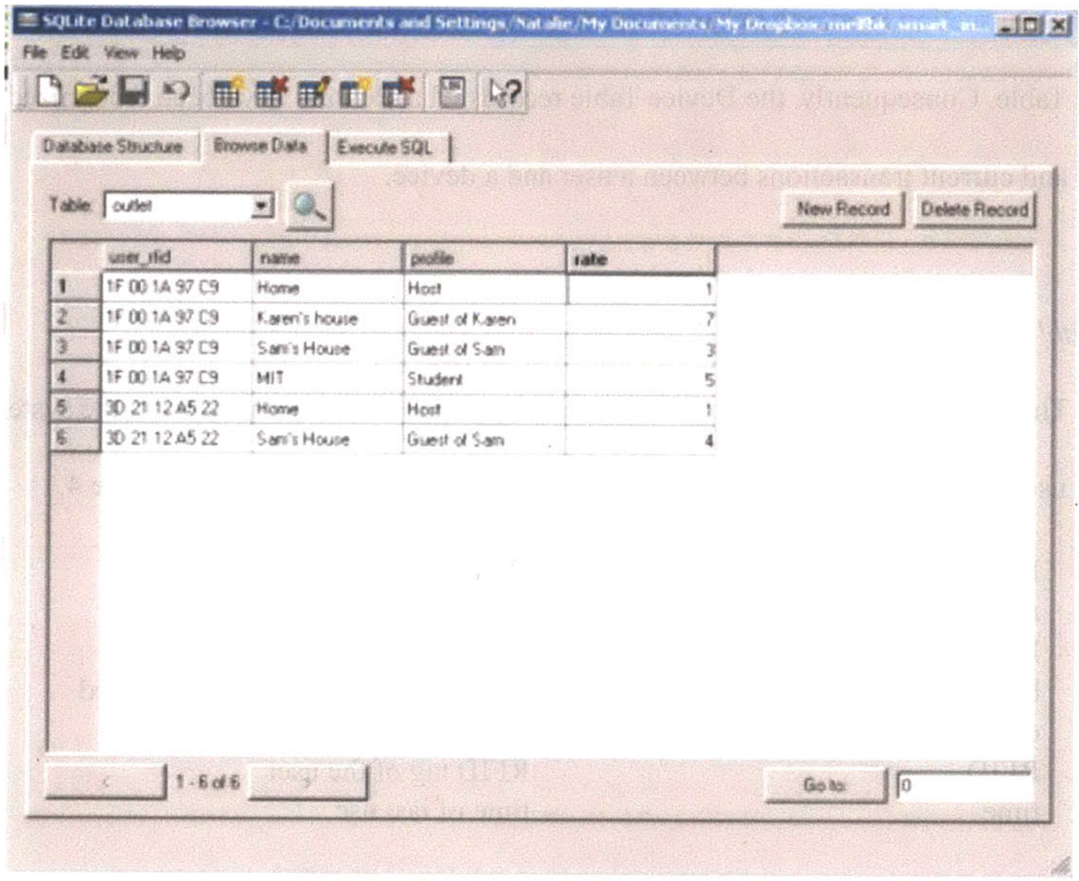
Parameter	**Definition**
name	name of the user
location	user is in or out of the house
bank	how much electricity user has used
connect	who the user is connected to
RFID	RFID tag of the user
time	time of last use

Table 4. Parameters and Definitions of the User Table in the Database

The only difference from the Device Table in the User Table is the definition of the bank parameter. Here, bank is defined as how much electricity has produced and consumed. Thus, the parameter can be a negative number if the user has consumed a lot of electricity or a positive number if the user has produced a lot of electricity. As previously described, the user ideally aims for a zero-sum balance in the bank.

Outlet Table

This table helps the system determine the cost after a transaction is completed. As seen in Figure 27, the Outlet Table lists every user and all the possible profiles that the user will ever be linked to. Table 5 explains the parameters to Figure 28.



	user_rfid	name	profile	rate
1	1F 00 1A 97 C9	Home	Host	1
2	1F 00 1A 97 C9	Karen's house	Guest of Karen	7
3	1F 00 1A 97 C9	Sam's House	Guest of Sam	3
4	1F 00 1A 97 C9	MIT	Student	5
5	3D 21 12 A5 22	Home	Host	1
6	3D 21 12 A5 22	Sam's House	Guest of Sam	4

Figure 28. Screenshot of the Outlet Table

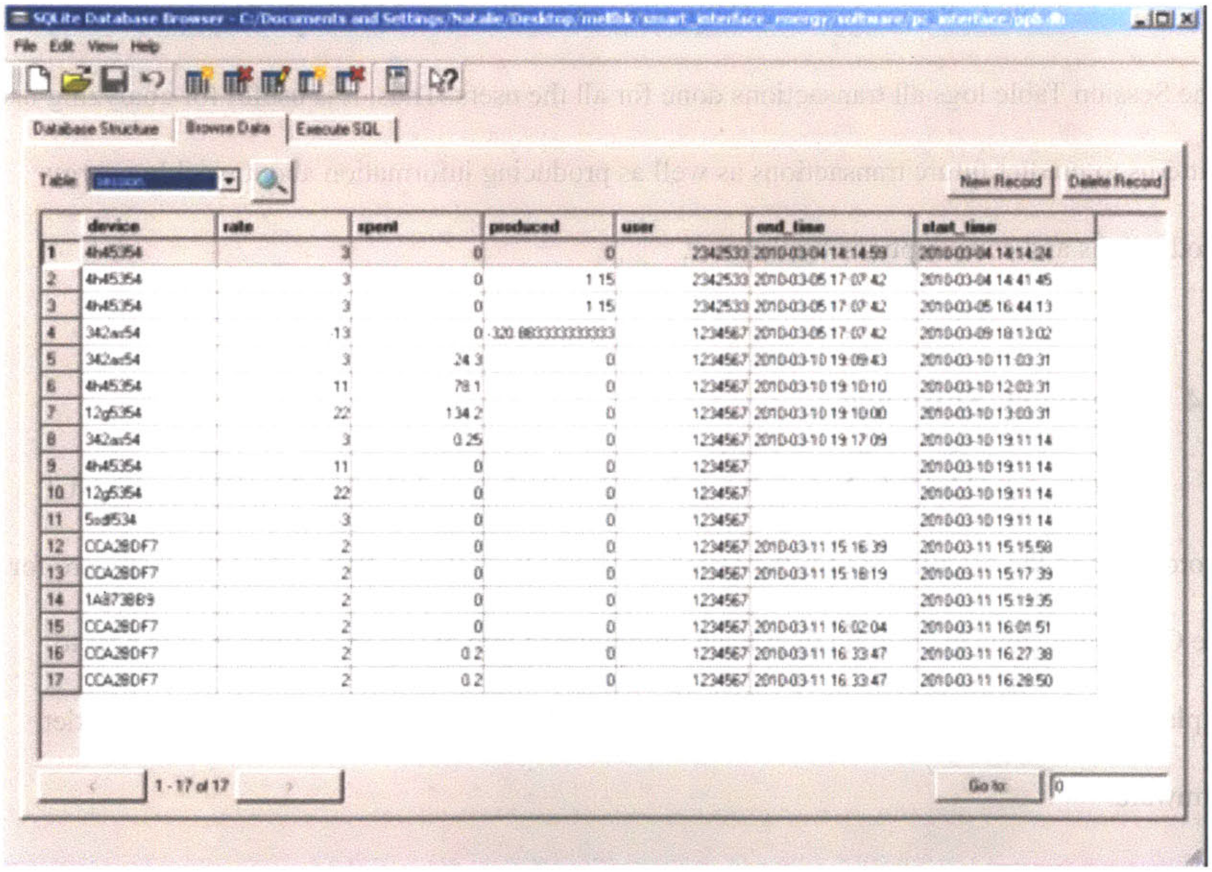
Parameters	**Definitions**
user_rfid	RFID tag of the user
name	name of the place/context
profile	context in which the user is in
rate	factor of electricity cost based on profile

Table 5. Parameters and Definitions to the Outlet Table in the Database

In Figure 28, there are two users that have a different number of profiles. The profile and rate are dependent on the user and are set when the user enters the context for the first time. The records in the table do not change when a transaction is called. Instead, the system is programmed to use the Outlet Table to calculate the cost of every transaction completed.

Session Table

This table lists all the sessions and transactions done by every user to any device. When a session is recorded, it lists the RFID tag of the device, the current rate of electricity, and the RFID tag of the user. When a transaction starts, the start time is recorded as seen in rows 9-11 and 14 of Figure 29.


SQLite Database Browser - C:/Documents and Settings/Natalie/Desktop/meltbk/smart_interface_energy/software/pc_interface/gpb.db
File Edit View Help
Database Structure
Browse Data
Execute SQL
Table: session
New Record
Delete Record
1 - 17 of 17
Go to: 0

	device	rate	spent	produced	user	end_time	start_time
1	4h45354	3	0	0	2342530	2010-03-04 14:14:59	2010-03-04 14:14:24
2	4h45354	3	0	1.15	2342530	2010-03-05 17:07:42	2010-03-04 14:41:45
3	4h45354	3	0	1.15	2342530	2010-03-05 17:07:42	2010-03-05 16:44:13
4	342ad54	-13	0	-320.883333333333	1234567	2010-03-05 17:07:42	2010-03-09 18:13:02
5	342ad54	3	24.3	0	1234567	2010-03-10 19:09:43	2010-03-10 11:03:31
6	4h45354	11	78.1	0	1234567	2010-03-10 19:10:10	2010-03-10 12:03:31
7	12g5354	22	134.2	0	1234567	2010-03-10 19:10:00	2010-03-10 13:03:31
8	342ad54	3	0.25	0	1234567	2010-03-10 19:17:09	2010-03-10 19:11:14
9	4h45354	11	0	0	1234567		2010-03-10 19:11:14
10	12g5354	22	0	0	1234567		2010-03-10 19:11:14
11	5sdf534	3	0	0	1234567		2010-03-10 19:11:14
12	CCA28DF7	2	0	0	1234567	2010-03-11 15:16:39	2010-03-11 15:15:58
13	CCA28DF7	2	0	0	1234567	2010-03-11 15:18:19	2010-03-11 15:17:39
14	1A873BB9	2	0	0	1234567		2010-03-11 15:19:35
15	CCA28DF7	2	0	0	1234567	2010-03-11 16:02:04	2010-03-11 16:01:51
16	CCA28DF7	2	0.2	0	1234567	2010-03-11 16:33:47	2010-03-11 16:27:38
17	CCA28DF7	2	0.2	0	1234567	2010-03-11 16:33:47	2010-03-11 16:28:50

Figure 29. Screen shot of the Session Table in the Database

After the user is done with the device, the database is updated with the end time of the transaction as well as how much energy the user has produced or spent. The parameters of the Session Table are found in Table 6.

Parameters	**Definitions**
device	what the device is connected to
rate	current price of electricity
spent	total amount spent on electricity
produced	total amount in the transaction produced
user	the user connected in this session
end_time	time finished with the transaction
start_time	time started for the transaction

Table 6. Parameters and Definitions of the Session Table in the Database

The Session Table logs all transactions done for all the users. Thus, it is useful for analyzing user patterns to predict future transactions as well as producing information about weekly energy productions and consumptions by the user.

6.4 Network Software

The Energy Mobility Network was coded in Python and C code. The Energy ID GUI prototype and the calculations for the Database were all created in Python and the firmware for the Outlet was done in Arduino then flash programmed into the microcontroller. It is best to explain the Server software first, before discussing the Active Energy ID GUI and the Outlet firmware.

6.4.1 Server Software

The software involved on the server side includes retrieving and sending information to the database, calculating the cost of using the device, and receiving information serially from the XBee chip.

Inserting and Receiving Information from the Database

All actions for the database require the DB-API 2.0 interface for SQLite databases. Inserting a new outlet definition to the Outlet Table can be done as shown:

```
def insertNewOutlet(self, user_rfid, name, profile, rate):        1
        '''Inserts a new outlet that is paired with a user '''      2
        conn = sqlite.connect(self.dbName)                          3
        cursor = conn.cursor()                                      4
        ts = TimeStamp.TimeStamp()                                  5
        cursor.execute('INSERT INTO outlet VALUES (?, ?, ?, ?)',    6
                          (user_rfid, name, profile, rate))         7
        conn.commit()                                               8
```

To connect to the database, a Connection object that symbolizes the database must be made as seen in line 3. In order to perform SQL commands for the database, the designer must call the execute() method of the cursor object created (lines 4, 6, 7). Inserting a new account for the device and user are done in the same fashion.

To receive information from the database, the same process is done as well, but the SQL command changes from INSERT INTO to SELECT * FROM. By using these particular commands, it is easy to access a plethora of information from the database. The functions created can be seen in Table 7.

Function	Description
insertNewOutlet	Inserts a new outlet that is paired with a user and his/her context
insertNewAcct	Inserts a new record (device or user) if a new RFID card id is detected
getData	Gets data from the requested table

Function	Description
getEnergy	Gets the amount of energy the user has produced at the time
getConnect	Locates what is connected to the device/user
idExists	Checks if the rfid of user/device is in the database
insertUpdate	Adds a new record to the device and user table stating the connection
deleteUpdate	After a session ends, the connections between the device and user are deleted
startSession	Starts a new session that links a device to a user at a certain kilowatt rate
calculateDiffTime	Calculates the time lapsed between the start and end of a transaction and returns the cost of the transaction
estimateEndSession	Estimates how much the transaction will be
sessionRate	Calculates the rate of the transaction
endSession	Ends the session between the user and device
convert	Converts the name to the equivalent RFID tag
convertid	Converts the RFID tag to the equivalent name
listDB	Lists all the users and devices stored in the database

Table 7. Functions to Access the Database

The cost of a transaction is calculated by multiplying the rate of the user's profile, the current electricity rate, and the time that the device was on. The code for the cost of the transaction as well as retrieving and transmitting information to the database can be found in Appendix A under database.py.

Receiving Information from the XBee Controller

Information is received from the XBee Controller by connecting serially to the specified USB port:

```
self.conn = serial.Serial(port = self.port, baudrate = 9600, timeout = 1)
```

Once a connection is made, the program waits until there is something to read in the Serial input. It uses regular expressions to parse the RFID tag information or the command that is sent. After computing the information, the program writes to the Serial port the data that needs to be sent back to the XBee End Device or Router. The code can be found in Appendix A under SerialReader.py.

6.4.2 Active Energy ID GUI Software

The Energy ID GUI used the Smart Interface structure to create the next GUI prototype for the user. PyQt4 was used to create the GUI structure and widgets; the code can be found in Appendix B and is labelled as Gui.py.

Initialization of GUI:

The initialization of the GUI includes the database that it is connected to, the username of the Energy ID, and any guests linked to the user's network. The serial reader connected to the Energy ID is initialized. Also, the six widgets of the GUI are added into the interface to create the basic structure of the Energy ID.

Profile Widget:

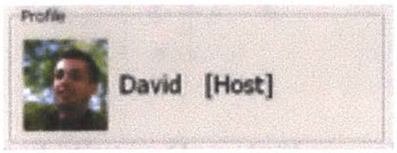


Figure 30. Profile Widget

The Profile Widget as seen in Figure 30 above is created by taking the Energy ID user and grabbing his/her picture from the imgs folder. The widget is created in a grid-like format; that is, the user's picture is set at the coordinates (0,0), the user's name is set at (0, 1), the user's profile is set at (0,2). The user's profile is obtained by searching for the closest outlet the user is by and obtaining the profile attached to the outlet as found in the Outlet Table in the database.

Network Widget:

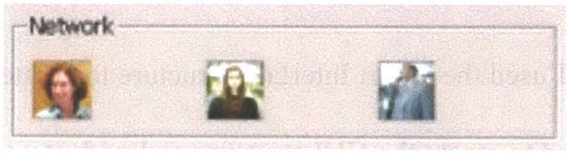


Figure 31. Network Widget

The Network Widget is initiated by grabbing three empty pictures and hiding it on the widget. When a guest taps into the system, the program determines if the serial information just sent is user information or device information by searching the device database to see if any record of the serial information pops up. If nothing shows, the program assumes that it is user information, in particular a guest's RFID. Consequently, the guest's image is grabbed from the `imgs` folder, replaces the empty image in the widget, and is set to be shown in the GUI.

New Device Widget:

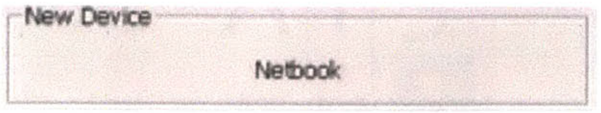
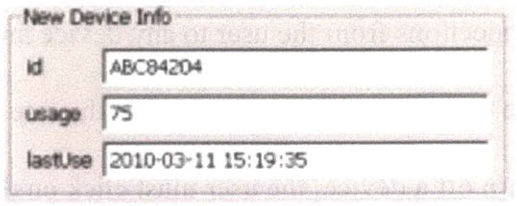


Figure 32. New Device Widget

This widget is a QGroupBox Widget. It has a label of "<no device>" when initialized, but when the program determines that the serial RFID information sent pertains to devices, the database is searched to find the device that matches the serial information. The name of the device replaces the "no device" label.

New Device Info Widget:

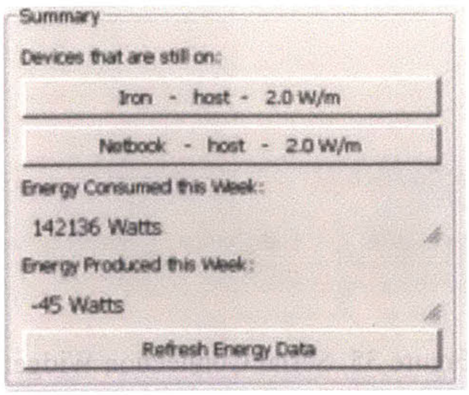


id	ABC84204
usage	75
lastUse	2010-03-11 15:19:35

Figure 33. New Device Info Widget

Similar to the New Device Widget, the New Device Info Widget uses a QGroupBox Widget and labels the box with device ID Tags ("id", "usage", "lastUse"). These remain blank until the server serially receives a RFID device tag. The device information is grabbed from the database and replaces the empty labels in the widget.

Summary Widget:



Summary

Devices that are still on:

Iron - host - 2.0 W/m

Netbook - host - 2.0 W/m

Energy Consumed this Week:

142136 Watts

Energy Produced this Week:

-45 Watts

Figure 34. Summary Widget

This widget uses the QGroupBox, QLabel, and QPushButtons. The QGroupBox creates the main layout of the Summary widget. QLabel shows the energy production and consumption data, which is grabbed from the database when the GUI is initialized and when the "Refresh Energy Data" button is pushed. The QPushButtons are used for the devices that are still on (in this case, the iron and the netbook) and the "Refresh Energy Data" as seen in Figure 34. When the GUI is initialized, the connections from the user to any device are found and created into buttons along with the profile that the user carries as well as the rate of electricity charged.

If the user wants to turn off a device, the user must click on the button detailing the device. This activates several things. First, the clicked button gets hidden from the GUI and sends a end transaction command to the database. The program then calculates the final cost of the transaction and updates the "Energy Consumed" or "Energy Produced" section of the widget. A command from the XBee Controller to the Router is sent to turn off the device that is connected to the Linksprite modules. It is important to note that when the GUI is initialized, two QPushButtons are hidden in the GUI so that the user can turn on up to two devices when using his/her Energy ID at that time.

Serial Connection Widget:

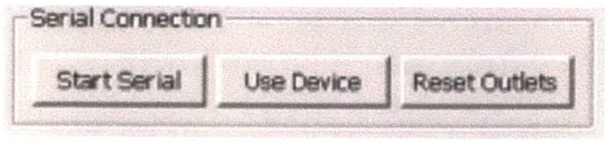


Figure 35. Serial Connection Widget

This widget uses QPushButtons to turn on and use the Energy ID. "Start Serial" allows serial information to be read into the program. If the user has already tapped in a device, the

"Use Device" button will activate a QMessageBox which asks about sharing the costs with the guests in the network. When sharing the costs with the guests, the program starts transactions with the other guests and will add a QPushButton of the device to the guests' Energy ID GUIs in addition to the user's own Energy ID GUI. The "Reset Outlets" button serves two purposes. When proposing the design to sponsors, the button was a demo button that was linked to a Flash page which explained the system to the sponsors. However, the "Reset Outlets" button is used to turn off all the devices linked to the user in the general mode of the system. This is done by hiding all QPushButtons of the devices that are connected to the user and ending all the user's transactions in the database. Also, the energy data is updated in the Summary box. The code for the two options of the "Reset Outlets" button as well as all of the other widgets can be found in Appendix B under Gui.py.

6.4.3 Outlet Board Software

The outlet board described is designed for passive users. During setup and initialization, the pins are connected to variables that signify control pins in the chips, variables are created for reading the RFID sensor, the serial connection is created, and the start of the finite state machine is introduced. There are eight stages in the finite state machine -- the transition from stage to stage is created through an outside action: when a RFID tag is tapped into the outlet or when a button is pressed by the user. In addition to the finite state machine, code is written to interact with the SPI interface, load and display the images to the Kent Display RAM, read the RFID tag from the ID-20, and communicate to the XBee and Linksprite. All of the code can be found in Appendix C.

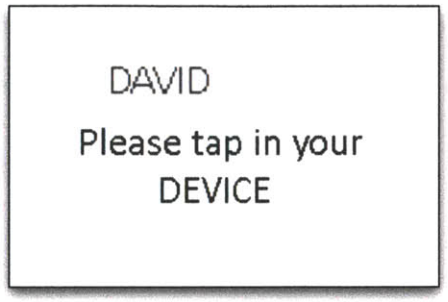
Finite State Machine

In the "Start" state, the Serial Peripheral Control Register for the Kent Display are set and the Kent Display is cleared. The images are loaded into the RAM of the ChLCD and the "Energy Mobility Network" is displayed on the screen. On the bottom of the screen, the user selects if he/she wants to use a remote device or if he/she wants to turn on a device that is attached to the outlet. The transition to the next state, state 0, is automatic.

In State 0, the system waits to receive an input from the user. If the user chooses to use the Remote Device feature, a screen is displayed asking if the device should be turned on or turned off. The system transitions to State 6, the state for remote devices. If the user chooses to use the device which is attached to the Outlet, the system requests his/her Energy ID. The system transitions to State 1.

In State 1, the outlet waits to receive to read RFID data, the Energy ID tag or Device tag , from the ID-20. Once the tag is present, the XBee Router sends the data to the XBee Controller to identify who the user or what the device is and the system transitions to State 2.

When the Router receives the identity from the Controller in State 2, the name of the user or device is then displayed on the screen. The system moves to State 3 only after both the user and device are detected. See Figure 36 for an example of what the screen displays.


DAVID

Please tap in your
DEVICE

Figure 36. State 2 of the Kent Display

State 3 is where the user sees the just-in-time message. Once a session has been connected in the database, the XBee controller sends information back to the outlet detailing the current and predicted costs of using the device. As seen in Figure 37, the user is given the choice to use the device now or later, depending on his/her preferences.

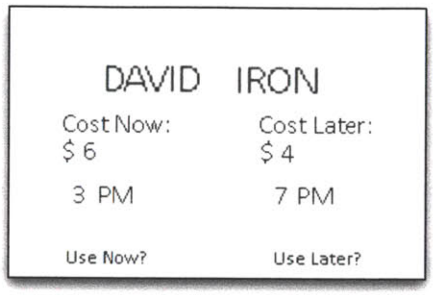


Figure 37. State 3 of the Kent Display

The Finite State Machine moves to State 4, irregardless if the user presses the “Use Now?” button below the text or if “Use Later?” is opted.

In State 4, there are two options. If the “Use Now?” button is pressed, the database is notified that the device and the user have entered a transaction stage. The text changes to show that the device is “now in use” as seen in Figure 38.

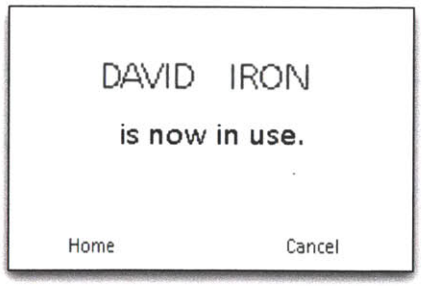


Figure 38. State 4 of the Kent Display when Device is in Use

Furthermore, the microcontroller activates the Linksprite controller to turn on the device. The current state becomes State 5. If the “User Later?” button is chosen, the database is notified that the session should be deleted and the connections between the user and device cancelled. The text changes to show that the device is not in use as seen in Figure 39.

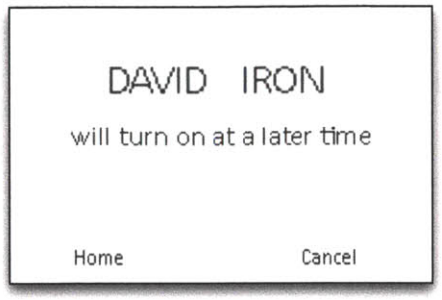


Figure 39. State 4 of the Kent Display when Device is not in Use

The microcontroller does not send any command to the Linksprite and the current state moves to State 5.

State 5 is in exit action when either the “Home” or “Cancel” button are pressed. If the “Home” button is selected, the Start State is returned and the Finite State Machine starts over again. However, if the “Cancel” button is selected, the current state changes depending on if the system is being used as a remote device or not. The XBee Router communicates to the database that the transaction has been ended and charges are calculated. In addition, the microcontroller commands the Linksprite to turn off the device.

State 6 is purely designed for remote device interaction. Here, it waits for the user’s input to decide if the remote device should be turned on or if the remote device should be turned off.

Interacting with SPI Interface

In order to communicate with the Kent Display, the ChLCD slave select pin must be pulled low before sending the command and pulled high after the command is sent. Commands are in hex and follow the hex commands found on the Kent Display datasheet. The functions to communicate with the Kent Display were found on Sparkfun's example code. A custom function was created to add the high byte to the low byte of a memory address to create a two byte address.

Writing and Reading Kent Display Images

In order to load data into the Kent Display RAM, the flash memory must be called instead of SRAM. Thus, PROGMEM is used. The hex values of the images, which are found through LCD Assistant, are cut and pasted into the PROGMEM array. Then, the array is loaded one hex value at a time to the display's memory in a similar fashion as the Kent Display datasheet. It is important to note that because of the limited space in the RAM, each image that is loaded onto the RAM does not signify the image that the user sees. Instead, each RAM image contains multiple images that the user may see as seen in Figure 40 and Figure 41.

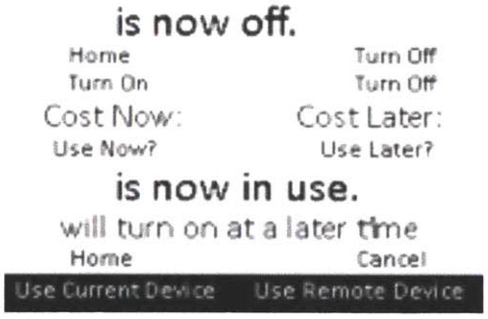


Figure 40. Image of Constants Loaded into Kent Display RAM

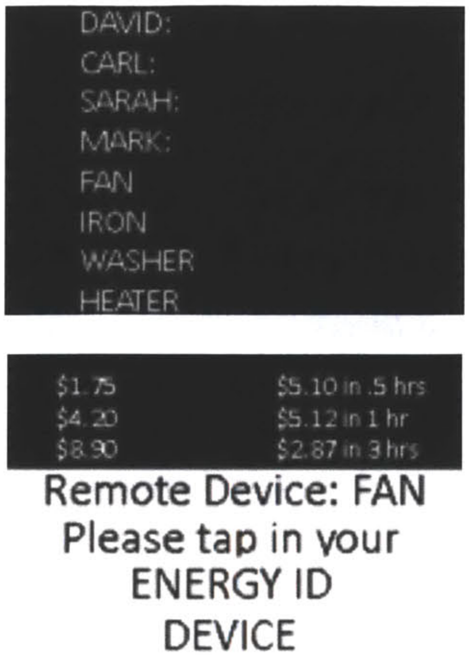


Figure 41. Images of Variables Loaded into Kent Display RAM

This is done to minimize the time taken to update a screen as well as to save as much space as possible in the RAM. In the Kent Display datasheet, there is a specific command that allows the designer to update part of the screen by cutting and pasting a range of rows from the RAM. This is particularly useful for uploading text that is constant such as the ones seen in Figure 40. In Figure 41, it is necessary not only to grab a range of rows, but also a range of columns of the image, to select the right variable name for the screen. This custom function is done by reading the data and pasting each hex into a part of the RAM allocated for receiving this specific function. After the selected text is written into the RAM, the program reads the RAM for a partial update to the screen. It is necessary to use this function because the partial update command found in the Kent Display reads full rows of data instead of specific sections of each row of data.

Reading the ID-20 Data

When sending the specified RFID tag to the XBee Router, the software serial library was used. Because the software serial library has minimal capabilities, each byte was read and saved into a variable before being sent to the XBee Controller. To confirm that the data is a RFID tag, the header of the data was checked before saving it into a variable.

Communicating with the XBee Coordinator and Linksprite

The XBee Router/End Device and Linksprite transmission lines are tied to the same pins on the ATMega644P through a multiplexer. Consequently, every time the outlet must talk with the XBee or Linksprite, the multiplexer control pins need to be changed to the particular module. After the digital writes to the control pins, the microcontroller can send and receive information serially from each component. It is best to keep the multiplexer control pins tied to the XBee so that the XBee is ready to receive any command sent from the server at any time. In the Arduino mode, the multiplexer was not needed and the XBee and Linksprite were hardwired together.

The XBee Router/End Device data received will be parsed using regular expressions to determine what kind of data is being received. The data will contain code words which will symbolize what data is sent so that the microcontroller will know what to do with the information. Such data may include the RFID tag of the user or device which is then sent to the Kent Display, the just-in-time message for the screen, or a remote command to turn on a device tied to the particular outlet.

6.4.4 System Software Testing

Testing the software requires creating test cases and checking if the returned answer matched the functionality of the definition. In particular, this is the case for testing out the server software. Each definition is checked by looking at the response printed on the serial port. For instance, to test the startSession definition, parameters were entered as such:

```
    db.startSession("12g5354", '22', "1234567")
```

The database is then evaluated to see if the Session Table now included the new record.

The GUI is tested in a similar fashion. An ID-20 sensor with a Sparkfun USB Breakout Board is connected serially and a RFID tag is tapped to the sensor to see if the correct device name would pop up on the GUI. When the "Use Device" button is clicked, the database checks to see if the new record is entered in as a transaction and if other guests, if any, are charged and at what rate. Finally, when the devices are turned off from the GUI, the device itself is checked to see if it is in the off state.

The outlet board is tested through user trials as well. The ATMega644P pins are measured to see if the digital writes to the pins are accurate. Each component is checked with a voltmeter to test the pin input or output. Most of the tests are not through software, but through a manual check of the components -- making sure that the Kent Display screen prints out the correct image, the XBee Router sends the right data to the Coordinator, the Linksprite Controller turns on/off the device, etc. All in all, the majority of the tests are done as user tests. I would act as the user, going through all the possible scenarios and making sure that none of the software broke as well as ensuring that the information produced is accurate.

6.5 Energy Etiquettes Sketch

Given the design and prototype discussed in Sections 6.1-6.4, the Energy Mobility Network has the ability to extend its functionalities to include the social realm. Because the system can be configured to calculate co-shared costs, such as when two people are watching TV together, social interactions have been created with the system. In this example, one person can treat the other to the cost of the activity, watching television -- similar to the case of a person buying a movie ticket for the companion on a date. This gesture, coined as an energy gesture, can be further extended to a person treating a group of people to an activity; a mother inviting her child's friends and family on a play date and paying for the electrical expense of the play date. In return, other mothers will be prone to extend a similar invitation as a gesture of thanks. This social interaction shows a new form of etiquette that will be created with the system. People can offer a gesture of gratitude by sharing their electricity with others.

The social energy etiquettes can be expanded with our system. Because the system has the ability to log what and when devices have been turned on, users could post their favorite or most used electricity producers or consumers to their friends. Friends could do multiple things with this information: they could gift a transaction for the user as a thank-you, learn about new devices, ask the user if they could try the device, suggest similar devices to the user that will lower the electricity cost or improve the electricity production, and more. Additionally, users can ask friends for help to reach their zero-sum balance for the month. By doing so, the users have created a bond among them, incentivizing each other to maintain the zero-sum balance as well as helping a friend reach their goal. In a similar fashion, if a user needs to charge a device or wash the dishes, another user may realize that the electrical rate has gone significantly down and remind the user to use the device at that time or even turn on the device for them. Small energy etiquettes like this strengthen the social friendships among users as well as creating a new social realm in their lives.

7. Results

After the design and technical specifications of the Energy Mobility Network, the prototype was created and analyzed to see if it effectively achieved the goals set in the beginning. Furthermore, the system is compared to other "smart" outlets available to see where it stands among the competitors.

7.1 Demonstration of the Energy Mobility Network

The design and fabrication of the Energy Mobility Network ended up in a prototype as seen in the following figures.

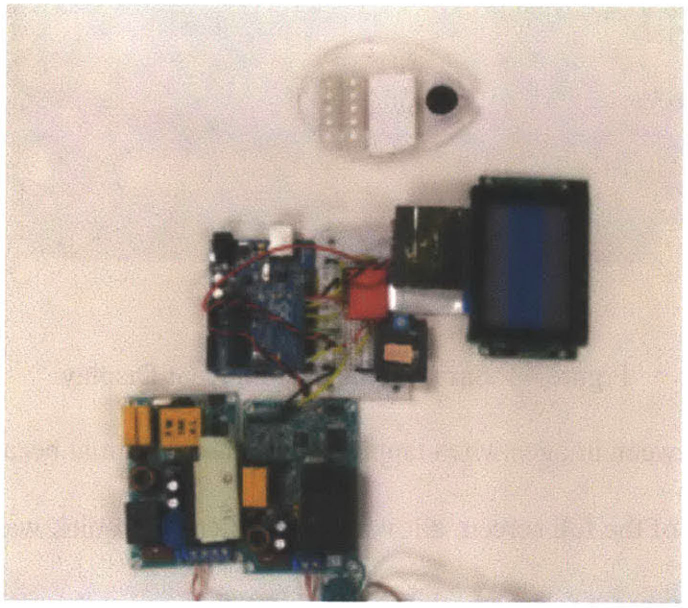


Figure 42. Final Prototype of the Energy Mobility Network Outlet

In Figure 42, the main circuit board is shown constructed, with the off-the-shelf components attached to it. To the right, is the Kent Display. Above the main PCB board is an example of the passive Energy ID which presents user's energy consumptions and productions. On the bottom, the Linksprite Outlet is connected to the device (not shown). With the outlet prototype, the user is able to communicate wirelessly with the server to turn on and off devices and gather data about his/her electricity productions and consumptions.

As a passive user, the Kent Display will be used to communicate with the server. As explained in Section 6.4.3, the user will see multiple images on the screen when using the outlet board; the images will display just-in-time prompt and will allow the user to turn on and off the attached device. A sample image can be seen in Figure 43.

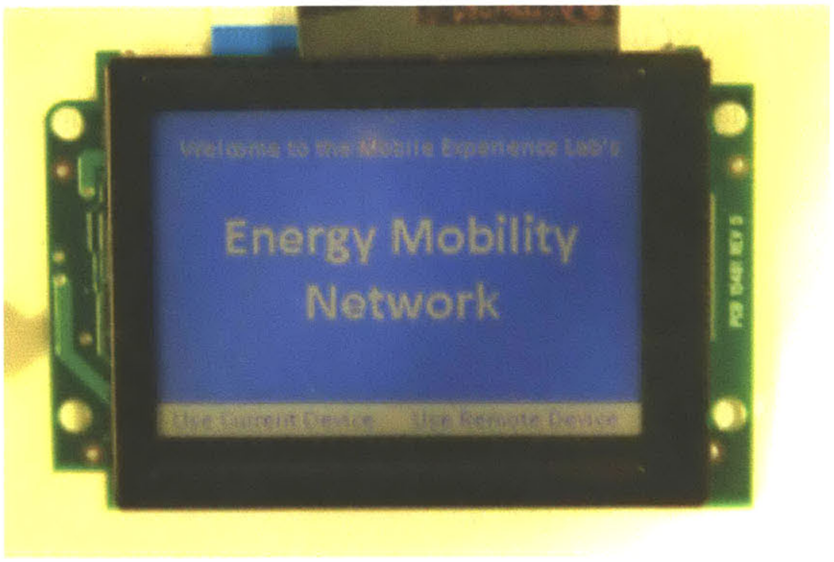


Figure 43. Sample Image from Kent Display

The transition time between images when tapping an RFID is minimal because the code updates partial screens instead of the full screen, allowing an efficient and quick way to turn on the outlet. The just-in-time message is essential with helping the user make an informed decision about his/her energy consumptions.

On the server side, the XBee Controller is attached to the computer via a USB XBee board as seen in Figure 44.

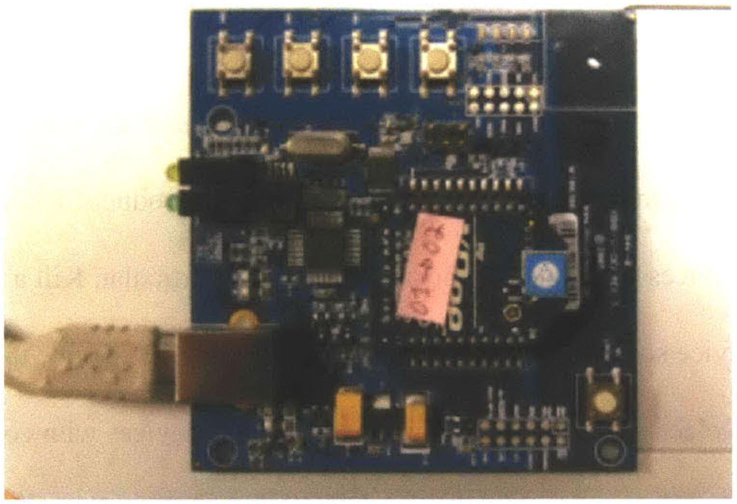


Figure 44. The Server: XBee Attached to the Computer

Because everything is wireless, the range with which active Energy ID gadgets communicate with the XBee Router on the Outlet or XBee Coordinator on the Server is above the Smart Interface; the system allows the user to have remote access to the outlet without having to go to the outlet to communicate to the network.

Finally, the active Energy ID GUI, has the capability of accessing the database to add and delete records of the transactions and sessions with or without guests sharing the device. The GUI shows the ability to create energy etiquettes because of the social network widget and the message box which appears when the user wants to use the device. Consequently, the Energy ID GUI will be essential in testing the social energy etiquettes with the users.

7.2 Comparing the System with Other Available Outlets

When analyzing the Energy Mobility Network to commercial smart outlets, the Energy Mobility Network excels with precise logging of transactions as well as informing the user of current and future costs. Furthermore, the proposed system has the ability to create a new social realm - a concept that is novel to the field.

Our system can record and analyze which devices are used the most based on the transaction history found in the database. In most commercial products, the user has to mentally take note of which devices are using the most electricity. In particular, Kill a Watt, counts the consumption of a device so that the user can personally analyze the device to see if the device is expending too much electricity. Our system provides a list of devices with consumption costs, which the system or user can analyze to see if any devices are expending too much electricity.

In most, if not all, of the commercial products, users receive data by going to the commercial product itself and viewing the information there. However, with the system that we have proposed, active users just look at their personal Energy ID. Consequently, this allows the users to have more flexibility and control with when they can view their consumption costs.

One product which does log data about energy consumptions is Wattson. It also uses passive lighting to depict how much electricity has been consumed in the day -- a feature which is not found in most appliances. However, Wattson does not give the user just-in-time prompts which could further reduce electrical costs for the user, a feature that the Energy Mobility Network promotes. Furthermore, Wattson gathers the amount of electricity consumed in the home, not for each device as done in our system. Although there is a community available to Wattson users which allow users to view the collective energy used, there is no social link among users. Users do not communicate among themselves; thus the social motivation of minimizing electrical consumptions is lost.

Commercial products related to our project provide the user with information about the user's appliances and the electricity consumption costs. These products have the capability of telling real-time information about electricity. For instance, Energy Aware's PowerTab In-Home Display produces the current accumulated electricity use. It also updates the user with upcoming pricing changes by producing different colored LEDs which signify a price change. However, unlike our system, it does not specify exact times where the price of electricity will be cheaper. Consequently, just-in-time prompt will give the user precise information about prices so the user knows the exact amount that he/she will save and spend. Using lights to signify price changes allows the user to have an estimate with what is saved, but depending on his/her needs, this may be too little information. Thus, the commercial products found are not as effective as the system created.

7.3 Privacy of the Data

Privacy of data is maintained in three manners: centralizing the stored information, adding a layer of abstraction to the user's location, and obscuring the Energy ID. Centralizing the data maintains privacy in the system. Each outlet does not contain any past information about the user or the device. Instead, the outlet is used to pass information through to the Energy ID when accessed. The server, which contains the database, stores and saves all the information.

In order to minimize breaching the user's privacy, the location of the user is defined but cannot be pin pointed. For instance, a school building can have many floors, which each have many rooms and many outlets installed. The user's profile can be defined as a student if he/she is studying in the building, a researcher if he/she is working for the school, or even a guest if he/she is accompanying a friend who works at the school. The profile does not specify the user's exact location, but gives the system enough information to charge a justifiable rate to the user. Consequently, the user's location is abstracted to fit his/her privacy.

The passive Energy IDs are camouflaged as everyday wear. As explained earlier, they resemble key fobs, phone dangles, watches, etc. Only the user knows what his/her Energy ID is -- minimizing possible identity thefts as well as allowing the user to personalize his/her Energy ID. For future work, the RFID tag and sensor should be carefully evaluated as it is easy for the user to be unaware of RFID sensors lifting his/her RFID tag when walking by. Such actions could lead to identity theft. It would be possible to add a layer of security by identifying the user through another set of means. This should be carefully researched as the user should seamlessly interact with the system.

7.4 Discussion of the Design Process

My focus for the Energy Mobility Network was primarily on the technical details. I had to learn new software languages to program the Arduino and access the database, familiarize myself with the circuit board design program as well as laying out a board, as well as apply the knowledge garnered from classes to produce a prototype. In addition, I also immersed myself in the design process of the system, which proved to be beneficial to creating the technical specifications of the project.

Much of the design process was a collaborative effort in the interdisciplinary group. Research was done on methods to motivate users, ideas to communicate to the system, as well as the user interface -- all researched in detail before presenting the information amongst ourselves.

Subsequently, main features were selected to represent the system: social interactions among users, just-in-time messages to inform the user, an accurate list of devices in use and the electrical consumptions and productions, and more. Such features were also determined by the technical ease and feasibility when prototyped. If an idea was could not be easily implemented technically, we worked around the technical aspects or reframed the idea to fit technical aspects.

Much of the social design was interwoven with the engineering process. Input from the team was taken to how the GUI would look to the user, what was the best way to stimulate the user and effectively get the data across. For instance, the phrases of the GUI widgets were phrased to have a positive connotation and the placement of the social gesture was deliberately placed as a message box to implicitly remind the user of social gestures.

Teammates often collaborated to come up with a design for blocks of the system. For example, as discussed in the passive Energy ID sketch, David Boardman came up the concept of the Energy ID, which Carl Yu implemented as a prototype. Research was also done on RFID form factors that could be integrated into the prototype. The collaborative effort achieved within our group allowed the Energy Mobility Network design to be systematically detailed.

8. Future Work

The work done to create the Energy Mobility Network is substantial, but there is always room for improvement. The design and goals of the system are solidified, as well as the general structure of the outlet board. However, the future work should be done to improve the Energy ID gadgets, device producers, further improving the outlet design, testing of the system, as well as adding more innovative features to the system.

Physical active Energy IDs should be made to include the GUI and functionalities designed in this version. These active Energy ID embodiments may be made into an application on a smart phone or a custom designed embodiment that users can carry around. As for passive Energy ID gadgets, the UV system in the outlet needs to be incorporated into the future version of the board to present the user's energy consumptions on the screen as well as on the passive Energy ID.

In addition to the ultraviolet lighting in the outlet, the next version of the outlet could include an SD slot to allow more images to be stored for the Kent Display. By doing so, the transition time between images will be faster and more text can be placed onto the outlet board and not transferred from the server. Additionally, it would be useful for the server software to include analysis of the user's transactions. This will allow users to garner information about what devices have been used the most, suggest times that users may use devices at a cheaper rate, and predict when certain devices will be used. This software can only be used with a lot of users tests completed, which was not done during the production phase. Consequently, it would be useful to place the prototypes inside the Connected Home to test the users' reactions as well as communications with the system for a month.

When placing the prototypes inside a building, it will be important to ensure the safety of the user. Currently, the Linksprite Outlet can only accept 2000 Watts, which does not cover all of the electrical devices in the market. Thus, when using devices with the system, caution must be taken with the maximum power rating of the device.

The subsequent steps to the prototype discussed is to add the SD slot to allow more images to be displayed on the Kent ChLCD screen. After integrating the additional space to the outlet, it would be be best to gather user tests from the prototypes. By setting the prototypes in a room and allowing electrical devices under 2000 Watts to be plugged into the outlets, the system will be able to gather real data from the users. This while provide the system with information to analyze and predict users' patterns as well as help debug any software or hardware problems which users may face. More importantly, doing these tests will allow us to evaluate how well the social energy etiquettes as well as potential etiquettes that were not hypothesized in the thesis. The user will be able to give personal feedback on the system; allowing the designer to take note of potential pitfalls that may have been overlooked.

While proceeding with the tests, the designer can create the physical active Energy ID for the users as well as the UV system for the outlet. These should be used in the second phase of testing; after analyzing how users communicate with the system and improving the system based on the users' comments.

9. Conclusion

The Energy Mobility Network presents users with a new way to view their energy expenditures. Instead of waiting for a monthly electric bill to arrive, users are given just-in-time message about their potential transactions with devices. Such data allows users to make an informed decision with how much they are willing to spend on using the device versus the necessity of the device at the time. Users communicate with the system through their personal Energy ID gadget - a product tailored to the user, which allows them to view their weekly electrical consumption and production costs. Designing a centralized system was useful because it allowed data to pass through a main hub as well as enforced data integrity. Unlike any other products today, the system has created a new realm of social etiquettes. Users can now treat other users to electricity – thus creating a new kind of social network. The system allows users to have a more in depth knowledge about their electricity consumptions as well as opens a new concept of energy gestures and etiquettes to the users.

Appendix A - Server Code

Database.py

```
import time
import datetime
from datetime import datetime, date, time, timedelta
from pysqlite2 import dbapi2 as sqlite
import TimeStamp
import re

'''
sql comments:
ALTER TABLE users ADD COLUMN usage int NULL
'''
class DataBase:

    '''Creates the interface for the sqlite3 database'''

    def __init__(self, dbName):

        self.dbName = dbName
        self.cursor = None
        self.openDB(dbName)

    def createDB(self):
        '''Database is already created'''
        pass

    def openDB(self, dbName):
        try:
            self.conn = sqlite.connect(dbName)
            self.cursor = self.conn.cursor()
        except:
            print 'Cannot create DB connection for: ', dbName

    def listDB(self):
        '''Lists all the users and devices stored in the database'''
        if self.cursor != None:
            self.cursor.execute('select * from user')
            print 'All Users'
            for sq in self.cursor:
                print row
            print 'All Devices'
            self.cursor.execute('select * from device')
            for row in self.cursor:
                print row

    def insertNewOutlet(self, user_rfid, name, profile, rate):
       '''Inserts a new outlet that is paired with a user and the context the user is
in '''
       conn = sqlite.connect(self.dbName)
        cursor = conn.cursor()
        ts = TimeStamp.TimeStamp()
        cursor.execute('INSERT INTO outlet VALUES (?, ?, ?, ?)',
                           (user_rfid, name, profile, rate))
        conn.commit()

    def insertNewAcct(self, acct_type, name, location, bank, connect, rfid):
          '''Inserts a new account (device or user, based on acct_type) if a new RFID
card id is detected
        As the class is used in a multi-thread environment
        each time we created a new connection with the database
        '''
        conn = sqlite.connect(self.dbName)
        cursor = conn.cursor()
        ts = TimeStamp.TimeStamp()
        if acct_type == 'device':
            cursor.execute('INSERT INTO device VALUES (?, ?, ?, ?, ?, ?)',
                           (name, location, bank, connect, rfid, ts.getTime()))
        elif acct_type == 'user':
            cursor.execute('INSERT INTO user VALUES (?, ?, ?, ?, ?, ?)',
                           (name, location, bank, connect, rfid, ts.getTime()))
        else:
            print 'Type of account is not found'
        conn.commit()

    def getData(self, acct_type, data, outlet_data):
        ''' Gets data from the specific acct_type table'''
        conn = sqlite.connect(self.dbName)
        cursor = conn.cursor()
        if acct_type == 'device':
            cursor.execute('SELECT * FROM device WHERE connect IS ?', (None,))
            for row in cursor:
                length = len(row)
                for unit in range(length):
                    if row[unit] == data:
                        return row
        elif acct_type == 'user':
            cursor.execute('SELECT * FROM user WHERE connect IS ?', (None,))
            for row in cursor:
                length = len(row)
                for unit in range(length):
                    if row[unit] == data:
                        return row
        elif acct_type == 'outlet':
            cursor.execute('SELECT * FROM outlet WHERE name IS ?', (outlet_data,))
            for row in cursor:
                length = len(row)
                for unit in range(length):
                    if row[unit] == data:
                        return row
        conn.close()

    def getEnergy(self, user_id, action):
        ''' gets how much energy the user has produced at that moment'''
        conn = sqlite.connect(self.dbName)
        cursor = conn.cursor()
        produced = 0
        spent = 0
        cursor.execute('SELECT * FROM session WHERE user IS ?', (user_id,))
        for row in cursor:
            produced = produced + row[3]
            spent = spent + row[2]
        if action == "produced":
            return produced
        if action == "spent":
            return spent
        else:
            print "produced or spent is not the value of action"

    def getConnect(self, acct_type, data):
        ''' Gets what is connected to the device/person'''
        conn = sqlite.connect(self.dbName)
        cursor = conn.cursor()
        lst_connect = []
        if acct_type == 'device':
            cursor.execute('SELECT * FROM device WHERE connect IS NOT ?', (None,))
            for row in cursor:
                length = len(row)
                for unit in range(length):
                    if row[unit] == data:
                        connected = self.getData("user", row[3], None)
                        lst_connect.append(connected[0])

        elif acct_type == 'user':
            cursor.execute('SELECT * FROM user WHERE connect IS NOT ?', (None,))
            for row in cursor:
                length = len(row)
                for unit in range(length):
                    if row[unit] == data:
                        connected = self.getData("device", row[3], None)
                        lst_connect.append(connected[0])
        conn.close()

        return lst_connect


    def idExists(self, rfid):
        '''
        checks if rfid is in the database. if it is, outputs a 1, prints 'True'
        '''
        conn = sqlite.connect(self.dbName)
        cursor = conn.cursor()
        count = 0
        cursor.execute('SELECT * FROM device WHERE rfid IS ?', (rfid,))
        for row in cursor:
            if len(row) != 0:
                #print row
               #print 'TRUE'
                return 1
        #print 'FALSE'
        return 0

        conn.close()


    def insertUpdate(self, device, user):
        ''' adds a row to the device and user table stating what it is connected to'''
        conn = sqlite.connect(self.dbName)
        cursor = conn.cursor()

        ts = TimeStamp.TimeStamp()

        print device

        print self.getData("device", device, None)

        [device_name,device_location,device_bank,
              device_connect,device_rfid,device_time] = self.getData("device", device,
None)
        [user_name,user_location,user_bank,
         user_connect,user_rfid,user_time] = self.getData("user", user, None)

        conn.close()

              self.insertNewAcct("device", device_name, device_location, device_bank,
user_rfid, device_rfid)
          self.insertNewAcct("user", user_name, user_location, user_bank, device_rfid,
user_rfid)
        # maybe add a error thing if device / user are not in database?
        # maybe add a checking to see if device and user already connected?

        print "Added Connection"

    def deleteUpdate(self, device_id, user_id):
        ''' after a session is ended, the connections between
        device and user in the "device" and "user" table are deleted'''
        conn = sqlite.connect(self.dbName)
        cursor = conn.cursor()
        cursor.execute('DELETE FROM device WHERE (connect = ? AND rfid = ?)',
                       (user_id, device_id))
        cursor.execute('DELETE FROM user WHERE (connect = ? AND rfid = ?)',
                       (device_id, user_id))
        conn.commit()
        print "Deleted Connection"

    def startSession(self, device_id, rate, user_id):
        ''' starts a new session that links a device to a user at a certain kw rate'''
        conn = sqlite.connect(self.dbName)
        cursor = conn.cursor()
        ts = TimeStamp.TimeStamp()
        cursor.execute('INSERT INTO session VALUES (?, ?, ?, ?, ?, ?, ?)',
                       (device_id, rate, 0, 0, user_id, '', ts.getTime()))
        conn.commit()
        ## change user and device table s.t. the user is connected to device
        print device_id
        print user_id
        print "session will start"
        self.insertUpdate(device_id, user_id)
        print "Session started"

    def calculateDiffTime(self, start_time, end_time, rate):
##        print "Start time"
##        print start_time
##        print "end_time"
##        print end_time
              split_start_time = re.match(r"(\d+)\-(\d+)-(\d+).(\d+)\:(\d+):(\d+)",
start_time)
        split_end_time = re.match(r"(\d+)\-(\d+)-(\d+).(\d+)\:(\d+):(\d+)", end_time)
        [year1, month1, day1, hour1, minute1, second1] = split_start_time.groups()
        [year2, month2, day2, hour2, minute2, second2] = split_end_time.groups()
##        print "split_start_time"
##        print split_start_time.groups()
##        print "split_end_time"
##        print split_end_time.groups()

        d1 = date(int(year1), int(month1), int(day1))
        t1 = time(int(hour1), int(minute1), int(second1))

        d2 = date(int(year2), int(month2), int(day2))
        t2 = time(int(hour2), int(minute2), int(second2))


        totaltime = datetime.combine(d2,t2)-datetime.combine(d1,t1)
##        print "totaltime"
##        print totaltime
##
##        print (year2 == year1)
##        print (day1 != day2)
##        print (month2 == month1)


        split_time = re.match(r"(\d+):(\d+)",str(totaltime))
        if (split_time == None): ## more than 1 day
```

```
            split_time = re.match(r"(\d+).day.* (\d+):(\d+)",str(totaltime))
            print "SPLIT GROUP2"
            [days, hours, minutes] = split_time.groups()
        else:
            [hours, minutes] = split_time.groups()
            days = 0

        time_minutes = int(days)*1440+int(hours)*60+int(minutes)

        cost = float(time_minutes)*rate

        return cost

    def estimateEndSession(self, device_id, user_id):
        ''' guesses how much a session will cost'''
        conn = sqlite.connect(self.dbName)
        cursor = conn.cursor()

        ts = TimeStamp.TimeStamp()
        time = ts.getTime()

        outlet_location_row = self.getData("device", device_id, None)
        profile_row = self.getData("outlet", user_id, outlet_location_row[1])


         cursor.execute('SELECT * FROM session WHERE (end_time == ? AND device = ? and
user = ?)',
                       ('', device_id, user_id))
        for row in cursor:
            start_time = row[6]
            rate = float(row[1])*float(profile_row[3])
            cost = self.calculateDiffTime(start_time, time, rate)
        conn.commit()

        return cost

    def sessionRate(self, device_id, user_id):
       '''Calculates the rate of the transaction'''
        conn = sqlite.connect(self.dbName)
        cursor = conn.cursor()

        ts = TimeStamp.TimeStamp()
        time = ts.getTime()

        outlet_location_row = self.getData("device", device_id, None)
        profile_row = self.getData("outlet", user_id, outlet_location_row[1])


         cursor.execute('SELECT * FROM session WHERE (end_time == ? AND device = ? and
user = ?)',
                       ('', device_id, user_id))
        for row in cursor:
            start_time = row[6]
            rate = float(row[1])*float(profile_row[3])
        conn.commit

        return rate

## cost will be positive for spent, negative for produced
        print "estimate session complete"



    def endSession(self, device_id, user_id):
        ''' ends a session that links a device to a user '''
        conn = sqlite.connect(self.dbName)
        cursor = conn.cursor()

        ts = TimeStamp.TimeStamp()
        time = ts.getTime()

        outlet_location_row = self.getData("device", device_id, None)
        profile_row = self.getData("outlet", user_id, outlet_location_row[1])


         cursor.execute('SELECT * FROM session WHERE (end_time == ? AND device = ? and
user = ?)',
                       ('', device_id, user_id))
        for row in cursor:
            start_time = row[6]
            rate = float(row[1])*float(profile_row[3])
            cost = self.calculateDiffTime(start_time, time, rate)

            if (rate > 0):
                 cursor.execute('UPDATE session SET spent = ?  WHERE (end_time == ? AND
device = ? AND user = ?)',
                      (cost, '',device_id, user_id))
            if (rate < 0):
                  cursor.execute('UPDATE session SET produced = ?  WHERE (end_time == ?
AND device = ? AND user = ?)',
                      (cost, '',device_id, user_id))
             cursor.execute('UPDATE session SET end_time = ?  WHERE (end_time == ? AND
device = ? AND user = ?)',
                      (time, '',device_id, user_id))

            ## update bank acct for user
            user_row = self.getData("user", user_id, None)
            cost_before = int(user_row[2])
            total_cost = cost + cost_before
            #print "total  " + str(total_cost)


                cursor.execute('UPDATE user SET bank = ? WHERE (connect IS ? AND rfid
= ?)',
                           (total_cost, None, user_id))

        conn.commit()
        ## change user and device table s.t. the user is NOT connected to device
        self.deleteUpdate(device_id, user_id)

        print "Session completed"

    def convert(self,name):
        ''''convert name to rfid'''
        answer = self.getData("device", name, None)
        if answer == None:
            answer = self.getData("user",name, None)
        return answer[4]

    def convertid(self,name):
        ''''convert rfid to name'''
        answer = self.getData("device", name, None)
        print answer
        if answer == None:
            answer = self.getData("user",name, None)
        return answer[0]
```

```
if __name__ == "__main__":
        db = DataBase('ppb.db')
##        db.insertNewOutlet("1234567", "home outlet", "host", '1')
##        db.insertNewOutlet("1234567", "lab outlet", "student", '.8')
##        db.insertNewOutlet("1234567", "guest outlet", "guest", '.3')
##        db.insertNewOutlet("1234567", "bathroom outlet", "host", '1')
        #print db.getData("outlet", "1234567", "guest outlet")

##        #db.insertUpdate("gameboy", "David")
##        #db.endSession("2f4df5213", "1234567")
##        #db.deleteUpdate("gameboy", "1234567")
##        #db.endSession("12g5354", "2sd3431")
####        #db.getData("device", "12g5354")
        db.startSession("342as54", '3', "1234567")
        db.startSession("4h45354", '-11', "1234567")
        db.startSession("12g5354", '22', "1234567")
        db.startSession("5sdf534", '-3', "1234567")
##        #db.endSession("342as54", "1234567")
##        #db.calculateDiffTime('2010-01-04 14:42:17', '2011-03-04 14:45:17', 6)
##        #db.getConnect("user", "David")
##        #db.convert('David')
        #print db.convertid("CCA2BDF7")
```

SerialReader.py

```
import re
import time
import random
from xbee import XBee
import serial

from PyQt4 import QtCore

from pysqlite2 import dbapi2 as sqlite

class Timer(QtCore.QThread):

    def __init__(self, parent=None):

        QtCore.QThread.__init__(self)

        print 'Timer Created'
        self.prevTime = time.time()
        self.newTime = None
        self.ready = True

    def run(self):
        self.newTime = time.time()
        diff = self.newTime - self.prevTime
        print 'diff', diff
        if diff > 5:
            self.ready = True
            self.prevTime = self.newTime
        else:
            self.ready = False


class SerialReader(QtCore.QThread):

    def __init__(self, parent=None, gui=None):

        QtCore.QThread.__init__(self)

        print 'Serial Reader Created'
        self.gui = gui
        self.defaultCredit=60

        self.xbee = None

        self.newUser = None
        self.newDevice = None

        self.newID = None

        self.timer = Timer()

        try:
            rfid = serial.Serial(port = 46,
                                 baudrate = 9600,
                                 timeout = 1)
            self.xbee = XBee(rfid)
        except:
            print "\tno rfid connection"


    def run(self):
        while True:
            try:
                print '<-rfid'
##

                ## grabs data from XBee Controller
                response = self.xbee.wait_read_frame()
                print response
                ## Finds Name of User/Device in Database, sends data to Xbee Router
                if (response["id"] == "rx"):
                    response_split= re.split('Code: ', response["rf_data"])
                    if (len(response_split) == 1): # not RFID match
                            ## print "Not a RFID Code. Check if it is a LS Cmd for the
outlet"
                        linksprite_data = re.split('AT', response["rf_data"])
                        if (len(linksprite_data) == 1):
                            ## check if it is a Remote Code:
                            remote_split = re.split('REMOTE: ', response["rf_data"])
                            if (len(remote_split) == 1):
                                print "disregard"
                                print remote_split
                            else:
                                print "Remote Control On"
                                if (response["source_addr"] == 'Vx'):
                                    print "outlet info from ID 101 outlet"
                                    print remote_split
                                    if (remote_split[1] == 'ON'):
                                        time.sleep(2)
                                        self.xbee.send('tx',
                                                       frame_id='A',
                                                       dest_addr='\x56\x79',
                                                       data = '<RATON 0\n>')
                                    elif (remote_split[1] == 'OFF'):
                                        time.sleep(2)
                                        self.xbee.send('tx',
                                                       frame_id='A',
                                                       dest_addr='\x56\x79',
                                                       data = '<RATOF 0\n>')
                                elif (response["source_addr"] == 'Vy'):
                                    print "outlet info from ID 0 outlet"
                                    if (remote_split[1] == 'ON'):
                                        time.sleep(2)
                                        self.xbee.send('tx',
                                                       frame_id='A',
                                                       dest_addr='\x56\x78',
                                                       data = '<RATON 101\n>')
                                    elif (remote_split[1] == 'OFF'):
                                        time.sleep(2)
                                        self.xbee.send('tx',
                                                       frame_id='A',
                                                       dest_addr='\x56\x78',
                                                       data = '<RATOF 101\n>')

                        else:

                            print "direct LS command"
                            print linksprite_data[1]
##                                  if (!(self.newUser == None) && !(self.newDevice ==
None)):
##                                session_data = re.split(" ", linksprite_data[1])
##                                print session_data
##                                if (session_data[0] == "ON"):
##                                         self.gui.db.startSession(self.newDevice, .8,
self.newUser)
##                                else:
##                                               self.gui.db.endSession(self.newDevice,
self.newUser)
##
##                                self.newDevice = None
##                                self.newUser = None

                    else: ##is a RFID code
                        xbee_data = re.split('\r\n', response_split[1])
                        xbee_data = xbee_data[0]
                        ##print xbee_data
                        device = self.gui.db.getData("device", xbee_data, None)
                        if device == None:
                            # user data
                            user = self.gui.db.getData("user", xbee_data, None)
                            length = 5 - len(user[0])
                            spaces = ""
                            for i in range(0, length, 1):
                                spaces = spaces + " "
                            answer = '<U:' + str(user[0]) + spaces + '>'

                            # for the GUI update
                            self.newUser = user[0]
                        else:
                            length = 5 - len(device[0])
                            spaces = ""
                            for i in range(0, length, 1):
                                spaces = spaces + " "
                            answer = '<D:' + str(device[0]) + spaces + '>'

                            # for the GUI update
                            self.newDevice = device[0]
                        if (response["source_addr"] == 'Vx'):
                            time.sleep(2)
                            self.xbee.send('tx',
                                           frame_id='A',
                                           dest_addr='\x56\x78',
                                           data = answer)

                        if (response["source_addr"] == 'Vy'):
                            time.sleep(2)
                        self.xbee.send('tx',
                                       frame_id='A',
                                       dest_addr='\x56\x79',
                                       data = answer)

                if (response["id"] == "tx_status"):
                    print "sent tx data"

                ## Updates GUI
            except:
                print "No valid reading"
                raise
```

OutletTester.py

```
import serial
from random import randint

print "Establishing Communication:"

ser = serial.Serial(port = 32,
                    baudrate = 9600,
                    timeout = 1, rtscts = 0)
print "Which port is used: ", ser.portstr

input_=''
input_2=''
x = 1

while 1:
    command = raw_input("> ")
  #  number = command.split()[1]
    command = (command + '\n')
    ser.write(command)
    print 'writing: ', command
    input = ser.readline()
    print 'reading input_: ', input
    input_2 = ser.readline()
    print 'reading input_2: ', input_2


#    if "On" in input_:
        # create a text file
#        print 'read:', input_2
 #       input_2=''
 #       text_file = open("%s.txt" % (number) , "w")
  #      text_file.write("%s\n" % number )
  #      text_file.close()
  #      print 'number_found and saved'


while 0:
    for  x in range(5000, 10000):
        number = x
        command = ('ATOF ' + str(number))
        ser.write(command + '\n')
        print 'writing: ', command       # if "on" in ser.readline():
            # create a text file
       #     text_file = open("%s.txt" % (number) , "w")
       #     text_file.write("%s\n" % number )
        #    text_file.close()
       #     print 'number_found and saved'
```

Appendix B - Active Energy ID GUI Code

```
Gui.py

#ALTER TABLE users ADD COLUMN label varchar(10)

from PyQt4 import QtCore, QtGui
import re
import sys
import DataBase
import SerialReader
import TimeStamp
import socket

HOST = '127.0.0.1'
PORT = 10200
BUFSIZ = 1024
ADDR = (HOST, PORT)



class MyThread(QtCore.QThread):
    def run(self):
                n = 0
                step = 1
                while True:
                        n += step
                        print n


class Gui(QtGui.QWidget):

    def __init__(self, parent=None, dBase=None, username=None, guest=None):

        QtGui.QGroupBox.__init__(self, parent)

        self.defaultCredit = 60
        self.deviceIdTags = ['id', 'usage', 'lastUse']
        self.deviceIdElements = {'id': [],
                                 'usage': [],
                                 'lastUse': []}
        self.device_rfid = []
        self.outlets = {'1':[],
                        '2':[],
                        '3':[]}

        self.input_ = None
        self.db = dBase
        self.timeStamp = TimeStamp.TimeStamp()
        ##
        self.user = db.getData("user",username, None)
        print "David User"

        print self.user

        self.setWindowTitle('Energy ID Interface')
        self.setWindowOpacity(1)
        self.setFont(QtGui.QFont("CorporateS-Regular", 10.5) )

        self.buildGUIElements()

        self.serialReaderThread = SerialReader.SerialReader(gui=self)

        mainLayout = QtGui.QVBoxLayout()
        mainLayout.addWidget(self.energyManagementBox)
        mainLayout.addWidget(self.GuestBox)
        mainLayout.addWidget(self.horizontalGroupBox1)
        mainLayout.addWidget(self.deviceInfoGroupBox)
        mainLayout.addWidget(self.distributionInfoGroupBox)
        mainLayout.addWidget(self.horizontalGroupBox2)


        self.setLayout(mainLayout)

        mainLayout.setEnabled(True)


## 11/30/2010 i dont remember what the commented out thing does
##        if (guest == None):
##            s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
##            s.bind((HOST, PORT))
##            s.listen(1)
##            self.conn, addr = s.accept()

        self.demo = "0"


    def useOutletButtonAction(self):

        reply = QtGui.QMessageBox.question(self, 'Social Gesture',
                                                        "Do you want to share the cost with
guests?",
                                                                     QtGui.QMessageBox.Yes,
QtGui.QMessageBox.No)
        if reply == QtGui.QMessageBox.Yes:
            '''add button to guest and change rate!'''
            print self.demo
            print "SELF DEMO"
            if self.demo == "1":
                data = "100"
                self.conn.send(data + '\0')


            if (self.emptyButton.isVisible()):
                device_name = str(self.nameLine2.displayText())
            else:
                device_name = str(self.nameLine.displayText())

            if self.guestpic3.isVisible():
                matched_name = re.match("imgs\/m([a-zA-Z]+)", str(self.guestimage3))
                guest_name = matched_name.groups()[0]
                db.startSession(device_name, '1', db.convert(guest_name))
                    guest3 = Gui(dBase = self.db, username = str(guest_name), guest =
"Yes" )
                guest3.GuestBox.hide()
                guest3.deviceInfoGroupBox.hide()
                guest3.horizontalGroupBox2.hide()
                guest3.horizontalGroupBox1.hide()
                guest3.show()

            if self.guestpic2.isVisible():
                matched_name = re.match("imgs\/m([a-zA-Z]+)", str(self.guestimage2))
                guest_name = matched_name.groups()[0]
                db.startSession(device_name, '1', db.convert(guest_name))
                    guest2 = Gui(dBase = self.db, username = str(guest_name), guest =
"Yes" )
                guest2.GuestBox.hide()
                guest2.deviceInfoGroupBox.hide()
                guest2.horizontalGroupBox2.hide()
                guest2.horizontalGroupBox1.hide()
                guest2.show()
            if self.guestpic.isVisible():
                matched_name = re.match("imgs\/m([a-zA-Z]+)", str(self.guestimage1))
                guest_name = matched_name.groups()[0]
                db.startSession(device_name, '1', db.convert(guest_name))
                    guest1 = Gui(dBase = self.db, username = str(guest_name), guest =
"Yes" )
                guest1.GuestBox.hide()
                guest1.deviceInfoGroupBox.hide()
                guest1.horizontalGroupBox2.hide()
                guest1.horizontalGroupBox1.hide()
                guest1.show()
            else:
                print "There are no guests to share this with!"

        else:

            print self.demo
            print "SELF DEMO"

            if self.demo == "1":
                data = "100"
                self.conn.send(data + '\0')

        if (self.emptyButton.isVisible()):
                      self.db.startSession(str(self.nameLine2.displayText()), '2', str
(self.db.convert(self.user[0])))
            id_ = self.nameLine2.displayText()
            device_text = []
            device_text.append(str(self.db.convertid(id_)))
            device_text.append(str((db.getData("outlet", self.user[4],
                                                    (db.getData("device", db.convert
(self.db.convertid(id_)), None))[1]))[2]))
            device_text.append(str(db.sessionRate(db.convert(id_), self.user[4])))
             self.nameLine2.setText(device_text[0] + "    -    " + device_text[1]+ "    -
" + device_text[2]+" W/m")
            self.emptyButton2.setText(self.nameLine2.displayText())
            self.emptyButton2.show()
                            self.smart_connect(self.emptyButton2,  self.amtEnergy,
self.device_off_action, self.db.convert(id_)) #fixlater

        else:
                       self.db.startSession(str(self.nameLine.displayText()), '2', str
(self.db.convert(self.user[0])))
            id_ = self.nameLine.displayText()
            device_text = []
            device_text.append(str(self.db.convertid(id_)))
            device_text.append(str((db.getData("outlet", self.user[4],
                                                    (db.getData("device", db.convert
(self.db.convertid(id_)), None))[1]))[2]))
            device_text.append(str(db.sessionRate(db.convert(id_), self.user[4])))
              self.nameLine.setText(device_text[0] + "    -    " + device_text[1]+ "    -
" + device_text[2]+" W/m")
            self.emptyButton.setText(self.nameLine.displayText())
            self.emptyButton.show()
                            self.smart_connect(self.emptyButton,  self.amtEnergy,
self.device_off_action, self.db.convert(id_)) #fixlater

    def demoAction(self):

        print self.demo
        print "SELF DEMO"
        if self.demo == "1":
            self.demo = "0"
            data = "reset"
            self.conn.send(data + '\0')
        else:
            self.demo = "1"
            data = "reset"
        print "done changing demo action"
        print self.demo


    def startSerialReadButtonAction(self):
        self.serialReaderThread.run()
        if self.demo == "1":
            data = "start serial"
            self.conn.send(data + '\0')

    def device_off_action(self,device, amtEnergy, device_text):
        print "Device_text"
        print device_text
        print len(device_text)
        rfid = self.db.convert(device_text)
        print "Rfid = "
        print rfid
        print len(rfid)
        self.db.endSession(rfid, self.user[4])
        #print str(device.text()) + " now off"
        device.close()
        energy = db.getData("user", self.user[0], None)
        amtEnergy.showMessage(str(energy[2]) + " Watts")
        self.refreshdata()


        print self.demo
        print "SELF DEMO"


        if self.demo == "1":
            data = "z"
            self.conn.send(data + '\0')

     def smart_connect(self, device_on, amtEnergy, device_off_action, device_text): ##
got from online
                proxy_slot  =  lambda  checked:  device_off_action(device_on,  amtEnergy,
device_text)
        device_on.clicked.connect(proxy_slot)

    def refreshdata(self):
        connected = db.getConnect("user", self.user[0])

        energy = db.getEnergy(self.user[4], "spent")
        energy2 = db.getEnergy(self.user[4], "produced")
        for device in connected:
            cost = self.db.estimateEndSession(self.db.convert(device), self.user[4])
            print device
            print cost
            if cost > 0:
                energy = energy + cost
            elif cost < 0:
                energy2 = energy2 + cost
            else:
                print "cost is zero right now"
        self.amtEnergy.showMessage(str(int(energy)) + " Watts ")
        self.amtEnergy2.showMessage(str(int(energy2*-1)) + " Watts ")

    def createDistributionInfo(self):

        self.distributionInfoGroupBox = QtGui.QGroupBox("Summary")
        layout = QtGui.QGridLayout()
        userLabel2 = QtGui.QLabel(self)
        userLabel2.setText("Devices that are still on: ")
        userLabel2.setAlignment(QtCore.Qt.AlignLeft)

        #self.energyBankBox = QtGui.QGroupBox("Energy Bank")
        ## ENERGY spent
        userLabel = QtGui.QLabel(self)
        userLabel.setText("Energy Consumed this Week:")
        userLabel.setAlignment(QtCore.Qt.AlignLeft)
        energy = db.getEnergy(self.user[4], "spent")
        self.amtEnergy = QtGui.QStatusBar()
        self.amtEnergy.showMessage(str(int(energy)) + " Watts ")
        self.amtEnergy.setFont(QtGui.QFont("CorporateS-Regular", 11))

        ## ENERGY PRODUCED
        energy_produced = QtGui.QLabel(self)
        energy_produced.setText("Energy Produced this Week:")
        energy_produced.setAlignment(QtCore.Qt.AlignLeft)
        energy2 = db.getEnergy(self.user[4], "produced")
        print energy2
        print "ENREGY TWO!"
        print int(energy2*-1)

        self.amtEnergy2 = QtGui.QStatusBar()
        self.amtEnergy2.showMessage(str(int(energy2*-1)) + " Wattsss ")
        self.amtEnergy2.setFont(QtGui.QFont("CorporateS-Regular", 11))

        connected = db.getConnect("user",self.user[0])


        layout.addWidget(userLabel2, 0, 0)

        for i in range(len(connected)):
            device_on = QtGui.QPushButton(self)

            device_text = []
            device_text.append(str(connected[i]))

            device_text.append(str((db.getData("outlet", self.user[4],
                                                    (db.getData("device", db.convert
(connected[i]), None))[1]))[2]))
              device_text.append(str(db.sessionRate(db.convert(connected[i]), self.user
[4])))

             device_on.setText(device_text[0] + "    -    " + device_text[1]+ "   -    " +
device_text[2]+" W/m")
              self.smart_connect(device_on, self.amtEnergy, self.device_off_action, str
(connected[i]))

            layout.addWidget(device_on, i+1, 0)

        refresh = QtGui.QPushButton(self)
        refresh.setText("Refresh Energy Data")
        refresh.clicked.connect(self.refreshdata)

        layout.addWidget(userLabel, len(connected) + 2, 0)
        layout.addWidget(self.amtEnergy, len(connected) + 3, 0)
        layout.addWidget(energy_produced, len(connected) + 4, 0)
        layout.addWidget(self.amtEnergy2, len(connected) + 5, 0)

        layout.addWidget(refresh, len(connected) + 6, 0)

        self.refreshdata()

        self.nameLine = QtGui.QLineEdit()
        self.emptyButton = QtGui.QPushButton()
        self.emptyButton.setText(self.nameLine.displayText())
        self.emptyButton.hide()

        self.nameLine2 = QtGui.QLineEdit()
        self.emptyButton2 = QtGui.QPushButton()
        self.emptyButton2.setText(self.nameLine2.displayText())
        self.emptyButton2.hide()

        layout.addWidget(self.emptyButton, len(connected) + 1, 0)
        layout.addWidget(self.emptyButton2, len(connected) + 2, 0)

        self.distributionInfoGroupBox.setLayout(layout)
##        self.distributionInfoGroupBox.hide()

    def createEnergyManagementBox(self):
        self.energyManagementBox = QtGui.QGroupBox("Profile")
        layout = QtGui.QGridLayout()

        self.profilepicLabel = QtGui.QLabel(self)
        self.userimage = "imgs/"+str(self.user[0]) +".jpg"
        self.profilepicLabel.setPixmap(QtGui.QPixmap(self.userimage))
##        self.profilepicLabel.setAlignment(QtCore.Qt.AlignLeft)
        self.userLabel = QtGui.QLabel(self)
        self.userinfo = self.user[0]

        if (self.userinfo == "David"):
            self.userLabel.setText(self.userinfo + "    [" + "Host]")
        else:
            self.userLabel.setText(self.userinfo)

        self.userLabel.setFont(QtGui.QFont("CorporateS-Regular", 15.5))
        self.userLabel.setAlignment(QtCore.Qt.AlignCenter)

        layout.addWidget(self.profilepicLabel, 0, 1)
        layout.addWidget(self.userLabel, 0, 2)

        self.energyManagementBox.setLayout(layout)
##        self.energyManagementBox.hide()

    def createHorizontalGroupBox1(self):
        '''
        Device Labels
        '''
        self.horizontalGroupBox1 = QtGui.QGroupBox("New Device")
        layout = QtGui.QHBoxLayout()

        self.deviceName = QtGui.QLabel("<no device>", self)
        self.deviceName.setAlignment(QtCore.Qt.AlignCenter)

        layout.addWidget(self.deviceName)

        self.horizontalGroupBox1.setLayout(layout)
##        self.horizontalGroupBox1.hide()
```

```
    def createHorizontalGroupBox2(self):
        '''
        Control buttons
        '''
        self.horizontalGroupBox2 = QtGui.QGroupBox("Serial Connection")
        layout = QtGui.QHBoxLayout()

        self.startSerialButton = QtGui.QPushButton("Start Serial", self)
        self.startSerialButton.clicked.connect(self.startSerialReadButtonAction)

        useOutlet = QtGui.QPushButton("Use Device", self)
        useOutlet.clicked.connect(self.useOutletButtonAction)

        demoButton = QtGui.QPushButton("Demo Flash", self)
        demoButton.clicked.connect(self.demoAction)


        layout.addWidget(self.startSerialButton)
        layout.addWidget(useOutlet)
        layout.addWidget(demoButton)
        self.horizontalGroupBox2.setLayout(layout)

    def createGuestBox(self):
        ''' if there are guest ids present'''
        self.GuestBox = QtGui.QGroupBox("Network")
        layout = QtGui.QGridLayout()

        self.guestpic = QtGui.QLabel(self)
        self.guestimage = "imgs/m"+str(self.user[0]) +".jpg"
        self.guestpic.setPixmap(QtGui.QPixmap(self.guestimage))
        self.guestpic.setAlignment(QtCore.Qt.AlignLeft)
        self.guestpic.hide()

        self.guestpic2 = QtGui.QLabel(self)
        self.guestimage2 = "imgs/m"+str(self.user[0]) +".jpg"
        self.guestpic2.setPixmap(QtGui.QPixmap(self.guestimage))
        self.guestpic2.setAlignment(QtCore.Qt.AlignLeft)
        self.guestpic2.hide()

        self.guestpic3 = QtGui.QLabel(self)
        self.guestimage3 = "imgs/m"+str(self.user[0]) +".jpg"
        self.guestpic3.setPixmap(QtGui.QPixmap(self.guestimage))
        self.guestpic3.setAlignment(QtCore.Qt.AlignLeft)
        self.guestpic3.hide()

        layout.addWidget(self.guestpic, 1, 1)
        layout.addWidget(self.guestpic2, 1, 2)
        layout.addWidget(self.guestpic3, 1, 3)


        self.GuestBox.setLayout(layout)

##          self.GuestBox.hide()

    def buildGUIElements(self):

        self.createEnergyManagementBox()
        self.createDistributionInfo()
        self.createDeviceInformationBox()
        self.createHorizontalGroupBox1()
##          self.createHorizontalGroupBox3()
        self.createHorizontalGroupBox2()
        self.createGuestBox()
    def createDeviceInformationBox(self):
        '''
        Registered Device info from Database
        '''
        self.deviceInfoGroupBox = QtGui.QGroupBox("New Device Info")
        layout = QtGui.QGridLayout()

        for i in range( len(self.deviceIdTags) ):

            label = QtGui.QLabel( self.deviceIdTags[i] )
            lineEdit = QtGui.QLineEdit()

            self.deviceIdElements[ self.deviceIdTags[i] ] = [label, lineEdit]

            layout.addWidget(label, i + 1, 0)
            layout.addWidget(lineEdit, i + 1, 1)

        self.deviceInfoGroupBox.setLayout(layout)


    def updateDeviceFields(self, id_):

        print self.demo
        print "SELF DEMO"


        if self.demo == "1":
            data = "c"
            self.conn.send(data + '\0')

##        id_ = id_[:-2]
        print id_

        if db.getData("device", id_, None) == None:
            if self.guestpic.isHidden():
                self.guestimage1 = "imgs/m"+str(db.convertid(id_)) +".jpg"
                self.guestpic.setPixmap(QtGui.QPixmap(self.guestimage1))
                self.guestpic.show()
            elif self.guestpic2.isHidden():
                self.guestimage2 = "imgs/m"+str(db.convertid(id_)) +".jpg"
                self.guestpic2.setPixmap(QtGui.QPixmap(self.guestimage2))
                self.guestpic2.show()
            elif self.guestpic3.isHidden():
                self.guestimage3 = "imgs/m"+str(db..convertid(id_)) +".jpg"
                self.guestpic3.setPixmap(QtGui.QPixmap(self.guestimage3))
                self.guestpic3.show()
        else:

            '''Update device id'''
            lineEdit = self.deviceIdElements['id'][1]
            lineEdit.setText(id_ )

            self.deviceName.setText(self.db.convertid(id_))

            if self.emptyButton.isVisible():
                self.nameLine2.setText(id_)
            else:
                self.nameLine.setText(id_)

            '''Update device energy usage'''
            deviceUsageData  = self.db.getData('device', str(id_), None)
            deviceUsageLineEdit = self.deviceIdElements['usage'][1]
            deviceUsageLineEdit.setText( str(deviceUsageData[2]) )
            '''Update device energy usage'''
            deviceLastUseData  = self.db.getData('device', str(id_), None)
            deviceLastUseDataLineEdit = self.deviceIdElements['lastUse'][1]
            deviceLastUseDataLineEdit.setText( str(deviceLastUseData[5]) )


    def getDeviceNameFromDB(self,id_): ### USED IN SERIAL READER
##        id_ = id_[:-2]
        name = self.db.getData("device", id_, None)
        print "THIS IS NAME"
        print name
        return name

    def updateDeviceId(self, id_):
##        id_ = id_[:-2]

        self.deviceName.setText( self.getDeviceNameFromDB(id_)  )

        if not self.db.idExists(id_):
            self.db.insertNewUser(self.timeStamp.getTime(), id_, self.defaultCredit)
            print 'new RFID:', id_, self.timeStamp.getTime(), self.defaultCredit

        else:
            print id_ , 'already exists!'
            self.updateDeviceFields(id_)

if __name__ == "__main__":


    db = DataBase.DataBase('ppb.db')
    app = QtGui.QApplication(sys.argv)

    g = Gui(dBase = db, username = "David", guest = None )
    g.show()

    sys.exit(app.exec_())
```

Appendix C - Outlet Code

```
// for the ATMEGA644
#include <avr/pgmspace.h>
#include <inttypes.h>
#include <SoftwareSerial.h>

///////////////////
// WIRING PINS  ///
///////////////////
//Kent Display
#define DATAOUT 11 //MOSI (SI line on Sparkfun breakout board)
#define DATAIN 12 //MISO (SO line on Sparkfun breakout board)
#define SPICLOCK 13 //sck (SCK line on Sparkfun breakout board)
#define SLAVESELECT 10 //ss (CS line on Sparkfun breakout board)
//Buttons
#define SWITCH1 7
#define SWITCH2 6
//SoftSerial
#define rxPin  9
#define txPin  8
// extra pins: 4,5,6

///////////////////
// VARIABLES    ///
///////////////////
//RFID
byte hexCard[15];
boolean done = false;
byte i = 0;
//Buttons
int switch1Val;
int switch2Val;
//Kent
char clr = 0;
//XBee
boolean header = false;
char readXbee[10];
int j=0;
//System
boolean start = true;
int state = 1;
boolean remote_device_on = false;
boolean remote_device_off = false;
boolean user = false;
boolean device = false;
boolean user_done = false;
boolean device_done = false;
boolean remote = false;
boolean on_device = false;
boolean off_device = false;

//Software Serial
SoftwareSerial mySerial = SoftwareSerial(rxPin,txPin);

// User/Cost/Device -> Name found in RAM
char cost3_High = 0x1E;
char cost3_Low = 0xF0;
char cost2_High = 0x21;
char cost2_Low = 0x0C;
char cost1_High = 0x22;
char cost1_Low = 0x56;

char HEATER_High = 0x00;
char HEATER_Low = 0x02;
char WASHER_High = 0x02;
char WASHER_Low = 0x94;
char IRON_High = 0x04;
char IRON_Low = 0xEC;
char FAN_High = 0x07;
char FAN_Low = 0x62;

char MARK_High = 0x09;
char MARK_Low = 0xBA;
char CARL_High = 0x0E;
char CARL_Low = 0xA6;
char DAVID_High = 0x11;
char DAVID_Low = 0x1C;
char SARAH_High = 0x0C;
char SARAH_Low = 0x30;

char cost_Low;
char cost_High;
char name_Low;
char name_High;
char device_Low;
char device_High;

void setup()
{
  ///////////////////
  // INIT I/O PINS///
  ///////////////////
  //KENT
  pinMode(DATAOUT, OUTPUT);
  pinMode(DATAIN, INPUT);
  digitalWrite(DATAIN,LOW);
  pinMode(SPICLOCK,OUTPUT);
  pinMode(SLAVESELECT,OUTPUT);
  digitalWrite(SLAVESELECT,HIGH);
  //Buttons
  pinMode(SWITCH1, INPUT);
  pinMode(SWITCH2, INPUT);
  //Software Serial
  pinMode(rxPin, INPUT);
  pinMode(txPin, OUTPUT);

  // set the data rate for the ports
  mySerial.begin(9600);
  Serial.begin(9600);
}

void loop()
{
  // Start: Initialize the Kent Display, clear the screen and load the data into the
Kent RAM
  if (start) {
     SPCR = (1 << SPE) | (1 << MSTR) | (1 << SPR0) | (1 << SPR1) | (1 << CPHA); //
Initialize SPI communication with the display
    clr=SPSR;
    clr=SPDR;
    delay(25); //  Discovered that some delay is needed after setting SCPCR
      CLR_DISP_BRT();     //   This calls the function that clears the entire screen,
leaving it all white.
    delay(800);      // This delay allows the screen to be drawn.

    LoadData();  // This loads the array below to the display's RAM
    delay(800);
    DISP_FULLSCRN(0x38,0x40);
    delay(2000);
    DISP_PARTSCRN(0x26, 0x34, 0x00, 0x0E) ; // Use Remote Device
    delay(1000);

    remote_device_on = false;
    remote_device_off = false;
    user = false;
    device = false;
    user_done = false;
    device_done = false;
    remote = false;
    on_device = false;
    off_device = false;



    start = false;
    state = 0;
  }

  if (state == 0)  {
    switch1Val = digitalRead(SWITCH1);
    switch2Val = digitalRead(SWITCH2);
    if (switch2Val == LOW) { // Remote Device
      CLR_DISP_BRT();
      delay(1000);
      // remote device activated
      DISP_PARTSCRN(0x1B, 0xC6, 0x55, 0x70) ; // Remote Device: FAN
      delay(1000);
      DISP_PARTSCRN(0x32, 0x28, 0x01, 0x0E); // Turn on / Turn off:
      delay(1000);
      state = 6;
    }
    else if (switch1Val == LOW) { // Use Device
      CLR_DISP_BRT();
      delay(1000);
      DISP_PARTSCRN(0x15, 0xAE, 0x35, 0x6B) ; // ENERGY ID
      delay(1000);
      state = 1;
    }
    else { // for remote control purposes
      delay(500);
      state = 2;
    }
  }
  // State 1 = Grab the User's ID from RFID Sensor, send via XBee to Server
  if (state == 1) {
    if (mySerial.read()==2){ // check that it is the RFID Tag Header
      RFID();
      state = 2;
    }
  }
  if (state == 2) {
    if (Serial.available() > 0) {
      XBee_Data();
      char command = readXbee[0];
      if (command == 'U')
      {
        //Serial.print("User ID Identified as: ");
        Serial.println(readXbee);
        Serial.flush();
        user = true;
      }
```

```
 if (command == 'D')
 {
   Serial.print("Device ID Identified as: ");
   Serial.println(readXbee);
   Serial.flush();
   device = true;
 }

 if (command == 'R')
 {

   Serial.println("Remote Device Used: ");
   delay(5000);
   for (i=1; i <j; i++){
     Serial.write(readXbee[i]);
   }
   delay(5000);
   Serial.flush();
   remote = true;
 }

 if (user) {
   char command = readXbee[2];
   if (command == 'S') { //Sarah
     name_Low = SARAH_Low;
     name_High = SARAH_High;
   }
   else if (command == 'C') { //CARL
     name_Low = CARL_Low;
     name_High = CARL_High;
   }
   else if (command == 'M') { // MARK
     name_Low = MARK_Low;
     name_High = MARK_High;
   }
   else if (command == 'D') { //DAVID
     name_Low = DAVID_Low;
     name_High = DAVID_High;
   }

   DISP_PARTSCRN(name_High, name_Low, 0x78, 0x83); // display selected user
   delay(1000);
   if (!device_done) {
     DISP_PARTSCRN(0x13, 0x1A,0x30,0x4B); //  DEVICE
     delay(1000);

   }
   user = false;
   user_done = true;
 }
 if (device) {
   char command = readXbee[2];

   if (command == 'H') { //HEATER
     device_Low = HEATER_Low;
     device_High = HEATER_High;
   }
   else if (command == 'W') { //WASHER
     device_Low = WASHER_Low;
     device_High = WASHER_High;
   }
   else if (command == 'I') { //IRON
     device_Low = IRON_Low;
     device_High = IRON_High;
```

```
        }
        else if (command == 'F') { //FAN
          device_Low = FAN_Low;
          device_High = FAN_High;
        }
          DISP_PARTSCRN(device_High, device_Low+0x07, 0x6B, 0x76); // display selected
device
        delay(1000); //

        device = false;
        device_done = true;
      }

      if (user_done && device_done) {
        state = 3;
        user_done = false;
        device_done = false;
        delay(100);
      }
      else {
        state = 1;
      }
      if (remote) {
        delay(1000);
        state = 0;
        remote = false;
      }
    }
    else{
      delay(500);
      state = 0;
    }
  }

  if (state == 3) {
//////////////////////////////////////////////////////////////////////////////////////
//////////////////////////
    // CHOICE SCREEN
//////////////////////////////////////////////////////////////////////////////////////
//////////////////////////
    //display partial screen cleared; leave the device and user name intact
    CLR_DISP_BRT();
    delay(1000);
    DISP_PARTSCRN(name_High, name_Low, 0x78, 0x83); // selected user
    delay(1000);
    DISP_PARTSCRN(device_High, device_Low+0x07, 0x6B, 0x76); //selected device
    delay(1000);

    DISP_PARTSCRN(0x2F, 0xEE, 0x55, 0x69); // Cost Now / Cost later

    delay(1000);

    int randNumber = random(1, 4);
    if (randNumber == 1){
      cost_Low = cost1_Low;
      cost_High = cost1_High;
    }
    else if (randNumber == 2) {
      cost_Low = cost2_Low;
      cost_High = cost2_High;
    }
    else if (randNumber == 3) {
        cost_Low = cost3_Low;
        cost_High = cost3_High;
      }

      DISP_PARTSCRN(cost_High,cost_Low,0x46,0x50); // fake costs
      delay(1000);

      DISP_PARTSCRN(0x2E, 0x2C,0x01,0x0E); // use now use later buttons
      delay(1000);
      state = 4;
    }


    if (state == 4)  {
      switch1Val = digitalRead(SWITCH1);
      switch2Val = digitalRead(SWITCH2);

      if (switch1Val == LOW) {

////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////
        // NOW IN USE

////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////

        CLR_DISP_BRT();
        delay(1000);

        DISP_PARTSCRN(name_High, name_Low, 0x78, 0x83); // selected user
        delay(1000);
        DISP_PARTSCRN(device_High, device_Low+0x07, 0x6B, 0x76); //selected device
        delay(1000);

        DISP_PARTSCRN(0x2B, 0xB6, 0x55, 0x69) ; // is now in use
        delay(1000);


        DISP_PARTSCRN(0x33, 0xEA, 0x01, 0x0E); // Home / Turn off:
        delay(1000);

        Serial.write("ATON 0\n");
        delay(1000);

        state = 5;
      }

      if (switch2Val == LOW) {

////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////
        // USE AT A LATER TIME

////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////

        CLR_DISP_BRT();
        delay(1000);

        DISP_PARTSCRN(name_High, name_Low, 0x78, 0x83); // selected user
        delay(1000);
        DISP_PARTSCRN(device_High, device_Low+0x07, 0x6B, 0x76); //selected device
        delay(1000);
      DISP_PARTSCRN(0x29, 0x7C, 0x55, 0x69); // will turn on at a later time
      delay(1000);


      DISP_PARTSCRN(0x27, 0x9C, 0x01, 0x0E); // Home/Cancel
      delay(1000);

////////////////////////////////////////////////////////////////////////////////////
//////////////////////////

      Serial.println("Done!");
      state = 5;
    }
  }

  if (state == 5)  {
    switch1Val = digitalRead(SWITCH1);
    switch2Val = digitalRead(SWITCH2);
    if (switch1Val == LOW) { // HOME button
      start = true;
    }

    if (switch2Val == LOW) { // CANCEL Button
      if (remote_device_on) {
        //turn off remote device
        Serial.print("REMOTE: OFF");
        delay(1000);
        remote_device_on = false;
        start = true;
      }
      else if (remote_device_off) {
        // turn on remote device
        Serial.print("REMOTE: ON");
        delay(1000);
        remote_device_off = false;
        start = true;
      }
      else{

        Serial.write("ATOF 0\n");
        delay(1000);
        state = 3;
      }
    }

  }
  if (state == 6) {
    switch1Val = digitalRead(SWITCH1);
    switch2Val = digitalRead(SWITCH2);
    if (switch1Val == LOW) { // Turn ON remote device
      Serial.print("REMOTE: ON");
      delay(2000);
      DISP_PARTSCRN(0x2B, 0xB6, 0x41, 0x55) ; // is now in use
      delay(1000);
      DISP_PARTSCRN(0x27, 0x9C, 0x00, 0x0D); // Home/Cancel
      delay(1000);
      remote_device_on = true;
      state = 5;
    }
    if (switch2Val == LOW) { // Turn OFF remote device
      Serial.print("REMOTE: OFF");
      delay(2000);
      DISP_PARTSCRN(0x35, 0x8E, 0x41, 0x55); // is now off.
      delay(1000);
      DISP_PARTSCRN(0x27, 0x9C, 0x00, 0x0D); // Home/Cancel
      delay(1000);
      state = 5;
      remote_device_off = true;
    }
  }

}
// Grabbing Data from XBee Controller
void XBee_Data(void){
  j = 0;
  if (Serial.read() == 60) { //60 == "<"
    header = true;
    Serial.println("Header Found"); // for some reason, need to serial.println here..
  }
  while (Serial.available() >0 ) {
     if (header && (Serial.peek() == 62)) {// reached the end of the data; 62 dec ==
">"
      Serial.read();
      //Serial.println("readXbee Completed");
    }
    else if (header) {
      readXbee[j] = byte(Serial.read());
      j++;

    }
  }
  header = false;
}


void RFID(void) {
  for (i = 0; i < 14; i++)
  {
    hexCard[i] = mySerial.read();
    done = true;
  }

  if (done) {
    Serial.print("Code: ");
    for (i = 1; i < 11; i++)
    {
      Serial.print(hexCard[i], BYTE); // sends it to XBee Controller
    }
    Serial.println("");
    done = false;
  }
}


// ---------------------------------------------
// Below are functions for the SPI interface
// ---------------------------------------------

// Adding hex addresses
int add( char highAddress,  char lowAddress, char value) {

  uint16_t  address = highAddress * 0x100;
  //  Serial.print("Adding Function High Address: ");
  //  Serial.println(address,HEX);
   uint8_t lowAddr = lowAddress; // does adding FFFF work for all cases? only tested
for 0xF0 case
  address = highAddress * 0x100 + lowAddr ;
  //   Serial.print("Adding Function Low Address: ");
  //  Serial.println(lowAddr,HEX);

  address = address  + value;
  //   Serial.print("Adding Function Value Address: ");
  //  Serial.println(address,HEX);
  return address;
}


//Clear Display Bright clears entire screen to be bright including the border, fixed
length command
void CLR_DISP_BRT() {
  select();
  spi_transfer(0x10); //Clear Display Bright Command
  deselect();
}

//Display Fullscreen triggers a full screen update from a specified image buffer in
the onboard image RAM, fixed length command
void DISP_FULLSCRN(volatile char HighAddress, volatile char LowAddress) {
  select();
  spi_transfer(0x18);          //Display Fullscreen command
  spi_transfer(HighAddress);   //High byte of the target memory address
  spi_transfer(LowAddress);    //Low byte of the target memory address
  deselect();
}

// Display partial screen, use rows 00-9F
void DISP_PARTSCRN(volatile char HighAddress0, volatile char LowAddress0, volatile
char LowAddress1,volatile char LowAddress2)
{
  select();
  spi_transfer(0x19);          //DISP_PARTSCRN command
  spi_transfer(HighAddress0);   //High byte of the first address from RAM
  spi_transfer(LowAddress0);    //Low byte of the first address from RAM
  spi_transfer(0x00);   //High byte of the first address
  spi_transfer(LowAddress1);    //Low byte of the first address
  spi_transfer(0x00);   //High byte of the last address
  spi_transfer(LowAddress2);    //Low byte of the last address
  deselect();
}



char READ(int HighAddress, int LowAddress) {
  select();
  spi_transfer(0x04);  //Read Command
  spi_transfer(HighAddress);  //High byte of the target memory address
  spi_transfer(LowAddress);  //Low byte of the target memory address
  spi_transfer(0x00);  //Dummy byte
  spi_transfer(0x00);  //Dummy byte
   return(spi_transfer(0x00));   //Final dummy byte, Screen will transfer data (on SO
line) during reciept of this byte
}

//Optional if more data needs to be read
char READmore() {
  return(spi_transfer(0x00));  //Additional dummy byte, Screen will transfer data (on
SO line) during reciept of this byte
}

//Deselect screen so that it knows there is no more data to be read
void READend() {
  deselect();
}


// Write to RAM per ROW
//Writes data to screen RAM starting at the target memory address, variable length
command
//Note: the WRITEend() function must be called after sending the variable amount of
Data Bytes
void WRITE(int HighAddress, int LowAddress, int Data) {
  select();
  spi_transfer(0x00);  //Write Command
  spi_transfer(HighAddress);  //High byte of the target memory address
  spi_transfer(LowAddress);  //Low byte of the target memory address
  spi_transfer(Data);  //The first value to be written, more may follow
}

//Optional if more data needs to be sent
void WRITEmore(int Data) {
  spi_transfer(Data);  //Send additional byte to screen
}

//Deselect screen so that it knows there is no more data to be sent
void WRITEend() {
  deselect();
}

void FILL(volatile char HighAddress1, volatile char LowAddress1, volatile char
HighAddress2, volatile char LowAddress2, volatile char data)
{
  select();
  spi_transfer(0x01);         //Fill command
  spi_transfer(HighAddress1);   //High byte of the first address
  spi_transfer(LowAddress1);    //Low byte of the first address
  spi_transfer(HighAddress2);   //High byte of the last address
  spi_transfer(LowAddress2);    //Low byte of the last address
  spi_transfer(data);   //Fill Value
  deselect();
}


void select() {
  digitalWrite(SLAVESELECT,LOW);
}

void deselect() {
  digitalWrite(SLAVESELECT,HIGH);
}

void SLEEP() {
  select();
  spi_transfer(0x20); //Sleep command
  deselect();
}

char spi_transfer(volatile char data)
{
  SPDR = data; // Start the transmission
  while (!(SPSR & (1<<SPIF))){ //Wait for the end of the transmission
  }
  return SPDR; // return the received byte
}

// ---------------------------------------------
// Below are functions for grabbing the text
// --------------------------------------------


//Grab Letter
// take specific start address, write to RAM, repeat for 14 rows
void GRAB_LETTER(volatile char highAddressWRITE,volatile char lowAddressWRITE, int
amt, volatile char grabHeight, volatile char highAddressREAD, volatile char
lowAddressREAD, volatile char letterColumn) { // remember addresses need to be in HEX
  uint16_t startAddressWRITE;
  uint16_t startAddressREAD;
  uint16_t startAddressREAD0 = add(highAddressREAD,lowAddressREAD,letterColumn);
  uint16_t rowAdder = 0x001E;
  uint16_t startAddressWRITE0 = add(highAddressWRITE ,  lowAddressWRITE, 0x00);

  Serial.print("Initial startAddressREAD: ");
  Serial.println(startAddressREAD0, HEX);
  Serial.print("Initial startAddressWRITE: ");
  Serial.println(startAddressWRITE0, HEX);

  for (int i = 0x00; i < grabHeight; i++) {

    startAddressREAD = startAddressREAD0 + rowAdder*i;
    startAddressWRITE = startAddressWRITE0 + rowAdder*i;
    Serial.print("startAddressREAD: ");
    Serial.println(startAddressREAD, HEX);
    Serial.print("startAddressWRITE: ");
    Serial.println(startAddressWRITE, HEX);
    uint8_t newLowAddress = startAddressREAD;
    char ReceivedData[amt] ;

    ReceivedData[0] = READ(startAddressREAD/0x100, newLowAddress);
    uint8_t newWriteAddress = startAddressWRITE;
    for (int j = 1; j < amt; j++){
      ReceivedData[j] =  READmore();

      Serial.println("Received DATA:" );
      Serial.println(ReceivedData[j], HEX);

    }
    READend();

    delay(25);

    WRITE(startAddressWRITE/0x100, newWriteAddress, ReceivedData[0]);
    for (int j = 1; j < amt; j++){
      WRITEmore(ReceivedData[j]);
    }
    WRITEend();
    delay(25);
  }
}


void LoadData()
{
  int std_delay = 25;
  static prog_uint8_t SampleImage[] PROGMEM = {

    //The code to print data to the display goes here.
    //    DISP_PARTSCRN(0x13, 0x1A, 0x35, 0x50); // DEVICE
    //    DISP_PARTSCRN(0x15, 0xAE, 0x35, 0x50) ; // ENERGY ID
    //    DISP_PARTSCRN(0x18, 0xBA, 0x35, 0x50); // PLEASE TAP IN YOUR ENERGY ID
    //    DISP_PARTSCRN(0x1B, 0xC6, 0x35, 0x50); // REMOTE DEVICE: FAN
  //    DISP_PARTSCRN(0x1E, 0xF0, 0x04, 0x10) ; // cost3
  //    DISP_PARTSCRN(0x21, 0x0C, 0x04, 0x10); // cost2
  //    DISP_PARTSCRN(0x21, 0xDE, 0x04, 0x10); // cost1

  //       DISP_PARTSCRN(0x00, 0x1E, 0x35, 0x40); // HEATER
  //       DISP_PARTSCRN(0x02, 0x94, 0x35, 0x3F) ; // WASHER
  //       DISP_PARTSCRN(0x04, 0xEC, 0x35, 0x3F); // IRON
  //       DISP_PARTSCRN(0x07, 0x62, 0x35, 0x3F); // FAN

  //       DISP_PARTSCRN(0x09, 0xBA, 0x35, 0x40) ; // MARK:
  //       DISP_PARTSCRN(0x0C, 0x30, 0x35, 0x40); // SARAH:
  //       DISP_PARTSCRN(0x0E, 0xA6, 0x35, 0x3F); // CARL:
  //       DISP_PARTSCRN(0x11, 0x1C, 0x35, 0x3F); // DAVID:
  //       DISP_PARTSCRN(0x25, 0xDA, 0x05, 0x15) ; // Use Remote Device
  //       DISP_PARTSCRN(0x27, 0x9C, 0x05, 0x15); // Home/Cancel
  //       DISP_PARTSCRN(0x29, 0x7C, 0x15, 0x29); // will turn on at a later time
  //       DISP_PARTSCRN(0x2B, 0xB6, 0x35, 0x49) ; // is now in use
  //       DISP_PARTSCRN(0x2E, 0x2C, 0x05, 0x15); // Use now / use later?
  //       DISP_PARTSCRN(0x2F, 0xEE, 0x15, 0x29); // Cost Now / Cost later
  //       DISP_PARTSCRN(0x32, 0x28, 0x05, 0x13); // Turn on / Turn off:
  //       DISP_PARTSCRN(0x33, 0xEA, 0x15, 0x23); // Home / Turn off:
  //       DISP_PARTSCRN(0x35, 0x8E, 0x35, 0x49); // is now off.
  //HEX VALUES WERE DELETED FROM APPENDIX
};


// Test to read from Array
Serial.print("Size of Array: ");
Serial.println(sizeof(SampleImage));


//PUT_LETTER(25,30, 350, (GRAB_LETTER(25, 30, 450, SampleImage)), NewImage);


// The following code writes the array above one at a time to the display's memory.
select();
spi_transfer(0x00);  // Transmit command
spi_transfer(0x00);  // High byte of the target memory
spi_transfer(0x00);  // Low byte of the target memory
for(int j = 0x00; j < 0x4B00; j++) //
{
  spi_transfer(pgm_read_byte(&SampleImage[j]));
}
deselect();



}
```

Bibliography

[1] Arduino, "Arduino Uno". [Online]. Available: http://www.arduino.cc/en/Main/ArduinoBoardUno. [Accessed: Jan. 29, 2011].

[2] Cornell University. ""Vampire" Appliances -- They Suck Electricity Even When Switched Off -- Cost Consumers $3 Billion A Year, Says Cornell Energy Expert." ScienceDaily 27 September 2002. [Online]. Available: http://www.sciencedaily.com-/releases/2002/09/020926065912.htm. [Accessed: Jan. 11, 2011].

[3] Digi International Inc., "XBee/XBee-PRO RF Modules," 2009. [Online]. Available: www.sparkfun.com/datasheets/Wireless/Zigbee/XBee-Datasheet.pdf. [Accessed: Jan. 29, 2011].

[4] Energy Aware Technology Inc., "The PowerTab In-Home Display,". [Online]. Available: http://www.energy-aware.com/wp-content/uploads/2011/01/Product-Info-Sheet-PowerTab.pdf. [Accessed: Jan. 29, 2011].

[5] Energy Optimizers Limited, "Plogg". [Online]. Available: http://www.plogginternational.com/ploggproducts.html. [Accessed: Jan. 29, 2011].

[6] Fairchild SemiConductor, "Triple 2-Channel Analog Multiplexer," 74VHC4053 datasheet, May 2007. [Online]. Available: www.fairchildsemi.com/ds/74/74VHC4051.pdf. [Accessed: Jan. 29, 2011].

[7] Feldman, Assaf, "ReachMedia: On-the-move interaction with everyday objects," Ambient Intelligence Group, MIT Media Laboratory. [Online]. Available: http://web.media.mit.edu/~assaf/ReachMedia/ISWC_final.pdf. [Accessed: Jan. 29, 2011].

[8] B.J. Fogg, *Persuasive Technologies - Introduction.* Communications of the ACM, 42 (5) pp. 26-29, 1999.

[9] B. J. Fogg, "A Behavior Model for Persuasive Design," *Persuasive 2009,* April 2009. [Online]. Available: http://www.bjfogg.com/fbm_files/page4_1.pdf. [Accessed: Jan. 11, 2011].

[10] Google Inc., "Google powermeter," *Google.org.* [Online]. Available: http://www.google.com/powermeter/about/about.html. [Accessed: Jan. 29, 2011].

[11] ID Innovations, "ID Series Datasheet," RFID datasheet, March 2005. [Online]. Available: www.sparkfun.com/datasheets/Sensors/ID-12-Datasheet.pdf. [Accessed: Jan. 29, 2011].

[12] LinkSprite, *Smart Outlet User Manual*, LinkSprite Technologies Inc., Jan 2009. [Online]. Available: www.linksprite.com/pub/SmartOutlet_english.pdf. [Accessed: Jan. 29, 2011].

[13] Maxim Integrated Products, “2.2MHz, Dual, Step-Down DC-DC Converters, Dual LDOs, and RESET,” LDO datasheet, Nov. 2010. [Online]. Available: http://www.datasheets.maxim-ic.com/en/ds/MAX16922.pdf. [Accessed: Jan. 29, 2011].

[14] P3 International, “P4400 Kill A Watt Operation Manual,”. [Online]. Available: http://www.p3international.com/manuals/p4400_manual.pdf. [Accessed: Jan. 29, 2011].

[15] “Rusty Nail Workshop Projects,” August, 2010. [Online]. Available: http://www.rustynailworkshop.com/Projects/Entries/2010/8/9_Arduino_Powered_Kent_Display.html. [Accessed: Jan. 29, 2011].

[16] DIY Kyoto, “Wattson,” [Online]. Available: http://www.diykyoto.com/uk. [Accessed: Jan. 29, 2011].