Lab: Add-Sub unit with Register file (3-ported RAM)

Chue Zhang

Csc343 Fall 2021

Professor Gertner

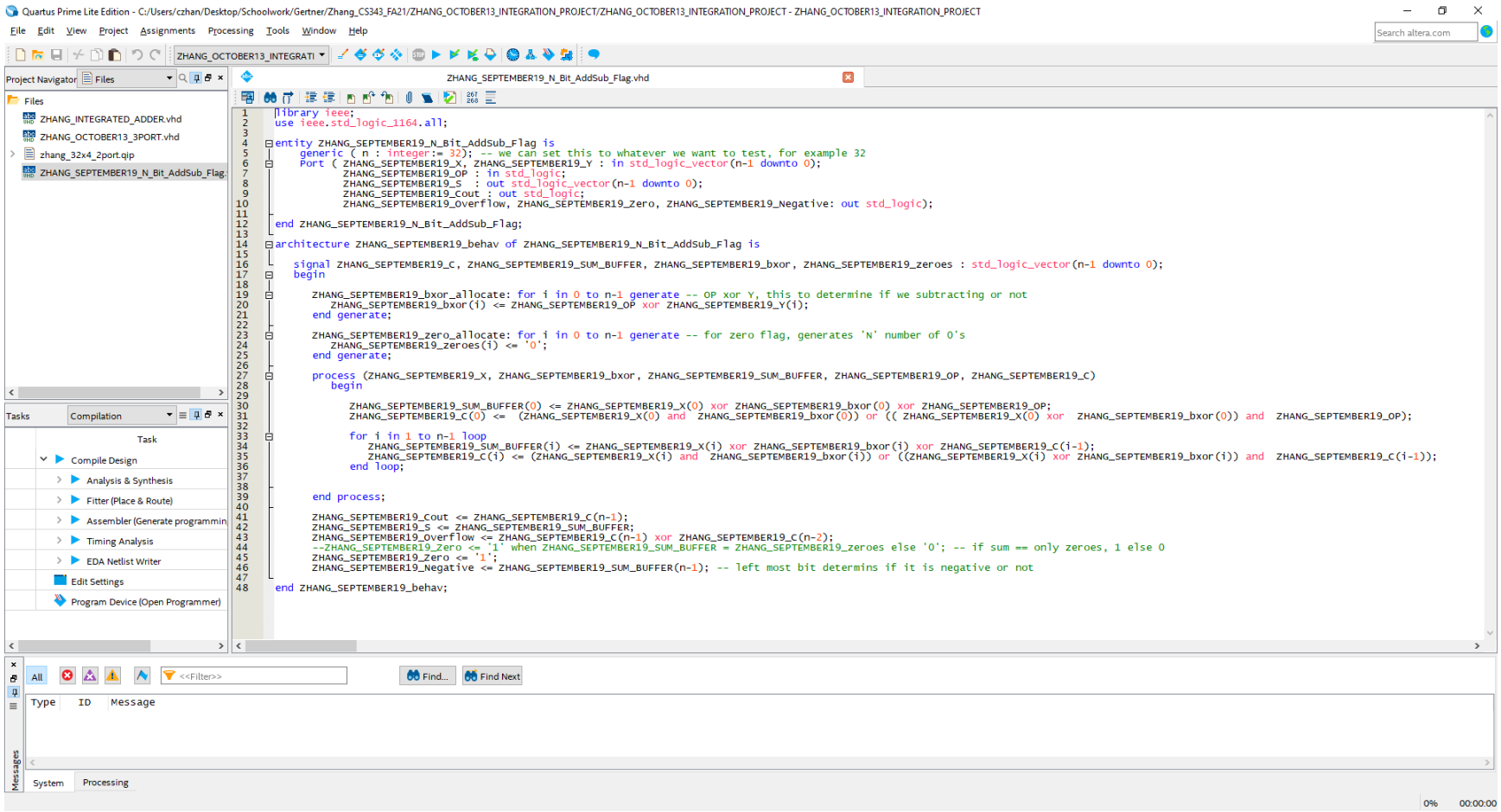
Table of contents

Objective

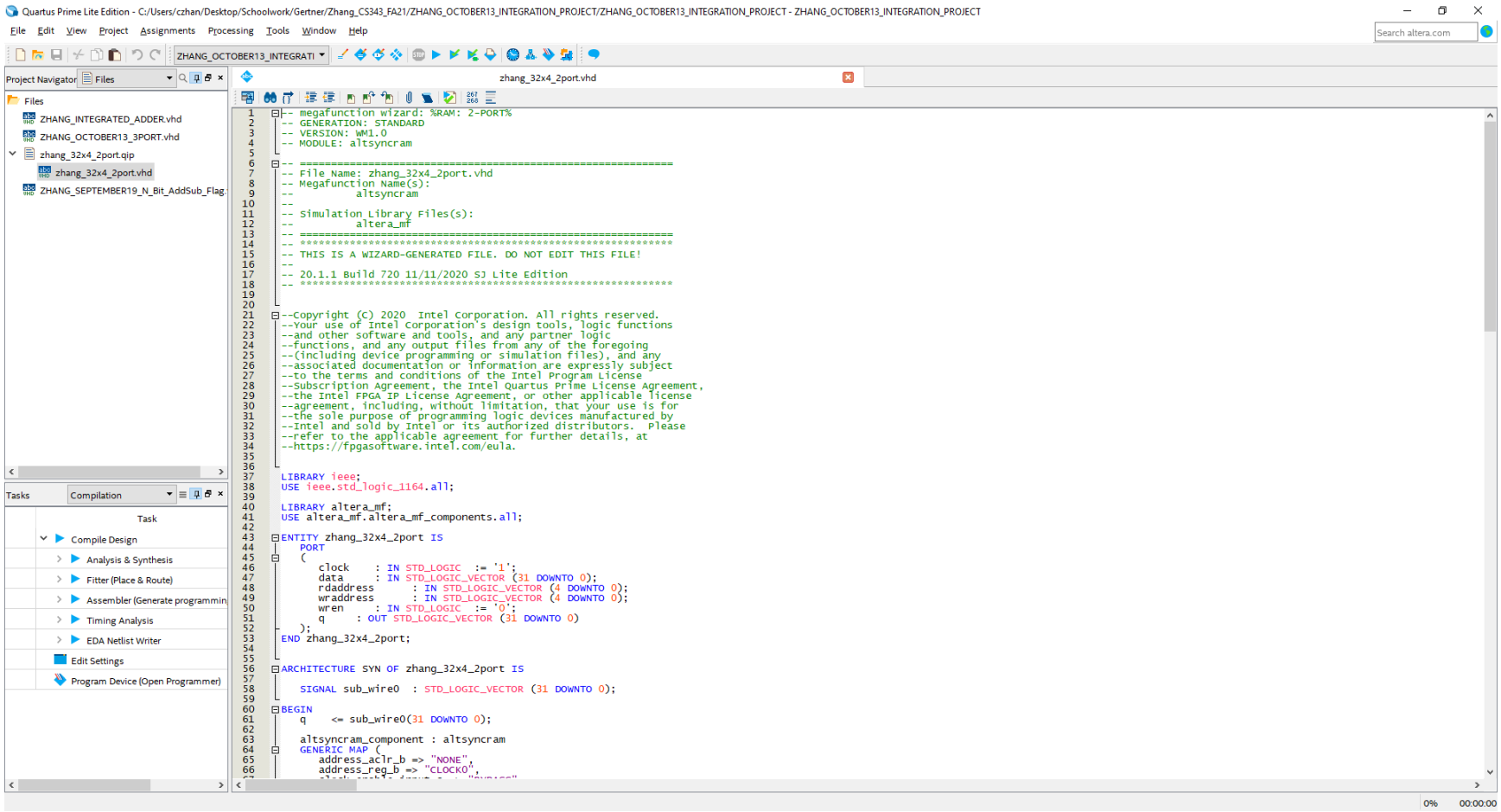
In this Laboratory Assignment, we will be Integrating an 32-bit Adder/Subtractor Unit into a 32 x 32 3-Port RAM with a Memory Instruction File(MIF). Furthermore, what will be included is a 32-bit MIPS like instruction that will be read and inputted into the 32-bit Adder/Subtractor unit.

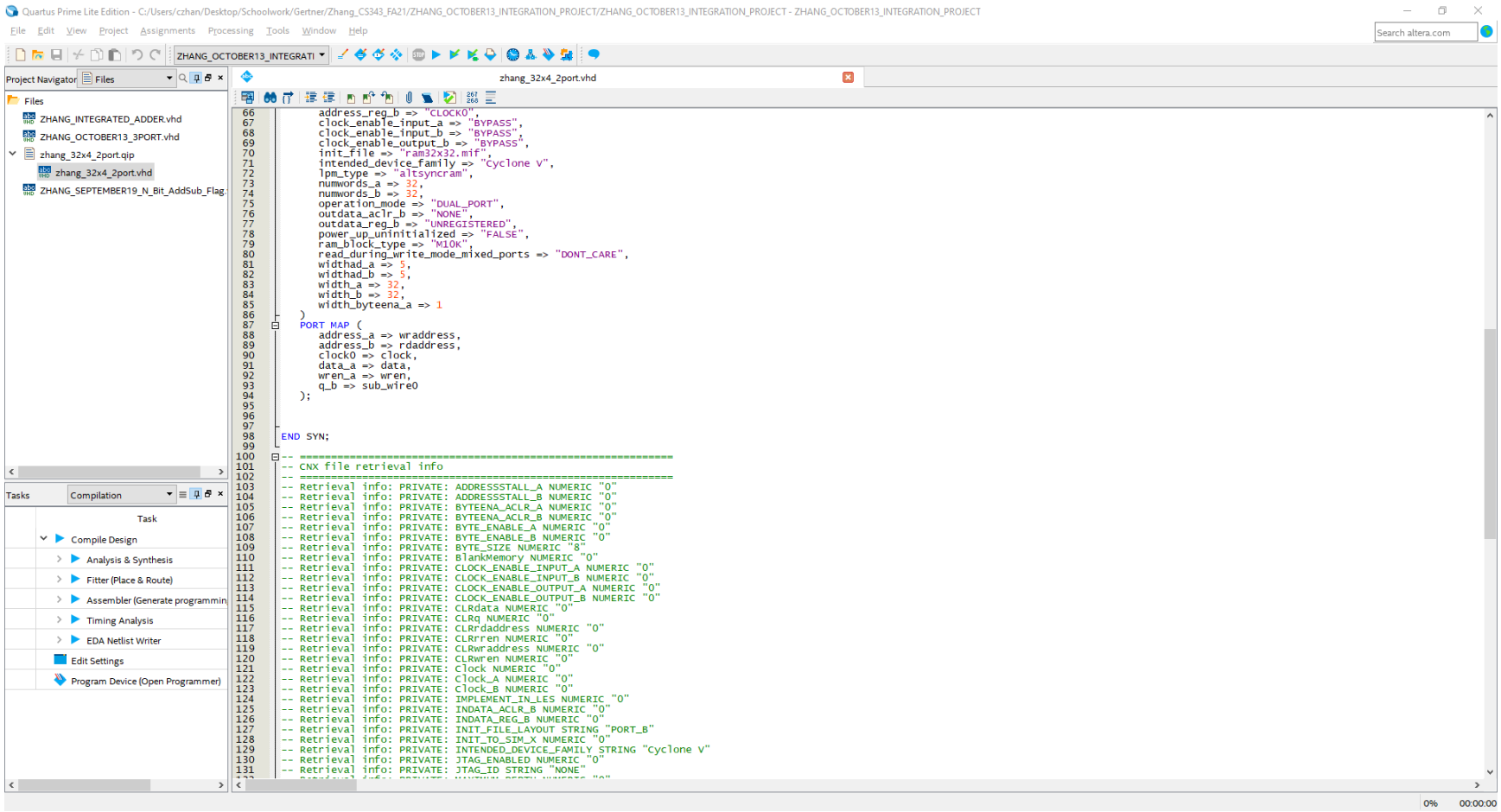
Specification

A 32-Bit MIPS like instruction is inputted in which is broken down in the **operation code [6 bits], RS[5 bits], RT[5 bits], RD[5 bits]** with the remaining bits being 0's**. Operation code** is 6-bit longs where if "000000" , it will add and "000001" would be to subtract. Furthermore, **RS** and **RT** are the input values that we insert into the Adder/Subtractor unit and **RD** is where we are writing to. Next, we integrate an Adder/Subtractor Unit into the 3-Port Ram where the first 3 addresses of our MIF file are instantiated with the **Most Positive 32-Bit Integer**, **Most Negative 32-bit Integer and 1.** The User will specify the address in the 32-bit instruction to obtain the 32-Bit Integer in which Addition or Subtraction can be performed on these values. After computation is completed, the value will be written into and of the 29 other addresses that were not instantiated in the MIF file. For example, we compute **Most Positive 32-Bit Integer + 1,** where once computations are finished, we can write the value into the 4^{th} address in the MIF file which has not been instantiated.

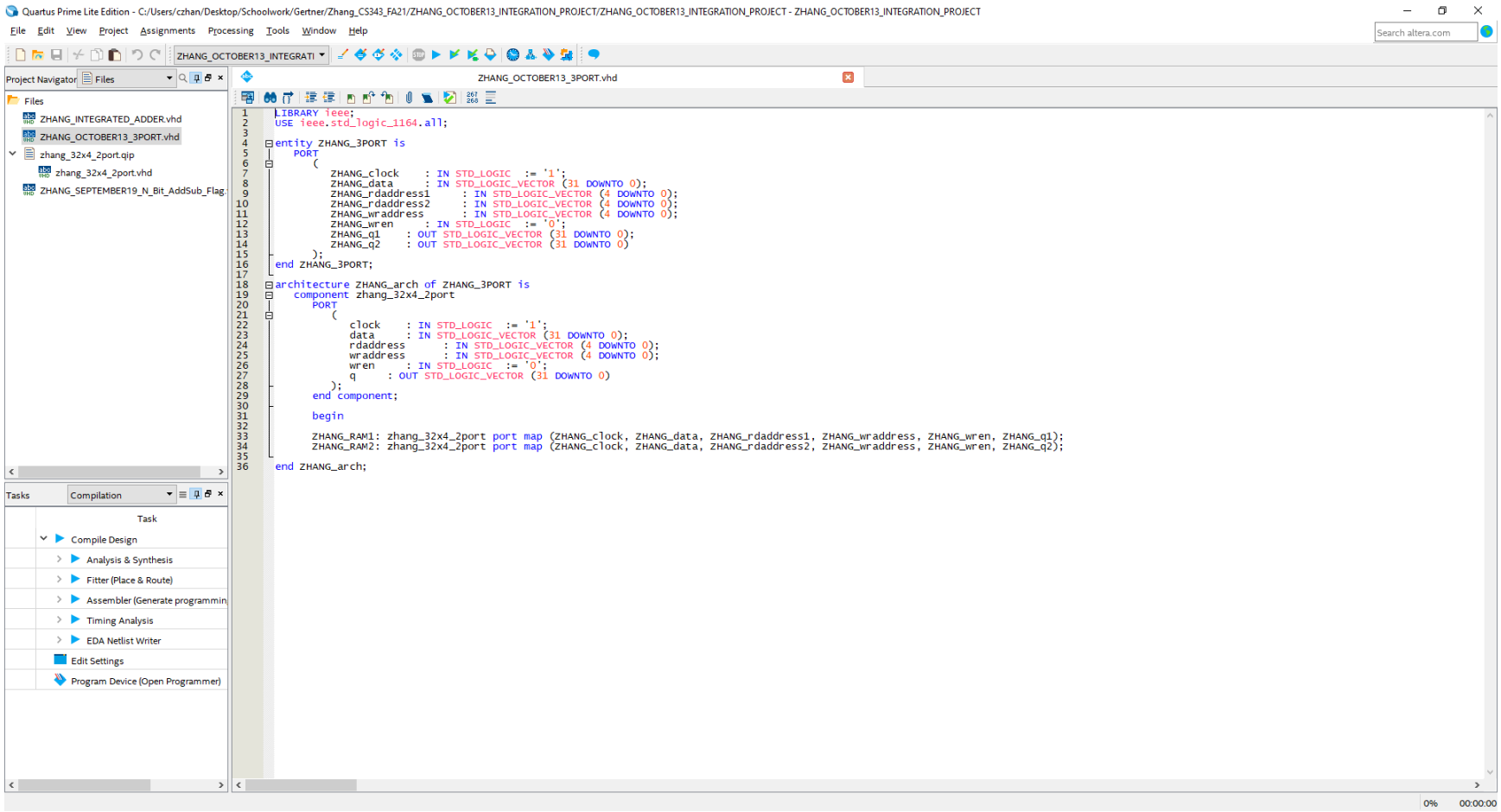


32-Bit Adder/subtractor unit that was created previously in the Adder Laboratory assignment. What is included is logic that detects for Overflow, Negative and Zero as well as an Operator Bit that determines whether or not a number should perform subtraction or addition.

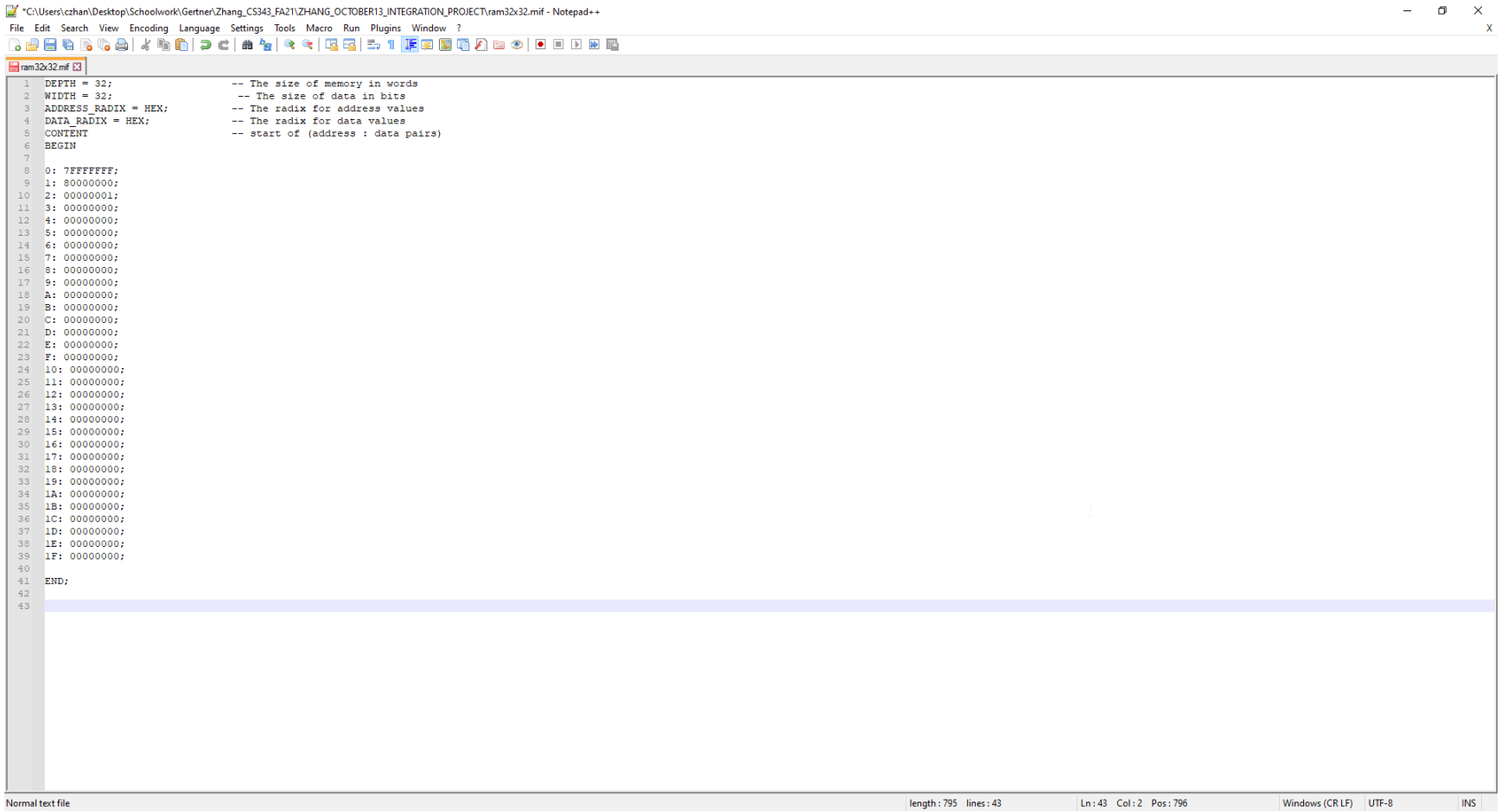

Quartus Prime Lite Edition - C:/Users/czhan/Desktop/Schoolwork/Gertner/Zhang_CS343_FA21/ZHANG_OCTOBER13_INTEGRATION_PROJECT/ZHANG_OCTOBER13_INTEGRATION_PROJECT - ZHANG_OCTOBER13_INTEGRATION_PROJECT
File Edit View Project Assignments Processing Tools Window Help
Search altera.com
Project Navigator
Files
ZHANG_INTEGRATED_ADDER.vhd
ZHANG_OCTOBER13_3PORT.vhd
zhang_32x4_2port.qip
zhang_32x4_2port.vhd
ZHANG_SEPTEMBER19_N_Bit_AddSub_Flag
Tasks
Compilation
Task
Compile Design
Analysis & Synthesis
Fitter (Place & Route)
Assembler (Generate programmin
Timing Analysis
EDA Netlist Writer
Edit Settings
Program Device (Open Programmer)
zhang_32x4_2port.vhd
-- megafunction wizard: %RAM: 2-PORT%
-- GENERATION: STANDARD
-- VERSION: WM1.0
-- MODULE: altsyncram
-- File Name: zhang_32x4_2port.vhd
-- Megafunction Name(s):
-- altsyncram
-- Simulation Library Files(s):
-- altera_mf
-- THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
-- 20.1.1 Build 720 11/11/2020 SJ Lite Edition
--Copyright (C) 2020 Intel Corporation. All rights reserved.
--Your use of Intel Corporation's design tools, logic functions
--and other software and tools, and any partner logic
--functions, and any output files from any of the foregoing
--(including device programming or simulation files), and any
--associated documentation or information are expressly subject
--to the terms and conditions of the Intel Program License
--Subscription Agreement, the Intel Quartus Prime License Agreement,
--the Intel FPGA IP License Agreement, or other applicable license
--agreement, including, without limitation, that your use is for
--the sole purpose of programming logic devices manufactured by
--Intel and sold by Intel or its authorized distributors. Please
--refer to the applicable agreement for further details, at
--https://fpgasoftware.intel.com/eula.
LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY altera_mf;
USE altera_mf.altera_mf_components.all;
ENTITY zhang_32x4_2port IS
PORT
(
clock : IN STD_LOGIC := '1';
data : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
rdaddress : IN STD_LOGIC_VECTOR (4 DOWNTO 0);
wraddress : IN STD_LOGIC_VECTOR (4 DOWNTO 0);
wren : IN STD_LOGIC := '0';
q : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
);
END zhang_32x4_2port;
ARCHITECTURE SYN OF zhang_32x4_2port IS
SIGNAL sub_wire0 : STD_LOGIC_VECTOR (31 DOWNTO 0);
BEGIN
q <= sub_wire0(31 DOWNTO 0);
altsyncram_component : altsyncram
GENERIC MAP (
address_aclr_b => "NONE",
address_reg_b => "CLOCK0",
0%
00:00:00



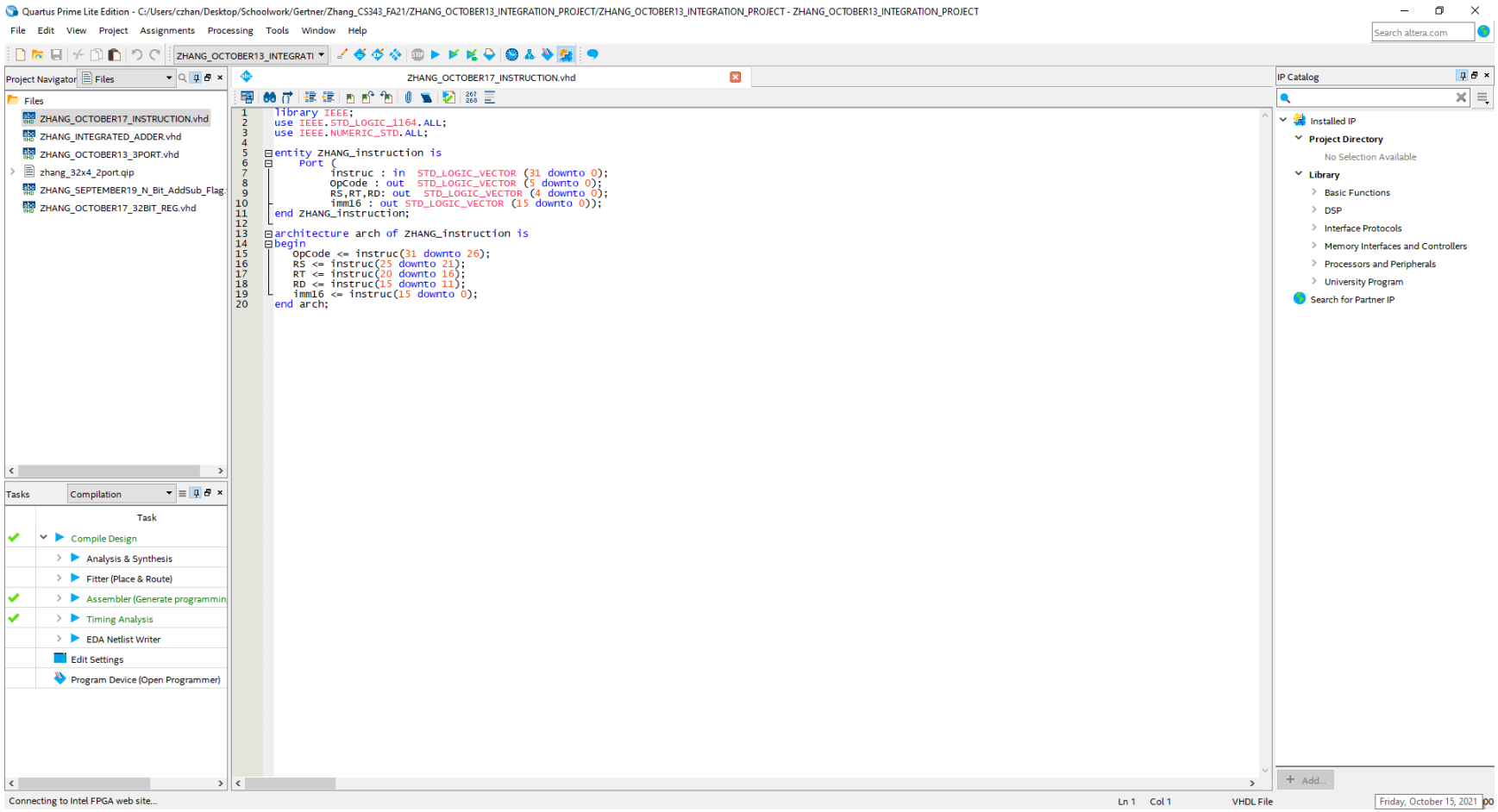
Here we have a 2-Port with 32x32 Memory that was generated using LPM in Quartus. This file was generated to require a MIF File to perform any functionality.



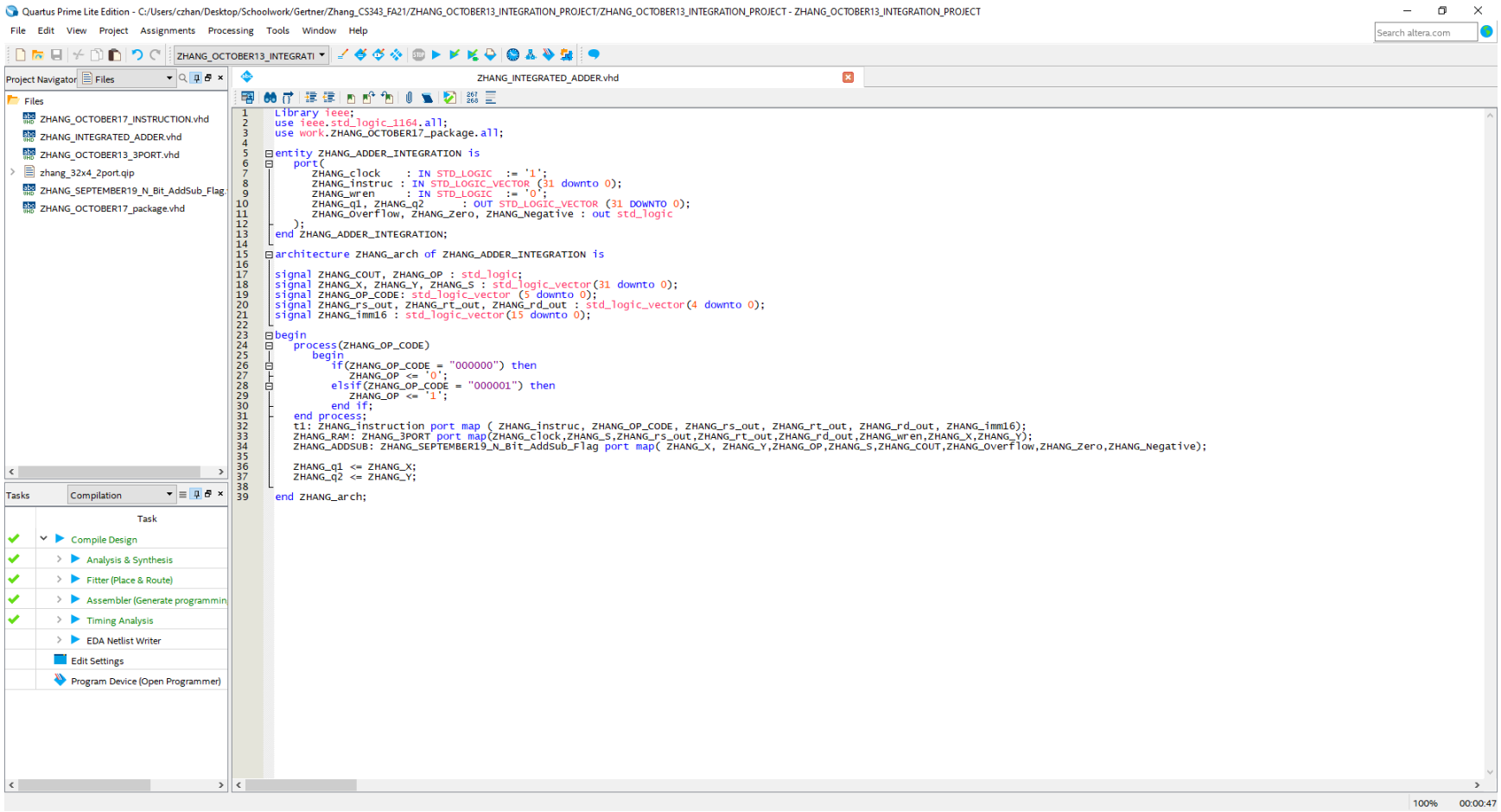
The code above showcases the usage of the previously generated 2-Port RAM in developing a 3-Port RAM. In a 3-port RAM, what is included is 2 read addresses and two q's which are the outputs after reading an address. Notice ZHANG_raddress1, ZHANG_raddress2, ZHANG_q1 and ZHANG_q2.



Above is the MIF file that will be used for reading and writing to. Notice that the 1st address points towards the Largest Positive value, the 2nd address points to the Largest Negative value and the 3rd address points to 1. The rest of the address point to 0 as a placeholder where we will later be writing new values into.



In the screenshot above showcases how I process the 32-bit MIPS like instruction



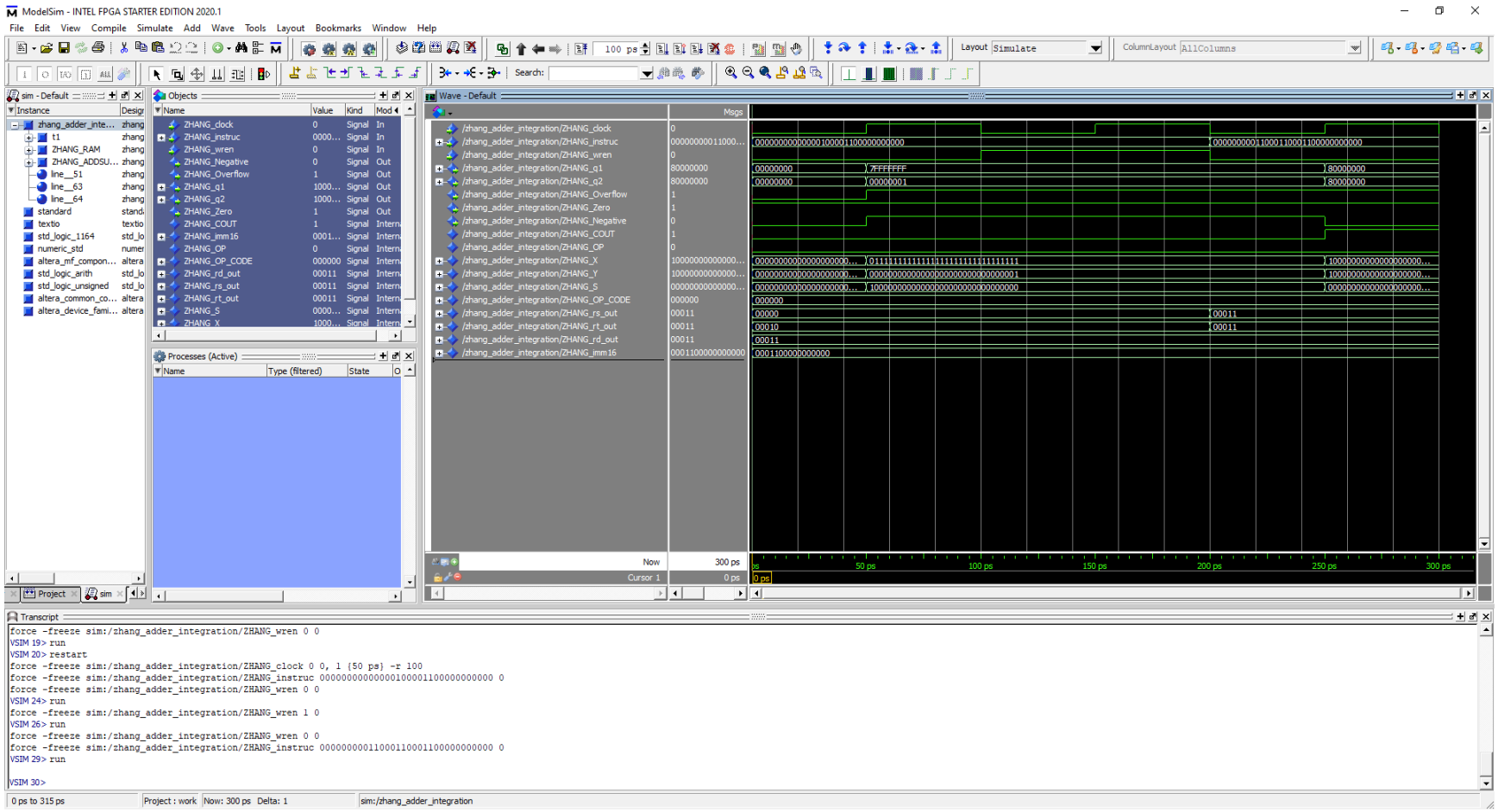
Above showcases the usage of the Instruction File, the 3-port ram and the 32-Bit Adder/Subtractor Unit

Simulations

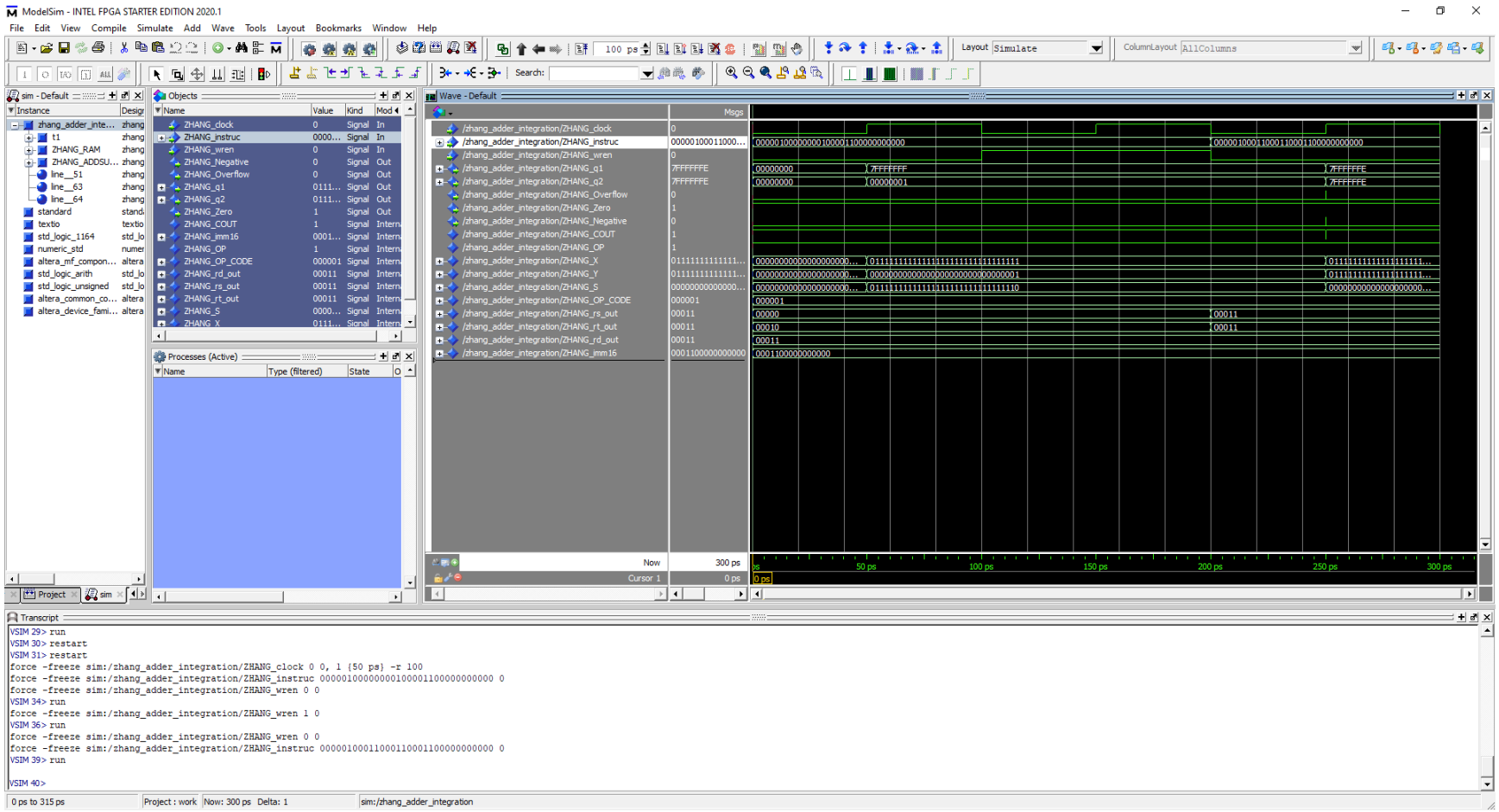
Simulations will be done to verify correctness of the 3-Port RAM Adder/Sub Unit and performed on Modelsim with waveform screenshots and explanations to further verify correctness.

What will be Tested is the following

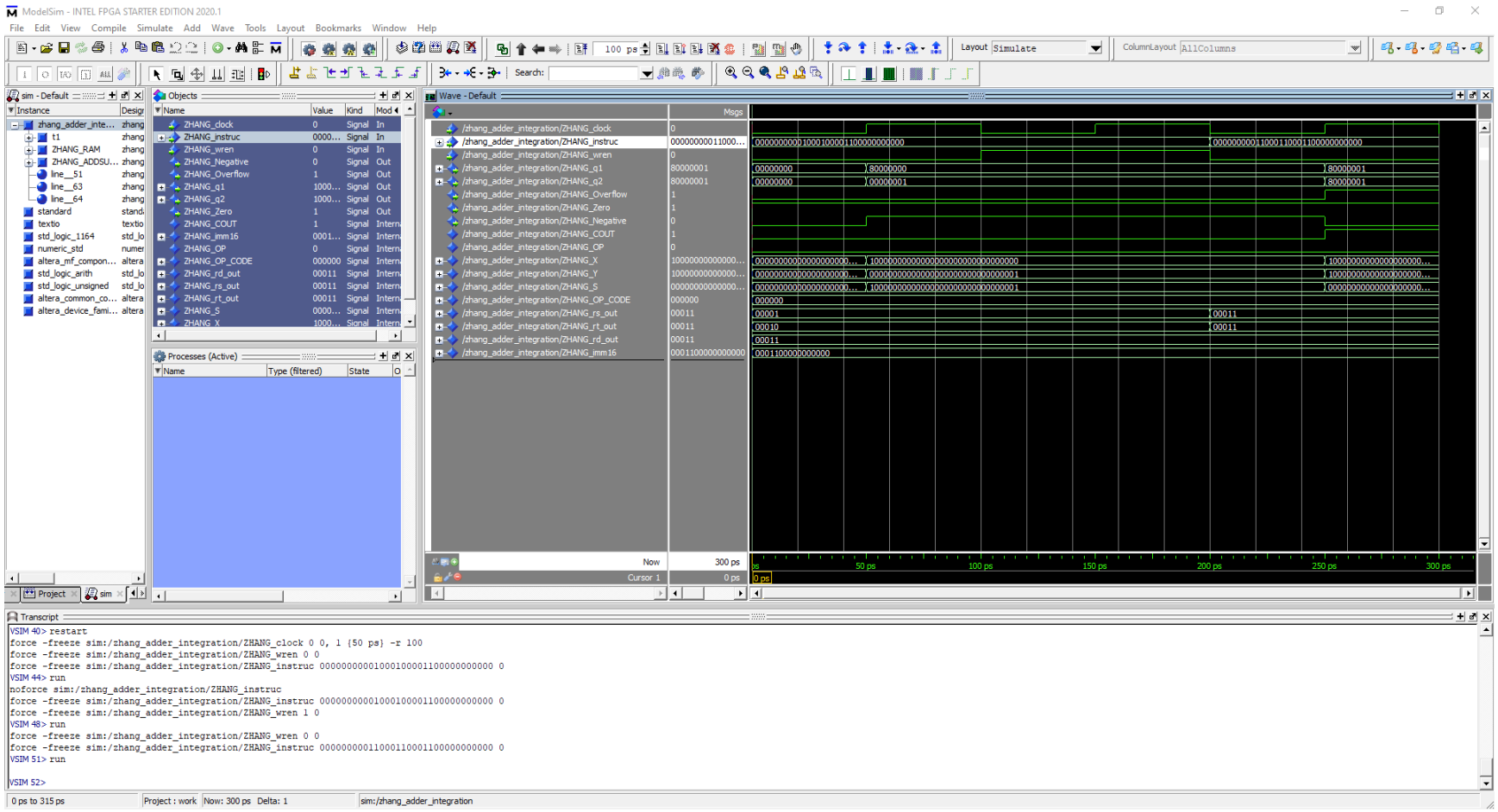
1. **Most Positive 32-Bit Integer + 1**
2. **Most Positive 32-Bit Integer – 1**
3. **Most Negative 32-Bit Integer + 1**
4. **Most Negative 32-Bit Integer – 1**
5. **Most Positive 32-Bit Integer – Most Negative 32-Bit Integer**
6. **Most Positive 32-Bit Integer + Most Positive 32-Bit Integer**
7. **Most positive 32-Bit Integer – Most Positive 32-Bit Integer**



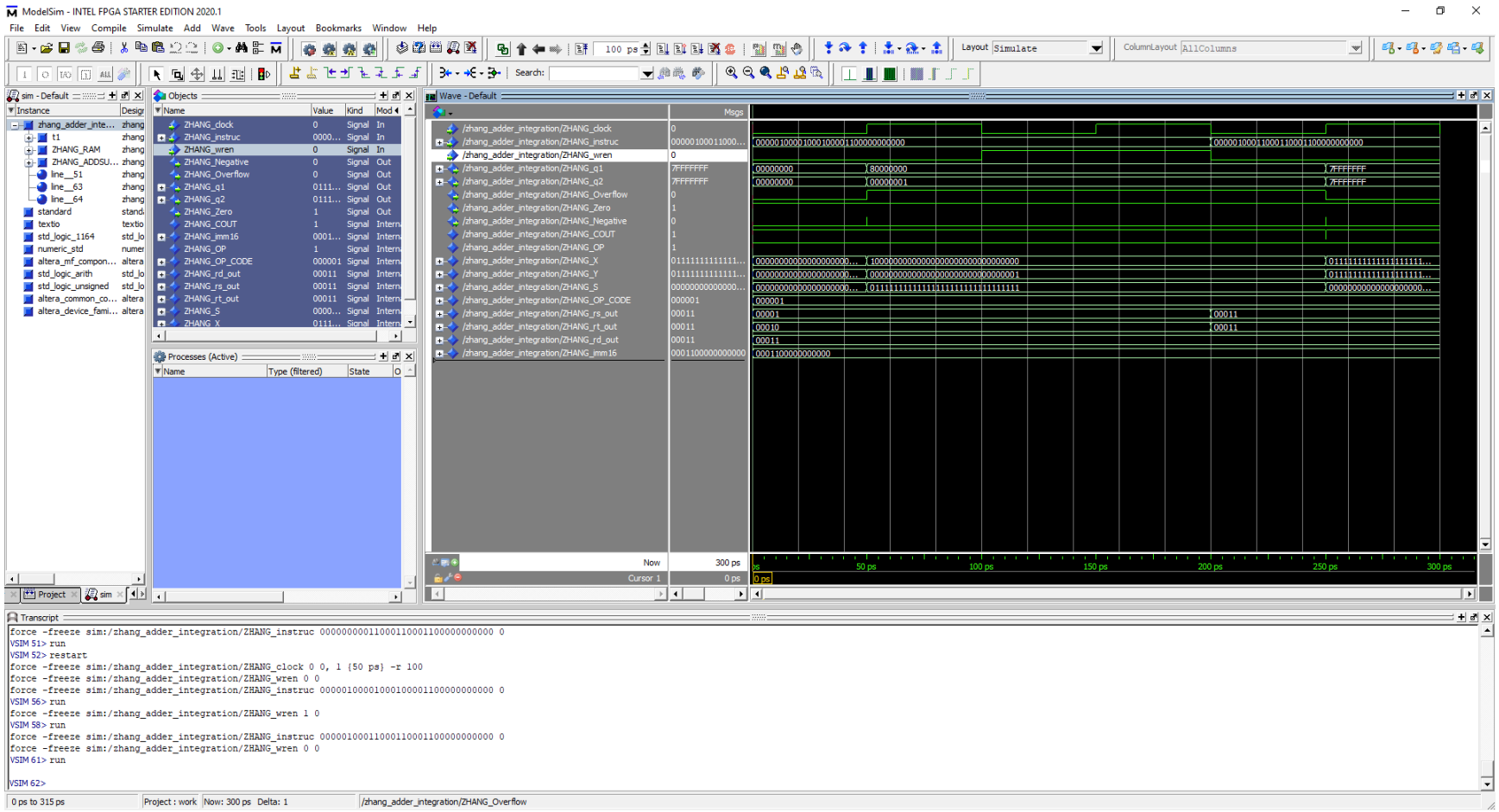
What is performed is the **Most Positive 32-Bit Integer + 1** where a 32-bit Instruction is read where once the instruction is broken down they will be inputted into the adder/subtractor where the 3-port ram calls the address from in the MIF File so that the Mathematical process can be performed then stored. Instruction bit is 000000 00000 00010 00011 00000000000, OP code, RS, RT, RD, Function respectively. Notice that the 1st Address and the 3rd Address is being called from the MIF, computed, then written into the 4th Address. Furthermore, Overflow flag and Negative flag has been triggered.



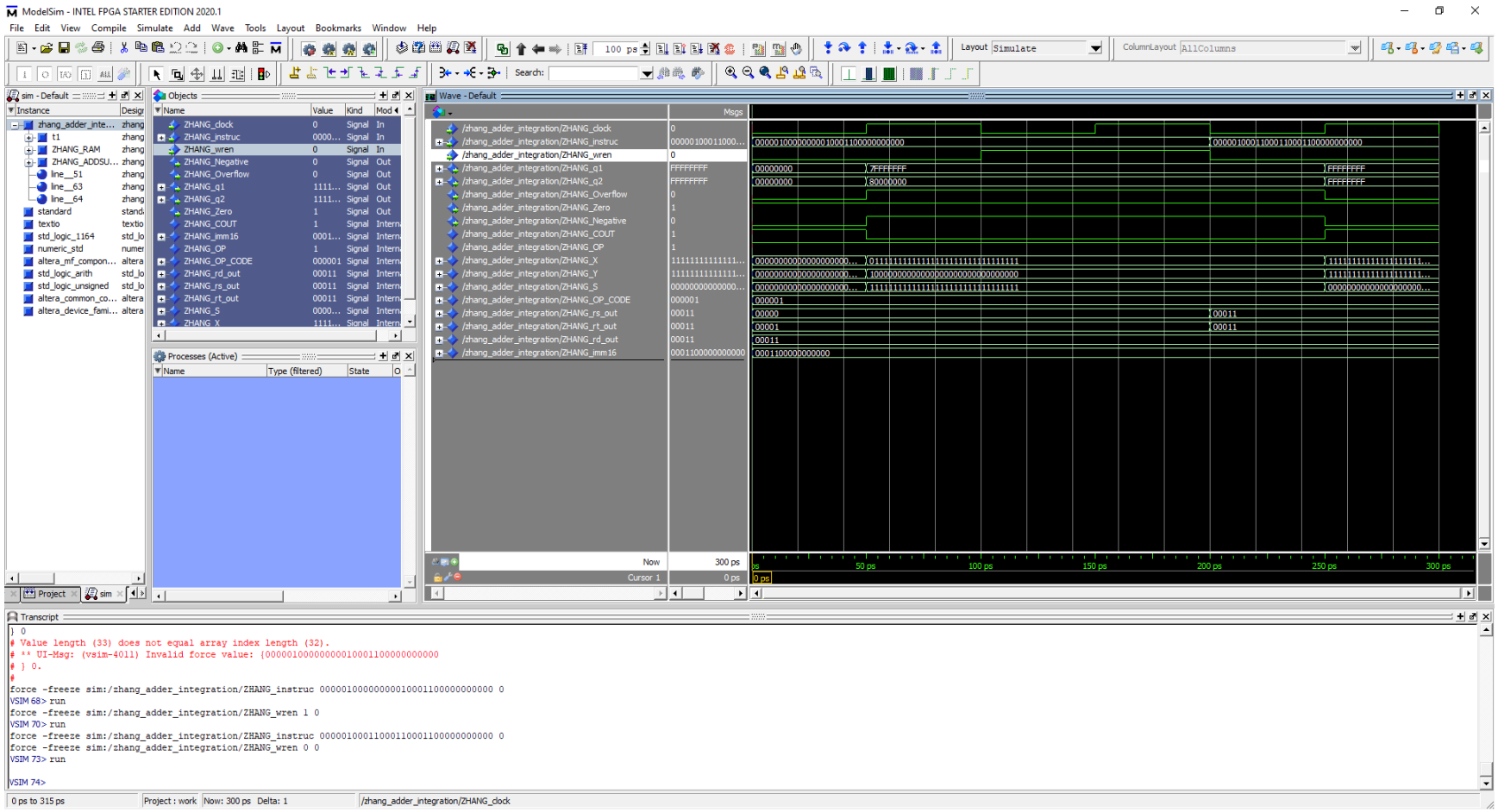
What is performed is the **Most Positive 32-Bit Integer - 1** where a 32-bit Instruction is read where once the instruction is broken down they will be inputted into the adder/subtractor where the 3-port ram calls the address from in the MIF File so that the Mathematical process can be performed then stored. Instruction bit is 000001 00000 00010 00011 00000000000, OP code, RS, RT, RD, Function respectively. Notice that the 1st Address and the 3rd Address is being called from the MIF, computed, then written into the 4th Address. Furthermore, zero flag has been triggered



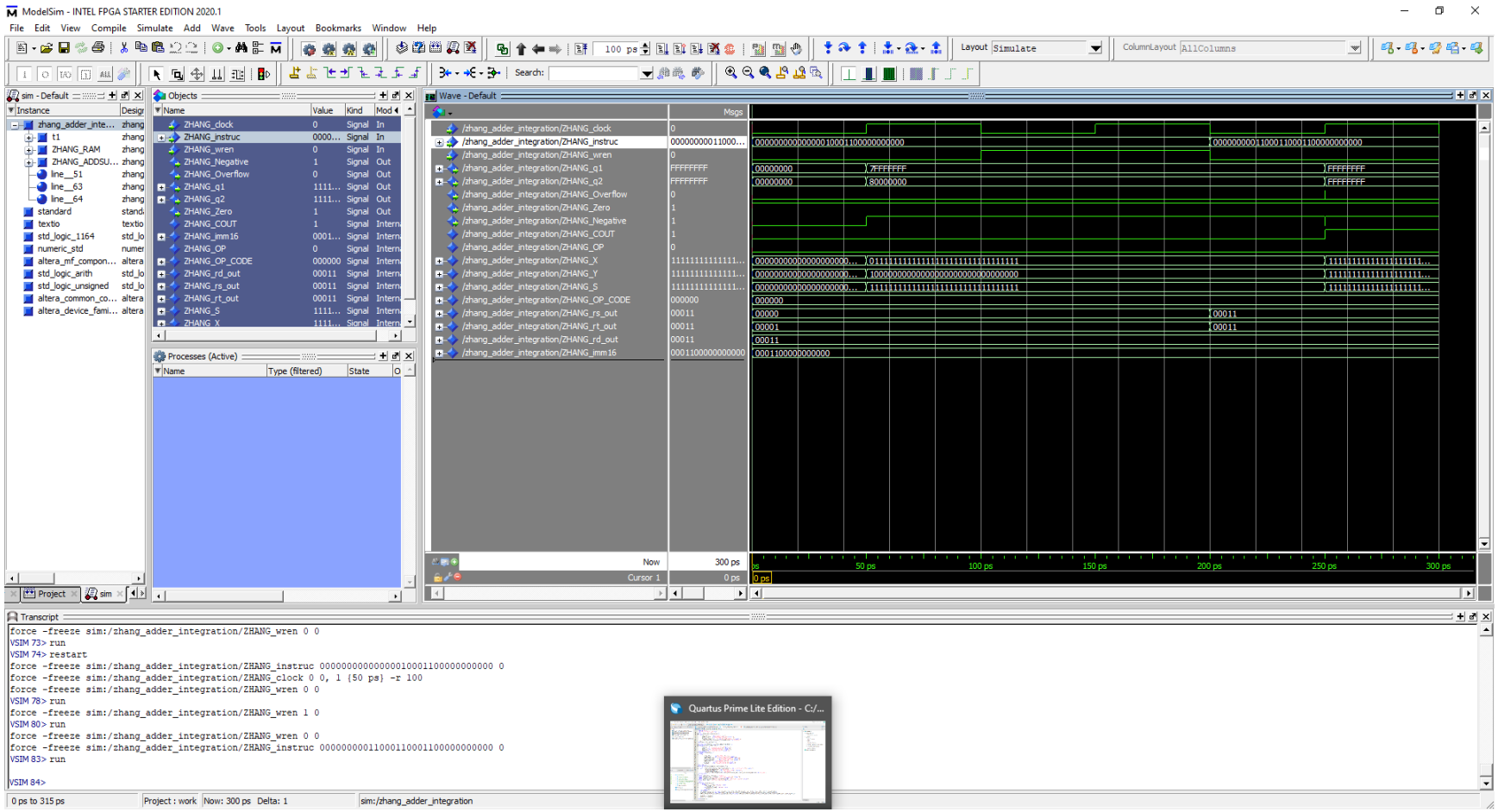
What is performed is the **Most Negative 32-Bit Integer + 1** where a 32-bit Instruction is read where once the instruction is broken down they will be inputted into the adder/subtractor where the 3-port ram calls the address from in the MIF File so that the Mathematical process can be performed then stored. Instruction bit is 000000 00001 00010 00011 00000000000, OP code, RS, RT, RD, Function respectively. Notice that the 2nd Address and the 3rd Address is being called from the MIF, computed, then written into the 4th Address. Furthermore, notice the Overflow Flag.



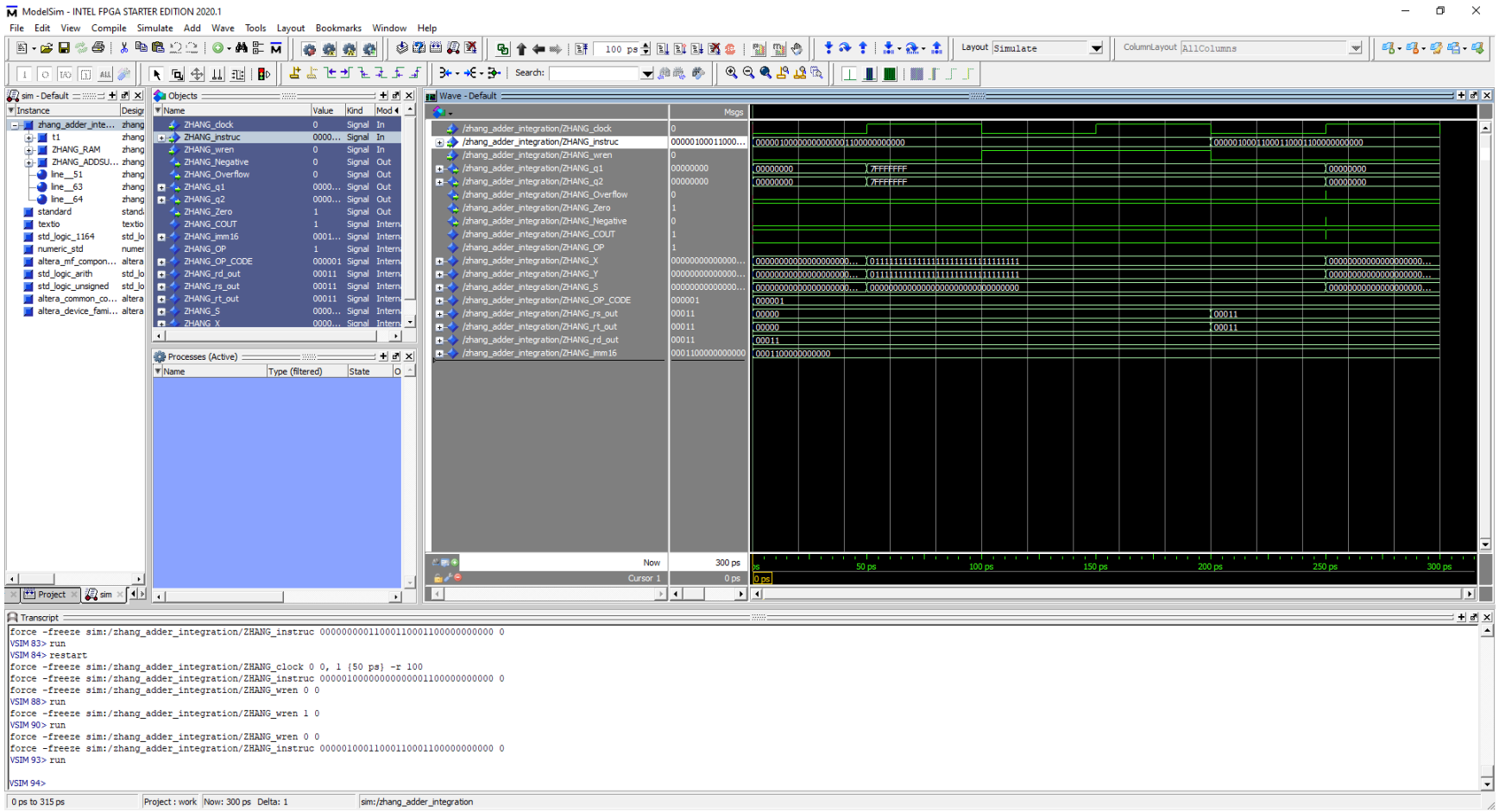
What is performed is the **Most Negative 32-Bit Integer - 1** where a 32-bit Instruction is read where once the instruction is broken down they will be inputted into the adder/subtractor where the 3-port ram calls the address from in the MIF File so that the Mathematical process can be performed then stored. Instruction bit is 000001 00001 00010 00011 00000000000, OP code, RS, RT, RD, Function respectively. Notice that the 2nd Address and the 3rd Address is being called from the MIF, computed, then written into the 4th Address.



What is performed is the **Most Positive 32-Bit Integer - Most Negative 32-Bit Integer** where a 32-bit Instruction is read where once the instruction is broken down they will be inputted into the adder/subtractor where the 3-port ram calls the address from in the MIF File so that the Mathematical process can be performed then stored. Instruction bit is 000001 00000 00001 00011 00000000000, OP code, RS, RT, RD, Function respectively. Notice that the 1st Address and the 2nd Address is being called from the MIF, computed, then written into the 4th Address.



What is performed is the **Most Positive 32-Bit Integer + Most Negative 32-Bit Integer** where a 32-bit Instruction is read where once the instruction is broken down they will be inputted into the adder/subtractor where the 3-port ram calls the address from in the MIF File so that the Mathematical process can be performed then stored. Instruction bit is 000000 00000 00001 00011 00000000000, OP code, RS, RT, RD, Function respectively. Notice the 1st Address and the 2nd Address is being called from the MIF, computed, then written into the 4th address. Notice the negative flag being triggered.



What is performed is the **Most Positive 32-Bit Integer - Most Positive 32-Bit Integer** where a 32-bit Instruction is read where once the instruction is broken down they will be inputted into the adder/subtractor where the 3-port ram calls the address from in the MIF File so that the Mathematical process can be performed then stored. Instruction bit is 000001 00000 00000 00011 00000000000, OP code, RS, RT, RD, Function respectively. Notice the Zero flag being triggered.

Conclusion

By doing this laboratory assignment, I was able to get familiar with the usage of the MIF file and how it works. I also learned more about how Register block works as well as how things are read and written. By the end of this lab, I have gained much knowledge on the importance of the usage of RAM as well as how to integrate units into it.