System I

Sequential Logic Design

Bo Feng, Lei Wu, Rui Chang

Zhejiang University

Disclaimer

- **Many images and resources used in this lecture are collected from the Internet, and they are used only for the educational purpose. The copyright belong to the original owners, respectively.**

- **Part of slides credit to**
  - **David Money Harris and Sarah L. Harris. Digital Design and Computer Architecture, 2nd Edition.**
  - **Morris R. Mano , Charles R. Kime and Tom Martin. Logic & Computer Design Fundamentals, Fifth Edition.**
  - **Prof. Yabo Dong @ ZJU**
  - **CSE 140: Components and Design Techniques for Digital Systems (Prof. C.K. Cheng @ UCSD)**
  - **CEG 360/560; EE 451/651: Digital System Design (Prof. Travis Doom @ Wright State University)**

Overview

- Introduction to sequential circuits
- Basic sequential logic elements
- Sequential logic design
- Classic sequential logic elements

Sequential logic design

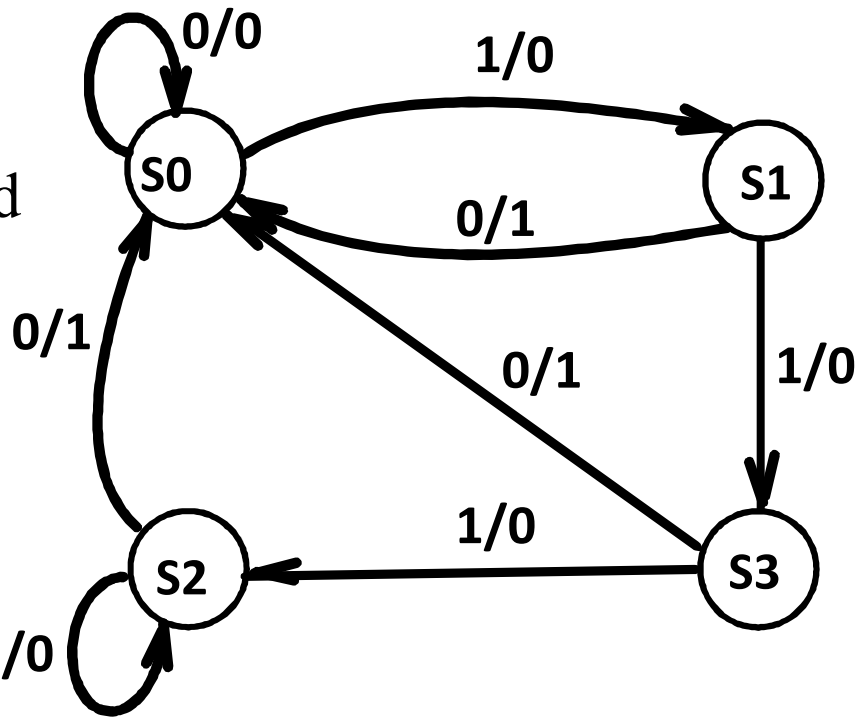
- Some definitions
- The design procedure
- State machine design

Equivalent State

- Two states are ***equivalent*** if their response for each possible input sequence is an identical output sequence.
- Alternatively, two states are ***equivalent*** if their outputs produced for each input symbol is identical and their next states for each input symbol are the same or equivalent.

Equivalent State Example

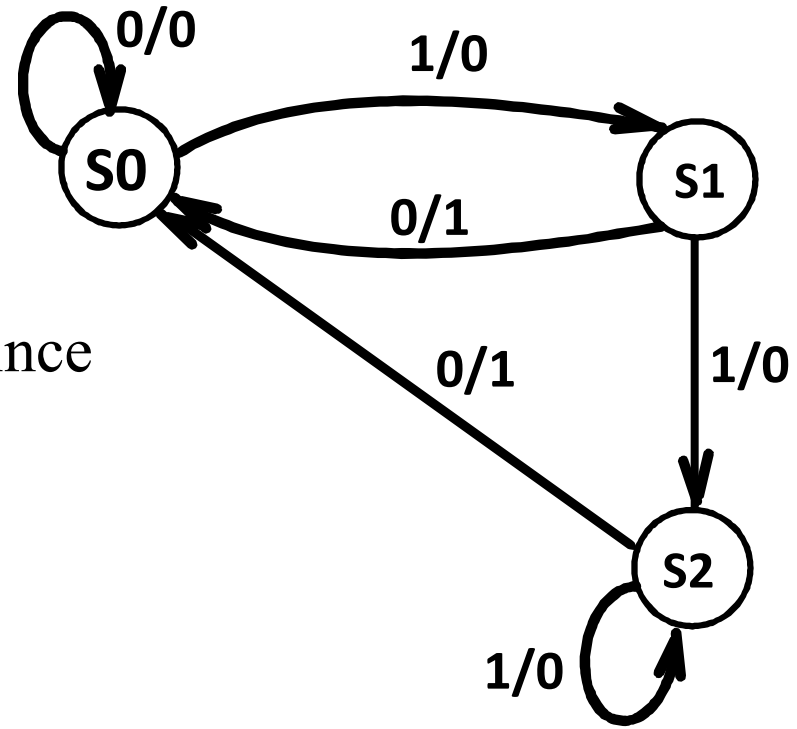
- For states S3 and S2
  - the output for input 0 is 1 and input 1 is 0, and
  - the next state for input 0 is S0 and for input 1 is S2

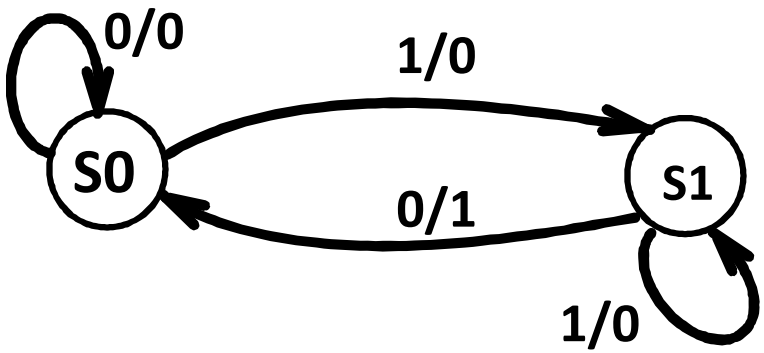


- By the alternative definition, states S3 and S2 are equivalent.

Equivalent State Example

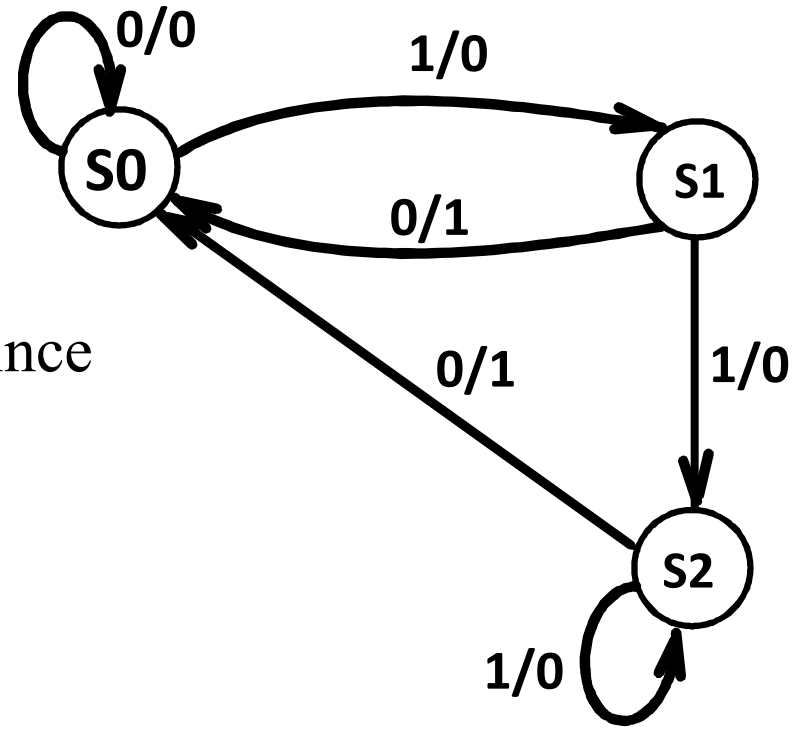
- Replacing S3 and S2 by a single state gives state diagram:
- Examining the new diagram, states S1 and S2 are equivalent since
  - their outputs for input 0 is 1 and input 1 is 0
  - their next state for input 0 is S0 and for input 1 is S2
- Replacing S1 and S2 by a single state gives state diagram:

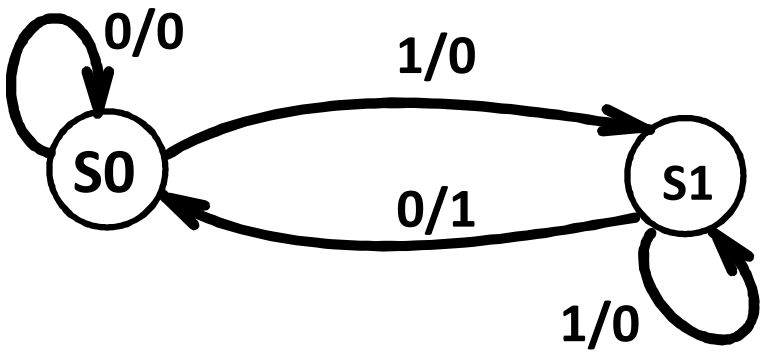




Equivalent State Example

- Replacing S3 and S2 by a single state gives state diagram:
- Examining the new diagram, states S1 and S2 are equivalent since
  - their outputs for input 0 is 1 and input 1 is 0
  - their next state for input 0 is S0 and for input 1 is S2
- Replacing S1 and S2 by a single state gives state diagram:





Moore and Mealy Models

- There are two formal models of FSMs:

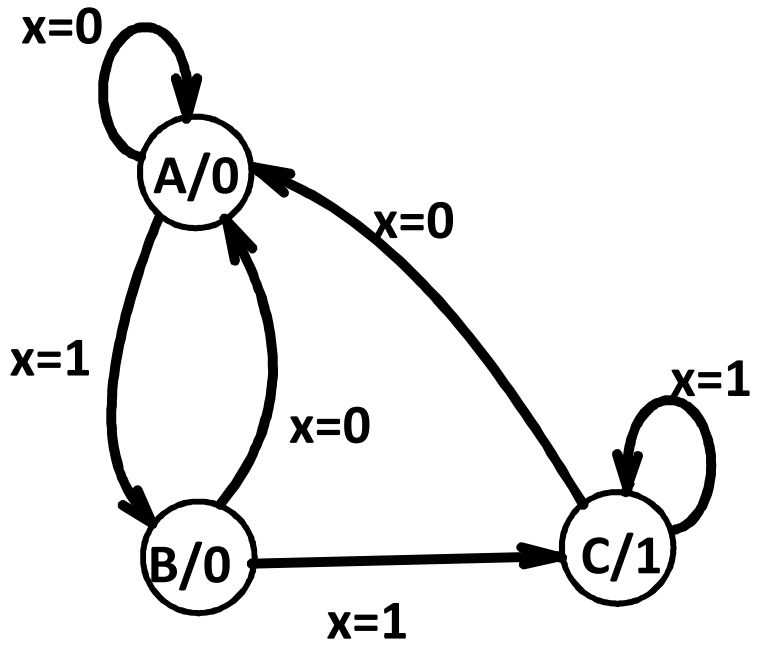
- Moore model
  - Named after E.F. Moore
  - Outputs are a function ONLY of states
  - Usually specified on the states.

- Mealy model
  - Named after G. Mealy
  - Outputs are a function of inputs AND states
  - Usually specified on the state transition arcs.

Moore and Mealy Example

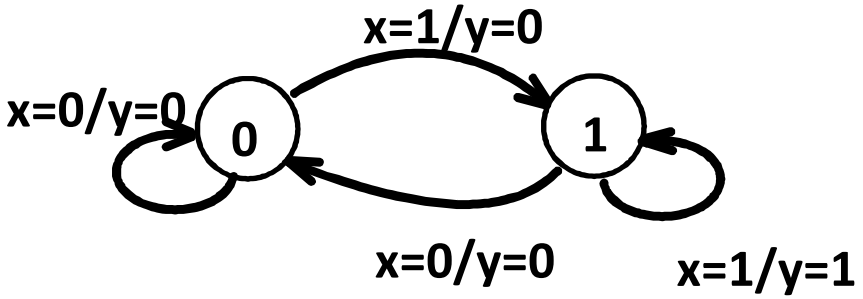
- Moore model state table/diagram maps **states to outputs**

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



- Mealy model state table/diagram maps **inputs and state to outputs**

Present State	Next State x=0	Next State x=1	Output x=0	Output x=1
0	0	1	0	0
1	0	1	0	1



The Design Procedure

- **Specification**
- **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 and 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**
  - Find circuit from equations and map to 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 circuit (including power-up reset or system reset).
  - The interpretation of "past inputs" is tied to the synchronous operation of the circuit. E.g., an input value (other than an asynchronous reset) is measured 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 (cont'd)

- 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 occurrence.
- 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

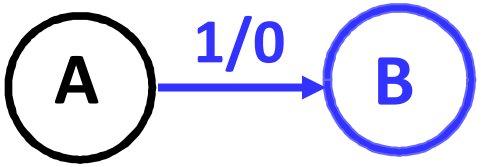
- To develop a sequence recognizer state diagram:
  - Begin in an initial state in which NONE of the initial portion of the sequence has occurred (typically "reset" state).
  - Add a state that recognizes that the first symbol has occurred.
  - Add states that recognize each successive symbol occurring.
  - The final state represents the input sequence (possibly less than the final input value) occurrence.
  - Add state transition arcs which specify what happens when a symbol ***NOT*** in the **proper sequence** has occurred.
  - Add other arcs on non-sequence inputs which transition to states that represent the input subsequence that has occurred.
- The last step is required because the circuit must recognize the input sequence ***regardless of where it occurs within the overall sequence applied since "reset"***.

Sequence Recognizer Procedure (cont'd)

- Example: Recognize the sequence 1101
  - Note that the sequence 1111101 contains 1101 and "11" is a proper sub-sequence of 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.
- And, 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

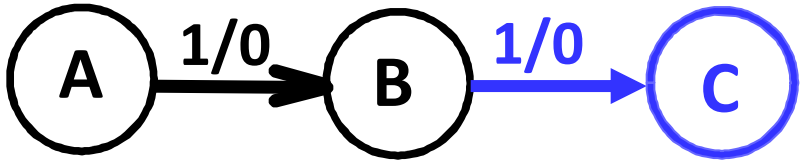
- Define states for the sequence to be recognized:
  - assuming it starts with first symbol,
  - continues through each symbol in the sequence to be recognized, and
  - uses output 1 to mean the full sequence has occurred,
  - with output 0 otherwise.
- Starting in the initial state (Arbitrarily named "A"):
  - Add a state that recognizes the first "1."



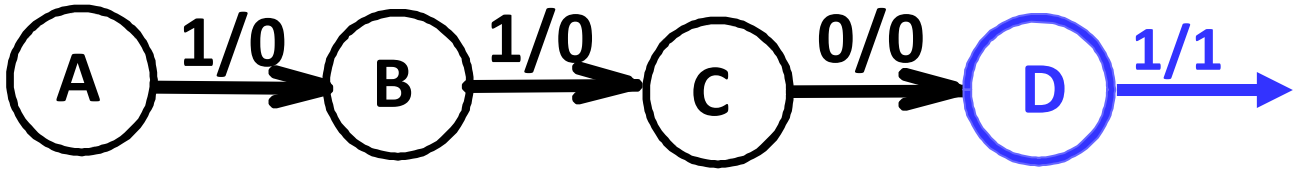
  - State "A" is the initial state, and state "B" is the state which represents the fact that the "first" one in the input subsequence has occurred. The output symbol "0" means that the full recognized sequence has not yet occurred.

Example: Recognize 1101 (cont'd)

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

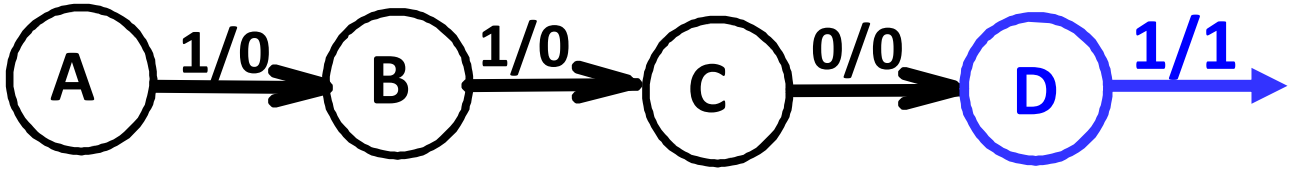


- Finally, after 110 and a 1, we have:

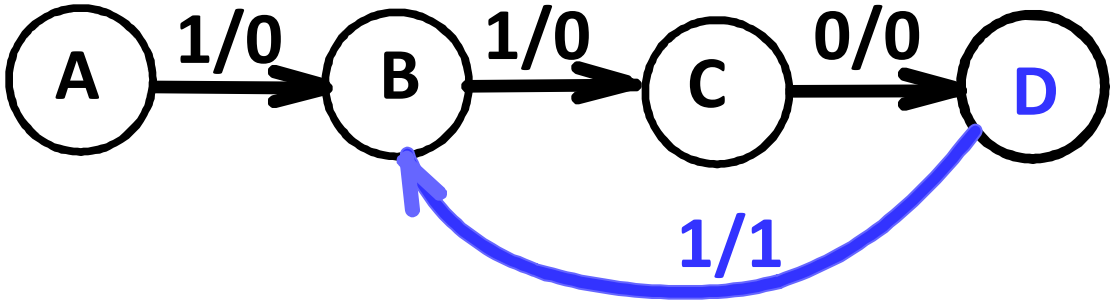


- Transition arcs are used to denote the output function (**Mealy model**)
- Output 1 on the arc from D means the sequence has been recognized
- To what state should the arc from state D go? Remember: 1101<u>1101</u> ?
- Note that D is the last state but the output 1 occurs for the input applied in D. This is the case when a ***Mealy model*** is assumed.

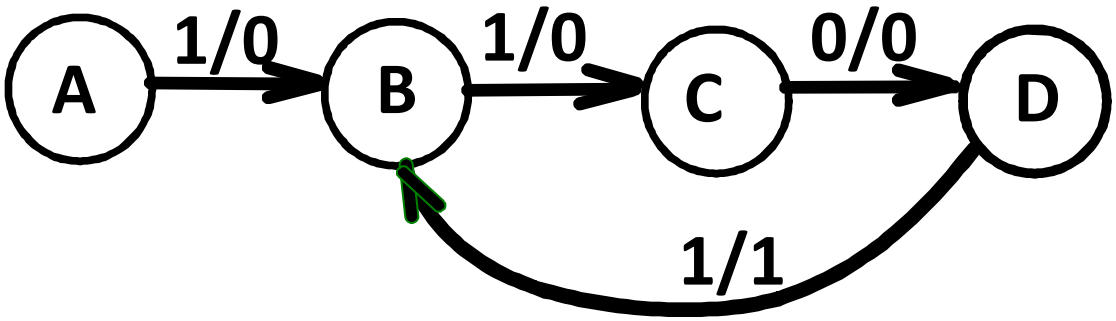
Example: Recognize 1101 (cont'd)



- Clearly the final 1 in the recognized sequence 1101 is a sub-sequence of 1101.
- It must be the first 1 in the sequence, since it cannot be preceded by a 1. Thus it should represent ***the same state reached from the initial state after a first 1 is observed***. We obtain:



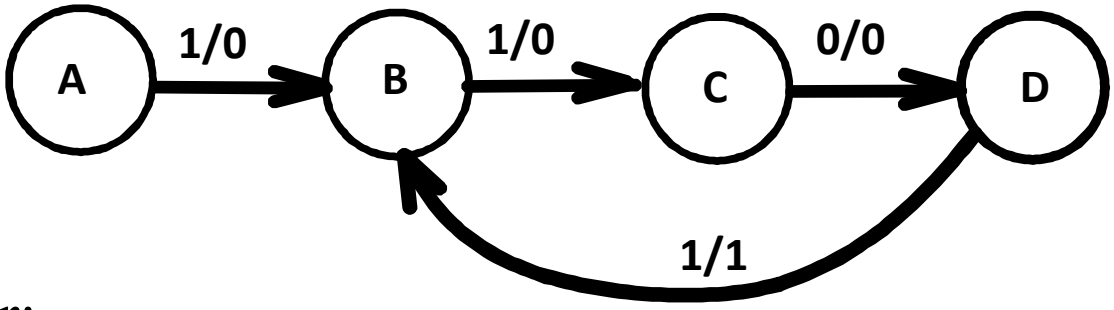
Example: Recognize 1101 (cont'd)



- The state have the following abstract meanings:
  - A: No proper sub-sequence of the 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 has occurred and thus, the sequence is recognized.

Example: Recognize 1101 (cont'd)

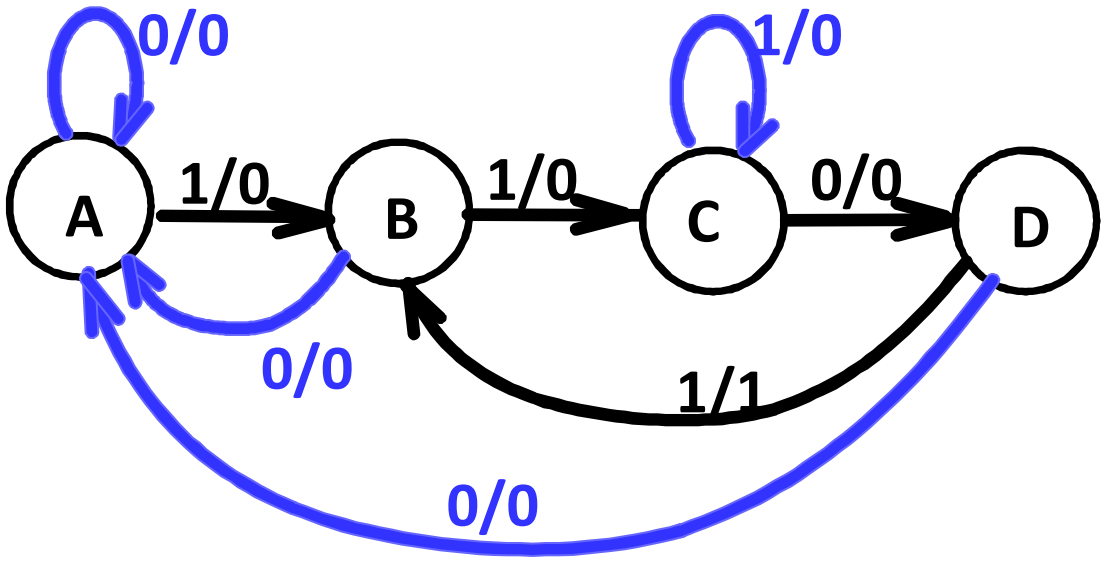
- The other arcs are added to each state for inputs not yet listed. Which arcs are missing?



- Answer:
- "0" arc from A
  "0" arc from B
  "1" arc from C
  "0" arc from D.

Example: Recognize 1101 (cont'd)

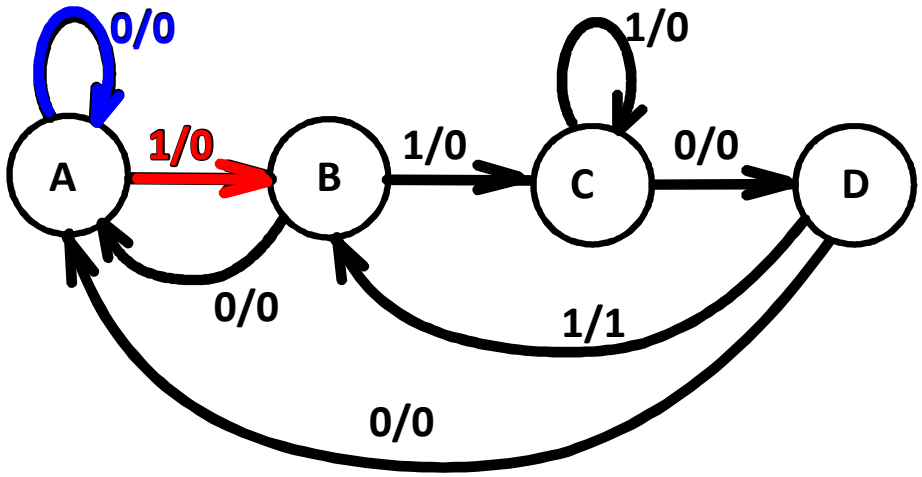
- State transition arcs must represent the fact that an input subsequence has occurred. Thus, we get:



- Note that the 1 arc from state C to state C implies that State C means ***two or more 1's have occurred***.

Formulation: Find State Table

- From the state diagram, we can fill in the state table.
- There are 4 states, one input, and one output. We will choose the form 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	A	C	0	0
C	D	C	0	0
D	A	B	0	1

Note that state table can also be used to merge equivalent states.

What would the state diagram and state table look like for the Moore model?

Example: Moore Model for Sequence 1101

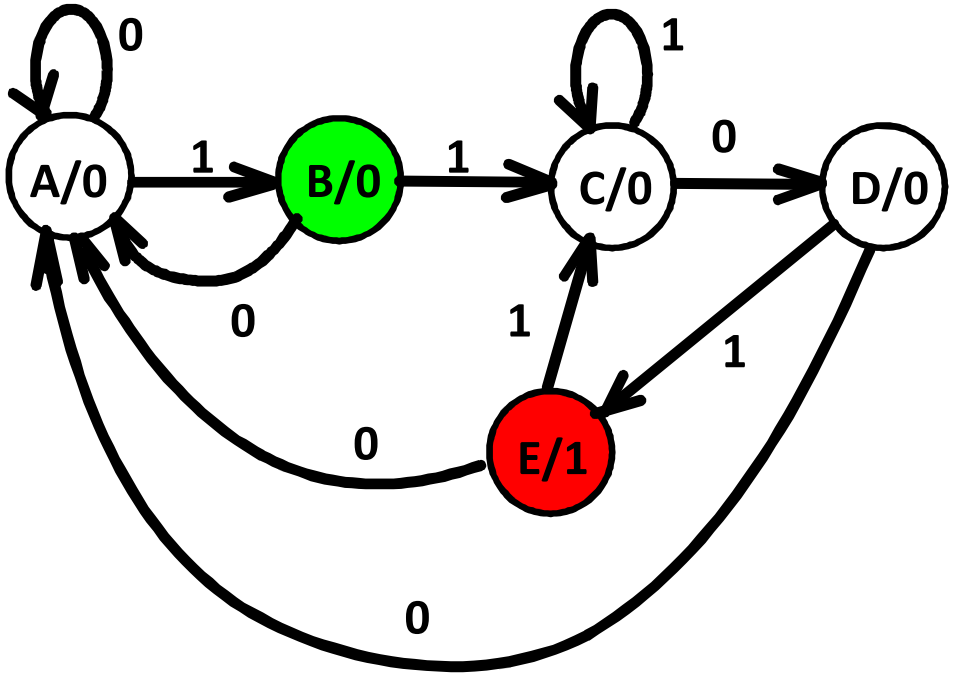
- For the Moore model, outputs are associated with states.

- 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 B.

- The Moore model for a sequence recognizer usually has more states than the Mealy model.

Example: Moore Model (cont'd)

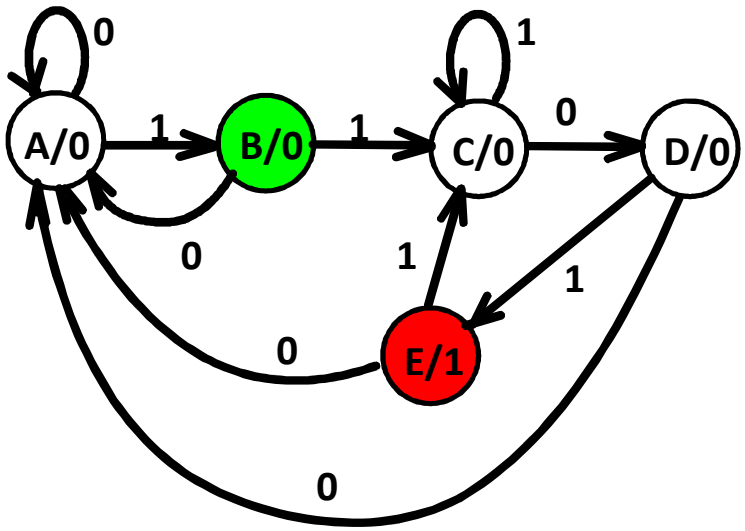
- We mark outputs on states for Moore model
- Arcs now show only state transitions
- Add a new state E to produce the output 1



- Note that the new state, E produces the same behavior in the future as state B. But it gives a different output at the present time. Thus, these states do represent a ***different abstraction*** of the input history.

Example: Moore Model (cont'd)

- Fill in the state table according to the stable diagram
- Memory aid re more state in the Moore model: "Moore is More."



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

State Assignment

- Each of the ***m*** states must be assigned a unique code
- Minimum number of bits required 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
- There are $2^n - m$ unused states

State Assignment: Example 1

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

- How many assignments of codes with a minimum number of bits?
  - Two: A = 0, B = 1 or A = 1, B = 0
- Does it make a difference?
  - Only in variable inversion, so small, if any.

State Assignment: Example 2

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

- How many assignments of codes with a minimum number of bits?

- Does code assignment make a difference in cost?

State Assignment: Example 2 (cont'd)

- Counting Order Assignment: A = 00, B = 01, C = 10, D = 11
- The resulting coded state table:

Present State $Y_1\ Y_2$	Next State $D_1\ D_2$ (x = 0)	Next State $D_1\ D_2$ (x = 1)	Output (x = 0)	Output (x = 1)
0 0	0 0	0 1	0	0
0 1	0 0	1 0	0	0
1 0	1 1	1 0	0	0
1 1	0 0	0 1	0	1

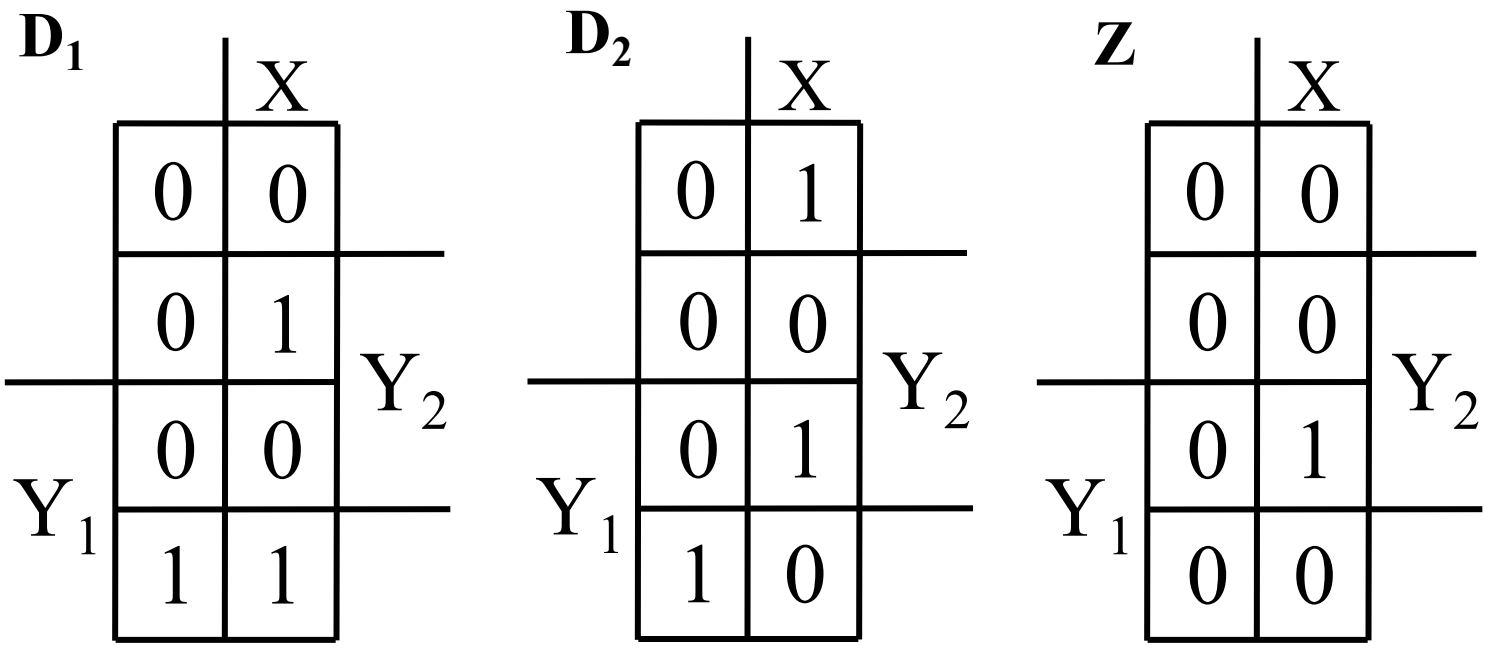
State Assignment: Example 2 (cont'd)

- Gray Code Assignment: A = 00, B = 01, C = 11, D = 10
- The resulting coded state table:

Present State $Y_1\ Y_2$	Next State $x=0$ $D_1\ D_2$	Next State $x=1$ $D_1\ D_2$	Output $x=0$	Output $x=1$
0 0	0 0	0 1	0	0
0 1	0 0	1 1	0	0
1 1	1 0	1 1	0	0
1 0	0 0	0 1	0	1

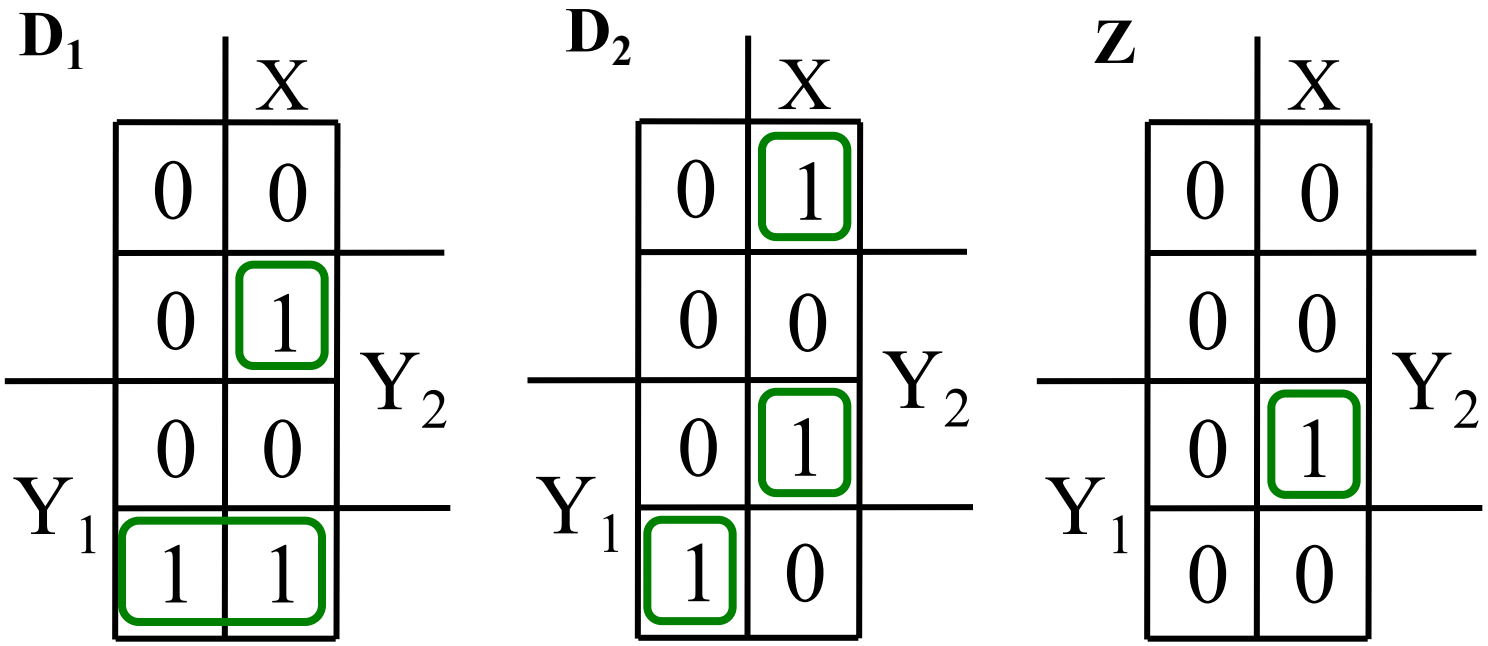
Find Flip-Flop Input and Output Equations: Counting Order Assignment

- Assume D flip-flops
- Interchange the bottom two rows of the state table, to obtain K-maps for D1, D2, and Z:



Optimization: Counting Order Assignment

- Performing two-level optimization:

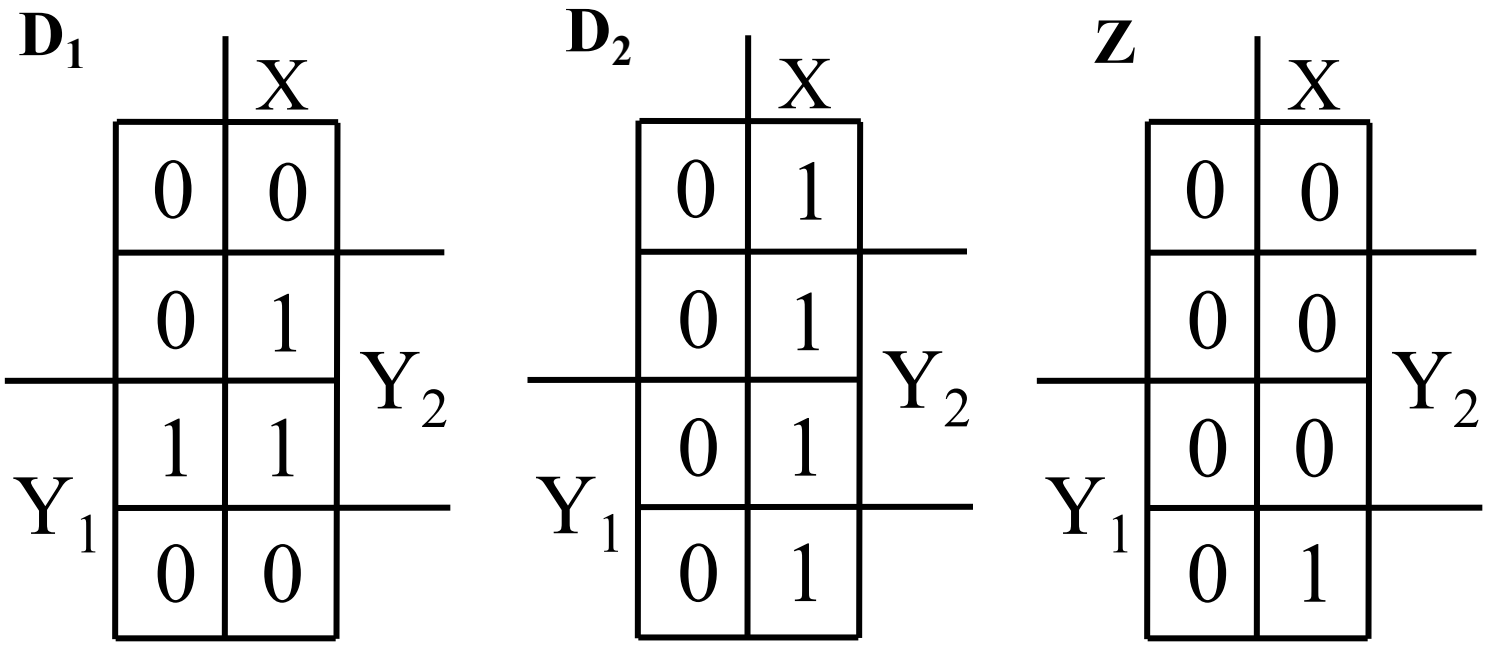


- $D_1 = Y_1\overline{Y}_2 + X\overline{Y}_1Y_2$
  $D_2 = \overline{X}Y_1\overline{Y}_2 + X\overline{Y}_1\overline{Y}_2 + XY_1Y_2$
  $Z = XY_1Y_2$

Gate Input Cost = ?

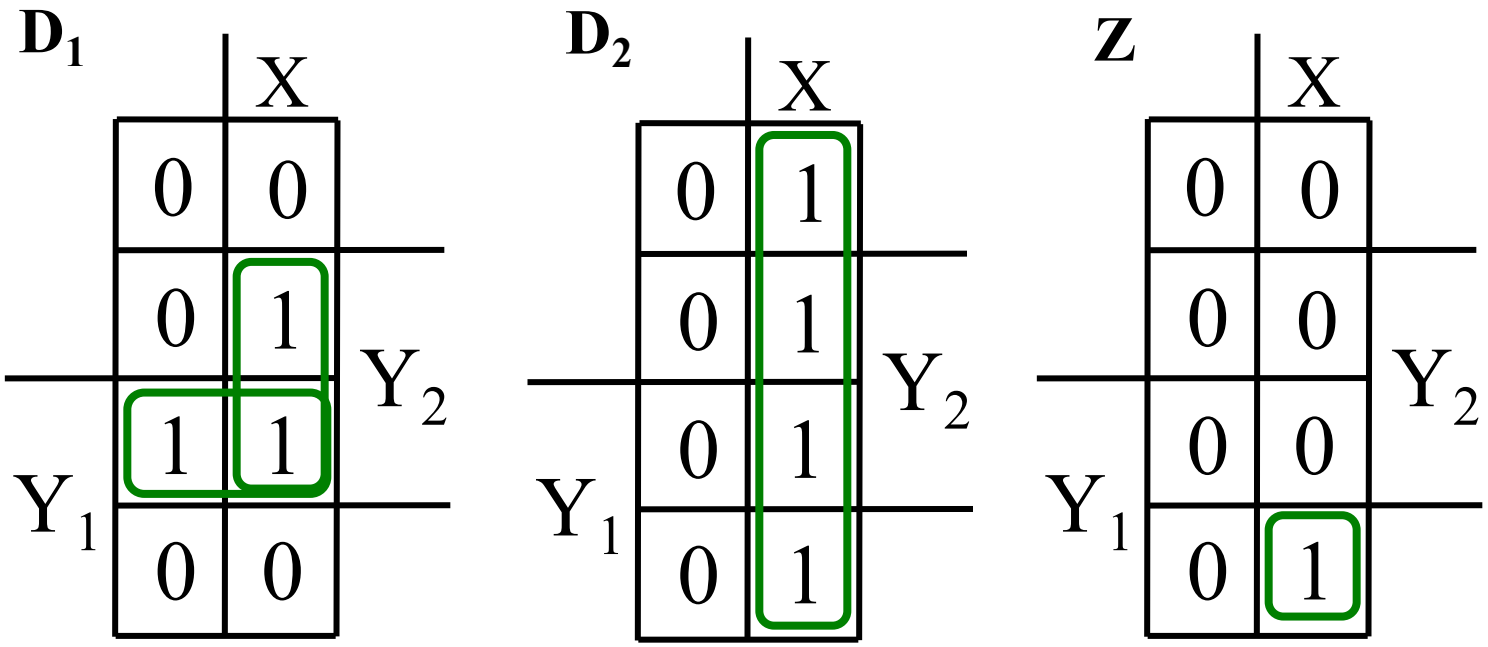
Find Flip-Flop Input and Output Equations: Gray Code Assignment

- Assume D flip-flops
- Obtain K-maps for D1, D2, and Z:



Optimization: Gray Code Assignment

- Performing two-level optimization:



- $D_1 = Y_1Y_2 + XY_2$
  $D_2 = X$
  $Z = XY_1\overline{Y}_2$

Gate Input Cost = ?

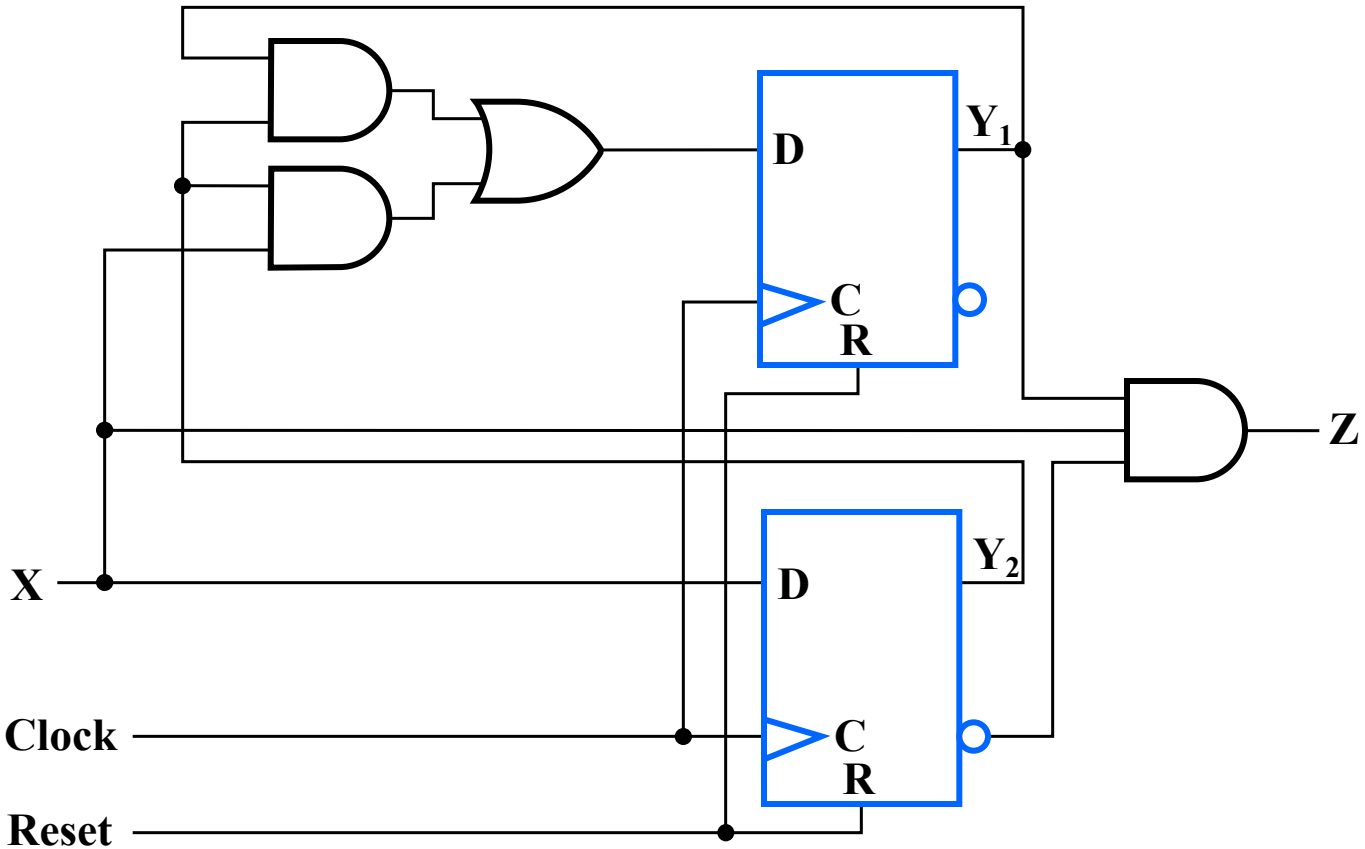
Rules for State Assignment

- States which have the same next state for a given input should be given adjacent assignment.
- States which are the next states of the same state should be given adjacent assignment.
- States which have the same output for a given input should be given adjacent assignment (for output simplification.)

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

Map Technology: Gray Code Assignment

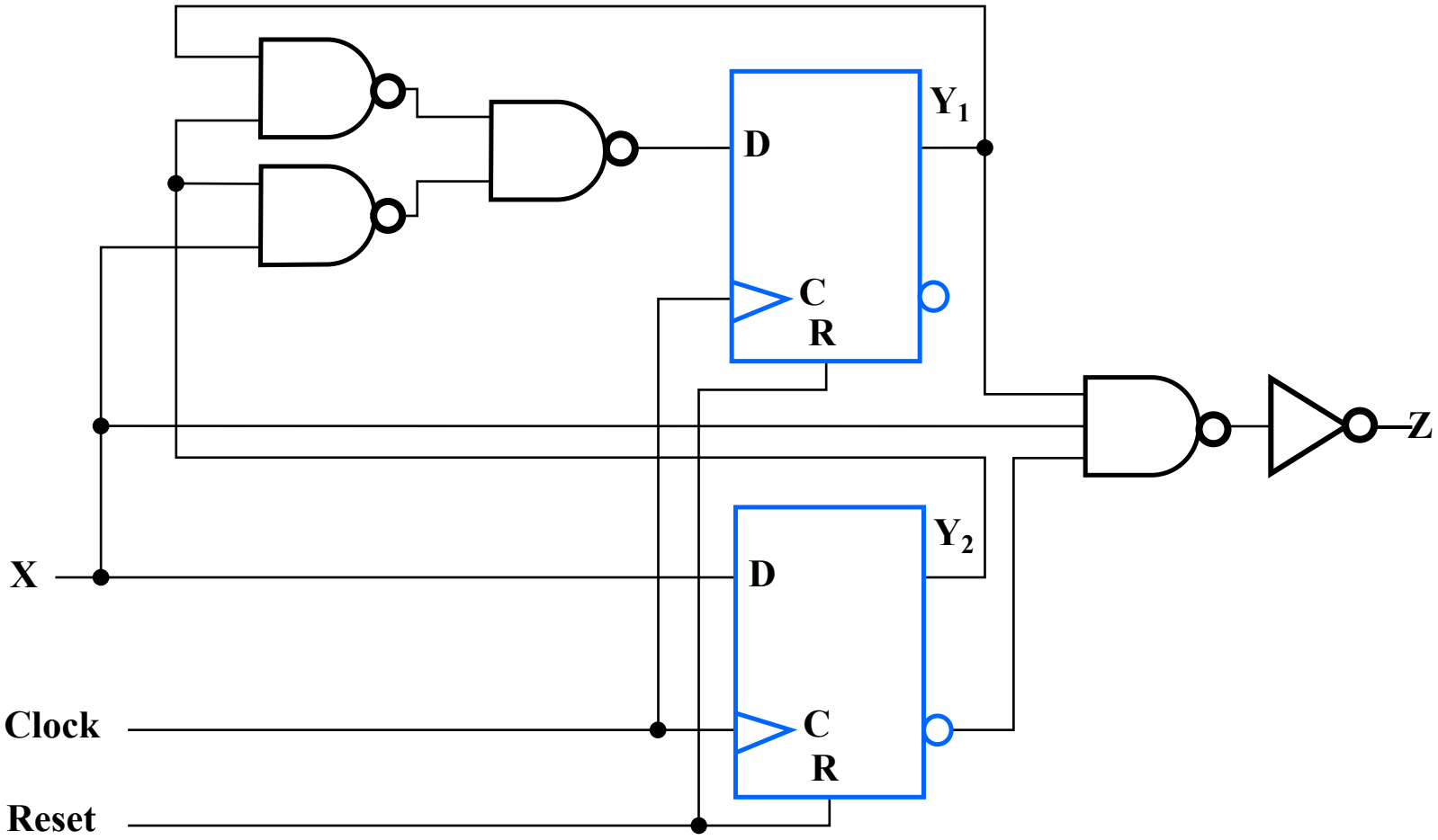
- D Flip-flops with Reset (not inverted)



Initial circuit

Mapped Circuit

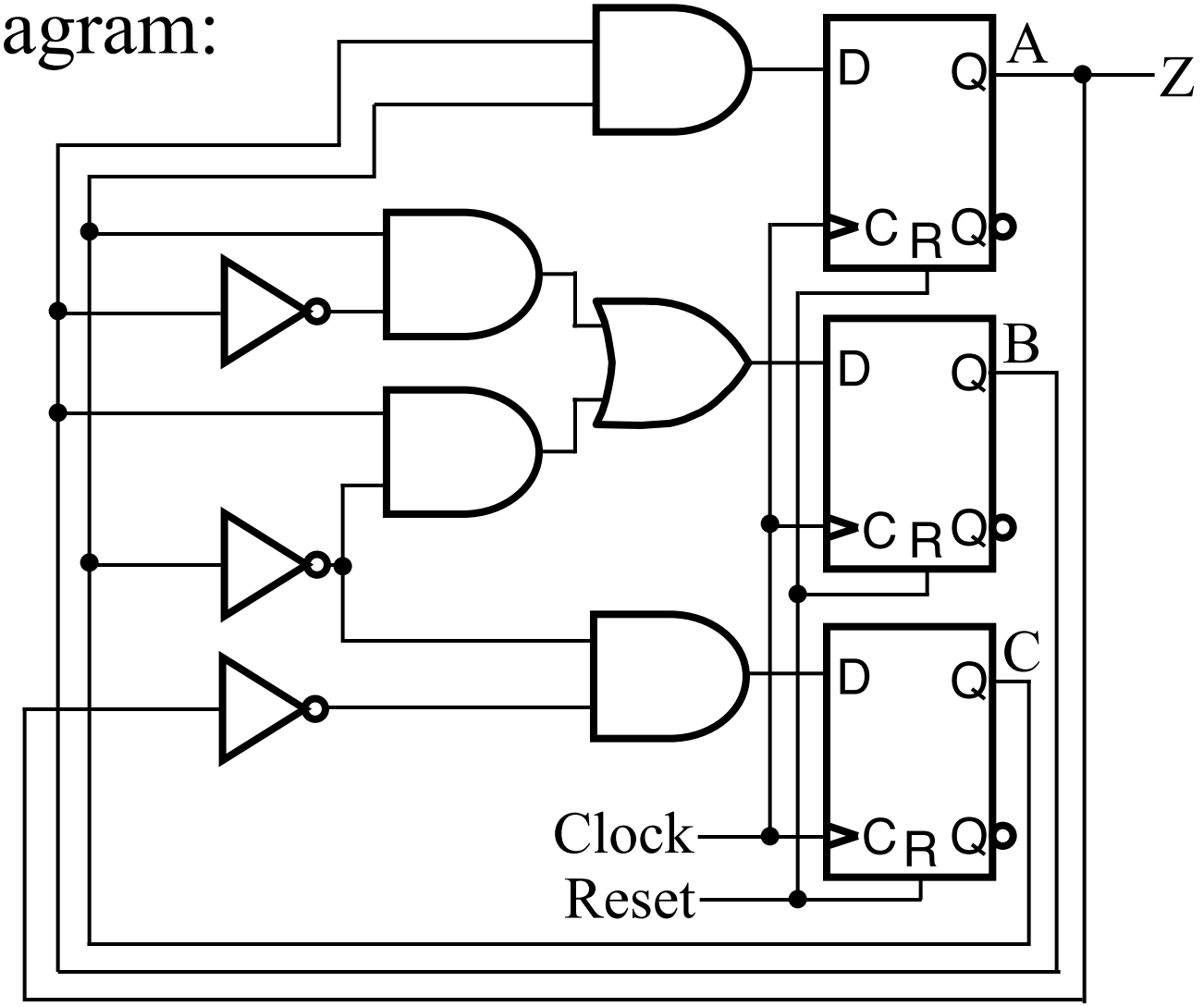
- NAND gates with up to 4 inputs and inverters



Final circuit

Design vs. Analysis: An Example

- Logic diagram:



Flip-Flop Input Equations

- Variables
  - Inputs: None
  - Outputs: Z
  - State Variables: A, B, C
- Initialization: Reset to (0,0,0)
- Equations
  - A(t+1) = Z =
  - B(t+1) =
  - C(t+1) =

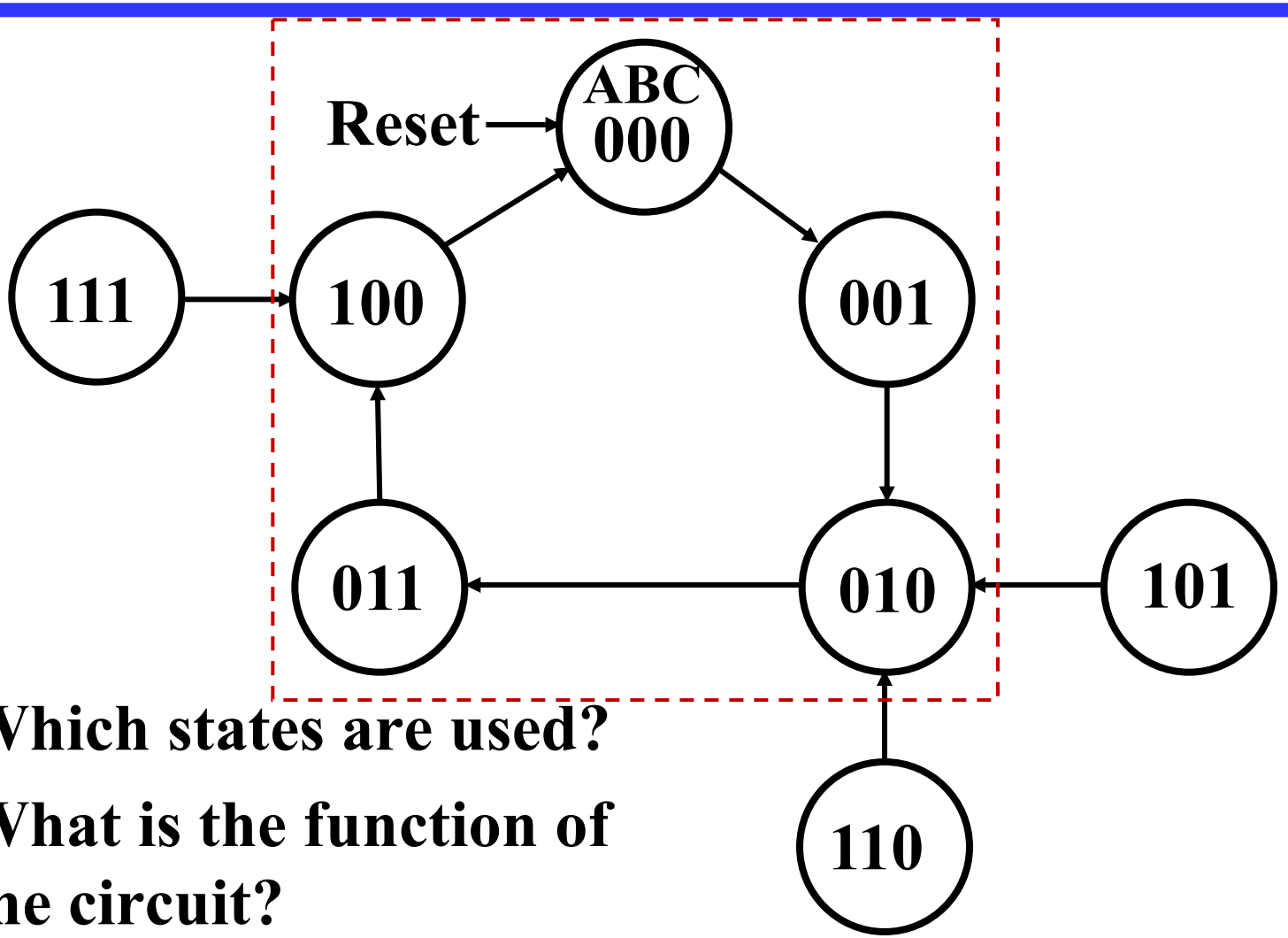
State Table

- $A(t+1) = BC$
- $B(t+1) = \overline{B}C + B\overline{C}$
- $C(t+1) = \overline{A}\overline{C}$
- $Z = A$

- Let $Y=X(t+1)$

A B C	$Y_A Y_B Y_C$	Z
0 0 0	0 0 1	0
0 0 1	0 1 0	0
0 1 0	0 1 1	0
0 1 1	1 0 0	0
1 0 0	0 0 0	1
1 0 1	0 1 0	1
1 1 0	0 1 0	1
1 1 1	1 0 0	1

State Diagram



- Which states are used?
- What is the function of the circuit?

Unused States: The State Assignment Problem

- The previous example has only 5 states, represented by 3-bit values. So, we have three unused state assignments in this case.

- This is no real problem, except that at power up, the machine may start up in one of the unused states. Or, it could get into an unused state due to electrical noise (lightning, bad connections, etc).

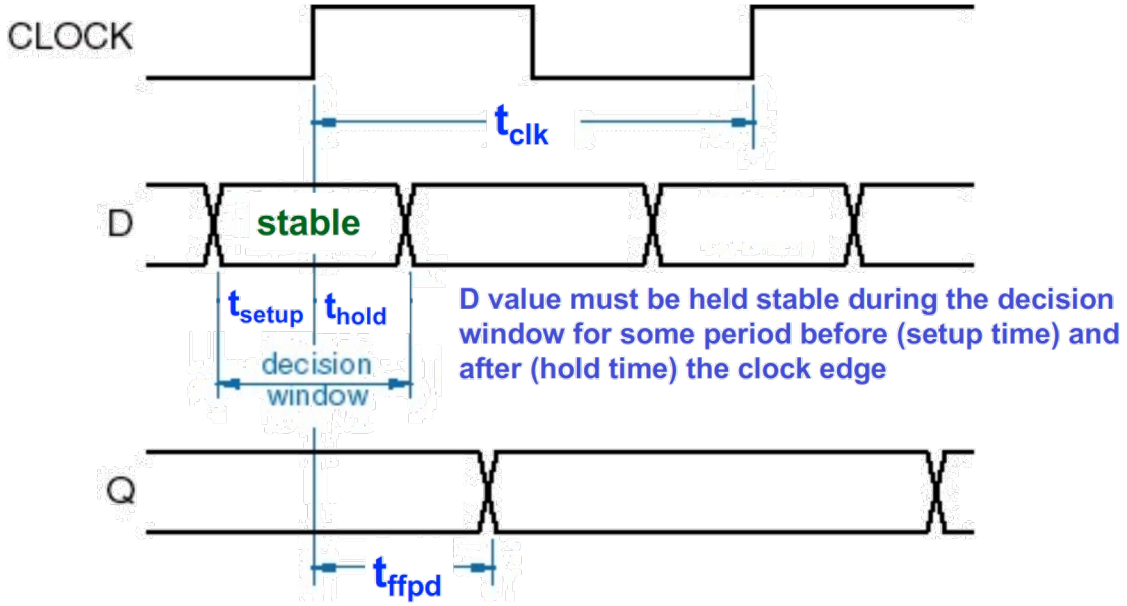
- One possibility is to add **RESET** logic to force the machine into state 000 on power up, or on detect of unused state. If entry to an unused state is considered fatal, then stop the FSM (force it into an ERROR state, where it stays) and transmit an error code.

Unused States: In the hardware Implementation of the FSM

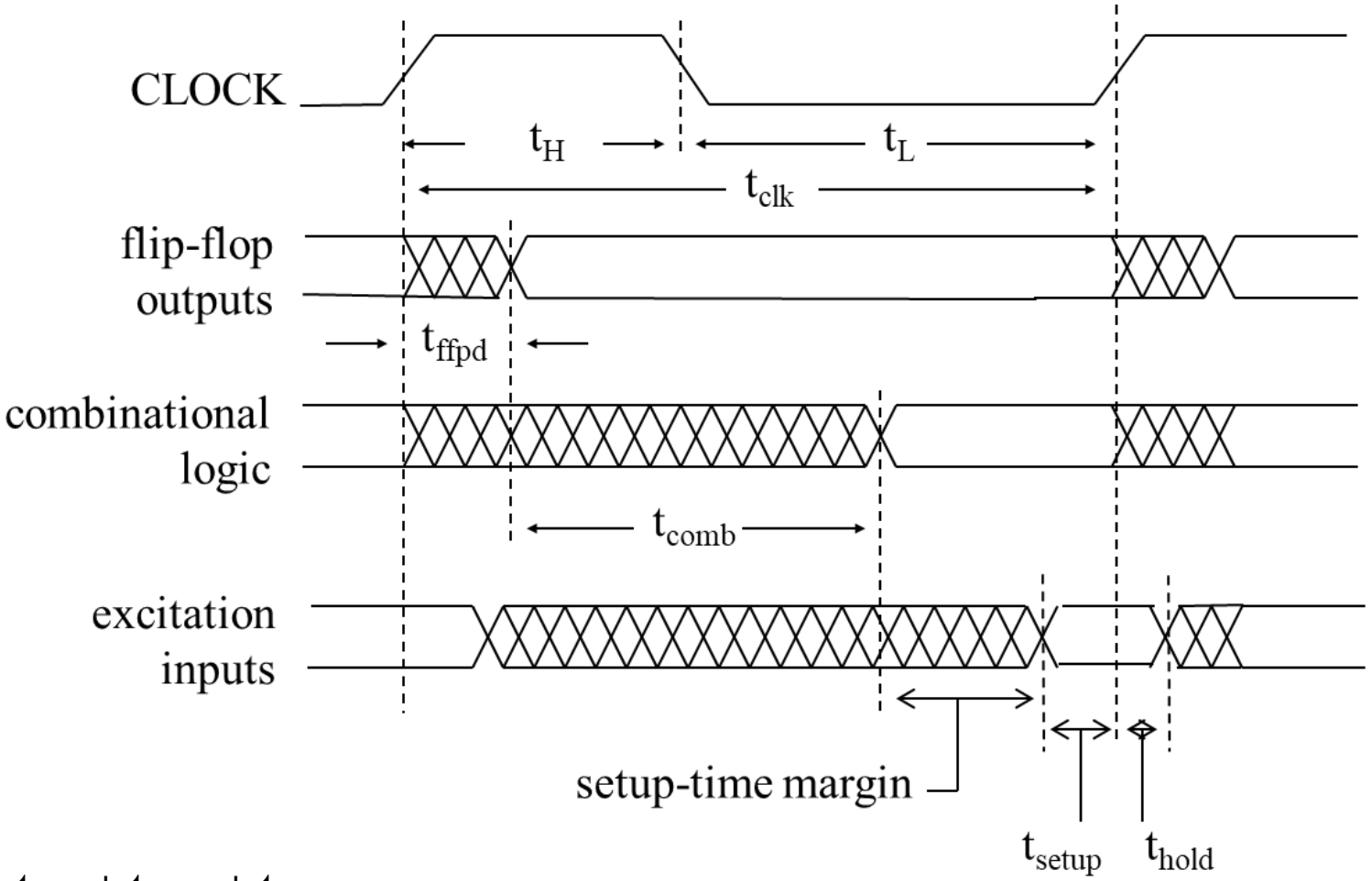
- A nasty problem is that of the machine appearing to **freeze up** if it gets into an unused state. This can happen if the unused states point to themselves or each other.
- If all unused states point to used states, the machine is said to be "**self starting**" and, after power up, will function correctly. If the machine is self-starting, entering an unused state will cause a brief erratic operation, then the machine will resume correct operation.
- So, if the application is fault-tolerant, and the machine is self-starting, we are OK to ignore the problem beyond verifying self start capability.
- We can guarantee self start (assuming state 000 is a state we are using) by replacing the Xs in the implementation diagram above with 0s. This will force the machine into state 000 whenever an unused state is entered. The problem, of course, is more complex steering logic.

Flip-Flop Timing

- E.g., D flip-flop
  - t_{setup} - required time of stable input before CLK, input before CLK
  - t_{hold} - required time of stable input after CLK, input after CLK
  - t_{ffpd} - flip-flop propagation delay after CLK, aka Clk-to-Q time



Timing Diagram of Sequential Circuits



$t_{clk} \geq t_{ffpd} + t_{comb} + t_{setup}$

$t_{hold} \leq t_{ffpd} + t_{comb}$

More Materials (Optional)

- **State Table & State Equation**
- **Basic Flip-Flop Descriptors**
- **Flip-Flop Behaviors**

More Materials (Optional)

- **State Table & State Equation**
- **Basic Flip-Flop Descriptors**
- **Flip-Flop Behaviors**

State Table: Two Alternative Forms

Present State		Input	Next State		Output
A	*B*	*x*	*A*	*B*	*y*
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0

Present State	Next State		Output	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
AB	*AB*	*AB*	*y*	*y*
00	00	01	0	0
01	00	11	1	0
10	00	10	1	0
11	00	10	1	0

State Equation

- The behavior of a clocked sequential circuit can be described algebraically by means of state equations.

- A state equation (aka next-state equation) specifies the next state as a function of the present state and inputs.

- Specifically, a state equation is an algebraic expression that specifies the condition for a flip-flop state transition. The left side of the equation with (t+1) denotes the next state of the flip-flop one clock edge later. The right side of the equation is Boolean expression that specifies the present state and input conditions that make the next state equal to 1.

More Materials (Optional)

- State Table & State Equation
- **Basic Flip-Flop Descriptors**
- Flip-Flop Behaviors

Basic Flip-Flop Descriptors

- Used in analysis
  - *Characteristic table* - defines the next state of the flip-flop in terms of flip-flop inputs and current state
  - *Characteristic equation* - defines the next state of the flip-flop as a Boolean function of the flip-flop inputs and the current state
- Used in design
  - *Excitation table* - defines the flip-flop input variable values as function of the current state and next state
  - *Excitation equation* (aka *input equation*) - defines the combinational logic that drives the flip-flops

S-R Flip-Flop Descriptors

- 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

- Characteristic Equation

$$Q(t+1) = S + \overline{R} \cdot Q(t),\ S \cdot R = 0$$

- 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

D Flip-Flop Descriptors

- Characteristic Table

D	Q(t+1)	Operation
0	0	Reset
1	1	Set

- Characteristic Equation

$$Q(t+1) = D$$

- Excitation Table

Q(t +1)	D	Operation
0	0	Reset
1	1	Set

J-K Flip-Flop Descriptors

- 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

- Characteristic Equation

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

- 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

T Flip-Flop Descriptors

- Characteristic Table

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

- Characteristic Equation

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

- Excitation Table

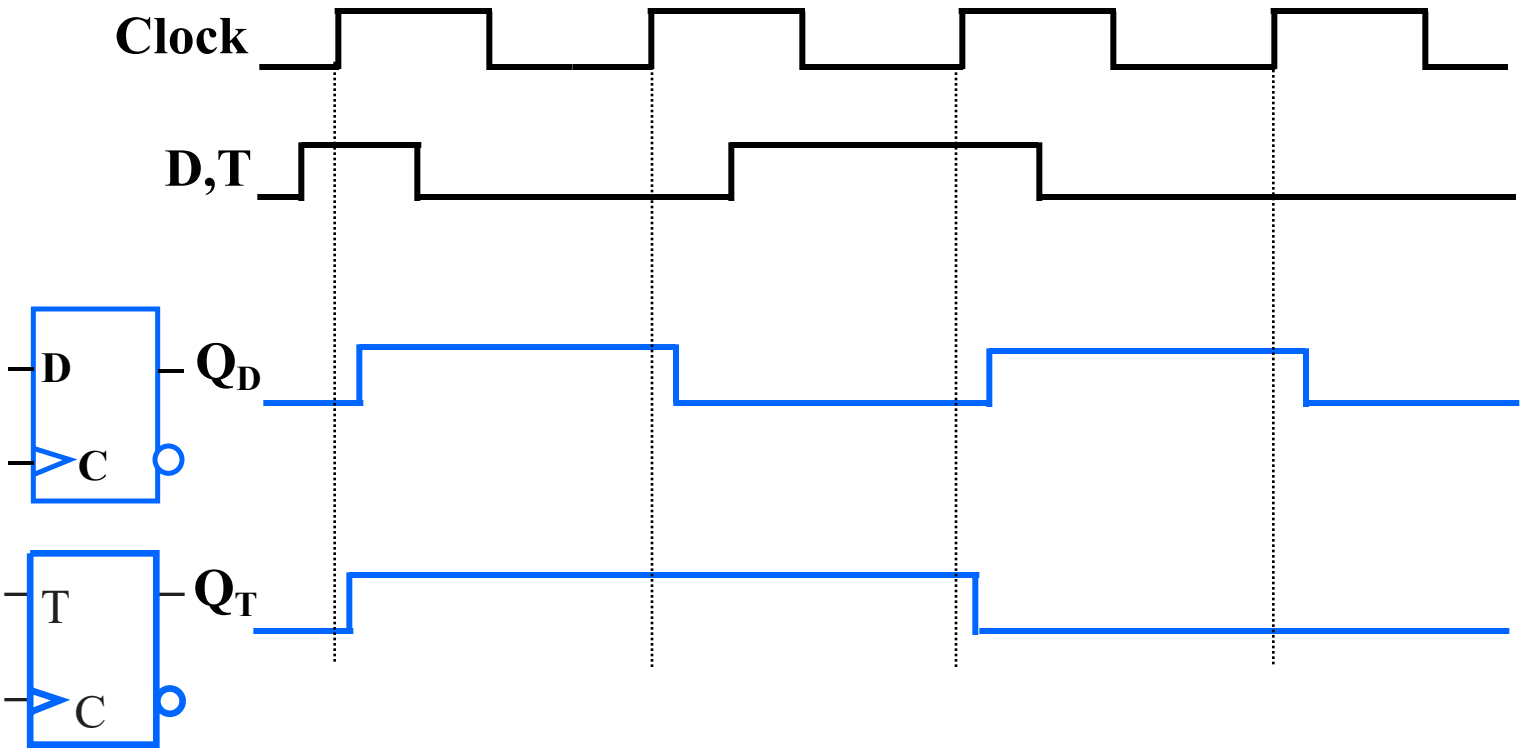
Q(t+1)	T	Operation
$Q(t)$	0	No change
$\overline{Q}(t)$	1	Complement

More Materials (Optional)

- State Table & State Equation
- Basic Flip-Flop Descriptors
- **Flip-Flop Behaviors**

Flip-flop Behavior Example

- Use the characteristic tables to find the output waveforms for the flip-flops shown:



Flip-Flop Behavior Example (continued)

- Use the characteristic tables to find the output waveforms for the flip-flops shown: