PROJECT A/B ECE4095

Wirelessly connected smart meter

Final Report

Student

William Robertson

Supervisors

Dr. Ahmet Sekercioglu

Prof. Jean Armstrong

Abstract

This document gives an overview of the development of a wirelessly connected smart meter, and the details of how it satisfies the requirements set forth in the requirements analysis. The core of the report is the method and results sections, which are broken up into the hardware and software sides of the implementation. The hardware specification goes into the details of the physical aspect of the project, discussing design choices and implementation methods. The software specification explores the way the two main components will interact with each other, and provides an overview of the required functions of the devices. The results section discusses the effectiveness of the implementation, and an overview of the final product. Appendices have been included for completeness, which contain full listings of the source code for both devices in the system, schematic diagrams, and PCB layouts.

Contents

Introduction

Over the past few years' the public have become increasingly aware of energy usage as a key driver of greenhouse emissions, and nowhere more than Victoria, Australia is this true. Victoria is home to the world's most polluting power station – Hazelwood coal fired power station which produces 1.58 megatonnes of carbon dioxide per terawatt-hour of energy output[1] – so a reduction of energy usage in Victoria will have a greater contribution to reducing carbon output than most other places in the world.

Energy efficiency has become not only an important strategy for reducing impact on the environment, but also a business strategy. Products touting their reduced energy consumption have become commonplace, but without a way to measure the individual impact of these devices it is difficult for consumers to differentiate between various techniques of saving energy – they might ask, "Should I replace my incandescent lights with CFL globes, or should I replace my old fridge?".

The Victorian government has begun rollout of their own smart meter system to households mostly in Melbourne, however many areas have yet to receive them and may not even have a scheduled installation date, especially in more regional areas.

This report goes into the processes in the development of a smart energy meter designed to be a low-cost interim device for areas which have not received their government sponsored meters, and contains the specific details of the construction of the system.

Background and Problem description

The fundamental purpose of the system is to measure power usage. Power is the product of both voltage and current, so for an accurate measurement of power, both of these values must be considered.

Measurement of voltage is relatively straightforward, as most microcontrollers contain the necessary hardware to directly measure analogue voltage. The measurement of current however poses a more challenging problem. There are several methods of measuring current:

- Shunt resistor
- Hall effect sensors
- Current clamp

Each method has its own advantages and disadvantages. A shunt resistor is a very common method, as it is extremely simple to implement and can provide very accurate results. However, it is required to be placed in series with the device under test. This posed a serious issue with the implementation, as although the system was to be tested in a low-voltage environment, it was desirable for the system to be scaled up for measurement at 240 volts. In addition to this, if the device was measuring high currents, a very large power resistor would be needed which can be expensive and waste significant amounts of power.

[1] http://www.wwf.org.au/articles/feature34/

Hall Effect sensors do not require the direct contact to the primary circuit that shunt resistors do. This is extremely advantageous as electrical isolation between the circuit under test and the measurement platform allows for current measurement in high voltage applications, as well as being able to measure low voltage situations as well. Hall Effect sensors however put out much smaller voltages and therefore require some kind of amplification to be able to be measured accurately, although many commercially available packages have this amplification built into them.

Current clamp sensors actually use a current transformer to induce a voltage from a current passing through them, but are very similar in operation to Hall Effect sensors, and have similar advantages and disadvantages. One main difference is that current transformer based sensors are only able to measure AC waveforms. However, this is a non-issue as all currents that need to be tested are AC. Another difference with these current clamps is that they are able to be placed around a conductor without dismantling the circuit, making installation much easier than that of a shunt resistor, or even a Hall Effect sensor. Again, amplification circuitry is required by the measurement hardware because of the small scale signal that is produced.

Method

Hardware

Measurement

At the core of the measurement hardware we have three main items:

- An ATXmega microcontroller
- An XBee wireless communication
- A Current clamp

The ATXmega microcontroller is a high powered device, with a large number of inputs and outputs and various hardware functions such as dedicated pulse width modulation, direct memory access architecture, analogue to digital conversion, digital to analogue conversion, and support for various communication protocols including I^{2}C, SPI, UART, ISP, and JTAG. The use of an ATXmega microcontroller is a non-functional requirement, and as such it is necessary to include in the design, despite being somewhat overpowered for the particular application. In this light, a 6 pin header has also been included, providing access to the 3.3v rail, GND rail, and the first four bits of the microcontroller's "PORT A". These pins can be used on the microcontroller as ADC inputs, which would allow further use of the same PCB as a general purpose wireless ADC. In the final version of the board, some of these pins have been used to allow for monitoring of voltage via an external circuit.

A shrouded 5x2 pin male header has been included on the PCB to allow the easy connection of a JTAG debugging cable. This is used to program the firmware of the ATXmega microcontroller as well as provide debugging function during the development of the software.

Various elements have been added to the design of the PCB to ease the debugging process. These include:

- A bi-colour status LED allowing the microcontroller to indicate which state it in (ie Green for OK, red for error, orange for warning)
- Four single colour LEDs which indicate the status of the XBee module.
- JTAG header mentioned above

For wireless communications a pair of XBee modules was selected. These devices were chosen due to the simplicity of operation and their ability to operate in "transparent mode" where the module can simply be passed serial data and it handles packet formation, and retransmission of lost packets. Configuration of the modules is necessary before use, and for this the digi supplied utility "X-CTU"[2] was used. The devices need to operate on the same channel of the 2.4GHz spectrum, must be a part of the same PAN and must set the destination and source addresses respectively.

One issue with the implementation of the XBee system was that the device required a very accurate USART baud rate. Initially while the ATXmega microcontroller was being clocked at the full 32MHz calculation of the necessary clock division to achieve accurate baud rates proved to be inaccurate enough that while the values had theoretically less than 1% error (2% error is common with USART baud rates), in practice the baud rates were off by enough that the microcontroller failed to communicate with the XBee module. Fortunately when the processor clock speed was reduced to 2MHz the accuracy improved dramatically and communication using calculated values was possible. Another issue with the XBee on the measuring hardware was the power draw as when the device transmits it draws a spike of power. The regulator was unable to supply the necessary current to the wireless module, and thus it was not able to transmit. This problem was fixed by the inclusion of a 47uF bypass capacitor which is able to provide the module with the current it requires for transmission.

To be measured by the ADC on the microcontroller, the current to be measured must be converted to a voltage. Different methods of conversion have been discussed in the problem description section of this report. A current clamp was chosen as the measurement device. Due to the small scale of signals that can be produced by these amplification was necessary. By using the differential ADC of the ATXmega microcontroller it was possible to use a built in programmable gain stage, which is adjustable from 1x to 64x gain. The current clamp selected was the Seeed Studio SCT-013-030[3]. This current clamp is able to measure between 0 - 30A and outputs a voltage proportional to the measured current.

[2] http://www.digi.com/support/productdetl.jsp?pid=3352&osvid=57&s=316&tp=5&tp2=0
[3]http://www.seeedstudio.com/depot/noninvasive-ac-current-sensor-30a-max-p-519.html?cPath=84_91

Internet Relay

The internet relay is designed to accept data wirelessly from the measurement hardware and relay this information to the internet.

The main concern for the selection of the hardware for this was the 802.11b/g wireless interface, as the other parts were relatively straightforward to find. On researching my options, I found two main viable devices:

- Rabbit MiniCore RCM5600W
- A Linux capable device

The Rabbit microcontroller was a promising option, however it was quite expensive, and was also unable to be coded in traditional C – development was in a proprietary Dynamic C® environment.

On the other hand, a Linux capable device could be had quite cheaply, and development was extremely flexible – mainly being based in C, but with the ability to install other packages. The advantage of this would be the ability to not rely on closed, proprietary code, while having access to many drivers for the hardware already. The specific device chosen was a Linksys WRT54GSv1.1, which has distributions of OpenWRT router firmware available for it. OpenWRT is a version of Linux designed specifically for routers. The router needed to be modified to expose the built in serial ports to facilitate the interface between the router and the wireless communication board, used to receive data from the measurement hardware. Much time was spent debugging these serial ports, as there was a peculiar quirk with certain versions of OpenWRT whereby the serial ports were unable to receive any data, but were able to transmit fine. The device was only required to receive serial transmissions this rendered the ports useless. Fortunately the issue was found to be build specific, and by changing to a different version of OpenWRT the problem was able to be solved.

Data is collected from the serial port by a script (see appendices for full listings), which filters out any data that is not in the simple packet format using the Linux commands cat, grep and sed. The filtered packets are then passed to another SH script which handles the data queue. Each reading is added to a queue, and an index is incremented to keep track of how many samples have been collected.

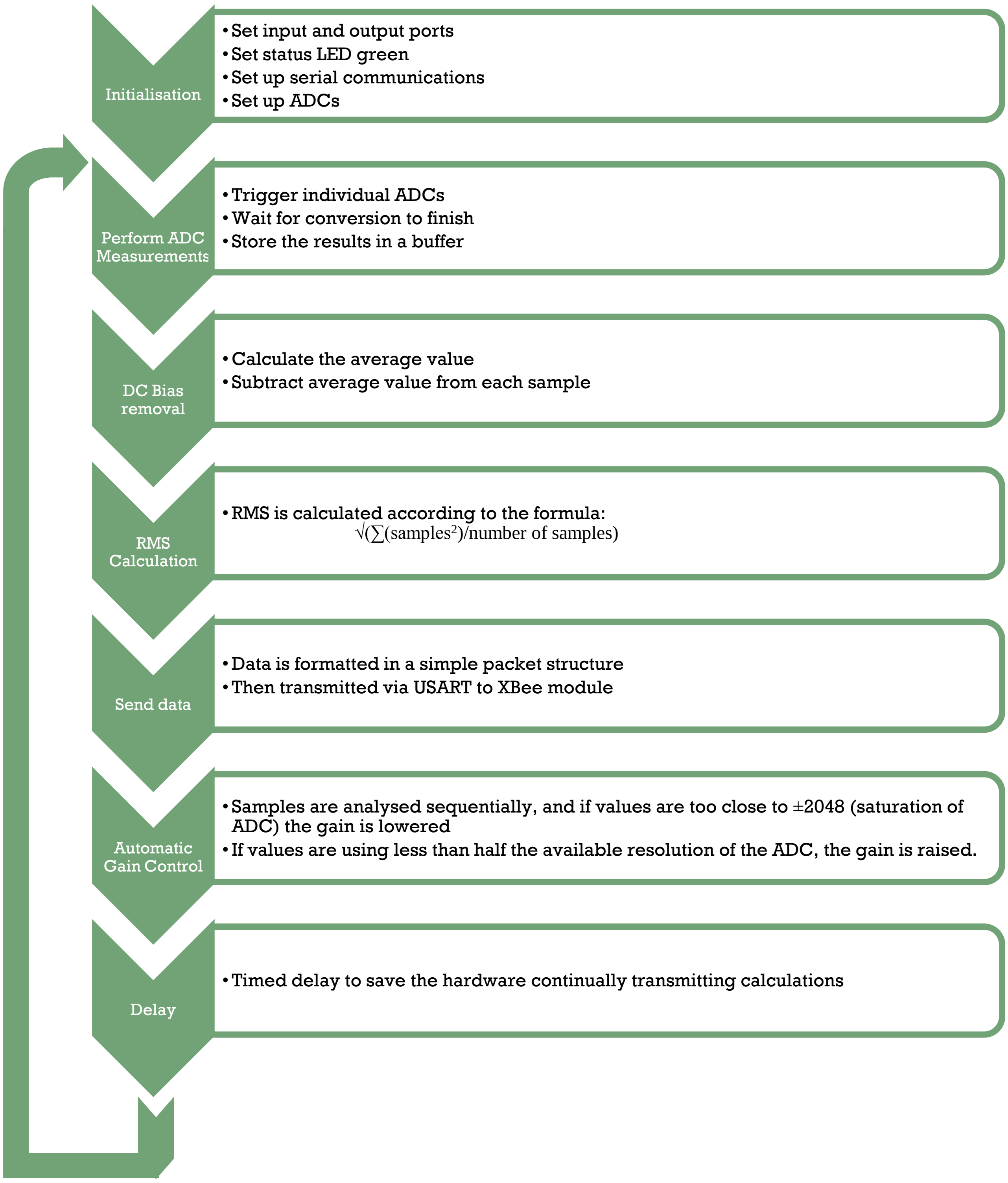
Once enough samples to warrant reporting have been collected, the queuing system invokes another script, responsible for taking the queue and feeding the data to the Google servers, clearing the queue and resetting the index. In this manner, each separate function of the system is contained in its own script for easy debugging and overall robustness.

Software

Measurement

The role of the measurement hardware is to attach to the device under measurement, detect the voltage and the current used by it, and then convert these into an instantaneous power usage. This power usage measurement is transmitted to the serial interface of the internet relay hardware wirelessly via an XBee module. The role of the software in this device will be to control the flow of data from the ADC of the microcontroller to the wireless module.

The program flows as follows:



Initialisation
- Set input and output ports
- Set status LED green
- Set up serial communications
- Set up ADCs

Perform ADC Measurements
- Trigger individual ADCs
- Wait for conversion to finish
- Store the results in a buffer

DC Bias removal
- Calculate the average value
- Subtract average value from each sample

RMS Calculation
- RMS is calculated according to the formula:
  $\sqrt{(\sum(\text{samples}^2)/\text{number of samples})}$

Send data
- Data is formatted in a simple packet structure
- Then transmitted via USART to XBee module

Automatic Gain Control
- Samples are analysed sequentially, and if values are too close to ±2048 (saturation of ADC) the gain is lowered
- If values are using less than half the available resolution of the ADC, the gain is raised.

Delay
- Timed delay to save the hardware continually transmitting calculations

Current and voltage measurements are made at roughly 12 kHz (ADC is clocked at 125 kHz, but there is some overhead while the data is stored, and it takes multiple clocks for the measured values to settle). The stored data is then processed by the subsequent functions of the hardware. To minimise power consumption during this stage, a transistor has been placed in the voltage measuring circuit which enables and disables a voltage divider dynamically, meaning that when a measurement is not being taken there is no parasitic current draw from the voltage divider used to drop the voltage to within the reference voltage.

The first alteration of the measured data is to remove any DC offset that is present. Although filter capacitors have been placed in series with the inputs, there is still significant DC offset measured by the ADC. If this were left in the samples, the RMS calculation would be wildly inaccurate. The DC bias is removed in software by calculating an average of the samples, then subtracting this average from each sample one by one.

The root means square (RMS) of the waveform is then calculated with the formula

$$RMS = \sqrt{\frac{\sum_i sample(i)^2}{number\ of\ samples}}$$

This is somewhat computationally expensive for the device to calculate as there is no built in floating point processor on the ATXMega, and thus must be performed with integer arithmetic. Fortunately the device is clocked sufficiently fast, and is able to compute the RMS in a timely manner. By using a true RMS calculation rather than the common $V_p/\sqrt{2}$ approximation we are able to accurately measure the power on non-sinusoidally varying waveforms, such as switchmode power supplies, which are becoming increasingly common in electronics. This RMS calculation is done for both the voltage and current measurements.

Once the RMS has been calculated, the voltage and current values are multiplied together and then transmitted wirelessly via the XBee module to the Internet relay hardware. This is achieved by placing the value in a simple packet string, “rd:125.51” for example, indicating 125.51mW of power being consumed.

Finally, the samples are analysed to check for cases where the gain of the amplifier on the ADC should be changed. If samples with a magnitude of close to 2048 are found it is likely that the ADC is saturated and is clipping the samples. In this case the gain of the amplifier is reduced by a factor of two. However, in the opposite case, where the gain needs to be increased the automatic gain correction algorithm detects if less than half of the resolution of the ADC is being used (and thus if the gain is doubled, no clipping will occur) the gain is increased by a factor of two. This mechanism allows for the maximum resolution of the ADC to be in use at all times.

Internet Relay

There are three main functions of the internet relay:

1. Accept wireless communications from the measurement hardware
2. Store up to 24 hours worth of measurements in the case of internet access being unavailable for a day.
3. Report measurements to the Google powermeter API, and handle all secure communications

In order to wirelessly communicate with the measurement hardware, the router has been outfitted with an XBee module, connected to the device's serial port. When data is received it is able to be accessed via the special device blocks available to the Linux operating system. A program checks this for any received measurements, and stores these measurements in a queue, waiting to be transmitted to Google. While the internet is functioning normally, this queue is 60 records long as the communications to Google can only occur at a maximum of once per 10 minutes. However, in the case that the internet is down up to 8640 records must be stored.

The Google powermeter API is specific about what features are necessary to implement, and even goes into detail about how they are to be implemented. For example, it specifies that 24 hours of records must be kept, and that when 24 hours worth of samples is reached that the oldest data is overwritten with the newest data, so that the device keeps data for the most recent 24 hours. In order to achieve this, a helper script manages the flow of data, accepting new data and discarding old data as it is replaced.

When the time comes to transmit the measurements to Google (once every 10 minutes) the entire queue is sent and then cleared from the local memory once confirmation has been received.

While using the Google powermeter API, the client (internet relay hardware) must have authenticated with the Google servers to allow secure transmission of the measurement data. This must be achieved by a web interface (as specified by Google[4]). This web interface allows users to "activate" their power meter by signing into their Google account. This interface then saves a security hash token specific to the user, allowing secure communication between Google and the device, while never actually storing any of the user's data. This hash is then used to encrypt all communication allowing for completely secure transmission of data.

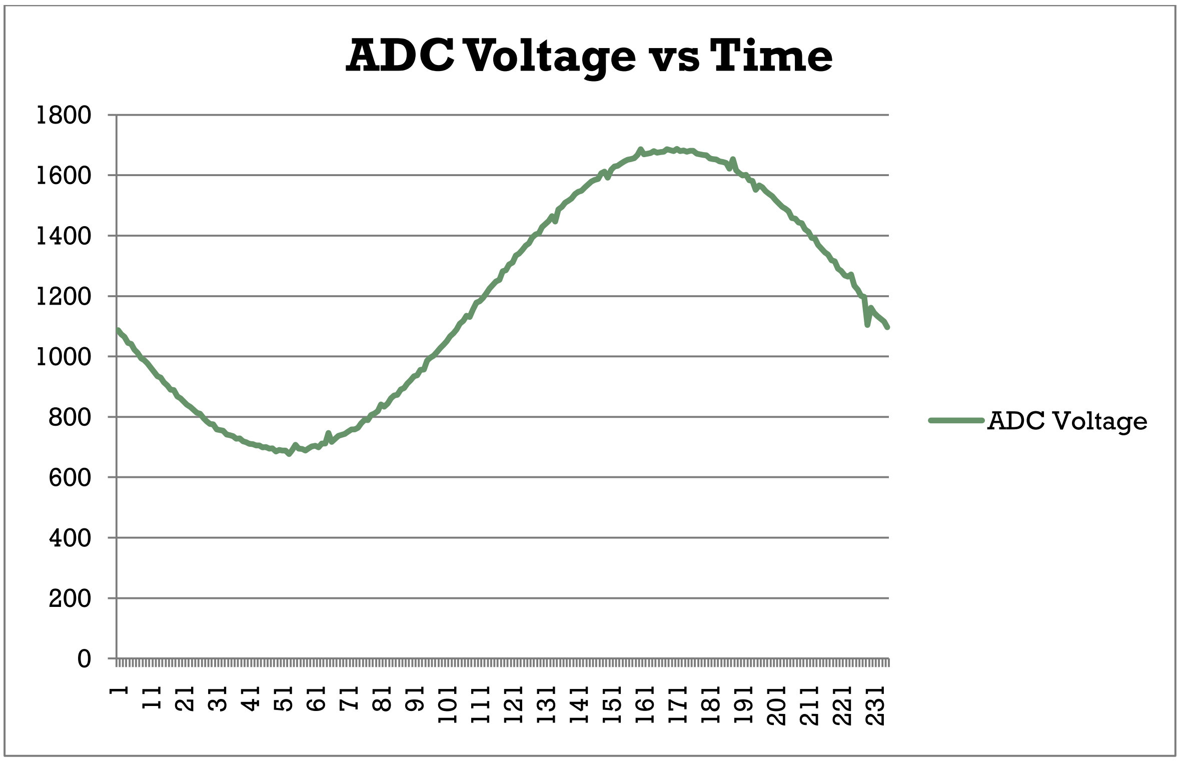
The web interface also allows the user to view their power usage, update their account details, as well as edit the network settings of the router.

[4] http://code.google.com/apis/powermeter/docs/powermeter_device_activation.html

Results

Measuring Hardware

The measuring hardware has been designed to be somewhat transparent once set up in the final system, communicating only with the gateway. There is no configuration necessary, or indeed possible. In the final version of the hardware, the only output is the wireless signal, sending a serial string, for example "rd:512.21" to represent 512.21mW of power being used.



As can be seen by the above graph, the voltage measurements are relatively stable, providing good accuracy, with only minor fluctuations from a pure sine wave. The largest of these deviations is ~80mV, and when averaged over the entire waveform become a very minor contributor to error, less than 1%.

However, it can be seen clearly that the AC signal has a ~1.1V DC offset, which is highly undesirable. DC offsets were a problem that was designed around, with input series capacitors being placed in the circuit in an attempt to mitigate the effects. It appears that these capacitors were insufficient to remove the offset and as a result it needed to be compensated for in software by calculating the average value of the waveform and subtracting this average from each sample. This allows for the RMS to be calculated accurately, but is only a stop-gap solution to the problem, as in worst case scenarios where the voltage offset is large; it could lead to clipping of the waveform and thus produce inaccurate RMS calculations. In order to mitigate the potential for this occurring, a system for automatic gain control on the differential amplifiers has been put in place, as described in the method section.

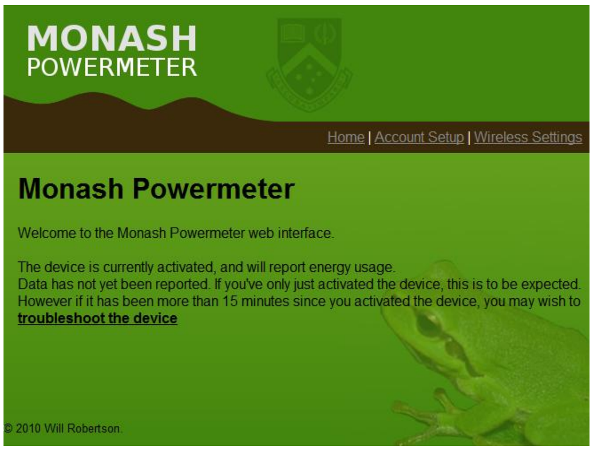
Gateway

Web interface

The web interface is the end user's sole interaction with the system for configuration and setup. It performs two main functions:

- Network configuration
- Device activation

On the network configuration page, the web interface allows the user to configure the wireless by entering the SSID, encryption type and security key while they are connected to the wired interface. This is achieved by the use of a PHP script which can write the necessary configuration to the */etc/config/network* and */etc/config/wireless* files on the router's filesystem. A troubleshooting page has also been included which checks for internet connectivity by testing a connection to www.google.com and reports to the user if this was successful or not.



Monash Powermeter

Welcome to the Monash Powermeter web interface.

The device is currently activated, and will report energy usage.
Data has not yet been reported. If you've only just activated the device, this is to be expected.
However if it has been more than 15 minutes since you activated the device, you may wish to **troubleshoot the device**

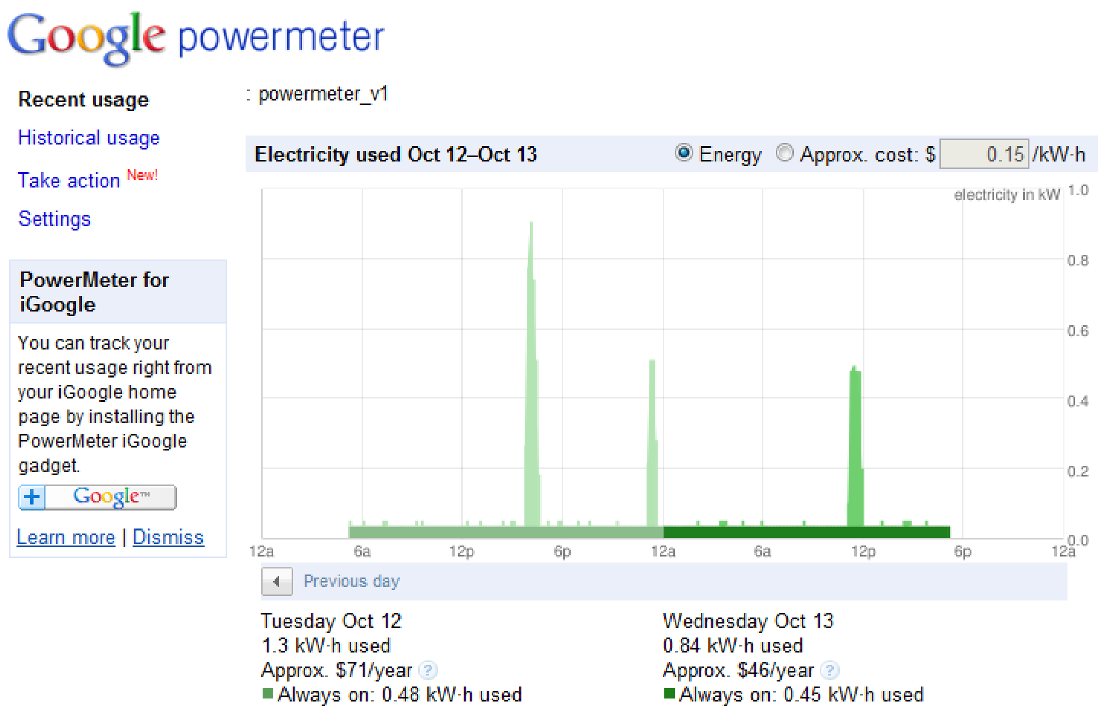
© 2010 Will Robertson.

Figure 1: One of the screens of the web interface

The second function of the web interface is to perform the device activation. The device can detect if it is currently activated, and if it is not, displays a button to provide users a simple way to begin the process. When the button is clicked they are redirected to a Google sign in page where their details are taken and the gateway is associated with the account. From this point, a hash of the user's details (no sign in information is stored on the device for security) is stored, and the device is able to begin reporting of power data.

Internet

Once the data is sent off by the gateway, all of the heavy lifting of the data analysis is handled by Google. For a simple demonstration of the capabilities of this system some data has been generated and submitted. The data can be seen below:



PowerMeter for iGoogle

You can track your recent usage right from your iGoogle home page by installing the PowerMeter iGoogle gadget.

Tuesday Oct 12
1.3 kW·h used
Approx. $71/year
🟩 Always on: 0.48 kW·h used

Wednesday Oct 13
0.84 kW·h used
Approx. $46/year
🟩 Always on: 0.45 kW·h used

As can be seen, a base “always on” load is highlighted, and any power on top of this is drawn in a lighter colour. In addition to this various usage statistics have been included, such as total power usage, estimated costs and the ability to view historical data.

Conclusion

At the conclusion of this semester I have been successful in implementing the goals set forth in the requirements analysis. These are be listed below and will be described with respect to the level of completion achieved.

Must measure AC current

By the use of the current clamp and differential ADC in the microprocessor, the system is fully able to measure AC current.

Must transmit this current measurement to a ‘base station’ wirelessly

Measurements are transmitted via the XBee modules using the IEEE 802.15.4 protocol wirelessly and received by the gateway unit.

Must relay electrical power usage information to the Google Powermeter API

The gateway unit connects to the internet and is able to send power data to Google via this API, once the device has been associated with a Google account.

Current measuring device must be low power

This non-functional requirement has been further specified as a measurable goal – the device is to consume less than 150mW of power. When supplied by 5V, the device consumes 63mA for a total power consumption of 315mW. Unfortunately this does not meet the design requirements – however 51.5mA of the 63mW is due to the XBee. This alone consumes 257.5mW of power. This power could have been saved by the inclusion of a MOSFET or transistor to disable the XBee while not in use, and further power could have been saved by utilising the sleep mode of the microcontroller.

Must be able to measure a variety of currents

The device must be able to measure currents from 0A to at least 10A. The current clamp I have selected for use is capable of measuring currents between 0 and 30A, and in conjunction with the variable gain on the ADC, the entire range of currents is able to be measured.

Must be as accurate as possible

An accuracy of 5% was specified by the requirements analysis. As demonstrated in the results section of the report, the ADC does have minor errors where individual samples can have errors of up to 6%. However, these occur very infrequently (less than 10 of the total 235 samples have a significant deviation), and thus when the RMS is calculated these errors become insignificant, less than 1% of the total power.

Must not directly come into contact with 240V AC mains

Through the use of the current clamp, non-invasive measurements of current are able to be taken. However the device has since been re-targeted at a 12V AC application and is able to measure the voltage safely by the use of a voltage divider.

Further work

Several options for further work could be considered. Exploration of power saving methods to minimise consumption of device is the obvious choice, as unfortunately that aspect of the design requirement was unable to be fulfilled.

Integration of the power supply of measuring device into connection to AC mains could also be considered, however this violates one of the design requirements of this project in that it would need direct contact to 240V AC mains. One benefit of this could be the ability of the measuring hardware to turn the device on or off under certain conditions – malfunctioning devices could be shut down before damage is done, or power hungry devices could be rationed based on their energy usage.

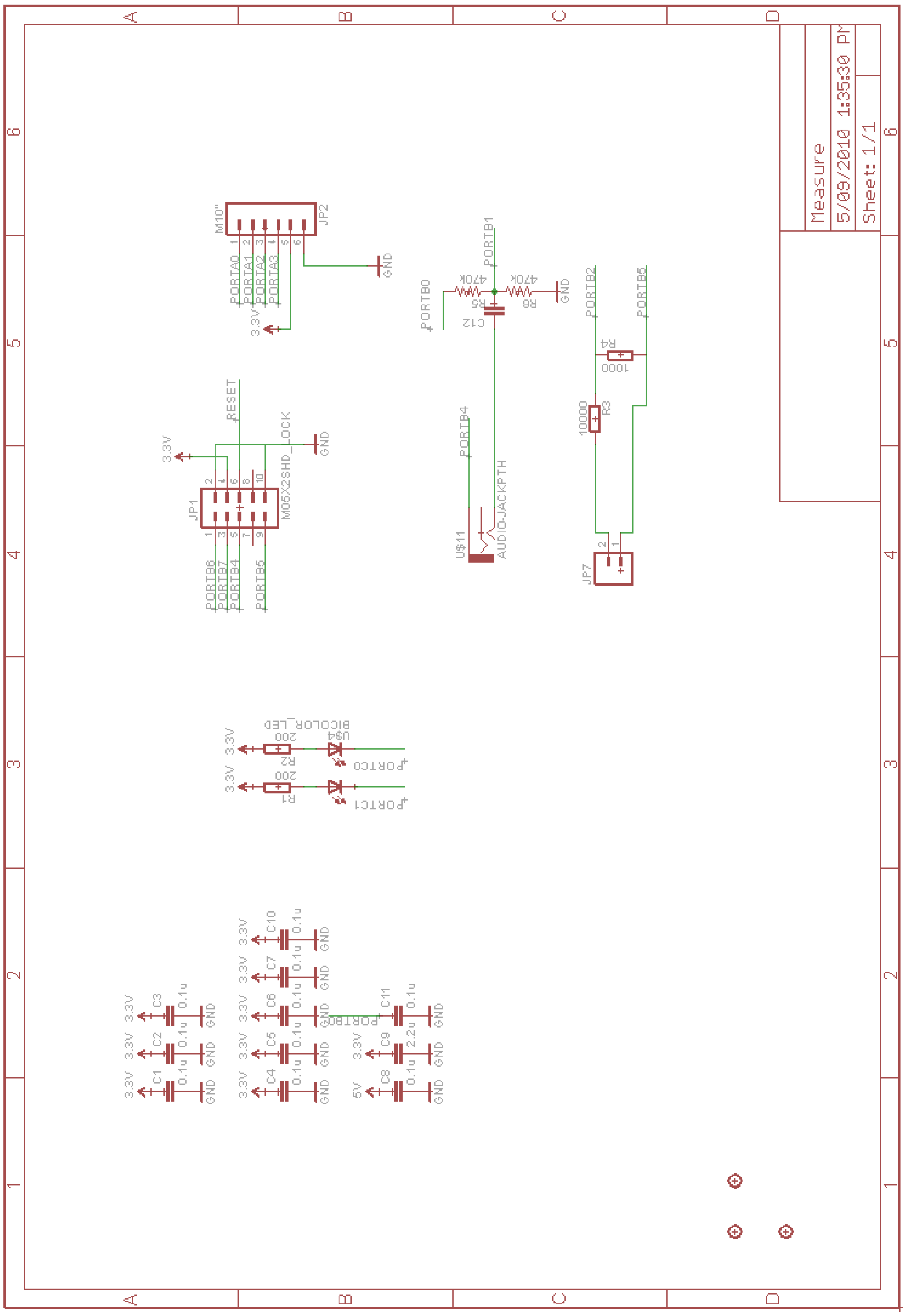
The ability to connect multiple measuring devices to a single gateway could also be a desirable trait, where different sections of a house could be measured, or in fact individual devices could be measured.

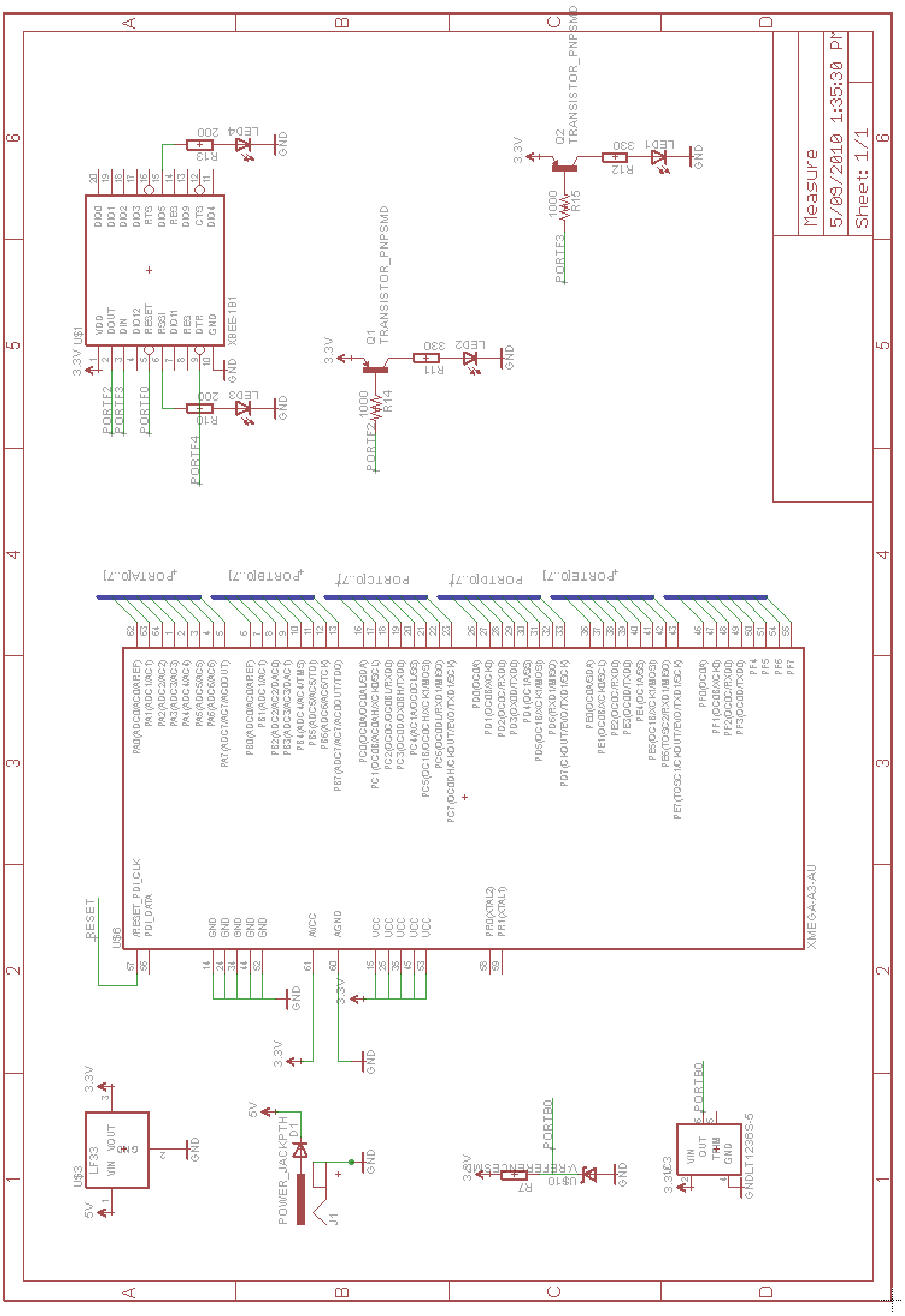
Device profiling could also be performed based on energy signatures put forward by the device. For example it may be possible to identify something like a fridge compressor turning on and if it turns on frequently it may indicate an inefficient device.

Appendices

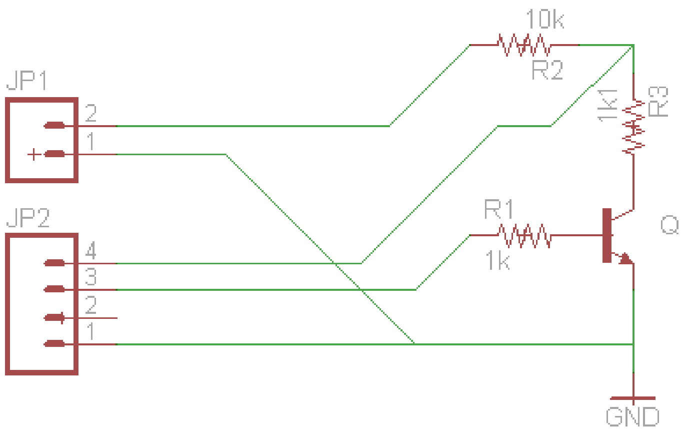
Schematic Diagrams

Measurement hardware

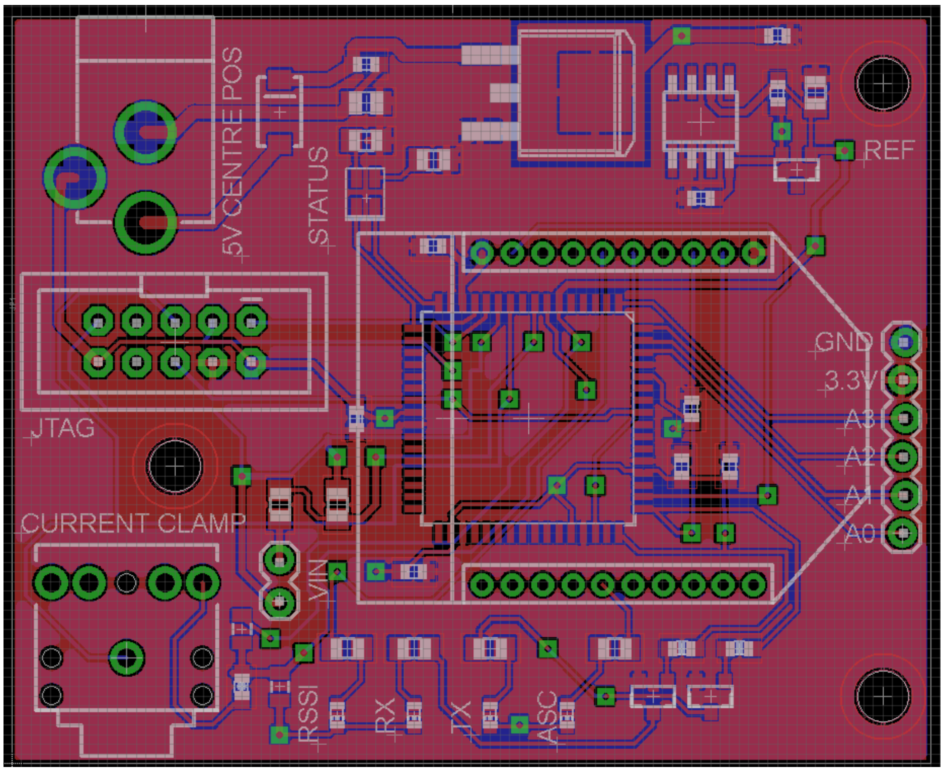



Measure
5/09/2010 1:35:30 PM
Sheet: 1/1
XMEGA-A3-AU
XBEE1B1
U$1
U$6
U$3
LF33
U$10
LT1236S-5
POWER_JACKPTH
J1
D1
Q1
Q2
TRANSISTOR_PNPSMD
LED1
LED2
LED3
LED4
R7
R10
R11
R12
R13
R14
R15
1000
330
200
3.3V
5V
GND
RESET
VREFERENCESMD
PORTA[0..7]
PORTB[0..7]
PORTC[0..7]
PORTD[0..7]
PORTE[0..7]
PORTF2
PORTF3
PORTF0
PORTF4
PORTB0

Voltage measurement circuit



Printed Circuit Board design



Source Code

Measuring board

```
#define F_CPU 2000000UL
#include <avr/io.h>
#include <stdio.h>
#include <math.h>
#include <util/delay.h>

#define NUM_SAMPLES 235  // Number of samples required to capture
exactly one cycle of 50Hz

int16_t buffer[NUM_SAMPLES];
int16_t voltage[NUM_SAMPLES];
```

```
char serialstring[20];

enum statuses {
      RED,
      GREEN,
      ORANGE
};

void set_status(enum statuses status) {
      switch(status) {
            case RED:
                  PORTC.OUT = 0x01;
                  break;
            case GREEN:
                  PORTC.OUT = 0x02;
                  break;
            case ORANGE:
                  PORTC.OUT = 0x00;
                  break;
      }
}

void usart_write(unsigned char data) {
    USARTF0.DATA = data;
      if(!(USARTF0.STATUS&USART_DREIF_bm))
            while(!(USARTF0.STATUS & USART_TXCIF_bm)); // wait for TX
complete
                  USARTF0.STATUS |= USART_TXCIF_bm;  // clear TX
interrupt flag
};

void send_string(char* string, int len) {
      int i = 0;
      while(string[i] != '\0' && i<len) {
            usart_write(string[i]);
            i++;
      }
};

void set_clock_32m(void) {
  CCP = CCP_IOREG_gc;
  OSC.CTRL = OSC_RC32MEN_bm;
  while(!(OSC.STATUS & OSC_RC32MRDY_bm));
  CCP = CCP_IOREG_gc;
  CLK.CTRL = 0x01;
};

int abs (int i) {
   return i < 0 ? -i : i;
}

int main() {
      int i;
      int gain = 1;
      int gaincontrol = 0;
```

```
	/* INIT SECTION */
	PORTC.DIR = 0x03; // configure PORTC[1:0] as output
	set_status(GREEN);

	/* SET UP USART */
	PORTF.DIR |= (1<<3); // set TX pin as output
	PORTF.OUT |= (1<<3); // set PORTF[3] high
	USARTF0.BAUDCTRLA = 11&0xFF;
	USARTF0.BAUDCTRLB = (-7<<4) | (11>>8); // 115.1kbps (0.5%
error)
	USARTF0.CTRLB = USART_TXEN_bm | USART_RXEN_bm; // enable tx
and rx on USART

	/* SET UP ADC */
	ADCB.CTRLA |= 0x1; // enable adc
	PORTB.DIR = 0x00;
	ADCB.CTRLB = 0x10 | ADC_RESOLUTION_12BIT_gc; // 12 bit signed
conversion (pos 11bits)
	ADCB.REFCTRL = ADC_REFSEL_AREFB_gc; // use external reference
	ADCB.PRESCALER = ADC_PRESCALER_DIV16_gc;
	ADCB.CH0.CTRL = ADC_CH_INPUTMODE_DIFFWGAIN_gc |
ADC_CH_GAIN_1X_gc;
	ADCB.CH0.MUXCTRL = ADC_CH_MUXPOS_PIN1_gc |
ADC_CH_MUXNEG_PIN4_gc;
	ADCA.CTRLA |= 0x1; // enable adc
	PORTA.DIR = 0x08;
	ADCA.CTRLB = 0x10 | ADC_RESOLUTION_12BIT_gc; // 12 bit signed
conversion (pos 11bits)
	ADCA.REFCTRL = ADC_REFSEL_AREFB_gc; // use external reference
	ADCA.PRESCALER = ADC_PRESCALER_DIV16_gc;
	ADCA.CH0.CTRL = ADC_CH_INPUTMODE_DIFF_gc;
	ADCA.CH0.MUXCTRL = ADC_CH_MUXPOS_PIN2_gc |
ADC_CH_MUXNEG_PIN1_gc;

	/* MAIN LOOP */
	while(1) {
		double average = 0.0;

		/* FILL BUFFER WITH ADC DATA */
		for(i=0; i<NUM_SAMPLES; i++) {
			PORTA.OUT = 0x08; // Enable voltage divider on input
			ADCB.CH0.INTFLAGS |= 1; // Clear interrupt flag
			ADCA.CH0.INTFLAGS |= 1; // Clear interrupt flag
			ADCB.CTRLA |= (1<<2); // Start single conversion
			ADCA.CTRLA |= (1<<2); // Start single conversion
			while(!ADCB.CH0.INTFLAGS);
			buffer[i] = ADCB.CH0RES;
			while(!ADCA.CH0.INTFLAGS);
			buffer[i] = ADCA.CH0RES;
			PORTA.OUT = 0x00; // Disable voltage divider on
input
		}

		for(i=0;i<NUM_SAMPLES;i++){
```

```
                if(abs(buffer[i]) > 2040) {
                        // ADC has been saturated
                        gaincontrol -= 1;
                }
                average += buffer[i];
                snprintf(serialstring, sizeof(serialstring), "%d,",
buffer[i]);
                send_string(serialstring, sizeof(serialstring)); //
write string
        }

        average /= NUM_SAMPLES;

        /* PERFORM RMS CALCULATION */
        double accumulator = 0.0;
        double voltage_acc = 0.0;
        for(i=0; i<NUM_SAMPLES; i++) {
                accumulator += pow(buffer[i]-average,2);
                voltage_acc += pow(voltage[i],2);
        }
        accumulator = sqrt(accumulator/NUM_SAMPLES);
        voltage_acc = sqrt(voltage_acc/NUM_SAMPLES);
        voltage_acc /= 1000; // Convert to voltage

        if(M_SQRT2*accumulator+abs(average) < 1000) {
                // Less than half of the resolution is being used
                gaincontrol += 1;
        }

        /* SEND VALUE */
        snprintf(serialstring, sizeof(serialstring), "rd:%.2f\n",
voltage_acc*accumulator/gain);
        send_string(serialstring, sizeof(serialstring)); // write
string

        /* PERFORM GAIN CONTROL, IF NECESSARY */
        if(gaincontrol) {
                snprintf(serialstring, sizeof(serialstring), "g =
%d, gc = %d\n", gain, gaincontrol);
                send_string(serialstring, sizeof(serialstring)); //
write string

                switch(gain) {
                        case 1:
                                if(gaincontrol>0) {
                                        gain = 2;
                                        ADCB.CH0.CTRL =
ADC_CH_INPUTMODE_DIFFWGAIN_gc | ADC_CH_GAIN_2X_gc;
                                }
                                break;
                        case 2:
                                if(gaincontrol>0) {
                                        gain = 4;
                                        ADCB.CH0.CTRL =
ADC_CH_INPUTMODE_DIFFWGAIN_gc | ADC_CH_GAIN_4X_gc;
```

```
                           } else {
                                gain = 1;
                                ADCB.CH0.CTRL = 
ADC_CH_INPUTMODE_DIFFWGAIN_gc | ADC_CH_GAIN_1X_gc;
                           }
                           break;
                      case 4:
                           if(gaincontrol>0) {
                                gain = 8;
                                ADCB.CH0.CTRL = 
ADC_CH_INPUTMODE_DIFFWGAIN_gc | ADC_CH_GAIN_8X_gc;
                           } else {
                                gain = 2;
                                ADCB.CH0.CTRL = 
ADC_CH_INPUTMODE_DIFFWGAIN_gc | ADC_CH_GAIN_2X_gc;
                           }
                           break;
                      case 8:
                           if(gaincontrol>0) {
                                gain = 16;
                                ADCB.CH0.CTRL = 
ADC_CH_INPUTMODE_DIFFWGAIN_gc | ADC_CH_GAIN_16X_gc;
                           } else {
                                gain = 4;
                                ADCB.CH0.CTRL = 
ADC_CH_INPUTMODE_DIFFWGAIN_gc | ADC_CH_GAIN_4X_gc;
                           }
                           break;
                      case 16:
                           if(gaincontrol>0) {
                                gain = 32;
                                ADCB.CH0.CTRL = 
ADC_CH_INPUTMODE_DIFFWGAIN_gc | ADC_CH_GAIN_32X_gc;
                           } else {
                                gain = 8;
                                ADCB.CH0.CTRL = 
ADC_CH_INPUTMODE_DIFFWGAIN_gc | ADC_CH_GAIN_8X_gc;
                           }
                           break;
                      case 32:
                           if(gaincontrol>0) {
                                gain = 64;
                                ADCB.CH0.CTRL = 
ADC_CH_INPUTMODE_DIFFWGAIN_gc | ADC_CH_GAIN_64X_gc;
                           } else {
                                gain = 16;
                                ADCB.CH0.CTRL = 
ADC_CH_INPUTMODE_DIFFWGAIN_gc | ADC_CH_GAIN_16X_gc;
                           }
                           break;
                      case 64:
                           if(gaincontrol<0) {
                                gain = 32;
                                ADCB.CH0.CTRL = 
ADC_CH_INPUTMODE_DIFFWGAIN_gc | ADC_CH_GAIN_32X_gc;
```

```
                            }
                            break;
                    }
                    gaincontrol = 0;
            }
            _delay_ms(10000);
    }

}
```

Queue handling scripts

Readserial.sh – Responsible for reading and filtering serial data

```
#!/bin/sh
{
        while read line; do
                line_filtered=`echo -e "$line\n" | grep "^rd:[0-9]"
| sed "s_rd:__"`
                size=`echo $line_filtered | wc -c`
                if test $size -gt 1; then
                        echo "`date '+%Y-%M-%d %H:%M:%S'`
./adddata.sh $line_filtered"
                        ./adddata.sh $line_filtered
                fi
        done
} < /dev/tts/0
```

Adddata.sh – responsible for queuing reported variables

```
#!/bin/sh

if ! test -d ~/powerdata
then
    mkdir ~/powerdata
fi

if test -f ~/powerdata/index
then
    INDEX=`cat ~/powerdata/index`
else
    echo "No index found, creating one."
    INDEX=1
    echo $INDEX > ~/powerdata/index
fi

echo -e "`date "+%s"`\t$1" >> ~/powerdata/queue.txt

if test $INDEX -eq 60
then
    #sh ~/report.sh
    echo "Index = 60, reporting and reseting index"
    echo 1 > ~/powerdata/index
else
    echo "Index != 60, bumping index"
    echo `expr $INDEX + 1` > ~/powerdata/index
```

```
fi
```

Report.sh – responsible for calling the Google Powermeter API

```
#!/bin/sh

TOKEN=`cat ~/token`
VARIABLE=`cat ~/variable`".c1"

python ~/google-powermeter-api/post_readings_devices.py $TOKEN
$VARIABLE --split_on_tab --time_column=1 --reading_column=2 <
~/powerdata/queue.txt

rm ~/powerdata/queue.txt
```

Web interface

Account.php

```
<?php

include('header.php');

echo("    <h1>Account Setup</h1>\n");
if($loggedin) {
  echo("    Your device is currently activated. If you would like to
de-activate the device, please click the button below.<br>\n");
  echo("    <form action=\"deactivate.php\">\n");
  echo("      <input type=\"submit\" value=\"De-activate
device\">\n");
  echo("    </form>\n");
} else {
  $snonce = rand(0,255);
  $fd = fopen('/tmp/snonce', 'w');
  fwrite($fd, $snonce);
  fclose($fd);

  $rurl = "http://" . "192.168.0.50" . "/rurl.php";

  echo("    This device has not yet been activated - please click
the button to begin the activation process. <br>\n");
  echo("    <form
action=\"https://www.google.com/powermeter/device/activate\"
method=\"get\">\n");
  echo("      <input type=\"hidden\" name=\"mfg\"
value=\"monash\">\n");
  echo("      <input type=\"hidden\" name=\"model\"
value=\"powermeter_v1\">\n");
  echo("      <input type=\"hidden\" name=\"did\" value=\"1\">\n");
  echo("      <input type=\"hidden\" name=\"cvars\"
value=\"1\">\n");
  echo("      <input type=\"hidden\" name=\"snonce\"
value=\"$snonce\">\n");
  echo("      <input type=\"hidden\" name=\"rurl\"
value=\"$rurl\">\n");
```

```
  echo("      <input type=\"submit\" value=\"Activate Device\">\n");
  echo("    </form>\n");
}

echo("    <a 
href=\"http://www.google.com/powermeter/privacy.html\">Google 
PowerMeter Privacy Policy</a><br>\n");

include('footer.php');

?>
```

Deactivate.php

```
<?php

include('header.php');

if(isset($_POST[doit])) {
  $doit = $_POST[doit];
  if(strcmp($doit,"no")) {
    unlink('/root/token');
    echo("    <h1>De-activate your device</h1>\n");
    echo("    Your device is now de-activated. Should you wish to 
being reporting power usage data again, simply re-activate the 
device.<br>\n");
  }
} else {
  echo("    <h1>De-activate your device</h1>\n");
  echo("    Are you sure you want to de-activate your device? It 
will no longer record any power usage data until you re-
activate.<br><br>\n");
  echo("    <form action=\"$_SERVER[PHP_SELF]\" 
method=\"post\">\n");
  echo("      <input type=\"hidden\" name=\"doit\" 
value=\"yes\">\n");
  echo("      <input type=\"submit\" value=\"Yes, de-activate my 
device\">\n");
  echo("    </form>\n");
  echo("    <form action=\"/cgi-bin/index.php\">\n");
  echo("      <input type=\"submit\" value=\"No, leave my device as 
it is\">\n");
  echo("    </form>\n");
}

include('footer.php');

?>
```

Footer.php

```
<?php
```

```
echo("  </div>
  <p class=\"footer\">&copy; 2010 Will Robertson.</p>
  <div class=\"logo\">&nbsp;</div>
  </body>
</html>");

?>
```

Header.php

```
<?php

$loggedin = file_exists('/root/token');
if($loggedin) {
  $token = file_get_contents('/root/token', false);
}

echo("<html>
  <head>
    <title>Monash Powermeter Web UI</title>
    <link rel=\"stylesheet\" href=\"/resources/style.css\"
type=\"text/css\" media=\"screen\">
  </head>
  <body>
  <div id=\"heading\">
  </div>
  <div id=\"menu\">
    <a href=\"index.php\">Home</a> | <a href=\"account.php\">Account
Setup</a> </div>
  <div id=\"content\">
");

?>
```

Index.php

```
<?php

include('header.php');

echo("    <h1>Monash Powermeter</h1>");
echo("    <p>Welcome to the Monash Powermeter web
interface.<br><br>\n");

if($loggedin) {
  echo("    The device is currently activated, and will report
energy usage.<br>\n");
  if(file_exists('/root/lasttime')) {
    echo("    Last communication of data: ");
    echo(file_get_contents('/root/lasttime'));
  } else {
    echo("Data has not yet been reported. If you've only just
activated the device, this is to be expected.<br>\n");
```

```
    echo("However if it has been more than 15 minutes since you
activated the device, you may wish to <a
href=\"troubleshoot.php\">troubleshoot the device</a><br>\n");
  }
} else {
  echo("    You are not currently logged in. <a
href=\"account.php\">Click here</a> to get started!</p>\n");
}

include('footer.php');

?>
```

Rurl.php

```
<?php

include("header.php");

echo("    <h1>Device activation</h1>\n");

if(file_exists('/tmp/snonce')) {
  $snonce = file_get_contents('/tmp/snonce');

  if($snonce == $_REQUEST[snonce]) {
    $token = $_REQUEST[token];
    $variable = $_REQUEST[path];

    $fd = fopen('/root/token', w);
    fwrite($fd, $token);
    fclose($fd);

    $fd = fopen('/root/variable', w);
    fwrite($fd, $variable);
    fclose($fd);

    echo("    <b>Congratulations!</b><br>\n");
    echo("    Your device has been successfully activated, and can
now begin reporting power usage information to your Google
account.<br><br>\n");
    echo("    For some great energy saving tips to help you lower
the cost of your bill and your impact on the environment, visit the
Save Energy website by clicking <a
href=\"http://saveenergy.vic.gov.au\">here</a>.<br>\n");
  } else {
    echo("    Secure nonce response does not match sent value --
potential third party or replay attack! Aborting.<br>\n");
  }
} else {
  echo("    Secure nonce file not present -- security of connection
impossible to determine; aborting.<br>\n");
}

include("footer.php");
```

```
?>
```

Troubleshoot.php

```
<?php

include('header.php');

echo("    <h1>Troubleshooting</h1>\n");

$connection = fopen("http://www.google.com", 'r');
if($connection) {
    $connected = true;
    fclose($connection);
} else {
    $connected = false;
}

echo("    <b>Internet connection</b><br>\n");

if($connected) {
  echo("    Connection to the internet is functional.<br>\n");
} else {
  echo("    Unable to contact google.com -- internet connection may
be unavailable. ");
}

include('footer.php');

?>
```