Unit - 4: Registers and Counters

REGISTERS:

•A clocked sequential circuit consists of a **group of flip-flops and combinational gates**. The flip-flops are essential because, in their absence, the circuit reduces to a purely combinational circuit.

•A *register is a **group of flip-flops**, each one of which shares a **common clock** and is* capable of storing one bit of information.

•An ***n -bit register*** *consists of a group of* ***n*** **flip-flops** capable of storing ***n bits*** *of binary information.*

•*In addition to the flip-flops, a* register may have combinational gates that perform certain data-processing tasks.

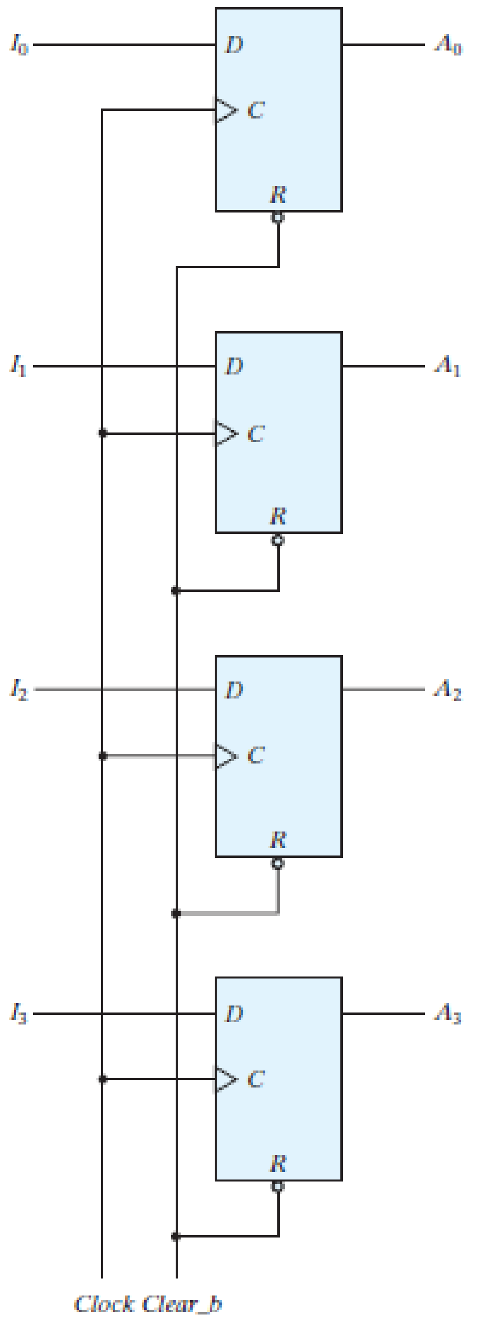


FIGURE: Four-bit register

Register with Parallel Load

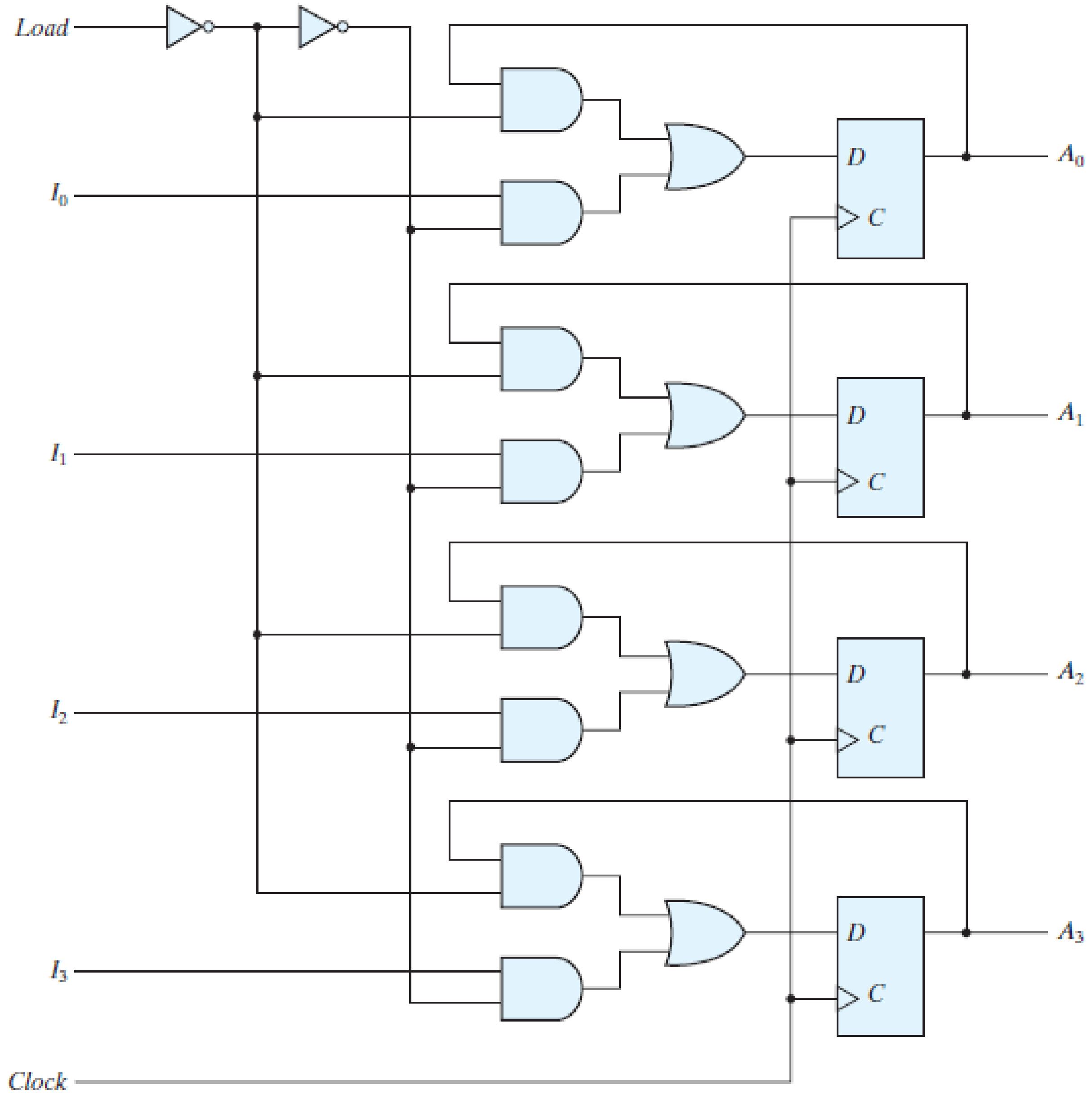


FIGURE: Four-bit register with parallel load

<u>SHIFT REGISTERS:</u>

•A register capable of shifting the binary information held in each cell to its neighboring cell, in a selected direction, is called a ***shift register***.

•*The logical configuration of a shift* register consists of a chain of flip-flops in cascade, with the output of one flip-flop connected to the input of the next flip-flop.

•All flip-flops receive common clock pulses, which activate the shift of data from one stage to the next.

•The simplest possible shift register is one that uses only flip-flops, as shown in the following Fig. The output of a given flip-flop is connected to the *D input of the* ***flip-flop at its right****. This* shift register is **unidirectional (left-to-right)**. Each clock pulse shifts the contents of the register one bit position to the right.

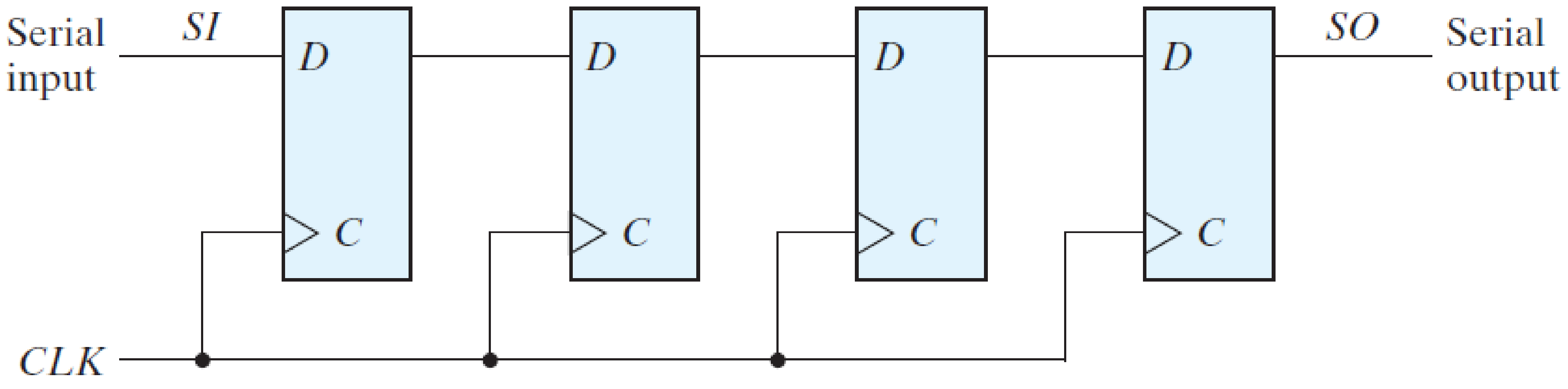
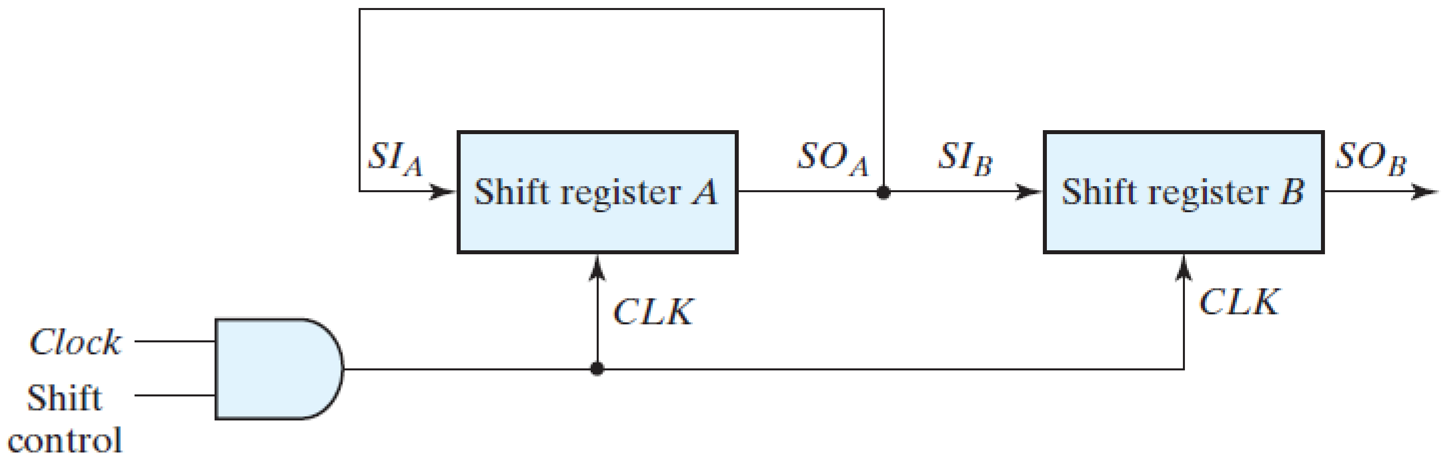


FIGURE: Four-bit shift register

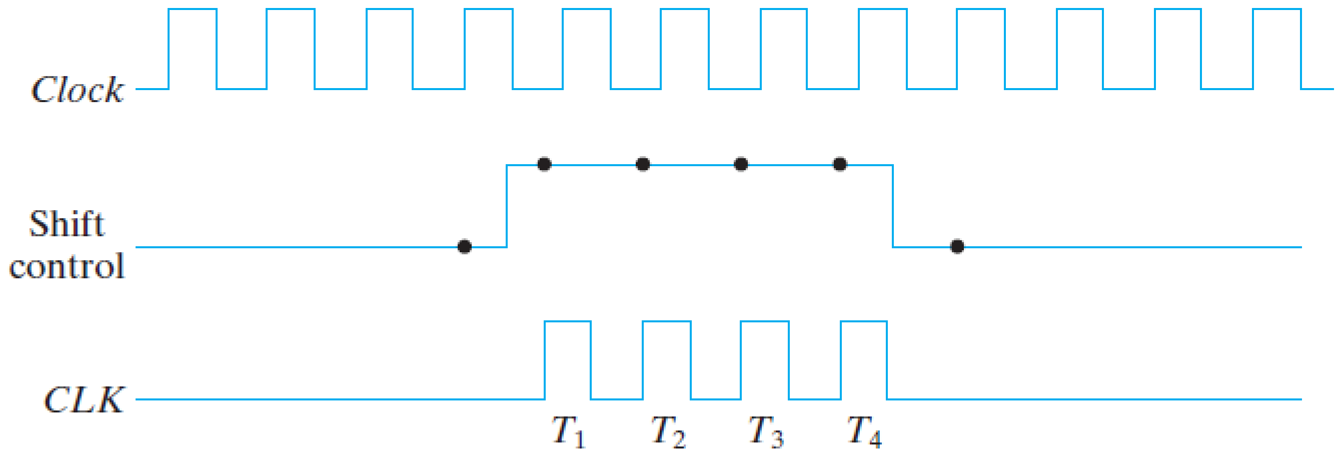
Serial Transfer

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



(a) Block diagram



(b) Timing diagram

FIGURE: Serial transfer from register *A to register B*

Serial Addition:

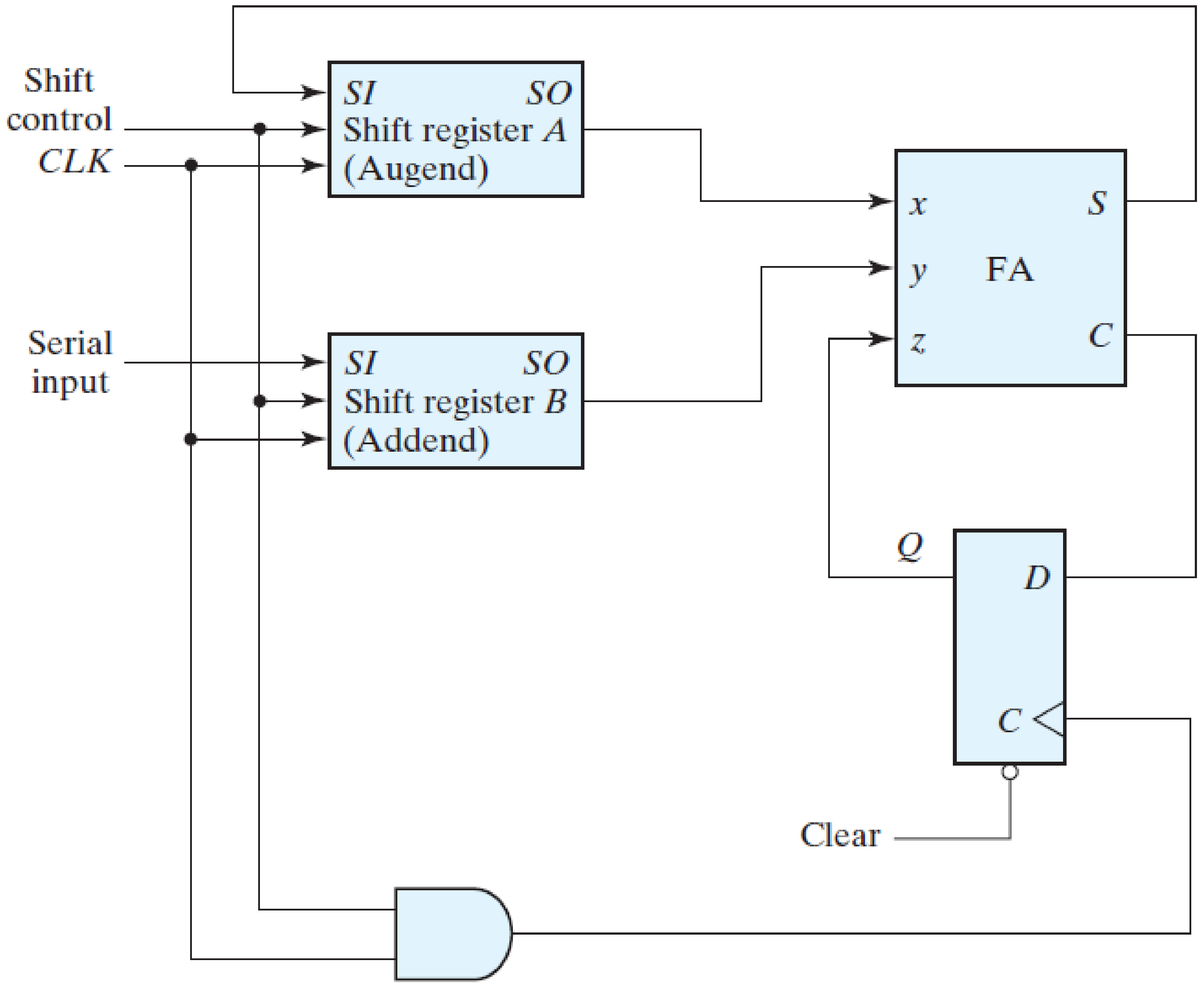


FIGURE: Serial adder

•The operation of the serial adder is as follows:

•Initially, register *A holds the augend,* register *B holds the addend, and the carry flip-flop is cleared to 0.*

•*The outputs (SO) of A* and *B provide a pair of significant bits for the full adder at x and y.*

•*Output Q of the flip-flop* provides the input carry at *z.*

•*The shift control enables both registers and the carry flip-flop,* so at the next clock pulse, both registers are shifted once to the right, the sum bit from *S* enters the leftmost flip-flop of *A, and the output carry is transferred into flip-flop Q.*

Redesign the serial adder with the use of a state table (using JK flip-flop.):

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

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

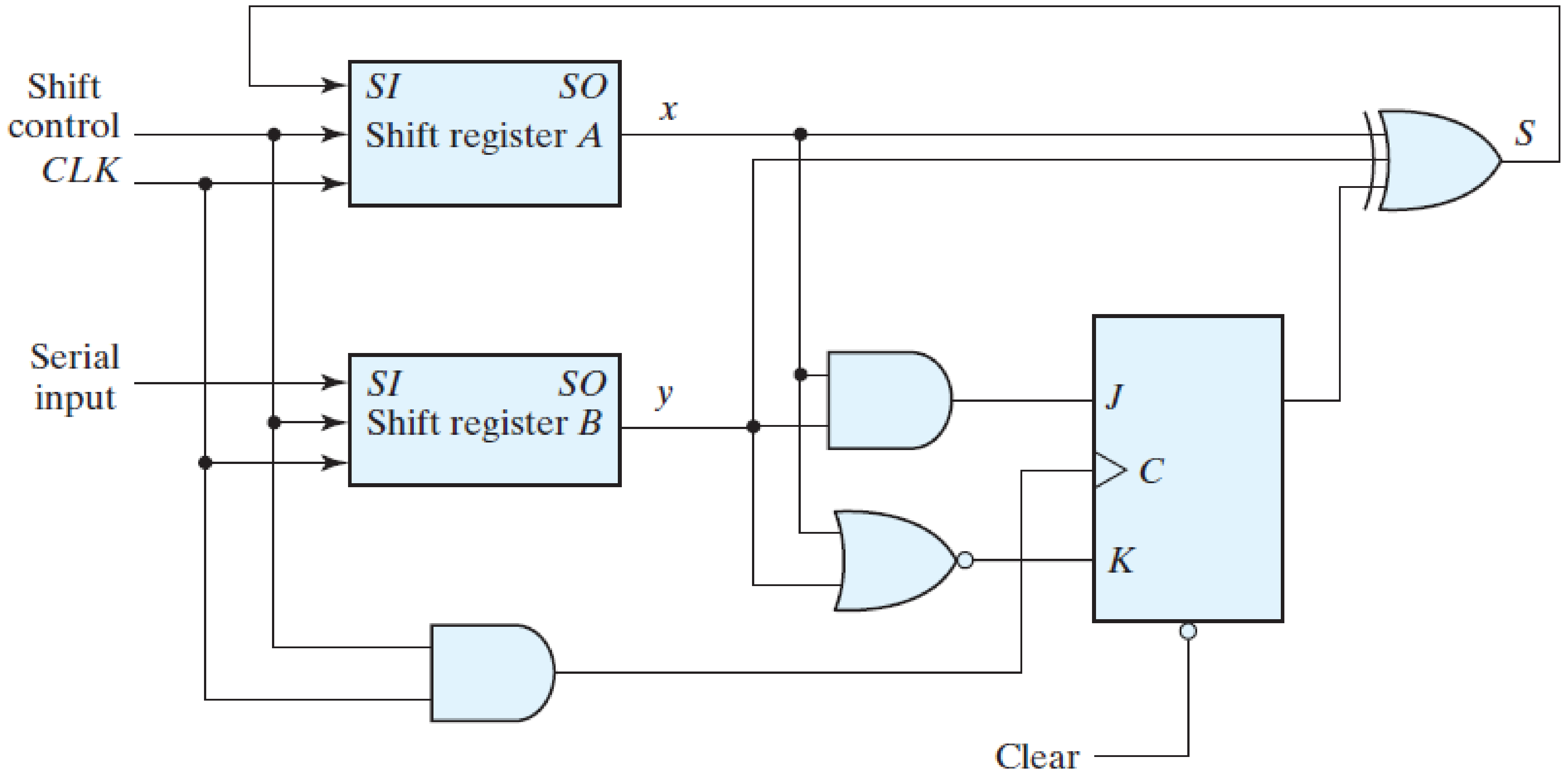


FIGURE: Second form of serial adder using JK Flip-flop.

Universal Shift Register:

•Some shift registers provide the necessary input and output terminals for parallel transfer. They may also have both shift-right and shift-left capabilities.

•The most general shift register has the following capabilities:

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 with the shift right.*

4. A *shift-left control to enable the shift-left operation and the serial input and output* lines associated with the shift left.

5. A *parallel-load control to enable a parallel transfer and the n input lines associated* with the parallel transfer.

6. *n parallel output lines.*

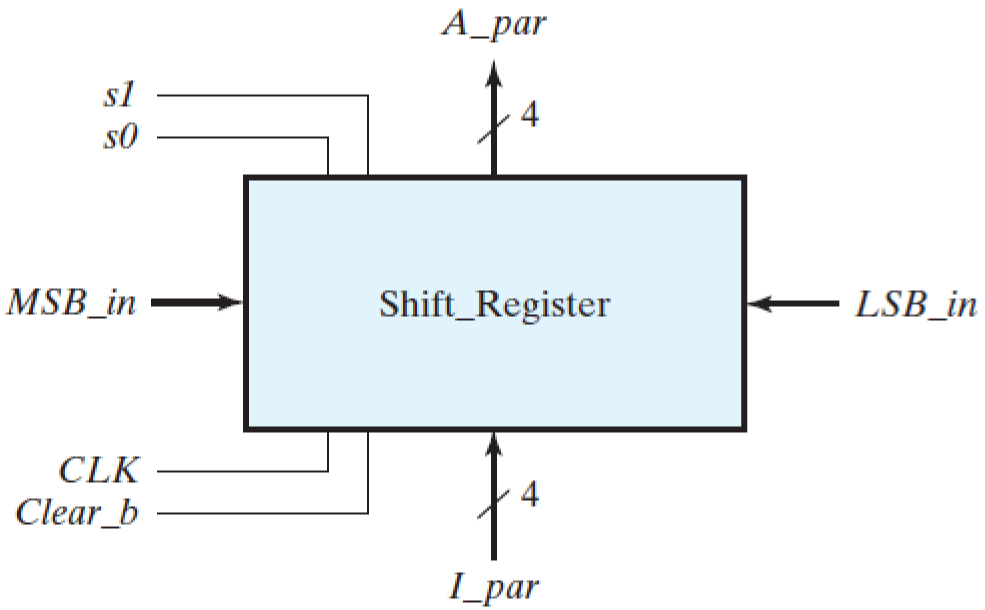
7. A control state that leaves the information in the register unchanged in response to the clock. Other shift registers may have only some of the preceding functions, with at least one shift operation.

•A register capable of shifting in one direction only is a *unidirectional shift register.*

•One that can shift in both directions is a *bidirectional shift register. If the register has* both shifts and parallel-load capabilities, it is referred to as a ***universal shift register***.

Function Table for the Register of Fig. 6.7

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



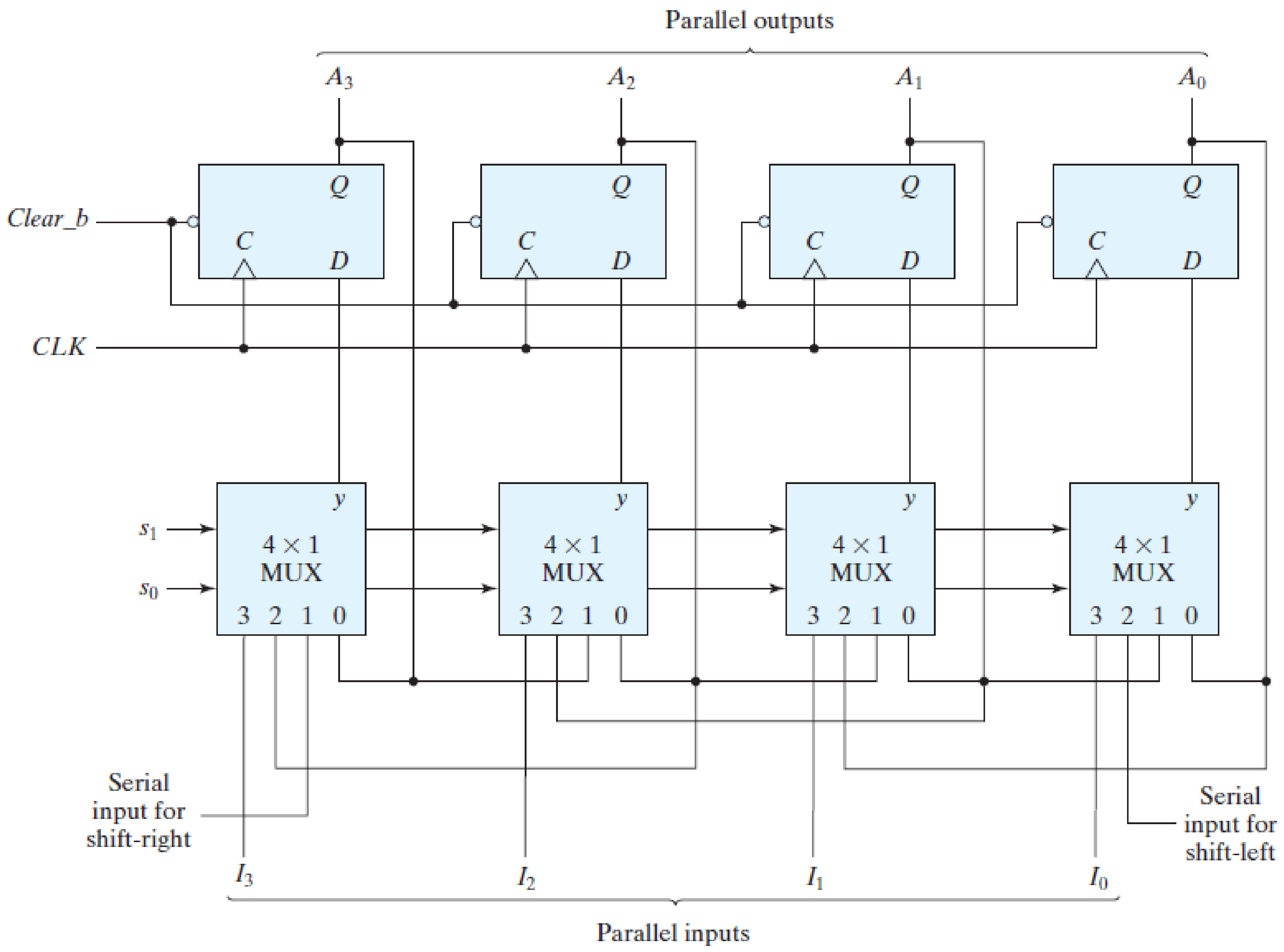


FIGURE: Four-bit universal shift register

HDL FOR REGISTERS AND COUNTERS:

HDL Example 6.1 (Universal Shift Register-Behavioral Model)

```
// 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_b                       // Clock and Clear
);

    always @ (posedge CLK, negedge Clear_b)    // V2001, 2005
      if (Clear_b == 0) 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
```

•Consider the following alternative **case statement for the shift** register model:

```
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
```

•Without the case item 2’b00, the **case statement would not find a match between** {*s1, s0*} *and the case items, so register A_par would be left unchanged.*

•A structural model of the universal shift register can be described by referring to the logic diagram of Fig. (b).
•The diagram shows that the register has **four multiplexers and four *D flip-flops.***
•*A **mux and flip-flop together are modeled as a stage of the shift register.*** The stage is a structural model, too, with an instantiation and interconnection of a module for a mux and another for a *D flip-flop.*
•*For simplicity, **the lowest-level modules of the*** **structure are behavioral models of the multiplexer and flip-flop**. Attention must be paid to the details of connecting the stages correctly.
•The top-level module declares the inputs and outputs and then instantiates four copies of a stage of the register.
•The **four instantiations specify the interconnections between the four stages** and provide the detailed construction of the register as specified in the logic diagram.

HDL Example 6.2 (Universal Shift Register-Structural Model)

```
// 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_b       // 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_b);
 stage ST1 (A_par[1], A_par[2], A_par[0], I_par[1], A_par[1], select, CLK, Clear_b);
 stage ST2 (A_par[2], A_par[3], A_par[1], I_par[2], A_par[2], select, CLK, Clear_b);
 stage ST3 (A_par[3], MSB_in, A_par[2], I_par[3], A_par[3], select, CLK, Clear_b);
endmodule
```

```
// One stage of shift register
module stage (i0, i1, i2, i3, Q, select, CLK, Clr_b);
  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_b;     // 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_b);
endmodule
```

```
// 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_b);
  output        Q;
  input         D, CLK, Clr;
  reg           Q;

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

RIPPLE COUNTERS (Asynchronous Counters):

Binary Ripple Counter:

- A binary ripple counter consists of a series connection of complementing flip-flops, with the output of each flip-flop connected to the *C input of the next higher order flip-flop.*
- The flip-flop holding the least significant bit receives the incoming count pulses.
- A **complementing flip-flop** can be obtained from a *JK flip-flop with the J and K inputs tied* together or from a *T flip-flop.*
- *A third possibility is to use a D flip-flop with the complement* output connected to the *D input.*

Table 6.4
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

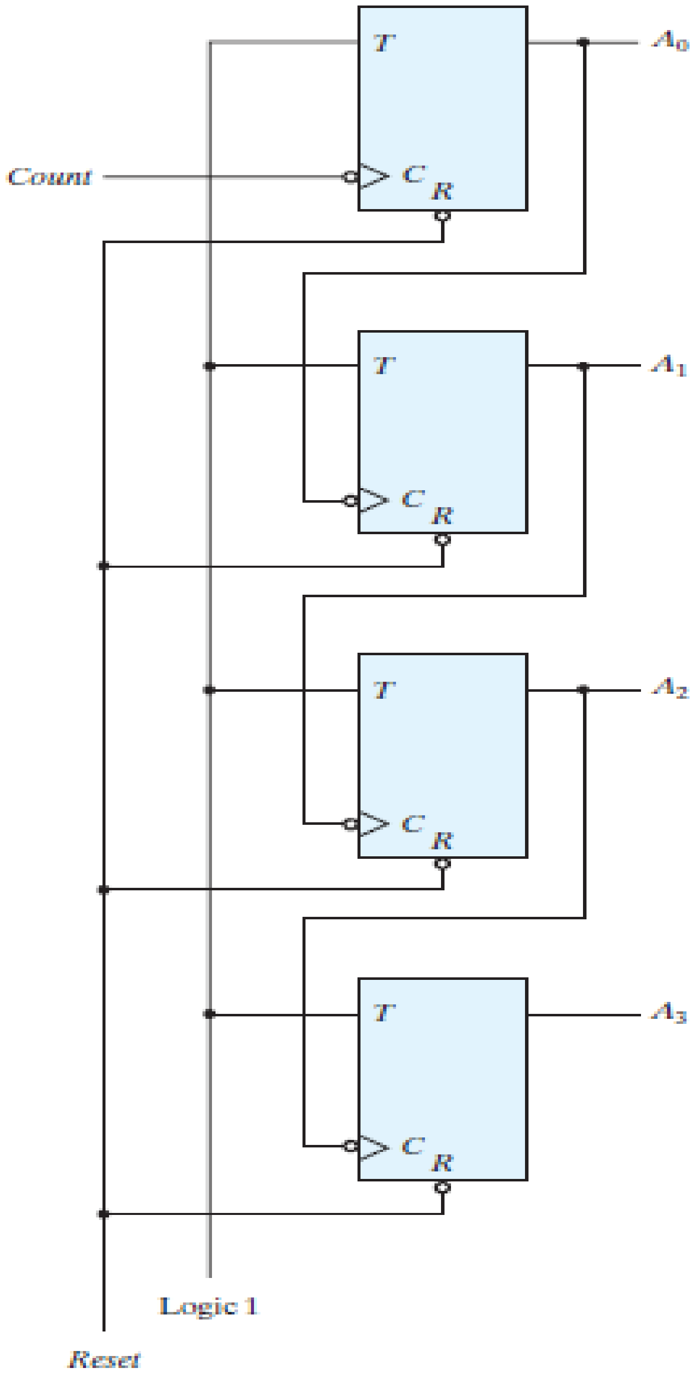


FIGURE: Four-bit binary ripple counter with T Flip-flops.

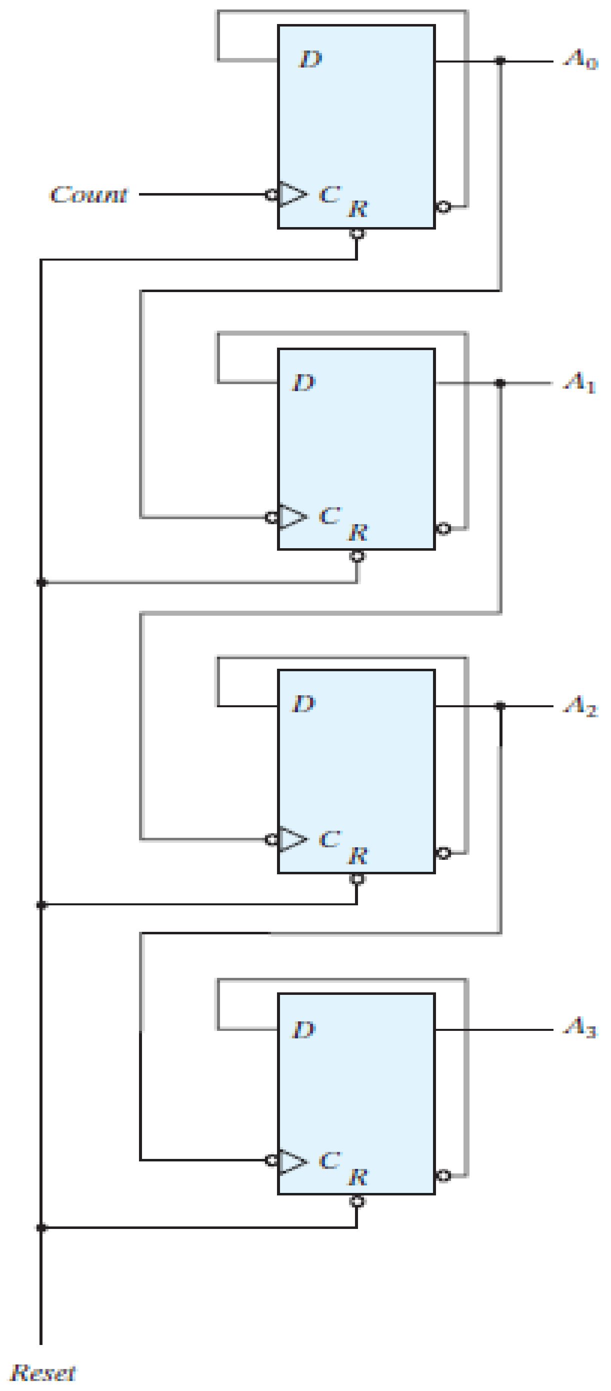


FIGURE: Four-bit binary ripple counter with D Flip-flops.

HDL Example 6.4 (Ripple Counter)

```
// 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
```

```
// 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
```

<u>BCD Ripple Counter:</u>

•A ripple counter is an asynchronous sequential circuit. Signals that affect the flip-flop transition depend on the way they change from 1 to 0.

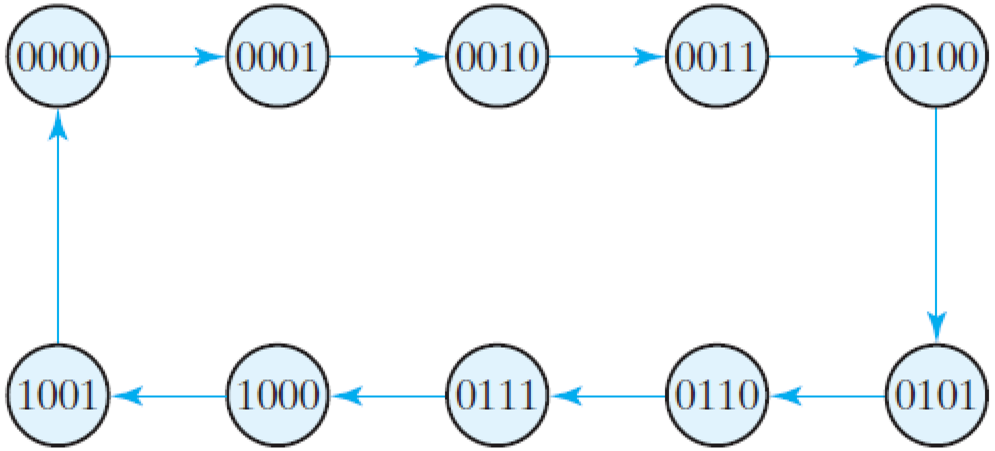


FIGURE: State diagram of a decimal BCD counter

•Remember that when the *C input goes from 1 to 0,* the flip-flop is set if *J = 1, is cleared if K = 1, is complemented if J = K = 1, and is left unchanged if J = K = 0.*

Counting sequence table

Q_8	Q_4	Q_2	Q_1
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
1	0	0	1
0	0	0	0

•*Q1 changes state after each clock* pulse.

•*Q2 complements every time Q1 goes from 1 to 0, as long as Q8 = 0.*

•*When Q8* becomes 1, *Q2 remains at 0.*

•*Q4 complements every time Q2 goes from 1 to 0.*

•*Q8 remains* at 0 as long as *Q2 or Q4 is 0. When both Q2 and Q4 become 1, Q8 complements when Q1* goes from 1 to 0.

•*Q8 is cleared on the next transition of Q1.*

•The BCD counter of above Fig. is a *decade counter, since it counts from 0 to 9. To* count in decimal from 0 to 99, we need a two-decade counter. To count from 0 to 999, we need a three-decade counter.

•A three-decade counter is shown in the following Fig. The inputs to the second and third decades come from *Q8 of the* previous decade. When *Q8 in one decade goes from 1 to 0, it triggers the count for the* next higher order decade while its own decade goes from 9 to 0.

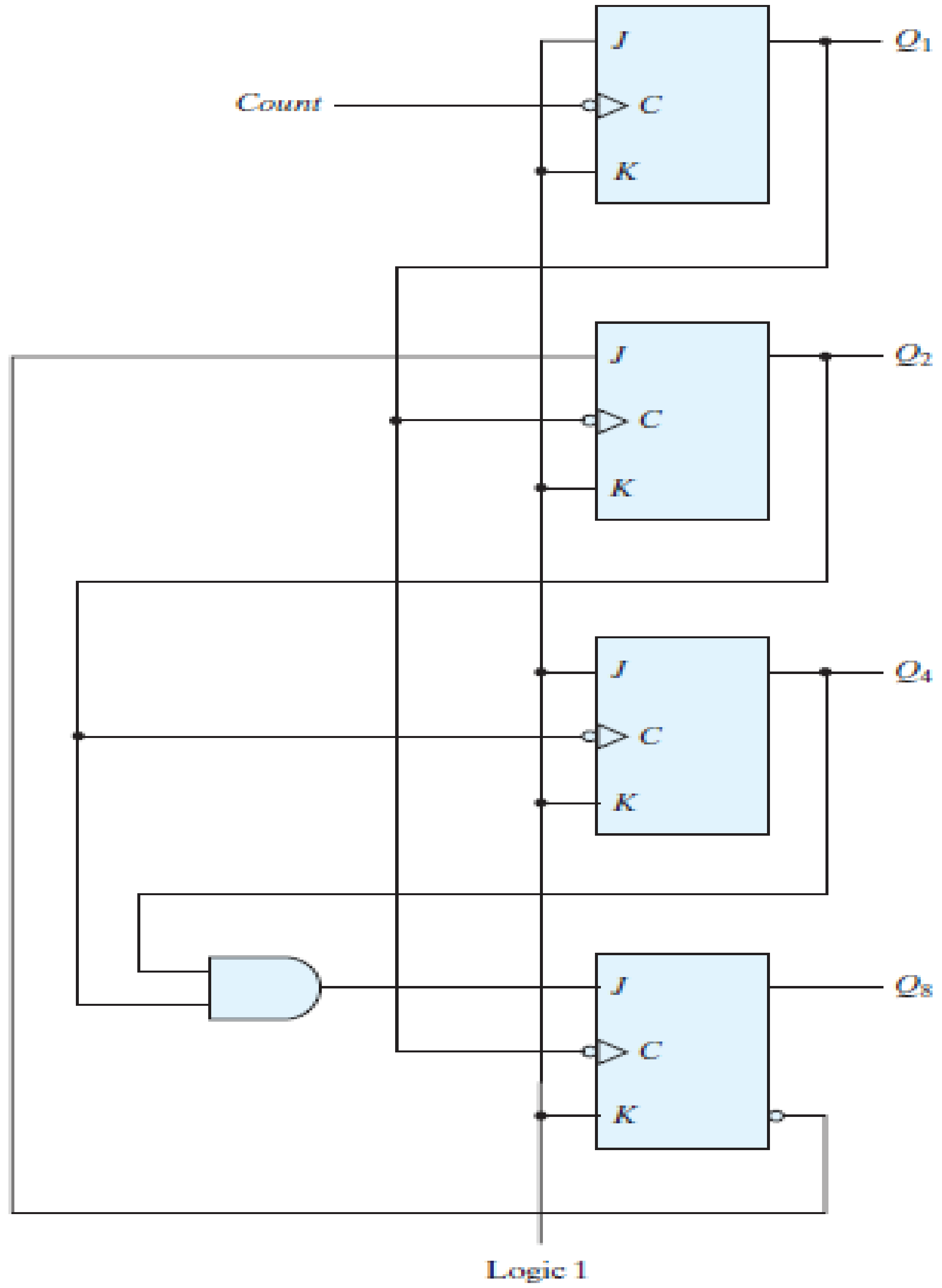


FIGURE: BCD ripple counter

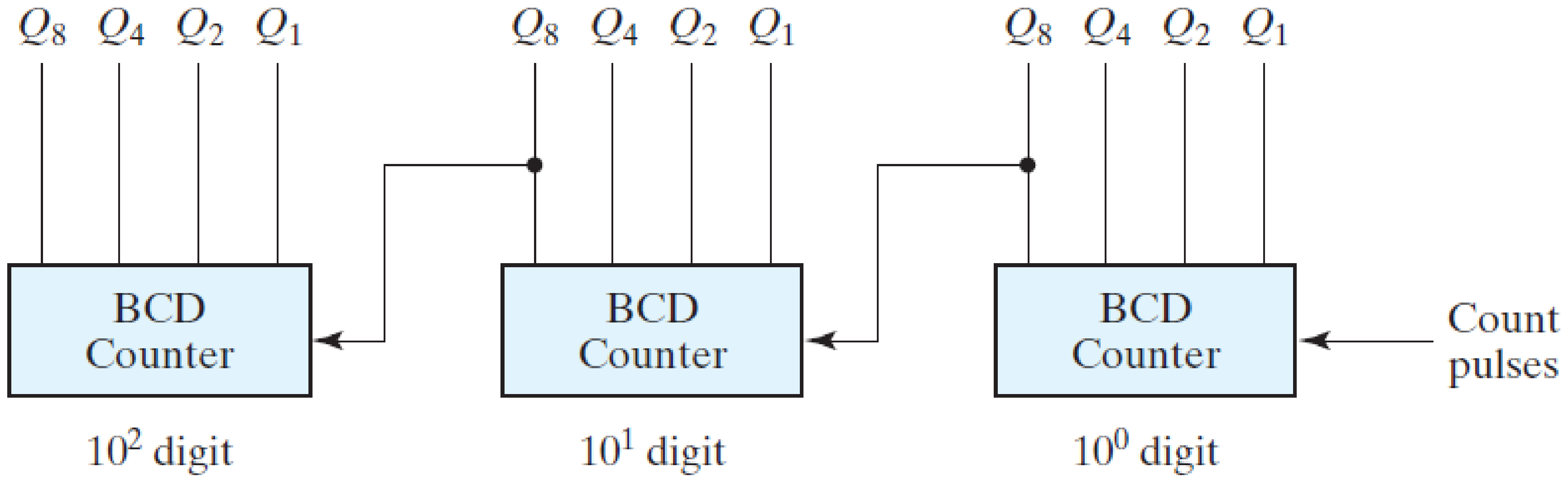


FIGURE: Block diagram of a three-decade decimal BCD counter.

- **Design of BCD ripple up counter using asynchronous CLR (active low) input.**

- **What is a divide by n counter?**

SYNCHRONOUS COUNTERS

•Synchronous counters are different from ripple counters in that clock pulses are applied to the inputs of all flip-flops.

•A common clock triggers all flip-flops simultaneously, rather than one at a time in succession as in a ripple counter.

Binary Counter:

•In a synchronous binary counter, the flip-flop in the least significant position is complemented with every pulse.

•*A flip-flop in any other position is complemented when all the bits in the lower significant positions are equal to 1.*

•For example, if the present state of a four-bit counter is *A3A2A1A0 = 0011, the next* count is 0100.

•*A0 is always complemented. A1 is complemented because the present* state of *A0 = 1. A2 is complemented because the present state of A1A0 = 11. However, A3 is not complemented, because the present state of A2A1A0 = 011, which does not* give an all-1's condition.

Q_3	Q_2	Q_1	Q_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
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

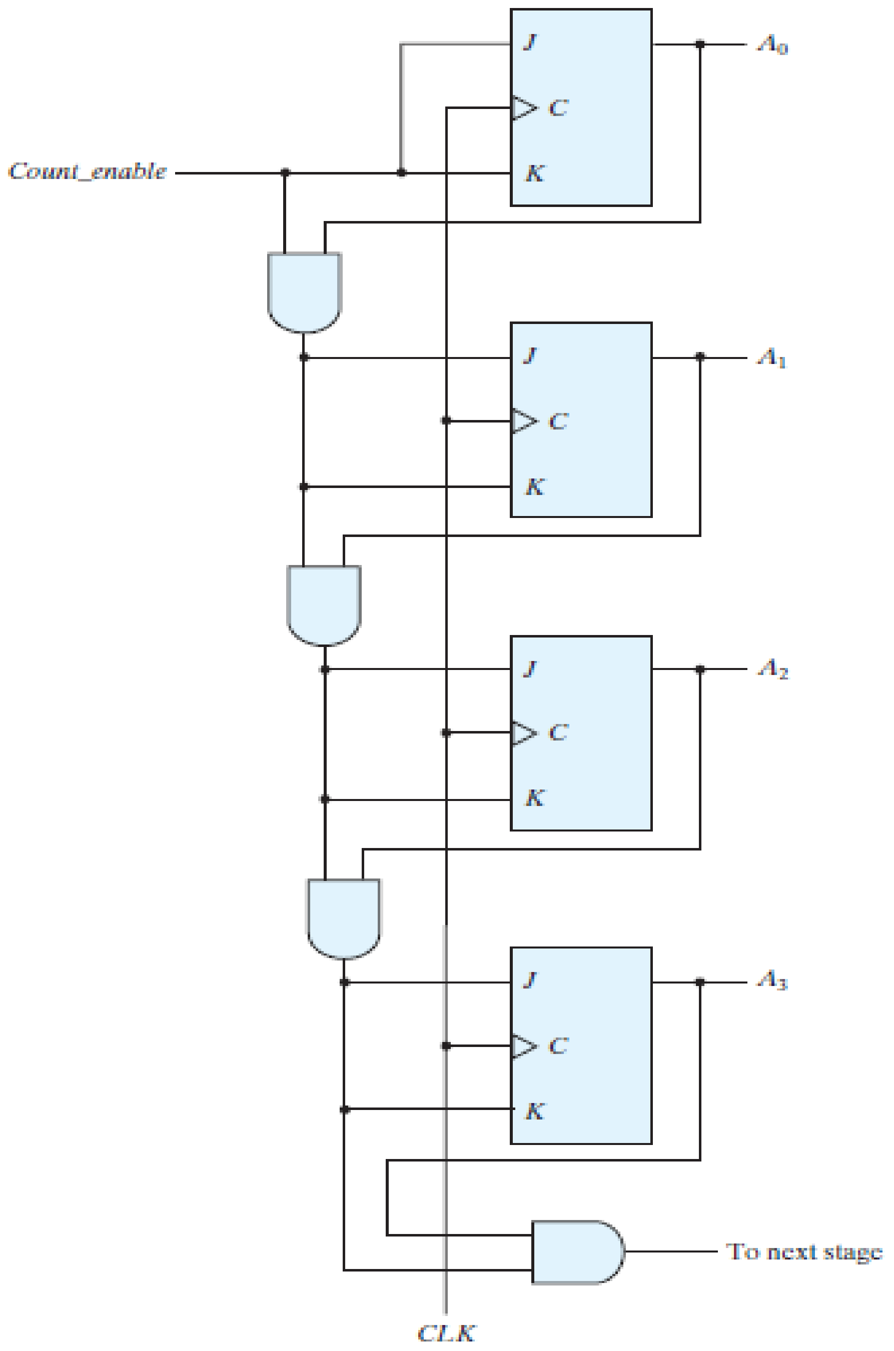


FIGURE: Four-bit synchronous binary counter

•It is possible to design **a countdown counter** in the usual manner, but the result is predictable by inspection of the downward binary count**. (Home work)**

•The bit in the least significant position is complemented with each pulse. ***A bit in any other position is complemented if all lower significant bits are equal to 0***.

•For example, the next state after the present state of 0100 is 0011. The least significant bit is always complemented. The second significant bit is complemented because the first bit is 0. The third significant bit is complemented because the first two bits are equal to 0. But the fourth bit does not change, because not all lower significant bits are equal to 0.

•A countdown binary counter can be constructed as shown in the above Fig., **except that the inputs to the AND gates must come from the complemented outputs, instead of the normal outputs, of the previous flip-flops**.

Up–Down Binary Counter:

The two operations can be combined in one circuit to form a counter capable of counting either up or down. The circuit of an up–down binary counter using *T flip-flops is shown in Fig. 6.13 (page no. 292) . It has an up control* input and a down control input. When the **up input is 1**, the circuit **counts up**, since the *T inputs receive their signals from the values of the previous normal outputs of the* flip-flops. When the **down input is 1** and the **up input is 0**, the circuit **counts down**, since the complemented outputs of the previous flip-flops are applied to the *T inputs.* When **the up and down inputs are both 0, the circuit does not change** state and remains in the same count. When the up and down inputs are **both 1, the circuit counts up**. This set of conditions ensures that only one operation is performed at any given time. **Note that the up input has priority over the down input.**

BCD Synchronous Counter:

•Design a BCD synchronous counter using positive edge triggered T FF.

State Table for BCD Counter

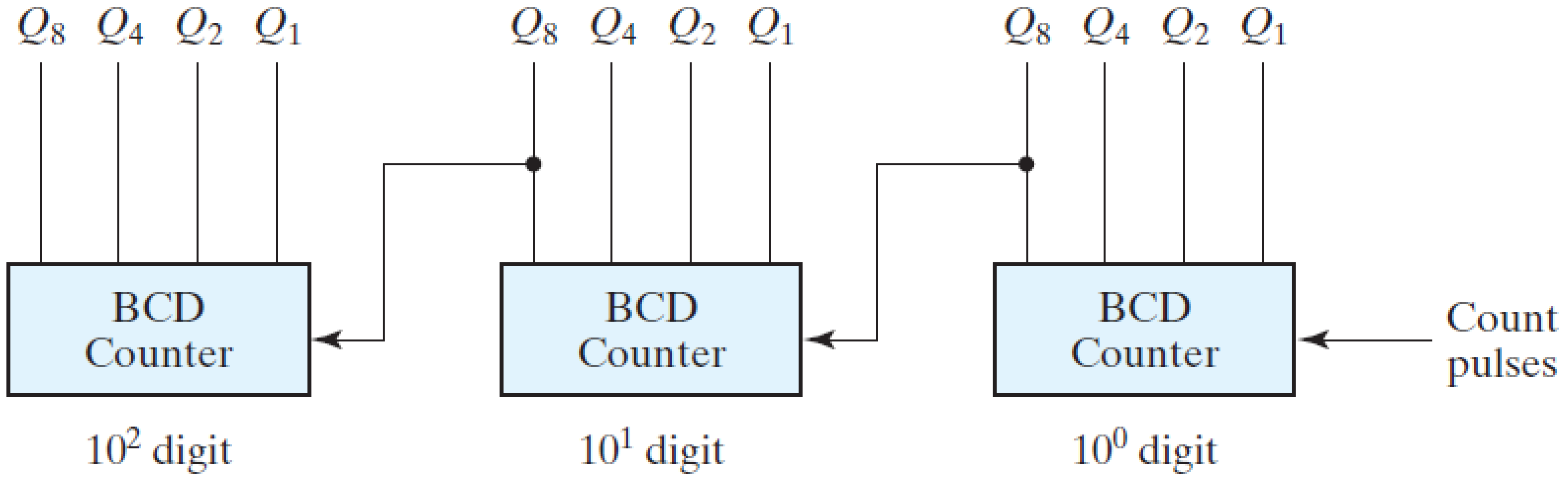
Present State				Next State				Output	Flip-Flop Inputs			
Q_8	Q_4	Q_2	Q_1	Q_8	Q_4	Q_2	Q_1	y	TQ_8	TQ_4	TQ_2	TQ_1
0	0	0	0	0	0	0	1	0	0	0	0	1
0	0	0	1	0	0	1	0	0	0	0	1	1
0	0	1	0	0	0	1	1	0	0	0	0	1
0	0	1	1	0	1	0	0	0	0	1	1	1
0	1	0	0	0	1	0	1	0	0	0	0	1
0	1	0	1	0	1	1	0	0	0	0	1	1
0	1	1	0	0	1	1	1	0	0	0	0	1
0	1	1	1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

•Also shown in the table is an output *y, which is equal to 1 when the present state is 1001. In this way, y can* enable the count of the next-higher significant decade while the same pulse switches the present decade from 1001 to 0000.

$$\begin{aligned}
T_{Q1} &= 1 \\
T_{Q2} &= Q_8'Q_1 \\
T_{Q4} &= Q_2Q_1 \\
T_{Q8} &= Q_8Q_1 + Q_4Q_2Q_1 \\
y &= Q_8Q_1
\end{aligned}$$

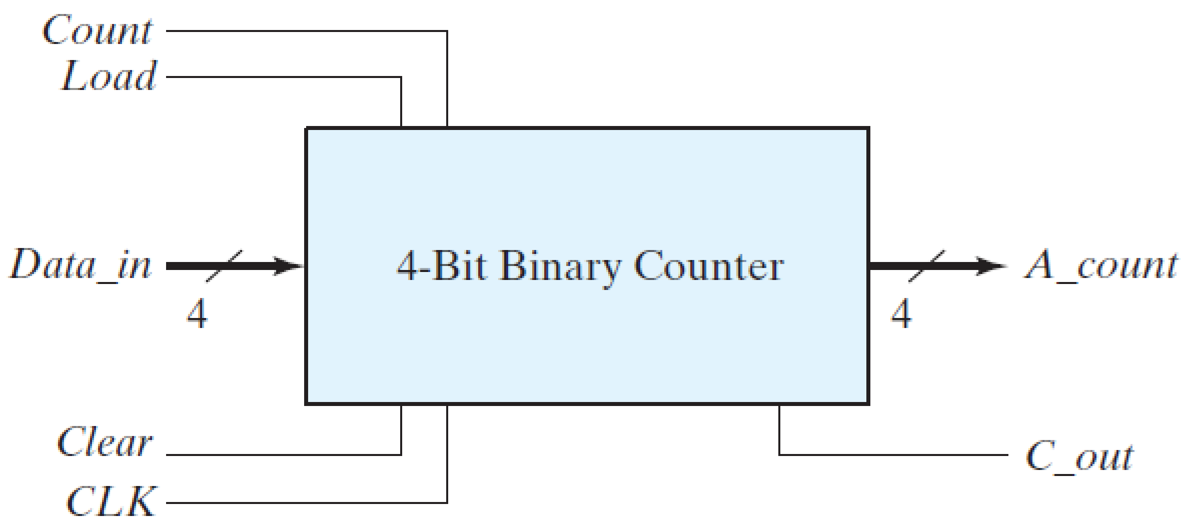
- Please write the circuit diagram for above design.

- The cascading is done as in Fig. 6.11 , except that output *y must be connected* to the count input of the next-higher significant decade.



HDL Example 6.3 (Synchronous Counter)

```
// 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_b      // Active low
);
assign C_out = Count && (~Load) && (A_count == 4'b1111);
always @ (posedge CLK, negedge Clear_b)
   if (~Clear_b)               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
```



Function Table for the Counter of Fig. 6.14

Clear	CLK	Load	Count	Function
0	X	X	X	Clear to 0
1	$\uparrow$	1	X	Load inputs
1	$\uparrow$	0	1	Count next binary state
1	$\uparrow$	0	0	No change

<u>OTHER COUNTERS:</u>

•Counters can be designed to generate any desired sequence of states.

•A divide-by- *N* counter (also known as a modulo- *N counter) is 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 are used to generate timing signals to control the sequence of operations in a digital system.

•Counters can also be constructed by means of shift registers (Ring and Johnson counters).

Counter with Unused States

Design a synchronous counter using positive edge triggered T FFs to generate the following sequence: 0, 1, 2, 4, 5, 6.

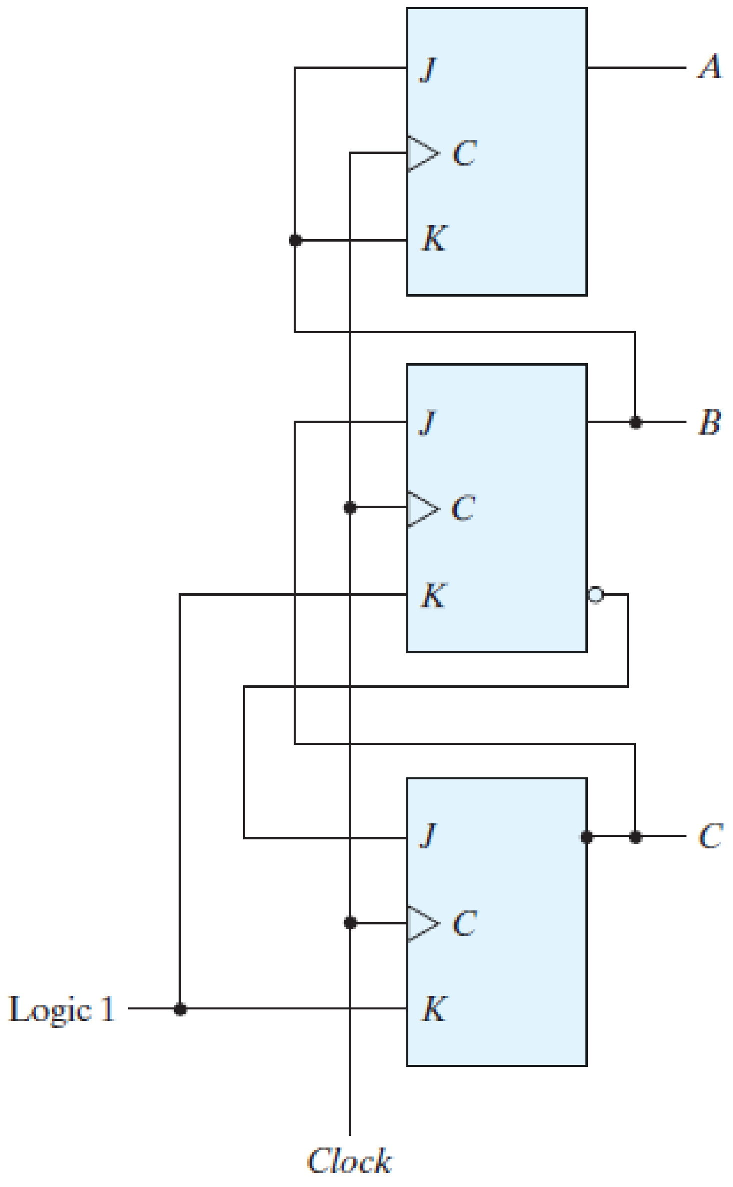
$$\begin{tikzcd}
 & 000 \arrow[dl] & & 111 \arrow[ll] \\
001 \arrow[d] & & 110 \arrow[ul] & \\
010 \arrow[dr] & & 101 \arrow[u] & \\
 & 100 \arrow[ur] & & 011 \arrow[ll]
\end{tikzcd}$$

Fig: State transition diagram

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

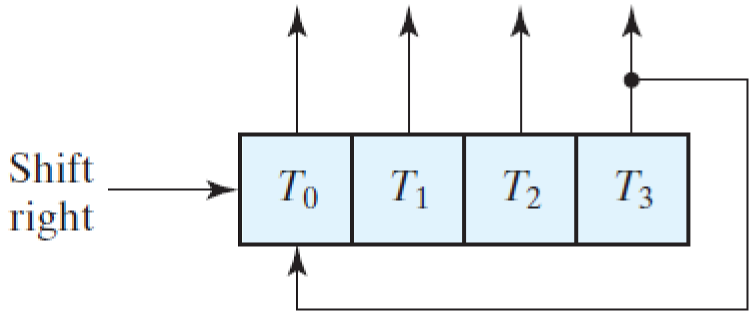
$$J_A = B \qquad K_A = B$$
$$J_B = C \qquad K_B = 1$$
$$J_C = B' \qquad K_C = 1$$



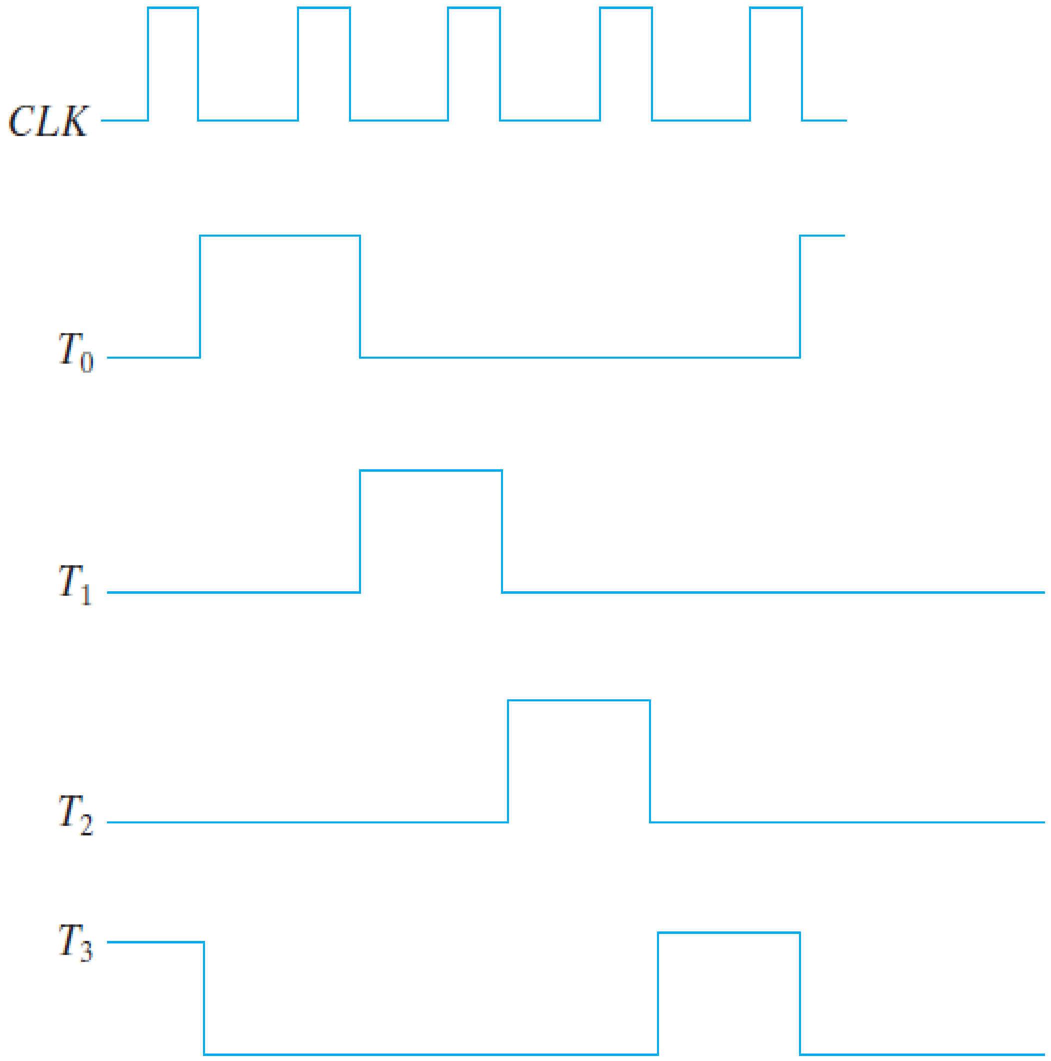
(a) Logic circuit diagram

Ring Counter:

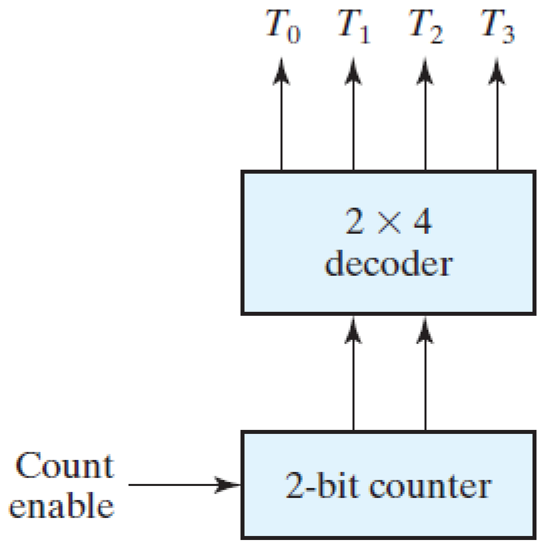
- A *ring counter is a circular shift* register with only one flip-flop being set at any particular time; all others are cleared.
- The single bit is shifted from one flip-flop to the next to produce the sequence of timing signals.
- Figure shows a four-bit shift register connected as a ring counter.
- The initial value of the register is 1000 and requires Preset/Clear flip-flops.



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



(b) Sequence of four timing signals

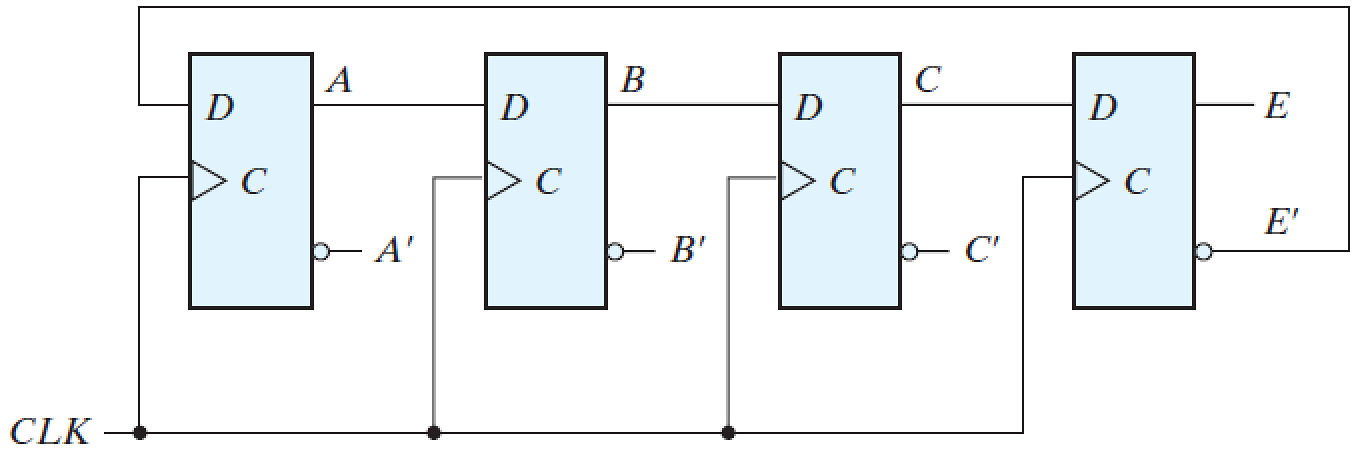


(c) Counter and decoder

- For an alternative design, the timing signals can be generated by a two-bit counter that goes through four distinct states.
- The decoder shown in Fig. (c) decodes the four states of the counter and generates the required sequence of timing signals.
- To generate 2^*n timing signals, we need either a shift register with 2^n flip-flops or an n -bit binary counter together with an n-to-2^n-line decoder.*

<u>Johnson Counter (*switch-tail or twisted ring counter*):</u>

- A *k -bit ring counter circulates a single bit among the flip-flops to provide k distinguishable* states.
- The number of states can be doubled if the shift register is connected as a *switch-tail ring counter.*
- *A switch-tail ring counter is a circular shift register with the* complemented output of the last flip-flop connected to the input of the first flip-flop.



(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

FIGURE: Construction of a Johnson counter

- Johnson counters can be constructed for any number of timing sequences.
- The number of flip-flops needed is one-half the number of timing signals.
- The number of decoding gates is equal to the number of timing signals, and only two-input gates are needed.