

**BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI**  
**First Semester 2014-2015**  
**Lab Sheet – 5**  
**Topic: Register File Implementation**

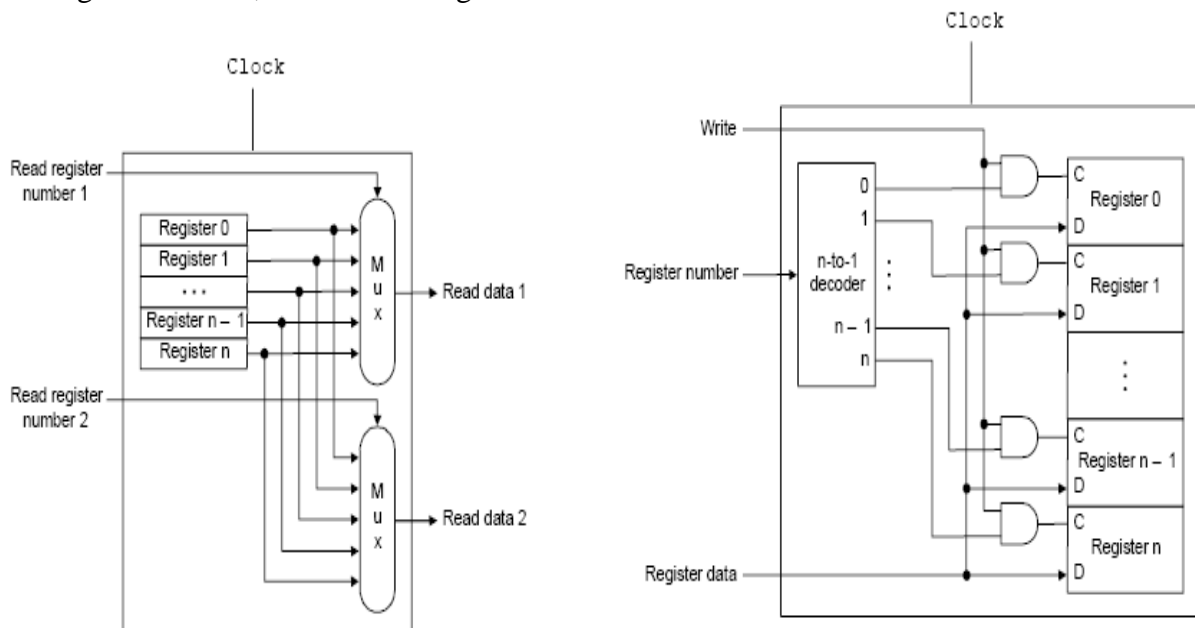
---

**Learning objectives:**

**1) Designing a sample 32-bit register file for a MIPS processor**

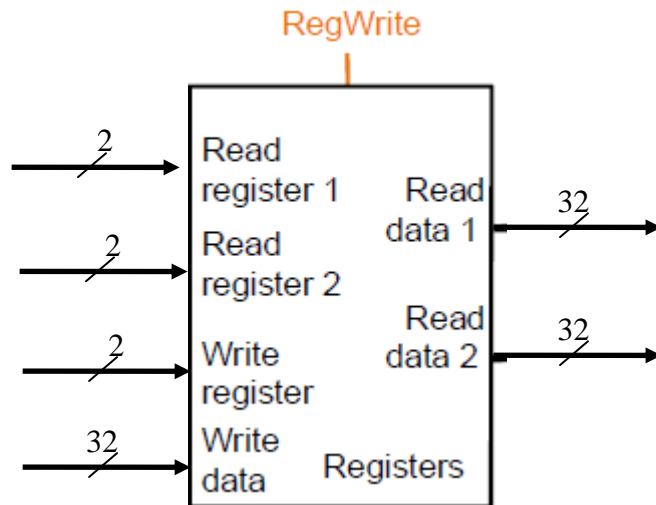
In the last lab session we have designed the ALU and the Control Unit of the ALU. In this session, we will design the register file for MIPS processor. Remember that MIPS has 32-bit registers. So the register file should be 32-bit wide. To simplify our task, by designing a register file with just four registers. The formal problem statement is as follows:

- (1) Implement a register file using verilog HDL. The following circuit diagram explains reading and writing data to a register in the register file. Assume there are four 32-bit registers in a register file. Each register is made up of D flip-flop. Register file can read two registers at a time. Two register numbers and clock are the input to the read register module and the output will be the register data as shown in the diagram. Two multiplexers are used for selecting the register for the inputted register number. For writing to the register file, a 2:4 decoder has to be designed which takes register number as input and enables required register. Input to register write module will be register data, register number, write enable signal and clock.



**Fig1: Read and Write Operations of the Register File**

The block diagram representation of the register file will look like as follows.



**Fig2: Block diagram of the Register File**

We will develop the components of the register file part-by-part in the following sub-tasks. Also it is recommended to test each of the components individually to ease the final integration into the register file.

### **Sub task 1.1: Designing the registers**

You have already designed the D\_FF using behavioral modeling in previous labs. You can use the same module to develop a 32-bit register. The D-FF is an active-low reset FF i.e. it is reset when reset is '0'.

The outline of these modules is as follows:

```
(1) module d_ff(q,d,clk,reset);
```

```
(2) module reg_32bit(q,d,clk,reset); // Register
```

Test the 32-bit Register using the following test bench.

```
module tb32reg;
reg [31:0] d;
reg clk,reset;
wire [31:0] q;
reg_32bit R(q,d,clk,reset);
always @(clk)
#5 clk<=~clk;
initial
begin
clk= 1'b1;
reset=1'b0;//reset the register
#20 reset=1'b1;
#20 d=32'hAFAFAFAF;
#200 $finish;
end
endmodule
```

**Exercise:** Design a 32-bit register module and test it with the above test bench.

### **Sub task 1.2: Designing the decoder and multiplexer**

This is pretty simple and straight forward. The output of each of these registers i.e. the content of these registers is connected to each of the inputs of the multiplexer (mux). So the register whose data is to be read on a particular read port is selected using this mux. Thus it will be 4:1 mux as we have 4 registers. What will be the width of the input and output data lines of this mux? i.e. it is \_\_\_\_ bit wide. What will be the width of the select lines of this mux?

Also during the write operation we specify the address of the register to be written i.e. “Write Register” in Fig2. This address is used to enable the writing to a register by using a decoder. So we use a 2:4 decoder.

The outline of these modules is given below.

```
(3) module mux4_1 (regData, q1, q2, q3, q4, reg_no);
```

```
(4) module decoder2_4 (register, reg_no);
```

**Exercise:** Design 4:1 mux and 2:4 decoder.

### **Sub task 1.3: Integrating the sub components**

Refer to Fig.1 for writing operation. As we stated before we have to enable writing of only one register depending on the address specified by “Write Register”. So one way we implement this by anding the Register File clock, decoder output for a register and RegWrite Signal. The output of this and gate is provided as input to the clock of the corresponding register. The scheme looks as shown below:

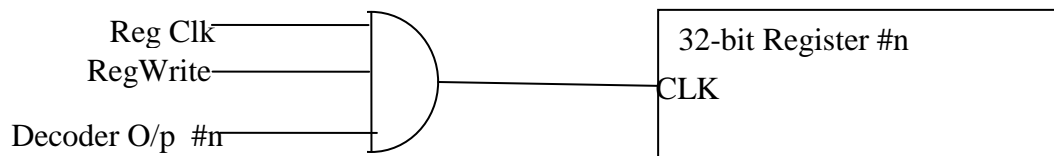


Fig. 3 Clock Gating

So for Register #0, its CLK will be ANDing of Reg Clk, Reg Write and Decoder O/p #0. So only when register #0 is written, it will be provided with a Clock input. At all other times clock input of the register is 0. This scheme is formally called as Clock gating. This technique is simple and also saves power. But it is due the delay incorporated by this AND gate, there will be some problems in practical designs called clock skew, etc. Of course these topics are outside the purview of this course.

Now create a module for Register File that would instantiate the modules created in the previous tasks.

The outline of this module is as follows.

```
module
```

```
RegFile (clk, reset, ReadReg1, ReadReg2, WriteData, WriteReg, RegWrite, ReadData1, ReadData2) ;
```

Now the question is that how will we test the functionality of this Register file? Since the number of registers is small, we will fill the registers with some data in the test bench first and do some read operations to check if it reads correctly.

Remember:

- i. First reset the registers initially.
- ii. Also a clock has to be “created”.

**Exercise:** Integrate the component modules and write a suitable test bench to test the functionality of the Register file using a suitable test bench.

**To be done later:**

Extend the above register file to a full fledged MIPS register file by incorporating 32 registers instead of 4. What all components need to be changed?

-----