Computer Interfacing

Nate Schulz

Partner: Tim Struble
Data Recorded September 19 - October 7, 2008
Report Submitted October 22, 2008

1 Introduction

Much of science today involves taking analog measurements with digital devices, such as a PC. The purpose of this lab is to familiarize ourselves with the basic concepts, devices, and programming languages used to take these measurements. In this lab we use and analyze the performance of Measurement Computing's 1208-FS USB-2.0 measurement board. The first part of the lab serves to refresh our ANSI C programming knowledge and requires us to familiarize ourselves with the Application Programming Interface (API) libraries provided by the manufacturer. The 1208-FS APIs allow us to make calls to the board direct from our standard C command-line program. After mastering the C APIs, to set an output voltage on the experiment board, we move our coding practice into the graphical programming tool, LabView. LabView is developed by National Instruments (http://www.ni.com/labview/) and is a graphical development environment geared toward rapid and powerful prototyping of scientific data gathering software.

The LabView environment allows us to create "Virtual Instruments" or VIs, and it is from within these instruments that we collect data from interface boxes or devices. This extensionable architecture enables the same environment to record data from a variety of PCI card and USB devices. For this lab we create a variety of VIs and use the aforementioned 1208-FS USB-2.0 data acquisition device manufactured by Measurement Computing to collect data from a signal generator or output analog data to an oscilloscope.

The remainder of the lab exercise is completed using LabView's graphical programming environment since it has tools created specifically for taking and recording scientific data.

2 Theory

In order for digital devices to accept data into a software program like LabView or a custom command-line executable written in C, Basic, or any other compiled language, the analog data must first be converted to digital data through a process known as analog to digital conversion. This process is commonly referred to as A/D conversion or just A/D. In this exercise, the 1208-FS USB device will serve as both our A/D converter as well as our D/A converter. D/A is the process by which digital (binary) data is converted into an analog voltage which can be read on an oscilloscope, voltmeter, or used to power some minimal electrical load like a small filament lightbulb.

The 1208-FS has a 12-bit range in both D/A and A/D processes meaning its maximum digital range is 2^{12}, or 4096 different input/output levels. Remembering that digital processes begin with 0, we define our range to be 0 to 4095. This is the value that gets read to or from the computer. The basic architecture of a A/D or D/A converter can be seen in figure 1 of a 4-bit A/D converter which is built around a R-2R ladder circuit.

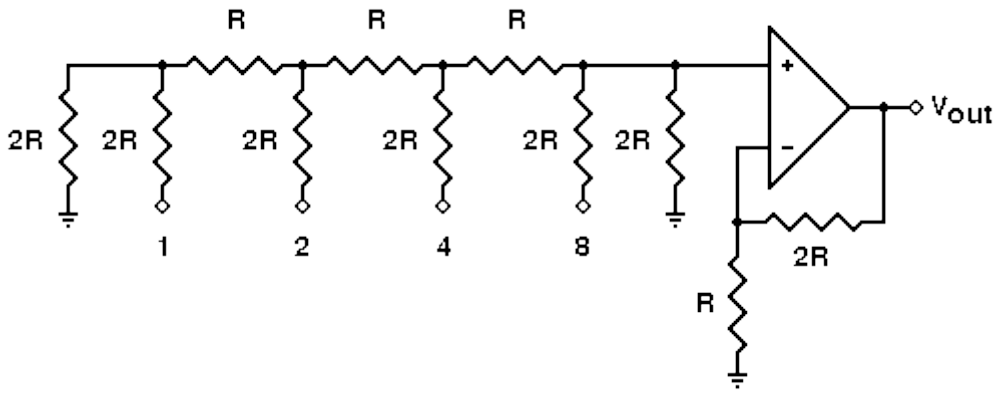


Figure 1. A 4-bit D/A converter using a R-2R ladder circuit. (Kelty)

The 1208-FS utilizes a larger 12-bit D/A converter but has a similar circuit for both its D/A and A/D converters which feed into a USB micro-controller. The micro-controller handles the transmission of data to and from the computer. The basic structure of the 1208-FS is shown in the block diagram below. The A/D converter is represented by the analog input box and the D/A converter is represented by the analog output box.

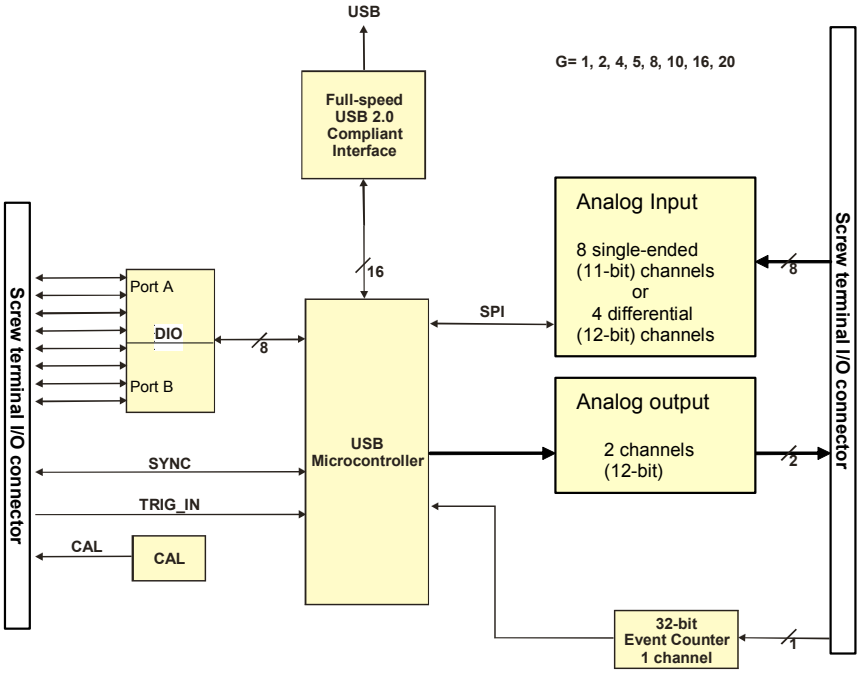


Figure 2. 1208-FS block diagram. (1208-FS User Guide)

The 1208-FS can convert A/D or D/A in software-paced or hardware-paced modes. Software-paced mode is driven by the computer and places high amounts of traffic on the USB bus. Software-paced mode also typically requires more CPU time and

is unable to capture or output data at the same speeds as hardware mode. In hardware-paced mode, the 1208-FS will capture at a specified sampling frequency and then send all of the data back to the computer as a data block once a specified number of samples have been taken.

We will take our software and hardware paced datasets and run fast Fourier transforms or FFTs on them in OriginLab's Origin plotting software. A fast Fourier transform should produce a graph with a visible peak near our input frequency if our sampling rate is sufficient. We expect the maximum frequency that we are able to correctly sample and complete a FFT on to be one-half that of the maximum sampling frequency of the acquisition device. We know this from the sampling theorem, which shows mathematically that the maximum correctly sampled frequency can be only one-half of the sampling frequency since there must be a minimum of two sampling points per curve to adequately store the max and min values of the curve. Therefore, we expect the maximum sampled value to be different for software and hardware paced modes as hardware paced sampling allows for much higher sampling rates.

3 Apparatus and Procedures

This lab requires the use of the 1208-FS USB-2.0 interface box, a wave generator, a multimeter, oscilloscope, PC with Microsoft Visual Studio, National Instruments LabView and OriginLab's Origin data plotting software. Additional items include single strand wire cut to various lengths, banana leads, a small flat-head screwdriver for tightening the wire terminals on the 1208-FS interface box. Microsoft Excel can be used for creating and logging quick data sets and performing basic data table manipulation before graphing in Origin.

Digital to Analog

We begin by connecting a multimeter to pins 13 and 15 where 15 is the ground. Once we have the device connected to our meter we begin writing our C program by creating a new Microsoft Visual Studio project. This allows us to understand what lies underneath a program like LabView. After including the Measurement Computing header file, and adding the cbw32.lib library to our project we write our main() method so that it sets the output value voltage of the 1208-FS board. The source code to this simple program is 19 lines long and is shown here.

```
1  #include "\Program Files\Measurement Computing\DAQ\C\cbw.h"
2  #include <iostream>
3  #include <windows.h>
4
5  void main()               // begin main method
6  {
7     short voltage = 2048;  // Set the output voltage value
```

```
    int BoardNum = 0;       // Specify the desired board device
    int Channel = 0;        // Specify the output channel
    int Range = UNI5VOLTS; // Output range setting (device constant)
    int result = 0;         // Initialize result variable

    // Call into the board API to set an output voltage
    result = cbAOut(BoardNum, Channel, Range, voltage);

    if (result == 1)        // cbAOut method returns 1 on success
        std::cout << "Output confirmed: " << voltage <<std::endl;
    else                    // something went wrong, display error
        std::cout << "An error occurred " << std::endl;
} // close main method
```

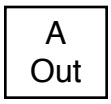
After we successfully output a single voltage with our C program, we proceed to test the time stability of the output voltage. Time stability is how steady the voltage remains over time. Typically devices will heat up as current passes through them and may effect the output voltage. We test this stability for 3 minutes.

Next we create sawtooth and square wave outputs to test our understanding of the C language and APIs. We now hook our output terminals to the oscilloscope to view the changing output voltage over time.

```
#include "\Program Files\Measurement Computing\DAQ\C\cbw.h"
#include <iostream>
#include <windows.h>

// Begin main method
void main()
{
    short voltage = 0;            // Declare and set initial voltage
    int BoardNum = 0;             // Specify device number
    int Channel = 0;              // Specify output channel
    int Range = UNI5VOLTS;        // Specify output range (Device API Constant)
    int result = 0;               // Declare and initialize result to 0
    bool scan = true;             // Used to specify whether we are scanning or setting
    float rampUpTime = 0.0;       // Declare and initialize rampUpTime
    float rampDownTime = 0.0;     // Declare and initialize rampDownTime
    std::cout.precision(20);      // Set cout precision level
    if (scan)                     // Scan output voltages if true
    {
        bool goingUp = true;              // Start scanning up
        double startTime = time(NULL);    // Store start time

        while (voltage < 4096 && result == 0)    // While output is within range and
        {                                        // no errors have occurred continue scanning
            if (goingUp) voltage++;              // if we're still going up,increment voltage
            else voltage--;                      // we're going down, decrement voltage
            result = cbAOut(BoardNum,Channel,Range,voltage); // Set actual voltage on device

            if (voltage + 1 == 4096)             // If we're at the top, turn around
            {
                goingUp = false;
```

```
31                 double endTime = time(NULL);      // Get end time of ramp up
32                 rampUpTime = endTime - startTime; // calculate ramp up time
33                 std::cout << "Ramp Up Time: " << rampUpTime << " ms" <<std::endl; // display
34                 startTime = time(NULL);           // reset start time
35             }
36             if (voltage - 1 == -1)               // If we're at the bottom, turn around
37             {
38                 goingUp = true;
39                 double endTime = time(NULL);        // Get end time of ramp down
40                 rampDownTime = endTime - startTime; // calculate ramp down time
41                 std::cout << "Ramp Down Time: " << rampDownTime << " ms" <<std::endl;//display
42                 startTime = time(NULL);             // reset start time
43             }
44         }
45         double endTime = time(NULL);               // get end time after loop
46         rampUpTime = endTime - startTime;          // Display elapsed time
47     }
48     else                                           // Don't scan, just set an output
49     {
50         result = cbAOut(BoardNum, Channel, Range, voltage); // set output on device
51     }
52     std::cout << "Result: " << result << std::endl;  // display result of cbAout method
53     std::cout << "Required Time: " << rampUpTime << " ms" <<std::endl;//display required time
54 } // close main method
```

Once we are satisfied that we have the general idea behind programming instrument interfaces with our C program, we move to LabView's graphical development environment. Our computer has already been set up with the necessary VIs for reading and writing data to the USB1208-FS device. In this lab we use the "A Out" VI to do D/A conversions and the "A In" VI to read A/D conversions.

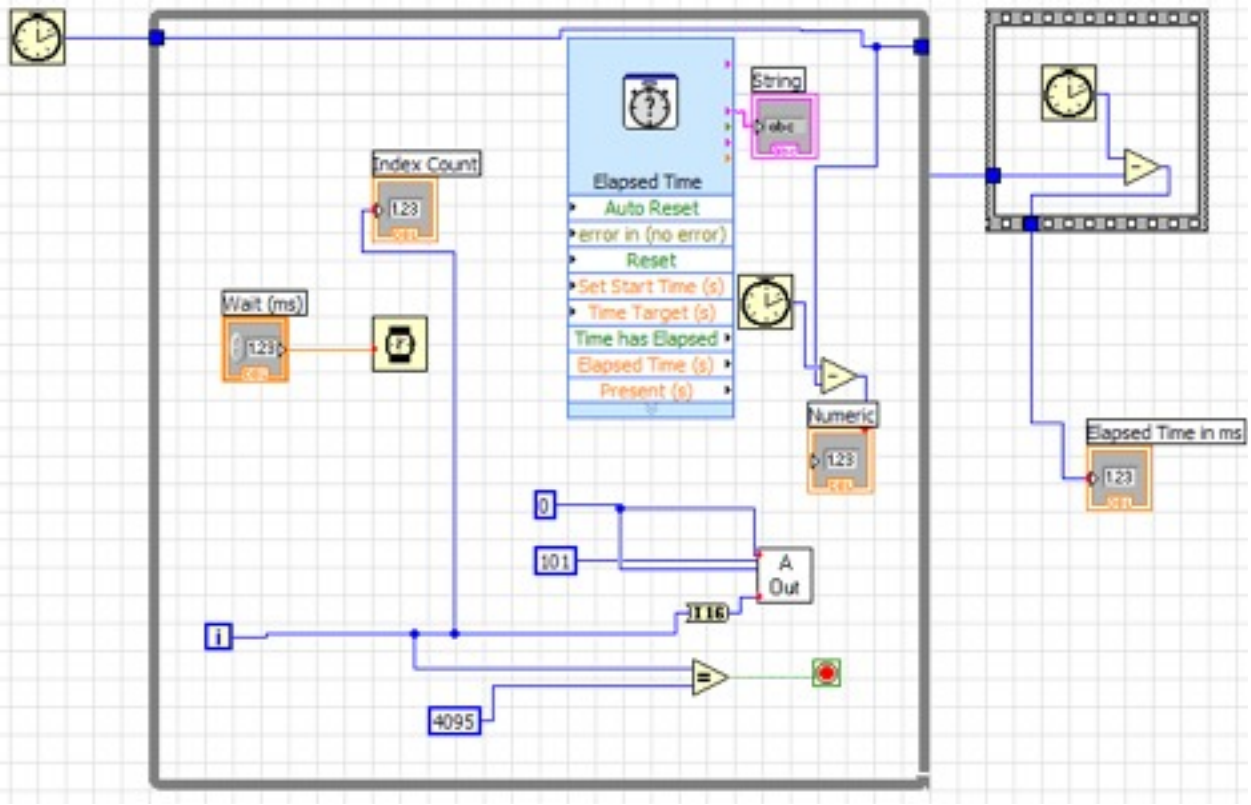


Manufacturer provided D/A VI.

Manufacturer provided A/D VI.

We proceed directly to creating our own LabView VI which utilizes these pre-build components. We mimic our C program functionality before continuing with the rest of the lab to familiarize ourselves with the LabView environment. Once our basic voltage output is working, we create various D/A VIs that are used to measure any gain, voltage offset, and non-linearity in the 1208-FS output. These VIs include, a basic voltage scanner, square wave generator, and a sine wave generator. The first VI we build is the basic voltage scanner.

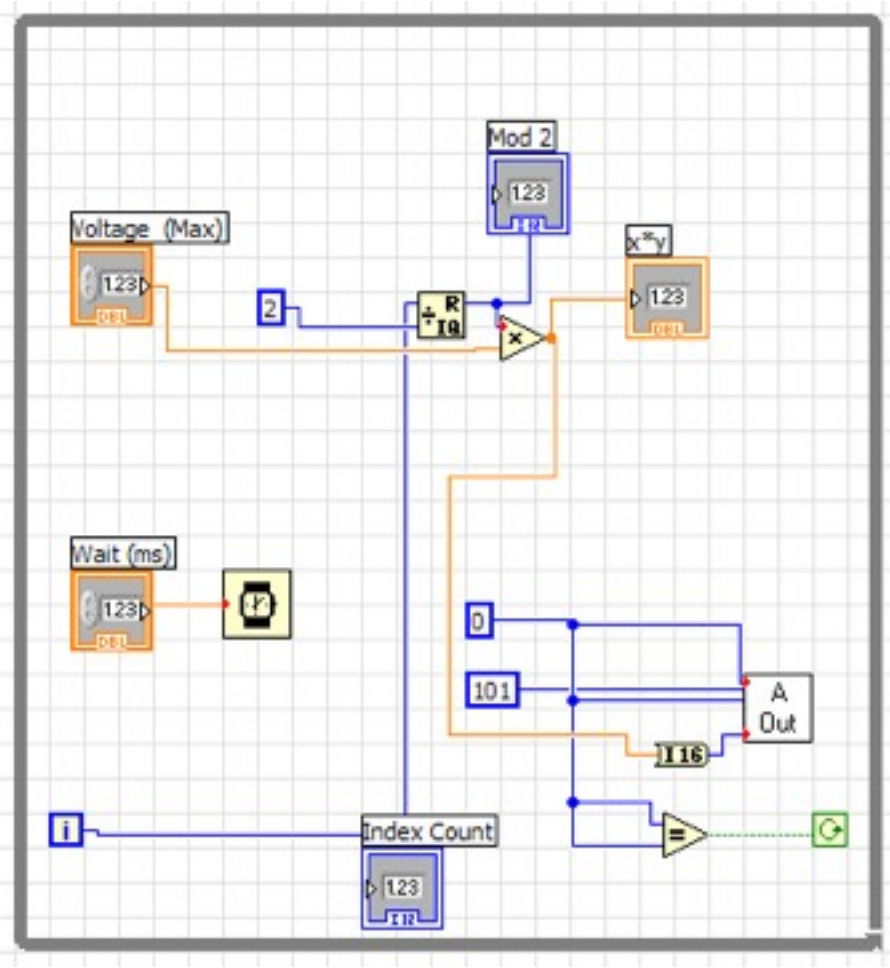
Voltage Scanner - Block Diagram



We begin with a simple voltage scanner. Here we see a screenshot of our block diagram for our voltage scanner VI. This creates the same output as our voltage scanning C code. Utilizing a tick counter to the left of the while loop, represents our static start time at time of execution. The second frame to the right of the while loop is an isolation loop which causes the components inside of it to execute only after the while loop is complete. This effectively records our end time and calculates the difference between our start and end times and displays the output to the front panel element labeled, “Elapsed time in ms.”

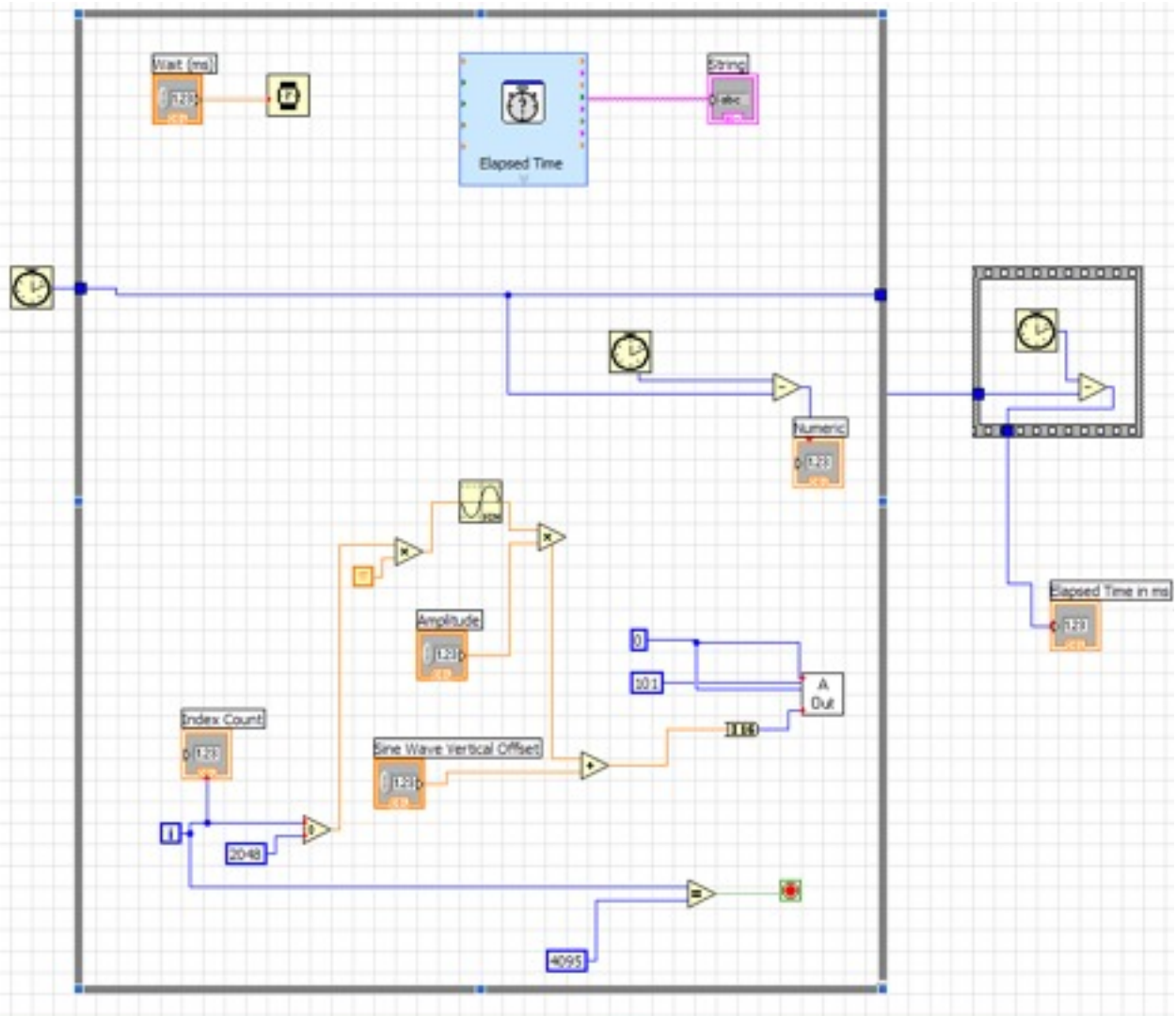
Inside the while loop we use the index count (*i*) to specify the output voltage and terminate the loop after 4096 executions, counting the initial loop as 0 index. This effectively scans through the entire output range of the box from 0 - 4095. Also inside our while loop is an elapsed time counter which gives us a readout of time elapsed per loop execution, as determined by the wait component. The A Out VI is where the actual output gets set to the device. The constant 0 is used to specify channel and device numbers, while the constant 101 is the range value specifying that the device is to use the analog range from 0-4.096V. This constant is found in the header files for the C APIs.

Square Wave Generator - Block Diagram



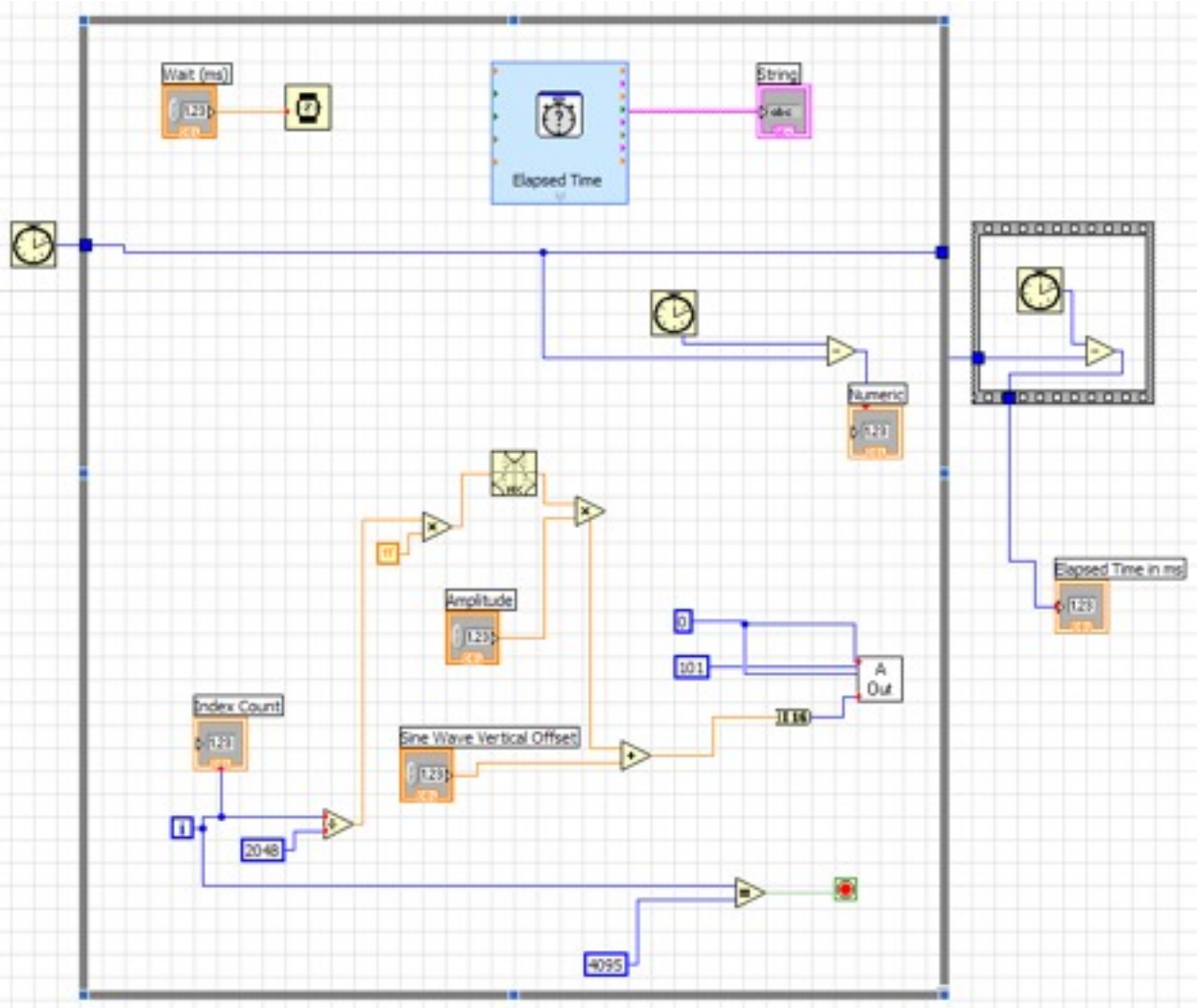
After successfully creating a voltage scanning VI, we continue by building a square wave generator utilizing our previous VI as a foundation. We modify it to include a modulo 2 (0 or 1) number by using a divide with remainder component. We use the loop index and pass it through our division component resulting in a 0 for even indices and a 1 for odd indices. This binary result is then multiplied with a user-specified output voltage with an input field on the front panel for easy, real-time adjustments. Finally, we add a wait component to limit the speed of the loop execution to several seconds per loop. This allows our output wave to be clearly visible on the oscilloscope.

Sine Wave Generator



After completing our square wave generator, we turn our focus towards creating a sine wave using the 1208-FS device. The sine wave generator is slightly more complex than the VIs of the previous two steps. Since our output range is 0-4.095 V we must offset the 0 point of our sine wave. In order to render the full range of the curve on the oscilloscope, we must shift our time axis t up 2.048V. This will allow the full curve to be generated in positive values between 0 and 4.095V. We use the sine math component to do the actual calculation of the voltage as t is based on the loop index and multiply that result by our desired amplitude of 2048, or half of the full range of the device. The result is a double and is converted to a 16-bit integer in the I16 component before sending the value to the 1208-FS device.

Secant Wave Generator



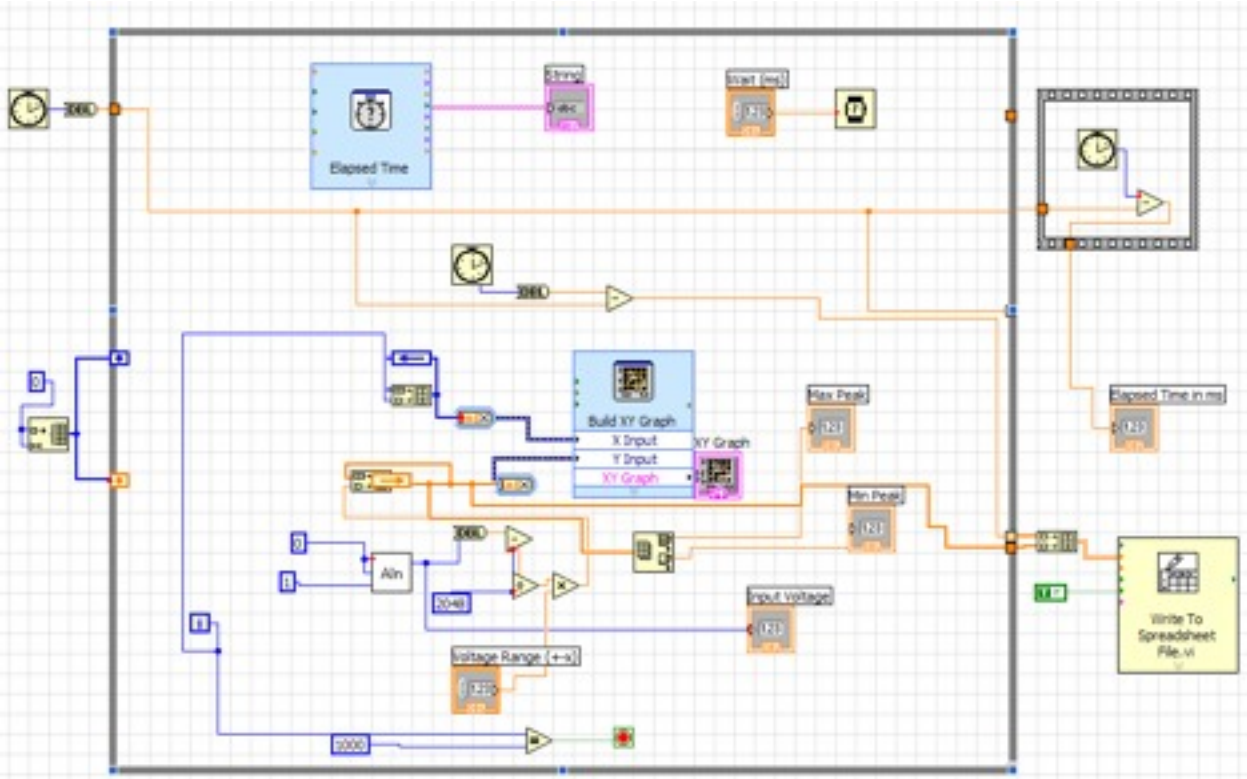
Our secant wave generator VI is similar to our sine wave generator. Simply changing out the sine function with a secant function creates the desired effect. As with our sine wave generator, we are able to to adjust the period and amplitude of the wave output.Analog to Digital

With the basics of D/A conversion covered, we proceed to examine the device's A/D conversion abilities. We begin by connecting a DC power supply to pins 1 and 3 on the 1208-FS. Pin 1 is the positive terminal and pin 3 is an analog ground. The 1208-FS has an acceptable input voltage range of ± 10V. To ensure that we do not go outside of this range, we hook our oscilloscope up in parallel with the capture board to monitor our voltage and later the waveform shape and amplitude. We test our setup to check that the computer is reading in a voltage, and that the value corresponds to the correct voltage. We then test for any offset, gain, and non-linearity, followed by testing for time-

stability while reading a voltage. To do this, we use a similar process as used in the D/A conversion test, however this time we reverse the process.

After collecting our specification data, we connect pins 1 and 3 to a waveform generator and build our real-time graphing VI.

Analog to Digital VI - Block Diagram - Software Paced



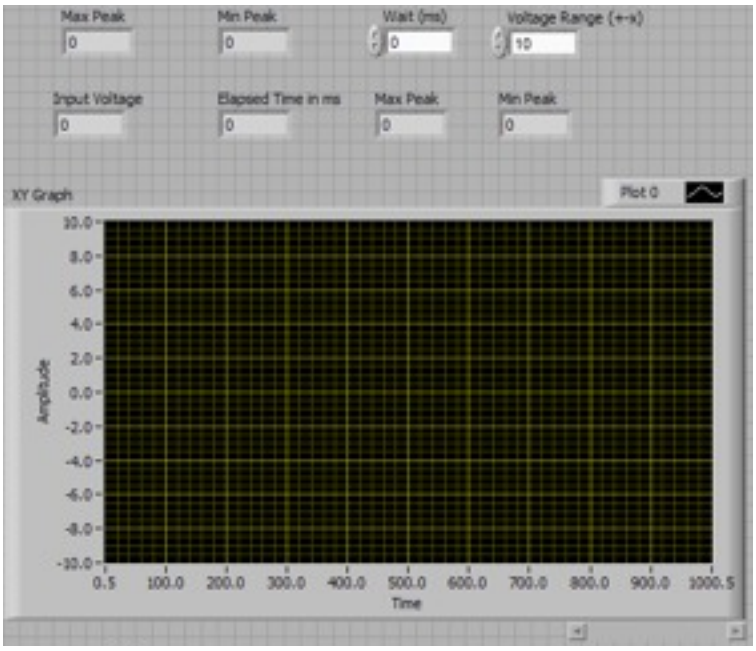
While the reading of the analog data itself is a trivial task, thanks to the AIn VI, graphing and storing that data as an array and writing the results to a spreadsheet compatible file is an in-depth process. Our A/D VI represents the most complex VI required in this lab. It serves as the measurement tool for the remainder of the lab exercises.

We begin with the same tick counters before, during, and after the loop execution frame. This serves as our time stamp as well as providing an elapsed time readout during the loop execution. For our own delight and real-time results, we placed our graph and graph builder components inside the execution loop. This enables us to display the captured wave form on the front panel, just as would be seen on an oscilloscope. Another advantage of doing the graphing in real-time is that we know immediately whether or not our data is recording correctly.

Since our input voltage range is ±10V, we need to translate the 0-4095 value that gets read from the 1208-FS device into positive and negative values. To do this, we set up the appropriate math components to evaluate the following expression.

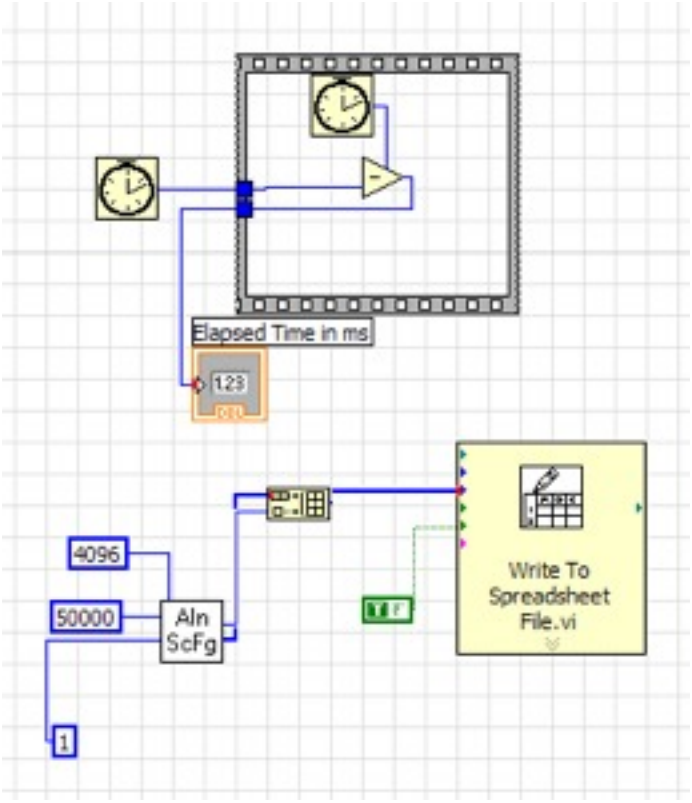
$$((V_{in} - 2048) \ / \ 2048) * V_{amplitude}$$

Where V_{in} is the value between 0 and 4095 from the 1208 box, and $V_{amplitude}$ is the magnitude of the voltage range. In our case $V_{amplitude} = 10.0$. The result of this expression is then our analog voltage reading. We bundle this value with the loop index as our time value into a new array and feed the array into the waveform graph builder. Our front panel (pictured below) serves as a basic readout for all relevant information, including max and min values, and elapsed time.



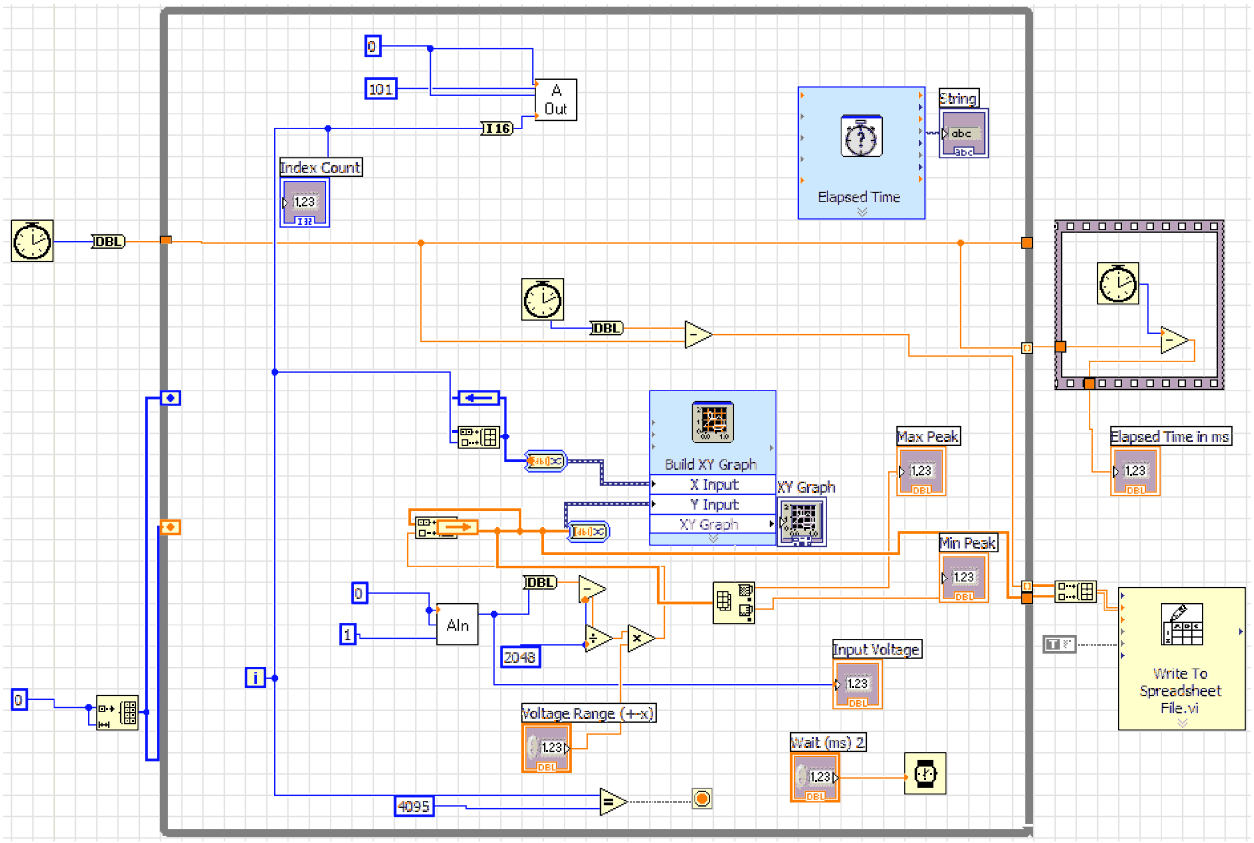
With our data gathering VI ready to go, we proceed with the lab by taking datasets at multiple software-paced frequencies to determine our maximum sampling rate in software paced mode. Once taking this data, we use OriginLab's Origin plotting software to do a fast Fourier transform (FFT) to determine the accuracy of our sampling rate.

Analog to Digital VI - Block Diagram - Hardware Paced



The hardware paced VI is simpler as it is only counting elapsed time and displaying it in milliseconds to the front panel after reading data off the acquisition box in one block. This VI automatically writes the received data block to a spreadsheet file. We repeat our datasets using a similar procedure to the one used for software-paced sampling. This time we expect to see a greater accuracy at higher frequencies since the box is driving the sampling rate and transferring a data block after gathering all of the samples.

Voltage Machine A/D and D/A conversion - Software Paced



This VI is a modified version of our software-paced A/D converter. In this version, we output a voltage and monitor an input voltage simultaneously. This tests the 1208's ability to perform A/D and D/A simultaneously and allows for maximum precision while gathering data with the device. We now take a ramp up data set to test for non-linearity. Since we are gathering data with the acquisition box, we are able to take a full set of data points from 0 to 4095. These data points are then collected and saved automatically to a tab-separated file which is then read into Microsoft Excel or OriginLab's Origin plotting software.

4 Results and Analysis

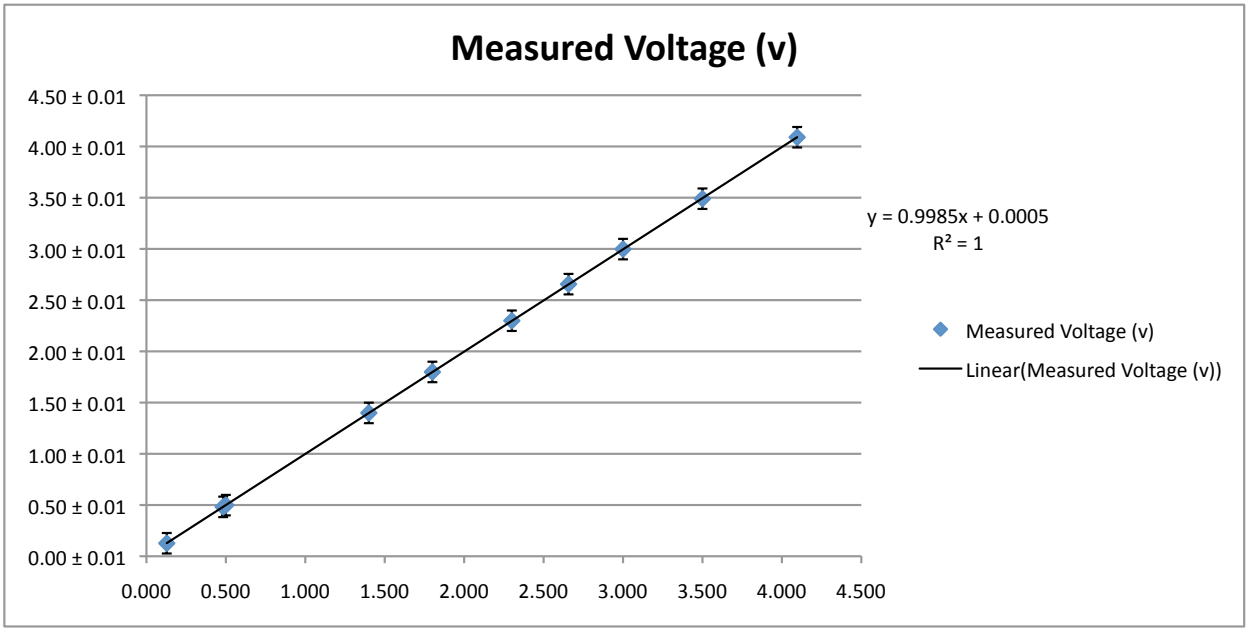
Digital to Analog

Time Required to Sweep Voltage

Trial	Elapsed Time (ms)	Number of Loops	Time Per Loop (ms)
1	16400	4096	4.00390625
2	16423	4096	4.009521484375
3	16401	4096	4.004150390625
4	16418	4096	4.00830078125
5	16466	4096	4.02001953125

Trial	Expected Output (V)	Measured Voltage (V) ($\sigma \pm 0.001$)
1	4.096	4.094
2	3.500	3.498
3	1.800	1.799
4	0.500	0.499
5	0.483	0.482
6	0.128	0.127
7	1.400	1.399
8	3.000	2.998
9	2.657	2.656
10	2.300	2.299

Our measured voltage is about 1mV (0.001V) below our expected output. This is due to the fact that our VI is in fact including 4096 in its calculation of expected voltage, when the correct range should be only 0 to 4095. There is however, an occasional voltage difference when measured to mVs. This difference is not linear and does not seem to follow any specific pattern, it could be due to the accuracy of the multimeter and our circuit wiring.



After examining expected voltage vs measured voltage by eye, we use Excel to perform a quick plot and fit a trendline to the dataset to determine its linearity. We find that it is extremely linear and has a slope of 0.9985 which is to be expected given our standard deviation in the measurements taken from the multimeter. The y-intercept is calculated to be 0.5 mV which could represent a voltage offset, but is rather small and is more likely to be simply a result of the Excel determined trendline that best matches our particular set of data points.

Square Wave Output Performance

Trial	Voltage (V)	Max *f (Hz)*	Wait (ms)
1	1	125	0
2	0.95	125	0
3	4	125	0
4	4.095	125	0
5	3.25	125	0
6	2.8	125	0
7	1.8	125	0
8	2.25	125	0

We begin to see that our maximum sampling rate in software-paced mode is approximately 125Hz which we examining in further detail later in the lab. Using 5

different output voltages we record the execution time in software-paced mode while also recording the rise and fall times on both the computer plot and our oscilloscope.

In-Loop Execution Times in Microseconds

Trial	Elapsed Time (ms)	Number of Loops	Time per loop (ms)
1	16400	4096	4.00390625
2	16423	4096	4.009521484375
3	16401	4096	4.004150390625
4	16418	4096	4.00830078125
5	16466	4096	4.02001953125

Execution time per loop is 4.00918 on average which corresponds to a maximum software-paced sampling rate of approximately 249.427 Hz.

Rise and Fall Times in microseconds

Trial	Voltage	Rise Time	Fall Time	Start Rise	End Rise	Start Fall	End Fall
1	4096	5.75	5.2	14.9	9.15	12.25	17.45
2	2250	2.5	3.05	--	--	12.45	15.5
3	3854	4.9	4.8	14.9	10	12.55	17.35
4	1850	2.55	2.6	14.85	12.3	12.45	15.05
5	500	1.2	1.7	27.9	26.7	26.4	28.1

We can clearly see a correlation between rise/fall times with voltage. At higher voltages, rise time is higher. Similarly lower voltages result in shorter rise/fall times. We can say that rise/fall time is proportional to voltage.

Analog to Digital

In order to determine an A/D gain, we take 10 different voltage amplitudes on a sine wave and capture maximum and minimum voltages both in LabView using the 1208 interface box and compare our results to max and min values determined by the oscilloscope.

Trial	Max Computer	Min Computer	Max Scope (V)	Min Scope (V)	Difference (V)	Difference (V)	Gain (mV)	Gain (mV)	Gain Average
1	9.17	-9.77	9.4	-10	0.23	0.23	230	230	230

Trial	Max Computer	Min Computer	Max Scope (V)	Min Scope (V)	Difference (V)	Difference (V)	Gain (mV)	Gain (mV)	Gain Average
2	8.44	-8.98	8.6	-9.2	0.16	0.22	160	220	190
3	7.63	-8.1	7.8	-8.4	0.17	0.3	170	300	235
4	1.19	-0.51	1.2	-0.56	0.01	0.05	10	50	30
5	2.07	-1.62	2.08	-1.76	0.01	0.14	10	140	75
6	2.79	-2.5	2.8	-2.64	0.01	0.14	10	140	75
7	3.43	-3.28	3.44	-3.36	0.01	0.08	10	80	45
8	4.78	-4.88	4.8	-5.04	0.02	0.16	20	160	90
9	5.52	-5.74	5.6	-5.84	0.08	0.1	80	100	90
10	6.31	-6.63	6.4	-6.8	0.09	0.17	90	170	130

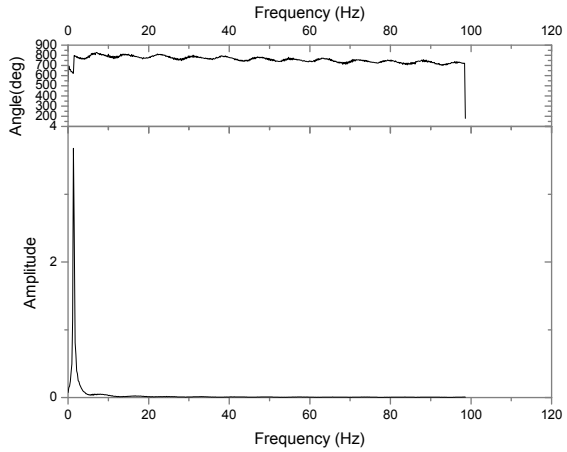
With this data we find a typical gain of 230mV for ±10V range, 90mV for ±5V range, 75mV for ± 2V range and 30mV for ±1V range. This values are off by a magnitude of 10 from the specifications but match the leading digit quite close which leads us to believe that our data simply is off by a factor of 10 somewhere in our formulae.

Fast Fourier Transforms

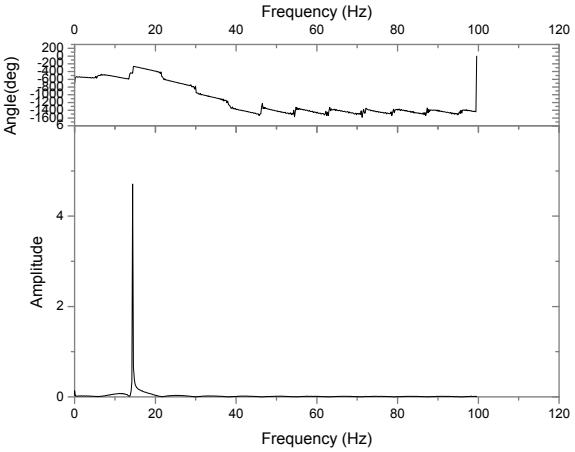
Software-Paced FFTs

Using Origin, we complete FFTs for each of our datasets. We have 1.5Hz, 15Hz, 40Hz, 80Hz, 100Hz, 120Hz, 150Hz, and 1500Hz using software-paced sampling. With our sampling rate (f_s) at maximum (approximately 250Hz), we expect to see our disconnect appear around 120-125Hz.

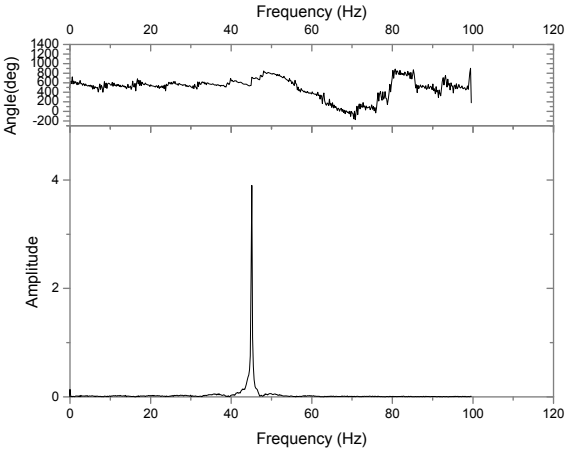
Software-Paced - 1.5Hz FFT



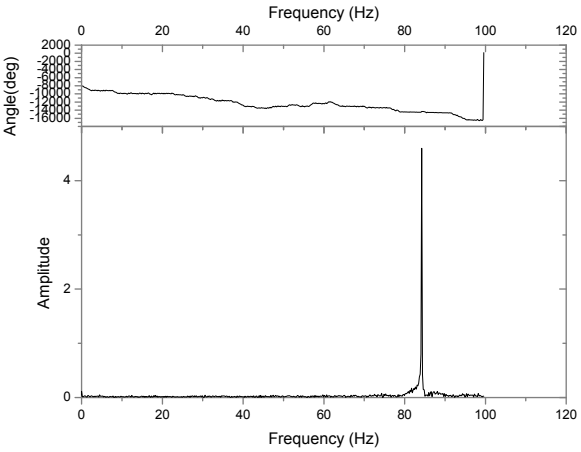
Software-Paced - 15Hz FFT



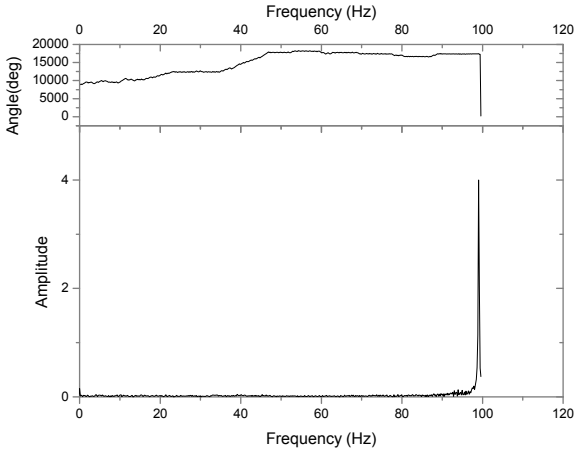
Software-Paced 40Hz



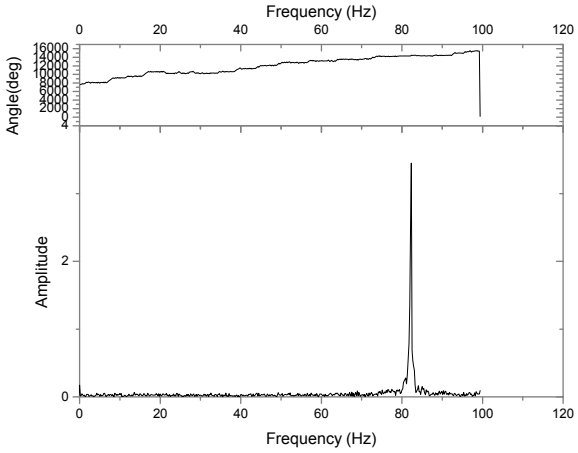
Software-Paced 80Hz



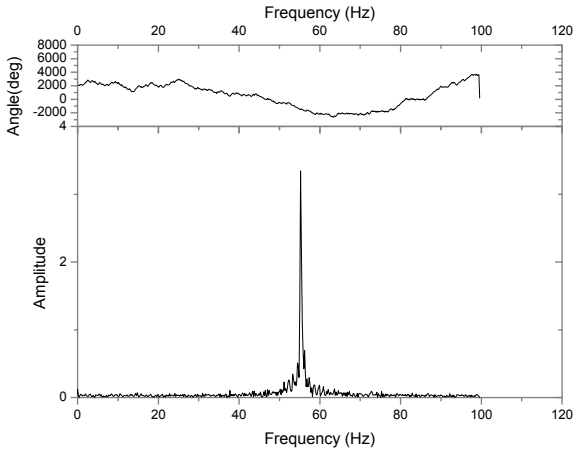
Software-Paced 100Hz



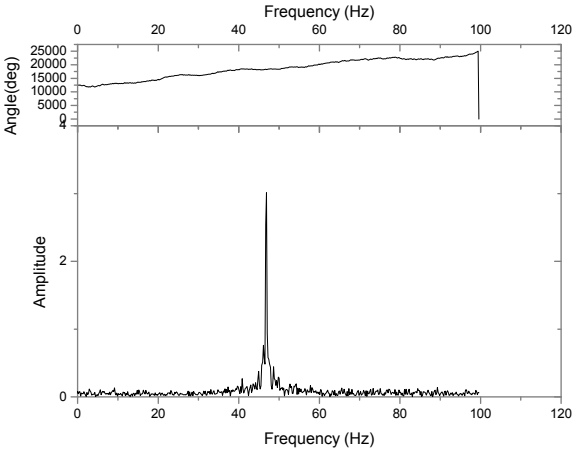
Software-Paced 120Hz



Software-Paced 150Hz



Software-Paced 1500Hz

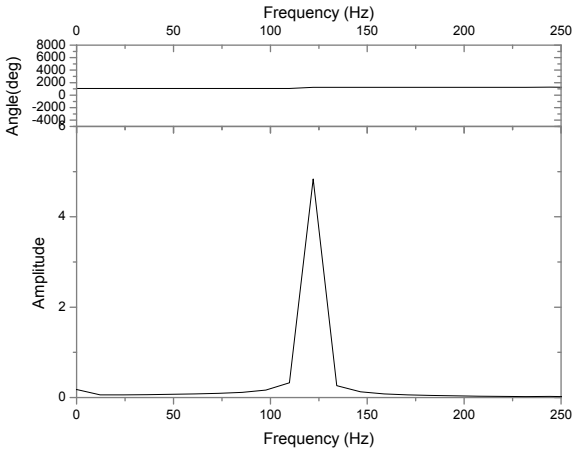


The sampling limit is clearly visible beyond our 100Hz input frequency resulting in incorrect FFT peaks.

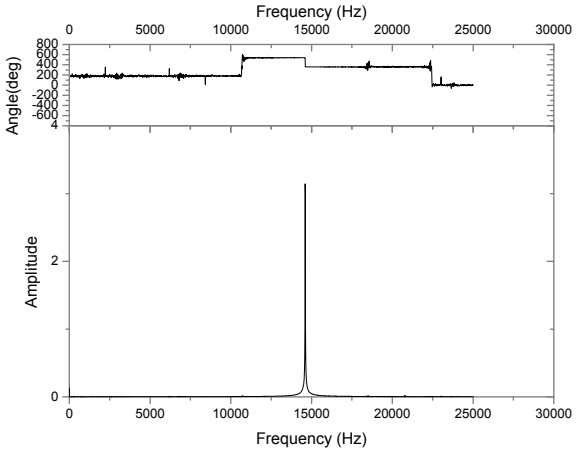
Hardware-Paced FFTs

Using hardware-paced sampling at the 1208's maximum hardware sampling rate of 50000Hz, we took samples at 125Hz, 15kHz, 20kHz, 25kHz, 30kHz, and 40kHz. We expect to see a disconnect around 20-25kHz since 25kHz is $f_s/2$.

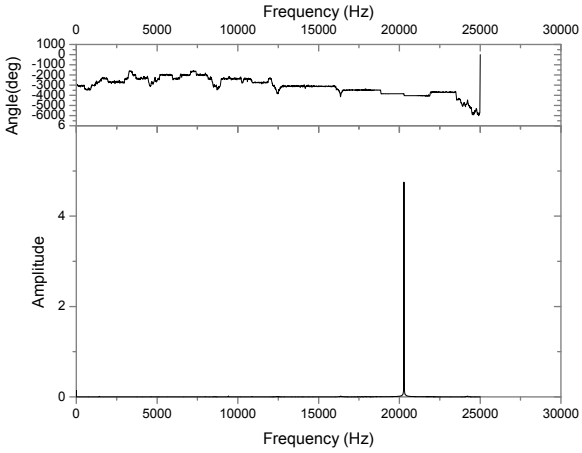
Hardware-Paced 125Hz



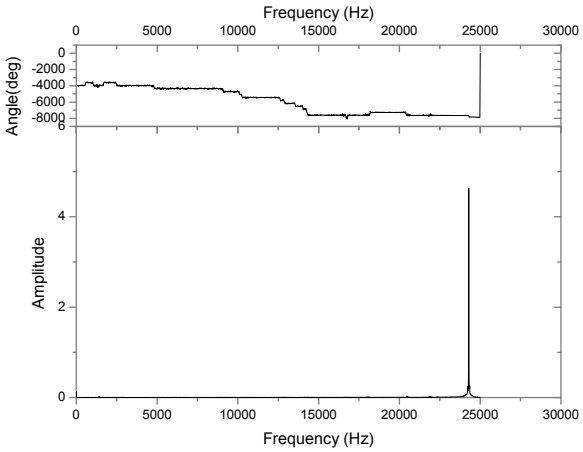
Hardware-Paced 15kHz



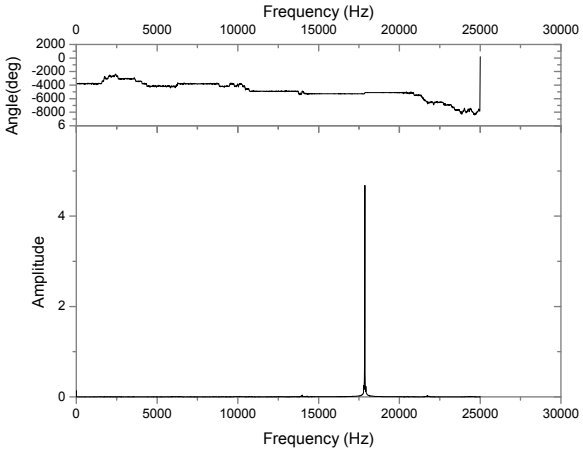
Hardware-Paced 20kHz



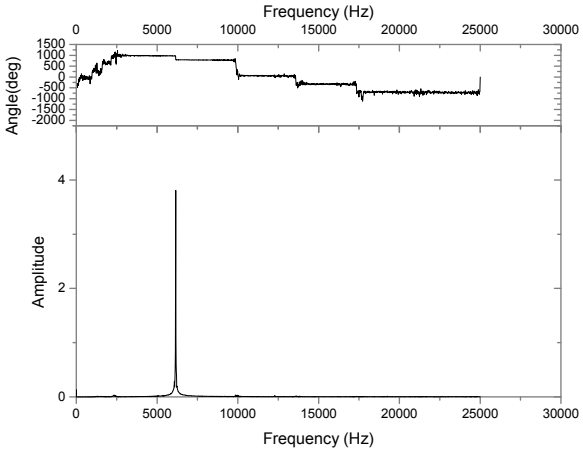
Hardware-Paced 25kHz



Hardware-Paced 30kHz



Hardware-Paced 40kHz



Here our sampling breakdown is clearly visible beyond our 25kHz input frequency and is shown to be beginning to break down in our 25kHz dataset. This could be caused by simple inaccuracy in the waveform generator at such a high output frequency possibly due to the lack of definition in the frequency control knob and/or waveform breakdown at high frequency.

5 Discussion and Conclusions

This lab examined the features and manufacturer specifications for Measurement Computing's 1208-FS USB-2.0 measurement board. By taking detailed datasets, we tested its D/A conversion as well as A/D conversion in both software and hardware-paced sampling modes. We found the maximum software sampling rate to be 249.427Hz which is very near the specified 250Hz. We consider this result to be a successful confirmation of the defined specification. To test for time-stability we monitored different output voltages for 3 minutes and found the output voltage to remain extremely steady to the mV on our multimeter and oscilloscope

We then gathered several data sets to look for a voltage offset over the range of input voltages in the device and were unable to find any voltage offset to match the 9.766mV offset specified in the 1208 specifications. Using additional data points over the same range and checking the calibration of the oscilloscope may have produced a more accurate offset calculation. If we were to do this lab again in the future, we may benefit from using an additional oscilloscope as well. We were able to find a gain with an appropriate first digit, but our magnitude was off by a factor of 10. This caused our gain to result in hundreds of mV as opposed to the tens of mV defined in the 1208 specifications table. We determined that there is most likely a magnitude error in our formulae. We intend to compare our results with another team who has completed this lab.

In addition to examining the 1208-FS usb interface device in great detail, we were exposed to the theory behind the sampling theorem of information theory. This mathematical formula defines the maximum possible input frequency to be sampled can be no greater than one-half the sampling rate. This is implied by the fact that a sampled waveform must have a least two data points per curve in order to determine an accurate input frequency. We found these limits to be very accurate for both software and hardware paced sampling modes. Therefore our results support the sampling theorem.

In addition to discussing and working with the sampling theorem, this lab allowed us to gain experience in LabView, Origin, and Microsoft Visual Studio. These are tools that we expect to see in our future experiments and will serve us in nearly any chosen field of research or employment.

References

1208-FS User Guide by Measurement Computing, 2007.
http://www.measurementcomputing.com/PDFManuals/usb-1208fs.pdf

LabView User Manual, 2003.
http://www.ni.com/pdf/manuals/320999e.pdf

1208-FS API Manual "sm-ul-functions.pdf", 2007.

Kelty, J.R., "Phys 231 Lab 13 Ladder Digital to Analog Conversion"
http://my.unl.edu/@@1DC91B2C079E43F6B4F60434A92B8E93/courses/1/PHYS231150.20082/content/_2521293_1/Lab%2013%20Ladder%20DAC.doc

Smith, J.O. "Mathematics of Discrete Fourier Transform (DFT) with Audio Applications", Second Edition, http://ccrma.stanford.edu/~jos/mdft/, 2007, online book, accessed 20-10-2008.

Smith, J.O. "Sampling Theorem", http://ccrma.stanford.edu/~jos/r320/Sampling_Theorem.html, 2008, accessed 20-10-2008.