Control System of Quantum Degenerate Gases Laboratory

Quantum Degenerate Gases Laboratory
University of British Columbia

Sanaz Fotoohi

76152040

Dr. Kirk Madison
UBC Department of Physics and Astronomy

May 7, 2006

Abstract

The primary focus of the research at Quantum Degenerate Gases Laboratory at UBC is on the applications of ultra-cold gases to the physics of many-body quantum systems. The experiments in the lab run by a computer control system. Building the control system can be broken up into projects such as Fast UT Bus, DDS System, Laser Lock Box, Magnetic Coil Drivers, 2W RF Amplifiers, Photo-diode Amplifiers, Digital and Analog Output Boards, and Ethernut to GPIB Interface. An overview of the experimental control system of QDG lab can be found on http://www.phas.ubc.ca/~qdg/ControlSystem/. Two important projects of the control system are the DDS and GPIB-Ethernut. The DDS is a radio frequency generator that is controlled by a software called TestDriveApp. The GPIB-Ethernut is a GPIB controller box that allows the user to control multiple instruments. These two projects were originally started by Raymond Gao, the former COOP student. In this report I explain the changes that I've done to the DDS Test Drive and the GPIB control system to complete these projects. I also spent some time on a more technical mini project, building a power supply for a Picomotor.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The experiments in Quantum Degenerate Gases Laboratory include complex electronics which need to be monitored by a complicated control system. The control system communicates with the devices through a 50-pin digital bus. Previous work has been done to develop the computer control system of QDG lab at UBC. Two most important projects of this system are the Direct Digital Synthesizer and the GPIB-Ethernut. This report summarizes the details of the work I have done on each project and the codes that I implemented for the control system. The report also explains the changes that I made to a power supply so that it will regulate the outputs properly.

Chapter 2

Computer Control System for Atomic Physics Experiment

The group of Prof. Mark Raizen at the University of Texas at Austin have developed a powerful, easy to construct experiment control system to communicate with different laboratory electronics. The hardware side of this control system includes a design for devices like Digital to Analog Convertor (16-bit DACs), Analog to Digital Convertor (ADC), and a Direct Digital Synthesis device (DDS) on a general purpose parallel bus which is easy to expand upon and interface with a variety of devices. The hardware use BNC-type outputs and line drive capacity to connect the devices under computer control to the real experimental devices. The communication line is a parallel 16-bit bidirectional data bus, parallel 8-bit address bus, and a strobe signal. The software is intended for cold atom experiments and contains the code to talk with the developed hardware. The advantage of this control system is that these devices are cheap and easy to use. Figure 2.1 gives an overview of the system. The set up uses a National Instrument Board (NI6533).

In Figure 2.1 the NI card uses a special cable which is connected to an adaptor board A. The board will sort out the signals and divide them into 8 bit flat ribbon connecters. Cable B combines these four 8 bit ribbon cables in one 25 bit and one 7 bit flat ribbon cable. Part C is the strobe bit generator and transceiver board. It buffers the NI6533 card signals in addition to producing a time shortened strobe bit signal. Part D is the bus system 50 line flat ribbon cable. Part Ex is the 16-bit devices.

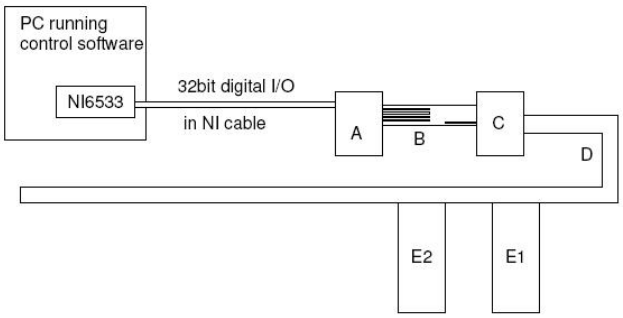


Figure 2.1: General Layout of the Control System

The communication line in this set up consists of a 16-bit data bus, an 8-bit address bus, and a clock which is converted into a strobe signal. When the input 8-bit address bus matches that of the DIP switches on the Printed Circuit Board, the device accepts the strobe signal. Therefore we are able to connect 2^8 devices to the ribbon cable. The data loading is accomplished in three clock cycles: data input, strobe on, strobe off. That is why it is important to have a strobe line.

The bus bits are the computer output. The computer can create the output in different ways. In this case, we used a NI6533 card which outputs a synchronous 32-bit parallel bus and clock signal. You can see the pin configuration of the 50pin connector in Figure 2.2. As mentioned before this system has the flexibility to be connected to various devices one of which is the DDS.

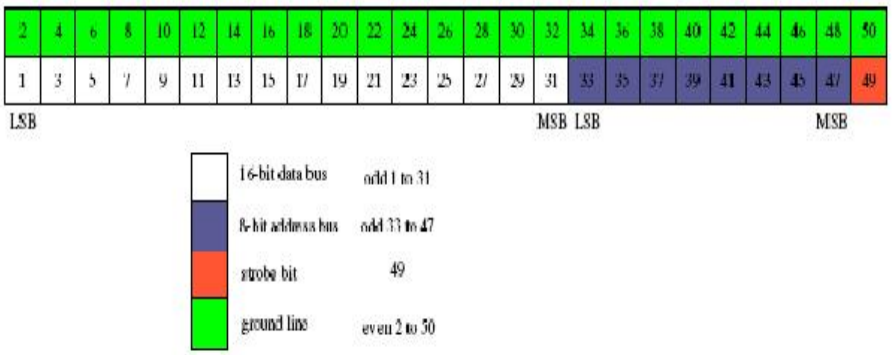


Figure 2.2: Pin Configuration for 50-pin Connector

Chapter 3

DDS

3.1 Direct Digital Synthesizer

DDS is a Direct Digital Synthesis device that produces radio frequency signal between DC and 135MHz. The most important part of the design is an AD9852 chip from Analog Devices. This chip is a highly flexible device and has different features such as a programmable reference clock multiplier, an inverse sinc filter, a digital multiplier, two 12-bit/300 MHz DACs, a high speed analog comparator, and an interface logic.

The digital inputs to the board consist of a 24-bit bus and a strobe bit. The first 16 bits carry the data and the next 8 bits carry the address. The 16 bit data bus is broken into two parts. The first 2 bits of the bus determine the status of the strobe, whether it is high or low. The higher 6 bits of the bus determine the address of the board. If they do match with the address of the DIP switches on the board, the strobe and, as a result, the data is accepted; in other words the data is latched.

All different functionalities of the DDS originate from the various features of the chip on board. Programming the chip, we are able to use these functionalities. There are five programmable modes of operation of AD9852: Single tone mode, Unramped FSK, Ramped FSK, Chirp, and BSK mode. We used three of these modes: Single tone mode, Unramped FSK, and Ramped FSK. For further information on each mode and how to program it, refer to AD9852 data sheet on http://www.alldatasheet.com/.

3.2 How Does the DDS Work?

To understand how the DDS works, we first need to know how the AD9852 works. The board that I describe here is based on the Version 2 schematics of the DDS.

First in order for programming changes to be transferred from the I/O buffer registers to the active core of the DDS, a clock signal must be externally supplied to pin 20 or internally generated by the 32 bit Update Clock.

The clock for the DDS can be set up in one of three ways one of which is to use a single ended external clock which enters onto BNC J5A on the board. Then the clock multiplier of the DDS multiplies the signal by an integer between 4 and 20. Our design runs at a maximum of frequency 300 MHz.

The AD9852 has two high speed outputs. Output 1 is the RF cosine output and output 2 is an arbitrary control DAC. The output BNCs are J1 to J4 and J6. J1 and J2 are the filtered outputs. On version 2 of the board J1 is the Cos DAC and J4 is the control DAC. The output of the Cos DAC is a cosine wave and it drives the Cos DAC output of the DDS. The output of the control DAC is a straight DC voltage level on the oscilloscope and can provide DC control levels to external circuitry. One can change the amplitude of the cosine DAC via Shaped on/off keying. This feature controls the amplitude vs. time slope of the Cos DAC.

As mentioned before the DDS has several modes one of which is Ramped FSK. This method changes the frequency of the Cos DAC output from frequency1 to frequency2. It implies that many intermediate frequencies between F1 and F2 can be the output in addition to the former frequencies. This functionality is controlled via BNC output J8 on the board. You can test this by setting this output to high or low (+5 or 0 V) and view the changes on the scope.

There are many other functionalities on the board that can be activated by soldering on jumpers (0 ohm resistors) labeled W on the schematic For now Cos DAC, control DAC, Frequency Shaped Keying, and Ramped FSK are the ones that I test for each DDS board.

3.3 Programming the DDS

To program the DDS, we have to build a set of instructions based upon the values of the address bits and the data bits, write the values to the chips buffer, and move the data from the buffer to the internal registers. The different values of the bits for each functionality are found in the chip's data sheet on http://www.alldatasheet.com/datasheet-pdf/pdf/48612/AD/AD9852.html.

Raymond Gao, was the coop student who worked on this project in the first place and he implemented quite a few functionalities on the DDS. You can access his code on the QDG Admin account at C:\RayStuff. My job was to implement more functionalities of the chip and complete the Driver Test he created to test the DDS boards. The Driver Test Application tests three modes of operation of the chip: Single-Tone mode, Unramped FSK, and Ramped FSK. It is also capable of testing the Cos DAC and control DAC, changing the amplitude of the Cos DAC and the offset of the control DAC, and master resetting the whole device. The implemented code for these functionalities is saved on the Admin account at C:\Sanazstuff\RayStuff\NIDAQDDS\NIDAQDDS.dev. The code for the Driver Test Application itself is saved at C:\Sanazstuff\RayStuff\DriversTestApp\DriversTestApp.dev. Figure 3.1 is an overview of the Test Driver.

For programming instructions and further information on the values of bits and parallel addresses refer to Table 4 Register Layout, on the chip data sheet.

The most important steps for writing a DDS driver function are as follow:

1. Clear the buffer.
2. Check the input range.
3. Build the parallel address values where the data needs to be sent.
4. Build the data values_ based on the description of the data sheet each value creates a particular instruction.
5. Combine the address and the data to to make the 32 bit instruction value.

6. Add each and every instruction to the buffer_ one may need to use the "for" loop for this purpose.

7. Return the buffer size

Here is a sample code that accomplishes these seven steps:

```
 /* Controls the output amplitude vs. time slope of the cos DAC
    output signal*/
  int32_t DDSDriver::SetAmplitudeMultiplier(vector<int32_t>& buf,int16_t amp_mult){
  int i,j;
  buf.clear();

/* Checking the range of the input*/
  if(amp_mult < 0 || amp_mult > 0xfff)
      return 0;

/* Corresponds to the address of the board*/
  uint8_t addr = (address << 2) + LOAD_DATA;

/*Enabling the digital multiplier*/

  uint16_t data0 = ( MULT_CTRL_REG20 << 8) + MULT_PAR_ADDR20;
  uint32_t instruction = ( addr << 16) +  data0;
  buf.push_back(instruction);

/* Getting the last 8 digits of the input and sending it to register addr 22 */

  uint16_t data1 = ( (amp_mult & 0xff) << 8) + MULT_PAR_ADDR22;
  instruction = (addr << 16) + data1;
  cout << "to 22: " << instruction <<endl ;
```

```
    buf.push_back(instruction);

   /* Getting the first 4 digits of the input and sending it to register addr 21 */

    uint16_t data = ( amp_mult >> 8 ) & 0x0f;
    uint16_t data2 = (data << 8) + MULT_PAR_ADDR21;
    instruction = (addr << 16) + data2;
    cout << "to 21: " << instruction <<endl ;
    buf.push_back(instruction);

  return buf.size();
}
```

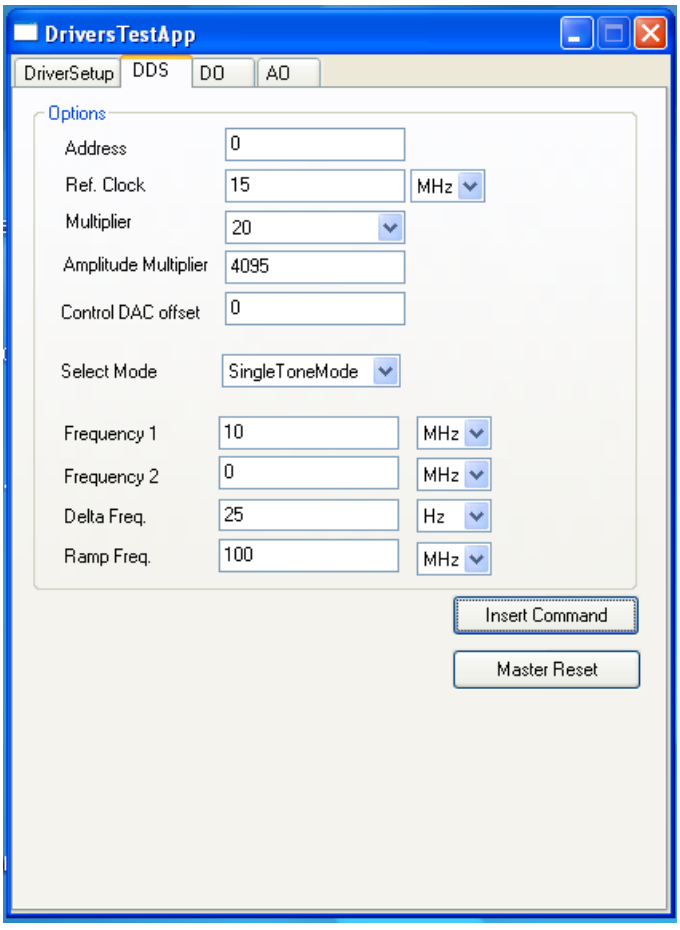


Figure 3.1: Overview of the Test Driver

3.4 Problems With the Second Version of the DDS Board

One of the main problems with Version 2 of the board was the high DC offset of both the Cos and control DAC output with respect to Version 1. We recently realized that all of the 50 ohm resistors on the boards have been misplaced with 50 Kohm resistors due to a mistake on the schematics. Once the E-shop corrected this mistake, the high DC offset of the outputs disappeared. Table 3.1 and 3.2 shows the status of a newly tested board.

DDS	Cos DAC DC offset	Control DAC DC offset	Peak-Peak voltage (mV) Mhz	Peak-Peak voltage at 100 Mhz	Peak-Peak voltage at 135 Mhz
Version 1	120mV	120mV	220 mV	150mV	64 mV
Version 2	100mV	120mV	180 mV	90mV	36 mV

Table 3.1: Version 1 of the DDS Board vs Version 2

Frequency 1(MHz)	Root Mean Square (mV)	Peak-Peak voltage (mV)
10	82.9	246
20	80.6	232
30	78.4	228
40	76.8	224
50	75.2	220
60	72.4	212
70	68.1	200
80	63.6	188
90	59.9	180
100	55.5	166
110	49.1	146
120	42.7	132
130	34.4	108
140	13.3	44
150	1.1	3.5

Table 3.2: RMS and Peak to Peak Voltage of the Cos DAC vs Frequency on DDS Version 2.0

In Table 3.1 both versions were running at 0.953 Amps and 5 Volts. As you see from the data points in Table 3.2, RMS and the Peak to Peak Voltage of the Cos DAC outputs are decreasing functions of frequency.

So far 18 boards have been built by the Electronic shop at UBC. The boards are tested and they work properly.

In case of any problems in the future design and testing of the DDS boards, you can contact the designer of the boards Todd Meyrath at tm4@pi4homer.physik.uni-stuttgart.de or Gabriel Price at gabriel@physics.utexas.edu.

Chapter 4

GPIB

The Ethernut-GPIB Interface Project is the alternative of purchasing expensive GPIB controller boxes. The GPIB controller allows the user to control multiple instruments such as Oscilloscope, Multimeter, OSA, and other electronic devices from a system controller like a base computer, run different experiments, and monitor their status. Raymond Gao has completed this project and he also wrote a thorough manual on how to use the GPIB board and connect it to the network in the lab. Most of the programming for this device has been done in C++ and Java so that it can run on both Windows and Linux.
Each device, connected to the GPIB, has its own manual that allows the user to program it so that new data and information can be recorded or sent. In this case I programmed an oscilloscope so that the user would be able to take up to 10000 data points from the output and save it to a file. I originally started programming the scope and the GPIB in Java as Ray started writing the code in Java, but finally we decided to reprogram both the GPIB and the scope in Labview because of some complications. Bruce Klappauf wrote a Labview program for connecting the GPIB to the network. Running that program along with my Java code, one can save the data points to a file and view the scope's image on the computer screen.

4.1 A Little about the GPIB

All the details here refer to Ethernut V2.1.
Please go through Chapter 3 of the GPIB manual to connect it to the network. To communicate with the Ethernut, we need a terminal emulation program. I used a Hyperterminal

for Windows. Go to Start/Programs/Accessories/Communication/Hyperterminal and start the terminal with the following settings:
Connect using: COM2
Bits per second: 38400
Data bits: 8
Parity: None
Stop bit: 1
Flow control: Xon/Xoff
Once you hit the reset button on the GPIB box, a message will be displayed on the terminal that contains the information regarding the Ethernut version and its IP address. The message looks like this:

```
Ethernut-GPIB 1.1.21 Server Started
Initializing Drivers...
Server Configured with Address: 172.16.1.100
<<INIT SEQ COMPLETE>>
```

This is how you determine the IP address of the Ethernut connected to your network. Also the GPIB instruments are running on port 15000. Whenever we run a program that uses the GPIB setting somehow, the IP address and the port number are the two important inputs that the user should always provide. Once you determined the IP address, searching for http://IP address on the network will lead you to the web page that communicates with the device connected to the GPIB; one has to skim over the device's manual (ie the scope) for a set of commands to talk to the device through the GPIB.

4.2 Java Code and Complications

The latest version of Java can be obtained from http://java.sun.com. Also on this web page you can find useful information on different classes and user interface programming. The Java Forum on this web site at http://forum.java.sun.com/index.jspa is really useful and I suggest you to join the forum and use it if you continue with coding in Java.
Although many believe Java is one of the more convenient programming languages as it

does not use pointers, it has many obstacles. First and foremost plotting in Java is not easy. There are methods in Java to plot data, but all the data points have to be converted to integers. The best choice is most likely to scale the data, ie multiply all the data points by a large number which might change from case to case depending on their range. In standard plots, the Y coordinate starts in the middle and is positive going up and negative going down. In Java, Y starts at the top and is positive going down. Another issue is that Java does not plot negative values. This requires changing all the data points; therefore plotting the data in another program such as Mathematica, Matlab or etc is recommended.
As a result a piece of code must be called from another program for plotting the data in Java. The program Jlink is a toolkit that integrates Mathematica and Java. It allows Mathematica programmers to load arbitrary Java classes into Mathematica, create objects of these classes, call methods and fields, and so on. It also allows Java programmers to launch Mathematica and use it as a computational engine. You can download Jlink at
http://www.wolfram.com/solutions/mathlink/jlink/.
One drawback of using Jlink to link the Java code to Mathematica is that one needs to have both Mathematica and Java on each computer on the network. One can download Java for free, but Mathematica needs a license. As a result it was decided that I start using the Labview program to program the GPIB from scratch. Bruce Klappauf implemented a basic code for displaying the image of the scope on the monitor. Using my Java code along with the Labview program one can read the data off the scope and save it to a file.

4.3 Getting the Waveform from the Oscilloscope TDS3014

I implemented the Java code GETWFMCH.java to get the waveform displayed on the scope. The file is saved in the directory C:\Sanazstuff\GPIB\javafiles. For this piece of code to be compiled, one needs the functions in EGInterface.java. These are the functions that talk to the device connected to the GPIB through the GPIB. The functions are : ibWrite, ibRead, ibConnectGPIBAddress, ibFindInstrument, ibCloseConnection, and etc. For further information on each function, please read Ray's manual, especially refer to Table 3.3 in Ethernut-GPIB Manual. The EGInterface.java file is included in the same directory as

GETWFMCH.java.
To compile a Java code, just type:

```
javac ‘‘name of the file’’.java
```

as Javac is your compiler. Once you compiled the code, the “name of the file”.class files are generated.
The GETWFMCH gets two inputs: the IP address and the port number. In our case, we type:

```
java GETWFMCH 172.16.1.100 15000
```

This program ask you for several inputs such as:

1. The address of the device that is connected to the Ethernut
2. The channel number of which you want to see the waveform
3. The record length along the horizontal axis (you can choose between 500 and 10000)
4. The starting data point for the waveform transfer which changes from 1 to the record length
5. The last data point to be transferred which changes from 1 to the record length

Once you provide the program with all these information, the data points are transferred to a file called **wfmdata.txt** in the same directory that the other files are saved, in this case C:\Sanazstuff\GPIB\javafiles.
The code for both files are in the appendix.

Chapter 5

Power Supply

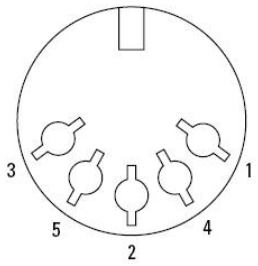
Among the tasks that I finished during my COOP term in the lab was ordering pieces of a power supply and putting them together to makes a Power Supply for a Picomotor driver. The Picomotor needed 3 different voltages +5, and +/-12 Volts.

5.1 What is a Picomotor?

Picomotor is an extremely compact high-resolution linear actuator. The Picomotor uses a piezo-electric transducer to turn a screw, resulting in incremental linear motion of less than 30 nm (100 nm for the Tiny Picomotor). Additionally, since the piezo is used to turn the screw and not to hold a position, the Picomotor can hold its position to better than 10 nm even when the power is off. The Picomotor uses a power supply to plug in the Picomotor driver.

The Picomotor comes with a control pad. This control pad turns the screw and adjusts its position. There are three sliders labeled A, B, and C on the control pad which adjust the corresponding Picomotors. Moving a slider up drives its corresponding screw clockwise and forwards. Moving the slider down drives the screw counter-clockwise and back. For turning in one direction the Picomotor will use the +12V output of the power supply and for turning in the opposite direction it uses -12V. The pin outs for the 5-pin DIN Power In connector on the back of the Model 8701 and 8801 Picomotor drivers are as follow: one common ground, one +5V, one -12V, one +12V, and one pin is not connected at all.

Figure 5.1 details the pinouts for the 5-pin DIN Power In connector on the back of the Model 8701 and 8801 Picomotor drivers.



Pin	Voltage
1	COM
2	Not Connected
3	+5 V
4	-12 V
5	+12 V

Figure 5.1: Power In Connector (pinouts looking at the back of the driver)

Based on this description I ordered an open frame power supply PT-45 series from Astrodyne Corporation. This power supply has 6 outputs: two common grounds, two +5V's, one +12V, and one -12V output.

5.2 Changes that I Made to the Power Supply

At first it seems easy to just order the part and then put it in a box and connect some wires. But generally in multiple output power supplies, a Minimum Load (minimum current draw required for voltages to be in a specified range) is required on the main output to ensure regulation of the auxiliary outputs, auxiliary outputs like +/-12V in this case. Some multiple output supplies require a Minimum Load on all outputs. If unsure, a Minimum Load of 10% to 20% is a good rule of thumb. Although most single output power supplies do not have a Minimum Load requirement.

I referred to Astrodyne detailed specification sheets provided in downloadable PDF format on their web site. I calculated 10% of the full load on the +5V output which was 0.3 Amps to determine the resistance of the minimum load. Connecting a 15 ohm resistor at 3 Watts between the +5V and ground of the power supply regulated the outputs. Before connecting the power supply to the Picomotor, I tried its functionality on a 33 ohm resistor at 10 Watts to see whether the value of current on the testing resistor is correct. I figured out the value of the testing resistor by measuring the current that the Picomotor was drawing from the previous power supply it was connected to. As a result the Picomotor uses +/-12V outputs

of the power supply to turn the screw in both directions: clockwise and counter clockwise. The important thing about the power supply is to have the minimum load on the auxiliary outputs so that all the outputs are regulated properly. Also it is better to order parts from companies inside Canada so that if you needed to ship them back for repair or replacement, you will not have to pay an extra cost for customs.

Chapter 6

Conclusion

During my four months COOP term in Quantum Degenerate Gases Laboratory at UBC, I worked on three different projects: the DDS, GPIB-Ethernut, and the Power Supply. My main focus was on the Direct Digital Synthesizer to test the newly built boards with the software that was implemented by Raymond Gao and me. I also worked on the GPIB Ethernut to create a user friendly interface, a combination of my Java code and Bruce's code will satisfy this purpose. The power supply was a more technical project that I completed in the lab.

In conclusion I enjoyed working on part of a complex control system which will be used to study fundamental physics at QDG lab. I thank the COOP program for giving me this opportunity.

Appendix A

The Java Code

A.1 EGInterface.java

```
import java.net.*;
import java.io.*;
import java.util.*;

public class EGInterface {

    private Socket sock;
    private BufferedReader in;
    private BufferedWriter out;
    private int devaddr;

    static final String SETIADDR = "!SETIADDR";
    static final String IDN = "*IDN?";
    static final long TIMEOUT = 20000;

    public EGInterface(String haddr,int port) {

        try {
           sock = new Socket(haddr,port);
           out = new BufferedWriter(new OutputStreamWriter(sock.getOutputStream()));
```

```
        in = new BufferedReader(new InputStreamReader(sock.getInputStream()));
    }
    catch (UnknownHostException e) {
        System.err.println("Trying to connect to unknown host: " + e);
    }
    catch (IOException e) {
        System.err.println("IOException:  " + e);
    }

}

int getAddress() {
    return this.devaddr;
}

int ibWrite(String s) {

    if(!sock.isConnected()) {
        System.err.println("Socket is not connected");
        return -1;
    }

    try {
        out.write(s,0,s.length());
        out.newLine();
        out.flush();

    } catch (IOException e) {
        System.err.println("IOException:  " + e);
    }
```

```
    return 0;
}

int ibRead(StringBuffer s) {

    if(!sock.isConnected()) {
        System.err.println("Socket is not connected");
        return -1;
    }

    try {
        Date d = new Date();

        long time = d.getTime();
        while (!in.ready()) {
            if( (new Date().getTime()-time) > TIMEOUT) {
                System.err.println("Timed Out Waiting for Response");
                return -1;
            }
        }
        System.out.println("Ready! trying to read");
        String t;
        while( (t = in.readLine()).compareTo("")==0 );

        s = s.insert(0,t);

    } catch (IOException e) {
        System.err.println("IOException:  " + e);
    }

    return 0;
```

```
}

int ibConnectGPIBAddress(int devaddr) {

    if(!sock.isConnected()) {
        System.err.println("Socket is not connected");
        return -1;
    }
    if(devaddr<0 || devaddr>31) {
        System.err.println("Invalid Address Range");
        return -1;
    }
    try {
        Date d = new Date();

        String s = new String(SETIADDR + " " + devaddr);
        out.write(s,0,s.length());
        out.newLine();
        out.flush();

        String responseLine;
        long time = d.getTime();
        while (!in.ready()) {
            if( (new Date().getTime()-time) > TIMEOUT) {
                System.err.println("Timed Out Waiting for Response");
                return -1;
            }
        }
        responseLine = in.readLine();

        if(responseLine.charAt(0)=='x') {
```

```
            System.err.println("Error Trying to Change GPIB Address");
                return -1;
        }

    } catch (IOException e) {
        System.err.println("IOException:  " + e);
    }

    this.devaddr = devaddr;

    return 0;
}

int ibFindInstrument(int devaddr,StringBuffer ID) {

    int tempAddr = this.devaddr;

    if(this.ibConnectGPIBAddress(devaddr)!=0) {
        System.err.println("Unable to Connect");
        return -1;
    }
    try {
        Date d = new Date();

        out.write(IDN,0,IDN.length());
        out.newLine();
        out.flush();

        long time = d.getTime();
        while (!in.ready()) {
            if( (new Date().getTime()-time) > TIMEOUT) {
```

```
                    System.err.println("Timed Out Waiting for Response");
                    return -1;
                }
            }
            ID = ID.insert(0,in.readLine());

        } catch (IOException e) {
            System.err.println("IOException:  " + e);
        }

        if(this.ibConnectGPIBAddress(tempAddr)!=0) {
            System.err.println("Unable to Connect");
            return -1;
        }

        return 0;
    }


    void ibCloseConnection() {
            try {
                out.close();
                in.close();
                sock.close();
            } catch (UnknownHostException e) {
                System.err.println("Trying to connect to unknown host: " + e);
            } catch (IOException e) {
                System.err.println("IOException:  " + e);
            }
    }
}
```

A.2 GETWFMCH.java

```
import java.lang.*; import java.net.*; import java.io.*;import
java.util.*;

public class GETWFMCH {

    public static void main(String[] args) /*throws IOException*/ {

        if(args.length!=2) {
            System.err.println("Invalid Arguments");
            return;
        }

        try {

        String IP = args[0];
        int port = Integer.parseInt(args[1]);

        EGInterface eg = new EGInterface(IP,port);

/*Asking for the device address connected to the Ethernut*/


    // 1. Create an InputStreamReader using the standard input stream.
        InputStreamReader isr = new InputStreamReader( System.in );

    // 2. Create a BufferedReader using the InputStreamReader created.
        BufferedReader stdin = new BufferedReader( isr );
```

```
    // 3. Don't forget to prompt the user
        System.out.println
        ("Enter the device address that is connected to the Ethernut: ");


/* Getting the input from the user and saving it in the variable
Daddr*/

        String Daddr_str = stdin.readLine();
        int Daddr = Integer.parseInt(Daddr_str);

/*Finding the device*/

        StringBuffer ID = new StringBuffer();
      //eg.ibFindInstrument(Daddr,ID);

/* Ask for CH number */

        System.out.println
        ("Enter the channel number of which you want to see the waveform:");

    // Save the number in CH_num variable
        String CH_num_str = stdin.readLine();
        int CH_num = Integer.parseInt(CH_num_str);

/*Asking got RECORDLENGTH # along the horizontal axis, the START and
STOP points*/

        System.out.println
        ("Enter the RECORDLENGTH # along the horizontal axis_
```

```
        Valid lengths are 500 and 10000 points.");

    // Save the number in RECORDLENGTH variable
       String RECORDLENGTH = stdin.readLine();

        System.out.println
        ("Enter the starting data point for waveform transfer_
        the # ranges from 1 to RECORDLENGTH.");

    // Save the number in START variable
        String START = stdin.readLine();

        System.out.println
        ("Enter the last data point that will be transfered_
        the # ranges from 1 to RECORDLENGTH.");

    // Save the number in STOP variable
        String STOP = stdin.readLine();

/*Connect to Daddr*/

        eg.ibConnectGPIBAddress(Daddr);

/*Open the output file*/

        BufferedWriter out = new BufferedWriter(new FileWriter("wfmdata.txt"));

/*Variables that the outputs of the ibRead will be saved in */

        StringBuffer s = new StringBuffer();
        StringBuffer f = new StringBuffer();
```

```
        StringBuffer nr_pt = new StringBuffer();
        StringBuffer yoff = new StringBuffer();
        StringBuffer ymult = new StringBuffer();
        StringBuffer xincr = new StringBuffer();
        StringBuffer pt_off = new StringBuffer();
        StringBuffer XUNIT = new StringBuffer();
        StringBuffer YUNIT = new StringBuffer();
        StringBuffer HORIZONTAL_SCALE = new StringBuffer();
        StringBuffer VERTICAL_SCALE = new StringBuffer();
        StringBuffer COUPLING = new StringBuffer();
        StringBuffer IMPEDANCE = new StringBuffer();
        StringBuffer BANDWIDTH = new StringBuffer();
        StringBuffer OFFSET = new StringBuffer();
        StringBuffer DATE = new StringBuffer();

/*Prepare to read the waveform data from the specified channel*/

        eg.ibWrite("DATA:SOURCE CH"+CH_num_str);
      if((eg.ibWrite("DATA:SOURCE CH"+CH_num_str)< 0) ||
         (eg.ibWrite("DATA:ENCDG ASCII") < 0)||
         (eg.ibWrite("HORIZONTAL:RECORDLENGTH "+RECORDLENGTH) < 0) ||
         (eg.ibWrite("DATA:START "+START) < 0)||
         (eg.ibWrite("DATA:STOP "+STOP) < 0)||
         (eg.ibWrite("HEADER OFF") < 0))
         {
           System.out.println("ibwrite Error: Unable to Setup waveform parameters");
/*break in java*/
           return;
         }

/* Make sure setup changes have taken effect and a new waveform is acquired */
```

```
        if (eg.ibWrite("ACQUIRE:STATE RUN") < 0)
         {
           System.out.println
           ("Error: GPIB error or timeout waiting to acquire waveform");
           return;
         }

/* Read screen image data. */

     if (eg.ibWrite("CURVE?") < 0)
        {
        System.out.println("ibwrite Error: CURVE?");
        return;
        }
       eg.ibRead(s);

/*I can also do one iBWrite and then tokenize the string,try this if
the method below did not work. */

/*Read waveform preamble */

       eg.ibWrite("WFMPRE:CH"+CH_num_str+":NR_PT?");
       eg.ibRead(nr_pt);
       // converts a String into an int value
       int NR_PT = Integer.parseInt( nr_pt.toString() );

       eg.ibWrite("WFMPRE:CH"+CH_num_str+":YOFF?");
       eg.ibRead(yoff);
       // converts a String into an int value
       float YOFF = Float.parseFloat( yoff.toString() );
```

```
        eg.ibWrite("WFMPRE:CH"+CH_num_str+":YMULT?");
        eg.ibRead(ymult);
        // converts a String into an int value
        float YMULT = Float.parseFloat( ymult.toString() );

        eg.ibWrite("WFMPRE:CH"+CH_num_str+":XINCR?");
        eg.ibRead(xincr);
        // converts a String into an int value
        float XINCR = Float.parseFloat( xincr.toString() );

        eg.ibWrite("WFMPRE:CH"+CH_num_str+":PT_OFF?");
        eg.ibRead(pt_off);
        // converts a String into an int value
        float PT_OFF = Float.parseFloat( pt_off.toString() );

        eg.ibWrite("WFMPRE:CH"+CH_num_str+":XUNIT?");
        eg.ibRead(XUNIT);

        eg.ibWrite("WFMPRE:CH"+CH_num_str+":YUNIT?");
        eg.ibRead(YUNIT);

/* Process waveform data*/

/*What Date*/
         eg.ibWrite("DATE?");
         eg.ibRead(DATE);
         out.write(DATE.toString() + '\r'+'\n');

/* Output header (x-, y-units, and source)*/
```

```
        out.write
        ("XUNIT: " + XUNIT + "  ,YUNIT: " + YUNIT +
        "  ,CH: CH" + CH_num + '\r' + '\n');

/* Getting the scale on the axis's.*/

        eg.ibWrite("HORIZONTAL:SCALE?");
        eg.ibRead(HORIZONTAL_SCALE);

        eg.ibWrite("CH"+CH_num_str+":SCALE?");
        eg.ibRead(VERTICAL_SCALE);

        out.write
        ("Horizontal scale is: " + HORIZONTAL_SCALE + " s" +'\r' + '\n'
        +"Vertical Scale is: " + VERTICAL_SCALE + " V." +'\r' + '\n');

/* What Coupling */

        eg.ibWrite("CH"+CH_num_str+":COUPLING?");
        eg.ibRead(COUPLING);
        out.write("CH" + CH_num_str + " has " + COUPLING + " COUPLING."+'\r'+'\n');

/*What Impedence*/

        eg.ibWrite("CH"+CH_num_str+":IMPedance?");
        eg.ibRead(IMPEDANCE);
        out.write
        ("CH" + CH_num_str + " has " + IMPEDANCE + " OHM impedance."+'\r'+'\n');

/*What Bandwidth*/
```

```
        eg.ibWrite("CH"+CH_num_str+":BANdwidth?");
        eg.ibRead(BANDWIDTH);
        out.write
        ("CH" + CH_num_str + " BANDWIDTH is " + BANDWIDTH + "." + '\r'+'\n');

/*What OFFSET*/

        eg.ibWrite("CH"+CH_num_str+":OFFSET?");
        eg.ibRead(OFFSET);
        out.write("CH" + CH_num_str + "'s OFFSET is " + OFFSET + " V." + '\r'+'\n');

/* Pay attention here, they might not be equal and then it does not
compile so check it by printing to the screen first*/

       String[] result = s.toString().split(",");

       out.write("The length of result is :" + result.length + '\r' + '\n');
       out.write(NR_PT + "   THESE TWO VALUES SHOULD BE EQUAL!!!" + '\r' + '\n');

       for (int j=0; j<result.length; j++){
            System.out.println(result[j]);}

 /* Output scaled x, y values in (Sec, Volts)
 * Time[i] = (i - PTOFF) * XINCR
 * Volts[i] = (point value - YOFF) * YMULT */

       out.write("*****************************"+'\r' + '\n');
       out.write("XUNIT  ,  YUNIT: " + '\r' + '\n');
       int i;
       double[] xunit = new double[NR_PT];
       double[] yunit = new double[NR_PT];
```

```
        for(i=0;i<NR_PT;i++)
       {
          System.out.println( ( (i-PT_OFF)*(XINCR) ) + " , " +
             ( (Float.parseFloat(result[i]) - YOFF) * YMULT) ) ;

             xunit[i] = (i-PT_OFF)*(XINCR);
             yunit[i] = (Float.parseFloat(result[i]) - YOFF) * YMULT;
             out.write
             ( ( (i-PT_OFF)*(XINCR)) + " , " +
             ( (Float.parseFloat(result[i]) - YOFF) * YMULT) + '\r' + '\n');

         }
         out.close();

         eg.ibCloseConnection();

         }
         catch (IOException e) { }
    }
}
```

Appendix B

Useful Web sites and Datasheets

B.1 The DDS

- Web site for different datasheets:
  http://www.alldatasheet.com/

- The AD9852 datasheet_ the main chip used on the DDS board:
  http://www.alldatasheet.com/datasheet-pdf/pdf/48612/AD/AD9852.html

- The web pages on "Control System for Cold Atom Experiments"_ it contain all the manuals for Hardware and Software designed at U-Texas such as the DDS:
  http://george.ph.utexas.edu/~control/
  http://george.ph.utexas.edu/~control/bus_system.pdf
  http://george.ph.utexas.edu/~control/DDS.pdf

B.2 The GPIB and Java programming

- The web page that contains the latest news on Java, its downloads, classes, and tutorials:
  http://java.sun.com

- The public forum on Java code, joining this forum is extremely recommended as Java Sun experts answer most of the questions on compiling errors, functions, and classes: http://forum.java.sun.com/index.jspa

- The web page for downloading the latest version of Jlink: http://www.wolfram.com/solutions/mathlink/jlink/

- Raymond Gao's report on GPIB and the GPIB-Ethernut manual: http://www.phas.ubc.ca/ qdg/internal/?path=./projects/ComputerControl/GPIBENUTMan

B.3 The Power Supply

http://www.astrodyne.com/astro/default.asp

References

[1] Control System for Cold Atom Experiments at U Texas Official Website: http://george.ph.utexas.edu/~control/.

[2] Raymond Gao, "Ethernut-GPIB Interface Manual", COOP student, QDG Lab at UBC, 2005.

[3] http://www.astrodyne.com/astro/default.asp