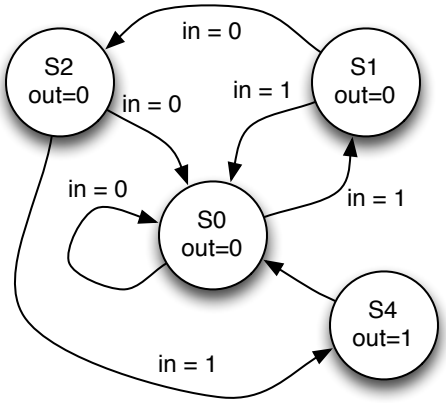
EECS 151/251A Homework 4

Due Wednesday, February 14th, 2018

Problem 1: More Verilog

1. In the space below write out the Verilog code for a module that implements a finite state machine with the behavior of the following state transition diagram:

> No canonical solution available yet. We're looking for state machines defined in the style introduced in the lecture and in the lab/supplementary materials.



2. Will this Verilog snippet result in any sequential elements? Why?

```
module m(input [2:0] b, output a);
  always @(a or b) begin
    case (b)
      2'b01: a = 1'b1;
      2'b10: a = 1'b1;
    endcase
  end
endmodule
```

> Not elaborating every value of the case statement, or using a `default`, will probably cause the tool to insert a latch to hold `a`'s last value. That is, the value of `a` is not clear as a purely-combinational function of the inputs.

3. What difference is there, if any, between how simulators and synthesizers targeting hardware handle the special value `1'bx`?

> Synthesiers tend to assign it a value that simplifies the circuit. Simulators will propagate the X value itself throughout the simulation. You can't be sure that simulated and the synthesised circuits will behave consistently.

4. **EECS 251A Only.** Using flip-flops, simple gates, and multiplexors, draw a diagram for the circuit resulting from synthesizing the following Verilog code. Use individual flip-flops (not N-bit wide registers) and draw out every individual signal path.

```
module foo(ld, X, CNTL, clk, y);
  parameter N = 4;
  input ld, clk;
  input [2:0] CNTL;
  input [N-1:0] X;
  output y;
  parameter LOAD = 00, SHIFT = 01, COMP = 10;
  reg [N-1:0] S;
  wire [N-1:0] W;
  assign W = S ^ {0, W[N-1:1]};
  assign y = | W;
  always @ (posedge clk)
    case (CNTL)
      LOAD    : S <= X;
      SHIFT   : S <= X >> 1;
      COMP    : S <= W;
      default : S <= X;
    endcase
endmodule
```

> No canonical solution available yet.

Problem 2: Karnaugh Maps

Take this truth table consisting of 6 input variables and 1 output:

A	B	C	D	E	F	Out	A	B	C	D	E	F	Out
0	0	0	0	0	0	0	1	0	0	0	0	0	1
0	0	0	0	0	1	0	1	0	0	0	0	1	0
0	0	0	0	1	0	0	1	0	0	0	1	0	1
0	0	0	0	1	1	0	1	0	0	0	1	1	1
0	0	0	1	0	0	0	1	0	0	1	0	0	0
0	0	0	1	0	1	0	1	0	0	1	0	1	0
0	0	0	1	1	0	0	1	0	0	1	1	0	0
0	0	0	1	1	1	x	1	0	0	1	1	1	1
0	0	1	0	0	0	0	1	0	1	0	0	0	0
0	0	1	0	0	1	x	1	0	1	0	0	1	0
0	0	1	0	1	0	1	1	0	1	0	1	0	1
0	0	1	0	1	1	1	1	0	1	0	1	1	1
0	0	1	1	0	0	0	1	0	1	1	0	0	0
0	0	1	1	0	1	1	1	0	1	1	0	1	0
0	0	1	1	1	0	1	1	0	1	1	1	0	0
0	0	1	1	1	1	1	1	0	1	1	1	1	1
0	1	0	0	0	0	0	1	1	0	0	0	0	0
0	1	0	0	0	1	0	1	1	0	0	0	1	0
0	1	0	0	1	0	1	1	1	0	0	1	0	1
0	1	0	0	1	1	1	1	1	0	0	1	1	1
0	1	0	1	0	0	0	1	1	0	1	0	0	0
0	1	0	1	0	1	0	1	1	0	1	0	1	0
0	1	0	1	1	0	0	1	1	0	1	1	0	0
0	1	0	1	1	1	1	1	1	0	1	1	1	1
0	1	1	0	0	0	0	1	1	1	0	0	0	1
0	1	1	0	0	1	x	1	1	1	0	0	1	1
0	1	1	0	1	0	1	1	1	1	0	1	0	1
0	1	1	0	1	1	1	1	1	1	0	1	1	1
0	1	1	1	0	0	0	1	1	1	1	0	0	1
0	1	1	1	0	1	1	1	1	1	1	0	1	1
0	1	1	1	1	0	1	1	1	1	1	1	0	1
0	1	1	1	1	1	1	1	1	1	1	1	1	1

where x means “don’t care”.

1. Use a 6-variable Karnaugh Map (2D) to derive a simplified boolean function from this truth table using the Sum of Products method.

ABC \ DEF	000	001	011	010	110	111	101	100
000	0	0	0	0	0	x	0	0
001	0	x	1	1	1	1	1	0
011	0	x	1	1	1	1	1	0
010	0	0	1	1	0	1	0	0
110	0	0	1	1	0	1	0	0
111	1	1	1	1	1	1	1	1
101	0	0	1	1	0	1	0	0
100	1	0	1	1	0	1	0	0

2. Use a 6-variable Karnaugh Map (2D) to derive a simplified boolean function from this truth table using the Product of Sums method.

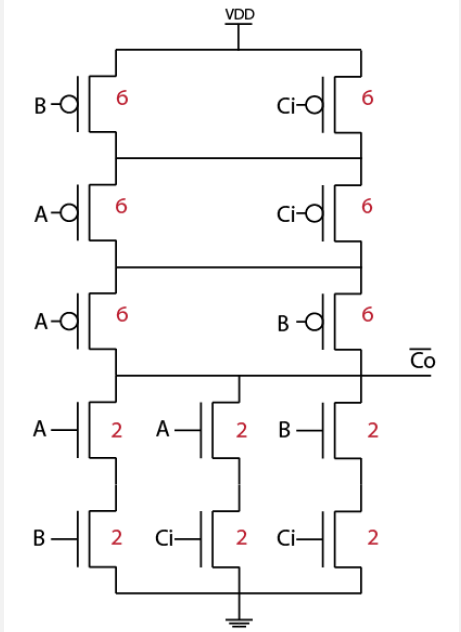
ABC \ DEF	000	001	011	010	110	111	101	100
000	0	0	0	0	0	x	0	0
001	0	x	1	1	1	1	1	0
011	0	x	1	1	1	1	1	0
010	0	0	1	1	0	1	0	0
110	0	0	1	1	0	1	0	0
111	1	1	1	1	1	1	1	1
101	0	0	1	1	0	1	0	0
100	1	0	1	1	0	1	0	0

3. Are your two answers to this question the same? Why or why not?

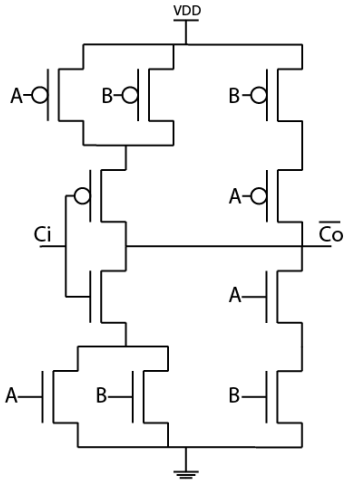
If you circled the any x value in only 1 of the previous parts, then the answer would be yes. However, to maximize the size of the cells circled in either case the x for ABC=000, DEF=111 will be a 1 in the SOP but a 0 in the POS and therefore they are not the same.

Problem 3: Static Complementary CMOS Logic

1. Implement the logic function $\overline{C_o} = \overline{AB + AC_i + BC_i}$ using a complementary pull-up and pull-down network.



2. A friend proposes the following implementation of the function in subquestion (1). Does this perform the same function as the gate from (1)? If so, what advantage does it have over your implementation in (1)? If not, what is the function it implements?



We construct the truth table for the given gate and find an expression for the output. As shown, the proposed circuit implements the same function.

A	B	C_i	$\overline{C_o}$
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

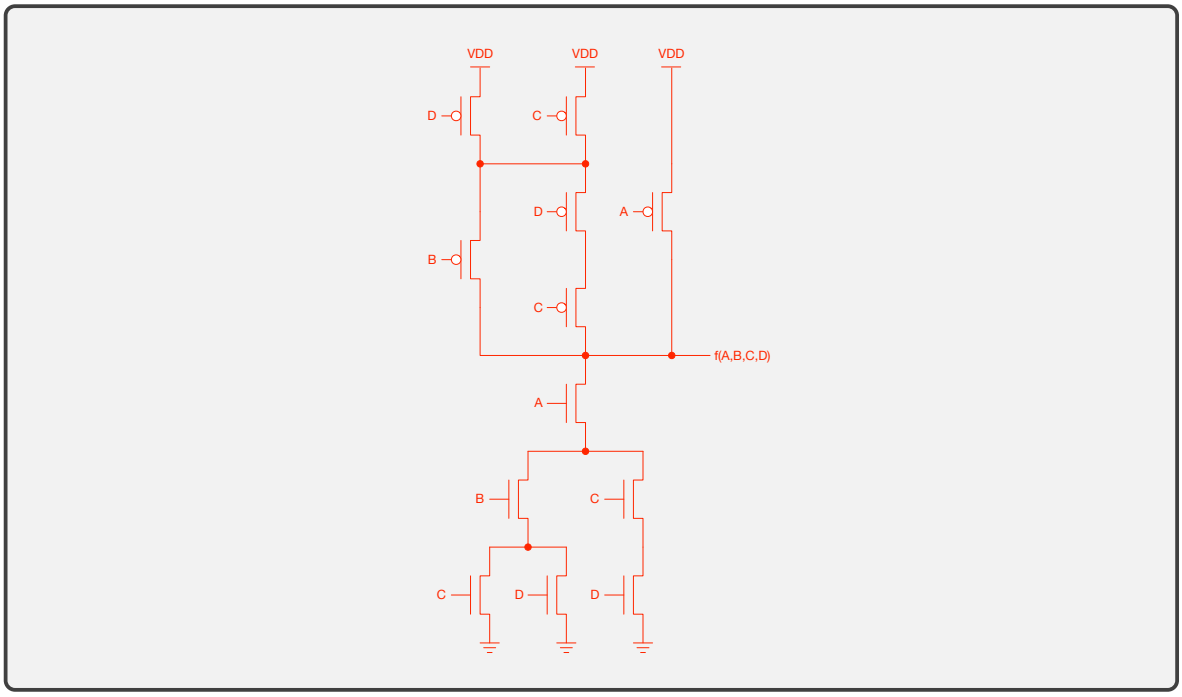
$$\begin{aligned}\overline{C_o} &= \overline{\overline{A}BC_i + A\overline{B}C_i + AB\overline{C_i} + ABC_i} \\ &= \overline{AB + AC_i + BCi}\end{aligned}$$

This gate has the advantage of using less transistors; as we will understand later, it also has the benefit of having smaller transistor stacks and thus less input capacitance on each input.

3. Is the gate shown in (2) a static CMOS gate? Explain your answer.

This is a static CMOS gate even though the pull-down and pull-up networks are not complementary. For any combination of inputs, C_o is always connected to V_{DD} or ground via a low impedance path and is never connected to both rails at the same time. The property of this logic equation that allows the pull-down and pull-up networks to be symmetric instead of straightforward duals is called self-duality.

4. **EECS 251A Only.** Design a complex CMOS logic gate that implements the function below.
   $f(A, B, C, D) = \overline{A \cdot (B \cdot (C + D) + C \cdot D)}$


VDD
VDD
VDD
D
C
D
A
B
C
f(A,B,C,D)
A
B
C
C
D
D