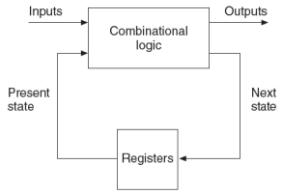
Finite State Machines

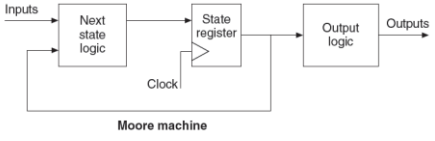
General Sequential System



Models of synchronous sequential systems

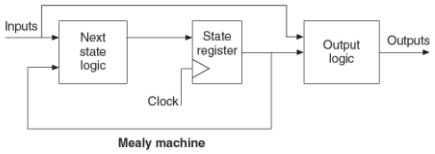
- Two common models of synchronous sequential systems
  - Moore machines
  - Mealy machines

Moore Machine



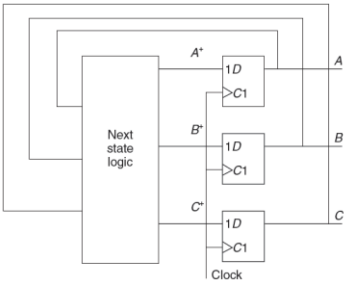
Moore machine

Mealy Machine



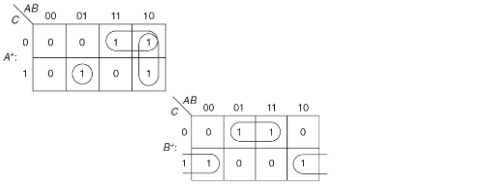
Mealy machine

Design of a three-bit counter



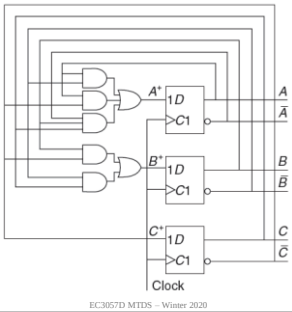
ABC	$A^+B^+C^+$
000	001
001	010
010	011
011	100
100	101
101	110
110	111
111	000

K-maps for 3-bit counter



$$A^+ = A.\overline{C} + A.\overline{B} + \overline{A}.B.C$$
$$B^+ = B.\overline{C} + \overline{B}.C$$
$$C^+ = \overline{C}$$

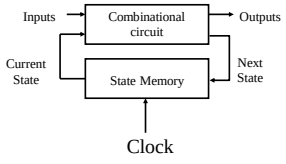
3-bit counter circuit



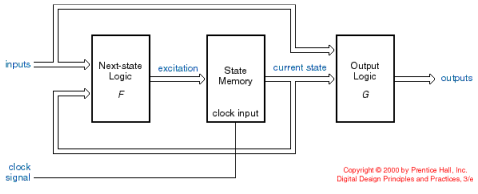
Synchronous Sequential Circuit Analysis

Synchronous Sequential Circuit

- State Memory – A set of *n* edge-triggered flip-flops that store the current state of the machine
  - All flip-flops are triggered from the same master clock signal
  - All flip-flops change state together
- Combinational circuit
  - Next state logic
  - Output logic – Mealy and Moore



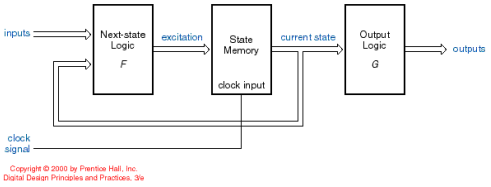
Mealy Model



next state = F (current state, inputs)

outputs = G (current state, inputs)

Moore Model



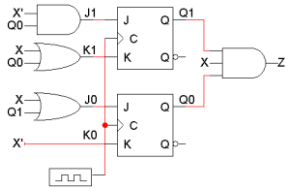
next state = F (current state, inputs)

outputs = G (current state)

Analysis - Goals

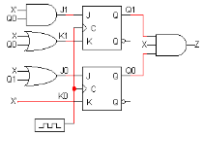
- Characterize as Mealy or Moore machine
- Determine next state equations, i.e., find the function F
  - next state = F (current state, inputs)
- Determine output equations
  - Mealy: outputs = G (current state, inputs), or
  - Moore: outputs = G (current state)
- Express as machine behavior
  - State table, or
  - State diagram
- Formulate English description of machine behavior

An example sequential circuit



- A sequential circuit with two JK flip-flops
- State or memory: Q1Q0
- One input: X; One output: Z

State table of example circuit



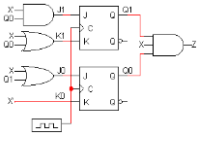
Present State Q_1	Q_0	Inputs X	Next State Q_1	Q_0	Outputs Z
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

Output Equations

- From the diagram, you can see that

$Z = Q_1 Q_0 X$

Mealy model circuit !!!



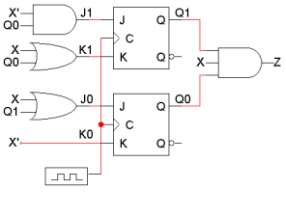
Present State Q_1	Q_0	Inputs X	Next State Q_1	Q_0	Outputs Z
0	0	0			0
0	0	1			0
0	1	0			0
0	1	1			0
1	0	0			0
1	0	1			0
1	1	0			0
1	1	1			1

Next State Equations – Q(t+1)

- Find the flip-flop input equations/excitation equations
- Substitute excitation equations in the flip-flop's characteristic equation

J1 = X' Q0
K1 = X + Q0

J0 = X + Q1
K0 = X'



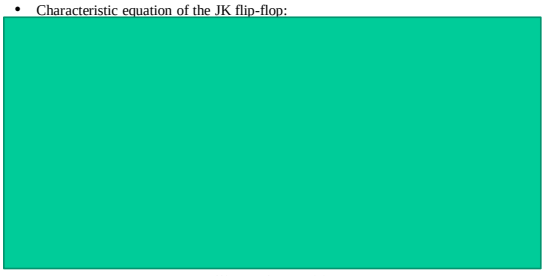
Next State Equations – Q(t+1)

- Excitation equations:

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

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

- Characteristic equation of the JK flip-flop:



Next State Equations – Q(t+1)

- Excitation equations:

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

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

- Characteristic equation of the JK flip-flop:

  $Q(t+1) = K'Q(t) + JQ'(t)$

- Next state equations:

Next State Equations – Q(t+1)

- Excitation equations:

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

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

- Characteristic equation of the JK flip-flop:

  $Q(t+1) = K'Q(t) + JQ'(t)$

- Next state equations:

$Q_1(t+1) = K_1'Q_1(t) + J_1Q_1'(t)$

$= (X + Q_0(t))' Q_1(t) + X' Q_0(t) Q_1'(t)$

$= X' (Q_0(t)' Q_1(t) + Q_0(t) Q_1(t)')$

$= X' (Q_0(t) \oplus Q_1(t))$

$Q_0(t+1) = K_0'Q_0(t) + J_0Q_0'(t)$

$= X Q_0(t) + (X + Q_1(t)) Q_0'(t)$

$= X + Q_0(t)' Q_1(t)$

State Table & Next State Equations

- $Q_1(t+1) = X' (Q_0(t) \oplus Q_1(t))$
  - $Q_1=0, Q_0=0, X= 0 \Rightarrow Q_1(t+1)= 0$
- $Q_0(t+1) = X + Q_0(t)' Q_1(t)$
  - $Q_1=0, Q_0=0, X= 0 \Rightarrow Q_0(t+1)= 0$

Present State Q_1	Present State Q_0	Inputs X	Next State Q_1	Next State Q_0	Outputs Z
0	0	0	0	0	0
0	0	1			0
0	1	0			0
0	1	1			0
1	0	0			0
1	0	1			0
1	1	0			0
1	1	1			1

State Table & Next State Equations

- $Q_1(t+1) = X' (Q_0(t) \oplus Q_1(t))$
  - $Q_1=0, Q_0=1, X= 1 \Rightarrow Q_1(t+1)= 0$
- $Q_0(t+1) = X + Q_0(t)' Q_1(t)$
  - $Q_1=0, Q_0=1, X= 1 \Rightarrow Q_0(t+1)= 1$

Present State Q_1	Present State Q_0	Inputs X	Next State Q_1	Next State Q_0	Outputs Z
0	0	0	0	0	0
0	0	1			0
0	1	0			0
0	1	1	0	1	0
1	0	0			0
1	0	1			0
1	1	0			0
1	1	1			1

State Table & Next State Equations

- $Q_1(t+1) = X' (Q_0(t) \oplus Q_1(t))$
- $Q_0(t+1) = X + Q_0(t)' Q_1(t)$

Present State Q_1	Present State Q_0	Inputs X	Next State Q_1	Next State Q_0	Outputs 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

State Table & Characteristic Table

- The general JK flip-flop characteristic equation is:

$$Q(t+1) = K'Q(t) + JQ'(t)$$

- We can also determine the next state for each input/current state combination directly from the characteristic table

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

State Table & Characteristic Table

- With these equations, we can make a table showing J_1, K_1, J_0 and K_0 for the different combinations of present state Q_1Q_0 and input X

$J_1 = X' Q_0$ $\quad$ $J_0 = X + Q_1$

$K_1 = X + Q_0$ $\quad$ $K_0 = X'$

Present State Q_1	Present State Q_0	Inputs X	Flip-flop Inputs J_1	K_1	J_0	K_0
0	0	0	0	0	0	1
0	0	1	0	1	1	0
0	1	0	1	1	0	1
0	1	1	0	1	1	0
1	0	0	0	0	1	1
1	0	1	0	1	1	0
1	1	0	1	1	1	1
1	1	1	0	1	1	0

State Table & Characteristic Table

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

Present State Q_1	Present State Q_0	Inputs X	FF Inputs J_1	K_1	J_0	K_0	Next State Q_1	Next State Q_0
0	0	0	0	0	0	1		
0	0	1	0	1	1	0		
0	1	0	1	1	0	1	1	
0	1	1	0	1	1	0		
1	0	0	0	0	1	1		
1	0	1	0	1	1	0		
1	1	0	1	1	1	1		
1	1	1	0	1	1	0		

State Table & Characteristic Table

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

Present State Q_1	Present State Q_0	Inputs X	FF Inputs J_1	K_1	J_0	K_0	Next State Q_1	Next State Q_0
0	0	0	0	0	0	1		
0	0	1	0	1	1	0		
0	1	0	1	1	0	1	1	0
0	1	1	0	1	1	0		
1	0	0	0	0	1	1		
1	0	1	0	1	1	0		
1	1	0	1	1	1	1		
1	1	1	0	1	1	0		

A different look

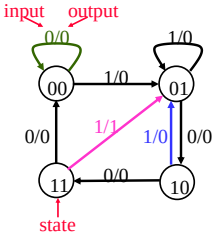
Present State Q_1	Present State Q_0	Inputs X	Next State Q_1	Next State Q_0	Outputs Z
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	0	0
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	0	1	0
1	1	0	0	0	0
1	1	1	0	1	1

Present State Q1	Present State Q0	Next State Input X= 0		Next State Input X= 1		Output Z X= 0	Output Z X= 1
0	0	0	0	0	1	0	0
0	1	1	0	0	1	0	0
1	0	1	1	0	1	0	0
1	1	0	0	0	1	0	1

State diagrams (Mealy model)

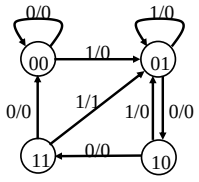
- We can also represent the state table graphically with a state diagram
- A diagram corresponding to our example state table is shown below

Present State Q_1	Present State Q_0	Inputs X	Next State Q_1	Next State Q_0	Outputs 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

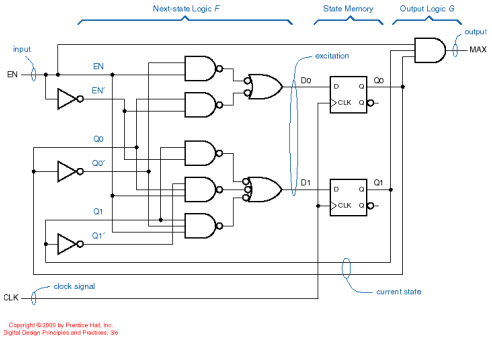


Sizes of state diagrams

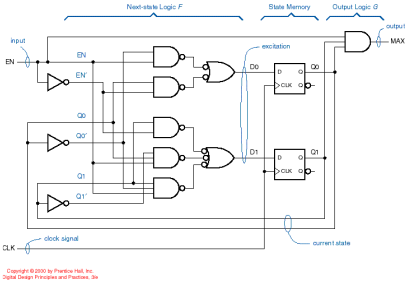
- Always check the size of your state diagrams
  - If there are *n* flip-flops, there should be 2^n nodes in the diagram
  - If there are *m* inputs, then each node will have 2^m outgoing arrows
- In our example,
  - We have two flip-flops, and thus four states or nodes.
  - There is one input, so each node has two outgoing arrows.



Another Mealy Circuit

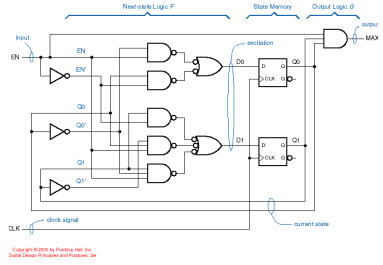


Excitation Equations



- $D_0 = EN' Q_0 + EN Q_0'$
- $D_1 = EN' Q_1 + EN Q_1' Q_0 + EN Q_1 Q_0'$

Next State/Output Equations



- $Q_0(t+1) = D_0 = EN' Q_0 + EN Q_0'$
- $Q_1(t+1) = D_1 = EN' Q_1 + EN Q_1' Q_0 + EN Q_1 Q_0'$
- $MAX = EN Q_1 Q_0$

Mealy State Table

- $Q_0(t+1) = D_0 = EN' Q_0 + EN Q_0'$
- $Q_1(t+1) = D_1 = EN' Q_1 + EN Q_1' Q_0 + EN Q_1 Q_0'$
- $MAX = EN Q_1 Q_0$

Present State Q1	Present State Q0	Next State Input EN= 0		Next State Input EN= 1		Output MAX EN= 0	Output MAX EN= 1
0	0	0	0	0	1	0	0
0	1	0	1	1	0	0	0
1	0	1	0	1	1	0	0
1	1	1	1	0	0	0	1

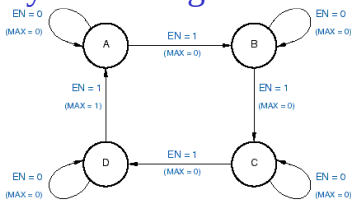
Mealy State Table

Present State Q1	Present State Q0	Next State Input EN= 0		Next State Input EN= 1		Output MAX EN= 0	Output MAX EN= 1
0	0	0	0	0	1	0	0
0	1	0	1	1	0	0	0
1	0	1	0	1	1	0	0
1	1	1	1	0	0	0	1

State Q1	State Q0	State Name
0	0	A
0	1	B
1	0	C
1	1	D

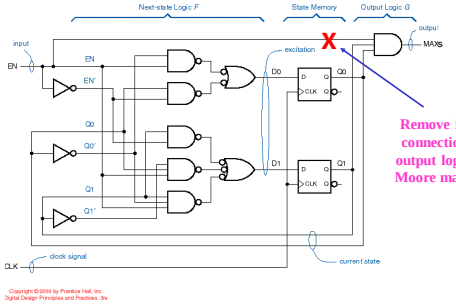
Present State Q1 Q0	Next State Input EN= 0	Next State Input EN= 1	Output MAX EN= 0	Output MAX EN= 1
A	A	B	0	0
B	B	C	0	0
C	C	D	0	0
D	D	A	0	1

Mealy State Diagram



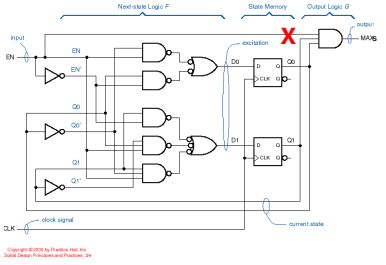
Present State Q1 Q0	Next State Input EN= 0	Next State Input EN= 1	Output MAX EN= 0	Output MAX EN= 1
A	A	B	0	0
B	B	C	0	0
C	C	D	0	0
D	D	A	0	1

Moore Circuit



Remove input connection to output logic => Moore machine

Next State/Output Equations



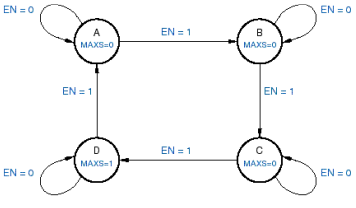
- $Q_0(t+1) = D_0 = EN' Q_0 + EN Q_0'$
- $Q_1(t+1) = D_1 = EN' Q_1 + EN Q_1' Q_0 + EN Q_1 Q_0'$
- $MAXS = Q_1 Q_0$

Moore State Table

- $Q_0(t+1) = D_0 = EN' Q_0 + EN Q_0'$
- $Q_1(t+1) = D_1 = EN' Q_1 + EN Q_1' Q_0 + EN Q_1 Q_0'$
- $MAXS = Q_1 Q_0$

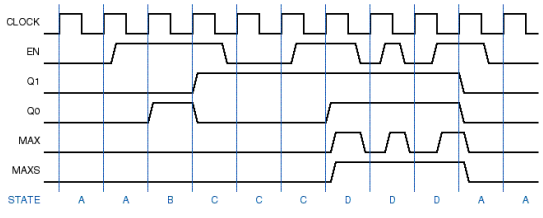
Present State Q1	Present State Q0	Next State Input EN= 0		Next State Input EN= 1		Output MAXS
0	0	0	0	0	1	0
0	1	0	1	1	0	0
1	0	1	0	1	1	0
1	1	1	1	0	0	1

Moore State Diagram



Present State Q1	Present State Q0	Next State Input EN= 0		Next State Input EN= 1		Output MAXS
0	0	0	0	0	1	0
0	1	0	1	1	0	0
1	0	1	0	1	1	0
1	1	1	1	0	0	1

State Transitions



- MAX : Output of the Mealy circuit
- MAXS : Output of the Moore circuit

Sequential circuit analysis summary

- To analyze sequential circuits, you have to:
  - Find Boolean expressions for the outputs of the circuit and the flip-flop inputs
  - Use these expressions to fill in the output and flip-flop input columns in the state table
  - Finally, use the characteristic equation or characteristic table of the flip-flop to fill in the next state columns.
- The result of sequential circuit analysis is a state table or a state diagram describing the circuit

Design of Sequence Detector

Sequence Detector

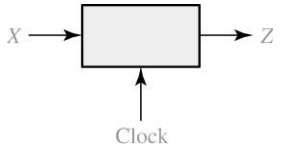
- A circuit that detects the occurrence of a particular pattern on its input is referred to as a sequence detector.

Design a circuit that examine a string of 0's and 1's applied to the input X and for any input sequence ending in 101 will produce an output Z=1 coincident with the last 1.

The circuit does not reset when a 1 output occur.

We assume that the input X can only change between clock pulses

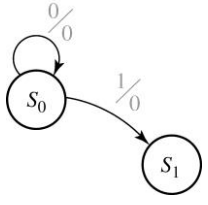
101 Sequence Detector



X =	**0**	**0**	**1**	**1**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**0**
Z =	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**1**	**0**	**0**
(time:	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15)**

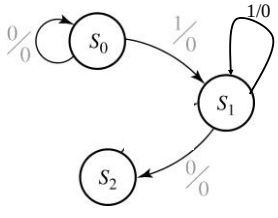
Design of 101 Sequence Detector

- State Diagram:



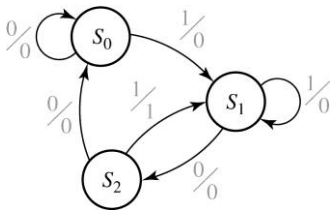
Design of 101 Sequence Detector

- State Diagram:



Design of 101 Sequence Detector

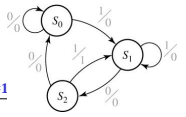
- State Diagram (final):



Design of 101 Sequence Detector

- State Table:

Present state	Next State X = 0	Next State X = 1	Present Output X = 0	Present Output X =1
S_0	S_0	S_1	0	0
S_1	S_2	S_1	0	0
S_2	S_0	S_1	0	1



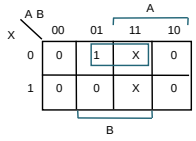
Assign a unique binary code to each state name (State Assignment)

- State Table with State Assignment:

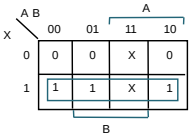
AB	$D_A D_B$ = A^+B^+ X = 0	$D_A D_B$ = A^+B^+ X = 1	Z X = 0	Z X =1
00	00	01	0	0
01	10	01	0	0
10	00	01	0	1

Design of Sequence Detector

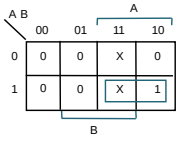
- Derive Boolean Equations:



$D_A = X'.B$

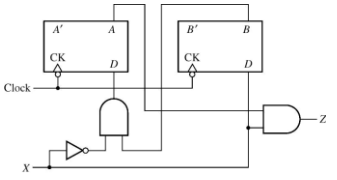


$D_B = X$

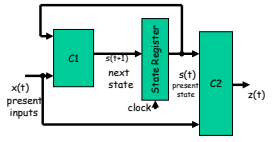


$Z = X.A$

Design of Sequence Detector



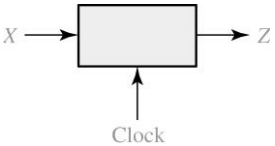
Compare with Typical Mealy Machine



Design of Sequence Detector

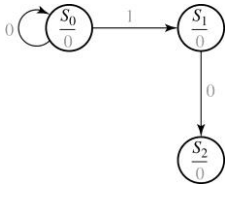
- A Moore Sequence Detector:

101 sequence Detector

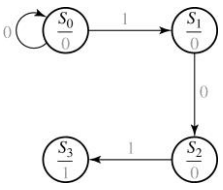


X =	0	0	1	1	0	1	1	0	0	1	0	1	0	1	0	0
Z =	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0
(time:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15)

Design of a Sequence Detector



Design of a Sequence Detector



S_0: start

S_1: got 1

S_2: got 10

S_3: got 101

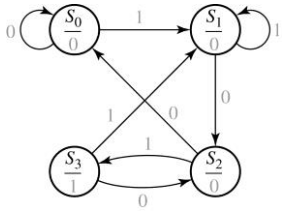
Design of a Sequence Detector

S_0: start

S_1: got 1

S_2: got 10

S_3: got 101



Design of a Sequence Detector

State Table

Present state	Next State X = 0	Next State X = 1	Present Output (Z)
S_0	S_0	S_1	0
S_1	S_2	S_1	0
S_2	S_0	S_3	0
S_3	S_2	S_1	1

Transition Table with State assignment

AB	A^+B^+ X = 0	A^+B^+ X = 1	Z
00	00	01	0
01	11	01	0
11	00	10	0
10	11	01	1

State Diagram Development

- To develop a sequence detector state diagram:
  1. Construct some sample input and output sequences to make sure that you understand the problem statement.
  2. Begin in an initial state in which NONE of the initial portion of the sequence has occurred (typically "reset" state).
  3. Add a state that recognizes that the first symbol has occurred.
  4. Add states that recognize each successive symbol occurring.
  5. Each time you add an arrow to the state graph, determine it can go to one of the previously defined states or whether a new state must be added
  6. The final state represents the input sequence occurrence.
  7. Add state transition arcs which specify what happens when a symbol *not* in the proper sequence has occurred.
  8. Check your state graph for completeness and non-redundant arcs.
  9. When your state graph is complete, test it by applying the input sequences formulated in part1 and making sure the output sequences are correct

Sequential circuit design procedure

Step 1:
Make a state table based on the problem statement. The table should show the present states, inputs, next states and outputs. (It may be easier to find a state diagram first, and then convert that to a table)

Step 2:
Assign binary codes to the states in the state table, if you haven't already. If you have n states, your binary codes will have at least $\lceil \log_2 n \rceil$ digits, and your circuit will have at least $\lceil \log_2 n \rceil$ flip-flops

Step 3:
For each flip-flop and each row of your state table, find the flip-flop input values that are needed to generate the next state from the present state. You can use flip-flop excitation tables here.

Step 4:
Find simplified equations for the flip-flop inputs and the outputs.

Step 5:
Build the circuit!

Another Example

Sequence detector (Mealy)

- A sequence detector is a special kind of sequential circuit that looks for a special bit pattern in some input
- The detector circuit has only one input, X
  - One bit of input is supplied on every clock cycle
  - This is an easy way to permit arbitrarily long input sequences
- There is one output, Z, which is 1 when the desired pattern is found
- Our example will detect the bit pattern "1001":

  Inputs: 1 1 1 0 0 1 1 0 1 0 0 1 0 0 1 1 0 …
  Outputs: 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0 0 …

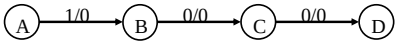
- A sequential circuit is required because the circuit has to "remember" the inputs from previous clock cycles, in order to determine whether or not a match was found

Step 1: Making a state table

- 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 is easier to first find a state diagram 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

A basic Mealy state diagram

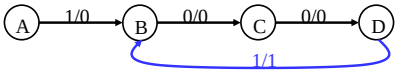
- What state do we need for the sequence detector?
  - We have to "remember" inputs from previous clock cycles
  - For example, if the previous three inputs were 100 and the current input is 1, then the output should be 1
  - In general, we will have to remember occurrences of parts of the desired pattern—in this case, 1, 10, and 100
- We'll start with a basic state diagram:



State	Meaning
A	None of the desired pattern (1001) has been input 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

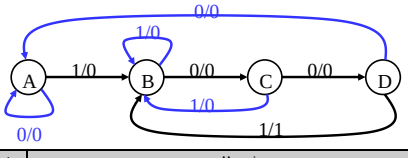
- What happens if we're in state D (the last three inputs were 100), and the current input is 1?
  - The output should be a 1, because we've found the desired pattern
  - But this last 1 could also be the start of another occurrence of the pattern! For example, 1001001 contains *two* occurrences of 1001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



State	Meaning
A	None of the desired pattern (1001) has been input 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.

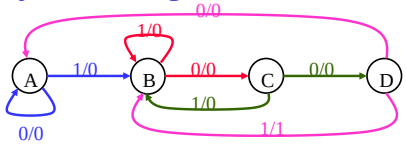
Filling in the other arrows

- *Two* outgoing arrows for each node, to account for the 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 input 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.

Mealy state diagram & 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

Step 2: Assigning binary codes to states

- We have four states ABCD, so we need at least two flip-flops Q_1Q_0
- The easiest thing to do is represent state A with Q_1Q_0 = 00, B with 01, C with 10, and D with 11
- The state assignment can have a big impact on circuit complexity, but we won't worry about that too much in this class

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 Q_1	Q_0	Input X	Next State Q_1	Q_0	Output 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. 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 Q_1	Q_0	Input X	Next State Q_1	Q_0	Flip flop inputs J_1	K_1	J_0	K_0	Output 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

- 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 Q_1	Q_0	Input X	Next State Q_1	Q_0	Flip flop inputs J_1	K_1	J_0	K_0	Output 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

- 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
- 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 Q_1	Q_0	Input X	Next State Q_1	Q_0	Flip flop inputs J_1	K_1	J_0	K_0	Output 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

FF input equations

Present State Q_1	Q_0	Input X	Next State Q_1	Q_0	Flip flop inputs J_1	K_1	J_0	K_0	Output 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 K-map (Q_1Q_0 columns 00, 01, 11, 10; X rows 0, 1):

X	00	01	11	10
0	0	1	x	x
1	0	0	x	x

$J_1 = X'Q_0$

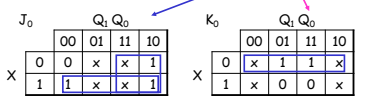
K_1 K-map (Q_1Q_0 columns 00, 01, 11, 10; X rows 0, 1):

X	00	01	11	10
0	x	x	1	0
1	x	x	1	1

$K_1 = X + Q_0$

FF input equations

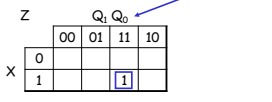
Present State Q_1	Present State Q_0	Input X	Next State Q_1	Next State Q_0	J_1	K_1	J_0	K_0	Output Z
0	0	0	0	0	0	x	0	x	0
0	0	1	0	1	0	x	1	x	0
0	1	0	1	0	1	x	x	1	0
0	1	1	0	1	0	x	x	0	0
1	0	0	1	1	x	0	1	x	0
1	0	1	0	1	x	1	1	x	0
1	1	0	0	0	x	1	x	1	0
1	1	1	0	1	x	1	x	0	1



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

Output equation

Present State Q_1	Present State Q_0	Input X	Next State Q_1	Next State Q_0	J_1	K_1	J_0	K_0	Output 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



$Z = X\,Q_1\,Q_0$

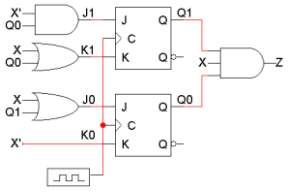
Step 5: Build the circuit

- Lastly, we use these simplified equations to build the completed circuit

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

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

$Z = Q_1 Q_0 X$



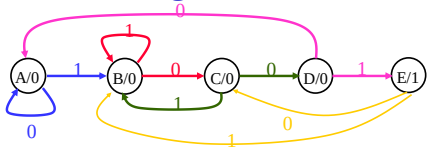
Sequence detector (Moore)

- A sequence detector is a special kind of sequential circuit that looks for a special bit pattern in some input
- The detector circuit has only one input, X
  - One bit of input is supplied on every clock cycle
  - This is an easy way to permit arbitrarily long input sequences
- There is one output, Z, which is 1 when the desired pattern is found
- Our example will detect the bit pattern "1001":

  Inputs: 1 1 1 0 0 1 1 0 1 0 0 1 0 0 1 1 0 …
  Outputs: 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0 0 …

- A sequential circuit is required because the circuit has to "remember" the inputs from previous clock cycles, in order to determine whether or not a match was found

Moore state diagram & 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	E	0
E	0	C	1
E	1	B	1

Circuit design is left as an exercise !

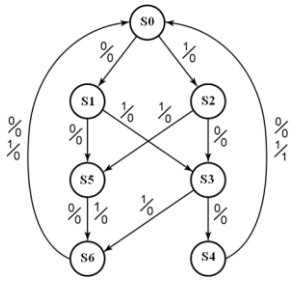
Comparison of Mealy and Moore FSM

- **Mealy** machines have **less states**
  - outputs are on transitions (n^2) rather than states (n)
- **Moore** machines are **safer** to use
  - outputs change at clock edge (always one cycle later)
  - in Mealy machines, input change can cause output change as soon as logic is done – a big problem when two machines are interconnected – asynchronous feedback may occur if one isn't careful
- **Mealy** machines **react faster** to inputs
  - react in same cycle – don't need to wait for clock
  - outputs may be considerably shorter than the clock cycle
  - in Moore machines, more logic may be necessary to decode state into outputs – there may be more gate delays after clock edge

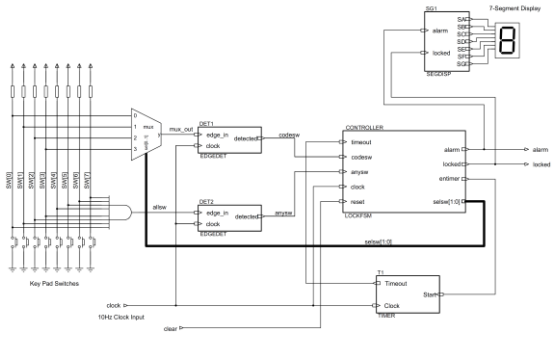
Example: Sequence Detector

A sequential circuit has one input (X) and one output (Z). The circuit examines groups of four consecutive inputs and produces an output $Z = 1$ if the input sequence 0101 or 1001 occurs. The circuit resets after every four inputs. Find a Mealy state graph. A typical input and output sequence is:

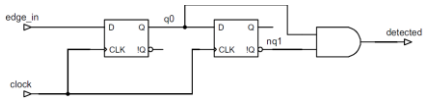
X =	0101	0010	1001	0100
Z =	0001	0000	0001	0000



Combination Lock FSM with Automatic Lock Feature



Logic diagram of edge detector `edgedet`




```
1  module edgedet(input edge_in,
2                 output detected,
3                 input clock);

4  wire q0, q1;

5  dff dff0(.q(q0), .d(edge_in), .clk(clock));
6  dff dff1(.q(q1), .d(q0), .clk(clock));

7  assign detected = q0 & ~q1;

8  endmodule
```

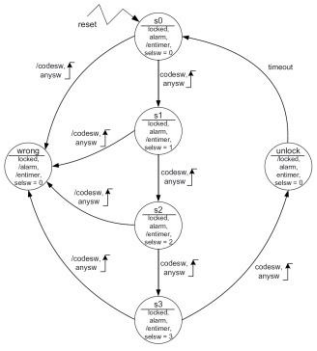

Timer

```
1  module Timer(input Clock, Start, output Timeout);
2  //time delay value in clk pulses
3  localparam NUMCLKS = 300;

4  reg[ 8:0] q;
5  always @(posedge Clock)
6  begin
7    if (!Start||(q == NUMCLKS))
8      q <= 9'b0;
9    else
10     q <= q + 1;
11 end
12 //decode counter output
13 assign Timeout = (q == NUMCLKS);

14 endmodule
```


Seven Segment Display

```
1  module segdisp(input locked,alarm,
2                 output SA,SB,SC,SD,SE,SF,SG);

3  reg[ 6:0] seg;

4  always @(locked or alarm)
5  begin
6    if (alarm == 0)
7      seg = 7'b0001000; //display 'A'
8    else if (locked == 0)
9      seg = 7'b1000001; //display 'U'
10   else
11     seg = 7'b1110001; //display 'L'
12 end

13 assign{SA, SB, SC, SD, SE, SF, SG} = seg;

14 endmodule
```

FSM




```
1  module lockfsm(input clock, reset,
2              codesw, anysw,
3              output reg [1:0] selsw,
4              output locked, alarm, entimer,
5              input timeout);

6  localparam s0=3'b000, s1=3'b001, s2=3'b010,
7              s3=3'b011,
8              wrong=3'b100, unlock=3'b101;

9  reg [2:0] lockstate;

10 always @(posedge clock or posedge reset)
11 begin
12 if (reset == 1'b1)
13   lockstate <= s0;
14 else
15   case (lockstate)
16     s0 : if (anysw & codesw)
17        lockstate <= s1;
18      else if (anysw)
19        lockstate <= wrong;
20      else
21        lockstate <= s0;
22     s1 : if (anysw & codesw)
23        lockstate <= s2;
24      else if (anysw)
25        lockstate <= wrong;
26      else
27        lockstate <= s1;
28     s2: if (anysw & codesw)
29        lockstate <= s3;
30      else if (anysw)
31        lockstate <= wrong;
```



```
32      else
33        lockstate <= s2;
34     s3: if (anysw & codesw)
35        lockstate <= unlock;
36      else if (anysw)
37        lockstate <= wrong;
38      else
39        lockstate <= s3;
40     wrong: lockstate <= wrong;
41     unlock: if (timeout)
42        lockstate <= s0;
43      else
44        lockstate <= unlock;
45    default: lockstate <= 3'bx;
46   endcase
47 end
```



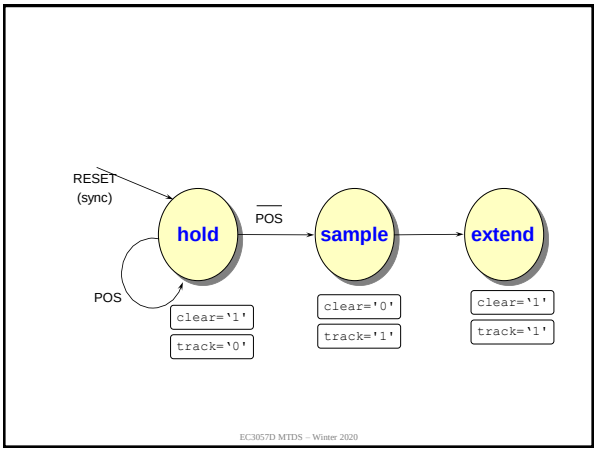
```
48 always @(lockstate)
49 begin
50   case(lockstate)
51     s0: selsw = 0;
52     s1: selsw = 1;
53     s2: selsw = 2;
54     s3: selsw = 3;
55     wrong: selsw = 0;
56     unlock: selsw = 0;
57     default: selsw = 2'bx;
58   endcase
59 end

60 assign locked = (lockstate == unlock) ? 0: 1;
61 assign alarm = (lockstate == wrong) ? 0: 1;

62 assign entimer = (lockstate == unlock) ? 1: 0;

63 endmodule
```



```
1  module comblock(input clock, clear,
2              input [7:0] switches,
3              output alarm, locked,
4              output SA, SB, SC, SD, SE, SF, SG);

5  wire mux_out, anysw, codesw,
6       allsw, entimer, timeout;

7  wire [1:0] selsw;

8  //4-to-1 multiplexor
9  assign mux_out = selsw == 0 ? switches[0] :
10             (selsw == 1 ? switches[1] :
11             (selsw == 2 ? switches[2] :
12             (selsw == 3 ? switches[3] : 1'b0)));

13 //AND gate for all switches
14 assign allsw = &switches;

15 edgedet det1(.edge_in(mux_out),
16              .detected(codesw),
17              .clock(clock));

18 edgedet det2(.edge_in(allsw),
19              .detected(anysw),
20              .clock(clock));

21 Timer t1(.Clock(clock),
22             .Start(entimer),
23             .Timeout(timeout));

24 lockfsm controller(.clock(clock),
25             .reset(clear),
26             .codesw(codesw),
27             .anysw(anysw),
28             .selsw(selsw),
29             .locked(locked),
30             .alarm(alarm),
31             .entimer(entimer),
32             .timeout(timeout));
```



```
33 segdisp sg1(.locked(locked),
34              .alarm(alarm),
35              .SA(SA),

36              .SB(SB),
37              .SC(SC),
38              .SD(SD),
39              .SE(SE),
40              .SF(SF),
41              .SG(SG));
42 endmodule
```


RESET
(sync)
hold
POS
sample
extend
POS
clear='1'
track='0'
clear='0'
track='1'
clear='1'
track='1'
EC3057D MTDS – Winter 2020