Timing Rules

p. 32 - 43

→ max propagation delay through combo circuit: t_{cmax}

→ max clock-to-Q delay: t_{pmax}
↳ aka max time b/t clock change & FF output change

→ Clock period: t_{ck}

→ set-up time: t_{su}

→ hold time: t_h

FF to FF Paths

1. Clock period should be long enough to satisfy FF set-up time

$$t_{ck} \geq \underbrace{t_{pmax} + t_{cmax} + t_{su}}_{\text{Set-up time margin}}$$

→ to check set-up time violations, check if:

$$t_{ck} - t_{pmax} - t_{cmax} - t_{su} \geq 0$$

2. Minimum circuit delays should be long enough to satisfy FF hold time. (t_h)

$$t_{pmin} + t_{cmin} \geq t_h$$

★ If a circuit has a hold time violation, it cannot be corrected by changing the clock freq. of the circuit.

Input to FF Path

3. External input changes must satisfy FF set-up time

satisfied if: $t_x \geq t_{cxmax} + t_{su}$

t_{cxmax} = max. prop. delay from X to FF

4. External Input changes must satisfy FF hold times

satisfied if: $t_y \geq t_h - t_{cxmin}$

★ if t_y is negative, X changing before active clock edge will satisfy t_h.

Minimum Clock period

$$t_{ckmin} = t_{pmax} + t_{cmax} + t_{su}$$

Maximum Clock Frequency

$$f_{max} = \frac{1}{t_{pmax} + t_{cmax} + t_{su}}$$

Timing Rules for Circuits w/ Skew

With Positive Skew

$$t_{ck} \geq t_{pmax} + t_{cmax} - t_{skew} + t_{su}$$

$$t_{pmin} + t_{cmin} \geq t_h + t_{skew}$$

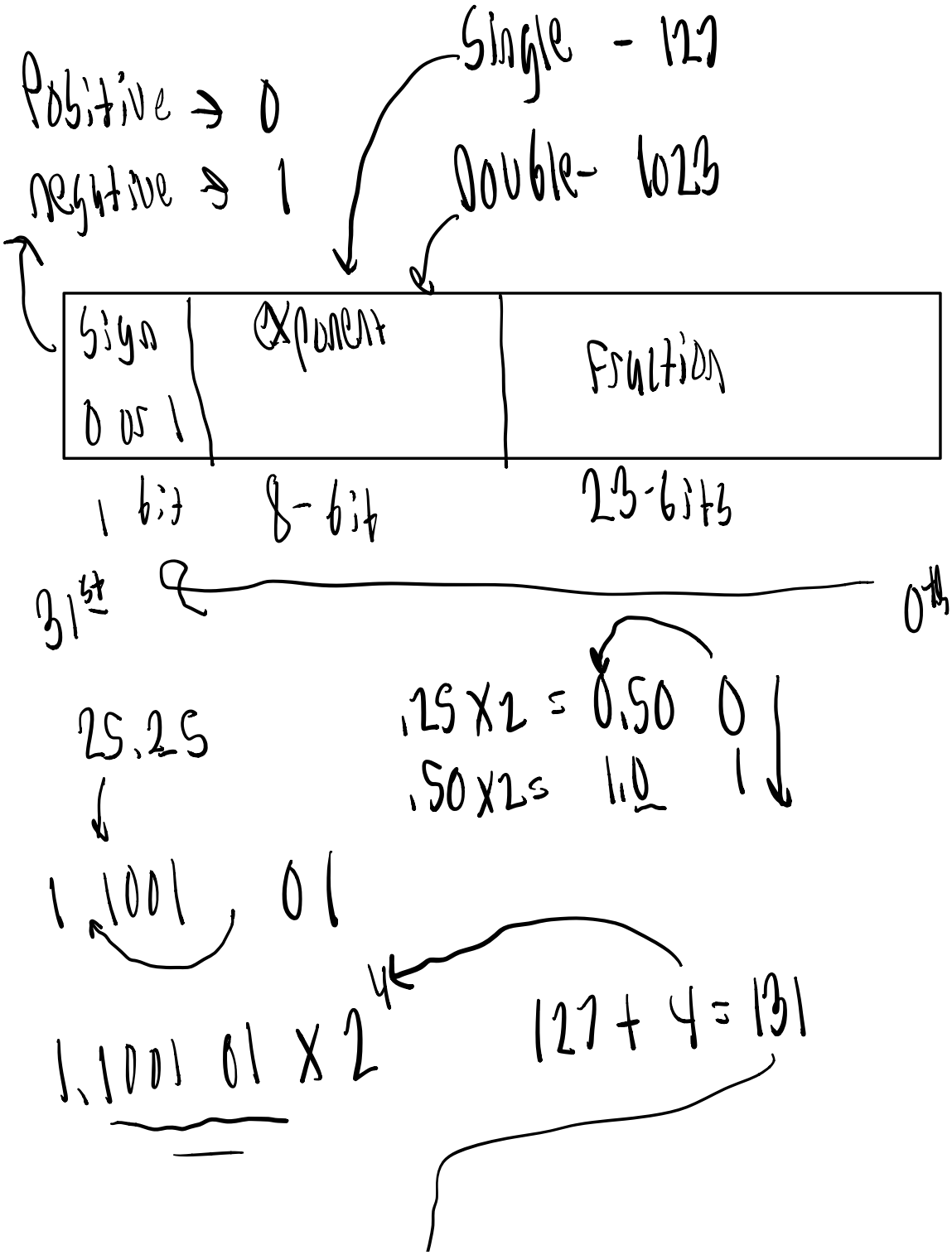
With Negative Skew

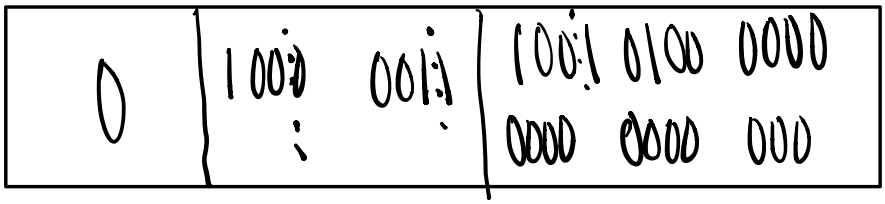
$$t_{ck} \geq t_{pmax} + t_{cmax} + t_{skew} + t_{su}$$

$$t_{pmin} + t_{cmin} \geq t_h - t_{skew}$$

→ Timing Chart for rising edge device → pg. 46

7.2 Convert the following decimal numbers in the IEEE single precision format:
(i) 25.25, **(ii)** 2000.22, **(iii)** 1, **(iv)** 0, **(v)** 1000, **(vi)** 8000, **(vii)** 10^6, **(viii)** -5.4, **(ix)** 1.0×2^{-140}, **(x)** 1.5×10^9





4 1 C A 0 0 0 0

7.3 Convert the following decimal numbers to IEEE double precision format:
(i) 25.25, **(ii)** 2000.22, **(iii)** 1, **(iv)** 0, **(v)** 1000, **(vi)** 8000, **(vii)** 10^6, **(viii)** -5.4, **(ix)** 1.0×2^{-140}, **(x)** 1.5×10^9

2000.22

111 1101 0000 . 0011 1000 0101

0001 1111

Sign = 0

1023 + 10 = 1033 = 0100 0000 1001

0	100 0000 1001	1111 0100 0000 1110 0001 0100 0111 1100 0000...

1-bit 11-bit 52-bit

4 0 9 F 4 0 0 1 4 7 C

-5.4 Single

101.0 1100 1100 1100....

127+2=129 1000 0001

Sign=1

1	1000 0001	0101 1001 1001 1001 1001 100

.4 x 2 = .8 0

.6 = 1

.2 = 1

.4 = 0

.8 = 0

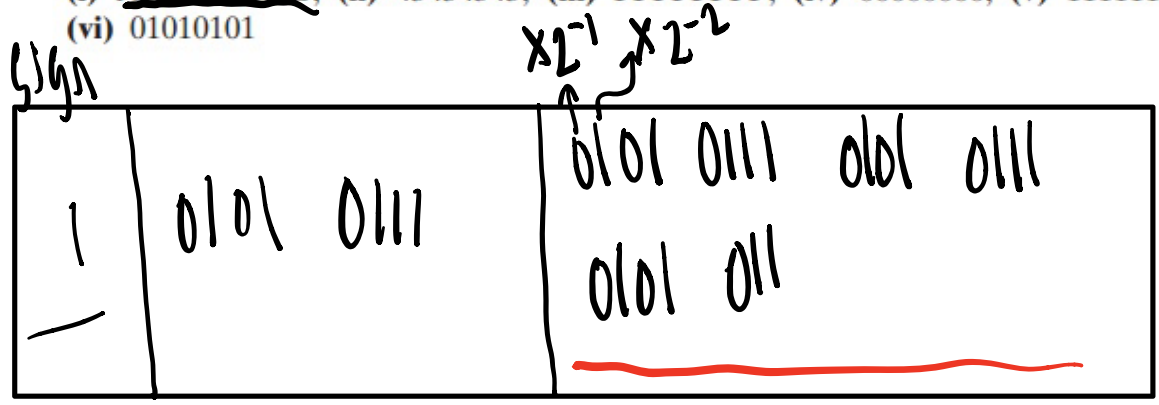
.6 = 1

.2 = 1

.4 = 0

⋮

7.4 What do the following hex representations mean if they are in IEEE single precision format?
(i) ABABABAB, **(ii)** 45454545, **(iii)** FFFFFFFF, **(iv)** 00000000, **(v)** 11111111, **(vi)** 01010101



Neg

87

Remember this is decimal value

$127 - x = 87$

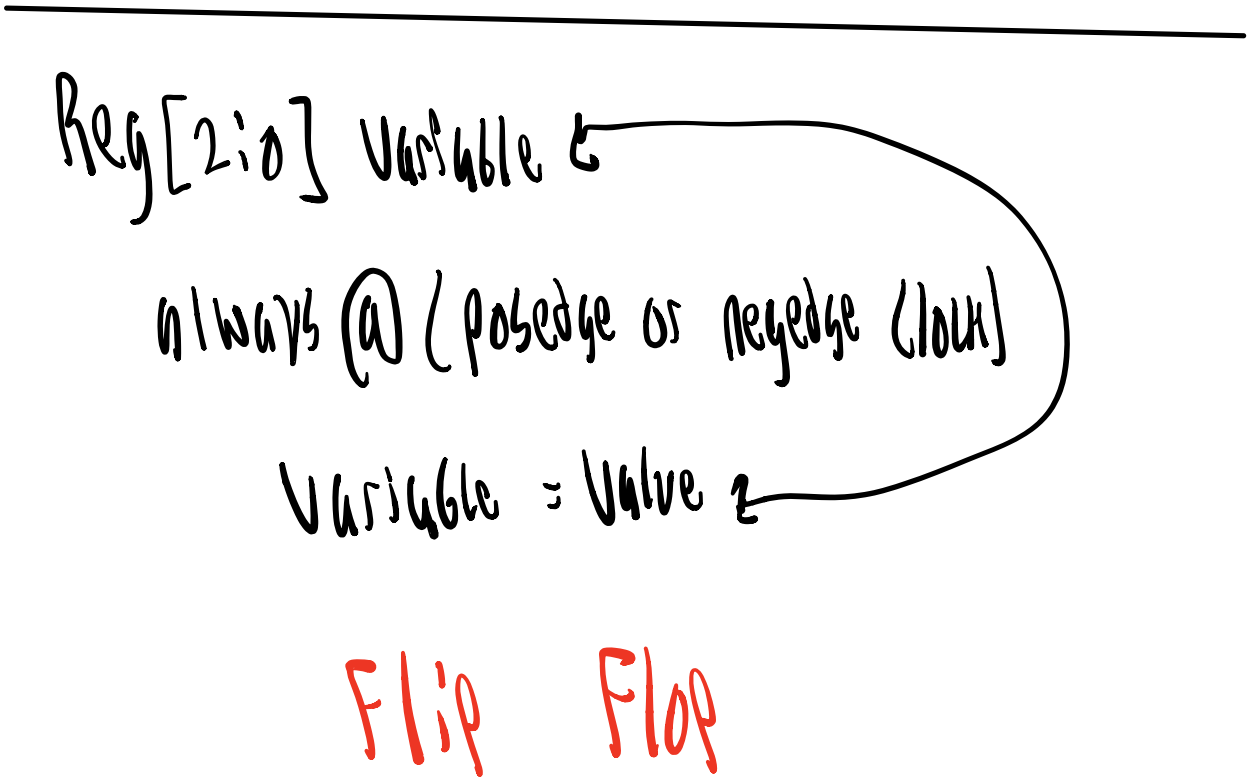
$x = -40$

1.3411764×2^{-40}

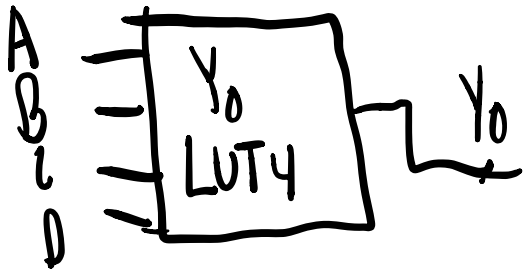
BCD Rep. 0-9 102 $1001_2 = 9_{10}$

BCD: 0001 0000 0010

Binary: 0110 0110


Reg [2:0] Variable
always @ (posedge or negedge Clock)
Variable = Value;

Flip Flop



$$Z(a,b,c,d,e,f) =$$

$$a' \cdot Z(0,b,c,e,d,f) +$$

$$a \cdot Z(1,b,c,d,e,f) = a' z_0 + a z_1$$

6.17 Use Shannon's expansion theorem around *a* and *b* for the function

$$Y = abcde + cde'f + a'b'c'def + bcdef' + ab'cd'ef' + a'bc'de'f + abcd'e'f$$

so that it can be implemented using only 4-variable function generators. Draw a block diagram to indicate how Y can be implemented using only 4-variable function generators. Indicate the function realized by each 4-variable function generator.

$$Y = a'b'Y_0 + a'bY_1 + ab'Y_2 + abY_3$$

case 1 Y_0

a=0 b=0

cde'f + c'def

case 2 Y_1

a=0 b=1

cde'f + cdef'

c'de'f

case 3 Y_2

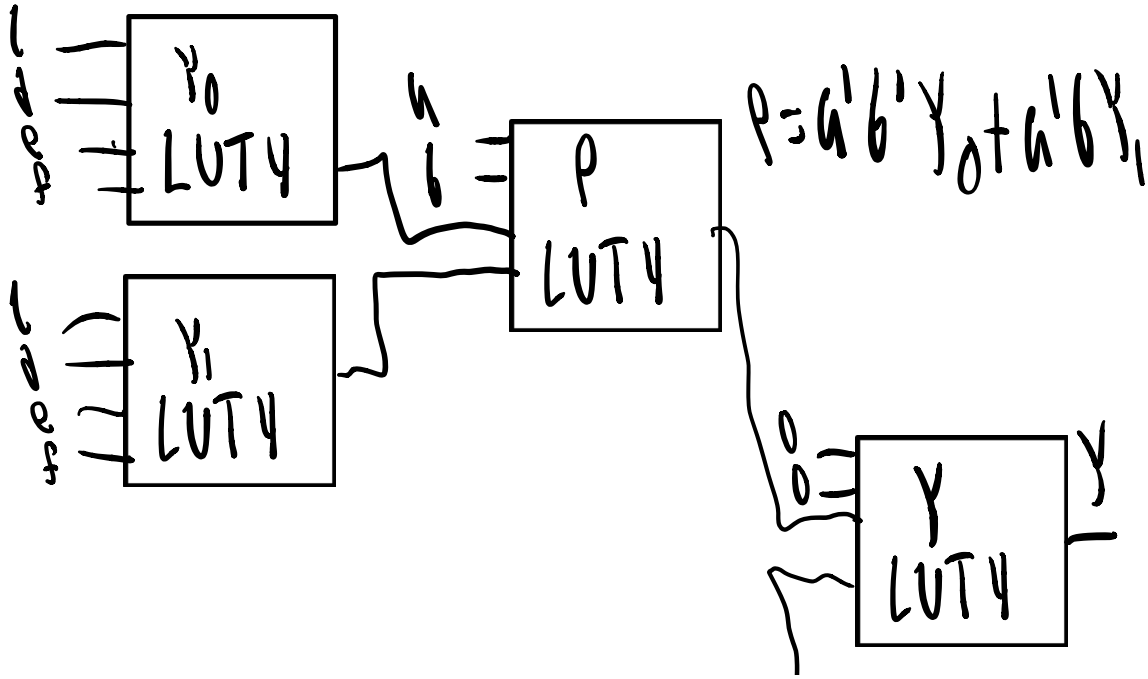
a=1 b=0

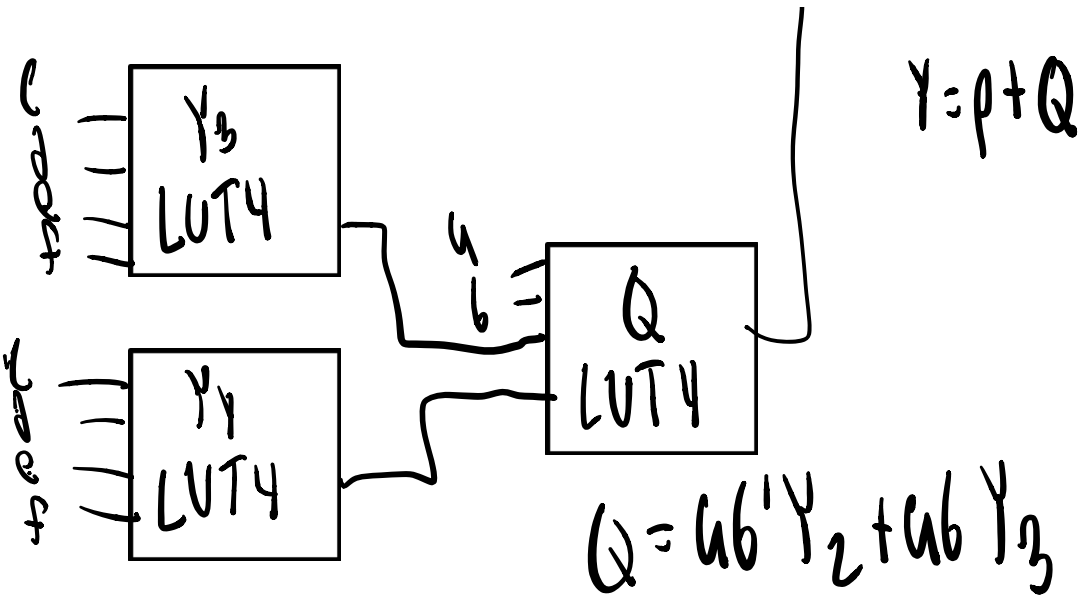
cde'f + c'ef'

case 4 Y_3

a=1 b=1

cde + cde'f + cdef' + c'de'f





6.18 Use Shannon's expansion theorem around *e* and *f* for the function

$$Y = ab'cdef + a'bc'd'e + b'c'ef' + abcde'f$$

so that it can be implemented using a minimum number of 4-variable functions. Rewrite *Y* to indicate how it will be implemented using 4-variable function generators and draw a block diagram. Indicate the function generated by each function generator.

$Y = e'f' Y_0 + e'f Y_1 + ef' + Y_2 + ef Y_3$

Case 1 Y_0 Case 2 Y_1 Case 3 Y_2

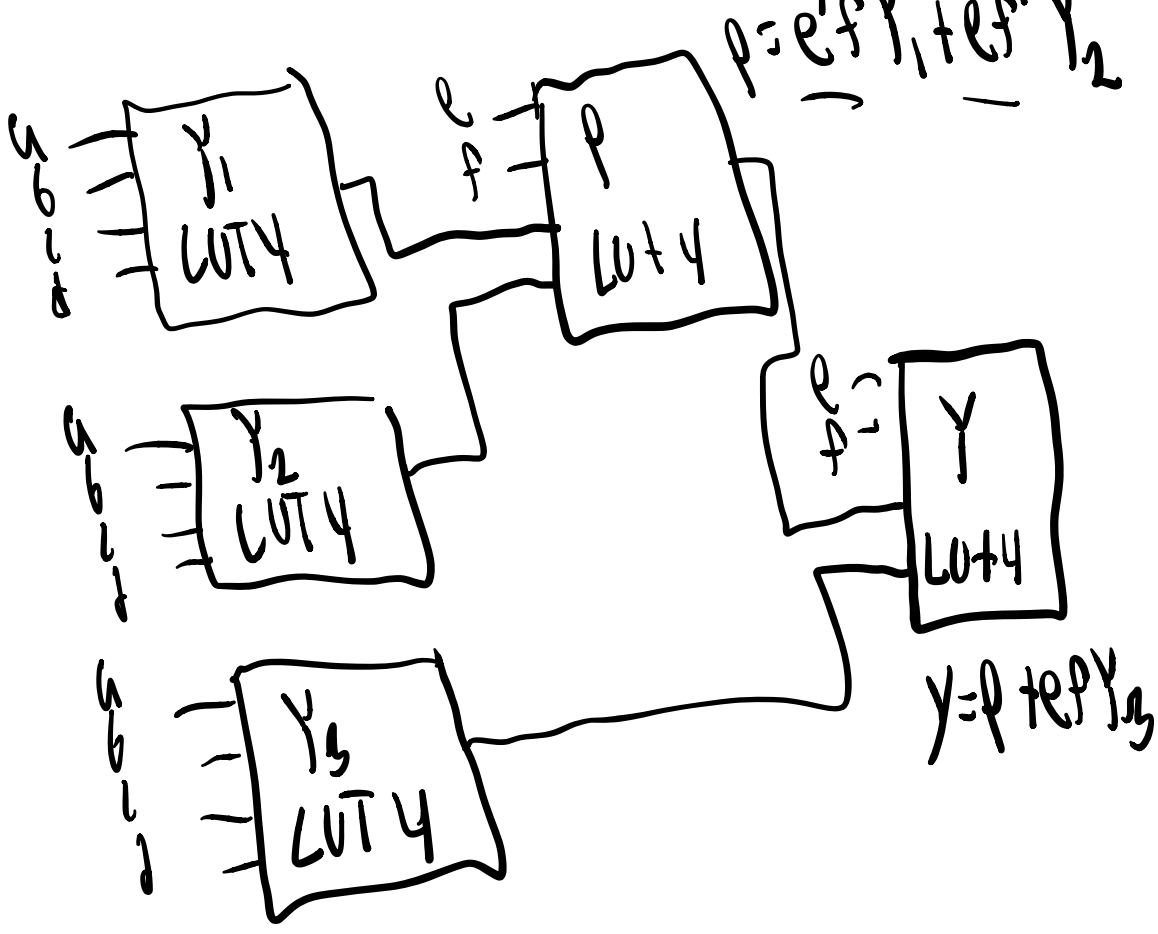
e=0 f=0 e=0 f=1 e=1 f=0

$Y_3 = 0$ abcd $a'bc'd' + b'c'$

Case 4 Y_3

$e=1 \quad f=1$

$Y_3 = ab'cd + a'bc'd'$



Definitions + Quick Tips

* Verilog + VHDL are **not** the same
  - ↳ Verilog is case sensitive
  - ↳ VHDL is case insensitive
  - ↳ both are hardware desc. langs. (HDLs)
* **Combinational Logic:**
  - → output depends solely on the input
  - → has no memory → depends only on present input.
* **Sequential Logic**
  - → has memory
  - → present output depends on sequence of inputs. (past + current ↑)
  - → has states
* Sequential logic is **not** always synchronous.
* Synchronous → has a clock
  - ↳ state changes happen after active clock edge.
* FPGA = Field Programmable Gate Arr.
* Blocking + Non-blocking Asgs.
  - → blocking (=) → eval. before next statement in block. (↷ HAS to be done before next line)
  - → non-blocking (<=) → allows several statements to be eval'd. at same time

* Syntax assign #(delay in ns) F = ____;

* Procedural Asgs: Initial v. Always

* Initial executes only once at time zero.
* Always can be executed over and over again
  - ↳ starts at time 0
  - ↳ has a sensitivity list.

* Mealy vs. Moore

* Mealy → depends on present state and present inputs
* Moore → depends only on present state

P. 29 State Table Reduc.

Steps

Present State	Next State X=0	Next State 1	Present Output X=0	Present Output 1
a	c	~~f~~ e	0	0
~~b~~	~~d~~	~~e~~	~~0~~	~~0~~
c	~~a~~	g	0	0
~~d~~	~~b~~	~~g~~	~~0~~	~~0~~
e	e	b	0	1
f	f	a	0	1
g	c	g	0	1
~~h~~	~~c~~	~~f~~	~~0~~	~~0~~

(a) State table reduction by row matching

1. Cross out any obv. matches.
2. ✓ make chart + cross out states w/ dif. outputs
3. ✓ work through + write all implied pairs
4. ✓ work through until you run out of X's
5. remaining → reduced table

	a	b	c	d	e	f
b	c-d, e-f					
c	~~f-g~~	a-d, e-g				
d	~~b-e~~, ~~f-g~~	~~e-g~~	a-b			
e	X	X	X	X		
f	X	X	X	X	a-b	
g	X	X	X	X	~~a-e~~, ~~b-g~~	~~a-f~~, ~~a-g~~

→ a + b are eq.

→ c has no eq. states

→ c + d are eq.

→ e has no eq. states

→ e + f are eq.

→ g has no eq. states

Final Table

PS	X = 0	1	X = 0	1
a	c	~~f~~ e	0	0
c	a	g	0	0
~~e~~	~~e~~	~~b~~ a	0	1
g	c	g	0	1

1.2 Fill in the truth table for the subtractor and find SOP + POS for Diff and Bout

→ shows − ans.

Diff = X − y − Bin When X < (y + Bin), Bout is set

Truth Table for Sub.

X	y	Bin	Diff	Bout
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	0
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Diff

Bin \ xy	00	01	11	10
0		1		1
1	1		1	

take 1's

SOP: $X'y'B_{in} + X'yB_{in}' + XyB_{in} + Xy'B_{in}'$

POS: $(X'+y'+B_{in})(X'+y+B_{in}')(X+y+B_{in})(X+y'+B_{in}')$

take 0's

Bout

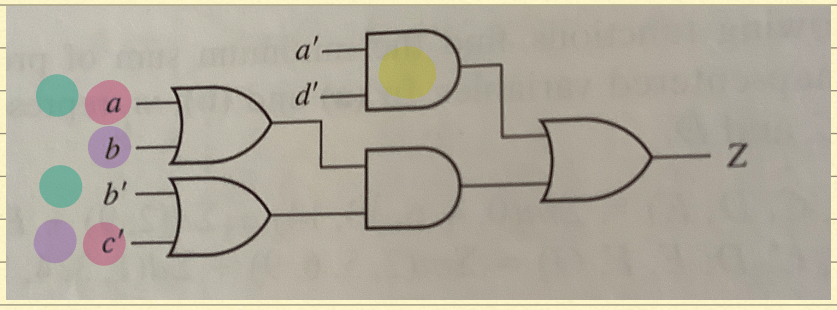
Bin \ xy	00	01	11	10
0		1		
1	1	1	1	

SOP: $X'B_{in} + X'y + yB_{in}$

POS: $(X'+B_{in})(X'+y)(y+B_{in})$

1.8)

a) Find all the static hazards + state the conditions in which they occur.



Info on Static hazards: p. 13

Info to make a circuit w/o hazards p. 14

CD \ AB	00	01	11	10
00	1	1	1	1
01	0	1	1	1
11	0	0	0	1
10	1	1	0	1

★ 1 Prime implicants to add.

★ Start at gates and work to inputs for k-map

Static 1-hazard: happens w/ adjacent 1's in dif. terms

$ABC'D$ happens from 0000 to 1000
and from 0010 to 1010

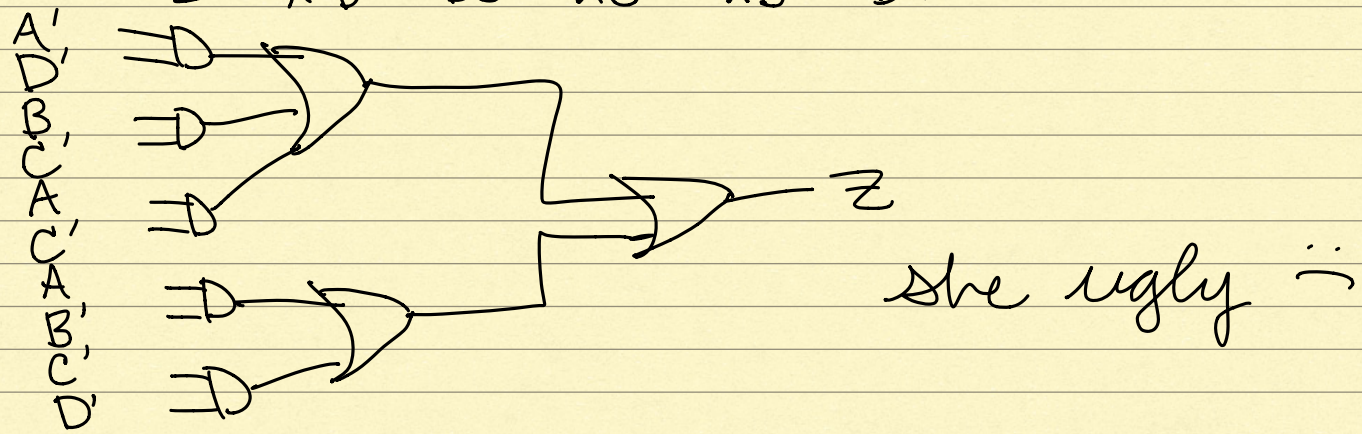
Static 0-hazard: happens w/ adjacent 0's in dif. terms

$ABCD$ happens from 0011 to 0111

b.) Steps

1. Find SOP eq. that includes all prime implicants
2. Use eq. to make a 2-level AND-OR circuit. ↝ added PI

$$Z = A'D' + BC' + AC' + AB' + B'D'$$



she ugly ∴

1.11 0101 → followed by 11 and then reset
0110

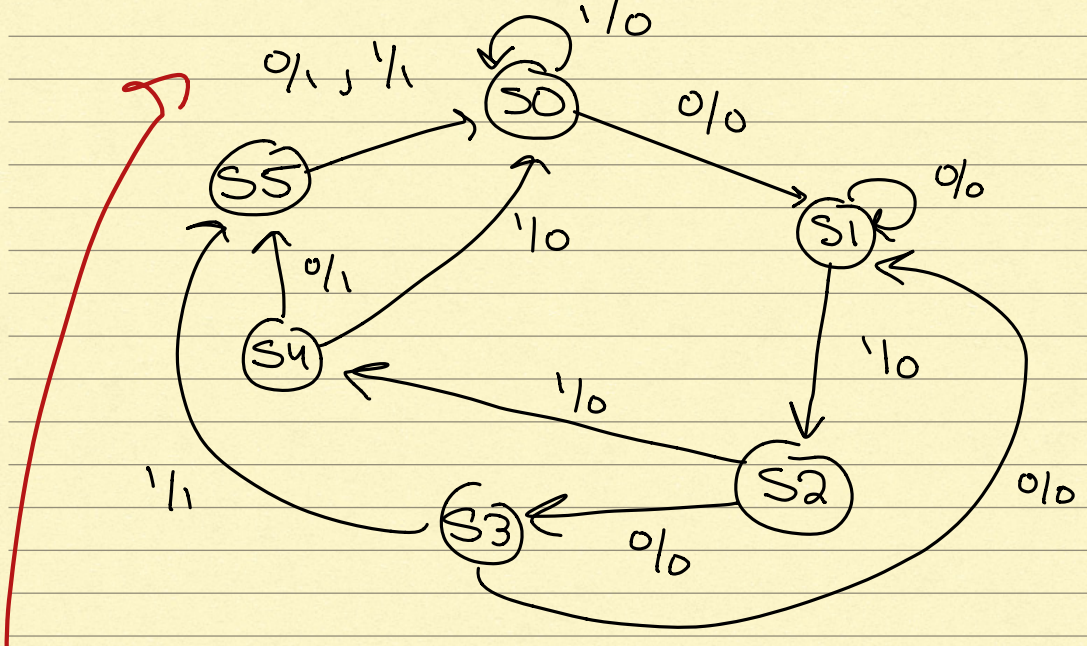
input X = 0100111010101101101

output Z = 0000000000111000011

Mealy

PS		NS (input) x=0	NS (input) 1	Output x=0	Output 1
Reset	S0	S1	S0	0	0
0	S1	S1	S2	0	0
01	S2	S3	S4	0	0
010	S3	S1	S5	0	1
011	S4	S5	S0	1	0
001 or 0110	S5	S0	S0	1	1

input/output



State Assignment

Guidelines on pg. 22

S0 = 000
S1 = 001
S2 = 010
S3 = 011
S4 = 100
S5 = 101

plug in for transition table

JK FF Excitation Table

Q Output		Inputs	
Present State Q	Next State Q^+	J_n	K_n
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

Transition Table → made using state asg. + state table

$Q_1Q_2Q_3$	$Q_1^+Q_2^+Q_3^+$ X=0	X=1	Z X=0	X=1
000	001	000	0	0
001	001	010	0	0
010	011	100	0	0
011	001	101	0	1
100	101	000	1	0
101	000	000	1	1
110	XXX	XXX	X	X
111	XXX	XXX	X	X

pg. 23 for D FFs

ε → don't care b/c we don't use these states

J_1 (rows Q_2Q_3, cols XQ_1):

Q_2Q_3 \ XQ_1	00	01	11	10
00		X	X	
01		X	X	
11		X	X	1
10		X	X	1

$J_1 = XQ_2$

K_1 (rows Q_2Q_3, cols XQ_1):

Q_2Q_3 \ XQ_1	00	01	11	10
00	X		1	X
01	X	1	1	X
11	X	X	X	X
10	X	X	X	X

$K_1 = X + Q_3$

J_2 (rows Q_2Q_3, cols XQ_1):

Q_2Q_3 \ XQ_1	00	01	11	10
00				
01				1
11	X	X	X	X
10	X	X	X	X

$J_2 = XQ_1'Q_3$

K_2 (rows Q_2Q_3, cols XQ_1):

Q_2Q_3 \ XQ_1	00	01	11	10
00	X	X	X	X
01	X	X	X	X
11	1	X	X	1
10		X	X	1

$K_2 = X + Q_3$

J_3 (rows Q_2Q_3, cols XQ_1):

Q_2Q_3 \ XQ_1	00	01	11	10
00	1	1		
01	X	X	X	X
11	X	X	X	X
10	1	X	X	

$J_3 = X'$

K_3 (rows Q_2Q_3, cols XQ_1):

Q_2Q_3 \ XQ_1	00	01	11	10
00	X	X	X	X
01		1	1	1
11		X	X	
10	X	X	X	

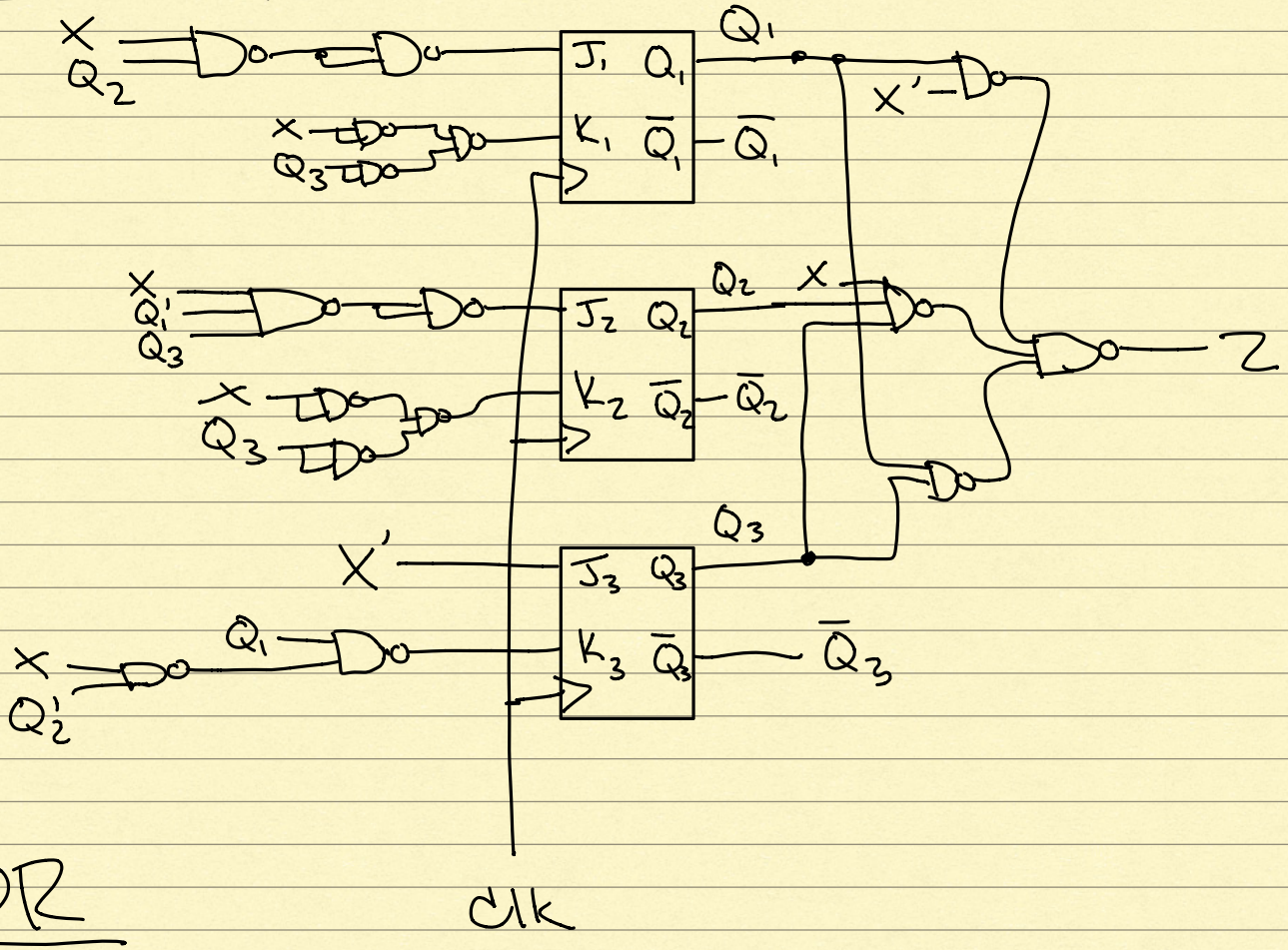
$K_3 = Q_1 + XQ_2'$

Z (rows Q_2Q_3, cols XQ_1):

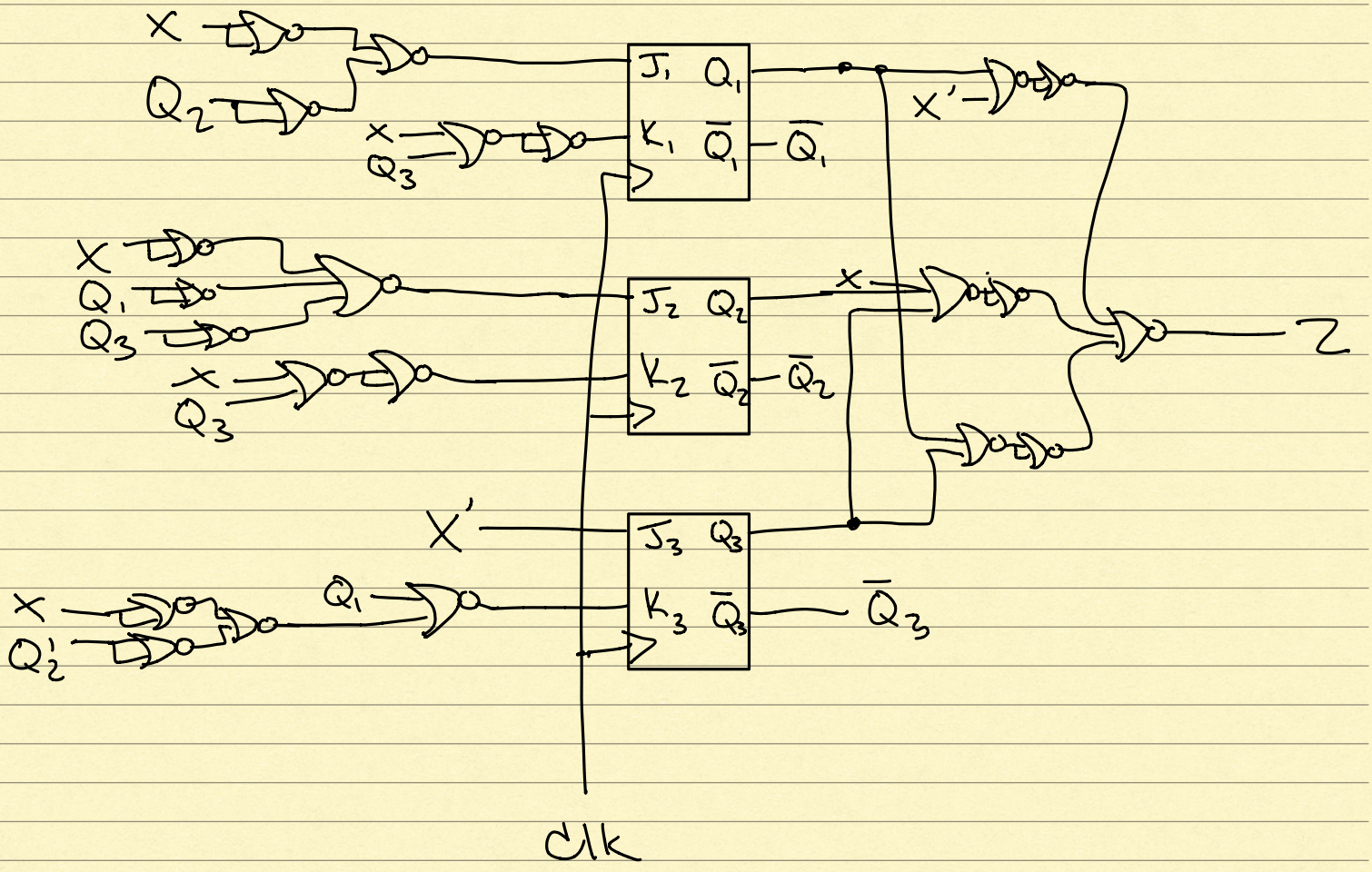
Q_2Q_3 \ XQ_1	00	01	11	10
00		1		
01		1	1	
11		X	X	1
10		X	X	

$Z = X'Q_1 + Q_1Q_3 + XQ_2Q_3$

NAND


X
Q2
J1 Q1
Q1
X'
K1 Q̄1
Q̄1
Q3
X
Q1'
Q3
J2 Q2
Q2
X
K2 Q̄2
Q̄2
Q3
X'
J3 Q3
Q3
Q1
K3 Q̄3
Q̄3
X
Q2'
Z
clk

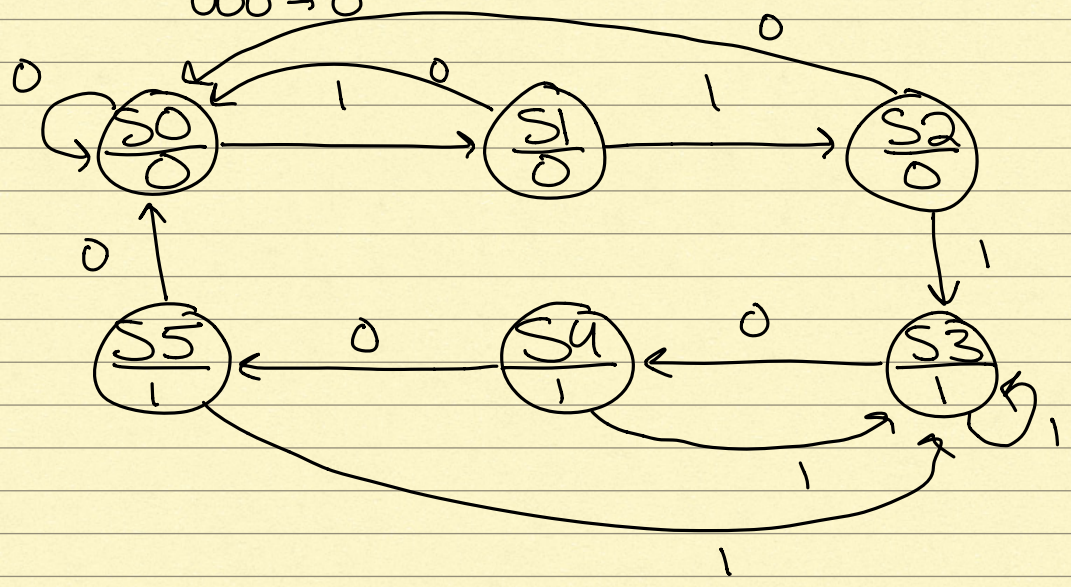
NOR

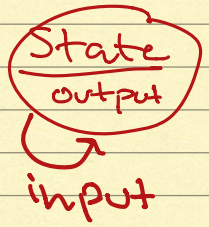

X
Q2
J1 Q1
X'
K1 Q̄1
Q̄1
Q3
X
Q1
Q3
J2 Q2
X
K2 Q̄2
Q̄2
Q3
X'
J3 Q3
Q1
K3 Q̄3
Q̄3
X
Q2'
Z
clk

1.15 | 111 → 1
000 → 0

Output holds all other times

Moore

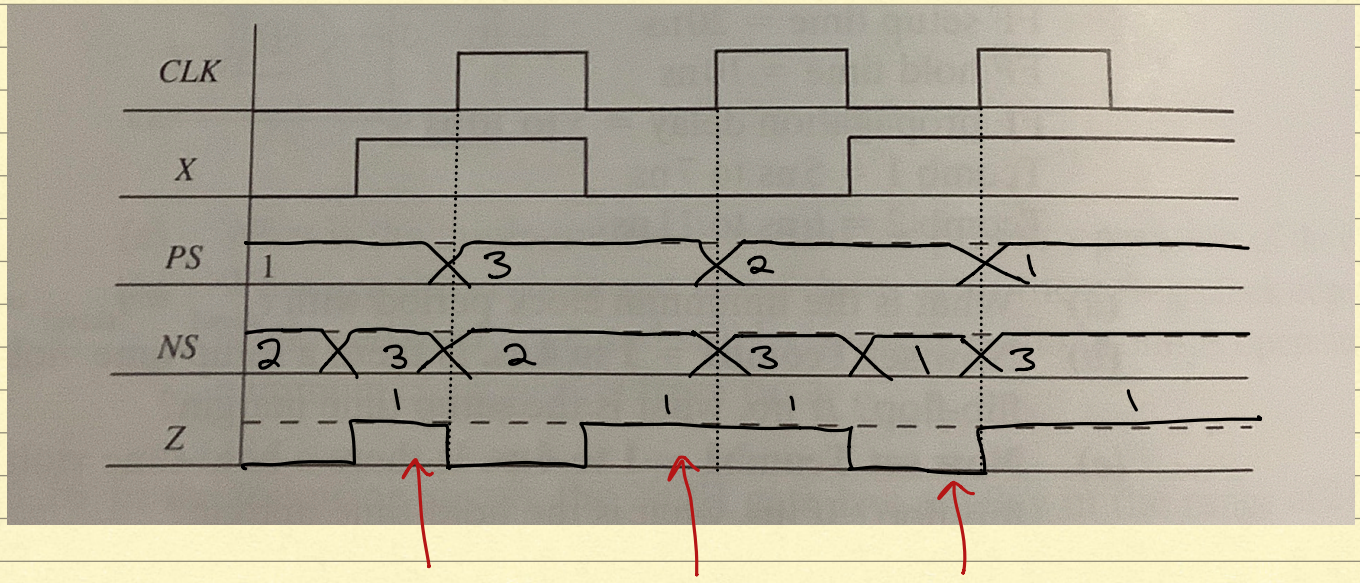




PS		NS (input) X=0	1	Output
000	S0	S0	S1	0
001	S1	S0	S2	0
011	S2	S0	S3	0
111	S3	S4	S3	1
110	S4	S5	S3	1
100	S5	S0	S3	1

1.29 | Complete the Timing Diagram

★ All states change after rising clk edge

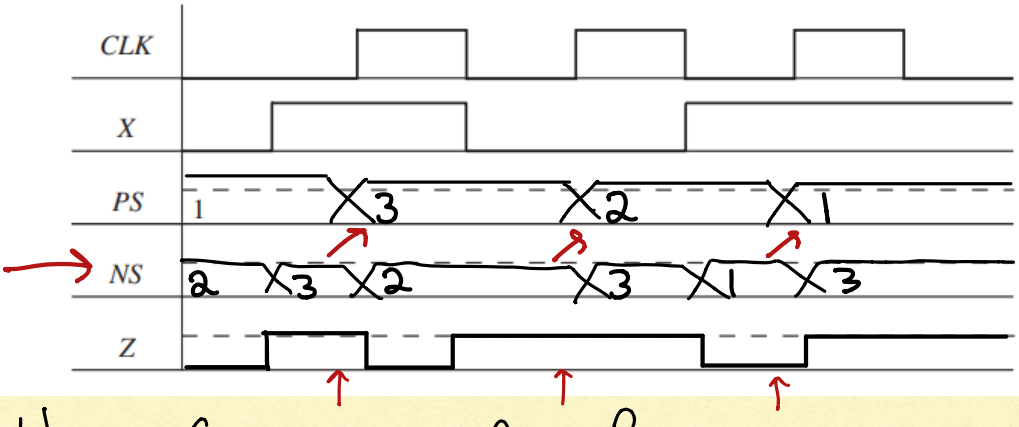


1.29 A Mealy sequential machine has the following state table:

PS	NS		Z	
	X = 0	X = 1	X = 0	X = 1
1	2	3	0	1
2	3	1	1	0
3	2	2	1	0

★ going at posedge + any change always (*)

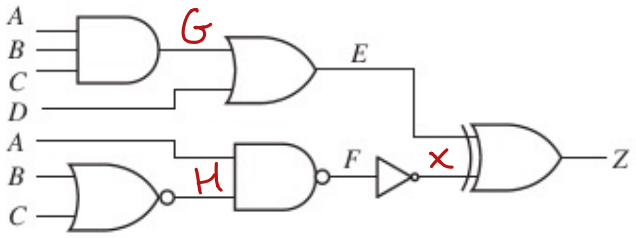
Complete the following timing diagram. Clearly mark on the diagram the times at which you should read the values of Z. All state changes occur after the rising edge of the clock. Assume the machine is initialized to state 1



Z output
110

2.1 | Get dets from answer key or text book

2.4) Write a Verilog description of the following combinational circuit using concurrent statements. Each gate has a 5-ns delay, excluding the inverter, which has a 2-ns delay.

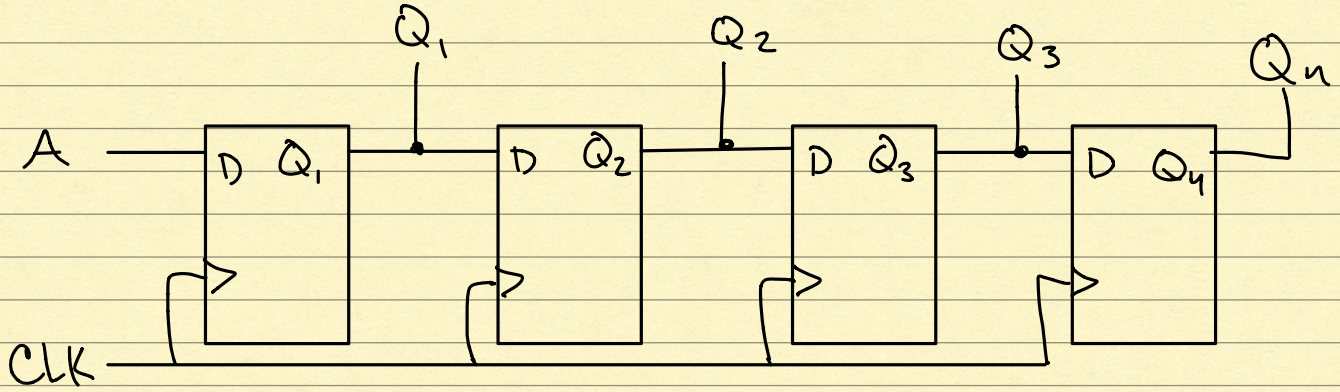



```
module Circuit(A,B,C,D,Z)
  input  A,B,C,D
  output Z
  wire   G,H,E,F,X

    assign #5  G = A & B & C
    assign #5  H = ~(B|C)
    assign #5  E = G | D
    assign #5  F = ~(A & H)
    assign #2  X = ~F
    assign #5  Z = E ^ X
                ↑ Delay
endmodule
```

2.21) Draw the hardware obtained if the following code is synthesized:

```
module  reg3 (Q1,Q2,Q3,Q4, A,CLK);
input   A;
input   CLK;
output  Q1,Q2,Q3,Q4;

reg     Q1,Q2,Q3,Q4;

always @(posedge CLK)
begin
  Q4 = Q3;
  Q3 = Q2;
  Q2 = Q1;
  Q1 = A;
end

endmodule
```

4 bit shift register



2.36

Draw the circuit represented by the following Verilog process:

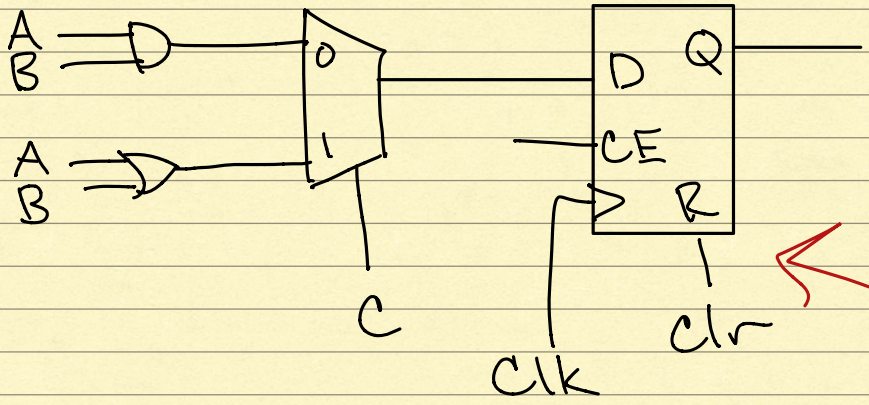
```
always @(clk,clr)
begin
    if(clr == 1'b1)
        Q <= 1'b0;
    else if(clk == 1'b0 && CE == 1'b1)
        begin
        if(C == 1'b0)
            Q <= A & B;
        else
            Q <= A | B;
        end
end
```

reset

clock enable (?)

Select line

Why is *clr* on the sensitivity list whereas *C* is not?



★ example D FF w/ reset

2.34 Consider the following Verilog code:

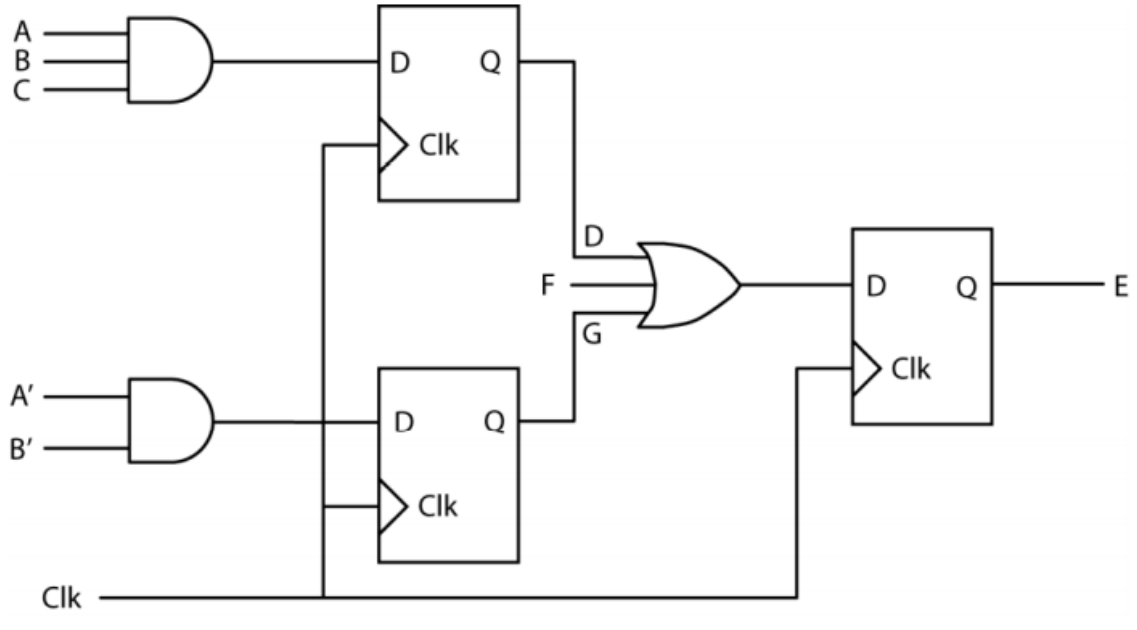
```
module Q3(A,B,C,F,Clk,E);
input A,B,C,F,Clk;
output reg E;

reg D,G;

initial
begin
    E = 1'b0;
    D = 1'b0;
    G = 1'b0;
end

always @(posedge Clk)
begin
    D <= A & B & C;
    G <= ~A & ~B;
    E <= D | G | F;
end
endmodule
```

init. to zero

clock = FF

Combo statement = gate

(a) Draw a block diagram for the circuit (no gates and at block level only).
(b) Give the circuit generated by the preceding code (at the gate level).

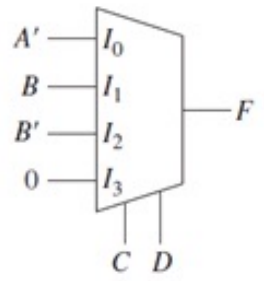


★ Shown as example

2.37 **(a)** Write a conditional signal assignment statement to represent the 4-to-1 MUX shown subsequently. Assume that there is an inherent delay in the MUX that causes the change in output to occur 10 ns after a change in input.

(b) Repeat **(a)** using an if-else statement.

(c) Repeat **(a)** using a case statement.



P. 102-103

Syntax

assign F = (A)? I_1 : I_0;

F = (A)? (B? I_3 : I_2) : (B? I_1 : I_0)

a.)

```
assign #10 F = (C==0)?((D==0)?~A:B):((D==0)?~B:0)
```

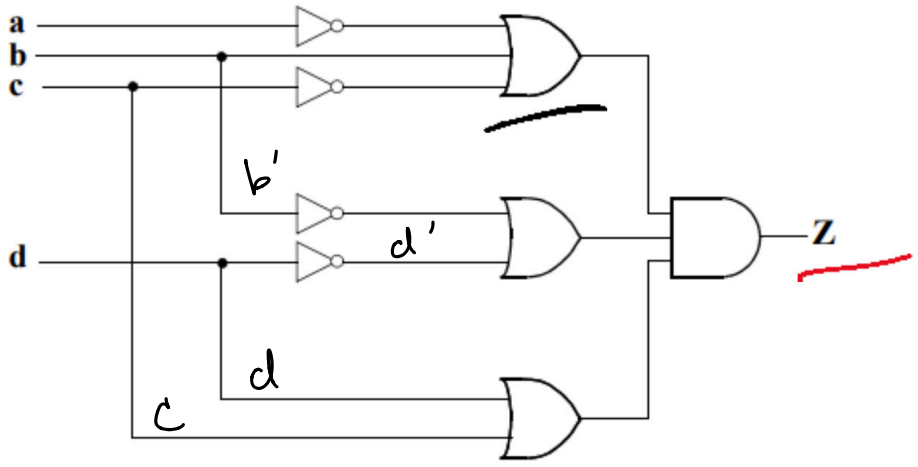
b.

```
always @(I0, I1, I2, I3, C, D)
  begin
    if (C==0 && D==0)
        #10 F = ~A;
    else if (C==0 && D==1)
        #10 F = B;
    else if (C==1 && D==0)
        #10 F = ~B;
    else
        #10 F = 0;
  end
```

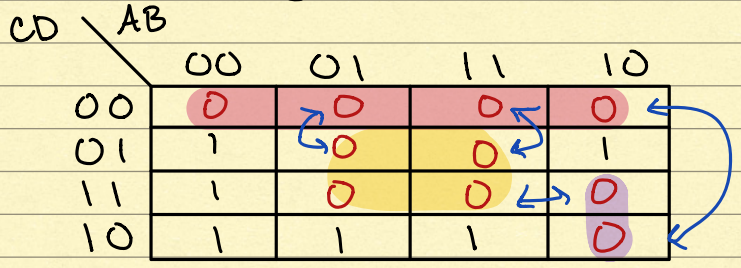
c.)

```
always @(*)
  begin
    case ( Sel )   → {C, D}

      0: #10 F = ~A;
      1: #10 F = B;
      2: #10 F = ~B;
      3: #10 F = 0;
    endcase
  end
```

[12.5 points] For the network shown below, find all static 0 hazards. For each 0-hazard found, specify the conditions which will cause the hazard to appear at the output (i.e. specify the logic values of the variables which are constant and clearly specify the variable that is assumed to be changing). If there are no 0-hazards found, use a K'map to show why this is the case.



$(A' + B + C')(B' + D')(D + C)$

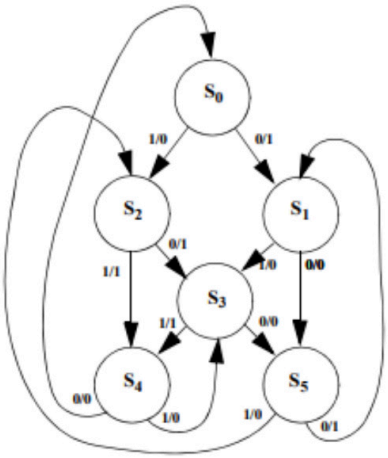


Format ABCD 0-hazards at:

0100	to	0101
1100	to	1101
1111	to	1011
1000	to	1010

[10 points] Reduce the following state graph to a minimum number of states clearing identifying the states that are equivalent with one another. Show the final reduced state graph or table.

input
output

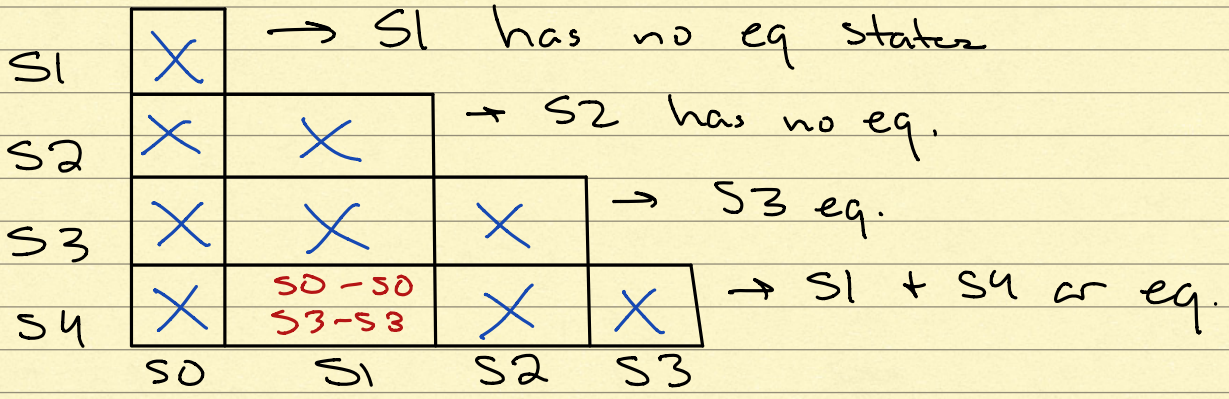


S5 ≡ S4

S4 ≡ S1

PS	NS (input) x=0	NS (input) 1	Output x=0	Output 1
S0	S1	S2	1	0
S1	~~S5~~ S0	S3	0	0
S2	S3	~~S4~~ S1	1	1
S3	~~S5~~ S0	~~S4~~ S1	0	1
~~S4~~	S0	S3	0	0
~~S5~~	S1	S2	1	0

eq. to S0



→ S1 has no eq states

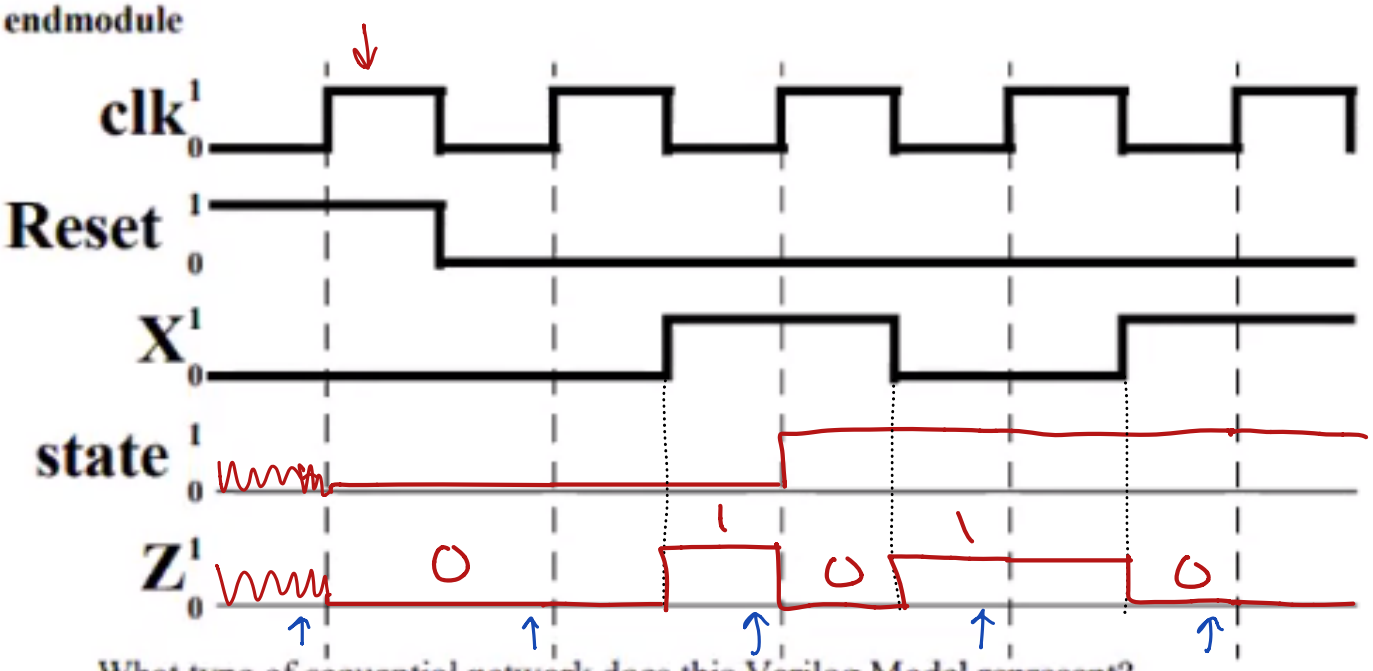
→ S2 has no eq.

→ S3 eq.

→ S1 + S4 or eq.

4. [12.5 points] Complete the following timing diagram for the output signal, **Z** and the internal input signal, **state**. Assume that a basic functional RTL Simulation is to be performed.

```
module fsm_network (input clk, X, Reset, output reg Z);
    reg state, next_state;

    always @ (state, X)
        case (state)
            0 : if (X) begin Z=1; next_state=1; end
                else  begin Z=0; next_state=0; end
            1 : if (X) begin Z=0; next_state=1; end
                else  begin Z=1; next_state=1; end
        endcase

    always @ (posedge clk)
      if (Reset) state=0;
      else state=next_state;

endmodule
```



What type of sequential network does this Verilog Model represent?

Mealy → depends on state + inputs

Write down the output sequence for Z.

X 0 1 1 0

Are there any 'false' outputs on the timing diagram? If so clearly identify them.

↳ don't know what state and Z start as

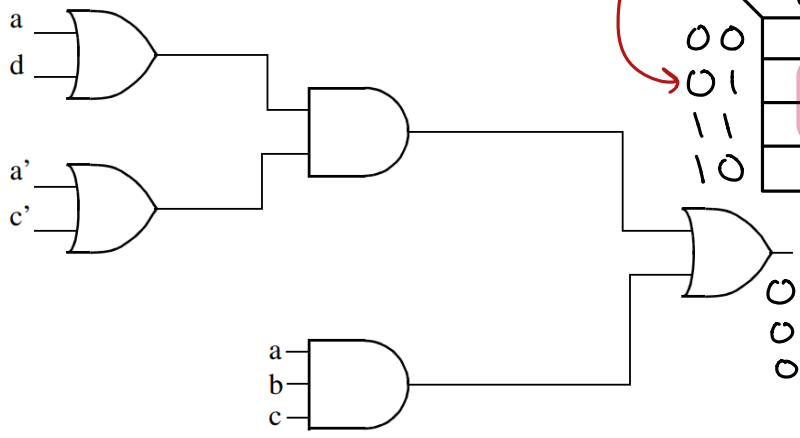
Exam 1 Retry Probs.

Natalie Taggart

Problem 1 Retry: #1 → 6/10 need 2 right

1. 1.

For the network shown below, find any/all static 1-hazards. For any 1-hazard found specify the conditions which will cause the hazard to appear at the output (i.e. specify the logic values of the variables which are constant and the variable which is changing)



CD \ AB	00	01	11	10
00	0	0	1	1
01	1	1	1	1
11	1	1	1	0
10	0	0	1	0

ABCD

0101 to 1101 → A 0 to 1

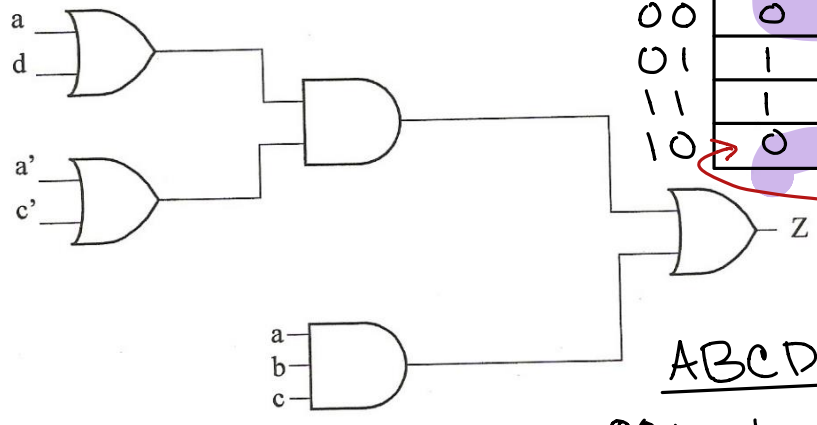
0111 to 1111 → A 0 to 1

0001 to 1001 → A 0 to 1

$abc + (a+d)(c'+c') = ab + ac' + a'd$

2.

For the network shown below, find any/all static 0-hazards. For any 0-hazard found specify the conditions which will cause the hazard to appear at the output (i.e. specify the logic values of the variables which are constant and the variable which is changing) [10 points].



CD \ AB	00	01	11	10
00	0	0	1	1
01	1	1	1	1
11	1	1	1	0
10	0	0	1	0

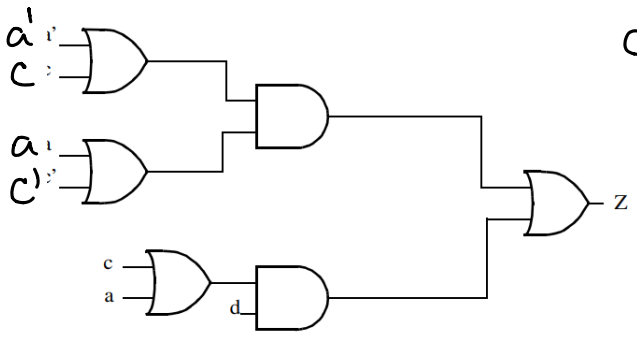
ABCD

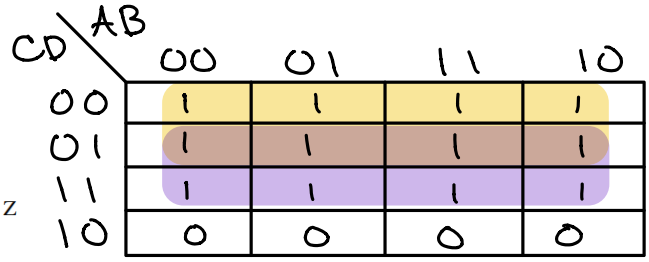
0010 to 1010 → A 0 to 1

$ab'c + a'd'$

1. 3.

For the network shown below, using the k-map methods outline in the class, find all static 0-hazards (or state that there are no static 0-hazards present). For any 0-hazard found specify the conditions which will cause the hazard to appear at the output (i.e. specify the logic values of the variables which are constant and the variable which is changing).



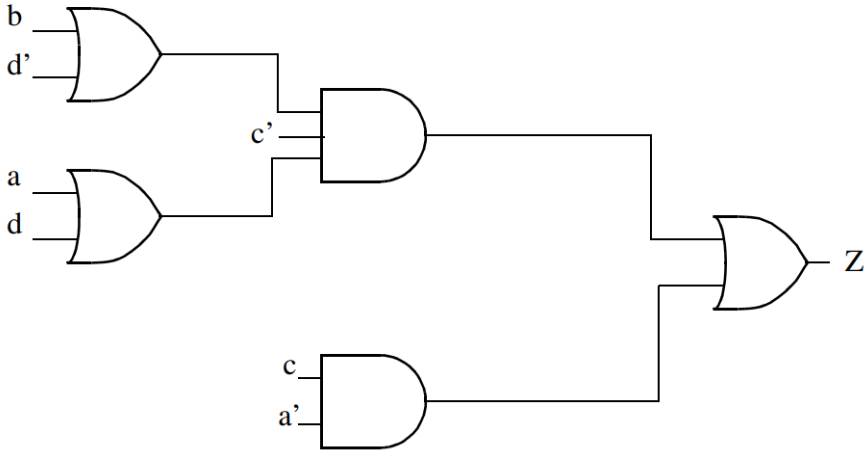


$(c' + d)' \rightarrow cd'$

no zero hazards

1. 4.

For the network shown below, find all static 0-hazards. For each 0-hazard found specify the conditions which will cause the hazard to appear at the output (i.e. specify the logic values of the variables which are constant and clearly specify the variable that is assumed to be changing). If there are no 0-hazards found, use a K'map to show why this is the case.

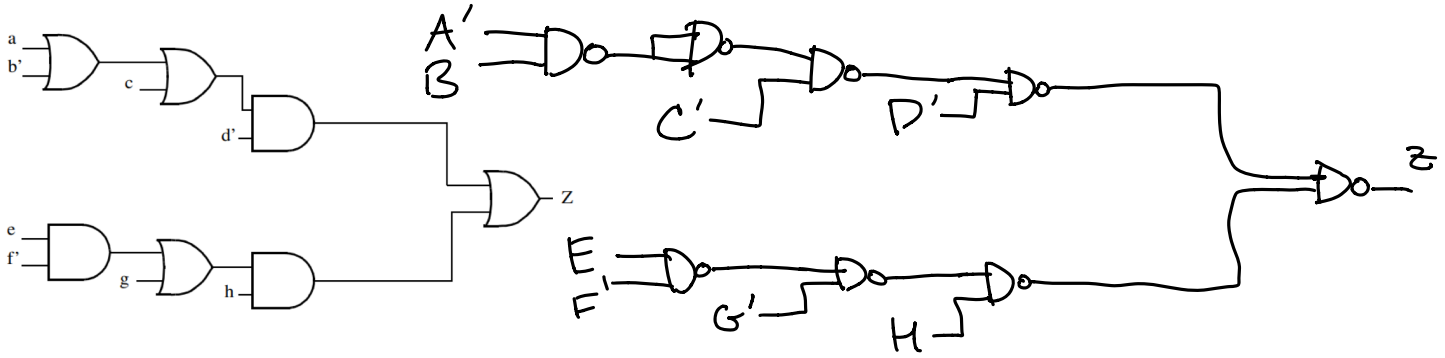


#3 → 4/10 → need 3 right

Problem 3 Retry:

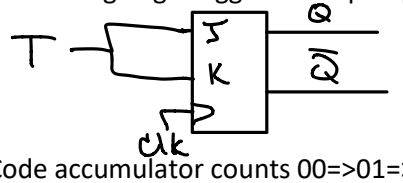
3. 1.

Draw an equivalent multi-level network that is composed entirely of multiple two-input **NAND** gates. Assume that all input variables (a through h) are available in their true and complemented form.



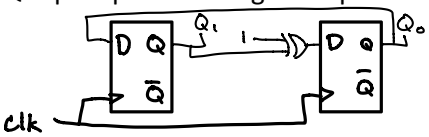
3. 2.

Use a single rising edge triggered JK Flip-Flop with no additional components (other than connecting wires) to create a rising edge triggered T flip flop (see Fig. 1-13 of the textbook for an example).



3. 3.

A 2-bit Gray Code accumulator counts 00=>01=>11=>10=>00 and continues repeatedly. Using 2 rising-edge triggered DQ Flip Flops and a single 2-input XOR gate, draw a circuit that continually generates this pattern.



Format

Q_1Q_0

3. 4.

A 3-bit down-counter counts 111=>110=>101=>100=>011=>010=>001=>000=>111 and continues repeatedly. By first creating a truth table or K-map to determine next state (based on current state as the input), create a 3-bit down counter circuit with the use of only NAND gates and 3 rising-edge triggered Flip Flops.

3. 5.

A ring oscillator is a circuit with an odd number of inverters attached to each other in a daisy chain loop (output of the final inverter is tied to the input of the first inverter). Draw a ring oscillator comprising 9 inverters in one daisy chain feedback loop. If each inverter has an inertial delay of exactly 200 picoseconds, and neglecting propagation delay on the wires between inverters (i.e. 0 additional delay), at what frequency will the oscillator operate (in MHz)? If a 10th inverter is added to the loop, how does the circuit behave?

#4 → 6/8 → need 1 right

Problem 4 Retry:

4. 1. Write a short Verilog description of a transparent D Latch with D and G inputs and Q output.

module DLATCH (**input** D, G, **output** Q) Done on separate doc.

4. 2. Write a short Verilog description of a rising-edge triggered T Flip-Flop (see Fig. 1-13 on pp. 16 of the textbook for definition), with both Q and QN outputs (QN is the inverse of Q). In addition to its normal T and CLK inputs, add an active-low asynchronous SETN input that sets Q high when SETN is low (regardless of clock edge).

module TFF (**input** T, CLK, SETN, **output** Q, QN)

3. Write a short Verilog description of a falling-edge triggered JK Flip-Flop, with J and K inputs, and an active-high asynchronous clear input CLR that sets the output Q immediately to 0.

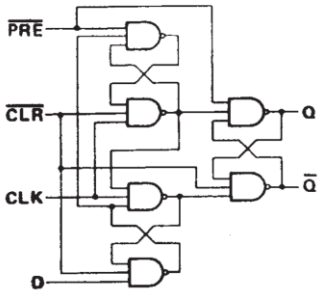
module JKFF (**input** J, K, CLK, CLR, **output** Q) Done on separate doc.

4. 4. The following truth table and circuit diagram is from the Texas Instruments SN7474 datasheet, showing one basic design for a DQ flip-flop with active-low preset (PREN) and active-low clear (CLRN). Note that all gates drawn are NAND gates!

FUNCTION TABLE

INPUTS				OUTPUTS	
$\overline{PRE}$	$\overline{CLR}$	CLK	D	Q	$\overline{Q}$
L	H	X	X	H	L
H	L	X	X	L	H
L	L	X	X	H†	H†
H	H	↑	H	H	L
H	H	↑	L	L	H
H	H	L	X	Q_0	$\overline{Q}_0$

logic diagram (positive logic)



a) Are PREN and CLRN asynchronous or synchronous inputs?

b) Write a short Verilog description of the SN7474 circuit as shown in the diagram above ***USING ONLY assign OPERATORS (NO always @(…) procedures)***

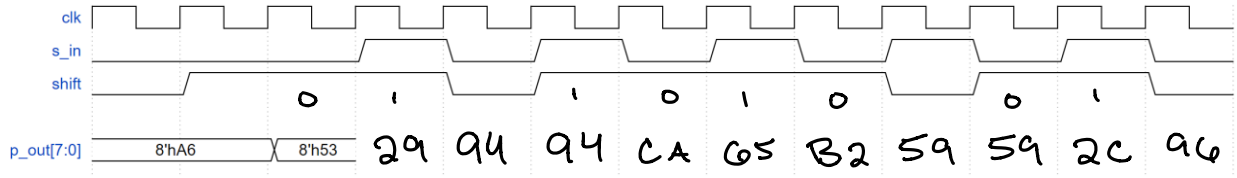
module SN7474 (**input** PREN, CLRN, CLK, D, **output** Q, QN)

#6 → 6/10 → need 2 right

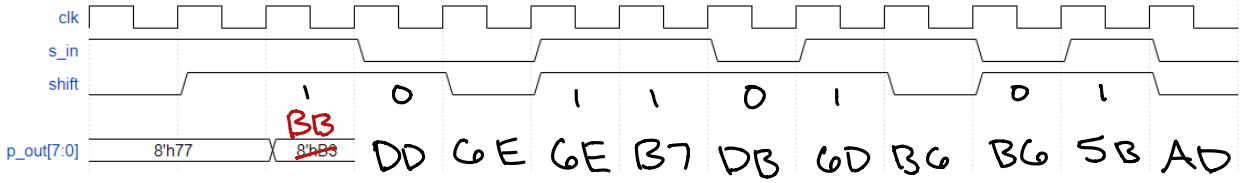
Problem 6 Retry:

INSTRUCTIONS: In this problem set, all state machines are intended to behave as Moore state machines; the p_out[7:0] and accum[7:0] outputs must change in value one clock cycle after the inputs change, as shown in the first 3 provided values in the timing diagrams.

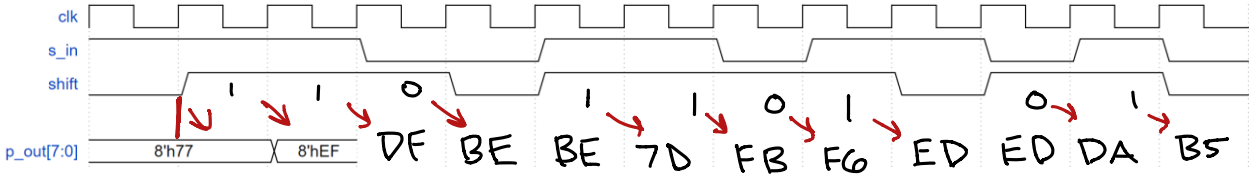
6. 1. Complete the following timing diagram of a serial-in/parallel-out shift register, where serial data shifts in bit-by-bit from the **s_in** input into the most-significant bit side (bit 7; newest/most-recent) towards the least-significant bit side (bit 0; oldest/least-recent). Data only shifts when the **shift** input is high; when shift is low **p_out**[7:0] does not change in value. You can either complete the **p_out**[7:0] signal values in hexadecimal or binary.



6. 2. Complete the following timing diagram of a serial-in/parallel-out shift register, where serial data shifts in bit-by-bit from the **s_in** input into the most-significant bit side (bit 7; newest/most-recent) towards the least-significant bit side (bit 0; oldest/least-recent). Data only shifts when the **shift** input is high; when shift is low **p_out**[7:0] does not change in value. You can either complete the **p_out**[7:0] signal values in hexadecimal or binary.



6. 3. **NOTE: THIS PROBLEM STATEMENT DIFFERS FROM #1 & #2…** Complete the following timing diagram of a serial-in/parallel-out shift register, where serial data shifts in bit-by-bit from the **s_in** input into the ***least-significant*** bit side (bit 0; newest/most-recent) towards the ***most-significant*** bit side (bit 0; oldest/least-recent). Data only shifts when the **shift** input is high; when shift is low **p_out**[7:0] does not change in value. You can either complete the **p_out**[7:0] signal values in hexadecimal or binary.



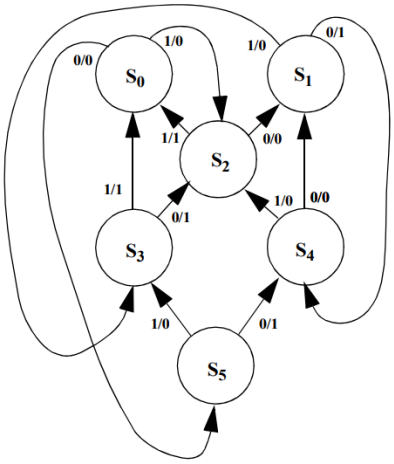
#7 → 3/10 → med 4 right

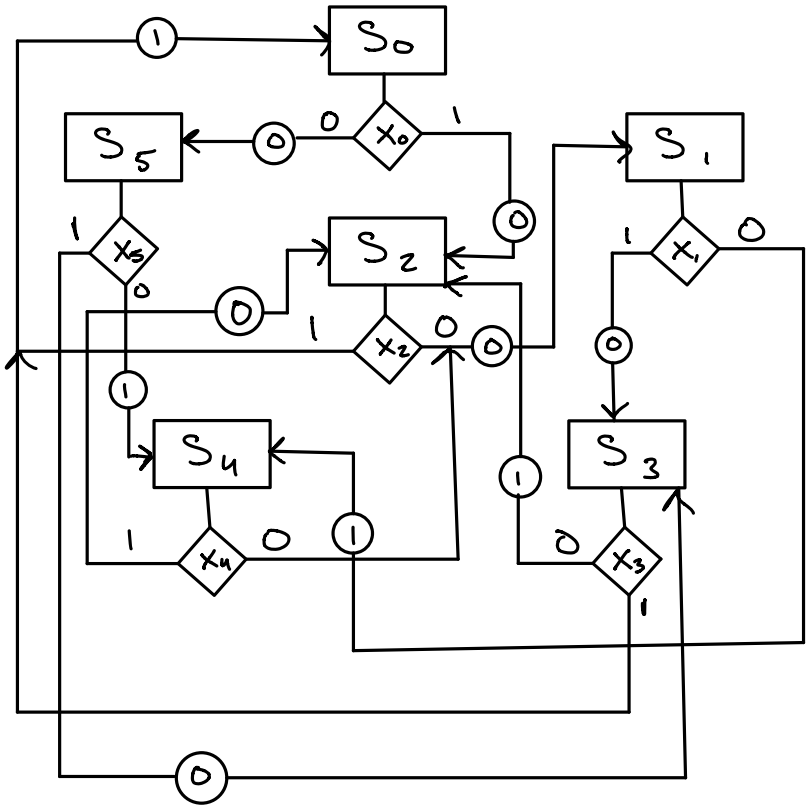
Problem 7 Retry:

INSTRUCTIOS: FOR THE FOLLOWING 4 STATE MACHINES, DO THE FOLLOWING:

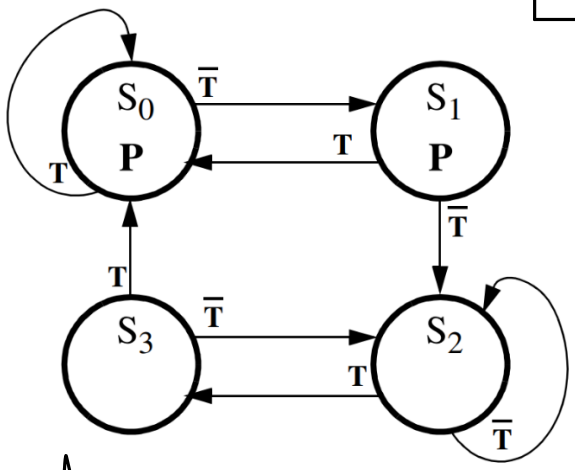
1) **Create an equivalent Algorithmic State Chart representation, using rectangular boxes to show states, diamonds to show input/decision blocks, and ovals to represent conditional outputs. See the Module/Slides for Algorithmic State Machine Representation for examples.**
2) **Write a behavioral Verilog HDL model that implements the state transition graph. In your model include a synchronous active high reset input signal that will always place the design in state S0 on the active edge of the next clock pulse regardless of the current state of the network.**

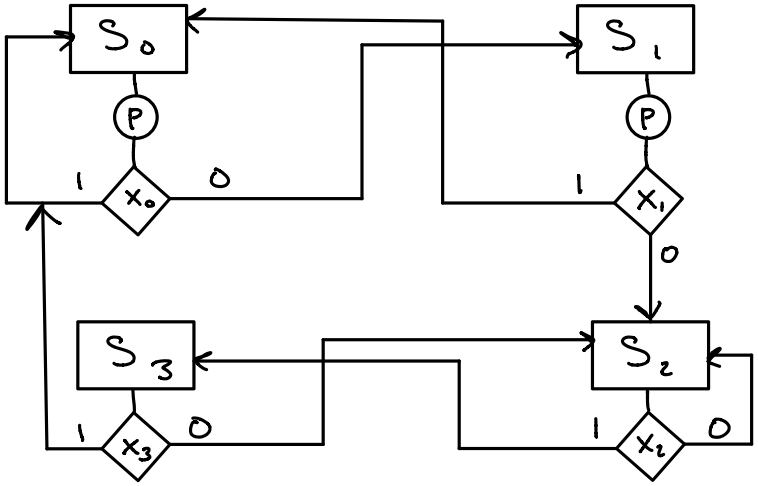
A:





B:



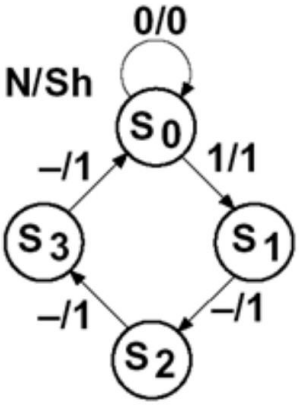


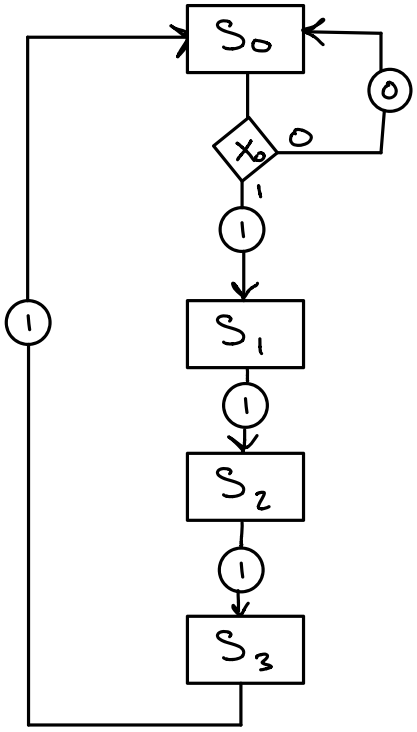
Let

T = 1

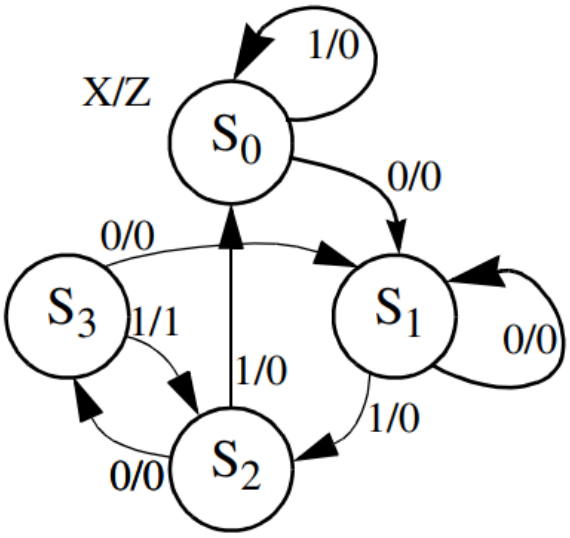
T̄ = 0

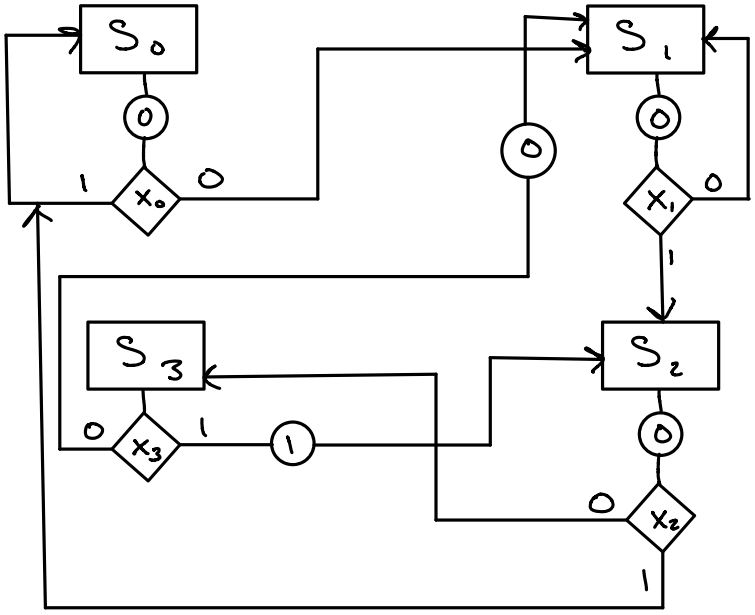
C:





D:





All Verilog done on separate doc.

Problem 10 Retry:

1. Develop a Verilog model for a 4-bit barrel shifter, which produces a 4-bit output that performs "Rotate Shift Left" or "Circular Shift Left". The 4-bit input vector shall be named I[3:0], the number of bit positions to shift is SH[1:0], and the output is O[3:0].

Note that a barrel shifter (or rotate/circular shift operation) shifts any MSbits back around to the LSbit position. For example, if I[3:0]=4'b0111 and SH[1:0]=3, then O[3:0]=4'b1011.

module BSHIFT (**input** [3:0] I, **input** [1:0] SH, **output** [3:0] O)

2. Develop a Verilog model for a 6-to-1 multiplexer where the general inputs represent a 6-bit bus and the output is a single signal. When S[2:0] is 6, choose I[2] as the output, and when S[2:0] is 7, choose I[3] as the output. If using a case statement, please use a default clause; if using an if/else statement, assign O in all clauses including a final "else" clause to avoid creating a latch. Label the inputs I[5:0], S[2:0], and the output O. Verilog on separate doc.

module MUX6TO1 (**input** [5:0] I, [2:0] S, **output** O)

3. Develop a Verilog model for a 2-bit synchronous static random access memory with the following specifications:

Inputs: CLK, WDAT, WE, WA, RA

Outputs: RDAT

Storage: Two Flip-Flops, named RAM

a) Whenever WE is high at the rising edge of CLK, one bit of data (WDAT) is written to the flip-flop in RAM at the address specified by the WA input. I.e. RAM[0] is assigned the value WD when WE is high and WA is 0, and RAM[1] is assigned the value WD when WE and WA are both high.
b) The output RDAT is always immediately assigned the value of the bit of RAM selected by RA. You may either read RDAT out with one cycle of delay or with zero delay (i.e. with or without an extra flip-flop in the path after the RAM flip-flops are selected down by RA).

module SRAM2 (**input** CLK, WDAT, WE, WA, RA, **output** RD)

reg [1:0] RAM;

6. [15 points] Use Shannon's expansion theorem around **a** and **b** for the function

Y = ab'cdef + a'b'c'd'e + b'c'ef' + abcde'f

so that it can be implemented using only four-variable function generators (4 variable LUT). Draw a block diagram to indicate how **Y** can be implemented using only four-variable function generators. Indicate the function realized by each four-variable function generator.

Shannon Practice

$$f(x,y,z) = (x+y+z')(x'+y'+z)(z+y)$$

→ Expansion around x

Form $f(x,y,z) = [x' + f(0,y,z)] \cdot [x + f(1,y,z)]$

$$f(x,y,z) = [x' + y_0] \cdot [x + y_1]$$

↳ 2 variable funcs. → LUT2s

Case 1: $x = 0$

$y_0 = (y+z')(1)(z+y) = \boxed{(y+z')(y+z)}$

$(x+y)(x+z) = x + yz$

$y_0 = y + zz'$ → 0

$y_0 = y \rightarrow x' + y_0 = \underline{x' + y}$

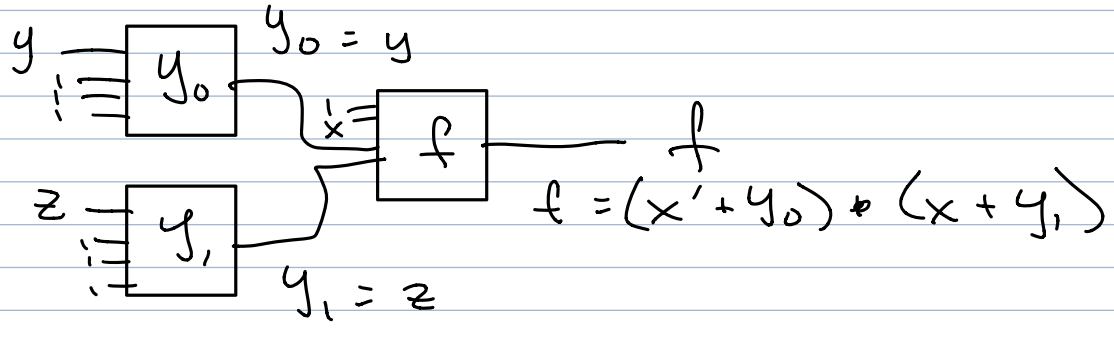
Case 2: $x = 1$

$y_1 = (1)\boxed{(y'+z)(z+y)} = z + yy'$

$y_1 = z \rightarrow x + y_1 = \underline{x + z}$

$$f(x,y,z) = (x'+y_0)\cdot(x+y_1) \leftarrow$$

$$= \boxed{(x'+y)\cdot(x+z)}$$



6.16 Decompose the following function using Shannon's decomposition around the variable *X6*. Do not simplify the function.

$$F = X_1'X_2X_3'X_4X_6 + X_2'X_3'X_4X_6' + X_2'X_4' + X_3X_4X_5X_6 + X_3'X_4X_6' + X_1X_3$$

Write an expression for F in terms of the decomposed functions and X_6.

6.17 Use Shannon's expansion theorem around *a* and *b* for the function

$$Y = abcde + cde'f + a'b'c'def + bcdef' + ab'cd'ef' + a'bc'de'f + abcd'e'f$$

so that it can be implemented using only 4-variable function generators. Draw a block diagram to indicate how Y can be implemented using only 4-variable function generators. Indicate the function realized by each 4-variable function generator.

$$Y = a'Z_0 + aZ_1 = a'b'y_0 + a'by_1 + ab'y_2 + aby_3$$

y_0 at $a=0$ $b=0$

$$y_0 = cde'f + c'def$$

y_1 at $a=0$ $b=1$

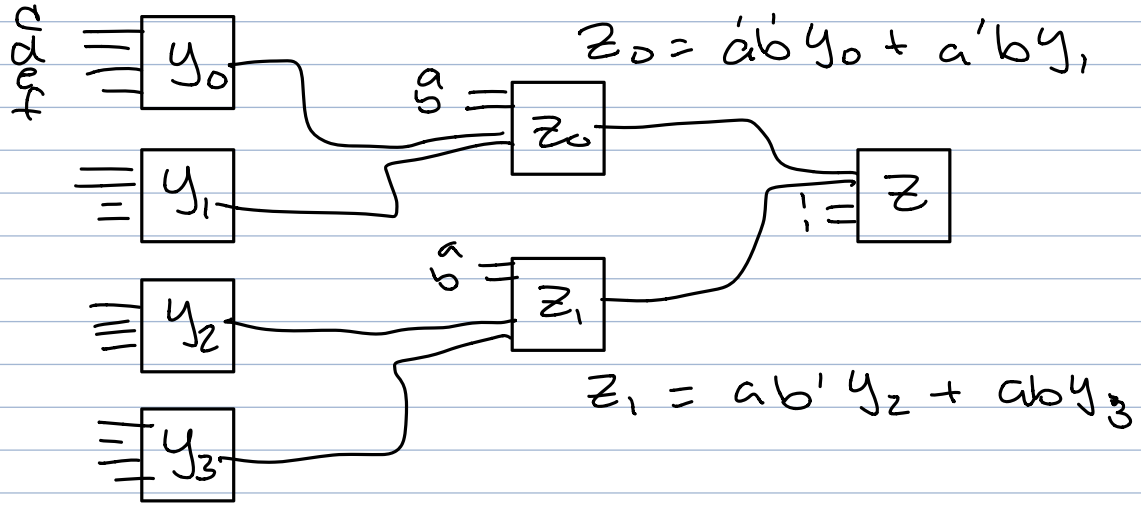
$$y_1 = cde'f + cdef' + c'de'f$$

y_2 at $a=1$ $b=0$

$$y_2 = cde'f + cd'ef'$$

y_3 at $a=1$ $b=1$

$$y_3 = cde + cde'f + cdef' + cd'e'f$$



6.18 Use Shannon's expansion theorem around *e* and *f* for the function

$$Y = ab'cdef + a'bc'd'e + b'c'ef' + abcde'f$$

so that it can be implemented using a minimum number of 4-variable functions. Rewrite *Y* to indicate how it will be implemented using 4-variable function generators and draw a block diagram. Indicate the function generated by each function generator.

$y = e'z_0 + ez_1 = e'f'y_0 + e'fy_1 + ef'y_2 + efy_3$

y_0 at $e=0\ f=0$

$y_0 = 0$

y_1 at $e=0\ f=1$

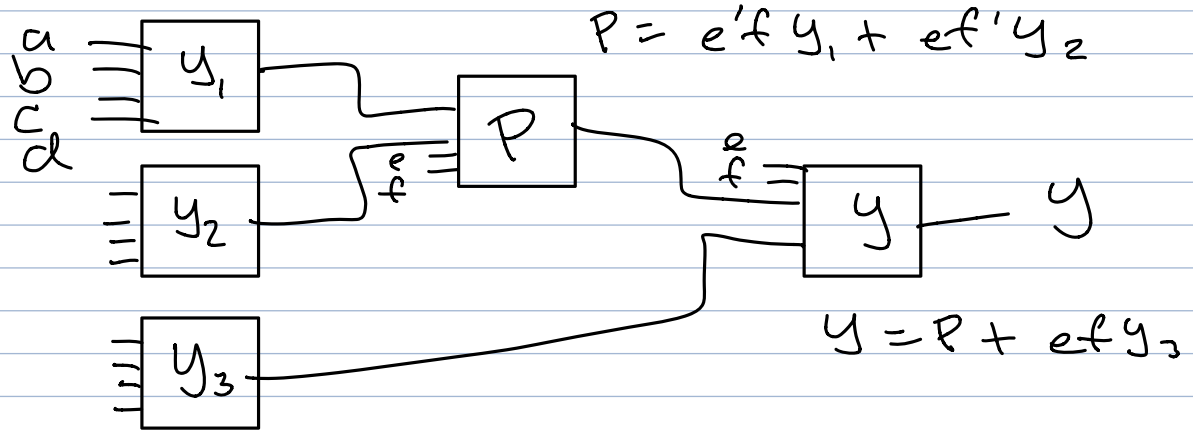
$y_1 = abcd$

y_2 at $e=1\ f=0$

$y_2 = a'bc'd' + b'c'$

y_3 at $e=1\ f=1$

$y_3 = ab'cd + a'bc'd'$



$P = e'fy_1 + ef'y_2$

$y = P + efy_3$

Van - Shaman:

6 vars + LUT 4s

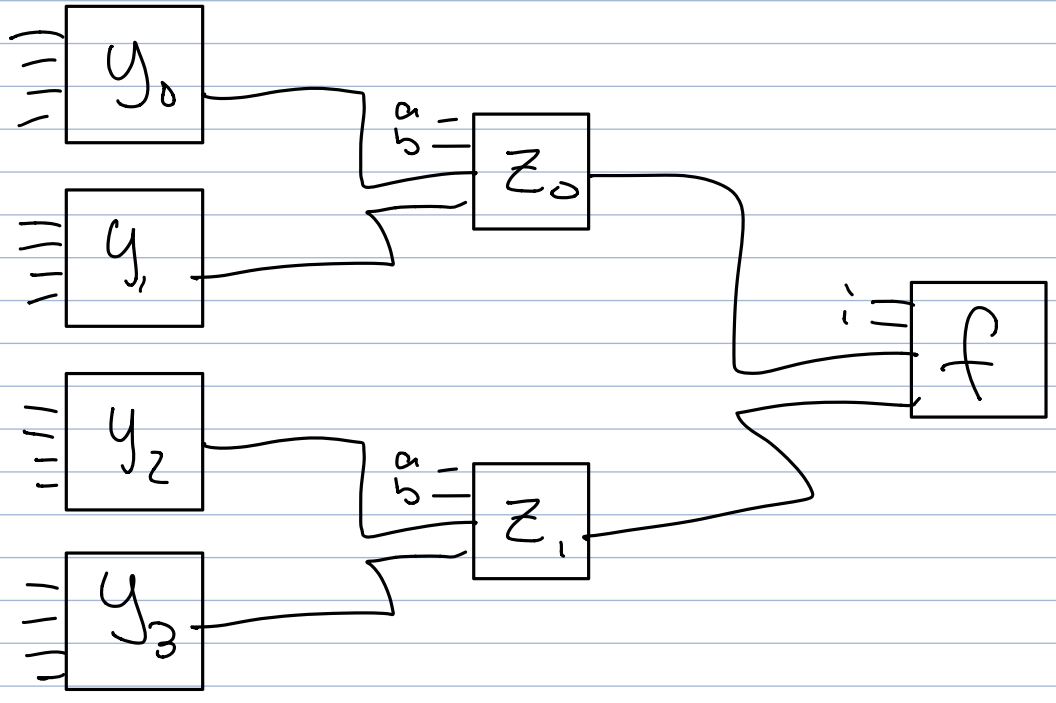
$f(a,b,c,d,e,f)$

$f = a'b'y_0 + a'b\,y_1 + ab'y_2 + ab\,y_3$

↳ $y(c,d,e,f)$

LUT 4 ✓

$f = z_0 + z_1 = a'b'y_0 + a'b\,y_1 + ab'y_2 + ab\,y_3$



6.18 Use Shannon's expansion theorem around e and f for the function

$$Y = ab'cdef + a'bc'd'e + b'c'ef' + abcde'f$$

so that it can be implemented using a minimum number of 4-variable functions. Rewrite Y to indicate how it will be implemented using 4-variable function generators and draw a block diagram. Indicate the function generated by each function generator.

→ LUT4 w/o Shannon's decomp.

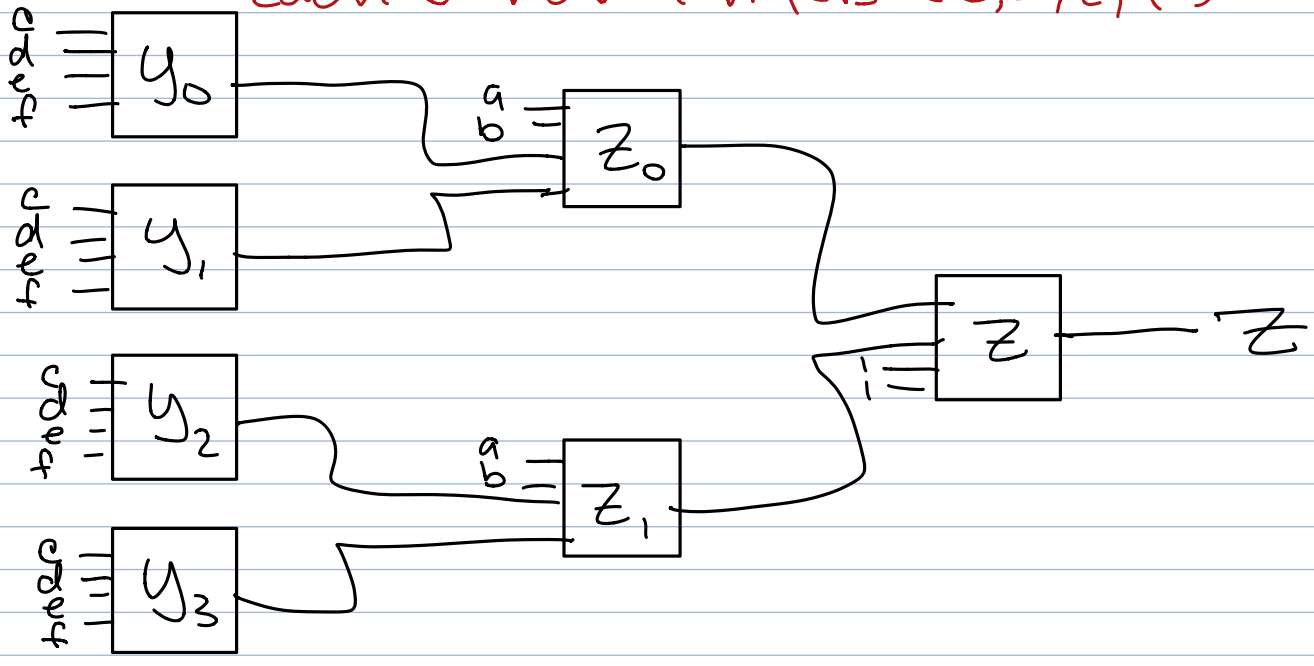
$Z(a,b,c,d,e,f)$

$Z = a'Z_0 + aZ_1$ → $Z_{0/1}$ have 5 inputs

$Z = a'b'y_0 + a'by_1 + ab'y_2 + aby_3$

↓

each y has 4 inputs (c,d,e,f)



6.19 **(a)** Use Shannon's expansion theorem around a for the function

$$Y = ab'cd'e + a'bc'd'e + b'c'e + abcde$$

so that it can be implemented using 4-variable functions.

$y(a,b,c,d,e) = a'z_0 + az_1$

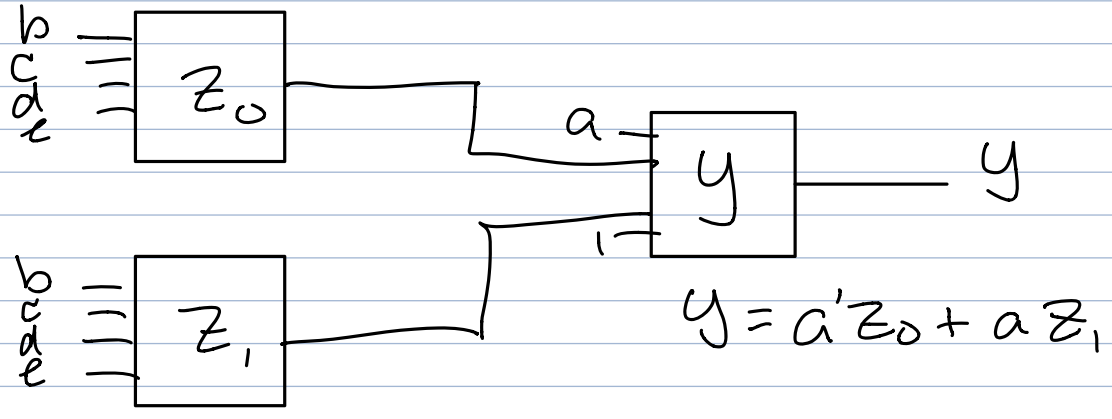
z_0 and z_1 have vars. b,c,d,e

Case 1: $a = 0$

$z_0 = bc'd'e + b'c'e +$

Case 2: $a = 1$

$z_1 = b'cd'e + b'c'e + bcde$



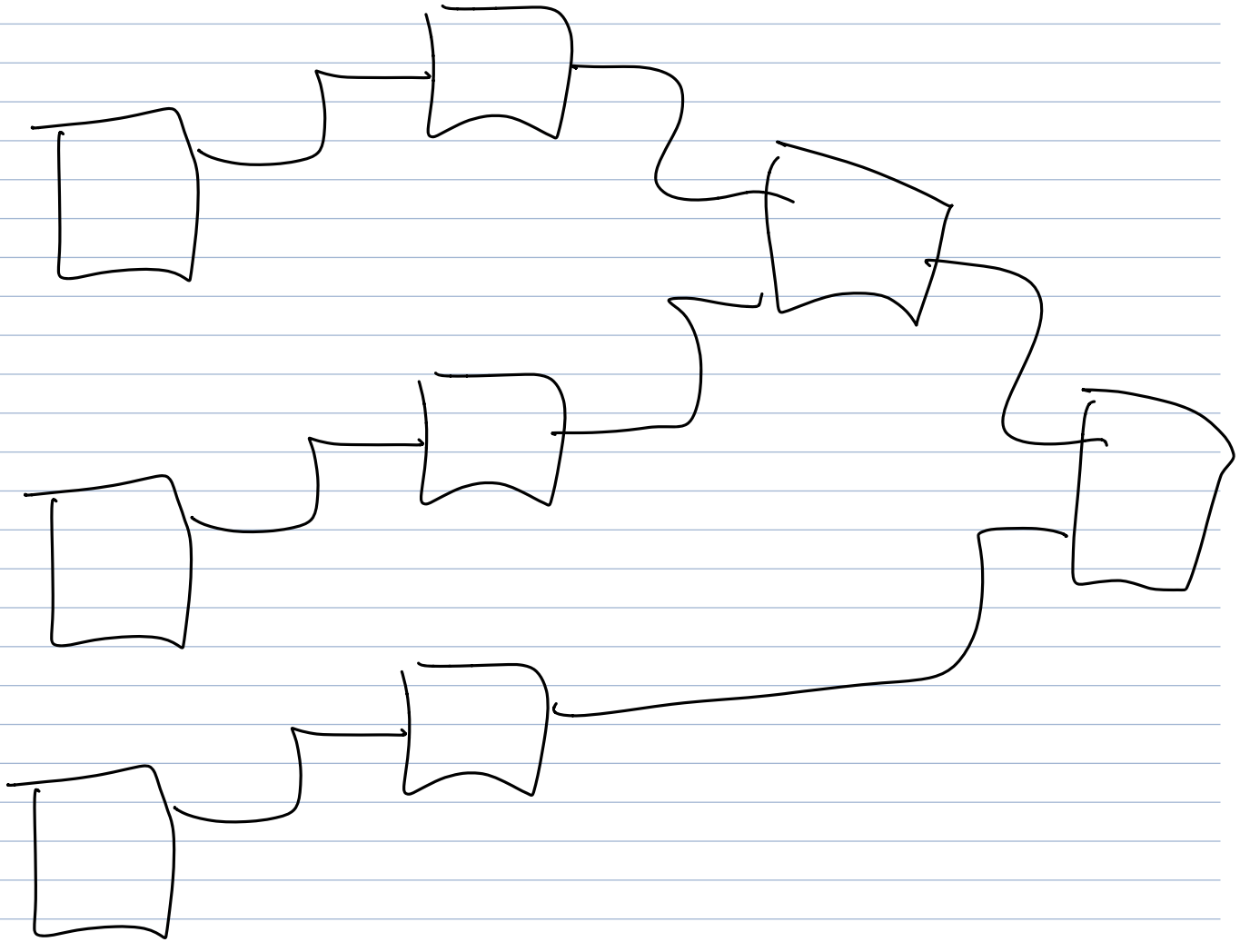
6.20 **(a)** If logic blocks of Figure 6-1(a) are used, how many LUTs are required to build a 4-bit adder with accumulator?

(b) If an FPGA with built-in carry-chain logic as shown in Figure 6-11 is used, how many 4-input LUTs are required?

(c) Design a 4-bit adder-subtracter with accumulator using an FPGA with carry-chain logic and 4-input LUTs. Assume a control signal *Su* which is 0 for addition and 1 for subtraction. Show the required connections on a diagram similar to that shown in Figure 6-11 and give the function realized by each LUT.

6.22 **(a)** Use Shannon's expansion theorem to expand the following function around A and then expand each sub-function around D:

$$Z = AB'CD'E'F + A'BC'D'EF' + B'C'E'F + A'BC'E'F' + ABCDE$$



6.36 Consider the Verilog code

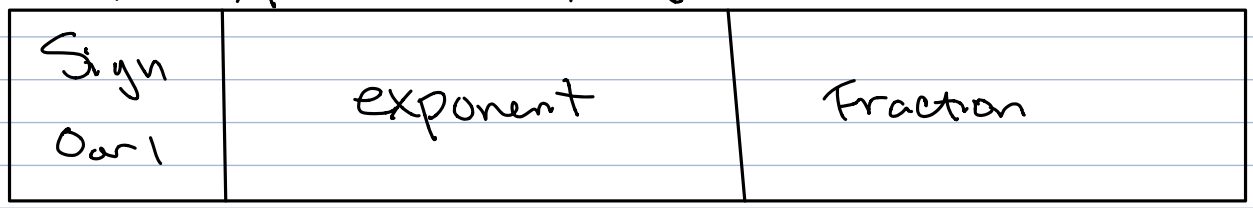
```
module example(a,b);
input[1:0] a;
output[1:0] b;
reg[1:0] b;

always @(a)
begin
   case(a)
   0: b = 2'd3;
   1: b = 2'd2;
   2: b = 2'd1;
   3: b = 2'd1;
   endcase
end

endmodule
```

(a) Show the hardware you would obtain if you synthesize the foregoing Verilog code without any optimizations. Explain your reasoning.

(b) Show optimized hardware emphasizing minimum area. Show the steps and the reasoning by which you obtained the optimized hardware.

7.2 Convert the following decimal numbers in the IEEE single precision format:
(i) 25.25, **(ii)** 2000.22, **(iii)** 1, **(iv)** 0, **(v)** 1000, **(vi)** 8000, **(vii)** 10^6, **(viii)** -5.4, **(ix)** 1.0×2^{-140}, **(x)** 1.5×10^9

0, positive . 1, negative



Want 1.____ stuff

i.) 25.25

Bias

$25 = 11001.01$

$1.1001 \rightarrow$ exp. 2^4

Single prec = 127
Double prec = 1023

fraction

for bias: $127 + 4 = 131$

0	1000 0011	1001 0100 0000 0000 0000 0000

→ fill zeros

.25 ×2 0 ↓
.5 ×2 1

(viii.) −5.4

Sign = 1

$5.4 = 101.0110\ 0110\ 0110\ \ldots$ $\times 2^2$

Exp. = 127 + 2 = 129 = 1000 0001

Frac. = 0101 1001 1001 1001 ...

7.3 Convert the following decimal numbers to IEEE double precision format:
(i) 25.25, **(ii)** 2000.22, **(iii)** 1, **(iv)** 0, **(v)** 1000, **(vi)** 8000, **(vii)** 10^6, **(viii)** -5.4, **(ix)** 1.0×2^{-140}, **(x)** 1.5×10^9

ii) 2000.22 = 111 1101 0000. 0011 1000 0101 0001 1111

bias = 1023

$\times 2^{10}$

Sign = 0

Exp = 1023 + 10 = 1033 = 0100 0000 1001

Frac = 11 1101 0000 0011 1000 0101 0001 1111

Sign	Exp.	Frac.
1 bit	11 - bit	52 - bit

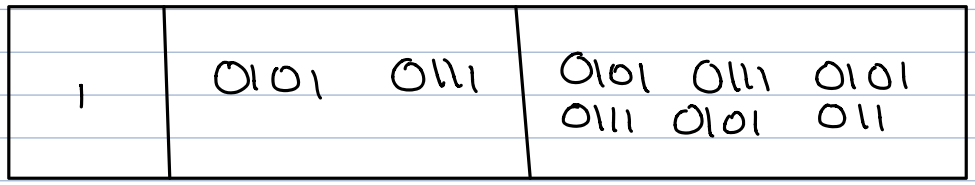
63 ... 0

0	100 0000 1001	1111 0100 0000 1110 0001 0100 0111 1100 0000...

* doesn't match book answer

7.4 What do the following hex representations mean if they are in IEEE single precision format?
(i) ABABABAB, **(ii)** 45454545, **(iii)** FFFFFFFF, **(iv)** 00000000, **(v)** 11111111, **(vi)** 01010101

(i.) ABABABAB



$-1.$

$127 - x = 87$

$x = -40$

$.34117\overline{64}$

$$-1.34117\overline{64} \times 2^{-40}$$

★ Remember to put decimal (.) in converter or mult by 2^{-x}

BCD → reps. 0-9

1 0 2

BCD: 0001 0000 0010

→ each digit in dec. is a single bin #

8.10 Hamming codes are used for error detection and correction in communication and memory systems. Error detection and correction capability is incorporated in these codes by inserting extra bits into the data word. Addition of one parity bit can detect odd number of bit flips, but no error correction is possible with one parity bit. A (7,4) Hamming code has 4 bits of data but 7 bits in total, including the 3 parity bits. It can detect two errors and correct one error. This code can be constructed as follows: If we denote data bits as $d_4d_3d_2d_1$, the encoded code word would be $d_4d_3d_2p_4d_1p_2p_1$, where p_4, p_2, and p_1 are the added parity bits. These bits must satisfy the following conditions for even parity:

$$p_4 = d_2 \text{ XOR } d_3 \text{ XOR } d_4; \quad \ldots\ldots(1)$$
$$p_2 = d_1 \text{ XOR } d_3 \text{ XOR } d_4; \quad \ldots\ldots(2)$$
$$p_1 = d_1 \text{ XOR } d_2 \text{ XOR } d_4; \quad \ldots\ldots(3)$$

When the 7 bits are received/decoded, an error syndrome $S_3S_2S_1$ is calculated as follows:

$$p_4 \text{ XOR } d_2 \text{ XOR } d_3 \text{ XOR } d_4 = S_3; \quad \ldots\ldots(4)$$
$$p_2 \text{ XOR } d_1 \text{ XOR } d_3 \text{ XOR } d_4 = S_2; \quad \ldots\ldots(5)$$
$$p_1 \text{ XOR } d_1 \text{ XOR } d_2 \text{ XOR } d_4 = S_1; \quad \ldots\ldots(6)$$

The syndrome indicates which bit is wrong. For example, if the syndrome is 110, it indicates that bit 6 from right end (i.e., d3) has flipped. If $S_3S_2S_1$ is 000, there is no error.

(a) Is there any error in the code word 0110111? If yes, which bit? What was the original data? What must be the corrected code word?

(b) How will these 6 equations get modified for odd parity? Write the 6 equations for odd parity.

(c) Write a Verilog module for error detection without using tasks. The inputs to the module are the 7-bit encoded data word and the type of parity, and the output is the syndrome. The type of parity is encoded as 0 for odd parity and 1 for even parity.

```
module  error_detector(data, PARITY, Syndrome)
{
}
```

S[2] = (Parity) ? (XOR: $p_4\ d_2\ d_3\ d_4$) : (XNOR: $p_4\ d_2\ d_3\ d_4$)

5 variable eq.

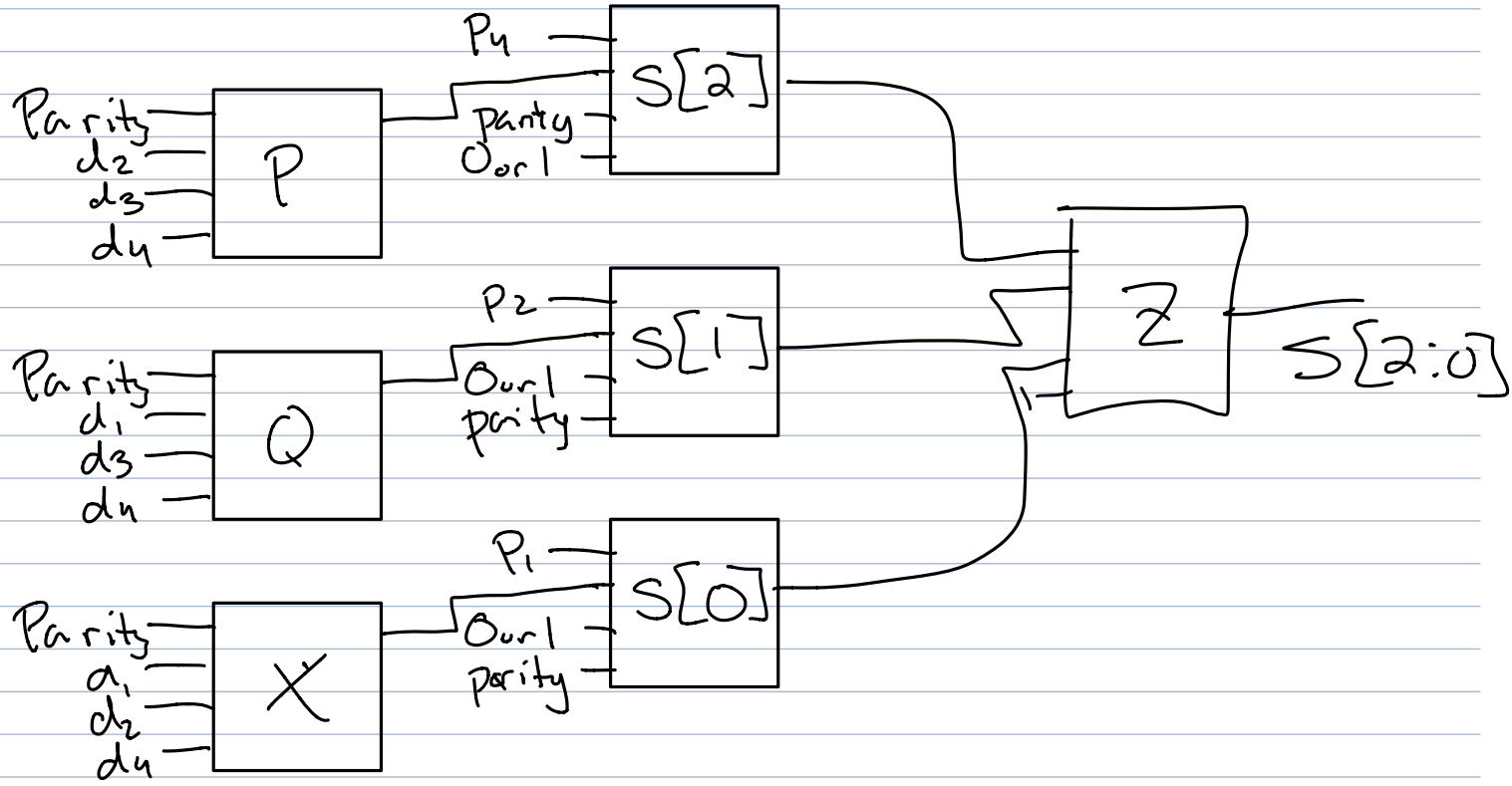
Sub func:

S[2] (parity, p_4, d_2, d_3, d_4)

S[1] (parity, p_2, d_1, d_3, d_4)

S[0] (parity, p_1, d_1, d_2, d_4)

Vars.

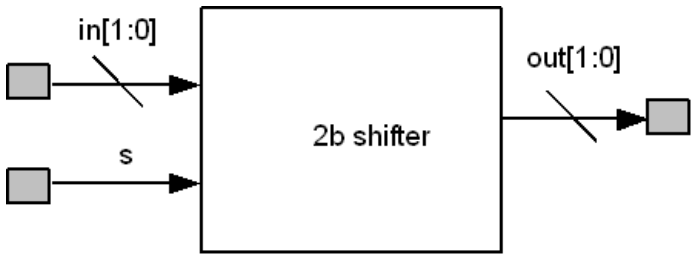

Parity
d_2
d_3
d_4
P
P_4
Parity
0 or 1
S[2]
Parity
d_1
d_3
d_4
Q
P_2
0 or 1
parity
S[1]
Parity
d_1
d_2
d_4
X
P_1
0 or 1
parity
S[0]
Z
1
S[2:0]

8.24 Write a Verilog model for an N-bit comparator using an iterative circuit. In the module, use the parameter N to define the length of the input bit vectors A and B. The comparator outputs should be $EQ = 1$ if $A = B$, and $GT = 1$ if $A > B$. Use a

for loop to do the comparison on a bit-by-bit basis, starting with the high-order bits. Even though the comparison is done on a bit-by-bit basis, the final values of EQ and GT apply to A and B as a whole.

Exercise 1 – 2b shifter

- This is the same logic block as in Q2a of HW #2, but with bus notation for the inputs and outputs. This is a purely combinational logic block. The logic equations are:
  - out[0] = s' ● in[0]
  - out[1] = s' ● in[1] + s ● in[0]

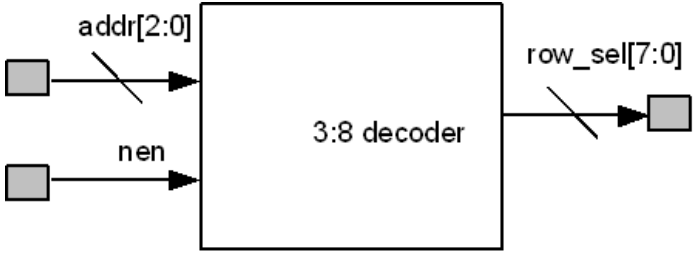


Ex 1 Solution

```
module shifter(in, out, s);
   input [1:0] in;
   input       s;

   output [1:0] out;

   wire   s;
   wire [1:0] in;
   wire   out;

   assign out[1] = (~s) & in[1] | s & in[0];
   assign out[0] = (~s) & in[0];

endmodule
```

Ex 2 – 3:8 row decoder with enable

- This decoder has inputs `addr[2:0]` and an active low enable `nen`. It drives 8 active high output lines `row_sel[7:0]`, one of which is driven when `nen` is asserted.



Ex 2 Solution

```
module row_decoder(row_sel, addr, nen);
   input [2:0] addr;
   input       nen;

   output [7:0] row_sel;

   wire [2:0]    addr;
   wire  nen;
   reg [7:0]     row_sel;

   // Use a case statement
   always @(addr or nen)
     begin
    if (nen)
      row_sel = 8'h0;
    else
      case (addr)
        3'h0: row_sel = 8'b0000_0001; // The _ is just for readability
        3'h1: row_sel = 8'b0000_0010;
        3'h2: row_sel = 8'b0000_0100;
        3'h3: row_sel = 8'b0000_1000;
        3'h4: row_sel = 8'b0001_0000;
        3'h5: row_sel = 8'b0010_0000;
        3'h6: row_sel = 8'b0100_0000;
        3'h7: row_sel = 8'b1000_0000;
      endcase // case(addr)
     end // always @ (addr or nen)

endmodule // row_decoder
```

Ex 3 – 8b register with load and synchronous reset

- This block implements an 8b wide register from DFFs. All flops are driven by a common clock and have a common reset `rst`. An input mux allows new data to be loaded into the register when `ld` is high; otherwise, the old data is recirculated.

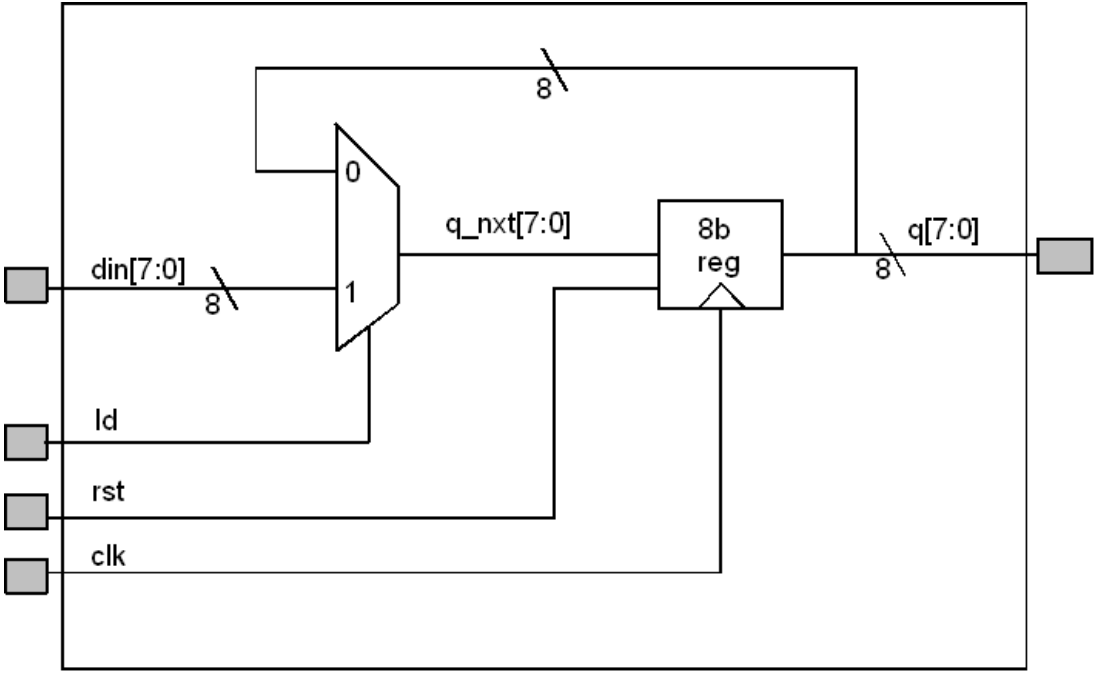


FIGURE 2-55: Sequential Machine Model Using Equations

```
// The following is a description of the sequential machine of
// the BCD to Excess-3 code converter in terms of its next state
// equations obtained as in Figure 1-25. The following state assignment was
// used: S0-->0; S1-->4; S2-->5; S3-->7; S4-->6; S5-->3; S6-->2

module Code_Converter(X, CLK, Z);
   input    X;
   input    CLK;
   output   Z;

   reg      Q1;
   reg      Q2;
   reg      Q3;

   always @(posedge CLK)

   begin
   Q1 <= #10 (~Q2);
   Q2 <= #10 Q1;
   Q3 <= #10 (Q1 & Q2 & Q3) | ((~X) & Q1 & (~Q3)) |
            (X & (~Q1) & (~Q2));
   end
   assign #20 Z =  ((~X) & (~Q3)) | (X & Q3);

endmodule
```

LUTs

blocking =

non-blocking <=

← LUT + FF

← FF

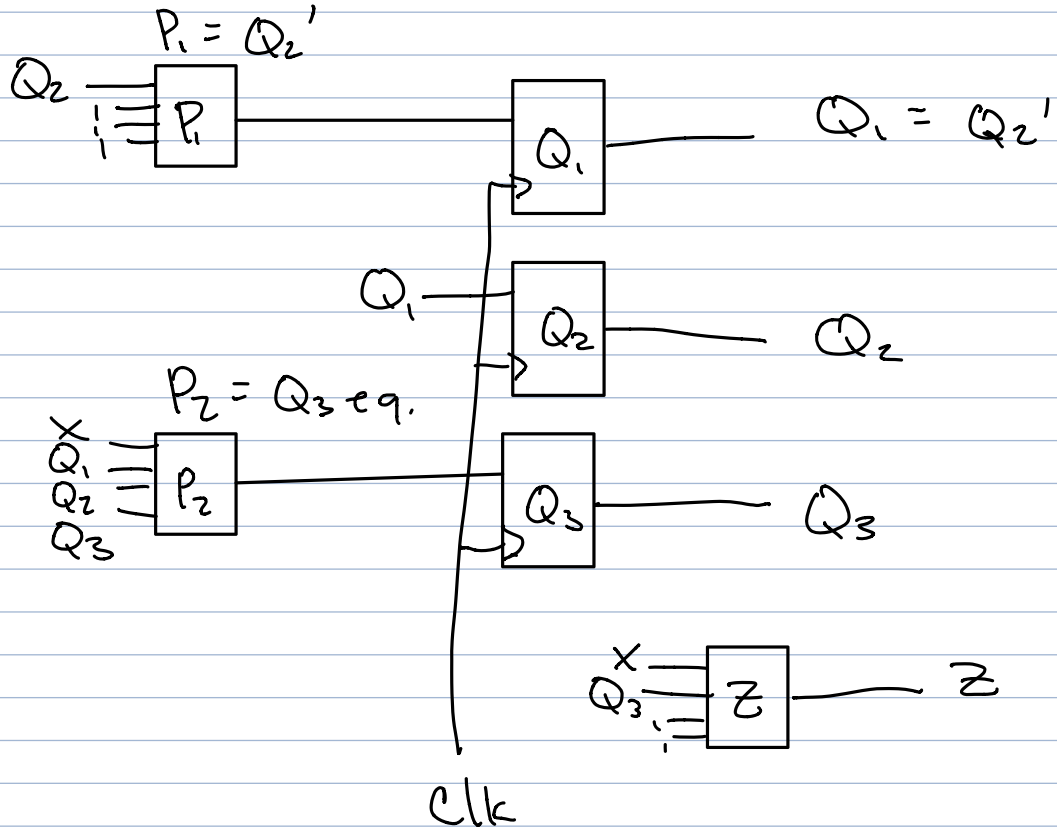
← FF + 1 LUT

$Q_3 = Q_1Q_2Q_3 + x'Q_1Q_3' + xQ_1'Q_2'$

$Z = x'Q_3' + xQ_3$ → 1 LUT

→ Value set at Clk = FF

total: 3 FF + 3 LUTs

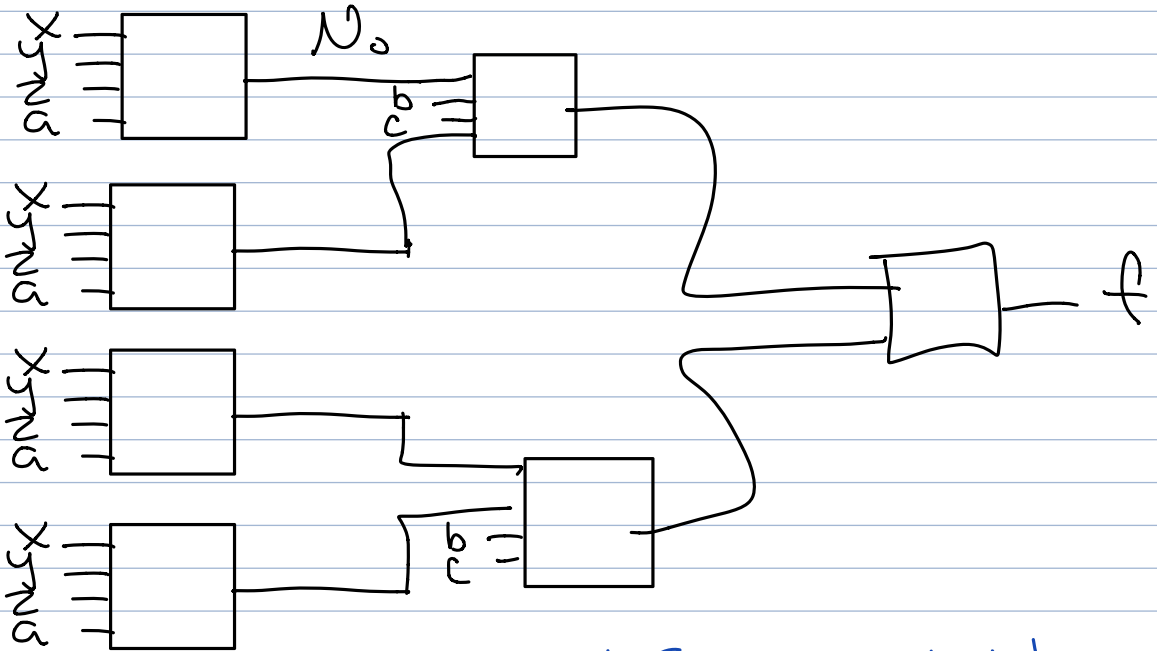


Guides for FFs + LUTs

* FF if a value is set at a clock edge
* LUT # depends on # of variables in your equation.

Ex. LUT4s and Gen. func.

f(x, y, z, a, b, c)



reg [7:0] Q4

```
always @ (posedge clk) begin
    Q4 <= Q1;
end
```

makes 4 FFs

Ex 3 Solution

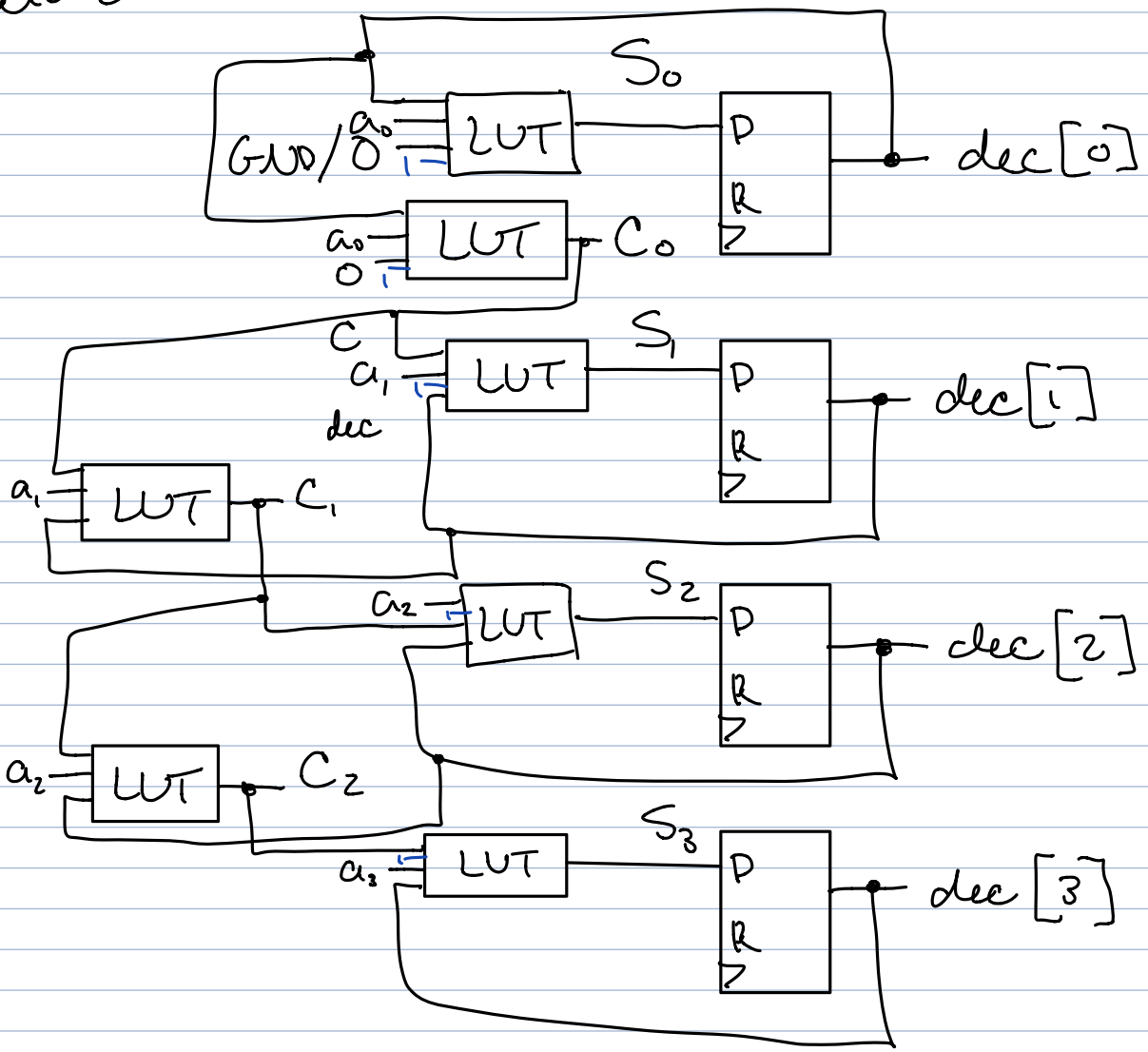
```
module ld_reg8(din, clk, rst, ld, q);

   input [7:0] din;
   input       clk, rst, ld;
   output [7:0] q;

   wire [7:0]    din, q_nxt;
   wire   clk, rst, ld;
   reg [7:0]  q;

   //Logic on register inputs
   assign     q_nxt = ld ? din : q;

   // Update register
   always @(posedge clk)
     q <= rst ? 8'h0 : q_nxt;

endmodule // row_decoder
```

```
module decrementer (input [2:0] a, clk, rst,
                    output reg [3:0] dec)
  always @(posedge clk) begin
    if (rst) → given FF w/ Rst.
      dec <= 4'b0;
    else
      dec <= dec + a;
  end
endmodule
```

[3:0] [2:0] → [3:0]

operation

★ already know: 4 FFs

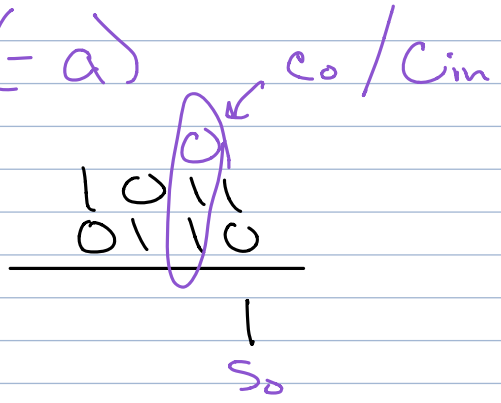


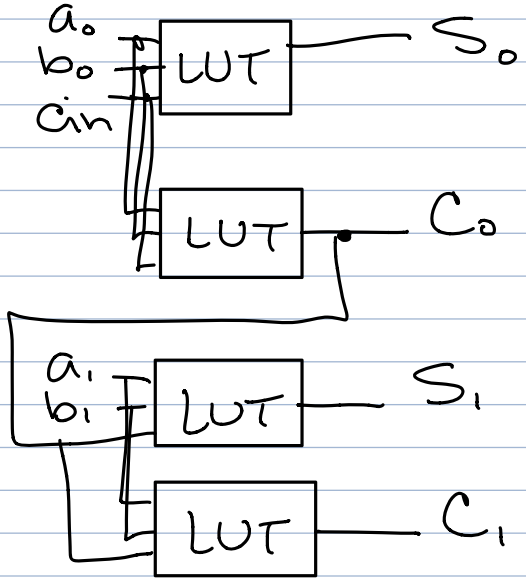
★ this is too hard for the exam!

Adder refresh:

dec = dec - a

↳ dec = dec + (-a)





Topic: Shannon's Decomp + Mapping to LUTs

→ Assign values to variables to simplify into
↓ subfunctions

★ # of variables in subfunction = # of inputs on LUT

Ex. function $z(a,b,c,d)$ can be broken into

$$z(a,b,c,d) = a' \cdot z(0,b,c,d) + a \cdot z(1,b,c,d) = a'z_0 + az_1$$

Ex. $z = abcd'ef' + a'b'c'def' + b'cde'f$

↳ 6 var function, wmt to use LUT5s

- **Decomp. about a:** $z = a'z_0 + az_1$

  set a = 0

  $z_0 = 0 \cdot bcd'ef' + 1 \cdot b'c'def' + b'cde'f$ (0 · bcd'ef' → 0)

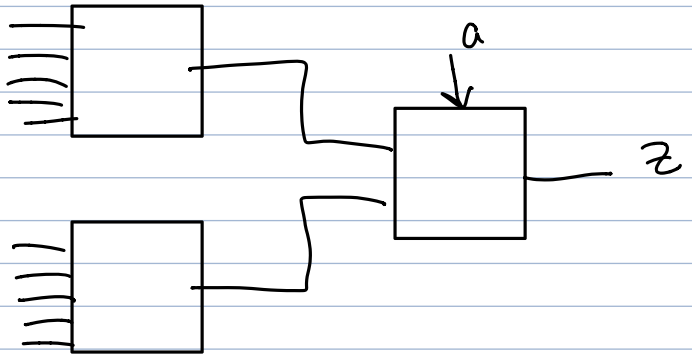
  $z_0 = b'c'def' + b'cde'f$

  set a = 1

  $z_1 = 1 \cdot bcd'ef' + 0 \cdot b'c'def' + b'cde'f$ (0 · b'c'def' → 0)

  $z_1 = bcd'ef' + b'cde'f$

→ 5 vars = 5 input LUT



Ex. (same as above) $Z = abcd'ef' + a'b'c'def' + b'cde'f$

↳ 6 var function, want to use LUT4s

→ Decomp around a then b or a+b at the same time

$Z(a,b,c,d,e,f) = a'Z_0 + aZ_1 = a'b'y_0 + a'by_1 + ab'y_2 + aby_3$ ($y_1 \to 0$)

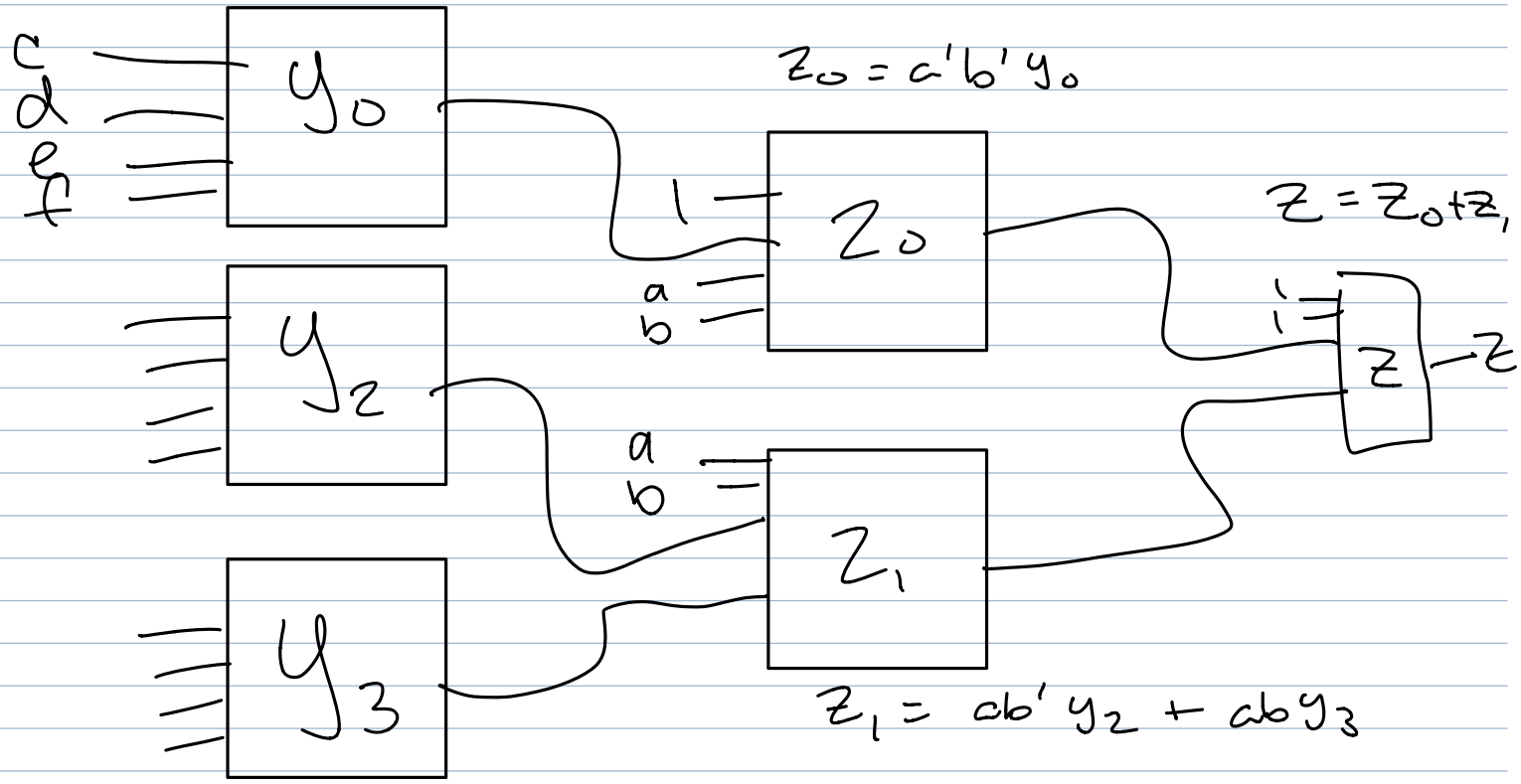
y eq. are 4 var eqs.

Set $a=0$ $b=0$ $\quad y_0 = c'def' + cde'f$

Set $a=0$ $b=1$ $\quad y_1 = 0$ → excluded / null

Set $a=1$ $b=0$ $\quad y_2 = cde'f$

Set $a=1$ $b=1$ $\quad y_3 = cd'ef'$

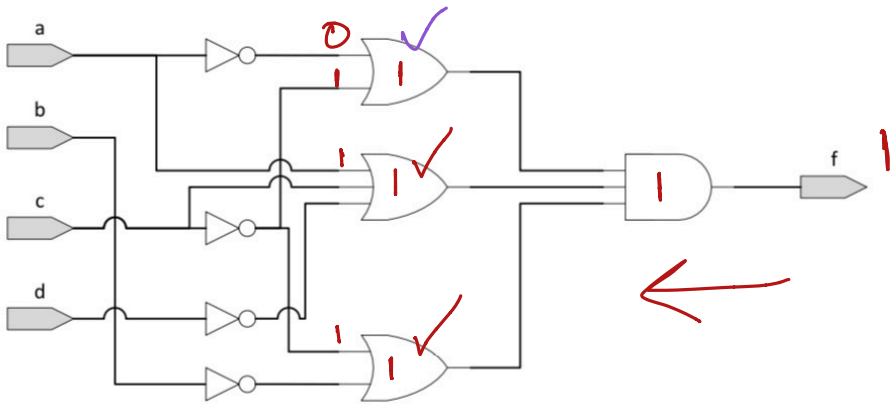


Topic: IEE Floating Point

* NF

1. [10 points] For the logic network shown below, find all static 0 hazards. For each 0-hazard found, specify the conditions which will cause the hazard to appear at the output (i.e. specify the logic values of the variables which are constant and clearly specify the variable that is assumed to be changing). If there are no 0-hazards found, use a K-map to show why this is the case.

ab=
10
c=0



cd \ ab	00	01	11	10
00	1	1	1	1
01	0	0	1	1
11	1	0	0	0
10	1	0	0	0

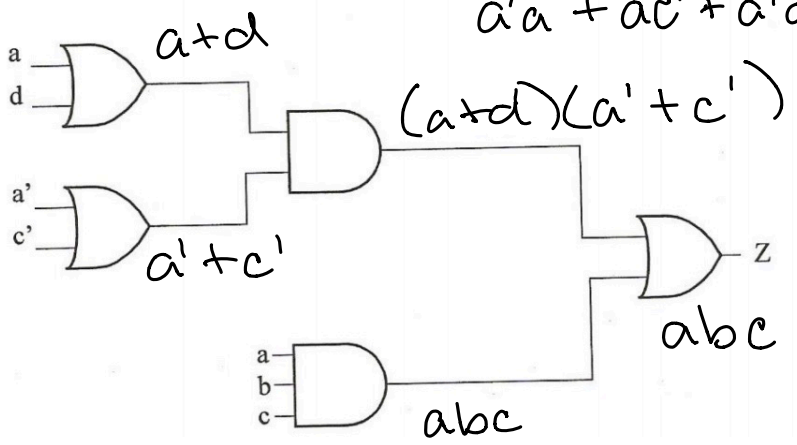
ABCD

0101 ↔ 0111

C changing 0 to 1

2.

For the network shown below, find any/all static 0-hazards. For any 0-hazard found specify the conditions which will cause the hazard to appear at the output (i.e. specify the logic values of the variables which are constant and the variable which is changing) [10 points].



$a'a + ac' + a'd + c'd$

$abc + (a+d)(a'+c')$

cd \ ab	00	01	11	10
00	0	0	1	1
01	1	1	1	1
11	1	1	1	0
10	0	0	1	0

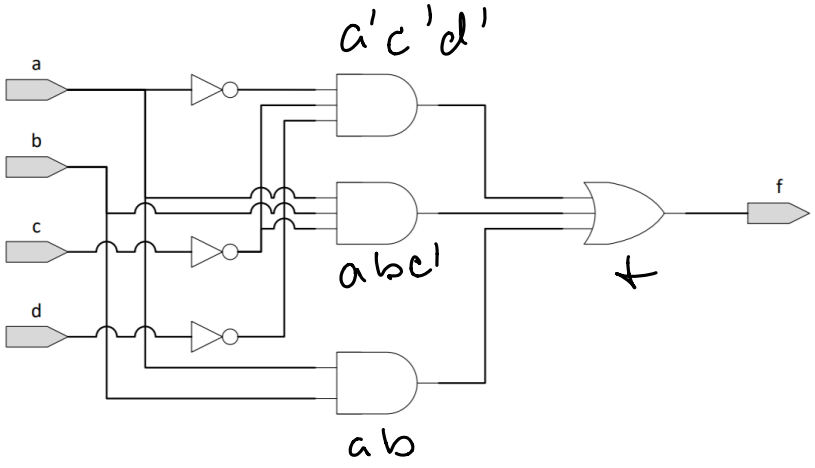
ABCD

0010 ↔ 1010

A Changes

0 ↔ 1

1. [10 points] For the logic network shown below, find all static 1 hazards. For each 1-hazard found, specify the conditions which will cause the hazard to appear at the output (i.e. specify the logic values of the variables which are constant and clearly specify the variable that is assumed to be changing). If there are no 1-hazards found, use a K-map to show why this is the case.



cd \ ab	00	01	11	10
00	1	1	1	0
01	0	0	1	0
11	0	0	1	0
10	0	0	1	0

* all ANDs into an OR is easy w/ the equation method!

Equation:

$$a'c'd' + abc' + ab$$

ABCD: 0100 to 1100

A is changing from 0 ↔ 1

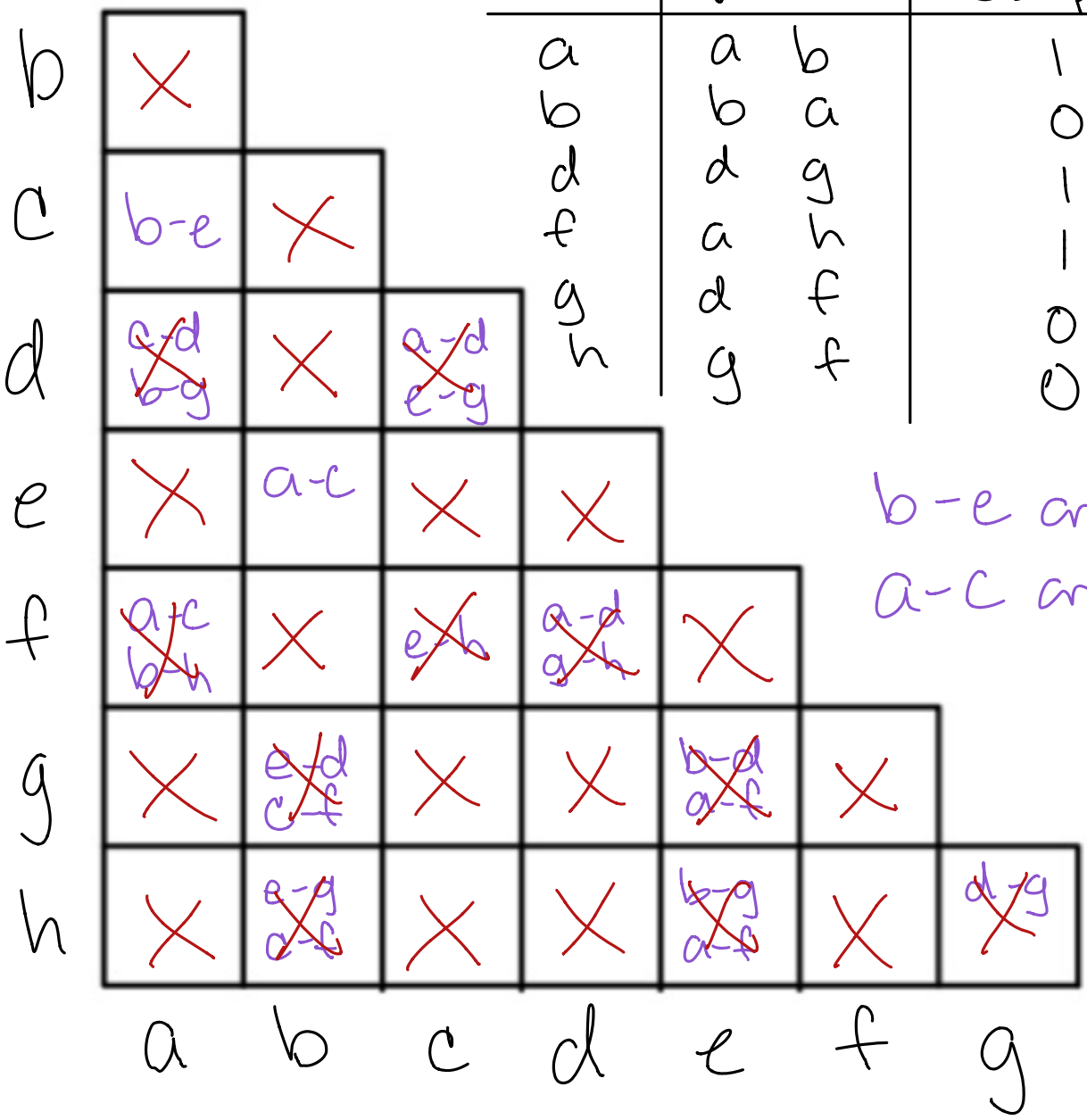
8. [10 points] Reduce the following state table to a minimum number of states clearly identifying the states that are equivalent with one another. Show the final reduced state table.

1. X obvious matches
2. fill table
3. X dif. outputs
4. 'implied pairs,

Present State	Next State X=0	Next State X=1	Present Output
a	a c	b	1
b	b e	a c	0
~~c~~	~~a~~	~~e~~	~~1~~
d	d	g	1
e	b	a	0
f	a	h	1
g	d	f	0
h	g	f	0

For your convenience, an iteration chart is provided below – please label grid along axes, and create additional copies if needed.

PS	NS		Output
a	a	b	1
b	b	a	0
d	d	g	1
f	a	h	1
g	d	f	0
h	g	f	0

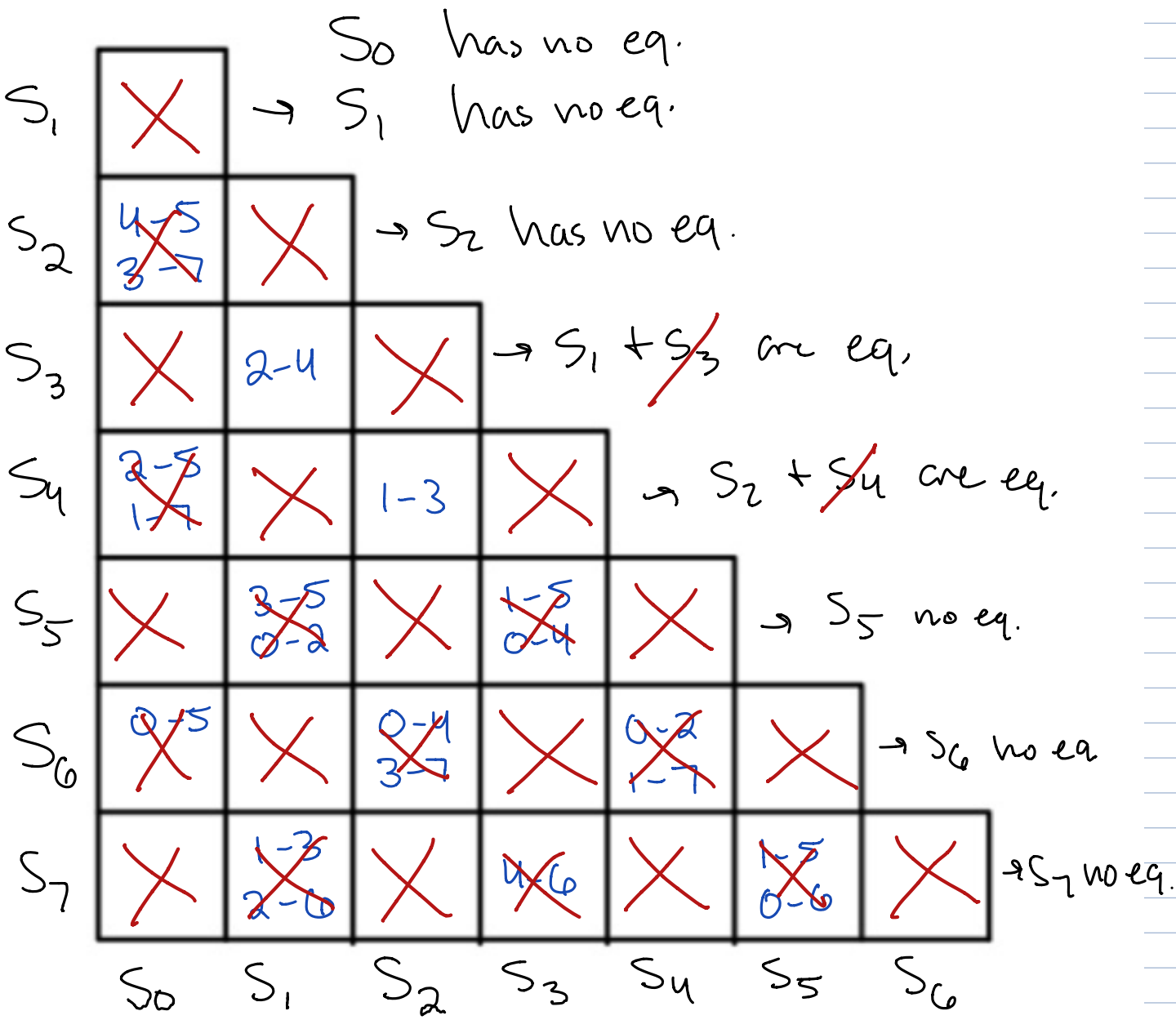


b-e are eq.

a-c are eq.

2.

present state	next state X=0	1	present output
S_0	S_5	S_7	0
S_1	~~S_3~~ 1	S_2	1
S_2	2 ~~S_4~~	1 ~~S_3~~	0
~~S_3~~	~~S_1~~	~~S_4~~	~~1~~
~~S_4~~	~~S_2~~	~~S_1~~	~~0~~
S_5	S_5	S_0	1
S_6	S_0	S_7	0
S_7	S_1	S_6	1



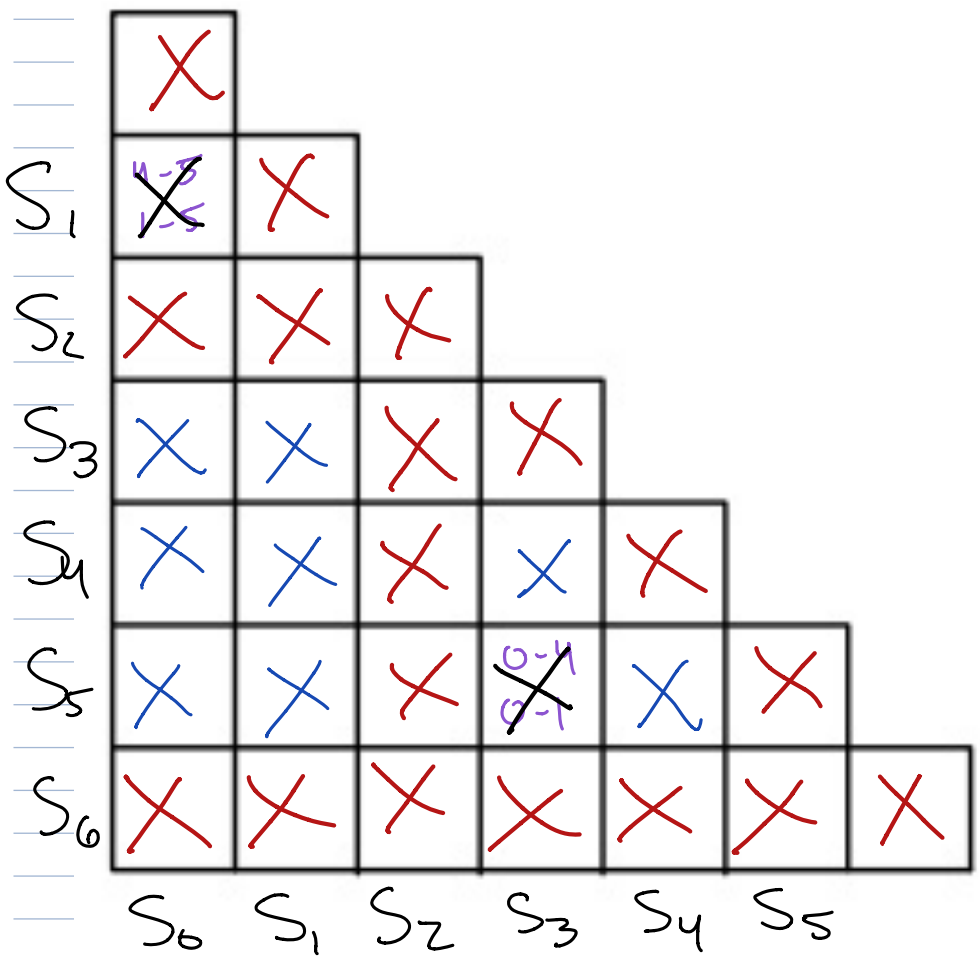
3.

Present State	Next State		Output	
	X=0	X=1	X=0	X=1
S_0	S_4	S_1	1	0
S_1	~~S_6~~ S_5	S_5	1	0
~~S_2~~	~~S_4~~	~~S_1~~	~~1~~	~~0~~
S_3	S_4	S_1	0	1
S_4	~~S_6~~ S_5	S_5	1	1
S_5	S_0	~~S_2~~ 0	0	1
~~S_6~~	~~S_0~~	~~S_2~~	~~0~~	~~1~~

eg.

eg.

⇒ outputs don't match



1. [20 points] Use Shannon's expansion theorem around a and b for the following function

Y(a,b,c,d,e,f) = ((a ^ b) & c & ~d & f) + (a & b & ~c & d & e & f) +
(~a & ~b & (c | e | ~f))

so that it can be implemented using only four-variable function generators (4-input LUTs). Draw a block diagram to indicate how Y can be implemented using only four-variable function generators. Indicate the function realized by each four-variable function generator.

Note the operators in the above are written in Verilog syntax; ~ inversion, & bitwise-AND, | bitwise-OR, ^ bitwise-XOR

$$y = a'b'y_0 + a'by_1 + ab'y_2 + aby_3$$

Case: a=0 b=0

$$y_0 = 0 + 0 + c|e|f' = c|e|f'$$

Case: a=0 b=1

$$y_1 = cd'f + 0 + 0 = cd'f$$

Case: a=1 b=0

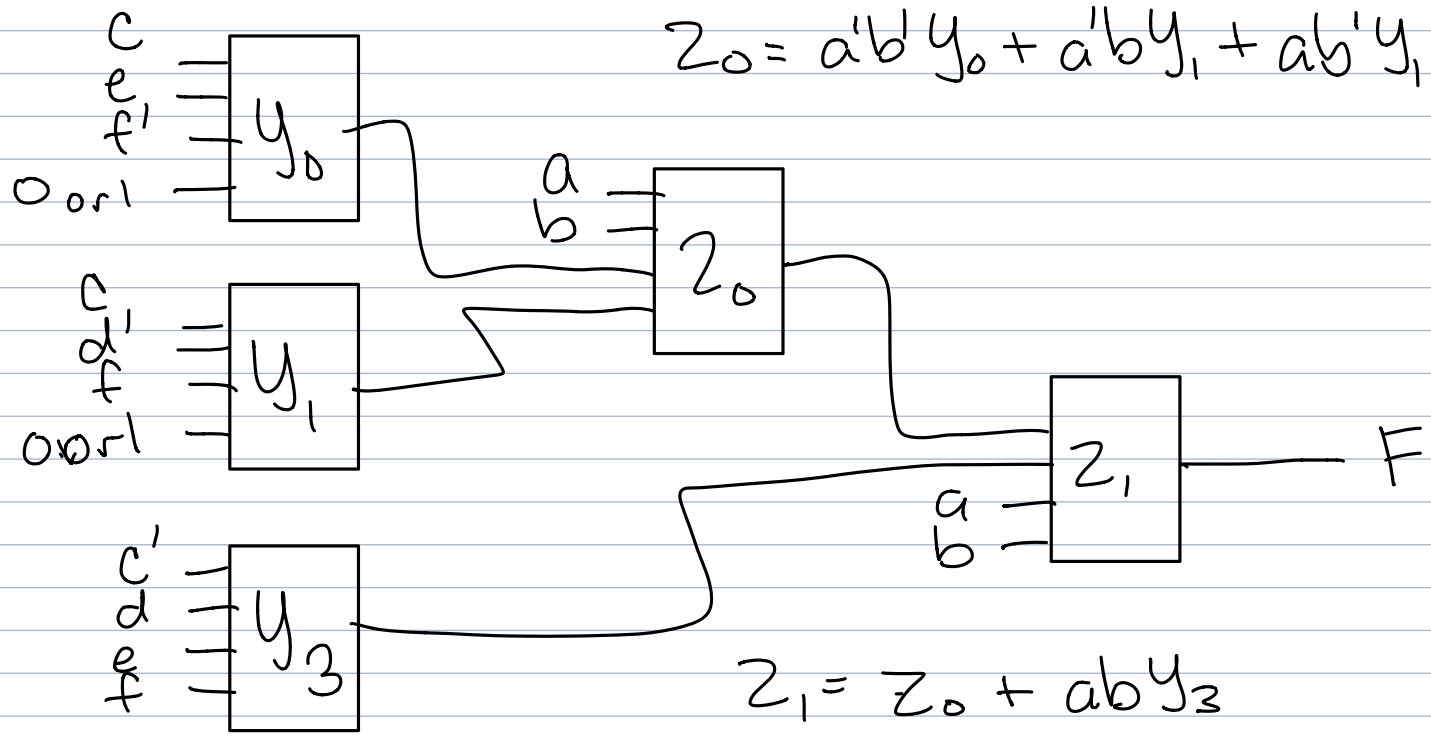
$$y_2 = cd'f + 0 + 0 = cd'f$$

$y_1 = y_2$ so we only use y_1

Case: a=1 b=1

$$y_3 = 0 + c'def + 0 = c'def$$

* remember to add back what you took out.



$$z_0 = a'b'y_0 + a'by_1 + ab'y_1$$

$$z_1 = z_0 + aby_3$$

Shannon: $Z = abcd'ef' + a'b'c'def' + b'cde'f$

$y = a'b'y_0 + a'by_1 + ab'y_2 + aby_3$

Case: $a=0$ $b=0$

$y_0 = c'def' + cde'f$

Case: $a=0$ $b=1$

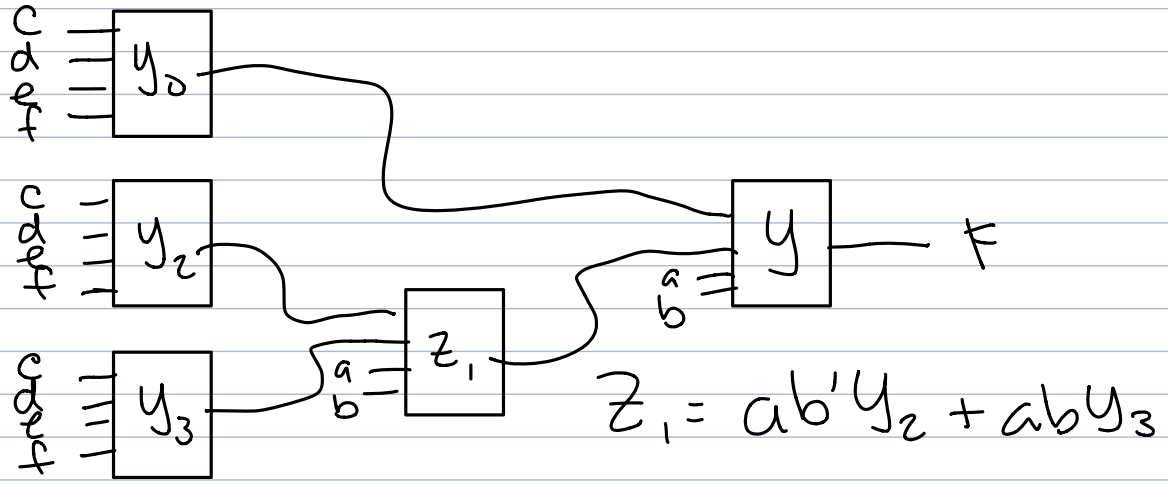
$y_1 = 0$

Case: $a=1$ $b=0$

$y_2 = cde'f$

Case: $a=1$ $b=1$

$y_3 = cd'ef'$



Number Conversions

2. [20 points] Number Conversions

a. Convert the following signed decimal number to a 16-bit 2's-complement signed binary format:

-2021 0000 0111 1110 0101

2's comp: 1111 1000 0001 1011

*make sure you sign extend.

b. Convert the following decimal number to a 16-bit binary coded decimal value:

322

3 → 0011
2 → 0010
2 → 0010

*need 16 bits

0000 0011 0010 0010

c. Convert the following decimal number into a 32-bit unsigned hexadecimal value:

789,514

000C 0C0A

2021
-1024 → 2^{10}
997
-512 → 2^9
485
-256 → 2^8
229
-128 → 2^7
101
-64 → 2^6
37
-32 → 2^5
5
-4 → 2^2
1
-1 → 2^0
0

789514
-786432 → C^4

3082
-3072 → C^2
10 → A^0

d. Convert the following decimal number into 32-bit short precision floating point, using 1 bit for sign, 8 bits for the exponent with a bias of 127, and with 23 bits for the mantissa:

$$\text{value} = (-1)^{\text{sign}} \times 2^{(E-127)} \times \left(1 + \sum_{i=1}^{23} b_{23-i} 2^{-i}\right)$$

expressed as the contatenated value {sign, E, b}

7.40234375 = 0111 0110 0111 = 1.1101100111_2

Sign = 0

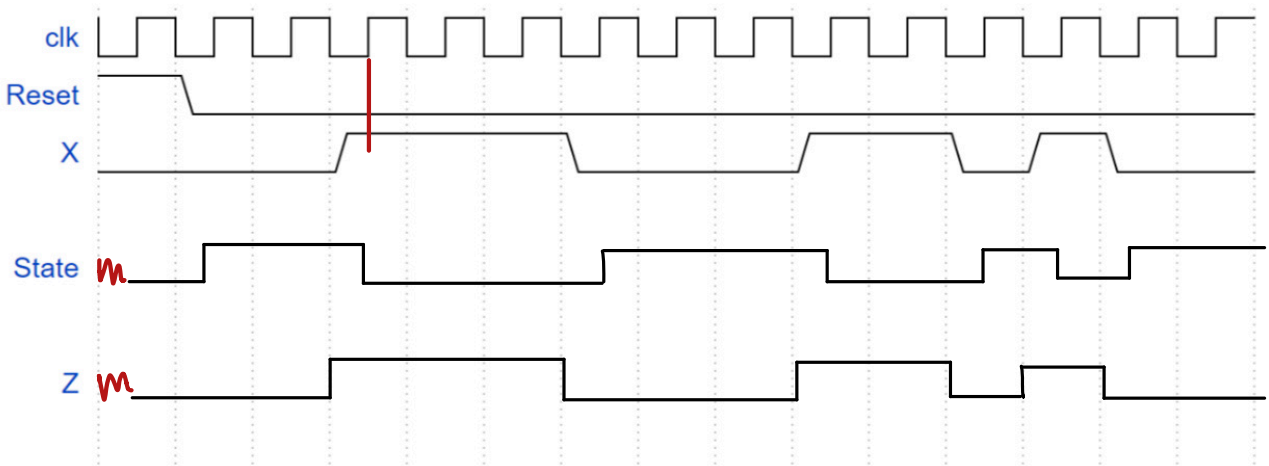
Exp = 127 + 2 = 129 = 1000 0001

b = 11011 00111 → fill to 23 bits.

11. [10 points] Complete the following timing diagram for the output signal **Z**, and the internal input signal **state**. Assume that a basic functional RTL simulation is to be performed.

```
module fsm_network (input clk, X, Reset, output reg Z);
reg state, next_state;
always @(state, X)
   case (state)
         0 : if (X) begin Z=1; next_state=0; end
             else begin Z=0; next_state=1; end
         1 : if (X) begin Z=1; next_state=0; end
             else begin Z=0; next_state=1; end
   endcase
always @(posedge clk)
   if (Reset) state=0;
   else state = next_state;
```

* Check Sensitivity list!



What type of sequential network does this Verilog Model represent?

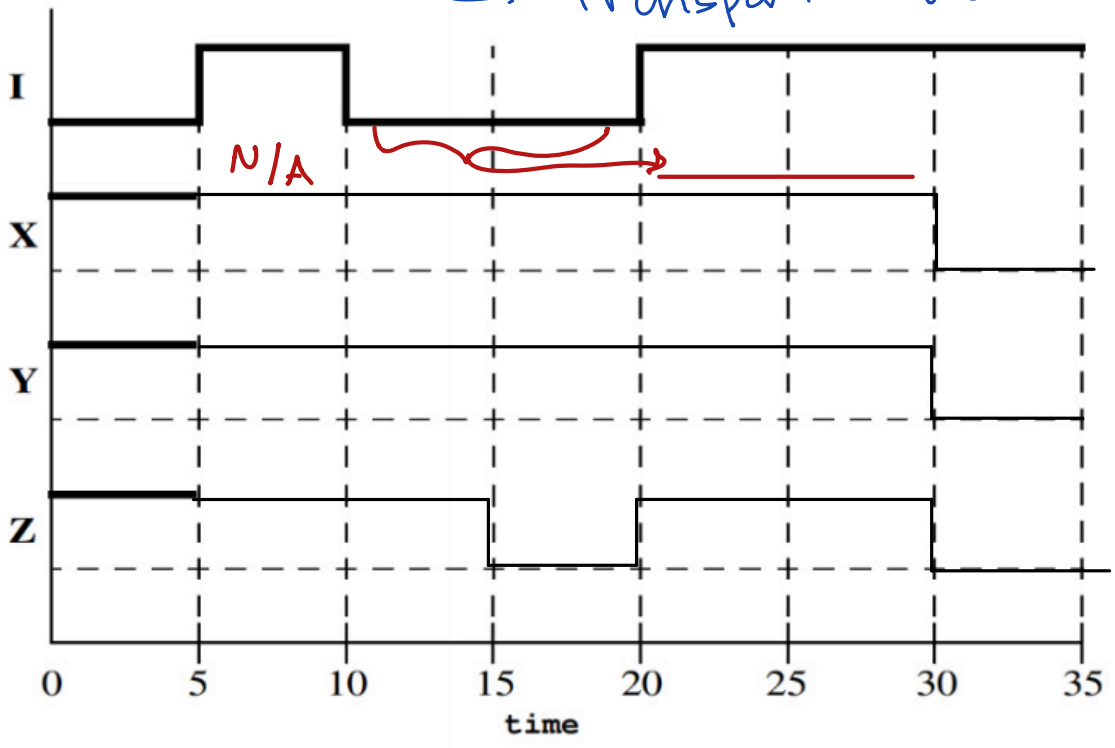
1.

Complete the Timing Diagram for the simulation of the following Verilog Model where input I is driven in the manner that is shown on the diagram.

```
module v_model(input I, output X,Y, output reg Z);
   assign #10 X = ~I;
   not #(10) G1 (Y,I);
   always @(I) Z <= #10 ~I;
endmodule;
```

inertial → pulse ≥ 10 to have an effect

transport → wait to show



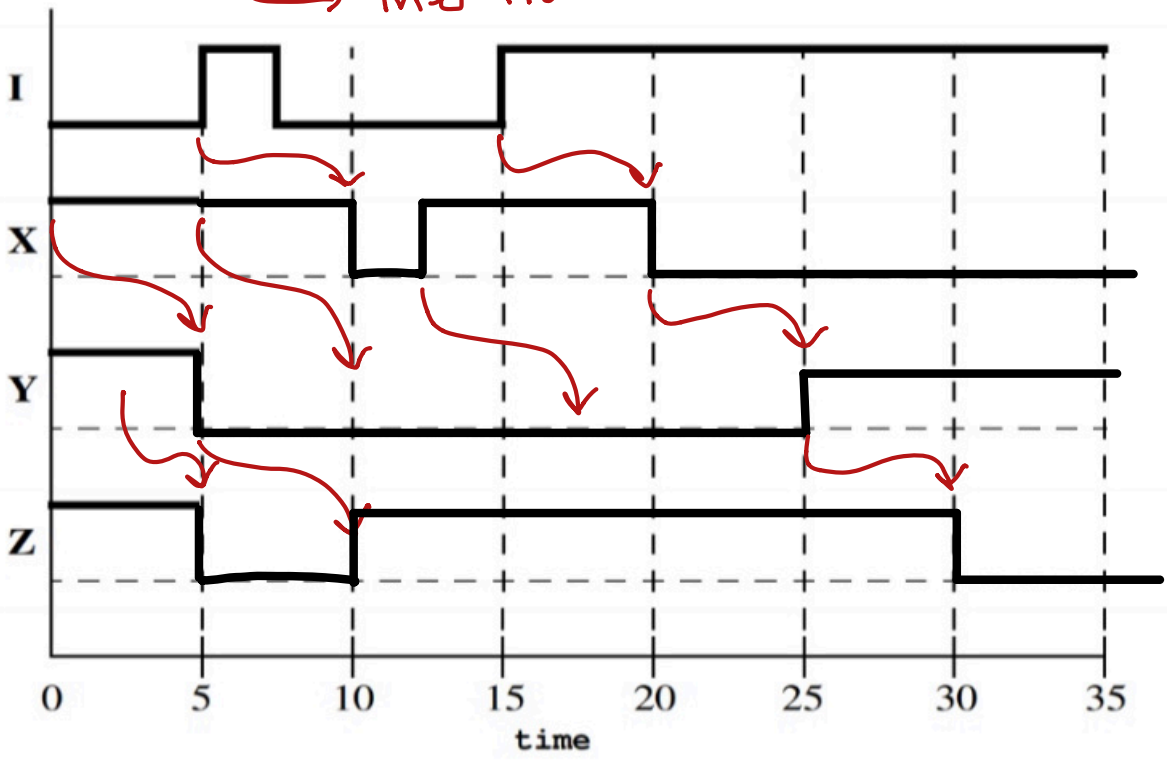
3.

Complete the Timing Diagram for the simulation of the following Verilog Model where input I is driven in the manner that is shown on the diagram.

```
module v_model(input I, output reg X, output Y,Z);
   always @(I) X <= #5 ~I;
   not #(5) G1 (Y,X);
   assign #5 Z = ~Y;
endmodule;
```

transport

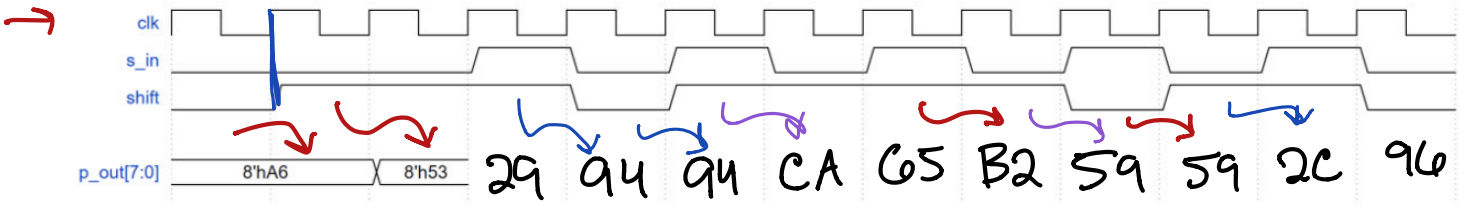
inertial



primitive inst. name (result, operands)

not G0 (Y, X)

1. Complete the following timing diagram of a serial-in/parallel-out shift register, where serial data shifts in bit-by-bit from the **s_in** input into the most-significant bit side (bit 7; newest/most-recent) towards the least-significant bit side (bit 0; oldest/least-recent). Data only shifts when the **shift** input is high; when shift is low **p_out**[7:0] does not change in value. You can either complete the **p_out**[7:0] signal values in hexadecimal or binary.



53 → 0101 0011

0 → 0010 1001 29

1 → 1001 0100 94

1 → 1100 1010 CA

0 → 0110 0101 65

1 → 1011 0010 B2

0 → 0101 1001 59

0 → 0010 1100 2C

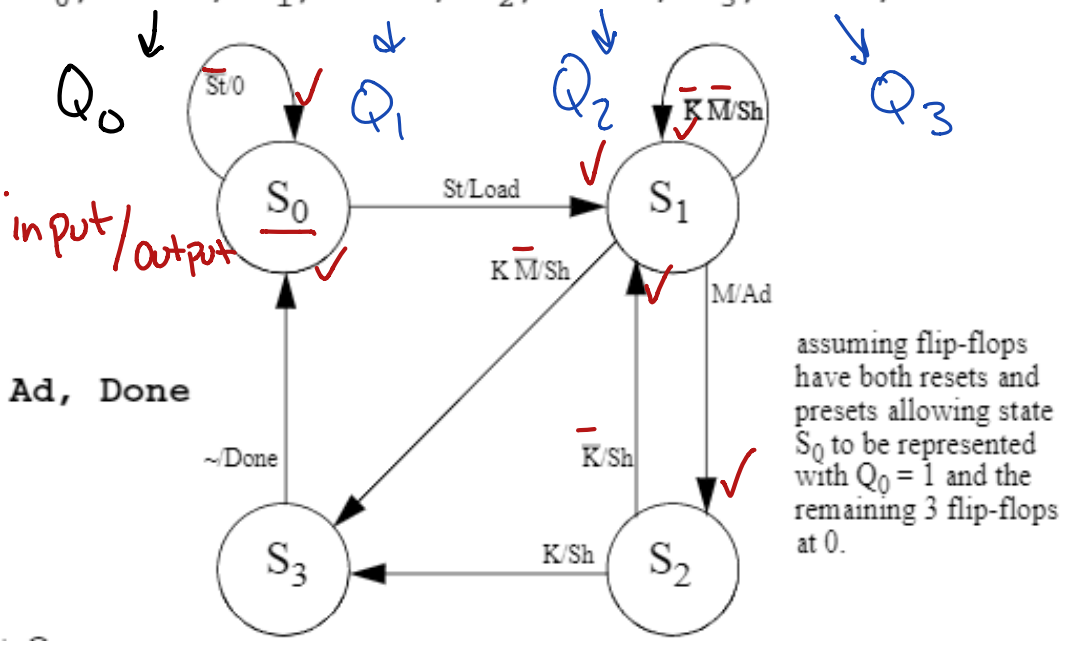
1 → 1001 0110 96

* you read just before the clock edge you care about.

7. [10 points] For the given state graph, derive the simplified next-state and output equations by inspection. Use the following one-hot state assignments for the flip-flops $Q_0Q_1Q_2Q_3$: S_0, 1000; S_1, 0100; S_2, 0010; S_3, 0001;

Inputs:
St,K,M

Outputs:
Load,Sh, Ad, Done



assuming flip-flops have both resets and presets allowing state S_0 to be represented with $Q_0 = 1$ and the remaining 3 flip-flops at 0.

$$Q_0^+ = \overline{St}\,Q_0 + Q_3$$

$$Q_1^+ = \overline{K}\,\overline{M}\,Q_1 + St\,Q_0 + \overline{K}\,Q_2$$

$$Q_2^+ = M Q_1 +$$

$$Q_3^+ = K Q_2 + K\overline{M}\,Q_1$$

$$Load = St\,Q_0$$

$$Ad = M Q_1$$

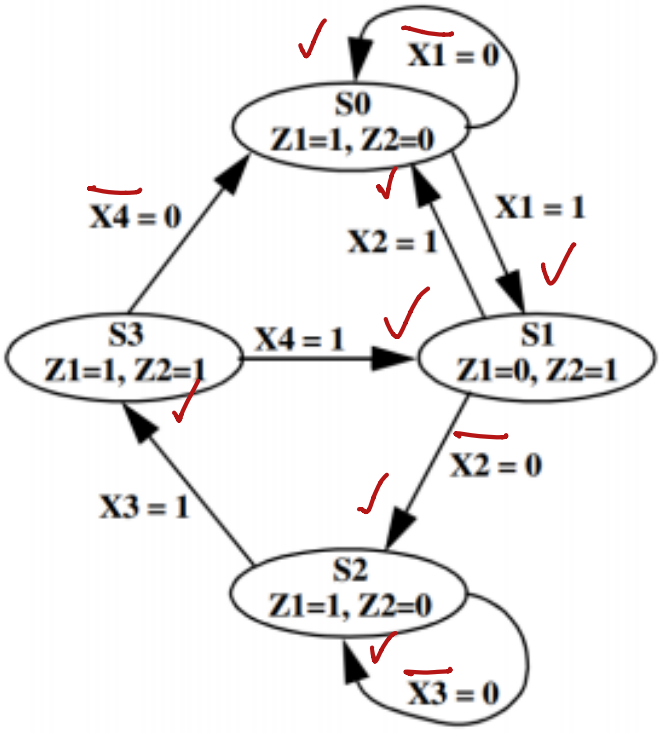
$$Done = Q_3$$

$$Sh = \overline{K}\,\overline{M}\,Q_1 + K\overline{M}\,Q_1 + \overline{K}\,Q_2 + K Q_2$$

5. [20 points] For the given state graph:

Inputs:
X1, X2, X3, X4

Outputs:
Z1, Z2



a. Derive the simplified next-state and output equations by inspection. Use the following one-hot state assignments for the flip-flops $Q_0Q_1Q_2Q_3$:
S0, 1000; S1, 0100; S2, 0010; S3, 0001;

Q_0 Q_1 Q_2 Q_3

$$Q_0^+ = \overline{X1}\,Q_0 + X2\,Q_1 + \overline{X4}\,Q_3$$

$$Q_1^+ = X1\,Q_0 + X4\,Q_3$$

$$Q_2^+ = \overline{X2}\,Q_1 + \overline{X3}\,Q_2$$

$$Q_3^+ = X3\,Q_2$$

$$Z_1 = Q_0 + Q_2 + Q_3 = \overline{Q_1}$$

$$Z_2 = Q_1 + Q_3$$

b. Provide Verilog code to implement the above state graph. Note that the Z1 and Z2 outputs, which depend only on the 4-bit State register, must NOT be delayed by a clock cycle with respect to the current value of State. An additional sheet of paper is provided for your convenience:

```
module prob5_moore (input clk, rst, X1, X2, X3, X4, output Z1, Z2);
reg [3:0] State;

always @ (posedge clk) begin
  if (rst)
      State <= 4'b1000;   // S0
  else begin
      State[0] <= ~X1 Q0 | X2 Q1 | ~X4 Q3;
      State[1] <= X1 Q0 | X4 Q3;
      State[2] <= ~X2 Q1 | ~X3 Q2;
      State[3] <= X3 Q2;
  end
end
  assign Z1 = State[0] | State[2] | State[3];
  assign Z2 = State[1] | State[3];
endmodule
```

4. [20 points] Read the following Verilog code module, and answer the following questions from the perspective of an FPGA synthesis tool:

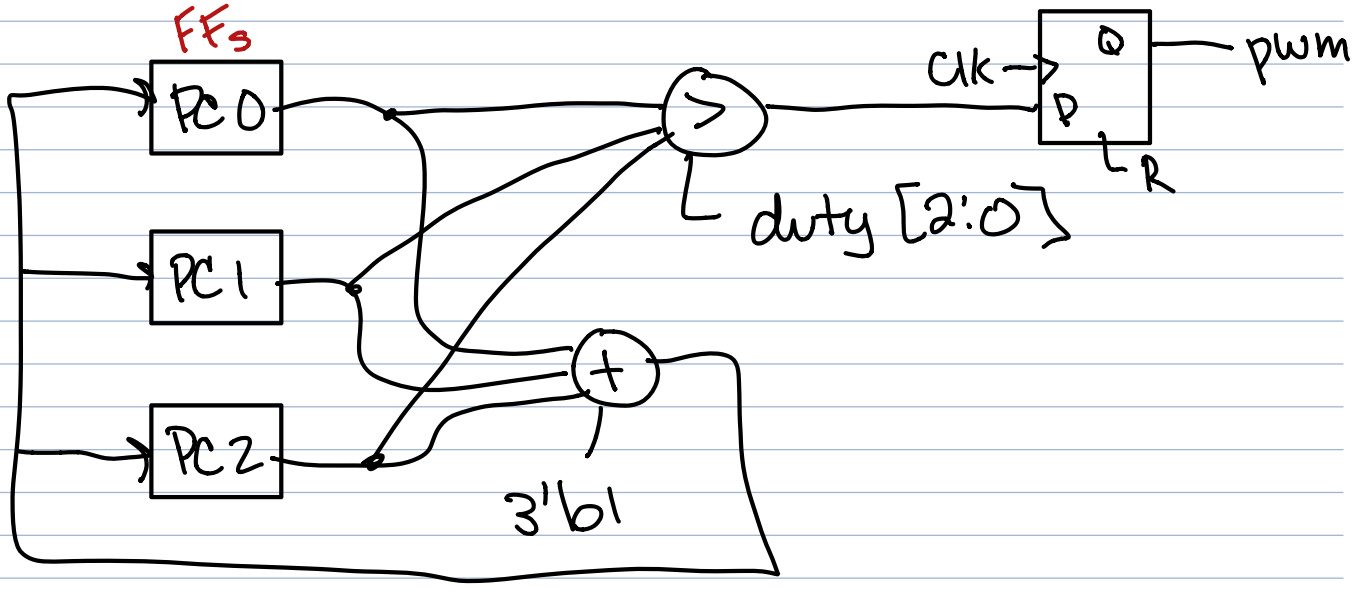
```
module pulse_width_mod (input clk, rst, [2:0] duty, output reg pwm);
  reg [2:0] pulse_cnt;
  always @(clk) begin
    if (rst) begin
      pwm <= 1'b0;
      pulse_cnt <= 3'b000;
    end else begin
      pulse_cnt <= pulse_cnt + 3'd1;
      if (pulse_cnt[2:0] > duty[2:0])
        pwm <= 1'b0;
      else
        pwm <= 1'b1;
    end
  end
endmodule
```

DFF (3) pulse_cnt [2:0]

DFF pwm

a. Assume that all flip-flops are DQ flip-flops with active-high synchronous reset inputs R. Also assume that the (+) adder symbol shown in Problem 3 can be used in your block diagram. Draw a small block diagram of the circuit, including registers, input and output signals.

→ 4 FFs total.



→ 1 LUT for each you are adding.

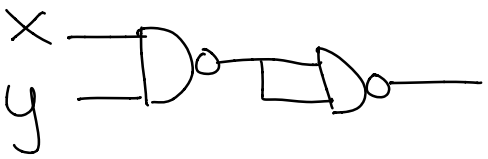
* adding two things that change will need more LUTs to handle carry.

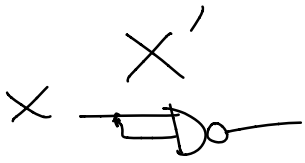
→ 1 LUT for each compared bit

→ 6 LUTs total.

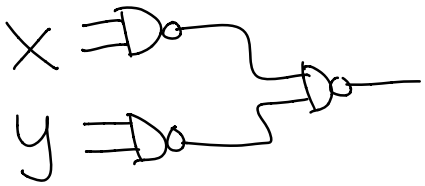
NAND Implementation

product term (AND) for XY



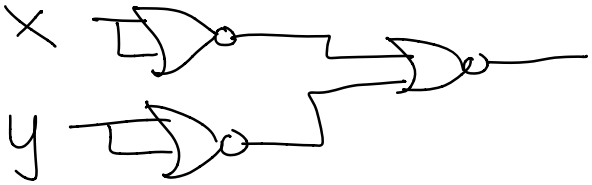


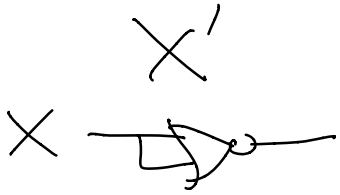
OR for X + y



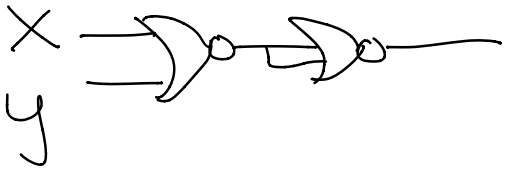
NOR Implementation

product term (AND) for X.y

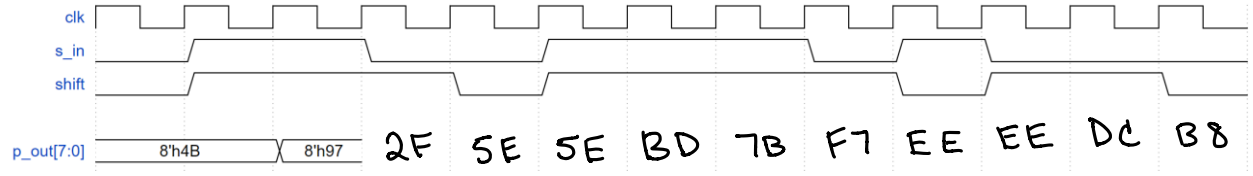




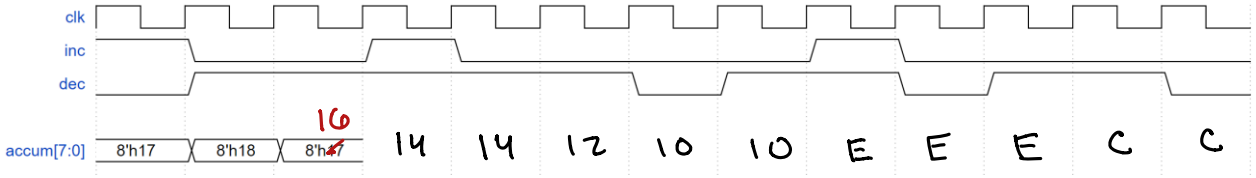
OR for X + y



4. **NOTE: THIS PROBLEM STATEMENT DIFFERS FROM #1 & #2…** Complete the following timing diagram of a serial-in/parallel-out shift register, where serial data shifts in bit-by-bit from the **s_in** input into the ***least-significant*** bit side (bit 0; newest/most-recent) towards the ***most-significant*** bit side (bit ~~0~~ 7; oldest/least-recent). Data only shifts when the **shift** input is high; when shift is low **p_out**[7:0] does not change in value. You can either complete the **p_out**[7:0] signal values in hexadecimal or binary.



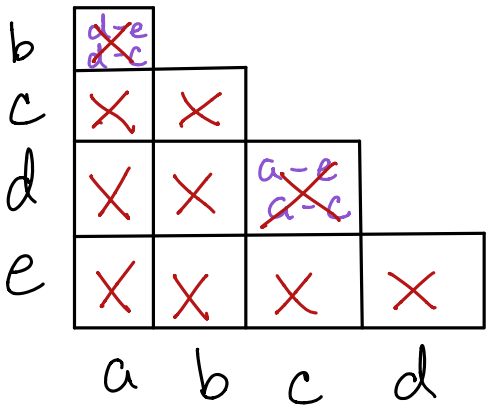
5. **NOTE: THIS PROBLEM DIFFERS FROM PROBLEMS 1-4:**
Complete the following timing diagram of an 8-bit accumulator with 1-bit increment (**inc**) and 1-bit decrement (**dec**) inputs. The output **accum**[7:0] increases by 1 when *only* the **inc** input is high, and decreases by 2 when *only* the **dec** input is high. If neither is high or if both are high, the value of **accum**[7:0] remains the same.



Problem 8 Retry: Reduce the following state tables to the minimum number of states:

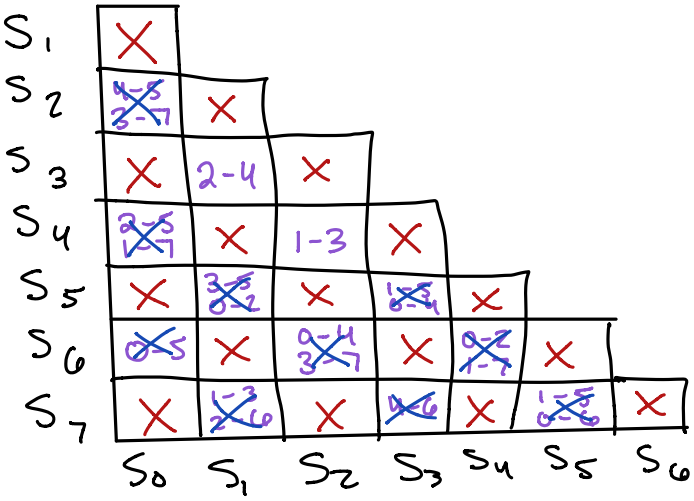
1.

present state	next state X=0	1	present output X=0	1
a	d	f	1	0
b	e	c	1	0
c	a	g	0	1
d	e	c	0	1
e	a	g	1	1
~~f~~	~~c~~	~~c~~	~~0~~	~~1~~
~~g~~	~~d~~	~~f~~	~~1~~	~~0~~



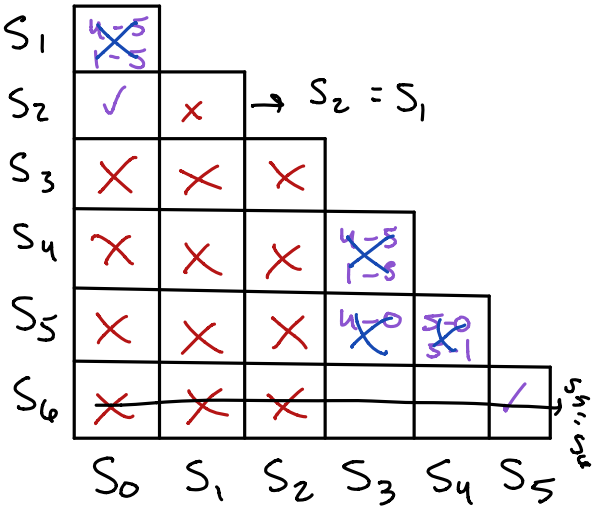
2.

present state	next state X=0	1	present output
S_0	S_5	S_7	0
S_1	~~S_3~~	S_2	1
S_2	~~S_4~~	~~S_3~~	0
~~S_3~~	~~S_1~~	~~S_4~~	~~1~~
~~S_4~~	~~S_2~~	~~S_1~~	~~0~~
S_5	S_5	S_0	1
S_6	S_0	S_7	0
S_7	S_1	S_6	1



3.

Present State	Next State X=0	Next State X=1	Output X=0	Output X=1
S_0	S_4	S_1	1	0
S_1	~~S_5~~	S_5	1	0
~~S_2~~	~~S_4~~	~~S_1~~	~~1~~	~~0~~
S_3	S_4	S_1	0	1
S_4	~~S_5~~	S_5	1	1
S_5	S_0	~~S_1~~	0	1
~~S_6~~	~~S_0~~	~~S_1~~	~~0~~	~~1~~



4.

Current State	Next State X=0	Next State X=1	Output X=0	Output X=1
S_0	S_1	S_2	0	0
S_1	S_0	S_3	1	0
S_2	S_1	S_0	0	1
S_3	S_2	S_0	1	1
~~S_4~~	~~S_1~~	~~S_2~~	~~0~~	~~0~~
~~S_5~~	~~S_4~~	~~S_3~~	~~1~~	~~0~~

no equal outputs. reduced ✓

5.

Current State	Next State X=0	Next State X=1	Output, Z X=0	Output, Z X=1
S_0	S_1	S_2	1	0
S_1	S_0	S_3	0	0
S_2	S_3	S_1	1	1
S_3	S_0	S_1	0	1
~~S_4~~	~~S_0~~	~~S_3~~	~~0~~	~~0~~
~~S_5~~	~~S_1~~	~~S_2~~	~~1~~	~~0~~

no equal outputs. reduced ✓

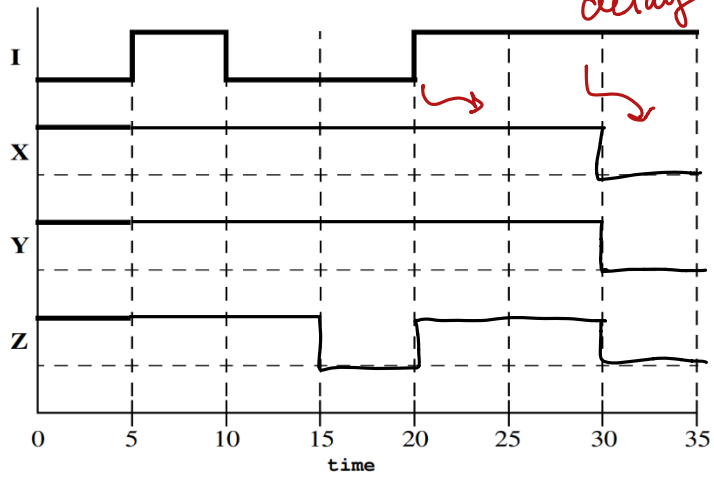
Problem 11 Retry (first part only):

1.

Complete the Timing Diagram for the simulation of the following Verilog Model where input I is driven in the manner that is shown on the diagram.

```
module v_model(input I, output X,Y, output reg Z);
   assign #10 X = ~I;
   not #(10) G1 (Y,I);
   always @(I) Z <= #10 ~I;
endmodule;
```

transport delay



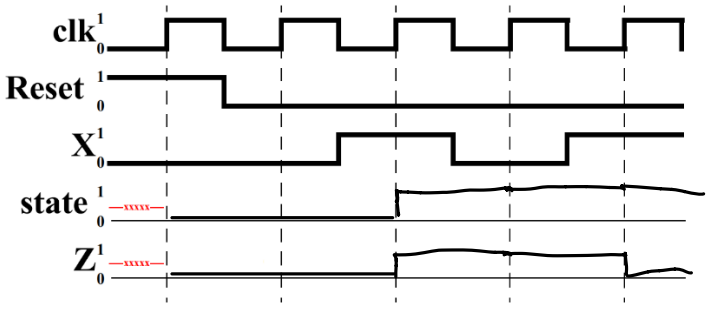
inertial delay ↳ pulse ≥ delay to have effect.

→ previous HW

2.

Complete the following timing diagram for the output signal, **Z** and the internal input signal, **state**. Assume that a basic functional RTL Simulation is to be performed.

```
module fsm_network (input clk, X, Reset, output reg Z);
    reg state, next_state;

    always @ (state, X)
        case (state)
            0 : if (X) begin Z=1; next_state=1; end
                else  begin Z=0; next_state=0; end

            1 : if (X) begin Z=0; next_state=1; end
                else  begin Z=1; next_state=1; end
        endcase

    always @ (posedge clk)
        if (Reset) state=0;
        else state=next_state;

endmodule
```



inertial ↳ in code

transport ↳ delay on wire