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 11. All code and simulation/screenshots files can be found in the project's Github repository.

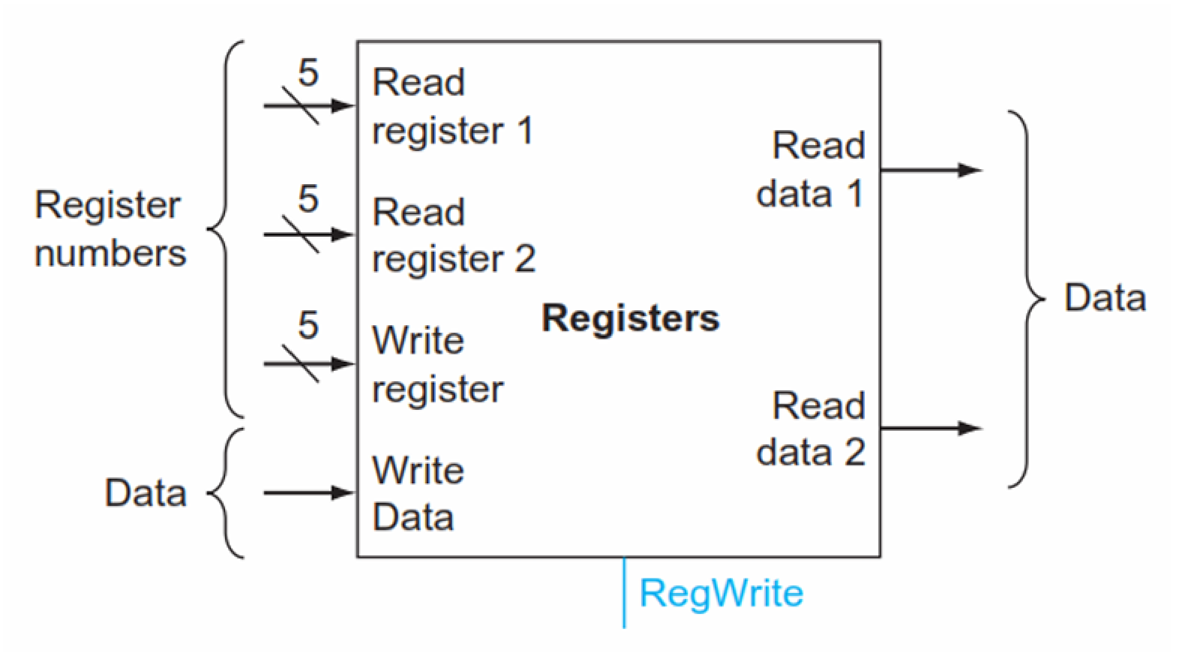


Figure 1: Register File for MIPS Architecture

2 Register File – Source Code

```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
```

Code Snippet 1: `regFile:` Used Libraries

```vhdl
ENTITY regFile IS
  -- numerical constants are defined using generics (to avoid magic nums)
  GENERIC(
    addressBits_GEN : INTEGER := 5;  -- 0 -> 31
    dataLength_GEN  : INTEGER := 32; -- 32-bit data
    numOfReg_GEN    : INTEGER := 32  -- number of registers = 2 ^ addressBits
  );
  PORT(
    -- input signals
    RsSel_IN   : IN  STD_LOGIC_VECTOR((addressBits_GEN - 1) DOWNTO 0); -- read reg 1
    RtSel_IN   : IN  STD_LOGIC_VECTOR((addressBits_GEN - 1) DOWNTO 0); -- read reg 2
    RdSel_IN   : IN  STD_LOGIC_VECTOR((addressBits_GEN - 1) DOWNTO 0); -- write reg
    DataW_IN   : IN  STD_LOGIC_VECTOR((dataLength_GEN - 1) DOWNTO 0); -- write this data in Rd

    -- clock + control signals
    WrtEN      : IN  STD_LOGIC; -- write enable
    CLK        : IN  STD_LOGIC; -- clock (write on falling and read on rising)

    -- output signals
    DataRs_OUT : OUT STD_LOGIC_VECTOR((dataLength_GEN - 1) DOWNTO 0); -- data written in Rs
    DataRt_OUT : OUT STD_LOGIC_VECTOR((dataLength_GEN - 1) DOWNTO 0)  -- data written in Rd
  );

END regFile;
```

Code Snippet 2: `regFile:` Entity Definition

```vhdl
ARCHITECTURE regFile_ARCH OF regFile IS
  -- define a type as a 1D array of numOfReg elements
  TYPE regFile_TYP IS ARRAY(0 TO (numOfReg_GEN - 1)) OF
     STD_LOGIC_VECTOR((dataLength_GEN - 1) DOWNTO 0);
```

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

```vhdl
    -- define registerFile variable to hold register data as a 1D array of numOfReg register elements
    VARIABLE registerFile_VAR : regFile_TYP := (
              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

```vhdl
    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_VAR(TO_INTEGER(UNSIGNED(RdSel_IN))) := DataW_IN;
      END IF;
    END IF;
    -- read Rs && Rt
    IF (RISING_EDGE(CLK)) THEN
      DataRs_OUT <= registerFile_VAR(TO_INTEGER(UNSIGNED(RsSel_IN)));
      DataRt_OUT <= registerFile_VAR(TO_INTEGER(UNSIGNED(RtSel_IN)));
    END IF;
```

Code Snippet 5: `regFile:` Architecture Definition—Register File 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 4 test cases were created:

1. Test Case 1: Read Rs = \$7 and Rt = \$8, write in Rd = \$9 a value of X"$AAAAAAAA$". This test takes 2 clock cycles at asserted write enable. This test's main purpose is to make sure the register file reads and writes successfully.

2. Test Case 2: Read Rs = \$0 and Rt = \$9 (Rd in previous test), write in Rd = \$0 a value of X"$ABCDDCBA$". This test takes 2 clock cycles at asserted write enable. This test's main purpose is to make sure the value written in \$9 was written successfully and that one cannot write in \$0.

3. Test Case 3: Read Rs = \$12 and Rt = \$0 (Rd in previous test), write in Rd = \$12 a value of X"$EEEEEEEE$". This test takes 2 clock cycles at asserted write enable. This test's main purpose is to make sure the value written in \$0 is still 0 and that one can read in the first half of a clock cycle and write in the second half successfully.

4. Test Case 4: Read Rs = \$12 and Rt = \$0 (Rd in previous test), write in Rd = \$12 a value of X"$EEEEEEEE$". This test takes 2 clock cycles at deasserted write enable. This test's main purpose is to see that writing does not happen if write enable was deasserted.

```vhdl
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.numeric_std.all;
USE IEEE.math_real.all;

-- this is a testbench for the 32x32-bit register file for a 32-bit MIPS
   microprocessor defined in regFile.vhd

ENTITY testBench IS
  -- nothing to see here
END testBench;
```

Code Snippet 6: `testBench:` Libraries and Entity

```vhdl
  -- declare constant clock period
  CONSTANT clkPeriod_CON    : TIME    := 100 ps;
  -- define number of address bits
  CONSTANT addressBits_CON  : INTEGER := 5;
  -- define number of 32-bit data
  CONSTANT dataLength_CON   : INTEGER := 32;
  -- number of registers = 2 ^ addressBits
  CONSTANT numOfReg_CON     : INTEGER := 32;
```

Code Snippet 7: `testBench:` Defined Constants

```vhdl
-- define regFile as a component
COMPONENT regFile
  PORT(
  -- input ports
  RsSel_IN  : IN  STD_LOGIC_VECTOR((addressBits_CON - 1) DOWNTO 0); -- read reg 1
  RtSel_IN  : IN  STD_LOGIC_VECTOR((addressBits_CON - 1) DOWNTO 0); -- read reg 2
  RdSel_IN  : IN  STD_LOGIC_VECTOR((addressBits_CON - 1) DOWNTO 0); -- write reg
  DataW_IN  : IN  STD_LOGIC_VECTOR((dataLength_CON - 1) DOWNTO 0); -- write this data in Rd

  -- clock && necessary control ports(s)
  WrtEN     : IN  STD_LOGIC; -- write enable
  CLK       : IN  STD_LOGIC; -- clock (write on falling and read on rising)

  -- output ports
  DataRs_OUT : OUT STD_LOGIC_VECTOR((dataLength_CON - 1) DOWNTO 0); -- data written in Rs
  DataRt_OUT : OUT STD_LOGIC_VECTOR((dataLength_CON - 1) DOWNTO 0)  -- data written in Rd
);
END COMPONENT;
```

Code Snippet 8: `testBench:` Define `regFile:` as a Component

```vhdl
-- input signals
SIGNAL RsSel_IN_SIG : STD_LOGIC_VECTOR((addressBits_CON - 1) DOWNTO 0);
SIGNAL RtSel_IN_SIG : STD_LOGIC_VECTOR((addressBits_CON - 1) DOWNTO 0);
SIGNAL RdSel_IN_SIG : STD_LOGIC_VECTOR((addressBits_CON - 1) DOWNTO 0);
SIGNAL DataW_IN_SIG : STD_LOGIC_VECTOR((dataLength_CON - 1) DOWNTO 0);
-- control && clk signal
SIGNAL WrtEN : STD_LOGIC := '1'; -- WrtEN has to have an initial value
SIGNAL CLK   : STD_LOGIC := '0'; -- clk has to have an initial value
-- output signals
SIGNAL DataRs_OUT_SIG : STD_LOGIC_VECTOR((dataLength_CON - 1) DOWNTO 0);
SIGNAL DataRt_OUT_SIG : STD_LOGIC_VECTOR((dataLength_CON - 1) DOWNTO 0);
```

Code Snippet 9: `testBench:` Defined Signals

```vhdl
-- mapping signals to component ports
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 10: `testBench:` Port Map

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

Code Snippet 11: `testBench:` Process for Clock

```vhdl
-- 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;
```

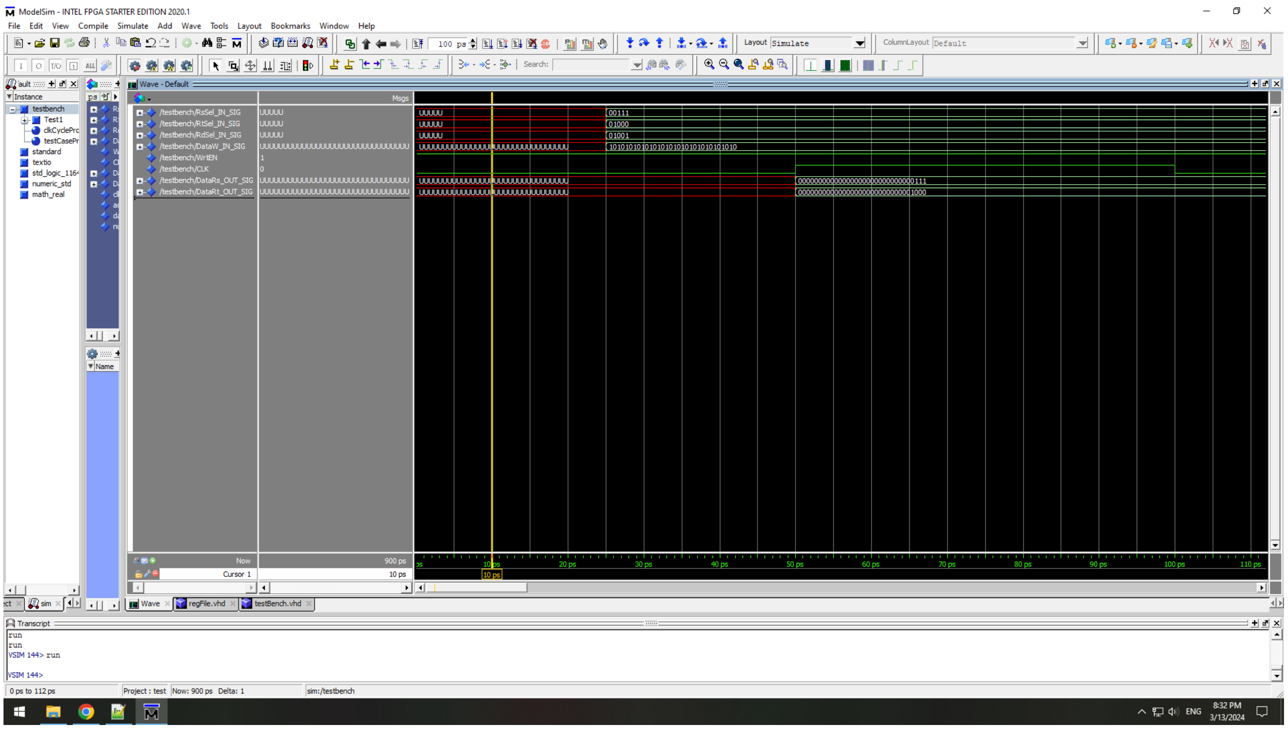
Code Snippet 12: `testBench:` Procedure to Change Input Values Inside testCaseProcess

```
BEGIN
  -- Initial wait duration before doing anything
  WAIT FOR clkPeriod_CON / 4;
  -- Test case 1:
  ---- -> read rs && rt && try to write a diff value in rd (will not
     write successfully until WrtEN is 1)
  ---- ->
     Rs=7,Rt=8,Rd=9,DataWrite=X"AAAAAAAA",WrtEN='1',Duration=200ps (4
     clk cycles)
  regFileInputsTest("00111","01000","01001",X"AAAAAAAA",'1',200 ps);
  -- Test case 2:
  ---- -> read $0 as rs and read $9 as rt then try to write some value
     in rs ($0 value should remain 0)
  ---- ->
     Rs=0,Rt=9,Rd=0,DataWrite=X"ABCDDCBA",WrtEN='1',Duration=200ps (4
     clk cycles)
  regFileInputsTest("00000","01001","00000",X"ABCDDCBA",'1',200 ps);
  -- Test case 3:
  ---- -> read what is in rs and write in rd (first clock cycle should
     show read and second should show write)
  ---- ->
     Rs=12,Rt=0,Rd=12,DataWrite=X"EEEEEEEE",WrtEN='1',Duration=200ps
     (4 clk cycles)
  regFileInputsTest("01100","00000","01100",X"EEEEEEEE",'1',200 ps);
  -- Test case 4:
  ---- -> repeat test case 3 but WrtEN will be low
  ---- ->
     Rs=12,Rt=0,Rd=12,DataWrite=X"FFFFFFFF",WrtEN='0',Duration=200ps
     (4 clk cycles)
  regFileInputsTest("01100","00000","01100",X"FFFFFFFF",'0',200 ps);
END PROCESS testCaseProcess;
```

Code Snippet 13: `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.

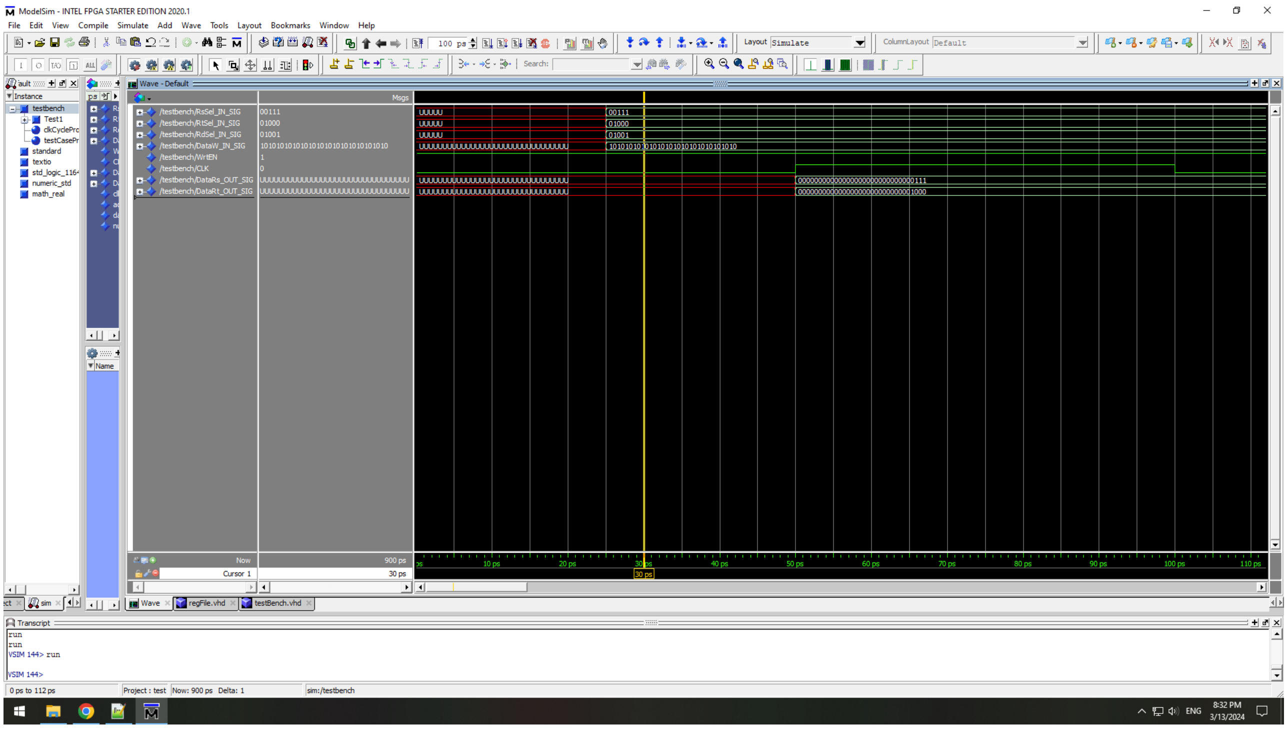


```
/testbench/RsSel_IN_SIG      UUUUU
/testbench/RtSel_IN_SIG      UUUUU
/testbench/RdSel_IN_SIG      UUUUU
/testbench/DataW_IN_SIG      UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
/testbench/WrtEN             1
/testbench/CLK               0
/testbench/DataRs_OUT_SIG    UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
/testbench/DataRt_OUT_SIG    UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU

Now 900 ps
Cursor 1 10 ps
```

```
run
run
VSIM 144> run

VSIM 144>
```

ModelSim - INTEL FPGA STARTER EDITION 2020.1
File Edit View Compile Simulate Add Wave Tools Layout Bookmarks Window Help
Wave - Default
00111
01000
01001
10101010101010101010101010101010
00000000000000000000000000000111
00000000000000000000000000001000
0 ps to 112 ps
Project : test Now: 900 ps Delta: 1
sim:/testbench

Figure 2: Initial Wait Period Before Tests

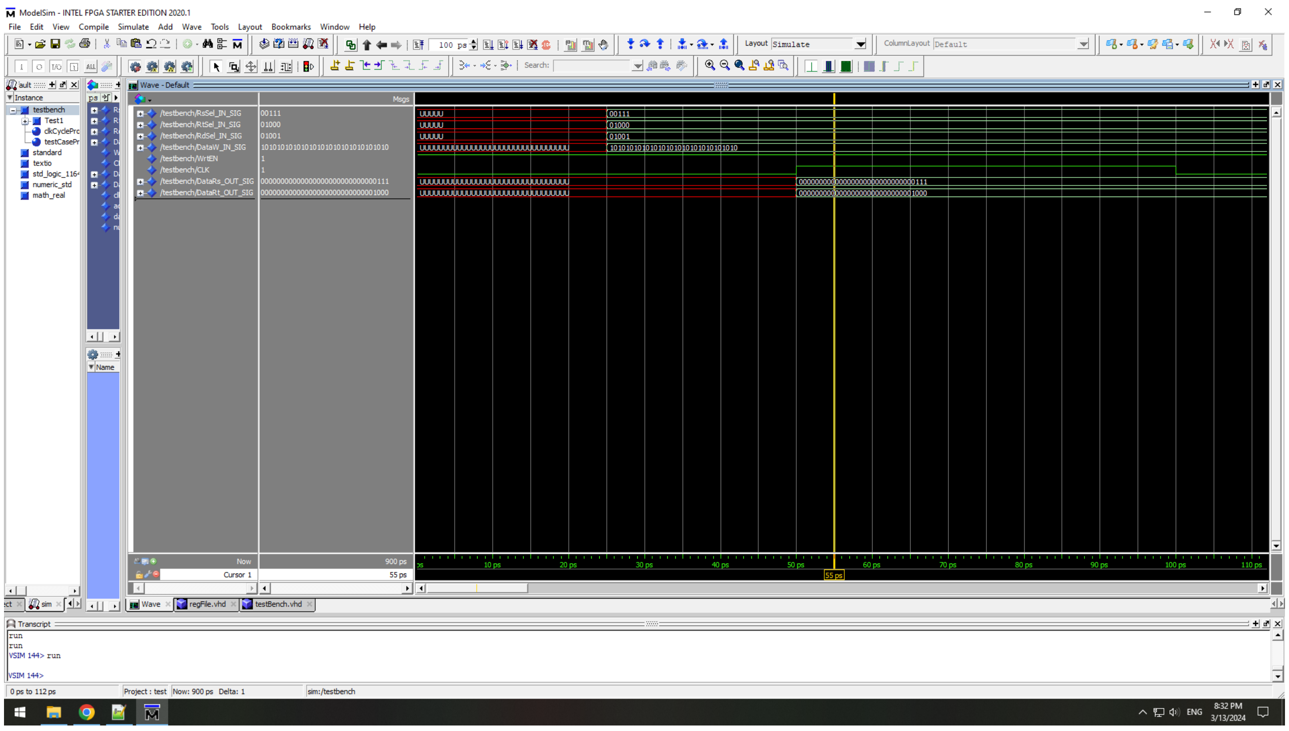


```
/testbench/RsSel_IN_SIG      00111
/testbench/RtSel_IN_SIG      01000
/testbench/RdSel_IN_SIG      01001
/testbench/DataW_IN_SIG      10101010101010101010101010101010
/testbench/WrtEN             1
/testbench/CLK               0
/testbench/DataRs_OUT_SIG    UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
/testbench/DataRt_OUT_SIG    UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU

Now 900 ps
Cursor 1 30 ps
```

```
run
run
VSIM 144> run

VSIM 144>
```

ModelSim - INTEL FPGA STARTER EDITION 2020.1
File Edit View Compile Simulate Add Wave Tools Layout Bookmarks Window Help
Wave - Default
00111
01000
01001
10101010101010101010101010101010
00000000000000000000000000000111
00000000000000000000000000001000
0 ps to 112 ps
Project : test Now: 900 ps Delta: 1
sim:/testbench

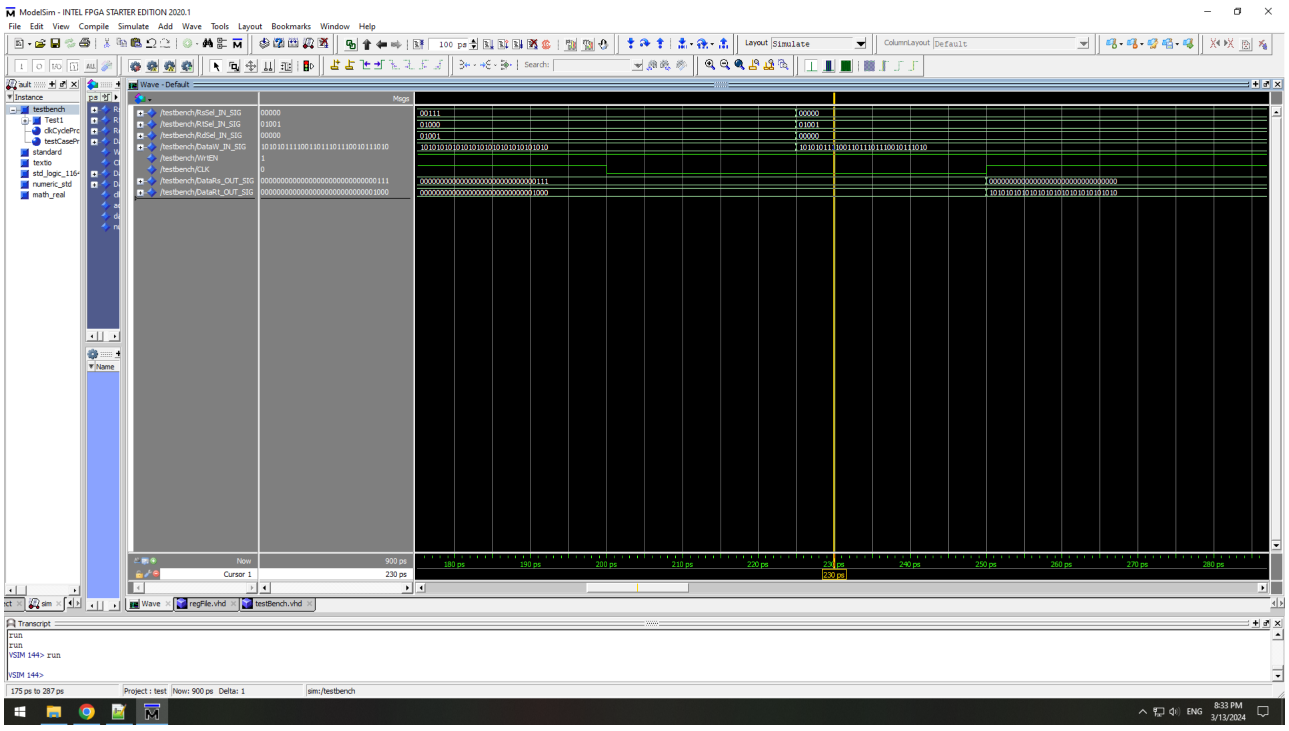
Figure 3: Test Case 1: Input Values Assigned



```
/testbench/RsSel_IN_SIG      00111
/testbench/RtSel_IN_SIG      01000
/testbench/RdSel_IN_SIG      01001
/testbench/DataW_IN_SIG      10101010101010101010101010101010
/testbench/WrtEN             1
/testbench/CLK               1
/testbench/DataRs_OUT_SIG    00000000000000000000000000000111
/testbench/DataRt_OUT_SIG    00000000000000000000000000001000

Now       900 ps
Cursor 1  55 ps
```

```
run
run
VSIM 144> run

VSIM 144>
```

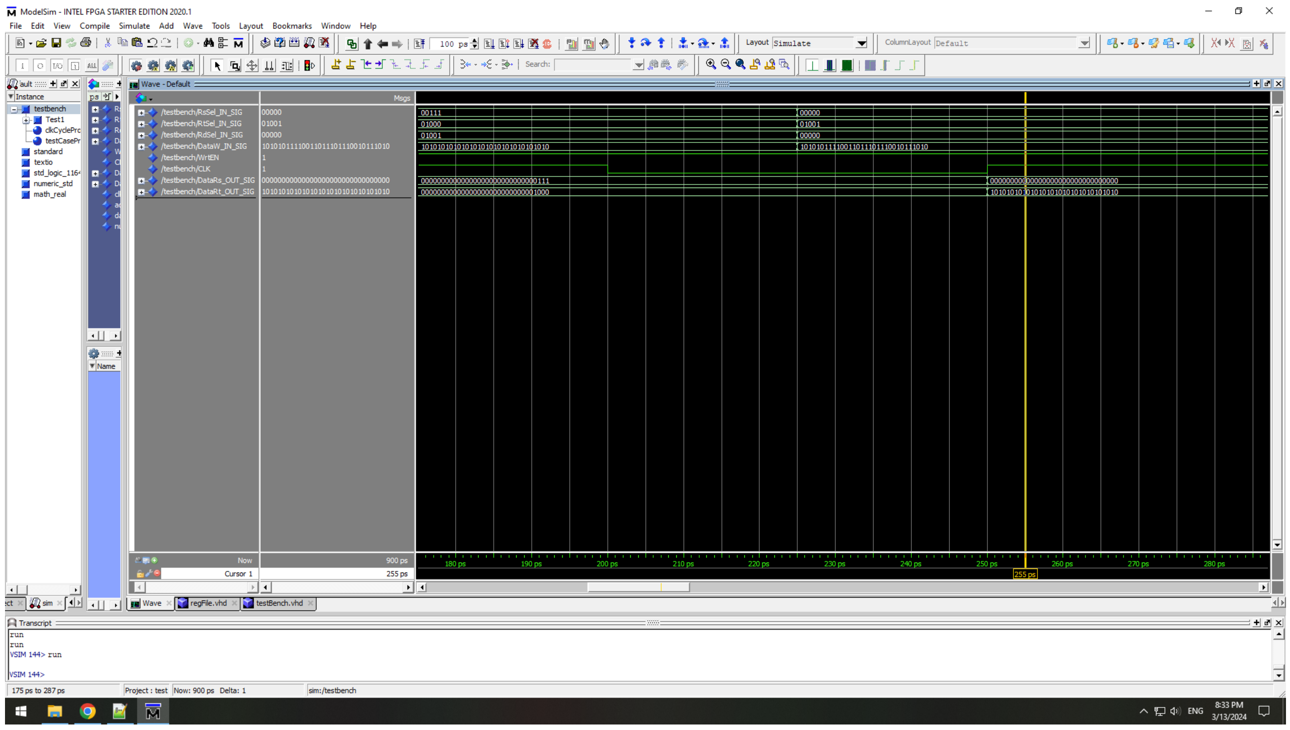
Figure 4: Test Case 1: Output Values Observed



```
/testbench/RsSel_IN_SIG      00000
/testbench/RtSel_IN_SIG      01001
/testbench/RdSel_IN_SIG      00000
/testbench/DataW_IN_SIG      10101011110011011101110010111010
/testbench/WrtEN             1
/testbench/CLK               0
/testbench/DataRs_OUT_SIG    00000000000000000000000000000111
/testbench/DataRt_OUT_SIG    00000000000000000000000000001000

Now       900 ps
Cursor 1  230 ps
```

```
run
run
VSIM 144> run

VSIM 144>
```

Figure 5: Test Case 2: Input Values Assigned

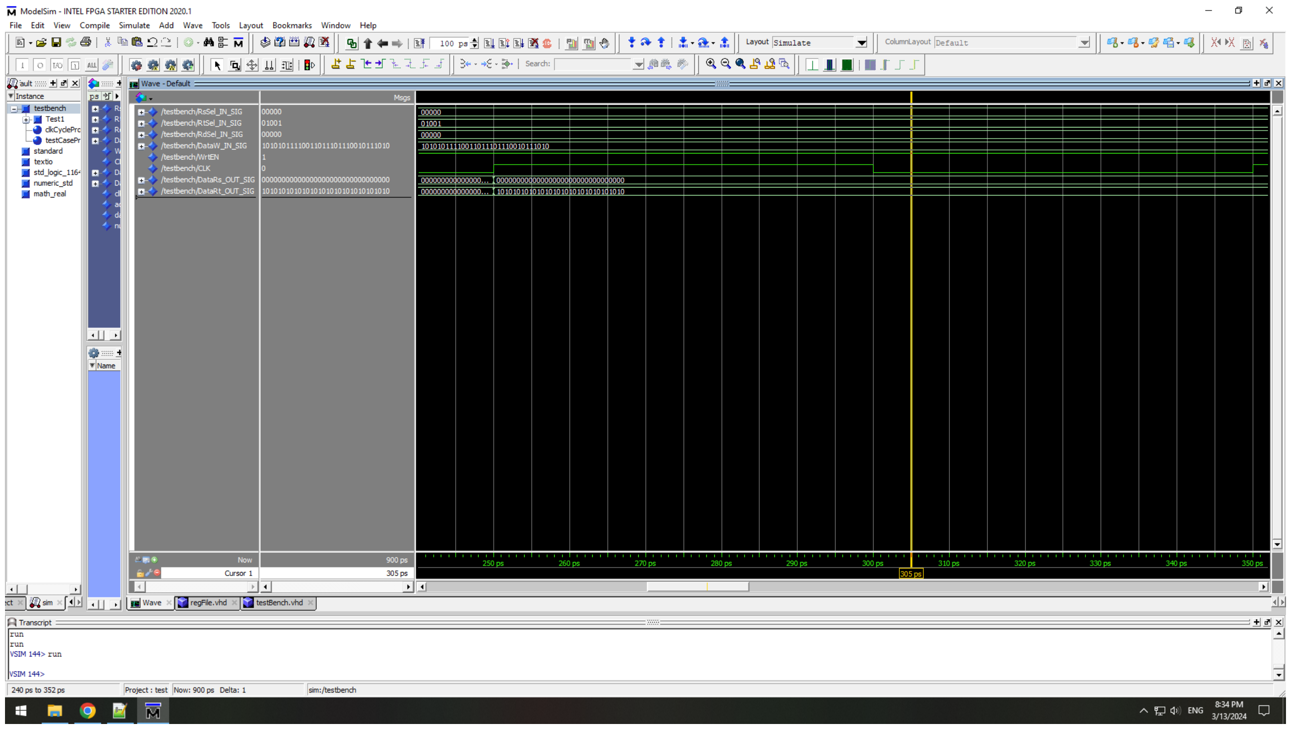


```
/testbench/RsSel_IN_SIG     00000
/testbench/RtSel_IN_SIG     01001
/testbench/RdSel_IN_SIG     00000
/testbench/DataW_IN_SIG     10101011110011011101110010111010
/testbench/WrtEN            1
/testbench/CLK              1
/testbench/DataRs_OUT_SIG   00000000000000000000000000000000
/testbench/DataRt_OUT_SIG   10101010101010101010101010101010
```

```
Now       900 ps
Cursor 1  255 ps
```

```
run
run
VSIM 144> run

VSIM 144>
```

Figure 6: Test Case 2: Output Values Observed at Rising Edge of Clock

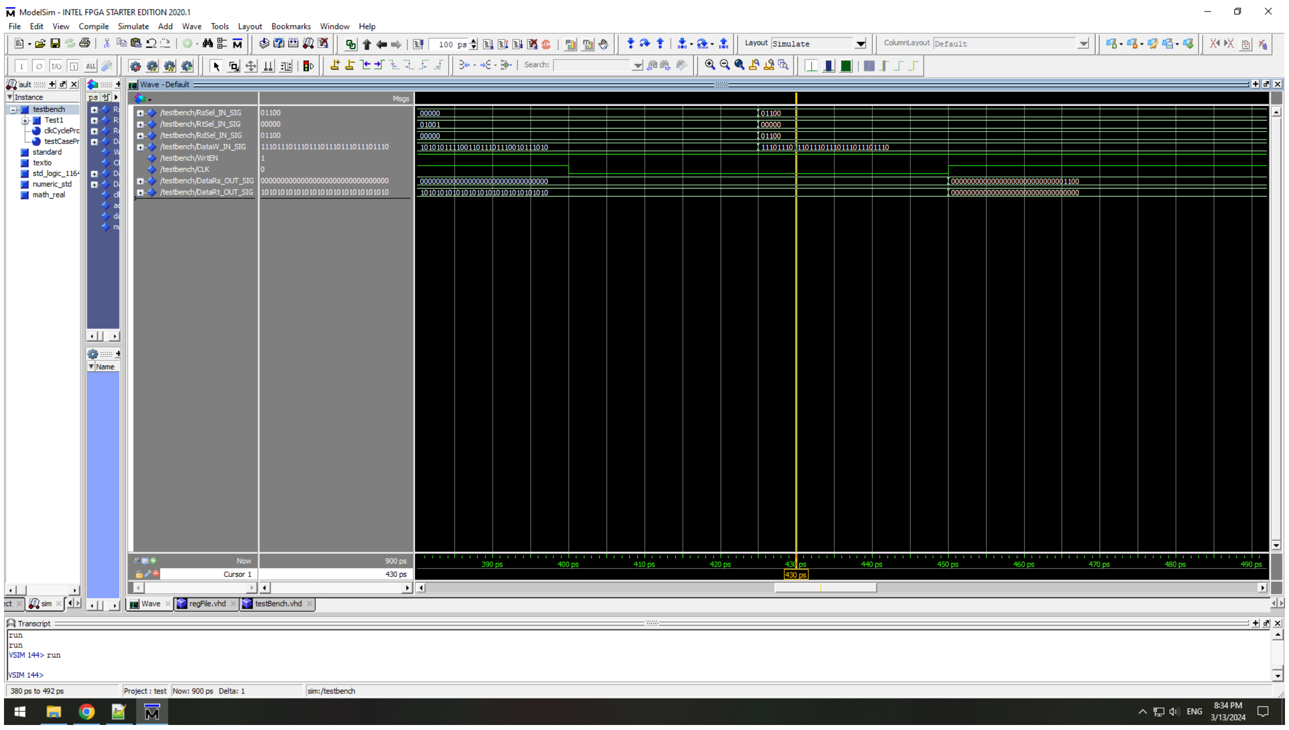


```
/testbench/RsSel_IN_SIG     00000
/testbench/RtSel_IN_SIG     01001
/testbench/RdSel_IN_SIG     00000
/testbench/DataW_IN_SIG     10101011110011011101110010111010
/testbench/WrtEN            1
/testbench/CLK              0
/testbench/DataRs_OUT_SIG   00000000000000000000000000000000
/testbench/DataRt_OUT_SIG   10101010101010101010101010101010
```

```
Now       900 ps
Cursor 1  305 ps
```

```
run
run
VSIM 144> run

VSIM 144>
```

Figure 7: Test Case 2: Output Values Observed at Falling Edge of Clock

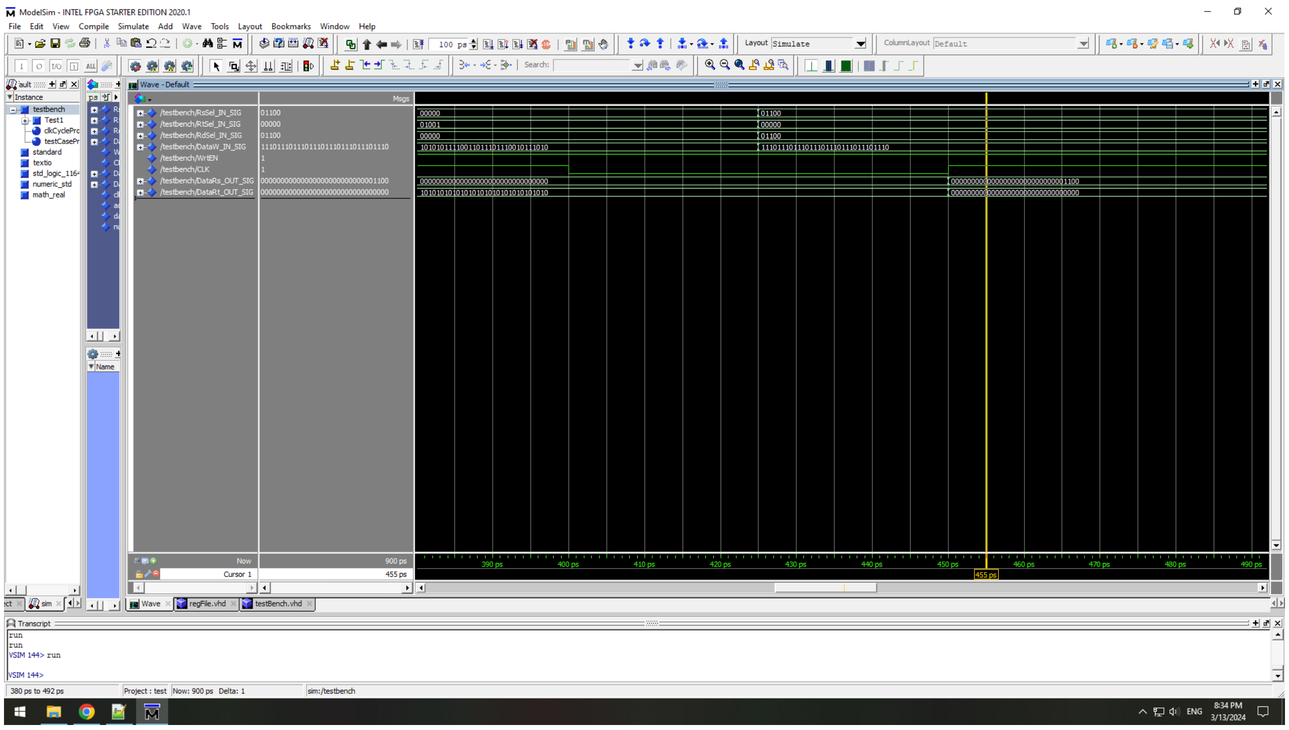


```
/testbench/RsSel_IN_SIG      01100
/testbench/RtSel_IN_SIG      00000
/testbench/RdSel_IN_SIG      01100
/testbench/DataW_IN_SIG      11101110111011101110111011101110
/testbench/WrtEN             1
/testbench/CLK               0
/testbench/DataRs_OUT_SIG    00000000000000000000000000000000
/testbench/DataRt_OUT_SIG    10101010101010101010101010101010

Now        900 ps
Cursor 1   430 ps
```

```
run
run
VSIM 144> run

VSIM 144>
```

```
380 ps to 492 ps    Project : test    Now: 900 ps  Delta: 1    sim:/testbench
```

Figure 8: Test Case 3: Input Values Assigned

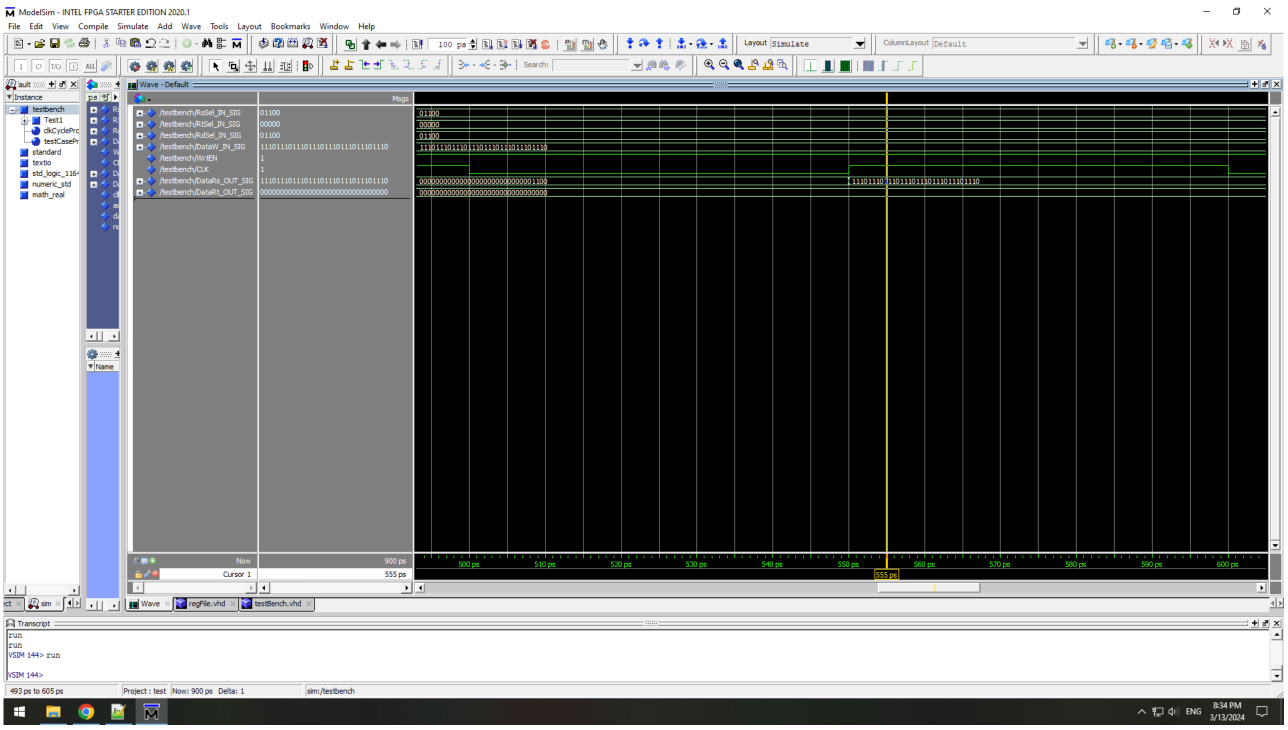


```
/testbench/RsSel_IN_SIG      01100
/testbench/RtSel_IN_SIG      00000
/testbench/RdSel_IN_SIG      01100
/testbench/DataW_IN_SIG      11101110111011101110111011101110
/testbench/WrtEN             1
/testbench/CLK               1
/testbench/DataRs_OUT_SIG    00000000000000000000000000001100
/testbench/DataRt_OUT_SIG    00000000000000000000000000000000

Now        900 ps
Cursor 1   455 ps
```

```
run
run
VSIM 144> run

VSIM 144>
```

```
380 ps to 492 ps    Project : test    Now: 900 ps  Delta: 1    sim:/testbench
```

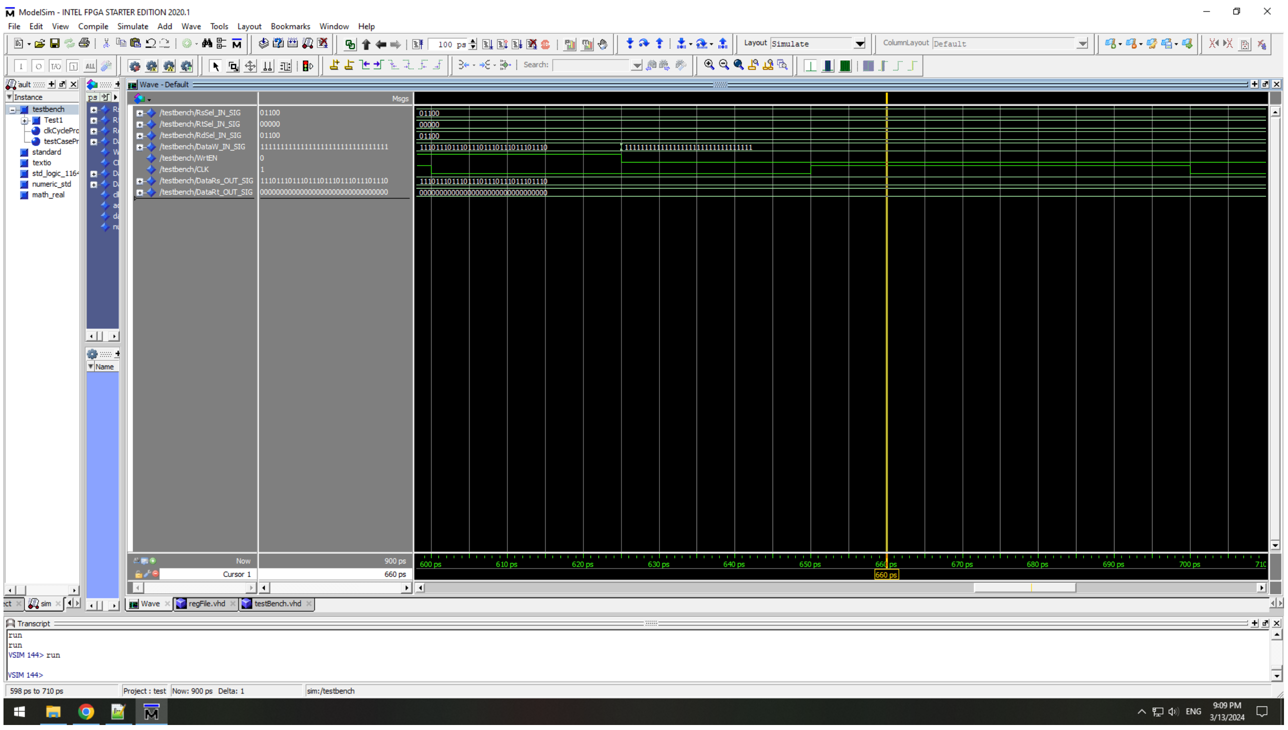
Figure 9: Test Case 3: Output Values Observed at Rising Edge of Clock



```
/testbench/RsSel_IN_SIG     01100
/testbench/RtSel_IN_SIG     00000
/testbench/RdSel_IN_SIG     01100
/testbench/DataW_IN_SIG     11101110111011101110111011101110
/testbench/WrtEN            1
/testbench/CLK              1
/testbench/DataRs_OUT_SIG   11101110111011101110111011101110
/testbench/DataRt_OUT_SIG   00000000000000000000000000000000

Now       900 ps
Cursor 1  555 ps
```

```
run
run
VSIM 144> run

VSIM 144>
```

Figure 10: Test Case 3: Output Values Observed at Falling Edge of Clock



```
/testbench/RsSel_IN_SIG     01100
/testbench/RtSel_IN_SIG     00000
/testbench/RdSel_IN_SIG     01100
/testbench/DataW_IN_SIG     11111111111111111111111111111111
/testbench/WrtEN            0
/testbench/CLK              1
/testbench/DataRs_OUT_SIG   11101110111011101110111011101110
/testbench/DataRt_OUT_SIG   00000000000000000000000000000000

Now       900 ps
Cursor 1  660 ps
```

```
run
run
VSIM 144> run

VSIM 144>
```

Figure 11: Test Case 4: Output Observed at Falling Edge of Clock