CSE 140L Final Exam

Prof. Tajana Simunic Rosing

Spring 2008

- 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 170 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. 10 points
2. 20 points
3. 20 points
4. 15 points
5. 30 points
6. 15 points
7. 30 points
8. 30 points
9. 30 points

Total (200 pts)
1. [10 pts total, 1pt each] Write T for True or F for False for your answer.

a) ___F___ For the 4-bit full adder in Lab 1, the path with worst case delay is from input to the highest bit of sum, assuming all inputs arrive simultaneously.

b) ___T___ For the 4 to 1 MUX in Lab 1, using pass gate logic can give smaller worst case delay than using complementary CMOS logic.

c) ___F___ In the calculator of Lab 2, the overflow signal is equivalent to the carry-out signal of the 4-bit full adder inside the calculator.

d) ___F___ In Lab 3, the song_reader sends note and duration to note_player only when it gets note_done signal from the note_player.

e) ___T___ In Lab 3, the note_player loads note and duration only when it gets the load_new_note signal from the song_reader.

f) ___T___ In Lab 3, if the speed takes effect and changes the output of the song_reader immediately, the note_player may act give an incorrect result.

g) ___T___ Rp = 2Rn assumption for estimating transistor delay is true since PMOS has mobility that is twice slower compared to NMOS.

h) ___T___ The memory chip has 16 Mega words with 1 byte per each word. It uses 24-bit long address, 8-bit long input/output data signals. Its total size is 128 Mega bits.

i) ___F___ Adding synchronizing flip-flops after asynchronous input can eliminate metastability.

j) ___T___ Clock skew and glitches may cause synchronization failure.
2. [20pts, 2pts each] Multiple choices questions – circle all that apply.

1) Assume you are writing Verilog code and you use functional simulation to verify your design. Which statement is correct? (A)

(A) If an output is not driven by any source, the value will be “Z”. If an output is driven by multiple sources, the value can be “X”.
(B) If an output is not driven by any source, the value will be “X”. If an output is driven by multiple sources, the value can be “X”.
(C) If an output is not driven by any source, the value will be “Z”. If an output is driven by multiple sources, the value can be “Z”.
(D) If an output is not driven by any source, the value will be “X”. If an output is driven by multiple sources, the value can be “Z”.
(E) None of the above.

2) For a completely specified state diagram, what is the maximum number of transitions that exit each state for a logic design with 2 inputs? (D)

(A) 1
(B) 2
(C) 3
(D) 4
(E) Cannot be determined from the information given.
(F) None of the above.

3) In Lab 1, LED[3:0] and SW[3:0] are used as inputs and outputs. Which statement is true? (D)

(A) LEDs turn on when the signal is 1; when SWs switch on, the signal is 1.
(B) LEDs turn on when the signal is 0; when SWs switch on, the signal is 1.
(C) LEDs turn on when the signal is 1; when SWs switch on, the signal is 0.
(D) LEDs turn on when the signal is 0; when SWs switch on, the signal is 0.

4) In Lab 3, if there is no beat_counter in the note_player, how many states would be enough to make note_player work? Assume the longest duration is 48 beats. (C)

(A) Less than or equal to 8
(B) More than 8, less than or equal to 32
(C) More than 32, less than or equal to 64
(D) None of the above
5) In Lab 3, if we connect the beat_generator with a 48KHz clock instead of the system clock, what would be the STOP value in beat_generator? (A)
   (A) 1000
   (B) 100
   (C) 3000
   (D) 2083333
   (E) None of the above

6) Which modules should be changed to implement instructions for loading data from memory to R1 and writing data from R1 to memory in miniCPU design of Lab 4? (A, B, C, D)
   (A) Instruction set
   (B) Instruction ROM
   (C) Instruction decoder
   (D) Register file
   (E) Data path
   (F) Program counter

7) Two Verilog code segments are given below. Both use one hot encoded my_in as an input. Which statement is true about these segments? (A)

```
always@(my_in)
begin
  casex (my_in)
    4'bxxx1: y=2'b00;
    4'bx1x: y=2'b01;
    4'bx1xx: y=2'b10;
    4'b1xxx: y=2'b11;
    default: y=2'bxx;
  endcase
end
```  

```
always@(my_in)
begin
  casex (my_in)
    4'b0001: y=2'b00;
    4'b0010: y=2'b01;
    4'b0100: y=2'b10;
    4'b1000: y=2'b11;
    default: y=2'b00;
  endcase
end
```

(A) Code A generates smaller circuit than B after synthesis.
(B) Code B generates smaller circuit than A after synthesis.
(C) Code A and B generate the same circuit after synthesis.
(D) Cannot be determined from the information given
(E) None of the above.

8) Which statement is true for the clock counter in Lab 2? Assume the counter counts
up from 00 to 59 and goes back to 00, and the reset is implemented using the following format: (The actual assignments are replaced by “…”) (A)

\[
\begin{array}{l}
... \\
  \text{always@}(\text{reset}) \text{ begin} \\
  ... \\
  \text{end} \\
  \text{always@}(\text{posedge clk}) \text{ begin} \\
  ... \\
  \text{end}
\end{array}
\]

(A) It has an asynchronous reset. The reset is asserted when the counter output is 60.
(B) It has a synchronous reset. The reset is asserted when the counter output is 59.
(C) It has an asynchronous reset. The reset is asserted when the counter output is 59.
(D) None of the above

9) Given the Verilog code segments below, which statement is not true? (C)

\[
\begin{array}{c}
\text{A} \\
1 \text{ Initial} \\
2 \text{ begin} \\
3 \quad a = 1; \ c = 2; \\
4 \quad \#3 \ b <= a; \\
5 \quad \#6 \ x <= b+c; \\
6 \text{ end}
\end{array}
\]

\[
\begin{array}{c}
\text{B} \\
1 \text{ Initial} \\
2 \text{ begin} \\
3 \quad a = 1; \ c = 2; \\
4 \quad \#3 \ b = a; \\
5 \quad x <= \#6 \ b+c; \\
6 \text{ end}
\end{array}
\]

(A) Line 4 in code A grabs a at t=0 and assign b at t=3.
(B) Line 4 in code B grabs a at t=3 and assign b at t=3.
(C) Line 5 in code A grabs (b+c) at t=3, and assign x at t=9.
(D) Line 5 in code B grabs (b+c) at t=3, and assign x at t=9.
(E) Both (A) and (C).
(F) Both (B) and (D).

10) Given the Verilog code segment below, which statement is not true? (B)

\[
\begin{array}{l}
\text{reg q;} \\
\text{always @}(\text{posedge clock or posedge reset}) \text{ begin} \\
  \text{if (reset) } q <= 0; \\
  \text{else if (set) } q <= 1; \\
\text{end}
\end{array}
\]
(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)
3. [20 pts]
   a) [10 pts] Draw the CMOS transistor level implementation of the design shown below (Between XOR and inverter is a pass gate controlled by input C). Assume only signals a, b and c are available.

   b) [10 pts] Calculate the worst case delay of the circuit from input a to point F before the inverter. Assume PMOS on resistance is 2Rn, NMOS is Rn, gate capacitance is Cg, pass gate capacitance at connection to the XOR is Cg. What input combination produces the worst case delay?

   When c=0, F=Z.
   When c=1, F=output of XOR gate, T_{pass}=(2Rn/Rn)2Cg=4RnCg/3.
   Worst case delay happens when a=1, b=0 or a=0, b=1.

   Worst case delay
   = 4RnCg(inverter delay) + 4RnCg(XOR delay)+4RnCg/3(pass gate delay)
   = 28RnCg/3
4. [15pts] Given the Verilog module below and specified inputs, draw the output waveforms

```verilog
Module test (clk, a, b, f, g);
Input clk, a, b;
Output f,g;
Reg f,g;
    always@(negedge clk)
        f <= #5 a | b;
    always@(clk)
        g = #10 f ^ b;
endmodule
```

![Waveform diagram](image_url)
5. [30 pts] A clock generator circuit is given below. Assume the delay of inverter is $t_I$, the delay of MUX and delay element is $t_M=3\ t_I$, and the delay of AND gate is $t_A=2\ t_I$. The control signals of MUX are now shown.

![Timing Diagram](image)

a. [10 pts] Draw the timing diagram for signals a, b and c.

```
<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2t</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3t</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6t</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8t</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

b. [10 pts] Calculate the clock duty cycle when the control signal of MUX is set to connect d with a.

```
<table>
<thead>
<tr>
<th></th>
<th>e</th>
<th>d=a</th>
<th>clk</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2t</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3t</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6t</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8t</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9t</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

The waveform of e and d are the same as a, therefore the duty cycle = 50%.

c. [10 pts] Calculate the clock duty cycle when the control signal of MUX is set to connect d with b.

```
<table>
<thead>
<tr>
<th></th>
<th>e</th>
<th>d=b</th>
<th>clk</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1t</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4t</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6t</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7t</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9t</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

When connected to b, d has different phase with e, therefore they are both 1 for only one $t_I$, and the duty cycle = 1/6.
6. [15 pts] You are given three predesigned logic elements: a full-adder, a MUX and a D flip-flop. Using these elements as building blocks and minimum number of extra logic gates, draw the diagram of an arithmetic logic unit with the specifications given in the table below. The external outputs are Q[3:0], while the external inputs are P[1:0], A and clk.

<table>
<thead>
<tr>
<th>P₁P₀</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Nop (keep Q the same)</td>
</tr>
<tr>
<td>01</td>
<td>Increment Q</td>
</tr>
<tr>
<td>10</td>
<td>Shift Q (e.g. if Q[3:0] = 1111 and I=0, then Q[3:0]' = 1110)</td>
</tr>
<tr>
<td>11</td>
<td>Qᵢ' = A</td>
</tr>
</tbody>
</table>

(a) [5 pts] Buggy Verilog code of the three elements is given below. Correct all errors.

```
Module fulladder(x, y, c_in, c_out, sum);
Input x, y, c_in;
Output c_out, sum;
Assign sum = (x ^ y) ^ c_in;
Assign c_out = x&y | x&c_in | y&c_in;
endmodule

Module mux (x, s, z);
Input [3:0] x;
Input [1:0] s;
Output z;
Wire a, b;
assign a = s[0]?x[1]:x[0];
assign b = s[0]?x[3]:x[2];
assign z = s[1]?b:a;
endmodule
```
(b) [10 pts] Draw the block diagram of the design with FA, MUX and D-ff using as few extra logic as you can.
7. [30 pts] The song reader from the Lab 3’s music player has the following FSM diagram:

a) [10 pts] Add the rewind function to the music player. Inputs are play, reset, speed, note_done, and rewind, which determines if the music player should rewind or not. When the music player is rewinding, play=0 pauses it. When it reaches the beginning of the current song, it stops and sends song_done=1 to MCU. Song_rom_addr_counter uses new decrement input: when it is 1, the counter counts down. Draw the new FSM diagram of the song reader.

a: look_up_addr; b: new_note; c: song_done

---

a: look_up_addr; b: new_note; c: song_done; d: decrement
b) [15 pts] Write the Verilog code for the FSM of the new song reader.

Parameter s0 = 4’b0000, s1=4’b0001, s2=4’b0010, s3=4’b0011, s4=4’b0100, s5=4’b0101, s6=4’b0110;

always@ (posedge clk) begin
    if(reset) begin
        look_up_addr <= 0;
        new_note <= 0;
        song_done <= 0;
        decrement <= 0;
    end
    else begin
        case(state)
        s0: begin
            if(play) begin
                look_up_addr <= 1; s<=s1; end
            end
        s1: begin
            new_note <=1; look_up_addr <=0; s <= s2; end
        s2: begin
            if(note_down) begin
                if(rewind == 0) begin
                    look_up_addr <=1; s<=s1; end
                else begin
                    decrement <=1; s<=s6; end
                end
            else if( (addr == 0) & (rewind == 0) ) s <= s3;
            else if( (addr == 31) & (rewind == 1) ) s <= s3;
            else if(play == 0) s <= s4;
            else s <= s2;
        end
        s3: begin
            if(note_done) begin song_done <=1; s <= s0; end
        end
        s4: begin
            if(play) s <= s2;
            else s <= s4;
        end
        s5: begin
            if(play) s <= s3;
            else s<=s5;
        end
        s6: begin
new_note <=1; decrement <=0; s<=s2; end
decase
end //always
c) [5 pts] Add volume control to the music player. The amplitude of audio output should change according to the value of new input signal volume. Write down the changes needed to the original Verilog code in the sine_reader in order to add the volume control. The IOs of the new sine reader are given below:

```verilog
module sine_reader( clk, reset, step_size, play, volume, generate_next, sample_ready,
                    sample );
  input clk, reset, generate_next, play;
  input [19:0] step_size;
  input [1:0] volume;
  output sample_ready, [17:0] sample;
Assume the instance of sample_calculator is as follows:
sample_calculator
  samcal(.clk(clk), .reset(reset), .enable(latch_sample), .sample_in(sine_rom_out), .quadrant(next_address[21]), .sample_out(sample_out));
assign sample = sample_out*volume;
```
8. [30 pts] Assume a computer system has a simple instruction set described as follows:

<table>
<thead>
<tr>
<th>Command</th>
<th>Instruction</th>
<th>Data</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Move2</td>
<td>01</td>
<td>--</td>
<td>Move data in register R1 to register R2.</td>
</tr>
<tr>
<td>Add</td>
<td>10</td>
<td>--</td>
<td>Add the content of R1 and R2 and store the result back to R1.</td>
</tr>
<tr>
<td>Shift</td>
<td>11</td>
<td>M[2:0]</td>
<td>Left shift the content of R1 by M[2:0] bits and store the result back into R1.</td>
</tr>
</tbody>
</table>

The registers block has two 6-bit input data ports: M[5:0] and D[5:0]; the former is from the memory source and the latter is from the datapath. The communication between R1 and R2 happens inside the registers block. The Control signals R1_en and R2_en are the enable signals for R1 and R2 respectively. Control signal R1_sel is used to select the source of R1 (either M or D busses).

![Diagram of the registers block](image)

a) [10 pts] Write the truth table of the instruction decoder. Use the 2-bit instruction as input and R1_en, R2_en, R1_sel signals as outputs.

<table>
<thead>
<tr>
<th>2-bit</th>
<th>R1_en</th>
<th>R2_en</th>
<th>R1_sel</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>1</td>
<td>x</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
b) [10 pts] Complete the following table which describes the function of the registers block:

<table>
<thead>
<tr>
<th>R1_en</th>
<th>R2_en</th>
<th>R1_sel</th>
<th>clk</th>
<th>R1[5:0]</th>
<th>R2[5:0]</th>
<th>Instruction(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>↑</td>
<td>M[5:0]</td>
<td>No Change</td>
<td>Move 1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>↑</td>
<td>D[5:0]</td>
<td>No Change</td>
<td>Add, shift</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>X</td>
<td>↑</td>
<td>No change</td>
<td>M[5:0]</td>
<td>Move 2</td>
</tr>
</tbody>
</table>

c) [10 pts] Complete the following program that performs the multiply-by-33 function using four instructions provided. By the end of the program you should have 33* a[5:0] stored in R1. Assume there is no overflow.

move1 a[5:0]     // move data a[5:0] into R1
move2 a[5:0]  //move data a[5:0] into R2
shift 000101   //left shift 5 bits
add     //add R1 and R2
9. [20 pts] Implement a sequential multiplier that performs multiplication by doing shift and add. The multiplicand A has 4 bits and the multiplier B has 2 bits. The result is $F = A \times B$.
(a) [10 pts] Show the high level RTL implementation.
(b) [10 pts] Draw a diagram of control and data path; show all connections. Give the detailed implementation for the datapath.
10. [20 pts] A flip-flop chain circuit is shown below. Assume that all flip-flops start with \( Q_0=Q_1=Q_2=Q_3=0 \). Setup and hold times are zero. The clock has a 50\% duty cycle. D-FF propagation delay satisfies the clock period condition.

a) [10pts] Draw a timing diagram for the circuit. Include \( \text{out}[3:0] \), D0, D1, D2, D3 and clk.
b) [5pts] What does this circuit do?

It is a counter. The initial value is 0. After the first clock rise edge it becomes 15, and counts down by 1 for every clock cycle.

c) [5pts] What is not good about this circuit? How could you improve it?

The output [3:0] has glitches due to D-ff propagation delay since the output of lower bit serves as clock input for higher bit.

To eliminate the glitches, all D-ffs should connect to the same clock input and the D input of each D-ff should be generated by combinational circuit.
11.  [30pts] The circuit shown below includes three 16 bit registers (labeled R0, R1, and ACC) implemented with positive-edge triggered D flip-flops, an 16-bit adder, and 2-to-1 multiplexors.

![Circuit Diagram]

a)  [5 pts] The table below has one row for each clock cycle of interest. Given values of S0, S1, S2, S3, fill out what is stored in each register after the rising edge of the clock.

<table>
<thead>
<tr>
<th>Cycle #</th>
<th>S0</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>R0</th>
<th>R1</th>
<th>ACC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>R0</td>
<td>R0</td>
<td>Acc + R0</td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>R0</td>
<td>R1</td>
<td>Acc + R1</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>R0</td>
<td>R1</td>
<td>Acc + R0</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>R0</td>
<td>ACC</td>
<td>Acc + R0</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>ACC</td>
<td>R1</td>
<td>Acc + R1</td>
</tr>
<tr>
<td>14</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
b) [10 pts] Add to the original design shown below minimum logic elements needed to implement a subtract operation so that ACC = ACC – R0. Define all signal values (S0-3) and any new signals you introduce as well.
module divide_by_subtract(clk, r0, r1, acc, start, s0, s1, s2, s3, s4);
input clk, start;
inout [w-1:0] r1, r0, acc;
output s0, s1, s2, s3, s4;
reg [w-1:0] q;
reg [2:0] state;

parameter Start = 3'b000, Compare = 3'b001, Subtract = 3'b010, Finish = 3'b011, Done = 3'b100;

always@(posedge clk) begin
  if(start)
    state <= Start;
  else begin
    case(state)
      Start : begin
        q <= 0;
        state <= Compare;
      end
      Compare: begin
        if(r0 > acc) state <= Done;
        else state <= Subtract;
      end
      Subtract: begin
        s0 <= 1; s1 <= 1; s2 <= 0; s3 <=0; s4 <= 1;
        q <= q+1;
        state <= Compare;
      end
      Finish: begin
        s0 <= 1; s1 <= 0; s2 <= 0; s3 <= 1; s4 <= 0;
        acc <= q;
        state <= Done;
      end
      Done: begin
        state <= Done;
      end
    endcase
  end
end

[15 pts] Starting with the original design, write the Verilog code that implements divide-by-subtraction operation with minimum additional logic. Assume ACC holds the dividend, and R0 has the divisor. At the end of the operation ACC holds the result of the division ACC/R0 and R1 has the remainder.
end
end
endmodule