Experiment 3 - Function Generator

Digital Logic Design Lab - Fall 2022

Erfan Panahi
ID: 810198369

Sogol Goodarzi
ID: 810198467

FUNCTION GENERATOR

INTRODUCTION

In this experiment, we are to design an Arbitrary Generator that is capable of generating each of the aforementioned waveforms with wide range for frequency selection.

I. WAVEFORM GENERATOR

Question 1. Write the Verilog code for the Waveform Generator Processor in (figure 4 - "DLDLAB_3_Fall_1401.pdf") and use LPM or Megawizard functions for ROM memory, Multiplexer and other required gates.You will finally get all the components together in a schematic design. For your total design consider a 10-bit input named SW that recalls the switch inputs in FPGA. Dedicate 3 bits of this 10-bit input to the selectors of the waveform generator (SW[9:7]). The remaining bits will be used for the frequency and amplitude selectors.
The Verilog code for the waveform generator, Multiplexer, Register and adder is shown in figure 1-4. With the use of Quartus, the implementation of ROM is done and then we uploaded the Verilog codes of the mentioned blocks in Quartus and do the wirings between these blocks and create DDS block diagram which is shown in figure 5.

Question 2. After completing the waveform generator block diagram in Quartus, synthesize the final design and include the synthesis summary in your report.
After creating the symbol for the waveform generator, we create the block diagram shown in figure 6. By synthesizing the final design the synthesis summary will be like figure 7.

Question 3. Before adding the frequency and amplitude selector, it is better to test your design in Modelsim. Therefore, write a simple testbench and verify the functionality of your design. For this part use the 50-MHz clock frequency. Include the Modelsim results for all the waveforms mentioned in (Table 1. - "DLDLAB_3_Fall_1401.pdf") Set the value of Phase_cntrl to 1 for this part.
The written test bench for testing the block of the waveform generator is illustrated in figure 8. In the test bench we change the 'func' bits shown in figure and the waveform results are in figure 10.

```
`timescale 1ns/1ns
module WaveformGenerator(input clk, rst, input[2:0] func, output reg[7:0] Wave);
  wire [7:0] Reciprocal, Square, Triangle;
  reg  signed [15:0] sin, cos;
  reg [7:0] counter;
  wire co;

  always @(posedge clk, posedge rst) begin
        if (rst)
          counter <= 8'd0;
        else if (co)
          counter <= 8'd0;
    else
          counter <= counter + 1;
    end
    assign co = &counter;
    assign Reciprocal = 8'd255 / (9'd256 - counter);
    assign Square = (counter <= 8'd128)? 8'd255 : 8'b0;
    assign Triangle = counter < 8'd128 ? (2*counter) : 8'd255 - (2*counter);

    always @(posedge clk, posedge rst) begin
       if (rst) begin
        sin <= 16'd0;
        cos <= 16'd30000;
        end
       else begin
        sin = sin + {{6{cos[15]}},cos[15:6]};
        cos = cos - {{6{sin[15]}},sin[15:6]};
        end
    end

    always @(posedge clk, posedge rst) begin
        case(func)
          // Reciprocal
          3'b000: Wave = Reciprocal;
          // Square
          3'b001: Wave = Square;
          // Triangle
          3'b010: Wave = Triangle;
          // Sine
          3'b011: Wave = sin[15:8] + 8'd127;
          // Full-wave rectified
          3'b100: Wave = sin[15] ? -sin[15:8] + 8'd127 : sin[15:8] + 8'd127;
          //Half-wave rectified
          3'b101: Wave = sin[15] ? 8'd127 : sin[15:8] + 8'd127;
          default: Wave = 8'd128;
        endcase
    end
endmodule
```

Fig. 1. Verilog code for the "Waveform Generator"

```
`timescale 1ns/1ns
module Multiplexer (input [7:0] A,B, input sel, output [7:0] Y);
    assign Y = sel ? A : B;
endmodule
```

Fig. 2. Verilog code for the "Multiplexer"

```
`timescale 1ns/1ns
module Register (input clk, rst, input [7:0] Y, output reg [7:0] B);
    always@(posedge clk, posedge rst)
        if(rst)
          B <= 8'd0;
        else
          B <= Y;
endmodule
```

Fig. 3. Verilog code for the "Register"

```
`timescale 1ns/1ns
module Adder (input [1:0] A, input [7:0] B, output [7:0] Y);
    assign Y = {6'd0,A} + B;
endmodule
```

Fig. 4. Verilog code for the "Adder"

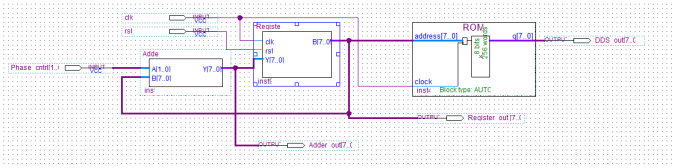

Fig. 5. DDS block diagram

II. Digital to Analog Conversion using PWM

Question 1. Write a Verilog code for DAC module. The output of the PWM module will go to an external board with an RC low pass filter.

The Verilog code for the DAC module is shown in figure 11. As it is mentioned in Laboratory manual file, The output of the PWM module will go to an external board with an RC low pass filter.

Question 2. You can use a random data input from the switches to test the accuracy of your design.

This part is done when we finished creationg block diagrams and write all Verilog codes and then we program the FPGA board and test the accuracy of the design.

Question 3. Connect the PWM module to the design in the previous section, synthesize the design and program the FPGA.

The block diagram of waveform after connecting PWM module is shown in figure 12. We synthesis the design and program the FPGA and then we connect the output of PWM module of FPGA to an external board with an RC low pass filter.

Question 4. Observe the functions on the Oscilloscope and include them in your reports.

By changing the 'func' bits, all the mentioned functions can

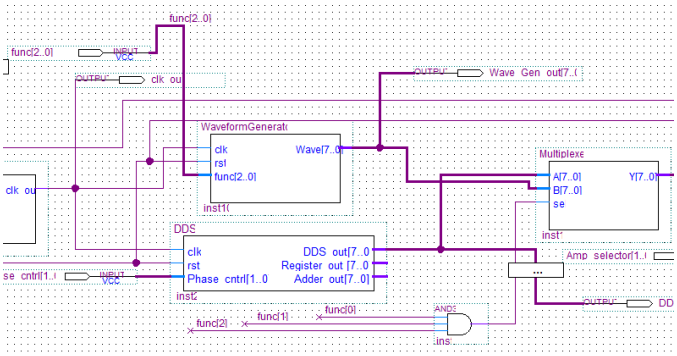

Fig. 6. Waveform generator block diagram

	Task	Time
✓	Compile Design	00:00:12
✓	Analysis & Synthesis	00:00:03
✓	Fitter (Place & Route)	00:00:05
✓	Assembler (Generate programming files)	00:00:01
✓	TimeQuest Timing Analysis	00:00:02
✓	EDA Netlist Writer	00:00:01
	Program Device (Open Programmer)	

Fig. 7. Synthesis summary of waveform generator block diagram

```
`timescale 1ns/1ns
module Waveform_TB();
  wire [7:0] Wave;
  reg [2:0] func;
  reg rst;
  reg clk = 0;

  WaveformGenerator WG(.clk(clk), .rst(rst),
                       .func(func), .Wave(Wave));

  initial
    begin
      #1 rst = 1'b1;
      #1 rst = 1'b0;
      #10 func = 3'b000;
      #1 rst = 1'b1;
      #1 rst = 1'b0;
      #5000 func = 3'b001;
      #1 rst = 1'b1;
      #1 rst = 1'b0;
      #5000 func = 3'b010;
      #1 rst = 1'b1;
      #1 rst = 1'b0;
      #5000 func = 3'b011;
      #1 rst = 1'b1;
      #1 rst = 1'b0;
      #5000 func = 3'b100;
      #1 rst = 1'b1;
      #1 rst = 1'b0;
      #5000 func = 3'b101;
      #10000 $stop;
  end

  always
    begin
      #1 clk = ~clk;
  end

endmodule
```

Fig. 8. Waveform generator testbench

func[2:0]	Function
3'b000	Reciprocal
3'b001	Square
3'b010	Triangle
3'b011	Sine
3'b100	Full-wave rectified
3'b101	Half-wave rectified
3'b110	DDS

Fig. 9. Function Selection

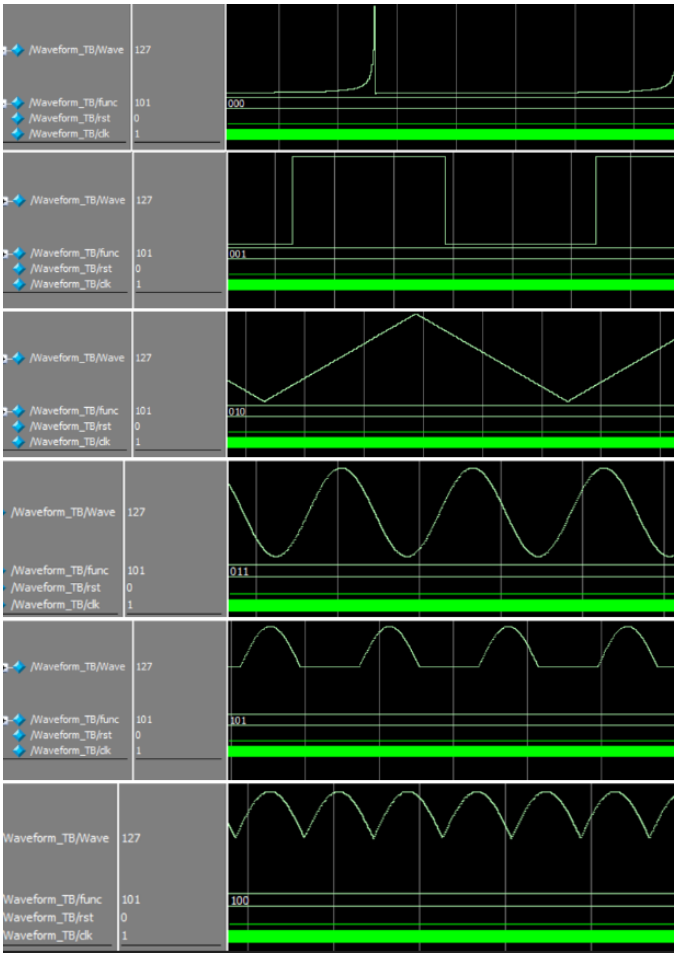

Fig. 10. Output waveforms of waveform generator block diagram

```
`timescale 1ns/1ns
module Digital_To_Analog(input clk, rst,
                         input[7:0] X, output Y);
  reg [7:0] counter;
  wire co;

  always @(posedge clk, posedge rst) begin
        if (rst)
          counter <= 8'd0;
        else if (co)
          counter <= 8'd0;
        else
          counter <= counter + 1;
  end

  assign co = &counter;
  assign Y  = (counter <= X)? 1'b1 : 1'b0;

endmodule
```

Fig. 11. The verilog code for DAC module

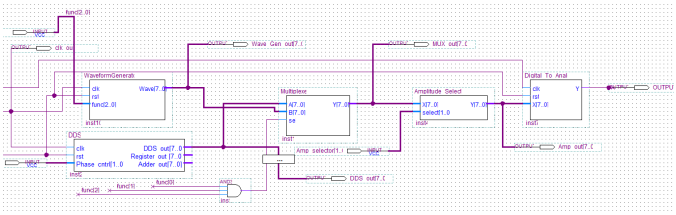

Fig. 12. The block diagram of waveform after connecting PWM module

be seen on the Oscilloscope and they are shown in figure 13-18.

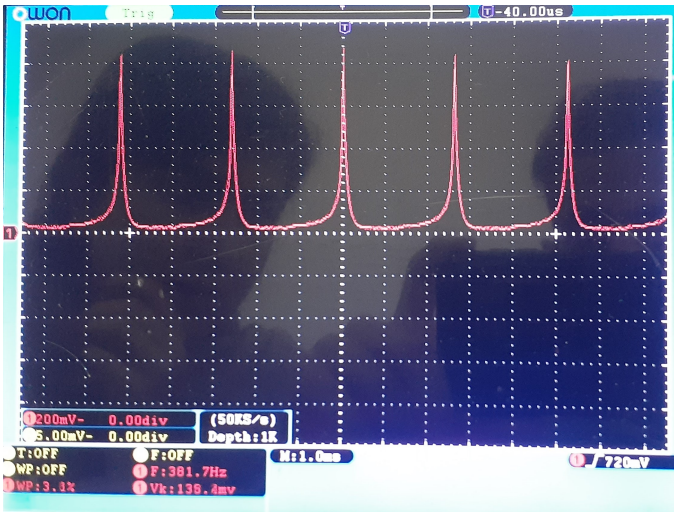

Fig. 13. The reciprocal waveform (func = 000)

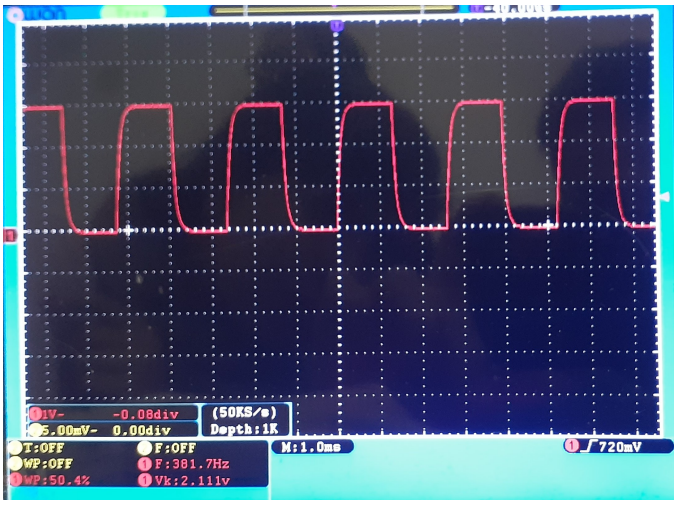

Fig. 14. The square waveform (func = 001)

III. Frequency Selector

Question 1. After adding the clock divider to the waveform generator design, synthesize your design.

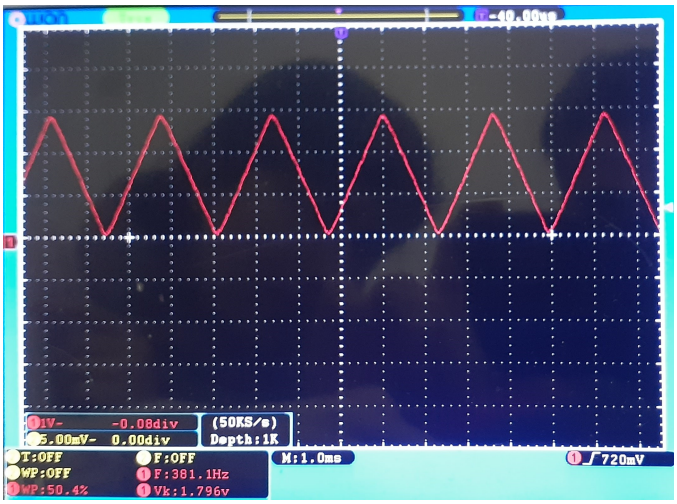

Fig. 15. The triangle waveform (func = 010)

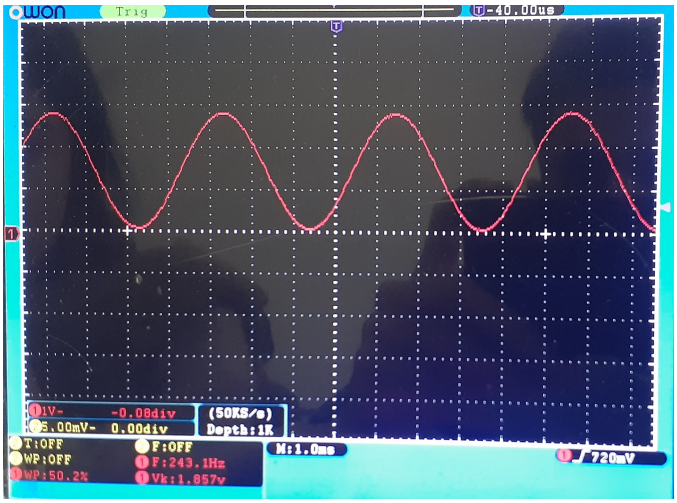

Fig. 16. The sine waveform (func = 011)

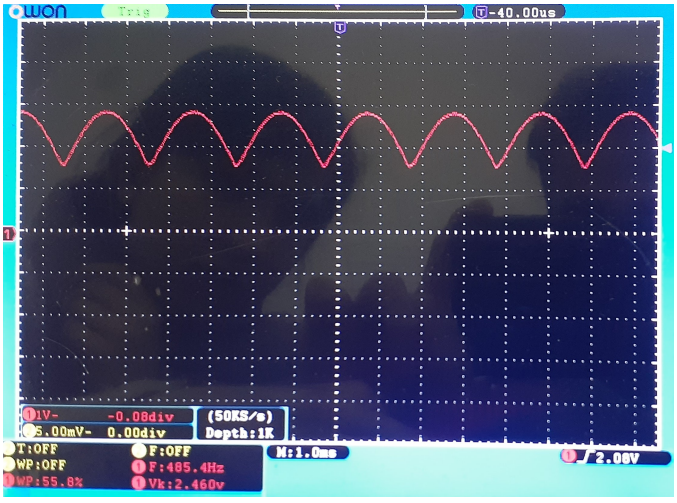

Fig. 17. The full-wave rectified waveform (func = 100)

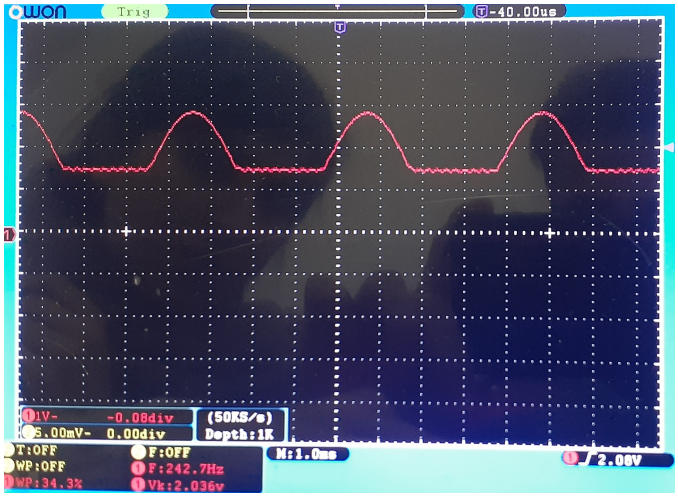

Fig. 18. The half-wave rectified waveform (func = 101)

The block diagram of function generator after adding clock divider (frequency selector) is shown in figure 19.

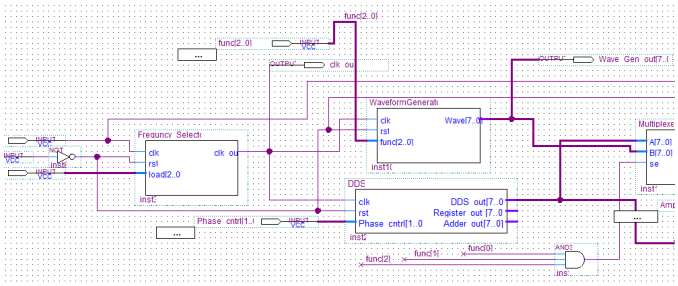

Fig. 19. The block diagram of function generator after adding clock divider (frequency selector)

Question 2. Set the 4 least significant bits of the counter to a fixed value in your code and set the remaining 5 bits as the input pins of the counter. Use SW[4:0] for this parallel loads of the divider.

We set the 6 least significant bits of the counter to a fixed value (6'd0). Then we set the remaining 3 bits as the input pins of the counter. The Verilog code for frequency selector is shown in figure 20.

Question 3. Modify the testbench of section one so that you can verify the design for different desired frequency. To do this, change the parallel loads (SW[4:0]) for at least three different frequencies.

The written testbench for frequency selector is shown in figure 21.

Question 4. Include the waveforms of each desired frequency in your report with mentioning the achieved frequencies, the input parallel loads and expected frequencies.

As it is mentioned in question 2., we set the 3 most bits with the values of 000, 001, 010, and 011. The results are shown

```
`timescale 1ns/1ns
module Frequncy_Selector(input clk, rst,
                         input [2:0] load, output clk_out);
  reg [8:0] cnt;

  assign clk_out = &cnt;

    always @(posedge clk, posedge rst) begin
      if (rst)
         cnt <= 9'd0;
      else if (clk_out)
        cnt <= {load, 6'b000000};
      else
          cnt <= cnt + 1;
    end

endmodule
```

Fig. 20. The Verilog code for frequency selector

```
`timescale 1ns/1ns
module FS_TB();
  wire  clk_out;
  reg [2:0] load = 3'd0;
  reg rst;
  reg clk = 0;

  Frequncy_Selector fs(clk, rst, load, clk_out);

  initial
    begin
      #1 rst = 1'b1;
      #1 rst = 1'b0;
      #5 load = 3'b000;
      #50000 load = 3'b001;
      #50000 load = 3'b010;
      #50000 load = 3'b011;
      #50000 $stop;
  end

  always
    begin
      #10 clk = ~clk;
  end

endmodule
```

Fig. 21. The testbench for frequency selector

in figure 22.

Fig. 22. The waveform for testbench of frequency selector

Now we set the 3 most bits with the values of 000, 001, and 010. The waveforms can be seen on the Oscilloscope and they are shown in figure 23-25.

Question 5. As mentioned the DDS module can generate sine wave with different phases based on the Phase_cntrl input. As a different way of frequency selection verify your design

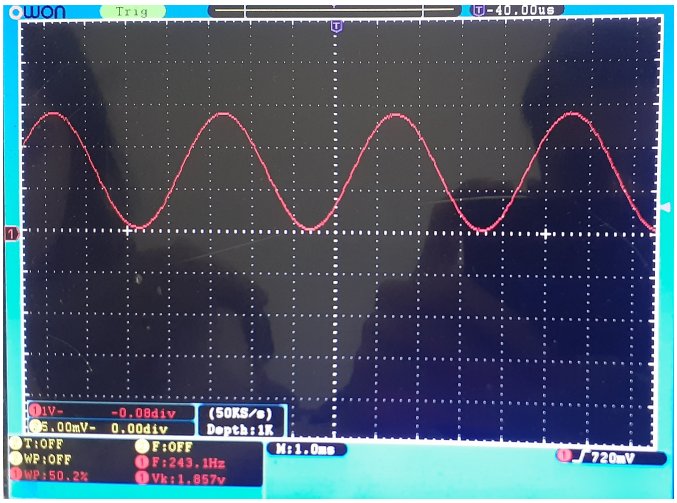

Fig. 23. The sine waveform (freq_sel = 000)

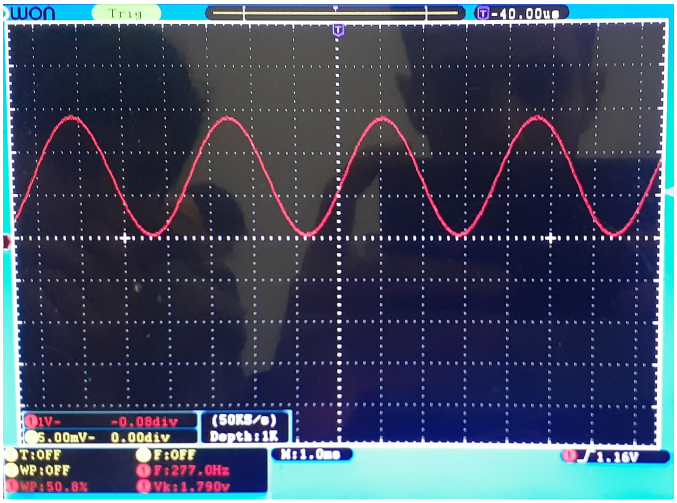

Fig. 24. The sine waveform (freq_sel = 001)

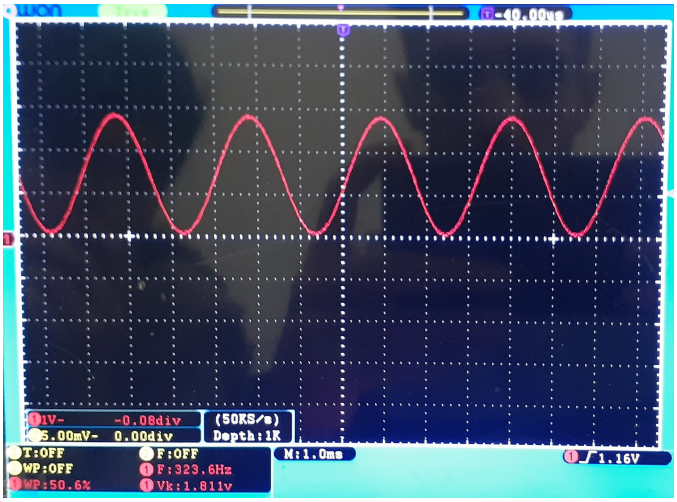

Fig. 25. The sine waveform (freq_sel = 010)

for at least 3 values of Phase_cntrl and observe the frequency change in the waveforms. Include the waveforms in your report and explain about the expected and achieved frequency.
We set the Phase_ctrl with the values of 00, 01, 10 and 11 as it is shown testbench figure 26. The DDS output waveforms with different Phase_ctrl can be seen on the Oscilloscope and they are shown in figure 27-29.

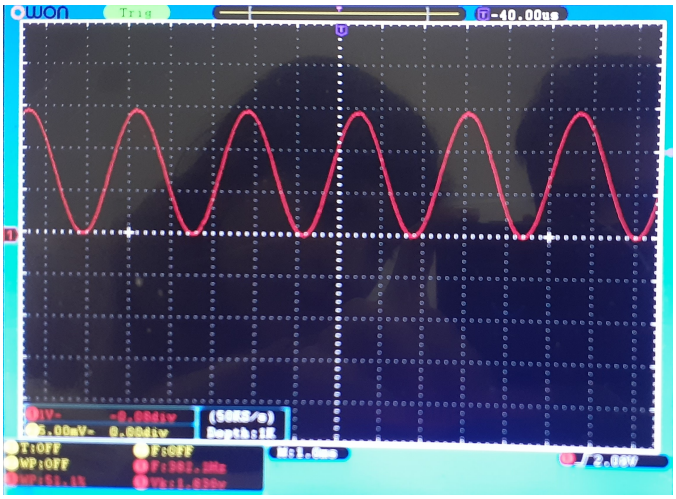

Fig. 26. The DDS output waveform (Phase_ctrl = 000)

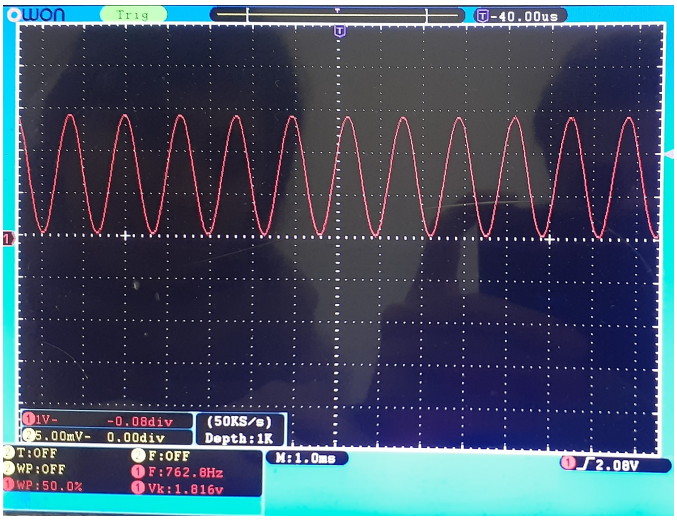

Fig. 27. The DDS output waveform (Phase_ctrl = 001)

IV. Amplitude Selector

Question 1. After adding the amplitude selector module to the previous design, synthesize your design.
The written amplitude selector module is shown in figure 29 and the block diagram of function generator after adding amplitude selector is shown in figure 30.

Question 2. Modify the testbench of section two so that you can verify the design for different amplitude. To do this,

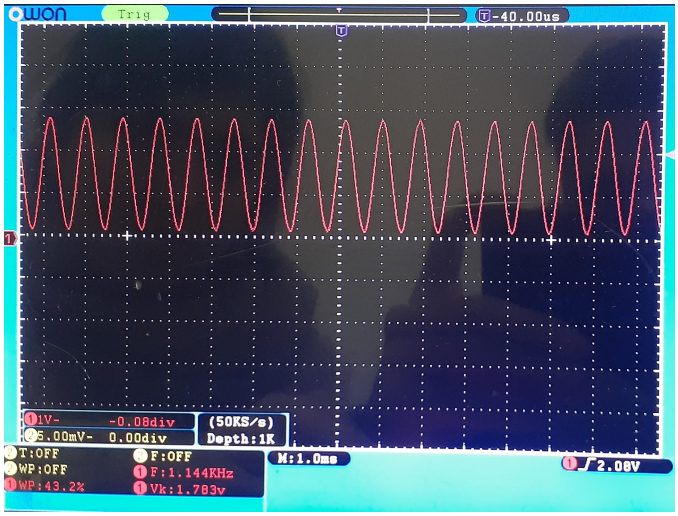

Fig. 28. The DDS output waveform (Phase_ctrl = 010)

```
module Amplitude_Selector(input [7:0] X, input [1:0] select,
                          output reg[7:0] Y);
  always @(X)begin
    case(select)
        2'b00: Y <= X;
        2'b01: Y <= {X[6:0],1'b0};
        2'b10: Y <= {X[5:0],2'b0};
        2'b11: Y <= {X[4:0],3'b0};
        endcase
   end
endmodule
```

Fig. 29. The amplitude selector Verilog code

change (SW[6:5])in your testbench.
The final testbench for function generator that is synthesised using Quartus is shown in figure 31.
Question 3. Include the waveforms of each desired amplitude in your report.
Now we change the value of amplitude selector like testbench of figure 31. The results are shown in figure 32-35.

V. The Total Design

The final block diagram for function generator that is synthesised with Queartus and uploaded on FPGA board is

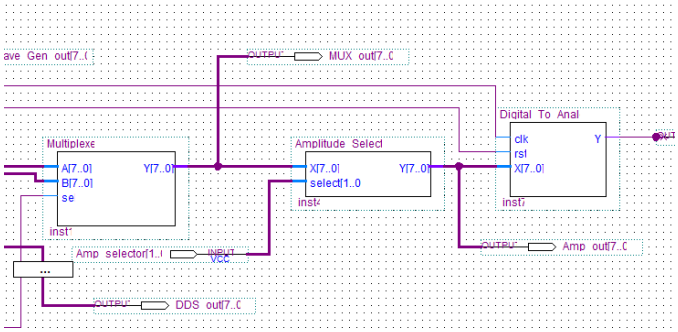

Fig. 30. The amplitude selector block diagram

```
`timescale 1ns/1ns
module Lab3_TB();

  reg rst;
  reg clk = 0;
  wire  clk_out;
  reg   [2:0] Load = 3'd0;
  wire  [7:0] Amp_out;
  reg   [1:0] Amp_selector = 2'd0;
  reg   [2:0] func;
  reg   [1:0] Phase_cntrl = 2'd0;
  wire  [7:0] DDS_out;
  wire  [7:0] MUX_out;
  wire  OUTPUT ;
  wire  [7:0] Wave_Gen_out;

  Lab3 lab3(.clk(clk), .rst(rst), .Phase_cntrl(Phase_cntrl),
            .Load(Load), .Amp_selector(Amp_selector), .func(func),
            .clk_out(clk_out), .Amp_out(Amp_out), .DDS_out(DDS_out),
            .MUX_out(MUX_out), .OUTPUT(OUTPUT), .Wave_Gen_out(Wave_Gen_out));


  initial
    begin
      #1 rst = 1'b1;
      #21 rst = 1'b0;
      #5 func = 3'b000;
      #2000000 Amp_selector = 2'd0;
      #2000000 Amp_selector = 2'd1;
      #2000000 Amp_selector = 2'd2;
      #2000000 Amp_selector = 2'd3;
      #5000000 func = 3'b001;
      #2000000 Amp_selector = 2'd0;
      #2000000 Amp_selector = 2'd1;
      #2000000 Amp_selector = 2'd2;
      #2000000 Amp_selector = 2'd3;
      #5000000 func = 3'b010;
      #2000000 Amp_selector = 2'd0;
      #2000000 Amp_selector = 2'd1;
      #2000000 Amp_selector = 2'd2;
      #2000000 Amp_selector = 2'd3;
      #5000000 func = 3'b011;
      #2000000 Amp_selector = 2'd0;
      #2000000 Amp_selector = 2'd1;
```

Fig. 31. The final testbench for function generator

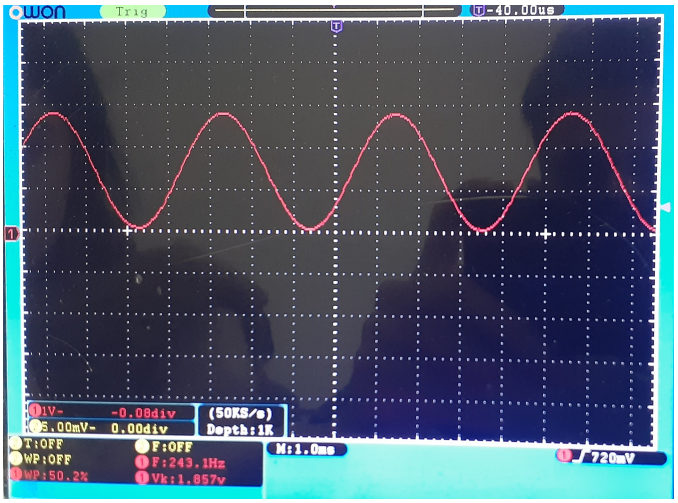

Fig. 32. The sine waveform (amp_sel = 00)

shown in figure 36.

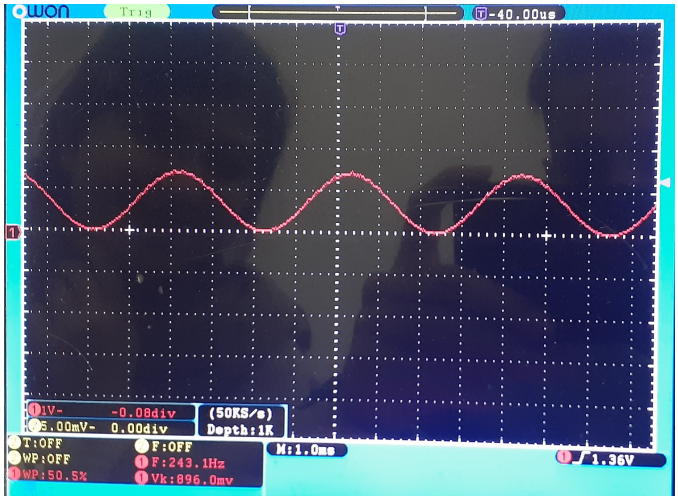

Fig. 33. The sine waveform (amp_sel = 01)

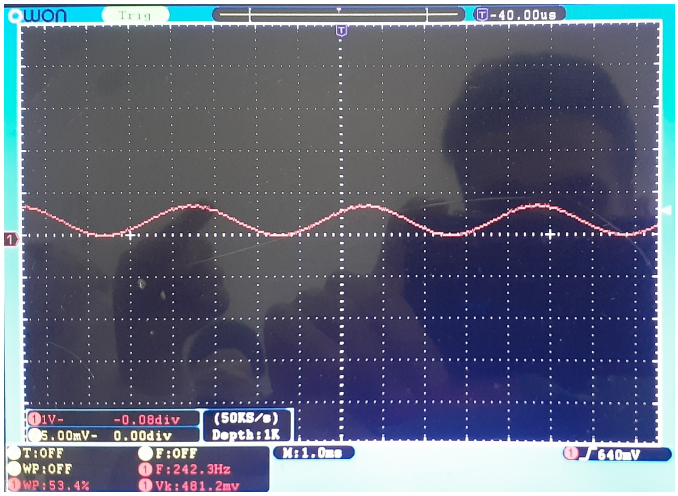

Fig. 34. The sine waveform (amp_sel = 10)

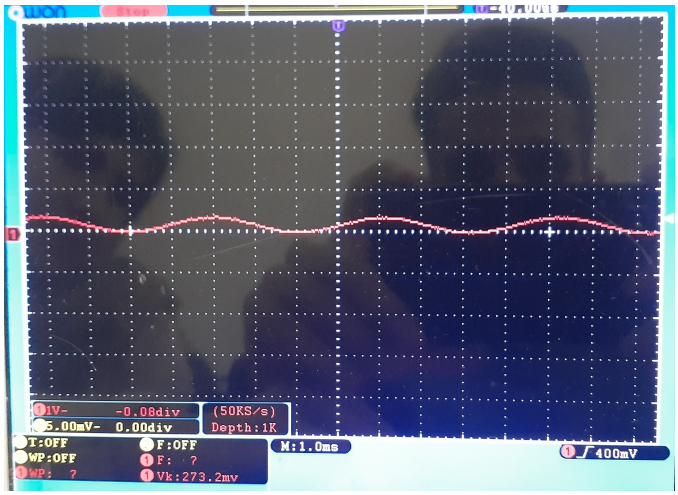

Fig. 35. The sine waveform (amp_sel = 11)

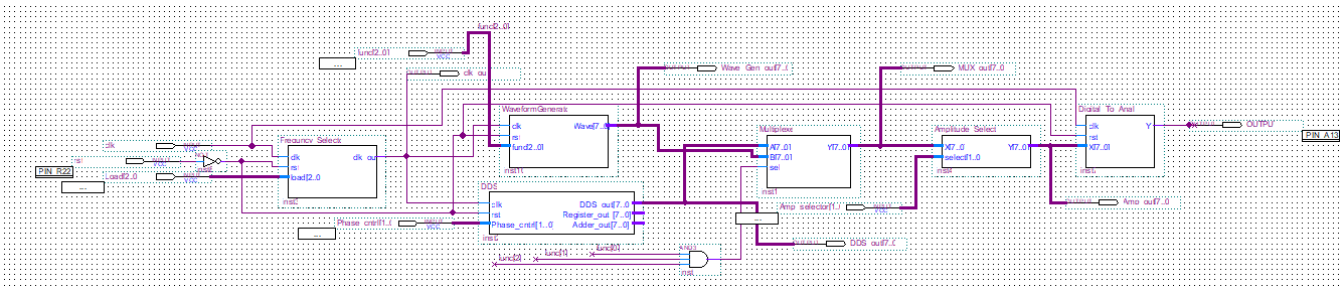

Fig. 36. The final block diagram of function generator