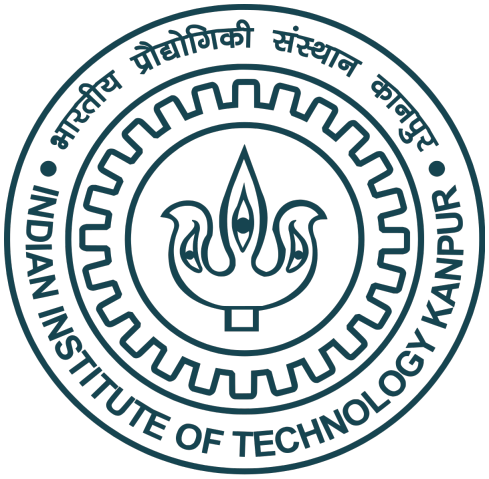
CS220A: Computer Organization



Assignment 3 Report

Aditya Tanwar
200057

Akhil Agrawal
200076

Suket Raj
201013

February, 2022

Question 1

Construct a finite state machine for a sequence detector that detects the sequence 1010. The FSM should permit overlapping too. For example if the input sequence is 01101010, the corresponding output sequence is 00000101.

i. Construct the state diagram, transition and output table, excitation table.

ii. Then build the K-map for the same and design the circuitry or logic diagram.

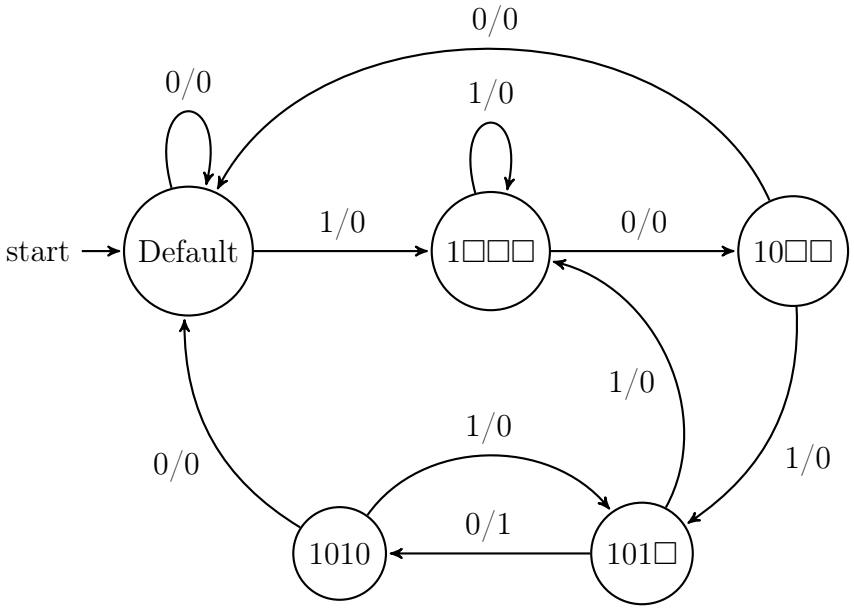
iii. Write the Verilog code to implement the sequence detector.

iv. Add a test bench to test the sequence detector given fifteen different inputs. Put five-time unit delay between consecutive inputs.

States: The given problem can be solved using a finite state machine with *5 states*. Since a minimum of 3 input bits (denoted by x_i) are required to generate 5 states, we define the states as follows:

State	x_1	x_2	x_3
S_0 (Default)	0	0	0
S_1 (1□□□)	0	0	1
S_2 (10□□)	0	1	0
S_3 (101□)	0	1	1
S_4 (1010)	1	0	0

State Diagram : This is the state diagram of the *FSM* for detecting the sequence *1010-*



It is easily observable that the drawn *FSM* is a *Mealy Machine*.

Further, the input (i) and output (o) corresponding to each transition is written above its (the transition's) arrow like

$$P.S. \xrightarrow{i/o} N.S.$$

Excitation Table : The "Excitation Table" corresponding to the *FSM* made above is drawn below-

P.S.	$x_1(t)$	$x_2(t)$	$x_3(t)$	Input	*N.S.*	$x_1(t+1)$	$x_2(t+1)$	$x_3(t+1)$	Output
S_0	0	0	0	0	S_0	0	0	0	0
				1	S_1	0	0	1	0
S_1	0	0	1	0	S_2	0	1	0	0
				1	S_1	0	0	1	0
S_2	0	1	0	0	S_0	0	0	0	0
				1	S_3	0	1	1	0
S_3	0	1	1	0	S_4	1	0	0	1
				1	S_1	0	0	1	0
S_4	1	0	0	0	S_0	0	0	0	0
				1	S_3	0	1	1	0

Karnaugh Map : *4* separate *K-maps* have been built, one corresponding to each of $x_1(t+1)$, $x_2(t+1)$, $x_3(t+1)$, o, where o denotes *output*. We also use x_1 to refer to $x_1(t)$ as shorthand and X_1 to refer to $x_1(t+1)$ as shorthand, and i to denote input.

$i\,x_1$ \ $x_2\,x_3$	00	01	11	10
00	0	0	1	0
01	0	-	-	-
11	0	-	-	-
10	0	0	0	0

$X_1 = (\neg i \wedge x_2 \wedge x_3)$

$i\,x_1$ \ $x_2\,x_3$	00	01	11	10
00	0	1	0	0
01	0	-	-	-
11	1	-	-	-
10	0	0	0	1

$X_2 = (\neg i \wedge \neg x_2 \wedge x_3) \vee (i \wedge x_1) \vee (i \wedge x_2 \wedge \neg x_3)$

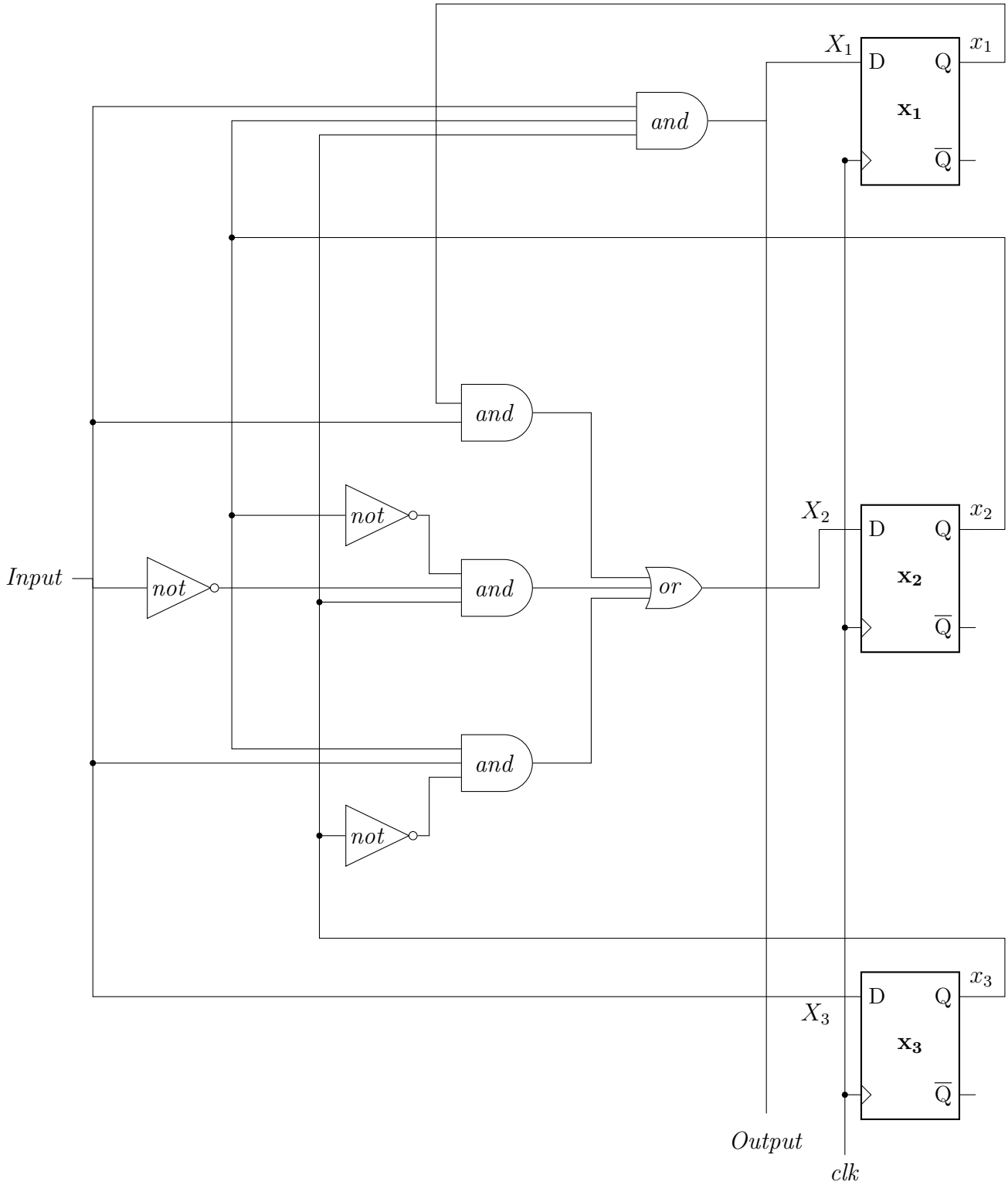
$i\,x_1$ \ $x_2\,x_3$	00	01	11	10
00	0	0	0	0
01	0	-	-	-
11	1	-	-	-
10	1	1	1	1

$X_3 = (i)$

$i\,x_1$ \ $x_2\,x_3$	00	01	11	10
00	0	0	1	0
01	0	-	-	-
11	0	-	-	-
10	0	0	0	0

Out $= (\neg i \wedge x_2 \wedge x_3)$

Circuit Diagram : The input to the D-FlipFlops is the new state ($X_i = x_i(t+1)$) that is required to be stored, and the output of the flipflops is the current state ($x_i = x_i(t)$) that is required in the calculation of the new states as well as the output.



Transition and Output Table : Following is the "Transition and Output Table" for the drawn *FSM*-

P.S.	*N.S.*		*O*	
	i		i	
	0	1	0	1
S_0	S_0	S_1	0	0
S_1	S_2	S_1	0	0
S_2	S_0	S_3	0	0
S_3	S_4	S_1	1	0
S_4	S_0	S_3	0	0

P.S. stands for the Present State.

N.S. stands for the Next State.

O stands for Output.

i stands for Input.

The state variables (x_1, x_2, x_3) have not been shown in the interest of space.

Question 2

Construct a finite state machine for the 3-bit odd parity bit generator. A serial parity-bit generator is a two-terminal circuit that receives coded messages and adds a parity bit to every m bits of the message so that the resulting outcome is an error detecting code. The inputs are assumed to arrive in strings of three symbols (m=3) and the string is spaced apart by single time units (i.e., the fourth place is blank). The parity bits are inserted in the appropriate spaces so that the resulting outcome is a continuous string of symbols without spaces. For even parity, a parity bit 1 is inserted, if and only if the numbers of 1s in the preceding string of three symbols are odd. For odd parity, a parity bit 1 is inserted, if and only if the numbers of 1s in the preceding string of three symbols is even. Here we will focusing on designing only odd parity generator.

i. Construct the state diagram, state table, transition and output table, excitation table.

ii. Then build the K-map for the same and design the circuitry or logic diagram.

iii. Then write the Verilog code module to implement the 3-bit odd parity bit generator.

iv. Now, add a test bench to test the 3-bit odd parity bit generator given fifteen different inputs. Put five-time unit delay between consecutive inputs.

States: We have used *2* states in the *FSM*, namely S_E (for even) and S_O (for odd). They are represented by the state variable s.

State	s
S_E	0
S_O	1

We briefly describe the significance of each state individually for the sake of completeness-

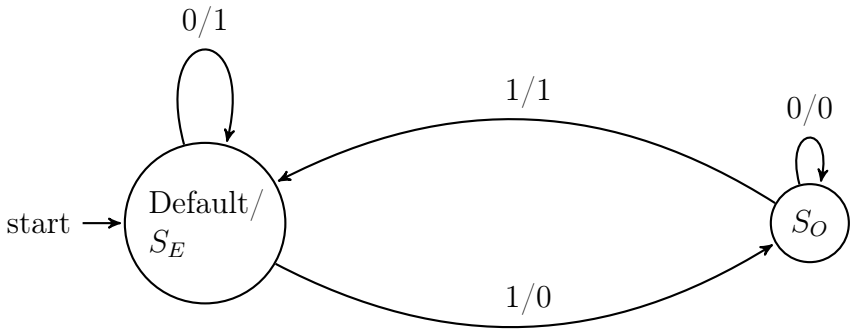
S_E: Even state, represents the state where number of set bits obtained yet is of even parity. By default, the number of set bits will be zero. i.e. even.

S_O: Odd state, represents the state where number of set bits obtained yet is of odd parity.

Assumptions : It is assumed that the output will only be read when the 3 input bits have been read because the intermediate output will not be of much significance. This is because the output is needed only at the end of the three bits, while it also needs to be specified in the transitions, some of which might be intermediate and should not have had any output.

An easy work-around for this technicality is to expand the number of states the *FSM* can possibly have. This was the original approach and an *FSM* with *5* states has been drawn at the end for reference.

State Diagram : This is the state diagram of an *FSM* with *2* states for a *3-bit* odd parity bit generator-



Further, the input (i) and output (o) corresponding to each transition is written above its (the transition's) arrow like

$$P.S. \xrightarrow{i/o} N.S.$$

It is easily observable that the *FSM* is a *Mealy Machine*.

Excitation Table : The "Excitation Table" corresponding to the *FSM* made above is drawn below.

P.S.	$s(t)$	*Input*	*N.S.*	$s(t+1)$	Output
S_E	0	0	S_E	0	1
		1	S_O	1	0
S_O	1	0	S_O	1	0
		1	S_E	0	1

Transition and Output Table : "Transition and Output Table" for the given FSM is:

P.S.	*N.S.*		*O*	
	i		i	
	0	1	0	1
S_E	S_E	S_O	1	0
S_O	S_O	S_E	0	1

Karnaugh Map : *2* separate *K-maps* have been drawn, one corresponding to each of $s(t+1)$, o, where o denotes output. We also use s to refer to $s(t)$, S to refer to $s(t+1)$, and i to refer to input as shorthand.

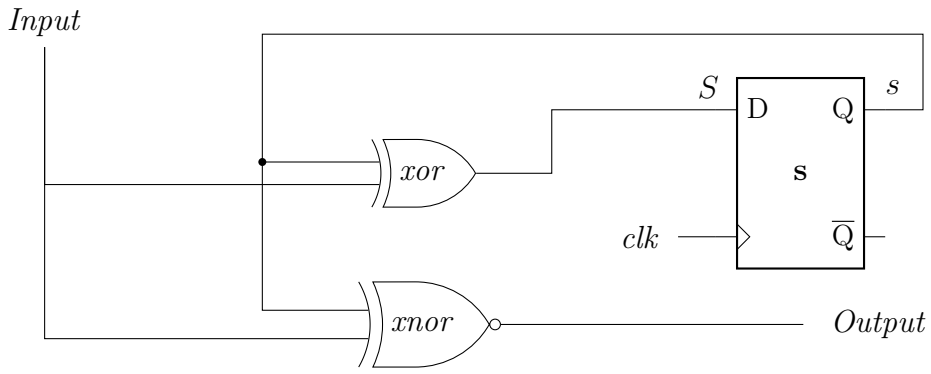
i \ s	0	1
0	0	1
1	1	0

i \ s	0	1
0	1	0
1	0	1

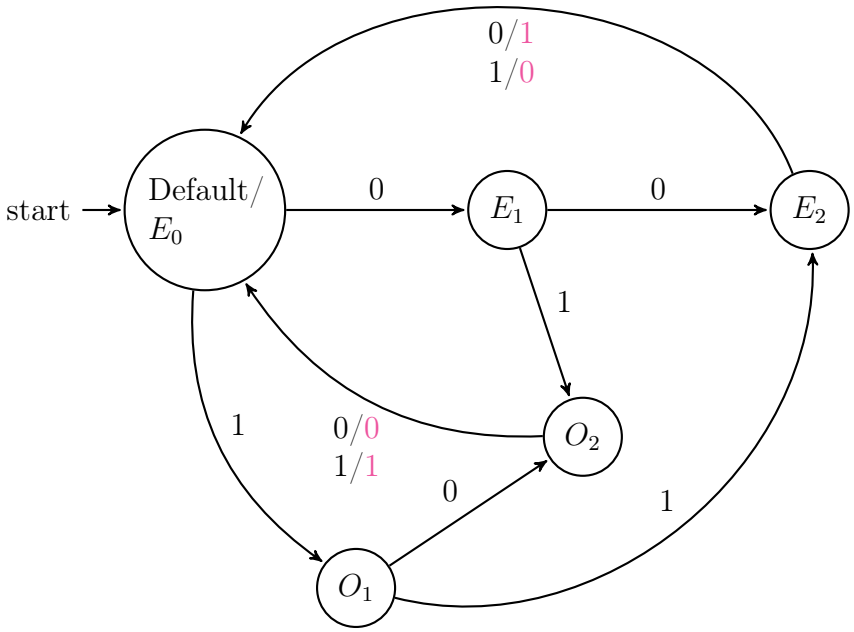
$$S = (\neg i \wedge s) \vee (i \wedge \neg s)$$
$$= (i \oplus s)$$

$$\text{Out} = (i \wedge s) \vee (\neg i \wedge \neg s)$$
$$= \neg(i \oplus s)$$

Circuit Diagram : The input to the D-FlipFlop is the new state ($S = s(t+1)$) that is required to be stored, and the output of the flipflops is the current state ($s = s(t)$) that is required in the calculation of the new state and the output.



The alternate approach which has *5-states* has been elaborated below. We design the *FSM* to not give any output for some transitions, but give outputs specifically from transitions from the states E_2 and O_2. The output, wherever applicable, has been colored.



The states of the *FSM* have been described below briefly-

E_0: This is the default state, the starting state, as well as the final state (i.e., after 3 bits have been read). ∵ No bits have been read yet, the number of *1's* read are zero, and hence even (E).

E_1: State that can be derived only from E_0. Denotes the state when one bit has been read and it was not *1*.

O_1: State that can be derived only from E_0. Denotes the state when one bit has been read and it was *1*.

E_2: State that always leads to E_0. Denotes the state when two bits have been read and either both of them were *1* or none of them were *1*. When an input bit has been read in this state, the next state is E_0 since the *3-bit* input has been completely read, thus, returning an appropriate output as well as resetting the state of the *FSM*.

O_2: State that always leads to E_0. Denotes the state when two bits have been read and exactly one of them was *1*. When an input bit has been read in this state, the next state is E_0 since the *3-bit* input has been completely read, thus, returning an appropriate output as well as resetting the state of the *FSM*.

Note: The "Excitation Table", "Transition and Output Table", etc. have been omitted for this since this is not the final approach used in code, but rather an alternate approach. Regardless, the code for this implementation has also been provided unofficially.