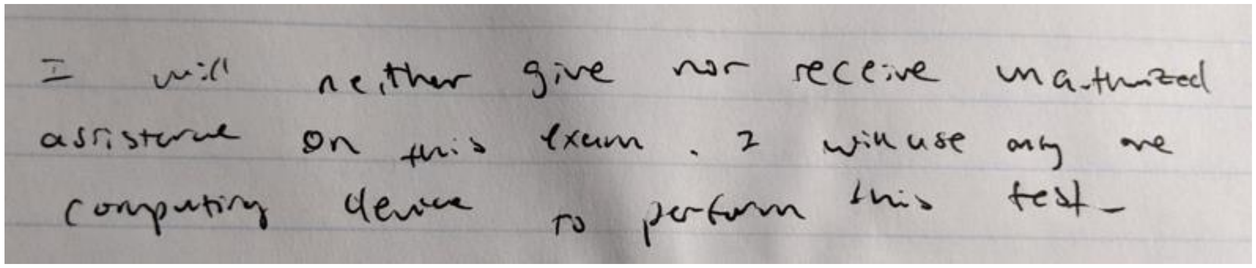
Lab : ALU

Chue Zhang

Csc343 Fall 2021

Professor Gertner

Signature :



I will neither give nor receive unauthorized assistance on this exam. I will use only one computing device to perform this test.

Table of Contents

Objective 4
Specification 5
    Or 6
    And 7
    Nor 8
    Shift Left(SL) 9
    Shift Right(SR) 10
    Shift Right Arithmetic(SRA) 11
    Sign Extend 12
    Control Unit 14
    ALU 15
    Main 17
Waveforms and Explanations 18
    3-Port Ram, Memory Access 19
    Add 20
    Addu 21
    Addi 22
    Addiu 23
    Sub 24
    Subu 25
    And 26
    Andi 27
    Or 28

Ori 29
Nor 30
Shift Left 31
Shift Right 32
Shift Right Arithmetic 33
Concluding Statement 34

Objective

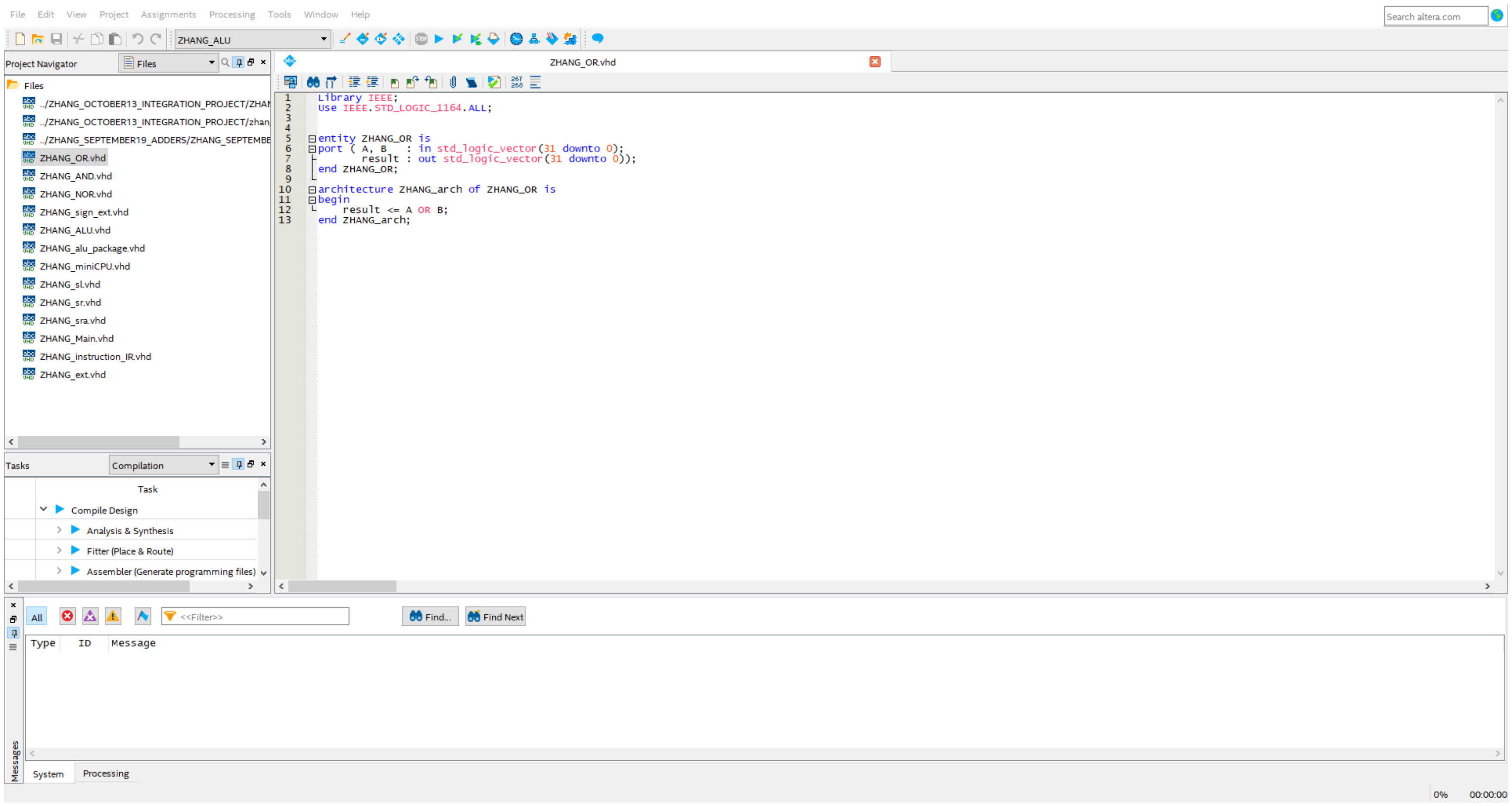
Objective of this laboratory assignment is to create an ALU that follows the MIPS green pages. A side objective of this assignment is to learn more about how an ALU performs including its components such as RS, RT, RD, Shamt, Funct and the immediate 16. All of which are extremely important for the development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

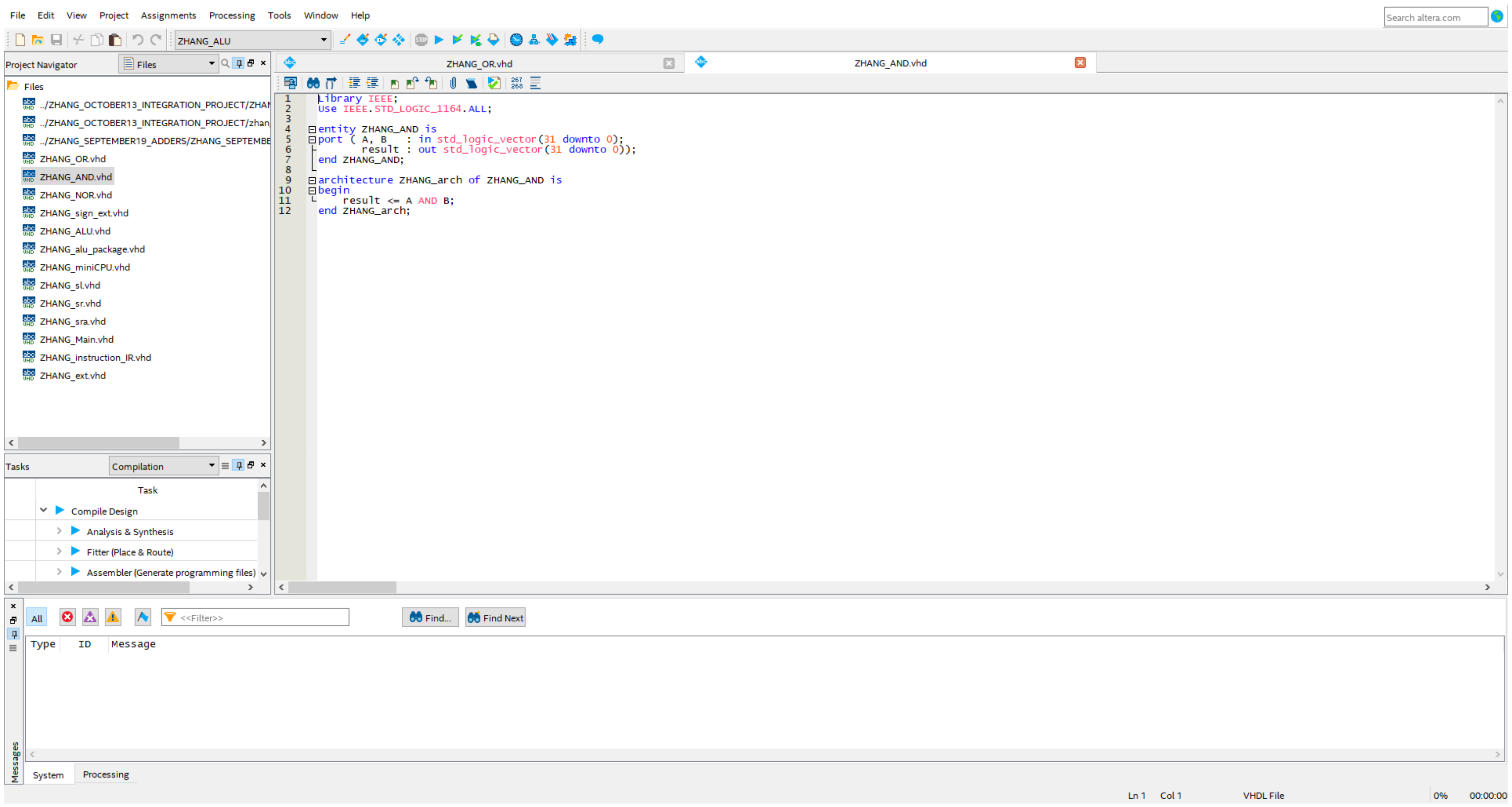


Files:
- ../ZHANG_OCTOBER13_INTEGRATION_PROJECT/ZHAN
- ../ZHANG_OCTOBER13_INTEGRATION_PROJECT/zhan
- ../ZHANG_SEPTEMBER19_ADDERS/ZHANG_SEPTEMBE
- ZHANG_OR.vhd
- ZHANG_AND.vhd
- ZHANG_NOR.vhd
- ZHANG_sign_ext.vhd
- ZHANG_ALU.vhd
- ZHANG_alu_package.vhd
- ZHANG_miniCPU.vhd
- ZHANG_sl.vhd
- ZHANG_sr.vhd
- ZHANG_sra.vhd
- ZHANG_Main.vhd
- ZHANG_instruction_IR.vhd
- ZHANG_ext.vhd

```
Library IEEE;
Use IEEE.STD_LOGIC_1164.ALL;


entity ZHANG_OR is
port ( A, B   : in std_logic_vector(31 downto 0);
       result : out std_logic_vector(31 downto 0));
end ZHANG_OR;

architecture ZHANG_arch of ZHANG_OR is
begin
    result <= A OR B;
end ZHANG_arch;
```

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

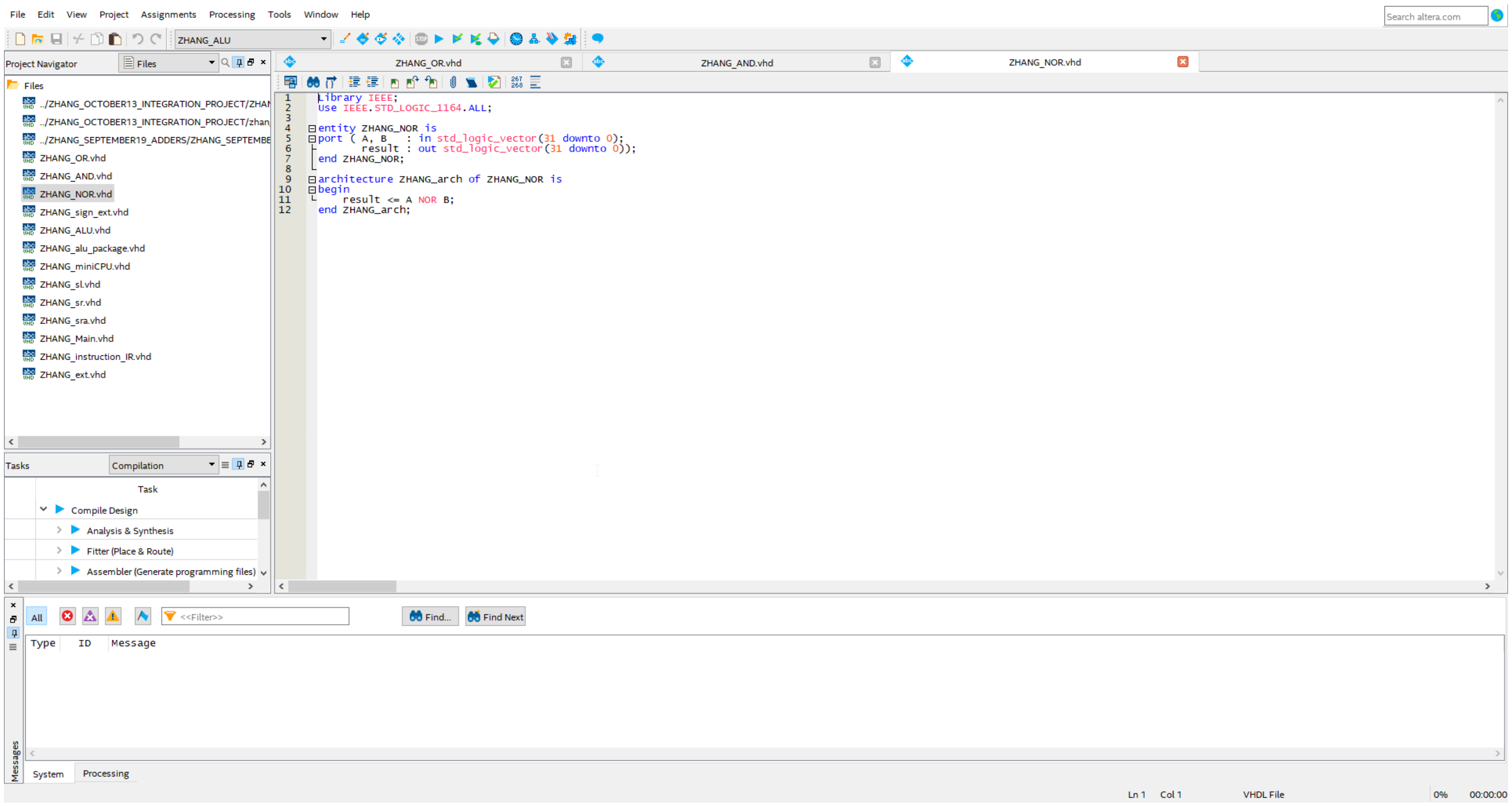
And



```vhdl
Library IEEE;
Use IEEE.STD_LOGIC_1164.ALL;

entity ZHANG_AND is
port ( A, B   : in std_logic_vector(31 downto 0);
       result : out std_logic_vector(31 downto 0));
end ZHANG_AND;

architecture ZHANG_arch of ZHANG_AND is
begin
    result <= A AND B;
end ZHANG_arch;
```

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

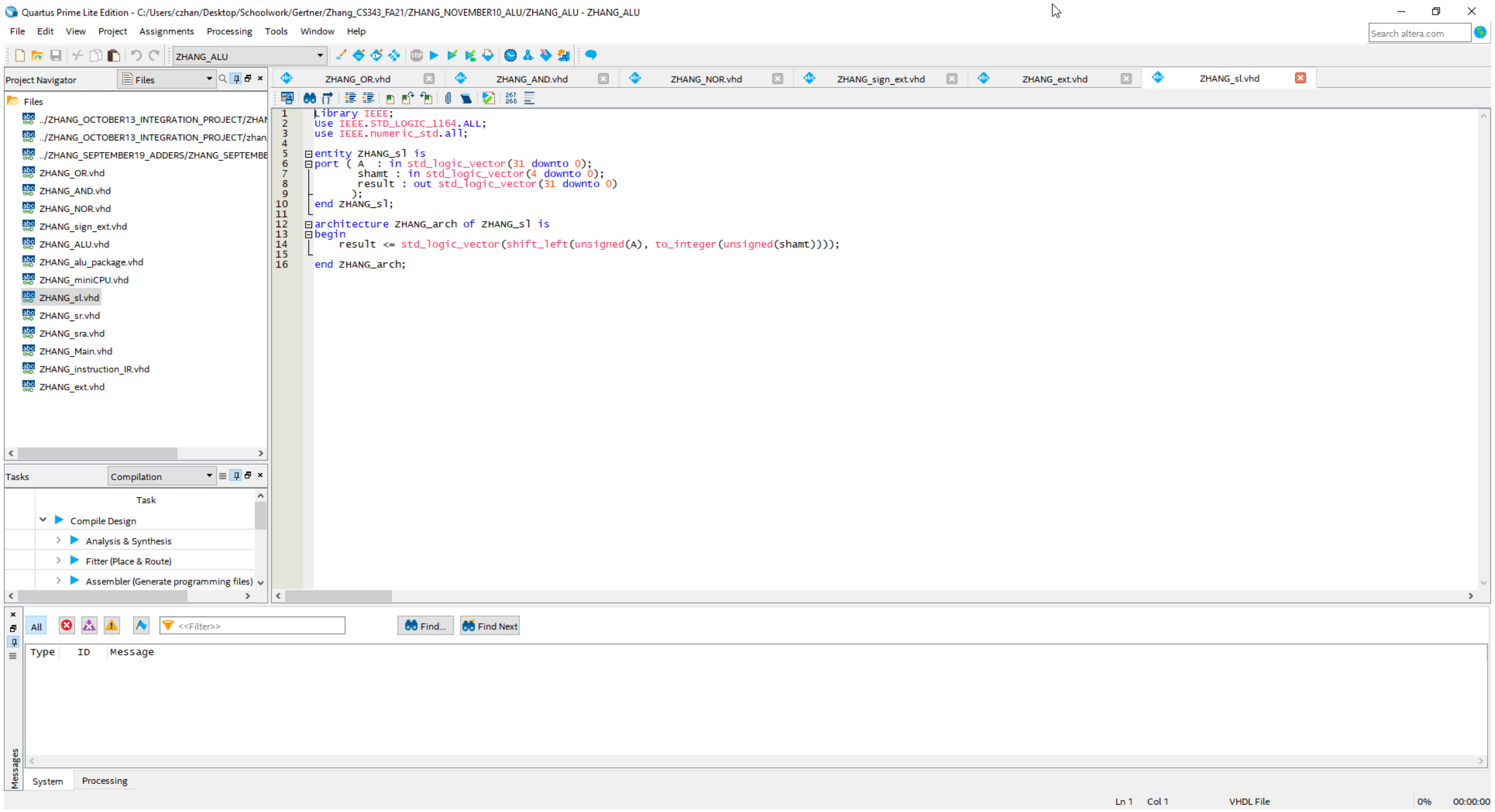
Nor



```vhdl
Library IEEE;
Use IEEE.STD_LOGIC_1164.ALL;

entity ZHANG_NOR is
port ( A, B   : in std_logic_vector(31 downto 0);
       result : out std_logic_vector(31 downto 0));
end ZHANG_NOR;

architecture ZHANG_arch of ZHANG_NOR is
begin
    result <= A NOR B;
end ZHANG_arch;
```

Nor VHDL code

Shift Left(SL)

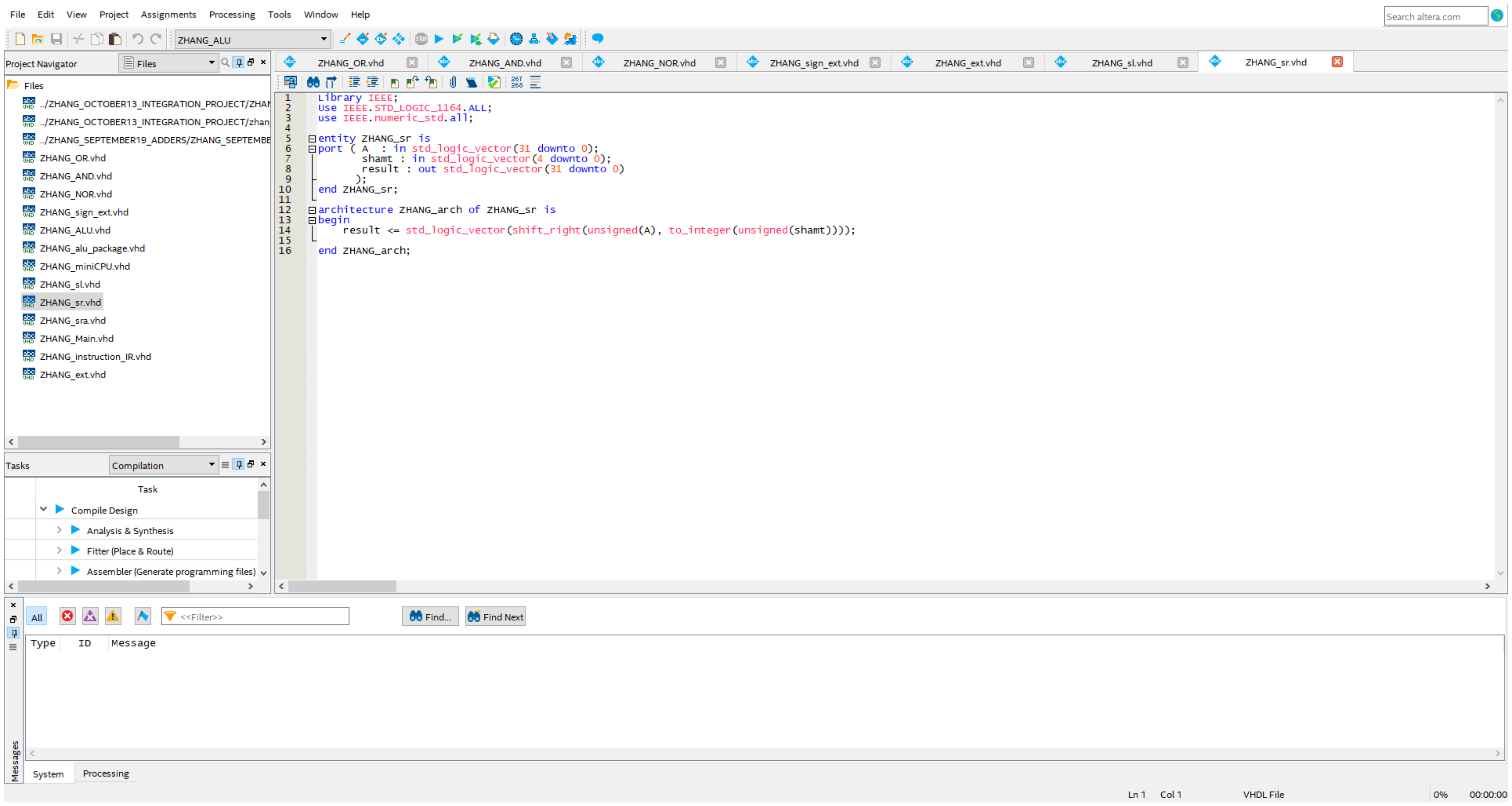

Quartus Prime Lite Edition - C:/Users/czhan/Desktop/Schoolwork/Gertner/Zhang_CS343_FA21/ZHANG_NOVEMBER10_ALU/ZHANG_ALU - ZHANG_ALU

Files:
- ../ZHANG_OCTOBER13_INTEGRATION_PROJECT/ZHAN
- ../ZHANG_OCTOBER13_INTEGRATION_PROJECT/zhan
- ../ZHANG_SEPTEMBER19_ADDERS/ZHANG_SEPTEMBE
- ZHANG_OR.vhd
- ZHANG_AND.vhd
- ZHANG_NOR.vhd
- ZHANG_sign_ext.vhd
- ZHANG_ALU.vhd
- ZHANG_alu_package.vhd
- ZHANG_miniCPU.vhd
- ZHANG_sl.vhd
- ZHANG_sr.vhd
- ZHANG_sra.vhd
- ZHANG_Main.vhd
- ZHANG_instruction_IR.vhd
- ZHANG_ext.vhd

```vhdl
Library IEEE;
Use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;

entity ZHANG_sl is
port ( A  : in std_logic_vector(31 downto 0);
       shamt : in std_logic_vector(4 downto 0);
       result : out std_logic_vector(31 downto 0)
      );
end ZHANG_sl;

architecture ZHANG_arch of ZHANG_sl is
begin
    result <= std_logic_vector(shift_left(unsigned(A), to_integer(unsigned(shamt))));

end ZHANG_arch;
```

Shift left VHDL code

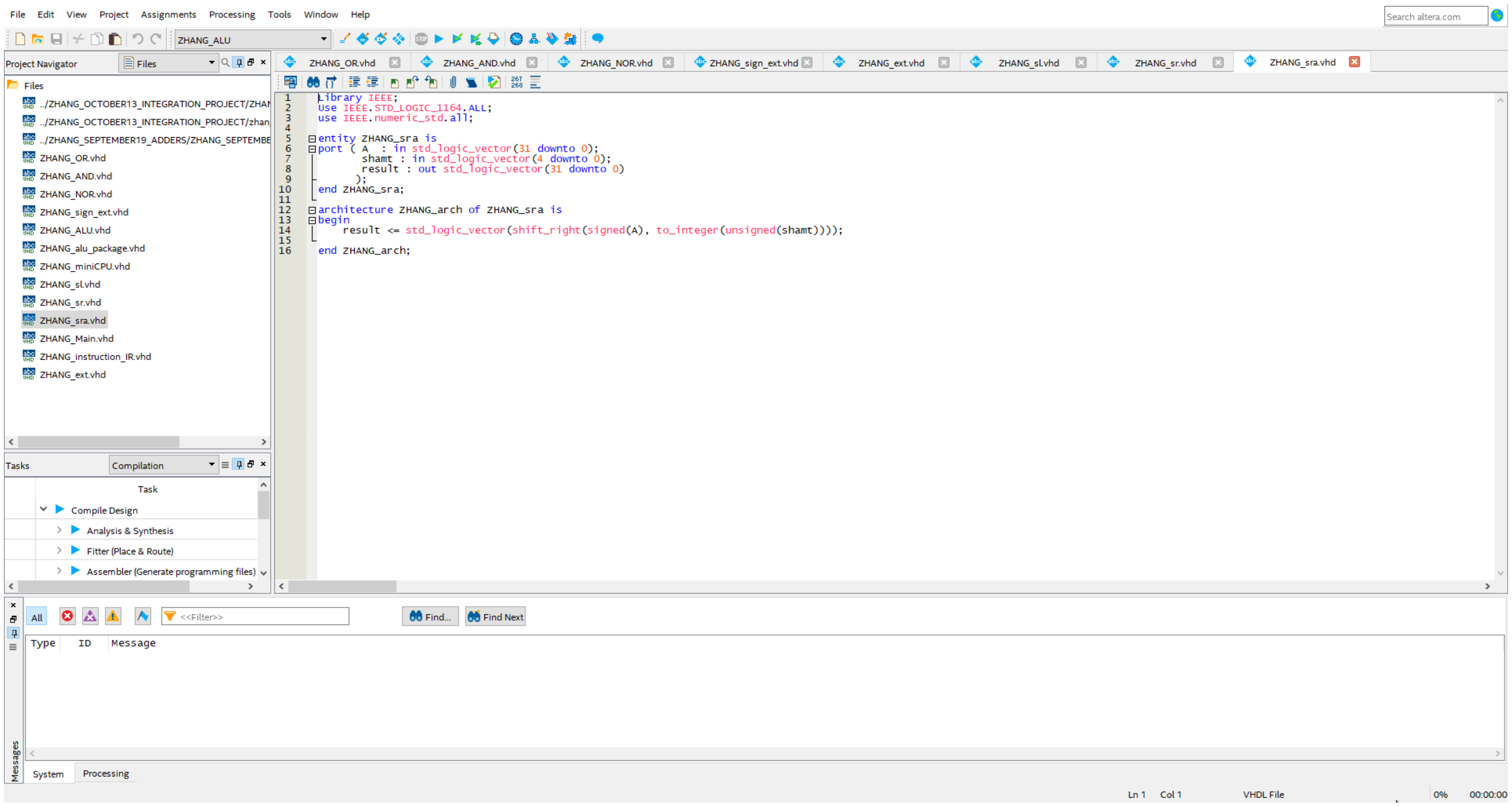
Shift Right(SR)



```vhdl
Library IEEE;
Use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;

entity ZHANG_sr is
port ( A   : in std_logic_vector(31 downto 0);
       shamt : in std_logic_vector(4 downto 0);
       result : out std_logic_vector(31 downto 0)
      );
end ZHANG_sr;

architecture ZHANG_arch of ZHANG_sr is
begin
    result <= std_logic_vector(shift_right(unsigned(A), to_integer(unsigned(shamt))));

end ZHANG_arch;
```

Shift right VHDL code

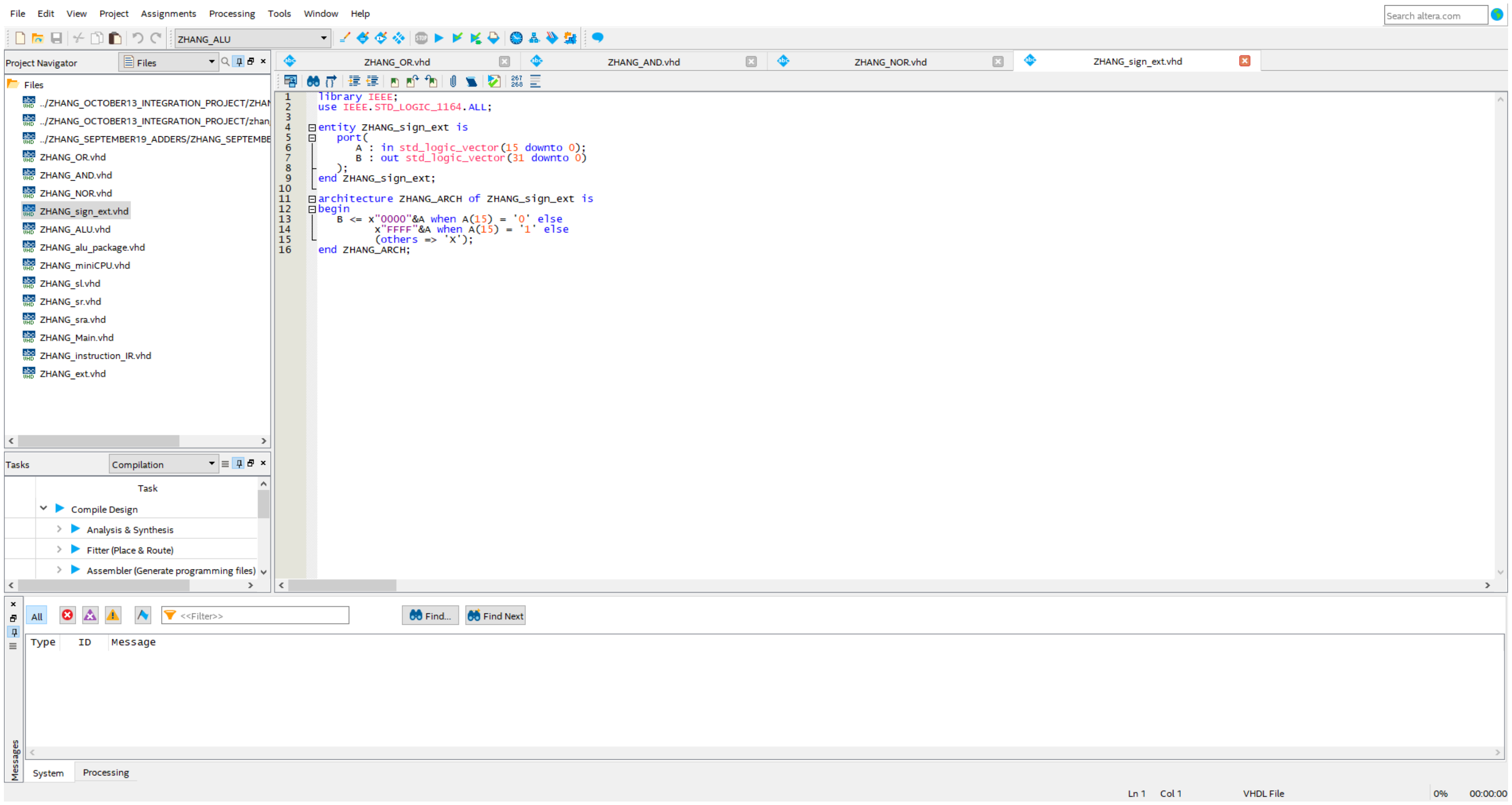
Shift Right Arithmetic(SRA)



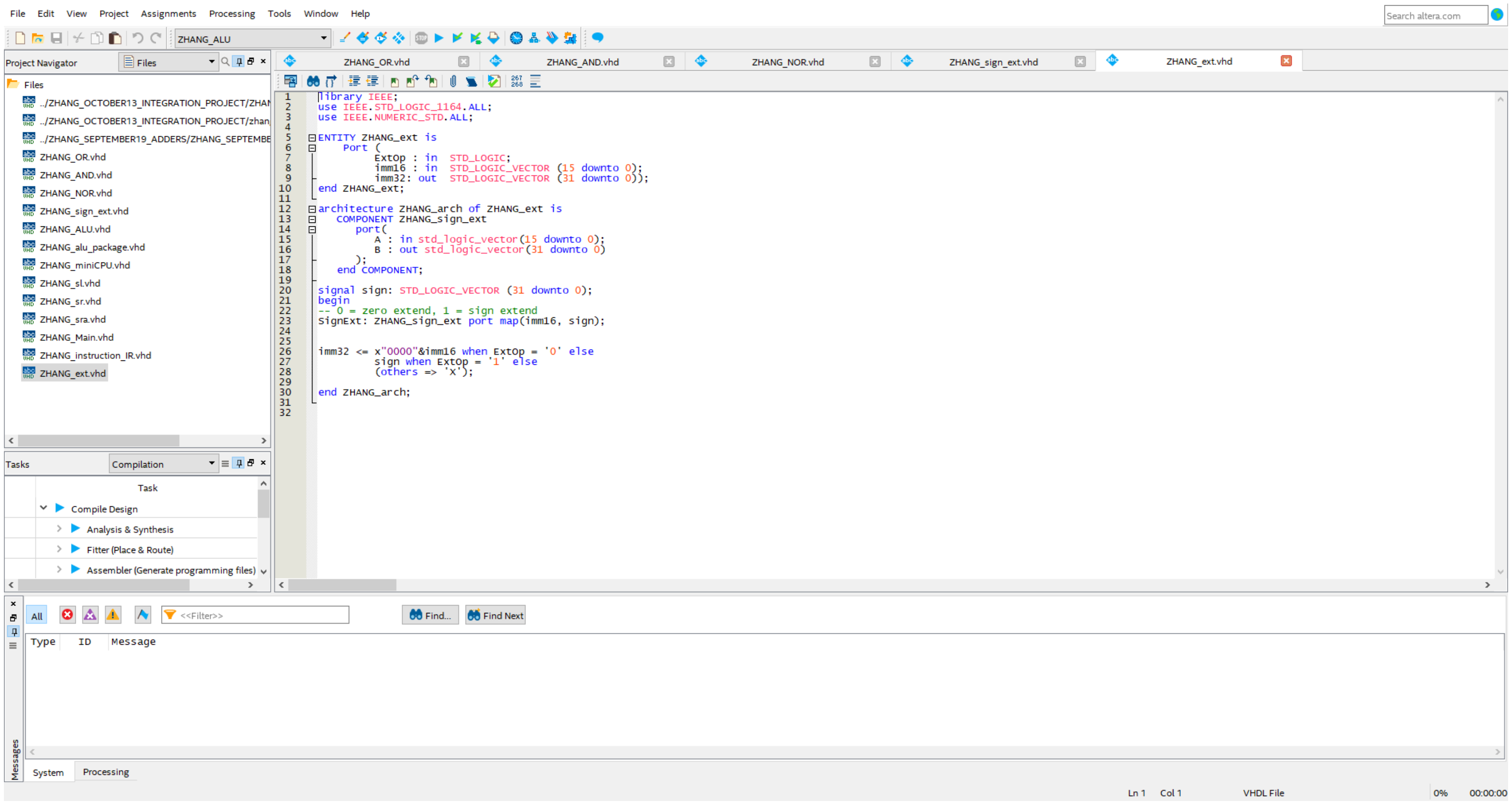
```vhdl
Library IEEE;
Use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;

entity ZHANG_sra is
port ( A   : in std_logic_vector(31 downto 0);
       shamt : in std_logic_vector(4 downto 0);
       result : out std_logic_vector(31 downto 0)
      );
end ZHANG_sra;

architecture ZHANG_arch of ZHANG_sra is
begin
    result <= std_logic_vector(shift_right(signed(A), to_integer(unsigned(shamt))));

end ZHANG_arch;
```

Shift Right Arithmetic VHDL code

Sign Extend



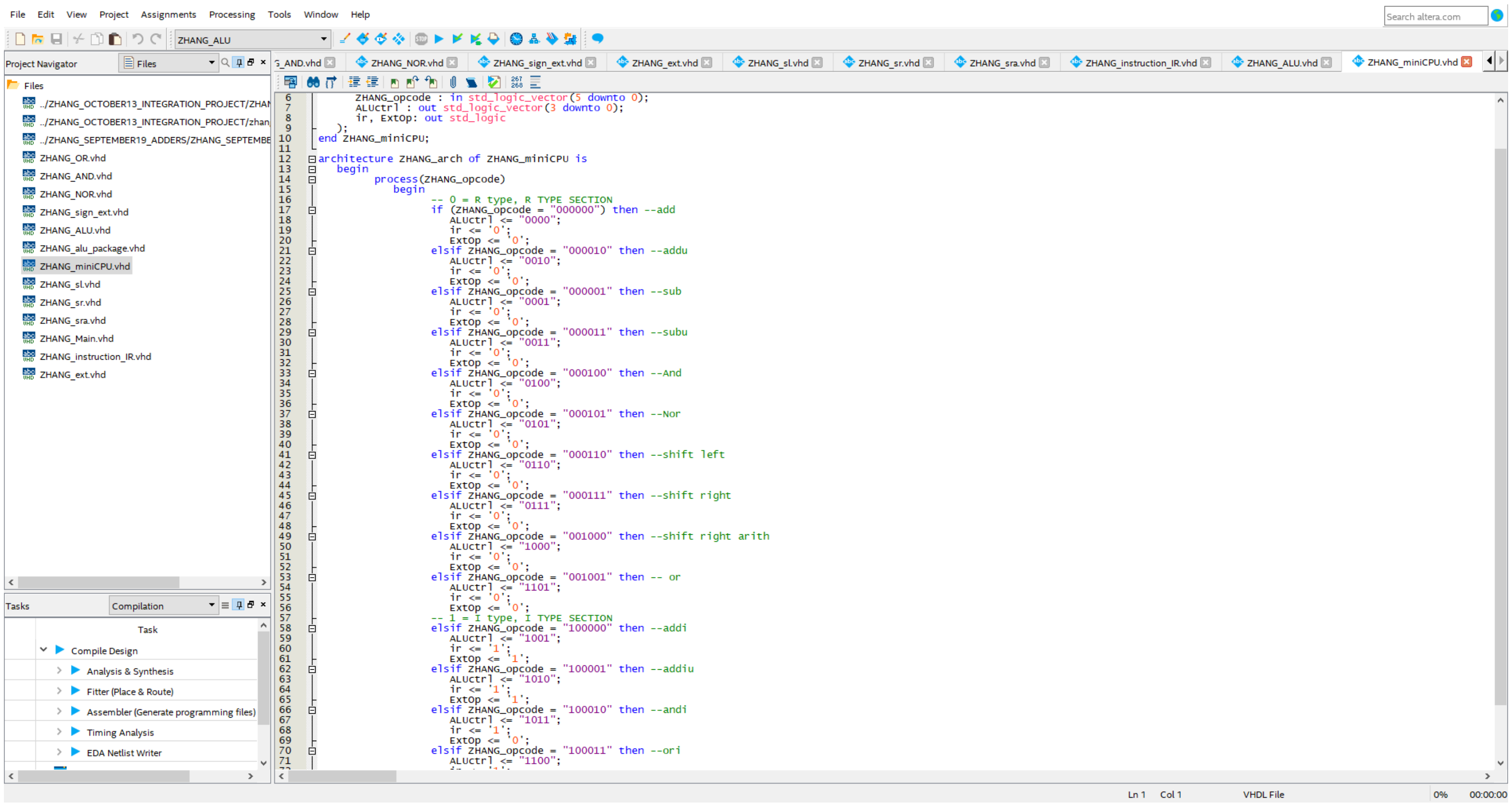
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity ZHANG_sign_ext is
    port(
        A : in std_logic_vector(15 downto 0);
        B : out std_logic_vector(31 downto 0)
    );
end ZHANG_sign_ext;

architecture ZHANG_ARCH of ZHANG_sign_ext is
begin
    B <= x"0000"&A when A(15) = '0' else
         x"FFFF"&A when A(15) = '1' else
         (others => 'X');
end ZHANG_ARCH;
```



```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

ENTITY ZHANG_ext is
    Port (
          ExtOp : in  STD_LOGIC;
          imm16 : in  STD_LOGIC_VECTOR (15 downto 0);
          imm32: out  STD_LOGIC_VECTOR (31 downto 0));
end ZHANG_ext;

architecture ZHANG_arch of ZHANG_ext is
   COMPONENT ZHANG_sign_ext
      port(
          A : in std_logic_vector(15 downto 0);
          B : out std_logic_vector(31 downto 0)
      );
   end COMPONENT;

signal sign: STD_LOGIC_VECTOR (31 downto 0);
begin
-- 0 = zero extend, 1 = sign extend
SignExt: ZHANG_sign_ext port map(imm16, sign);


imm32 <= x"0000"&imm16 when ExtOp = '0' else
          sign when ExtOp = '1' else
          (others => 'X');

end ZHANG_arch;
```

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.

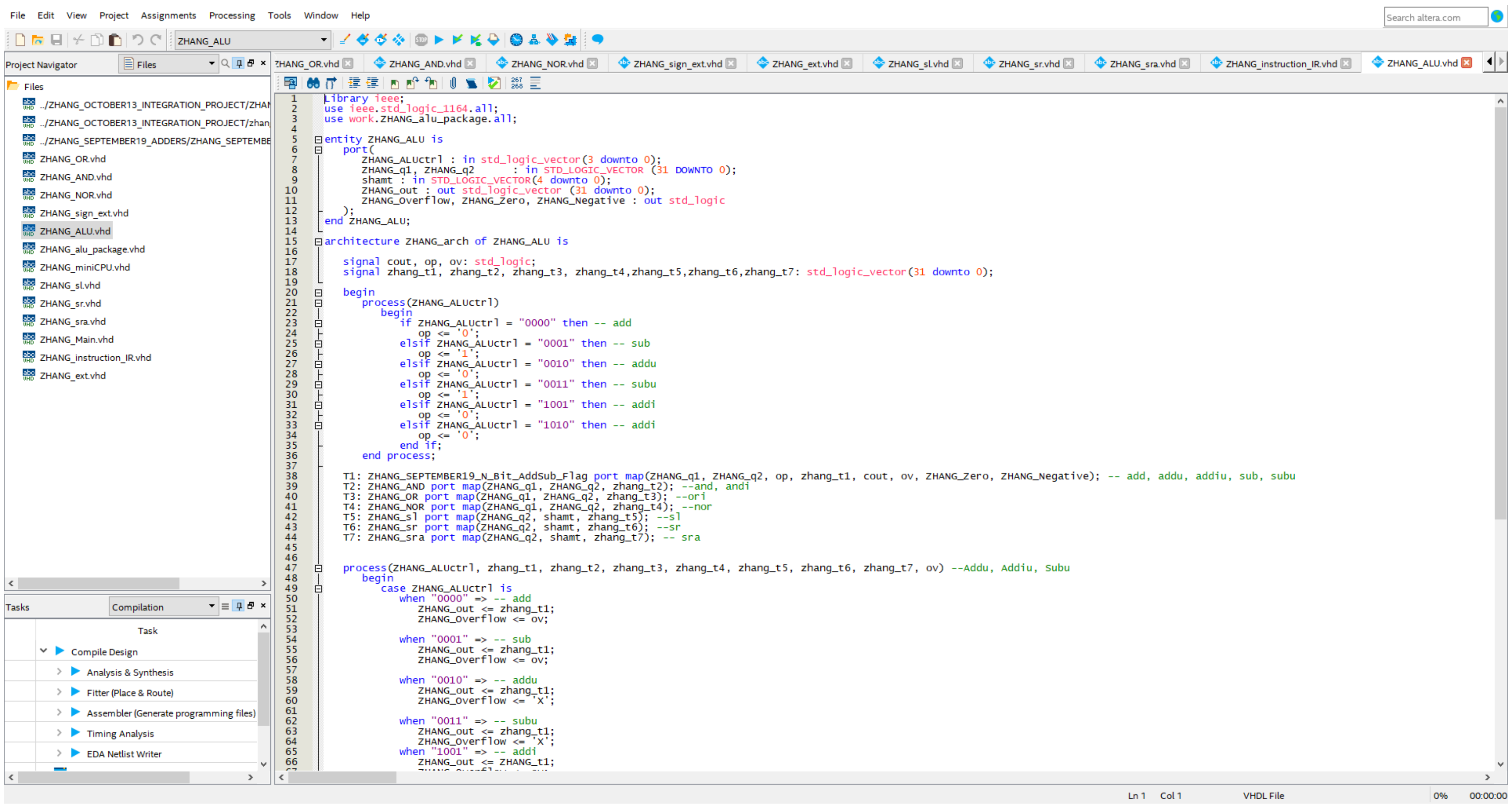
Control Unit



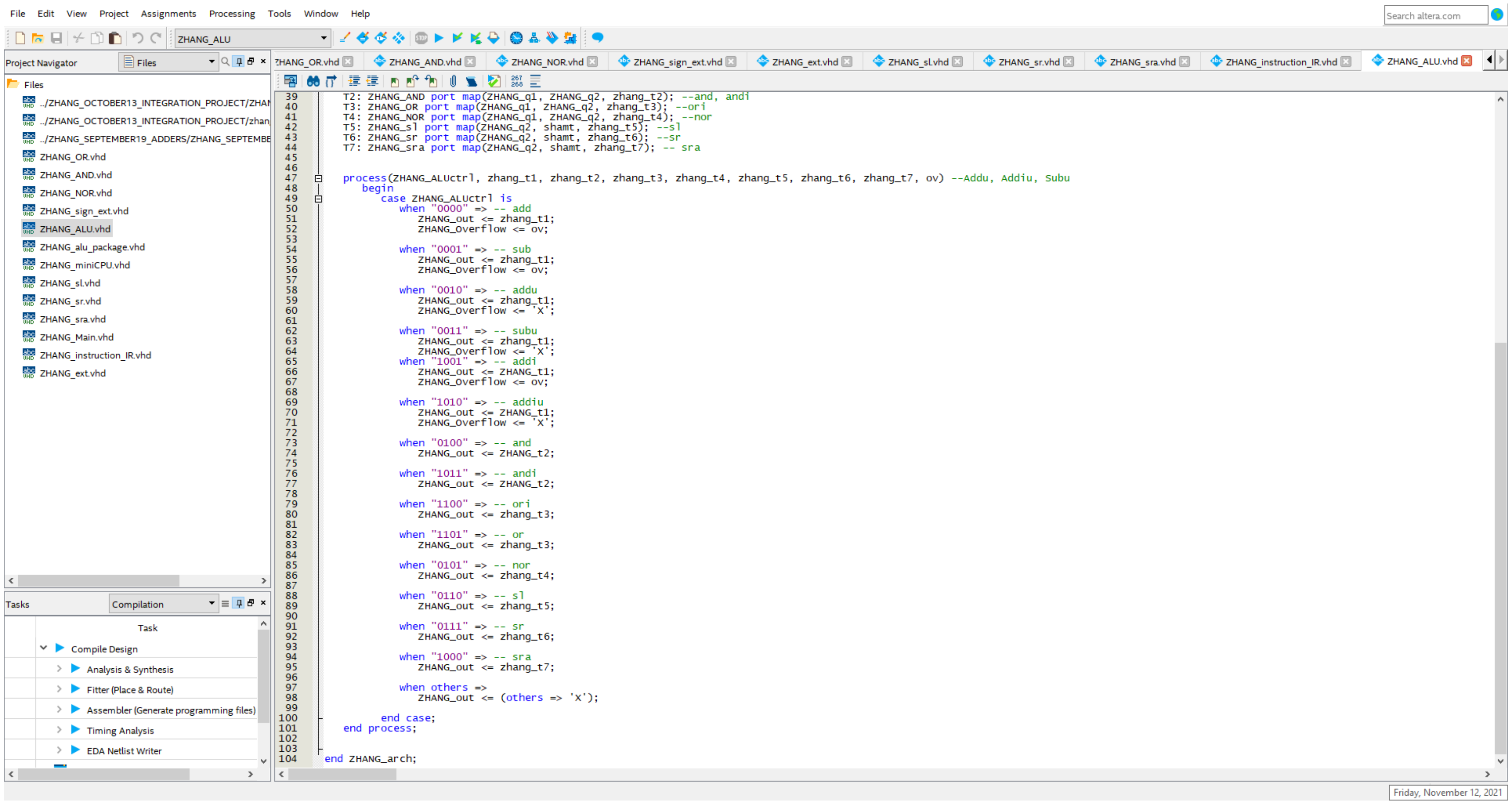
```
        ZHANG_opcode : in std_logic_vector(5 downto 0);
        ALUctrl : out std_logic_vector(3 downto 0);
        ir, Extop: out std_logic
    );
end ZHANG_miniCPU;

architecture ZHANG_arch of ZHANG_miniCPU is
   begin
        process(ZHANG_opcode)
           begin
                -- 0 = R type, R TYPE SECTION
                if (ZHANG_opcode = "000000") then --add
                   ALUctrl <= "0000";
                   ir <= '0';
                   Extop <= '0';
                elsif ZHANG_opcode = "000010" then --addu
                   ALUctrl <= "0010";
                   ir <= '0';
                   Extop <= '0';
                elsif ZHANG_opcode = "000001" then --sub
                   ALUctrl <= "0001";
                   ir <= '0';
                   Extop <= '0';
                elsif ZHANG_opcode = "000011" then --subu
                   ALUctrl <= "0011";
                   ir <= '0';
                   Extop <= '0';
                elsif ZHANG_opcode = "000100" then --And
                   ALUctrl <= "0100";
                   ir <= '0';
                   Extop <= '0';
                elsif ZHANG_opcode = "000101" then --Nor
                   ALUctrl <= "0101";
                   ir <= '0';
                   Extop <= '0';
                elsif ZHANG_opcode = "000110" then --shift left
                   ALUctrl <= "0110";
                   ir <= '0';
                   Extop <= '0';
                elsif ZHANG_opcode = "000111" then --shift right
                   ALUctrl <= "0111";
                   ir <= '0';
                   Extop <= '0';
                elsif ZHANG_opcode = "001000" then --shift right arith
                   ALUctrl <= "1000";
                   ir <= '0';
                   Extop <= '0';
                elsif ZHANG_opcode = "001001" then -- or
                   ALUctrl <= "1101";
                   ir <= '0';
                   Extop <= '0';
                -- 1 = I type, I TYPE SECTION
                elsif ZHANG_opcode = "100000" then --addi
                   ALUctrl <= "1001";
                   ir <= '1';
                   Extop <= '1';
                elsif ZHANG_opcode = "100001" then --addiu
                   ALUctrl <= "1010";
                   ir <= '1';
                   Extop <= '1';
                elsif ZHANG_opcode = "100010" then --andi
                   ALUctrl <= "1011";
                   ir <= '1';
                   Extop <= '0';
                elsif ZHANG_opcode = "100011" then --ori
                   ALUctrl <= "1100";
```

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



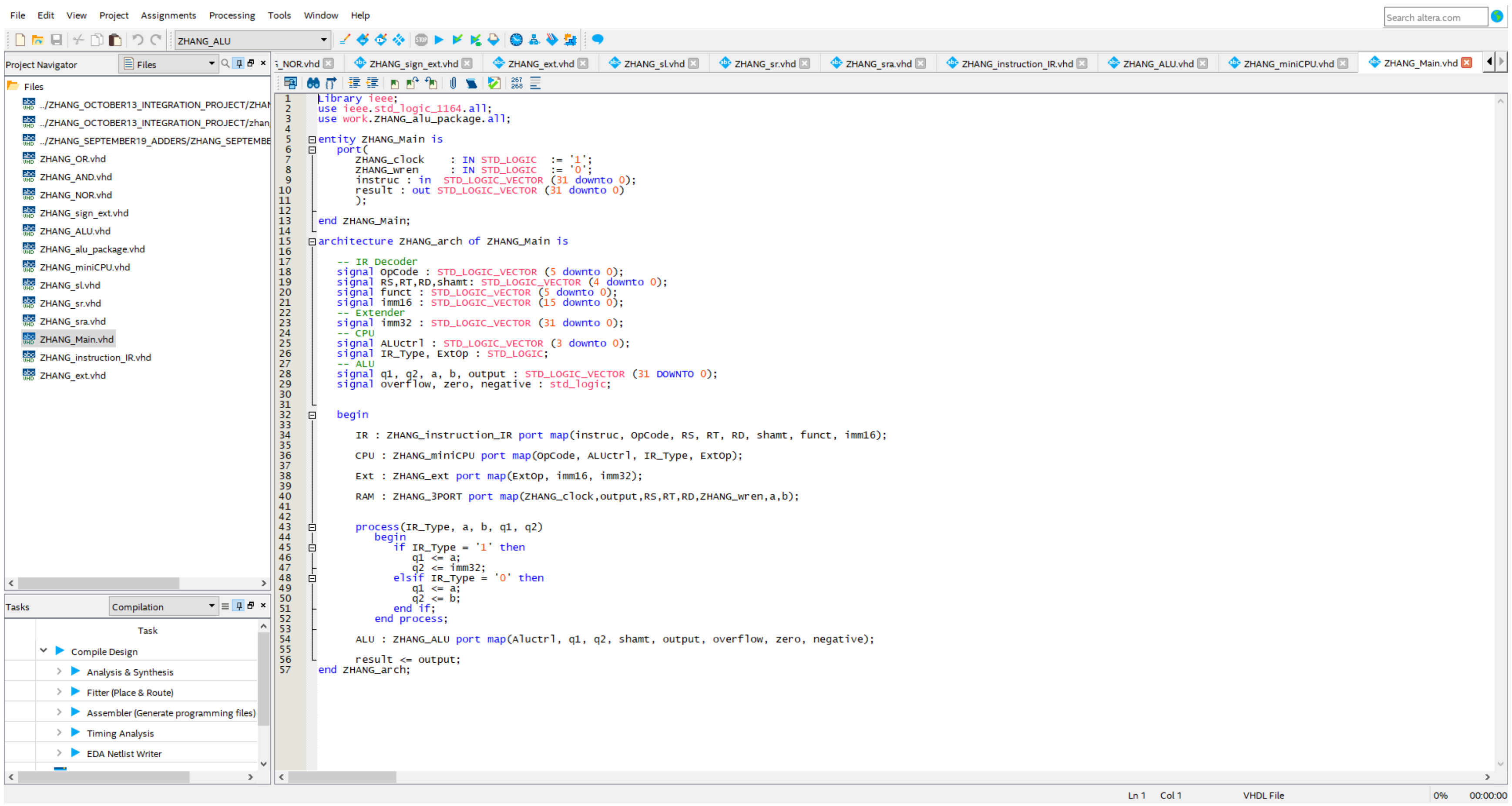
```vhdl
Library ieee;
use ieee.std_logic_1164.all;
use work.ZHANG_alu_package.all;

entity ZHANG_ALU is
   port(
      ZHANG_ALUctrl : in std_logic_vector(3 downto 0);
      ZHANG_q1, ZHANG_q2       : in STD_LOGIC_VECTOR (31 DOWNTO 0);
      shamt : in STD_LOGIC_VECTOR(4 downto 0);
      ZHANG_out : out std_logic_vector (31 downto 0);
      ZHANG_Overflow, ZHANG_Zero, ZHANG_Negative : out std_logic
   );
end ZHANG_ALU;

architecture ZHANG_arch of ZHANG_ALU is

   signal cout, op, ov: std_logic;
   signal zhang_t1, zhang_t2, zhang_t3, zhang_t4,zhang_t5,zhang_t6,zhang_t7: std_logic_vector(31 downto 0);

   begin
      process(ZHANG_ALUctrl)
         begin
            if ZHANG_ALUctrl = "0000" then -- add
               op <= '0';
            elsif ZHANG_ALUctrl = "0001" then -- sub
               op <= '1';
            elsif ZHANG_ALUctrl = "0010" then -- addu
               op <= '0';
            elsif ZHANG_ALUctrl = "0011" then -- subu
               op <= '1';
            elsif ZHANG_ALUctrl = "1001" then -- addi
               op <= '0';
            elsif ZHANG_ALUctrl = "1010" then -- addi
               op <= '0';
            end if;
      end process;

   T1: ZHANG_SEPTEMBER19_N_Bit_AddSub_Flag port map(ZHANG_q1, ZHANG_q2, op, zhang_t1, cout, ov, ZHANG_Zero, ZHANG_Negative); -- add, addu, addiu, sub, subu
   T2: ZHANG_AND port map(ZHANG_q1, ZHANG_q2, zhang_t2); --and, andi
   T3: ZHANG_OR port map(ZHANG_q1, ZHANG_q2, zhang_t3); --ori
   T4: ZHANG_NOR port map(ZHANG_q1, ZHANG_q2, zhang_t4); --nor
   T5: ZHANG_sl port map(ZHANG_q2, shamt, zhang_t5); --sl
   T6: ZHANG_sr port map(ZHANG_q2, shamt, zhang_t6); --sr
   T7: ZHANG_sra port map(ZHANG_q2, shamt, zhang_t7); -- sra


   process(ZHANG_ALUctrl, zhang_t1, zhang_t2, zhang_t3, zhang_t4, zhang_t5, zhang_t6, zhang_t7, ov) --Addu, Addiu, Subu
      begin
         case ZHANG_ALUctrl is
            when "0000" => -- add
               ZHANG_out <= zhang_t1;
               ZHANG_Overflow <= ov;

            when "0001" => -- sub
               ZHANG_out <= zhang_t1;
               ZHANG_Overflow <= ov;

            when "0010" => -- addu
               ZHANG_out <= zhang_t1;
               ZHANG_Overflow <= 'X';

            when "0011" => -- subu
               ZHANG_out <= zhang_t1;
               ZHANG_Overflow <= 'X';
            when "1001" => -- addi
               ZHANG_out <= ZHANG_t1;
```



```
    T2: ZHANG_AND port map(ZHANG_q1, ZHANG_q2, zhang_t2); --and, andi
    T3: ZHANG_OR port map(ZHANG_q1, ZHANG_q2, zhang_t3); --ori
    T4: ZHANG_NOR port map(ZHANG_q1, ZHANG_q2, zhang_t4); --nor
    T5: ZHANG_sl port map(ZHANG_q2, shamt, zhang_t5); --sl
    T6: ZHANG_sr port map(ZHANG_q2, shamt, zhang_t6); --sr
    T7: ZHANG_sra port map(ZHANG_q2, shamt, zhang_t7); -- sra


    process(ZHANG_ALUctrl, zhang_t1, zhang_t2, zhang_t3, zhang_t4, zhang_t5, zhang_t6, zhang_t7, ov) --Addu, Addiu, Subu
       begin
          case ZHANG_ALUctrl is
             when "0000" => -- add
                ZHANG_out <= zhang_t1;
                ZHANG_Overflow <= ov;

             when "0001" => -- sub
                ZHANG_out <= zhang_t1;
                ZHANG_Overflow <= ov;

             when "0010" => -- addu
                ZHANG_out <= zhang_t1;
                ZHANG_Overflow <= 'X';

             when "0011" => -- subu
                ZHANG_out <= zhang_t1;
                ZHANG_Overflow <= 'X';
             when "1001" => -- addi
                ZHANG_out <= ZHANG_t1;
                ZHANG_Overflow <= ov;

             when "1010" => -- addiu
                ZHANG_out <= ZHANG_t1;
                ZHANG_Overflow <= 'X';

             when "0100" => -- and
                ZHANG_out <= ZHANG_t2;

             when "1011" => -- andi
                ZHANG_out <= ZHANG_t2;

             when "1100" => -- ori
                ZHANG_out <= zhang_t3;

             when "1101" => -- or
                ZHANG_out <= zhang_t3;

             when "0101" => -- nor
                ZHANG_out <= zhang_t4;

             when "0110" => -- sl
                ZHANG_out <= zhang_t5;

             when "0111" => -- sr
                ZHANG_out <= zhang_t6;

             when "1000" => -- sra
                ZHANG_out <= zhang_t7;

             when others =>
                ZHANG_out <= (others => 'X');

          end case;
    end process;

end ZHANG_arch;
```

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



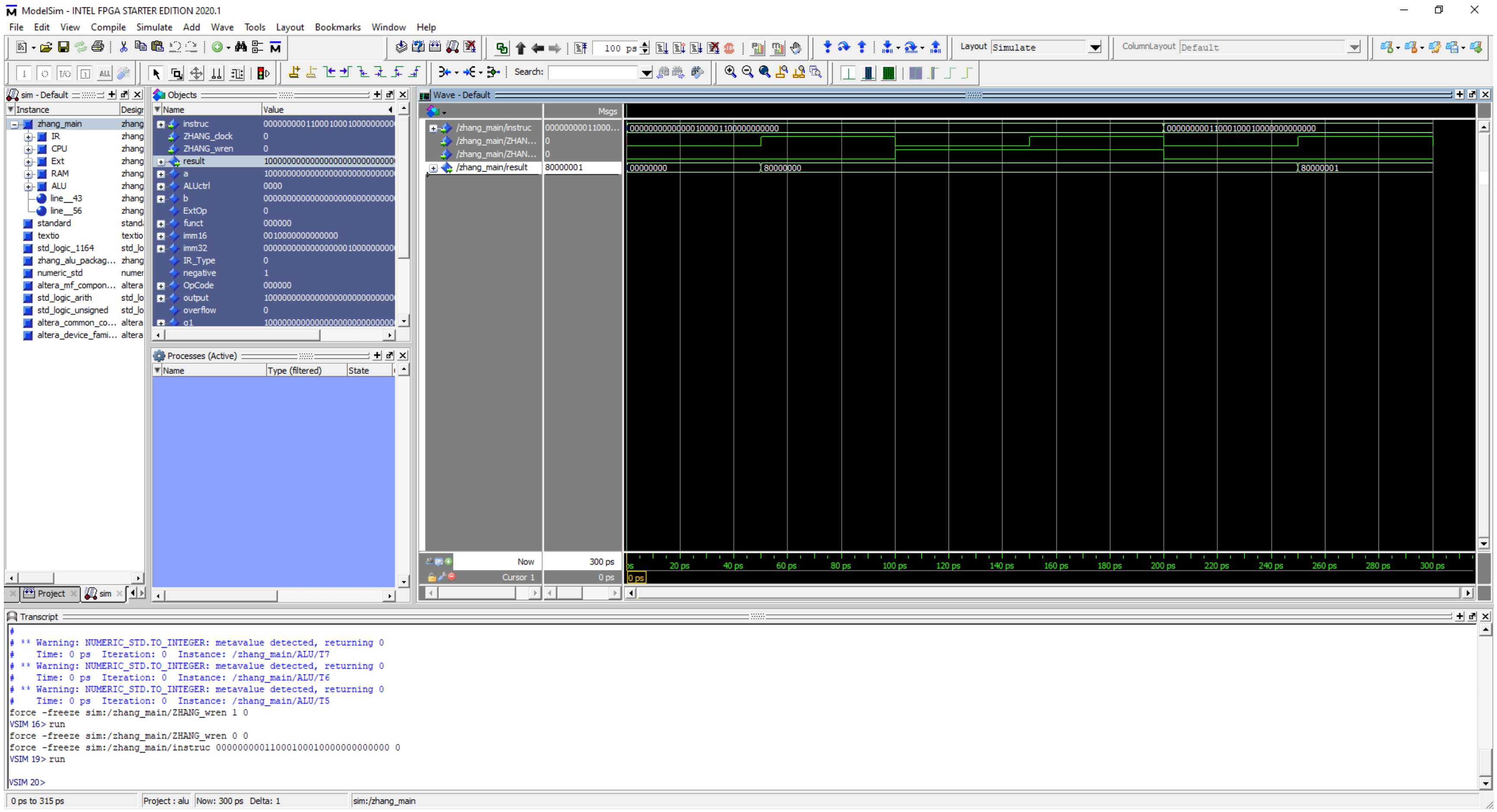
```
Library ieee;
use ieee.std_logic_1164.all;
use work.ZHANG_alu_package.all;

entity ZHANG_Main is
   port(
      ZHANG_clock     : IN STD_LOGIC   := '1';
      ZHANG_wren      : IN STD_LOGIC   := '0';
      instruc : in  STD_LOGIC_VECTOR (31 downto 0);
      result : out STD_LOGIC_VECTOR (31 downto 0)
      );

end ZHANG_Main;

architecture ZHANG_arch of ZHANG_Main is

   -- IR Decoder
   signal OpCode : STD_LOGIC_VECTOR (5 downto 0);
   signal RS,RT,RD,shamt: STD_LOGIC_VECTOR (4 downto 0);
   signal funct : STD_LOGIC_VECTOR (5 downto 0);
   signal imm16 : STD_LOGIC_VECTOR (15 downto 0);
   -- Extender
   signal imm32 : STD_LOGIC_VECTOR (31 downto 0);
   -- CPU
   signal ALUctrl : STD_LOGIC_VECTOR (3 downto 0);
   signal IR_Type, ExtOp : STD_LOGIC;
   -- ALU
   signal q1, q2, a, b, output : STD_LOGIC_VECTOR (31 DOWNTO 0);
   signal overflow, zero, negative : std_logic;


   begin

      IR : ZHANG_instruction_IR port map(instruc, OpCode, RS, RT, RD, shamt, funct, imm16);

      CPU : ZHANG_miniCPU port map(OpCode, ALUctrl, IR_Type, ExtOp);

      Ext : ZHANG_ext port map(ExtOp, imm16, imm32);

      RAM : ZHANG_3PORT port map(ZHANG_clock,output,RS,RT,RD,ZHANG_wren,a,b);


      process(IR_Type, a, b, q1, q2)
         begin
            if IR_Type = '1' then
               q1 <= a;
               q2 <= imm32;
            elsif IR_Type = '0' then
               q1 <= a;
               q2 <= b;
            end if;
         end process;

      ALU : ZHANG_ALU port map(Aluctrl, q1, q2, shamt, output, overflow, zero, negative);

      result <= output;
end ZHANG_arch;
```

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



Name	Value
instruc	0000000001100010001000000000
ZHANG_clock	0
ZHANG_wren	0
result	1000000000000000000000000000
a	1000000000000000000000000000
ALUctrl	0000
b	0000000000000000000000000000
ExtOp	0
funct	000000
imm16	0010000000000000
imm32	0000000000000000001000000000
IR_Type	0
negative	1
OpCode	000000
output	1000000000000000000000000000
overflow	0
q1	1000000000000000000000000000

	Msgs
/zhang_main/instruc	00000000011000...
/zhang_main/ZHAN...	0
/zhang_main/ZHAN...	0
/zhang_main/result	80000001

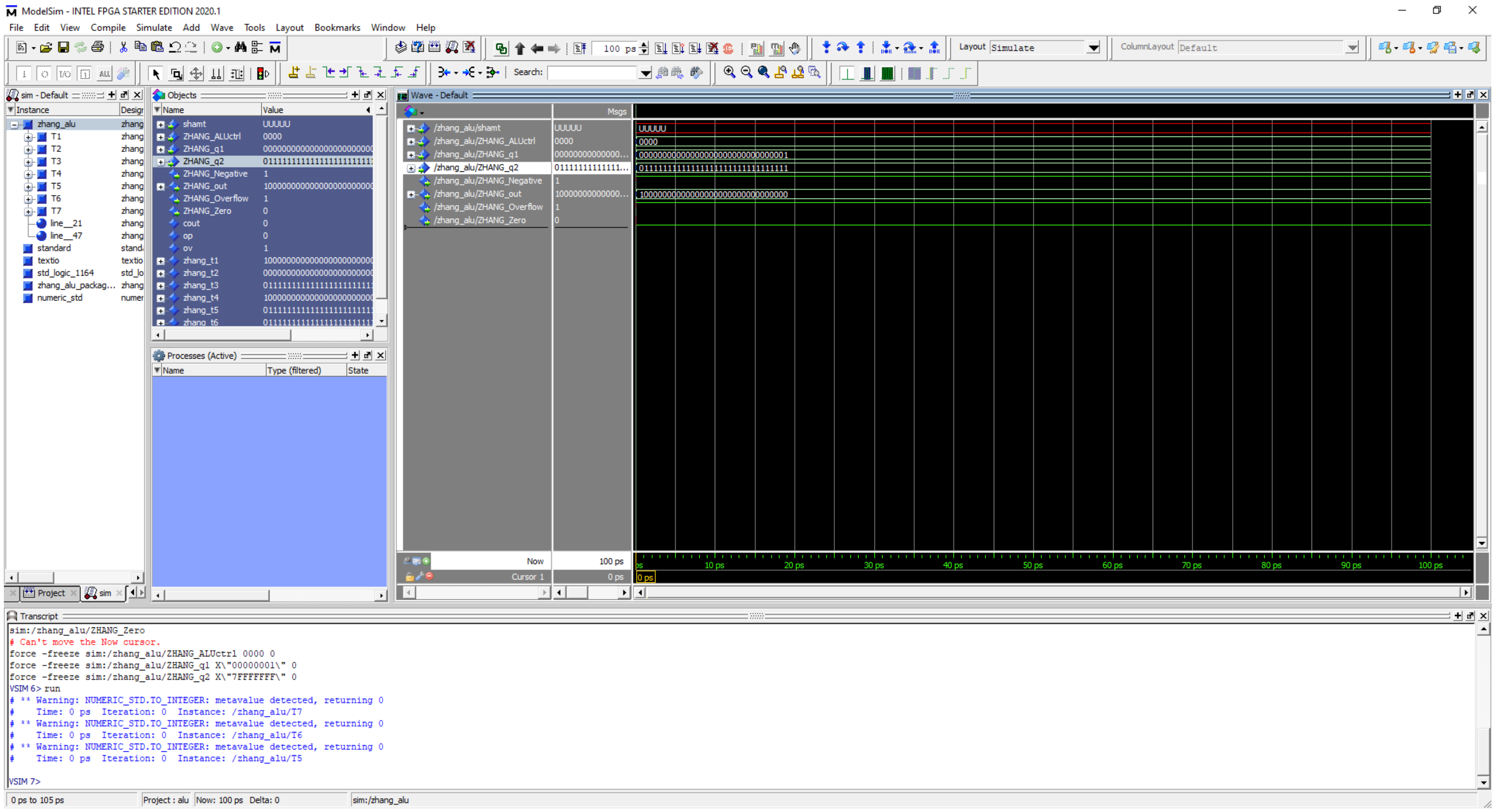
```
#
# ** Warning: NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0
#    Time: 0 ps  Iteration: 0  Instance: /zhang_main/ALU/T7
# ** Warning: NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0
#    Time: 0 ps  Iteration: 0  Instance: /zhang_main/ALU/T6
# ** Warning: NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0
#    Time: 0 ps  Iteration: 0  Instance: /zhang_main/ALU/T5
force -freeze sim:/zhang_main/ZHANG_wren 1 0
VSIM 16> run
force -freeze sim:/zhang_main/ZHANG_wren 0 0
force -freeze sim:/zhang_main/instruc 00000000011000100010000000000000 0
VSIM 19> run

VSIM 20>
```

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 = “0001100000000000”

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



ModelSim - INTEL FPGA STARTER EDITION 2020.1

Objects

```
Name              Value
shamt             UUUUU
ZHANG_ALUctrl     0000
ZHANG_q1          0000000000000000000000000
ZHANG_q2          0111111111111111111111111
ZHANG_Negative    1
ZHANG_out         1000000000000000000000000
ZHANG_Overflow    1
ZHANG_Zero        0
cout              0
op                0
ov                1
zhang_t1          1000000000000000000000000
zhang_t2          0000000000000000000000000
zhang_t3          0111111111111111111111111
zhang_t4          1000000000000000000000000
zhang_t5          0111111111111111111111111
zhang_t6          0111111111111111111111111
```

Wave - Default

```
/zhang_alu/shamt              UUUUU              UUUUU
/zhang_alu/ZHANG_ALUctrl      0000               0000
/zhang_alu/ZHANG_q1           00000000000000...  00000000000000000000000000000001
/zhang_alu/ZHANG_q2           01111111111111...  01111111111111111111111111111111
/zhang_alu/ZHANG_Negative     1
/zhang_alu/ZHANG_out          10000000000000...  10000000000000000000000000000000
/zhang_alu/ZHANG_Overflow     1
/zhang_alu/ZHANG_Zero         0
Now      100 ps
Cursor 1   0 ps
```

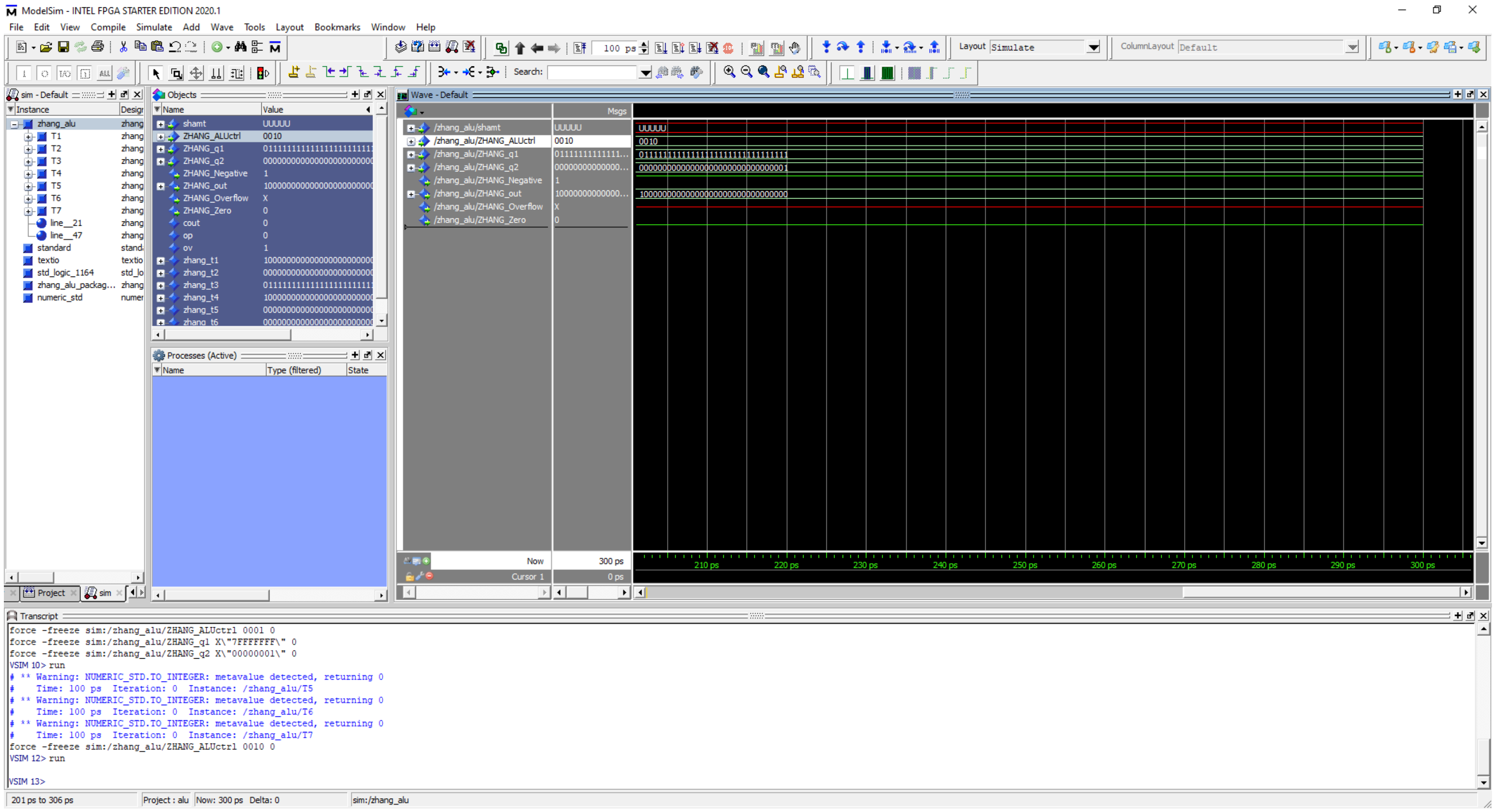
Transcript

```
sim:/zhang_alu/ZHANG_Zero
# Can't move the Now cursor.
force -freeze sim:/zhang_alu/ZHANG_ALUctrl 0000 0
force -freeze sim:/zhang_alu/ZHANG_q1 X\"00000001\" 0
force -freeze sim:/zhang_alu/ZHANG_q2 X\"7FFFFFFF\" 0
VSIM 6> run
# ** Warning: NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0
#    Time: 0 ps  Iteration: 0  Instance: /zhang_alu/T7
# ** Warning: NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0
#    Time: 0 ps  Iteration: 0  Instance: /zhang_alu/T6
# ** Warning: NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0
#    Time: 0 ps  Iteration: 0  Instance: /zhang_alu/T5

VSIM 7>
```

```
0 ps to 105 ps    Project : alu    Now: 100 ps  Delta: 0    sim:/zhang_alu
```

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

Addu



ModelSim - INTEL FPGA STARTER EDITION 2020.1

Name	Value
shamt	UUUUU
ZHANG_ALUctrl	0010
ZHANG_q1	0111111111111111111111
ZHANG_q2	0000000000000000000000
ZHANG_Negative	1
ZHANG_out	1000000000000000000000
ZHANG_Overflow	X
ZHANG_Zero	0
cout	0
op	0
ov	1
zhang_t1	1000000000000000000000
zhang_t2	0000000000000000000000
zhang_t3	0111111111111111111111
zhang_t4	1000000000000000000000
zhang_t5	0000000000000000000000
zhang_t6	0000000000000000000000

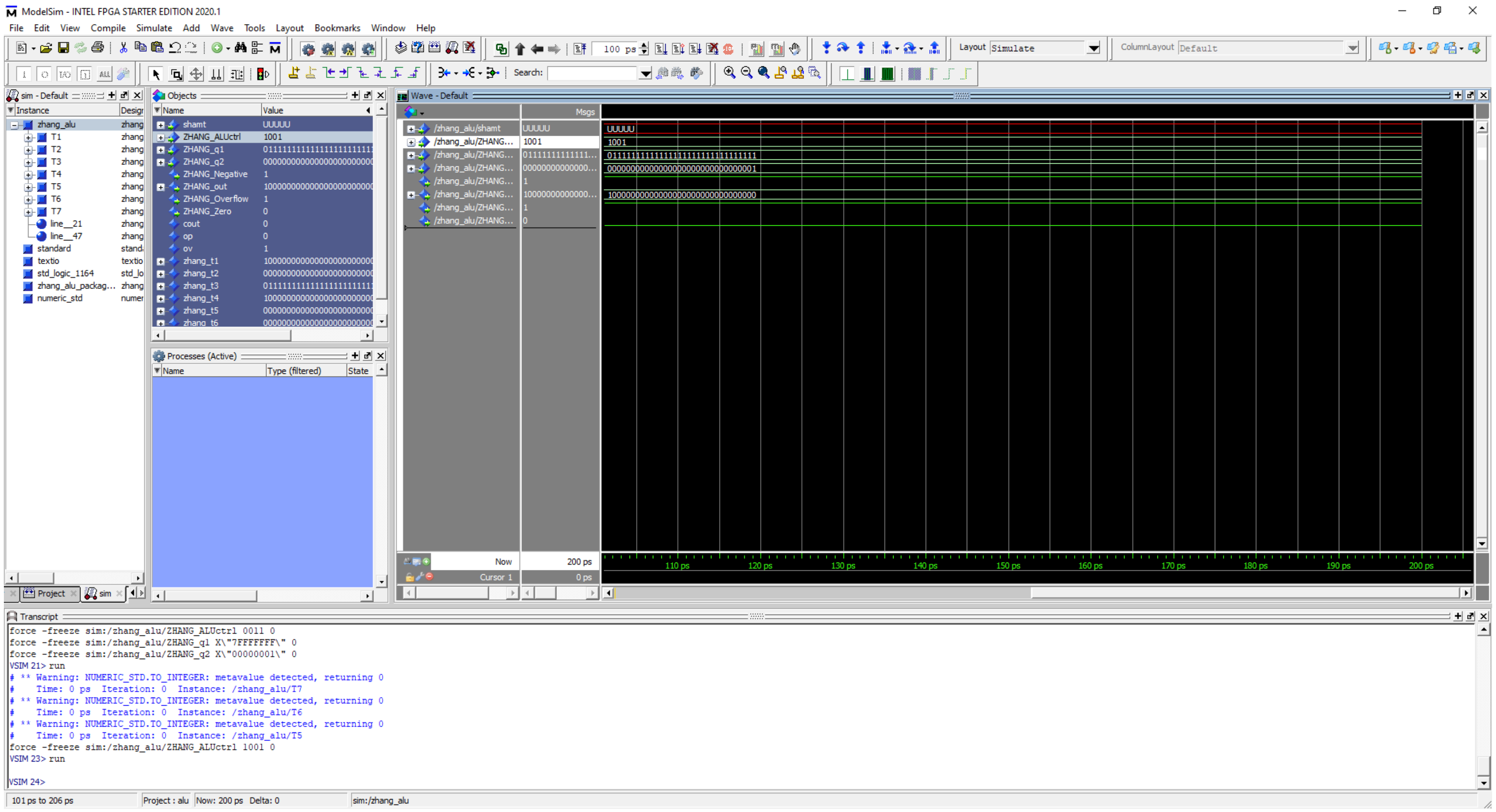
	Msgs	
/zhang_alu/shamt	UUUUU	UUUUU
/zhang_alu/ZHANG_ALUctrl	0010	0010
/zhang_alu/ZHANG_q1	01111111111111...	01111111111111111111111111111111
/zhang_alu/ZHANG_q2	00000000000000...	00000000000000000000000000000001
/zhang_alu/ZHANG_Negative	1	
/zhang_alu/ZHANG_out	10000000000000...	10000000000000000000000000000000
/zhang_alu/ZHANG_Overflow	X	
/zhang_alu/ZHANG_Zero	0	

```
force -freeze sim:/zhang_alu/ZHANG_ALUctrl 0001 0
force -freeze sim:/zhang_alu/ZHANG_q1 X\"7FFFFFFF\" 0
force -freeze sim:/zhang_alu/ZHANG_q2 X\"00000001\" 0
VSIM 10> run
# ** Warning: NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0
#    Time: 100 ps  Iteration: 0  Instance: /zhang_alu/T5
# ** Warning: NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0
#    Time: 100 ps  Iteration: 0  Instance: /zhang_alu/T6
# ** Warning: NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0
#    Time: 100 ps  Iteration: 0  Instance: /zhang_alu/T7
force -freeze sim:/zhang_alu/ZHANG_ALUctrl 0010 0
VSIM 12> run

VSIM 13>
```

201 ps to 306 ps   Project : alu   Now: 300 ps Delta: 0   sim:/zhang_alu

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

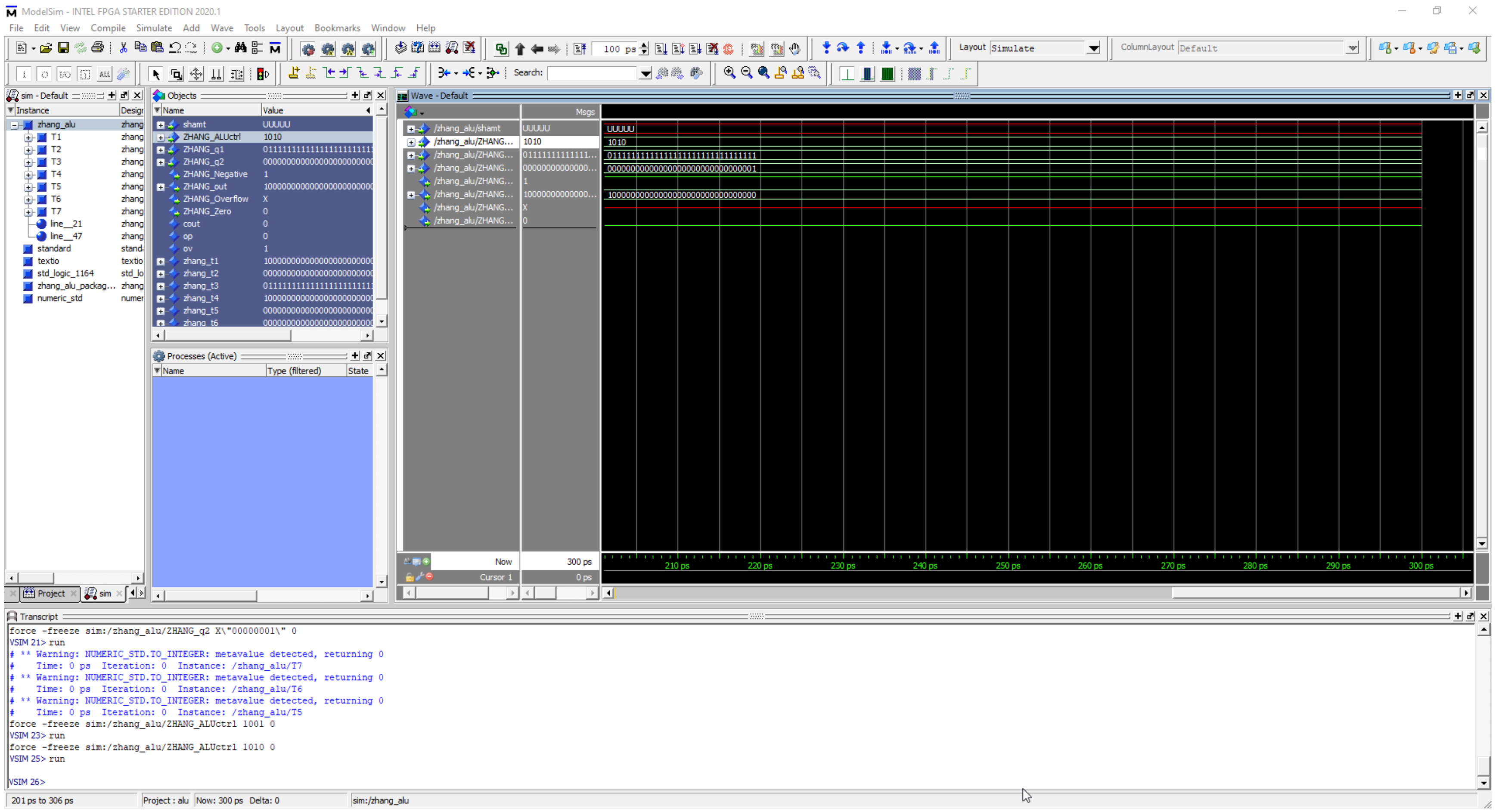


Name	Value
shamt	UUUUU
ZHANG_ALUctrl	1001
ZHANG_q1	0111111111111111111111
ZHANG_q2	0000000000000000000000
ZHANG_Negative	1
ZHANG_out	1000000000000000000000
ZHANG_Overflow	1
ZHANG_Zero	0
cout	0
op	0
ov	1
zhang_t1	1000000000000000000000
zhang_t2	0000000000000000000000
zhang_t3	0111111111111111111111
zhang_t4	1000000000000000000000
zhang_t5	0000000000000000000000
zhang_t6	0000000000000000000000

```
force -freeze sim:/zhang_alu/ZHANG_ALUctrl 0011 0
force -freeze sim:/zhang_alu/ZHANG_q1 X\"7FFFFFFF\" 0
force -freeze sim:/zhang_alu/ZHANG_q2 X\"00000001\" 0
VSIM 21> run
# ** Warning: NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0
#    Time: 0 ps  Iteration: 0  Instance: /zhang_alu/T7
# ** Warning: NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0
#    Time: 0 ps  Iteration: 0  Instance: /zhang_alu/T6
# ** Warning: NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0
#    Time: 0 ps  Iteration: 0  Instance: /zhang_alu/T5
force -freeze sim:/zhang_alu/ZHANG_ALUctrl 1001 0
VSIM 23> run

VSIM 24>
```

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



ModelSim - INTEL FPGA STARTER EDITION 2020.1

Name	Value
shamt	UUUUU
ZHANG_ALUctrl	1010
ZHANG_q1	0111111111111111111111
ZHANG_q2	0000000000000000000000
ZHANG_Negative	1
ZHANG_out	1000000000000000000000
ZHANG_Overflow	X
ZHANG_Zero	0
cout	0
op	0
ov	1
zhang_t1	1000000000000000000000
zhang_t2	0000000000000000000000
zhang_t3	0111111111111111111111
zhang_t4	1000000000000000000000
zhang_t5	0000000000000000000000
zhang_t6	0000000000000000000000

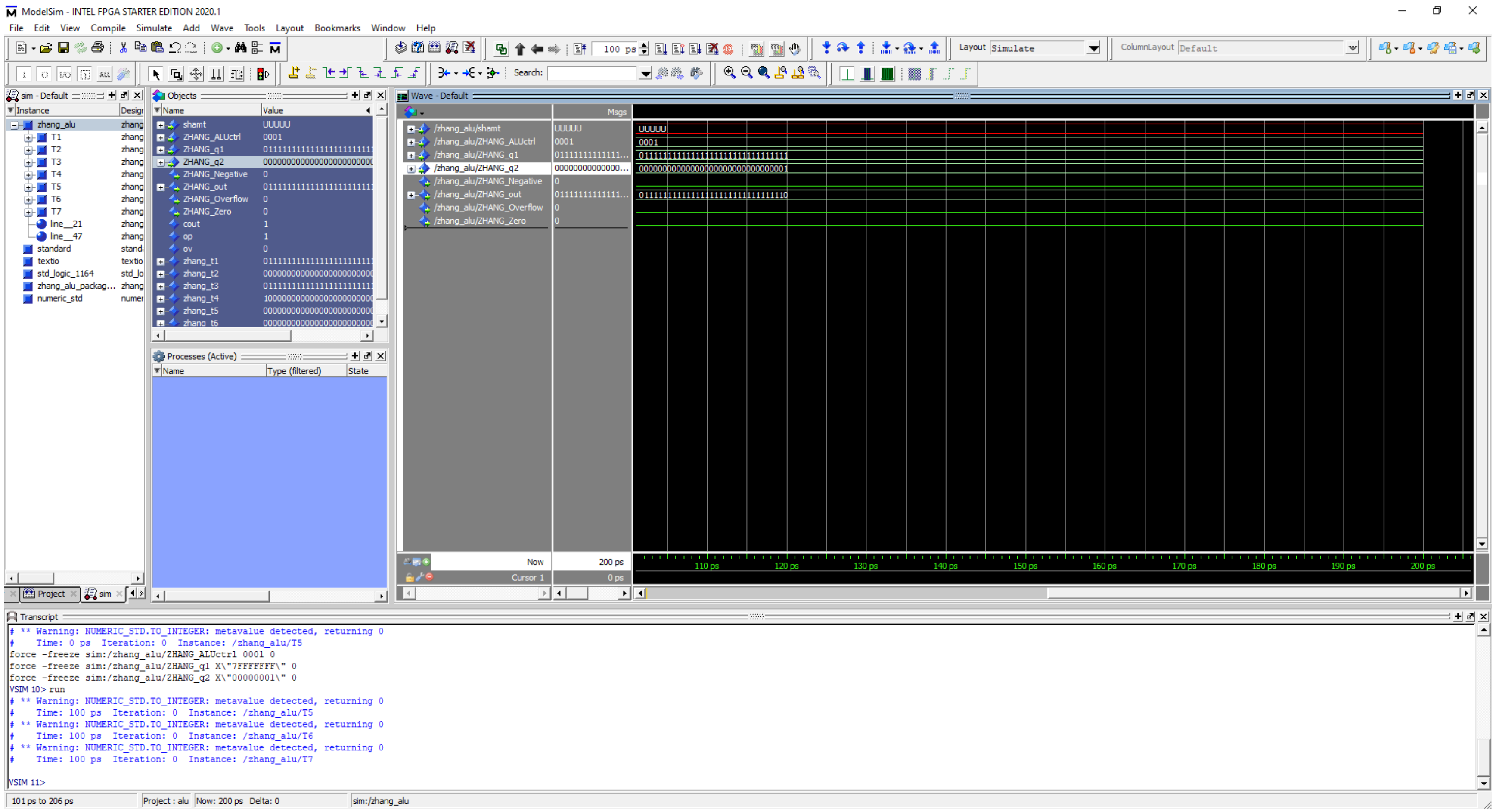
	Msgs	
/zhang_alu/shamt	UUUUU	UUUUU
/zhang_alu/ZHANG...	1010	1010
/zhang_alu/ZHANG...	01111111111111...	01111111111111111111111111111111
/zhang_alu/ZHANG...	00000000000000...	00000000000000000000000000000001
/zhang_alu/ZHANG...	1	
/zhang_alu/ZHANG...	10000000000000...	10000000000000000000000000000000
/zhang_alu/ZHANG...	X	
/zhang_alu/ZHANG...	0	

```
force -freeze sim:/zhang_alu/ZHANG_q2 X\"00000001\" 0
VSIM 21> run
# ** Warning: NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0
#    Time: 0 ps  Iteration: 0  Instance: /zhang_alu/T7
# ** Warning: NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0
#    Time: 0 ps  Iteration: 0  Instance: /zhang_alu/T6
# ** Warning: NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0
#    Time: 0 ps  Iteration: 0  Instance: /zhang_alu/T5
force -freeze sim:/zhang_alu/ZHANG_ALUctrl 1001 0
VSIM 23> run
force -freeze sim:/zhang_alu/ZHANG_ALUctrl 1010 0
VSIM 25> run

VSIM 26>
```

201 ps to 306 ps | Project : alu | Now: 300 ps Delta: 0 | sim:/zhang_alu

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



ModelSim - INTEL FPGA STARTER EDITION 2020.1

Name	Value
shamt	UUUUU
ZHANG_ALUctrl	0001
ZHANG_q1	0111111111111111111111111
ZHANG_q2	0000000000000000000000000
ZHANG_Negative	0
ZHANG_out	0111111111111111111111111
ZHANG_Overflow	0
ZHANG_Zero	0
cout	1
op	1
ov	0
zhang_t1	0111111111111111111111111
zhang_t2	0000000000000000000000000
zhang_t3	0111111111111111111111111
zhang_t4	1000000000000000000000000
zhang_t5	0000000000000000000000000
zhang_t6	0000000000000000000000000

Wave	Msgs	
/zhang_alu/shamt	UUUUU	UUUUU
/zhang_alu/ZHANG_ALUctrl	0001	0001
/zhang_alu/ZHANG_q1	01111111111111...	01111111111111111111111111111111
/zhang_alu/ZHANG_q2	00000000000000...	00000000000000000000000000000001
/zhang_alu/ZHANG_Negative	0	
/zhang_alu/ZHANG_out	01111111111111...	01111111111111111111111111111110
/zhang_alu/ZHANG_Overflow	0	
/zhang_alu/ZHANG_Zero	0	

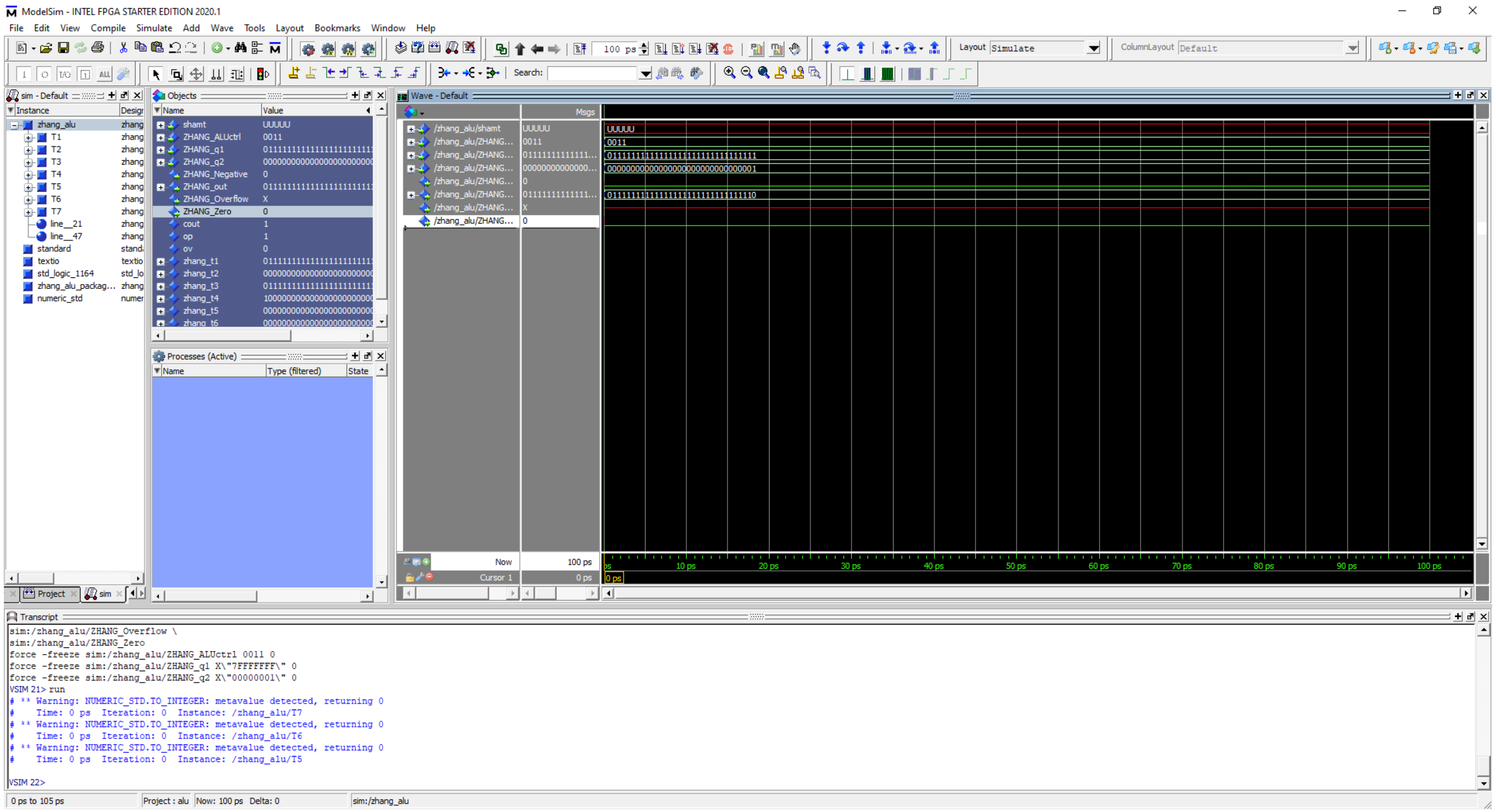
Now 200 ps
Cursor 1 0 ps

```
# ** Warning: NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0
#    Time: 0 ps  Iteration: 0  Instance: /zhang_alu/T5
force -freeze sim:/zhang_alu/ZHANG_ALUctrl 0001 0
force -freeze sim:/zhang_alu/ZHANG_q1 X\"7FFFFFFF\" 0
force -freeze sim:/zhang_alu/ZHANG_q2 X\"00000001\" 0
VSIM 10> run
# ** Warning: NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0
#    Time: 100 ps  Iteration: 0  Instance: /zhang_alu/T5
# ** Warning: NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0
#    Time: 100 ps  Iteration: 0  Instance: /zhang_alu/T6
# ** Warning: NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0
#    Time: 100 ps  Iteration: 0  Instance: /zhang_alu/T7

VSIM 11>
```

101 ps to 206 ps | Project : alu | Now: 200 ps Delta: 0 | sim:/zhang_alu

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

Subu



```
ModelSim - INTEL FPGA STARTER EDITION 2020.1
```

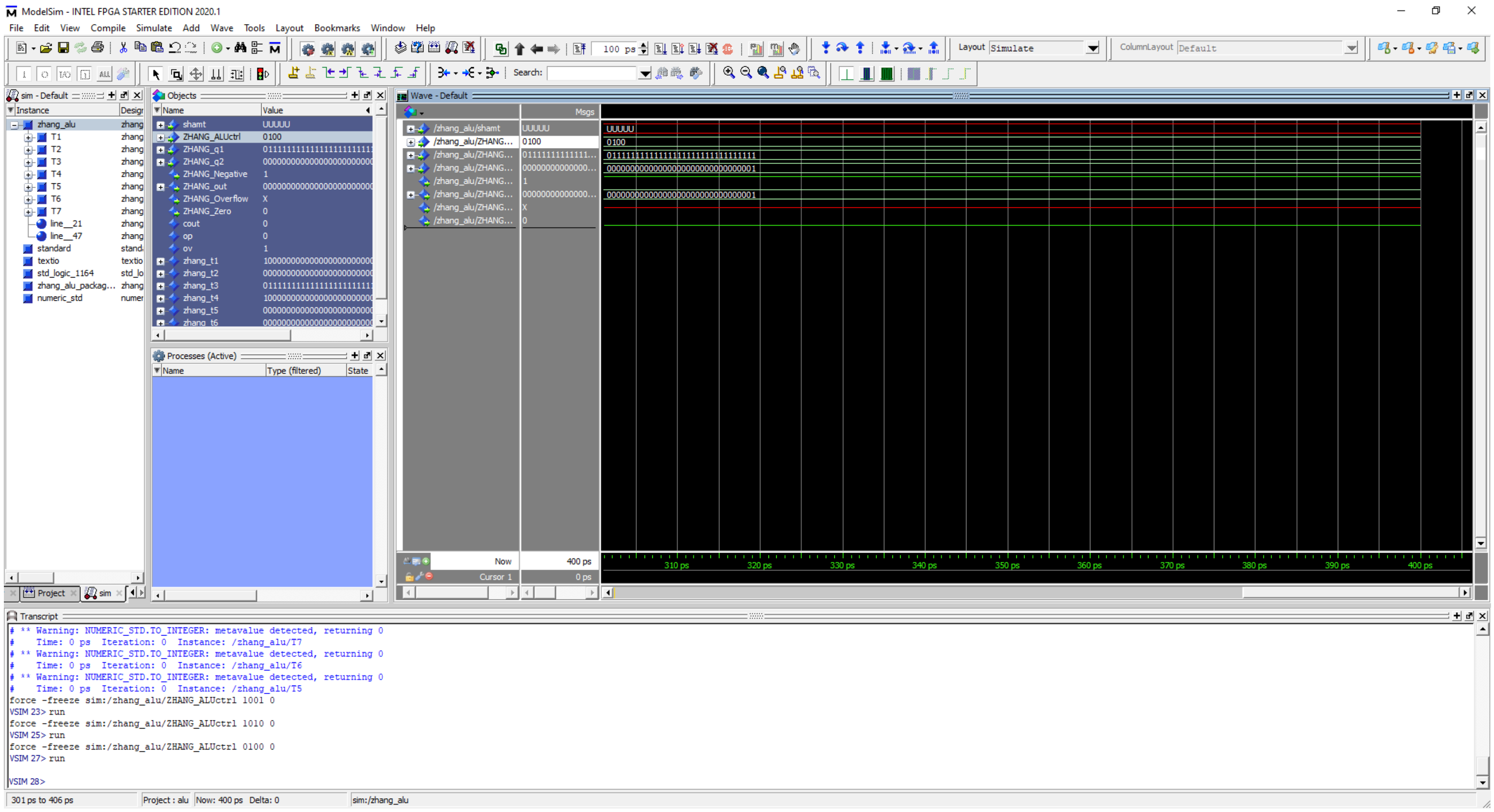
```
Objects
Name              Value
shamt             UUUUU
ZHANG_ALUctrl     0011
ZHANG_q1          0111111111111111111111
ZHANG_q2          0000000000000000000000
ZHANG_Negative    0
ZHANG_out         0111111111111111111111
ZHANG_Overflow    X
ZHANG_Zero        0
cout              1
op                1
ov                0
zhang_t1          0111111111111111111111
zhang_t2          0000000000000000000000
zhang_t3          0111111111111111111111
zhang_t4          1000000000000000000000
zhang_t5          0000000000000000000000
zhang_t6          0000000000000000000000
```

```
Wave - Default                      Msgs
/zhang_alu/shamt       UUUUU              UUUUU
/zhang_alu/ZHANG...    0011               0011
/zhang_alu/ZHANG...    01111111111111...  01111111111111111111111111111111
/zhang_alu/ZHANG...    00000000000000...  00000000000000000000000000000001
/zhang_alu/ZHANG...    0
/zhang_alu/ZHANG...    01111111111111...  01111111111111111111111111111110
/zhang_alu/ZHANG...    X
/zhang_alu/ZHANG...    0
Now 100 ps
Cursor 1 0 ps
```

```
Transcript
sim:/zhang_alu/ZHANG_Overflow \
sim:/zhang_alu/ZHANG_Zero
force -freeze sim:/zhang_alu/ZHANG_ALUctrl 0011 0
force -freeze sim:/zhang_alu/ZHANG_q1 X\"7FFFFFFF\" 0
force -freeze sim:/zhang_alu/ZHANG_q2 X\"00000001\" 0
VSIM 21> run
# ** Warning: NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0
#    Time: 0 ps  Iteration: 0  Instance: /zhang_alu/T7
# ** Warning: NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0
#    Time: 0 ps  Iteration: 0  Instance: /zhang_alu/T6
# ** Warning: NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0
#    Time: 0 ps  Iteration: 0  Instance: /zhang_alu/T5

VSIM 22>

0 ps to 105 ps    Project : alu    Now: 100 ps  Delta: 0    sim:/zhang_alu
```

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

And


ModelSim - INTEL FPGA STARTER EDITION 2020.1

Name	Value
shamt	UUUUU
ZHANG_ALUctrl	0100
ZHANG_q1	0111111111111111111111
ZHANG_q2	0000000000000000000000
ZHANG_Negative	1
ZHANG_out	0000000000000000000000
ZHANG_Overflow	X
ZHANG_Zero	0
cout	0
op	0
ov	1
zhang_t1	1000000000000000000000
zhang_t2	0000000000000000000000
zhang_t3	0111111111111111111111
zhang_t4	1000000000000000000000
zhang_t5	0000000000000000000000
zhang_t6	0000000000000000000000

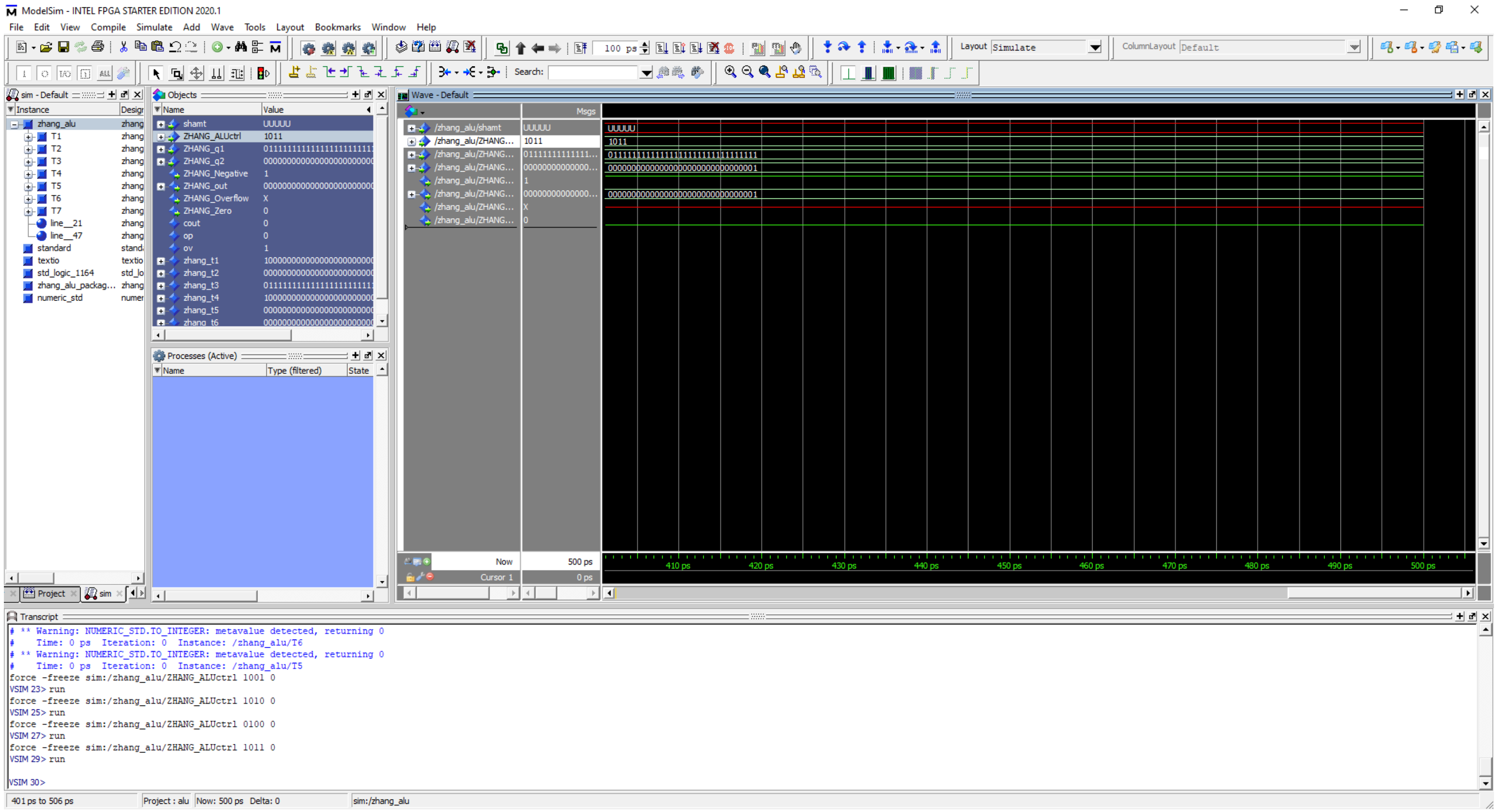
	Msgs	
/zhang_alu/shamt	UUUUU	UUUUU
/zhang_alu/ZHANG...	0100	0100
/zhang_alu/ZHANG...	01111111111111...	01111111111111111111111111111111
/zhang_alu/ZHANG...	00000000000000...	00000000000000000000000000000001
/zhang_alu/ZHANG...	1	
/zhang_alu/ZHANG...	00000000000000...	00000000000000000000000000000001
/zhang_alu/ZHANG...	X	
/zhang_alu/ZHANG...	0	

```
# ** Warning: NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0
#    Time: 0 ps  Iteration: 0  Instance: /zhang_alu/T7
# ** Warning: NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0
#    Time: 0 ps  Iteration: 0  Instance: /zhang_alu/T6
# ** Warning: NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0
#    Time: 0 ps  Iteration: 0  Instance: /zhang_alu/T5
force -freeze sim:/zhang_alu/ZHANG_ALUctrl 1001 0
VSIM 23> run
force -freeze sim:/zhang_alu/ZHANG_ALUctrl 1010 0
VSIM 25> run
force -freeze sim:/zhang_alu/ZHANG_ALUctrl 0100 0
VSIM 27> run

VSIM 28>
```

301 ps to 406 ps | Project : alu | Now: 400 ps Delta: 0 | sim:/zhang_alu

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

Andi


ModelSim - INTEL FPGA STARTER EDITION 2020.1
File Edit View Compile Simulate Add Wave Tools Layout Bookmarks Window Help
Layout Simulate
ColumnLayout Default
sim - Default
Objects
Processes (Active)
Wave - Default
Transcript

Instance	Design
zhang_alu	zhang
T1	zhang
T2	zhang
T3	zhang
T4	zhang
T5	zhang
T6	zhang
T7	zhang
line__21	zhang
line__47	zhang
standard	stand
textio	textio
std_logic_1164	std_lo
zhang_alu_packag...	zhang
numeric_std	numer

Name	Value
shamt	UUUUU
ZHANG_ALUctrl	1011
ZHANG_q1	0111111111111111111111111
ZHANG_q2	0000000000000000000000000
ZHANG_Negative	1
ZHANG_out	0000000000000000000000000
ZHANG_Overflow	X
ZHANG_Zero	0
cout	0
op	0
ov	1
zhang_t1	1000000000000000000000000
zhang_t2	0000000000000000000000000
zhang_t3	0111111111111111111111111
zhang_t4	1000000000000000000000000
zhang_t5	0000000000000000000000000
zhang_t6	0000000000000000000000000

	Msgs	
/zhang_alu/shamt	UUUUU	UUUUU
/zhang_alu/ZHANG...	1011	1011
/zhang_alu/ZHANG...	01111111111111...	01111111111111111111111111111111
/zhang_alu/ZHANG...	00000000000000...	00000000000000000000000000000001
/zhang_alu/ZHANG...	1	
/zhang_alu/ZHANG...	00000000000000...	00000000000000000000000000000001
/zhang_alu/ZHANG...	X	
/zhang_alu/ZHANG...	0	

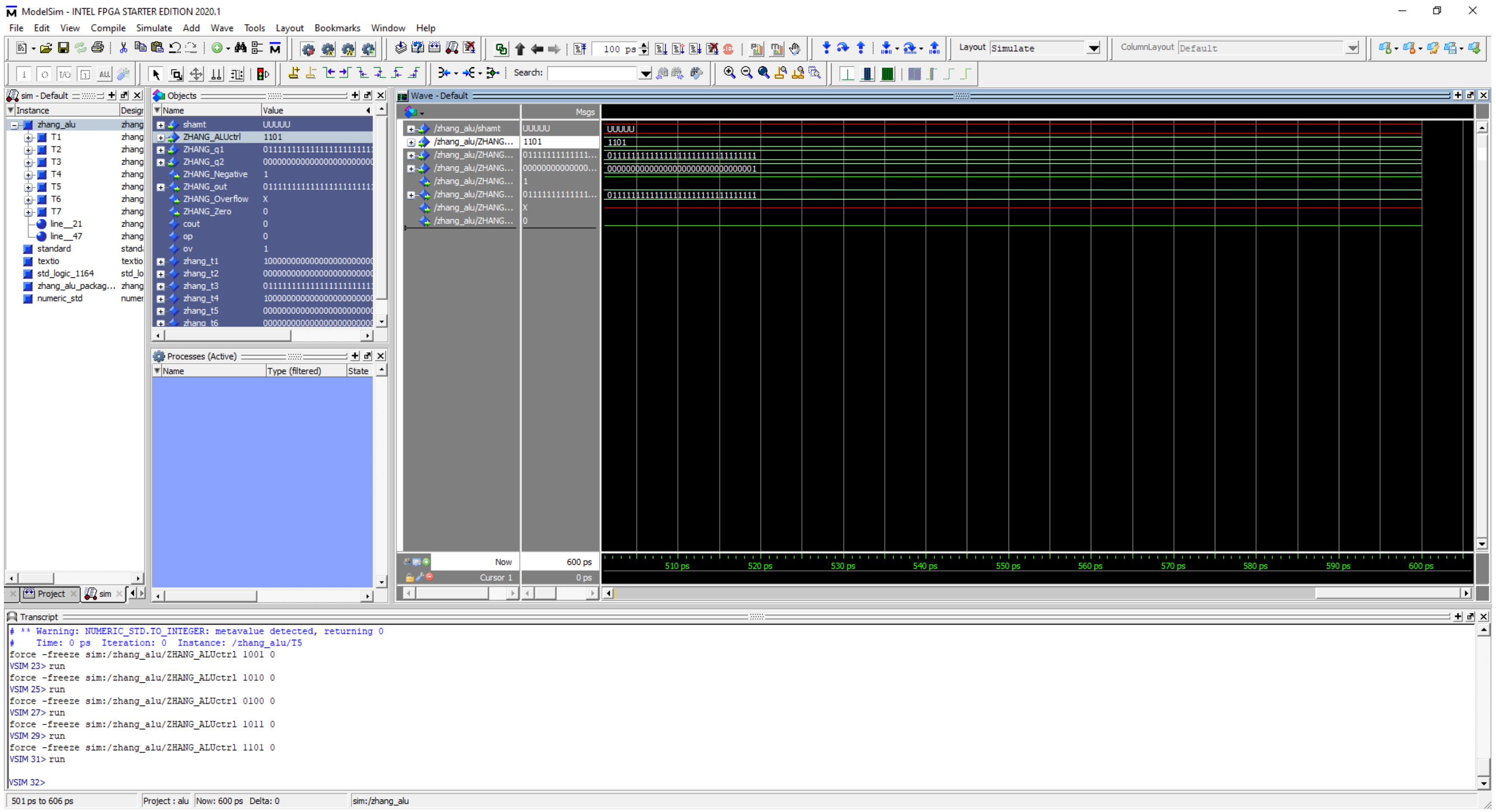
Now 500 ps
Cursor 1 0 ps
410 ps 420 ps 430 ps 440 ps 450 ps 460 ps 470 ps 480 ps 490 ps 500 ps

```
# ** Warning: NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0
#    Time: 0 ps  Iteration: 0  Instance: /zhang_alu/T6
# ** Warning: NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0
#    Time: 0 ps  Iteration: 0  Instance: /zhang_alu/T5
force -freeze sim:/zhang_alu/ZHANG_ALUctrl 1001 0
VSIM 23> run
force -freeze sim:/zhang_alu/ZHANG_ALUctrl 1010 0
VSIM 25> run
force -freeze sim:/zhang_alu/ZHANG_ALUctrl 0100 0
VSIM 27> run
force -freeze sim:/zhang_alu/ZHANG_ALUctrl 1011 0
VSIM 29> run

VSIM 30>
```

401 ps to 506 ps Project : alu Now: 500 ps Delta: 0 sim:/zhang_alu

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

Or



ModelSim - INTEL FPGA STARTER EDITION 2020.1

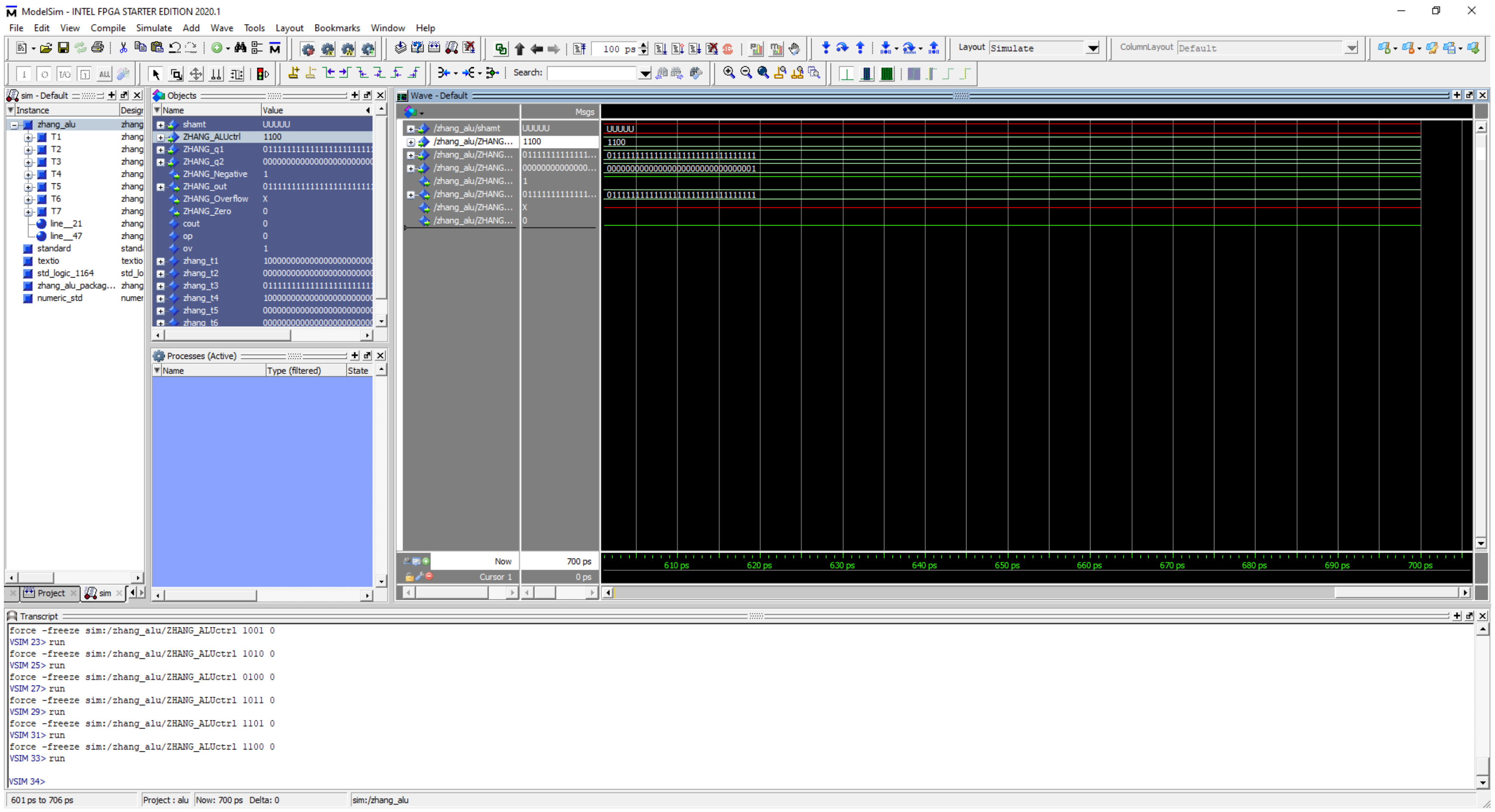
Name	Value
shamt	UUUUU
ZHANG_ALUctrl	1101
ZHANG_q1	0111111111111111111111
ZHANG_q2	0000000000000000000000
ZHANG_Negative	1
ZHANG_out	0111111111111111111111
ZHANG_Overflow	X
ZHANG_Zero	0
cout	0
op	0
ov	1
zhang_t1	1000000000000000000000
zhang_t2	0000000000000000000000
zhang_t3	0111111111111111111111
zhang_t4	1000000000000000000000
zhang_t5	0000000000000000000000
zhang_t6	0000000000000000000000

```
# ** Warning: NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0
#    Time: 0 ps  Iteration: 0  Instance: /zhang_alu/T5
force -freeze sim:/zhang_alu/ZHANG_ALUctrl 1001 0
VSIM 23> run
force -freeze sim:/zhang_alu/ZHANG_ALUctrl 1010 0
VSIM 25> run
force -freeze sim:/zhang_alu/ZHANG_ALUctrl 0100 0
VSIM 27> run
force -freeze sim:/zhang_alu/ZHANG_ALUctrl 1011 0
VSIM 29> run
force -freeze sim:/zhang_alu/ZHANG_ALUctrl 1101 0
VSIM 31> run

VSIM 32>
```

501 ps to 606 ps | Project : alu | Now: 600 ps Delta: 0 | sim:/zhang_alu

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

Ori


ModelSim - INTEL FPGA STARTER EDITION 2020.1

Name	Value
shamt	UUUUU
ZHANG_ALUctrl	1100
ZHANG_q1	0111111111111111111111111
ZHANG_q2	0000000000000000000000000
ZHANG_Negative	1
ZHANG_out	0111111111111111111111111
ZHANG_Overflow	X
ZHANG_Zero	0
cout	0
op	0
ov	1
zhang_t1	1000000000000000000000000
zhang_t2	0000000000000000000000000
zhang_t3	0111111111111111111111111
zhang_t4	1000000000000000000000000
zhang_t5	0000000000000000000000000
zhang_t6	0000000000000000000000000

Wave - Default	Msgs	
/zhang_alu/shamt	UUUUU	UUUUU
/zhang_alu/ZHANG...	1100	1100
/zhang_alu/ZHANG...	01111111111111...	01111111111111111111111111111111
/zhang_alu/ZHANG...	00000000000000...	00000000000000000000000000000001
/zhang_alu/ZHANG...	1	
/zhang_alu/ZHANG...	01111111111111...	01111111111111111111111111111111
/zhang_alu/ZHANG...	X	
/zhang_alu/ZHANG...	0	

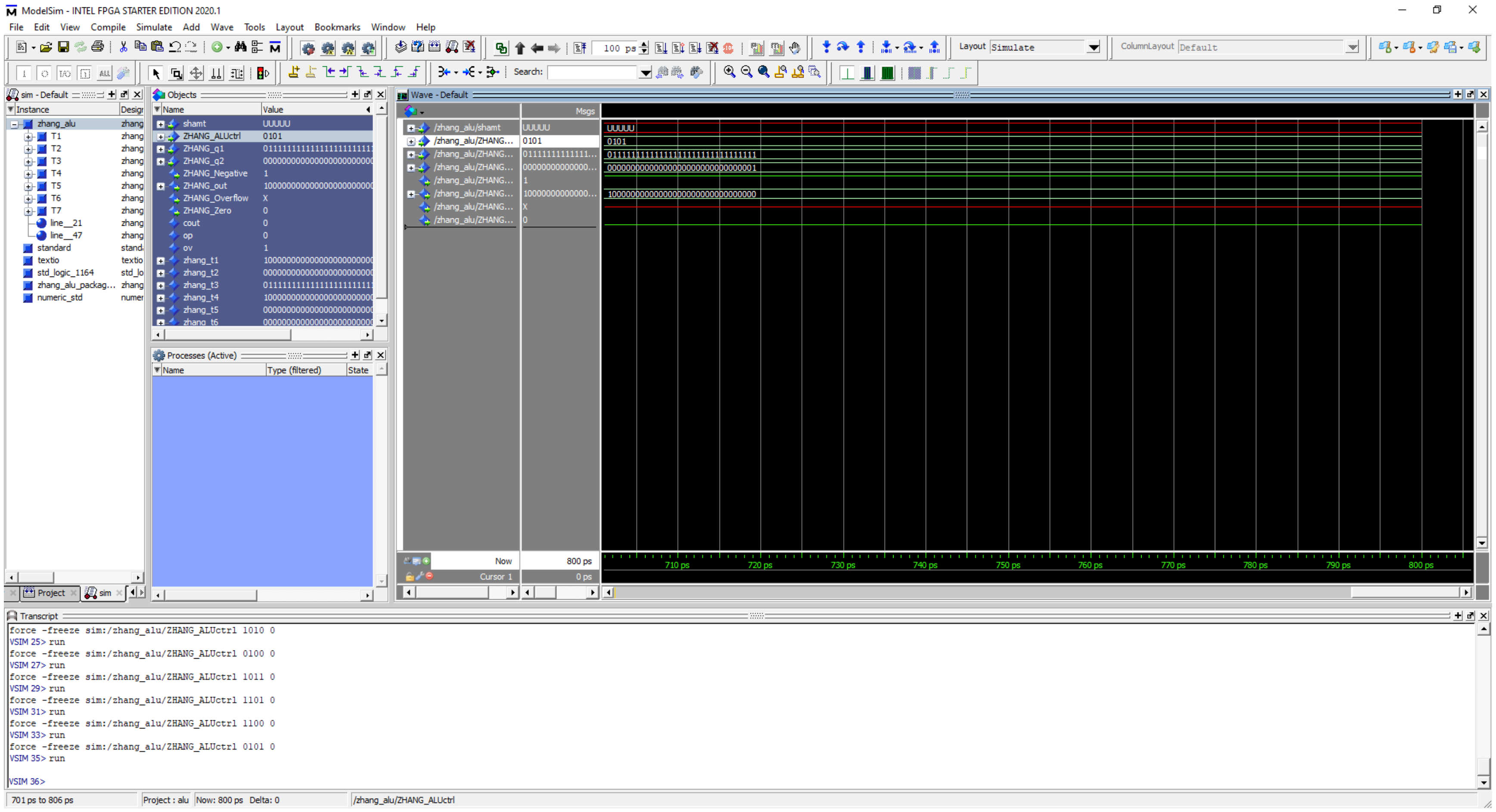
Now 700 ps
Cursor 1 0 ps

```
force -freeze sim:/zhang_alu/ZHANG_ALUctrl 1001 0
VSIM 23> run
force -freeze sim:/zhang_alu/ZHANG_ALUctrl 1010 0
VSIM 25> run
force -freeze sim:/zhang_alu/ZHANG_ALUctrl 0100 0
VSIM 27> run
force -freeze sim:/zhang_alu/ZHANG_ALUctrl 1011 0
VSIM 29> run
force -freeze sim:/zhang_alu/ZHANG_ALUctrl 1101 0
VSIM 31> run
force -freeze sim:/zhang_alu/ZHANG_ALUctrl 1100 0
VSIM 33> run

VSIM 34>
```

601 ps to 706 ps   Project : alu   Now: 700 ps   Delta: 0   sim:/zhang_alu

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

Nor



ModelSim - INTEL FPGA STARTER EDITION 2020.1

Name	Value
shamt	UUUUU
ZHANG_ALUctrl	0101
ZHANG_q1	0111111111111111111111
ZHANG_q2	0000000000000000000000
ZHANG_Negative	1
ZHANG_out	1000000000000000000000
ZHANG_Overflow	X
ZHANG_Zero	0
cout	0
op	0
ov	1
zhang_t1	1000000000000000000000
zhang_t2	0000000000000000000000
zhang_t3	0111111111111111111111
zhang_t4	1000000000000000000000
zhang_t5	0000000000000000000000
zhang_t6	0000000000000000000000

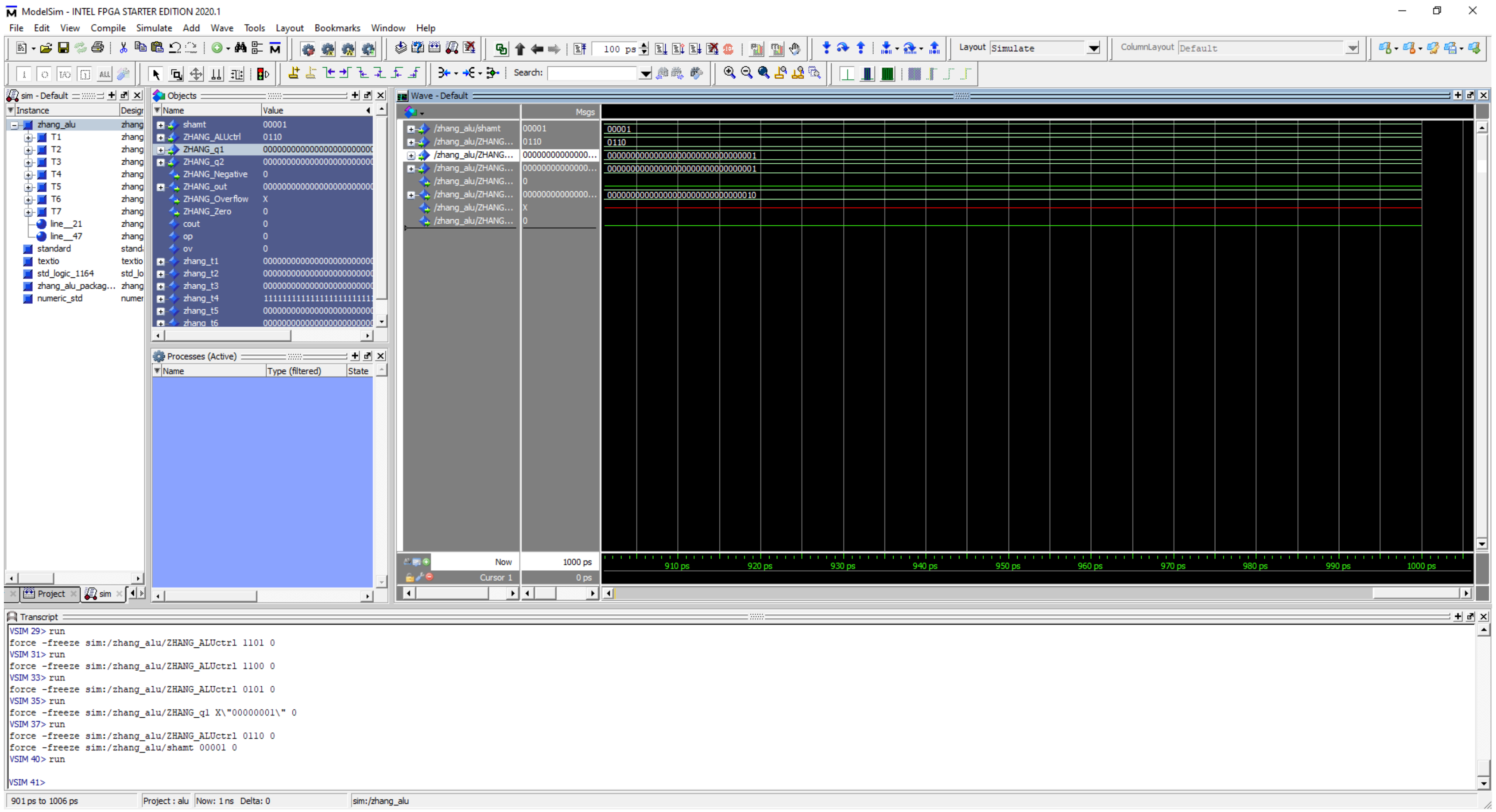
```
/zhang_alu/shamt      UUUUU             UUUUU
/zhang_alu/ZHANG...   0101              0101
/zhang_alu/ZHANG...   01111111111111... 01111111111111111111111111111111
/zhang_alu/ZHANG...   00000000000000... 00000000000000000000000000000001
/zhang_alu/ZHANG...   1
/zhang_alu/ZHANG...   10000000000000... 10000000000000000000000000000000
/zhang_alu/ZHANG...   X
/zhang_alu/ZHANG...   0
Now     800 ps
Cursor 1  0 ps
```

```
force -freeze sim:/zhang_alu/ZHANG_ALUctrl 1010 0
VSIM 25> run
force -freeze sim:/zhang_alu/ZHANG_ALUctrl 0100 0
VSIM 27> run
force -freeze sim:/zhang_alu/ZHANG_ALUctrl 1011 0
VSIM 29> run
force -freeze sim:/zhang_alu/ZHANG_ALUctrl 1101 0
VSIM 31> run
force -freeze sim:/zhang_alu/ZHANG_ALUctrl 1100 0
VSIM 33> run
force -freeze sim:/zhang_alu/ZHANG_ALUctrl 0101 0
VSIM 35> run

VSIM 36>
```

```
701 ps to 806 ps   Project : alu   Now: 800 ps  Delta: 0   /zhang_alu/ZHANG_ALUctrl
```

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

Shift Left



ModelSim - INTEL FPGA STARTER EDITION 2020.1

Name	Value
shamt	00001
ZHANG_ALUctrl	0110
ZHANG_q1	000000000000000000000000
ZHANG_q2	000000000000000000000000
ZHANG_Negative	0
ZHANG_out	000000000000000000000000
ZHANG_Overflow	X
ZHANG_Zero	0
cout	0
op	0
ov	0
zhang_t1	000000000000000000000000
zhang_t2	000000000000000000000000
zhang_t3	000000000000000000000000
zhang_t4	111111111111111111111111
zhang_t5	000000000000000000000000
zhang_t6	000000000000000000000000

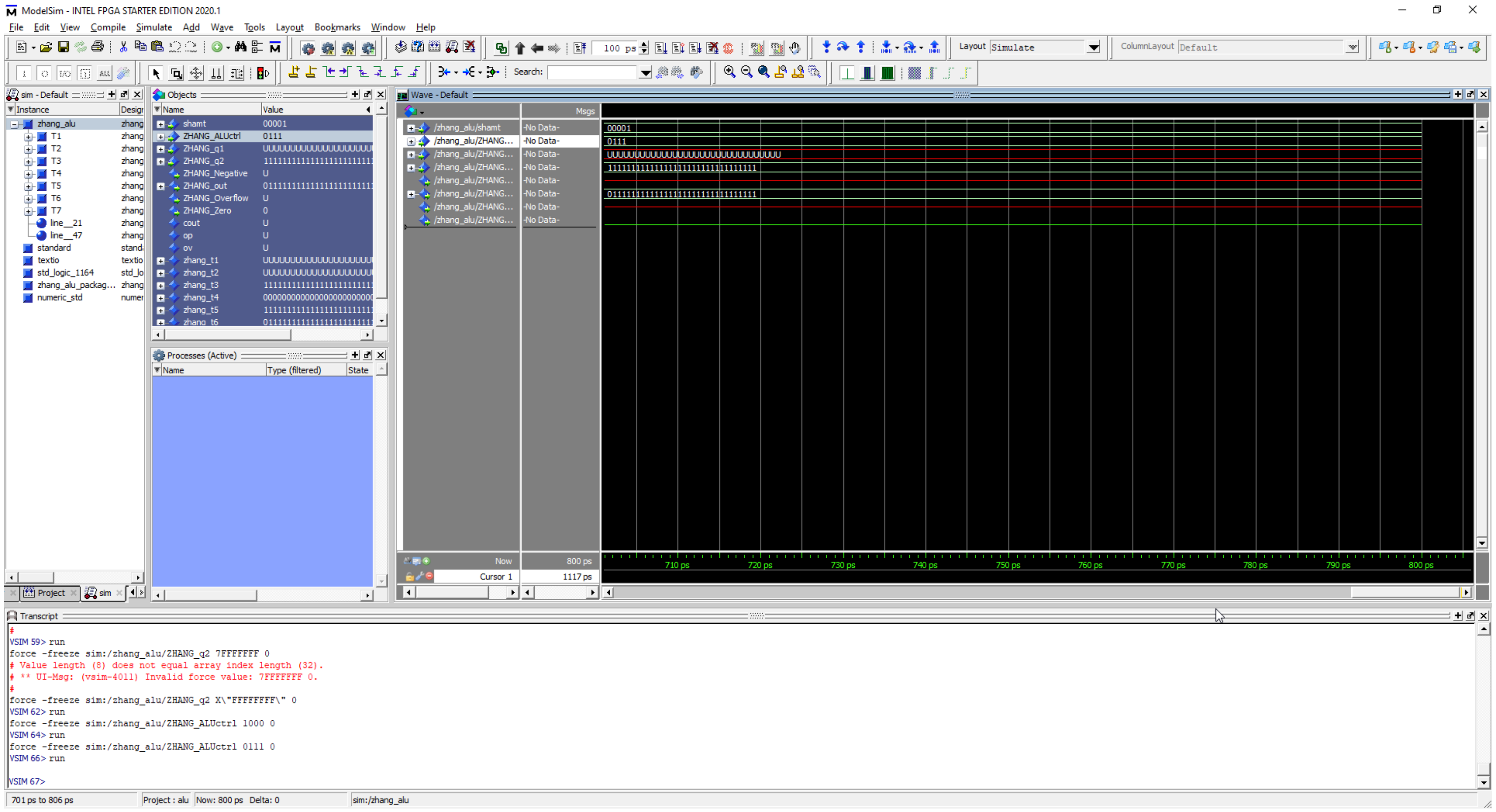
	Msgs	
/zhang_alu/shamt	00001	00001
/zhang_alu/ZHANG...	0110	0110
/zhang_alu/ZHANG...	00000000000000...	00000000000000000000000000000001
/zhang_alu/ZHANG...	00000000000000...	00000000000000000000000000000001
/zhang_alu/ZHANG...	0	
/zhang_alu/ZHANG...	00000000000000...	00000000000000000000000000000010
/zhang_alu/ZHANG...	X	
/zhang_alu/ZHANG...	0	

```
VSIM 29> run
force -freeze sim:/zhang_alu/ZHANG_ALUctrl 1101 0
VSIM 31> run
force -freeze sim:/zhang_alu/ZHANG_ALUctrl 1100 0
VSIM 33> run
force -freeze sim:/zhang_alu/ZHANG_ALUctrl 0101 0
VSIM 35> run
force -freeze sim:/zhang_alu/ZHANG_q1 X\"00000001\" 0
VSIM 37> run
force -freeze sim:/zhang_alu/ZHANG_ALUctrl 0110 0
force -freeze sim:/zhang_alu/shamt 00001 0
VSIM 40> run

VSIM 41>
```

901 ps to 1006 ps | Project : alu | Now: 1 ns Delta: 0 | sim:/zhang_alu

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

Shift Right



```
ModelSim - INTEL FPGA STARTER EDITION 2020.1
```

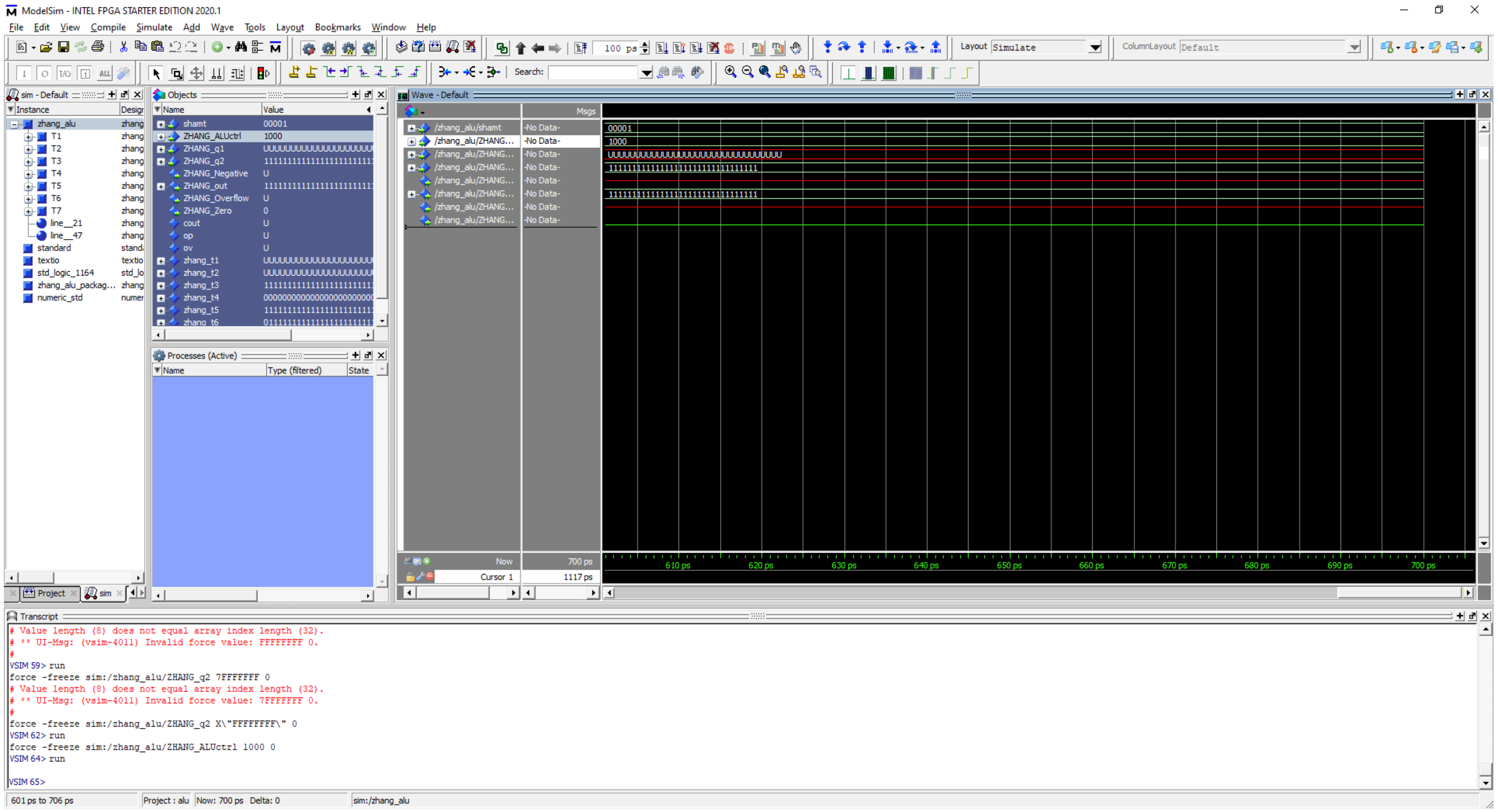
Name	Value
shamt	00001
ZHANG_ALUctrl	0111
ZHANG_q1	UUUUUUUUUUUUUUUUUUUU
ZHANG_q2	11111111111111111111
ZHANG_Negative	U
ZHANG_out	01111111111111111111
ZHANG_Overflow	U
ZHANG_Zero	0
cout	U
op	U
ov	U
zhang_t1	UUUUUUUUUUUUUUUUUUUU
zhang_t2	UUUUUUUUUUUUUUUUUUUU
zhang_t3	11111111111111111111
zhang_t4	00000000000000000000
zhang_t5	11111111111111111111
zhang_t6	01111111111111111111

```
#
VSIM 59> run
force -freeze sim:/zhang_alu/ZHANG_q2 7FFFFFFF 0
# Value length (8) does not equal array index length (32).
# ** UI-Msg: (vsim-4011) Invalid force value: 7FFFFFFF 0.
#
force -freeze sim:/zhang_alu/ZHANG_q2 X\"FFFFFFFF\" 0
VSIM 62> run
force -freeze sim:/zhang_alu/ZHANG_ALUctrl 1000 0
VSIM 64> run
force -freeze sim:/zhang_alu/ZHANG_ALUctrl 0111 0
VSIM 66> run

VSIM 67>
```

```
701 ps to 806 ps    Project : alu    Now: 800 ps  Delta: 0    sim:/zhang_alu
```

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



Name	Value
shamt	00001
ZHANG_ALUctrl	1000
ZHANG_q1	UUUUUUUUUUUUUUUUUUUU
ZHANG_q2	1111111111111111111111
ZHANG_Negative	U
ZHANG_out	1111111111111111111111
ZHANG_Overflow	U
ZHANG_Zero	0
cout	U
op	U
ov	U
zhang_t1	UUUUUUUUUUUUUUUUUUUU
zhang_t2	UUUUUUUUUUUUUUUUUUUU
zhang_t3	1111111111111111111111
zhang_t4	0000000000000000000000
zhang_t5	1111111111111111111111
zhang_t6	0111111111111111111111

```
# Value length (8) does not equal array index length (32).
# ** UI-Msg: (vsim-4011) Invalid force value: FFFFFFFF 0.
#
VSIM 59> run
force -freeze sim:/zhang_alu/ZHANG_q2 7FFFFFFF 0
# Value length (8) does not equal array index length (32).
# ** UI-Msg: (vsim-4011) Invalid force value: 7FFFFFFF 0.
#
force -freeze sim:/zhang_alu/ZHANG_q2 X\"FFFFFFFF\" 0
VSIM 62> run
force -freeze sim:/zhang_alu/ZHANG_ALUctrl 1000 0
VSIM 64> run

VSIM 65>
```

```
601 ps to 706 ps    Project : alu    Now: 700 ps  Delta: 0    sim:/zhang_alu
```

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.