**Experiment No 6: Implementation of Register File**

|  |  |  |
| --- | --- | --- |
| **Sl No** | **Name** | **ID No** |
| **1** | **Dhruv Makwana** | **2019A3PS0381H** |

A register file is one of the important components of the RISC V data path. It can read two registers and write in to one register. The RISC V register file contains total of 32 32-bit registers. Hence 5-bits are used to specify the register numbers that are to be read or written.

**Register Read:** Register file always outputs the contents of the register corresponding to read register numbers specified. **Reading a register is not dependent on any other signals.**

**Register Write:** Register writes are controlled by a control signal **RegWrite**. Additionally the register file has a **clock signal**. The write should happen if **RegWrite signal is made 1 and if there is positive edge of clock**.

**32**

**32**

**32**

**5**

**5**

**5**

**Write Data**

**RegWrite**

**Read Register number 1**

**Read Data 2**

**Read Data 1**

**Write Register**

**Read Register number 2**

**Clock**

# Exercise 6.1 Implement RISC V register file with above specifications using Behavioral modeling. Test the Register file using Testbench.

Hint: Registers are similar to memory. So 32 32-bit registers is like 32 memory locations with 32 bits each.

Syntax to read from memory is:

**RegData = RegMemory[RegAddress];** (if used inside always block)

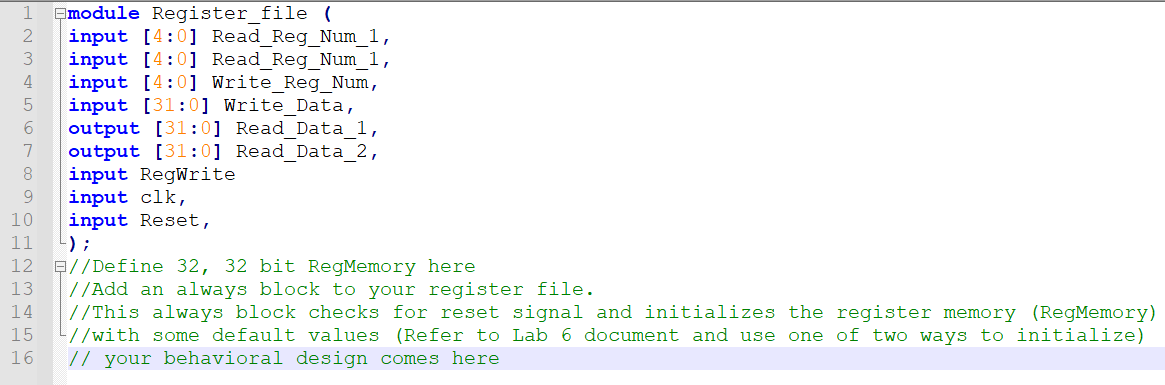
**assign RegData = RegMemory[RegAddress];** (if used outside always block)

Where **RegData** is the data read from address location specified by **RegAddress**.

Syntax to write in to memory location is:

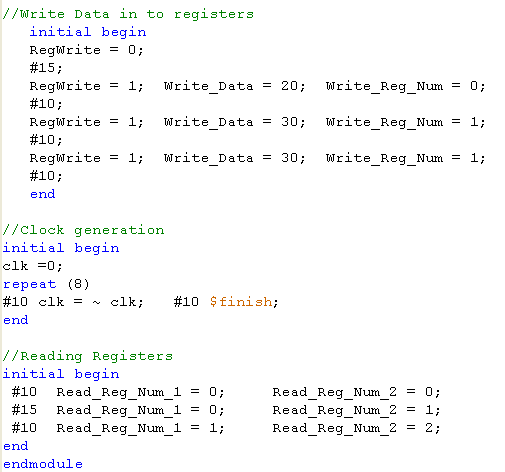
**RegMemory[RegAddress] = NewData;** (if used inside always block), where **NewData** is written in to the **RegMemory** to address location specified by **RegAddress.** Since we define RegMemory as Reg type, the above assignment is not done outside always block.

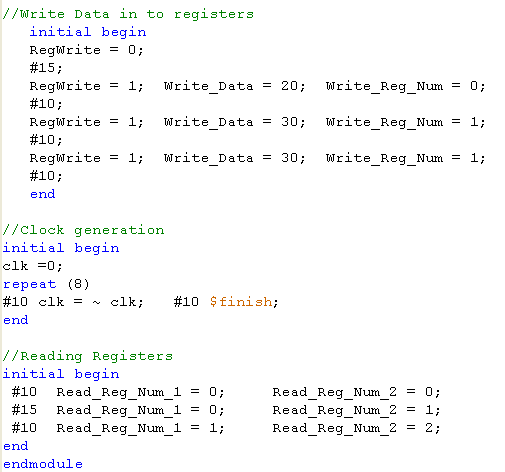
You can use the module definition shown below for register file. Please note that the following is just the module definition which specifies input, output ports. **You have to describe the behavior of Register file to make the design complete.** Also if any of the outputs are to be assigned inside the always block then you have to additionally define those outputs as **reg**.



**Testbench:**

You can test your Register file with the following test pattern. (below code comes after instantiation of the test module). The test bench contains three initial blocks one read register 0 and register 1 and register 2, Second initial block for generation of signals to check writing in to register 0 (when Register\_Num is 00000) and register 1(Register\_Num is 00001). The third initial block is used to generate the clock. If you have reset as input, you have to add one more initial block where the reset (active low) is low for some time initially and then changed to 1.



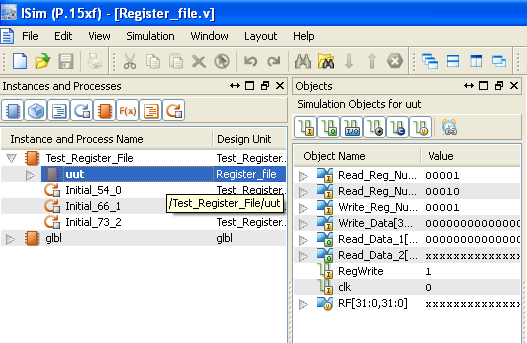


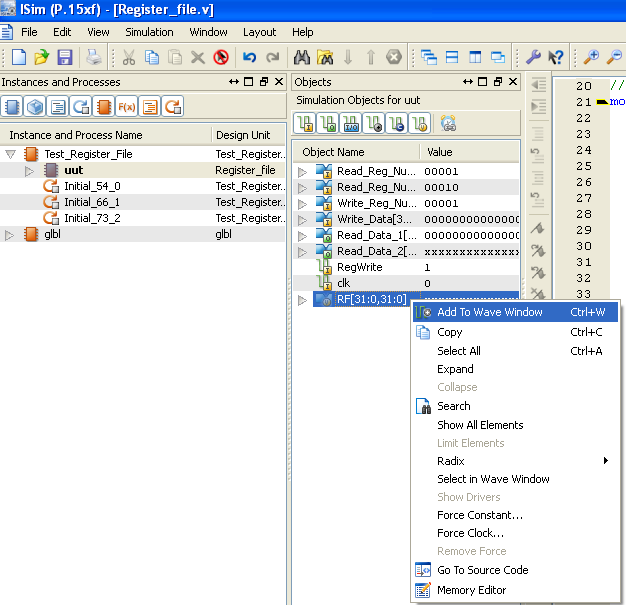
For easy viewing of the testbench waveforms, you can change the view in simulation window to decimal/Hexadecimal mode. Right click on all the input output signals select Radix and then click on signed decimal/Hexadecimal.

**Additional Information:**

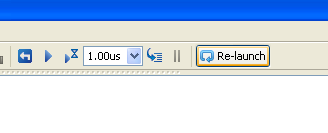
For verifying the design you might have to monitor the internal signals which could be defined as wires or Reg. In the above Register file design **RegMemory** will be defined as reg and is neither input nor output. The steps for adding internal signal (in this case RegMemory) into the wave form window is shown below.

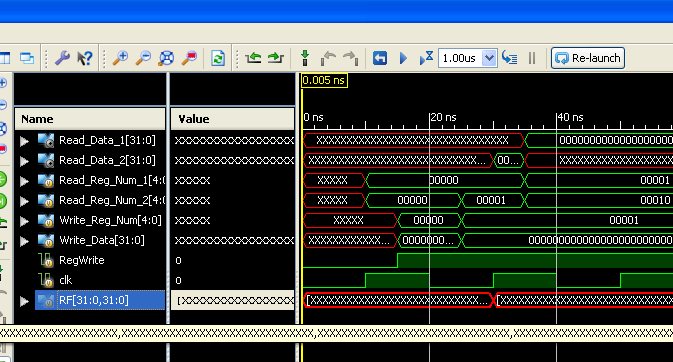
1. To check the internal signals of a module (e.g. Register File), Click on the corresponding module in the ISim window (after you run the test bench) and then right click 🡪 Add to Wave window.





1. Then Click on Re-launch Icon. Click ‘No’ when prompted to save the waveform. This will run the testbench with the newly added internal signal.

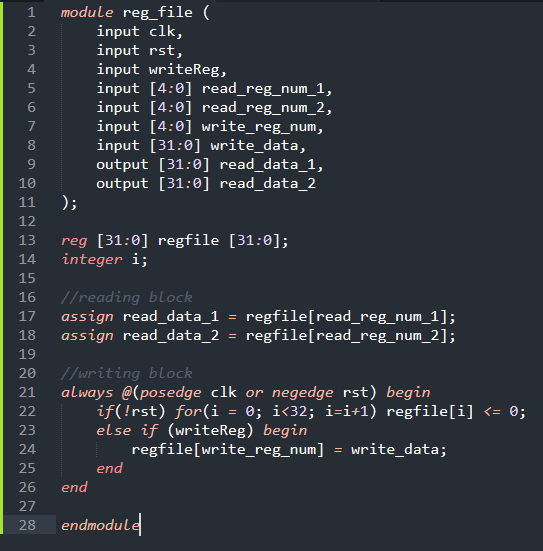




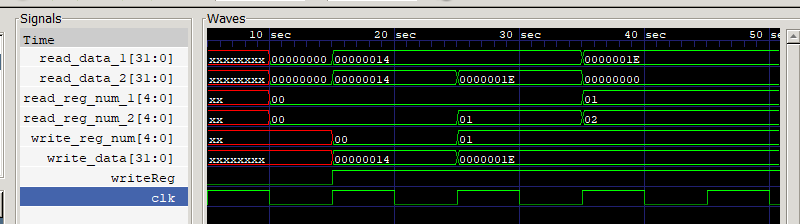
1. **Did you get any errors during the check syntax of design file or when simulating the testbench? If yes then, how did you solve the error?**

Answer: No, I did not.

1. **Copy the image of final working code of the register file here.**

Answer: 

1. **Copy the image of waveform window that is generated for the given Testbench? Waveforms should be in decimal/ hexadecimal view.**

Answer: 

1. **List the concepts you learnt from this experiment (Conclusions/Observations)**

Answer: We learned how to model a register file in Verilog and verify its functionality