Experiment 3 – Function Generator

Iman Rasouli – 810199425 / Parsa Sattari – 810199436

***Abstract:** In this experiment, we are to design an Arbitrary Function Generator which is capable of generating various waveforms, and then implement it on FPGA board.*

***Keywords:** Function Generator – Waveform – AFG – Sine wave -selector*

I. EXPERIMENT

A. Waveform Generator

The waveform generator's output must be one of the 7 waveforms mentioned in the manual. For this purpose, a 7-to-1 multiplexer can be used so that any of its 7 inputs corresponds to one of those waveforms.

For the first 3 waveforms, which are reciprocal, square and triangle, we need a counter. (Fig. 1)

```
always@(posedge clk, posedge rst)begin
if(rst)
    count <= 8'b00000000;
else
    count <= count + 1;
end
```

Fig 1. Counter's Verilog Description

To have reciprocal waveform, we approximate a function close to it. In this case, we considered $\frac{255}{255-x}$. (Fig. 2)

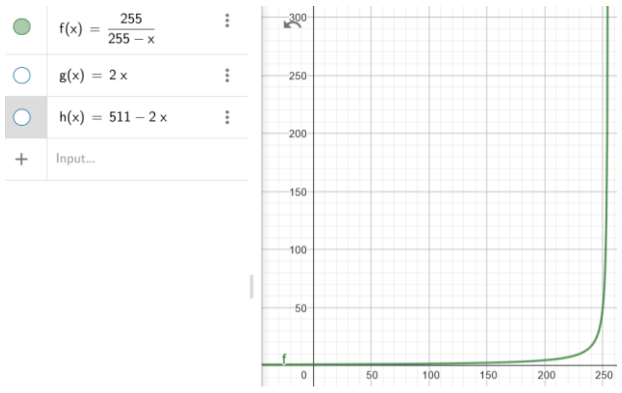


Fig 2. Plot of the function

We just need to assign this value to the output of the reciprocal module. (Fig. 3)

```
always@(posedge clk)begin
    OutRec <= 255/(255 - count);
end
```

Fig 3. Reciprocal wave Verilog Description

Forming the square waveform is as easy as setting the output to 1 for the first half of the count and 0 for the rest. (Fig. 4)

```
always@(posedge clk)begin
    if (count < 128)
        OutSquare <= 8'b0000000;
    else
        OutSquare <= 8'b11111111;
end
```

Fig 4. Square wave Verilog Description

For the third waveform, this piecewise function does the work. (Fig. 5)

$$f(x) = \begin{cases} 2x, & x < 128 \\ 511 - 2x, & x \geq 128 \end{cases}$$

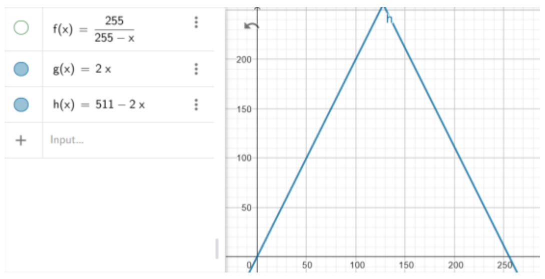


Fig 5. Plot of the function

Like what we've done before, we assign this value to the proper output. (Fig. 6)

```
always@(posedge clk)begin
    if (count < 128)
        OutTri <= count << 1;
    else
        OutTri <= (511 - count << 1);
end
```

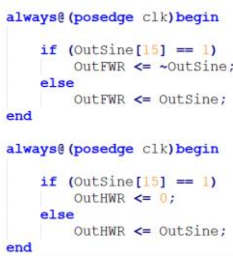
Fig 6. Triangle wave Verilog Description

Sine wave can be generated by the recursive equations mentioned in the manual and to implement that, we use 4 registers. To have a better resolution, all the calculations are performed with 16-bit precision and eventually, the 8 most significant bits are considered as the output. Also, a constant offset is added to the wave because all the values must be positive. The corresponding Verilog description can be seen below. (Fig. 7)

```
always@(posedge clk)begin
    OutSine = A + {B[15],B[15],B[15],B[15],B[15],B[15], B[15:5]};
    C = B - {OutSine[15], OutSine[15], OutSine[15], OutSine[15], OutSine[15], OutSine[15], OutSine[15:5]};
    A <= OutSine;
    B <= C;
end
```

Fig 7. Sine wave Verilog Description

Full-wave and half-wave rectified waveforms are made from sine wave. As far as the original sine wave is positive, they have exactly the same value, but when the sine wave turns negative, it has to be inverted for FW and zero for HW. (Fig. 8)



```
always@(posedge clk)begin

    if (OutSine[15] == 1)
        OutFWR <= ~OutSine;
    else
        OutFWR <= OutSine;
end

always@(posedge clk)begin

    if (OutSine[15] == 1)
        OutHWR <= 0;
    else
        OutHWR <= OutSine;
end
```

Fig 8. Full-wave and half-wave rectified Sine waves Verilog Description

One of function generator outputs is DDS. DDS is used for generating arbitrary phase tunable output from a single fixed-frequency reference clock. The output of DDS module is a quantized version of the output files (in this experiment a sinusoid). The period of this signal is controlled by a phase control value. To generate DDS wave, we need a ROM to store "sine.mif" file's data. As we set the "Phase_ctrl" signal the ROM sends data in order of "Phase_ctrl" value.

To implement this functionality, we need an adder to set ROM address and a register due to synchronize adder and ROM (because the ROM has clock).

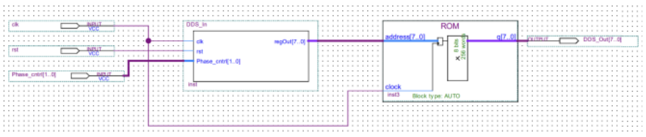
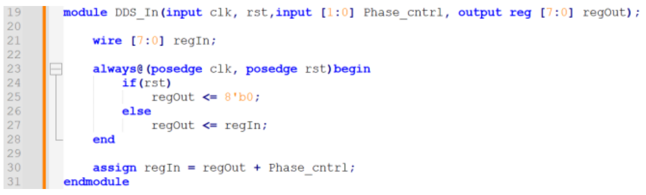


Fig 9. DDS Schematic Diagram



```
module DDS_In(input clk, rst,input [1:0] Phase_cntrl, output reg [7:0] regOut);

    wire [7:0] regIn;

    always@(posedge clk, posedge rst)begin
        if(rst)
            regOut <= 8'b0;
        else
            regOut <= regIn;
    end

    assign regIn = regOut + Phase_cntrl;
endmodule
```

Fig 10. DDS Verilog Description

Note: To avoid using multiplexer between waveform generator and DDS output, we consider the DDS output as an input for waveform generator and it will be considered as one of waveform generator's multiplexer inputs.

Therefore we have the wave generator full description:

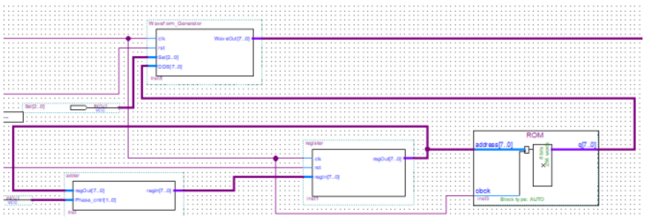
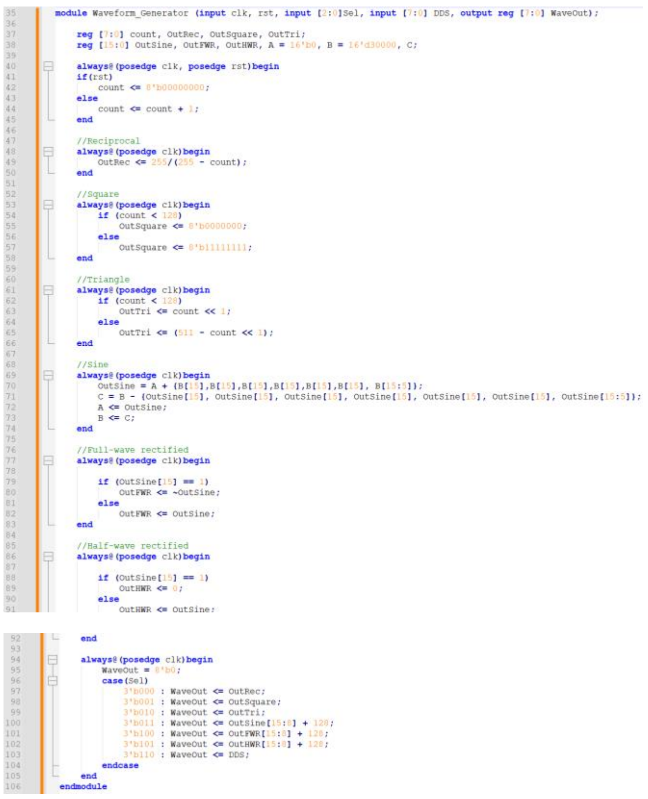


Fig 11. Waveform Generator Schematic Diagram



```
module Waveform_Generator (input clk, rst, input [2:0]Sel, input [7:0] DDS, output reg [7:0] WaveOut);

    reg [7:0] count, OutRec, OutSquare, OutTri;
    reg [15:0] OutSine, OutFWR, OutHWR, A = 16'b0, B = 16'd30000, C;

    always@(posedge clk, posedge rst)begin
    if(rst)
        count <= 8'b00000000;
    else
        count <= count + 1;
    end

    //Reciprocal
    always@(posedge clk)begin
        OutRec <= 255/(255 - count);
    end

    //Square
    always@(posedge clk)begin
        if (count < 128)
            OutSquare <= 8'b0000000;
        else
            OutSquare <= 8'b11111111;
    end

    //Triangle
    always@(posedge clk)begin
        if (count < 128)
            OutTri <= count << 1;
        else
            OutTri <= (511 - count << 1);
    end

    //Sine
    always@(posedge clk)begin
        OutSine = A + {B[15],B[15],B[15],B[15],B[15],B[15], B[15:5]};
        C = B - {OutSine[15], OutSine[15], OutSine[15], OutSine[15], OutSine[15], OutSine[15], OutSine[15:5]};
        A <= OutSine;
        B <= C;
    end

    //Full-wave rectified
    always@(posedge clk)begin

        if (OutSine[15] == 1)
            OutFWR <= ~OutSine;
        else
            OutFWR <= OutSine;
    end

    //Half-wave rectified
    always@(posedge clk)begin

        if (OutSine[15] == 1)
            OutHWR <= 0;
        else
            OutHWR <= OutSine;
     end

     always@(posedge clk)begin
         WaveOut = 8'b0;
         case(Sel)
             3'b000 : WaveOut <= OutRec;
             3'b001 : WaveOut <= OutSquare;
             3'b010 : WaveOut <= OutTri;
             3'b011 : WaveOut <= OutSine[15:8] + 128;
             3'b100 : WaveOut <= OutFWR[15:8] + 128;
             3'b101 : WaveOut <= OutHWR[15:8] + 128;
             3'b110 : WaveOut <= DDS;
         endcase
     end
endmodule
```

Fig 12. Waveform Generator Verilog Description

By providing a testbench for module above, we have the waveforms:

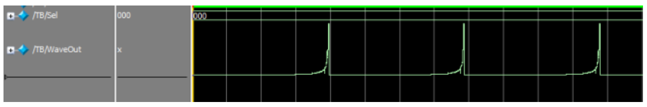


Fig 13. Waveform Generator Output_ Reciprocal

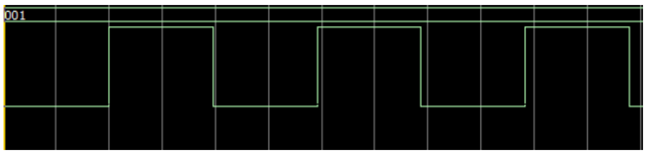


Fig 14. Waveform Generator Output_ Square

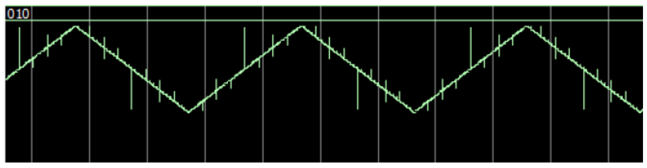


Fig 15. Waveform Generator Output_ Triangle

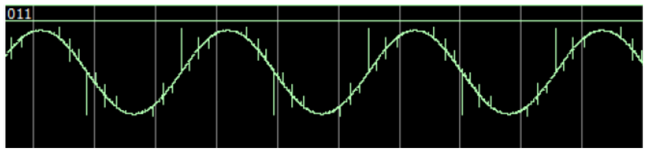


Fig 16. Waveform Generator Output_ Sine

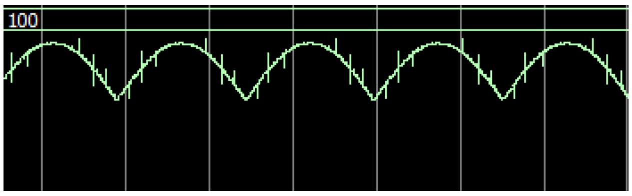


Fig 17. Waveform Generator Output_ Full-wave Rectified

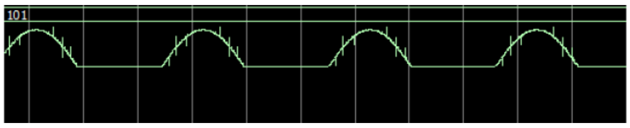


Fig 18. Waveform Generator Output_ Half-wave Rectified

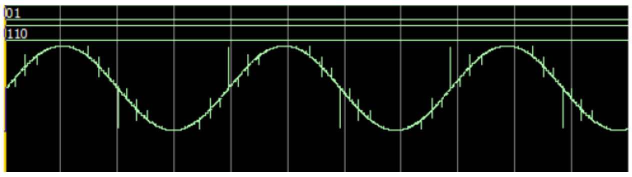


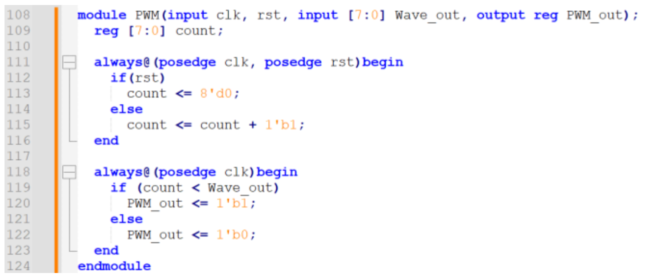
Fig 19. Waveform Generator Output_ DDS_ Phase_ctrl = 1

B. Digital to Analog conversion using PWM

Waveform generator has a digital output. It means it has a discrete value between 0 and 255. But we need to convert it to an analog signal. There are two major methods for this purpose. The one that we used was Pulse Width Modulation (PWM). In this method, the period of each pulse depends on the value of the signal in a specified period.

PWM module has a counter in it. At every single clock, if the value of the input is higher than the count, the PWM output becomes one.

By Averaging this signal, which is done by adding a low-pass filter (RC circuit) at the end part of it, our signal can be converted to analog.


```
module PWM(input clk, rst, input [7:0] Wave_out, output reg PWM_out);
  reg [7:0] count;

  always@(posedge clk, posedge rst)begin
    if(rst)
      count <= 8'd0;
    else
      count <= count + 1'b1;
  end

  always@(posedge clk)begin
    if (count < Wave_out)
      PWM_out <= 1'b1;
    else
      PWM_out <= 1'b0;
  end
endmodule
```

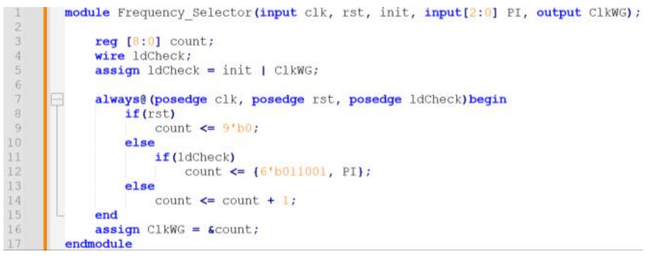
Fig 20. PWM Verilog Description

C. Frequency Selector

In order to set the frequency of the output signal, a frequency selector is required. This can be implemented by a frequency divider which is a simple up-counter.

Due to the limited frequency of the FPGA, we cannot adjust the frequency to whatever we want; We assign our 3-bit input to the least significant bits of the frequency selector.

In this case, our frequency won't change significantly but we won't cross the limits either.


```
module Frequency_Selector(input clk, rst, init, input[2:0] PI, output ClkWG);

    reg [8:0] count;
    wire ldCheck;
    assign ldCheck = init | ClkWG;

    always@(posedge clk, posedge rst, posedge ldCheck)begin
        if(rst)
            count <= 9'b0;
        else
            if(ldCheck)
                count <= {6'b011001, PI};
        else
            count <= count + 1;
    end
    assign ClkWG = &count;
endmodule
```

Fig 21. Frequency Selector Verilog Description

We need a “init” signal that when issued, counter starts to count. After the counting is finished, the counter should reset to its parallel load and start counting from its Parallel load. So, the “init” signal is just for initializing the counter and it should be issued once at the beginning of the function generator progress.

Here is the functionality of frequency selector:

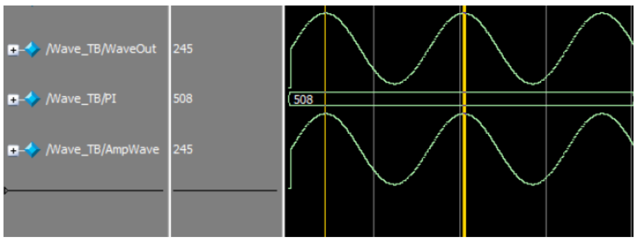


Fig 22. Frequency Selector output for “PI = 508”

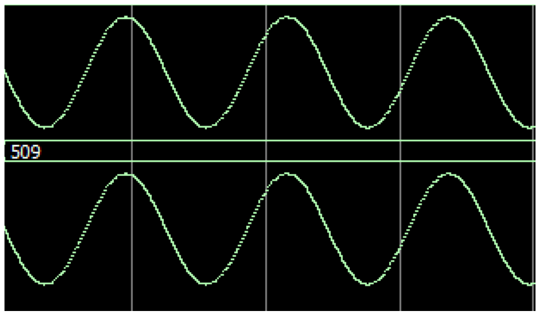


Fig 23. Frequency Selector output for “PI = 509”

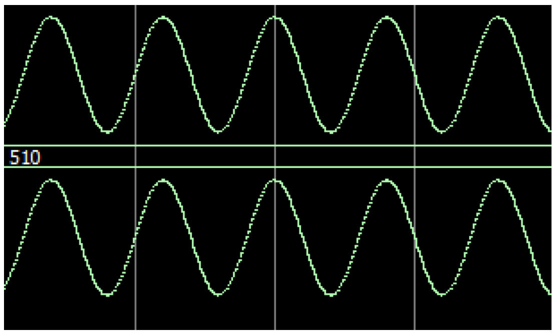


Fig 24. Frequency Selector output for “PI = 510”

And a general view:

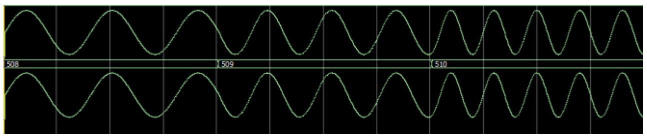


Fig 25. Frequency Selector output general view

According to wave, the higher “PI” will result higher frequency (up-counter counts less, so frequency will divide by a smaller number).

Here is the functionality of frequency selector based on “Phase_ctrl” value in DDS module:

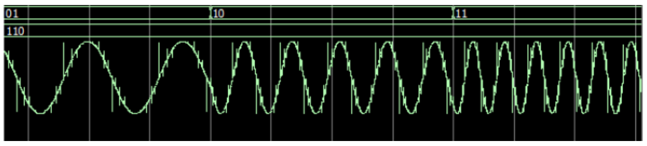
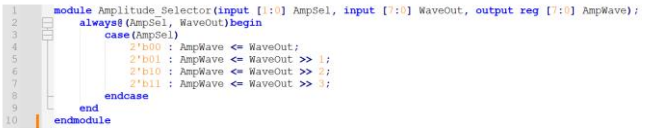


Fig 26. Frequency Selector checking accuracy using DDS

As it shown in waveform, the higher the “Phase_ctrl” value, the higher the frequency will be.

D. Amplitude Selector

One option in function generator is the amplitude of the generated wave. This module scales down the amplitude of waveforms by dividing the output amplitude by a number. For this, we have 4 modes for the output duo to table 2 in manual. This can be implemented by a 4 to 1 multiplexer.



```
module Amplitude_Selector(input [1:0] AmpSel, input [7:0] WaveOut, output reg [7:0] AmpWave);
    always@(AmpSel, WaveOut)begin
        case(AmpSel)
            2'b00 : AmpWave <= WaveOut;
            2'b01 : AmpWave <= WaveOut >> 1;
            2'b10 : AmpWave <= WaveOut >> 2;
            2'b11 : AmpWave <= WaveOut >> 3;
        endcase
    end
endmodule
```

Fig 27. Amplitude Selector Verilog Description

To have desired amplitude, we should set the “AmpSel” that is the selector of multiplexer.

Here we have different modes of amplitude selector via sine wave:

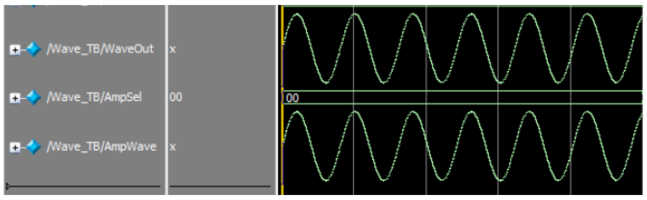


Fig 28. Amplitude Selector output for “AmpSel = 0”

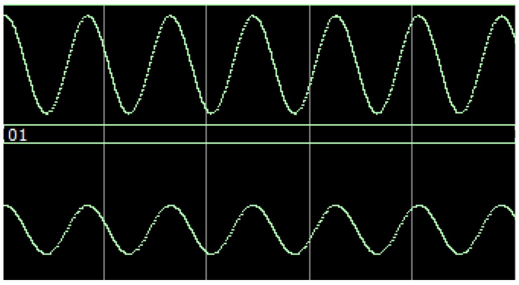


Fig 29. Amplitude Selector output for “AmpSel = 1”

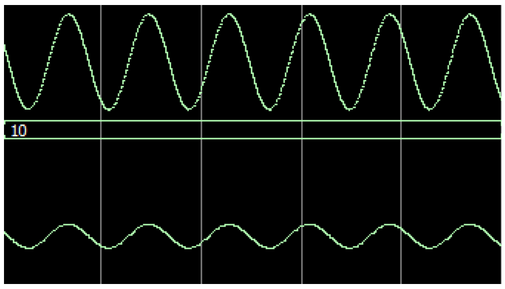


Fig 30. Amplitude Selector output for “AmpSel = 2”

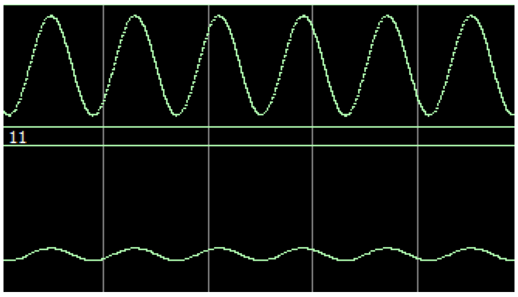


Fig 31. Amplitude Selector output for “AmpSel = 3”

According to the waves, the amplitude is divided by 2, 4 and 8 respectively.

E. The Total design

We have all the components together:

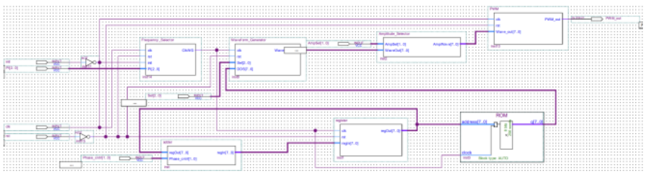


Fig 32. Function Generator Total Design

After synthesizing the design and program the FPGA, via RC circuit and oscilloscope we have the function generator:

I. Reciprocal

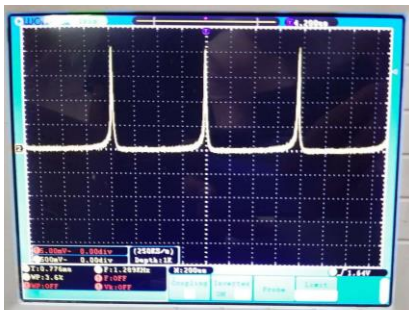


Fig 33. Reciprocal wave

II. Square

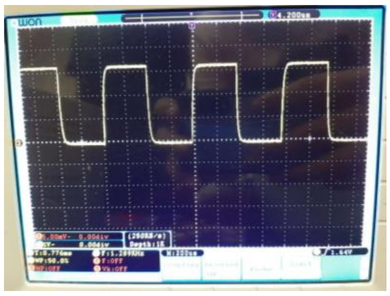


Fig 34. Square wave

III. Triangle

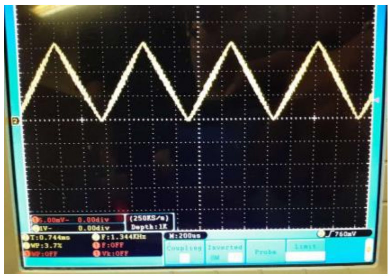


Fig 35. Triangle wave

IV. Sine

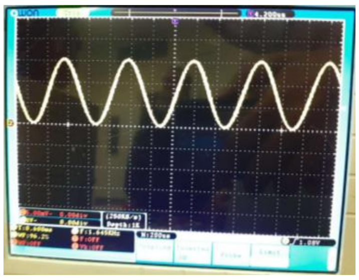


Fig 36. Sine wave

V. Full-wave rectified

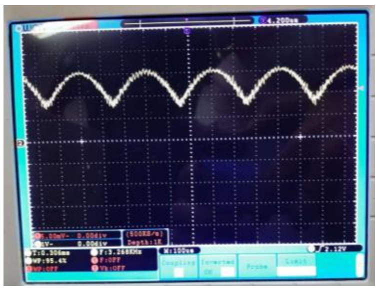


Fig 37. Full-wave rectified

VI. Half-wave rectified

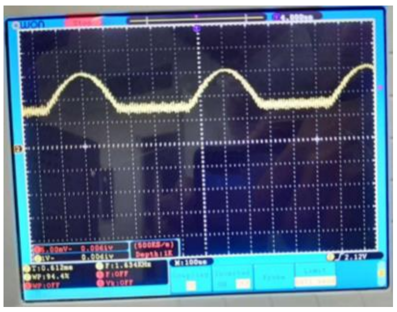


Fig 38. Half-wave rectified

VII. DDS

Based on value of "Phase_ctrl" we have 3 waves:

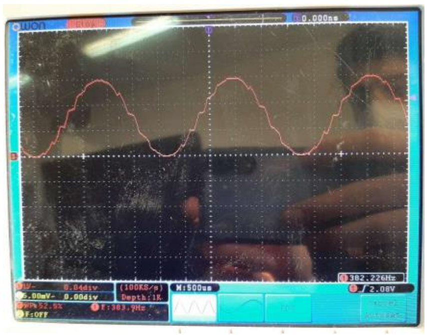


Fig 39. DDS wave with "Phase_ctrl = 1"

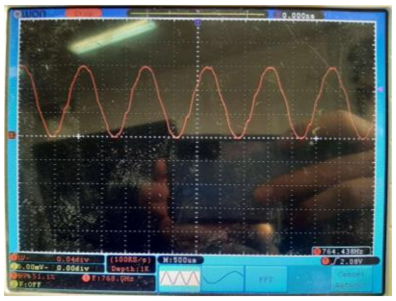


Fig 40. DDS wave with "Phase_ctrl = 2"

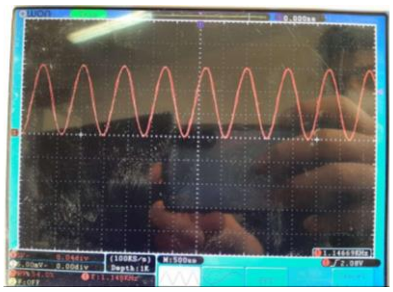


Fig 41. DDS wave with "Phase_ctrl = 3"

As it shown in scope, we can see the relation between "Phase_ctrl" and wave's frequency.

For amplitude selection (via Sine wave) we have:

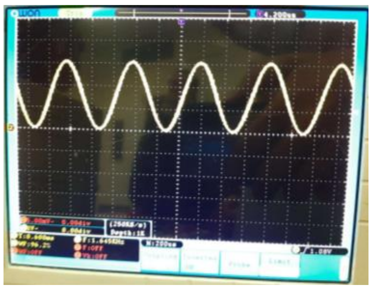


Fig 42. Sine wave, Amplitude = 1.5

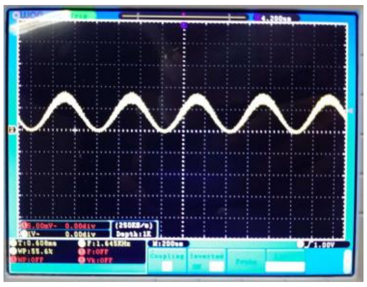


Fig 43. Sine wave, Amplitude = 0.75

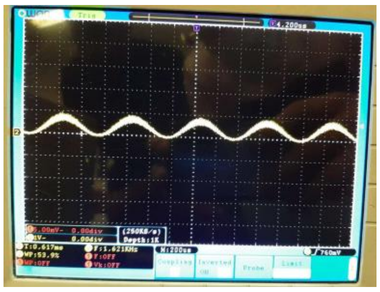


Fig 44. Sine wave, Amplitude = 0.375

For frequency selection (via Triangle wave), we will have:

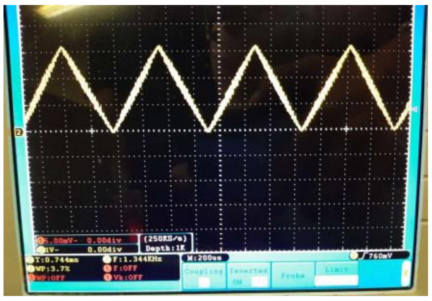


Fig 45. Triangle wave, Frequency = 1.344 KHz

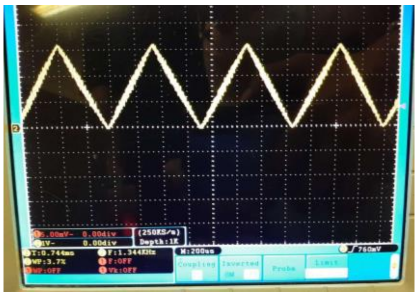


Fig 46. Triangle wave, Frequency = 1.302 KHz

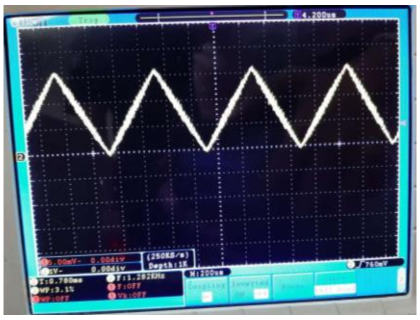


Fig 47. Triangle wave, Frequency = 1.282 KHz

As we see, frequency changes are not very noticeable because we consider "PI" as least significant bits of up-counter parallel load.

II. CONCLUSIONS

In this experiment we designed an arbitrary function generator with different waves and amplitude frequency selection. First provide a Verilog description for module, then synthesizing design and map on FPGA board, and finally convert digital output to analog using PWM and RC circuit to have the waveforms in analog space.

III. REFERENCES

[1] Katayoon Basharkhah and Zahra Jahanpeim and Zain Navabi, *Digital Logic Laboratory*, University of Tehran, Fall 1401.