Digital Oscilloscope and Spectral Analyzer

Swapnil Acharya

ECE 421

5/4/2019

Contents

[Introduction: 3](#_Toc8246163)

[Design and Implementation: 6](#_Toc8246164)

[Results and Analysis: 28](#_Toc8246165)

[Contemporary: 40](#_Toc8246166)

[Conclusion: 40](#_Toc8246167)

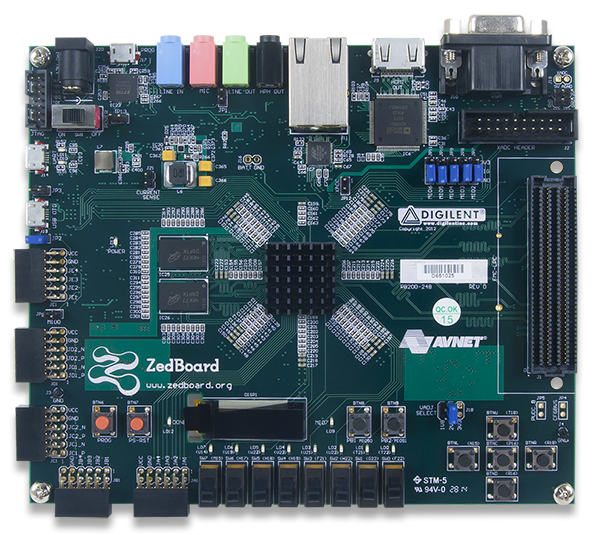
[Acknowledgement: 40](#_Toc8246168)

[Reference: 41](#_Toc8246169)

# Introduction:

1. Background:

This project is the Final Project of ECE 421 Computer Architecture and Design which is designed to test the knowledge of System on Chip (SoC) design. The project focuses on using preexisting and designing custom Intellectual Property (IP) in Verilog and VHDL language. The labs from the class gave core knowledge of timer interrupts, ADC, sampling, led control, GPIO control which were very essential in the development of this project. SoC design nowadays are foundation of majority of digital systems available today, so this project will help understand how SOC are programmed and used in Industry. The SoC used in project is Zedboard. ZedBoard is a complete development kit for designers interested in exploring designs using the Xilinx : Zynq® -7000 All Programmable SoC. The board contains all the necessary interfaces and supporting functions to enable a wide range of applications [1].

[](https://www.google.com/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=2ahUKEwid-tDal4XiAhUFX60KHRUfAJYQjRx6BAgBEAU&url=https://reference.digilentinc.com/reference/programmable-logic/zedboard/start&psig=AOvVaw2O1efnoZ0iXXBza8VmVdTo&ust=1557173024337962)

Zedboard Top View [2]

1. Objective

The objective of this project is to design and develop a function generator, a digital oscilloscope and a spectral analyzer using Zynq/Zed board and C# Graphical User Interface (GUI).

1. Components and Devices used

Zedboard, Personal Computer (PC) and Oscilloscope were the devices used in this project. The components used in Zedboard Were Zynq 7000 SOC, XADC and UART bridge to connect to bridge. The Oscilloscope was used as function generator, which was connect to XADC pins in Zedboard as input. C# GUI was used to digitize samples read by Zedboard.

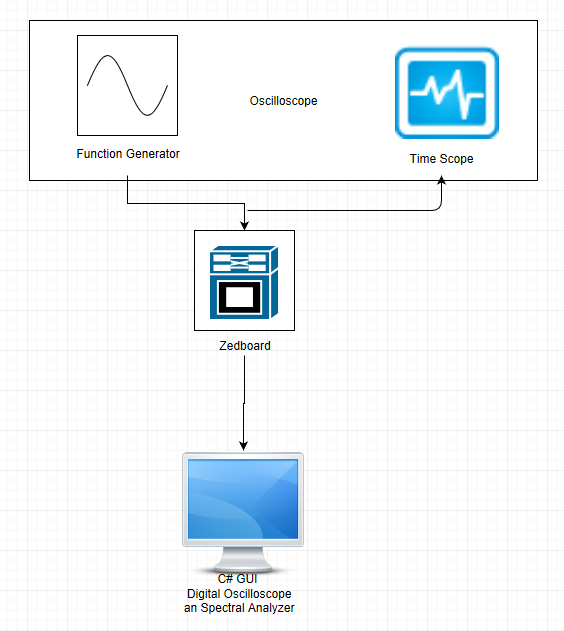
1. Design Specifications

The design specification for the project are listed below:

1. Using the procedure and method of Lab 4, implement the DAC for sinusoidal wave with controllable frequency from 100 Hz to 10kHz. There is a frequency entering box in the C# GUI to control the frequency of the generated sinusoidal signal. There is a button on the GUI to start the generation of the sinusoidal signal.
2. The sinusoidal signal (either from the DAC or a function generator), will be digitized by an ADC of the Zedboard using a single channel, using the procedure of Lab 7. There is an entering box for controlling the ADC sampling frequency. The sampling frequency can be selected from: maximum sampling frequency fs, fs/4, fs/16, fs/100, fs/1000 by using a time-span button on GUI. You may have more selections for sampling frequencies. You may digitize 256 to 512 samples for each plot. It is recommended to have 3 to around 10 cycles per plot. It is also recommended that the total sampling number N matches with the sampling frequency so that sampling frequency/N is an integer number (5%).
3. There is a GUI plot window to show the digitized samples. The plot on the C# GUI should be displayed in an order of a few frames per second or higher. The input signals in the range from 100 Hz to 10 kHz should be reliably displayed on the GUI continuously without delay and without distortion/discontinuity (35%).
4. The trigger value can be selected from 0.0 to 1.0 V, by using a GUI button or a sliding bar. (5%)
5. Magnitude span can be centered on 0.5 V with at least 4 scales: 2 (0.5±0.5/2), 8, 32, 100 (0.5±0.5/100), by using a GUI button (+, -), to show small signals. The setting of the scale should be displayed on GUI. You may have a continue span that allows more levels. (5%)
6. There is GUI plot for the magnitudes of the DFT of the sampled data. There is a button to select linear or dB scale of the plot (25%)
7. The horizontal and vertical axes must be properly labeled with correct units for the both plots (voltage, time, Hz). You need get the correct sampling frequency or sampling period to achieve this by using a timer. Sampling frequency must be displayed. (5%)
8. There is a quick measure button to display peak-to-peak value and frequency of the sampled data. The search of maximum and minimum is useful to determine the peak-to-peak value. The frequency may be obtained by the peak location of the DFT by searching from the 10th results to N/2 results of the DFT (try to avoid the value from DC) or using the period of cross 0.5 v for finding the frequency. (10%)
9. There are a start and stop buttons on the GUI to start the plots and stop the plot and sampling. (5%):
10. There will be a 5% deduction for any restart or repeated check off.
11. (NOT easy) DMA is used to transfer sampled data from PL to DDR, then the PS will access the DDR for the data transfer to the GUI (5%).

# Design and Implementation:

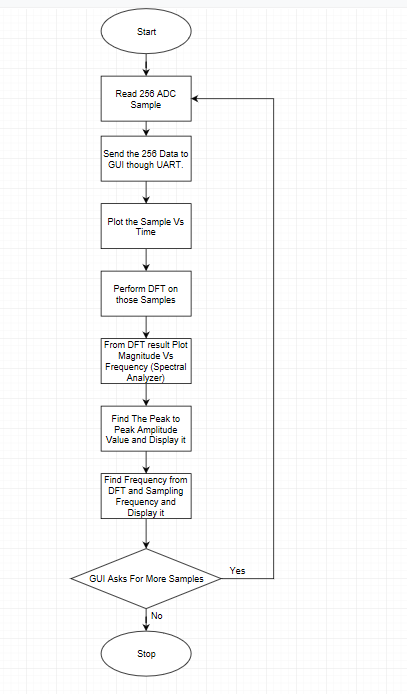
1. System Block Diagram



System Block Diagram

The picture above gives a general overview of the designed System. A Sinusoid signal is generator from Function generator in the Oscilloscope, which is fed both to the oscilloscope’s Time Scope and the XADC input pins in ZedBoard. The XADC IP in ZedBoard digitizes the analog input sinusoid then transmits the digitized samples to a C# via UART to USB bridge where these samples are plotted vs time which results in a Digital Oscilloscope. Discreet Fourier Transform (DFT) is also performed on the digitized samples and the magnitude data from DFT is Plotted vs Frequency which results in a Spectral Plot.

1. Overall Flow Chart of your code



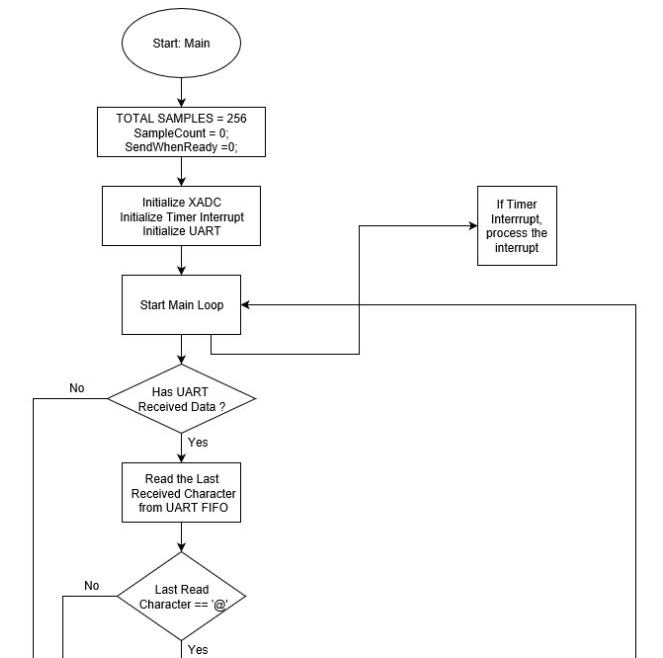
Overall Flowchart of Code

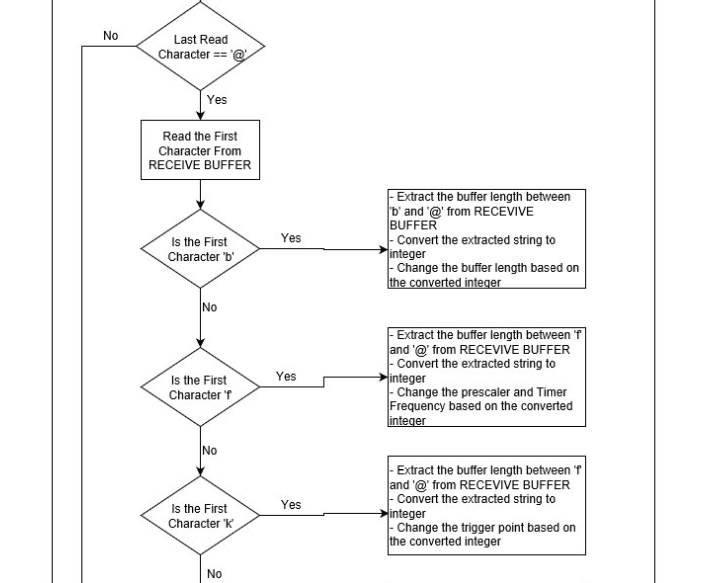
The flowchart above shows that the Hardware Side initially reads 256 samples and then those 256 digited samples are sent to C# Gui via UART to USB bridge. Once the C# GUI receives 256 samples it plots the samples vs time, performs DFT on the samples, the magnitude calculated from DFT is plotted vs frequency. The Amplitude and Frequency of the signal is calculated and displayed. Once this is done the GUI send a command to hardware side asking for another set of 256 samples. Then this process is repeated until the stop button is pressed on GUI.

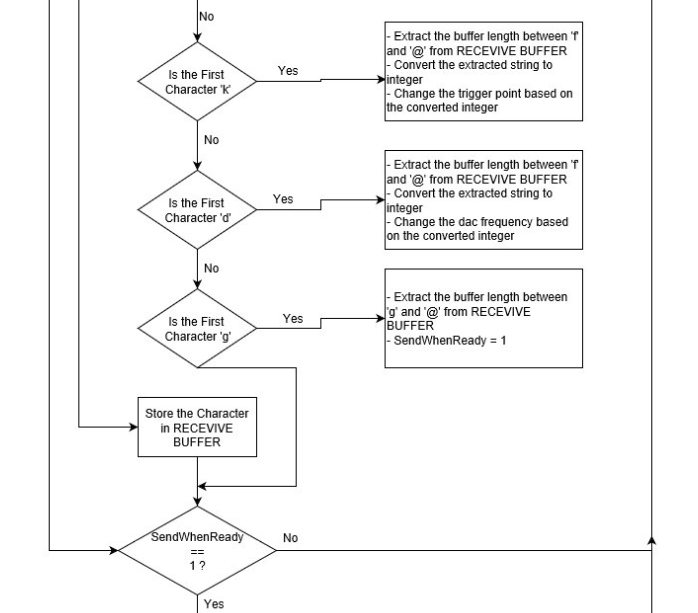
1. Detailed Flow Chart of your code   
   The Detailed flow chart of the code consists of two components, Flow chart of code on

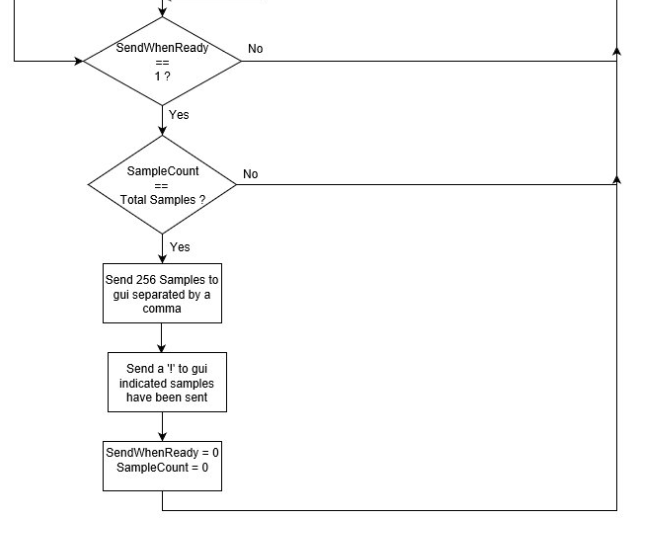
ZedBoard side and Flow chart of code on C# GUI Side.

ZedBoard Side (Flow Chart of Code)

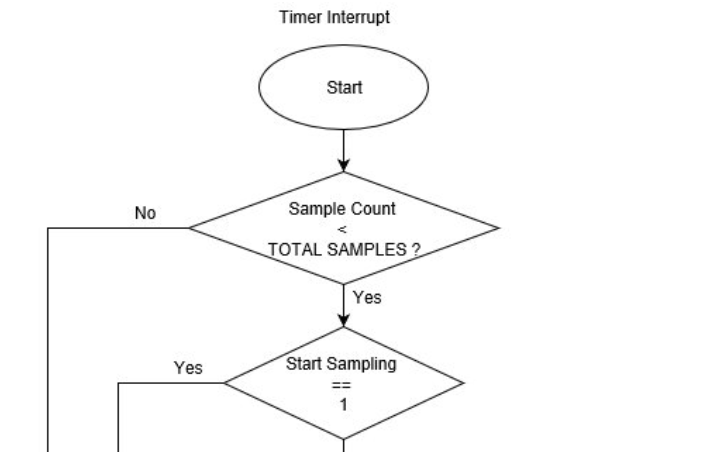


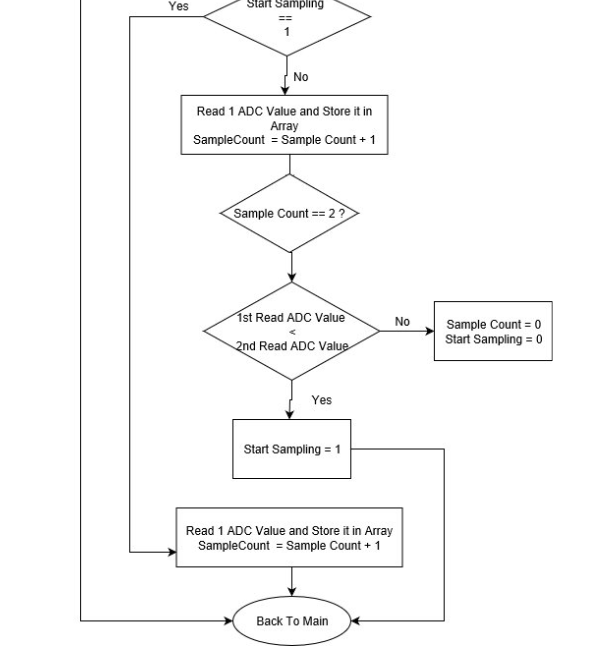






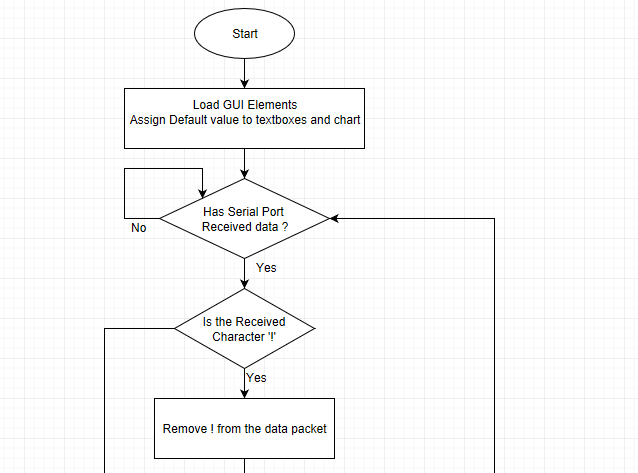
Timer Interrupt Code

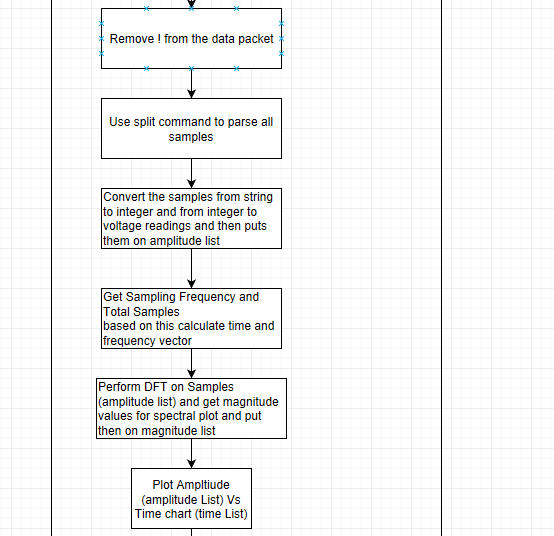


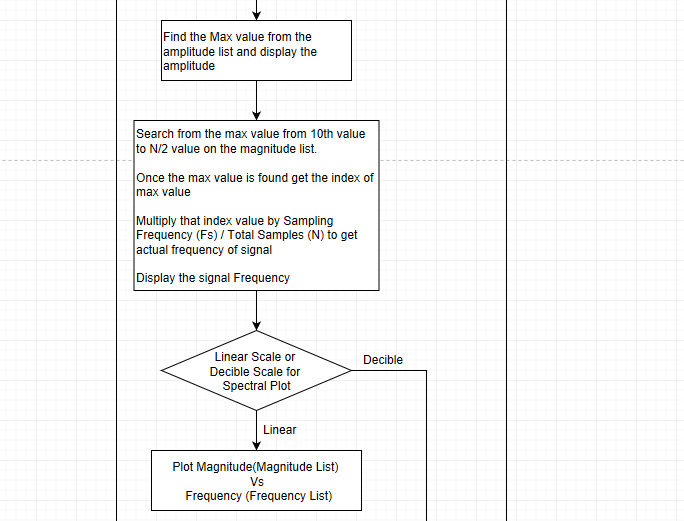


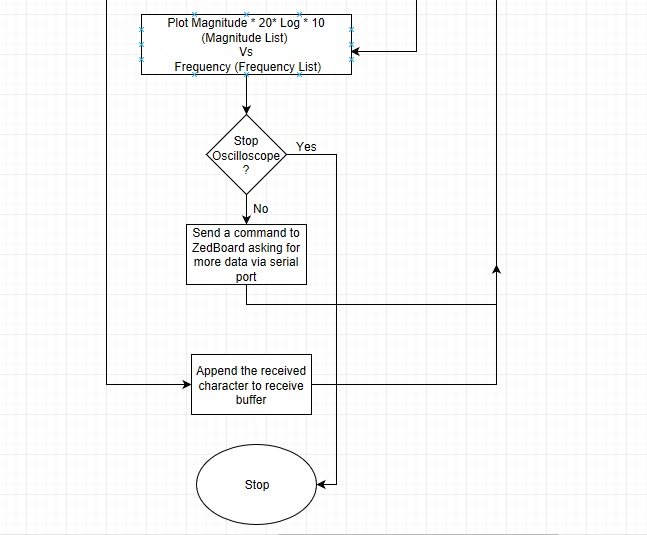
The ZedBoard side code Flow charts show how the system samples data and send the sampled data to Gui for further processing. The System Samples data in the time interrupt. Whenever timer interrupt happens the system samples data. In the Timer Interrupt the system first waits until the trigger point is found. Once the trigger point is found the system samples 2nd data and if the 2nd data is greater than the first data the system finally finds the trigger point and then starts sampling. The system samples data in timer interrupt until 256 samples are sampled. Once 256 samples are read and if the gui ready to receive data the system finally send those data to C# Gui via UART.

C# GUI Side (Flow Chart of Code)









Gui is setup on a Serial event trigger mode. Whenever the GUI receives any data in serial it reads the data and appends the data to receive buffer until a termination character, in this case “!”is read. Once this character is read the GUI parses the data convert the data from string to integer and then converts the raw data to voltage readings. Then puts the voltage readings on a list structure and using this list structure GUI produces time list, frequency list and then DFT magnitude list, then using all these lists the Gui plots a amplitude vs time, Magnitude vs frequency and displays peak to peak reading and signal frequency.

1. Description of GUI and RS232

The GUI designed for this project is designed using Microsoft Visual Studio 2019 Community Edition. C# is the Programming language used to design this Gui. The GUI was designed to visualize and perform spectral analysis on sampled XADC Data. The design GUI mimics an Oscilloscope thus the name digital Oscilloscope is given to the GUI. The GUI send commands to ZedBoard via Serial Port. The GUI receives data via serial port. Once data is received the GUI process data according to data packet. In this case, the gui parses the data packet and extracts the raw voltage readings, converts them to voltage readings and performs DFT on the data then plots time scope and spectral plot.

RS 232 is based on Serial Communication. In telecommunication, the process of sending data sequentially over a computer bus is called as serial communication, which means the data will be transmitted bit by bit [3]. The Serial Communication used in the Project uses UART Protocol. The Serial port on a PC is a type of device that uses an UART chip, a Universal Asynchronous Receiver Transmitter [3]. UARTs transmit data *asynchronously*, which means there is no clock signal to synchronize the output of bits from the transmitting UART to the sampling of bits by the receiving UART. Instead of a clock signal, the transmitting UART adds start and stop bits to the data packet being transferred. These bits define the beginning and end of the data packet so the receiving UART knows when to start reading the bits [3].

Once all the data Packet Termination character is received, the GUI uses Visual elements such as button to invoke delegates which are assigned to a function. Delegates basically acts like pointer to functions. Once the delegate is invoked the Gui triggers a certain function which then does all the required tasks such as updating Time Scope, Spectral Plot, peak to peak voltage, signal frequency and ask for more data from ZedBoard.

1. Design and code (with explanations, same in following texts) for digital oscilloscope

The GUI stores all the raw voltage readings in a structure and generate time vector based on Sampling frequency. The Time vector is generated by first fetching the Sampling Frequency (Fs) and then Getting Sampling Interval (Ts = 1/Fs). Then generating time vector at increments of Ts.

Then Finally Plotting the Tigger value Vs Time Vector, voltage readings VS Time Vector and then Finally displaying the Peak to Peak Value By finding Maximum Value in the Amplitude Vector.

1. data = parseResponse(data, "@", "!");
2. string[] allSamples = data.Split(',');
3. double Fs = getCurrentSamplingFrequency();
4. double Ts = 1 / Fs;

7. //define containers
9. //voltage samples
10. List<double> \_channel1Amplitude = new List<double>();
11. List<double> \_channel1Time = new List<double>();


15. //store amplitude and time values
16. double timeStart = 0;
17. double timeIncrement = Ts;
18. for (int i = 0; i < (allSamples.Length - 1); i++)
19. {
20. double val = Convert.ToDouble(allSamples[i]);
21. val = rawReadingToVoltage(val);
22. \_channel1Amplitude.Add(val);
23. \_channel1Time.Add(timeStart); //store time values
24. timeStart += Ts;
25. }



30. //clear chart
31. clearCharts();

34. //display peak to peak voltage
35. lbl\_y2.Text = \_channel1Amplitude.Max().ToString("##0.000");

38. //plot ampitude vs time chart
39. for (int i = 0; i < \_channel1Amplitude.Count; i++)
40. {
41. Chart1\_timeDomain.Series["ch1\_time"].Points.AddY( \_channel1Amplitude.ElementAt(i)); //plot time response
42. chart1\_timeDomain.Series["TRIGGER"].Points.AddY(Convert.ToDouble(tb\_triggerConvertedVoltage.Text));//plot trigger
43. }
45. Design and code for GUI buttons
46. private void exitUtilityToolStripMenuItem\_Click(object sender, EventArgs e)
47. {
48. serialPort1.Close();
49. Application.Exit();
50. }
52. private void connectToolStripMenuItem\_Click(object sender, EventArgs e)
53. {
54. openSerialPort();
55. MessageBox.Show("Connected");
56. }
58. private void disconnectToolStripMenuItem\_Click(object sender, EventArgs e)
59. {
60. serialPort1.Close();
61. MessageBox.Show("Disconnected");
62. }
64. private string parseResponse(string dataPacket, string token1, string token2)
65. {
66. int token1Index = dataPacket.IndexOf(token1);
67. int token2Index = dataPacket.IndexOf(token2);
68. string result = dataPacket.Substring(token1Index + 1, token2Index - token1Index - 1);
69. return result;
70. }
72. private void openSerialPort()
73. {
74. try
75. {
76. serialPort1.PortName = cb\_selectComPort.Text;
77. serialPort1.BaudRate = Int32.Parse(cb\_selectBaudRate.Text);
78. serialPort1.DataBits = Int32.Parse(cb\_selectDataBits.Text);
79. serialPort1.Handshake = Handshake.None; // no flow control
80. if (cb\_selectParity.Text == "Even")
81. {
82. serialPort1.Parity = Parity.Even;
83. }
84. else if (cb\_selectParity.Text == "Odd")
85. {
86. serialPort1.Parity = Parity.Odd;
87. }
88. else
89. {
90. serialPort1.Parity = Parity.None;
91. }
93. if (cb\_selectStopBits.Text == "0")
94. {
95. serialPort1.StopBits = StopBits.None;
96. }
97. else
98. {
99. serialPort1.StopBits = StopBits.One; //one stop bits
100. }
102. serialPort1.Open();
103. }
104. catch (Exception et)
105. {
106. MessageBox.Show("Serial Port canot be connected\n" + et.ToString());
107. }
108. }
110. private void clearCharts()
111. {
112. foreach (var series in chart1\_timeDomain.Series)
113. {
114. series.Points.Clear();
115. }
117. foreach (var series in chart1\_frequencyDomain.Series)
118. {
119. series.Points.Clear();
120. }
121. }

124. private void cb\_selectComPort\_Click(object sender, EventArgs e)
125. {
126. updateSerialDevices();
127. }
128. private void Rb\_startPlot\_CheckedChanged(object sender, EventArgs e)
129. {
130. try
131. {
132. serialPort1.Write("g1@");
133. }
134. catch (Exception ex)
135. {
137. }
138. }
140. private void Rb\_stopPlot\_CheckedChanged(object sender, EventArgs e)
141. {
142. //clearCharts();
143. //serialPort1.Write("g0@");
144. }
146. private double rawReadingToVoltage(double rawData)
147. {
148. rawData = (rawData / ((double)4095));
149. return rawData;
150. }
152. private void Cb\_selectScale\_SelectedIndexChanged(object sender, EventArgs e)
153. {
154. double max = 1;
155. double min = 0;
156. if (cb\_selectScale.Text == "0-1")
157. {
158. min = 0;
159. max = 1;
160. }
161. else if (cb\_selectScale.Text == "0.5±(0.5/2)")
162. {
163. max = 0.5 + (0.5 / 2);
164. min = 0.5 - (0.5 / 2);
165. }
166. else if (cb\_selectScale.Text == "0.5±(0.5/8)")
167. {
168. max = 0.5 + (0.5 / 8);
169. min = 0.5 - (0.5 / 8);
170. }
171. else if (cb\_selectScale.Text == "0.5±(0.5/32)")
172. {
173. max = 0.5 + (0.5 / 32);
174. min = 0.5 - (0.5 / 32);
175. }
176. else if (cb\_selectScale.Text == "0.5±(0.5/100)")
177. {
178. max = 0.5 + (0.5 / 100);
179. min = 0.5 - (0.5 / 100);
180. }
182. chart1\_timeDomain.ChartAreas[0].AxisY.Minimum = min;
183. chart1\_timeDomain.ChartAreas[0].AxisY.Maximum = max;
184. }
186. private void Btn\_changeDacFrequency\_Click(object sender, EventArgs e)
187. {
188. if (serialPort1.IsOpen == true)
189. {
190. double dacFreq = 100000 / Convert.ToDouble(tb\_dacFrequency.Text);
191. int dFeq = (int)dacFreq;
192. string cmd = "d" + dFeq.ToString() + "@";
193. serialPort1.Write(cmd);
194. }
195. else
196. {
197. MessageBox.Show("Serial POrt NOt Connected");
198. }
199. }
201. private void Cb\_timeScale\_SelectedIndexChanged(object sender, EventArgs e)
202. {
203. double max = chart1\_timeDomain.ChartAreas[0].AxisX.Maximum;
204. if (cb\_timeScale.Text == "0.5")
205. {
206. max = max / 2;
207. }
208. else if (cb\_timeScale.Text == "0.25")
209. {
210. max = max / 4;
211. }
212. else if (cb\_timeScale.Text == "0.125")
213. {
214. max = max / 8;
215. }
216. else if (cb\_timeScale.Text == "1")
217. {
218. max = Convert.ToDouble(cb\_bufferLength.Text);
219. }
220. chart1\_timeDomain.ChartAreas[0].AxisX.Maximum = max;
221. }
222. }
223. }

1. Design and code for Spectral Analyzer
2. data = parseResponse(data, "@", "!");
3. string[] allSamples = data.Split(',');
5. double Fs = getCurrentSamplingFrequency();
6. double Ts = 1 / Fs;

9. //define containers
11. //voltage samples
12. List<double> \_channel1Amplitude = new List<double>();
13. List<double> \_channel1Time = new List<double>();
15. //DFT
16. List<double> \_channel1FrequencyPositive = new List<double>();
17. List<double> \_channel1FrequencyNegative = new List<double>();
19. List<double> \_channel1Mag = new List<double>();
20. List<double> \_channel1MagnitudePositive = new List<double>();
21. List<double> \_channel1MagitudeNegative = new List<double>();
22. List<double> \_channel1MagnitudeSearch = new List<double>();
24. List<double> \_channel1Phase = new List<double>();
26. List<double> \_freqList = new List<double>();

29. //store amplitude and time values
30. double timeStart = 0;
31. double timeIncrement = Ts;
32. for (int i = 0; i < (allSamples.Length - 1); i++)
33. {
34. double val = Convert.ToDouble(allSamples[i]);
35. val = rawReadingToVoltage(val);
36. \_channel1Amplitude.Add(val);
37. \_channel1Time.Add(timeStart); //store time values
38. timeStart += Ts;
39. }

42. //calculate positive frequency values
43. double harmonic = Fs / \_channel1Amplitude.Count;
45. //double \_freqIncrements = 0;
46. int nCount = \_channel1Amplitude.Count / 2;
47. for (int i = 0; i < nCount; i++)
48. {
49. \_channel1FrequencyPositive.Add(i \* harmonic);
50. }
52. //calacute negative frequency values
53. int temp1 = nCount-1;
54. for (int i = temp1; i > 0; i--)
55. {
56. \_channel1FrequencyNegative.Add( ( -1 \* \_channel1FrequencyPositive.ElementAt(i)) );
57. }

60. //calculate values for frequency domain chart
61. double \_maxVal = 0.0;
62. int \_maxValIndex = 0;
63. for (int n = 0; n < (allSamples.Length - 1); n++)
64. {
65. double dftReal = 0.0;
66. double dftImag = 0.0;
67. for (int k = 0; k < (allSamples.Length - 1); k++)
68. {
69. dftReal += \_channel1Amplitude.ElementAt(k) \* Math.Cos((2 \* Math.PI \* n \* k) / (double)\_channel1Amplitude.Count); //real
70. dftImag += \_channel1Amplitude.ElementAt(k) \* Math.Sin((2 \* Math.PI \* n \* k) / (double)\_channel1Amplitude.Count); //imaginay
71. }
73. double tempMag = (dftReal \* dftReal) + (dftImag \* dftImag);
74. \_channel1Mag.Add(tempMag);
75. \_channel1Phase.Add((Math.Atan(dftImag / dftReal) \* ((double)180 / Math.PI)));
77. if( (n > 5) && (n < (allSamples.Length/2)) )
78. {
79. if (\_maxVal < tempMag)
80. {
81. \_maxVal = tempMag;
82. \_maxValIndex = n;
83. }
84. }
85. }

88. //calculate absolute magnitude values
89. for (int i = 0; i < (\_channel1Mag.Count / 2); i++)
90. {
91. double temp = \_channel1Mag.ElementAt(i);
92. temp = Math.Abs(temp);
93. \_channel1MagnitudePositive.Add(temp);
94. }

97. //store values for magnitude response negative
98. int temp2 = \_channel1MagnitudePositive.Count - 1;
99. for (int i = temp2; i > 0; i--)
100. {
101. \_channel1MagitudeNegative.Add(\_channel1Mag.ElementAt(i));
102. }

105. //clear chart
106. clearCharts();

109. //display peak to peak voltage
110. lbl\_y2.Text = \_channel1Amplitude.Max().ToString("##0.000");
111. lbl\_y1.Text = \_channel1Amplitude.Min().ToString("##0.000");
112. lbl\_y2\_y1.Text = (\_channel1Amplitude.Max() - \_channel1Amplitude.Min()).ToString("##0.000");
114. //display frequency val
115. lbl\_freqIndex.Text = \_maxValIndex.ToString();
116. double bufLen = Convert.ToDouble(cb\_bufferLength.Text);
117. chart1\_timeDomain.ChartAreas[0].AxisX.Maximum.ToString();



122. //plot spectrum
123. if (rb\_magnitudeResponse.Checked == true)
124. {
126. for (int i = 0; i < \_channel1MagitudeNegative.Count; i++)
127. {
128. if (rb\_magnitudeDecibel.Checked == true)
129. {
130. chart1\_frequencyDomain.Series["ch1\_mag"].Points.AddXY(\_channel1FrequencyNegative.ElementAt(i), (20 \* Math.Log10(\_channel1MagitudeNegative.ElementAt(i)))); //plot magnitude response
131. }
132. else
133. {
134. chart1\_frequencyDomain.Series["ch1\_mag"].Points.AddXY(\_channel1FrequencyNegative.ElementAt(i), \_channel1MagitudeNegative.ElementAt(i)); //plot magnitude response

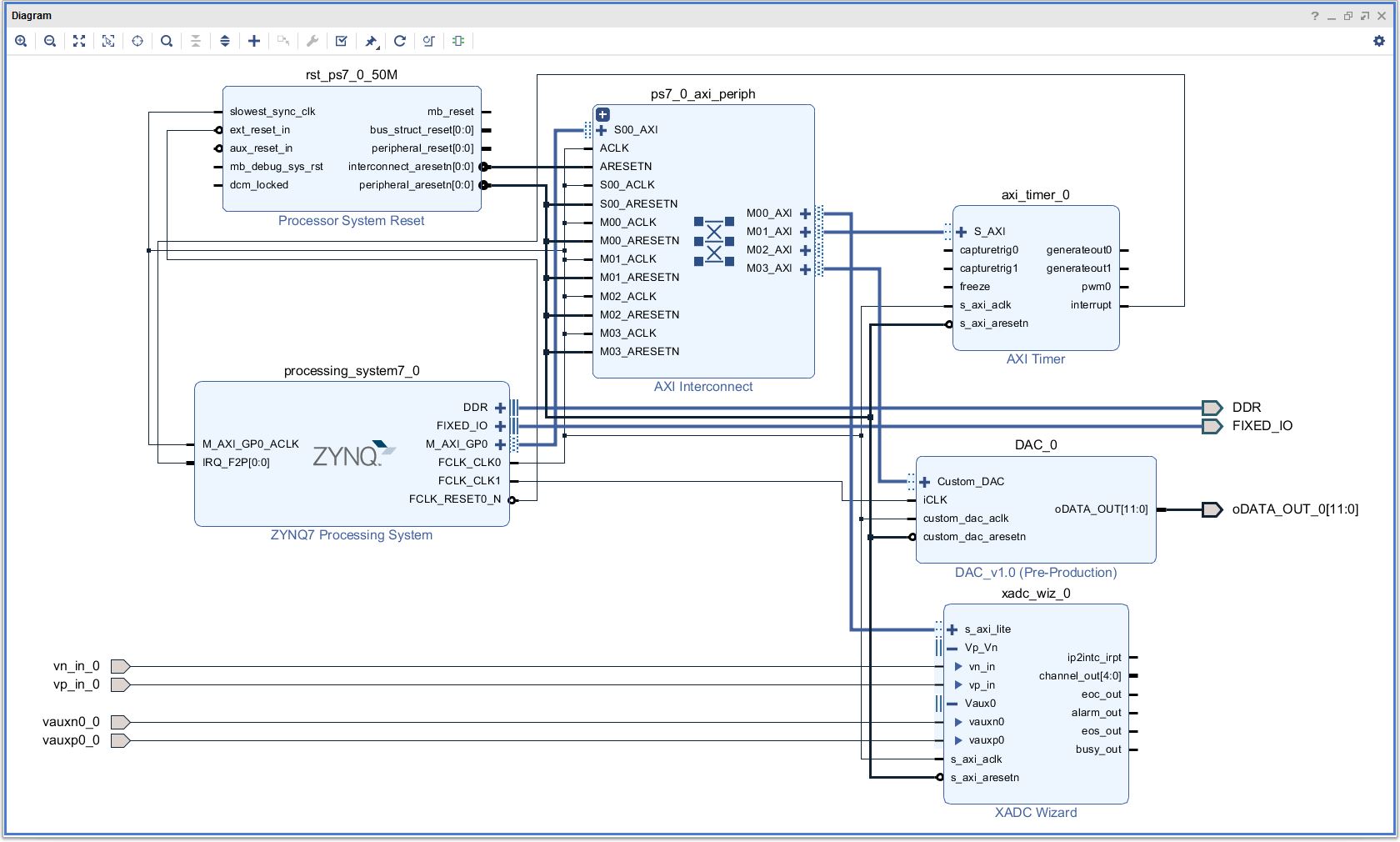
137. }
139. }

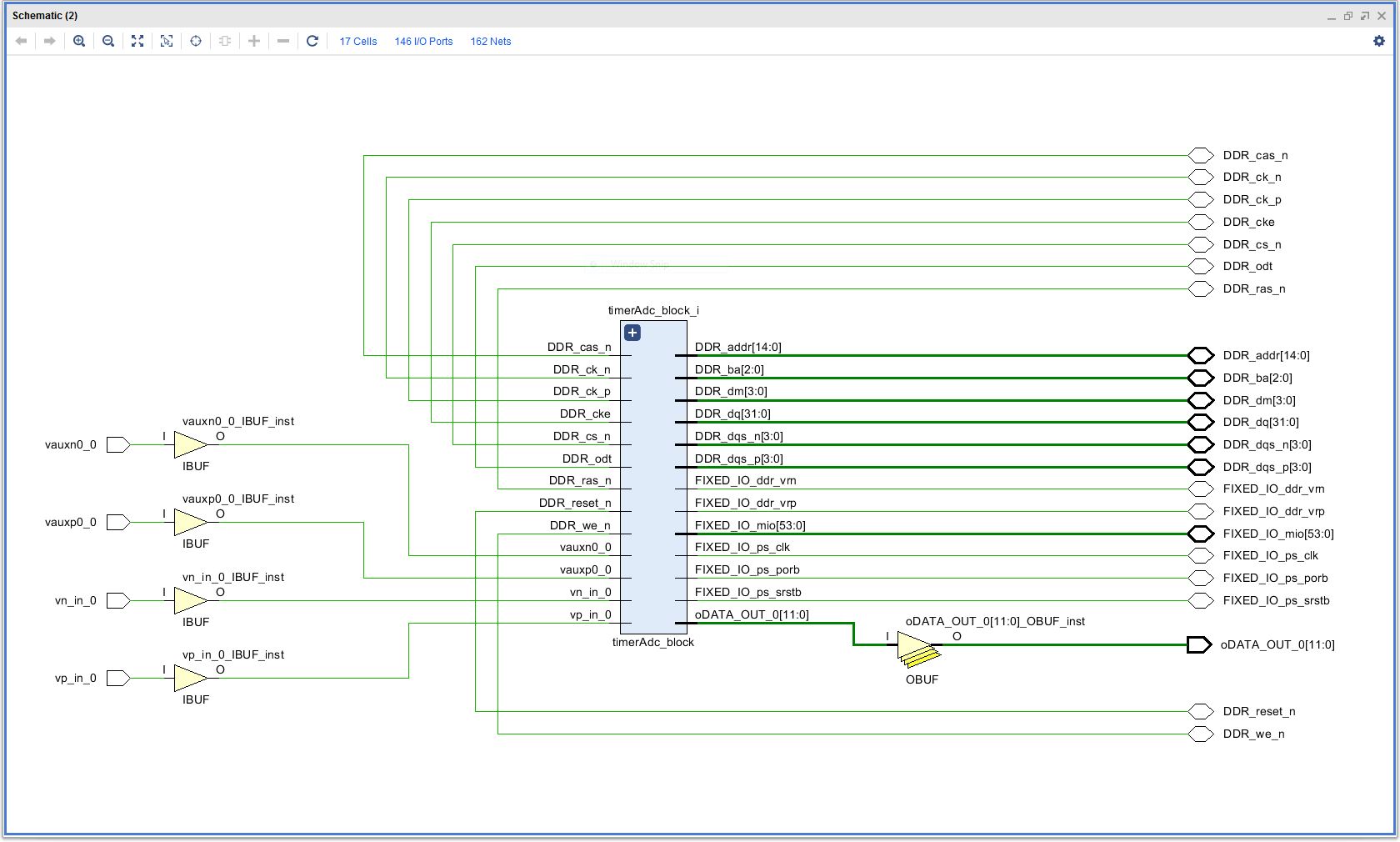
142. //plot right half
143. for (int i = 0; i < \_channel1MagnitudePositive.Count; i++)
144. {
146. if (rb\_magnitudeDecibel.Checked == true)
147. {
148. chart1\_frequencyDomain.Series["ch1\_mag"].Points.AddXY(\_channel1FrequencyPositive.ElementAt(i), (20 \* Math.Log10(\_channel1MagnitudePositive.ElementAt(i)))); //plot magnitude response
149. }
150. else
151. {
152. chart1\_frequencyDomain.Series["ch1\_mag"].Points.AddXY(\_channel1FrequencyPositive.ElementAt(i), \_channel1MagnitudePositive.ElementAt(i)); //plot magnitude response
153. }
155. }
157. }
158. else
159. {
160. //pllot phase
161. for (int i = 0; i < \_channel1Phase.Count; i++)
162. {
163. chart1\_frequencyDomain.Series["ch1\_phase"].Points.AddY(\_channel1Phase.ElementAt(i)); //plot magnitude response
164. }
165. }
167. rxData = String.Empty;
169. if (rb\_startPlot.Checked == true)
170. {
171. serialPort1.Write("g@");
172. }
173. else
174. {
175. clearCharts();
176. }
177. System Performance (What your system can do)

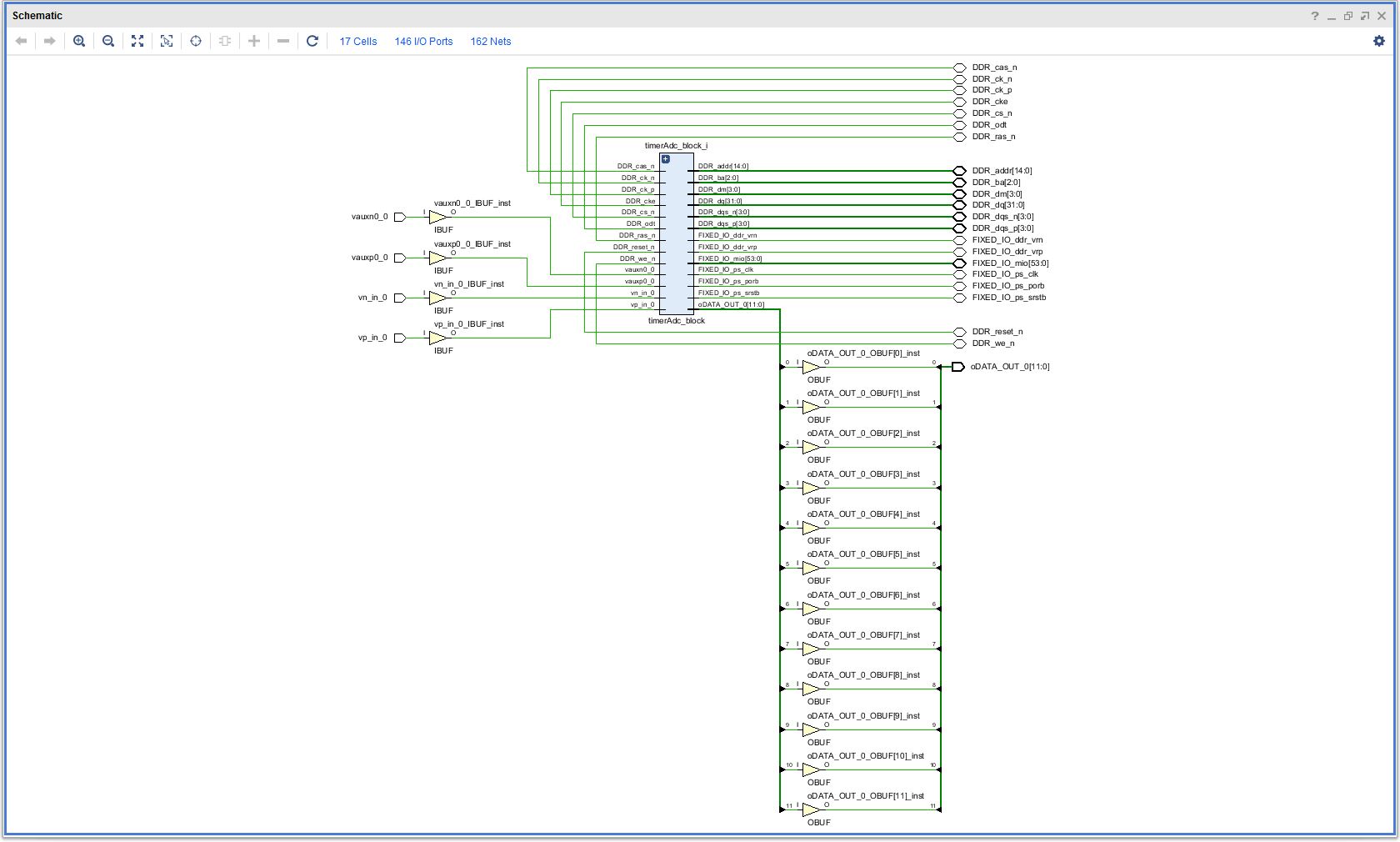
The System Can do the Following as specified in Design Specifications

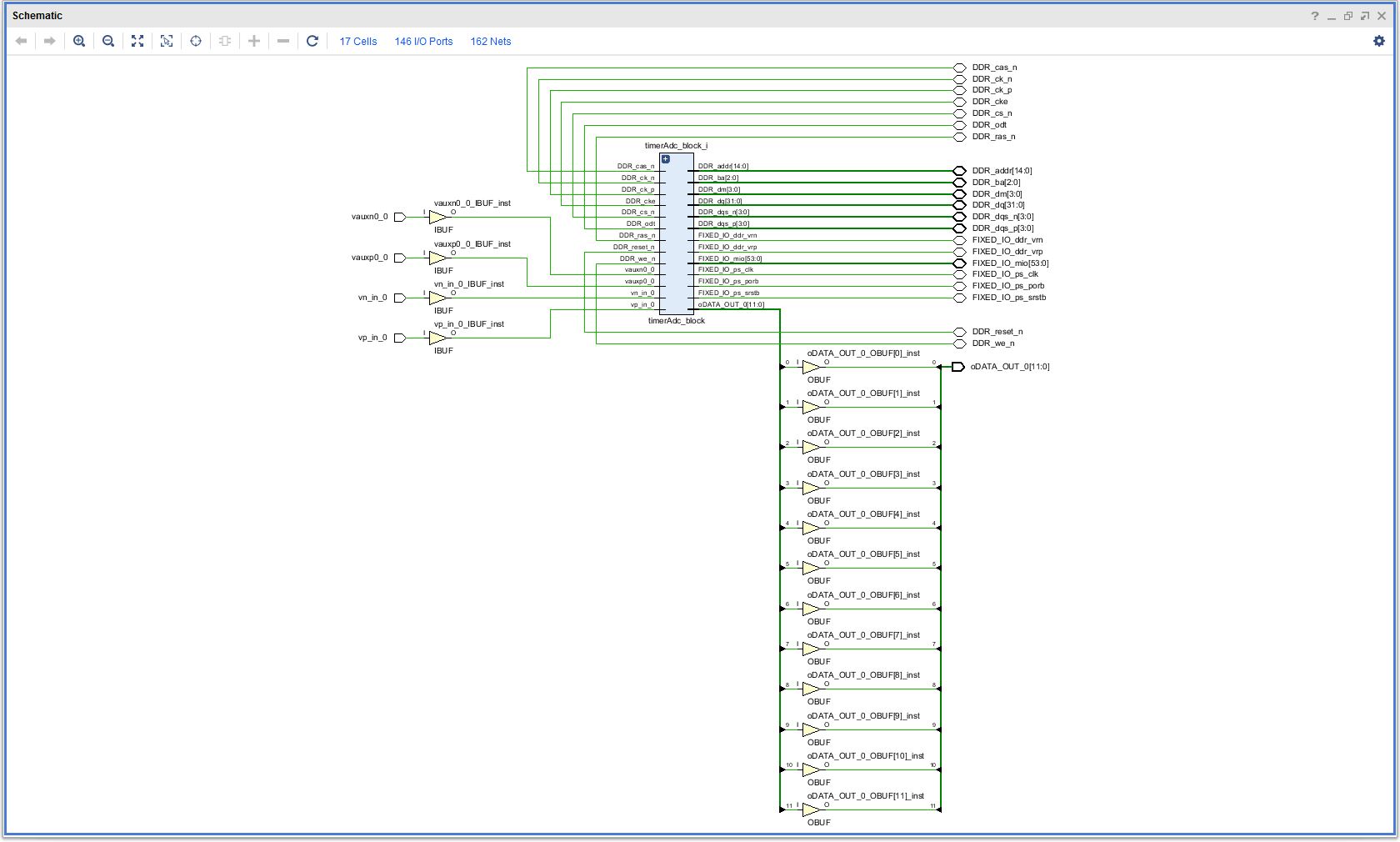
* 1. Start Plot and Stop Plot
  2. Change the Sampling Frequency
  3. Change the Buffer Length (256, 512, 1024)
  4. Change the DAC Frequency
  5. Change the Trigger Value
  6. Change the Trigger Edge
  7. Plot Time Scope/ Oscilloscope (Amplitude Vs Time)
  8. Scale the Oscilloscope Y -Axis for Small Signals
  9. Scale the Oscilloscope X-Axis To Zoom
  10. Display Peak To Peak Voltage Value
  11. Perform DFT on Samples
  12. Plot the DFT Magnitude in Linear and Decibel Scale
  13. Measure Signal Frequency
  14. Zoom on Spectral Plot

# Results and Analysis:

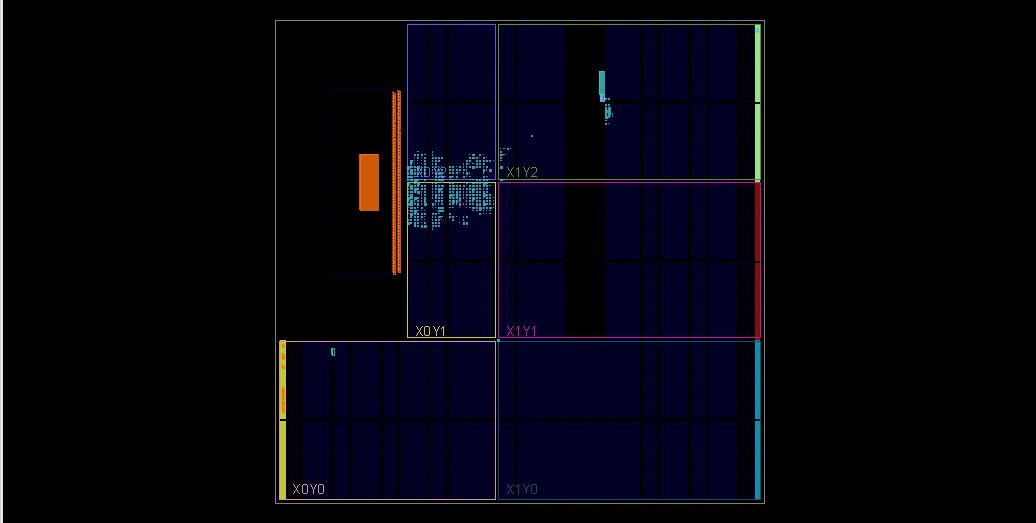
Block Diagram

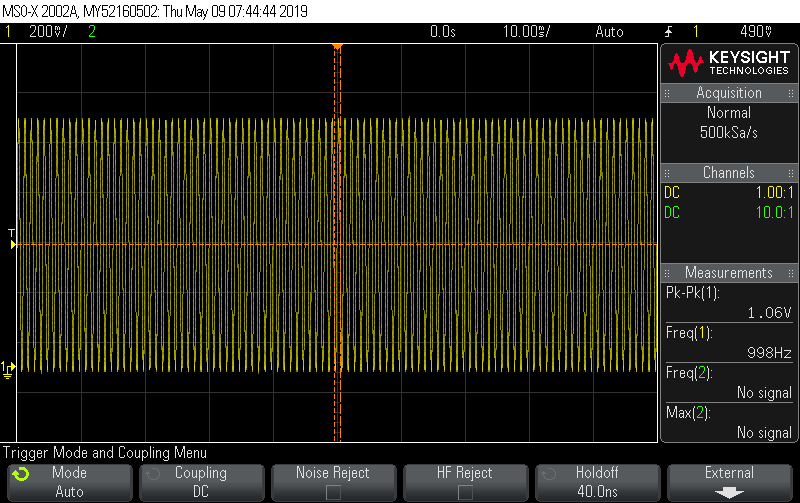
Elaborated Design

Synthesized Design

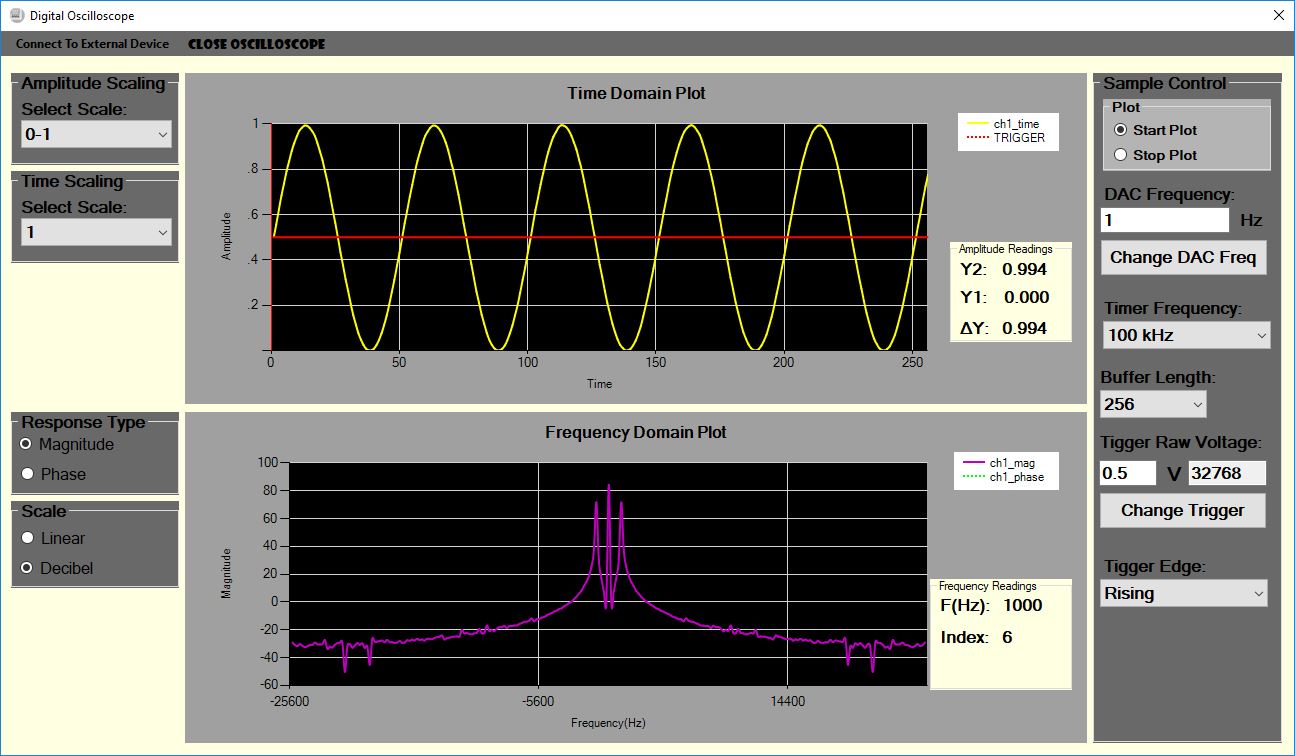


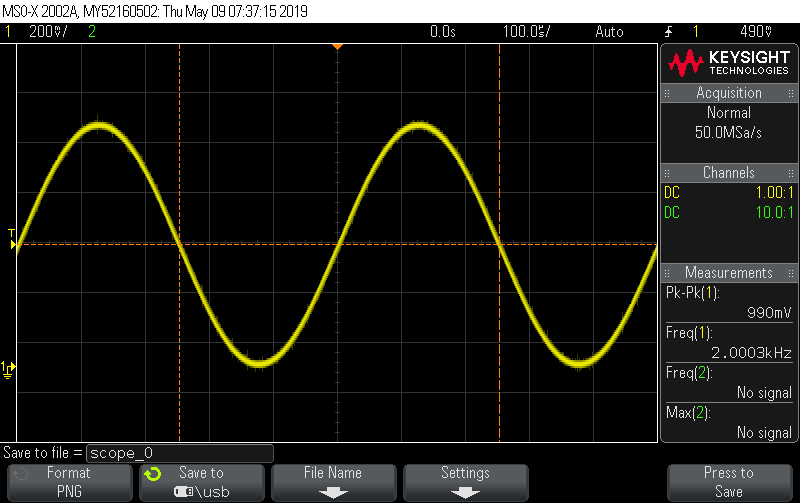
Implemented Design

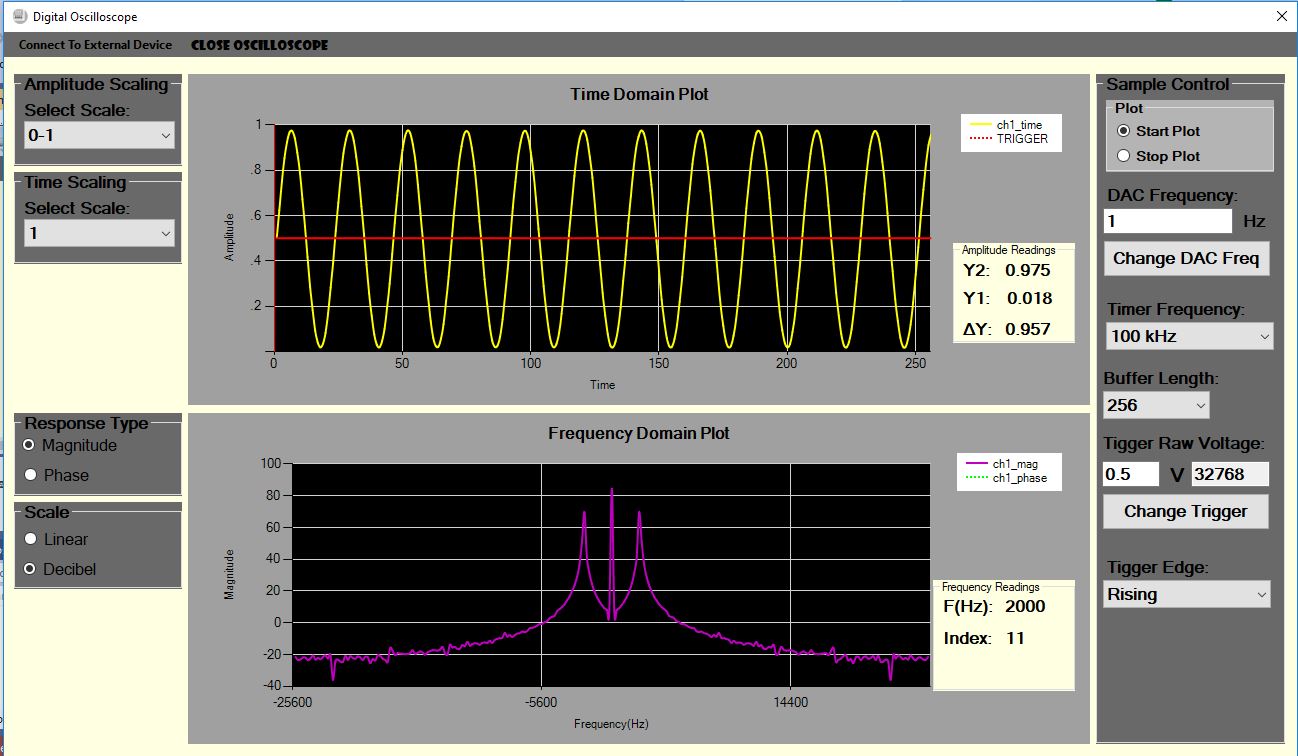
FPGA Status

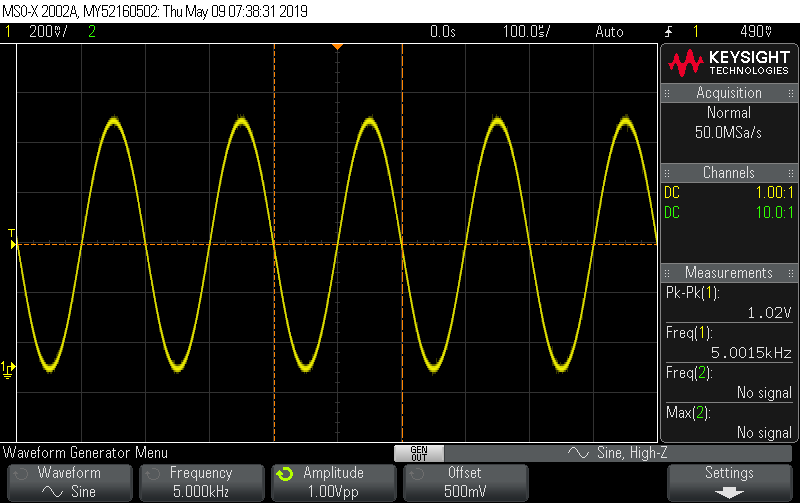


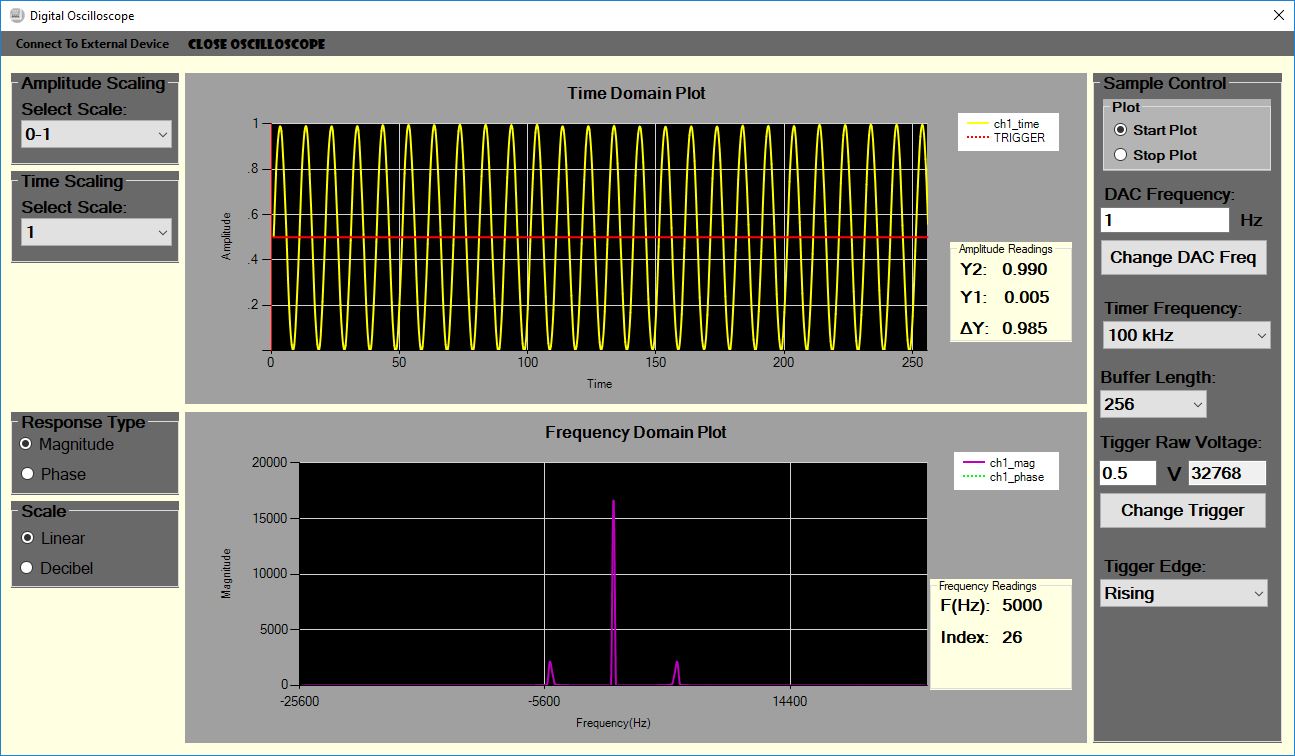
Oscilloscope at 1khz, 1 Vpp

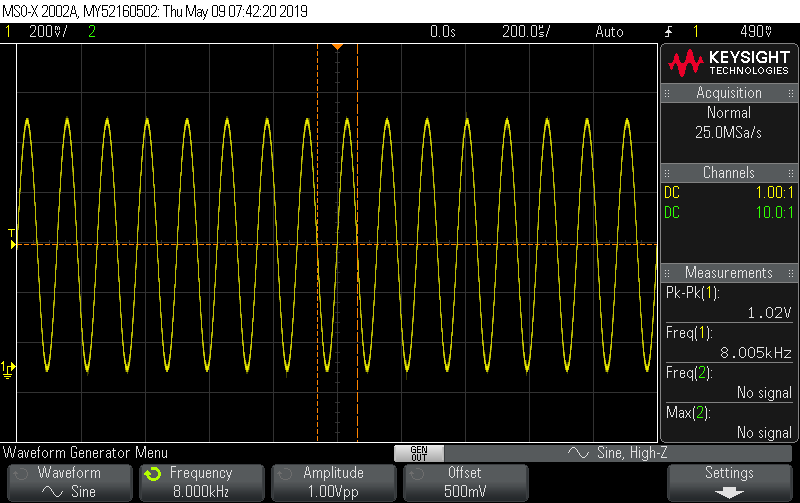
Digital Oscilloscope and Spectral Analyzer at 1kHz

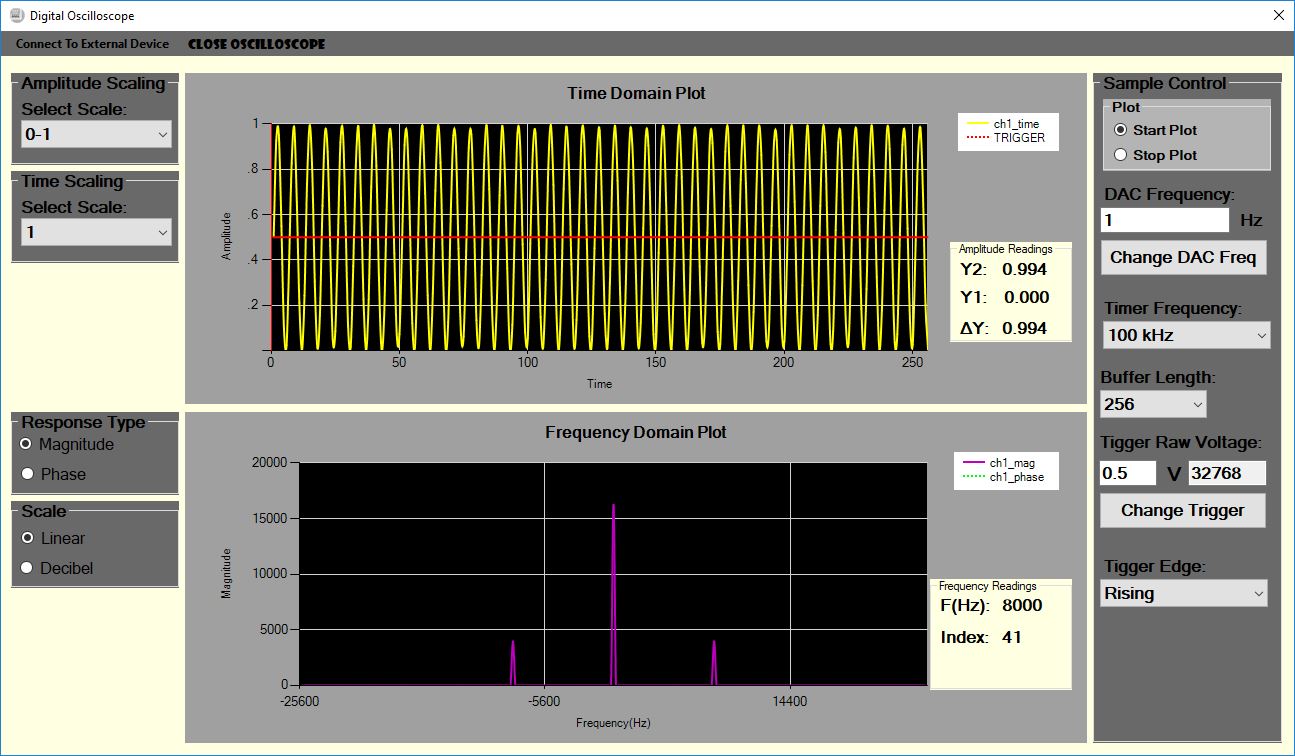
Oscilloscope at 2 kHz, 1Vpp

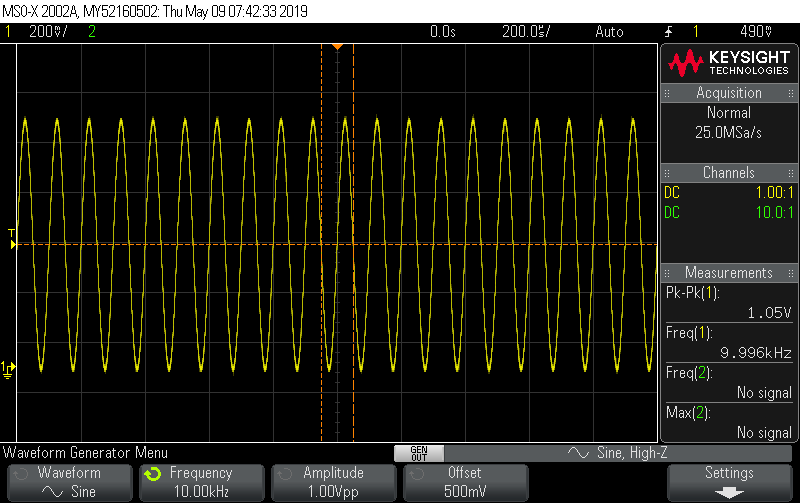
Digital Oscilloscope and Spectral Analyzer at 2kHz

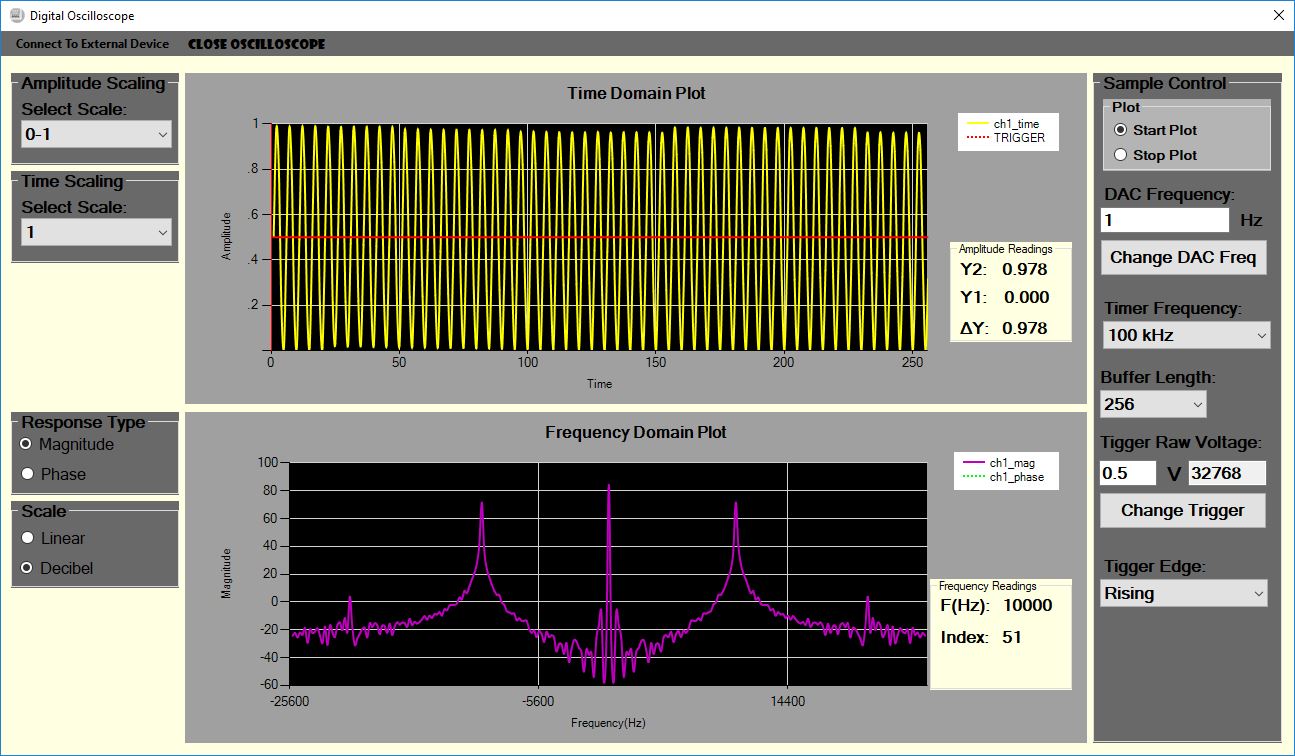
Oscilloscope at 5 kHz, 1Vpp

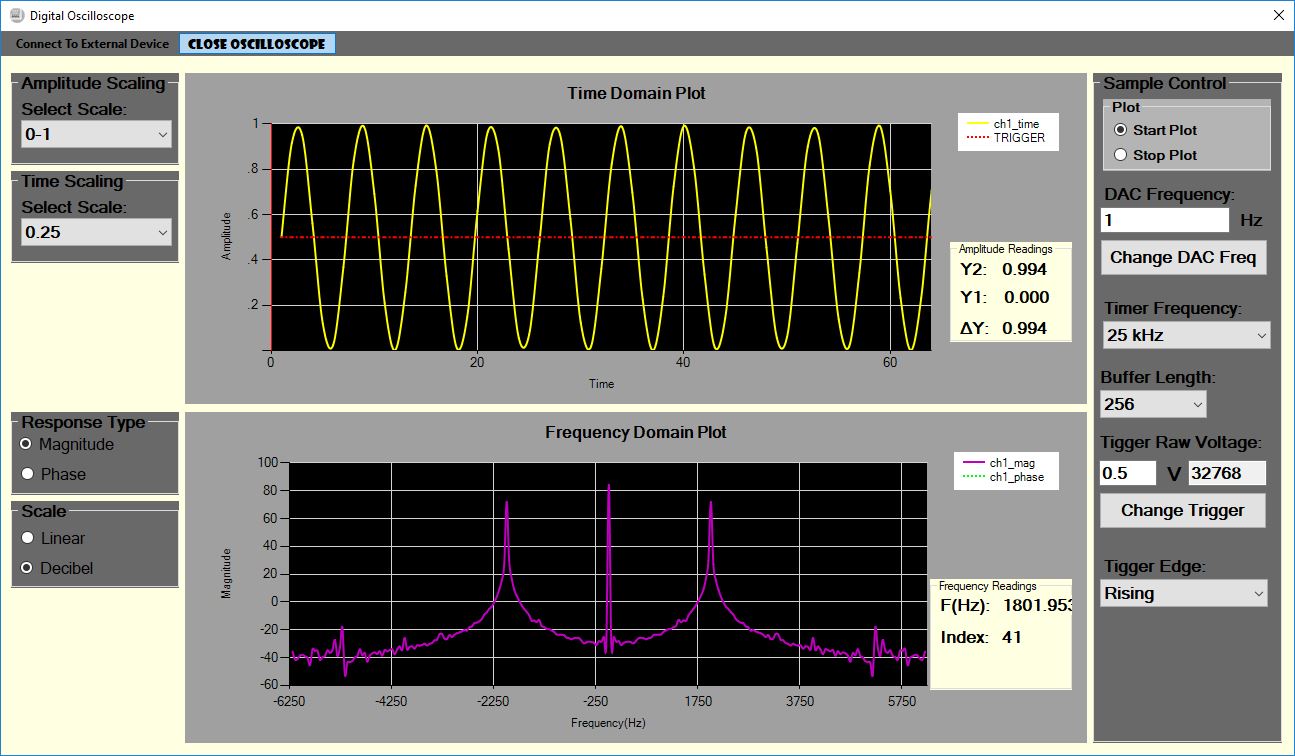
Digital Oscilloscope and Spectral Analyzer at 2kHz

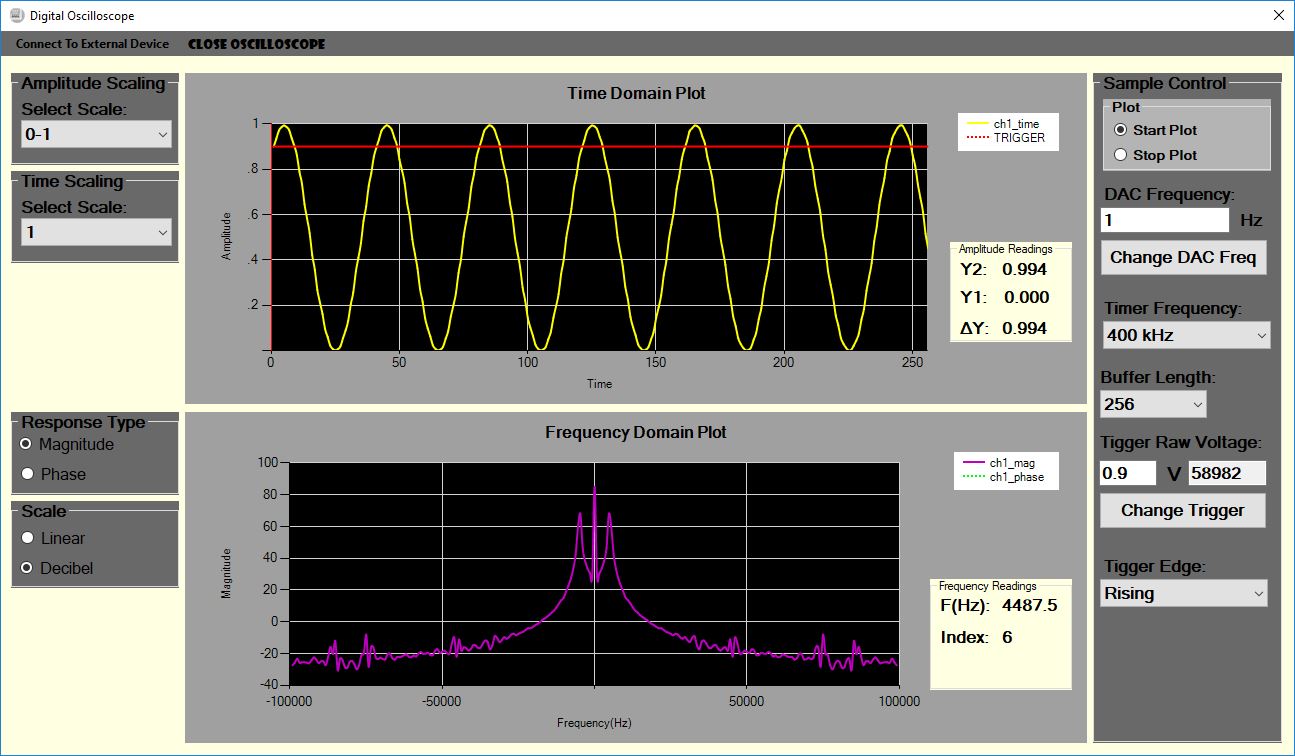
Oscilloscope at 5 kHz, 1Vpp

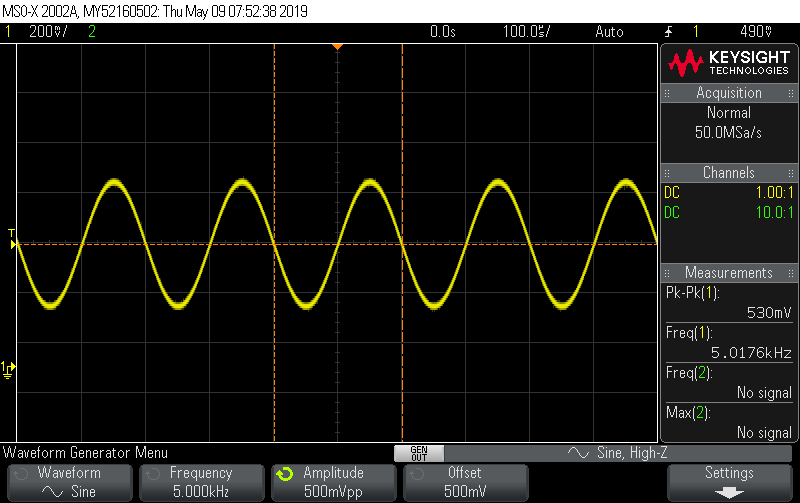
Digital Oscilloscope and Spectral Analyzer at 8kHz

Oscilloscope at 5 kHz, 1Vpp

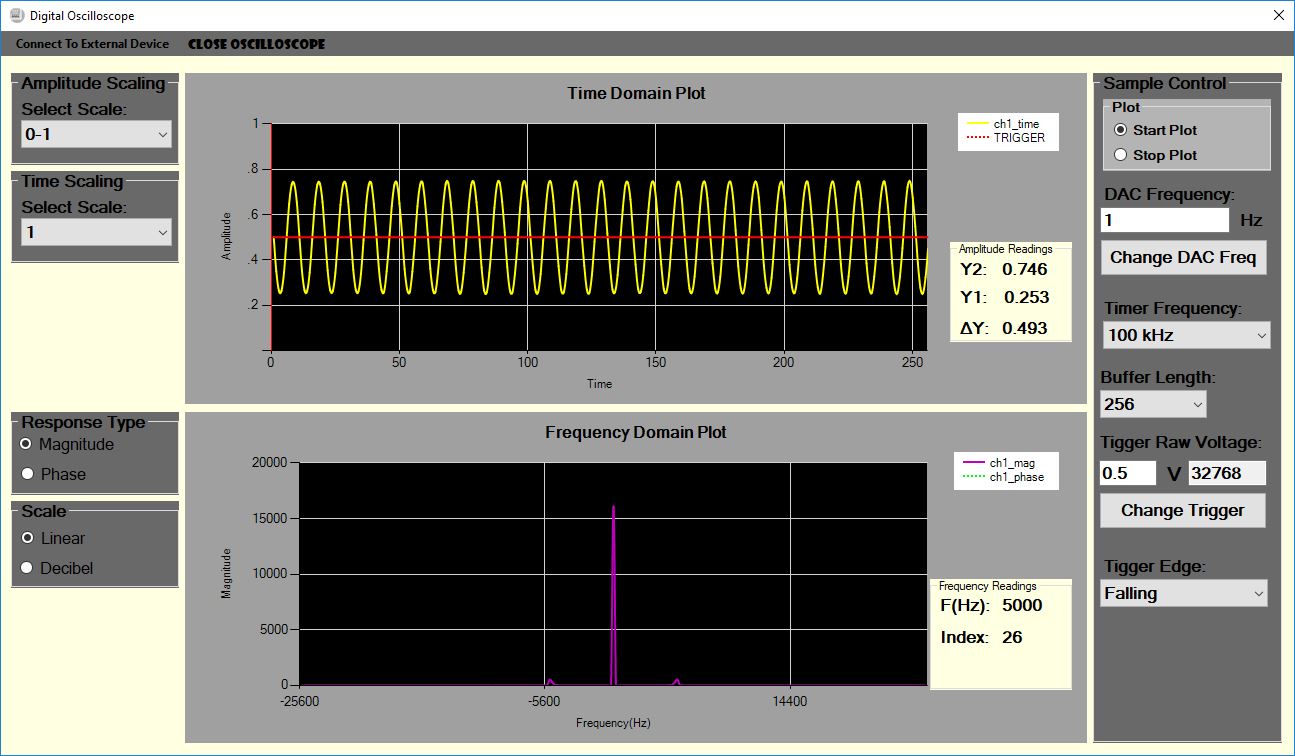
Digital Oscilloscope and Spectral Analyzer at 10kHz

Digital Oscilloscope and Spectral Analyzer at 10kHz, Fs = 25kHz

Digital Oscilloscope and Spectral Analyzer at 448kHz, Fs = 400kHz

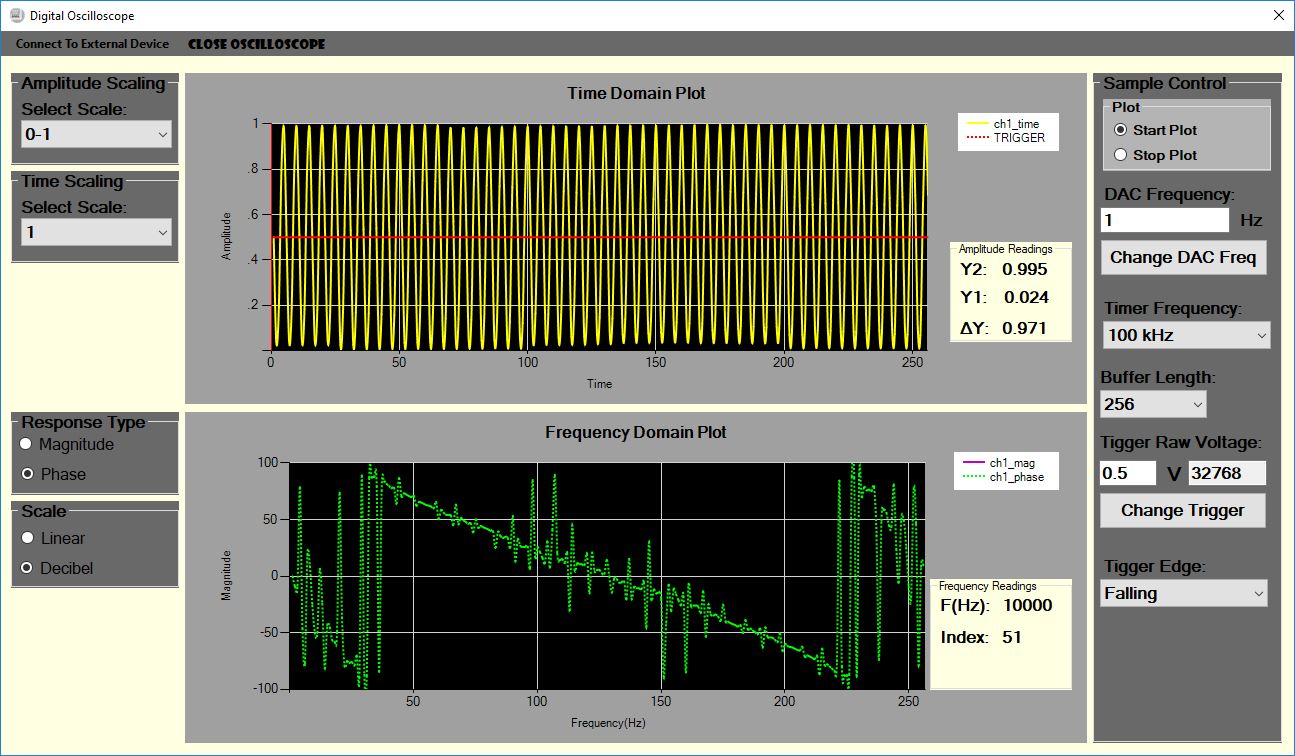


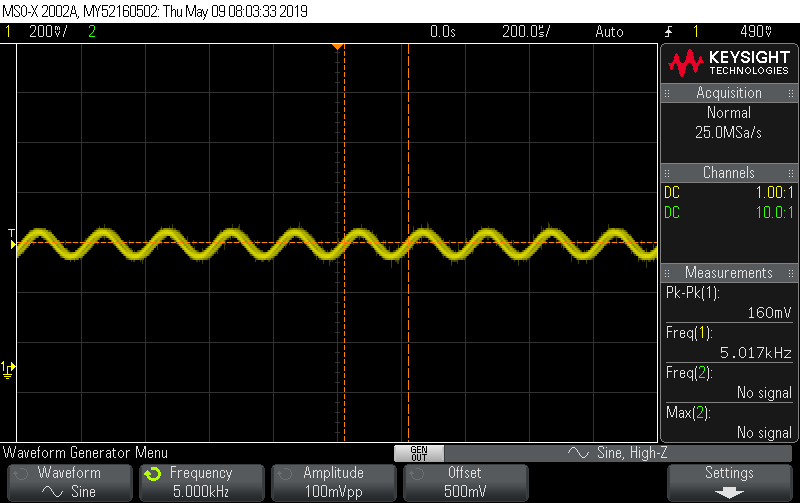
Oscilloscope at 5 kHz, 500mVpp

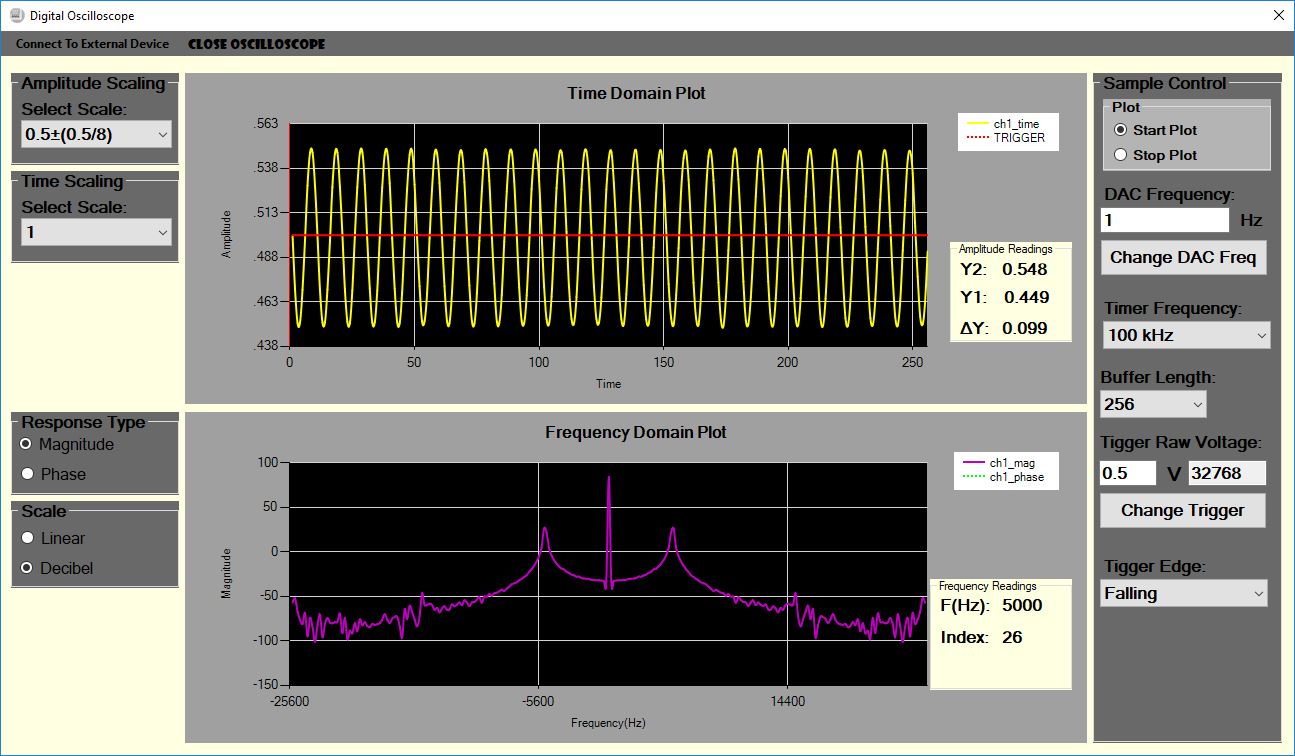


Digital Oscilloscope and Spectral Analyzer at 5kHz, Amplitude = 500mVpp

Digital Oscilloscope and Spectral Analyzer at 5kHz, Amplitude = 500mVpp, Scaled

Digital Oscilloscope and Spectral Analyzer at 10kHz, Phase Response

Oscilloscope at 5 kHz, 100mVpp

Digital Oscilloscope and Spectral Analyzer at 5kHz, Amplitude = 100mVpp

The traces above show that the, majority of the system Requirements were met, the digital oscilloscope and spectral analyzer can detect and plot frequency ranging from 0-1vpp and 100-10kHz range and accurately display the peak to peak voltage and signal frequency. The pictures above also show that the Digital Spectral Analyzer can also plot phase response and display magnitude response in linear and dB scale. The Gui also allows user to Zoom in to the digital oscilloscope to view smaller signal as per the design specifications. This oscilloscope also allows user to change the number of samples from 256, 512, 1024. The digital Oscilloscope also allows user to change trigger point between 0 and 1Vpp. The Digital Oscilloscope also allows user to change trigger edge such as Falling, Rising and Either. The digital Oscilloscope also allows user to change sampling frequency. Overall the Design Digital Oscilloscope has met majority of the design specification aside from axes being labeled properly. This project in general helped to understand ZedBoard components like XADC and the Zynq7000 SoC. The project helped to learn basic serial communication. Most importantly the project helped learn a good application of DFT and how to perform DFT manually without using libraries.

# Contemporary:

Oscilloscope as we are physical Equipment’s and Are expensive. There are budget oscilloscopes available in market today however they are not quite up to pace when it comes to performance. The class slowly taught the foundations in making a digital oscilloscope with SoC design, which could possibly eliminate the use of Oscilloscope in future making signal processing/analysis easier. The class taught how to use SoC for Basic Signal Processing, Design GUI that mimics an Oscilloscope Interface, implement function generator (DAC) using the SOC, plot Time scope and Spectral Plot.

# Conclusion:

A Digital Oscilloscope and Spectral Analyzer with SoC design was designed and Implemented. The SoC design helped increase knowledge on VHDL, Verilog, Block Design and gave an increased knowledge on how to use ZedBoard for various tasks. The GUI designed in C# mimics the interface of a real Oscilloscope with the SoC doing most of the work on background. The C# Gui was able to Communicate with ZedBoard through Serial Communication. A Practical Application DFT was displayed in this project. This project overall helped understand SoC design and Especially helped understand Discreet Fourier Transform.

# Acknowledgement:

* 1. Dr Zheng For Giving Opportunity To Learn ZedBoard.
  2. Dr Zheng for proving DFT Equations.
  3. Steven Kamm for Help designing Labs which helped in Project.

# Reference:

# “ZedBoard.” Zedboard, zedboard.org/product/zedboard[1].

# “ZedBoard.” ZedBoard [Reference.Digilentinc], reference.digilentinc.com/reference/programmable-logic/zedboard/start[2].

1. “Basics of UART Communication.” Circuit Basics, 11 Apr. 2017, www.circuitbasics.com/basics-uart-communication/[3].