ECE532 Project Report

9th April 2018

Electrical Impedance Tomography

Group 13

Duan Ran

Jaimin Joshi

Jianxiong Xu

Contents

Overview

1.1 Project Motivation

In recent times, Wireless Current Sensing of Biological chemicals and Electrical Impedance Tomography (EIT) have become important phenomena in medical imaging of body tissues. EIT is the These techniques are effective for intensive care diseases such as lung conditions. While other medical imaging systems such as MRIs offer higher resolution, they cannot be used in integrated circuit-based microsystems to enable the creation of implantable medical devices that can alarm early signs of the diseases. Moreover, the same IC helps to monitor the vital parameters on the body by placing the bio-sensing material on the top of the electrodes. The fully-implantable system can monitor the drug and glucose level in the body continuously for long-term usage which can be utilized in personalized therapy.

EIT is a non-invasive type of medical imaging in which the electrical conductivity, permittivity, and impedance of a part of the body is inferred from surface electrode measurements and used to form a tomographic image of that part, as shown in the Figure 1.1.1. Electrical conductivity varies considerably among various biological tissues (absolute EIT), or the movement of fluids and gases within tissues (difference EIT). The majority of EIT systems apply small alternating currents at a single frequency. However, some EIT systems use multiple frequencies to better differentiate between normal and suspected abnormal tissue within the same organ (multifrequency-EIT or electrical impedance spectroscopy) (Wikipedia, Electrical Impedance Tomography, 2018).

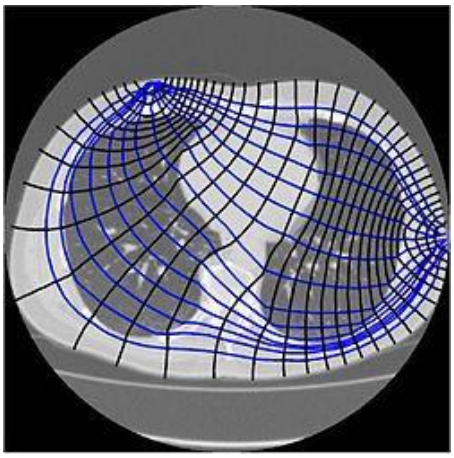

Figure 1. A cross section of a human thorax from an X-ray CT showing current stream lines and equi-potentials from drive electrodes (Picture courtesy of EIDORS)

1.2 Project Background

A fully-custom ASIC is currently under test in the Intelligent Sensory Microsystems Laboratory. The ASIC is intended to be utilised in EIT applications, namely, as a brain implant and monitoring peripheral nerve activity.

This ASIC has two main blocks - one being the op-amp less second order delta-sigma ADC which is used for detecting the analogy signals from outside by different channels. The other is a stimuli block, which is acting as a DAC, used for generating impulse signals for the sample under test.

The ASIC is able to interface with a Nexys Video FPGA board through a custom PCB via the FMC mezzanine connector. The Nexys Video FPGA generates 13 different clock and control signals in order to drive the ADC while simultaneously receiving the bit stream for digital signal processing, both of which are via the FMC mezzanine connector.

For the scope of this project, frequency and phase response characteristics were implemented on the Nexys Video FPGA. The target is to obtain the frequency and phase response from any two-port network in order to characterise the impendence from the given network. Taking a first order low-pass filter (-3dB bandwidth = 100Mhz) for example, if the frequency of the input signal is around DC, the amplitude of the output signal should be equivalent to that of the input signal. If the frequency is greater than 100Mhz, ideally, the amplitude of the output signal should be less than that of the input signal. In order to obtain the frequency response, the frequency of input signal can be swept and the amplitude of the output signal can be observed.

$$A_v(f) = \frac{V_{in}(f)}{V_{out}(f)}$$

The frequency response can be obtained by observing the amplitude as shown above, while measuring the delay between the input and the output signals provides the phase response.

The features of the proposed system that makes it an IOT project is the fact that the FPGA board is intended to transfer data in real time over to the computer in order to carry out FFT analysis. Furthermore, the frequency response of the two-port network was displayed on an external display.

The ultimate capabilities of this project include measuring the impedance of a sample under test eg. rodent's brain, peripheral body nerve in order to characterise the condition of the tissue.

The need for this system arises from the fact that although equipment such as oscilloscope and signal generators are readily available in a laboratory environment, the target ASIC is a low-power brain implant that is not easily accessible. Therefore, the implemented logic and algorithms have to be compared and calibrated with the results observed on an oscilloscope to ensure that in future, the brain-implant ASIC, the logic and algorithm implemented is functional.

A fundamental part of the project involved driving the 2nd order Delta-Sigma ADC on the ASIC. "Delta-sigma (ΔΣ) modulation is a method for encoding analogue signals into digital signals." "The first step in a delta-sigma modulation is delta modulation. In delta modulation the change in the signal (its delta) is encoded, rather than the absolute value. The result is a stream of pulses, as opposed to a stream of numbers as is the case with pulse code modulation (PCM). In delta-sigma modulation, the accuracy of the modulation is improved by passing the digital output through a 1-bit DAC and adding (sigma) the resulting analogue signal to the input signal (the signal before delta modulation), thereby reducing the error introduced by the delta-modulation." (Wikipedia, Delta-Sigma Modulation, 2018)

The key features of this ΔΣ-ADC are course quantization, filtering, feedback, and oversampling. This ΔΣ-ADC is used for measuring the brain signals that has a bandwidth form DC to 5 KHz. For a regular ADC, to make the anti-alias filter feasible, we need OSR from 2 to 3. For this ΔΣ-ADC, the quantization is quite course (as low as 1-bit), but the effective resolution can still be as high as 16 bits. Thus, to get adequate quantization noise suppression and to remove the signals between f_b and f_s digitally, an OSR of more than 64 is required. In other words, a higher clock rate is needed.

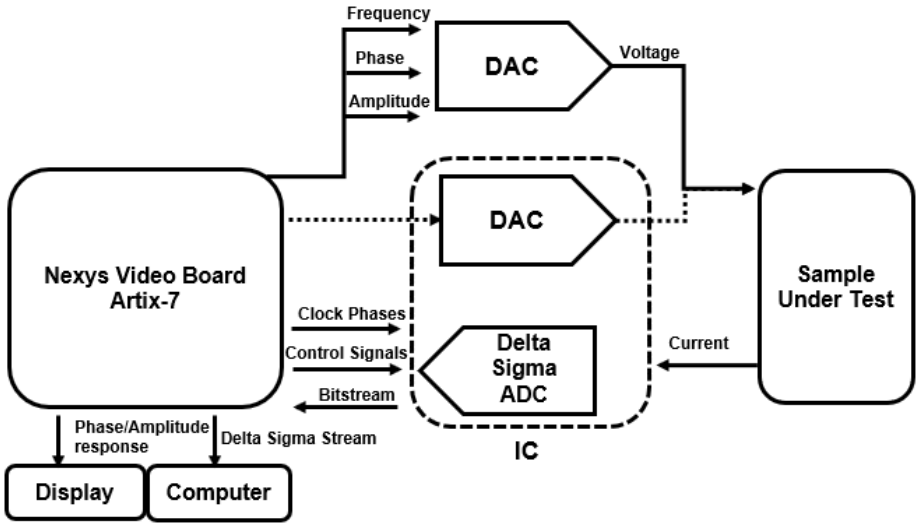

Figure 2. Block diagram of initially proposed system

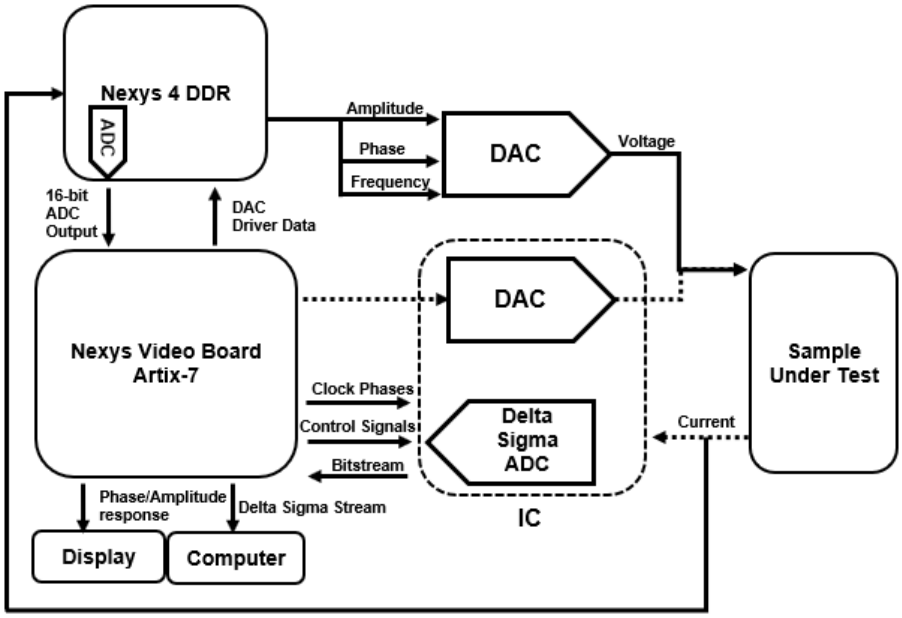

Figure 3. Block diagram of the final setup

The block diagram in Figure 2 illustrates the originally proposed system while Figure 3 illustrates an overview of the final setup. The data that is exchanged between the main components of the system is indicated in this diagram. The initial proposed design did not include the Nexys 4 DDR board. This alteration to the proposed system is further discussed in the outcome of this project.

For test purposes, a simple RC network was used to represent the sample under test as the body tissue is expect to have finite impedance characteristics.

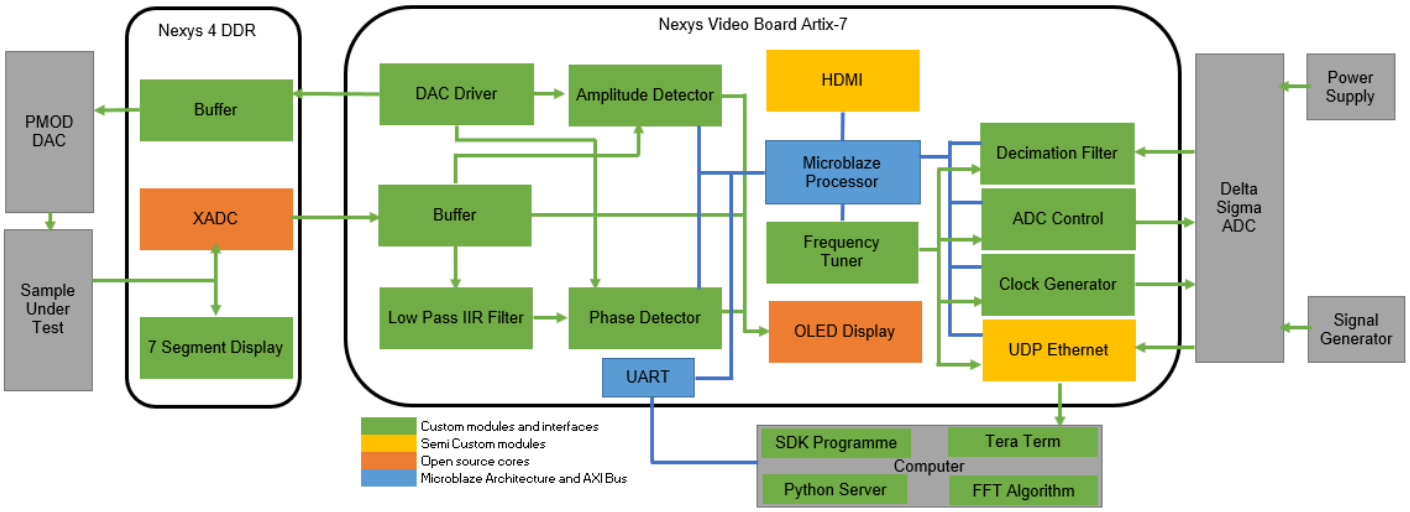

Figure 4. Block diagram of entire system

The block diagram above illustrates in detail the various components of the system and the origin of the designs. As seen above, the operation of the system can be split into three distinct categories and can be summarised as follows:

Driving ADC

1. SDK programme sets tuner frequency
2. Frequency tuner obtains data from MicroBlaze processor and adjusts output frequency
3. Clock generator tunes output clock pulses going into ADC
4. ADC generates Delta Sigma stream with a certain over sampling ratio
5. Delta Sigma bit-stream is transmitted to computer through the UDP module, over Ethernet
6. Python Server stores the data in a file
7. MATLAB reads the file and plots FFT response

Signal Processing

1. Delta Sigma bit-stream is passed into decimation filter (this step would continue into the output being passed to the IIR filter if the Delta Sigma ADC was used for the final setup).
2. DAC driver is programmed to sweep a range of frequencies
3. DAC drive output is buffered on the Nexys 4 board and PMOD DAC is driven accordingly
4. Output of DAC is passed into one port of the RC network
5. Second port of the RC network is connected to input of the ADC on board the Nexys 4 DDR
6. Output of the ADC is displayed on the LEDS and fed into the IIR Filter on the Nexys Video through a buffer
7. IIR Filter output is fed into phase and amplitude detector modules
8. Phase and Amplitude detectors carry out computations based on reference inputs from the DAC driver.
9. Outputs of Phase and Amplitude detectors are displayed on OLED

HDMI Display

1. Screen first displays a test pattern.
2. Input video stream is decoded into AXI stream format
3. RGB data is stored in DDR
4. VDMA reads image from memory to RGB converter
5. RGB to DVI video encoder used to show frame on monitor
6. Tera term used for inputting commands
7. MATLAB required to produces background pictures of plots

1.3 Project Goals

The ultimate goal and application of this project is for the custom IP and logic designed during the project to be incorporated into a future iteration of the chip, such that that system is self-contained SOC (system-on-chip).

The following **Primary Goals** were set and achieved for the scope of this project:

- Driving Delta Sigma ADC

  -Generate variable clock signals using clock generator

  -Control ADC through Microblaze using Xilinx SDK

  -Retrieve Delta Sigma stream on computer through UDP module and carry out FFT analysis on MATLAB

- Signal Processing

  -Design Decimation filter & IIR filter

  -Drive PMOD DAC

  -Compute frequency response through Phase and Amplitude detectors

- HDMI External Display

  -Display frequency response on external display

Additionally, the following stretch goals were also targeted at the beginning of the project:

- Design custom DFT
- Display spectrum and real time frequency/phase response on display
- Control multiple channels of fully-custom Delta Sigma ADC

Driving the ADC was a fundamental part of the project - the operation of the ADC was essential in order to determine the frequency response of the signals. However, due to the challenges encountered while trying to produce high frequency clocks to drive the ADC, a significant amount of time was spent on this part of the project, which consequently reduced the time available to pursue stretch goals.

1.4 IP Description

IP	Function	Type	Source
Hardware IP			
MicroBlaze Processor	Soft-core microprocessor to interface between IP as well as SDK on computer	Xilinx IP	IP Catalog
Frequency Tuner	Tune the frequency of clock signals driving the Delta Sigma ADC	Custom	Team
Clock Generator	Generator clock signals with various phases and periodic times to drive Delta Sigma ADC	Custom	Team
Decimation Filter	Down-sample and average incoming bit-stream from Delta Sigma ADC	Custom	Team
Low Pass IIR Filter	Filter out high frequency noise from the decimation filter output	Custom	Team
UDP Ethernet	Transfer Delta Sigma bit-stream to computer in real time	Semi-custom	Hamsterworks
Phase Detector	Detect the phase difference between the reference signal driving the DAC and the output of the low-pass IIR Filter	Custom	Team
Amplitude Detector	Detect the amplitude difference between the reference signal driving the DAC and the output of the low-pass IIR Filter	Custom	Team
DAC Driver	Drives the PMOD DAC	Custom	Team
7-Segment Display	Displays numbers and hex values	Custom	Team
XADC	Detects and converts analogue signals to digital domain	Semi-Custom	Digilent
HDMI	Displays the frequency response on an external display	Semi-custom	Digilent

OLED Display	Displays the phase, amplitude, and frequency information data in real time.	Custom	Team
DDR	Storing the background pictures	Custom	Team
Software			
SDK Controller	Asserts various control signals, parameters for the clock generator, and reads data from memory locations to display results through HDMI	Custom	Team
Python Server	Reads incoming Delta-Sigma bitstream in real time, and stores the data in a file	Custom	Team
FFT Analysis	MATLAB script that reads the file containing Delta-Sigma bitstream and carries out FFT	Custom	Team

Outcome

2.1 Results

The primary goals of this project were successfully achieved such that the system not only matches the initial specification, but also exceeds the expected performance in certain areas of the design. The system was fully functional with minor alterations in the initially proposed project plan.

As mentioned previously, the operation and results of the entire system can be places into three distinct categories:

1. Driving the fully custom Delta Sigma ADC on the ASIC and analysing real time-data
2. Characterising the signals passing through the two-port network
3. Displaying the results on an external display

2.1.1 Driving Delta Sigma ADC

In order to drive the Delta-Sigma ADC, variable frequency and perodic-time clock signals were generated from the Nexys Video board. The frequency parameters, and hence the oversampling ratio of the ADC, are controlled using Xilinx SDK through the MicroBlaze softcore processor. The following figure illustrates the various clock signals generated from the Nexys Video board, displayed on an oscilloscope.

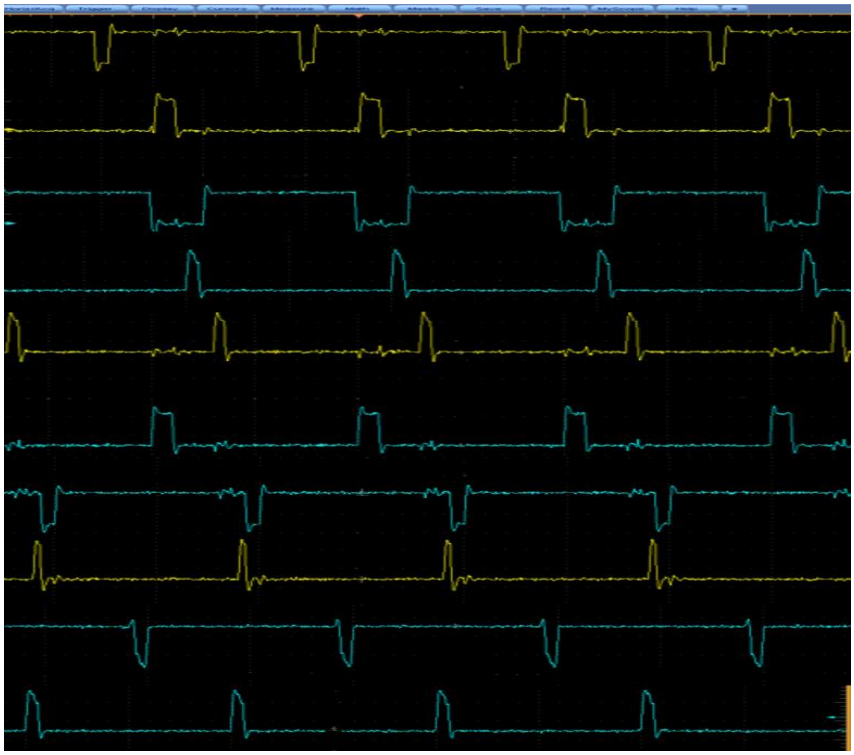

Figure 5. Various clock signals generated to drive the ADC

The figure above illustrates a clock frequency of 1MHz.

The following figure shows the sine wave input into the ADC from the signal generator and output bitstream from the ADC based on the clock configuration highlighted above:

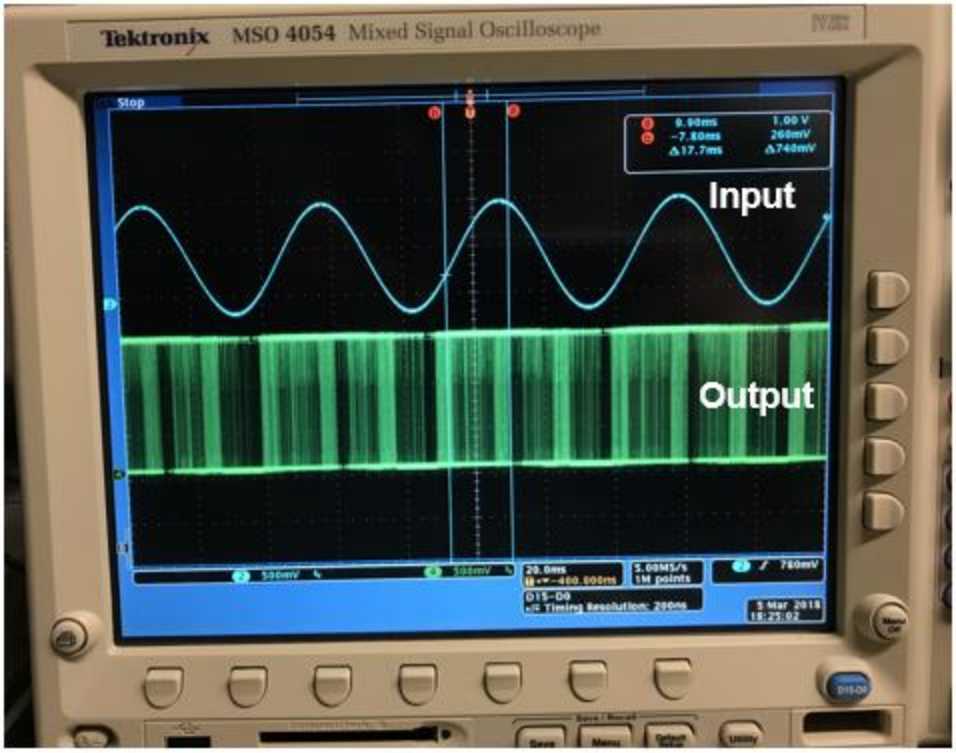

Figure 6. Signal generator input and output bitstream of ADC

The figure above shows an input frequency of 100KHz. From the oscilloscope output, it can be seen that the bitstream contains a significant amount of '0's at the lowest point of the sine wave whilst a significant amount of '1's can be observed at the peak of the sine wave. This indicates that the clock generation module drove the Delta Sigma ADC successfully.

The bitstream was transferred from the Nexys Video to the computer in real-time over the ethernet interface.

The python server created on the computer reads the incoming data from the specified IP address and port, and stores the data in a file. The format of the contents of the file is made to be such that MATLAB can read each bit separately and create a data matrix.

The figure below illustrates the FFT response of the bitstream of a sine wave of input frequency = 1KHz:

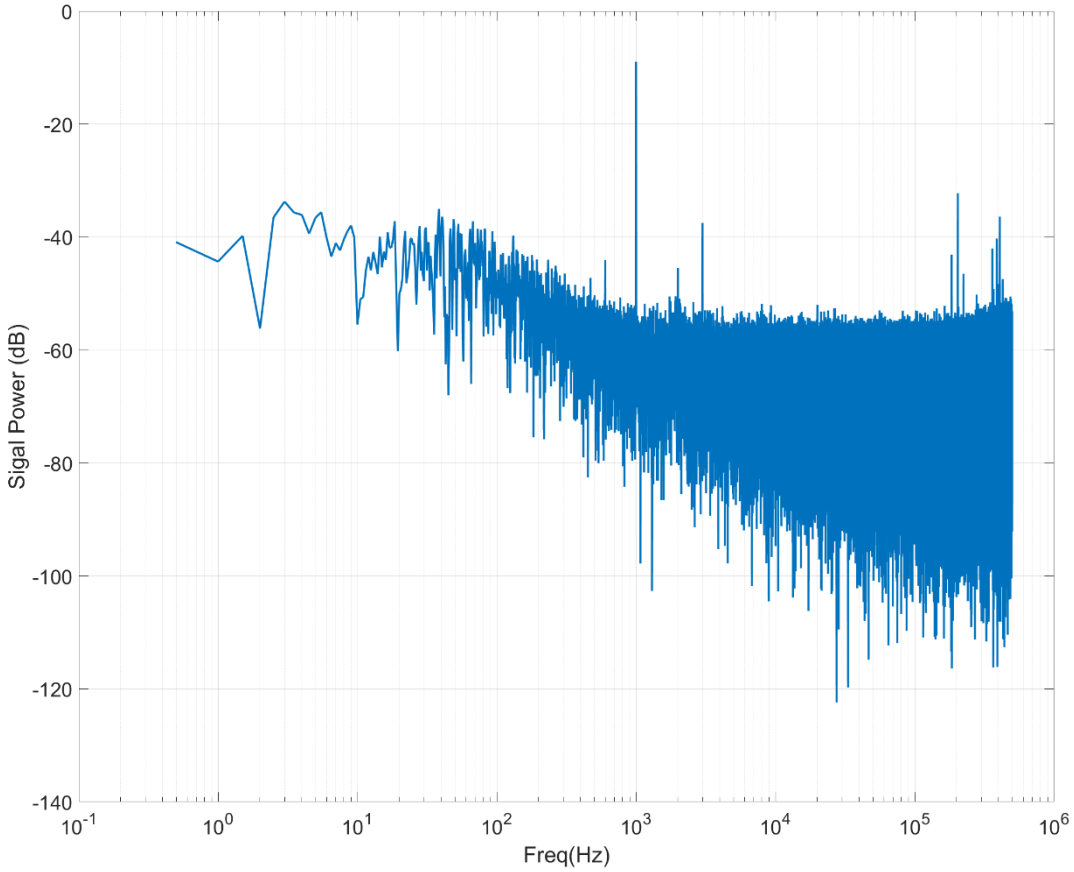

Figure 7. FFT response obtained from MATLAB

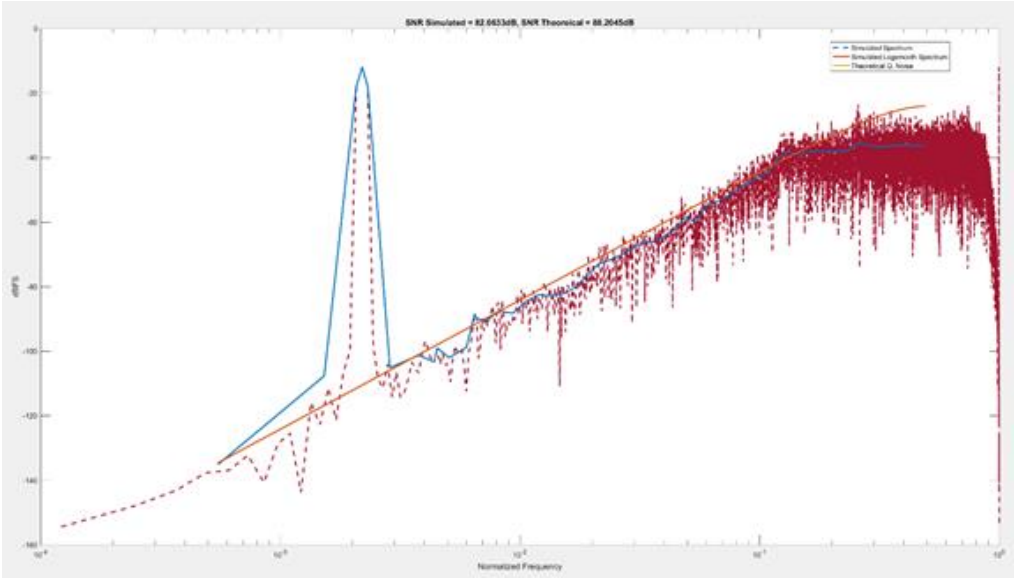

Figure 8. Ideal FFT response shape

The FFT plays two roles in this project, one is characterizing the ADC itself and the other is analysing the frequency of the input signals.

Quantization noise is considered to be the difference between the input signal and the output bit-stream. If the input single is 0.5V and the output bit stream is 1V, then the quantization noise is 1 – 0.5 = 0.5V. In other words, $error_{quantization\ noise} = V_{bitstream} - V_{input}$. Thus, quantization noise can be considered to be white band noise. The second order delta sigma ADC has a noise transfer function of $(1 - z^{-1})^2$. The spectrum of noise transfer function of the ADC should ideally as in Figure 8. As seen in Figure 7, the FFT response of the ADC does not match the ideal response in Figure 8. This is further reflected in the decimated output and low-pass filter output in Figure 9.

2.1.2 Signal Processing

The operation of the Delta Sigma ADC needed to be confirmed prior to incorporating it with the entire system. The decimation filter down-sampled and averaged the incoming bitstream of the ideal sine wave input from the Delta Sigma ADC. The decimation filter output was then passed into the low-pass IIR filter with Fc = 5KHz. The resulting digital sine wave output is illustrated in the simulation below:

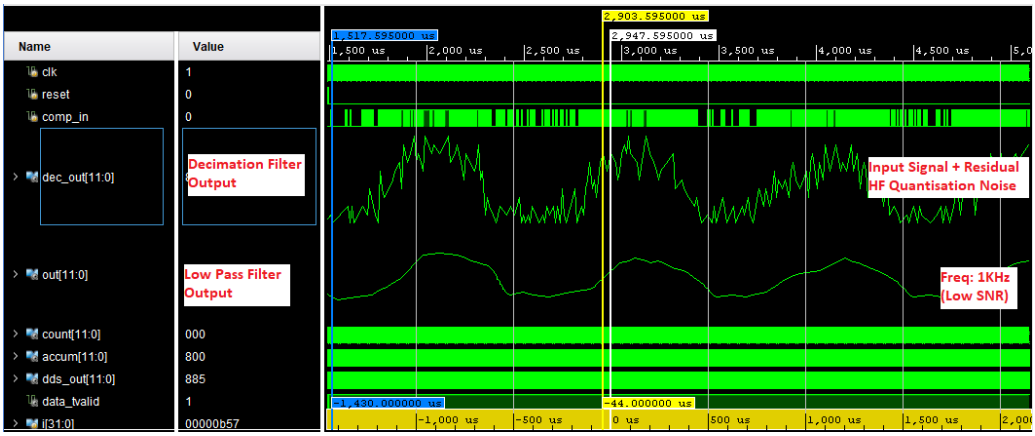

Figure 9. Simulation illustrating decimated output and low pas filter output

As seen in the figure above, the output of the low-pass filter was not the ideal sine wave output expected. It can be seen that this is due to the high frequency quantization noise present in the Delta Sigma bitstream, and is reflected in the filter outputs above. It was, therefore, determined that further validation of the Delta Sigma ADC is required as this could further affect the results of the other parts of this project. The ADC on-board the Nexys 4 DDR board was therefore used to achieve the milestones of this project.

The PMOD DAC was driven such that it sweeps the frequency of the output sine wave from 50Hz to 10KHz in steps of 5Hz, with 200 clock cycles for every frequency. The amplitude of the DAC output

was also attenuated gradually with increasing frequency in order to test the reliability of the phase/amplitude detection modules.

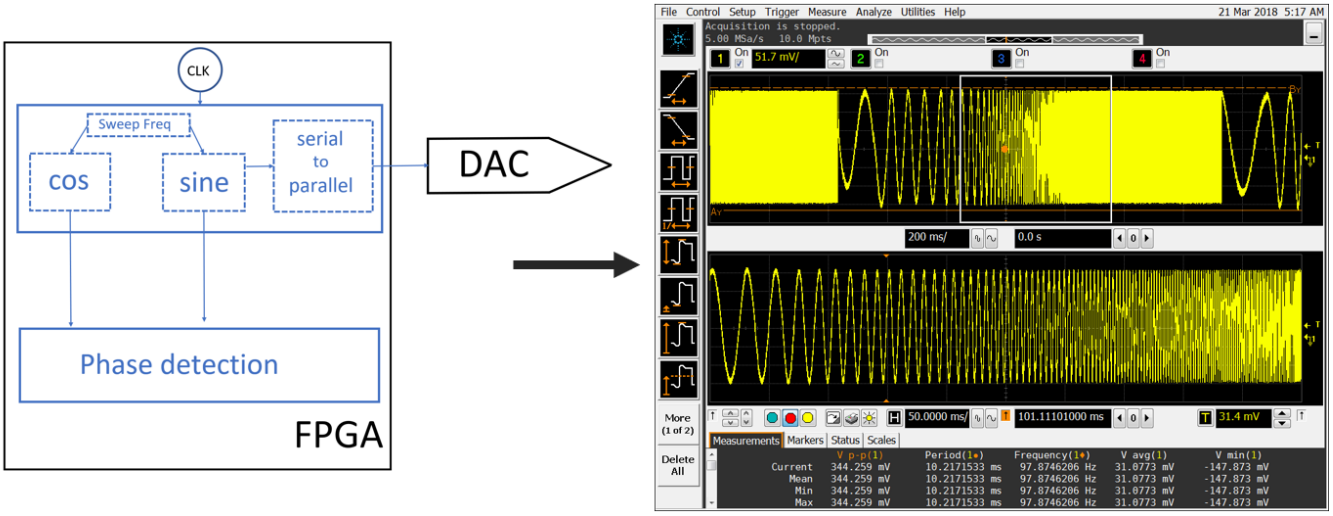

Figure 10. DAC output displayed on oscilloscope

The DAC output was connected to one port of the two-port network whilst the other port was connected to the input of the Nexys 4 DDR XADC. The output of the ADC was passed through the decimation filter, low-pass IIR filter. Finally, the phase/amplitude detection modules distinguished the phase and amplitude difference between the output of the low-pass IIR filter and the reference signal from the DAC.

2.1.3 HDMI External Display

The output of the phase/amplitude detection modules was read through Xilinx SDK and displayed on an external monitor over the HDMI interface, as illustrated below:

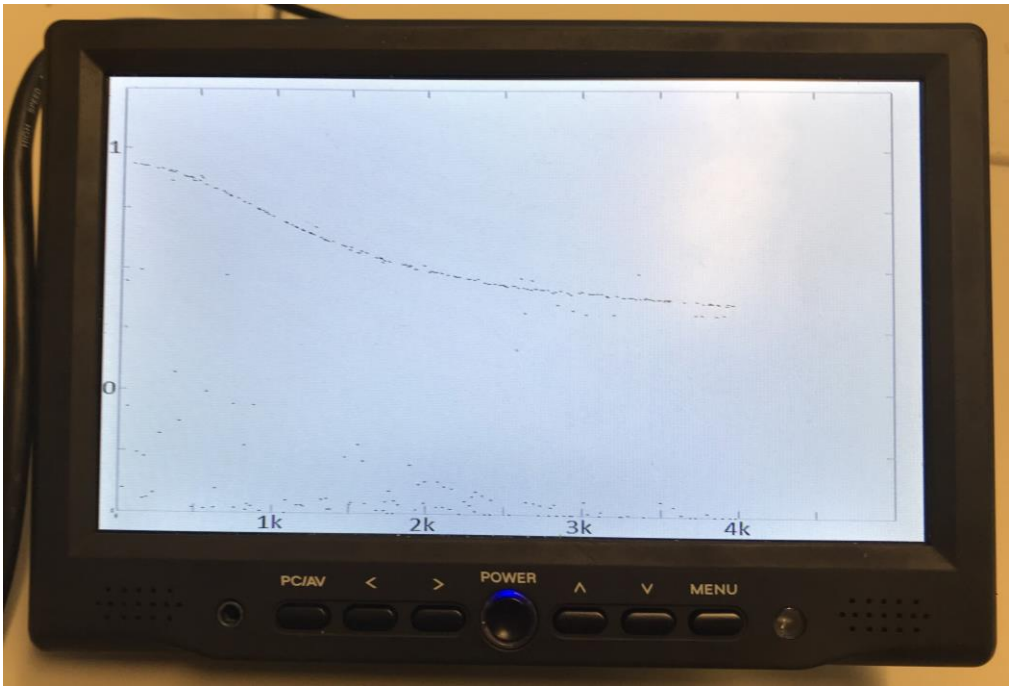

Figure 11. Amplitude response displayed on HDMI monitor

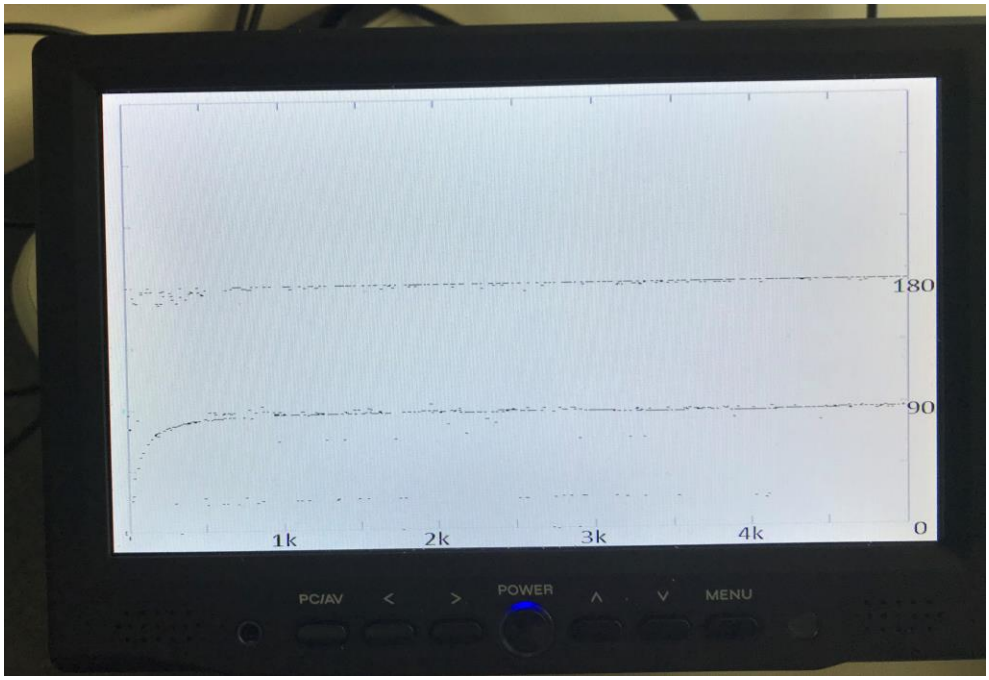

Figure 12. Phase response displayed on HDMI monitor

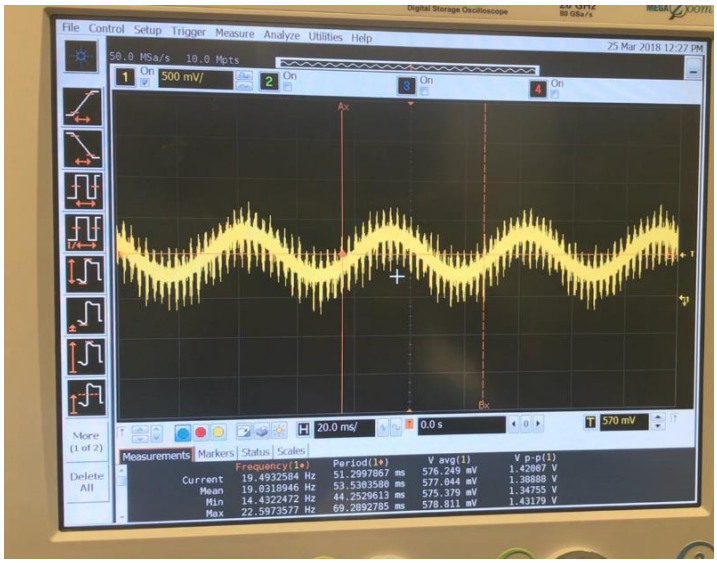

Figure 13. Output of the XADC

It can be seen from the XADC output that noise specks are present in the output, which results in the phase response in Figure 12 detecting a lot of phases differences as 180° or 0°. These results are, therefore, treated as anomalies. From the oscilloscope outputs at different frequencies, as seen in Figure 10, apart from expecting the amplitude to reduce, it is also expected that the phase difference between the two signals will increase as the frequency of the two signals increases.

As seen from the monitor outputs, the amplitude difference between the reference signal and the output of the IIR filter increases as the frequency increases, as was programmed in the DAC driver. Similarly, the phase of the two signals is the same initially, and approaches 90degrees as the frequency of the signals increases. This matches the results expected and the observations made from the oscilloscope output, and hence the phase/amplitude detection modules as well as the display module functioned accurately.

The system therefore met the initial specification and the primary goals of the project were achieved successfully.

Project Schedule

3.1 Milestones

Initially Proposed Milestones:

Milestone #1: Show test-bench with ADC working. The goal is to drive 1-channel of ADC to generate bit stream.

Milestone #2: Show test-bench with Ethernet transfer and data display working, transfer data to computer through Ethernet, and display test data on computer accurately in standard format.

Milestone #3: Get different channels working on computer. Additionally, implement variable clock frequency and test the result. Finally, optimize results through filter simulation in MATLAB.

Milestone #4: Get picture display working on VGA with software driver. Design custom IP block (counter, filter, and so forth).

Milestone #5: Mid-Project Demo. Get data display working on VGA. Show test-bench with DDS working. Furthermore, finish the design of counter and filter.

Milestone #6: Get DFT and IDFT working. This will the most intensive task for this milestone. Additionally, display spectre on computer and VGA monitor.

Milestone #7: Final Demo Done! Obtain the frequency and phase response from two-port network, and display the result on VGA monitor

Actual Milestones:

Milestone #2: Successfully transferred real-time data over Ethernet interface from Nexys Video to computer and became familiar with the HDMI interface

Milestone #3: Designed variable frequency clock generation module (contained jitter) and successfully stored data from ethernet interface to a file

Milestone #4: Improved total negative slack of clock generator, commenced digital low-pass IIR filter design and image could be displayed on monitor

Milestone #5: Fully optimised and pipelined clock generator design and concluded digital low-pass filter design

Milestone #6: UDP transferred block functioned accurately with variable data rates, designed decimation filter, plot on HDMI monitor and drove the DAC to sweep the frequency

Milestone #7: Designed phase and amplitude detection modules, drove Nexys 4 DDR ADC, integrated the system and displayed results on external monitor

3.2 Project Deviations

As mentioned under the risk plans in the project proposal, a significant learning curve was involved in order to drive the ADC. The clock generation module used to drive the ADC took significantly longer than expected. As reflected in the initial and actual milestone plans, this task was planned to be completed in the third week of the scheduled project plan. The ADC could successfully be driven at fixed frequencies. However, with a variable frequency mechanism, jitter was observed at certain frequencies and significantly more time (approximately 3 weeks) had to be spent on creating a highly optimised and pipelined design. This also affected the UDP Ethernet module, which also had to be revised in order for it to transfer variable data rates. This restricted the group from working on stretch goals of the project, namely, implementing DFT in hardware.

Another deviation from the initial project plan was the use of the fully custom ADC in order to determine the frequency response. As covered in the previous section, high-frequency quantisation noise was observed at the output of the ADC, and it was therefore decided to use the Nexys 4 DDR on-board ADC in order ensure the project outcome is not affected.

The Nexys Video board was used as the primary FPGA board and HDMI interface was there utilised in order to display the results, as opposed to the VGA interface on Nexys 4 DDR.

Development of other proposed milestones, such as filters, phase and amplitude detection, was on schedule apart from the constant improvements made after the initial designs were concluded.

Description of Blocks

4.1 UDP Ethernet

This module was a semi-custom IP that was based on a core from Hamsterworks with little to no instructions/guide available about the function of the core or instructions on using the core. This core originally used 4 GPIO switches on the Nexys Video board to set the transmission rate, ranging from 1 packet per/s to 152,439 packets/s. Only header bytes were transferred in every packet with no data bytes.

In order for the core to be able to communicate with the computer, a static IP address has to be assigned to the computer's ethernet adapter. A static IP address of 192.168.1.11 can be assigned to the computer while the FPGA is assigned 192.168.1.10. The adapter's MAC address also needs to be found by checking its IPV4 properties. The core was updated with the source/destination IP addresses, MAC addresses and ports respectively. The user needs to ensure that the destination port set in the ethernet module is a registered port (between 1024 and 49151) as ports 0-1023 are system ports used for various system protocols (Wikipedia, Port (computer networking), 2018). The data from the FPGA was transmitted to port number 5010.

The format of the packet is such that the packet is constructed of 1082 bytes in total- 42 header bytes and 1040 data bytes. Header bytes contain information such as source and destination IP addresses, source and destination MAC addresses, UDP ports.

Since the module is able to transmit 1040 data bytes in a single packet, and the Delta-Sigma bitstream therefore has to be accumulated for 1040 clock cycles before a packet can be constructed. For this purpose, the incoming data from the ADC was stored in a dual distributed memory block, clocked at 125MHz. When 1040 bytes have been stored, the clock generator module asserts the 'start_sending' signal, which triggers the ethernet module to construct the header bytes. The module then uses a counter to read each data byte from the distributed memory and construct the packet. As the frequency of the ethernet module is 125MHz, the module is able to read the data much faster that the rate at which data is written to the memory (maximum freq = 10MHz). The data from the first memory location is read before new data is written to this location and, therefore, no data loss occurs.

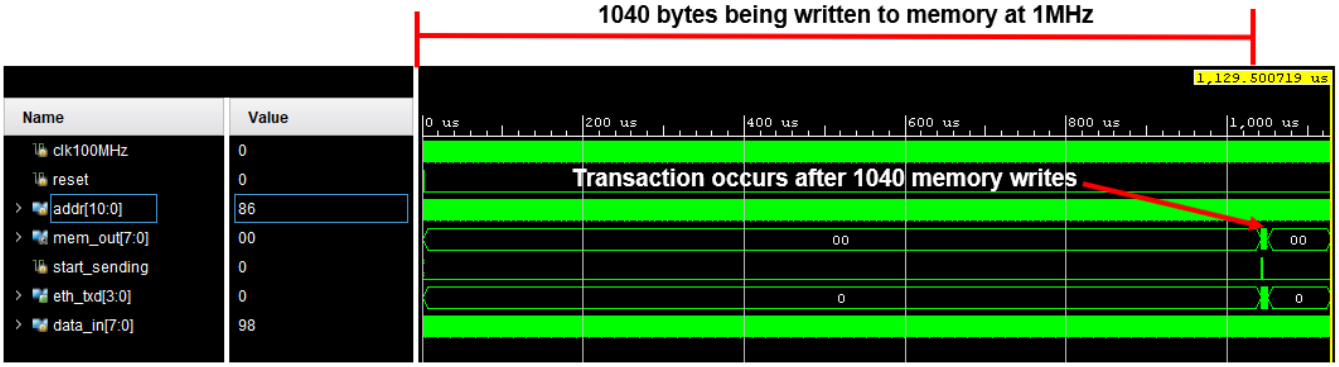

Figure 14. Simulation of 1040 bytes being written to memory and packet being constructed at 125MHz

Data continues to be constructed into packets in this manner and is transmitted in real time.

4.1.1 Python Server

On the receiving end, WireShark application can be used to monitor the network traffic through a specified IP address. This confirms the format of the data being transmitted by the FPGA:

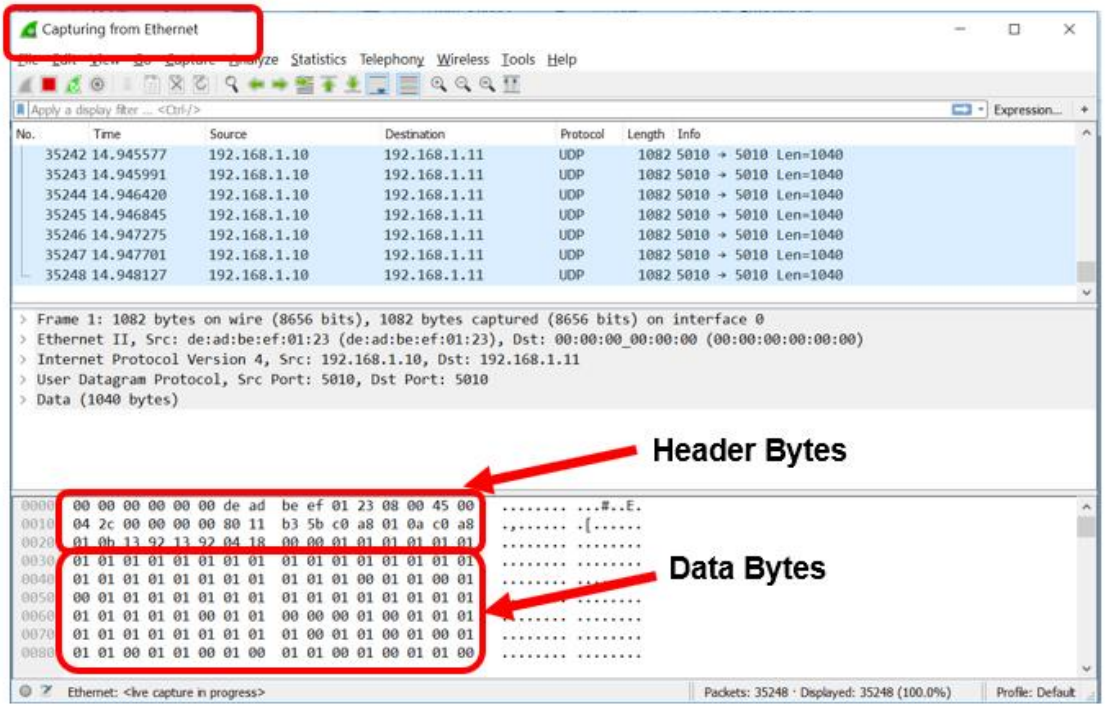

Figure 15. WireShark output from address 192.168.1.10

As seen in the figure above, each packet contains a set of header bytes followed by the actual data transmitted.

A Python Server was setup to receive and store the data in a file. The script binds to the specified IP address and port, checks for incoming data while and stores each packet on a single line. Once the user decides to stop the transmission from the FPGA (using the global reset button), the programme creates temporary files to format each data bit onto a new line.

```
data,addr = s.recvfrom(buf)
try:
    while(data):
        s.settimeout(0.5)
        data,addr = s.recvfrom(buf)
        z=list(data)
        f.write('%s'% z)
except timeout:
    f.close()
    s.close()
    with open('UDPData_temp1.csv', 'r') as infile, open('UDPdata_temp2.csv', 'w+') as outfile:
        temp1 = infile.read().replace("[", "")
        outfile.write(temp1)
    with open('UDPdata_temp2.csv', 'r') as infile, open('UDPdata_temp3.csv', 'w+') as outfile:
        temp2 = infile.read().replace("]", ",")
        outfile.write(temp2)
    with open('UDPdata_temp3.csv', 'r') as infile, open('UDPdata.csv', 'w+') as outfile:
        temp3 = infile.read().replace(",", "\n")
        outfile.write(temp3)
    print ("File Downloaded")
```

Figure 16. Section of Python script to receive and store packets

4.2 FFT Analysis

FFT analysis was carried out using MATLAB. In order to ensure that the MATLAB code functions correctly, code to test a conventional second order delta sigma ADC was used and was compared with the ideal result.

4.3 Decimation Filter

The decimation filter is intended to down-sample the over-sampled bitstream coming in from the Delta-Sigma ADC. The mechanism implemented is such that the logic utilised an accumulator that has a reference value of 2048. On every rising edge of the clock, the logic determines if the input is 1 or 0, and adds/subtracts the accumulator value accordingly. Every N clock cycles the current accumulator value is set as the decimation filter output and the accumulator value is reset to 2048. The number N varies with the frequency of the signals from the clock generator, and hence ADC the oversampling ratio. The number is controlled by the SDK programme such that the number of samples per one cycle of sine wave is always 100, which assures a high-resolution output.

This decimation mechanism outputs a 12-bit signal that roughly represents the original sine wave input into the ADC when the 12-bits are observed in analogue representation. The high frequencies in this output further needs to be filtered out to produce a smoother output. This is done using the digital low-pass IIR filter.

4.4 Digital Low-Pass IIR Filter

The Digital Low-Pass IIR Filter takes in a specified multi-bit input signal, representing an analog of a particular frequency, and filters out frequencies above 5KHz.

The filter was used to filter out the high frequency noise from the decimation filter output. An IIR filter was settled upon as IIR filters are more efficient and require less computation resources as opposed to an FIR filter as IIR filters are able to match a specification with a much lower order (MathWorls, 2018).

The Filter Design and Analysis tool in MATLAB (FDATool) enables users to design filters with filter types, specified cut-off frequencies and bandwidths. In order to filter out noise from the decimation filter output, a 4^{th} order IIR Butterworth filter with F_c = 5 KHz and F_s = 100 KHz was designed in FDAtool.

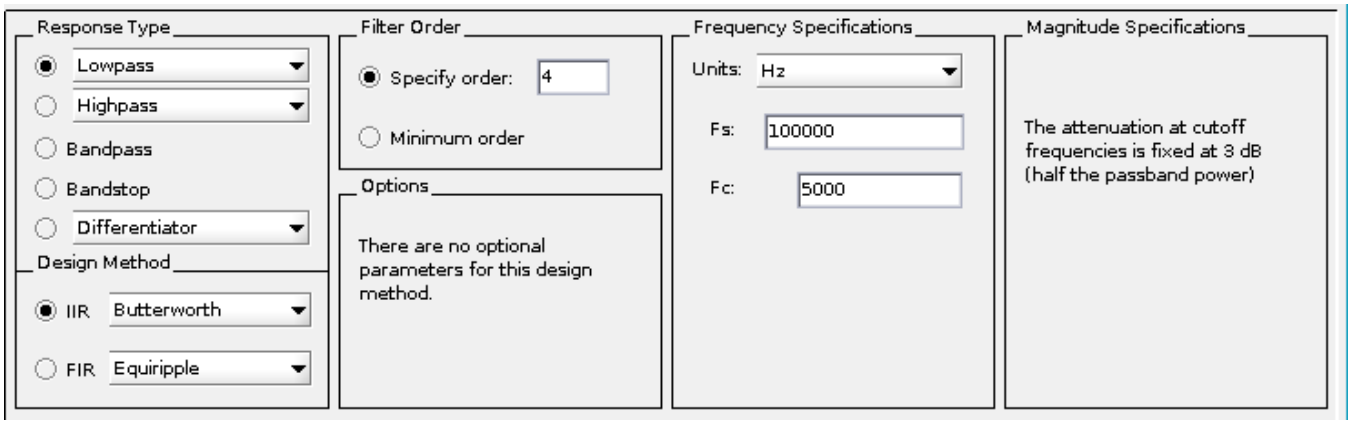

Figure 17. Specifications of the IIR Butterworth filter designed

The theoretical response of the filter is as follows:

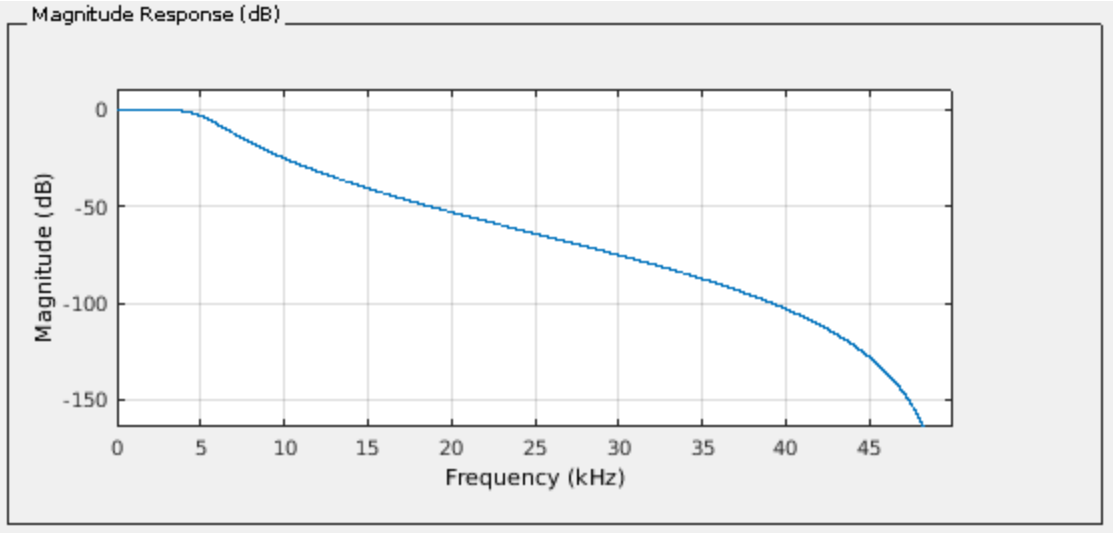

Figure 18. Theoretical filter response

The filter can be realised by clicking on the 'Realise Model' using basic elements. The resulting model illustrates the elements and coefficients of the filter. This particular filter has 2 sections.

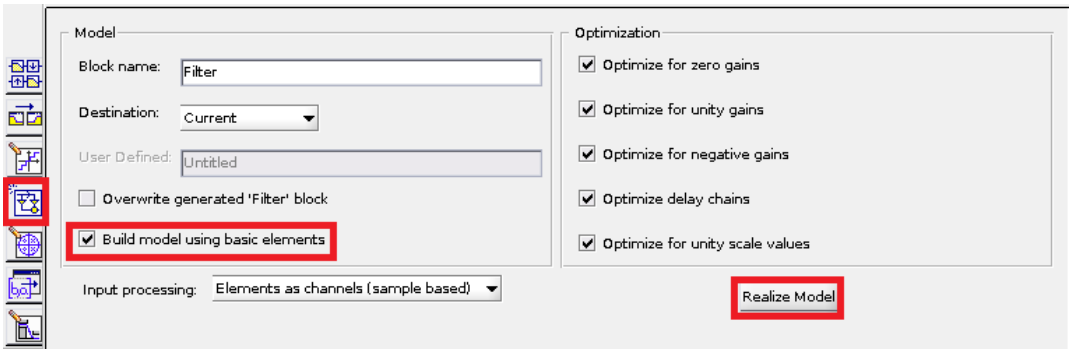

Figure 19. Realise model window

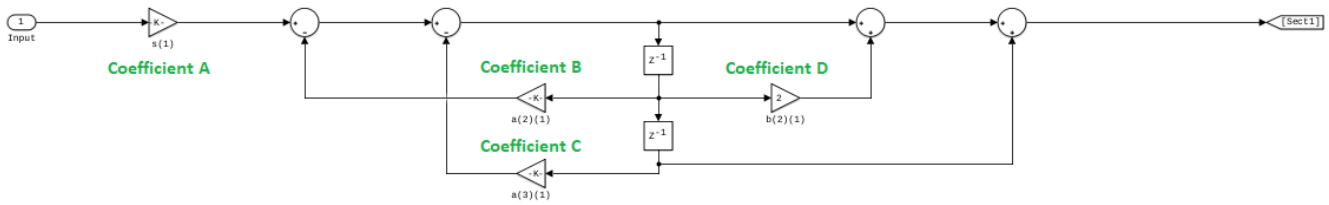

Figure 20. Illustration of one section of the filter

Using the FDATool window, the coefficients can be exported into a separate file : File > Export > Coefficient File (ASCII). The coefficients in the file are represented in the following format:

```
% Coefficient Format: Decimal

% Discrete-Time IIR Filter (real)
% -------------------------------
% Filter Structure    : Direct-Form II, Second-Order Sections
% Number of Sections  : 2
% Stable              : Yes
% Linear Phase        : No

   D
SOS Matrix:           B                                        C
1  2  1  1  -1.9681033112560972764271127743995748460292  0.969074930869832762425630789948627352715
1  1  0  1  -0.969067417193793301244397753180237486959  0

Scale Values:  A
0.000242904903433892478941541481773924716
0.015466291403103363255588931224338011816
```

Figure 21. Contents of coefficient file

Note the coefficients placement for each element in Figure 20 and Figure 21. A coefficient is provided for each section in the realised filter model, in ascending order.

The target is to implement the filter in FPGA hardware and floating point numbers shown in the coefficients file above therefore need to be converted to fixed point. In order to do this, an excel spreadsheet can be utilised. The smallest coefficient in the filter design needs to be considered and the remaining coefficients need to be scaled with this value.

2^2	2^1	2^0	.	2^-1	2^-2	2^-3
4	2	1	.	0.5	0.25	0.125

As seen with positive powers of two, this is the minimum number of bits required to cover a given range of integers is given by this number. Similarly, in order to cover the smallest coefficient in the filter design, 12-bits are required to represent the floating point coefficients:

$$2^{-12} = 0.00024414$$

The number above is almost equivalent to the smallest coefficient. At least 12 bits are therefore required to accurately represent the floating-point coefficients. All coefficients then need to be scaled with this number:

0.000242905	-1.968103311	0.969074931	2	0.000244141		1	-8061	3969	8192
0.015466291	-0.969067417	0	1	0.000244141		63	-3969	0	4096

In terms of hardware design, the basic elements are created as separate functions and instantiated at a higher level to construct the section(s) illustrated in Figure 20:

- Add - output = $input_1 + input_2$

- Subtract - output = $input_1 - input_2$
- Delay - output = input

The multiplication element is required to have a multiplication compensation factor that is equivalent to the number of bits accounted for when carrying out the floating point to fixed point conversion. In this case, it is 12-bits. The output therefore needs to be divided by $2^{12} = 4096$:

- Multiply - output = (input x multiplication factor) / ($2^{\text{number of floating point bits}}$)

Once the functions have been instantiated as illustrated in the filter design, a multi-bit input can be passed into the digital low pass filter. The width of the filter input must be set at the top-level of the filter, and the rest of the design adjusts the passage of data accordingly.

4.5 Frequency tuner

4.5.1 Variable frequency tuning for the clock generation and optimization

In order to generate variable clock signals, binary counters were utilised which were driven by 500MHz clock signals and were placed into DSP blocks. When the counter is at a certain value, the clock generation block outputs high signals. When the counter reaches a certain value, the clock asserts a reset signal that reset the counters. The block diagrams are as shown below:

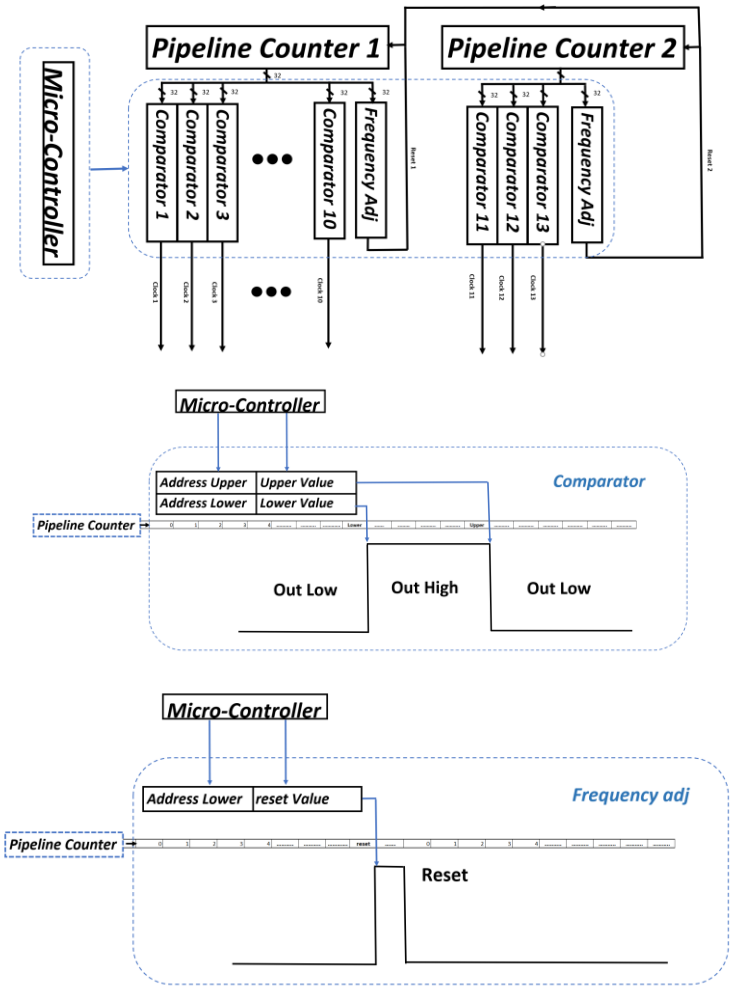

Figure 22. Block diagram illustrating the operation of the binary counters

In order to optimise the clock generation block, the clock generator was pipelined, and the comparators were mapped into DSP blocks.

While the module is custom-designed, the binary counters used were provided by the Xilinx. To ensure that the clock generator was functional, the output of the clock generation block was monitored through an oscilloscope to determine whether the clocks are generated without jitter. Furthermore, the clock outputs were connected to the custom ADC to observe whether the ADC was able to generate the bitstream correctly, which was also monitored on an oscilloscope, as seen in Figure 6.

4.6 Synchronising UDP with Clock Generator

The Clock Generator was used to generate a reference signal which was fed into the UDP block. The UDP module utilised this reference clock to ensure that the Distributed Memory samples the data (bit stream) at the correct frequency and exactly at the midpoint of a sample.

An FFT analysis was carried out in MATLAB to confirm whether the data being receive through the UDP was accurate, and therefore, whether the UDP Ethernet block was accurately synchronised with the clock generator.

4.7 SDK Controller

The SDK programme writes to memory addresses of various control signals. The functions of the programme can be summarised as follows:

- Control different channels of the ADC
- Sets the frequency and the phase of the clocks generated by the Clock Generator.
- Read from DDR and display background picture on HDMI monitor
- Read amplitude and phase data from BRAM and plot the result

4.8 Phase Detector

The phase detector module takes two 16-bit input signals, a reference signal and a test signal, and returns a signed output value between -180 to 180, indicating whether the test signal is leading of lagging the reference signal.

The logic detects the most significant bit of the input signals and compares the phase of these two signals. As this module was to be used in conjunction with the DAC that sweeps the input frequency, as mentioned previously, the MSB register was reset with every changing frequency and the phase detector determines the most significant every time the frequency changes.

Four flags are asserted for the rest of the phase detector logic to initially determine the state of the signals: Lead Flag, Lag Flag, Null Flag and Equal Flag. These flags are used in conjunction with a counter that is reset once every cycle of the input signal. The total counts therefore varies with the input frequency. The phase detector module, therefore, has a frequency input that indicates the current frequency, and the total count is given by the system clock frequency(100MHz)/frequency.

- Lead Flag – if MSB of test signal is high before the MSB of reference signal, and counter value is **less than half** it's total counts, the test signal is **leading**.
- Lag Flag – if MSB of reference signal is high before the MSB of test signal, and counter value is **more than half** it's total counts, the test signal is **lagging**.
- Null Flag – if MSB of test signal is high and counter value is within ((Total Counts ± (Total Counts/360)), the reference and test signals are **in phase**.
- Equal Flag – if MSB of test signal is high and the counter value is within ((Total Counts/2) ± (Total Counter/360)), the reference and test signals are **180°** out of phase.

Based on these flags, the logic carries out arithmetic operations within a finite state machine to return the value of the phase difference between the two signals. If either null or equal flags are asserted, the logic outputs the phase as 0° or 180° respectively. However, if either lead or lag flags are asserted, the logic determines the counts between the rising edges of the two MSBs, and the phase is calculated as follows:

$$360 / (\text{Total Counts}/\text{Count Value})$$

If, however, the values are inputted as they are, the phase output would only be multiples of integers as the hardware would not be able to carry out floating point operations:

- 360/10 = 36°
- 360/9 = 40°
- 360/8 = 45°
- 360/7 = 51°
- 360/6 = 60°
- 360/5 = 72°
- 360/4 = 90°
- 360/3 = 120°
- 360/2 = 180°

As seen in the example above, the accuracy and the range of phases differences detectable by the logic in which floating point numbers are not accounted for decreases as the phase difference increases.

As the desired accuracy is up to 1°, the smallest difference that needs to be accounted for is:

$$(360/179) - (360/180) = 0.011173$$

$$0.011173 / 2^{-8} = 2.8$$

Dividing the smallest difference by 2^{-8} gives a number above 1. The values in the arithmetic operations, therefore, needs to be left shifted by 8, or multiplied by 256. Taking a case where the total counts is 5000:

$$360 * 256 = 92{,}160$$

$$5000 * 256 = 1{,}280{,}000$$

- 92160 / (1,280,000/2487) = 179°

The constants were therefore scaled by 256 and the arithmetic operations were carried out using the fixed-point values.

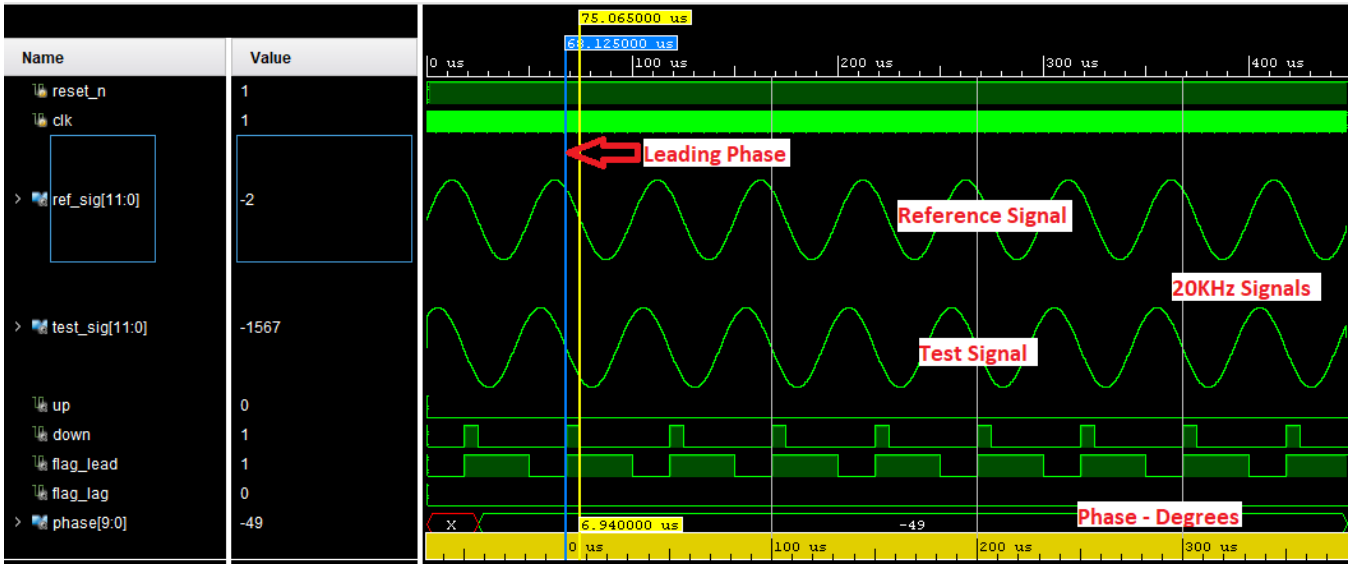

Figure 23. An example simulation of the phase difference between two signals

4.9 HDMI Display

HDMI video display module was built according to the HDMI demo on the Digilent website. The block diagram is shown in Figure 24. When executed, the screen would first show the test pattern as shown in Figure 25. The input video stream passes through a DVI to RGB video format converter and is decoded into a AXI stream format. ”Video In” block is used for transferring RGB data to AXI-4 stream, which can be stored in DDR . VDMA then reads the image from memory and passes it through AXI stream to RGB converter. Finally, an RGB to DVI video encoder is needed such that the frame can be shown on the HDMI monitor. Tera Term is used for entering and printing “user input ”.

MATLAB is needed to produce one picture which would be shown on monitor as background. It must include x axis and y axis for amplitude and phase response.

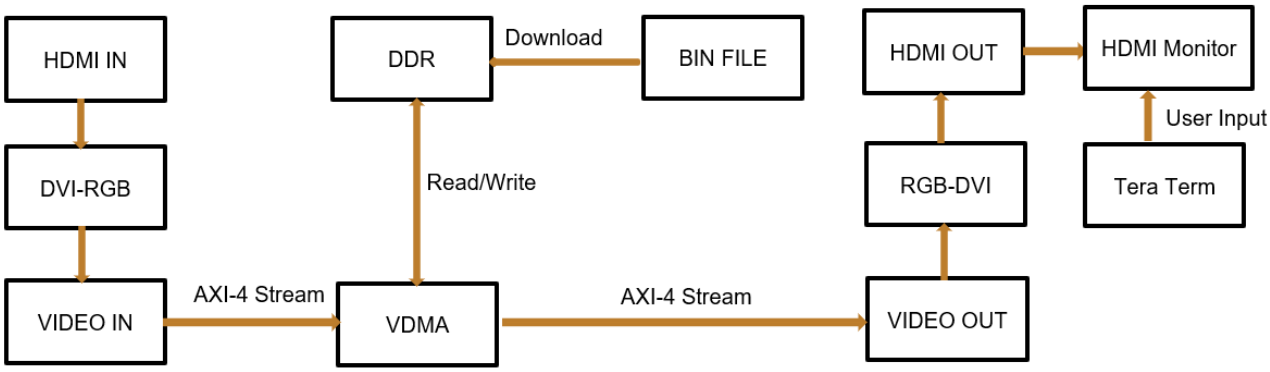

Figure 24. Block diagram of HDMI display module

To make the pictures backgrounds) show on the monitor, we need to change the bmp file into coe file, and then into bin file, which could be loaded in DDR. Relative C codes are, therefore, needed for achieving these conversions

To download the bin file into DDR, some commands are required in xsdb, like “download”, “write” and “read”.

A deeper understating of the code used to configure and control the VDMA was gained.

Finally, a plotting algorithm was implemented to display the frequency response. The x axis shows the frequency while y axis shows magnitude and phase; two thousand points were plotted in total. The frequency ranges from 50Hz to 10kHz with a step of 5Hz and the y value is obtained from the beginning address of phase detector and amplitude detector. If a point is shown at certain pixel, the RGB of this pixel would be 24’h000;.

A “zoom in” algorithm was also implemented to amplify the critical part of the frequency response. Figure 26 illustrates an example of this algorithm. In order to achieve this, values of x coordinate and y coordinate are changed to twice the original value and a new background is also used, whose unit coordinates are twice of the original unit coordinates.

Figure 25. HDMI test pattern

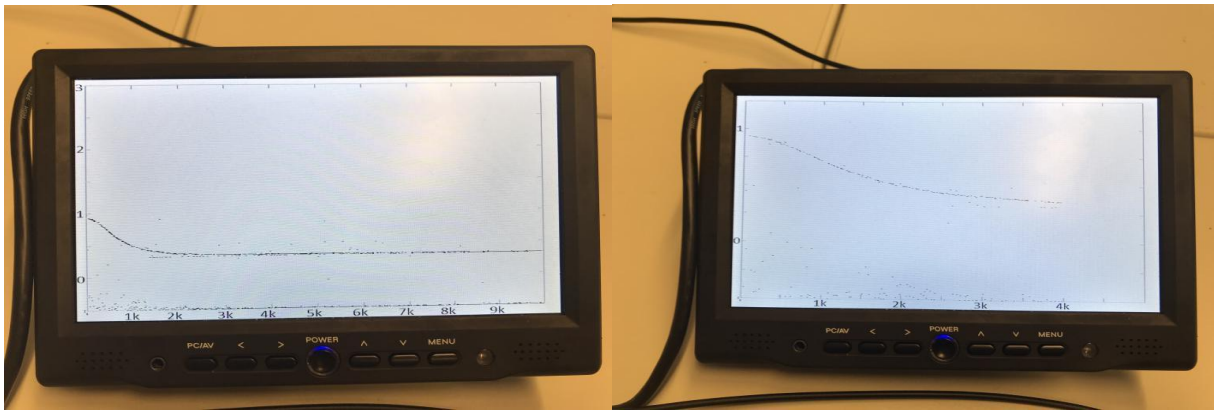

Figure 26. Amplitude plot on the left and magnified amplitude plot on the right, using "zoom in" algorithm

Existing IP:

- DVI to RGB Video Decoder - A video decoder, which translates video signals into RGB data.
- DVI to RGB Video Encoder - A video encoder, which translates RGB data into video signals.
- Video in to AXI4-Stream - converts common parallel video signals to an AXI4-Stream interface.
- AXI4-Stream to Video Out - converts AXI4-Stream interface signals to standard parallel video output interface with timing signals.
- Video Timing Controller - Generate the video timing required for AXI4-Stream to Video Out conversion and generate the video timing required for Video in to AXI4-Stream conversion).
- AXI Video Direct Memory Access - provides high-bandwidth direct memory access between memory and AXI4-Stream.
- DDR Memory - For buffering video frame.

4.10 DAC Driver

The block diagram of the DAC driver and the output signals is shown in the Figure 10. The Sine and Cosine generation algorithm inside the DAC driver block generates 16-bit Sine and Cosine signals. A single Sine/Cosine period consists of 2¬12 clock periods; at each clock period the Sine/Cosine signal changes by 3-bits. By changing the clock frequency used to drive the Sine/Cos, the frequency of the DAC output can be swept in specific intervals.

4.11 XADC and 7-segment display

The XADC and 7-segment display modules provided by the Xilinx were modified and were used to display the amplitude response in real time.

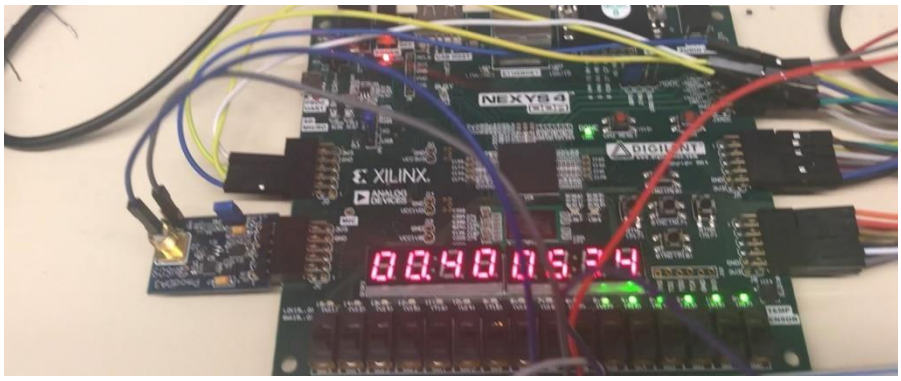

Figure 27. 7 segment display showing amplitude data in real time

4.12 OLED display

The OLED display module provided by the Xilinx website was modified and used for displaying the real time frequency data, phase response and amplitude data. The IP was tested by comparing the displayed results with the oscilloscope output.

4.13 Amplitude Detector

The amplitude detector receives the frequency indicator signal from the DAC driver, the output of the XADC from the Nexys 4 DDR and outputs the difference in the maximum and minimum amplitude of the XADC signal. The logic is such that when the frequency changes, the maximum and minimum values are reset and the amplitude difference is stored in a new register. This value is then set as the new output of the amplitude detector.

Description of Design Tree

Github Repository : https://github.com/jaiminj/-ece532-EITgroup13

Nexys-Video-HDMI : main project design

- **proj/HDMI.xpr** : Nexys Video project file

MBS_V1 : back up standalone clock generator project if user has access to a spare Nexys Video board, such that TNS is not affected by other modules in the project

- **MicroBlazeServer.xpr :** Stand alone clock generator project file

IPs : All custom and semi-custom IP repository

-**CLK_V5** : clock generator and frequency tuner IP

-**DAC** : DAC driver IP

-**DecimationFilter** : decimation filter IP

-**FPGA_ADC** : IP to display ADC data on OLED

-**Filter_IIR** : low-pass IIR filter IP

-**IP_for_Testing** : counter IP to test UDP module

-**PhaseDetector** : phase detector IP

-**UDP_V2** : UDP Ethernet IP

-**ampDec** : amplitude detector IP

-**control_v1** : controller IP for ADC

DDRXADC : Nexys 4 DDR ADC project

- **proj/XACD.xpr :** XADC project file

Doc : documentation

-**FinalReort_Group13.pdf** : group report

References

Aadler. (n.d.). *Modeling EIT current flow in a human thorax model*. Retrieved from EIDORS: http://eidors3d.sourceforge.net/tutorial/netgen/extrusion/thoraxmdl.shtml

Hamsterworks. (n.d.). *GigabitTX*. Retrieved from http://hamsterworks.co.nz/mediawiki/index.php/GigabitTX

MathWorls. (2018). *IIR Filter Design*. Retrieved from https://www.mathworks.com/help/signal/ug/iir-filter-design.html

Wikipedia. (2018, April 4). *Delta-Sigma Modulation*. Retrieved from https://en.wikipedia.org/wiki/Delta-sigma_modulation

Wikipedia. (2018). *Electrical Impedance Tomography*. Retrieved from https://en.wikipedia.org/wiki/Electrical_impedance_tomography

Wikipedia. (2018, April 5). *Port (computer networking)*. Retrieved from https://en.wikipedia.org/wiki/Port_(computer_networking)

End of Report