ENGR xD52: HW b100

This homework is intended to introduce you to behavioral Verilog and create your first memories. Aspects of this homework will be reused for several future labs.

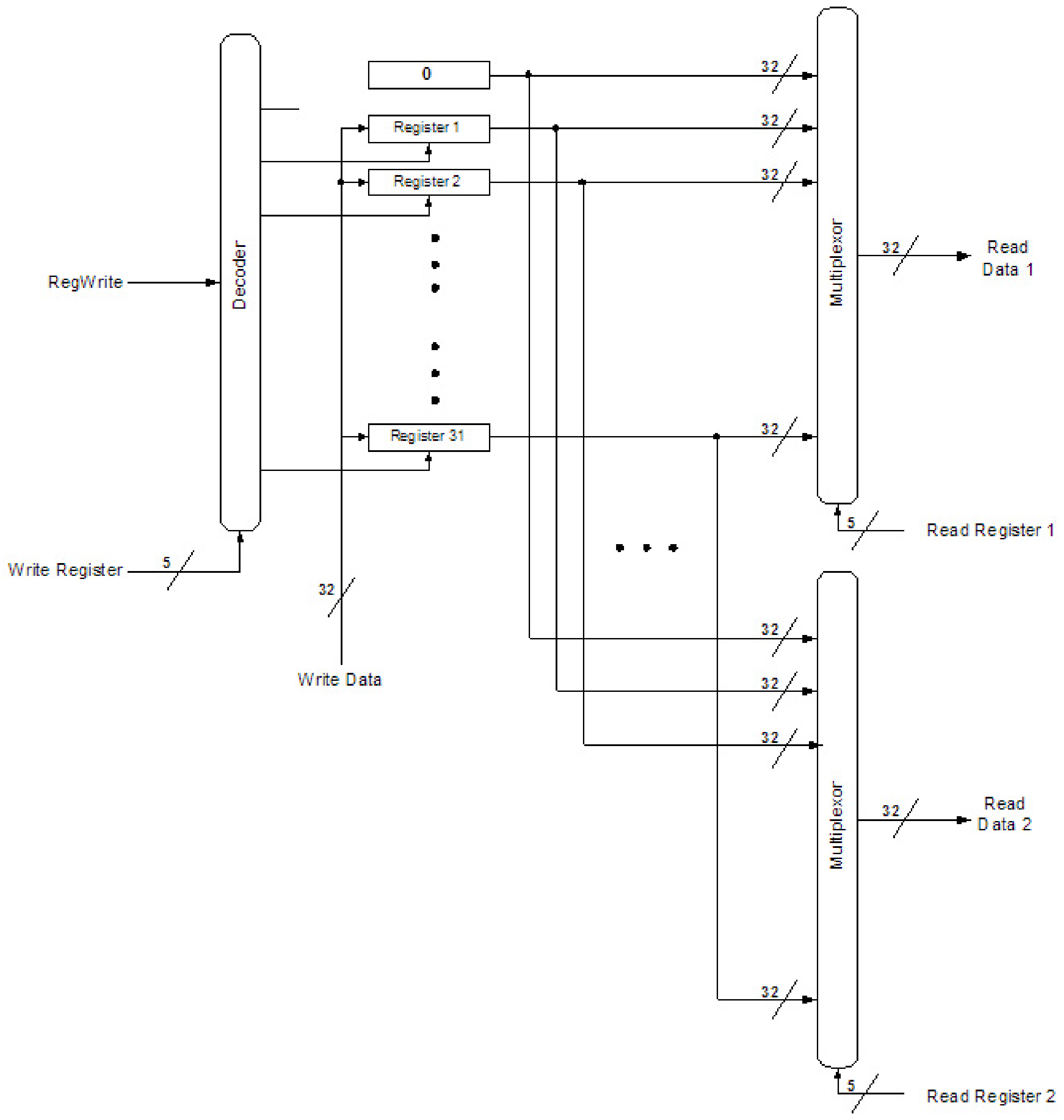
The Register File

The register file is an extremely small, extremely fast memory at the heart of your CPU. They vary per architecture, but you will create one with the following specifications:

Width: 32 bits
Depth: 32 words
Write Port: Synchronous, Positive Edge Triggered
Read Port 1: Asynchronous
Read Port 2: Asynchronous

We are mimicking the MIPS architecture, which has one unusual feature in its register file: The first "register" is actually just the constant value zero. We will exploit this oddity later when we write assembly programs for the processor.

The overall structure is shown below. The core is the 32-bit registers – 31 of them and the aforementioned constant zero. The read ports are a pair of giant multiplexers. The write port wires its data in to all registers, and then a decoder optionally enables one to be written to. This homework will build these units up in Behavioral Verilog (wrapped in structural shells), and then assemble them to create the full register file.



Register

It is critically important to write registers in Behavioral Verilog so that the synthesizer can figure out what to do.

Here is the behavioral description of a D Flip Flop with enable, positive edge triggered:

```
module register(q, d, wrenable, clk);
input  d;
input  wrenable;
input  clk;
output reg q;
```

```
always @(posedge clk) begin
    if(wrenable) begin
       q = d;
    end
end

endmodule
```

Note the enable logic. It may feel more natural to instead “gate the clock” like this:

```
always @(posedge (clk & wrenable)) begin
       q = d;
end
```

Theoretically this would work – the clock signal would be steady FALSE when disabled, and only have positive edges when enabled. However, “gating the clock” is terrible in practice – what happens if that enable signal has glitches? Also, FPGAs are typically designed to only support a few distinct clocks. The one you are using only has 4 clock managers!

Deliverable 1

Draw a circuit diagram of the structural equivalent of the two D Flip Flops above.

Deliverable 2

Create a module named “register32”. This module should exactly match the first definition above, with the exception that it has 32 bits worth of D Flip Flops, and its d and q ports increase width accordingly. If you’d like, try parameterizing this width.

Deliverable 3

Create a module named “register32zero”. This module should match the definition above, but instead of storing information it ignores its inputs and always outputs zero.

Behavioral Muxes

Behavioral Verilog makes it very easy to create a multiplexer through its array syntax. This syntax is very similar to that of the procedural languages (e.g. MATLAB, Python, C, Java, etc) you may already be familiar:

```
wire[31:0]    inputsofmux;
wire          outputofmux;
assign outputofmux=inputsofmux[address];
```

Deliverable 4

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

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

```
// your code

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. This one has more typing involved than other options, but it will allowed better flexibility later. Match the following module definition:

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

wire[31:0] mux[31:0];      // Creates a 2d Array of wires
assign mux[0] = input0;    // Connects the sources of the array
// repeat 31 times…

assign out=mux[address];   // Connects the output of the array

endmodule
```

Decoder

The decoder selects which register of the register file is being written to. Here is the full definition:

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

assign out = enable<<address;
endmodule
```

Deliverable 6

Provide a written description of how the assign statement works. Why does this result in a decoder?

Stitch it all together

You now have all the components necessary to create your register file. Use the following module definition and structure to create your register file:

```
module regfile(ReadData1,         // Contents of first register read
               ReadData2,         // Contents of second register read
               WriteData,         // Contents to write to register
               ReadRegister1,     // Address of first register to read
               ReadRegister2,     // Address of second register to read
               WriteRegister,     // Address of register to write
               RegWrite,          // Enable writing of register when High
               Clk);              // Clock (Positive Edge Triggered)
```

Deliverable 7

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

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.

These will be graded by instantiating intentionally broken register files with your tester. Your tester must return true (works!) or false (broken!) as appropriate.

It is to your advantage to test more than just these cases to better ensure that your good register file is actually good. However, those further tests do not need to be submitted.

Submission

All deliverables should be submitted in a single compressed archive and emailed to CompArch14@gmail.com with the subject line "[HW4] Your Name"

Rubric

Code portions of this assignment will be graded automatically by scripts. It is therefore critical to follow the module definitions exactly – same port definitions, same names.

No partial credit will be given for these portions.

Deliverable	Weight	Grading
1	10	Manual
2	5	Automatic
3	5	Automatic
4	10	Automatic
5	10	Automatic
6	10	Manual
7	25	Automatic
8	25	Automatic
Total	100	

Notes

Work is to be done individually. If you seek help from another individual, note that **per deliverable**.

We are not doing any time delay related analysis for this assignment. Do not include time delays.