Android Application Controlled Surge Protector

By

Lok An

Senior Project

ELECTRICAL ENGINEERING DEPARTMENT
California Polytechnic State University
San Luis Obispo, CA

June 2011
## Table of Contents

I. Acknowledgements .................................................................................................................. 4  
II. Abstract ................................................................................................................................ 5  
III. Introduction ............................................................................................................................. 6  
IV. Background ............................................................................................................................. 8  
V. System Requirements ............................................................................................................... 10  
VI. System Architecture ............................................................................................................... 11  
VII. Component Design ................................................................................................................. 13  
   A. Current Sensor .................................................................................................................... 13  
   B. Relay .................................................................................................................................. 14  
   C. Microcontroller .................................................................................................................. 15  
   D. Bluetooth .......................................................................................................................... 15  
   E. Android Phone/Remote ....................................................................................................... 16  
VIII. Component Integration ......................................................................................................... 17  
IX. Integration and Test Results .................................................................................................. 18  
   A. Test Plan ............................................................................................................................ 18  
   B. Developing the Android Application ................................................................................. 19  
   C. Connecting Hardware with Phone .................................................................................... 20  
   D. Current Sensor Measurements ......................................................................................... 21  
   E. Graphing Data ................................................................................................................... 23  
   F. Power Analysis ................................................................................................................... 23  
X. Problems and Solutions .......................................................................................................... 25  
   A. Supplying Current to the Relay and Flyback Voltage Protection ...................................... 25  
   B. Bluetooth Issues ............................................................................................................... 26  
   C. Measuring Current ............................................................................................................ 26  
XI. Conclusion .............................................................................................................................. 28  
XII. Bibliography .......................................................................................................................... 30  
XIII. Appendix .............................................................................................................................. 32  
   A. Parts List ............................................................................................................................ 32
List of Figures and Tables

Figure 1: MOV V-I Transfer Characteristic ................................................................. 8
Figure 2: Top Level System Block Diagram ................................................................. 11
Figure 3: System Flowchart ....................................................................................... 11
Figure 4: Graphical User Interface ............................................................................ 12
Figure 5: ACS712 Current Sensor on a Breakout Board ........................................... 13
Figure 6: Omron G5SB-145VDC Relay Schematic .................................................. 14
Figure 7: Arduino Nano Layout .................................................................................. 15
Figure 8: BlueSMiRF Silver Wireless Modem ............................................................. 16
Figure 9: System Schematic Diagram ........................................................................ 17
Figure 10: System Fully Integrated .......................................................................... 18
Figure 11: Typical Input Data Stream ......................................................................... 20
Figure 12: Raw ADC Value Plotted ........................................................................... 21
Figure 13: Output Power for a 75 Light Bulb .............................................................. 23
Figure 14: Output Current with Surge Protector Switched Off .................................. 27

Table 1: Standby Power Consumption of Common Electronic Devices ...................... 7
Table 2: System Power Consumption ........................................................................ 24
Acknowledgements

I would first like to thank my parents for their continued support and dedication of my education. I would also like to thank Dr. Bryan Mealy for supervising this project and assisting me as best as he could. In addition, I would like to thank the members of the Response Measurement Team (Eric Aker, Emily Gu, Michelle Mooi, Kim Ngan, and Pete Bespaly) for assisting me through my first Android development experience. Lastly, I would like to thank Dr. John Oliver for sparking the initial idea of this project.
Abstract

A significant amount of consumer electronics cannot be switched off completely without being unplugged. Standby power, also known as phantom power, refers to the electric power consumed by electronic appliances when they are switched off. There are some devices that need constant power; items like alarm clocks and security systems need constant power to function correctly. Some devices, such as video game consoles and televisions, consume power without offering any features. Electricity lost from standby power has been increasing with the amount of consumer electronics increasing in households.

This project’s goal is to create a low power remote that can turn off the surge protector when it is not in use as well as monitor power consumption. Using an Android powered smartphone as the remote, an Arduino microcontroller, a solid-state relay as the switch, a Hall Effect current sensor to sense current, and a BlueSMiRF Bluetooth module, I created a system that was able to perform these functions. This paper focuses on the uses and integration of these components and the viability of such systems to reduce standby power.
Introduction

A surge protector is a strip of sockets that allow for multiple devices to be plugged into a single electrical socket and protects these devices from voltage spikes. It achieves this by either blocking or by shorting to ground any unwanted voltages above a safe threshold. When the increase lasts three nanoseconds or more, it’s called a surge. When it only lasts for one or two nanoseconds, it’s called a spike. Surge protectors are typically placed under tables or behind furniture which makes them difficult to reach. This makes having a remote to switch the surge protector on and off a practical solution. Today, you can buy a surge protector that comes with its own dedicated remote [3]. This project’s idea is to make the remote your smart phone. This would eliminate the clutter of having an extra remote and the trouble of finding it.

A smartphone is an advanced mobile phone that runs a complete operating system that provides a platform for application developers. According to a research report by Berg Insight, global shipments of smartphones increased 74 percent in 2010 to 295 million units [4]. A smartphone is the perfect remote because of its advanced features and its growing popularity.

The key motivation for this project is to find a solution to eliminate standby power. Standby power, also known as phantom power, refers to the electric power consumed by electronic appliances when they are switched off. There are some devices that need constant power; items like alarm clocks and security systems need constant power to function correctly. Some devices, such as video game consoles and televisions, consume power without offering any features. Table 1 shows standby power consumption of common
electronic devices. Cutting out the standby usage of this list of items would save $67 and eliminate 1,140 pounds of carbon dioxide output per year [12].

<table>
<thead>
<tr>
<th>Equipment</th>
<th>Typical Standby Use</th>
<th>Annual Cost</th>
<th>$0</th>
<th>$5</th>
<th>$10</th>
<th>$15</th>
<th>CO₂ Emitted/Yr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Television</td>
<td>4 watts</td>
<td>$3.80</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>65 lbs</td>
</tr>
<tr>
<td>VCR</td>
<td>6 watts</td>
<td>$5.69</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>97 lbs</td>
</tr>
<tr>
<td>DVD</td>
<td>2 watts</td>
<td>$1.90</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>32 lbs</td>
</tr>
<tr>
<td>Cable box</td>
<td>12 watts</td>
<td>$11.39</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>194 lbs</td>
</tr>
<tr>
<td>Satellite receiver</td>
<td>15 watts</td>
<td>$14.24</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>243 lbs</td>
</tr>
<tr>
<td>Component stereo</td>
<td>7 watts</td>
<td>$6.64</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>113 lbs</td>
</tr>
<tr>
<td>Game console</td>
<td>1 watt</td>
<td>$0.95</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>16 lbs</td>
</tr>
<tr>
<td>Cell phone charger</td>
<td>2 watts</td>
<td>$1.90</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>32 lbs</td>
</tr>
<tr>
<td>Desktop computer</td>
<td>4 watts</td>
<td>$3.80</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>65 lbs</td>
</tr>
<tr>
<td>Notebook computer</td>
<td>1 watt</td>
<td>$0.95</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>16 lbs</td>
</tr>
<tr>
<td>Computer monitor</td>
<td>1 watt</td>
<td>$0.95</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>16 lbs</td>
</tr>
<tr>
<td>Modem</td>
<td>5 watts</td>
<td>$4.75</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>81 lbs</td>
</tr>
<tr>
<td>Wireless router</td>
<td>2 watts</td>
<td>$1.90</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>32 lbs</td>
</tr>
<tr>
<td>Ink-jet printer</td>
<td>2 watts</td>
<td>$1.90</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>32 lbs</td>
</tr>
<tr>
<td>Rechargeable power tool</td>
<td>4 watts</td>
<td>$4.18</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>71 lbs</td>
</tr>
<tr>
<td>Battery charger</td>
<td>2 watts</td>
<td>$2.09</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>36 lbs</td>
</tr>
</tbody>
</table>

Table 1: Standby Power Consumption of Common Electronic Devices [12]

It’s worth noting that there have been active efforts to reduce standby power. For example, the Federal Energy Management Program (FEMP) requires federal agencies to buy products with a standby power level of 1 watt or less [13]. In addition, advancements in technology have been able to lower standby power for many devices to be less than one watt. For this project to be feasible, I believe it has to consume less than 2 watts of power. Using the data from table 1, its annual cost should be less than $1.90 and emit only 32 pounds of carbon dioxide a year.
Background

A standard surge protector uses a metal oxide varistor (MOV) to divert the voltage surges. At low applied voltages, the MOV looks like an open circuit because of its unique two-phase material assumes the properties of an insulator. When applied voltage exceeds rated clamping voltage, the device effectively becomes a short circuit, protecting the component that it shunts [5]. Using a regular power strip instead of a surge protector would make no difference for this project since the MOV only has an effect on voltage surges. Figure 1 shows the V-I transfer characteristic for the MOV. It has a non-linear V-I curve, which means it can pass varying currents over a narrow voltage range.

Figure 1: MOV V-I Transfer Characteristic [5]

The Android platform was used to develop the application. A broad alliance of leading technology and wireless companies joined forces and announced the development of Android, the first truly open and comprehensive platform for mobile devices, on November 5, 2007. Google Inc., Intel, T-Mobile, Sprint, HTC, Qualcomm, Motorola and others have collaborated on the development of Android through the Open Handset Alliance, a multinational alliance of technology and mobile industry leaders [6]. Android is based on a Linux 2.6.25 kernel, with a Dalvik virtual machine featuring just-in-time (JIT) compilation.
In order to run Android, a device must adhere to its Compatibility Definition Document (CDD) requirements. This document covers both software and hardware compatibility as well as test procedures. Bluetooth is one device implementation that is described. The document states that “devices that include a Bluetooth transceiver MUST enable the radio frequency (RFCOMM) based Bluetooth API [8].” This ensures that Bluetooth implementation is consistent over every Android device. The RFCOMM protocol is a simple set of transport protocols that emulate RS-232 serial ports. Also called serial port emulation, RFCOMM provides a simple, reliable data stream and is the most widely supported Bluetooth protocol.

The surge protector and Android application will communicate via Bluetooth. Bluetooth is a very flexible wireless standard that acts as a complement and an extension to existing wireless technologies, addressing the short-range and inter-device connectivity. The technology enables the design of low-power, small-sized, and low-cost radios that can be embedded in a wide range of products [7]. Bluetooth operates at 2.45 GHz and uses a Service Discovery Protocol (SDP) to discover a device and its associated parameters. As part of SDP, devices must publish a Universally Unique Identifier (UUID) for each service they have available. The purpose of this is to specify the expected communication pattern for a given device [15]. For example, there are many UUIDs for handsfree headset functionality. So when a handsfree headset is connected to a device using Bluetooth, the SDP will be used to determine which Bluetooth profiles are supported by the headset.
System Requirements

The system must perform two main functions: turn the surge protector on and off and measure amount of current being drawn from a smartphone. In addition, the system must be able to present the power data graphically. In order for this project to be feasible, it must have low power consumption. Only one relay and current sensor will be used to control the whole surge protector and measure power. This will make the project simpler and require less hardware.

Summarized Feature List

- Switch Surge Protector On/Off via Smartphone
- Measure Current/Power Consumption
- Present Data Graphically
- Low Power Consumption
System Architecture

Figure 2: Top Level System Block Diagram

Figure 2 shows the top level system block diagram and figure 3 shows an overview of the graphical user interface (GUI) and its functions.

Figure 3: Graphical User Interface
Figure 4: System Flowchart

Figure 4 shows the system flowchart. The processes on the left are performed by the Android phone and the processes on the right are performed by the Arduino microcontroller. Current will be constantly sensed and sent to the phone.
Component Design

Current Sensor

The three main ways to measure current are via a sensing resistor, a current transformer, or a Hall Effect current sensor. A sensing resistor is simply a series resistor with an extremely low resistance value. Since we know the input voltage and resistance, the current can be easily calculated by using Ohm’s Law. However, this does not take into account when loads are plugged in and the resistance changes. The next two options, current transformers and Hall Effect sensors, detect current based on changes in the magnetic field. I chose to use a Hall Effect current sensor because of its size and cost advantage of the transformer.

Allegro’s ACS712 Hall Effect-based current sensor was used to measure the current. It has an 185mV/A output sensitivity and senses current with a low 1.2mΩ internal conductor resistance. This sensor is rated up to 5A. Figure 3 shows the ACS712 on a breakout board. Typical applications for this sensor are for motor control, load detection, and overcurrent fault protection. It outputs instantaneous current and needs to be converted into root-mean-square (RMS) current in order to measure power consumption. More details about this in the integration and test results section.

Figure 5: ACS712 Current Sensor on a Breakout Board
Relay

Next, I needed a component that could switch the electrical line on and off with. A relay is a device that provides an electrical connection between two or more points in response to a control signal. In addition, it has full isolation between control signal and the controlled circuit. A solid state relay (SSR) was used for switching on/off. A SSR is an electric switching device in which a small control signal controls a larger voltage. It serves the same function as an electromechanical relay, but it’s smaller, faster, and has no moving parts when switching.

Omron’s G5SB-145 VDC is single pole, double throw (SPDT) compact solid state relay with high insulation. An SPDT relay is a simple switch where the common connection is switched between two connections. It’s rated up to 5A at 250VAC so it can handle any type of AC wall power. The coil to control the relay is rated at 5VDC and requires at least 80mA [9]. Figure 4 shows the relay as well as its schematic. Pins 1 and 5 are the control coil terminals, pin 2 is the common connection and pins 3 and 4 are the switching connections. When the coil is energized, the arm that connects terminals 2 and 4 will pivot to terminals 2 and 3.

![Figure 6: Omron G5SB-145VDC Relay Schematic](image-url)
**Microcontroller**

Based on the Atmega328, the Arduino Nano 3.0 is a surface mount breadboard compatible board. Arduino is an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software. It has a 10-Bit ADC and it can be powered by either USB or a battery. Figure 5 shows the Arduino Nano and its pin layout. For this project, the TX, RX, digital, analog, 5V, and ground pins will be utilized.

![Arduino Nano Layout](image)

*Figure 7: Arduino Nano Layout*

**Bluetooth**

The BlueSMiRF Silver module is based off of Roving Networks RN-42 Class 2 Bluetooth Module. It has a built-in compact ceramic antenna and a maximum data transmission rate of 115.2Kbps. The BlueSMiRF Silver is identical to the BlueSMiRF Gold, except it only has 50 to 60 ft. range compared to the Gold’s 100m (328 ft. range). Figure 6 shows a picture of the modem. It addition to its small size, its operational voltage of 3.3V to 6V makes it easy for integration with other devices.
Android Phone/Remote

A Samsung Moment SPH-M900 running Android 2.1 will be used as the remote. The phone has to be set up properly in order to be used for development. Under settings, the phone must enable USB Debugging. Then on your computer, a USB driver for the Android Debug Bridge (adb) must be installed. The adb allows you to manage port forwarding, copy files to/from the phone, and run shell commands.
Component Integration

Figure 7 shows the system schematic diagram. On the BlueSMiRF, the CTS-1 and RTS-0 are connected because hardware flow control was not used. A 1nF filter capacitor was used on the ACS712 to improve sensing accuracy. The BJT was used to increase the current supplied to the relay and the diode is used to protect the microcontroller from flyback voltage. These are described in more detail in the problems and solutions section.

Figure 9: System Schematic Diagram
Integration and Test Results

Figure 10 shows the project 12 gauge wire was used for connecting the relay and current sensor to the surge protector. This was done to match the wire gauge to the surge protector and minimize losses.

Figure 10: System Fully Integrated

Test Plan

The test plan was to interface the relay and current sensor with the microcontroller first. The relay and current sensor were then connected in series with the surge protector. Once this setup was verified to be working correctly, interfacing was done between the Bluetooth and microcontroller. Once this is all done, the final step was to create an Android
application and establish communication to the microcontroller. Finally, the current sensor will be tested for accuracy and a power analysis for the system will be performed.

**Developing the Android Application**

Before I developed my own application, I tested Bluetooth connectivity with the Amarino API [11]. Amarino is a toolkit that connects Arduino and Android. I was able to get functionality with Amarino, but I did not like its implementation. First, the API was based off an unofficial Bluetooth API called Easy Bluetooth. This is mainly because Amarino was developed before the official Android Bluetooth API was released. Secondly, Amarino required its own dedicated application running in the background to connect and it would crash intermittently.

I decided to develop my own implementation that would run independently and use Android’s official Bluetooth API. To get connectivity, there are four major tasks necessary to connect with Bluetooth: setting up Bluetooth, finding devices that are either paired or available in the local area, connecting devices, and transferring data between devices [14].

The BluetoothAdapter class represents the local Bluetooth radio on the phone. It allows you to perform fundamental Bluetooth tasks, such as initiate device discovery, query a list of paired devices, instantiate a device using a known MAC address, and create a server socket to listen for communications from other devices.

The BluetoothSocket class is used to both initiate an outgoing connection and to manage the connection. Since the Bluetooth stack uses the SDP to gather information about the devices and its parameters, I chose the generic serial port UUID for my connection. Once the socket is connected, data can be exchanged with another Bluetooth device via
InputStream and OutputStream objects which read and send data in bytes. This allows the application to send and receive data serially via the RFCOMM Bluetooth layer.

**Receiving and Sending Data**

Once connection was established, I needed a standard to parse the constant data packets received. In addition, I would need packet flags to determine the status of the relay. Using a “/” as a delimiter, I set “7777” as flag for when the relay was switched on and “8888” for when the relay was switched off. Figure 11 shows an example of what a typical input data stream would look like. I sent the current data as a RMS value in milliamps. This is discussed in detail in the Current Sensor Measurements section.

```
20/26/32/76/43/7777/43/123/154/234/…
```

**Figure 11: Typical Input Data Stream**

For sending data to the microcontroller, a delimiter wasn’t used because there was not a constant stream. When Bluetooth is connected initially, a “3” flag is sent to check the status of the relay. A “1” is sent to switch the relay on and a “2” is sent to switch it off.

So when the phone sends a “1”, the relay is switched on and “7777” is sent back to the phone. The message flags were an easy way to keep the phone and microcontroller on the same page.
**Current Sensor Measurements**

The ACS712 outputs a voltage that shows how much current is currently being sensed. This value is then sent to the microcontroller which converts it into a digital value. The following formula was used to convert the value back into amperage.

\[
Amp = \frac{(ADC \ Value - 511) \times \frac{1 \ Bit}{204.8 \ V} \times \frac{1 \ Amp}{0.185 \ V}}
\]

Since the sensor outputs 2.5V for 0A, about half of the ADC value was subtracted (1023 /2) in order to account for this offset. And even though 5V was supplied to the sensor, its sensitivity was 185mV/A so it would only output 3.425V for 5A.

When using the current sensor to measure power, it is important to recognize that the sensor outputs instantaneous current. Figure 12 shows the raw ADC value plotted to prove this.

![Figure 12: Raw ADC Value Plotted](image)
In order to measure power, instantaneous current needs to be converted into RMS current. The following formula was used to find RMS current.

\[ X_{RMS} = \sqrt{\frac{X_1^2 + X_2^2 + \cdots + X_n^2}{n}} \]

I let \( n = 50 \) for 50 data points over a period and calculated what the sampling rate should be below.

\[ \frac{1}{60 \text{ Hz}} = 16.67 \text{ ms} \quad \text{and} \quad \frac{16.67}{50} = 333\mu s \]

This means I would need to take 1 data point every 330µs 50 times to sample 120VAC at 60Hz. Figure 13 shows the output power for a 75W light bulb. This delay ended up being calibrated to 250µs. The read command was place in a for loop and then the 50 data points taken in were monitored with the load from figure 12 connected. There was some code delay from executing the for loop and reading the ADC. Once a reliable RMS current was being read, the formula below was used to calculate power.

\[ \text{Power}_{RMS} = V_{RMS} \ast I_{RMS} \]

I set VRMS to 120V even though power from the wall is not always exact. Overall, my power measurement was accurate up to about ±3W.
**Graphing Data**

For real time graphing, I used a custom View called GraphView created by Arno den Hond [16]. GraphView creates a scaled line graph with optional x and y axis labels. It has simple functions to set the max value, adjust plot size, change the color, and add data points. Data points were constantly added as current values were received.

**Power Analysis**

I measured total system power usage by using an ammeter to find out how much current each component draws and multiplying it by its supply voltage. Table 2 shows individual components power consumption and the total system power consumption. For Arduino, I powered it with four AA batteries and measured how much current it was
drawing at idle and load. I did idle and load analysis for both the relay and Bluetooth module while the current sensor was always on. Idle power consumption is when the surge protector is switch off and disconnected from the phone. Typical power consumption is when the surge protector is switched on and disconnected from the phone. Max power consumption is when the surge protector is switched on and connected to the phone.

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Source Voltage (V)</th>
<th>Measured Current (mA)</th>
<th>Power Usage (mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino - Idle</td>
<td>6</td>
<td>12.4</td>
<td>74.4</td>
</tr>
<tr>
<td>Arduino - Load</td>
<td>6</td>
<td>25.6</td>
<td>153.6</td>
</tr>
<tr>
<td>Relay - Idle</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Relay - Switching</td>
<td>5</td>
<td>213</td>
<td>1065</td>
</tr>
<tr>
<td>Relay - Switched On</td>
<td>5</td>
<td>7.6</td>
<td>38</td>
</tr>
<tr>
<td>Bluetooth - Sleep Idle</td>
<td>5</td>
<td>0.26</td>
<td>1.3</td>
</tr>
<tr>
<td>Bluetooth - Data Transfer</td>
<td>5</td>
<td>30.6</td>
<td>153</td>
</tr>
<tr>
<td>Current Sensor</td>
<td>5</td>
<td>11.7</td>
<td>58.5</td>
</tr>
<tr>
<td><strong>Idle Power Consumption</strong></td>
<td></td>
<td></td>
<td><strong>134.2</strong></td>
</tr>
<tr>
<td><strong>Typical Power Consumption</strong></td>
<td></td>
<td></td>
<td><strong>172.2</strong></td>
</tr>
<tr>
<td><strong>Max Power Consumption</strong></td>
<td></td>
<td></td>
<td><strong>1371.6</strong></td>
</tr>
</tbody>
</table>

Table 2: System power Consumption

Overall, I feel like my power analysis was satisfactory. Idle power consumption was about 135mW, which is lower than most electronic devices. If more components are connected to the surge protector at once, the amount of reduced standby power becomes more significant. A slight drawback is the system consumes 172mW extra when the surge protector is switched on. This is due to the inductor copper losses in the relay and the BJT driving the relay.
Problems and Solutions

Since I’ve had limited or no experience using most of the components in this project, there were inevitable problems that occurred. My problems varied from not supplying enough current to the relay, the Bluetooth module crashing intermittently, and measuring current accurately.

**Supplying Current to the Relay and Flyback Voltage Protection**

According to the datasheets for the relay and the Arduino, the relay requires at least 80mA in order to switch and the Arduino digital pin only supplies up to 40mA. To fix this, I used a BJT and a 1kΩ resistor to increase the supplied current. So when the digital pin is high, the BJT will operate in forward active mode.

\[ I_b = \frac{5V - 0.7V}{1k\Omega} = 4.3mA \]

\[ I_c = \beta f \times I_b = 50 \times 4.3mA = 215mA \]

The minimum DC current gain (Beta) for a 2N2222A BJT is ~50. This circuit provides enough current to switch the relay.

Since the relay’s control terminals are controlled by a coil, it is an inductive load. When the flow of current to an inductor is suddenly stopped, a large voltage appears across that inductor in the opposite direction of the current that was stopped. This, flyback voltage, appears because energy is stored in the inductor’s magnetic field before the current is stopped. After the current has stopped, the magnetic field in the inductor releases its energy as electricity back into the circuit but in the reverse direction that it was supplied [10]. To fix this, a diode is placed in parallel with the inductor so the diode becomes active and conducts
current back through the inductor. This can cause serious damage to hardware if not addressed correctly.

**Bluetooth Issues**

I had issues with Bluetooth crashing throughout the project. In addition, I’m not sure whether it’s the BlueSMiRF module or the phone’s Bluetooth connection that is causing the crashing or if they both contribute to it. I was able to reduce the crashing by placing a delay in how often the phone reads from the input stream. I believe the constant stream of data overloaded the buffer. One drawback to this solution was the phone would occasionally miss a receive command that adjusts the status indicator. Placing a delay in how often the phone checks the input stream causes data to not be read and lost.

Another solution I tried to implement was limiting the amount of data the microcontroller sends to the phone. However, I had to slow the data stream outputting by about one second before it had a worthy effect on the crashing. Unfortunately this made the microcontroller very unresponsive to any of the commands sent from the phone.

My final implementation uses the delay of polling the input stream on the phone. Even though there is data loss, it has less of an effect on functionality.

**Measuring Current**

Even though the current measurements were fairly accurate, there are some flaws to its implementation. The ACS712’s accuracy was only 27mA per bit and it did not have a stable reading even when no connection was made to the sensing pins. Figure 14 shows the sensor outputting 51mA even though the relay was switched off. I believe one cause of this
issue is the current sensor is Hall Effect based and the relay is inductive. The relay may contribute some noise to the current sensor as they are in series with each other.

Figure 14: Output Current with Surge Protector Off
Conclusion

As the number of consumer devices increases in households, standby power becomes more of an issue. This project aims to fix this issue by creating a system that can consume less power when electronic devices are not in use. When the surge protector is switched off, only about 134mW is consumed. This amounts to less than a typical, single electronic device. If multiple devices are connected, the power savings becomes more significant. When the relay is switching, power consumption jumps to about 1.4W. However, this spike is temporary as the relay coil behaves like a short circuit when once it charges up. 172 mW of power is still consumed when the relay is switched on mainly due to inductor copper losses.

I believe raising awareness of standby power to average consumers would change usage habits. If people were aware of how much standby power their devices were using, there is a greater chance that they will find a way to switch it off and save power. My answer to this was to present power consumption graphically. Being able to visually see how much power is being used by components when in use and when not in use could easily convey usage habits to the user.

Even though this system cuts down on standby power and is an improvement over available surge protectors with remote switching, the cost of the components make it less practical. For this system implemented, the relay, Bluetooth module, and microcontroller cost about $77. It would take using this system diligently for about two years to pay for itself. An implementation with cheaper components could make this system a more viable solution.
Finally, the interfacing of Arduino to Android via Bluetooth is a viable implementation that could be used for any type of project. As of May 10th, 2011, Google announced the Android Open Accessory Development Kit (ADK). This provides an implementation library of an Android USB accessory that is based off the Arduino platform. Currently, implementation via USB is only supported but Bluetooth support is planned for the future. Since Android ADK is only supported for Android versions 2.3.4 and up, the implementation done in this project can be used for older versions or to get a head start on projects using Bluetooth.
Bibliography


8. Android 2.3 Compatibility Definition. Google Inc., 2010. PDF.


Appendix

Parts List

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino Nano Microcontroller</td>
<td>34.95</td>
</tr>
<tr>
<td>ACS 712 Current Sensor</td>
<td>9.95</td>
</tr>
<tr>
<td>Omron G5SB-5VDC Relay</td>
<td>1.95</td>
</tr>
<tr>
<td>BlueSMiRF Silver</td>
<td>39.95</td>
</tr>
<tr>
<td>Surge Protector</td>
<td>9.99</td>
</tr>
<tr>
<td>12 Gauge Wire (10 ft)</td>
<td>6.99</td>
</tr>
<tr>
<td>Protoboard</td>
<td>3.95</td>
</tr>
<tr>
<td>NPN 2N2222A BJT</td>
<td>0.35</td>
</tr>
<tr>
<td>1N4001 Diode</td>
<td>0.10</td>
</tr>
<tr>
<td>1 kΩ resistor</td>
<td>0.10</td>
</tr>
<tr>
<td>1 nF capacitor</td>
<td>0.10</td>
</tr>
<tr>
<td>Total</td>
<td>$108.38</td>
</tr>
</tbody>
</table>

Schedule

<table>
<thead>
<tr>
<th>Task</th>
<th>Begin Date</th>
<th>End Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Hardware</td>
<td>1/20/2011</td>
<td>2/7/2011</td>
</tr>
<tr>
<td>Control Relay with Arduino</td>
<td>2/7/2011</td>
<td>2/12/2011</td>
</tr>
</tbody>
</table>
#include <NewSoftSerial.h>

// (RX, TX) for bluesmirf connections
NewSoftSerial nss(0, 1);

// Flags sent to phone
String flagon = "7777/";
String flagoff = "8888/";

// ascii to int pointer used
char* val = "0";

int RelaySwitchpin = 2; // sets relay to digital pin 2
int Currentpin = A7; // sets current sensor to analog pin 7
int temp[50]; // array used to store adc values
int Value;

// counting variables
int i;
int j;
int k;

// variables to convert instantaneous current to rms current
float AmpValue[50];
float sum;
float Irms;

void setup()
{
    // initialize serial communications at 57600 bps:
    nss.begin(57600);

    // sets relay pin to an output
    pinMode(RelaySwitchpin, OUTPUT);
}

void loop()
{
    sum = 0;
    i = 0;
    // take 50 adc readings over a single period
    for(i = 0; i < 50; i++)
    {
        temp[i] = analogRead(Currentpin);
        // math calculated as 333 microseconds, optimized with code delays as 250
delayMicroseconds(250);
    }
    j = 0;
    for(j = 0; j < 50; j++)
    {
        AmpValue[j] = (temp[j] - 511)*.02638; // converts analog reading to amps
        sum = sum + AmpValue[j] * AmpValue[j]; // performing squaring and summing step of rms
    }
    sum = sum / 50; // dividing by 50
    Irms = 1000*sqrt(sum); // rms value * 1000 to round value to nearest mA
    Value = (int) Irms; // typecasting to remove decimals
nss.print(Value); //send data
nss.print("/"); //send ACK flag
delay(190); //wait for 190 ms
val[0] = nss.read(); //receive data from phone
k = atoi(val); //convert string to int
//perform 1 of 3 functions depending on value received from the phone
if(k == 1)
{
    turnon();
}
if(k == 2)
{
    turnoff();
}
if(k == 3)
{
    check();
}

//turns on relay and sends phone the ON flag
void turnon(){
digitalWrite(RelaySwitchpin, HIGH);
nss.print(flagon);
delay(100);
}

//turns off relay and sends phone the OFF flag
void turnoff(){
digitalWrite(RelaySwitchpin, LOW);
nss.print(flagoff);
delay(100);
}

//performs check function that notifies the phone the relay's status
//check is only performed when the phone connects with the bluetooth
void check(){
if(digitalRead(RelaySwitchpin) == HIGH)
{
    nss.print(flagon);
delay(100);
}
else
{
    nss.print(flagoff);
delay(100);
}

Android Code
Surge.java – Main Activity
package com.calpoly.surge;
import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.Intent;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

/**
 * This is the main Activity that displays app functions.
 */
public class Surge extends Activity implements OnClickListener{
    // Debugging
    private static final String TAG = "Surge";
    private static final boolean D = true;

    // Message types sent from the BluetoothService Handler
    public static final int MESSAGE_STATE_CHANGE = 1;
    public static final int MESSAGE_READ = 2;
    public static final int MESSAGE_WRITE = 3;
    public static final int MESSAGE_DEVICE_NAME = 4;
    public static final int MESSAGE_TOAST = 5;

    // Key names received from the BluetoothService Handler
    public static final String DEVICE_NAME = "device_name";
    public static final String TOAST = "toast";

    // Intent request codes
    private static final int REQUEST_CONNECT_DEVICE = 1;
    private static final int REQUEST_ENABLE_BT = 2;

    // Layout Views
    private Button ConnectBtn;
    private Button DisconnectBtn;
    private TextView CurrentSensorValue;
    private TextView DisplayAmpValue;
    private TextView DisplayPowerValue;
    private TextView mTitle;
    private GraphView mGraph;

    // Name of the connected device
    private String mConnectedDeviceName = null;

    // Local Bluetooth adapter
    private BluetoothAdapter mBluetoothAdapter = null;

    // Member object for the chat services
    private BluetoothService mChatService = null;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        if(D) Log.e(TAG, "+++ ON CREATE +++");

        // Set up the window layout
requestWindowFeature(Window.FEATURE_CUSTOM_TITLE);
setContentView(R.layout.main);
getWindow().setFeatureInt(Window.FEATURE_CUSTOM_TITLE, R.layout.custom_title);

// Set up the custom title
mTitle = (TextView) findViewById(R.id.title_left_text);
mTitle.setText(R.string.app_name);

mTitle = (TextView) findViewById(R.id.title_right_text);

// Get local Bluetooth adapter
mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();

// Set up Layout Buttons and Views
ConnectBtn = (Button) findViewById(R.id.connect_btn);
DisconnectBtn = (Button) findViewById(R.id.disconnect_btn);

CurrentSensorValue = (TextView) findViewById(R.id.rawValue);
DisplayAmpValue = (TextView) findViewById(R.id.currentValue);
DisplayPowerValue = (TextView) findViewById(R.id.powerValue);

mGraph = (GraphView) findViewById(R.id.graph);

mGraph.setMaxValue(800);
ConnectBtn.setOnClickListener(this);
DisconnectBtn.setOnClickListener(this);

// If the adapter is null, then Bluetooth is not supported
if (mBluetoothAdapter == null) {
    Toast.makeText(this, "Bluetooth is not available", Toast.LENGTH_LONG).show();
    finish();
    return;
}

@Override
public void onStart() {
    super.onStart();
    if (D) Log.e(TAG, "++ ON START ++");

    // If BT is not on, request that it be enabled.
    // setupChat() will then be called during onActivityResult
    if (!mBluetoothAdapter.isEnabled()) {
        Intent enableIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
        startActivityForResult(enableIntent, REQUEST_ENABLE_BT);
    } else {
        if (mChatService == null)
            setupChat();
    }
}

@Override
public synchronized void onResume() {
    super.onResume();
    if (D) Log.e(TAG, "+ ON RESUME +");

    // Performing this check in onResume() covers the case in which BT was
    // not enabled during onStart(), so we were paused to enable it...
    // onResume() will be called when ACTION_REQUEST_ENABLE activity returns.
    if (mChatService != null) {
        // Only if the state is STATE_NONE, do we know that we haven't started already
        if (mChatService.getState() == BluetoothService.STATE_NONE) {
            // Start the Bluetooth chat services
            mChatService.start();
        }
    }
}
private void setupChat() {
    Log.d(TAG, "setupChat()");
    // Initialize the BluetoothChatService to perform bluetooth connections
    mChatService = new BluetoothService(this, mHandler);
    //String check = "3";
    //sendMessage(check);
}

public void onClick(View v) {
    switch (v.getId()) {
        case R.id.connect_btn:
            String flagon = "1";
            sendMessage(flagon);
            break;
        case R.id.disconnect_btn:
            String flagoff = "2";
            sendMessage(flagoff);
            break;
        default:
            break;
    }
}

@Override
public synchronized void onPause() {
    super.onPause();
    if (D) Log.e(TAG, "- ON PAUSE -");
}

@Override
public void onStop() {
    super.onStop();
    if (D) Log.e(TAG, "-- ON STOP --");
}

@Override
public void onDestroy() {
    super.onDestroy();
    // Stop the Bluetooth chat services
    if (mChatService != null) mChatService.stop();
    if (D) Log.e(TAG, "--- ON DESTROY ---");
}

private void ensureDiscoverable() {
    if (D) Log.d(TAG, "ensure discoverable");
    if (mBluetoothAdapter.getScanMode() != BluetoothAdapter.SCAN_MODE_CONNECTABLE_DISCOVERABLE) {
        Intent discoverableIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
        discoverableIntent.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION, 300);
        startActivity(discoverableIntent);
    }
}

/**
* Sends a message.
* @param message  A string of text to send.
*/
private void sendMessage(String message) {
    // Check that we're actually connected before trying anything
    if (mChatService.getState() != BluetoothService.STATE_CONNECTED) {
        Toast.makeText(this, R.string.not_connected, Toast.LENGTH_SHORT).show();
        return;
    }
    // Get the message bytes and tell the BluetoothChatService to write
    byte[] send = message.getBytes();
    mChatService.write(send);
}

private void receive(String message) {
    String AmpOut = null;
    String PowerOut = null;
    float AmpValue;
    float Power;
    float Plot;
    //Parse string message into int
    final int bluetoothReading = Integer.parseInt(message);
    if(bluetoothReading == 7777){
        //If 7777 received, turn green indicator on
        findViewById(R.id.connected).setBackgroundResource(R.color.connected_on);
        findViewById(R.id.disconnected).setBackgroundResource(R.color.disconnected_off);
    } else if(bluetoothReading == 8888){
        //If 8888 received, turn red indicator on
        findViewById(R.id.connected).setBackgroundResource(R.color.connected_off);
        findViewById(R.id.disconnected).setBackgroundResource(R.color.disconnected_on);
    } else{
        //Convert value to Amps
        AmpValue = (float)bluetoothReading / 1000;
        Power = AmpValue*120;
        Plot = Power + 300;
        //Converts values to string to display as text
        AmpOut = String.valueOf(AmpValue);
        PowerOut = String.valueOf(Power);
        //Displays data
        CurrentSensorValue.setText(message);
        DisplayAmpValue.setText(AmpOut);
        DisplayPowerValue.setText(PowerOut);
        //Plots data with 300 offset
        mGraph.addDataPoint(Plot);
    }
}

// The Handler that gets information back from the BluetoothChatService
private final Handler mHandler = new Handler() {

    @Override
    public void handleMessage(Message msg) {
        switch (msg.what) {
        case MESSAGE_STATE_CHANGE:
            if(D) Log.i(TAG, "MESSAGE_STATE_CHANGE: " + msg.arg1);
            switch (msg.arg1) {
            
            }

        }
    }

    public void handleMessage(Message msg) {
        switch (msg.what) {
        case MESSAGE_STATE_CHANGE:
            if(D) Log.i(TAG, "MESSAGE_STATE_CHANGE: " + msg.arg1);
            switch (msg.arg1) {
            
            }

    }
case BluetoothService.STATE_CONNECTED:
    mTitle.setText(R.string.title_connected_to);
    mTitle.append(mConnectedDeviceName);
    break;
case BluetoothService.STATE_CONNECTING:
    mTitle.setText(R.string.title_connecting);
    break;
case BluetoothService.STATE_LISTEN:
    break;
case BluetoothService.STATE_NONE:
    mTitle.setText(R.string.title_not_connected);
    break;
}
break;
case MESSAGE_WRITE:
    byte[] writeBuf = (byte[]) msg.obj;
    String writeMessage = new String(writeBuf);
    break;
case MESSAGE_READ:
    byte[] readBuf = (byte[]) msg.obj;
    // construct a string from the valid bytes in the buffer
    String readMessage = new String(readBuf, 0, msg.arg1);
    //Splits string coming in by /
    String[] tokens = readMessage.split("/");
    int i = 0;
    String data = null;
    //converts string[] to string
    for(i = 0; i < tokens.length; i++)
    {
        data = tokens[i];
    }
    receive(data);
    break;
case MESSAGE_DEVICE_NAME:
    // save the connected device's name
    mConnectedDeviceName = msg.getData().getString(DEVICE_NAME);
    Toast.makeText(getApplicationContext(), "Connected to " + mConnectedDeviceName, Toast.LENGTH_SHORT).show();
    break;
case MESSAGE_TOAST:
    Toast.makeText(getApplicationContext(), msg.getData().getString(PORT), Toast.LENGTH_SHORT).show();
    break;
}
case REQUEST_ENABLE_BT:
    // When the request to enable Bluetooth returns
    if (resultCode == Activity.RESULT_OK) {
        // Bluetooth is now enabled, so set up a chat session
        setupChat();
    } else {
        // User did not enable Bluetooth or an error occurred
        Log.d(TAG, "BT not enabled");
        Toast.makeText(this, R.string.bt_not_enabled_leaving, Toast.LENGTH_SHORT).show();
        finish();
    }
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.option_menu, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
    case R.id.scan:
        // Launch the DeviceListActivity to see devices and do scan
        Intent serverIntent = new Intent(this, DeviceListActivity.class);
        startActivityForResult(serverIntent, REQUEST_CONNECT_DEVICE);
        return true;
    case R.id.discoverable:
        // Ensure this device is discoverable by others
        ensureDiscoverable();
        return true;
    }
    return false;
}

BluetoothService.java – Sets up and manages Bluetooth connection

package com.calpoly.surge;
* Copyright (C) 2009 The Android Open Source Project
* *
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
* *
* http://www.apache.org/licenses/LICENSE-2.0
* *
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
* */

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.UUID;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothServerSocket;
import android.bluetooth.BluetoothSocket;
import android.content.Context;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.util.Log;

/**
 * This class does all the work for setting up and managing Bluetooth
 * connections with other devices. It has a thread that listens for
 * incoming connections, a thread for connecting with a device, and a
 * thread for performing data transmissions when connected.
 */
public class BluetoothService {
    // Debugging
    private static final String TAG = "BluetoothService";
    private static final boolean D = true;

    // Name for the SDP record when creating server socket
    private static final String NAME = "BluetoothChat";

    // Unique UUID for this application
    private static final UUID MY_UUID = UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");

    // Member fields
    private final BluetoothAdapter mAdapter;
    private final Handler mHandler;
    private AcceptThread mAcceptThread;
    private ConnectThread mConnectThread;
    private ConnectedThread mConnectedThread;
    private int mState;

    // Constants that indicate the current connection state
    public static final int STATE_NONE = 0;       // we're doing nothing
    public static final int STATE_LISTEN = 1;     // now listening for incoming connections
    public static final int STATE_CONNECTING = 2; // now initiating an outgoing connection
    public static final int STATE_CONNECTED = 3;  // now connected to a remote device

    /**
     * Constructor. Prepares a new BluetoothChat session.
     * @param context  The UI Activity Context
     * @param handler  A Handler to send messages back to the UI Activity
     */
    public BluetoothService(Context context, Handler handler) {
        mAdapter = BluetoothAdapter.getDefaultAdapter();
        mState = STATE_NONE;
        mHandler = handler;
    }

    /**
     * Set the current state of the chat connection
     * @param state  An integer defining the current connection state
     */
    private synchronized void setState(int state) {
        // Code implementation
    }
}
if (D) Log.d(TAG, "setState() " + mState + " -> " + state);
mState = state;

// Give the new state to the Handler so the UI Activity can update
mHandler.obtainMessage(Surge.MESSAGE_STATE_CHANGE, state, -1).sendToTarget();
}

/**
 * Return the current connection state. */
public synchronized int getState() {
    return mState;
}

/**
 * Start the chat service. Specifically start AcceptThread to begin a
 * session in listening (server) mode. Called by the Activity onResume() */
public synchronized void start() {
    if (D) Log.d(TAG, "start");
    // Cancel any thread attempting to make a connection
    if (mConnectThread != null) {mConnectThread.cancel(); mConnectThread = null;}
    // Cancel any thread currently running a connection
    if (mConnectedThread != null) {mConnectedThread.cancel(); mConnectedThread = null;}
    // Start the thread to listen on a BluetoothServerSocket
    if (mAcceptThread == null) {
        mAcceptThread = new AcceptThread();
        mAcceptThread.start();
    }
    setState(STATE_LISTEN);
}

/**
 * Start the ConnectThread to initiate a connection to a remote device.
 * @param device The BluetoothDevice to connect
 */
public synchronized void connect(BluetoothDevice device) {
    if (D) Log.d(TAG, "connect to: " + device);
    // Cancel any thread attempting to make a connection
    if (mState == STATE_CONNECTING) {
        if (mConnectThread != null) {mConnectThread.cancel(); mConnectThread = null;}
    }
    // Cancel any thread currently running a connection
    if (mConnectedThread != null) {mConnectedThread.cancel(); mConnectedThread = null;}
    // Start the thread to connect with the given device
    mConnectThread = new ConnectThread(device);
    mConnectThread.start();
    setState(STATE_CONNECTING);
}

/**
 * Start the ConnectedThread to begin managing a Bluetooth connection
 * @param socket The BluetoothSocket on which the connection was made
 */
public synchronized void connected(BluetoothSocket socket, BluetoothDevice device) {  
    if (D) Log.d(TAG, "connected");

    // Cancel the thread that completed the connection
    if (mConnectThread != null) {mConnectThread.cancel(); mConnectThread = null;}

    // Cancel any thread currently running a connection
    if (mConnectedThread != null) {mConnectedThread.cancel(); mConnectedThread = null;}

    // Cancel the accept thread because we only want to connect to one device
    if (mAcceptThread != null) {mAcceptThread.cancel(); mAcceptThread = null;}

    // Start the thread to manage the connection and perform transmissions
    mConnectedThread = new ConnectedThread(socket);
    mConnectedThread.start();

    // Send the name of the connected device back to the UI Activity
    Message msg = mHandler.obtainMessage(Surge.MESSAGE_DEVICE_NAME);
    Bundle bundle = new Bundle();
    bundle.putString(Surge.DEVICE_NAME, device.getName());
    msg.setData(bundle);
    mHandler.sendMessage(msg);
    setState(STATE_CONNECTED);
}
/* Indicate that the connection attempt failed and notify the UI Activity. */
private void connectionFailed() {
    setState(STATE_LISTEN);
    // Send a failure message back to the Activity
    Message msg = mHandler.obtainMessage(Surge.MESSAGE_TOAST);
    Bundle bundle = new Bundle();
    bundle.putString(Surge.TOAST, "Unable to connect device");
    msg.setData(bundle);
    mHandler.sendMessage(msg);
}

/* Indicate that the connection was lost and notify the UI Activity. */
private void connectionLost() {
    setState(STATE_LISTEN);
    // Send a failure message back to the Activity
    Message msg = mHandler.obtainMessage(Surge.MESSAGE_TOAST);
    Bundle bundle = new Bundle();
    bundle.putString(Surge.TOAST, "Device connection was lost");
    msg.setData(bundle);
    mHandler.sendMessage(msg);
}

/* This thread runs while listening for incoming connections. It behaves like a server-side client. It runs until a connection is accepted (or until cancelled). */
private class AcceptThread extends Thread {
    // The local server socket
    private final BluetoothServerSocket mmServerSocket;

    public AcceptThread() {
        BluetoothServerSocket tmp = null;
        // Create a new listening server socket
        try {
            tmp = mAdapter.listenUsingRfcommWithServiceRecord(NAME, MY_UUID);
        } catch (IOException e) {
            Log.e(TAG, "listen() failed", e);
        }
        mmServerSocket = tmp;
    }

    @Override
    public void run() {
        if (D) Log.d(TAG, "BEGIN mAcceptThread" + this);
        setName("AcceptThread");
        BluetoothServerSocket mmServerSocket = tmp;

        public void run() {
            if (D) Log.d(TAG, "BEGIN mAcceptThread" + this);
            setName("AcceptThread");
            BluetoothServerSocket socket = null;

            // Listen to the server socket if we're not connected
            while (mState != STATE_CONNECTED) {
                try {
                    // This is a blocking call and will only return on a
// successful connection or an exception
socket = mmServerSocket.accept();
} catch (IOException e) {
    Log.e(TAG, "accept() failed", e);
    break;
}

// If a connection was accepted
if (socket != null) {
    synchronized (BluetoothService.this) {
        switch (mState) {
        case STATE_LISTEN:
        case STATE_CONNECTING:
            // Situation normal. Start the connected thread.
            connected(socket, socket.getRemoteDevice());
            break;
        case STATE_NONE:
        case STATE_CONNECTED:
            // Either not ready or already connected. Terminate new socket.
            try {
                socket.close();
            } catch (IOException e) {
                Log.e(TAG, "Could not close unwanted socket", e);
            }
            break;
        }
    }
    if (D) Log.i(TAG, "END mAcceptThread");
}

public void cancel() {
    if (D) Log.d(TAG, "cancel " + this);
    try {
        mmServerSocket.close();
    } catch (IOException e) {
        Log.e(TAG, "close() of server failed", e);
    }
}

/**
 * This thread runs while attempting to make an outgoing connection
 * with a device. It runs straight through; the connection either
 * succeeds or fails.
 */
private class ConnectThread extends Thread {
    private final BluetoothSocket mmSocket;
    private final BluetoothDevice mmDevice;

    public ConnectThread(BluetoothDevice device) {
        mmDevice = device;
        BluetoothSocket tmp = null;

        // Get a BluetoothSocket for a connection with the
// given BluetoothDevice
try {
    tmp = device.createRfcommSocketToServiceRecord(MY_UUID);
} catch (IOException e) {
    Log.e(TAG, "create() failed", e);
}
mmSocket = tmp;
}

public void run() {
    Log.i(TAG, "BEGIN mConnectThread");
    setName("ConnectThread");

    // Always cancel discovery because it will slow down a connection
    mAdapter.cancelDiscovery();

    // Make a connection to the BluetoothSocket
    try {
        // This is a blocking call and will only return on a
        // successful connection or an exception
        mmSocket.connect();
    } catch (IOException e) {
        connectionFailed();
        // Close the socket
        try {
            mmSocket.close();
        } catch (IOException e2) {
            Log.e(TAG, "unable to close() socket during connection failure", e2);
        }
        // Start the service over to restart listening mode
        BluetoothService.this.start();
        return;
    }

    // Reset the ConnectThread because we're done
    synchronized (BluetoothService.this) {
        mConnectThread = null;
    }

    // Start the connected thread
    connected(mmSocket, mmDevice);
}

public void cancel() {
    try {
        mmSocket.close();
    } catch (IOException e) {
        Log.e(TAG, "close() of connect socket failed", e);
    }
}

/**
 * This thread runs during a connection with a remote device.
 * It handles all incoming and outgoing transmissions.
 */
private class ConnectedThread extends Thread {

}
private final BluetoothSocket mmSocket;
private final InputStream mmInStream;
private final OutputStream mmOutStream;

public ConnectedThread(BluetoothSocket socket) {
    Log.d(TAG, "create ConnectedThread");
    mmSocket = socket;
    InputStream tmpIn = null;
    OutputStream tmpOut = null;

    // Get the BluetoothSocket input and output streams
    try {
        tmpIn = socket.getInputStream();
        tmpOut = socket.getOutputStream();
    } catch (IOException e) {
        Log.e(TAG, "temp sockets not created", e);
    }

    mmInStream = tmpIn;
    mmOutStream = tmpOut;
}

public void run() {
    Log.i(TAG, "BEGIN mConnectedThread");
    byte[] buffer = new byte[1024];
    int bytes;

    // Keep listening to the InputStream while connected
    while (true) {
        try {
            //300 ms delay
            Thread.sleep(300);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

        // Read from the InputStream
        bytes = mmInStream.read(buffer);

        // Send the obtained bytes to the UI Activity
        mHandler.obtainMessage(Surge.MESSAGE_READ, bytes, -1, buffer)
            .sendToTarget();
    } catch (IOException e) {
        Log.e(TAG, "disconnected", e);
        connectionLost();
        break;
    }
}

/**
 * Write to the connected OutStream.
 * @param buffer The bytes to write
 */
public void write(byte[] buffer) {
try {
    mmOutStream.write(buffer);

    // Share the sent message back to the UI Activity
    mHandler.obtainMessage(Surge.MESSAGE_WRITE, -1, -1, buffer)
        .sendToTarget();
} catch (IOException e) {
    Log.e(TAG, "Exception during write", e);
}

public void cancel() {
    try {
        mmSocket.close();
    } catch (IOException e) {
        Log.e(TAG, "close() of connect socket failed", e);
    }
}

DeviceListActivity.java – Pop up dialog that lists devices to pair

package com.calpoly.surge;

import java.util.Set;
import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.view.Window;
import android.view.View.OnClickListener;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.AdapterView.OnItemClickListener;

/**
 * This Activity appears as a dialog. It lists any paired devices and
devices detected in the area after discovery. When a device is chosen
by the user, the MAC address of the device is sent back to the parent
Activity in the result Intent.
*/
public class DeviceListActivity extends Activity {

    // Debugging
    private static final String TAG = "DeviceListActivity";
    private static final boolean D = true;

    // Return Intent extra
public static String EXTRA_DEVICE_ADDRESS = "device_address";

// Member fields
private BluetoothAdapter mBtAdapter;
private ArrayAdapter<String> mPairedDevicesArrayAdapter;
private ArrayAdapter<String> mNewDevicesArrayAdapter;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // Setup the window
    requestWindowFeature(Window.FEATURE_INDETERMINATE_PROGRESS);
    setContentView(R.layout.device_list);

    // Set result CANCELED incase the user backs out
    setResult(Activity.RESULT_CANCELED);

    // Initialize the button to perform device discovery
    Button scanButton = (Button) findViewById(R.id.button_scan);
    scanButton.setOnClickListener(new OnClickListener() {
        public void onClick(View v) {
            doDiscovery();
            v.setVisibility(View.GONE);
        }
    });

    // Initialize array adapters. One for already paired devices and
    // one for newly discovered devices
    mPairedDevicesArrayAdapter = new ArrayAdapter<String>(this,
                                                            R.layout.device_name);
    mNewDevicesArrayAdapter = new ArrayAdapter<String>(this, R.layout.device_name);

    // Find and set up the ListView for paired devices
    ListView pairedListView = (ListView) findViewById(R.id.paired_devices);
    pairedListView.setAdapter(mPairedDevicesArrayAdapter);
    pairedListView.setOnItemClickListener(mDeviceClickListener);

    // Find and set up the ListView for newly discovered devices
    ListView newDevicesListView = (ListView) findViewById(R.id.new_devices);
    newDevicesListView.setAdapter(mNewDevicesArrayAdapter);
    newDevicesListView.setOnItemClickListener(mDeviceClickListener);

    // Register for broadcasts when a device is discovered
    IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
    this.registerReceiver(mReceiver, filter);

    // Register for broadcasts when discovery has finished
    filter = new IntentFilter(BluetoothAdapter.ACTION_DISCOVERY_FINISHED);
    this.registerReceiver(mReceiver, filter);

    // Get the local Bluetooth adapter
    mBtAdapter = BluetoothAdapter.getDefaultAdapter();

    // Get a set of currently paired devices
    Set<BluetoothDevice> pairedDevices = mBtAdapter.getBondedDevices();

    // If there are paired devices, add each one to the ArrayAdapter
if (pairedDevices.size() > 0) {
    findViewById(R.id.title_paired_devices).setVisibility(View.VISIBLE);
    for (BluetoothDevice device : pairedDevices) {
        mPairedDevicesArrayAdapter.add(device.getName() + "\n" + device.getAddress());
    }
} else {
    String noDevices = getResources().getText(R.string.none_paired).toString();
    mPairedDevicesArrayAdapter.add(noDevices);
}

@Override
protected void onDestroy() {
    super.onDestroy();

    // Make sure we're not doing discovery anymore
    if (mBtAdapter != null) {
        mBtAdapter.cancelDiscovery();
    }

    // Unregister broadcast listeners
    this.unregisterReceiver(mReceiver);
}

/**
 * Start device discover with the BluetoothAdapter
 */
private void doDiscovery() {
    if (D) Log.d(TAG, "doDiscovery()");

    // Indicate scanning in the title
    setProgressBarIndeterminateVisibility(true);
    setTitle(R.string.scanning);

    // Turn on sub-title for new devices
    findViewById(R.id.title_new_devices).setVisibility(View.VISIBLE);

    // If we're already discovering, stop it
    if (mBtAdapter.isDiscovering()) {
        mBtAdapter.cancelDiscovery();
    }

    // Request discover from BluetoothAdapter
    mBtAdapter.startDiscovery();
}

// The on-click listener for all devices in the ListViews
private OnItemClickListener mDeviceClickListener = new OnItemClickListener() {
    public void onItemClick(AdapterView<?> av, View v, int arg2, long arg3) {
        // Cancel discovery because it's costly and we're about to connect
        mBtAdapter.cancelDiscovery();

        // Get the device MAC address, which is the last 17 chars in the View
        String info = ((TextView) v).getText().toString();
        String address = info.substring(info.length() - 17);

        // Create the result Intent and include the MAC address
    }

    // The on-click listener for all devices in the ListViews
    private OnItemClickListener mDeviceClickListener = new OnItemClickListener() {
        public void onItemClick(AdapterView<?> av, View v, int arg2, long arg3) {
            // Cancel discovery because it's costly and we're about to connect
            mBtAdapter.cancelDiscovery();

            // Get the device MAC address, which is the last 17 chars in the View
            String info = ((TextView) v).getText().toString();
            String address = info.substring(info.length() - 17);

            // Create the result Intent and include the MAC address
        }
    }
Intent intent = new Intent();
intent.putExtra(EXTRA_DEVICE_ADDRESS, address);

// Set result and finish this Activity
setResult(Activity.RESULT_OK, intent);
finish();
}
};

// The BroadcastReceiver that listens for discovered devices and
// changes the title when discovery is finished
private final BroadcastReceiver mReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        // When discovery finds a device
        if (BluetoothDevice.ACTION_FOUND.equals(action)) {
            // Get the BluetoothDevice object from the Intent
            BluetoothDevice device = intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
            // If it's already paired, skip it, because it's been listed already
            if (device.getBondState() != BluetoothDevice.BOND_BONDED) {
                mNewDevicesArrayAdapter.add(device.getName() + device.getAddress());
            }
            // When discovery is finished, change the Activity title
        } else if (BluetoothAdapter.ACTION_DISCOVERY_FINISHED.equals(action)) {
            setProgressBarIndeterminateVisibility(false);
            setTitle(R.string.select_device);
            if (mNewDevicesArrayAdapter.getCount() == 0) {
                String noDevices = getResources().getText(R.string.none_found).toString();
                mNewDevicesArrayAdapter.add(noDevices);
            }
        }
    }
};

GraphView.java – Activity that performs graphing

package com.calpoly.surge;

/**
 * GraphView creates a scaled line or bar graph with x and y axis labels.
 * @author Arno den Hond
 */
import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.util.AttributeSet;
import android.view.View;

public class GraphView extends View {


private Bitmap mBitmap;
private Paint mPaint = new Paint();
private Canvas mCanvas = new Canvas();

private float mSpeed = 1.0f;
private float mLastX;
private float mScale;
private float mLastValue;
private float mYOffset;
private int mColor;
private float mWidth;
private float maxValue = 1024f;

public GraphView(Context context) {
    super(context);
    init();
}

public GraphView(Context context, AttributeSet attrs) {
    super(context, attrs);
    init();
}

private void init(){
    mColor = Color.argb(192, 64, 128, 64);
    mPaint.setFlags(Paint.ANTI_ALIAS_FLAG);
}

public void addDataPoint(float value){
    final Paint paint = mPaint;
    float newX = mLastX + mSpeed;
    final float v = mYOffset + value * mScale;
    paint.setColor(mColor);
    mCanvas.drawLine(mLastX, mLastValue, newX, v, paint);
    mLastValue = v;
    mLastX += mSpeed;
    invalidate();
}

public void setMaxValue(int max){
    maxValue = max;
    mScale = -(mYOffset * (1.0f / maxValue));
}

public void setSpeed(float speed){
    mSpeed = speed;
}

@Override
protected void onSizeChanged(int w, int h, int oldw, int oldh) {
    mBitmap = Bitmap.createBitmap(w, h, Bitmap.Config.RGB_565);
    mCanvas.setBitmap(mBitmap);
    mCanvas.drawColor(0xFFFFFFFF);
}
myOffset = h;
mScale = -(myOffset * (1.0f / maxValue));
mWidth = w;
mLastX = mWidth;
super.onSizeChanged(w, h, oldw, oldh);
}

@Override
protected void onDraw(Canvas canvas) {
    synchronized (this) {
        if (mBitmap != null) {
            if (mLastX >= mWidth) {
                mLastX = 0;
                Canvas canv = mCanvas;
                canv.drawColor(0xFFFFFFFF);
                mPaint.setColor(0xFF777777);
                canv.drawLine(0, myOffset, mWidth, myOffset, mPaint);
            }
            canvas.drawBitmap(mBitmap, 0, 0, null);
        }
    }
}

Main.xml – Main GUI interface
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#ffffff" >

    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" >

        <ImageView
            android:id="@+id/logo"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginLeft="4dp"
            android:src="@drawable/logo"/>

        <TextView
            android:id="@+id/title"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="20dp"
            android:textSize="30sp"
            android:textStyle="bold"
            android:layout_weight="1"
            android:textColor="@color/title"
            android:text="#string/app_name"></TextView>

    </LinearLayout>

    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        android:id="@+id/title"
        android:textSize="30sp"
        android:textStyle="bold"
        android:layout_weight="1"
        android:textColor="#color/title"
        android:text="#string/app_name"></LinearLayout>
android:layout_width="fill_parent"
android:layout_height="wrap_content"
>

<Button
android:layout_marginLeft="4dp"
android:layout_height="50dp"
android:text="@string/on"
android:layout_marginTop="10dp"
android:layout_marginRight="4dp"
android:layout_width="100dp"
android:focusable="false"
android:id="@+id/connect_btn"
android:layout_alignParentTop="true">
</Button>

<Button
android:layout_height="50dp"
android:text="@string/off"
android:layout_marginTop="10dp"
android:layout_width="100dp"
android:layout_marginRight="45dp"
android:focusable="false"
android:id="@+id/disconnect_btn"
android:layout_toRightOf="@id/connect_btn">
</Button>

<View
android:layout_height="25dp"
android:layout_width="25dp"
android:id="@+id/connected"
android:layout_marginTop="10dp"
android:background="@color/connected_off"
android:layout_alignParentTop="true">
</View>

<View
android:layout_height="25dp"
android:layout_width="25dp"
android:id="@+id/disconnected"
android:layout_alignParentTop="true"
android:background="@color/disconnected_on"
android:layout_alignParentRight="true">
</View>

</LinearLayout>

<LinearLayout
android:orientation="horizontal"
android:layout_width="fill_parent"
android:layout_height="wrap_content">
<TextView
android:layout_marginLeft="4dp"
android:layout_height="wrap_content"
android:layout_marginTop="5dp"
android:textColor="@color/title"
android:layout_width="wrap_content"
android:textSize="12sp"
android:id="@+id/rawValue"></TextView>
<TextView
android:layout_height="wrap_content"
android:layout_marginTop="5dp"
android:layout_width="wrap_content"
android:layout_marginLeft="100dp"
android:textSize="12sp"
android:textColor="@color/title"
Device_list.xml – Dialog menu that lists devices to connect to

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView android:id="@+id/title_paired_devices"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content">
        Device_list.xml – Dialog menu that lists devices to connect to
    </TextView>
    <LinearLayout>
        <TextView android:layout_marginLeft="4dp"
            android:layout_height="wrap_content"
            android:layout_width="wrap_content"
            android:textColor="@color/title"
            android:layout_marginTop="5dp"
            android:text="@string/raw"
            android:id="@+id/rawText"></TextView>
        <TextView android:layout_marginLeft="50dp"
            android:layout_height="wrap_content"
            android:layout_width="wrap_content"
            android:textColor="@color/title"
            android:layout_marginTop="5dp"
            android:text="@string/current"
            android:id="@+id/currentText"></TextView>
        <TextView android:layout_marginLeft="70dp"
            android:layout_height="wrap_content"
            android:layout_width="wrap_content"
            android:textColor="@color/title"
            android:layout_marginTop="5dp"
            android:text="@string/power"
            android:id="@+id/powerText"></TextView>
    </LinearLayout>
    <com.calpoly.surge.GraphView
        android:id="@+id/graph"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="70dp"
        android:layout_marginLeft="true"
        android:layout_width="10dp"
        android:layout_height="4"></LinearLayout>
</LinearLayout>
```
Option_menu.xml – GUI for option menu popup

<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:id="@+id/scan"
    android:icon="@android:drawable/ic_menu_search"
    android:title="@string/connect" />
  <item android:id="@+id/discoverable"
    android:icon="@android:drawable/ic_menu_mylocation"
    android:title="@string/discoverable" />
</menu>

Custom_title.xml – Title bar that displays connection status

<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:gravity="center_vertical">
  <TextView android:id="@+id/title_left_text"
    android:layout_alignParentLeft="true"
    android:ellipsize="end"
    android:singleLine="true"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:gravity="center_vertical" />
  <TextView android:id="@+id/title_right_text"
    android:layout_alignParentRight="true"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" />
</RelativeLayout>
Strings.xml - Text definitions used in code

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Smart Surge</string>

    <!-- BluetoothChat -->
    <string name="send">Send</string>
    <string name="not_connected">You are not connected to a device</string>
    <string name="bt_not_enabled_leaving">Bluetooth was not enabled. Leaving Bluetooth Chat.</string>
    <string name="title_connecting">connecting...</string>
    <string name="title_connected_to">connected: </string>
    <string name="title_not_connected">not connected</string>

    <!-- DeviceListActivity -->
    <string name="scanning">scanning for devices...</string>
    <string name="select_device">select a device to connect</string>
    <string name="none_paired">No devices have been paired</string>
    <string name="none_found">No devices found</string>
    <string name="title_paired_devices">Paired Devices</string>
    <string name="title_other_devices">Other Available Devices</string>
    <string name="button_scan">Scan for devices</string>

    <!-- Options Menu -->
    <string name="connect">Connect a device</string>
    <string name="discoverable">Make discoverable</string>
    <string name="on">On</string>
    <string name="off">Off</string>
    <string name="raw">Value Received</string>
    <string name="current">Current Value</string>
    <string name="power">Power</string>
</resources>

Colors.xml – Color definitions used

<?xml version="1.0" encoding="UTF-8"?>
<resources>
    <color name="connected_on">#ff00ff00</color>
    <color name="connected_off">#2000ff00</color>
    <color name="disconnected_on">#ff000000</color>
    <color name="disconnected_off">#30ff0000</color>
    <color name="title">#00000D</color>
    <color name="title2">#ffffff</color>
</resources>
Hardware Configuration/ Layout