Chapter 4
Sequential Circuits

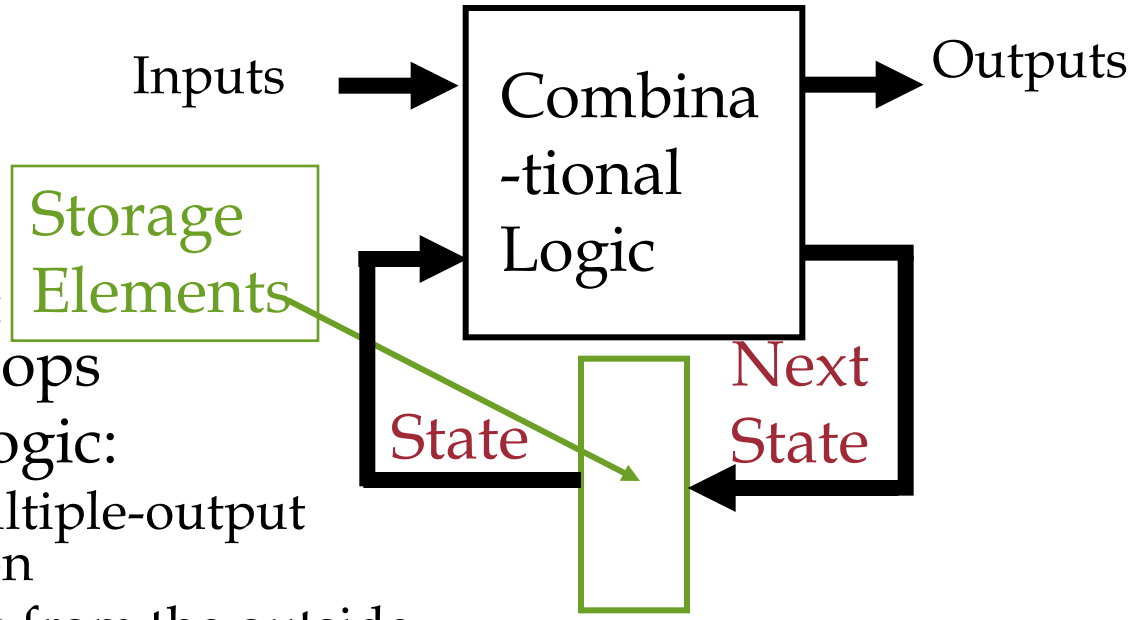
Dosen: Erdefi Rakun danTim Dosen PSD

Fasilkom UI

Overview

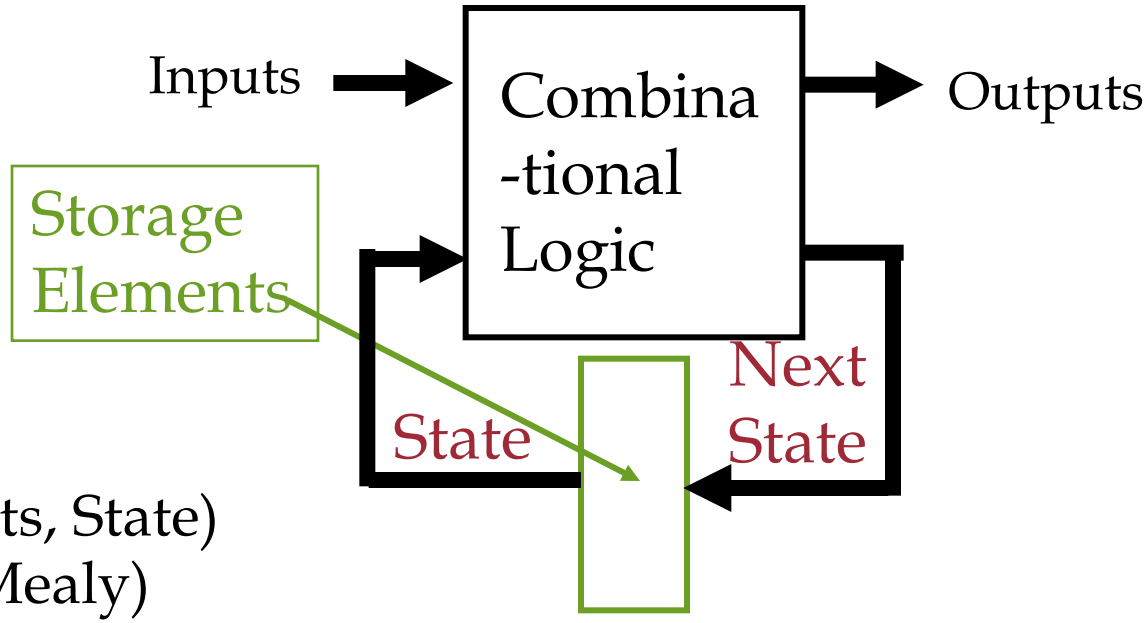
- Part 1 - Storage Elements and Analysis
  - Introduction to sequential circuits
  - Types of sequential circuits
  - Storage elements
    - Latches
    - Flip-flops
  - Sequential circuit analysis
    - State tables
    - State diagrams
    - Equivalent states
    - Moore and Mealy Models
- Part 2- Sequential Circuit Design
    - Specification
    - Formulation
    - State Assignment
    - Flip-Flop Input and Output Equation Determination
    - Verification

4.1. Sequential Circuit Definitions



- A Sequential circuit contains:
  - Storage elements: Latches or Flip-Flops
  - Combinational Logic:
    - Implements a multiple-output switching function
    - Inputs are signals from the outside.
    - Outputs are signals to the outside.
    - Other inputs, State or Present State, are signals from storage elements.
    - The remaining outputs, Next State are inputs to storage elements.

Introduction to Sequential Circuits



- Combinatorial Logic
  - *Next state function*
    Next State = f(Inputs, State)
  - *Output function* (Mealy)
    Outputs = g(Inputs, State)
  - *Output function* (Moore)
    Outputs = h(State)
- Output function type depends on specification and affects the design significantly

Types of Sequential Circuits

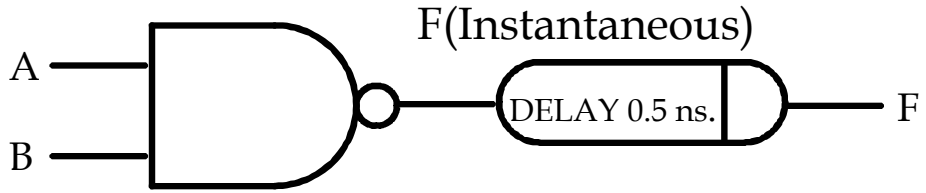
- Depends on the times at which:
  - storage elements observe their inputs, and
  - storage elements change their state
- Synchronous
  - Behavior defined from knowledge of its signals at discrete instances of time
  - Storage elements observe inputs and can change state only in relation to a timing signal (clock pulses from a clock)
- Asynchronous
  - Behavior defined from knowledge of inputs an any instant of time and the order in continuous time in which inputs change
  - If clock just regarded as another input, all circuits are asynchronous!
  - Nevertheless, the synchronous abstraction makes complex designs tractable!

Discrete Event Simulation

- In order to understand the time behavior of a sequential circuit we use discrete event simulation.
- Rules:
  - Gates modeled by an ideal (instantaneous) function and a fixed gate delay
  - Any change in input values is evaluated to see if it causes a change in output value
  - Changes in output values are scheduled for the fixed gate delay after the input change
  - At the time for a scheduled output change, the output value is changed along with any inputs it drives

Simulated NAND Gate

- Example: A 2-Input NAND gate with a 0.5 ns. delay:

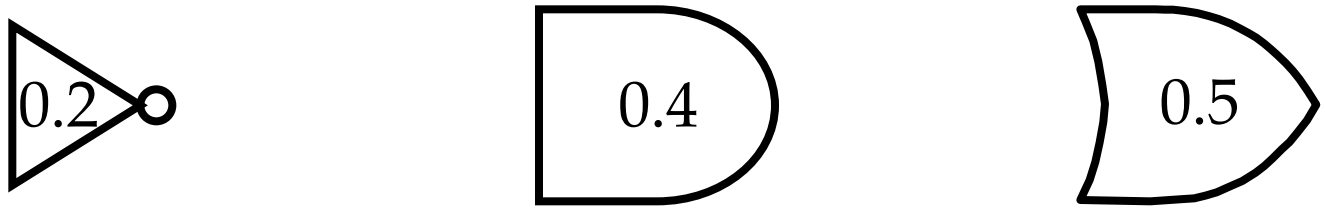


- Assume A and B have been 1 for a long time
- At time t=0, A changes to a 0 at t= 0.8 ns, back to 1.

t (ns)	A	B	F(I)	F	Comment
$-\infty$	1	1	0	0	A=B=1 for a long time
0	1⇨ 0	1	1⇦ 0	0	F(I) changes to 1
0.5	0	1	1	1⇦ 0	F changes to 1 after a 0.5 ns delay
0.8	1⇦ 0	1	1⇨ 0	1	F(Instantaneous) changes to 0
1.3	1	1	0	1⇨ 0	F changes to 0 after a 0.5 ns delay

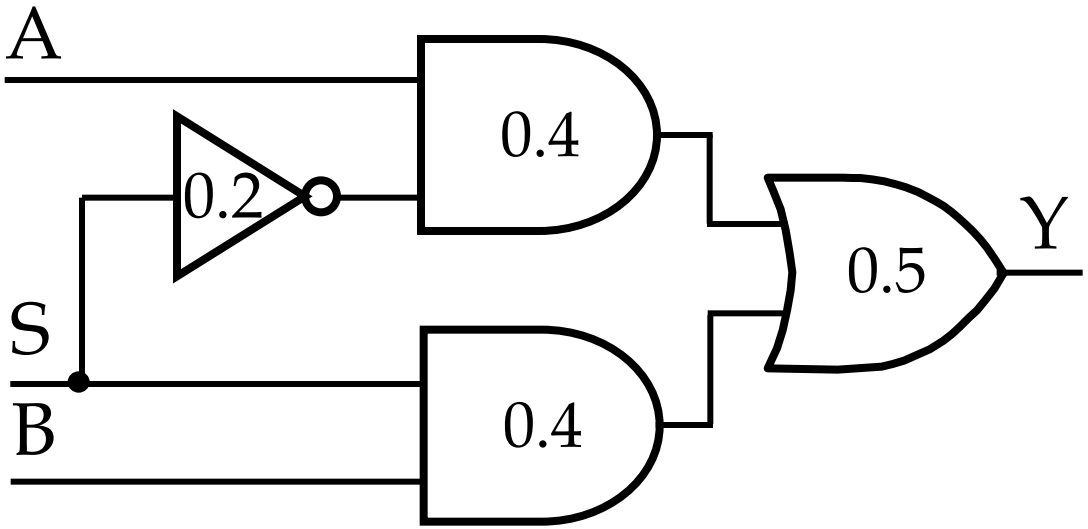
Gate Delay Models

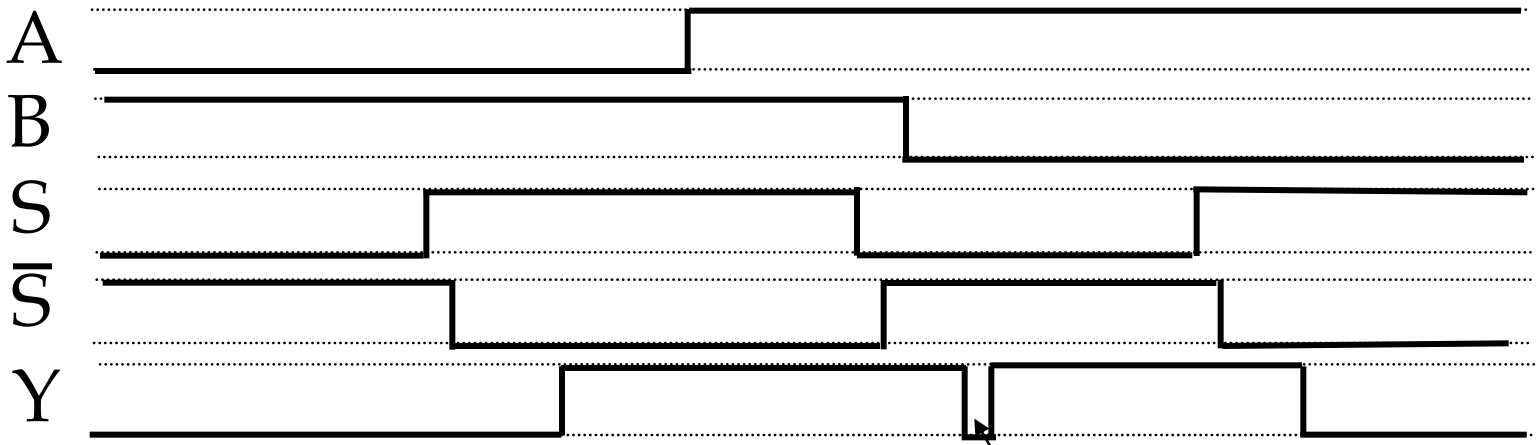
- Suppose gates with delay n ns are represented for n = 0.2 ns, n = 0.4 ns, n = 0.5 ns, respectively:



Circuit Delay Model

- Consider a simple 2-input multiplexer:
- With function:
  - Y = A for S = 0
  - Y = B for S = 1

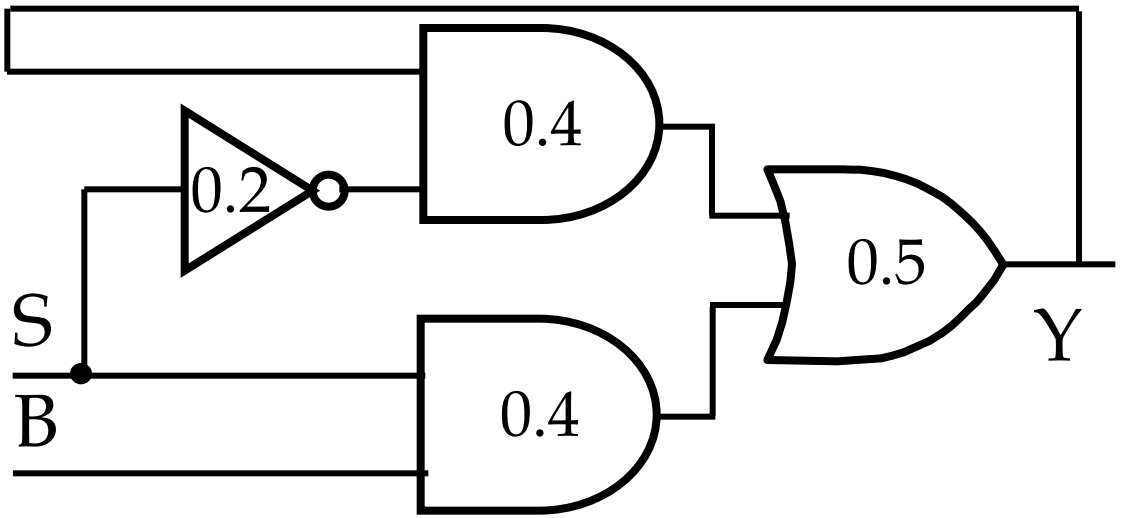


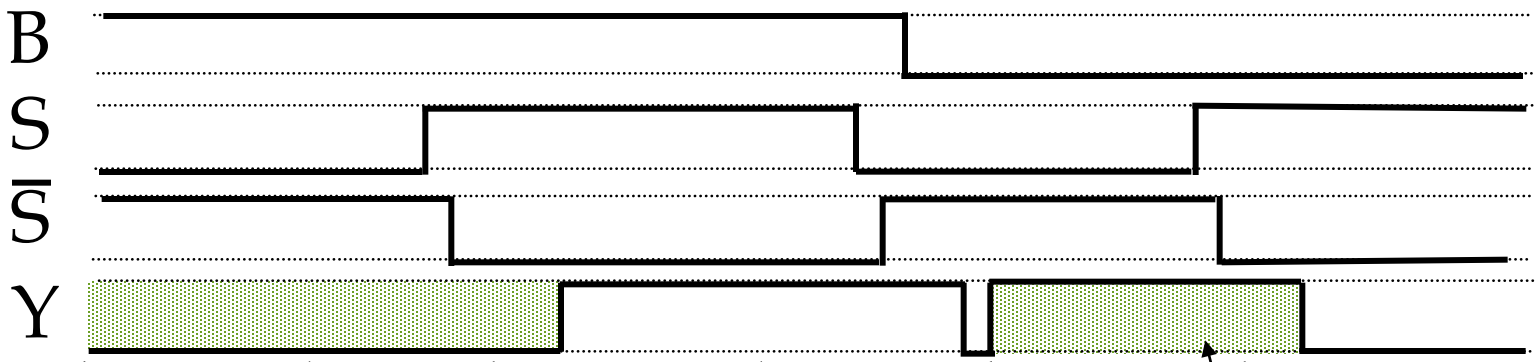


- "Glitch" is due to delay of inverter

Storing State

- What if A connected to Y?
- Circuit becomes:
- With function:
  - Y = B for S = 1, and Y(t) dependent on Y(t – 0.9) for S = 0





- The simple combinational circuit has now become a sequential circuit because its output is a function of a time sequence of input signals!

Y is stored value in shaded area

Storing State (Continued)

- Simulation example as input signals change with time. Changes occur every 100 ns, so that the tenths of ns delays are negligible.

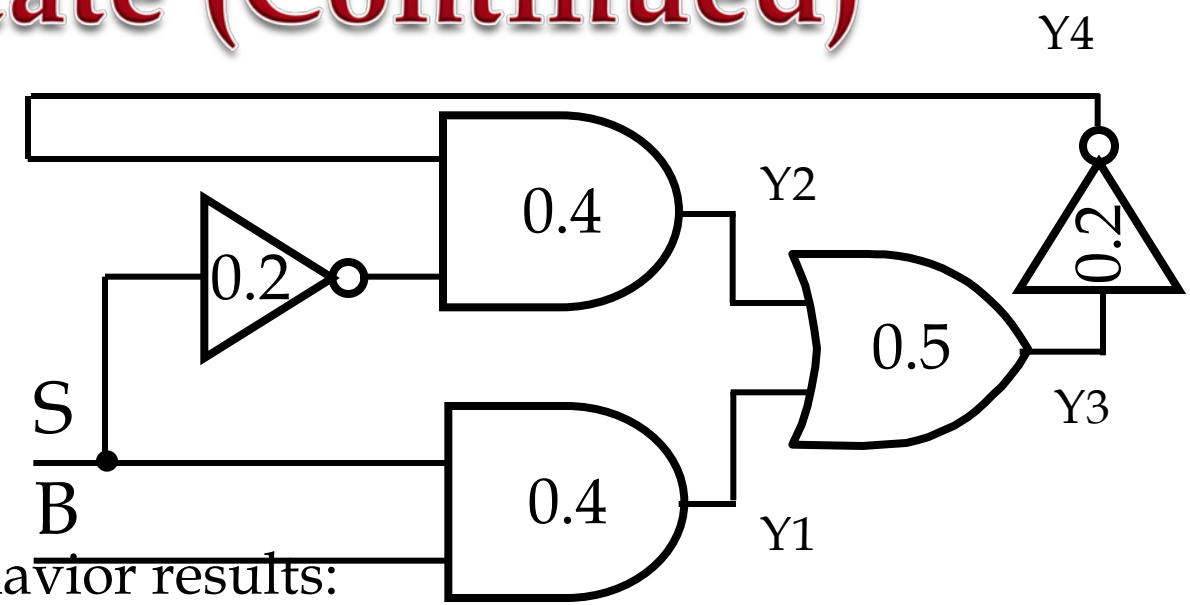
Time ↓

B	S	Y	Comment
1	0	0	Y "remembers" 0
1	1	1	Y = B when S = 1
1	0	1	Now Y "remembers" B = 1 for S = 0
0	0	1	No change in Y when B changes
0	1	0	Y = B when S = 1
0	0	0	Y "remembers" B = 0 for S = 0
1	0	0	No change in Y when B changes

- Y represent the state of the circuit, not just an output.

Storing State (Continued)

- Suppose we place an inverter in the "feedback path."

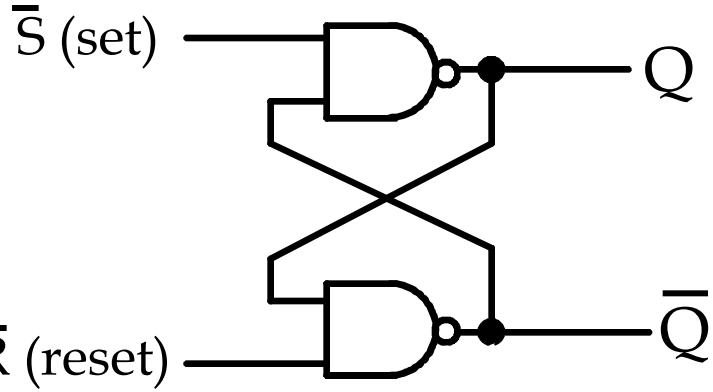


- The following behavior results:
- The circuit is said to be unstable.
- For S = 0, the circuit has become what is called an *oscillator*. Can be used as crude clock.

B	S	Y3	Comment
0	1	0	Y = B when S = 1
1	1	1	
1	0	1	Now Y "remembers" A
1	0	0	Y, 1.1 ns later
1	0	1	Y, 1.1 ns later
1	0	0	Y, 1.1 ns later

Basic (NAND) S – R Latch

- "Cross-Coupling" two NAND gates gives the $\bar{S}$ -$\bar{R}$ Latch:
- Which has the time sequence behavior:



Time

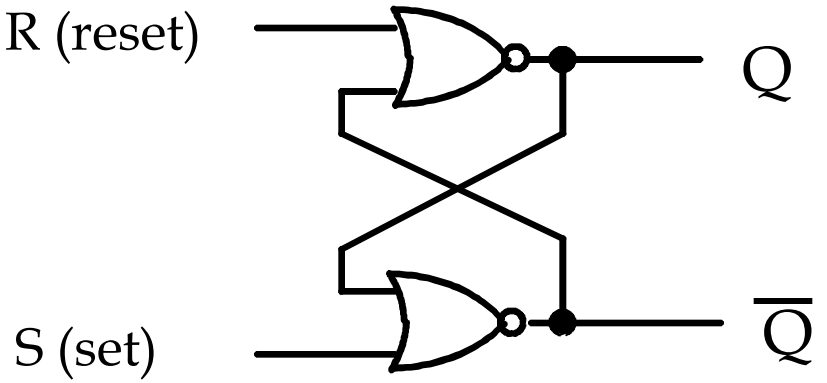
$\bar{R}$	$\bar{S}$	Q	$\bar{Q}$	Comment
1	1	?	?	Stored state unknown
1	0	1	0	"Set" Q to 1
1	1	1	0	Now Q "remembers" 1
0	1	0	1	"Reset" Q to 0
1	1	0	1	Now Q "remembers" 0
0	0	1	1	Both go high
1	1	?	?	Unstable!

- S = 0, R = 0 is <u>forbidden</u> as input pattern

4-2 Latches

Basic (NOR) S – R Latch

- Cross-coupling two NOR gates gives the S – R Latch:
- Which has the time sequence behavior:

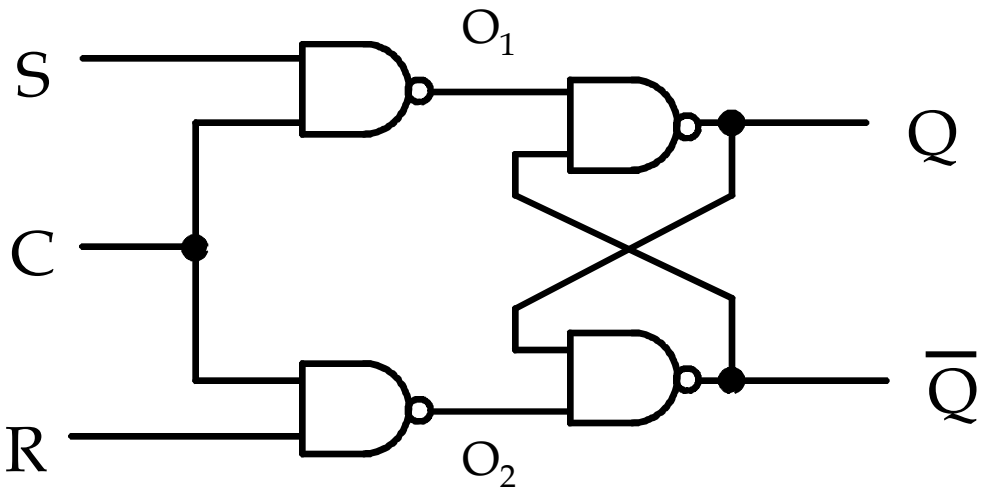


Time

R	S	Q	$\overline{Q}$	Comment
0	0	?	?	Stored state unknown
0	1	1	0	"Set" Q to 1
0	0	1	0	Now Q "remembers" 1
1	0	0	1	"Reset" Q to 0
0	0	0	1	Now Q "remembers" 0
1	1	0	0	Both go low
0	0	?	?	Unstable!

Clocked S - R Latch

- Adding two NAND gates to the basic $\overline{S}$ - $\overline{R}$ NAND latch gives the clocked S – R latch:



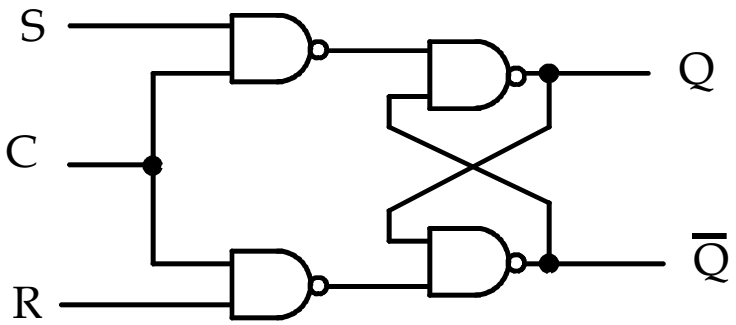
- Has a time sequence behavior similar to the basic S-R latch except that the S and R inputs are only observed when the line C is high.
- C means "control" or "clock".

Clocked S - R Latch

Clock	S	R	O1	O2	Q(t)
0	0	0	1	1	Q(t-1)
0	0	1	1	1	Q(t-1)
0	1	0	1	1	Q(t-1)
0	1	1	1	1	Q(t-1)
1	0	1	1	0	Reset
1	0	0	1	1	Q(t-1)
1	1	0	0	1	Set
1	0	0	1	1	Q(t-1)
1	1	1	0	0	Forbidden

Clocked S - R Latch (continued)

- The Clocked S-R Latch can be described by a table:

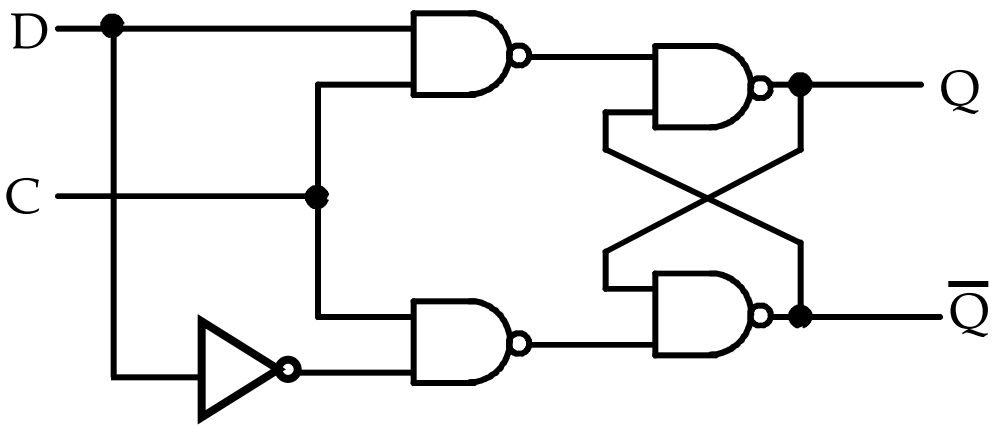


Q(t)	S	R	Q(t+1)	Comment
0	0	0	0	No change
0	0	1	0	Clear Q
0	1	0	1	Set Q
0	1	1	???	Indeterminate
1	0	0	1	No change
1	0	1	0	Clear Q
1	1	0	1	Set Q
1	1	1	???	Indeterminate

- The table describes what happens after the clock [at time (t+1)] based on:
  - current inputs (S,R) and
  - current state Q(t).

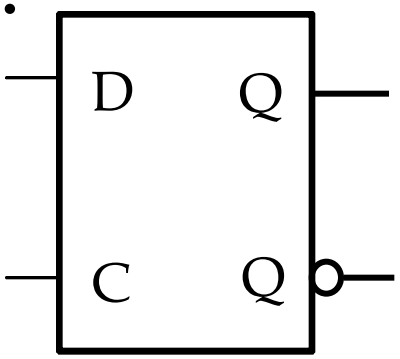
D Latch

- Adding an inverter to the S-R Latch, gives the D Latch:
- Note that there are no "indeterminate" states!



Q	D	Q(t+1)	Comment
0	0	0	No change
0	1	1	Set Q
1	0	0	Clear Q
1	1	1	No Change

The graphic symbol for a D Latch is:



4-3 Flip-Flops

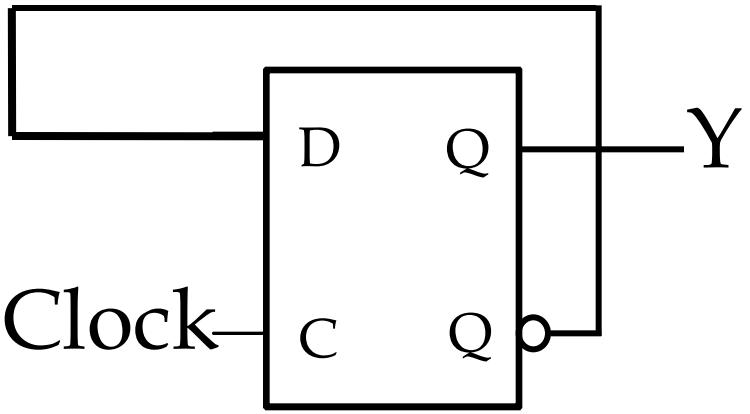
- The latch timing problem
- Master-slave flip-flop
- Edge-triggered flip-flop
- Standard symbols for storage elements
- Direct inputs to flip-flops

The Latch Timing Problem

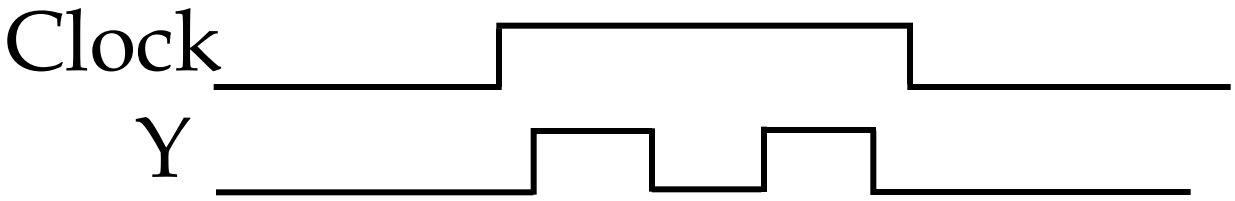
- In a sequential circuit, paths may exist through combinational logic:
  - From one storage element to another
  - From a storage element back to the same storage element
- The combinational logic between a latch output and a latch input may be as simple as an interconnect
- For a clocked D-latch, the output Q depends on the input D whenever the clock input C has value 1

The Latch Timing Problem (continued)

- Consider the following circuit:



- Suppose that initially Y = 0.



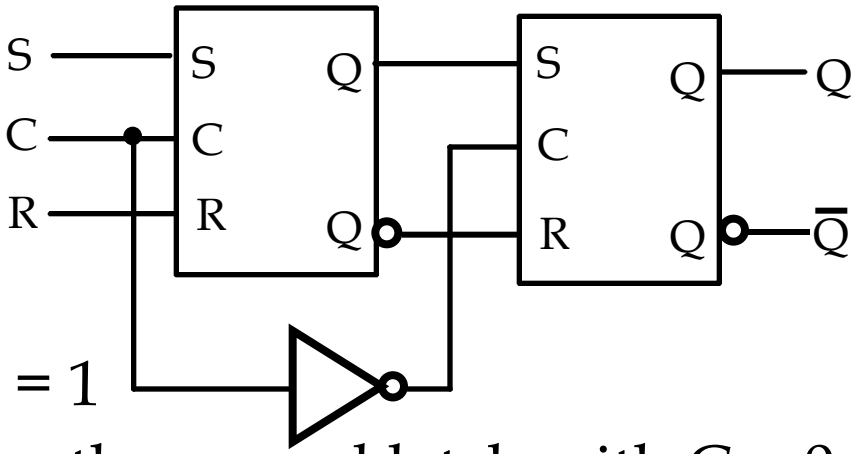
- As long as C = 1, the value of Y continues to change!
- The changes are based on the delay present on the loop through the connection from Y back to Y.
- This behavior is clearly unacceptable.
- Desired behavior: Y changes only once per clock pulse

The Latch Timing Problem (continued)

- A solution to the latch timing problem is to break the closed path from Y to Y within the storage element
- The commonly-used, path-breaking solutions replace the clocked D-latch with:
  - a master-slave flip-flop
  - an edge-triggered flip-flop

S-R Master-Slave Flip-Flop

- Consists of two clocked S-R latches in series with the clock on the second latch inverted
- The input is observed by the first latch with C = 1



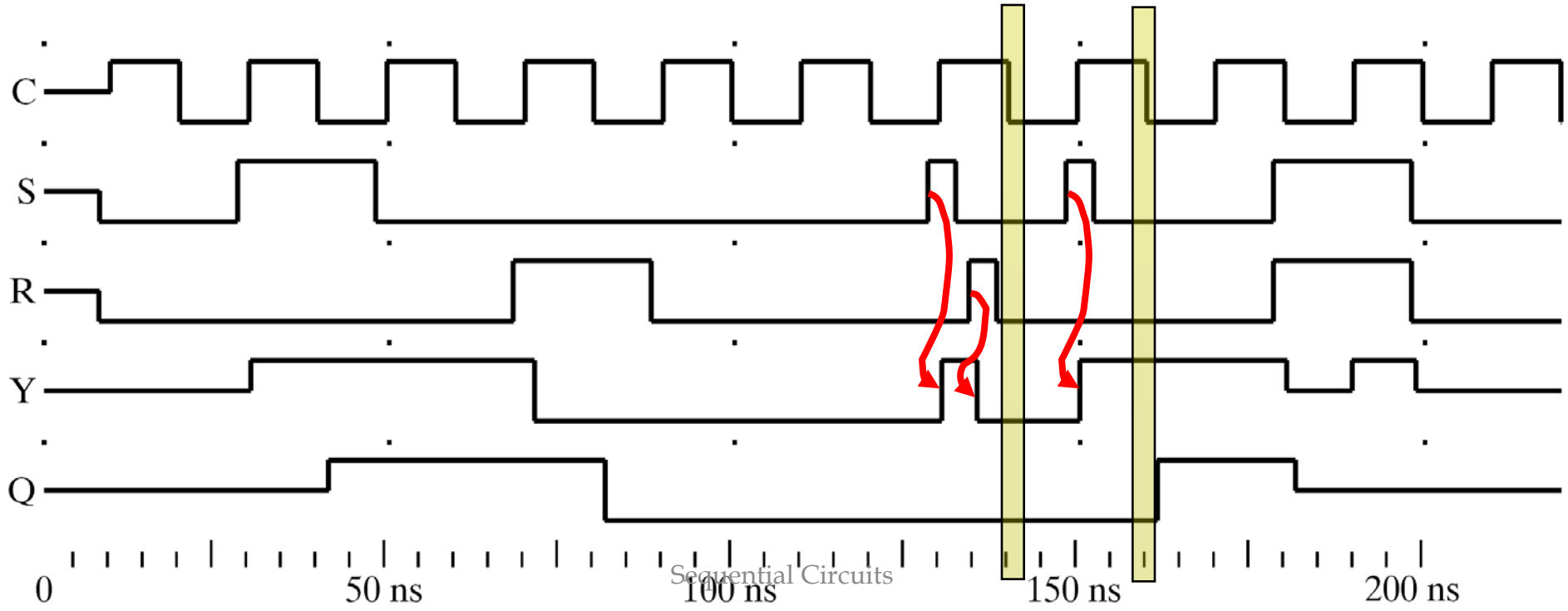
- The output is changed by the second latch with C = 0
- The path from input to output is broken by the difference in clocking values (C = 1 and C = 0).
- The behavior demonstrated by the example with D driven by Y given previously is prevented since the clock must change from 1 to 0 before a change in Y based on D can occur.

Flip-Flop Problem

- The change in the flip-flop output is delayed by the pulse width which makes the circuit slower or
- S and/or R are permitted to change while C = 1
  - Suppose Q = 0 and S goes to 1 and then back to 0 with R remaining at 0
    - The master latch sets to 1
    - A 1 is transferred to the slave
  - Suppose Q = 0 and S goes to 1 and back to 0 and R goes to 1 and back to 0
    - The master latch sets and then resets
    - A 0 is transferred to the slave
  - This behavior is called *1s catching*

Flip-Flop Problem

- If S and/or R are change while C = 1
  - Short pulse is "caught" by master S/R latch
    - *1s Catching*
- Not what we want or expect

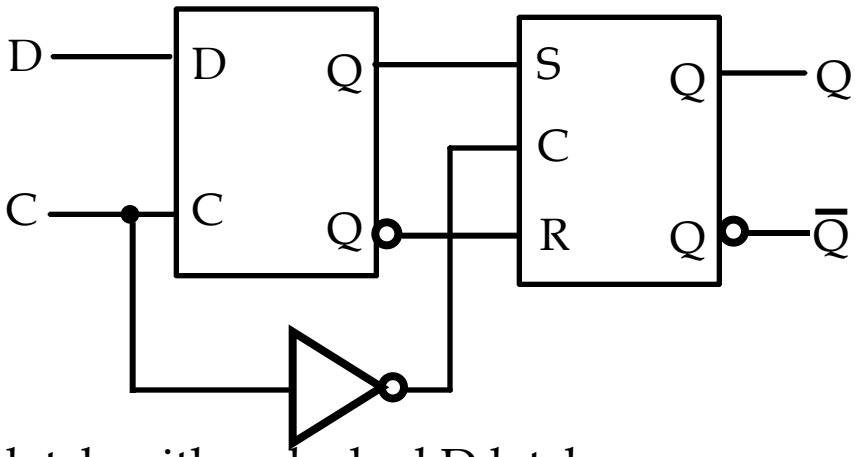


Flip-Flop Solution

- Use edge-triggering instead of master-slave
- An *edge-triggered* flip-flop ignores the pulse while it is at a constant level and triggers only during a transition of the clock signal
- Edge-triggered flip-flops can be built directly at the electronic circuit level, or
- A master-slave D flip-flop which also exhibits edge-triggered behavior can be used.

Edge-Triggered D Flip-Flop

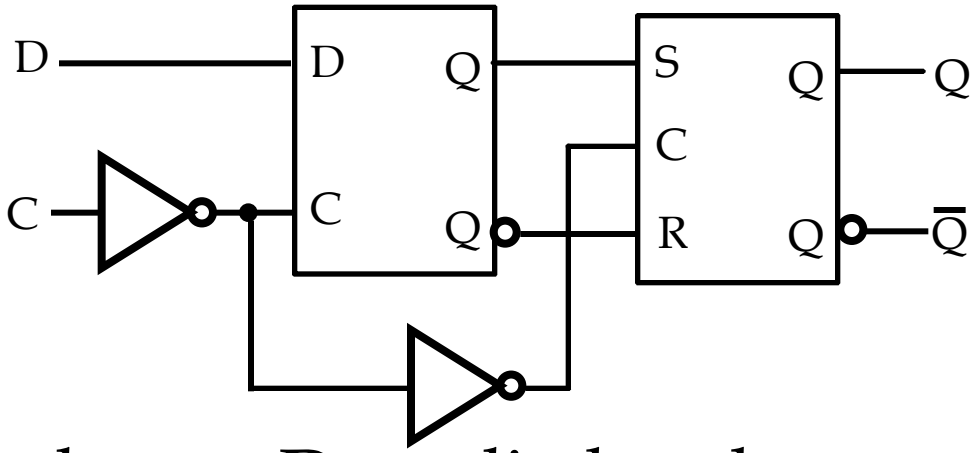
- The edge-triggered D flip-flop is the same as the master-slave D flip-flop



- It can be formed by:
  - Replacing the first clocked S-R latch with a clocked D latch or
  - Adding a D input and inverter to a master-slave S-R flip-flop
- The delay of the S-R master-slave flip-flop can be avoided since the 1s-catching behavior is not present with D replacing S and R inputs
- The change of the D flip-flop output is associated with the negative edge at the end of the pulse
- It is called a *negative-edge triggered* flip-flop

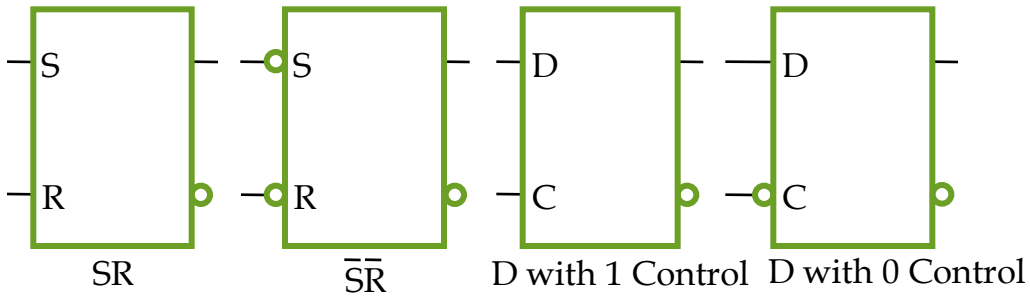
Positive-Edge Triggered D Flip-Flop

- Formed by adding inverter to clock input



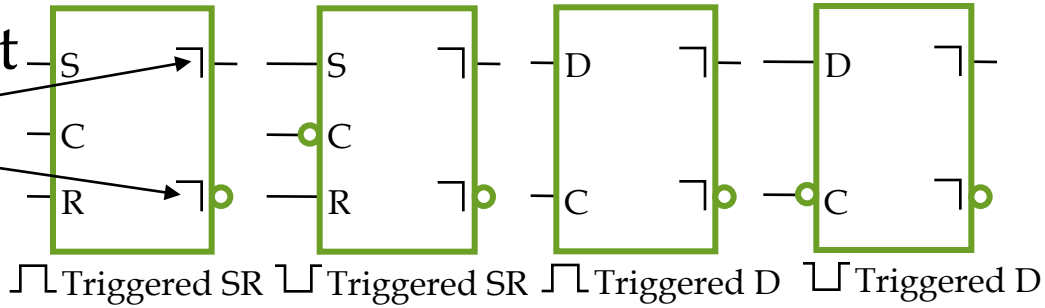
- Q changes to the value on D applied at the positive clock edge within timing constraints to be specified
- Our choice as the standard flip-flop for most sequential circuits

Standard Symbols for Storage Elements



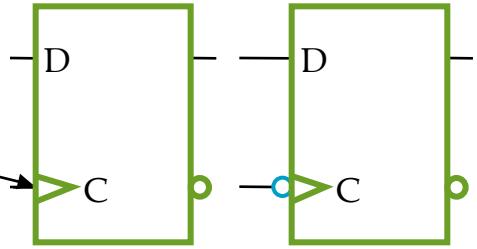
(a) Latches

- Master-Slave: Postponed output indicators



(b) Master-Slave Flip-Flops

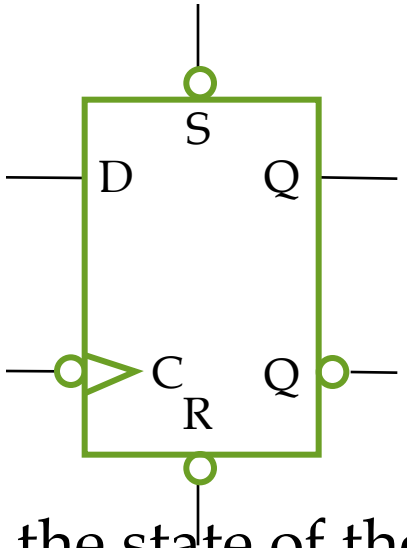
- Edge-Triggered: Dynamic indicator



(c) Edge-Triggered Flip-Flops

Direct Inputs

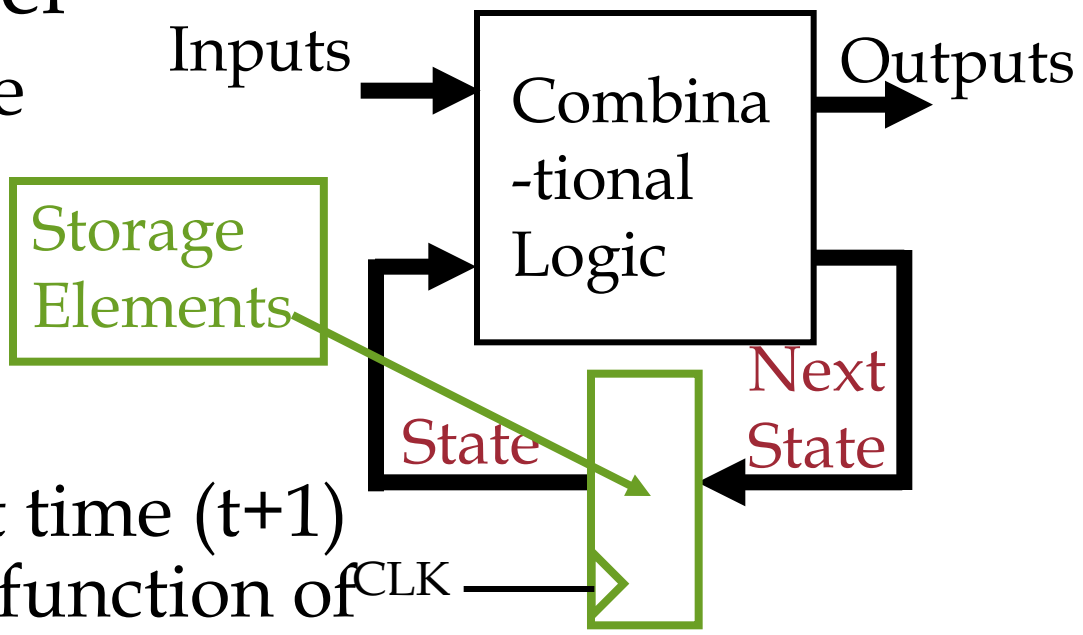
- At power up or at reset, all or part of a sequential circuit usually is initialized to a known state before it begins operation
- This initialization is often done outside of the clocked behavior of the circuit, i.e., asynchronously.



- Direct R and/or S inputs that control the state of the latches within the flip-flops are used for this initialization.
- For the example flip-flop shown
  - 0 applied to $\overline{R}$ resets the flip-flop to the 0 state
  - 0 applied to $\overline{S}$ sets the flip-flop to the 1 state

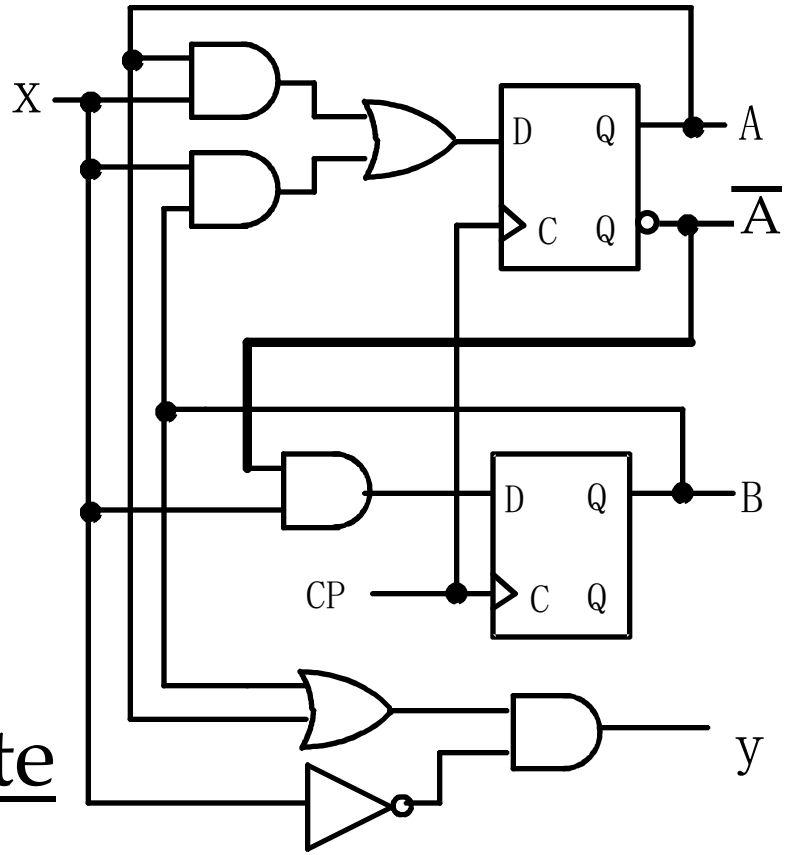
4-4 Sequential Circuit Analysis

- General Model
  - Current State at time (t) is stored in an array of flip-flops.
  - Next State at time (t+1) is a Boolean function of State and Inputs.
  - Outputs at time (t) are a Boolean function of State (t) and (sometimes) Inputs (t).



Example 1 (from Fig. 5-15)

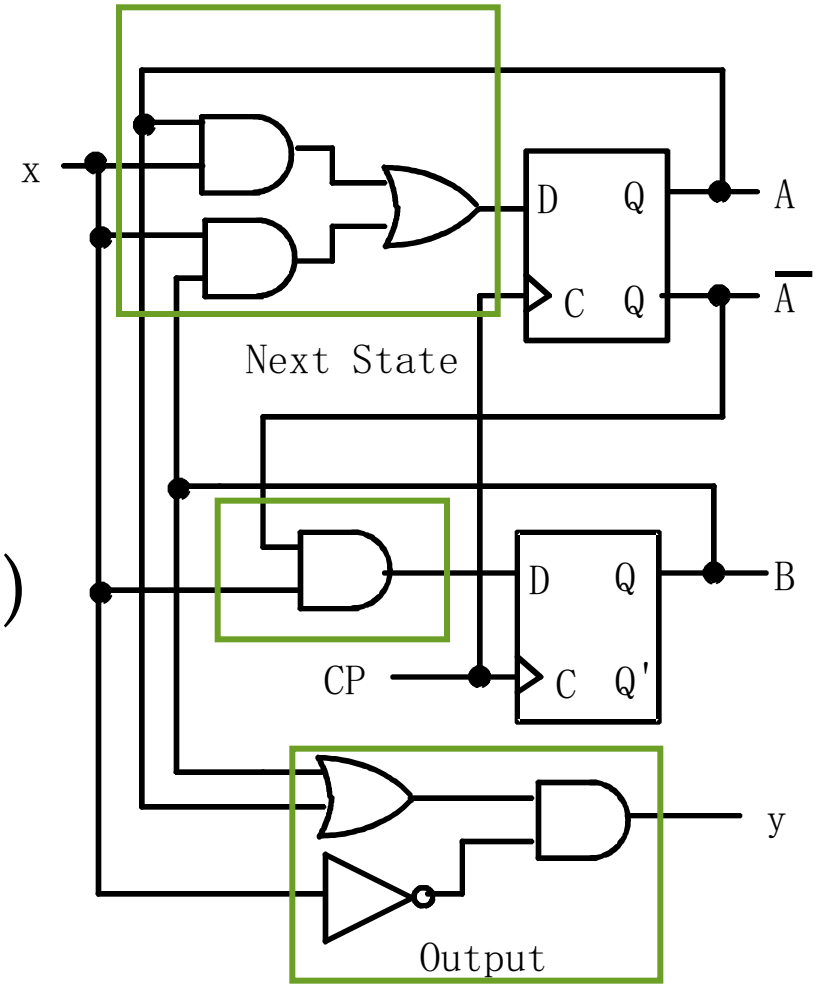
- Input: x(t)
- Output: y(t)
- State: (A(t), B(t))
- What is the Output Function?



- What is the Next State Function?

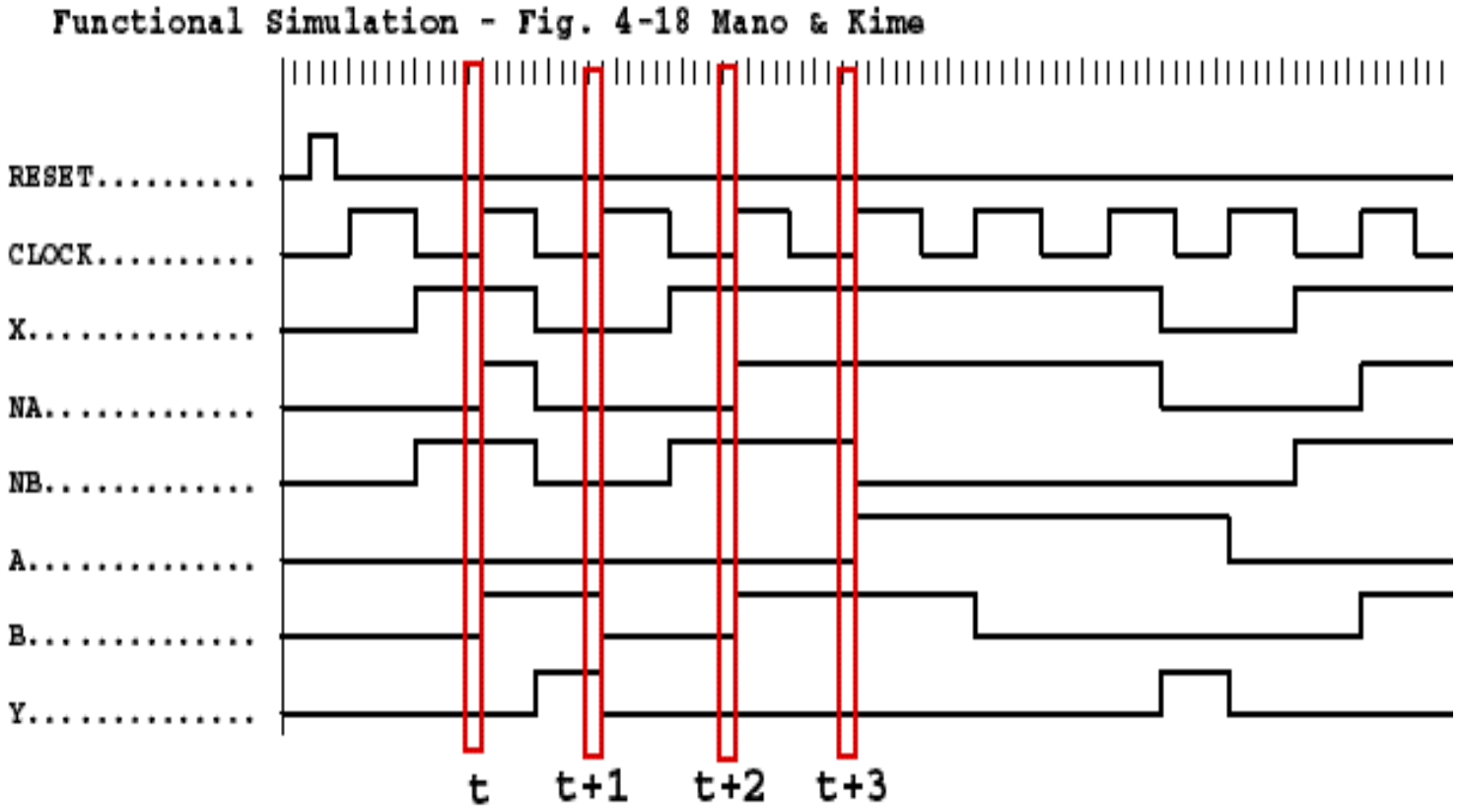
Example 1 (from Fig. 5-15) (continued)

- Boolean equations for the functions:
  - $A(t+1) = A(t)x(t) + B(t)x(t)$
  - $B(t+1) = \overline{A}(t)x(t)$
  - $y(t) = \overline{x}(t)(B(t) + A(t))$



Example 1 (from Fig. 5-15) (continued)

- Where in time are inputs, outputs and states defined?



State Table Characteristics

- *State table* – a multiple variable table with the following four sections:
  - *Present State* – the values of the state variables for each allowed state.
  - *Input* – the input combinations allowed.
  - *Next-state* – the value of the state at time (t+1) based on the present state and the input.
  - *Output* – the value of the output as a function of the present state and (sometimes) the input.
- From the viewpoint of a truth table:
  - the inputs are Input, Present State
  - and the outputs are Output, Next State

Example 1: State Table (from Fig. 5-15)

- The state table can be filled in using the next state and output equations: $A(t+1) = A(t)x(t) + B(t)x(t)$
- $B(t+1) = \bar{A}(t)x(t)$
  $y(t) = \bar{x}(t)(B(t) + A(t))$

Present State		Input	Next State		Output
A(t)	B(t)	x(t)	A(t+1)	B(t+1)	y(t)
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

Example 1: Alternate State Table

- 2-dimensional table that matches well to a K-map. Present state rows and input columns in Gray code order.
  - $A(t+1) = A(t)x(t) + B(t)x(t)$
  - $B(t+1) = \overline{A}(t)x(t)$
  - $y(t) = \overline{x}(t)(B(t) + A(t))$

Present State A(t) B(t)	Next State x(t)=0 A(t+1)B(t+1)	Next State x(t)=1 A(t+1)B(t+1)	Output x(t)=0 y(t)	Output x(t)=1 y(t)
0 0	0 0	0 1	0	0
0 1	0 0	1 1	1	0
1 0	0 0	1 0	1	0
1 1	0 0	1 0	1	0

State Diagrams

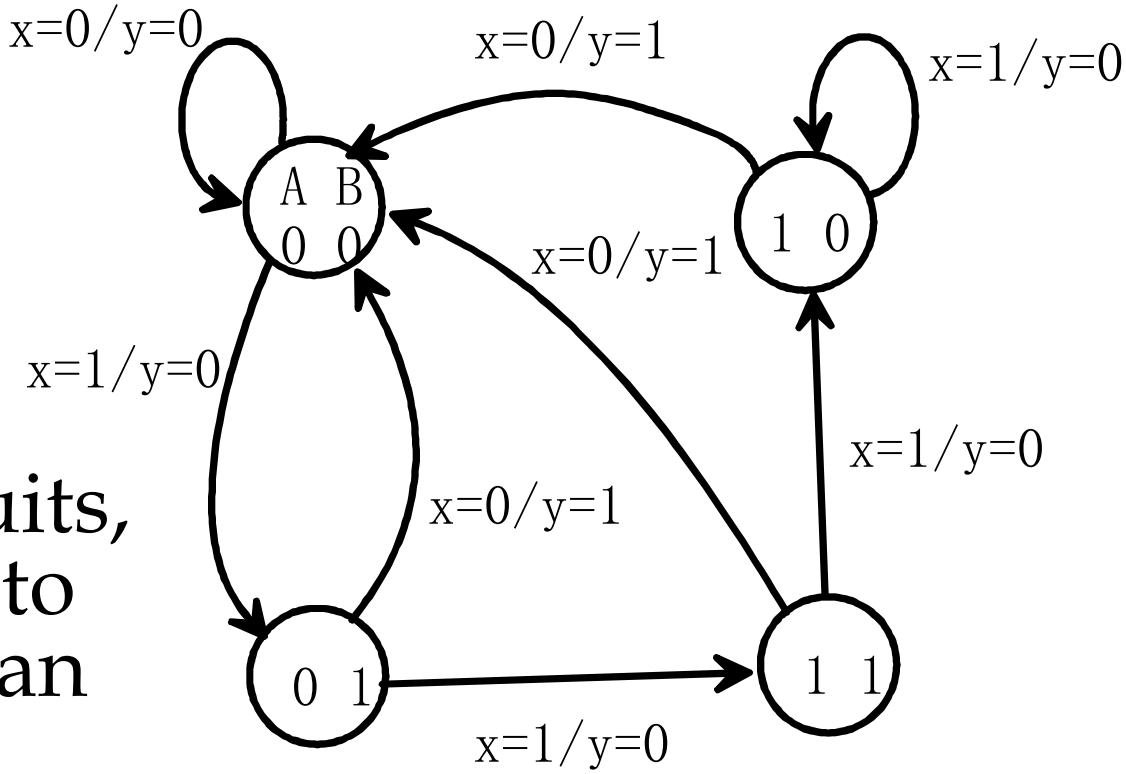
- The sequential circuit function can be represented in graphical form as a state diagram with the following components:
  - A circle with the state name in it for each state
  - A directed arc from the Present State to the Next State for each state transition
  - A label on each directed arc with the Input values which causes the state transition, and
  - A label:
    - On each circle with the output value produced, or
    - On each directed arc with the output value produced.

State Diagrams

- Label form:
  - On circle with output included:
    - state/output
    - Moore type output depends only on state
  - On directed arc with the output included:
    - input/output
    - Mealy type output depends on state and input

Example 1: State Diagram

- Which type?
- Diagram gets confusing for large circuits
- For small circuits, usually easier to understand than the state table

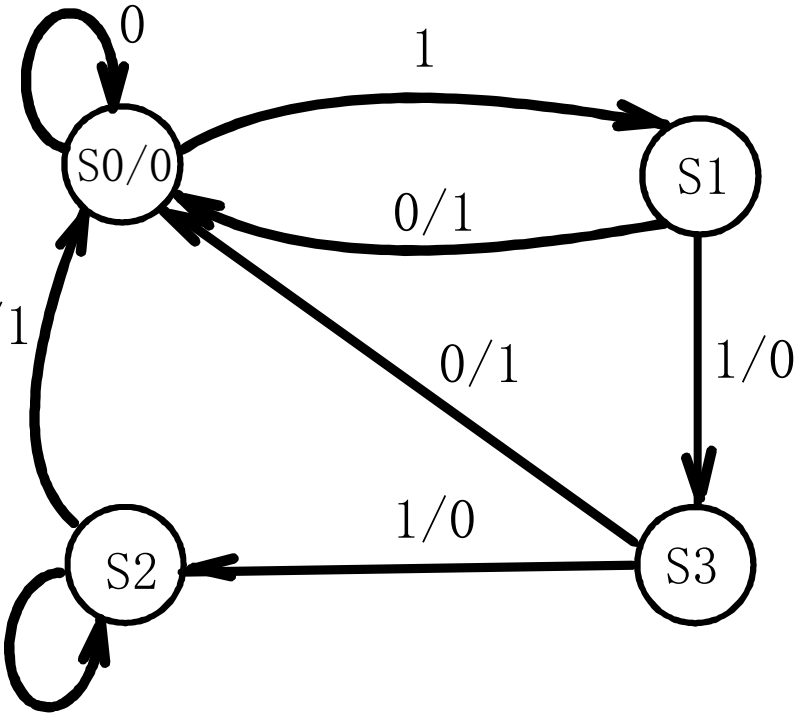


Equivalent State Definitions

- 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.

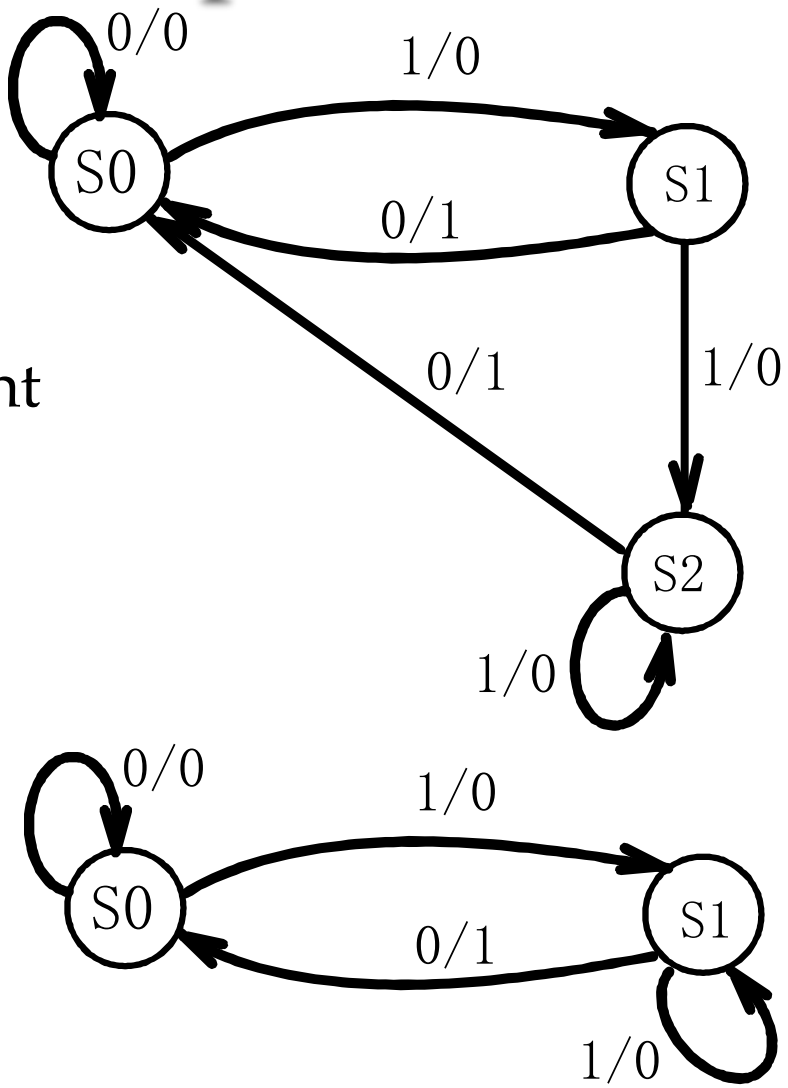
Equivalent State Example

- Text Figure 5-17(a):
- 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, and
  - 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:

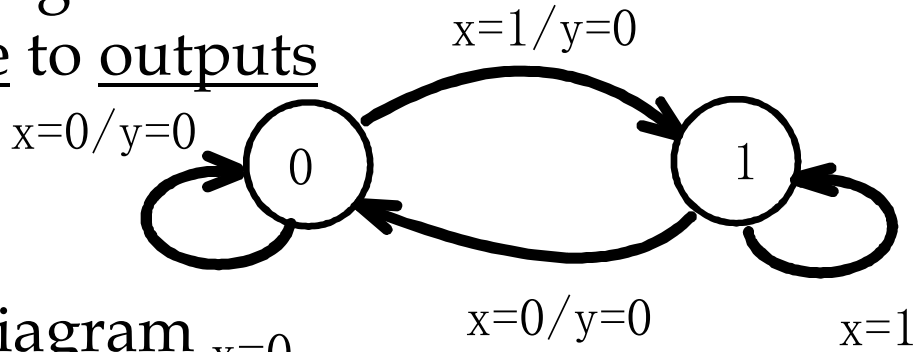


Moore and Mealy Models

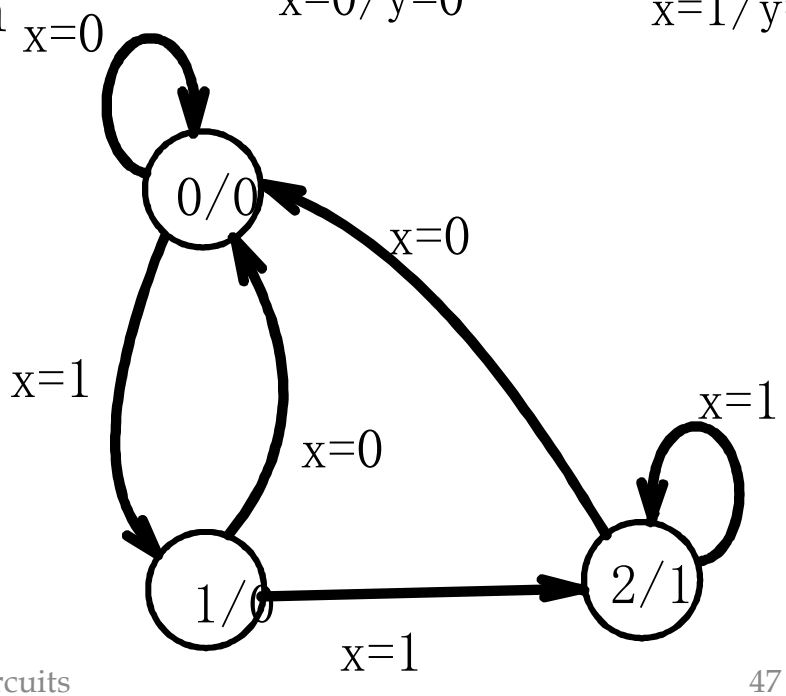
- Sequential Circuits or Sequential Machines are also called *Finite State Machines* (FSMs). Two formal models exist:
  - Moore Model
    - Named after E.F. Moore
    - Outputs are a function ONLY of states
    - Usually specified on the states.
  - Mealy Model
    - Named after G. Mealy
    - Outputs are a function of inputs AND states
    - Usually specified on the state transition arcs.

Moore and Mealy Example Diagrams

- Mealy Model State Diagram maps inputs and state to outputs



- Moore Model State Diagram maps states to outputs



Moore and Mealy Example Tables

- Moore Model state table maps state to outputs

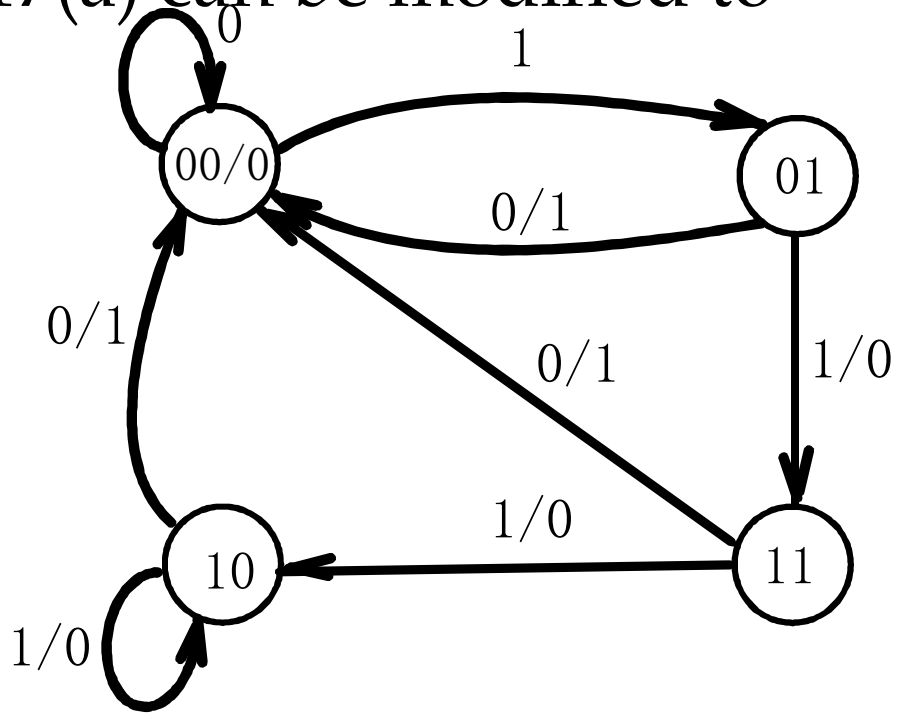
Present State	Next State x=0	Next State x=1	Output
0	0	1	0
1	0	2	0
2	0	2	1

- Mealy Model state table 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

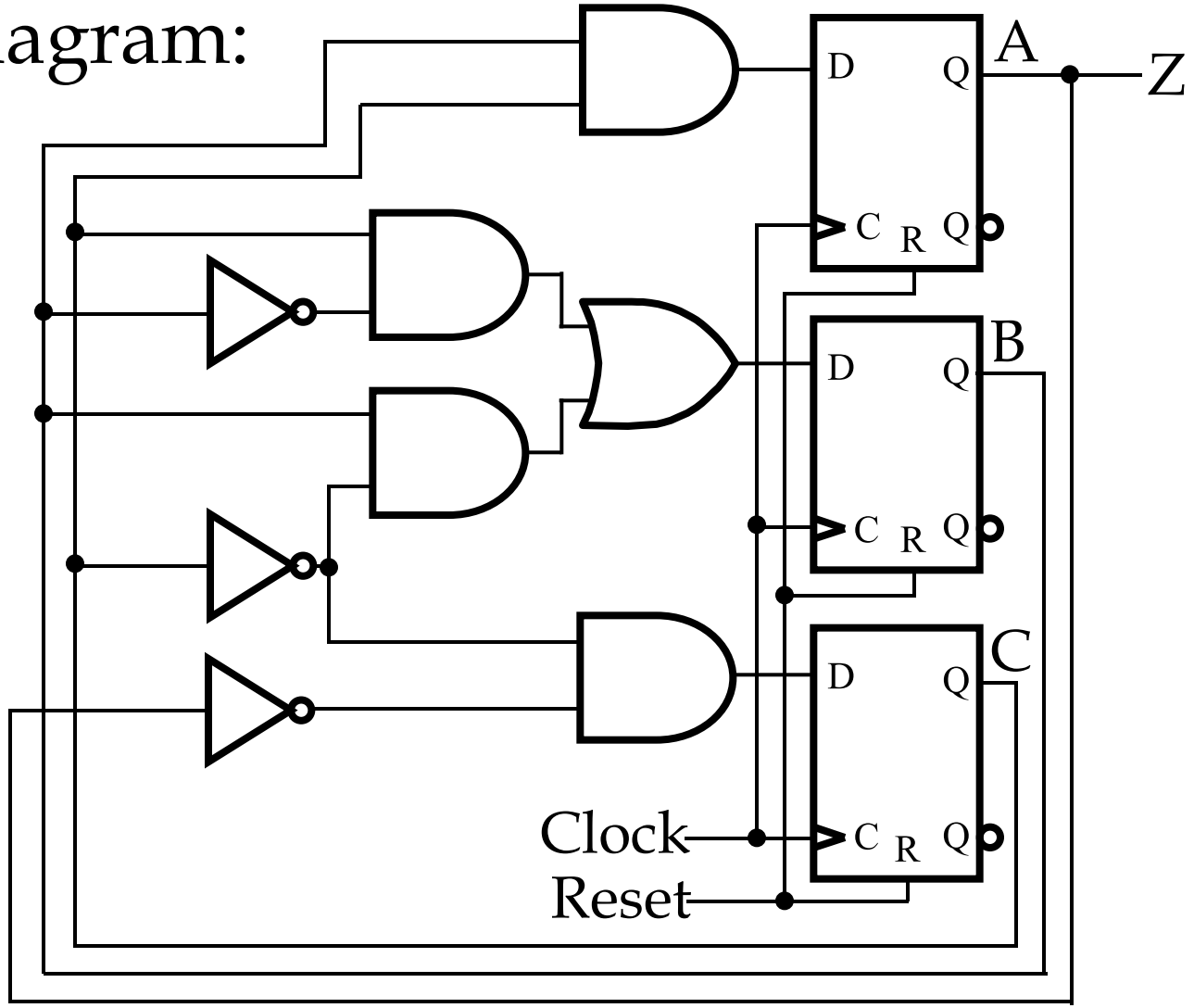
Mixed Moore and Mealy Outputs

- In real designs, some outputs may be Moore type and other outputs may be Mealy type.
- Example: Figure 5-17(a) can be modified to illustrate this
  - State 00: Moore
  - States 01, 10, and 11: Mealy
- Simplifies output specification



Example 2: Sequential Circuit Analysis

- Logic Diagram:



Example 2: Flip-Flop Input Equations

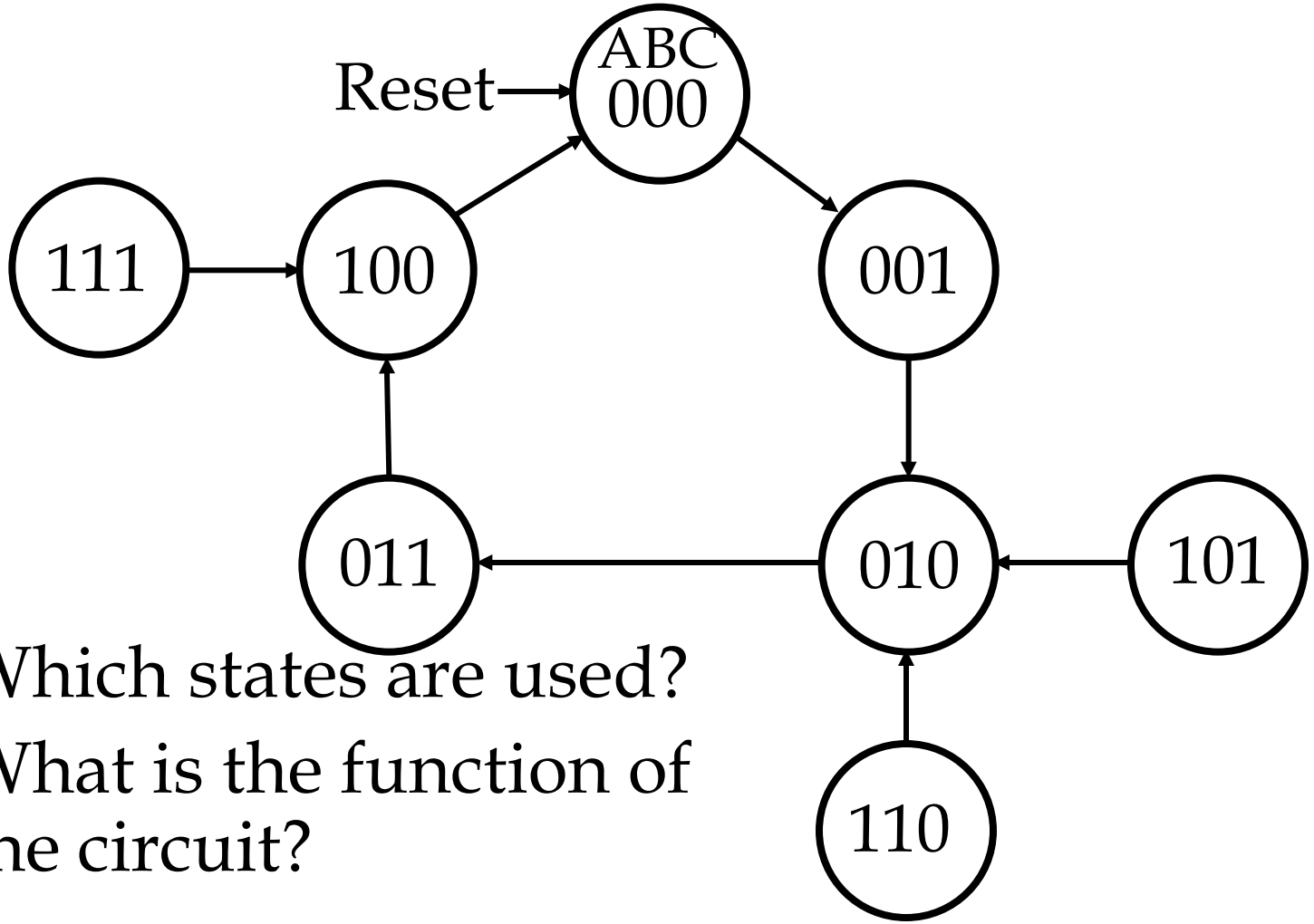
- Variables
  - Inputs: None
  - Outputs: Z
  - State Variables: A, B, C
- Initialization: Reset to (0,0,0)
- Equations
  - A(t+1) = Z =
  - B(t+1) =
  - C(t+1) =

Example 2: State Table

$X' = X(t+1)$

A B C	A'B'C'	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		

Example 2: State Diagram



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

4-5 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

Specification

- Component Forms of Specification
  - Written description
  - Mathematical description
  - Hardware description language*
  - Tabular description*
  - Equation description*
  - Diagram describing operation (not just structure)*
- Relation to Formulation
  - If a specification is rigorous at the binary level (marked with * above), then all or part of formulation may be completed

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

- 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

- 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 the final input value) occurence.
  - 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."*.

State Assignment

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

  $$n \geq \lceil \log_2 m \rceil$$

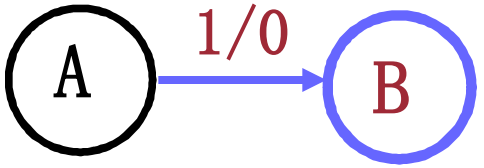
  where $\lceil n \rceil$ is the smallest integer $\geq n$
- There are useful state assignments that use more than the minimum number of bits
- There are 2^n - m unused states

Sequence Recognizer Example

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

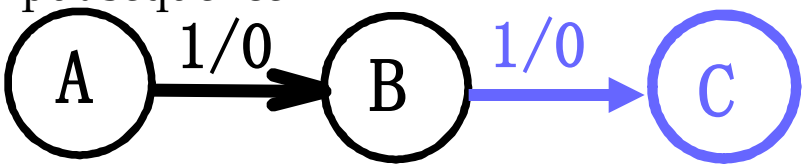
- Define states for the sequence to be recognized:
  - assuming it starts with first symbol,
  - continues through each symbol in the sequence to be recognized, and
  - uses output 1 to mean the full sequence has occurred,
  - with output 0 otherwise.
- Starting in the initial state (Arbitrarily named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



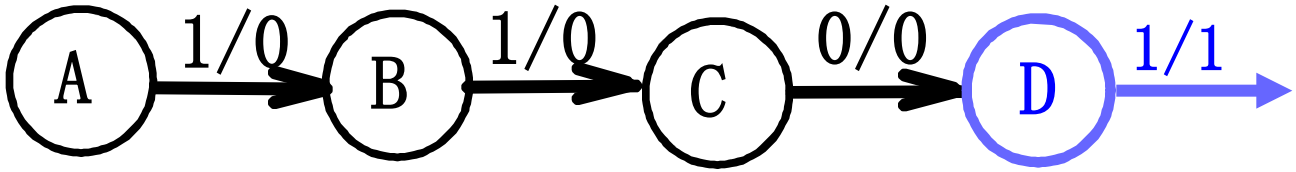
  - State "A" is the initial state, and state "B" is the state which represents the fact that the "first" one in the input subsequence has occurred. The output symbol "0" means that the full recognized sequence has not yet occurred.

Example: Recognize 1101 (continued)

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

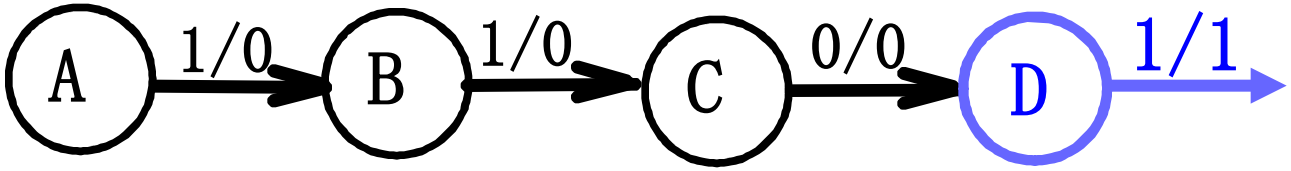


- Finally, after 110 and a 1, we have:

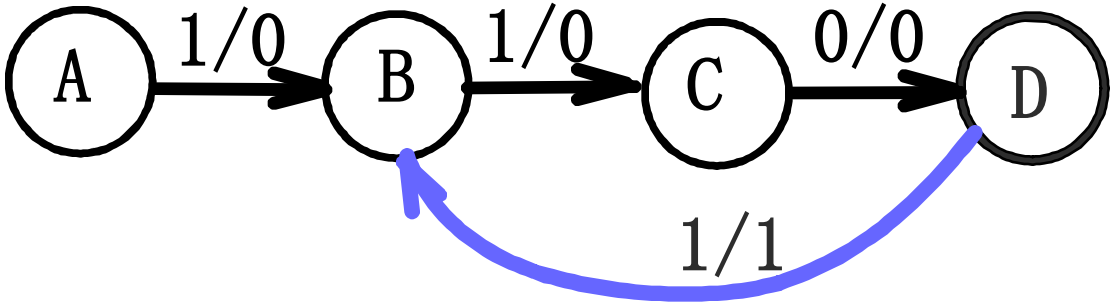


  - Transition arcs are used to denote the output function (Mealy Model)
  - Output 1 on the arc from D means the sequence has been recognized
  - To what state should the arc from state D go? Remember: 1101101 ?
  - Note that D is the last state but the output 1 occurs for the input applied in D. This is the case when a *Mealy model* is assumed.

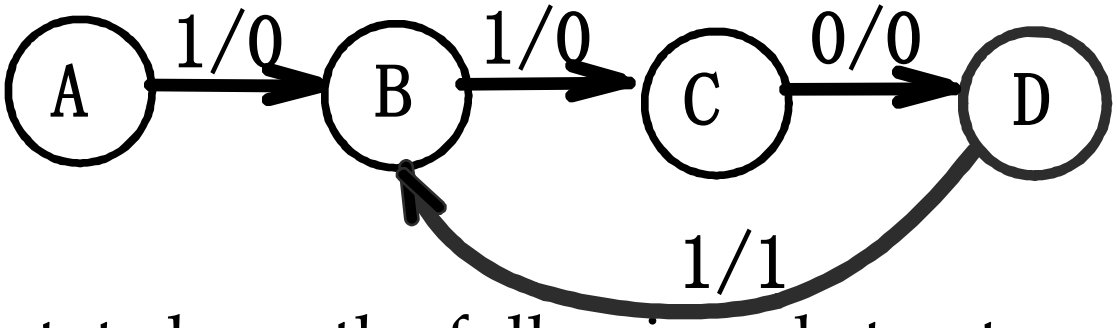
Example: Recognize 1101 (continued)



- 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:



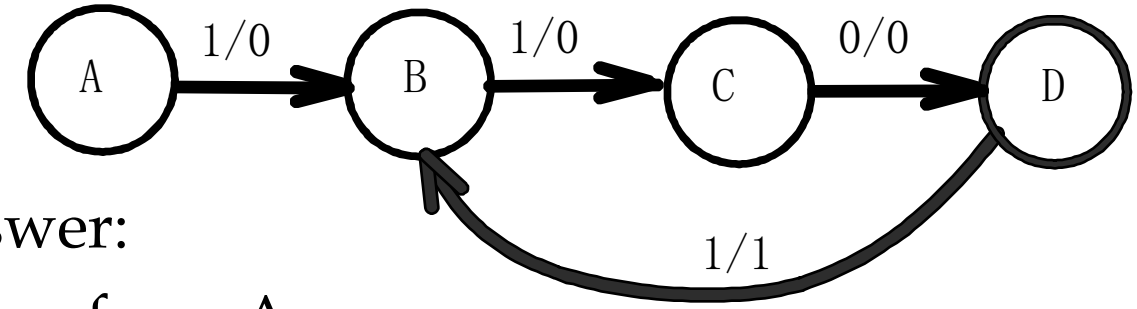
Example: Recognize 1101 (continued)



- The 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 (continued)

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



- Answer:

"0" arc from A

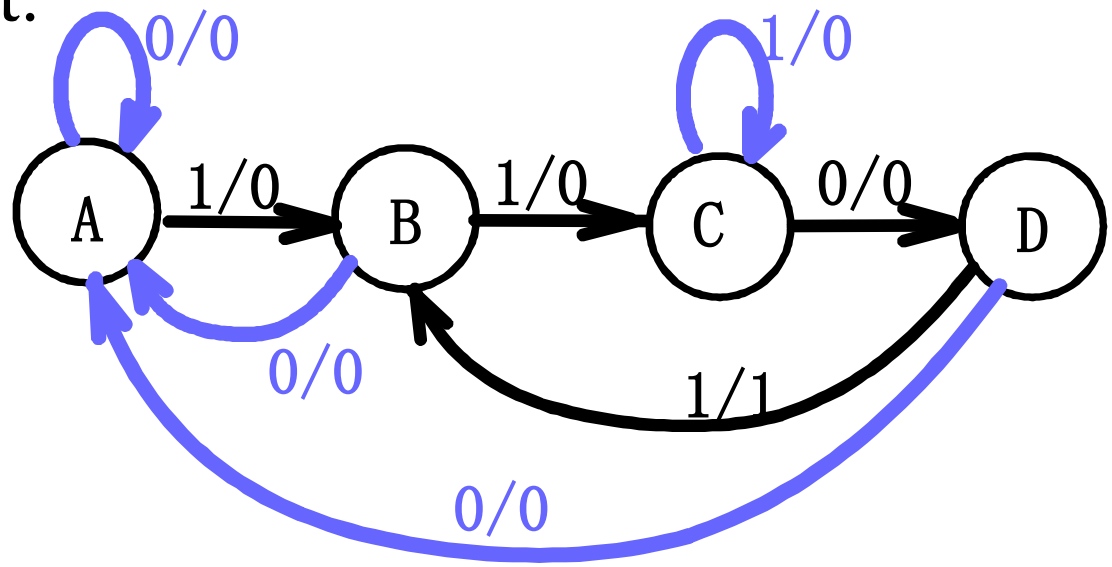
"0" arc from B

"1" arc from C

"0" arc from D.

Example: Recognize 1101 (continued)

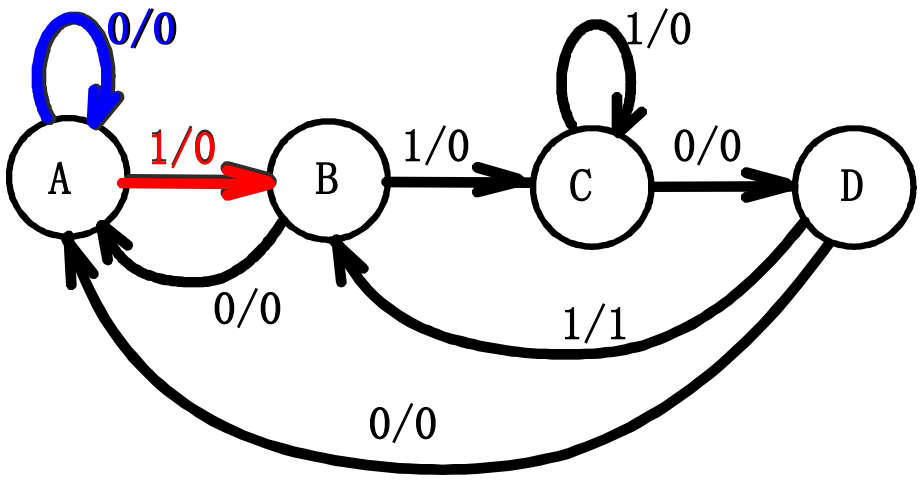
- State transition arcs must represent the fact that an input subsequence has occurred. Thus we get:



- 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

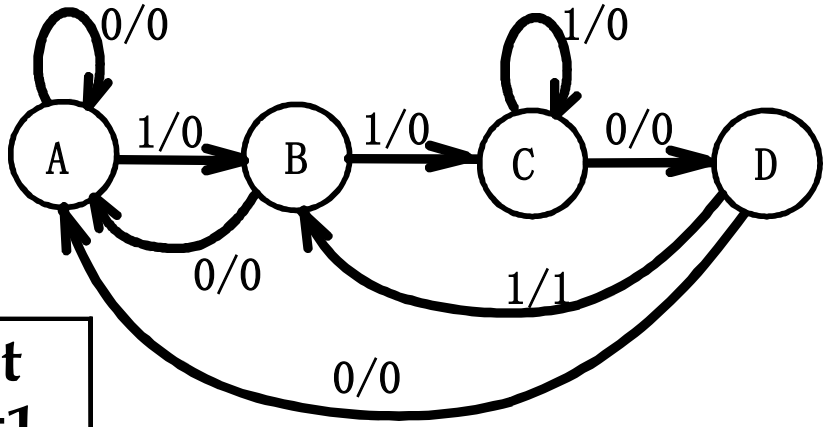
- From the State Diagram, we can fill in the State Table.
- There are 4 states, one input, and one output. We will choose the form with four rows, one for each current state.
- From State A, the 0 and 1 input transitions have been filled in along with the outputs.



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

Formulation: Find State Table

- From the state diagram, we complete 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

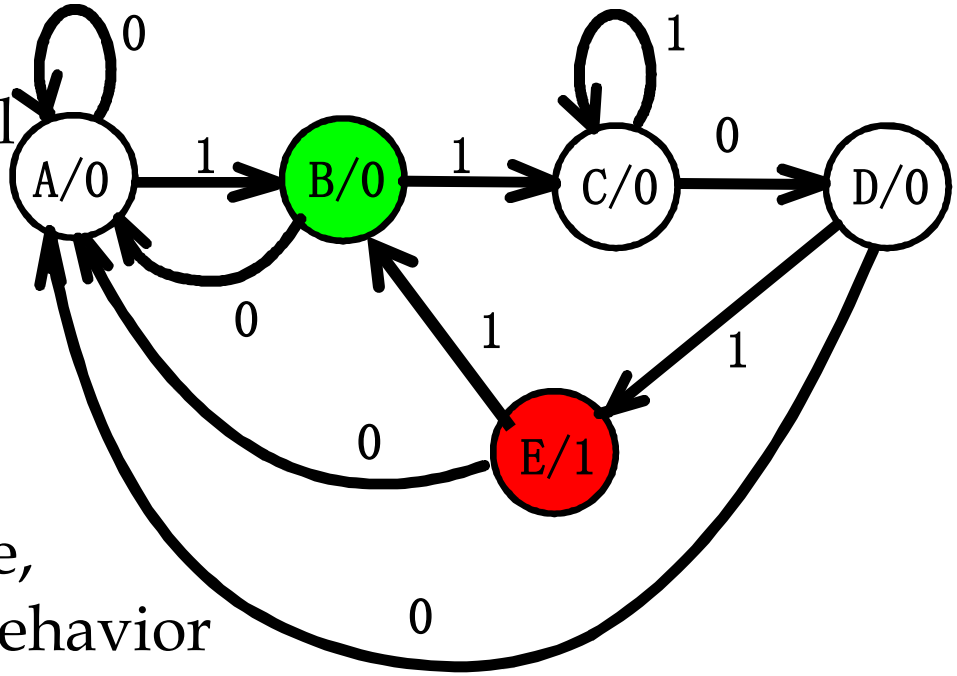
- What would the state diagram and state table look like for the Moore model?

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 D, would generate an output of 1 and thus be different from D.
- The Moore model for a sequence recognizer usually has *more states* than the Mealy model.

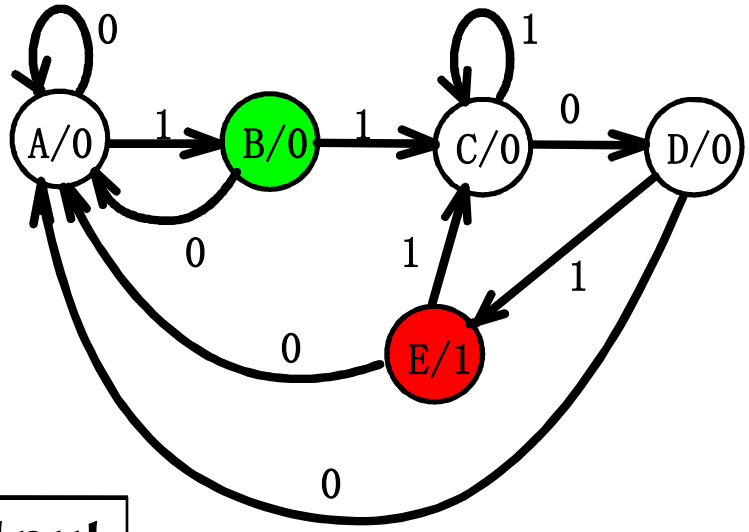
Example: Moore Model (continued)

- 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 (continued)

- The state table is shown below
- 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 – 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 may 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 may assignments of codes with a minimum number of bits?
  - $4 \times 3 \times 2 \times 1 = 24$
- Does code assignment make a difference in cost?

State Assignment – Example 2 (continued)

- Counting Order Assignment: A = 0 0, B = 0 1, C = 1 0, D = 1 1
- The resulting coded state table:

Present State	Next State		Output	
	x = 0	x = 1	x = 0	x = 1
Y1 Y2	D1 D2	D1D2	Z	Z
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

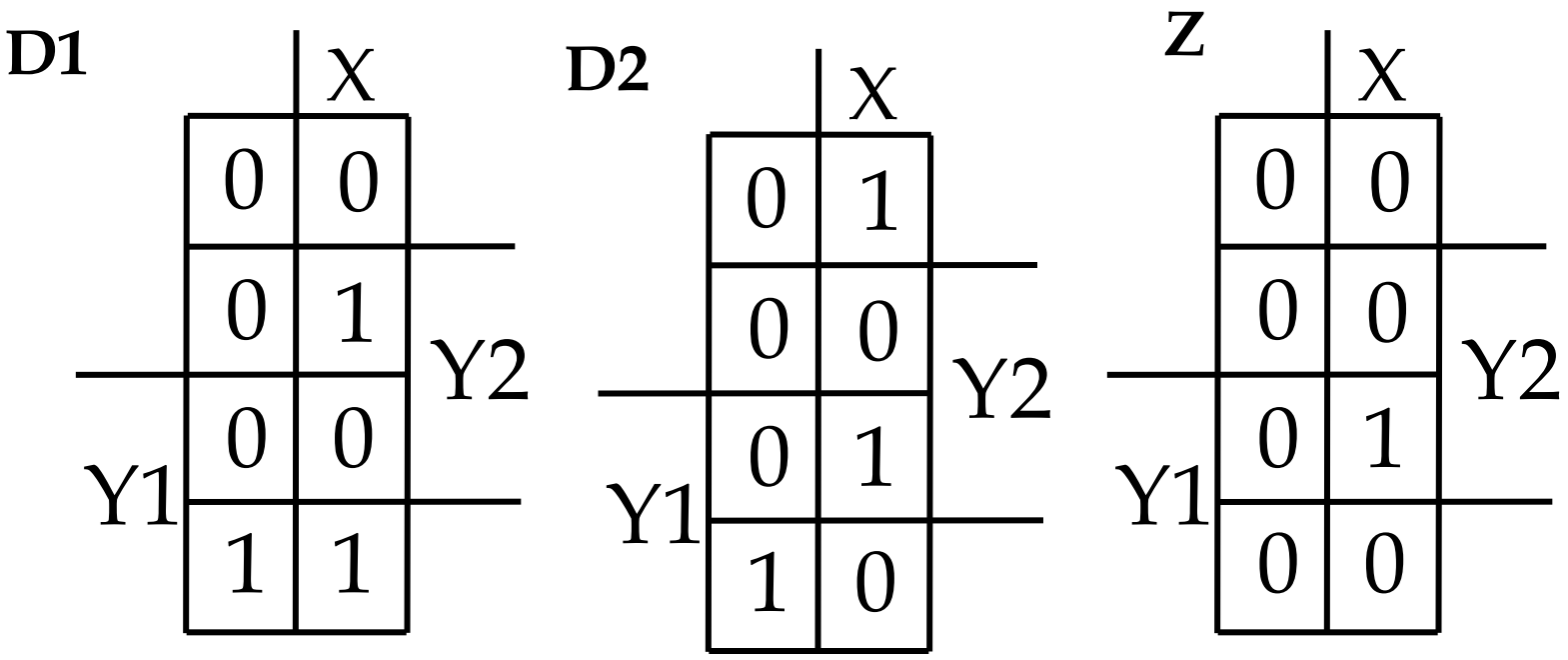
State Assignment – Example 2 (continued)

- Gray Code Assignment: A = 0 0, B = 0 1, C = 1 1, D = 1 0
- The resulting coded state table:

Present State	Next State		Output	
	x = 0	x = 1	x = 0	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: Example 2 – Counting Order Assignment

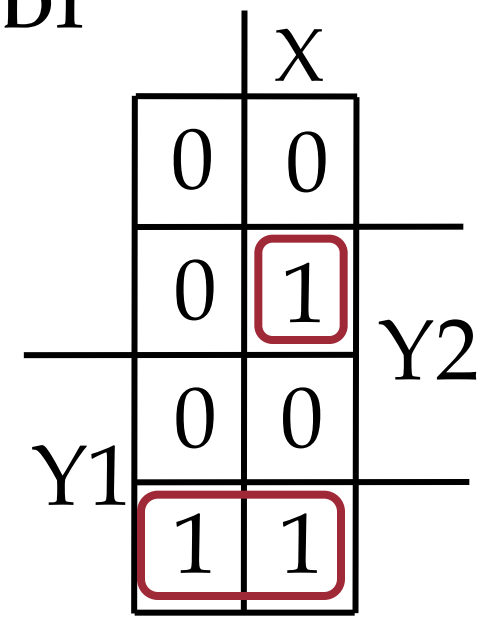
- **Assume D flip-flops**
- **Interchange the bottom two rows of the state table, to obtain K-maps for D1, D2, and Z:**



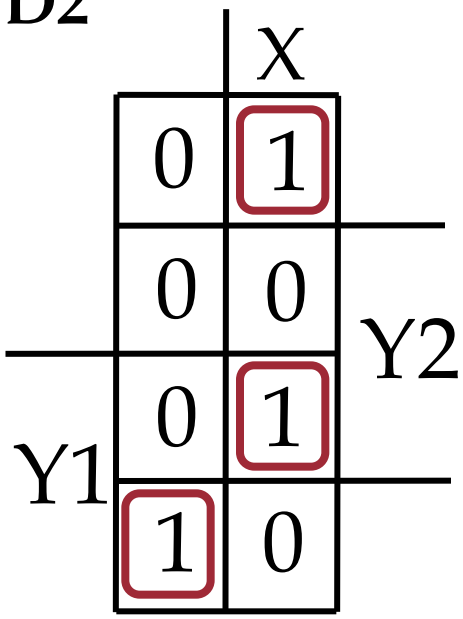
Optimization: Example 2: Counting Order Assignment

- Performing two-level optimization:

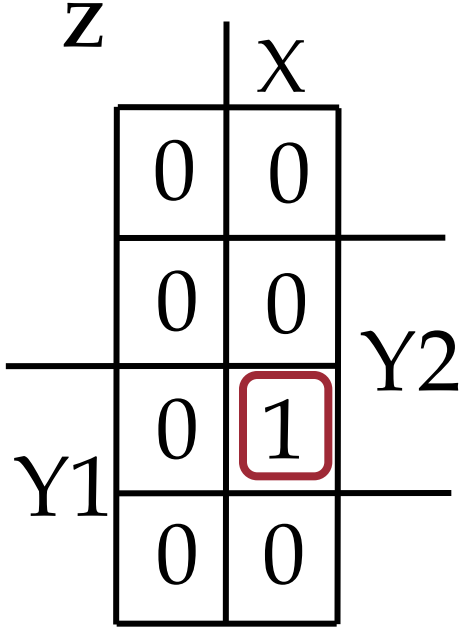
D1



D2



Z



$D_1 = Y_1Y_2' + XY_1'Y2$ (Gate Cost = 7)

$D_2 = X'Y_1Y_2' + XY_1'Y_2' + XY_1Y_2$ (Gate Cost = 12)

$Z = XY_1Y_2$ (Gate cost = 3) Total Gate Input Cost = 22

Find Flip-Flop Input and Output Equations: Example 2 – Gray Code Assignment

- Assume D flip-flops
- Obtain K-maps for D_1, D_2, and Z:

D1

Y1 \ X	0	1
0	0	0
1	1	1
0	0	1

Y2

D2

Y1 \ X	0	1
0	0	1
1	0	1
0	0	1
0	0	1

Y2

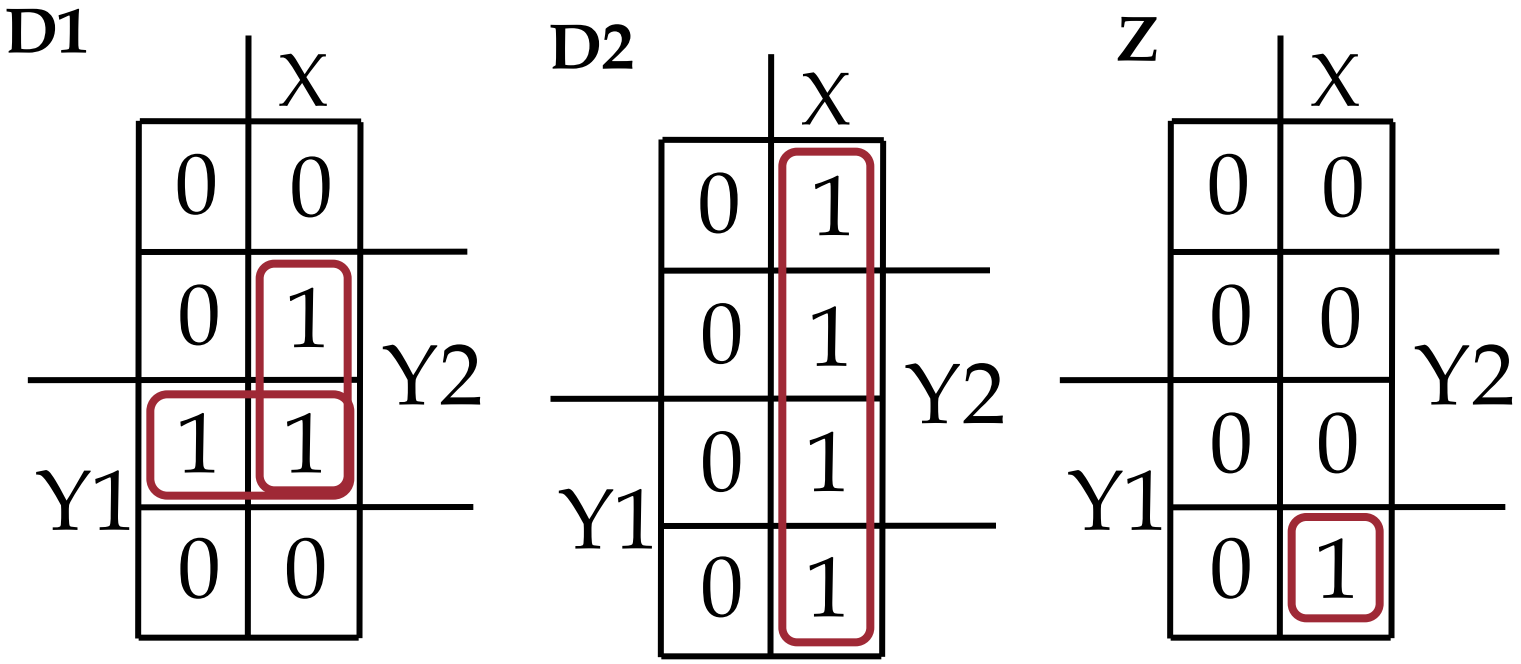
Z

Y1 \ X	0	1
0	0	0
1	0	0
0	0	0
0	0	1

Y2

Optimization: Example 2: Assignment 2

- Performing two-level optimization:



$D_1 = Y_1Y_2 + XY_2$
$D_2 = X$
$Z = XY_1Y_2'$

Gate Input Cost = 9
Select this state assignment to complete design in slide

One Flip-flop per State (One-Hot) Assignment

- Example codes for four states: (Y_3, Y_2, Y_1, Y_0) = 0001, 0010, 0100, and 1000.
- 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 all codes with 0 or two or more 1s have don't care next state values.
- Provides simplified analysis and design
- Combinational logic may be simpler, but flip-flop cost higher – may or may not be lower cost

State Assignment – Example 2 (continued)

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

Present State	Next State		Output	
	x = 0	x = 1	x = 0	x = 1
0001	0001	0010	0	0
0010	0001	0100	0	0
0100	1000	0100	0	0
1000	0001	0010	0	1

Optimization
Example 2: One Hot Assignment

- Equations read from 1 next state variable entries in table:

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

  $D_1 = X(Y_0 + Y_3)$

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

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

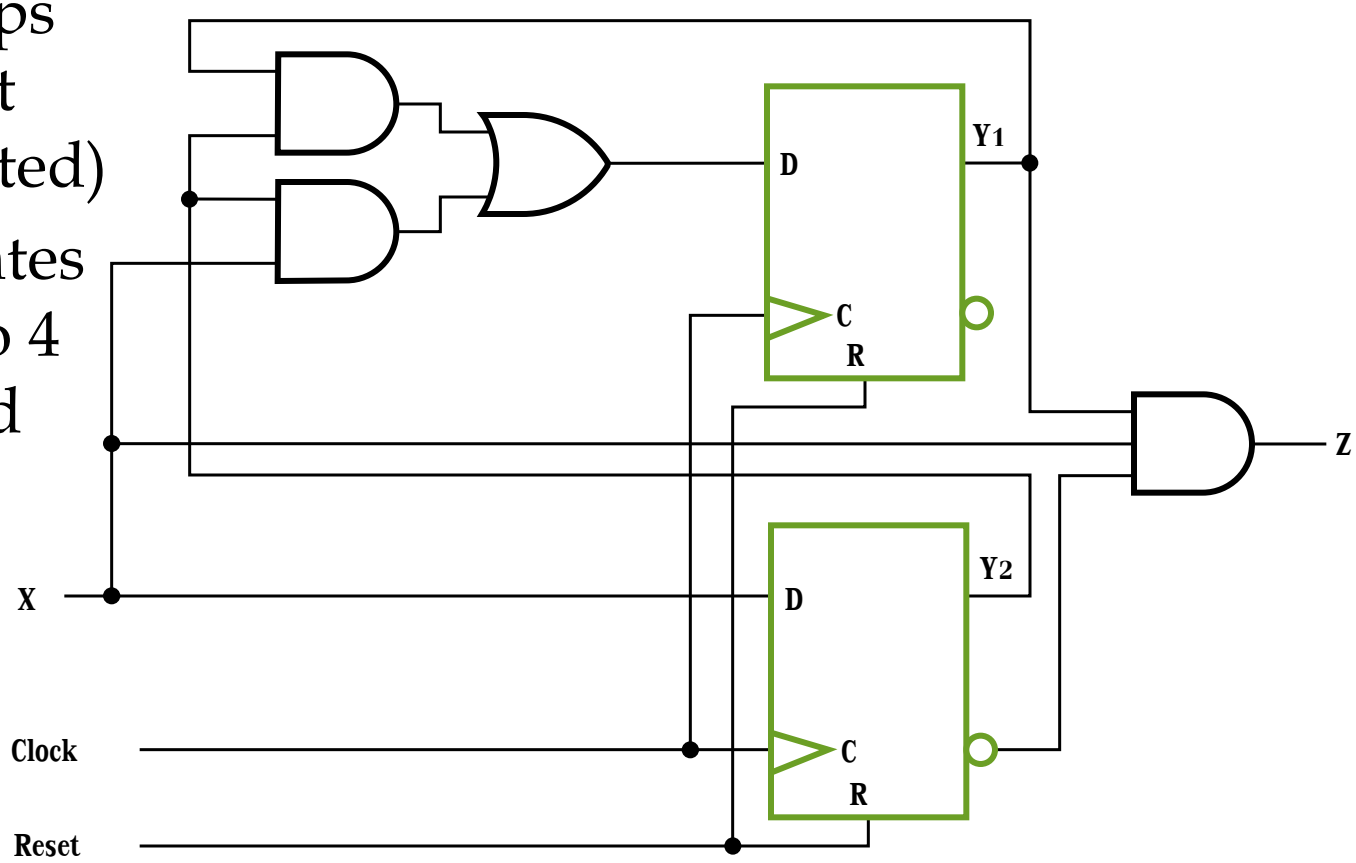
  $Z = XY_3$ Gate Input Cost = 15

- Combinational cost intermediate plus cost of two more flip-flops needed.

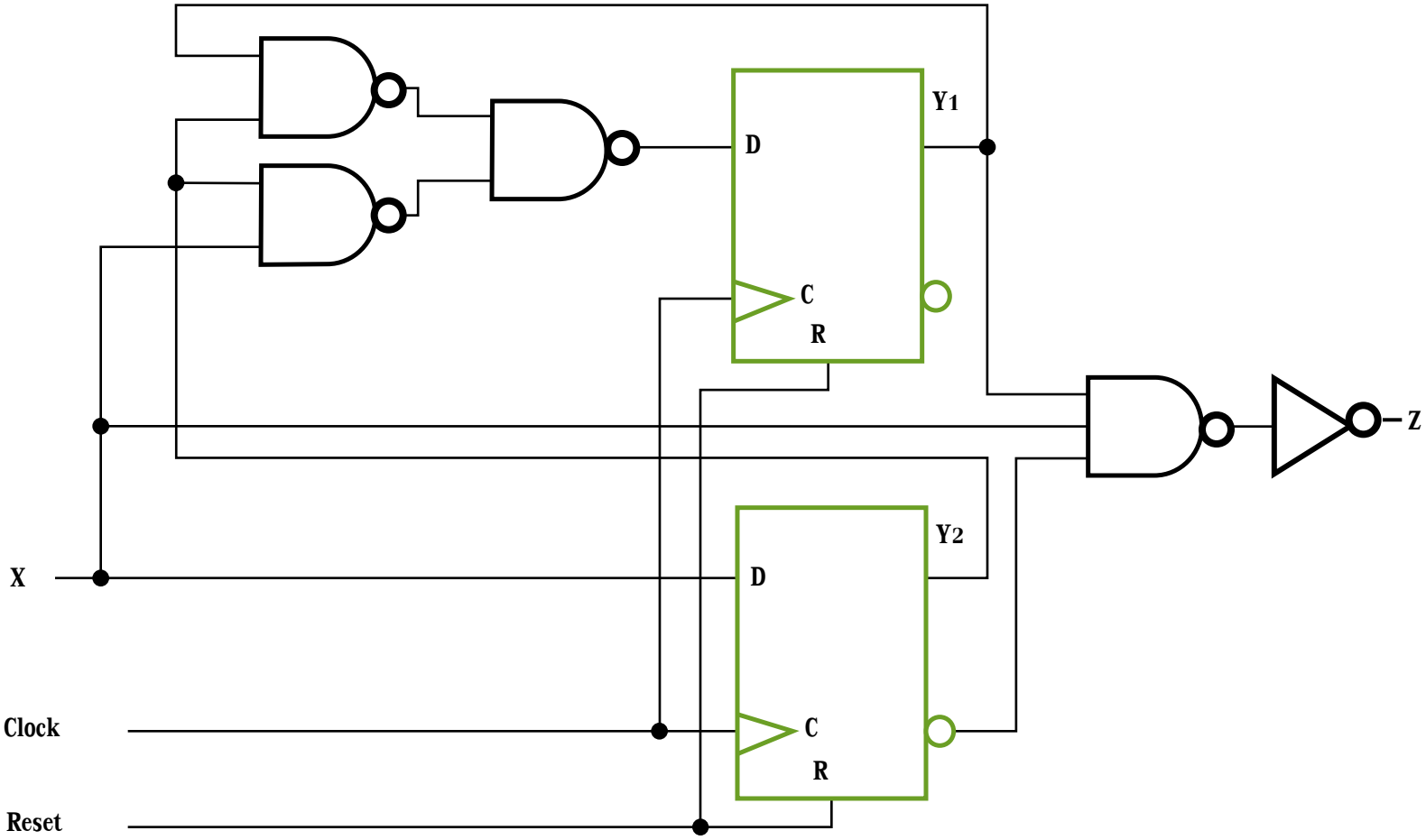
Map Technology

- Library:
  - D Flip-flops with Reset (not inverted)
  - NAND gates with up to 4 inputs and inverters

- **Initial Circuit:**

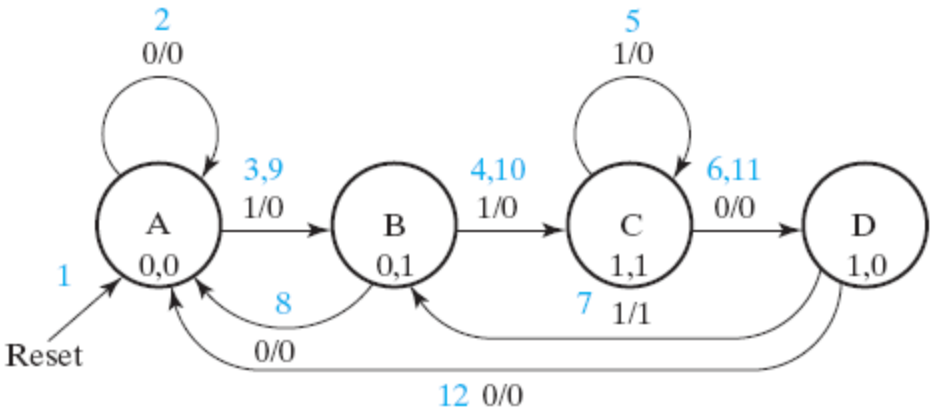


Mapped Circuit - Final Result



Verifying the Sequence Recognizer (1/2)

5-26



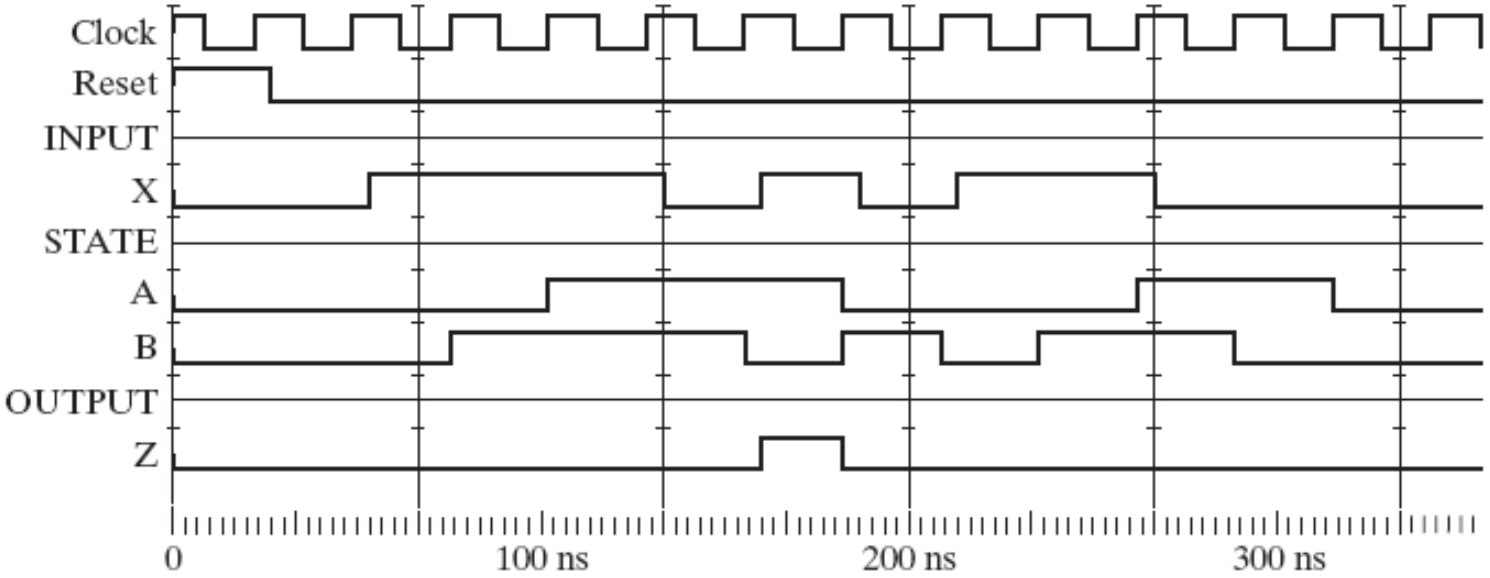
(a)

Clock Edge:	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Input R:	X	1	0	0	0	0	0	0	0	0	0	0	0	
Input X:	X	0	0	1	1	1	0	1	0	1	1	0	0	
State (A,B):	X,X	0,0*	0,0	0,0	0,1	1,1	1,1	1,0	0,1	0,0	0,1	1,1	1,0	0,0
Output Z:	X	0	0	0	0	0	0	1	0	0	0	0	0	

(b)

Verifying the Sequence Recognizer (1/2)

5-27

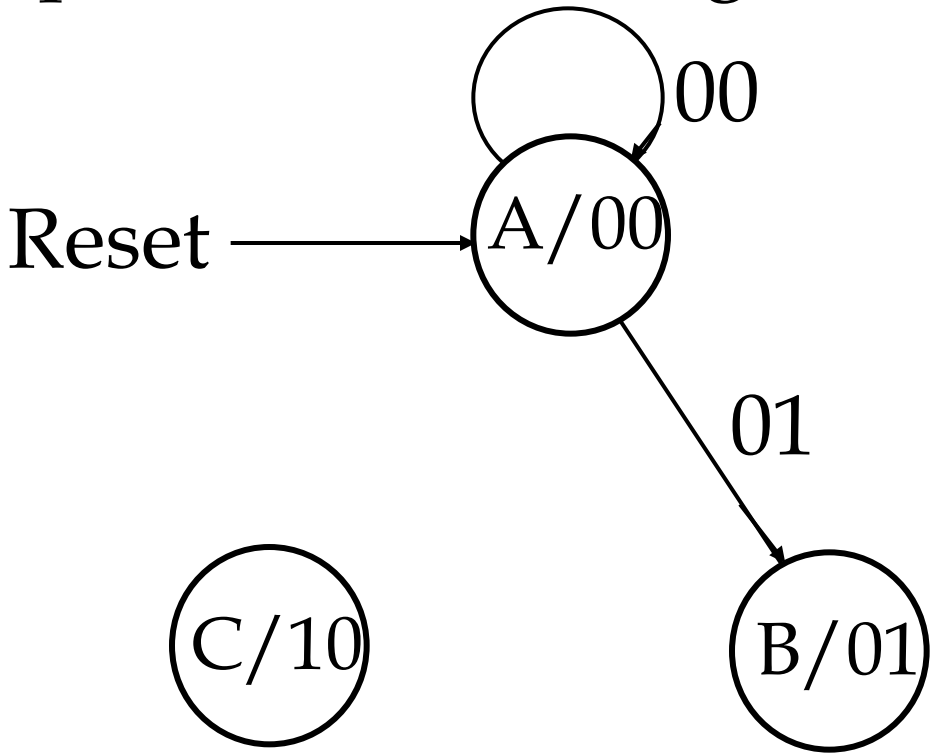


Sequential Design: Example 3

- Design a sequential modulo 3 accumulator for 2-bit operands
- Definitions:
  - Modulo *n* adder - an adder that gives the result of the addition as the remainder of the sum divided by *n*
    - Example: 2 + 2 modulo 3 = remainder of 4/3 = 1
  - Accumulator - a circuit that "accumulates" the sum of its input operands over time - it adds each input operand to the stored sum, which is initially 0.
- Stored sum: (Y_1,Y_0), Input: (X_1,X_0), Output: (Z_1,Z_0)

Example 3 (continued)

- Complete the state diagram:



Example 3 (continued)

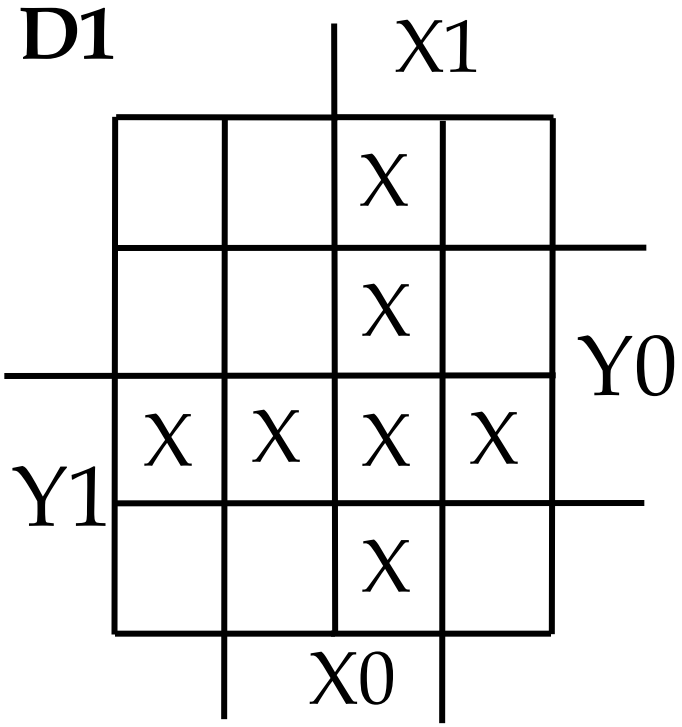
- Complete the state table

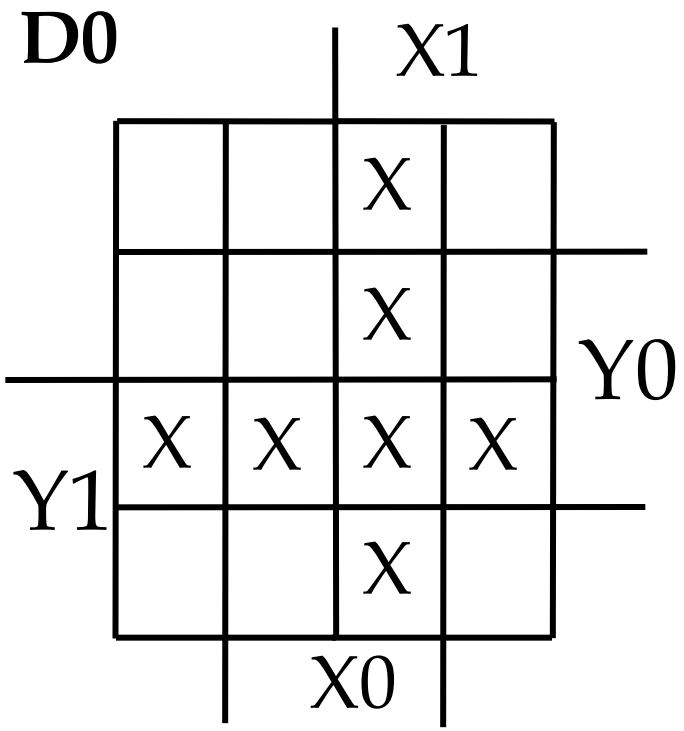
X1X0 / Y1Y0	00	01	11	10	Z_1Z_0
	$Y_1(t+1)$, $Y_0(t+1)$	$Y_1(t+1)$, $Y_0(t+1)$	$Y_1(t+1)$, $Y_0(t+1)$	$Y_1(t+1)$, $Y_0(t+1)$	
A (00)	**00**		**X**		**00**
B (01)			**X**		**01**
- (11)	**X**	**X**	**X**	**X**	**11**
C (10)			**X**		**10**

- State Assignment: $(Y_1,Y_0) = (Z_1,Z_0)$
- Codes are in gray code order to ease use of K-maps in the next step

Example 3 (continued)

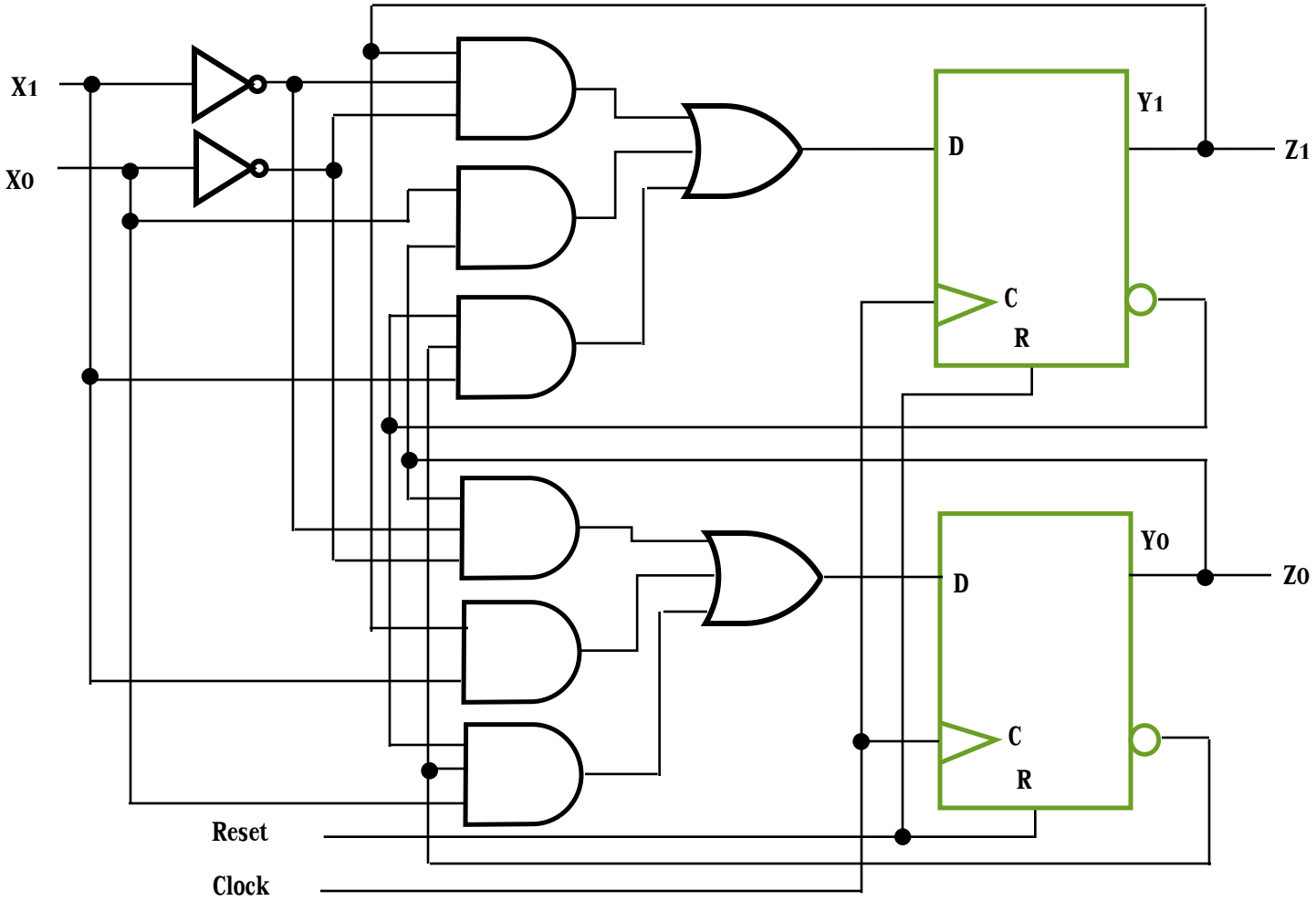
- Find optimized flip-flop input equations for D flip-flops





- D_1 =
- D_0 =

Circuit - Final Result with AND, OR, NOT



Other Flip-Flop Types

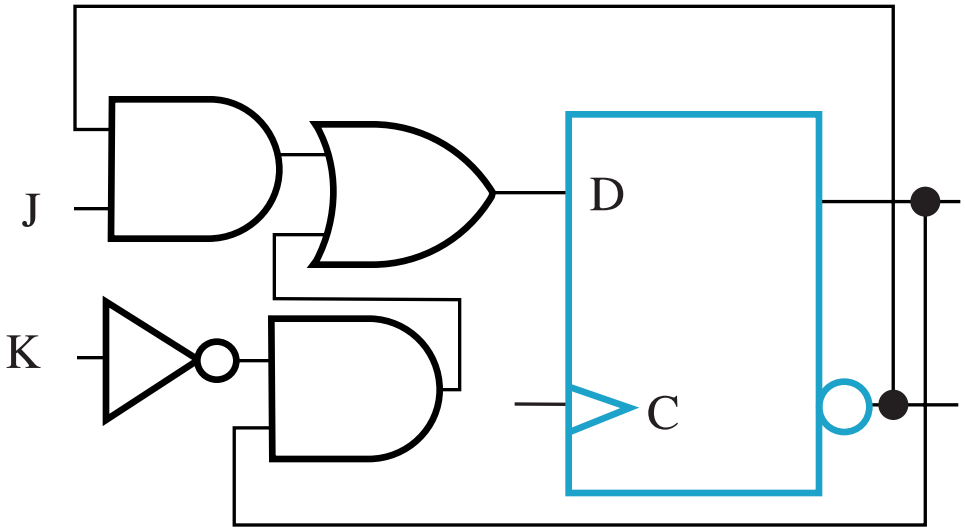
- J-K and T flip-flops
  - Behavior
  - Implementation
- Basic descriptors for understanding and using different flip-flop types
  - Characteristic tables
  - Characteristic equations
  - Excitation tables
- For actual use, see Reading Supplement - Design and Analysis Using J-K and T Flip-Flops

J-K Flip-flop

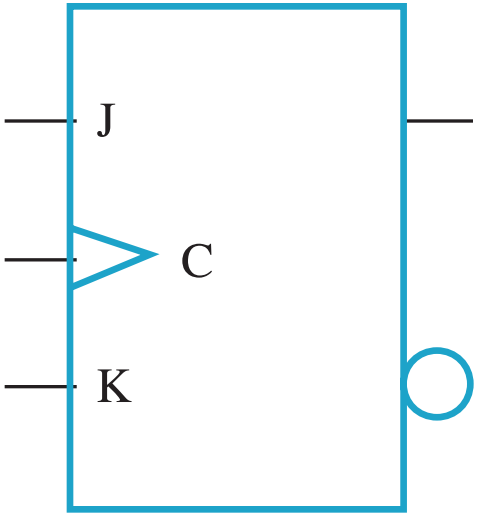
- Behavior
  - Same as S-R flip-flop with J analogous to S and K analogous to R
  - Except that J = K = 1 is allowed, and
  - For J = K = 1, the flip-flop changes to the *opposite state*
  - As a master-slave, has same "1s catching" behavior as S-R flip-flop
  - If the master changes to the wrong state, that state will be passed to the slave
    - E.g., if master falsely set by J = 1, K = 1 cannot reset it during the current clock cycle

J-K Flip-flop (continued)

- Implementation
  - To avoid 1s catching behavior, one solution used is to use an edge-triggered D as the core of the flip-flop



- **Symbol**

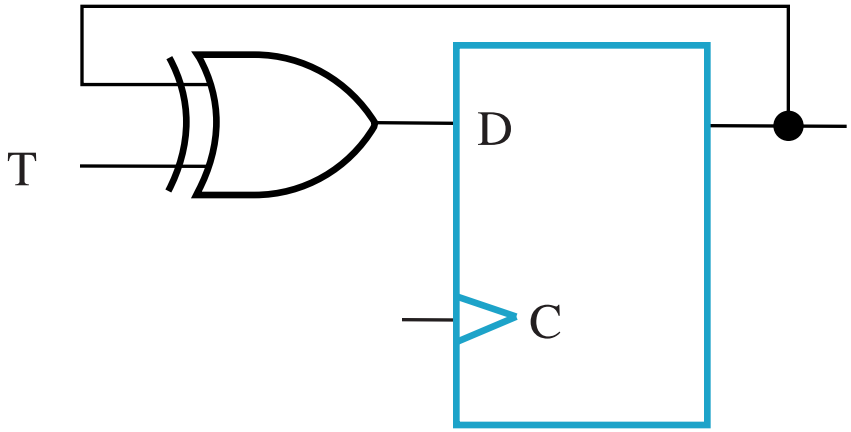


T Flip-flop

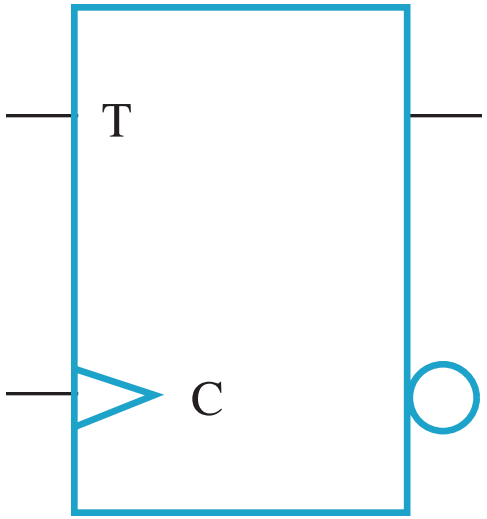
- Behavior
  - Has a single input T
    - For T = 0, no change to state
    - For T = 1, changes to opposite state
- Same as a J-K flip-flop with J = K = T
- As a master-slave, has same "1s catching" behavior as J-K flip-flop
- Cannot be initialized to a known state using the T input
  - Reset (asynchronous or synchronous) essential

T Flip-flop (continued)

- Implementation
  - To avoid 1s catching behavior, one solution used is to use an edge-triggered D as the core of the flip-flop



- **Symbol**



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

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

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

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

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

- 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

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

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

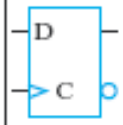
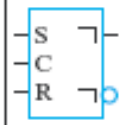
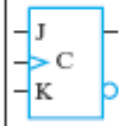
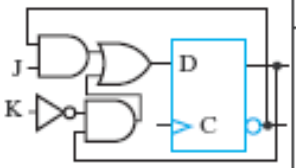
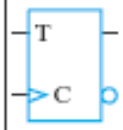
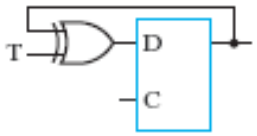
- Excitation Table

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

Summary

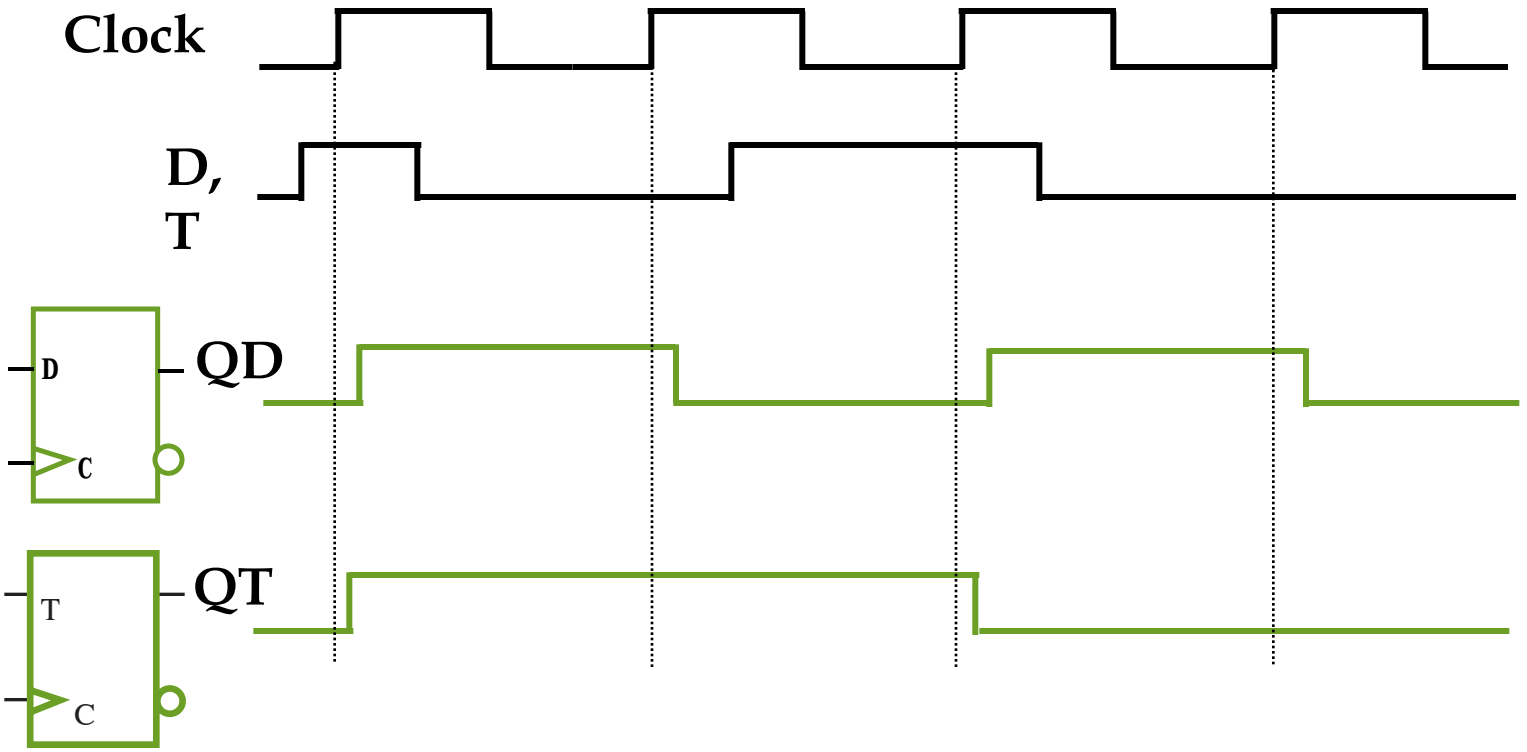
T 5-8

TABLE 5-8
Flip-Flop Logic, Characteristic Tables and Equations, and Excitation Tables

Type	Symbol	Logic Diagrams	Characteristic Table	Characteristic Equation	Excitation Table
D		See Figure 5-12	**D** | **Q(t+1)** | **Operation** 0 | 0 | Reset 1 | 1 | Set	$Q(t+1) = D(t)$	**Q(t+1)** | **D** | **Operation** 0 | 0 | Reset 1 | 1 | Set
SR		See Figure 5-9	**S** | **R** | **Q(t+1)** | **Operation** 0 | 0 | $Q(t)$ | No change 0 | 1 | 0 | Reset 1 | 0 | 1 | Set 1 | 1 | ? | Undefined	$Q(t+1) = S(t) + \overline{R}(t)\,Q(t)$	**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
JK			**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	$Q(t+1) = J(t)\,\overline{Q}(t) + \overline{K}(t)\,Q(t)$	**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			**T** | **Q(t+1)** | **Operation** 0 | $Q(t)$ | No change 1 | $\overline{Q}(t)$ | Complement	$Q(t+1) = T(t) \oplus Q(t)$	**Q(t+1)** | **T** | **Operation** $Q(t)$ | 0 | No change $\overline{Q}(t)$ | 1 | Complement

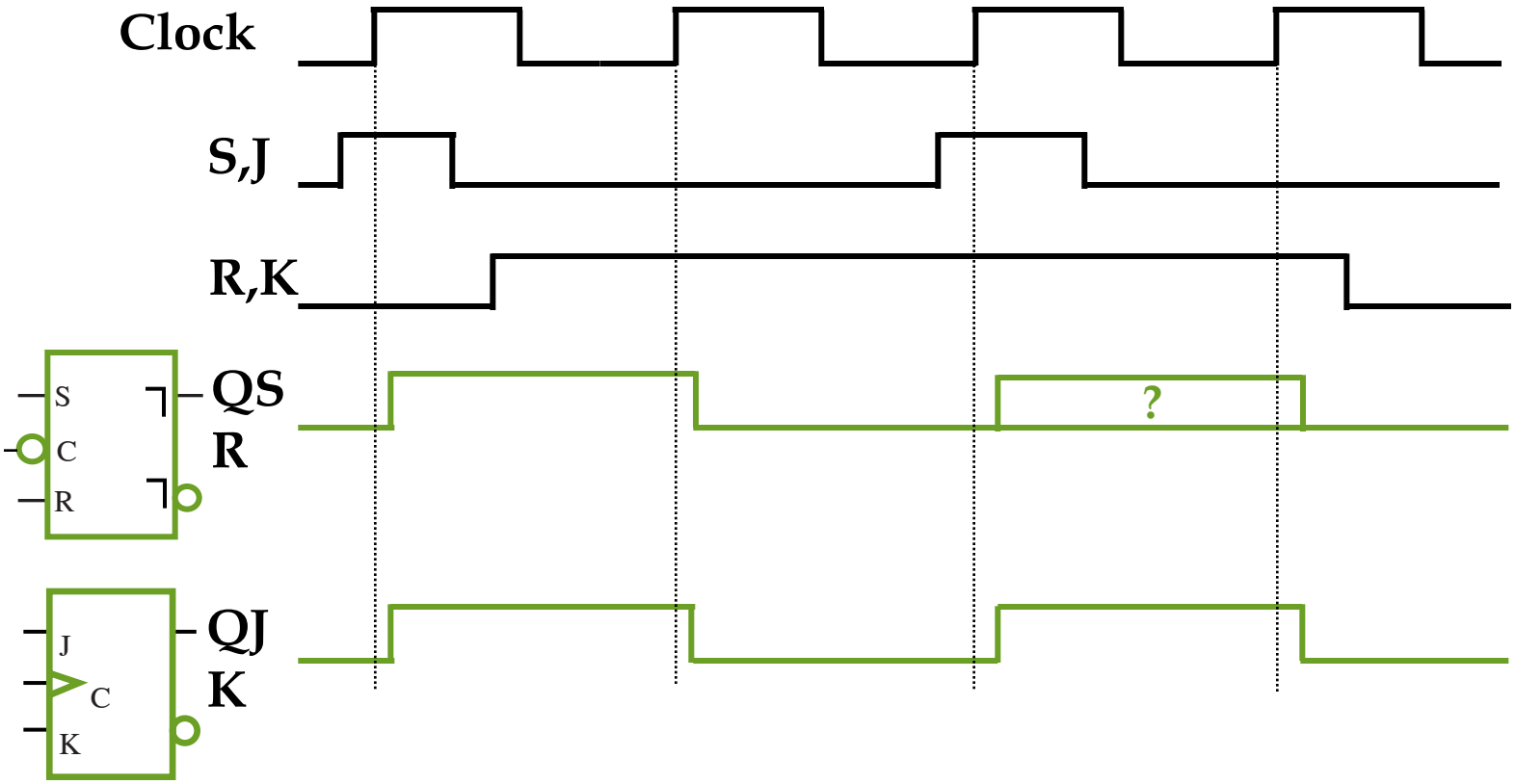
Flip-flop Behavior Example

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



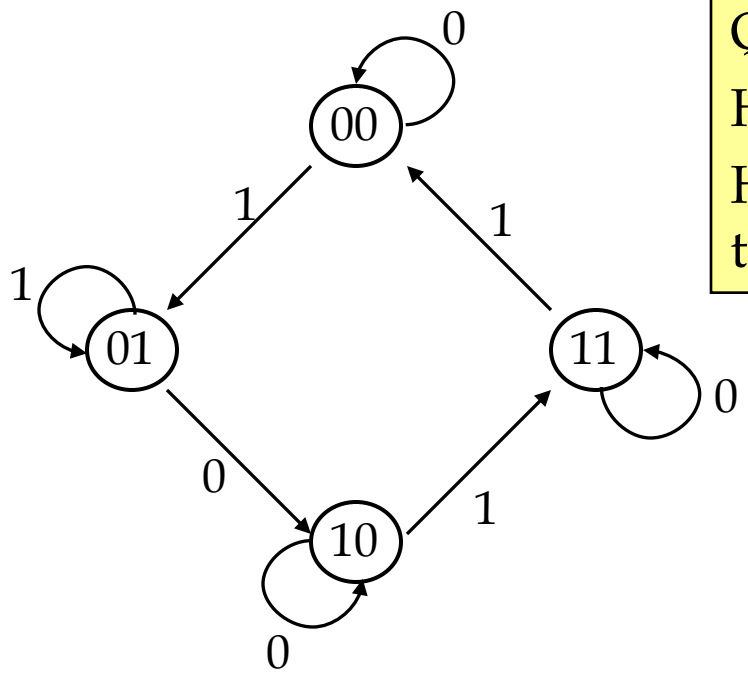
Flip-Flop Behavior Example (continued)

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



Design: Example #1 (1/5)

- Given the following state diagram, design the sequential circuit using *JK* flip-flops.



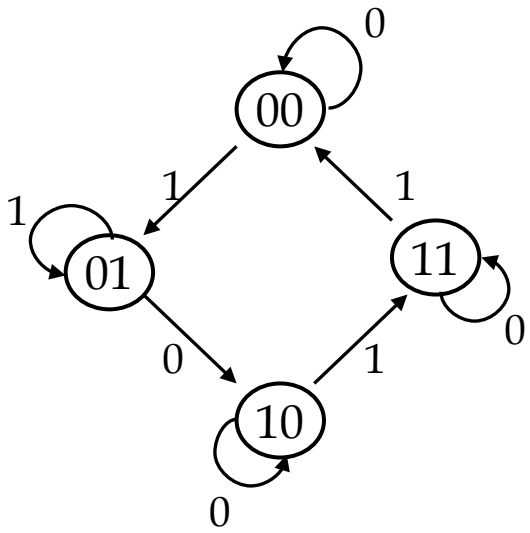
Questions:

How many flip-flops are needed?

How many input variable are there?

Design: Example #1 (2/5)

- Circuit state/excitation table, using *JK* flip-flops.



Present State	Next State x=0	Next State x=1
AB	A^+B^+	A^+B^+
00	00	01
01	10	01
10	10	11
11	11	00

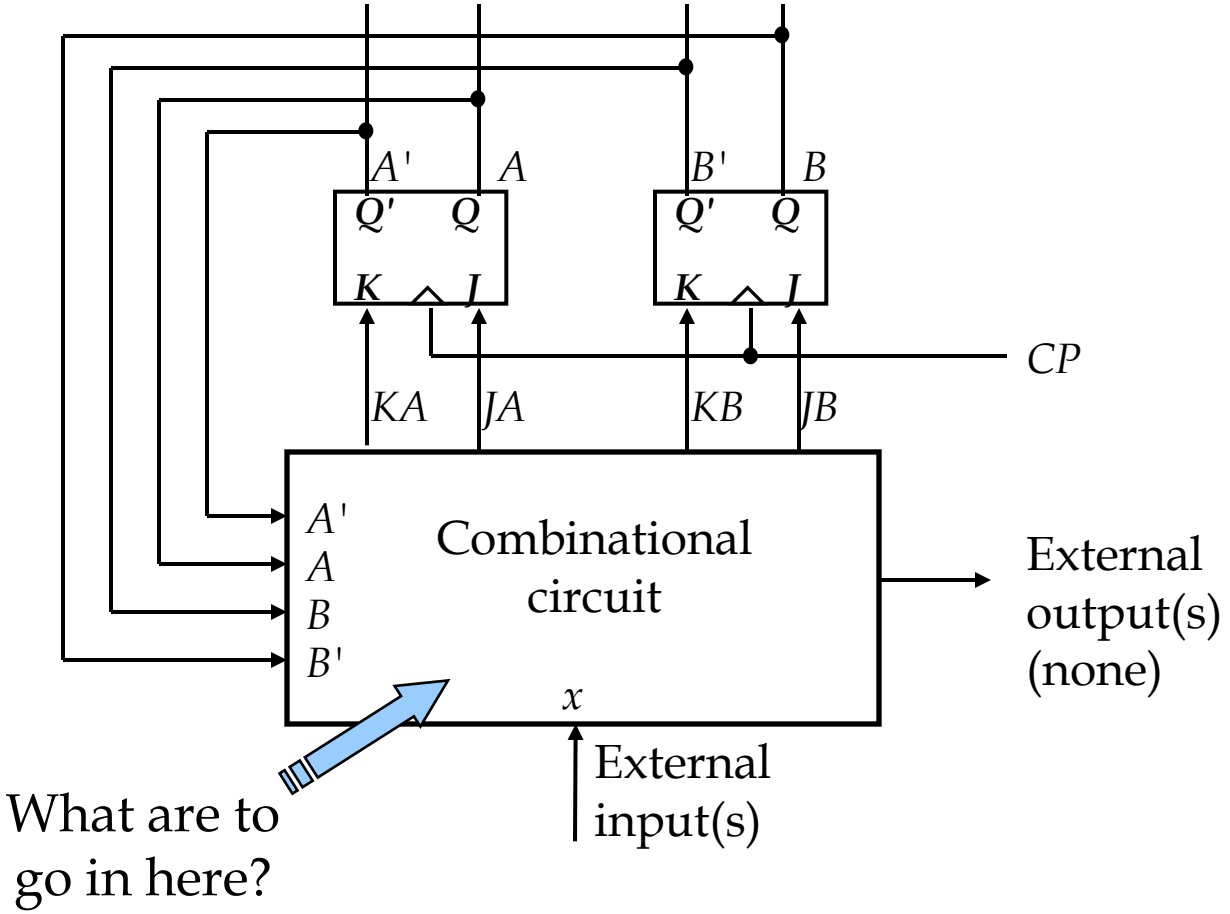
Q	Q^+	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

JK Flip-flop's excitation table.

Present state		Input	Next state		Flip-flop inputs			
A	B	x	A^+	B^+	JA	KA	JB	KB
0	0	0	0	0				
0	0	1	0	1				
0	1	0	1	0				
0	1	1	0	1				
1	0	0	1	0				
1	0	1	1	1				
1	1	0	1	1				
1	1	1	0	0				

Design: Example #1 (3/5)

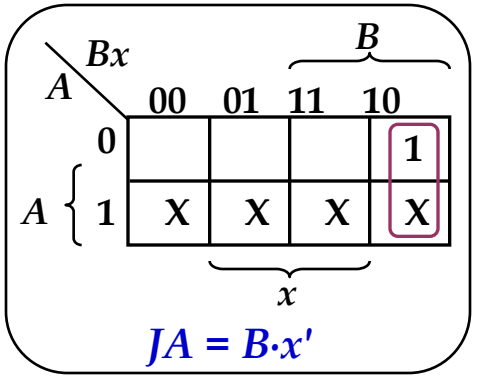
- Block diagram.

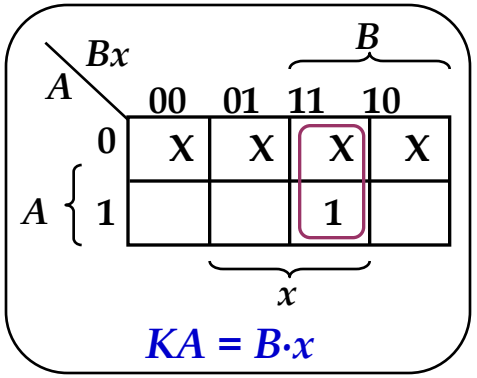


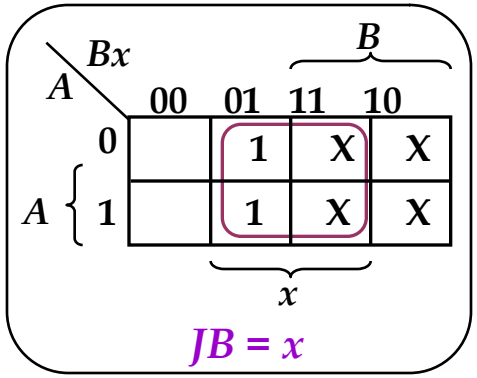
Design: Example #1 (4/5)

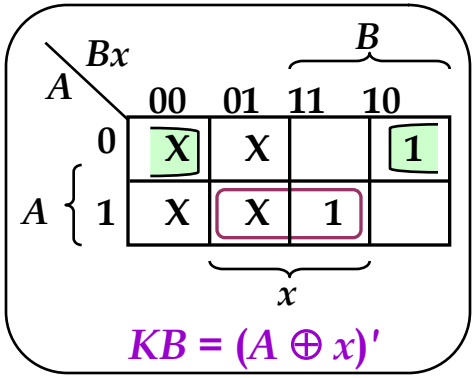
- From state table, get flip-flop input functions.

Present state		Input	Next state		Flip-flop inputs			
A	B	x	A^+	B^+	JA	KA	JB	KB
0	0	0	0	0	0	X	0	X
0	0	1	0	1	0	X	1	X
0	1	0	1	0	1	X	X	1
0	1	1	0	1	0	X	X	0
1	0	0	1	0	X	0	0	X
1	0	1	1	1	X	0	1	X
1	1	0	1	1	X	0	X	0
1	1	1	0	0	X	1	X	1









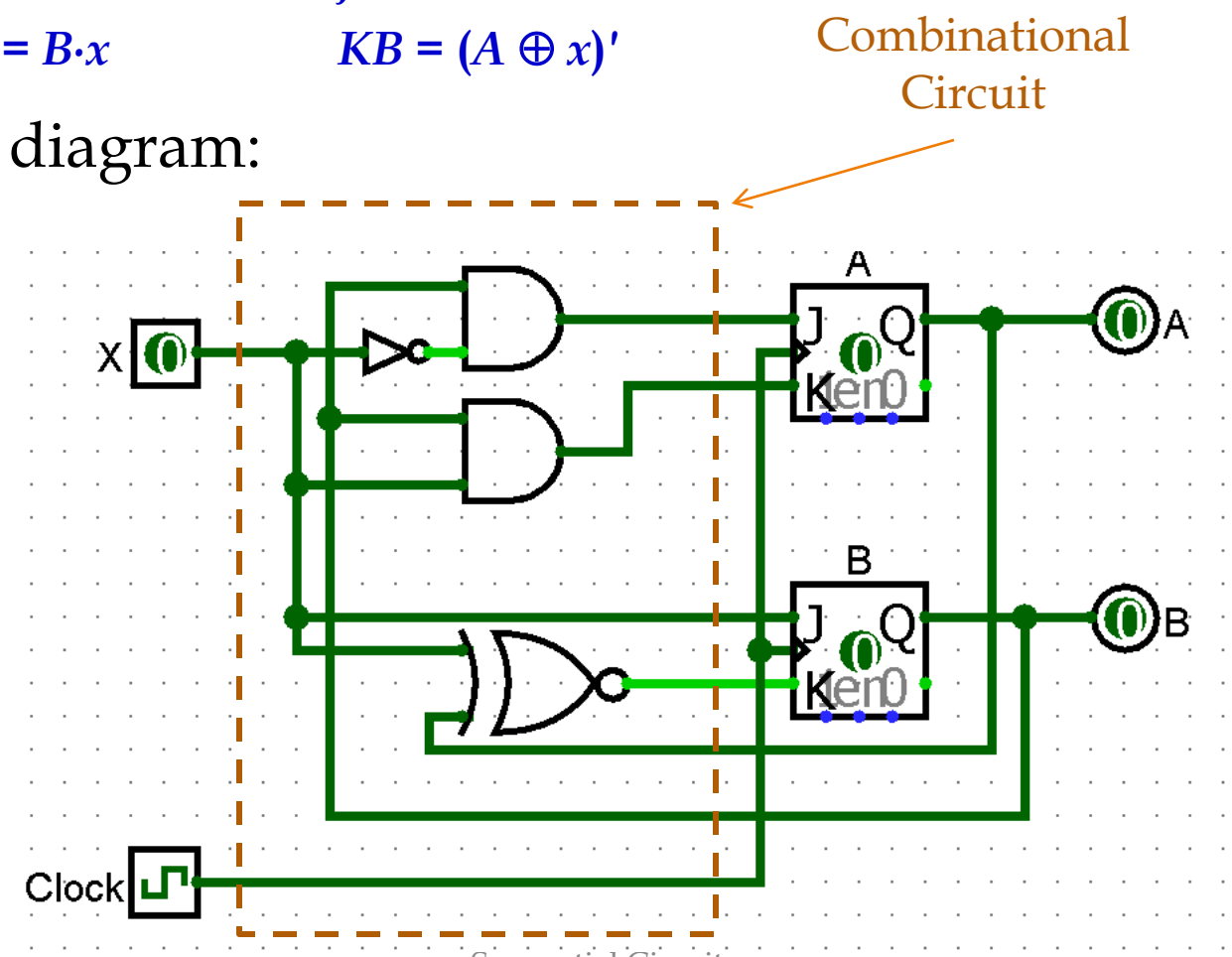
Design: Example #1 (5/5)

- Flip-flop input functions:

  $JA = B \cdot x'$ $\quad JB = x$

  $KA = B \cdot x$ $\quad KB = (A \oplus x)'$

- Logic diagram:



Design: Example #2 (1/3)

- Using *D* flip-flops, design the circuit based on the state table below. (Exercise: Design it using *JK* flip-flops.)

Present state		Input	Next state		Output
A	*B*	*x*	A^+	B^+	*y*
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	1	0	0
0	1	1	0	1	0
1	0	0	1	0	0
1	0	1	1	1	1
1	1	0	1	1	0
1	1	1	0	0	0

Design: Example #2 (2/3)

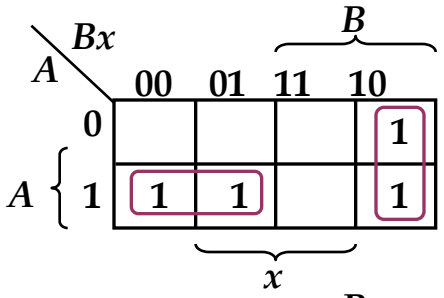
- Determine expressions for flip-flop inputs and the circuit output y.

Present state		Input	Next state		Output
A	B	x	A^+	B^+	y
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	1	0	0
0	1	1	0	1	0
1	0	0	1	0	0
1	0	1	1	1	1
1	1	0	1	1	0
1	1	1	0	0	0

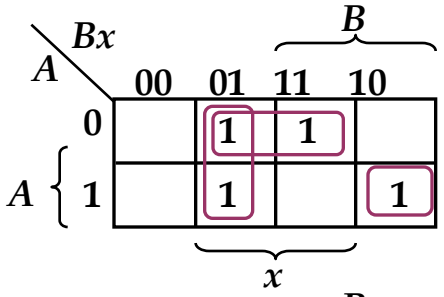
$DA(A,B,x) = \sum m(2,4,5,6)$

$DB(A,B,x) = \sum m(1,3,5,6)$

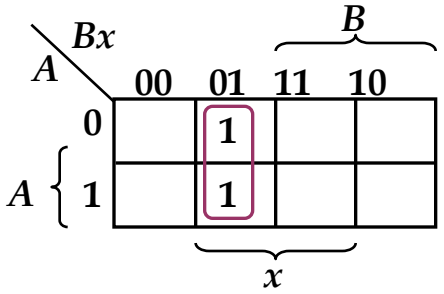
$y(A,B,x) = \sum m(1,5)$



$DA = A \cdot B' + B \cdot x'$



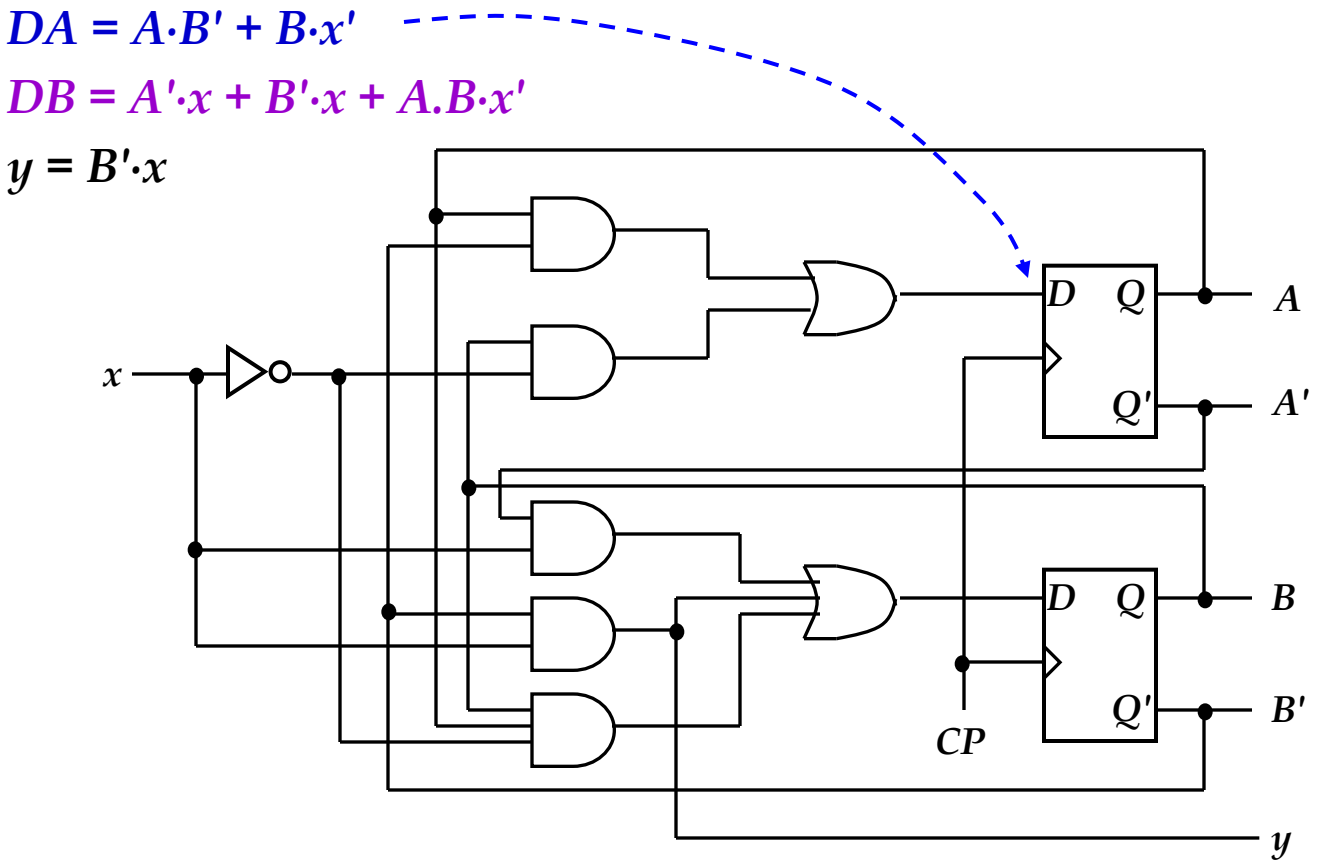
$DB = A' \cdot x + B' \cdot x + A \cdot B \cdot x'$



$y = B' \cdot x$

Design: Example #2 (3/3)

- From derived expressions, draw logic diagram:



Design: Example #3 (1/5)

- Design involving unused states. Use S-R Flip flop.

Preset State				Input		Next Sate				Output
A	B	C		X		A(t+1)	B(t+1)	C(t+1)		Y
0	0	1		0		0	0	1		0
0	0	1		1		0	1	0		0
0	1	0		0		0	1	1		0
0	1	0		1		1	0	0		0
0	1	1		0		0	0	1		0
0	1	1		1		1	0	0		0
1	0	0		0		1	0	1		0
1	0	0		1		1	0	0		1
1	0	1		0		0	0	1		0
1	0	1		1		1	0	0		1

Design: Example #3 (2/5)

- Design involving unused states.

Present state			Input	Next state			Flip-flop inputs						Output
A	B	C	x	A^+	B^+	C^+	SA	RA	SB	RB	SC	RC	y
0	0	1	0	0	0	1	0	X	0	X	X	0	0
0	0	1	1	0	1	0	0	X	1	0	0	1	0
0	1	0	0	0	1	1	0	X	X	0	1	0	0
0	1	0	1	1	0	0	1	0	0	1	0	X	0
0	1	1	0	0	0	1	0	X	0	1	X	0	0
0	1	1	1	1	0	0	1	0	0	1	0	1	0
1	0	0	0	1	0	1	X	0	0	X	1	0	0
1	0	0	1	1	0	0	X	0	0	X	0	X	1
1	0	1	0	0	0	1	0	1	0	X	X	0	0
1	0	1	1	1	0	0	X	0	0	X	0	1	1

Given these

Derive these

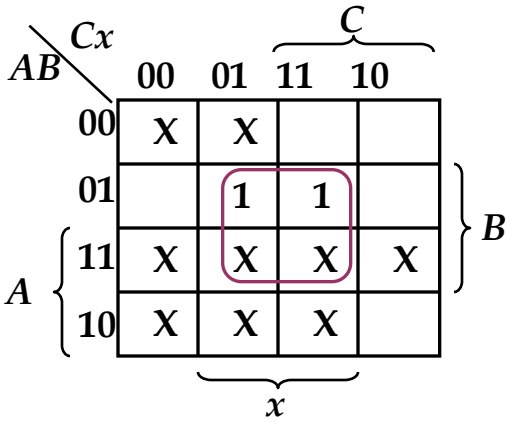
Are there other unused states?

Unused state 000:

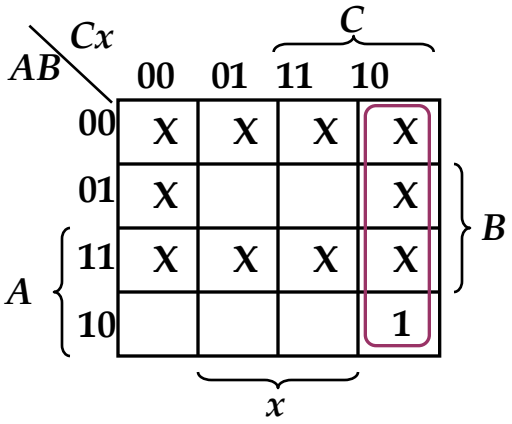
0	0	0	0	X	X	X	X	X	X	X	X	X	X
0	0	0	1	X	X	X	X	X	X	X	X	X	X

Design: Example #3 (3/5)

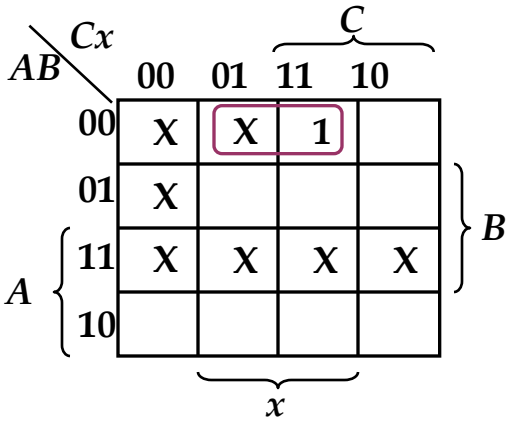
- From state table, obtain expressions for flip-flop inputs.



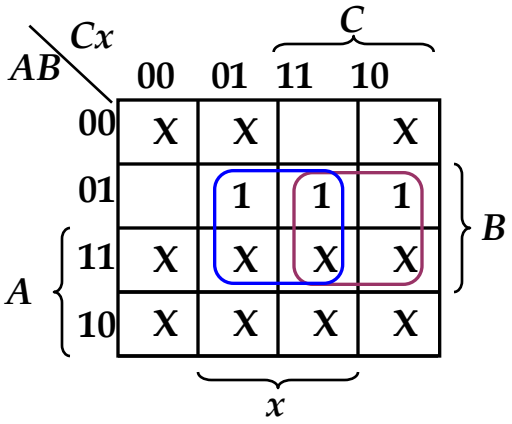
SA = ?



RA = ?



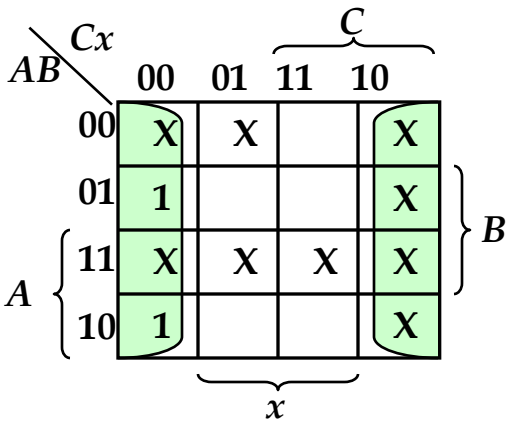
SB = ?



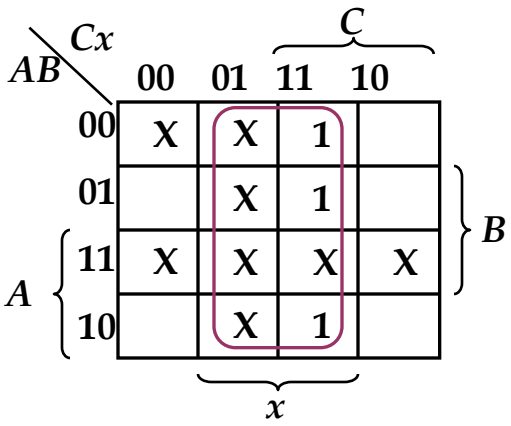
RB = ?

Design: Example #3 (4/5)

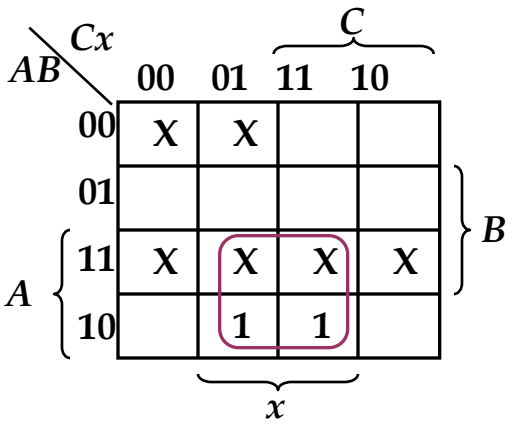
- From state table, obtain expressions for flip-flop inputs (cont'd).



SC = ?



RC = ?



y = ?

Design: Example #3 (5/5)

- From derived expressions, draw the logic diagram:

$SA = B \cdot x$	$SB = A' \cdot B' \cdot x$	$SC = x'$	$y = A \cdot x$
$RA = C \cdot x'$	$RB = B \cdot C + B \cdot x$	$RC = x$