University of California at Berkeley
College of Engineering
Department of Electrical Engineering and Computer Sciences

EECS151/251A J. Wawrzynek
Spring 2020 3/12/20

Exam 1 Solutions

Name: _____

Student ID number: _____

Class (EECS151 or EECS251A): _____

You have two hours to take the exam. This exam comprises a set of questions with 1 point per expected minute of completion with a total of approximately 90 points. You have two hours to complete the exam. As with homework problems, submit your solutions using Gradescope. At the end of the exam time, we will give you a few extra minutes for you to submit your answers.

You are allowed to refer to your notes, the class lecture notes, and any other reference materials that you have available. You are not allowed to speak or communicate with anyone on any topic related to the exam during the exam period. After completing the exam, sign the following statement attesting that you did not discuss the exam problems with anyone else. You may either scan this page or copy the statement word-for-word.

I hereby declare that I have not spoken with nor otherwise communicated with anybody regarding the content of this exam while taking the exam:

(sign here): _____

For each problem if you find yourself taking excessive time to work out a solution consider skipping the problem or a fresh approach. Also, start by answering the easier questions and then move on to the more difficult ones.

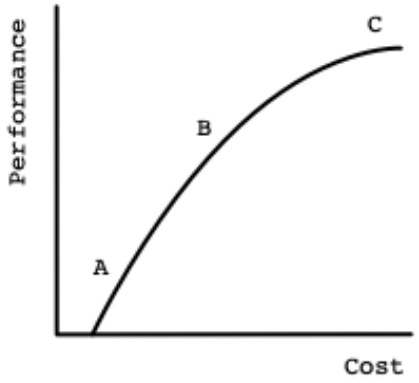
Neatness counts. We will deduct points if we need to work hard to understand your answer.

Before you turn in your exam, write your student ID number on all pages.

1. Pareto Optimality [5pts]

   Processors, as with all digital designs, exist at different points along the Pareto optimal frontier curve for cost and performance. List and briefly discuss three examples of processors in common use and describe approximately where they lie on the curve and how and why they make a tradeoff between cost and performance. Try to span a wide range of the tradeoff curve.

   Solution:

   (A) Low Cost and Low Performance: for example, calculators are very cheap and do not require high processing power, and therefore lie in the "A" region.

   (B) Medium Cost and Medium Performance: laptops and PCs are good examples of medium-priced electronics with higher processing power than calculators.

   (C) High Cost and High Performance: super computers and quantum computers are very expensive but deliver very high performance.

2. IC Costs [5pts]

   You work at a company that plans to design and sell an ASIC. Your NRE costs are \$1M and your production costs per packaged IC is \$10.

   (a) You expect to sell 100,000 ICs at a price of \$100 each. If you did, what would be your profit per IC?

   Solution:

   $$\text{profit} = \frac{100 \cdot 100,000 - 10 \cdot 100,000 - 1,000,000}{100,000} = \$80 \text{ per chip}$$

   (b) You are also considering offering a FPGA version instead of ASIC. In this case, NRE costs would be \$100,000 with a \$50 per part cost, and you would still plan to sell the design at a price of \$100 each. Would this option generate more profit? Justify your answers.

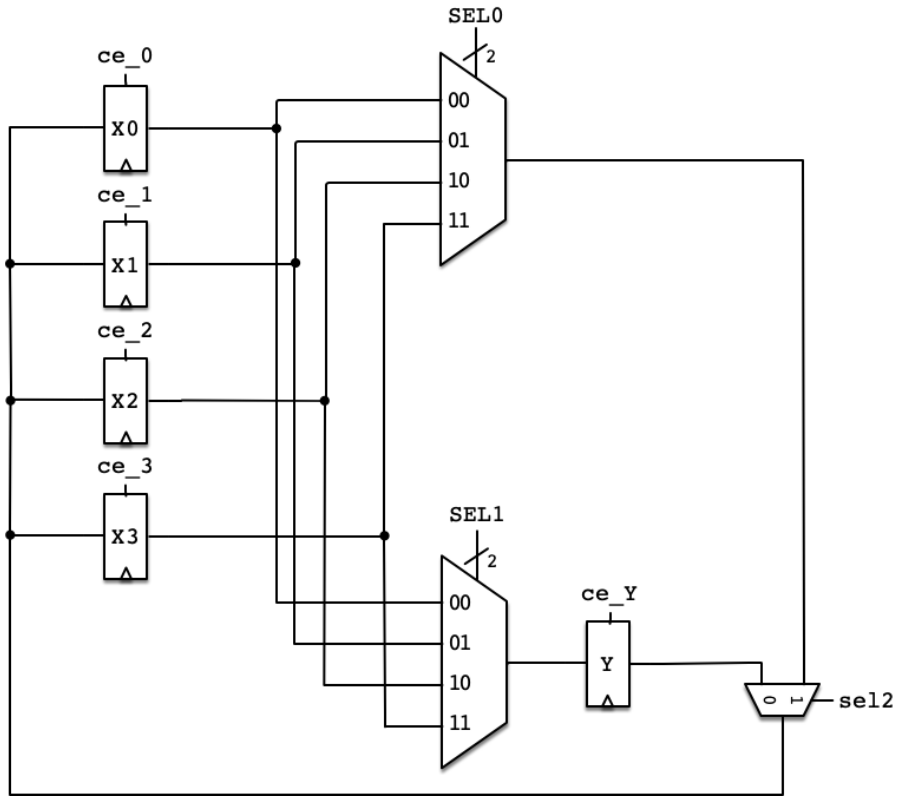
Solution:

$$\text{profit} = \frac{100 \cdot 100,000 - 50 \cdot 100,000 - 100,000}{100,000} = \$49 \text{ per chip}$$

At this volume, it is more profitable to go with the ASIC option.

3. Register Transfers [8pts]

In the circuit shown below, registers X0 through X3 hold the values a, b, c, d, respectively. Register Y is uninitialized. All registers are equipped with a clock enable input. Your job is to find the minimum number of cycles needed to reverse the ordering of the values in the X registers. Show your work and your final answer.



Solution:

We basically just need to swap X0 and X3, X1 and X2.

```
C1: Y <= X0; X0 <= X3
C2: Y <= X1; X3 <= Y
C3: Y <= X2; X2 <= Y
C4: X1 <= Y
```

4 cycles is needed.

5 cycles is also acceptable if you reason that the last cycle is for final register

update (or 4 clock edges).

4. Adder Logic Design [8pts]

An alternative way to implement the function of a full-adder cell is the following. We introduce two intermmediate signals:

$$p = a \oplus b$$
$$g = ab$$

and use them to generate the usual full-adder outputs:

$$r = p \oplus c_{in}$$
$$c_{out} = g + pc_{in}$$

Show that the logic functions of the two outputs is the same as those of the full-adder presented in lecture.

Solution:

From lecture:

$$r = a \oplus b \oplus c_{in}$$

$$c_o = ab + c_{in}(a + b)$$

a	b	c_{in}	r	c_o
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

a	b	p	g	c_{in}	r	c_o
0	0	0	0	0	0	0
0	0	0	0	1	1	0
0	1	1	0	0	1	0
0	1	1	0	1	0	1
1	0	1	0	0	1	0
1	0	1	0	1	0	1
1	1	0	1	0	0	1
1	1	0	1	1	1	1

The truth tables yield the same results, so the logic functions of the two outputs is the same as those of the full-adder presented in lecture.

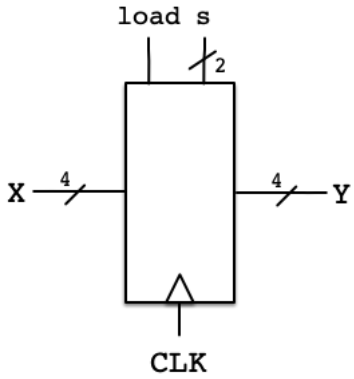
5. Sequential Circuits [16pts]

Below are the interface specification and the detailed diagram for a sequential circuit.

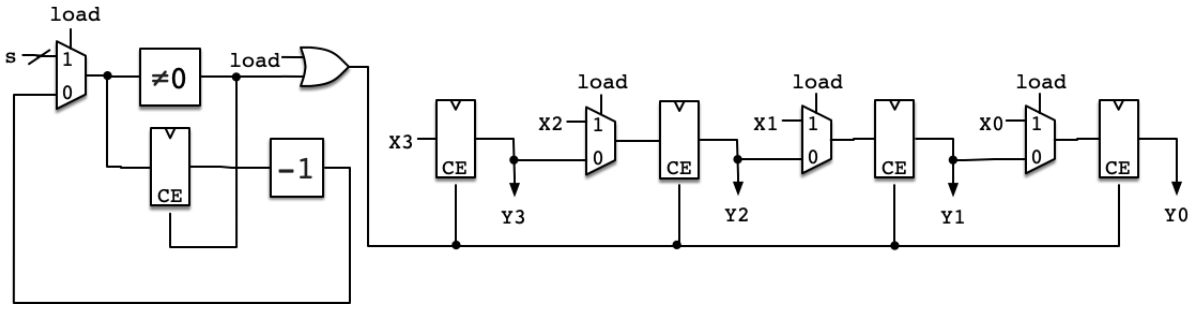
(a) Briefly describe the function of the circuit. (What is it used for?)

(b) Draw the clock waveform and waveforms for X, Y, s, and *load*, to demonstrate the circuit behavior. Assume that $X = 4'b0110$ and $S = 2'b10$.

(c) Write the Verilog generator description of the circuit where the width of X and Y, and of S are parameterized. To help clarity, use a hierarchical description.



Top Level Circuit



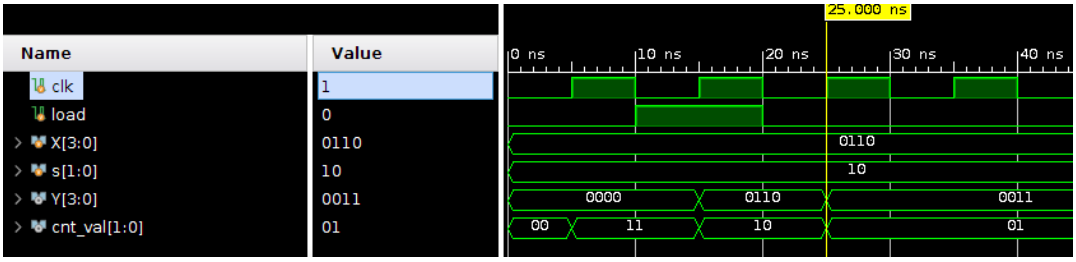
RTL Implementation

Solution:

(a) A shift register that performs arithmetic right shift $s-1$ times.

Solution:

(b)



Solution:

(c)

```
module Q5 #(
  parameter XW = 4,
  parameter SW = 2
```

```
) (
  input clk,
  input load,
  input [XW-1:0] X,
  input [SW-1:0] s,
  output [XW-1:0] Y
);

  wire [SW-1:0] cnt_val, cnt_next;
  wire cnt_ce;
  REGISTER_CE #(.N(SW)) cnt (
    .q(cnt_val),
    .d(cnt_next),
    .ce(cnt_ce),
    .clk(clk));

  wire [XW-1:0] Y_val, Y_next;
  wire Y_ce;
  REGISTER_CE #(.N(XW)) shift_reg (
    .q(Y_val),
    .d(Y_next),
    .ce(Y_ce),
    .clk(clk));

  assign cnt_next = load ? s : cnt_val - 1;
  assign cnt_ce   = cnt_next != 0;

  assign Y_next   = load ? X : {X[XW-1], Y_val[XW-1:1]};
  assign Y_ce     = load | (cnt_next != 0);

  assign Y = Y_val;

endmodule
```

6. FPGAs Logic Block [5pts] Using nothing other than 3-LUTs, demonstrate how you would construct a 6-LUT. Label your inputs $x_0, x_1, ..., x_5$, and output as y.

Solution:

A 3-LUT can be used as a 2-to-1 MUX (2 input plus 1 input selection).

Two 3-LUT with a 2-to-1 MUX forms a 4-LUT → We need three 3-LUTs to build a 4-LUT.

Two 4-LUT with a 2-to-1 MUX forms a 5-LUT → We need two 4-LUT with one 3-LUT to build a 5-LUT.

Two 5-LUT with a 2-to-1 MUX forms a 6-LUT → We need two 5-LUT with one 3-LUT to build a 6-LUT.

Therefore, we will need $(3 \times 2 + 1) \times 2 + 1 = 15$ 3-LUTs to build a 6-LUT.

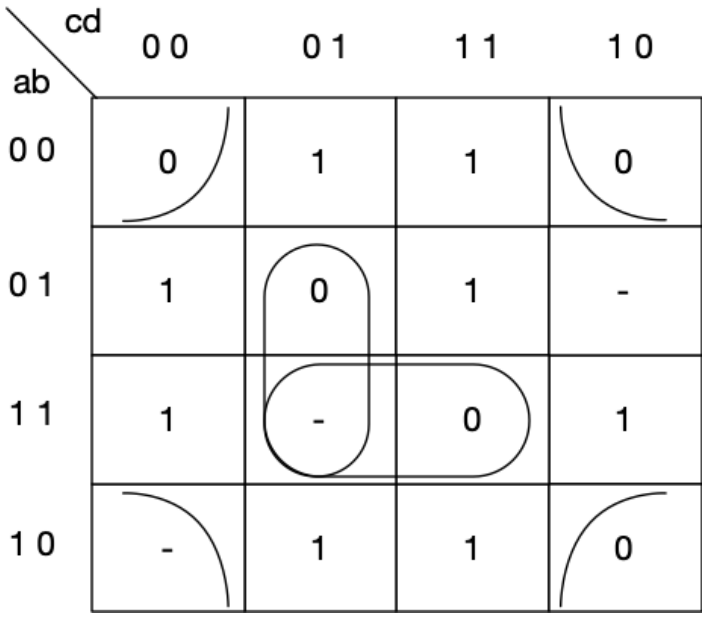
7. Boolean Logic [7pts]

For the following function expressed as a truth table:

a	b	c	d	f
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	-
0	1	1	1	1
1	0	0	0	-
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	-
1	1	1	0	1
1	1	1	1	0

(a) Express f in reduced product-of-sums (POS) form.

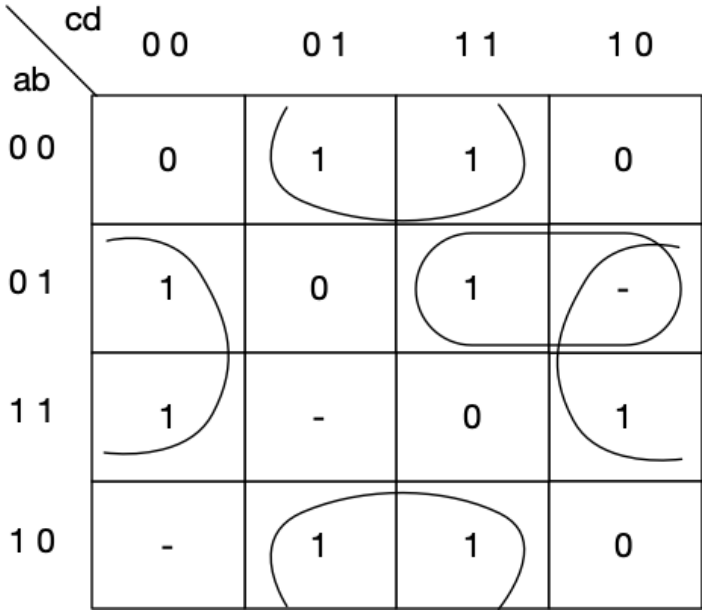
Solution:



$$f = (b + d)(\overline{b} + c + \overline{d})(\overline{a} + \overline{b} + \overline{d})$$

(b) Express $\overline{f}$ in reduced product-of-sums (POS) form.

Solution:

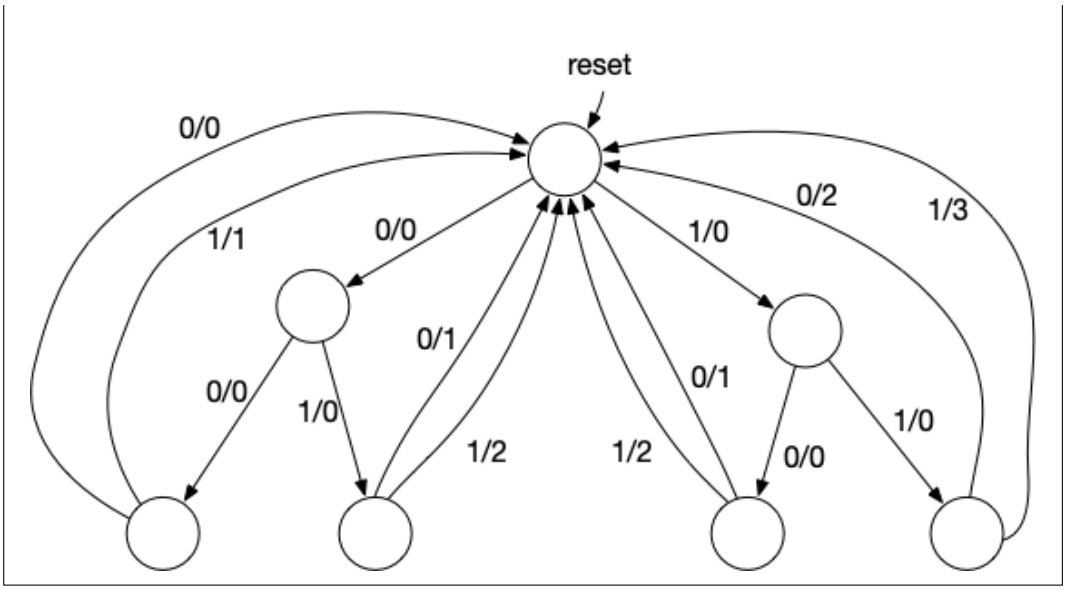


$$\overline{f} = (b + \overline{d})(\overline{b} + d)(a + \overline{b} + \overline{c})$$

8. Finite State Machines [6pts] We would like to design an FSM that accepts a string of bits, one per clock cycle. After each group of three bits, the FSM outputs a 2-bit integer that indicates how many 1's were in that group of 3 input bits. Then it keeps going with the next group of 3 input bits. (*Don't worry about minimizing the number of states.*)

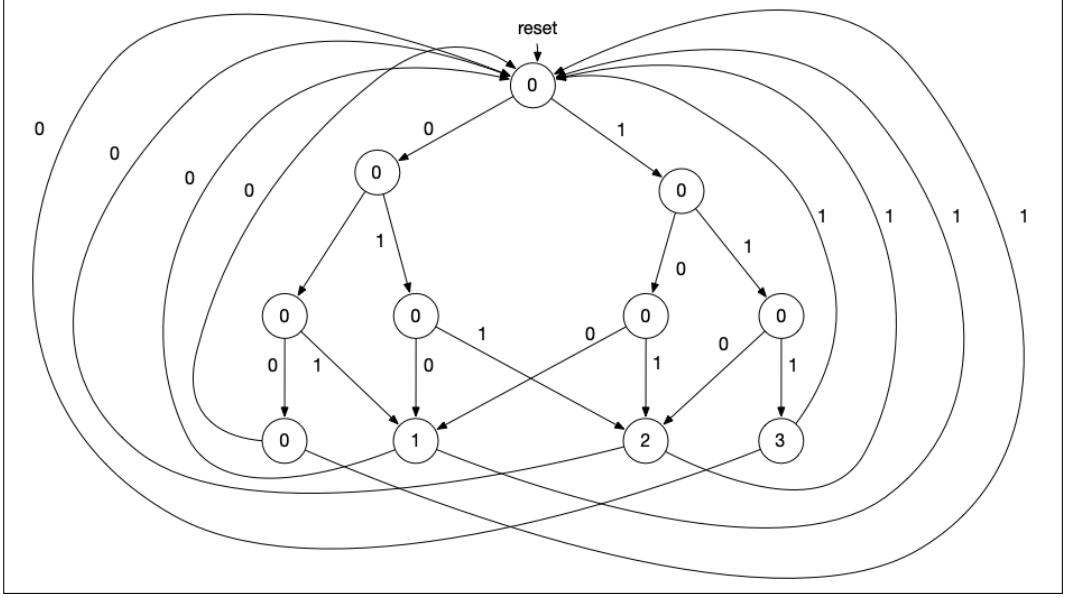
(a) Draw the state transition diagram for a Mealy machine that implements this function.

Solution:



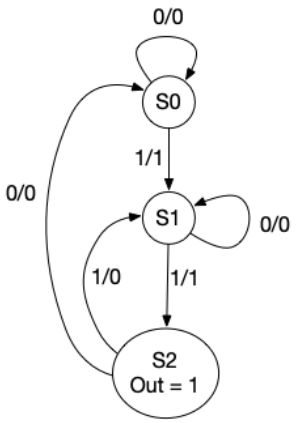
(b) Draw the state transition diagram for a Moore machine that implements this function.

Solution:



9. Finite State Machines [5pts]

Write the Verilog to describe an FSM whose behavior is shown in the state transition diagram shown below. Note that this machine has both Mealy and Moore outputs.



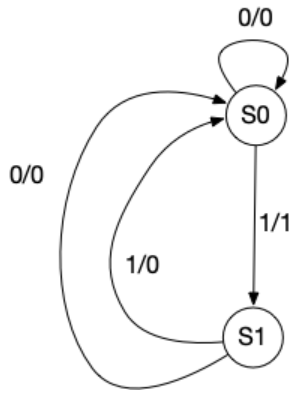
Solution:

```
module FSM9 (
  input clk,
  input rst,
  input in,

  output out_moore,
  output reg out_mealy
);

  localparam S0 = 2'b00;
  localparam S1 = 2'b01;
  localparam S2 = 2'b10;

  wire [1:0] state_val;
  reg [1:0]  state_next;

  REGISTER_R #(.N(2), .INIT(S0)) state_reg (
    .q(state_val),
    .d(state_next),
    .rst(rst), .clk(clk));

  // Moore output: depends on current state only
  // Mealy output: depends on current state and input

  // Just go with (*) for sensitivity list
  always @(*) begin
    // Don't forget default assigment statements
    state_next = state_val;
    out_mealy = 1'b0;

    case (state_val)
```

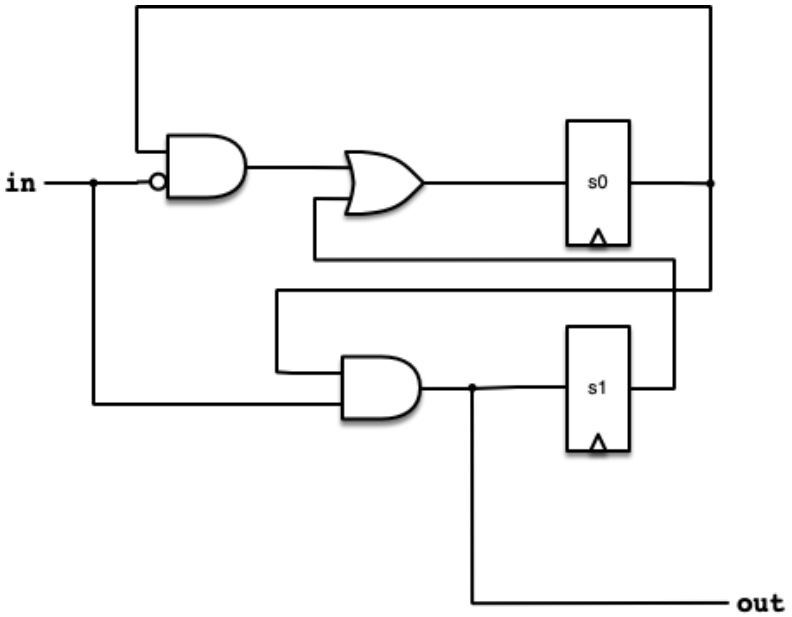
```
      S0: begin
        if (in == 1) begin
          state_next = S1;
          out_mealy  = 1'b1;
        end
      end

      S1: begin
        if (in == 1) begin
          state_next = S2;
          out_mealy  = 1'b1;
        end
      end

      S2: begin
        if (in == 0)
          state_next = S0;
        else
          state_next = S1;
      end

    endcase
  end

  assign out_moore = (state == S2);

endmodule
```

10. One-hot Encoding [5pts]

    Draw the circuit diagram that implements the FSM shown below using *one-hot state encoding*. Minimize the number of logic gates.

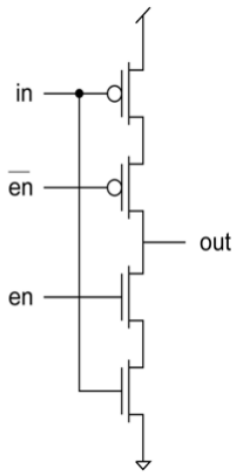


Solution:



11. Gate Delay [10pts]

Derive the propagation delay τ_p for the tri-state inverter circuit (shown below), based on the "in" input. Assume as we did in lecture that nfets (NMOS transistors) are twice as strong as pfets (PMOS transistors) per unit width, and that we would like to have balanced pull-up and pull-down delay. Leave your answer in terms of the unit inverter delay, τ_{p0}.



Solution:

To get the same C_{in} as an inverter we size the PMOS to be 2 and NMOS to be 1.

By inspection, the delay would then be twice that of the unit inverter since it

drives the same capacitance but with twice the drive resistance.

$$\tau_p = 2\tau_{p0}(1 + \frac{f}{\gamma})$$

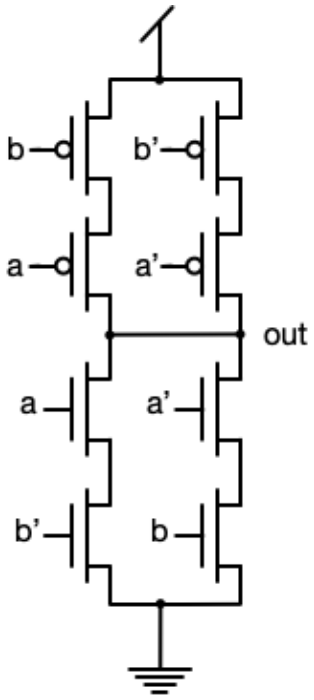
Full derivation:

$$\begin{aligned}\tau_p &= 0.69(2\frac{R_n}{W})\Big[3WC_d + C_L\Big] \\ &= 0.69\frac{R_n}{W}\Big[6W\gamma C_g + 2C_L\Big] \\ &= 0.69(3R_n\gamma C_g)\Big[2 + 2\frac{C_L}{3W\gamma C_g}\Big] \\ &= \tau_{p0}(2 + 2\frac{f}{\gamma})\end{aligned}$$

12. CMOS Logic Gates [6pts]

Derive the static CMOS logic gate implementation of the exclusive-NOR function of two inputs, a and b. (It's exclusive-or with inverted output). You may assume you have inverted and non-inverted inputs.

Solution:



13. Process Scaling [5pts]

Consider the effect of Dennard Scaling on wire delay. In this problem we will assume that all process dimensions, including wire thickness, scale down by a factor κ as proposed by Dennard.

What would be the effect on the delay of long wires? Show your work and justify your answer.

Solution:

The total resistance of the wire is

$$R_w = \frac{\rho L}{A_c}$$

where A_c is the cross sectional area of the wire. L scales by κ and A_c scales by κ^2 since the wire thickness also scales so the new resistance is

$$R'_w = \frac{\rho \frac{L}{\kappa}}{\frac{A_c}{\kappa^2}} = R_w \kappa$$

The total capacitance of the wire is

$$C_w = \frac{\epsilon A}{d}$$

where A is the area of the wire facing the ground plane or another wire. It can be rewritten in terms of thickness (W) and length (L)

$$C_w = \frac{\epsilon W L}{d}$$

W, L and d all scale by κ so the new capacitance is

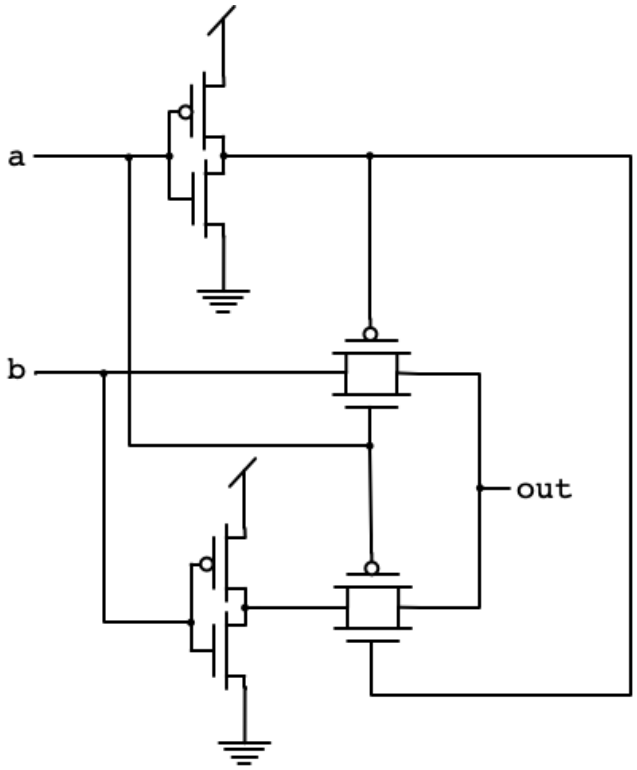
$$C'_w = \frac{\epsilon \frac{W}{\kappa} \frac{L}{\kappa}}{\frac{d}{\kappa}} = \frac{C_w}{\kappa}$$

Because wire delay is proportional to $R'_w C'_w$ and $R'_w C'_w = R_w \kappa \frac{C_w}{\kappa} = R_w C_w$, then wire delay does not change.

14. Logic Gates [5pts] **251A only** — *Optional* **Challenge Question for 151**

Again consider the design of the exclusive-NOR gate. Assuming inverted inputs are not available, but also that you are *not* restricted to *static CMOS implementation.* What is the minimum number of transistors needed to implement this gate? Justify your answer.

Solution:



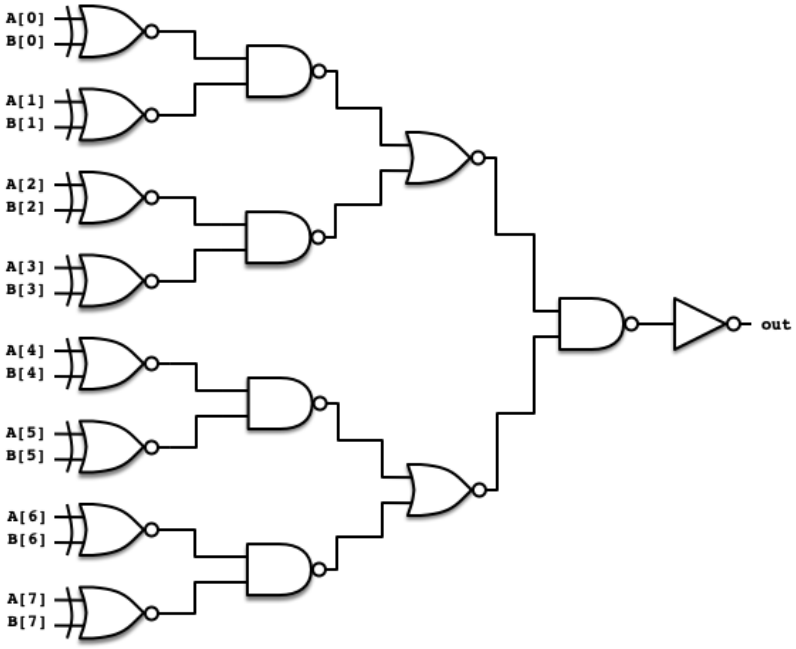
This solution requires 8 transistors compared to the 12 required for the static CMOS implementation.

15. Combination Logic [8pts] **251A only** — *Optional* **Challenge Question for 151**

Consider the design of a combinational logic block for equal compare of two N-bit integers, A and B. Assume N is always a power of 2. You are allowed to use inverters, and 2-input NAND, NOR, XOR, and XNOR gates. For this problem, assume that all of these gates have delay = 1. Design a circuit that implements the equal compare function and has minimal delay. Show your design for N=8. In general, what is the delay of such a circuit?

Solution:

For $N = 8$:



In general you can build an N-bit comparator block with 1 stage of N 2-input XNOR gates to compare each bit from A and B. You then need $\log_2(N)$ stages to build an N-bit NAND, and then finally an inverter for a total delay of

$$t_d = 2 + \log_2(N)$$