BRENNA MANNING

COMPARCH HW4: DELIVERABLES 2-8

Deliverable 2

Create a module named register32. This module should exactly match the register definition above, but with 32 bits worth of D Flip Flops (d and q ports should increase width accordingly). If you'd like, try parameterizing this width.

```
21  module register32
22  (
23  output reg[31:0]q,
24  input[31:0]             d,
25  input                   wrenable,
26  input                   clk
27  );
28
29          always @(posedge clk) begin
30                  if(wrenable) begin
31                          q = d;
32                  end
33          end
34  endmodule
```

Deliverable 3

Create a module named `register32zero`. This module should match the port definition above, but instead of storing data it should ignore its inputs and always output zero.

```
39  module register32zero
40  (
41  output reg[31:0]q,
42  input[31:0]             d,
43  input                   wrenable,
44  input                   clk
45  );
46
47          always @(posedge clk) begin
48                  if(wrenable) begin
49                          q = 0;
50                  end
51          end
52  endmodule
```

Deliverable 4

Create a 32:1 multiplexer with the following module definition:

```
module mux32to1by1
(
output      out,
input[4:0]  address,
input[31:0] inputs
);
  // Your code
endmodule
```

```
55  //DELIVERABLE 4: 32:1 multiplexer
56  module mux32to1by1
57  (
58  output      out,
59  input[4:0]  address,
60  input[31:0] inputs
61  );
62  assign out=inputs[address];
63  endmodule
```

Deliverable 5

Create a multiplexer that is 32 bits wide and 32 inputs deep. There are many syntaxes available to do so, and each of them have their own little bit of excitement. The version below has more typing involved than other options, but it will allow better flexibility later. Match the following module port definition:

```
module mux32to1by32
(
output[31:0]    out,
input[4:0]      address,
input[31:0]     input0, input1, input2, ..., input31
);

  wire[31:0] mux[31:0];         // Create a 2D array of wires
  assign mux[0] = input0;       // Connect the sources of the array
  // Repeat 31 times...
  assign out = mux[address];    // Connect the output of the array
endmodule
```

```
module mux32to1by32
(
output[31:0]    out,
input[4:0]      address,
input[31:0]     input0, input1, input2, input3, input4, input5, input6, ... input31
);

  wire[31:0] mux[31:0];         // Create a 2D array of wires
  assign mux[0] = input0;       // Connect the sources of the array
  assign mux[1] = input1;
  assign mux[2] = input2;
  assign mux[3] = input3;
  assign mux[4] = input4;
  assign mux[5] = input5;
  assign mux[6] = input6;
  assign mux[7] = input7;
  assign mux[8] = input8;
  assign mux[9] = input9;
  assign mux[10] = input10;
  assign mux[11] = input11;
  assign mux[12] = input12;
  assign mux[13] = input13;
  assign mux[14] = input14;
  assign mux[15] = input15;
  assign mux[16] = input16;
  assign mux[17] = input17;
  assign mux[18] = input18;
  assign mux[19] = input19;
  assign mux[20] = input20;
  assign mux[21] = input21;
  assign mux[22] = input22;
  assign mux[23] = input23;
  assign mux[24] = input24;
  assign mux[25] = input25;
  assign mux[26] = input26;
  assign mux[27] = input27;
  assign mux[28] = input28;
  assign mux[29] = input29;
  assign mux[30] = input30;
  assign mux[31] = input31;
  assign out = mux[address];    // Connect the output of the array
endmodule
```

Deliverable 6

Provide a brief written description of how this module works. How does this behavioral Verilog result in a decoder? The decoder selects which register of the register file is being written to. Here is the full definition:

```
module decoder1to32
(
output[31:0]    out,
input           enable,
input[4:0]      address
);
    assign out = enable<<address;
endmodule
```

This module works by left shifting the enable bit a number of bits determined by the address.

Enable is a single bit input and the address is a 5 bit input(between 0-31).

Out is a 32 bit output. If enable is set to 0, the output is 00000000000000000000000000000000.

If enable is set to 1, then the bit of the output corresponding to the address is 1.

Deliverable 7

Submit Verilog files that containing your register file and all supporting modules. Note that Deliverable 8 will help you with this.

Please see `regfile.v`, `register.v`, and `decoders.v` for this deliverable.

Deliverable 8

Expand the provided test bench to classify the following register files:

1. A fully perfect register file. Return True when this is detected, false for all others.
2. Write Enable is broken / ignored – Register is always written to.
3. Decoder is broken – All registers are written to.
4. Register Zero is actually a register instead of the constant value zero.
5. Port 2 is broken and always reads register 17.

Please see `regfile.t.v` for this deliverable.