Lab : ALU

Chue Zhang

Csc343 Fall 2021

Professor Gertner

Signature :

A picture containing text, whiteboard

Description automatically generated

Table of Contents

[Objective 4](#_Toc87582594)

[Specification 5](#_Toc87582595)

[Or 6](#_Toc87582596)

[And 7](#_Toc87582597)

[Nor 8](#_Toc87582598)

[Shift Left(SL) 9](#_Toc87582599)

[Shift Right(SR) 10](#_Toc87582600)

[Shift Right Arithmetic(SRA) 11](#_Toc87582601)

[Sign Extend 12](#_Toc87582602)

[Control Unit 14](#_Toc87582603)

[ALU 15](#_Toc87582604)

[Main 17](#_Toc87582605)

[Waveforms and Explanations 18](#_Toc87582606)

[3-Port Ram, Memory Access 19](#_Toc87582607)

[Add 20](#_Toc87582608)

[Addu 21](#_Toc87582609)

[Addi 22](#_Toc87582610)

[Addiu 23](#_Toc87582611)

[Sub 24](#_Toc87582612)

[Subu 25](#_Toc87582613)

[And 26](#_Toc87582614)

[Andi 27](#_Toc87582615)

[Or 28](#_Toc87582616)

[Ori 29](#_Toc87582617)

[Nor 30](#_Toc87582618)

[Shift Left 31](#_Toc87582619)

[Shift Right 32](#_Toc87582620)

[Shift Right Arithmetic 33](#_Toc87582621)

[Concluding Statement 34](#_Toc87582622)

# Objective

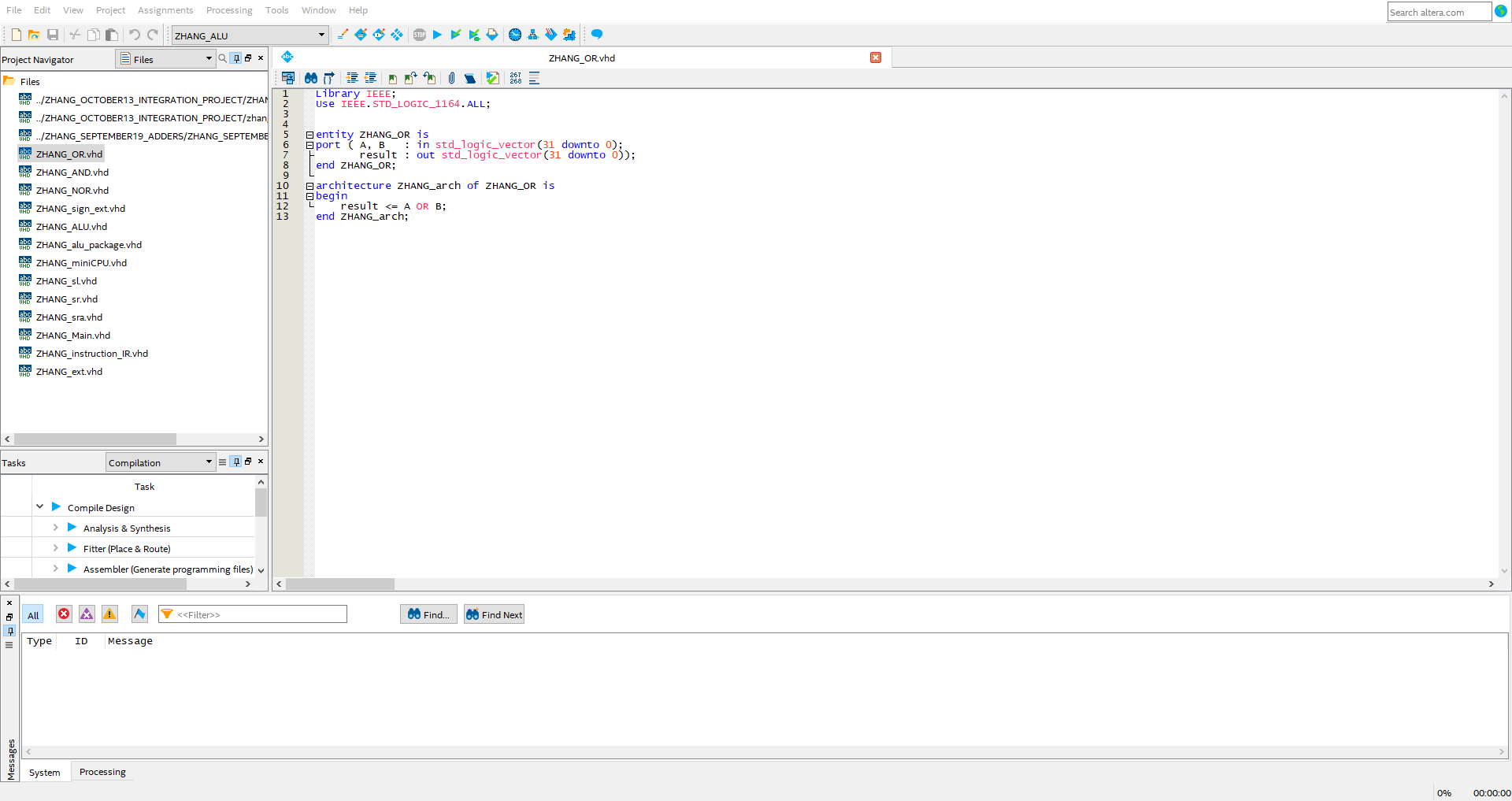
Objective of this laboratory assignment is to create an ALU that follows the MIPS green pages. A side objective of this assignment is to learn more about how an ALU performs including its components such as RS, RT, RD, Shamt, Funct and the immediate 16. All of which are extremely important for the development of the ALU.

# Specification

In this assignment, we are tasked with inputting an 32-bit instruction code which we will then decode and perform arithmetic or logical operations on operands based on the instructions provided. RS, RT and RD are 5-bit addresses that point towards a memory register in which what is returned is a 32-bit value which we will perform operations on. Operations are also 6 bits in length and can perform a variety of operations such as “And”, “Andi”, “Shift Left” and so on. The 6-bit operation goes through a control unit in which it processes the op code and returns a 4-bit ALU control code and several other important information. Lastly, when the operations have been conducted, the value is then stored into RD and we can check what is written in RD simply by loading the address content.

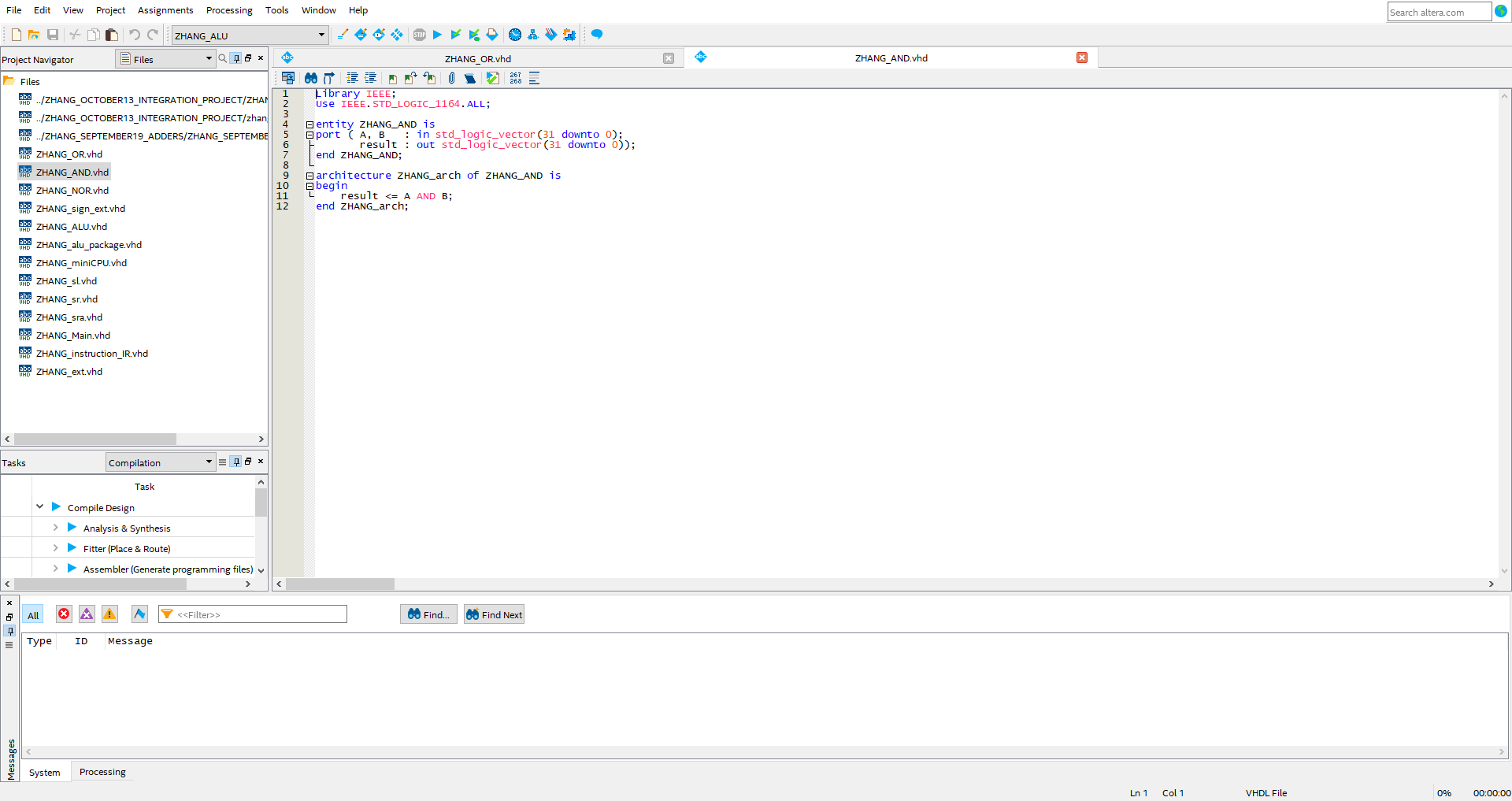
Below is Code that were key into the development of the ALU [Only new code will be shown in this laboratory assignment, any old code such as the 3-port RAM, IR Register and the N-bit adder sub will not be shown as a screenshot. Please refer to the October 17 3-port Adder/Sub unit laboratory assignment for information on those components]. The design of the ALU is that an instruction code is inputted by the user and the instruction code is then fed into the IR register and decoded into Opcode, RS, RT, RD, Shamt, Funct and imm16. The Opcode afterwards is fed into the control unit which returns “IR\_TYPE” or a Boolean output of 1 or 0 determining if the operation is I type or R type, “ALUctrl” or the ALU control which is the 4-bit operation code that we use to determine which operation we perform and “ExtOp” which helps us determine if the I-type instruction requires zero sign extension of signed extension. After obtaining this information, the RS, RT and RD are fed into the 3-port ram to retrieve 32-bit values. RS is basically input1, RT is input 2 and RD is the address in which we will be writing into the RAM. Once we obtain the 32-bit values of the RS and RT register, we feed it into the ALU for computation and whatever is outputted is stored into the address value that is RD. We can test to see if RD is stored by simply setting RS address to RD address and perform operations on RS and RT with new value.

## Or



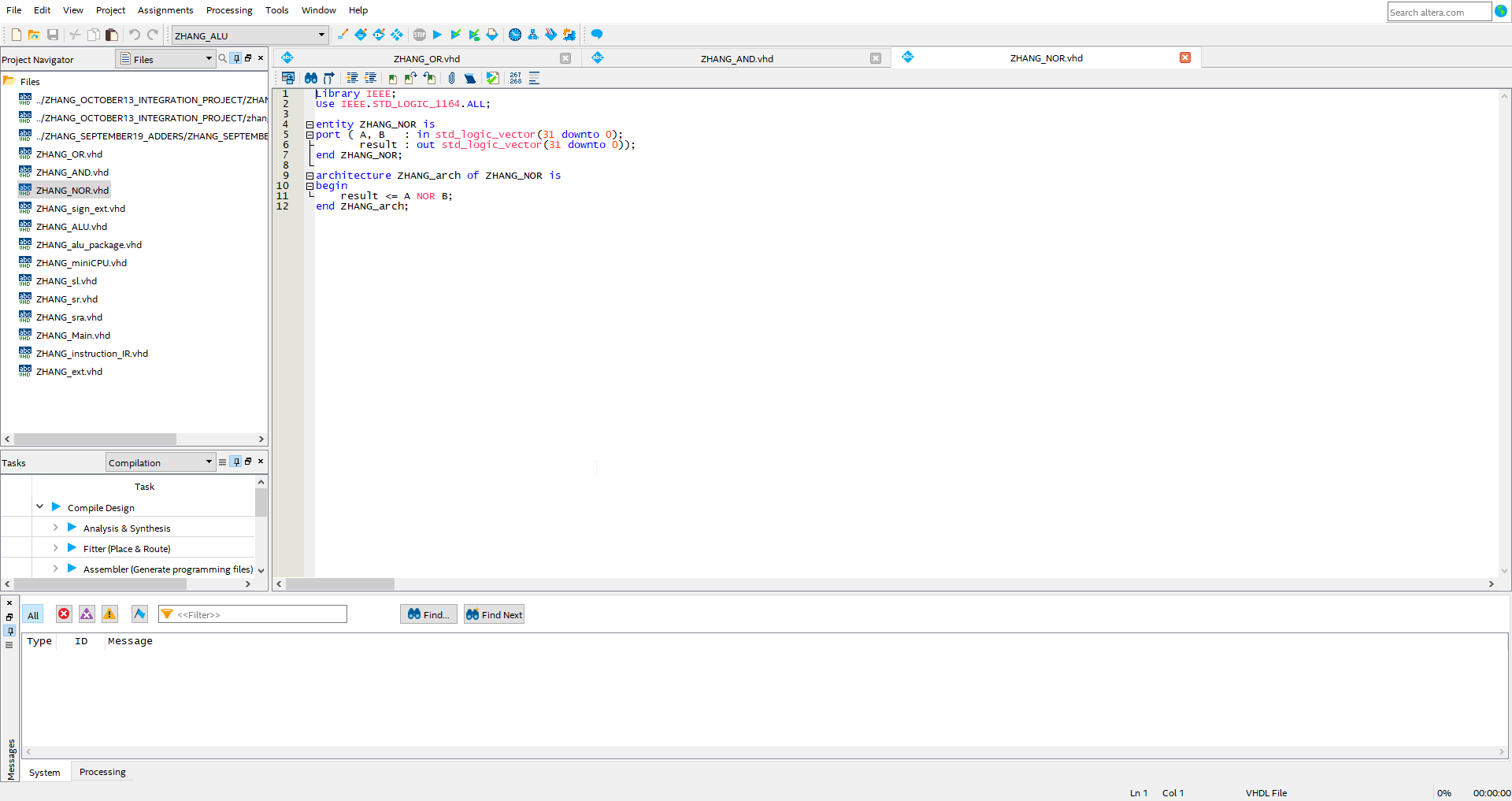
Or VHDL code which will be used for both “OR” and “ORI “ operations

## And



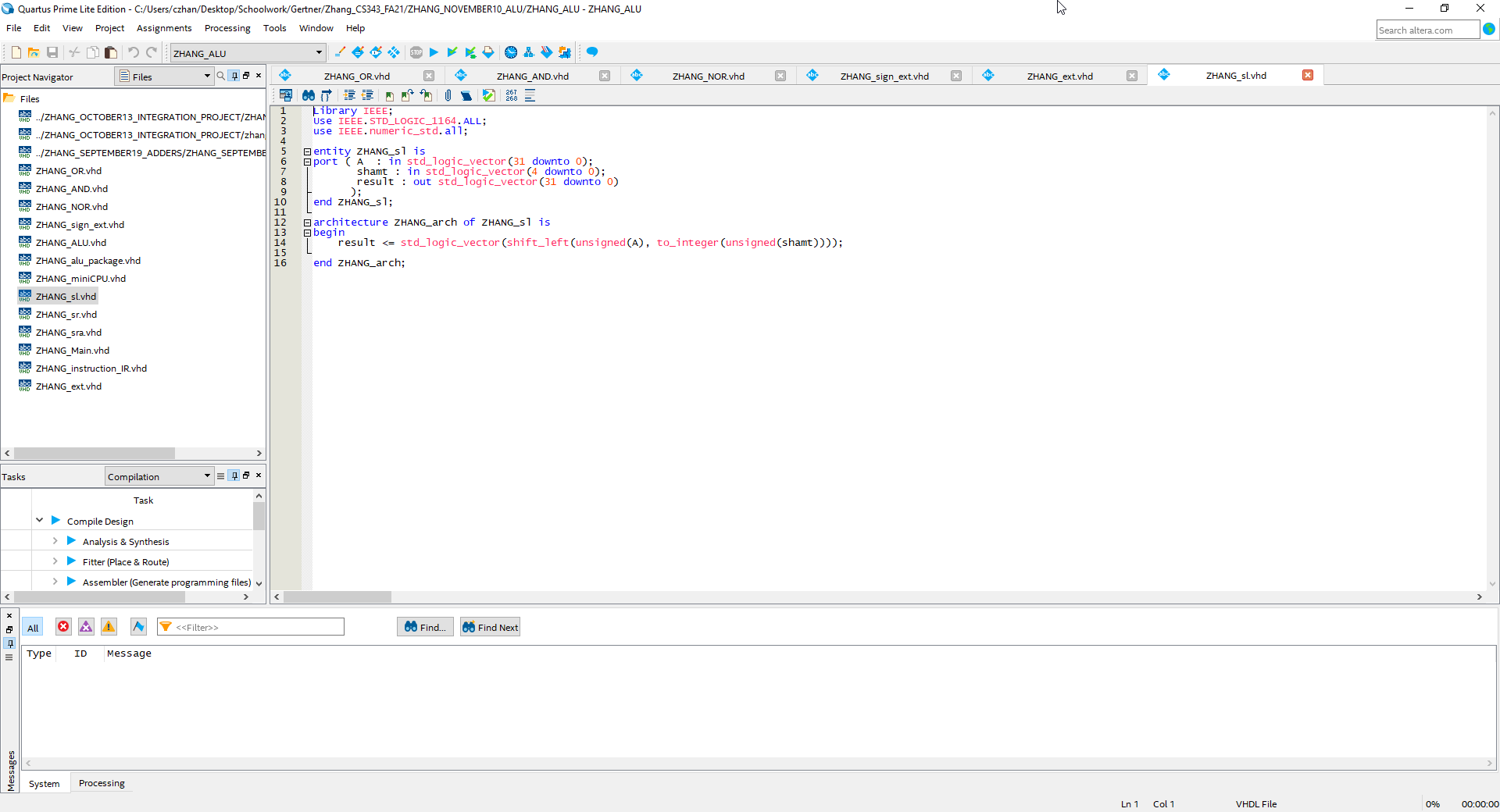
And VHDL code that will be used for both “AND and “ANDI” operations

## Nor



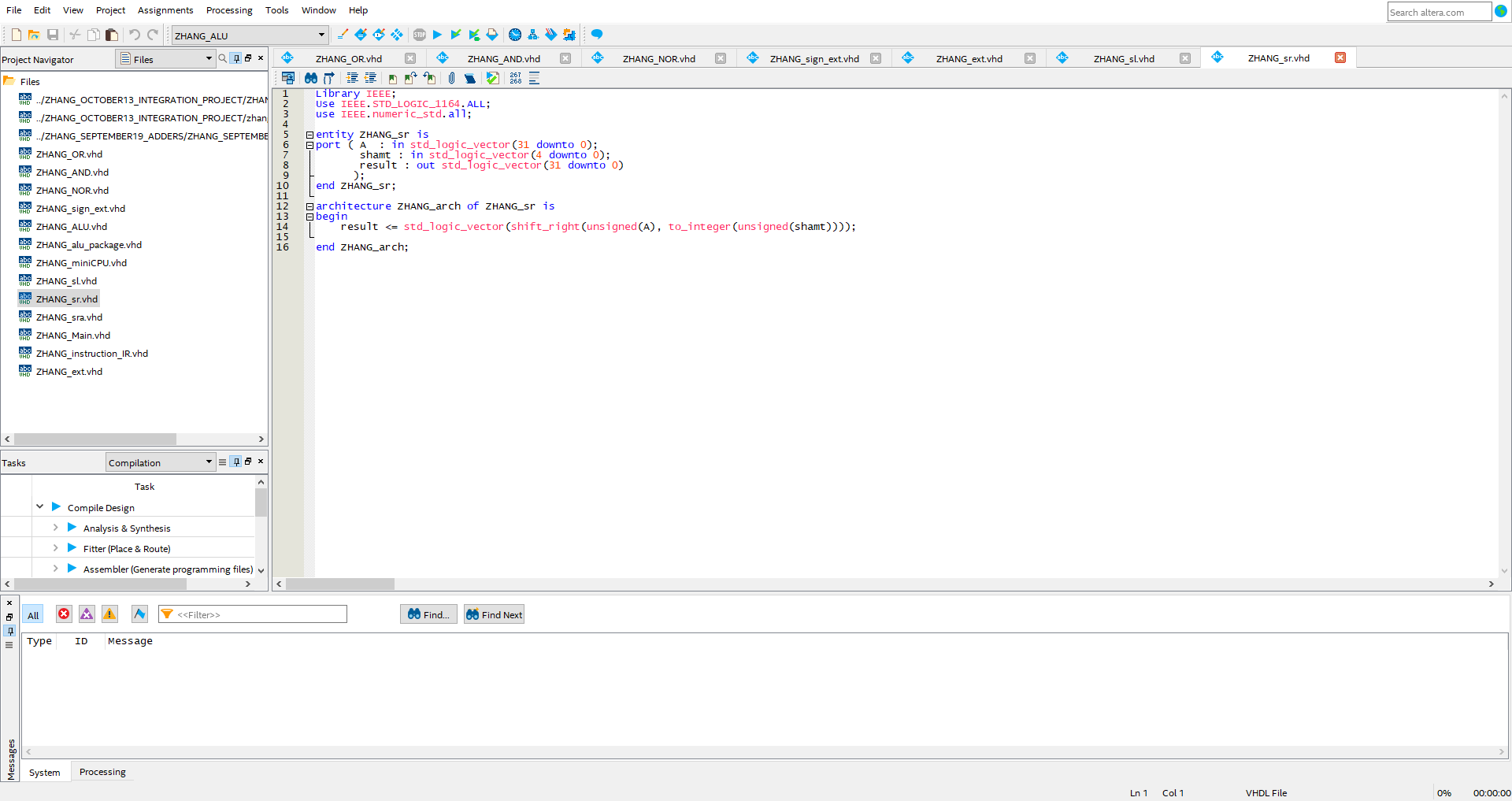
Nor VHDL code

## Shift Left(SL)



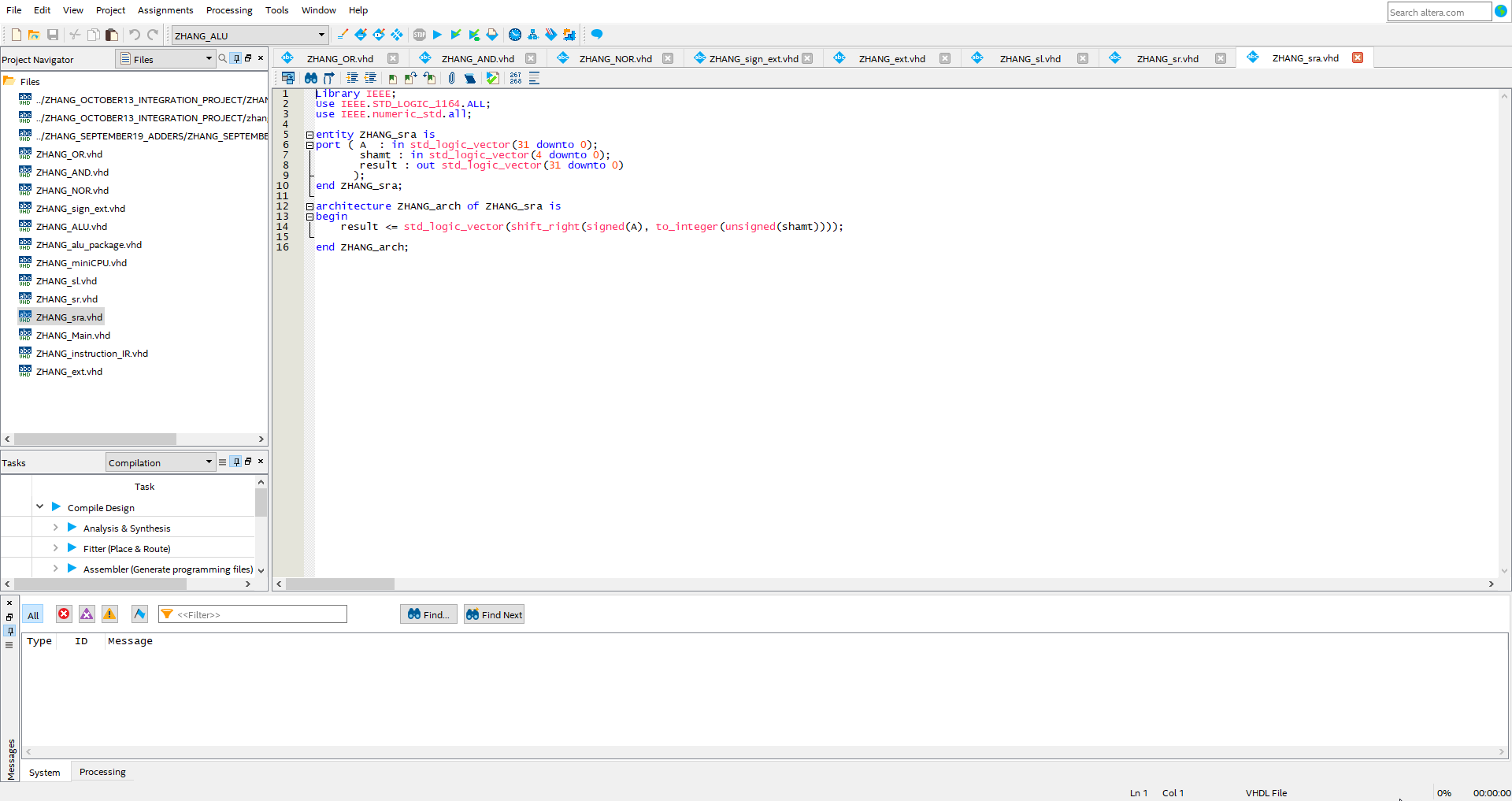
Shift left VHDL code

## Shift Right(SR)



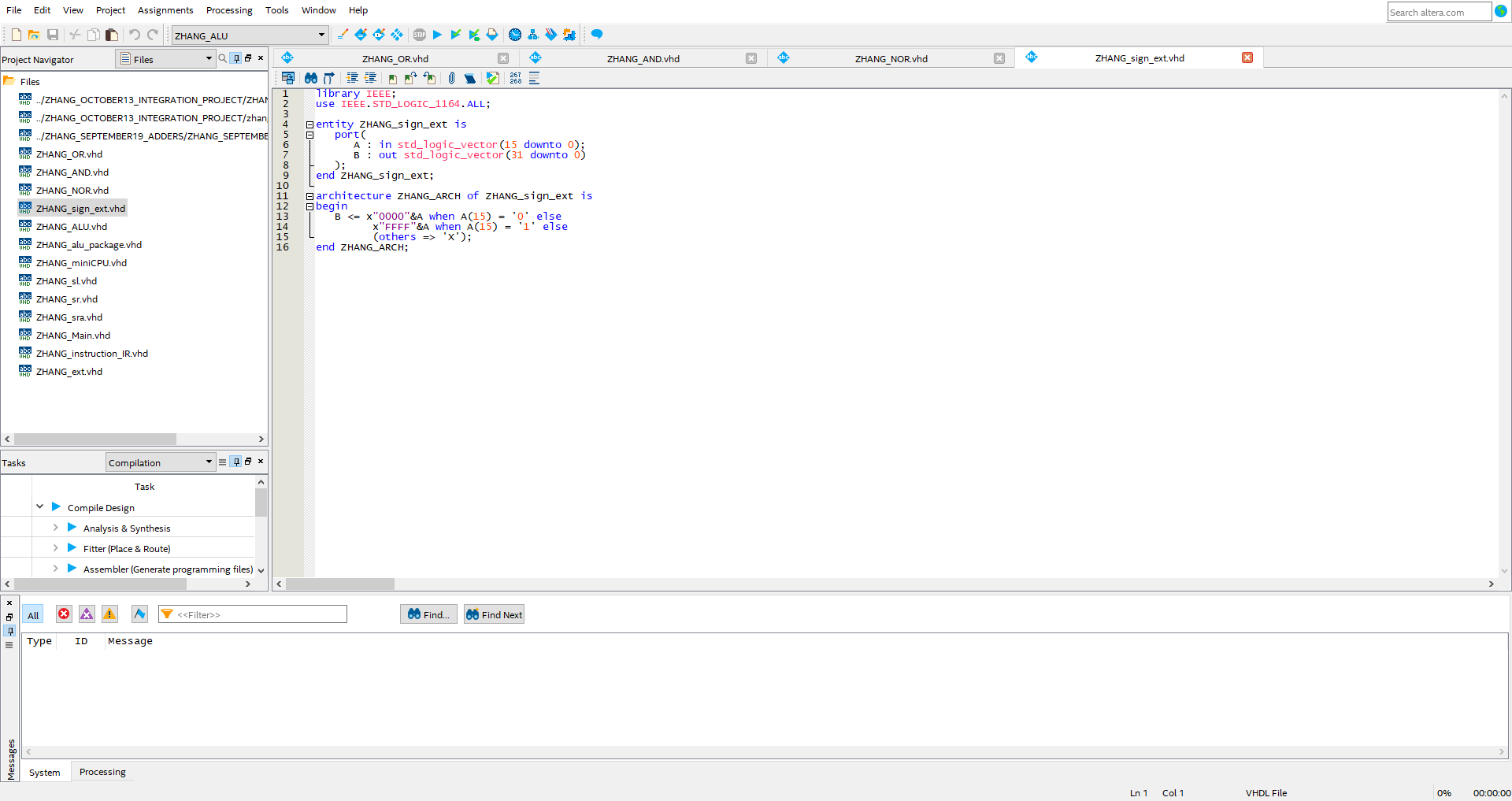
Shift right VHDL code

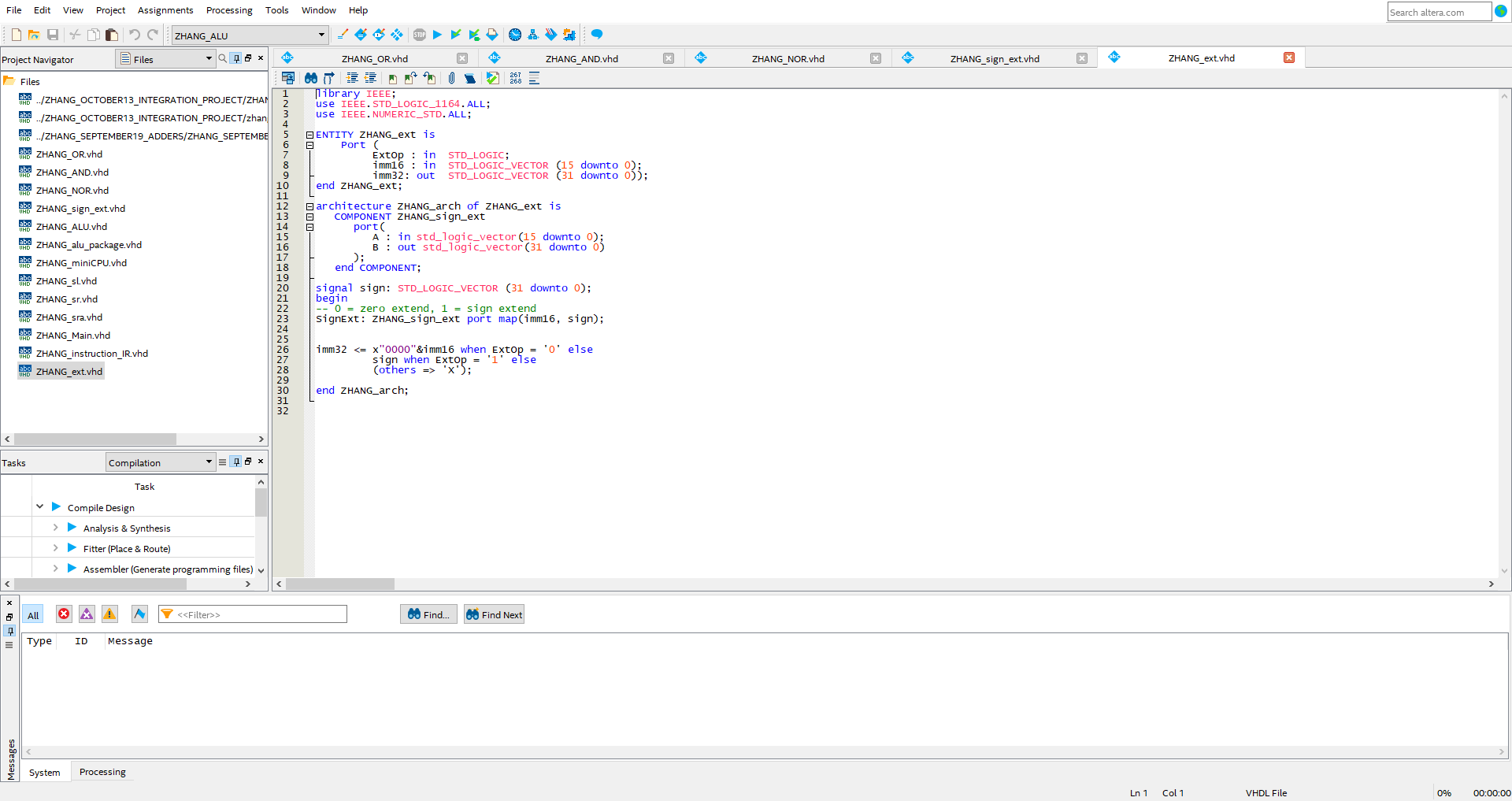
## Shift Right Arithmetic(SRA)



Shift Right Arithmetic VHDL code

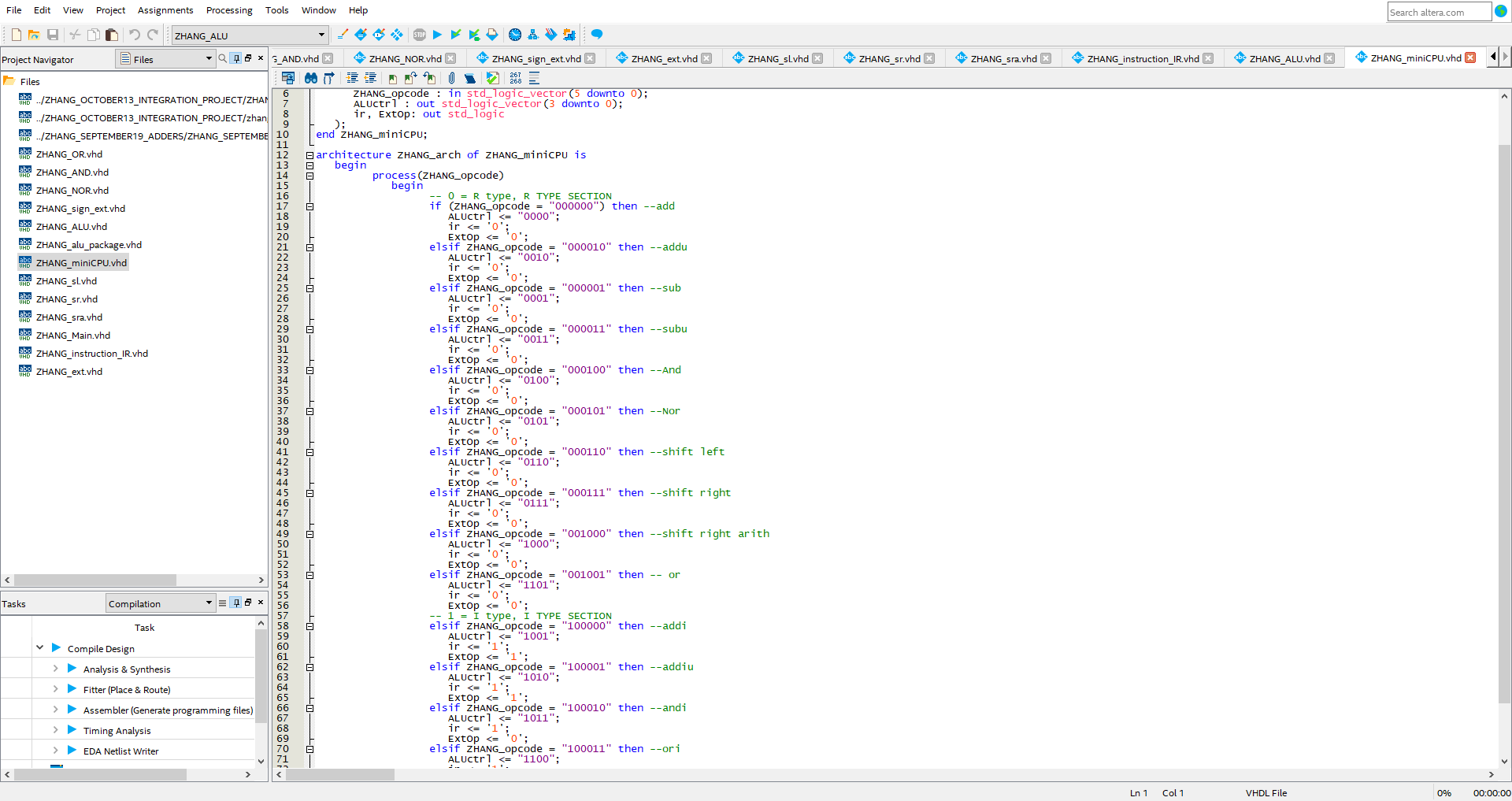
## Sign Extend





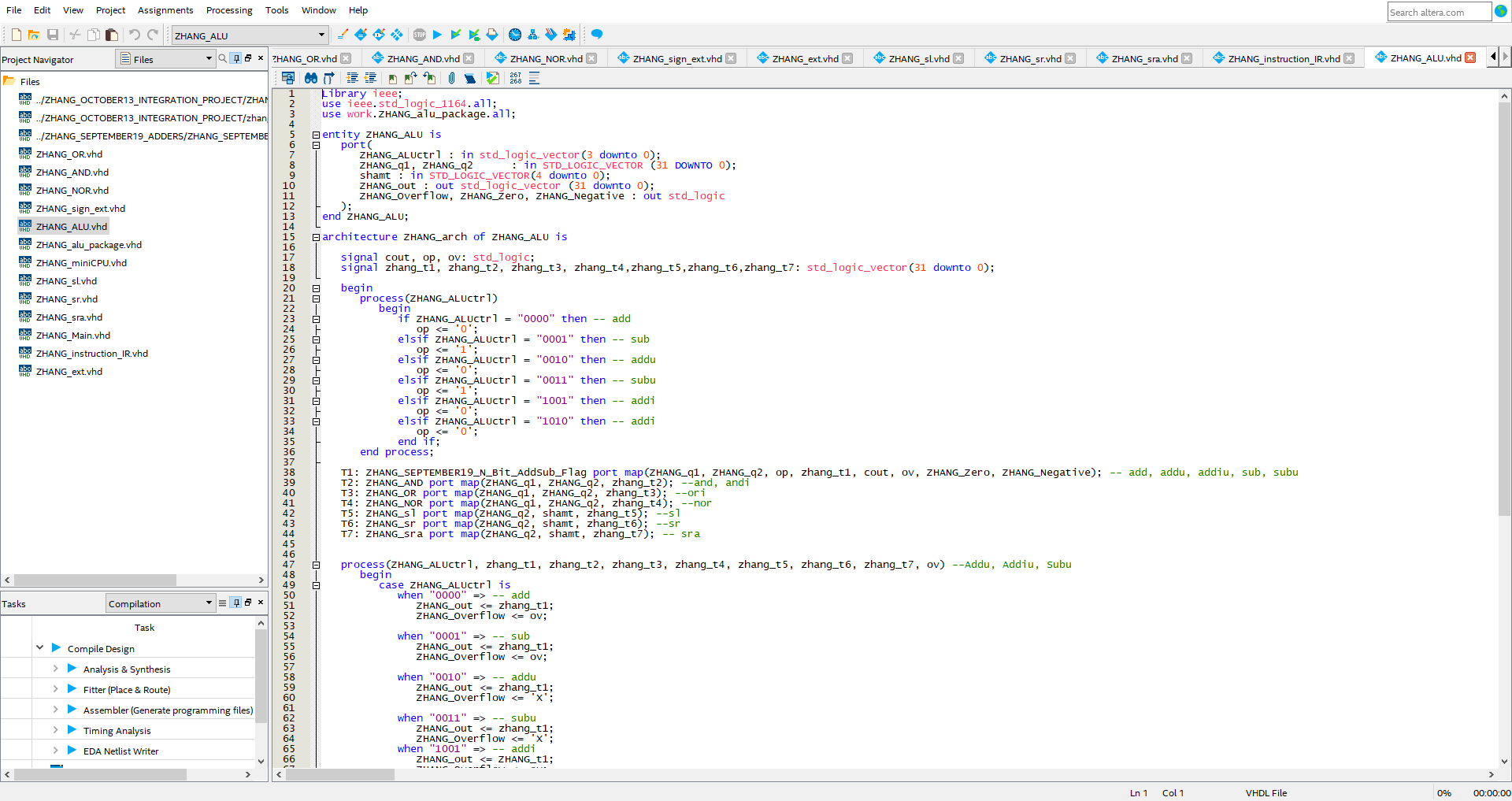
Sign Extend is broken down into two files, one called zhang\_sign\_extension and a main extension file called zhang\_extension. The purpose of this is to determine whether we perform zero extension on the imm16 or signed extension on the imm16.

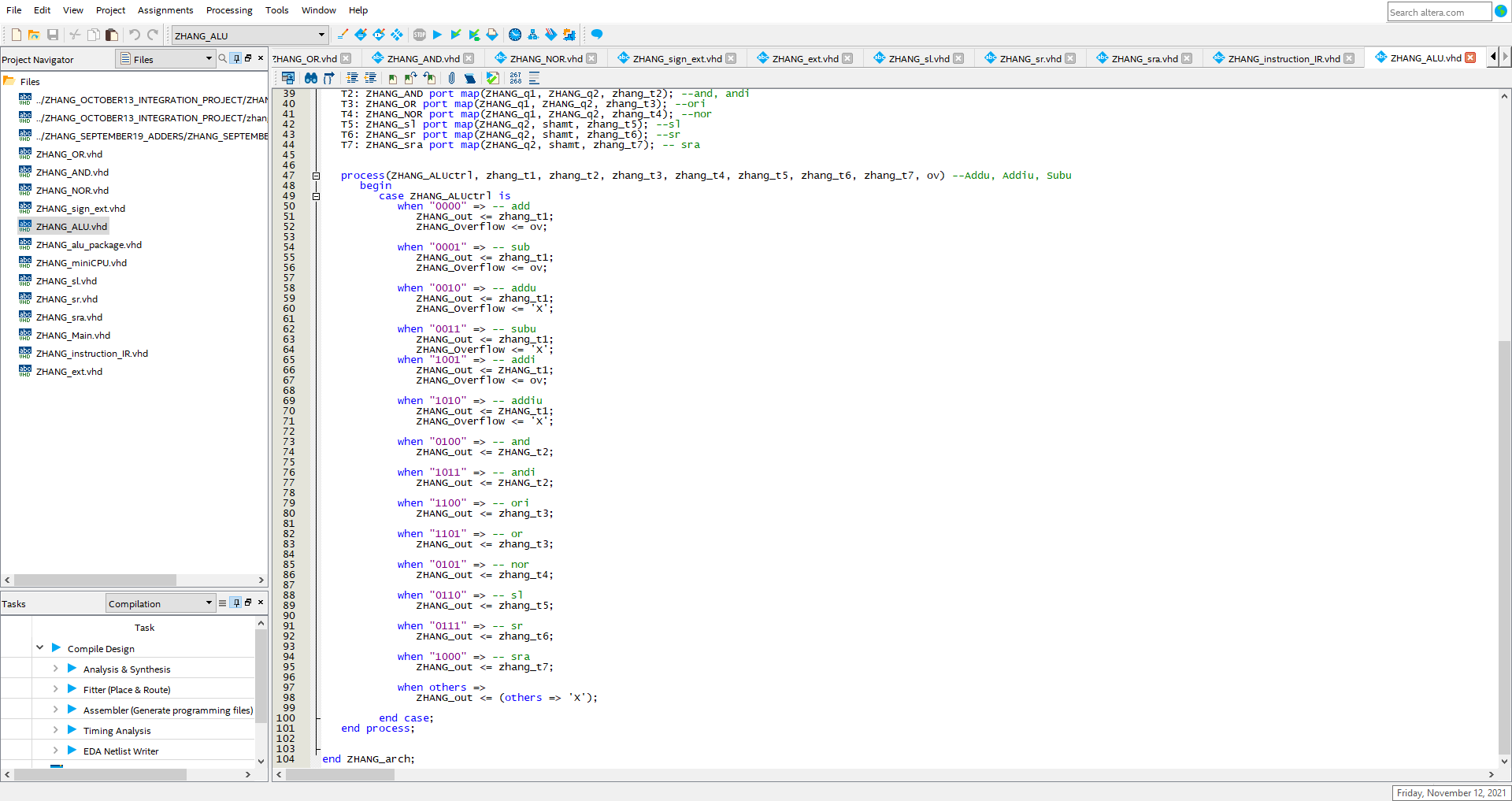
## Control Unit



A new file introduced to help process the data retrieved from the IR decoder. Return values are ALUctrl, IR and ExtOp. Return values are explained above in specification section.

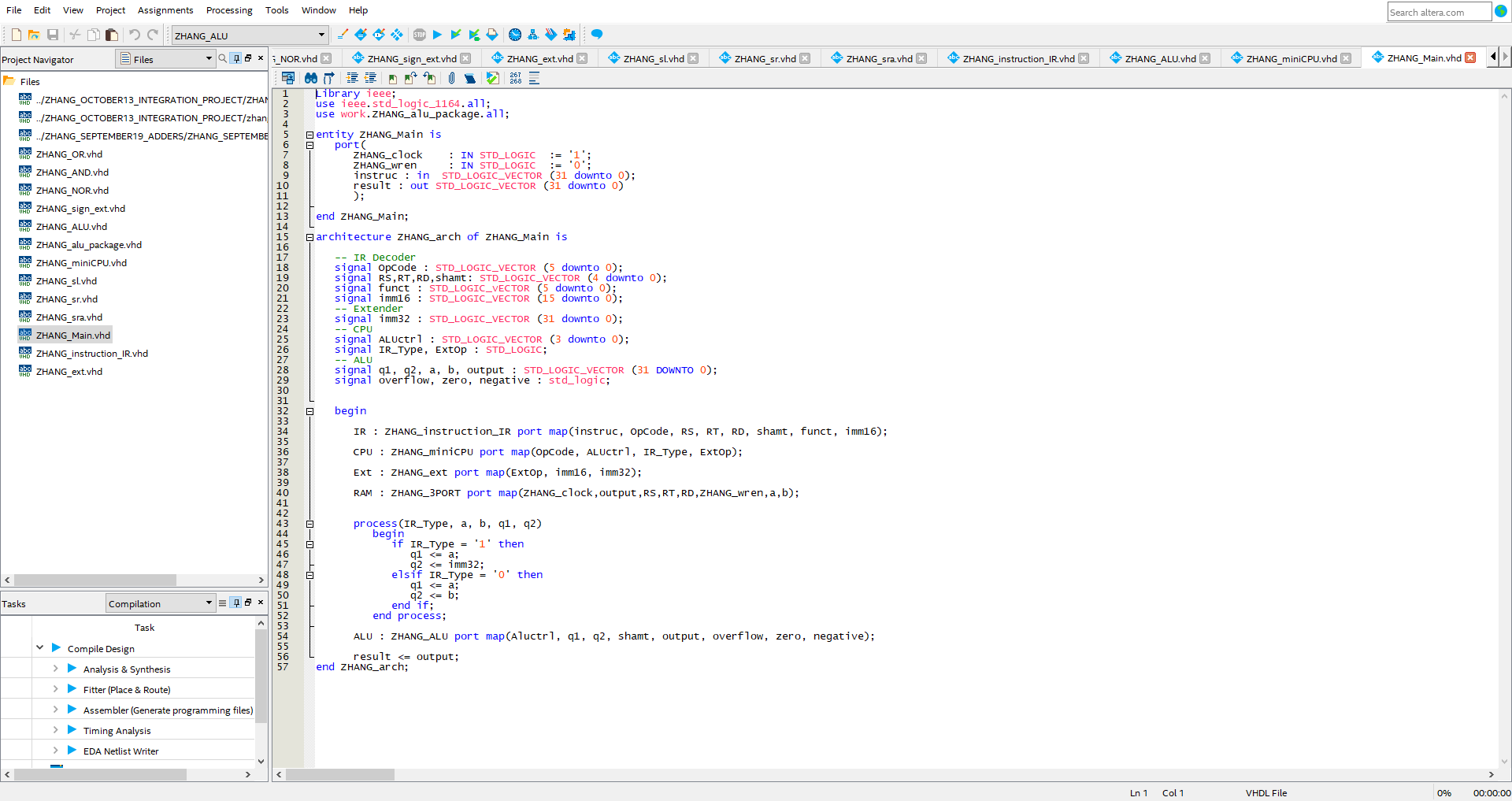
## ALU





The ALU which consists of combining the N-bit adder/sub unit and logic operations such as “AND”, “OR” and such. Case statement is used to help process the ALUctrl to know what should be outputted.

## Main

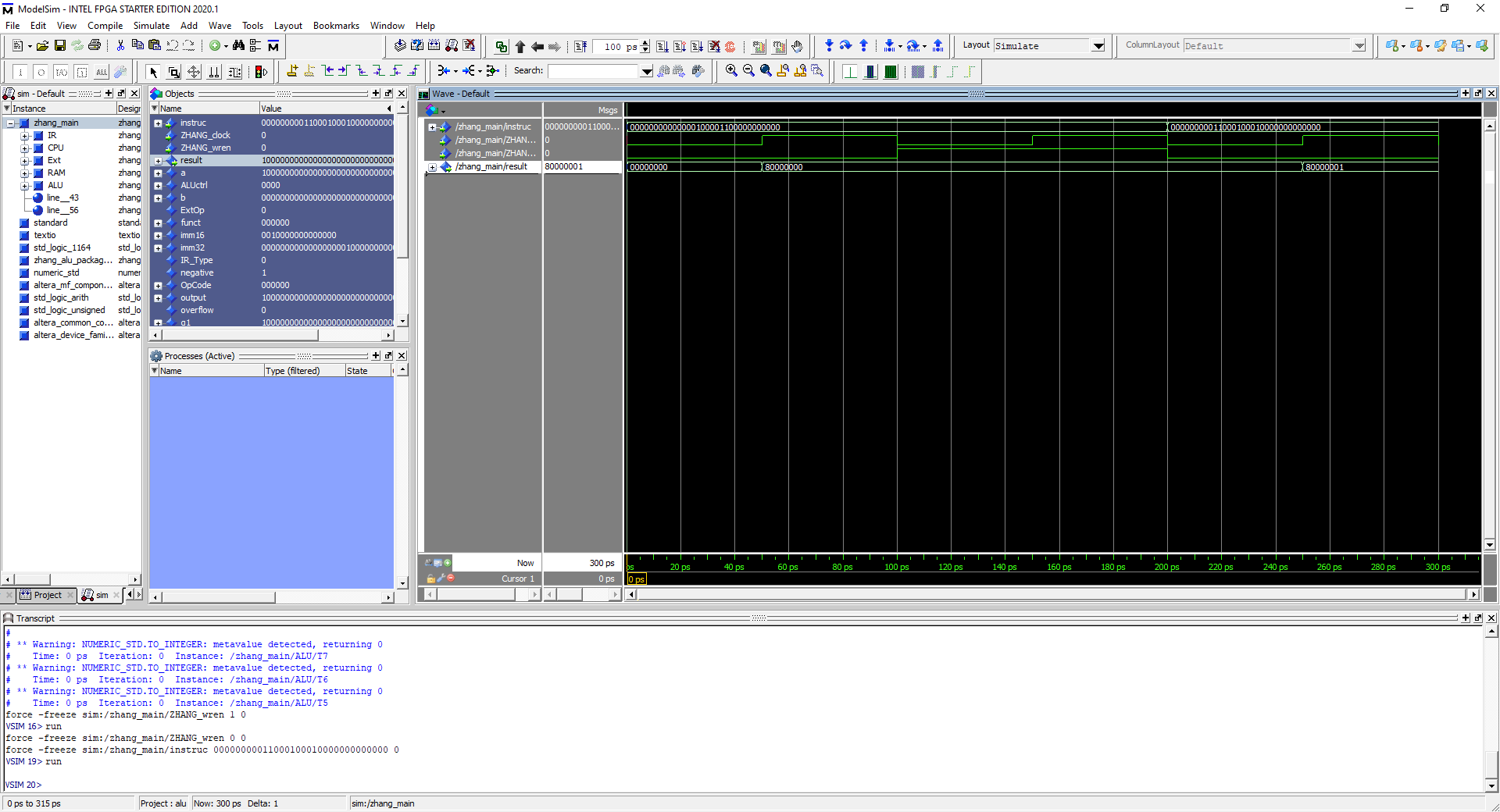


The Main file which houses every single function, the IR, The Control Unit, the Sign Extension, the 3 port RAM and the ALU. I have organized it in such a way to help prepare for future laboratory assignments.

# Waveforms and Explanations

In this section, I will be showcasing screenshots of the waveforms generated by the ALU as well as providing explanations as to what is being performed. There will also be a small breakdown of what is presented for the reader.

## 3-Port Ram, Memory Access

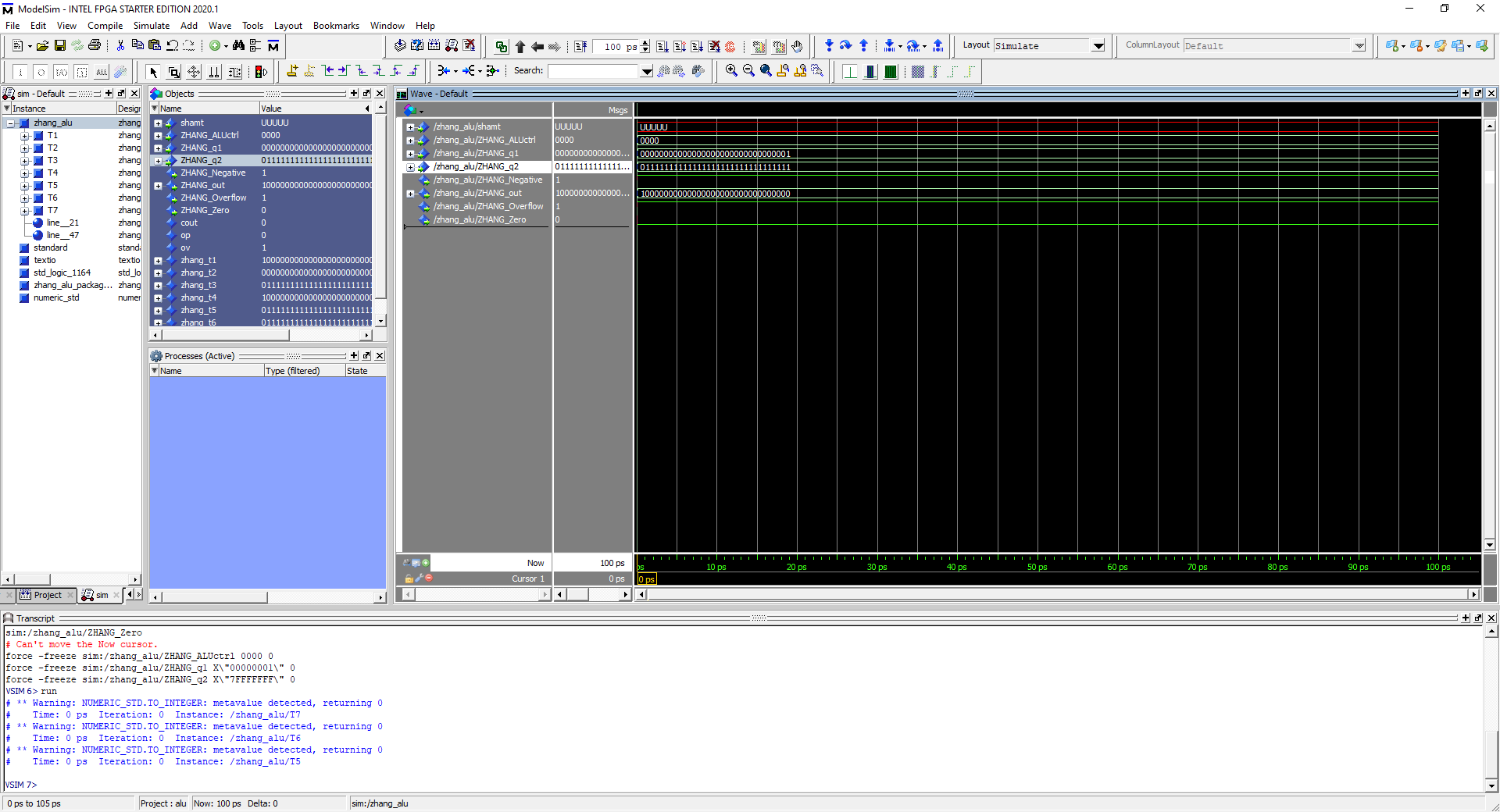


Overview of the 3-Port RAM being used to write into memory and read from memory. In the example above, instruction code is “000000 00000 00010 00011 00000000000” which breaks down to

OpCode = “000000” RS = “00000” , RT = “00010”, RD = “00011”, Shamt = “00000”, Funct = “00000”, imm16 = “000110000000000”

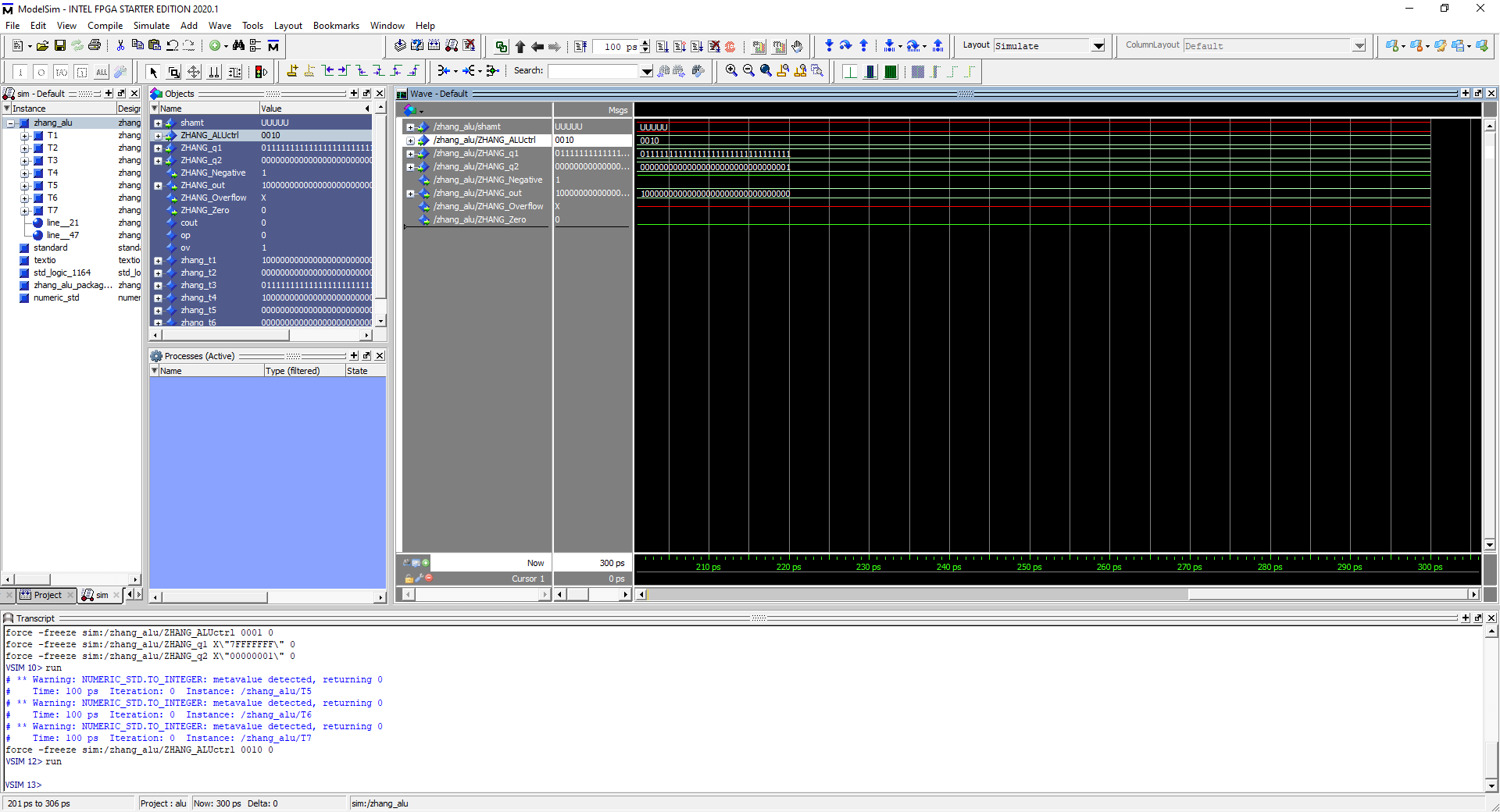
In the image above, what occurs is the add operation performed on the values stored in the address 00000 and 00010 then stored into 00011. The computation is done during the 1st clock cycle then stored into RD during the 2nd clock cycle then we read RD in the 3rd clock cycle

## Add

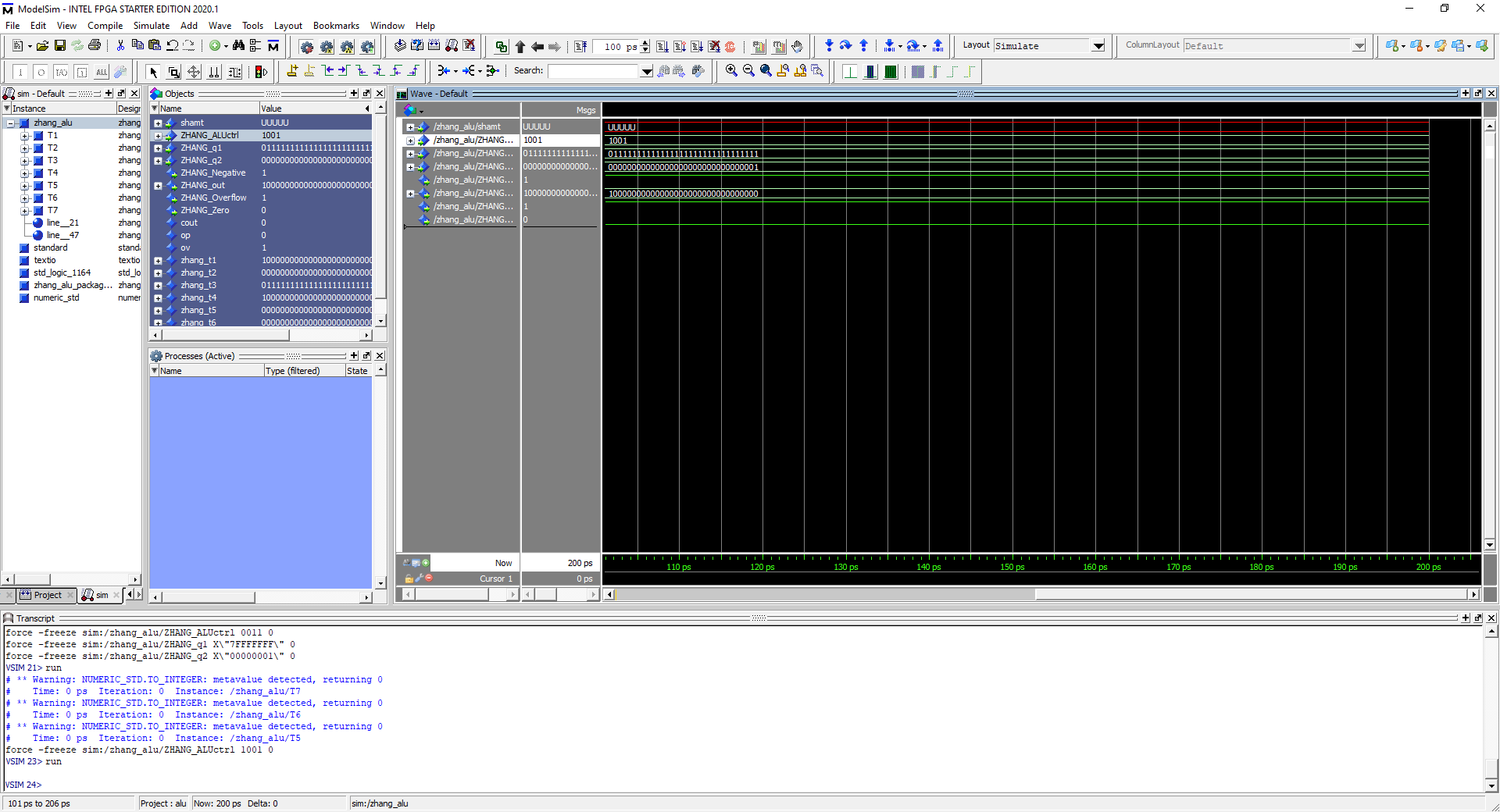


Op code = 0000, 7FFF FFFF add 0000 0001, Output = 8000 0000, All flags shown. Takes two operands and perform add operation onto them

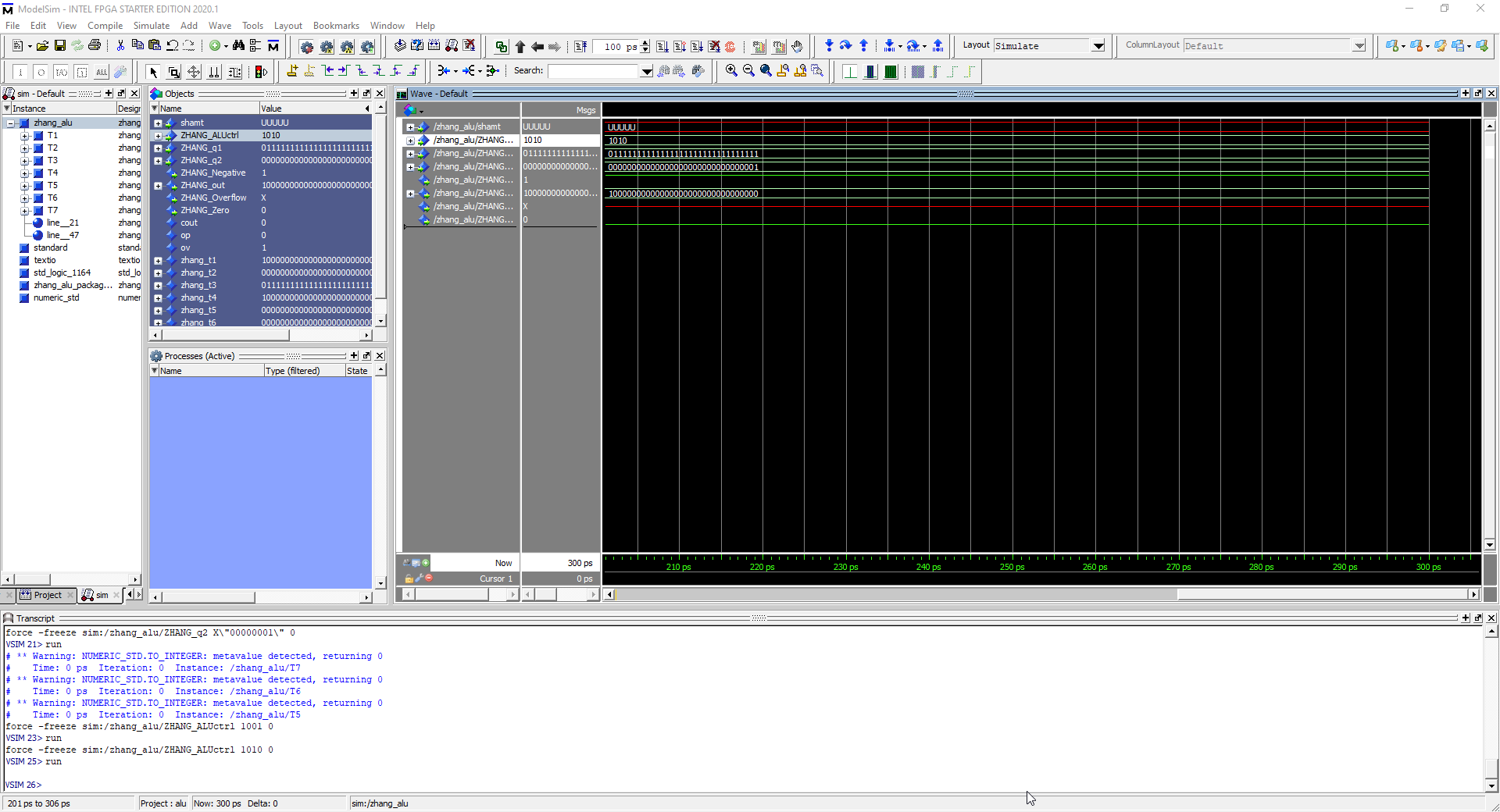
## Addu

 Opcode = 0010, 7FFF FFFF addu 0000 0001, Output = 8000 0000 with no overflow. Takes two operands and performs add operation on them and returns no overflow flag.

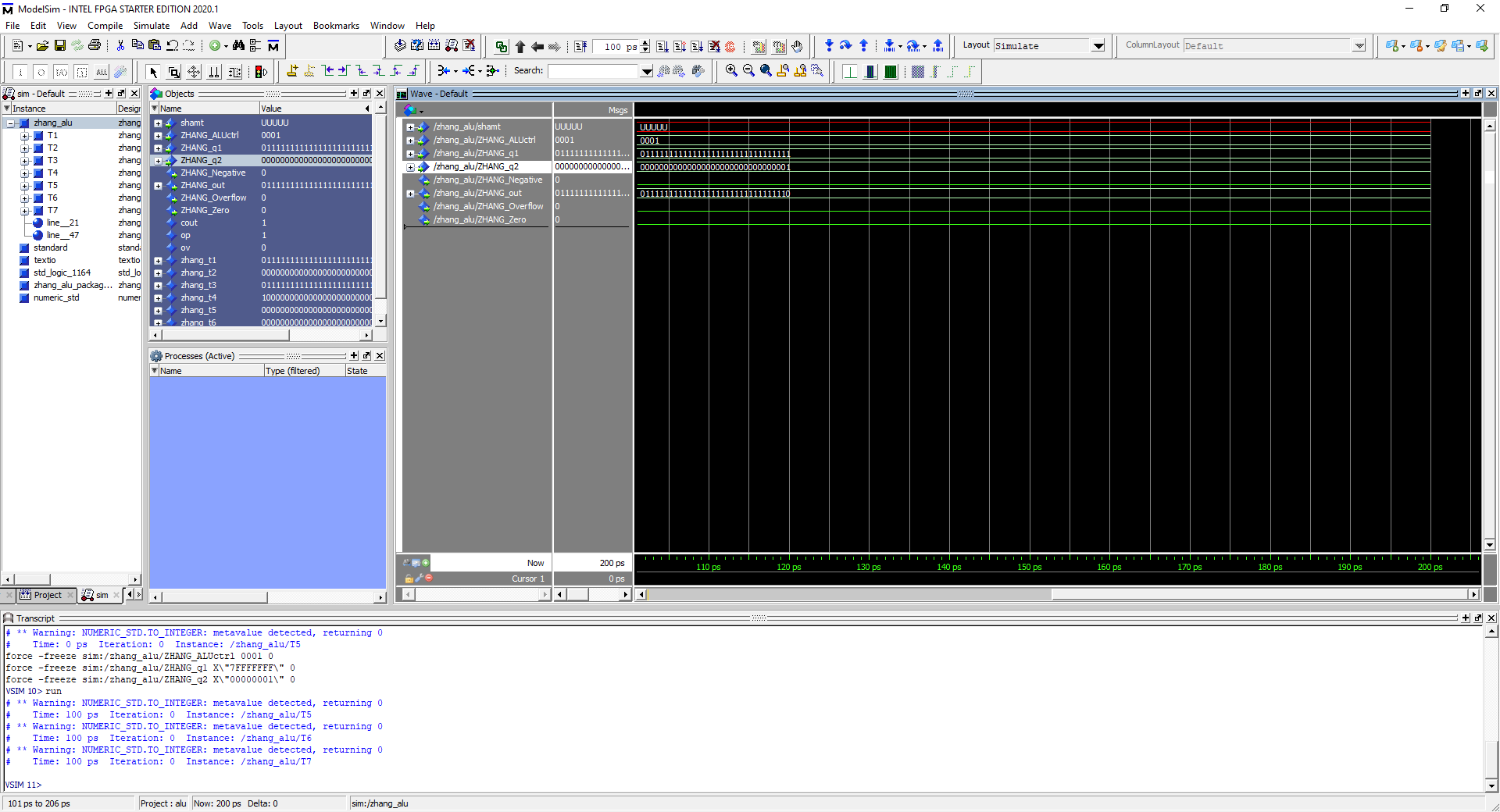
## Addi

 Opcode: 1001, 7FFF FFFF addi 0000 0001, Output = 8000 0000 with overflow flag. Performs Addi operation on operand 1 and imm16 with sign extension

## Addiu

 Opcode = 1010, 7FFF FFFF addiu 0000 0001, Output = 8000 0000 with no overflow. Addi performed on operand 1 and imm16 with sign extension without overflow flag.

## Sub

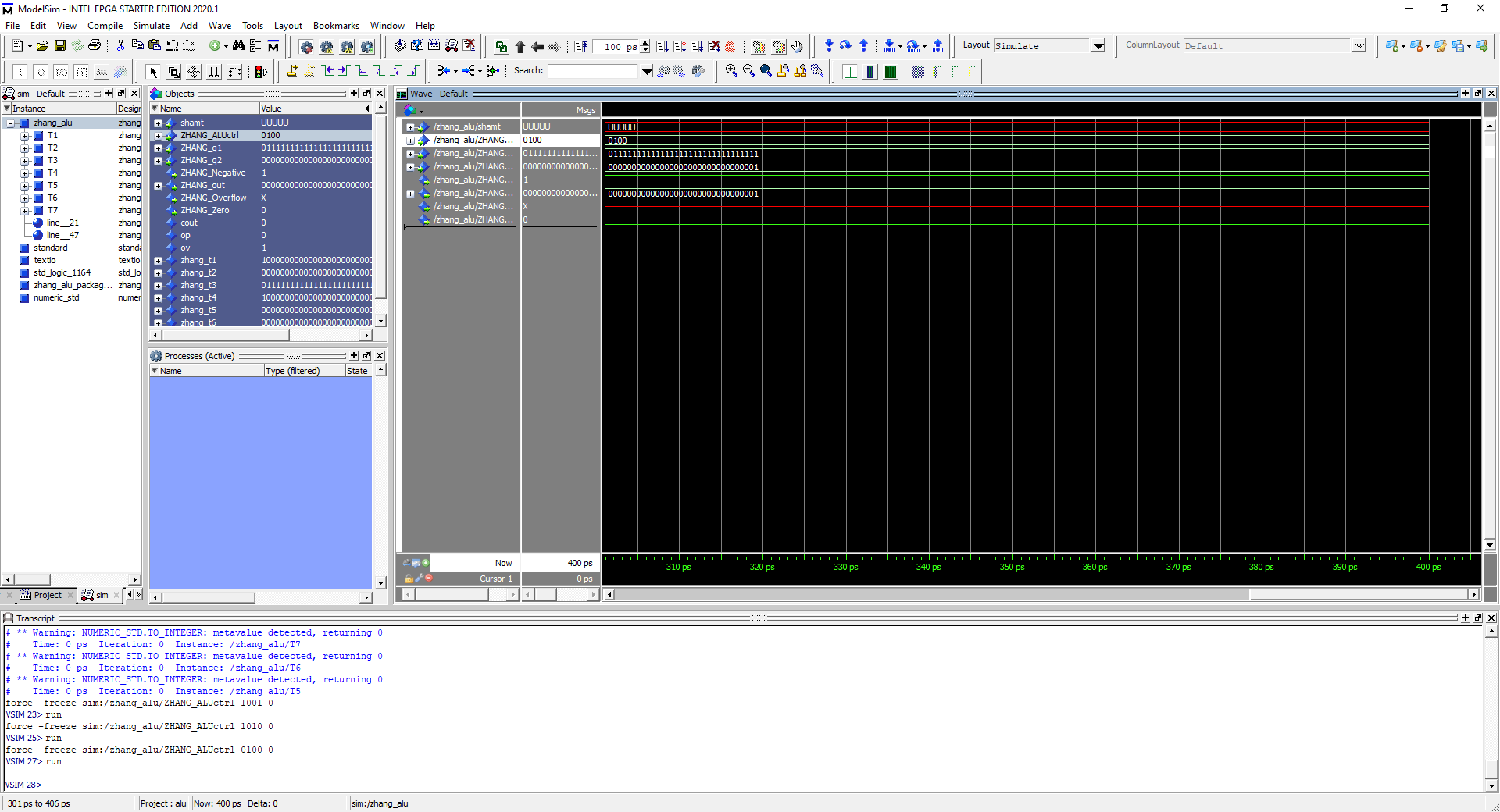
 Opcode = 0001, 7FFF FFFF – 0000 0001 = 7FFF FFFE, All flags shown. Sub performed on operands with overflow flag

## Subu

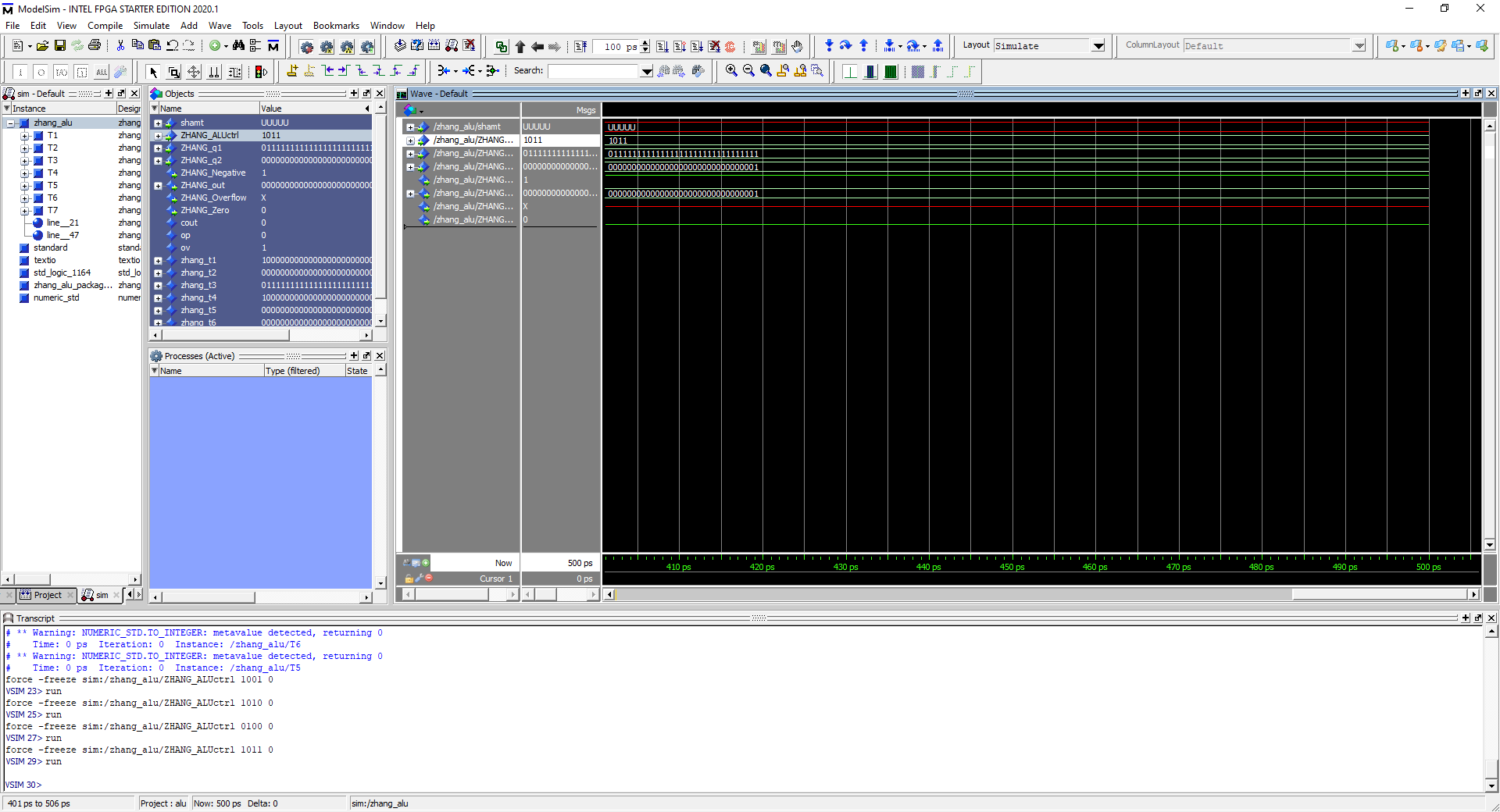
Graphical user interface

Description automatically generated Opcode = 0001, 7FFF FFFF subu 0000 0001 = 7FFF FFFE, no overflow flag shown. Sub Operation performed on both operands without overflow flag

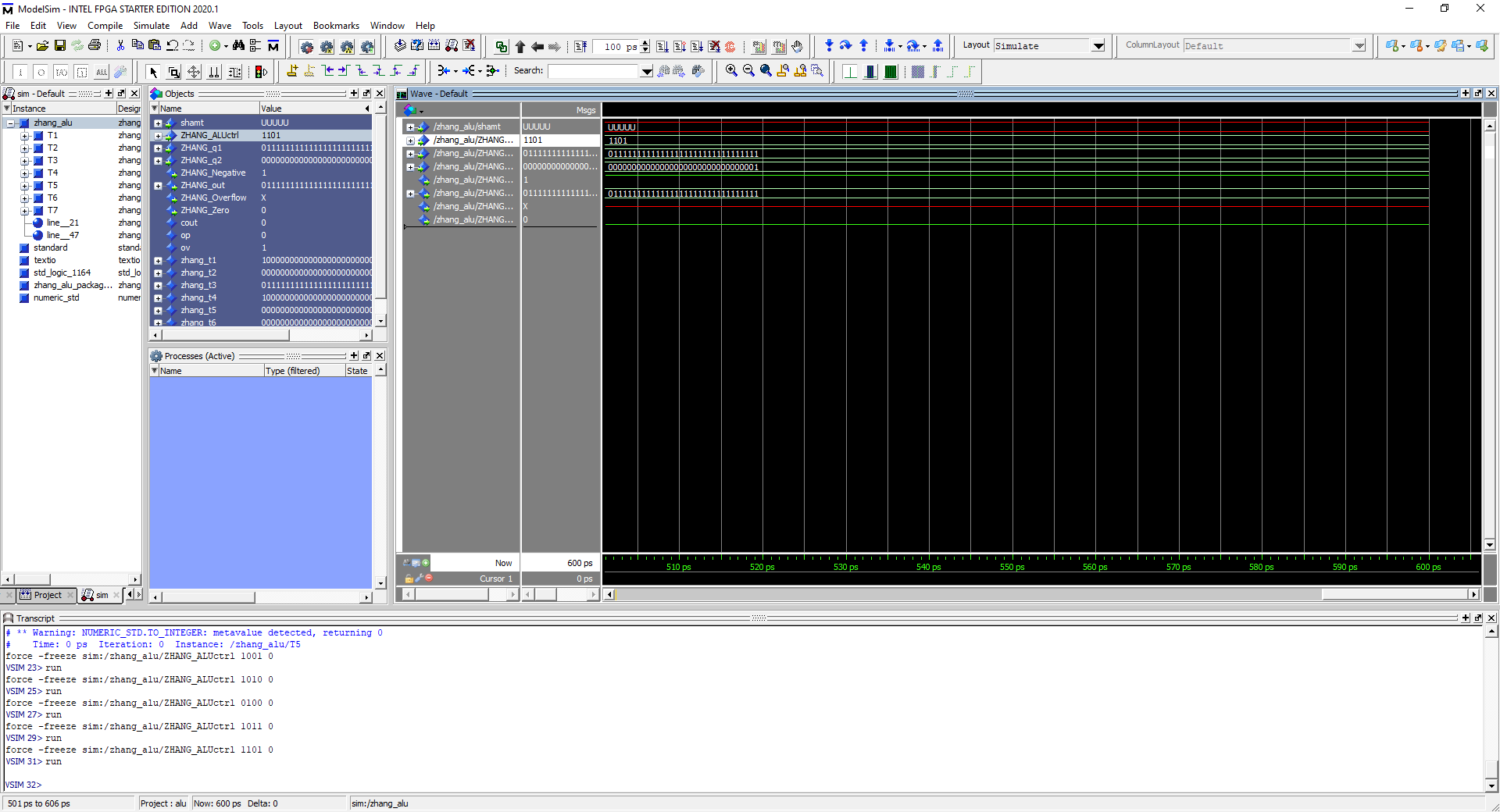
## And

 Opcode = 0100, 7FFF FFFF and 0000 0001 = 0000 0001 . And operation performed on both operands.

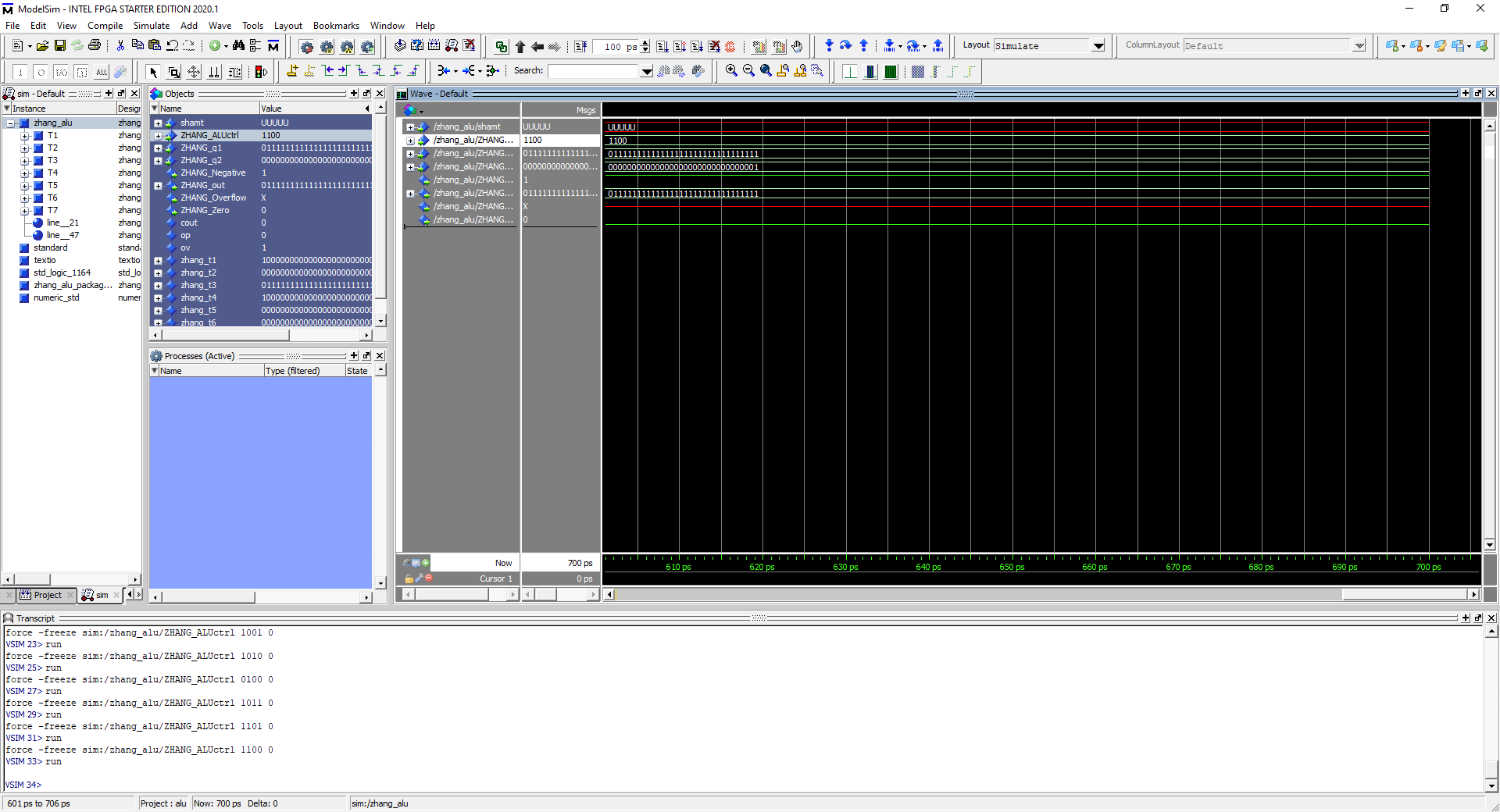
## Andi

 Opcode = 1011, 7FFF FFFF andi 0000 0001. And operation performed with operand 1 and imm16 with zero sign extension

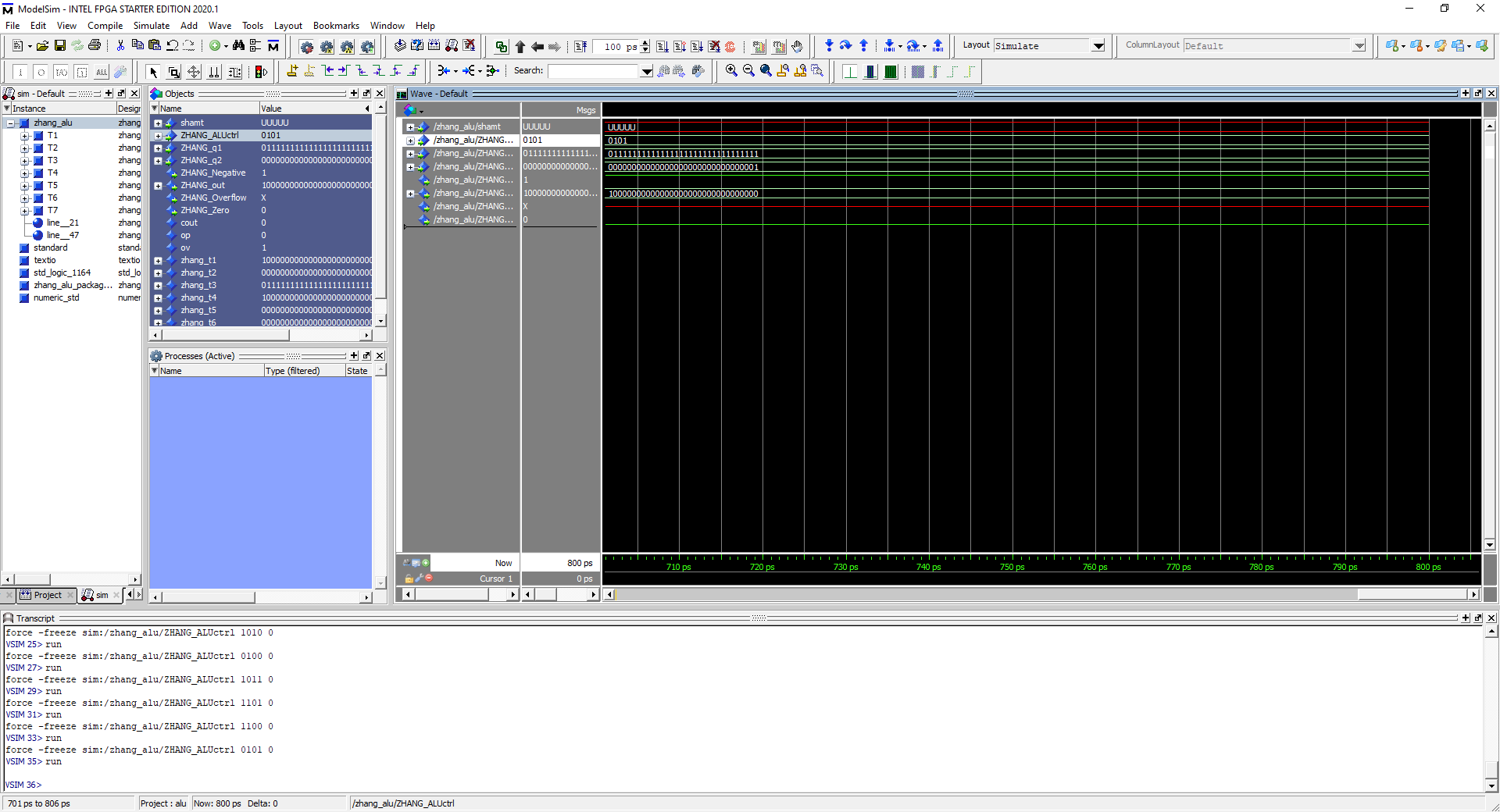
## Or

 Opcode = 1101, 7FFF FFFF or 0000 0001 = 7FFF FFFF. Or operation performed on both operands

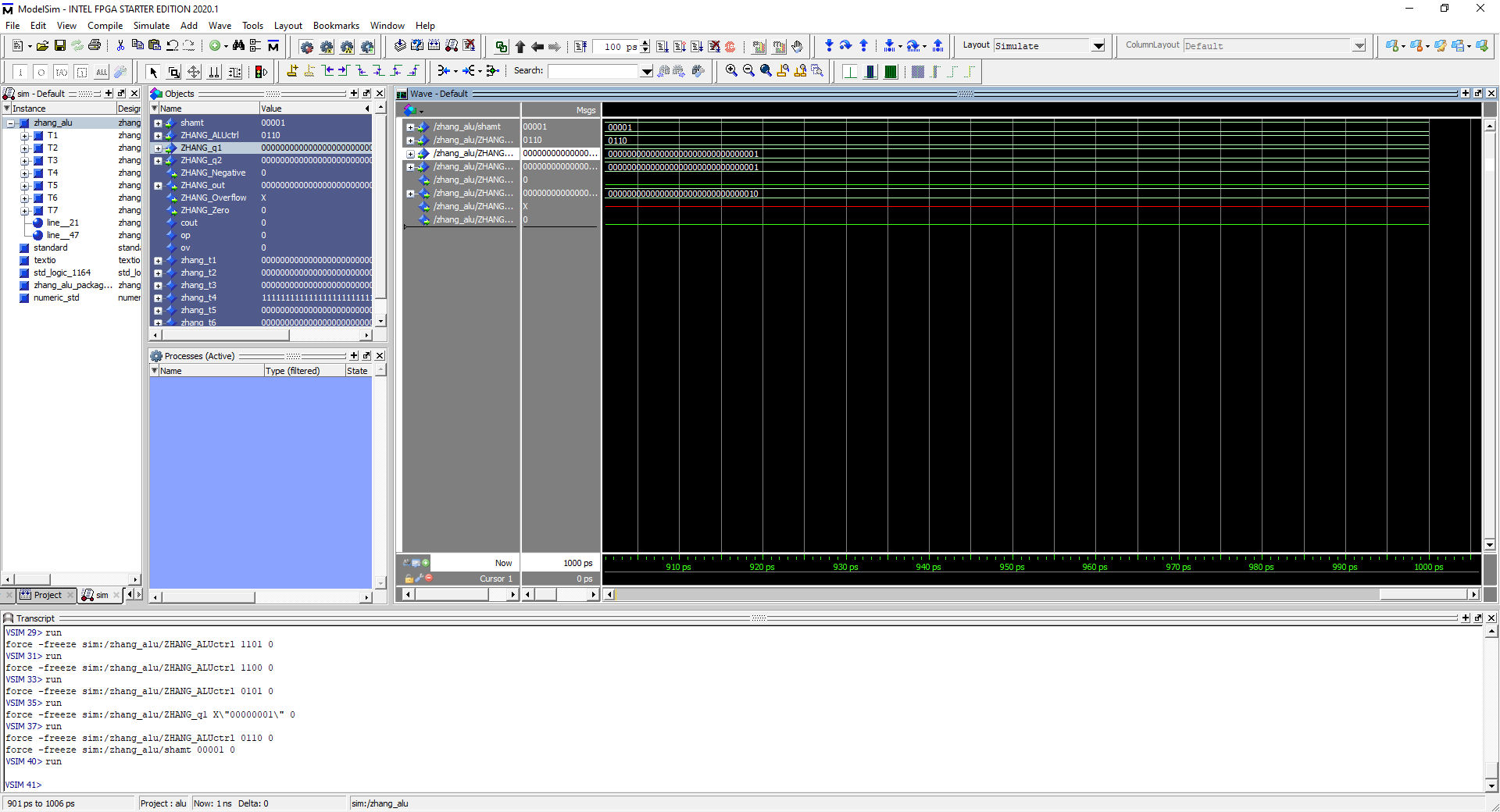
## Ori

 Opcode = 1100, 7FFF FFFF ori 0000 0001 = 7FFF FFFF. Or immediate operation performed on operand1 and imm16 bit with zero extensi

## Nor

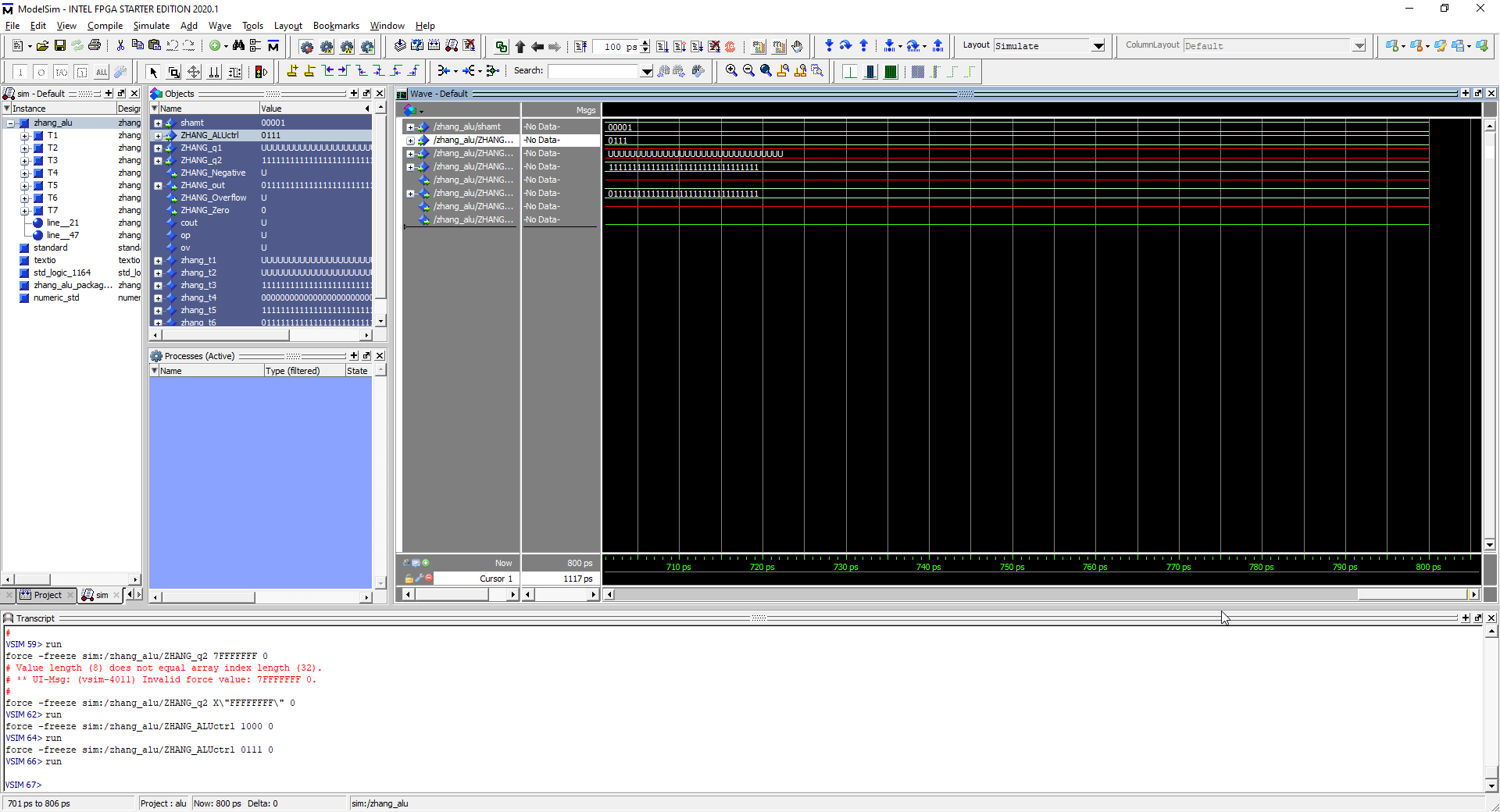
 Opcode = 0101, 7FFF FFFF nor 0000 0001 = 8000 0000, NOR operation performed on operand 1 and operands

## Shift Left



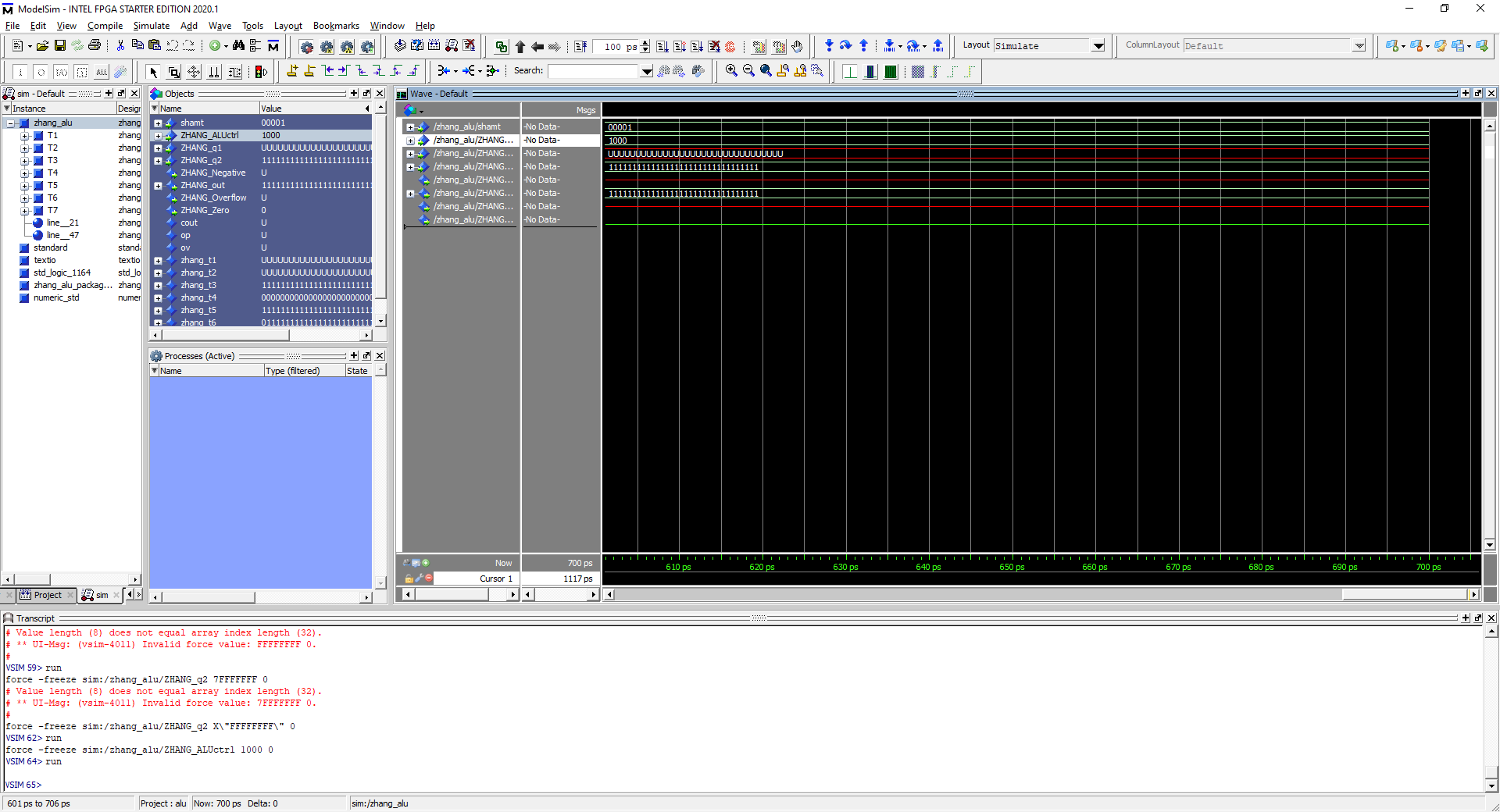
Opcode = 0110, Shamt = 00001, input = 0000 0001, Output = 0000 0010. Shift left operation performed on operand 2 with shift amount of 00001.

## Shift Right



Opcode = 0111, shamt = 00001, Input = 1111 1111, Output = 0111 1111. Shift right operand performed on Operand 2 with shift amount of 1 giving us 0111 1111.

## Shift Right Arithmetic

 Opcode = 1000, shamt = 00001, input = 1111 1111, output = 1111 1111. Shift right Arithmetic performed on operand 2. Unlike Shift right, 0’s don’t replace left most bit.

# Concluding Statement

In this laboratory assignment, I learned much about using a control unit and how important it may be when working with information decoded. With just the operation code, I was able to determine the ALUctrl, the Extension operation and the IR-type. Without a doubt, this will be helpful in future laboratory assignments and from the last laboratory assignment, I have improved my ability in connecting components together such as the IR-register, 3-Port Ram, Extension , Control Unit and the ALU.