# EE314 Digital Electronics Laboratory Project Final Report

## An FPGA Based Oscilloscope System

Mehmet Ali Açıkbaş, Anıl Aydın
Middle East Technical University
Department of Electrical and Electronics Engineering
Ankara, Turkey
acikbas.mehmet@metu.edu.tr, anil.aydin@metu.edu.tr

***Abstract*—This report includes all works related to EE314 Digital Electronics Laboratory Term Project namely An FPGA Based Oscilloscope System which is designed by authors.**

***Index Terms*— FPGA, VGA, ADC converter, verilog, frequency, amplitude, oscilloscope.**

I.INTRODUCTION

This project aims to analyse and visualize analog signals by using FPGA. We may say that this system works as an oscilloscope which is commonly used to evaluate analog signals.
The system has three main parts. These are analog to digital converter (ADC), computation unit and VGA screen. Although constructing a register was recommended in the project manual we did not need to do that.
ADC is the first part of our system. In this part, an analog signal is converted to digital data due to our FPGA works with digital logic. System's second part, computation unit, analyses discrete digital signals to find amplitude, offset, frequency and RMS values of the analog signal. Third part is VGA screen which visualized the signal in a certain monitor by using some code manipulations.

II. PROJECT EQUIPMENTS

· One signal generator (for testing)
· One FPGA
· One Monitor
· One Computer (for coding)
· Jumpers

III. SYSTEM DESIGN

*A. Analog to Digital Converter (ADC)*

ADC is essential in our system since input of our system is analog signal but FPGA needs digital signals for all logic operations. Our FPGA has an ADC within it. However, we had to write a suitable Verilog code to use ADC properly. We found a code structure on the internet and modified it for our purpose.
In the beginning of this process we examined user manual our FPGA, specifically ADC Part. We saw that ADC of our FPGA has 12 bits output. Therefore we defined an 12 bits output in our ADC module. Note that ADC gives 12 bits output for 16 clock cycle. For the first and second cycle ADC does not do any converting operation. For third, fourth and fifth cycles ADC creates three bits output, that is '000'. These bits represents that Pin0 of the ADC is used. Also, last bit of this signal, which corresponds to fifth cycle, is going to be first bit of the digital output.
To write a proper ADC code for our application, we found a source code from the internet and we modified it according to our needs.
In our code structure we have 'clock', 'reset', 'ADC_CS_N', 'ADC_DIN', 'ADC_SCLK', 'ADC_DOUT' and serial_parallel_data parameters. Reset should be zero when ADC works. 'ADC_CS_N' is used for chip selection. It works as an enable signal for ADC. When it is low our ADC works properly. 'ADC_SCLK' is the clock parameter of the ADC. Also we have a state machine in our ADC code and this simply decides to whether convert the signal or not. To determine the 16 cycle which is necessary for 12 bits output, we have a counter which is embedded to state machine and according to state transitions, the code gives the proper conversion. After the conversion process, we converted the serial output of ADC to a parallel data to use is properly.The state transition diagram for the ADC which we obtained in Quartus is shown in Figure 1.

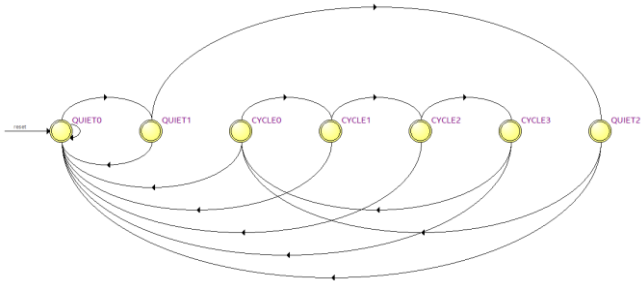


*Figure 1: State transition d diagram of ADC state machine*

To verify the ADC for proper operation, we constructed a simple circuit which gives varying DC analog outputs for the input of ADC and we assigned LEDs to see how the digital output changes according to changing analog input and we saw that the binary outputs is changing properly. After that we started to construct our computation unit with the output of our ADC.

## *B. Computation Unit*

Computation unit is designed to determine amplitude , offset voltage, frequency and RMS values of the signal. We created two separate modules for this operation.

1) Magnitude Module: This module is designed to evaluate amplitude, offset and RMS values. However, we could not find the RMS value practically. We tried to take the square of the discrete values of the input and then take the square root of this and finally divide this to the period to decide which waveform is given. However, we were not succeeded in this method.
In our module, we defined 'voltage', 'clk', 'vpp, 'dc_offset' parameters. Also, we defined an integer as a 'counter' and it counts up to 49999999 since our FPGA works in 50 Mhz to see the change as 1Hz. Voltage parameter is 12 bits inputs and this data comes from the output of the ADC. It's dimension can be change according to analog signal which is evaluated. 'clk' is used always in posedge. This means we have common clock (synchronous) sequential circuit in this module. To find amplitude and offset values we have to determine minimum and maximum peak values of the signal. For this purpose we defined 12 bits 'min' and 'max' terms in addition to 'amp' term. Using this additional terms, system evaluates a discrete signal, whether it is minimum/maximum peak value or not. After it finds the peak values, amplitude and offset values can be calculate by using following algorithms:

amp<=(max-min) dc_offset<=(max+min)/2

The simulation results for Vpp and DC offset values is shown in Figure 2. Test bench codes for the simulation in magnitude module is in Appendix A.

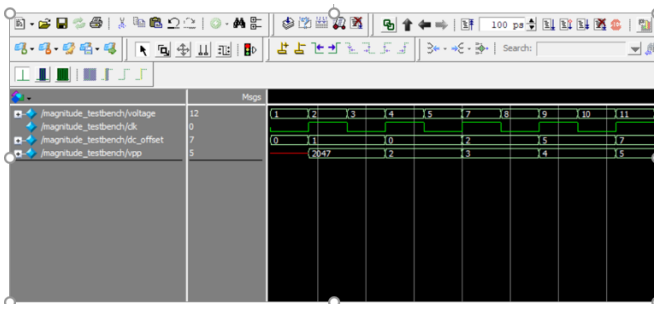

*Figure 2: Simulation results for vpp and dc offset*

In the simulation, there was a problem that when we take the average of the max and min values, since the result will be an integer, rational numbers are rounded to first digit of the result. To solve this problem, we multiplied the output with 50 and then assigned 2 digits for vpp and dc offset in VGA module. Thus, with this solution we did not lose any data.

*AC-DC mode*
After finding DC offset value, we automatically created an algorithm to use this oscilloscope in two modes. When this operates in DC mode, we see an offset voltage on the screen. When we change the mode to AC mode our offset voltage is zero. The state diagram for AC-DC mode is shown in Figure 3.

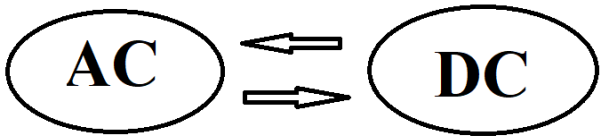


*Figure 3: State diagram for mode selection*

2) Frequency module: This module is designed to find the period of the signal and accordingly frequency of the signal. The method to find the period is measuring the time for the signal coming from the minimum to maximum value. This time is equal to half of the period. To measure this time, a counter is designed. Again not to lose any data, a constant is determined and by dividing the constant to counter, an integer was found. This integer is embedded to a 32 bits register and this value is the output of the frequency module.

There is a state machine in the frequency module which determines the counter value. The state diagram is shown in Figure 4.

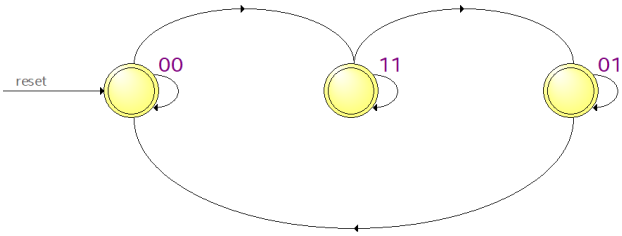


*Figure 4: State diagram for frequency module*

*For the frequency module, there is a simulation result as shown in Figure 5. The simulation results did not give the correct result. We think that this may be cause of an*

*overflow. Experimentally we obtained the correct frequency values.*

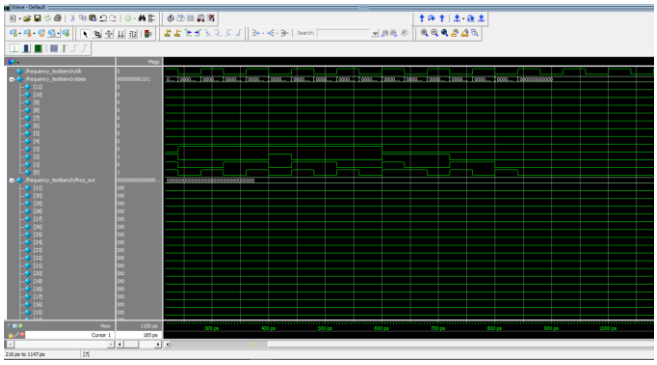

*Figure 5: Simulation results for frequency module*

Test bench codes for the simulation of frequency module is in Appendix A.

### *C. VGA MODULE*

The video graphics array is essential to visualize the waveforms and the computed values of an analog signal in tis project. To make a proper visualization, VGA must be synchronized properly. The timing diagram for the VGA is shown in Figure 6.

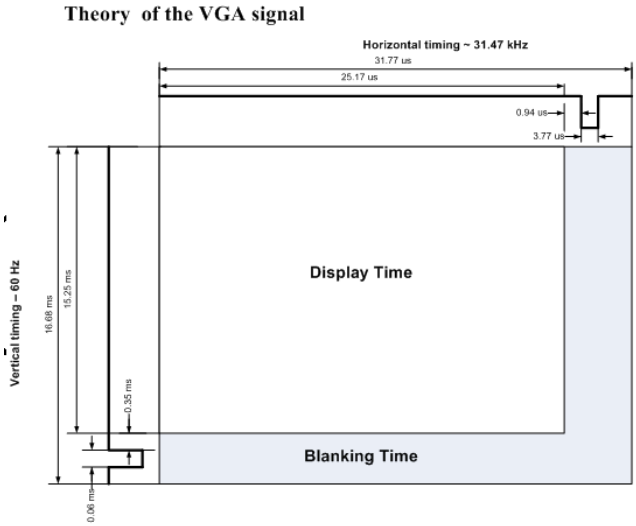


*Figure 6: Timing diagram of VGA*

The horizontal timing is 60 Hz and vertical timing is 31.47 kHz for the VGA. There is an active video region called display time and to write the pixels to active area, VGA must be triggered vertically and horizontally. 96 pixels are needed for horizontal synchronization and 2 lines are needed for vertical synchronization. At the positive edge of these pulses VGA starts to write.

After the synchronization, the horizontal and vertical axes are determined by writing codes for the determined pixels. Also, these axes are scaled by 50 pixels and 50 lines. In the template of the oscilloscope, we arranged the frequency, rem, amplitude and offset voltage values at the bottom of the screen as random values when there is no data.

The template of the screen is shown in Figure 7.

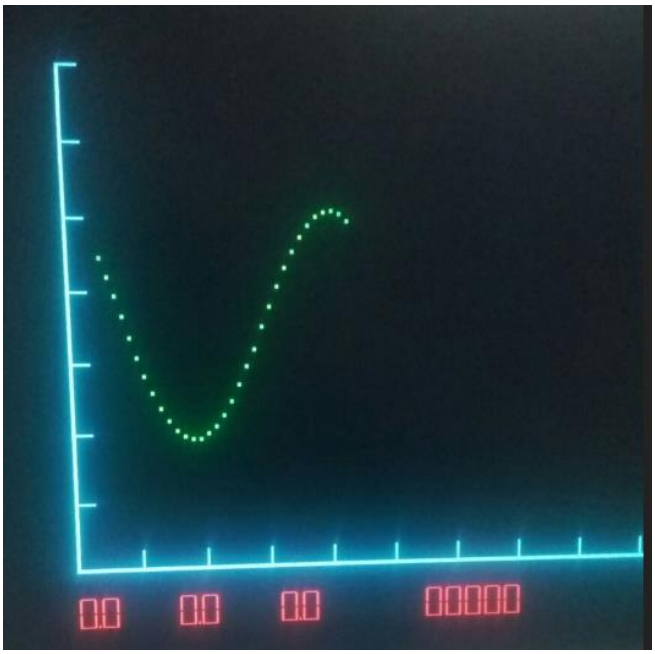

*Figure 7: Template of the screen*

Determination of the digits: For the amplitude value, first digit is found by taking the 10% of vpp data coming from the magnitude module. The second digit is found by dividing the data to 10 and taking mod of this value. The same procedure is used to find the digits of the other quantities. When a signal is connected to ADC and the process is started, these values are changes according to the data coming from the computation unit.

Determination of the waveform: Since we could not find the rms values of the signals, we were not able to determine the type of the waveform. However, we constructed templates for sinusoidal, square and triangular waves by using the same procedure as screen design. A sinusoidal waveform example is shown in Figure 8.

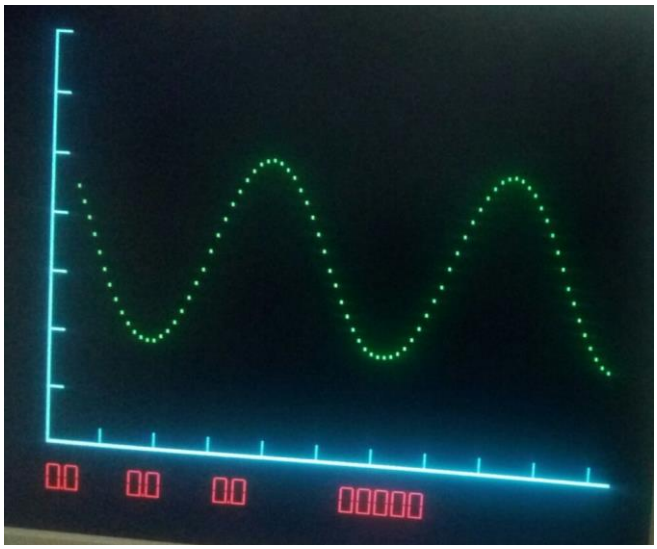

*Figure 8: Sinusoidal waveform*

Auto scaling: We designed our system such that when the signal period or amplitude is changed, time and volt division values are changed accordingly. To do that we assigned constants to the equations of the waveforms and when the frequency and amplitude inputs change, these constants are also changes and thus while time and volt divisions are changed waveform shapes are stay constant. However, we could not represent this system since we could not determine the type of the waveform.

## TOP MODULE

We combined all the modules that we wrote in a top module called "adc_vga". In this module we combined inputs and outputs for the overall system. The RTL view of the overall system is shown in Figure 9.

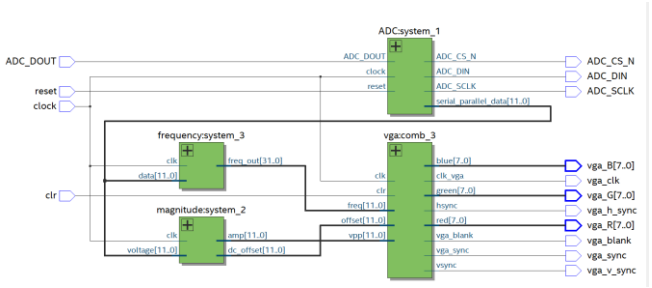


*Figure 9: Overall system*

## APPENDIX A

Test bench codes for magnitude module which contains offset voltage and vpp are shown in Figure 10.

```
module magnitude_testbench;
    reg [11:0] voltage;
    reg clk;
    wire [11:0] dc_offset;
    wire [11:0] vpp;
    magnitude DUT(voltage,clk,dc_offset,vpp);
    always begin
    clk=0; #40; clk=1; #40;
    end

    initial  begin
        voltage = 12'b000000000000;

        voltage = 12'b000000000001;#40; voltage = 12'b000000000010; #40;voltage = 12'b000000000011;
        #40;
        voltage = 12'b000000000100;#40; voltage = 12'b000000000101; #40;voltage = 12'b000000000111;
        #40;
        voltage = 12'b000000001000;#40; voltage = 12'b000000001001; #40;voltage = 12'b000000001010;
        #40;
        voltage = 12'b000000001011;#40; voltage = 12'b000000001100; #40;voltage = 12'b000000001011;
        #40;
        voltage = 12'b000000001010;#40; voltage = 12'b000000001001; #40;voltage = 12'b000000001000;
        #40;
        voltage = 12'b000000000111;#40; voltage = 12'b000000000101; #40;voltage = 12'b000000000100;
        #40;
        voltage = 12'b000000000011;#40; voltage = 12'b000000000010; #40;voltage = 12'b000000000001;
        #40;
        voltage = 12'b000000000000;#880;
    end
  endmodule
```

*Figure 10: Test bench codes for vpp and offset*

Test bench codes for frequency module are shown in Figure 11.

```
module frequency_testbench;
    reg [11:0] data;
    reg clk;
    wire [31:0] freq_out;
    frequency DUT(data,clk,freq_out);
    always begin
    clk=0; #40; clk=1; #40;
    end
    initial  begin
        data = 12'b000000000000;

        data = 12'b000000000001;#40; data = 12'b000000000010;#40;data = 12'b000000000011;
        #40;
        data = 12'b000000000100;#40; data = 12'b000000000101;#40;data = 12'b000000000111;
        #40;
        data = 12'b000000001000;#40; data = 12'b000000001001;#40;data = 12'b000000001010;
        #40;
        data = 12'b000000001011;#40; data = 12'b000000001100;#40;data = 12'b000000001011;
        #40;
        data = 12'b000000001010;#40; data = 12'b000000001001;#40;data = 12'b000000001000;
        #40;
        data = 12'b000000000111;#40; data = 12'b000000000101;#40;data = 12'b000000000100;
        #40;
        data = 12'b000000000011;#40; data = 12'b000000000010;#40;data = 12'b000000000001;
        #40;
        data = 12'b000000000000;
        #8800;
    end
endmodule
```

*Figure 11: Test bench codes for frequency module*

## IV. CONCLUSION

In this project, we mainly worked on fundamentals of logic design and Verilog language. We had some difficulties about finding right algorithms and coding them with minimum effort. Also, FPGA structure has deeply investigated. Using FPGA in efficiency ways is learned. We refreshed our knowledge about combinational and sequential circuits which are commonly used in logic design. All in all, we developed our understanding about logic structures and learned how to construct Verilog algorithms for a certain purpose. It was very beneficial experience.