HW #4

Q1

(i) State table :

Input	Present State	Next State	Output
0	S1	S4	0
1	S1	S5	1
0	S2	S2	0
1	S2	S1	1
0	S3	S3	0
1	S3	S4	0
0	S4	S6	0
1	S4	S5	1
0	S5	S6	1
1	S5	S3	0
0	S6	S1	0
1	S6	S5	1

$\Pi_1 = \{ (S1, S2, S4, S6), (S3), (S5) \}$

$\Pi_2 = \{ (S1, S4, S6), (S2), (S3), (S5) \}$

$\Pi_3 = \{ (S1, S4, S6), (S2), (S3), (S5) \}$

Thus, states S1, S4 and S6 are equivalent states. The minimized state table is :

input	Present State	Next State	Output
0	S146	S146	0
1	S146	S5	1
0	S2	S2	0
1	S2	S146	1
0	S3	S3	0
1	S3	S146	0
0	S5	S146	1
1	S5	S3	0

(ii) Replace don't-care conditions by 1's

State table:

Input	Present State	Next State	Output
0	S1	S4	0
1	S1	S5	1
0	S2	S2	0
1	S2	S1	1
0	S3	S3	0
1	S3	S4	1
0	S4	S6	0
1	S4	S5	1
0	S5	S6	1
1	S5	S3	0
0	S6	S1	1
1	S6	S5	1

$\Pi_1 = \{ (S1, S2, S3, S4), (S5), (S6) \}$

$\Pi_2 = \{ (S1), (S2, S3), (S4), (S5), (S6) \}$

$\Pi_3 = \{ (S1), (S2), (S3), (S4), (S5), (S6) \}$

Thus, there are no equivalent states and the state table can not be minimized.

(iii) In completely-specified FSM:

Compatible pairs relations

(S1, S2) ⇐ (S2, S4) and (S1, S5)
(S1, S3) ⇐ (S3, S4) and (S4, S5)
(S1, S4) ⇐ (S4, S6)
(S1, S6) ⇐ (S1, S4)
(S2, S3) ⇐ (S1, S4)
(S2, S4) ⇐ (S2, S6) and (S1, S5)
(S2, S6) ⇐ (S1, S2) and (S1, S5)
(S3, S4) ⇐ (S3, S6) and (S4, S5)
(S3, S6) ⇐ (S3, S1) and (S4, S5)
(S4, S6) ⇐ (S1, S6)

incompatible states

(S1, S5), (S2, S5), (S3, S5), (S4, S5), (S5, S6)

Due to the incompatible state pairs, we get the following compatible state pairs relations

(S1, S4) ⇐ (S4, S6)
(S1, S6) ⇐ (S1, S4)
(S2, S3) ⇐ (S1, S4)
(S4, S6) ⇐ (S1, S6)

From this, we get the following maximal compatible classes!

(S1, S4, S6)
(S2, S3) ⇐ (S1, S4) .

Thus, a minimum cover is :

{ (S1, S4, S6), (S2, S3), (S5) }

The minimized state table is

Input	Present State	Next State	output
0	S146	S146	0
1	S146	S5	1
0	S23	S23	0
1	S23	S146	1
0	S5	S146	1
1	S5	S23	0

Q2 Original state table :

input	Present State	Next State	Output
0	S1	S3	1
1	S1	S5	*
0	S2	S3	*
1	S2	S5	1
0	S3	S2	0
1	S3	S1	1
0	S4	S4	0
1	S4	S5	1
0	S5	S1	0
1	S5	S4	1

(i) Minimization based on implicant merging

input	Present State	Next state	output
0	S1, S2	S3	1
1	S1, S2, S4	S5	1
0	S3	S2	0
1	S3	S1	1
0	S4	S4	0
0	S5	S1	0
1	S5	S4	1

we can remove the third row if we add the covering constraint that S2 is covered by all other states. Also, by adding the covering constraint that S1 covers S4, we get the following minimized table :

input	Present State	Next state	output
0	S1, S2	S3	1
1	S1, S2, S4	S5	1
1	S3	S1	1
0	S4, S5	S4	0
0	S5	S1	0
1	S5	S4	1

However, this will prevent us from eliminating the last row by disjunctive relations. So, we will not use this constraint.

By making the disjunctive relation that $S4 = S1$ OR $S5$, the table can be minimzed by eliminating the last row as follows:

Input	Present State	Next State	output
0	S1, S2	S3	1
1	S1, S2, S4, S5	S5	1
1	S3, S5	S1	1
0	S4	S4	0
0	S5	S1	0

Input constraint matrix:

$$A = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$

Covering constraint matrix:

$$B = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Disjunctive constraint matrix:

$$C = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

(ii) Seed dichotomies:

S1a : { (S1, S2) , (S3) }
S1b : { (S3) , (S1, S2) }
S2a : { (S1, S2) , (S4) }
S2b : { (S4) , (S1, S2) }
S3a : { (S1, S2) , (S5) }
S3b : { (S5) , (S1, S2) }
S4a : { (S1, S2, S4, S5) , (S3) }
S4b : { (S3) , (S1, S2, S4, S5) }
S5a : { (S3, S5) , (S1) }
S5b : { (S1) , (S3, S5) }
S6a : { (S3, S5) , (S2) }
S6b : { (S2) , (S3, S5) }
S7a : { (S3, S5) , (S4) }
S7b : { (S4) , (S3, S5) }

We note that the following seed dichotomies do not satisfy the covering relations:

S1a, S2a, S3a, S4a, S6b

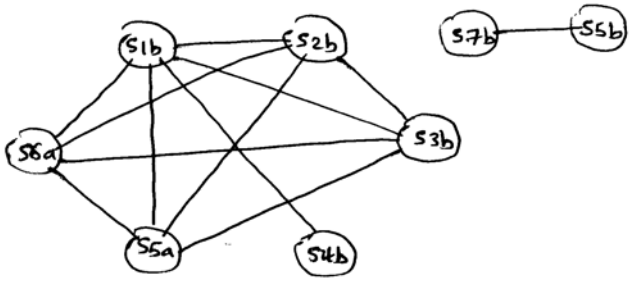
Also, the following seed dichotomies violate the disjunctive constraints:

S2a and S7a.

Thus, the seed dichotomies consistent with the constraints are:

S1b : { (S3) , (S1, S2) }
S2b : { (S4) , (S1, S2) }
S3b : { (S5) , (S1, S2) }
S4b : { (S3) , (S1, S2, S4, S5) }
S5a : { (S3, S5) , (S1) }
S5b : { (S1) , (S3, S5) }
S6a : { (S3, S5) , (S2) }
S7b : { (S4) , (S3, S5) }

Then, we form the compatibility graph:



Note that the seeds S2b and S7b are not compatible because they violate the disjunctive constraint. Merging the two seeds yields the dichotomy { (S4), (S1, S2, S3, S5) } which violates the disjunctive constraint S4 = S1 OR S5.

The seeds S1b, S2b, S3b, S5a and S6a form a clique and we get the prime dichotomy

p1 : { (S3, S4, S5), (S1, S2) }

The seeds S1b and S4b form a clique and we get the prime dichotomy

p2 : { (S3), (S1, S2, S4, S5) }

The seeds S7b and S5b form a clique and we get the prime dichotomy

p3 : { (S1, S4), (S3, S5) }

Thus, there are three prime dichotomies.

(iii) It is obvious that that the three prime dichotomies are essential and form a minimum cover.

The state codes derived are;

	P1	P2	P3
S1	0	0	1
S2	0	0	0
S3	1	1	0
S4	1	0	1
S5	1	0	0

Note that for P3 we assign S2 0 to make its code different from S1. Note also that the obtained code satisfies the covering and disjunctive constraints. The code of S2 is covered by all other codes. Also, the code of S4 = 101 is the OR of S1 = 001 and S5 = 100.

Also note that the input encoding constraints are all satisfied.

The codes of S1 and S2 give the cube 00- which does not intersect any of the other codes.

The codes of S1, S2, S4, and S5 give the cube -0- which does not intersect S3.

The codes of S3 and S5 produce the cube 1o- which does not intersect other codes.

(iv) Let us assume the input X, the output Z, and the flip-flops F3, F2, and F1. Then, the minimized table will be:

X	F3 F2 F1	F_3^+ F_2^+ F_1^+	Z
0	0 0 -	1 1 0	1
1	- 0 -	1 0 0	1
1	1 - 0	0 0 1	1
0	1 0 1	1 0 1	0
0	1 0 0	0 0 1	0

Thus, we guarantee that five product terms are sufficient to implement the next state and output equations.

Let us next, obtain the transition table from the original state table and use K-maps minimization to obtain the next-state and output equations.

Transition Table

X	F_3 F_2 F_1	F_3^+ F_2^+ F_1^+	Z
0	0 0 1	1 1 0	1
1	0 0 1	1 0 0	x
0	0 0 0	1 1 0	x
1	0 0 0	1 0 0	1
0	1 1 0	0 0 0	0
1	1 1 0	0 0 1	1
0	1 0 1	1 0 1	0
1	1 0 1	1 0 0	1
0	1 0 0	0 0 1	0
1	1 0 0	1 0 1	1

$Z =$

XF_3 \ F_2F_1	00	01	11	10
00	X	1	X	X
01	0	0	X	0
11	1	1	X	1
10	1	X	X	X

$Z = X + \bar{F}_3$

$F_3^+ =$

XF_3 \ F_2F_1	00	01	11	10
00	1	1	X	X
01	0	1	X	0
11	1	1	X	0
10	1	1	X	X

$F_3^+ = \bar{F}_3 + F_1 + X\bar{F}_2$

F_2^+

xF3 \ F2F1	00	01	11	10
00	1	1	x	x
01	0	0	x	0
11	0	0	x	0
10	0	0	x	x

$F_2^+ = \bar{x}\,\bar{F_3}$

F_1^+

xF3 \ F2F1	00	01	11	10
00	0	0	x	x
01	1	1	x	0
11	1	0	x	1
10	0	0	x	x

$F_1^+ = \bar{x}\,F_3\,\bar{F_2} + x\,F_3\,\bar{F_1}$

Note that the obtained equations are consistent with the minimized table given in page 8 and are more optimized due to the use of don't care conditions.

Running the sis command stg-to-network with the obtained code produces the following solution:

$$z = \bar{x}\,\bar{F_3} + x\,\bar{F_2} + x\,F_3\,\bar{F_1}$$
$$F_3^+ = \bar{x}\,\bar{F_3} + x\,\bar{F_2} + \bar{x}\,F_1$$
$$F_2^+ = \bar{x}\,\bar{F_3}$$
$$F_1^+ = x\,F_3\,\bar{F_1} + \bar{x}\,F_3\,\bar{F_2}$$

(5 product terms)
(21 literals)

Note that this is less optimized than what we obtained because we did single-output optimization while espress performs multiple-outp. optimization minimizing the number of product terms. Note that the number of product terms obtained is five.

To run espresso with single-output optimization, we use the command stg-to-network -e 2 and we obtain the solution:

$$Z = X + \bar{F}_3$$
$$F_3^+ = \bar{F}_3 + F_1 + X\bar{F}_2$$
$$F_2^+ = \bar{X}\bar{F}_3$$
$$F_1^+ = \bar{X}F_3\bar{F}_2 + XF_3\bar{F}_1 \qquad (14 \text{ literals})$$

which is identical to the solution obtained using K-map.

(V) Running the SIS command state_assign nova produces the following state codes:

S1	000
S2	001
S3	110
S4	011
S5	101

The logic produced is:

$$Z = X + \bar{X}\bar{F}_3\bar{F}_2$$
$$F_3^+ = \bar{X}\bar{F}_3\bar{F}_2 + X\bar{F}_3$$
$$F_2^+ = XF_3F_1 + \bar{X}\bar{F}_3\bar{F}_2 + \bar{X}F_1F_2 \qquad (5 \text{ product terms}, 25 \text{ literals})$$
$$F_1^+ = XF_3F_1 + X\bar{F}_3 + \bar{X}F_2$$

Note that the number of product terms required is the same as our solution but the number of literals in our solution is less. Thus, the state assignment obtained by us is better than that obtained by Nova.

(VI) One-hot encoding

State assignment using the command
state-assign jedi -e h

The state codes are :

S1	10000
S2	01000
S3	00100
S4	00010
S5	00001

The equations produced are :

$$Z = x F_2 + \bar{F}_2 \bar{F}_3 \bar{F}_5 \bar{x} + x \bar{F}_2 \bar{F}_3 + x F_3$$

$$F_1 = x F_2 + \bar{x} F_3$$

$$F_2 = \bar{x} \bar{F}_2 \bar{F}_3 \bar{F}_5$$

$$F_3 = x \bar{F}_2 \bar{F}_3$$

$$F_4 = \bar{x} F_2$$

$$F_5 = \bar{x} F_5 + x F_3$$

Note that the number of product terms is 7 and the number of literals is 28, which is an inferior solution and also uses 5 FFs instead of 3.

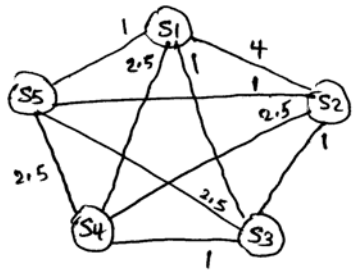
(vii) From the original state table, we extract the following equations of all states and output

$$Z = \bar{x}\,S1 + x\,S2 + x\,S3 + x\,S4 + x\,S5$$
$$S1 = x\,S3 + \bar{x}\,S5$$
$$S2 = \bar{x}\,S3$$
$$S3 = \bar{x}\,S1 + \bar{x}\,S2$$
$$S4 = \bar{x}\,S4 + x\,S5$$
$$S5 = x\,S1 + x\,S2 + x\,S4$$

We next compute the weights between all state pairs.

$$M_{S1,S2} = (1*1) + \frac{3}{2}(0*0 + 0*0 + 1*1 + 0*0 + 1*1)$$
$$= 4$$
$$M_{S1,S3} = (1*1) + \frac{3}{2}(0*1 + 0*1 + 1*0 + 0*0 + 1*0)$$
$$= 1$$
$$M_{S1,S4} = (1*1) + \frac{3}{2}(0*0 + 0*0 + 1*0 + 0*1 + 1*1)$$
$$= 2.5$$
$$M_{S1,S5} = (1*1) + \frac{3}{2}(0*1 + 0*0 + 1*0 + 0*1 + 1*0)$$
$$= 1$$
$$M_{S2,S3} = (1*1) + \frac{3}{2}(0*1 + 0*1 + 1*0 + 0*0 + 1*0)$$
$$= 1$$
$$M_{S2,S4} = (1*1) + \frac{3}{2}(0*0 + 0*0 + 1*0 + 0*1 + 1*1)$$
$$= 2.5$$
$$M_{S2,S5} = (1*1) + \frac{3}{2}(0*1 + 0*0 + 1*0 + 0*1 + 1*0)$$
$$= 1$$
$$M_{S3,S4} = (1*1) + \frac{3}{2}(1*0 + 1*0 + 0*0 + 0*1 + 0*1)$$
$$= 1$$
$$M_{S3,S5} = (1*1) + \frac{3}{2}(1*1 + 1*0 + 0*0 + 0*1 + 0*0)$$
$$= 2.5$$
$$M_{S4,S5} = (1*1) + \frac{3}{2}(0*1 + 0*0 + 0*0 + 1*1 + 1*0)$$
$$= 2.5$$

The obtained fanout-oriented graph:



There could be several solutions to this problem. First, S1 and S2 need to be assigned uni-distance codes. Let S1 = 000 and S2 = 001. Next, we need to assign the code for S4 to be uni-distance to S1 and S2. This is not possible, so, we assign S4 = 010. Next, we assign S5 to be uni-distance to S4; S5 = 011. Finally, S3 needs to be uni-distance to S5; S3 = 111.

So, the obtained code is:

S1 = 000, S2 = 001, S3 = 111, S4 = 010, S5 = 011

Running the command stg-to-network using these codes followed by fx we obtain the following equations:

$$z = x + \bar{F}_2$$

$$F_1^+ = \bar{x}\bar{F}_2$$

$$F_2^+ = x\bar{F}_1 + \bar{F}_2 + \bar{F}_3$$

$$F_3^+ = \bar{x}F_1 + \bar{F}_2 + x\bar{F}_3 \qquad (13 \text{ literals})$$

using the command state-assign jedi -e o
we obtain the following state assignment :
S1 = 111 , S2 = 110 , S3 = 101 , S4 = 011 , S5 = 001
This results in the following equations :

$$Z = x + \bar{x} F_1 F_2$$
$$F_1^+ = \bar{x} F_1 F_2 + F_1 \bar{F}_2 + \bar{x} \bar{F}_2$$
$$F_2^+ = \bar{F}_2 + \bar{x} F_1 F_2$$
$$F_3^+ = x + \bar{x} \bar{F}_1 + \bar{x} F_1 F_2 \qquad (20 \text{ literals})$$

Note that F_3 is redundant in this implementation and can be eliminated. This shows that this machine can be implemented with two FFs only. Having 5 states, this means that there are compatible states. Examining the state table, one can notice that states S1 and S2 are compatible.

(viii) Fan-in oriented algorithm

we first compute the weight between all state pairs.

$M_{S1,S2} = (1*0) + (1*1) + 3(0*0 + 0*0 + 1*1 + 0*0 + 1*0)$
$= 4$

$M_{S1,S3} = (1*0) + (1*2) + 3(0) = 2$

$M_{S1,S4} = (1*1) + (1*1) + 3(1) = 5$

$M_{S1,S5} = (1*3) + (1*0) + 3(0) = 3$

$M_{S2,S3} = (0*0) + (1*2) + 3(0) = 2$

$M_{S2,S4} = (0*1) + (1*1) + 3(0) = 1$

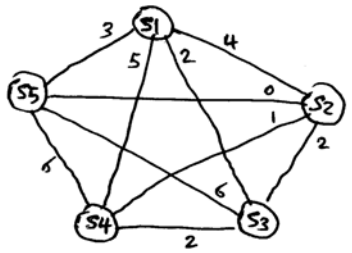
$M_{S2,S5} = (0*3) + (1*0) + 3(0) = 0$

$M_{S3,S4} = (0*1) + (2*1) + 3(0) = 2$

$M_{S3,S5} = (0*3) + (2*0) + 3(1*1 + 1*1) = 6$

$M_{S4,S5} = (1*3) + (1*0) + 3(1*1) = 6$

The obtained fan-in graph:



Since the highest weight is 6, we assign codes of S3 and S4 to be uni-distance to S5.

Let S5 = 000, then S3 = 001, and S4 = 010.

Next, we need to assign S1 to be unidistance from S4; S1 = 011. Then, we assign S2 to be unidistance from S1; S2 = 111. So, the obtained state codes:

S1 = 011, S2 = 111, S3 = 001, S4 = 010, S5 = 000.

Running the command stg-to-network, followed by fx, we obtain the following equations:

<u>without FX</u>:

$Z = x + \bar{x} F_2 F_3$

$F_1^+ = \bar{x} \bar{F}_2 F_3$

$F_2^+ = \bar{F}_2 + \bar{x} \bar{F}_3$

$F_3^+ = \bar{x} F_2 F_3 + \bar{F}_2 F_3 + \bar{x} F_2$ (17 literals)

After running FX:

$t = \bar{x} F_3$

$z = x + F_2 t$

$F_1^+ = \bar{F_2} + \bar{x} \bar{F_3}$

$F_3^+ = t F_2 + \bar{F_2} F_3 + \bar{x} \bar{F_2}$ (14 literals)

Note that F_2 is redundant and is eliminated by FX.

Running the command state-assign jedi -e i we get the following state assignment:

$s1 = 010$, $s2 = 000$, $s3 = 001$, $s4 = 100$, $s5 = 101$

and we get the following equations: (without FX)

$z = x F_1 + x \bar{F_1} F_3 + \bar{F_1} \bar{F_3}$

$F_1^+ = F_1 \bar{F_3} + x F_1 + x \bar{F_3}$

$F_2^+ = \bar{x} F_1 F_3 + x \bar{F_1} F_3$

$F_3^+ = x \bar{F_3} + \bar{F_1} \bar{F_3}$ (23 literals)

Note that F_2 is redundant and can be eliminated, with FX, we get the following equations:

$t1 = F_1 + \bar{F_3}$

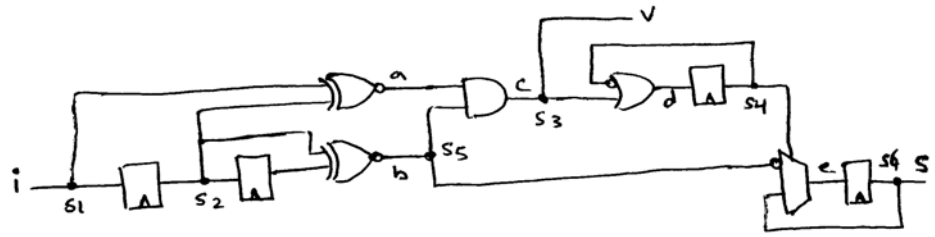
$t2 = x + \bar{F_1}$

$z = x F_1 + \bar{F_1} \bar{F_3} + x \bar{t_1}$

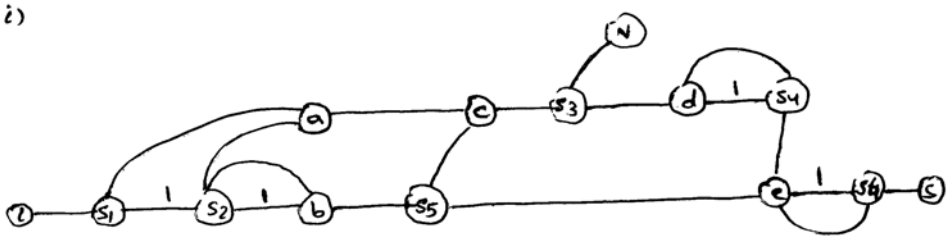
$F_1^+ = F_1 \bar{F_3} + x t1$

$F_3^+ = t_2 \bar{F_3}$ (16 literals)

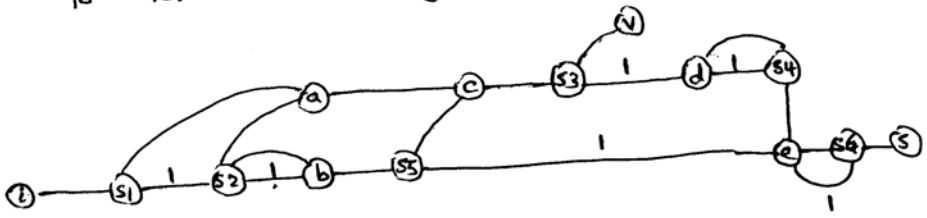
Q3

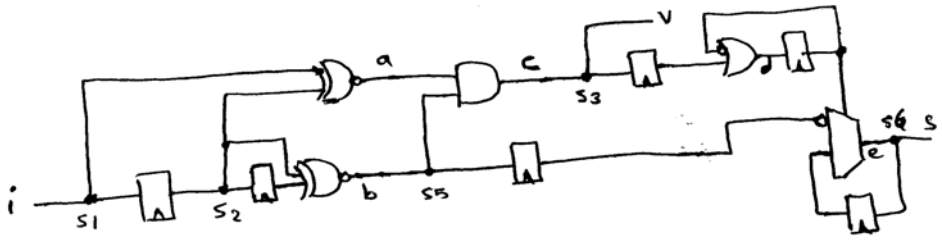


(i)

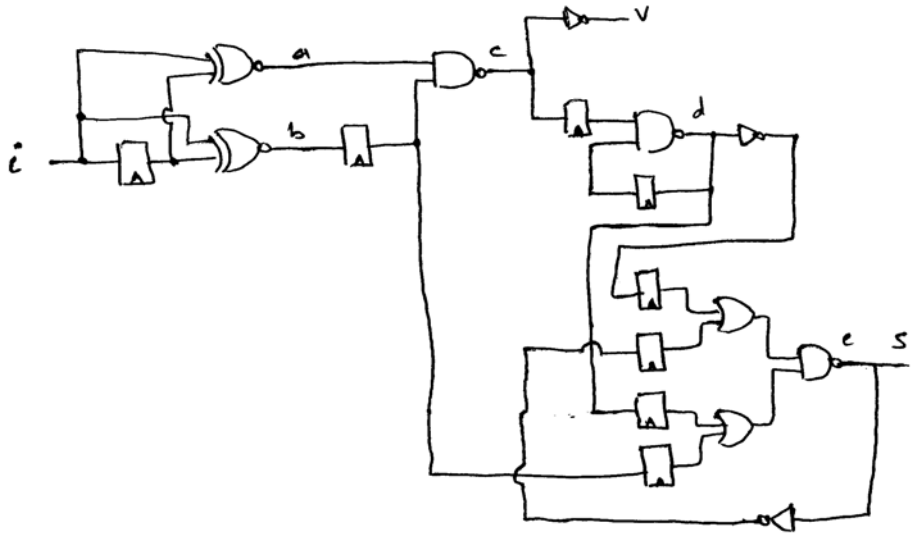


(ii) Assuming that the delay of each gate is 1, the longest delay is 3. we will assume that the delay of XNOR and MUX larger than the other gates. We can retime the graph to reduce the delay as follows:





(iii) Using the SIS retime command, we obtain the following solution after mapping to the library synch.genlib:



Note that in this solution one FF is moved forward across S2 and the 2 FFs are then moved from the inputs of b to its output. However, this is unnecessary as it does not reduce the delay. Also, note that in this solution there are 2 FF's at the MUX enable, one for the enable and one for its complement. These two FF's can be made 1 to obtain our solution. However, this will increase the delay by one inverter.