CSE 140L Exam

Prof. Tajana Simunic Rosing

Winter 2010

- Do not start the exam until you are told to.
- Turn off any cell phones or pagers.
- Write your name and PID at the top of every page. Do not separate the pages.
- This is a closed-book, closed-notes, no-calculator exam. You may only refer to one 8 ½” x 11” page of your own handwritten notes.
- Do not look at anyone else’s exam. Do not talk to anyone but an exam proctor during the exam.
- If you have a question, raise your hand and an exam proctor will come to you.
- You have 50 minutes to finish the exam. When the time is finished, you must stop writing.
- Write your answers in the space provided.
- To get the most partial credit, clearly and neatly show all steps of your work.

1. 20 points
2. 15 points
3. 25 points
4. 20 points
5. 20 points

Total (100 pts)
1. [5 pts total, 1pt each] Write T for True or F for False for your answer.
   a) __F__ Carry lookahead adder provides the output faster but requires more gates than carry select adder.
   b) __T__ Synthesis converts Verilog to an implementation using technology-specific primitives
   c) __F__ Adding synchronizing flip-flops after asynchronous input can eliminate metastability.
   d) __F__ In the TicTacToe problem from lab3, if O goes first, the input should always have equal number of bits set in the two inputs, xin and oin.
   e) __F__ Given CPU instructions with 3 bit opcode & 4 bit data, adding a couple of instructions to the existing 8 instructions would require no change in the number of bits of the opcode.

[15pts, 3pts each] Multiple choices questions – circle all that apply.
1. Given the Verilog code segment below, which statements are not true? (B)
   ```verilog
   reg q;
   always @(posedge clock or posedge reset) begin
     if (reset) q <= 0;
     else if (set) q <= 1;
   end
   ```
   (A) When both reset and set are zero, q doesn’t change.
   (B) The reset is synchronous.
   (C) The reset has the highest priority.
   (D) Both (A) and (B)
   (E) Both (B) and (C)

2. Which statements are correct? (A, C, D)
   (A) Register file is faster than SRAM
   (B) DRAM is faster than SRAM
   (C) DRAM is smaller than register file
   (D) Flash can erase large blocks of data at the same time
   (E) EEPROM can erase large blocks of data at the same time

3. Which modules should be changed to implement an instruction which adds R1 to the data in a specified memory location in miniCPU design of Lab 4? (A, C)
   (A) Instruction set
   (B) Instruction ROM
   (C) Instruction decoder
   (D) Register file
   (E) Data path
   (F) Program counter
4. What are the common names for the x, y and z signals? (E)

```vhdl
module mystery(v, w, x, y, z);
  input v, w, x, y;
  output reg z;
  always @(posedge v)
    begin
      if(w == 1'b1)
        z <= 1'b0;
      else if(x == 1'b1)
        z <= y;
    end
endmodule
```

(A) x- Data input, y- enable, w – synchronous reset
(B) x-clock, y-Data input, w – synchronous reset
(C) x-enable, y-clock, w – asynchronous reset
(D) x-enable, y- Data input, w – asynchronous reset
(E) x-enable, y-Data input, w – synchronous reset

5. Which statements are true for the clock counter in Lab 3 given implementation shown below? The counter should count from 00 to 59 and back to 00. (A,C)

```vhdl
always@(reset) begin
  ...
  end
always@(posedge clk) begin
  ...
end
```

(A) Asynchronous reset is asserted when the counter output is 59.
(B) Synchronous reset is asserted when the counter output is 59.
(C) We can use only one 6 bit variable to keep track of the count
(D) None of the above
**Problem 2 [15 points]**

For the following circuit, find the maximum operational frequency given the on resistance of NMOS transistor is $R_n$, of PMOS is $2R_n$, and the gate capacitance of each transistor is $C_g$. Assume the input capacitance of the OR gate and D-Flip Flop is $2C_g$. Use the following timing values:

- $R_nC_g=1\text{ns}$
- OR gate delay: 1.5ns
- D FF delay: 2.5ns
- D FF setup time: 2ns
- D FF hold time: 2.5ns

Calculating the worst case delay in between the flip flops,

(NOR) Path 1 = $2.5\text{ns} + (4R_n*2C_g) + 1.5\text{ns} + 2\text{ns} = 14\text{ns}$

(NAND) Path 2 = $2.5\text{ns} + (2R_n*2C_g) + 1.5\text{ns} + 2\text{ns} = 10\text{ns}$

(3 ORs) Path 3 = $2.5\text{ns} + ([2R_n*2C_g] + [R_n*2C_g] + [2R_n*2C_g]) + 2\text{ns} = 14.5\text{ns}$

Worst case delay --> cycle time = 14.5

Therefore, Max freq = $1 / \text{cycle time} = 68.9\text{MHz}$
Problem 3 [25 points]
The instruction set of the CPU you have designed for LAB4 has been modified as follows:

<table>
<thead>
<tr>
<th>Command</th>
<th>operands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Move</td>
<td>op3,op1,op2</td>
<td>Move the content of op2 to op1 and op3</td>
</tr>
<tr>
<td>Add1</td>
<td>op3,op1,op2</td>
<td>Add the content of op1 and op2 and put the result in op3</td>
</tr>
<tr>
<td>Add2</td>
<td>op1,#xx</td>
<td>Add the content of op1 and immediate value xx and put the result in op1</td>
</tr>
<tr>
<td>Load</td>
<td>op1,#xx</td>
<td>Load the register with 8 bit data</td>
</tr>
<tr>
<td>Shift</td>
<td>op2,op1</td>
<td>Left shift the content of op1 by number of bits represented by op2.</td>
</tr>
<tr>
<td>Mult</td>
<td>op3,op1,op2</td>
<td>Multiply op1 and op2 and put the result in op3</td>
</tr>
<tr>
<td>RST</td>
<td></td>
<td>Clears all the registers</td>
</tr>
</tbody>
</table>

* Here op1, op2, op3 are 8 bit registers names. It can be any register among R1,R2,R3.
* #xx refers to immediate data of size 8 bits.

1. **What is the minimum instruction size?**

To encode 3 registers we need 2 bits. So need totally 6 bits for representing the register operands. But for Load and Add2 we need 10 bits to represent the operands. We need to choose highest among two values. Since there are totally 7 instructions we need 3 bits to represent the opcode. So totally we need 13 bits to encode these instructions.
2. **Design and write the Verilog code for the instruction decoder.**

module instruction_decoder (instruction, r1_src_en, r2_src_en, r3_src_en, 
                           imm, r1_dest_en, r2_dest_en, r3_dest_en);
input[0:IN_WIDTH] instruction; //IN_WIDTH is the instruction width.
output r1_src_en, r1_dest_en, r2_src_en, r2_dest_en, r3_src_en, r3_dest_en, imm;

r1_src, r2_src, r3_src - represents the signals for source registers
r1_dest, r2_dest, r3_dest - represents the signals for destination registers
imm – signal to represent immediate value in the instruction.

Module instruction_decoder(instruction, r1_src_en, r2_src_en, r3_src_en, imm, r1_dest_en, 
                           r2_dest_en, r3_dest_en);
input[0:13] instruction; //IN_WIDTH is the instruction width.
Output reg r1_src_en, r1_dest_en, r2_src_en, r2_dest_en, r3_src_en, r3_dest_en, imm;

Initial begin
r1_src_en = 0;
r2_src_en = 0;
r3_src_en = 0;
r1_dest_en = 0;
r2_dest_en = 0;
r3_dest_en = 0;
imm = 0;
end

    case(instruction[12:10])
        //Move op3, op1, op2
        4'b000: begin
            Case(instruction[9:8]) // op3
                2'b00: r1_dest_en = 1;
                2'b01: r2_dest_en = 1;
                2'b10: r3_dest_en = 1;
            End case
            Case(instruction[3:2]) // op1
                2'b00: r1_src_en = 1;
                2'b01: r2_src_en = 1;
                2'b10: r3_src_en = 1;
            End case
            Case(instruction[1:0]) // op2
                2'b00: r1_src_en = 1;
                2'b01: r2_src_en = 1;
                2'b10: r3_src_en = 1;
        end
    endcase
end
First Name:  Last Name:  PID:

End case
end

//Add op3,op1,op2
3'b001: begin
    Case(instruction[9:8]) // op3
        2'b00: r1_dest_en = 1;
        2'b01: r2_dest_en = 1;
        2'b10: r3_dest_en = 1;
    End case
    Case(instruction[3:2]) // op1
        2'b00: r1_src_en = 1;
        2'b01: r2_src_en = 1;
        2'b10: r3_src_en = 1;
    End case
    Case(instruction[1:0]) // op2
        2'b00: r1_src_en = 1;
        2'b01: r2_src_en = 1;
        2'b10: r3_src_en = 1;
    End case
end

//Add2 op1,#xx
4'b010: begin
    Case(instruction[9:8]) // op3
        2'b00: r1_dest_en = 1;
        2'b01: r2_dest_en = 1;
        2'b10: r3_dest_en = 1;
    End case
   Imm = 1;
end

//Load op1,#xx
4'b011: begin
    Case(instruction[9:8]) // op3
        2'b00: r1_dest_en = 1;
        2'b01: r2_dest_en = 1;
        2'b10: r3_dest_en = 1;
    End case
   Imm = 1;
end

//Shift op2,op1
4'b100: begin
    Case(instruction[9:8]) // op2
        2'b00: r1_dest_en = 1;
        2'b01: r2_dest_en = 1;
end
First Name:  
Last Name:  
PID:

2'b10: r3_dest_en = 1;
End case
Case(instruction[3:2]) // op1
2'b00: r1_src_en = 1;
2'b01: r2_src_en = 1;
2'b10: r3_src_en = 1;
End case
end
//Mult op3,op1,op2
4'b101: begin
  Case(instruction[9:8]) // op3
    2'b00: r1_dest_en = 1;
    2'b01: r2_dest_en = 1;
    2'b10: r3_dest_en = 1;
  End case
  Case(instruction[3:2]) // op1
    2'b00: r1_src_en = 1;
    2'b01: r2_src_en = 1;
    2'b10: r3_src_en = 1;
  End case
  Case(instruction[1:0]) // op2
    2'b00: r1_src_en = 1;
    2'b01: r2_src_en = 1;
    2'b10: r3_src_en = 1;
  End case
end
endcase
end
endmodule

**Problem 4 [20 points]**

Given the Verilog module below and specified inputs, draw the output waveforms

```
Module test (clk, a, b, f, g, h);
Input clk, a, b;
Output f, g, h;
Reg f, g, h;
    always@(negedge clk)
        f <= #5 a | b;
    always@(clk)
        g = #10 f ^ b;
    always@(f or g)
        h = #2 f & g;
endmodule
```
**Problem 5 [20 points] Sequence Detector:**

Write the complete Verilog description for a circuit with input x and two output z1 or z2. The circuit consists of two concurrent Mealy machines. The output z1 becomes 1 when 1011 sequence is found on the input, and the z2 output becomes 1 when a 110 sequence is found on x.

1011 detector

```
module seq3_detect_mealy(x,clk, z1,z2);
// Mealy machine for a three-1s sequence detection
input x, clk;
output z1,z2;
```

111 detector
reg z1, z2;
reg [1:0] pstate, nstate;
reg [1:0] pstate1, nstate1;
parameter S0 = 2'b00, S1 = 2'b01, S2 = 2'b10, S3 = 2'b11;

// Next state and output combinational logic
// Use blocking assignments "="
always @(x or pstate)
case (pstate)
S0: if (x)
    begin
        nstate = S1;
        z1 = 0;
    end
else
    begin
        nstate = S0;
        z1 = 0;
    end
S1: if (x)
    begin
        nstate = S2;
        z1 = 0;
    end
else
    begin
        nstate = S0;
        z1 = 0;
    end
S2: if (x)
    begin
        nstate = S3;
        z1 = 1;
    end
else
    begin
        nstate = S0;
        z1 = 0;
    end
S3: if (x)
    begin
        nstate = S3;
        z1 = 1;
    end
endcase;
else
    begin
    nstate = S0;
    z1 = 0;
    end
endcase

// Sequential logic, use nonblocking assignments "<="
always @(posedge clk)
pstate <= nstate;
endmodule

always @(x or pstate1)
case (pstate1)
S0: if (x)
    Begin
    nstate1 = S1;
    z2 = 0;
    end
else
    begin
    nstate1 = S0;
    z2 = 0;
    end
S1: if (x)
    begin
    nstate1 = S1;
    z2 = 0;
    end
else
    begin
    nstate1 = S2;
    z2 = 0;
    end
S2: if (x)
    begin
    nstate1 = S3;
    z2 = 0;
    end
else
    begin
    nstate1 = S0;
    z2 = 0;
    end
S3: if (x)
    begin
        nstate1 = S1;
        z2 = 1;
    end
else
    begin
        nstate1 = S2;
        z2 = 0;
    end
endcase

// Sequential logic, use nonblocking assignments "<="
always @(posedge clk)
pstate1 <= nstate1;
endmodule