Lab 7: Register File

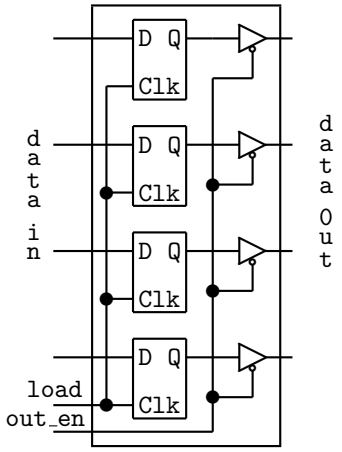
Objective: To making the register file (memory) in Verilog. The register file is made up of four registers and each register hold one nibble (half a byte, i.e.: four bits).

7.1 Make a D-type flip-flop

A D flip-flop holds 1 bit of data, and it only changes its data when the clock changes. We want a positive edge triggered D flip-flop. Design your "D_FF" module by following the interface:
module D_FF(output reg Q, input wire D, input wire enable);
Refer to the example 5.1 on page 221 of the textbook.

7.2 Make a four bit register with D flip-flops

Create a module to hold our four bit register. It is the same as IC 74374 you used in previous lab, just rename the "clk" as "load". Refer to the picture.



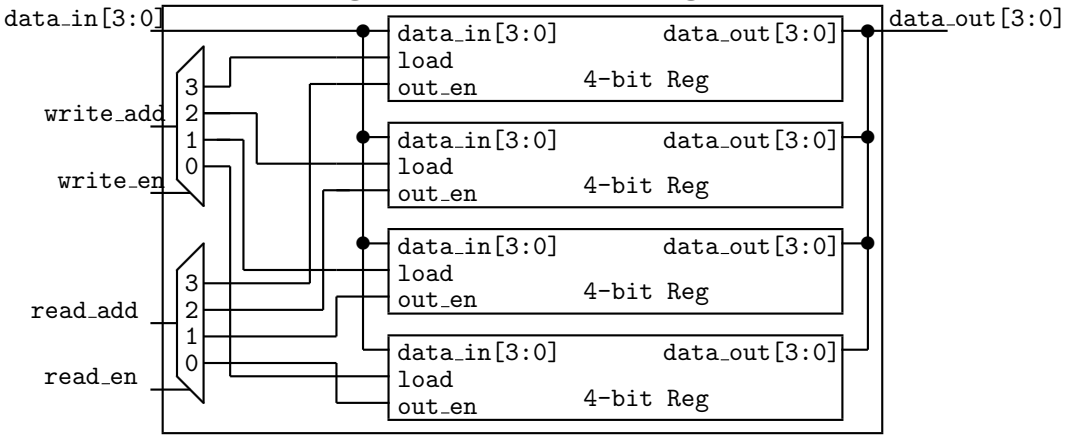

```
// name: Nibble_Reg
// desc: four bit register with output enable (low),
//        made from D flip-flops
// date:
// by  :
module Nibble_Reg(
  input  [3:0] data_in,
  input        load,out_en,
  output [3:0] data_out
  );

  // wires between flip-flops and tri-state gates
  wire   [3:0] dff_out;

  // instantiate tri-state gates to do output enable
  bufif0 t3(data_out[3],dff_out[3],out_en);
```

```
 // finish making instances

//instantiate D flip-flops here, refer to your D_FF module
D_FF Reg_Bit_3(                              ); //use Verilog 2001 syntax for port connectio

   // finish making instances

endmodule
```

7.3 Create a 2 to 4 line decoder

We will need two decoders in the final step of our design so we will create them now. Enter the 2 to 4 line decoder, "decoder_df" with the reference to the example in the textbook. Use the interface:
decoder_df (output [3:0] d_out, input en, input [1:0] d_in);
The decoder is acive low for outputs and enable line.
You can refer to the examle in page 172. Revise the example to follow the interface requested here.

7.4 Build the register file from the registers



Create a module to hold our register file. Just like the picture.

```
// name: Register_File
// desc: 4x4 register file
// date:
// by  :
module Register_File(
  input  [3:0] data_in,            // data to write
  input  [1:0] read_add,write_add, // read address and write address
  input        read_en,write_en,   // read and write enable
  output [3:0] data_out           // data to read
  );

  wire   [3:0] read_sel,write_sel;

  //instantiate decoders here
  decoder_df Dec_Read(.en(read_en), .d_in(read_add), .dout(read_sel));
  decoder_df Dec_Write(                                              );


  //instantiate registers here
  Nibble_Reg Reg_0(.data_output(data_out),.data_in(data_in),.load(write_sel[0],
                                                  .out_en(outread_sel[0]));

     // finish making instances

endmodule
```

7.5 Simulation

Make a test bench to prove to the instructor or TA your register file works correctly for both reading and writing.