Experiment #3 - Function Generator

Mohammad Sadegh Aghili- 810100274 / Ali Banihashemi- 810100245

***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.

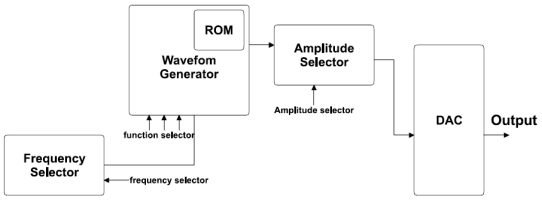

Fig. 1 Block diagram of the Arbitrary Generator (AFG)

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.

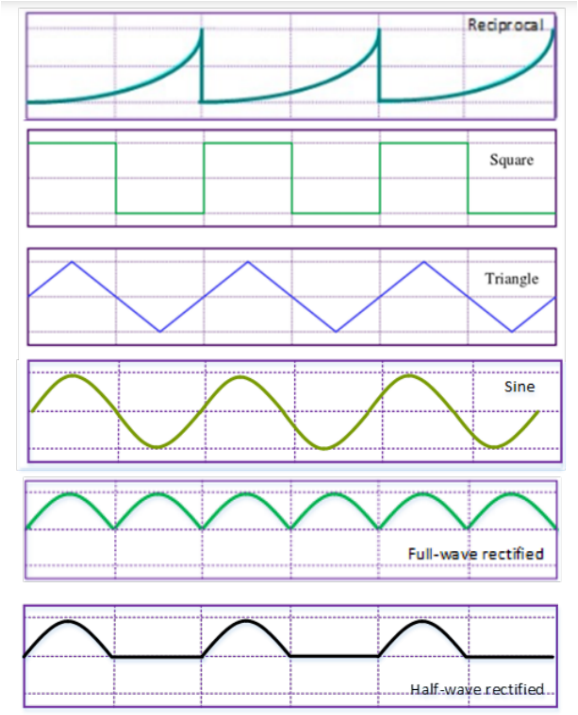

Fig. 2 wave forem generator.

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 descriptions and waveform for square,triangle&recip waves:

```
module waveform(clk,rst,sqr,tra,wave);
  input clk,rst;
  output [7:0] sqr;
  output [7:0] tra;
  output [7:0] wave;
  wire [7:0] count;
  counter8 cnt(clk,rst,count);
  assign sqr = (count>127)?8'b11111111:8'b0;
  assign tra = (count>127)?count:256-count;
  assign wave = 256/(256-count);
endmodule
```

Fig. 3 square,triangle&recip Verilog description.

Fig. 4 wave forms.

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

```
module sinwave(clk,rst,sin,half,full);
  input clk,rst;
  output [7:0] sin;
  output [7:0] half;
  output [7:0] full;
  wire [7:0] count;
  counter8 cnt(clk,rst,count);
  DDS dds(clk,rst,sin);
  assign half = (count<=127)?sin:8'd128;
  assign full = (count<=127)?sin:-sin;
endmodule
```

Fig. 5 sine, full&half waves description.

```
module DDS(clk,rst,out);
  input clk;
  input rst;
  output [7:0] out;
  wire co;
  wire [5:0] count;
  counter6 cnt(clk,rst,co,count);
  wire phase;
  wire sign;
  controller ctrl(clk,rst,co,phase,sign);
  wire [5:0] romin;
  mux6 mux6b(phase,count,~count+1'b1,romin);
  wire [7:0] bnumber;
  ROM rom(romin,bnumber);
  wire [7:0] res;
  mux8 mux8b((~(|count)&phase),8'b11111111,bnumber,res);
  assign out = sign?~(res)-8'd255:(res+7'd128);
endmodule
```

Fig. 6 DDS module.

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

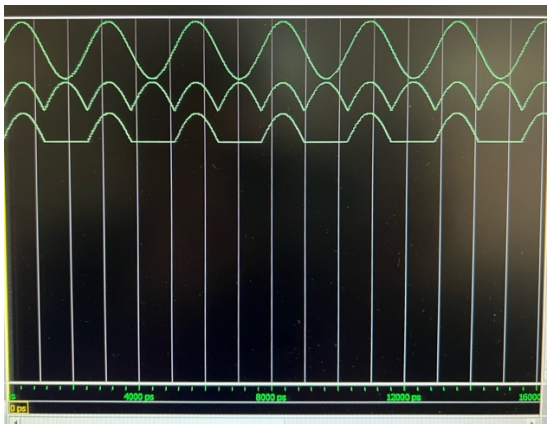

Fig. 7 wave forms.

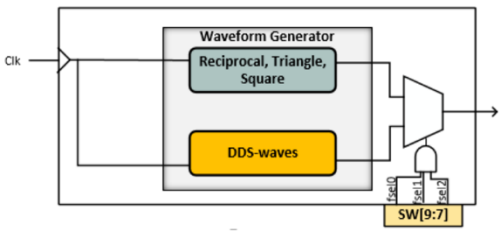

Fig. 8 Block diagram of waveform generator

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. **ROM**

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. 9 ROM command

```
module ROM(adr,read);
  input [5:0] adr;
  output [7:0] read;
  reg [7:0] datastore[0:63];
  (* romstyle = "M9K" *)(* ram_init_file = "sine.mif" *) reg [7:0] rom [63:0];
  reg [7:0] readval;
  always@(adr)
  begin
    readval = datastore[adr];
  end
  assign read = readval;
endmodule
```

Fig. 10 ROM module

IV. 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.**

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

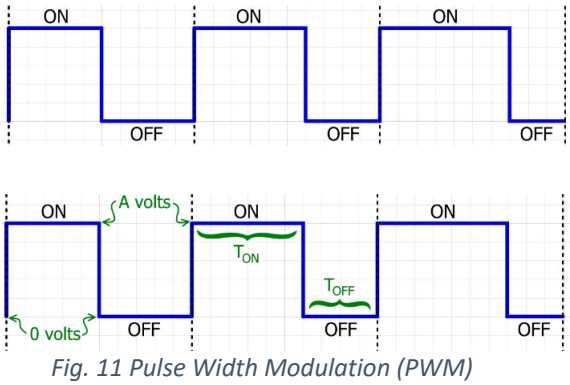

Fig. 11 Pulse Width Modulation (PWM)

Here is the Verilog description of the PWM:

```
module PWM(in,clk,rst,out);
  input [7:0] in;
  input clk;
  input rst;
  output  out;
  wire [7:0] count;
  counter8 cnt(clk,rst,count);
  assign out = (count<in)?1'b1:1'b0;
endmodule
```

Fig. 12 PWM Verilog description.

V. **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.

```
module counter9(sw,clk,rst,ld,co);
  input ld,clk,rst;
  input wire [4:0] sw;
  output co;
  reg [8:0] count;
  reg temp;
  always@(posedge clk, posedge rst)
  begin
      count = rst ? 9'b0:temp?{sw,4'b0}:(count==9'b111111111)?{sw,4'b0}:count+1;
  end
  assign co = count == 9'b111111111;
  wire w2 = &count;
  assign temp = w2|ld ;
endmodule
```

Fig. 13 Frequency selector Verilog description.

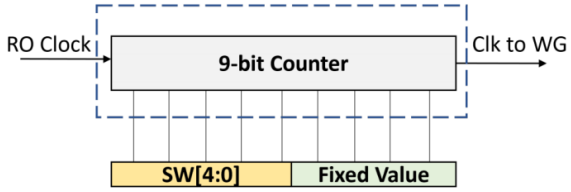

Fig.14 Block diagram of frequency selector

VI. **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 amp( in , sel ,  out);
  input [7:0]in;
  input [1:0] sel;
  output [7:0]out;
  assign out = (sel==2'b00)?in : (sel==2'b01)?in>>1 : (sel==2'

endmodule
```

Fig. 15 Amplitude selector Verilog description.

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.

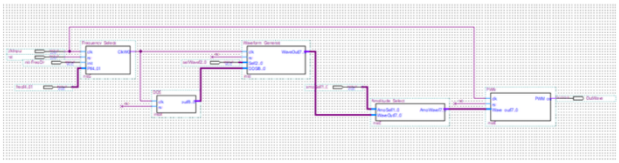

Fig. 16 Final block diagram in Quartus.

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.

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.