Experiment 2: Digital Data Acquisition and Analysis

Elise Jortberg
Lab Partner: Dena Guo
Feb. 11, 2016

1. INTRODUCTION

The purpose of this lab is to acquire waveforms at various frequencies from a function generator through LabWindows/CVI for data analysis. The data acquisition (DAQ) card interfaces the function generator to the computer. The DAQ converts the analog signal from the function generator to a digital signal that the computer can then read and store. The analysis of this collected data follows the fundamental principles of signal processing. Using LabWindows/CVI's spectrum function, we performed a Fast Fourier Transform (FFT) on the data to visualize both the time and frequency domains.

We built a program that required user inputs of acquisition time and frequency at which the DAQ card would sample the analog signal. We observed the original signal and the FFT at 1 kHz to confirm that the FFT was working as expected and then varied the input frequency and type of waveform (sine, square, triangle) on the function generator. Finally, we captured the inputted waveform at below and above the calculated Nyquist frequency to quantify the effects of aliasing. When Nyquist frequency is below the frequency of the inputted waveform, we witnessed significant information loss, as the input frequency appeared lower than the true value.

2. MATERIALS AND METHODS

2.1 Hardware

To generate the analog waveform, we used a GWinstek function generator [GFG-82192]. The function generator connected to the DAQ card [National Instruments CB-68LP], which acts as an analog to digital converter (ADC), via a standard PNC cable. The DAQ card fed data to the computer through a USB cable. We created a virtual channel in the LabWindows/CVI framework to configure the DAQ card so that it would read, sample, and store the waveform based on user-inputted parameters.

2.2 Software

Our program consisted of three primary functions: data acquisition, the FFT, and output of time and frequency data streams. Figure 2 (in Results) displays the user interface of our program. The user sets the acquisition time and acquisition frequency. Pressing acquire data triggers the program to collect the waveform, automatically perform the FFT, and visualize the results on the graphical displays.

2.2.1 Data Acquisition

The DAQ discretely samples the continuous waveform to create a digital signal that the computer requires. This process is illustrated in Figure 1 [1].

The size (N) of the saved array is

$$N = r * t \qquad (1)$$

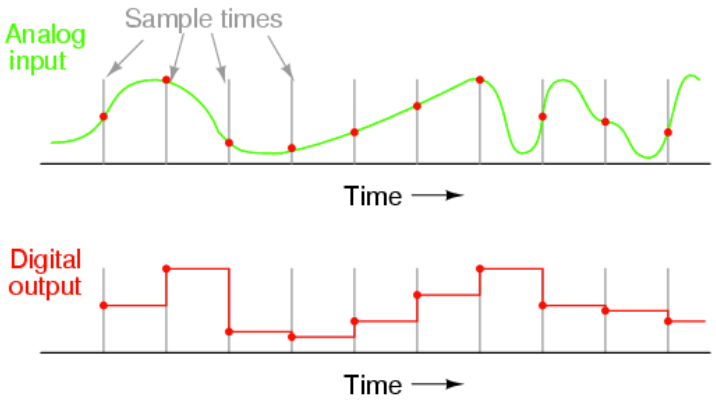


Figure 1: Analog to Digital Conversion

elements long, where r is the sampling rate (in Hz) and t is the length (in seconds) of acquisition. DAQmxReadAnalogF64 reads in and saves the voltage array. We then save the corresponding time and frequency information to separate arrays:

```
for (i = 0; i < numSamples; i++)
        {
            timeAq[i] = i * (inputAcquisitionTime / numSamples);
            frequency[i] = i * (1 / inputAcquisitionTime);
}
```

This voltage signal as a function of time is then inputted into the FFT.

2.2.2 The Fast Fourier Transform (FFT)

The FFT converts a signal from the time domain to the frequency domain. We represent input signal $\mu(t)$ as a summation of sines and cosines with a weighting function, $\mu(\omega)$ that is proportional to the dominant frequencies of the signal. Symmetrically, $\mu(\omega)$ is the integral over all space of $\mu(t)$ times the inverse relation of the input:

$$\mu(t) = \Sigma\mu(\omega)e^{it} \quad (2)$$

$$\mu(\omega) = \frac{1}{2\pi}\int_{-\infty}^{\infty} u(t)e^{-i\omega t}dt \quad (3)$$

We use the LabWindows/CVI function Spectrum, which computes the "power spectrum of the input array" according to the following formula [2]:

$$ps = \frac{|FFT(X)|^2}{n^2} \quad (4)$$

Spectrum returns the transformed data, which can then be saved for future analysis.

2.2.3 Output

The user has the option to save the time domain and frequency domain data of their generated signal. Save Time Data organizes the time and voltage arrays, then uses the ArrayToFile function that writes the data to a tab delimited text file:

```
for (i=0; i<numSamples; i++){
outputarray_time[i] = timeAq[i];
}
for (i=numSamples; i<(2*numSamples); i++){
outputarray_time[i] = voltage[i-numSamples];
}

//Write New File
ArrayToFile (timefilename, outputarray_time, VAL_DOUBLE,
2*numSamples, 2,
VAL_GROUPS_TOGETHER, VAL_GROUPS_AS_COLUMNS,
VAL_SEP_BY_TAB, 10, VAL_ASCII, VAL_APPEND);
```

We use the same logic to save the frequency and FFT data when the user selects Save Frequency Data.

3. RESULTS

We collected sinusoidal, square, and triangular waveforms. Figure 2 shows the result of collection 1, where the function generator was set to 5 Hz, the sampling frequency 40 Hz, and the acquisition time was 2 seconds. We set the sampling frequency

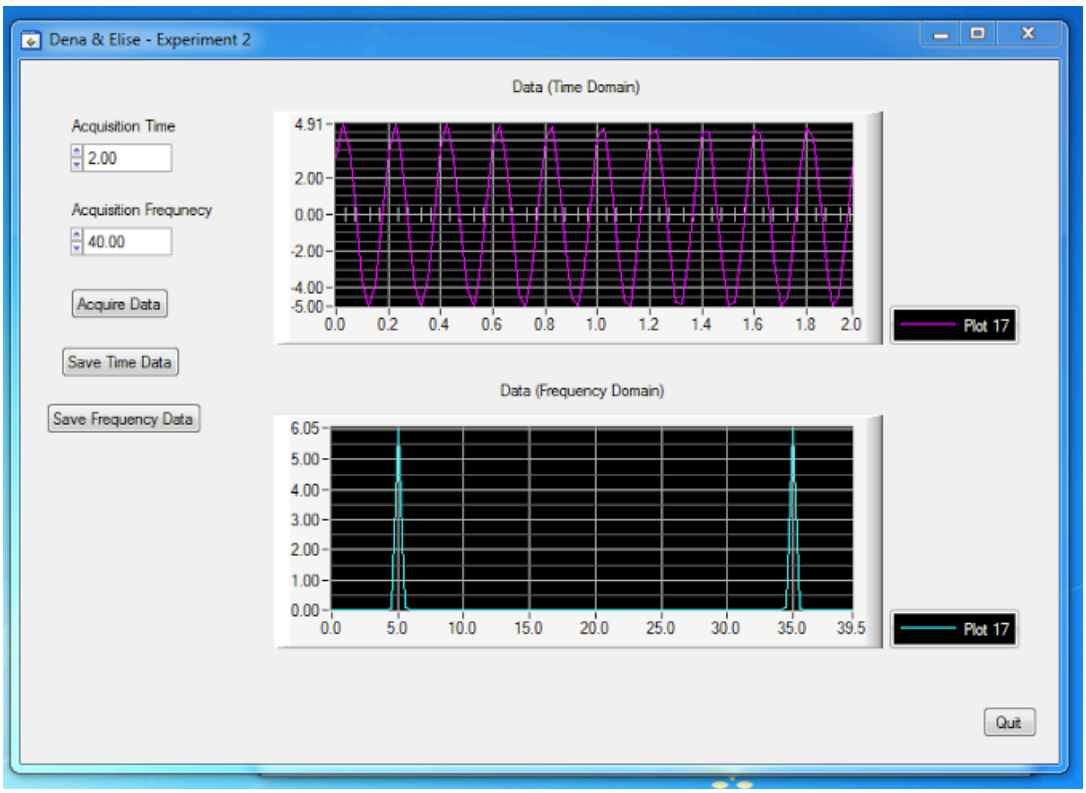


Figure 2: Program GUI. Input: sinusoid, freq=5Hz

to be significantly greater than the wave frequency so we would lose no information. As expected, at 5 Hz collected for two seconds, we see 10 peaks in the first graph. The program successfully calculated the FFT, which has a peak at 5 Hz. This process was repeated with extended Acquisition Times to capture more of the signal. These signals and their FFT's are shown in the Appendix. The Appendix also provides examples of plots created through MatLab and Excel using the saved time and frequency domain data.

The mirror image of the frequency at 35 Hz is an artifact of the Discrete Fourier Transform (DFT) used for the FFT. The transform of the sinusoid

$$V_k = \Sigma v_n e^{-2\pi nk/N} = e^{2*\pi*i*n*(N-fo)/N} \qquad (5)$$

where $v_n = sin(2\pi*f_o*n/N)$ (6)

and N = acquisition frequency. Thus, there is an image of the signal at N-k_o, which indicates directionality of the true frequency on the sampling frequency range. The image therefore exists at N-f_o (in this case 40Hz – 5Hz). The image is erect because the power spectrum takes the square of the FFT.

Figure 3 shows a square wave and its resultant FFT at 5 Hz. The square wave is a summation of sine waves.

$$V(t) = \Sigma(1/n)\ sin(2\pi f_o nt)\ where\ n=odd \qquad (7)$$

The FFT of the square wave will have images that decrease by magnitude of the square root of power.

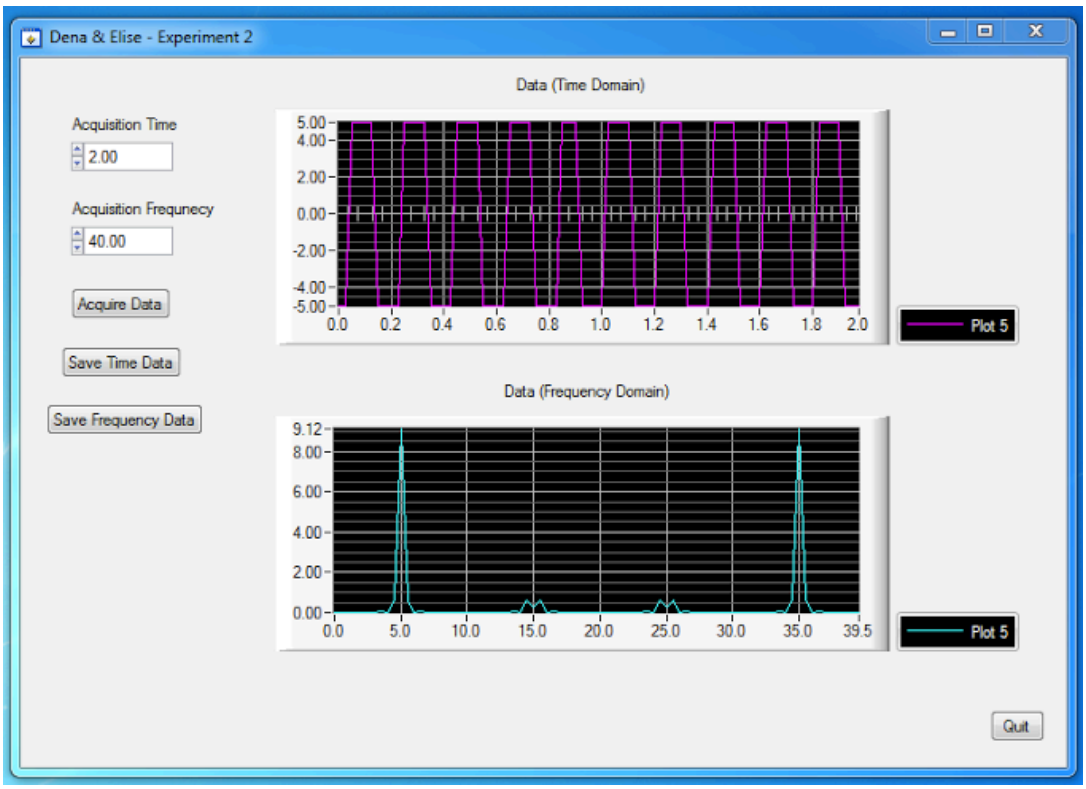


Figure 3: Square wave, input freq=5 Hz

Finally, we collected a triangle wave also at 5 Hz that again shows the mirror image of the input frequency.

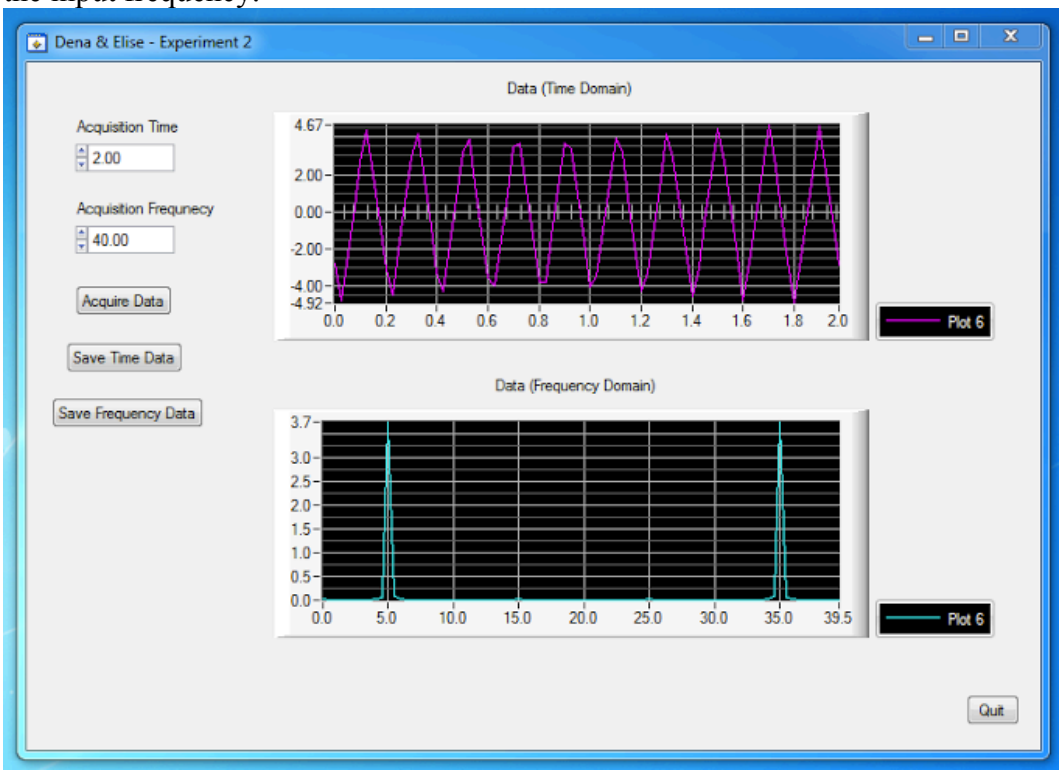


Figure 4: Triangle Wave, freq= 5Hz

The highest frequency a user can detect is inherently limited by the sampling frequency. If the sampling frequency is not higher than at least twice the input frequency, the ADC will drop necessary data points from the original signal. As a result, the digital signal cannot accurately reconstruct the original signal. For cyclical waveforms, the loss of information results in aliasing. When aliasing occurs, the frequency displayed in the FFT will be lower than the true frequency. To accurately represent the waveform, the sampling frequency must be greater than or equal to twice the input frequency. This frequency is called the Nyquist frequency:

$$f_{Nyquist} = \frac{1}{2\tau} \qquad (8)$$

To test the Nyquist frequency, we generated a signal at 20, 30, and 40 Hz and then set the sampling frequency also to 40 Hz. For the 20 Hz signal the FFT correctly identifies the input frequency. However, the amplitude information has been lost.

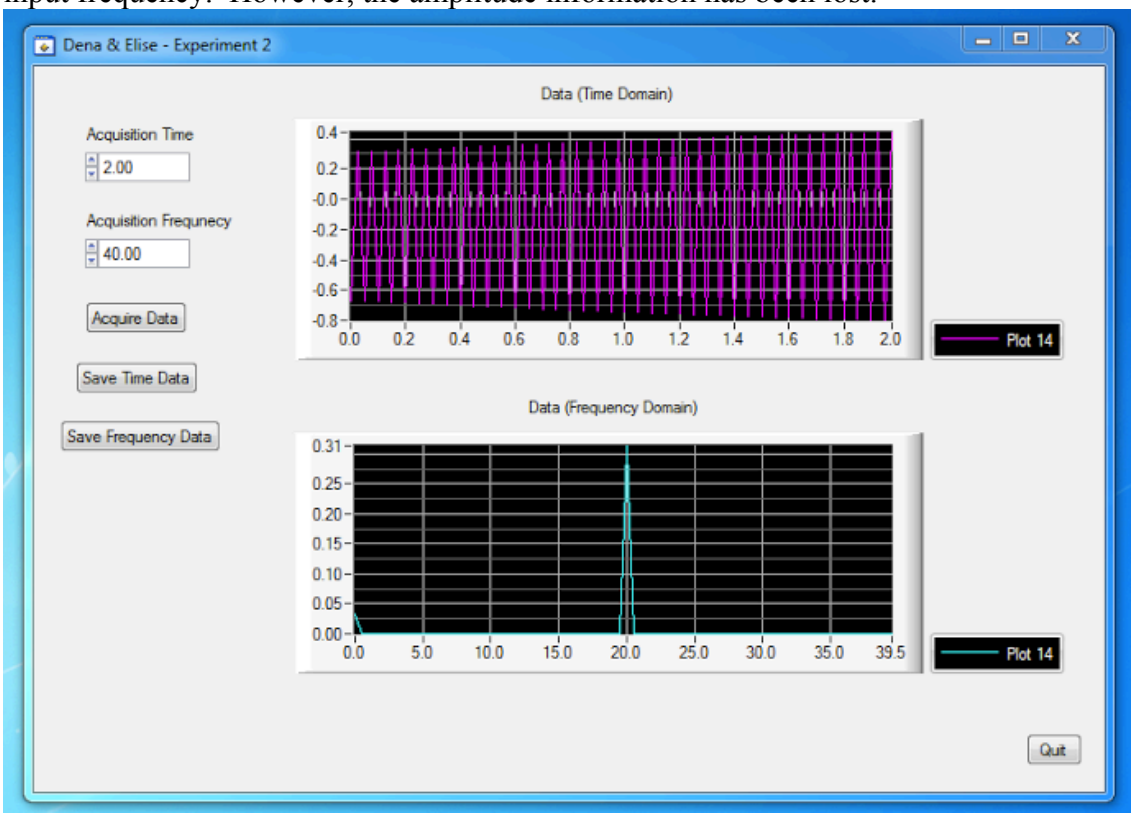


Figure 5: Sinusoid, Input freq = 20 Hz

However, at 30 Hz, the FFT identifies the input frequency as 10 Hz instead of 30 Hz. Note: the mirror image does show 30 Hz, however this is just a coincidence as $N\text{-}f_{calculated}$ = 40-10 = 30 Hz.

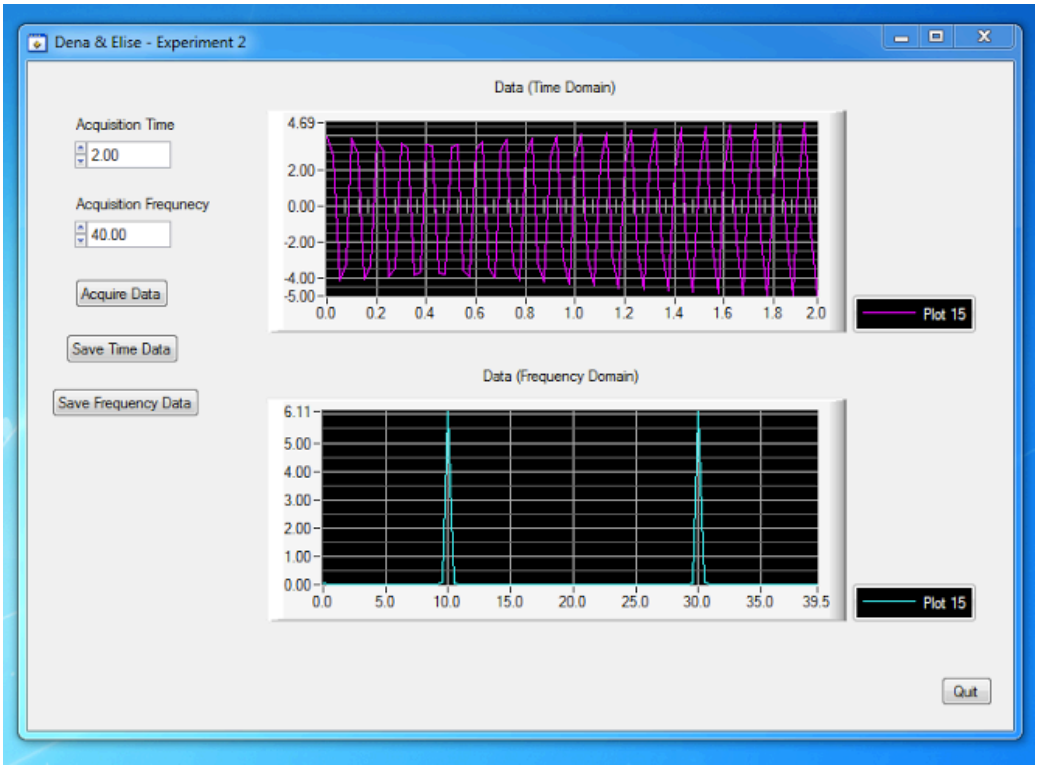


Figure 6: Sinusoid, Input freq = 30 Hz

When input frequency is set to 40 Hz, the time domain plot shows a sinusoid that is clearly at a frequency far lower than 40 Hz. The resultant FFT, shown in Figure 7, calculates the Frequency to be at 0.5 Hz, far lower than the true value of 40 Hz.

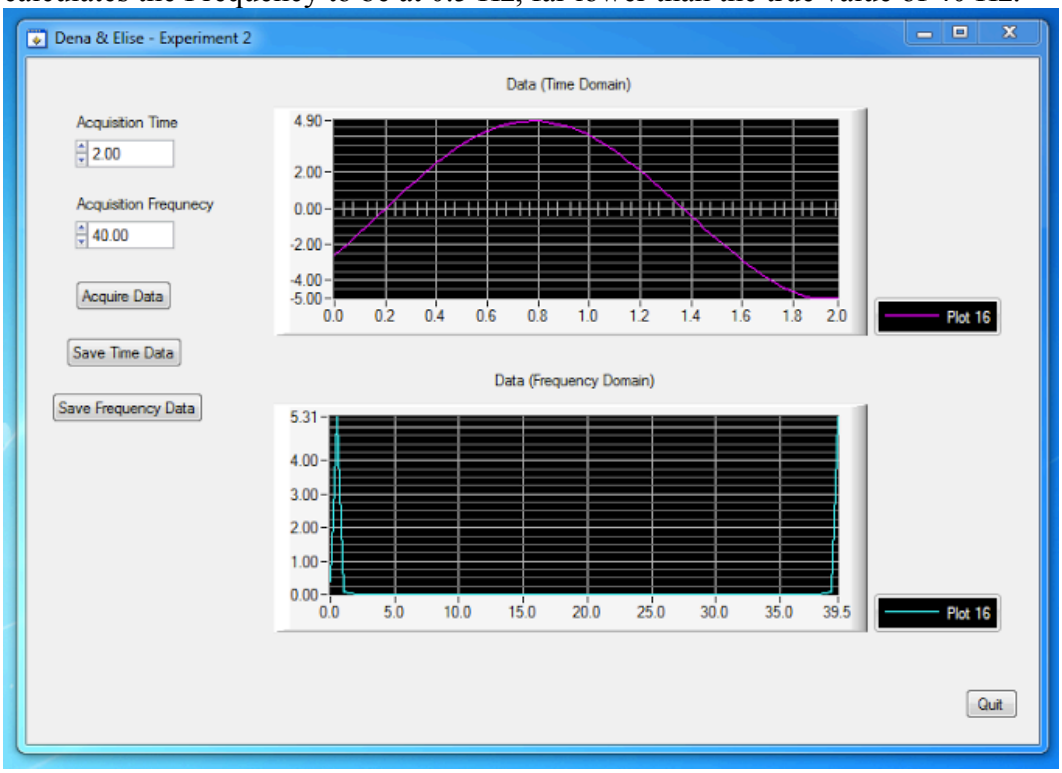


Figure 7: Sinusoid, Input freq=40 Hz

According to *(8)*, we need a sampling frequency of at least 80 Hz to capture the input frequency of 40 Hz. Due to this principle, we set all of our sampling frequencies to a much higher value than our input frequency in the prior examples to avoid aliasing.

4. DISCUSSION

We were able to generate a variety of waveforms, display the collected data, and display the data in the frequency domain. The Spectrum function calculated the power of the signal; therefore, our frequency domain plot showed the rectified mirror image (magnitude of imaginary portion of the result created by the FFT *(3)*) of the calculated input frequency. We saw for sinusoids and triangle waves, the mirrors appeared as spikes in equal height, where as for square waves the mirrors had a height proportional to the square root of power from the extra factor of 1/n in V(t) (7).

By altering the sampling frequency to below and above the input frequency, we were able to compare the collected signals. For signal processing, the Nyquist frequency is the necessary frequency to resolve the input waveform's frequency with Fourier analysis. Signals collected below the Nyquist frequency showed the effects of aliasing. In our example, the Nyquist frequency was 40 Hz. The Nyquist could resolve an input frequency of up to 20 Hz, however it could not resolve input signals of 30 and 40 Hz. The FFT of signals collected below Nyquist frequency calculated frequencies below the actual frequency because the data capture did not have enough data points to represent the true, inputted signal.

5. CONCLUSIONS

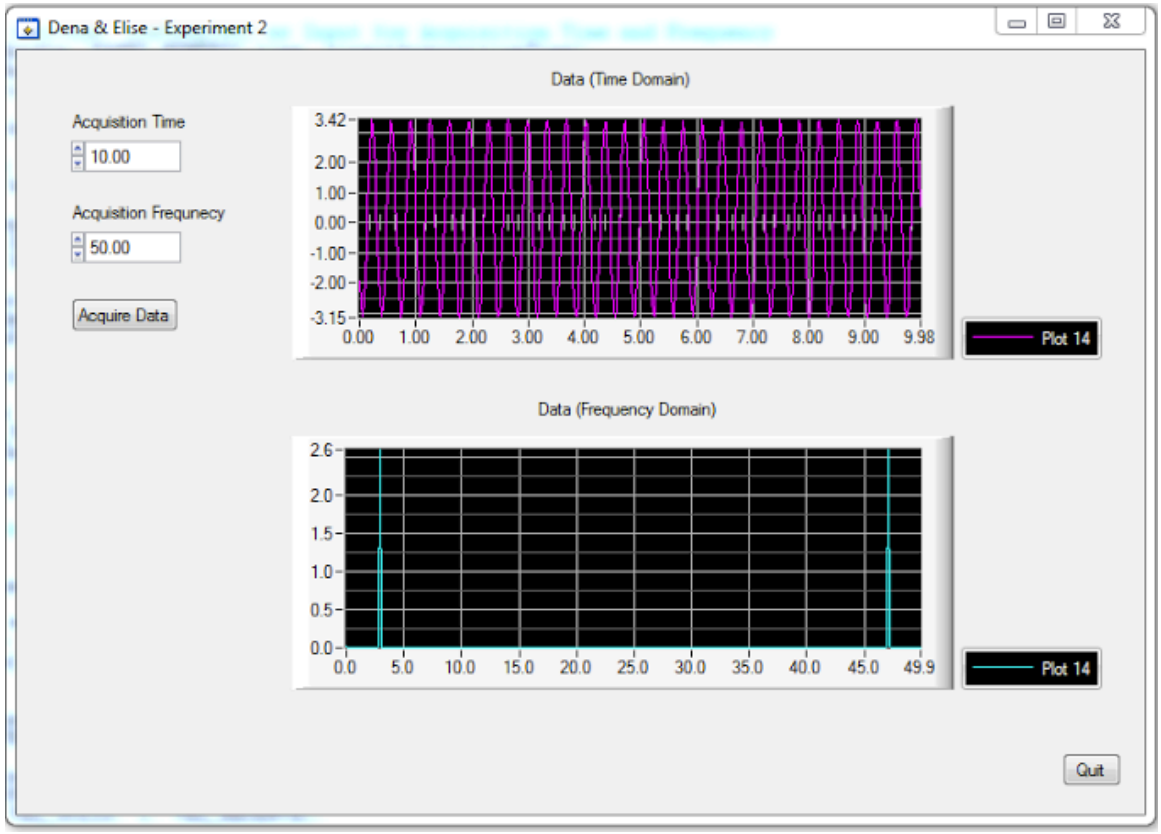
The results calculated by our program verified our expectations of the FFT. We collected input signals across multiple waveforms and frequencies created by the function generator. The DAQ card transcribed this analog signal into discrete values that were then read in and analyzed by our program. We collected three waveforms (sinusoid, square, and triangle) and their resultant transforms in frequency space. We then incrementally increased our input frequency while setting a Nyquist sampling frequency of 40 Hz. Once the input frequency rose past 20 Hz, we witnessed aliasing in both the time and frequency domains. This lab reaffirms the importance in signal processing of having a large enough sampling frequency to accurately reflect the real, physical process.

BIBLIOGRAPHY

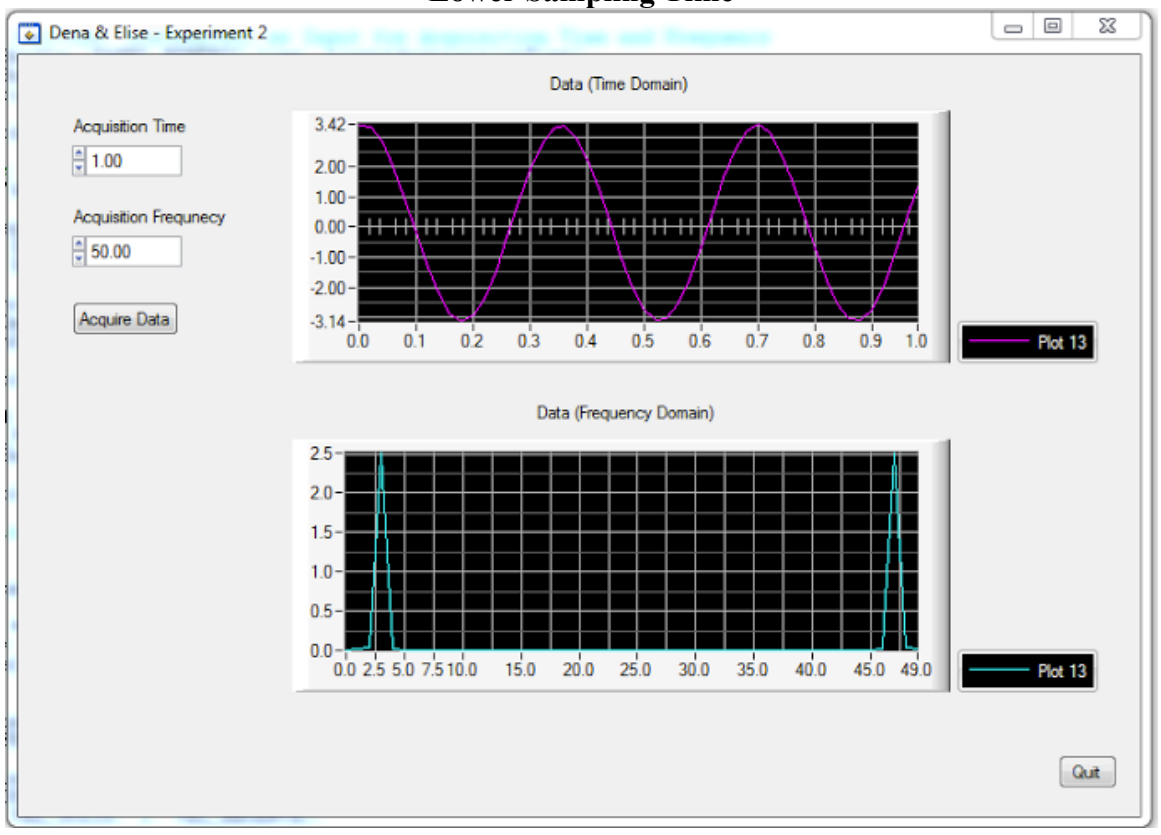
[1] Lessons in Electric Circuits: Digital Analog Conversion. (C) 2000-2015 Tony R. Kuphaldt. https://www.ibiblio.org/kuphaldt/electricCircuits/Digital/DIGI_13.html
[2] Spectrum function description. NSI. http://zone.ni.com/reference/en-XX/help/370051V-01/cvi/libref/cvispectrum/
[3] ADC Spec Sheet. http://www.ni.com/pdf/manuals/373853a.pdf

APPENDIX
Figures

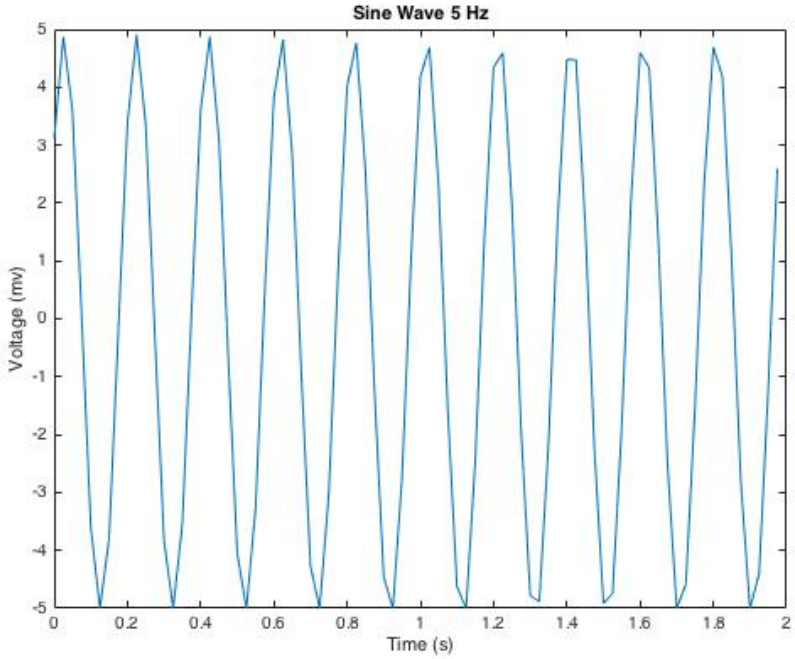
Higher Acquisition Time

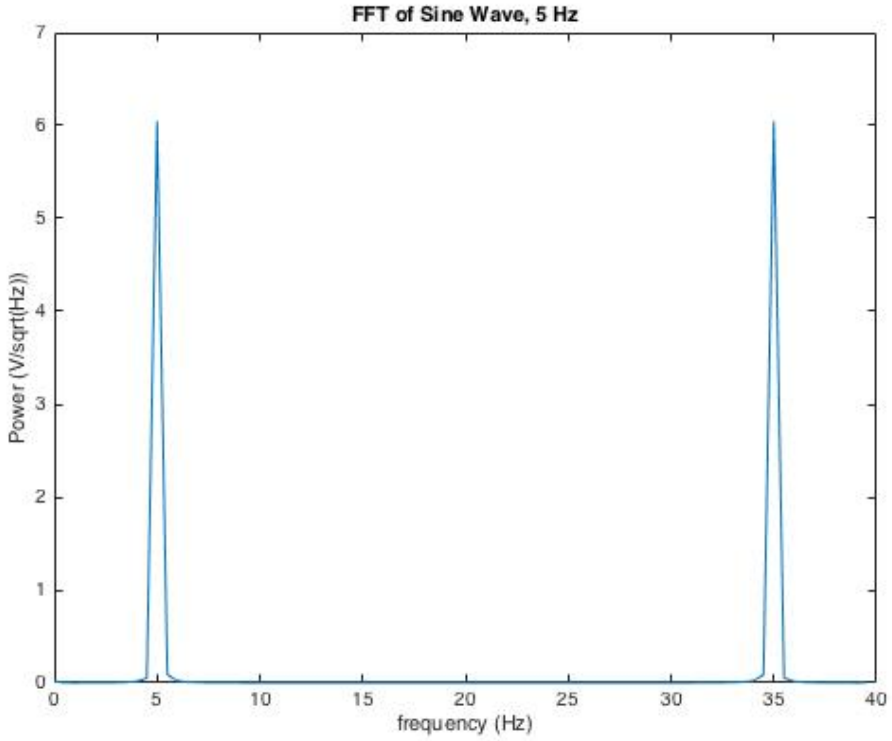


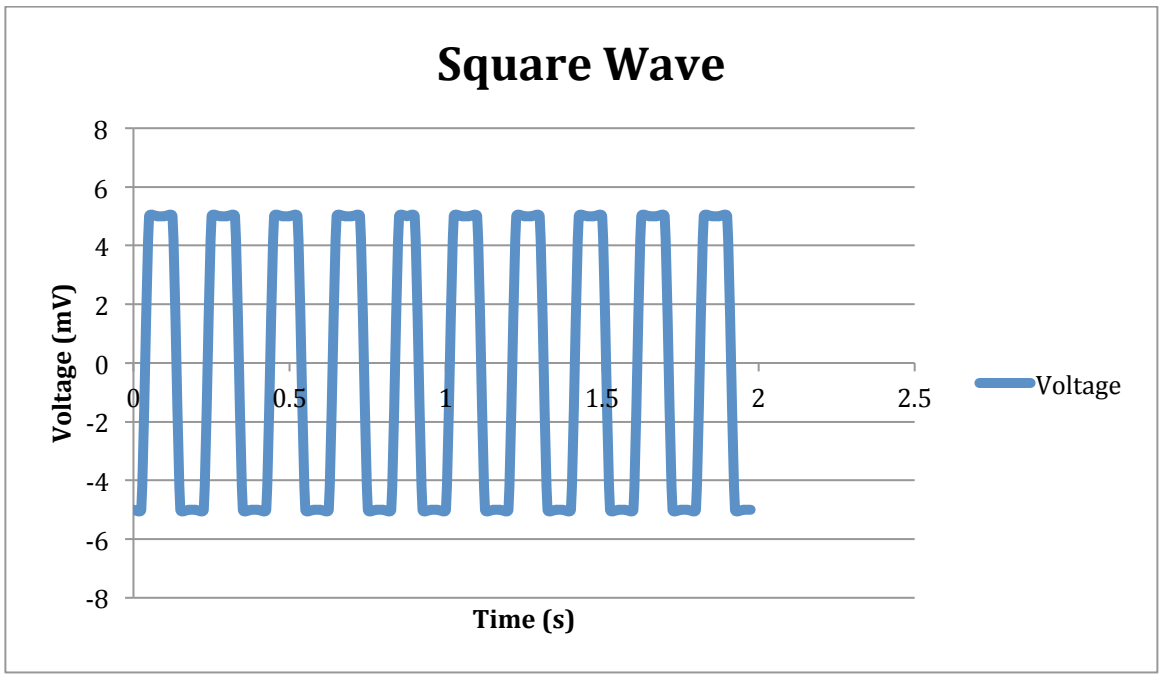
Lower Sampling Time

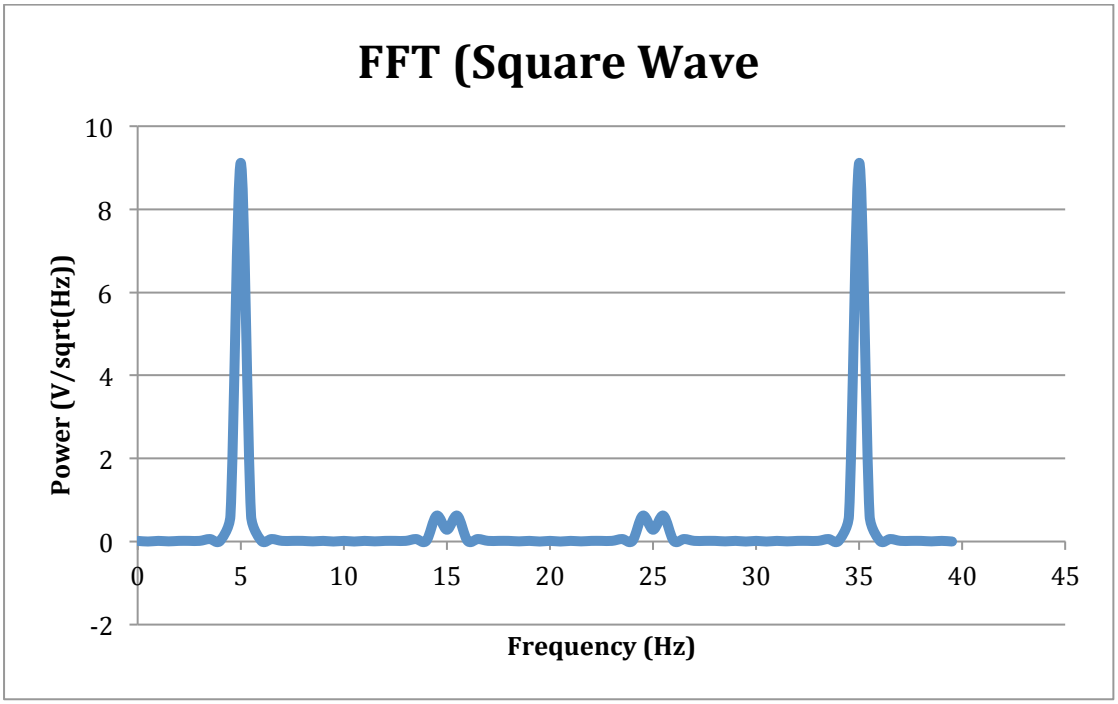


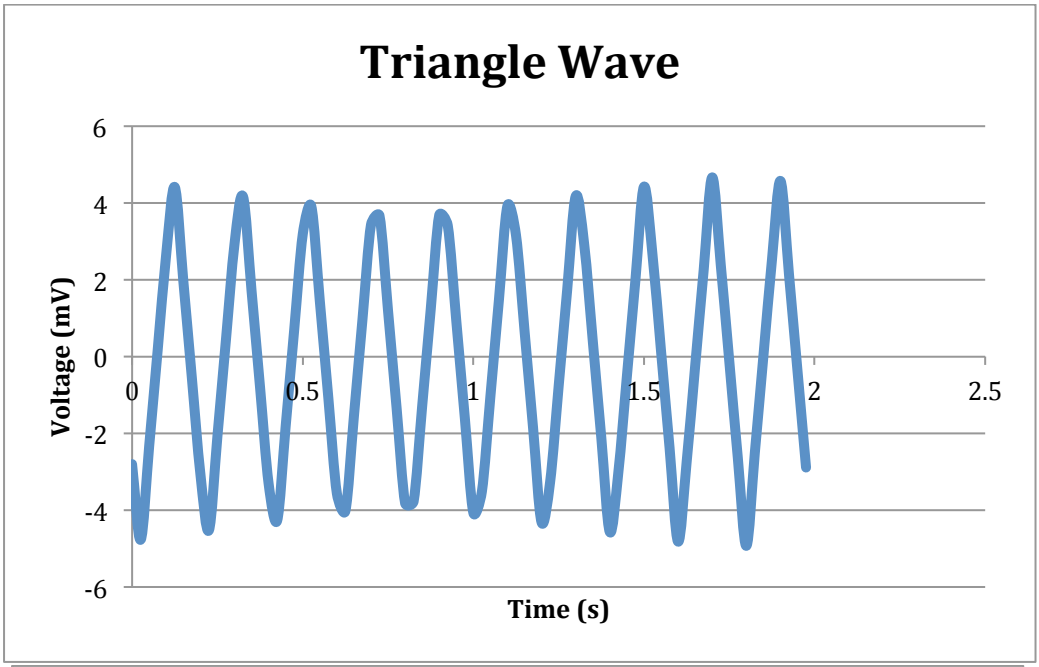
Plot created using outputted text files

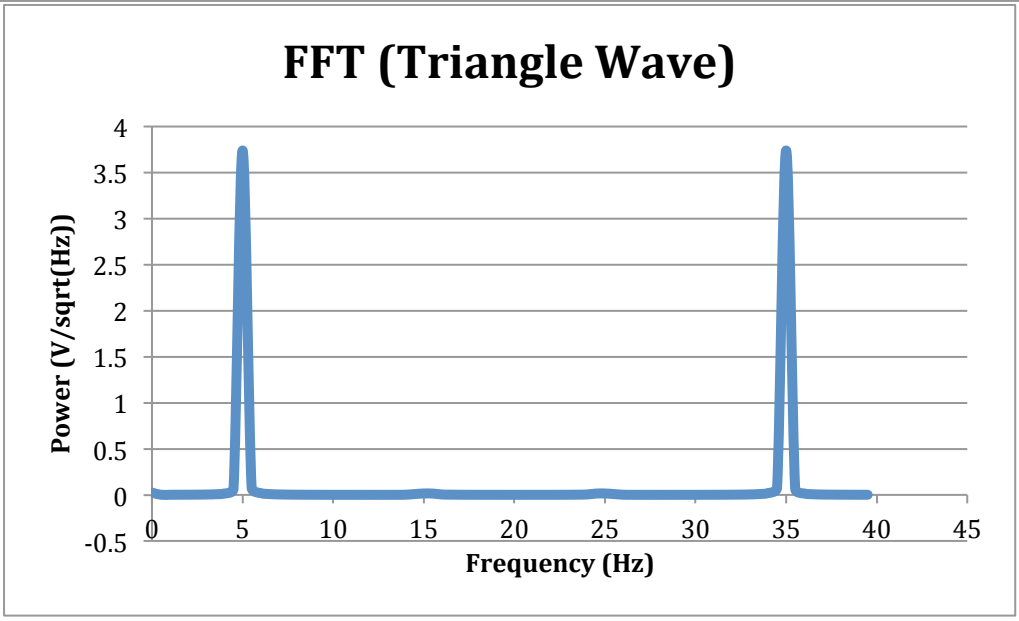





Square Wave
8
6
4
2
0
-2
-4
-6
-8
Voltage (mV)
0
0.5
1
1.5
2
2.5
Time (s)
Voltage


FFT (Square Wave
10
8
6
4
2
0
-2
Power (V/sqrt(Hz))
0
5
10
15
20
25
30
35
40
45
Frequency (Hz)





CODE

```
#include <formatio.h>
#include <ansi_c.h>
#include <analysis.h>
#include <NIDAQmx.h>
#include <cvirte.h>
#include <userint.h>
#include "Experiment 2 Data Acquisition.h"
#include <stdio.h>


// Declare Global Variables
static TaskHandle voltageHandle;
```

```
static int panelHandle;
static double inputAcquisitionTime;
static float64 inputAcquisitionFreq;
static float64 voltage[1000000];
static uInt64 numSamples;
static double timeAq[1000];
static double frequency[1000];
static double fouriervoltage[1000];
static double outputarray_time[10000];
static double outputarray_fft[10000];

int main (int argc, char *argv[])
{
    if (InitCVIRTE (0, argv, 0) == 0)
        return -1;  /* out of memory */
    if ((panelHandle = LoadPanel (0, "Experiment 2 Data
Acquisition.uir", PANEL)) < 0)
        return -1;
    DisplayPanel (panelHandle);
    RunUserInterface ();
    DiscardPanel (panelHandle);



    return 0;
}

int CVICALLBACK acquireData (int panel, int control, int event,
                             void *callbackData, int eventData1, int
eventData2)
{
    switch (event)
    {
        case EVENT_COMMIT:

        // Get UsSpectrum (voltage, numSamples);er Input for
Acquisition Time and Frequency
        GetCtrlVal (panelHandle, PANEL_NUMERIC_time,
&inputAcquisitionTime);
        GetCtrlVal (panelHandle, PANEL_NUMERIC_frequency,
            &inputAcquisitionFreq);

        numSamples = inputAcquisitionTime * inputAcquisitionFreq;

        // Create DAQ Task
        DAQmxCreateTask ("Acquire Data", &voltageHandle);

        DAQmxCreateAIVoltageChan (voltageHandle, "Dev1/ai1", "Analog
2",
                          DAQmx_Val_Cfg_Default, -5.0, 5.0,
                          DAQmx_Val_Volts, "");

        DAQmxCfgSampClkTiming (voltageHandle, "OnboardClock",
inputAcquisitionFreq,
                       DAQmx_Val_Rising, DAQmx_Val_FiniteSamps,
                       numSamples); //keeps time
```

```
        DAQmxStartTask (voltageHandle); //loads task with specified
settings

        DAQmxReadAnalogF64 (voltageHandle, DAQmx_Val_Auto, 10.0,
                    DAQmx_Val_GroupByChannel, voltage,
                    2*numSamples, &numSamples, 0); //reads analog
voltage into readVoltageArray

        DAQmxClearTask (voltageHandle); //stops acquiring data

        // Plot Time Domain Graph
        int i;

        for (i = 0; i < numSamples; i++)
        {
            timeAq[i] = i * (inputAcquisitionTime / numSamples);
            frequency[i] = i * (1 / inputAcquisitionTime);
            fouriervoltage[i] = voltage[i];
        }

        DeleteGraphPlot (panelHandle, PANEL_GRAPH_time, -1,
                 VAL_DELAYED_DRAW); //delete any previous graphs

        PlotXY (panelHandle, PANEL_GRAPH_time, timeAq, voltage,
        numSamples, VAL_DOUBLE, VAL_DOUBLE, VAL_THIN_LINE,
        VAL_EMPTY_SQUARE, VAL_SOLID, 1, VAL_MAGENTA);

        // Transform into Frequency Domain and Plot Graph
        Spectrum (fouriervoltage, numSamples);

        DeleteGraphPlot (panelHandle, PANEL_GRAPH_frequency, -1,
                 VAL_DELAYED_DRAW); //delete any previous graphs

        PlotXY (panelHandle, PANEL_GRAPH_frequency, frequency,
fouriervoltage,
        numSamples, VAL_DOUBLE, VAL_DOUBLE, VAL_THIN_LINE,
        VAL_EMPTY_SQUARE, VAL_SOLID, 1, VAL_CYAN);

            break;
        case EVENT_RIGHT_CLICK:

            break;
    }
    return 0;
}

int CVICALLBACK Bye (int panel, int control, int event,
                     void *callbackData, int eventData1, int
eventData2)
{
    switch (event)
    {
        case EVENT_COMMIT:

            QuitUserInterface (0);

            break;
        case EVENT_RIGHT_CLICK:
```

```
            break;
    }
    return 0;
}

int CVICALLBACK Save_Time (int panel, int control, int event,
                           void *callbackData, int eventData1, int
eventData2)
{
    switch (event)
    {
        case EVENT_COMMIT:

            // Create File to Save
            char timefilename[300];

            FileSelectPopup ("", "*.*", "", "", VAL_SAVE_BUTTON, 0, 0,
1, 1,
                 timefilename);

            // Write Time and Frequency Arrays in a Single Output Array
            int i=0;


            for (i=0; i<numSamples; i++){
                outputarray_time[i] = timeAq[i];
            }
            for (i=numSamples; i<(2*numSamples); i++){
                outputarray_time[i] = voltage[i-numSamples];
            }

            // Write New File
            ArrayToFile (timefilename, outputarray_time, VAL_DOUBLE,
2*numSamples, 2,
             VAL_GROUPS_TOGETHER, VAL_GROUPS_AS_COLUMNS,
             VAL_SEP_BY_TAB, 10, VAL_ASCII, VAL_APPEND);

            break;
        case EVENT_RIGHT_CLICK:

            break;
    }
    return 0;
}

int CVICALLBACK Save_FFT (int panel, int control, int event,
                          void *callbackData, int eventData1, int
eventData2)
{
    switch (event)
    {
        case EVENT_COMMIT:

            // Create File to Save
            char fftfilename[300];
            FileSelectPopup ("", "*.*", "", "", VAL_SAVE_BUTTON, 0, 0,
1, 1,
```

```
                fftfilename);

            // Write FFT Arrays into a Single Output Array
            int i=0;

            for (i=0; i<numSamples; i++){
                outputarray_fft[i] = frequency[i];
            }
            for (i=numSamples; i<(2*numSamples); i++){
                outputarray_fft[i] = fouriervoltage[i-numSamples];
            }

            ArrayToFile (fftfilename, outputarray_fft, VAL_DOUBLE,
2*numSamples, 2,
             VAL_GROUPS_TOGETHER, VAL_GROUPS_AS_COLUMNS,
             VAL_SEP_BY_TAB, 10, VAL_ASCII, VAL_APPEND);

            break;
        case EVENT_RIGHT_CLICK:

            break;
    }
    return 0;
}
```