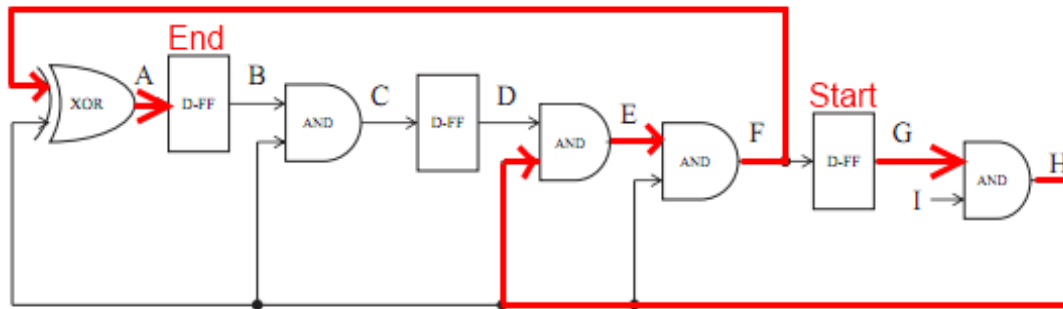


Clock Period/ Delay Analysis:

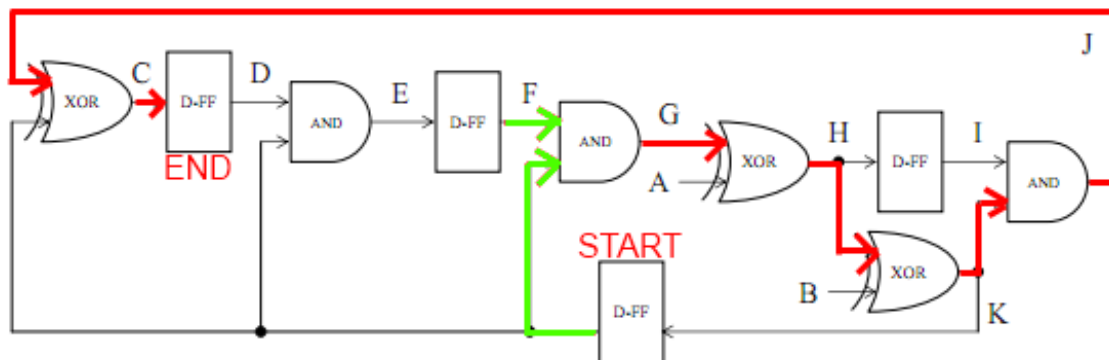
Find longest possible path (time-wise) between two flip-flops.



If 2ns for AND and 3ns for XOR, with $T_{delayFF} = 1\text{ns}$ and $T_{setupFF} = 1\text{ns}$.

So the total time is: 1 setupFF + 3 AND + 1 XOR + 1 delayFF = $(1*1) + (3*2) + (1*3) + (1*1) = 11\text{ ns}$

ex 2:



AND = 2ns, XOR = 3ns, FF setup = 1ns, Ffdelay = 1ns.

Total time is: 1 setup, 2 AND, 3 XOR, 1 delay = $(1*1) + (2*2) + (3*3) + (1*1) = 15\text{ns}$

K-Maps:

Moore Machine:

Consider the following Moore machine with initial state A. ("*" means don't care.)

$S_1 S_0$	$S_1^+ S_0^+$				F
	JK = 00	JK = 01	JK = 10	JK = 11	
A	A	B	C	A	1
B	B	A	A	C	*
C	C	B	B	A	1

Current State	$S_1 S_0$	Next State				F
		$JK = 00$	$JK = 01$	$JK = 10$	$JK = 11$	
A	00	A	B	C	A	1
B	01	B	A	A	C	*
C	10	C	B	B	A	1

Reading the chart (simplified):

- If it is in state A, and I give the input 01, it moves to state B.
- if it is in state B, and I give the input 11, it moves to state C.

In more detail:

State	$S_1 S_0$
A	00
B	11
C	10

- If it is in state A ($S_1 S_0 = 00$), if I give the input $JK = 01$, then $S_1^+ S_0^+ = B$ (11)

How to answer the question (fill in the Kmaps):

JK	S_1^+ Left Digit				JK	S_0^+ Right Digit			
	00	01	11	10		00	01	11	10
$S_1 S_0$					$S_1 S_0$				
A 00					A 00				
* 01					* 01				
B 11					B 11				
C 10					C 10				

Pay Attention, it's not 00, 01, 10, 11 !!!!

Then just use the chart to fill it in. (rows are current state, columns are input.)

JK	S_1^+ Left Digit				JK	S_0^+ Right Digit			
	00	01	11	10		00	01	11	10
$S_1 S_0$					$S_1 S_0$				
A 00	0	1	0	1	A 00	0	1	0	0
* 01	*	*	*	*	* 01	*	*	*	*
B 11	1	0	1	0	B 11	1	0	0	0
C 10	1	1	0	1	C 10	0	1	0	1

The last part:

State	$S_1 S_0$	F
A	00	1
B	11	*
C	10	1

State:

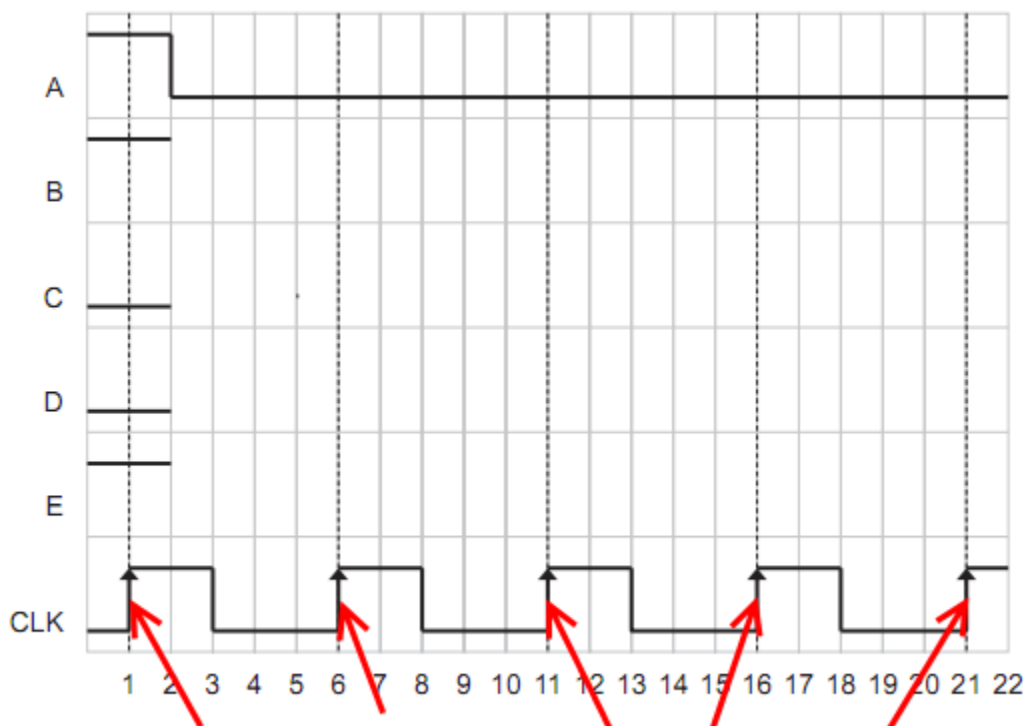
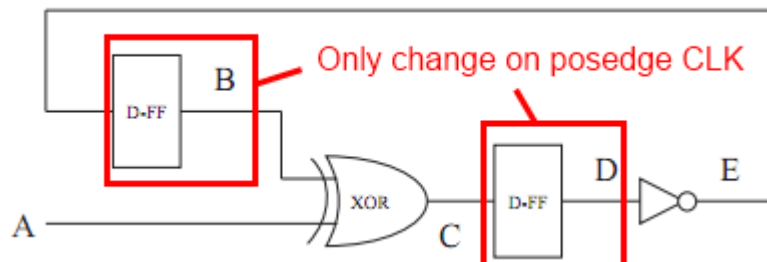
		S_0	
		0	1
S_1	0	A	*
	1	C	B

Output:

		F	
		0	1
S_1	0	1	*
	1	1	0

Timing analysis:

All gates take 1ns, ff delay and setup are 1ns.



Flip Flops (B and D) change 1 ns after these.

You only need to worry about edges which would be caused to change immediately after a given change.

In this example, if A changes, that can ONLY effect C. If C changes, that can ONLY affect D, and so on.

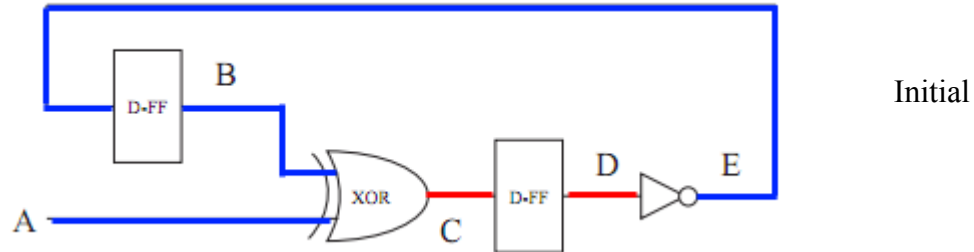
Therefore, D will ONLY change if C changes, and C will ONLY change if A changes. So you need only consider these when a relevant change happens.

EXTREME STEP-BY-STEP BREAKDOWN: (blue = 1, red = 0)

***NOTE: You ONLY have to consider the NEXT wire to change. (A can ONLY change C).

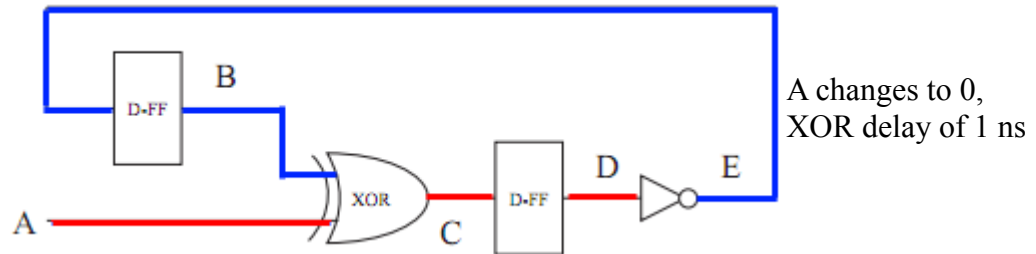
ns 1-2:

Clock: 1



ns 2-3:

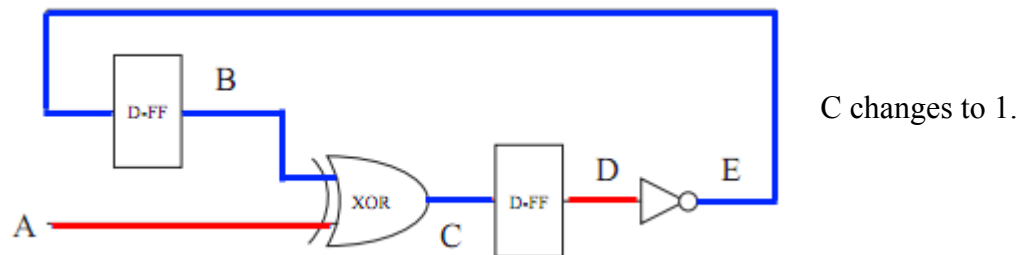
Clock: 1



-----Negedge-----

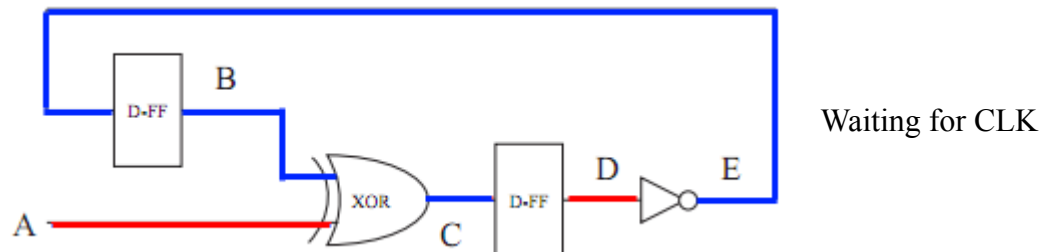
ns 3-4:

Clock: 0



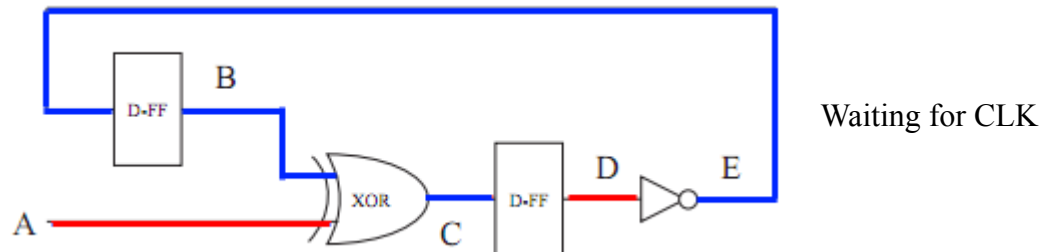
ns 4-5:

Clock: 0



ns 5-6:

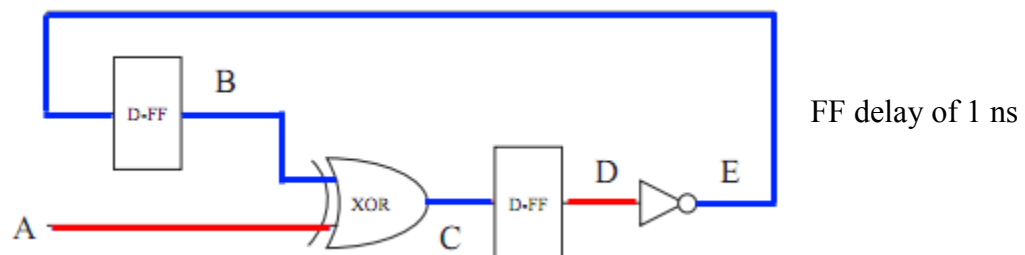
Clock: 0



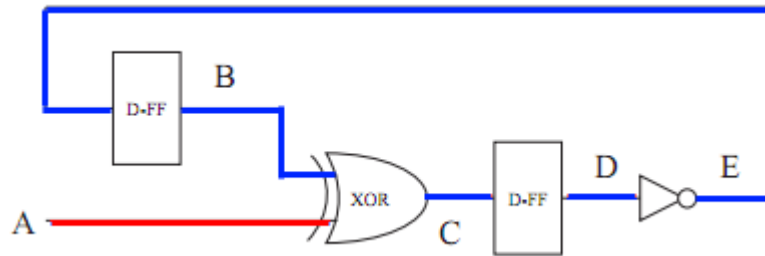
-----Posedge-----

ns 6-7:

Clock: 1



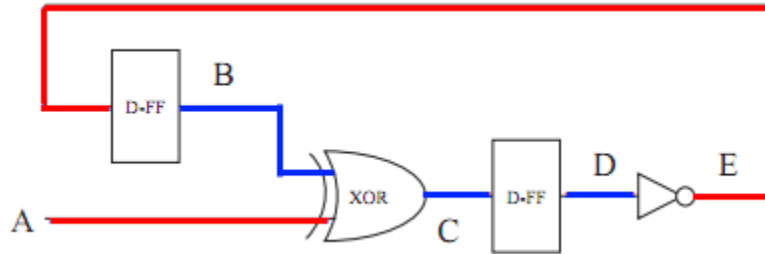
ns 7-8:
Clock: 1



D changes to 1,
INV delay of 1 ns

-----Negedge-----

ns: 8-9
Clock: 0

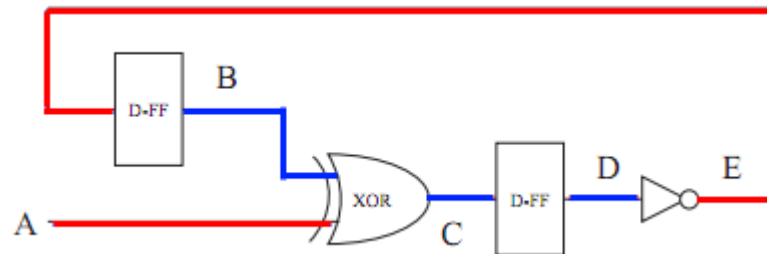


E changes to 0,
waiting for CLK.

.....
.....

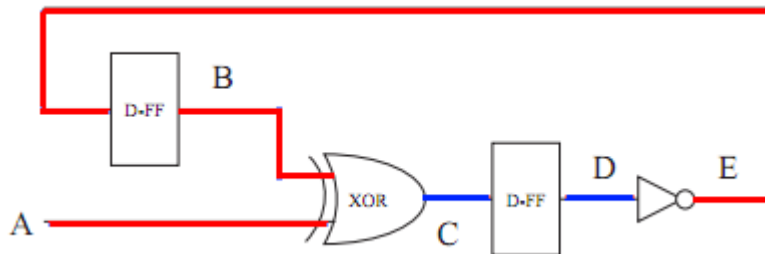
-----Posedge-----

ns: 11-12
Clock: 1



FF delay of 1 ns

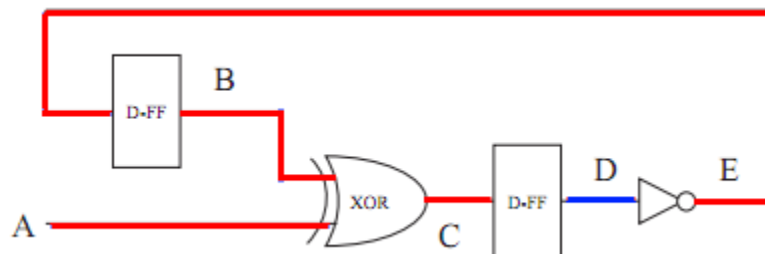
ns: 12-13
Clock: 1



B changes to 0,
XOR delay of 1 ns.

-----Negedge-----

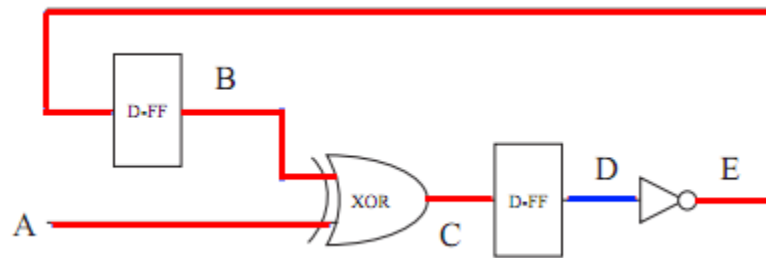
ns: 13-14
Clock: 0



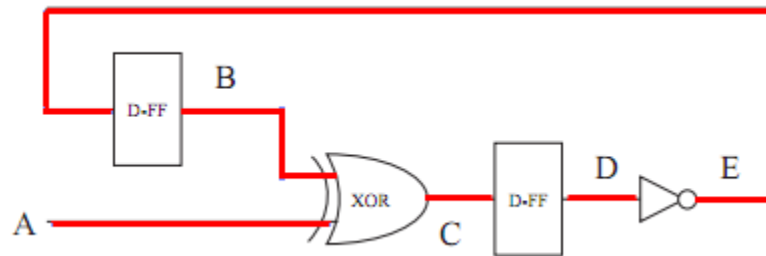
C changes to 0,
waiting for CLK.

.....
.....

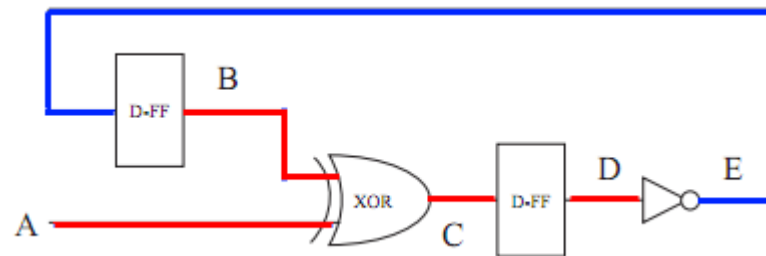
ns: 16-17
Clock: 1



ns: 17-18
Clock: 1

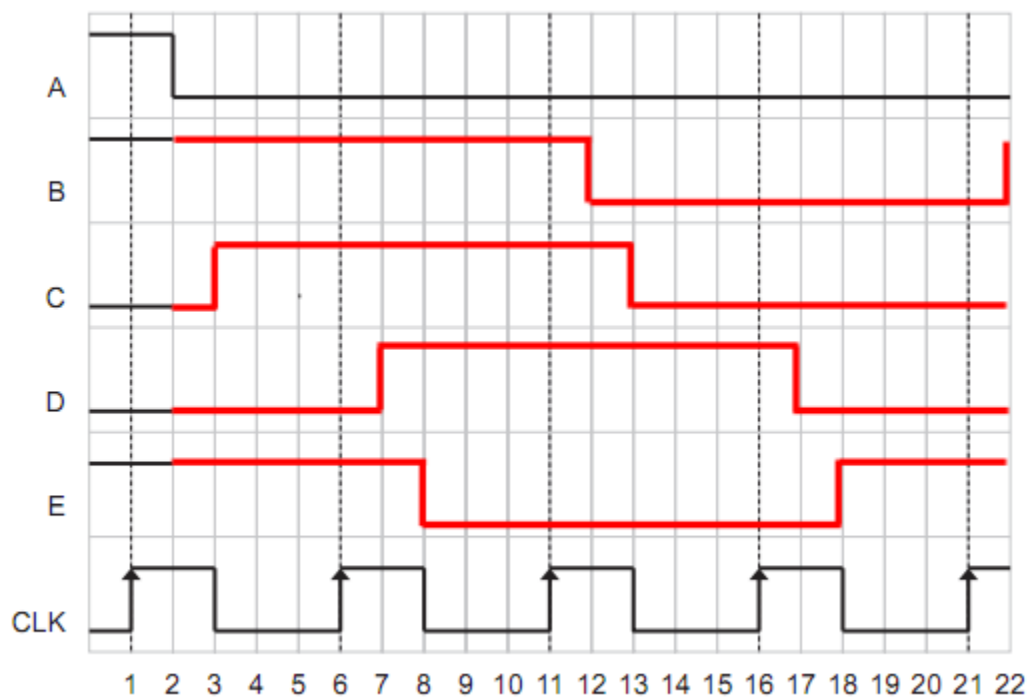


-----Negedge-----

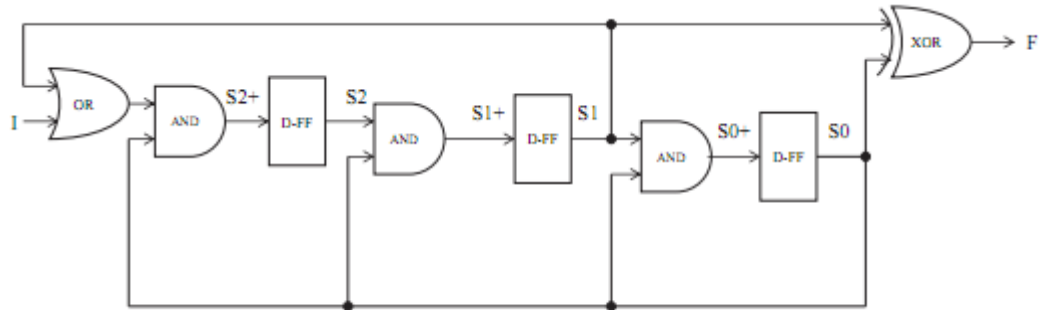


E changes to 1,
waiting for CLK

.....
.....
.....
Table:

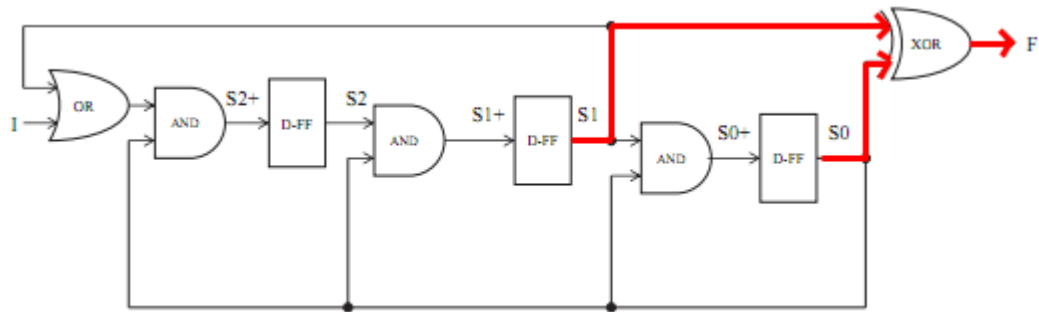


Reverse engineering sequential circuits:



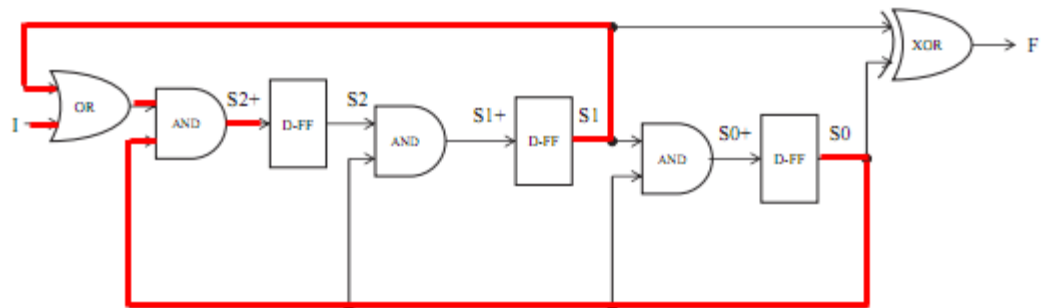
Need to find equations for F , S_2+ , S_1+ , S_0+ .

To Find F :



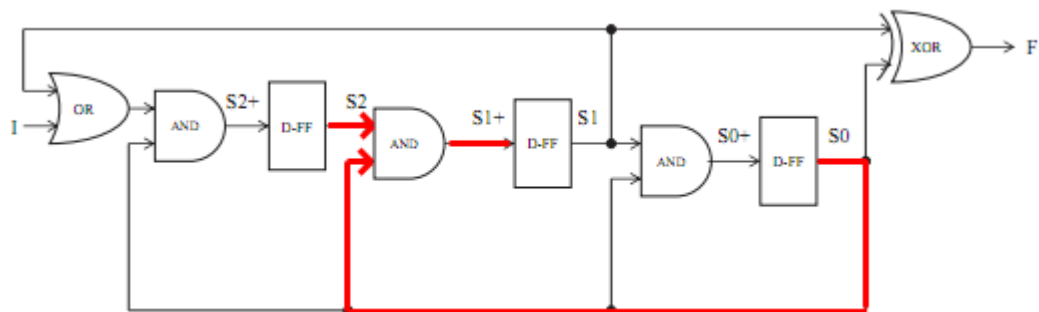
Therefore: $F = S_0 (+) S_1$

To Find S_2+ :



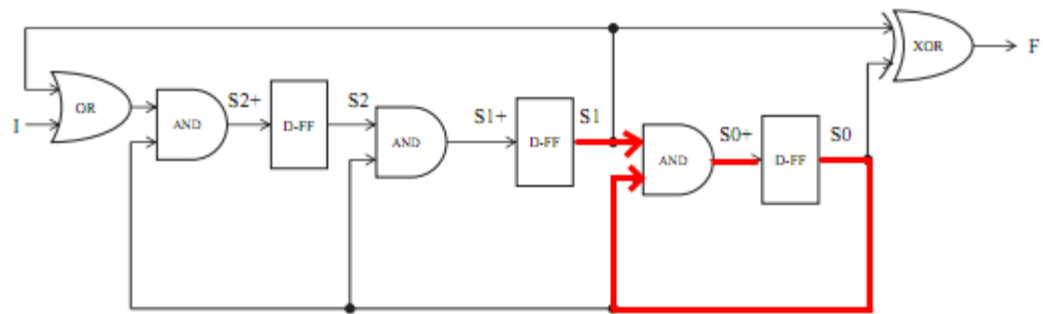
Therefore: $S_2+ = (S_1 + I) * S_0$

To Find S_1+ :



Therefore: $S_1+ = S_2 * S_0$

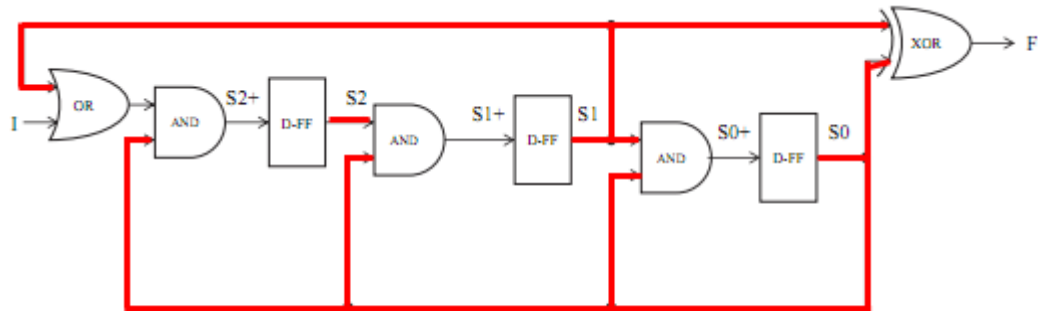
To Find $S0+$:



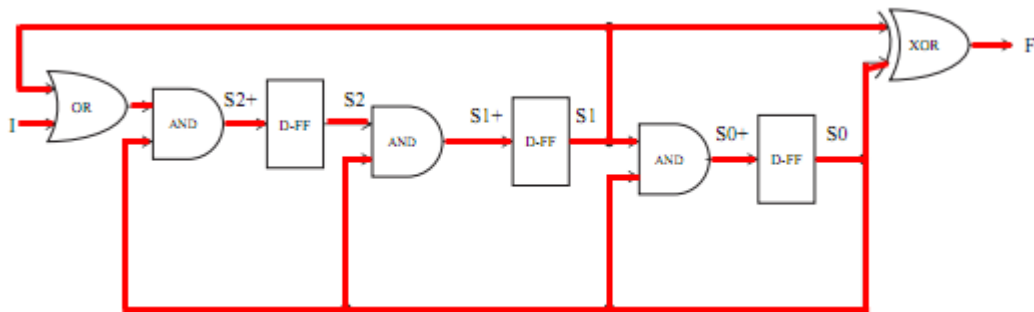
Therefore: $S0+ = S0 * S1$

Now for the State Table:

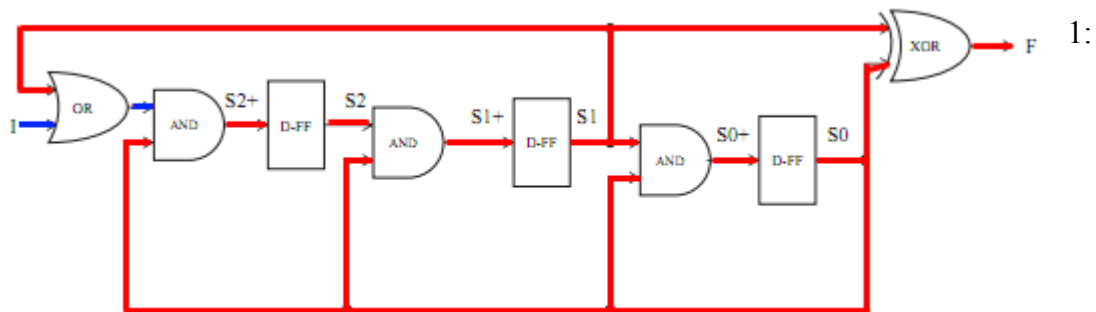
If initial state $S_2S_1S_0 = 000$:



If $I = 0$:



If $I =$

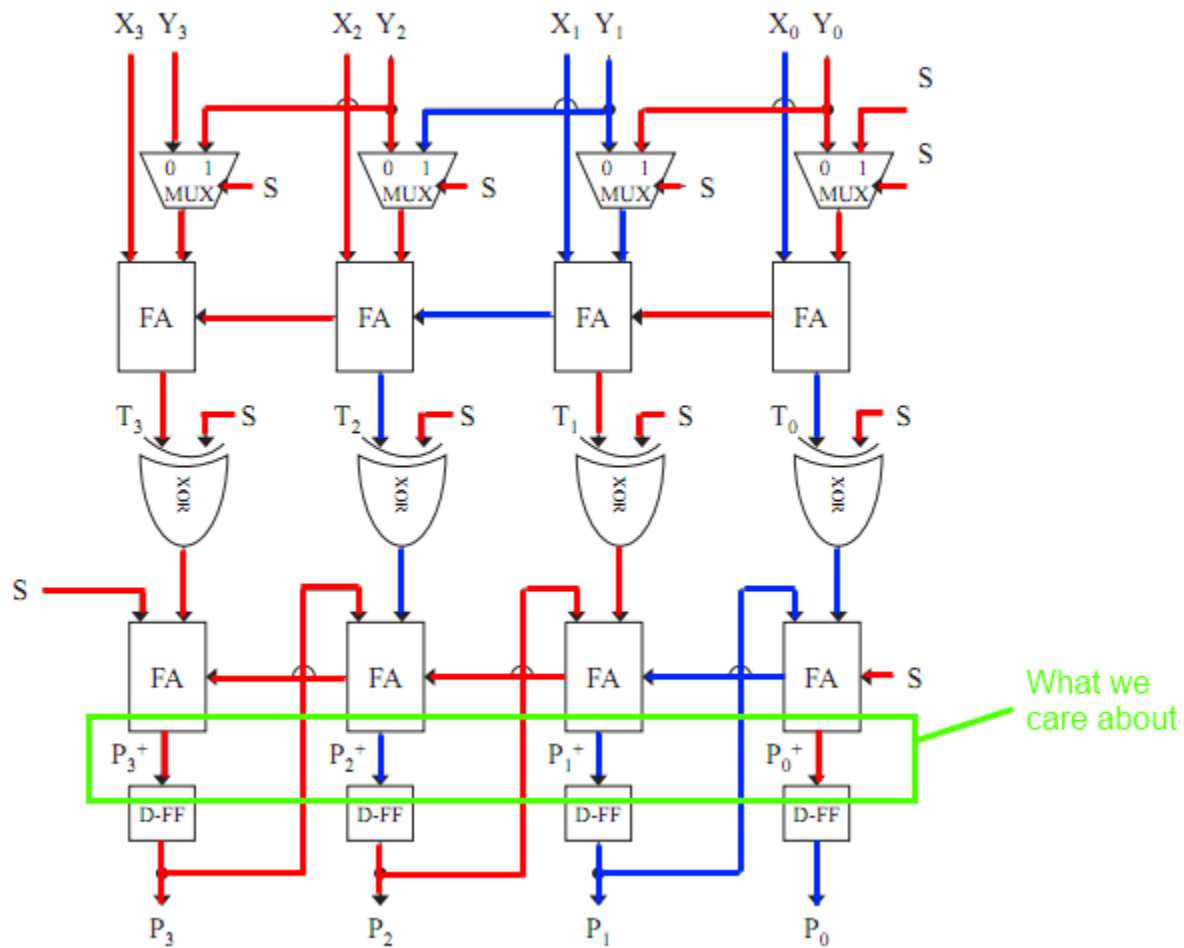


(Too Many AND gates to change State from 000)

DataPaths/Registers: (this is complicated and I'm getting lazy so I'm not going to do the whole thing.

S	$X_3X_2X_1X_0$	$Y_3Y_2Y_1Y_0$	$P_3P_2P_1P_0$	$P_3^+P_2^+P_1^+P_0^+$
0	0011	0010	0011	

With all the givens:

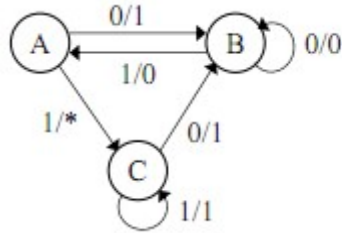


So, the answer is:
 $P_3^+P_2^+P_1^+P_0^+ = 0110$

Just essentially do the same for other inputs.

NOTE: FA is an adder.

Sequential Logic Implementation (Mealy Machine):



State	$S_1 S_0$
A	00
B	10
C	01

$$F =$$

$$S_1^+ =$$

$$S_0^+ =$$

Simplification using a state table:

		I	
		0	1
A	00	B	C
C	01	B	C
-	11	*	*
B	10	B	A

Then use appropriate $S_1 S_0$ conversion: A=00, B=10, C=01 to generate the S_1^+, S_0^+ tables.

Find Minimum Two-Level Logic for F, S_1^+ , S_0^+ .

F =

		I	
		0	1
00	$S_1 S_0$	1	*
01		1	1
11		*	*
10		0	0

S_1^+ :

		I	
		0	1
00	$S_1 S_0$	1	0
01		1	0
11		*	*
10		1	0

S_0^+ :

		I	
		0	1
00	$S_1 S_0$	0	1
01		0	1
11		*	*
10		0	0

F =

		I	
		0	1
00	$S_1 S_0$	1	*
01		1	1
11		*	*
10		0	0

S_1^+ :

		I	
		0	1
00	$S_1 S_0$	1	0
01		1	0
11		*	*
10		1	0

S_0^+ :

		I	
		0	1
00	$S_1 S_0$	0	1
01		0	1
11		*	*
10		0	0

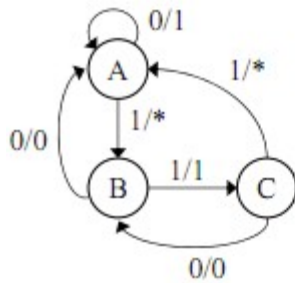
(Red = essential primes, Blue = other Primes.)

$$F = \overline{S_1} \overline{I}$$

$$S_1^+ = \overline{I}$$

$$S_0^+ = \overline{S_1} * I$$

Sequential Logic Example 2: Mealy Machine:



State	$S_2 S_1 S_0$
A	100
B	010
C	001

Note this has a 3-bit state.

You can think of it as having a 4-bit input:

(I, S_2, S_1, S_0)

Therefore you make a 4-bit K-map.

Simplification using a state table:

NOTE: try to organize the state variables so that the states match $\rightarrow I, S_2, S_1, S_0$ means that you don't have to think about the states very much, while I, S_0, S_1, S_2 would require you to flip the states (and leaves more chance for error).

I	S_2	$S_1 S_0$			
		00	01	11	10
0	00				
	01				
	11				
	10				

Now fill it in $\rightarrow$

I	S_2	$S_1 S_0$			
		00	01	11	10
0	00	*	B	*	A
	01	A	*	*	*
	11	B	*	*	*
	10	*	A	*	C

Then break up the filled in table into the appropriate S_2, S_1, S_0 : A=100, B=010, C=001.

And make an additional table for F:

I	S_2	$S_1 S_0$				S_2^+			
		00	01	11	10	00	01	11	10
0	00	x	0	x	1	x	0	x	1
	01	1	x	x	x	1	x	x	x
	11	0	x	x	x	0	x	x	x
	10	x	1	x	0	x	1	x	0

I	S_2	$S_1 S_0$				S_1^+			
		00	01	11	10	00	01	11	10
0	00	x	1	x	0	x	1	x	0
	01	0	x	x	x	0	x	x	x
	11	1	x	x	x	1	x	x	x
	10	x	0	x	0	x	0	x	0

I	S_2	$S_1 S_0$				S_0^+			
		00	01	11	10	00	01	11	10
0	00	x	0	x	0	x	0	x	0
	01	0	x	x	x	0	x	x	x
	11	0	x	x	x	0	x	x	x
	10	x	0	x	1	x	0	x	1

I	S_2	$S_1 S_0$				F			
		00	01	11	10	00	01	11	10
0	00	x	0	x	0	x	0	x	0
	01	1	x	x	x	1	x	x	x
	11	x	x	x	x	x	x	x	x
	10	x	x	x	1	x	x	x	1

State Minimization:

Given:

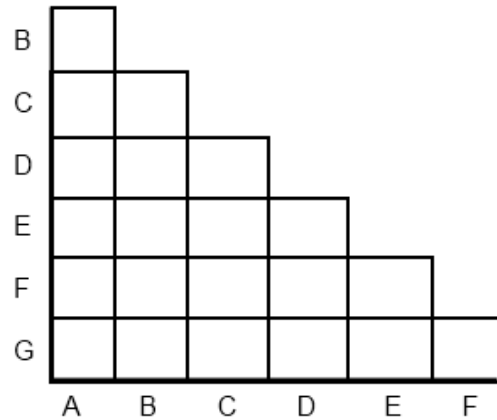
PS	Next State		Out
	X=0	X=1	
A	B	C	0
B	A	C	0
C	D	C	0
D	D	E	1
E	A	F	0
F	B	G	0
G	A	E	0

How to create the table:

1.) Label your horizontal axis using A-F (1, 2, ..., n-1).

Label your vertical axis using B-G (2, 3, ..., n).

2.) Draw the boxes.

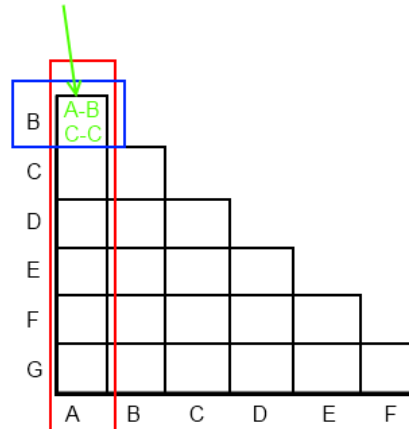


Yes I know,
they aren't the
same size...

3.) Look at the table, and plug in values. (complicated, going to use pictures.)

PS	Next State		Out
	X=0	X=1	
A	B	C	0
B	A	C	0
C	D	C	0
D	D	E	1
E	A	F	0
F	B	G	0
G	A	E	0

Pairs which need to match



** You use the coordinates from the minimization chart (right), then look on the state table (left), make the pairs by input. →

So for the coordinates (A,C), look at rows A and C on the state table.

	x=0	x=1	Out
A	B	C	0
C	D	C	0

Then make the pairs based on the inputs:

x = 0: B-D

x = 1: C-C

Repeat until the chart is full.

Initial:

PS	Next State		Out
	X=0	X=1	
A	B	C	0
B	A	C	0
C	D	C	0
D	D	E	1
E	A	F	0
F	B	G	0
G	A	E	0

B	A-B C-C					
C	B-D C-C	A-D C-C				
D	D-B E-C	D-A E-C	D-D E-C			
E	A-B C-F	A-A C-F	A-D C-F	A-D F-E		
F	B-B C-G	A-B C-G	B-D C-G	B-D G-E	A-B F-G	
G	A-B C-E	A-A C-E	A-D C-E	A-D E-E	A-A E-F	A-B E-G
	A	B	C	D	E	F

Look for states with outputs that do not match (swap the state [A, B, etc] with it's output [0,1], then match):

B	0	0,0					
C	0	0,0	0,0				
D	1	1,0	1,0	1,0			
E	0	0,0	0,0	0,0	0,1		
F	0	0,0	0,0	0,0	0,1	0,0	
G	0	0,0	0,0	0,1	0,0	0,0	
		0	0	0	1	0	0
		A	B	C	D	E	F

Do Not Match

Cross out

squares where

outputs conflict:

B	A-B C-C					
C	B-D C-C	A-D C-C				
D	~~D-B E-C~~	~~D-A E-C~~	~~D-D E-C~~			
E	A-B C-F	A-A C-F	A-D C-F	~~A-B F-F~~		
F	B-B C-G	A-B C-G	B-D C-G	~~B-B C-F~~	A-B F-G	
G	A-B C-E	A-A C-E	A-D C-E	~~A-B E-F~~	A-A E-F	A-B E-G
	A	B	C	D	E	F

Outputs Conflict

D is not compatible with anything else.
(There will always be one [or more]
letters that you can rule out in this first
step).

**** FOR THE FOLLOWING STEPS: Think of “A-B” as “A=B”. So, since we know D does NOT = anything else, in any case where “D-__” the square is crossed out. If the square was “D-D” it would NOT be crossed out.

Since we know D is not compatible, any square with a D that is not D-D is ruled out:

B	A-B C-C					
C	B-D C-C	A-D C-C				
D	~~D-B E-C~~	~~D-A E-C~~	~~D-D E-C~~			
E	A-B C-F	A-A C-F	A-D C-F	~~A-B F-F~~		
F	B-B C-G	A-B C-G	B-D C-G	~~B-B C-F~~	A-B F-G	
G	A-B C-E	A-A C-E	A-D C-E	~~A-B E-F~~	A-A E-F	A-B E-G
	A	B	C	D	E	F

Look for Ds

B	A-B C-C					
C	~~B-D C-C~~	~~A-D C-C~~				
D	~~D-B E-C~~	~~D-A E-C~~	~~D-D E-C~~			
E	A-B C-F	A-A C-F	~~A-D C-F~~	~~A-B F-F~~		
F	B-B C-G	A-B C-G	~~B-D C-G~~	~~B-B C-F~~	A-B F-G	
G	A-B C-E	A-A C-E	~~A-D C-E~~	~~A-B E-F~~	A-A E-F	A-B E-G
	A	B	C	D	E	F

1st Iteration

2nd Iteration

Now (by looking at the columns), we can see that C has been ruled out, so we look for all squares with C-__

B	A-B C-C					
C	~~A-B~~ ~~C-C~~	~~A-A~~ ~~C-C~~				
D	~~A-B~~ ~~C-C~~	~~A-A~~ ~~C-C~~	~~A-B~~ ~~C-C~~			
E	A-B C-F	A-A C-F	~~A-B~~ ~~C-C~~	~~A-B~~ ~~C-C~~		
F	B-B C-G	A-B C-G	~~A-B~~ ~~C-C~~	~~A-B~~ ~~C-C~~	A-B F-G	
G	A-B C-E	A-A C-E	~~A-B~~ ~~C-C~~	~~A-B~~ ~~C-C~~	A-A E-F	A-B E-G
	A	B	C	D	E	F

1st Iteration
2nd Iteration
Look for Cs

B	A-B C-C					
C	~~A-B~~ ~~C-C~~	~~A-A~~ ~~C-C~~				
D	~~A-B~~ ~~C-C~~	~~A-A~~ ~~C-C~~	~~A-B~~ ~~C-C~~			
E	~~A-B~~ ~~C-F~~	~~A-A~~ ~~C-F~~	~~A-B~~ ~~C-C~~	~~A-B~~ ~~C-C~~		
F	~~B-B~~ ~~C-G~~	~~A-B~~ ~~C-G~~	~~A-B~~ ~~C-C~~	~~A-B~~ ~~C-C~~	A-B F-G	
G	~~A-B~~ ~~C-E~~	~~A-A~~ ~~C-E~~	~~A-B~~ ~~C-C~~	~~A-B~~ ~~C-C~~	A-A E-F	A-B E-G
	A	B	C	D	E	F

1st Iteration
2nd Iteration
3rd Iteration

Clarification:

B	A-B C-C					
C	~~A-B~~ ~~C-C~~	~~A-A~~ ~~C-C~~				
D	~~A-B~~ ~~C-C~~	~~A-A~~ ~~C-C~~	~~A-B~~ ~~C-C~~			
E	A-B C-E	A-A C-E	~~A-B~~ ~~C-C~~	~~A-B~~ ~~C-C~~		
F	B-B C-G	A-B C-G	~~A-B~~ ~~C-C~~	~~A-B~~ ~~C-C~~	A-B F-G	
G	A-B C-E	A-A C-E	~~A-B~~ ~~C-C~~	~~A-B~~ ~~C-C~~	A-A E-F	A-B E-G
	A	B	C	D	E	F

Hey! That has a C!!
C was ruled out!!
You're cheating!

Treat _ _ as _ = _
So C=C is okay,
because it is NOT
comparing C to
something that is
not C

Now we look at the rows/columns:

D was ruled out in iteration 1,
C was ruled out in iteration 2,

A, B, E, F and G cannot be ruled out (they all have squares which are valid).

Now, examining those squares (top left)

A-B C-C	→ gives A = B and C = C.
A-B F-G	→ gives A = B, and F = G.
A-A E-F	→ gives A = A, and E = F.
A-B E-G	→ gives A = B, and E = G.

***If coordinates do not match the inputs in the boxes cross out the box.

So, removing duplicates: A = B, E = F = G.

**** Note: You should also look at the coordinates of non-crossed out boxes: as they show equivalence as well. (B, A), (E, F), (E, G), (F, G) means: B = A, E = F = G (we just didn't have to do that in this example)

The state table can then be reduced accordingly: (Replace B with A, replace F and G with E).

PS	Next State		Out
	X=0	X=1	
A	~~B~~ A	C	0
~~B~~	~~A~~	~~C~~	~~0~~
C	D	C	0
D	D	E	1
E	A	~~F~~ E	0
~~F~~	~~B~~	~~G~~	~~0~~
~~G~~	~~A~~	~~E~~	~~0~~

PS	Next State		Out
	X=0	X=1	
A	A	C	0
C	D	C	0
D	D	E	1
E	A	E	0

