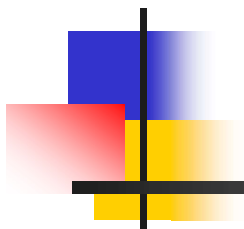


數位電路設計

Digital Circuit Design



莊仁輝

交通大學資訊工程系



Chapter 6

Registers and Counters



Outline

6.1 Registers

6.2 Shift Registers

6.3 Ripple Counters

6.4 Synchronous Counters

6.5 Other Counters

6.6 HDL Models of Registers and Counters



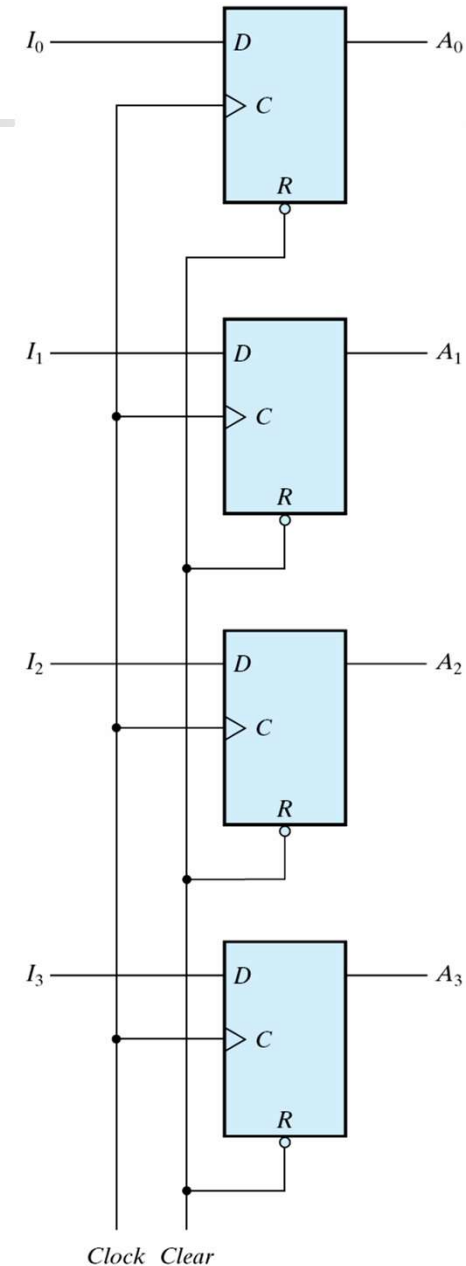
6.1 Registers

- Clocked sequential circuits
 - a group of **flip-flops** and **optional** combinational **gates** connected to form a feedback path
- Register:
 - the **flip-flops** hold the **binary** information
 - the **gates** determine how the information is transferred into the register
- Counter:
 - a **register** that goes through a **predetermined** sequence of **states**

6-1 Registers

- A n -bit register
 - n flip-flops capable of **storing** n bits of binary information

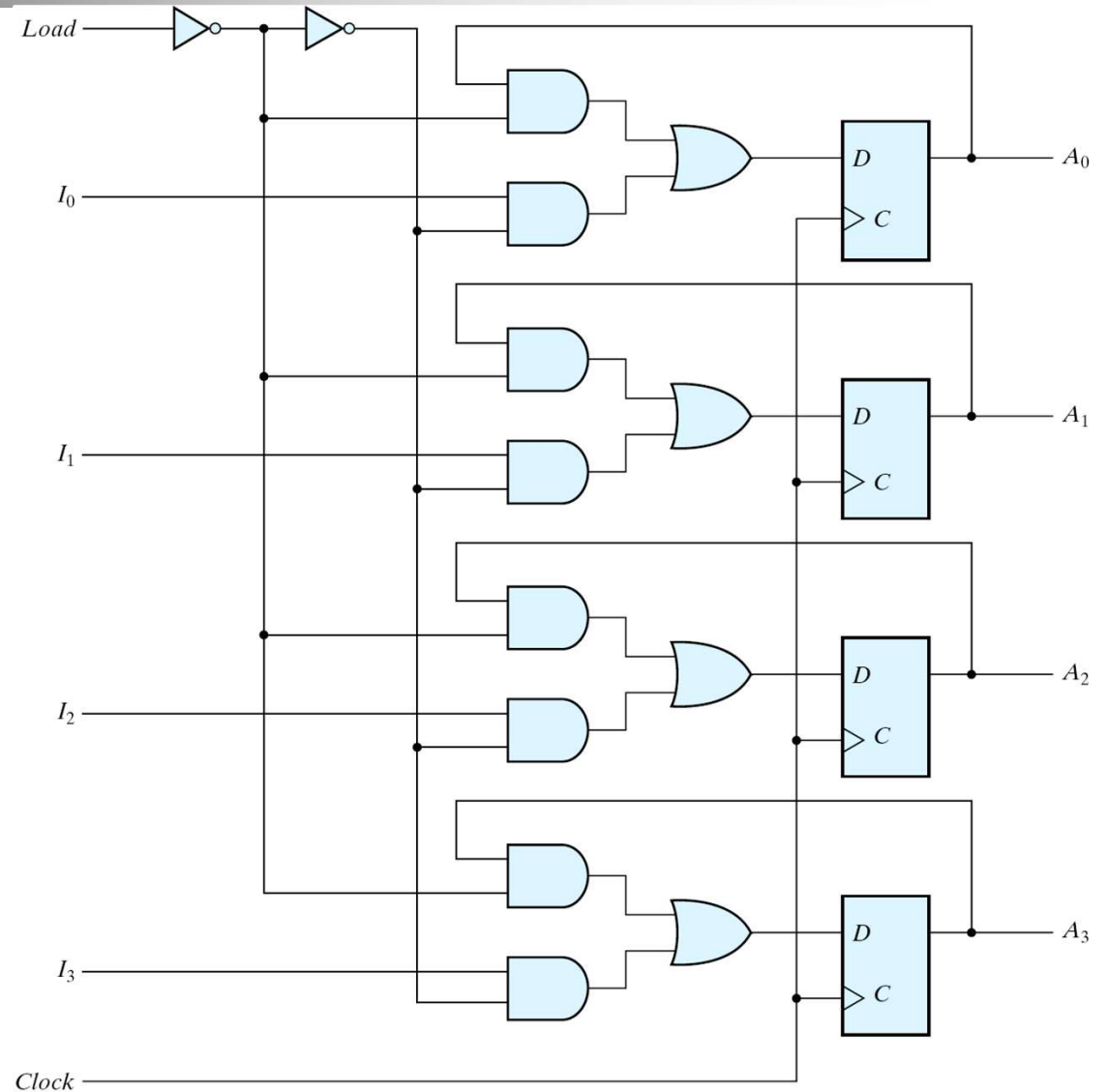
Ex. A **4-bit** register



6-1 Registers

Ex. 4-bit register
with parallel load

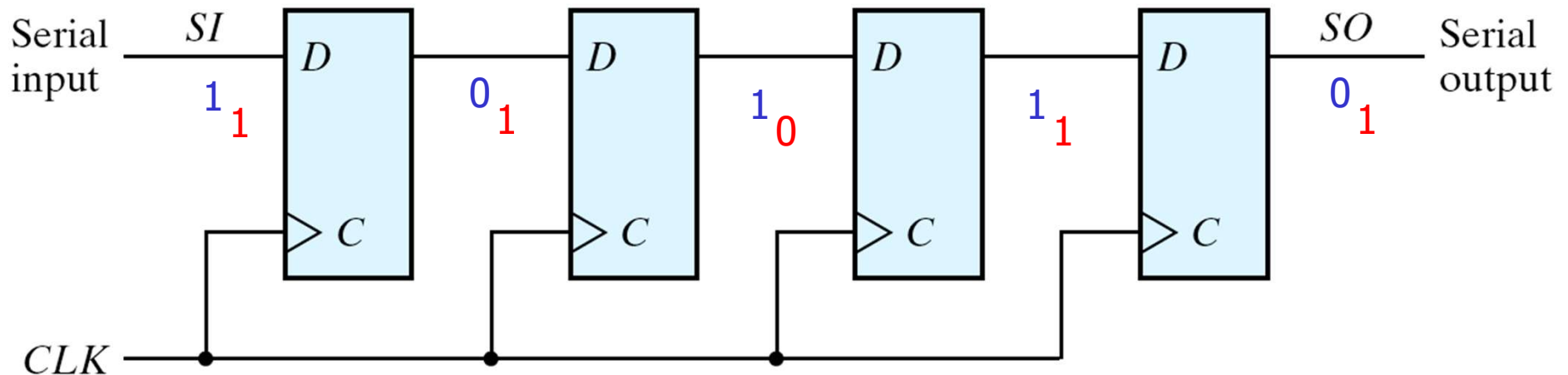
- use latches?
- setup/hold time?



6-2 Shift Registers

- Shift register
 - a register capable of **shifting** its binary information in one or both directions

Ex. a **four-bit** shift register



- use latches?
- setup/hold time?



6-2 Shift Registers

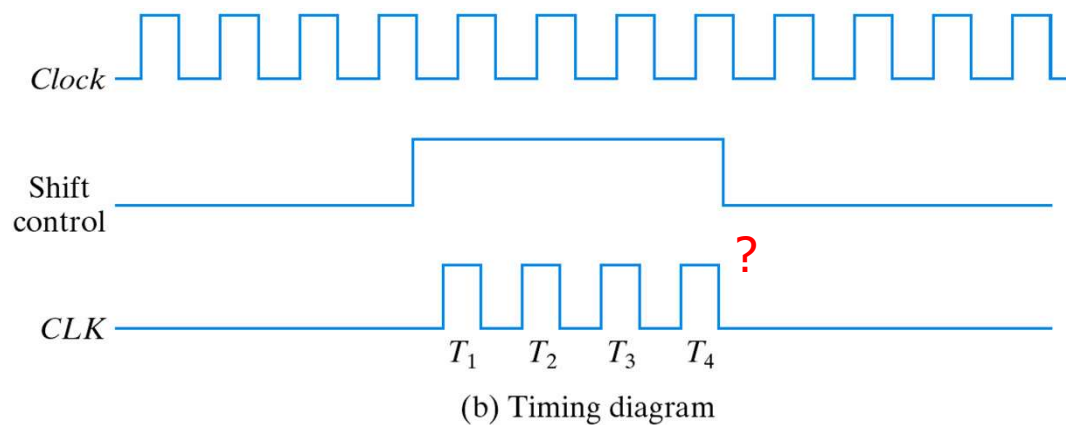
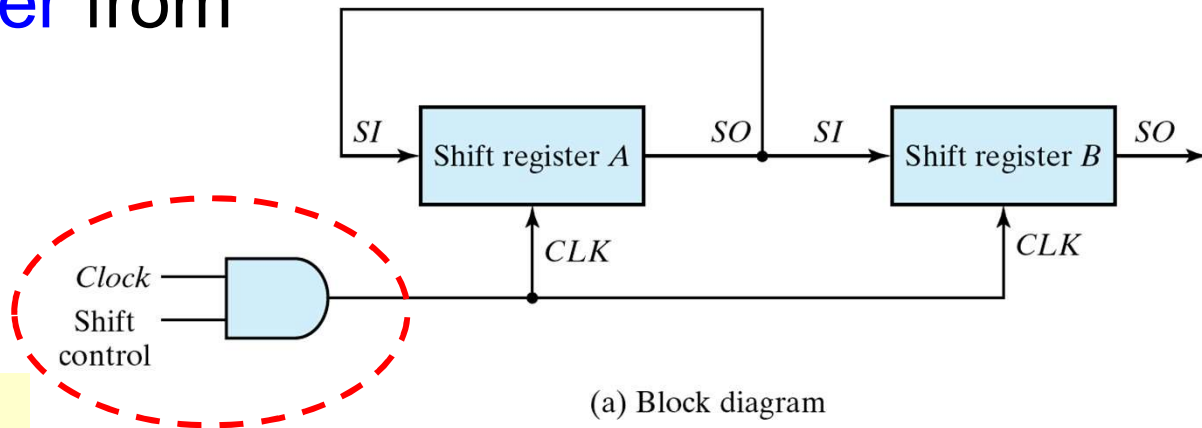
- Serial transfer vs. Parallel transfer
 - Serial transfer
 - Information is transferred **one bit at a time**
 - Data bits are shifted out of the **source** register into the **destination** register
 - Parallel transfer:
 - **All the bits** of the register are transferred at the **same time**

6-2 Shift Registers

Ex. Serial transfer from reg A to reg B

gated clock →

- low power
- bad for timing
 1. pulse width
 2. pulse delay





6-2 Shift Registers

Ex. (cont.)

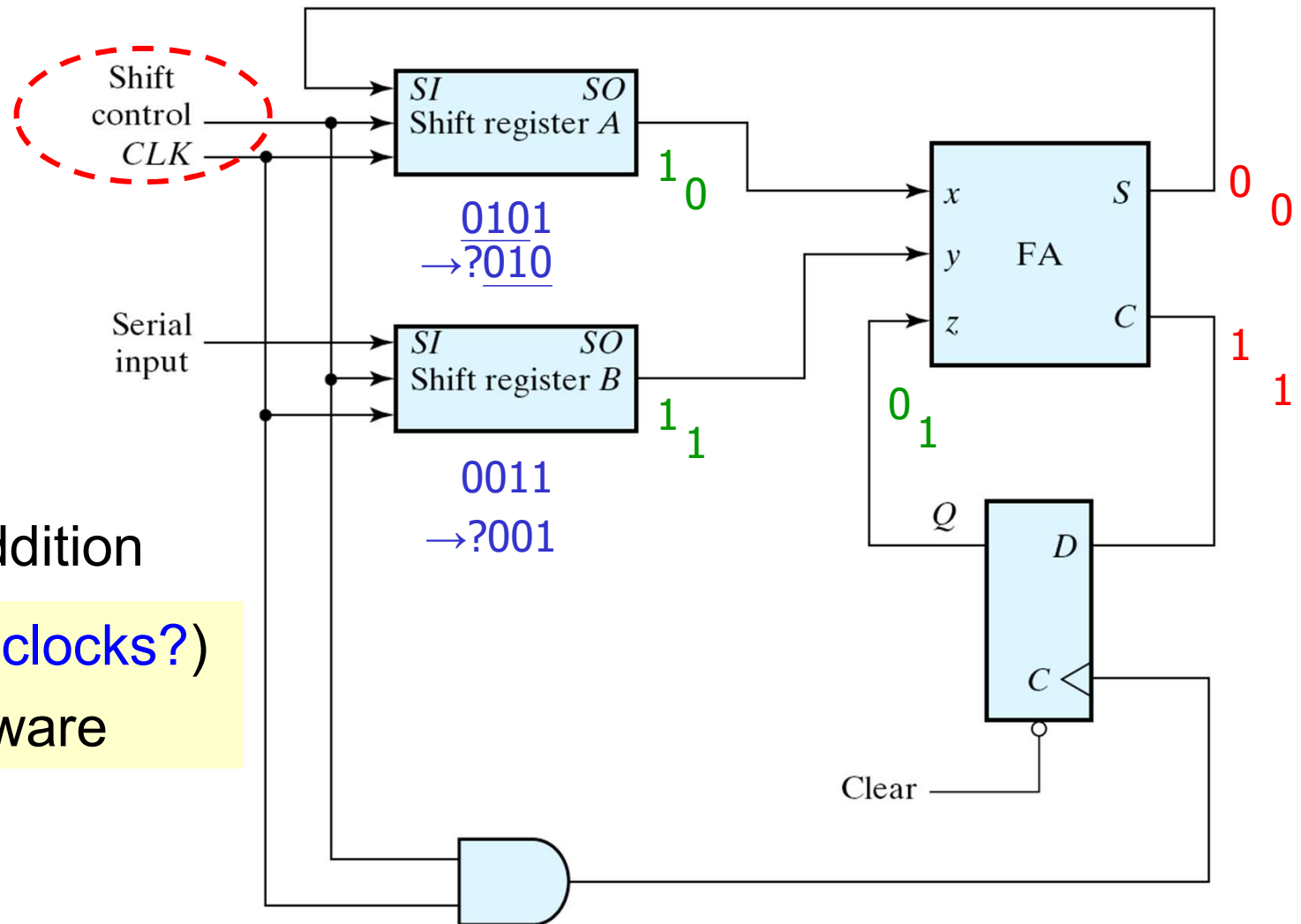
Serial-Transfer Example

Timing Pulse	Shift Register A				Shift Register B			
Initial value	1	0	1	1	0	0	1	0
After T_1	1	1	0	1	1	0	0	1
After T_2	1	1	1	0	1	1	0	0
After T_3	0	1	1	1	0	1	1	0
After T_4	1	0	1	1	1	0	1	1

6-2 Shift Registers

Ex. Serial addition

- slower (8 clocks?)
- less hardware



6-2 Shift Registers

Ex. serial adder using JK flip-flops

Table 6.2
State Table for Serial Adder

Present State	Inputs		Next State	Output	Flip-Flop Inputs	
Q	x	y	Q	S	J _Q	K _Q
0	0	0	0	0	0	X
0	0	1	0	1	0	X
0	1	0	0	1	0	X
0	1	1	1	0	1	X
1	0	0	0	1	X	1
1	0	1	1	0	X	0
1	1	0	1	0	X	0
1	1	1	1	1	X	0

$$J_Q = x y$$

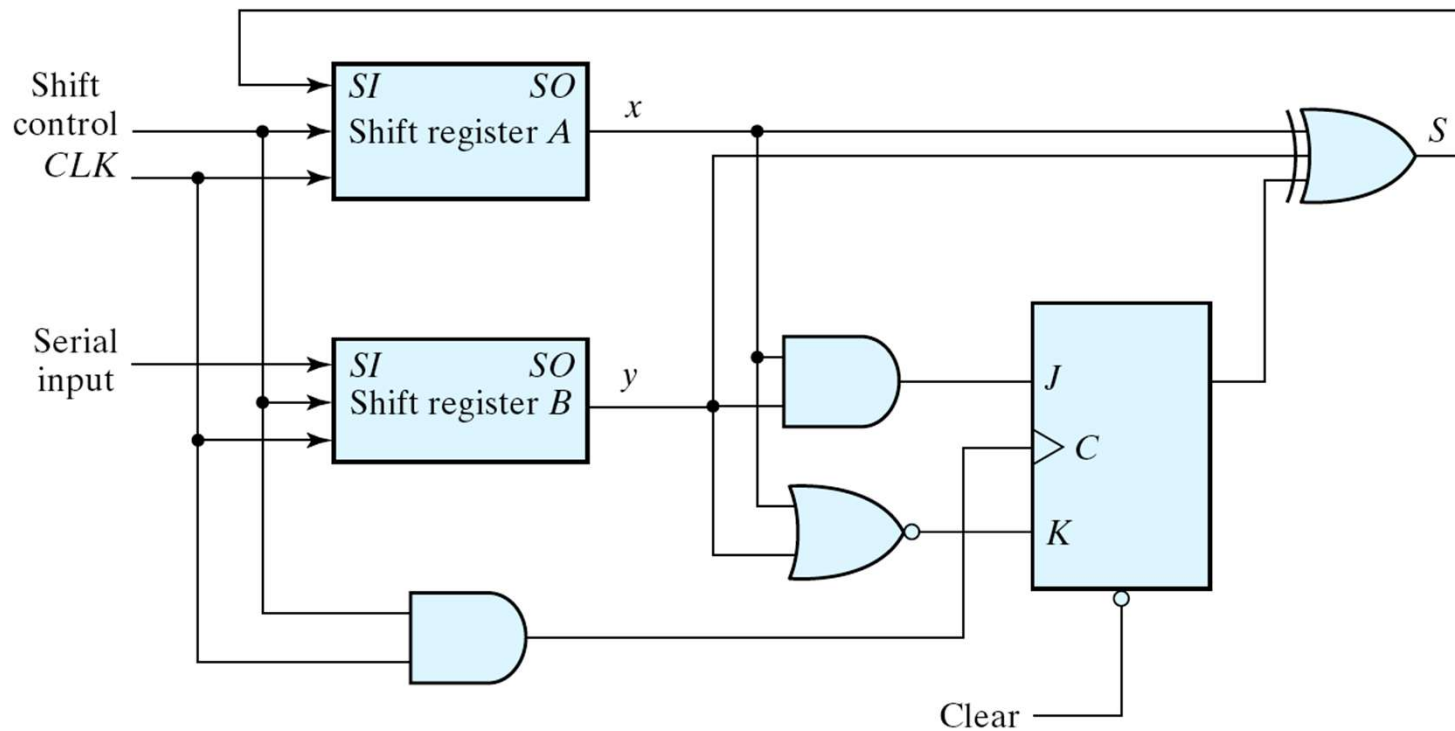
$$K_Q = x' y' = (x + y)'$$

$$S = x \oplus y \oplus Q$$

6-2 Shift Registers

Ex. (cont.)

- Circuit diagram



Subtractor?

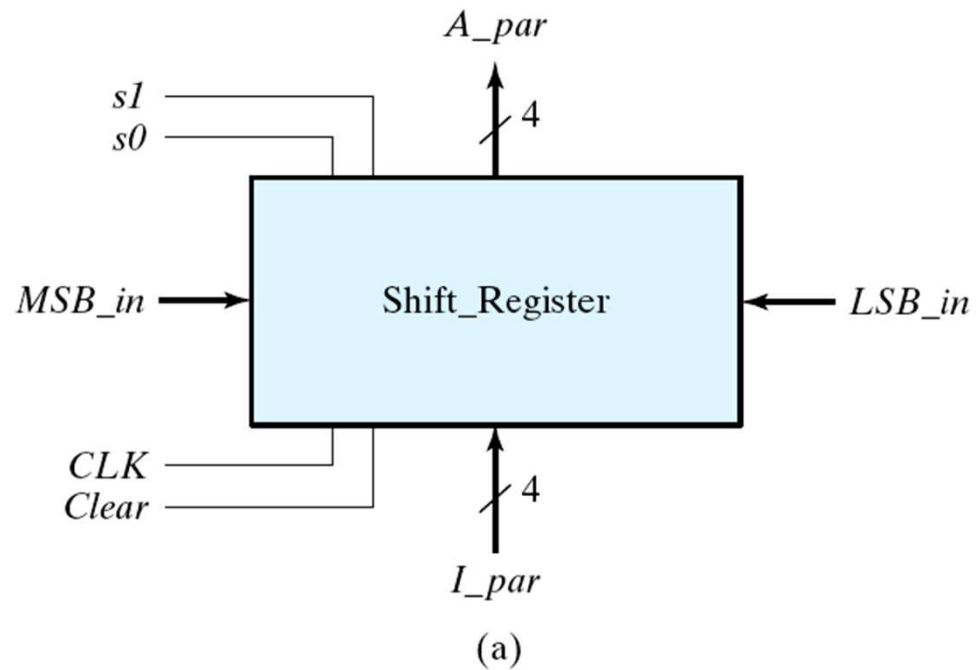


6-2 Shift Registers

- Capability of a **universal shift register**:
 1. A **clear** control to clear the register to 0.
 2. A **clock** input to synchronize the operations.
 3. A **shift-right** control to enable the shift right operation and the *serial input* and *output* lines associated w/ the shift right.
 4. A **shift-left** control to enable the shift left operation and the *serial input* and *output* lines associated w/ the shift left.
 5. A **parallel-load** control to enable a parallel transfer and the *n parallel input* lines associated w/ the parallel transfer.
 6. *n parallel output* lines.
 7. A control state that leaves the information in the register **unchanged** in the presence of the clock.

6-2 Shift Registers

Ex. 4-bit **universal** shift register

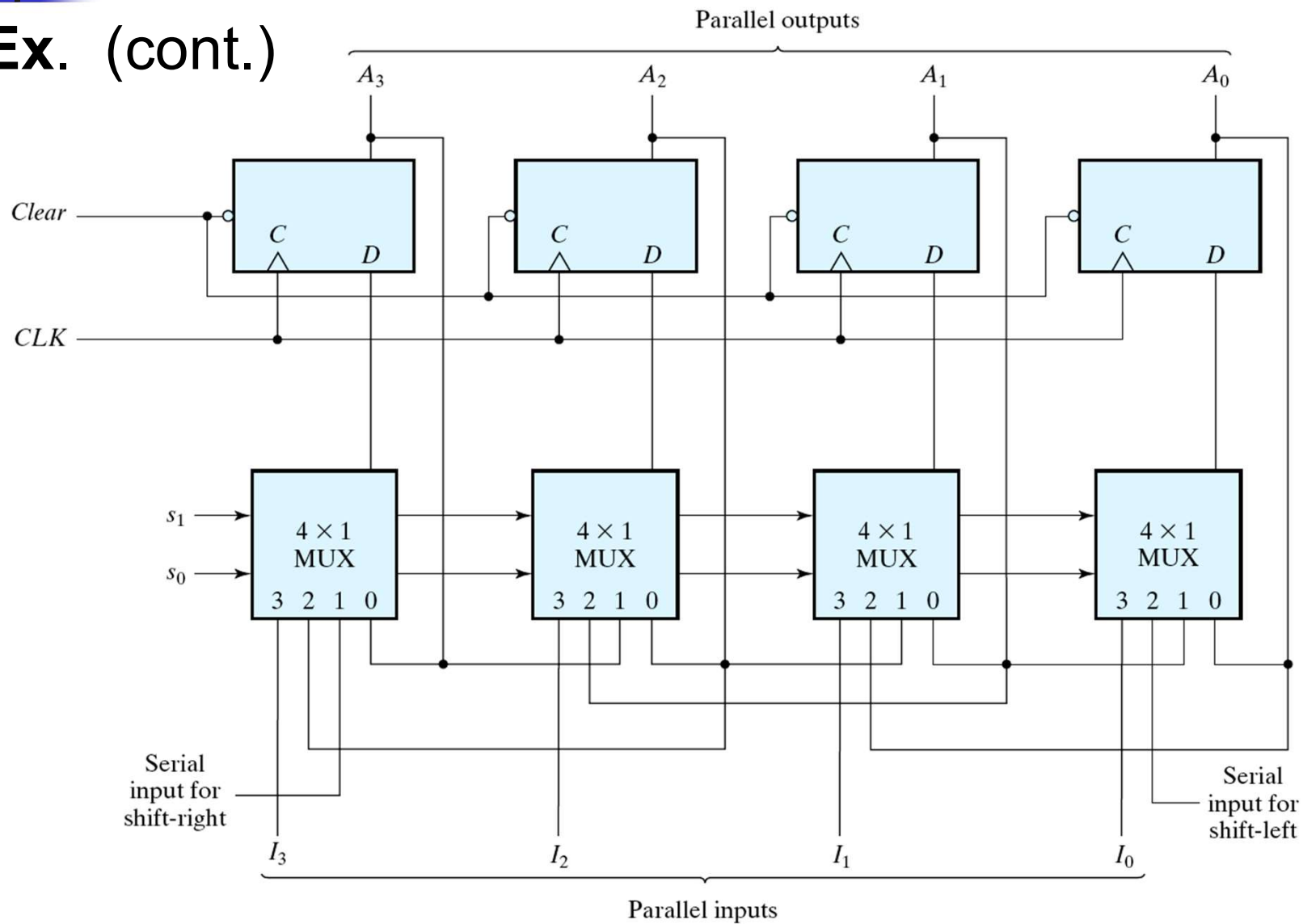


Function Table for the Register of Fig. 6.7

Mode Control		Register Operation
s_1	s_0	
0	0	No change
0	1	Shift right
1	0	Shift left
1	1	Parallel load

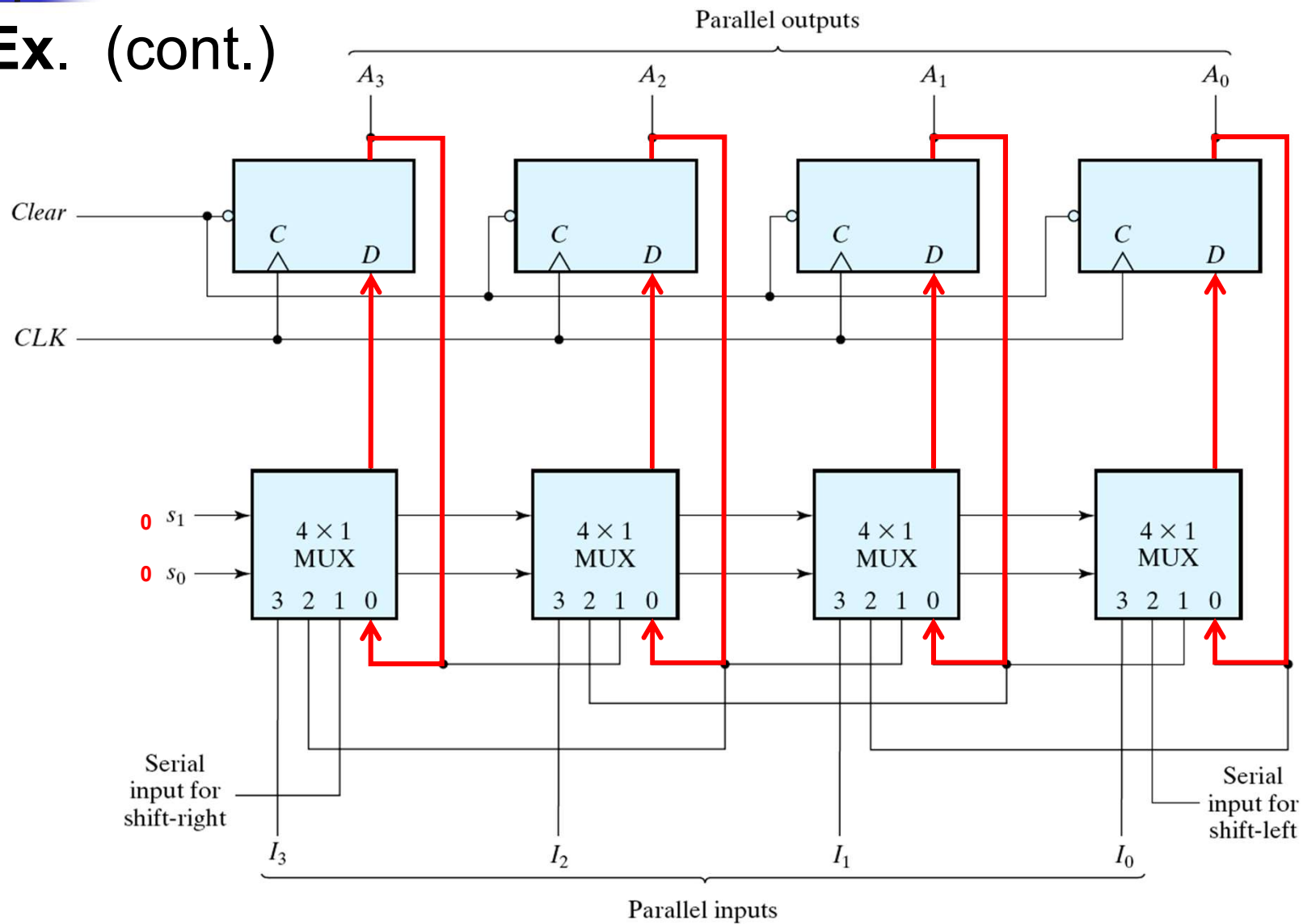
6-2 Shift Registers

Ex. (cont.)



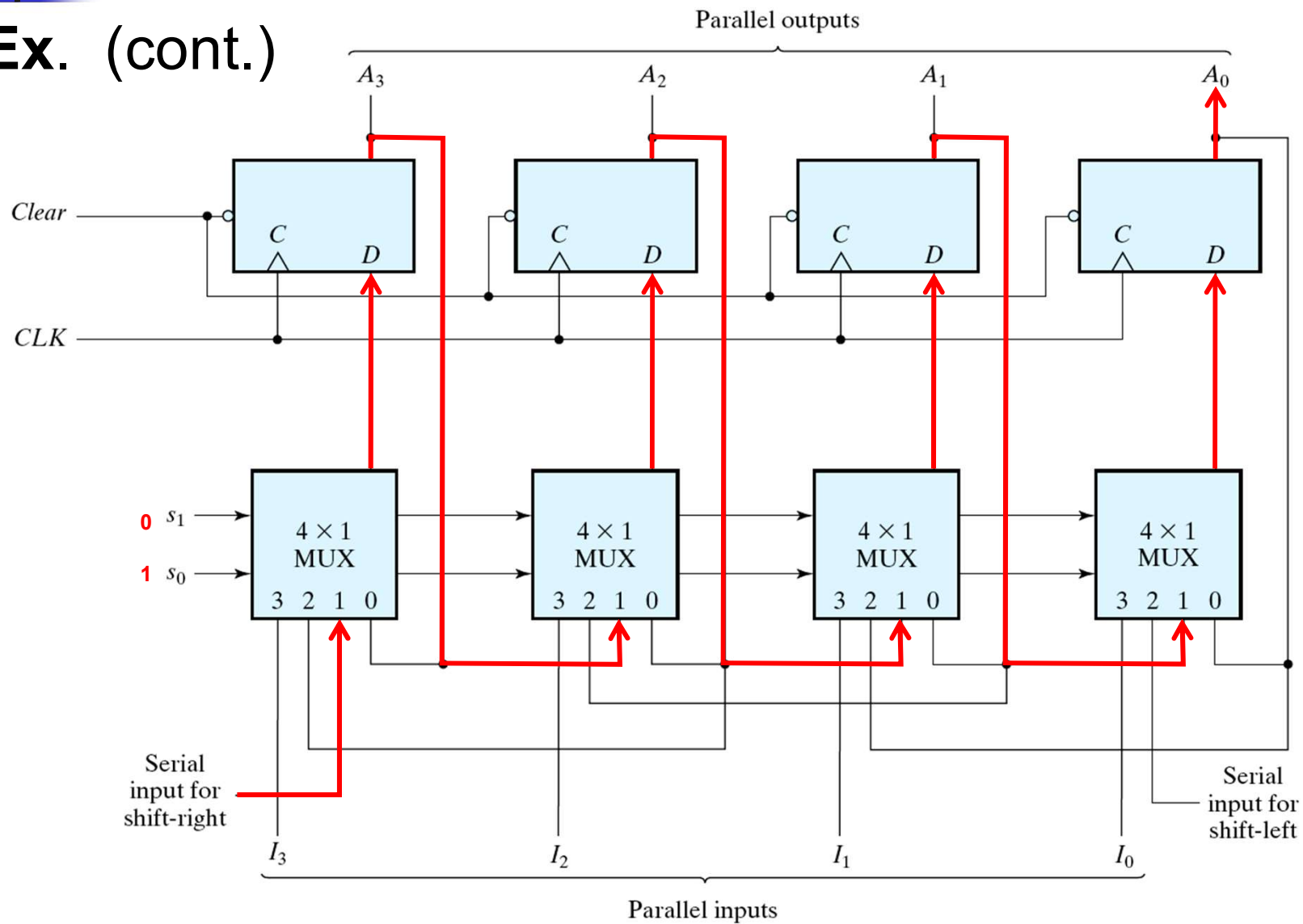
6-2 Shift Registers

Ex. (cont.)



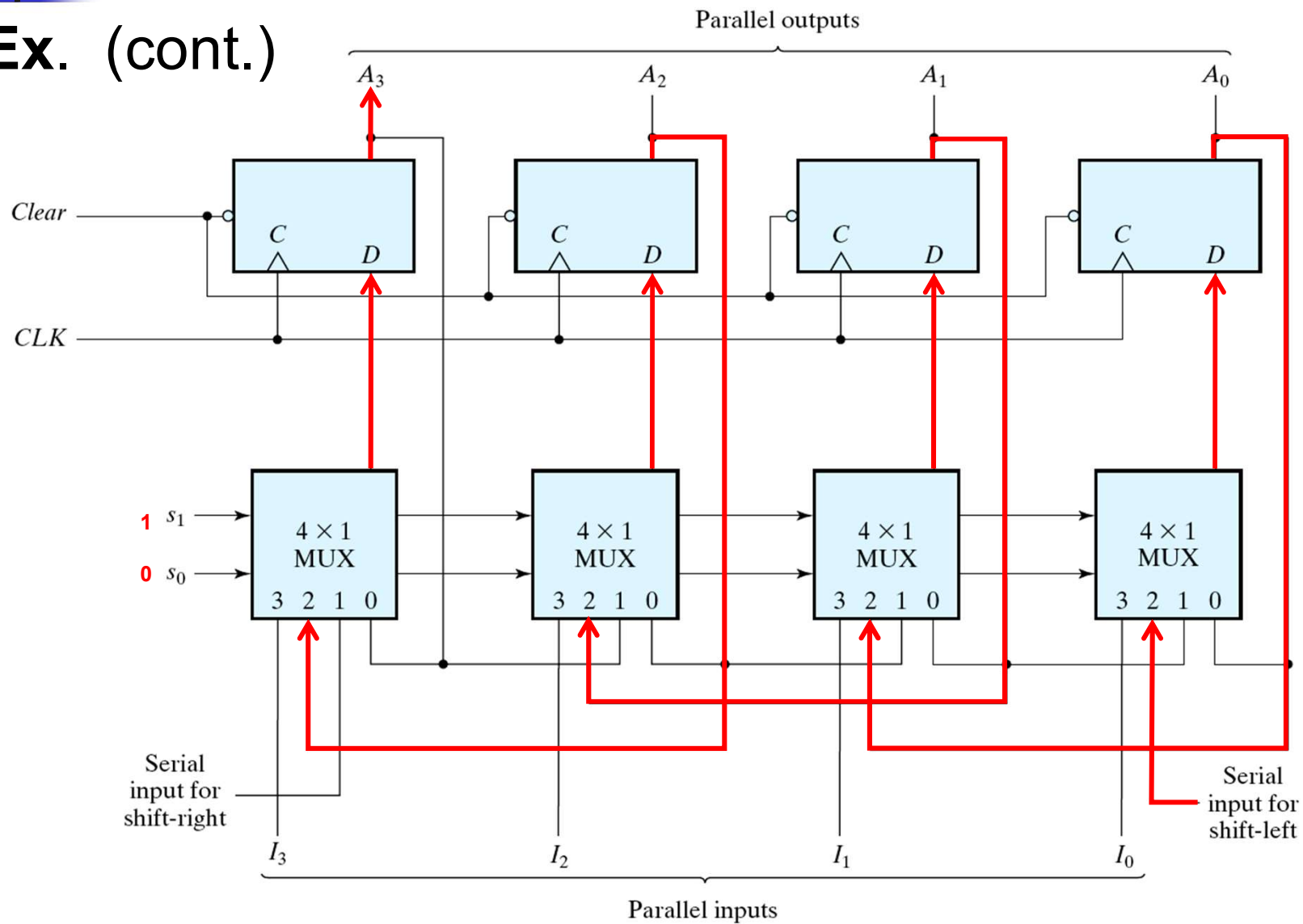
6-2 Shift Registers

Ex. (cont.)



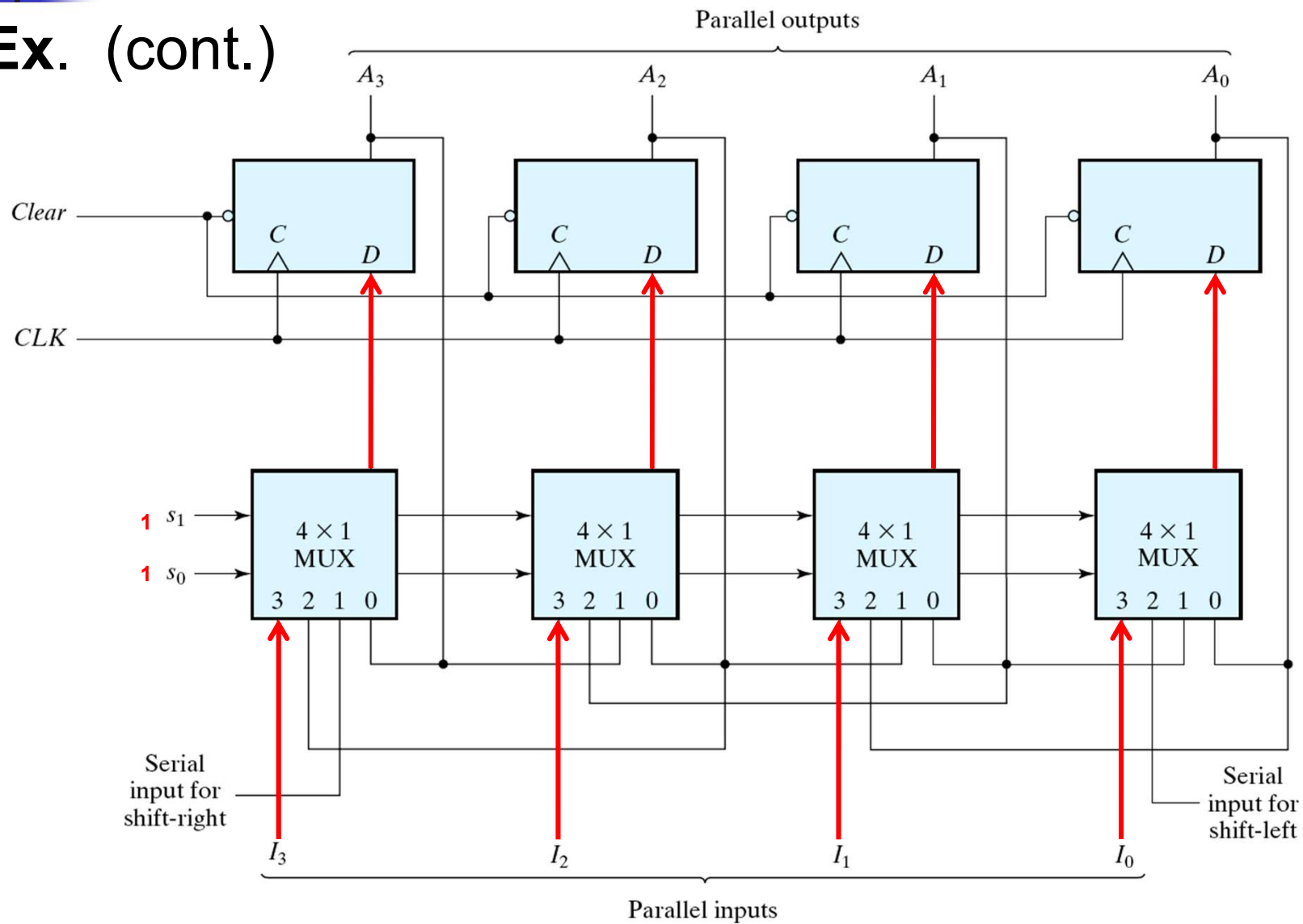
6-2 Shift Registers

Ex. (cont.)



6-2 Shift Registers

Ex. (cont.)





6-3 Ripple Counters

- Counter:
 - A register that goes through a **prescribed** sequence of states upon the application of input pulses
 - Input pulses: may be **clock pulses** or originate from some **external source**
 - The sequence of states: may follow the binary number sequence (**binary counter**) or any **other** sequence of states



6-3 Ripple Counters

- Categories of counters
 - **Ripple** counters
 - The **flip-flop output transition** serves as a source for triggering other flip-flops
 - **no common clock pulse** (not synchronous)
 - **Synchronous** counters:
 - The CLK inputs of all flip-flops receive a **common clock**



6-3 Ripple Counters

Ex. 4-bit binary ripple counter

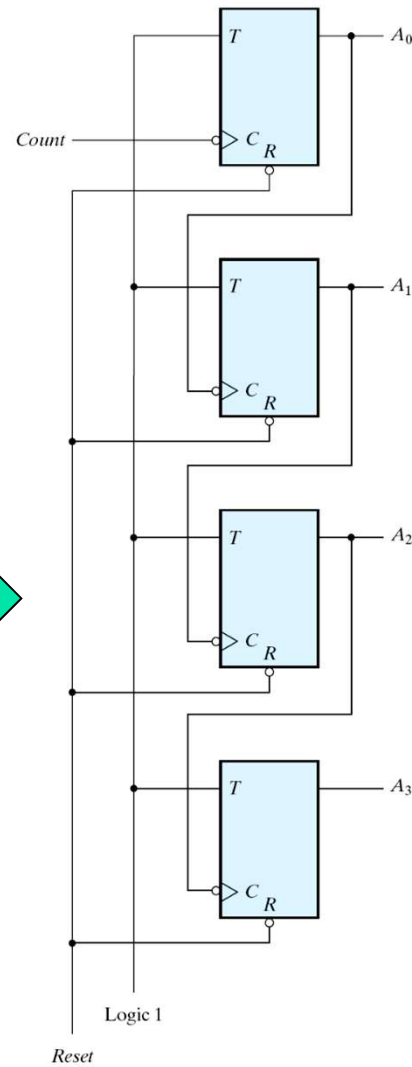
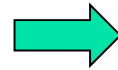
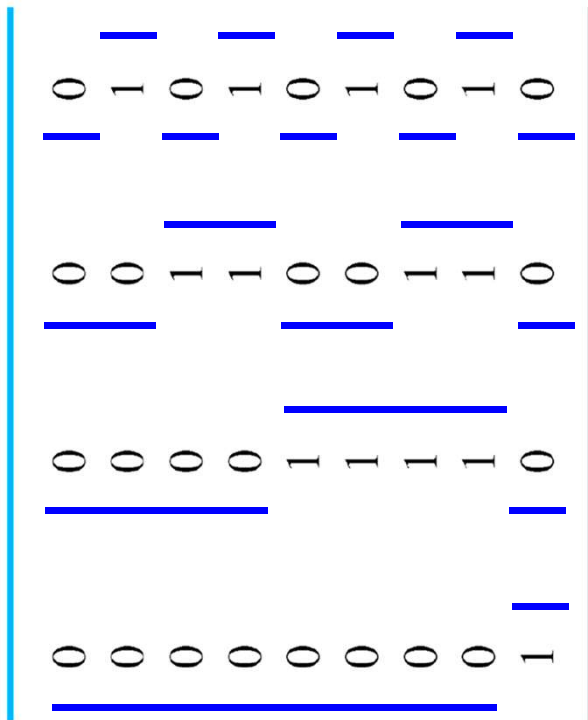
Binary Count Sequence

A_3	A_2	A_1	A_0
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0

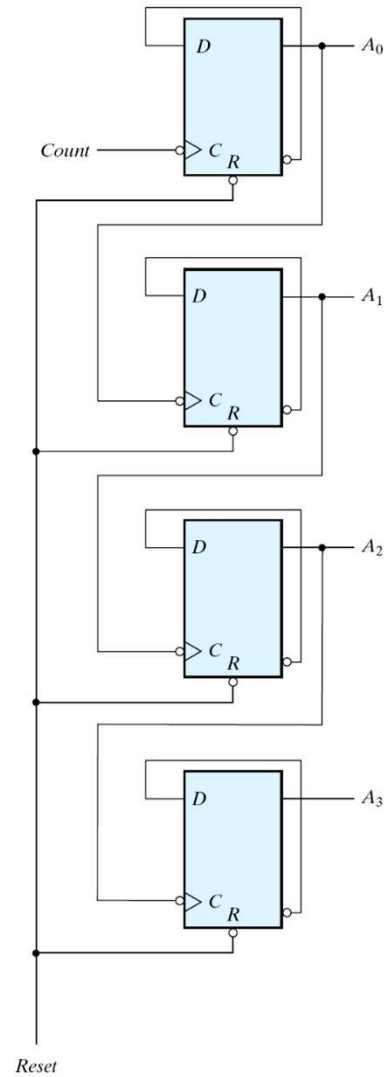
← *incomplete*

6-3 Ripple Counters

Ex. (cont.)



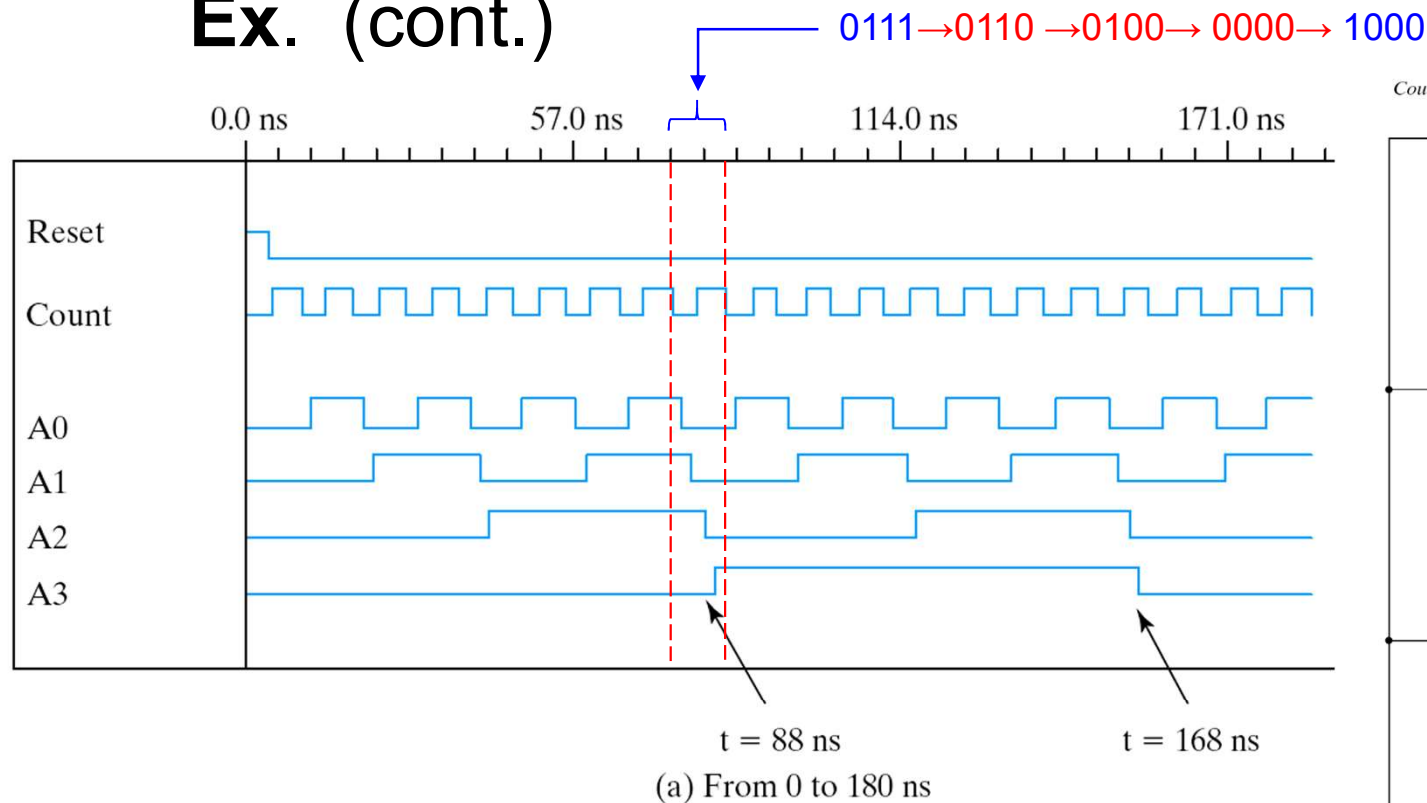
(a) With T flip-flops



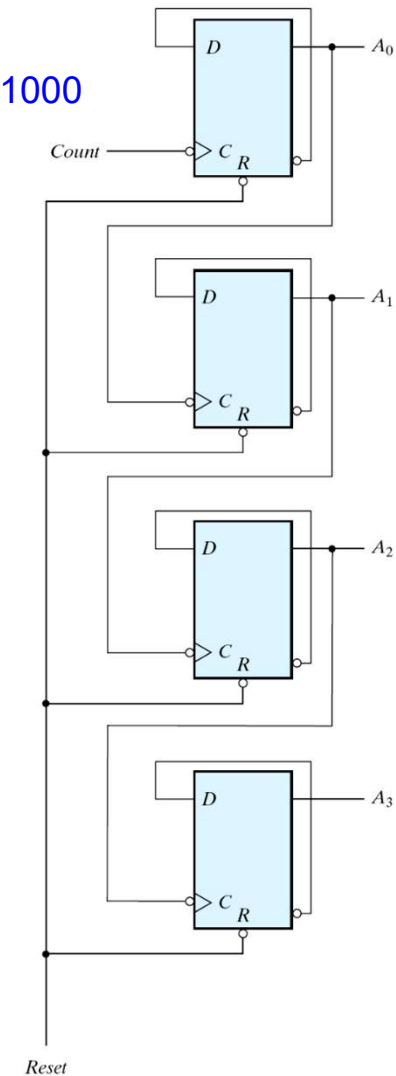
(b) With D flip-flops

6-3 Ripple Counters

Ex. (cont.)



- longer propagation delay or more bits
→ cannot read at the inactive clock edge

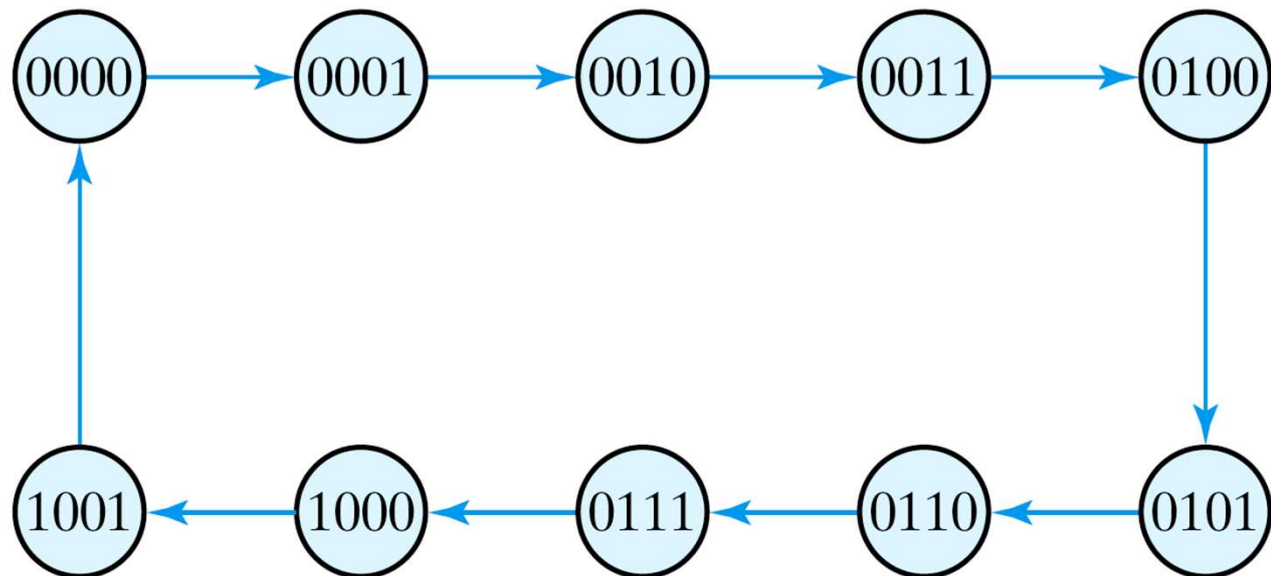


(b) With D flip-flops



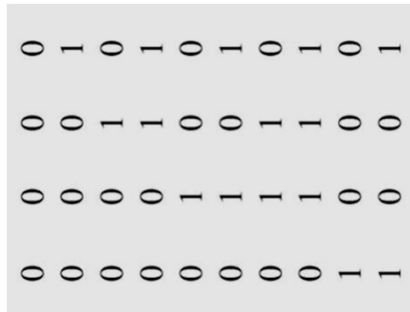
6-3 Ripple Counters

Ex. BCD ripple counter

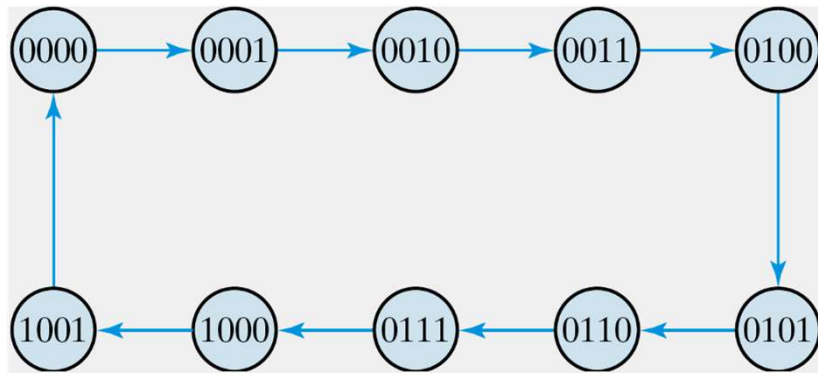
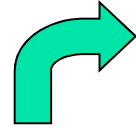


6-3 Ripple Counters

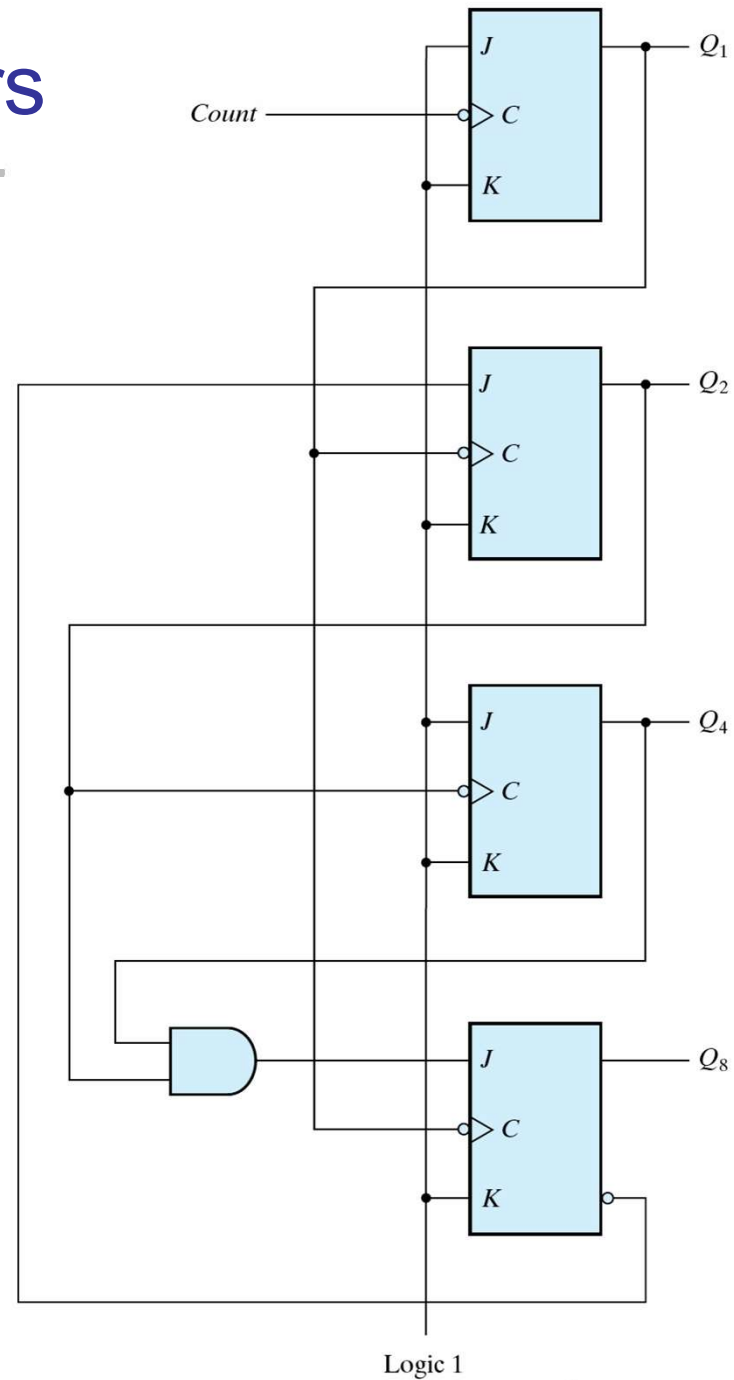
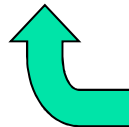
Ex. (cont.)



design?

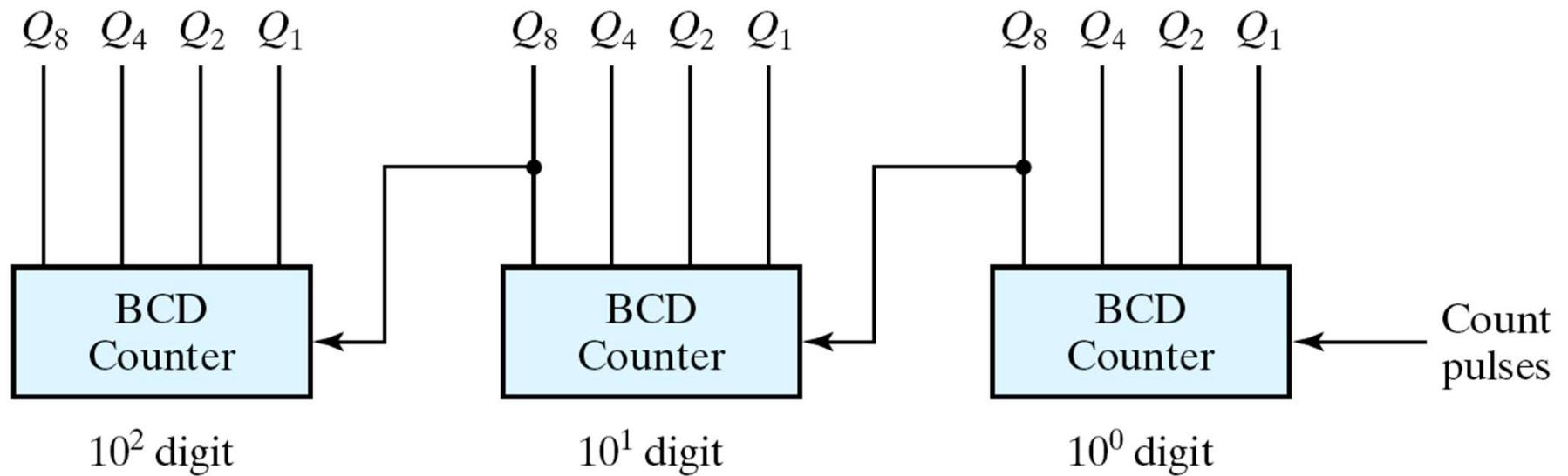


analysis



6-3 Ripple Counters

Ex. Three-decade BCD counter





6-4 Synchronous Counters

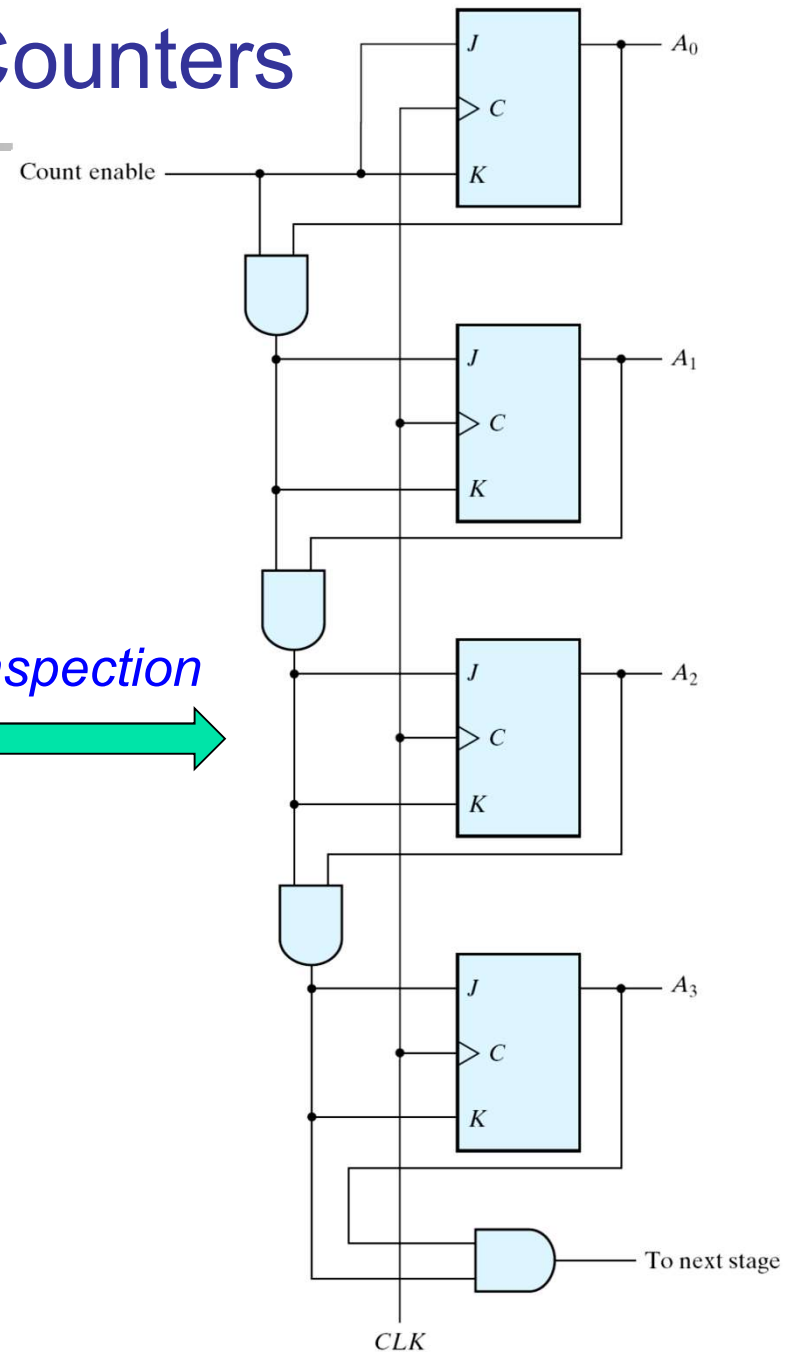
- Synchronous counter
 - A **common clock** triggers all flip-flops simultaneously
- **Design** procedure
 - Apply the same procedure discussed in **Sec. 5.8**
- **Binary** counter
 - The design is **so simple** that there is **no need** to go through a sequential circuit design process

6-4 Synchronous Counters

Ex. 4-bit **binary** counter

A_2	A_1	A_0
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

inspection →



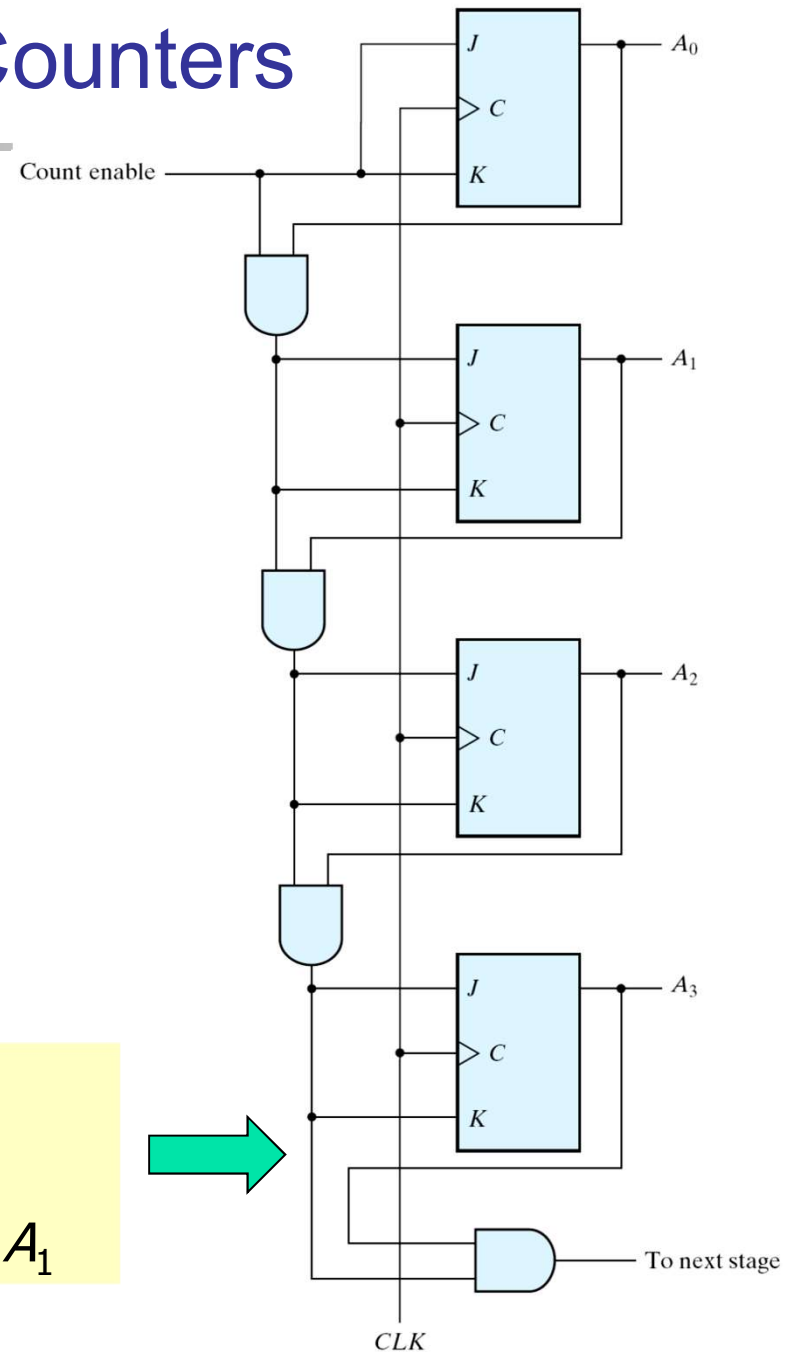
6-4 Synchronous Counters

Ex. (cont.)

A_2	A_1	A_0
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

design

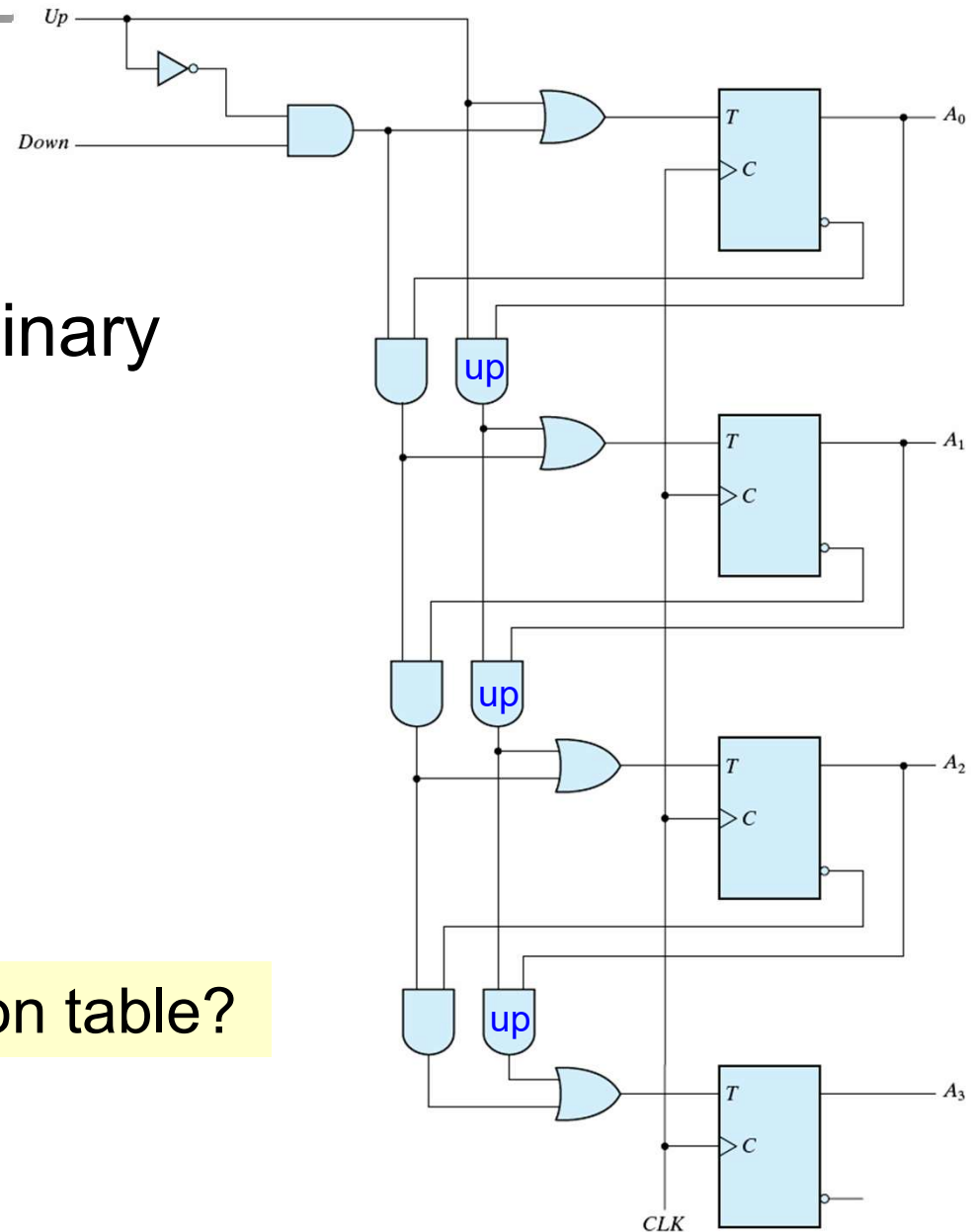
$$\begin{aligned}
 J_{A0} &= K_{A0} = E \\
 J_{A1} &= K_{A1} = E A_0 \\
 J_{A2} &= K_{A2} = E A_0 A_1
 \end{aligned}$$



6-4 Synchronous Counters


Ex. 4-bit **up/down** binary counter

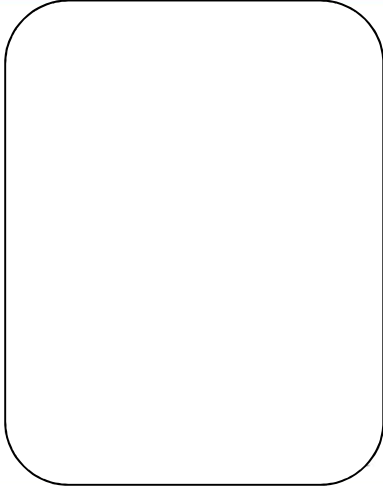
■ function table?



6-4 Synchronous Counters


Ex. BCD counters



Present State				Next State				Output	Flip-Flop Inputs			
Q_8	Q_4	Q_2	Q_1	Q_8	Q_4	Q_2	Q_1	y	TQ_8	TQ_4	TQ_2	TQ_1
0	0	0	0	0	0	0	1	0				
0	0	0	1	0	0	1	0	0				
0	0	1	0	0	0	1	1	0				
0	0	1	1	0	1	0	0	0				
0	1	0	0	0	1	0	1	0				
0	1	0	1	0	1	1	0	0				
0	1	1	0	0	1	1	1	0				
0	1	1	1	1	0	0	0	0				
1	0	0	0	1	0	0	1	0				
1	0	0	1	0	0	0	0	1				

6-4 Synchronous Counters

Ex. (cont.)



Present State				Next State				Output	Flip-Flop Inputs			
Q_8	Q_4	Q_2	Q_1	Q_8	Q_4	Q_2	Q_1	y	TQ_8	TQ_4	TQ_2	TQ_1
0	0	0	0	0	0	0	1	0	0	0	0	1
0	0	0	1	0	0	1	0	0	0	0	1	1
0	0	1	0	0	0	1	1	0	0	0	0	1
0	0	1	1	0	1	0	0	0	0	1	1	1
0	1	0	0	0	1	0	1	0	0	0	0	1
0	1	0	1	0	1	1	0	0	0	0	1	1
0	1	1	0	0	1	1	1	0	0	0	0	1
0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	0	0	1	0	0	0	0	1
1	0	0	1	0	0	0	0	1	1	0	0	1

- Simplified functions:

$$T_{Q1} = 1$$

$$T_{Q2} = Q_8'Q_1$$

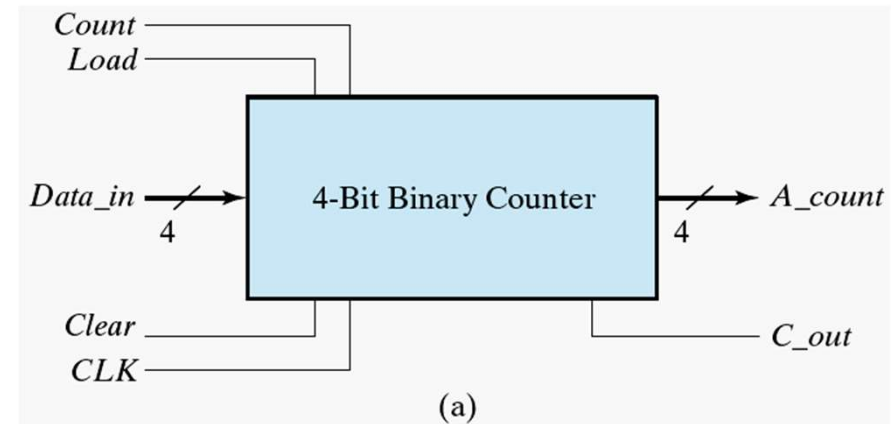
$$T_{Q4} = Q_2Q_1$$

$$T_{Q8} = Q_8Q_1 + Q_4Q_2Q_1$$

$$y = Q_8Q_1$$

6-4 Synchronous Counters

Ex. 4-bit **binary** counter w/
parallel load



Function Table for the Counter of Fig. 6.14

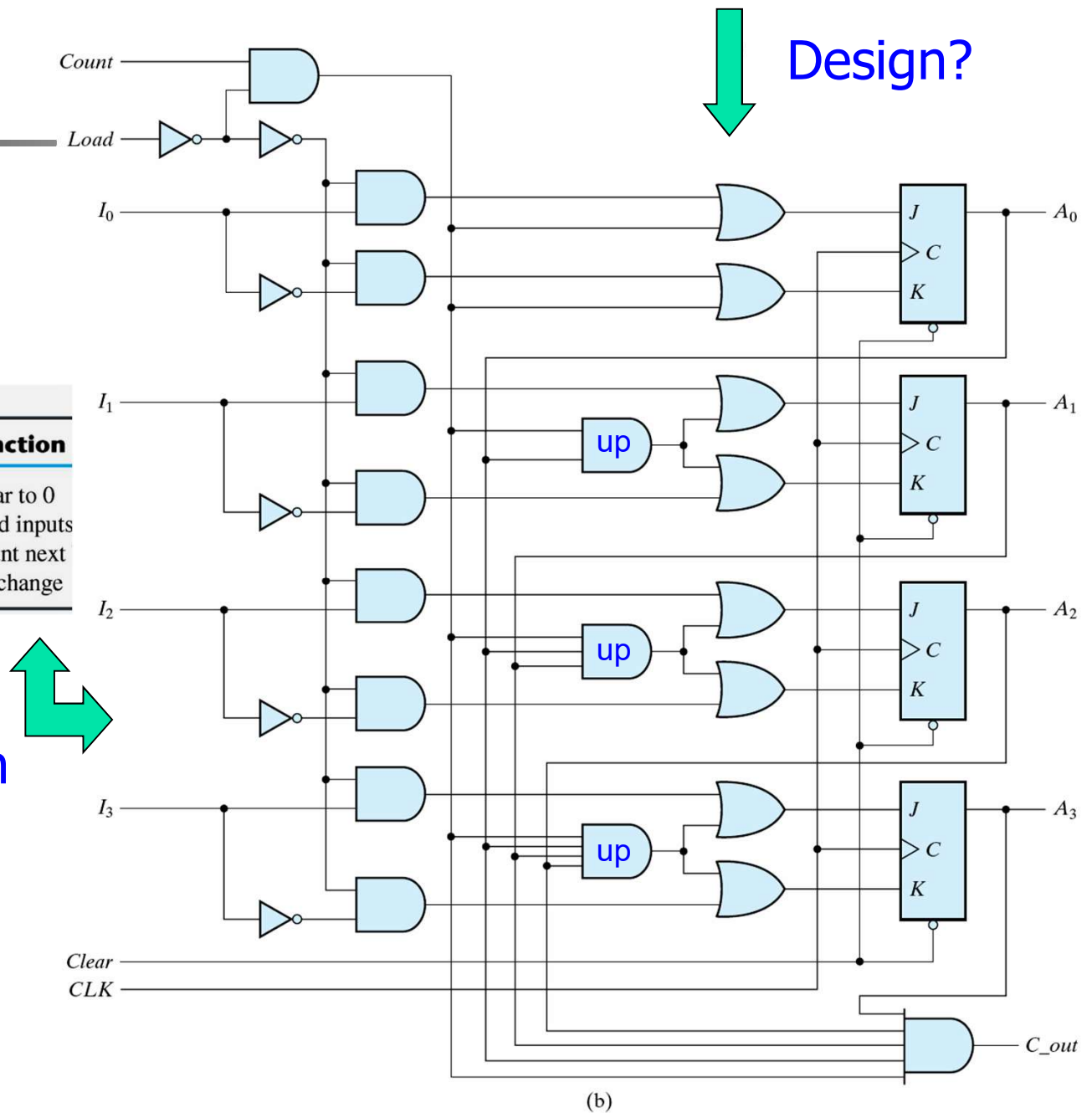
Clear	CLK	Load	Count	Function
0	X	X	X	Clear to 0
1	↑	1	X	Load inputs
1	↑	0	1	Count next binary state
1	↑	0	0	No change

Ex. (cont.)

Function Table for the Counter of Fig. 6.14

Clear	CLK	Load	Count	Function
0	X	X	X	Clear to 0
1	↑	1	X	Load inputs
1	↑	0	1	Count next
1	↑	0	0	No change

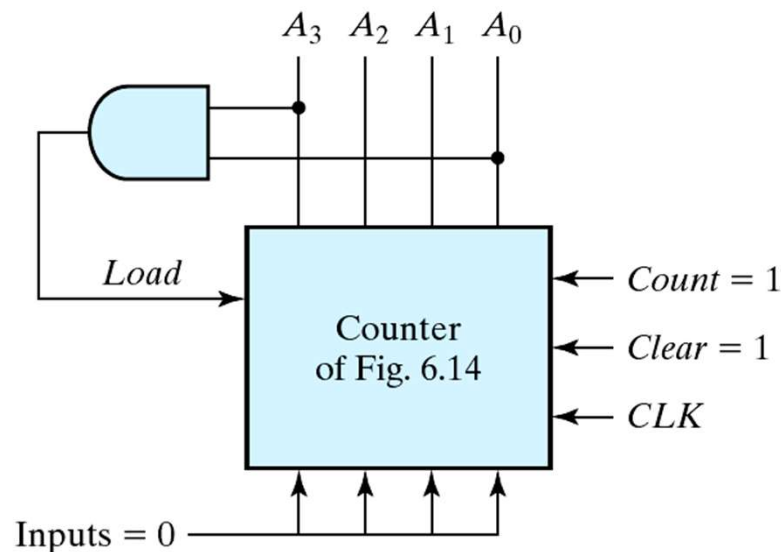
analysis/verification



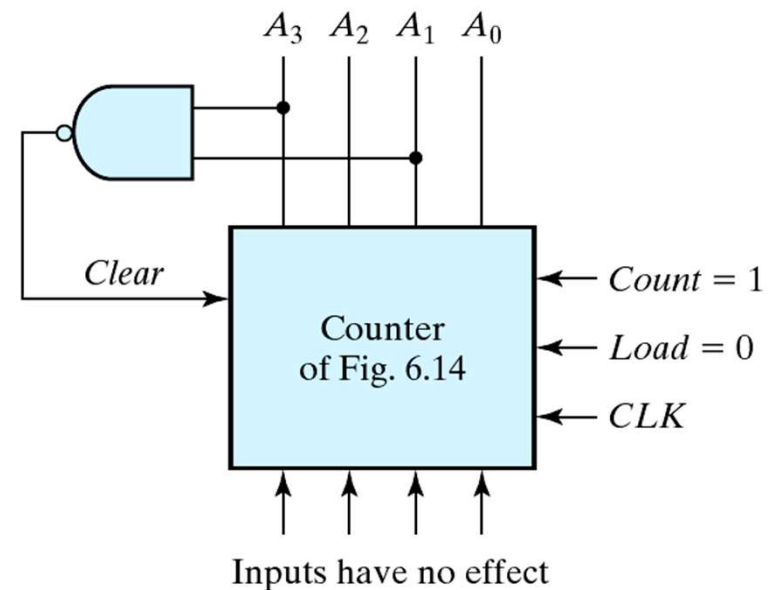
6-4 Synchronous Counters

Ex. Generate **any** count sequence

- **BCD** counter \Leftarrow Counter w/ parallel load



... 8 $\rightarrow \rightarrow \uparrow \rightarrow$ 9 $\rightarrow \rightarrow \uparrow \rightarrow$ 0



... 8 $\rightarrow \rightarrow \uparrow \rightarrow$ 9 $\rightarrow \rightarrow \uparrow \rightarrow$ **10** \rightarrow 0

(Spike in A_1 ?)



6-5 Other Counters

- Counters:
 - can be designed to generate **any** desired sequence of states
- **Divide-by- N** counter (**modulo- N** counter)
 - a counter that goes through a repeated sequence of **N states**
 - The sequence may follow the **binary** count or may be any other **arbitrary** sequence



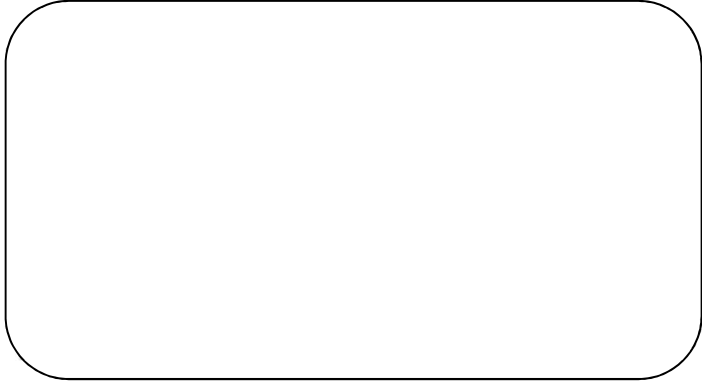
Counters with Unused States

- n flip-flops $\Rightarrow 2^n$ binary states
- **Unused** states for a state table/diagram:
 - may be treated as **don't-care** conditions or
may be assigned **specific next states**
- **Self-correcting** counter
 - When a circuit enters one of its **unused states**, it will eventually go into one of the **valid states** ...
normal operation.

\Rightarrow **Analysis/verification** for a designed circuit

Counters with Unused States

Ex. A divide-by-6 counter

Present State			Next State			Flip-Flop Inputs					
A	B	C	A	B	C	J_A	K_A	J_B	K_B	J_C	K_C
0	0	0	0	0	1						
0	0	1	0	1	0						
0	1	0	1	0	0						
1	0	0	1	0	1						
1	0	1	1	1	0						
1	1	0	0	0	0						

Counters with Unused States

Ex. (cont.)

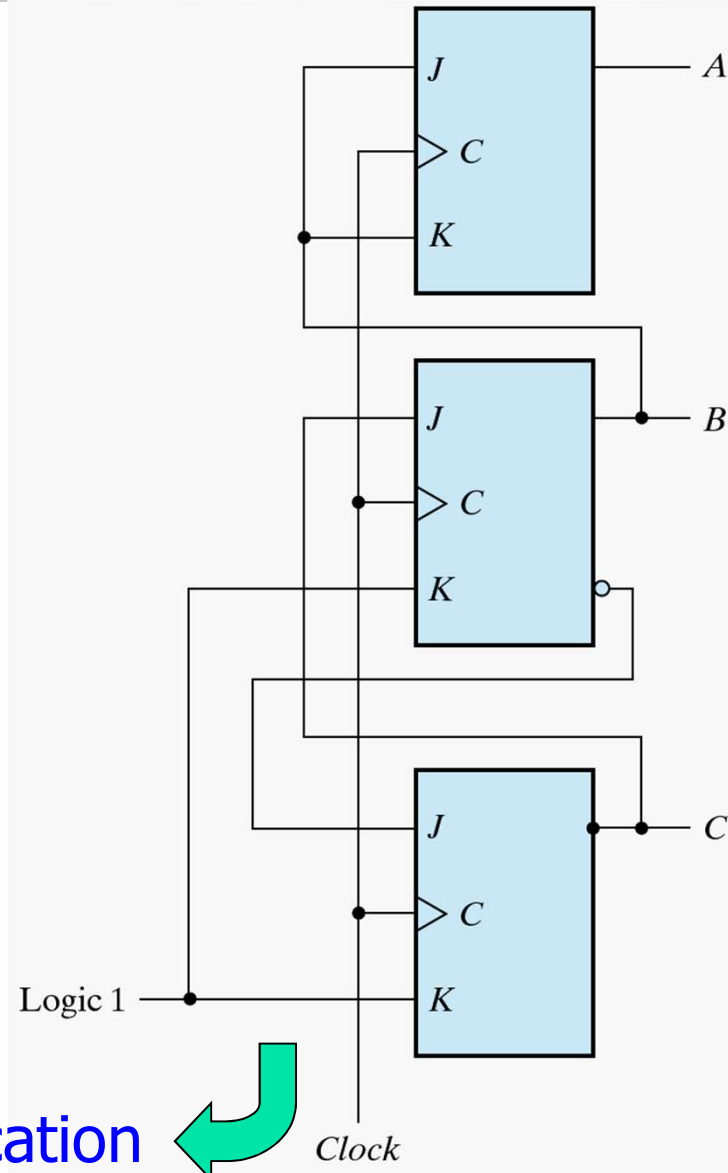

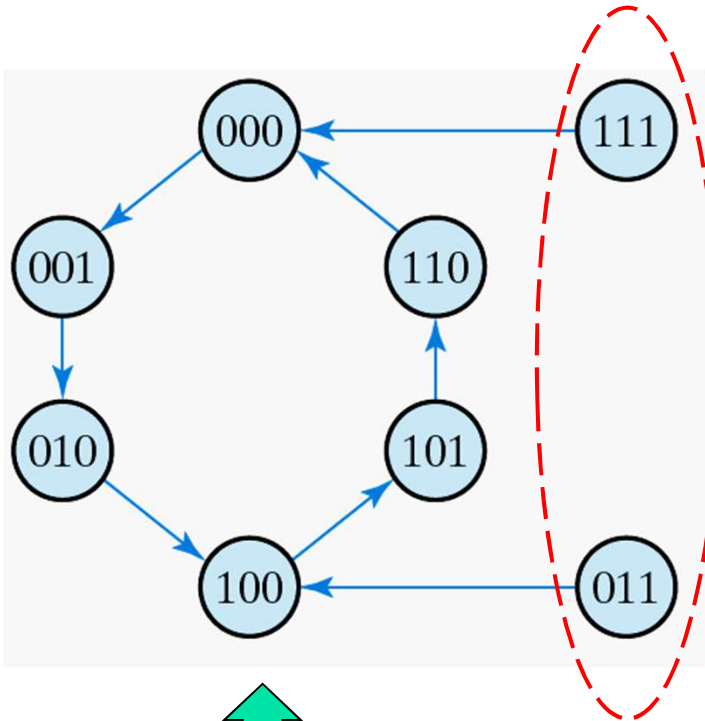
Present State			Next State			Flip-Flop Inputs					
A	B	C	A	B	C	J_A	K_A	J_B	K_B	J_C	K_C
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	0	0	X	1	X	X	1
0	1	0	1	0	0	1	X	X	1	0	X
1	0	0	1	0	1	X	0	0	X	1	X
1	0	1	1	1	0	X	0	1	X	X	1
1	1	0	0	0	0	X	1	X	1	0	X

$$J_A = B, \quad J_B = C, \quad J_C = B',$$

$$K_A = B, \quad K_B = 1, \quad K_C = 1$$

Ex. (cont.)

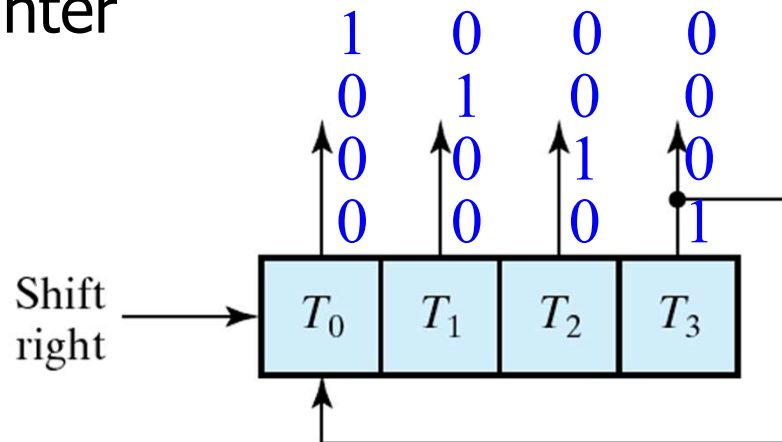
- The logic diagram & state diagram



Ring Counter

- Ring counter:
 - a **circular shift** register w/ **only one** flip-flop being **set** at any particular time, all others are cleared (initial value = **1 0 0 ... 0**)
 - The single bit is shifted from one flip-flop to the next to produce the sequence of **timing signals**

Ex. A 4-bit ring counter



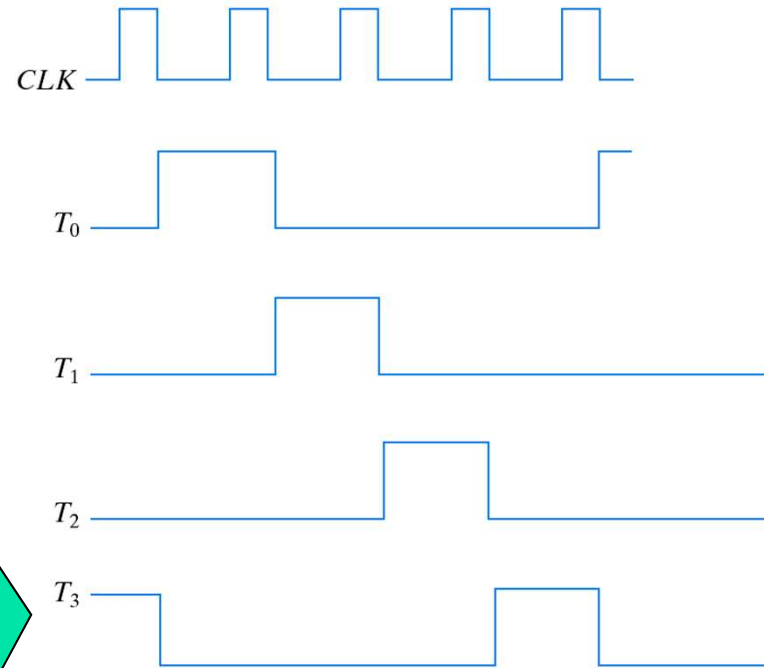
(a) Ring-counter (initial value = 1000)

Ring Counter

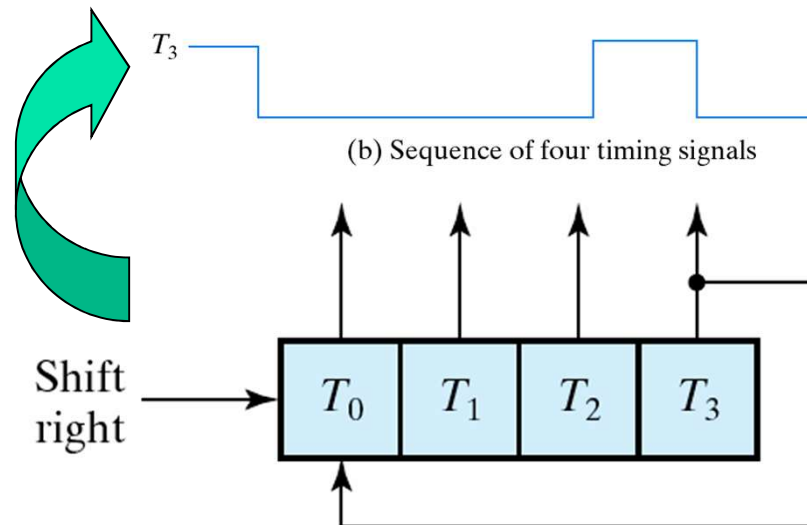
Ex. (cont.)

- **timing signals:**

pulses generated by the clock of a digital computer to **synchronize** its activities.



(b) Sequence of four timing signals

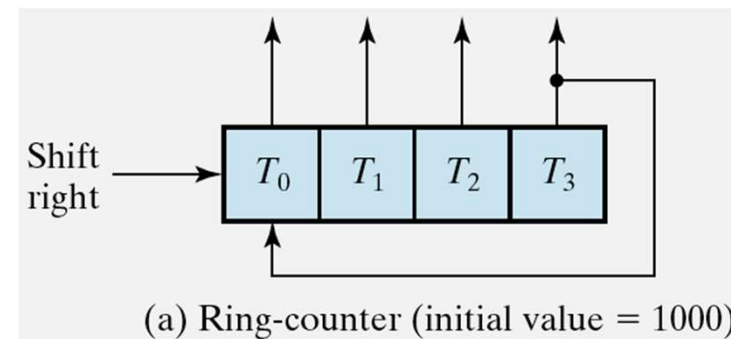
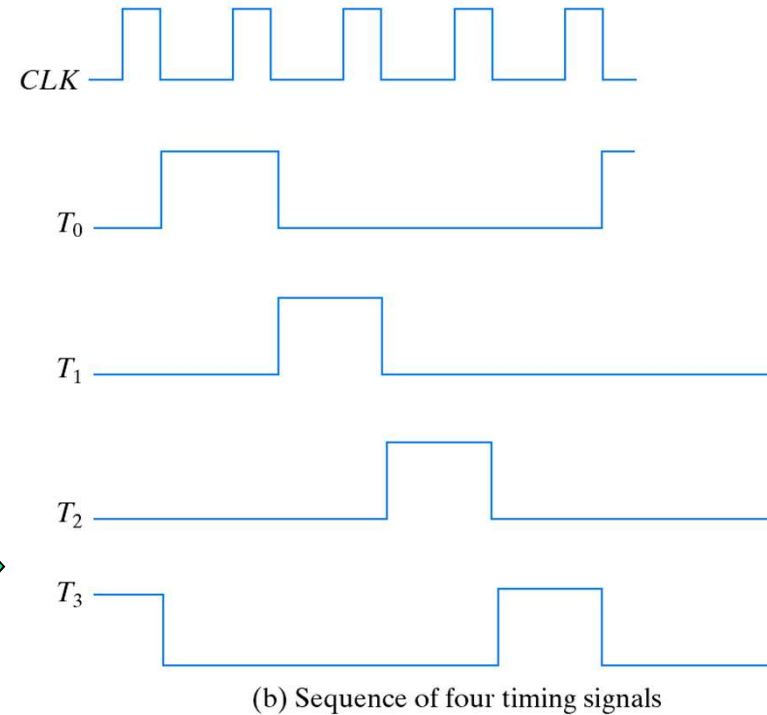
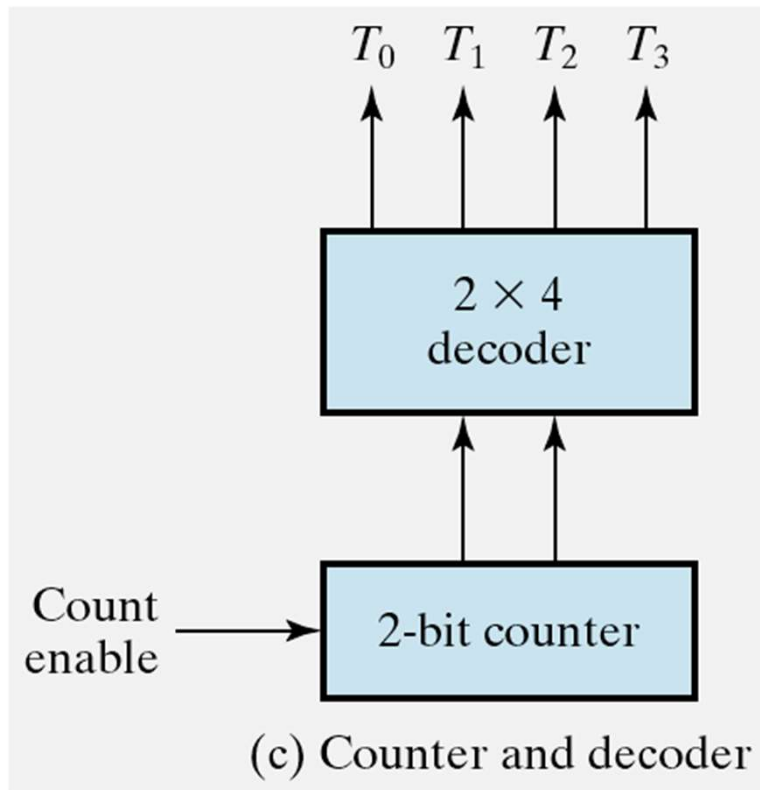


(a) Ring-counter (initial value = 1000)

Ring Counter

Ex. (cont.)

- An alternative





Ring Counter

- Approaches for generation of 2^n timing signals
 1. a shift register with 2^n flip-flops
 2. an n -bit binary counter together with an n -to- 2^n -line decoder

Ex. 16 timing signals

1. a shift register with 16 flip-flops
2. an 4-bit binary counter together with an 4-to-16-line decoder

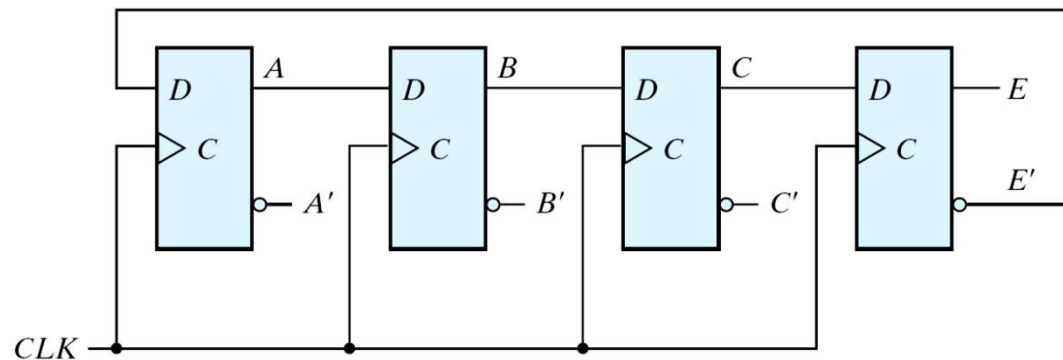


Johnson Counter

- Ring counter vs. Switch-tail ring counter
 - Ring counter
 - a k -bit ring counter circulates a single bit among the flip-flops to provide k distinguishable states.
 - Switch-tail ring counter
 - a shift register with complement output of the last flip-flop connected to input of the first flip-flop
 - a k -bit switch-tail ring counter will go through a sequence of $2k$ distinguishable states. (initial value = 0 0 ... 0)

Johnson Counter

Ex. Construction of a 4-bit Johnson counter



(a) Four-stage switch-tail ring counter

0	0	0	0
1	0	0	0
1	1	0	0
1	1	1	0
1	1	1	1
0	1	1	1
0	0	1	1
0	0	0	1

Switch-tail



Johnson Counter

Ex. (cont.)

- 8 timing signals (8 decoding gates)
- 2 inputs per decoding gate

Sequence number	Flip-flop outputs				AND gate required for output
	A	B	C	E	
1	0	0	0	0	$A'E'$
2	1	0	0	0	AB'
3	1	1	0	0	BC'
4	1	1	1	0	CE'
5	1	1	1	1	AE
6	0	1	1	1	$A'B$
7	0	0	1	1	$B'C$
8	0	0	0	1	$C'E$

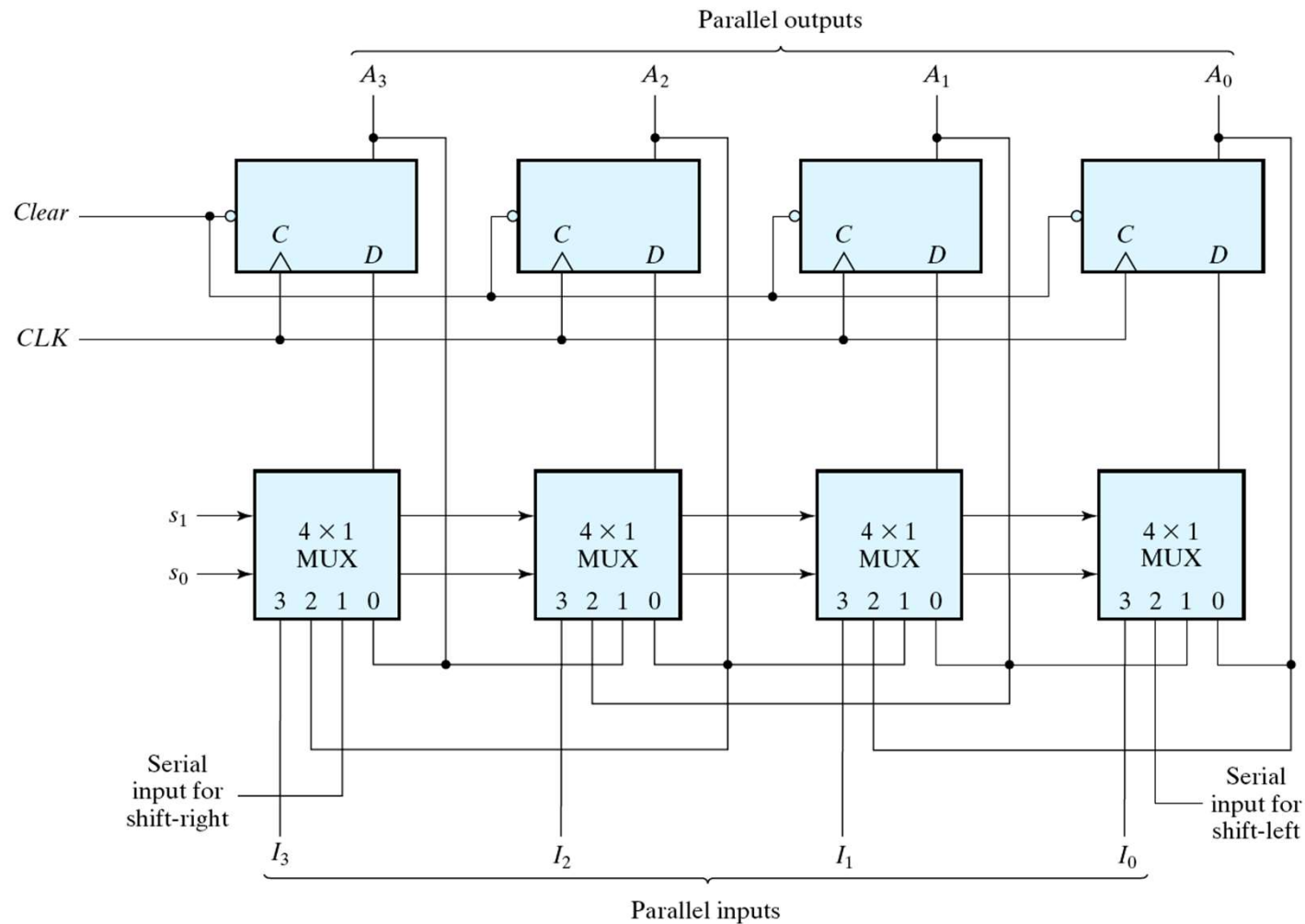


Johnson Counter

- Disadvantage of the switch-tail ring counter
 - if it finds itself in an **unused state**, it will persist to circulate in the invalid states and never find its way to a valid state.
 - One **correcting** procedure: $D_C = (A + C) B$
- Summary:
 - Construction of a Johnson counter for k timing sequences:
 1. $k/2$ flip-flops
 2. k decoding gates with 2 inputs per gate

6-6 HDL for Registers and Counters

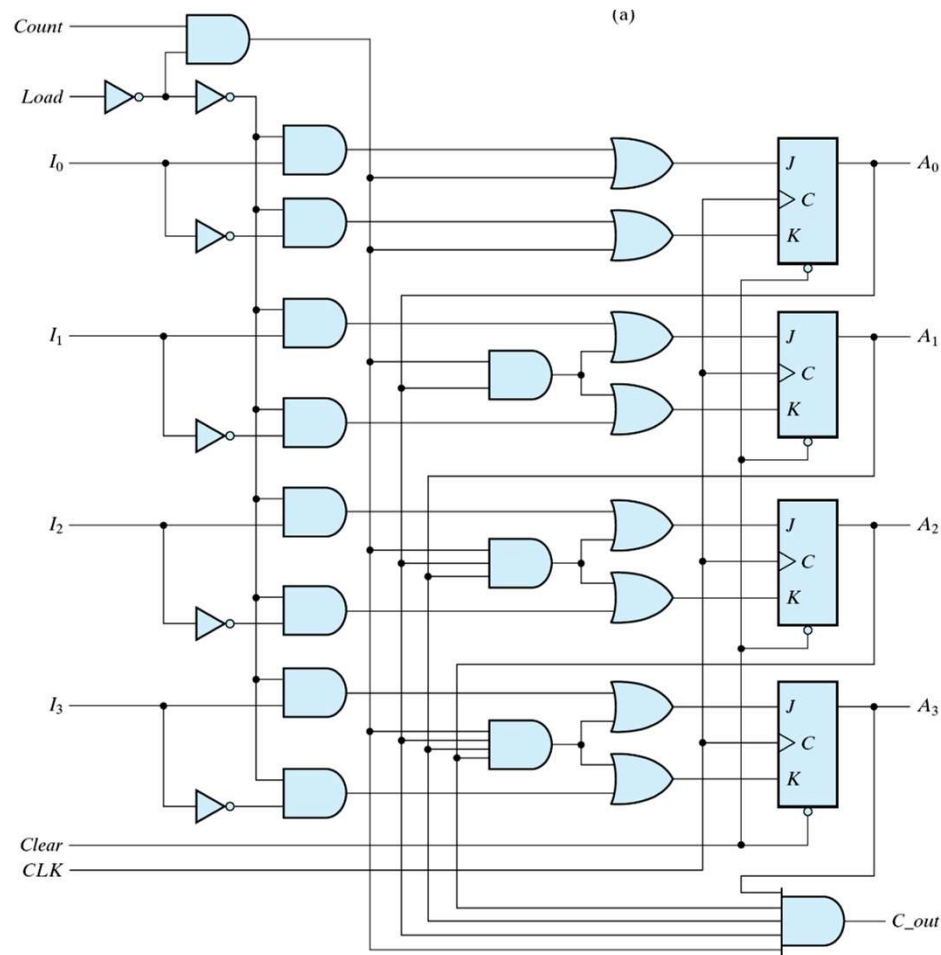
- Register: 4-bit **universal** shift register



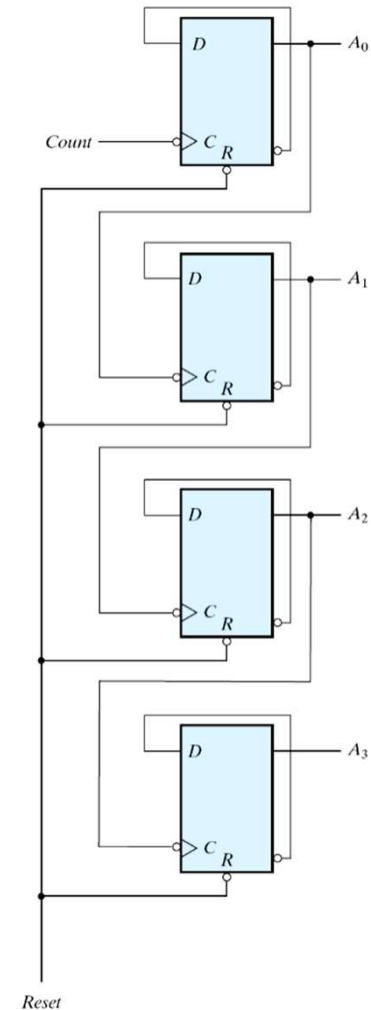
6-6 HDL for Registers and Counters

■ Counters:

4-bit counter w/ parallel load



Ripple counters



HDL Example 6.1

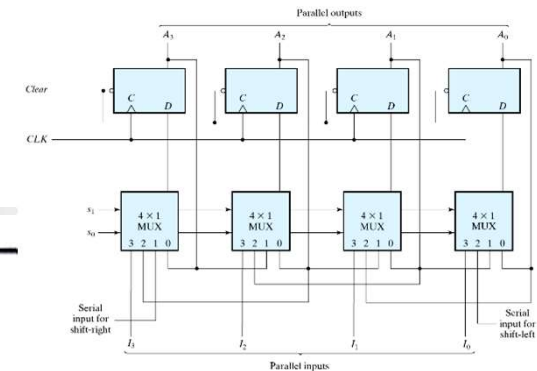
// Behavioral description of a 4-bit universal shift register
// Fig. 6.7 and Table 6.3

```

module Shift_Register_4_beh (
    output reg      [3: 0]  A_par,           // V2001, 2005
                                           // Register output
    input          [3: 0]  I_par,           // Parallel input
    input          s1, s0,                 // Select inputs
                                           // Serial inputs
                                           // Clock and Clear
    MSB_in, LSB_in,
    CLK, Clear

);
always @ (posedge CLK, negedge Clear) // V2001, 2005
    if (~Clear) A_par <= 4'b0000;
    else
        case ({s1, s0})
            2'b00: A_par <= A_par;           // No change ← optional
            2'b01: A_par <= {MSB_in, A_par[3: 1]}; // Shift right
            2'b10: A_par <= {A_par[2: 0], LSB_in}; // Shift left
            2'b11: A_par <= I_par;           // Parallel load of input
        endcase
    endmodule

```



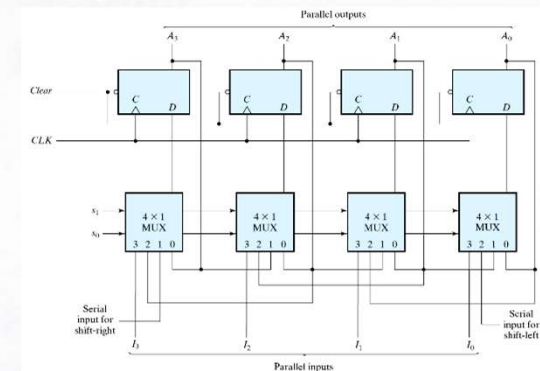
HDL Example 6.2

```
// Structural description of a 4-bit universal shift register (see Fig. 6.7)
module Shift_Register_4_str (                                // V2001, 2005
    output [3: 0] A_par,                                     // Parallel output
    input [3: 0] I_par,                                     // Parallel input
    input      s1, s0,                                     // Mode select
    input      MSB_in, LSB_in, CLK, Clear                 // Serial inputs, clock, clear
);
```

```
// bus for mode control
assign  [1:0] select = {s1, s0};
```

```
// Instantiate the four stages
```

```
stage ST0 (A_par[0], A_par[1], LSB_in, I_par[0], A_par[0], select, CLK, Clear);
stage ST1 (A_par[1], A_par[2], A_par[0], I_par[1], A_par[1], select, CLK, Clear);
```



HDL Example 6.2 (cont.)

```

    stage ST2 (A_par[2], A_par[3], A_par[1], I_par[2], A_par[2], select, CLK, Clear);
    stage ST3 (A_par[3], MSB_in, A_par[2], I_par[3], A_par[3], select, CLK, Clear);
endmodule

```

// One stage of shift register

```

module stage (i0, i1, i2, i3, Q, select, CLK, Clr);

```

```

    input      i0,          // circulation bit selection
                  i1,          // data from left neighbor or serial input for shift-right
                  i2,          // data from right neighbor or serial input for shift-left
                  i3;          // data from parallel input

```

```

    output     Q;

```

```

    input [1:0] select;      // stage mode control bus

```

```

    input      CLK, Clr;     // Clock, Clear for flip-flops

```

```

    wire       mux_out;

```

// instantiate mux and flip-flop

```

    Mux_4_x_1 M0      (mux_out, i0, i1, i2, i3, select);

```

```

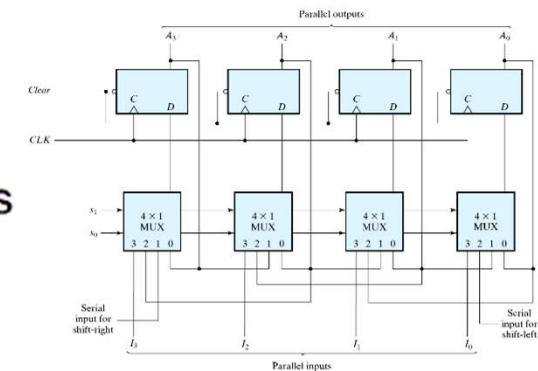
    D_flip_flop M1     (Q, mux_out, CLK, Clr);

```

```

endmodule

```



HDL Example 6.2 (cont.)

```
// 4x1 multiplexer           // behavioral model
module Mux_4_x_1 (mux_out, i0, i1, i2, i3, select);
```

```
    output      mux_out;
```

```
    input       i0, i1, i2, i3;
```

```
    input [1: 0] select;
```

```
    reg        mux_out;
```

```
    always @ (select, i0, i1, i2, i3)
```

```
        case (select)
```

```
            2'b00:    mux_out = i0;
```

```
            2'b01:    mux_out = i1;
```

```
            2'b10:    mux_out = i2;
```

```
            2'b11:    mux_out = i3;
```

```
        endcase
```

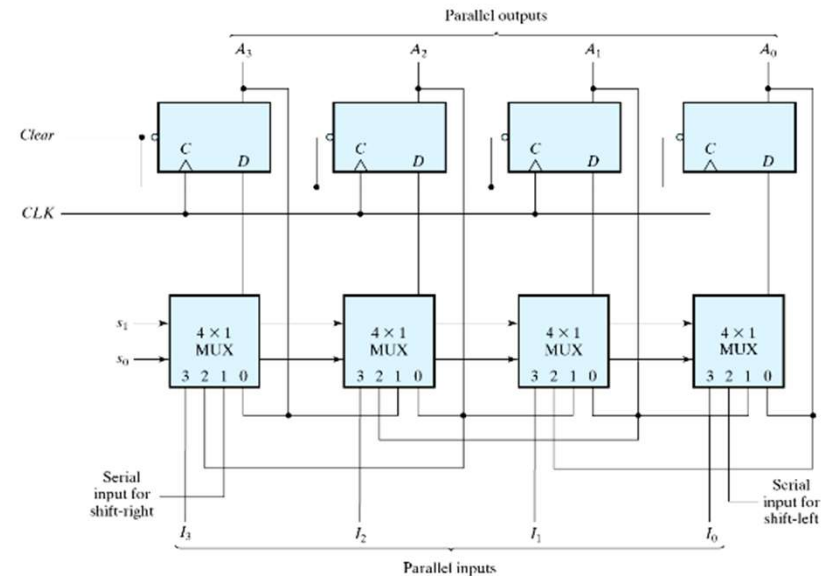
```
    endmodule
```

```
// Behavioral model of D flip-flop
module D_flip_flop (Q, D, CLK, Clr);
```

```
    output      Q;
```

```
    input       D, CLK, Clr;
```

```
    reg        Q;
```



```
    always @ (posedge CLK, negedge Clr)
        if (~Clr) Q <= 1'b0; else Q <= D;
    endmodule
```


HDL Example 6.3

// Four-bit binary counter with parallel load (V2001, 2005)

// See Figure 6.14 and Table 6.6

module Binary_Counter_4_Par_Load (

output reg [3: 0]

 A_count,

 // Data output

output

 C_out,

 // Output carry

input [3: 0]

 Data_in,

 // Data input

input

 Count,

 // Active high to count

 Load,

 // Active high to load

 CLK,

 // Positive-edge sensitive

 Clear

 // Active low

);

assign C_out = Count & (~Load) & (A_count == 4'b1111);

always @ (posedge CLK, negedge Clear)

if (~Clear)

 A_count <= 4'b0000;

else if (Load)

 A_count <= data_in;

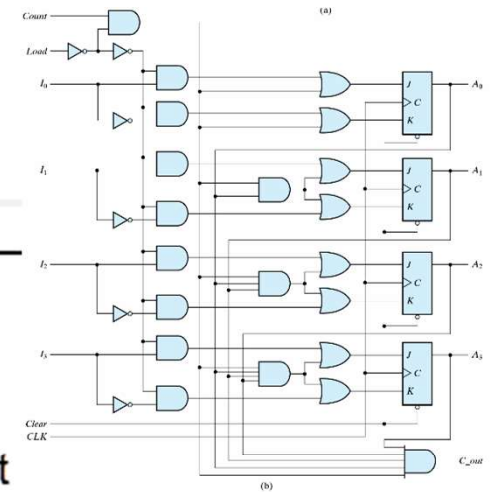
else if (Count)

 A_count <= A_count + 1'b1;

else

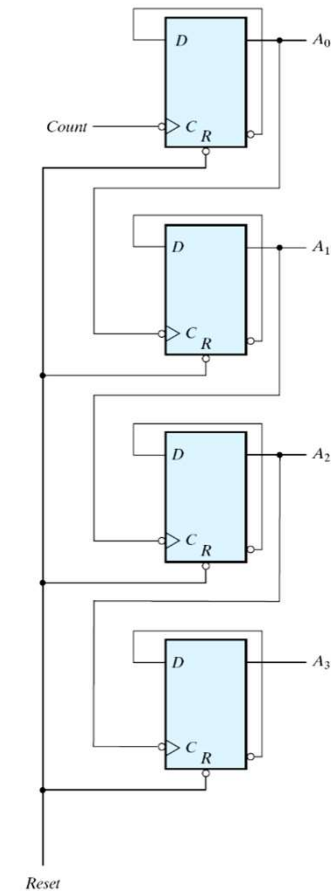
 A_count <= A_count; // redundant statement

endmodule



HDL Example 6.4

```
// Ripple counter (See Fig. 6.8(b))
`timescale 1ns / 100 ps
module Ripple_Counter_4bit (A3, A2, A1, A0, Count, Reset);
    output A3, A2, A1, A0;
    input  Count, Reset;
// Instantiate complementing flip-flop
    Comp_D_flip_flop F0 (A0, Count, Reset);
    Comp_D_flip_flop F1 (A1, A0, Reset);
    Comp_D_flip_flop F2 (A2, A1, Reset);
    Comp_D_flip_flop F3 (A3, A2, Reset);
endmodule
// Complementing flip-flop with delay
// Input to D flip-flop = Q'
module Comp_D_flip_flop (Q, CLK, Reset);
    output  Q;
    input   CLK, Reset;
    reg     Q;
    always @ (negedge CLK, posedge Reset)
        if (Reset) Q <= 1'b0;
        else Q <= #2 ~Q;           // intra-assignment delay
endmodule
```



(b) With D flip-flops

HDL Example 6.4 (cont.)

// Stimulus for testing ripple counter

```
module t_Ripple_Counter_4bit;
```

```
    reg      Count;
```

```
    reg      Reset;
```

```
    wire     A0, A1, A2, A3;
```

// Instantiate ripple counter

```
    Ripple_Counter_4bit M0 (A3, A2, A1, A0, Count, Reset);
```

```
always
```

```
    #5 Count = ~Count;
```

```
initial
```

```
    begin
```

```
        Count = 1'b0;
```

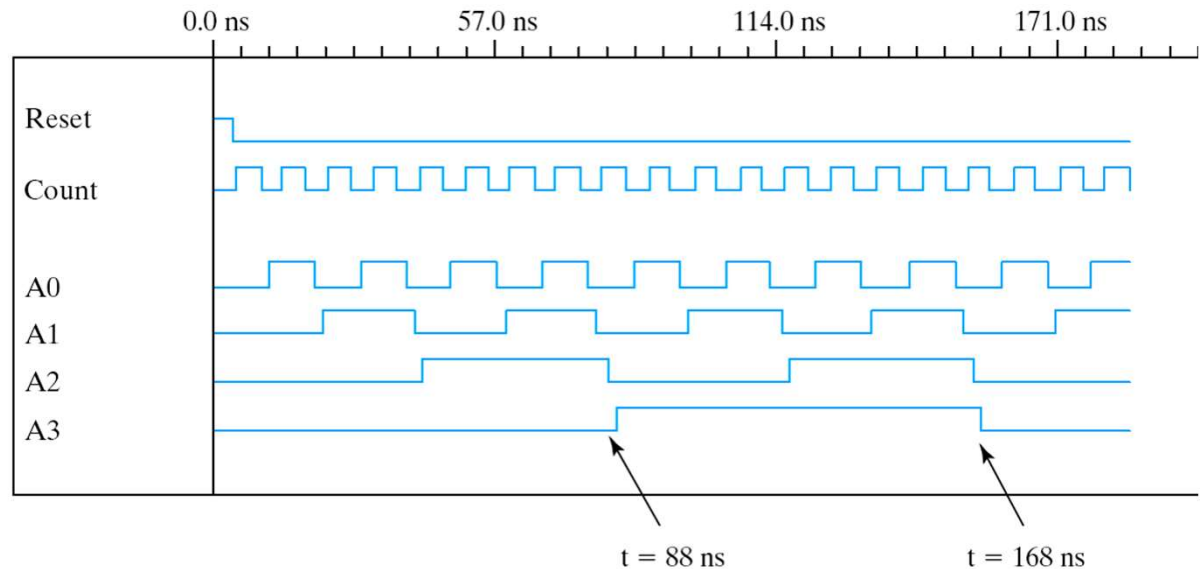
```
        Reset = 1'b1;
```

```
        #4 Reset = 1'b0;
```

```
    end
```

```
    initial #170 $finish;
```

```
endmodule
```



(a) From 0 to 180 ns