MID SWEDEN UNIVERSITY

VHDL Project Report

by

Ferrari Nico

Contents

Chapter 1

Introduction

This report describes the implementation of our project considering a monitoring system that detects the position of the closest object in front of it. This kind of monitoring systems are of great use recently, for example, car monitoring system can be seen as an application. The system plots the positions on the graph from left to right. To store the position of the object over a period of time we created a memory and displayed the plot on the monitor screen. We have surrounded the plot by two different Flag patterns. This project is designed using VHDL programming and implemented on Digilent BASYS 3 FPGA Board.

1.1 Project Description

In our project we have used ultrasound range sensor SRF05 to detect the position of the object in front of it. The sensor detects distance of closest object in front of it up to 450cm at a maximum frequency rate of 20 Hz. 512 position samples from this rage sensors are stored in FPGA embedded memory and the content of memory are displayed at the center of video monitor. The memory content are then padded by the flag pattern. The project was implemented after completing 3 Labs divided into different parts and putting all these together. This project developed a discrete understanding of such monitoring systems.

Chapter 2

Hardware Requirements

There are some hardware requirements to implement our project and are given as bellow:

- Digilent BASYS 3 Board;
- Range Sensor SRF05;
- VGA Monitor;

2.1 Digilent BASYS 3 Board

The Basys 3 is an entry-level FPGA development board designed exclusively for Vivado Design Suite, featuring Xilinx Artix-7 FPGA architecture. It includes enough switches, LEDs and other I/O devices to allow a large number of designs to be completed without the need for any additional hardware, and enough uncommitted FPGA I/O pins to allow designs to be expanded using Digilent Pmods or other custom boards and circuits. It has 16 user switches, 4-digit 7 segment display, 12-bit VGA output, Digilent USB-JTAG port for FPGA programming and communication, three Pmod ports, 16 user LEDs, 5 user push buttons, serial flash and many more advanced features and ports.

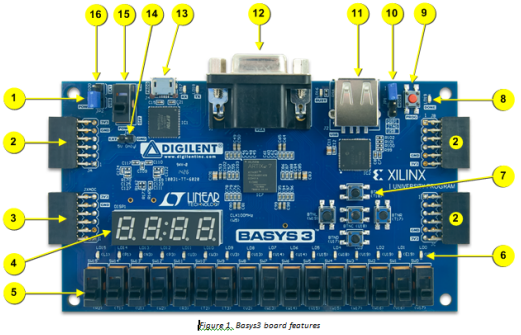


Figure 1. Basys3 board features

Callout	Component Description	Callout	Component Description
1	Power good LED	9	FPGA configuration reset button
2	Pmod connector(s)	10	Programming mode jumper
3	Analog signal Pmod connector (XADC)	11	USB host connector
4	Four digit 7-segment display	12	VGA connector
5	Slide switches (16)	13	Shared UART/ JTAG USB port
6	LEDs (16)	14	External power connector
7	Pushbuttons (5)	15	Power Switch
8	FPGA programming done LED	16	Power Select Jumper

FIGURE 2.1: Basys 3 Artix-7 FPGA Trainer Board: Recommended for Introductory Users

Features that are most relevant to the project is listed in Table 2.1

Xilinx Artix-7 FPGA: XC7A35T-1CPG236
33,280 logic cells in 5200 slices
USB-powered (micro-B)
1,800 Kbits of fast block RAM
12-bit VGA output
16 user switches
16 user LEDs
5 user pushbuttons
4-digit 7-segment display

TABLE 2.1: Basys 3 Artix-7 FPGA Trainer Board

2.2 Range Sensor SRF05

Ultrasonic Range sensor SRF05 is used to detect object in front of it. It is mainly used to measure the distance of the object in front of it. It has the range of up to 4 meters. In this improved version by connecting the mode pin to the ground allows SRF05 to use a single pin for both echo and trigger and when the mode pin is left unconnected the range sensor operates separately with trigger and echo pins. It has a small delay before the echo pulse to give slower controllers such as the Basic Stamp and Picaxe time to execute their pulse in commands.

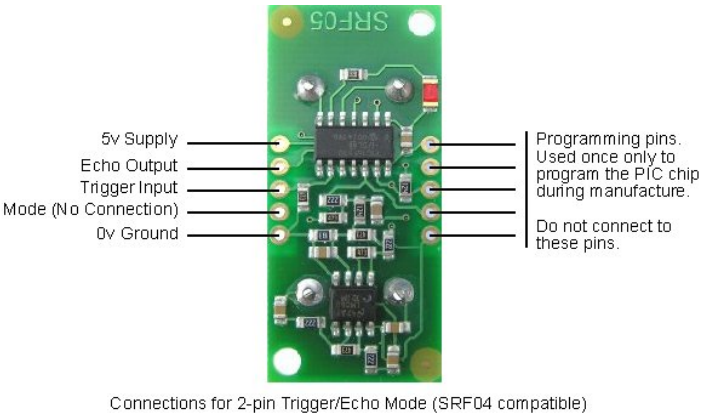


FIGURE 2.2: Ultrasonic Range Sensor SRF05 pinout

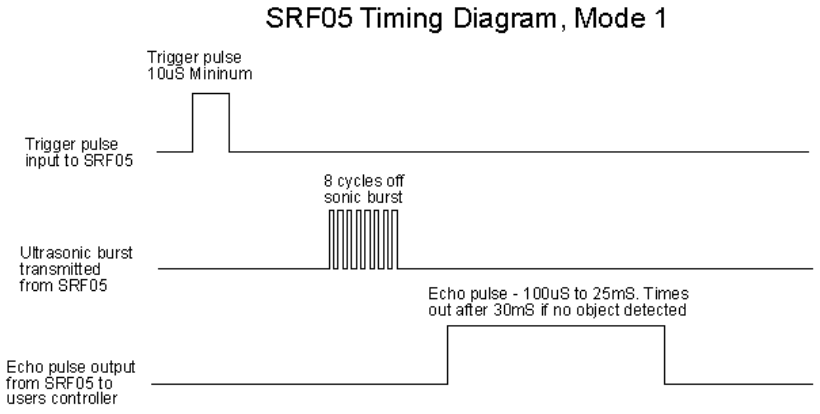


FIGURE 2.3: Ultrasonic Range Sensor SRF05 MODE 1 timing diagram

As we can see in Figure 2.3, the sensor requires input trigger pulses that have a minimum width of 10s to activate the module to start detecting. Then, the sensor automatically sends eight 40khz square waves, and detects when there is a reflect signal reciving a reflect signal back, the ECHO I/O, whose high-level signal's duration is proportional to distance. Giving the length of this signal, is then possible to calculate the distance in centimeters from the object with this formula:

$$\frac{echoSignal[\mu s]}{58} = distance$$

2.3 VGA Monitor

VGA is a standard interface for controlling analog monitors. The computing side of the interface provides the monitor with horizontal and vertical sync signals, color magnitudes, and ground references. The Basys3 board uses 14 FPGA signals to create a VGA port with 4 bits-per-color and the two standard sync signals (HS Horizontal Sync, and VS Vertical Sync). VGA display take the form of X-columns and Y-rows of picture elements (pixels) starting from top left. Each pixel can be monochrome (Y) or colour (RGB red, green and blue).

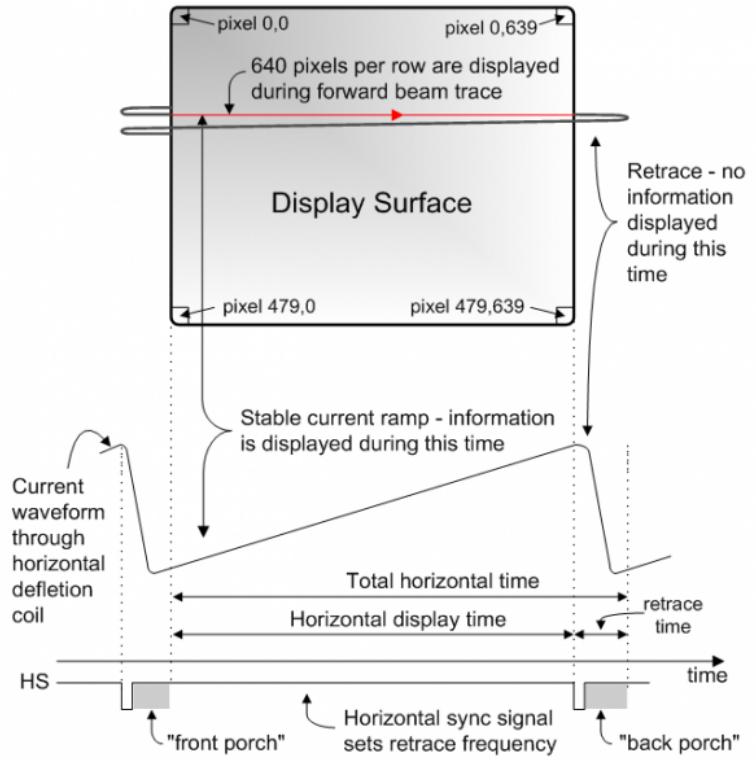


FIGURE 2.4: VGA scheme

We used VGA monitor with the Screen resolution of our choice to visualize the results we obtain from our ultrasonic range sensor (SRF05). The purpose of VGA is to overlook the communication between BASYS 3 FPGA board and our monitor via VGA port. The FPGA provides an 12-bit VGA port, that means RGB colors represented by 4 bits each, hence RGB444.

General timing

Pixel freq	108 MHz
Screen refresh rate	60 Hz

Horizontal timing (line)

Polarity of horizontal sync pulse is positive

Scanline part	Pixels
Visible area	1280
Front Porch	48
Sync pulse	112
Back porch	248
Whole line	1688

Vertical timing (frame)

Polarity of vertical sync pulse is positive

Scanline part	Lines
Visible area	1024
Front Porch	1
Sync pulse	3
Back porch	38
Whole frame	1066

TABLE 2.2: VGA timing data for 1280x1024 resolution

The project is built using the VGA resolution 1280x1024 at a 60 Hz timing frequency. This requires a pixel clock of 108MHz.

Chapter 3

System Design

3.1 Design Overview

In the following diagram (Figure 3.1)we can clearly see the basic design flow of our project. We can see the most important elements involved in our project and how they are interconnected and how they interact with each other. First of all using VHDL we designed the project and interfacing with the hardware we get the desired results. VHDL makes it easier to design and implement projects like ours.

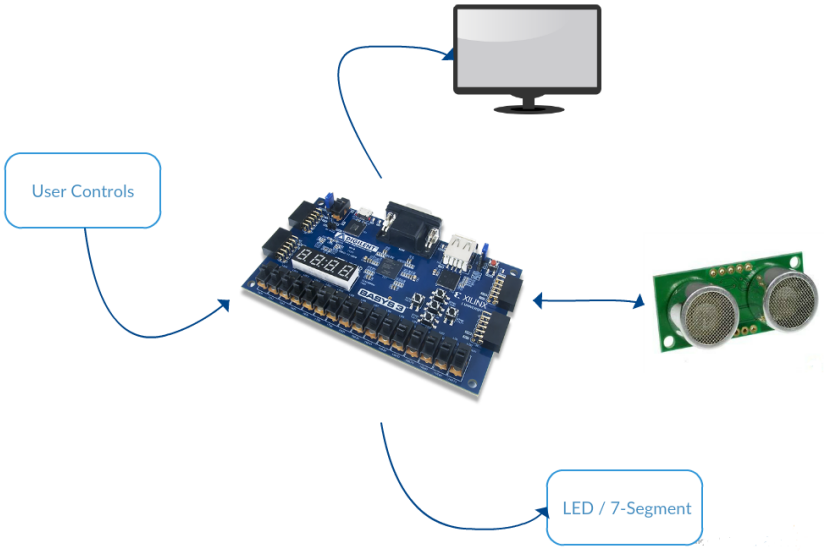


FIGURE 3.1: Design Diagram of the Project

Figure 3.2 is the hierarchy that show how all of the VHDL components are structured and connected.

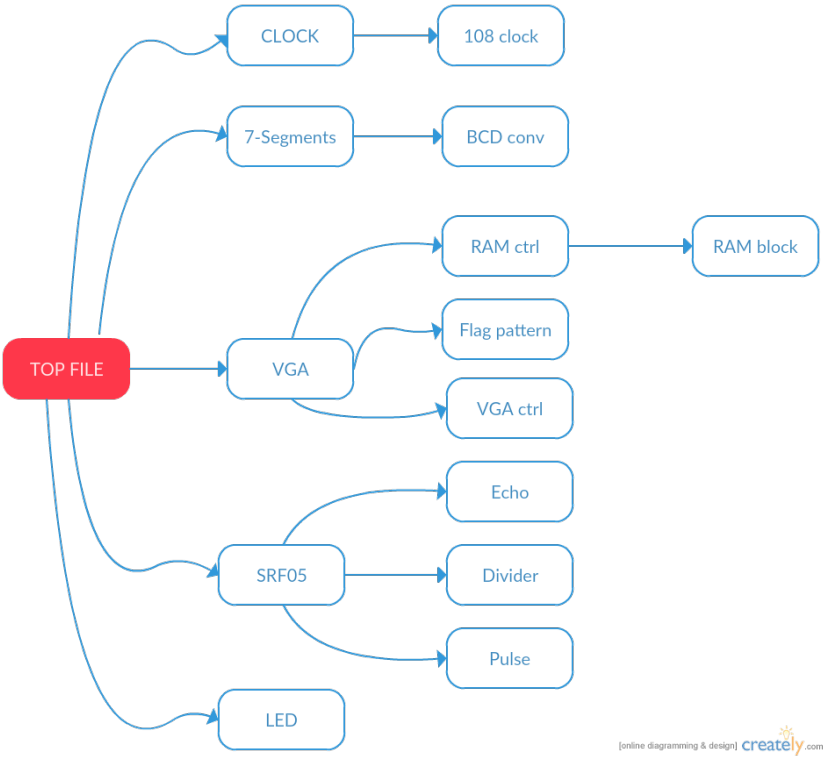


FIGURE 3.2: Design overview, block diagram

3.2 Code Description

All of the code is developed in Vivado 2015.3, bringing attention to create a modular system and using material learned during the course. The system is represented by a main top file that connects modules together.

General named signals into the models have the suffix IN or OUT to avoid confusion. To get rid of the redundancy encountered when instantiating with a component, entity instantiation has been used, making the code less error-prone.

Chapter 4

Modules

4.1 Clock Generator

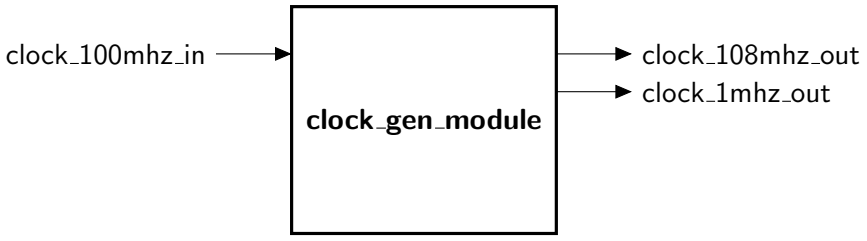


FIGURE 4.1: Entity of clock_gen_module

The clock generator module generates all of the necessary clock frequencies, 1MHz and 108MHz, for this project. The module has as input the board clock which is a 100MHz clock, downscaled to 1MHz by a counter that counts to 50 (half period) keeping the signal value and then it inverts it,generating a 50% duty-cycle. The value of the counter is increased every cycle of clock (10ns) therefore it generates a "slow clock" with a period of 100μs (1MHz). This module contains another module that generates a 108MHz clock.

4.1.1 Clock 108MHz

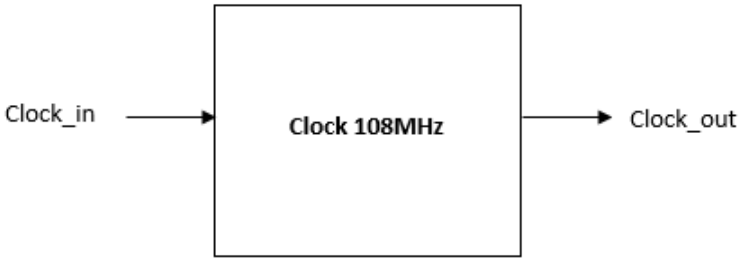


FIGURE 4.2: Clock 108MHz Module

This clock is generated from a clock wizard provided by Vivado and it is used only for the VGA display and RAM reading.

4.2 SRF05

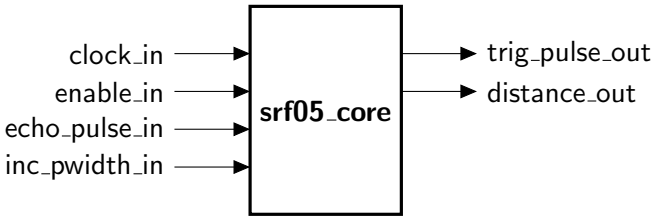


FIGURE 4.3: Entity of srf05_core

The SRF05 core is the driver for the SRF05 Range Sensor used in this project. It is composed by 3 modules:

- pulse generator
- echo pulse analyzer
- divider

4.2.1 Pulse generator

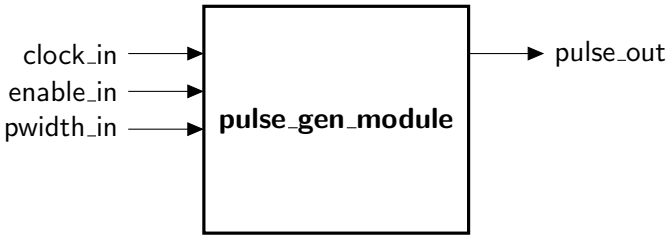


FIGURE 4.4: Entity of pulse_gen_module

The pulse generator module activates the range measurement by sending a trigger pulse to the sensor.
Due to the requirements from laboratory 3,it generates pulses of 10 s up to 20s width that are required by the range sensor as trigger signals with a frequency of 10Hz. The code is structured so that the pulse width can be adjusted between 15μs and 20μs during runtime.
This module is composed by the following ports:

- clock_in: The frequency of this clock is 1 MHz
- enable_in: This port is used to control the generation of the pulse, if it is activated, the generation of the signal is stopped.
- pwidth_in: If activated, it switch the pulse lenght from 15μs to 20μs
- pulse_out: The pulse signal is kept high (1) for all the needed μs specified by pwidth signal and put low (0) when the desired width is reached by a specific counter.

4.2.2 Echo Pulse Analyzer

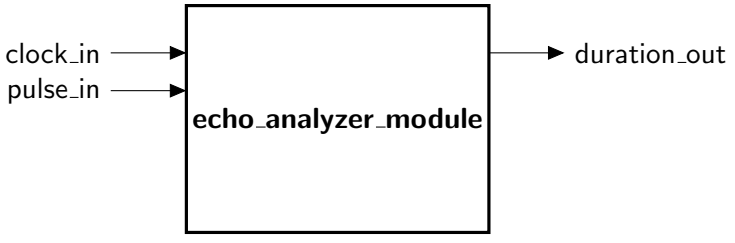


FIGURE 4.5: Entity of echo_analyzer_module

The purpose of the echo pulse analyzer is to measure the duration of the response pulse from the SRF05 sensor. It executes an analysis of the received signal getting the pulse

duration (pulse_in) as number of clock_in cycles in the echo, using the 1μs clock, to produce a 16bits duration_out that will later be converted in the centimeters. According to the SRF05 datasheet, a timeout is handled if the duration exceeds 30 ms.

4.2.3 Divider

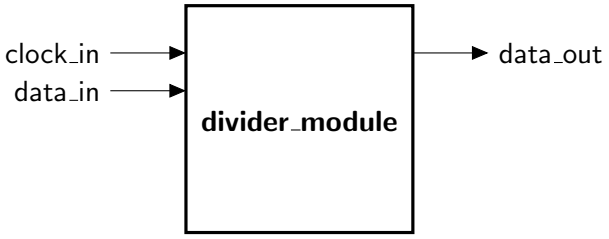


FIGURE 4.6: Entity of divider_module

This module performs the integer division of the duration recived from the Echo Analyser module (data_in) by 58 to obtain the distance expressed in centimeters.
For all values below 58 the data_out signal is set to 0. Other values is multiplied with 1130 into a temporary storage of 32 bits to avoid overflow and then it is right shifted 16 positions and resized to a depth of 16 bits. The value 1300 come from

$$f = \frac{2^{16}}{58} = 1129.93 \approx 1130$$

where 16 is the size of the vector.

4.3 VGA Core

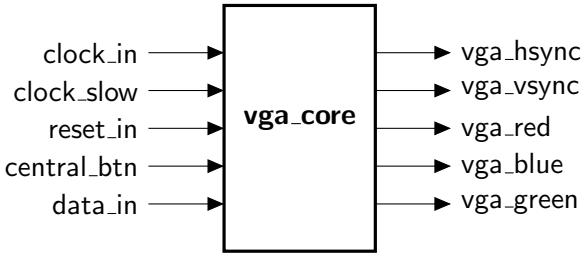


FIGURE 4.7: Entity of vga_core

This module consists of two processes: sending data to the RAM controller and present the data on screen controlling the position and layout of the presented data. This module is dependent of the RAM controller module, Flag Pattern modules and the VGA controller module. The module has as input a button signal (central_btn) that is

set high when the central button on the board is pressed and it is used to switch between two different flag patterns. VGA core is composed by two main processes:

Data storage process

This is a synchronous process using the slow clock (1MHz), acquires the sensor data within 0 to 450 and sends it to the RAM controller which takes care of when and how it is saved.

Graphics process

This process takes care of the graphical elements drawn on the VGA display:

- the flag pattern used as background
- the graph frame.

The data is read from the RAM controller with the VGA clock at the frequency of 108MHz. To avoid an initial delay the address request is advanced by 2 iterations.

4.3.1 VGA Controller

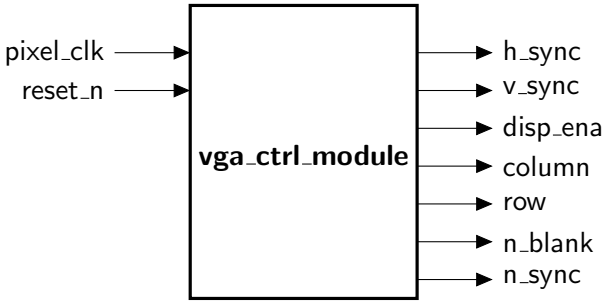


FIGURE 4.8: Entity of vga_ctrl_module

This module is a VGA interface controller that generates the signal timing for it. This is the same module given for the first laboratory, the only thing that had to be changed were the monitor parameters.

```
GENERIC(
  h_pulse  :  INTEGER   := 112;   --horiztonal sync pulse width in pixels
  h_bp     :  INTEGER   := 248;   --horiztonal back porch width in pixels
  h_pixels :  INTEGER   := 1280;  --horiztonal display width in pixels
  h_fp     :  INTEGER   := 48;    --horiztonal front porch width in pixels
  h_pol    :  STD_LOGIC := '1';   --horizontal sync pulse polarity (1 = positive, 0 = negative)
  v_pulse  :  INTEGER   := 3;     --vertical sync pulse width in rows
  v_bp     :  INTEGER   := 38;    --vertical back porch width in rows
  v_pixels :  INTEGER   := 1024;  --vertical display width in rows
  v_fp     :  INTEGER   := 1;     --vertical front porch width in rows
  v_pol    :  STD_LOGIC := '1');  --vertical sync pulse polarity (1 = positive, 0 = negative)
```

FIGURE 4.9: SRF05 Module

4.3.2 Flag Pattern

FIGURE 4.10: Entity of italian_flag_pattern

This module uses the row and column positions given by the VGA controller to determine which RGB444 color that should be sent to the VGA display. This represents the background pattern and in the project have been implemented two different patterns: the Italian and the Norwegian(that is more difficult to implement) flag patterns.

4.4 RAM

RAM block is used to store data collected by the range sensor providing them to the FPGA as time series. It consists of two modules:

- Ram Block
- Ram Controller

Each module is created with the purpose of flexibility and reusability using generic variables to set the RAM parameters.

4.4.1 RAM Block

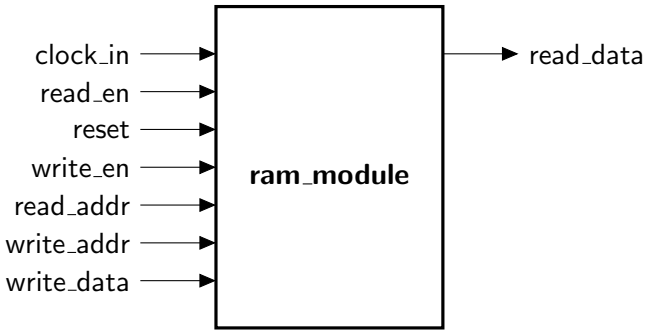


FIGURE 4.11: Entity of ram_module

The RAM module is implemented as a generic RAM module with enable signals for read and write, a signal for reset, signals for data input and output, and a signal for setting the read address and write data. In this project the storage in the RAM is limited to the minimum size necessary which is a depth of 9 bits and 512 addresses, giving the RAM a size of 512 x 9 bits = 4k.

4.4.2 RAM Controller

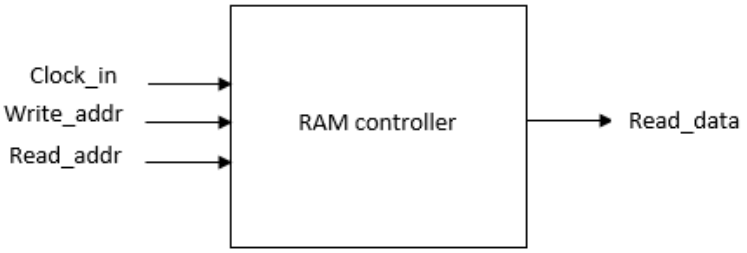


FIGURE 4.12: RAM controller Module

The RAM controller works as a circular buffer3. One counter keeps track of the write index of the RAM memory. This index increases for each data that is written, and the read address is shifted accordingly in the controller as shown in Figure 4.13.

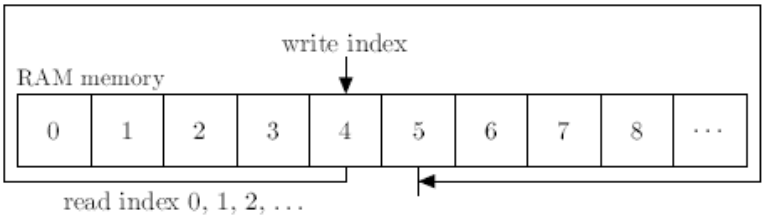


FIGURE 4.13: Circular buffere

With this structure we start to read the RAM from the last saved data.

4.5 7-segment

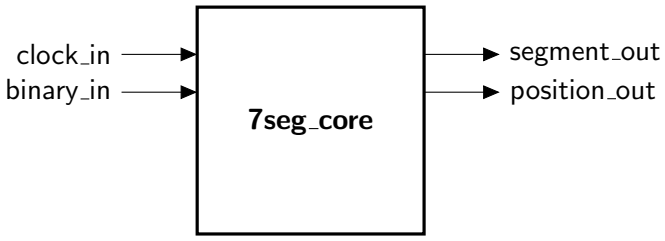


FIGURE 4.14: Entity of 7seg_core

The FPGA onboard 7-segment display is composed by four digits and each of these is composed of seven segments arranged in a pattern, with an LED embedded in each segment. Segment LEDs can be individually illuminated. The anodes of the seven LEDs forming each digit are tied together into one common anode circuit node, but the LED cathodes remain separate. The common anode signals are available as four digit enable input signals to the 4-digit display. The cathodes of similar segments on all four displays are connected into seven circuit nodes labeled CA through CG. These seven cathode signals are available as inputs to the 4-digit display. This signal connection scheme creates a multiplexed display, where the cathode signals are common to all digits but they can only illuminate the segments of the digit whose corresponding anode signal is asserted. To illuminate a segment, the anode should be driven high while the cathode is driven low. However, since the Basys3 uses transistors to drive enough current into the common anode point, the anode enables are inverted. Therefore, both the AN0..3 and the CA..G/DP signals are driven low when active. A scanning display controller circuit can be used to show a four-digit number on this display. This circuit drives the anode signals and corresponding cathode patterns of each digit in a repeating, continuous succession at an update rate that is faster than the human eye can detect. Each digit is illuminated just one-fourth of the time, but because the eye cannot perceive the darkening of a digit before it is illuminated again, the digit appears continuously illuminated. If the update, or refresh, rate is slowed to around 45 hertz, a flicker can be noticed in the display.

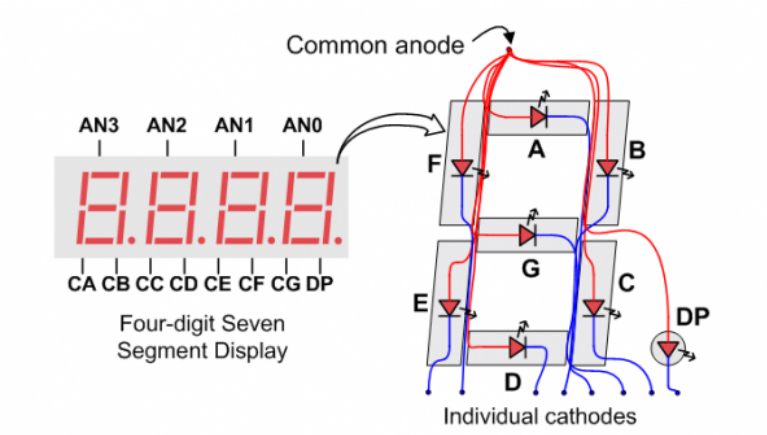


FIGURE 4.15: 7-segment display

The 7-segment display module deals with the representation of the real-time detected measurements by the embedded 7-segment display on the FPGA board. This core serves as an direct indicator for the sensor value, meaning it does not store the data in RAM but output the data to the 7-segment display directly. The 7-segment display is driven by the slow clock (1MHz). Two case structures are used, one for translating the BCD into the 7-segments respectively and one for the position of the digit. The update frequency of the display is set to 1 kHz, this is the fastest possible refresh rate according to the documentation.

4.5.1 Binary to BCD

FIGURE 4.16: Entity of bin_to_bcd

This module converts a 16 bit binary number into 4 binary-coded decimals (BCD). These 4-bit BCD numbers are still binary numbers but each one only goes from 0 to 9, even though 4-bits can go from 0 to 15, to represent the 10 possibilities of a decimal number.

4.6 LED Core

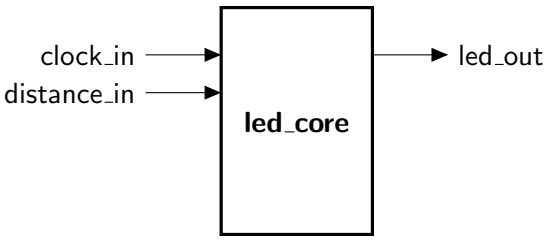


FIGURE 4.17: Entity of led_core

This module converts the distance into binary ones respectively using a static case structure where each case represent a step of 30 cm up to 450 cm.

```
case lim is
    when   0         => num := 0;
    when   1 to  30 => num := 1;
    when  31 to  60 => num := 3;
    when  61 to  90 => num := 7;
    when  91 to 120 => num := 15;
    when 121 to 150 => num := 31;
    when 151 to 180 => num := 63;
    when 181 to 210 => num := 127;
    when 211 to 240 => num := 255;
    when 241 to 270 => num := 511;
    when 271 to 300 => num := 1023;
    when 301 to 330 => num := 2047;
    when 331 to 360 => num := 4095;
    when 361 to 390 => num := 8191;
    when 391 to 420 => num := 16383;
    when 421 to 450 => num := 32767;
    when others => num := 52428; -- out-of-range pattern
end case;

led_out <= std_logic_vector(to_unsigned(num, 16));
```

4.7 Top File

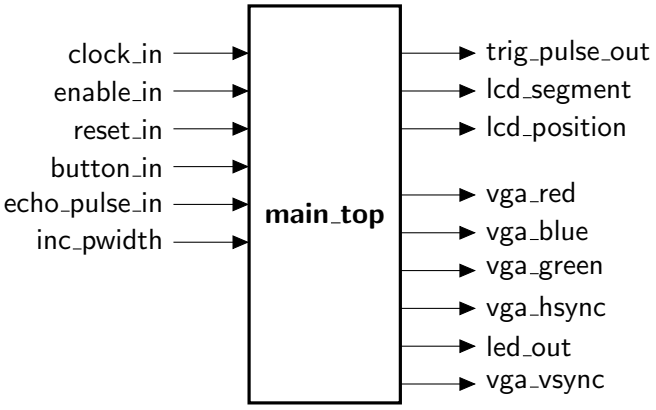


FIGURE 4.18: Entity of main_top

The Top module works as a container for all the modules just explained. The input/output signals are distributed to each of the modules respectively and to the board using the constraint file.

Chapter 5

Simulations and Validation

This chapter describes the process of simulation and tests for the validation of the project. For this phase have been used:

- Oscilloscope: GwInstek GDS-1042 (40 MHz, 250 MSa/s, 2 Ch, USB);
- Signal generator: HAMEG Instruments HM8150 - Programmable Function Generator;

5.1 Clock Generator

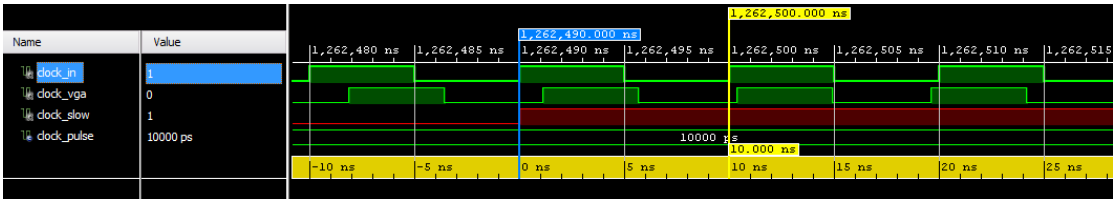


FIGURE 5.1: Input clock simulation

In this section, the clock generation module has been simulated. As we can see, given in input a clock with a period of 10 ns (100MHz), we generate the clock used for VGA (Figure 5.2) with a period of 9.258 ns:

$$f = \frac{1}{9.258ns} = 108014690Hz \approx 108MHz$$

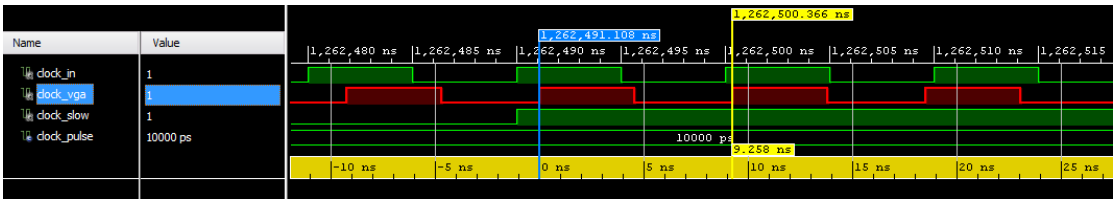


FIGURE 5.2: VGA clock simulation

and a slow clock (Figure 5.3) with a period of 1000ns

$$f = \frac{1}{1000ns} = 1MHz$$

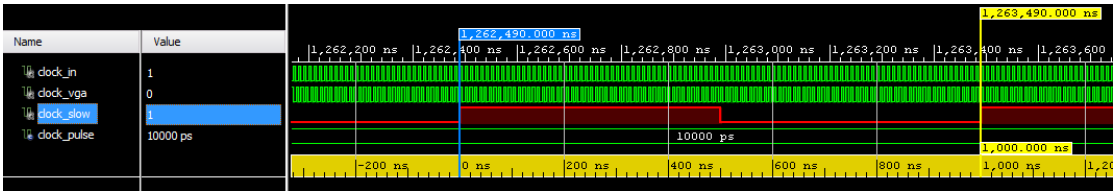


FIGURE 5.3: Slow clock simulation

Clock_pulse is a constant used in the testbench to generate the input clock and represents the period of the clock.

5.2 SRF05

SRF05 module had been simulated generating an echo pulse for every trigger pulse. The echo pulse is increased by $58\mu s$ every iteration to simulate a distance increased by $1cm$ from time to time. In Figure 5.4 this module has been simulated with inc_pwidth set to '0' so, as we can see, the width of the trigger pulse is $15\mu s$.

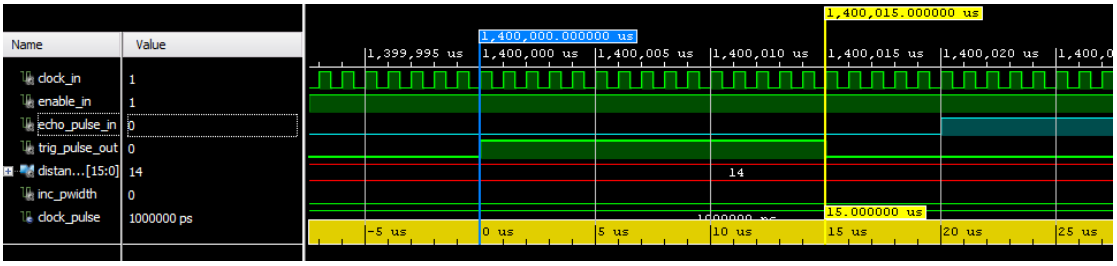


FIGURE 5.4: Trigger pulse simulation

Setting inc_pwidth to '1' (Figure 5.5), trigger pulse signal's width become $20\mu s$ as requested.

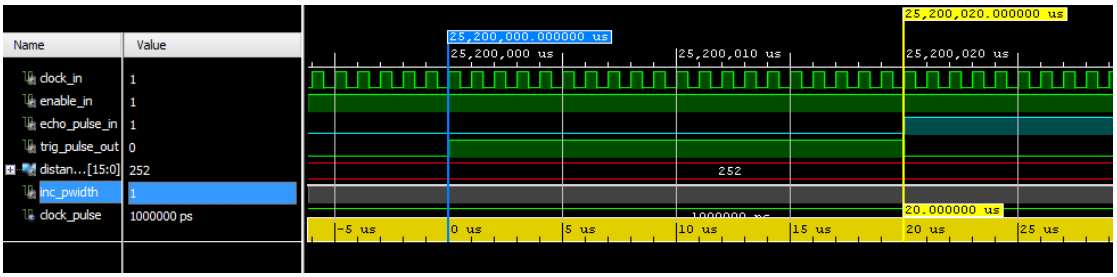


FIGURE 5.5: Trigger pulse simulation with inc_pwidth signal high ('1')

The width of trigger pulse has been tested also with oscilloscope as we can see in FIgure 5.8

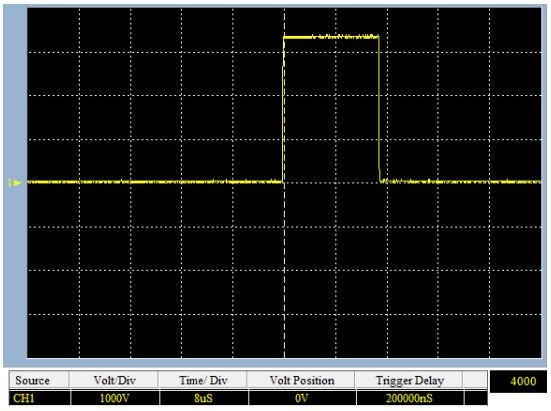


FIGURE 5.6: inc_pwidth signal set to '0'

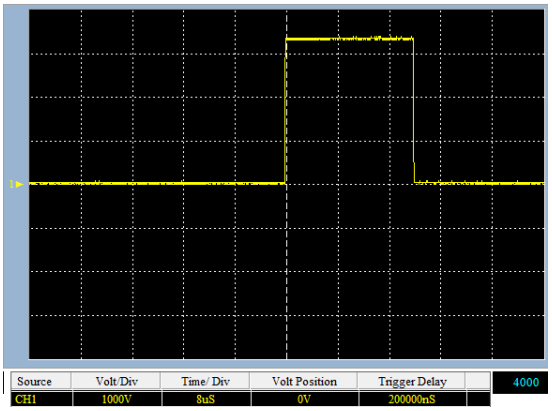


FIGURE 5.7: inc_pwidth signal set to '1'

FIGURE 5.8: Trigger pulse tests using oscilloscope

In Figure 5.9 has been simulated with Vivado an echo pulse with a width of 870μs and the distance obtained is 15 cm as expected:

$$d = \frac{870}{58} = 15cm$$

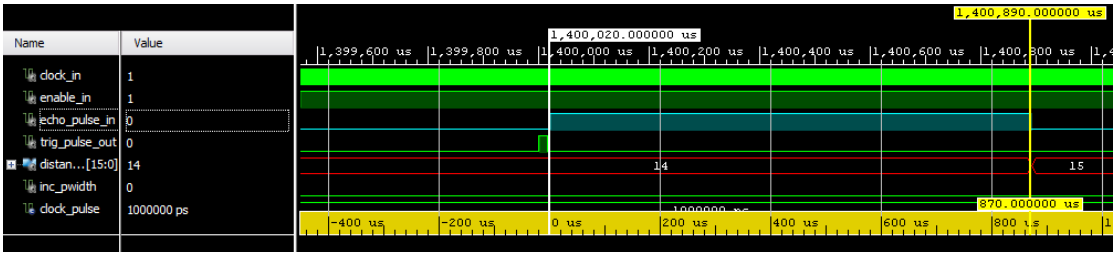


FIGURE 5.9: Echo pulse simulation

To prove that the echo analyser module really works implementing it on the FPGA, the echo module has been tested with the function generator, to simulate the echo signal in input to the FPGA and the onboard 7 segment display to show the results.

FIGURE 5.10: Function Generator

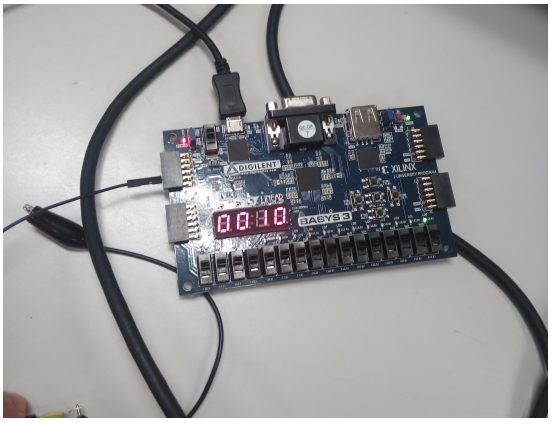


FIGURE 5.11: Result showed on 7-segments display

FIGURE 5.12: Echo pulse analyzer test using the function generator

As we can see in Figure 5.12, the function generator generates an echo signal of $580.0\mu s$ obtaining a measure of $10cm$ showed on the 7-segment display demonstrating that the Echo and Divider modules are working as expected.

$$d = \frac{580}{58} = 10cm$$

5.3 VGA Core

The purpose of the VGA simulator is to act as a real VGA display, therefore, it requires only the standard VGA signals to save one VGA frame as BMP. In this simulation false data are sent to the VGA core to simulate the distances acquired from the sensor filling the RAM with data from 0 to 511.

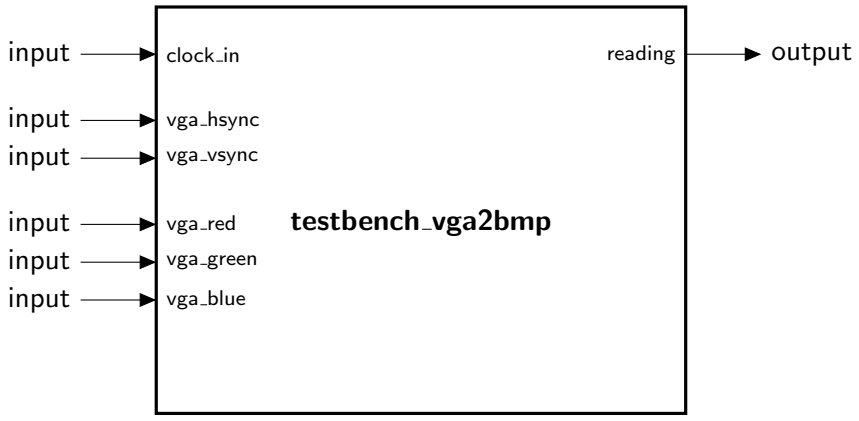


FIGURE 5.13: Entity of testbench_vga2bmp

This testbench component was created with reuse in mind, meaning the only configuration needed is width and height of the display. The module deals with the generation of a BMP using a specific header as shown in Figure 5.14

```
header( 0 to  1) := (0 => character'pos('B'), 1 => character'pos('M')); -- signature
header( 2 to  5) := int_to_4bytes(bmp_size+54); -- total size width*height*3 + 54
header( 6 to  9) := (others => 0);              -- reserved, must be 0
header(10 to 13) := int_to_4bytes(54);          -- offset to start of image data in bytes
header(14 to 17) := int_to_4bytes(40);          -- size of BITMAPINFOHEADER structure, must be 40
header(18 to 21) := int_to_4bytes(bmp_width);   -- image width in pixels
header(22 to 25) := int_to_4bytes(bmp_height);  -- image height in pixels
header(26 to 27) := (0 => 1,  1 => 0);          -- number of planes in the image, must be 1
header(28 to 29) := (0 => 24, 1 => 0);          -- number of bits per pixel (1, 4, 8, or 24)
header(30 to 33) := (others => 0);              -- compression type (0=none, 1=RLE-8, 2=RLE-4)
header(34 to 37) := int_to_4bytes(bmp_size);    -- size of image data in bytes (including padding)
header(38 to 41) := (others => 0);              -- horizontal resolution in pixels per meter (unreliable)
header(42 to 45) := (others => 0);              -- vertical resolution in pixels per meter (unreliable)
header(46 to 49) := (others => 0);              -- number of colors in image, or zero
header(50 to 53) := (others => 0);              -- number of important colors, or zero
```

FIGURE 5.14: Generation of the BMP header

The frame capturing works by waiting for the first vertical sync signal. When the vertical sync is received it compensates for the vertical back porch, before it starts capturing the RGB data between each horizontal sync pulse and horizontal back porch compensation. The RGB data is converted from RGB444 to RGB888 and saved to a 2-dimensional array during capturing. When the frame is full the BMP header is created and the RGB array is reversed and written to the output file. When the module has to read from the RAM there is a delay as shown in Figure 5.15

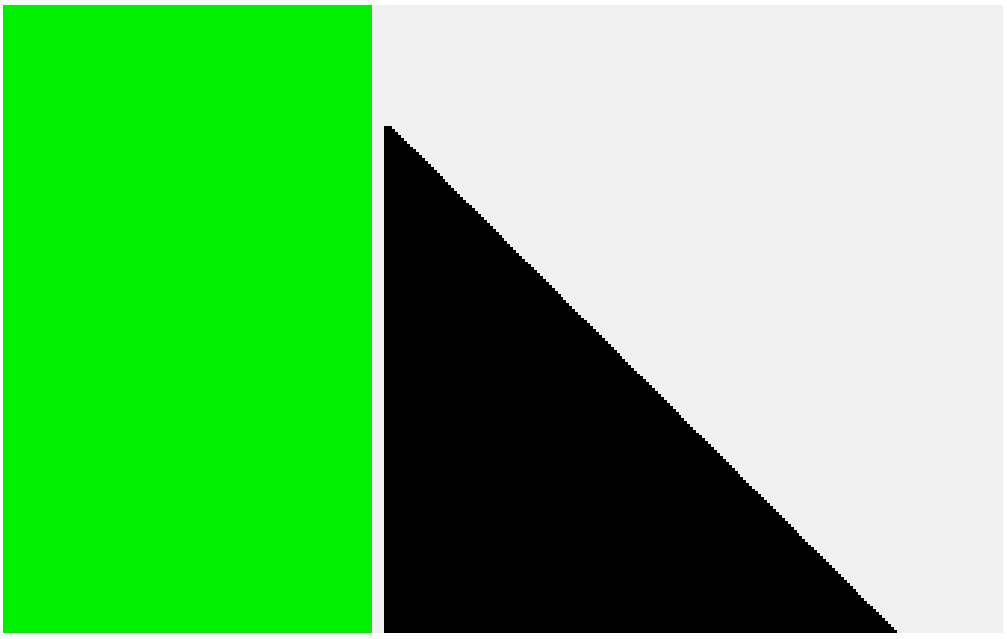


FIGURE 5.15: Output of the VGA with delay

To avoid this problem the address request is advanced and this is the result:

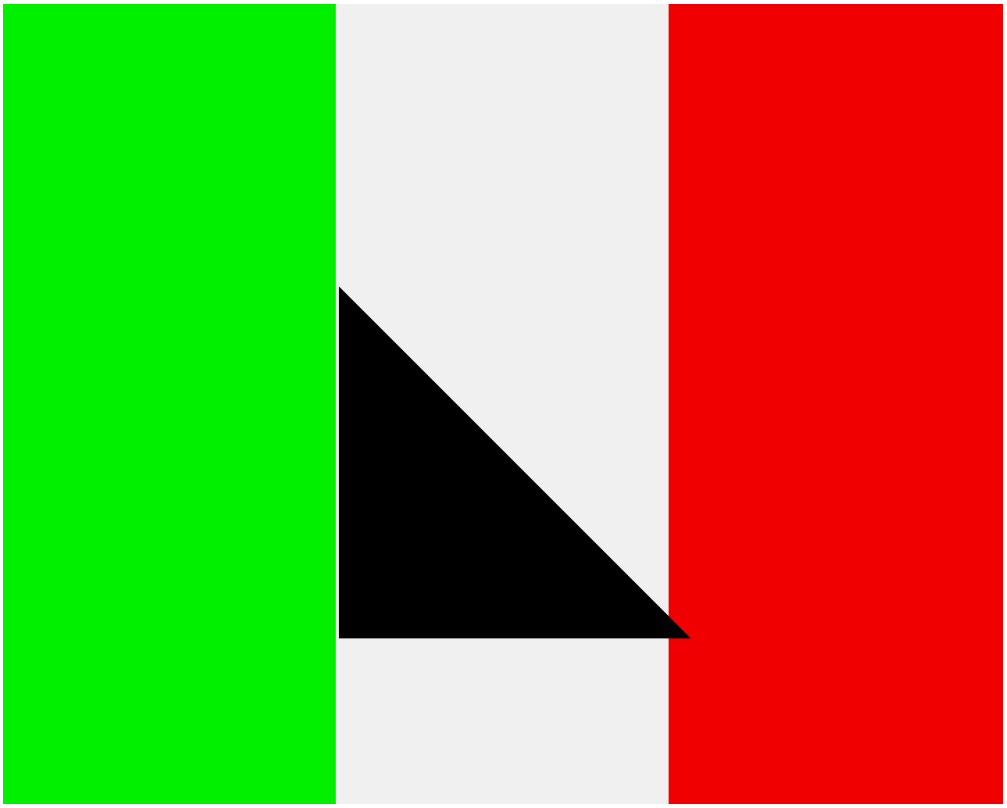


FIGURE 5.16: Output of the VGA without delay

To check if the data written and the size of bars on the graph plotted are correct the following Matlab script has been used:

```
%VGA_output is the input image
%We only need to check the pixels in a specific part of the image
graph=VGA_output(301:300+512,431:430+512,1);
imshow(graph)
%convert black pixels value in 1 and the others in 0
graph(graph==0)=1;
graph(graph==240)=0;
%result contains the total number of black pixels per column
result=sum(graph,1);
```

FIGURE 5.17: Matlab Script to analyze the output image

5.4 RAM

To check the RAM has been created a test bench module that generates signals and tests the output of the RAM module. According to the project requirements to accomplish the goal the RAM module has 512 address locations and 9-bits at each location to store data from 0 to 450 (range sensor limit). The implemented RAM is sensitive to the rising edge of the clock and asynchronous active high reset signal. In Figure 5.18 has been simulated and it is set high for 200ns and in this period all output and internal signals are set to '0'.

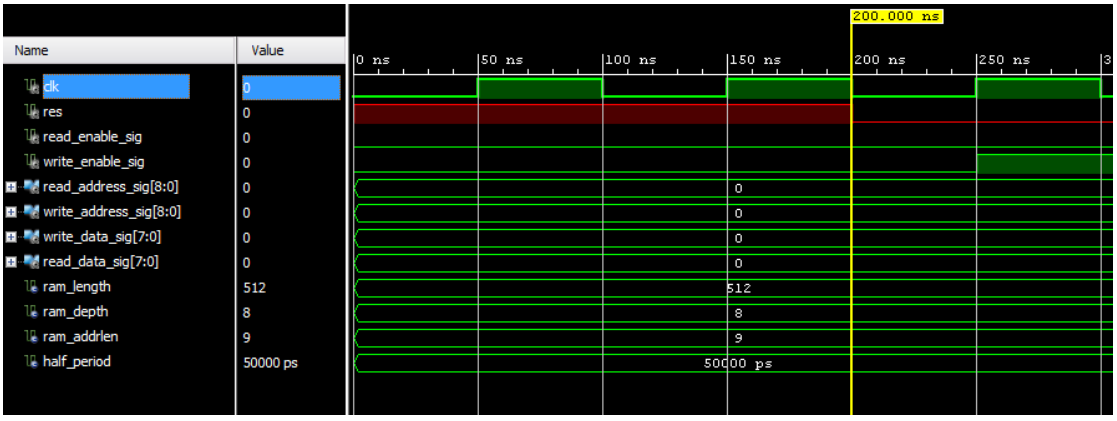


FIGURE 5.18

To test the functions of the RAM a finite state machine has been implemented as requested during the second laboratory. When the reset signal is low, the FSM jumps to Phase A where the RAM starts to write data to memory from 0 to 511. In Figure 5.19 is shown Phase A, and we can see that every rising edge of the clock it write on a N address a N value (in this case N=38)

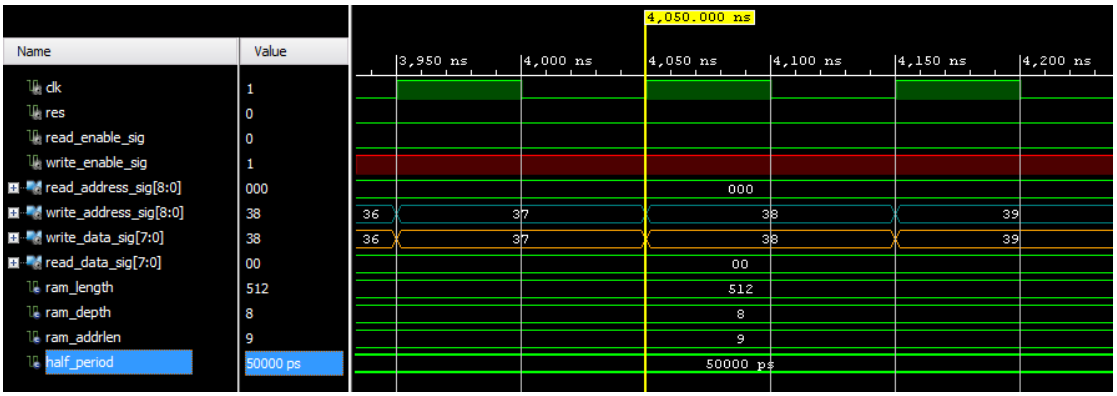


FIGURE 5.19

When Phase A is finished it jumps to Phase AB where the RAM waits for 100 clock cycles (where the input clock has a period of 100ns). In Figure 5.20 is shown this phase, and as we can see, the RAM waits for $100ns$

$$clock\_cycles = \frac{10\mu s}{100ns} = \frac{10\mu s}{0.10\mu s} = 100$$

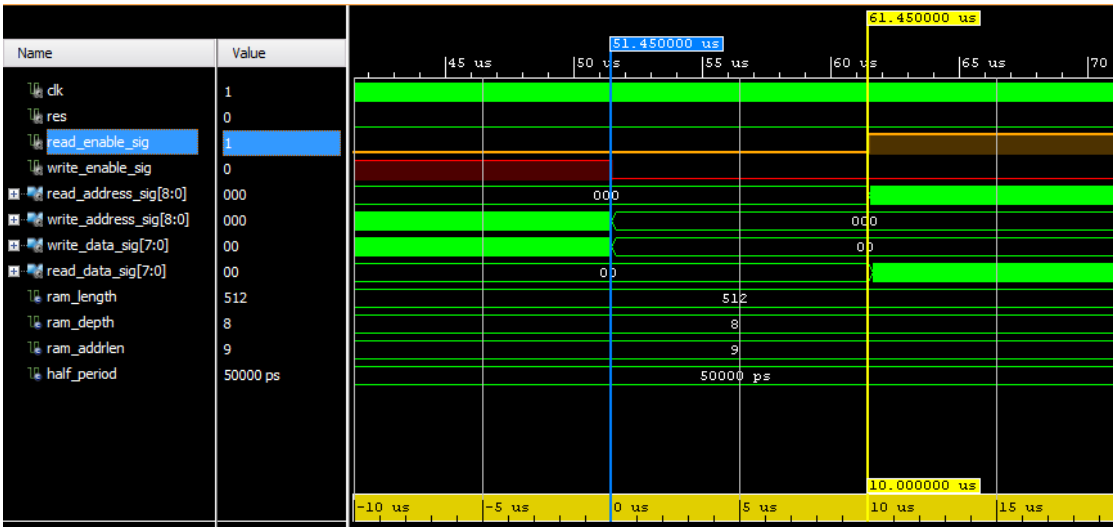


FIGURE 5.20

After this it jumps to Phase B where it reads data from RAM as shown in Figure 5.21.

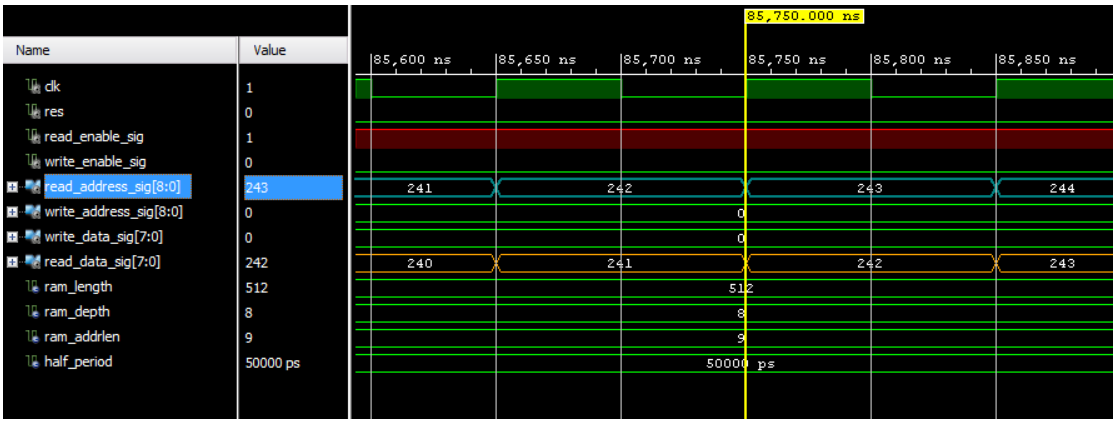


FIGURE 5.21

As we can see, unlike the writing phase, the memory gets from the address specified in the signal read_addres the value stored in it with 1 clock cycle delay.

5.5 7 segments display and LED core

The 7-Segment and LED testbench was simulated using as input a distance value.

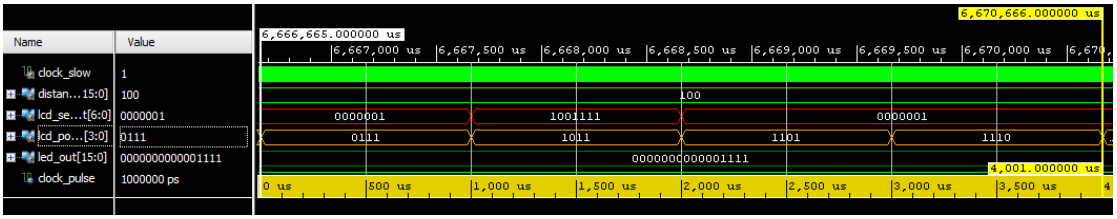
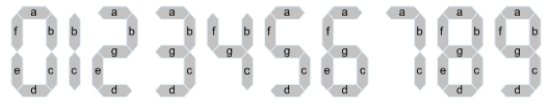


FIGURE 5.22: Simulation of 7-segments display



Individual Segments							Display
a	b	c	d	e	f	g	
×	×	×	×	×	×		0
	×	×					1
×	×		×	×		×	2
×	×	×	×			×	3
	×	×			×	×	4
×		×	×		×	×	5
×		×	×	×	×	×	6
×	×	×					7

Individual Segments							Display
a	b	c	d	e	f	g	
×	×	×	×	×	×	×	8
×	×	×	×		×	×	9
×	×	×		×	×	×	A
		×	×	×	×	×	b
×			×	×	×		C
	×	×	×	×		×	d
×			×	×	×	×	E
×				×	×	×	F

FIGURE 5.23

Figure 5.22, according with Figure 5.23, show that the indicators are working as expected.

5.6 Top Module

The testbench for the top file simulates an echo pulse directly instead of waiting for the trigger pulse that requires long time. This echo pulse is 0.638 ms wide and should return a measured distance of 11 cm.

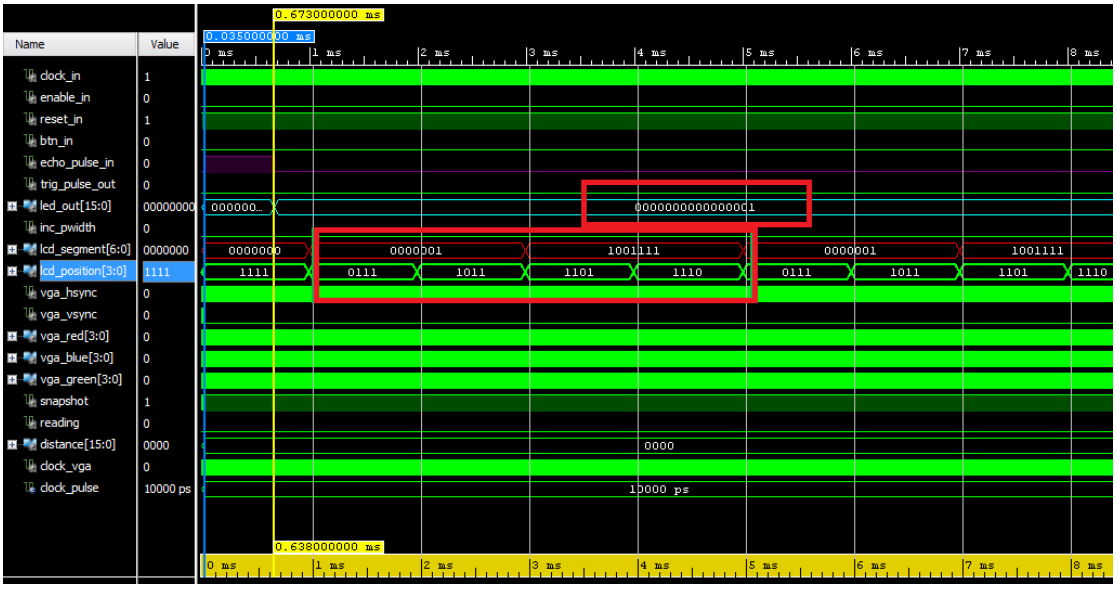


FIGURE 5.24: Simulation of Top Module with 0.638 ms width long echo pulse as input

Figure 5.24 shows that the LED value is: 0000000000000001 and since each LED represent 11 cm, this corresponds to a distance between 1 to 30 cm which is correct. Taking a closer look at the 7-segment display output and comparing it against Figure 5.23 gives the number 11 which is correct. Furthermore, analyzing the BMP output image using the Matlab script shown in Figure 5.17, we check that the column plotted on the graph is 11 pixels high as expected (Figure 5.26).

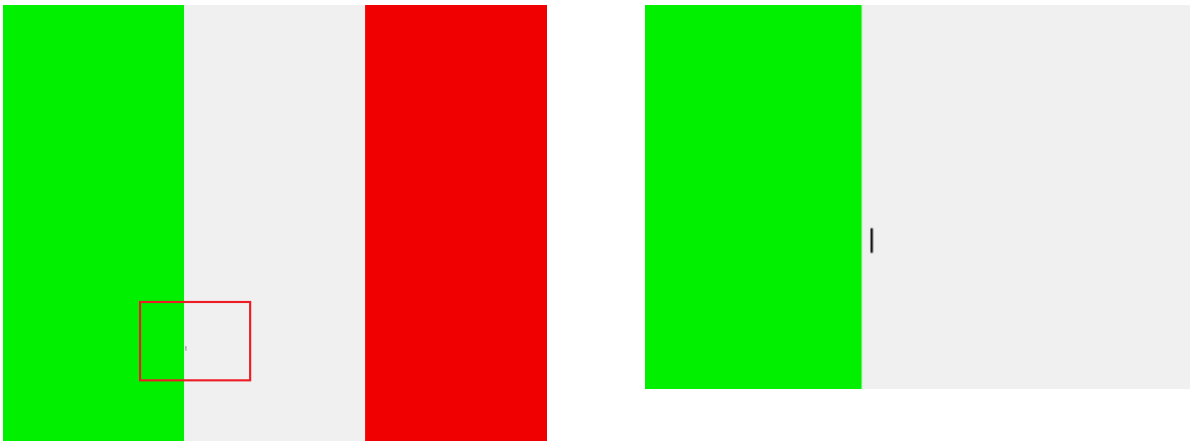


FIGURE 5.25

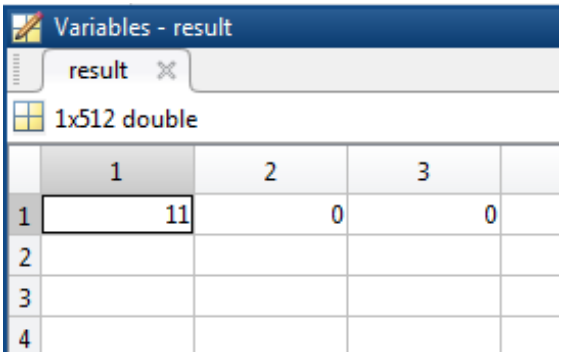


FIGURE 5.26: Value of variable 'result' containing the high of the black columns plotted on the BMP output file

Chapter 6

Conclusions

6.1 Results evaluation

After setting up the system, it has been tested in order to estimate the error of our system. The maximum detected distance is up to 400 cm and because of power issues (different provided voltage, 3.3V instead of 5.0V as requested) the minimum detected distance is up to 7 cm.

Figure 6.1

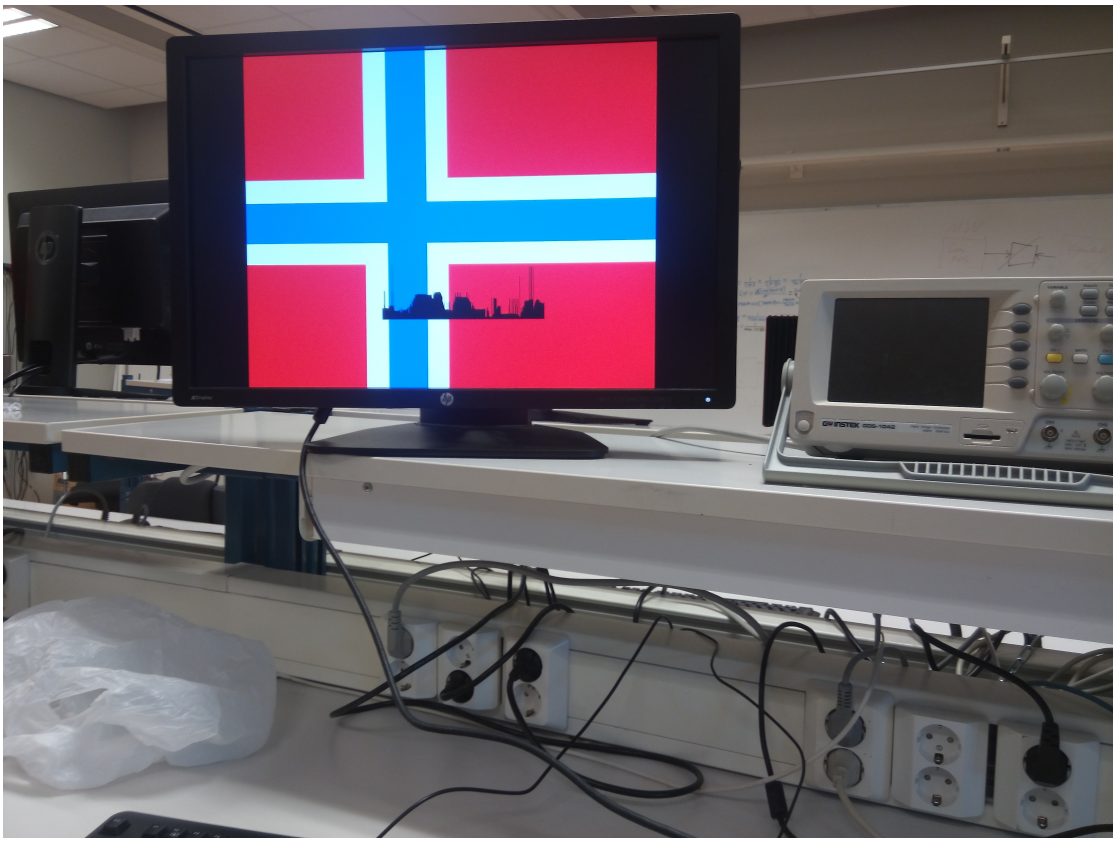


FIGURE 6.2

To estimate the error of the the system some measures have been taken as shown in Figure 6.3.

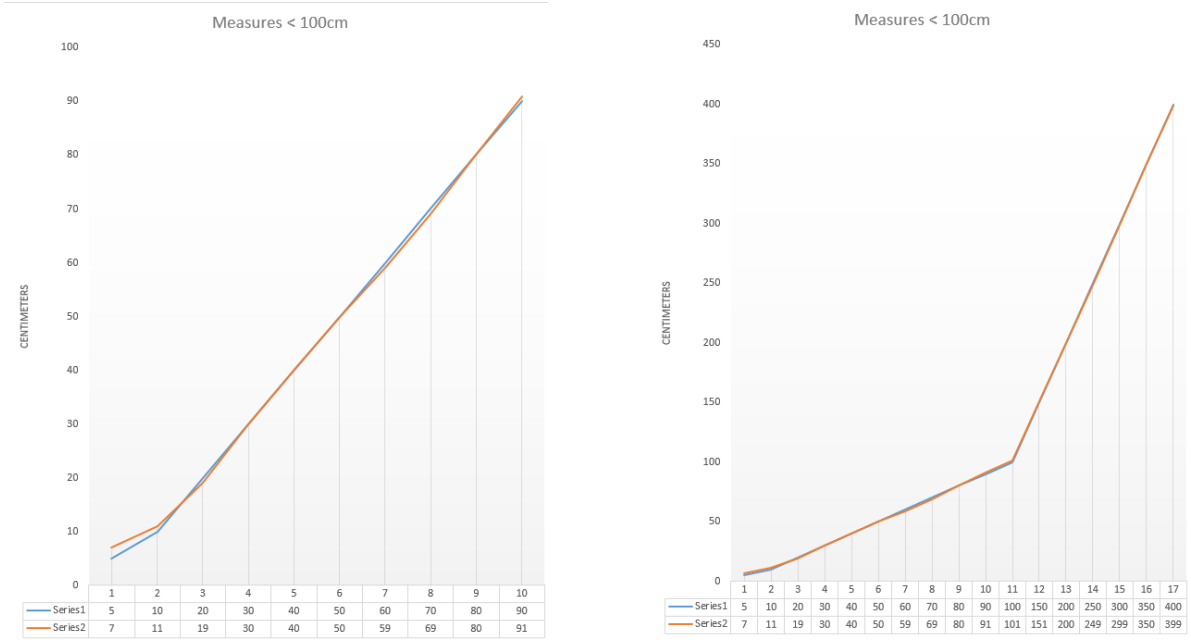


FIGURE 6.3: Diagram of measures: Series 1 is the data series of the real distance and Series 2 the detected measures.

The mean value of the relative errors given by the test is 3.5%, particularly affected by the first measurement that generates a high error due to the sensor limit for the wrong voltage. Leaving aside that measurement, the relative error is 1.37% and the deviation is 1 cm in all cases.

6.2 FPGA Resources

By looking at Figure 6.4 we can see that the MMCM, mixed-mode clock manager, is causing the majority of the power consumption.

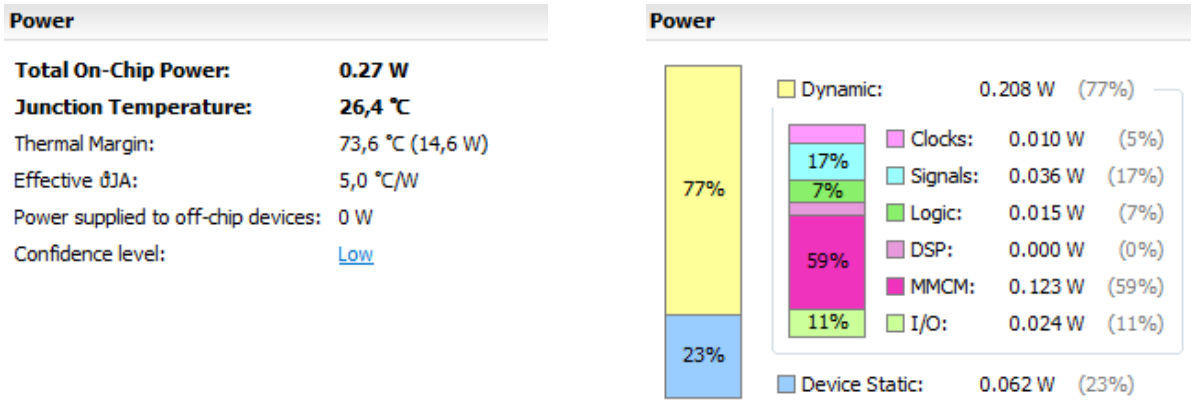
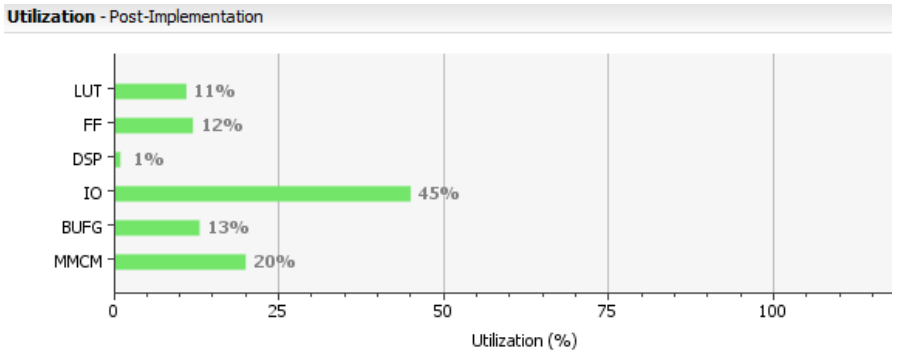


FIGURE 6.4: Power consumption



Resource	Utilization	Available	Utilization %
LUT	2259	20800	10.86
FF	5034	41600	12.10
DSP	1	90	1.11
IO	48	106	45.28
BUFG	4	32	12.50
MMCM	1	5	20.00

FIGURE 6.5: Resource utilization

List of Figures

List of Tables

References

[1] URL http://www.electronics-tutorials.ws/combination/comb_6.html.

[2] URL https://reference.digilentinc.com/basys3/refmanual.

[3] URL https://eewiki.net/pages/viewpage.action?pageId=15925278.

[4] URL https://www.xilinx.com/support/documentation/user_guides/ug472_7Series_Clocking.pdf.

[5] URL https://elearn20.miun.se/moodle/pluginfile.php/422771/course/section/56707/VHDLreport.pdf.