



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

Experiment 3
Sessions 5,6
Function Generator

Digital Logic Laboratory
ECE 045
Laboratory Manual

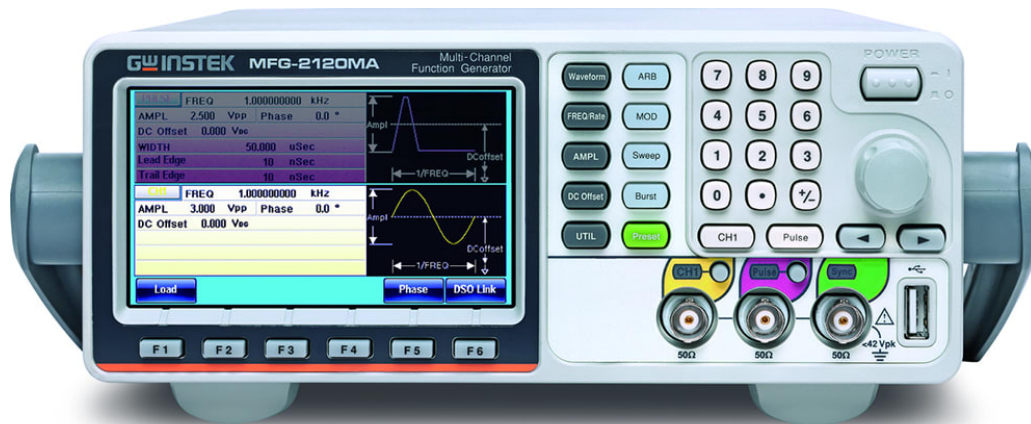
Fall 1401



Contents

Contents	1
Introduction	2
1 Waveform Generator	3
2 Digital to Analog conversion using PWM	6
3 Frequency Selector	8
4 Amplitude Selector	8
5 The Total design	9
Acknowledgment	10

Figure 1: A single channel AFG from Instek company



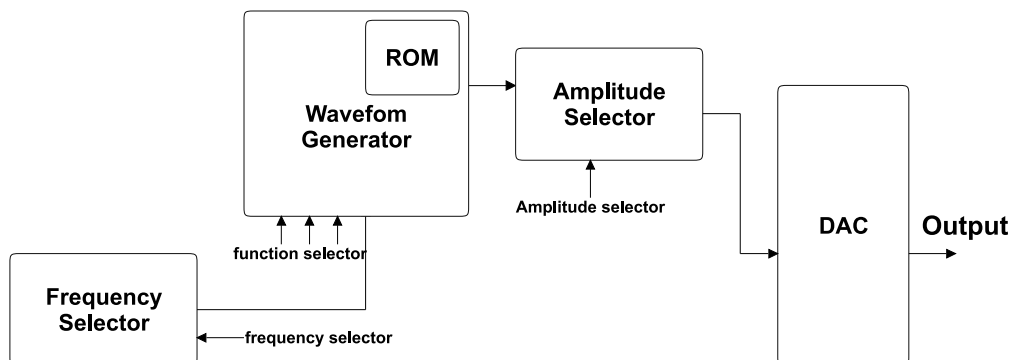
Introduction

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 Generator that is capable of generating each of the aforementioned waveforms with 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 waveform based on the function selectors' value, a frequency selector that sets the output signal frequency, an amplitude selector. In an AFG a DAC is also used to convert the digital output to an analog signal, which can be

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



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:

- Waveform Generator
- Frequency Selector
- Amplitude Selector

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. Output of this module is an 8-bit digital representing the amplitude of 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).
- The full-wave and half-wave rectified waveforms are both generated based on the sine wave. So before generating these waves, you first need to generate a sine wave. The following second order differential equation can be used to generate the sine function:

$$\begin{aligned}\sin(n) &= \sin(n-1) + a.\cos(n-1) \\ \cos(n) &= \cos(n-1) - a.\sin(n)\end{aligned}$$

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:

$$\begin{aligned}\sin(n) &= \sin(n-1) + \frac{1}{64}.\cos(n-1) \\ \cos(n) &= \cos(n-1) - \frac{1}{64}.\sin(n)\end{aligned}$$

Assuming values 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 compatibility with other parts of this experiment, we add an offset of 127, making the range of our signal between 0 and 256.

Figure 3: Different waveforms of function generator

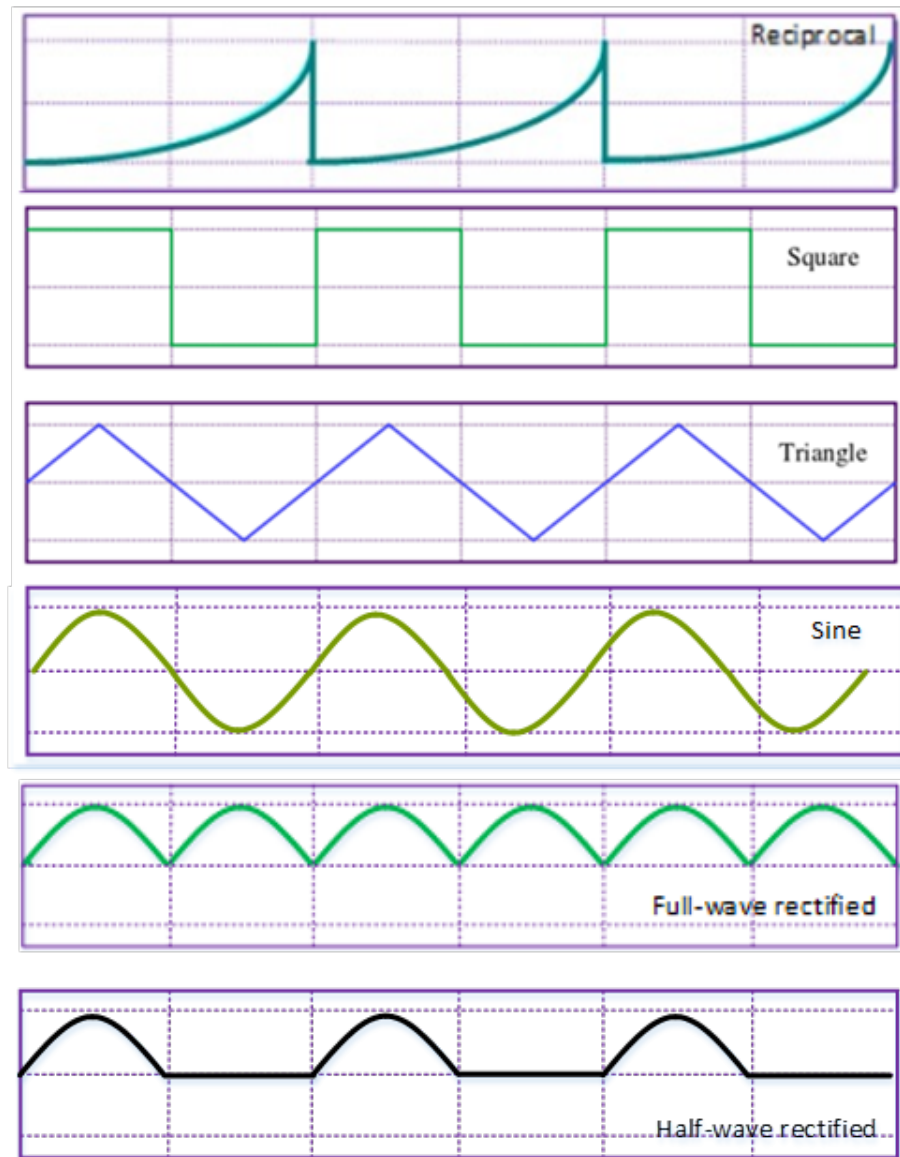


Figure 4: Block diagram of waveform generator

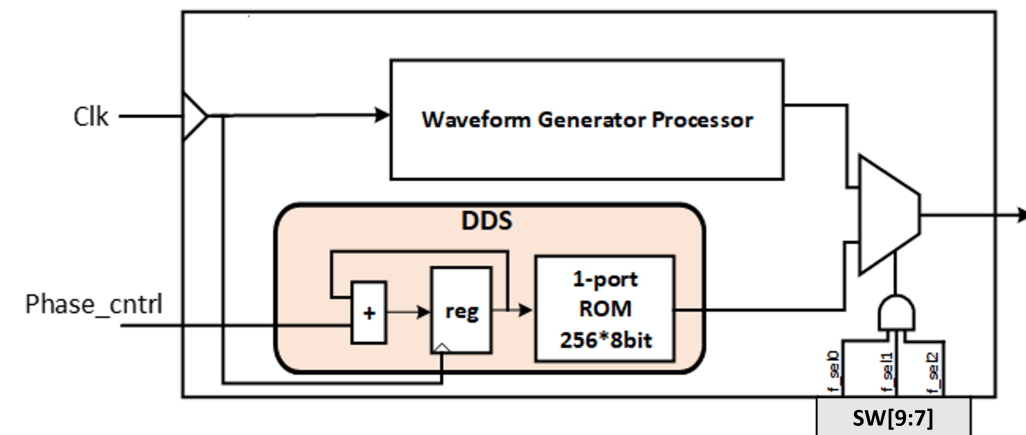


Table 1: Function selection

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

- Now use the sine wave and the counters in a way that generates waveforms similar to figure 3.
- 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 DDS module is a quantized version of the output files (usually a sinusoid). The period of this signal is controlled by a Phase control value. 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. You will receive a file named **sine.mif** that is used for ROM initialization. A register and adder can generate the address location of this memory. This is done by incrementing the phase of the signal each time by the value of the **Phase_cntrl**. This module is usually called phase accumulator. 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 next sections.

Figure 4 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 Processor in figure 4 and use LPM or Megawizard functions for ROM memory, 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. After completing the waveform generator block diagram in Quartus, synthesize the final design and include the synthesis summary in your report.
3. 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. Set the value of Phase_cntrl to 1 for this part.

2 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 (PWM). Figure 5 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 increase or decrease 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 6, 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, period of PWM is fixed to 256 clocks and its pulse width is the value on 8-bit input of module. Figure 6 shows sample PWM waves. In each cycle, first PW clock output value is 1 and it is 0 for rest of cycle.

1. 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.
2. You can use a random data input from the switches to test the accuracy of your design.
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.

Figure 5: Block diagram of Digital to Analog Converter

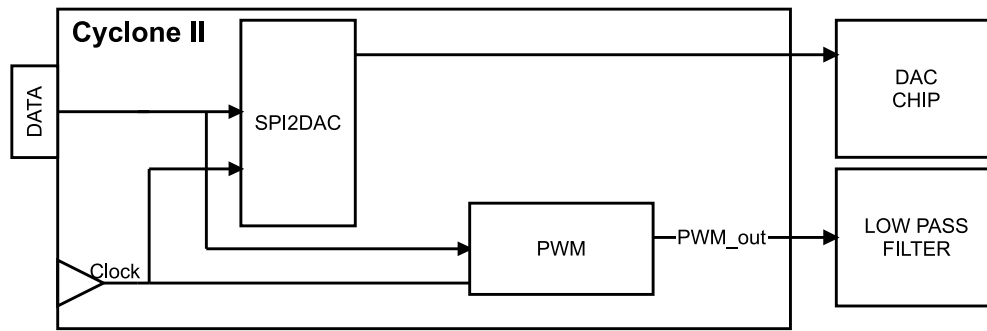


Figure 6: Pulse Width Modulation (PWM)

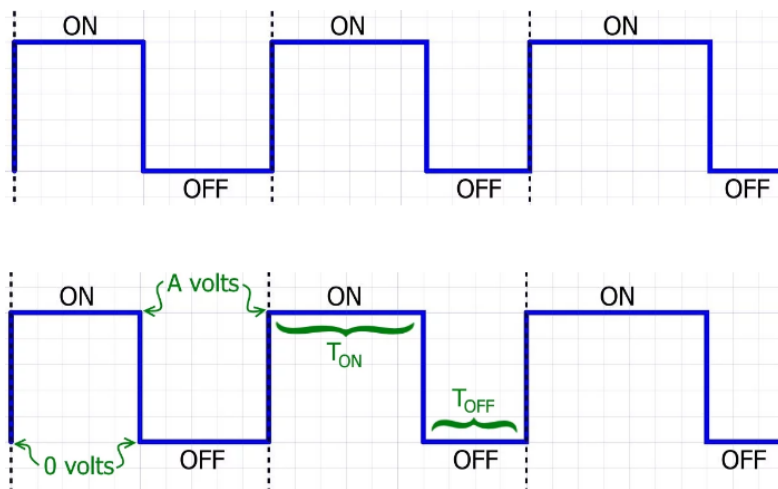
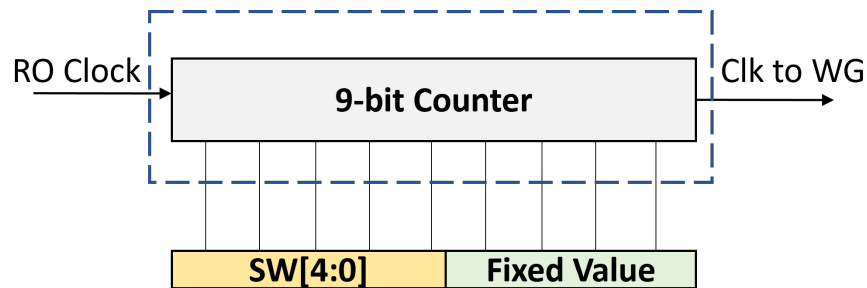


Figure 7: Block diagram of frequency selector



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 7.

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 this parallel loads of the divider.
3. Modify the testbench of section one so that you can verify the design for different desired frequency. 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 with mentioning the achieved frequencies, the input parallel loads and expected frequencies.
5. As mentioned the DDS module can generate sine wave with different phases based on the `Phase_cntrl` input. As a different way of frequency selection verify your design for at least 3 values of `Phase_cntrl` and observe the frequency change in the waveforms. Include the waveforms in your report and explain about the expected and achieved frequency.

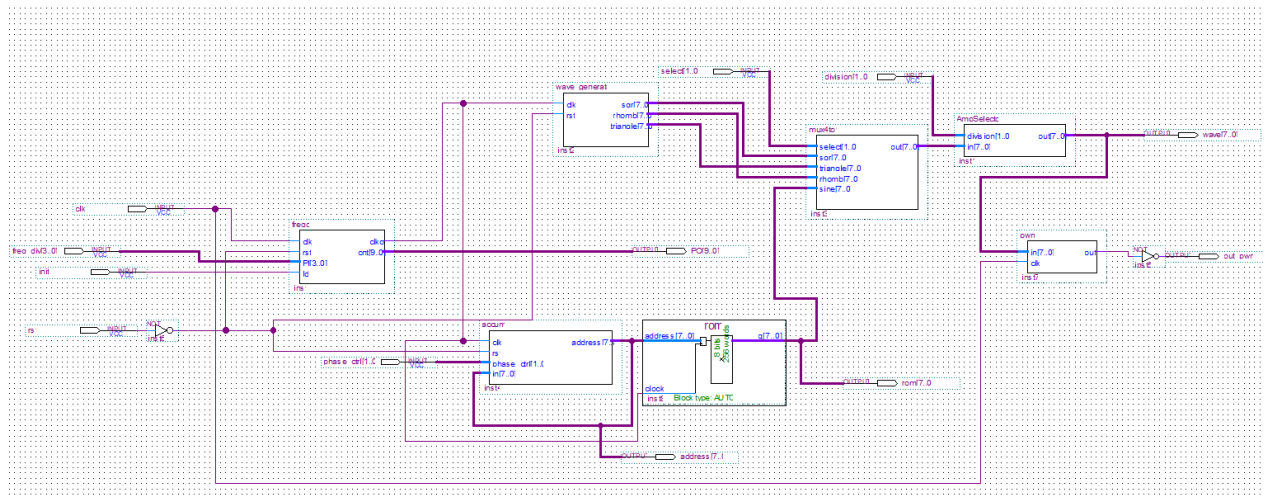
4 Amplitude Selector

One option in function generator is the amplitude of 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

Table 2: Amplitude selection

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

Figure 8: Final block diagram in Quartus II



a number. The value of divisor is chosen by a 2-bit input. Dedicate two bits of input SW (SW[6:5]) to this selector inputs.

Type of amplitude is selected by SW[6:5] according to table 2.

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 amplitude. To do this, change (SW[6:5]) in your testbench.
3. Include the waveforms of each desired amplitude in your report.

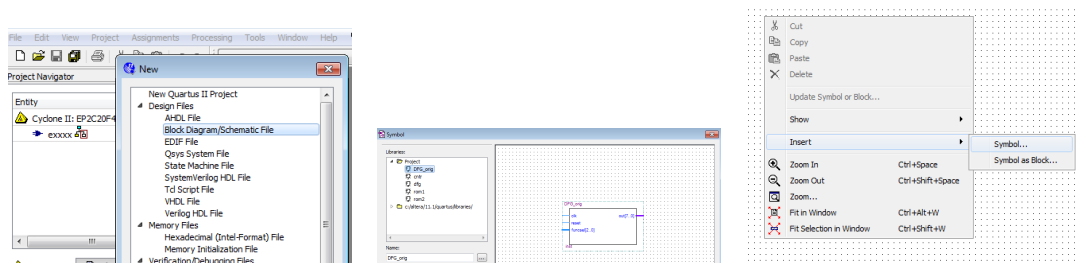
Finally your top-level design must look like the block diagram of figure 8.

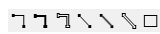
5 The Total design

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 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.

Figure 9: Quartus II



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.

Expand the hierarchy in the Libraries box. First expand libraries, then expand the library primitives, followed by expanding the library logic comprises 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.

Acknowledgment

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

This manual has been revised and edited by **Zahra Jahanpeima**, PHD student of Digital Systems at University of Tehran.