## 5.1 REGISTERS

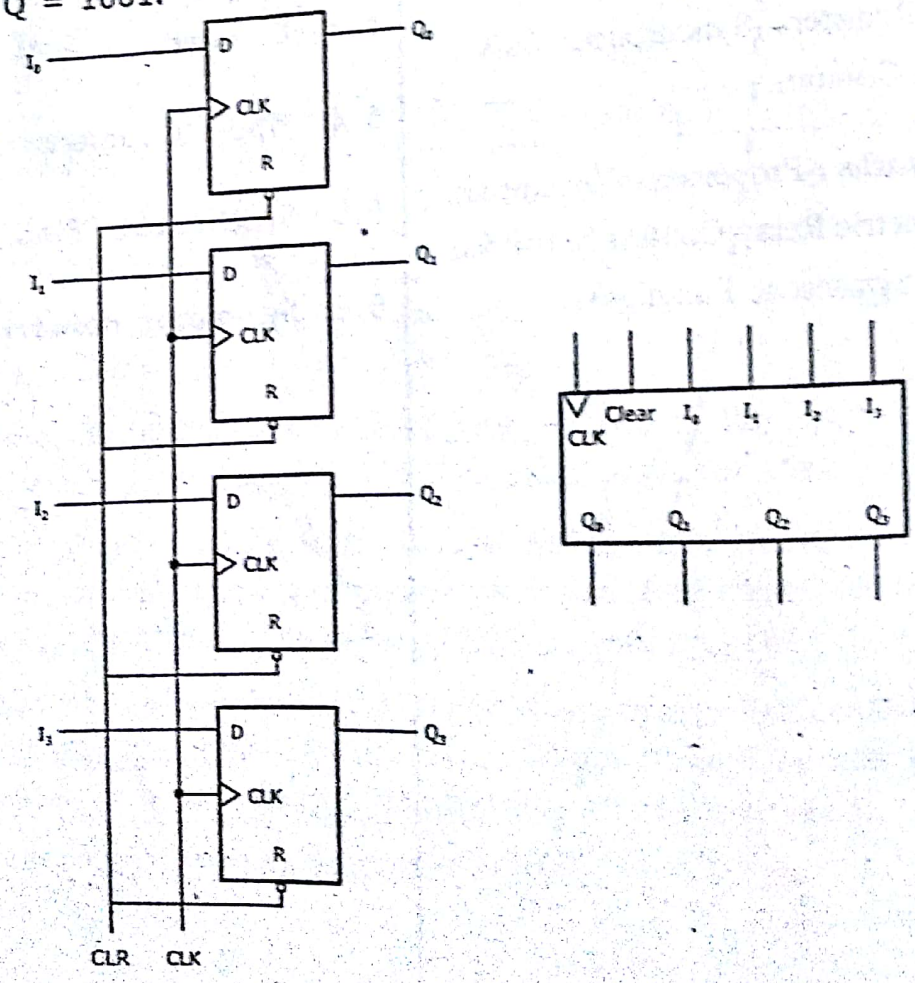
A single flip-flop can store a single bit of information. So, n flip-flops will store n-bits of information. The device consisting of n flip-flops and capable of storing and passing n-bits of data is called a register.

Depending upon the functions to be performed, registers are categorized as following types,

1) Buffer Registers.
2) Shift Registers.

### 5.1.1 Buffer Register

Fig. 5.1.1(a) shows the simplest 4-bit register known as buffer register constructed with four D flip-flops. Each D flip-flop is triggered with a common positive edge clock pulse. The input I-bits ($I_0I_1I_2I_3$) set up the flip-flops for loading. Therefore when the first clock pulse arrives, all the I-bits are loaded into D-inputs of all the flip-flops. Then the stored binary information is $Q = Q_3Q_2Q_1Q_0 = I_3I_2I_1I_0$. For example, if we want to store a 4-bit binary information 1001 in the buffer register, then these bits are applied at I-bits as $I_3 = 1$, $I_2 = 0$, $I_1 = 0$ and $I_0 = 1$. Whenever the first clock pulse is applied, then the stored binary word is Q = 1001.




(a) Logic Diagram (b) Logic Symbol

**Fig. 5.1.1** A simple 4 Bit Buffer Register

## 5.1.2 Registers With Parallel Load

- Transferring new information into a register is called as loading or updating the register. Parallel load refers to loading all the bits to the register simultaneously with a common clock pulse.
- For the design in Fig. 5.1.1(a), the clock must be prevented from reaching CLK inputs of all the flip flops, if the contents of the register must be left unchanged. This can be accomplished by including a separate control signal load.
- A four bit data buffer register with a load control input is shown in Fig. 5.1.2. The load control input is directed through gates and determines which action to be taken with each clock pulse.

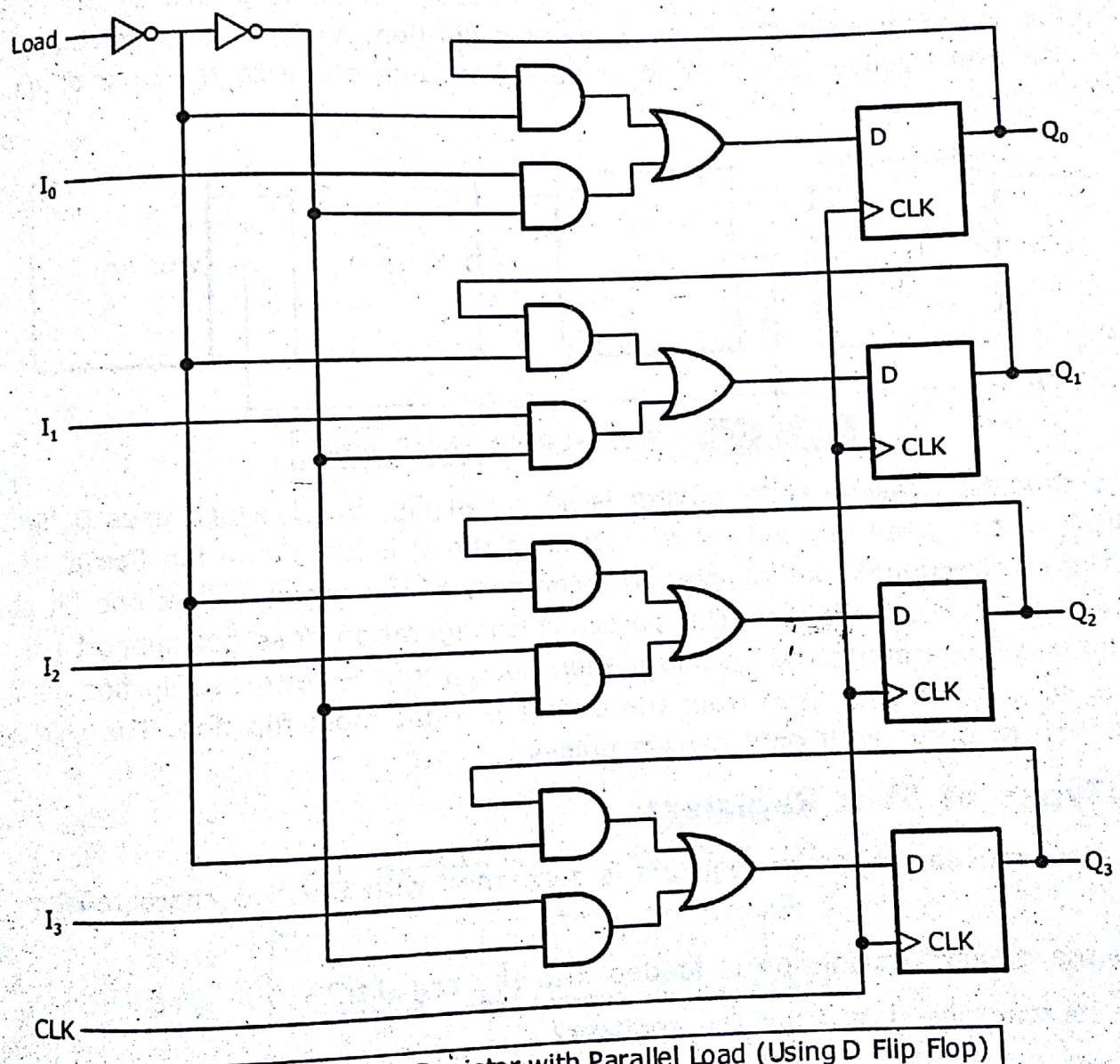



**Fig. 5.1.2** Four Bit Register with Parallel Load (Using D Flip Flop)

### Working Operation

- Whenever LOAD signal is at LOW, then I-bits cannot reach the flip-flops. The contents of register are unchanged at this condition.

- When LOAD signal is HIGH, the I-bits are transmitted to the data inputs and the flip-flops are ready for loading in the presence of clock pulse. The stored binary information is,

$$Q = Q_3Q_2Q_1Q_0 = I_3I_2I_1I_0$$

- Thus, load input determines whether the next pulse will accept new information or leave the information in the register.

## 5.2 SHIFT REGISTERS

Shift registers are capable of shifting their binary information to their neighboring cell (either left or right), in the predetermined direction.

A shift register consists of a cascaded flip-flops, connected such that the output of one flip-flop is connected to the input of the next flip-flop. All flip-flops receive a common clock pulse which causes the shifting of data from one stage to the next stage.

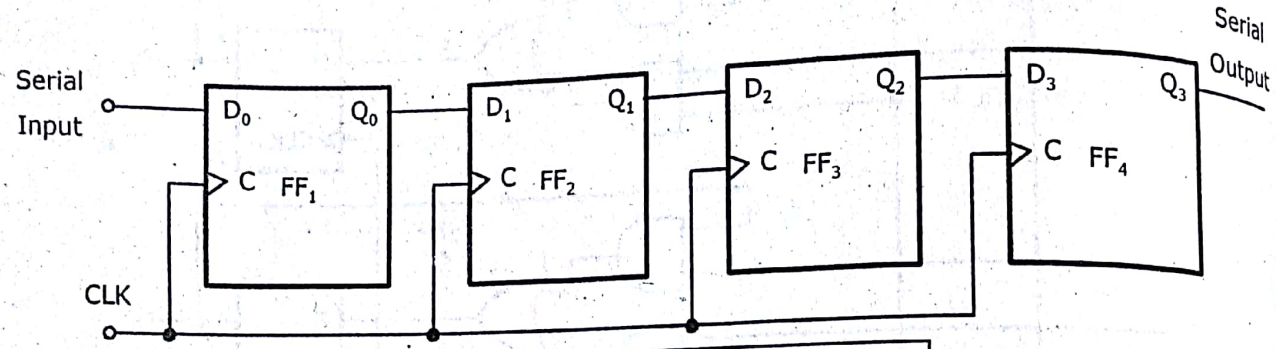



**Fig. 5.2.1** 4 Bit Shift Right, Shift Register

The simplest possible shift register is shown in Fig. 5.2.1 which uses D flip flops. The output of the given flip flops is connected to the D input of the flip flop at its right. This is the unidirectional shift register. The contents of the register shift one bit position to the right at each clock pulse. (This particular configuration does not support left shift). The serial input determines the bit information goes from the leftmost flip flop during the shift. The serial output is taken from the output of right most flip flop. The shifting can be controlled to occur with only certain pulses.

### 5.2.1 Types of Shift Registers

The data movement in the registers is concerned with the two characteristics. They are,

1) How the binary information is loaded into the register?

2) How to read the data from the register?

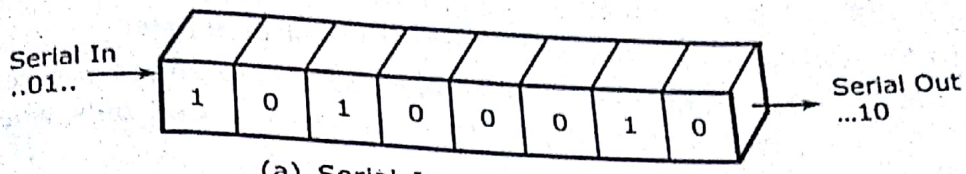
According to that, the registers are classified into four possible modes of operations. They are,

1) **Serial In - Serial Out :** The data is loaded into and read from the shift register serially as shown in Fig. 5.2.2(a).

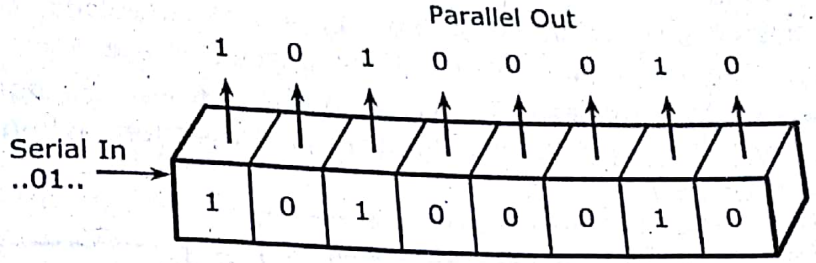
3) **Serial In - Parallel Out :** The data is loaded into the register serially but read in parallel (i.e., data is available from all bits simultaneously as shown in Fig. 5.2.2(b).

3) **Parallel In - Serial Out :** The data is loaded in parallel, i.e., the bits are entered simultaneously in their respective stages and read serially as shown in Fig. 5.2.2(c).

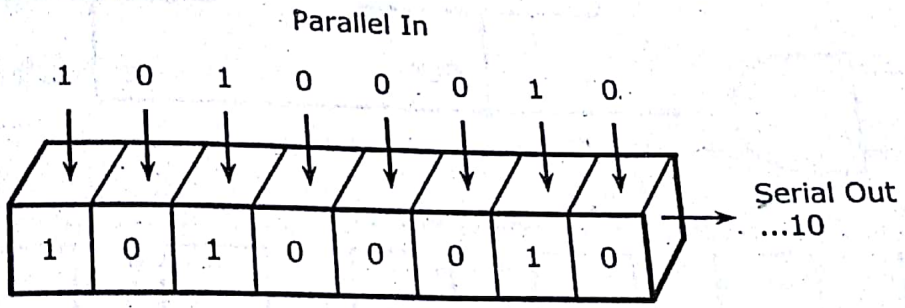
4) **Parallel In - Parallel Out :** The data is loaded and read from the register in parallel, i.e., all bits are loaded simultaneously and read simultaneously as shown in Fig. 5.2.2(d).



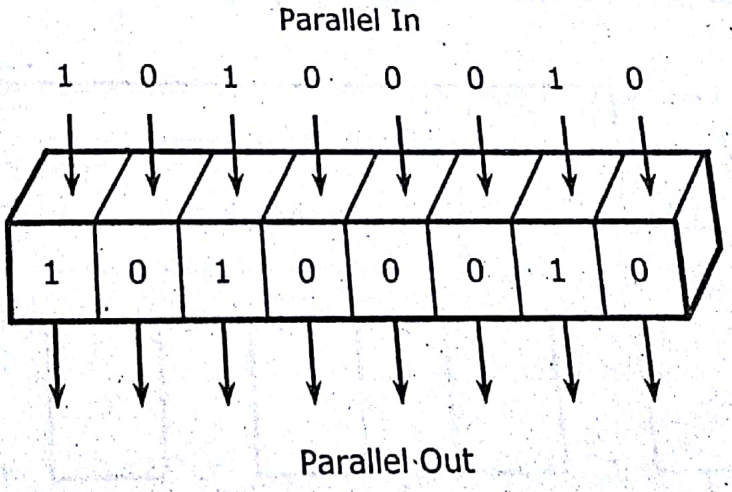

(a) Serial In - Serial Out (SISO)




(b) Serial In - Parallel Out (SIPO)




(c) Parallel In - Serial Out (PISO)




(d) Parallel In - Parallel Out (PIPO)

**Fig. 5.2.2** Shift Register Operations

## 5.2.2 Serial Transfer

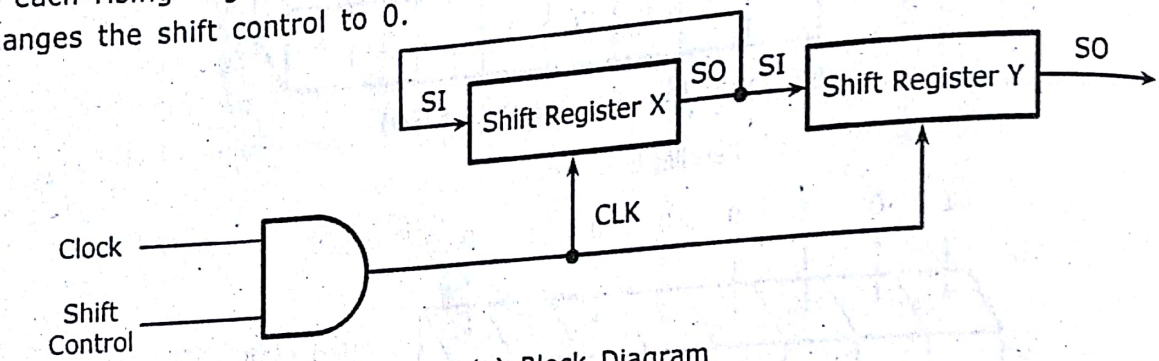
Shift registers are used to serially transfer the information from one register (source) to another register (destination).

Block diagram (Fig. 5.2.3) shows that information from register X is transferred to register Y. The single serial output (SO) of register X is connected to the single serial input (SI) of register Y. (Here the SO of source register X is also connected to SI of the same register to prevent the information). The initial content of register Y will be lost.

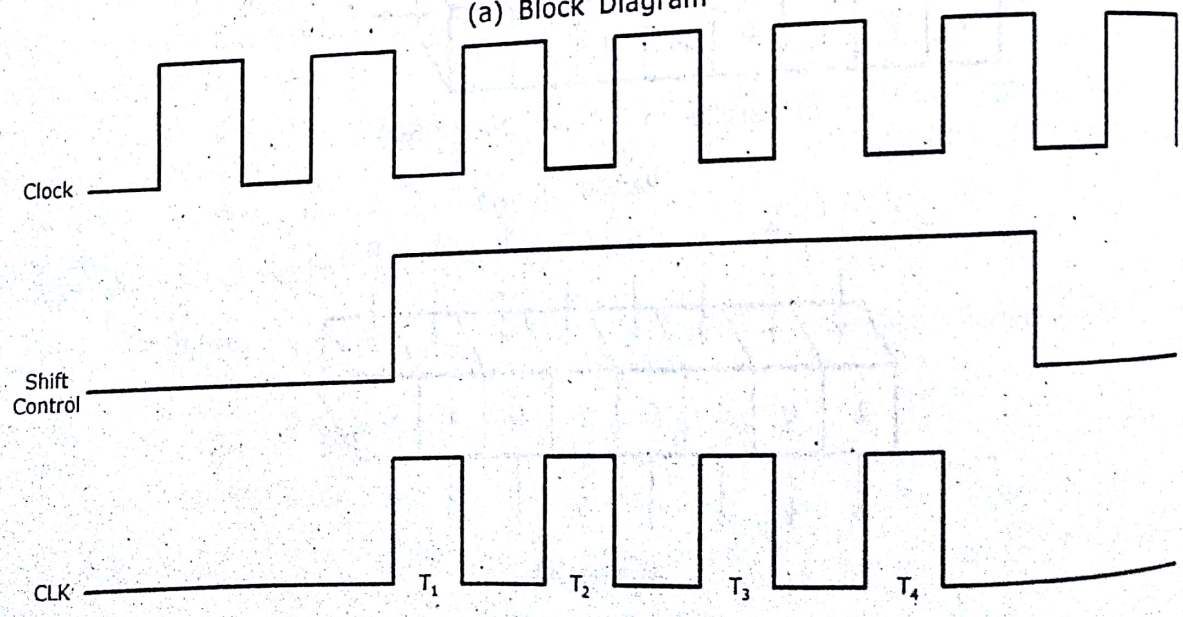
It can also be prevented by transferring into the third register.

The shift control input determines when and how many times the registers are shifted. (Here, AND gate allows clock pulses to pass into the CLK only when the shift control is active).

Assume, that X and Y registers are of 4 bits each. Timing diagram (shown in Fig. 5.2.3(b)) shows that shift control signal is fixed for a 4 clock pulses. The four pulses find the shift control signal in the active state so the output of the AND gate connected to the CLK inputs produces four pulses : $T_1$, $T_2$, $T_3$ and $T_4$. A shift in both registers occurs at each rising edge of the pulse. Shift registers are disabled when the fourth pulse changes the shift control to 0.




(a) Block Diagram




(b) Timing Diagram

**Fig. 5.2.3** Serial Transfer from Register X to Register Y

Let, before shift, content of X = 1011 and that of Y = 0010. The four steps involved in serial transfer are shown in Fig. 5.2.4.

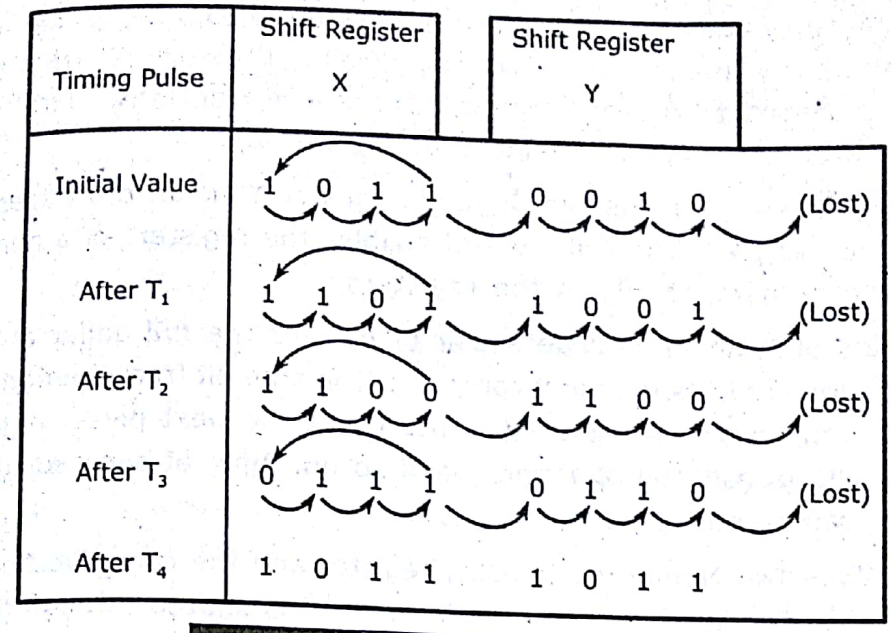



**Fig. 5.2.4** Serial Transfer Example

With the first pulse $T_1$, the rightmost bit of register X is shifted into the register Y and is also re circulated into the leftmost position of X. After the fourth shift, shift control goes to 0 and registers X and Y both have the value 1011. Thus contents of X are copied into Y and contents of X remain unchanged.

## 5.2.3 Serial Addition

The serial adder is discussed here to explain the serial mode of operation. The Fig. 5.2.5 shows the circuit for serial adder.

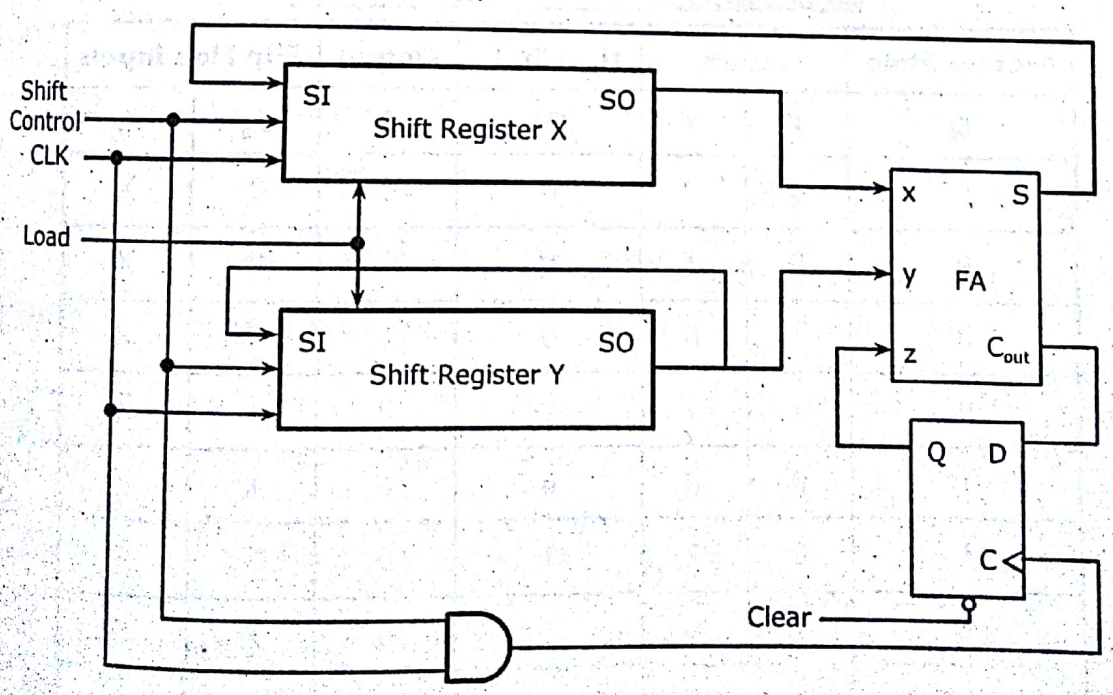



**Fig. 5.2.5** Serial Adder

The two binary numbers to be added serially are stored in two shift registers X and Y having "load" facility. A full adder adds a pair of bits starting from the least significant bits in X and Y. The carry out of the full adder is transferred to a D flip flop whose output is then used as the carry input to the next pair of bits. The sum bit from the S output of full adder is transferred to the third register (here it is transferred into X, when load input becomes 0 after addition, the result is in register X).

**Working :** Initially, register X holds the augend, register Y holds the addend and carry $C_{out}$ of flip flop is cleared to 0. The shift control enables the registers of a number of clock pulses equal to the number of bits in the registers.

The SO outputs of X and Y provide a pair of bits for the full adder at x and y and output Q of the flip-flop provides the input carry at z. The sum bit from S enters the leftmost flip flop of X and output carry is transferred into Q. At the next pulse, registers shifted and a new pair of bits transferred to x and y and so on. Also, at succeeding clock pulse, a new sum bit is transferred to X.

**How to Add more Than Two Numbers :** Initially, register and the carry flip flop are cleared to 0 and then the first number is added from Y. While Y is shifted through the full adder, a second number is transferred to it through its SI. This is added with the content of X while a third number is transferred serially into Y. This can be repeated.

**Example Problem 5.1**

**Design a Sequential circuit as serial adder using JK flip flop.**

*Sol. :*

To design the circuit, first we derive the state table for serial adder (Table 5.2.1).

**Table 5.2.1** 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 |

The present state Q is the current value of the carry which is added together with inputs X and Y to produce the sum bit in output S. The next state $Q^+$ is the output carry after adding X and Y.

To design the circuit using JK flip flop, it is necessary to find out 3 input and K input with the help of state table and excitation table of JK flip flop. Table. 5.2.1 also have a column for J and K inputs. The simplified flip flop input and output(s) equations are,

$$J_Q = xy$$

$$K_Q = \overline{x}\,\overline{y} = \overline{(x + y)}$$

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

Fig. 5.2.6 shows the implementation of serial adder using JK flip-flop.

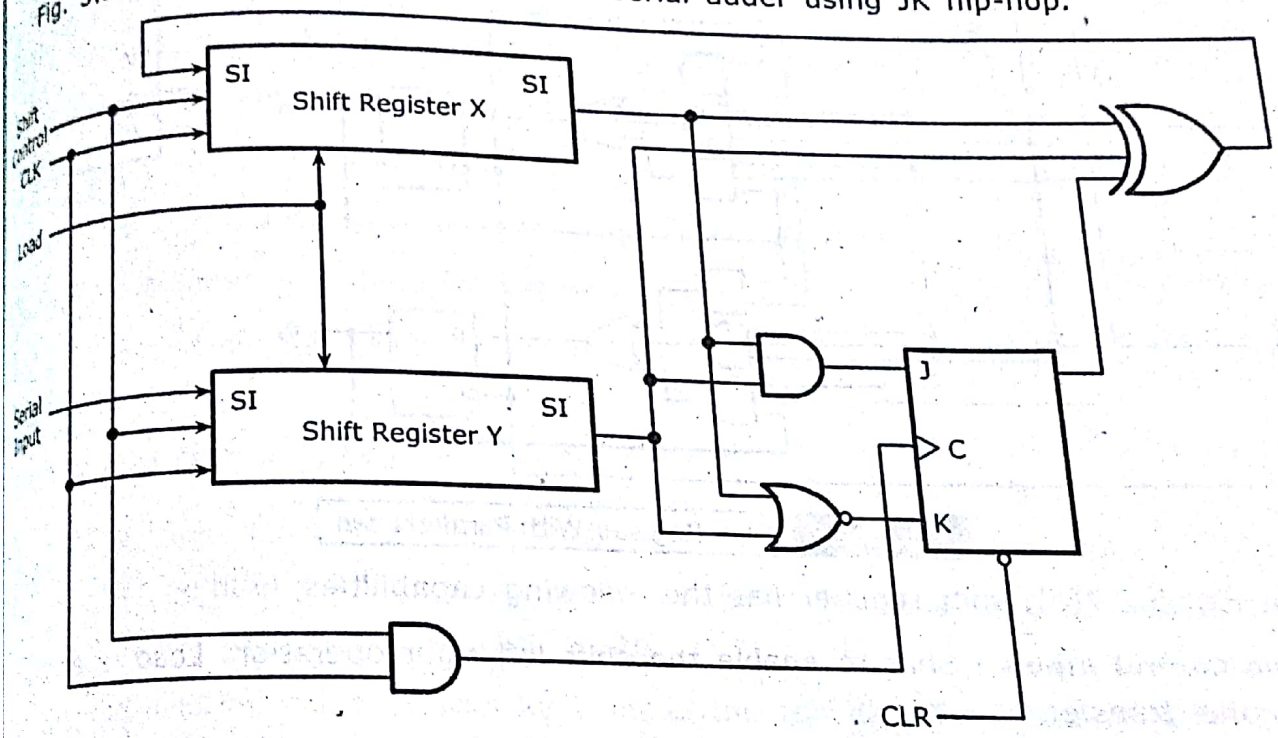



**Fig. 5.2.6** Serial Adder Using JK Flip Flop

## 5.2.4 Shift Register With Parallel Load

- Suppose if all the outputs of a shift register are accessible, then information shifted in serially can be read but parallel from the outputs of all of the flip flops.
- If this register is supplied with a parallel form load input, then information entered in parallel can be read out serially by shifting out the data in the register.
- Thus if a shift register having accessible flip flop outputs provided with a parallel load capability, then the incoming serial input can be converted to parallel output and vice versa.

❖ **Logic Diagram :** Fig. 5.2.7 (a) depicts the logic diagram of shift register with parallel load capability and its logic symbol is shown in Fig. 5.2.7(b)

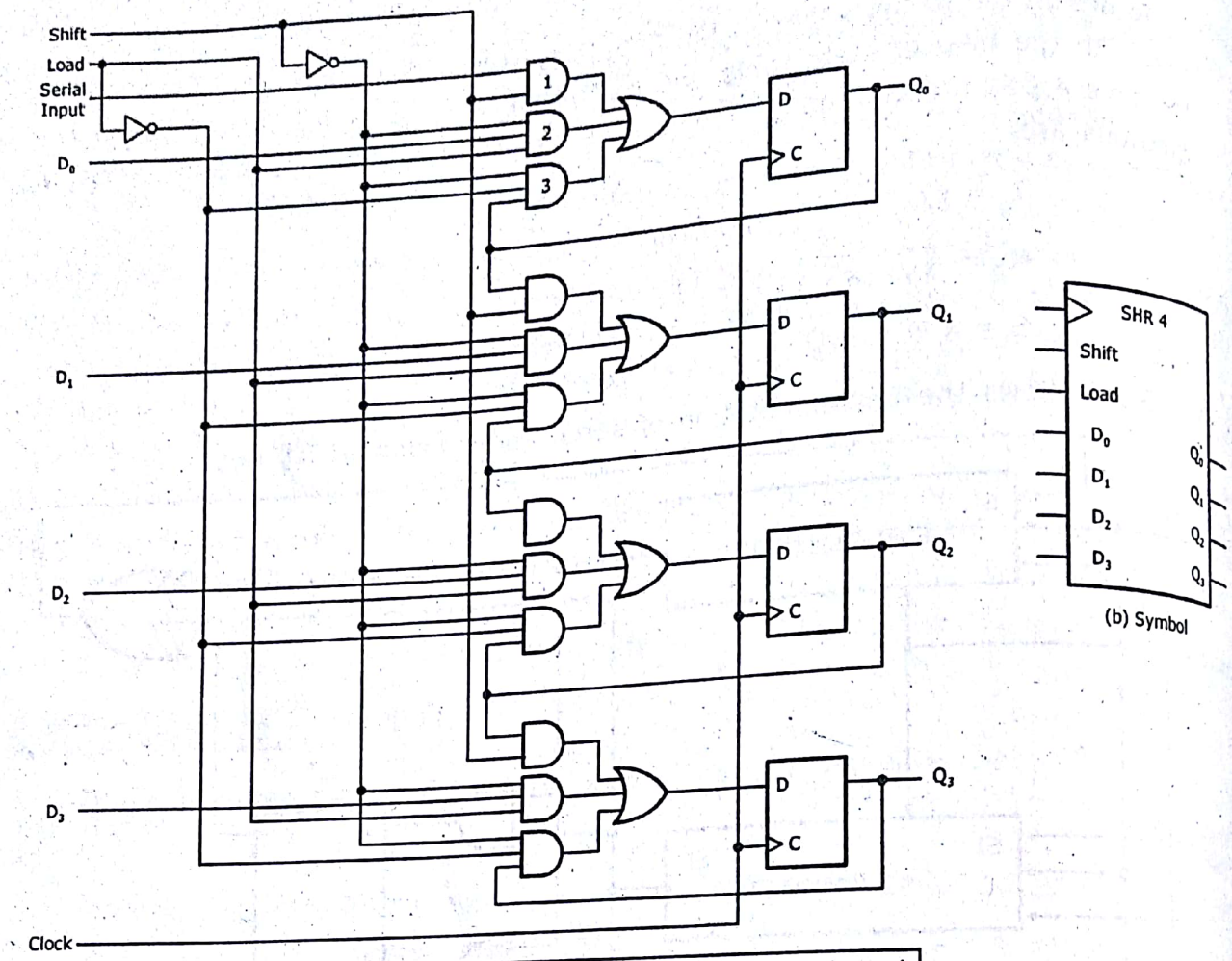



**Fig. 5.2.7** Shift Register With Parallel Load

In Fig. 5.2.7(a), shift register has the following capabilities,

1) **Two Control Inputs :** Shift to enable the shift left/right operation. Load to enable a parallel transfer.

2) **Clock :** To synchronize the operations.

❖ As seen in Fig. 5.2.7(a), each stage of a register consists of a D flip flop, an OR gate and three AND gates,

- ➢ The first AND gate enables the shift operation.
- ➢ The second AND gate enables the input data.
- ➢ The third AND gate restores the contents of the register when no operation is required.

❖ **Working Operation :** The mode of operations of the register is controlled through the control inputs shift and load as specified in Table 5.2.2.

**Table 5.2.3** Function Table for the Register of Figure 5.2.7

| Shift | Load | Operation |
|---|---|---|
| 0 | 0 | No change |
| 0 | 1 | Load parallel data |
| 1 | x | Shift down from $Q_0$ to $Q_3$ |

These 3 cases can be explained clearly as below,

**Case 1 (When Shift = 0, Load = 0)**

- The third AND gate in each stage is enabled.
- Output of each flip-flop is applied to its D input.
- A positive transition of the clock restores the contents of the register.
- Finally, no change in output occurs.

**Case 2 (When Shift = 0, Load = 1)**

- The second AND gate of each stage is enabled.
- The D input is applied to corresponding flip-flop D input.
- The next positive clock transition results in transferring the input data into the register.

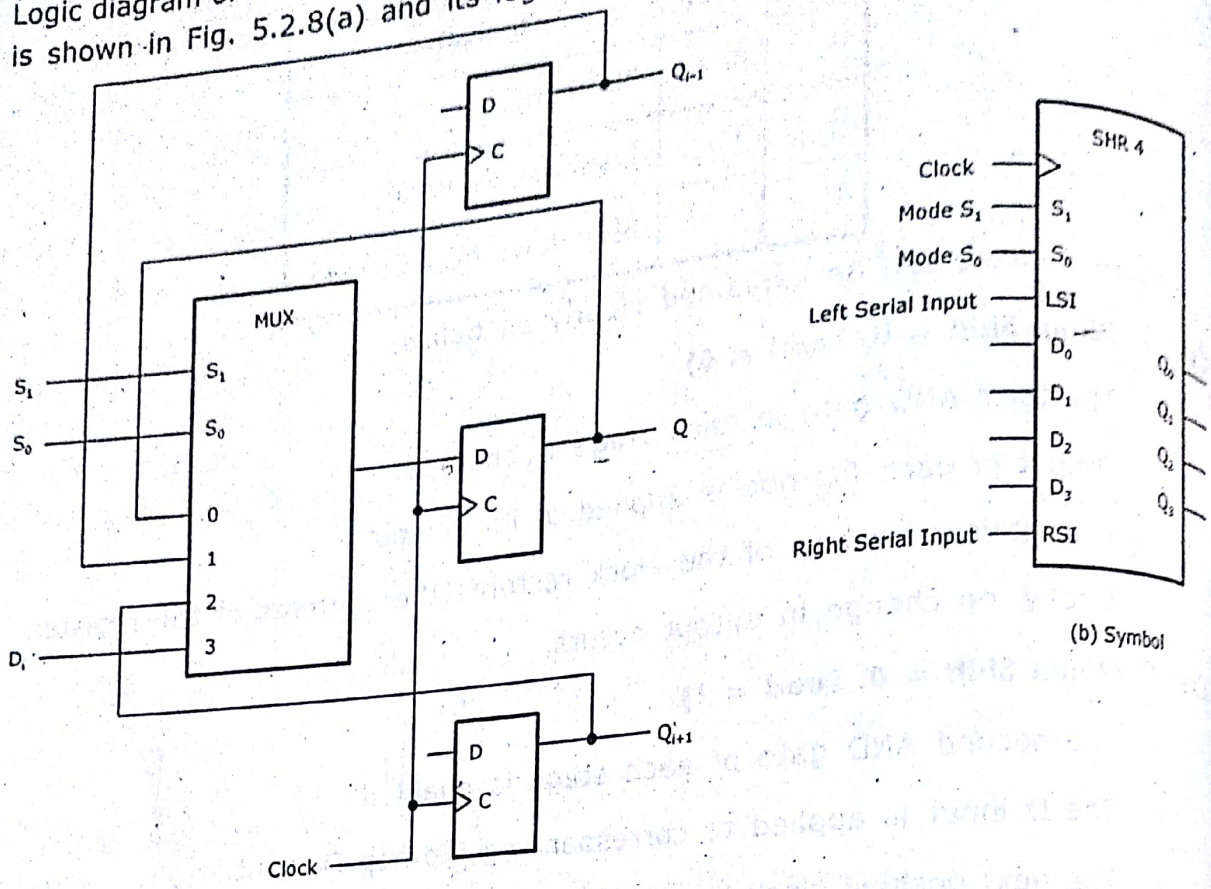
**Case 3 (When Shift = 1, Load = 0 or 1)**

- The first AND gate of each stage is enabled.
- The load here is marked as a don't care condition, since the Load input is disabled by Shift into the second AND gate,
- The shift operation works by transferring the data from the serial input SI to flip flop $Q_0$, the output of $Q_0$ is then transferred to flip flop $Q_1$, and so on down the line on a positive transition of the clock.

## 5.2.5 Bidirectional Shift Register

- A register capable of shifting in one direction only is a unidirectional shift register. A register capable of shifting in both directions is a bidirectional shift register.
- A small modifications to the logic diagram shown in Fig. 5.2.7(a) lets the data to be shifted in the upward direction. This can be accomplished by adding a fourth AND gate to each stage.
- Since four AND gates, together with the OR gate, means a multiplexer. So each stage of a bidirectional shift register consists of a D flip flop and multiplexer.

❖ Logic diagram of one stage of a bi-directional shift register with parallel load capability is shown in Fig. 5.2.8(a) and its logic symbol is shown in Fig. 5.2.8(b).




(a) Logic Diagram of One Typical Stage

(b) Symbol

**Fig. 5.2.8** Bidirectional Shift Register with Parallel Load

❖ **Working Operation :** The mode of operations of the register is controlled through the two select input lines of 4 × 1 multiplexer as specified in Table. 5.2.3.

**Table 5.2.3** Function Table for the Register of Figure 5.2.8

| Mode Control | | Register Operation |
|---|---|---|
| $S_1$ | $S_0$ | |
| 0 | 0 | No change |
| 0 | 1 | Shift down |
| 1 | 0 | Shift up |
| 1 | 1 | Parallel load |

➢ **When $S_1S_0$ = 00 :** Input 0 of the multiplexer is selected forming a path from the output of each flip flop into its own input. The next clock transition results in transfer of the binary value held previously in each flip-flop and no change of state occurs.

- **When $S_1S_0$ = 01 :** The terminal marked 1 on the multiplexer forms a path to the D input of each flip flop. This path causes a shift operation, which occurs by transfer of the serial input into the first stage, and the contents of each stage $Q_{i-1}$, are transferred into stage $Q_i$ resulting in shift down operation.
- **When $S_1S_0$ = 10 :** A shift up operation results in the serial input going into the last stage. The process is done by transfer of the value in each stage $Q_{i+1}$ into stage $Q_i$.
- **When $S_1S_0$ = 11 :** When $S_0S_1$ = 11, the binary information on each parallel input line is transferred into the corresponding flip flop, resulting in a parallel load.

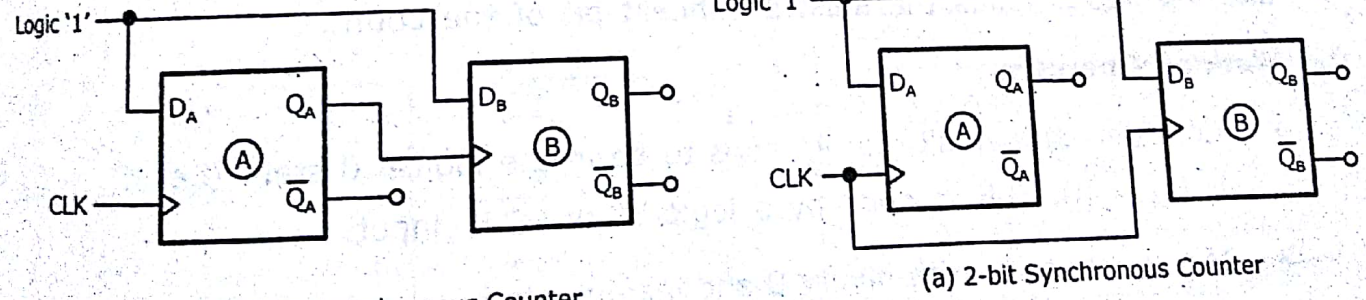
## 5.3 COUNTERS

A counter, is a set of flip-flops connected in a suitable manner to count the sequence of the input pulses arrived at its clock input.

- **Categories :** Depending upon the manner in which the flip-flops are triggered, counters can be classified into two major categories, namely,

1) Asynchronous counters (or Ripple Counters).
2) Synchronous counters.

- In asynchronous counter (also called as ripple counters) the clock pulse is applied to the first flip-flop, i.e., the least significant bit stage of the counter and the successive flip-flop is triggered by the output of the previous flip-flop and thus the counter has a cumulative settling time. Hence, its speed of operation is limited.
- In synchronous counters, the speed limitation of ripple counters is overcome by applying clock pulses simultaneously to all the flip-flops which leads to the settling time of the counter being equal to the propagation delay of a single flip-flop. Hence synchronous counters are also called parallel counters.




(a) 2-bit Asynchronous Counter (a) 2-bit Synchronous Counter

**Fig. 5.3.1** Comparison of Synchronous and Asynchronous Counter

- If a counter counts in such a way that the decimal equivalent of the output increases with successive clock pulses, it is known as up counter. If the decimal equivalent of output decreases with successive clock pulses, it is called as down counter. An up/down counter can count in any direction depending upon the control input.

## 5.4 RIPPLE COUNTERS

In ripple counters, the flip-flops are connected in such a manner that the output of first flip-flop drives the clock input for the second flip-flop, the output of the second flip-flop to the clock input of the third flip-flop and so on, which means that all the FFs are not clocked simultaneously. As the triggers move through the flip-flops like a ripple, hence the name ripple counter.

### 5.4.1 Design of a 4 Bit Binary Ripple Counter

- A counter that follows the binary number sequence is known as a binary counter. An n-bit binary counter said to consist of n flip flops and can count in binary, from 0 through $2^n - 1$.
- The logic diagram of a 4 bit binary ripple up counter is shown in Fig. 5.4.1. A 4-bit ripple counter counts in the sequence 0, 1, 2, 3, 4, .... 14, 15, 0, 1, 2, ...., i.e., 0000, 0001, ..... 1111, 0000, 0001, ....

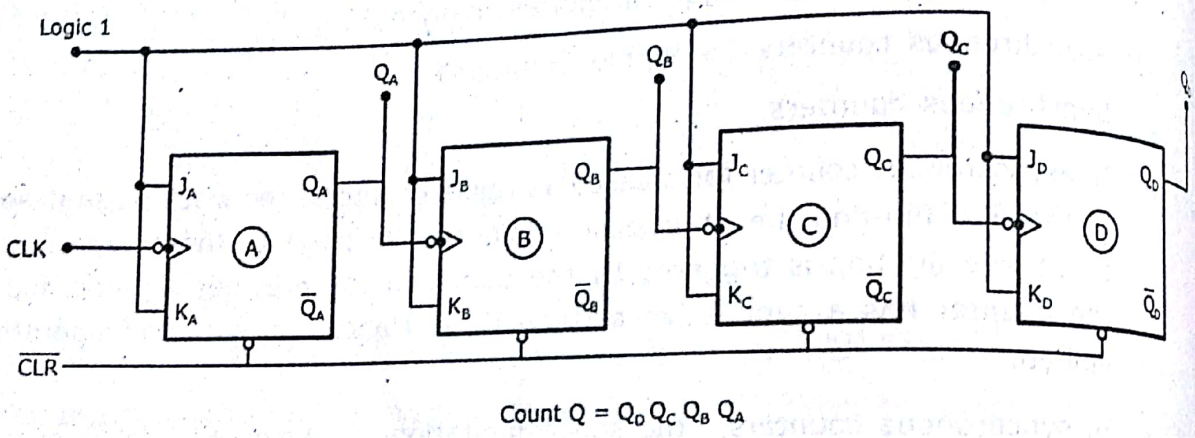


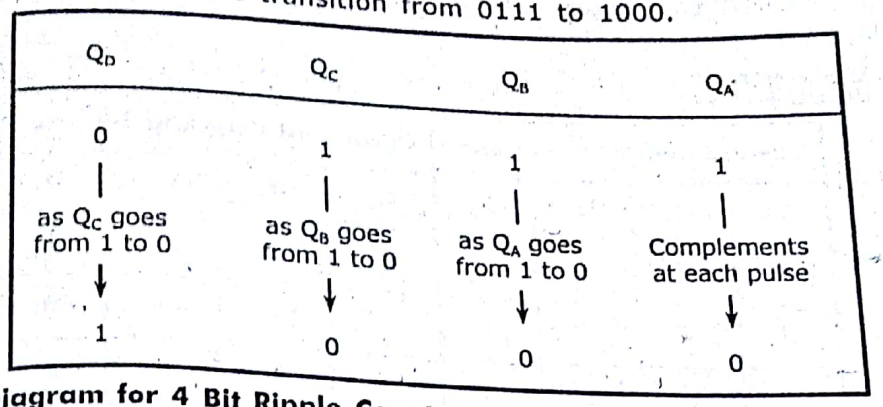

**Fig. 5.4.1** Logic Diagram of 4-bit Asynchronous up Counter

The four-bit asynchronous counter is obtained by the use of four JK flip-flops, named as A, B C and D, where D flip-flop holds the most significant bit of the count and flip-flop A holds the least significant bit of the count.

#### Working Operation

- Consider initially all the flip flops to be in the logical 0 state (i.e., $Q_A = Q_B = Q_C = Q_D = 0$). This is done by a logic '0' on $\overline{CLR}$ input.
- The count starts with binary 0 and increments by one for every count pulse input. Once the count reaches 15, the counter goes back to 0 and repeats the count again.
- The least significant bit ($Q_A$) in this circuit is complemented with each count pulse input. Whenever the $Q_A$ goes from 1 to 0, it complements $Q_B$. When $Q_B$ goes from 1 to 0, it complements $Q_C$, and so on for any higher order bits in the ripple counter.

For example let's see the transition from 0111 to 1000.




**Timing Diagram for 4 Bit Ripple Counter**

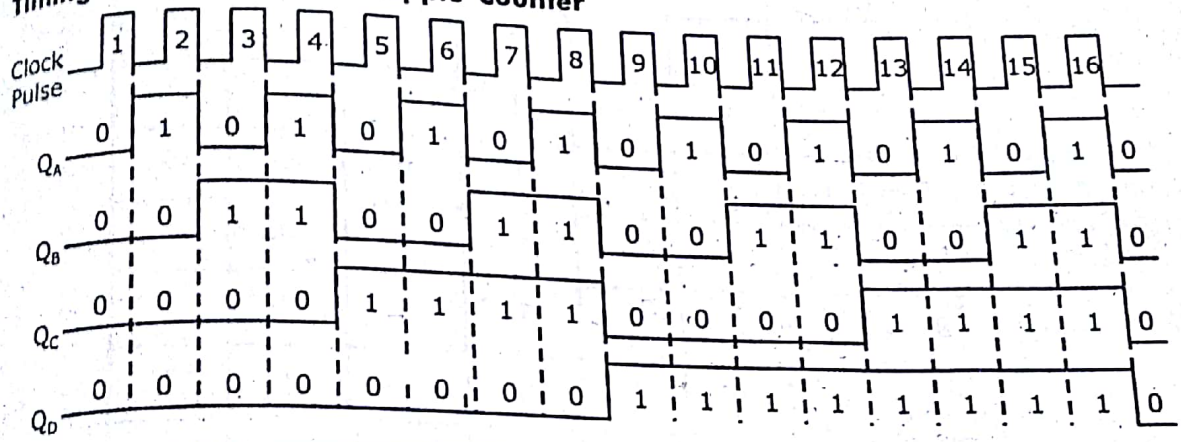



**Fig. 5.4.2** Timing Diagram for 4-bit Ripple Counter

**4 Bit Ripple Down Counter :** So far we have counted upward from zero, that is, it is an up counter. A ripple counter that counts the sequence from maximum count to zero is shown in Fig. 5.4.3.

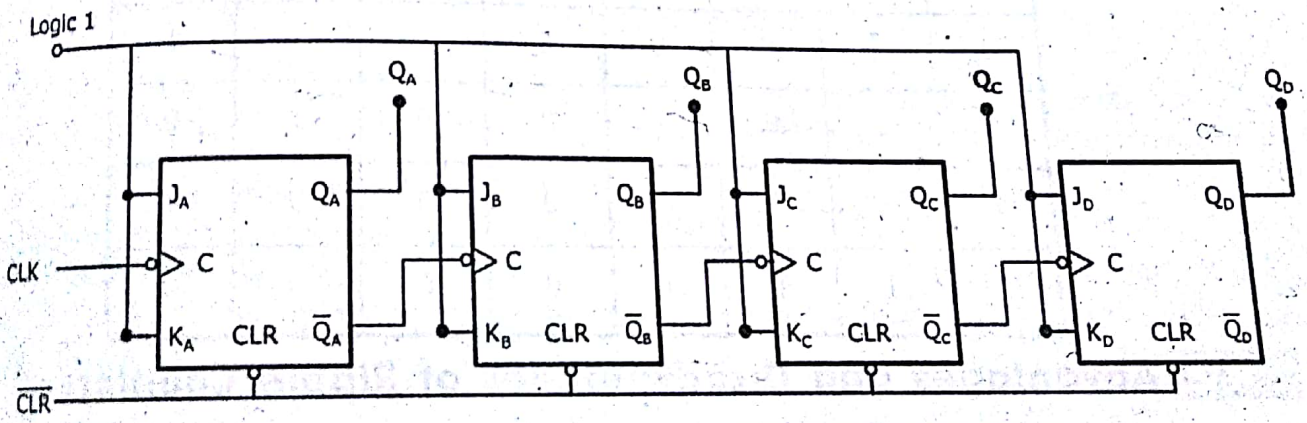



**Fig. 5.4.3** Logic Diagram of 4 bit Ripple Down Counter

Downward counting can be achieved,

1) By Connecting the complement output of each flip flop to the C input of the next flip flop.
2) By using positive edge triggered flip flops.

❖ A ripple counter that counts upward and downward counting sequence is shown in Table. 5.4.1.

**Table 5.4.1** Upward and Downward 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 |

## 5.4.2 Advantages and Disadvantages of Ripple Counters

**Advantages of Asynchronous Counters**

1) Very simple and straight forward operation and these counters are constructed by suitably interconnecting a number of JK or D flip-flops.
2) They require fewer components.
3) Very easy to design any MOD-counter.

**Disadvantages of Asynchronous Counters**

1) The speed of operation is low, because each flip-flop in the counter is not clocked simultaneously. So, these counters are used where the speed of operation is not of particular importance.
2) The propagation delay is the major drawback, because it limits the rate at which the counter can be clocked and creates decoding problems.
3) To obtain a truncated sequence, it is necessary to force these counters to recycle before going through all of it's normal states.

## 5.5 SYNCHRONOUS BINARY COUNTERS

In synchronous counters, the speed limitation of ripple counters is overcome by applying clock pulses simultaneously to all the flip flops. Hence synchronous counters are also called parallel counters.

### 5.5.1 Design of Binary Counters

The design procedure for a synchronous counter is the same as with any other synchronous sequential circuit. We will now go through a complete synchronous counter design procedure. Although we will do it for a specific counting sequence, the same steps can be followed for any desired sequence.

**Step 1 :** Determine the number of flip-flops needed, i.e., desired number of bits and the desired counting.

**Step 2 :** Choose the type of flip-flop to be used, either JK, D or T etc.

**Step 3 :** Draw the state diagram showing all possible states, including those that are not a part of the desired counting sequence.

**Step 4 :** Use state transitions to obtain the state table that lists all present states and next states.

**Step 5 :** From the state table, derive the circuit excitation table.

**Step 6 :** Use K-map or any other simplification method to derive the circuit as flip-flop input functions i.e., Design the logic circuits to generate the levels required at each flip-flop input.

**Step 7 :** Draw the logic diagram, i.e., implement the final expression.

**Example Problem 5.2**

**Design a 4 bit binary counter that counts in sequence from 0000 to 1111 and then back to 0000.**

Sol. :

- **Number of Flip-flops Required :** The number of flip-flops required to implement the given counter is obtained by equation,

$$2^{n-1} \leq N \leq 2^n$$

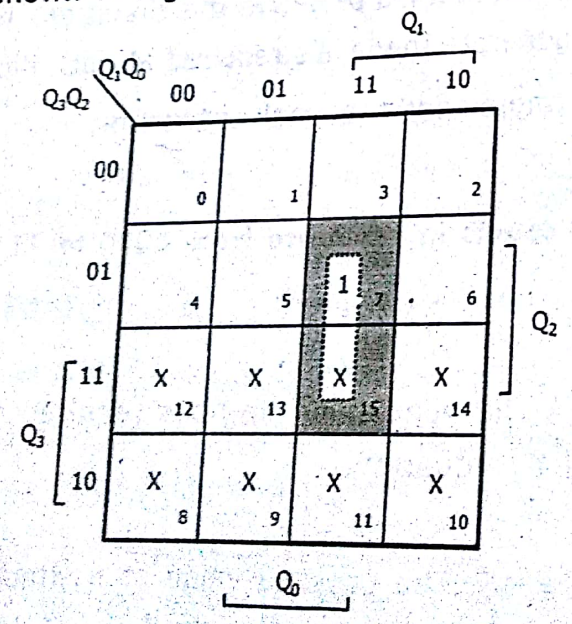
For the given problem N = 15. The possible value of n which satisfies the above equation is n = 4. Thus 4 flip-flops are required.

❖ **State Table and Flip Flop Inputs :** The state table having entries for flip flop inputs $J_{Q3}K_{Q3}$, $J_{Q2}K_{Q2}$, $J_{Q1}K_{Q1}$ and $J_{Q0}K_{Q0}$ is given in Table. 5.5.1 (here notice flip flop input is obtained using the excitation table of JK flip flop explained in unit 4).

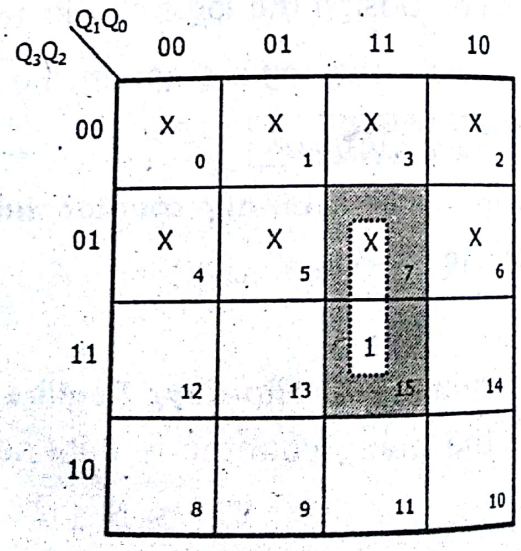
**Table 5.5.1** State Table and Flip Flop Inputs for Binary Counter

| Present State | | | | Next State | | | | Flip Flop Inputs | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $Q_3$ | $Q_2$ | $Q_1$ | $Q_0$ | $Q^+_3$ | $Q^+_2$ | $Q^+_1$ | $Q^+_0$ | $J_{Q3}$ | $K_{Q3}$ | $J_{Q2}$ | $K_{Q2}$ | $J_{Q1}$ | $K_{Q1}$ | $J_{Q0}$ | $K_{Q0}$ |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | × | 0 | × | 0 | × | 1 | × |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | × | 0 | × | 1 | × | × | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | × | 0 | × | × | 0 | 1 | × |
| 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | × | 1 | × | × | 1 | × | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | × | × | 0 | 0 | × | 1 | × |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | × | × | 0 | 1 | × | × | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | × | × | 0 | × | 0 | 1 | × |
| 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | × | × | 1 | × | 1 | × | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | × | 0 | 0 | × | 0 | × | 1 | × |
| 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | × | 0 | 0 | × | 1 | × | × | 1 |
| 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | × | 0 | 1 | × | × | 1 | × | 1 |
| 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | × | 0 | × | 0 | 0 | × | 1 | × |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | × | 0 | × | 0 | 1 | × | × | 1 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | × | 0 | × | 0 | × | 0 | 1 | × |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | × | 1 | × | 1 | × | 1 | × | 1 |

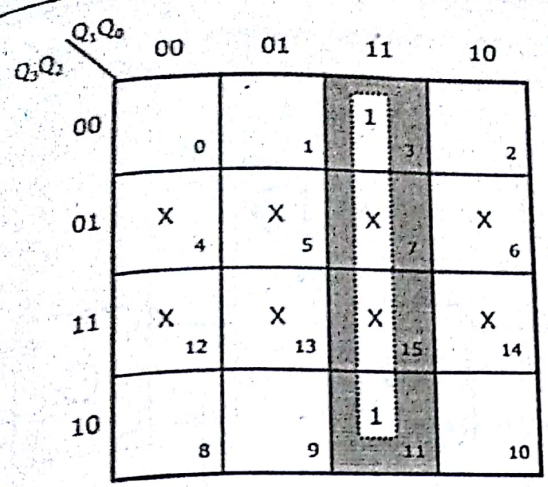
❖ ***K-Map Simplification :*** The flip flop input equations can be obtained using K-maps shown in Fig. 5.5.1.



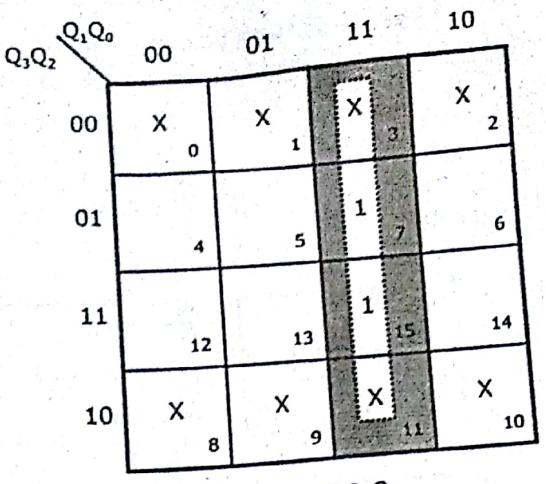

$J_{Q3} = Q_0Q_1Q_2$



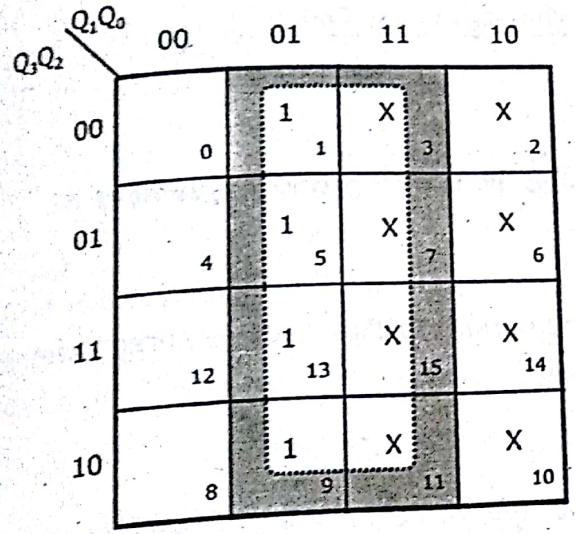

$K_{Q3} = Q_0Q_1Q_2$



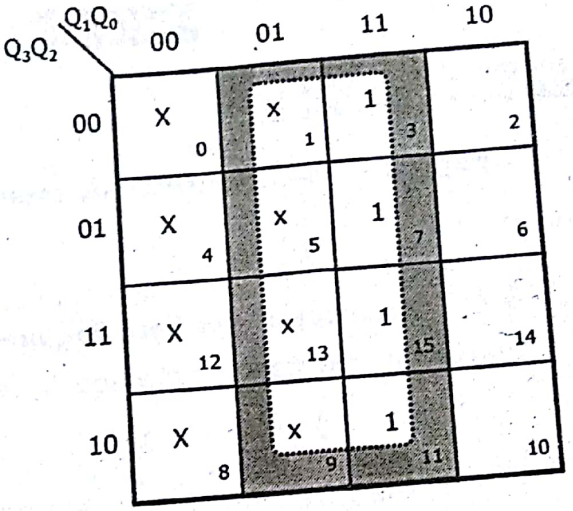

$J_{Q2} = Q_0Q_1$



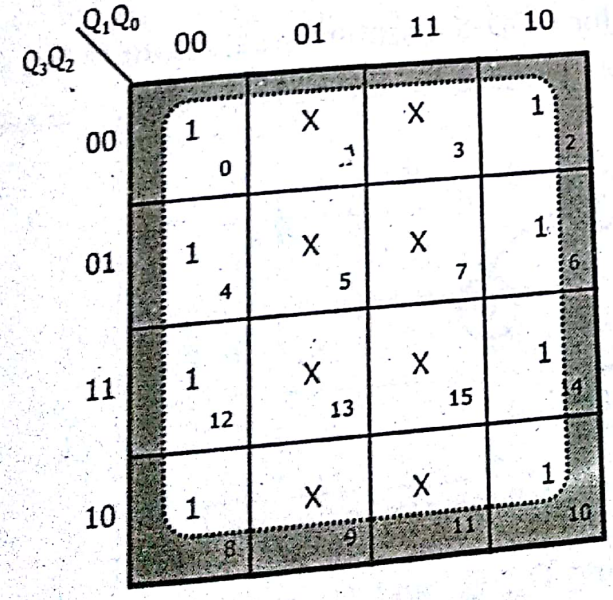

$K_{Q2} = Q_0Q_1$



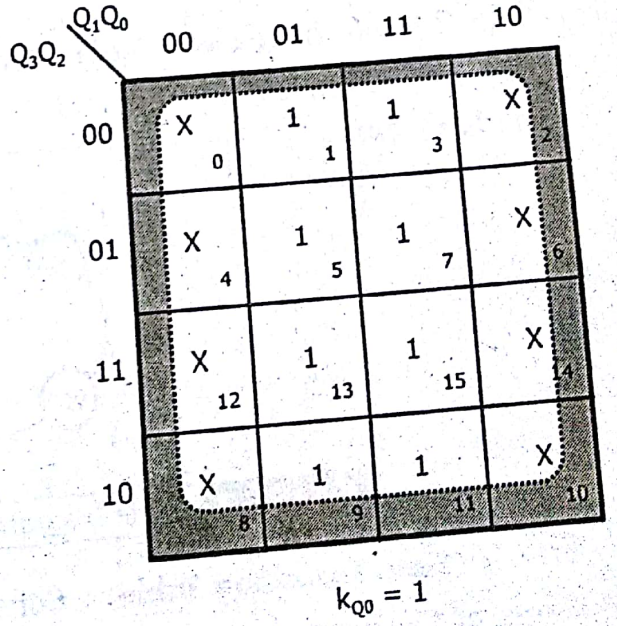

$J_{Q1} = Q_0$




$K_{Q1} = Q_0$




$J_{Q0} = 1$




$k_{Q0} = 1$

**Fig. 5.5.1** Maps for Input Equations of a Binary Counter

❖ **Logic Diagram :** Using the above K-map simplified equations, the logic diagram for the 4 bit synchronous binary counter is shown in Fig. 5.5.2.

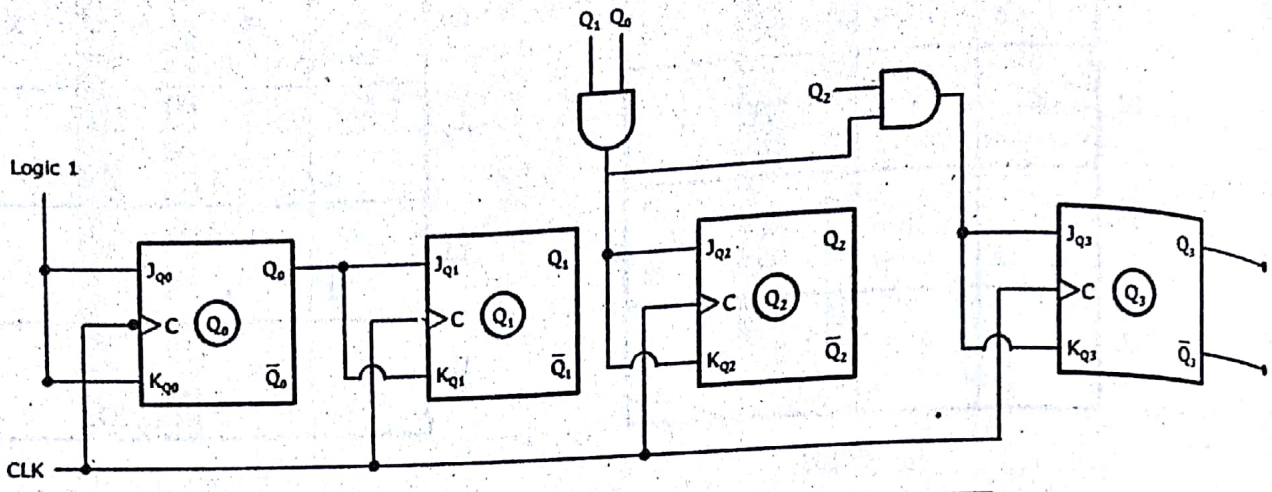



**Fig. 5.5.2** 4 Bit Synchronous Binary Counter

**Example Problem 5.3**

**Design a MOD-5 synchronous counter using JK flip-flop and implement it?**

*Sol. :*

**STEP 1 (Number of Flip-flops Required) :** The number of flip-flops required to implement the given counter is obtained by equation,

$$2^{n-1} \leq N \leq 2^n$$

For the given problem N = 5. The possible value of n which satisfies the above equation is n = 3. Thus 3 flip-flops are required.

**STEP 2 (State Diagram) :** State diagram for MOD-5 counter is as shown in Fig. 5.5.3. For the counts 101, 110 and 111 the next state is not defined and is considered as don't care.

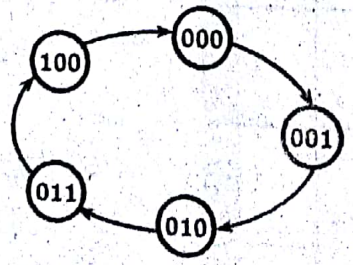



**Fig. 5.5.3** State Diagram of MOD-5 Synchronous Counter

**STEP 3 (State Transition Table) :** Considering $Q_A$, $Q_B$ and $Q_C$ representing the present states of flip-flops and $Q_{A+1}$, $Q_{B+1}$, $Q_{C+1}$ representing the next states of flip-flops. The state table for MOD-5 counter is as shown in Table. 5.5.2.

**Table 5.5.2** State Table for MOD-5 Counter

| Present State | | | Next State | | |
|---|---|---|---|---|---|
| $Q_C$ | $Q_B$ | $Q_A$ | $Q_{C+1}$ | $Q_{B+1}$ | $Q_{A+1}$ |
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | × | × | × |
| 1 | 1 | 0 | × | × | × |
| 1 | 1 | 1 | × | × | × |

**STEP 4 (Excitation Table)** : The excitation table having entries for flip-flop inputs ($J_CK_C$, $J_BK_B$ and $J_AK_A$) can be drawn from the above state table (and using the excitation table of JK flip-flops as shown in Table. 5.5.3.

**Table 5.5.3** Excitation Table for MOD-5 Counter

| Present State | | | Next State | | | Flip-flop Inputs | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $Q_C$ | $Q_B$ | $Q_A$ | $Q_{C+1}$ | $Q_{B+1}$ | $Q_{A+1}$ | $J_C$ | $K_C$ | $J_B$ | $K_B$ | $J_A$ | $K_A$ |
| 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 | 0 | 1 | 1 | 0 | X | X | 0 | 1 | X |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 | X | X | 1 | X | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | X | 1 | 0 | X | 0 | X |
| 1 | 0 | 1 | X | X | X | X | X | X | X | X | X |
| 1 | 1 | 0 | X | X | X | X | X | X | X | X | X |
| 1 | 1 | 1 | X | X | X | X | X | X | X | X | X |

**Step 5 (K-Map Simplification) :** The simplification procedure is as shown in Fig. 5.5.4.

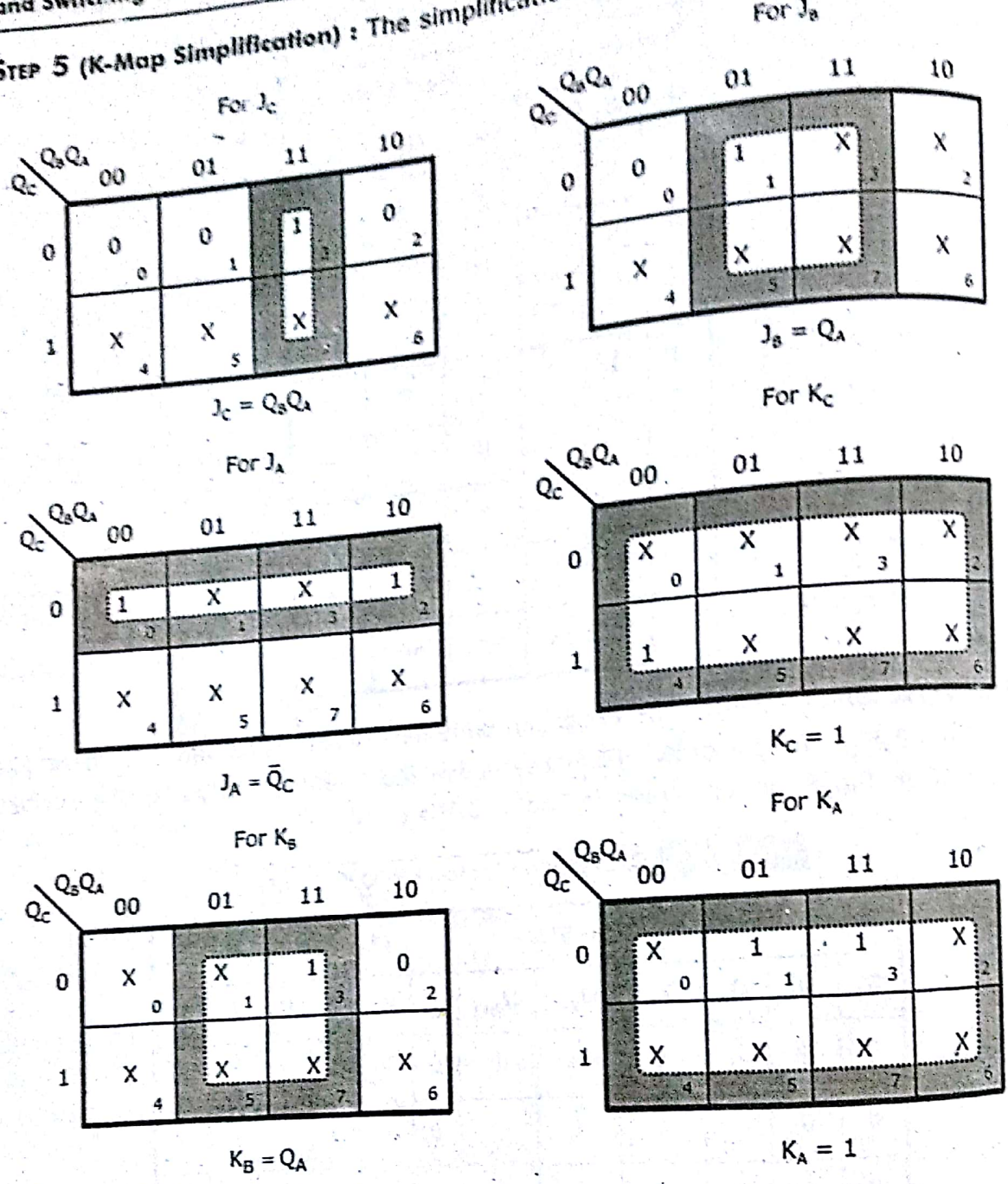



**Fig. 5.5.4** K-maps for Flip-flop Inputs

From the above K-maps, the simplified functions are,

$J_C = Q_BQ_A$, $K_C = 1$

$J_B = Q_A$, $K_B = Q_A$

$J_A = \bar{Q}_C$, $K_A = 1$

**Step 6 (Logic Diagram) :** Using the above K-map simplified equations, the logic diagram for the MOD-5 counter is as shown in Fig. 5.5.5.

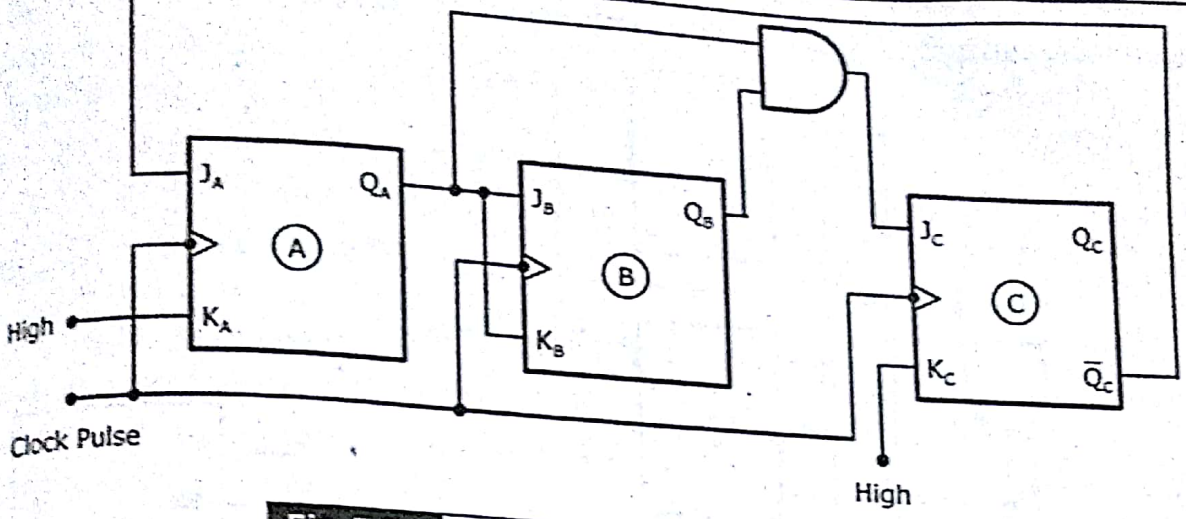



**Fig. 5.5.5** Logic Diagram of MOD-5 Counter

## 5.5.2 Design of Counter with Count Enable Input

In most of the applications, it is desirable to have control over the operation of a counter. This can be achieved with a 'counter enable' input, represented as EN (this differs entirely from E·N which is E and N).

The flip flop input equations for a binary counter using enable, EN, can be given as,

$$J_{Q0} = K_{Q0} = EN$$

$$J_{Q1} = K_{Q1} = Q_0 \cdot EN$$

$$J_{Q2} = K_{Q2} = Q_0 \cdot Q_1 \cdot EN$$

$$J_{Q3} = K_{Q3} = Q_0 \cdot Q_1 \cdot Q_2 \cdot EN$$

The functioning of the circuit when 'enable' is used, can be explained as below,

**Case 1 (When EN = 0)**

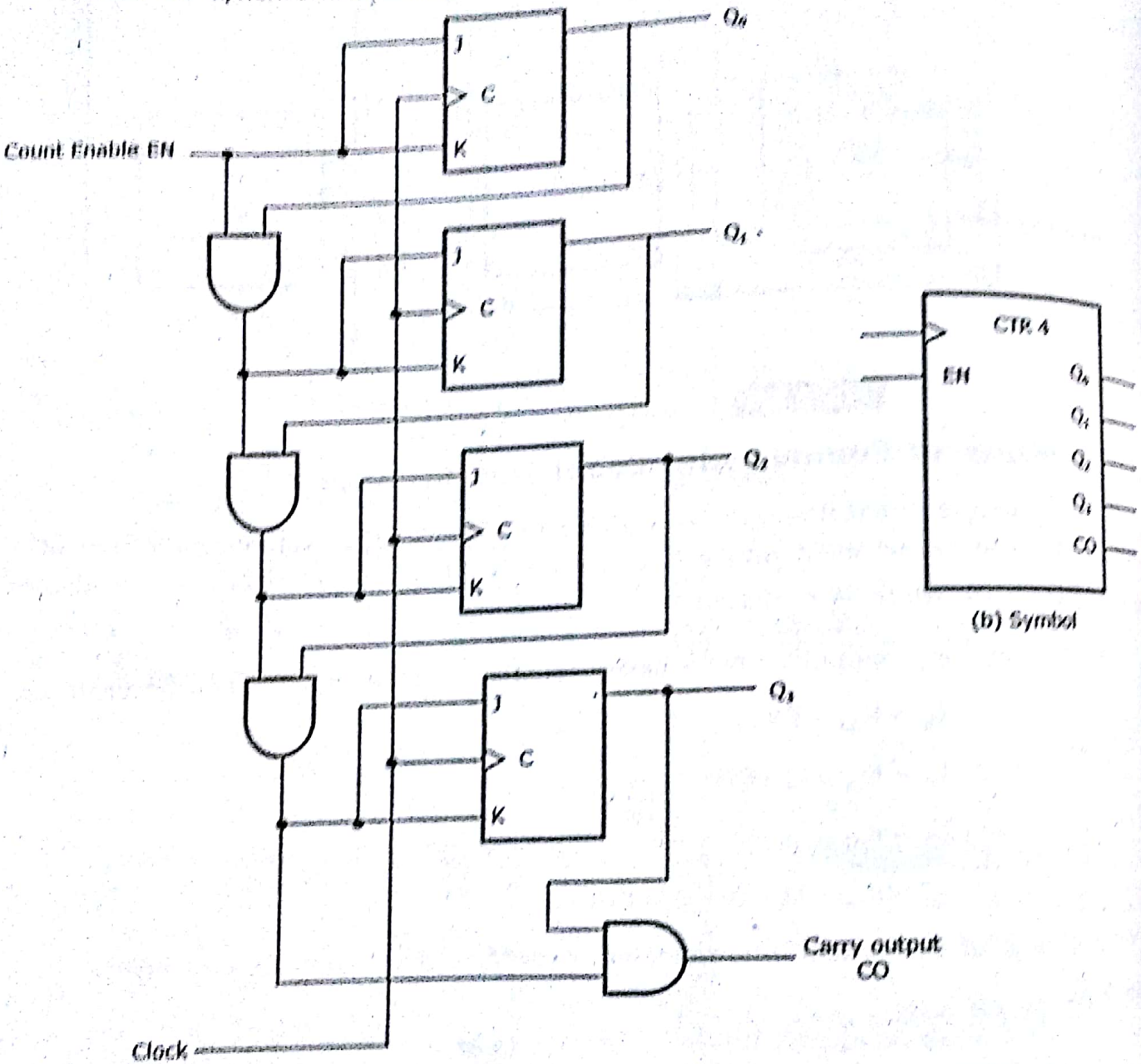
- J and K inputs – are 0.
- Flip flops – remains in same state

**Case 2 (When EN = 1)**

- J and K inputs reduce to equations derived in Fig. 5.5.6.
- flip flops. The LSB flip flop is complemented for every pulse transition. And the other position flip flops are complemented with a clock pulse transition when all previous least significant bits are equal to 1.

**Example :** Consider the operation of a 4 bit synchronous binary counter using control enable pin. We know, all synchronous binary counters have a regular pattern.

The 4 bit synchronous binary counter is given as follows,




(a) Logic Diagram

(b) Symbol

**Fig. 5.5.6** 4 Bit Synchronous Binary Counter With Control Enable

In the above Fig. 5.5.6, all the flip flops are connected to the common clock pulses. The LSB, $Q_0$ stage is complemented when counter is enabled, i.e., EN = 1. When all the previous stages are 1 and count is 1, the other J and K inputs are 1. The final carry output, CO, can be used for extending the counter to more stages. And each stage should have an additional AND gate and flip flop.

The synchronous counter can be triggered with both positive or negative clock transition, unlike the ripple counter, where polarity is essential.

## Serial and Parallel Counters

These are two ways in which binary counters can be designed.

1) Serial counter (or) Serial gating
   - When gates are used such that the present stage depends on the immediate prior it is said to be a serial counter.
   - This is analogous to the carry logic in a ripple carry adder.
   - Example : The Fig. 5.5.7(a) represents a chain of 2 input AND gates is used to provide information to each stage about the state of prior stages of the counter.

2) Parallel counter (or) Parallel gating
   - Here, all the prior stages inputs are given directly given to the gates connecting the counters, instead of having the immediate prior stage as an input.
   - Delay is reduced using parallel counter.
   - Thus, the counter operational speed is faster.
   - Example : The Fig. 5.5.7(b) represents the same logic of Fig. 5.5.7(a) but in a parallel fashion.
   - The advantage in parallel gating logic is that, in going from state 1111 to state 0000, only one AND gate delay occurs instead of the four AND gate delays that occur in the serial counter.

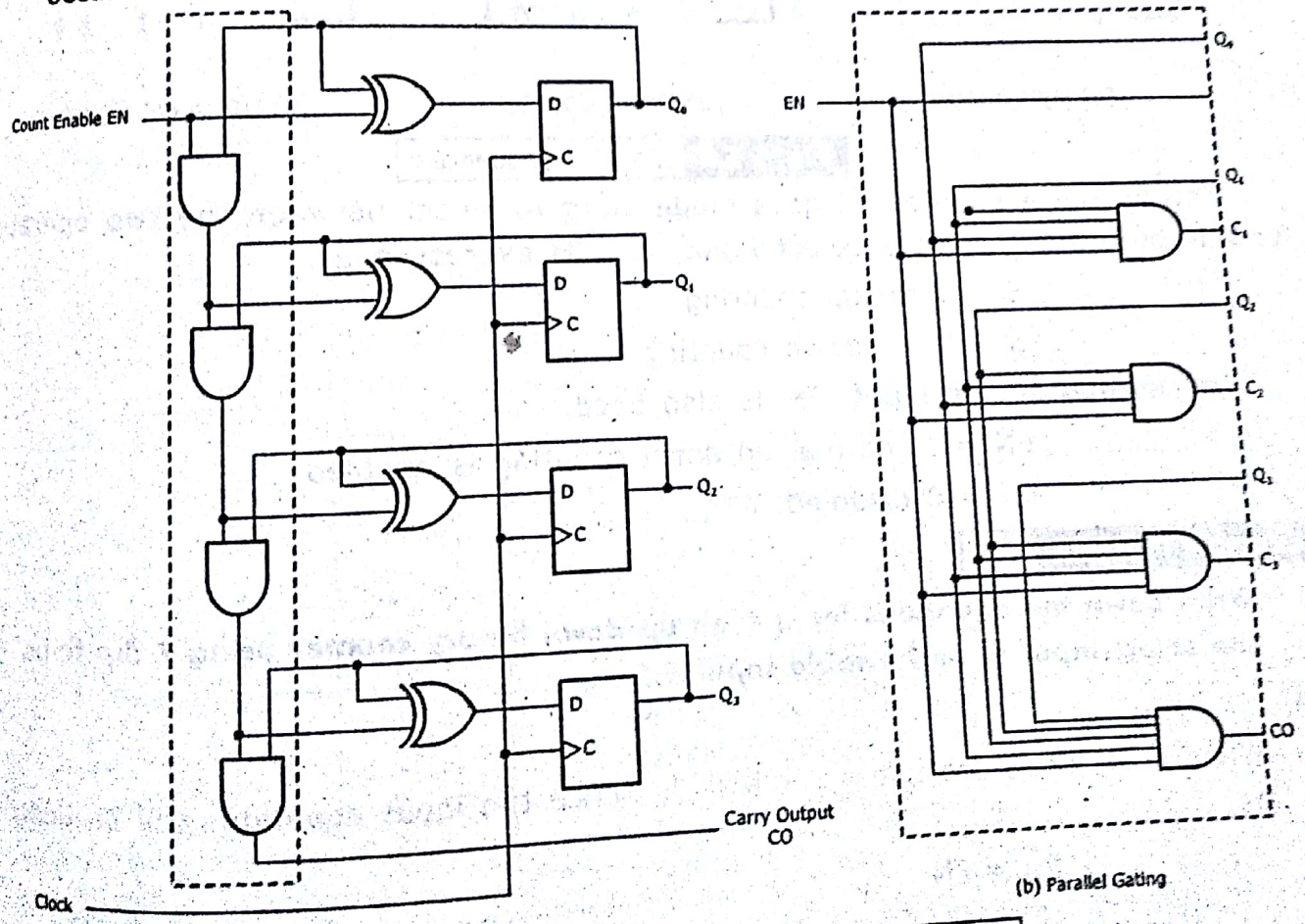


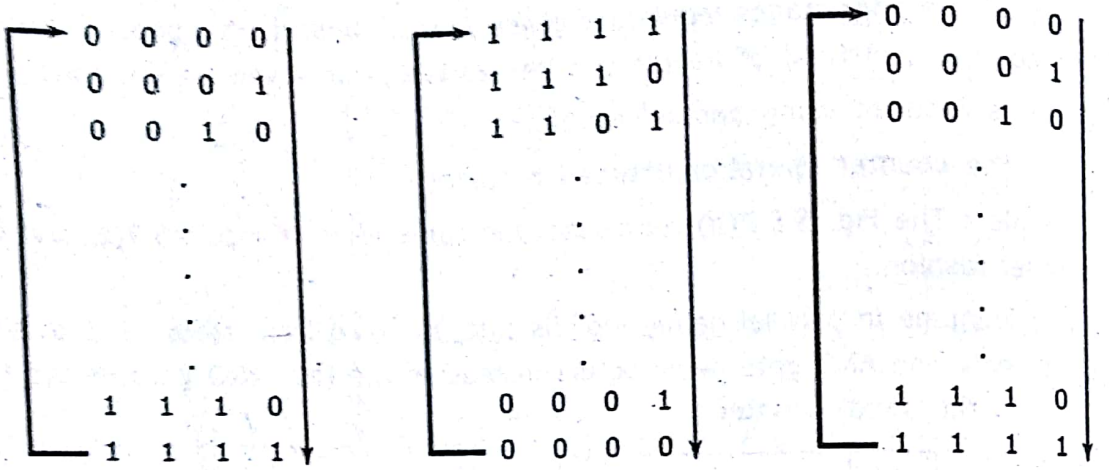

**Fig. 5.5.7** 4 Bit Binary Counter with D Flip Flops

## 5.5.4 Up Down Binary Counter

**Synchronous Count Down Binary Counter :** A synchronous count down binary counter counts the binary states in the reverse order from 1111 to 0000 and back to 1111 to re count the same. This represented in Fig. 5.5.8(a).

**Synchronous Count Up Binary Counter :** The functioning of the synchronous count up binary counter is opposite to the count down binary counter. It counts from 0000 to 1111 and back to 0000 to re count the sequence, as represented in Fig. 5.5.8(b).

**Synchronous Up Down Counter :** The operation of the two above explained counters is combined, a counter that can count both up and down is formed. This is referred to as 'up down counter'. The count proceeds as shown in Fig. 5.5.8(c).




(a) Up Counter (b) Down Counter (c) Up Down Counter

**Fig. 5.5.8** Counter Sequences

The up down counter needs a mode input to select between the two operations for this purpose, we use a select input, S. It is expressed as,

S = 1 for up counting

S = 0 for down counting

The normal enable input, EN, is also used,

EN = 1, normal up/down counting is enabled

EN = 0 disabled.

**Example Problem 5.4**

**Write down the equations for a 4 bit up/down binary counter using T flip flops, with the select input S, and enable input EN.**

*Sol. :*

When T flip flops are used, J – K = T. Hence the input equations will be described as,

$$T_{A0} = EN$$

$$T_{A1} = Q_0 \cdot S \cdot EN + \overline{Q}_0 \cdot \overline{S} \cdot EN$$

$$T_{A2} = Q_0 \cdot Q_1 S \cdot EN + \overline{Q}_0 \cdot \overline{Q}_1 \cdot \overline{S} \cdot EN$$

$$T_{A3} = Q_0 \cdot Q_1 Q_2 \cdot S \cdot EN + \overline{Q}_0 \cdot \overline{Q}_1 \cdot \overline{Q}_2 \cdot \overline{S} \cdot EN$$

The carry output equations will be given as,

$$C_{up} = Q_0 \cdot Q_1 \cdot Q_2 \cdot Q_3 \cdot S \cdot EN$$

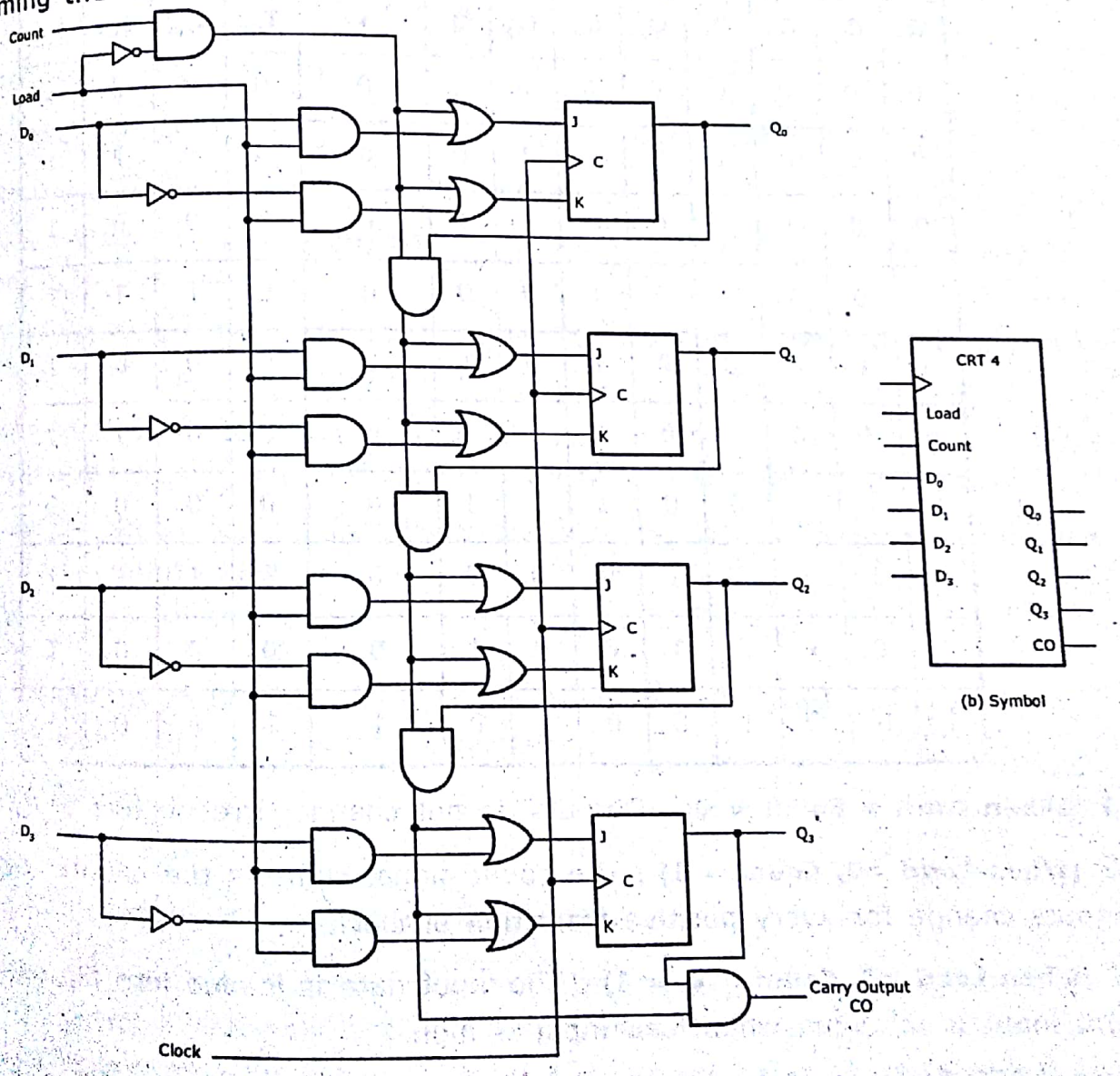
$$C_{dn} = \overline{Q}_0 \cdot \overline{Q}_1 \cdot \overline{Q}_2 \cdot \overline{Q}_3 \cdot S \cdot EN$$

Thus, a generalized expression for a synchronous binary up down counter can be given as,

$$T_{Ai} = Q_0 \cdot Q_1 \cdot Q_2 \ldots Q_{i-1} \cdot S \cdot EN + \overline{Q}_0 \cdot \overline{Q}_1 \cdot \overline{Q}_2 \ldots \cdot S \cdot EN$$

## 5.5.5 Binary Counter with Parallel Load

In most of the digital systems that employ counters, they require a parallel load capability. This enables transferring an initial binary number into the counter before performing the counter operation.




(a) Logic Diagram

(b) Symbol

**Fig. 5.5.9** 4 Bit Binary Counter with Parallel Load

From the Fig. 5.5.9, we can see that there are 3 control inputs count, load, clock. The device will operate in two modes.

1) **Load Operation (Load = 1)**
   - The input load control disables the count operation.
   - Causes a transfer of data from the four parallel inputs into the four flip flops.
2) **Counter Operation (Load = 0, Count = 1)**
   - The circuit operates as a binary counter.
   - When all flip flops are equal to 1, the carry output C0 becomes 1.

The output of the circuit in Fig. 5.5.9 is described in the Table 5.5.4.

**Table 5.5.4** State Table and Flip Flop Inputs 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 | $T_{Q8}$ | $T_{Q4}$ | $T_{Q2}$ | $T_{Q1}$ |
| 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 |

**CASE 1 (When Load = Count = 0) :** Outputs do not change, irrespective of the clock.

**CASE 2 (When Load =0, Count = 1) :** The count signal controls the counter operation. The outputs change for every positive transition of clock.

**CASE 3 (When Load =0, Count = 0 or 1) :** The input data is loaded into flip flops. Here, the count input is inhibited when load input is high.

**Synchronous BCD Counter :** A binary counter when connected with an external AND gate can be converted into a BCD counter (without load input), as shown in Fig. 5.5.10 below.

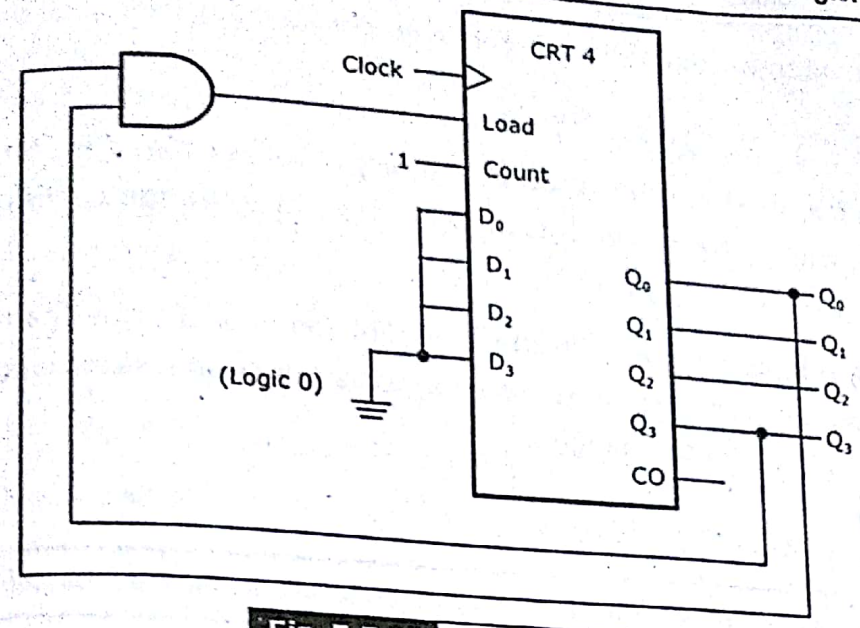



**Fig. 5.5.10** BCD Counter

Functioning of BCD counter

1) The count input is active at all times in the BCD counter.
2) The count starts with all zero outputs.
3) For every positive clock transition, the count is incremented by 1, assuming that AND gate output is always 0.
4) When the output becomes 1001, $Q_0$ and $Q_3$ become 1, making the AND output 1. Hence, the 'load' input becomes active.
5) The next clock transition now loads its inputs, instead of counting. As $D_0$, $D_1$, $D_2$, $D_3$ are grounded, 0000 is loaded and the counter operation starts again.

Thus, the circuit acts as a BCD counter, counting from 0000 through 1001 and back to 0000 to re count again.

## 5.6 S Y M M

Symmetric functions are a common occurrence while syntherizing practical circuits. They are widely used for practical applications.

**Example :** Functional description of decoders, sum and carry, difference and borrow functions in binary addition and binary subtraction, etc.

### 5.6.1 Properties of Symmetric Functions

**Definition 1 :** A switching function of n variables $f(x_1, x_2, ....., x_n)$ is called symmetric (or totally symmetric) if and only if it is invariant under any permutation of its variables; it is called partially symmetric in the variables $x_i$, $x_j$, where $\{x_i, x_j\}$ is a subset of $\{x_1, x_2, ....,$ $x_n\}$, if and only if the interchange of the variables $x_i$, $x_j$ leaves the function unchanged.

Let us consider an example for better understanding.

**Example :** Consider the function f(a, b, c)

**CASE 1 (When f(a, b, c) = a'b'c + ab'c' + a'bc') :** We say that f is "totally symmetric" because even when the variables a, b, c are interchanged, the same result is obtained.

**CASE 2 (When f(a, b, c) = a'b'c + ab'c') :** This function is said to be "partially symmetric". Because, the function remains same when a, c variables are interchanged but it is not valid if a, b or b, c are interchanged.

**NOTE**

*The variable is which a function is symmetric are known as "variables of symmetry".*

**Theorem 1 : The necessary and sufficient condition for a function $f(x_1, x_2, ..., x_n)$ to be symmetric is that it may be specified by a set of numbers $\{a_1, a_2, ....., a_n\}$, where $0 \le a_i \le n$, such that it assumes the value 1 when and only when $a_i$ of the variables are equal to 1. The numbers in the set $\{a_1, a_2, ..., a_k\}$ are called the a-numbers.**

*Proof :*

Let assume a function $f(x_1, x_2, .....x_n)$ is symmetric.

- This function is assumed to be equal to 1 when some $a_i$ variables are equal to 1.
- But, we said that the function is symmetric, hence the function is equal to 1 for any permutations of $a_i$ variables.
- Conversely, when 'f' can be specified by a set of numbers, the permutations will only relabel the variables but not change the values of these a-numbers.
- Hence, can conclude that the function is invariant under any permutation of its variables and is symmetric.

In general,

- Symmetric function is denoted is $S_{a_1, a_2, ..., a_k}(x_1, x_2, ...., x_n)$.
- $a_1, a_2, ...., a_k$ are the a-numbers.
- $x_1, x_2, ..... x_n$ are variables of symmetry.

**Example :** The function $f(x, y, z) = x'y'z + x'yz' + xy'z'$ assumes the value 1 when and only when any one out of its three variables is equal to 1. This function is thus denoted $S_1(x,y,z)$. Similarly, the symmetric function $S_{1,3}(x, y, z)$ is $f(x, y, z) = xyz + x'y'z + xy'z' + x'yz'$.

**Property :** Let M and N designate two sets of a-numbers, then the sum and product of two symmetric functions $S_M(x_1, x_2,..., x_n)$ and $SN(x_1, x_2,..., x_n)$, which have the same variables of symmetry, are symmetric and equal to

$$S_M(x_1, x_2,..., x_n) + S_N(x_1, x_2,..., x_n) = S_{M\cup N}(x_1, x_2,..., x_n)$$

$$S_M(x_1, x_2,..., x_n) \cdot S_N(x_1, x_2,..., x_n) = S_{M\cap N}(x_1, x_2,..., x_n)$$

Where $M \cup N$ and $M \cap N$ correspond, respectively, to the union and intersection of the sets of the a-number.

Let $f_1(a, b, c, d) = S_{0,2,4}(a, b, c, d)$ and

$f_2(a, b, c, d) = S_{3,4}(a, b, c, d)$

Then, $f_3(a, b, c, d) = f_1 + f_2 = S_{0,2,3,4}(a, b, c, d)$

And $f_4(a, b, c, d) = f_1 \cdot f_2 = S_4(a, b, c, d)$

**Property :** The complement $S'_M(x_1, x_2,..., x_n)$ of a symmetric function $S_M(x_1, x_2,..., x_n)$ is also a symmetric function whose a-numbers are those numbers included in the set {0, 1,..., n} and not included in M.

**Example :** $S'_{0,1,4}(a, b, c, d) = S_{2,3}(a, b, c, d)$.

**Theorem 2 (Shanno's Expansion Theorem)**

The application of Shannon's expansion theorem to a symmetric function renders.

$$S_M(x_1, x_2,..., x_n) = x'_1 S_M(0, x_2,..., x_n) + x_1 S_M(1, x_2,..., x_n).$$

**Theorem 3 (Expansion Theorem for Symmetric Functions)**

Expansion theorem for symmetric function is as follows,

$$S_{a_1, a_2,..., a_k}(x_1, x_2,..., x_n)$$

$$= x'_1 S_{a_1, a_2,..., a_k}(x_1, x_2,..., x_n) + x_1 S_{a_1-1, a_2-1,..., a_k-1}(x_2,..., x_n)$$

Where the a-number n (if it exists) is eliminated from the first term, because it is a function of only n–1 variable. Similarly, the a-number 0 (if it exists) is eliminated from the second term.

## 5.6.2 Symmetric Relay Contact Networks

A basic network for symmetric functions is a multiple-output network with single input which is grounded and n+1 outputs, numbered 0 through n. Such a network is called symmetric network. This is shown in Fig. 5.6.1 below.

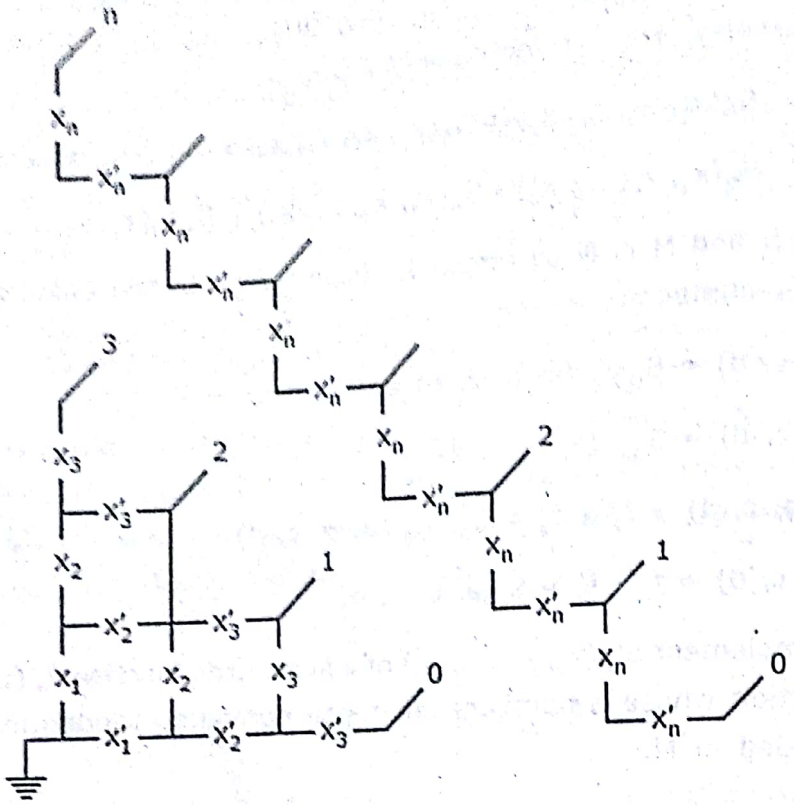



**Fig. 5.6.1** Basic Contact Symmetric Netowork

**Properties of Symmetric Network**

- The contacts are arranged such that ground can propagate from input terminal in two paths.
  - From bottom to top.
  - From left to right.
- Contacts of operated relays shift ground upward to successive levels.
- Contacts of unoperated relays shift ground to right.

**Example :** Draw the symmetric network to realize the function $S_{1,3}(x_1, x_2, x_3)$.

**Step 1 :** Let us construct the basic symmetric network for the 3 given variables $x_1$, $x_2$, $x_3$. It will have a grounded input and four outputs numbered 0 to 'n' corresponding to its a-numbers.

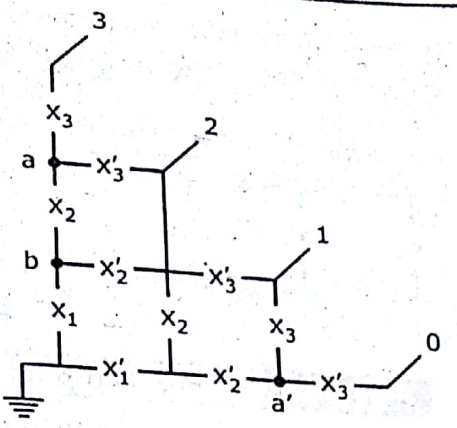



**Fig. 5.6.2** Basic Symmetric Network

**Step 2 :** we are required to realize the function $S_{1,3}(x_1, x_2, x_3)$. This is obtained by joining the output terminals labelled 1 and 3. This is shown in dotted lines in Fig. 5.6.3.

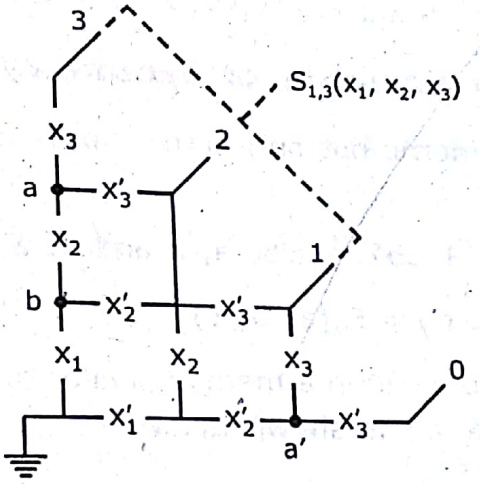



**Fig. 5.6.3** Basic Symmetric Network for Three Variables Illustrating Connections for $S_{1,3}(x_1, x_2, x_3)$

**Step 3 :** Now, we have to delete the lines leading to output terminal 0 and 2. The result is obtained as,

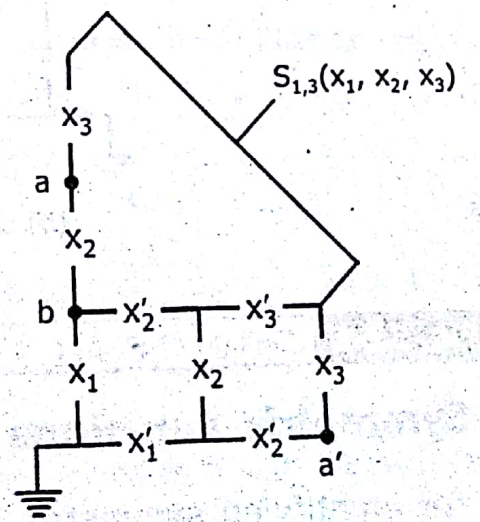



**Fig. 5.6.4** Reduced Network

**STEP 4 :** The transmission from 2 vertices a, a' depend on terminal $x_3$ only. Hence, it is advantages to eliminate an extra link. This is given by,

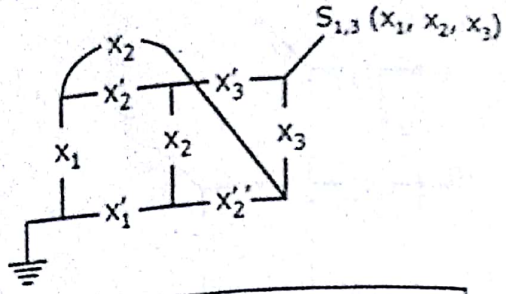



**Fig. 5.6.5** Minimal Network

This method of folding the circuit gives a minimal network. From the tie sets of the network, we can say that transmission function is equal to 1 when,

$$T(x_1, x_2, x_3) = S_{1,3}(x_1, x_2, x_3)$$

$$= x_1x_2x_3 + x_1x'_2, x'_3 + x'_1x_2x'_3 + x'_1x'_2x_3$$

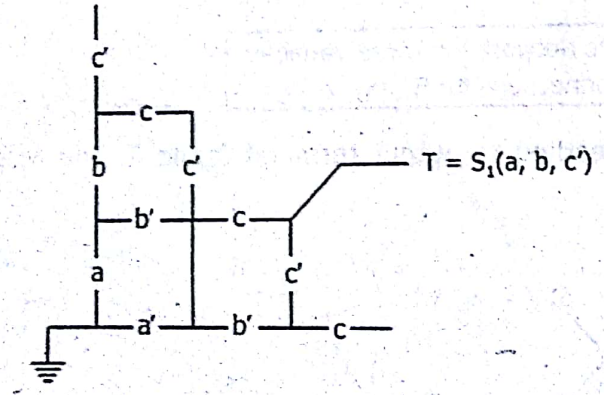
## 5.6.2.1 Complemented Variable of Symmetry

A function may be symmetric not only with respect to primed number's but also to unprimed numbers.

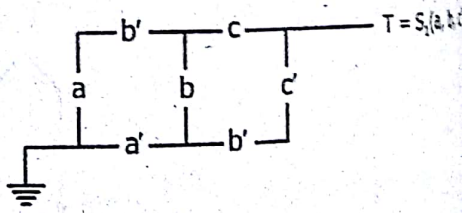
**Example :** T(a, b, c) = a'b'c' + ab'c + a'bc a, b and c', a',b' and c

i.e., T(a, b, c) = $S_1$(a, b, c') = $S_2$(a', b', c)

These networks can be obtained in a method similar to the one explained in example the network realizing $S_1$(a, b, c') is shown below.




(a) Basic Symmetric Network for the Variables a, b, c'




(b) Simplified Network

**Fig. 5.6.6** Design of $S_1$(a, b, c')

## 5.6.3 Identification of Symmetric Functions

An algorithmic procedure for identifying symmetric functions and the variables symmetry is illustrated below by means of an example.

**Example Problem 5.5**

**Find whether the function, $F(p, q, r, s) = \Sigma(0, 1, 3, 5, 8, 10, 11, 12, 13, 15)$ is symmetric and if so express the function in symmetric notation.**

Sol. :

The true minterms of the function are listed in binary code in Table. 4.15.1(a) and the column sums are noted below. A little reflection reveals that for any totally symmetric function, all column sums must be equal since the permutation of variables of symmetry does not change the function. In Table. 5.6.1(a), there are two different column sums 6 and 4. Suppose now we complement the columns q and r, that is, we consider p, q', r' and s as the variables of the function, then the list of minterms will be as in Table 4.15.1(b). This process does not alter the function since q assuming the value 0 is the same as q' assuming the value 1 and vice versa. The process is sometimes referred to as double complementation as both the variable and the entries in the corresponding column are complemented. In Table. 5.6.1(b), we find that all column sums are identical. If there were to be more than two different column sums or if the sum of the two column sums were to be not equal to the number of rows, complementing any number of columns would not result in equal column sums. Thus, we obtain a necessary condition. For a function to be symmetric, either all column sums must be equal or else there should be only two different column sums.

Now, focus your attention on the row's sums indicated in the $r_1$ column of Table. 5.6.1 column (b). If a row sum r occurs, it should occur $n_{C_r}$ times in order that the function be symmetric. This follows from the fact that any r variable may assume the value 1 to make the function equal to 1. This leads to another necessary condition.

To summarize, the two necessary conditions are namely, equal column sums and occurrence of each row sum $n_{C_r}$ times, together constitute sufficiency to declare the function as a symmetric function. The row sums become the alpha numbers.

In Table 5.6.1(b), observe that the row sum 2 occurs $4_{C_2} = \frac{4 \times 3}{2 \times 1} = 6$ times and the row sum 3 occurs $4_{C_3} = 4$ times.

Hence, the function is symmetric and can be written as,

$$F(p, q, r, s) = S^4_{2,3}(p, q', r', s)$$

**Table 5.6.1** Test for Symmetry of the Function $F = \Sigma (0, 1, 3, 5, 8, 10, 11, 12, 13, 15)$

| p | r | s | t | | p | r' | s' | t | $r_1$ | | p' | r | s | t' | $r_1$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | | 0 | 1 | 1 | 0 | 2 | | 1 | 0 | 0 | 1 | 2 |
| 0 | 0 | 0 | 1 | | 0 | 1 | 1 | 1 | 3 | | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | | 0 | 1 | 0 | 1 | 2 | | 1 | 0 | 1 | 0 | 2 |
| 0 | 1 | 0 | 1 | | 0 | 0 | 1 | 1 | 2 | | 1 | 1 | 0 | 0 | 2 |
| 1 | 0 | 0 | 0 | | 1 | 1 | 1 | 0 | 3 | | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | | 1 | 1 | 0 | 0 | 2 | | 0 | 0 | 1 | 1 | 2 |
| 1 | 0 | 1 | 1 | | 1 | 1 | 0 | 1 | 3 | | 0 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | | 1 | 0 | 1 | 0 | 2 | | 0 | 1 | 0 | 1 | 2 |
| 1 | 1 | 0 | 1 | | 1 | 0 | 1 | 1 | 3 | | 0 | 1 | 0 | 0 | 2 |
| 1 | 1 | 1 | 1 | | 1 | 0 | 0 | 1 | 2 | | 0 | 1 | 1 | 0 | 2 |
| 6 | 4 | 4 | 6 | | 6 | 6 | 6 | 6 | | | 4 | 4 | 4 | 4 | |
| | (a) | | | | | (b) | | | | | | (c) | | | |

In Table. 5.6.1(a), instead of complementing r, s columns, if we choose to complement p, t columns, we obtain Table. 5.6.1(c), which gives another symmetric form of the same function.

$$F = (p, q, r, s) = S^4_{1,2}\,(p', q, r, s')$$

A little more difficult situation is encountered in testing the following function.

**Example Problem 5.6**

**Test for symmetry**

**T(P, Q, R, S) = S(0, 3, 5, 10, 12, 15)**

*Sol. :*

The true minterms are listed in Table. 5.6.2(a). We find that all the column sums are identical satisfying the first necessary condition. But the second condition is not satisfied. The row sum 2 does not occur $4_{C_2} = 6$ times as required for the function to be symmetric. The column sums do not give us any hint about which columns could be complemented to satisfy the sufficiency of row sum occurrence.

In this situation, the column sum should be equal to one half the number of rows, otherwise any complementation of column (s) upsets the equality of column sums. This condition being satisfied in Table. 5.6.2(a), we now proceed to expand the function about any one of its variables say P. This can be achieved in the tabular form as in Table. 5.6.2(b). The partial functions of Q, R, S are easily recognized as symmetric in Q', R', S'. This would mean.

$$F(P, Q, R, S) = P' \cdot S_2(Q', R', S) + P\, S_1(Q', R', S)$$

which may be written as $S_2(P, Q', R', S)$ using the expansion theorem. Alternatively we, once again, write out the tabular form of the entire function with Q and R columns complemented, as in Table. 5.6.2(c). This table satisfies the sufficiency of occurrence of row sums. Hence the function will be declared as symmetric and written as,

**Table 5.6.2** Test for Symmetry Involving and Expansion of the Function $F = \Sigma (0, 3, 5, 10, 12, 15)$

| P | R | S | T | $r_1$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 2 |
| 0 | 1 | 0 | 1 | 2 |
| 1 | 0 | 1 | 0 | 2 |
| 1 | 1 | 0 | 0 | 2 |
| 1 | 1 | 1 | 1 | 4 |
| 3 | 3 | 3 | 3 | |

(a)

| P | Q | R | S |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| | 1 | 1 | 2 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |
| | 2 | 2 | 1 |

(b)

| P | Q' | R' | S | $r_1$ |
|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 2 |
| 0 | 1 | 0 | 1 | 2 |
| 0 | 0 | 1 | 1 | 2 |
| 1 | 1 | 0 | 0 | 2 |
| 1 | 0 | 1 | 0 | 2 |
| 1 | 0 | 0 | 1 | 2 |
| 3 | 3 | 3 | 3 | |

(c)

$$F(P, Q, R, S) = S_2^4 (P. Q', R', S)$$

$$= S_2^4 (P'. QR, S')$$

The entire procedure is summarized below,

### Step 1 : Obtain Column Sums

1) If more than two different sums occur, the function is not symmetric.
2) If two different sums occur, compare the total of these two sums with the number of rows in the table. If they are not equal, the function is not symmetric. If they are equal, complement the columns corresponding to either one of the column sums and continue to step 2.
3) If all column sums are identical, compare the sum with half the number of rows in the table. If they are not equal, go to step 2. If they are equal, go to step 3.

**STEP 2 :** Obtain row sums and check for sufficient occurrence that is, if is a row sum and n is the number of variables, then that row sum must occur $\binom{n}{r}$ times.

1) If any row sum does not occur the required number of times, the function is not symmetric.
2) If all row sums occur the required number of times, the function is symmetric in the variables given at the top of the current table and the alpha numbers are given by the deferent row sums.

**STEP 3 : Obtain Row Sums and Check Each for Sufficient Occurrence.**

1) If all rows sums occur the required number of times, the function is symmetric.
2) If any row sum does not occur the required number of times, expand the function about any one of its variables, that is, find functions g and h such that f = x'g + xh.

Determine all variable complementations required for the identification of symmetries in g and h. Test f under the same variable complementations. If all row sums occur the required number of times, f is symmetric, otherwise, it is not.

❖ ❖ ❖