Experiment 3 - Function Generator

Mohammad Ghareh Hasanloo, 810198461

Danial Saeedi, 810198571

Abstract— **This is the report of experiment 3 and in this experiment, we have create an Arbitrary Function Generator (AFG) which generates a wide variety of waveforms with different amplitude and frequency. The shape of output can be sine, square, rhomboid, saw-tooth and any arbitrary waveform.**

Keywords— **Waveform Generator, Amplitude Selector, DAC, ROM, Frequency Selector, Full-wave rectified, Half-wave rectified, Pulse Width Modulation**

1 Waveform Generator

In this part we are going to create waveform generator which can display shapes include reciprocal, square, triangle, sine, full waved rectified and half wave rectified. These shapes are based on a counter. The output of frequency selector is the input working as clock for the waveform. The results can be seen below.

Fig. 1 Triangle Output

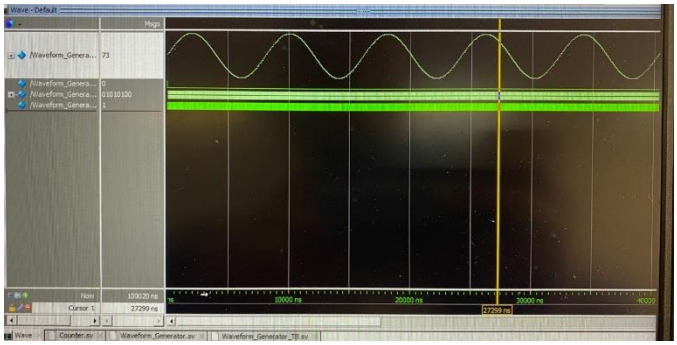

Fig. 2 Sine Output

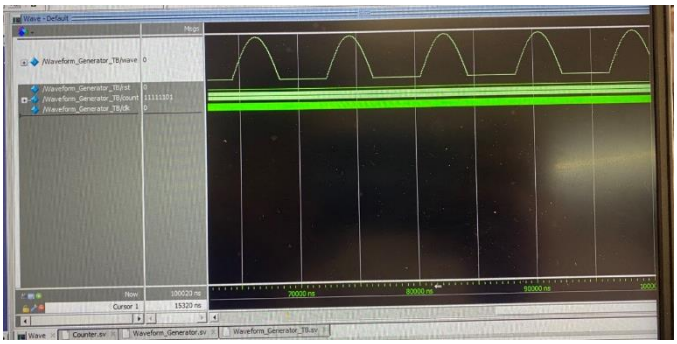

Fig. 3 Half-waved Rectified Output

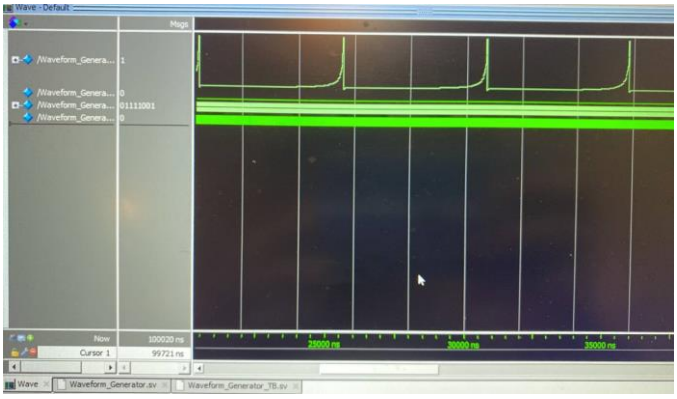

Fig. 4 Reciprocal Output

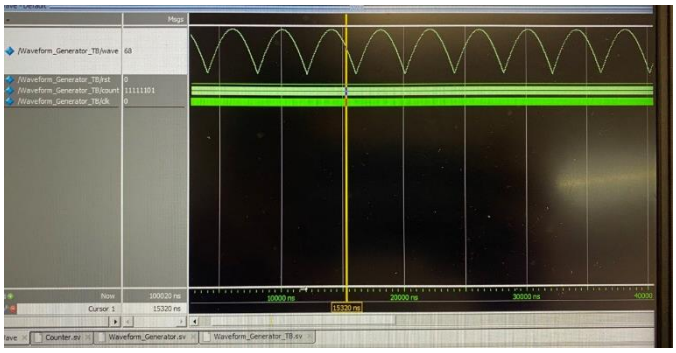

Fig. 5 Full-waved Rectified Output

Fig. 6 Square Output

Also beside all of mentioned output shapes, Direct Digital Synthesis (DDS) is designed to generate arbitrary phase output from a clock.

The final design of circuit can be seen below.

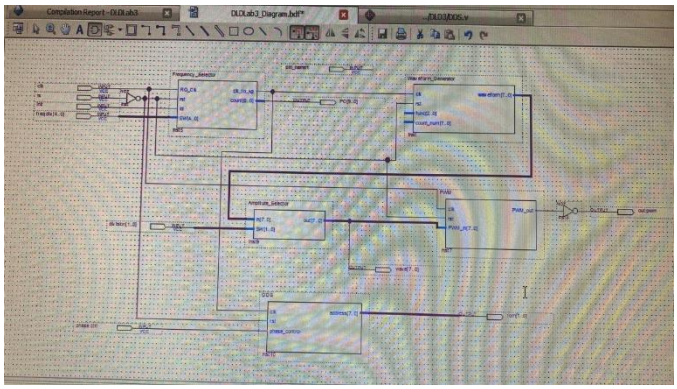

Fig. 7 Final Design in Quartus

Before Adding frequency and amplitude selector, the function generator is tested by the code below.

```
`timescale 1ns/1ns

module Waveform_Generator_TB ();

    reg clk = 0 , rst = 1;
    wire [7:0]count , wave;

    Waveform_Generator CUT (clk, rst, 3'b010, count, wave);
    counter UUT(.clk(clk),.pin(),.select(1'b1),.ld(),.rst(rst),.en(1'b1),.pout(count),.co());

    always #10 clk = ~clk;
    initial begin
        #20 rst = 0;
        #100000 $stop;
    end

endmodule
```

Fig. 8 Waveform Generator Testbench

2 Digital to Analog conversion using PWM

In this part, a Pulse Width Modulation (PWM) is used for digital to analog conversion. The code is shown below. It checks that if input of PWN is less than counter, then output of PWN is set to one, otherwise zero.

```
`timescale 1ns/1ns
module PWM(
    input clk,rst,
    input [7:0] PWM_in,
    output PWM_out
);

reg [7:0] cnt;
always @(posedge clk, posedge rst)begin
    if(rst)
        cnt<=8'b00000000;
    else
        cnt <= cnt + 1'b1;
end

assign PWM_out =(PWM_in < cnt) ? 1'b1 : 1'b0;
endmodule
```

Fig. 9 PWM Verilog Code

The output of PWM's testbench is shown below.

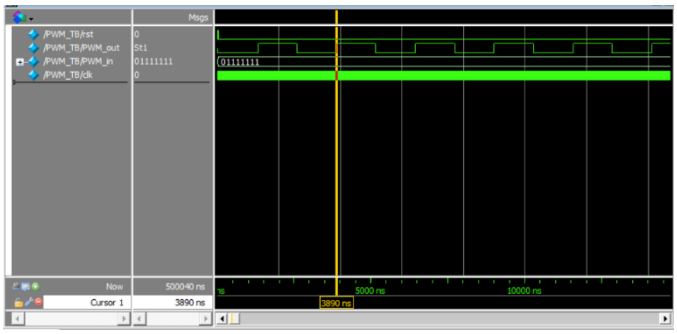

Fig. 10 Counter Verilog Description

3 Frequency Selector

A frequency selector is added before waveform generator to divide the input to a value and you are able to choose what value you want to divide. The output is shown below.

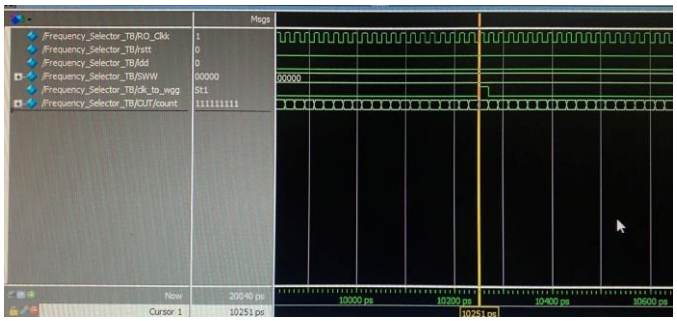

Fig. 11 Frequency Selector Testbench Output

```
module Frequency_Selector_TB();
  reg RO_Clkk, rstt ,ldd;
  reg [4:0] SWW;
  wire clk_to_wgg;
  wire [8:0] countt;

  Frequency_Selector CUT(RO_Clkk,rstt, ldd, SWW, clk_to_wgg, countt);

  initial begin
    RO_Clkk = 1'b0;
    repeat(20000) #10 RO_Clkk = ~RO_Clkk;
  end

  initial begin rstt = 1'b0; end

  initial begin SWW = 5'b0; end

  initial begin
  #20 ldd = 1;
  #20 ldd = 0;
  #20000 $stop;
  end

endmodule
```

Fig. 12 Frequency Detector Testbench Output

4 Amplitude Selector

This function is designed to divide the output by a number. Here we have Amplitudes 1, 2, 4 and 8. The related testbench is shown below.

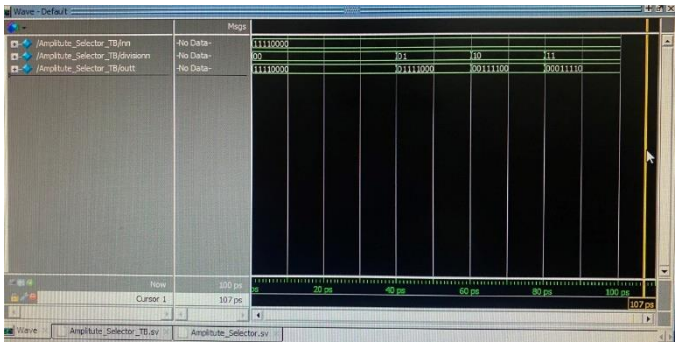

Fig. 13 Amplitude Selector Output

```
module Amplitute_Selector_TB();
  reg [7:0] inn;
  reg[1:0] SWW;
  wire [7:0] outt;

  Amplitute_Selector UUT(inn, SWW, outt);

  initial begin
    inn = 8'b11110000;
  end

  initial begin SWW = 2'b00; end

  initial begin
  #20 SWW = 2'b00;
  #20 SWW = 2'b01;
  #20 SWW = 2'b10;
  #20 SWW = 2'b11;
  #20 $stop;
  end
endmodule
```

Fig. 14 Amplitude Selector Testbench