EECS 151/251A Homework 4

Due Monday, Feb 25$^{\text{th}}$, 2019

Problem 1: Correct JohnW's lecture mistake! [5 pts]

For the "BCD Incrementer Example" presented in lecture 6, derive the minimized logic equations for the outputs, expressed in sum-of-products (SOP) form.

Solution:

From the truth tables for w, x, y and z:

w

cd \ ab	0 0	0 1	1 1	1 0
0 0	0	0	x	1
0 1	0	0	x	0
1 1	0	1	x	x
1 0	0	0	x	x

x

cd \ ab	0 0	0 1	1 1	1 0
0 0	0	1	x	0
0 1	0	1	x	0
1 1	1	0	x	x
1 0	0	1	x	x

y

cd \ ab	0 0	0 1	1 1	1 0
0 0	0	0	x	0
0 1	1	1	x	0
1 1	0	0	x	x
1 0	1	1	x	x

z

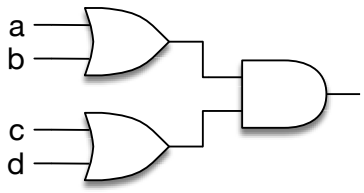
cd \ ab	0 0	0 1	1 1	1 0
0 0	1	1	x	1
0 1	0	0	x	0
1 1	0	0	x	x
1 0	1	1	x	x

$$w = bcd + a\overline{d}$$
$$x = b\overline{c} + b\overline{d} + \overline{b}\,cd$$
$$y = \overline{a}\,\overline{c}\,d + c\overline{d}$$
$$z = \overline{d}$$

Problem 2: Bubble Pushing [14 pts]

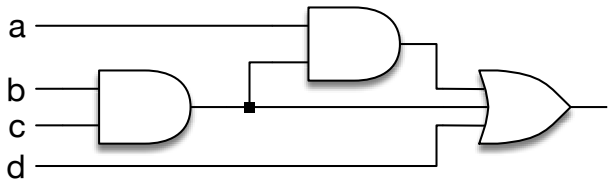
Two-level OR/AND circuits can be implemented as two-level NOR/NOR circuits.

(a) For the example below, using the "bubble pushing" technique, show the series of steps that you would take to convert the circuit to use only NORs. [4pts]



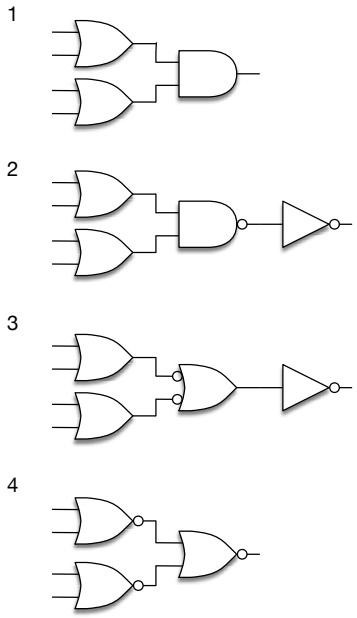
(b) Now convert the OR/AND circuit to use only NANDs (and possibly inverters). [5pts]

(c) Convert the following circuit to all NANDs (and possibly inverters). [5pts]

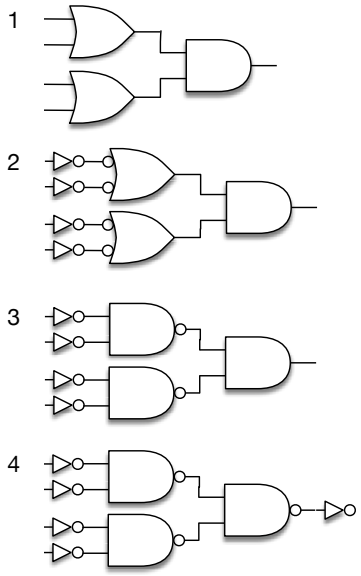


Solution:

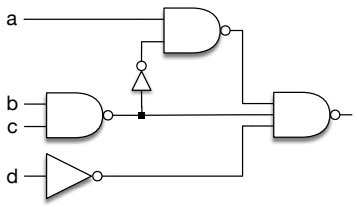
(a) :



(b) :



(c) :



Problem 3: SOP, POS, and Minimum Expressions [9pts]

Consider the function f defined with the truth-table below:

x_0	x_1	x_2	x_3	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	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	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	1	0	1
1	1	1	1	1

(a) Write out the sum-of-products (SOP) and product-of-sums (POS) canonical forms. [3pts]

(b) Derive the minimized expressions for f and for f' in SOP and POS forms (four answers required here). Hint: use a K-map. [6pts]

Solution:

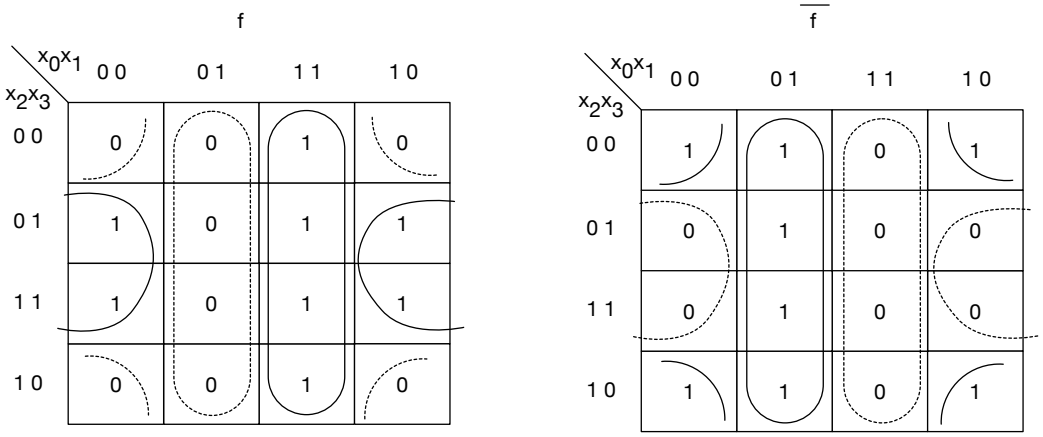
(a) *Sum-of-products (SOP)*:

$$f = \overline{x_0}\,\overline{x_1}\,\overline{x_2}\,x_3 + \overline{x_0}\,\overline{x_1}\,x_2x_3 + x_0\overline{x_1}\,\overline{x_2}\,x_3 + x_0\overline{x_1}\,x_2x_3 + x_0x_1\overline{x_2}\,\overline{x_3} + x_0x_1\overline{x_2}\,x_3 + x_0x_1x_2\overline{x_3} + x_0x_1x_2x_3$$

Product-of-sums (POS):

$$f = (x_0 + x_1 + x_2 + x_3)(x_0 + x_1 + \overline{x_2} + x_3)(x_0 + \overline{x_1} + x_2 + x_3)(x_0 + \overline{x_1} + x_2 + \overline{x_3}\,)(x_0 + \overline{x_1} + \overline{x_2} + x_3)(x_0 + \overline{x_1} + \overline{x_2} + \overline{x_3}\,)(\overline{x_0} + x_1 + x_2 + x_3)(\overline{x_0} + x_1 + \overline{x_2} + x_3)$$

(b) Build a Karnaugh map for f and $\overline{f}$:



From the map for f we can derive:

$$f = x_0 x_1 + \overline{x_1}\, x_3 \qquad \text{(SOP, solid line)}$$
$$f = (x_0 + \overline{x_1}\,)(x_1 + x_3) \qquad \text{(POS, dotted line)}$$

And from the map for $\overline{f}$ we can derive:

$$\overline{f} = \overline{x_1}\,\overline{x_3} + \overline{x_0}\, x_1 \qquad \text{(SOP, solid line)}$$
$$\overline{f} = (\overline{x_0} + \overline{x_1}\,)(x_1 + \overline{x_3}\,) \qquad \text{(POS, dotted line)}$$

Notice that the SOP form for f and the POS form for $\overline{f}$ are equivalent under De Morgan's law - likewise the POS form for f and the SOP form for $\overline{f}$.

Problem 4: Simplification with Boolean Algebra [6pts]

For the following function g defined with the truth-table below, use algebraic manipulation to derive its minimized form.

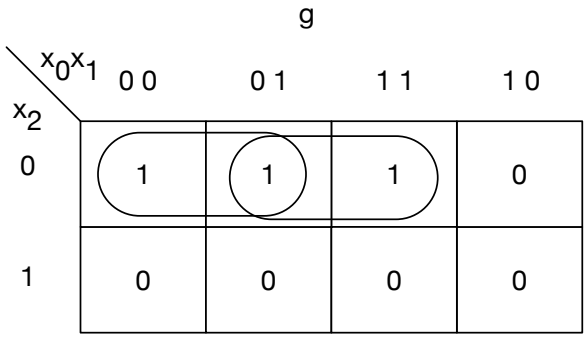
x_0	x_1	x_2	g
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

Solution:

The algebraic derivation starts by reading the SOP form from the truth table and then proceeds as follows:

$$\begin{aligned} g &= \overline{x_0}\,\overline{x_1}\,\overline{x_2} + \overline{x_0}\,x_1\overline{x_2} + x_0x_1\overline{x_2} \\ &= \overline{x_0}\,x_1\overline{x_2} + \overline{x_0}\,\overline{x_1}\,\overline{x_2} + \overline{x_0}\,x_1\overline{x_2} + x_0x_2\overline{x_2} \\ &= \overline{x_0}\,\overline{x_2}\,(x_1 + \overline{x_1}) + x_1\overline{x_2}\,(x_0 + \overline{x_0}) \\ &= \overline{x_0}\,\overline{x_2}\,(1) + x_1\overline{x_2}\,(1) \\ &= \overline{x_0}\,\overline{x_2} + x_1\overline{x_2} \end{aligned}$$

We can confirm this by looking at the Karnaugh-map:



. . . from which the simplified expression is also:

$$g = \overline{x_0}\,\overline{x_2} + x_1\overline{x_2}$$

Problem 5: Building a Better Saturating Incrementer [10pts]

Consider the design of a 3-bit binary unsigned incrementer with saturating output. Saturating in this case means that if the input is the maximum representable value (in this case 111_2), then it is passed through unchanged. In Verilog we might try defining the incrementer with:

```
assign y = (x == 3'b111) ? 3'b111 : x + 1;
```

A naive logic synthesizer might turn this into a comparitor, a multiplexor, and an adder. Your job is to derive a simpler circuit by hand.

Show your derivation of logic equations for the 3-output bits, y[0], y[1], y[2].

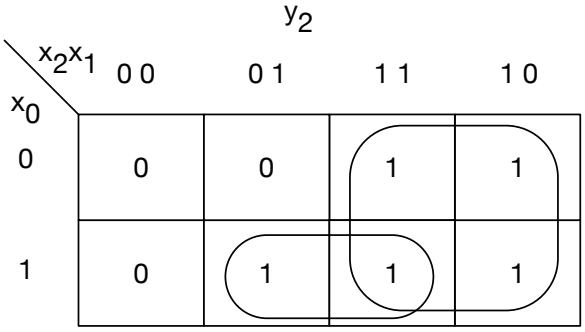
Solution:

First derive an expression for g as a function of the current value x:

x_2	x_1	x_0	y_2	y_1	y_0
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	1	1	1

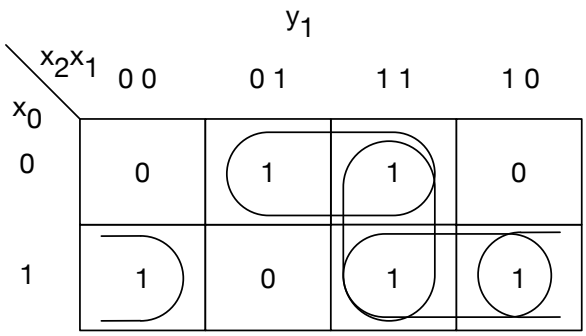
Then derive simplified expressions for each signal in y using a Karnaugh-map (or otherwise):

y_2:



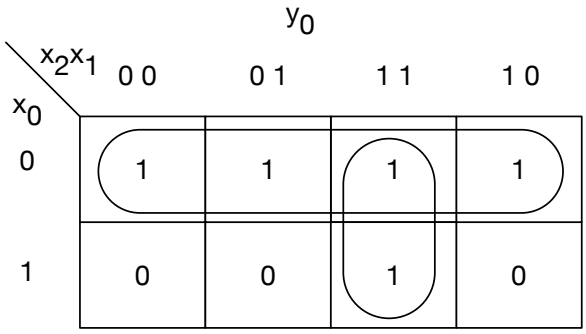
$$y_2 = x_0x_1 + x_2$$

y_1:



$$y_1 = \overline{x_0}\, x_1 + x_2 x_1 + x_0 \overline{x_1}$$

y_0:



$$y_0 = \overline{x_0} + x_2 x_1$$

Problem 6: Writing a FSM in Verilog [8pts]

Write the behavior Verilog description for the Combinational Lock FSM example presented in lecture 7.

Solution:

```
module CombinationalLock(clk, reset, code, enter, open, error);
  input clk, reset;
  input [7:0] code;  // Assume code is 8 bits.
  input enter;
```

```
output open, error;

parameter START = 3'd0;
parameter OK1   = 3'd1;
parameter OK2   = 3'd2;
parameter BAD1  = 3'd3;
parameter BAD2  = 3'd4;

// Let's pretend that the code is '12'.
parameter CODE1 = 8'd1;
parameter CODE2 = 8'd2;

reg open;
reg error;
reg [2:0] present_state, next_state;

always @(posedge clk)
  if (reset)  present_state <= START;
  else        present_state <= next_state;

always @(present_state or enter)
  case (present_state)
    START: begin
      open = 1'b0;
      error = 1'b0;
      if (enter == 1'b0)      next_state = START;
      else if (code == CODE1) next_state = OK1;
      else                    next_state = BAD1;
    end
    OK1: begin
      open = 1'b0;
      error = 1'b0;
      if (enter == 1'b0)      next_state = OK1;
      else if (code == CODE2) next_state = OK2;
      else                    next_state = BAD2;
    end
    OK2: begin
      open = 1'b1;
      error = 1'b0;
      next_state = START;
    end
    BAD1: begin
      open = 1'b0;
      error = 1'b0;
      if (enter == 1'b1)      next_state = BAD2;
      else                    next_state = BAD1;
    end
```

```
        BAD2: begin
          open = 1'b0;
          error = 1'b1;
          next_state = BAD2;
        end
        default: begin
          open = 1'b0;
          error = 1'b0;
          next_state = START;
        end
      endcase
endmodule
```

Problem 7: Designing an FSM [11pts]

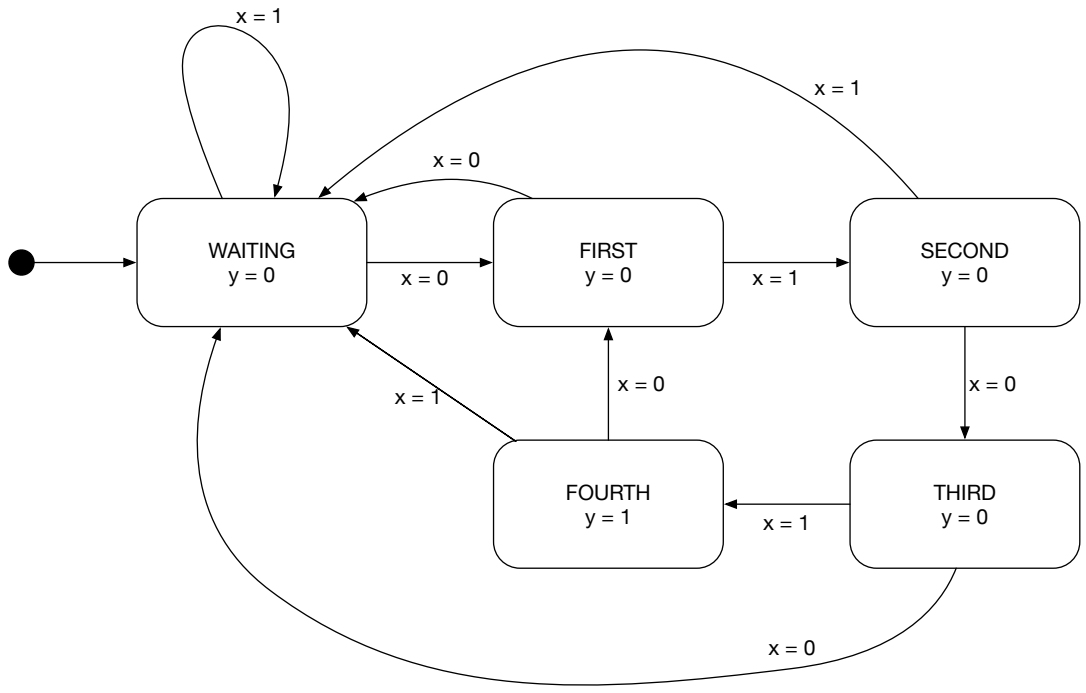
Consider the design of a Moore-style FSM with a 1-bit input (x), and a 1-bit output (y). The FSM accepts inputs one bit at a time and outputs a 0 until it sees the sequence 0101 (with no other bit values in between). After seeing the second 1 in the sequence it outputs a 1 and then starts over.

(a) Draw the state transition diagram. Label all nodes and arcs. [3pts]

(b) Derive the next state logic and the output logic equations. [8pts]

Solution:

(a) :



(b) One way is to use one-hot encoding and 5 bits to express states:

$$\begin{aligned} S_{WAITING} &= 00001 \\ S_{FIRST} &= 00010 \\ S_{SECOND} &= 00100 \\ S_{THIRD} &= 01000 \\ S_{FOURTH} &= 10000 \end{aligned}$$

Then each of the 5 bits of the next state (n) are derived from the current state c and input x as follows, also assuming that there is a reset r:

$$\begin{aligned} y &= c_4 \\ n_0 &= r + c_1\overline{x} + c_2 x + c_3\overline{x} + c_4 x + c_0 \\ &= r + x(c_2 + c_4 + c_0) + \overline{x}\,(c_1 + c_3) \\ n_1 &= \overline{r}\,c_0\overline{x} \\ n_2 &= \overline{r}\,c_1 x \\ n_3 &= \overline{r}\,c_2\overline{x} \\ n_4 &= \overline{r}\,c_3 x \end{aligned}$$

Another way is to express states as integers, S_0 through S_4, using only 3 bits of state.

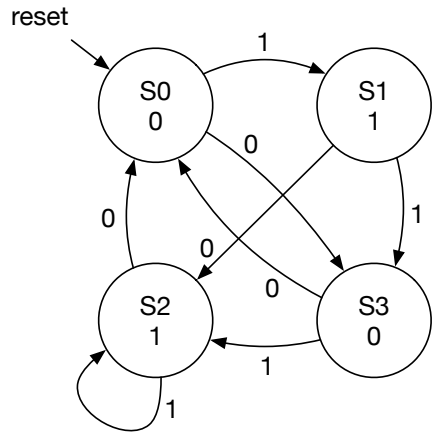
$$\begin{aligned} S_0 &= 000 \\ S_1 &= 001 \\ S_2 &= 010 \\ S_3 &= 011 \\ S_4 &= 100 \end{aligned}$$

You can then use Karnaugh-maps to derive simplified expressions for each bit of the next state n from the current state c and input x in the same way as was done for problem 5.

$$\begin{aligned} y &= n_2\overline{n_1}\,\overline{n_0} \\ n_2 &= c_1 c_0 x \\ n_1 &= \overline{c_1}\,c_0 x + c_1\overline{c_0}\,\overline{x} \\ n_0 &= \overline{c_0}\,\overline{x} + \overline{x_1}\,\overline{x} \end{aligned}$$

Problem 8: Implementing an FSM [13pts]

Consider the state transition diagram shown below:



(a) Write the Verilog behavior description for the corresponding circuit. [5pts]

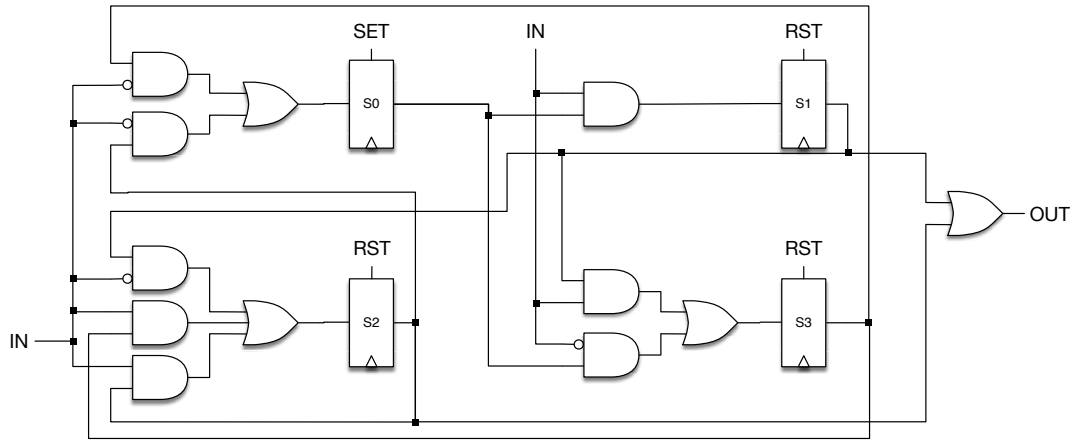
(b) Draw the circuit diagram for one-hot encoded implementation. (You may assume that flip-flops have both "set" and "reset" inputs, but you must label which one you would use for each flip-flop.) [8pts]

Solution:

(a)
```
module Problem(clk, rst, in, out);
  input clk, rst;
  input in;
  output out;

  parameter S0 = 2'b00;
  parameter S1 = 2'b01;
  parameter S2 = 2'b10;
  parameter S3 = 2'b11;

  reg out;
  reg [1:0] present_state, next_state;

  always @(posedge clk)
    if (rst)  present_state <= S0;
    else      present_state <= next_state;

  always @(present_state or in)
    case (present_state)
      S0: begin
        out = 1'b0;
```

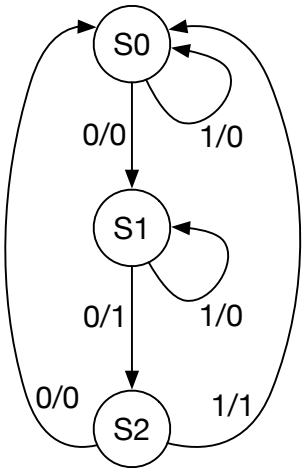
```
            if (in == 1'b1) next_state = S1;
            else            next_state = S3;
          end
          S1: begin
            out = 1'b1;
            if (in == 1'b1) next_state = S3;
            else            next_state = S2;
          end
          S2: begin
            out = 1'b1;
            if (in == 1'b1) next_state = S2;
            else            next_state = S0;
          end
          S3: begin
            out = 1'b0;
            if (in == 1'b1) next_state = S2;
            else            next_state = S0;
          end
          // No 'default' case needed.
        endcase
    endmodule
```

(b) :



Problem 9: Converting from Mealy to Moore Style [6pts]

Convert the Mealy-style state transition diagram to a Moore-style.



Solution: