Sequential Circuit Design

Overview

- **Sequential Circuit Design**
  - **Specification**
  - **Formulation**
  - **State Assignment**
  - **Flip-Flop Input and Output Equation**
  - **Verification**

The Design Procedure

- **Specification - Description of the Problem**
- **Formulation - Obtain a state diagram or state table**
- **State Assignment - Assign binary codes to the states**
- **Flip-Flop Input Equation Determination**
  - **Select flip-flop types**
  - **Derive flip-flop equations from next state entries in the table**
- **Output Equation Determination**
  - **Derive output equations from output entries in the table**
- **Optimization - Optimize the equations**
- **Technology Mapping - Use available flip-flops and gate technology**
- **Verification - Verify correctness of final design**

Formulation: Finding a State Diagram

- A State is an abstraction of the history of the past applied inputs to the sequential circuit
- A state is used to remember something about the history of input combinations applied to the circuit
  - The interpretation of past inputs is tied to the synchronous operation of the circuit
  - An input value is considered only during the setup-hold time interval for an edge-triggered flip-flop.
- Examples:
  - State A represents the fact that a '1' input has occurred among the past inputs.
  - State B represents the fact that a '0' followed by a '1' have occurred as the most recent past two inputs.

Formulation: Finding a State Diagram

- In specifying a circuit, we use states to remember meaningful properties of past input sequences that are essential to predicting future output values
- A sequence recognizer is a sequential circuit that produces a distinct output value whenever a prescribed pattern of input symbols occur in sequence, i.e, recognizes an input sequence occurence
- We will develop a procedure specific to sequence recognizers to convert a problem statement into a state diagram
- Next, the state diagram, will be converted to a state table from which the circuit will be designed

Sequence Recognizer Procedure

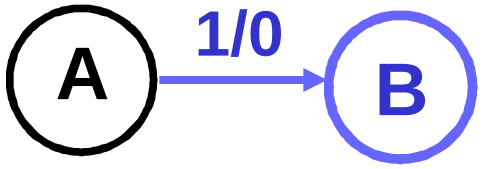
- **Begin in an initial state in which NONE of the initial portion of the sequence has occurred (reset state)**
- **Add a state that recognizes that first symbol has occurred**
- **Add states that recognize each successive symbol**
- **The final state represents the input sequence occurence**
- **Add state transition arcs which specify what happens when a symbol not in the proper sequence has occurred**
- **Add other arcs which transition to states that represent the input subsequence that has occurred**
  - **The circuit must recognize the input sequence regardless of where it occurs within the overall sequence**

Sequence Recognizer Example

- **Example: Recognize the sequence 1101**
  - **Example: the sequence 1111101 contains 1101**
- **Thus, the sequential machine must remember that the first two one's have occurred as it receives another symbol**
- **Also, the sequence 1101101 contains 1101 as both an initial subsequence and a final subsequence with some overlap, i. e., 1101101 or 1101101**
- **The 1 in the middle, 1101101, is in both subsequences**
- **The sequence 1101 must be recognized each time it occurs in the input sequence**

Example: Recognize 1101

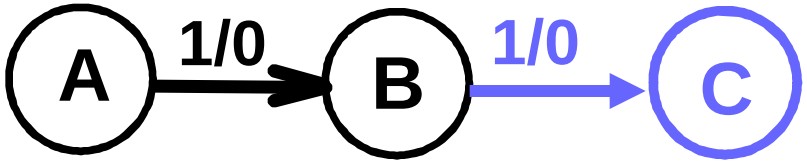
- **Define states for the sequence to be recognized:**
  - **Assuming it starts with first symbol**
  - **Continues through each symbol in the sequence to be recognized**
  - **Uses output 1 to mean the full sequence has occurred**
  - **With output 0 otherwise**
- **Start in the initial state**



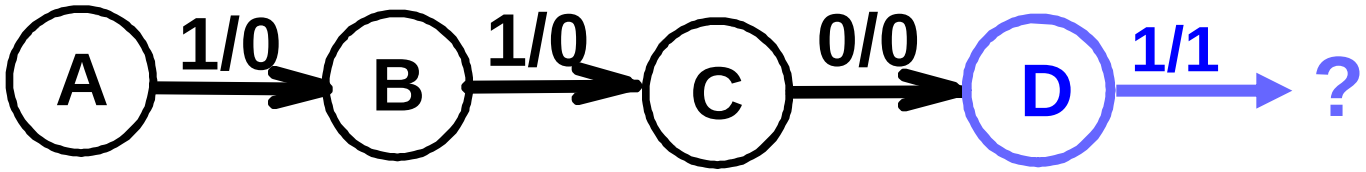
  - **State ‘A’ is the initial state**
  - **Add a state ‘B’ that recognizes the first ‘1’**
  - **State ‘B’ is the state which represents the fact that the first ‘1’ in the input subsequence has occurred. The output symbol ‘0’ means that the full recognized sequence has not yet occurred**

Example: Recognize 1101 (continued)

- After one more '1', we have:
  - C is the state obtained when the input sequence has two '1's.

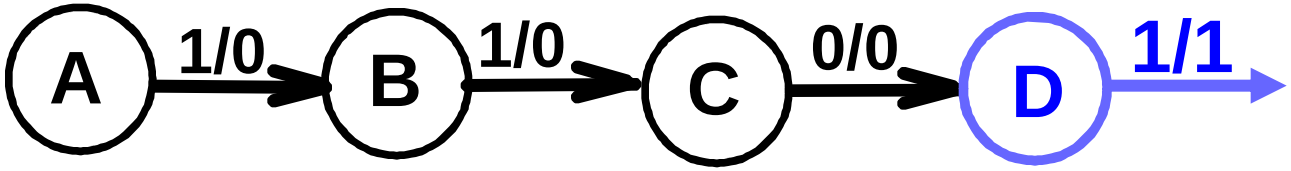


- Finally, after '110' and a '1', we have:

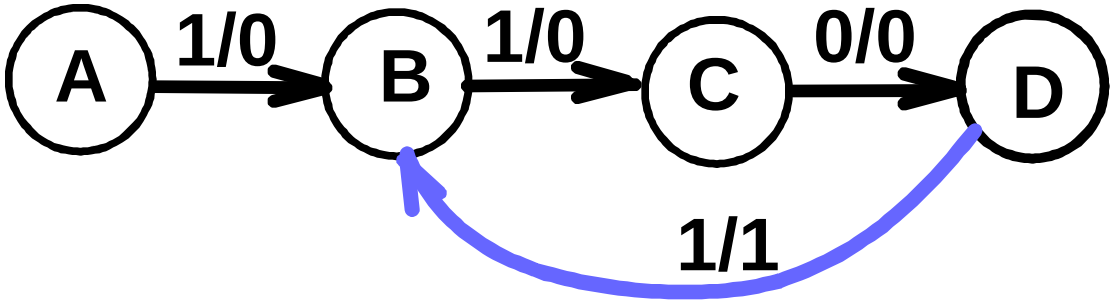


  - Transition arcs are used to denote the output function
  - Output '1' on the arc from D means the sequence is recognized
  - To what state should the arc from state D go? recall 1101<u>1101</u>

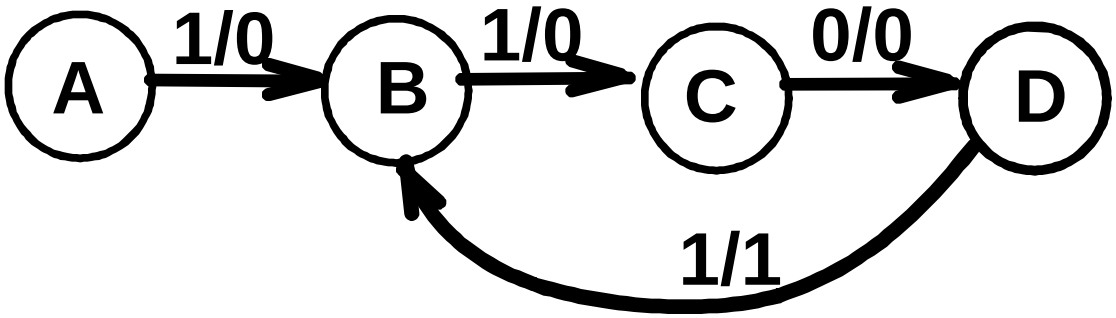
Example: Recognize 1101 (continued)



- Clearly the final '1' in the recognized sequence 1101 is a sub-sequence of 1101. It follows a '0' which is not a sub-sequence of 1101. Thus it should represent *the same state reached from the initial state after a first* '1' *is observed.* We obtain:



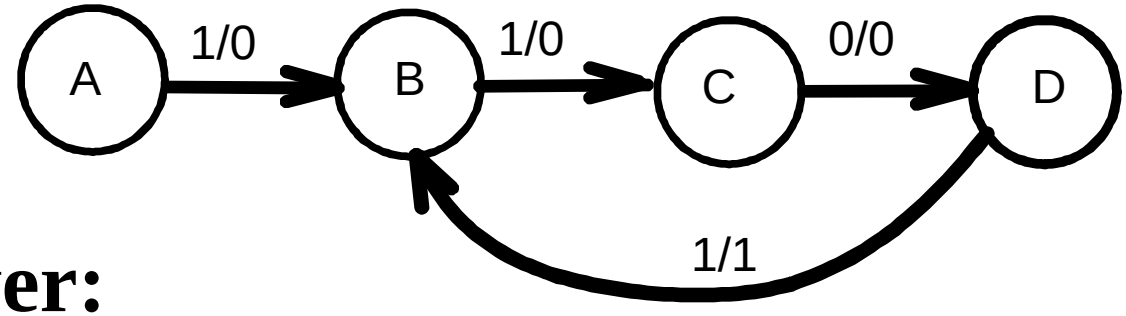
Example: Recognize 1101 (continued)



- The states have the following meanings:
  - A: Start state, no sub-sequence has occurred
  - B: The sub-sequence ‘1’ has occurred
  - C: The sub-sequence ‘11’ has occurred
  - D: The sub-sequence ‘110’ has occurred
  - The 1/1 on the arc from D to B means that the last ‘1’ in 1101 has occurred and thus, the output is ‘1’

Example: Recognize 1101 (continued)

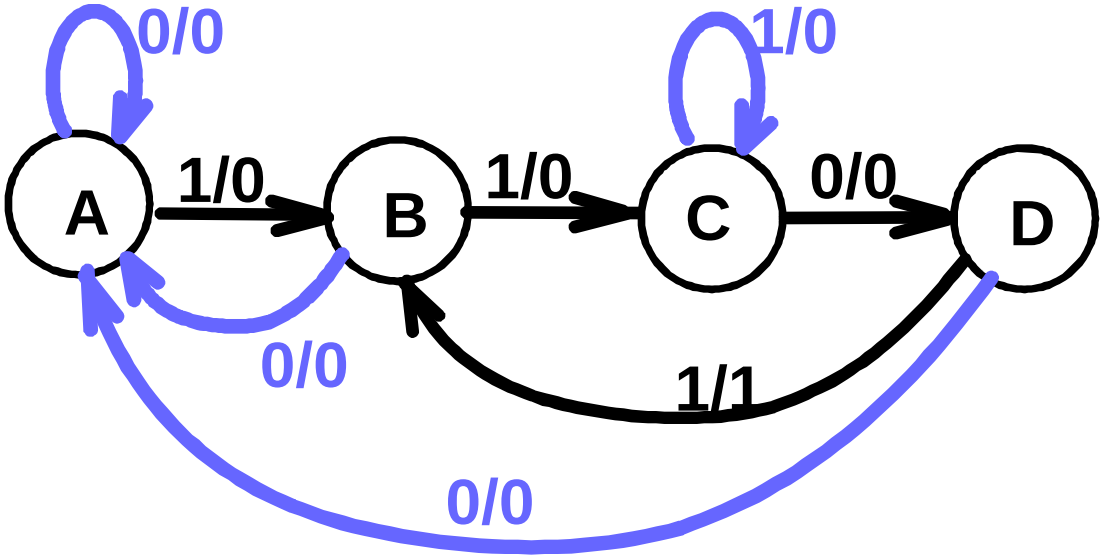
- **The other arcs are added to each state for inputs are not yet listed. Which arcs are missing?**



- **Answer:**
  - **‘0’ arc from state A**
  - **‘0’ arc from state B**
  - **‘1’ arc from state C**
  - **‘0’ arc from state D**

Example: Recognize 1101 (continued)

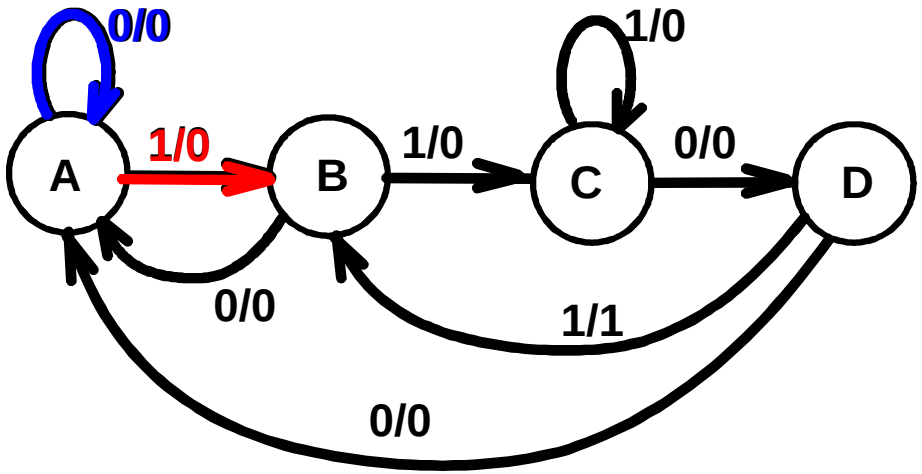
- **Add the arcs for missing inputs at any state to make the state diagram complete. We get:**



- **The ‘1’ arc from state C to itself implies that State C means *two or more 1's have occurred*.**

Formulation: Find the State Table

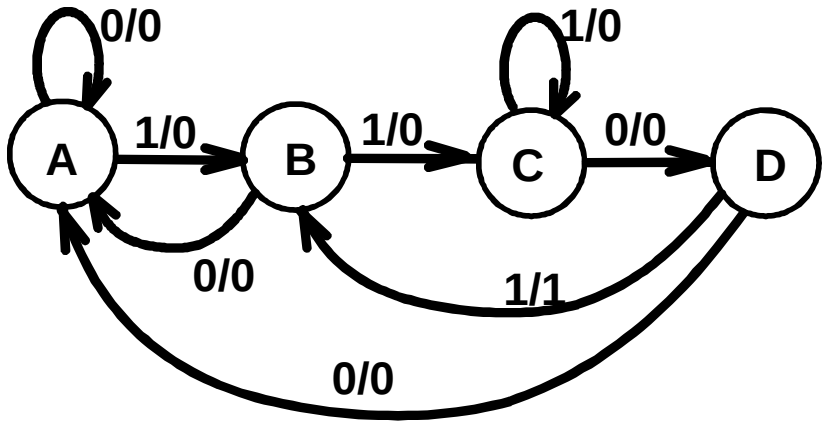
- From the State Diagram, we can fill in the State Table
- There are 4 states, one input, and one output
- We will draw a table with four rows, one for each current state
- From State A, the '0' and '1' input transitions have been filled in along with the outputs



Present State	Next State x=0	Next State x=1	Output x=0	Output x=1
A	A	B	0	0
B				
C				
D				

Formulation: Find State Table

- From the state diagram, we obtain the state table



Present State	Next State x=0	Next State x=1	Output x=0	Output x=1
A	A	B	0	0
B	A	C	0	0
C	D	C	0	0
D	A	B	0	1

State Assignment

- **Each state must be assigned a unique code**
- **Minimum number of bits required for *m* states in the state diagram is *n* such that**

  $n \geq \lceil \log_2 m \rceil$, **where** $\lceil x \rceil$ **is the smallest integer** $\geq x$

- **There are useful state assignments that use more than the minimum number of bits**
- **If *n* bits are used, there are $2^n - m$ unused states**

State Assignment – Example

Present State	Next State x=0	Next State x=1	Output x=0	Output x=1
A	A	B	0	0
B	A	C	0	0
C	D	C	0	0
D	A	B	0	1

- What is the minimum number of bits to code 4 states?
  - Answer: 2 bits ($\log_2 4 = 2$). Therefore, 2 flip-flops are required
  - With 2 bits, we can have 4 codes: 00, 01, 10, and 11
- How may assignments of 2-bit codes to the 4 states?
  - Answer: $4 \times 3 \times 2 \times 1 = 24$ possible assignments
- Does code assignment make a difference in cost?
  - Answer: yes, it affects the cost of the combinational logic

Counting Order State Assignment

- **One possible assignment is Counting Order**

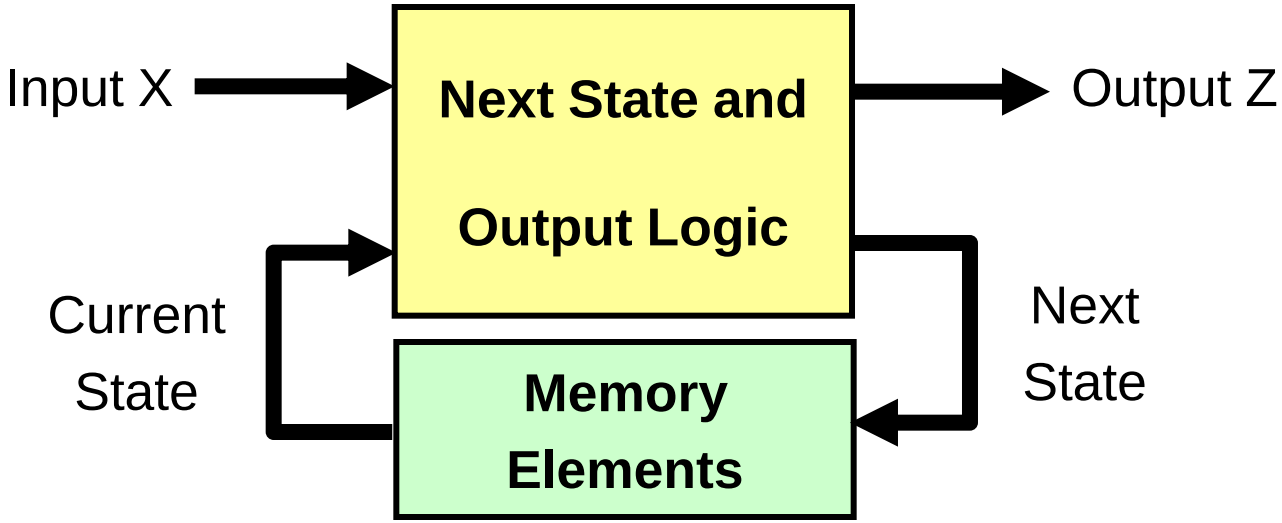
  A = 00, B = 01, C = 10, D = 11

- **The resulting coded state table:**

Present State $Y_1\ Y_2$	Next State x = 0	Next State x = 1	Output Z x = 0	Output Z x = 1
A = 0 0	0 0	0 1	0	0
B = 0 1	0 0	1 0	0	0
C = 1 0	1 1	1 0	0	0
D = 1 1	0 0	0 1	0	1

General Structure of Sequence Detector

- **To implement the 1101 sequence detector**
  - **Choose the type of flip-flops that will be used as memory elements**
  - **Determine and minimize the next state and output equations**
  - **These equations are functions of input and current state**
  - **Implement the next state and output combinational logic**



Find Next State and Output K-maps

- Assume D flip-flops & counting order assignment
- Obtain the K-maps for flip-flop inputs D_1, D_2, and output Z

D_1

Y_1Y_2 \ X	0	1
00	0	0
01	0	1
11	0	0
10	1	1

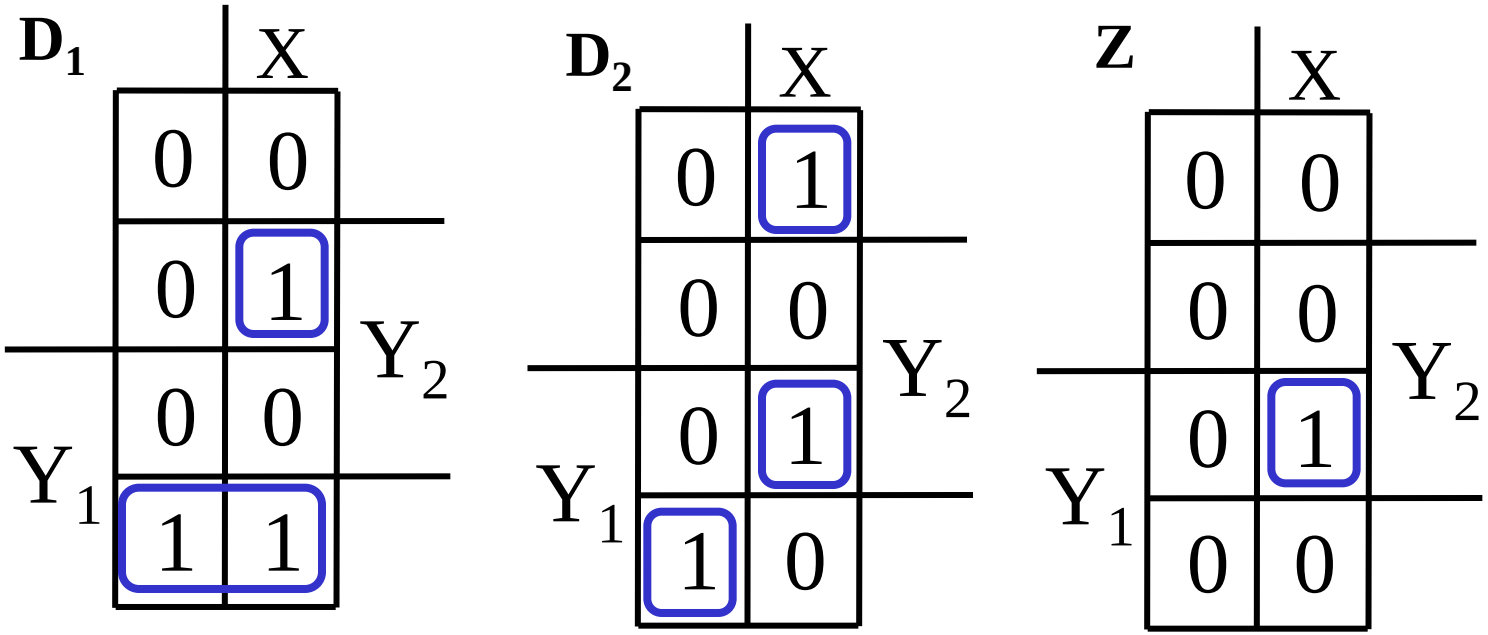
D_2

Y_1Y_2 \ X	0	1
00	0	1
01	0	0
11	0	1
10	1	0

Z

Y_1Y_2 \ X	0	1
00	0	0
01	0	0
11	0	1
10	0	0

Perform two-level optimization



$$D_1 = Y_1\bar{Y}_2 + X\bar{Y}_1Y_2$$

$$D_2 = \bar{X}Y_1\bar{Y}_2 + X\bar{Y}_1\bar{Y}_2 + XY_1Y_2$$

$$Z = XY_1Y_2$$

Gate Input Cost = 22

Gray Code State Assignment

- **Another possible assignment is Gray Code:**

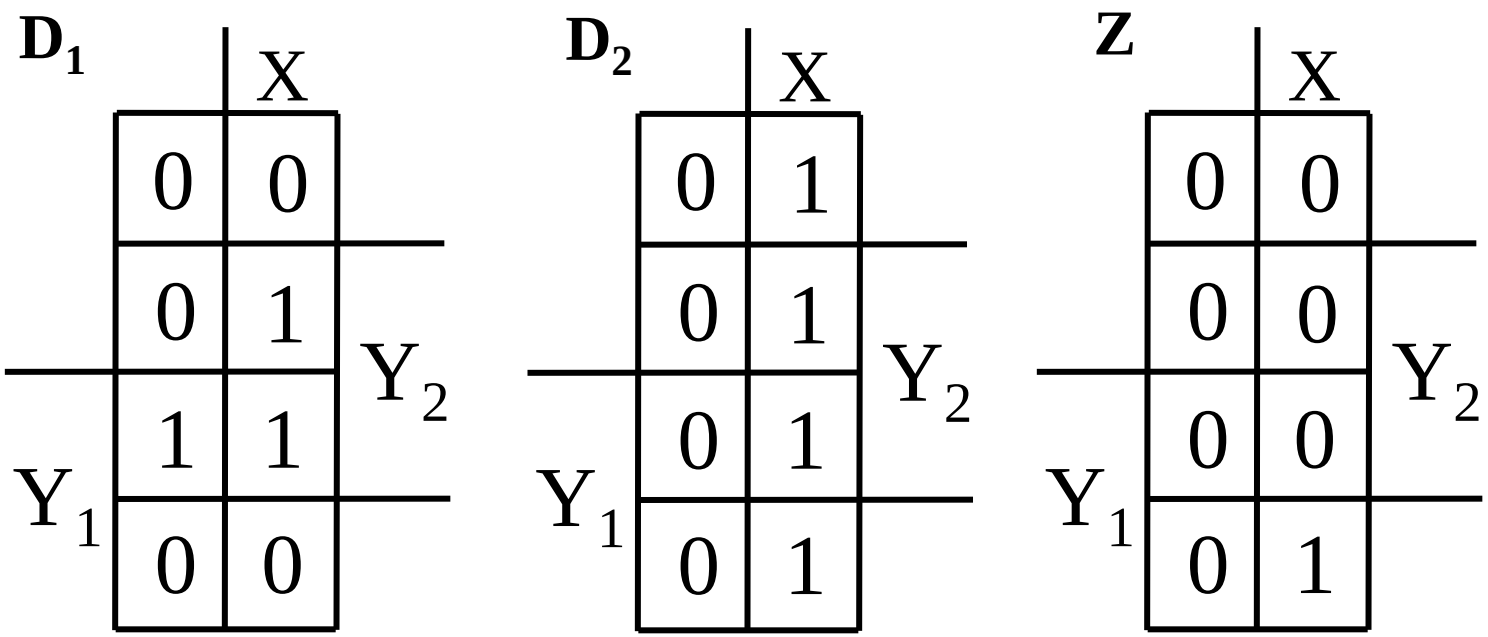
  A = 00, B = 01, C = 11, D = 10

- **The resulting coded state table:**

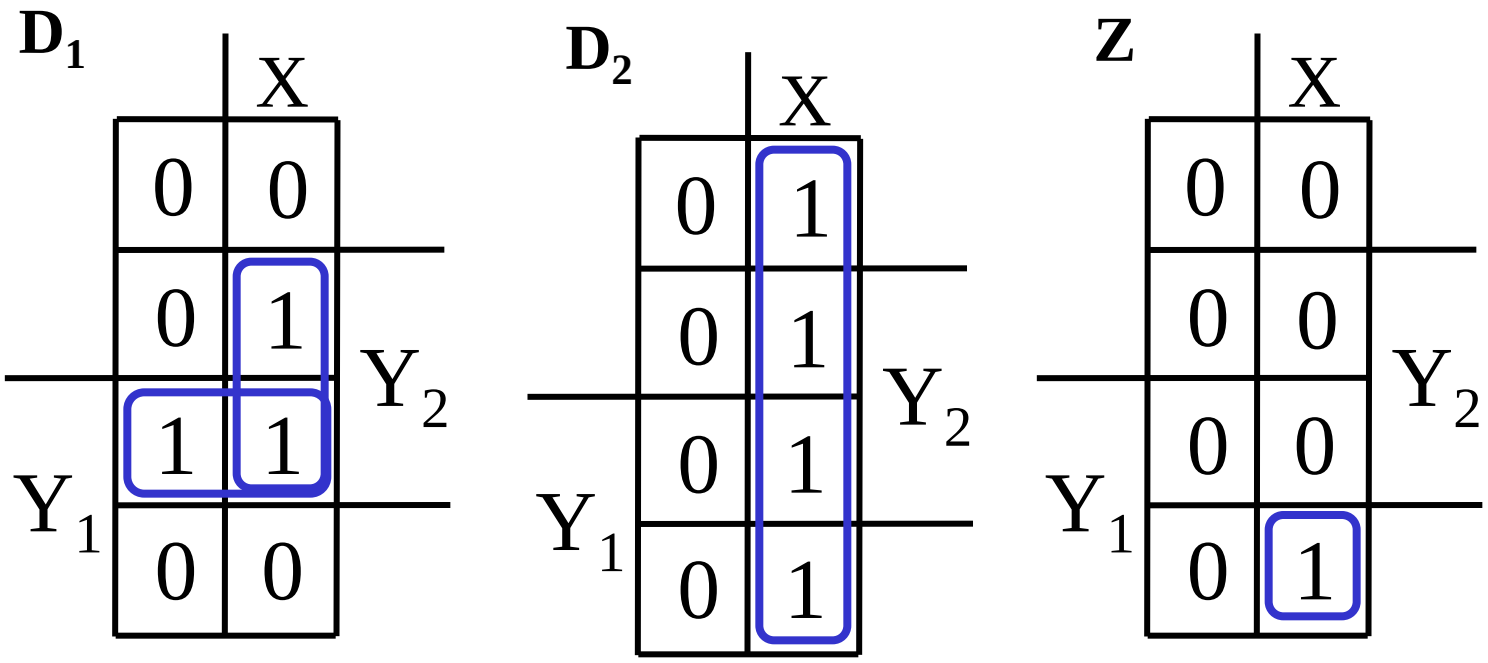
Present State $Y_1\ Y_2$	Next State x = 0	Next State x = 1	Output Z x = 0	Output Z x = 1
A = 0 0	0 0	0 1	0	0
B = 0 1	0 0	1 1	0	0
C = 1 1	1 0	1 1	0	0
D = 1 0	0 0	0 1	0	1

K-Maps for Gray Code State Assignment

- **Assume D flip-flops and gray code assignment**
- **Obtain K-maps for D_1, D_2, and Z:**



Perform two-level optimization



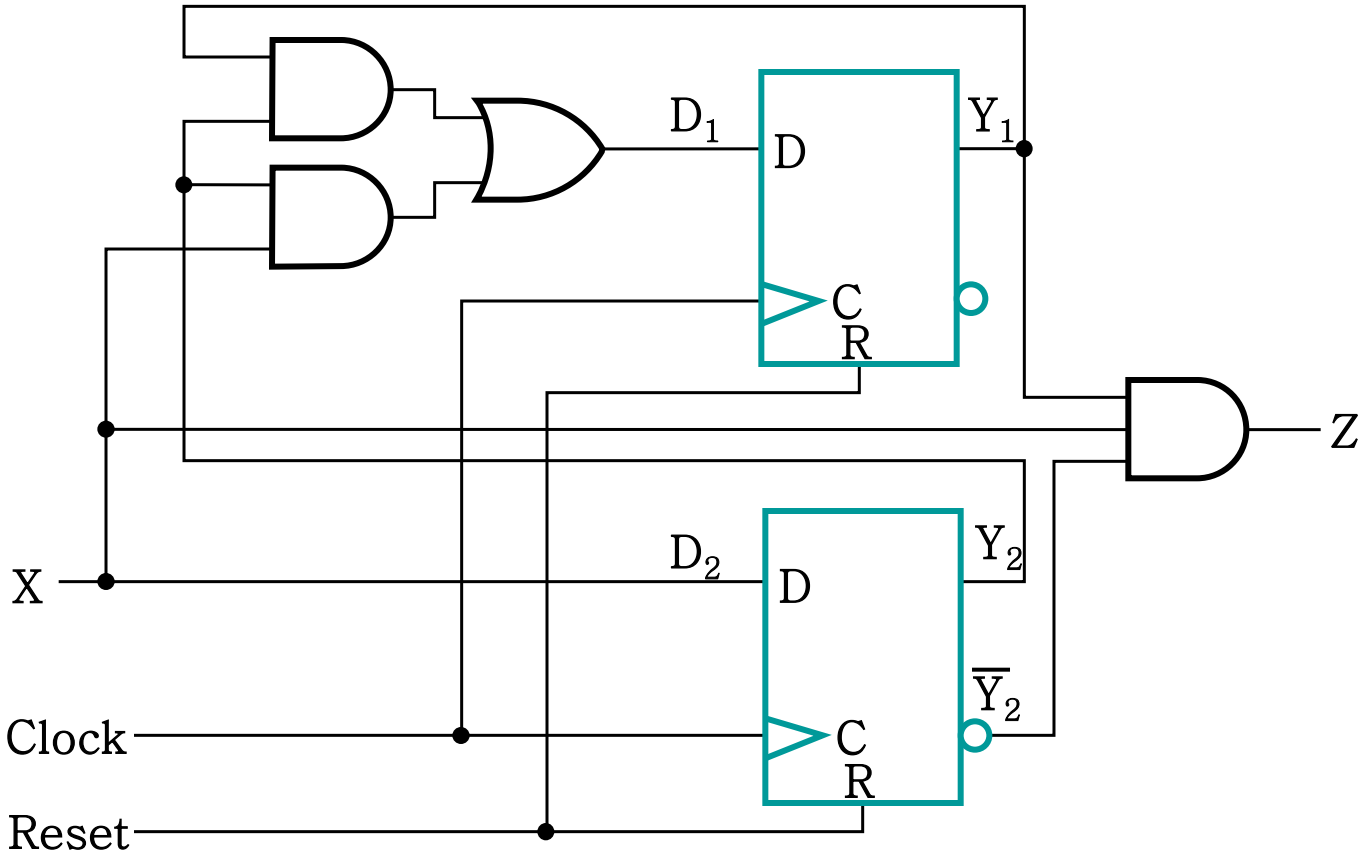
$D_1 = Y_1Y_2 + XY_2$ **Gate Input Cost = 9**

$D_2 = X$ **Select this state assignment to**

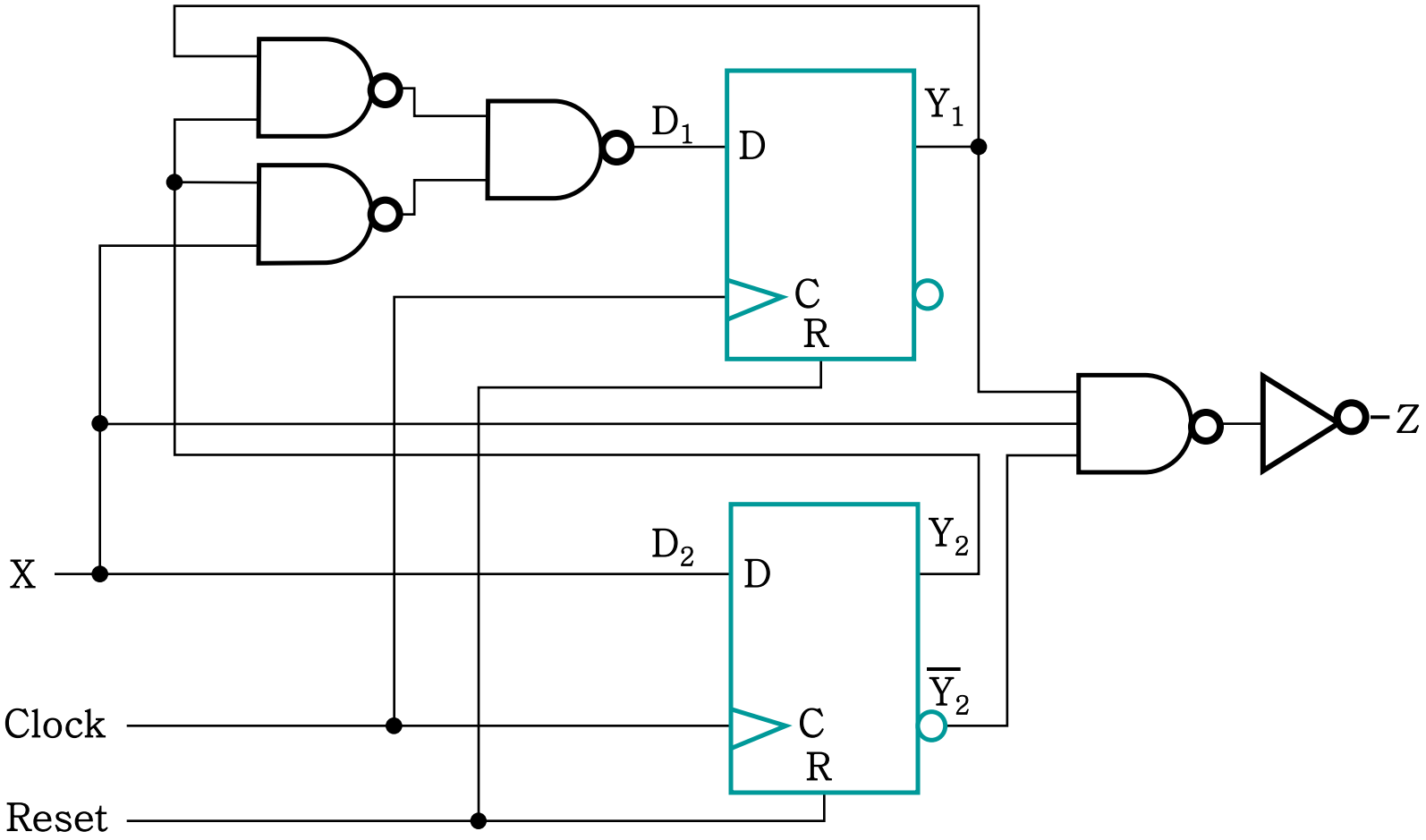
$Z = XY_1\overline{Y}_2$ **complete the design**

Map to Technology

- **Library: D-type Flip-Flops with Reset input**
  - **Reset input is used to reset to start state: $Y_1\ Y_2$ = '00'**



Circuit Implementation with NAND



Using SR, JK, and T Flip-Flop Types

- **Characteristic table (used in analysis)**
  - **Defines the next state of the flip-flop in terms of flip-flop inputs and current state**
- **Characteristic equation (used also in analysis)**
  - **Obtained from characteristic table**
  - **Defines the next state of the flip-flop as a Boolean function of the flip-flop inputs and the current state**
- **Excitation table (used in design)**
  - **Defines the flip-flop input variable values as function of the current state and next state**

SR Flip-Flop

- **Characteristic Table**

S	R	Q(t+1)	Operation
0	0	$Q(t)$	No change
0	1	0	Reset
1	0	1	Set
1	1	?	Undefined

- **Excitation Table**

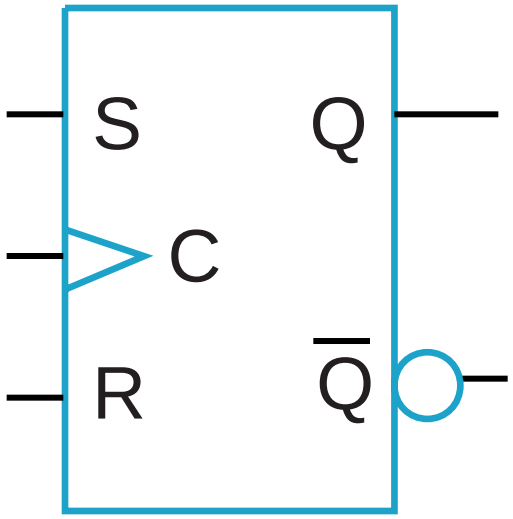
Q(t)	Q(t+1)	S	R	Operation
0	0	0	X	No change
0	1	1	0	Set
1	0	0	1	Reset
1	1	X	0	No change

- **Characteristic Equation**

$Q(t+1) = S + \overline{R}\, Q(t)$

S R = 0 (S and R cannot be 1 simultaneously)

Symbol



JK Flip-Flop

- **Characteristic Table**

J	K	Q(t+1)	Operation
0	0	$Q(t)$	No change
0	1	0	Reset
1	0	1	Set
1	1	$\overline{Q}(t)$	Complement

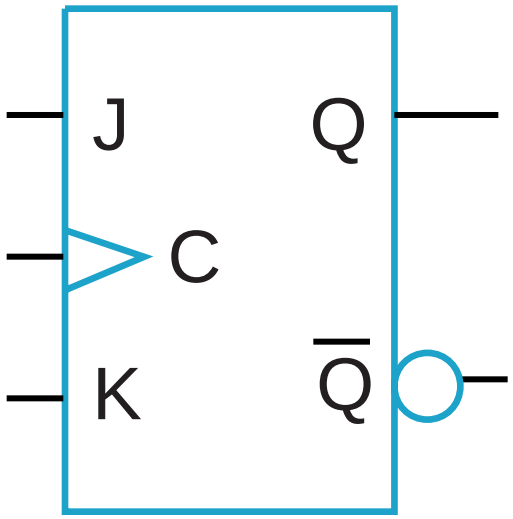
- **Excitation Table**

Q(t)	Q(t+1)	J	K	Operation
0	0	0	X	No change
0	1	1	X	Set
1	0	X	1	Reset
1	1	X	0	No Change

- **Characteristic Equation**

$$Q(t+1) = J\,\overline{Q}(t) + \overline{K}\,Q(t)$$

Symbol



T Flip-Flop

- **Characteristic Table**

T	Q(t+1)	Operation
0	$Q(t)$	No change
1	$\overline{Q}(t)$	Complement

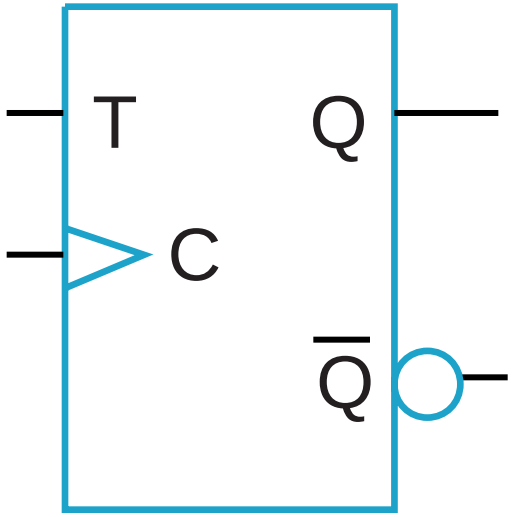
- **Characteristic Equation**

$$Q(t+1) = T \oplus Q(t)$$

- **Excitation Table**

Q(t)	Q(t+1)	T	Operation
0	0	0	No change
0	1	1	Complement
1	0	1	Complement
1	1	0	No Change

Symbol



Obtaining Excitation Table for Flip Flops

- **Use T flip flop for Y_1 and JK flip flop for Y_2**
- **Use Gray code assignment for states**
- **Obtain Excitation table for T and JK inputs:**

Present State	Next State		T input for Y_1		JK input for Y_2		Output Z	
Y_1 Y_2	x=0	x=1	x=0	x=1	x=0	x=1	x=0	x=1
A = 0 0	0 0	0 1	0	0	0 X	1 X	0	0
B = 0 1	0 0	1 1	0	1	X 1	X 0	0	0
C = 1 1	1 0	1 1	0	0	X 1	X 0	0	0
D = 1 0	0 0	0 1	1	1	0 X	1 X	0	1

Optimize Equations for T, J, and K

T

Y_1Y_2 \ X	0	1
00	0	0
01	0	1
11	0	0
10	1	1

J

Y_1Y_2 \ X	0	1
00	0	1
01	X	X
11	X	X
10	0	1

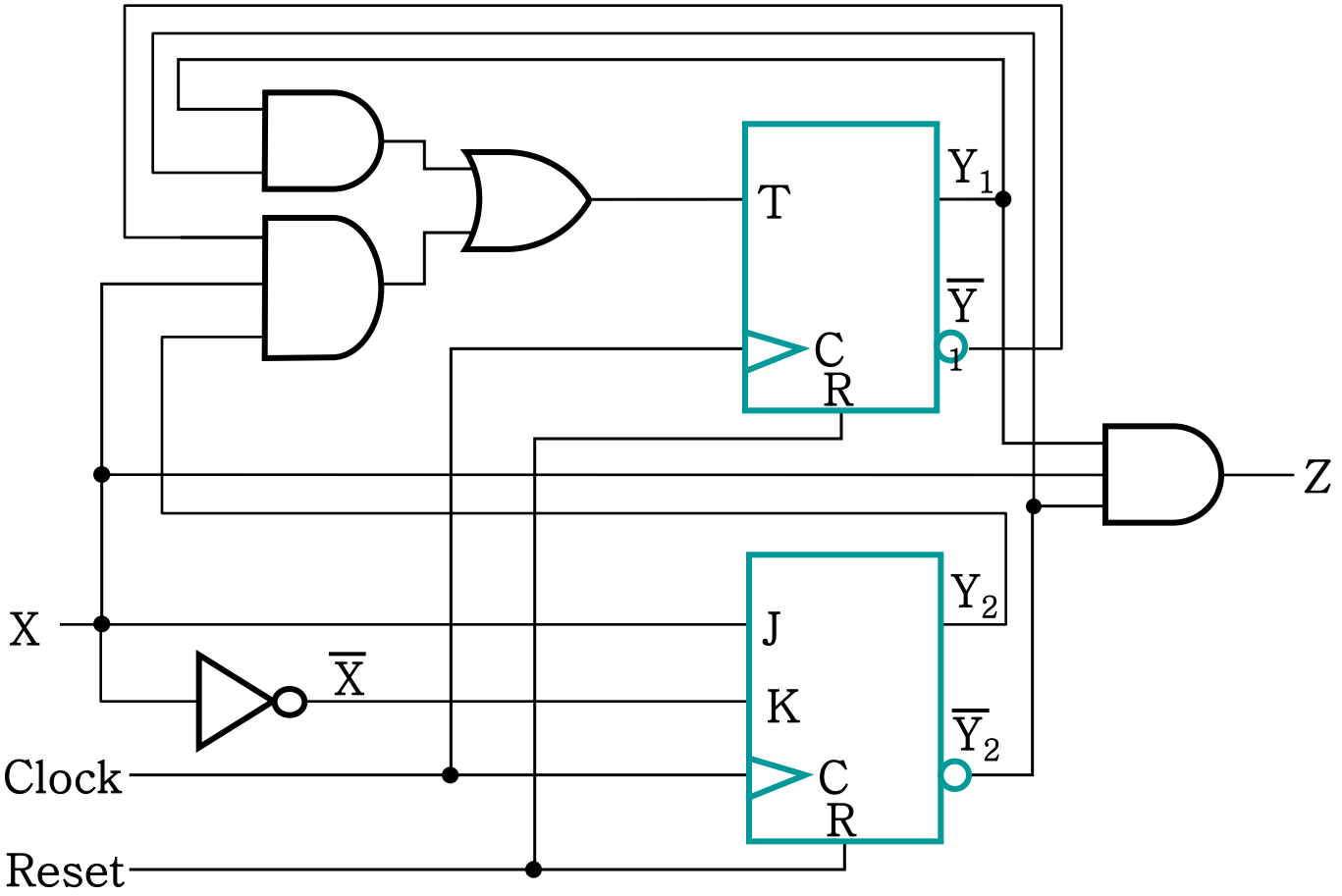
K

Y_1Y_2 \ X	0	1
00	X	X
01	1	0
11	1	0
10	X	X

- $T = Y_1 \overline{Y_2} + X \overline{Y_1} Y_2$, $J = X$, $K = \overline{X}$
- $Z = X Y_1 \overline{Y_2}$ (no change)
- Total Gate Input Cost = 7 + 3 = 10

Circuit Implementation

- **Reset input is used to reset Y_1Y_2 to '00' (start state)**



One-Hot Assignment

- **Use one flip-flop per state: *m* states $\Rightarrow$ *m* flip-flops**
  - **$Y_3Y_2Y_1Y_0$ = 0001 (state A), 0010 (B), 0100 (C), 1000 (D)**
- **Flip-flop cost is higher but combinational logic might be simpler**
- **Provides simplified analysis and design**
  - **In equations, need to include only the variable that is 1 for the state, e. g., state with code 0001, is represented in equations by Y_0 instead of $\overline{Y}_3\ \overline{Y}_2\ \overline{Y}_1\ Y_0$ because if Y_0 is '1' then the remaining state variables will be '0'**

One-Hot State Assignment

- **A = 0001, B = 0010, C = 0100, D = 1000**
- **The resulting coded state table:**

Present State $Y_3\ Y_2\ Y_1\ Y_0$	Next State x = 0	Next State x = 1	Output x = 0	Output x = 1
0001	0001	0010	0	0
0010	0001	0100	0	0
0100	1000	0100	0	0
1000	0001	0010	0	1

Optimization: One Hot Assignment

- **No need for K-map, flip-flop input equations can be obtained directly from the state table**
- **Assume D Flip-Flops**

  $D_0 = \overline{X}(Y_0 + Y_1 + Y_3)$ or $\overline{X}\,\overline{Y_2}$

  $D_1 = X(Y_0 + Y_3) = X\,\overline{Y_1}\,\overline{Y_2}$

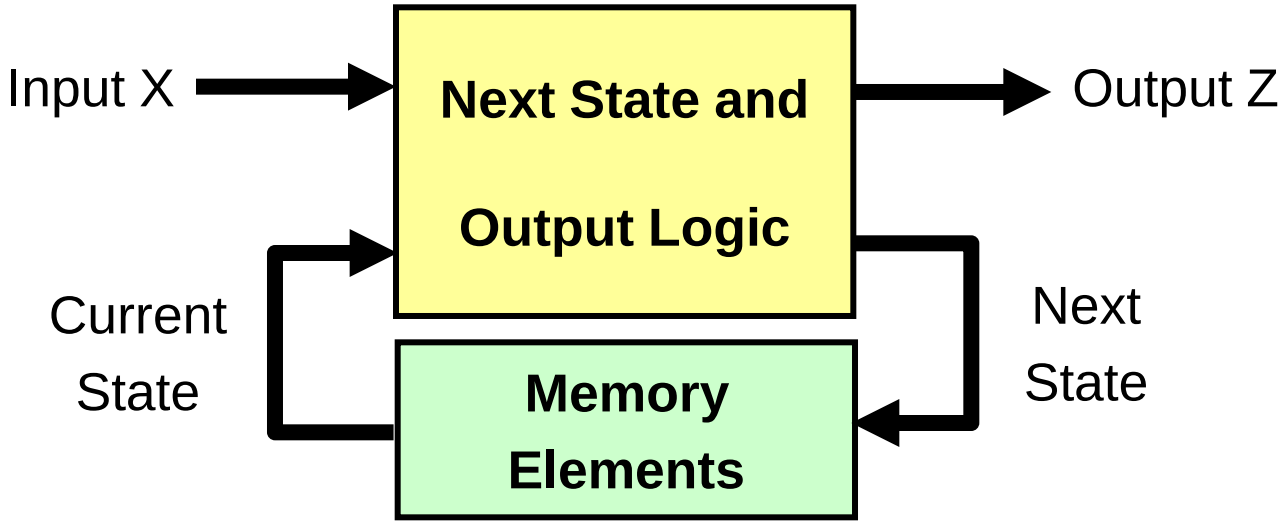
  $D_2 = X(Y_1 + Y_2) = X\,\overline{Y_0}\,\overline{Y_3}$

  $D_3 = \overline{X}\,Y_2$

  $Z = XY_3$ Gate Input Cost = 12
- **Total cost = combinational circuit cost + cost of four flip-flops**

Mealy and Moore Sequential Circuits

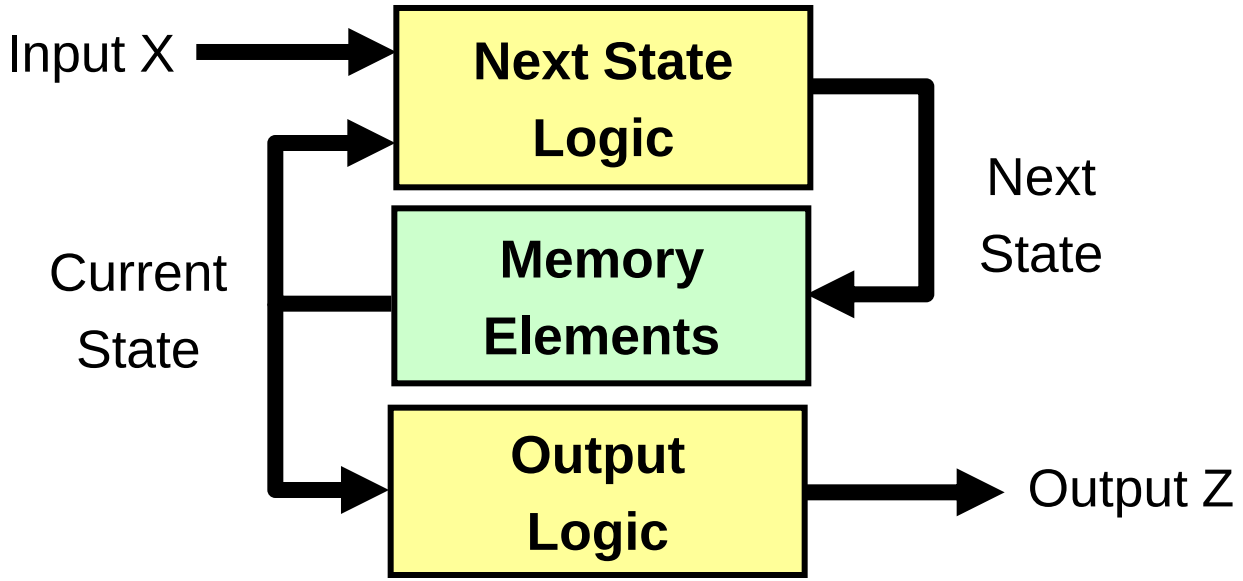
- **Two ways to design clocked sequential circuits**
  - **Mealy and Moore type sequential circuits**
- **Mealy type sequential circuit**
  - **Output is a function of current state and input**



  - **Example: 1101 sequence detector discussed above**

Moore Type Sequential Circuits

- **Output depends on current state only**
  - **Output does not depend on input**
- **Combinational logic is divided into two parts**
  - **Next state logic depends on input and current state**
  - **Output logic depends on current state only**

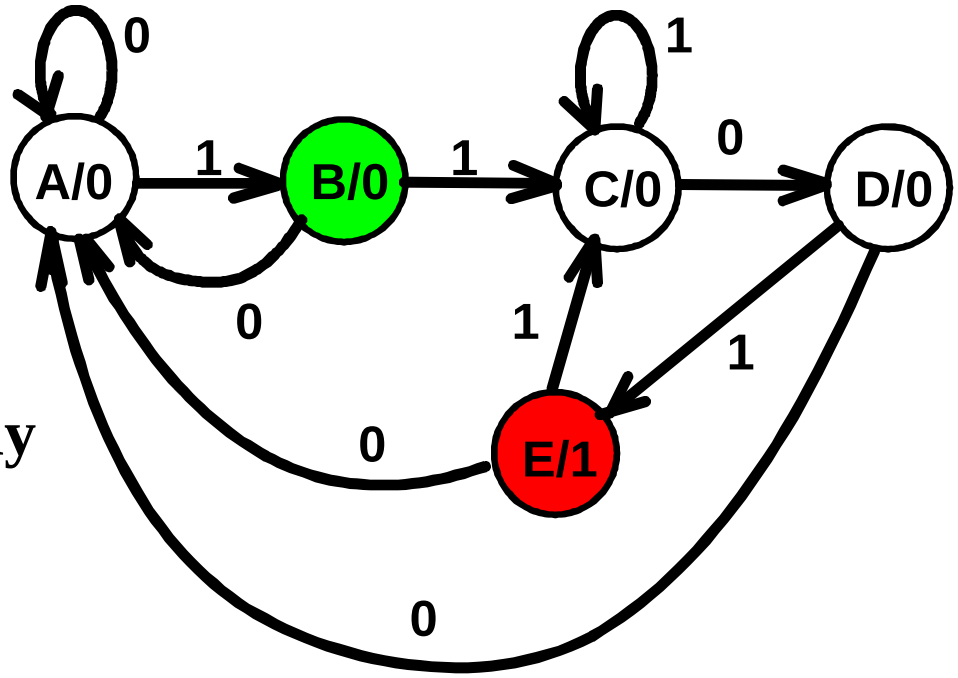


Moore Model for Sequence 1101 Detector

- **For the Moore Model, outputs depend on states**
- **We need to add a state E with output value ‘1’ for the final ‘1’ in the recognized input sequence**
  - **This new state E, though similar to B, would generate an output of ‘1’ and thus be different from state B**
- **The Moore model for a sequence recognizer usually has *more states* than the Mealy model**

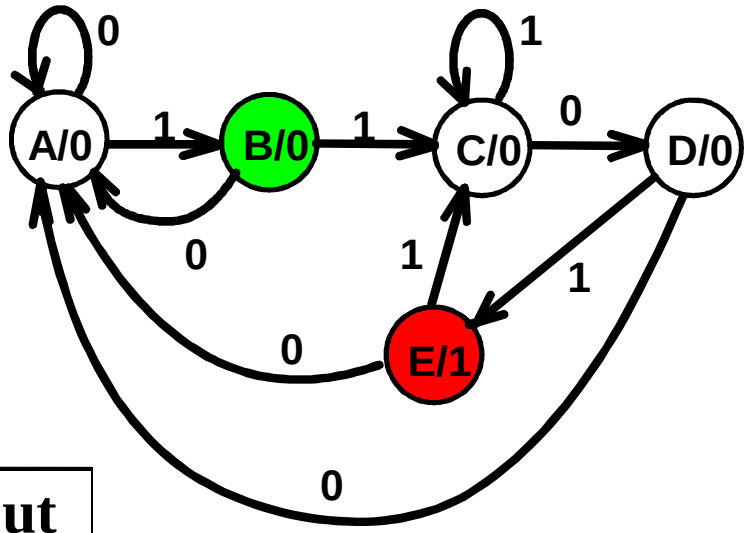
Moore State Diagram

- We mark outputs on states for Moore model
- For Mealy, outputs were marked on arcs
- Arcs now show state transitions and input only
- Add a new state E to produce the output 1
- Note that the new state E produces the same behavior as state B, but gives a different output: ‘1’ rather than ‘0’



Moore State Table

- State and output tables are shown below
- Observe that output y does not depend on input x



Present State	Next State x=0	Next State x=1	Output y
A	A	B	0
B	A	C	0
C	D	C	0
D	A	E	0
E	A	C	1

Moore state diagram typically results in More states

State Assignment for Moore Detector

- **As in the Mealy sequence detector, the Moore sequence detector follows the same procedure for state assignment and optimizing the circuit implementation**
- **Using one-hot assignment, 5 state variables are required**

Present State	Next State		Output
$Y_4\ Y_3\ Y_2\ Y_1\ Y_0$	x=0	x=1	Z
A = 0 0 0 0 1	0 0 0 0 1	0 0 0 1 0	0
B = 0 0 0 1 0	0 0 0 0 1	0 0 1 0 0	0
C = 0 0 1 0 0	0 1 0 0 0	0 0 1 0 0	0
D = 0 1 0 0 0	0 0 0 0 1	1 0 0 0 0	0
E = 1 0 0 0 0	0 0 0 0 1	0 0 1 0 0	1

Equations for Moore Sequence Detector

- **Using D Flip Flops, input equations:**

  $D_0 = \overline{X}\,(Y_0 + Y_1 + Y_3 + Y_4) = \overline{X}\,\overline{Y}_2$

  $D_1 = X\,Y_0$

  $D_2 = X\,(Y_1 + Y_2 + Y_4) = X\,\overline{Y}_0\,\overline{Y}_3$

  $D_3 = \overline{X}\,Y_2$

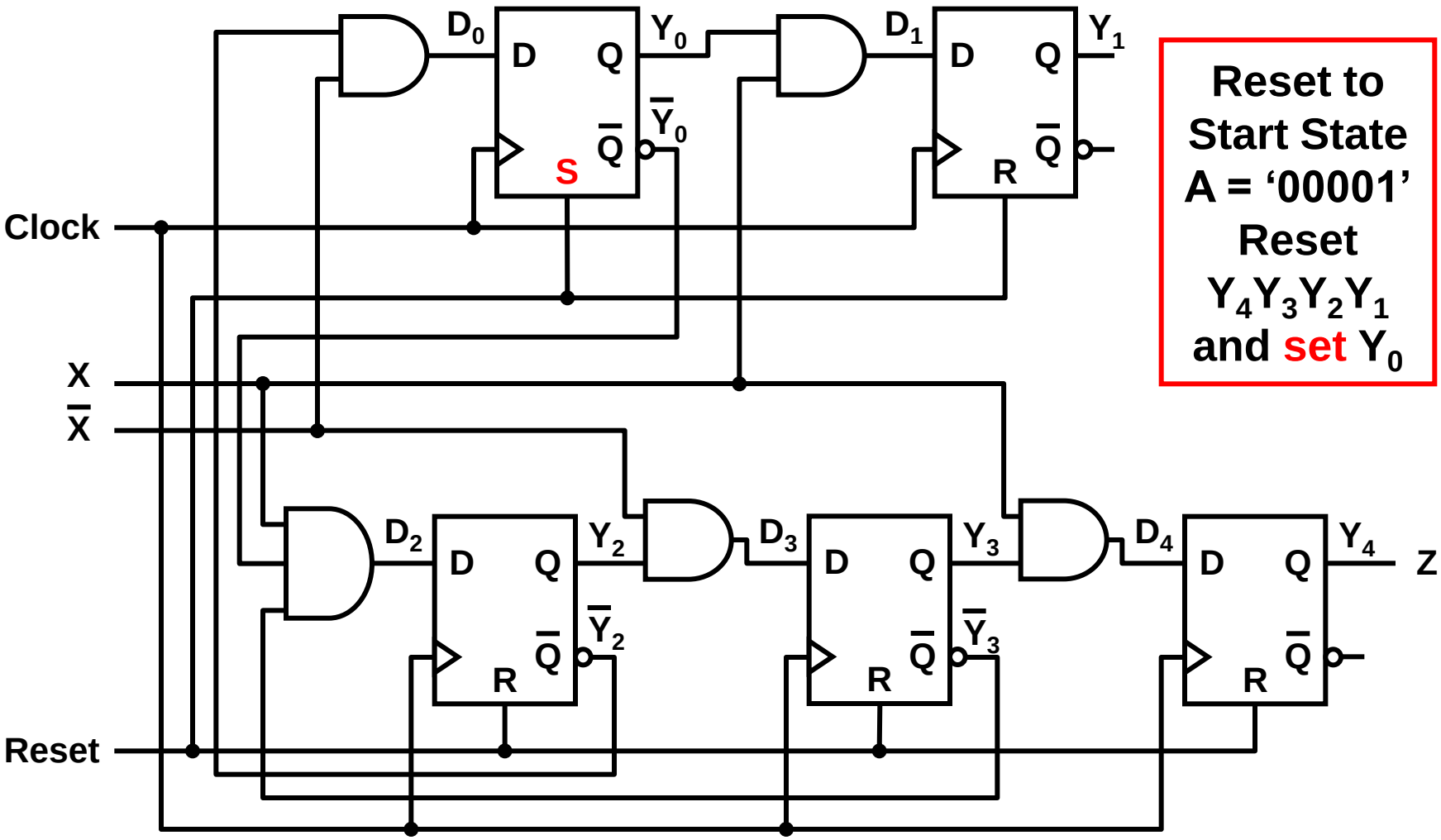
  $D_4 = X\,Y_3$

- **Output Equation Z**

  $Z = Y_4$

- **Gate input cost = 11**

Circuit Implementation



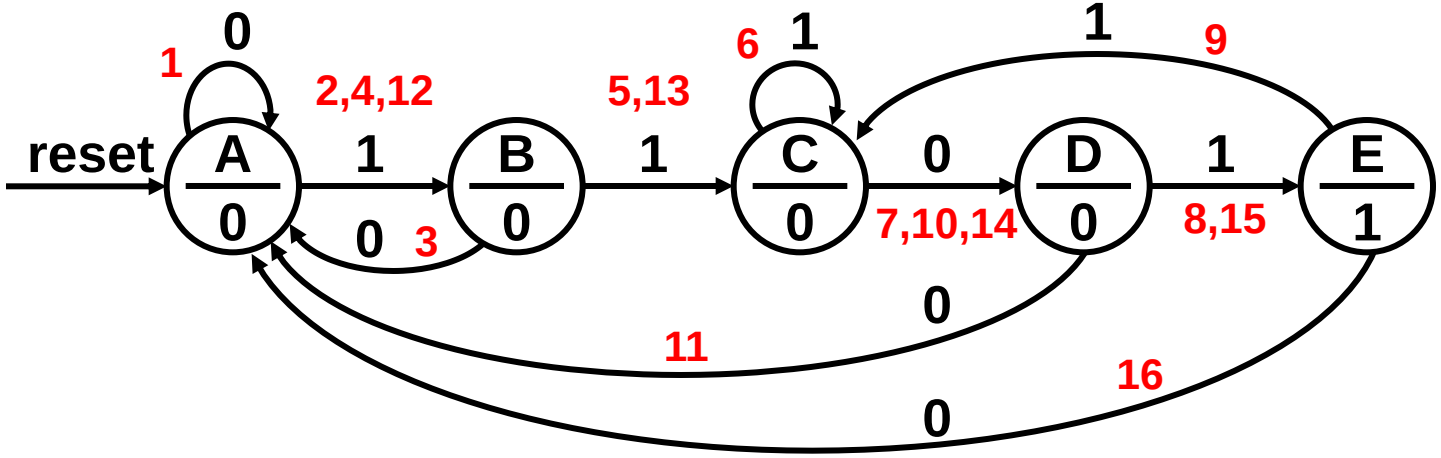
Verification

- **Sequential circuits should be verified by showing that the circuit produces the original state diagram**
- **Verification can be done manually, or with the help of a simulation program**
- **All possible input combinations are applied at each state and the state variables and outputs are observed**
- **A reset input is used to reset the circuit to its initial state**
- **Apply a sequence of inputs to test all the state-input combinations, i.e., all transitions in the state diagram**
- **Observe the output and the next state that appears after each clock edge in the timing diagram**

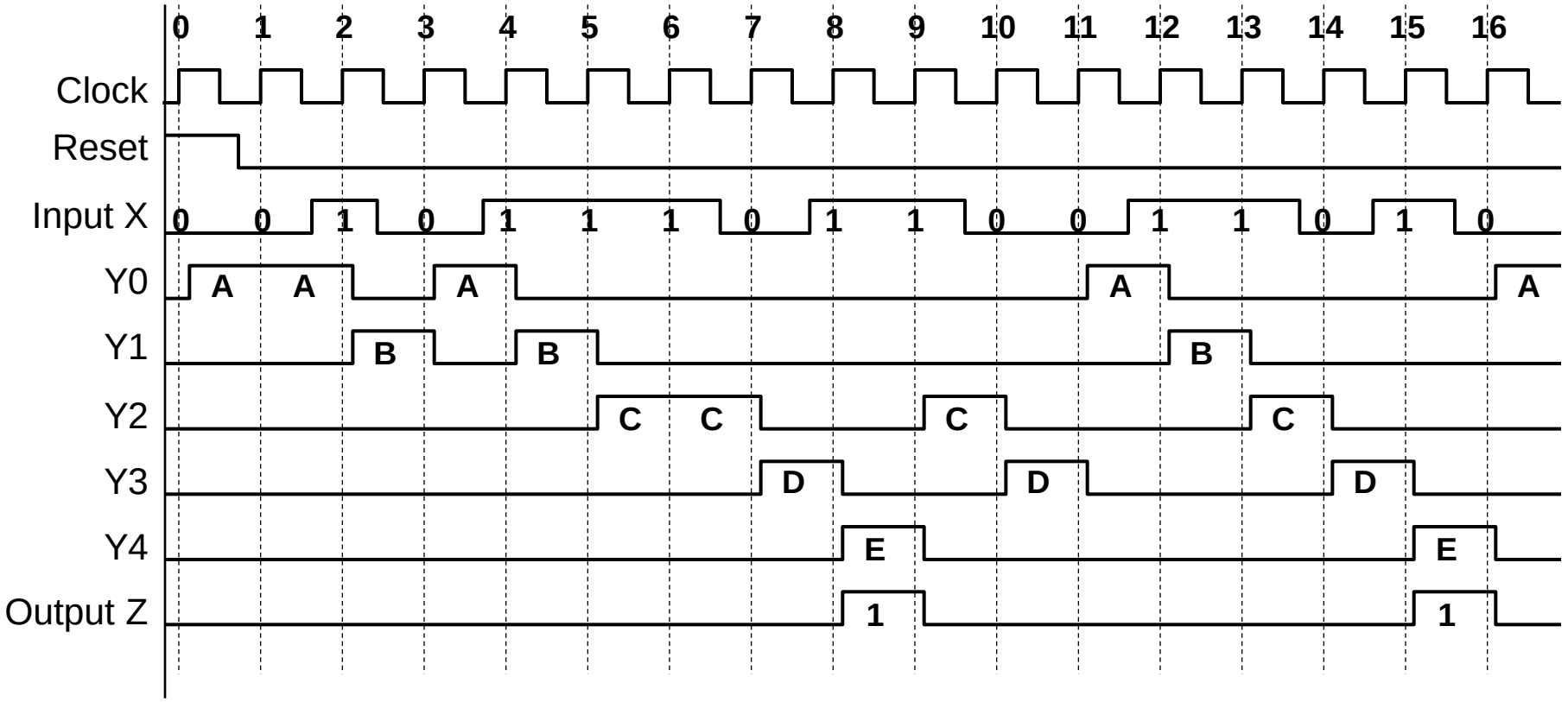
Input Test Sequence

- An **input test sequence** is required to verify the correct operation of a sequential circuit
- It should **test each state transition** of the state diagram
- Test sequences can be generated from the state diagram
- Consider the Moore sequence detector. Starting at A (after reset), we can generate an input test sequence:

  X = 0 1 0 1 1 1 0 1 1 0 0 1 1 0 1 0



Verifying the Moore Sequence Detector

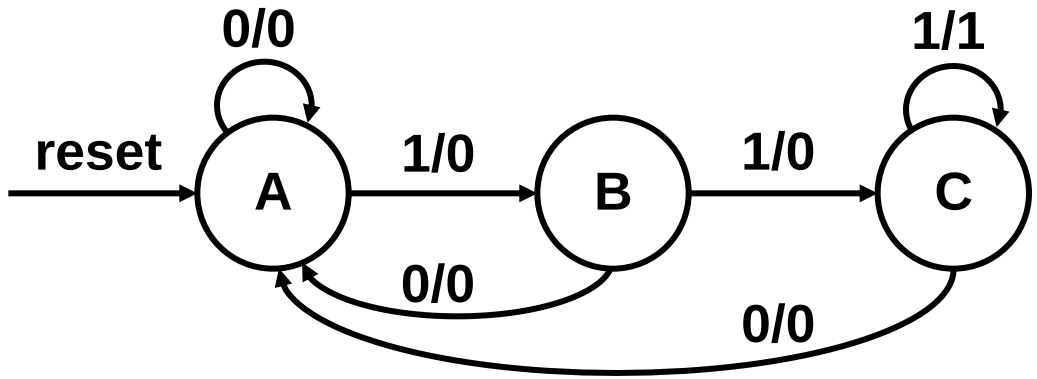


Moore versus Mealy Sequential Circuits

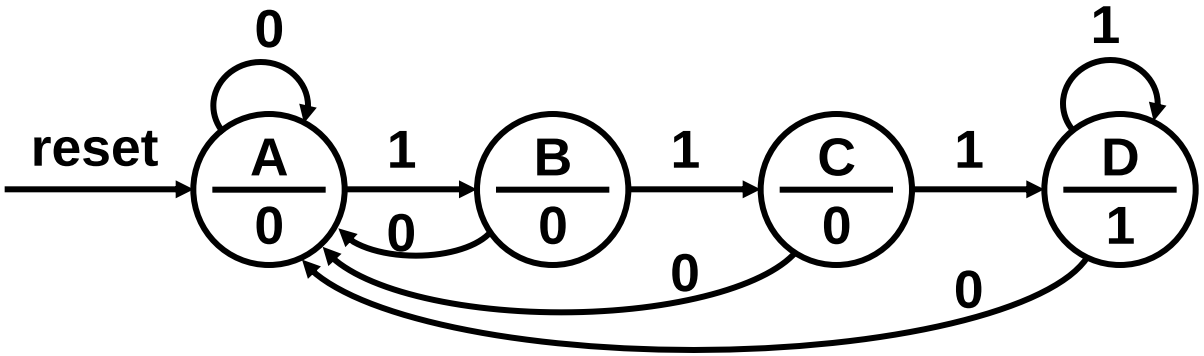
- **Output in a Moore sequential circuit is associated with a state, while output in a Mealy circuit is associated with a transition between states**
- **In general, Moore state diagrams have more states than corresponding Mealy state diagrams and the Moore sequential circuit implementation might have higher cost**
- **Since the output in a Mealy machine is a combination of present state and input values, an unsynchronized input may result in an invalid output (drawback of Mealy)**
- **A Moore state diagram produces a unique output for every state irrespective of inputs. Output of a Moore machine is synchronized with the clock (better)**

Designing a '111' Sequence Detector

- To illustrate the drawback of the Mealy machine, consider the design of a '111' sequence detector



Mealy State Diagram



Moore State Diagram

State Assignment and Equations

- **A minimum of 2 state variables are required**
- **Using Gray Code state assignment and D flip flops**

Mealy State Table

Present State	Next State		Output	
$Y_1\ Y_0$	x=0	x=1	x=0	x=1
A = 0 0	0 0	0 1	0	0
B = 0 1	0 0	1 1	0	0
C = 1 1	0 0	1 1	0	1

- $D_1 = X\ Y_0$
- $D_0 = X$
- $Z = X\ Y_1$

Moore State Table

Present State	Next State		Output
$Y_1\ Y_0$	x=0	x=1	Z
A = 0 0	0 0	0 1	0
B = 0 1	0 0	1 1	0
C = 1 1	0 0	1 0	0
D = 1 0	0 0	1 0	1

- $D_1 = X\ (Y_0 + Y_1)$
- $D_0 = X\ \overline{Y}_1 \quad Z = Y_1\ \overline{Y}_0$

Timing Diagrams