Computer Architecture
Lab One

Prepared By

Ahmed Aly Gamal El-Din El-Ghannam

Electronics and Communications - Level 4
ID: 19015292

Yahia Walid El-Dakhakhny

Electronics and Communications - Level 4
ID: 19016891

March 2024

Contents

List of Figures

Listings

1 Introduction

This report contains a design for a register file according to MIPS architecture as depicted in the lab manual—as shown in figure 1. All code and simulation/screenshots files can be found in the project's Github repository.

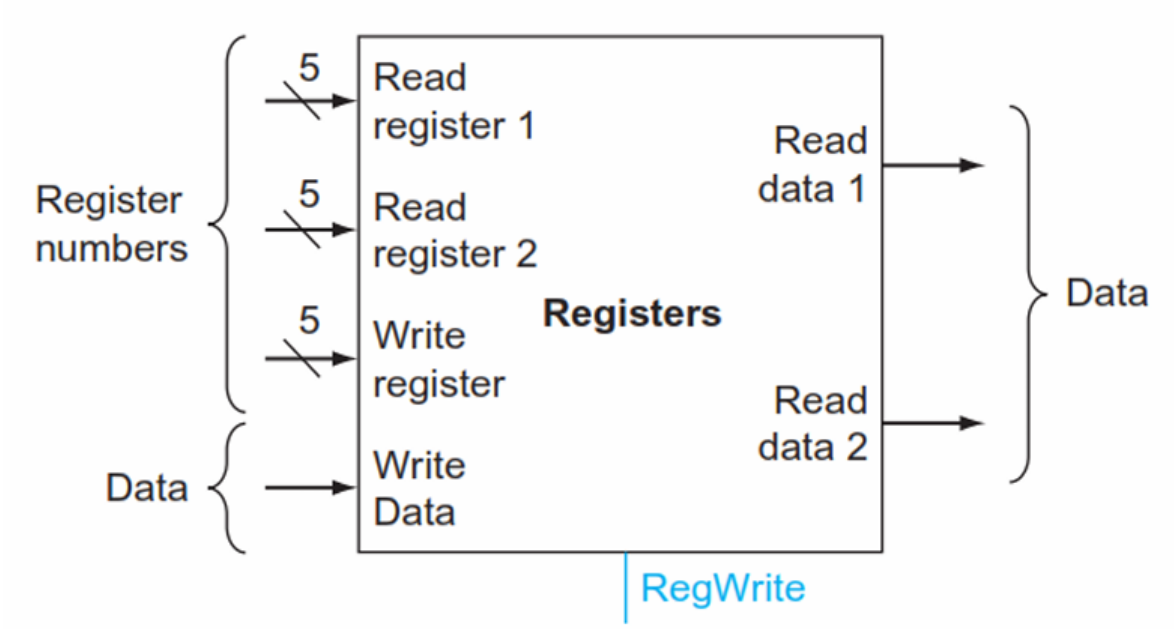

Figure 1: Register File for MIPS Architecture

2 Register File – Source Code

```
1  LIBRARY IEEE;
2  USE IEEE.std_logic_1164.all;
3  USE IEEE.numeric_std.all;
```

Code Snippet 1: `regFile:` Used Libraries

```
1  ENTITY regFile IS
2    -- numerical constants are defined using generics (to avoid magic nums)
3    GENERIC(
4      addressBits_GEN : INTEGER := 5;  -- 0 -> 31
5      dataLength_GEN  : INTEGER := 32; -- 32-bit data
6      numOfReg_GEN    : INTEGER := 32  -- number of registers = 2 ^
           addressBits
7    );
8    PORT(
9      -- input signals
10     RsSel_IN  : IN  STD_LOGIC_VECTOR((addressBits_GEN - 1) DOWNTO 0); --
           read reg 1
11     RtSel_IN  : IN  STD_LOGIC_VECTOR((addressBits_GEN - 1) DOWNTO 0); --
           read reg 2
12     RdSel_IN  : IN  STD_LOGIC_VECTOR((addressBits_GEN - 1) DOWNTO 0); --
           write reg
13     DataW_IN  : IN  STD_LOGIC_VECTOR((dataLength_GEN - 1) DOWNTO 0); --
           write this data in Rd
14
15     -- clock + control signals
16     WrtEN     : IN  STD_LOGIC; -- write enable
17     CLK       : IN  STD_LOGIC; -- clock (write on falling and read on
           rising)
18
19     -- output signals
20     DataRs_OUT : OUT STD_LOGIC_VECTOR((dataLength_GEN - 1) DOWNTO 0); --
           data written in Rs
21     DataRt_OUT : OUT STD_LOGIC_VECTOR((dataLength_GEN - 1) DOWNTO 0)  --
           data written in Rd
22   );
23
24 END regFile;
```

Code Snippet 2: `regFile:` Entity Definition

```
1  ARCHITECTURE regFile_ARCH OF regFile IS
2    -- define a type as a 1D array of numOfReg elements
3    TYPE regFile_TYP IS ARRAY(0 TO (numOfReg_GEN - 1)) OF
       STD_LOGIC_VECTOR((dataLength_GEN - 1) DOWNTO 0);
4    -- define registerFile signal to hold register data as a 1D array of
       numOfReg register elements
5    SIGNAL registerFile_SIG : regFile_TYP := (
```

Code Snippet 3: `regFile:` Architecture Definition—Array Type Definition

```
                X"00000000",
                X"00000001",
                X"00000002",
                X"00000003",
                X"00000004",
                X"00000005",
                X"00000006",
                X"00000007",
                X"00000008",
                X"00000009",
                X"0000000A",
                X"0000000B",
                X"0000000C",
                X"0000000D",
                X"0000000E",
                X"0000000F",
                X"00000010",
                X"00000011",
                X"00000012",
                X"00000013",
                X"00000014",
                X"00000015",
                X"00000016",
                X"00000017",
                X"00000018",
                X"00000019",
                X"0000001A",
                X"0000001B",
                X"0000001C",
                X"0000001D",
                X"0000001E",
                X"0000001F"
```

Code Snippet 4: `regFile:` Architecture Definition—Register File Initial Data Values

```
writeProcess: -- write data
PROCESS(CLK) IS
BEGIN
  IF (FALLING_EDGE(CLK) AND WrtEN = '1') THEN -- write data in Rd on
     falling edge && @ WrtEN = 1
    IF (RdSel_IN /= "00000") THEN -- if destination is NOT reg_zero,
       write
      registerFile_SIG(TO_INTEGER(UNSIGNED(RdSel_IN))) <= DataW_IN;
    END IF;
  END IF;
END PROCESS writeProcess;
```

Code Snippet 5: `regFile:` Architecture Definition—Register File Write Process

```
1   readProcess: -- read data
2   PROCESS IS
3   BEGIN
4     -- read Rs && Rt
5     DataRs_OUT <= registerFile_SIG(TO_INTEGER(UNSIGNED(RsSel_IN)));
6     DataRt_OUT <= registerFile_SIG(TO_INTEGER(UNSIGNED(RtSel_IN)));
7     WAIT FOR 1 ps; -- has to wait to avoid infinite loop warning
8   END PROCESS;
```

Code Snippet 6: `regFile:` Architecture Definition—Register File Read Process

3 Register File – TestBench Code

A simple testbench was designed to test the register file by changing the inputs while monitoring the outputs to check their validity. A total of three test cases were created:

1. Test Case 1: Read $Rs = \$7$ and $Rt = \$8$, write in $Rd = \$7$ a value of X"AAAAAAAA". This test takes 1 clock cycle at asserted write enable. This test's main purpose is to make sure the register file reads and writes successfully in the same clock cycle.

2. Test Case 2: Read $Rs = \$0$ and $Rt = \$9$, write in $Rd = \$0$ a value of X"ABCDDCBA". This test takes 1 clock cycle at asserted write enable. This test's main purpose is to make sure that no data can be written in \$0.

3. Test Case 3: Read $Rs = \$12$ and $Rt = \$0$, write in $Rd = \$12$ a value of X"FFFFFFFF". This test takes 1 clock cycles at deasserted write enable. This test's main purpose is to see that writing does not happen if write enable was deasserted.

```
1  LIBRARY IEEE;
2  USE IEEE.std_logic_1164.all;
3  USE IEEE.numeric_std.all;
4
5  -- this is a testbench for the 32x32-bit register file for a 32-bit MIPS
       microprocessor defined in regFile.vhd
6
7  ENTITY testBench IS
8    -- nothing to see here
9  END testBench;
```

Code Snippet 7: `testBench:` Libraries and Entity

```
1   CONSTANT clkPeriod_CON    : TIME    := 100 ps;
2   -- define number of address bits
3   CONSTANT addressBits_CON  : INTEGER := 5;
4   -- define number of 32-bit data
5   CONSTANT dataLength_CON   : INTEGER := 32;
6   -- number of registers = 2 ^ addressBits
7   CONSTANT numOfReg_CON     : INTEGER := 32;
```

Code Snippet 8: `testBench:` Defined Constants

```
1  -- define regFile as a component
2  COMPONENT regFile
3    PORT(
4    -- input ports
5    RsSel_IN  : IN  STD_LOGIC_VECTOR((addressBits_CON - 1) DOWNTO 0); --
        read reg 1
6    RtSel_IN  : IN  STD_LOGIC_VECTOR((addressBits_CON - 1) DOWNTO 0); --
        read reg 2
7    RdSel_IN  : IN  STD_LOGIC_VECTOR((addressBits_CON - 1) DOWNTO 0); --
        write reg
8    DataW_IN  : IN  STD_LOGIC_VECTOR((dataLength_CON - 1) DOWNTO 0); --
        write this data in Rd
9
10   -- clock && necessary control ports(s)
11   WrtEN     : IN  STD_LOGIC; -- write enable
12   CLK       : IN  STD_LOGIC; -- clock (write on falling and read on
        rising)
13
14   -- output ports
15   DataRs_OUT : OUT STD_LOGIC_VECTOR((dataLength_CON - 1) DOWNTO 0); --
        data written in Rs
16   DataRt_OUT : OUT STD_LOGIC_VECTOR((dataLength_CON - 1) DOWNTO 0)  --
        data written in Rd
17 );
18 END COMPONENT;
```

Code Snippet 9: `testBench:` Define `regFile:` as a Component

```
1  -- input signals (initialized to avoid 'numeric_std.to_integer:
      metavalue detected' error)
2  SIGNAL RsSel_IN_SIG : STD_LOGIC_VECTOR((addressBits_CON - 1) DOWNTO 0)
      := (OTHERS => '0');
3  SIGNAL RtSel_IN_SIG : STD_LOGIC_VECTOR((addressBits_CON - 1) DOWNTO 0)
      := (OTHERS => '0');
4  SIGNAL RdSel_IN_SIG : STD_LOGIC_VECTOR((addressBits_CON - 1) DOWNTO 0)
      := (OTHERS => '0');
5  SIGNAL DataW_IN_SIG : STD_LOGIC_VECTOR((dataLength_CON - 1) DOWNTO 0)
      := (OTHERS => '0');
6  -- control && clk signal
7  SIGNAL WrtEN : STD_LOGIC := '1'; -- WrtEN has to have an initial value
8  SIGNAL CLK   : STD_LOGIC := '1'; -- clk has to have an initial value
9  -- output signals
10 SIGNAL DataRs_OUT_SIG : STD_LOGIC_VECTOR((dataLength_CON - 1) DOWNTO
      0) := (OTHERS => '0');
11 SIGNAL DataRt_OUT_SIG : STD_LOGIC_VECTOR((dataLength_CON - 1) DOWNTO
      0) := (OTHERS => '0');
```

Code Snippet 10: `testBench:` Defined Signals

```
Test1:
regFile PORT MAP (
  RsSel_IN  => RsSel_IN_SIG,
  RtSel_IN  => RtSel_IN_SIG,
  RdSel_IN  => RdSel_IN_SIG,
  DataW_IN  => DataW_IN_SIG,
  WrtEN     => WrtEN,
  CLK   => CLK,
  DataRs_OUT  => DataRs_OUT_SIG,
  DataRt_OUT  => DataRt_OUT_SIG
);
```

Code Snippet 11: `testBench:` Port Map

```
-- testBench process for CLK
clkCycleProcess :
PROCESS(CLK)
BEGIN
  CLK <= NOT CLK AFTER (clkPeriod_CON / 2);
END PROCESS clkCycleProcess;
```

Code Snippet 12: `testBench:` Process for Clock

```
-- testBench process for test cases
testCaseProcess:
PROCESS
  -- define procedure to set input signals values and wait
  PROCEDURE regFileInputsTest(
    CONSTANT RsSel_IN_PROC      : STD_LOGIC_VECTOR((addressBits_CON -
       1) DOWNTO 0);
    CONSTANT RtSel_IN_PROC      : STD_LOGIC_VECTOR((addressBits_CON -
       1) DOWNTO 0);
    CONSTANT RdSel_IN_PROC      : STD_LOGIC_VECTOR((addressBits_CON -
       1) DOWNTO 0);
    CONSTANT DataW_IN_PROC      : STD_LOGIC_VECTOR((dataLength_CON -
       1) DOWNTO 0);
    CONSTANT WrtEN_PROC     : STD_LOGIC;
    CONSTANT testCaseDuration_PROC    : TIME
  ) IS
  BEGIN
    -- set inputs by passed values
    RsSel_IN_SIG <= RsSel_IN_PROC;
    RtSel_IN_SIG <= RtSel_IN_PROC;
    RdSel_IN_SIG <= RdSel_IN_PROC;
    DataW_IN_SIG <= DataW_IN_PROC;
    WrtEN      <= WrtEN_PROC;
    -- wait to view results of this test case
    WAIT FOR testCaseDuration_PROC;
  END PROCEDURE regFileInputsTest;
BEGIN
```

Code Snippet 13: `testBench:` Procedure to Change Input Values Inside testCaseProcess

```
1     -- Test case 1:
2     ---- -> read rs && rt && try to write a diff value in rd
3     ---- ->
         Rs=7,Rt=8,Rd=7,DataWrite=X"AAAAAAAA",WrtEN='1',Duration=100ps (1
         clk cycle)
4     regFileInputsTest("00111", "01000", "00111", X"AAAAAAAA", '1',
         clkPeriod_CON);
5     -- Test case 2:
6     ---- -> read $0 as rs and read $9 as rt then try to write some value
         in rs ($0 value should remain 0)
7     ---- ->
         Rs=0,Rt=9,Rd=0,DataWrite=X"ABCDDCBA",WrtEN='1',Duration=100ps (1
         clk cycle)
8     regFileInputsTest("00000", "01001", "00000", X"ABCDDCBA", '1',
         clkPeriod_CON);
9     -- Test case 3:
10    ---- -> read what is in rs and write in rd but WrtEN will be low
11    ---- ->
         Rs=12,Rt=0,Rd=12,DataWrite=X"FFFFFFFF",WrtEN='0',Duration=100ps
         (1 clk cycle)
12    regFileInputsTest("01100", "00000", "01100", X"FFFFFFFF", '0',
         clkPeriod_CON);
13  END PROCESS testCaseProcess;
```

Code Snippet 14: `testBench:` Process to Change Inputs According to Test Cases

4 Register File – Simulation

The following simulation results were produced on Intel's ModelSim. The aforementioned test cases were used to produce this output.

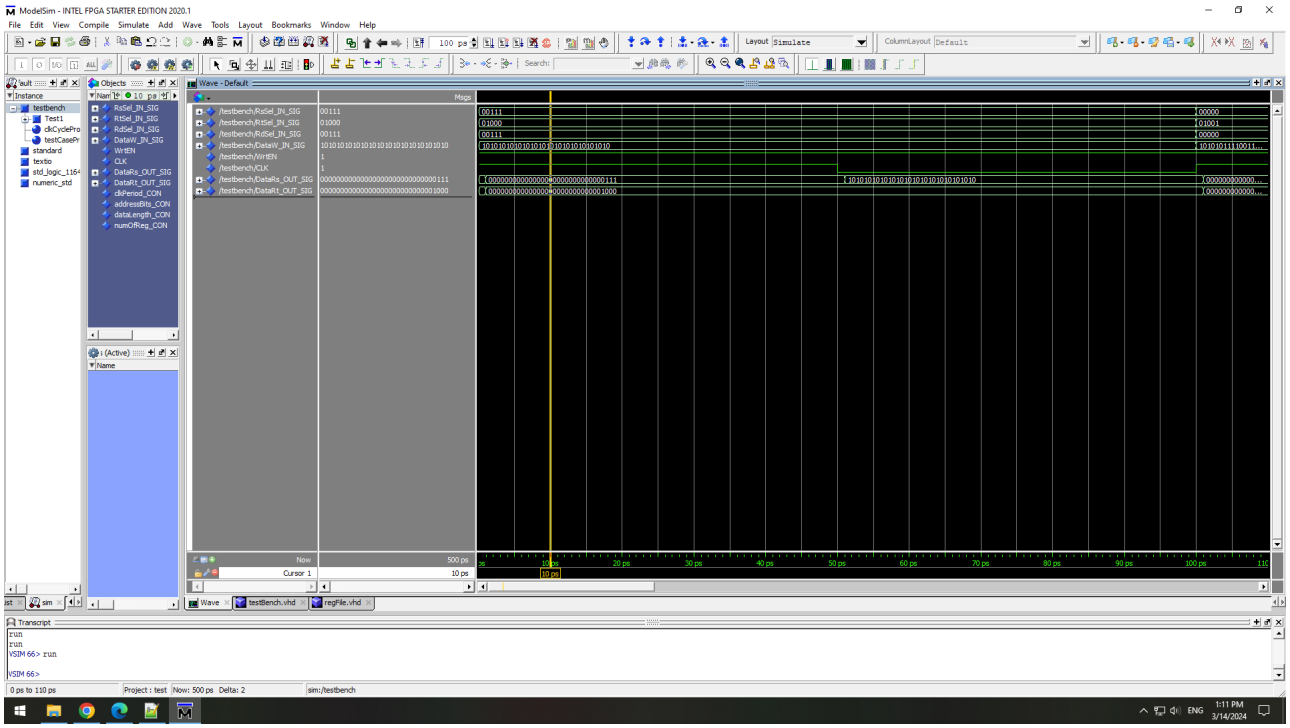

Figure 2: Test Case 1: Input Values Assigned

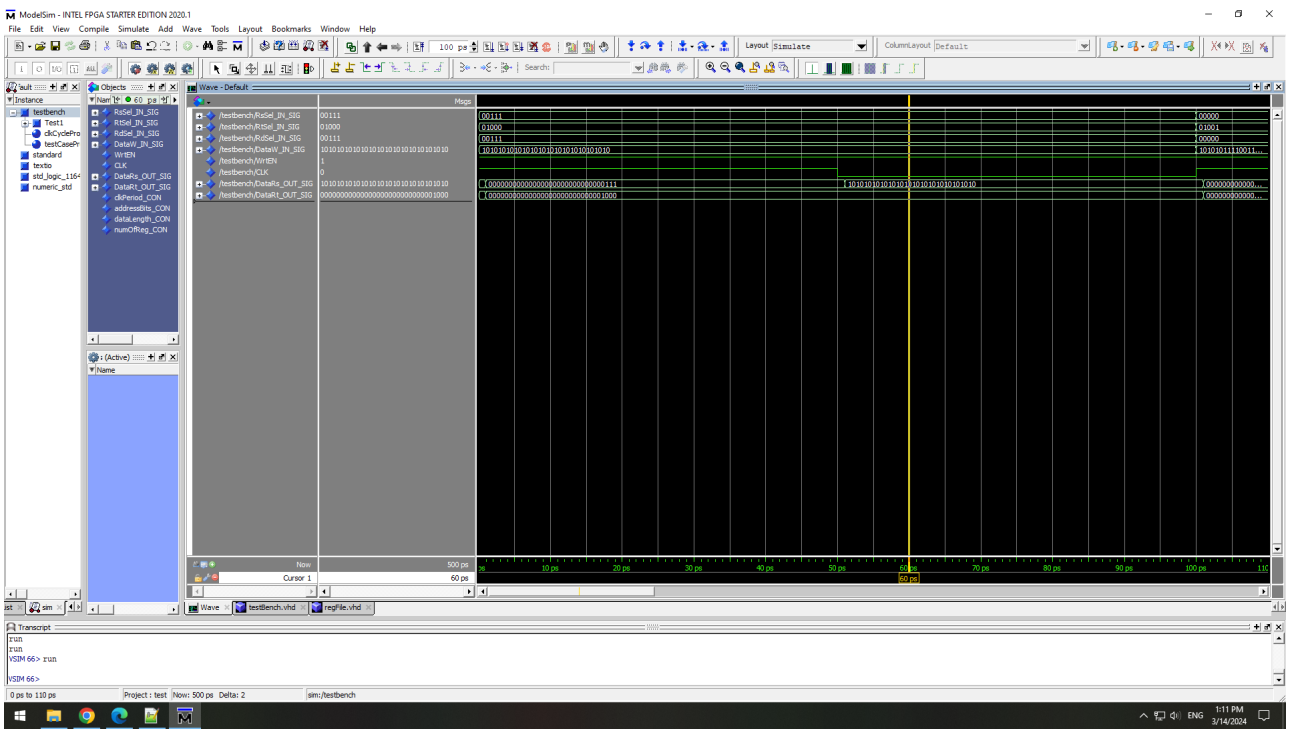

Figure 3: Test Case 1: Output Values Observed—Read in First Half Cycle and Write in Second Half Cycle

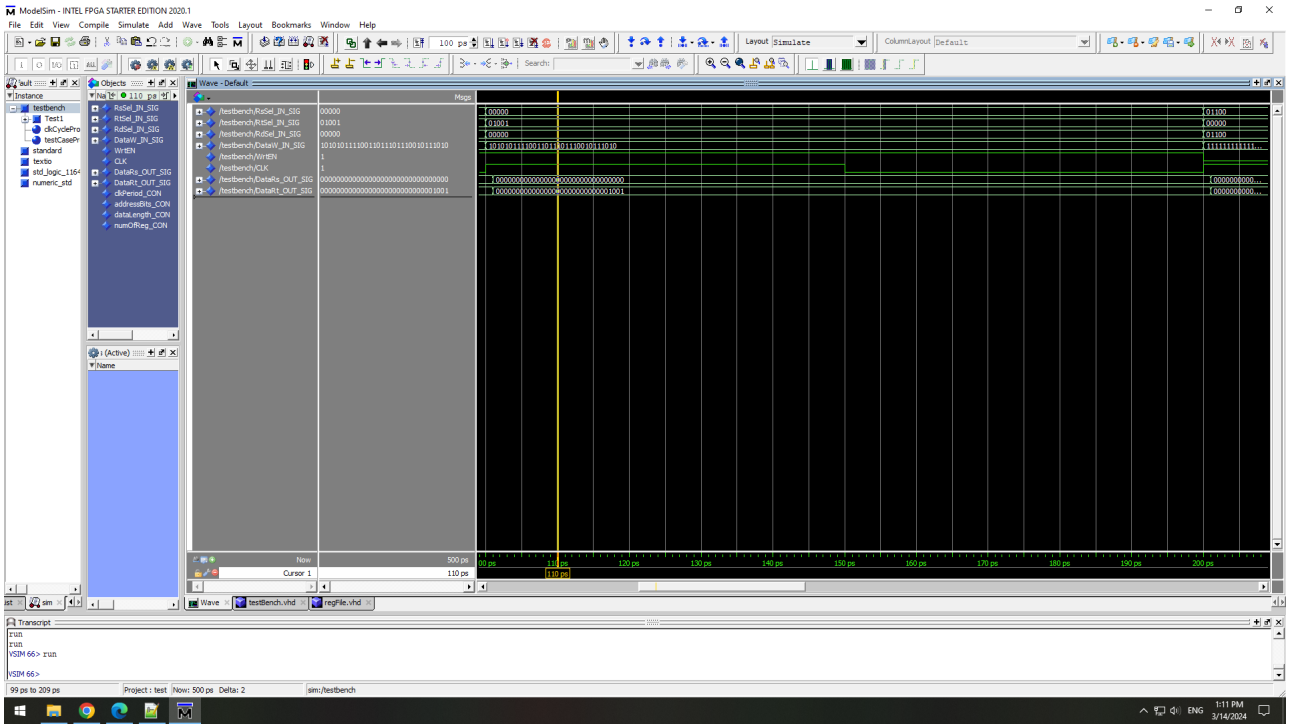

Figure 4: Test Case 2: Input Values Assigned

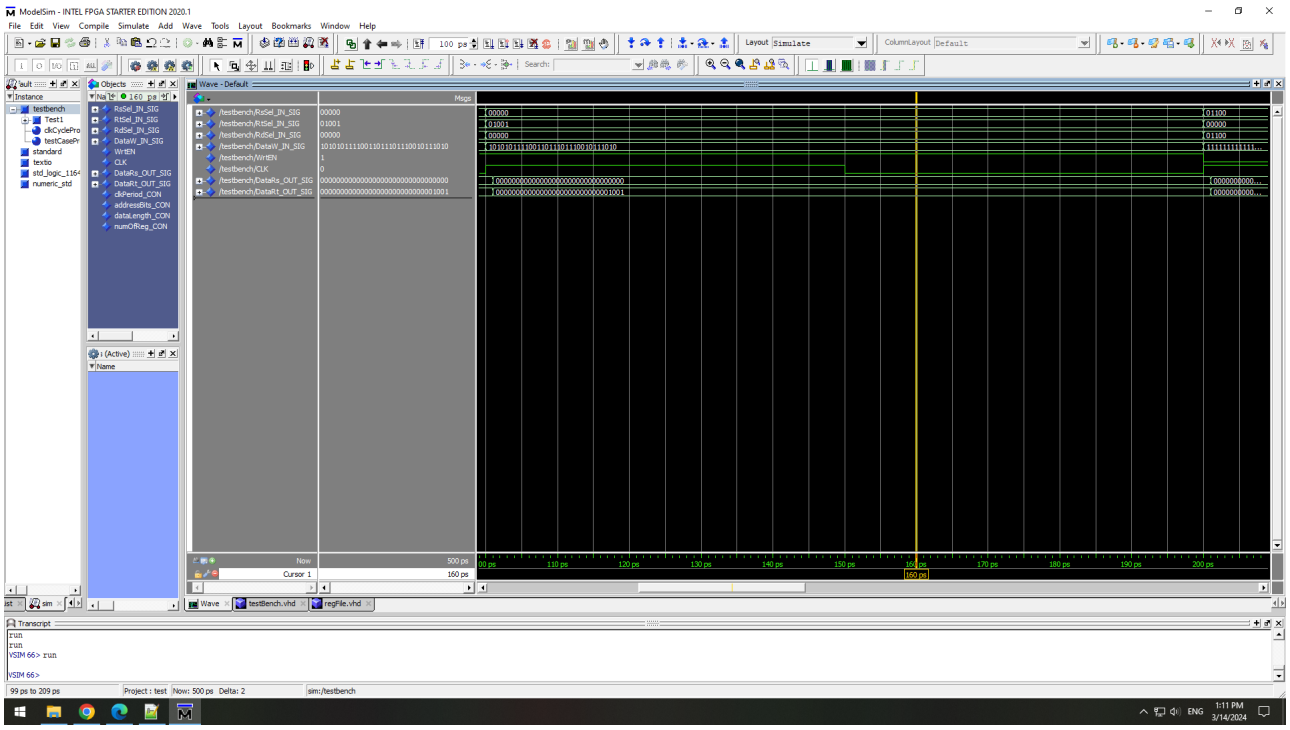

Figure 5: Test Case 2: Output Values Observed—No Change in Value of $0

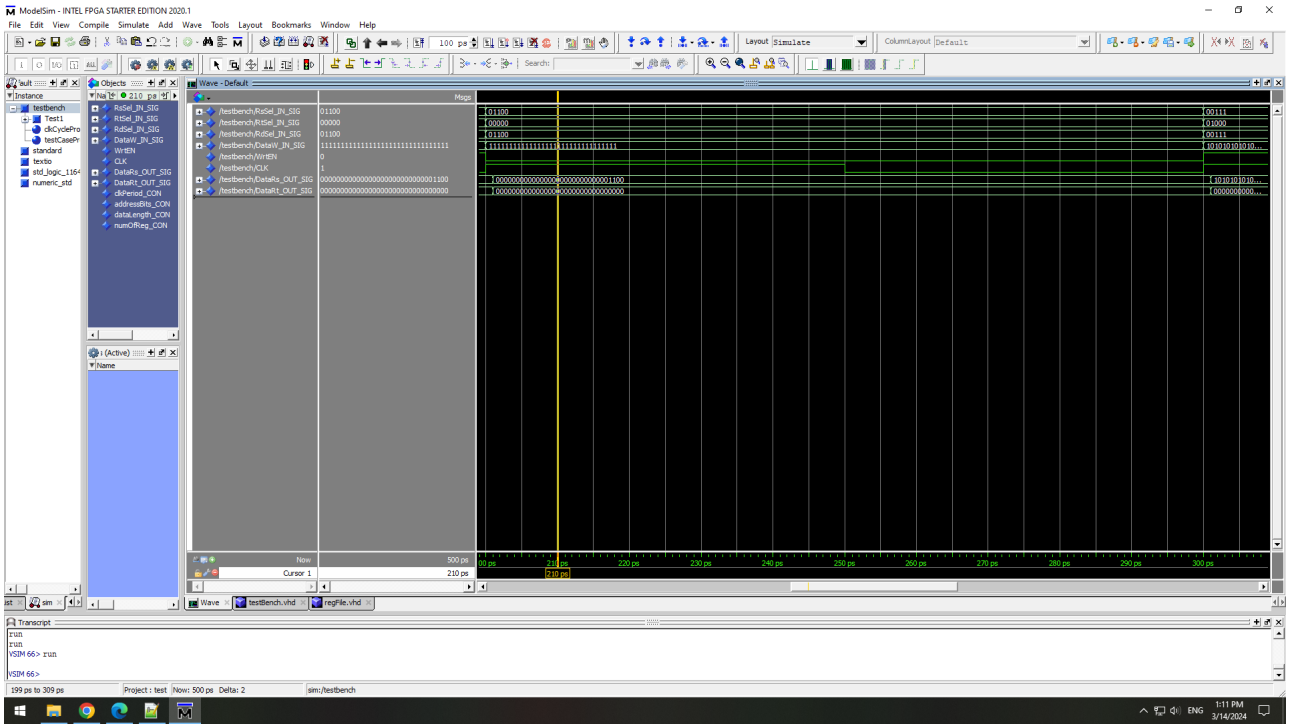

Figure 6: Test Case 3: Input Values Assigned

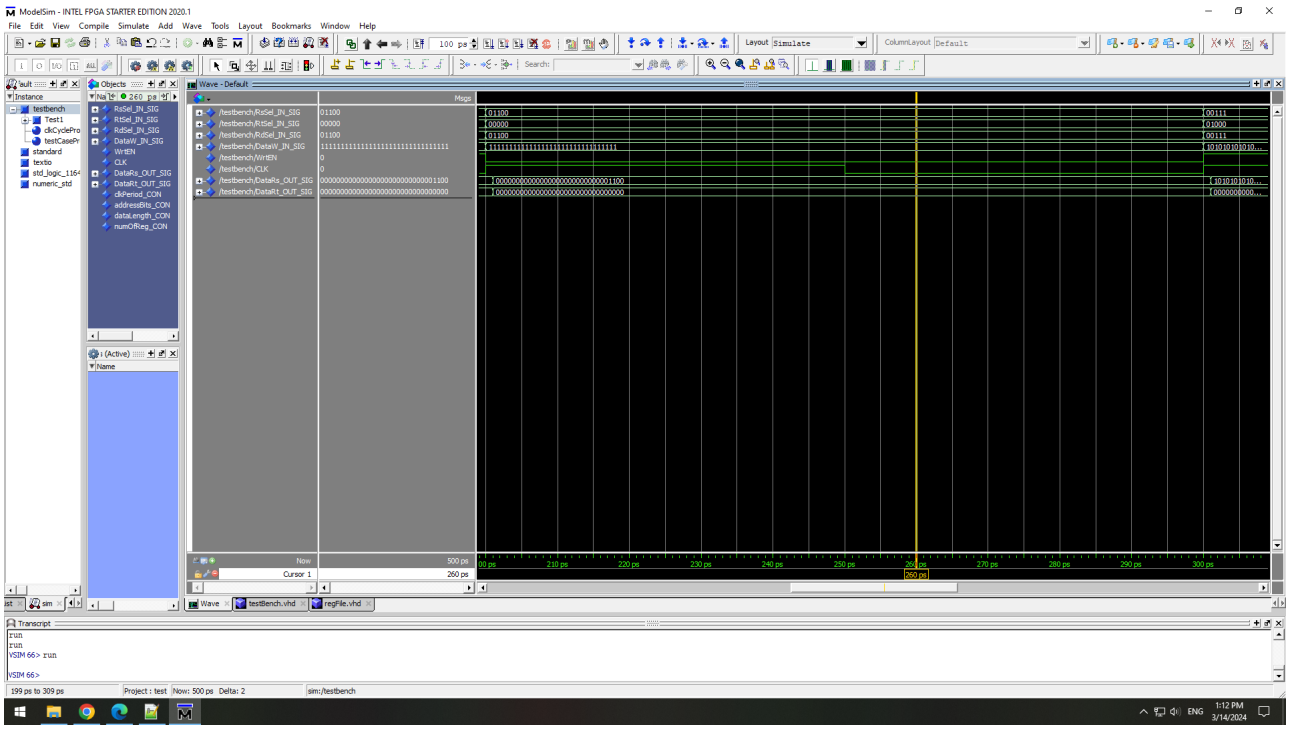

Figure 7: Test Case 3: Output Values Observed—No Change in $12 At Deasserted WrtEN