M A S S A C H U S E T T S   I N S T I T U T E   O F   T E C H N O L O G Y
DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

6.111 Introductory Digital Systems Laboratory

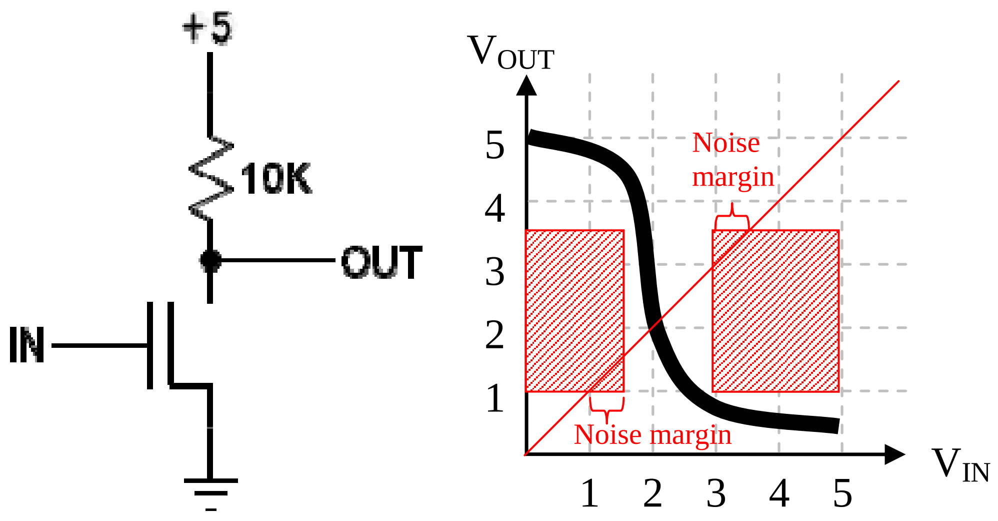
Fall 2005

Midterm Exam: November 2, 2005

Name ANSWER KEY	*Score* *(out of 100)*

Problem 1. (25 Points)

Congratulations, you're now a consultant for Hot Logic, Inc., a startup interested in building combinational logic devices that (of course) obey the static discipline. Their chief engineer has given you the following schematic and voltage-transfer curve for their prototype inverter design, the HL-INV:



A. (10 Points) Choose values for V_{OL}, V_{IL}, V_{IH} and V_{OH} so that the prototype device obeys the static discipline and has positive, non-zero noise margins. Give your values below or briefly explain why no such values can be found.

Briefly explain why values can't be found or specify voltage levels: V_{OL} 1.0

Other values are possible…

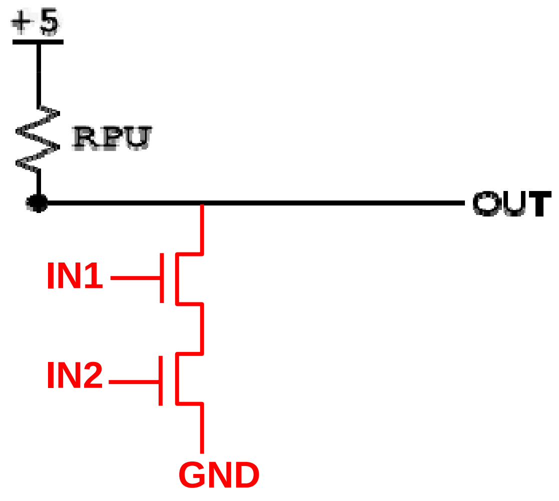
V_{IL} 1.5

V_{IH} 3.0

V_{OH} 3.5

Noise margins 0.5

B. (5 Points) Add the appropriate n-channel mosfets to the schematic below to implement a two-input HL-NAND gate with inputs IN1 and IN2 that computes $\overline{IN1 \bullet IN2}$.



C. (5 Points) Assume that all mosfets in Hot Logic's gates have the same width and length. If we want the V_{OL} for the HL-NAND gate to be the same as the V_{OL} for the HL-INV gate, what value should we choose for RPU?

V_{OL} is proportional to RPD/RPU so if we double RPD we have to double RPU to keep V_{OL} unchanged.

Value for RPU (in ohms): 2*10K = 20K

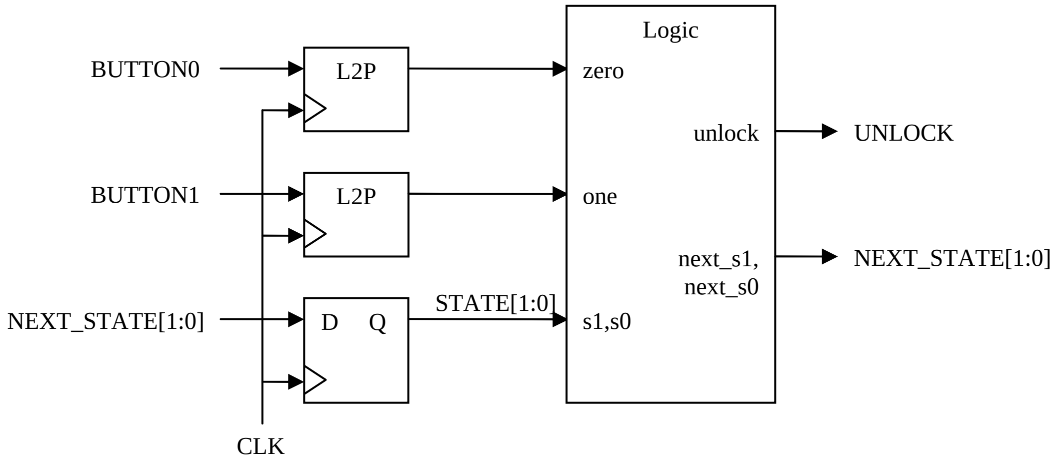
D. (5 Points) Briefly compare the static power dissipation and noise margins of Hot Logic gates with CMOS gates.

In the steady state Hot Logic gates dissipate considerable static power when they output a "0" since there is current flowing through the pullup and pulldown paths. CMOS dissipates essentially zero static power.

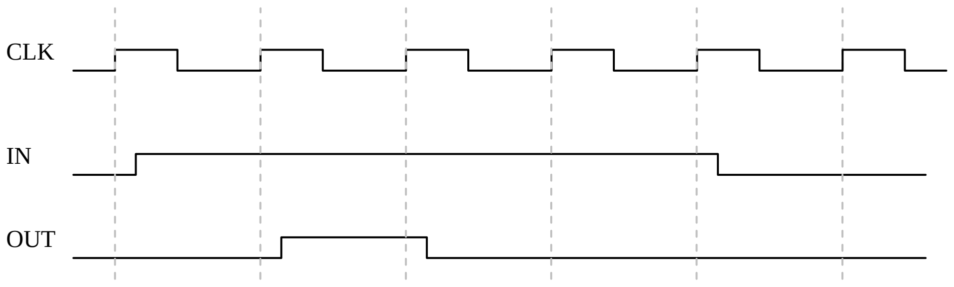
CMOS has better noise margins because CMOS gates have higher gain in the middle region of the voltage transfer curve (Vil and Vih can move towards each other) and Vol = 0V, Voh=power supply (maximizing the noise margins).

Problem 2. (60 Points)

The diagram below shows the schematic for an electronic lock with a combination consisting of some sequence of 0's and 1's. The user enters the combination by pressing the ZERO and ONE buttons in the correct sequence. If the last N digits that have been entered match the N-digit combination, the UNLOCK signal is asserted and the lock opens. You can assume that the button0 and button1 signals have been properly debounced and synchronized with CLK.



A. (10 Points) The L2P module is a level-to-pulse converter that asserts its output high for 1 clock cycle after a 0-to-1 transition on its input, as shown in the timing diagram below.



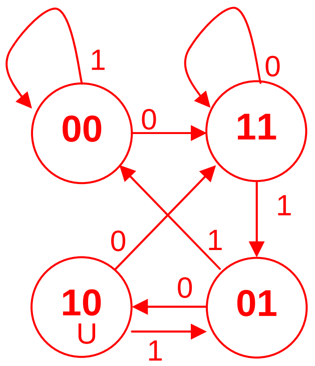
Complete the following Verilog implementation for the L2P module:

```
module L2P(clk,in,out);
  input clk;
  input in;
  output out;

  // remember value of IN at last clock edge
  reg old,out;
  always @ (posedge clk) begin
    old <= in;
    out <= in & ~old;
  end

endmodule
```

Here's the truth table for the lock's Logic module:



s1	*s0*	*one*	*zero*	*unlock*	*next_s1*	*next_s0*
0	**0**	**0**	**0**	**0**	**0**	**0**
0	**0**	**0**	**1**	**0**	**1**	**1**
0	**0**	**1**	**0**	**0**	**0**	**0**
0	**0**	**1**	**1**	**0**	**1**	**1**
0	**1**	**0**	**0**	**0**	**0**	**1**
0	**1**	**0**	**1**	**0**	**1**	**0**
0	**1**	**1**	**0**	**0**	**0**	**0**
0	**1**	**1**	**1**	**0**	**1**	**0**
1	**0**	**0**	**0**	**1**	**1**	**0**
1	**0**	**0**	**1**	**1**	**1**	**1**
1	**0**	**1**	**0**	**1**	**0**	**1**
1	**0**	**1**	**1**	**1**	**1**	**1**
1	**1**	**0**	**0**	**0**	**1**	**1**
1	**1**	**0**	**1**	**0**	**1**	**1**
1	**1**	**1**	**0**	**0**	**0**	**1**
1	**1**	**1**	**1**	**0**	**1**	**1**

B. (10 Points) What's the N-digit combination for the lock, i.e., what sequence of 0's and 1's will leave you with UNLOCK asserted regardless of what state the FSM was in when you started to enter the combination? Hint: use the truth table to draw a state transition diagram. You can use the back of the quiz pages as scratch paper.

Lock combination: 010

C. (5 Points) Briefly describe what happens if both buttons are pressed simultaneously.

The lock behaves as if only the ZERO button had been pressed, i.e., the ZERO button overrides the ONE button.

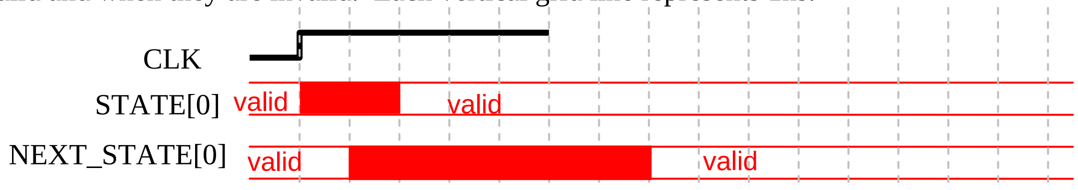
D. (8 Points) Here are the timing parameters for the two D-Registers used to hold the current state: t_{CD} = 0ns, t_{PD} = 2ns, t_{SETUP} = 2ns, t_{HOLD} = 1ns. What are the constraints on t_{CD} and t_{PD} of the Logic module if we want the circuit to operate reliably with a 100MHz clock?

$t_{CD,R} + t_{CD,LOGIC} > t_{HOLD}$
$t_{PD,R} + t_{PD,LOGIC} + t_{SETUP} < t_{CLK}$

Constraints on Logic module's t_{CD}: $t_{CD,LOGIC} > 1ns$

Constraints on Logic module's t_{PD}: $t_{PD,LOGIC} < 6ns$

E. (6 Points) For this question assume the register timing specifications given in part (D) and that the timing specifications of the Logic module are t_{CD} = 1ns and t_{PD} = 5ns. Complete the timing diagram below carefully indicating when STATE[0] and NEXT_STATE[0] are valid and when they are invalid. Each vertical grid line represents 1ns.



F. (6 Points) If the Logic module is implemented using a ROM, what size ROM would we need?

4 address lines
3 outputs

Number of locations in ROM: $2^4 = 16$

Number of bits/location in ROM: 3

G. (10 Points) Give a minimized sum-of-products expression for next_s[1] output of the Logic module. Hint: the Karnaugh Map template given below may be useful.

Minimized sum-products expression for next_s[1]: **zero | (s1 & (~one))** pardon my Verilog!

one/zero

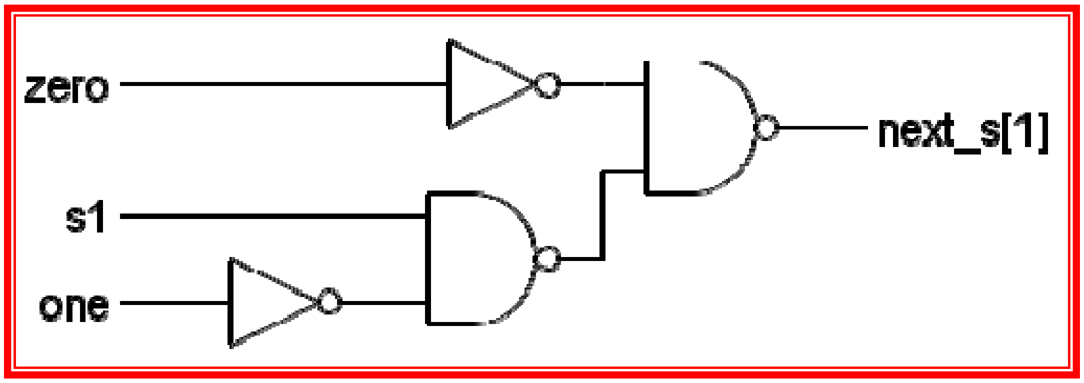
s1/s0

	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	1	1	1	0
10	1	1	1	0

zero

s1 & ~one

H. (5 Points) Draw a schematic diagram for a circuit that implements next_s[1]. Your circuit should using only inverters, 2-input NAND gates and 2-input NOR gates.



Problem 3. (15 Points)

The following Verilog test jig, initially sets a to 1, b to 0 and clk to 0, waits 10 time units, sets clk to 1, waits another 10 time units and the prints out the values of a, b and c.

```
module testjig();
  reg clk,a,b;
  wire c;

  assign c = ~a;

  // CODE SNIPPETS ARE INSERTED HERE

  initial begin
    a = 1;
    b = 0;
    clk = 0;
    #10    // wait 10 time units
    clk = 1;
    #10    // wait 10 time units
    $display(“a=%d b=%d c=%d\n”,a,b,c);
    $stop;
  end
endmodule
```

A series of tests is run, each using one of the code snippets below substituted into the testjig module at the indicated location. For each snippet indicate the values printed out for a, b and c. Write ??? for a value if it cannot be determined using standard Verilog semantics.

C = ~A might or might not happen here.

A. `always @ (posedge clk) begin a = b; b = c; end`

Values: a = 0 **, b =** ??? **, c =** 1

B. `always @ (posedge clk) begin a <= b; b <= c; end`

Values: a = 0 **, b =** 0 **, c =** 1

C.
```
always @ (posedge clk) a = b;
always @ (posedge clk) b = c;
```
C = ~A might or might not happen here.

Values: a = 0 **, b =** ??? **, c =** 1

D.
```
always @ (posedge clk) a <= b;
always @ (posedge clk) b <= c;
```

Values: a = 0 **, b =** 0 **, c =** 1

E. `always @ (posedge clk) begin a <= b; b = c; end`

Values: a = 0 **, b =** 0 **, c =** 1