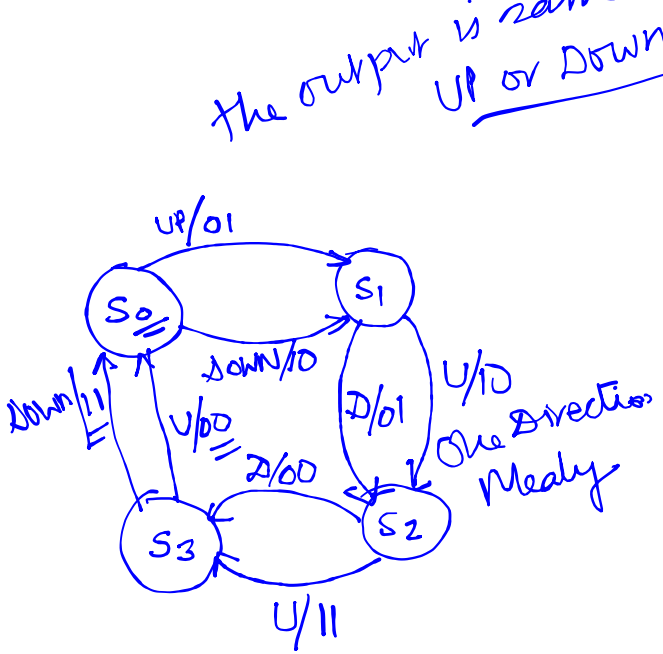
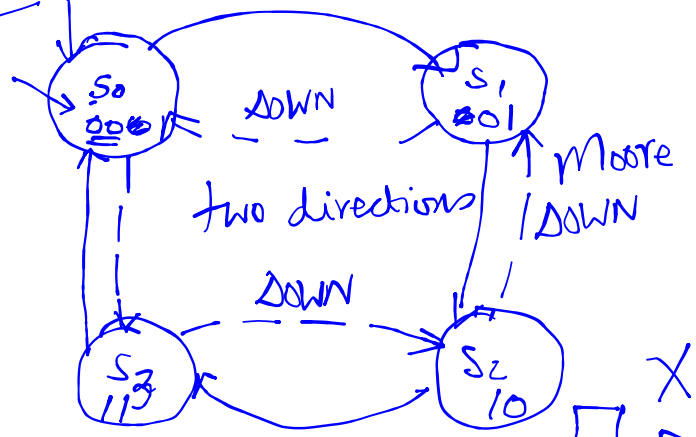
$$Y_{out} = f[\text{input, present output/state}]$$ ⟸ Sequential circuit.

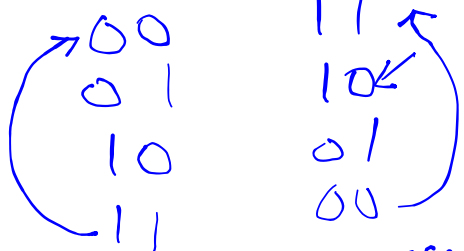
000 → 001 → 010 → 011 Binary counter



the output is same UP or DOWN

Moore UP

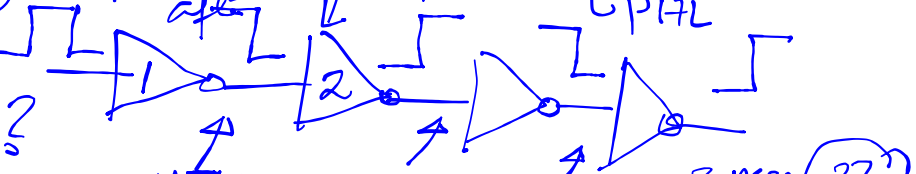




X responds after 1 complete

$t_{PLH} = 8$ nsec

$t_{PHL} = 7$ nsec

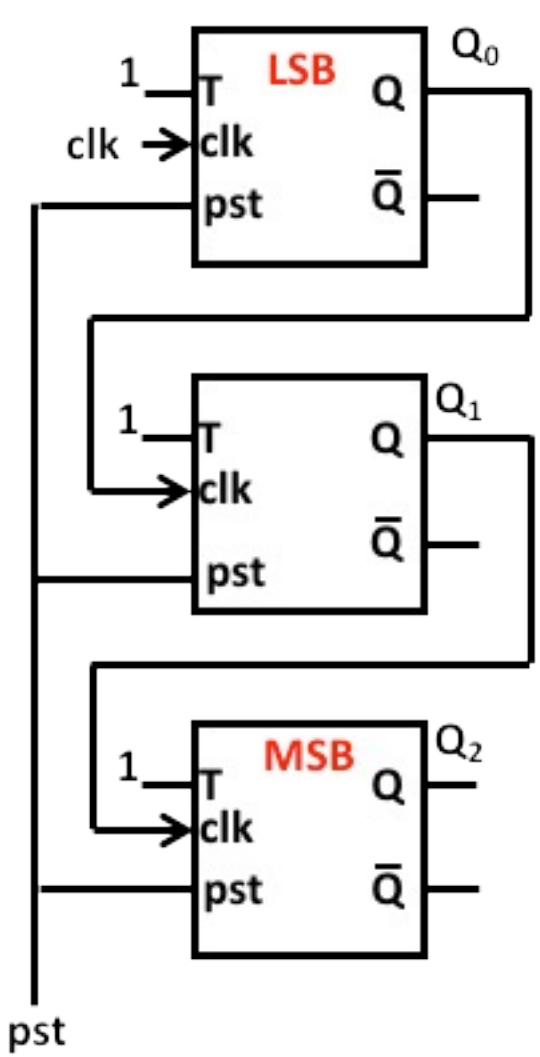


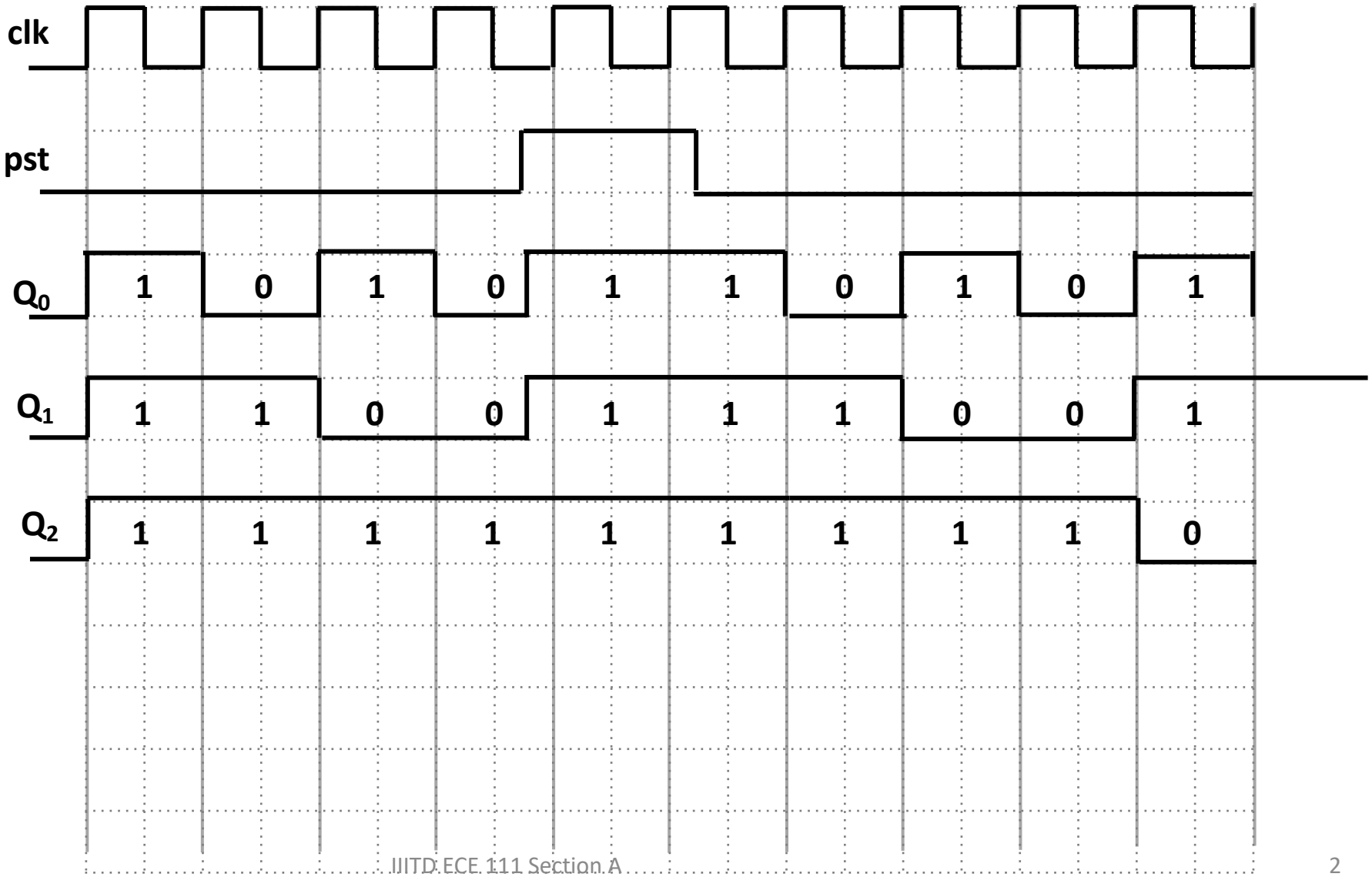
assumption?

over estimating delay

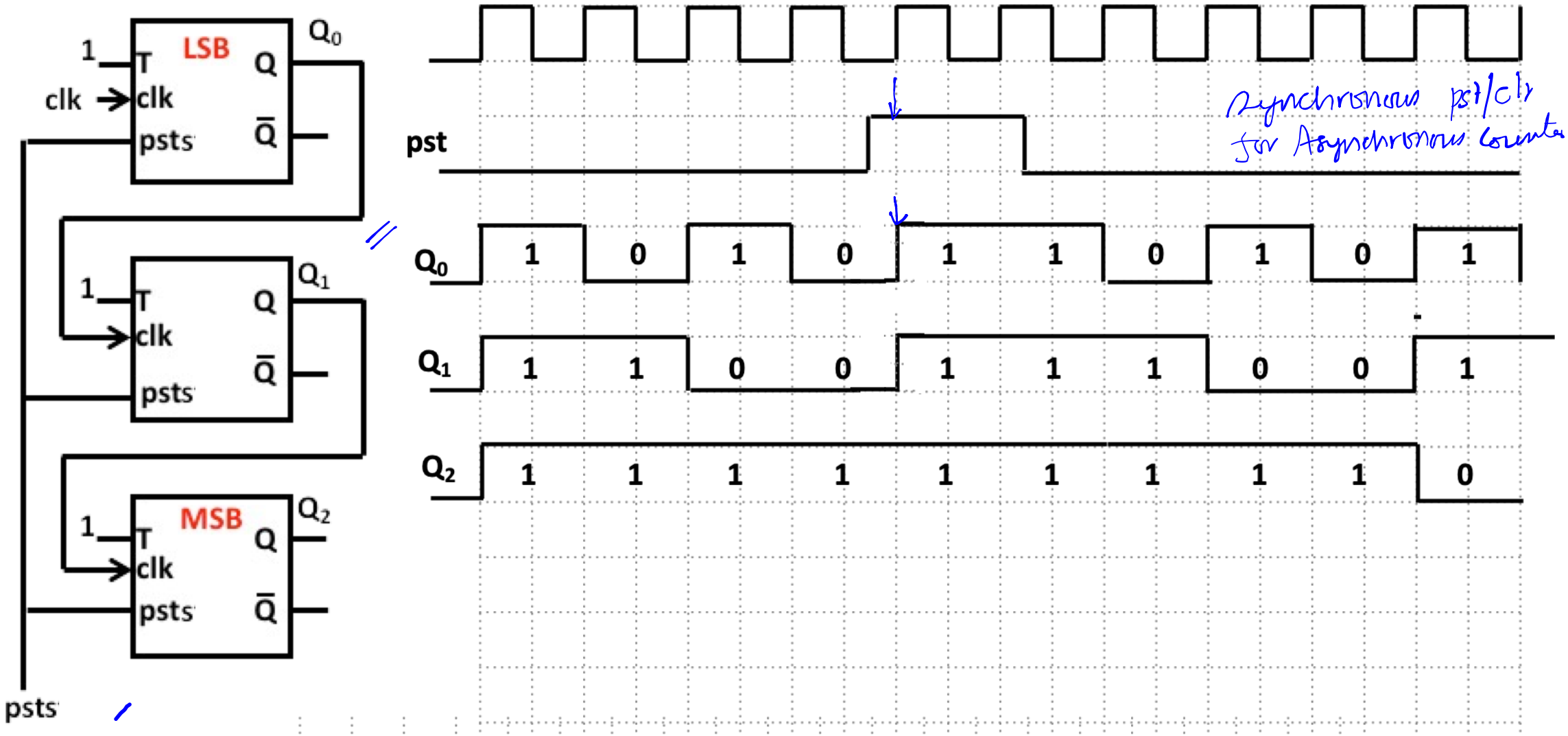
Practical Cases.

Unsigned Binary Down Counter with Asynchronous Preset:

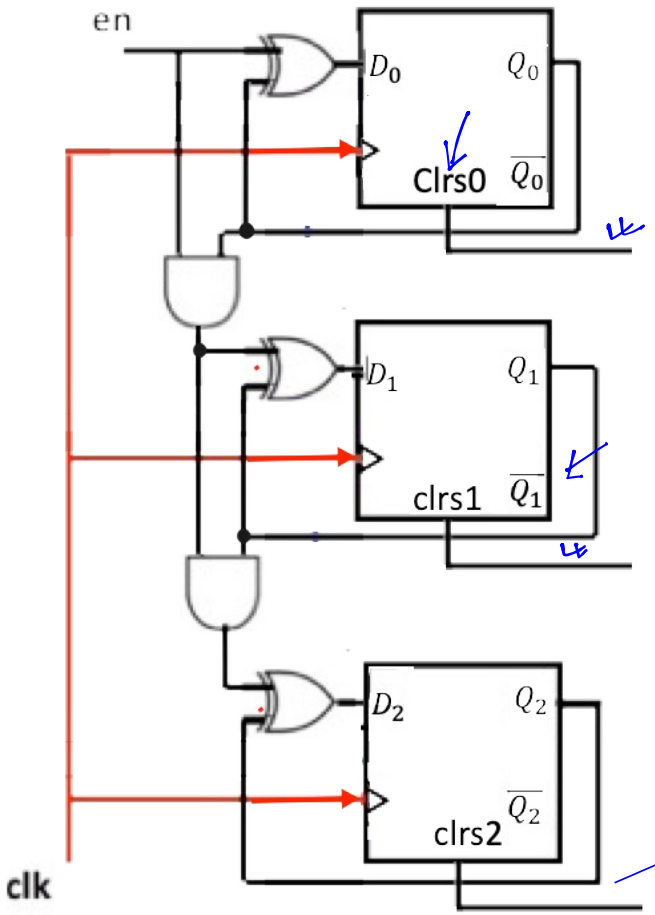




Unsigned Binary Down Counter with Synchronous Preset:



Synchronous Mod 6 counter with synchronous clear:



This is a 3-bit synchronous counter. If we need to convert it to a mod 6 counter, it should go to 000 after 101. If we put $clrs0 = clrs1 = clrs2 = Q_2 \cdot \overline{Q_1} \cdot Q_o$, the counter will go to 000 and continue the count.

In the figure the DFF is shown with an input clrs, which represents a synchronous clear input.

Since we design a synchronous counter using STD or STT we do not resort to the use of synchronous clr or pst inputs. Hence, commercially FFS that have a clr and/or pst inputs all have asynchronous clr/pst. Hence, we do not use clr/pst with synchronous counters.

FSM:

- Design counter whose output states $000 \rightarrow 011 \rightarrow 110$ using only two D-FF.

Present State		Next State		FF Inputs		Output		
P_1	P_0	N_1	N_0	D_1	D_0	O_2	O_1	O_0
0	0	0	1	0	1	0	0	0
0	1	1	0	1	0	0	1	1
1	0	0	0	0	0	1	1	0
1	1	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ

$$D_1 = P_0 \; ; \quad D_0 = \overline{P_1}\,\overline{P_0} \; ;$$

$$O_2 = P_1 \; ; O_1 = P_1 P_0 \; ; O_0 = P_0$$

Pattern Detection:

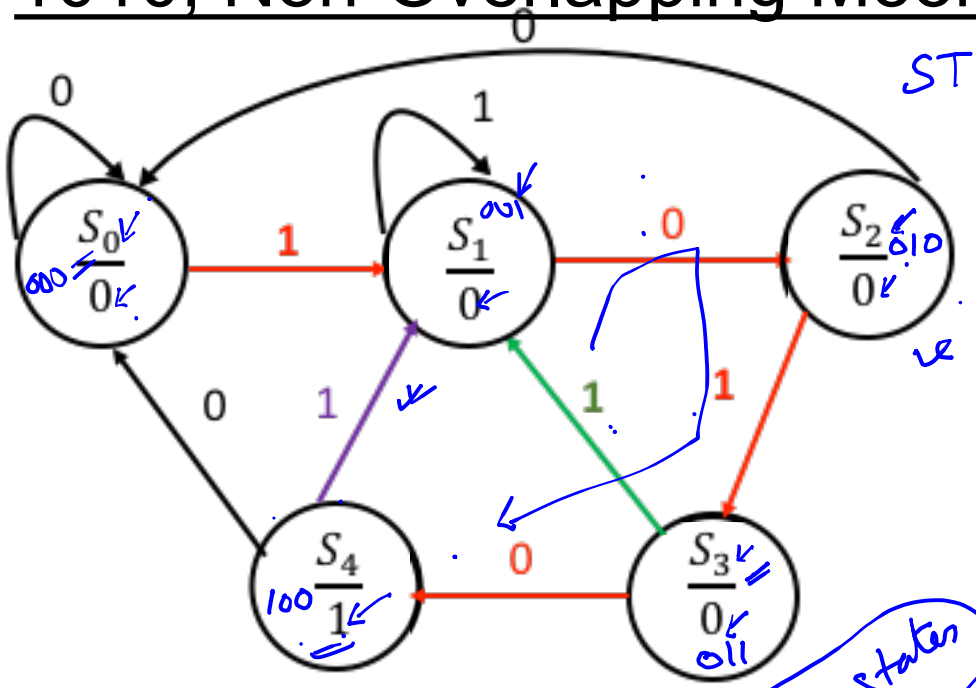
codes & Locks:

Non-Overlapping Pattern

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	0	1	0	0	1	0	1	1	0	1	0	1	0	1	0	1	0	1	0
0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1
0	0	0	1	0	0	0	0	0	0	0	1	0	1	0	1	0	1	0	1
0	0	0	1	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	0
0	0	0	1	0	0	1	0	0	1	0	1	0	1	0	1	0	1	0	1

This is a long sequence of bits received. We need to detect whenever the sequence shows a pattern 1010 without an overlap.

1010, Non-Overlapping Moore:



1010

STD 1011010 1 whenever the pattern is detected

S_0 S_3 S_1 S_4

Output is 1

Convenient workspace

Exactly same as STD or STG

PS	I	NS	output
S_0	0	S_0	0
S_0	1	S_1	0
S_1	0	S_2	0
S_1	1	S_1	0
S_2	0	S_0	0
S_2	1	S_3	0
S_3	0	S_4	1
S_3	1	S_1	0
S_4	0	S_0	0
S_4	1	S_1	0

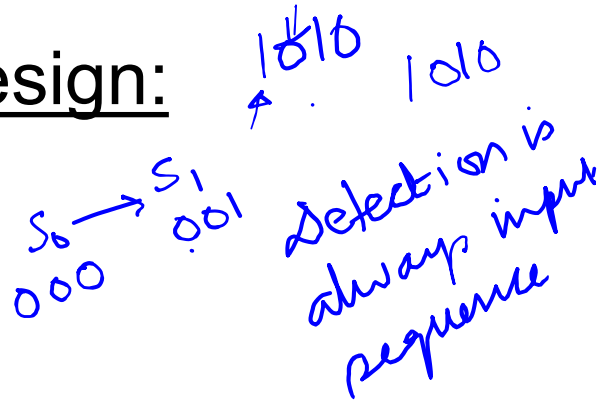
$Q_2 \bar{Q}_1 \bar{Q}_0$

Design the circuit avoid unused states

$S_0 \rightarrow 000; S_1 \rightarrow 001\ ; S_2 \rightarrow 010\ ; S_3 \rightarrow 011\ ; S_4 \rightarrow 100$

Output $Z = Q_2$

Design:



The Design Using D-FF is left as a HW to the class.

STT

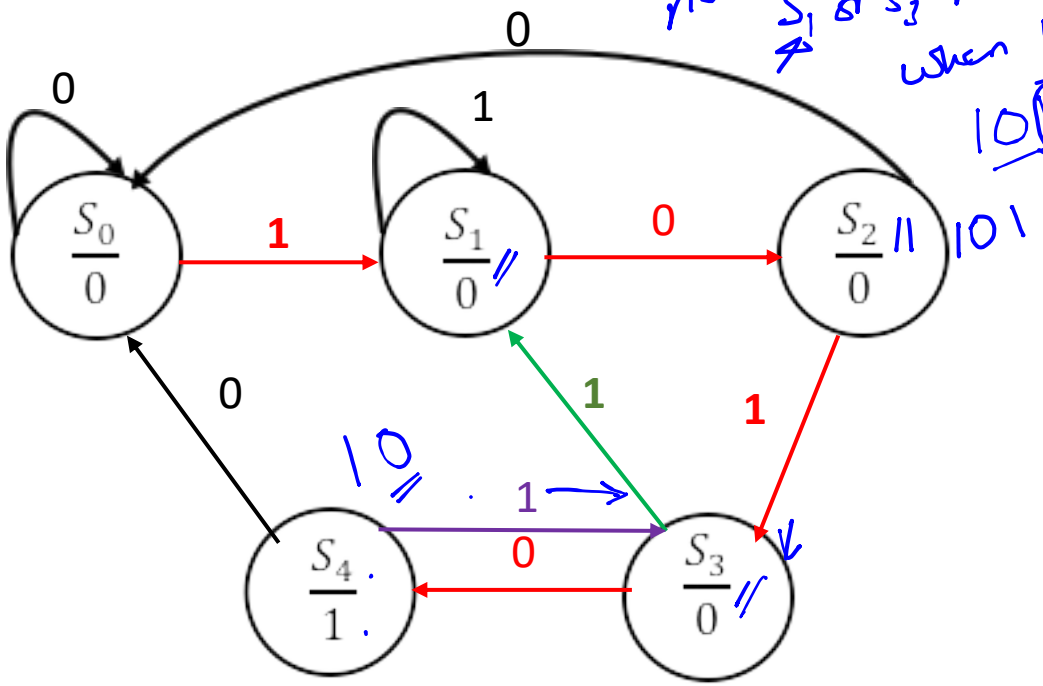
PS $Q_2Q_1Q_0$	I	NS $Q_2Q_1Q_0$	Q_2 PS	Q_2 NS	D_2	Q_1 PS	Q_1 NS	D_1	Q_0 PS	Q_0 NS	D_0
000 S_0	0	000 S_0	0	0	0	0	0	0	0	0	0
000	1	001 S_1	0	0	0	0	0	0	0	1	1
001 S_1	0	010 S_2	0	0	0	0	1	1	1	0	0
001 S_1	1	001 S_1	0	0	0	0	0	0	1	1	1
010 S_2	0	000 S_0	0	0	0	1	0	0	0	0	0
010 S_2	1	011 S_3	0	0	0	1	1	1	0	1	1
011 S_3	0	100 S_4	0	1	1	1	0	0	0	1	1
011 S_3	1	001 S_1	0	0	0	1	0	0	1	1	1
100 S_4	0	000 S_0	1	0	0	0	0	0	0	0	0
100 S_4	1	001 S_1	1	0	0	0	0	0	0	1	1
101	ϕ	ϕ	ϕ	ϕ	$\phi=0$	$\phi=0$	ϕ	$\phi=0$	$\phi=0$	ϕ	$\phi=0$
110	ϕ	ϕ	ϕ	ϕ	$\phi=0$	$\phi=0$	ϕ	$\phi=0$	$\phi=0$	ϕ	$\phi=0$
111	ϕ	ϕ	ϕ	ϕ	$\phi=0$	$\phi=0$	ϕ	$\phi=0$	$\phi=0$	ϕ	$\phi=0$

Overlapping Pattern Detection:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	0	1	0	0	1	0	1	1	0	1	0	1	0	1	0	1	0	1	0
0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1
0	0	0	1	0	0	0	0	0	0	0	1	0	1	0	1	0	1	0	1
0	0	0	1	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	0
0	0	0	1	0	0	1	0	0	1	0	1	0	1	0	1	0	1	0	1

We need to detect whenever the sequence shows a pattern 1010 taken any consecutive four bits at a time.

1010, Overlapping, Moore:



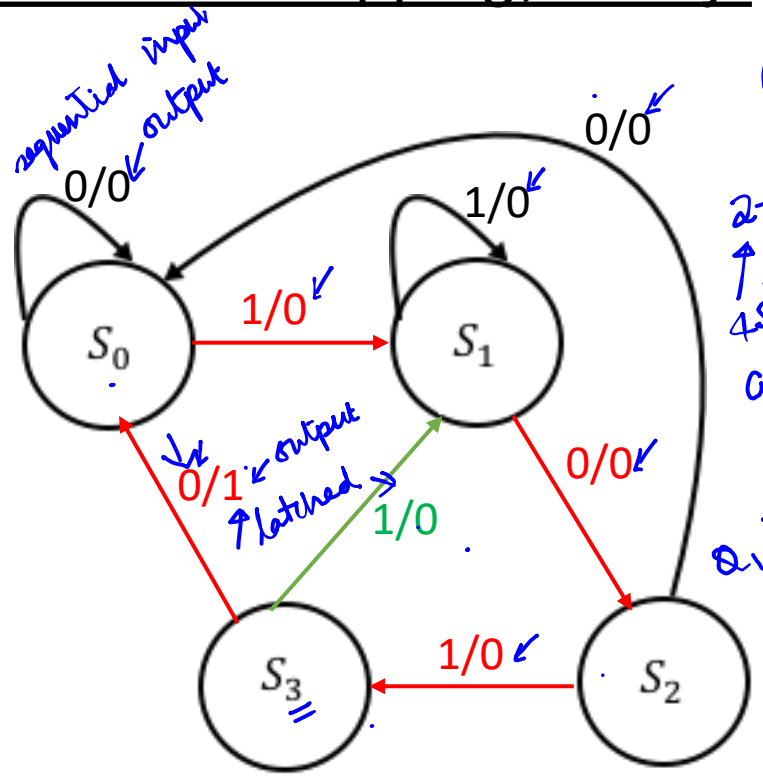
PS	I	NS	Output
S_0	0	S_0	0
S_0	1	S_1	0
S_1	0	S_2	0
S_1	1	S_1	0
S_2	0	S_0	0
S_2	1	S_3	0
S_3	0	S_4	1
S_3	1	S_1	0
S_4	0	S_0	0
S_4	1	S_1	0

$S_0 \rightarrow 000; S_1 \rightarrow 001\ ; S_2 \rightarrow 010\ ; S_3 \rightarrow 011\ ; S_4 \rightarrow 100$

Output $Z = Q_2$

The Hardware Design is left to the class as a HW exercise.

1010, Non-Overlapping, Mealy:



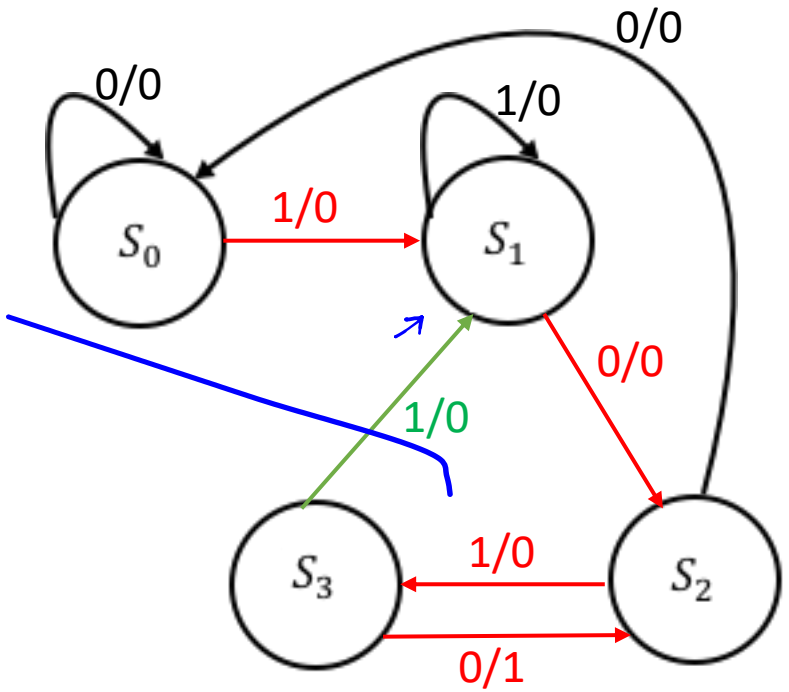
PS	I	NS	output
S_0	0	S_0	0
S_0	1	S_1	0
S_1	0	S_2	0
S_1	1	S_1	0
S_2	0	S_0	0
S_2	1	S_3	0
S_3	0	S_0	1
S_3	1	S_1	0

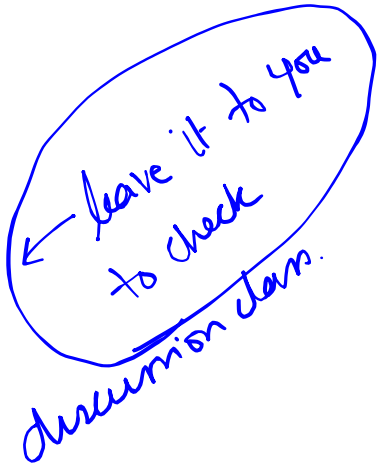
$$S_0 \rightarrow 00; S_1 \rightarrow 01; S_2 \rightarrow 10; S_3 \rightarrow 11;$$

$$Z = \bar{I} Q_1 Q_0$$

The Hardware Design is left to the class as a HW exercise.

1010, Overlapping, Mealy:





PS	I	NS	output
S_0	0	S_0	0
S_0	1	S_1	0
S_1	0	S_2	0
S_1	1	S_1	0
S_2	0	S_0	0
S_2	1	S_3	0
S_3	0	S_0	1
S_3	1	S_1	0

$$S_0 \rightarrow 00; S_1 \rightarrow 01\ ; S_2 \rightarrow 10\ ; S_3 \rightarrow 11\ ;$$

$$Z = \bar{I} Q_1 Q_0$$

The Hardware Design is left to the class as a HW exercise.

Homework:

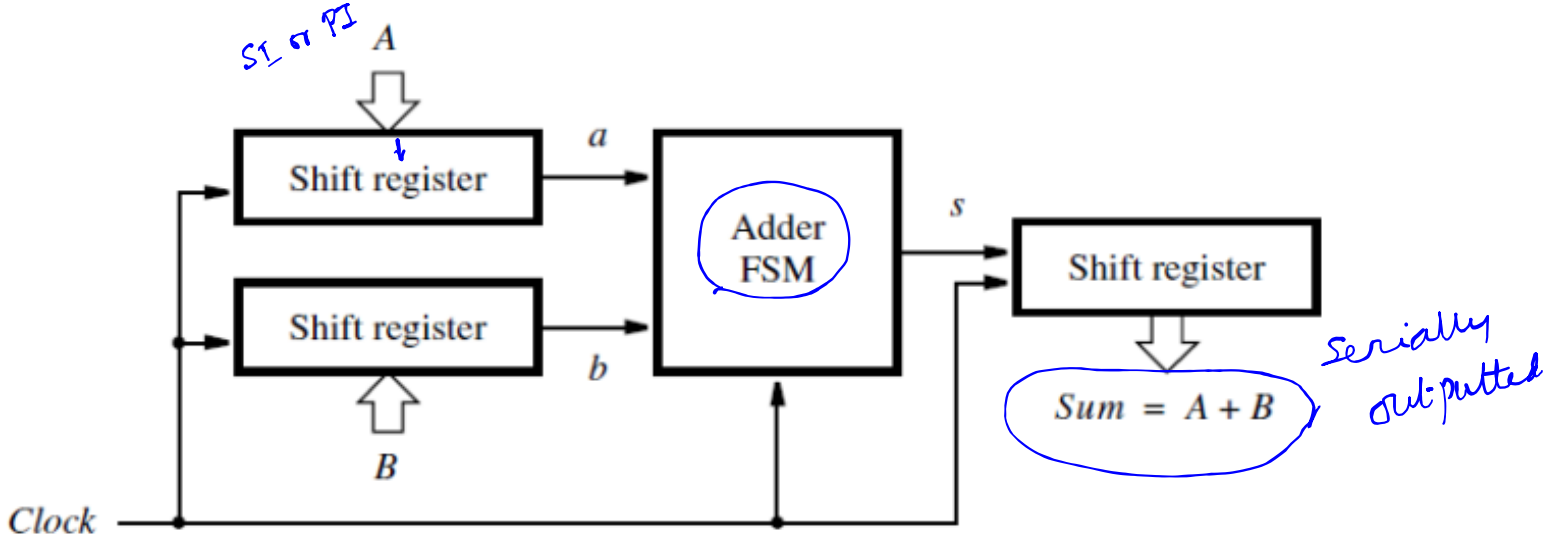
- 1011 -> Overlapping, Mealy
- 1011 -> Overlapping, Moore
- 1011 -> Non-overlapping, Mealy
- 1011 -> Non-overlapping, Moore

Binary Serial Adder:

Let $A = a_{n-1}a_{n-2} \cdots a_0$ and $B = b_{n-1}b_{n-2} \ldots b_0$ be two unsigned numbers that have to be added to produce $Sum = s_{n-1}s_{n-2} \cdots s_0$.

Our task is to design a circuit that will perform serial addition, dealing with a pair of bits in one clock cycle.

The process starts by adding bits a_0 and b_0. In the next clock cycle, bits a_1 and b_1 are added, including a possible carry from the bit-position 0, and so on.



Block diagram of a possible implementation.

Binary Serial Adder:

It includes three shift registers that are used to hold *A*, *B*, and *Sum* as the computation proceeds. Assuming that the input shift registers have parallel-load capability, the addition task begins by loading the values of *A* and *B* into these registers. 1 adder only

Then in each clock cycle, a pair of bits is added by the adder FSM, and at the end of the cycle the resulting sum bit is shifted into the *Sum* register.

We will use positive-edge-triggered flip-flops in which case all changes take place soon after the positive edge of the clock, depending on the propagation delays within the various flip-flops.

At this time the contents of all three shift registers are shifted to the right; this shifts the existing sum bit into *Sum*, and it presents the next pair of input bits a_i and b_i to the adder FSM.

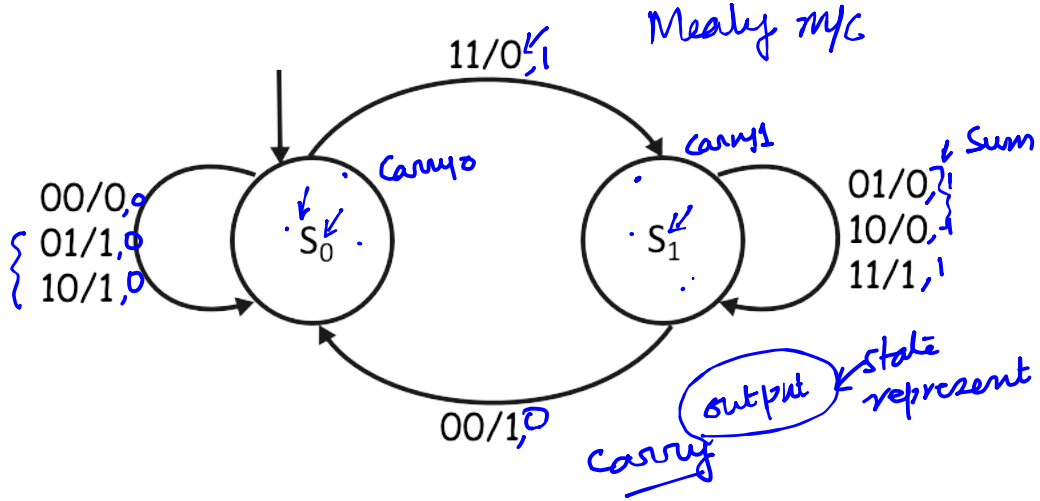
Binary Serial Adder:

State Transition Table for the serial adder FSM.

Present State	Next State					Output s			
	ab	00	01	10	11	00	01	10	11
S_0		S_0	S_0	S_0	S_1	0	1	1	0
S_1		S_0	S_1	S_1	S_1	1	0	0	1

Present State Value, y	Next State value, Y				Output s			
	ab	00	01	10	11	00	01	10	11
0		0	0	0	1	0	1	1	0
1		0	1	1	1	1	0	0	1

State diagram for the serial adder FSM.



S_0 represents the case when carry is 0.

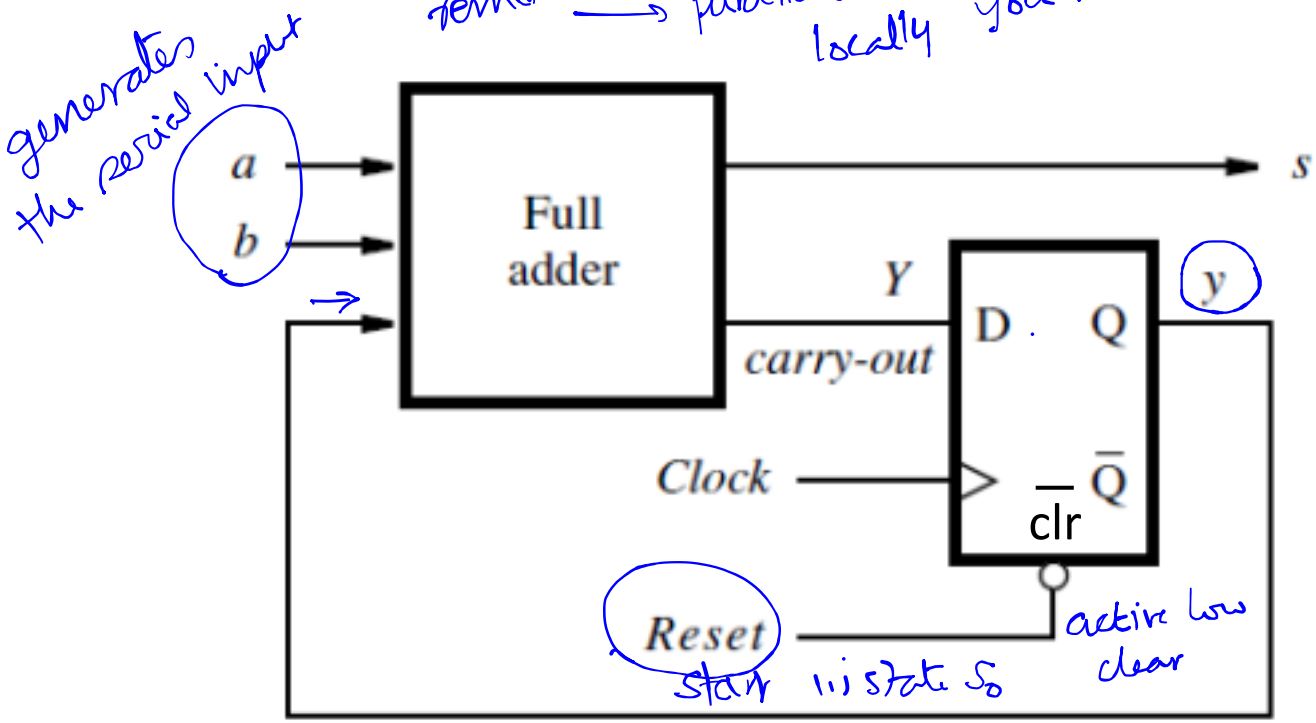
S_1 represents the case when carry is 1.

Let the state assignment for state S_0 be 0.

Let the state assignment for state S_1 be 1.

Use a DFF for design. The state value is given by the Q of the DFF

Binary Serial Adder:

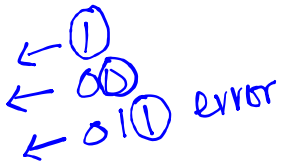


Circuit for the adder FSM

$$Y = ab + ay + by$$

$$s = a \oplus b \oplus y$$

A Tougher Example:



Let us design a Mealy state machine to realize a 3-bit number lock that opens with the code 010. A green light glows when the lock is opened. However, if a wrong code sequence is entered, at any stage, the lock automatically becomes a 4-bit number lock that opens with the code 1001. If an incorrect code sequence is entered, at any stage, when detecting the 4-bit sequence, the lock will go to a permanently locked status turning a red light ON and will not open with any of the codes. .

Number of states required: ???

Lock Operation:

1. Lock in state S_0, it is locked, M = 0 and the lock is in the 3-bit mode. If the first bit entered is 0 go to state S_1, else go to state S_4.
2. Lock in state S_1, and mode is 0. If the next bit entered is 1 go to state S_2, else go to state S_4.
3. Lock in state S_2, and mode is 0. If the next bit entered is 0 go to state S_3, else go to state S_4.
4. In state S_3 the lock is open, O = 1 and the green light turns ON.
5. In state S_4, M = 1 and the lock is in the 4-bit mode. If the first bit entered is 1 go to state S_0, else go to state S_5.
6. In state S_0, M = 1 and the lock is in the 4-bit mode. If the next bit entered is 0 go to state S_1, else go to state S_5.
7. In state S_1, M = 1 and the lock is in the 4-bit mode. If the next bit entered is 0 go to state S_2, else go to state S_5.
8. In state S_2, M = 1 and the lock is in the 4-bit mode. If the next bit entered is 1 go to state S_3, else go to state S_5.
9. In state S_3 the lock is open, O = 1 and the green light turns ON.
10. In state S_5 the lock is locked out, R = 1 and the red light turns ON.

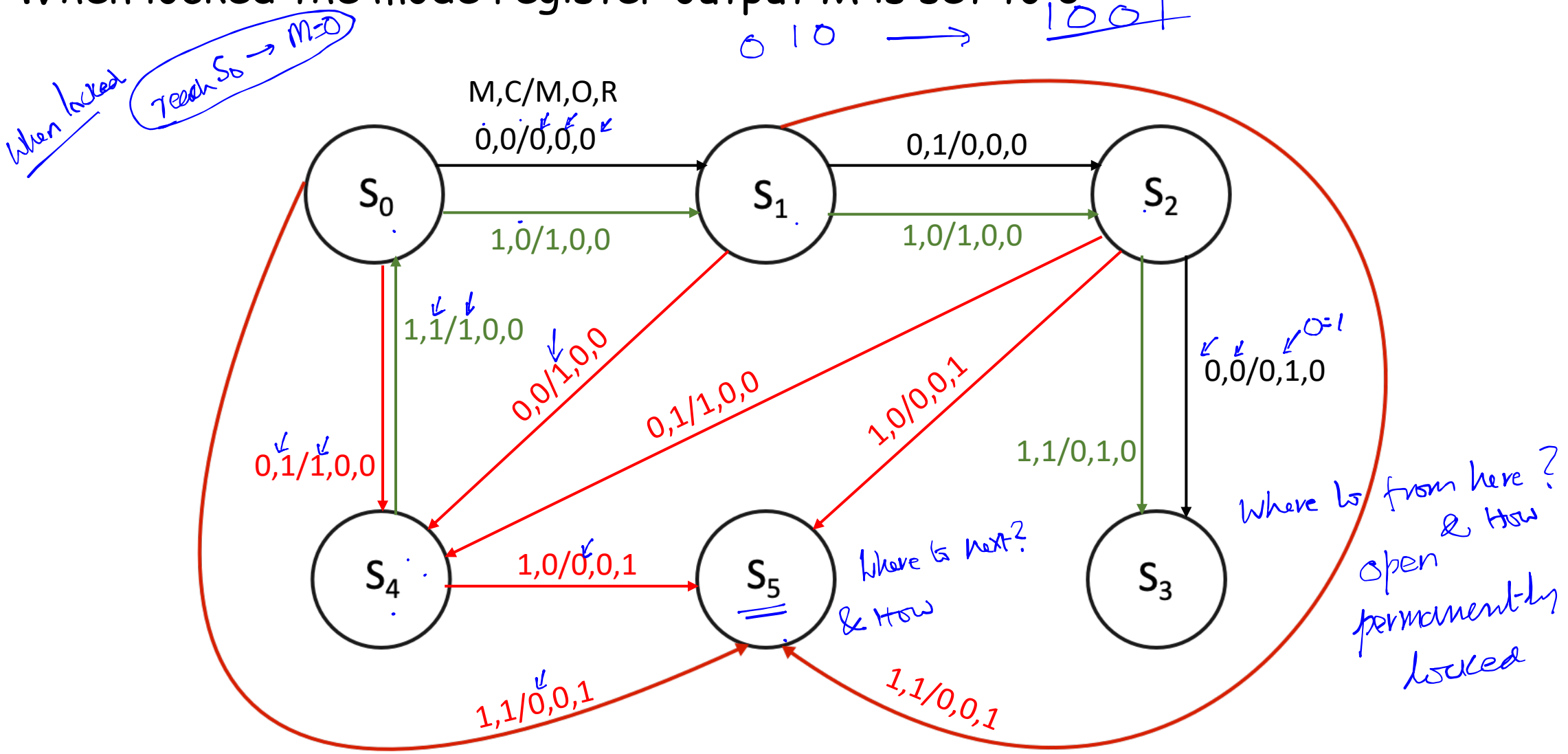
Inputs: Primary input: C --- The Lock code

Secondary input: M --- The mode that determines a 3-bit or 4-bit code. It is generated internally. M = 0 sets in 3 bit mode and M = 1 sets in 4 bit mode.

Outputs: Primary Outputs:

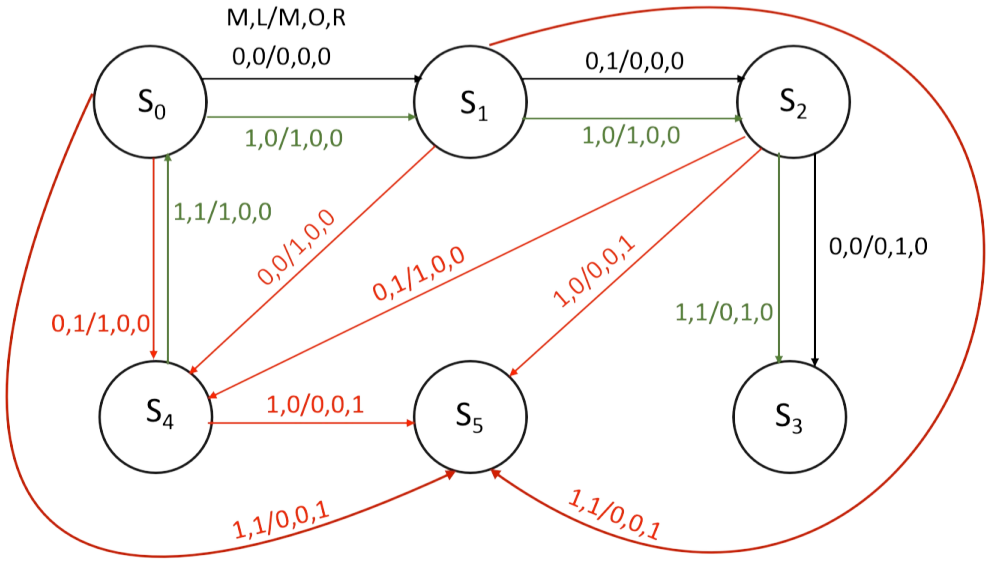
- M --- Mode to determine code length.
  M = 0 3-bit code; M = 1 4-bit code
- O = 1 --- Opens the Lock and turns the Green Light ON
- R = 1 --- Puts the lock in a permanently locked state and turns the Red Light ON

When locked the mode register output M is set to 0



Assign the following values:

$S_0 = 000;\ S_1 = 001;\ S_2 = 010;$
$S_3 = 011;\ S_4 = 100\ \&\ S_5 = 101$



How?

Input		Present State			Next State			Output		
M	C	Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	M	R	O
0	0	0	0	0	0	0	1	0	0	0
0	1	0	0	0	1	0	0	1	0	0
1	0	0	0	0	0	0	1	1	0	0
1	1	0	0	0	1	0	1	0	1	0
0	0	0	0	1	1	0	0	1	0	0
0	1	0	0	1	0	1	0	0	0	0
1	0	0	0	1	0	1	0	0	0	0
1	1	0	0	1	1	0	1	0	1	0
0	0	0	1	0	0	1	1	0	0	1
0	1	0	1	0	1	0	0	1	0	0
1	0	0	1	0	1	0	1	0	1	0
1	1	0	1	0	0	1	1	0	0	1
ϕ	ϕ	0	1	1	0	0	0	0	0	0
ϕ	ϕ	1	0	1	0	0	0	0	0	0

A question yet to be answered: Where does the machine go after the state $S_3 = 011$ and $S_4 = 101$.

We introduce two more variables L(for Lock) applied through a push button and Re(for Reset) to be generated from a remote location. The state diagram will get modified as

In the newly introduced branches the inputs are L and Re