LABORATORY MANUAL

Experiment 3
Function Generator

UNIVERSITY OF TEHRAN

School of Electrical and Computer Engineering

Digital Logic Laboratory
ECE 045

Fall 1396

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

EXPERIMENT 3 (7, 8)

Function Generator

INTRODUCTION

A 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.
In this experiment, you are to design an arbitrary/function generator that is capable of generating each of the aforementioned waveforms with wide range for frequency selection. You will make use of the frequency regulator that you have designed in expeiment2.

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

- The principle of function generators
- PWM Digital to analog converters
- Using ROM memories in your design
- Schematic design in QuartusII

Some specifications are considered for a function generator: The bandwidth that is the frequency range of the source signal, the sample rate that is the maximum clock rate of the source signal and the vertical resolution. The smallest voltage increment that can be programmed in a signal source. Resolution is the data word width, in bits, of the instrument's DAC. Figure 1 shows the simplified block diagram of an arbitrary/function generator. Based on these specifications there is a main component that generates one of the desired waveform based on the function selectors' value, a frequency selector that sets the output signal frequency, an amplitude selector and a DAC that converts the digital output to an analog signal, which can be observed and evaluated via an oscilloscope. A ROM memory is usually embedded to store any other arbitrary waveform. Accordingly, Below is the topics that are explained in the following of this experiment in details:

- Frequency Selector
- Waveform Generator
- Amplitude Selector
- DAC

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

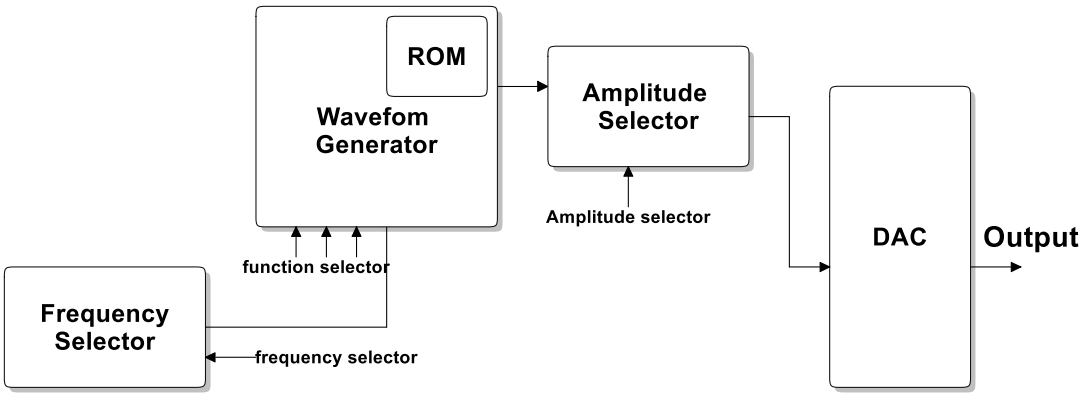


Figure1: block diagram of the function generator

PART I
Digital to Analog conversion using PWM

There are different methods for digital to analog conversion. You can either use an external chip like using an add-on-board card that consists of both ADC and DAC or it can be implemented using pulse width modulation. Figure 2 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.

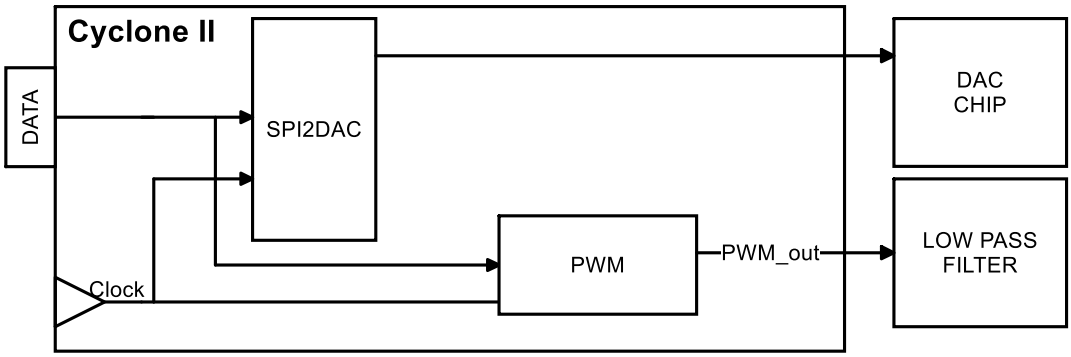


Figure 2: Block diagram of Waveform generator

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 increase r decrease vice versa. The duty cycle of the PWM signal is equal to:

$$dutycycle = \frac{Ton}{Ton + Toff}$$

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

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 diagram, 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.

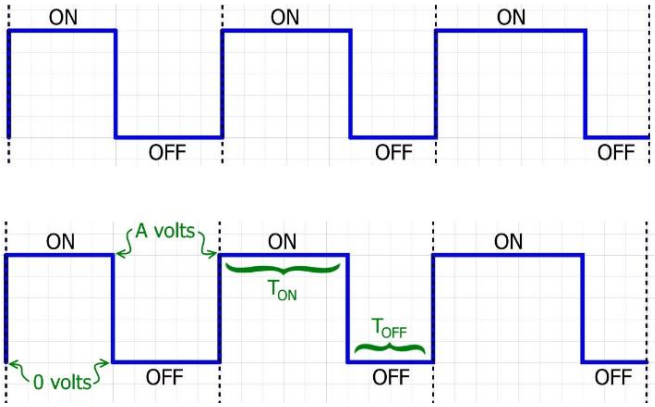


Figure 3: Pulse Width Modulation

Period of PWM has to be small enough so the effect of on/off switching be unnoticeable on load. Pulse Width (PW) determines amount of power that is fed to load. In this experiment period of PWM is fixed to 256 clocks and its pulse width is the value on 8-bit input of module. Figure 2 shows sample PWM waves. In each cycle, first 'PW' clock output value is '1' and it is '0' for rest of cycle.

- Write a Verilog code for DAC module. The output of the PWM module will go to an external board with an RC low pass filter.
- You can use a random data input from the switches to test the accuracy of your design.

Design Verification:

- ✓ **Demonstrate the Digital to Analog converted signal to your TA.**
- ✓ **The TA should verify the accuracy of your results.**
- ✓ **Have your TA sign the part of your Lab-book corresponding to work you did for this part.**

PART II
Frequency Selector

In order to set the frequency of the output signal, a frequency selector (or regulated frequency) is required. The frequency selector consists of a counter that divides a high source input signal to the desired value. You can take advantage of your previous design of the frequency regulator. The Block diagram of this component is shown in Figure 4.

There are several options for constructing this circuit, two of which are described below. Alternatively a VCO can be used for this paper.

1- You can construct the circuit as shown in Figure 2 inside the FPGA using the 50MHz input clock of the FPGA as the source signal. Use a counters to divide the frequency of input clock. Set the switches of the FPGA as the parallel load required for the desired frequency. Write the Verilog code for this part
2- As an optional, you can use the frequency divider of Experiment 1 that implements a ring oscillator as the source signal. In this case, the frequency selector of your function generator will be an external circuit. In this case, if you take this option use dip switches to turn the parallel load inputs on and off.

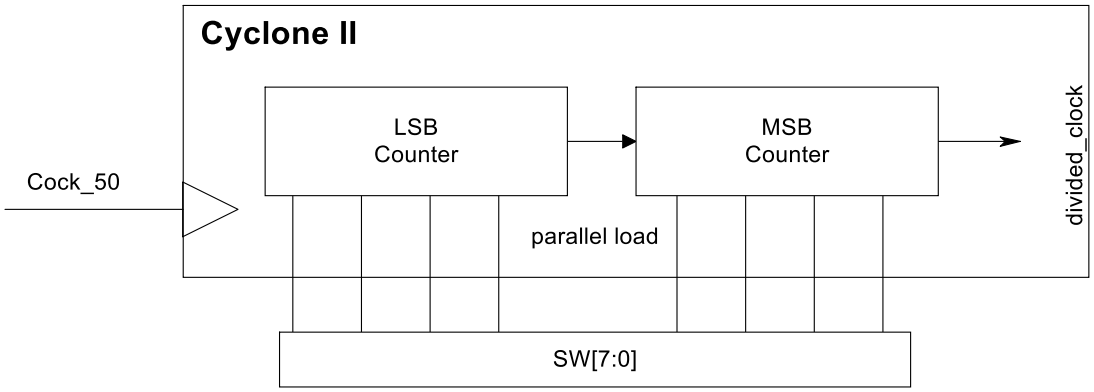


Figure 4: Block diagram of frequency selector

Design Verification:

- ✓ **Demonstrate the frequency divider to your TA.**
- ✓ **The TA should verify the accuracy of your results.**
- ✓ **Have your TA sign the part of your Lab-book corresponding to work you did for this part.**

PART III
Waveform Generator

This module is the heart of this project. It produces desired functions. Functions generated by this module have the fixed period of 256 clocks. Output of this module is an 8-bit digital representing the amplitude of signal. It should be noted that due to characteristics of implemented DAC, the valid range on output is between Vcc and Gnd. The supported functions (shown in Figure 5) are rhomboid, sine, square, triangle and saw-tooth. Waveforms for the mentioned functions are.

A) As shown, waveforms, rhomboid, square 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 for this module that determines the discrete incremental values of this signal (resolution).

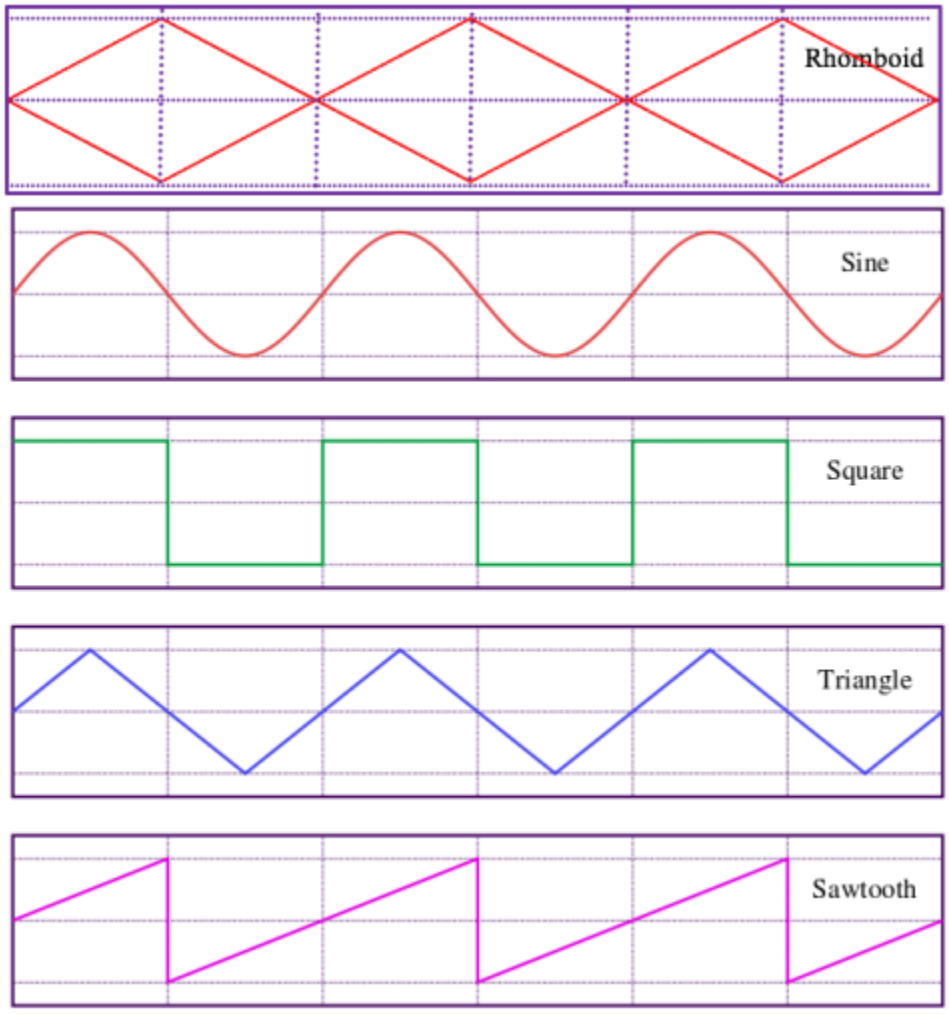


Figure 5: Different waveforms of function generator

Design Verification:

- ✓ **Demonstrate the result of part A to your TA.**
- ✓ **The TA should verify the accuracy of your results.**
- ✓ **Have your TA sign the part of your Lab-book corresponding to work you did for this part.**

B) Generation of Sine function can be somewhat different. The following second order differential equation can be used to generate the Sine function:

$$sin(n) = sin(n-1) + a\,cos(n-1)$$
$$\cos(n) = \cos(n-1) - a\sin(n)$$

In order to do the mathematical operations with reasonable accuracy, operations are done in 16-bit fixed point. Also considering the period of about 256 clock cycles from frequency selector the equations turn to:

$$sin(n) = sin(n-1) + \frac{1}{64}\cos(n-1)$$
$$\cos(n) = \cos(n-1) + \frac{1}{64}\sin(n)$$

Assuming value are between -32768 to 32767 for Sin and Cos

Initialization of first values in differential equations is necessary (Use '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 compatability with other parts of this experiment, we add an offset of 127 making the range of our signal between 0 and 256.

Design Verification:

- ✓ **Demonstrate the results of part B to your TA.**
- ✓ **The TA should verify the accuracy of your results.**
- ✓ **Have your TA sign the part of your Lab-book corresponding to work you did for this part.**

C) To generate the arbitrary function, you should use a 1-port ROM memory to store the value of the desired signal at each increment inside the memory addresses. You will receive a file named "arbitray.mif" that is used for ROM initialization. Refer to the Appendix A to learn about ROM memories and the way they are generated and initialized through MegaWizard in Quartus II. Figure 6 shows the block diagram of the waveform generator module.

Design Verification:

- ✓ **Demonstrate the results of part C to your TA.**
- ✓ **The TA should verify the accuracy of your results.**
- ✓ **Have your TA sign the part of your Lab-book corresponding to work you did for this part.**

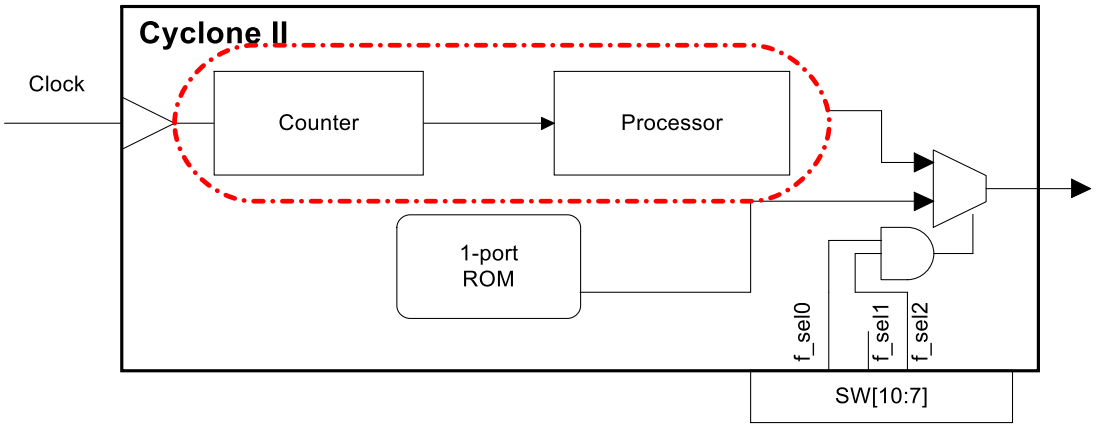


Figure 6: Block diagram of Waveform generator

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

Table1: Function selection

Func[2:0]	Function
3'b000	rhomboid
3'b001	Sine
3'b010	Square
3'b011	Triangle
3'b100	Sawtooth
3'b101	Arbitrary

Write the Verilog code for the red box and use LPM or Megawizard functions for ROM memory or Multiplexers. You will finally get all the components together in a schematic design.

PART IV

Amplitude Selector

One option in function generator is the amplitude of generated wave. This can be done by adding a module between digital function generator and DAC. Task of this module is to divide the input taken from digital function generator and divide it by a desired value and feed the result to DAC. Value of divisor is chosen by a 2-bit input.

Type of amplitude is selected by a 2-bit input according to Table 2.

Table 2: Amplitude selection

Func[1:0]	Amplitude
2'b00	1
2'b01	2
2'b10	4
2'b11	8

Design Verification:

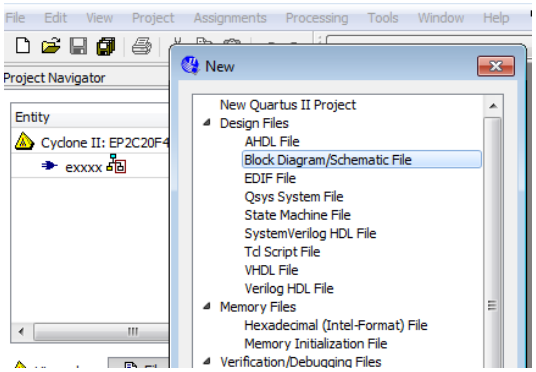
- ✓ **Demonstrate the generated amplitudes to your TA on the oscilloscope.**
- ✓ **The TA should verify the accuracy of your results.**
- ✓ **Take the TA signature for this part in your Lab-book**

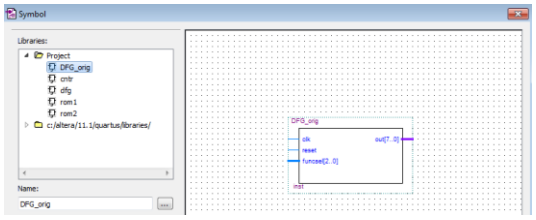
PART V

The Total design

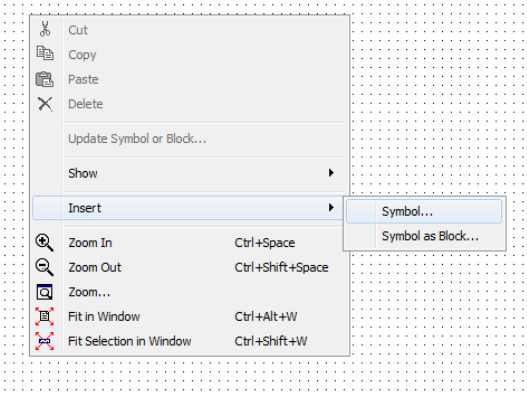
When all the components are designed and verified you can put them together in a schematic design. QuartusII has this capability to generate symbols for any HDL code with the corresponding inputs and outputs.

1- Click on **File> New project wizard**
2- Create a good directory for your project and complete the form as the previous experiments.
3- In the "**ADD/ Remove files in project" page** add all the Verilog code you have written.



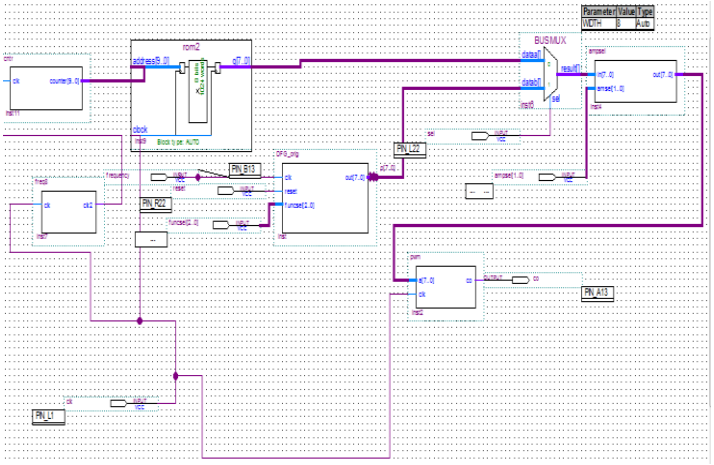


4- By right click on a Verilog code, you can create a symbol for it. If the symbol generation is done successfully, a file with suffix " .bdf" will be generated.



,

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 hat looks like an AND gate. A pop-up box will appear.
Expand the hierarchy in the Libraries box as shown in the figure. First expand libraries, then expand the library primitives, followed by expanding the library logic which comprisesthe 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.

Design Verification:

- ✓ **Demonstrate the final output to your TA on the oscilloscope.**
- ✓ **The TA should verify the accuracy of your results.**
- ✓ **Take the TA signature for this part in your Lab-book**

PRELAB ASSIGNMENT

Before coming to the lab, answer these questions. The pre-lab needs to be handed in at the start of the lab.

1- Write a preliminary Verilog code for generating the rhomboid and the sine wave generation.
2- Draw the datapath of the complete design including all the input and output signals for each component and also any control signal that you used.