# Discussion Session 5

CS/ECE 552

Ramkumar Ravi

27 Feb 2012

# Introduction

- Rules for HW will be up shortly (similar to previous homeworks) -> Please follow instructions
  - HW3 is due on **03/07**

- In today's section, we will cover all questions except Problem 2 and FIFO design (spend some time on these questions and we will discuss next week)

- WARNING: Codes here are for demonstration purposes only; Not tested and might have bugs as well

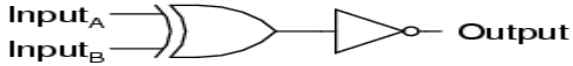
# Problem 1

- Expect just 3-4 lines of your opinion for each instruction
  - As an example, consider the "Bit Equal" instruction
  - This instruction does a bit-for-bit compare between two registers. For each bit i, if bit i of $rs is equal to bit i of $rt, set bit i of $rd; otherwise set bit i of $rd to zero.

*Exclusive-NOR gate*

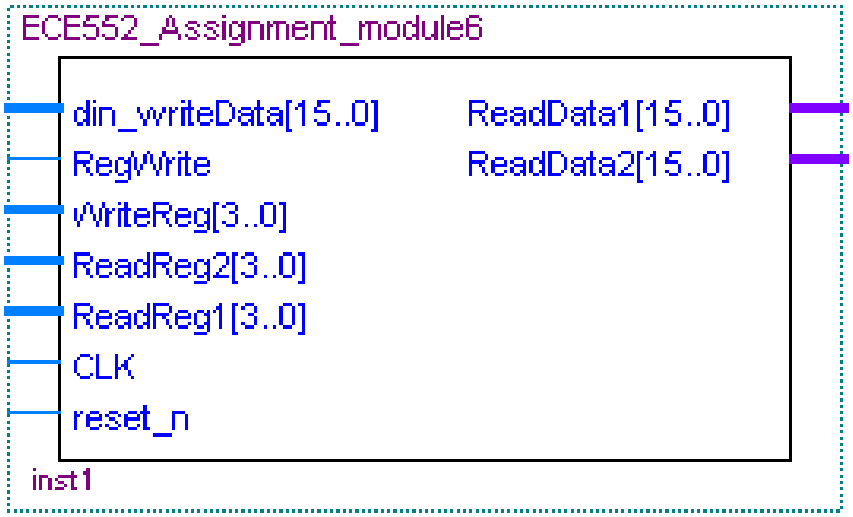

| A | B | Output |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

*Equivalent gate circuit*




  - Bit equal is hence equivalent to an XNOR instruction. So to incorporate the XNOR instruction in your data path, what changes/modifications will you need to make ? (4 lines)
  - Split register: Changes to Register file ??

# Problem 3 - Register file design




1. din_writeData [15:0] ➔ writedata [15:0]
2. RegWrite ➔ write
3. WriteReg [3..0] ➔ writeregsel [2:0]
4. ReadReg2[3..0] ➔ read2regsel [2:0]
5. ReadReg1 [3..0] ➔ read1regsel [2:0]
6. CLK ➔ clk
7. Reset_n ➔ rst
8. ReadData1 [15..0] ➔ read1data [15:0]
9. ReadData2 [15:0] ➔ read2data [15:0]

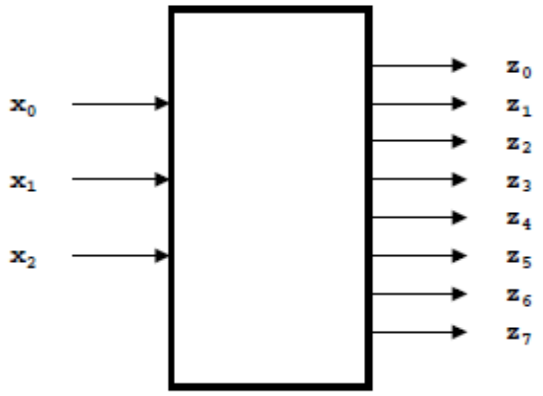
Representative Diagram (16x16 register file)

# Register File Interface

```
parameter WIDTH = 16;
input clk, rst;
input [2:0] read1regsel;
input [2:0] read2regsel;
input [2:0] writeregsel;
input [WIDTH-1:0] writedata;
input       write;
output [WIDTH-1:0] read1data;
output [WIDTH-1:0] read2data;
output      err;
```

# Register File design

- Lets start with a 3-8 Decoder




- If X0X1X2 = 3'b000; select z0 and so on..

```
wire [7:0] we, awe;
decode3_8 decoder (.sel(writeregsel), .Out(we));
and2 inst[7:0] (.in1(we), .in2({8{write}}), .out(awe));
```

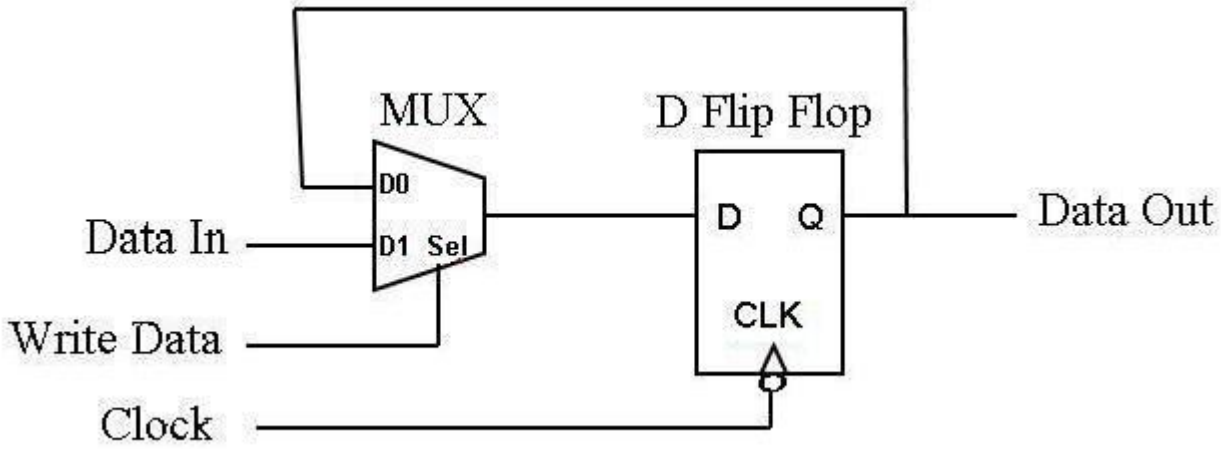
Example: if Sel is 3'b010; we[2] is 1'b1 and hence awe[2] is 1'b1

# Register file – The registers

**wire** [WIDTH-1:0] q0, q1, q2, q3, q4, q5, q6, q7;

**register** regs7 (.q(q7), .d(writedata), .clk(clk), .rst(rst), .we(awe[7]));
**register** regs6 (.q(q6), .d(writedata), .clk(clk), .rst(rst), .we(awe[6]));
and so on..

Now what is the **register** module ? (trying to do something like figure below)



# **REGISTER** module

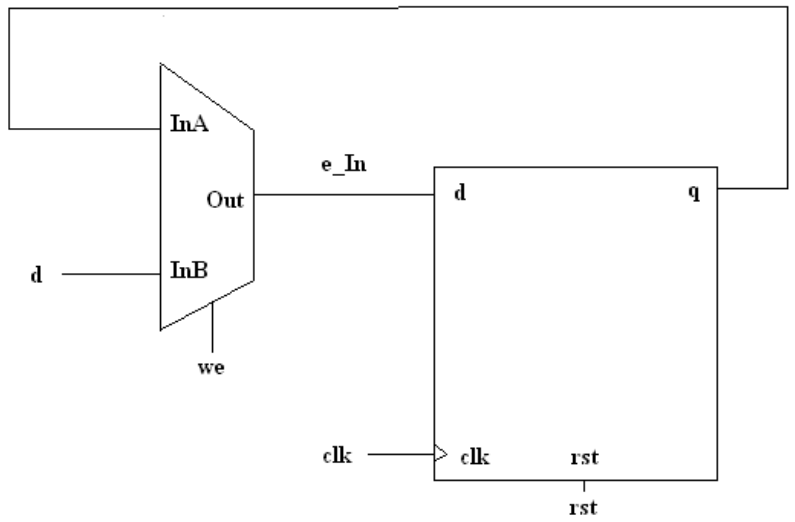
```
module register (input [15:0] d, input clk, rst, we, output [15:0] q);

wire [15:0] e_In;
mux2_1 mux[15:0] (.InA (q), .InB(d), .S({16{we}}), .Out(e_In));
dff inst[15:0] (.q(q), .d(e_In), .clk({16{clk}}), .rst({16{rst}}));
```

2:1 MUX -> If S is 1'b1; InB is selected else InA is selected

16 copies of the DFF module

NOTE:

You might also need a 8:1 MUX for
giving the READ output (not included)



# Problem 4 - Saturating Counter

```
module sc( clk, rst, ctr_rst, out, err);
    input clk;
    input rst;
    input ctr_rst;
    output [2:0] out;
    output err;
endmodule
```

- **rst**: Synchronous reset that sets output to zero at pos clock edge
- **ctr_rst**: ctr_rst is different from the global rst signal

- The ctr_rst line is active high, i.e. a logical value of 1 will reset the counter, while a logical value of 0 will let the counter increment.
- If ctr_rst is high while the counter is still counting, the output should reset to 0. If **ctr_rst** is held high in consecutive clock cycles, the counter should hold at 0.

# Code Example – One Possible implementation

```
reg [2:0] nextState;
dff inst [2:0](out, nextState, clk, rst);     /* Out maps to q; nextState maps to d */

always@(out, ctr_rst)
      begin
            case(out)                          // Every time out changes, keep evaluating by reading ctr_rst
                                               // If ctr_rst is 0; keep incrementing; if not nextState is 0
                                               // Observe that nextState is feeding the D Flip-Flop
                  3'd0: begin
                              nextState=ctr_rst? 3'd0:3'd1;
                              err=1'd0;
                        end
                  3'd1: begin
                              nextState=ctr_rst? 3'd0:3'd2;
                              err=1'd0;
                        end
                  ... ... ... ???
```

Do not forget to include a DEFAULT state if you chose to implement this way

You can approach this problem in the traditional Boolean reduction method as well (Draw state machine; encode truth table; get equations; use basic gates). Next slide contains some sample outputs

# Counter – Sample Output

- Shown at the right are the values of **ctr_rst** and **out** for a sample simulation run

1. **Out** is initially **X** (cycle 0)
2. **Out** is 0 on posedge (**rst=1**). By definition, **rst** will be held HIGH in first 2 cycles (Cycle 100)
3. **Out** is 0 (**rst=1**) (Cycle 200)
4. **Out** is being incremented from 0->1->2 (see cycles 300,400 and 500). At 500, **ctr_rst** is 1
5. So, **Out=0** on next posedge (cycle 600)
6. Again, **Out** is being incrmented from 0->1->2 (cycles 600, 700 & 800). At 800, **ctr_rst** is 1
7. So, **Out=0** on next posedge (cycle 900)

```
# time:                              0 Cycle
#  clk 1 ctr_rst 0 Out x
#
# time:                            100 Cycle
#  clk 1 ctr_rst 0 Out 0
#
# time:                            200 Cycle
#  clk 1 ctr_rst 0 Out 0
#
# time:                            300 Cycle
#  clk 1 ctr_rst 0 Out 0
#
# time:                            400 Cycle
#  clk 1 ctr_rst 0 Out 1
#
# time:                            500 Cycle
#  clk 1 ctr_rst 1 Out 2
#
# time:                            600 Cycle
#  clk 1 ctr_rst 0 Out 0
#
# time:                            700 Cycle
#  clk 1 ctr_rst 0 Out 1
#
# time:                            800 Cycle
#  clk 1 ctr_rst 1 Out 2
#
# time:                            900 Cycle
#  clk 1 ctr_rst 0 Out 0
#
# time:                           1000 Cycle
#  clk 1 ctr_rst 0 Out 1
#
# time:                           1100 Cycle
#  clk 1 ctr_rst 0 Out 2
#
# time:                           1200 Cycle
#  clk 1 ctr_rst 1 Out 3
#
# time:                           1300 Cycle
#  clk 1 ctr_rst 1 Out 0
```

# Sample Output - Continued

1. **Out** is being continuously incremented now from 0->1->2->3->4->5 (see cycles 2000,2100,2200,2300,2400,2500)
2. From 2600 onwards, **Out** retains the final value of 5 (see cycles 2600, 2700 and 2800)
3. At cycle 2800, **ctr_rst** goes HIGH
4. At the next clock, **Out** is reset to ZERO

```
# time:                    2000 Cycle
#  clk 1 ctr_rst 0 Out 0
#
# time:                    2100 Cycle
#  clk 1 ctr_rst 0 Out 1
#
# time:                    2200 Cycle
#  clk 1 ctr_rst 0 Out 2
#
# time:                    2300 Cycle
#  clk 1 ctr_rst 0 Out 3
#
# time:                    2400 Cycle
#  clk 1 ctr_rst 0 Out 4
#
# time:                    2500 Cycle
#  clk 1 ctr_rst 0 Out 5
#
# time:                    2600 Cycle
#  clk 1 ctr_rst 0 Out 5
#
# time:                    2700 Cycle
#  clk 1 ctr_rst 0 Out 5
#
# time:                    2800 Cycle
#  clk 1 ctr_rst 1 Out 5
#
# time:                    2900 Cycle
#  clk 1 ctr_rst 0 Out 0
#
```

# Sample Output – Special Case (Hold Out at Zero)

1. Over cycles 1000, 1100 and 1200, Counter is incrementing
2. At 1200, it saw **ctr_rst** is 1. So in the next cycle (cycle 1300), **Out** is ZERO
3. At 1300, **ctr_rst** is still1. So in next cycle (cycle 1400), **Out** is still ZERO
4. At 1400, **ctr_rst** is still 1. So in next cycle (cycle 1500), **Out** is still ZERO
5. At 1500, **ctr_rst** is 0 due to which counter resumes counting (You can see that it incremented from 0 to 1 in cycle 1600)

```
# time:                    1000 Cycle
#  clk 1 ctr_rst 0 Out 1
#
# time:                    1100 Cycle
#  clk 1 ctr_rst 0 Out 2
#
# time:                    1200 Cycle
#  clk 1 ctr_rst 1 Out 3
#
# time:                    1300 Cycle
#  clk 1 ctr_rst 1 Out 0
#
# time:                    1400 Cycle
#  clk 1 ctr_rst 1 Out 0
#
# time:                    1500 Cycle
#  clk 1 ctr_rst 0 Out 0
#
# time:                    1600 Cycle
#  clk 1 ctr_rst 0 Out 1
```