FPGA/ASIC Course

Course Project

Phase 1 Report

Final Project | Phase I Report

Due on Monday, June 23, 2024

Parsa Parsamanesh - SID: 99106888

Mohammad Ghafourian - SID: 99106493

Shervin Mehrtash - SID: 400102052

Spring 2024

1 | Brief Explanation:

The final project for this course concentrates on the application of signal processing techniques using Zynq Boards.
In the initial phase of the project, the primary objective is to implement an FFT (Fast Fourier Transform) block to analyze and manipulate the provided signal data.
Subsequently, the project will progress to encompass additional components such as the integration of visual displays to showcase the processed results on a monitor through the HDMI port. This comprehensive approach will allow for a thorough exploration of signal processing capabilities within the context of Zynq Boards.
As explained in the given manual, we start the process by generating number of waveforms (Simple Pulse, Sawtooth, Triangular, and Sinusoidal) in PS section of the Zynq board. One of the selected waveforms will be transferred to PL section using proper configuration and will be delivered to the FFT block to perform the desired operation on the received signal.
By doing this, we would have two signals ready to be displayed on the monitor, the initial waveform, followed by another processed signal passed through FFT Block.
It's worth to mention that our given structure is different from what we were provided in the manual, as we return the FFT-applied signal to the PS and follow rest of the flow from that point.
In the next section of the report, we will delve into the details of each step.

2 | Summary of what we did:

The following is the schematic of the block design:

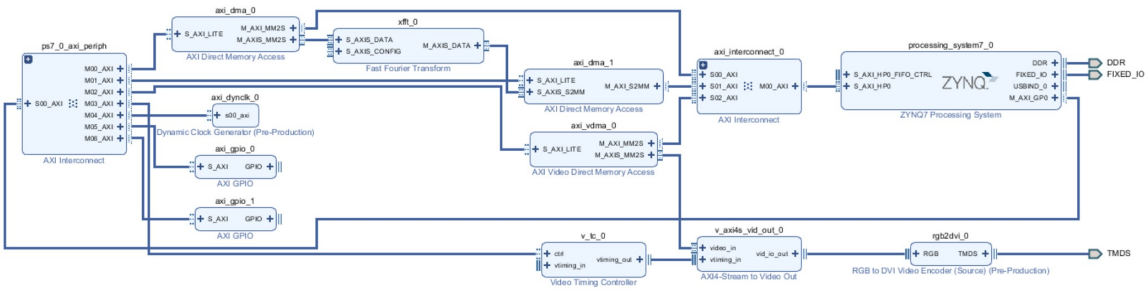


Figure 1: The final block design of the phase 1

Above block design would result in displaying both the initial signal and a secondary signal passed through FFT block on the connected monitor via HDMI port.
It should be highlighted that we were given two different projects including essential blocks for the first phase of the project. However, we started from scratch and put the blocks together to function properly.

Steps are explained below:

- Different Waveforms are generated in the ZYNQ7 Processing System (PS) and will be stored in the DDR Memory respectively. PS Block is connected to peripheral block to be properly configured, and it's also connected to an AXI Interconnect to transmit and receive the desired data.
- As we have our waveforms ready to be used, they'll get passed to a DMA Block to be read and converted to Stream.
- The FFT block will receive a stream of the selected waveform and applies FFT on the received signal.
- The final signal (which is FFT of the primary received waveform) should be stored in the memory again. To do so, the FFT-applied signal will be transferred to another DMA block to convert the Received stream to Memory Map and store the Secondary signal in DDR Memory respectively.
- At this point, we have both signals in the DDR Memory, so we can proceed with displaying section.
- Both stored signals (Main and FFT-applied) will be passed to an AXI Video DMA Block to be converted from Memory Map to Stream.
- This DMA block will pass the signal to AXI4-Stream to Video Out to convert the stream to a suitable format for display.

Points that need to be highlighted:

1. AXI Peripheral Block is responsible for proper configuration of other blocks, so that they can function properly.
2. AXI GPIO blocks are responsible for Interrupt handling.
3. The last block in the mentioned schematic requires two different clock signals, so a Dynamic Clock Generator is placed in the design to handle these signals.

3 | Challenges we faced:

The schematic of the design is attached above in this report. However, there has been number of challenges we've faced during this phase. Number of these critical configurations are listed below for more clarification of what has been done:

- FFT Block needs to be configured properly in order to function smoothly. Configuration of other blocks are handled using peripheral block. However, FFT Block couldn't be configured using mentioned block. As a result, two Constant 1s were connected to the config. ports of the FFT block to ensure its functionality.
- There has been number of challenges with the width of Input and Output Data. According to the setup of DMA block, it's working with 32-bit input/output data, while we've stored 16-bit data in the PS (DDR Memory). To ensure the proper functionality, we need to convert the stored 16-bit data to 32-bit, considering that the FFT block interpret the first half of the data (LSB Side) as the Real component, and the MSB Side as the Imaginary Component. By adding zeros in between our stored data, we can ensure that proper data is delivered to the FFT block, and the interpretation is correct.
- VDMA Block converts received data to 24-bit output data, which is three 8-bit data representing RGB values. The default input width is 64 bits, but we changed it to 32-bit input in order to be synced with other blocks.
- During the Synthesis process, we've encountered a critical warning as a result of different clocks we have in our setup. These different clocks were only used to ensure the functionality of last block (RGB to DVI Video Encoder) and thus, the mentioned warning could be ignored.
- Video Timing Controller Block should be configured properly with respect to the LCD (Monitor) we use.

4 | Simulation Section:

To assess the functionality of the FFT Block in Vivado, we must adopt a verification methodology to confirm its correct operation. This involves utilizing a testbench file that is designed to evaluate the specified structure.

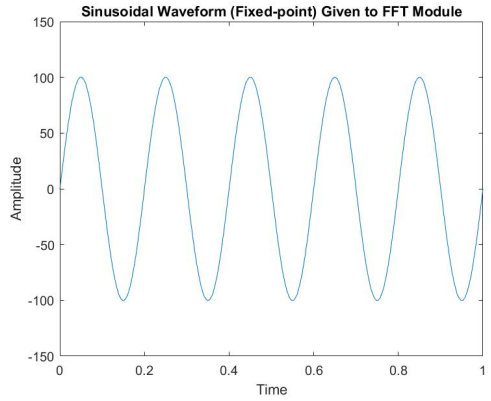
Testbench file needs data to feed the FFT block. The required data could be generated randomly inside the verilog script, or using a MATLAB script. As mentioned in the manual of this project, we first generated required data in MATLAB environment and loaded them into the verilog testbench respectively.

The following is the MATLAB script used to generate required waveforms:

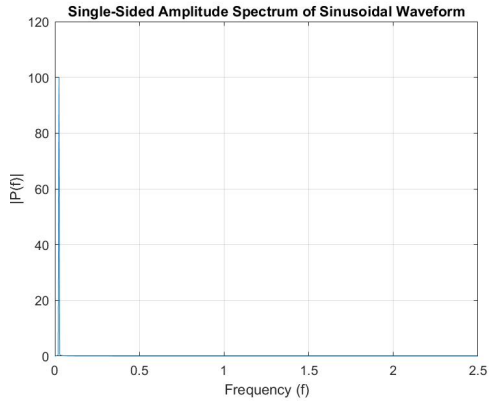
```
% Parameters
numSamples = 1024;
bitDepth = 8;
amplitude = 100;

% Time vector
t = 0:1/numSamples:1;

% Generate waveforms
sineWave = amplitude * sin(10 * pi * t);
cosWave = amplitude * sin(10 * pi * t + pi/2);

sawtoothWave = sawtooth(10 * pi * t);
sawtoothWave = (sawtoothWave + 1) * amplitude;

triangularWave = sawtooth(10 * pi * t, 0.5);
triangularWave = (triangularWave + 1) * amplitude;

pulseWave = square(10 * pi * t);
pulseWave = (pulseWave + 1) * amplitude;


% Convert to 8-bit fixed-point format
sineWave_fixed = fi(sineWave, 1,8,0);
cosWave_fixed = fi(cosWave, 1,8,0);
sawtoothWave_fixed = fi(sawtoothWave, 0,8,0);
triangularWave_fixed = fi(triangularWave, 0,8,0);
pulseWave_fixed = fi(pulseWave, 0,8,0);

% Convert to binary strings
sineWave_bin = dec2bin(sineWave_fixed);
cosWave_bin = dec2bin(cosWave_fixed);
sawtoothWave_bin = dec2bin(sawtoothWave_fixed);
triangularWave_bin = dec2bin(triangularWave_fixed);
pulseWave_bin = dec2bin(pulseWave_fixed);

% Prepend 8 zeros to make 16-bit binary strings
sineWave_bin_16 = strcat(cosWave_bin, sineWave_bin);
sawtoothWave_bin_16 = strcat('00000000', sawtoothWave_bin);
triangularWave_bin_16 = strcat('00000000', triangularWave_bin);
pulseWave_bin_16 = strcat('00000000', pulseWave_bin);

% Save to text files
writematrix(sineWave_bin_16, 'sineWave.txt');
writematrix(sawtoothWave_bin_16, 'sawtoothWave.txt');
writematrix(triangularWave_bin_16, 'triangularWave.txt');
writematrix(pulseWave_bin_16, 'pulseWave.txt');
```

Prior to analyzing the results obtained from Vivado Simulation, it is essential to verify that the waveforms generated in MATLAB meet our requirements. Additionally, as we aim to compute the FFT of these signals, it is better to examine both the waveform and its FFT in the MATLAB environment to enhance the verification process in the subsequent Verilog step.

The waveforms and their corresponding FFTs are as follows:

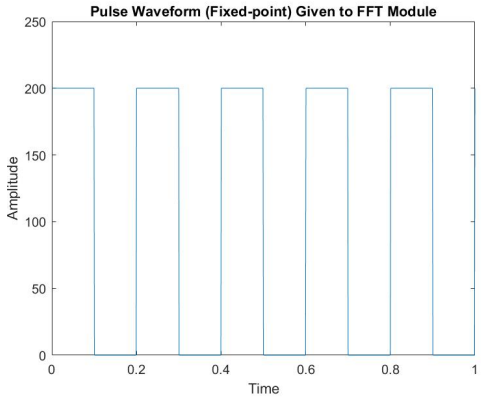


(a) Sinusoidal Waveform Generated in MATLAB

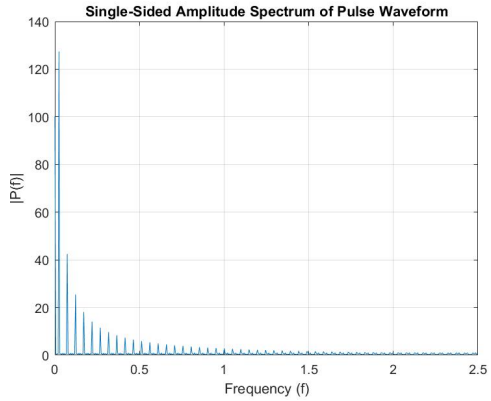


(b) FFT of Sinusoidal Waveform

Figure 2: Generating Sinusoidal Waveform in MATLAB Environment

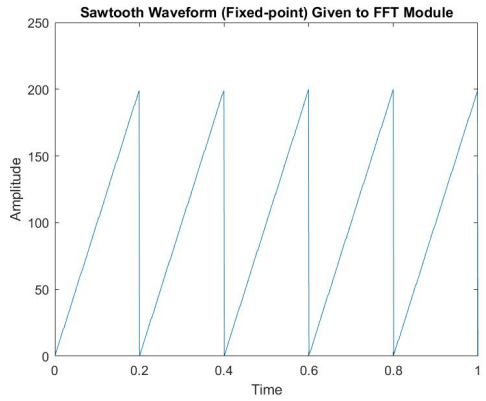


(a) Pulse Waveform Generated in MATLAB

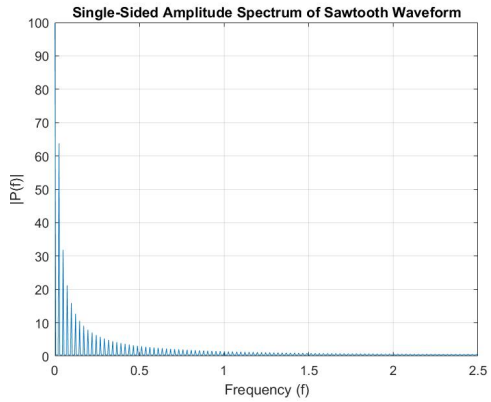


(b) FFT of Pulse Waveform

Figure 3: Generating Pulse Waveform in MATLAB Environment

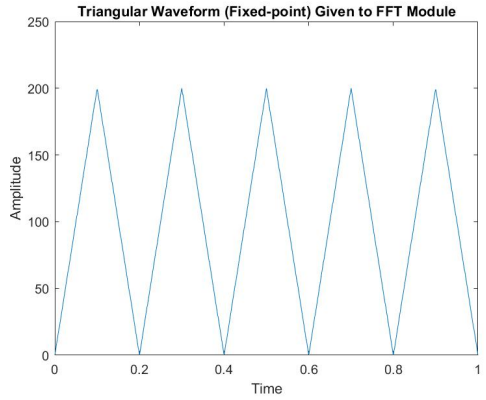


(a) Sawtooth Waveform Generated in MATLAB

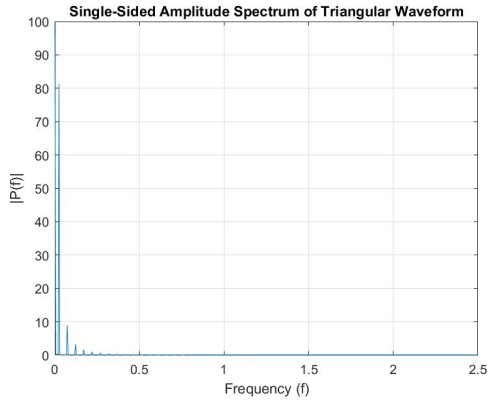


(b) FFT of Sawtooth Waveform

Figure 4: Generating Sawtooth Waveform in MATLAB Environment



(a) Triangular Waveform Generated in MATLAB



(b) FFT of Triangular Waveform

Figure 5: Generating Triangular Waveform in MATLAB Environment

As presented above, the fixed-point generated signals are as we expected, so we can make sure the delivered signal to the FFT block meets our requirements.

• **Verilog Main Module and Testbench Scripts:**
As discussed earlier, the desired waveforms are generated in MATLAB and are ready to be read in a Verilog Testbench file.

The Verilog Script for both main module (Instantiating FFT block) and Testbench are as follows:

• **Verilog Main Module Script:**

```
`timescale 1ns / 1ps

module FFT_Module(
input clk,
input [15:0] In_Data,
input [7:0] Waveform,
input In_Valid,
output In_Ready,
input In_Last,
output [8:0] Out_Data,
output Out_Valid,
input Out_Ready,
output Out_Last,
input [15:0] Config_Data,
input Config_Valid,
output Config_Ready
);

// Defining a Temporary Output port for FFT
wire [15:0] Temp_Out;

// Defining Other Optional ports of the FFT Block
wire
event_frame_started,
event_tlast_unexpected,
event_tlast_missing,
event_status_channel_halt,
event_data_in_channel_halt,
event_data_out_channel_halt;

// Instantiating FFT Block
FFT_Block your_instance_name (
.aclk(clk),
.s_axis_config_tdata(Config_Data),
.s_axis_config_tvalid(Config_Valid),
.s_axis_config_tready(Config_Ready),
.s_axis_data_tdata(In_Data),
.s_axis_data_tvalid(In_Valid),
.s_axis_data_tready(In_Ready),
.s_axis_data_tlast(In_Last),
.m_axis_data_tdata(Temp_Out),
.m_axis_data_tvalid(Out_Valid),
.m_axis_data_tready(Out_Ready),
.m_axis_data_tlast(Out_Last),
```

```
        .event_frame_started(event_frame_started),
        .event_tlast_unexpected(event_tlast_unexpected),
        .event_tlast_missing(event_tlast_missing),
        .event_status_channel_halt(event_status_channel_halt),
        .event_data_in_channel_halt(event_data_in_channel_halt),
        .event_data_out_channel_halt(event_data_out_channel_halt)
        );

        // Handling the Approximation in Finding Output
        assign Out_Data = (Temp_Out[15]&&Temp_Out[7]) ?
         -(Temp_Out[15:8]+ Temp_Out[7:0]):
        (Temp_Out[15]&&!Temp_Out[7]) ? Temp_Out[7:0]-Temp_Out[15:8]:
        (!Temp_Out[15]&&Temp_Out[7]) ? Temp_Out[15:8]-Temp_Out[7:0]:
        Temp_Out[15:8]+Temp_Out[7:0];

        endmodule
```

- **Verilog Testbench Script:**

```
        `timescale 1ns / 1ps

        module FFT_Module_TB();

        // Inputs:
        reg clk;
        reg [15:0] In_Data;
        reg [7:0] Waveform;
        reg In_Valid;
        reg In_Last;
        reg Out_Ready;
        reg [15:0] Config_Data;
        reg Config_Valid;

        // Outputs:
        wire In_Ready;
        wire [8:0] Out_Data;
        wire Out_Valid;
        wire Out_Last;
        wire Config_Ready;

        reg [15:0] File_Input [0:1023];
        reg [7:0] File_Waveform [0:1023];
        integer i;

        FFT_Module Inst1 (
        .clk(clk),
        .In_Data(In_Data),
        .Waveform(Waveform),
        .In_Valid(In_Valid),
        .In_Last(In_Last),
        .Out_Ready(Out_Ready),
        .Config_Data(Config_Data),
        .Config_Valid(Config_Valid),
        .In_Ready(In_Ready),
        .Out_Data(Out_Data),
        .Out_Valid(Out_Valid),
```

```
38          .Out_Last(Out_Last),
39          .Config_Ready(Config_Ready)
40          );
41
42          always #5 clk = ~clk;
43
44          // Initializing variables
45          initial begin
46          clk=0;
47          In_Valid=1'b0;
48          In_Data=16'd0;
49          Waveform = 8'd0;
50          In_Last=1'b0;
51          Out_Ready=1'b1;
52          Config_Data=16'd0;
53          Config_Valid=1'b0;
54          $readmemb("pulseWave.txt",File_Input);
55          $readmemb("pulse.txt",File_Waveform);
56          end
57
58          // Configuration Initial Block
59          initial begin
60          #100
61          Config_Data=1; // 1: Forward FFT
62          #5
63          Config_Valid=1;
64
65          while (Config_Ready==0) begin
66          Config_Valid=1;
67          end
68          // FFT is Configured
69          @(posedge clk) Config_Valid=0;
70          end
71
72          // Input Port Initial Block
73          initial begin
74          #100
75          for (i=0; i<1024; i=i+1) begin
76          #10
77          Waveform = File_Waveform[i];
78          end
79
80          for (i=0; i<1024; i=i+1)begin
81          #10
82
83          In_Data = File_Input[i];
84          In_Valid = 1;
85          In_Last = (i == 1023);
86
87          while (In_Ready == 0) begin
88          In_Valid = 1;
89          end
90          end
91
92          #10;
93          In_Valid = 0;
```

```
94          In_Last = 0;
95          end
96
97          // Output Port Initial Block
98          initial begin
99          #100
100         wait(Out_Last == 1'b1);
101         #300 Out_Ready = 1'b0;
102         end
103         endmodule
```

- **Mentioning some points about the Testbench Script:**
  - As you can see in the script above, there is another signal called '**waveform**' which demonstrates the first 8 bits of the input signal. According to the specifications of FFT Block, it receives both real and imaginary parts of the number in a single input, which in our case, the first 8 bits represent the real component and the last 8 bits represent the imaginary component of the number.
    For simplicity, we assumed that the imaginary component is zero, except for the sinusoidal waveform which in that case, the imaginary component is cos waveform.
  - Generated Textfiles by MATLAB store 1024 records of 16-bit data, which is loaded using '**readmemb**' command.
  - For Configuration, Input, and Output Initializing sections, Ready, Valid, and Last signals represent specific role as follows:
    - **Ready Signal:**
      It switches to high, when the process is done (FFT is ready, configuration is done, etc.)
    - **Last Signal:**
      It switches to high, when the last bit is received, transferred, etc.
    - **Valid Signal:**
      Valid signal is high, when a specific section is in progress.

• **Results obtained from Vivado Simulation:**
The following figures represent the results obtained from Vivado Simulation. It should be highlighted that there are three figures for each waveform, representing the input waveform, elicited FFT, and an overview of all signals' flow.

• **Sinusoidal Waveform:**

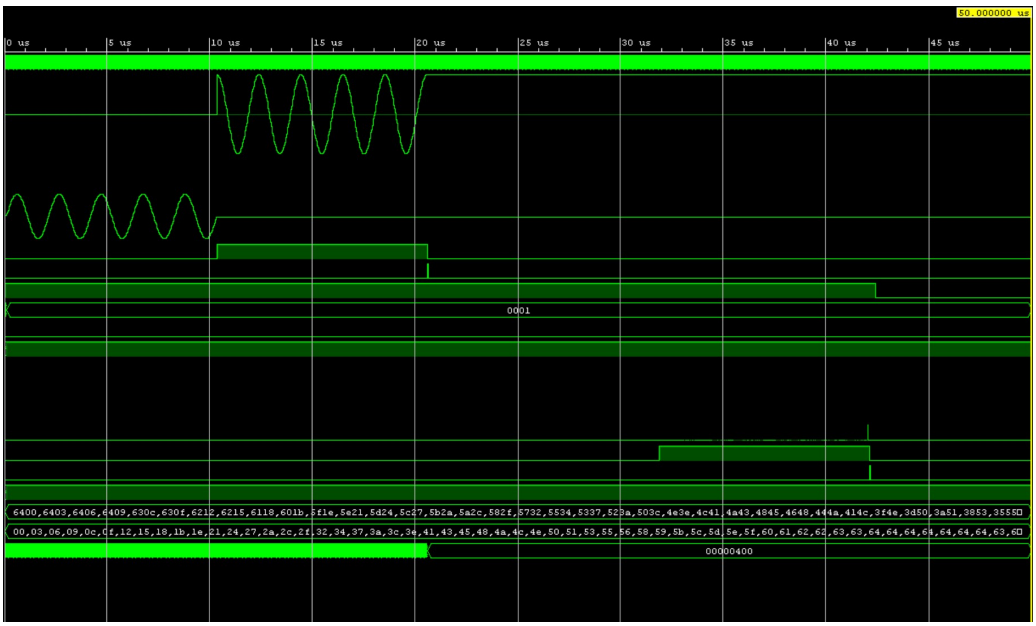


Figure 6: Representation of all signals, running for $50\mu s$

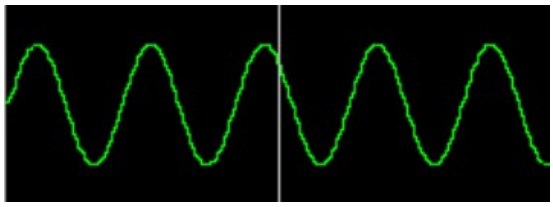


Figure 7: Representation of Sinusoidal Waveform given to the FFT Block

Figure 8: Output signal passed through FFT Block

- **Pulse Waveform:**

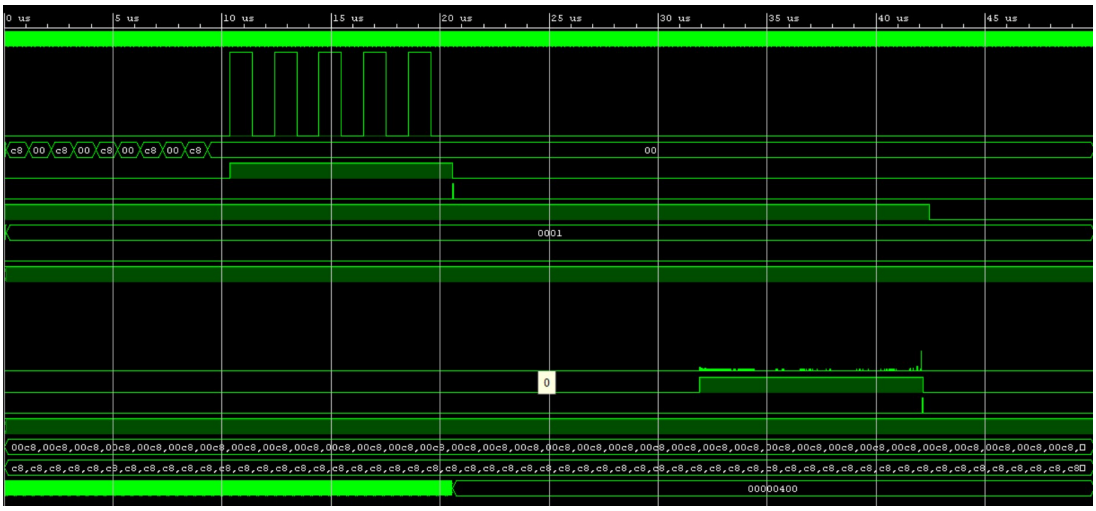


Figure 9: Representation of all signals, running for $50\mu s$

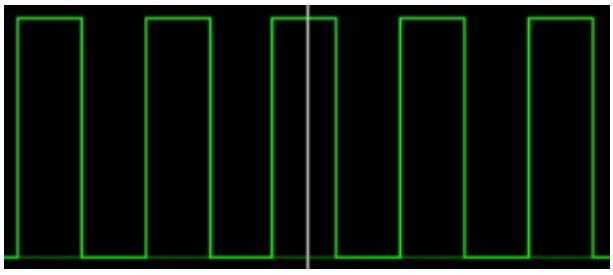


Figure 10: Representation of Pulse Waveform given to the FFT Block

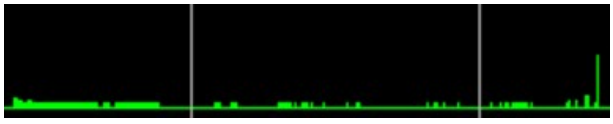


Figure 11: Output signal passed through FFT Block

- **Sawtooth Waveform:**

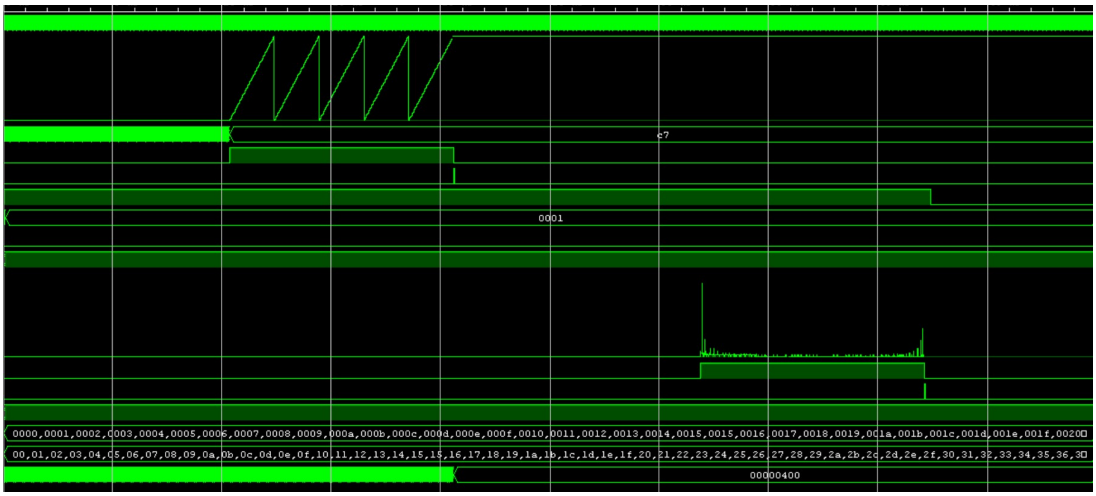


Figure 12: Representation of all signals, running for $50\mu s$

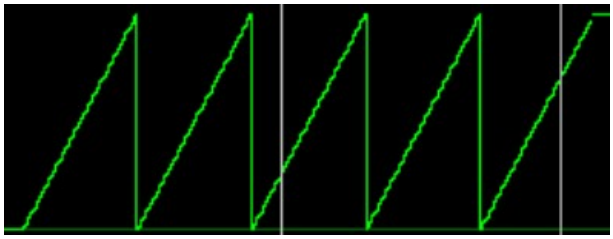


Figure 13: Representation of Sawtooth Waveform given to the FFT Block

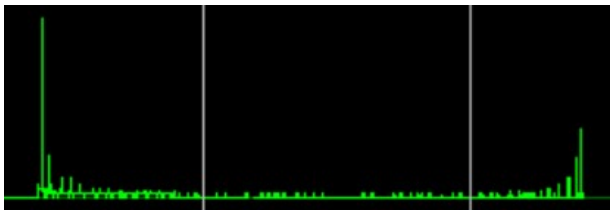


Figure 14: Output signal passed through FFT Block

- **Triangular Waveform:**

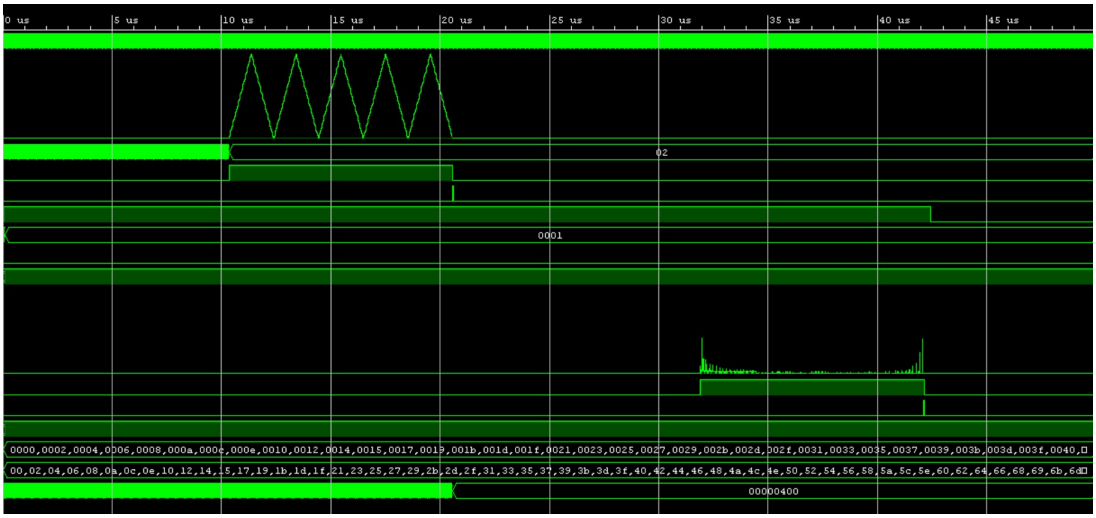


Figure 15: Representation of all signals, running for $50\mu s$

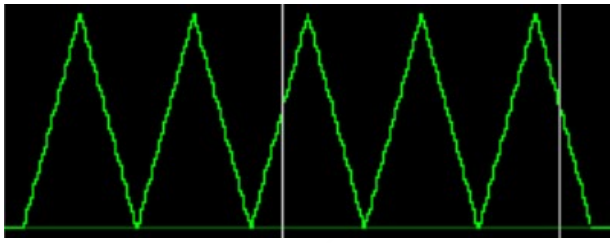


Figure 16: Representation of Triangular Waveform given to the FFT Block

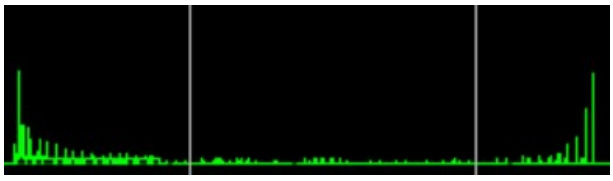


Figure 17: Output signal passed through FFT Block

5 | Conclusion:

In this project, we aimed to examine the FFT IP Core and tried to demonstrate a signal with its FFT on a monitor using ZYNQ Boards. Moreover, in the simulation section, we successfully implemented and tested the FFT IP Core in Vivado, utilizing MATLAB to generate the input data. The primary objective was to compare the results of the FFT operation performed by Vivado against those generated in MATLAB. Our analysis included a variety of waveforms: sinusoidal, pulse, sawtooth, and triangular.

- **Key Findings During Phase 1:**

  - **FFT Accuracy:**
    The FFT results obtained from Vivado's FFT IP Core were largely consistent with those generated by MATLAB. This demonstrates the accuracy and reliability of the Vivado FFT IP Core for signal processing tasks. Minor differences observed are within acceptable limits and can be attributed to variations in implementation specifics and numerical precision between the two platforms.

  - **Integration Challenges:**
    Several integration challenges were encountered and resolved during the project. Notably, configuring the FFT block correctly and handling the data width conversion between different components required careful attention. These steps were crucial to ensure that the FFT block received and processed the correct data formats.

  - **Performance and Functionality:**
    The project verified that the FFT block in Vivado performs as expected when integrated into a larger system. The design successfully passed both the initial and the FFT-processed signals through to the display, confirming that the end-to-end system functioned correctly.