University of Tehran
College of Engineering
School of Electrical & Computer Engineering

Experiment 3
Sessions 6,7,8

Function Generator

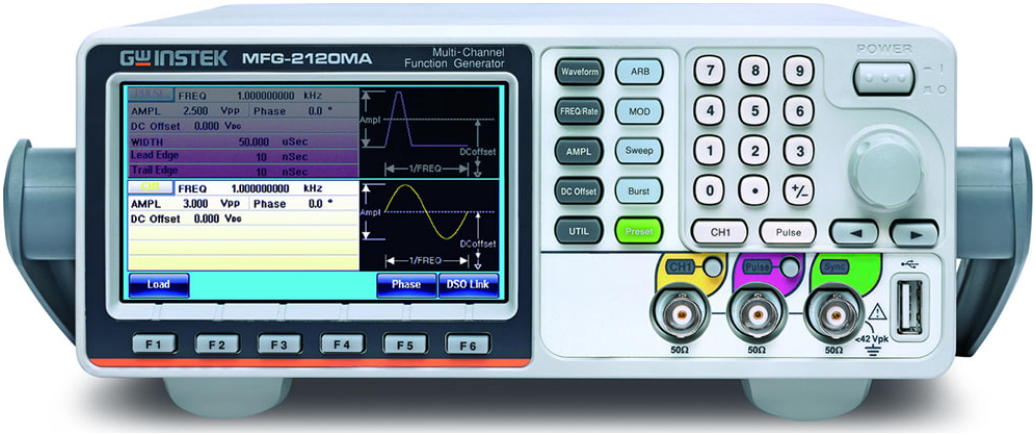
Digital Logic Laboratory
ECE 045
Laboratory Manual

Spring 1403

Contents

Prepared and developed by Katayoon Basharkhah under supervision of Professor Z. Navabi

Figure 1: A single channel AFG from Instek company



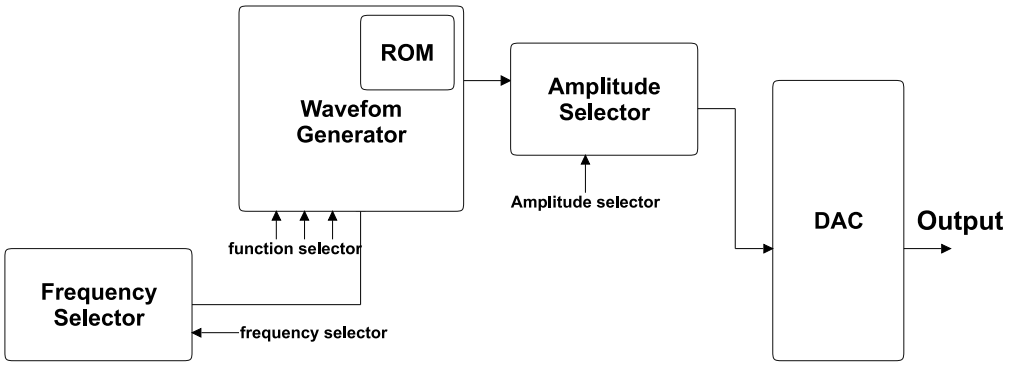
Introduction

An Arbitrary Function Generator (AFG) is an electronic test instrument that generates a wide variety of waveforms with different amplitude and frequency. Among them are sine, square, rhomboid, saw-tooth, and any arbitrary waveform. You 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. figure 1 shows a real AFG from Instek Company.

In this experiment, you are to design an Arbitrary Function Generator that is capable of generating each of the aforementioned waveforms with a wide range for frequency selection.

Figure 2 shows the simplified block diagram of an arbitrary generator. Based on these specifications there is a main component that generates one of the desired waveforms based on the function selectors' value, a frequency selector that sets the output signal frequency, and an amplitude selector. In an AFG a DAC is also used to convert the digital output to an analog signal, which can

Figure 2: Block diagram of the Arbitrary Function Generator (AFG)



be observed and evaluated via an oscilloscope. A ROM memory is usually embedded to store any other arbitrary waveform.

Accordingly, Below are the topics that are explained in the following experiment in detail:

- Waveform Generator
- Frequency Selector
- Amplitude Selector
- Digital to Analog Converter (DAC)

By the end of this experiment, you should have learned:

- The principle of function generators
- Using ROM memories in your design
- Schematic design in Quartus II

1 Waveform Generator

This module is the heart of this project. It produces desired functions. The output of this module is an 8-bit digital representing the amplitude of the signal. The supported functions, shown in figure 3, are sine, square, reciprocal, triangle, full-wave, and half-wave rectified signals.

- Waveforms square, reciprocal, and triangle are based on a counter that counts up or down with each clock for the period of the waveform. The output of the frequency selector is the input clock for this module that determines the discrete incremental values of this signal (resolution).
- Most modern function generators use Direct Digital Synthesis (DDS) for generating their output waveforms. DDS is used for generating arbitrary phase tunable output from a single fixed-frequency reference clock (like an oscillator). The output of the DDS module is a quantized version of the output files (usually a sinusoid). Figure 5 shows the hardware design of the DDS module. To generate the DDS signal, you should use a 1-port ROM memory to store the value of a sine wave for several clock cycles. This ROM module has 64 entries and only contains one quadrant of the sine wave. You will receive a file named `sine.mif` that is used for ROM initialization.
  A phase accumulator module, PA, can generate the address location of this memory. The PA will enumerate all the data point positions of the entire period of the waveform. Two bits of the PA output denoted as "*signbit*" and "*phasepos*", are used to indicate the quadrant information. The next 6 bits of the PA output, denoted as "*addr*", are fed to the ROM address input. Figure 4 shows how 2^{n+2} data points are generated from the 2^n entries stored in the ROM.
  Signed and magnitude numbers are used to represent the sinusoidal signal waveform. The "*phasepos*" bit of the PA output is used to select either "*addr*" or its 2's complementary code as the ROM address. The former case represents the first and third quadrants of the waveform and the latter corresponds to the second and the fourth quadrants. When "$phasepos = 1$"

Figure 3: Different waveforms of function generator

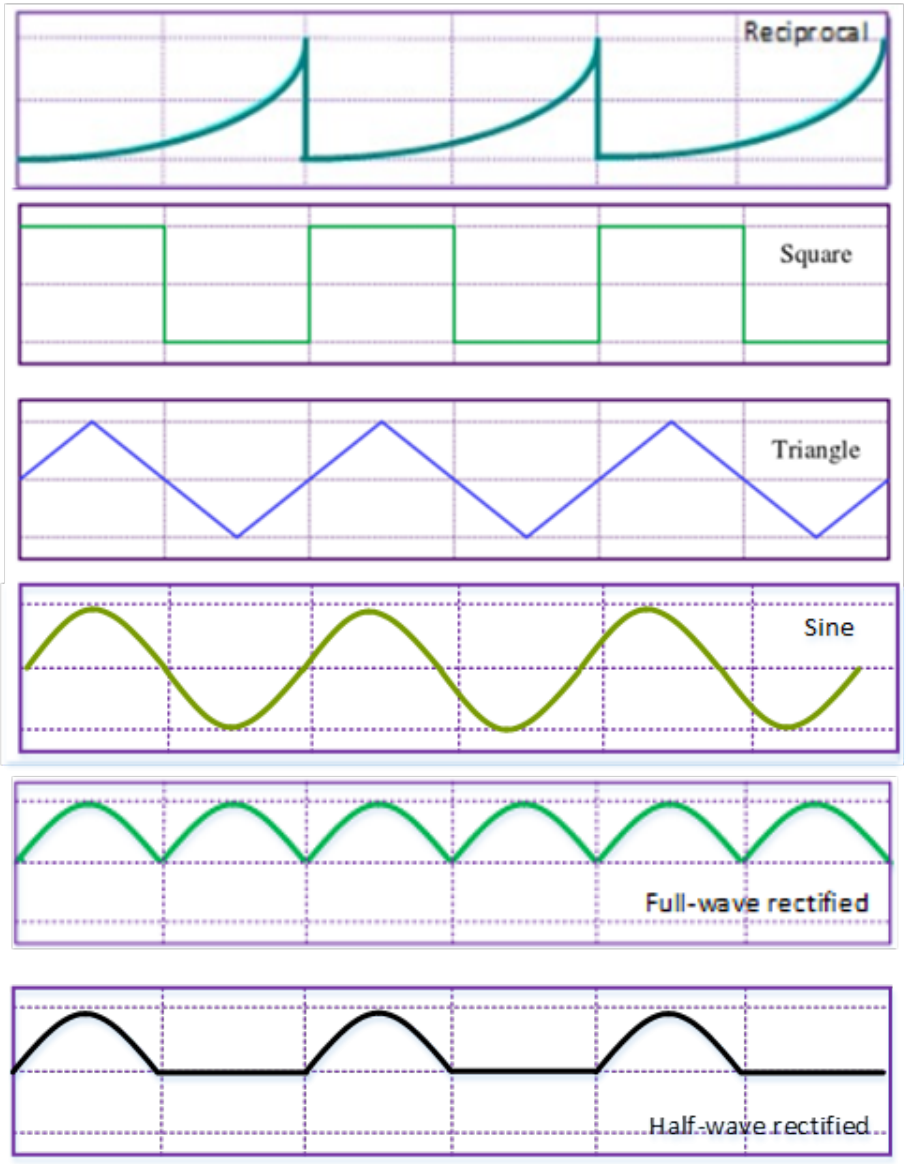
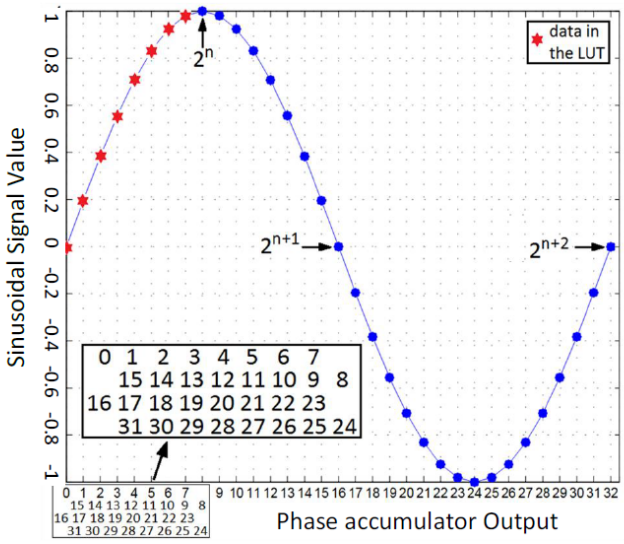


Figure 4: Phase accumulator output generation



and "$addr$ = 0", the corresponding data points are the positive and negative peaks of the sinusoidal waveform, whose magnitude value is not stored in the ROM. The all-zero-detection circuit and the following AND-gate are used to detect such cases. The output of the AND gate guides the output multiplexer to select either the ROM output or the digital code "11..11" as the signal magnitude accordingly.
The clock signal for DDS module should be a stable clock frequency. In this part, the 50-MHz clock frequency of FPGA is used but in the total design, this clock will be changed to the ring oscillator divided clock in the next sections.

- The full-wave and half-wave rectified waveforms are both generated based on the sine wave.

Figure 5: DDS hardware design

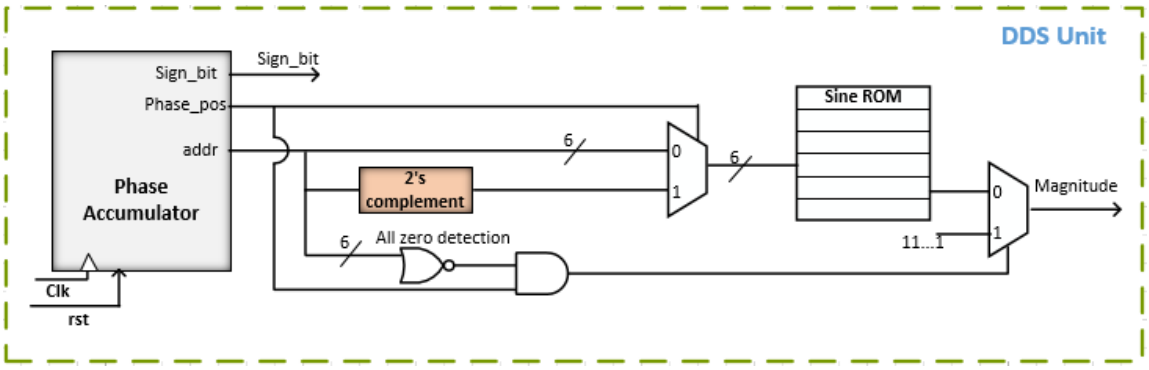


Figure 6: Block diagram of waveform generator

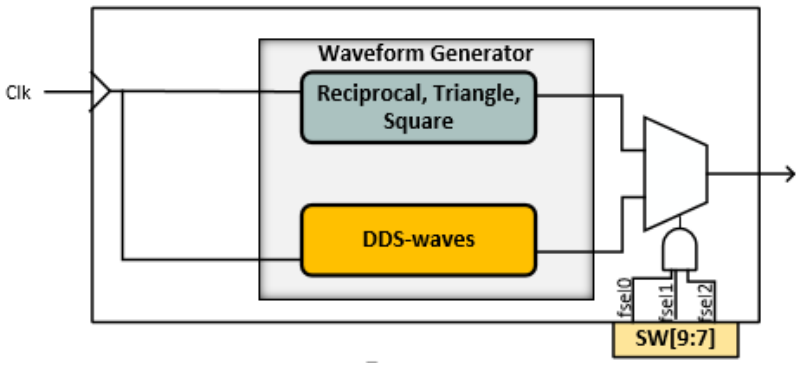


Table 1: Function selection

`func[2:0]`	Function
`3'b000`	Reciprocal
`3'b001`	Square
`3'b010`	Triangle
`3'b100`	DDS-Sine
`3'b101`	Full-wave rectified
`3'b110`	Half-wave rectified

- Now use the DDS sine wave and the counters in a way that generates waveforms similar to figure 3.

  Figure 6 shows the block diagram of the waveform generator module.

Table 1 shows the order of function selection. These are the control signals for the waveform generator module.

1. Write the Verilog code for the Waveform Generator module in figure 6 and use Multiplexer, and other required gates. You will finally get all the components together in a schematic design. For your total design consider a 10-bit input named SW that recalls the switch inputs in FPGA. Dedicate 3 bits of this 10-bit input to the selectors of the waveform generator (SW[9:7]). The remaining bits will be used for the frequency and amplitude selectors.

2. To learn more about the memory element and how they would be synthesized on the FPGA, you are to implement the ROM memory using as following:

   - Write a simple ROM module in Verilog as you have learned in the Digital Logic course.
   - Write the same ROM module in Verilog but specify the memory implementation method this time. For this, you can specify for the FPGA to use either FPGA memory blocks or memory logics. Figure 7 shows the way you can use the romstyle keyword for this purpose.

Figure 7: FPGA memory block specification

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

- For each item, synthesize the ROM memory and report the FPGA resources. Explain the differences based on your experience and the teacher assistant's explanations.

3. After completing the waveform generator block diagram in Quartus, synthesize the final design and include the synthesis summary in your report.

4. Before adding the frequency and amplitude selector, it is better to test your design in Modelsim. Therefore, write a simple testbench and verify the functionality of your design. For this part use the 50-MHz clock frequency. Include the Modelsim results for all the waveforms mentioned in Table 1.

2 Digital to Analog conversion using PWM

There are different methods for digital to analog conversion. You can either use an external chip like an add-on-board card that consists of both ADC and DAC or it can be implemented using Pulse Width Modulation (PWM). Figure 8 shows the DAC circuit. As can be seen for realizing an external DAC chip a serial to parallel interface is required between the FPGA board and the chip. In this experiment, you will use the second method to have a digital to analog conversion. The following is a brief description of how PWM works as a DAC.

A PWM signal is a sequence of periods in which the duration of the logic-high (or logic-low) voltage varies according to external conditions, and these variations can be used to transmit information. The PWM carrier frequency is constant, so the active and inactive state duration increases or decreases and vice versa. The duty cycle of the PWM signal is equal to:

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

The nominal DAC voltage observed at the output of the low-pass filter is determined by just two parameters, namely, the duty cycle and the PWM signal's logic-high voltage; in the figure 9, A denotes this logic-high voltage for "amplitude." The relationship between duty cycle, amplitude, and nominal DAC voltage is fairly intuitive: In the frequency domain, a low-pass filter suppresses higher-frequency components of an input signal. The time-domain equivalent of this effect is smoothing, or averaging. Thus, by low-pass filtering a PWM signal, we are extracting its average value.

In this experiment, the period of PWM is fixed to 256 clocks and its pulse width is the value on the 8-bit input of the module. Figure 9 shows a sample PWM wave. For the first PW number of clock cycles, the output value is 1 and it is 0 for the rest of the cycles.

1. Write a Verilog code for the DAC module. The output of the PWM module will go to an external board with an RC low pass filter. The values of the resistor and capacitor are 1K ohm and 10 nF respectively.

2. You can use random data input from the switches to test the accuracy of your design.

Figure 8: Block diagram of Digital to Analog Converter

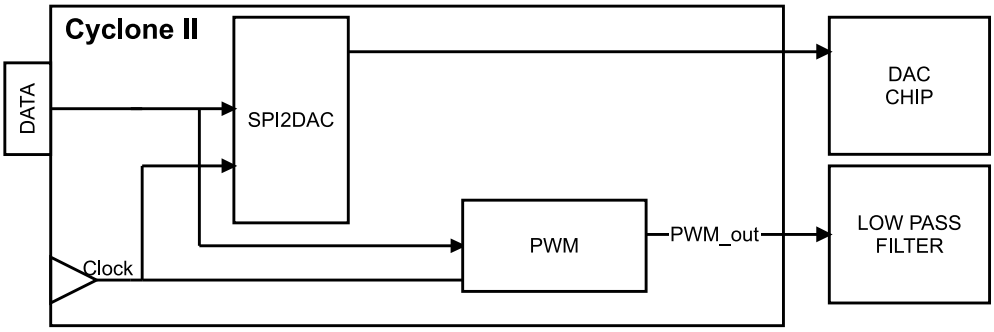


Figure 9: Pulse Width Modulation (PWM)

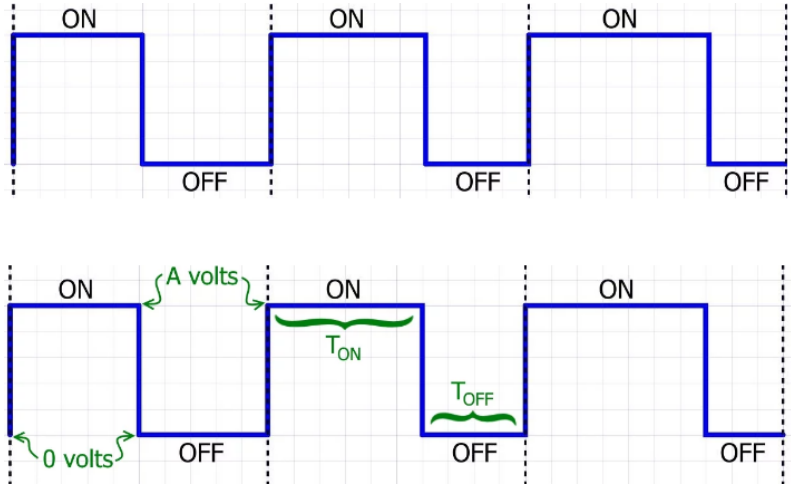
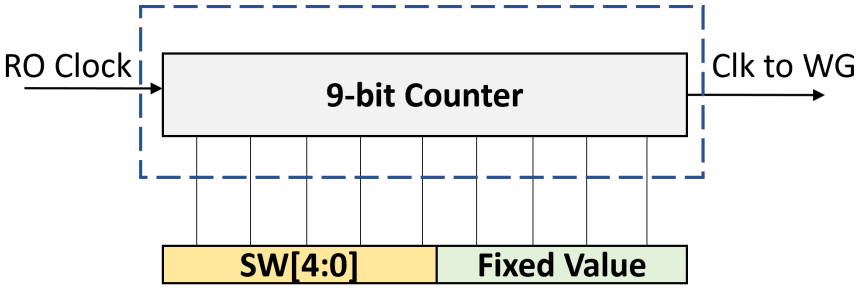


Figure 10: Block diagram of frequency selector



3. Connect the PWM module to the design in the previous section, synthesize the design, and program the FPGA.

4. Observe the functions on the Oscilloscope and include them in your reports.

3 Frequency Selector

In order to set the frequency of the output signal a frequency selector is required. The frequency selector consists of a counter that divides a high source input signal to the desired value. For this purpose, you can write the Verilog description of a simple 9-bit counter like figure 10. For loading the division load value, you need to use an external load signal. Use one of the FPGA push buttons for this purpose.

1. After adding the clock divider to the waveform generator design, synthesize your design.

2. Set the 4 least significant bits of the counter to a fixed value in your code and set the remaining 5 bits as the input pins of the counter. Use SW[4:0] for parallel loads of the divider. Use one push button (KEY0) for the frequency divider load.

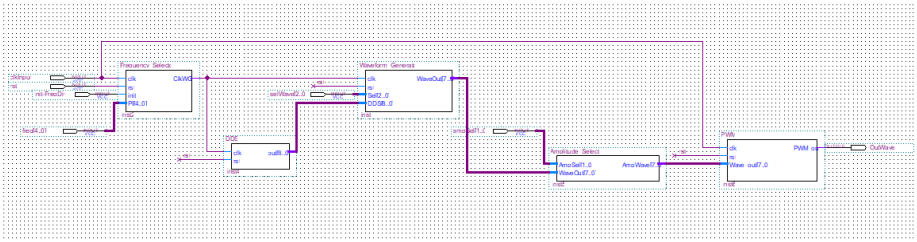
3. Modify the testbench of section one so that you can verify the design for different desired frequencies. To do this, change the parallel loads (SW[4:0]) for at least three different frequencies.

4. Include the waveforms of each desired frequency in your report while mentioning the achieved frequencies, the input parallel loads, and expected frequencies.

4 Amplitude Selector

One option in the function generator is the amplitude of the generated wave. 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. Dedicate two bits of input SW (SW[6:5]) to these selector inputs. The type of amplitude is selected by SW[6:5] according to table 2.

Figure 11: Final block diagram in Quartus II



1. After adding the amplitude selector module to the previous design, synthesize your design.
2. Modify the testbench of section two so that you can verify the design for different amplitudes. To do this, change SW[6:5] in your testbench.
3. Include the waveforms of each desired amplitude in your report.

Finally, your top-level design must look like the block diagram of figure 11.

5 The Total design

When all the components are designed and verified, you can put them together in a schematic design. Quartus II has this capability to generate symbols for any HDL code with the corresponding inputs and outputs.

1. Click on *File* ▷ *New Project Wizard*.
2. Create a good directory for your project and complete the form as in the previous experiments.
3. In the *ADD/Remove Files in Project* page add all the Verilog code you have written.
4. By right-clicking on a Verilog code, you can create a symbol for it. If the symbol generation is done successfully, a file with the suffix `.bdf` will be generated.

You should use bus or line tool from the top toolbar menu ┐┐┐\\\□ and make all the interconnects between components.

Double-click on the blank space in the Graphic Editor window, or click on the icon in the toolbar that looks like an `AND` gate. A pop-up box will appear.

Table 2: Amplitude selection

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

Expand the hierarchy in the Libraries box. First, expand libraries, then expand the library primitives, followed by expanding the library logic comprising the logic gates. Use the gates if any is needed. When the schematic design is completed, compile it as before. Program the design on FPGA and record all the results.

Deliverables

The report must be written based on the given template.

A- Waveform Generator

1. Present the simulation outcomes for all waveforms listed in Table 1.
2. Compare the FPGA resource utilization between the ROM with the romstyle keyword and without it.
3. Provide the synthesis report for the Waveform Generator's design.

B- PWM (Pulse Width Modulation)

1. Briefly explain the operation of the PWM.
2. Show the simulation results for three data inputs and evaluate the precision of your PWM design.

C- Frequency Selector

1. Show the simulation results for three selected frequencies.

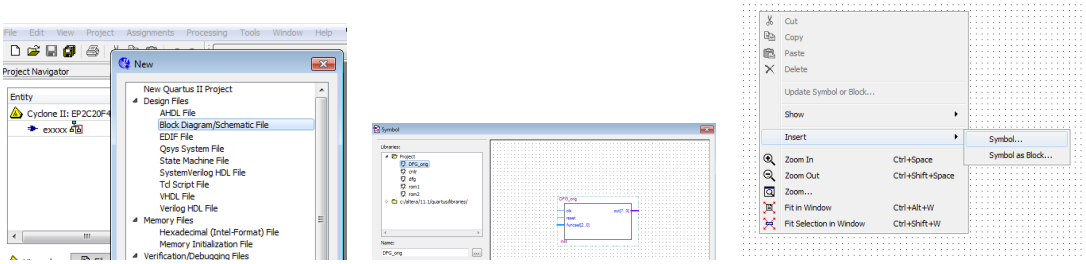
D- Amplitude Selector

1. Confirm the accuracy of your design across three distinct amplitude levels for all waveforms using Modelsim.

E- Implementation

1. Exhibit the schematic diagram of the Function Generator within Quartus.
2. Demonstrate the Oscilloscope visualization of different waveforms at two distinct amplitude and frequency settings.

Figure 12: Quartus II



Acknowledgment

This lab manual was prepared and developed by Katayoon Basharkhah, Ph.D. student of Digital Systems at the University of Tehran, under the supervision of Professor Zain Navabi.

This manual has been revised and edited by Zahra Jahanpeima, Ph.D. student of Digital Systems at the University of Tehran, Iman Rasouli, and Zahra Hojati undergraduate students of Electrical Engineering at the University of Tehran.