Experiment #3 - Function Generator

Mohammad Amin Alinejad- 810199463 / Amirali Shahriary- 810100173

***Abstract*—In this experiment, we are going to design an Arbitrary function generator (AFG) that is capable of generating each of the aforementioned waveforms with wide range for frequency selection. Create different waves, frequencies, and amplifier, and then use Quartus and upload it on FPGA.**

***Keywords*— Waveform Generator, Frequency, ROM, Selector, Amplitude Selector, PWM, DDS, Counter, Function Generator, Arbitrary Function Generator**

I. Introduction

In this document, we are going to design an Arbitrary Generator that consists of awaveform generator, Amplitidude selector, DAC and frequency selector. An Arbitrary Function Generator (AFG) is an electronic test instrument that generates a wide variety of waveforms with different amplitude and frequency. We can use AFGs to simply generate a series of basic test signals, replicate real-world signals, or create signals that are not otherwise available. These signals can then be used to learn more about how a circuit works, to characterize an electronic component, and to verify electronic theories.

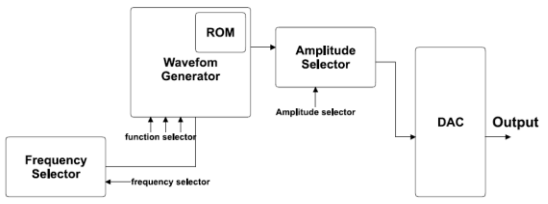


Fig. 1 Block diagram of the Arbitrary Generator (AFG)

II. Waveform Generator

In Waveform generator we designed 6 different weveforms and we use a ROM for showing another waveform that we've got before. Output of this module is an 8-bit digital representing the amplitude of signal.
In this part we have to desing this waves :
Sine, square, reciprocal, triangle, full-wave and half-wave rectified signals.
For generating Square, Triangle, and Reciprocal we use a 8-bit counter inside the wave generator ,and for the next three waves, at first we need to know how to create sin and after that we create full and half wave rectified.
And for the next three waves, at first we need to know how to create sin and after that we create full and half waverectified.

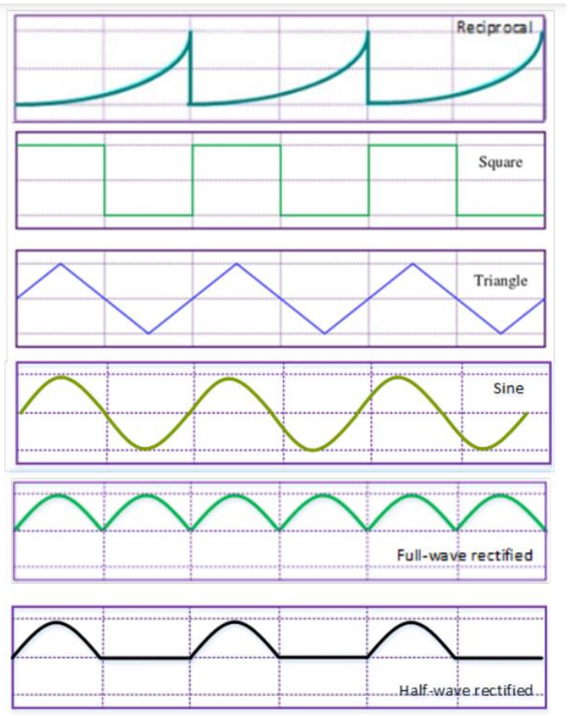
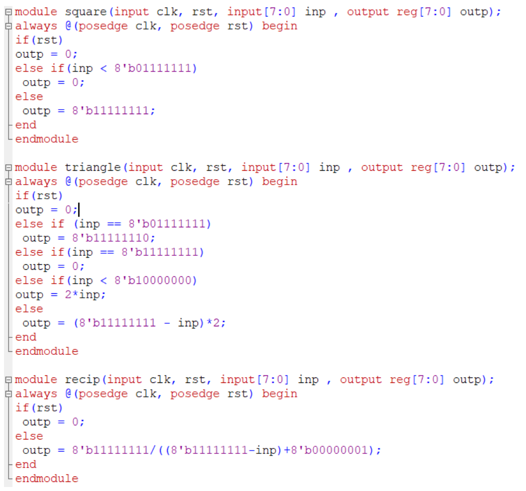


Fig. 2 wave forem generator.

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'b111	DDS

Table.1 function selection.

Here is the Verilog description for square,triangle&recip waves:



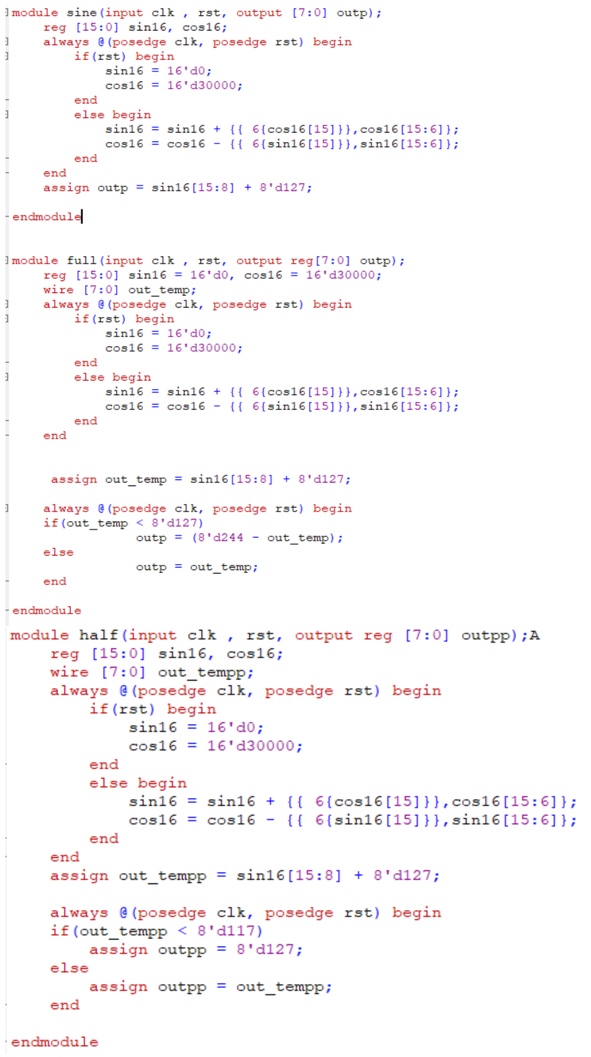
```
module square(input clk, rst, input[7:0] inp , output reg[7:0] outp);
always @(posedge clk, posedge rst) begin
if(rst)
outp = 0;
else if(inp < 8'b01111111)
 outp = 0;
else
 outp = 8'b11111111;
end
endmodule

module triangle(input clk, rst, input[7:0] inp , output reg[7:0] outp);
always @(posedge clk, posedge rst) begin
if(rst)
outp = 0;
else if  (inp == 8'b01111111)
 outp = 8'b11111110;
else if(inp == 8'b11111111)
 outp = 0;
else if(inp < 8'b10000000)
outp = 2*inp;
else
 outp = (8'b11111111 - inp)*2;
end
endmodule

module recip(input clk, rst, input[7:0] inp , output reg[7:0] outp);
always @(posedge clk, posedge rst) begin
if(rst)
 outp = 0;
else
 outp = 8'b11111111/((8'b11111111-inp)+8'b00000001);
end
endmodule
```

Fig. 3 square,triangle&recip Verilog description.

Here is the Verilog description for sine, full&half waves :



```
module sine(input clk , rst, output [7:0] outp);
    reg [15:0] sin16, cos16;
    always @(posedge clk, posedge rst) begin
        if(rst) begin
            sin16 = 16'd0;
            cos16 = 16'd30000;
        end
        else begin
            sin16 = sin16 + {{ 6{cos16[15]}},cos16[15:6]};
            cos16 = cos16 - {{ 6{sin16[15]}},sin16[15:6]};
        end
    end
    assign outp = sin16[15:8] + 8'd127;

endmodule


module full(input clk , rst, output reg[7:0] outp);
    reg [15:0] sin16 = 16'd0, cos16 = 16'd30000;
    wire [7:0] out_temp;
    always @(posedge clk, posedge rst) begin
        if(rst) begin
            sin16 = 16'd0;
            cos16 = 16'd30000;
        end
        else begin
            sin16 = sin16 + {{ 6{cos16[15]}},cos16[15:6]};
            cos16 = cos16 - {{ 6{sin16[15]}},sin16[15:6]};
        end
    end


     assign out_temp = sin16[15:8] + 8'd127;

    always @(posedge clk, posedge rst) begin
    if(out_temp < 8'd127)
                outp = (8'd244 - out_temp);
    else
                outp = out_temp;
    end

endmodule
module half(input clk , rst, output reg [7:0] outpp);A
    reg [15:0] sin16, cos16;
    wire [7:0] out_tempp;
    always @(posedge clk, posedge rst) begin
        if(rst) begin
            sin16 = 16'd0;
            cos16 = 16'd30000;
        end
        else begin
            sin16 = sin16 + {{ 6{cos16[15]}},cos16[15:6]};
            cos16 = cos16 - {{ 6{sin16[15]}},sin16[15:6]};
        end
    end
    assign out_tempp = sin16[15:8] + 8'd127;

    always @(posedge clk, posedge rst) begin
    if(out_tempp < 8'd117)
        assign outpp = 8'd127;
    else
        assign outpp = out_tempp;
    end

endmodule
```

Fig. 4 sine, full&half waves description.

For generating sin we use 2nd ODE formula and waveform clock as you can see:

$$\sin(n) = \sin(n-1) + \frac{1}{64}\cos(n)$$

$$\cos(n) = \cos(n-1) + \frac{1}{64}\sin(n)$$

$$Initial\ Value\colon \sin(0) = 0\ \&\ \cos(0) = 3000$$

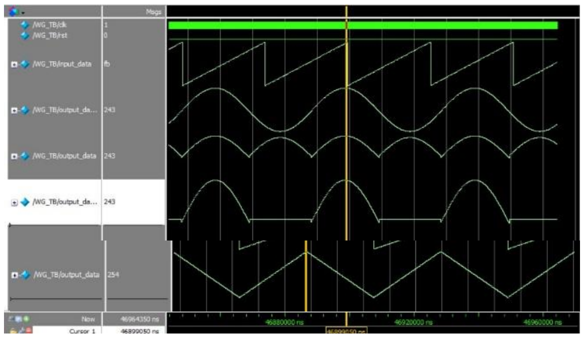


Fig. 5 wave forms.

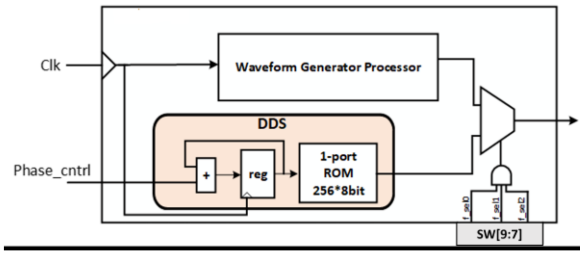


Fig. 6 Block diagram of waveform generator

Another way to generate wave is Direct Digital Synthesis (DDS) in this method we use adder , reg and ROM that used saved data and we can change its period with phase cntrl.

III. PWM

In this part we need a digital to analog conversion (DAC) and one of the cheapest methode is Pulse Width Modulation that we use and it use 8-bit counter inside it.

- When the input signal is larger than counter the output is **1.**
- When the input signal is less than counter the output is **0.**

$$duty\ cycle = \frac{T_{on}}{T_{on} + T_{off}}$$

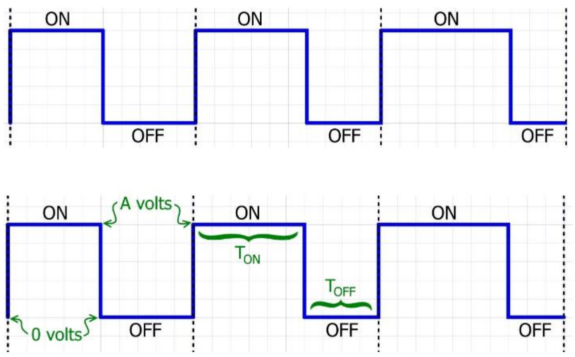
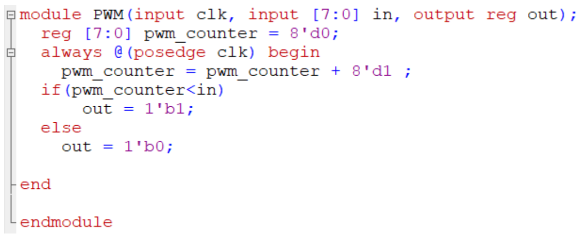


Fig. 7 Pulse Width Modulation (PWM)

Here is the Verilog description of the PWM:



```
module PWM(input clk, input [7:0] in, output reg out);
  reg [7:0] pwm_counter = 8'd0;
  always @(posedge clk) begin
    pwm_counter = pwm_counter + 8'd1 ;
  if(pwm_counter<in)
      out = 1'b1;
  else
    out = 1'b0;

end

endmodule
```

Fig. 8 PWM Verilog description.

IV. Frequency Selector

Arbitrary Generator usually have frequency selector to change its wave frequency according to the need of user.

In this part we use 9-bit counter and we set the first three bit for the change frequency and other bits our define before.

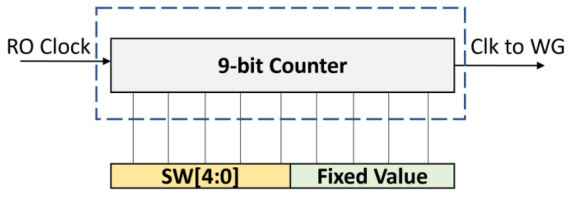


Fig.9 Block diagram of frequency selector

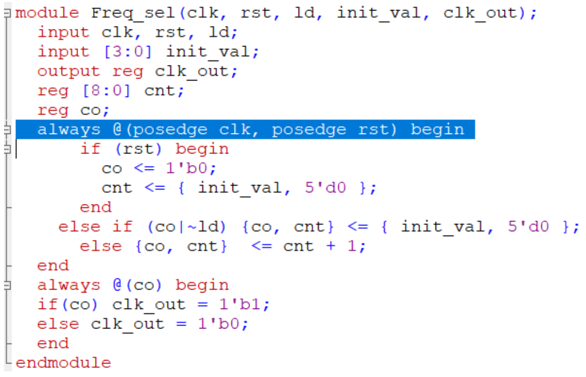
V. Amplitude Selector

Another option of Arbitrary Generator is a amplitude selector that scale down the amplitude of the waveforms.

SW[6:5]	Amplitude
2'b00	1
2'b01	2
2'b10	4
2'b11	8

Table 2: Amplitude selection

Here is the Verilog description of the frequency selector:



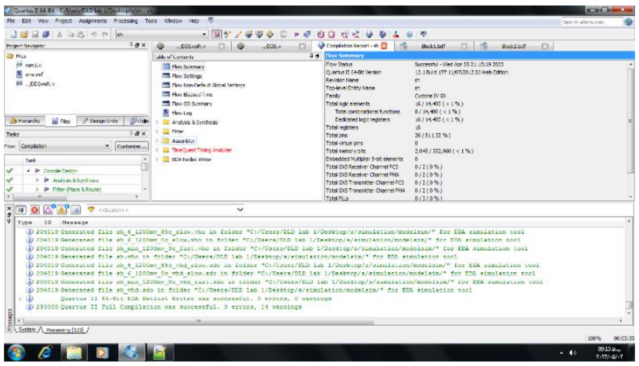
```
module Freq_sel(clk, rst, ld, init_val, clk_out);
  input clk, rst, ld;
  input [3:0] init_val;
  output reg clk_out;
  reg [8:0] cnt;
  reg co;
  always @(posedge clk, posedge rst) begin
      if (rst) begin
        co <= 1'b0;
        cnt <= { init_val, 5'd0 };
      end
    else if (co|~ld) {co, cnt} <= { init_val, 5'd0 };
      else {co, cnt}  <= cnt + 1;
  end
  always @(co) begin
  if(co) clk_out = 1'b1;
  else clk_out = 1'b0;
  end
endmodule
```

Fig. 10 Frequency selector Verilog description.

VI. ROM

For implementing the ROM unit, we have 3 options. First is to use wizard tool and make a ROM in quartus based on a file which named "sine.mif." to do this we need to synthesize DDS unit in quartus and connect it to the ROM unit.

The chip planner and details of this method can be seen in fig.12 & fig.13.



Flow Summary

Flow Status	Successful - Wed Apr 05 21:15:19 2023
Quartus II 64-Bit Version	12.1 Build 177 11/07/2012 SJ Web Edition
Revision Name	sh
Top-level Entity Name	sh
Family	Cyclone IV GX
Total logic elements	16 / 14,400 (< 1 %)
Total combinational functions	8 / 14,400 (< 1 %)
Dedicated logic registers	16 / 14,400 (< 1 %)
Total registers	16
Total pins	26 / 81 (32 %)
Total virtual pins	0
Total memory bits	2,048 / 552,960 (< 1 %)
Embedded Multiplier 9-bit elements	0
Total GXB Receiver Channel PCS	0 / 2 (0 %)
Total GXB Receiver Channel PMA	0 / 2 (0 %)
Total GXB Transmitter Channel PCS	0 / 2 (0 %)
Total GXB Transmitter Channel PMA	0 / 2 (0 %)
Total PLLs	0 / 3 (0 %)

```
Type  ID     Message
      204019 Generated file sh_6_1200mv_85c_slow.vho in folder "C:/Users/DLD lab 1/Desktop/s/simulation/modelsim/" for EDA simulation tool
      204019 Generated file sh_6_1200mv_0c_slow.vho in folder "C:/Users/DLD lab 1/Desktop/s/simulation/modelsim/" for EDA simulation tool
      204019 Generated file sh_min_1200mv_0c_fast.vho in folder "C:/Users/DLD lab 1/Desktop/s/simulation/modelsim/" for EDA simulation tool
      204019 Generated file sh.vho in folder "C:/Users/DLD lab 1/Desktop/s/simulation/modelsim/" for EDA simulation tool
      204019 Generated file sh_6_1200mv_85c_vhd_slow.sdo in folder "C:/Users/DLD lab 1/Desktop/s/simulation/modelsim/" for EDA simulation tool
      204019 Generated file sh_6_1200mv_0c_vhd_slow.sdo in folder "C:/Users/DLD lab 1/Desktop/s/simulation/modelsim/" for EDA simulation tool
      204019 Generated file sh_min_1200mv_0c_vhd_fast.sdo in folder "C:/Users/DLD lab 1/Desktop/s/simulation/modelsim/" for EDA simulation tool
      204019 Generated file sh_vhd.sdo in folder "C:/Users/DLD lab 1/Desktop/s/simulation/modelsim/" for EDA simulation tool
             Quartus II 64-Bit EDA Netlist Writer was successful. 0 errors, 0 warnings
      293000 Quartus II Full Compilation was successful. 0 errors, 14 warnings
```

Fig. 11 area report

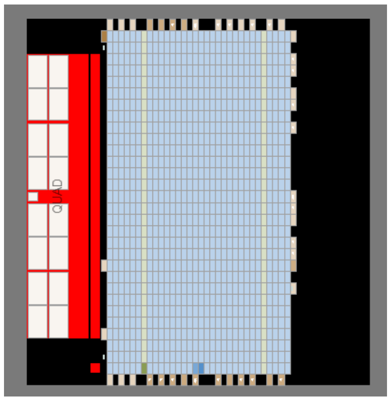
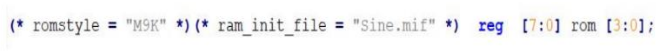


Fig. 12 pin planner

It can be seen that both logic units and memory units of FPGA are used to implement this ROM.

2 other options to build a ROM is using a specific command in quartus. The difference of this 2 method is using the `(* romstyle = "M9K" *)` part or don't use it. By using this command, quartus will use memory units of FPGA to build a ROM. But if we ignore that part of command, it will make a ROM using logic units of FPGA.



```
(* romstyle = "M9K" *)(* ram_init_file = "Sine.mif" *)  reg  [7:0] rom [3:0];
```

Fig. 13 ROM command

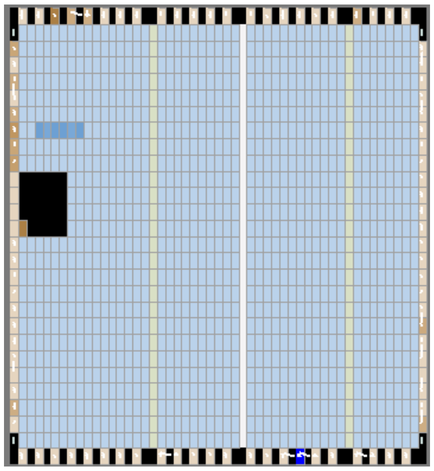


Fig. 14 ROM pin planner without using "M9K"

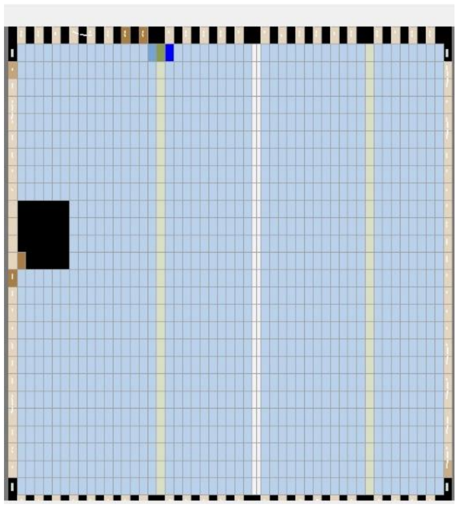


Fig. 15 ROM pin planner using "M9K"

VII. QUARTUS IMPLEMENTATION

Arter generating Verilog codes for each unit and building a ROM unit, we built a new project in quartus and compiled that codes there. After that to connecting all the blocks and units, we made block symbols for each unit and connect them by wires and buses. Besides that we define some input and output modules in block diagram. This modulse used for implementation on FPGA by pin assignments.

we compiled the schemetic design and synthesize a Verilog code of our top module and verify it's operation via a test bench in modelsim and it worked perfect.

After that we planned the chip planner and assigned each input and output to specific unit on FPGA board.

After that we compiled the design again and drive FPGA board with our code and design.

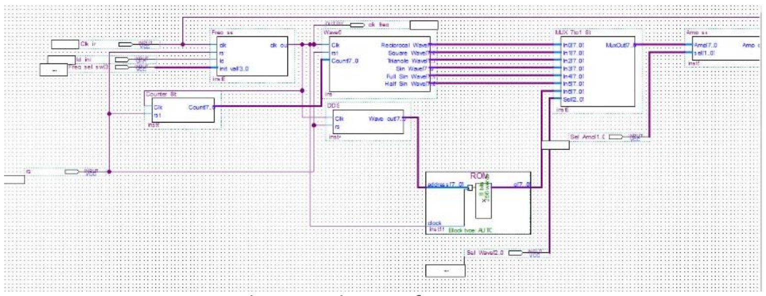
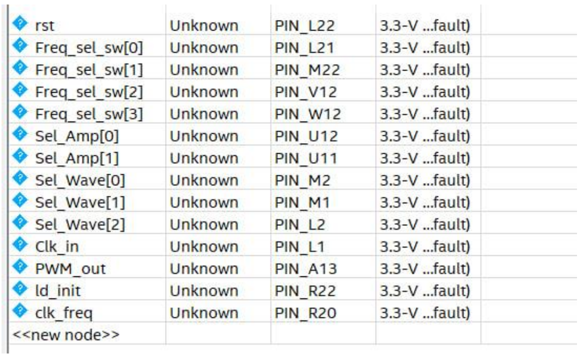


Fig. 16 schemetic design of wave generator



rst	Unknown	PIN_L22	3.3-V ...fault)
Freq_sel_sw[0]	Unknown	PIN_L21	3.3-V ...fault)
Freq_sel_sw[1]	Unknown	PIN_M22	3.3-V ...fault)
Freq_sel_sw[2]	Unknown	PIN_V12	3.3-V ...fault)
Freq_sel_sw[3]	Unknown	PIN_W12	3.3-V ...fault)
Sel_Amp[0]	Unknown	PIN_U12	3.3-V ...fault)
Sel_Amp[1]	Unknown	PIN_U11	3.3-V ...fault)
Sel_Wave[0]	Unknown	PIN_M2	3.3-V ...fault)
Sel_Wave[1]	Unknown	PIN_M1	3.3-V ...fault)
Sel_Wave[2]	Unknown	PIN_L2	3.3-V ...fault)
Clk_in	Unknown	PIN_L1	3.3-V ...fault)
PWM_out	Unknown	PIN_A13	3.3-V ...fault)
ld_init	Unknown	PIN_R22	3.3-V ...fault)
clk_freq	Unknown	PIN_R20	3.3-V ...fault)
<<new node>>			

Fig. 17 Chip planner assignment

VIII. IMPLEMENTATION

Now on a FPGA device we can choose to watch which of generated waves and set it's amplitude and frequency. To watch the analog waveform, we need to a RC circuit and a oscilloscope. We connect the output port of waveform generator to the resistor and connect the other base of it into the capacitor. another base of capacitor will be grounded. The resistance of resistor is 1KΩ and the capacitance is 10nF.

And as final work, we will connect the ground of oscilloscope port into the grounded base of capacitor and connect the observer port into the midle base of resistor and capacitor.

By now, every waveforms can be seen and the work is done.

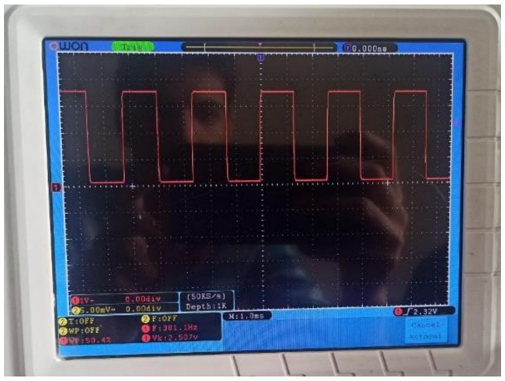


Fig. 18 Square wave

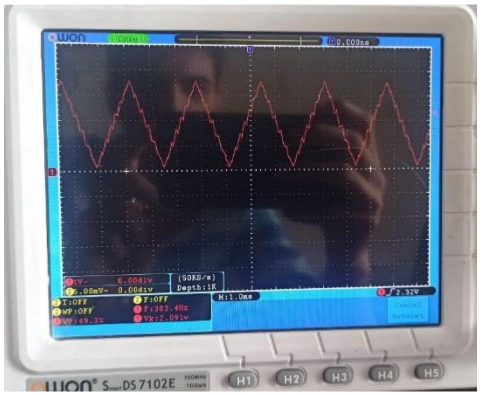


Fig.19 Triangle wave

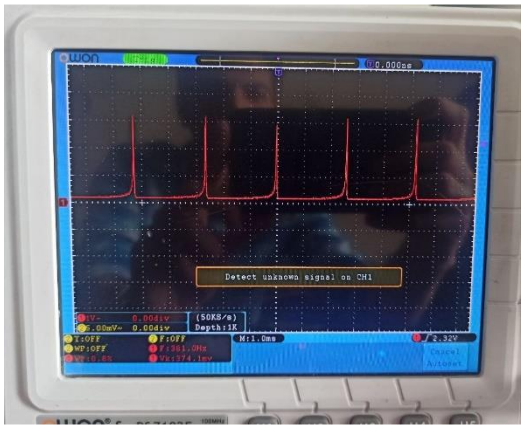


Fig. 20 Recipal wave

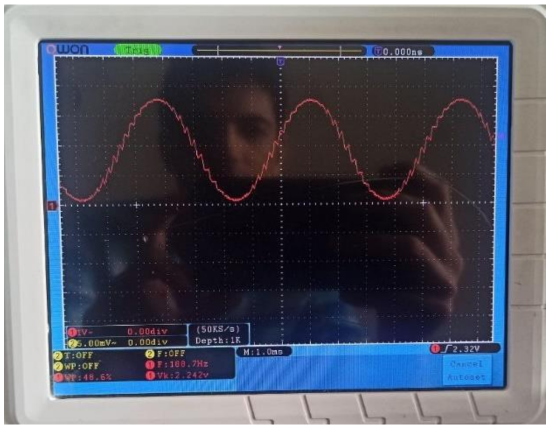


Fig. 21 Sine wave

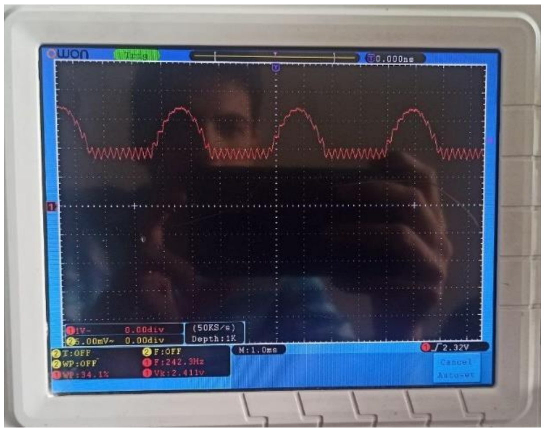


Fig. 22 Half wave

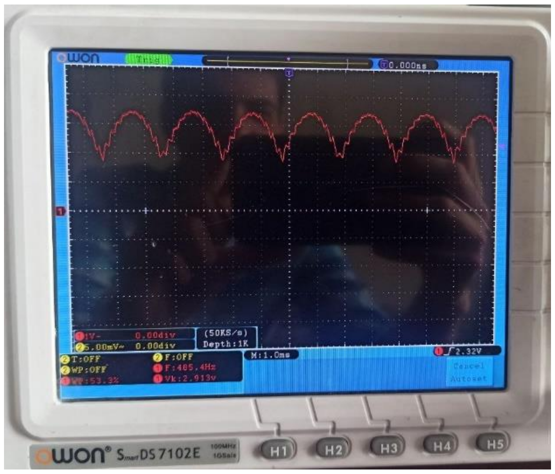


Fig23. full wave.

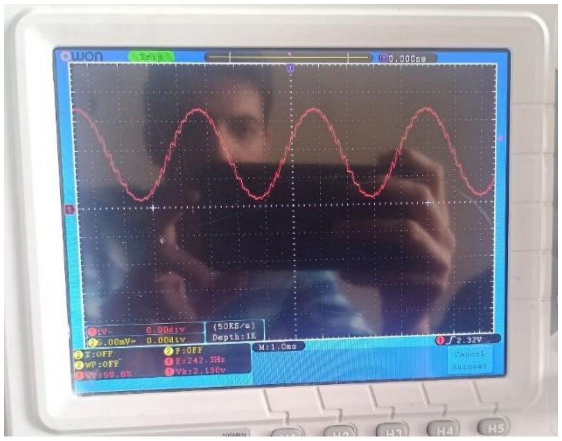


Fig. 24 DDS ROM sine wave

VII. Conclusion

In this experiment, we programmed a **Waveform Generator** with ModelSim and Quartus using Cyclone II and we designed different waves and change their amplitude and frequency, at last, we created a PWM to convert a digital signal into an analog signal.

We learned about unit implementation of FPGA and the difference between logic and memory units.