Design of Digital Systems II
Sequential Logic Design Principles (2)

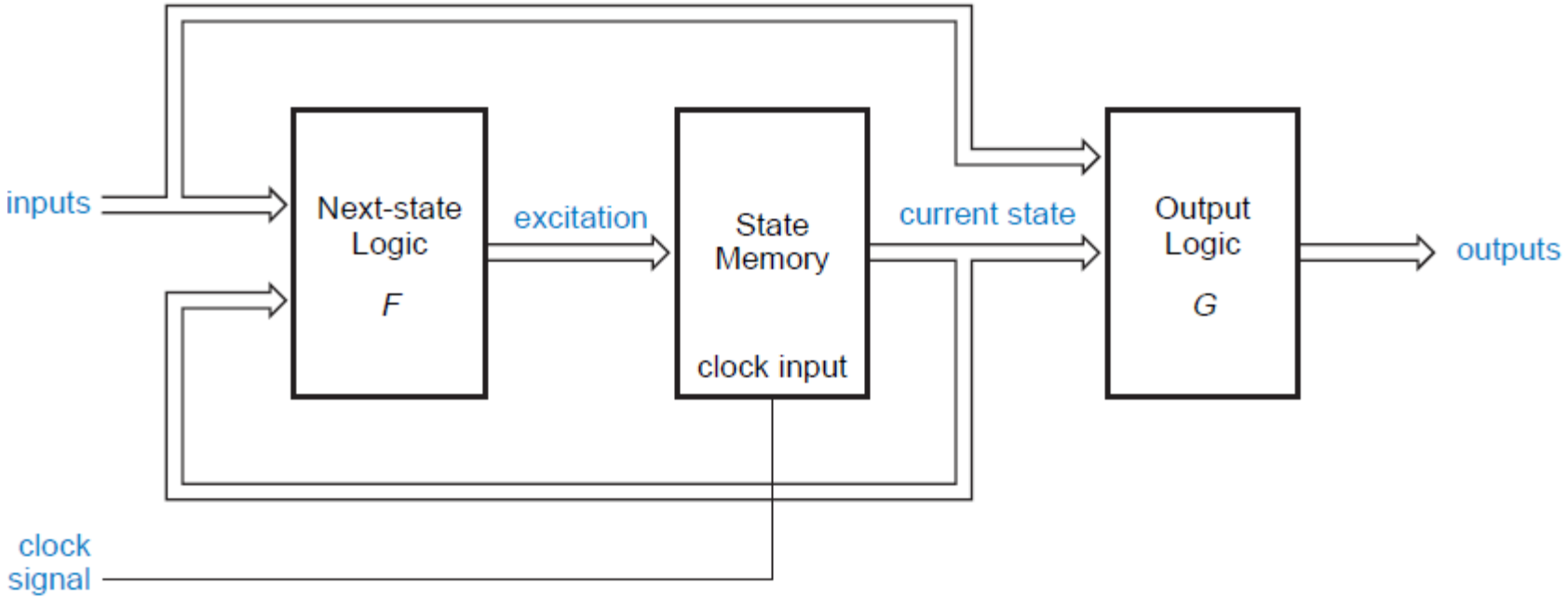
Moslem Amiri, Václav Přenosil
Masaryk University

Resource: “Digital Design: Principles & Practices”
by John F. Wakerly

Clocked Synchronous State-Machine Analysis

State-Machine Structure

- **General structure** of a clocked synchronous state machine (**Mealy machine**):



- **State memory** is a set of n flip-flops that store current state of machine, and has 2^n distinct states.
- Flip-flops are all connected to a common clock signal that causes **flip-flops** to **change state at each tick of clock**.

State-Machine Structure

- For positive-edge-triggered D flip-flops, a tick is rising edge of clock signal.
- In previous figure, both F and G are combinational logic circuits.

  Next state = F (current state, input)

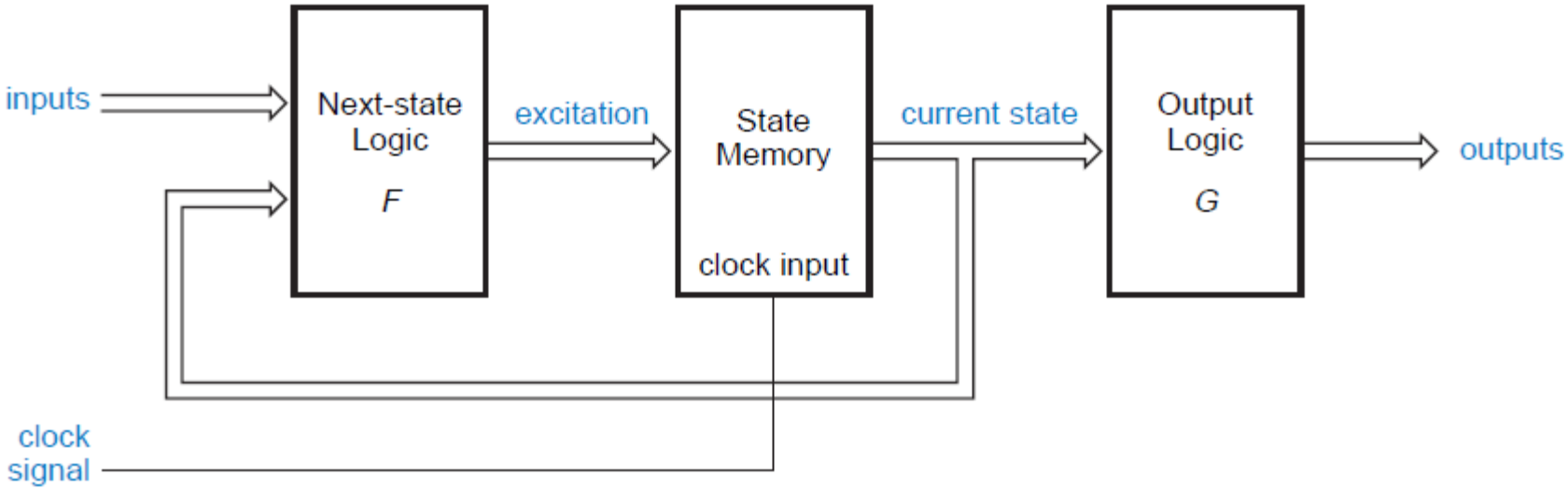
  Output = G (current state, input)

Output Logic

- A sequential circuit whose **output depends on both state and input** as shown in previous figure is called a **Mealy machine**.
- In some sequential circuits, **output depends on state alone**:

  Output = G (current state)

  such a circuit is called a **Moore machine**:



Characteristic Equations

- **Functional behavior of a latch or flip-flop** can be described formally by a **characteristic equation** that specifies flip-flop's next state (Q*) as a function of its current state (Q) and inputs.

Device Type	Characteristic Equation
D latch	Q* = D
Edge-triggered D flip-flop	Q* = D
D flip-flop with enable	Q* = EN.D + EN'.Q
T flip-flop	Q* = Q'
T flip-flop with enable	Q* = EN.Q' + EN'.Q

Analysis of State Machines with D Flip-Flops

- **Formal definition of a state machine**:

  Next state = F (current state, input)

  Output = G (current state, input)

- Recalling that "state" embodies all we need to know about past history of circuit, first equation tells us that what we next need to know can be determined from what we currently know and current input.
- Second equation tells us that current output can be determined from the same information.
- Goal of sequential circuit analysis is to determine next-state and output functions so that behavior of a circuit can be predicted.

Analysis of State Machines with D Flip-Flops

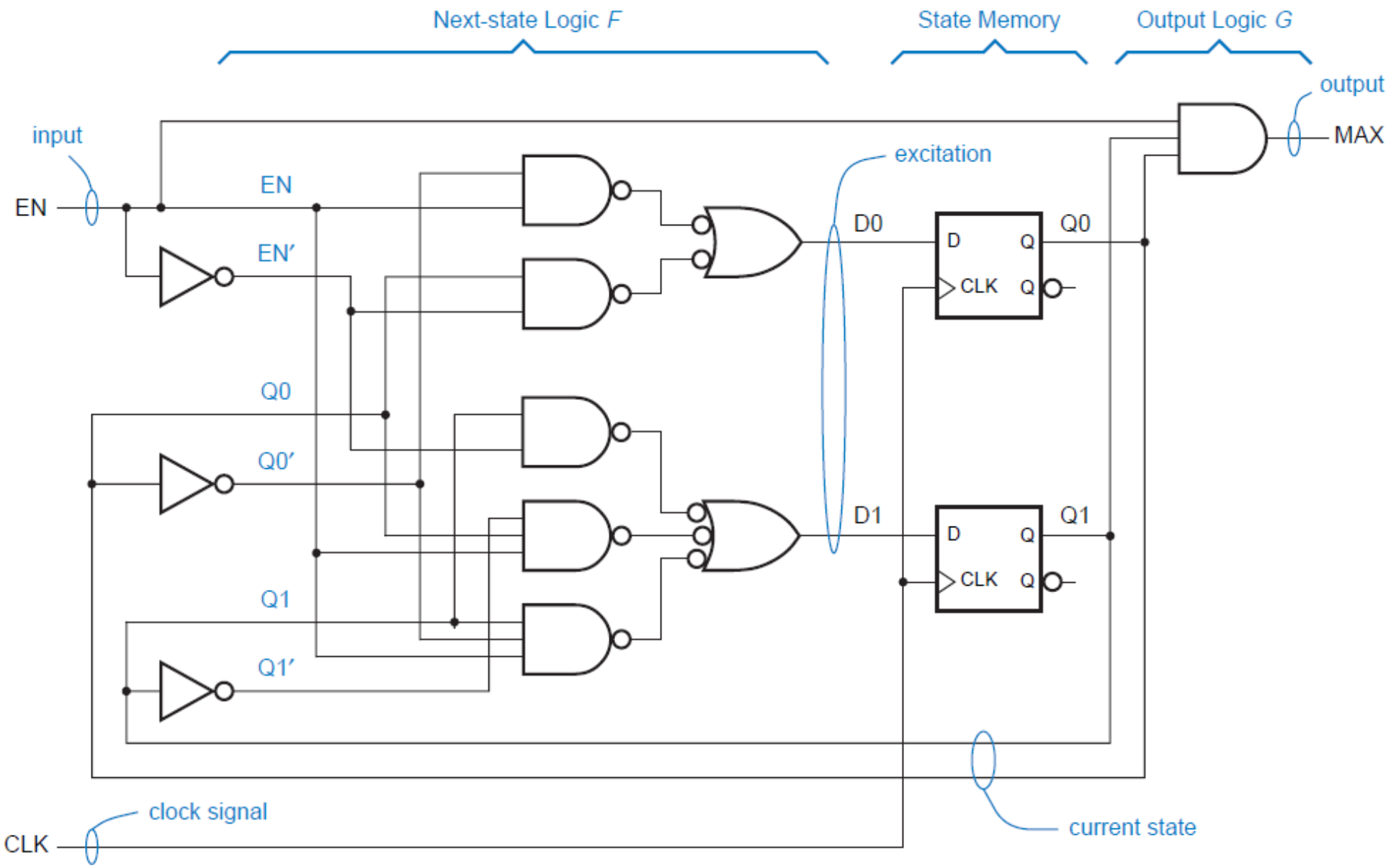
- **Analysis** of a clocked synchronous state machine has three steps:

  1) Determine **next-state and output functions** F and G,

  2) Use F and G to construct a **state/output table** that completely specifies next state and output of circuit for every possible combination of current state and input,

  3) (Optional) Draw a **state diagram** that presents information from previous step in graphical form.

Analysis of State Machines with D Flip-Flops



Analysis of State Machines with D Flip-Flops

- To determine next-state function F, we must first consider behavior of state memory. At rising edge of clock signal, each D flip-flop samples its D input and transfers this value to its Q output; characteristic equation of a D flip-flop is **Q*=D**. Therefore, **to determine Q*, we must first determine current value of D**.
- In previous figure, there are two D flip-flops with signal names on their outputs Q0 and Q1. These two outputs are state variables; their value is current state of machine.
- Signals on D inputs, D0 and D1, provide **excitation** for D flip-flops at each clock tick.
- Logic equations that express excitation signals as functions of current state and input are called **excitation equations** and can be derived from circuit diagram:

  D0 = Q0.EN' + Q0'.EN

  D1 = Q1.EN' + Q1'.Q0.EN + Q1.Q0'.EN

Analysis of State Machines with D Flip-Flops

- Using characteristic equation of D flip-flops, Q* = D, we can describe next-state function of example machine with equations for next value of state variables:

  Q0* = D0

  Q1* = D1

- Substituting excitation equations for D0 and D1, we can write

  Q0* = Q0.EN' + Q0'.EN

  Q1* = Q1.EN' + Q1'.Q0.EN + Q1.Q0'.EN

  These equations, which express next value of state variables as a function of current state and input, are called **transition equations**.

- For each combination of current state and input value, transition equations predict next state. **Each state is described by two bits**, current values of Q0 and Q1: **(Q1 Q0) = 00, 01, 10, or 11**. For each state, our example machine has just **two possible input values**, **EN = 0 or EN = 1**, so there are a total of **8 state/input combinations.**

Analysis of State Machines with D Flip-Flops

- In general, a machine with *s* state bits and *i* inputs has 2^{s+i} state/input combinations.
- Transition (a), state (b), and state/output (c) tables for our example machine:

	EN	
Q1Q0	0	1
00	00	01
01	01	10
10	10	11
11	11	00
	Q1*Q0*	

(a)

	EN	
S	0	1
A	A	B
B	B	C
C	C	D
D	D	A
	S*	

(b)

	EN	
S	0	1
A	A, 0	B, 0
B	B, 0	C, 0
C	C, 0	D, 0
D	D, 0	A, 1
	S*, MAX	

(c)

- Transition table (a) is created by evaluating transition equations for every possible state/input combination.
- Function of our example machine is apparent from its transition table – it is a **2-bit binary counter with an enable** input EN.

Analysis of State Machines with D Flip-Flops

- We can assign alphanumeric **state names** to each state. Substituting state names for combinations of Q1 and Q0 (and Q1* and Q0*) in previous table (a) produces **state table** in (b).
- Here “S” denotes current state, and “S*” denotes next state of machine.
- In **complex machines** we can use **state names that have meaning**. So, a state table is easier to understand but contains less information than a transition table.
- In our example (a Mealy machine), there is a single **output equation**:

  MAX = Q1.Q0.EN
- Output behavior can be combined with next-state information to produce a **state/output table** as shown in previous table (c).

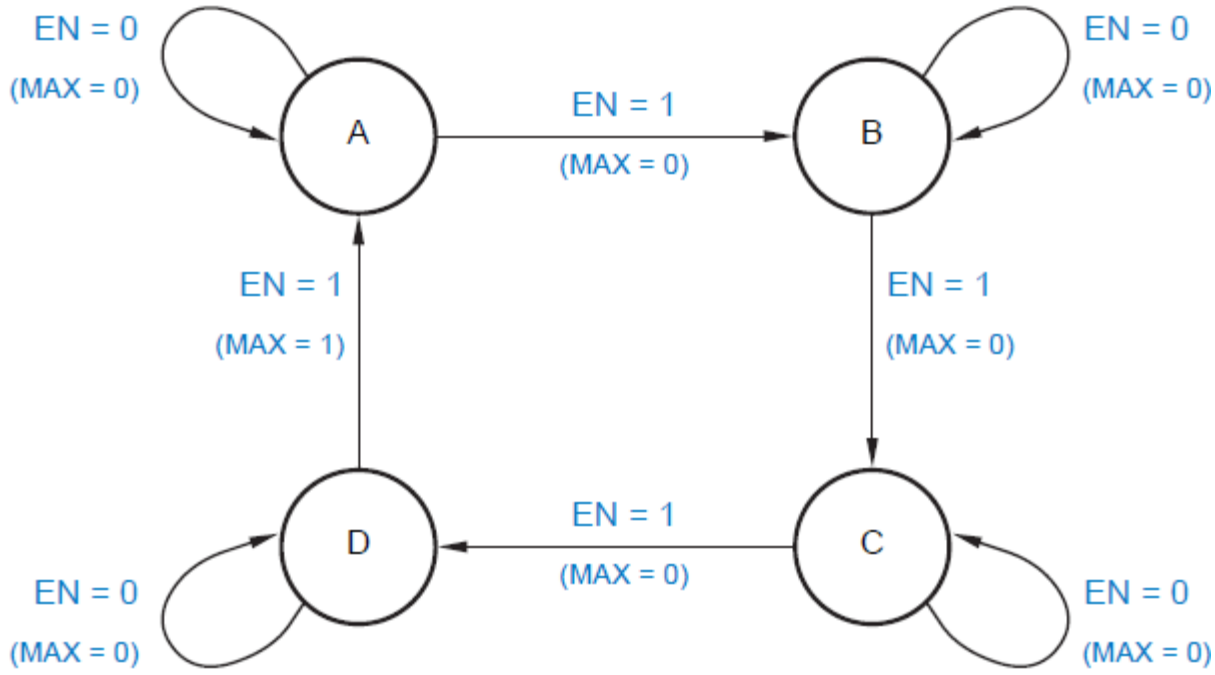
Analysis of State Machines with D Flip-Flops

- **State/output tables for Moore machines** are slightly simpler. For example, in circuit of example machine **suppose we removed EN signal from AND gate that produces MAX output**, producing a Moore-type output MAXS.
- MAXS is a function of state only, and state/output table can list MAXS in a single column, independent of input values:

	EN		
S	0	1	**MAXS**
A	A	B	0
B	B	C	0
C	C	D	0
D	D	A	1
	S*		

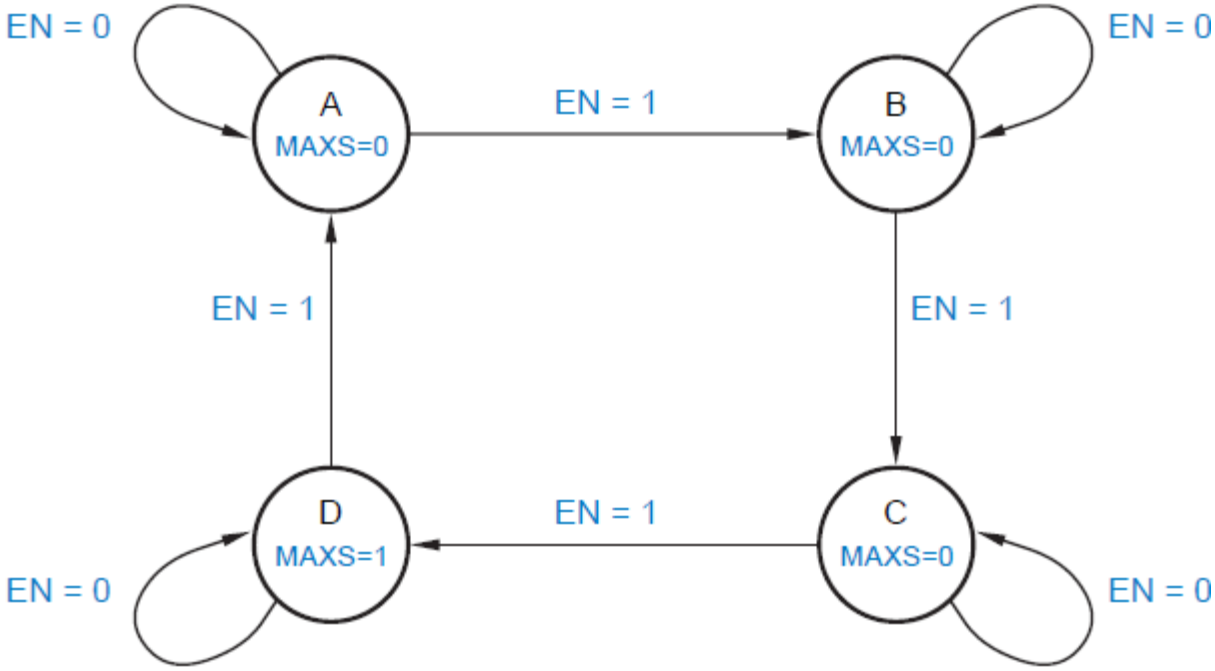
Analysis of State Machines with D Flip-Flops

- A **state diagram** presents information from state/output table in a graphical format. It has one circle (or **node**) for each state, and an arrow (or **directed arc**) for each transition.
- State diagram for our example state machine:



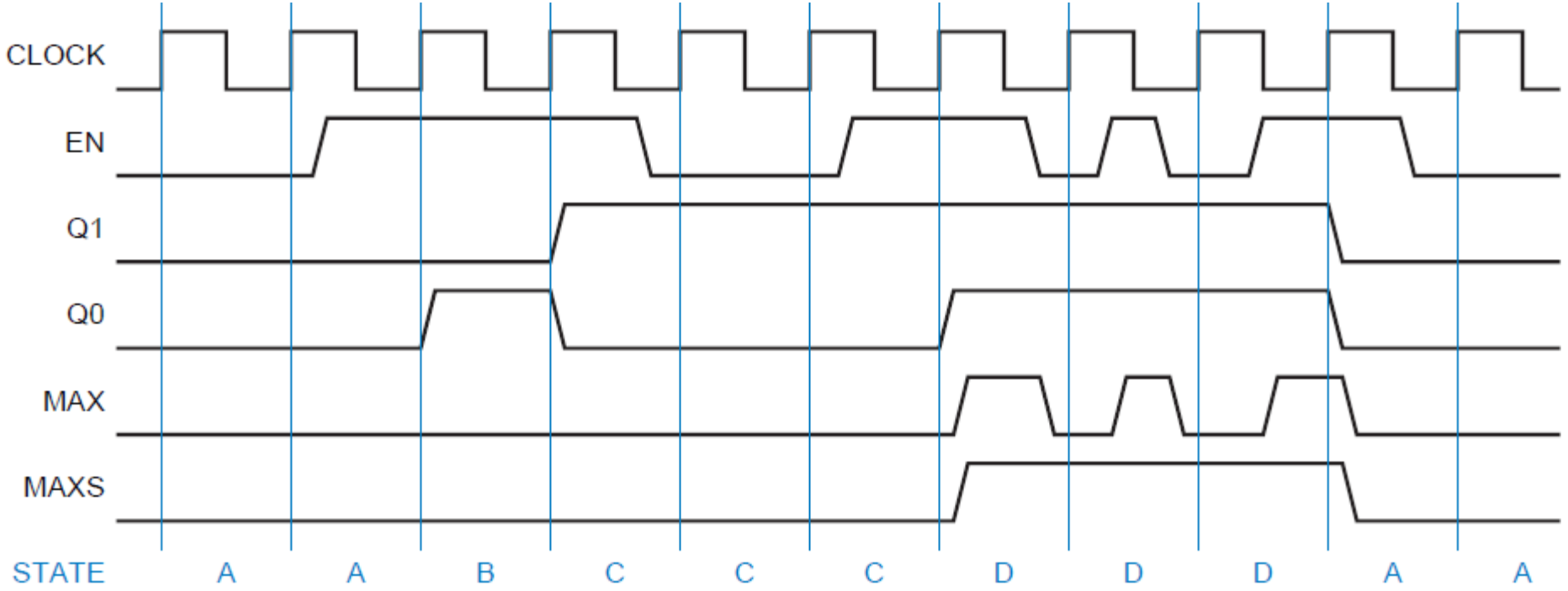
Analysis of State Machines with D Flip-Flops

- In state diagram for a **Moore machine**, output values can be shown inside each state circle, since they are functions of state only.



Analysis of State Machines with D Flip-Flops

- Using transition, state, and output tables, we can construct a **timing diagram** that shows behavior of a state machine for any desired starting state and input sequence. For our example machine:



- Since **MAX** is a **Mealy-type output**, its value is affected by EN at all times. Value of a **Moore-type output MAXS** depends only on state. Also, **combinational-logic delay** of output circuits is reflected in slightly later changes in MAX and MAXS.

Analysis of State Machines with D Flip-Flops

- In summary, **detailed steps for analyzing** a clocked synchronous state machine are as follows:

  1) Determine excitation equations for flip-flop control inputs,

  2) Substitute excitation equations into the flip-flop characteristic equations to obtain transition equations,

  3) Use transition equations to construct a transition table,

  4) Determine output equations,

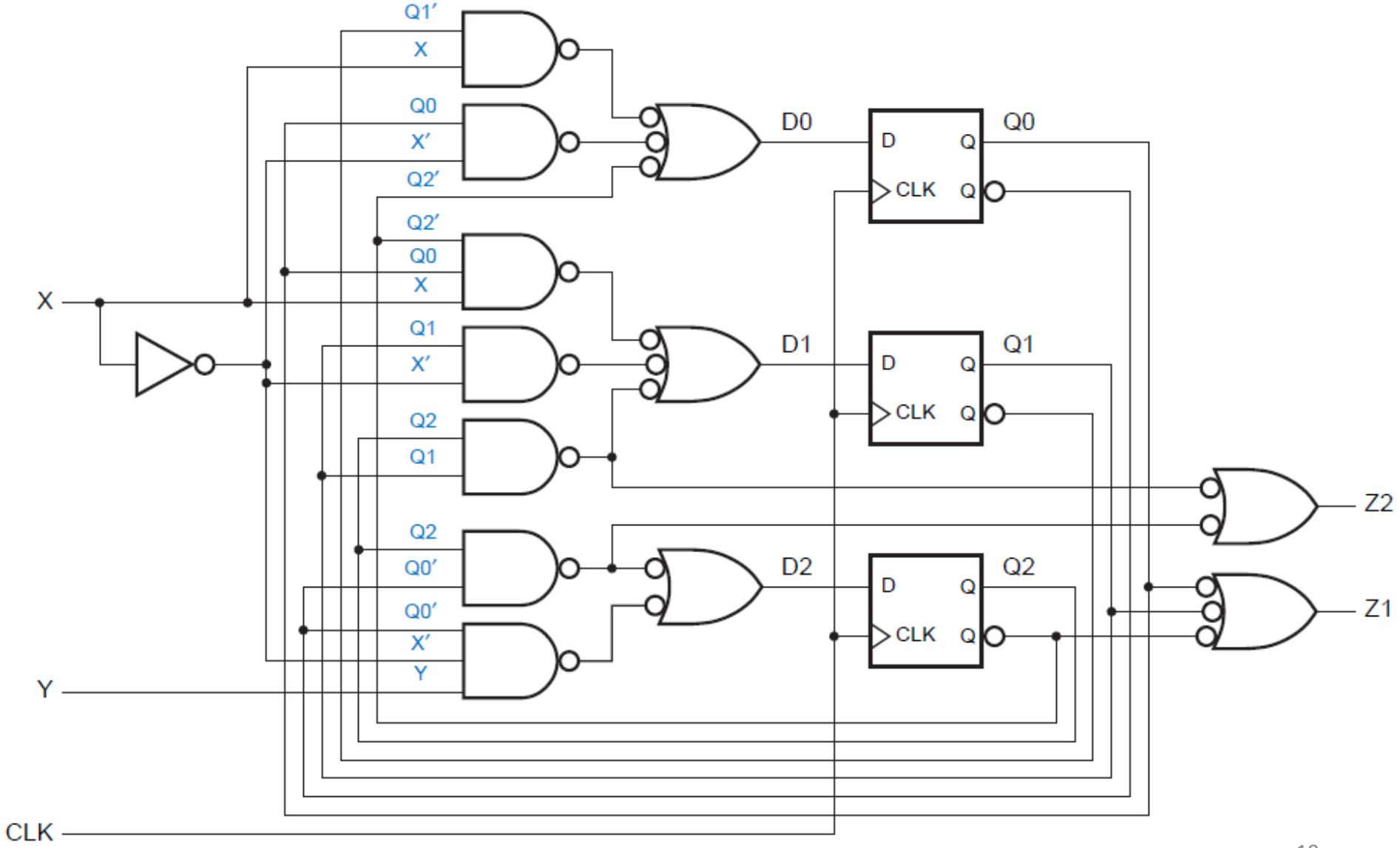
  5) Add output values to transition table for each state (Moore) or state/input combination (Mealy) to create a transition/output table,

  6) Name states and substitute state names for state-variable combinations in transition/output table to obtain a state/output table,

  7) (Optional) Draw a state diagram corresponding to state/output table.

Analysis of State Machines with D Flip-Flops

- Analyze following clocked synchronous state machine:



Analysis of State Machines with D Flip-Flops

- **Excitation equations**:

  D0 = Q1'.X + Q0.X' + Q2

  D1 = Q2'.Q0.X + Q1.X' + Q2.Q1

  D2 = Q2.Q0' + Q0'.X'.Y

- **Substituting into characteristic equation for D flip-flops**, we obtain **transition equations**:

  Q0* = Q1'.X + Q0.X' + Q2

  Q1* = Q2'.Q0.X + Q1.X' + Q2.Q1

  Q2* = Q2.Q0' + Q0'.X'.Y

- A transition table based on these equations is shown in following table (a).
- Two **output equations**:

  Z1 = Q2 + Q1' + Q0'

  Z2 = Q2.Q1 + Q2.Q0'

- Resulting output values are shown in last column of (a).

Analysis of State Machines with D Flip-Flops

- Transition/output and state/output tables:

	XY				
Q2Q1Q0	00	01	10	11	**Z1Z2**
000	000	100	001	001	10
001	001	001	011	011	10
010	010	110	000	000	10
011	011	011	010	010	00
100	101	101	101	101	11
101	001	001	001	001	10
110	111	111	111	111	11
111	011	011	011	011	11
	Q2*Q1*Q0*				

(a)

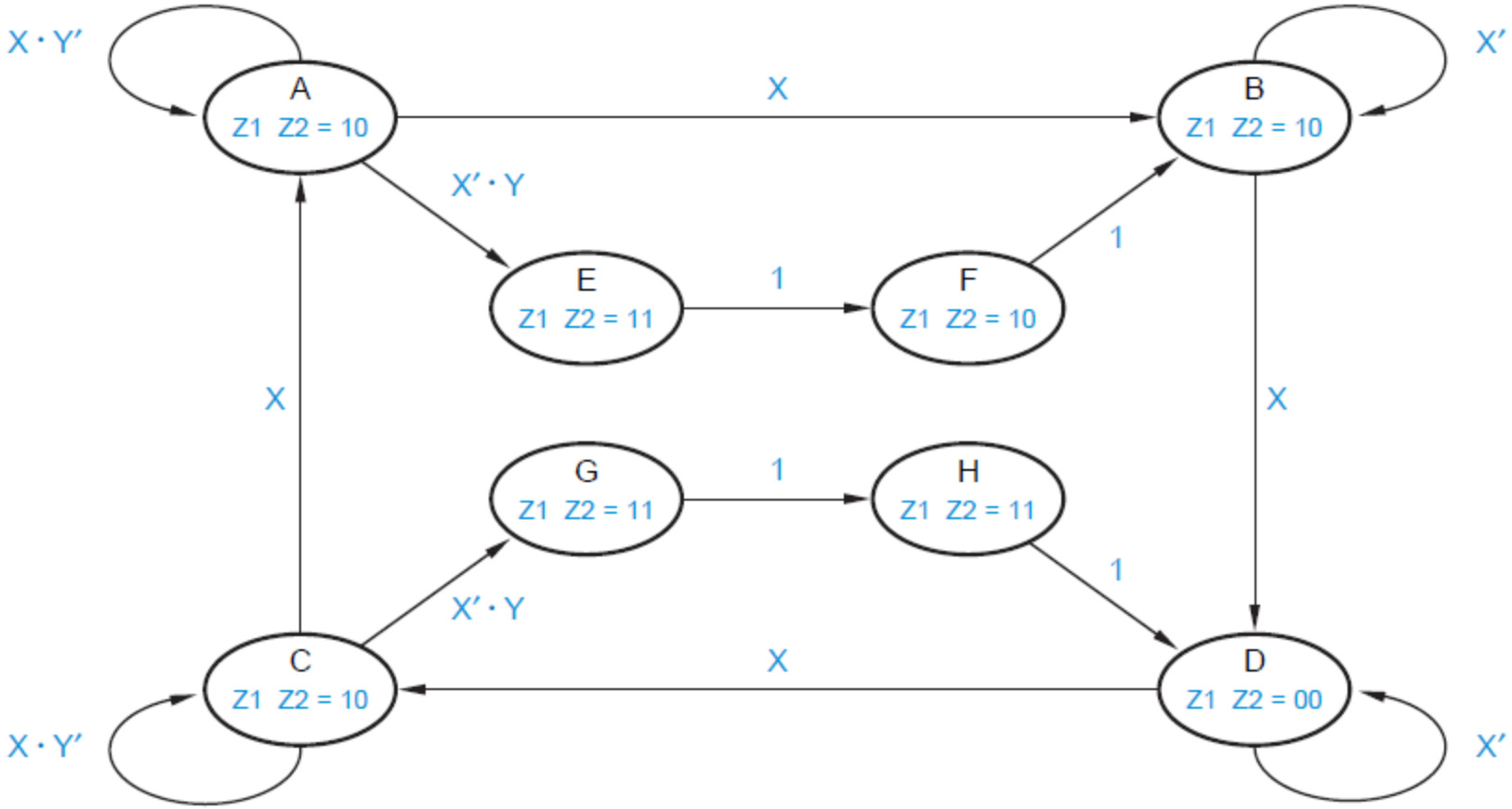
	XY				
S	00	01	10	11	**Z1Z2**
A	A	E	B	B	10
B	B	B	D	D	10
C	C	G	A	A	10
D	D	D	C	C	00
E	F	F	F	F	11
F	B	B	B	B	10
G	H	H	H	H	11
H	D	D	D	D	11
	S*				

(b)

- Assigning state names A-H, we obtain state/output table shown in (b).

Analysis of State Machines with D Flip-Flops

- A **state diagram** for example machine:



- Since our example is a Moore machine, output values are written with each state.

Analysis of State Machines with D Flip-Flops

- In previous diagram, each **arc** is labeled with a **transition expression**; a transition is taken for input combinations for which transition expression is 1. Transitions labeled “1” are always taken.
- **Transition expressions on arcs** leaving a particular state must be **mutually exclusive** and **all inclusive**, as explained below:

  a) No two transition expressions can equal 1 for same input combination, since a machine can’t have two next states for one input combination.

  b) For every possible input combination, some transition expression must equal 1, so that all next states are defined.
- Starting with state table, a **transition expression** for a particular current state and next state can be written as a **sum of minterms** for input combinations that cause that transition. If desired, the expression can then be **minimized** to give information in a more compact form.

Clocked Synchronous State-Machine Design

Introduction

- Steps for **designing a clocked synchronous state machine**, starting **from a word description or specification**, are **reverse of analysis** steps:

  1) Construct a state/output table corresponding to word description or specification, using mnemonic names for states (it is also possible to start with a state diagram),

  2) (Optional) Minimize number of states in state/output table,

  3) Choose a set of state variables and assign state-variable combinations to named states,

  4) Substitute state-variable combinations into state/output table to create a transition/output table that shows desired next state-variable combination and output for each state/input combination,

  5) Choose a flip-flop type (e.g., D) for state memory,

  6) Construct an excitation table that shows excitation values required to obtain desired next state for each state/input combination,

Introduction

7) Derive excitation equations from excitation table,

8) Derive output equations from transition/output table,

9) Draw a logic diagram that shows state-variable storage elements and realizes required excitation and output equations.

State-Table Design Example

- Design a clocked synchronous state machine with two inputs, A and B, and a single output Z that is 1 if:

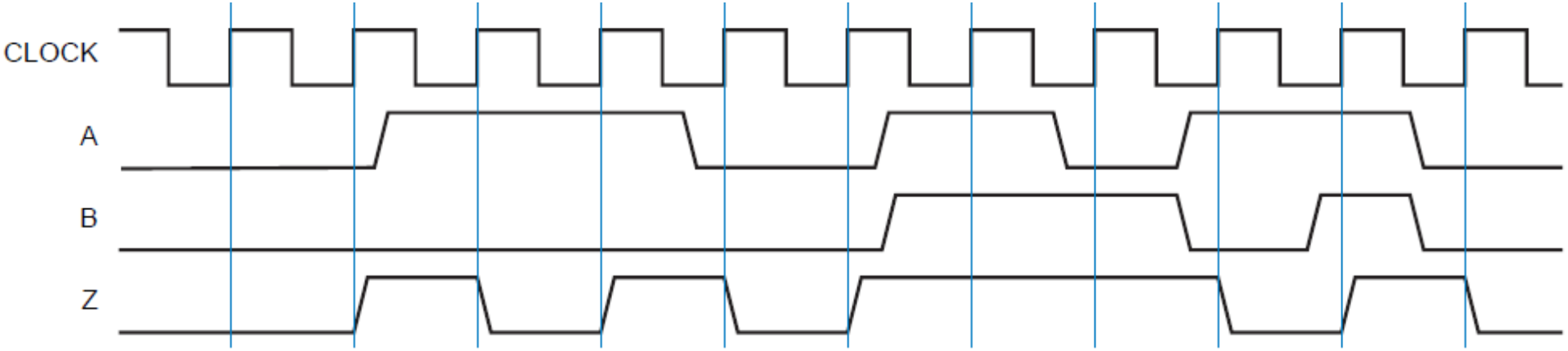
  - A had the same value at each of two previous clock ticks, **or**

  - B has been 1 since the last time that the first condition was true.

  Otherwise, output should be 0.

- If meaning of a specification is ambiguous, as an additional "**hint**" or requirement, state-table design problems often include **timing diagrams** that show state machine's expected behavior for one or more sequences of inputs.

- Following figure is such a timing diagram for our example state machine:



State-Table Design Example

- **First step** in state-table design is to construct a **template**.
- From word description, we know that our example is a **Moore machine** (output depends on what happened in previous clocks). Thus, we provide one next-state column for each possible input combination and a single column for output values:

		A B				
Meaning	S	00	01	11	10	Z
Initial state	INIT					0
	. . .					
	. . .					
	. . .					
		S*				

- We have written **order of input combinations** in **Karnaugh-map order** to simplify derivation of excitation equations later.
- In a Mealy machine we would omit output column and write output values along with next-state values under each input combination.
- Leftmost column is a reminder of meaning of each state or the “history” associated with it.

State-Table Design Example

- We assume that when power is first applied to system, machine enters an **initial state**. We can also fill in value of Z for INIT state; it should be 0 because there were no inputs beforehand.
- Next, we must fill in next-state entries for INIT row. Z output can't be 1 until we have seen at least two inputs on A, so we will provide two states, A0 and A1, that "remember" value of A on previous clock tick:

		A B				
Meaning	S	00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0					0
Got a 1 on A	A1					0
		S*				

- In both of these states, Z is 0, since we have not satisfied conditions for a 1 output yet.
- A0 means "Got A=0 on previous tick, A≠0 on tick before that, and B≠1 at some time since the previous pair of equal A inputs". State A1 is defined similarly.

State-Table Design Example

- In state A0 (A was 0 at previous clock tick), if A is 0 again, we go to a new state OK with Z=1:

		A B				
Meaning	S	00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK	OK	A1	A1	0
Got a 1 on A	A1					0
Got two equal A inputs	OK					1
		S*				

- If A is 1, we don't have two equal inputs in a row, so we go to state A1 to remember that we just got a 1.
- Likewise in state A1, we go to OK if we get a second 1 input in a row, or to A0 if we get a 0:

		A B				
Meaning	S	00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK	OK	A1	A1	0
Got a 1 on A	A1	A0	A0	OK	OK	0
Got two equal A inputs	OK					1
		S*				

State-Table Design Example

- Once we get into OK state, machine description tells us we can stay there as long as B=1, irrespective of A input:

		A B				
Meaning	S	00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK	OK	A1	A1	0
Got a 1 on A	A1	A0	A0	OK	OK	0
Got two equal A inputs	OK	?	OK	OK	?	1
		S*				

- If B=0, we have to look for two 1s or two 0s in a row on A again. However, we have a problem in this case. Current A input may or may not be the second equal input in a row. We defined OK state too broadly. Problem is solved by splitting OK into two states, OK0 and OK1, that "remember" previous A input:

		A B				
Meaning	S	00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK0	OK0	A1	A1	0
Got a 1 on A	A1	A0	A0	OK1	OK1	0
Two equal, A=0 last	OK0					1
Two equal, A=1 last	OK1					1
		S*				

State-Table Design Example

- All of next states for OK0 and OK1 can be selected from existing states:

		A B				
Meaning	S	00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK0	OK0	A1	A1	0
Got a 1 on A	A1	A0	A0	OK1	OK1	0
Two equal, A=0 last	OK0	OK0	OK0	OK1	A1	1
Two equal, A=1 last	OK1					1
		S*				

		A B				
Meaning	S	00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK0	OK0	A1	A1	0
Got a 1 on A	A1	A0	A0	OK1	OK1	0
Two equal, A=0 last	OK0	OK0	OK0	OK1	A1	1
Two equal, A=1 last	OK1	A0	OK0	OK1	OK1	1
		S*				

- We have achieved "**closure**" of state table, which now describes a **finite**-state machine.

State Minimization

- We designed a “minimal” state table for our original word description in previous section, in sense that it contains fewest possible states.
- However, following figure shows other state tables, with more states, that also do the job:

(a)

Meaning	S	A B 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK00	OK00	A1	A1	0
Got a 1 on A	A1	A0	A0	OK11	OK11	0
Got 00 on A	OK00	OK00	OK00	OKA1	A1	1
Got 11 on A	OK11	A0	OKA0	OK11	OK11	1
OK, got a 0 on A	OKA0	OK00	OK00	OKA1	A1	1
OK, got a 1 on A	OKA1	A0	OKA0	OK11	OK11	1
		S*				

(b)

Meaning	S	A B 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK00	OK00	A1	A1	0
Got a 1 on A	A1	A0	A0	OK11	OK11	0
Got 00 on A	OK00	OK00	OK00	A001	A1	1
Got 11 on A	OK11	A0	A110	OK11	OK11	1
Got 001 on A, B=1	A001	A0	AE10	OK11	OK11	1
Got 110 on A, B=1	A110	OK00	OK00	AE01	A1	1
Got bb...10 on A, B=1	AE10	OK00	OK00	AE01	A1	1
Got bb...01 on A, B=1	AE01	A0	AE10	OK11	OK11	1
		S*				

- A pair of **equivalent states** can be replaced by a single state.

State Minimization

- **Two states S1 and S2 are equivalent if** two conditions are true:

  1) S1 and S2 must produce **same values** at state-machine **output(s)**; in a Mealy machine, this must be true for all input combinations,

  2) For each input combination, S1 and S2 must have either **same next state or equivalent next states**.

- In previous figure (a), states OK00 and OKA0 are equivalent because they produce same output and their next-state entries are identical. Since states are equivalent, state OK00 may be eliminated and its occurrences in table replaced by OKA0, or vice versa.
- Likewise, states OK11 and OKA1 are equivalent.

State Minimization

- In following figure, **states OK00, A110, and AE10** all produce same output and have almost identical next-state entries, so they might be equivalent. They are **equivalent only if A001 and AE01 are equivalent**.
- Similarly, **OK11, A001, and AE01 are equivalent only if A110 and AE10 are equivalent**.
- In other words, states in first set are equivalent if states in second set are, and vice versa. So, **they are equivalent**.

		A B				
Meaning	S	00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK00	OK00	A1	A1	0
Got a 1 on A	A1	A0	A0	OK11	OK11	0
Got 00 on A	OK00	OK00	OK00	A001	A1	1
Got 11 on A	OK11	A0	A110	OK11	OK11	1
Got 001 on A, B=1	A001	A0	AE10	OK11	OK11	1
Got 110 on A, B=1	A110	OK00	OK00	AE01	A1	1
Got bb...10 on A, B=1	AE10	OK00	OK00	AE01	A1	1
Got bb...01 on A, B=1	AE01	A0	AE10	OK11	OK11	1
		S*				

State Assignment

- Next step in design process is to determine **how many binary variables are required** to represent states in state table, and to **assign a specific combination to each named state**.
- **Total number of states** in a machine with *n* flip-flops is 2^n, so the number of flip-flops needed to code *s* states is $\lceil \log_2 s \rceil$.
- State/output table of our example machine:

	AB				
S	00	01	11	10	**Z**
INIT	A0	A0	A1	A1	0
A0	OK0	OK0	A1	A1	0
A1	A0	A0	OK1	OK1	0
OK0	OK0	OK0	OK1	A1	1
OK1	A0	OK0	OK1	OK1	1
	S*				

State Assignment

- Above table has five states, so it requires three flip-flops. There will be 8 – 5 = 3 **unused states**.
- **Simplest assignment** of *s* coded states to 2^n possible states is to **use first *s* binary integers in binary counting order**, as shown in first assignment column of following table:

State name	Assignment			
	Simplest Q1-Q3	**Decomposed Q1-Q3**	**One-hot Q1-Q5**	**Almost one-hot Q1-Q4**
INIT	000	000	00001	0000
A0	001	100	00010	0001
A1	010	101	00100	0010
OK0	011	110	01000	0100
OK1	100	111	10000	1000

State Assignment

- Simplest state assignment does not always lead to simplest excitation equations, output equations, and resulting logic circuit.
- **State assignment** often has a **major effect on circuit cost**, and may interact with other factors, such as choice of storage elements (e.g., D vs. J-K flip-flops) and realization approach for excitation and output logic (e.g., SOPs, POSs, or ad hoc).
- In general, **only formal way to find best assignment is to try all** assignments. That is too much work.
- Several **practical guidelines** for making reasonable state assignments:

  - Choose an initial coded state into which machine can easily be forced at reset (00… 00 or 11… 11 in typical circuits).

  - Minimize number of state variables that change on each transition.

  - Maximize number of state variables that don’t change in a group of related states (i.e., a group of states in which most of transitions stay in the group).

State Assignment

- Exploit **symmetries** in problem specification and corresponding symmetries in state table. That is, suppose that one state or group of states means almost the same thing as another. Once an assignment has been established for the first, a similar assignment, differing only in one bit, should be used for the second.

- If there are **unused states**, then choose best of available state-variable combinations to achieve foregoing goals. That is, don't limit choice of coded states to first *s* n-bit integers.

- **Decompose** set of state variables into individual bits or fields where each bit or field has a well-defined meaning with respect to input effects or output behavior of machine.

- Consider using more than minimum number of state variables to make a decomposed assignment possible.

State Assignment

- Some of above ideas are incorporated in "**decomposed**" state assignment in previous table:

  - Initial state is 000, which is easy to force either asynchronously (applying RESET signal to flip-flop CLR inputs) or synchronously (by ANDing RESET' with all of D flip-flop inputs).

  - Assignment takes advantage of fact that there are only four states in addition to INIT, which is a fairly special state that is never re-entered once the machine gets going. Therefore, Q1 can be used to indicate whether or not machine is in INIT state, and Q2 and Q3 can be used to distinguish among four non-INIT states.

  - Q3 gives previous value of A, and Q2 indicates that conditions for a 1 output are satisfied in current state. By decomposing state-bit meanings in this way, we can expect next-state and output logic to be simpler than in a random assignment of Q2,Q3 combinations.

State Assignment

- **One-hot assignment** can be adapted to any state machine.
- This assignment **uses more than minimum number of state variables** – it uses one bit per state.
- A one-hot assignment is simple, and usually leads to **small excitation equations**, since **only one flip-flop must be set to 1 for transitions** into each state.
- A disadvantage of a one-hot assignment, especially for machines with many states, is that it requires (many) more than minimum number of flip-flops.
- One-hot encoding is **ideal** for a machine with *s* states that is required to have a set of **1-out-of-*s* coded outputs** indicating its current state. The one-hot-coded flip-flop outputs can be used directly for this purpose, with no additional combinational output logic.

State Assignment

- Last column of previous table is an "**almost one-hot assignment**" that uses "no-hot" combination for initial state. This makes sense for two reasons:

  - It is **easy to initialize most storage devices to all-0s state**,

  - **Initial state** in this machine is **never revisited** once machine gets going.

State Assignment

- Considering **unused states**, there are two approaches that make sense, depending on application requirements:

  Minimal risk: This approach assumes that it is possible for state machine to get into one of unused (illegal) states, perhaps because of a hardware failure, an unexpected input, or a design error. Therefore, all of unused state-variable combinations are identified, and explicit next-state entries are made so that, for any input combination, unused states go to “initial” state, “idle” state, or some other “safe” state.

  Minimal cost: This approach assumes that machine will never enter an unused state. Therefore, in transition and excitation tables, next-state entries of unused states can be marked as “don’t-cares”. In most cases, this simplifies excitation logic. However, machine’s behavior if it ever does enter an unused state may be pretty weird.

Synthesis Using D Flip-Flops

- Coded states are substituted for named states in state table to obtain a **transition table**. Transition table shows next coded state for each combination of current coded state and input.
- Following table (right) shows transition and output table that is obtained from example state machine (left) using **decomposed assignment** we had before:

	AB				
S	00	01	11	10	**Z**
INIT	A0	A0	A1	A1	0
A0	OK0	OK0	A1	A1	0
A1	A0	A0	OK1	OK1	0
OK0	OK0	OK0	OK1	A1	1
OK1	A0	OK0	OK1	OK1	1
	S*				

	AB				
Q1Q2Q3	00	01	11	10	**Z**
000	100	100	101	101	0
100	110	110	101	101	0
101	100	100	111	111	0
110	110	110	111	101	1
111	100	110	111	111	1
	Q1*Q2*Q3*				

Synthesis Using D Flip-Flops

- Next step is to write an **excitation table** that shows, for each combination of coded state and input, flip-flop excitation input values needed to make the machine go to desired next coded state.
- Structure and content of this table depend on type of flip-flops that are used (D, J-K, T, etc.).
- Each D flip-flop in a state machine has a single excitation input, D, and excitation table must show value required at each flip-flop's D input for each coded-state/input combination. Excitation table for our example problem:

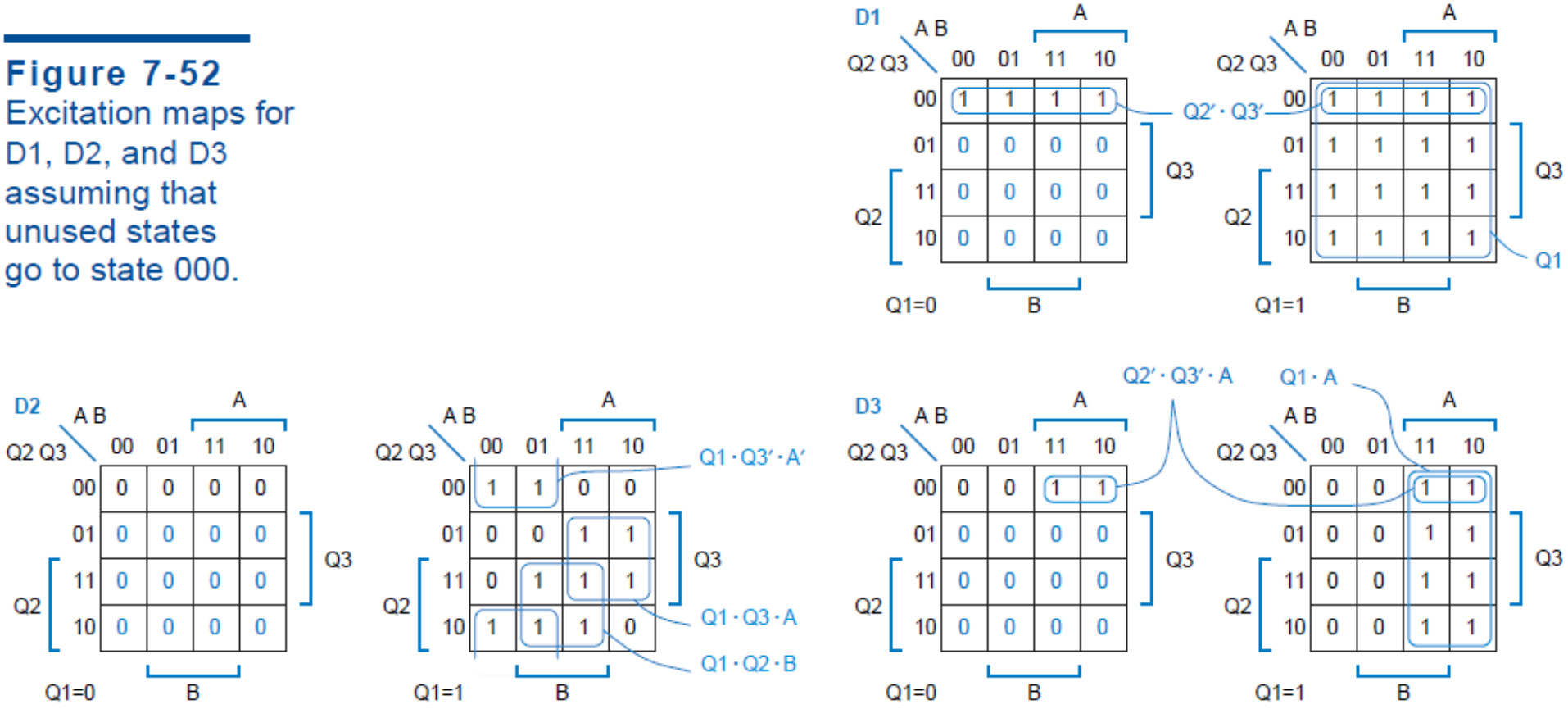
	AB				
Q1Q2Q3	00	01	11	10	**Z**
000	100	100	101	101	0
100	110	110	101	101	0
101	100	100	111	111	0
110	110	110	111	101	1
111	100	110	111	111	1
	D1D2D3				

Synthesis Using D Flip-Flops

- Since **D = Q***, **excitation table is identical to transition table**, except for labeling of its entries. We call the first table a **transition/excitation table**.
- Excitation table is like a truth table for three combinational logic functions (D1, D2, D3) of five variables (A, B, Q1, Q2, Q3).
- We can transfer information in excitation table to **Karnaugh maps** (called **excitation maps**) and find a minimal SOPs or POSs expression for each function.
- Excitation maps for our example state machine are shown below:

Synthesis Using D Flip-Flops

Figure 7-52
Excitation maps for D1, D2, and D3 assuming that unused states go to state 000.



	AB				
Q1Q2Q3	00	01	11	10	**Z**
000	100	100	101	101	0
100	110	110	101	101	0
101	100	100	111	111	0
110	110	110	111	101	1
111	100	110	111	111	1
	D1D2D3				

Synthesis Using D Flip-Flops

- Each function, such as D1, has five variables and therefore uses a 5-variable Karnaugh map. A 5-variable map is drawn as a pair of 4-variable maps.
- In excitation table, next-state information for unused states, 001, 010, and 011, is not specified. Here we must make a choice between a minimal-risk and a minimal-cost strategy for handling unused states.
- Previous figure has taken **minimal-risk** approach: next state for each unused state and input combination is 000, the INIT state. Three rows of colored 0s in each Karnaugh map are result of this choice.
- Minimal SOPs expressions for flip-flop excitation inputs:

  D1 = Q1 + Q2'.Q3'

  D2 = Q1.Q3'.A' + Q1.Q3.A + Q1.Q2.B

  D3 = Q1.A + Q2'.Q3'.A
- Output equation is a function of state only and can easily be developed:

  Z = Q1.Q2.Q3' + Q1.Q2.Q3 = Q1.Q2

Synthesis Using D Flip-Flops

- If we choose in our example to derive **minimal-cost** excitation equations, we write "don't-cares" in next-state entries for unused states. Colored d's in next figure are result of this choice.
- Excitation equations obtained from this map:

  D1 = 1

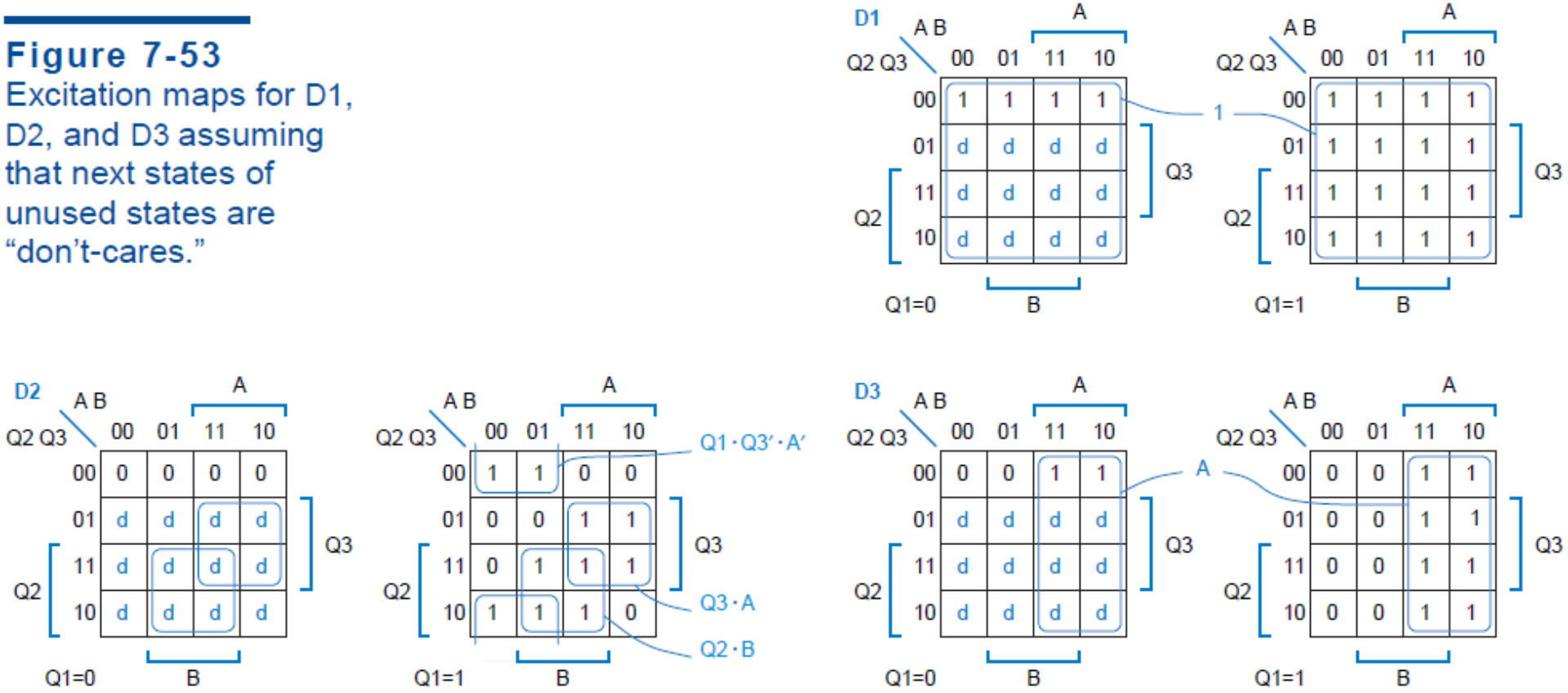
  D2 = Q1.Q3'.A' + Q3.A + Q2.B

  D3 = A
- For a minimal-cost output function, value of Z is a "don't-care" for unused states. Therefore:

  Z = Q2

Synthesis Using D Flip-Flops

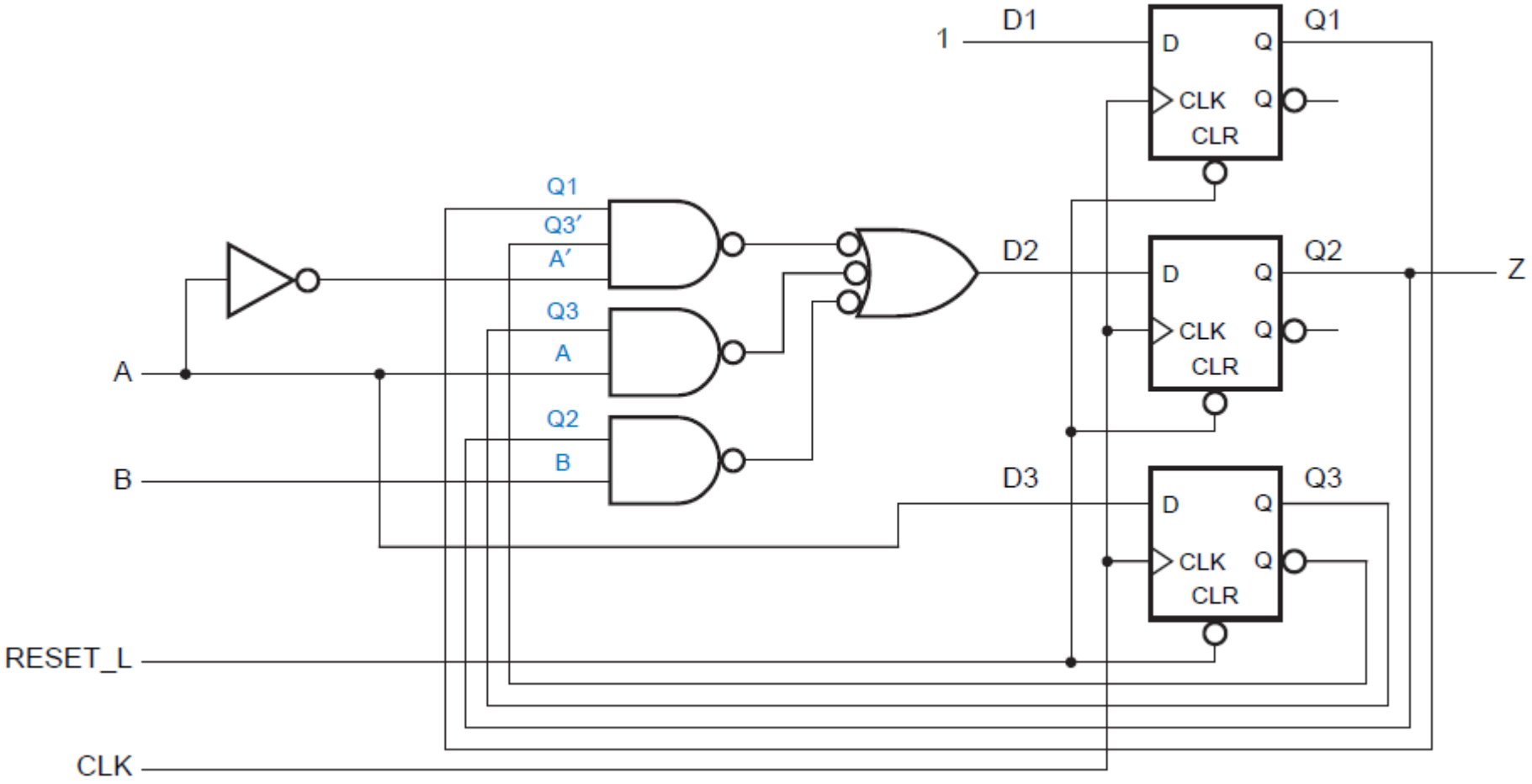
Figure 7-53
Excitation maps for D1, D2, and D3 assuming that next states of unused states are "don't-cares."



	AB				
Q1Q2Q3	00	01	11	10	**Z**
000	100	100	101	101	0
100	110	110	101	101	0
101	100	100	111	111	0
110	110	110	111	101	1
111	100	110	111	111	1
	D1D2D3				

Synthesis Using D Flip-Flops

- Logic diagram for **minimal-cost** solution:



State-Machine Design Using D Flip-Flops: Example 1

- "1s-counting machine":

  Design a clocked synchronous state machine with two inputs, X and Y, and one output, Z. The output should be 1 if the number of 1 inputs on X and Y since reset is a multiple of 4, and 0 otherwise.

State-Machine Design Using D Flip-Flops: Example 1

- "1s-counting machine":

  Design a clocked synchronous state machine with two inputs, X and Y, and one output, Z. The output should be 1 if the number of 1 inputs on X and Y since reset is a multiple of 4, and 0 otherwise.

- State and output table:

Meaning	S	XY: 00	01	11	10	Z
Got zero 1s (modulo 4)	S0	S0	S1	S2	S1	1
Got one 1 (modulo 4)	S1	S1	S2	S3	S2	0
Got two 1s (modulo 4)	S2	S2	S3	S0	S3	0
Got three 1s (modulo 4)	S3	S3	S0	S1	S0	0
		S*				

- Since output indicates number of inputs received modulo 4, four states are sufficient. S0 is initial state and total number of 1s received in S*i* is *i* modulo 4.

State-Machine Design Using D Flip-Flops: Example 1

- 1s-counting machine can use **two state variables** to code its four states, with **no unused states**. In this case, there are 4! possible assignments of coded states to named states.
- We will **assign coded states** to named states in **Karnaugh-map order** (00, 01, 11, 10) for two reasons:

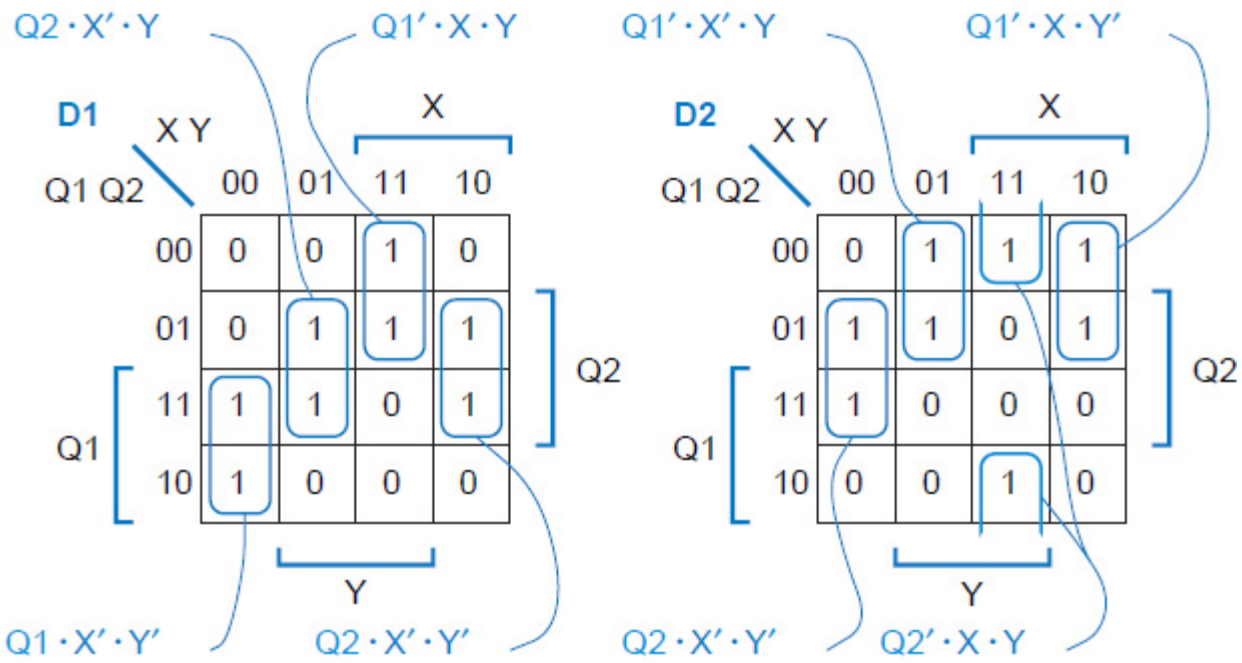
  - It minimizes number of state variables that change for most transitions, potentially simplifying excitation equations;

  - It simplifies mechanical transfer of information to excitation maps.
- Transition/excitation and output table:

	XY				
Q1Q2	00	01	11	10	**Z**
00	00	01	11	01	1
01	01	11	10	11	0
11	11	10	00	10	0
10	10	00	01	00	0
	Q1*Q2* or D1D2				

State-Machine Design Using D Flip-Flops: Example 1

- Corresponding Karnaugh maps for D1 and D2:



- Since there are no unused states, no choice is required between minimal-risk and minimal-cost approaches.

State-Machine Design Using D Flip-Flops: Example 1

- Excitation equations can be read from maps, and output equation can be read directly from transition/excitation table:

  D1 = Q2.X'.Y + Q1'.X.Y + Q1.X'.Y' + Q2.X.Y'

  D2 = Q1'.X'.Y + Q1'.X.Y' + Q2.X'.Y' + Q2'.X.Y

  Z = Q1'.Q2'

- A logic diagram using D flip-flops and AND-OR or NAND-NAND excitation logic can be drawn from these equations.

State-Machine Design Using D Flip-Flops: Example 2

- A “Combination lock” state machine that activates an “unlock” output when a certain binary input sequence is received:

  Design a clocked synchronous state machine with one input, X, and two outputs, UNLK and HINT. The UNLK output should be 1 if and only if X is 0 and the sequence of inputs received on X at the preceding seven clock ticks was 0110111. The HINT output should be 1 if and only if the current value of X is the correct one to move the machine closer to being in the “unlocked” state (with UNLK=1).

State-Machine Design Using D Flip-Flops: Example 2

- This is a **Mealy machine**. UNLK output depends on both past history of inputs and X's current value, and HINT depends on both state and current X (if current X produces HINT=0, then clued-in user will want to change X before clock tick).
- State and output table for combination-lock machine:

Meaning	S	X	
		0	1
Got zip	A	B, 01	A, 00
Got 0	B	B, 00	C, 01
Got 01	C	B, 00	D, 01
Got 011	D	E, 01	A, 00
Got 0110	E	B, 00	F, 01
Got 01101	F	B, 00	G, 01
Got 011011	G	E, 00	H, 01
Got 0110111	H	B, 11	A, 00
		S*, UNLK HINT	

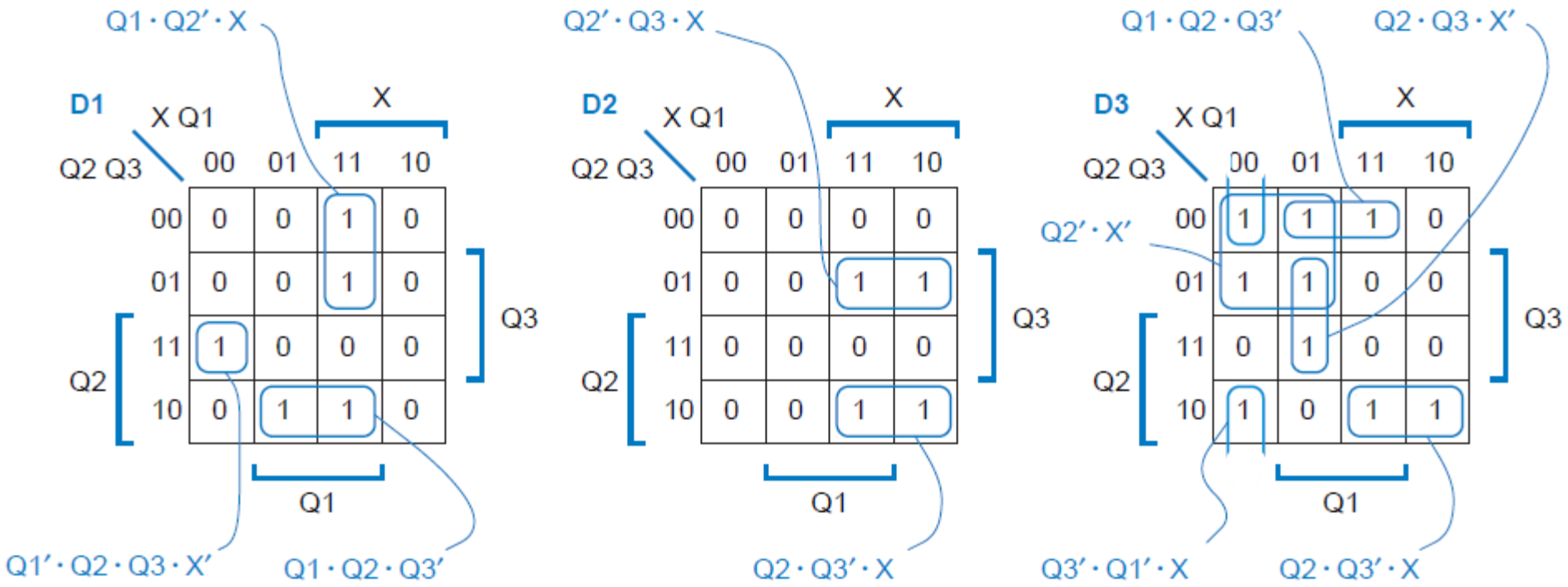
State-Machine Design Using D Flip-Flops: Example 2

- Combination lock's eight states can be coded with **three state variables**, leaving **no unused states**.
- There are 8! state assignments to choose from. To keep things simple, we will use the **simplest**, and assign states in **binary counting order**.
- Transition/excitation table for combination-lock machine:

	X	
Q1Q2Q3	0	1
000	001, 01	000, 00
001	001, 00	010, 01
010	001, 00	011, 01
011	100, 01	000, 00
100	001, 00	101, 01
101	001, 00	110, 01
110	100, 00	111, 01
111	001, 11	000, 00
	Q1*Q2*Q3*, UNLK HINT	

State-Machine Design Using D Flip-Flops: Example 2

- Corresponding Karnaugh maps for D1, D2, and D3:



- Excitation equations can be read from maps:

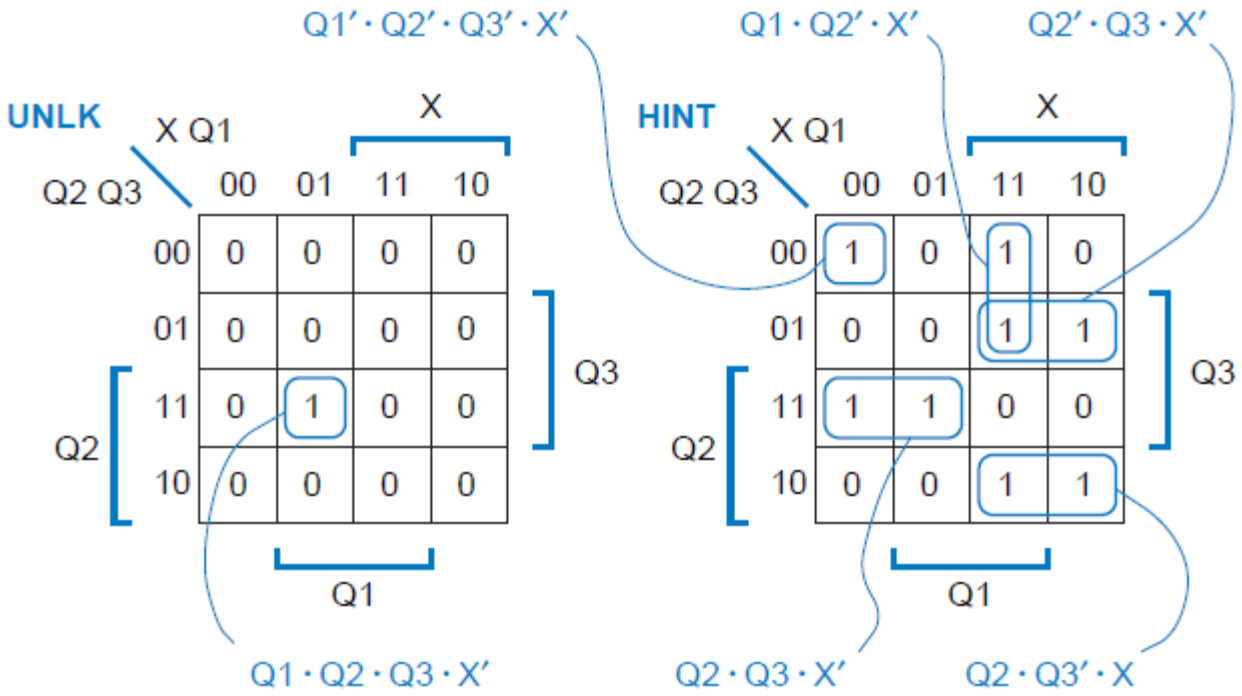
  D1 = Q1.Q2’.X + Q1’.Q2.Q3.X’ + Q1.Q2.Q3’

  D2 = Q2’.Q3.X + Q2.Q3’.X

  D3 = Q1.Q2’.Q3’ + Q1.Q3.X’ + Q2’.X’ + Q3’.Q1’.X’ + Q2.Q3’.X

State-Machine Design Using D Flip-Flops: Example 2

- Output values are transferred from transition/excitation and output table to another set of maps:



- Corresponding output equations are:

  UNLK = Q1.Q2.Q3.X’

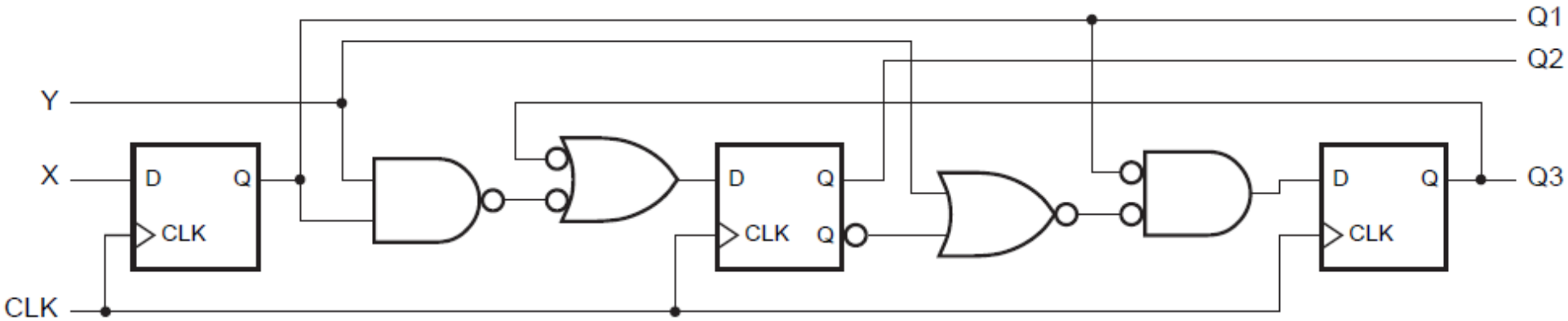
  HINT = Q1’.Q2’.Q3’.X’ + Q1.Q2’.X + Q2’.Q3.X + Q2.Q3.X’ + Q2.Q3’.X

State-Machine Design Using D Flip-Flops: Example 2

- Some product terms are repeated in excitation and output equations, yielding a slight savings in cost of AND-OR realization.

Exercises

- 1) Analyze the clocked synchronous state machine in the following figure. Write excitation equations, excitation/transition table, and state/output table (use state names A-H for Q1Q2Q3=000-111).



Exercises

- 2) Using D flip-flops, design a 3-bit counter that will count in the cyclic pattern of even numbers: 0, 2, 4, 6.
- 3) Using D flip-flops, design a BCD to Excess-3 code converter: a serially transmitted binary coded decimal (BCD word) is to be converted into an Excess-3 encoded serial bit stream (An Excess-3 code word is obtained by adding 3 (0011) to the value of the BCD word).