TE3060 Embedded Systems Laboratory

FIRST PARTIAL PROJECT

Hardware Specifications and Design

SEPTEMBER 20, 2020

Revision History

Rev. #	Date	Changes By	Description
0.1	09/10/20	Gabriela Hernández	Starts Excel for the Multiplication of 16x16 and first draft.
0.2	09/11/20	Vanessa Jaime	Completes the multiplication 16x16.
0.3	09/11/20	Gabriela Hernández	Update the multiplication 16x16 and verifies its correct behavior with the test bench.
0.4	09/16/20	Isaac Ipenza	Start the document and complete specificacion and introduction
0.4	09/16/20	Vanessa Jaime	First draft of ALU and divider. Draft of the State Machine of the divider. Draft of the block diagram of the divider.
0.5	09/19/20	Isaac Ipenza	Completes the ALU.
1.0	09/19/20	Vanessa Jaime	Completes the divider.
1.0	09/19/20	Isaac Ipenza	Complete the document.

Table of Contents

Table of Figures

1.0 Introduction

This is the first partial project for the Embedded System Laboratory class, this consists of making use of our knowledge in Verilog HDL to code an ALU which has inside a multiplier and a divider made with state machines. This is a step by step project. We needed to do first the multiplier, then the divider, and finally, we use these two previous codes to their respective operations in the ALU.

The complete project consist in a document with all de documentation of each one of the three tasks, that must include the specifications and designs of each one of these, the Verilog HDL code of each task with any other code needed to make them work and the test benches codes for this codes and a .do code that allows us to test it faster.

1.1 Goals of the project

The main goal of this project is to put in practice all the knowledge of Verilog HDL that we saw in the first partial classes to implement a task from zero; we must do each task from the first block design to the final code that meets all the requirements.

1.1.1 16x16 Multiplier

Design and implement up to simulation stage a 16x16 multiplier making use of the 8x8 multiplier that we do in class.

1.1.2 16x16 Divider

Design and implement up to simulation stage a 16x16 divider with the shift and subtraction approach.

1.1.3 ALU

Design and implement up to simulation stage a clocked ALU using the 16x16 bit multiplier and the 16 bit divider.

The operations that the ALU must support are:

- Addition / Subtraction
- Multiplication / Division
- Bitwise AND, OR, NAND, NOR, XOR
- Shift left / Shift right
- Circular shift left / Circular shift right
- Greater / smaller /equal comparison
- Carry out flag
- Zero flag

1.2 Scope

In this document you will find all the specifications of the first partial laboratory project, and most importantly all the design documentation and how this was implemented each one of the tasks in this project.The document contains the different diagrams needed to understand how this project works, and how each task is connected internally.

1.3 Intended Audience

This document can be read by anyone who is interested in learning more about programming in Verilog HDL, because in this document you can find all the details to reach 2 basic projects, and how to re-use them in another bigger project.

1.4 Terminology

Acronym	Description
ADC	Analog to Digital Converter
ALU	Arithmetic Logic Unit
DAC	Digital to Analog Converter
DDR	Double Data Rate
DI	Data Invalid
DMA	Direct Memory Access
DSP	Digital Signal Processing
EMIO	Extended Multiplexed I/O
FPGA	Field Programmable Gate Array
GEM	Gigabit Ethernet MAC
GMII	Gigabit Media Independent Interface
GUI	Graphical User Interface
HDL	Hardware Description Language
HLS	High Level Synthesis
LEA	Low Energy Analog
LED	Light Emitting Diode
LS	Light Source
LSB	Least Significant Bit
LVDS	Low-Voltage Differential Signaling
MAC	Medium Access Control
Mbps	Mega Bits per Second
MII	Media-Independent Interface
MMCM	Mixed-Mode Clock Manager
MMS	Manufacture Messaging Specification
MSB	Most Significant Bit
NCO	Numerically Controlled Oscillator
OTR	Out of Range
PC	Personal Computer
PD	Photo Detector
PHY	Physical Layer
PL	Programmable Logic
PLL	Phase Locked Loop
PPS	Pulse Per Second
PS	Processing System

RMII	Reduced Media Independent Interface
Rx	Reception
SFP	Small Form-Factor Pluggable Transceptor
SoC	System on Chip
SoW	Statement of Work
Tx	Transmission

Table 1 Acronyms

16x16 Multiplier

2.0 16x16 multiplier Requirements

This project multiplies 2 numbers of 16 bits each giving a result of 32 bits.

2.1 Block diagram

High-level block diagram of the system requirements and description of each system component

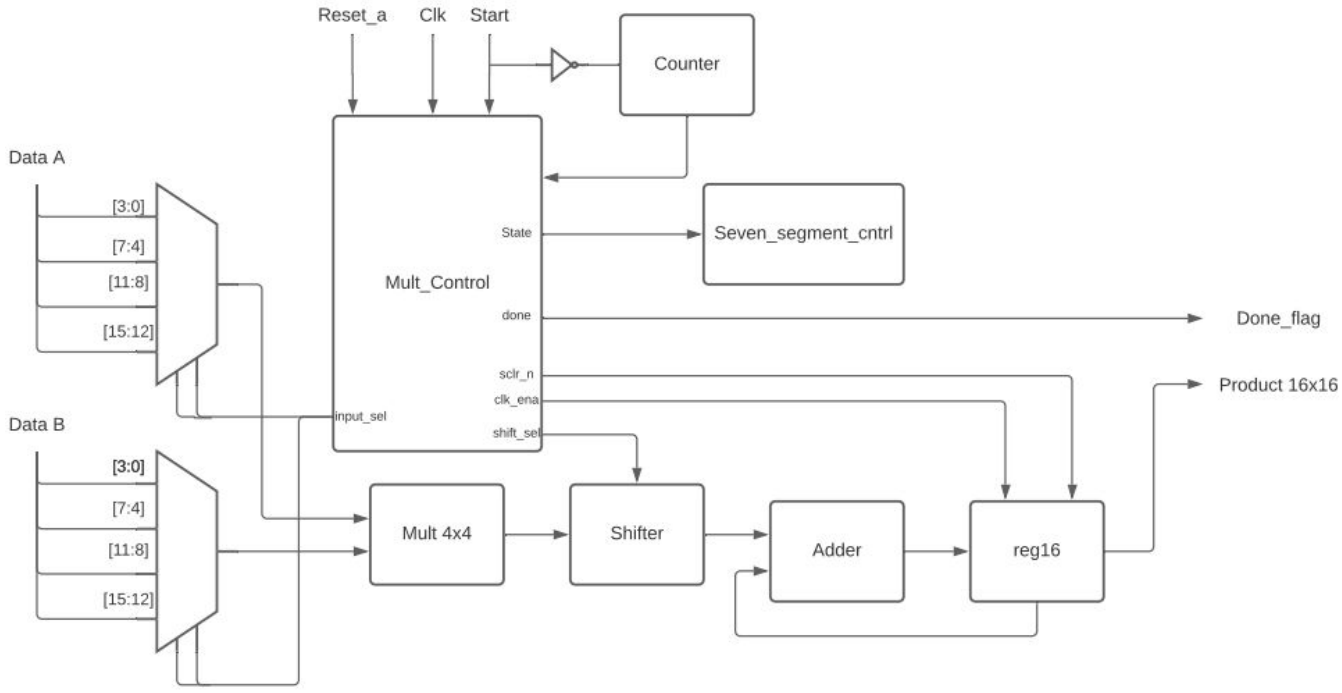

Figure 1 - 16x16 multiplier High level diagram

2.2 Overall functionality

This multiplier goes 4 by 4 multiplying and adding until it has all the multiplications of one power (for example 2^0) and then goes and multiply again all the numbers that goes in the next power and so until it arrives to the end. The next image gives a better example of how it works.

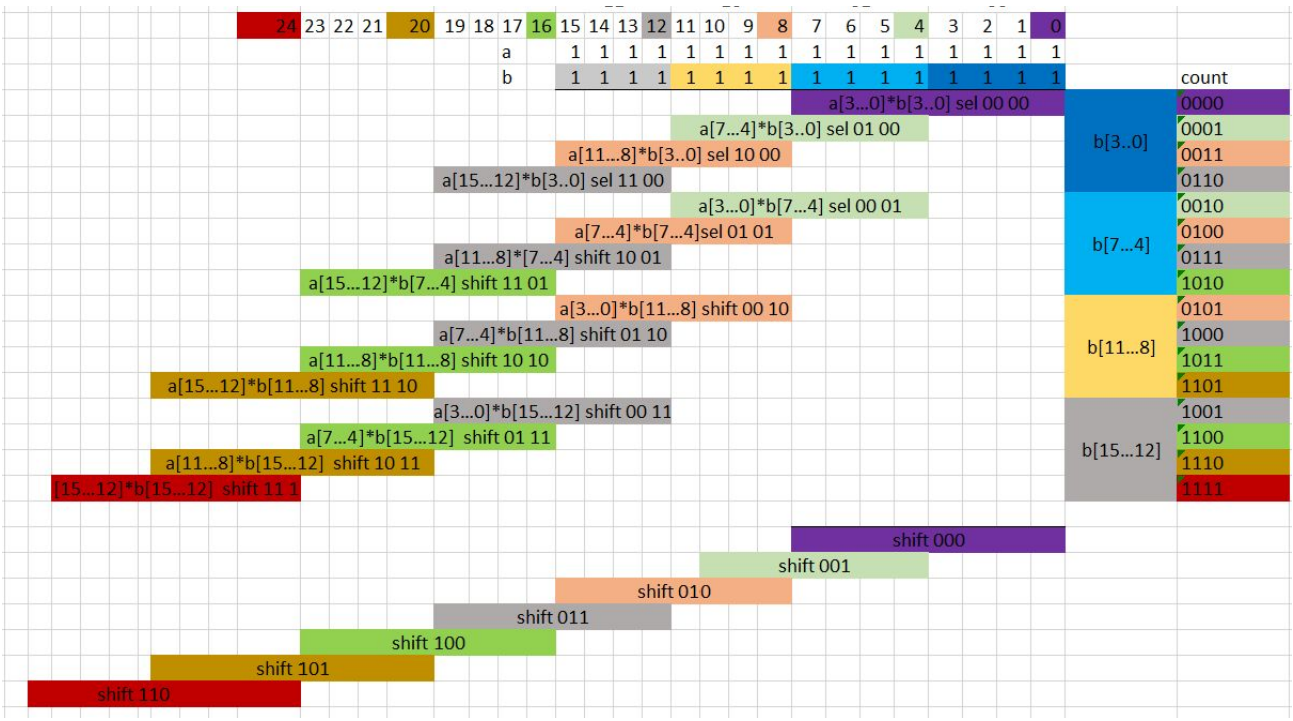

Figure 2 - 16x16 multiplication excel scheme

3.0 16x16 multiplier Design Description

In this project we have 4 main blocks, one counter and one seven segment control which helps to shows which state we are on in a seven segment display. The 4 main blocks contains: mult_control, this block control all the signals need it besides the inputs, mult 4x4 where a multiplication at a time is held multiplying only 4 bits by 4 bits, a shifter in order to put out result where is need it and an adder which adds all the results from the same power at the time.

3.1 Intricacies of the hardware architecture

The main intricacies were that we were required to use a 4x4 multiplier only making the multiplier bigger than expected with more states.

3.2 Hardware architecture at signal level

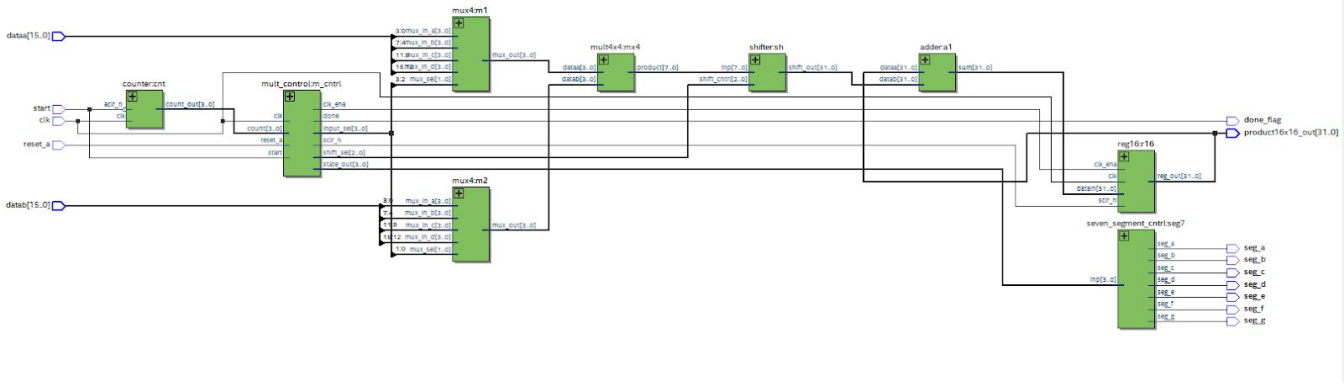

Figure 3 - 16x16 multiplier Hardware architecture at signal level

3.3 FSM controllers

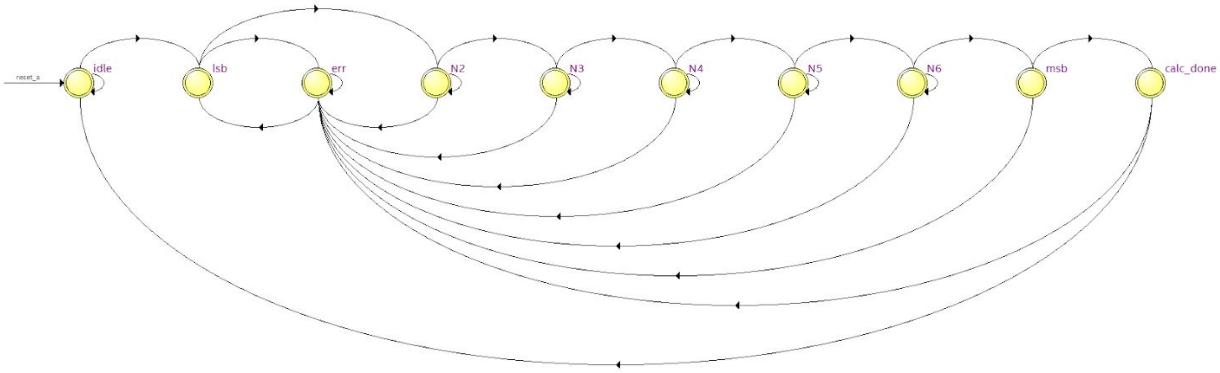

Figure 4 - 16x16 multiplier FSM

4.0 16x16 multiplier Test and Test Results

4.1 Test Strategy

In this project we did a test bench in order to test it. This test bench is called multi16x16_tb.v. In this test bench we enter some important inputs as reset and start in order to simulate our multiplier. We also set the clk to ticks every certain time. There are two numbers that are the inputs which change every done flag in order to test several times with different numbers our project.

4.2 Test Results

We multiply random numbers that were increasing when the first multiplication had done and all the cases were successful.

4.2.1 Synthesis results

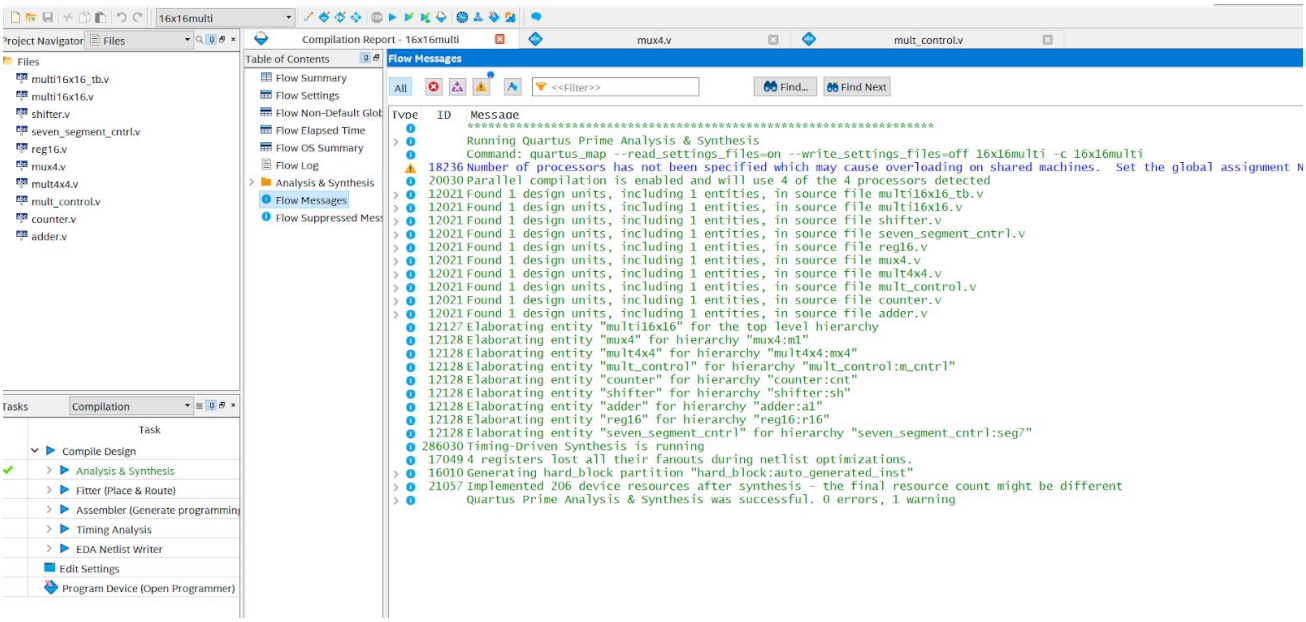

Figure 5 - 16x16 multiplier code synthesis results

4.2.2 Simulation results

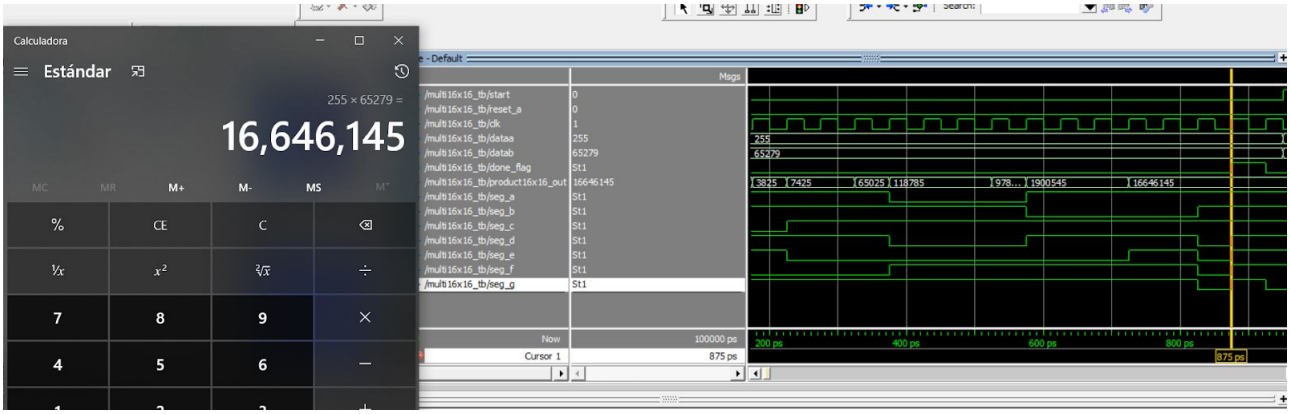

Figure 6 - 16x16 multiplier Simulation result 1

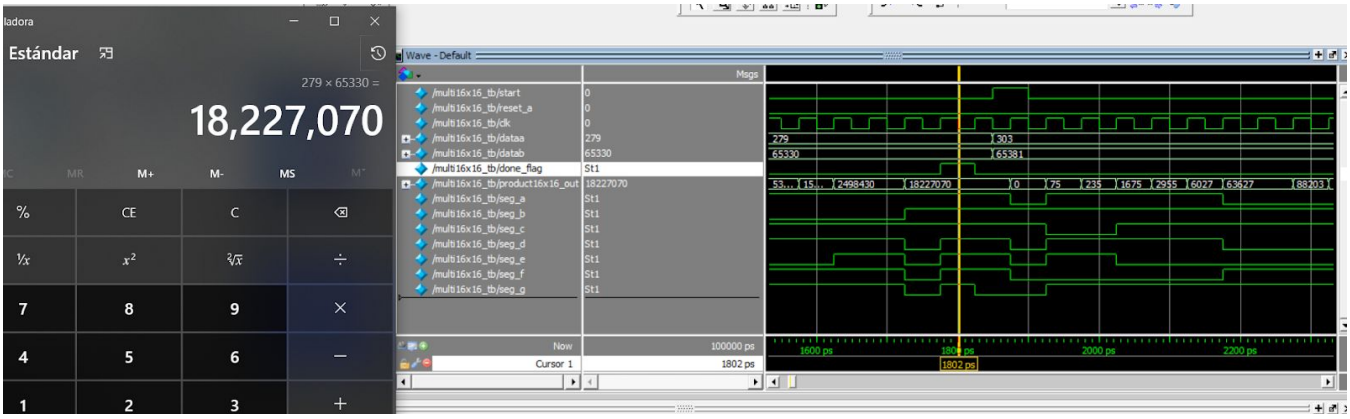

Figure 7 - 16x16 multiplier Simulation result 2

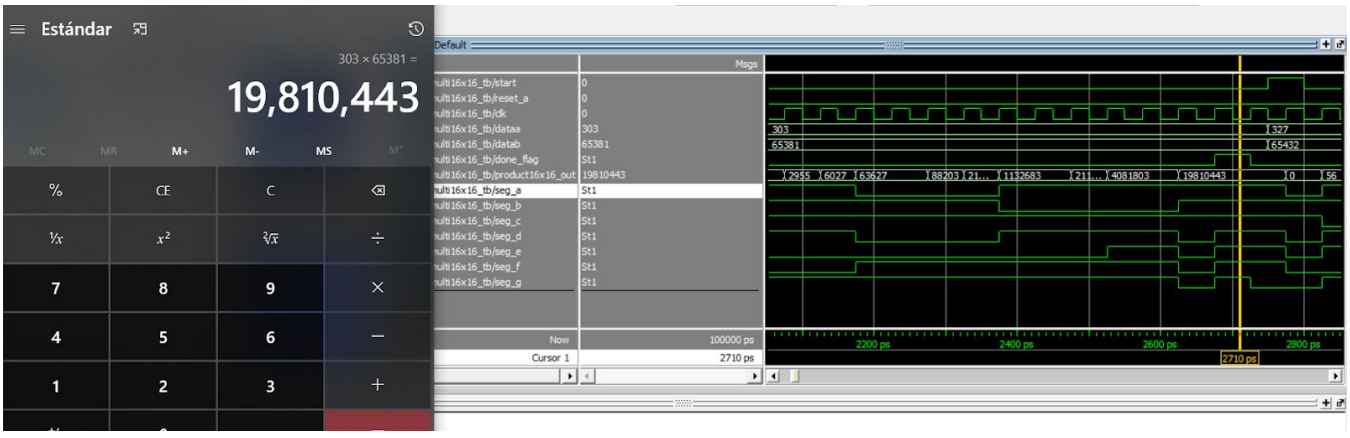

Figure 8 - 16x16 multiplier Simulation result 3

16x16 Divider

5.0 16x16 divider Requirements

This project divides one 16 bit number with another 16 bit number giving a result using a subtraction and shifting approach.

5.1 Block diagram

High-level block diagram of the system requirements and description of each system component

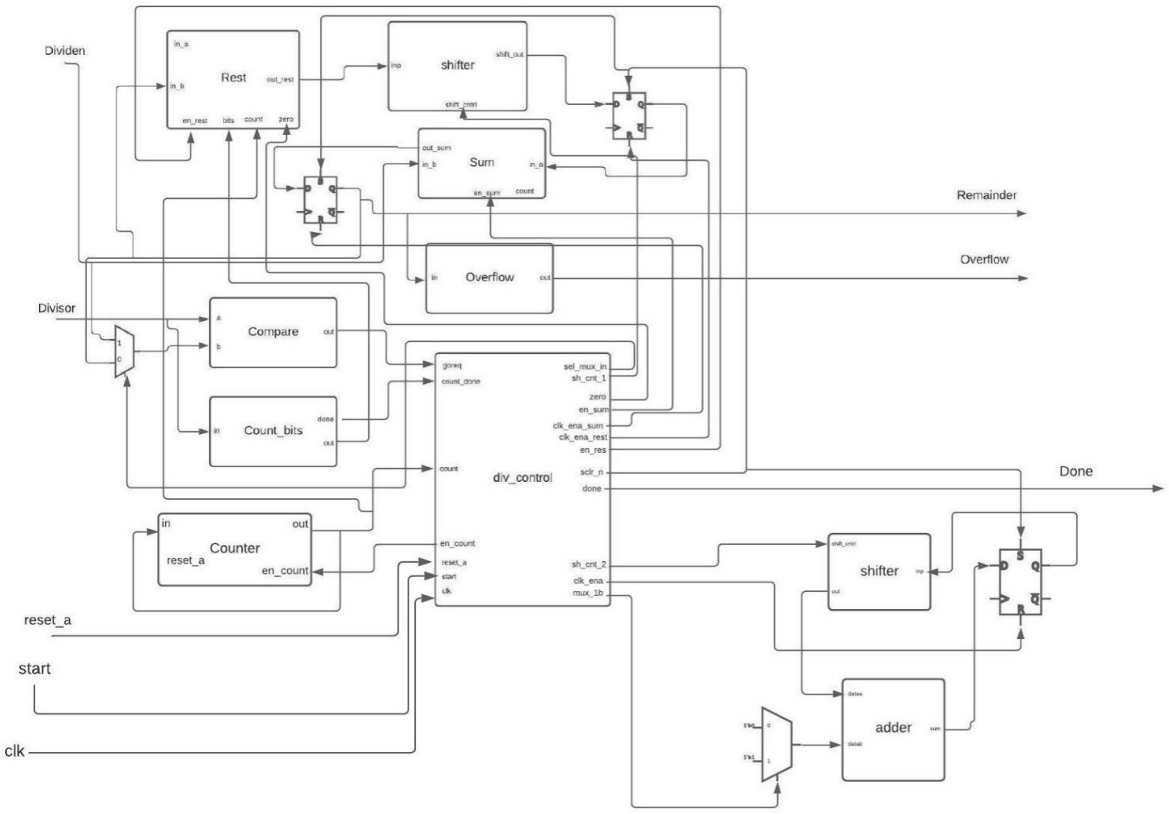

Figure 9 - 16x16 divider High level diagram

5.2 Overall functionality

This divider works as how we learnt it for the first time. First it checks if the dividend is bigger than the divisor, then it goes bit by bit checking if it's bigger adding zeros to the quotient and adding the result of the subtraction on the remainder. We also use 2 shifters, one for the remainder and the other for the quotient.

6.0 16x16 Divider Design Description

The most important blocks of this project are: rest which makes the subtraction between the divisor and the dividen, the shifter which helps us to get the right result and the sum block which sums the remainder with the next bit in line of the dividend.

6.1 All the intricacies of the hardware architecture

We had a delay of one cycle after the done flag was up and the reason was because of the register that gave this signal one clock after. We fixed it by adding a register at the end of the signal done.

We also have problems with the signals because we first tried a Mealy machine, but then, we realized that due to our design it was better a Moore machine so we changed it and gave us the right signals.

6.2 Hardware architecture at signal level

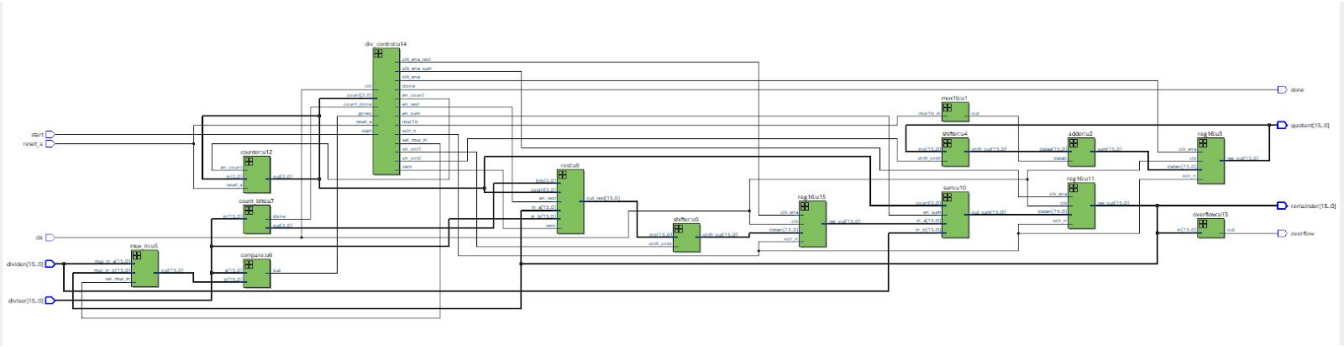

Figure 10 - 16x16 divider Hardware architecture at signal level

6.3 FSM controllers

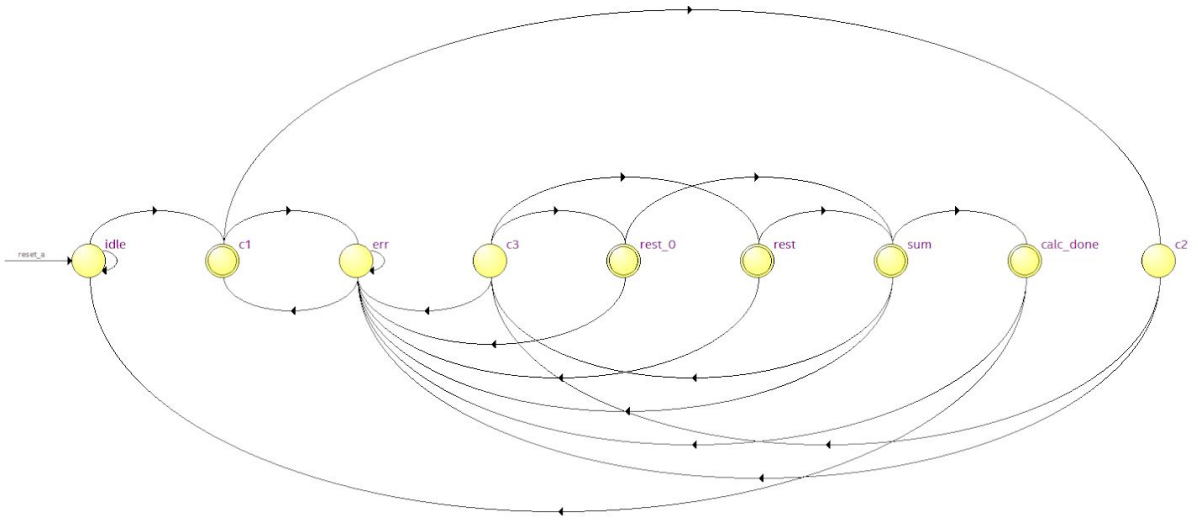

Figure 11 - 16x16 divider FSM

7.0 16x16 Divider Test and Test Results

7.1 Test Strategy

This part we tested it with a test bench name: divider_tb.v. The most significant signals to be tested were: quotient and reminder.

7.2 Test Results

We did 3 cases. One is when the divisor is bigger than the divider should go to the error state (1000) and this test passed. The second one was when the division has no overflow, therefore, the remainder is equal 0, and this test also passed. And the last one is when we have an overflow and this test passed too.

7.2.1 Synthesis results

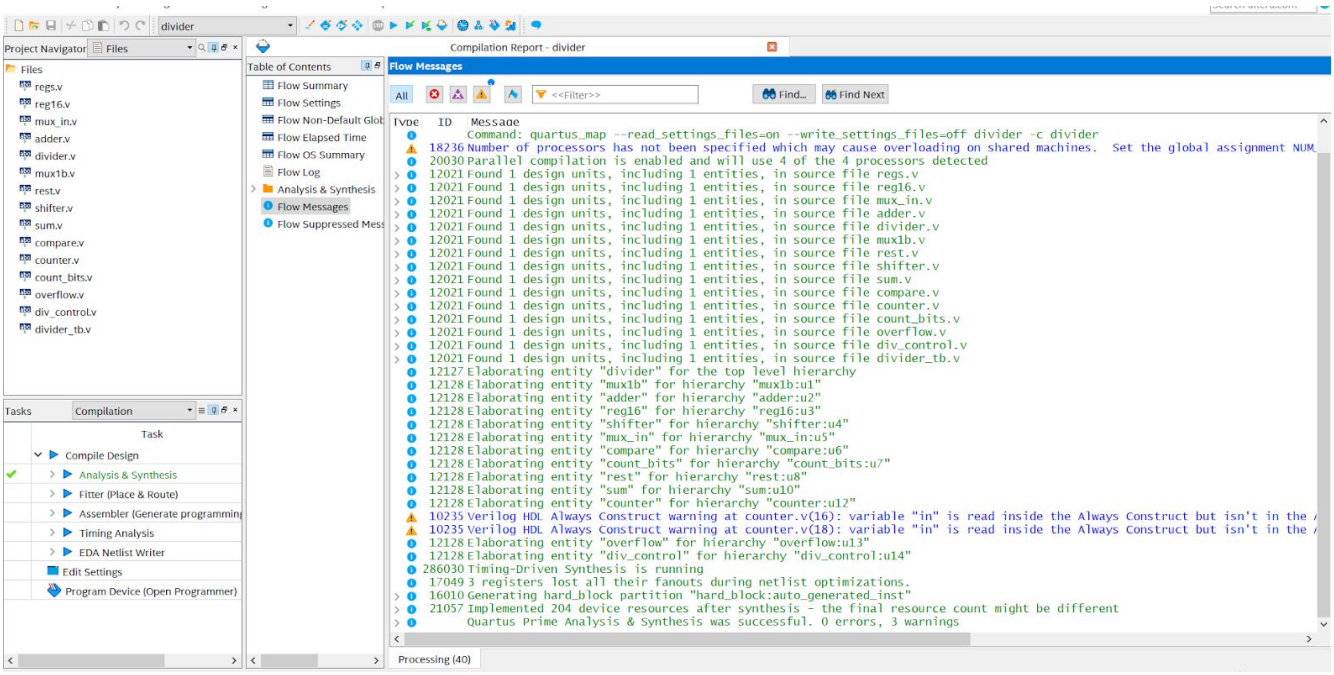

Figure 12 - 16x16 divider code Synthesis results

The reason for the 2 last warnings is because the output of the counter block we want it to depend extrictly from its enable and not by the change of the input.

7.2.2 Simulation results

7.2.2.1 Dividing 2 / 5 ;

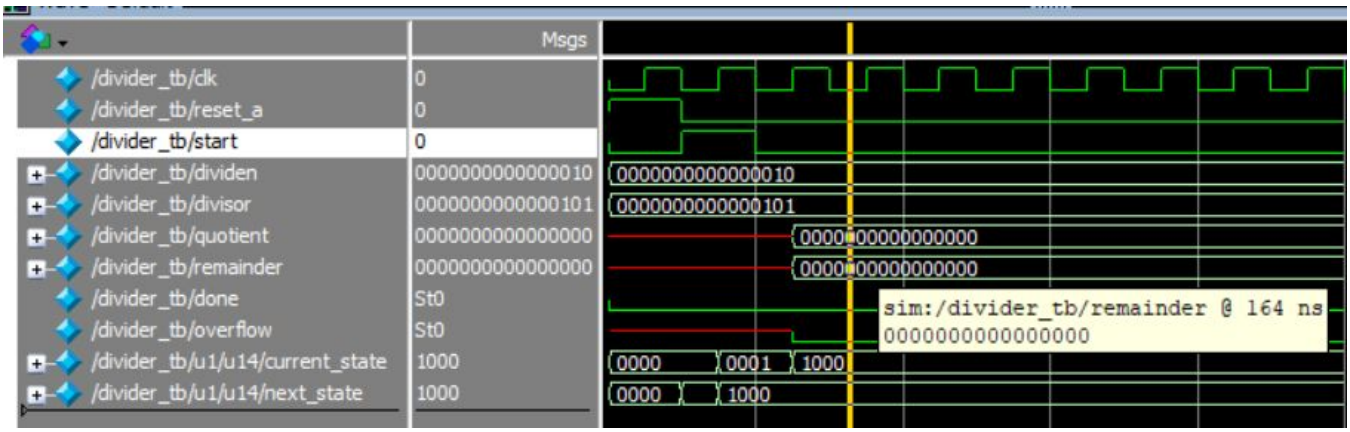

Figure 13 - 16x16 divider Simulation result 1a

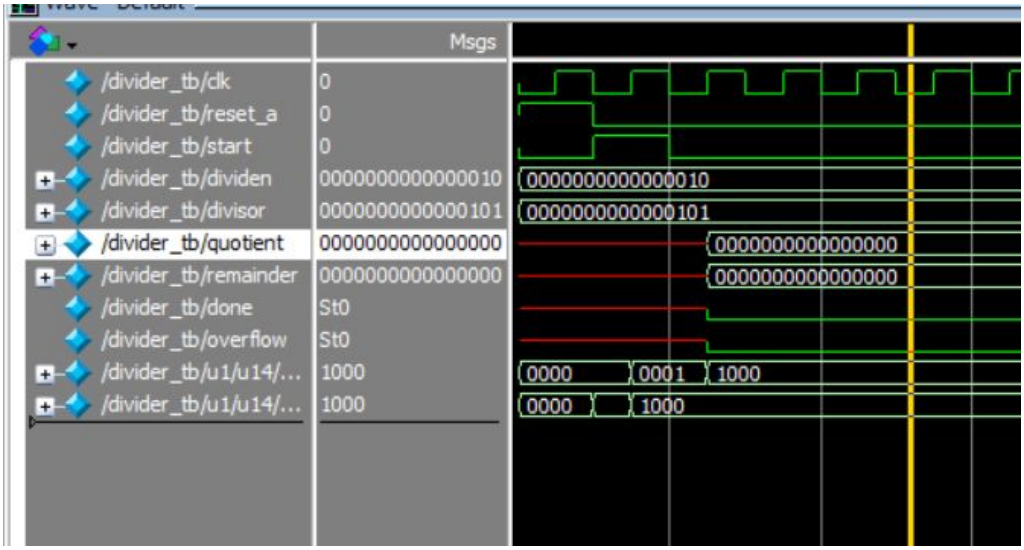

Figure 14 - 16x16 divider Simulation result 1b

7.2.2.2 Dividing 5 / 2 ;

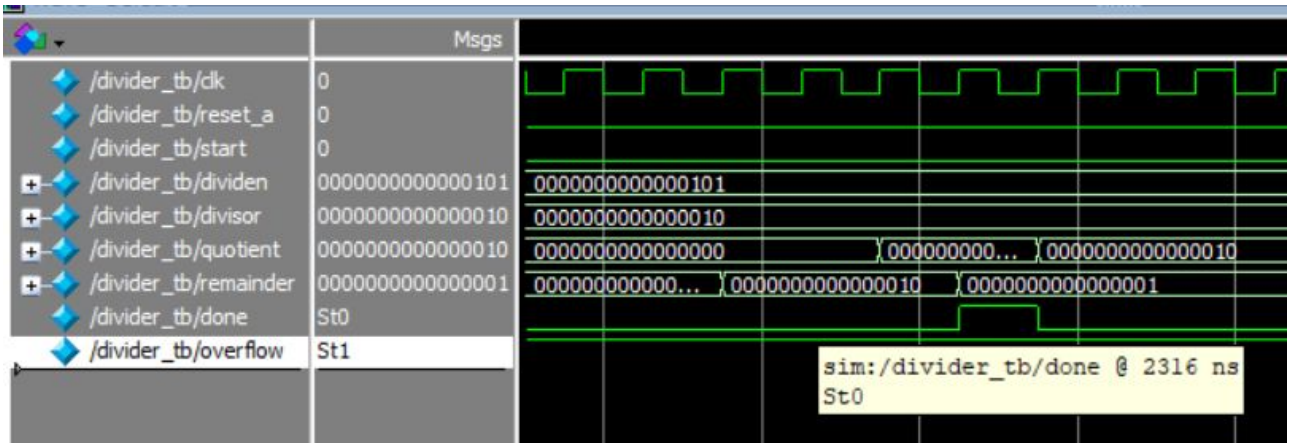

Figure 15 - 16x16 divider Simulation result 2a

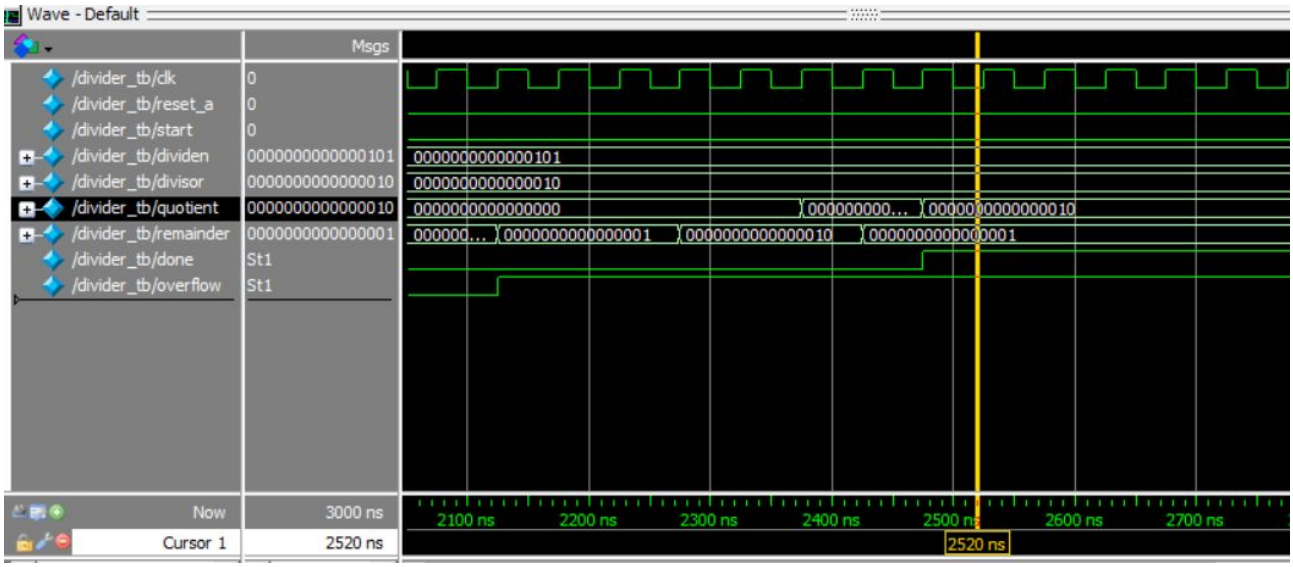

Figure 16 - 16x16 divider Simulation result 2b

7.2.2.3 Dividing 8/2:

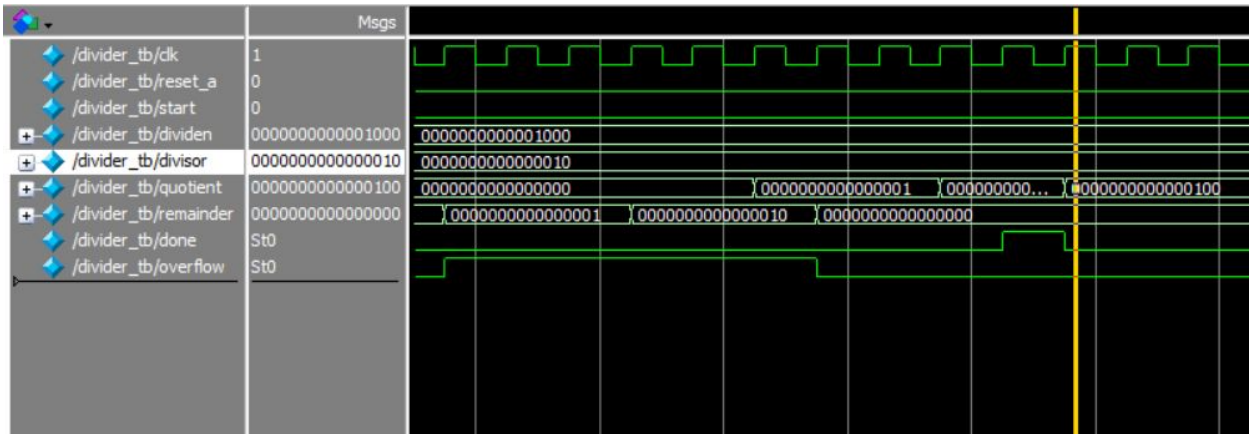

Figure 17 - 16x16 divider Simulation result 3a

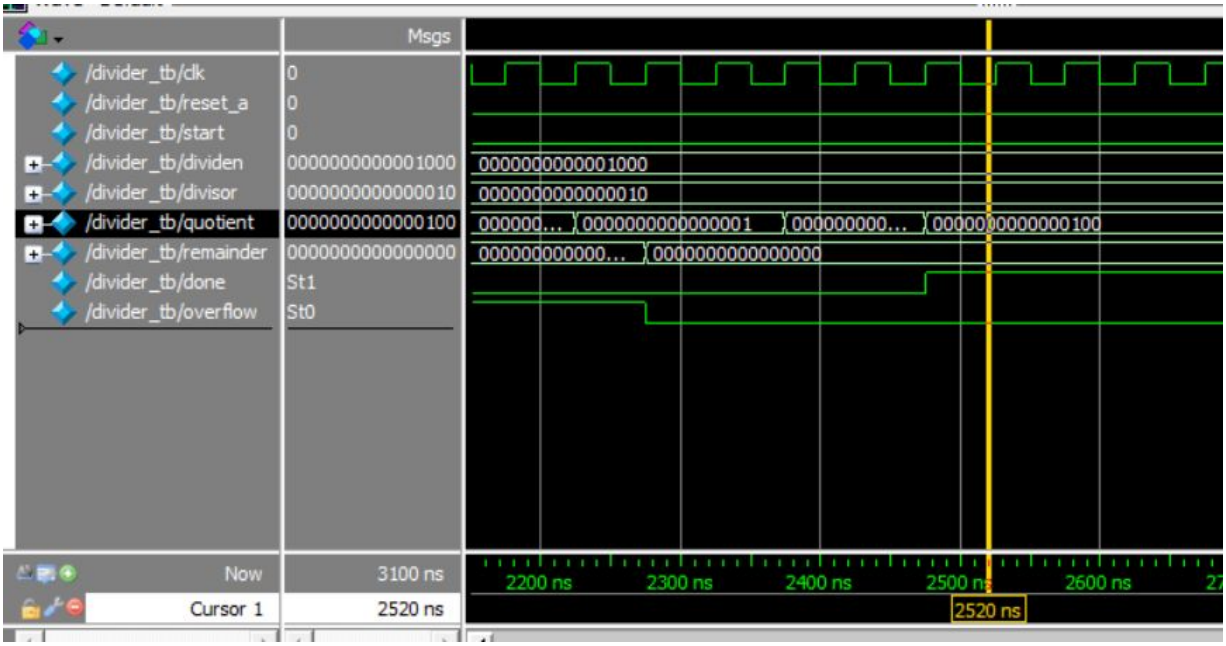

Figure 18 - 16x16 divider Simulation result 3b

ALU

8.0 ALU Requirements

This project is an arithmetic logic unit (ALU), this is a combinational digital circuit that performs arithmetic and bitwise operations on integer binary numbers, this project must implement the previous two projects, that are the 16x16 multiplier and the 16-bit divider.

The operations that the ALU must support are:

- Addition
- Subtraction
- Multiplication
- Division
- AND
- OR
- NAND
- NOR
- XOR
- Shift left
- Shift right
- Circular shift left
- Circular shift right
- Greater comparison
- Smaller comparison
- Equal comparison
- A carry out flag
- A zero flag

8.1 Block diagram

High-level block diagram of the system requirements and description of each system component

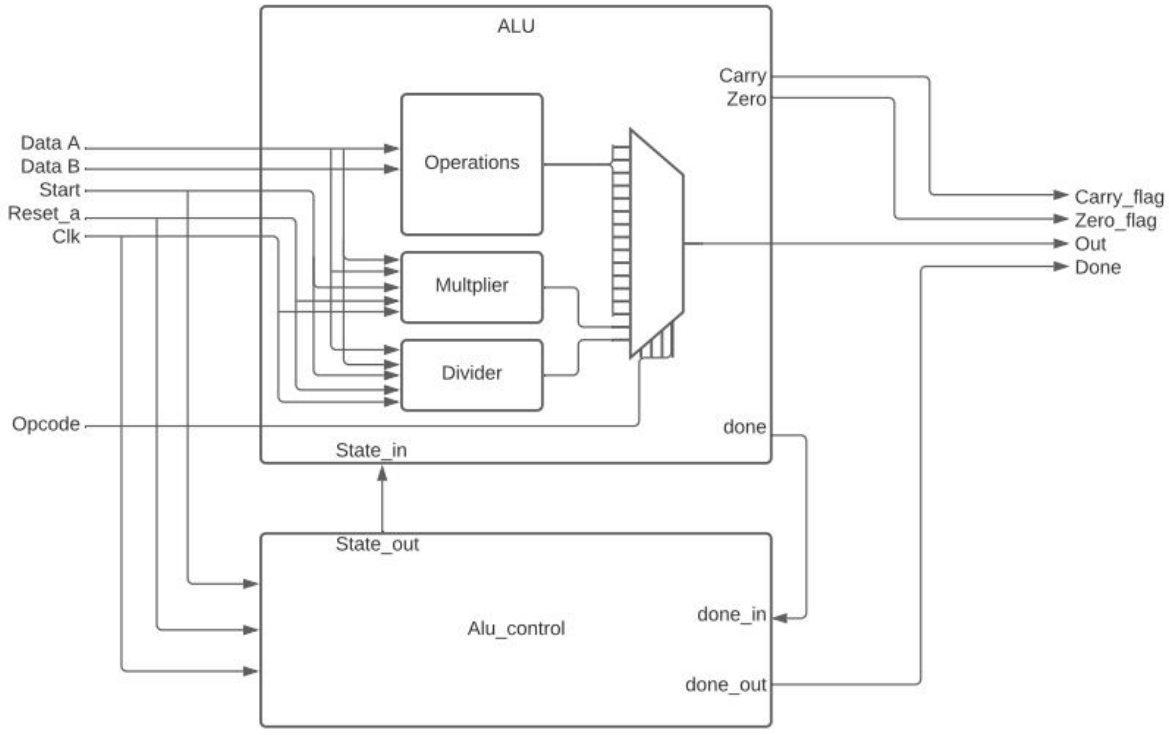

Figure 19 - ALU High level diagram

8.2 Overall functionality

This clocked ALU works like a big multiplexor of operations, that depending on the opcode that we receive, we make the corresponding operation and send out the result; we use the alu_control, mainly to make all control for output that we want to send with the multiplier and divider, because they need more that one clock cycle to get the result and the other purpose is to have a reset, and a start signal, that tell us when did we should make the operation and when no.

9.0 ALU Design Description

In this ALU project we have 2 main blocks that we use to process the inputs, that are mainly the both data inputs and the opcode input, and get the required outputs, that are the result and the three flags needed. The alu unit has inside the two previous projects that are the multiplier and the divider and that is the main reason that in this alu block we need to have the clk, reset_a and start inputs, to handle all the specifications with these inputs in these two previous projects; that is the same reason that we use the aluu control, to give the necessary clock cycle to get the result to these multiplier and divider blocks, because all the other operations are done with the basic Verilog HDL operators.

9.1 Intricacies of the of the hardware architecture

The main intricacies was when we needed to implement the 2 previous projects, because that's the only reason that we made a second block in this project to handle this. Another problem that we found, that was much easier to handle, was the way we have to connect each one of the inputs and output of these and the other two projects, and to take care with the names of the I/O and of each Verilog code that we need to call.

9.2 Hardware architecture at signal level

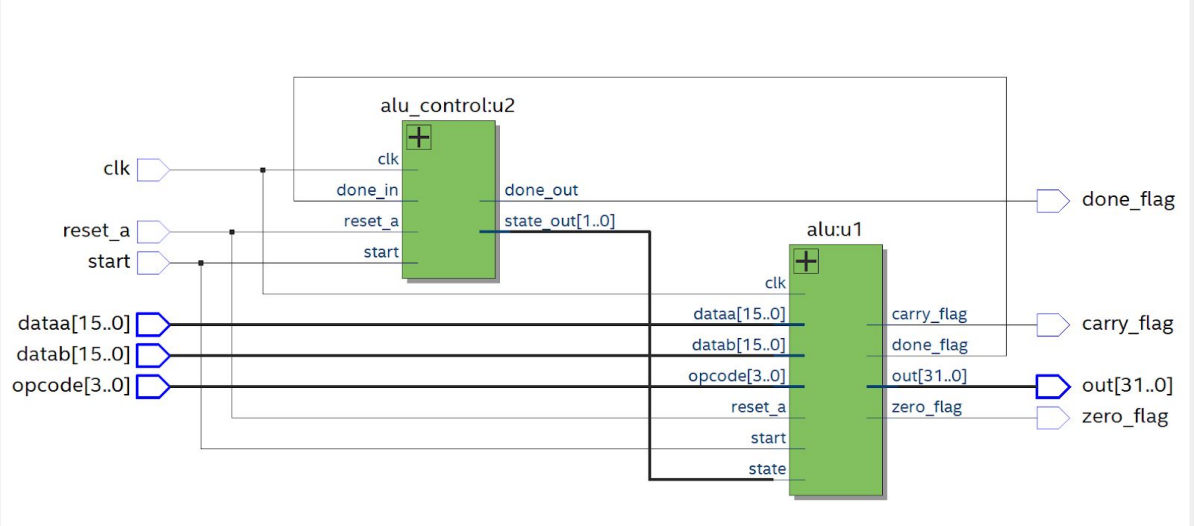

Figure 20 - ALU Hardware architecture at signal level

9.3 FSM controllers

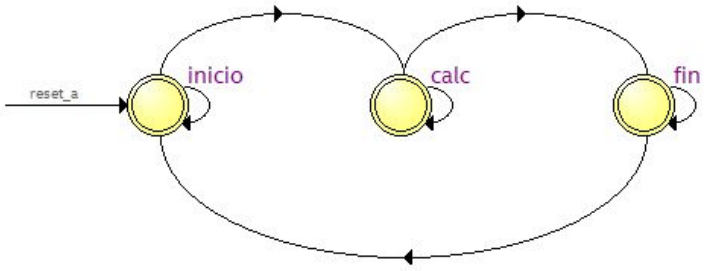

Figure 21 - ALU FSM

10.0 ALU Test and Test Results

10.1 Test Strategy

This project is tested with the a code that generates the clk signal and that will be giving the necessary values to the start and reset_a signals, in this same testbench we assign a specific value to each data input and finally we give values to the opcode signal, values from 0 to15 using a forever command that change the opcode value each time our alu send out the done flag, this will lend us test all the operations with this two data inputs.

10.2 Test Results

To test this project we used a test bench called: alu_top_tb.v. This test bench initialized all the required initial signals such as start, restar and the clock. It also declared 2 numbers to do some testing and it increments the op code to do all the required operations. The outputs to keep in track are: out (the output of the operation), zero flag (if the result gives zero) and done_flag (which shows when an operation is done).

10.2.1 Synthesis results

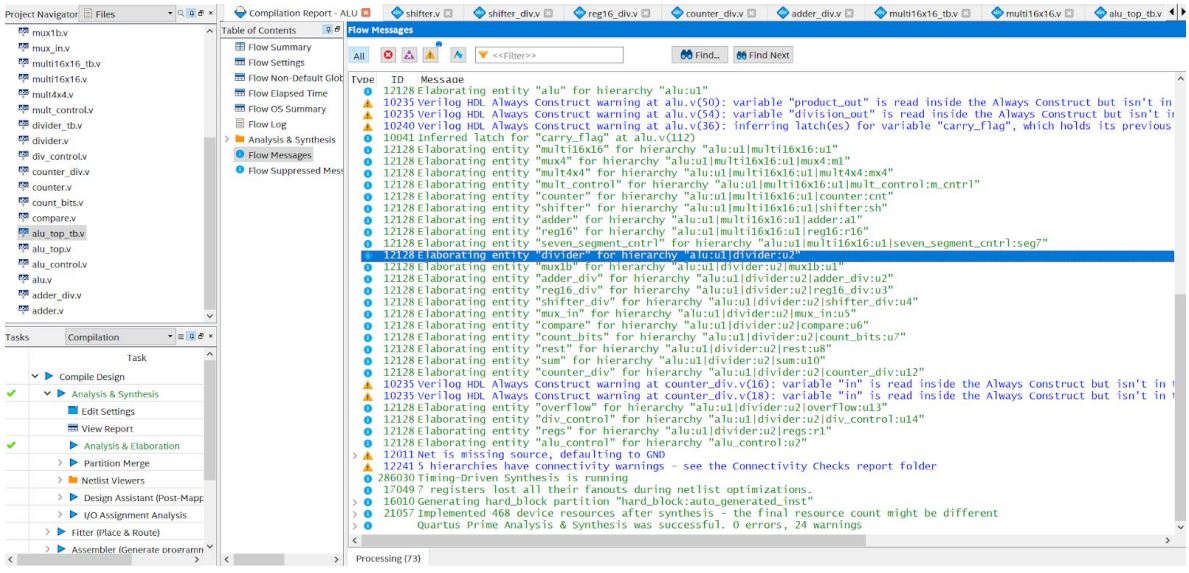

Figure 22 - ALU top code Synthesis result

10.2.2 Simulation results

Opcode:

ADD	4’b0000
SUB	4’b0001
MULT	4’b0010
DIV	4’b0011
AND	4’b0100
OR	4’b0101
NAND	4’b0110
NOR	4’b0111
XOR	4’b1000
SL	4’b1001
SR	4’b1010
CSL	4’b1011
CSR	4’b1100
GREATER	4’b1101
SMALLER	4’b1110
EQUAL	4’b1111

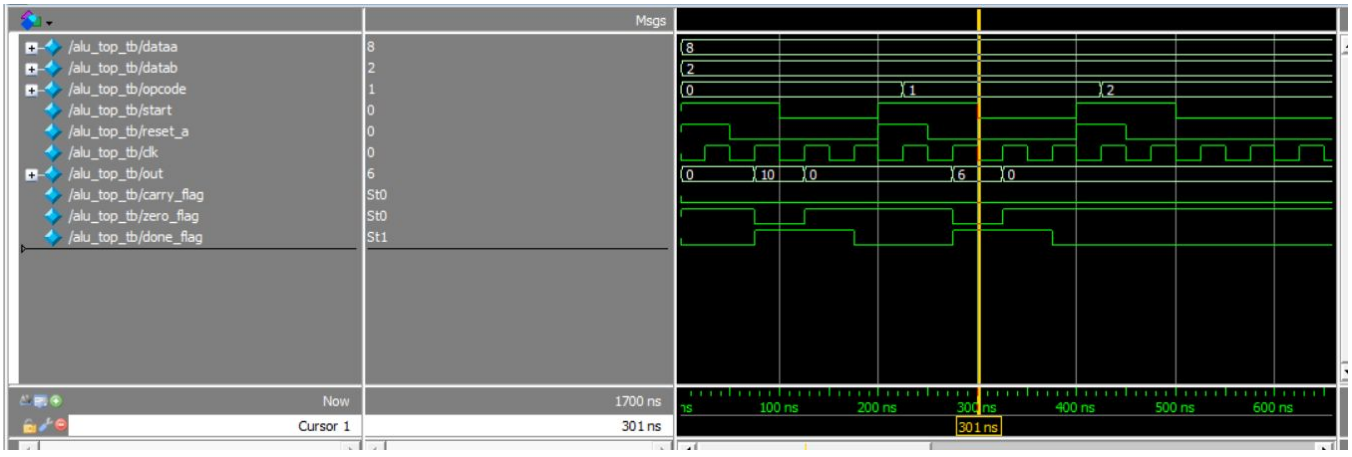

Figure 23 - ALU Simulation result 1

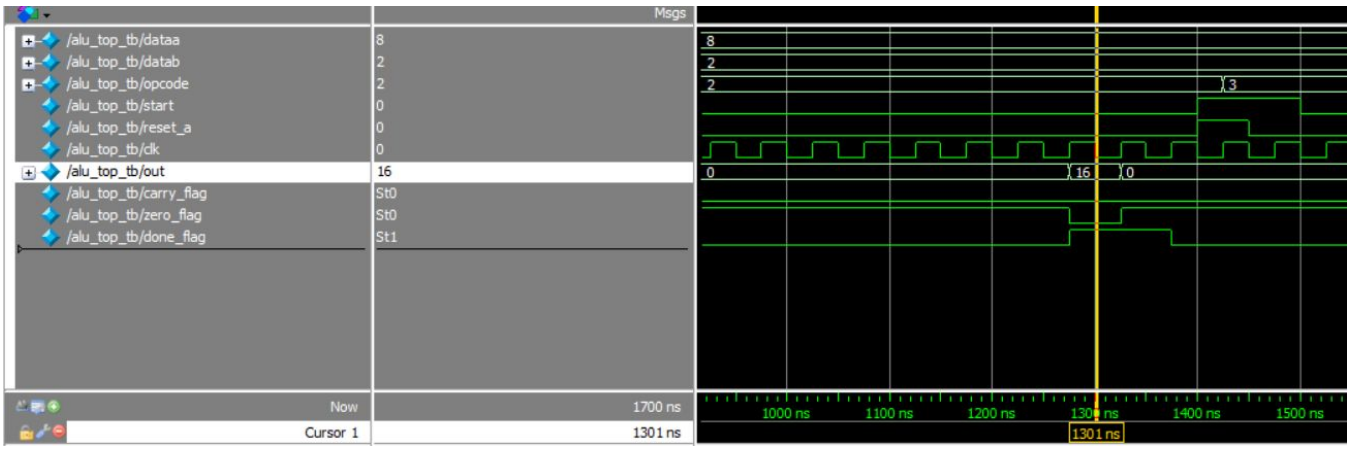

Figure 24 - ALU Simulation result 2

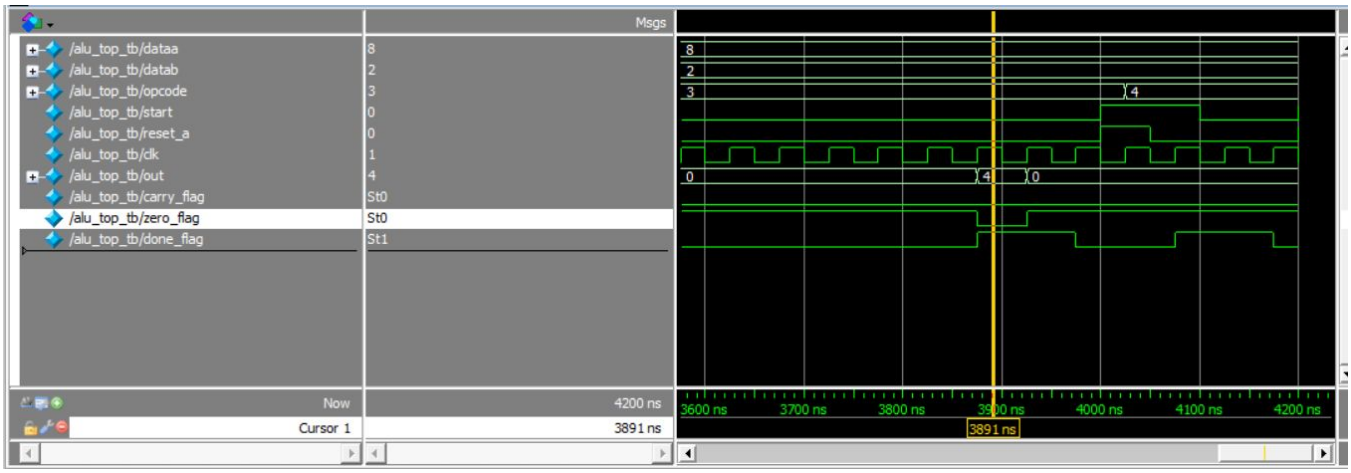

Figure 25 - ALU Simulation result 3

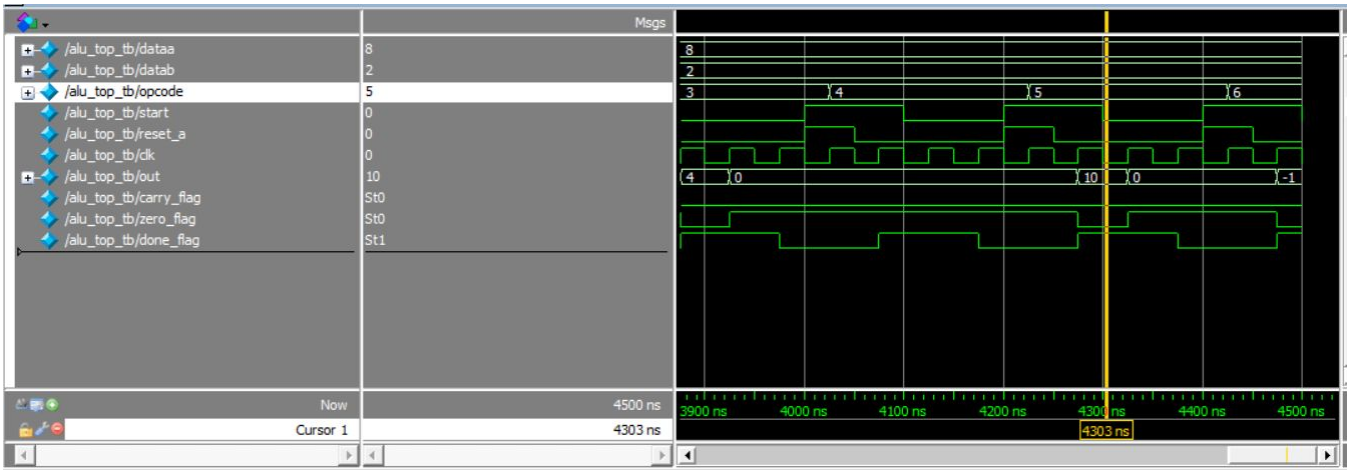

Figure 26 - ALU Simulation result 4

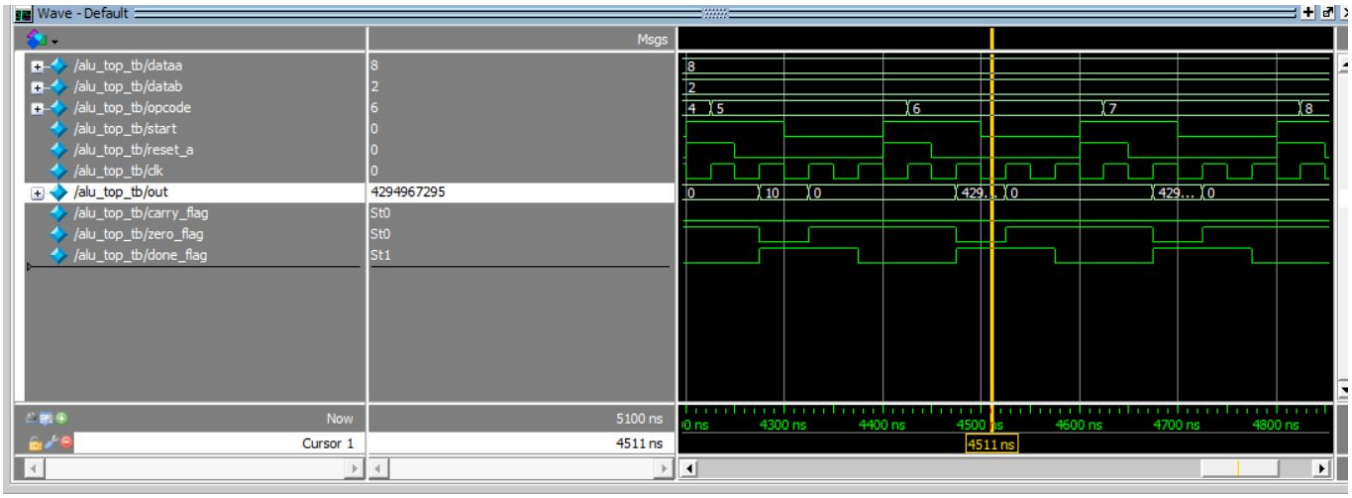

Figure 27 - ALU Simulation result 5

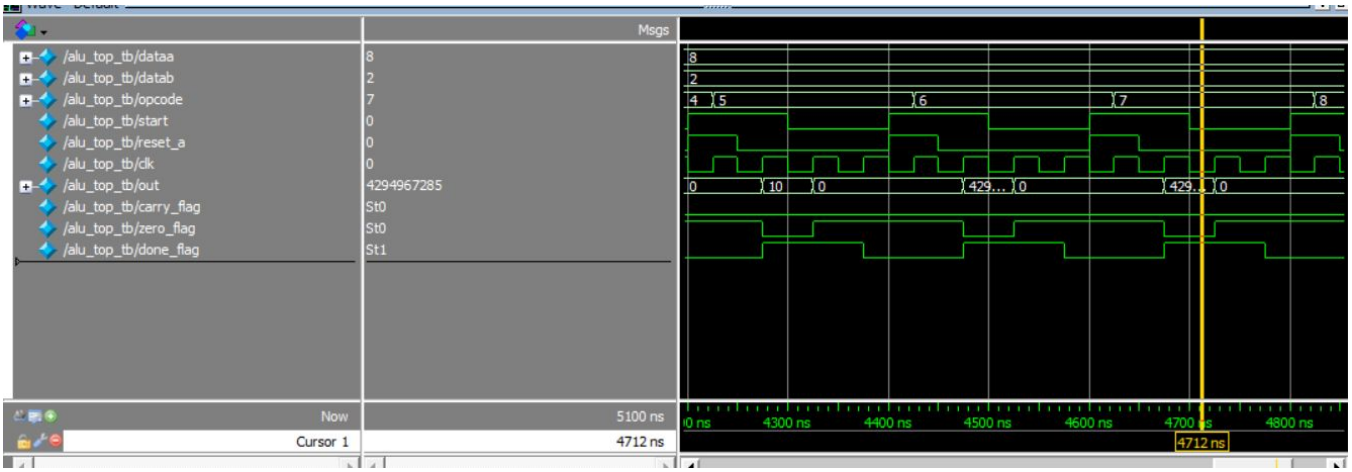

Figure 28 - ALU Simulation result 6

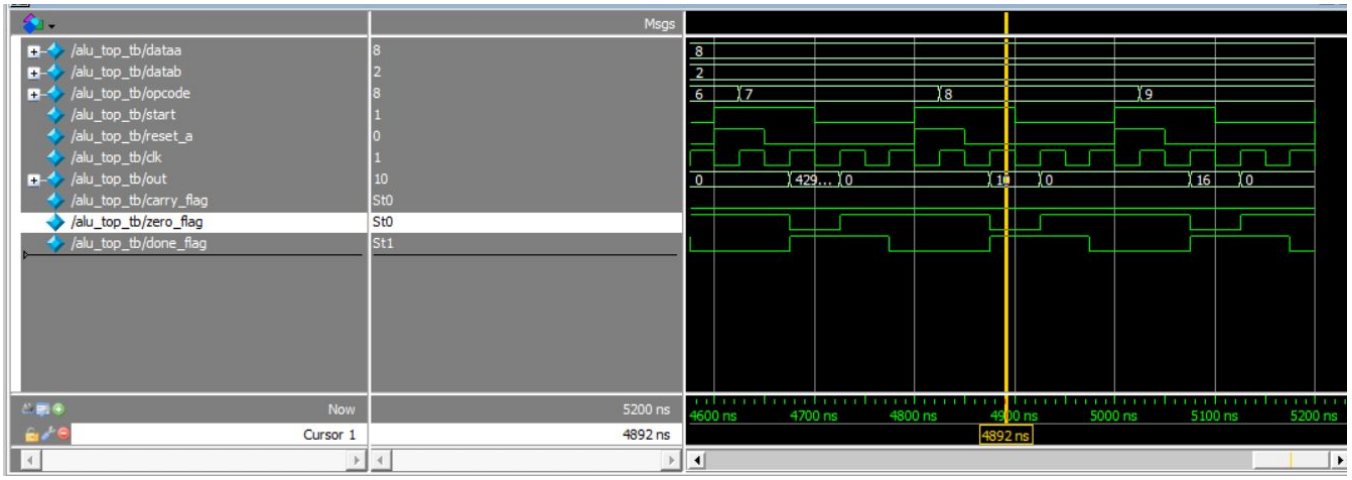

Figure 29 - ALU Simulation result 7

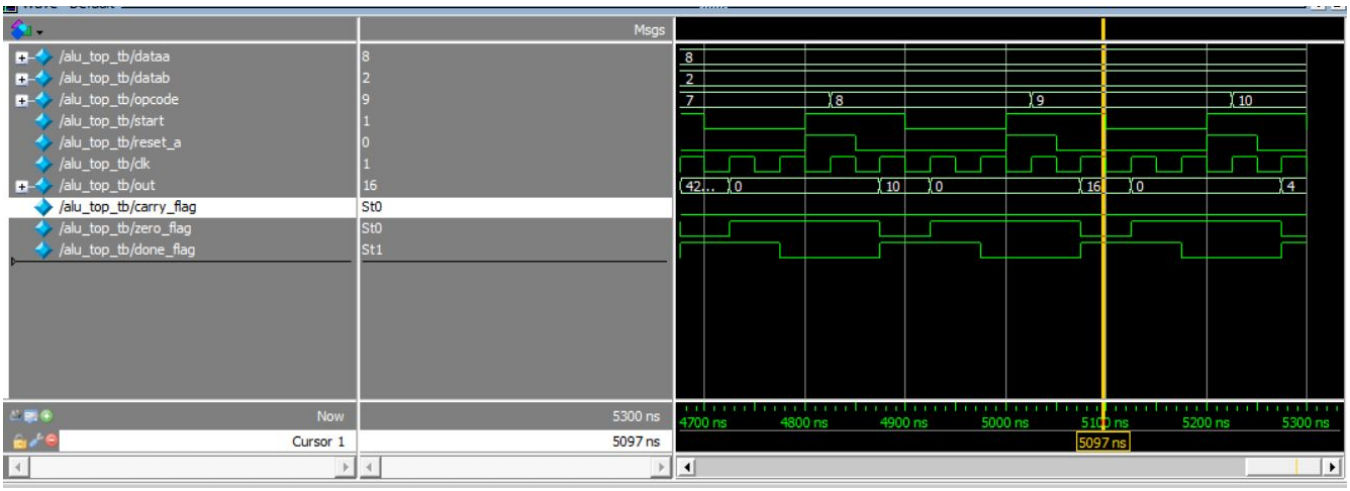

Figure 30 - ALU Simulation result 8

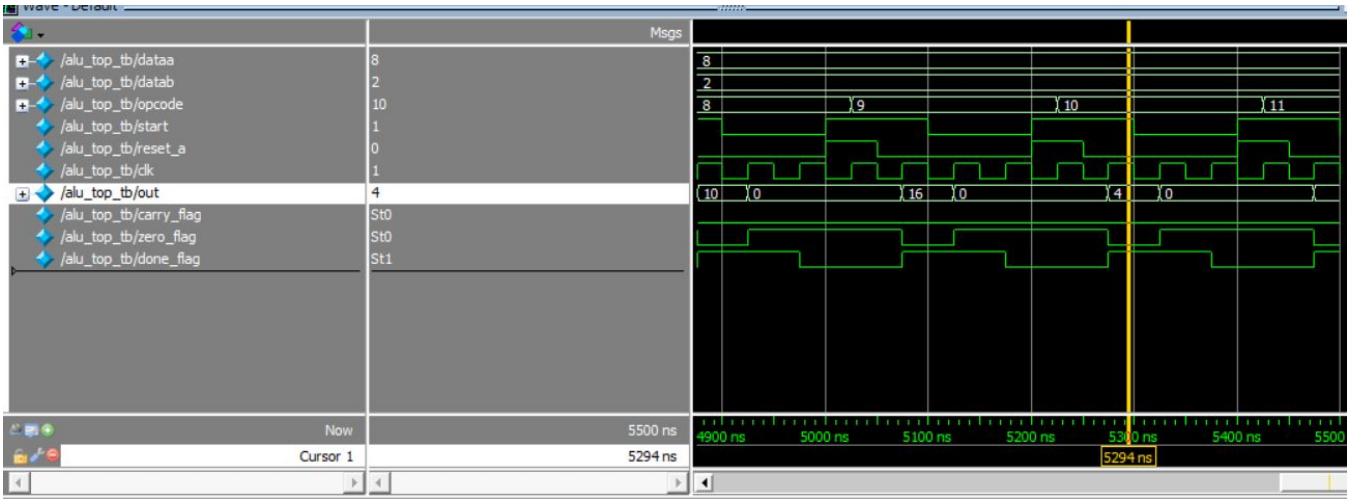

Figure 31 - ALU Simulation result 9

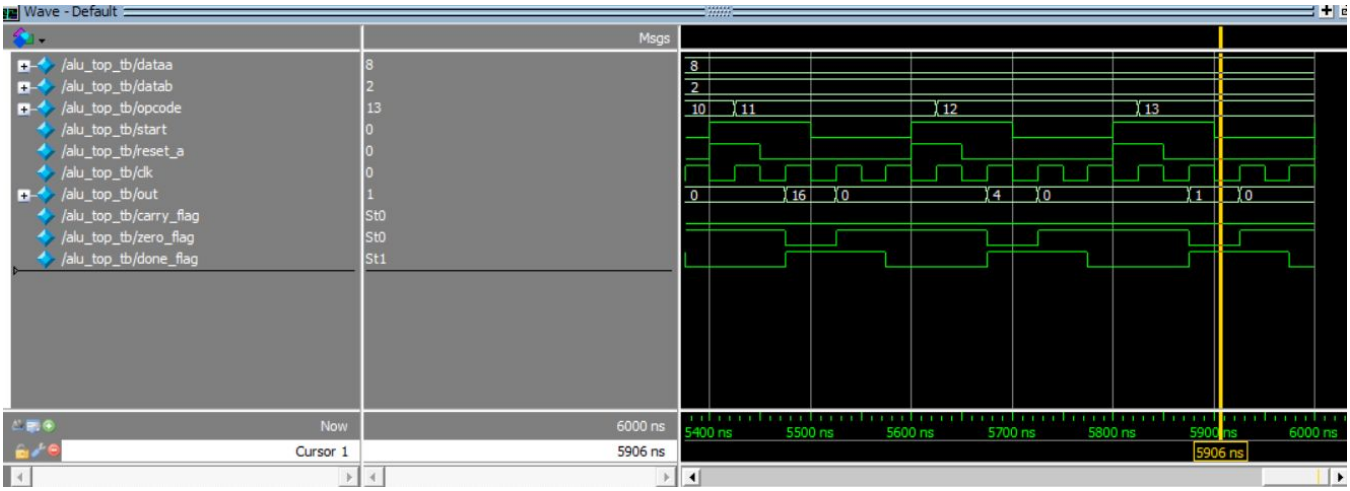

Figure 32 - ALU Simulation result 10

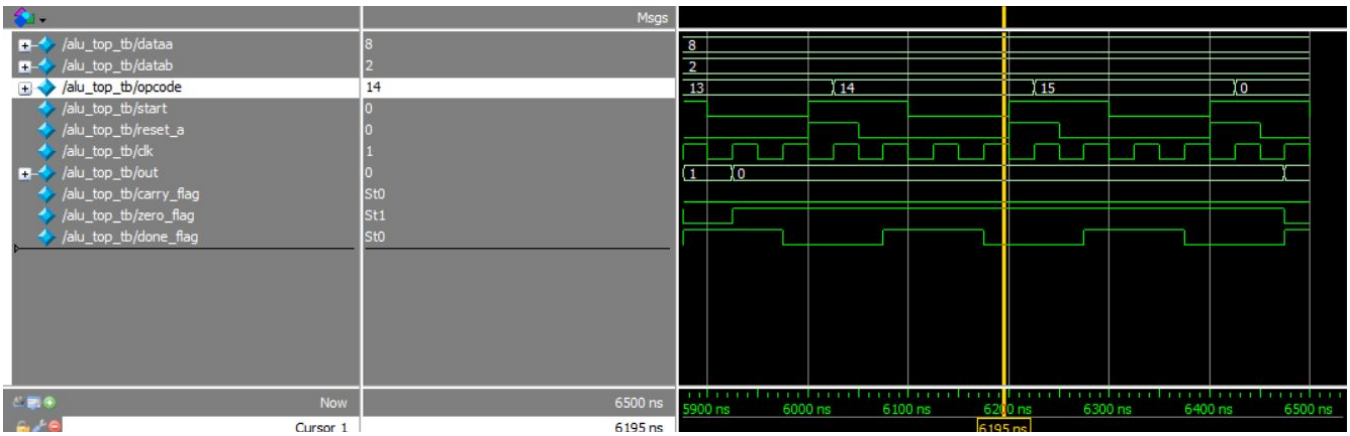

Figure 33 - ALU Simulation result 11