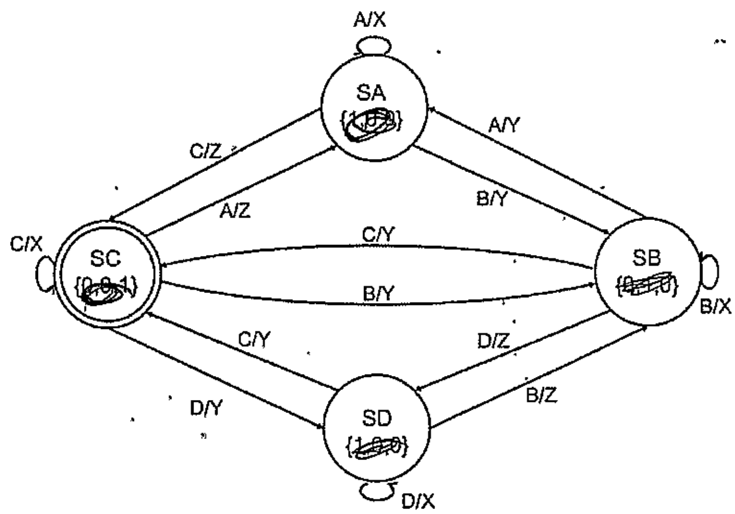
Question #1 State Machine to Logic (30)

A Mealy FSM state diagram is shown below. This is a decoder for a 3-level to 4-level encoding. A 4-level signal is communicated between two endpoints (A, B, C, and D). This signal is the input to the FSM. Transitions between the levels map to 3 symbols (X, Y, and Z); these symbols are the outputs. Note that some transitions are eliminated to enhance the quality of the communication.



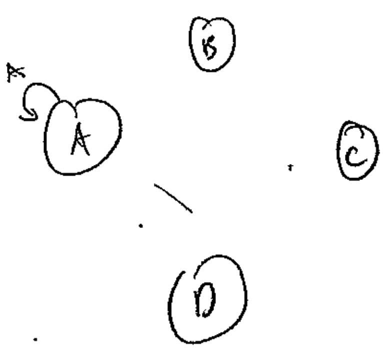
(a) (2) Explain the difference between a Mealy and a Moore FSM.

Mealy FSM: For each state, there could be different outputs depending on the input. Moore states only have 1 output per state.

(b) (4) Fill in the blanks in this partial state transition table.

state	in	out	nx_state
SA	A	X	SA
SB	B	X	SB
SD	B	Z	SB
SC	C	-X	- SC
SD	C	Y	SC
SC	A	Z	SA
SB	D	Z	SD

4 States
3 in
3 out



4 States
3 possible output each state

Assume for the following parts that inputs, outputs and states are all one-hot encoded,

(c) (3) How many bits are needed for the input, output, and states?

States		in		out	
0001	SA	0001	A	001	X
0010	SB	0010	B	010	Y
0100	SC	0100	C	100	Z
1000	SD	1000	D		

bits for in = 4
bits for out = 3
bits for state = 4

(d) (4) What is the logic for nx_state:SA? out:Z? You can define your mapping for part (c) to write this Boolean function.

nx_state:SA = $(SC \wedge A) \vee (SA \wedge A)$ ==> $\boxed{(0100 \wedge 0001) \vee (0001 \wedge 0001)}$

out:Z = $(SD \wedge B) \vee (SC \wedge A) \vee (SB \wedge D)$? $\boxed{(1000 \wedge 0010) \vee (0100 \wedge 0001) \vee (0010 \wedge 1000)}$

(e) (3) If nx_state:SA is written as a *fully-disjunctive* normal form, how many product terms are there?

product terms = 2

Now assume that states, *st[1:0]*, are assigned as gray code: SA=2'b00, SB=2'b01, SC=2'b11, SD=2'b10. The inputs, *in[1:0]*, are also assigned as gray code where A=2'b00, B=2'b01, C=2'b11, D=2'b10; and outputs, *out[1:0]*, are X=00, Y=01, Z=11.

states inputs ; output

(f) (2) How many columns (inputs+outputs) and rows are in this truth table?

columns = 4
rows = 14

Total Different Paths

(g) (5) Use the Karnaugh map below to determine the logic for *out[0]*. How many prime implicants are there? How many are essential?

In[1:0]

st[1:0]

out[0]	"00"	"01"	"11"	"10"
"00"	0	x	x	x
"01"	x	0	x	1
"11"	1	x	0	x
"10"	x	1	0	x

$(st[1] \wedge \overline{in[1]}) \vee$
$(in[1] \wedge \overline{in[0]})$

prime implicants = 3
essential prime implicants = 2

(h) (3) Write the Boolean expression for *out[0]*.

out[0] = $(st[1] \wedge \overline{in[1]}) \vee (in[1] \wedge \overline{in[0]})$

(i) (4) How many states do you need if you want to convert the FSM to a Moore Machine?

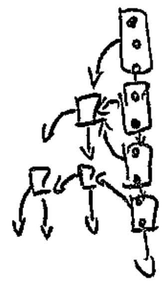
states = 12

4 states * 3 outputs = 12

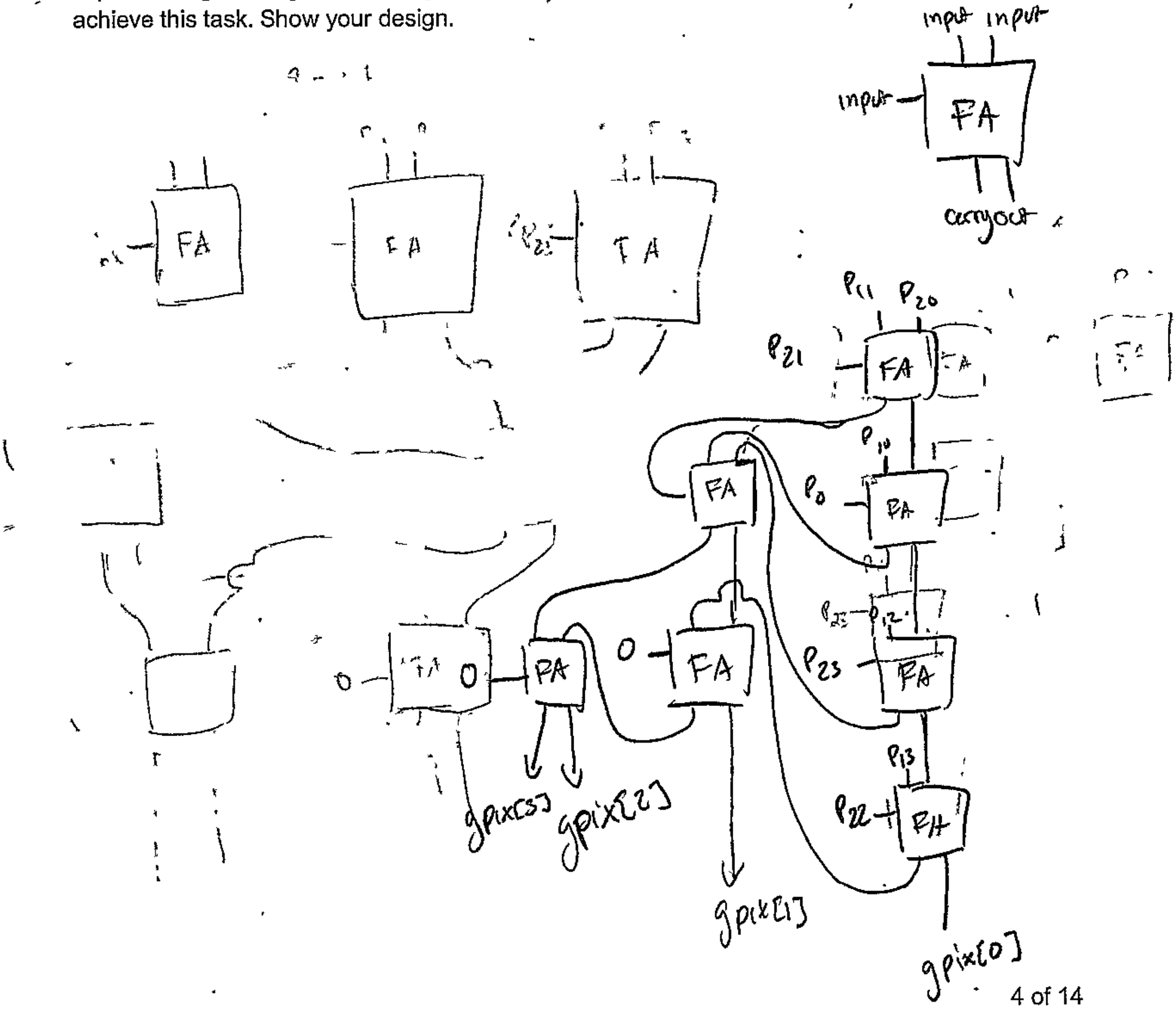
Question #2 Logic Design (15)

A field of black ("0") and white ("1") pixels can be "blurred" into gray values by taking a weighted-average that includes neighboring pixels as shown in the figure. Each converted gray-valued pixel, *gpix*, is a 4-bit value and is computed from 9 binary inputs, p_{xy}, based on the equation $gpix_0[3:0] = 3 \cdot p_0 + 2 \cdot \sum p_{1n} + \sum p_{2n}$.

p_{21}	p_{11}	p_{20}
p_{12}	p_0	p_{10}
p_{22}	p_{13}	p_{23}



You have available 1-bit Full Adders (FA with 3-inputs and 2-outputs) as building blocks for implementing a design. The design should output *gpix*[3:0]. Use the fewest number of FAs to achieve this task. Show your design.

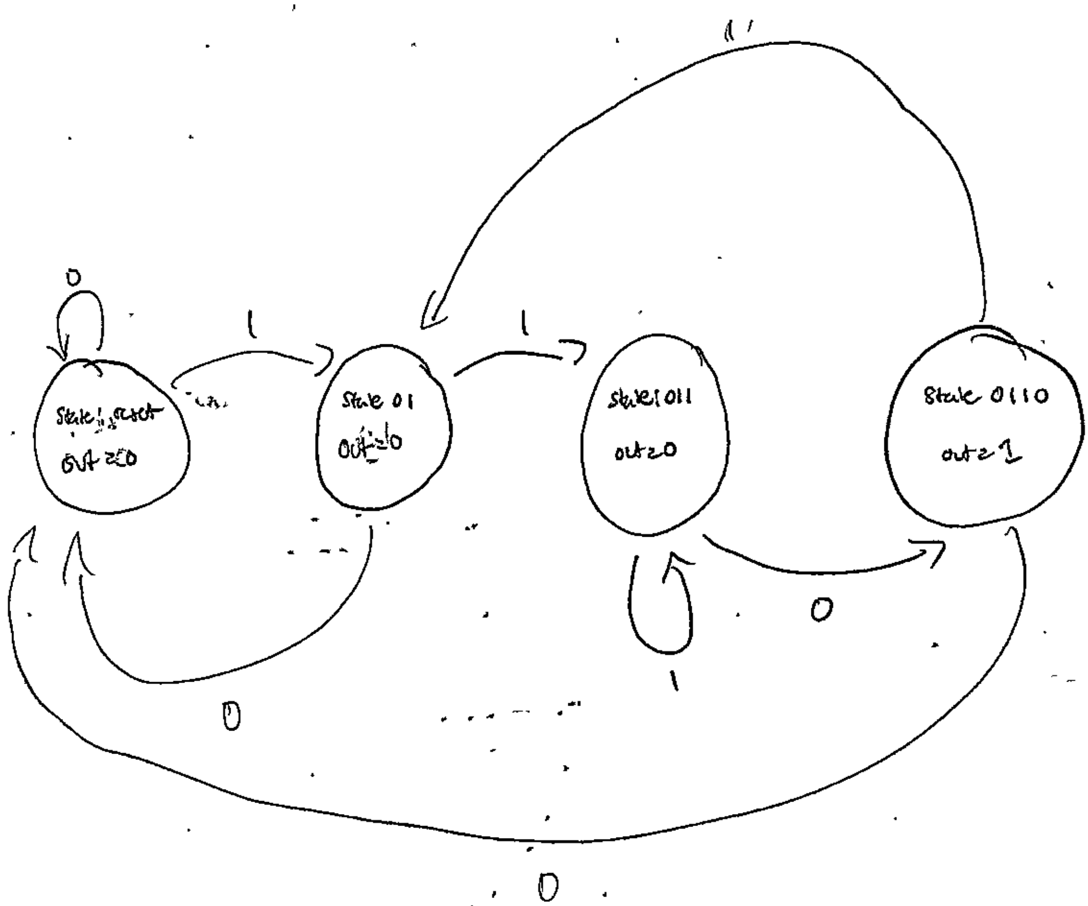


Question #3 FSM State Diagram (15)

The input to an FSM, *Y*, is a string of 1's and 0's. Design a **Moore** FSM that detects when a "01" sequence is followed a "10" sequence. The FSM is reset/initialized to a state for which prior inputs are all 0's. An example of the input and output is shown below and key transitions are underlined. Note that "010" does not constitute a "01" followed by a "10". The output, *Z*, asserts for only 1 cycle when the sequence is detected. As a design constraint, use the fewest number of states.

```
Y = 0000111000100101010101101111
Z = 0000000010000000100000001000
```

~~01010~~ 21



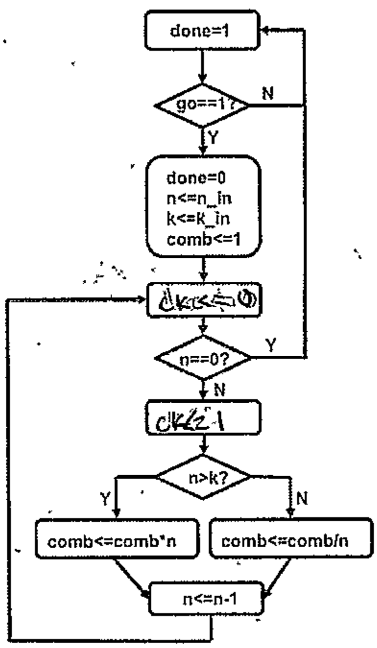
Question #4 System Partitioning (20)

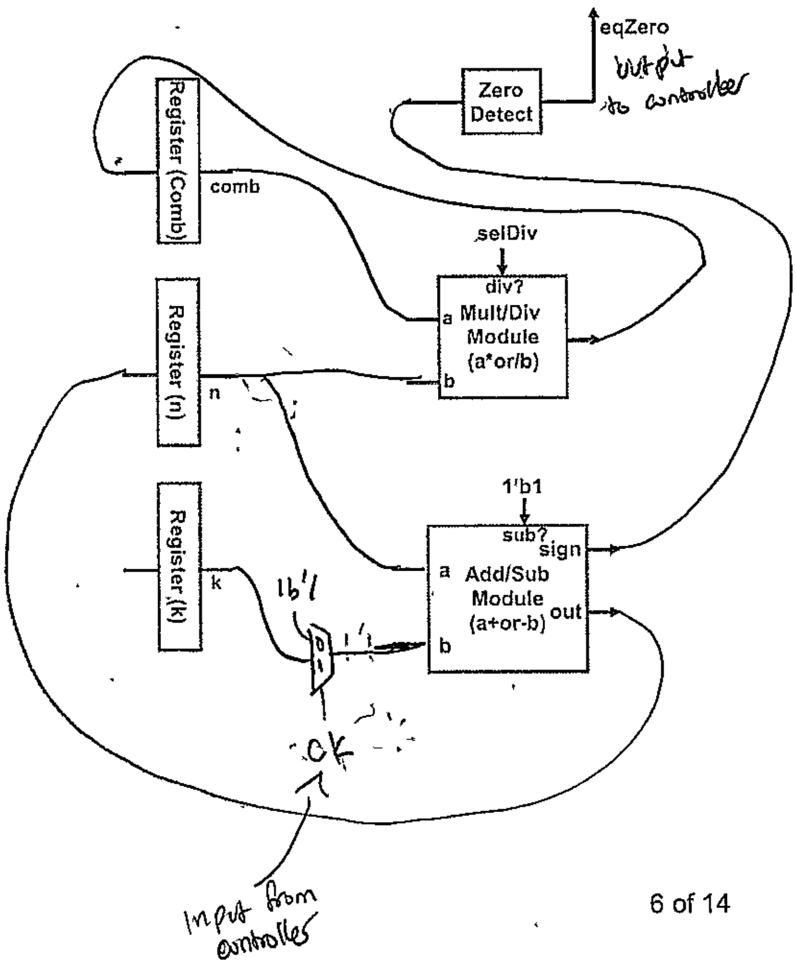
The following algorithm calculates the combinatorics function C(n,k)=n!/(n-k)!k! (commonly referred to as n-choose-k or nCk).

```
n = n_in;
k = k_in;
comb = 1;
while (n > 0) {
    if (n>k)
        comb = comb*n;
    else
        comb = comb/n;
    n--
}
done = 1
```

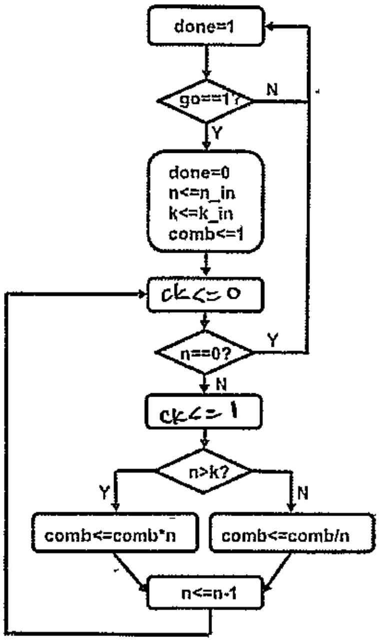
The flow diagram is already designed as shown below. A signal, *go*, is an input to the controller that triggers this algorithm. Output, *done*, is asserted by the controller when the algorithm completes and is waiting for the *go* signal. The previous computation is held in the register, *comb*. You are to complete the controller and datapath design.

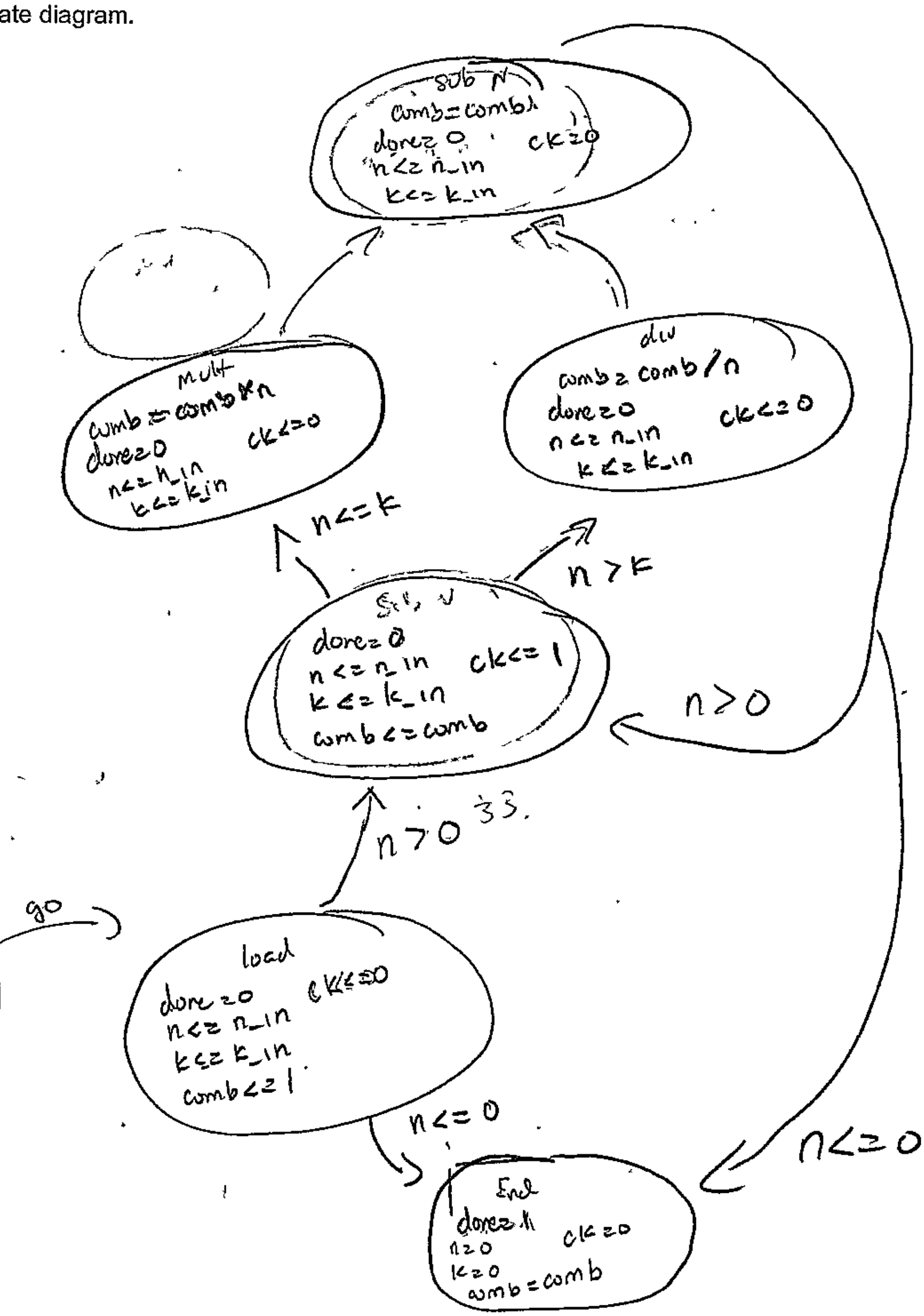
(a) (10) The datapath blocks available to you are also shown below (a combined multiply/divide module, an add/subtract module, and a zero detect module). You may also use as many 2:1 MUX as you choose (Note that you can only use 2:1 MUX so the select signals for each MUX is a single-bit signal from the controller). You can ignore the bit-width of any of the signals. Show the necessary connections within the datapath and any signals that need to pass as inputs to/from the controller.

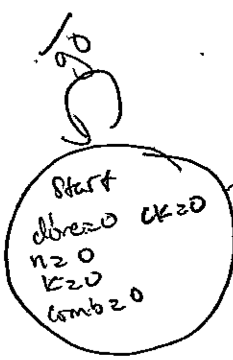




(b) (10) Design a Moore FSM for the controller. Indicate the desired control signals from controller to the datapath on the FSM state diagram.



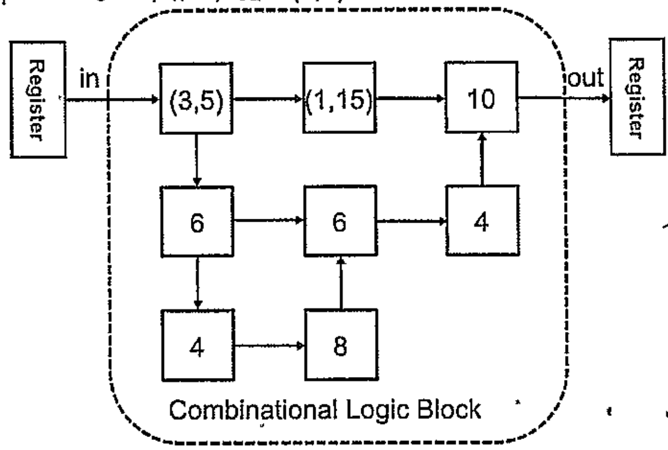




Question #5 Timing and Pipelining (18)

The following combinational logic block can be broken down into modules. Each module have their delay as shown. For each module, the propagation and contamination delay are the same ($t_c = t_d$) with the except of two blocks where the (t_c,t_d) is shown in the block. The registers comprise of DFF with the properties $t_S = 3$, $t_H=1$, $t_{CQ} = (1,2)$.

30



$t_{set} = 3$
$t_h = 1$
$t_{ckQ} = (1,2)$

(a) (4) Determine the contamination and propagation delay of the combinational logic block.

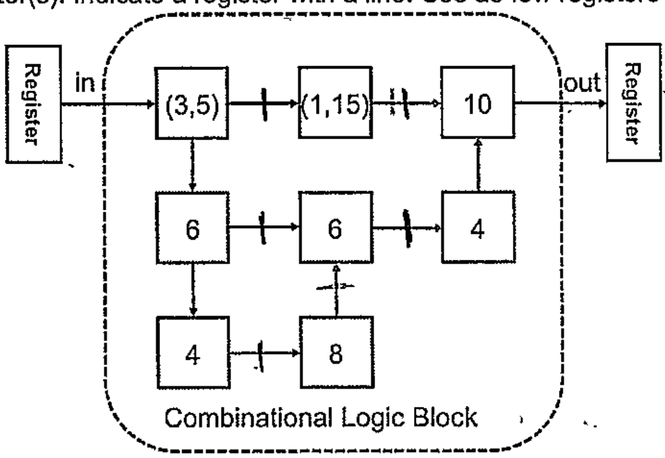
$t_{cCL} =$ 14

$t_{dCL} =$ 43

(b) (3) What is the minimum cycle time of the combinational logic block?

$\min(T_{cycle}) =$ $t_{dclk \to Q} + t_{set} + t_{d\,logic} = 2 + 3 + 43 = 48$

(c) (4) We can minimize the cycle time by inserting registers. Show on the diagram below where to insert the register(s). Indicate a register with a line. Use as few registers as possible.



11
+12
23

(d) (2) Based on the answer in (c) determine the new minimum cycle time.

min(T_{cycle}) = 24

$t_{setup} + t_{d\,clk \to Q} + t_{d\,logic} = 2 + 3 + 15 = 20$

(e) (3) During verification of the design in (d), an engineer found that the DFF hold time is actually longer, t_H=3. Does this pose a problem? Explain your answer.

Yes or No

Explain:

If tH is less than or equal to 15, then this doesn't pose an issue, but if thold is greater than 15, it poses an issue due to ~~violating~~ hold time violation

$t_{hold} <= t_{clk \to Q} + t_{c\,logic}$

$t_{hold} <= 1 + 14 = t_{hold} <= 15$

$3 <= 15$

(f) (2) Name as many ways as you can to fix this problem?

Add skew to decrease the slack of Tcycle time to improve thold's slack / fix it.

Add more registers to decrease minimum cycle time

Increase Contamination logic by adding delays

Increase $t_{clk \to Q}$

Question #6 (12)

An incomplete Verilog code for a module is shown below:

```
module final (
   input [3:0] st,
   output [3:0] nx_st,
   input [1:0] in,
   output [1:0] out,
   input go, reset, done, clock
);

<(a) missing TYPE> [1:0] out;
<(a) missing TYPE> [2:0] nx_st;

always @(<(b) missing activation list>) begin
   case (st)
      3'b000: nx_st=3'b001;
      3'b001: nx_st=3'b010;
      3'b010:
         if (in[0] != go)
              nx_st=3'b100;
           else
              nx_st=3'b010;
        3'b100:
           if (in[0] != go)
              nx_st=3'b100;
           else
              nx_st=3'b001;
        default:
           nx_st = {in[0],1'b0, reset};
    endcase
end

assign out[0] = nx_st[1] | in[1];
//(c) out[1] is the output of a mux that selects 1'b0 when reset else nx_st[0]
endmodule
```

(a) (2) What should be the declared type for the following signals:

reg [1:0] out;
reg [2:0] nx_st;

(b) (2) What should go in the activation list of the always @()? Choose only the signals that needs to be there. You may not use *.
Activation list = posedge clock, go, reset

(c) (5) The signal, *out[1]*, is the output of a 2:1 MUX that uses input, *reset*, to choose between input of 1'b0 (when *reset*==1), and *nx_st[0]* (when *reset*==0). Write the Verilog code for this signal in three different ways (continue next page):

```
// Library module provided
module mux21(muxout, muxselA, inputA, inputB);
   // muxselA ==1 chooses inputA
   // module details not shown
endmodule
```

Declarative Verilog:

module mux21 (out, input A, input B);
assign out = (reset) ? 1'b0 : nx_st[0];
input A ; input B,
end module

Structural Verilog (using the library module above)

mux21 mux (out, reset, 1'b0, nx_st[0]);

Procedural Verilog (note that *out* variable will need to be declared differently):

reg out;

always @ (*) begin
if (reset == 1)
out = 1'b0;
end else begin
out = nx_st[0];
end

(d) (3) Four different ways of implementing a function is shown below. Which of them are the same? Circle all that are the same.

```
(1)
always (@posedge clock) begin
   y <= z;
   x <= y;
end
(2)
always (@posedge clock) begin
   y = z;
   x = y;
end
(3)
always (@posedge clock) begin
   x = y;
   y = z;
end
(4)
always (@posedge clock) begin
   z = y;
   y = x;
end
```

Question #7 Short Answers (15)

(a) (4) For the following Karnaugh map, the Boolean expression for the function

$Z = (\neg A \wedge \neg B \wedge \neg C) \vee (A \wedge \neg B \wedge D) \vee (A \wedge \neg B \wedge C)$

AB

Z	"00"	"01"	"11"	"10"
"00"	1	0	0	0
"01"	1	0	0	1
"11"	0	0	0	1
"10"	0	0	0	1

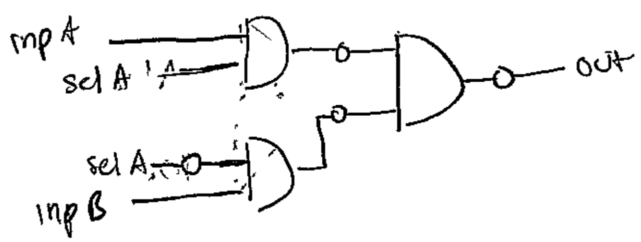
CD

What input conditions and transition has a potential for causing a glitch (static hazard) at the output? Not all prime implicants are covered in the boolean expression, meaning the input $\bar{C} \wedge \bar{D} \wedge \bar{B}$ could have a potential for a glitch.

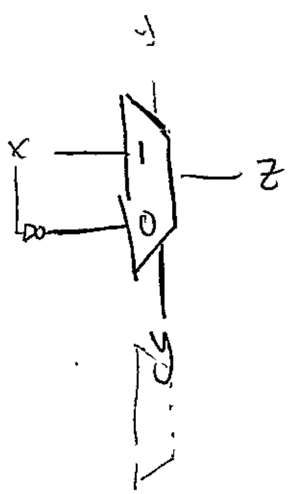
(b) (2) How would you resolve the issue in (a) by adjusting the Boolean expression?

Z = $(\bar{C} \wedge \bar{A} \wedge \bar{B}) \vee (C \wedge A \wedge \bar{B}) \vee (D \wedge A \wedge \bar{B}) \vee (\bar{C} \wedge D \wedge \bar{B})$

(c) (3) If you only have 2-input AND gates and Inverters available, how would you build a 2:1 multiplexer? (*out* selects between *inpA* and *inpB* with the select signal, *selA*)



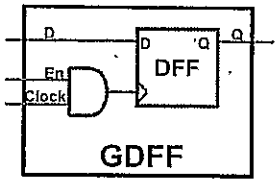
(d) (3) If you only have 2:1 MUX and Inverters available, how would you implement Z = X xor Y?



Y \ X	0	1
0	0	1
1	1	0

X ∧ Y.

(e) (3) A designer modified the basic DFF as shown below to make a GDFF where the clock signal is ANDed with an Enable signal. This approach is known as "clock gating". How does the GDFF's characteristics compare to that of the DFF? Select the answer.



Setup:	$t_{S\_GDFF} > t_{S\_DFF}$	$t_{S\_GDFF} = t_{S\_DFF}$	$t_{S\_GDFF} < t_{S\_DFF}$
Hold:	$t_{H\_GDFF} > t_{H\_DFF}$	$t_{H\_GDFF} = t_{H\_DFF}$	$t_{H\_GDFF} < t_{H\_DFF}$
Clock-Q Delay:	$t_{C2Q\_GDFF} > t_{C2Q\_DFF}$	$t_{C2Q\_GDFF} = t_{C2Q\_DFF}$	$t_{C2Q\_GDFF} < t_{C2Q\_DFF}$

<Extra blank page for work>