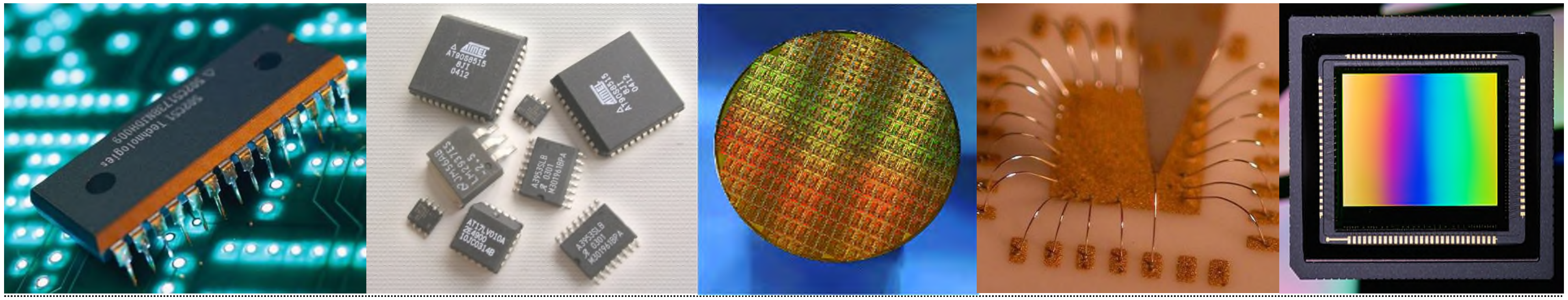


6. Registers and Counters

6.1 REGISTERS

Register- a group of binary cells suitable for holding binary information.

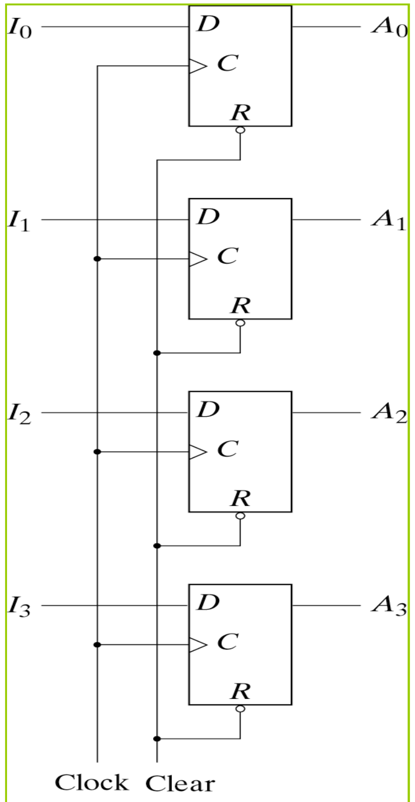


Fig. 6-1 4-Bit Register

- ❑Clock=1 ;input information transferred
- ❑Clock=0 ;unchanged
- ❑Clear=0 ;clearing the register to all 0’s prior to its clocked operation.

6.1 REGISTERS - Register with Parallel Load

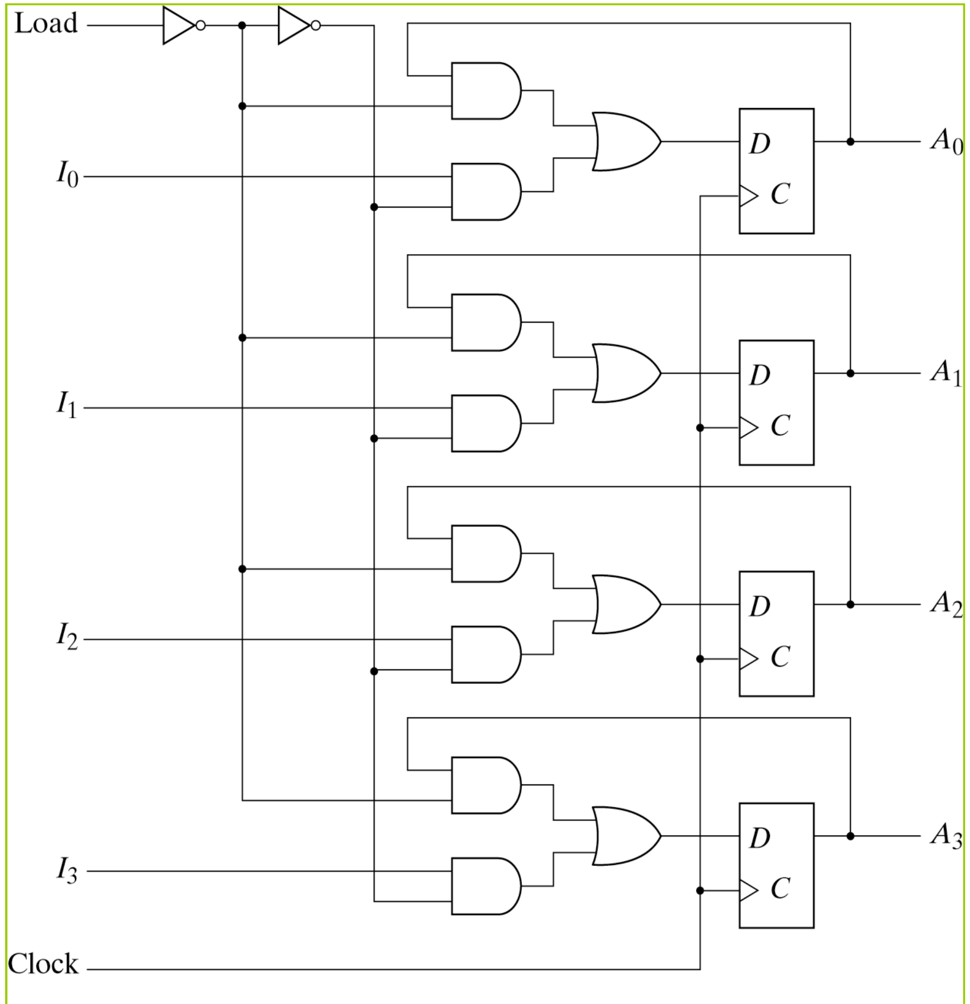


Fig. 6-2 4-Bit Register with Parallel Load

- Clock=1 ;input information
  ->loading
- Clock=0 ;the content of the register ->unchanged
- Load input=1 ; the I inputs are transferred into the register
- Load input=0 ; maintain the content of the register

6.2 SHIFT REGISTERS

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

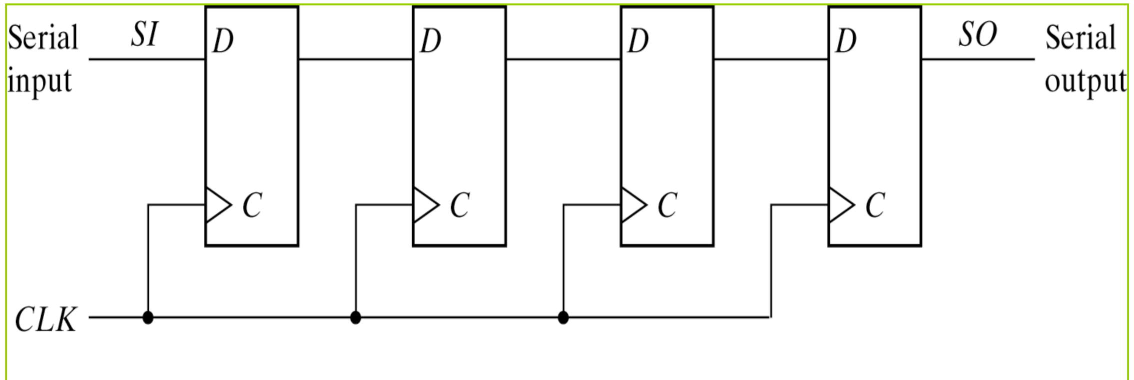


Fig. 6-3 4-Bit Shift Register

The simplest shift register

6.2 SHIFT REGISTERS - Serial Transfer

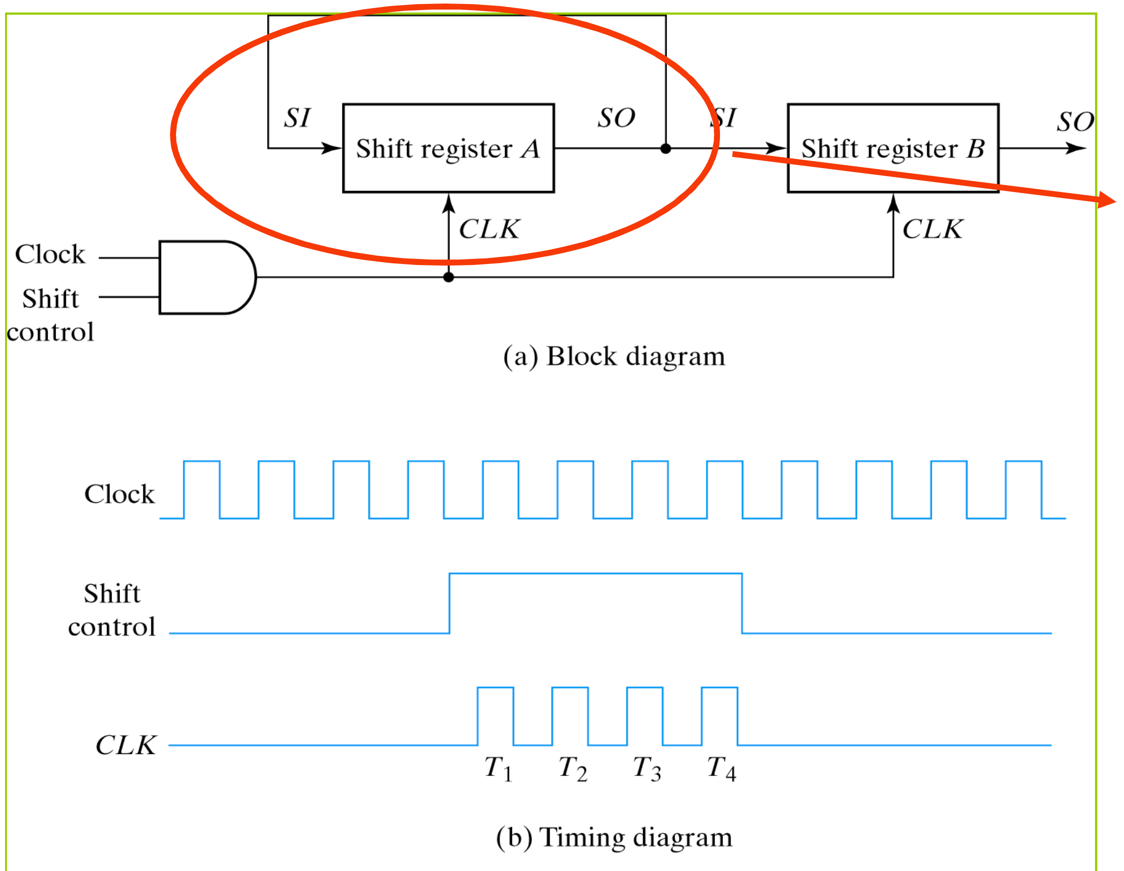


Fig. 6-4 Serial Transfer from Register *A* to register *B*

To prevent the loss of information stored in the source register

Serial-Transfer Example

Timing pulse	Shift register A	Shift register B	Serial output of B
Initial value	1 0 1 1	0 0 1 0	0
After T_1	1 1 0 1	1 0 0 1	1
After T_2	1 1 1 0	1 1 0 0	0
After T_3	0 1 1 1	0 1 1 0	0
After T_4	1 0 1 1	1 0 1 1	1

6.2 SHIFT REGISTERS - Serial Addition

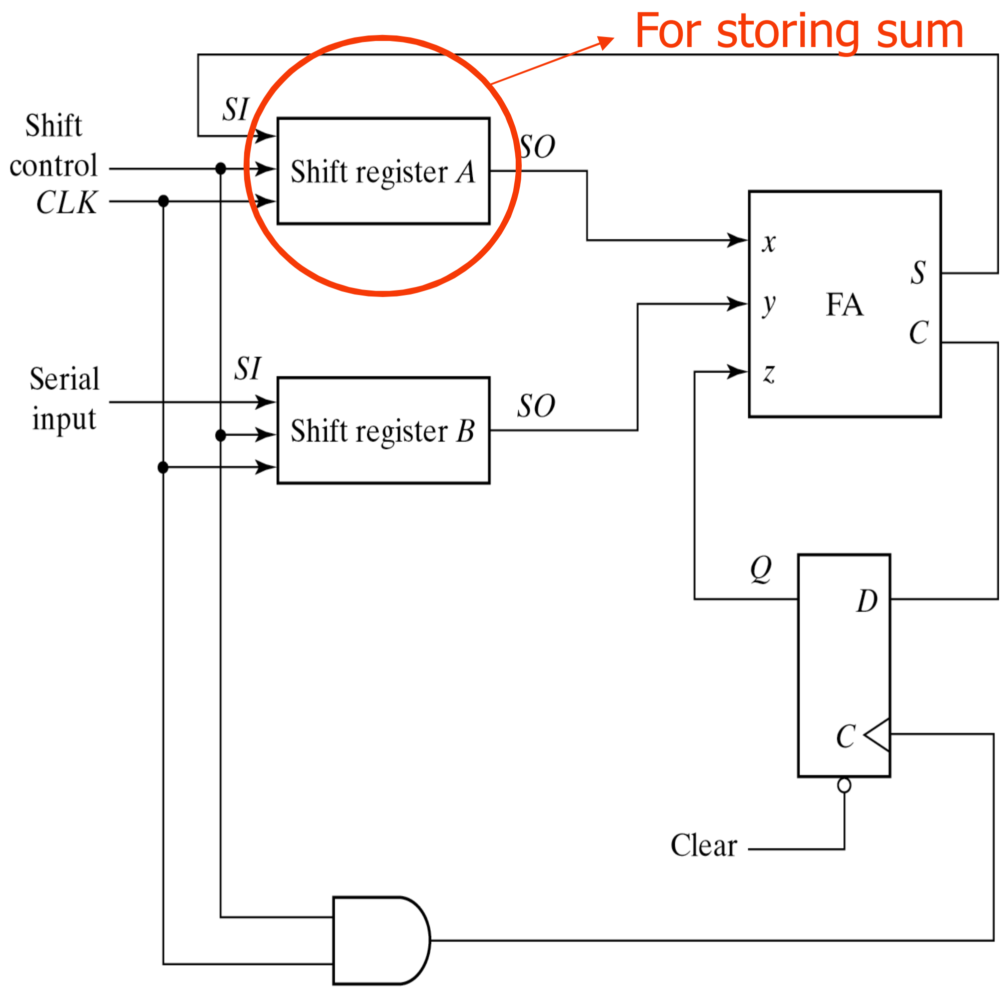


Fig. 6-5 Serial Adder

Operation

- ❑the **A** register ->augend ,the **B** register ->addend ,carry ->0
- ❑The **SO** of **A** and **B** provide a pair of significant bits for the **FA**
- ❑ Output **Q** gives the input carry at **z**
- ❑The shift-right control enables both registers and the carry flip-flop.
- ❑The sum bit from **S** enters the leftmost flip-flop of **A**

6.2 SHIFT REGISTERS - State Table for Serial Adder

Present value of carry

Output carry

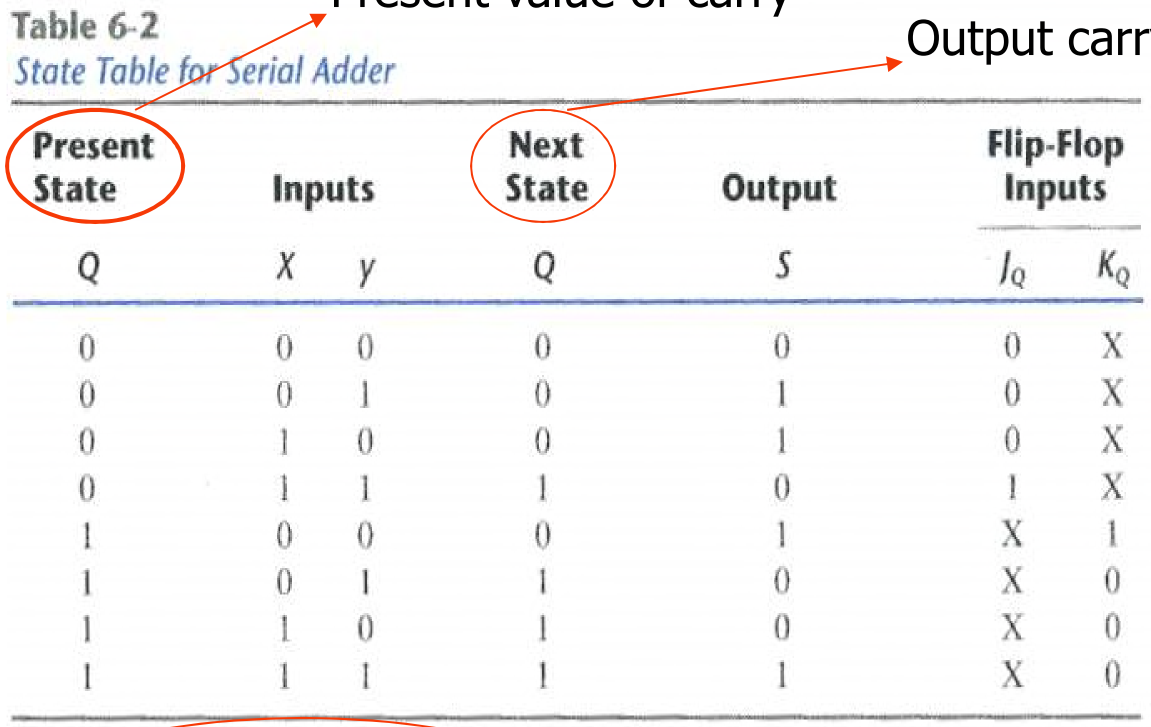


Table 6-2
State Table for Serial Adder

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

$$J_Q = xy$$
$$K_Q = x'y' = (x+y)'$$
$$S = x \oplus y \oplus Q$$

By k-map

6.2 SHIFT REGISTERS - Second form of Serial Adder

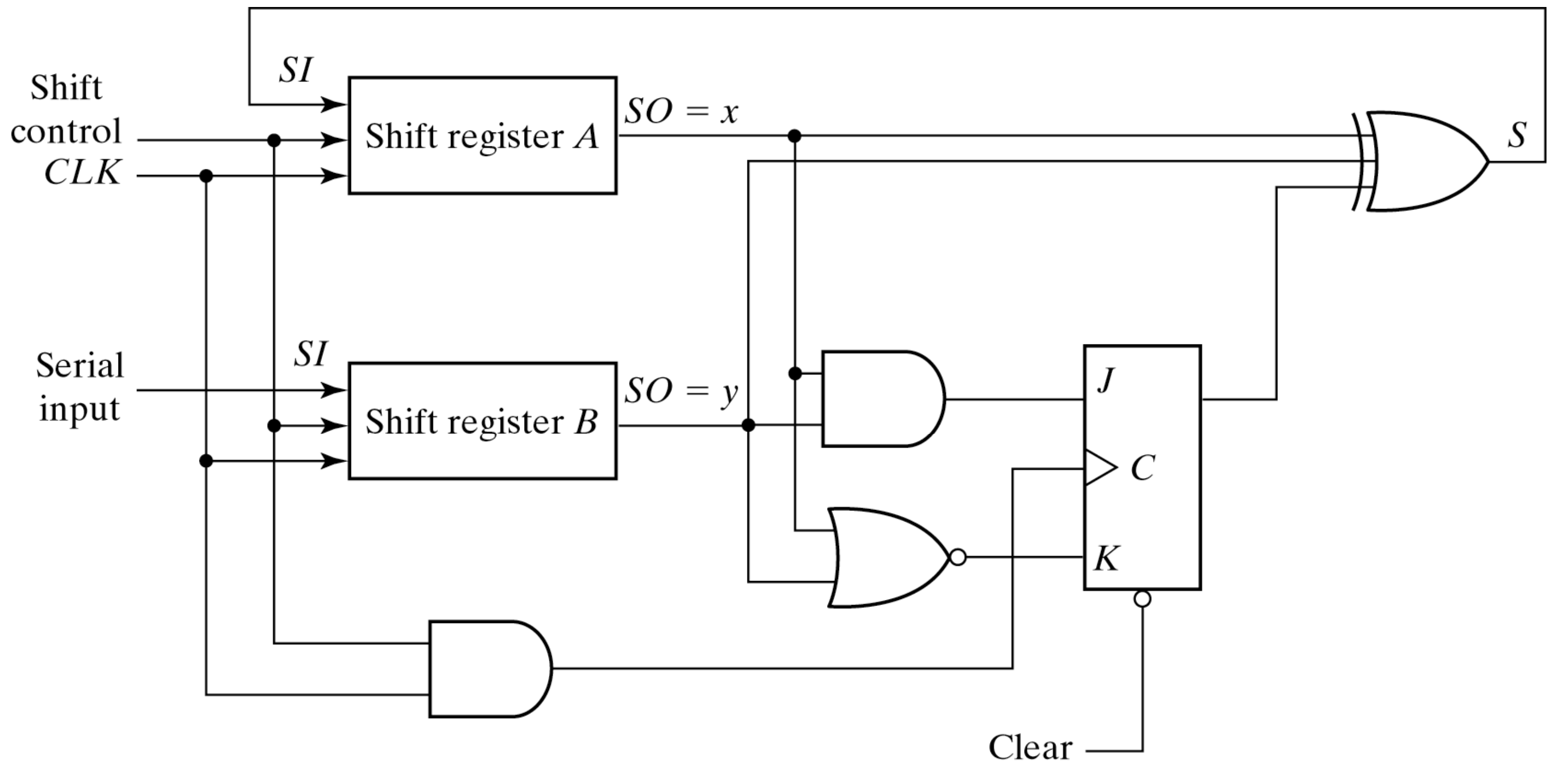


Fig. 6-6 Second form of Serial Adder

Universal Shift Register

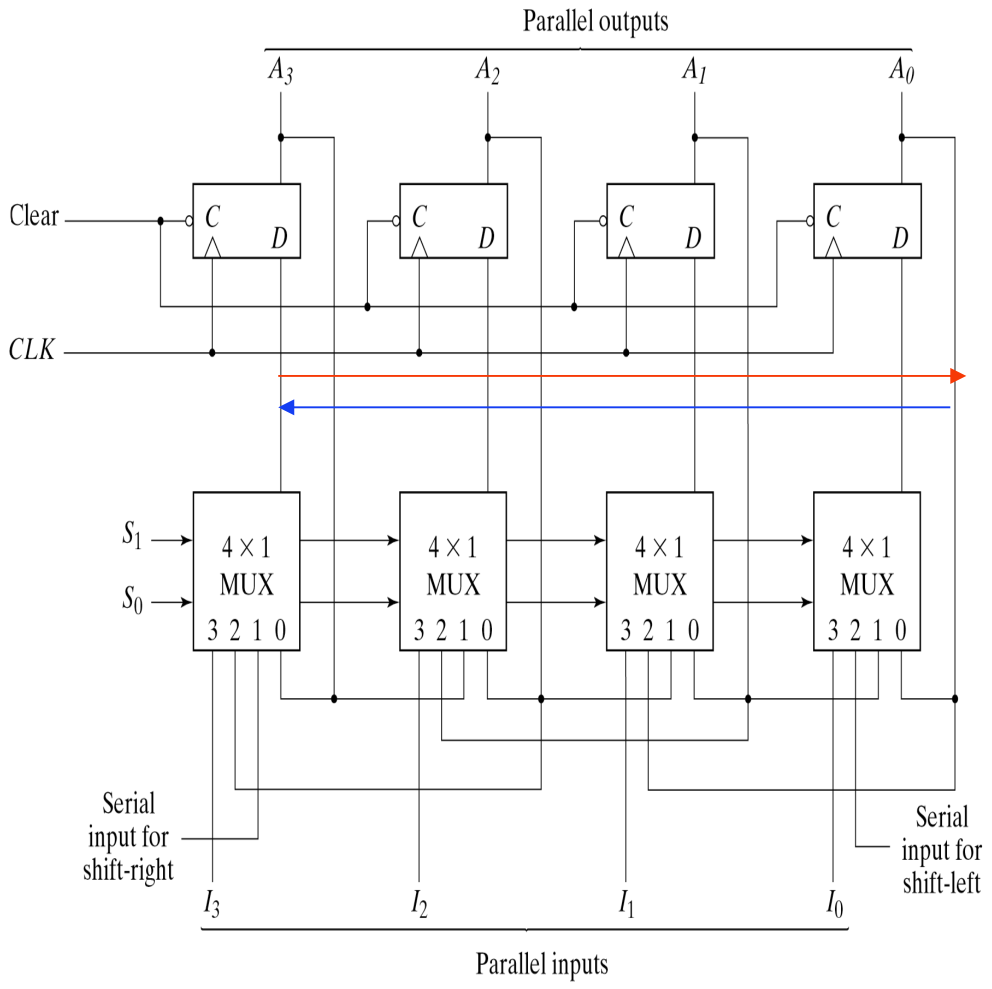
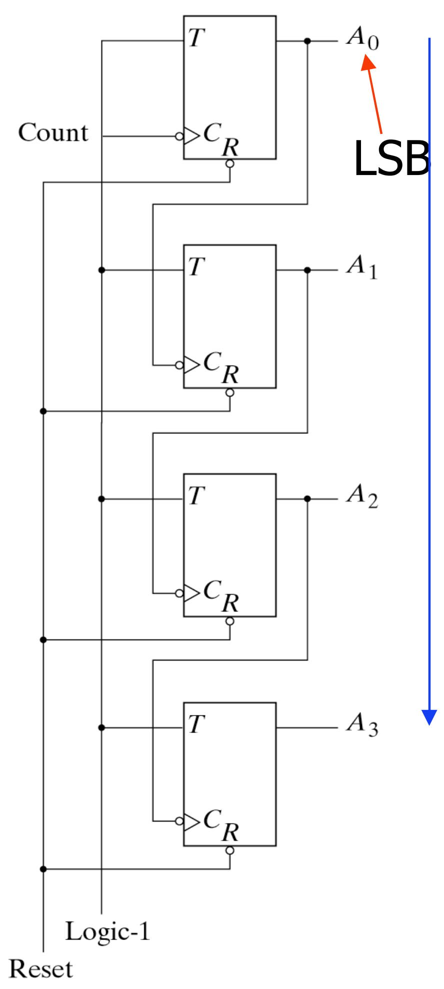


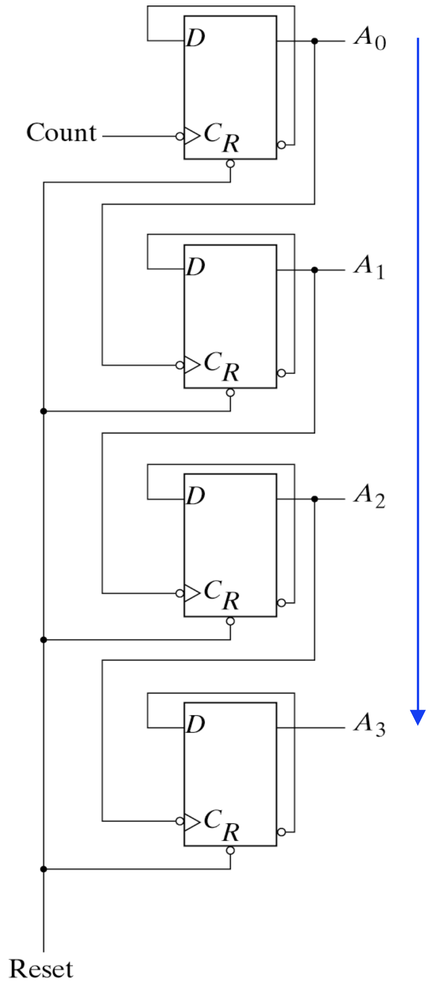
Fig. 6-7 4-Bit Universal Shift Register

- **S_1, S_0** -> 0, 0 **;No change**
- **S_1, S_0** -> 0, 1 **;Shift right**
- **S_1, S_0** -> 1, 0 **;Shift left**
- **S_1, S_0** -> 1, 1 **;Parallel load**

6.3 RIPPLE COUNTERS



(a) With T flip-flops



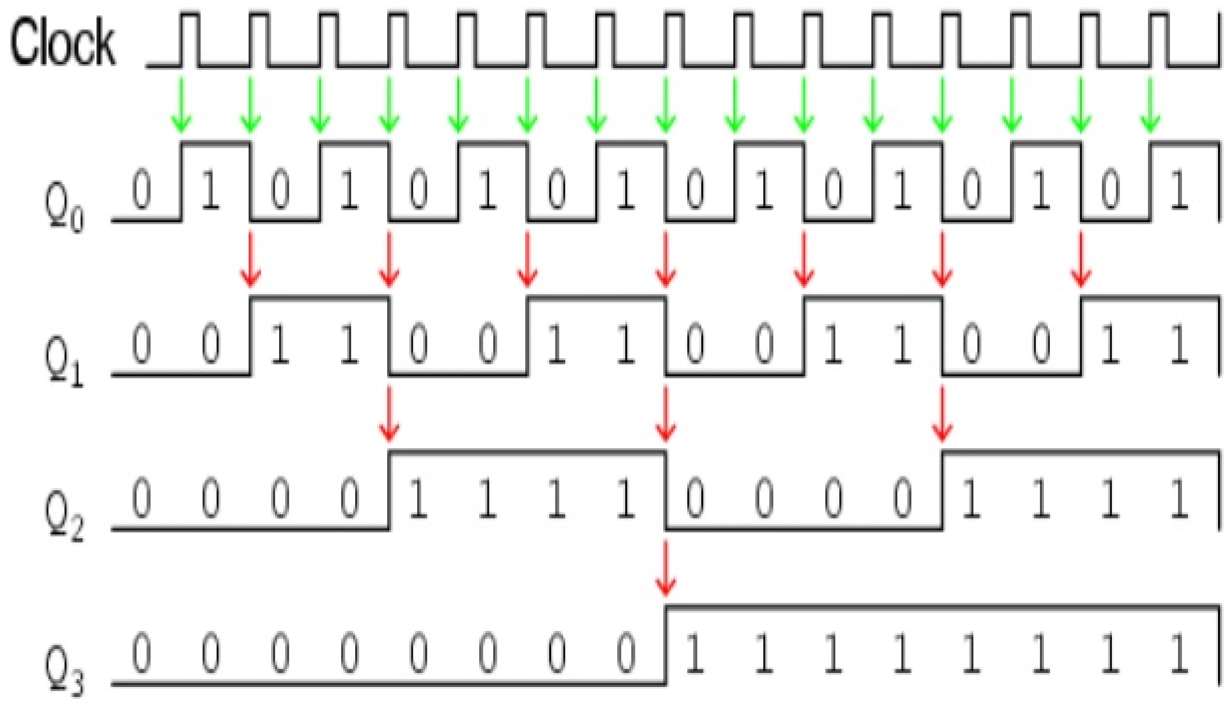
(b) With D flip-flops

Fig. 6-8 4-Bit Binary Ripple Counter

6.3 RIPPLE COUNTERS - Binary Ripple Counter

Count sequence A_3 A_2 A_1 A_0		**Conditions for Complementing**
0 0 0 0	Complement A_0	
0 0 0 1	Complement A_0	A_0 will go from 1 to 0 and complement A_1
0 0 1 0	Complement A_0	
0 0 1 1	Complement A_0	A_0 will go from 1 to 0 and complement A_1 ; A_1 will go from 1 to 0 and complement A_2
0 1 0 0	Complement A_0	
0 1 0 1	Complement A_0	A_0 will go from 1 to 0 and complement A_1
0 1 1 0	Complement A_0	
0 1 1 1	Complement A_0	
....		
1 0 0 0	and so on…	

6.3 RIPPLE COUNTERS



Clock
Q_0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
Q_1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1
Q_2 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1
Q_3 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1

6

https://www.slideshare.net/LeeDiaz2/counters-11983921

6.3 RIPPLE COUNTERS - BCD Ripple Counter

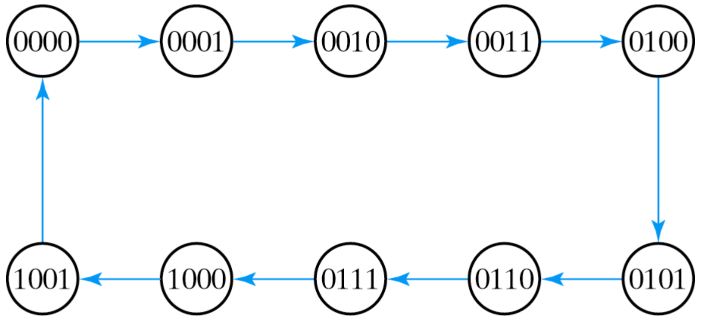


Fig. 6-9 State Diagram of a Decimal BCD-Counter

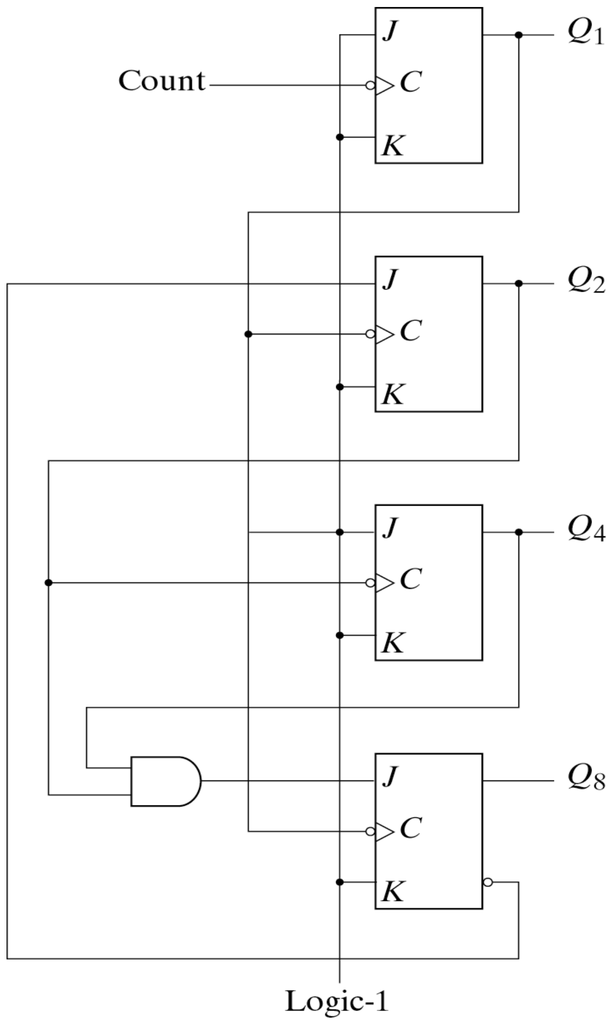


Fig. 6-10 BCD Ripple Counter

6.3 RIPPLE COUNTERS - BCD Ripple Counter

Operation

1. Q_1 is complemented on the negative edge of every count pulse.

2. Q_2 is complemented if Q_8=0 and Q_1 goes from 1 to 0. Q_2 is cleared if Q_8=1 and Q_1 goes from 1 to 0.

3. Q_4 is complemented when Q_2 goes from 1 to 0.

4 .Q_8 is complemented when Q_4Q_2=11 and Q_1 goes from 1 to 0. Q_8 is cleared if either Q_4 or Q_2 is 0 and Q_1 goes from 1 to 0

6.3 RIPPLE COUNTERS - BCD Ripple Counter

- Three-Decade Decimal BCD Counter

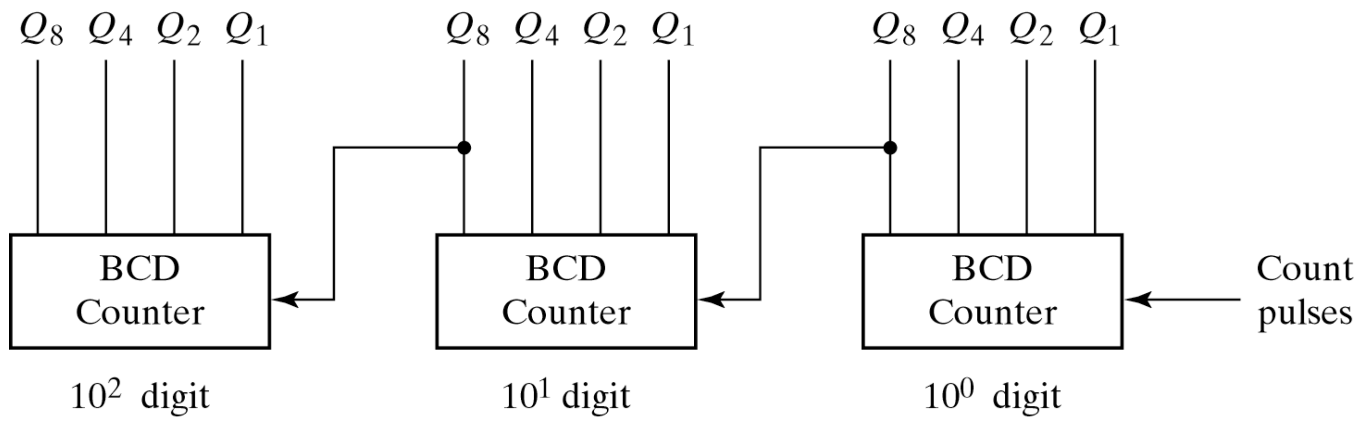


Fig. 6-11 Block Diagram of a Three-Decade Decimal BCD Counter

- To count from **0** to **999**, We need a three-decade counter.

6.4 SYNCHRONOUS COUNTERS

- Synchronous Counters
  - Binary Counter
  - Up-Down Binary Counter
  - BCD Counter
  - Binary Counter with Parallel Load
  - Other Counter

6.4 SYNCHRONOUS COUNTERS - Binary Counter

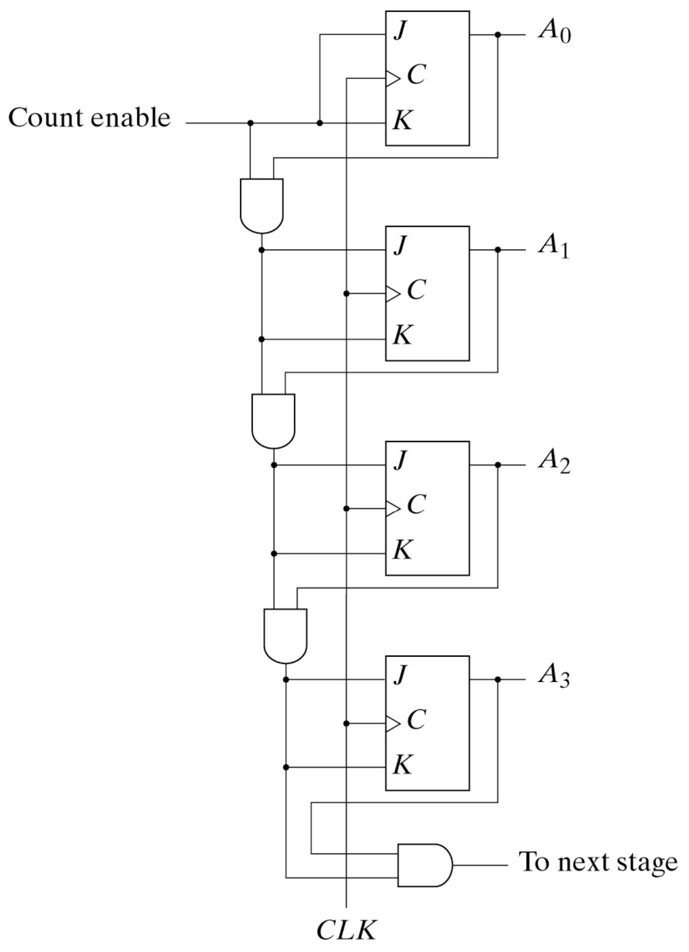


Fig. 6-12 4-Bit Synchronous Binary Counter

- ❑ The first stage **A_0** has its **J** and **K** equal to **1** if the counter is enabled .
- ❑The other **J** and **K** inputs are equal to **1** if all previous low-order bits are equal to **1** and the count is enabled.

6.4 SYNCHRONOUS COUNTERS - Up-Down Binary Counter

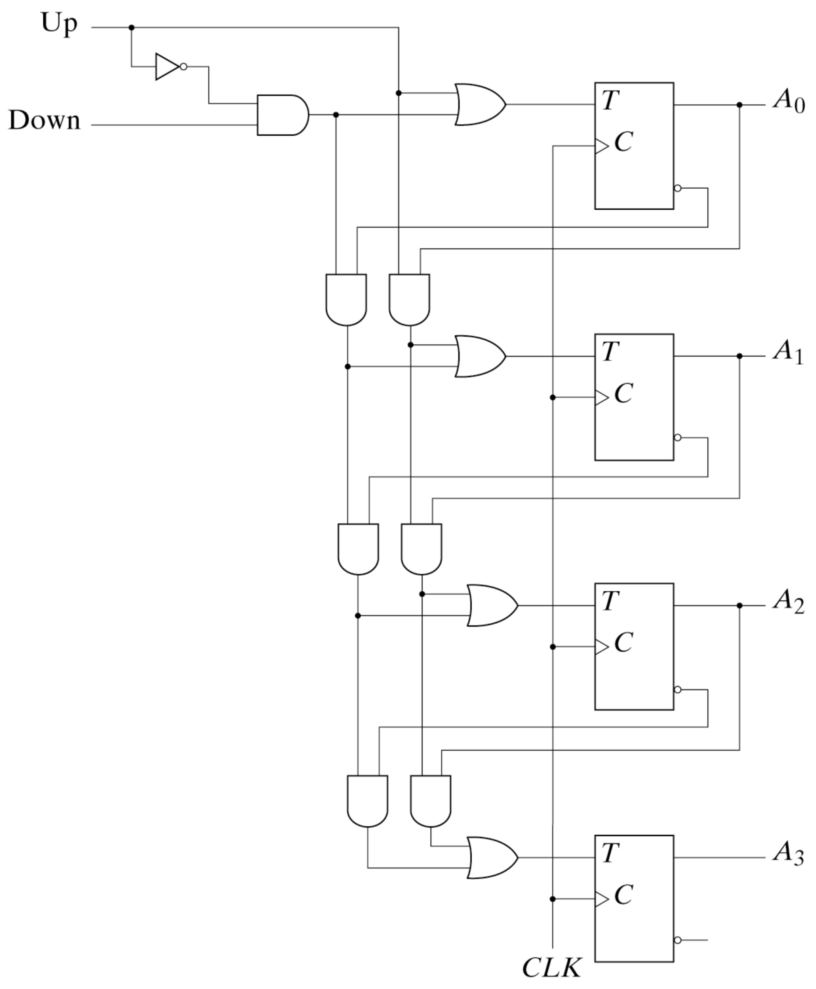


Fig. 6-13 4-Bit Up-Down Binary Counter

- **Up** input control=**1** ;count up (the **T** inputs receive their signals from the values of the previous normal outputs of the flip-flops.)
- **Down** input control=**1**, **up** input control=**0** ; count down
- **Up=down**=0 ;unchanged state
- **Up=down**=1 ;count up

6.4 SYNCHRONOUS COUNTERS - BCD Counter

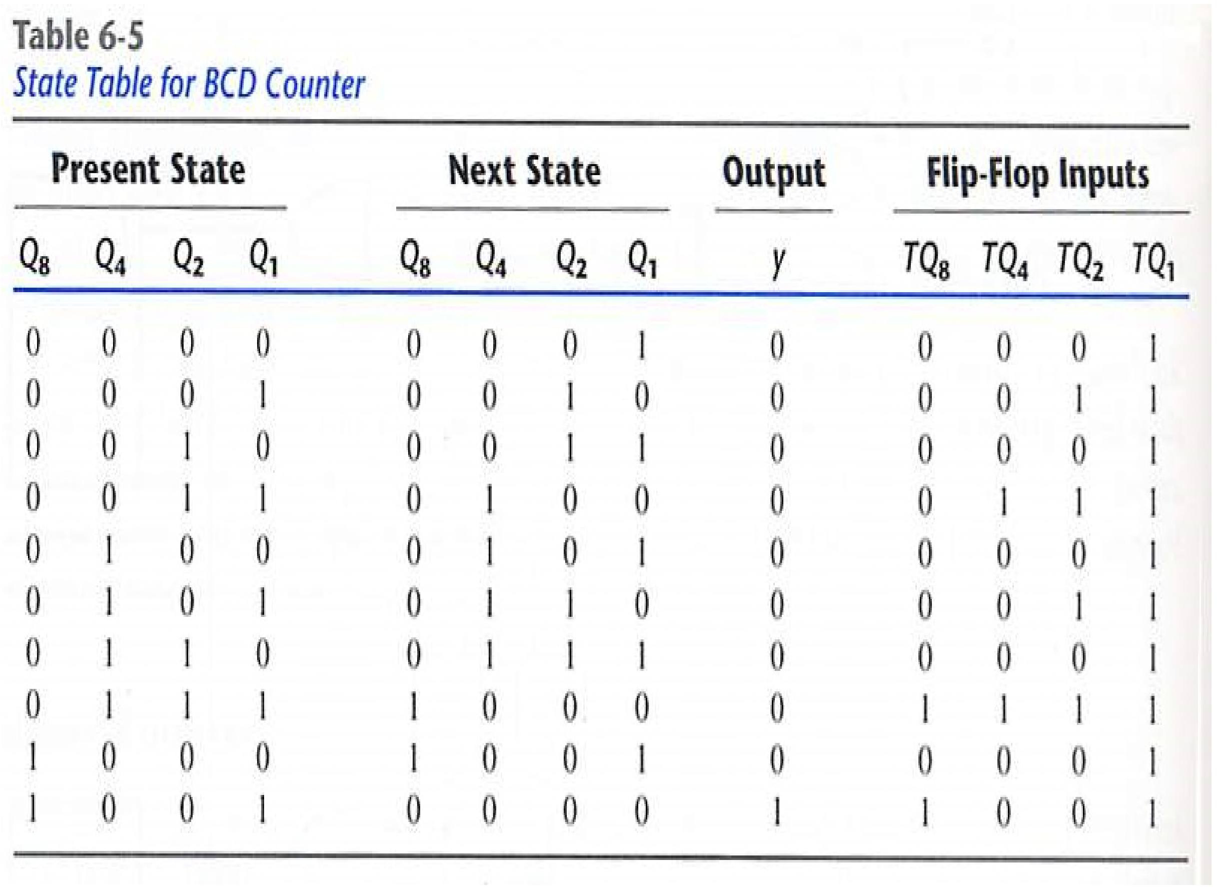


Table 6-5
State Table for BCD Counter

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

$$T_{Q1} = 1$$
$$T_{Q2} = Q'_8 Q_1$$
$$T_{Q4} = Q_2 Q_1$$
$$T_{Q8} = Q_8 Q_1 + Q_4 Q_2 Q_1$$
$$y = Q_8 Q_1$$

6.4 SYNCHRONOUS COUNTERS - Binary Counter with Parallel Load

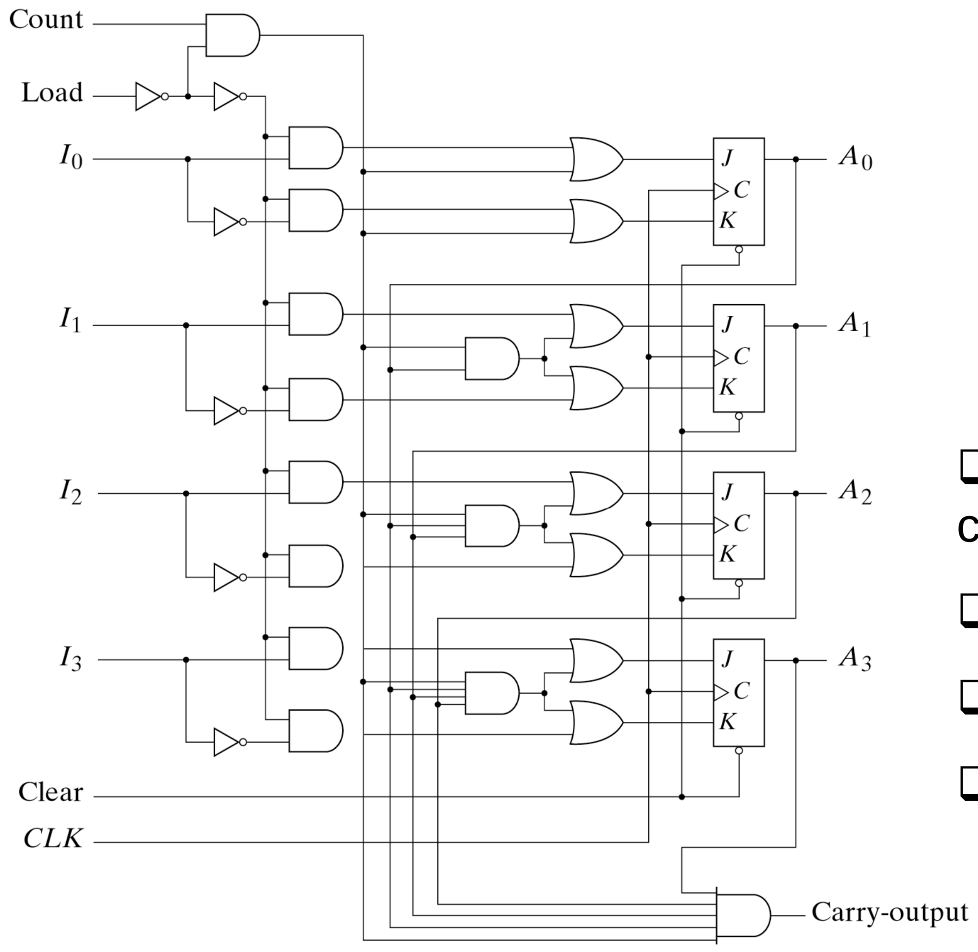


Fig. 6-14 4-Bit Binary Counter with Parallel Load

- Input **load** control=1 ; disables the count sequence ,data transfer
- **Load** =0 and **count**=1 ;count
- **Load**=0 and **count**=0 ;unchanged
- **Carry out**=1(all flip-flop=1)

6.4 SYNCHRONOUS COUNTERS - Binary Counter with Parallel Load

- BCD COUNTER using Binary Counter with Parallel Load

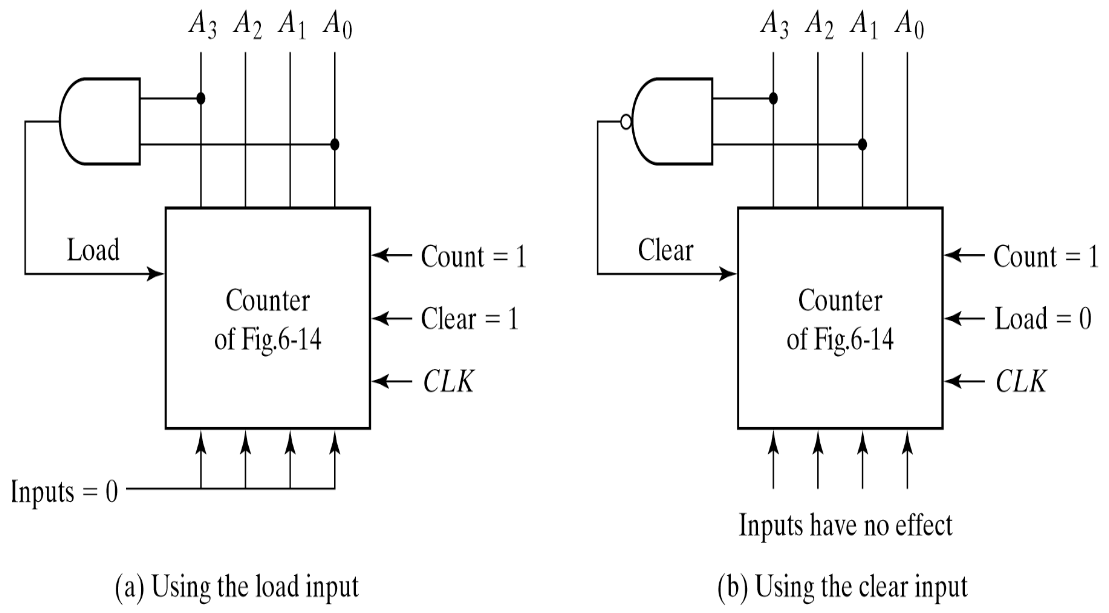


Fig. 6-15 Two ways to Achieve a BCD Counter Using a Counter with Parallel Load

❑The **AND** gate detects the occurrence of state **1001(9)** in the output. In this state, the load input is enabled and all-**0**'s input is loaded into register.

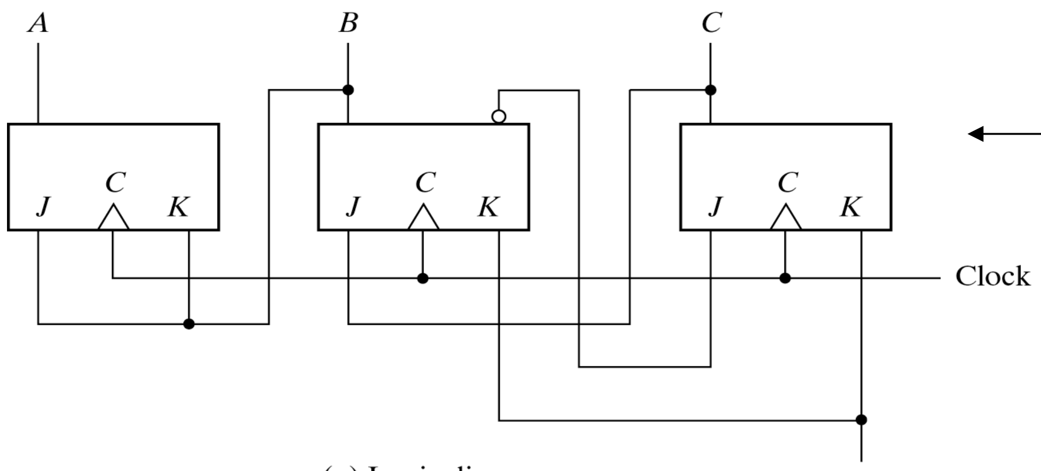
❑The **NAND** gate detects the count of **1010(10)** , as soon as this count occurs the register is **cleared.**

❑ A momentary **spike** occurs in output A2 as the count goes from **1001** to **1010** and immediately to **0000**

6.5 OTHER COUNTERS

- Counter with Unused States
  - Don’t care conditional Counter
- Ring Counter
- Johnson Counter

6.5 OTHER COUNTERS - Counter with Unused States



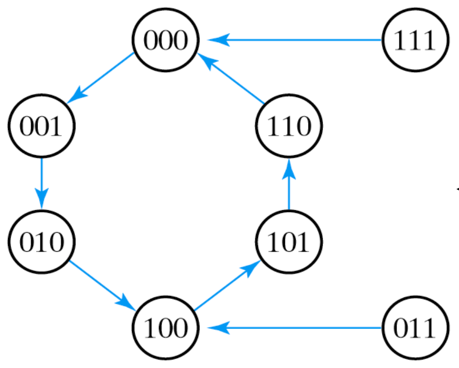
(a) Logic diagram

$J_A = B \qquad K_A = B$

$J_B = C \qquad K_B = 1$

$J_C = B' \qquad K_C = 1$

Except **011 ,111**



(b) State diagram

Fig. 6-16 Counter with Unused States

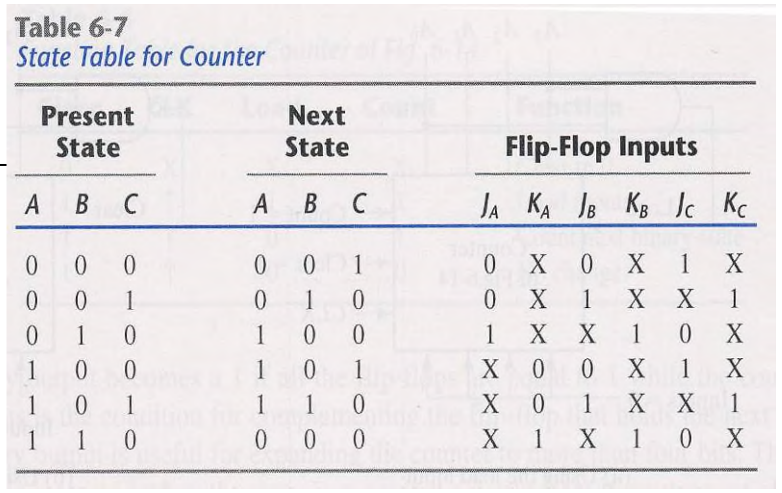
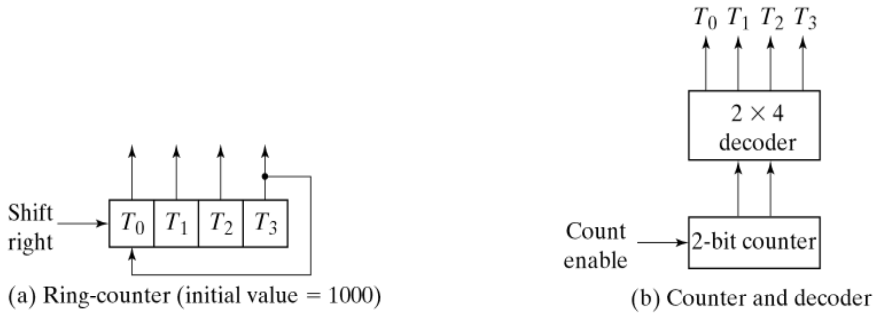


Table 6-7
State Table for Counter

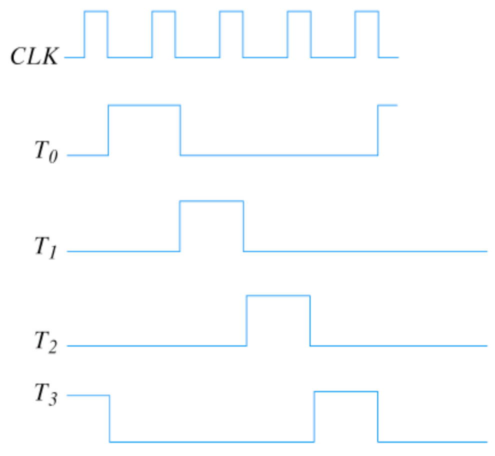
Present State			Next State			Flip-Flop Inputs					
A	*B*	*C*	*A*	*B*	*C*	J_A	K_A	J_B	K_B	J_C	K_C
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	0	0	X	1	X	X	1
0	1	0	1	0	0	1	X	X	1	0	X
1	0	0	1	0	1	X	0	0	X	1	X
1	0	1	1	1	0	X	0	1	X	X	1
1	1	0	0	0	0	X	1	X	1	0	X

6.5 OTHER COUNTERS - Ring Counter



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

(b) Counter and decoder

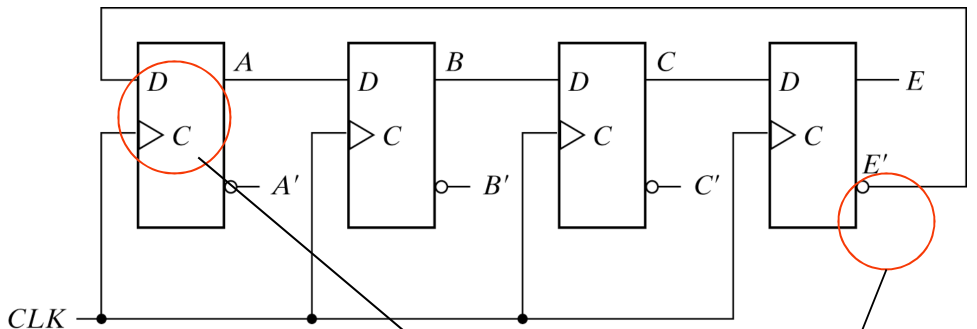


(c) Sequence of four timing signals

Fig. 6-17 Generation of Timing Signals

- ❑ A circular sift register with only one flip-flop being set at any **particular time**. ; all others are cleared.
- ❑**The single bit** is shifted from one flip-flop to the other.

6.5 OTHER COUNTERS - Johnson Counter



(a) Four-stage switch-tail ring counter

Sequence number	Flip-flop outputs A	B	C	E	AND gate required for output
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	$C'E$

(b) Count sequence and required decoding

Fig. 6-18 Construction of a Johnson Counter

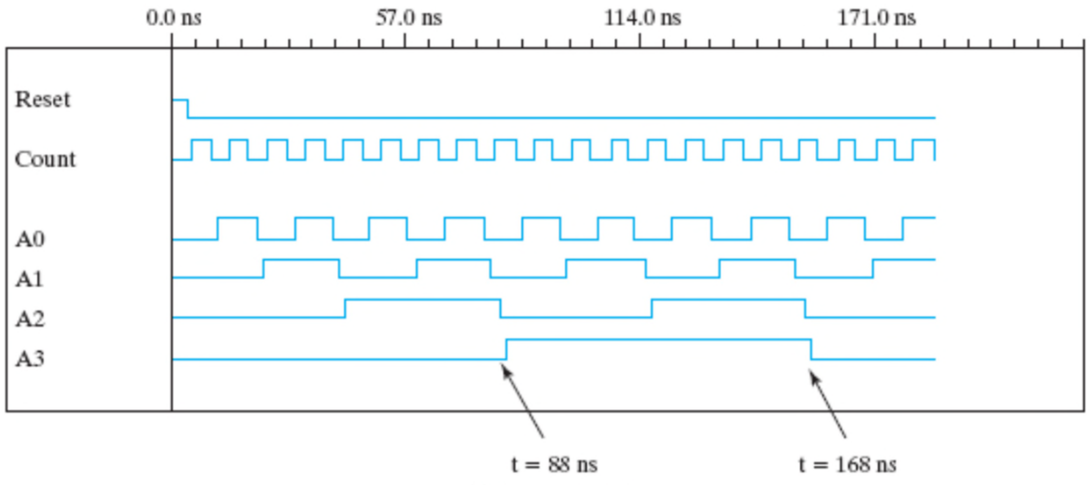
- A circular shift register with **the complement output of the last flip-flop** connected to the input of the first flip-flop.

6.6 HDL FOR REGISTERS AND COUNTERS - Shift Register

```
// Behavioral description of a 4-bit universal shift register
// Fig. 6.7 and Table 6.3
module Shift_Register_4_beh (                           // V2001, 2005
  output reg      [3: 0]          A_par,               // Register output
  input                           [3: 0]               I_par,              //
      Parallel input
  input                                                 s1, s0,             //
      Select inputs
                                  MSB_in, LSB_in,                           //
      Serial inputs
                                  CLK, Clear           // Clock and 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
        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

module t_Shift_Register_4_beh ();
  reg   s1, s0,                                        // Select inputs
        MSB_in, LSB_in,           // Serial inputs
        clk, reset_b;                                  // Clock and Clear
  reg   [3: 0]      I_par;                             // Parallel input
  wire  [3: 0]      A_par;                             // Register output

  Shift_Register_4_beh M0 (A_par, I_par,s1, s0, MSB_in, LSB_in, clk,
        reset_b);

  initial #200 $finish;
  initial begin clk = 0; forever #5 clk = ~clk; end

initial fork
  // test reset action load
  #3 reset_b = 1;
  #4 reset_b = 0;
  #9 reset_b = 1;

  // test parallel load
  #10 I_par = 4'hA;
  #10 {s1, s0} = 2'b11;

  // test shift right
  #30 MSB_in = 1'b0;
  #30 {s1, s0} = 2'b01;

  // test shift left
  #80 LSB_in = 1'b1;
  #80 {s1, s0} = 2'b10;

  // test circulation of data
 #130 {s1, s0} = 2'b11;
  #140 {s1, s0} = 2'b00;

 // test reset on the fly

  #150 reset_b = 1'b0;
  #160 reset_b = 1'b1;
  #160 {s1, s0} = 2'b11;

 join
endmodule
```

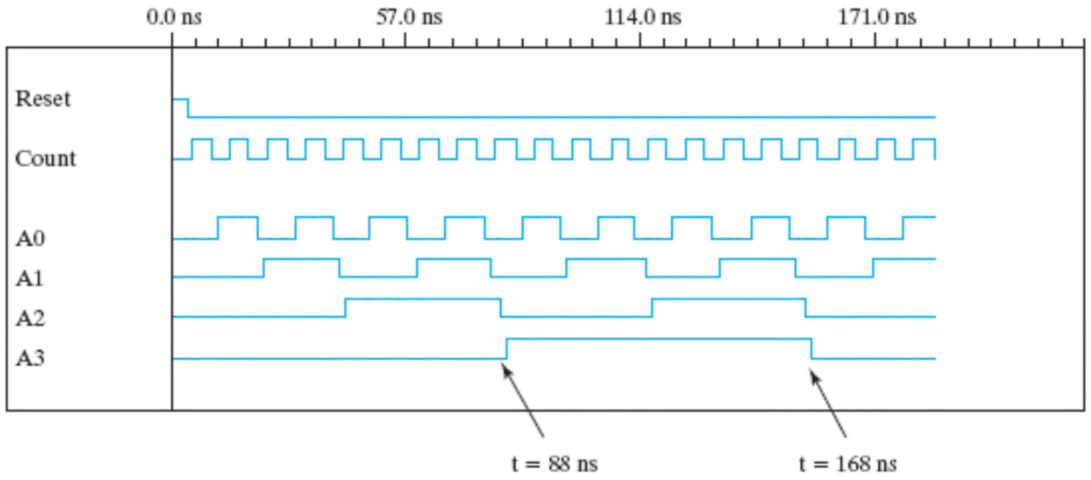
6.6 HDL FOR REGISTERS AND COUNTERS - Ripple Counter

```
`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;
// else젮Q <= #2 ~Q;젮젮?
endmodule
//Stimulus for testing ripple counter
module testcounter;
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
#200 $finish;

endmodule
```

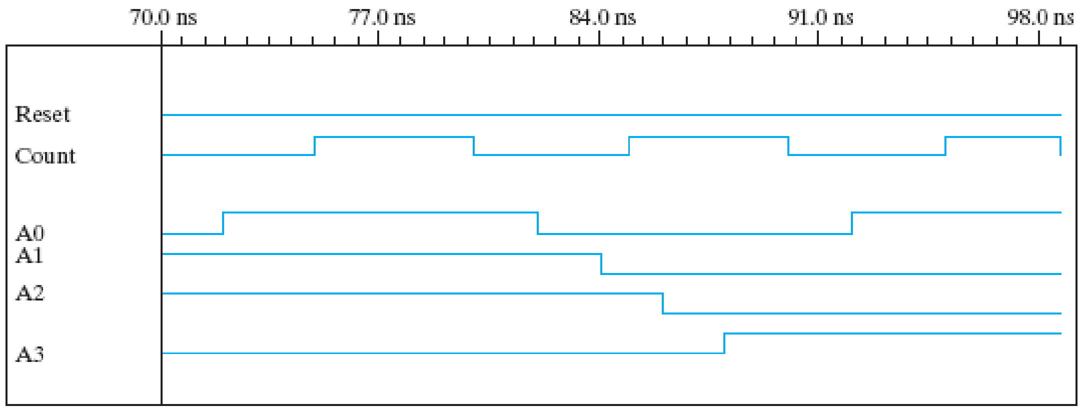


(a) From 0 to 180 ns

6.6 HDL FOR REGISTERS AND COUNTERS - Ripple Counter



(a) From 0 to 180 ns



(b) From 70 to 98 ns