```systemverilog
// Eugene Ngo
// 4/20/2023
// CSE 469
// Lab 1, Task 2

// A module to implement a 32 bit, 16 register, reg_file.
// This module takes in data and stores it in flip flops.
// The data can then be read via two read inputs and outputs.
// The data can be written in via 1 write input and 1 write enable.

module reg_file (input logic clk, wr_en, input logic [31:0] write_data,
                 input logic [3:0] write_addr,
                 input logic [3:0] read_addr1, read_addr2,
                 output logic [31:0] read_data1, read_data2);

   // Stores all the values.
   logic [15:0][31:0] memory;

   // Main logic unit. Clocked flip flops.
   always_ff @ (posedge clk) begin
   // if write enabled, write data
      if (wr_en) begin
         memory[write_addr] <= write_data;
      end
   end

   // Read out data the instant the read_data signals are updated.
   always_comb begin
      read_data1 = memory[read_addr1];
      read_data2 = memory[read_addr2];
   end

endmodule

// reg_file_testbench tests the behaviors of the reg_file by inputting
// expected testing values. This tests for cycle delays in writes,
// same-cycle reads, and 1 cycle delays for reads occuring after writes.
// The results are compared to expected values to determine if the
// functionality of the reg_file is correct.

module reg_file_testbench();
   logic clk, wr_en;
   logic [31:0] write_data, read_data1, read_data2;
   logic [3:0] write_addr, read_addr1, read_addr2;

   // Calls on the reg_file module to test it.
   reg_file dut (.clk, .wr_en, .write_data, .write_addr, .read_addr1, .read_addr2, .
read_data1, .read_data2);

   parameter clock_period = 10000;

   integer i;

   initial begin // Set up the clock
      clk <= 0;
      for (i=0; i<1000; i++) begin: clockCount
         forever #(clock_period /2) clk <= ~clk;
      end

   end

   initial begin
      $display("%t Behavior check", $time);

            // Testing functionality of
            // 1 cycle delay of writes.
      wr_en = 1'b1;                @(posedge clk);
      write_data = 32'b0;          @(posedge clk);
      write_addr = 4'b0010;        @(posedge clk);
                                   @(posedge clk);

      write_data = 32'b1;          @(posedge clk);
      write_addr = 4'b0011;        @(posedge clk);
```

```
                                        @(posedge clk);
                  // Testing functionality of
                  // same cycle reads.
            read_addr1 = 4'b0010;         @(posedge clk);
            read_addr2 = 4'b0011;         @(posedge clk);

                  // Testing the functionality of
                  // 1 cycle delayed reads
                  // after an updated write
            write_addr = 4'b0010;         @(posedge clk);
            read_addr1 = 4'b0010;         @(posedge clk);
                                          @(posedge clk);
            // read_data1 should update to 1 now.

            write_data = 32'b0;           @(posedge clk);
            write_addr = 4'b0011;         @(posedge clk);

            read_addr2 = 4'b0011;         @(posedge clk);
                                          @(posedge clk);
            // read_data2 should update to 0 now.

         end

      endmodule
```