Birla Institute of Technology & Science, Pilani
Hyderabad Campus

Innovate achieve lead

CS/ECE/EEE/INSTR F215:Digital Design

Lecture 26: ***Design of clocked sequential circuits***

Thu, 11 Nov 2021

BITS Pilani
Hyderabad Campus

Dr. R. N. Ponnalagu, EEE

A dream does not become reality
through magic;
it takes sweat, determination
and hard work

~ Colin Powell

Excitation Tables

J	K	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	Q’(t)

Characteristic Table

Q(t)	Q(t+1)	J	K	
0	0	0	X	(0 0) (0 1)
0	1	1	X	(1 0) (1 1)
1	0	X	1	(1 1) (0 1)
1	1	X	0	(1 0) (0 0)

Excitation Table

Design of Clocked sequential Circuits

D	Q(t+1)
0	0
1	1

Characteristic Table

Q(t)	Q(t+1)	D
0	0	0
0	1	1
1	0	0
1	1	1

Excitation Table

T	Q(t+1)
0	Q(t)
1	Q’(t)

Characteristic Table

Q(t)	Q(t+1)	T
0	0	0
0	1	1
1	0	1
1	1	0

Excitation Table

Sequence Detector

- A Sequence Detector (recogniser) look for a specific bit pattern in an input string.

- In the example it has an input line called X. One bit of input is applied on every clock and for example it would take 20 clock cycles to enter a 20 bit string.
- It has one output Z which is 1 when a desired pattern is found.

- To detect a pattern 1001

```
Ex:        Input:     11100110100100110
           Output:    0000010000000100100
```

  One input and one output appear on each clock
- The circuit need to remember bits to recognise a pattern

Step 1: Making a state table/state diagram

- The first thing you have to figure out is precisely how the use of state will help you solve the given problem.
  - Make a state table based on the problem statement. The table should show the present states, inputs, next states and outputs.
  - Sometimes it may be easier to come up with a state diagram first and then convert that to a table.
- This is usually the most difficult step. Once you have the state table, the rest of the design procedure is the same for all sequential circuits.
- Sequence recognizers are one of the harder examples we will see in this class, so if you understand this you're in good shape.

Sequence detector design procedure

States are used to remember meaningful properties of past input sequences that are essential for predicting future output values.

A sequence detector is a sequential circuit that produces a distinct output value whenever a prescribed pattern of input symbols occur in sequence, i.e., detects an input sequence occurrence.

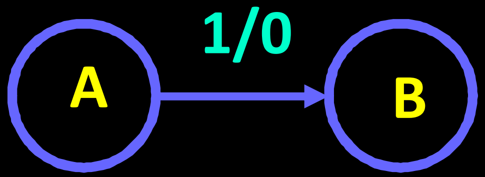
Begin in an initial state in which NONE of the initial portion of the sequence has occurred (reset state)

Add a state that detects that first symbol has occurred

Add states that detect each successive symbol

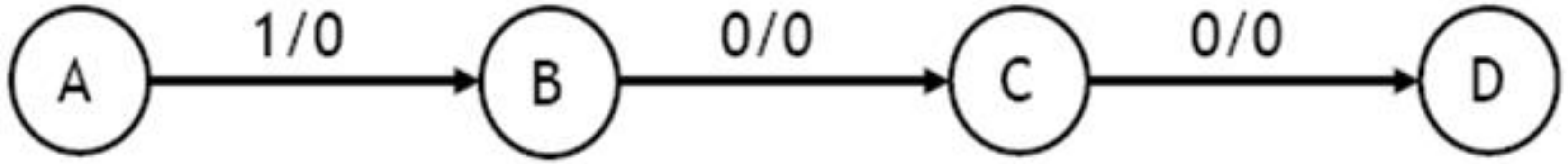
The final state represents the input sequence occurrence and produces the output

- Start in the initial state
  - State ‘A’ is the initial state
  - Add a state ‘B’ that recognizes the first ‘1’
  - State ‘B’ is the state which represents the fact that the first ‘1’ in the input subsequence has occurred. The output symbol ‘0’ means that the full recognized sequence has not yet occurred



$$\begin{tikzcd} A \arrow[r, "1/0"] & B \arrow[r, "0/0"] & C \end{tikzcd}$$

$$\begin{tikzcd} A \arrow[r, "1/0"] & B \arrow[r, "0/0"] & C \arrow[r, "0/0"] & D \end{tikzcd}$$

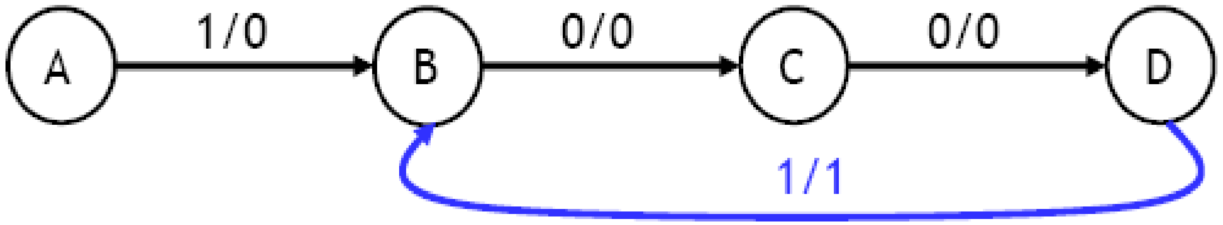


$$\begin{tikzcd} A \arrow[r, "1/0"] & B \arrow[r, "0/0"] & C \arrow[r, "0/0"] & D \end{tikzcd}$$

State	Meaning
A	None of the desired pattern (1001) has been entered yet
B	We've already seen the first bit (1) of the desired pattern
C	We've already seen the first two bits (10) of the desired pattern
D	We've already seen the first three bits (100) of the desired pattern

Overlapping occurrences of the pattern

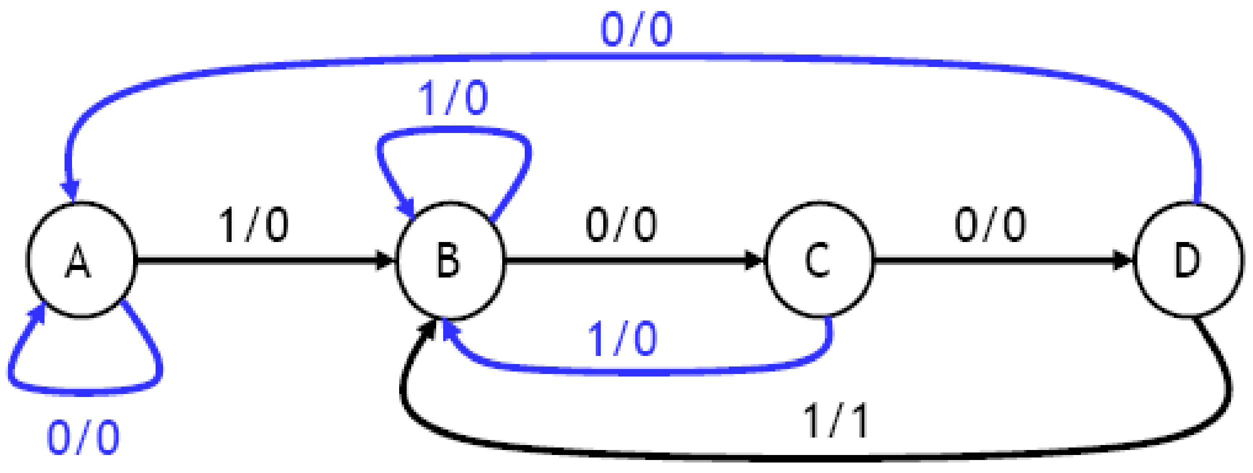
- What happens if we're in state D (the last three inputs were 100), and the current input is 1?
  - The output should be a 1, because we've found the desired pattern.
  - But this last 1 could also be the start of another occurrence of the pattern! For example, 1001001 contains two occurrences of 1001.
  - To properly detect overlapping occurrences of the pattern, the next state should be B.


$$\begin{tikzcd} A \arrow[r, "1/0"] & B \arrow[r, "0/0"] & C \arrow[r, "0/0"] & D \arrow[ll, bend left, "1/1"] \end{tikzcd}$$

State	Meaning
A	None of the desired pattern (1001) has been entered yet
B	We've already seen the first bit (1) of the desired pattern
C	We've already seen the first two bits (10) of the desired pattern
D	We've already seen the first three bits (100) of the desired pattern

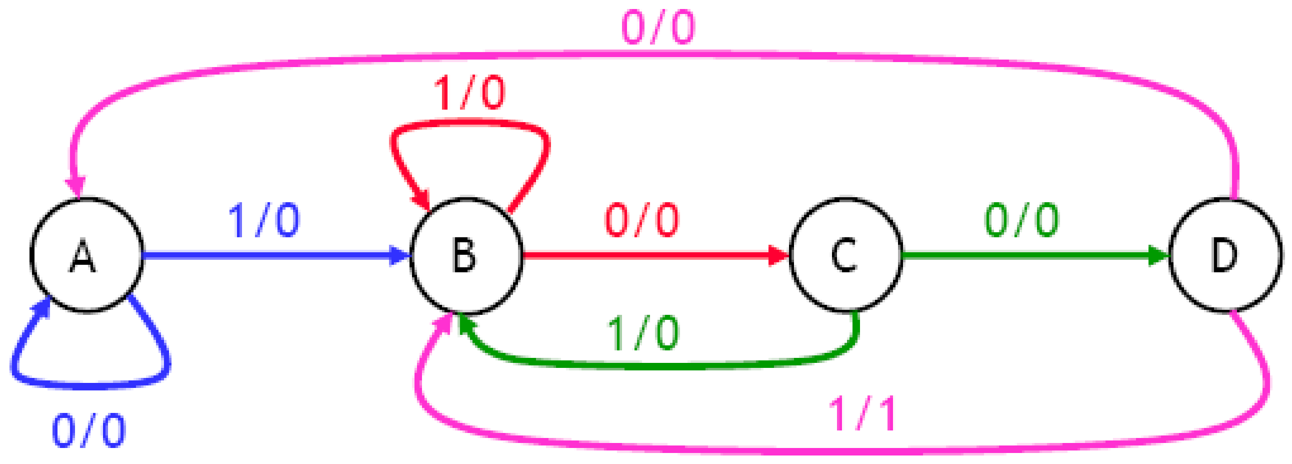
Filling in the other arrows

- Remember that we need *two* outgoing arrows for each node, to account for the two input possibilities of X = 0 and X = 1.
- The remaining arrows we need are shown in blue. They also allow for the correct detection of overlapping occurrences of 1001.

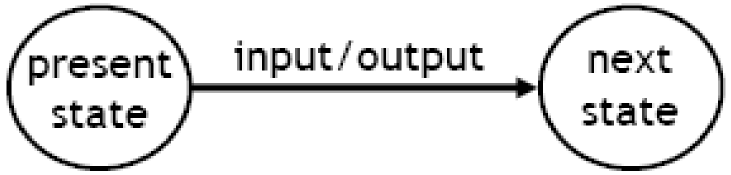


State	Meaning
A	None of the desired pattern (1001) has been entered yet
B	We've already seen the first bit (1) of the desired pattern
C	We've already seen the first two bits (10) of the desired pattern
D	We've already seen the first three bits (100) of the desired pattern

Finally, making the state table



Remember how the state diagram arrows correspond to rows of the state table.



Present State	Input	Next State	Output
A	0	A	0
A	1	B	0
B	0	C	0
B	1	B	0
C	0	D	0
C	1	B	0
D	0	A	0
D	1	B	1

We have four states ABCD, so we need at least two flip-flops Q_1Q_0.

With 2 bits, we can have 4 codes: 00, 01, 10, and 11

How many assignments of 2-bit codes for the 4 states?

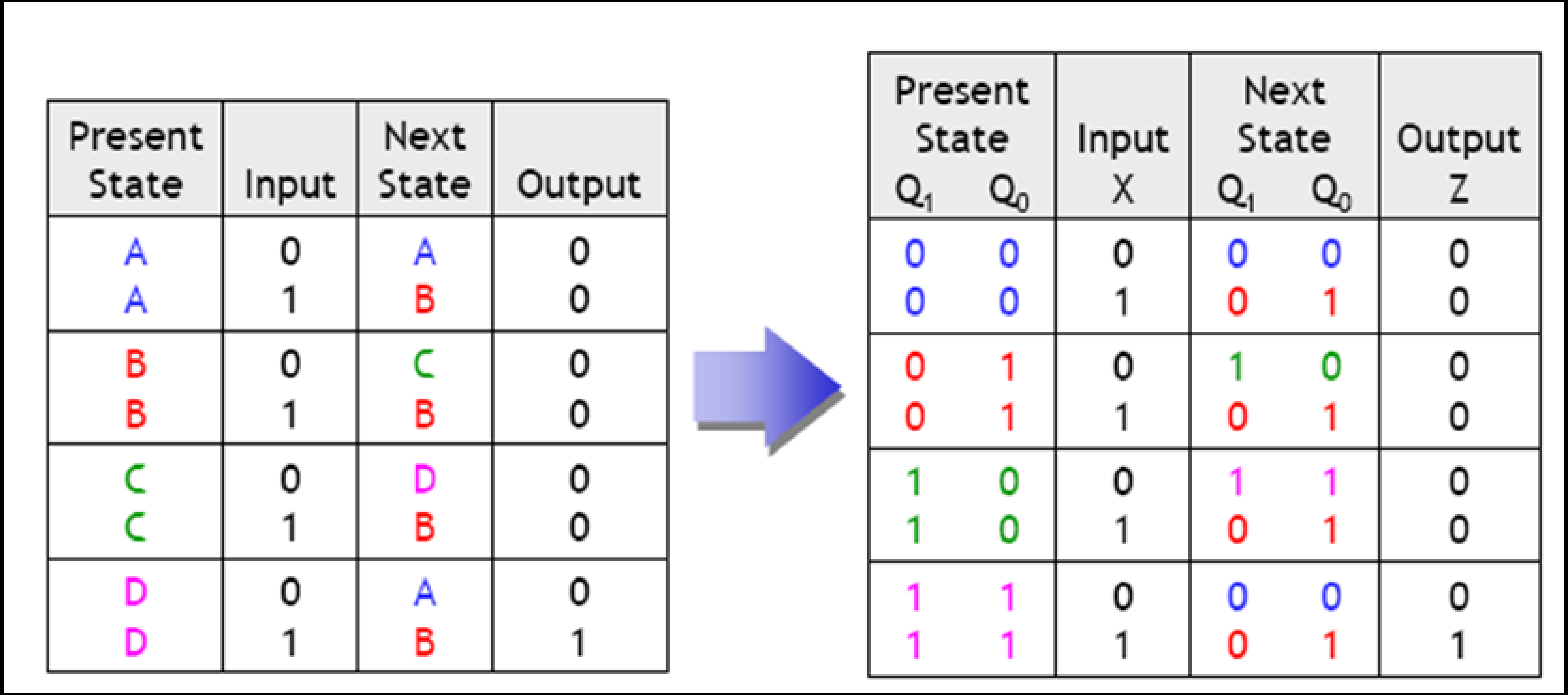
Answer: 4 × 3 × 2 × 1 = 24 possible assignments

The easiest thing is to represent state A with Q_1Q_0 = 00, B with 01, C with 10, and D with 11. (You could have used these codes in Step 1 too, rather than using temporary state labels like ABCD.)

Does code assignment make a difference in cost?

Answer: yes, it affects the cost of the combinational logic

The state assignment can have a big impact on circuit complexity,



Present State	Input	Next State	Output
A	0	A	0
A	1	B	0
B	0	C	0
B	1	B	0
C	0	D	0
C	1	B	0
D	0	A	0
D	1	B	1

Present State		Input	Next State		Output
Q_1	Q_0	X	Q_1	Q_0	Z
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	0	0
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	0	1	0
1	1	0	0	0	0
1	1	1	0	1	1

Step 3: Finding flip-flop input values

- Next we have to figure out how to actually make the flip-flops change from their present state into the desired next state.
- This depends on what kind of flip-flops you use! We'll use two JKs here.
- For each flip-flip Q_i, look at its present and next states, and determine what the inputs J_i and K_i should be in order to make that state change.

Present State		Input	Next State		Flip-flop Inputs				Output
Q_1	Q_0	X	Q_1	Q_0	J_1	K_1	J_0	K_0	Z
0	0	0	0	0					0
0	0	1	0	1					0
0	1	0	1	0					0
0	1	1	0	1					0
1	0	0	1	1					0
1	0	1	0	1					0
1	1	0	0	0					0
1	1	1	0	1					1

JK excitation table

- An excitation table shows what flip-flop inputs are required in order to make a desired state change.

Q(t)	Q(t+1)	J	K	Operation
0	0	0	x	No change/Reset
0	1	1	x	Set/Complement
1	0	x	1	Reset/Complement
1	1	x	0	No change/Set

- This is the same information that's given in the characteristic table, but presented "backwards."

J	K	Q(t+1)	Operation
0	0	Q(t)	No change
0	1	0	Reset
1	0	1	Set
1	1	Q'(t)	Complement

Back to the example

- We can now use the JK excitation table on the right to find the correct values for each flip-flop's inputs, based on its present and next states.

Q(t)	Q(t+1)	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

Present State		Input	Next State		Flip-flop Inputs				Output
Q_1	Q_0	X	Q_1	Q_0	J_1	K_1	J_0	K_0	Z
0	0	0	0	0	0	x	0	x	0
0	0	1	0	1	0	x	1	x	0
0	1	0	1	0	1	x	x	1	0
0	1	1	0	1	0	x	x	0	0
1	0	0	1	1	x	0	1	x	0
1	0	1	0	1	x	1	1	x	0
1	1	0	0	0	x	1	x	1	0
1	1	1	0	1	x	1	x	0	1

J_1

$$\begin{array}{c|cccc}
Q_1 \backslash Q_0X & \bar{Q}_0\bar{X} & \bar{Q}_0X & Q_0X & Q_0\bar{X} \\
\hline
\bar{Q}_1 & 0 & 0 & 0 & 1 \\
Q_1 & X & X & X & X
\end{array}$$

K_1

$$\begin{array}{c|cccc}
Q_1 \backslash Q_0X & & & & \\
\hline
 & X & X & X & X \\
 & 0 & 1 & 1 & 1
\end{array}$$

$K_1 = X + Q_0$

$J_1 = Q_0\bar{X}$

$J_0 = X + Q_1$

$K_0 = \bar{X}$

$Z = Q_1Q_0X$

Step 4: Find equations for the FF inputs and output

- Now you can make K-maps and find equations for each of the four flip-flop inputs, as well as for the output Z.
- These equations are in terms of the present state and the inputs.

Present State		Input	Next State		Flip-flop Inputs				Output
Q_1	Q_0	X	Q_1	Q_0	J_1	K_1	J_0	K_0	Z
0	0	0	0	0	0	x	0	x	0
0	0	1	0	1	0	x	1	x	0
0	1	0	1	0	1	x	x	1	0
0	1	1	0	1	0	x	x	0	0
1	0	0	1	1	x	0	1	x	0
1	0	1	0	1	x	1	1	x	0
1	1	0	0	0	x	1	x	1	0
1	1	1	0	1	x	1	x	0	1

Step 4: Find equations for the FF inputs and output

Use k-Maps to find out flip flop inputs and output Z

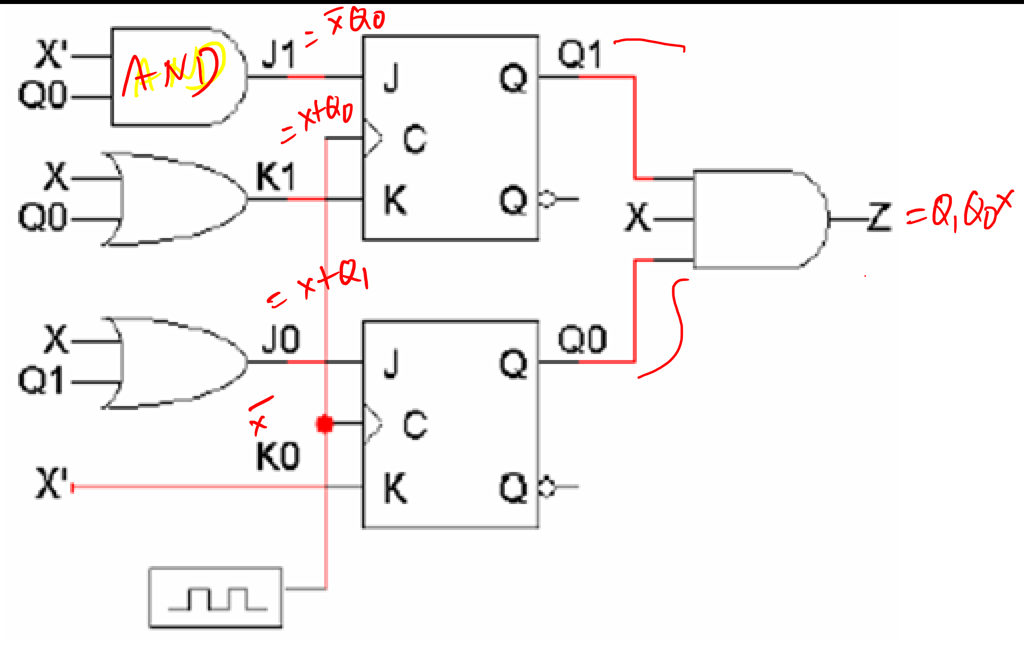
- The advantage of using JK flip-flops is that there are many don't care conditions, which can result in simpler MSP equations.

Present State		Input	Next State		Flip-flop Inputs				Output
Q_1	Q_0	X	Q_1	Q_0	J_1	K_1	J_0	K_0	Z
0	0	0	0	0	0	x	0	x	0
0	0	1	0	1	0	x	1	x	0
0	1	0	1	0	1	x	x	1	0
0	1	1	0	1	0	x	x	0	0
1	0	0	1	1	x	0	1	x	0
1	0	1	0	1	x	1	1	x	0
1	1	0	0	0	x	1	x	1	0
1	1	1	0	1	x	1	x	0	1

$J_1 = X'Q_0$
$K_1 = X + Q_0$

$J_0 = X + Q_1$
$K_0 = X'$

$Z = Q_1Q_0X$



$J_1 = X'Q_0$

$K_1 = X + Q_0$

$J_0 = X + Q_1$

$K_0 = X'$

$Z = Q_1Q_0X$

This is the same circuit which we have seen in analysis

circuit detects occurrences of the pattern 1001 in a serial input stream X.