Experiment 1 – Function Generator

Course title: Digital Logic Design Lab

Date : 96/2/13 and 96/2/20

Group 4

Hadi Safari

810194531

hadi.safari@ut.ac.ir

Seyed ahmad A. PouriHosseini

810194358

seyedahmadpourihosseini@yahoo.com

Abstract— **The goal of this experiment is to learn how to build a function generator. To achieve this, participants also learn how to build a Pulse Width Modulator, which is combined with a passive low pass filter to realize a digital to analog convertor. Conversion of mathematical formulas into hardware components is also an important part of this lab.**

Keywords— **function generator, RTL design, mathematical equations, pulse width modulator, low pass filter.**

I. DIGITAL TO ANALOG CONVERTER

Digital to analog converter was the first part that we designed in the lab. We first wrote the code for PWM in Quartus, compiled it and programmed it into the given FPGA (DE1) , and then combined it with the RC circuit which we built using a breadboard and R and C components. Then, we tested the whole module (the DAC module) to verify its correctness. The output was wrong at first due to incorrect assignment of the pins. But the problem was finally found and fixed.

II. DIGITAL FUNCTION GENERATOR

The second step was to build a Digital Function Generator. The list of needed functions were as follows:

- Rhomboid
- Sine
- Square
- Triangle
- Saw tooth

The main idea behind the implementation of all of these functions (except for Sine) was to use an 8-bit counter. The sawtooth function would then be realized by simply using the output of the counter as our final output, the square function by using a comparator (square = counter <128 ? 255 : 0), the triangle function using a MUX that selects between counter and 256 minus counter with a select bit coming from a comparator (counter < 128 ? counter : 256 – counter), and finally the Rhomboid function by swapping between the value of the triangle function, and its negated value. The sine function was implemented using the recursive formula given in the lab instruction file, and with the use of two registers, one for sine and one for cosine.

We first implemented one of the simple functions, namely the square function, in order to test the overall functionality of our function generator. After testing was finished, we also implemented the other more complex functions, including the sine function. The outputs are depicted here:

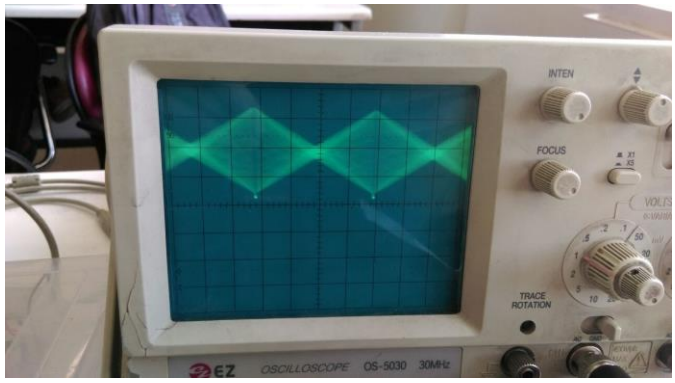

Figure 1. rhomboid function

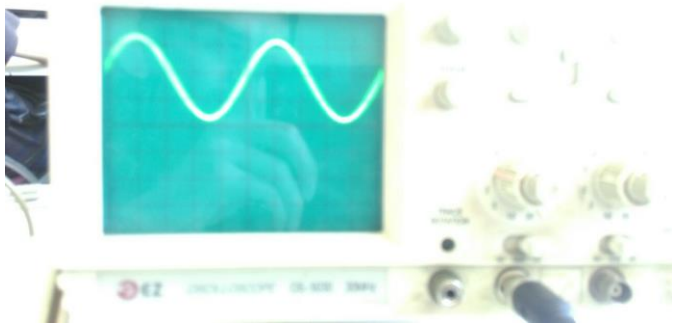

Figure 2. sine function

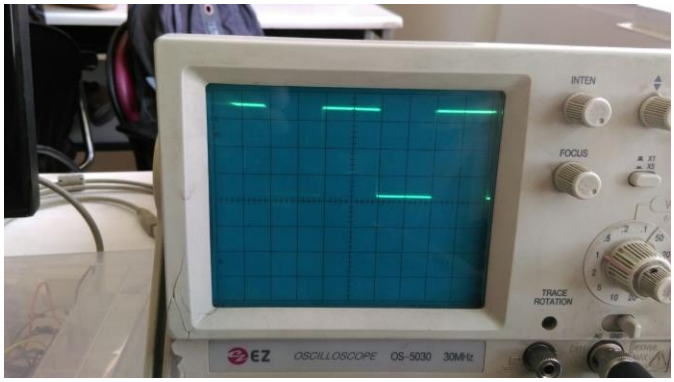

Figure 3. square function

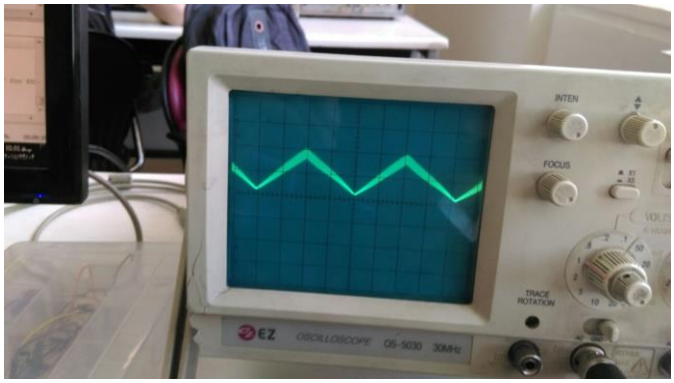

Figure 4. triangle function

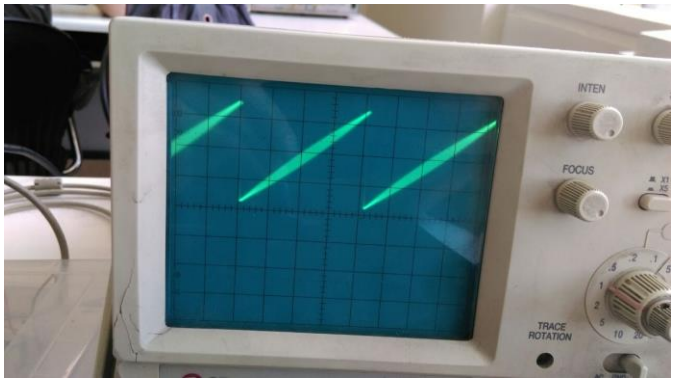

Figure 5. saw tooth function

III. Amplitude selector and frequency selector

We actually implemented a basic version of the frequency selector before we finished the function generator, because a division by at least 256 was needed for the proper functioning of the function generator module (The basic module simply divided the input clock frequency by 256.) but after completing the function generator module, we completed the implementation of these two selectors. The frequency selector was implemented using the classic clock division strategy (using a counter that uses the main clock, and resetting it when it reaches a certain value, while using its output as the new clock) and the amplitude selector was realized using logical shifting (because the divisors were all powers of two). We then assigned the select bits for these two modules to the switches available on the FPGA. At first, again due to incorrect pin assignment, the frequency selector wouldn't work correctly. Also, because the oscilloscope wasn't correctly calibrated (the knob wasn't fully turned to the right) the amplitude selector also seemed to malfunction. The problems were found and fixed.

We were also asked to calculate the expected frequency of one of our functions and to compare it to the observed frequency on the oscilloscope. We chose the square function. The results are as follows:

Expectation:

When frequency selector bits are 2'b00, the input to the function generator is 8KHz. Because the function generator needs 256 clock cycles to complete a whole period, the frequency must equal:

$$8 \text{ KHz} / 256 = 32 \text{ Hz}$$

Observation:

We observer 6.4 units of time for the Time period of square function when the TIME/DIV knob was set on 5ms. Thus, the observer frequency equals:

$$1/ (6.4 * 5 * 10^{-3}) = 32.25 \text{ Hz}$$

The small error (0.25 Hz) is understandable due to inaccurate measurement of the time period on the oscilloscope.

IV. Sound

In the final part, we had to implement the circuit that would play a sound file. The sound file was given to us via the CECM website. We were instructed to use a ROM and fill it with the sound data. Also, in order to reduce the frequency of the sound to a range that is audible to the human ear, we used the 12^{th} bit of the output of the frequency selector module, as the clock input for the counter that generated the read address for the ROM. We created the ROM using the mega wizard feature of the Quartus program, and then added it to our existing BDF file. The final wiring of the circuit is depicted below:

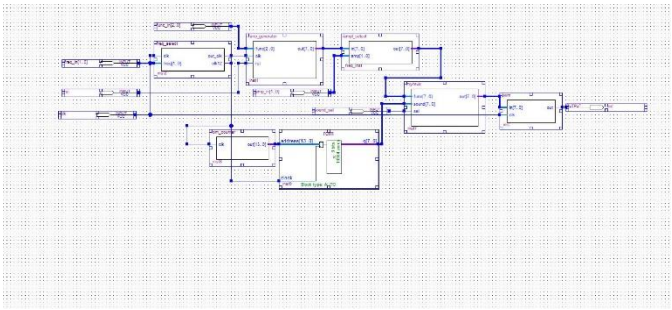

Figure 6. complete circuit BDF