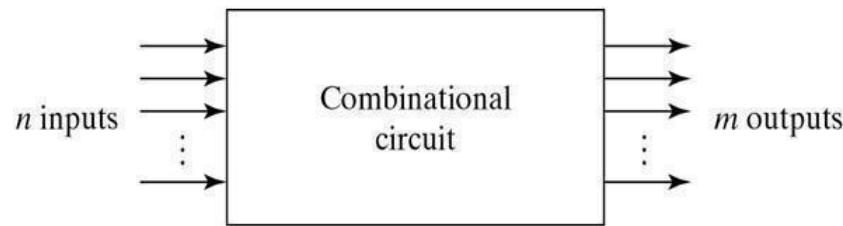


UNIT-III

COMBINATIONAL CIRCUITS

Combinational Logic

- Logic circuits for digital systems may be combinational or sequential.
- A combinational circuit consists of input variables, logic gates, and output variables.



For n input variables, there are 2^n possible combinations of binary input variables. For each possible input combination, there is one and only one possible output combination. A combinational circuit can be described by m Boolean functions, one for each output variable. Usually the inputs come from flip-flops and outputs go to flip-flops.

Design Procedure:

1. The problem is stated
2. The number of available input variables and required output variables is determined.
3. The input and output variables are assigned letters/symbols.
4. The truth table that defines the required relationship between inputs and outputs is derived.
5. The simplified Boolean function for each output is obtained.

Adders:

Digital computers perform variety of information processing tasks, the one is arithmetic operations. And the most basic arithmetic operation is the addition of two binary digits. i.e, 4 basic possible operations are:

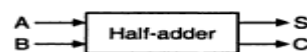
$$0+0=0, 0+1=1, 1+0=1, 1+1=10$$

The first three operations produce a sum whose length is one digit, but when augends and addend bits are equal to 1, the binary sum consists of two digits. The higher significant bit of this result is called a carry. A combinational circuit that performs the addition of two bits is called a half-adder. One that performs the addition of 3 bits (two significant bits & previous carry) is called a full adder. & 2 half adder can employ as a full-adder.

The Half Adder: A Half Adder is a combinational circuit with two binary inputs (augends and addend bits) and two binary outputs (sum and carry bits.) It adds the two inputs (A and B) and produces the sum (S) and the carry (C) bits. It is an arithmetic operation of addition of two single bit words.

Inputs		Outputs	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

(a) Truth table



(b) Block diagram

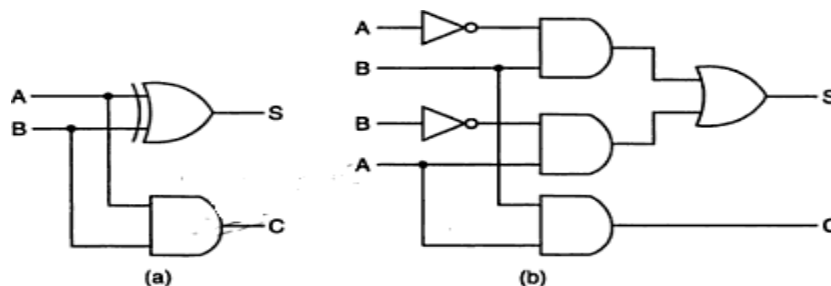
The Sum(S) bit and the carry (C) bit, according to the rules of binary addition, the sum (S) is the X-OR of A and B (It represents the LSB of the sum). Therefore,

$$S = A \oplus B$$

The carry (C) is the AND of A and B (it is 0 unless both the inputs are 1). Therefore,

$$C = AB$$

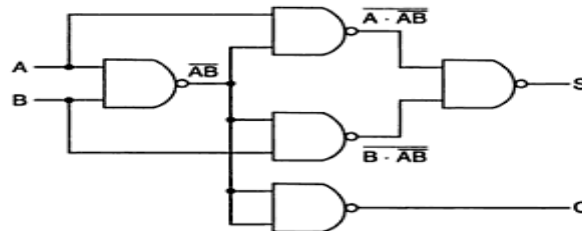
A half-adder can be realized by using one X-OR gate and one AND gate a



Logic diagrams of half-adder

NAND LOGIC:

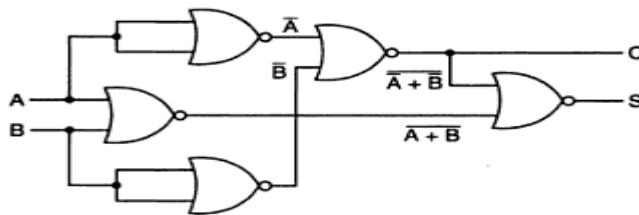
$$\begin{aligned}
 S &= A\bar{B} + \bar{A}B = A\bar{B} + A\bar{A} + \bar{A}B + B\bar{B} \\
 &= A(\bar{A} + \bar{B}) + B(\bar{A} + \bar{B}) \\
 &= A \cdot \bar{A}\bar{B} + B \cdot \bar{A}\bar{B} \\
 &= \overline{A \cdot \bar{A}\bar{B} \cdot B \cdot \bar{A}\bar{B}} \\
 C &= AB = \overline{\bar{A}\bar{B}}
 \end{aligned}$$



Logic diagram of a half-adder using only 2-input NAND gates.

NOR Logic:

$$\begin{aligned}
 S &= A\bar{B} + \bar{A}B = A\bar{B} + A\bar{A} + \bar{A}B + B\bar{B} \\
 &= A(\bar{A} + \bar{B}) + B(\bar{A} + \bar{B}) \\
 &= (A + B)(\bar{A} + \bar{B}) \\
 &= \overline{A + B + \bar{A} + \bar{B}} \\
 C &= AB = \overline{\bar{A}\bar{B}} = \overline{A + B}
 \end{aligned}$$



Logic diagram of a half-adder using only 2-input NOR gates.

The Full Adder:

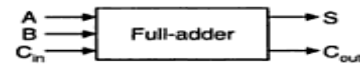
A Full-adder is a combinational circuit that adds two bits and a carry and outputs a sum bit and a carry bit. To add two binary numbers, each having two or more bits, the LSBs can be added by using a half-adder. The carry resulted from the addition of the LSBs is carried over to the next significant column and added to the two bits in that column. So, in the second and higher columns, the two data bits of that column and the carry bit generated from the addition in the previous column need to be added.

The full-adder adds the bits A and B and the carry from the previous column called the carry-in C_{in} and outputs the sum bit S and the carry bit called the carry-out C_{out} . The variable S gives the value of the least significant bit of the sum. The variable C_{out} gives the output carry. The

eight rows under the input variables designate all possible combinations of 1s and 0s that these variables may have. The 1s and 0s for the output variables are determined from the arithmetic sum of the input bits. When all the bits are 0s, the output is 0. The S output is equal to 1 when only 1 input is equal to 1 or when all the inputs are equal to 1. The C_{out} has a carry of 1 if two or three inputs are equal to 1.

Inputs			Sum	Carry
A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

(a) Truth table



(b) Block diagram

Full-adder.

From the truth table, a circuit that will produce the correct sum and carry bits in response to every possible combination of A,B and C_{in} is described by

$$S = \overline{A}\overline{B}C_{in} + \overline{A}B\overline{C_{in}} + A\overline{B}\overline{C_{in}} + ABC_{in}$$

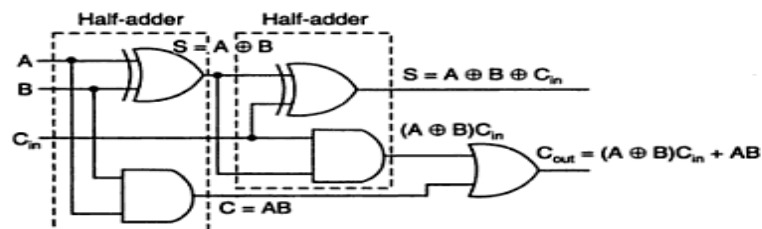
$$C_{out} = \overline{A}BC_{in} + A\overline{B}C_{in} + AB\overline{C_{in}} + ABC_{in}$$

and

$$S = A \oplus B \oplus C_{in}$$

$$C_{out} = AC_{in} + BC_{in} + AB$$

The sum term of the full-adder is the X-OR of A,B, and C_{in} , i.e, the sum bit the modulo sum of the data bits in that column and the carry from the previous column. The logic diagram of the full-adder using two X-OR gates and two AND gates (i.e, Two half adders) and one OR gate is



Logic diagram of a full-adder using two half-adders.

The block diagram of a full-adder using two half-adders is :



Block diagram of a full-adder using two half-adders.

Even though a full-adder can be constructed using two half-adders, the disadvantage is that the bits must propagate through several gates in accession, which makes the total propagation delay greater than that of the full-adder circuit using AOI logic.

The Full-adder neither can also be realized using universal logic, i.e., either only NAND gates or only NOR gates as

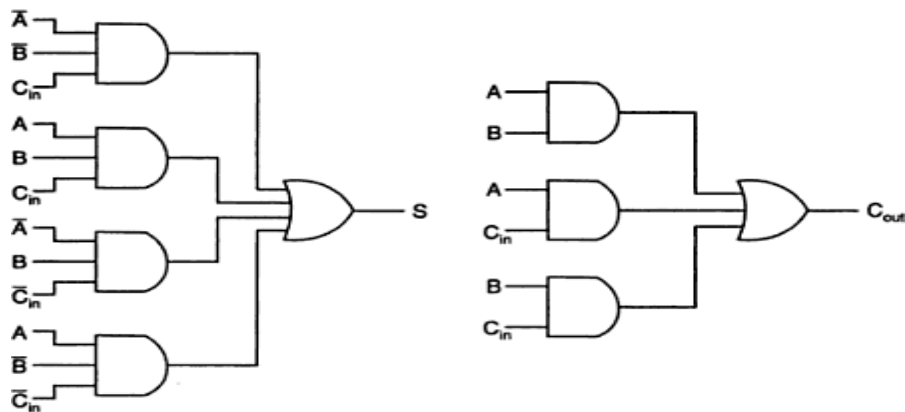
$$A \oplus B = \overline{\overline{A \cdot AB} \cdot \overline{B \cdot AB}}$$

Then

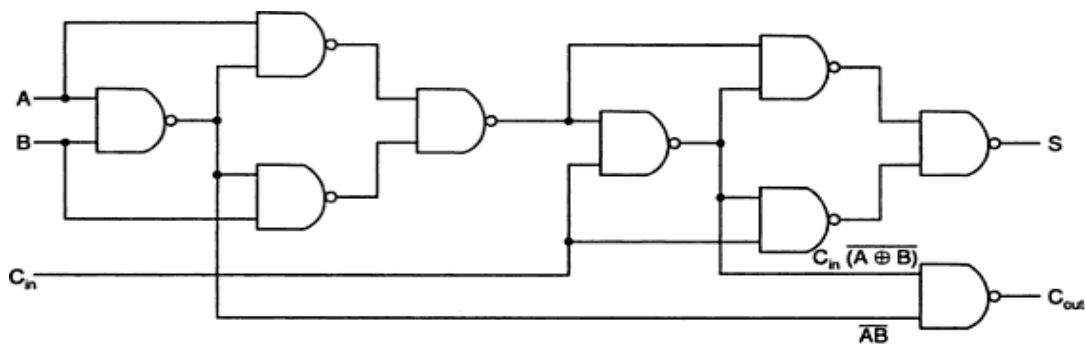
$$S = A \oplus B \oplus C_{in} = \overline{\overline{(A \oplus B) \cdot (A \oplus B)C_{in}} \cdot \overline{C_{in} \cdot (A \oplus B)C_{in}}}$$

NAND Logic:

$$C_{out} = C_{in}(A \oplus B) + AB = \overline{\overline{C_{in}(A \oplus B)} \cdot \overline{AB}}$$

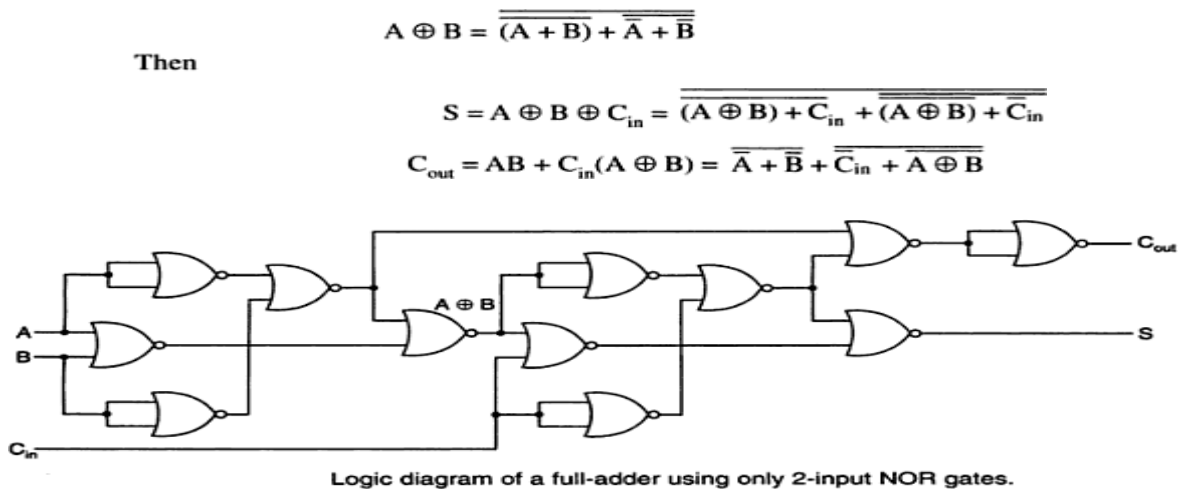


Sum and carry bits of a full-adder using AOI logic.



Logic diagram of a full-adder using only 2-input NAND gates.

NOR Logic:



Subtractors:

The subtraction of two binary numbers may be accomplished by taking the complement of the subtrahend and adding it to the minuend. By this, the subtraction operation becomes an addition operation and instead of having a separate circuit for subtraction, the adder itself can be used to perform subtraction. This results in reduction of hardware. In subtraction, each subtrahend bit of the number is subtracted from its corresponding significant minuend bit to form a difference bit. If the minuend bit is smaller than the subtrahend bit, a 1 is borrowed from the next significant position., that has been borrowed must be conveyed to the next higher pair of bits by means of a signal coming out (output) of a given stage and going into (input) the next higher stage.

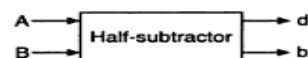
The Half-Subtractor:

A Half-subtractor is a combinational circuit that subtracts one bit from the other and produces the difference. It also has an output to specify if a 1 has been borrowed. . It is used to subtract the LSB of the subtrahend from the LSB of the minuend when one binary number is subtracted from the other.

A Half-subtractor is a combinational circuit with two inputs A and B and two outputs d and b. d indicates the difference and b is the output signal generated that informs the next stage that a 1 has been borrowed. When a bit B is subtracted from another bit A, a difference bit (d) and a borrow bit (b) result according to the rules given as

Inputs		Outputs	
A	B	d	b
0	0	0	0
1	0	1	0
1	1	0	0
0	1	1	1

(a) Truth table



(b) Block diagram

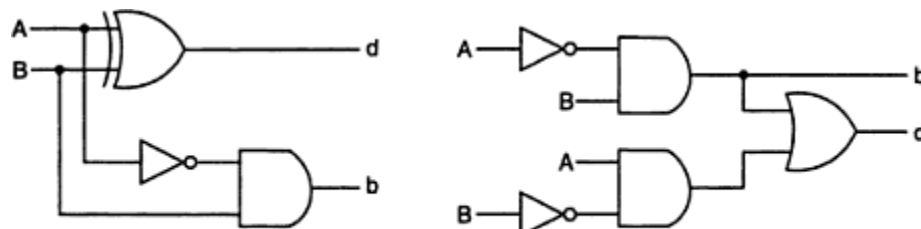
Half-subtractor.

The output borrow b is a 0 as long as $A \geq B$. It is a 1 for $A=0$ and $B=1$. The d output is the result of the arithmetic operation $2b + A - B$.

A circuit that produces the correct difference and borrow bits in response to every possible combination of the two 1-bit numbers is, therefore,

$$d = A \oplus B \text{ and } b = A \bar{B}$$

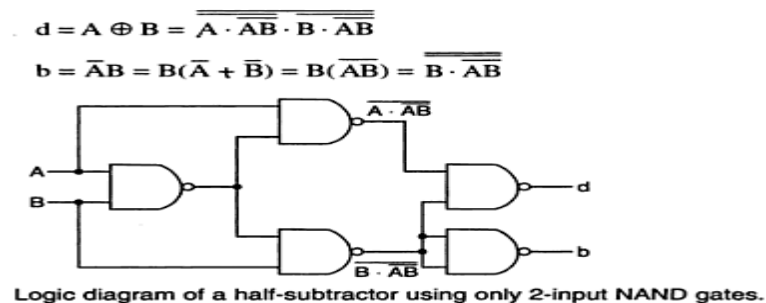
That is, the difference bit is obtained by X-OR ing the two inputs, and the borrow bit is obtained by ANDing the complement of the minuend with the subtrahend. Note that logic for this exactly the same as the logic for output S in the half-adder.



Logic diagrams of a half-subtractor.

A half-subtractor can also be realized using universal logic either using only NAND gates or using NOR gates as:

NAND Logic:



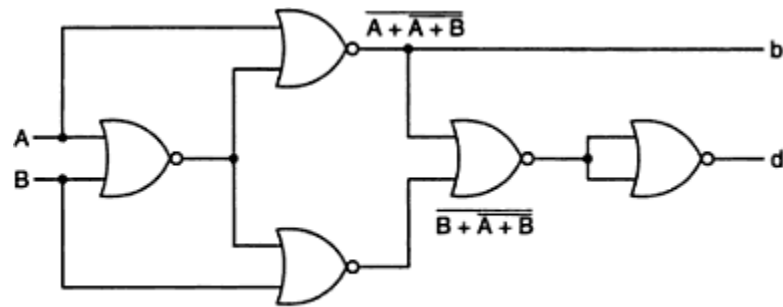
Logic diagram of a half-subtractor using only 2-input NAND gates.

NOR Logic:

$$d = A \oplus B = A\bar{B} + \bar{A}B = A\bar{B} + B\bar{B} + \bar{A}B + A\bar{A}$$

$$= \bar{B}(A + B) + \bar{A}(A + B) = \overline{B + A + B} + \overline{A + A + B}$$

$$d = \bar{A}B = \bar{A}(A + B) = \overline{\bar{A}(A + B)} = A + (\bar{A} + B)$$



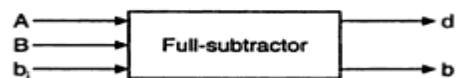
Logic diagram of a half-subtractor using only 2-input NOR gates.

The Full-Sub tractor:

The half-subtractor can be only for LSB subtraction. IF there is a borrow during the subtraction of the LSBs, it affects the subtraction in the next higher column; the subtrahend bit is subtracted from the minuend bit, considering the borrow from that column used for the subtraction in the preceding column. Such a subtraction is performed by a full-subtractor. It subtracts one bit (B) from another bit (A), when already there is a borrow b_i from this column for the subtraction in the preceding column, and outputs the difference bit (d) and the borrow bit(b) required from the next d and b. The two outputs present the difference and output borrow. The 1s and 0s for the output variables are determined from the subtraction of $A-B-b_i$.

Inputs			Difference	Borrow
A	B	b_i	d	b
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

(a) Truth table



(b) Block diagram

Full-subtractor.

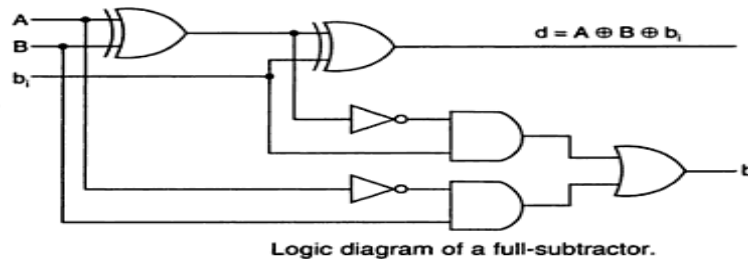
From the truth table, a circuit that will produce the correct difference and borrow bits in response to every possible combinations of A,B and b_i is

$$\begin{aligned}
 d &= \bar{A}\bar{B}b_i + \bar{A}B\bar{b}_i + A\bar{B}\bar{b}_i + ABb_i \\
 &= b_i(AB + \bar{A}\bar{B}) + \bar{b}_i(A\bar{B} + \bar{A}B) \\
 &= b_i(\overline{A \oplus B}) + \bar{b}_i(A \oplus B) = A \oplus B \oplus b_i
 \end{aligned}$$

and

$$\begin{aligned}
 b &= \bar{A}\bar{B}b_i + \bar{A}B\bar{b}_i + \bar{A}Bb_i + ABb_i = \bar{A}B(b_i + \bar{b}_i) + (AB + \bar{A}\bar{B})b_i \\
 &= \bar{A}B + (A \oplus B)b_i
 \end{aligned}$$

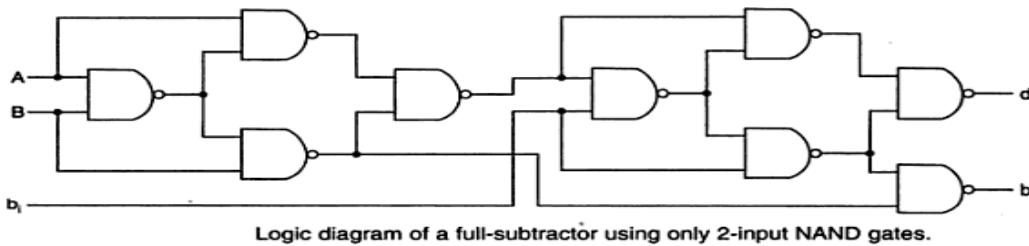
A full-subtractor can be realized using X-OR gates and AOI gates as



The full subtractor can also be realized using universal logic either using only NAND gates or using NOR gates as:

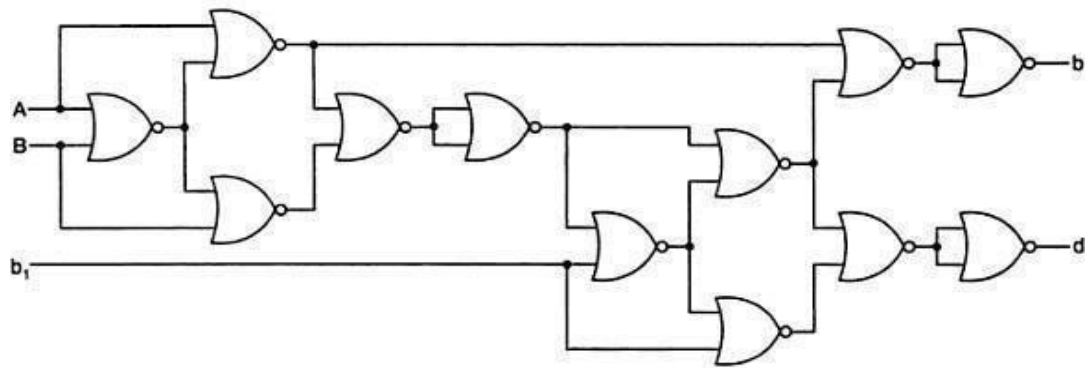
NAND Logic:

$$\begin{aligned}
 d &= A \oplus B \oplus b_i = \overline{(A \oplus B) \oplus b_i} = \overline{(A \oplus B)(A \oplus B)b_i \cdot b_i(A \oplus B)b_i} \\
 b &= \overline{AB} + b_i(\overline{A \oplus B}) = \overline{AB} + b_i(\overline{A \oplus B}) \\
 &= \overline{AB} \cdot b_i(\overline{A \oplus B}) = \overline{B(\overline{A} + \overline{B}) \cdot b_i(b_i + (A \oplus B))} \\
 &= \overline{B \cdot AB \cdot b_i[b_i \cdot (A \oplus B)]}
 \end{aligned}$$



NOR Logic:

$$\begin{aligned}
 d &= A \oplus B \oplus b_i = \overline{(A \oplus B) \oplus b_i} \\
 &= \overline{(A \oplus B)b_i + (A \oplus B)\overline{b_i}} \\
 &= \overline{[(A \oplus B) + (A \oplus B)\overline{b_i}][b_i + (A \oplus B)\overline{b_i}]} \\
 &= \overline{(A \oplus B) + (A \oplus B)b_i + b_i + (A \oplus B)b_i} \\
 &= \overline{(A \oplus B) + (A \oplus B)b_i + b_i + (A \oplus B)b_i} \\
 b &= \overline{AB} + b_i(\overline{A \oplus B}) \\
 &= \overline{A(A + B) + (A \oplus B)[(A \oplus B) + b_i]} \\
 &= \overline{A + (A + B) + (A \oplus B) + (A \oplus B)b_i}
 \end{aligned}$$

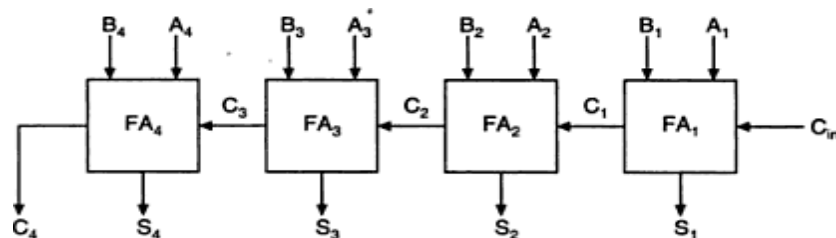


Logic diagram of a full subtractor using only 2-input NOR gates.

Binary Parallel Adder:

A binary parallel adder is a digital circuit that adds two binary numbers in parallel form and produces the arithmetic sum of those numbers in parallel form. It consists of full adders connected in a chain, with the output carry from each full-adder connected to the input carry of the next full-adder in the chain.

The interconnection of four full-adder (FA) circuits to provide a 4-bit parallel adder. The augends bits of A and addend bits of B are designated by subscript numbers from right to left, with subscript 1 denoting the lower-order bit. The carries are connected in a chain through the full-adders. The input carry to the adder is C_{in} and the output carry is C_4 . The S output generates the required sum bits. When the 4-bit full-adder circuit is enclosed within an IC package, it has four terminals for the augends bits, four terminals for the addend bits, four terminals for the sum bits, and two terminals for the input and output carries. An n-bit parallel adder requires n full adders. It can be constructed from 4-bit, 2-bit and 1-bit full adder ICs by cascading several packages. The output carry from one package must be connected to the input carry of the one with the next higher-order bits. The 4-bit full adder is a typical example of an MSI function.



Logic diagram of a 4-bit binary parallel adder.

Ripple carry adder:

In the parallel adder, the carry-out of each stage is connected to the carry-in of the next stage. The sum and carry-out bits of any stage cannot be produced, until sometime after the carry-in of that stage occurs. This is due to the propagation delays in the logic circuitry,

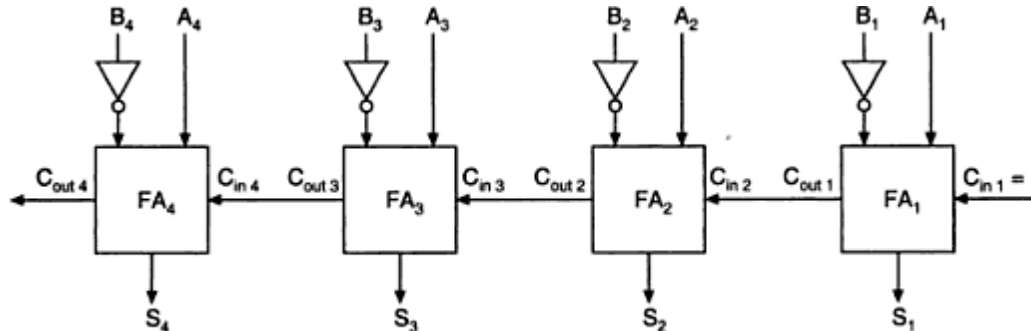
which lead to a time delay in the addition process. The carry propagation delay for each full-adder is the time between the application of the carry-in and the occurrence of the carry-out.

The 4-bit parallel adder, the sum (S_1) and carry-out (C_1) bits given by FA_1 are not valid, until after the propagation delay of FA_1 . Similarly, the sum S_2 and carry-out (C_2) bits given by FA_2 are not valid until after the cumulative propagation delay of two full adders (FA_1 and FA_2), and so on. At each stage, the sum bit is not valid until after the carry bits in all the preceding stages are valid. Carry bits must propagate or ripple through all stages before the most significant sum bit is valid. Thus, the total sum (the parallel output) is not valid until after the cumulative delay of all the adders.

The parallel adder in which the carry-out of each full-adder is the carry-in to the next most significant adder is called a ripple carry adder.. The greater the number of bits that a ripple carry adder must add, the greater the time required for it to perform a valid addition. If two numbers are added such that no carries occur between stages, then the add time is simply the propagation time through a single full-adder.

4- Bit Parallel Subtractor:

The subtraction of binary numbers can be carried out most conveniently by means of complements, the subtraction $A-B$ can be done by taking the 2's complement of B and adding it to A . The 2's complement can be obtained by taking the 1's complement and adding 1 to the least significant pair of bits. The 1's complement can be implemented with inverters as



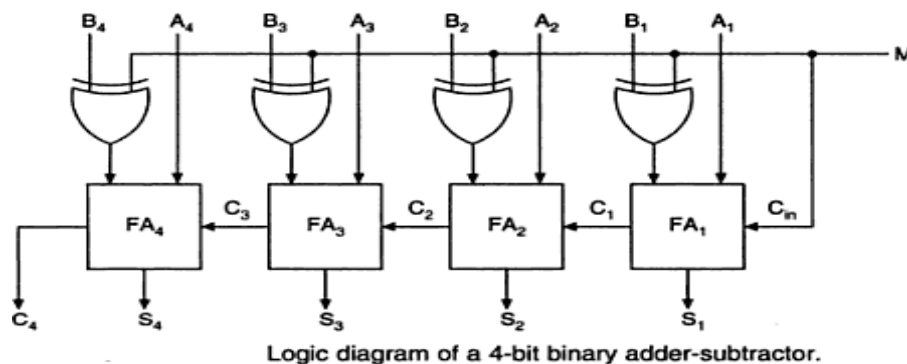
Logic diagram of a 4-bit parallel subtractor.

Binary-Adder Subtractor:

A 4-bit adder-subtractor, the addition and subtraction operations are combined into one circuit with one common binary adder. This is done by including an X-OR gate with each full-adder. The mode input M controls the operation. When $M=0$, the circuit is an adder, and when $M=1$, the circuit becomes a subtractor. Each X-OR gate receives input M and one of the inputs of B . When $M=0$, $B \oplus 0 = B$. The full-adder receives the value of B , the input carry is 0



and the circuit performs $A+B$, when $B \oplus 1 = B'$ and $C_1=1$. The B inputs are complemented and a 1 is through the input carry. The circuit performs the operation A plus the 2's complement of B.

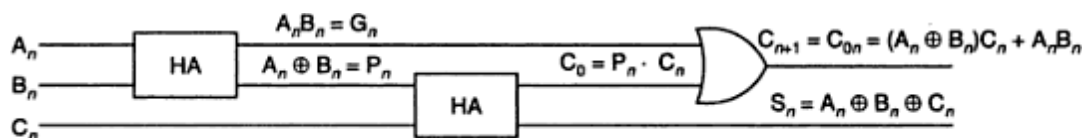


The Look-Ahead –Carry Adder:

In parallel-adder, the speed with which an addition can be performed is governed by the time required for the carries to propagate or ripple through all of the stages of the adder. The look-ahead carry adder speeds up the process by eliminating this ripple carry delay. It examines all the input bits simultaneously and also generates the carry-in bits for all the stages simultaneously.

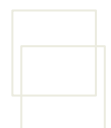
The method of speeding up the addition process is based on the two additional functions of the full-adder, called the carry generate and carry propagate functions.

Consider one full adder stage; say the n th stage of a parallel adder as shown in fig. we know that is made by two half adders and that the half adder contains an X-OR gate to produce the sum and an AND gate to produce the carry. If both the bits A_n and B_n are 1s, a carry has to be generated in this stage regardless of whether the input carry C_{in} is a 0 or a 1. This is called generated carry, expressed as $G_n = A_n \cdot B_n$ which has to appear at the output through the OR gate as shown in fig.



A full adder (n th stage of a parallel adder).

There is another possibility of producing a carry out. X-OR gate inside the half-adder at the input produces an intermediary sum bit- call it P_n –which is expressed as $P_n = A_n \oplus B_n$. Next P_n and C_n are added using the X-OR gate inside the second half adder to produce the final



sum bit and $S_n = P_n \oplus C_n$ where $P_n = A_n \oplus B_n$ and output carry $C_0 = P_n \cdot C_n = (A_n \oplus B_n) \cdot C_n$ which becomes carry for the (n+1) th stage.

Consider the case of both P_n and C_n being 1. The input carry C_n has to be propagated to the output only if P_n is 1. If P_n is 0, even if C_n is 1, the and gate in the second half-adder will inhibit C_n . the carry out of the nth stage is 1 when either $G_n=1$ or $P_n \cdot C_n=1$ or both G_n and $P_n \cdot C_n$ are equal to 1.

For the final sum and carry outputs of the nth stage, we get the following Boolean expressions.

$$S_n = P_n \oplus C_n \text{ where } P_n = A_n \oplus B_n$$

$$C_{n+1} = G_n + P_n \cdot C_n \text{ where } G_n = A_n \cdot B_n$$

Observe the recursive nature of the expression for the output carry at the nth stage which becomes the input carry for the (n+1)st stage .it is possible to express the output carry of a higher significant stage is the carry-out of the previous stage.

Based on these , the expression for the carry-outs of various full adders are as follows,

$$C_1 = G_0 + P_0 \cdot C_0$$

$$C_2 = G_1 + P_1 \cdot C_1 = G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot C_0$$

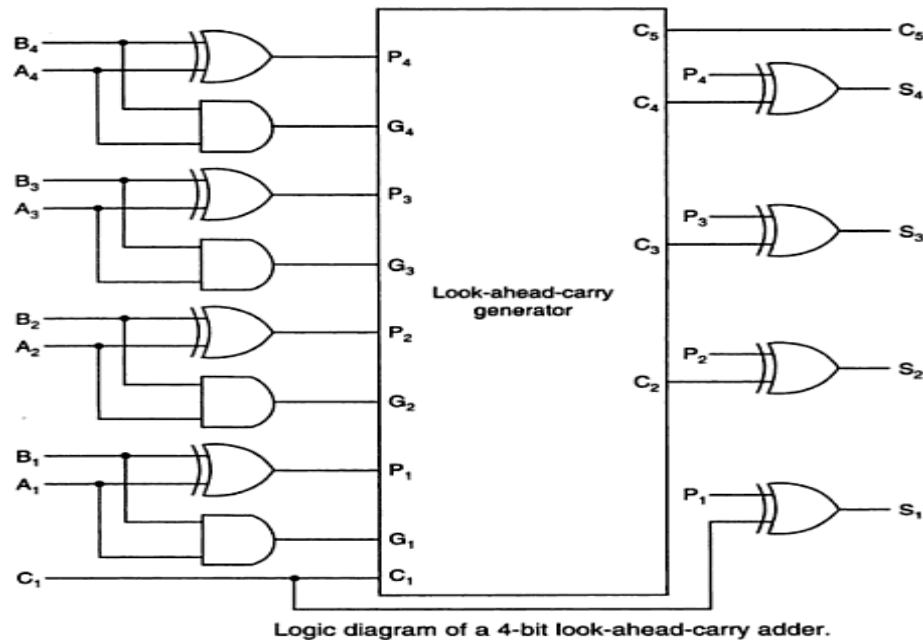
$$C_3 = G_2 + P_2 \cdot C_2 = G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot C_0$$

$$C_4 = G_3 + P_3 \cdot C_3 = G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_0$$

The general expression for n stages designated as 0 through $(n - 1)$ would be

$$C_n = G_{n-1} + P_{n-1} \cdot C_{n-1} = G_{n-1} + P_{n-1} \cdot G_{n-2} + P_{n-1} \cdot P_{n-2} \cdot G_{n-3} + \dots + P_{n-1} \cdot \dots \cdot P_0 \cdot C_0$$

Observe that the final output carry is expressed as a function of the input variables in SOP form. Which is two level AND-OR or equivalent NAND-NAND form. Observe that the full look-ahead scheme requires the use of OR gate with (n+1) inputs and AND gates with number of inputs varying from 2 to (n+1).



2's complement Addition and Subtraction using Parallel Adders:

Most modern computers use the 2's complement system to represent negative numbers and to perform subtraction operations of signed numbers can be performed using only the addition operation, if we use the 2's complement form to represent negative numbers.

The circuit shown can perform both addition and subtraction in the 2's complement. This adder/subtractor circuit is controlled by the control signal ADD/SUB'. When the ADD/SUB' level is HIGH, the circuit performs the addition of the numbers stored in registers A and B. When the ADD/Sub' level is LOW, the circuit subtracts the number in register B from the number in register A. The operation is:

When ADD/SUB' is a 1:

1. AND gates 1,3,5 and 7 are enabled, allowing B_0, B_1, B_2 and B_3 to pass to the OR gates 9,10,11,12. AND gates 2,4,6 and 8 are disabled, blocking $B_0', B_1', B_2',$ and B_3' from reaching the OR gates 9,10,11 and 12.
2. The two levels B_0 to B_3 pass through the OR gates to the 4-bit parallel adder, to be added to the bits A_0 to A_3 . The sum appears at the output S_0 to S_3 .
3. Add/SUB' = 1 causes no carry into the adder.

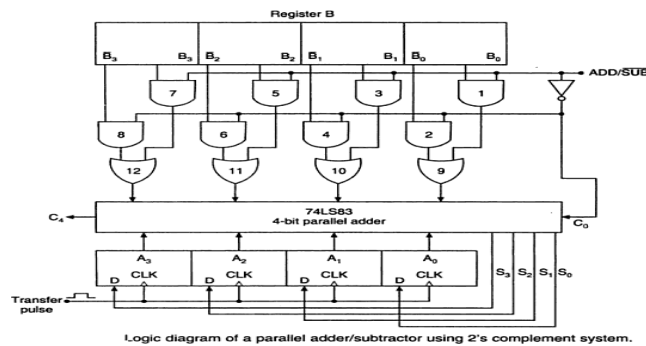
When ADD/SUB' is a 0:

1. AND gates 1,3,5 and 7 are disabled, allowing B_0, B_1, B_2 and B_3 from reaching the OR gates 9,10,11,12. AND gates 2,4,6 and 8 are enabled, blocking $B_0', B_1', B_2',$ and B_3' from reaching the OR gates.



2. The two levels B_0' to B_3' pass through the OR gates to the 4-bit parallel adder, to be added to the bits A_0 to A_3 . The C_0 is now 1. thus the number in register B is converted to its 2's complement form.
3. The difference appears at the output S_0 to S_3 .

Adders/Subtractors used for adding and subtracting signed binary numbers. In computers, the output is transferred into the register A (accumulator) so that the result of the addition or subtraction always end up stored in the register A. This is accomplished by applying a transfer pulse to the CLK inputs of register A.



Serial Adder:

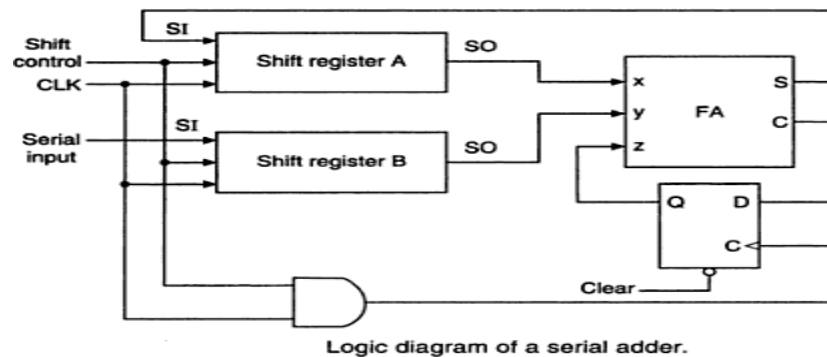
A serial adder is used to add binary numbers in serial form. The two binary numbers to be added serially are stored in two shift registers A and B. Bits are added one pair at a time through a single full adder (FA) circuit as shown. The carry out of the full-adder is transferred to a D flip-flop. The output of this flip-flop is then used as the carry input for the next pair of significant bits. The sum bit from the S output of the full-adder could be transferred to a third shift register. By shifting the sum into A while the bits of A are shifted out, it is possible to use one register for storing both augend and the sum bits. The serial input register B can be used to transfer a new binary number while the addend bits are shifted out during the addition.

The operation of the serial adder is:

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. The shift control enables both registers and carry flip-flop, so, at the clock pulse both registers are shifted once to the right, the sum bit from S enters the left most flip-flop of A, and the output carry is transferred into flip-flop Q. The shift control enables the registers for a number of clock pulses equal to the number of bits of the registers. For each succeeding clock pulse a new sum bit is transferred to A, a new carry is transferred to Q, and both registers are shifted once to the right. This process continues until the shift control is disabled. Thus the addition is accomplished by passing each pair of bits together with the previous carry through a single full adder circuit and transferring the sum, one bit at a time, into register A.



Initially, register A and the carry flip-flop are cleared to 0 and then the first number is added from B. While B is shifted through the full adder, a second number is transferred to it through its serial input. The second number is then added to the content of register A while a third number is transferred serially into register B. This can be repeated to form the addition of two, three, or more numbers and accumulate their sum in register A.



Difference between Serial and Parallel Adders:

The parallel adder registers with parallel load, whereas the serial adder uses shift registers. The number of full adder circuits in the parallel adder is equal to the number of bits in the binary numbers, whereas the serial adder requires only one full adder circuit and a carry flip-flop. Excluding the registers, the parallel adder is a combinational circuit, whereas the serial adder is a sequential circuit. The sequential circuit in the serial adder consists of a full-adder and a flip-flop that stores the output carry.

BCD Adder:

The BCD addition process:

1. Add the 4-bit BCD code groups for each decimal digit position using ordinary binary addition.
2. For those positions where the sum is 9 or less, the sum is in proper BCD form and no correction is needed.
3. When the sum of two digits is greater than 9, a correction of 0110 should be added to that sum, to produce the proper BCD result. This will produce a carry to be added to the next decimal position.

A BCD adder circuit must be able to operate in accordance with the above steps. In other words, the circuit must be able to do the following:

1. Add two 4-bit BCD code groups, using straight binary addition.



- Determine, if the sum of this addition is greater than 1101 (decimal 9); if it is, add 0110 (decimal 6) to this sum and generate a carry to the next decimal position.

The first requirement is easily met by using a 4-bit binary parallel adder such as the 74LS83 IC. For example, if the two BCD code groups $A_3A_2A_1A_0$ and $B_3B_2B_1B_0$ are applied to a 4-bit parallel adder, the adder will output $S_4S_3S_2S_1S_0$, where S_4 is actually C_4 , the carry-out of the MSB bits.

The sum outputs $S_4S_3S_2S_1S_0$ can range anywhere from 00000 to 100109 when both the BCD code groups are 1001 (=9). The circuitry for a BCD adder must include the logic needed to detect whenever the sum is greater than 01001, so that the correction can be added in. Those cases, where the sum is greater than 1001 are listed as:

S_4	S_3	S_2	S_1	S_0	Decimal number
0	1	0	1	0	10
0	1	0	1	1	11
0	1	1	0	0	12
0	1	1	0	1	13
0	1	1	1	0	14
0	1	1	1	1	15
1	0	0	0	0	16
1	0	0	0	1	17
1	0	0	1	0	18

Let us define a logic output X that will go HIGH only when the sum is greater than 01001 (i.e., for the cases in table). If examine these cases, see that X will be HIGH for either of the following conditions:

- Whenever $S_4=1$ (sum greater than 15)
- Whenever $S_3=1$ and either S_2 or S_1 or both are 1 (sum 10 to 15)

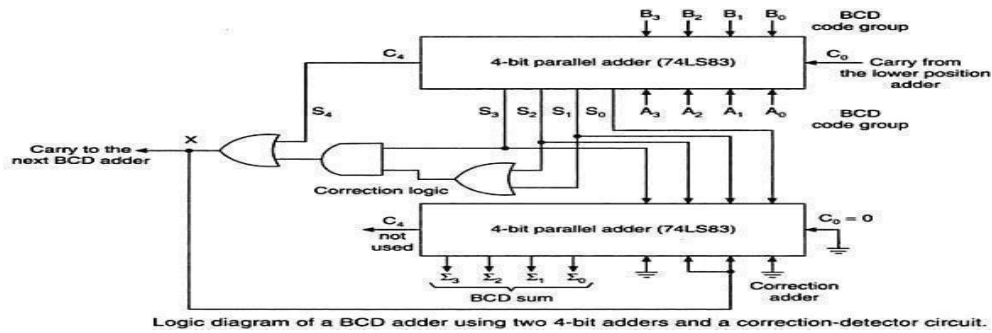
This condition can be expressed as

$$X = S_4 + S_3(S_2 + S_1)$$

Whenever $X=1$, it is necessary to add the correction factor 0110 to the sum bits, and to generate a carry. The circuit consists of three basic parts. The two BCD code groups $A_3A_2A_1A_0$ and $B_3B_2B_1B_0$ are added together in the upper 4-bit adder, to produce the sum $S_4S_3S_2S_1S_0$. The logic gates shown implement the expression for X. The lower 4-bit adder will add the correction 0110 to the sum bits, only when $X=1$, producing the final BCD sum output represented by $\sum_3\sum_2\sum_1\sum_0$. The X is also the carry-out that is produced when the sum is greater than 01001. When $X=0$, there is no carry and no addition of 0110. In such cases, $\sum_3\sum_2\sum_1\sum_0 = S_3S_2S_1S_0$.



Two or more BCD adders can be connected in cascade when two or more digit decimal numbers are to be added. The carry-out of the first BCD adder is connected as the carry-in of the second BCD adder, the carry-out of the second BCD adder is connected as the carry-in of the third BCD adder and so on.



UNIT-IV

SEQUENTIAL CIRCUITS

The Basic Latch

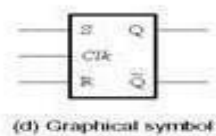
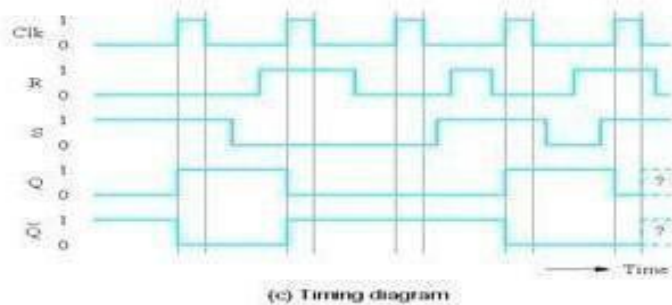
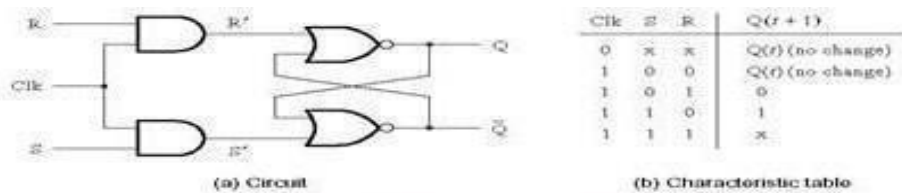
- **Basic latch** is a feedback connection of two NOR gates or two NAND gates
- It can store one bit of information

It can be set to 1 using the *S* input and reset to 0 using the *R* input

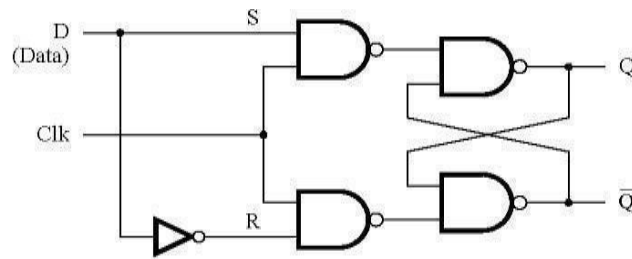
The Gated Latch

- **Gated latch** is a basic latch that includes input gating and a control signal
- The latch retains its existing state when the control input is equal to 0
- Its state may be changed when the control signal is equal to 1. In our discussion we referred to the control input as the clock
- We consider two types of gated latches:
 - ▮ **Gated SR latch** uses the *S* and *R* inputs to set the latch to 1 or reset it to 0, respectively.
 - ▮ **Gated D latch** uses the *D* input to force the latch into a state that has the same logic value as the *D* input.

Gated S/R Latch



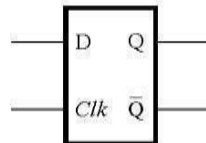
Gated D Latch



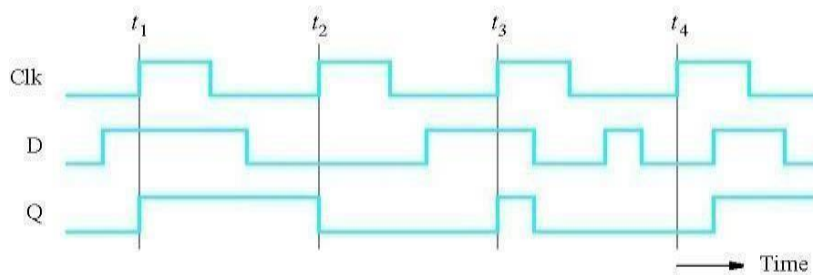
(a) Circuit

Clk	D	$Q(t+1)$
0	x	$Q(t)$
1	0	0
1	1	1

(b) Characteristic table



(c) Graphical symbol



(d) Timing diagram

Setup and Hold Times

● Setup Time t_{su}

The minimum time that the input signal must be stable prior to the edge of the clock signal.

● Hold Time t_h

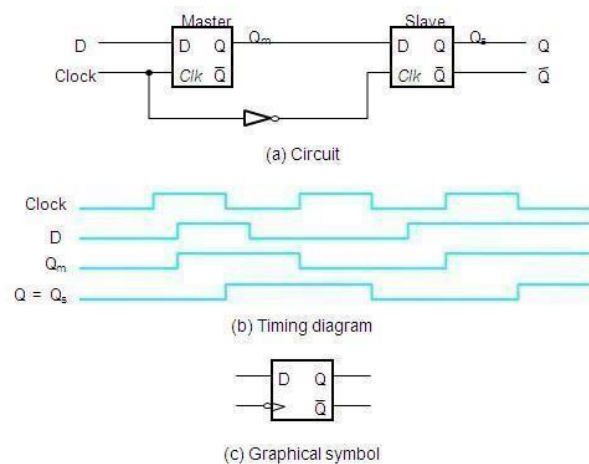
The minimum time that the input signal must be stable after the edge of the clock signal.

Flip-Flops

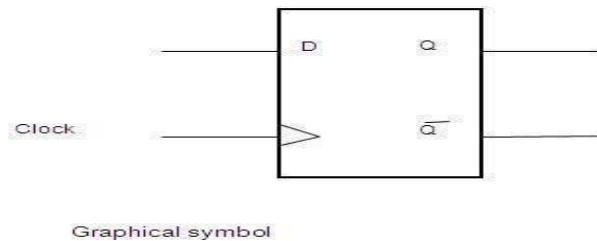
- A **flip-flop** is a storage element based on the gated latch principle
- It can have its output state changed only on the edge of the controlling clock signal
- We consider two types:

- **Edge-triggered flip-flop** is affected only by the input values present when the active edge of the clock occurs
- **Master-slave flip-flop** is built with two gated latches
- The master stage is active during half of the clock cycle, and the slave stage is active during the other half.
- The output value of the flip-flop changes on the edge of the clock that activates the transfer into the slave stage.

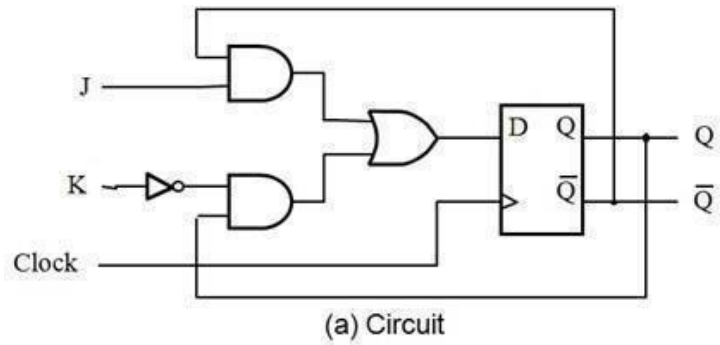
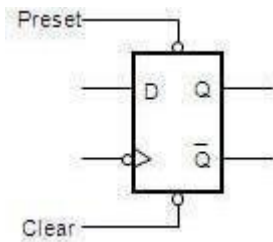
Master-Slave D Flip-Flop



A Positive-Edge-Triggered D Flip-Flop

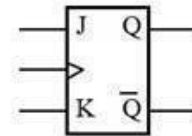


Master-Slave D Flip-Flop with *Clear* and *Preset*



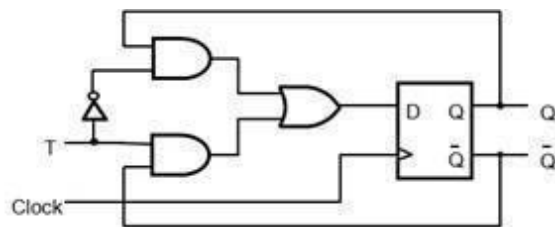
(a) Circuit

J	K	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	Q-bar(t)



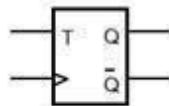
(b) Characteristic table (c) Graphical symbol

T Flip-Flop

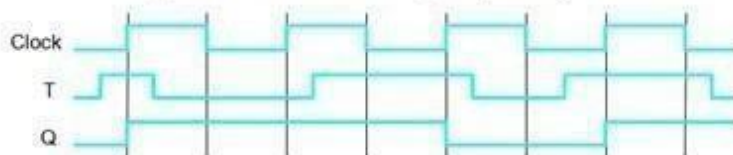


(a) Circuit

T	Q(t+1)
0	Q(t)
1	Q-bar(t)



(b) Characteristic table (c) Graphical symbol



(d) Timing diagram

Excitation Tables

Previous State -> Present State	D
0 -> 0	0
0 -> 1	1
1 -> 0	0
1 -> 1	1

Previous State -> Present State	J	K
0 -> 0	0	X
0 -> 1	1	X
1 -> 0	X	1
1 -> 1	X	0

Previous State -> Present State	S	R
0 -> 0	0	X
0 -> 1	1	0
1 -> 0	0	1
1 -> 1	X	0

Previous State -> Present State	T
0 -> 0	0
0 -> 1	1
1 -> 0	1
1 -> 1	0

Conversions of flip-flops

Example: Use JK-FF to realize D-FF

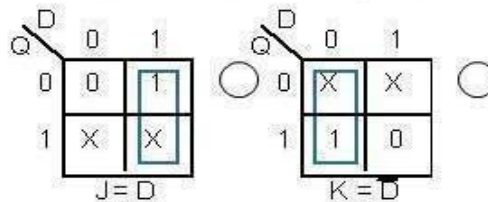
- 1) Start transition table for D-FF
- 2) Create K-maps to express J and K as functions of inputs (D, Q)
- 3) Fill in K-maps with appropriate values for J and K to cause the same state transition as in the D-FF transition table

D	Q	Q ⁺	J	K
0	0	0	0	X
0	1	0	X	1
1	0	1	1	X
1	1	1	X	0

State-Table

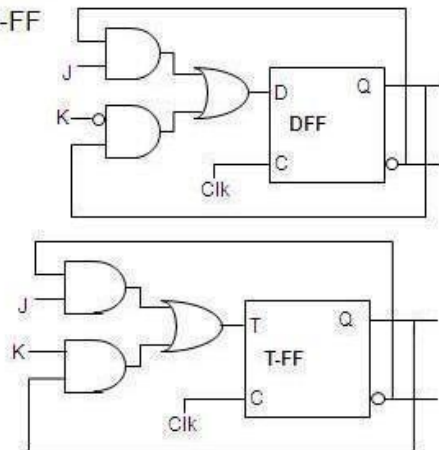
e.g.
when $D=Q=0$, then $Q^+=0$
the same transition $Q \rightarrow Q^+$
is realized with $J=0, K=X$

Q	Q ⁺	R	S	J	K	T	D
0	0	X	0	0	X	0	0
0	1	0	1	1	X	1	1
1	0	1	0	X	1	1	0
1	1	0	X	X	0	0	1



Example: Implement JK-FF using a D-FF

J	K	Q	Q ⁺	D	T
0	0	0	0	0	0
0	0	1	1	1	0
0	1	0	0	0	0
0	1	1	0	0	1
1	0	0	1	1	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	1



JK	Q	00	01	11	10
0	0	0	0	1	1
1	0	1	0	0	1

$$d = jQ + Kq$$

JK	Q	00	01	11	10
0	0	0	0	1	1
1	0	0	1	1	0

$$t = jQ + Kq$$

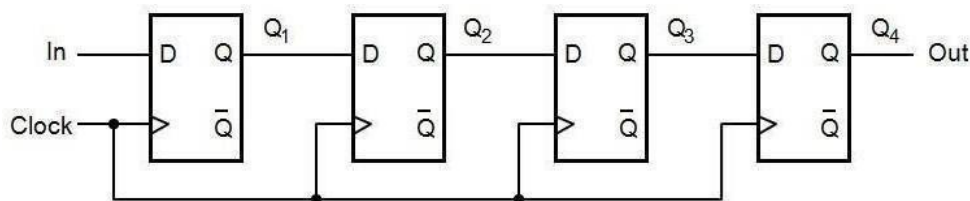
Sequential Circuit Design

- Steps in the design process for sequential circuits
- State Diagrams and State Tables □ Examples
- Steps in Design of a Sequential Circuit
 - 1. Specification – A description of the sequential circuit. Should include a detailing of the inputs, the outputs, and the operation. Possibly assumes that you have knowledge of digital system basics.
 - 2. Formulation: Generate a state diagram and/or a state table from the statement of the problem.
 - 3. State Assignment: From a state table assign binary codes to the states.
 - 4. Flip-flop Input Equation Generation: Select the type of flip-flop for the circuit and generate the needed input for the required state transitions
 - 5. Output Equation Generation: Derive output logic equations for generation of the output from the inputs and current state.
 - 6. Optimization: Optimize the input and output equations. Today, CAD systems are typically used for this in real systems.
 - 7. Technology Mapping: Generate a logic diagram of the circuit using ANDs, ORs, Inverters, and F/Fs.
 - 8. Verification: Use a HDL to verify the design

Registers and Counters

- An n -bit register is a cascade of n flip-flops and can store an n -bit binary data
- A counter can count occurrences of events and can generate timing intervals for control purposes

A Simple Shift Register

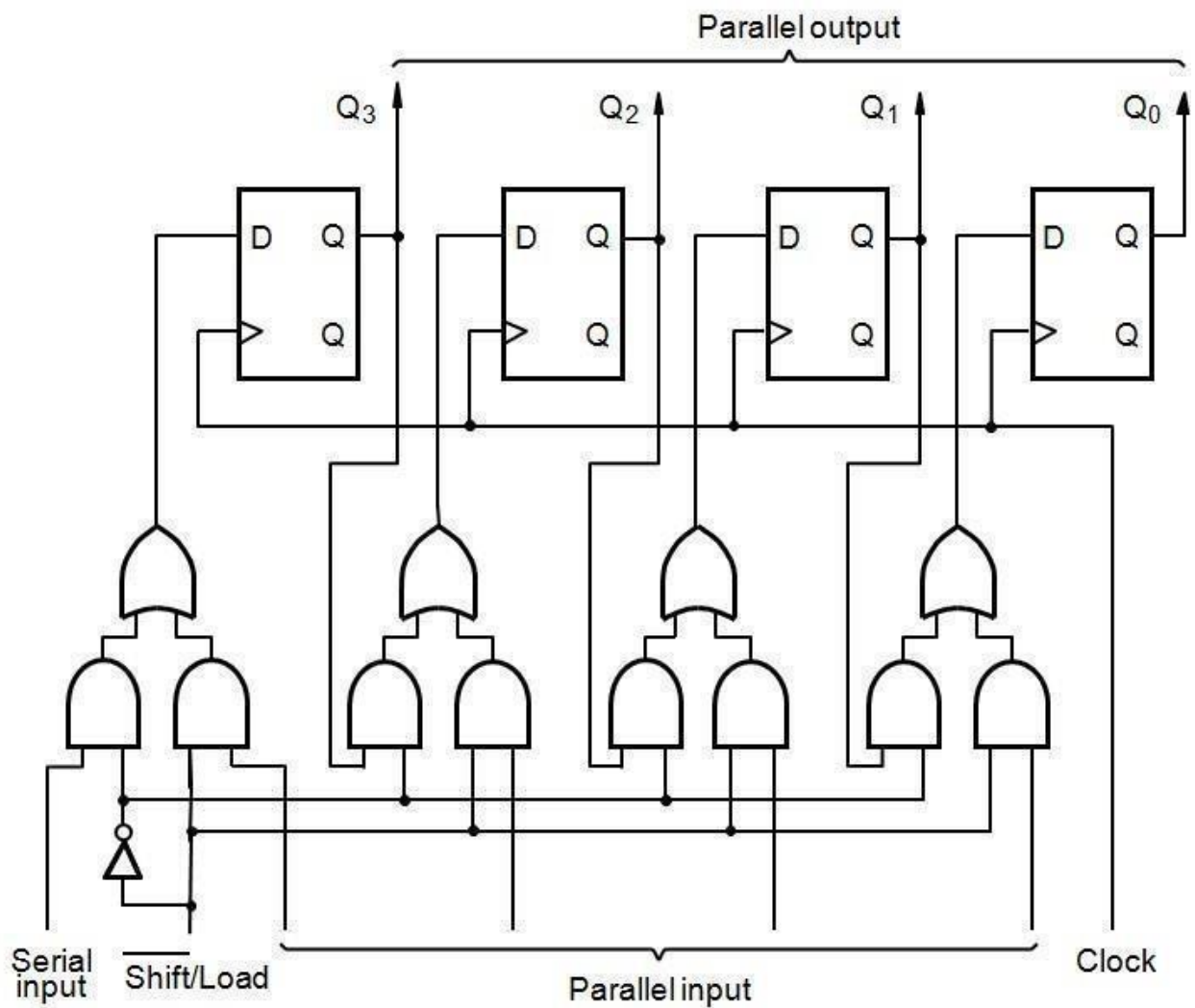


(a) Circuit

	In	Q ₁	Q ₂	Q ₃	Q ₄ = Out
t_0	1	0	0	0	0
t_1	0	1	0	0	0
t_2	1	0	1	0	0
t_3	1	1	0	1	0
t_4	1	1	1	0	1
t_5	0	1	1	1	0
t_6	0	0	1	1	1
t_7	0	0	0	1	1

(b) A sample sequence

Parallel-Access Shift Register



Counters

- Counters are a specific type of sequential circuit.
- Like registers, the state, or the flip-flop values themselves, serves as the “output.”
- The output value increases by one on each clock cycle.
- After the largest value, the output “wraps around” back to 0.
- Using two bits, we’d get something like this:

Present State		Next State	
A	B	A	B
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

Benefits of counters

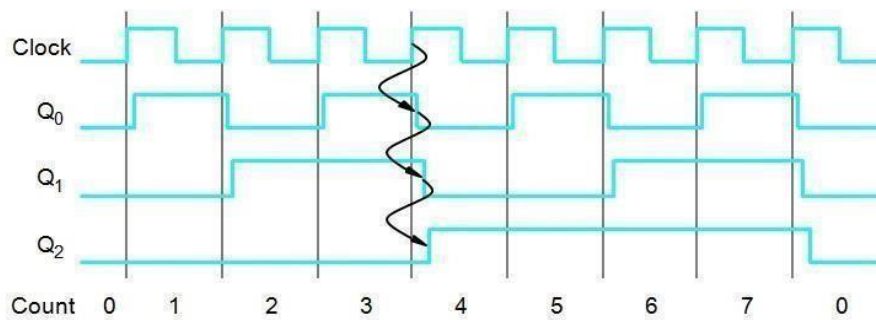
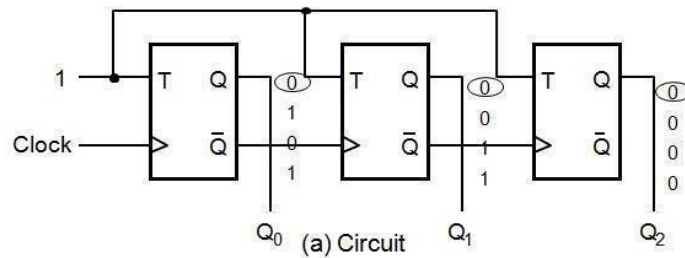
- Counters can act as simple clocks to keep track of “time.” •

You may need to record how many times something has happened.

- How many bits have been sent or received?
 - How many steps have been performed in some computation?
- All processors contain a program counter, or PC.
 - Programs consist of a list of instructions that are to be executed one after another (for the most part).
 - The PC keeps track of the instruction currently being executed.
 - The PC increments once on each clock cycle, and the next program instruction is then executed.

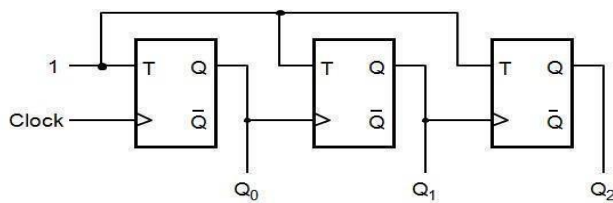
A Three-Bit Up-Counter

- Q_1 is connected to clk , Q_2 and Q_3 are clocked by Q' of the preceding stage (hence called asynchronous or ripple counter)

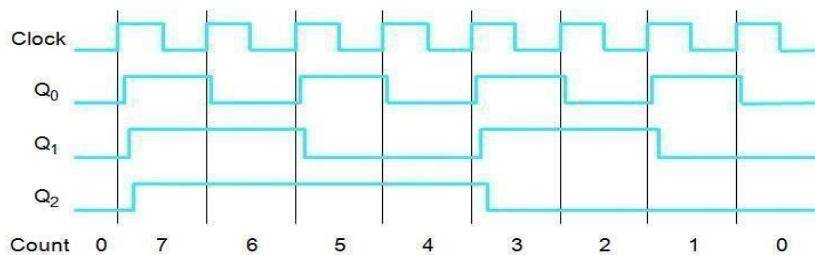


(b) Timing diagram

A Three-Bit Down-Counter



(a) Circuit



(b) Timing diagram

Shift registers:

In digital circuits, a **shift register** is a cascade of flip-flops sharing the same clock, in which the output of each flip-flop is connected to the "data" input of the next flip-flop in the chain, resulting in a circuit that shifts by one position the "bit array" stored in it, *shifting in* the data present at its input and *shifting out* the last bit in the array, at each transition of the clock input. More generally, a **shift register** may be multidimensional, such that its "data in" and stage outputs are themselves bit arrays: this is implemented simply by running several shift registers of the same bit-length in parallel.

Shift registers can have both parallel and serial inputs and outputs. These are often configured as **serial-in, parallel-out** (SIPO) or as **parallel-in, serial-out** (PISO). There are also types that have both serial and parallel input and types with serial and parallel output. There are also **bi-directional** shift registers which allow shifting in both directions: L→R or R→L. The serial input and last output of a shift register can also be connected to create a **circular shift register**

Shift registers are a type of logic circuits closely related to counters. They are basically for the storage and transfer of digital data.

Buffer register:

The buffer register is the simple set of registers. It simply stores the binary word. The buffer may be controlled buffer. Most of the buffer registers used D Flip-flops.

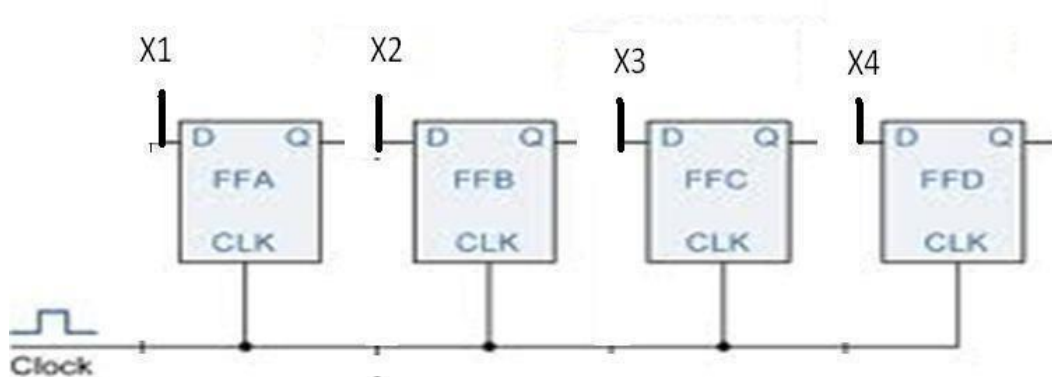


Figure: logic diagram of 4-bit buffer register

The figure shows a 4-bit buffer register. The binary word to be stored is applied to the data terminals. On the application of clock pulse, the output word becomes the same as the word applied at the terminals. i.e., the input word is loaded into the register by the application of clock pulse.

When the positive clock edge arrives, the stored word becomes:

$$Q_4Q_3Q_2Q_1 = X_4X_3X_2X_1$$
$$Q = X$$

Controlled buffer register:

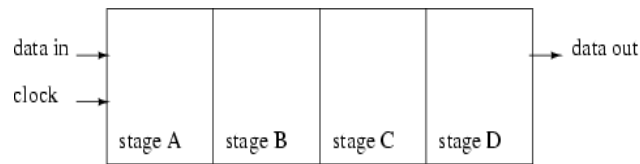
If goes LOW, all the FFs are RESET and the output becomes, $Q = 0000$.

When is HIGH, the register is ready for action. LOAD is the control input. When LOAD is HIGH, the data bits X can reach the D inputs of FF's.

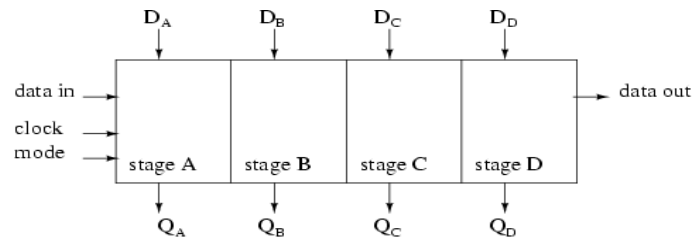
$$Q_4Q_3Q_2Q_1 = X_4X_3X_2X_1$$
$$Q = X$$

When load is low, the X bits cannot reach the FF's.

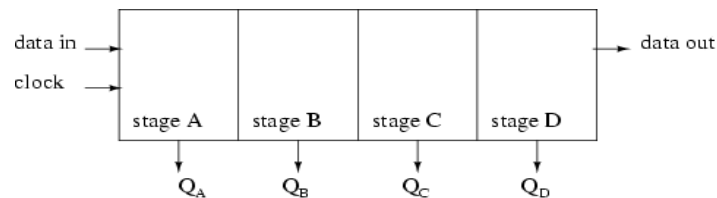
Data transmission in shift registers:



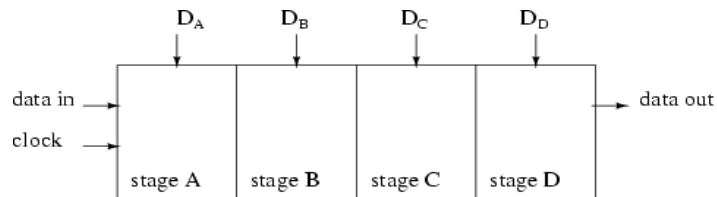
Serial-in, serial-out shift register with 4-stages



Parallel-in, parallel-out shift register with 4-stages



Serial-in, parallel-out shift register with 4-stages



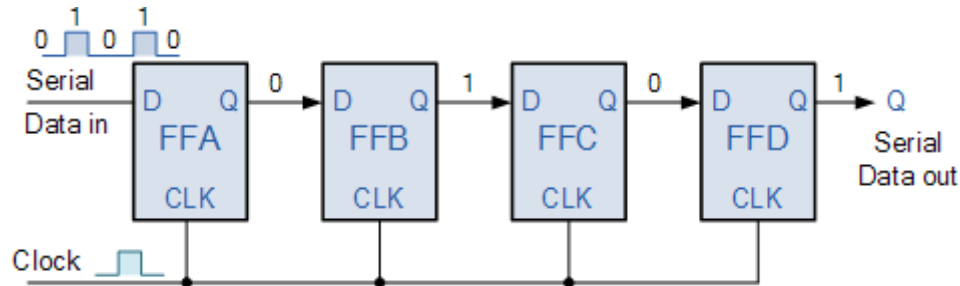
Parallel-in, serial-out shift register with 4-stages

A number of ff's connected together such that data may be shifted into and shifted out of them is called shift register. data may be shifted into or out of the register in serial form or in parallel form. There are four basic types of shift registers.

1. Serial in, serial out, shift right, shift registers
2. Serial in, serial out, shift left, shift registers
3. Parallel in, serial out shift registers
4. Parallel in, parallel out shift registers

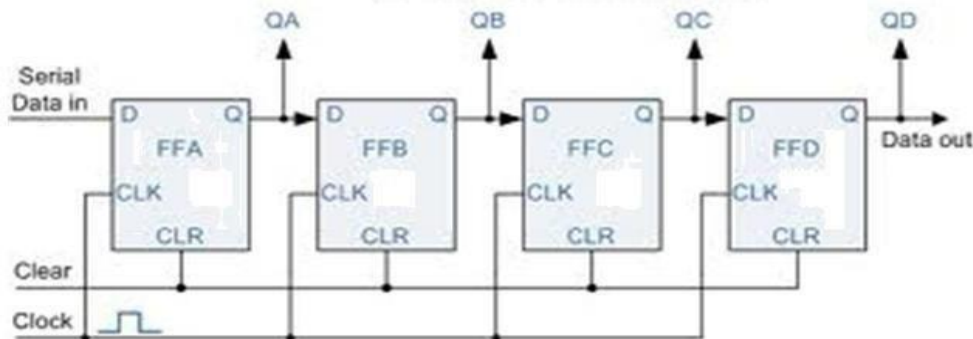
Serial IN, serial OUT, shift right, shift left register:

The logic diagram of 4-bit serial in serial out, right shift register with four stages. The register can store four bits of data. Serial data is applied at the input D of the first FF. the Q output of the first FF is connected to the D input of another FF. the data is outputted from the Q terminal of the last FF.



When serial data is transferred into a register, each new bit is clocked into the first FF at the positive going edge of each clock pulse. The bit that was previously stored by the first FF is transferred to the second FF. the bit that was stored by the Second FF is transferred to the third FF.

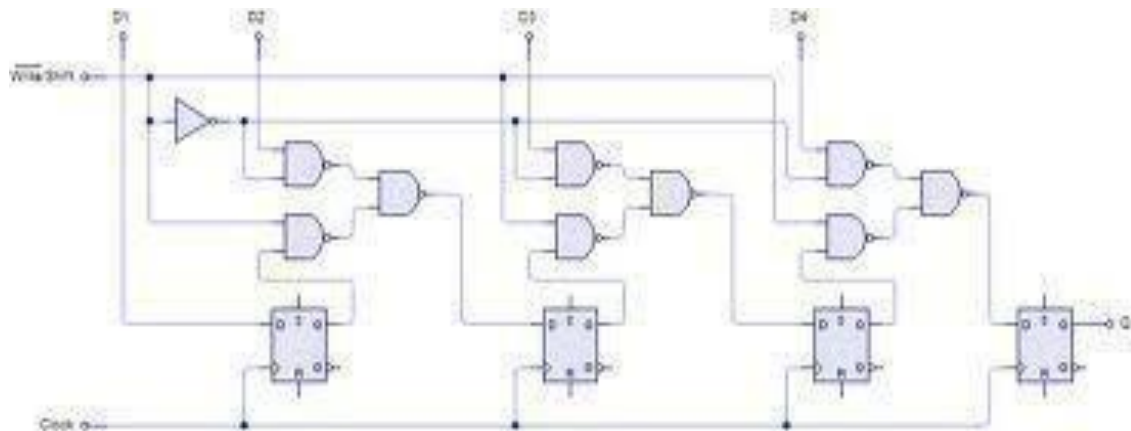
Serial-in, parallel-out, shift register:



In this type of register, the data bits are entered into the register serially, but the data stored in the register is shifted out in parallel form.

Once the data bits are stored, each bit appears on its respective output line and all bits are available simultaneously, rather than on a bit-by-bit basis with the serial output. The serial-in, parallel out, shift register can be used as serial-in, serial out, shift register if the output is taken from the Q terminal of the last FF.

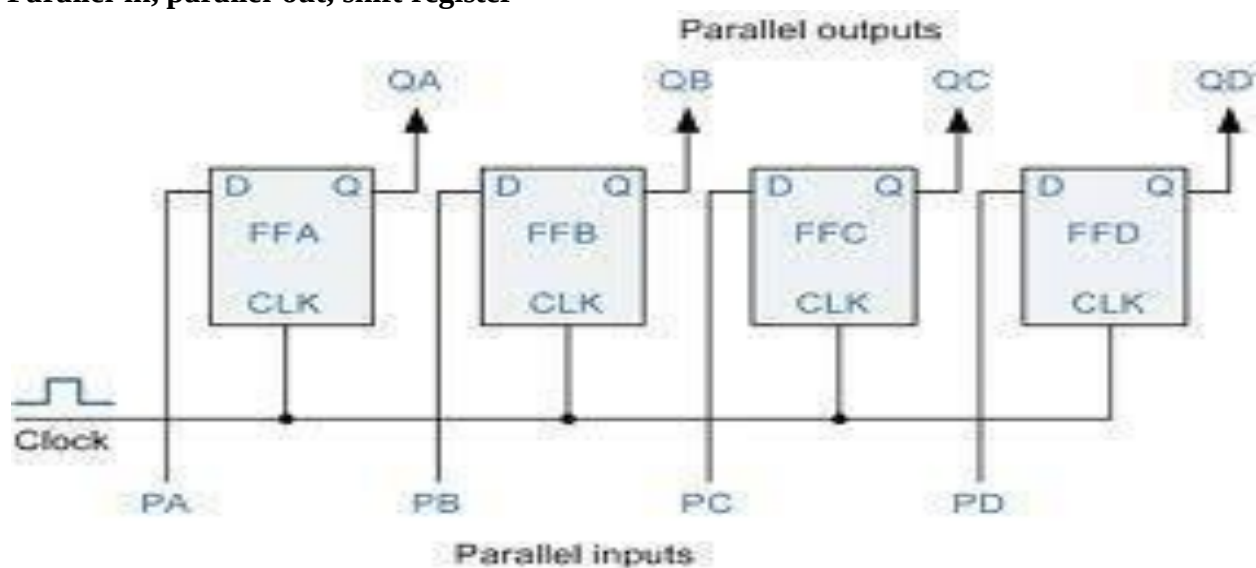
Parallel-in, serial-out, shift register:



For a parallel-in, serial out, shift register, the data bits are entered simultaneously into their respective stages on parallel lines, rather than on a bit-by-bit basis on one line as with serial data bits are transferred out of the register serially. On a bit-by-bit basis over a single line.

There are four data lines A,B,C,D through which the data is entered into the register in parallel form. The signal shift/ load allows the data to be entered in parallel form into the register and the data is shifted out serially from terminal Q4

Parallel-in, parallel-out, shift register



In a parallel-in, parallel-out shift register, the data is entered into the register in parallel form, and also the data is taken out of the register in parallel form. Data is applied to the D input terminals of the FF's. When a clock pulse is applied, at the positive going edge of the pulse, the D inputs are shifted into the Q outputs of the FFs. The register now stores the data. The stored data is available instantaneously for shifting out in parallel form.

Bidirectional shift register:

A bidirectional shift register is one which the data bits can be shifted from left to right or from right to left. A fig shows the logic diagram of a 4-bit serial-in, serial out, bidirectional shift register. Right/left is the mode signal, when right /left is a 1, the logic circuit works as a shift-register. the bidirectional operation is achieved by using the mode signal and two NAND gates and one OR gate for each stage.

A HIGH on the right/left control input enables the AND gates G1, G2, G3 and G4 and disables the AND gates G5, G6, G7 and G8, and the state of Q output of each FF is passed through the gate to the D input of the following FF. when a clock pulse occurs, the data bits are then effectively shifted one place to the right. A LOW on the right/left control inputs enables the AND gates G5, G6, G7 and G8 and disables the And gates G1, G2, G3 and G4 and the Q output of each FF is passed to the D input of the preceding FF. when a clock pulse occurs, the data bits are then effectively shifted one place to the left. Hence, the circuit works as a bidirectional shift register

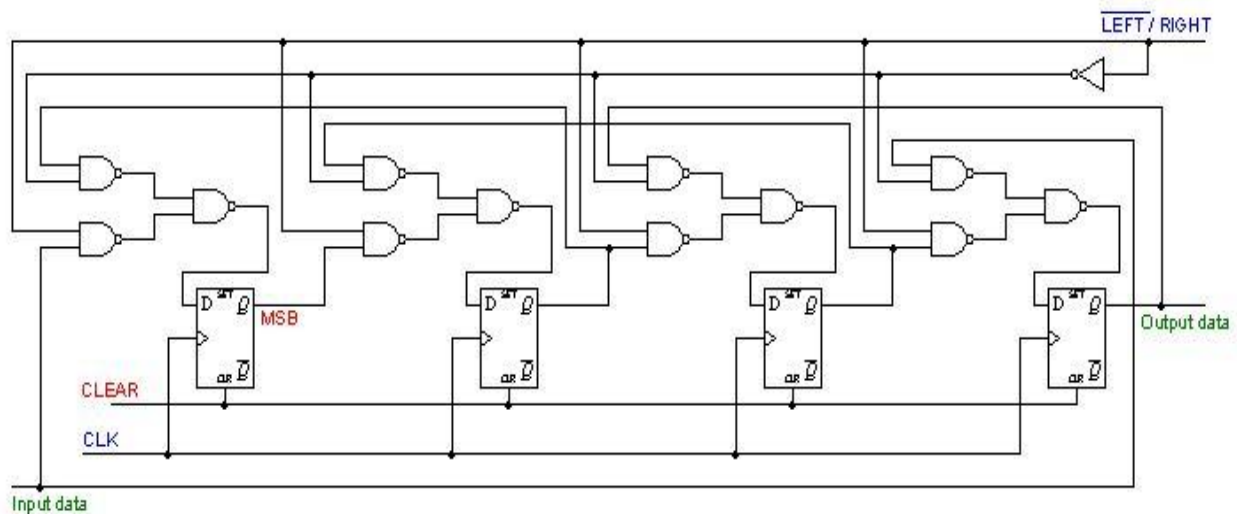


Figure: logic diagram of a 4-bit bidirectional shift register

Universal shift register:

A register is capable of shifting in one direction only is a unidirectional shift register. One that can shift 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 registers. Universal shift register is a bidirectional register, whose input can be either in serial form or in parallel form and whose output also can be in serial form or I parallel form.

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 serial input and output lines associated with the shift-right

4. A shift-left control to enable the shift-left operation and serial input and output lines associated with the shift-left
5. A parallel loads 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 the presence of the clock.

A universal shift register can be realized using multiplexers. The below fig shows the logic diagram of a 4-bit universal shift register that has all capabilities. It consists of 4 D flip-flops and four multiplexers. The four multiplexers have two common selection inputs s_1 and s_0 . Input 0 in each multiplexer is selected when $S_1S_0=00$, input 1 is selected when $S_1S_0=01$ and input 2 is selected when $S_1S_0=10$ and input 4 is selected when $S_1S_0=11$. The selection inputs control the mode of operation of the register according to the functions entries. When $S_1S_0=0$, the present value of the register is applied to the D inputs of flip-flops. The condition forms a path from the output of each flip-flop into the input of the same flip-flop. The next clock edge transfers into each flip-flop the binary value it held previously, and no change of state occurs. When $S_1S_0=01$, terminal 1 of the multiplexer inputs have a path to the D inputs of the flip-flop. This causes a shift-right operation, with serial input transferred into flip-flop A_4 . When $S_1S_0=10$, a shift left operation results with the other serial input going into flip-flop A_1 . Finally when $S_1S_0=11$, the binary information on the parallel input lines is transferred into the register simultaneously during the next clock cycle

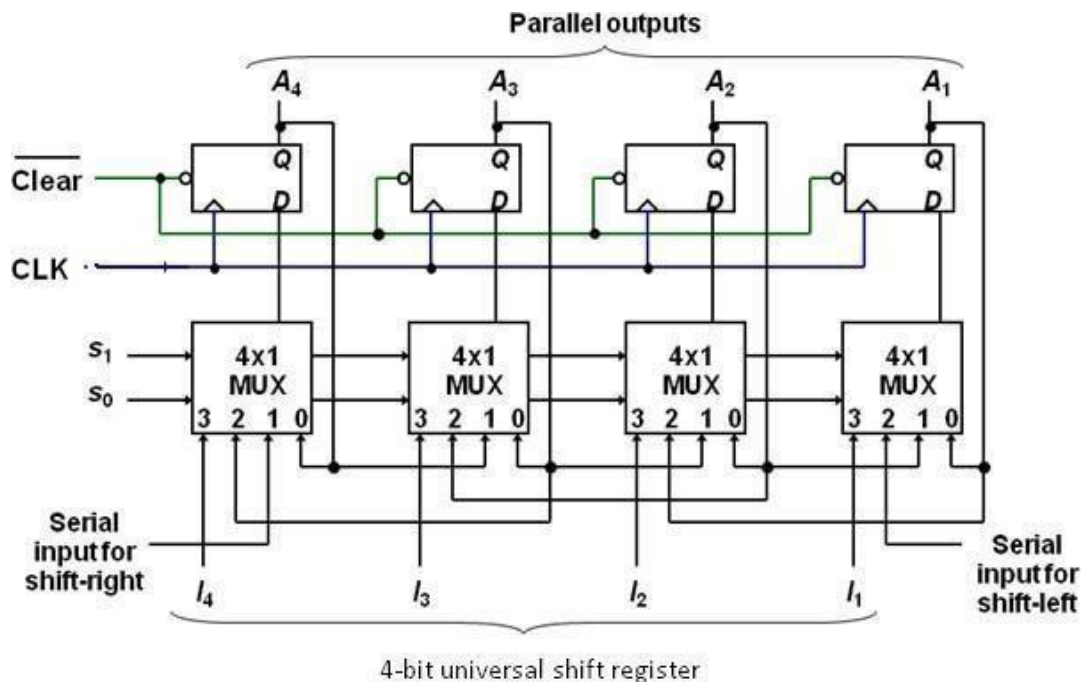


Figure: logic diagram 4-bit universal shift register

Function table for the register

mode control		
S0	S1	register operation
0	0	No change
0	1	Shift Right
1	0	Shift left
1	1	Parallel load

Counters:

Counter is a device which stores (and sometimes displays) the number of times particular event or process has occurred, often in relationship to a clock signal. A Digital counter is a set of flip flops whose state change in response to pulses applied at the input to the counter. Counters may be asynchronous counters or synchronous counters. Asynchronous counters are also called ripple counters

In electronics counters can be implemented quite easily using register-type circuits such as the flip-flops and a wide variety of classifications exist:

- Asynchronous (ripple) counter – changing state bits are used as clocks to subsequent state flip-flops
- Synchronous counter – all state bits change under control of a single clock
- Decade counter – counts through ten states per stage
- Up/down counter – counts both up and down, under command of a control input
- Ring counter – formed by a shift register with feedback connection in a ring
- Johnson counter – a *twisted* ring counter
- ☐ Cascaded counter
- ☐ Modulus counter.

Each is useful for different applications. Usually, counter circuits are digital in nature, and count in natural binary. Many types of counter circuits are available as digital building blocks, for example a number of chips in the 4000 series implement different counters.

Occasionally there are advantages to using a counting sequence other than the natural binary sequence such as the binary coded decimal counter, a linear feed-back shift register counter, or a gray-code counter.

Counters are useful for digital clocks and timers, and in oven timers, VCR clocks, etc.

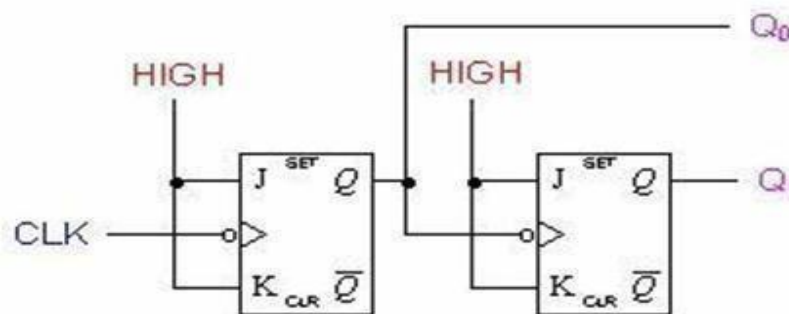
Asynchronous counters:

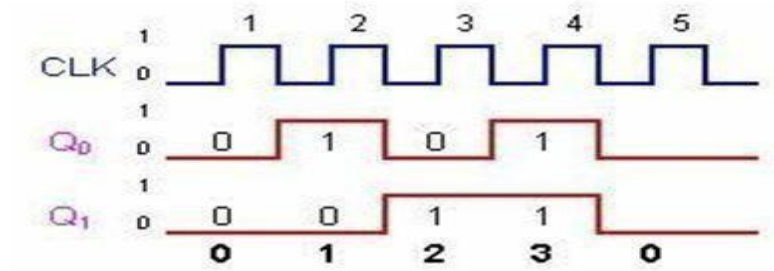
An asynchronous (ripple) counter is a single [JK-type flip-flop](#), with its J (data) input fed from its own inverted output. This circuit can store one bit, and hence can count from zero to one before it overflows (starts over from 0). This counter will increment once for every clock cycle and takes two clock cycles to overflow, so every cycle it will alternate between a transition from 0 to 1 and a transition from 1 to 0. Notice that this creates a new clock with a 50% [duty cycle](#) at exactly half the frequency of the input clock. If this output is then used as the clock signal for a similarly arranged D flip-flop (remembering to invert the output to the input), one will get another 1 bit counter that counts half as fast. Putting them together yields a two-bit counter:

Two-bit ripple up-counter using negative edge triggered flip flop:

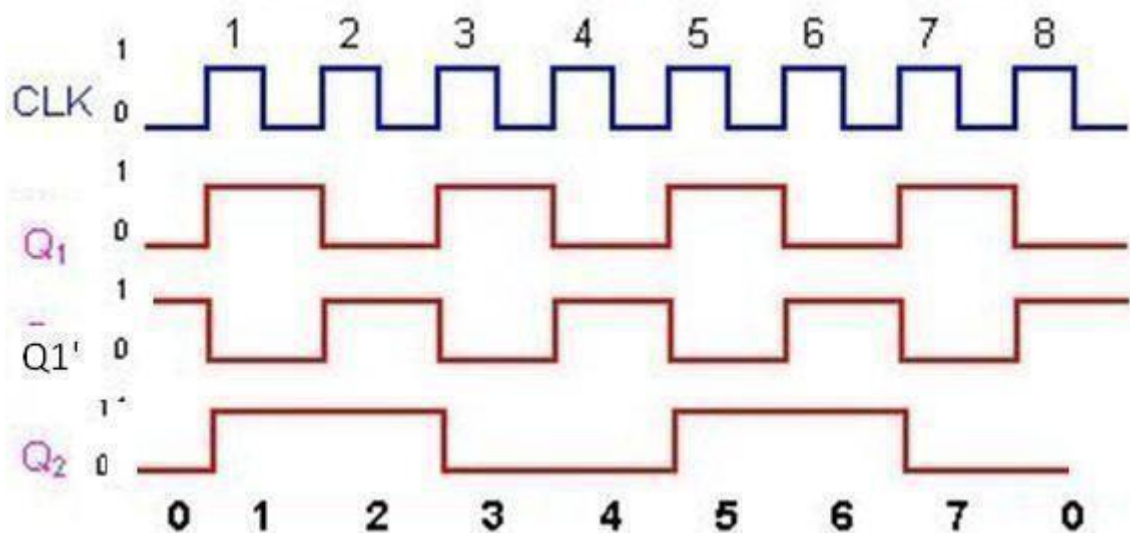
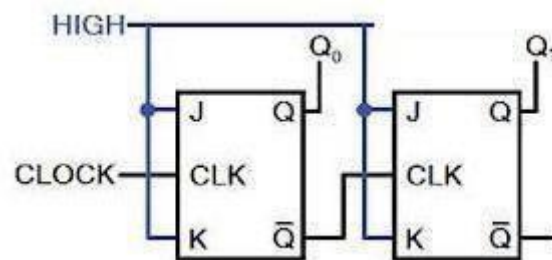
Two bit ripple counter used two flip-flops. There are four possible states from 2 – bit up-counting I.e. 00, 01, 10 and 11.

- The counter is initially assumed to be at a state 00 where the outputs of the two flip-flops are noted as Q_1Q_0 . Where Q_1 forms the MSB and Q_0 forms the LSB.
- For the negative edge of the first clock pulse, output of the first flip-flop FF_1 toggles its state. Thus Q_1 remains at 0 and Q_0 toggles to 1 and the counter state are now read as 01.
- During the next negative edge of the input clock pulse FF_1 toggles and $Q_0 = 0$. The output Q_0 being a clock signal for the second flip-flop FF_2 and the present transition acts as a negative edge for FF_2 thus toggles its state $Q_1 = 1$. The counter state is now read as 10.
- For the next negative edge of the input clock to FF_1 output Q_0 toggles to 1. But this transition from 0 to 1 being a positive edge for FF_2 output Q_1 remains at 1. The counter state is now read as 11.
- For the next negative edge of the input clock, Q_0 toggles to 0. This transition from 1 to 0 acts as a negative edge clock for FF_2 and its output Q_1 toggles to 0. Thus the starting state 00 is attained. Figure shown below





Two-bit ripple down-counter using negative edge triggered flip flop:



A 2-bit down-counter counts in the order 0,3,2,1,0,1.....,i.e, 00,11,10,01,00,11,etc. the above fig. shows ripple down counter, using negative edge triggered J-K FFs and its timing diagram.

- For down counting, $Q1'$ of FF1 is connected to the clock of Ff2. Let initially all the FF1 toggles, so, $Q1$ goes from a 0 to a 1 and $Q1'$ goes from a 1 to a 0.

- The negative-going signal at Q_1' is applied to the clock input of FF2, toggles FF2 and, therefore, Q_2 goes from a 0 to a 1. so, after one clock pulse $Q_2=1$ and $Q_1=1$, I.e., the state of the counter is 11.
- At the negative-going edge of the second clock pulse, Q_1 changes from a 1 to a 0 and Q_1' from a 0 to a 1.
- This positive-going signal at Q_1' does not affect FF2 and, therefore, Q_2 remains at a 1. Hence, the state of the counter after second clock pulse is 10
- At the negative going edge of the third clock pulse, FF1 toggles. So Q_1 , goes from a 0 to a 1 and Q_1' from 1 to 0. This negative going signal at Q_1' toggles FF2 and, so, Q_2 changes from 1 to 0, hence, the state of the counter after the third clock pulse is 01.
- At the negative going edge of the fourth clock pulse, FF1 toggles. So Q_1 , goes from a 1 to a 0 and Q_1' from 0 to 1. . This positive going signal at Q_1' does not affect FF2 and, so, Q_2 remains at 0, hence, the state of the counter after the fourth clock pulse is 00.

Two-bit ripple up-down counter using negative edge triggered flip flop:

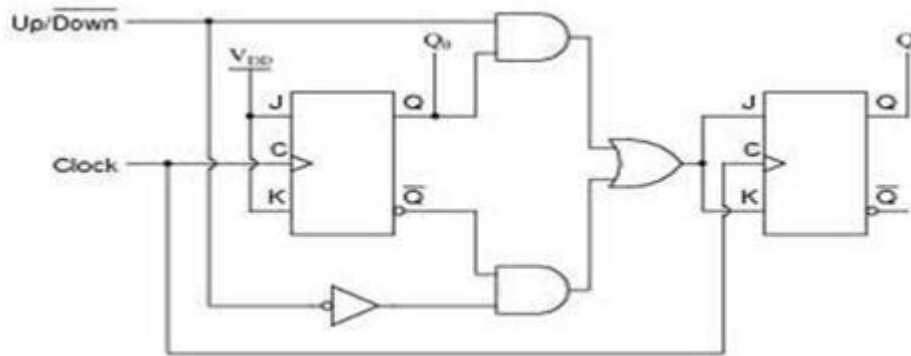


Figure: asynchronous 2-bit ripple up-down counter using negative edge triggered flip flop:

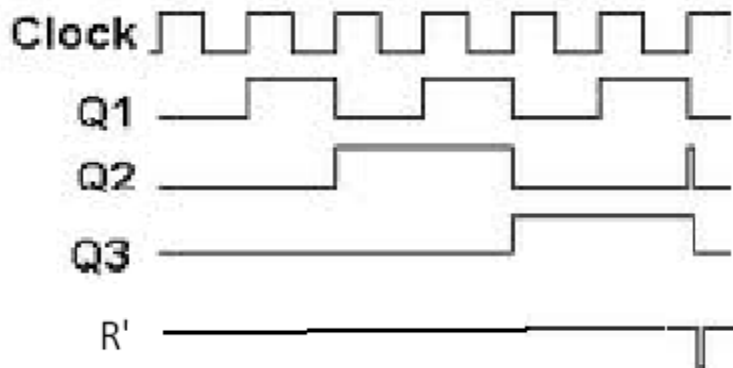
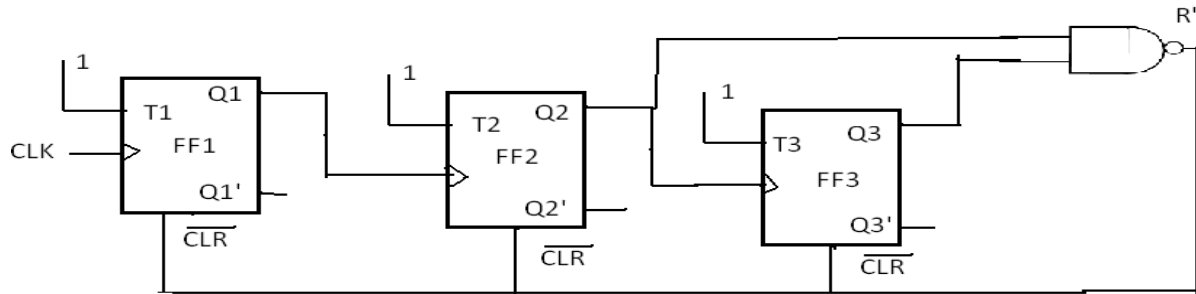
- As the name indicates an up-down counter is a counter which can count both in upward and downward directions. An up-down counter is also called a forward/backward counter or a bidirectional counter. So, a control signal or a mode signal M is required to choose the direction of count. When $M=1$ for up counting, Q_1 is transmitted to clock of FF2 and when $M=0$ for down counting, Q_1' is transmitted to clock of FF2. This is achieved by using two AND gates and one OR gates. The external clock signal is applied to FF1.
- Clock signal to FF2 = $(Q_1 \cdot \text{Up}) + (Q_1' \cdot \text{Down}) = Q_1 M + Q_1' M'$

Design of Asynchronous counters:

To design a asynchronous counter, first we write the sequence, then tabulate the values of reset signal R for various states of the counter and obtain the minimal expression for R and R' using K-Map or any other method. Provide a feedback such that R and R' resets all the FF's after the desired count

Design of a Mod-6 asynchronous counter using T FFs:

A mod-6 counter has six stable states 000, 001, 010, 011, 100, and 101. When the sixth clock pulse is applied, the counter temporarily goes to 110 state, but immediately resets to 000 because of the feedback provided. It is a divide-by-6 counter, in the sense that it divides the input clock frequency by 6. It requires three FFs, because the smallest value of n satisfying the condition $N \leq 2^n$ is $n=3$; three FFs can have 8 possible states, out of which only six are utilized and the remaining two states 110 and 111, are invalid. If initially the counter is in 000 state, then after the sixth clock pulse, it goes to 001, after the second clock pulse, it goes to 010, and so on.



After sixth clock pulse it goes to 000. For the design, write the truth table with present state outputs Q_3 , Q_2 and Q_1 as the variables, and reset R as the output and obtain an expression for R in terms of Q_3 , Q_2 , and Q_1 that decides the feedback into be provided. From the truth table, $R = Q_3Q_2$. For active-low Reset, R' is used. The reset pulse is of very short duration, of the order of nanoseconds and it is equal to the propagation delay time of the NAND gate used. The expression for R can also be determined as follows.

$$R=0 \text{ for } 000 \text{ to } 101, R=1 \text{ for } 110, \text{ and } R=X \text{ for } 111$$

Therefore,

$$R = Q_3Q_2Q_1' + Q_3Q_2Q_1 = Q_3Q_2$$

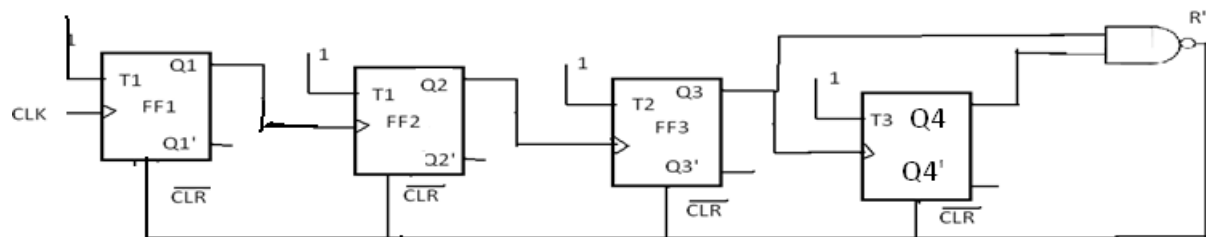
The logic diagram and timing diagram of Mod-6 counter is shown in the above fig.

The truth table is as shown in below.

After pulses	States			
	Q3	Q2	Q1	R
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	0
4	1	0	0	0
5	1	0	1	0
6	1	1	0	1
	↓	↓	↓	
	0	0	0	0
7	0	0	0	0

Design of a mod-10 asynchronous counter using T-flip-flops:

A mod-10 counter is a decade counter. It is also called a BCD counter or a divide-by-10 counter. It requires four flip-flops (condition $10 \leq 2^n$ is $n=4$). So, there are 16 possible states, out of which ten are valid and remaining six are invalid. The counter has ten stable states, 0000 through 1001, i.e., it counts from 0 to 9. The initial state is 0000 and after nine clock pulses it goes to 1001. When the tenth clock pulse is applied, the counter goes to state 1010 temporarily, but because of the feedback provided, it resets to initial state 0000. So, there will be a glitch in the waveform of Q2. The state 1010 is a temporary state for which the reset signal $R=1$, $R=0$ for 0000 to 1001, and $R=C$ for 1011 to 1111.



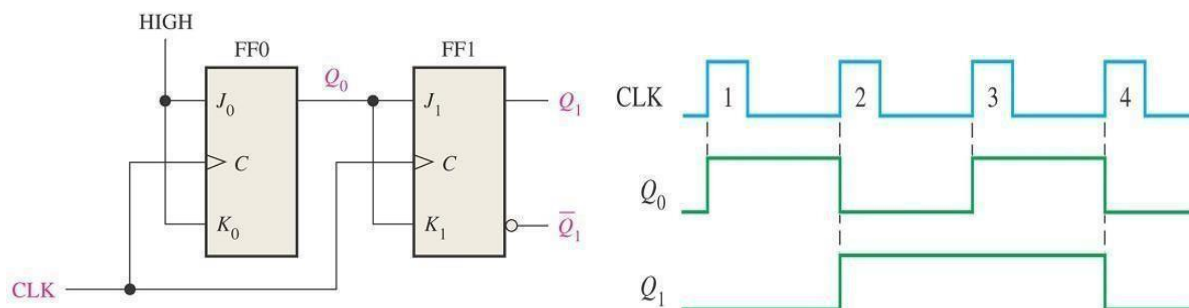
The count table and the K-Map for reset are shown in fig. from the K-Map $R=Q_4Q_2$. So, feedback is provided from second and fourth FFs. For active-HIGH reset, Q_4Q_2 is applied to the clear terminal. For active-LOW reset, $\overline{Q_4Q_2}$ is connected to all Flip-flops.

		Q2Q1			
		00	01	11	10
Q4Q3	00				
	01				
	11	X	X	X	X
	10		X	X	1

After pulses	Count			
	Q4	Q3	Q2	Q1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	0	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	0	1	0	1
10	0	0	0	0

Synchronous counters:

Asynchronous counters are serial counters. They are slow because each FF can change state only if all the preceding FFs have changed their state. If the clock frequency is very high, the asynchronous counter may skip some of the states. This problem is overcome in synchronous counters or parallel counters. Synchronous counters are counters in which all the flip-flops are triggered simultaneously by the clock pulses. Synchronous counters have a common clock pulse applied simultaneously to all flip-flops. □ A 2-Bit Synchronous Binary Counter



Design of synchronous counters:

For a systematic design of synchronous counters, the following procedure is used.

Step 1: State Diagram: draw the state diagram showing all the possible states. State diagram, which can also be called an n th transition diagram, is a graphical means of depicting the sequence of states through which the counter progresses.

Step 2: number of flip-flops: based on the description of the problem, determine the required number n of the flip-flops. The smallest value of n is such that the number of states $N \leq 2^n$ and the desired counting sequence.

Step 3: choice of flip-flops excitation table: select the type of flip-flop to be used and write the excitation table. An excitation table is a table that lists the present state (ps), the next state (ns), and required excitations.

Step4: minimal expressions for excitations: obtain the minimal expressions for the excitations of the FF using K-maps drawn for the excitation of the flip-flops in terms of the present states and inputs.

Step5: logic diagram: draw a logic diagram based on the minimal expressions

Design of a synchronous 3-bit up-down counter using JK flip-flops:

Step1: determine the number of flip-flops required. A 3-bit counter requires three FFs. It has 8 states (000,001,010,011,101,110,111) and all the states are valid. Hence no don't cares. For selecting up and down modes, a control or mode signal M is required. When the mode signal M=1 and counts down when M=0. The clock signal is applied to all the FFs simultaneously.

Step2: draw the state diagrams: the state diagram of the 3-bit up-down counter is drawn as

Step3: select the type of flip flop and draw the excitation table: JK flip-flops are selected and the excitation table of a 3-bit up-down counter using JK flip-flops is drawn as shown in fig.

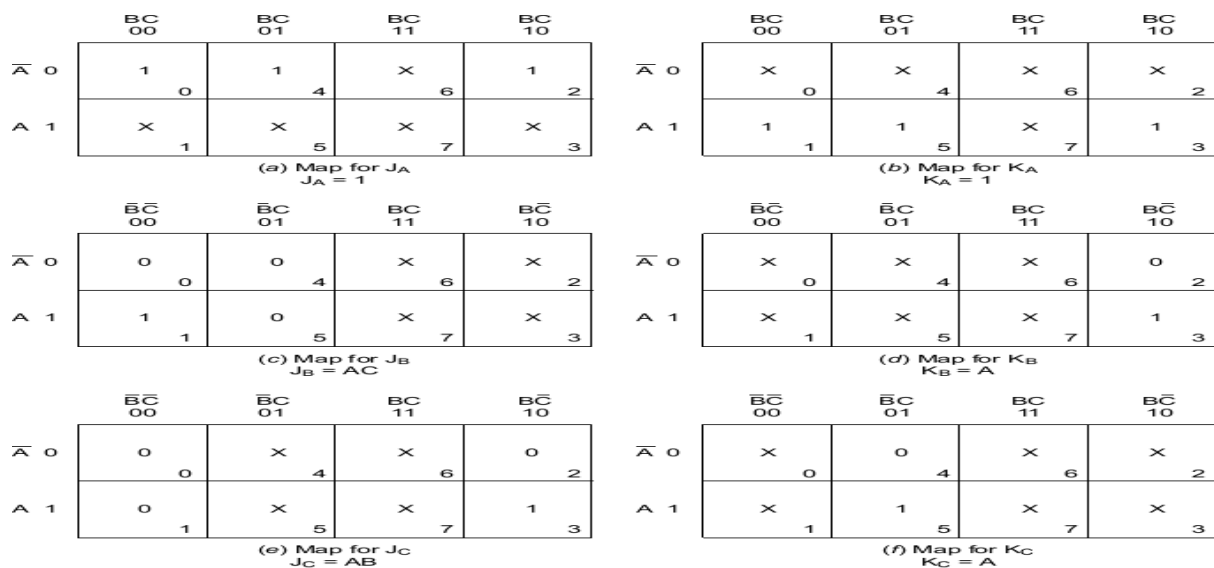
PS			mode	NS			required excitations					
Q3	Q2	Q1	M	Q3	Q2	Q1	J3	K3	J2	K2	J1	K1
0	0	0	0	1	1	1	1	x	1	x	1	x
0	0	0	1	0	0	1	0	x	0	x	1	x
0	0	1	0	0	0	0	0	x	0	x	x	1
0	0	1	1	0	1	0	0	x	1	x	x	1
0	1	0	0	0	0	1	0	x	x	1	1	x
0	1	0	1	0	1	1	0	x	x	0	1	x
0	1	1	0	0	1	0	0	x	x	0	x	1
0	1	1	1	1	0	0	1	x	x	1	x	1
1	0	0	0	0	1	1	x	1	1	x	1	x
1	0	0	1	1	0	1	x	0	0	x	1	x
1	0	1	0	1	0	0	x	0	0	x	x	1
1	0	1	1	1	1	0	x	0	1	x	x	1
1	1	0	0	1	0	1	x	0	x	1	1	x
1	1	0	1	1	1	1	x	0	x	0	1	x
1	1	1	0	1	1	0	x	0	x	0	x	1
1	1	1	1	0	0	0	x	1	x	1	x	1

Step4: obtain the minimal expressions: From the excitation table we can conclude that J1=1 and K1=1, because all the entries for J1 and K1 are either X or 1. The K-maps for J3, K3, J2 and K2 based on the excitation table and the minimal expression obtained from them are shown in fig.

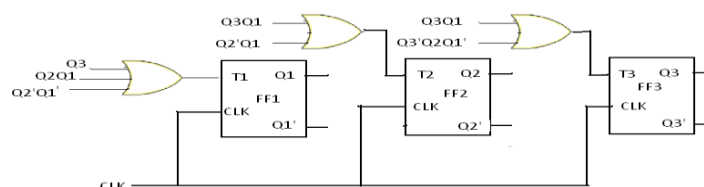
Step3: type of flip-flop and the excitation table: T flip-flops are selected and the excitation table of the mod-6 gray code counter using T-flip-flops is written as shown in fig.

PS			NS			required excitations		
Q3	Q2	Q1	Q3	Q2	Q1	T3	T2	T1
0	0	0	0	0	1	0	0	1
0	0	1	0	1	1	0	1	0
0	1	1	0	1	0	0	0	1
0	1	0	1	1	0	1	0	0
1	1	0	1	1	1	0	0	1
1	1	1	0	0	0	1	1	1

Step4: The minimal expressions: the K-maps for excitations of FFs T3,T2,and T1 in terms of outputs of FFs Q3,Q2, and Q1, their minimization and the minimal expressions for excitations obtained from them are shown if fig



Step5: the logic diagram: the logic diagram based on those minimal expressions is drawn as shown in fig.



Design of a synchronous BCD Up-Down counter using FFs:

Step1: the number of flip-flops: a BCD counter is a mod-10 counter has 10 states (0000 through 1001) and so it requires $n=4\text{FFs}$ ($N \leq 2^n$, i.e., $10 \leq 2^4$). 4 FFs can have 16 states. So out of 16 states, six states (1010 through 1111) are invalid. For selecting up and down mode, a control or mode signal M is required. , it counts up when $M=1$ and counts down when $M=0$. The clock signal is applied to all FFs.

Step2: the state diagram: The state diagram of the mod-10 up-down counter is drawn as shown in fig.

Step3: types of flip-flops and excitation table: T flip-flops are selected and the excitation table of the modulo-10 up down counter using T flip-flops is drawn as shown in fig.

The remaining minterms are don't cares ($\sum d(20,21,22,23,24,25,26,27,28,29,30,31)$) from the excitation table we can see that $T_1=1$ and the expression for T_4, T_3, T_2 are as follows.

$$T_4 = \sum m(0,15,16,19) + d(20,21,22,23,24,25,26,27,28,29,30,31)$$

$$T_3 = \sum m(7,15,16,8) + d(20,21,22,23,24,25,26,27,28,29,30,31)$$

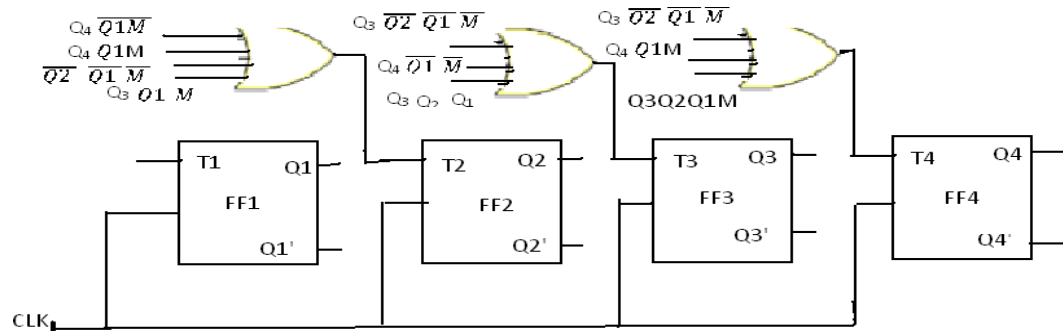
$$T_2 = \sum m(3,4,7,8,11,12,15,16) + d(20,21,22,23,24,25,26,27,28,29,30,31)$$

PS					mode	NS				required excitations			
Q4	Q3	Q2	Q1	M		Q4	Q3	Q2	Q1				
0	0	0	0	0		1	0	0	1	1	0	0	1
0	0	0	0	1		0	0	0	1	0	0	0	1
0	0	0	1	0		0	0	0	0	0	0	0	1
0	0	0	1	1		0	0	1	0	0	0	1	1
0	0	1	0	0		0	0	0	1	0	0	1	1
0	0	1	0	1		0	0	1	1	0	0	0	1
0	0	1	1	0		0	0	1	0	0	0	0	1
0	0	1	1	1		0	1	0	0	0	1	1	1
0	1	0	0	0		0	0	1	1	0	1	1	1
0	1	0	0	1		0	1	0	1	0	0	0	1
0	1	0	1	0		0	1	0	0	0	0	0	1
0	1	0	1	1		0	1	1	0	0	0	1	1
0	1	1	0	0		0	1	0	1	0	0	1	1
0	1	1	0	1		0	1	1	1	0	0	0	1
0	1	1	1	0		0	1	1	0	0	0	0	1
0	1	1	1	1		1	0	0	0	1	1	1	1
1	0	0	0	0		0	1	1	1	1	1	1	1
1	0	0	0	1		1	0	0	1	0	0	0	1
1	0	0	1	0		1	0	0	0	0	0	0	1
1	0	0	1	1		0	0	0	0	1	0	0	1

Step4: The minimal expression: since there are 4 state variables and a mode signal, we require 5 variable kmaps. 20 conditions of $Q_4Q_3Q_2Q_1M$ are valid and the remaining 12 combinations are don't cares. Minimizing K-maps for T2 we get

$$T_2 = Q_4Q_1'M + Q_4'Q_1M + Q_2Q_1'M' + Q_3Q_1'M'$$

Step5: the logic diagram: the logic diagram based on the above equation is shown in fig.



Shift register counters:

One of the applications of shift register is that they can be arranged to form several types of counters. The most widely used shift register counter is ring counter as well as the twisted ring counter.

Ring counter: this is the simplest shift register counter. The basic ring counter using D flip-flops is shown in fig. the realization of this counter using JK FFs. The Q output of each stage is connected to the D flip-flop connected back to the ring counter.

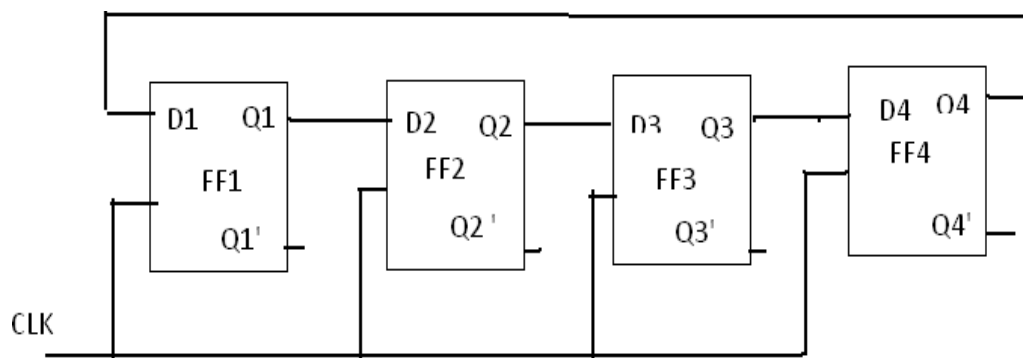
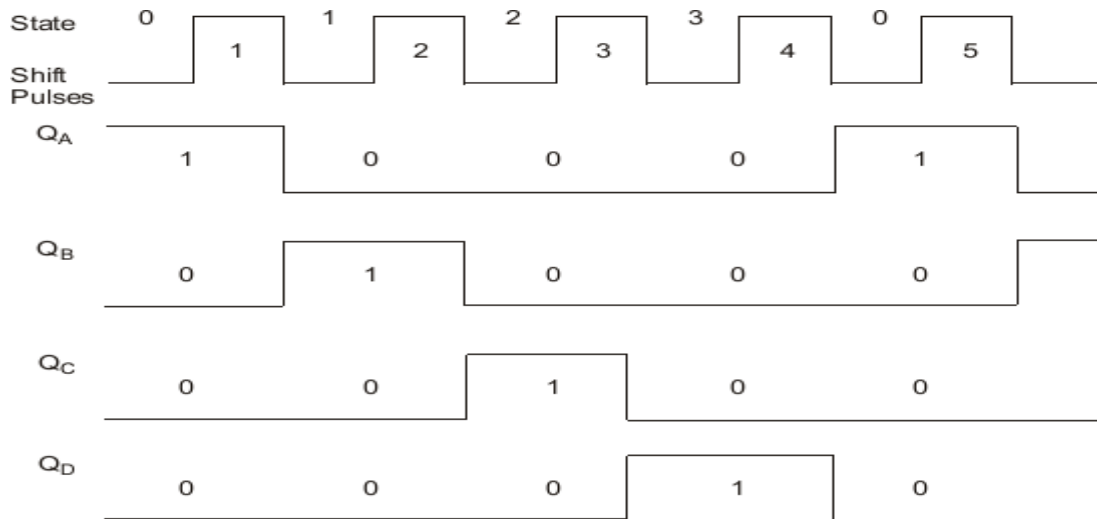


FIGURE: logic diagram of 4-bit ring counter using D flip-flops

Only a single 1 is in the register and is made to circulate around the register as long as clock pulses are applied. Initially the first FF is present to a 1. So, the initial state is 1000, i.e., $Q_1=1, Q_2=0, Q_3=0, Q_4=0$. After each clock pulse, the contents of the register are shifted to the right by one bit and Q_4 is shifted back to Q_1 . The sequence repeats after four clock pulses. The number

of distinct states in the ring counter, i.e., the mod of the ring counter is equal to number of FFs used in the counter. An n-bit ring counter can count only n bits, whereas n-bit ripple counter can count 2^n bits. So, the ring counter is uneconomical compared to a ripple counter but has advantage of requiring no decoder, since we can read the count by simply noting which FF is set. Since it is entirely a synchronous operation and requires no gates external FFs, it has the further advantage of being very fast.

Timing diagram:



Twisted Ring counter (Johnson counter):

This counter is obtained from a serial-in, serial-out shift register by providing feedback from the inverted output of the last FF to the D input of the first FF. the Q output of each is connected to the D input of the next stage, but the Q' output of the last stage is connected to the D input of the first stage, therefore, the name twisted ring counter. This feedback arrangement produces a unique sequence of states.

The logic diagram of a 4-bit Johnson counter using D FF is shown in fig. the realization of the same using J-K FFs is shown in fig.. The state diagram and the sequence table are shown in figure. The timing diagram of a Johnson counter is shown in figure.

Let initially all the FFs be reset, i.e., the state of the counter be 0000. After each clock pulse, the level of Q1 is shifted to Q2, the level of Q2 to Q3, Q3 to Q4 and the level of Q4' to Q1 and the sequences given in fig.

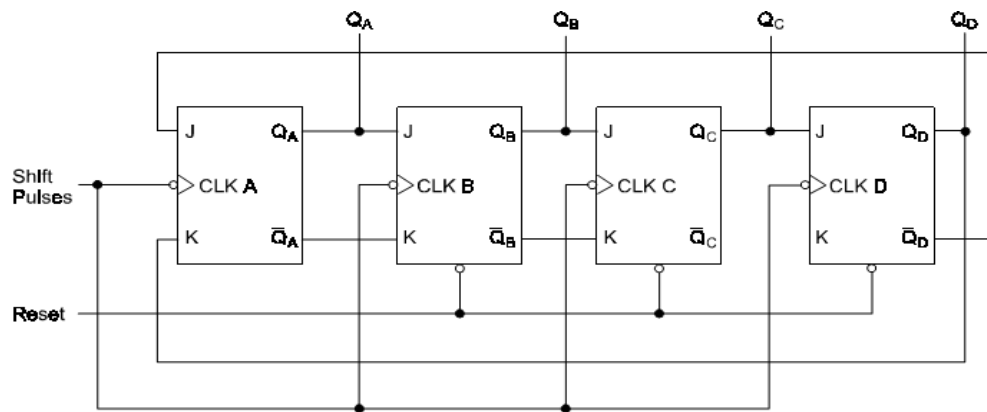


Figure: Johnson counter with JK flip-flops

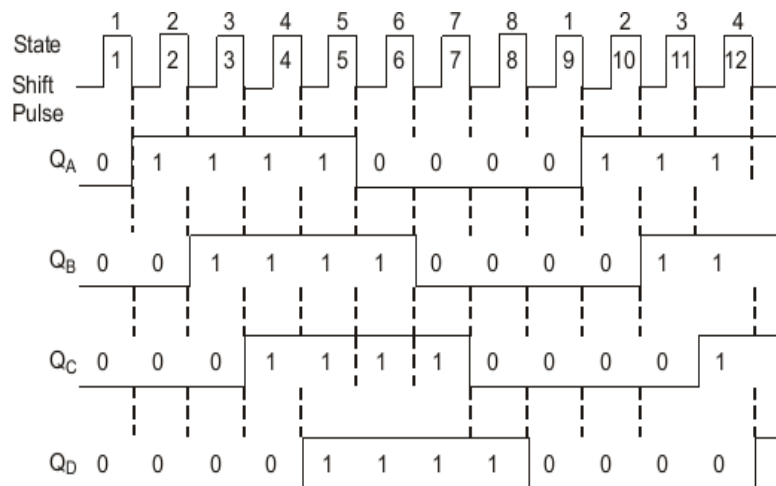


Figure: timing diagram