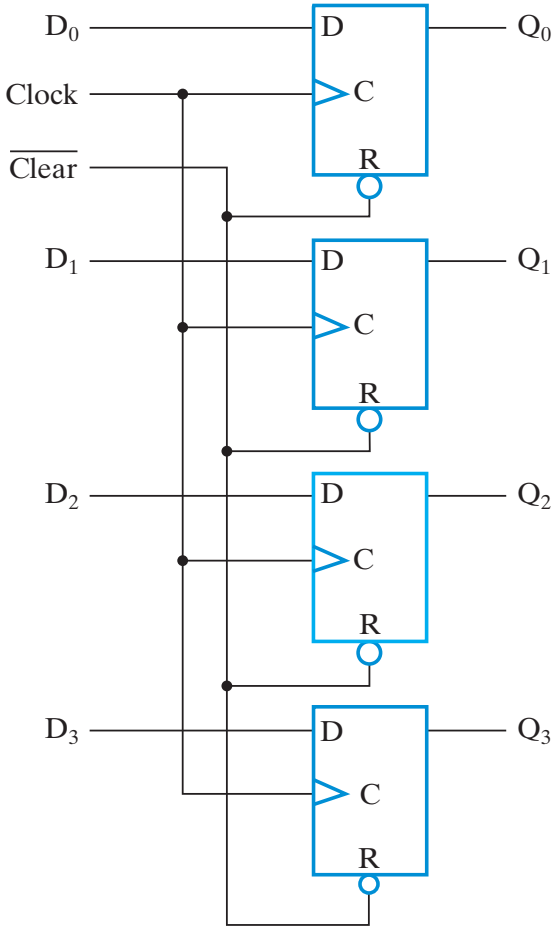
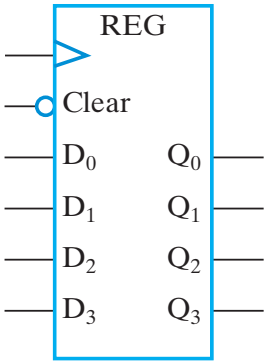
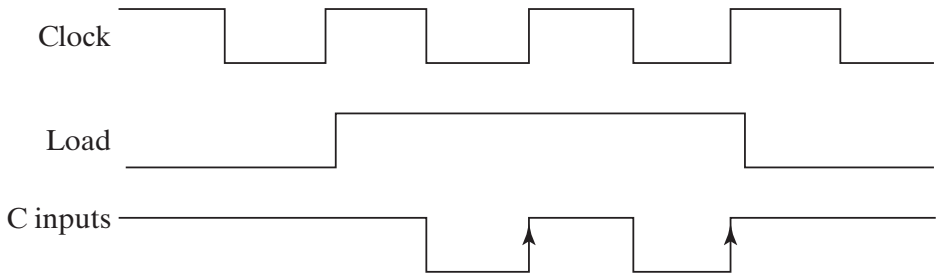


(a) Logic diagram

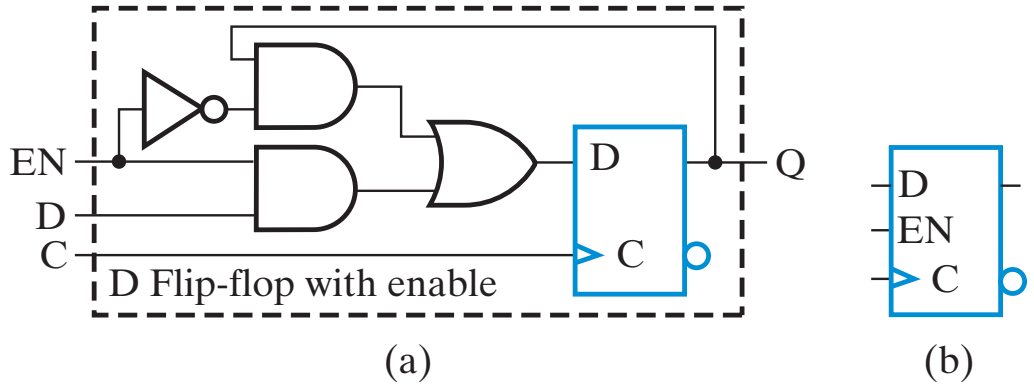


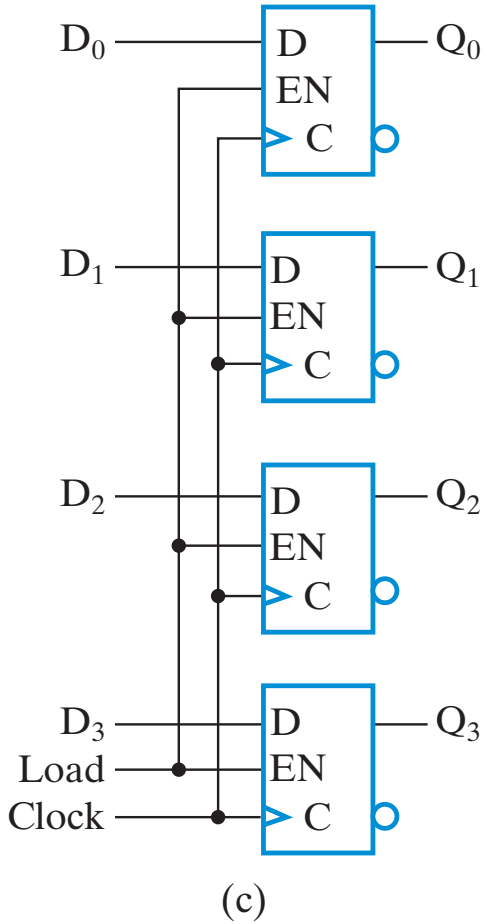
(b) Symbol

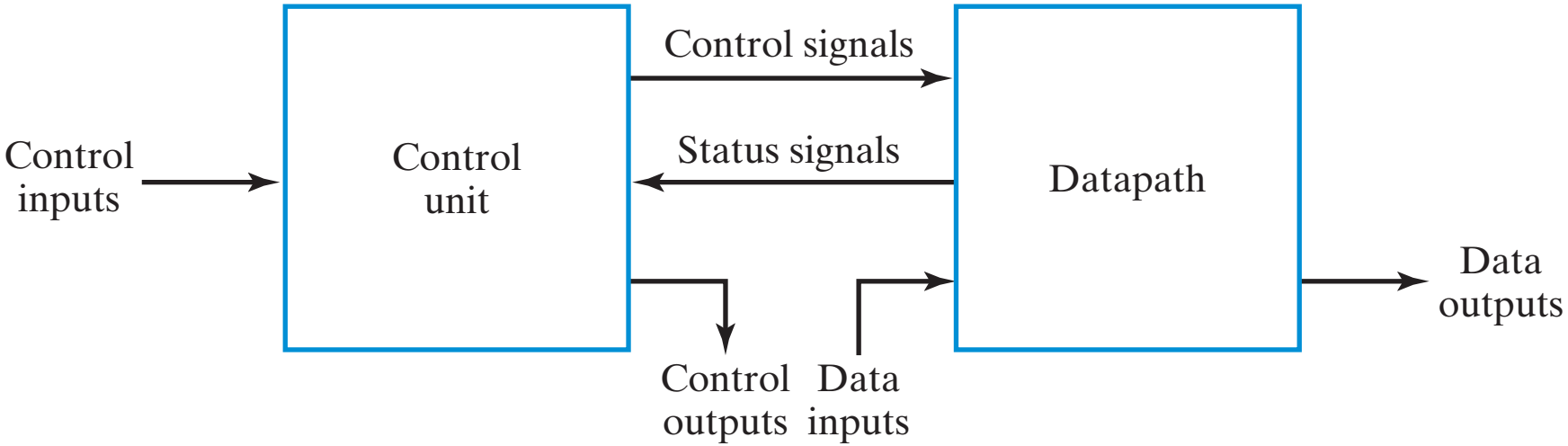
(c) Load control input



(d) Timing diagram

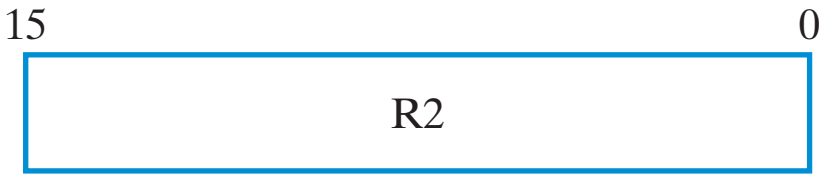




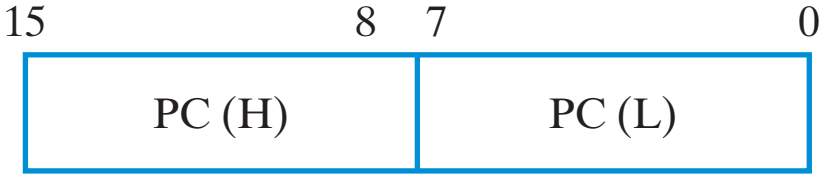


(a) Register R

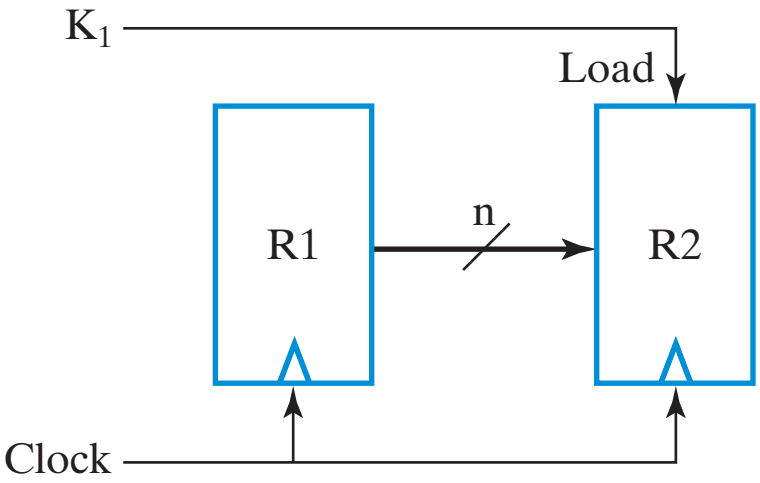
(b) Individual bits of 8-bit register



(c) Numbering of 16-bit register



(d) Two-part 16-bit register



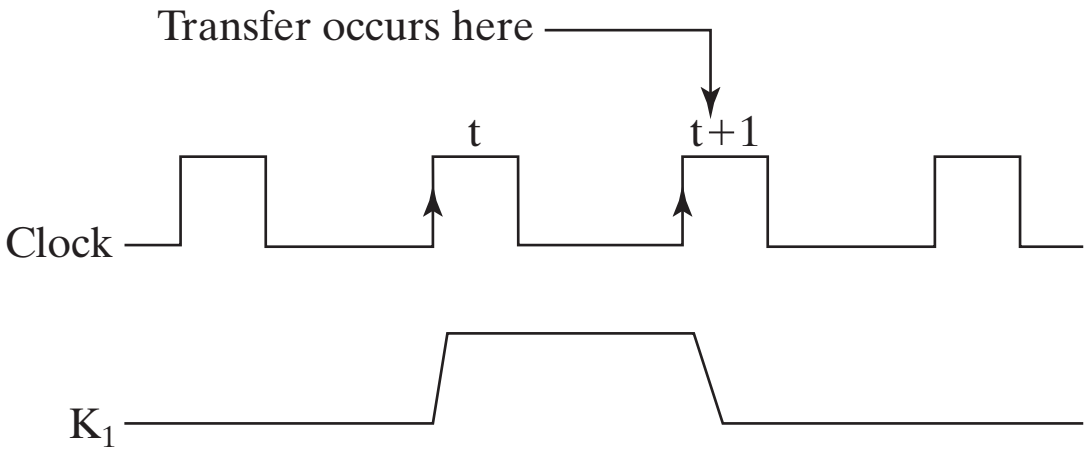


TABLE 7-1
Basic Symbols for Register Transfers

Symbol	Description	Examples
Letters (and numerals)	Denotes a register	$AR, R2, DR, IR$
Parentheses	Denotes a part of a register	$R2(1), R2(7{:}0), AR(L)$
Arrow	Denotes transfer of data	$R1 \leftarrow R2$
Comma	Separates simultaneous transfers	$R1 \leftarrow R2,\ R2 \leftarrow R1$
Square brackets	Specifies an address for memory	$DR \leftarrow M[AR]$

TABLE 7-2
Textbook RTL, VHDL, and Verilog Symbols for Register Transfers

Operation	Text RTL	VHDL	Verilog
Combinational assignment	=	<= (concurrent)	assign = (nonblocking)
Register transfer	←	<= (concurrent)	<= (nonblocking)
Addition	+	+	+
Subtraction	–	–	–
Bitwise AND	^	and	&
Bitwise OR	∨	or	|
Bitwise XOR	⊕	xor	^
Bitwise NOT	¯ (overline)	not	~
Shift left (logical)	sl	sll	<<
Shift right (logical)	sr	srl	>>
Vectors/registers	*A*(3:0)	*A*(3 down to 0)	*A*[3:0]
Concatenation	||	&	{ , }

TABLE 7-3
Arithmetic Microoperations

Symbolic designation	Description
$R0 \leftarrow R1 + R2$	Contents of $R1$ plus $R2$ transferred to $R0$
$R2 \leftarrow \overline{R2}$	Complement of the contents of $R2$ (1's complement)
$R2 \leftarrow \overline{R2} + 1$	2's complement of the contents of $R2$
$R0 \leftarrow R1 + \overline{R2} + 1$	$R1$ plus 2's complement of $R2$ transferred to $R0$ (subtraction)
$R1 \leftarrow R1 + 1$	Increment the contents of $R1$ (count up)
$R1 \leftarrow R1 - 1$	Decrement the contents of $R1$ (count down)

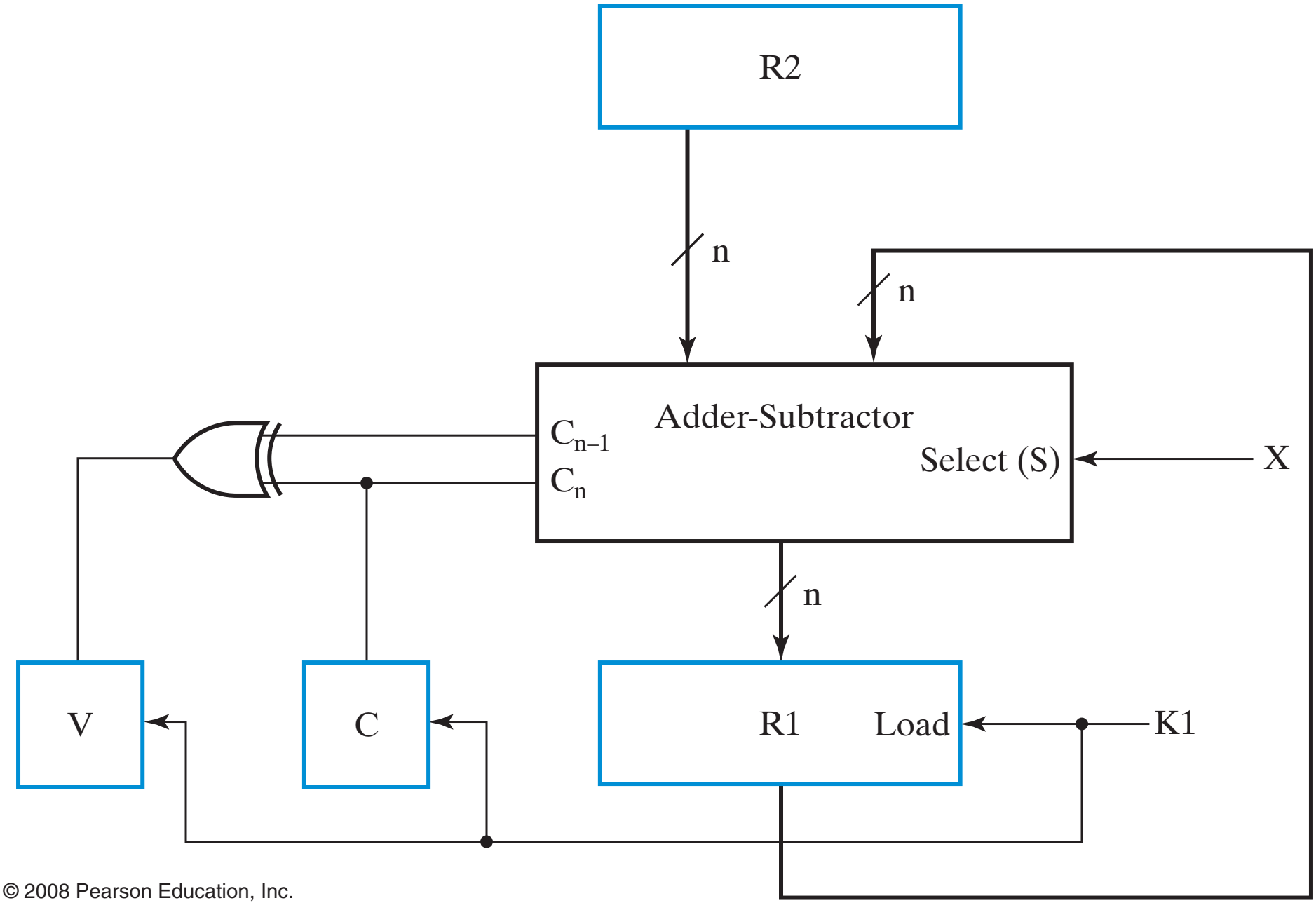
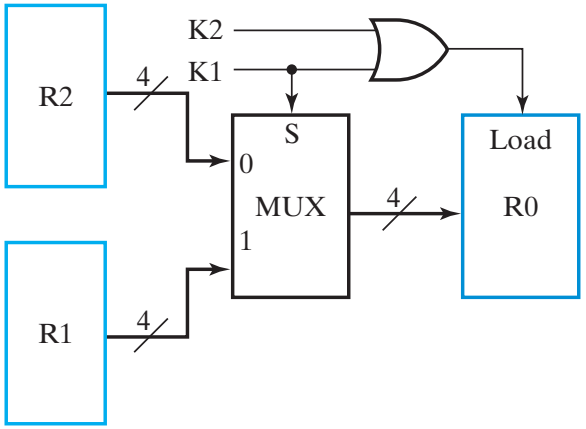


TABLE 7-4
Logic Microoperations

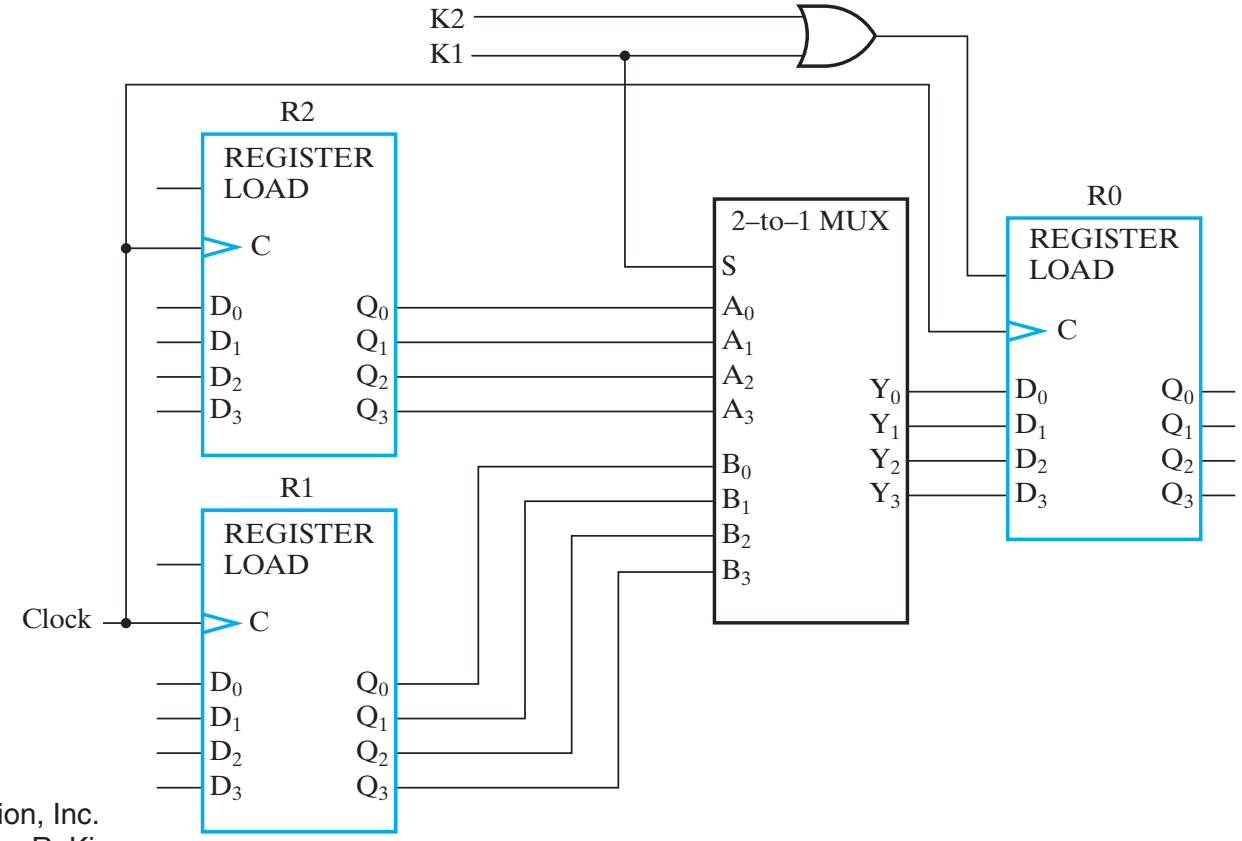
Symbolic designation	Description
$R0 \leftarrow \overline{R1}$	Logical bitwise NOT (1's complement)
$R0 \leftarrow R1 \wedge R2$	Logical bitwise AND (clears bits)
$R0 \leftarrow R1 \vee R2$	Logical bitwise OR (sets bits)
$R0 \leftarrow R1 \oplus R2$	Logical bitwise XOR (complements bits)

TABLE 7-5
Examples of Shifts

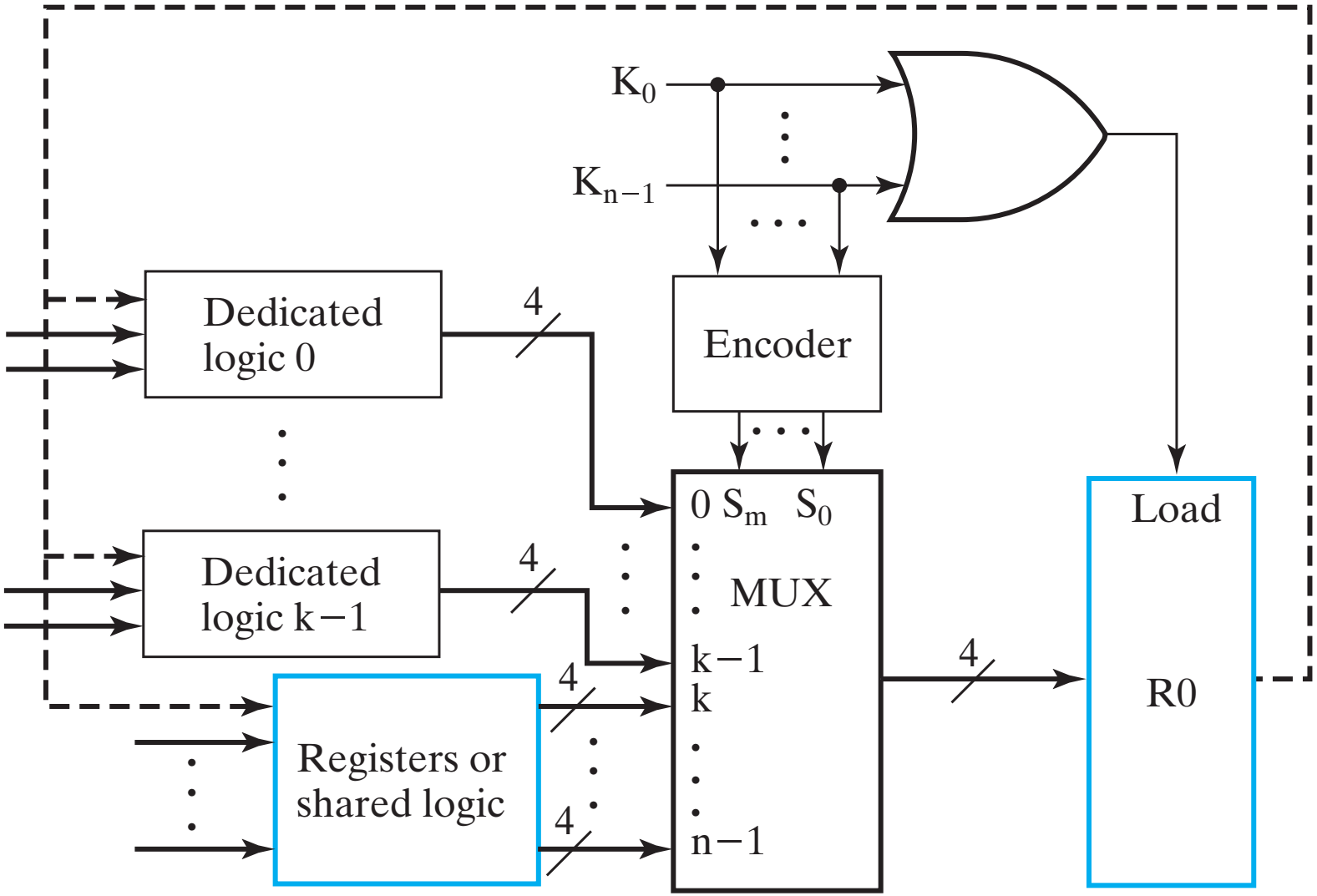
Type	Symbolic designation	Eight-bit examples: Source *R2*	Eight-bit examples: After shift: Destination *R1*
shift left	R1← sl R2	10011110	00111100
shift right	R1← sr R2	11100101	01110010

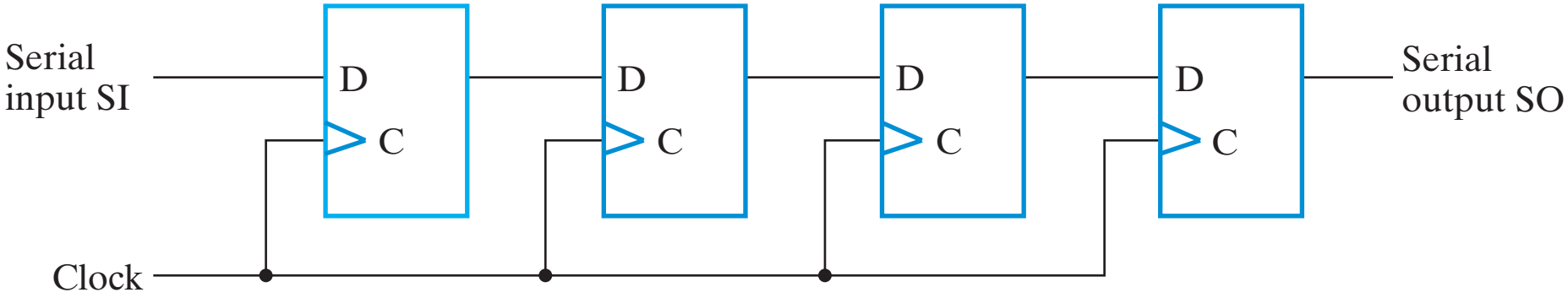


(a) Block diagram

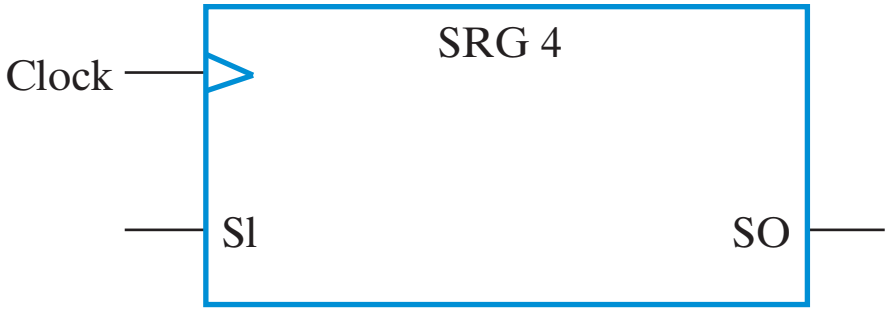


(b) Detailed logic

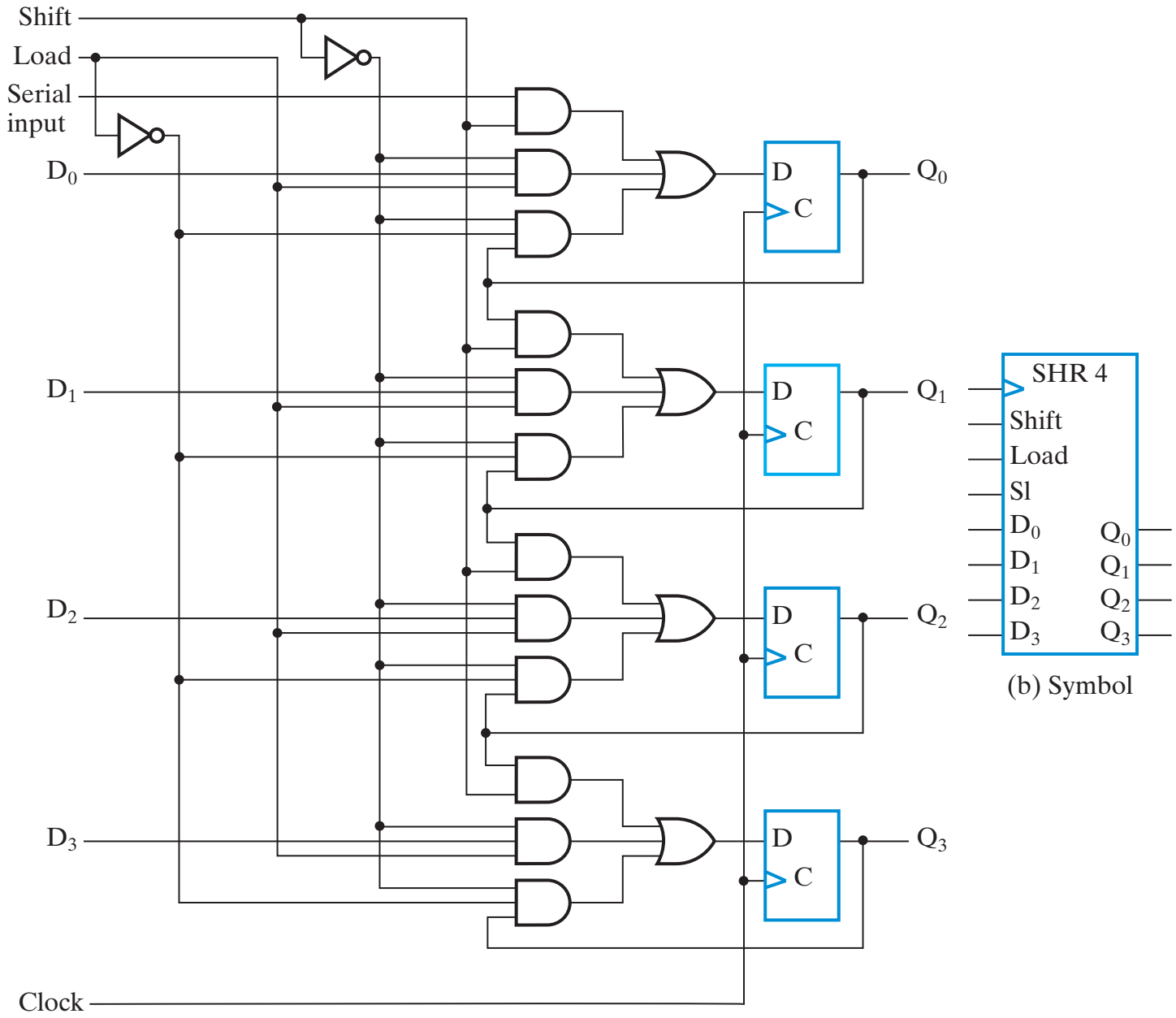




(a) Logic diagram



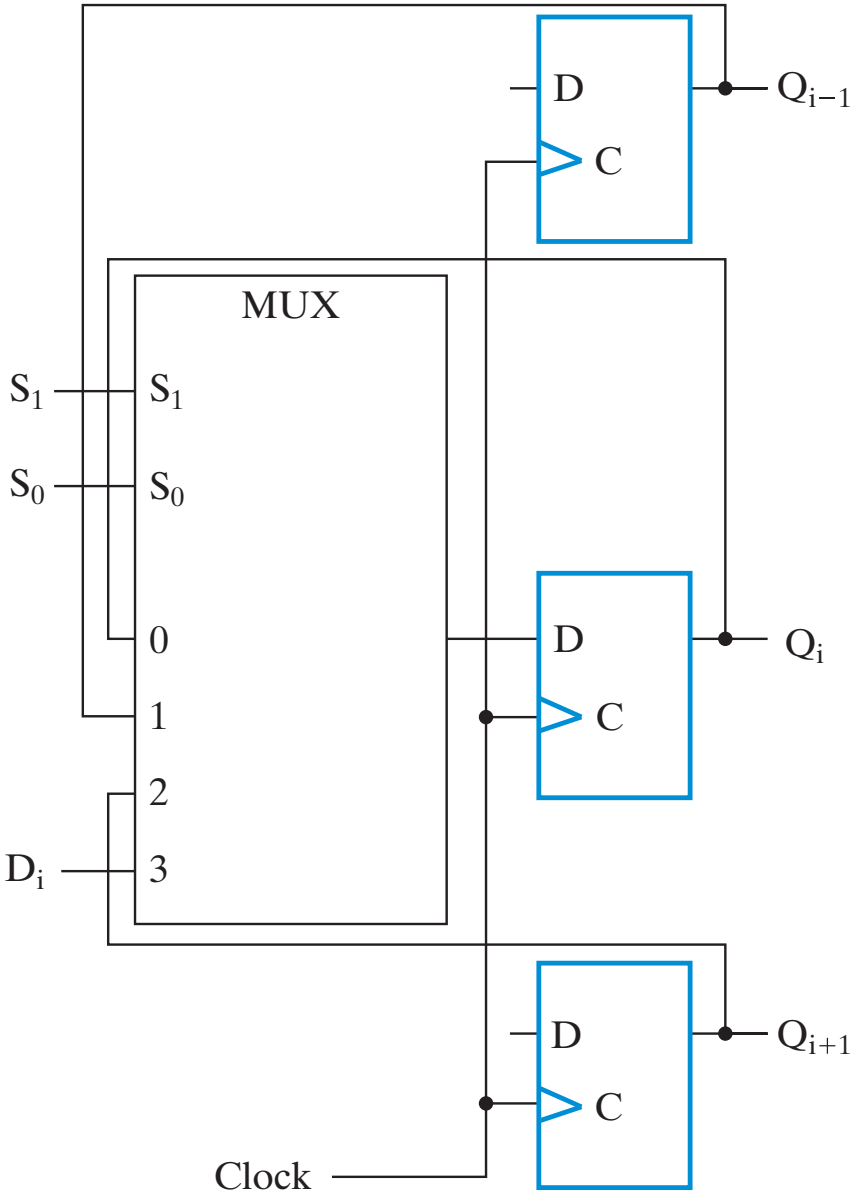
(b) Symbol



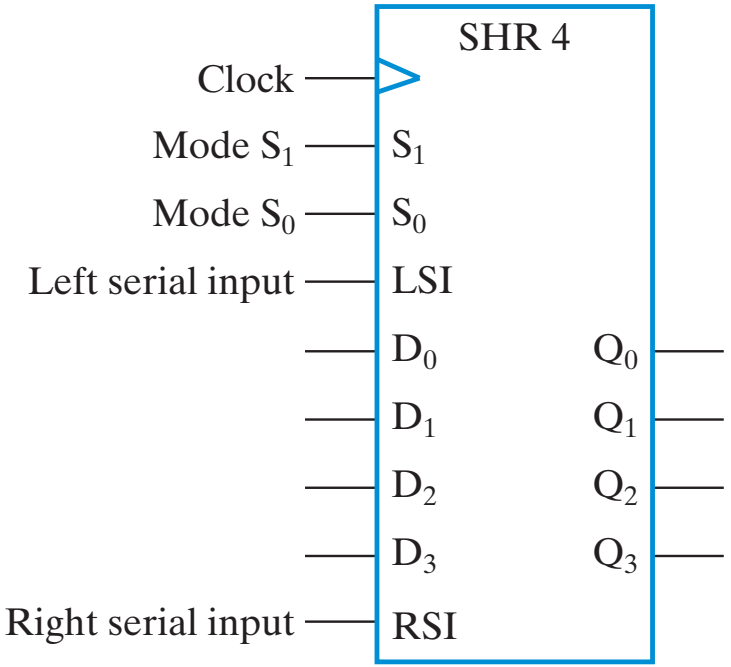
(b) Symbol

TABLE 7-6
Function Table for the Register of Figure 7-10

Shift	Load	Operation
0	0	No change
0	1	Load parallel data
1	X	Shift down from Q_0 to Q_3



(a) Logic diagram of one typical stage



(b) Symbol

TABLE 7-7
Function Table for the Register of Figure 7-7

Mode control		Register Operation
S_1	S_0	
0	0	No change
0	1	Shift down
1	0	Shift up
1	1	Parallel load

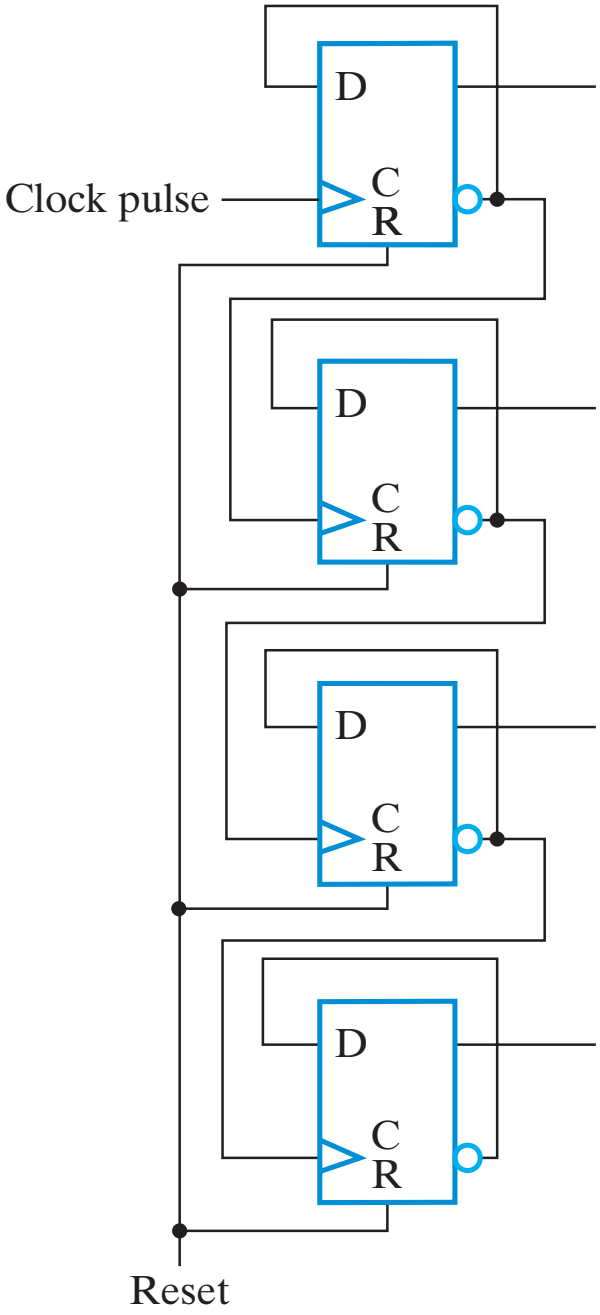
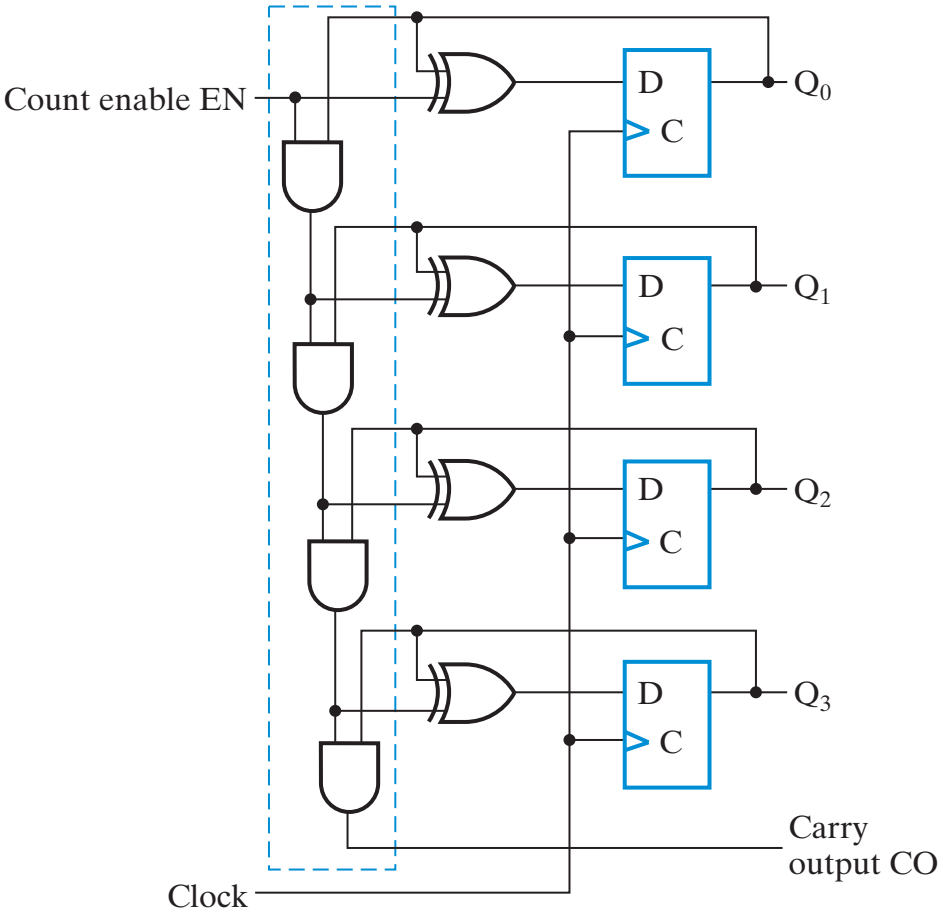
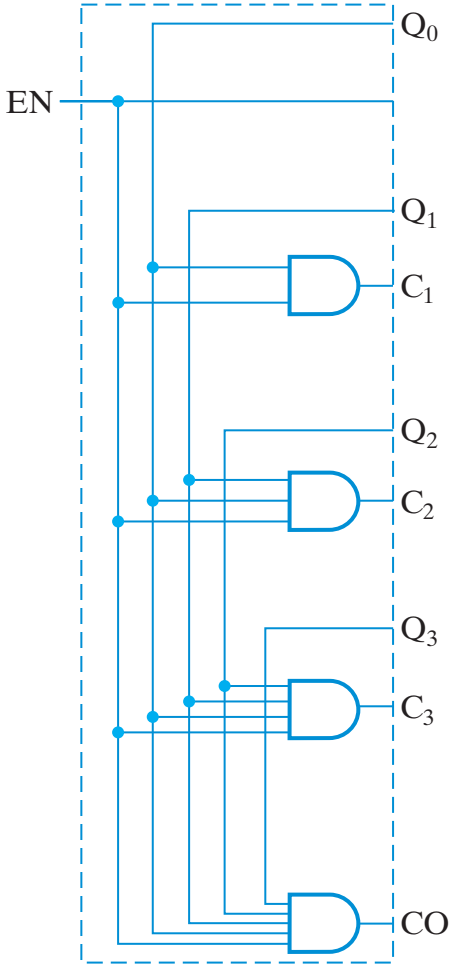


TABLE 7-8
Counting Sequence of Binary Counter

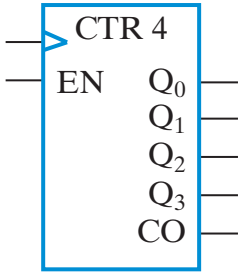
Upward Counting Sequence				Downward Counting Sequence			
Q_3	Q_2	Q_1	Q_0	Q_3	Q_2	Q_1	Q_0
0	0	0	0	1	1	1	1
0	0	0	1	1	1	1	0
0	0	1	0	1	1	0	1
0	0	1	1	1	1	0	0
0	1	0	0	1	0	1	1
0	1	0	1	1	0	1	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	0	0
1	0	0	0	0	1	1	1
1	0	0	1	0	1	1	0
1	0	1	0	0	1	0	1
1	0	1	1	0	1	0	0
1	1	0	0	0	0	1	1
1	1	0	1	0	0	1	0
1	1	1	0	0	0	0	1
1	1	1	1	0	0	0	0



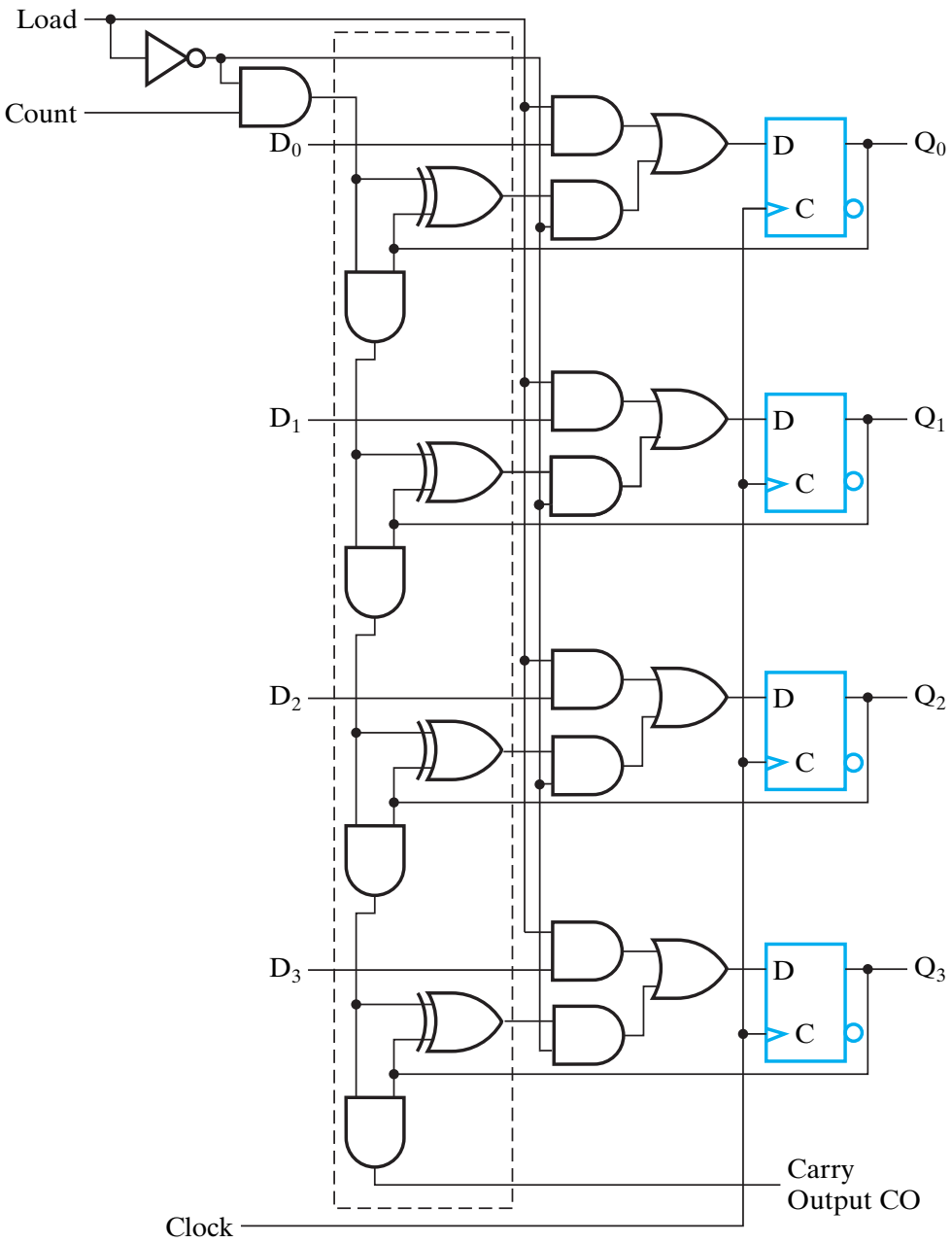
(a) Logic diagram-serial gating



(b) Logic diagram-parallel gating



(c) Symbol



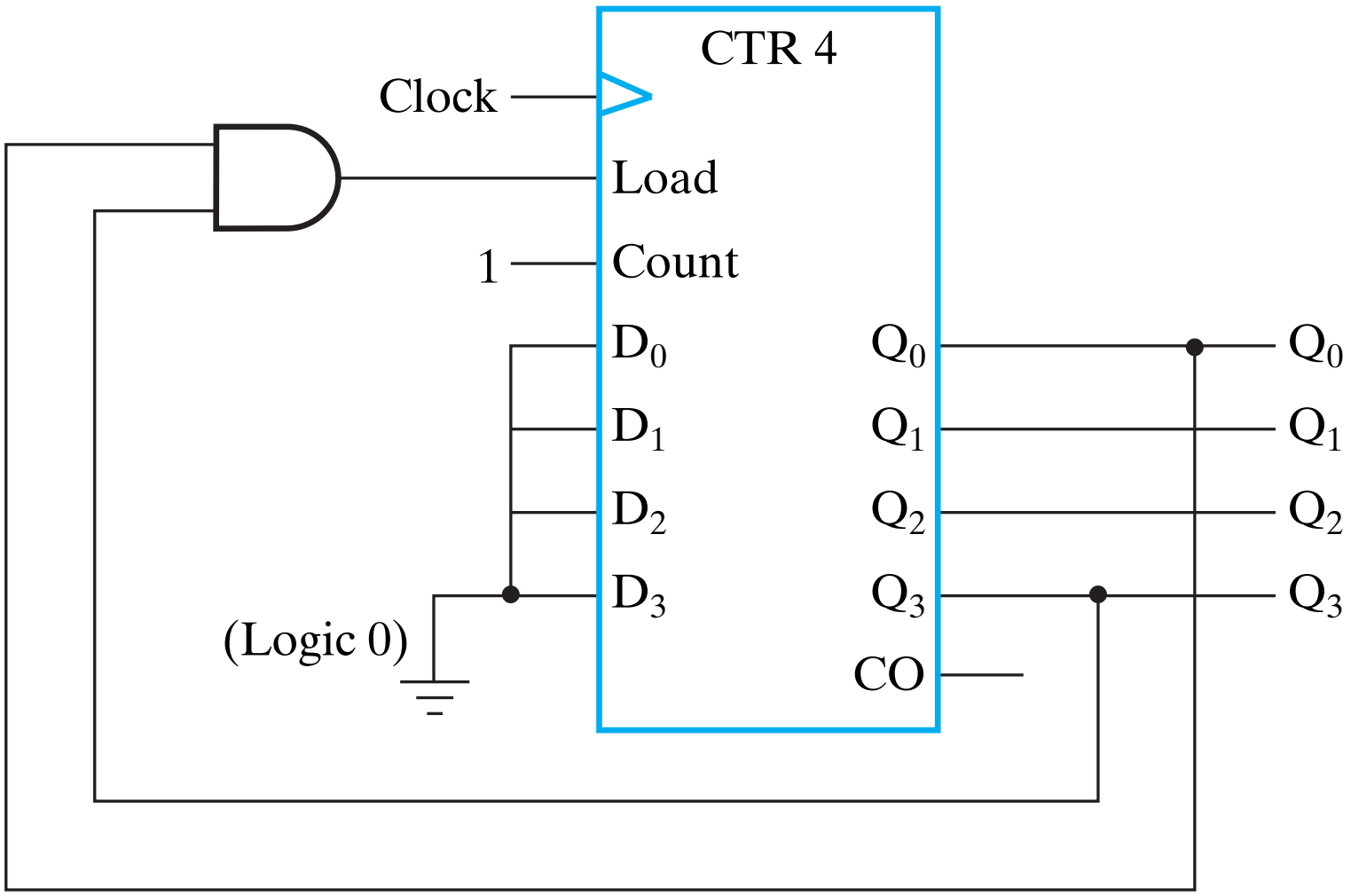
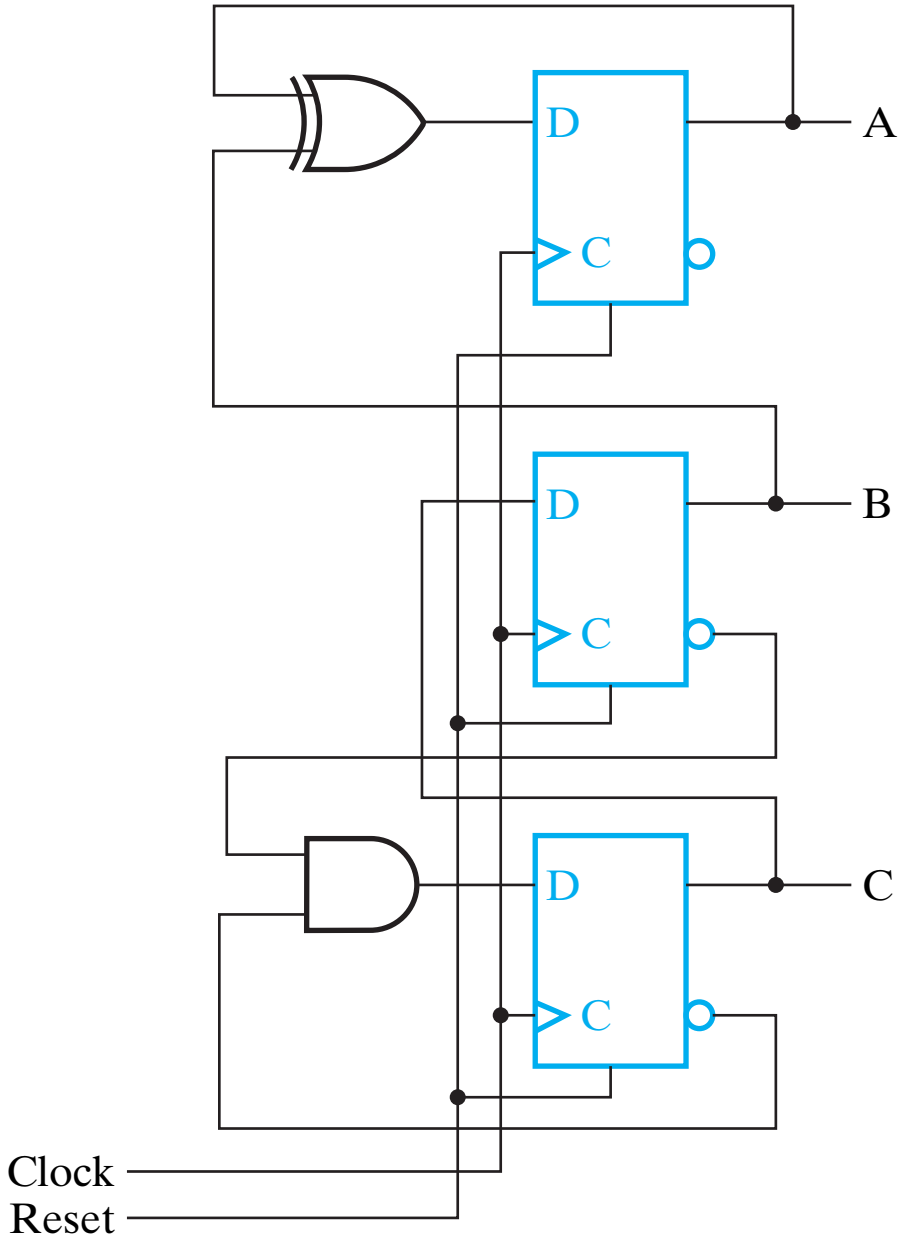


TABLE 7-9
State Table and Flip-Flop Inputs for BCD Counter

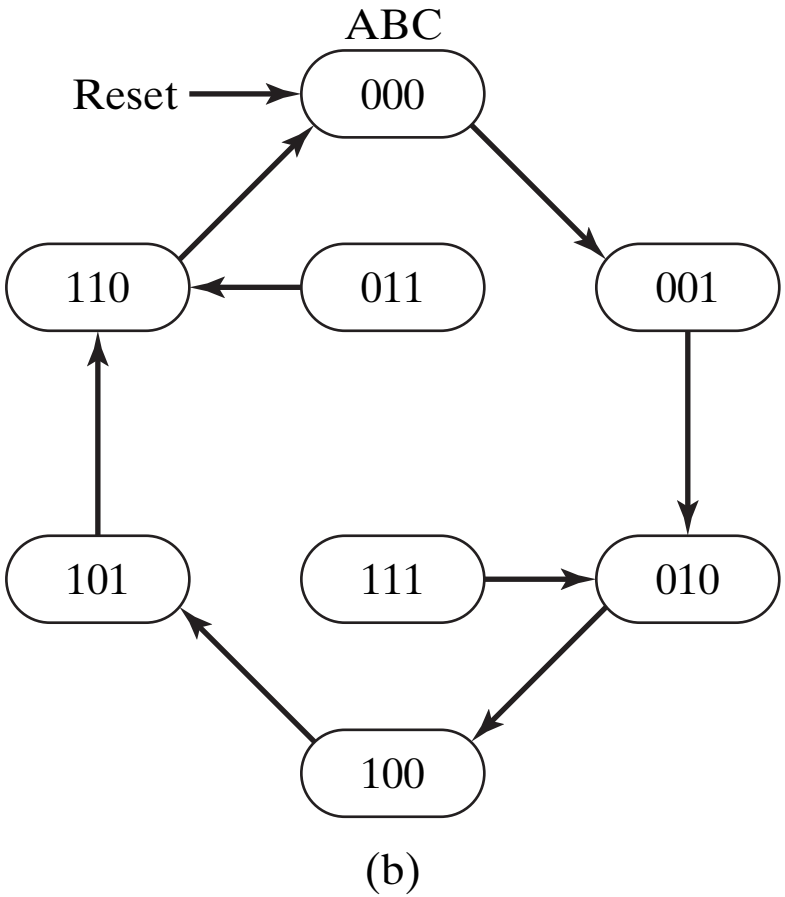
Present State				Next State				Output
Q_8	Q_4	Q_2	Q_1	$D_8 = Q_8(t+1)$	$D_4 = Q_4(t+1)$	$D_2 = Q_2(t+1)$	$D_1 = Q_1(t+1)$	Y
0	0	0	0	0	0	0	1	0
0	0	0	1	0	0	1	0	0
0	0	1	0	0	0	1	1	0
0	0	1	1	0	1	0	0	0
0	1	0	0	0	1	0	1	0
0	1	0	1	0	1	1	0	0
0	1	1	0	0	1	1	1	0
0	1	1	1	1	0	0	0	0
1	0	0	0	1	0	0	1	0
1	0	0	1	0	0	0	0	1

TABLE 7-10
State Table and Flip-Flop Inputs for Counter

Present State			Next State		
A	B	C	DA = A(t+1)	DB = B(t+1)	DC= C(t+1)
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



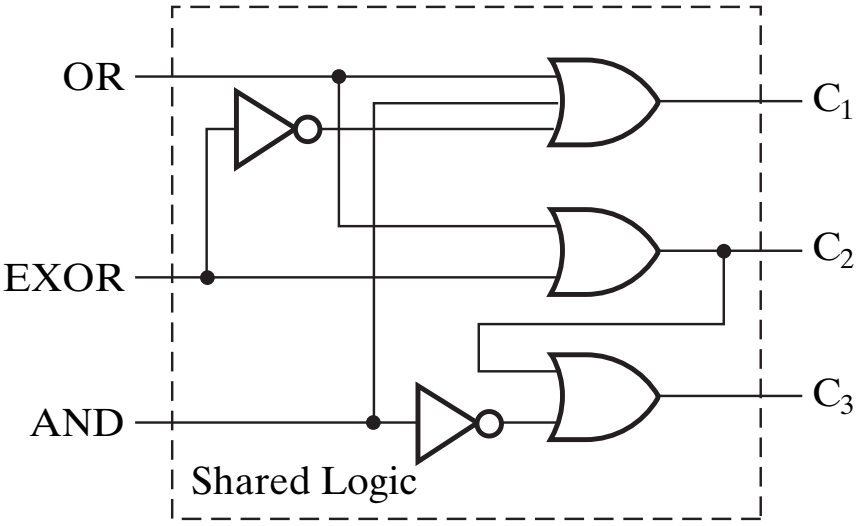
(a)



(b)

TABLE 7-11
State Table and Flip-Flop Inputs for Example 7-1

Present State A	Next State A(t + 1)						
	(AND = 0) ·(EXOR = 0) ·(OR = 0)	(OR = 1) ·(B = 0)	(OR = 1) ·(B = 1)	(EXOR = 1) ·(B = 0)	(EXOR = 1) ·(B = 1)	(AND = 1) ·(B = 0)	(AND = 1) ·(B = 1)
0	0	0	1	0	1	0	0
1	1	1	1	1	0	0	1



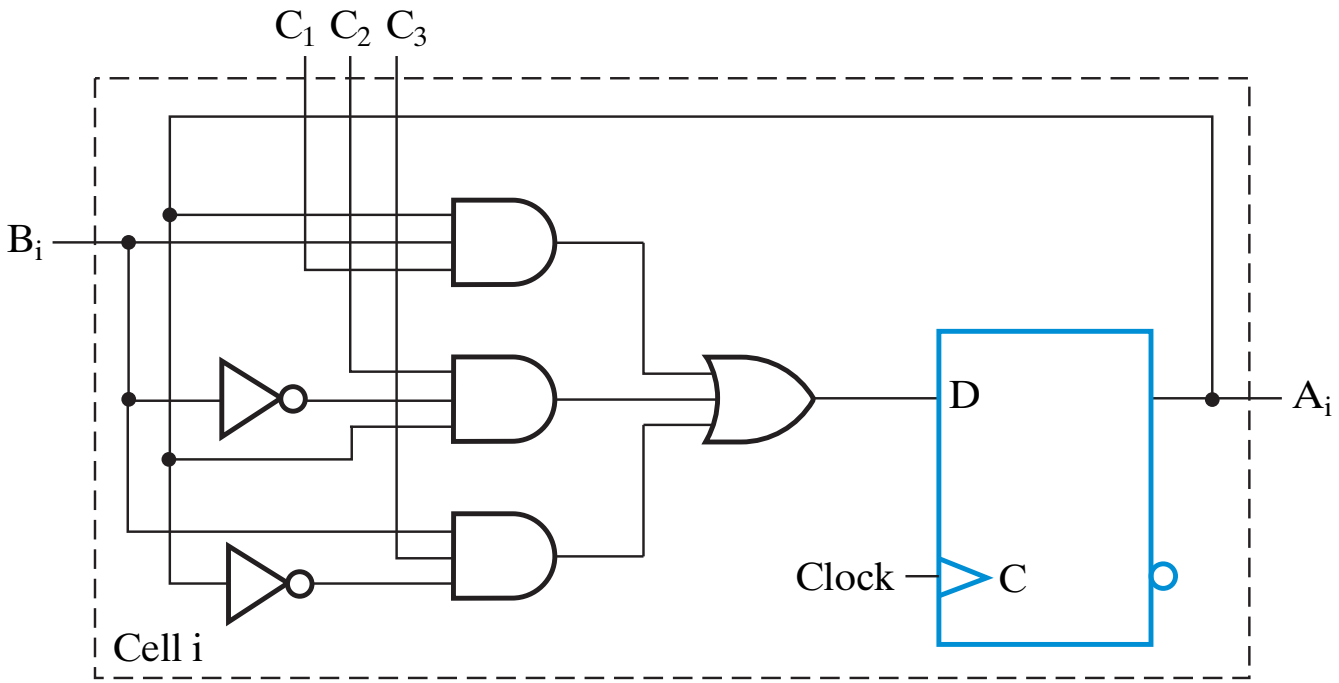
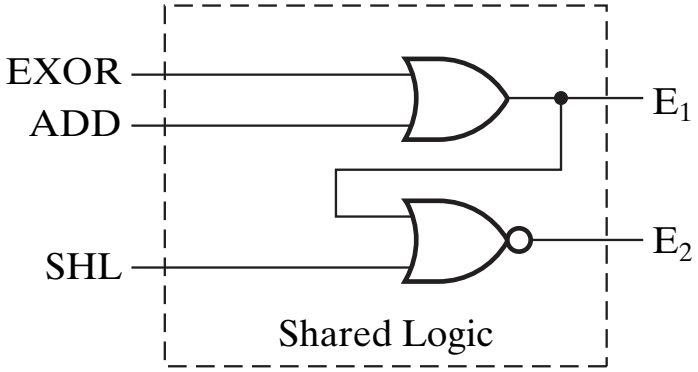
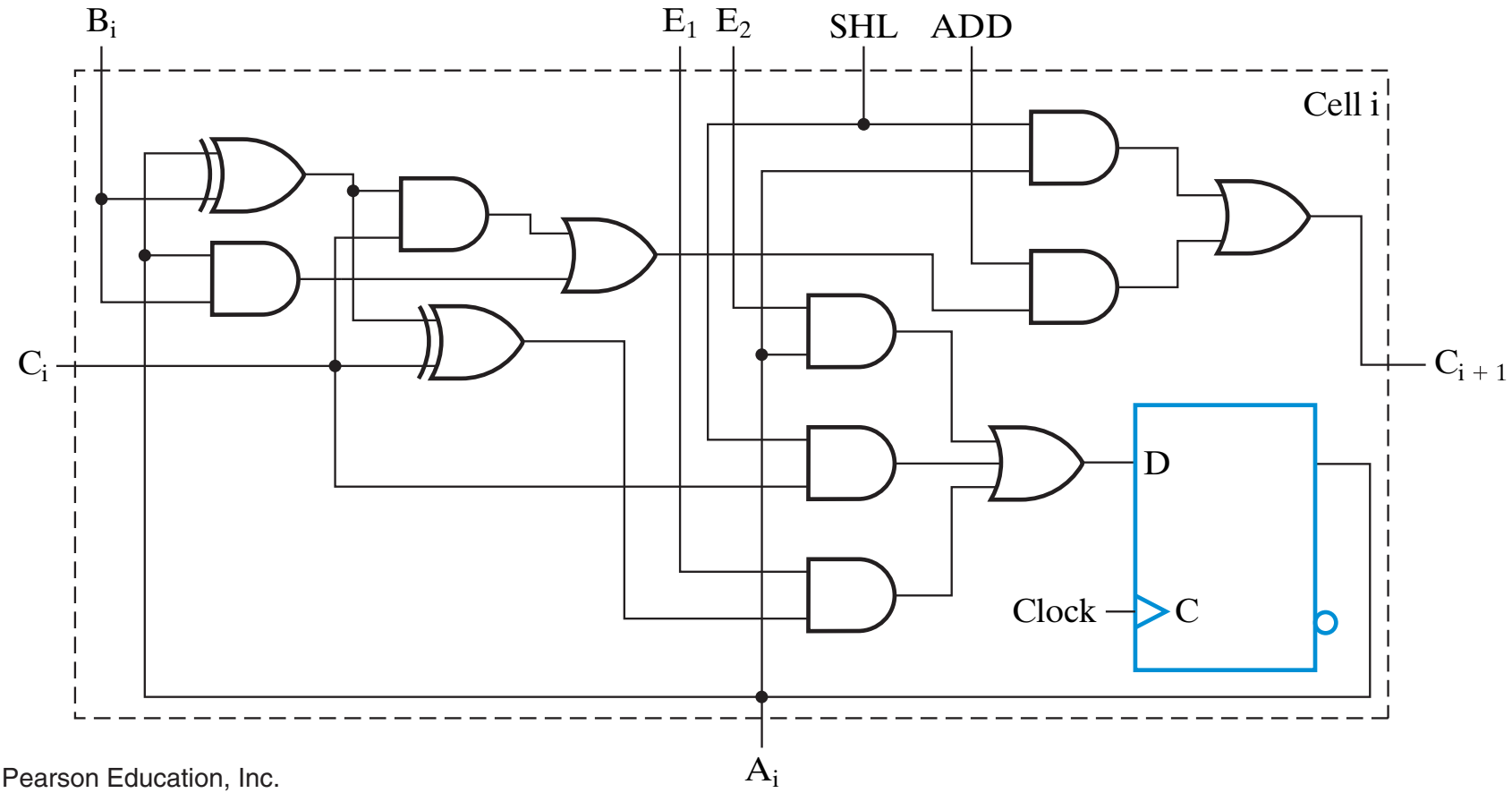
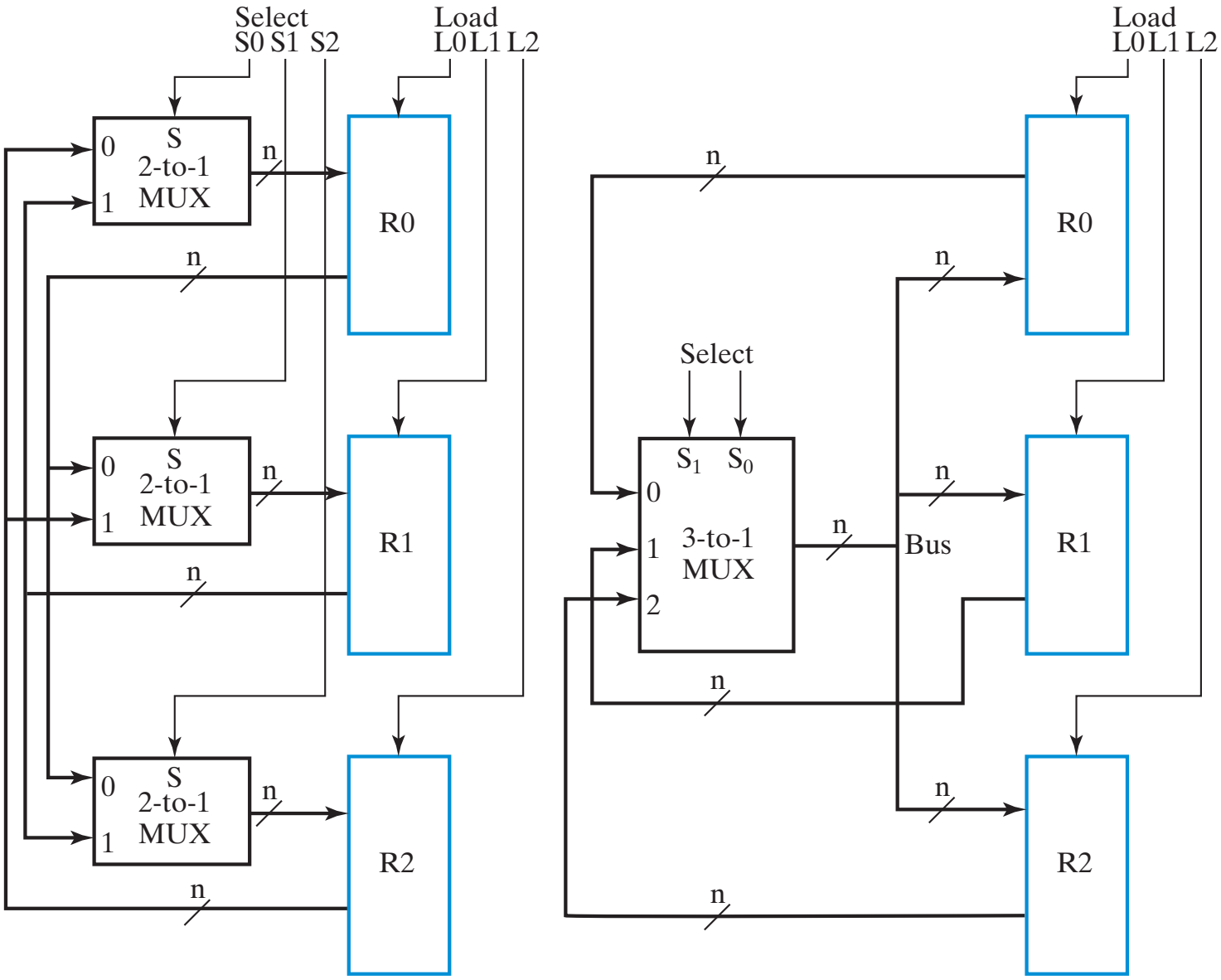


TABLE 7-12
State Table and Flip-Flop Inputs for Register-Cell Design in Example 7-2

Present State A_i	Inputs	Next State $A_i(t+1)$/Output C_{i+1}											
	SHL = 0	SHL = 1	1	1	1	EXOR = 1	1	ADD = 1	1	1	1
	EXOR = 0	B_i = 0	0	1	1	B_i = 0	1	B_i = 0	0	1	1
	ADD = 0	C_i = 0	1	0	1			C_i = 0	1	0	1
0	0/X	0/0	1/0	0/0	1/0	0/X	1/X	0/0	1/0	1/0	0/1
1	1/X	0/1	1/1	0/1	1/1	1/X	0/X	1/0	0/1	0/1	1/1





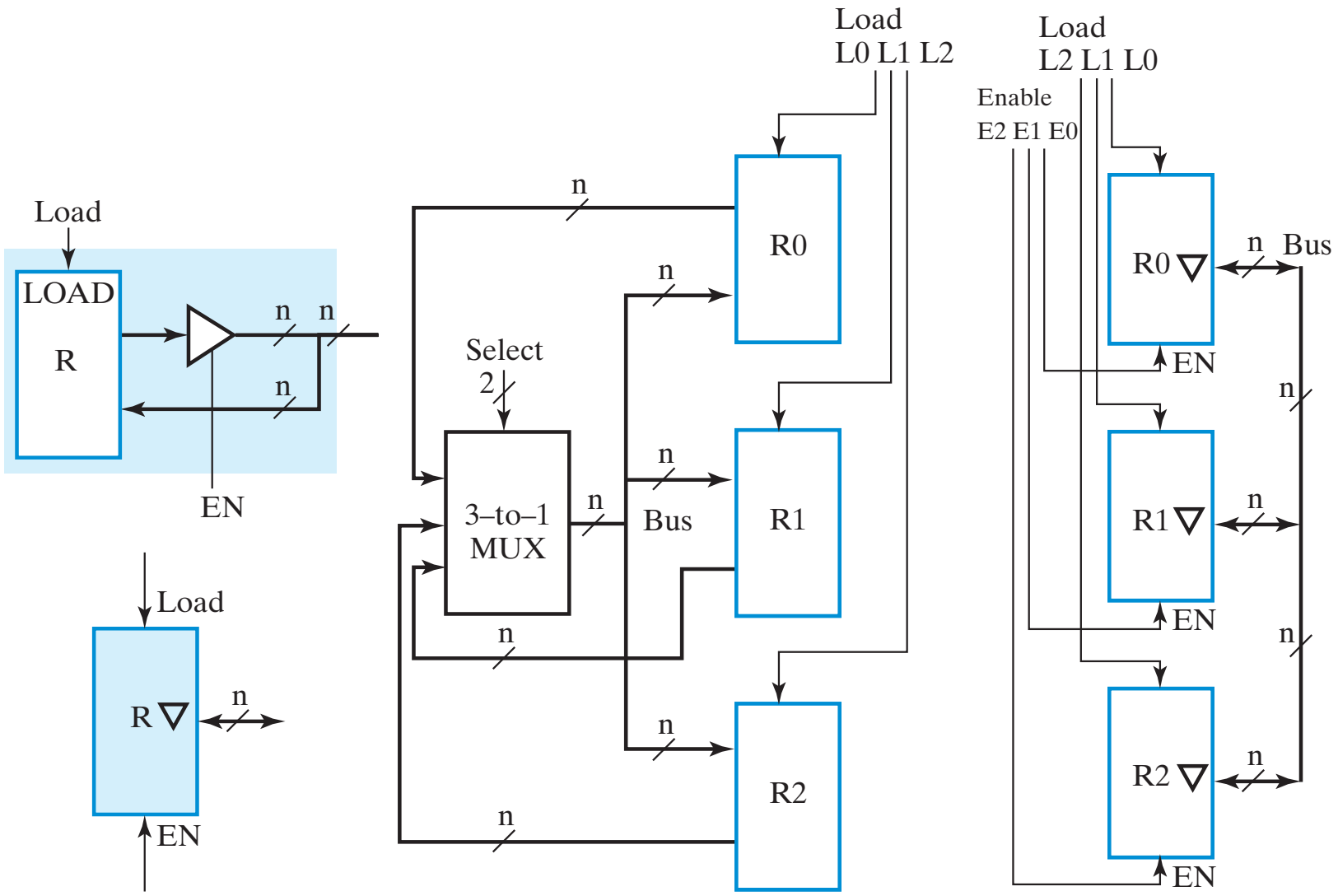


(a) Dedicated multiplexers

(b) Single bus

TABLE 7-13
Examples of Register Transfers Using the Single Bus in Figure 7-19(b)

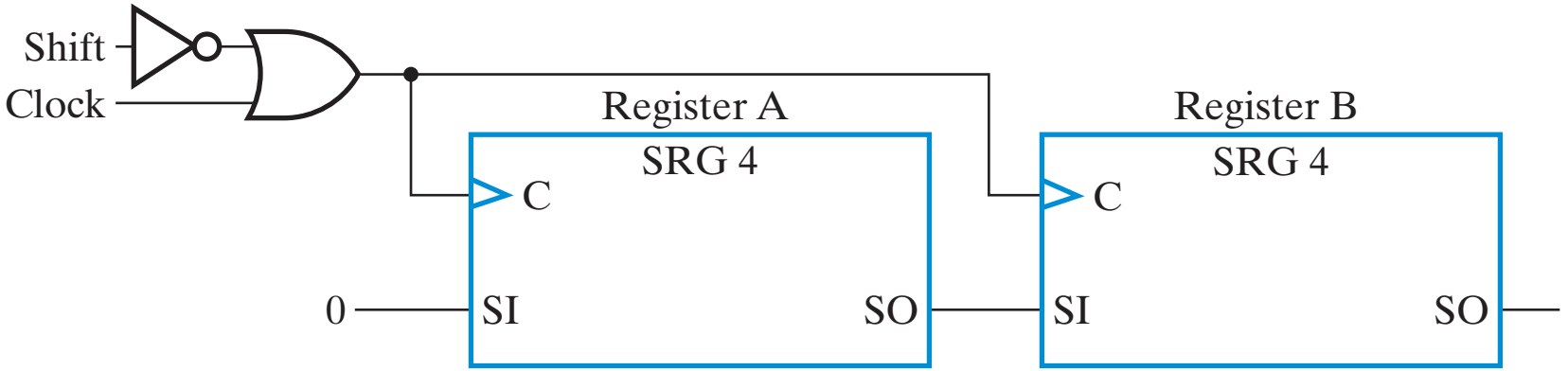
	Select		Load		
Register Transfer	**S1**	**S0**	**L2**	**L1**	**L0**
$R0 \leftarrow R2$	1	0	0	0	1
$R0 \leftarrow R1,\ R2 \leftarrow R1$	0	1	1	0	1
$R0 \leftarrow R1,\ R1 \leftarrow R0$	Impossible				



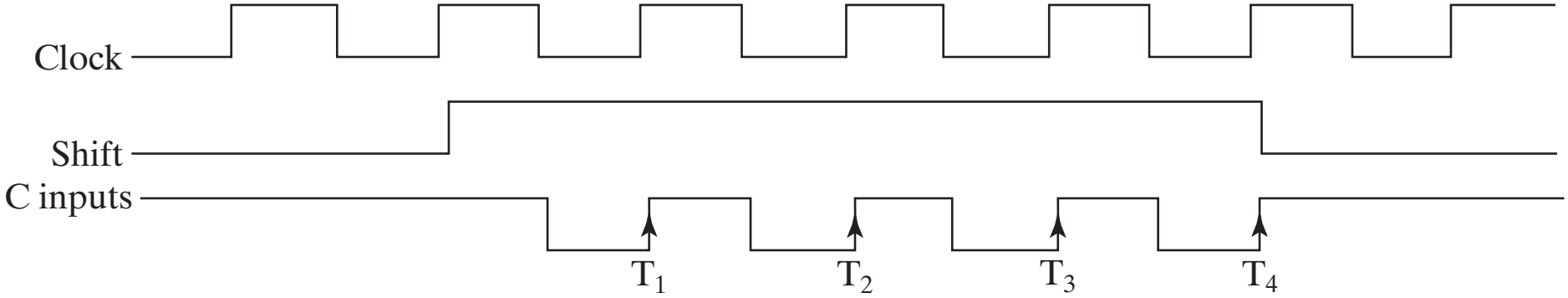
(a) Register with bidirectional input–output lines and symbol

(b) Multiplexer bus

(c) Three-state bus using registers with bidirectional lines



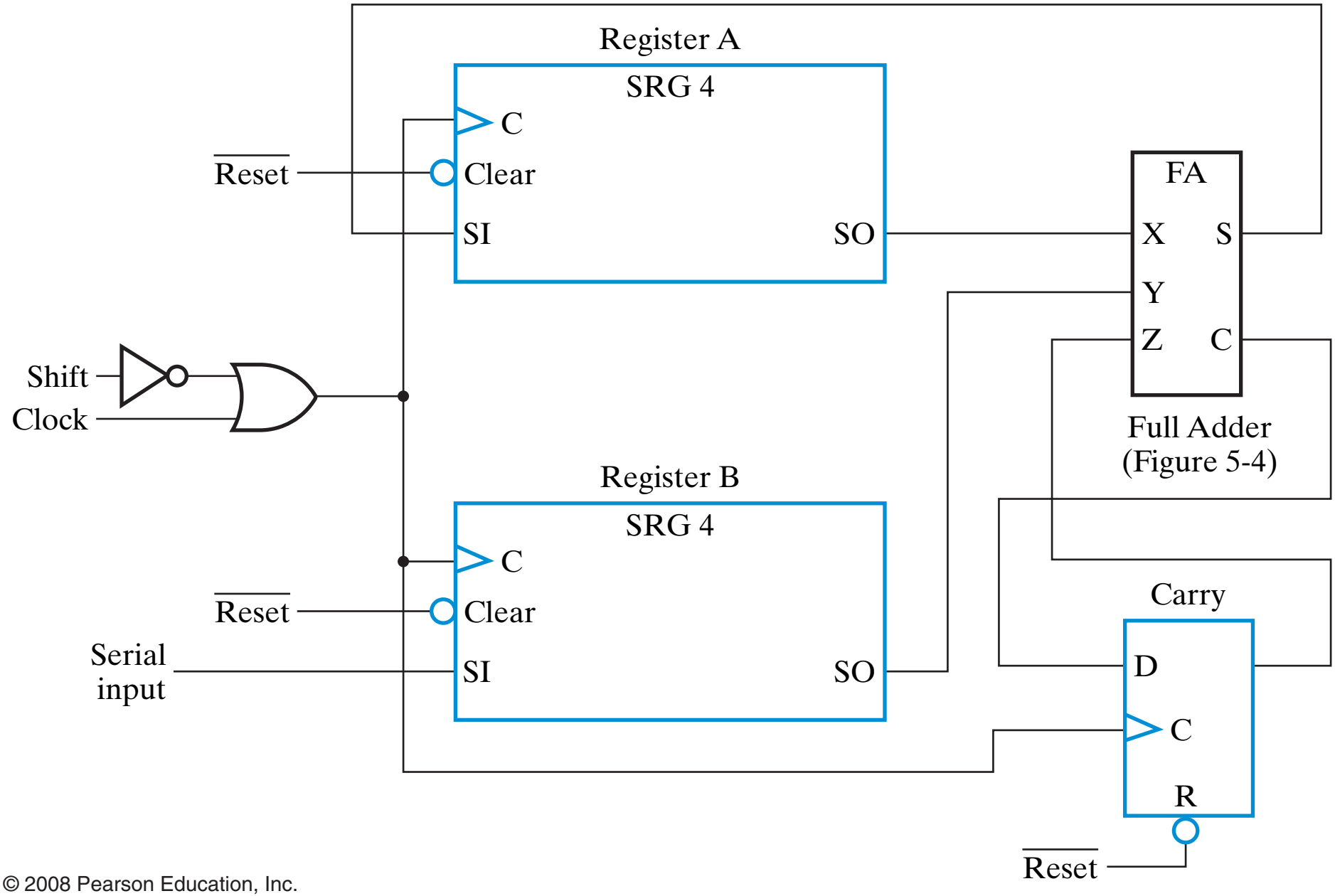
(a) Block diagram

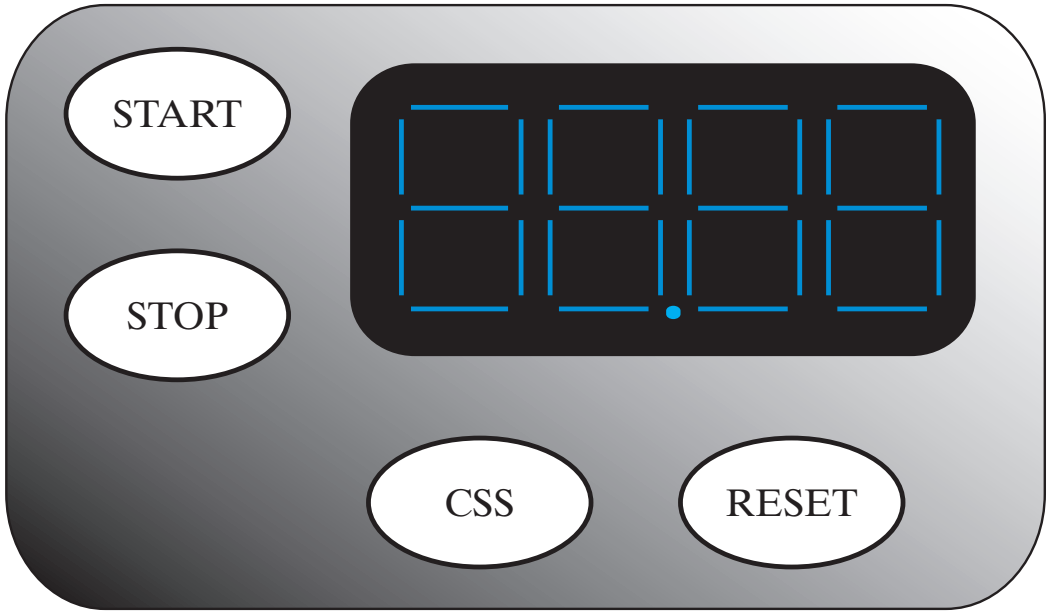


(b) Timing diagram

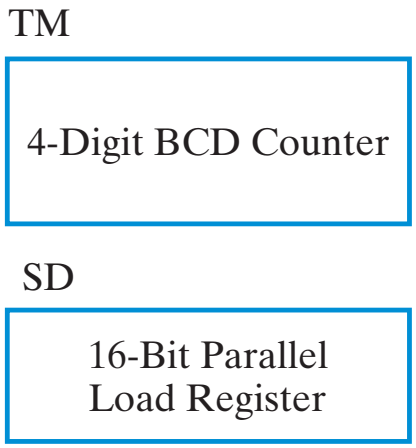
TABLE 7-14
Example of Serial Transfer

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





(a)



(b)

TABLE 7-15
Inputs, Outputs, and Registers of the DashWatch

Symbol	Function	Type
START	Initialize timer to 0 and start timer	Control input
STOP	Stop timer and display timer	Control input
CSS	Compare, store and display shortest dash time	Control input
RESET	Set shortest value to 10011001	Control input
B_1	Digit 1 data vector a, b, c, d, e, f, g to display	Data output vector
B_0	Digit 0 data vector a, b, c, d, e, f, g to display	Data output vector
DP	Decimal point to display (= 1)	Data output
B_{-1}	Digit –1 data vector a, b, c, d, e, f, g to display	Data output vector
B_{-2}	Digit –2 data vector a, b, c, d, e, f, g to display	Data output vector
B	The 29-bit display input vector (B_1, B_0, DP, B_{-1}, B_{-2})	Data output vector
TM	4-Digit BCD counter	16-Bit register
SP	Parallel load register	16-Bit register

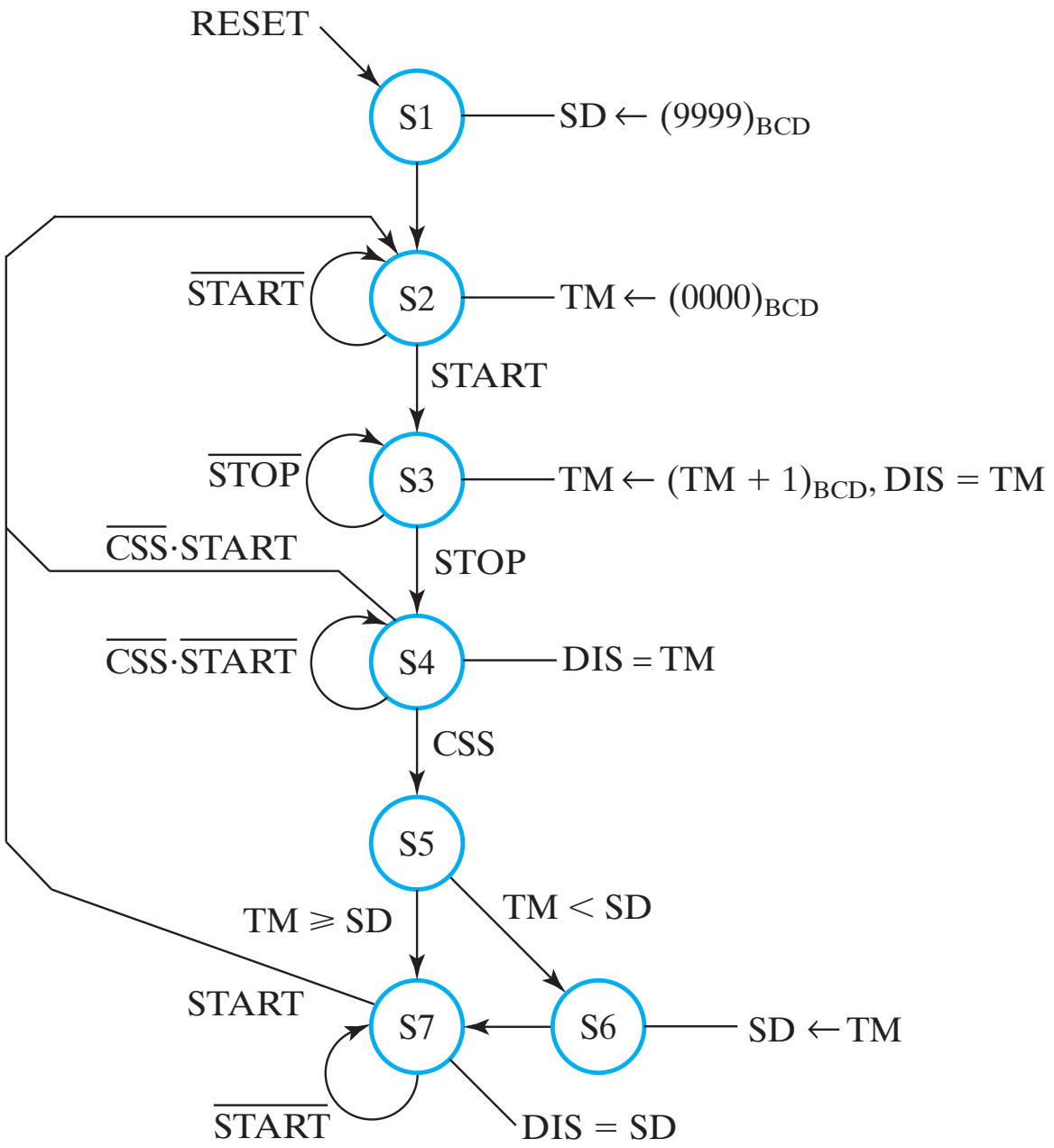
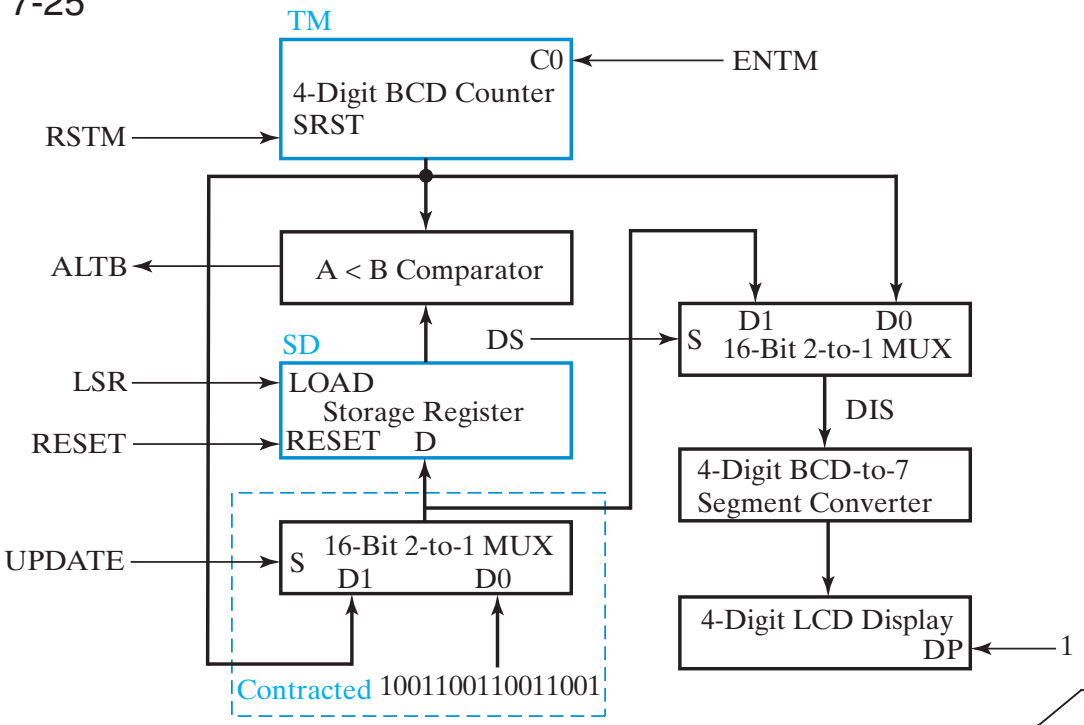
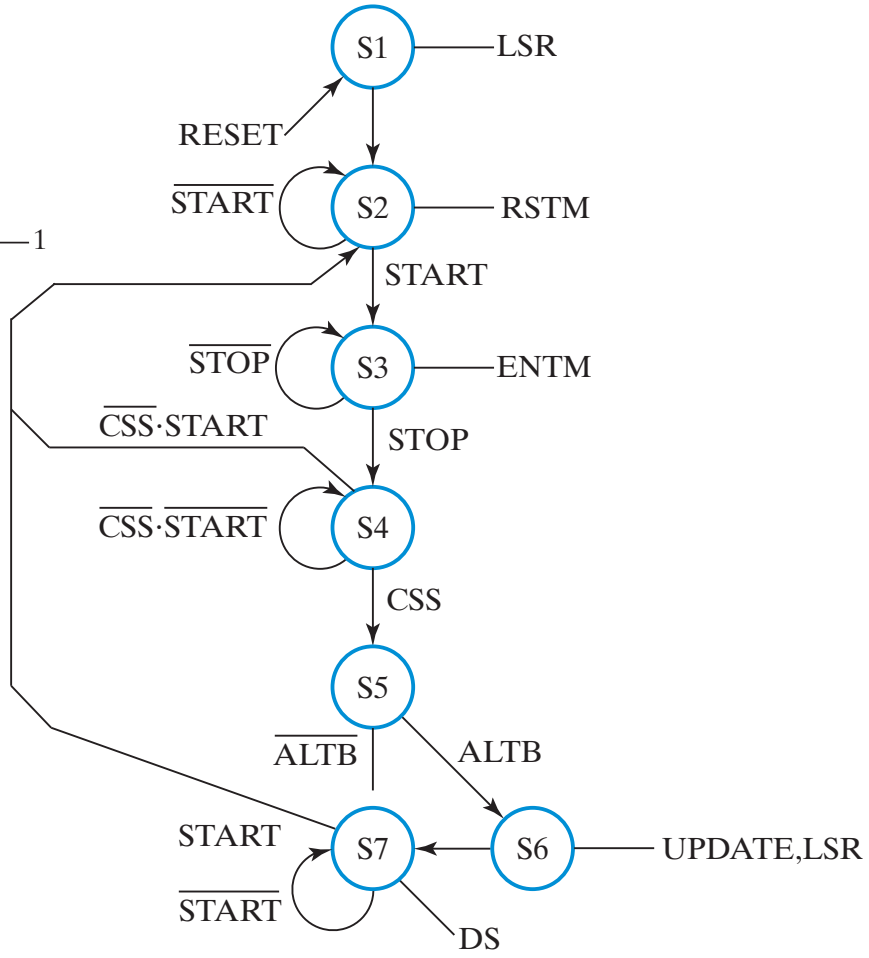


TABLE 7-16
Datapath Output Actions and Status Generation with Control and Status Signals

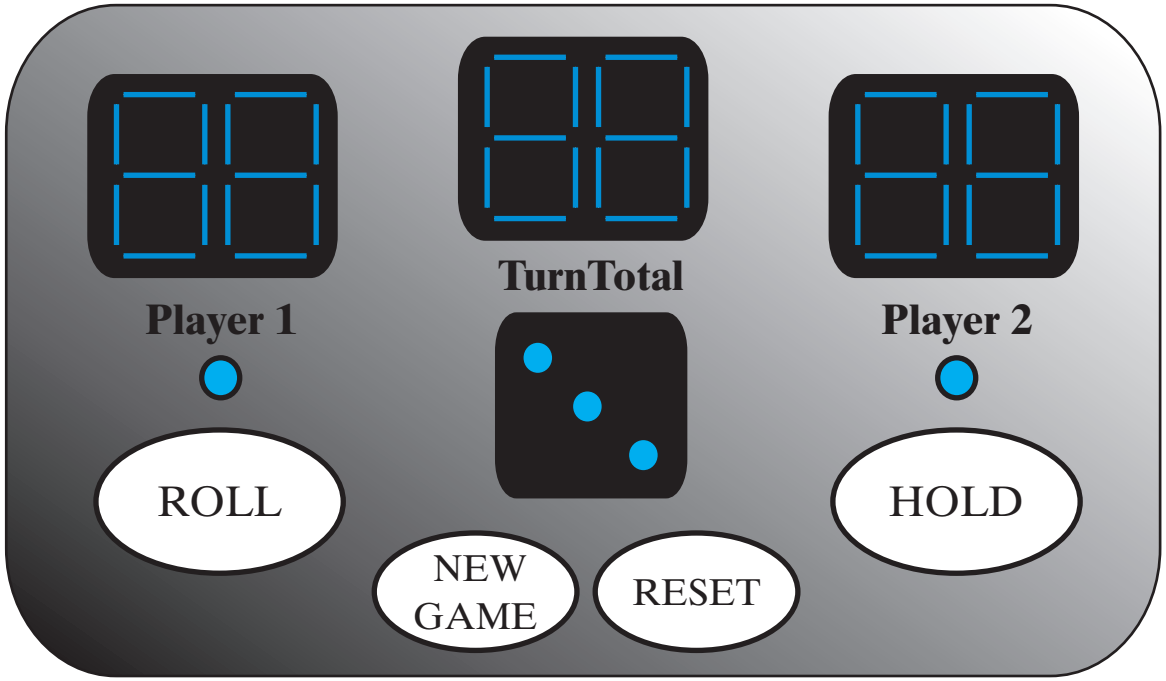
Action or Status	Control or Status Signals	Meaning for Values 1 and 0
$TM \leftarrow (0000)_{BCD}$	RSTM	1: Reset TM to 0 (synchronous reset) 0: No reset of TM
$TM \leftarrow (TM + 1)_{BCD}$	ENTM	1: BCD count up TM by 1, 0: hold TM value
$SD \leftarrow (9999)_{BCD}$	UPDATE LSR	0: Select 1001100110011001 for loading SD 1: Enable load SD, 0: disable load SD
$SD \leftarrow TM$	UPDATE LSR	1: Select TM for loading SD Same as above
DIS = TM DIS = SD	DS	0: Select TM for DIS 1: Select SD for DIS
TM < SD TM ≥ SD	ALTB	1: TM less than SD 0: TM greater than or equal to SD



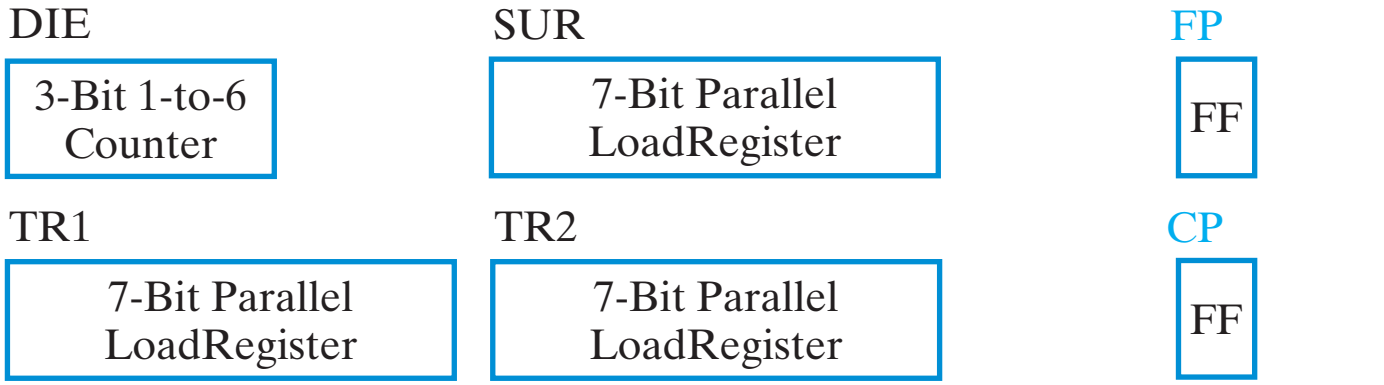
(a)



(b)



(a)



(b)

TABLE 7-17
Inputs, Outputs, and Registers of PIG

Symbol	Name/Function	Type
ROLL	1: Starts die rolling, 0: Stops die rolling	Control input
HOLD	1: Ends player turn, 0: Continues player turn.	Control input
NEW_GAME	1: Starts new game, 0: Continues current game	Control input
RESET	1: Resets game to INIT state, 0: No action	Control input
DDIS	7-bit LED die display array	Data output vector
SUB	14-bit 7-segment pair (a, b, c, d, e, f, g) to Turn Total display	Data output vector
TP1	14-bit 7-segment pair (a, b, c, d, e, f, g) to Player 1 display	Data output vector
TP2	14-bit 7-segment pair (a, b, c, d, e, f, g) to Player 2 display	Data output vector
P1	1: Player 1 LED on, 0: Player 1 LED off	Data output
P2	1: Player 2 LED on, 0: Player 2 LED off	Data output
DIE	Die value—Specialized counter to count 1,...,6,1,...	3-bit data register
SUR	Subtotal for active player—parallel load register	7-bit data register
TR1	Total for Player 1—parallel load register	7-bit data register
TR2	Total for Player 2—parallel load register	7-bit data register
FP	First player—flip-flop 0: Player 1, 1: Player 2	1-bit control register
CP	Current player—flip-flop 0: Player 1, 1: Player 2	1-bit control register

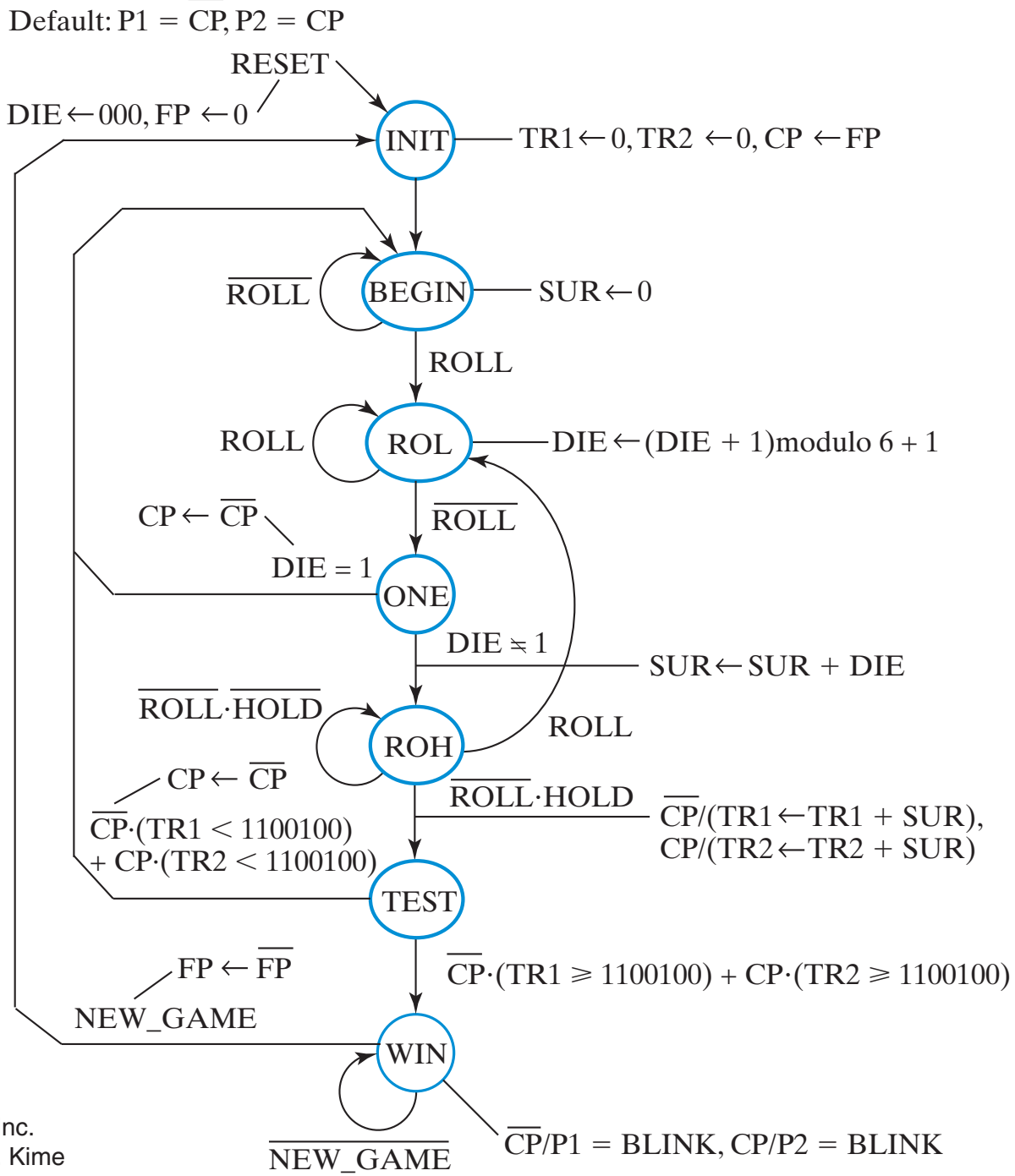
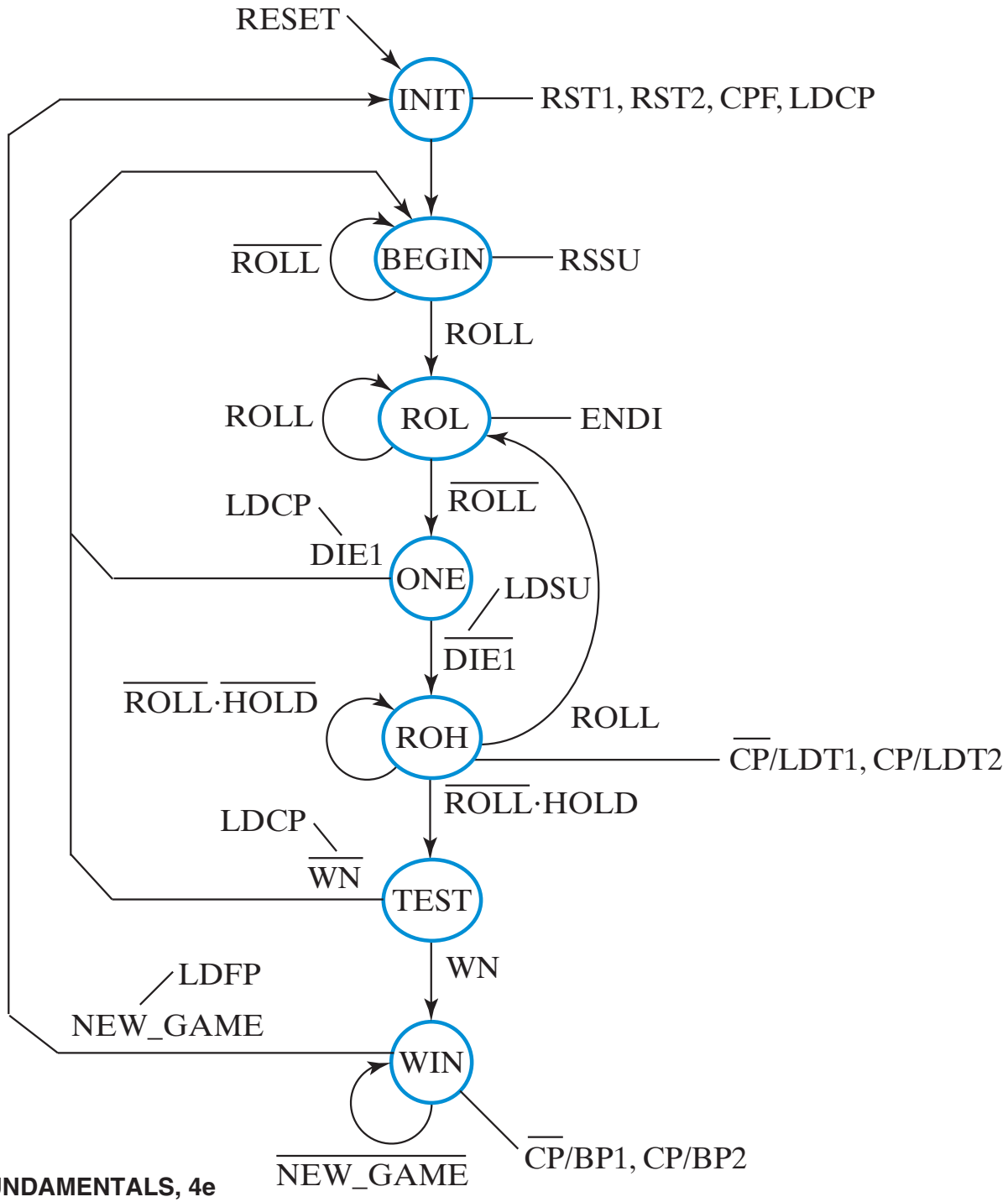
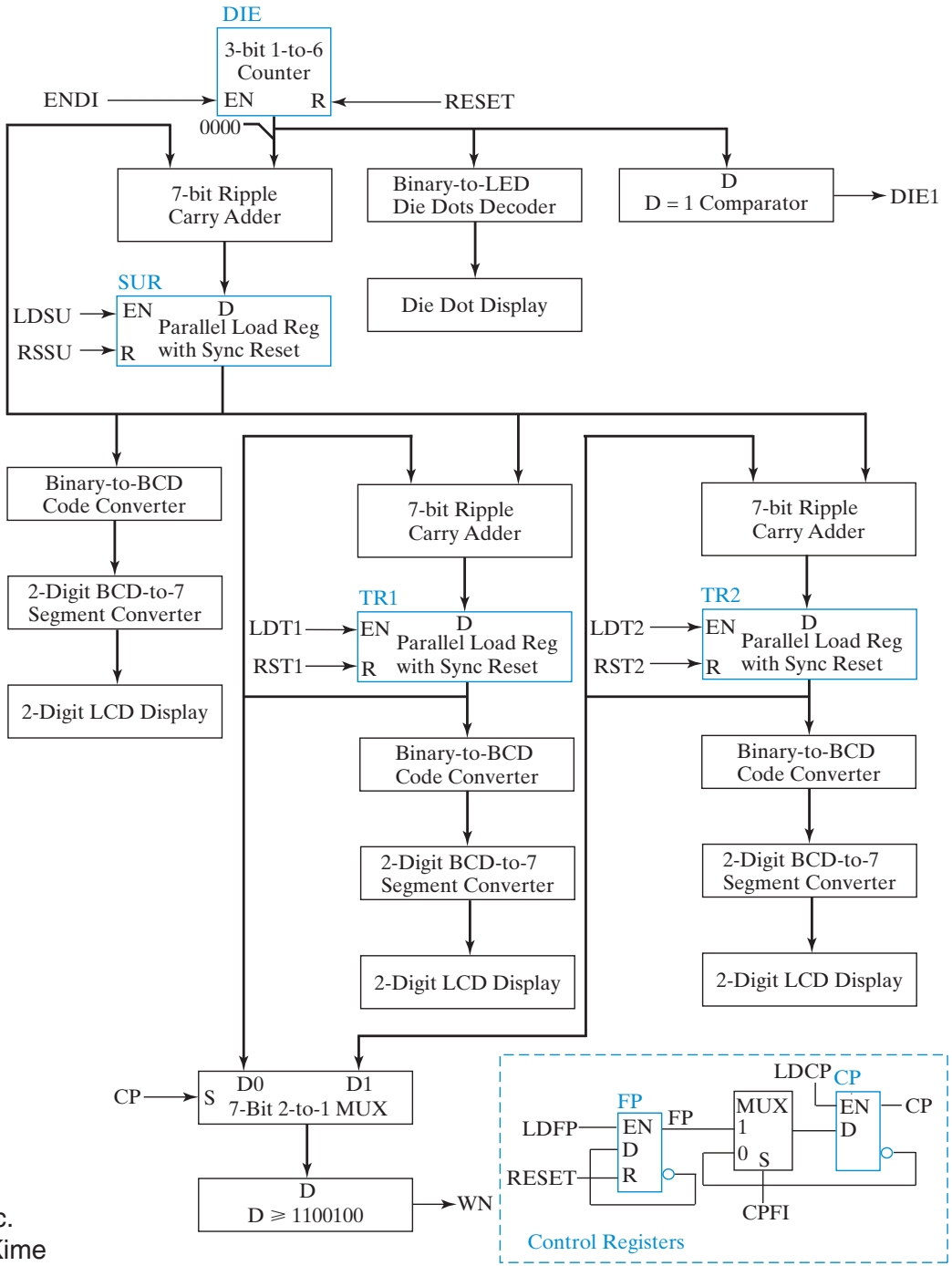


TABLE 7-18
Datapath Output Actions and Control and Status Signals for PIG

Action or Status	Control or Status Signals	Meaning for Values 1 and 0
TR1 ← 0 TR1 ← TR1 + SUR	RST1 LDT1	1: Reset TR1 (synchronous reset), 0: No action 1: Add SUR to TR1, 0: No action
TR2 ← 0 TR2 ← TR2 + SUR	RST2 LDT2	1: Reset TR2 (synchronous reset), 0: No action 1: Add SUR to TR2, 0: No action
SUR ← 0 SUR ← SUR + DIE	RSSU LDSU	1: Reset SUR (synchronous reset), 0: No action 1: Add DIE to SUR, 0: No action
DIE ← 000 if (DIE = 110) DIE ← 001 else DIE ← DIE + 1	RESET ENDI	1: Reset DIE to 000 (asynchronous reset) 1: Enable DIE to increment, 0: Hold DIE value
P1 = BLINK	BP1	1: Connect P1 to BLINK, 0: Connect P1 to 1
P2 = BLINK	BP2	1: Connect P2 to BLINK, 0: Connect P2 to 1
CP ← FP CP ← $\overline{CP}$	CPFI LDCP CPFI LDCP	1: Select FP for CP 1: Load CP, 0: No action 0: Select $\overline{CP}$ for CP 1: Load CP, 0: No action
FP ← 0 FP ← $\overline{FP}$	RESET FPI	Asynchronous reset 1: Invert FP, 0: Hold FP
DIE = 1 DIE ≠ 1	DIE1	1: DIE equal to 1 0: DIE not equal to 1
TR1 ≥ 1100100	CP WN	0: Select TR1 for ≥ 1100100 1: The selected TRi ≥ 1100100 0: The selected TRi < 1100100
TR2 ≥ 1100100	CP WN	1: Select TR2 for ≥ 1100100 1: The selected TRi ≥ 1100100 0: The selected TRi < 1100100





```
-- 4-bit Left Shift Register with Reset

library ieee;
use ieee.std_logic_1164.all;

entity srg_4_r is
  port(CLK, RESET, SI : in std_logic;
    Q : out std_logic_vector(3 downto 0);
    SO : out std_logic);
end srg_4_r;

architecture behavioral of srg_4_r is
signal shift : std_logic_vector(3 downto 0);
begin
process (RESET, CLK)
begin
  if (RESET = '1') then
    shift <= "0000";
  elsif (CLK'event and (CLK = '1')) then
    shift <= shift(2 downto 0) & SI;
  end if;
end process;
  Q <= shift;
  SO <= shift(3);
end behavioral;
```

```
-- 4-bit Binary Counter with Reset

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity count_4_r is
   port(CLK, RESET, EN : in std_logic;
        Q : out std_logic_vector(3 downto 0);
        CO : out std_logic);
end count_4_r;

architecture behavioral of count_4_r is
signal count : std_logic_vector(3 downto 0);
begin
process (RESET, CLK)
begin
   if (RESET = '1') then
      count <= "0000";
   elsif (CLK'event and (CLK = '1') and (EN = '1')) then
       count <= count + "0001";
   end if;
end process;
Q <= count;
CO <= '1' when count = "1111" and EN = '1' else '0';
end behavioral;
```

```
// 4-bit Left Shift Register with Reset

module srg_4_r_v (CLK, RESET, SI, Q,SO);
   input CLK, RESET, SI;
   output [3:0] Q;
   output SO;

reg [3:0] Q;

assign SO = Q[3];

always@(posedge CLK or posedge RESET)
begin
   if (RESET)
      Q <= 4'b0000;
   else
      Q <= {Q[2:0], SI};
end

endmodule
```

```
// 4-bit Binary Counter with Reset

module count_4_r_v (CLK, RESET, EN, Q, CO);
   input CLK, RESET, EN;
   output [3:0] Q;
   output CO;

reg [3:0] Q;

assign CO = (count == 4'b1111 && EN == 1'b1) ? 1 : 0;
always@(posedge CLK or posedge RESET)
   begin
   if (RESET)
     Q <= 4'b0000;
   else if (EN)
     Q <= Q + 4'b0001;
   end
endmodule
```