Monitoring Voltage and Current at various nodes in a Smart Grid

Internship report submitted to the

National University of Singapore, Singapore

for the completion of summer internship

by

Subhajit Sahu (Roll No: 110EC0181)

B.Tech., Electronics and Communication Engineering

National Institute of Technology, Rourkela

Odisha, India. PIN-769008

Under the guidance of

Prof. Sanjib Kumar Panda

Electronics and Computer Engineering

Faculty of Engineering

National University of Singapore, Singapore

Acknowledgements

It has been a great experience to work under the esteemed supervision of Prof. Sanjib Kumar Panda. I am very much privileged to have him as my mentor. I would like to thank him from the bottom of my heart for his involvement, guidance, most importantly his support and encouragement throughout the project work. In the same breath, I would also like to thank Hoang Duc Chinh, Bhuneswar Prasad, Krishnanand KR, for their guidance and support during the project work. I thank all members of EMD Lab, who have been a great source of support for my stay at NUS and Singapore.

My sincere thanks to all group fellow members for their knowledge sharing and filling enthusiasm in me to make my project work enjoyable. It is my immense pleasure to thank my parents and family members for their constant moral support and inspiration.

Abstract

Electricity is the most versatile and widely used form of energy and global demand is growing continuously. Generation of electrical energy, however, is currently the largest single source of carbon dioxide emissions, making a significant contribution to climate change. To mitigate the consequences of climate change, the current electrical system needs to undergo significant adjustments.

The electrical power system was built up over more than 100 years. It is now one of the most effective components of the infrastructure on which modern society depends. It delivers electrical energy to industry, commercial and residential consumers, meeting ever-growing demand.

Most of today's generation capacity relies on fossil fuels and contributes significantly to the increase of carbon dioxide in the world's atmosphere, with negative consequences for the climate and society in general.

To satisfy both the increasing demand for power and the need to reduce carbon dioxide emissions, we need an electric system that can handle these challenges in a sustainable, reliable and economic way.

Smart grids will provide more electricity to meet rising demand, increase reliability and quality of power supplies, increase energy efficiency, be able to integrate low carbon energy sources into power networks.

Smart grids possess demand response capacity to help balance electrical consumption with supply, as well as the potential to integrate new technologies to enable energy storage devices and the large-scale use of electric vehicles.

Electrical systems will undergo a major evolution, improving reliability and reducing electrical losses, capital expenditures and maintenance costs. A smarter grid will provide greater control over energy costs and a more reliable energy supply for consumers. Environmental benefits of a smarter grid include reduced peak demand, integration of more renewable power sources, and reduced CO_2 emissions and other pollutants.

This project work concentrates on the monitoring of voltage and current signals at multiple nodes in a smart grid system. This monitoring was done using ADC, XBee, and a PC using a program written Processing and MATLAB. The pages that follow, describes the journey from a simple wireless transmission of data from PC to PC; to the monitoring of voltage and current at multiple nodes on a PC.

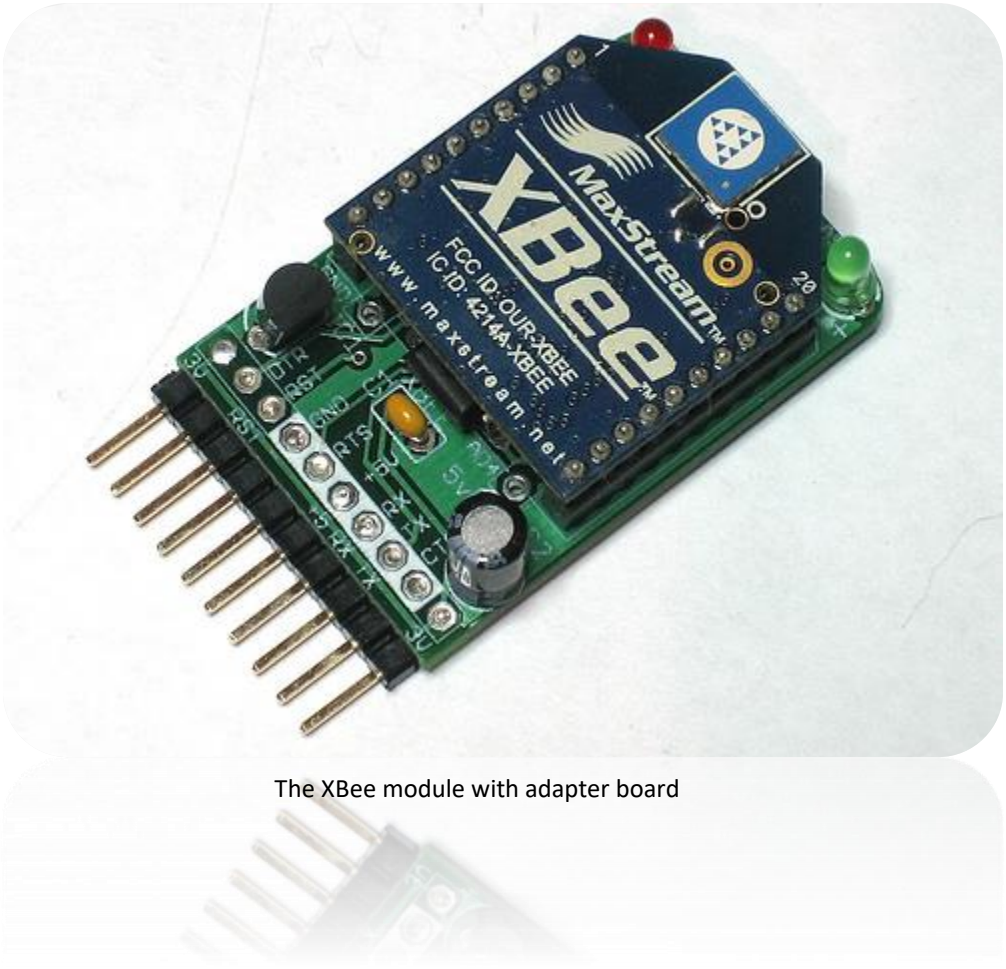
Contents

Contents

1. AN INTRODUCTION TO XBEE

XBee and ***XBee-PRO* 802.15.4** *OEM RF modules* are embedded solutions providing wireless end-point connectivity to devices. These modules use the ***IEEE 802.15.4*** networking protocol for fast point-to-multipoint or peer-to-peer networking. They are designed for high-throughput applications requiring low latency and predictable communication timing.

These modules are readily available and can be integrated into any project to add wireless communication capability to the system. Used throughout the industry, these devices help in building wireless sensor networks. Sensors transmit real-time data which are sent to XBee based gateways which save the requisite information onto a database on the internet.



The XBee module with adapter board

2. CONFIGURING AND TESTING XBEE MODULES

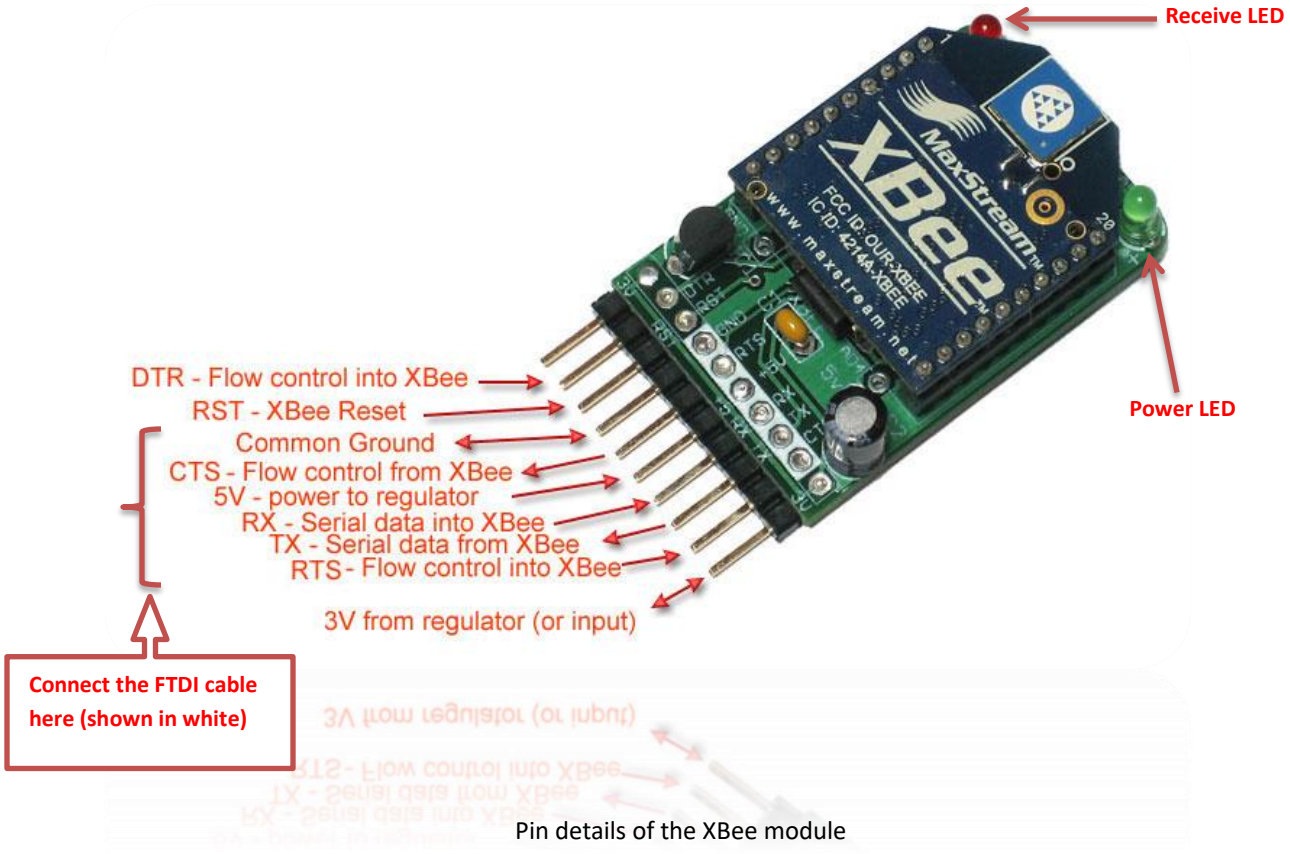
To test an XBee module we require *at least two* XBee modules (with an *adapter board*). The figure below shows the details of pins on the board and their operation. The original XBee works at ***3.3V voltage supply*** and all its I/O pins work at 3.3V. The adapter board has a voltage regulator (5V -> 3.3V), and a level converter IC which converts all ***5V logic*** to ***3.3V logic***. ***The Receive LED*** *(red)* stays on when the XBee module is receiving wireless data (say, from another XBee module). The ***Power LED*** *(green)* indicates the following depending upon how it blinks:

(i) *Not glowing*: No, or too low supply voltage.

(ii) *Blinking slowly*: XBee module is in sleep mode.

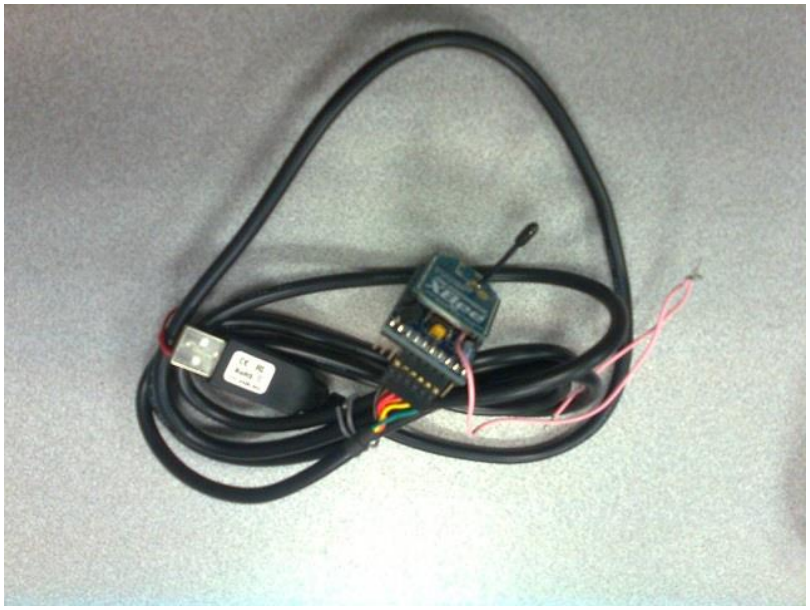
(iii) *Blinking normally*: Supply voltage is ok. XBee is working normally.

(iv) *Blinking rapidly*: Supply voltage is not proper.



Pin details of the XBee module

An ***FTDI cable*** is required to connect the module to the PC on the ***USB port***. The FTDI cable creates a ***Virtual COM port*** when connected to the PC. A device can communicate with the FTDI cable as if it were a ***UART device***. In the same way, programs on the PC can talk with the FTDI cable as if it was a UART device connected to a ***COM port*** of the PC. The FTDI cable acts as a middle man and allows the entities on both ends to communicate as if the USB port were a serial port.



FTDI cable to connect XBee to PC

To use the FTDI cable, we need a ***Virtual COM Port (VCP)*** *driver*. This can be downloaded from the following FTDI page: http://www.ftdichip.com/Drivers/VCP.htm

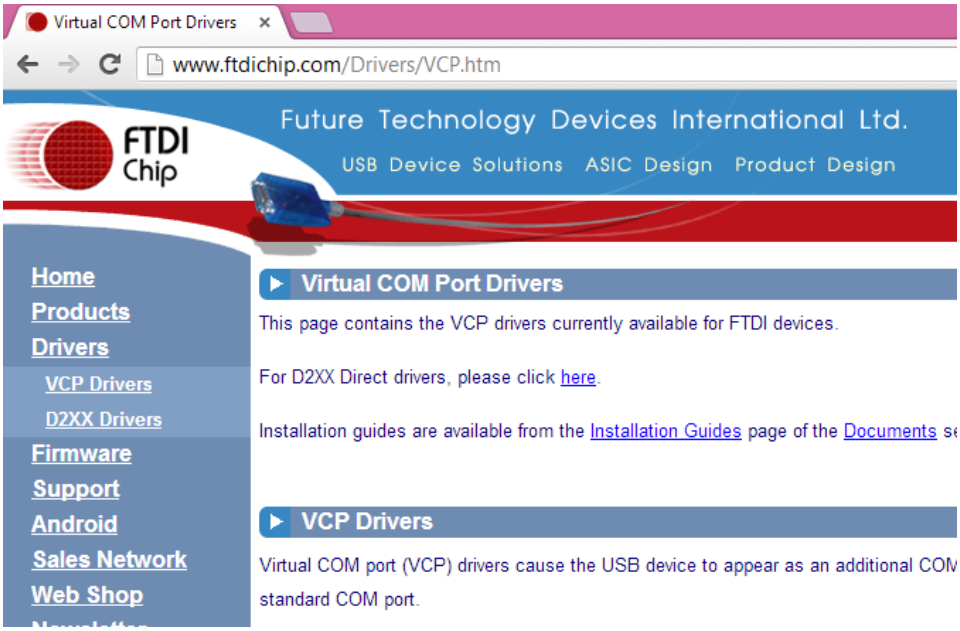
If a setup executable is available for the driver, then you can just download and install. However, if a ZIP package is only available, the follow the following installation procedure on a Windows OS:

(i) Insert the FTDI cable into a USP port on your PC. Windows will try to automatically install the driver for the device.

(ii) After it finishes, go to ***Control Panel***, and click on ***Add a device***, select your newly added device (the FTDI cable), and the click on next.

(iii) When it asks, choose to ***manually install*** the driver for the device from a location on your computer.

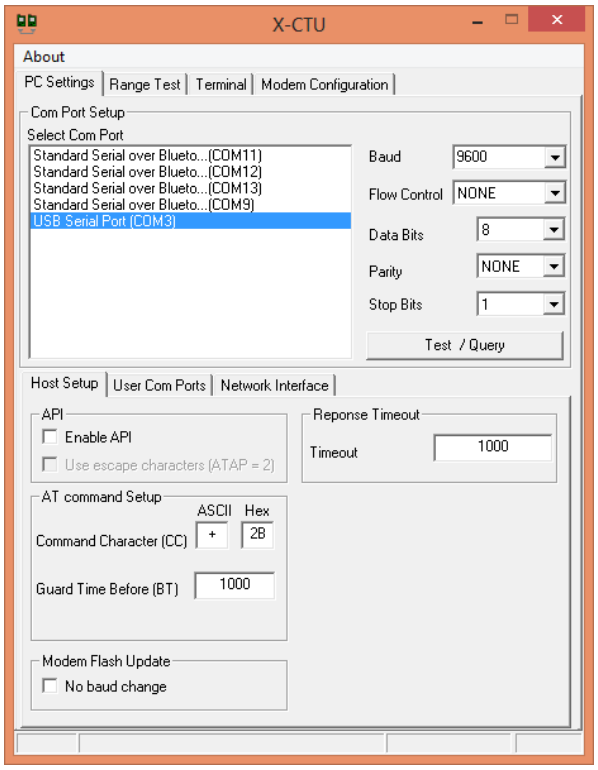
(iv) Extract the contents of the ZIP file, and then browse for the drivers in the extracted folder. Windows should now automatically search and install the driver.



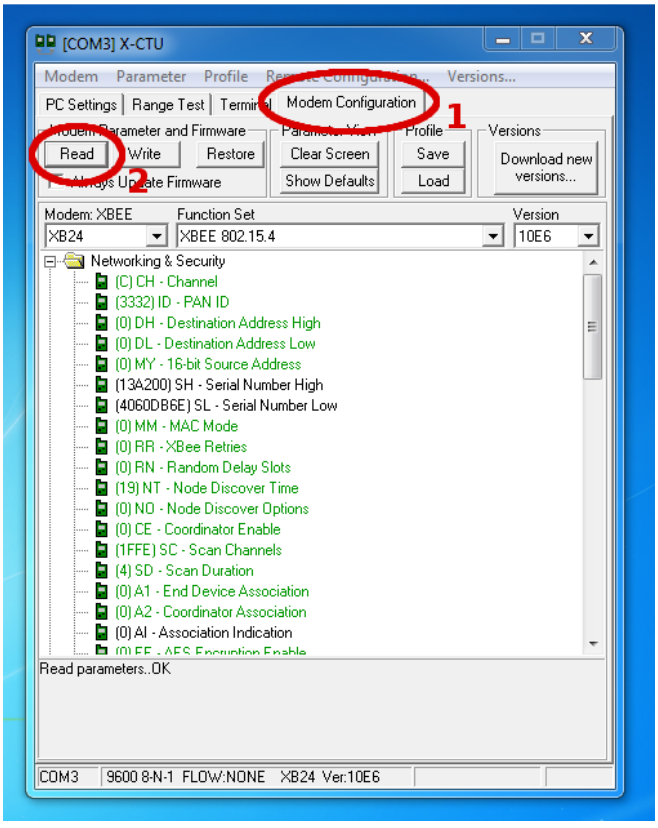
Virtual COM port driver:
http://www.ftdichip.com/Drivers/VCP.htm

With the VCP driver installed your PC will now treat the FTDI cable as a COM port on your PC. We can now use this COM port using *serial communication software* called ***X-CTU***. It can be downloaded from the ***General Diagnostics, Utilities and MIBs*** *section* of the following **Digi** (manufacturer of XBee) page: http://www.digi.com/support/productdetail?pid=3352

Connect the XBee adapter board to the FTDI cable and then connect the FTDI cable to the USB port of your computer. Run the X-CTU software. A window similar to the one shown below should show up, and most necessarily, should have *at least one COM port* in the Select COM port section to select. This indicates that the installation of driver was successful.



X-CTU for Serial Comm.:
http://www.digi.com/support/productdetail?pid=3



Getting XBee modem config on X-CTU

By *default* XBee modules are shipped with the following default settings:

(i) *Baud rate*: 9600 bps

(ii) *Parity*: None

(iii) *Stop bits*: 1

Therefore we do not need to make any changes in the ***PC Settings*** *tab*. If your XBee has a different setting, then select the appropriate settings. To move on to the ***Terminal*** *tab* and enter the following bold text:

+++ *[OK]*

ATRE <Enter>

[OK]

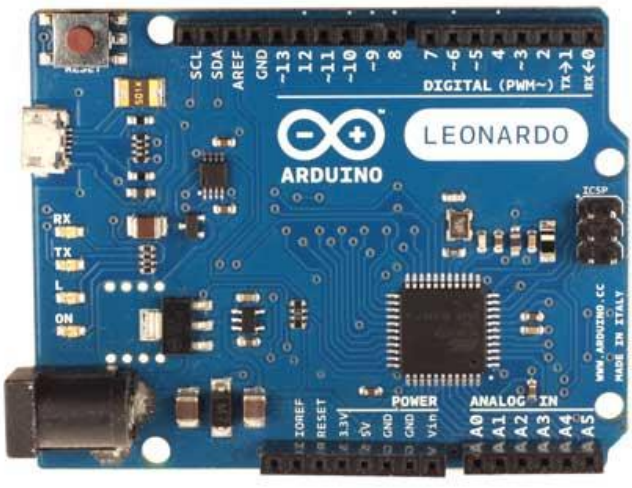
ATWR <Enter>

[OK]

ATCN <Enter>

[OK]

Note that all the ***data sent to the XBee*** are shown in ***blue color***, and the ***data received from the XBee*** are shown in ***red color***.



Arduino Leonardo board: (used this originally for testing the Xbee module)

The XBee module (or modem) can be controlled / configured using a set of commands called ***AT Commands***. Here is a list of AT commands which can be used to configure the XBee module: https://dlnmh9ip6v2uc.cloudfront.net/learn/materials/29/22AT%20Commands.pdf

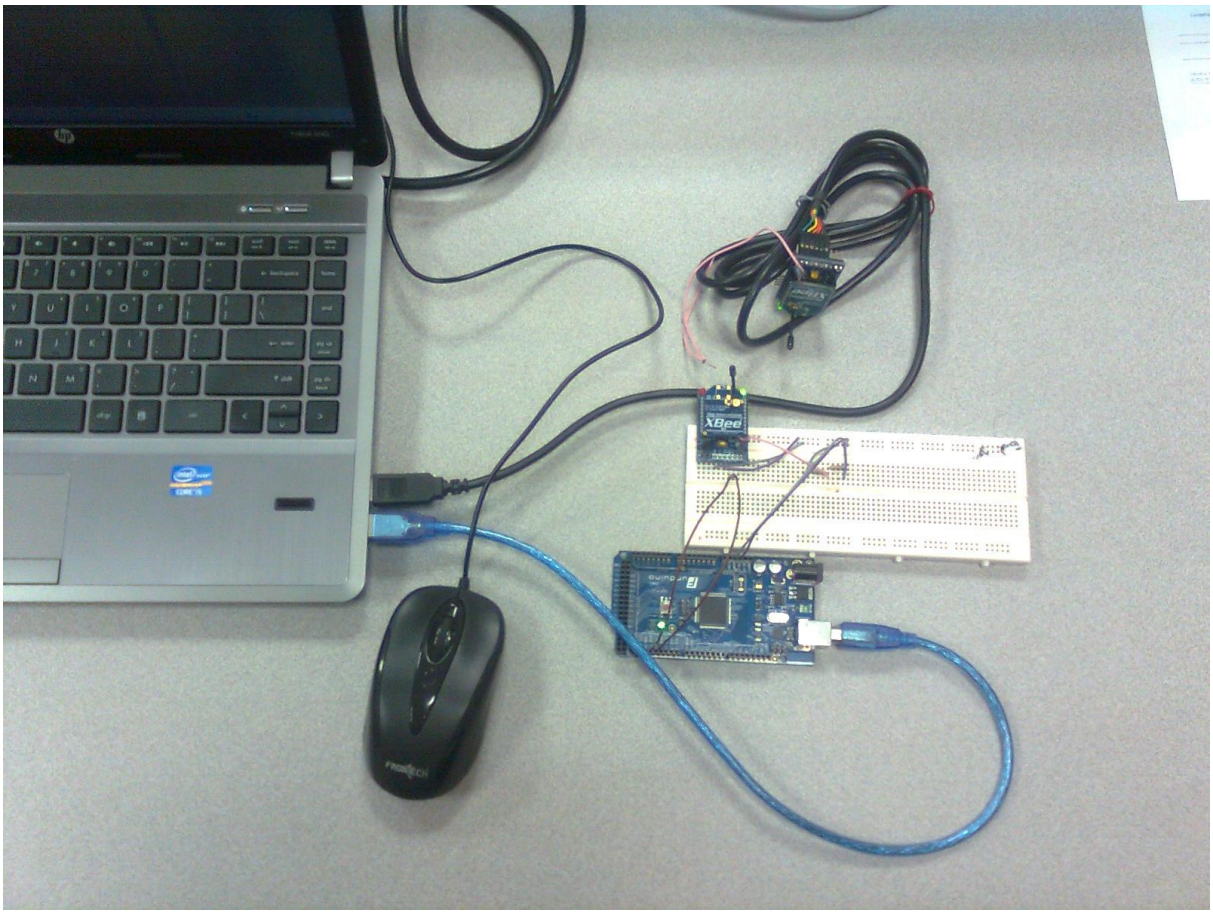
After an AT command is entered, the XBee module replies ***OK*** or ***ERROR*** depending upon whether the command was successfully executed or not. By default the XBee module transmits all data wirelessly, sent to it over serial port. It also sends all received wireless data to the serial port. However, when the ***+++*** character sequence is sent over the serial port to the XBee *within a time limit*, the XBee enters into ***AT Command Mode***. In this mode, the XBee is *ready to receive AT commands*. *This mode ends* when ***ATCN*** command is entered, or *after some time of not entering any AT commands*. The command ***ATRE*** *restores the factory settings*, and ***ATWR*** *write those settings to* ***non-volatile memory*** (which means that, the next time you connect your XBee to the PC, it would have retained the settings). Here is the link to XBee datasheet: https://www.sparkfun.com/datasheets/Wireless/Zigbee/XBee-Datasheet.pdf

The XBee can also be configured using the ***Modem Configuration*** tab. Click on the ***Read*** button i***n Modem Parameter and Firmware*** region. If an ***Action Required*** window pops up asking you to reset the modem, then press the reset button on the adapter board (if available) or short the reset terminal and common ground terminal. If successful, Modem name / type, Function set, and its Version will be displayed along with all its settings.

Set the ***PAN ID*** to ***1000*** (Personal Area Network ID), ***Destination address (high)*** to ***0*** (high DWORD of address of destination device / XBee), ***Destination address (low)*** to ***0*** and ***Source address*** to ***1*** (the source address of this XBee). Make sure that all other settings are set to *default* (shown in green color) Click on the ***Write*** *button* to save the updated configuration to XBee. Now detach that XBee module from the FTDI cable and connect another XBee module to the FTDI cable. Proceed to the ***Modem Configuration*** *tab* and click on the ***Read*** *button* in ***Modem Parameter and Firmware*** *region*. Set the ***PAN ID*** to ***1000***, ***Destination address (high)*** to ***0***, ***Destination address (low)*** to ***1*** and ***Source address*** to ***0***. Click on the ***Write*** *button* to save the updated configuration to XBee.

Basic XBee Settings

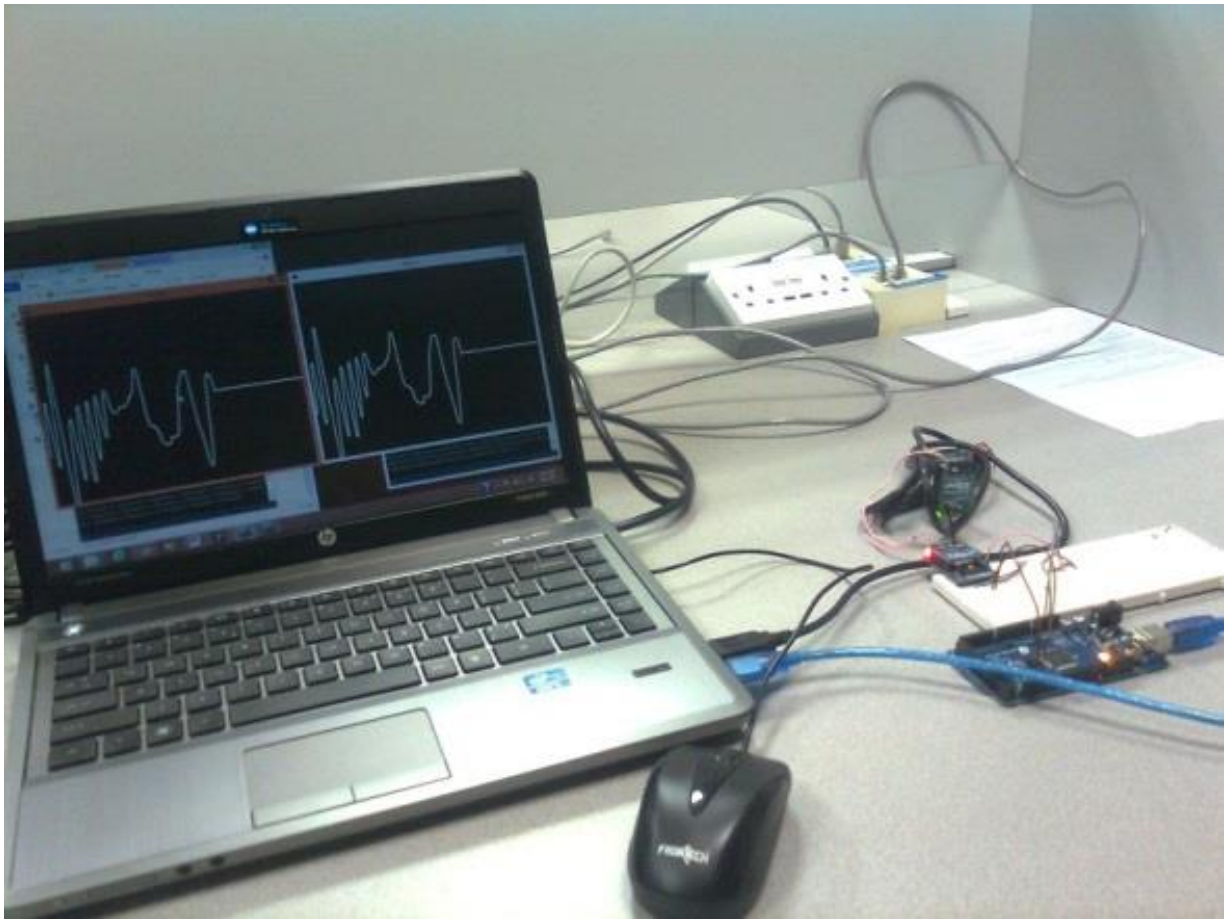
Pan ID	1000
Destination address (high)	0
Destination address (low)	1
Source address	0



Experimental setup for basic XBee to XBee communication

Both the XBees are now *ready to communicate* with each other. If you have two FTDI cables, you can simply connect both the XBee modules to your PC using the cables and open *two X-CTU windows* (with *appropriate serial ports selected* in ***PC Settings*** *tab* for each X-CTU window). Now enter something in the ***Terminal*** *tab* of a window, and it should appear in the other window.

However, in our case we did not have 2 FTDI cables, and so we used an ***Arduino Leonardo*** *board* in place of an FTDI cable. The first XBee was directly connected to the PC using FTDI cable, and the second XBee was connected to the ***serial port of the Arduino board***, and the *Arduino board* was itself connected to the PC over a USB port (which again behaves as a *Virtual COM port*). A program for Arduino was written so that it could behave in the same way as an FTDI cable. The program is uploaded at: https://github.com/wolfram77/SmartGrid/blob/master/Core/zOld/Xbee_Ard/SrTalk/SrTalk.ino

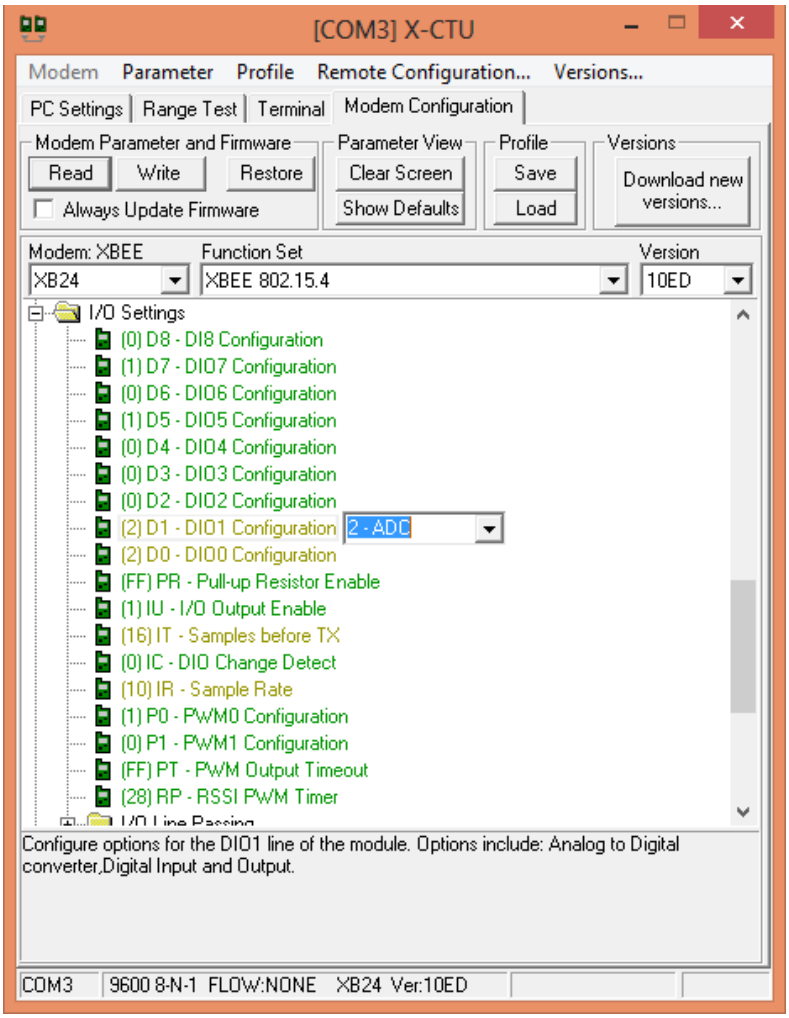


Sent and received signals (signals were generated manually using the PC)

The program was ***uploaded*** to the *Arduino board*. The connections are shown in the diagram below. ***Serial monitor*** was used to send and receive serial data for second XBee, and X-CTU was used for first XBee. This completed the test of XBee modules.

3. DISPLAYING UNIPOLAR ANALOG SIGNALS ON PC USING IN-BUILT ADC IN XBEE

In this experiment, we will be sending ***unipolar*** *analog signals* from the *function generator* to the PC for display, directly using the ***in-built ADC in XBee***. First, we need to configure the XBee module to use ADC. To do this, connect the XBee module to the FTDI cable and then connect it to the PC. Now open the X-CTU software, select the appropriate baud rate in the ***PC Settings*** *tab*. Now proceed to the ***Modem Configuration*** *tab* and click on the ***Read*** *button* in ***Modem Parameter and Firmware*** *region*. X-CTU should then load all the settings from the XBee module. Once loaded, in I/O configuration settings set ***I/O configuration*** for ***D0*** and ***D1*** as ***2 – ADC*** (ADC will be used for pins D0 and D1). Set the ***Samples before TX*** to ***16*** (after recording 16 samples of analog signal on each required pin a packet will be sent), ***Sample rate*** to ***10*** (a sample from each required pin will be taken once every 10ms). Let the rest of the settings be the same as in the previous experiment.



Setting up XBee for using in-built ADC using X-CTU

When ADC is enabled on any pin in XBee, it sends data in a particular ***packet format***.

XBee packet format:

Standard XBee packet example:
7E 00 1C 83 56 78 22 00 05 06 00 00 00 03 FF 00 00 03 FF 00 00 03 FF 00 00 03 FF 00 00 03 FF 77

Where the UART API data stream can be broken down as:

7E *Start Delimiter*

00 1C *Length Bytes*

83 *API Identifier Byte for 16bit A/D data (82 is for 64bit A/D data)*

56 78 *Source Address Bytes*

22 *RSSI Value Bytes*

00 *Option Byte*

05 *Sample Quantity Byte*

06 00 *00000110 00000000 Channel Indicator **

00 00 *Sample Data ADC0 (min value for A/D is 00 00)*

03 FF *Sample Data ADC1 (max value for A/D is 03 FF, indicating the value of ADC1 is Vref)*

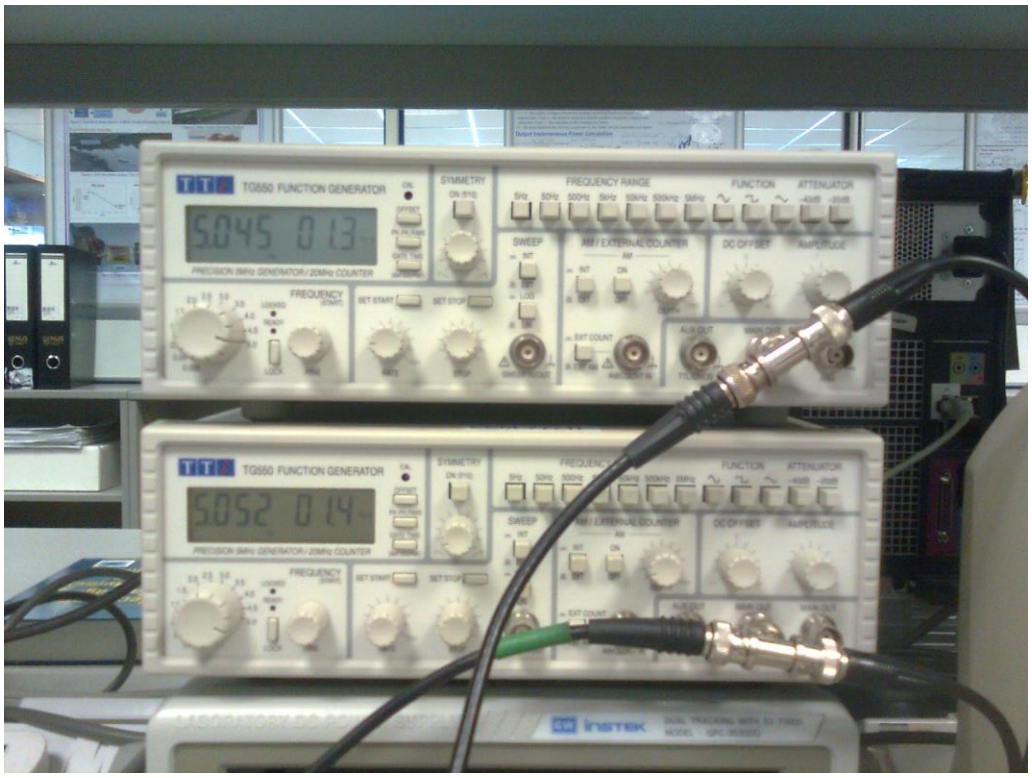
77 *Check Sum*

*** bit field where the bits line up as (Ax=ADCx & Dx=DIOx):*

MSB = 0x06 *LSB = 0x00*

0	*0*	*0*	*0*	*0*	*1*	*1*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
NA	*A5*	*A4*	*A3*	*A2*	*A1*	*A0*	*D8*	*D7*	*D6*	*D5*	*D4*	*D3*	*D2*	*D1*	*D0*

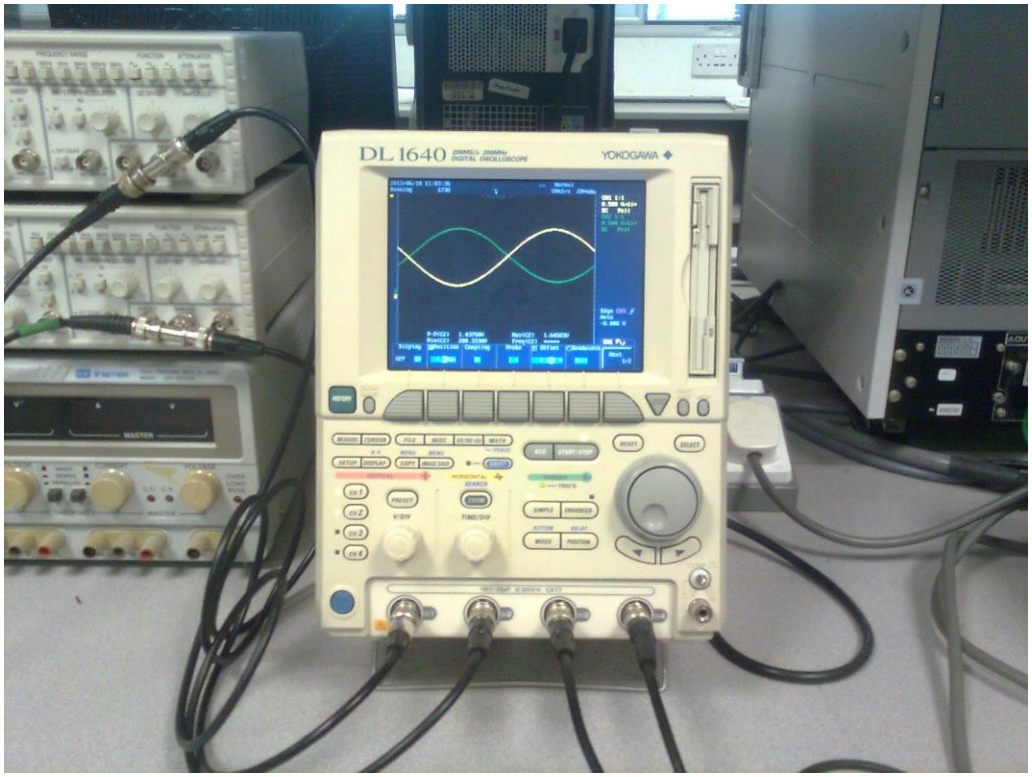
Pasted from <http://www.digi.com/support/kbase/kbaseresultdetl?id=2180>



Two function generators being used to generate 2 unipolar signals to be measured

Experimental setup for capturing 2 unipolar analog signals using in-built ADC present in XBee

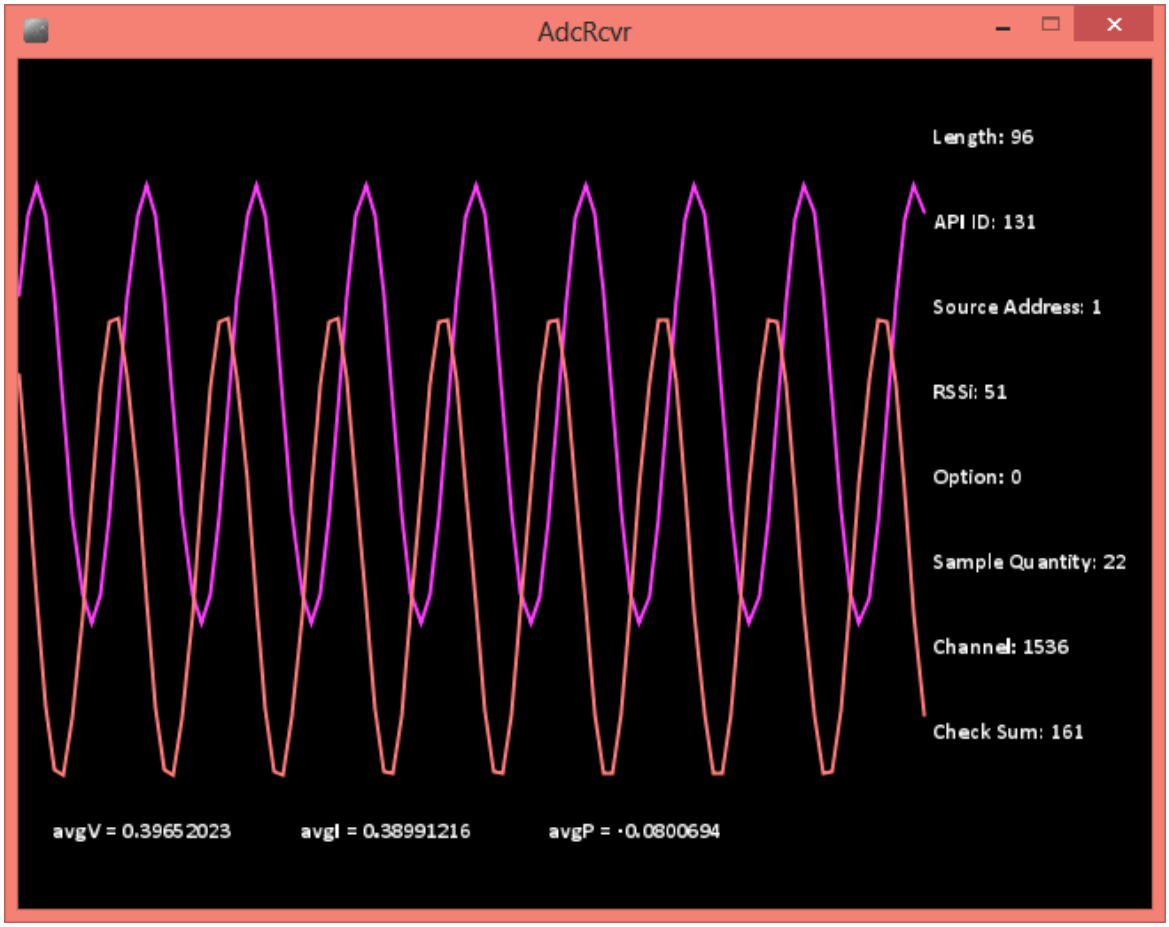
The first XBee module is configured to send ADC data to second XBee, and the second XBee is configured to receive. Now provide the first XBee module with 5V voltage supply. The XBee module connected to your PC should now be receiving some data (indicated by a *glowing red LED* on the XBee module). Go to the X-CTU ***Terminal*** *tab*. You should be able to see some characters being received. These are data sent by the first XBee to the XBee connected to your PC. Since these data are in a particular *packet format*, and not in a human readable form, it cannot be understood directly.



2 unipolar analog signals being displayed on an oscilloscope

The ***ADC Receiver*** *program* for decoding the packet format and accordingly displaying it on the screen can be used to view the *received analog signal*. It has been uploaded at: https://github.com/wolfram77/SmartGrid/tree/master/Core/zOld/Xbee_PC/AdcRcvr

Now we need to connect the analog signals to the XBee appropriately. Solder a ***4 pin right-angled berg strip*** to the ***pins 17-20*** (*IO3 - IO0*) of the XBee. Solder a ***wire*** to ***pin 14*** (*reference voltage pin*) of the XBee. Make connections as shown in the diagram below. Connect a ***potentiometer*** (*POT*) to the ***Vref*** *pin* of the XBee module and set the voltage at the pin to ***around 2.5V*** (using the POT). Make proper connections of I/O pins with the function generators and make sure that the *minimum output voltage* of

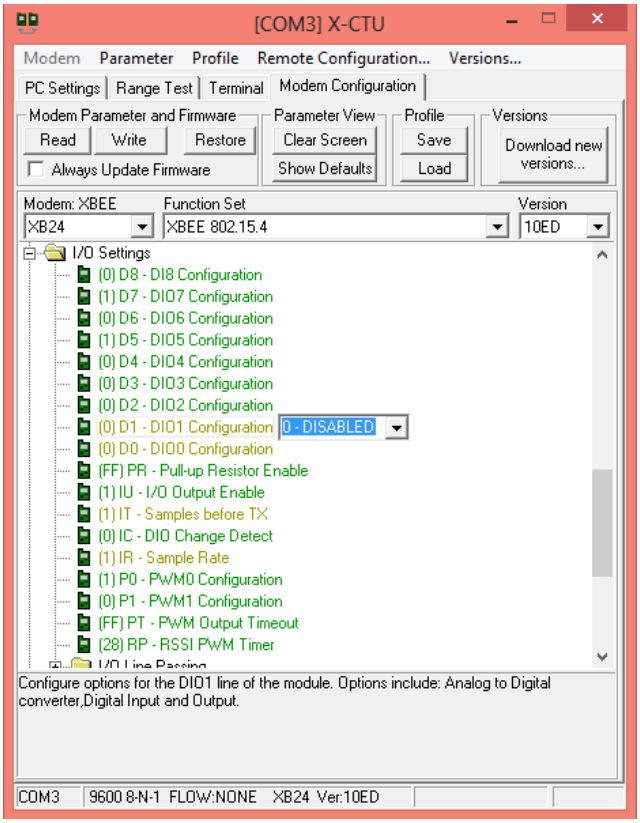


Received and decoded analog signals received on PC using the developed ADC receiver program (using Processing) at around 160sps

the output signals from the function generators is *greater than 0V*, and that the *maximum output voltage* from the function generators is *less than the reference voltage*. After it is done, turn on the first XBee. The *red LED* on the second XBee must glow. Open ***Processing*** (an open source IDE using which the program was written) and run the ***ADC Receiver*** *program*. You should now be able to see ***3 waveforms*** on your PC (the *third wave form is the product of the first two waveforms*, which would represent instantaneous power if the transmitted signals represented instantaneous voltage and current). This completes this experiment.

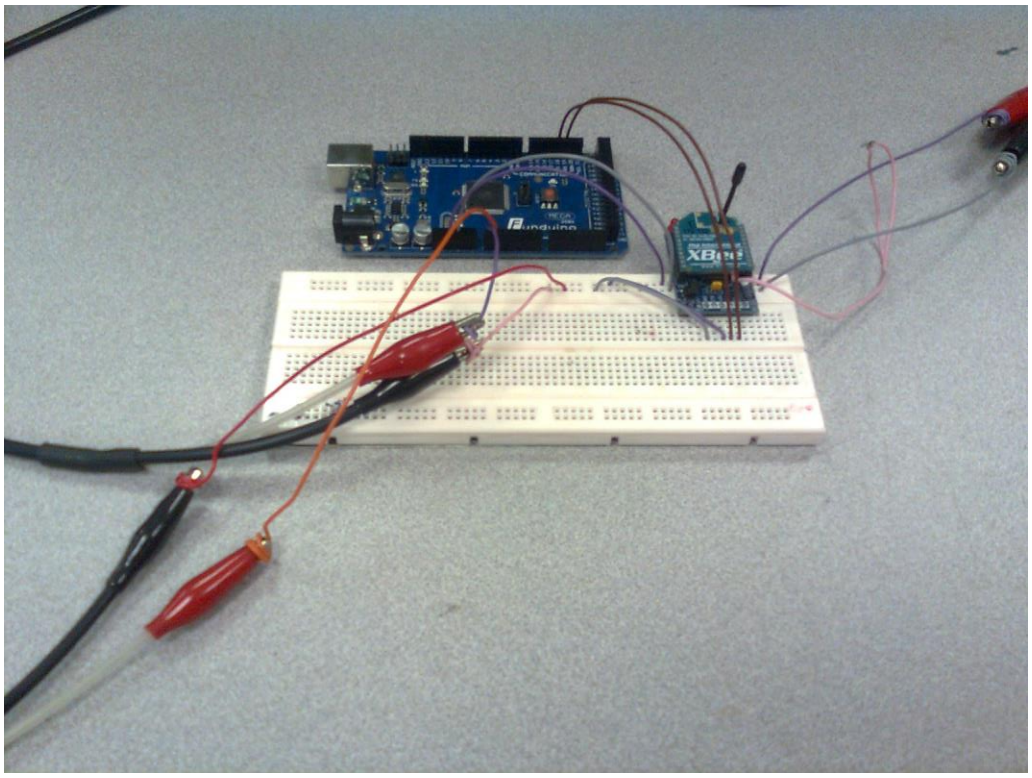
4. DISPLAYING UNIPOLAR ANALOG SIGNALS ON PC USING ARDUINO

In the previous experiment, we were able to send ***unipolar*** *analog signals* from the function generator to the PC for display, directly using the ***in-built ADC*** in XBee. However, this causes the XBee to continuously transmit ADC data to PC, and also at a *low sampling rate*. Due to continuous data transmission, the XBee *consumes a lot of power*. This is not desirable in most situations. In case of a *Wireless sensor network (WSN)*, we actually *do not need to send all sampled data to a server*. All we need to send is the changes in certain factors of voltage, current or power. In such situations we can measure the signals using a microcontroller and as per the condition, decide whether the data collected is worth transmitting. When using microcontrollers, we also gain the benefit of a *higher sampling rate* and hence are able to collect accurate values relating to the signals.

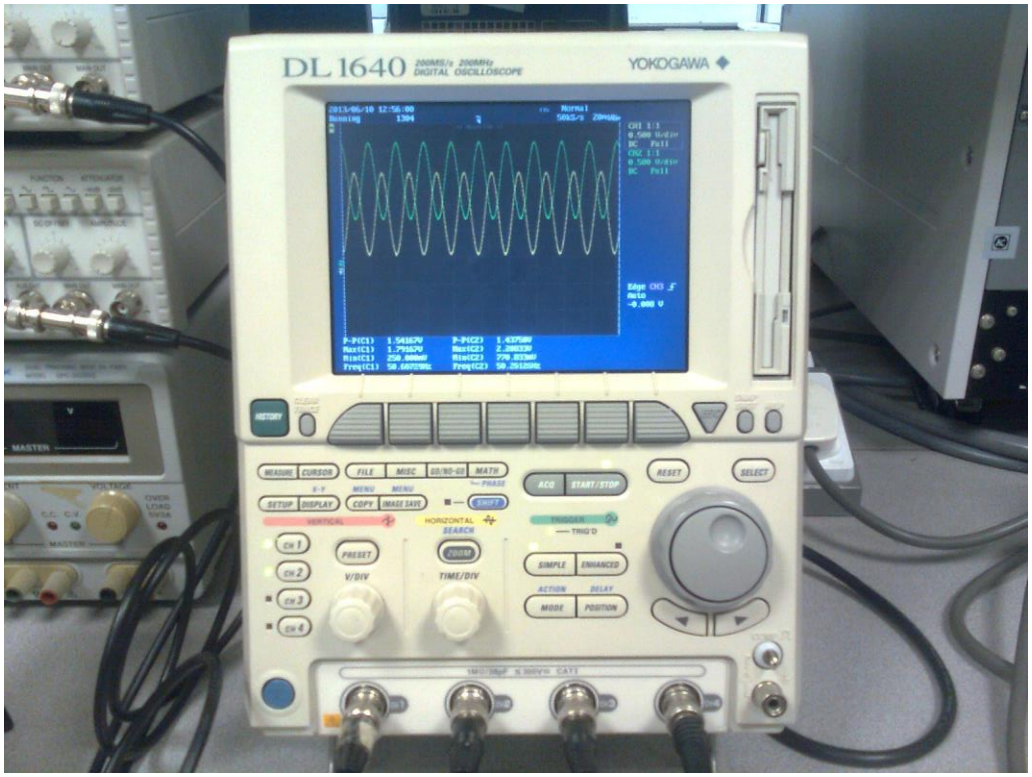


Configuring XBee to default settings

In this experiment we are using an ***Arduino Mega 2560*** *board* to display *2 unipolar analog signals* on the PC with a *sampling rate* of ***1ksps***. We will still be using the ***ADC Receiving*** *program* on the PC. Hence we

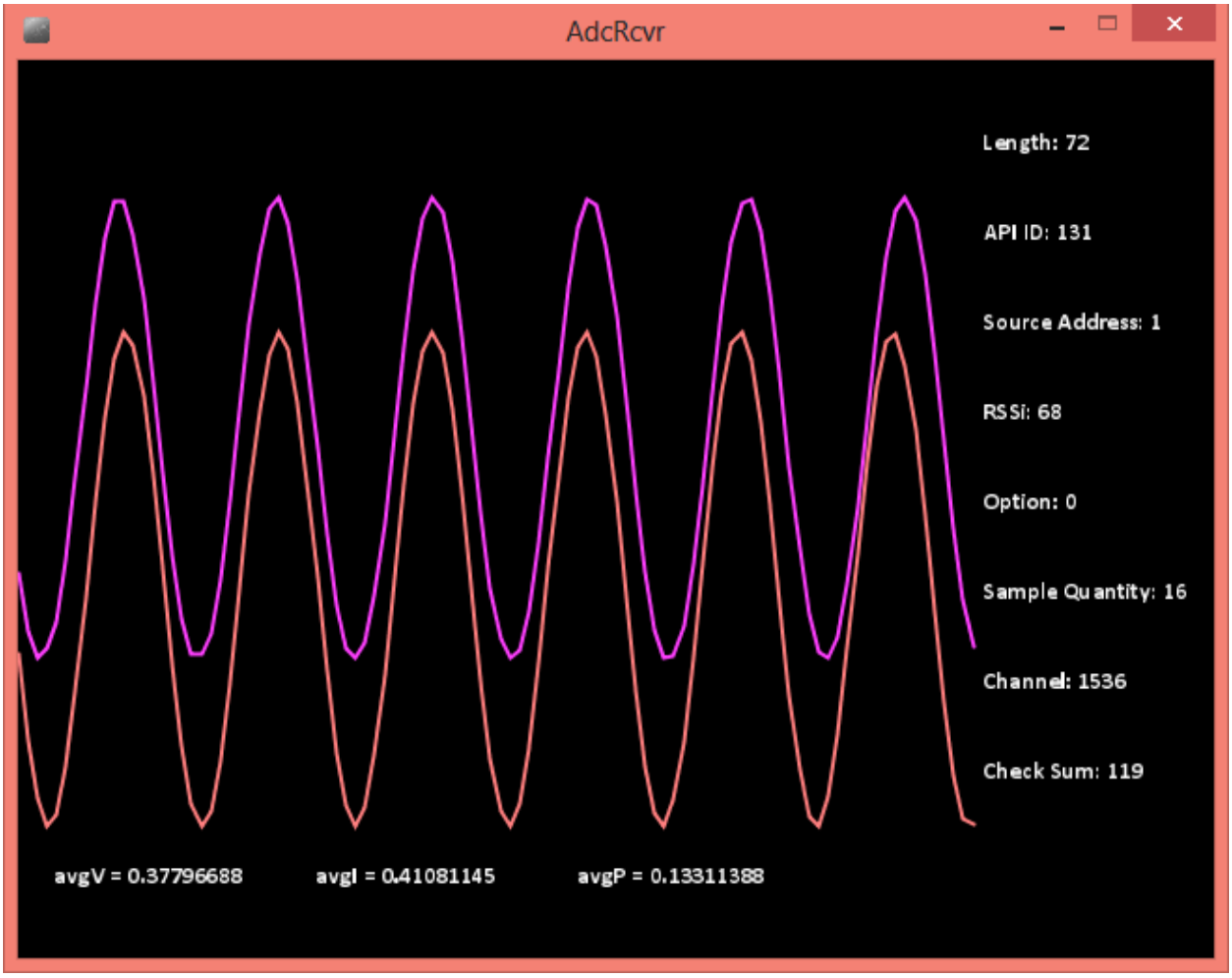


Experimental setup for sending unipolar analog signals to PC using Arduino



Original analog signals being sent by Arduino on oscilloscope

need a program to “emulate” XBee ADC. To do this we will *capture a set of samples* and transmit it to the PC in ***XBee packet format***. The ***Serial ADC*** *program* for the Arduino microcontroller board is uploaded at: https://github.com/wolfram77/SmartGrid/blob/master/Core/zOld/Xbee_Ard/AdcSr/AdcSr.ino

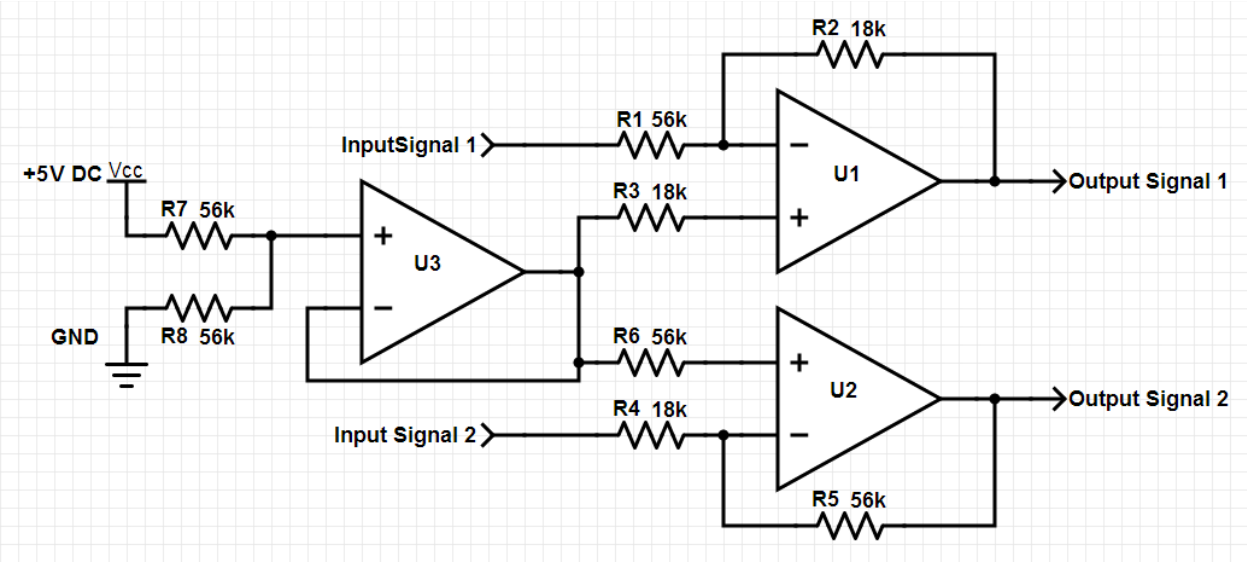


Received and decoded analog signals on PC with a sampling rate of around 1ksps

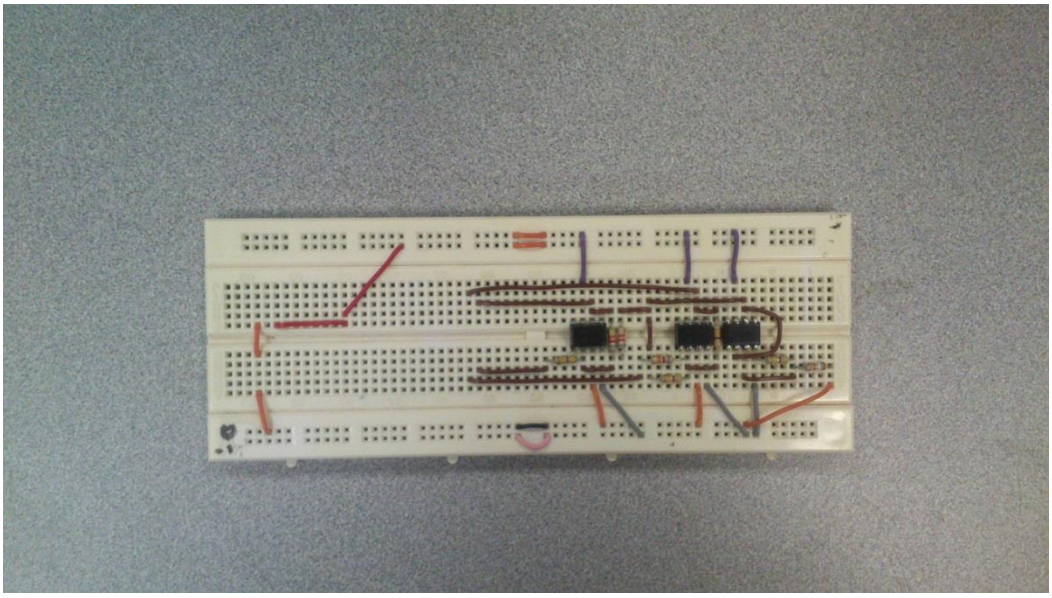
The new connections to be made (from the previous experiment) are shown below. Make sure that the output from the signal generator has *appropriate* ***DC offset voltage*** so that ***Vmin*** is always *greater than 0V*. Adjust the amplitude of the function generator so that ***Vmax*** is *less than 2.5V* (we are using *internal 2.56V analog reference* in Arduino). Once the ***ADC Receiver*** *program* is run on PC and the circuit is powered, we get a display of *2 analog signals* from the signal generators on the PC.

5. DISPLAYING BIPOLAR ANALOG SIGNALS ON PC USING ARDUINO

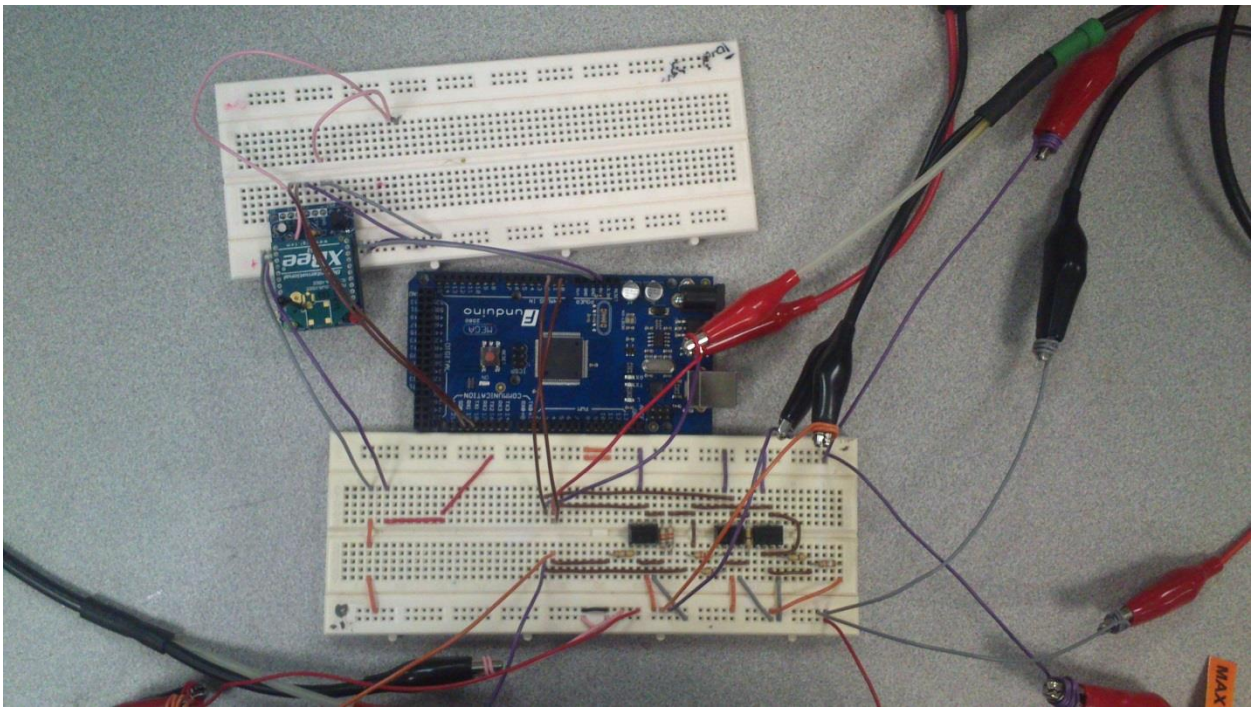
We are able to send ***unipolar*** *analog signals* from the function generator to the PC for display. However, in practical situations we need to be capable of displaying ***bipolar*** *analog signals*. We also need to do amplify / attenuate signals according to their *peak-peak voltage*. For this purpose, we design a ***level shifting and amplifying circuit*** with the following factors (these factors were obtained for use with transducers to be used in the next experiment):



Signal conditioning circuit designed to convert bipolar signals to unipolar signal centered at 2.5V and with scaling factors of 18/56 and 56/18 for signals 1 and 2 respectively

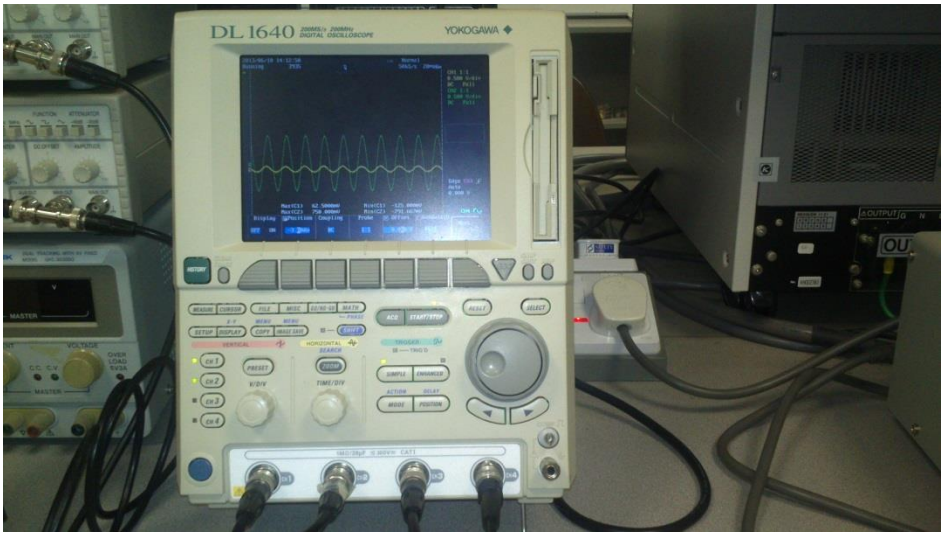


Signal Conditioning circuit on the bread board

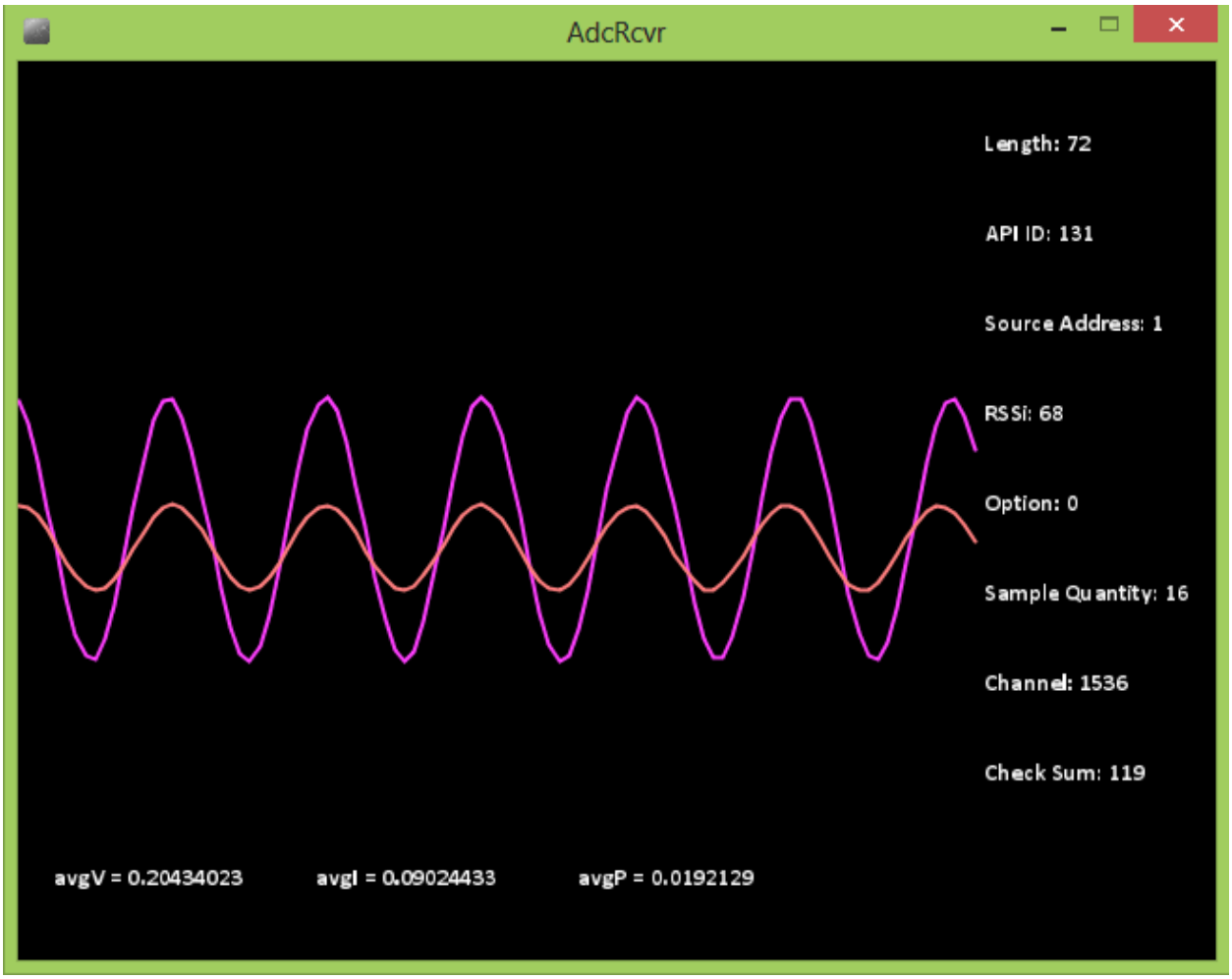


Experimental setup for capturing 2 bipolar analog signals using signal conditioning circuit and Arduino

1. *Signal 1*
    a. *Level Shift*: 1.24V [≈ 15V * (39k / 470k)]
    b. *Amplification*: 0.17 [≈ (82k / 470k)]
2. *Signal 2*
    a. *Level Shift*: 1.24V
    b. *Amplification*: 5.7 [≈ (470k / 82k)]



2 bipolar analog signals being displayed on an oscilloscope

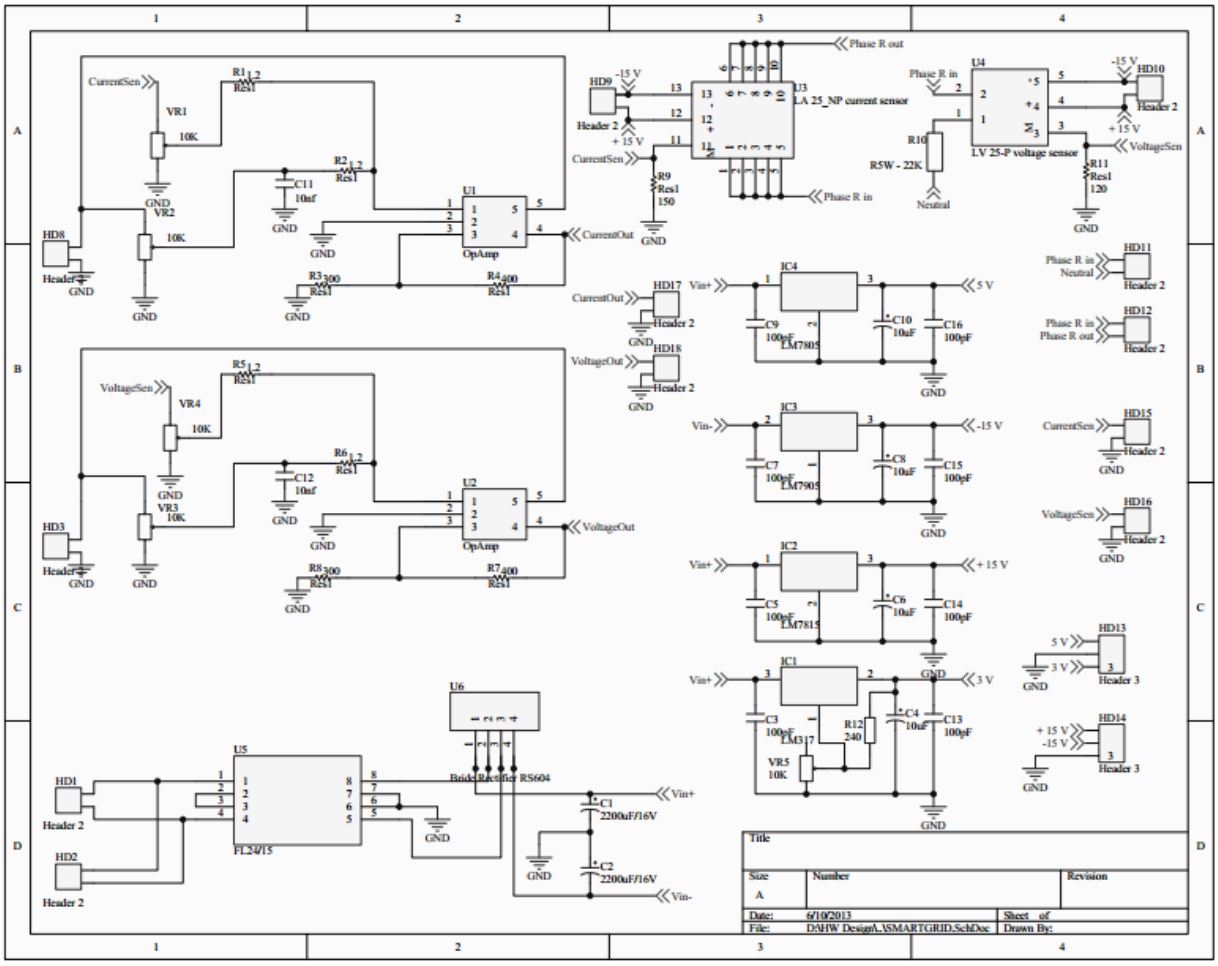


Received and decoded analog signals received on PC using the developed ADC receiver program (using Processing) at around 1ksps

The *level shifting and amplification* is done using a ***voltage summer with amplification circuit***. The circuit diagram of the circuit is shown below. The circuit requires ***+15V, 0V, -15V*** *voltage supply*. After connecting the *bipolar analog signals* (*within a limited range*) we obtain a level-shifted and scaled signal. This signal is then fed to the *Arduino board* with ***Serial ADC*** *program* which then sends over the data to PC using XBee to another XBee connected to XBee. The ***ADC receiver*** *program* on the PC then displays the waveforms. This completes this experiment.

6. MEASURING INSTANTANEOUS VOLTAGE, CURRENT AND POWER

Now that we are able to measure and transmit a ***bipolar*** *analog signal* to the PC for display, we can move on to measuring things that matter to us:

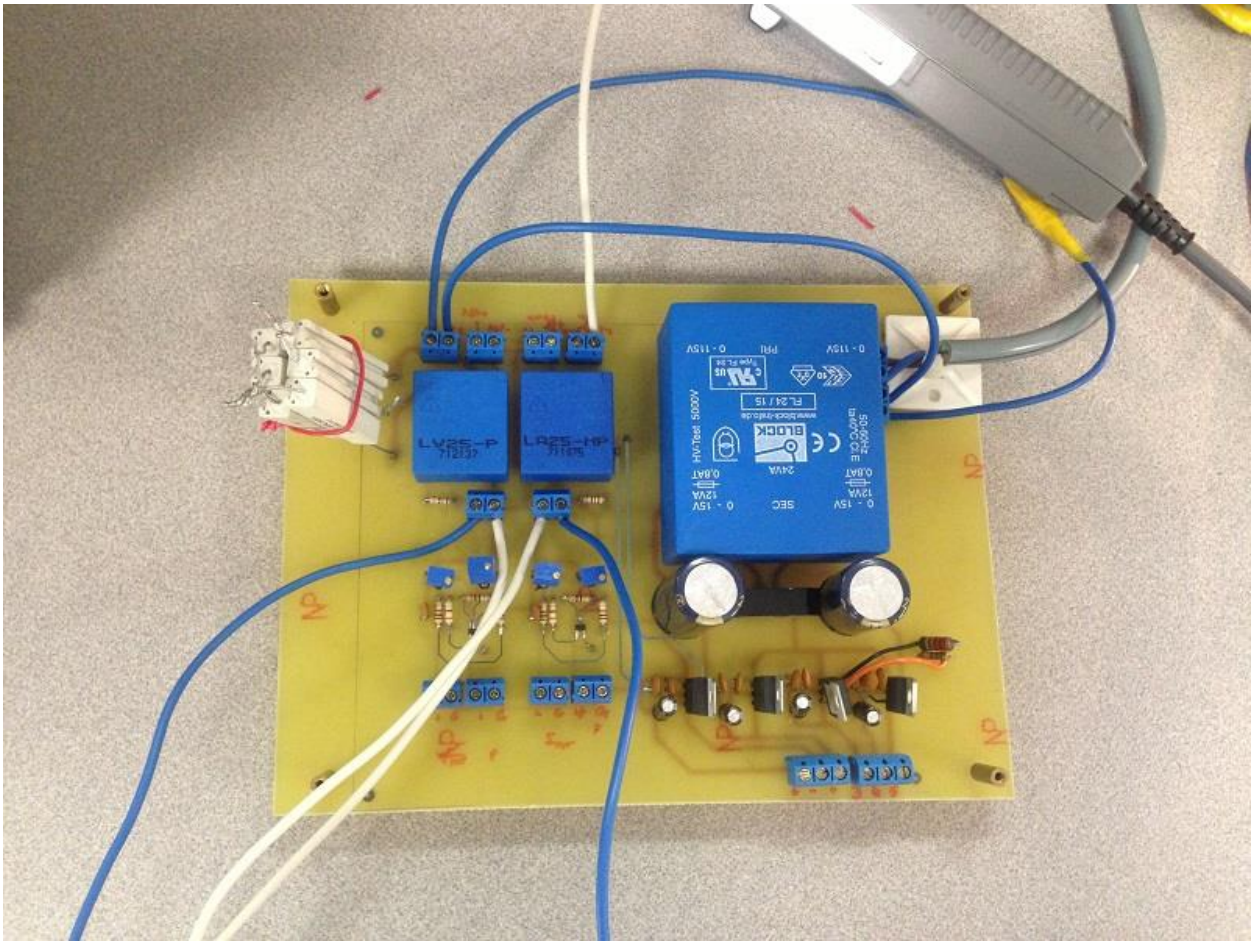


Voltage and Current sensing circuit with Power Supply (+15V, +5V, +3.3V, 0V, -15V) designed by Chinh & Bhuneswar to measure the voltage and current at an output socket

1. *Measure the Supply Voltage*:
    a. To determine the *purity of AC voltage supply*.
    b. *Automatically protect devices* from unsuitable voltage supply.
2. *Measure the Intake Current*:
    a. To *determine the fluctuations introduced by a device* to the supply, this may hamper other devices.

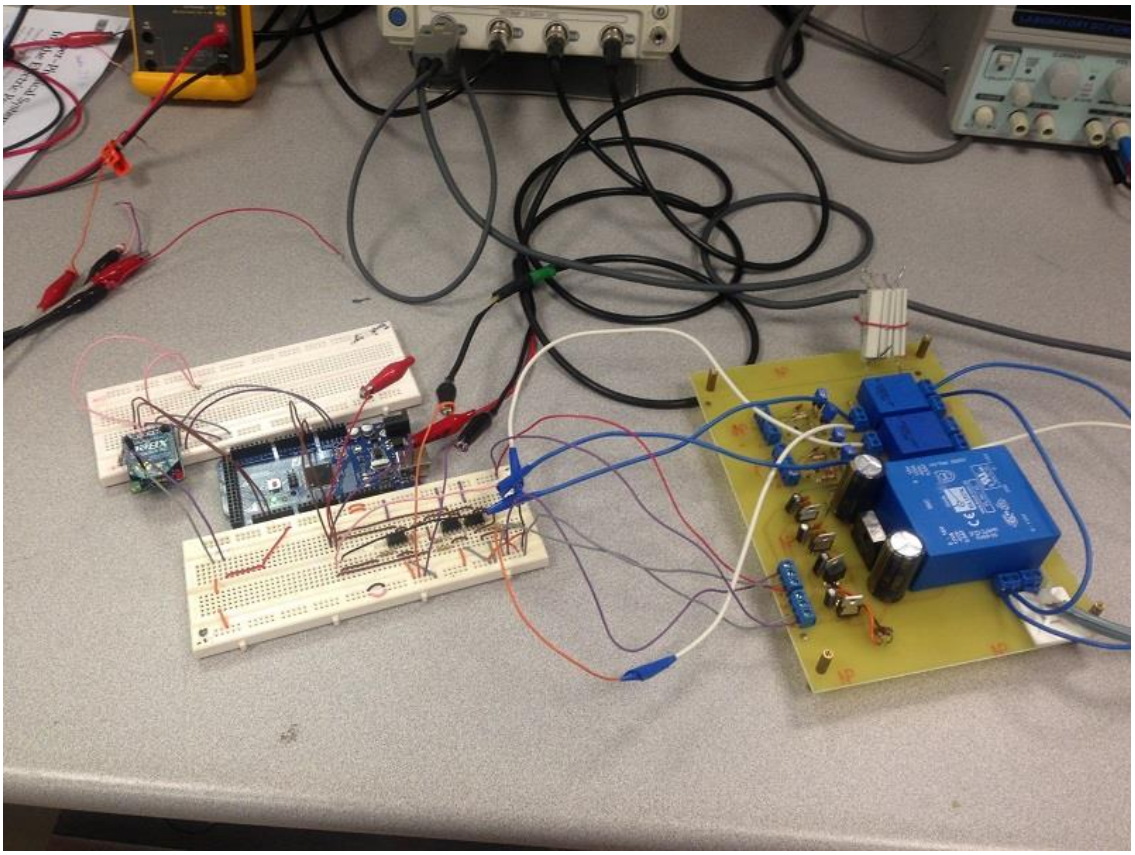
b. To *shut down a device* when it is taking unexpectedly high current.

3. *Measure the input Power*:
    a. To determine *which device is consuming the maximum power* at the present.
    b. To *automatically shut down devices*, when not in use.
    c. To *estimate power loss due to connecting wires*.

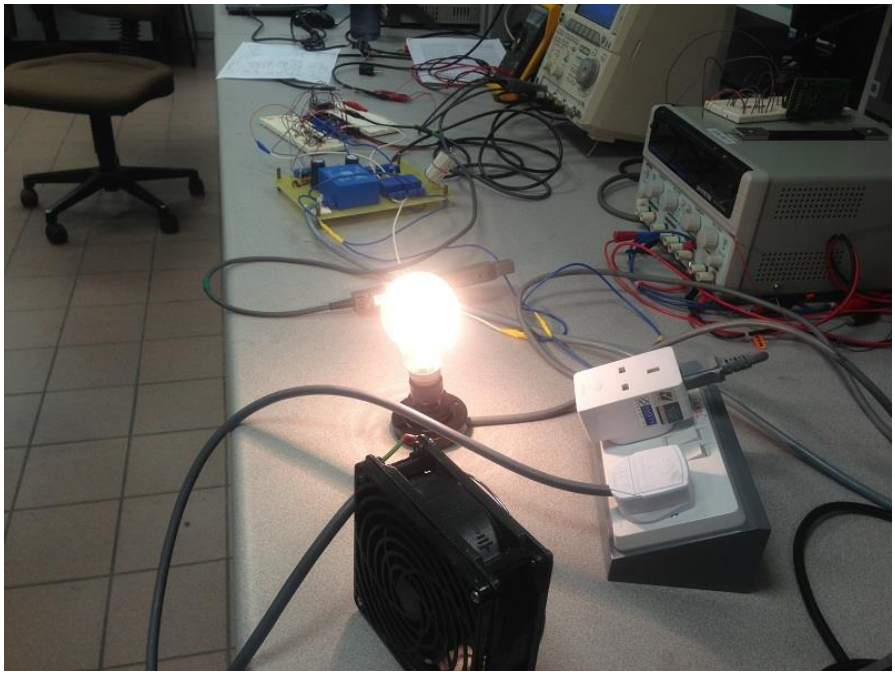


The Voltage and Current sensing circuit

Shown above is the schematic of the circuit we used to obtain voltage and current signals. The circuit provides ***+15V, +5V, +3.3V, 0V, -15V*** *as power supply*, and uses ***Voltage transducer LV-25P***, and ***Current Transducer LA-25NP***. These transducers are basically *transformers*, with appropriate resistors to provide a measurable voltage. They also provide *galvanic protection* which is very important in *high voltage applications*. The outputs of these transducers would be connected to the *level shifting circuit* we designed, and then to the Arduino board to convert the analog signals to digital and then transmit it

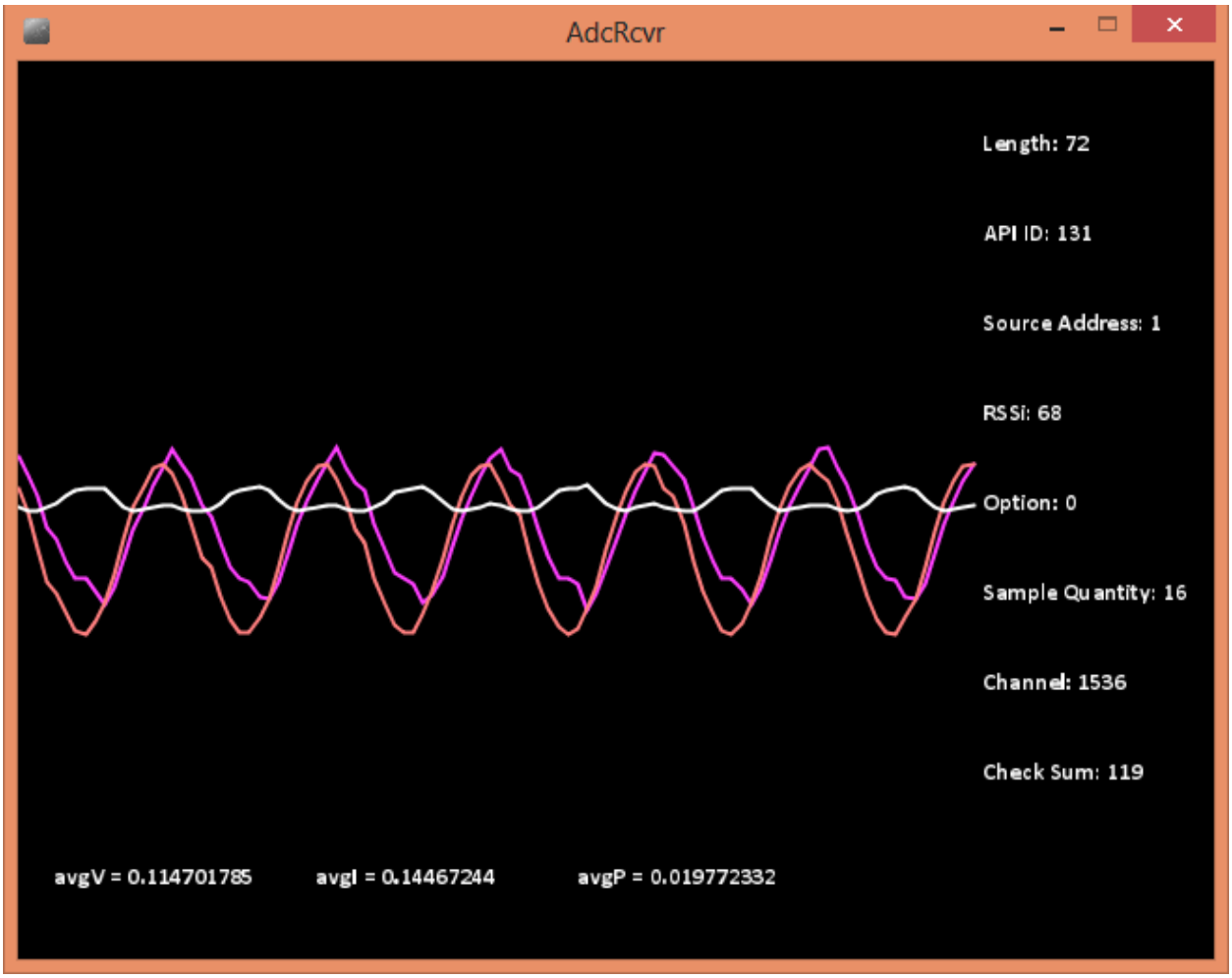


Experimental setup for sensing voltage and current used on a socket using the voltage and current sensing board along with the signal conditioning circuit and Arduino



Loads connected to the socket of the Voltage and Current sensing circuit

over to the PC using XBee, where it will be received on another XBee module connected to the PC and then displayed using our ***ADC Reciever*** *program*.

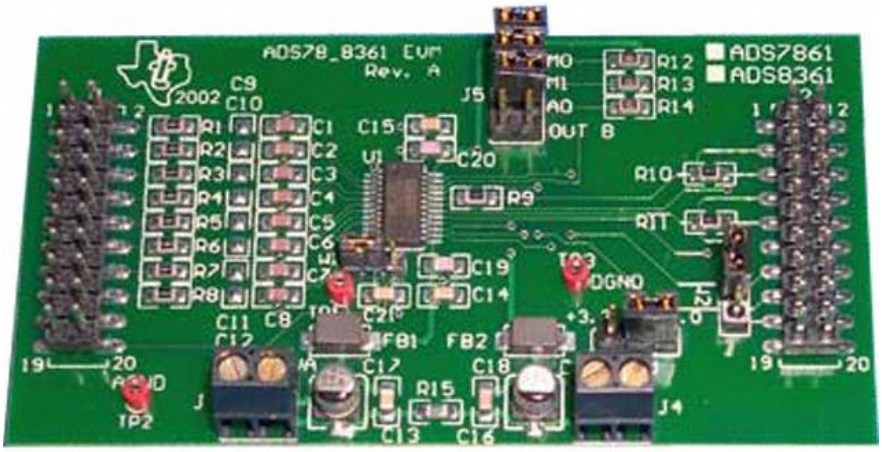


Received and decoded voltage and current sensing output received on PC using the developed ADC receiver program (using Processing) at around 1ksps

Once all the requisite connections are made, shown below**,** we start receiving voltage and current signals. In our case, we detected a small current even with no load connected. Though we have yet to understand its case, we see that the hardware is already providing some unexpected observations. However, we need to scale the received data to convert it to actual voltage, current and power values. We leave that for another experiment. For now we can observe the fluctuations in current due to load, which could give information about load quality.

7. MEASURING INSTANTANEOUS VOLTAGE, CURRENT AND POWER AT HIGHER SAMPLING RATES

In order to perform spectrum analysis of captured current and voltage samples it is required that the samples acquired for voltage and current be acquired at the same instant of time. It is also desirable to obtain the samples at a sufficiently high sampling rate. Neither of the conditions was satisfied in the previous experiment. For this experiment, we will be using ***ADS8361 EVM board*** which supports simultaneous sampling of two different channels, as well as offers a maximum sampling rate of 500ksps.



ADS8361 EVM board is a high-speed simultaneous sampling ADC board that supports sampling rate up to 500ksps, and can be interface with SPI protocol

The ADS8361 EVM board has ***4 channels*** and uses ***SPI protocol*** to transfer captured samples. We are using ***Arduino Mega 2560*** to obtain the samples from the ADC board and send it over wirelessly to a PC using XBee. Please note that XBee has a limited data rate (of *around 9 kbps* when using *9600 baud rate*). Due to this limitation we use the following method to transfer samples to the PC:

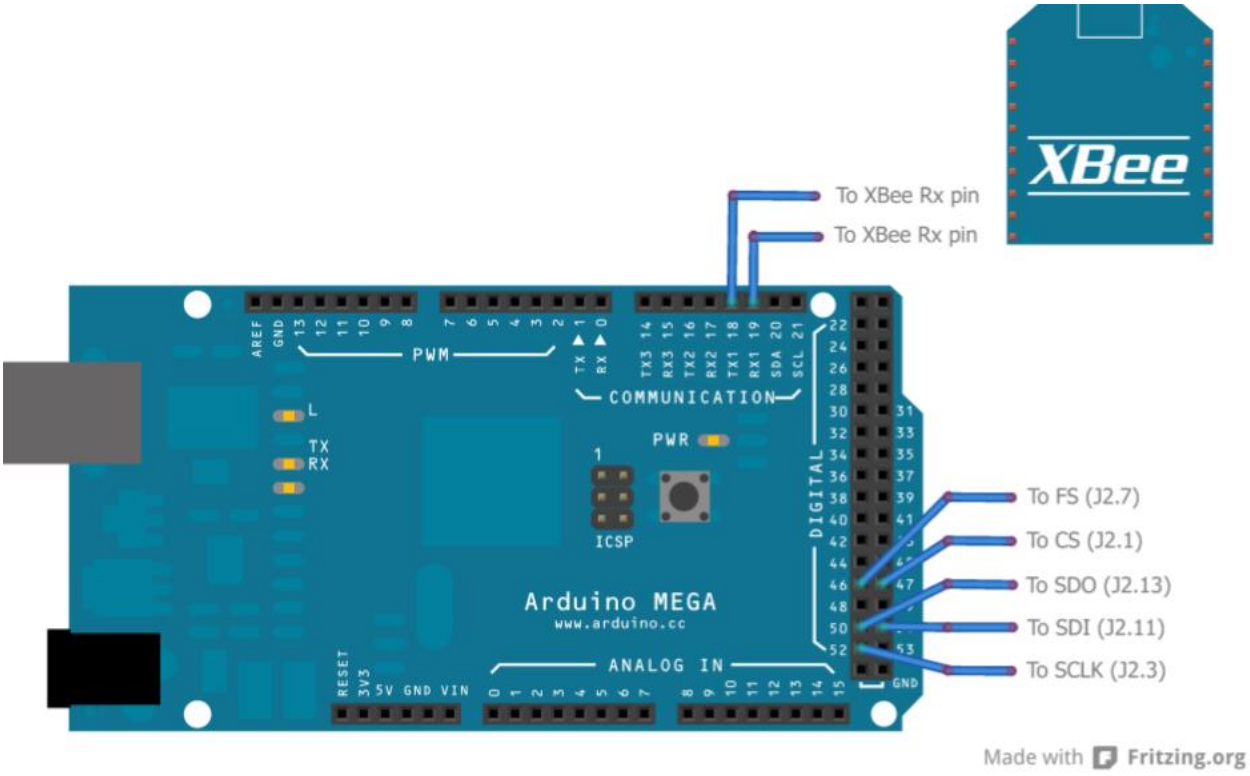
(i) Capture a fixed number of samples at a high sampling rate.

(ii) Send the captured samples to PC using XBee. During this time, no samples are captured. This step takes a lot of time compared to (i).

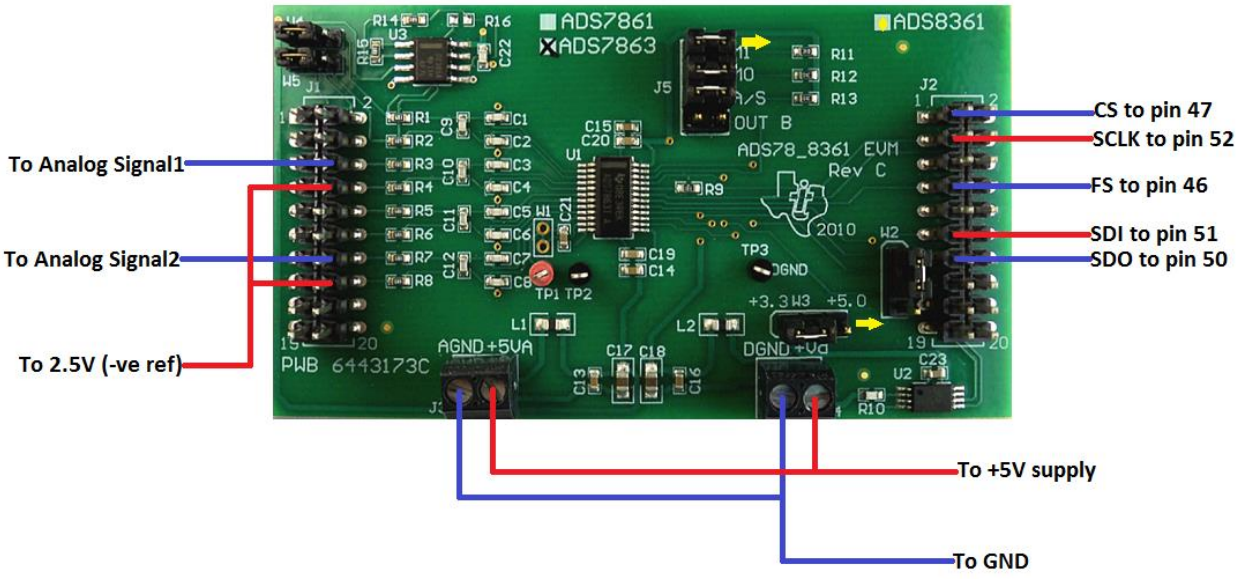
The connections are made as shown in the diagram below. The ***FS*** *(Frame Sync) pin* on the *ADS8361 EVM board* is the same as ***CONVST*** *(Conversion Start)* and ***RD*** *(Read) pins* (*CONVST* and *RD* pins are shorted by default). Refer to the datasheet of *ADS8361* and of *ADS8361 EVM board* for more details:

ADS8361 datasheet: http://www.ti.com/lit/ds/sbas230e/sbas230e.pdf

ADS8361 EVM board datasheet: http://www.ti.com/lit/ug/slau094a/slau094a.pdf



Connections to be made between Arduino Mega and the ADS8361 EVM board



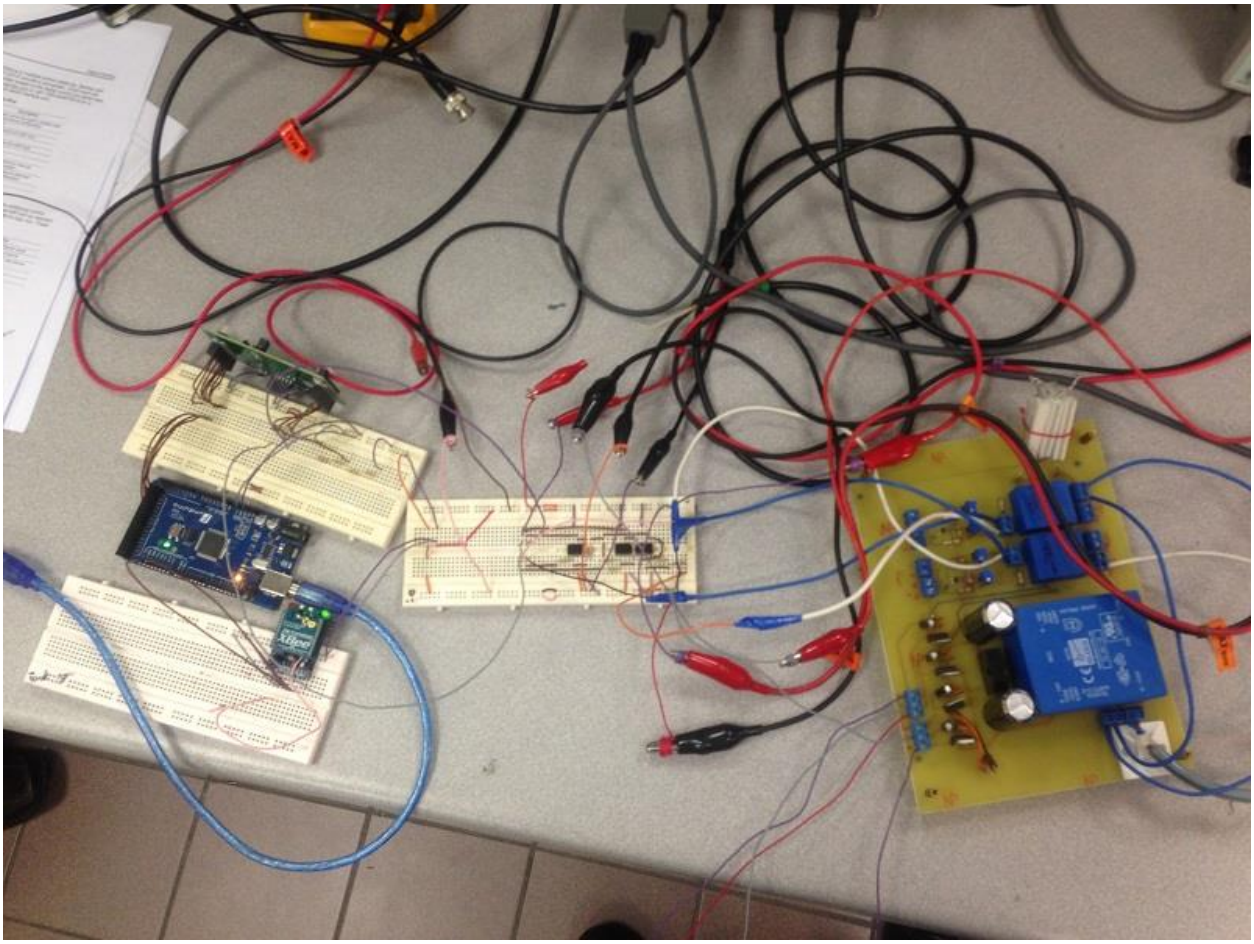
Connections to be made to ADS8361 EVM board

Now use jumper ***J5*** to configure the ADC board to work in four channel mode (though we will be using only two channels):

M0 = 1; M1 = 0; A0 = (0 / 1)

Now the hardware is all set and we need to proceed to programming. Find more information about the SPI protocol here: http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus

Regarding the ***Clock polarity (CPOL) and phase (CPHA)***, the *ADS8361* works on ***CPOL = 0*** and ***CPHA = 0***, i.e., ***SPI Mode 0***. The supplied clock frequency to ***SCLK pin*** of *ADS8361* should be *less than 10MHz* and the *maximum achievable sampling rate is SCLK / 20*. For our experiment, we plan to use a sampling rate of *12.5ksps* and hence we require a frequency of *250kHz* to *SCLK*. Since *Arduino Mega 2560* has a clock frequency of *16MHz*, we require a clock dividing factor of *64*.



Experimental setup for sensing voltage and current used on a socket using the voltage and current sensing board along with the ADS8361 EVM board, signal conditioning circuit and Arduino

Now, to read a sample from the ADC, the following steps were followed:

(i) Set ***FS*** pin to ***high (1)***. This tells the ADS8361 that you want to start ADC conversion and read the sample.

(ii) Transfer some random value to the ADC and obtain a byte of data. The *lower 4 bits (b3 – b0*) of this byte are *upper 4 bits (b15 – b12)* of the *16 bit sample*.

(iii) Set ***FS*** pin to ***low (0)***. The FS needs to be a pulse and not always on.

(iv) Transfer some random value to the ADC and obtain a byte of data. All *bits (b7 – b0)* of this byte are *middle 8 bits (b11 – b4)* of the sample.

(v) Again, transfer some random value to the ADC and obtain a byte of data. The *upper 4 bits (b7 – b4)* of this byte are *lower 4 bits (b3 – b0)* of the sample.

Quick Reference for configuring and running the ADS8361

Configuring ADC Operation:

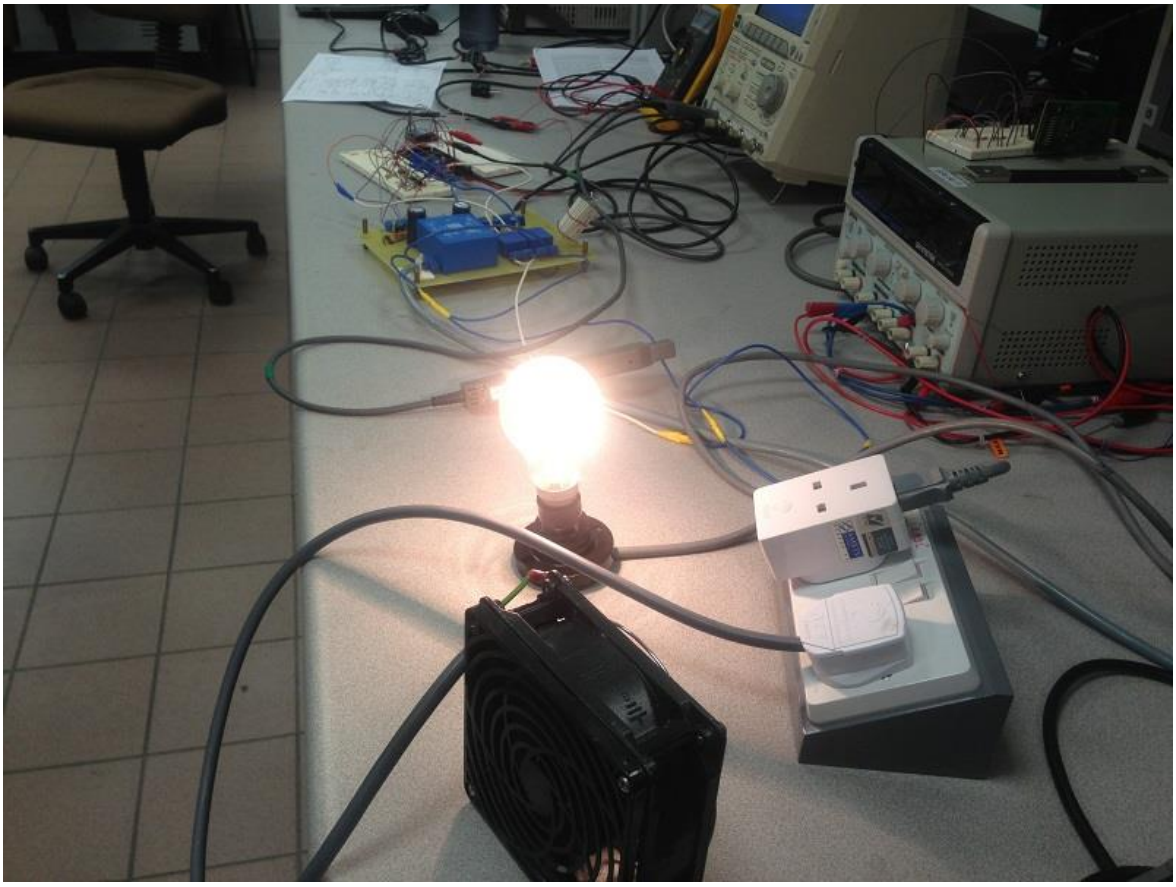
Jumper	*J5*
Mode	*4 channel*
M0	*1*
M1	*0*
A0	*(0 / 1)*

SPI Mode:

CPOL (Clock polarity)	*0*
CPHA (Clock Phase)	*0*
SPI mode	*0*

Reading a sample from ADS8361:

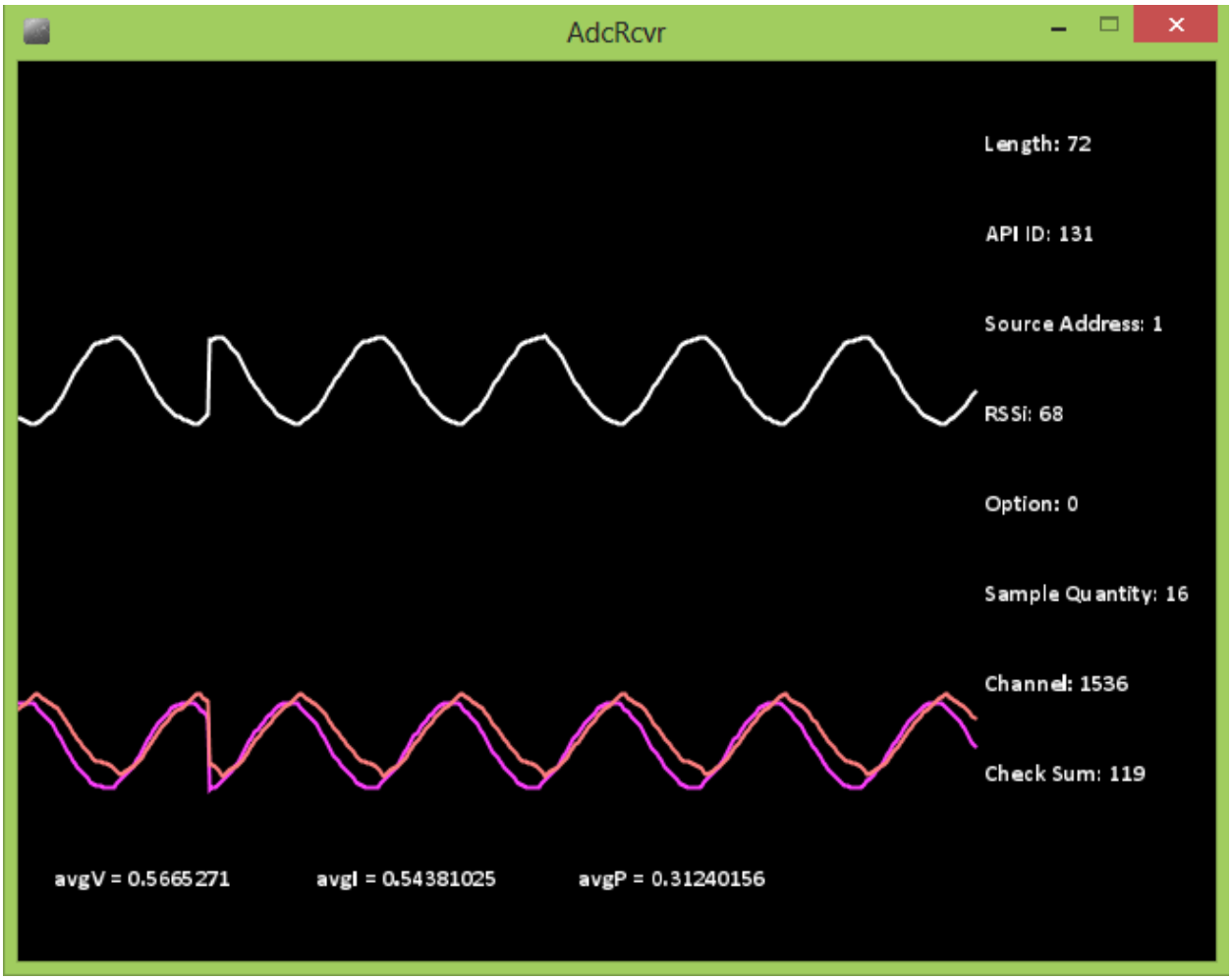
- Set ***FS*** pin to ***high (1).*** Start ADC conversion and read the sample.
- Transfer some random value to the ADC and obtain a byte of data. The *lower 4 bits (b3 – b0*) of this byte are *upper 4 bits (b15 – b12)* of the *16 bit sample*.
- Set ***FS*** pin to ***low (0)***. The FS needs to be a pulse and not always on.
- Transfer some random value to the ADC and obtain a byte of data. All *bits (b7 – b0)* of this byte are *middle 8 bits (b11 – b4)* of the sample.
- Again, transfer some random value to the ADC and obtain a byte of data. The *upper 4 bits (b7 – b4)* of this byte are *lower 4 bits (b3 – b0)* of the sample.



Loads connected to the socket of the Voltage and Current sensing circuit

However, the sample provided by *ADS8361* was level shifted by half of full magnitude. To correct this error, if captured sample was greater than *0x8000*, then *0x8000* was subtracted from it, else *0x8000* was added to the sample. This fixed the problem.

We need *simultaneous sampling* of two different channels. The *ADS8361* does that, but since we have *only one SPI port* in *Arduino Mega 2560*, we can obtain simultaneously sampled data from two different channels from the same SPI port sequentially. The first ADC sample read would be from channel A0 and the second ADC sample read would be from channel B0.



Received and decoded voltage and current sensing output received on PC using the developed ADC receiver program (using Processing) at around 12.5ksps

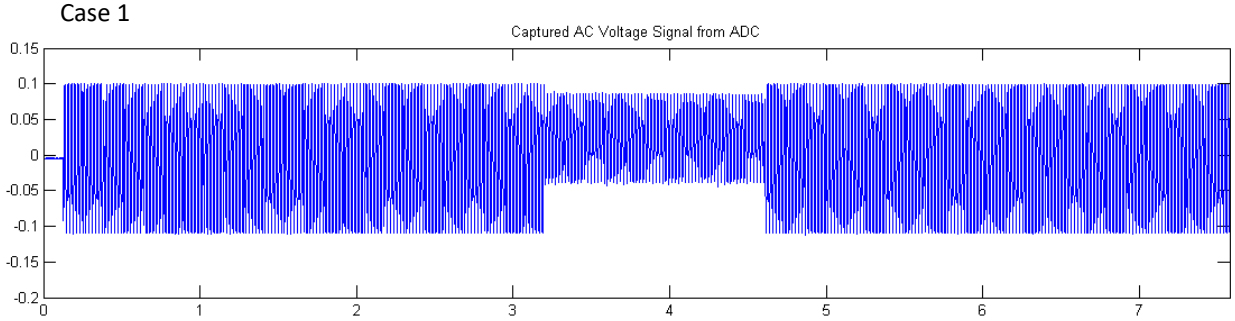
The *ADC SPI program* for this experiment is uploaded at:

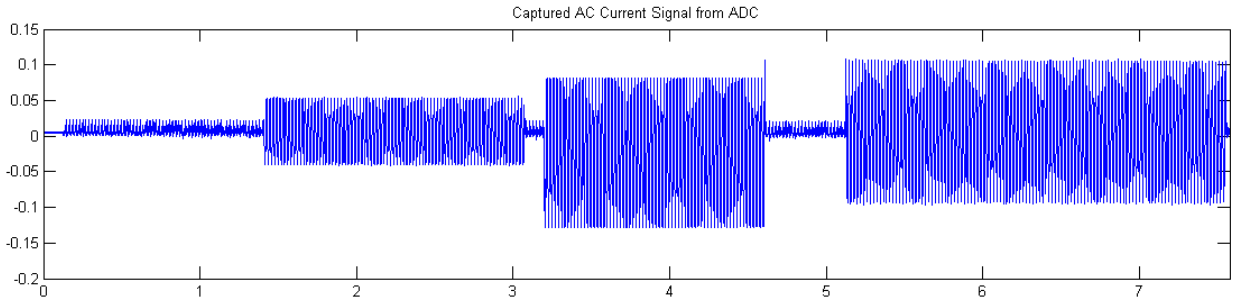
https://github.com/wolfram77/SmartGrid/tree/master/Core/zOld/Xbee_Ard/AdcSpi

As mentioned before, the *ADC SPI program* captures a fixed number of samples from ADC, and then sends it to the PC using *XBee*. The *ADC Receiver program* that runs on the PC has been uploaded at:

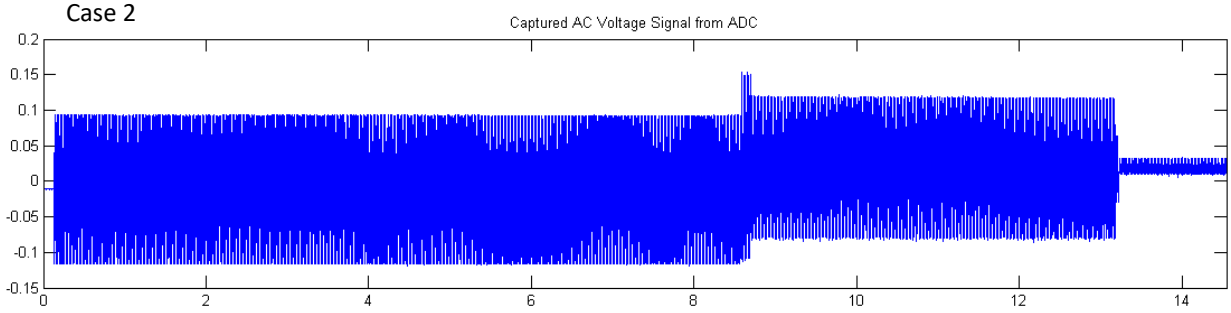
https://github.com/wolfram77/SmartGrid/tree/master/Core/zOld/Xbee_PC/AdcRcvr

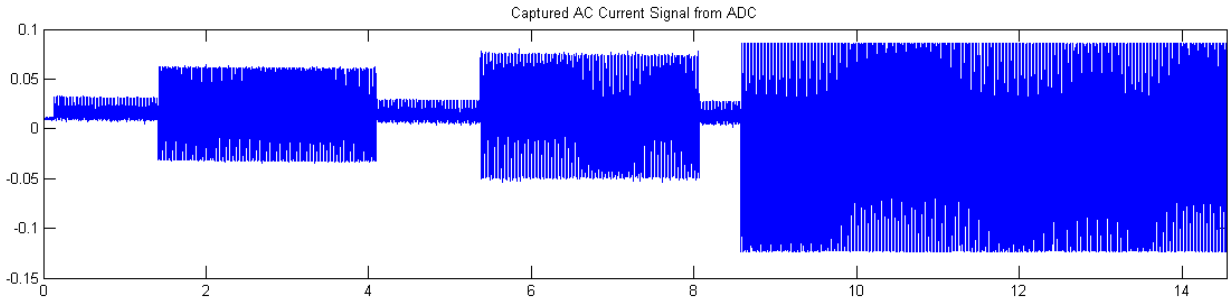
Case 1





Case 2





AC Supply voltage and current with varying loads (off / bulb / fan / both) with time with around 30,000 samples

The voltage and current values are obtained from voltage transducer and current transducer used in the previous experiment. The *ADC SPI program runs once*, after which it stops. The *ADC Receiver program records the received data as floating point values to two text files*.

These text files were opened in *MATLAB* to find out the waveform of the voltage and current. The *MATLAB program* has been uploaded at:

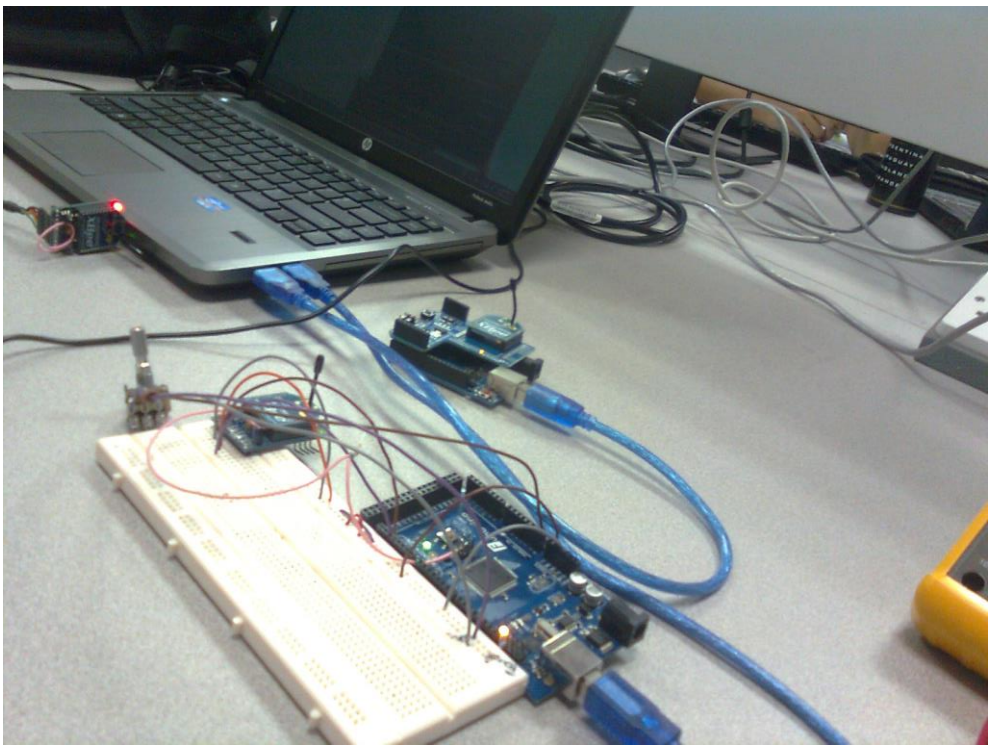
https://github.com/wolfram77/SmartGrid/blob/master/Core/zOld/Matlab

All outputs obtained are shown above. This completes this experiment.

8. MEASURING SIGNALS FROM MULTIPLE SENSOR NODES

In a Wireless Sensor Networks, captured signals from sensors need to be received to the main server from multiple wireless sensors. In this experiment, we have tried out, multiple signal transmission with two Arduino (sensor) modules sending data to one XBee module (to PC) directly, but with each having a different source id and ADC channels. The following are the observations when the XBee modules are used in transparent mode:

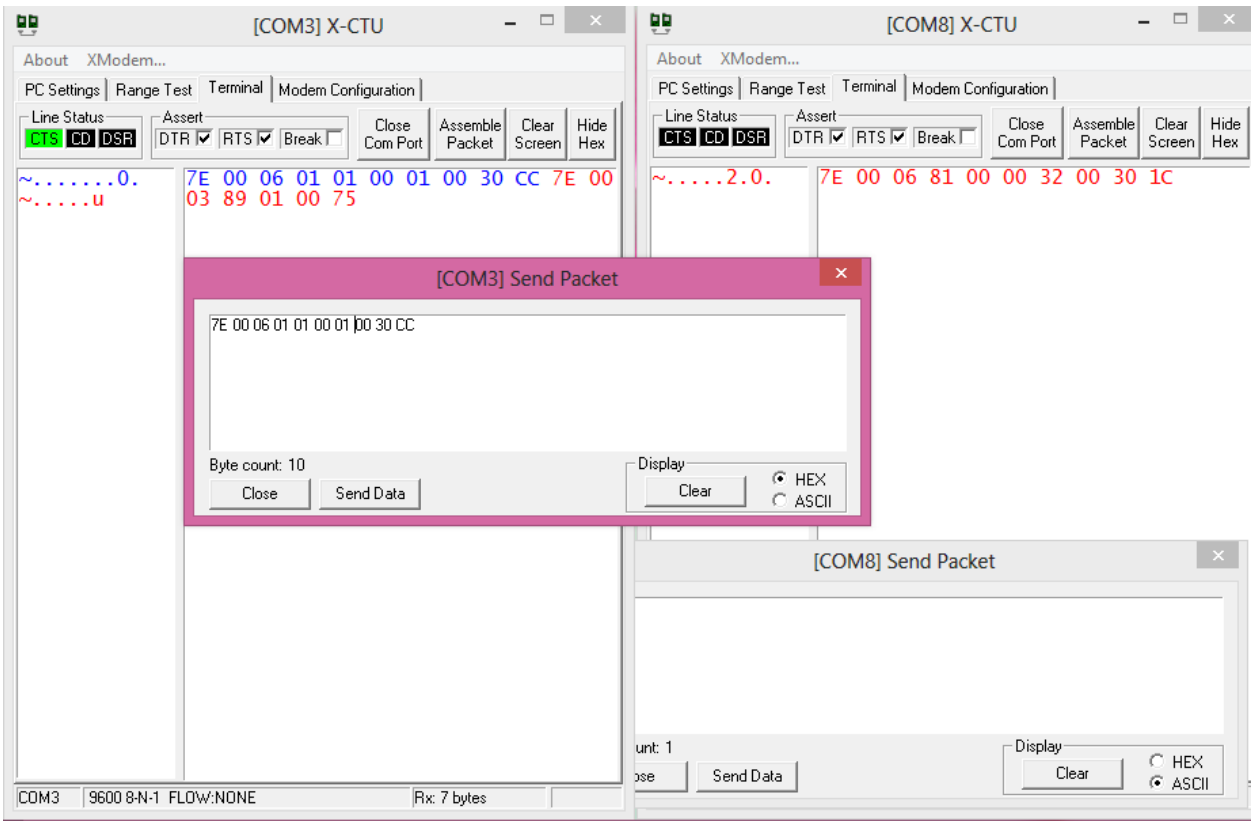
1. When ADC Receiver program and the two Arduino (sensor) modules are started, at a time, a signal from only one of the two sensor modules is received.
2. At some random time, the received signal gets switched from one sensor module to another.
3. The point of switching is not a proper transition, i.e., when switching occurs, it occurs quick enough that within a small period of time, the received signal gets switched several times before the final switch.
4. Within a few seconds, switching occurs in the middle of a packet, thus corrupting the received packet, and causing the ADC Receiver program running on the PC to halt with error.



Experimental setup for capturing analog signals from 2 different nodes and display it on the PC

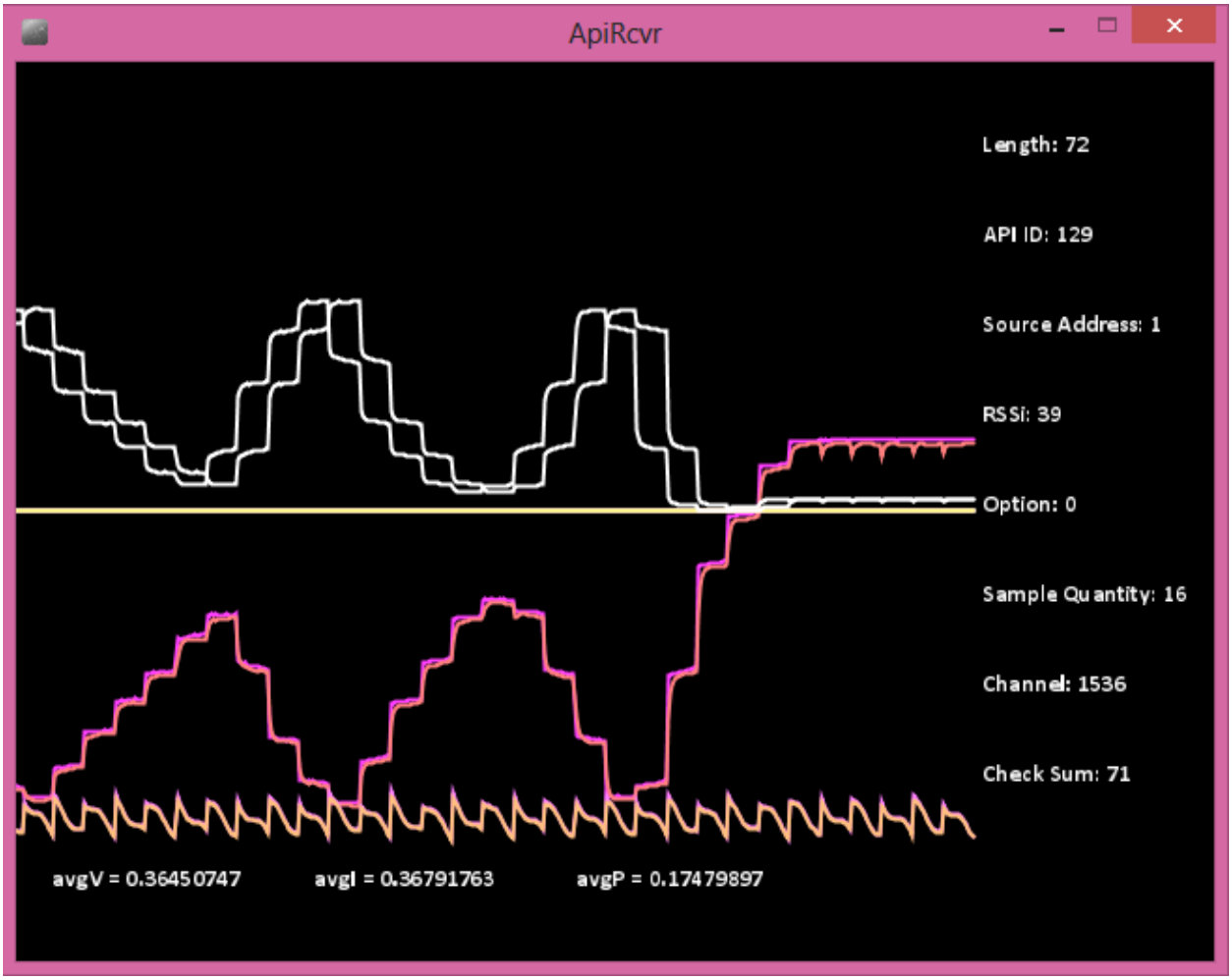
The following could be possible reasons of this kind of observation:

1. When one sensor module is transmitting, the other module usually detects this and does not transmit its own data. This also causes the non-transmitting XBee to lose data (due to its small internal buffer, which is around 200 bytes), and thus may produce corrupt packets in the near future.
2. Once the non-transmitting sensor module detects a free channel it starts its own transmission. However, it is very likely that the other XBee is also doing the same thing. Hence, they end up fighting with each other for RF transmission, and finally the one who transmits a little ahead of the other wins. This is probably why transition is not proper.
3. Though the chances of switching in the middle of a packet are less, it is possible, and thus it happens, and when it does happen, the ADC Receiver running on the PC side crashes.



XBee API mode is being used to send data from one XBee module to another. This mode allows data to be transmitted in form of RF packets, as well as provides additional information such as packet reception status. The reception status can be seen on the COM3 X-CTU window (left) which is shown in red color.

XBee supports a mode of data transfer called the API mode. Unlike the transparent mode, which simply acts as a serial line replacement, the API mode is a well-structured format of data exchange used in XBee. When using the API mode, data is transmitted in RF packets. These packets are single unit of data, and are not broken down by XBee during transmission. Hence, it helps avoid in-packet switching. It also features a response packet like in TCP/IP. When using the API mode, it is possible to send AT commands directly to the serial connected XBee, or to a remote XBee without entering into command mode. This can be very useful in automating XBee configuration. We may also connect CTS to Arduino in order to check buffer status of XBee so that we do not destroy any data in the XBee internal buffer. We could send data at random time slots and at reduced data rates to avoid clash.



Four signals being received from 2 nodes and being displayed on the PC. The PC is connected to one XBee module and receives data from the nodes in alternating fashion.

Accordingly, an ADC reading and transmitting program is written for both Arduino boards. Both the sensor boards send data to the PC in 200ms interval. They do this in a repeatedly, and they do not clash. The received signals are displayed on the PC using the same ADC receiver program.

ADC Reading program for Arduino:

https://github.com/wolfram77/SmartGrid/tree/master/Core/Level5/AdcSnsr_Mega

https://github.com/wolfram77/SmartGrid/tree/master/Core/Level5/AdcSnsr_Uno

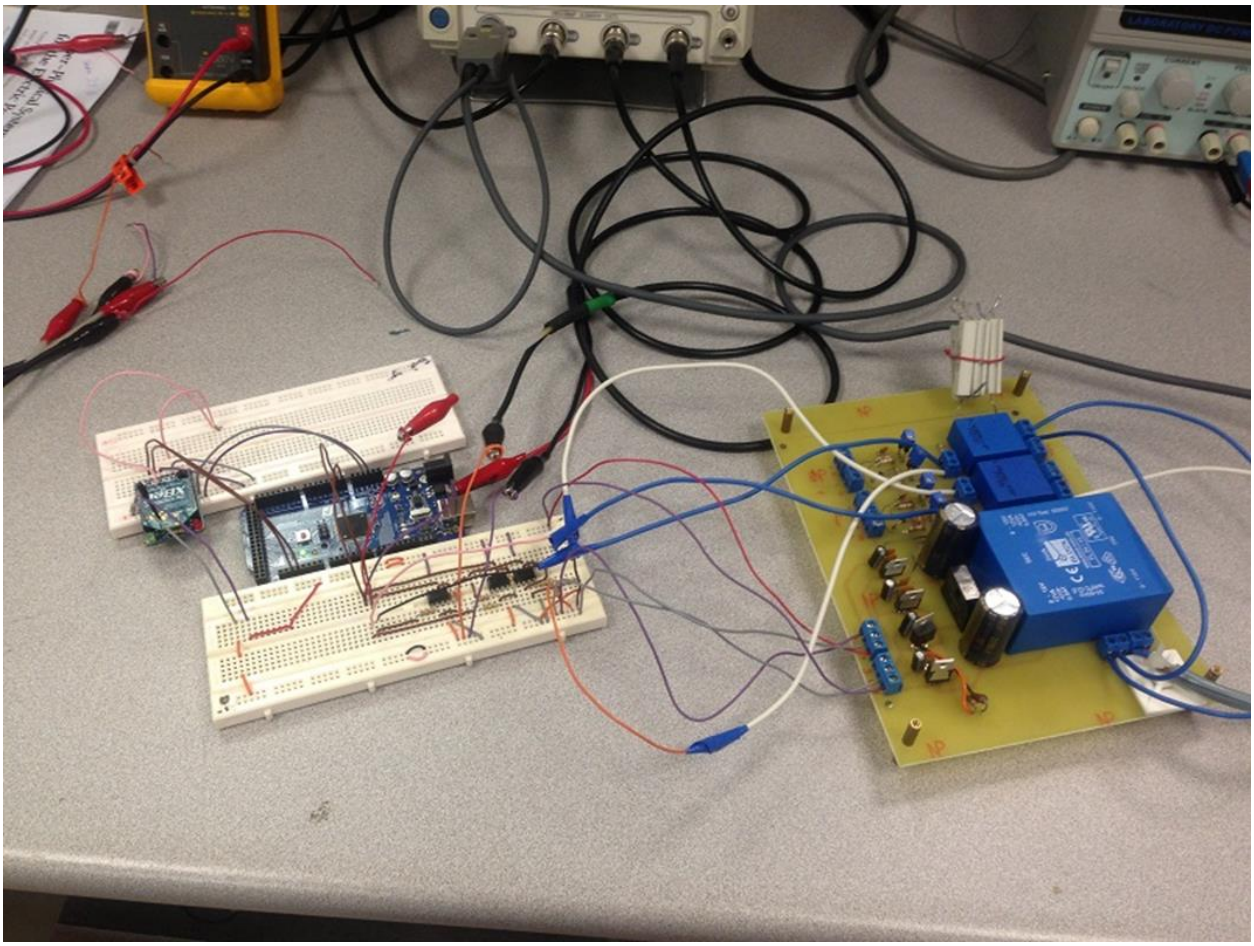
ADC Receiver program for PC:

https://github.com/wolfram77/SmartGrid/tree/master/Core/zOld/Xbee_PC/ApiRcvr

All four transmitted signals are received (two signals from each sensor board). This completes this experiment.

9. OBTAINING SPECTRUM OF VOLTAGE AND CURRENT IN MATLAB

We were interested to obtain a proper spectrum of voltage and current captured from voltage and current transducer. The voltage and current transducer are connected to a socket, across which voltage and current are measured. Here, we have tried to capture the voltage and current samples from the transducers using in-built ADC in Arduino Mega 2560 board and send it over to the base station (PC) using Xbee S1 modules, which are connected to both the microcontroller board, as well as the base station. We then write a code in "Processing" to capture the received data from the Serial port, and dump it to a text file. The captured samples in the text file are then analyzed using MATLAB, where, both the voltage and current samples are observed in a figure window, both in time domain and in frequency domain.

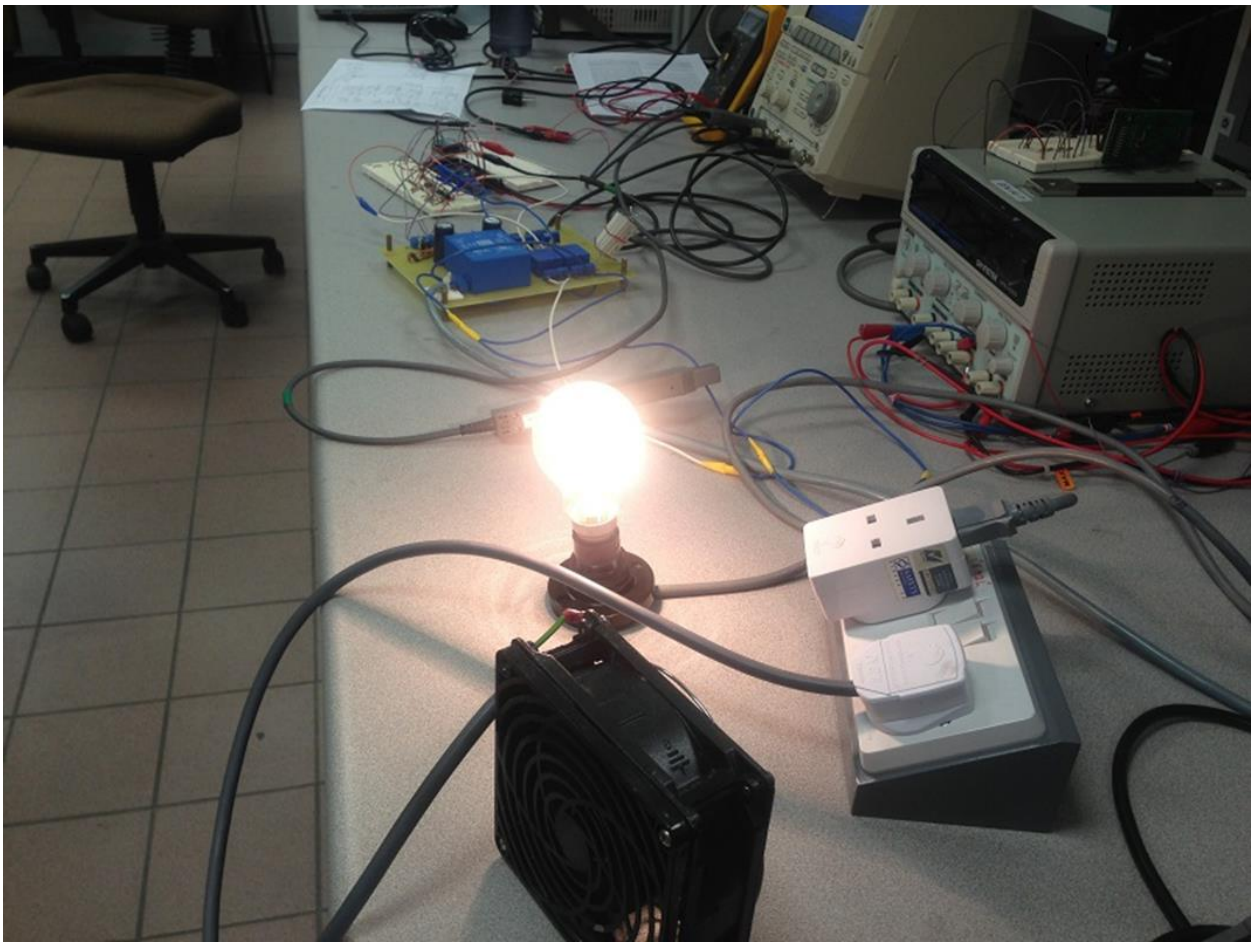


Experimental setup for sensing voltage and current used on a socket using the voltage and current sensing board along with the signal conditioning circuit and Arduino. The captured samples were sent to through Xbee to a base station connected to Xbee which received the samples and stored it in a text file. The samples stored in the text file was then used in MATLAB to obtain the spectrum of voltage and current samples captured.

Two different loads were connected to the socket (which was being sensed):

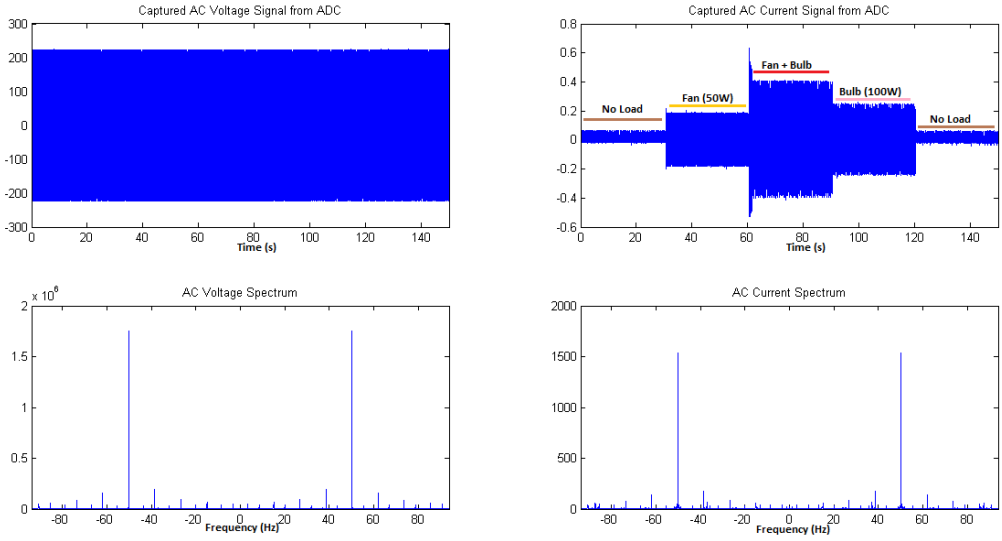
1. CPU Fan (50W)
2. Bulb (100W)

The experiment was originally started with no load. After 30s, the Fan was turned on, while the sensing continued. Then the Bulb was turned on after 30s. After that, the fan was turned off, and this was continued for 30s. Then finally, the Bulb was turned off (no load now), and this state was kept for 30s. The total capture duration was of 2 minutes, 30 seconds. All measured times were measured with a stopwatch, and may have ±0.1s error due to human factors. The microcontroller was programmed to have a sampling rate of 200 samples per second, but from the number of samples actually captured, divided by the time of capture gave a sampling rate of around 188 samples per second (this could be due to delay introduced while transmitting samples).



Loads connected to the socket (which was connected to voltage and current transducer) were changed (turned on and off) in certain amount of time. A bulb and a CPU fan (shown in this picture) were used as loads, which were turned on and off). This was done in order to see the effect of harmonics introduced in the supply when loads were changed. The voltage and current transducer board which captures the signals is on the top of the picture.

The samples that were sent by the microcontroller (using Xbee S1 module) to the base station (PC, with Xbee S1 module connected to USB port) were captured by a program written in “Processing” which dumped the samples to two text files (one for voltage samples, and the other for current samples), which could later be easily opened in MATLAB.



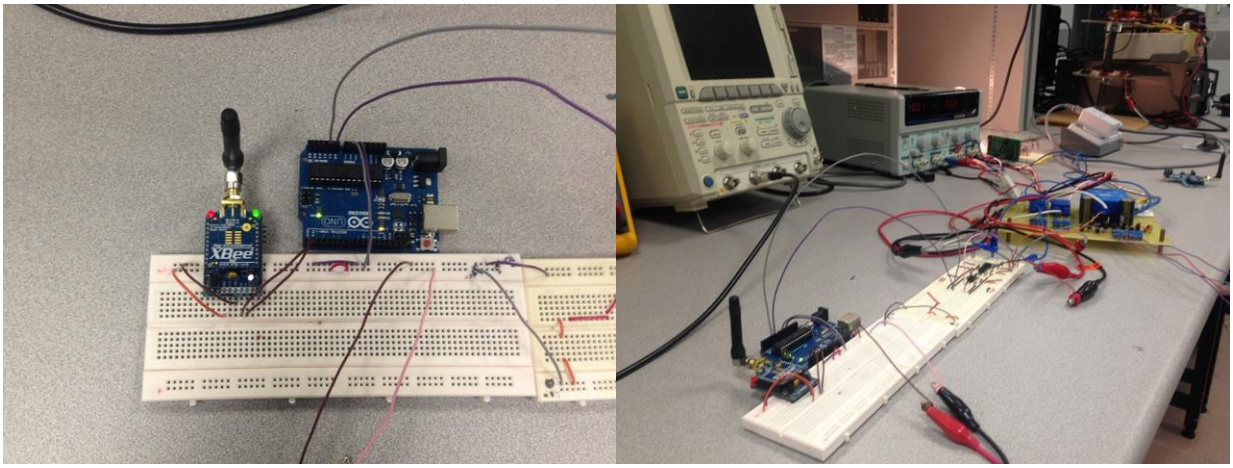
The AC voltage and current samples that were captured by the microcontroller plotted in this figure, using MATLAB. DC Offset in the signals were removed and they were scaled to appropriate values. The spectrum of both the signals was then plotted to observe the frequencies of noise introduced in the waveforms due to load switching.

Finally, in MATLAB, we loaded the samples from both the text files. We observed that both the signals had some DC offset. The DC offset was removed (by subtracting the average from the signals), and then the signals were multiplied with appropriate scaling factor to obtain voltage and current signals at proper levels. We then used FFT with FFT-shift to obtain the spectrum of both voltage and current samples, and plotted them on a single figure.

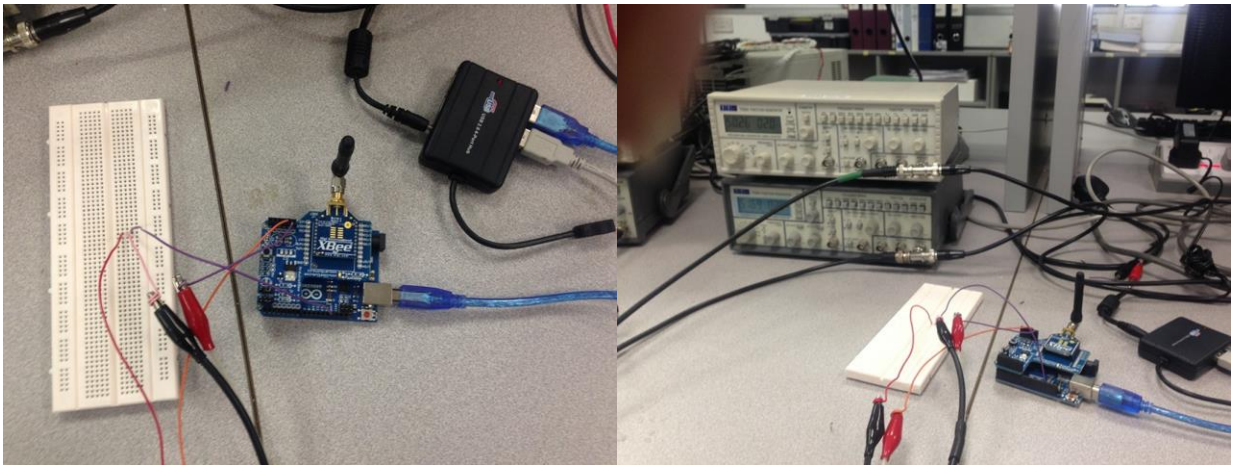
In the time-domain representations, we were clearly able to see that the voltage at the socket remained approximately constant, and the current varied with varying loads. We also observed the spectrum of both voltage and current which displayed a clear peak at 50Hz. However, we also observed the presence of noise in other frequencies in both voltage, as well as current. This finishes the experiment.

10. MEASURING VOLTAGE AND CURRENT FROM MULTIPLE SENSOR NODES

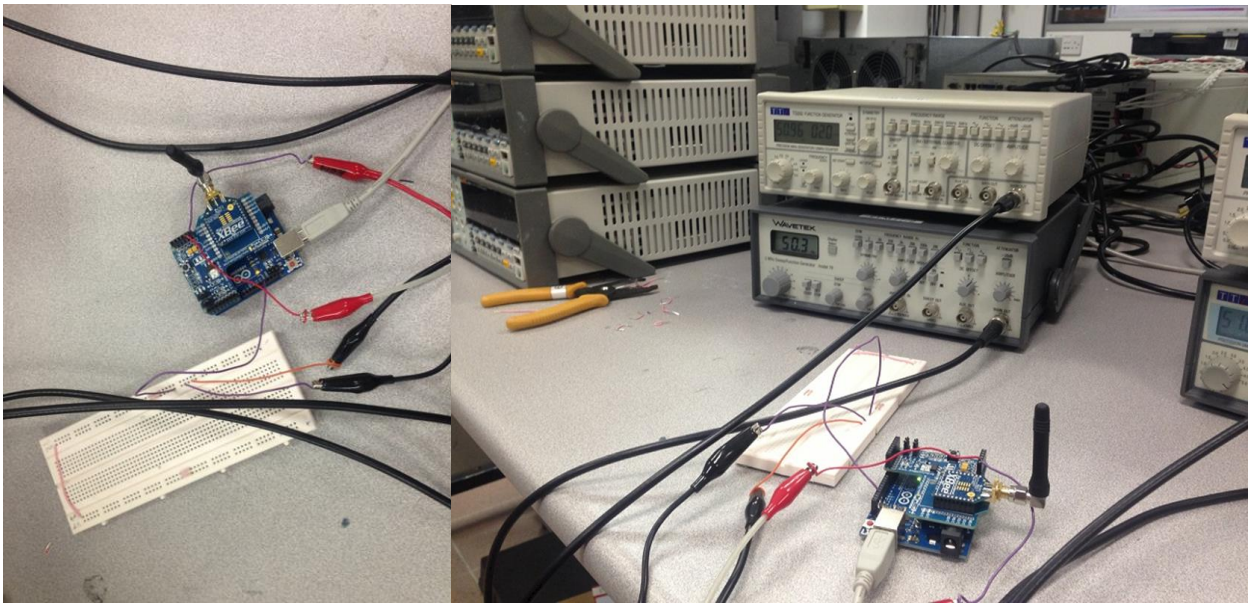
Here we prepare for measuring voltage and current sensor data from multiple sensor nodes directly linked to the base station. We use 3 Arduino Uno boards with Xbee Shield connected to Xbee S2 module. The base station is attached to an Xbee S2 module using an FTDI board for connecting to Xbee modules. One of the Arduino Uno boards (node 1) is connected to actual voltage and current sensors (which measure voltage and current across a socket), and the other two boards (node 2 and 3) are connected to a pair of function generators, each.



Node 1 using Arduino Uno and Xbee S2 wireless board is capturing supply current and voltage from voltage and current transducers connected to a socket which powers a bulb and a CPU fan (not seen in picture)

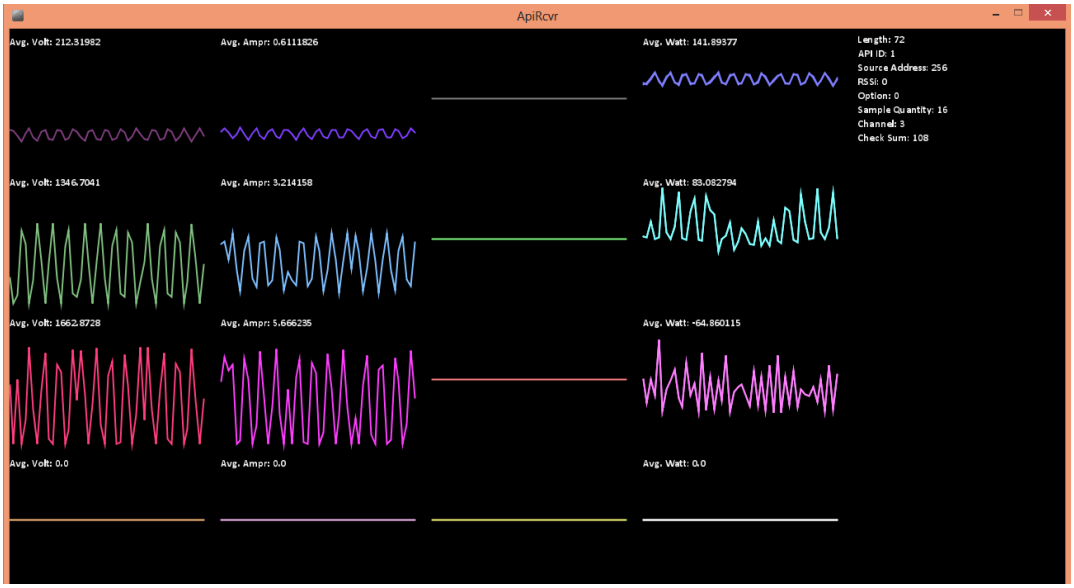


Node 2 with the same configuration, but is capturing signals from 2 function generators instead of voltage and current transducer



Node 3 with the same configuration, and is capturing signals from 2 other function generators instead of voltage and current transducer

Each node sends data to the base station in a predetermined packet format (but, the Xbee modules are configured to work in AT mode or transparent mode). The packet format used is the same as Xbee ADC format (which Xbee modules transmit when configured to record ADC data at any of its pins). All nodes capture conditioned voltage and current sensor samples (using a signal conditioning circuit designed before) using internal ADC at a sampling rate of 200 samples per second. Every 16 samples of voltage and current are formed into a packet, and sent to the base station.



Captured voltage and current signals captured from Node 1, 2 and 3 (at the base station connected to another Xbee S2 module) respectively from top to bottom. The graphs starting from the left to right denote received voltage, current and calculated power signals. In each case, the calculated average of each quantity is shown on the top of the graph. The voltage and current samples were received at a rate of 188 samples per second. This plot was displayed by capturing the received samples (which was sent in Xbee ADC packet format) and then displaying it using Processing. Each node sent its own identity information along with the samples so that they could be distinguished from samples from other nodes.

Each node is programmed separately to send data over a different channel (same as the one used in Xbee ADC format). This helps the base station distinguish between different signals. Instantaneous power is calculated from the received voltage and current samples with simple multiplication at the base station. Average voltage, current and power is calculated and shown above the respective plots. The base station program is written in Processing.

The plots obtained above are shown. Note that, the second and third plots are captured from a signal generator, and hence their average voltage and current and very different from the first one.