Experiment 3 - Function Generator

Shahriar Attar,
810100186,
attarshahriar@gmail.com

Elahe K. Nadrabadi,
810100132,
elahekhodavrdi@gmail.com

***Abstract*— This document is a report for experiment #3 of Digital Logic Design Laboratory at ECE department, University of Tehran. The purpose of this experiment is to design an AFG (Arbitrary Function Generator) to generate different waveforms with different frequencies and amplitudes.**

***Keywords*— Function Generator, Waveform generator, DAC, PWM, Frequency Selector, Amplitude Selector, FPGA Implementation, Altera Cyclone II**

I. Introduction

This experiment is to design an AFG to generate waves that may not be available in real-world signals or we want to replicate them. The generated signals can be used to verify electronic theories, test devices and so forth. We will generate all those waveforms and select which one to show based on the function selector and then we can manipulate it using frequency selector and amplitude selector and then convert digital to analog using PWM, finally we will check our design when we implement it using Quartus. The Block diagram of AFG is shown in Fig. 1.

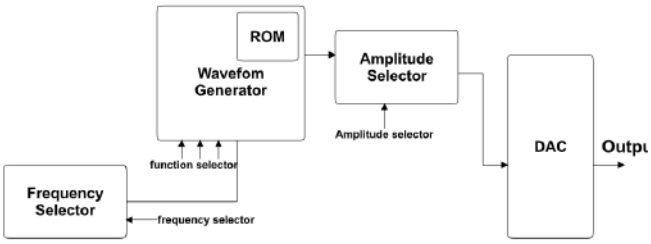

Fig. 1 Block diagram of the Arbitrary Generator (AFG)

II. Waveform Generator

It produces desired functions. Output of this module is an 8-bit digital representing the amplitude of signal. All of which are periodic function and we modify them so all of them has a period of about *256 * clk_cycle.*

A. *The supported functions that we create are*:

- Reciprocal (Fig. 1)
- Square (Fig. 2)
- Triangle (Fig. 3)
- Sine (Fig. 4)
- Full-wave Rectified (Fig. 5)
- Half-wave Rectified (Fig. 6)

For creating the first three waves we use a binary function based on *cnt* input, which is the result stored in our counter. And for the last two waves since they are based on Sine wave we just modified the wave a little bit to get our desirable outcome. Since we cannot use built-in functions for calculating sine wave, we calculate result for some discrete values and then connect them to get a continuous wave, this function is used to calculate sine value in discrete values:

$$sin(n) = sin(n-1) + \alpha.cos(n-1)$$

The α is a constant which helps us to determine the period, since we want to see a whole cycle in *256 * clk_cycle*, so if we solve the equation, α is $\frac{1}{64}$. For calculation cosine value we used:

$$cos(n) = cos(n-1) - \alpha.sin(n)$$

The amplitude range of sine and cosine values that we calculate is between -32768 to 32767. While computing we use 16 bits to have decent accuracy, but since we can only show 8 bits, we just output the 8 MSBs. We used 0 for sin(0) and 30000 for cos(0). The results of sine and cosine operations are signed and between -127 to +128.

However, for simplification and compatibility with other parts of this experiment, we add an offset of 127, making the range of our signal between 0 and 256. We use 3 bits of *SW* which is the input of the board as a function selector as shown in Table. 1.

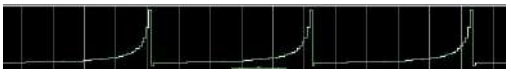

Fig. 2 Reciprocal wave

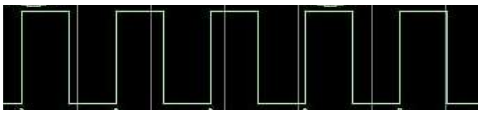

Fig. 3 Square wave

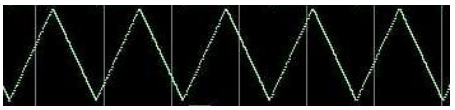

Fig. 4 Triangle wave

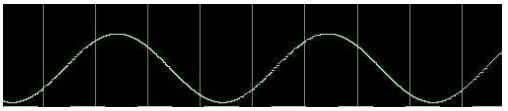

Fig. 5 Sine wave

Fig. 6 Full-wave rectified wave

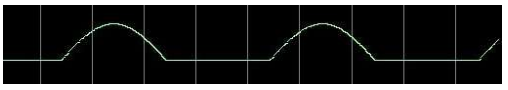

Fig. 7 Half-wave rectified wave

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'b110	DDS

Table. 1 Function Selection

B. DDS

Direct Digital Synthesis (DDS) is a method used to generate signals digitally. It works by using a reference clock signal and a phase accumulator to generate a waveform. The phase accumulator essentially counts the reference clock cycles and uses this count to determine the phase of the output waveform. The phase accumulator is then connected to a lookup table that contains amplitude values for the output waveform. The output waveform is then generated by reading the values from this lookup table. This is the general illustration:

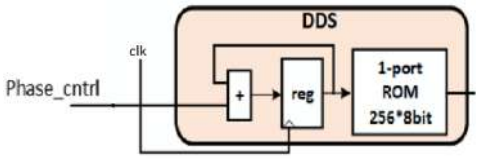

Fig. 8 DDS

We used *sine.mif* file for ROM initialization which has stored the value of a sine wave for several clock cycles. The part which updates the address is commonly known as phase accumulator. We wanted a stable clock frequency so we used the 50-MHz clock frequency of the FPGA board. We use 2 bits of *SW* as *Phase_cntrl*. The ROM memory of DDS was implemented in three different ways.

- Quartus Megawizard Plug-In
- Simple ROM module in Verilog
- ROM module in Verilog but for this, specify for the FPGA to use either FPGA memory blocks or memory logics. like shown in Fig. 9.

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

Fig. 9 Verilog implementation ROM

After creating Megawizard Plug-In and importing the module in the ModelSim we got this output

Fig. 10 Megawizard module output

The Normal module in Verilog created got us this result in Quartus.

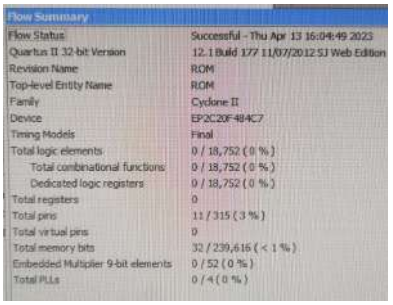

Fig. 11 Synthesis summary of normal ROM

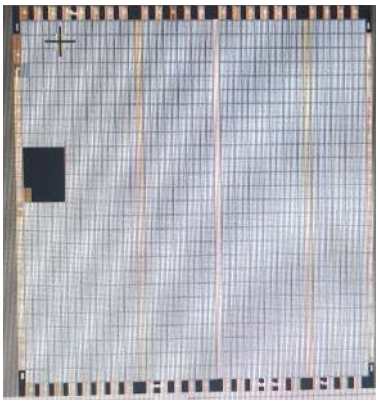

Fig. 12 Chip planner of normal memory

As you can see it didn't assign the chips as we planned and we don't have any logic element, so we specify Quartus to use the memory blocks or memory logics. The results are shown below.

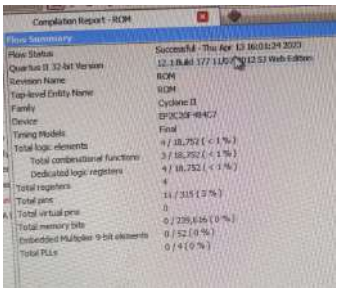

Fig. 13 Synthesis summary of FPGA ROM

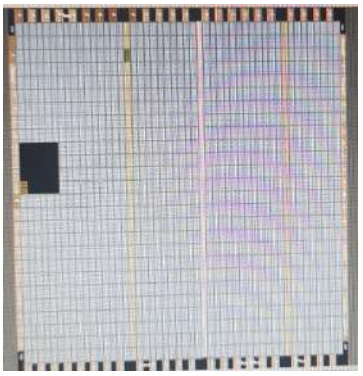

Fig. 14 Chip planner of FPGA memory

So we fixed that problem and now our chip planner has successfully designated parts and each memory and logic block is assigned correctly.

III. DAC Conversion Using PWM

PWM or Pulse-Width Modulation can be used as a DAC or Digital-to-Analog Converter. In simple terms, PWM involves varying a digital signal's duty cycle to generate an analog signal. This process can be useful in applications such as controlling motors or dimming LED lights.

To utilize PWM as a DAC, we need to generate a digital waveform with a specific frequency and duty cycle. The duty cycle will determine the analog output voltage level.

IV. Frequency Selector

To change the frequency of a wave we used a frequency selector which consists of a 9-bit counter, the 4 LSBs are fixed values and the other 5 bits are assigned using *SW*. Also for changing the frequency of the DDS we can change the *phase_cntrl* too. The effect of this module is depicted in Fig. 15 and Fig. 16. The effect on the FPGA board is also shown in Fig. 17 compared to Fig. 18 and Fig. 19 compared to Fig. 20.

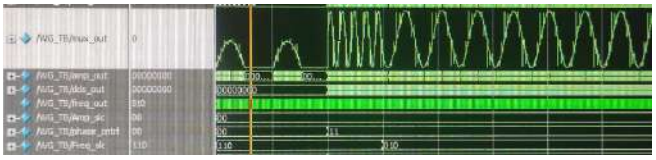

Fig.15 Changing frequency using different methods

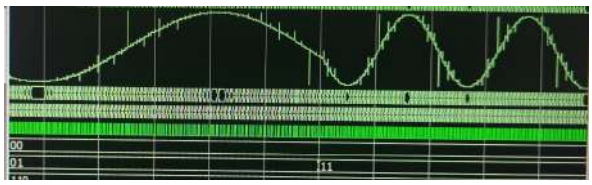

Fig.16 Changing frequency using *phase_cntrl*

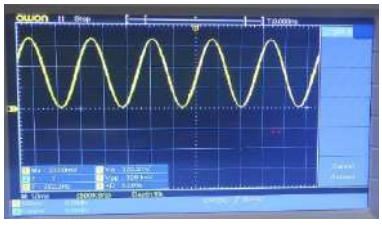

Fig. 17 FPGA sine wave with *phase_cntrl* = 0

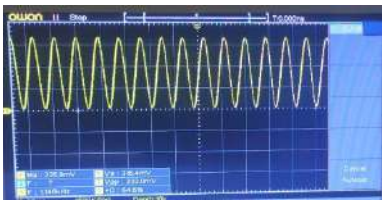

Fig. 18 FPGA sine wave with *phase_cntrl* = 3

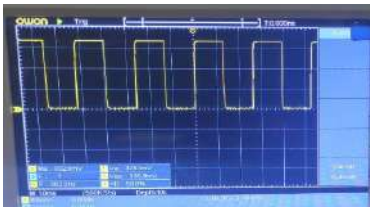

Fig. 19 FPGA square wave with *freq_slc = 000*

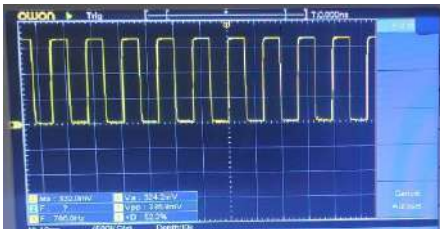

Fig. 19 FPGA square wave with *freq_slc = 100*

V. Amplitude Selector

The task of this module is to scale down the amplitude of the waveforms. This can be done by dividing the output amplitude by a number. The value of the divisor is chosen by a 2-bit input which we get from *SW*. This module divides the amplitude of the output wave by the numbers of following table.

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

Table. 2 Amplitude selection

The difference is shown in Fig. 20 compared with Fig. 21.

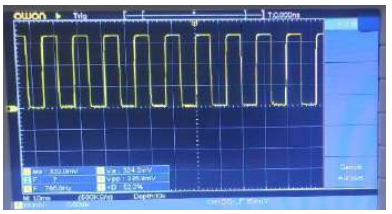

Fig.20 Square wave with *amp_slc* = 00

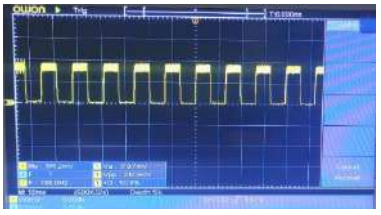

Fig.21 Square wave with *amp_slc* = 10

VI. TOTAL DESIGN

A. *Block Diagram*

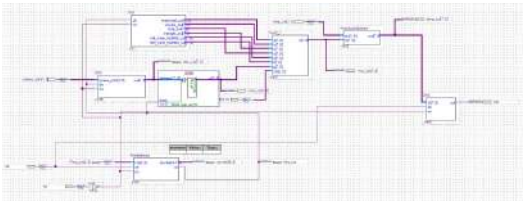

Fig.22 changing frequency using phase_cntrl

B. *FPGA implementation*

The waveforms are shown in the following figures.

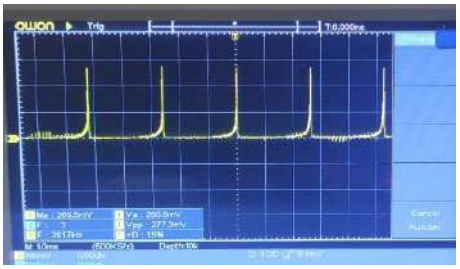

Fig.23 Reciprocal wave

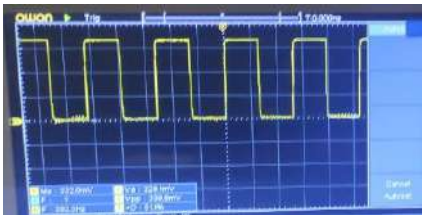

Fig.24 Square wave

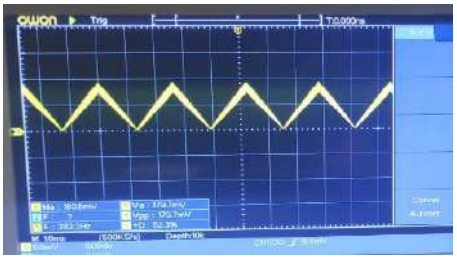

Fig.25 Triangle wave

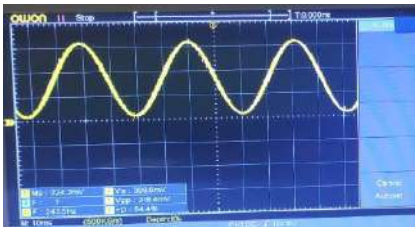

Fig.26 Sine wave

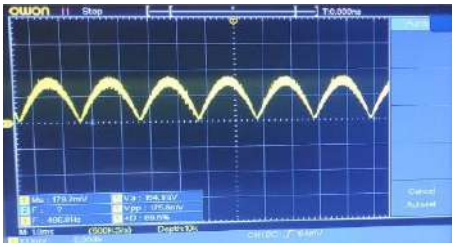

Fig.27 Full-wave rectified wave

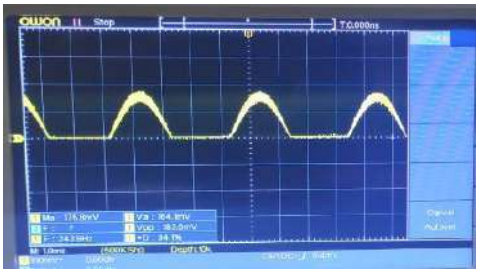

Fig.28 Half-wave rectified wave

VII. CONCLUSIONS

The general ideas discussed in this experiment were:

A. AFG
B. DDS and Memory Implementations
C. Waveform Generator
D. DAC
E. Frequency and Amplitude Selector

ACKNOWLEDGMENT

This report was prepared by Elahe K. Nadrabadi and Shahriar Attar for 5, 6, 7 sessions of DLD Laboratory at ECE department in 1401-1402 spring semester.

REFERENCES

[1] Waveform Generator Implemented in FPGA [Online]. Available: https://www.diva-portal.org/smash/get/diva2:19354/FULLTEXT01.pdf