Experiment #3 – Function Generator

Parsa Nassery, 810199561 Shayan Kashefi, 810199585

***Abstract*— This document is a student report to experiment #3 of Digital Logic Laboratory course at ECE Department, University of Tehran. In this series of experiments, the function generator that can make Reciprocal, Square, Triangle, Sin, Full-wave rectified, Half-wave rectified wave was created.**

***Keywords*— Waveform Generator, Frequency selector, Amplitude Selector, ROM.**

I.INTRODUCTION

In this experiment, a Waveform Generator, PWM, Frequency Selector , Amplitude Selector were made.

II. WAVEFORM GENERATOR

A. Verilog Description for Waveform Generator

```
module waveform_generator_processor(input clk, rst,
    input [2:0] select, output [7:0] out);
    reg [7:0] phase;
    reg [7:0]counter = 8'd0;
    reg [15:0] sine = 16'd0, cose = 16'd30000;
    reg [15:0] tmp = 16'd0;
    reg d = 1'd0;
    always@(posedge clk, posedge rst) begin
        counter = counter + 8'd1;
        tmp = {{6{cose[15]}}, cose[15:6]};
        sine = sine + tmp;
        tmp = {{6{sine[15]}}, sine[15:6]};
        cose = cose - tmp;
    end

    always@(posedge clk, posedge rst) begin
        if (clk) d = (counter == 8'd0) ? ~d : d;
        if(rst) phase = 8'd0;
        else begin
            case (select)
                3'd0 : phase = 8'd255 / (8'd255 - counter + 8'd1);
                3'd1 : phase = (counter > 8'd127) ? 8'd255 : 8'd0;
                3'd2 : phase = (d) ? (8'd255 - counter) : counter;
                3'd3 : phase = sine[15:8] + 8'd127;
                3'd4 : phase = (sine[15]) ? (8'd255 - sine[15:8] - 8'd128)
                                          sine[15:8] + 8'd128;
                3'd5 : phase = (sine[15]) ? 8'd127 : sine[15:8] + 8'd128;
            endcase
        end
    end

    assign out = phase;
endmodule
```

Fig. 1 Waveform Generator Verilog

We have a counter that counts by clock for generate Reciprocal, Square and Triangle waves.
We calculate sine wave with these formula:

```
tmp = {{6{cose[15]}}, cose[15:6]};
sine = sine + tmp;
tmp = {{6{sine[15]}}, sine[15:6]};
cose = cose - tmp;
```

Fig. 2 Sine and Cos formula

We use sine for Sine wave, Full-wave rectified and Half-wave rectified.

B. Reciprocal

The wave Reciprocal is obtained with this formula and if the input wave selector is 000, this wave is given to the output.

$$\frac{255}{255 - counter + 1}$$

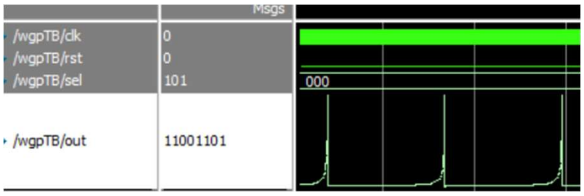

Fig. 3 The Reciprocal wave

C. Square

If wave selector is 001 then we put square wave in output.
To generate a square wave, we put a 255 in the output if the counter is greater than 127 and a 0 if the counter is less than 127.

$$(counter) > 8'd127)?\, 8'd255 : 8'd0$$

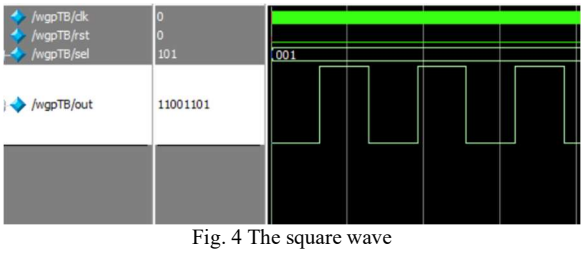

Fig. 4 The square wave

D. Triangle

If wave selector is 010 then we put triangle wave in output.
To detect the slope of the triangle wave, we define a reg d and when the counter becomes zero, we set d equal to ~d.
If d is 1, the output is 255-counter. Else the output is counter.

$$(d)?\,(8'd255 - counter):counter$$

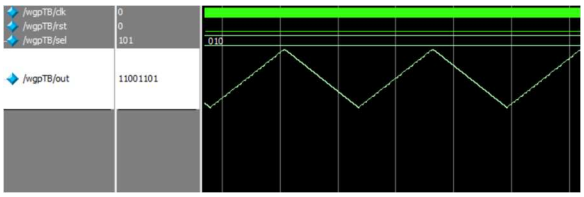

Fig. 5 The triangle wave

E. Sine

The sine wave is the first wave that we don't use counter to generate it. We use sine for generate the wave.
If the wave selector is 011, we put sine[15:8] + 8'd127 in output.

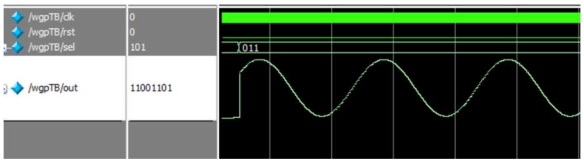

Fig. 6 The Sine wave

F. Full-wave rectified

If the wave selector is 100, then we put Full-wave in output.
If the sine is negative then we put 127 – sine[15:8] in output. Else we put sine[15:8] + 128.
To check whether the sine is positive or negative, we use sine[15].
If sine[15] is one, the sine is negative and if it is zero, the sine is positive.

$$(sine[15])?\,(8'd127 - sine[15:8]):\\ sine[15:8] + 8'd128$$

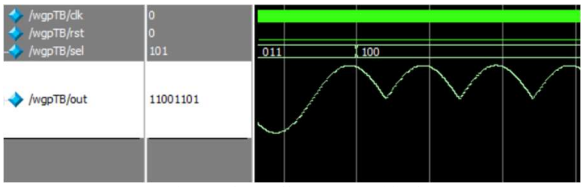

Fig. 7 The Full-wave rectified

G. Half- wave rectified

If the wave selector is 101, then we put Half-wave in output.
If the sine is negative we put 127 in output. Else we put sine[15:8] + 128 in output.
To check whether the sine is positive or negative, we use sine[15].
If sine[15] is one, the sine is negative and if it is zero, the sine is positive

$$(sine[15])?\,8'd127:sine[15:8] + 8'd128$$

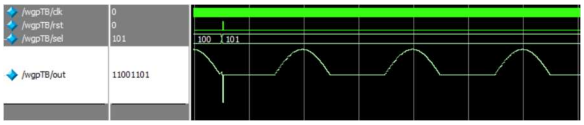

Fig. 8 The Half-wave rectified

H. Testbench for Waveform Generator Processor

```
`timescale 1ns / 1ns
module wgpTB();
    reg clk = 1'b0, rst = 1'b0;
    reg [2:0] sel = 3'd1;
    wire [7:0] out;
    waveform_generator_processor inst(.clk(clk),
    .rst(rst), .select(sel), .out(out));

    initial repeat(1000000) #50 clk = ~clk;
    initial begin
        #250000 sel = 3'd2;
        #500000 sel = 3'd0;
        #500000 sel = 3'd3;
        #250000 sel = 3'd4;
        #250000 sel = 3'd5;
        #200 rst = 1'b1;
        #200 rst = 1'b0;
    end
endmodule
```

Fig. 9 Testbench WGP

I. Block diagram for whole waveform generator

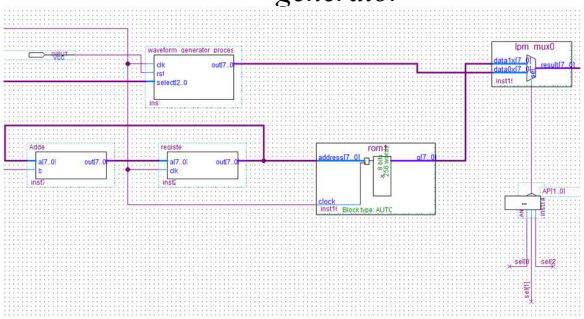

Fig. 10 Block diagram WG

We put an And with SW[9:7] to multiplexer select. And the multiplexer output is our waveform generator output.
For DDS we use a register, Adder and a ROM loaded with mif file.

We use these Verilog file for Adder and Register.

```
module register(input [7:0] a, input clk, output reg [7:0] out);
    always@(posedge clk) begin
        out <= a;
    end
endmodule
```

Fig. 11 Register Verilog

```
module Adder(input [7:0] a, input b, output [7:0]out);
    assign out = a + b + 1;
endmodule
```

Fig. 12 Adder Verilog

III. PWM

A. PWM Verilog

This is PWM Verilog. We have a counter that counts with clock. We get a 8-bit data and if counter is bigger than data, put zero in output else put one in output.

```
module PWM (input [7:0] data, input clk ,output reg out_pulse);
    reg [7:0] counter = 8'b0;
    always @(posedge clk) begin
        counter <= counter + 8'd1;
        out_pulse = (counter > data) ? 1'b0 : 1'b1;
    end
endmodule
```

Fig. 13 PWM Verilog

The input data specifies how many clock the output will be zero for. For first data clock, the output is zero, then for 255 - data, the output becomes one.

B. PWM testbench

This is our testbench.

```
`timescale 1ns / 1ns
module PWM_tb();
  reg clk = 1'b0;
  reg [7:0] data = 8'd20;
  wire PULSE_out;
  PWM inst(.clk(clk), .data(data), .out_pulse(PULSE_out));
  initial repeat(100000) #50 clk = ~clk;
  initial begin
    #26000 data = 8'd128;
  end
endmodule
```

Fig. 14 PWM testbench

We put 128 for input data. So the first 128 clock must be zero and the next 128 must be one, and this process continues again.
This is result.

Fig. 14 PWM testbench result

IV. FREQUENCY SELECTOR

A. Frequency Selector Description

We get a 3-bit SW in input. We concat SW with a 6-bit fixed number that we have chosen randomly and put it in our 9-bit counter. We count the counter with the clock. Carry out is set to one when all the digits of the counter are one, otherwise its value is zero. Whenever carry out becomes one, we put the value of concat SW and our random number on the counter.

B. Frequency Selector Verilog

We choose 101000 for our random fix number.

```
module frequencyslct (input[2:0] sw, input clk, output  carry_out);
    reg [8:0] count = 9'b000101000;
    always@(posedge clk) begin
        if(carry_out == 1'b1) begin
            count <= {sw ,6'b101000};
        end
        count <= count + 1'b1;
    end
    assign carry_out = &count;
endmodule
```

Fig. 15 Frequency selector Verilog

C. TB for Frequency selector

We put 101 as SW input.

```
`timescale 1ns / 1ns
module frequencyslct_tb();
  reg [2:0] sw = 3'b101;
  reg clk = 1'b0;
  wire carry_out;
  frequencyslct inst(.clk(clk), .sw(sw), .carry_out(carry_out));
  initial repeat(100000) #50 clk = ~clk;
endmodule
```

Fig. 16 Frequency Selector Testbench

D. Testbench Result

The output becomes one after 303 clock.

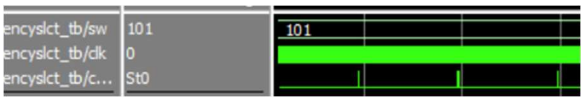

Fig. 17 Frequency Selector Testbench result

V. AMPLITUDE SELECTOR

A. Amplitude Selector Description

In the Amplitude Selector we get a 2-bit input SW, which decides what the output will be.
If SW is zero, the output will be the same as the input wave. Otherwise, the wave will shift to the right by SW.

B. Amplitude Selector Verilog

```
`timescale 1 ns/ 1 ns
module amplitudeselect ( input [7:0] wave,
 input [1:0] sw ,output reg [7:0] out_wave);
    always@(sw, wave) begin
        case(sw)
            2'b00:  out_wave = wave;
        2'b01:  out_wave = wave>>1;
        2'b10:  out_wave = wave>>2;
        2'b11:  out_wave = wave>>3;
        endcase
    end
endmodule
```

Fig. 18 Amplitude Selector Verilog

C. Amplitude Selector Testbench

We set wave to 11111111 and set SW to 00 at first. After a while change it to 11 then 01. So the output wave must be 11111111 first and then shifted three bits to the right and become 00011111. After changing SW, shift one unit and become 01111111.

```
`timescale 1ns / 1ns
module amplitudeslct_tb();
  reg [1:0] SW = 2'b00;
  reg [7:0] wave = 8'b11111111;
  wire [7:0] out_wave;
  amplitudeselect inst(wave,SW,out_wave);
  initial begin
    #200 SW=2'b11;
    #200 SW=2'b01;
    #200 $stop;
  end

endmodule
```

Fig. 19 Amplitude Selector testbench

D. Amplitude Selector Testbench Result

Fig. 20 Amplitude Selector Testbench Result

VI. TOTAL DESIGN

A. Total Block Diagram

We add the Frequency Selector, the Amplitude Selector, the PWM and the Waveform Generator that we made in the first part to our wizard project Quartus. We connected these parts in our block diagram to make our total design.

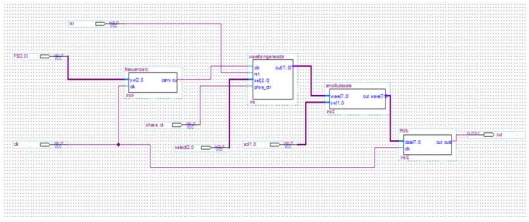

Fig. 21 Total Block Diagram

VII. FPGA IMPLEMETATION

A. Pin Planner

The pin planer is shown in Fig. 22

Node Name	Direction	Location	I/
AP[1]	Input	PIN_W12	7
AP[0]	Input	PIN_V12	7
clk	Input	PIN_L1	2
fs[2]	Input	PIN_M22	6
fs[1]	Input	PIN_L21	5
fs[0]	Input	PIN_L22	5
out	Output	PIN_A11	3
phse_ctr	Input	PIN_L2	2
rst	Input	PIN_R22	6
sel[2]	Input	PIN_M2	1
sel[1]	Input	PIN_U11	8
sel[0]	Input	PIN_U12	8

Fig. 22 Pin Planer

The total design was put on FPGA and we connected the output to the oscilloscope by wire. For the clock, we used FPGA's own clock, which clocks with a fixed value of 50-MHz.

We connected the frequency input, waveform select, reset and amplitude input to FPGA keys.

B. FGPA Implementation

After FPGA Implementation, the signals seen on the oscilloscope did not match the signals that should have been seen. The signal seen was just noise and even reset key on FPGA had no effect on the signal on the oscilloscope, while the circuit in Modelsim was fully checked.

Many ways were tested to solve this problem, but due to lack of time, the problem was not solved.

Even the Breadboard was replaced and the whole circuit was connected and checked again, but the problem was not solved.

Node Name	Direction	Location	I/O Standard
AP[1]	Unknown	PIN_W12	3.3-V ...fault)
AP[0]	Unknown	PIN_V12	3.3-V ...fault)
fs[2]	Unknown	PIN_M22	3.3-V ...fault)
fs[1]	Unknown	PIN_L21	3.3-V ...fault)
fs[0]	Unknown	PIN_L22	3.3-V ...fault)
phse_ctr	Unknown	PIN_L2	3.3-V ...fault)
rst	Unknown	PIN_M1	3.3-V ...fault)
sel[2]	Unknown	PIN_M2	3.3-V ...fault)
sel[1]	Unknown	PIN_U11	3.3-V ...fault)
sel[0]	Unknown	PIN_U12	3.3-V ...fault)
clk	Unknown	PIN_L1	3.3-V ...fault)
out	Unknown	PIN_A13	3.3-V ...fault)
tesuuuut[7]	Unknown	PIN_U18	3.3-V ...fault)
tesuuuut[6]	Unknown	PIN_Y18	3.3-V ...fault)
tesuuuut[5]	Unknown	PIN_V19	3.3-V ...fault)
tesuuuut[4]	Unknown	PIN_T18	3.3-V ...fault)
tesuuuut[3]	Unknown	PIN_Y19	3.3-V ...fault)
tesuuuut[2]	Unknown	PIN_U19	3.3-V ...fault)
tesuuuut[1]	Unknown	PIN_R19	3.3-V ...fault)
tesuuuut[0]	Unknown	PIN_R20	3.3-V ...fault)
COOOOOOO	Unknown	PIN_R17	3.3-V ...fault)
OUTKHODI	Unknown	PIN_Y21	3.3-V ...fault)
CLKKKK	Unknown	PIN_L18	3.3-V ...fault)

Fig. 23 pin planner for check

To check the total design and the output result, we connected it to FPGA LEDs from different parts of the circuit.
We checked the output of the WGP, the output of the Amplitude Selector in two separate times through eight LEDs in a row, and the LEDs were lit correctly.
We also gave each of the Clock, Frequency Selector and main output or PWM output to the single LEDs and they also worked correctly.
The accuracy of the LED lights was also checked and everything was as expected.

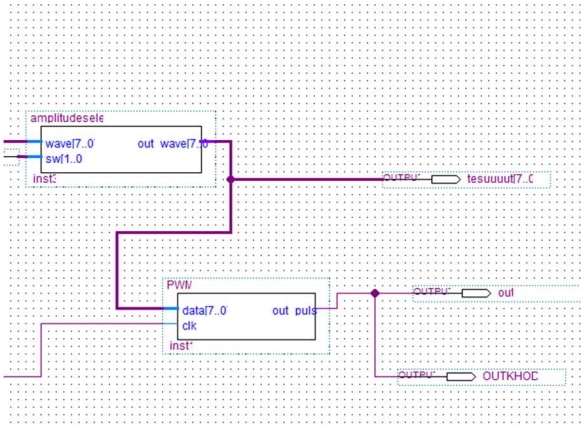

Fig. 24 example for check wiring

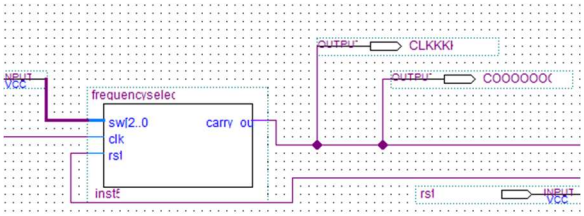

Fig. 25 example for check wiring

In the end, the problem was not identified and solved even with the help of TAs.

In Figure 23, you can see the pin planer that was used to check the LEDs.

Also, in Figure 24 and Figure 25, an example of outputting from the circuit for connecting to LEDs is shown.