Name: Duc Anh Nguyen

Course: Computer Engineering with lab

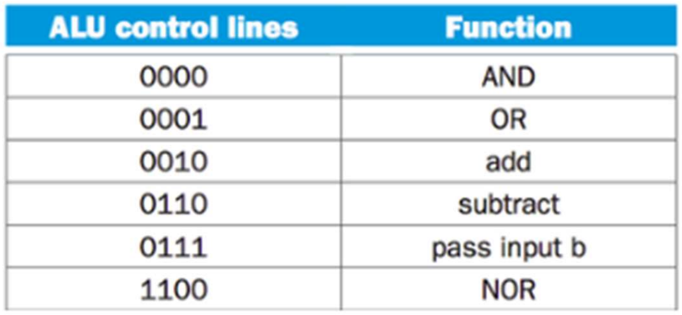
Date: 10.7.2022

Lab Report 2

Introduction: The goal of this lab is continuing the process of building components of LEGv8. The components for this lab are ALU, ADD, ALU control, Registers, Instruction memory, and data memory. Moreover, test benches are required to verify the functionality of the components.

Procedure:

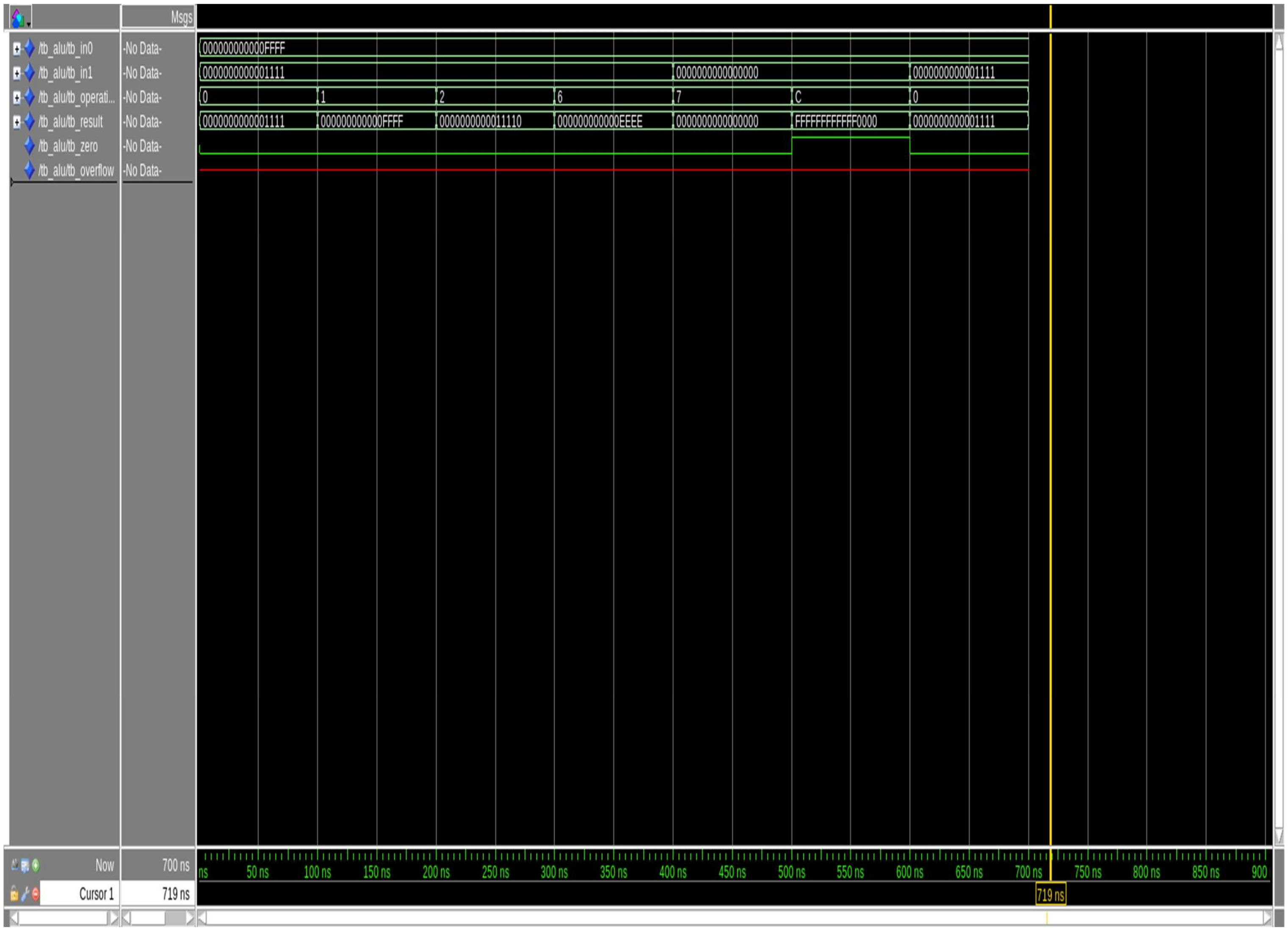
1. Components 1: Adder
   a. The adder is the basic arithmetic unit for the ALU, which will control the two operations of ALU, add and subtract
   b. By using structural behavior, a 64-bit adder can be built by using n-1 ripple full adder. I basically build a 2-bit half adder and a 2-bit full adder and use those components to build the 64-bit full adder
   c. These are the test case for my adder:
      i. Set a = X “000000000000FFFF” and b = X “0000000000001111”, Answer should be X “11110”
      ii. Set a = X “000000000000EEEE” and b = X “0000000000022222”, Answer should be X “31110”
2. Components 2: ALU control
   a. ALU control basically follows the following operation code and operation



ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	pass input b
1100	NOR

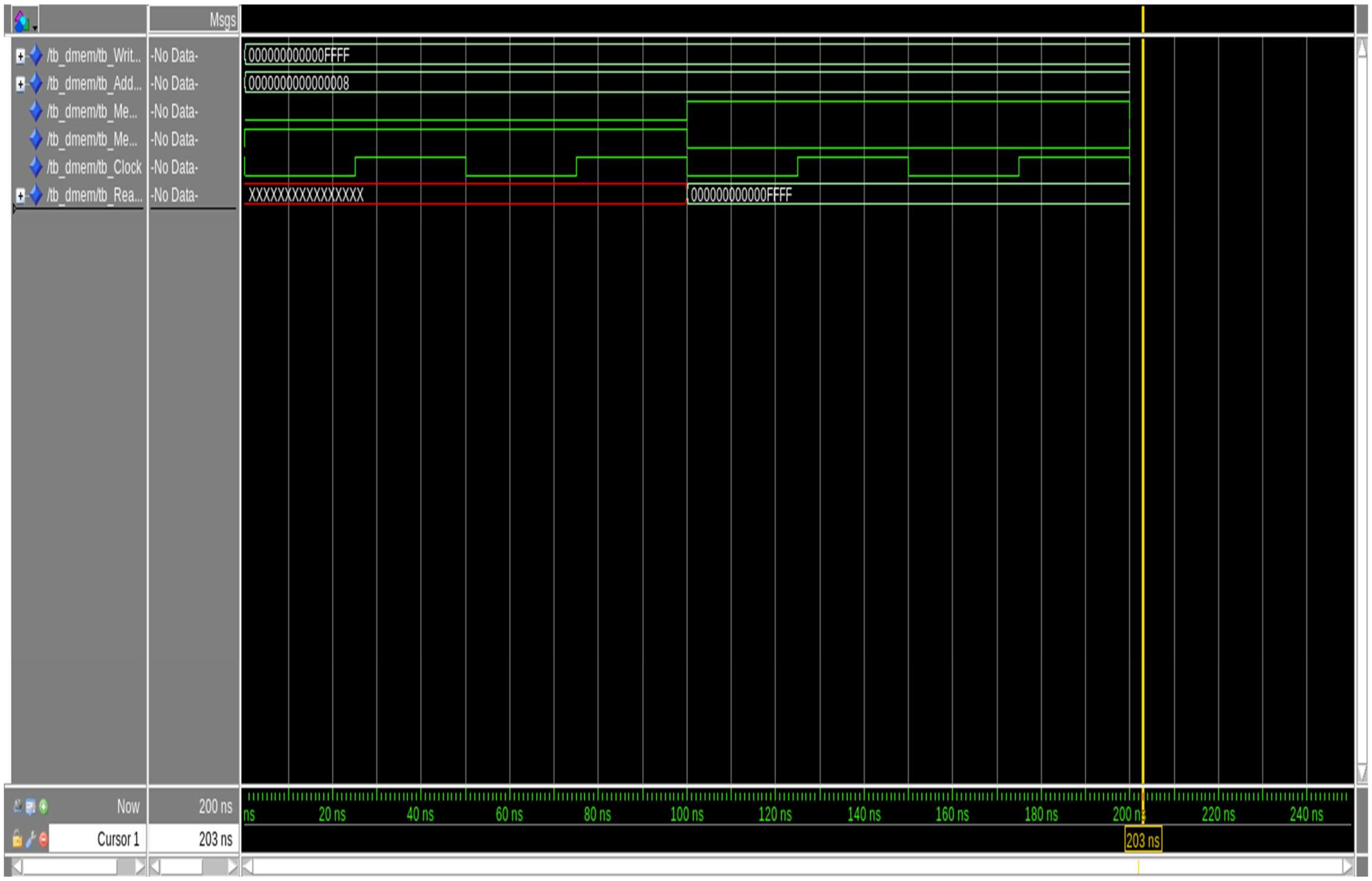
3. Components 3: ALU
   a. ALU will be following the ALU control lines in the previous components to perform certain operation. Moreover, it will give different output depends on the operation
   b. It performs 5 operations: ADD, OR, ADD, SUBTRACT, pass input b, and NOR.
   c. These are the test cases for my ALU:
      - There will be a constant value for input 0: X"000000000000FFFF" and input 1: X"0000000000001111"

- For operation AND, the answer should be X"0000000000001111"
- For operation OR, the answer should be X"000000000000FFFF"
- For operation ADD, the answer should be X"0000000000011110"
- For operation SUBTRACT, the answer should be X"000000000000EEEE"
- For operation "pass input b", the answer should be X"0000000000001111"
- For operation NOR, the answer should be X"FFFFFFFFFFFF0000"

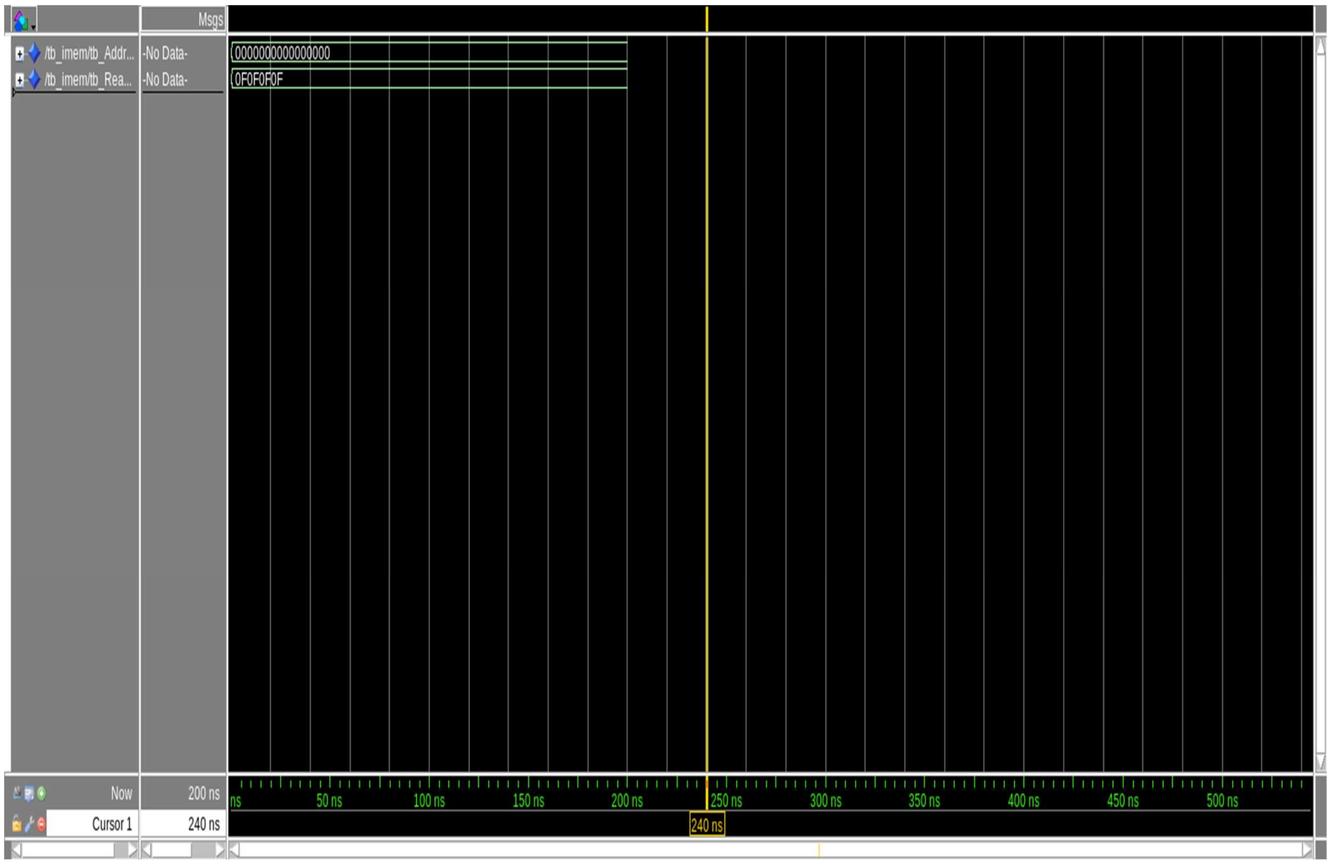

Msgs
/tb_alu/tb_in0 -No Data- 000000000000FFFF
/tb_alu/tb_in1 -No Data- 0000000000001111 0000000000000000 0000000000001111
/tb_alu/tb_operati... -No Data- 0 1 2 6 7 C 0
/tb_alu/tb_result -No Data- 0000000000001111 000000000000FFFF 0000000000011110 000000000000EEEE 0000000000000000 FFFFFFFFFFFF0000 0000000000001111
/tb_alu/tb_zero -No Data-
/tb_alu/tb_overflow -No Data-
Now 700 ns
Cursor 1 719 ns
ns 50 ns 100 ns 150 ns 200 ns 250 ns 300 ns 350 ns 400 ns 450 ns 500 ns 550 ns 600 ns 650 ns 700 ns 750 ns 800 ns 850 ns 900
719 ns

4. Components 4: Data memory
    a. The data memory is basically the memory that contains the data for the application. It can read or write into memory depends on the user by triggering the Read or Write Enable signal
    b. The data memory has the maximum capacity of 1 KB

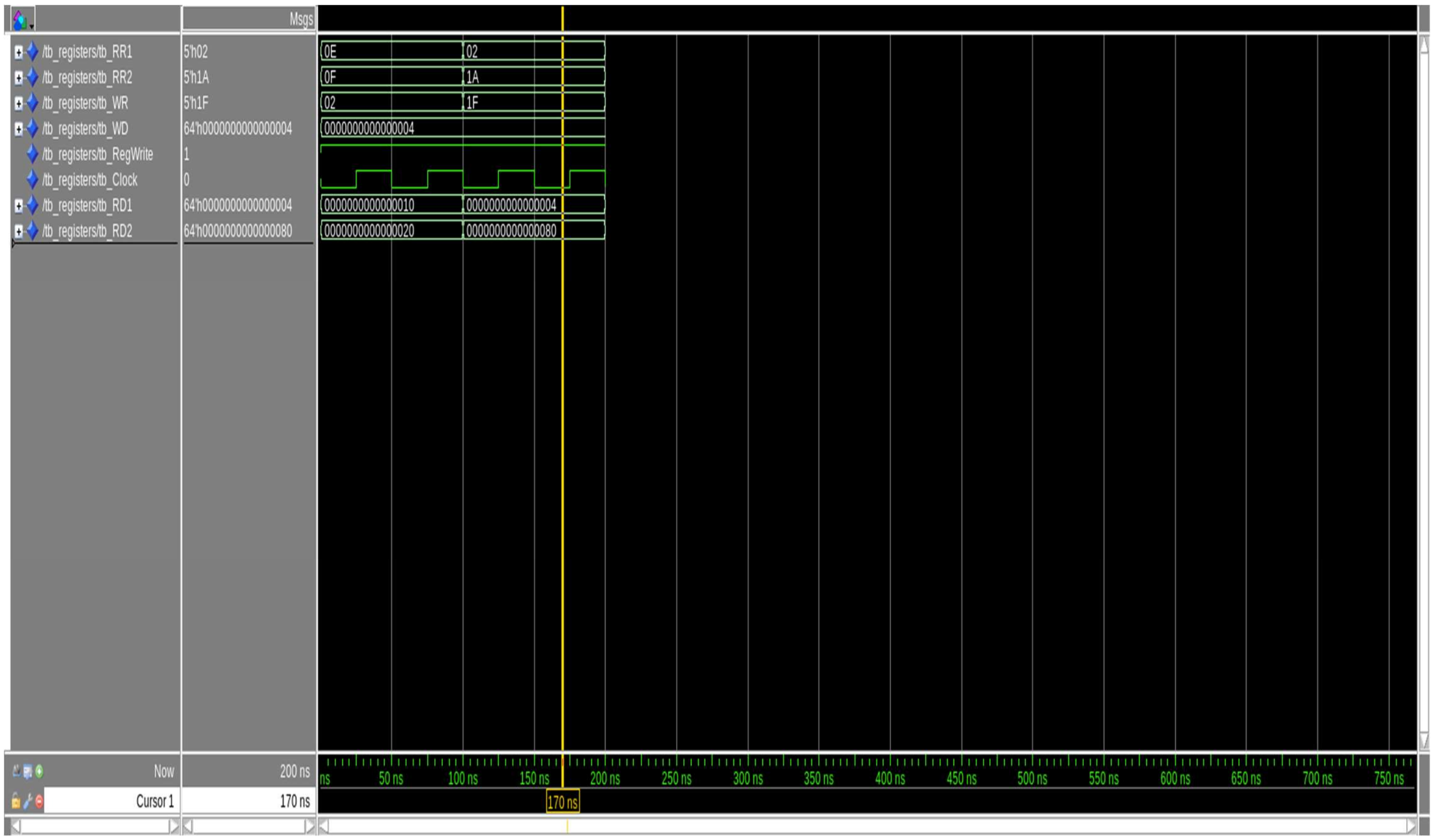
c. The test cases involve me trying to write X"000000000000FFFF" into X"0000000000000008" and reading from it



5. Components 5: CPU Control
    a. It basically generating signals for the CPU to know which instruction or operation are being use.
6. Components 6: Instruction Memory
    a. Like Data Memory, the Instruction Memory allocates the memory in where we user writes the program. We cannot write into the Instruction Memory, but we can read from it. It shares similarities with ROM.
    b. I put a few different values on the main code at address (0) and read from it.

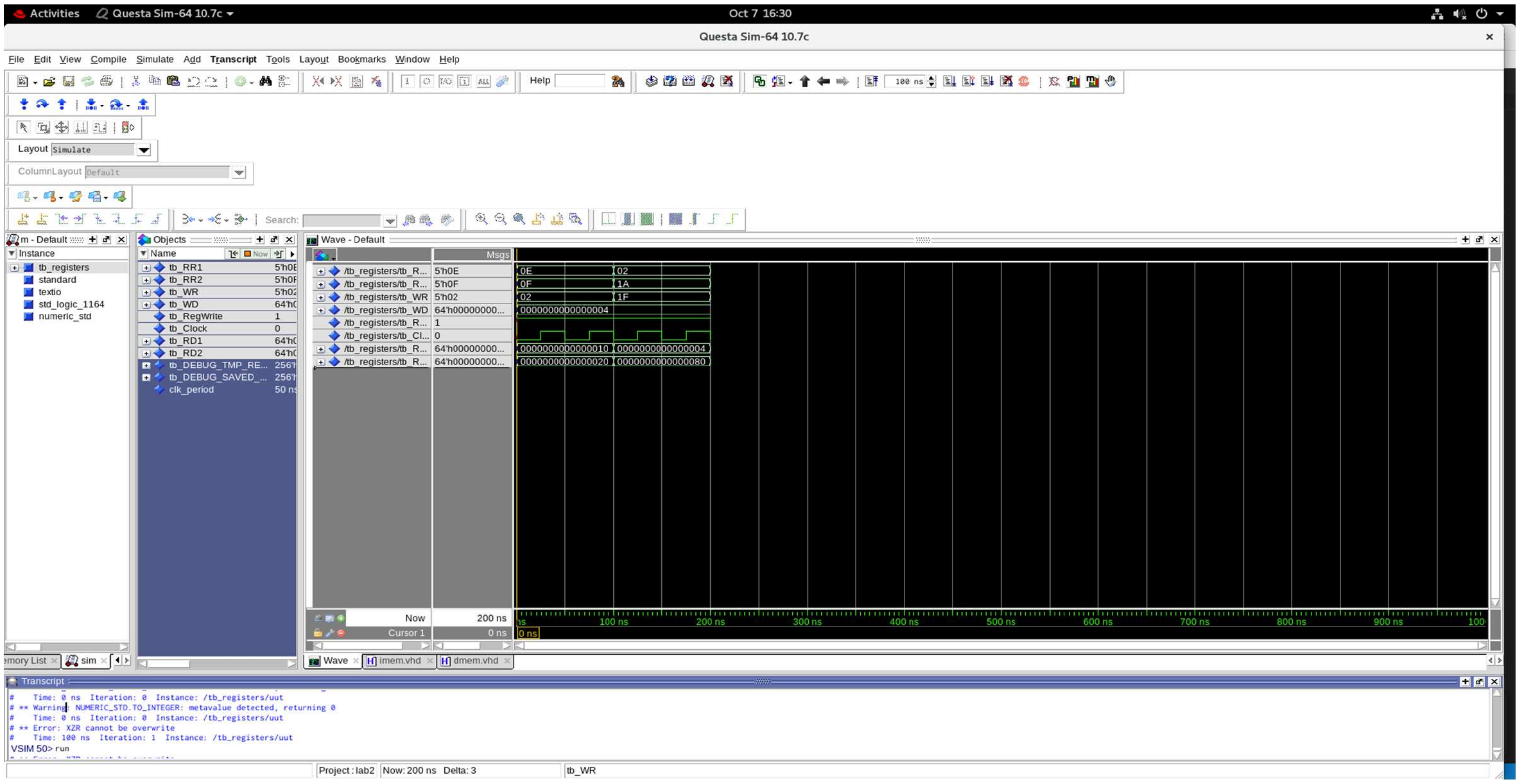


7. Components 7: Registers
    a. The registers follow the architecture of LEGv8 and contains 32 registers with 64-bits wide. The design of the register can help user write or read from these registers
    b. The test cases involving me trying to read and write from the registers. In the last cases, I attempted to write onto the XZR or X31 and receive an error message from Questasim



```
Msgs
/tb_registers/tb_RR1       5'h02                   0E                  02
/tb_registers/tb_RR2       5'h1A                   0F                  1A
/tb_registers/tb_WR        5'h1F                   02                  1F
/tb_registers/tb_WD        64'h0000000000000004    0000000000000004
/tb_registers/tb_RegWrite  1
/tb_registers/tb_Clock     0
/tb_registers/tb_RD1       64'h0000000000000004    0000000000000010    0000000000000004
/tb_registers/tb_RD2       64'h0000000000000080    0000000000000020    0000000000000080

Now       200 ns
Cursor 1  170 ns     170 ns
```



```
Instance
tb_registers
standard
textio
std_logic_1164
numeric_std
```

```
Name                  Now
tb_RR1                5'h0E
tb_RR2                5'h0F
tb_WR                 5'h02
tb_WD                 64'h0
tb_RegWrite           1
tb_Clock              0
tb_RD1                64'h0
tb_RD2                64'h0
tb_DEBUG_TMP_RE...    256'
tb_DEBUG_SAVED_...    256'
clk_period            50 ns
```

```
Msgs
/tb_registers/tb_R...    5'h0E                0E                  02
/tb_registers/tb_R...    5'h0F                0F                  1A
/tb_registers/tb_WR      5'h02                02                  1F
/tb_registers/tb_WD      64'h00000000...      0000000000000004
/tb_registers/tb_R...    1
/tb_registers/tb_Cl...   0
/tb_registers/tb_R...    64'h00000000...      0000000000000010    0000000000000004
/tb_registers/tb_R...    64'h00000000...      0000000000000020    0000000000000080

Now       200 ns    ns  100 ns  200 ns  300 ns  400 ns  500 ns  600 ns  700 ns  800 ns  900 ns  100
Cursor 1  0 ns      0 ns
```

```
#    Time: 0 ns  Iteration: 0  Instance: /tb_registers/uut
# ** Warning: NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0
#    Time: 0 ns  Iteration: 0  Instance: /tb_registers/uut
# ** Error: XZR cannot be overwrite
#    Time: 100 ns  Iteration: 1  Instance: /tb_registers/uut
VSIM 50> run
```

```
Project : lab2    Now: 200 ns  Delta: 3    tb_WR
```