Half adder

- A half-adder is a combinational circuit with two binary inputs (augund and addend bits) and two binary outputs (sum and carry bits).
- It adds the two inputs (single bit words A and B) and produces the sum (S) and the carry (C) bits.
- The truth table of a half-adder are shown below:

Inputs		Outputs	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

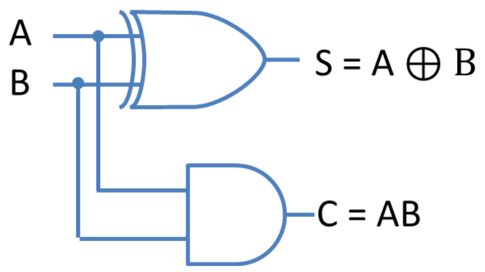
- The Sum (S) is the X-OR of A and B (It represent the LSB of the sum). Therefore,

$$S = AB' + BA' = A \oplus B$$

- 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, therefore, be realized by using one X-OR gate and one AND gate as shown in figure below.



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.
- When we want 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.
- 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 truth table of a full-adder are shown in figure below.

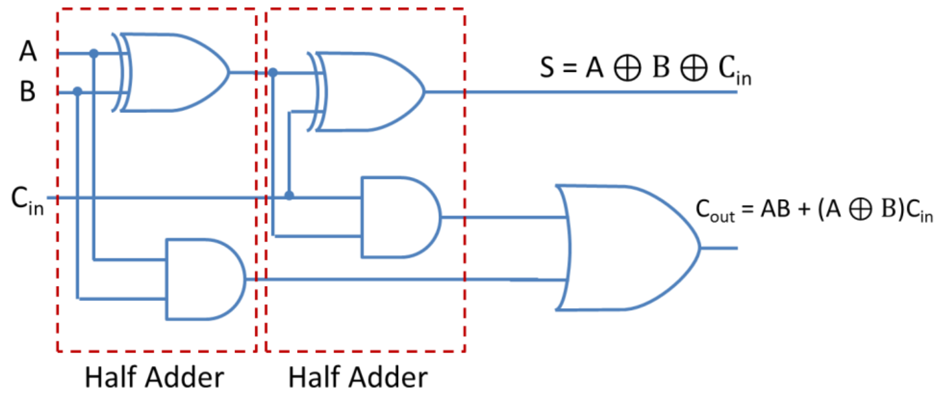
Inputs			Outputs	
A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

- The eight rows under the input variables designate all possible combinations of 1s and 0s that these variables may have.
- 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.
- 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

$$\begin{aligned} S &= A'B'C_{in} + A'BC_{in}' + AB'C_{in}' + ABC_{in} \\ &= (AB' + A'B)C_{in}' + (AB + A'B')C_{in} \\ &= (A \oplus B)C_{in}' + (A \oplus B)'C_{in} \\ &= A \oplus B \oplus C_{in} \end{aligned}$$

$$\begin{aligned} C_{out} &= A'BC_{in} + AB'C_{in} + ABC_{in}' + ABC_{in} \\ &= AB + (A \oplus B)C_{in} \end{aligned}$$

- The sum term of the full-adder is the X-OR of A, B and C_{in}, i.e., the sum bit is 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 shown in figure below.


A
B
C_{in}
$S = A \oplus B \oplus C_{in}$
$C_{out} = AB + (A \oplus B)C_{in}$
Half Adder
Half Adder

Half-Subtractor

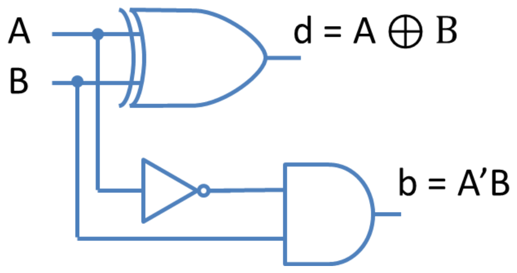
- A half-subtractor is a combinational circuit that subtracts one bit from the other and produces the difference.
- It also has an output to specify if a 1 has been borrowed.
- It is used to subtract the LSB of the subtrahend from the LSB of the minuend when one binary number is subtracted from the other.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

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

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

$$d = AB' + BA' = A \oplus B \text{ and } d = A'B$$

- That is, the difference bit is obtained by X-ORing the two inputs, and the borrow bit is obtained by ANDing the complement of the minuend with the subtrahend.
- Figure below shows logic diagrams of a half-subtractor.



Full Subtractor

- The half-subtractor can be used 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 column.
- So a full-subtractor is a combinational circuit with three inputs (A, B, bi) and two outputs d and b.
- The 1s and 0s for the output variables are determined from the subtraction of A - B - b_i.
- The truth table of a full-subtractor are shown in figure.

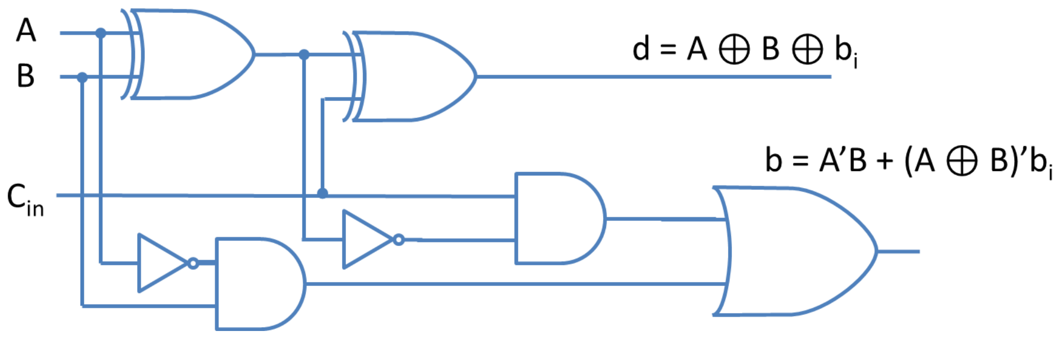
Inputs			Outputs	
A	**B**	**b_i**	**d**	**b**
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

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

$$\begin{aligned} d &= A'B'b_i + A'Bb_i' + AB'b_i' + ABb_i \\ &= (AB' + A'B)b_i' + (AB + A'B')b_i \\ &= (A \oplus B)b_i' + (A \oplus B)'b_i \\ &= A \oplus B \oplus b_i \end{aligned}$$

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

- A full-subtractor can, therefore, be realized using X-OR gates as shown below.



Design BCD to Excess-3 code converter circuit.

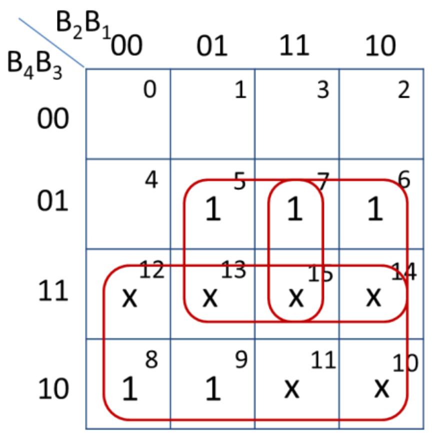
- BCD means 8421 BCD.
- The 4-bit input BCD code (B4 B3 B2 B1) and the corresponding output XS-3 code (X4 X3 X2 X1) numbers are shown in the conversion table in figure.

8421 code				XS-3 code			
B_4	B_3	B_2	B_1	X_4	X_3	X_2	X_1
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

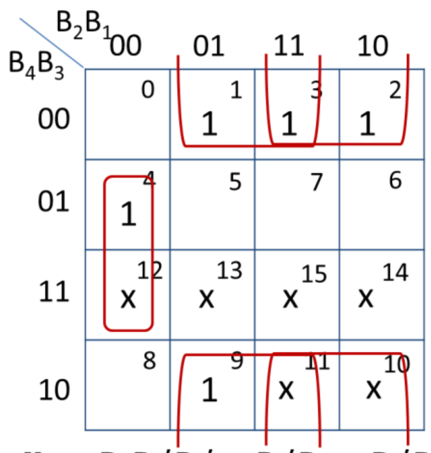
- The input combinations 1010, 1011, 1100, 1101, 1110, and 1111 are invalid in BCD. So they are treated as don't cares.
- From the above truth table, function can be realized as follows:

$$X_4 = \Sigma\, m(5, 6, 7, 8, 9) + d(10, 11, 12, 13, 14, 15)$$
$$X_3 = \Sigma\, m(1, 2, 3, 4, 9) + d(10, 11, 12, 13, 14, 15)$$
$$X_2 = \Sigma\, m(0, 3, 4, 7, 8) + d(10, 11, 12, 13, 14, 15)$$
$$X_1 = \Sigma\, m(0, 2, 4, 6, 8) + d(10, 11, 12, 13, 14, 15)$$

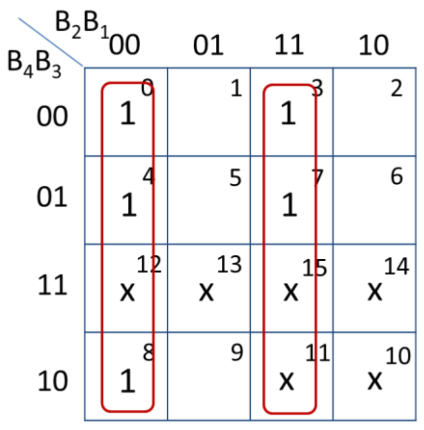
- Drawing K-maps for the outputs X_4, X_3, X_2, and X_1 in terms of the inputs B_4, B_3, B_2, and B_1 and simplifying them, as shown.



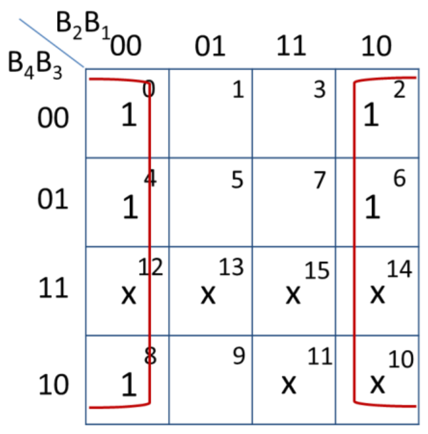
$X_4 = B_4 + B_3B_2 + B_3B_1$



$X_3 = B_3B_2'B_1' + B_3'B_1 + B_3'B_2$



$X_2 = B_2'B_1' + B_2B_1$



$X_2 = B_1'$

- The minimal expressions are

$X_4 = B_4 + B_3B_2 + B_3B_1$

$X_3 = B_3B_2'B_1' + B_3'B_1 + B_3'B_2$

$X_2 = B_2'B_1' + B_2B_1$

$X_1 = B_1'$

Design 4 bit binary to gray code converter

- The input to the 4-bit binary-to-Gray code converter circuit is a 4-bit binary and the output is a 4-bit Gray code.
- There are 16 possible combinations of 4-bit binary input and all of them are valid. Hence no don't cares.
- The 4-bit binary and the corresponding Gray code are shown in the conversion table below.

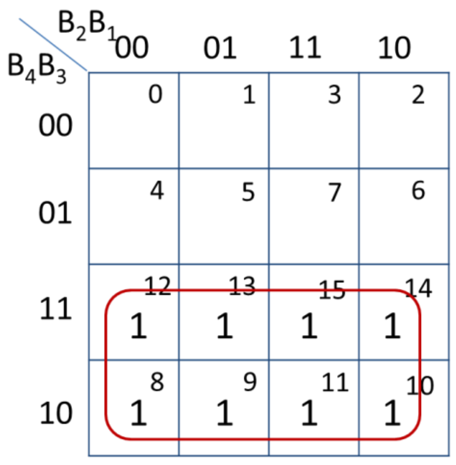
4-bit binary				4-bit Gray			
B_4	B_3	B_2	B_1	G_4	G_3	G_2	G_1
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1

0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

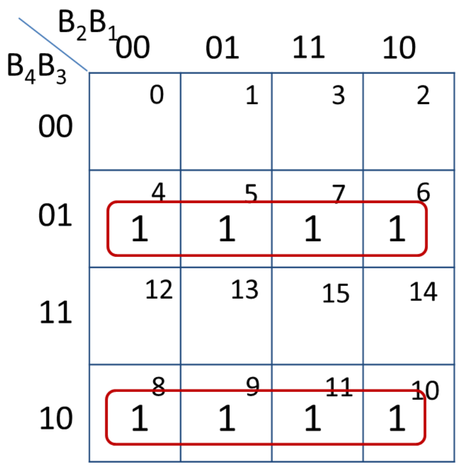
- From the conversion table, we observe that the expressions for the outputs G4, G3, G2, and G1 are as follows:

G4	= Σ m(8, 9, 10, 11, 12, 13, 14, 15)
G3	= Σ m(4, 5, 6, 7, 8, 9, 10, 11)
G2	= Σ m(2, 3, 4, 5, 10, 11, 12, 13)
G1	= Σ m(1, 2, 5, 6, 9, 10, 13, 14)

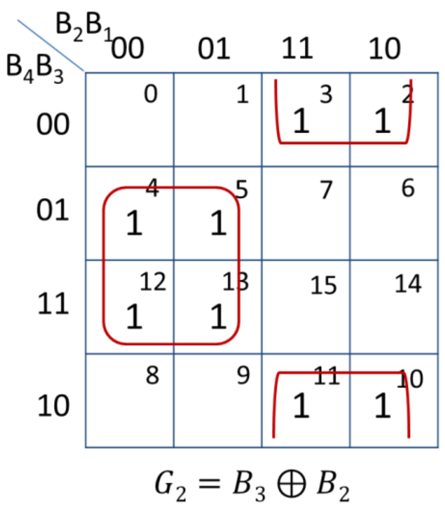
- The K-maps for G4, G3, G2, and G1 and their minimization are shown in figure below.

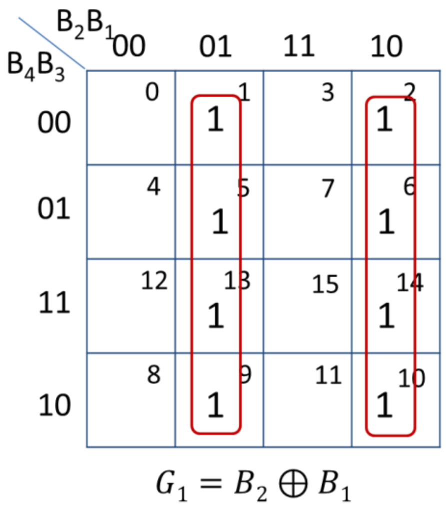


$G_4 = B_4$



$G_3 = B_4 \oplus B_3$

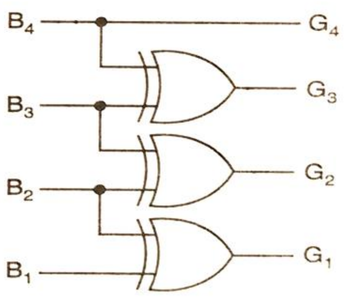




- The minimal expressions for the outputs obtained from the K-map are:

$$G_4 = B4$$
$$G3 = B4'B3 + B4B3' = B4 \oplus B3$$
$$G2 = B3'B2 + B3B2' = B3 \oplus B2$$
$$G1 = B2'B1 + B2B1' = B2 \oplus B1$$

- Logic diagram for the above is as follows.



Design a circuit for 2-bit magnitude comparator.

- The logic for a 2-bit magnitude comparator: Let the two 2-bit numbers be $A = A_1A_0$ and $B = B_1B_0$.
  1. If $A_1 = 1$ and $B_1 = 0$, then A > B or
  2. If A_1 and B_1 coincide and $A_0 = 1$ and $B_0 = 0$, then A > B. So the logic expression for A > B is

$$A > B : G = A_1B_1' + (A1 \odot B1)\, A_0B_0'$$

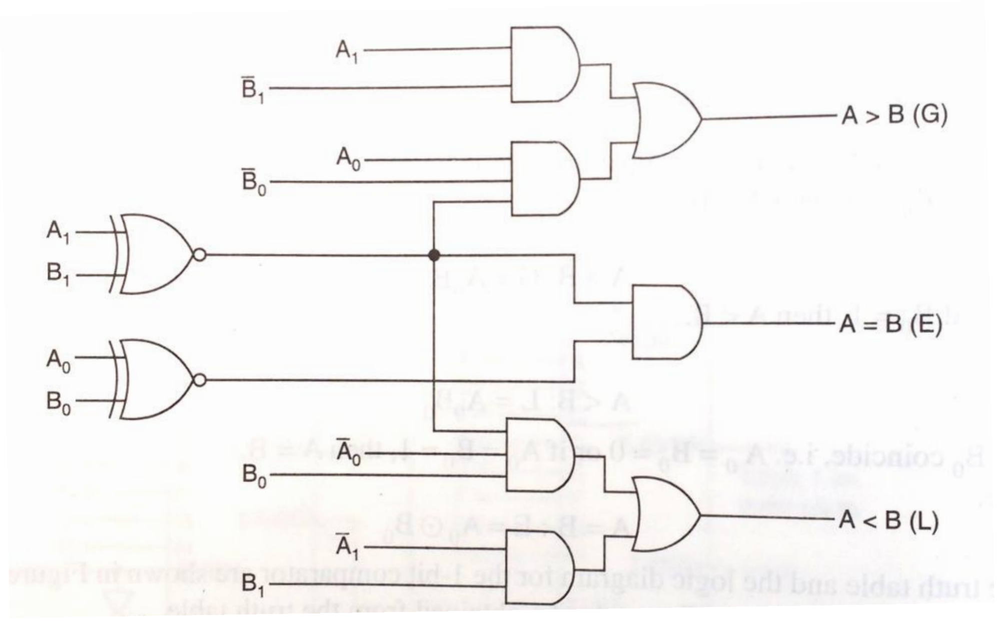
  1. If $A_1 = 0$ and $B_1 = 1$, then A < B or
  2. If A_1 and B_1 coincide and $A_0 = 0$ and $B_0 = 1$, then A < B. So the expression for A < B is

$$A < B : L = A_1'B_1 + (A_1 \odot B_1)\, A_0'B_0$$

  If A_1 and B_1 coincide and if A_0 and B_0 coincide then A = B. So the expression for A = B is

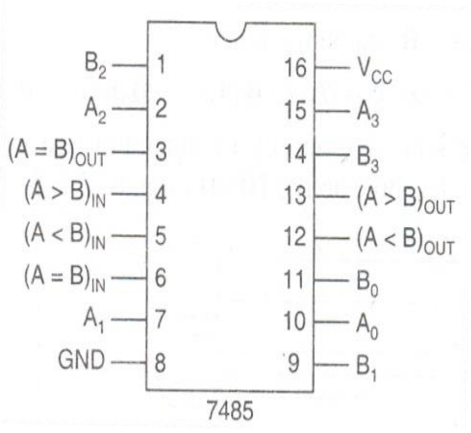
$$A = B : E = (A_1 \odot B_1)(A_0 \odot B_0)$$

- The logic diagram for a 2-bit comparator is as shown below:

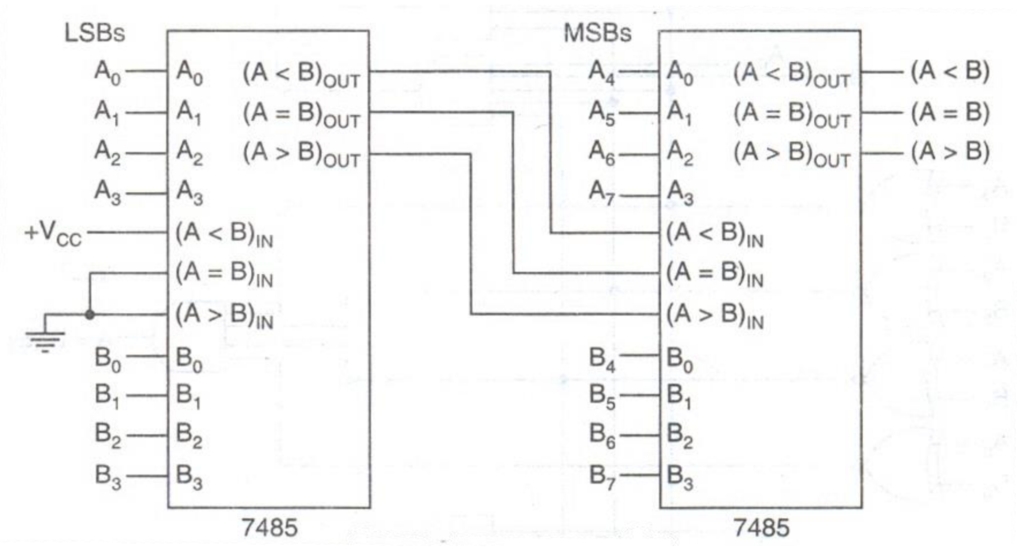


Draw & explain in brief pin diagram of 7485 four-bit magnitude comparator.

- Figure below shows the pin diagram of IC 7485, a 4-bit comparator.



- Pins labelled $(A < B)_{IN}$, $(A = B)_{IN}$, and $(A > B)_{IN}$ are used for cascading.
- Figure shows how two 4-bit comparator are cascaded to perform 8-bit comparisons.



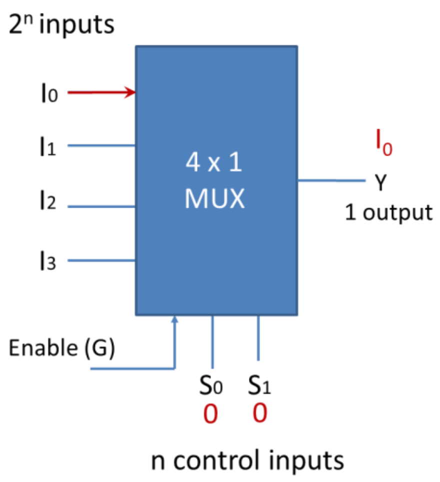
- The $(A < B)_{OUT}$, $(A = B)_{OUT}$ and $(A > B)_{OUT}$ outputs from the lower order comparator used for the least significant 4 bits, are connected to the $(A < B)_{IN}$, $(A = B)_{IN}$, and $(A > B)_{IN}$ inputs of the higher-order comparator.
- Note that, $(A < B)_{IN}$ input of the lower order comparator is connected to V_{CC}, and $(A = B)_{IN}$ and $(A > B)_{IN}$ inputs of the lower order comparator are connected to ground.

What is multiplexer? With logic circuit and function table explain the working of 4 to 1 line multiplexer.

- A multiplexer (MUX) is a device that allows digital information from several sources to be routed onto a single line for transmission over that line to a common destination.
- Consider an integer 'm', which is constrained by the following relation:

  $m = 2^n$, where m and n are both integers.
- A **m-to-1** Multiplexer has
  - m Inputs: $I_0, I_1, I_2, \ldots\ldots\ldots\ldots\ldots I_{(m-1)}$
  - One Output: Y
  - n Control inputs: $S_0, S_1, S_2, \ldots\ldots S_{(n-1)}$
  - One (or more) Enable input(s)

  such that Y may be equal to one of the inputs, depending upon the control inputs.
- The block diagram of 4 x 1 multiplexer is as follows.



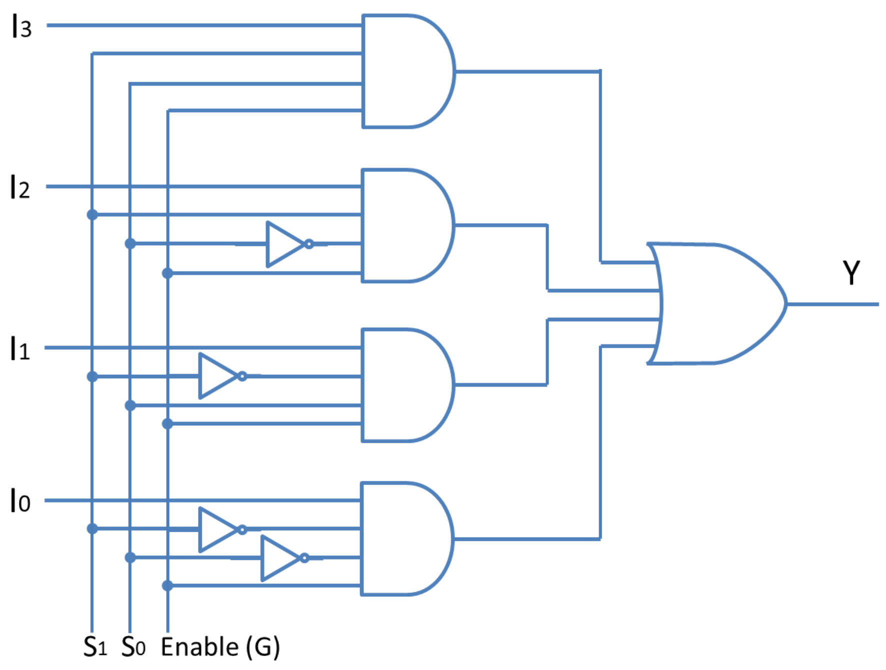
- The function table for the 4 x 1 multiplexer can be stated as below.

Select Inputs		Output
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

- The following logic function describes the above function table.

$$\boxed{Y = S_1'S_0'I_0 + S_1'S_0I_1 + S_1S_0'I_2 + S_1S_0I_3}$$

- The following figure describes the logic circuit for 4 x 1 multiplexer.



- Applications of Multiplexer is as follows:
    1. Logic function generation
    2. Data selection
    3. Data routing
    4. Operation sequencing
    5. Parallel-to-serial conversion
    6. Waveform generation

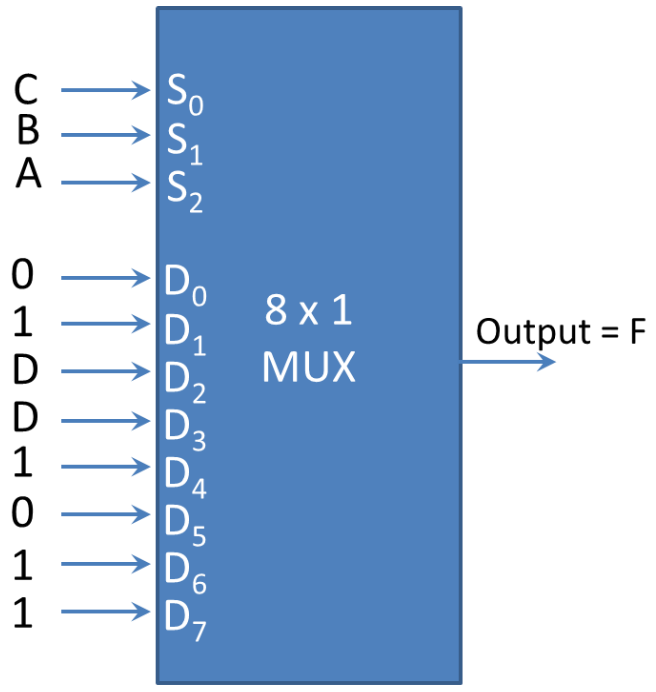
Implement following Boolean function using 8 : 1 multiplexer.

F(A,B,C,D) = Σ(2,3,5,7,8,9,12,13,14,15)

- The truth table for the above function is as follows:

S_2	S_1	S_0	D	F	
A	B	C			
0	0	0	0	0	F = 0
0	0	0	1	0	
0	0	1	0	1	F = 1
0	0	1	1	1	
0	1	0	0	0	F = D
0	1	0	1	1	
0	1	1	0	0	F = D
0	1	1	1	1	
1	0	0	0	1	F = 1
1	0	0	1	1	
1	0	1	0	0	F = 0
1	0	1	1	0	
1	1	0	0	1	F = 1
1	1	0	1	1	
1	1	1	0	1	F = 1
1	1	1	1	1	

- Based on the above truth table, the logic function can be implemented using 8 x 1 Multiplexer as follows:



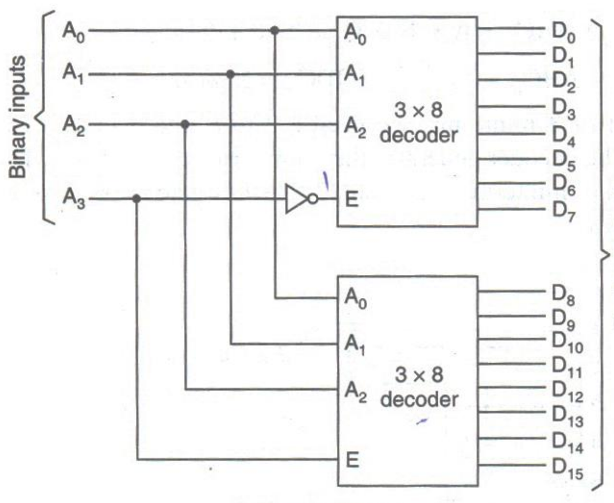
Exercise

Implement following Boolean function using 8 : 1 multiplexer.

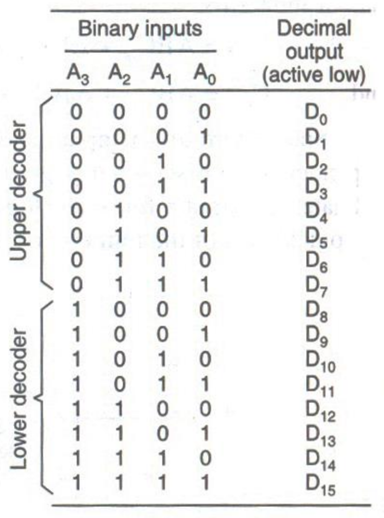
1) F(A, B, C) = ∑m (1, 3, 5, 6)
2) F(A, B, C) = ∑m (1,2,4,7)
3) F(A, B, C, D) = ∑m (0, 1, 3, 5, 7, 11, 13, 14, 15)
4) F(A, B, C, D) = ∑m (0,1,2,3,5,8,9,11,14)
5) F(A, B, C, D) = ∑m (0,1,3,4,8,9,15)

Design 4 x 16 decoder using two 3 x 8 decoder.

- Decoders with enable inputs can be connected together to form a larger decoder circuit.
- Figure shows the arrangement for using two 3-to-8 decoders, to obtain a 4-to-16 decoder.
- The most significant input bit A_3 is connected through an inverter to E on the upper decoder (for D_0 through D_7) and directly to E on the lower decoder (for D_8 through D_{15}).
- Thus, when A_3 is LOW, the upper decoder is enabled and the lower decoder is disabled.
- The bottom decoder outputs all 0s, and top 8 outputs generate minterms.
- When A_3 is HIGH, the lower decoder is enabled and the upper decoder is disabled. The bottom decoder outputs generate minterms 1000 to 1111 while the outputs of the top decoder are all 0s.



(a) Logic diagram



	Binary inputs				Decimal output (active low)
	A_3	A_2	A_1	A_0	
Upper decoder	0	0	0	0	D_0
	0	0	0	1	D_1
	0	0	1	0	D_2
	0	0	1	1	D_3
	0	1	0	0	D_4
	0	1	0	1	D_5
	0	1	1	0	D_6
	0	1	1	1	D_7
Lower decoder	1	0	0	0	D_8
	1	0	0	1	D_9
	1	0	1	0	D_{10}
	1	0	1	1	D_{11}
	1	1	0	0	D_{12}
	1	1	0	1	D_{13}
	1	1	1	0	D_{14}
	1	1	1	1	D_{15}

(b) Function table

Explain full adder and design a full adder circuit using 3 to 8 decoder and two OR gates.

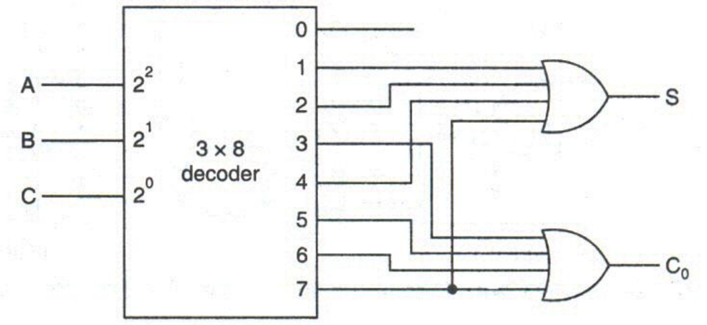
- A full-adder is a combinational circuit that adds two bits and a carry and outputs a sum bit and a carry bit.
- When we want to add two binary numbers, each having two or more bits, the LSBs can be added by using a half-adder.
- The carry resulted from the addition of the LSBs is carried over to the next significant column and added to the two bits in that column.
- 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 truth table of a full-adder are shown in figure below.

Inputs			Outputs	
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

- The eight rows under the input variables designate all possible combinations of 1s and 0s that these variables may have.
- 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.
- 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
- The function S and C_{out} can be represented in form of minterms as,

  $S = \Sigma\, m(1, 2, 4, 7)$

  $C_{out} = \Sigma\, m(3, 5, 6, 7)$
- The full adder can be implemented using decoder is as follows.



Implement Full Subtractor Circuit with the help of Decoder and logic gates.

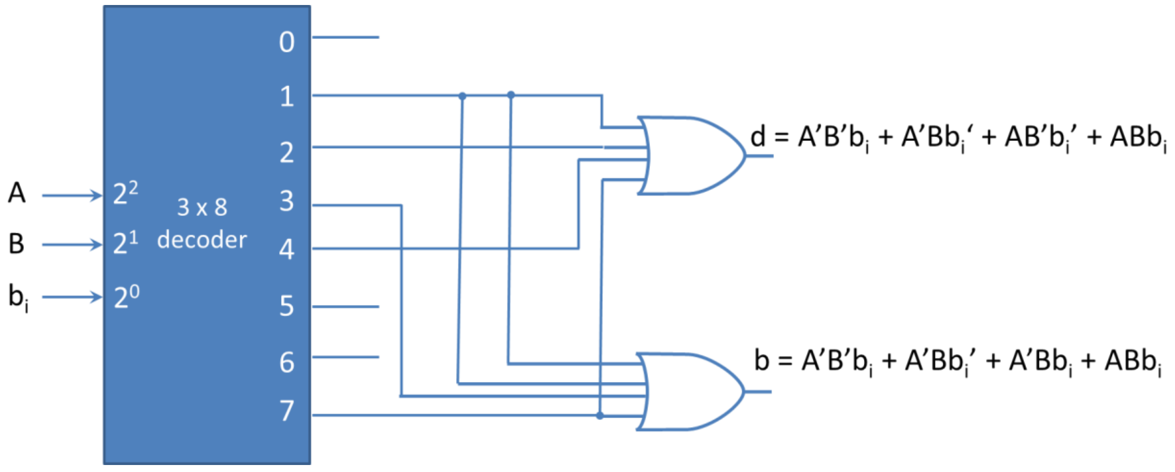
- The half-subtractor can be used only for LSB subtraction.
- If there is a borrow during the subtraction of the LSBs, it affects the subtraction in the next higher column; the subtrahend bit is subtracted from the minuend bit, considering the borrow from that column used for the subtraction in the preceding 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 column.
- So a full-subtractor is a combinational circuit with three inputs (A, B, bi) and two outputs d and b.
- The 1s and 0s for the output variables are determined from the subtraction of A - B - b_i.
- The truth table of a full-subtractor are shown in figure.

Inputs			Outputs	
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

- From the truth table, a circuit that will produce the correct difference and borrow bits in response to every possible combination of A, B, and b_i is described by
- The function d and b can be represented in form of minterms as,

  d = Σ m(1, 2, 4, 7)

  b = Σ m(1, 2, 3, 7)
- The full subtractor can be implemented using decoder is as follows.


A
B
b_i
2^2
2^1
2^0
3 x 8 decoder
0
1
2
3
4
5
6
7
d = A'B'b_i + A'Bb_i' + AB'b_i' + ABb_i
b = A'B'b_i + A'Bb_i' + A'Bb_i + ABb_i

Comparison between Combinational Circuit & Sequential Circuit

Combinational circuits	Sequential circuits
1. In combinational circuits, the output variables at any instant of time are dependent only on the present input variables.	1. In sequential circuits, the output variables at any instant of time are dependent not only on the present input variables, but also on the present state, i.e. on the past history of the system.
2. Memory unit is not required in combinational circuits.	2. Memory unit is required to store the past history of the input variables in sequential circuits.
3. Combinational circuits are faster because the delay between the input and the output is due to propagation delay of gates only.	3. Sequential circuits are slower than combinational circuits.
4. Combinational circuits are easy to design.	4. Sequential circuits are comparatively harder to design.

Flip-flop

- A flip-flop, known formally as bistable multivibrator, has two stable states.
- It can remain in either of the states indefinitely.
- Its state can be changed by applying the proper triggering signal.

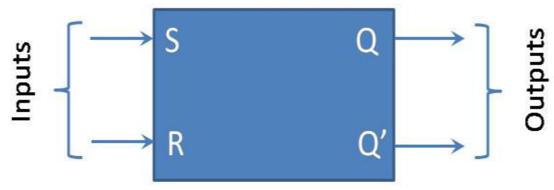
Latch

- Latch is used for certain flip-flop which are *non-clocked.*
- These flip-flops 'latch on' to a 1 or a 0 immediately upon receiving the input pulse called SET or RESET.
- The latch is sequential device that checks all its inputs continuously and changes its outputs accordingly at any time independent of a clock signal.

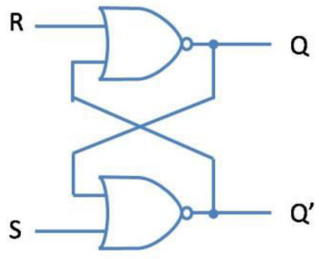
The S-R Latch

- The simplest type of flip-flop is called an S-R latch.
- It has two outputs labelled Q and Q' and two inputs labelled S and R. The state of the latch corresponds to the level of Q (HIGH or LOW, 1 or 0) and Q' is the complement of that state.
- It can be constructed using either two cross-coupled NAND gates or two-cross coupled NOR gates.
- Using two NOR gates, an active-HIGH S-R latch can be constructed and using two NAND gates an active-LOW S-R latch can be constructed.
- The name of the latch, S-R or SET-RESET, is derived from the names of its inputs.

NOR Gate S-R latch (Active High)



Logic Symbol

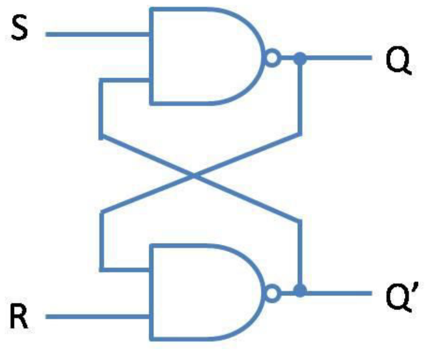


Logic diagram

S	R	Q_n	Q_{n+1}	State
0	0	0	0	No Change (NC)
0	0	1	1	
0	1	0	0	Reset
0	1	1	0	
1	0	0	1	Set
1	0	1	1	
1	1	0	X	Indeterminate (Invalid)
1	1	1	X	

- When the SET input is made HIGH, Q becomes 1.
- When the RESET input is made HIGH, Q becomes 0.
- If both the inputs S and R made LOW, there is no change in the state of the latch.
- If both the inputs are made HIGH, the output is unpredictable i.e. invalid.
- Figure shows the logic diagram of an active HIGH S-R latch composed of two cross-coupled NOR gates.
- Note that the output of each gate is connected to one of the inputs of the other gate.
- The latch works as per the truth table of figure, where Q_n represents the state of the flip-flop before applying inputs and Q_{n+1} represents the state of the flip-flop after applying inputs.

NAND Gate S-R latch (Active Low)

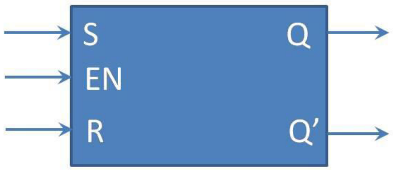


Logic diagram

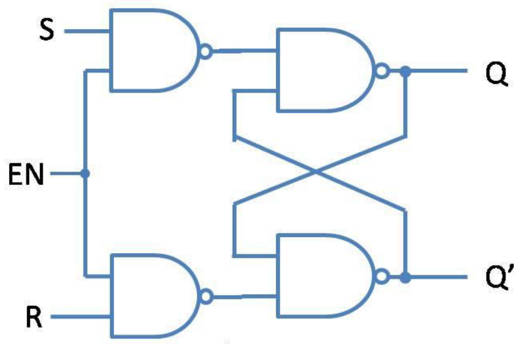
S	R	Q_n	Q_{n+1}	State
0	0	0	X	Indeterminate (Invalid)
0	0	1	X	
0	1	0	1	Set
0	1	1	1	
1	0	0	0	Reset
1	0	1	0	
1	1	0	0	No Change (NC)
1	1	1	1	

- The NAND gate is equivalent to an active LOW OR gate, am active LOW S-R latch using OR gates may also be represented.
- The operation of this latch is the reverse of the operation of the NOR gate latch.
- If the 0s are replaced by 1s and 1s by 0s, we get the same truth table as that of NOR gate latch.
- The SET and RESET inputs are normally resting in the HIGH state and one of them will be pulsed LOW, whenever we want to change the latch outputs.

Gated S-R Latch (Clocked Flip-flop)



Logic Symbol

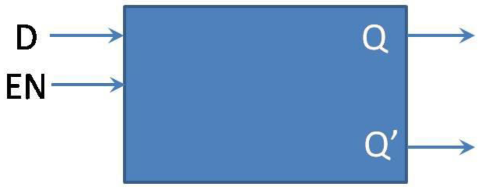


Logic diagram

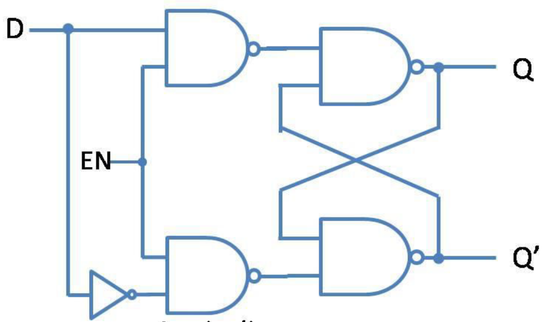
En	S	R	Q_n	Q_{n+1}	State
1	0	0	0	0	No Change (NC)
1	0	0	1	1	
1	0	1	0	0	Reset
1	0	1	1	0	
1	1	0	0	1	Set
1	1	0	1	1	
1	1	1	0	X	Indeterminate (Invalid)
1	1	1	1	X	
0	X	X	0	0	No Change (NC)
0	X	X	1	1	

- A gated S-R larch requires an ENABLE (EN) input.
- Its S and R inputs will control the state of the flip-flop only when the EN is HIGH.
- When EN is LOW, the inputs become ineffective and no change of state can take place.
- The EN input may be a clock. So, a gated S-R latch is also called a *clocked S-R latch*.
- Since this type of flip-flop responds to the changes in inputs only as long as the clock is HIGH, these types of flip-flops are also called *level triggered flip-flops*.

Gated D-Latch



Logic Symbol



Logic diagram

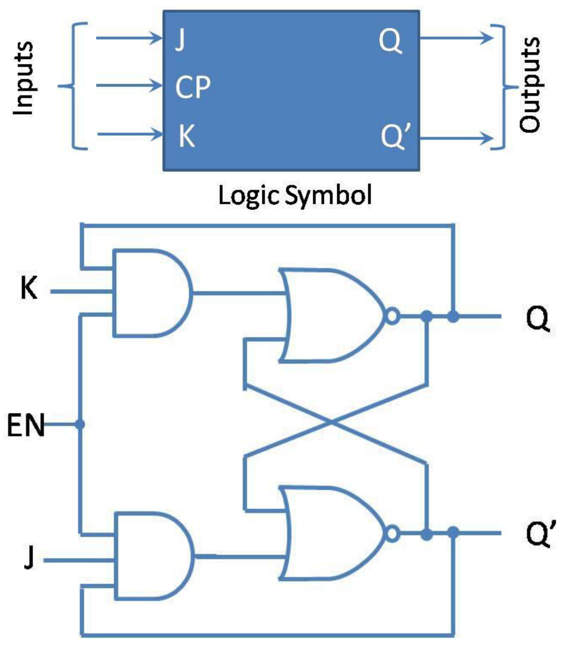
En	D	Q_n	Q_{n+1}	State
1	0	0	0	Reset
1	0	1	0	
1	1	0	1	Set
1	1	1	1	
0	X	0	0	No Change (NC)
0	X	1	1	

- It differs from the S-R latch in that it has only one input in addition to EN.
- When D=1, we have S=1 and R=0, causing the latch to SET when ENABLED.
- When D=0, we have S=0 and R=1, causing the latch to RESET when ENABLED.
- When EN is LOW, the latch is ineffective, and any change in the value of D input does not affect the output

at all.

- When EN is HIGH, a LOW D input makes Q LOW, i.e. resets the flip-flop and a HIGH D input makes Q HIGH, i.e. sets the flip-flop.
- In other words, we can say that the output Q follows the D input when EN is HIGH.

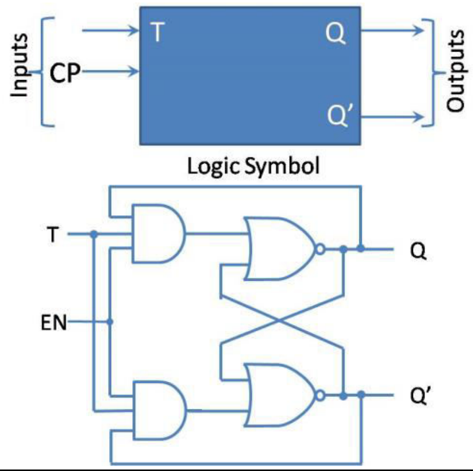
J-K Flip-Flop



En	J	K	Q_n	Q_{n+1}	State
1	0	0	0	0	No Change (NC)
1	0	0	1	1	
1	0	1	0	0	Reset
1	0	1	1	0	
1	1	0	0	1	Set
1	1	0	1	1	
1	1	1	0	1	Toggle
1	1	1	1	0	
0	X	X	0	0	No Change (NC)
0	X	X	1	1	

- The J-K flip-flop is very versatile and also the most widely used.
- The functioning of the J-K flip-flop is identical to that of the S-R flip-flop, except that it has no invalid state like that of S-R flip-flop.
- When J=0 and K=0, no change of state place even if a clock pulse is applied.
- When J=0 and K=1, the flip-flop resets at the HIGH level of the clock pulse.
- When J=1 and K=0, the flip-flop sets at the HIGH level of the clock pulse.
- When J=1 and K=1, the flip-flop toggles, i.e. goes to the opposite state at HIGH level of clock pulse.
- We can make edge triggered flip-flop as well by using positive and negative edges of clock instead of HIGH and LOW level.

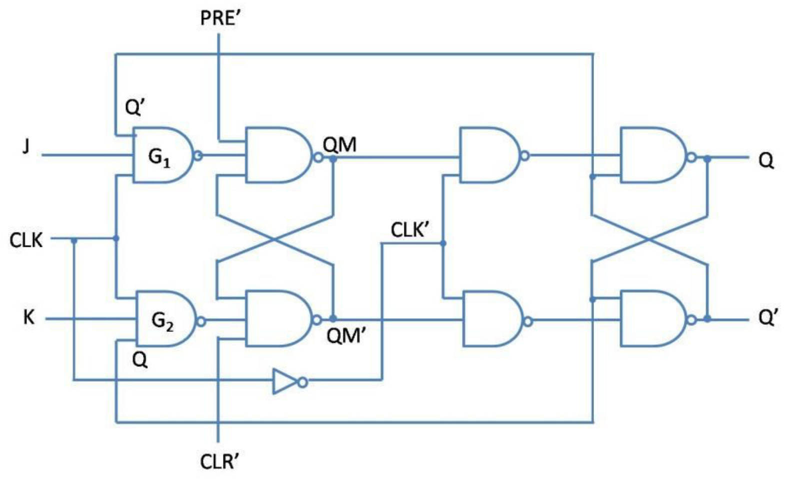
T Flip-Flop



En	T	Q_n	Q_{n+1}	State
1	0	0	0	No Change (NC)
1	0	1	1	
1	1	0	1	Toggle
1	1	1	0	
0	X	0	0	No Change (NC)
0	X	1	1	

- A T flip-flop has a single control input, labeled T for toggle.
- When T is HIGH, the flip-flop toggles on every new clock pulse.
- When T is LOW, the flip-flop remains in whatever state it was before.
- Although T flip-flops are not widely available commercially, it is easy to convert a J-K flip-flop to the functional equivalent of a T flip-flop by just connecting J and K together and labeling the common connection as T.
- Thus, when T = 1, we have J = K = 1, and the flip-flop toggles.
- When T = 0, we have J =K = 0, and so there is no change of state.

Master-Slave (Pulse-Triggered) J-K Flip-flop



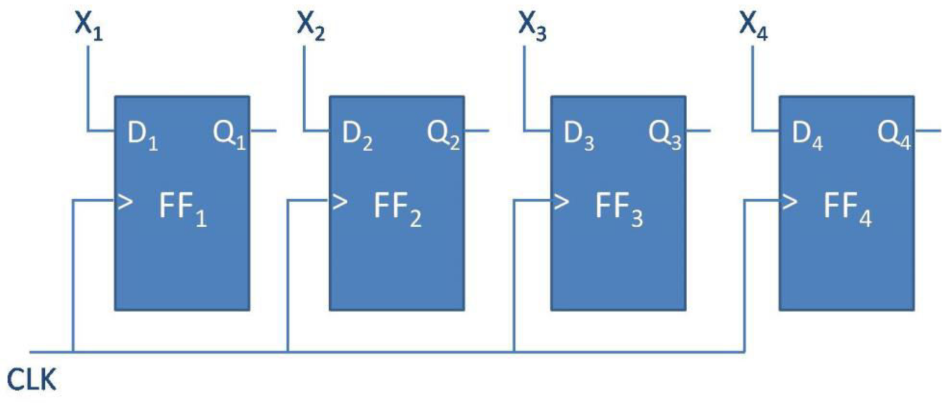
Inputs			Output	Comments
J	K	C	Q	
0	0	1	Q_0	No Change
0	1	1	0	Reset
1	0	1	1	Set
1	1	1	Q_0'	Toggle

- As shown in figure, the external control inputs J and K are applied to the master section.
- The master section is basically a gated JK latch, with Q output is connected back to the input of G_2 and Q' output is connected back to the input of G_1 and responds to the external J-K inputs applied to it at the positive edge of the clock signal.
- The slave section is the same as the master section except that it is clocked on the inverted clock pulse and thus responds to its control inputs at the negative edge of the clock pulse.
- Thus, the master section assumes the state determined by the J and K inputs at the positive edge of the clock pulse and the slave section copies the state of master section at negative edge of the clock pulse.
- The state of the slave then immediately appears on its Q and Q' outputs.

Registers

- As a flip-flop (FF) can store only one bit of data, a 0 or a 1, it is referred to as a single-bit register.
- A register is a set of FFs used to store binary data.
- The storage capacity of a register is the number of bits (1s and 0s) of digital data it can retain.
- Loading a register means setting or resetting the individual FFs, i.e. inputting data into the register so that their states correspond to the bits of data to be stored
- Loading may be serial or parallel.
- In serial loading, data is transferred into the register in serial form i.e. one bit at a time.
- In parallel loading, the data is transferred into the register in parallel form meaning that all the FFs are triggered into their new states at the same time.
- Types of Registers
  1. Buffer register
  2. Shift register
  3. Bidirectional shift register
  4. Universal shift register

4-bit Buffer Register

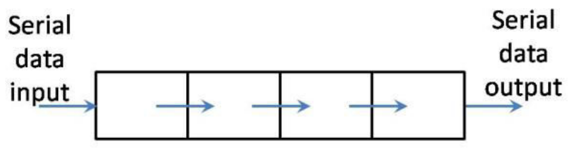


- Figure shows a 4-bit buffer register. The binary word to be stored is applied to the data terminals.
- On the application of clock pulse, the output word becomes the same as the word applied at the input terminals, i.e. the input word is loaded into the register by the application of clock pulse.
-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

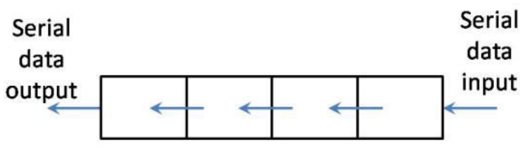
$$Q_4Q_3Q_2Q_1 = X_4X_3X_2X_1$$
$$Q = X$$

Shift Register

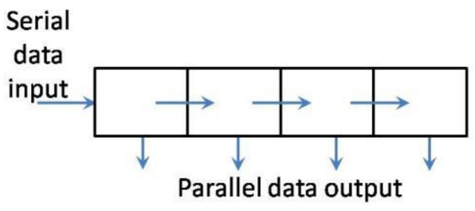
- A number of FFs connected together such that data may be shifted into and shifted out of them is called a shift register.
- Data may be shifted into or out of the register either in serial form or in parallel form.
- So, there are four basic types of shift registers:
    1. serial-in, serial-out
    2. serial-in, parallel out
    3. parallel-in, serial-out
    4. parallel-in, parallel-out
- Data may be rotated left or right. Data may be shifted from left to right or right to left at will, i.e. in a bidirectional way.
- Also, data may be shifted in serially (in either way) or in parallel and shifted out serially (in either way) or in parallel.



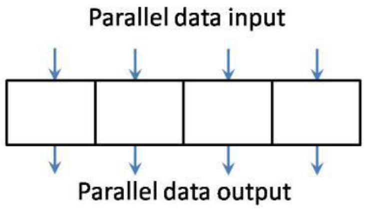
Serial-in, serial-out shift-right, shift register



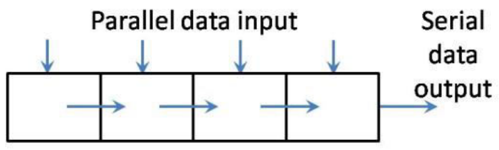
Serial-in, serial-out shift-left, shift register



Serial-in, parallel-out, shift register

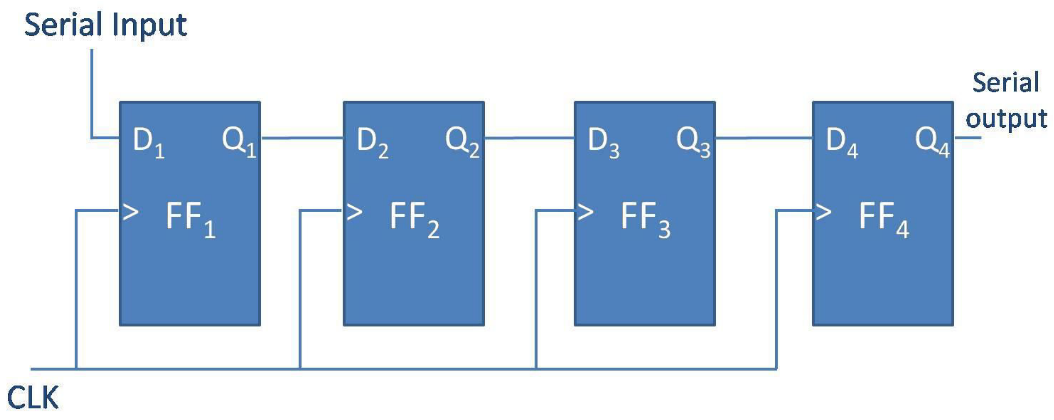


Parallel-in, parallel-out, shift register



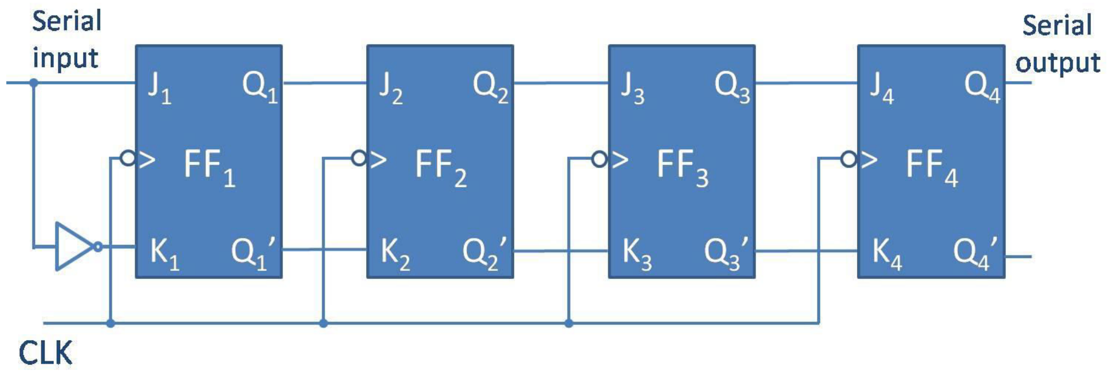
Parallel-in, serial-out, shift register

Serial-in, Serial-out, Shift register

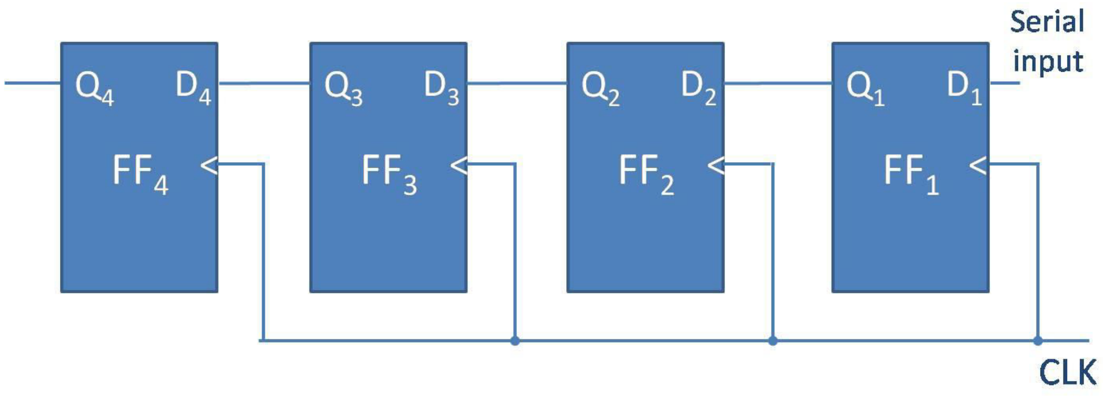


4 bit serial-in, serial-out, shift-right, shift register

- With four stages, i.e. four FFs, the register can store upto four bits.
- Serial data is applied at the D input of the first FF. The Q output of the first FF is connected to the D input of the second FF, the Q output of the second FF is connected to the D input of the third FF and the Q output of the third FF is connected to the D input of fourth FF.
- When serial data is transferred into a register, each new bit is clocked into the first FF at the positive 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, and so on.

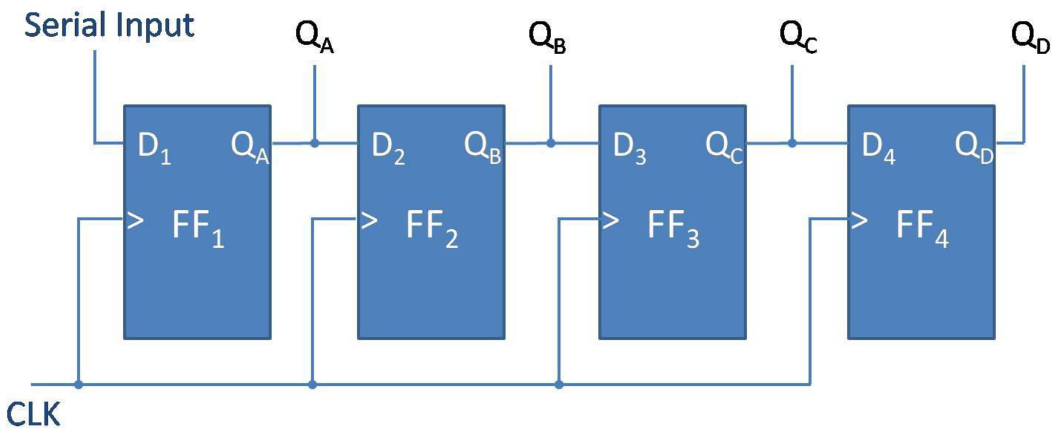


- A shift register can also be constructed using J-K FFs as shown in above figure.
- The data is applied at the J input of the first FF, the complement of this is fed to the K input of FF.
- The Q output of the first FF is connected to J input of the second FF, the Q output of the second FF to J input of the third FF, and so on.
- Also, Q_1' is connected K_2, Q_2' is connected to K_3, and so on.



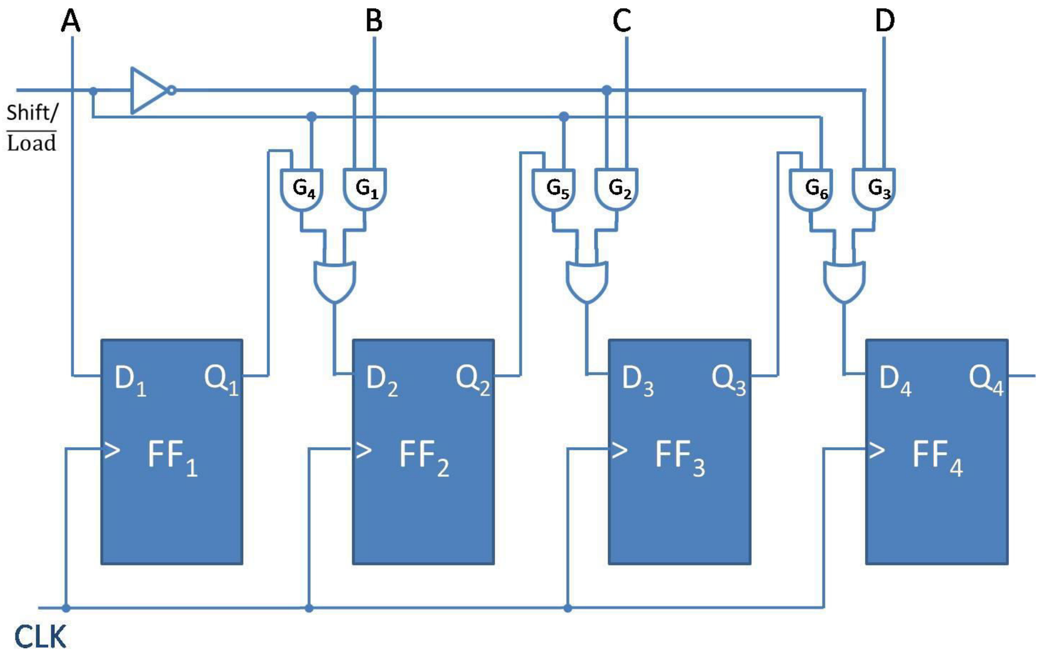
4-bit serial-in, serial-out, shift-left, shift register

Serial-in, Parallel-out, Shift register



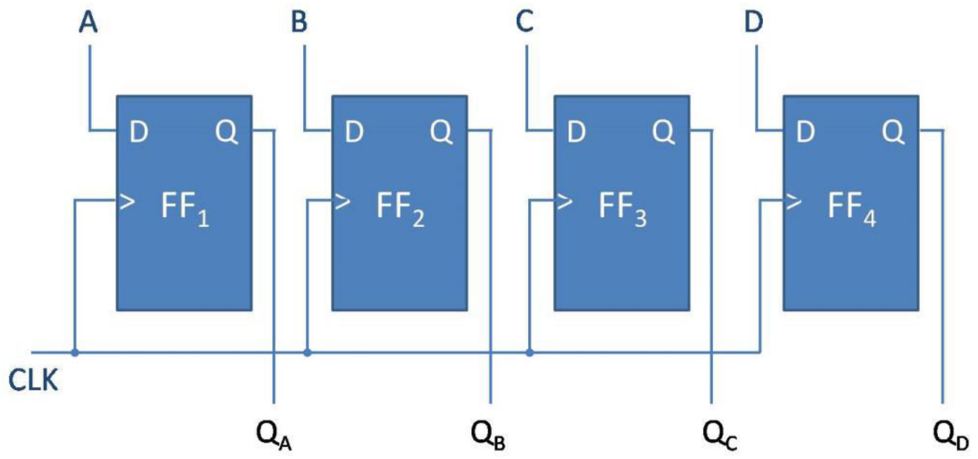
- In this type of register, the data bits are entered into the register serially, but the data stored in the register is shifted out in parallel form.
- Once the data bits are stored, each bit appears on its respective output line and all bits are available simultaneously, rather than on a bit-by-bit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

Parallel-in, Serial-out, Shift register



- There are four data lines A, B, C, and D through which the data is entered into the register in parallel form.
- The signal Shift/$\overline{LOAD}$ allows (a) the data to be entered in parallel form into the register and (b) the data to be shifted out serially from terminal Q_4.
- When Shift/$\overline{LOAD}$ line is HIGH, gates G_1, G_2, and G_3 are disabled, but gates G_4, G_5, and G_6 are enabled allowing the data bits to shift right from one stage to the next.
- When Shift/$\overline{LOAD}$ line is LOW, gates G_4, G_5, and G_6 are disabled, whereas gates G_1, G_2, and G_3 are enabled allowing the data input to appear at the D inputs of the respective FFs.
- When a clock pulse is applied, these data bits are shifted t the Q output terminals of the FFs and, therefore, data is inputted in one step.
- The OR gate allows wither the normal shifting operation or the parallel data entry depending on which NAD gates are enabled by the level on the Shift/$\overline{LOAD}$ input.

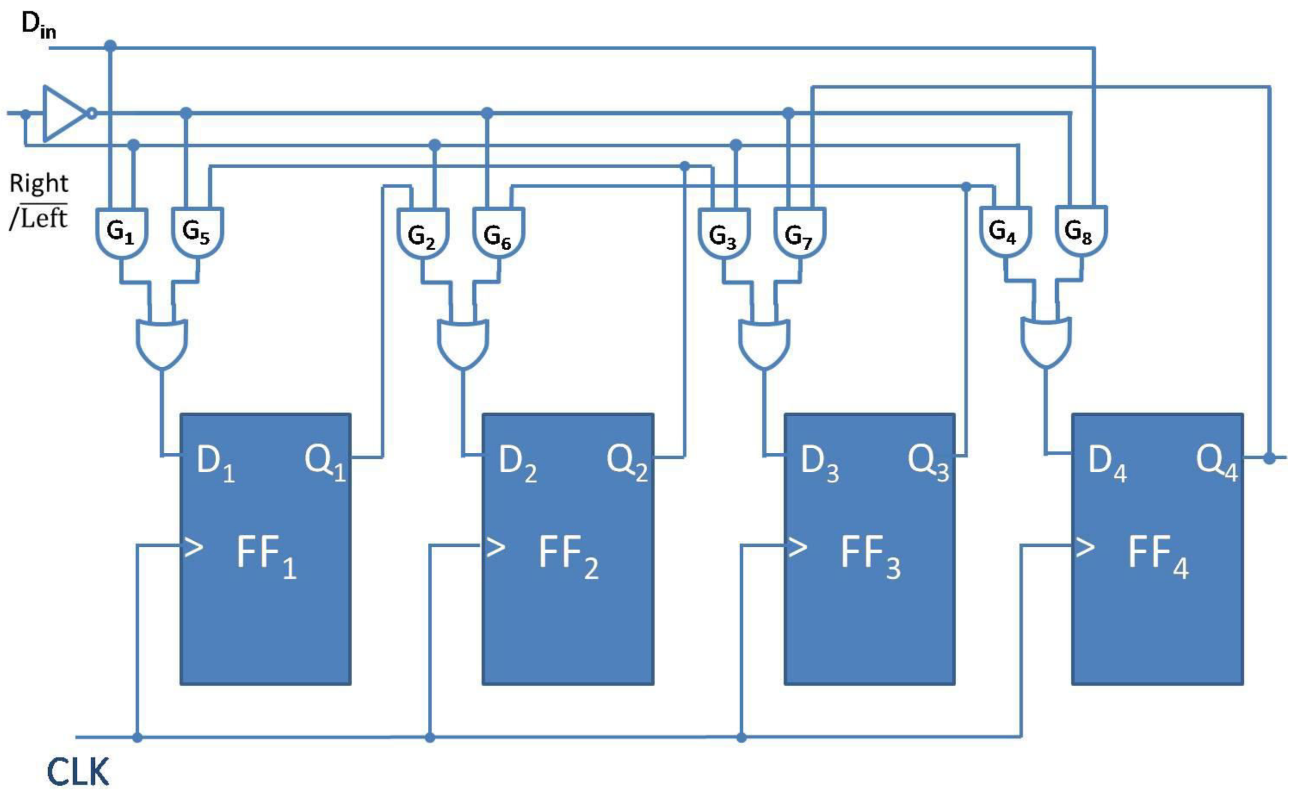
Parallel-in, Parallel-out, Shift register



- 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.

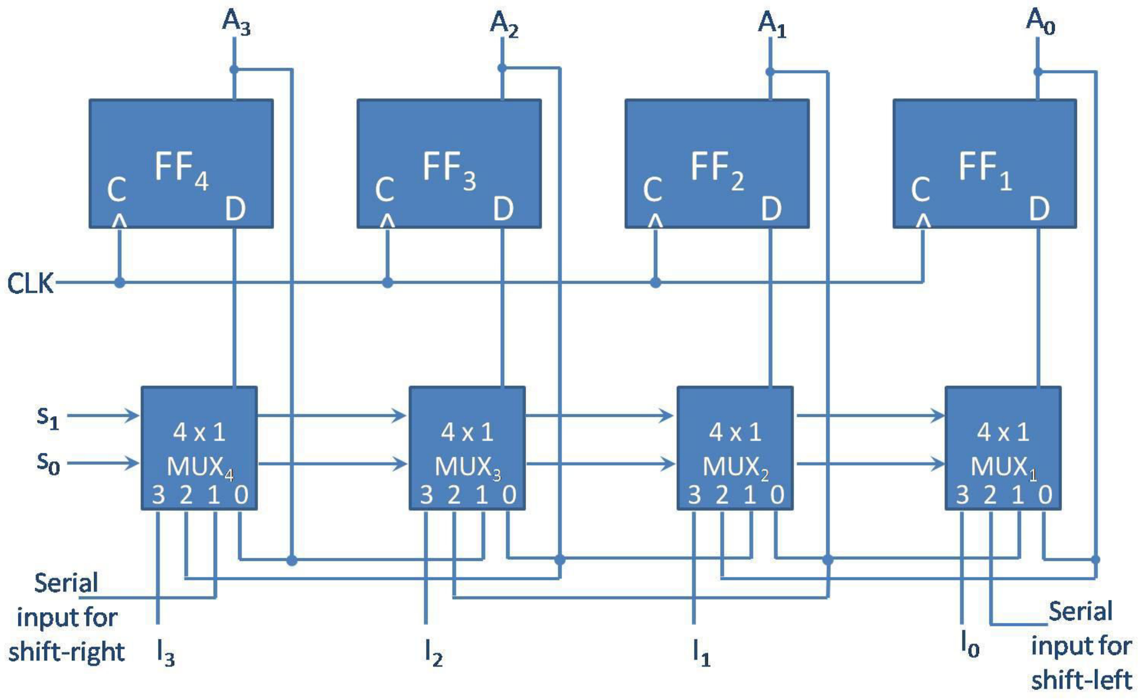
- Data is applied to the D input terminals of the FFs.
- When a clock pulse is applied, at the positive-going edge of that pulse, the D inputs are shifted into the Q outputs of the FFs.
- The register now stores the data. The stored data is available instantaneously for shifting out in parallel form.

Bidirectional Shift Register



- A bidirectional shift register is one in which the data bits can be shifted from left to right or from right to left.
- Right/$\overline{Left}$ is the mode signal. When Right/$\overline{Left}$ is a 1, the logic circuit works as a shift-right shift register. When Right/$\overline{Left}$ is a 0, the logic circuit works as a shift-left shift register.
- A HIGH on the Right/$\overline{Left}$ control input enables the AND gates G_1, G_2, G_3, and G_4 and disables the AND gates G_5, G_6, G_7, and G_8, 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/$\overline{Left}$ control input enables the AND gates G_5, G_6, G_7, and $G_8$and disables the AND gates G_1, G_2, G_3, and G_4, 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 left.

Universal Shift Register



- If the register has both shifts and parallel load capabilities, it is referred to as a universal shift register. So a 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 either in serial form or in parallel form.
- As per the figure shows, it consists of four 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 in each multiplexer is selected when S_1S_0 = 01, Input 2 in each multiplexer is selected when S_1S_0 = 10, and Input 3 in each multiplexer is selected when S_1S_0 = 11.
- The selection inputs control the mode of operation of the register according to the function entries in below table.

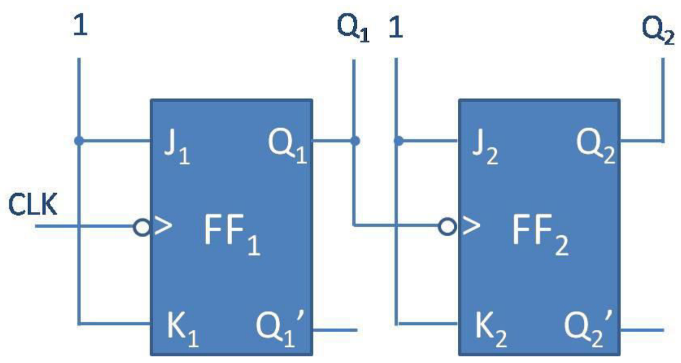
Mode Control		Register operation
S_1	S_0	
0	0	No Change
0	1	Shift right
1	0	Shift left
1	1	Parallel load

- When S_1S_0 = 00, the present value of the register is applied to the D inputs of flip-flops. This 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-flops, this causes a shift-right operation, with the serial input transferred into flip-flop FF_4.
- When S_1S_0 = 10, a shift-left operation results with the other serial input going into flip-flop FF_1.
- Finally S_1S_0 = 11, the binary information on the parallel input lines is transferred into the register simultaneously during the next clock edge.

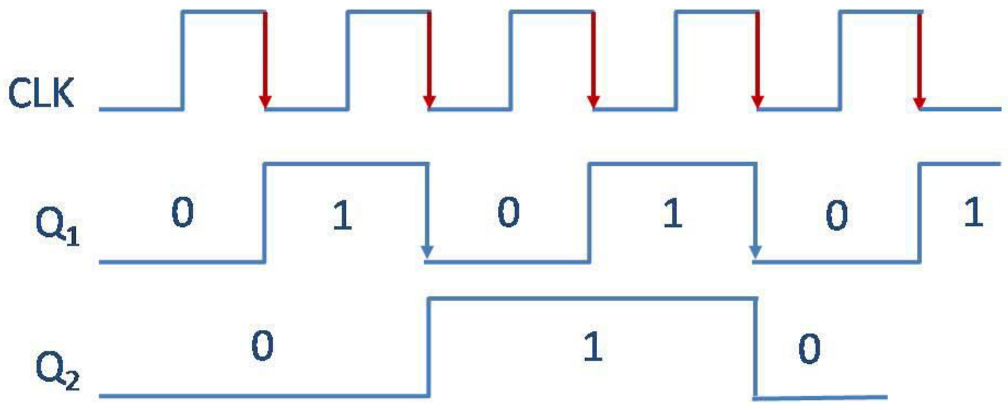
Asynchronous versus synchronous counters

	Asynchronous counters	Synchronous counters
1.	In this type of counter FFs are connected in such a way that the output of first FF drives the clock for the second FF; the output of the second drives the clock of the third and so on.	In this type of counter there is no connection between the output of first FF and clock input of next FF and so on.
2.	All the FFs are not clocked simultaneously.	All the FFs are clocked simultaneously.
3.	Design and implementation is very simple even for more number of states.	Design and implementation becomes tedious and complex as the number of states increases.
4.	Main drawback of these counters is their low speed as the clock is propagated through a number of FFs before it reaches the last FF.	Since clock is applied to all the FFs simultaneously, the total propagation delay is equal to the propagation delay of only one FF. Hence they are faster.

2-bit Ripple Up-Counter using Negative Edge-triggered Flip-Flop



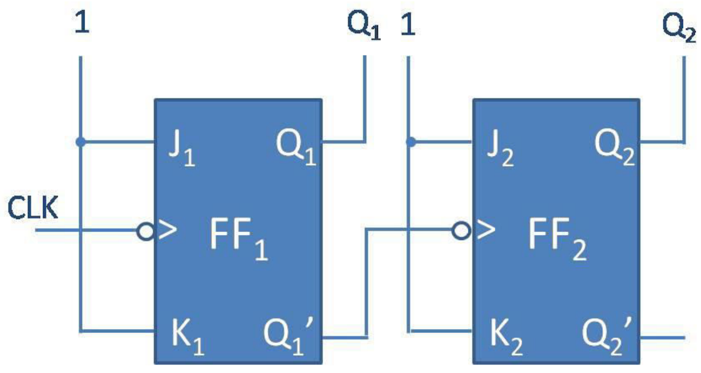
CLK	Present State		Next State	
	Q_1	Q_0	Q_1	Q_0
↓	0	0	0	1
↓	0	1	1	0
↓	1	0	1	1
↓	1	1	0	0



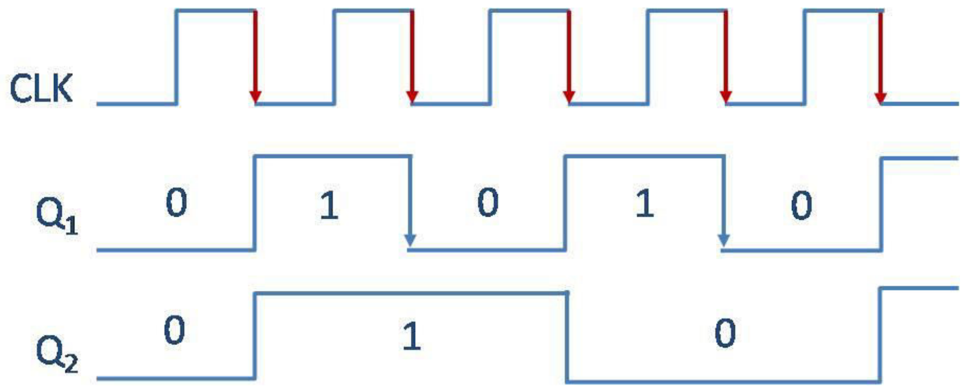
- The 2-bit up-counter counts in the order 0, 1, 2, 3, 0, … etc..
- The counter is initially reset to 00.
- When the first clock pulse is applied, FF_1 toggles at the negative edge of this pulse, therefore, Q_1 goes from LOW to HIGH.
- This becomes a positive edge at the clock input of FF_2. So FF_2 is not affected, and hence the state of the counter after one clock pulse is $Q_1 = 1$ and $Q_2 = 0$, i.e. 01.
- At the negative edge of the second clock pulse, FF_1 toggles.

- So Q_1 changes from HIGH to LOW and this negative edge clock applied to CLK of FF_2 activates FF_2, and hence, Q_2 goes from LOW to HIGH. Therefore, $Q_1 = 0$ and $Q_2 = 1$, i.e. 10 is the state of the counter after the second clock pulse.
- At the negative edge of the third clock pulse, FF_1 toggles.
- So Q_1 changes from a 0 to a 1. This becomes a positive edge to FF_2, hence FF_2 is not affected. Therefore, $Q_2 = 1$ and $Q_1 = 1$, i.e. 11 is the state of the counter after the third clock pulse.
- At the negative edge of the fourth clock pulse, FF_1 toggles.
- So, Q_1 changes from a 1 to a 0. This negative edge at Q_1 toggles FF_2, hence Q_2 also changes from a 1 to a 0. Therefore, $Q_2 = 0$ and $Q_1 = 0$, i.e. 00 is the state of the counter after the fourth clock pulse.
- So, it acts as a mod-4 counter with Q_1 as the LSB and Q_2 as the MSB.

2-bit Ripple Down-Counter using Negative Edge-triggered Flip-Flop



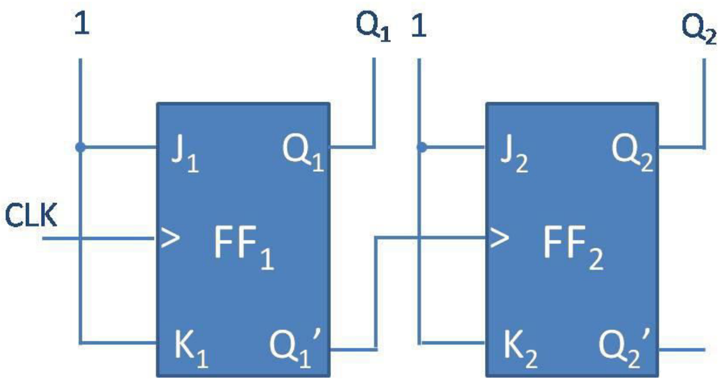
CLK	Present State		Next State	
	Q_1	Q_0	Q_1	Q_0
↓	0	0	1	1
↓	1	1	1	0
↓	1	0	0	1
↓	0	1	0	0



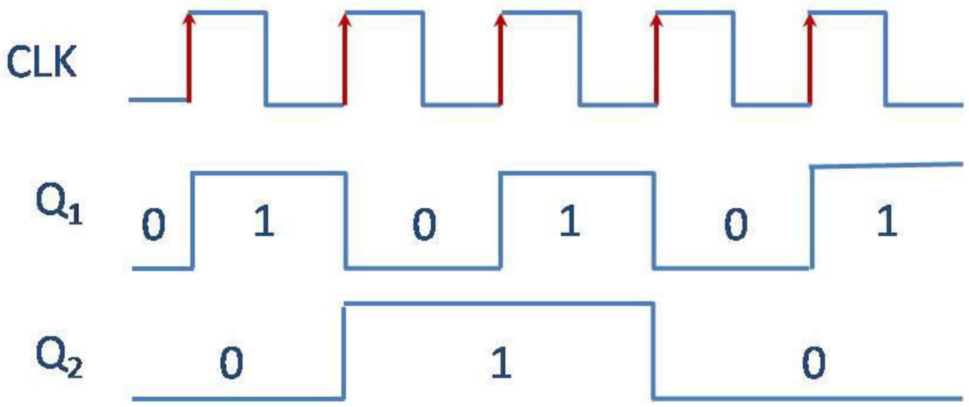
- A 2-bit down-counter counts in the order 0, 3, 2, 1, 0, …etc.
- For down counting, Q_1' of FF_1 is connected to the clock of FF_2. Let initially all the FFs be reset, i.e. 00.
- At the negative edge of the first clock pulse, FF_1 toggles. So, Q_1 goes from a 0 to a 1 and Q_1' goes from a 1 to a 0.
- This negative edge at Q_1' applied to the clock input of FF_2, toggles FF_2 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 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 edge at Q_1' does not affect FF_2 and, therefore Q_2 remains at a 1. Hence, the state of the counter after second clock pulse is 10.
- At the negative edge of third clock pulse, FF_1 toggles. So, Q_1 goes from a 0 to a 1 and Q_1' goes from a 1 to a 0.
- This negative edge at Q_1' applied to the clock input of FF_2, toggles FF_2 and, therefore, Q_2 goes from a 1 to a 0. Hence, the state of the counter is 01.
- At the negative edge of fourth clock pulse, FF_1 toggles. So, Q_1 goes from a 1 to a 0 and Q_1' goes from a 0 to a 1.
- This positive edge at Q_1' does not affect FF_2 and, therefore Q_2 remains at a 0. Hence, the state of the counter after fourth clock pulse is 00.

2-bit Ripple Up-Counter using Positive Edge-triggered Flip-Flop



CLK	Present State		Next State	
	Q_1	Q_0	Q_1	Q_0
↑	0	0	0	1
↑	0	1	1	0
↑	1	0	1	1
↑	1	1	0	0

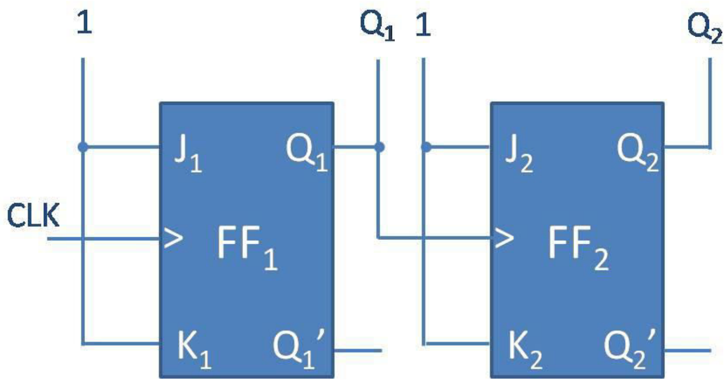


- A 2-bit up-counter counts in the order 0, 1, 2, 3, 0, …etc.
- For up counting in positive edge triggering, Q_1' of FF_1 is connected to the clock of FF_2. Let initially all the FFs be reset, i.e. 00.
- At the positive edge of the first clock pulse, FF_1 toggles. So, Q_1 goes from a 0 to a 1 and Q_1' goes from a 1 to a 0.
- This negative edge at Q_1' does not affect FF_2 and, therefore Q_2 remains at a 0. Hence, the state of the counter after first clock pulse is 01.
- At the positive 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 edge at Q_1' applied to the clock input of FF_2, toggles FF_2 and, therefore, Q_2 goes from a 0 to a

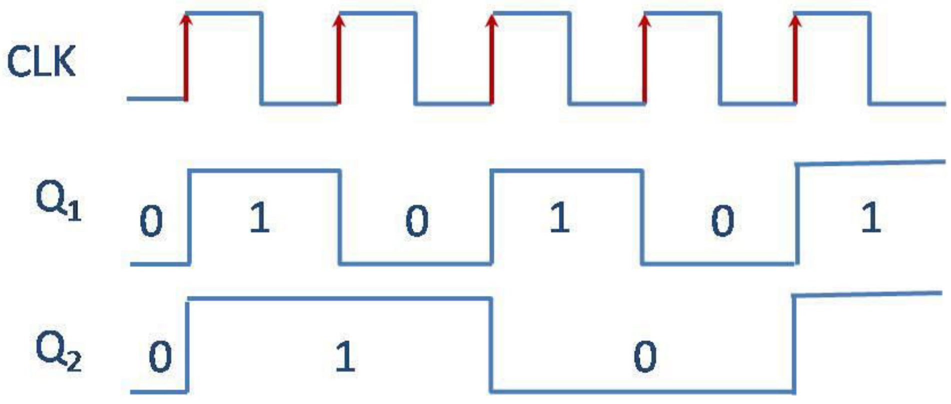
1. Hence, the state of the counter is 10.

- At the positive edge of third clock pulse, FF_1 toggles. So, Q_1 goes from a 0 to a 1 and Q_1' goes from a 1 to a 0.
- This negative edge at Q_1' does not affect FF_2 and, therefore Q_2 remains at a 1. Hence, the state of the counter after third clock pulse is 11.
- At the positive edge of fourth clock pulse, FF_1 toggles. So, Q_1 goes from a 1 to a 0 and Q_1' goes from a 0 to a 1.
- This positive edge at Q_1' applied to the clock input of FF_2, toggles FF_2 and, therefore, Q_2 goes from a 1 to a 0. Hence, the state of the counter is 00.

2-bit Ripple Down-Counter using Positive Edge-triggered Flip-Flop



CLK	Present State		Next State	
	Q_1	Q_0	Q_1	Q_0
↑	0	0	1	1
↑	1	1	1	0
↑	1	0	0	1
↑	0	1	0	0



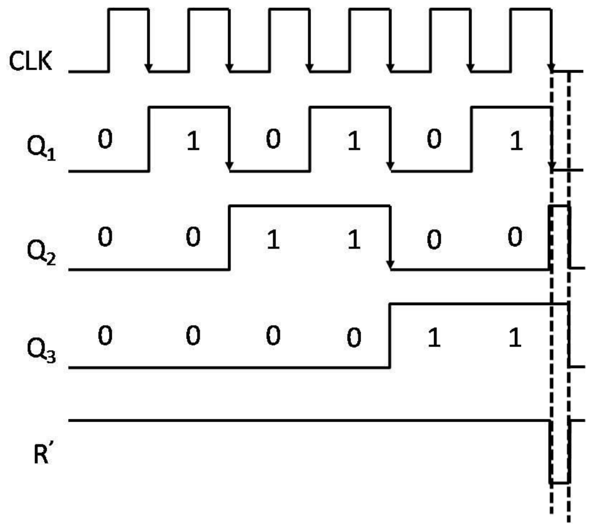
- The 2-bit down-counter counts in the order 0, 3, 2, 1, 0, … etc..
- The counter is initially reset to 00.
- When the first clock pulse is applied, FF_1 toggles at the positive edge of this pulse, therefore, Q_1 goes from LOW to HIGH.
- This positive edge at Q_1 toggles FF_2, hence Q_2 also changes from a 0 to a 1. Therefore, $Q_2 = 1$ and $Q_1 = 1$, i.e. 11 is the state of the counter after the first clock pulse.
- At the positive edge of the second clock pulse, FF_1 toggles.
- So Q_1 changes from a 1 to a 0. This becomes a negative edge to FF_2, hence FF_2 is not affected. Therefore, $Q_2 = 1$ and $Q_1 = 0$, i.e. 10 is the state of the counter after the second clock pulse.
- At the positive edge of the third clock pulse, FF_1 toggles.

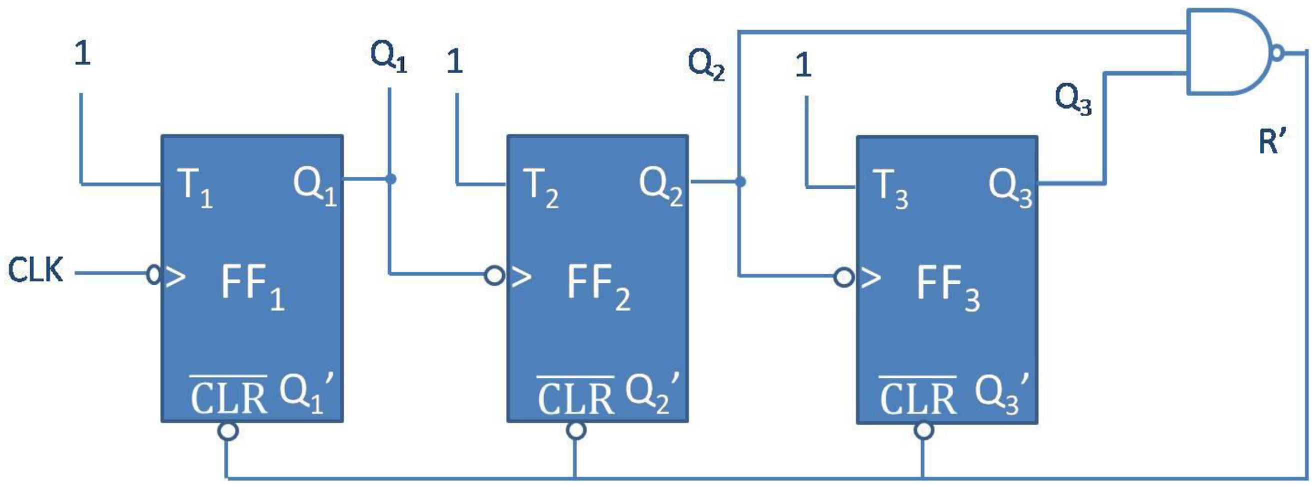
- So Q_1 changes from a 0 to a 1. This positive edge at Q_1 toggles FF_2, hence Q_2 also changes from a 1 to a 0. Therefore, $Q_2 = 0$ and $Q_1 = 1$, i.e. 01 is the state of the counter after the third clock pulse.
- At the positive edge of the fourth clock pulse, FF_1 toggles.
- So Q_1 changes from a 1 to a 0. This becomes a negative edge to FF_2, hence FF_2 is not affected. Therefore, $Q_2 = 0$ and $Q_1 = 0$, i.e. 00 is the state of the counter after the fourth clock pulse.

Mod-6 Asynchronous Counter Using T FFs

- A mod-6 counter has six stable states 000, 001, 010, 011, 100, 101.
- It is also known as "Divide by 6" counter and it requires 3 Flip-flops for designing.
- At the 6th clock pulse, it will again reset to 000 via feedback circuit.
- Reset signal R = 1 at time of 110, R = 0 for 000 to 101 and R = X for invalid states i.e. 111.
- Therefore, $R = Q_3Q_2Q_1' + Q_3Q_2Q_1 = Q_3Q_2$.

After Pulses	State			R
	Q_3	Q_2	Q_1	
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	





Design of Synchronous Counters

- Step 1. Number of flip-flops:
  Based on the description of the problem, determine the required number n of the FFs - the smallest value of n is such that the number of states $N \leq 2^n$ and the desired counting sequence.
- Step 2. State diagram:
  Draw the state diagram showing all the possible states.
- Step 3. Choice of flip-flops and excitation table:
  Select the type of flip-flops 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 the required excitations.
- Step 4. Minimal expressions for excitations:
  Obtain the minimal expressions for the excitations of the FFs using K-maps for the excitations of the flip-flops in terms of the present states and inputs.
- Step 5. Logic Diagram:
  Draw the logic diagram based on the minimal expressions.

Flip-Flop Excitation Tables

(1) S-R excitation table

PS	NS	Required inputs	
Q_n	Q_{n+1}	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

(2) J-K excitation table

PS	NS	Required inputs	
Q_n	Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

(3) D excitation table

PS	NS	Required inputs
Q_n	Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

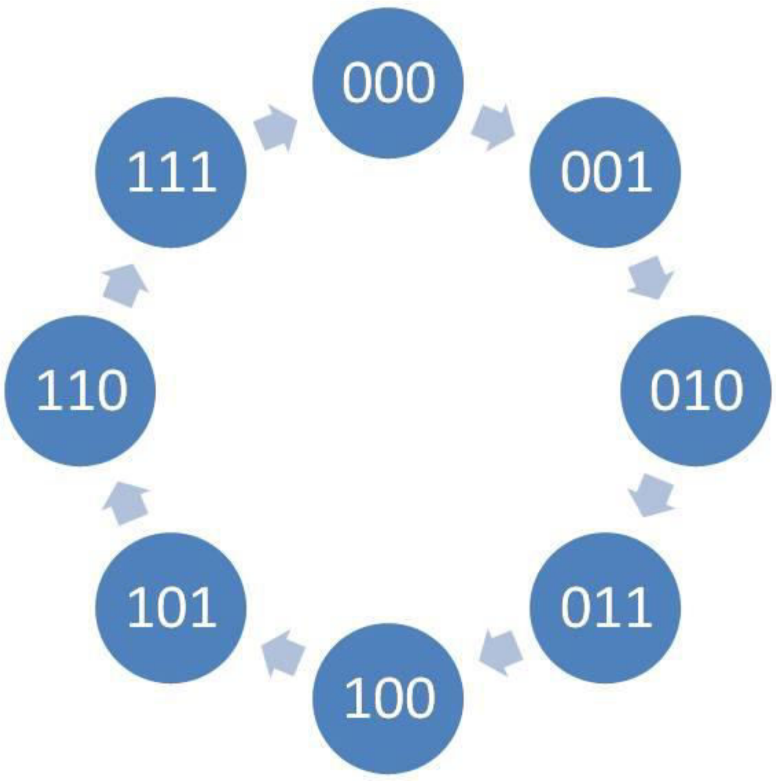
(4) T excitation table

PS	NS	Required inputs
Q_n	Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

Design of Synchronous 3-bit Up Counters

- Step 1. Number of flip-flops:
  A 3-bit up-counter requires 3 flip-flops. The counting sequence is 000, 001, 010, 011, 100, 101, 110, 111, 000 …

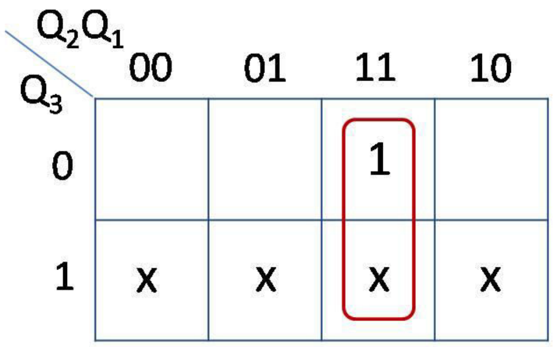
- Step 2. Draw the state diagram:



- Step 3. Select the type of flip-flops and draw the excitation table:
  JK flip-flops are selected and the excitation table of a 3-bit up-counter using J-K flip-flops is drawn as shown below.

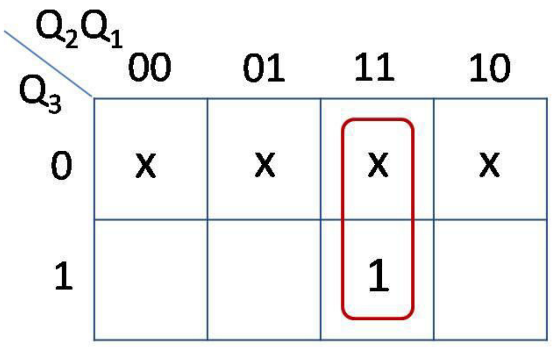
Present State			**Next State**			**Required Excitation**					
Q_3	Q_2	Q_1	Q_3	Q_2	Q_1	J_3	K_3	J_2	K_2	J_1	K_1
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	0	0	X	1	X	X	1
0	1	0	0	1	1	0	X	X	0	1	X
0	1	1	1	0	0	1	X	X	1	X	1
1	0	0	1	0	1	X	0	0	X	1	X
1	0	1	1	1	0	X	0	1	X	X	1
1	1	0	1	1	1	X	0	X	0	1	X
1	1	1	0	0	0	X	1	X	1	X	1

- Step 4. Obtain the minimal expressions
  From excitation table, $J_1 = K_1 = 1$.
  K – Maps for excitations J_3, K_3, J_2 and K_2 and their minimized form are as follows:



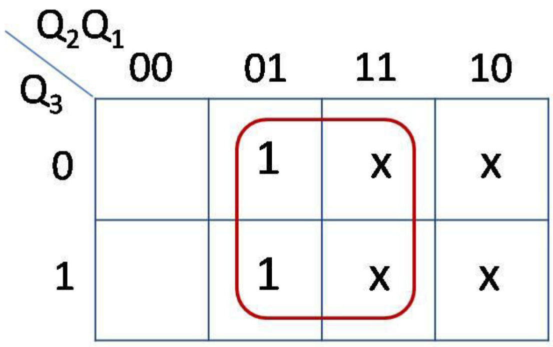
Q_3 \ Q_2Q_1	00	01	11	10
0			1	
1	x	x	x	x

$J_3 = Q_2Q_1$



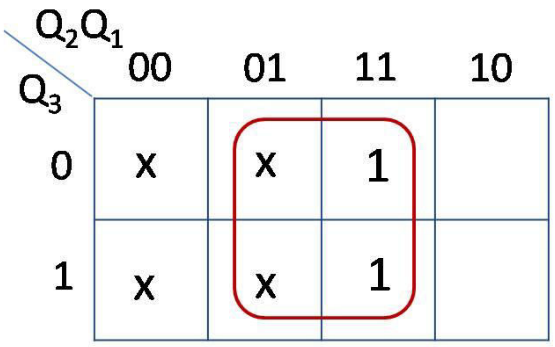
Q_3 \ Q_2Q_1	00	01	11	10
0	x	x	x	x
1			1	

$K_3 = Q_2Q_1$



Q_3 \ Q_2Q_1	00	01	11	10
0		1	x	x
1		1	x	x

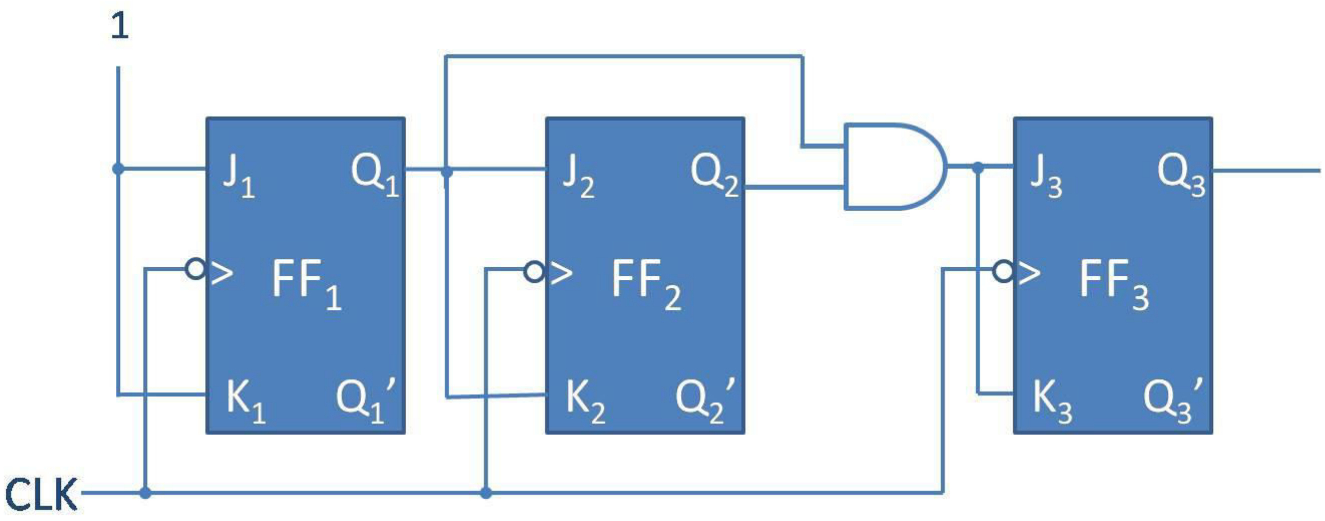
$J_2 = Q_1$



Q_3 \ Q_2Q_1	00	01	11	10
0	x	x	1	
1	x	x	1	

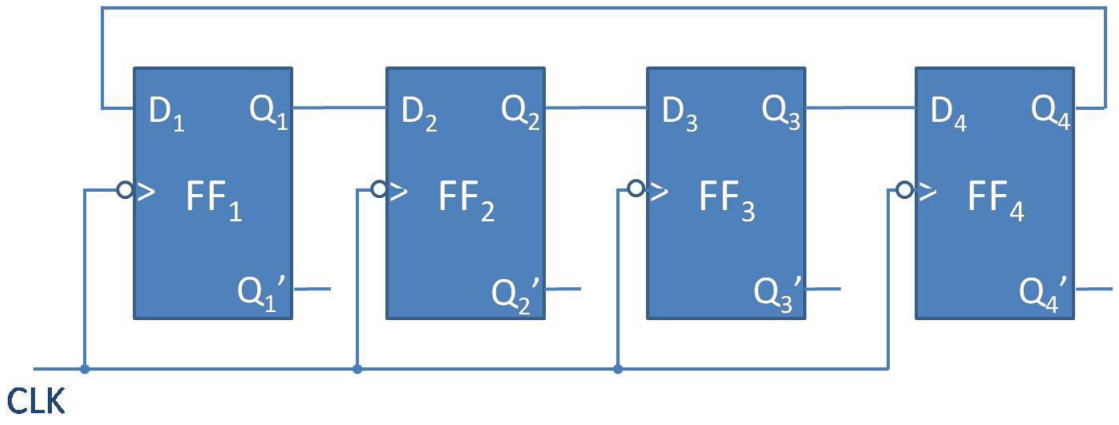
$K_2 = Q_1$

- Step 5. Draw the logic diagram

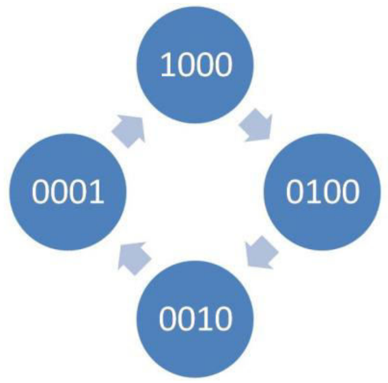


Ring Counter

- This is the simplest shift register counter. The basic ring counter using D FFs is shown in figure.
- The FFs are arranged as in a normal shift register, i.e. Q output of each stage is connected to the D input of the next stage, but the Q output of the last FF is connected back to the D input of the first FF such that the array of FFs is arranged in a ring, and therefore, the name *ring counter.*



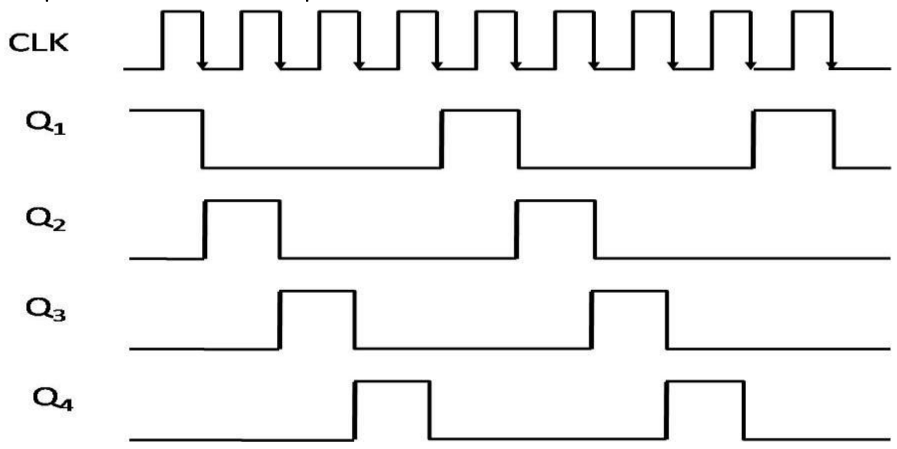
- State diagram:-



- Sequence table

Q_1	Q_2	Q_3	Q_4	After clock pulse
1	0	0	0	0
0	1	0	0	1
0	0	1	0	2
0	0	0	1	3
1	0	0	0	4
0	1	0	0	5
0	0	1	0	6
0	0	0	1	7

- In most instances, only 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. After each clock pulse, the contents of the register are shifted to the right by one bit and Q_4 is shifted to Q_1.
- The sequence repeats after four clock pulses.

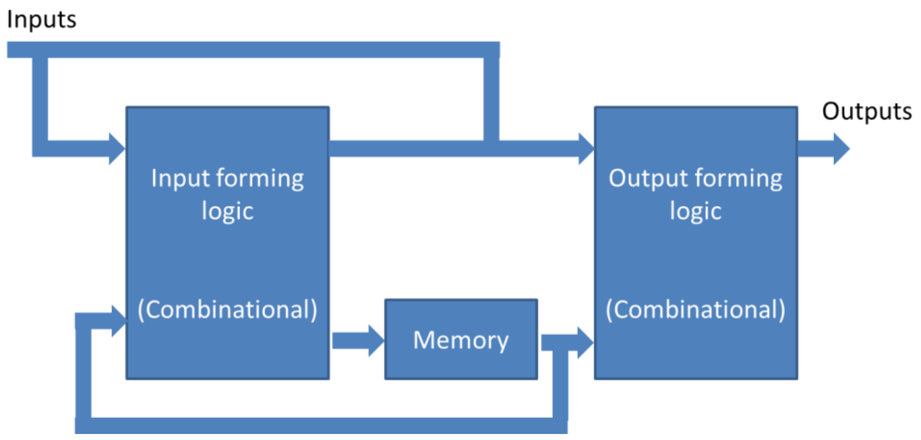


Need for State Machine or Advantages of State Machine

- The first advantage is that many electronic systems require the type of sequential operation exhibited by state machines. Therefore, state machine design can he applied to the solution of a wide variety of practical circuit problems.
- The second advantage is that state machine design methods lead to minimal design. In combinational circuits, we found that the Karnaugh map was useful in minimizing the number of gates required to implement a logic function. Although the importance of this tool has perhaps diminished as a result of the availability of MUXs, PLAs, PALS, and decoders, it remains a significant method in combinational design. The state machine design procedure relates to sequential circuit design the same way the K-map relates to combinational circuit design. This method results in the minimum number of required flip-flops and can minimize other circuitry in the system as well.
- The third advantage of the design method is that it is a well-developed, orderly procedure that anticipates and solves commonly occurring problems of sequential circuits. Trial-and-error design procedures often result in the appearance of very narrow unwanted pulses or glitches on output lines or occasional oscillation problems. State machine methods eliminate these problems and reduce the time taken to debug the implemented hardware.

General model for a sequential or state machine

- A large majority of practical state machines use clocked flip-flops as the storage elements.
- The general model of the sequential machine is shown in figure.



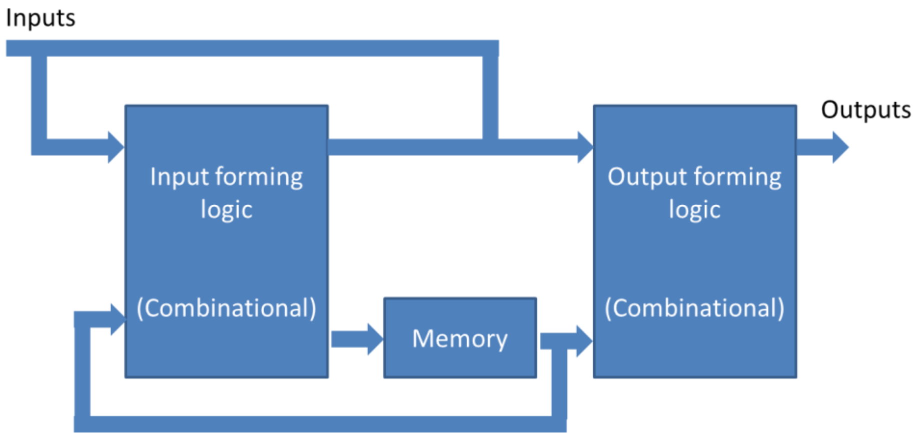
- This model is also called the Mealy machine after the man who first proposed the model.
- The input forming logic (IFL) and the output forming logic (OFL) sections are made up of combinational logic circuits.
- The memory section contains the state of the system.
- A path is provided from memory output to the IFL.
- Both input signals and present state signals drive the IFL to determine the next state of the system.
- The outputs are determined by the present state and the system inputs.
- A slight variation of the Mealy machine is the Moore machine, which uses only the memory to drive the OFL.
- In this case, the output is a function only of the state of the system.
- Another important characteristic of the state machine depends on whether the system is clock-driven or not.
- In many digital systems, a timing reference signal is required.
- Some type of astable multivibrator is generally used to produce a continuous clock signal. We refer to a variable as clock-driven if that variable changes value only at the time of a clock transition. When a variable is clock-driven, it is considered a synchronous variable.

- If a variable can change at a time not related to transitions of the reference clock, it is called an asynchronous variable.

Types of finite state machines

Mealy Model

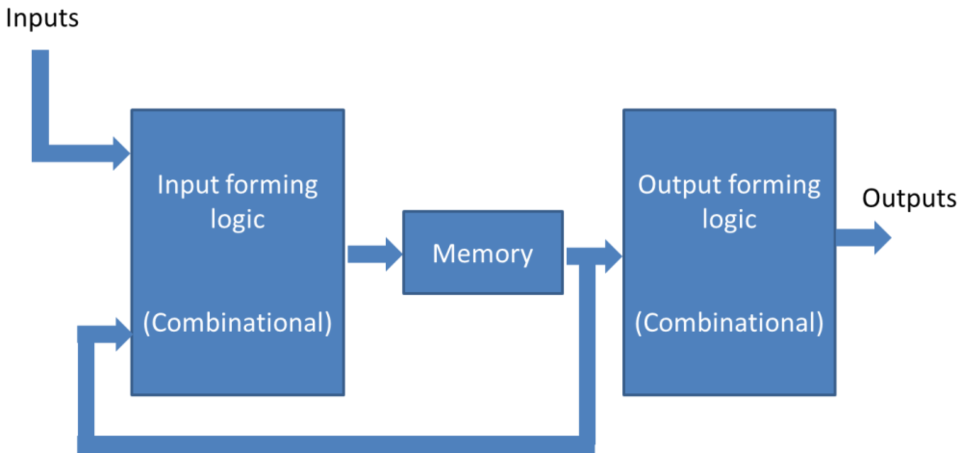
- When the output of the sequential circuit depends on both the present state of the flip-flops and on the inputs, the sequential circuit is referred to as Mealy circuit or Mealy machine.
- Mealy circuit can be represented with its block schematic as shown in the figure.



- A path is provided from memory output to the IFL.
- Both input signals and present state signals drive the IFL to determine the next state of the system.
- The outputs are determined by the present state and the system inputs.

Moore Model

- When the output of the sequential circuit depends only on the present state of the flip-flop, the sequential circuit is referred to as Moore circuit or Moore Machine.
- Moore circuit can be represented with its block schematic as shown in figure.



- A path is provided from memory output to the IFL.
- As shown in the figure, present state signals drive the IFL to determine the next state of the system.
- The outputs are determined by the present state and not the system inputs.

Design of synchronous counters.

- **Step 1. Number of flip-flops:**
  Based on the description of the problem, determine the required number n of the FFs - the smallest value of n is such that the number of states $N \le 2^n$ and the desired counting sequence.
- **Step 2. State diagram:**
  Draw the state diagram showing all the possible states.
- **Step 3. Choice of flip-flops and excitation table:**
  Select the type of flip-flops 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 the required excitations.
- **Step 4. Minimal expressions for excitations:**
  Obtain the minimal expressions for the excitations of the FFs using K-maps for the excitations of the flip-flops in terms of the present states and inputs.
- **Step 5. Logic Diagram:**
  Draw the logic diagram based on the minimal expressions.
- **Excitation Tables**

PS	NS	Required inputs	
Q_n	Q_{n+1}	S	R
0	0	0	x
0	1	1	0
1	0	0	1
1	1	x	0

S-R FF

PS	NS	Required inputs	
Q_n	Q_{n+1}	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

J-K FF

PS	NS	Required inputs
Q_n	Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

D FF

PS	NS	Required inputs
Q_n	Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

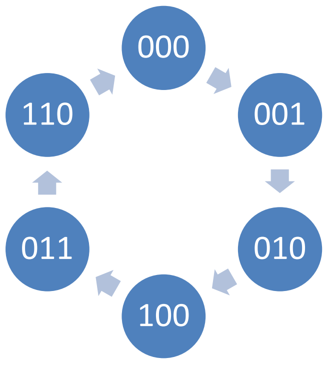
T FF

Design a counter to generate the repetitive sequence 0,1,2,4,3,6.

Step 1. Number of flip-flops:

A counter with repetitive sequence 0, 1, 2, 4, 3, 6 requires 3 flip-flops. The counting sequence is 000, 001, 010, 100, 011, 110, 000 …

Step 2. Draw the state diagram:



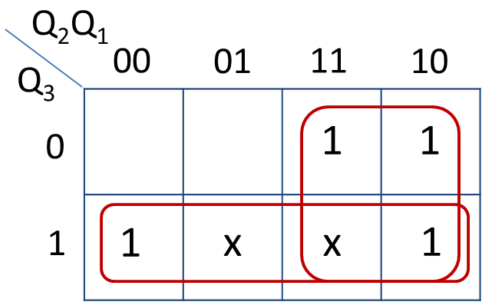
Step 3. Select the type of flip-flops and draw the excitation table:

T flip-flops are selected and the excitation table of a given sequence counter using T flip-flops is shown below.

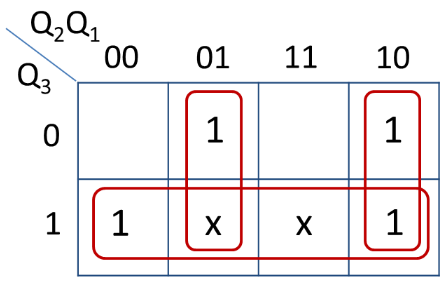
PS			NS			Required excitations		
Q_3	Q_2	Q_1	Q_3	Q_2	Q_1	T_3	T_2	T_1
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	1	0	0	1	1	0
1	0	0	0	1	1	1	1	1
0	1	1	1	1	0	1	0	1
1	1	0	0	0	0	1	1	0

Step 4. Obtain the minimal expressions

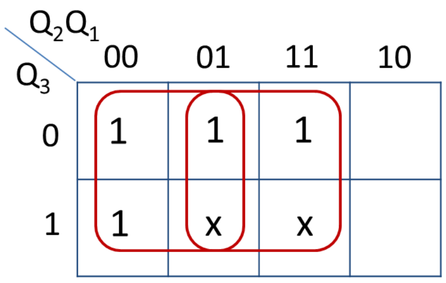
K – Maps for excitations T_3, T_2, and T_1 and their minimized form are as follows:



$T_3 = Q_3 + Q_2$

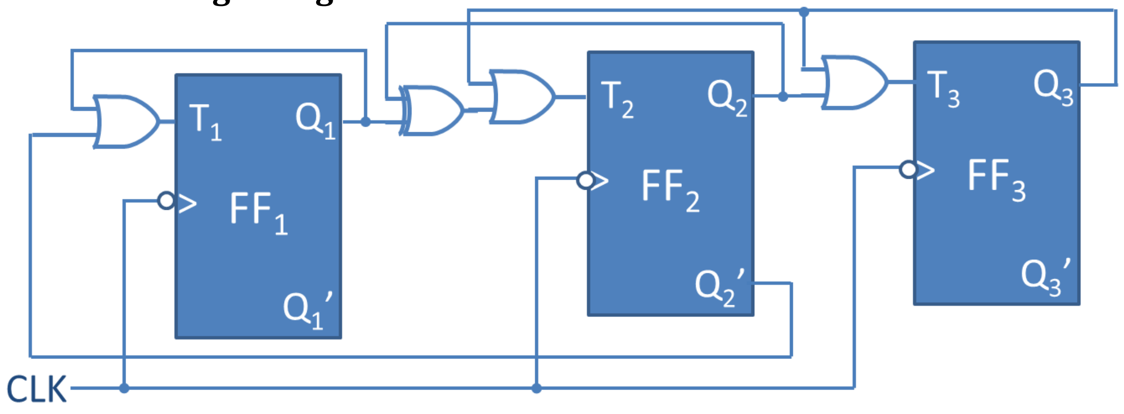


$T_2 = Q_3 + Q_1Q_2' + Q_1'Q_2$



$T_1 = Q_2' + Q_1$

Step 5. Draw the logic diagram

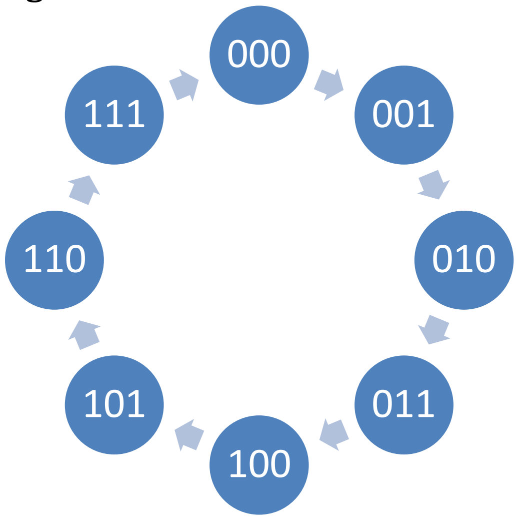


Design 3-bit synchronous up counter using T flip flop.

Step 1. Number of flip-flops:

A 3 bit up counter requires 3 flip-flops. The counting sequence is 000, 001, 010, 011, 100, 101, 110, 111, 000 …

Step 2. Draw the state diagram:



Step 3. Select the type of flip-flops and draw the excitation table:

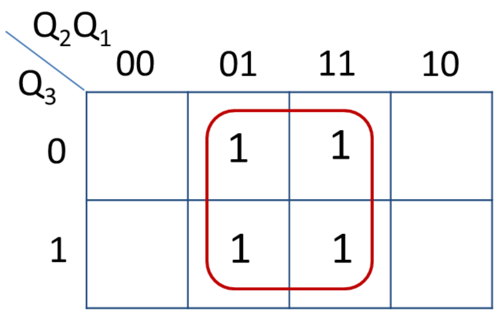
T flip-flops are selected and the excitation table of a given sequence counter using T flip-flops is shown below.

PS			NS			Required excitations		
Q_3	Q_2	Q_1	Q_3	Q_2	Q_1	T_3	T_2	T_1
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1
1	0	1	1	1	0	0	1	1
1	1	0	1	1	1	0	0	1
1	1	1	0	0	0	1	1	1

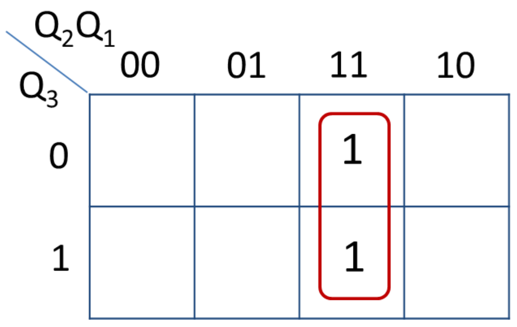
Step 4. Obtain the minimal expressions

From excitation table, $T_1 = 1$.

K – Maps for excitations T_3 and T_2 and their minimized form are as follows:

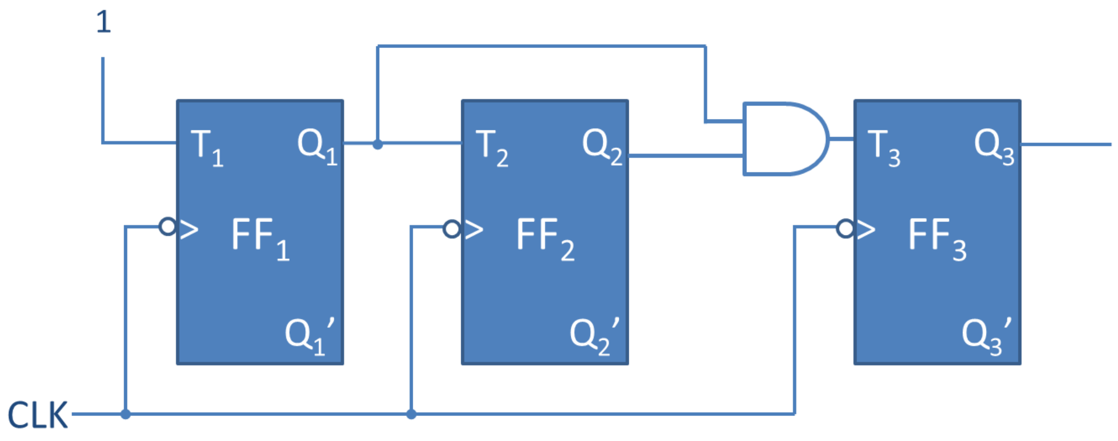


$T_2 = Q_1$



$T_3 = Q_1Q_2$

Step 5. Draw the logic diagram

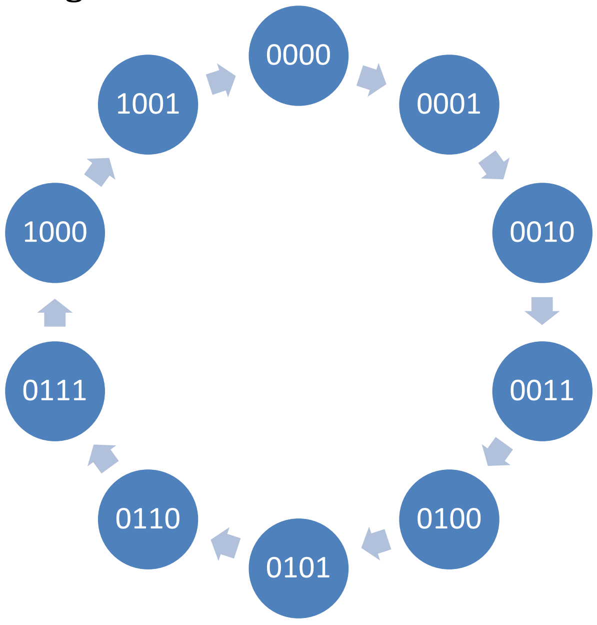


Design a synchronous BCD counter with JK flip-flops.

Step 1. Number of flip-flops:

A BCD counter requires 4 flip-flops. The counting sequence is 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 0000 …

Step 2. Draw the state diagram:



Step 3. Select the type of flip-flops and draw the excitation table:

J-K flip-flops are selected and the excitation table of a given sequence counter using J-K flip-flops is shown below.

PS				NS				Required excitations							
Q_4	Q_3	Q_2	Q_1	Q_4	Q_3	Q_2	Q_1	J_4	K_4	J_3	K_3	J_2	K_2	J_1	K_1
0	0	0	0	0	0	0	1	0	x	0	x	0	x	1	x
0	0	0	1	0	0	1	0	0	x	0	x	1	X	x	1
0	0	1	0	0	0	1	1	0	x	0	x	x	0	1	x
0	0	1	1	0	1	0	0	0	x	1	x	x	1	x	1
0	1	0	0	0	1	0	1	0	x	x	0	0	X	1	x
0	1	0	1	0	1	1	0	0	x	x	0	1	X	x	1
0	1	1	0	0	1	1	1	0	x	x	0	x	0	1	x
0	1	1	1	1	0	0	0	1	x	x	1	x	1	x	1
1	0	0	0	1	0	0	1	x	0	0	x	0	x	1	x
1	0	0	1	0	0	0	0	x	1	0	x	0	x	x	1

Step 4. Obtain the minimal expressions

From excitation table, it is clear that $J_1 = K_1 = 1$

K – Maps for excitations J_4, K_4, J_3, K_3, J_2, K_2, J_1 and K_1 and their minimized form are as follows:

$Q_4Q_3 \backslash Q_2Q_1$	00	01	11	10
00				
01			1	
11	x	x	x	x
10	x	x	x	x

$J_4 = Q_3Q_2Q_1$

$Q_4Q_3 \backslash Q_2Q_1$	00	01	11	10
00	x	x	x	x
01	x	x	x	x
11	x	x	x	x
10		1	x	x

$K_4 = Q_1$

$Q_4Q_3 \backslash Q_2Q_1$	00	01	11	10
00			1	
01	x	x	x	x
11	x	x	x	x
10			x	x

$J_3 = Q_1Q_2$

$Q_4Q_3 \backslash Q_2Q_1$	00	01	11	10
00	x	x	x	x
01			1	
11	x	x	x	x
10	x	x	x	x

$K_3 = Q_1Q_2$

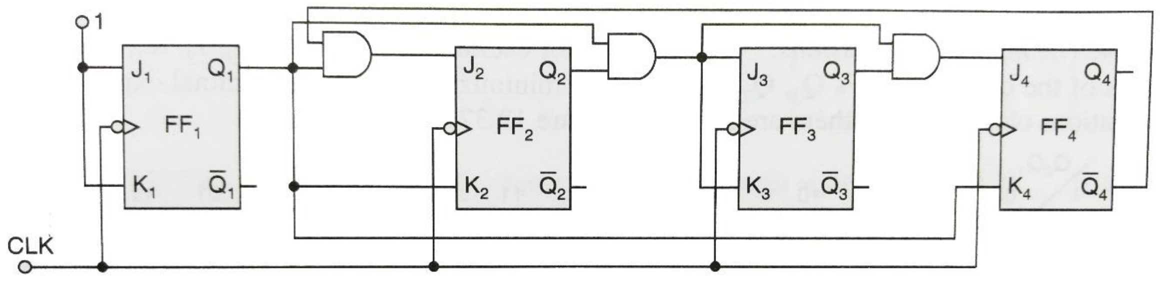
$Q_4Q_3 \backslash Q_2Q_1$	00	01	11	10
00		1	x	x
01		1	x	x
11	x	x	x	x
10			x	x

$J_2 = Q_1Q_4'$

$Q_4Q_3 \backslash Q_2Q_1$	00	01	11	10
00	x	x	1	
01	x	x	1	
11	x	x	x	x
10	x	x	x	x

$K_2 = Q_1$

Step 5. Draw the logic diagram

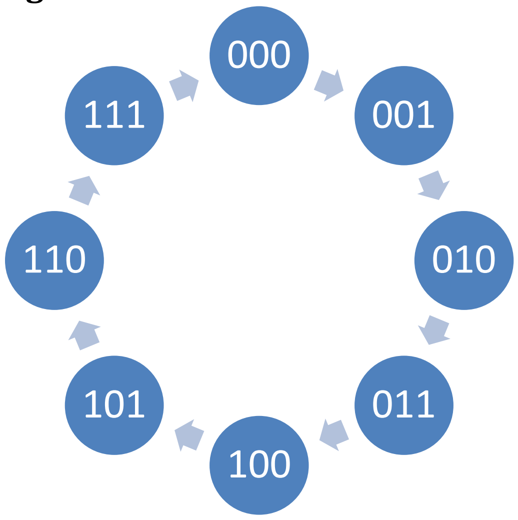


Design a 3-bit synchronous up counter using K-maps and positive edge-triggered JK FFs.

Step 1. Number of flip-flops:

A 3 bit up counter requires 3 flip-flops. The counting sequence is 000, 001, 010, 011, 100, 101, 110, 111, 000 …

Step 2. Draw the state diagram:



Step 3. Select the type of flip-flops and draw the excitation table:

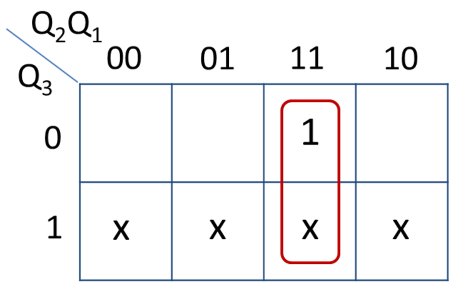
J-K flip-flops are selected and the excitation table of a given sequence counter using J-K flip-flops is shown below.

PS			NS			Required excitations					
Q_3	Q_2	Q_1	Q_3	Q_2	Q_1	J_3	K_3	J_2	K_2	J_1	K_1
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	1	0	1	x	0	0	x	1	x
1	0	1	1	1	0	x	0	1	x	x	1
1	1	0	1	1	1	x	0	x	0	1	x
1	1	1	0	0	0	x	1	x	1	x	1

Step 4. Obtain the minimal expressions

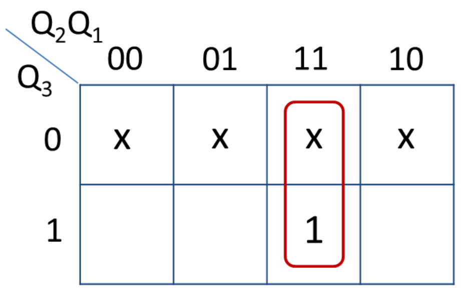
From excitation table, $J_1 = K_1 = 1$.

K – Maps for excitations J_3, K_3, J_2 and K_2 and their minimized form are as follows:



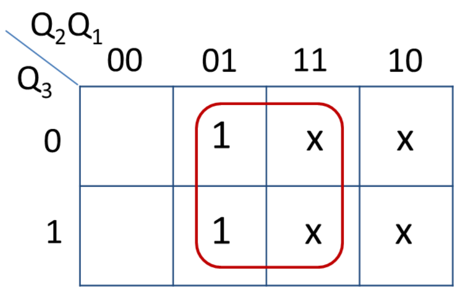
$Q_3 \backslash Q_2Q_1$	00	01	11	10
0			1	
1	x	x	x	x

$J_3 = Q_2Q_1$



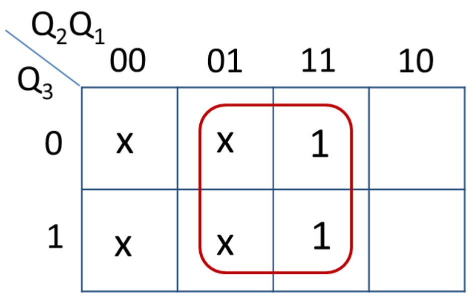
$Q_3 \backslash Q_2Q_1$	00	01	11	10
0	x	x	x	x
1			1	

$K_3 = Q_2Q_1$



$Q_3 \backslash Q_2Q_1$	00	01	11	10
0		1	x	x
1		1	x	x

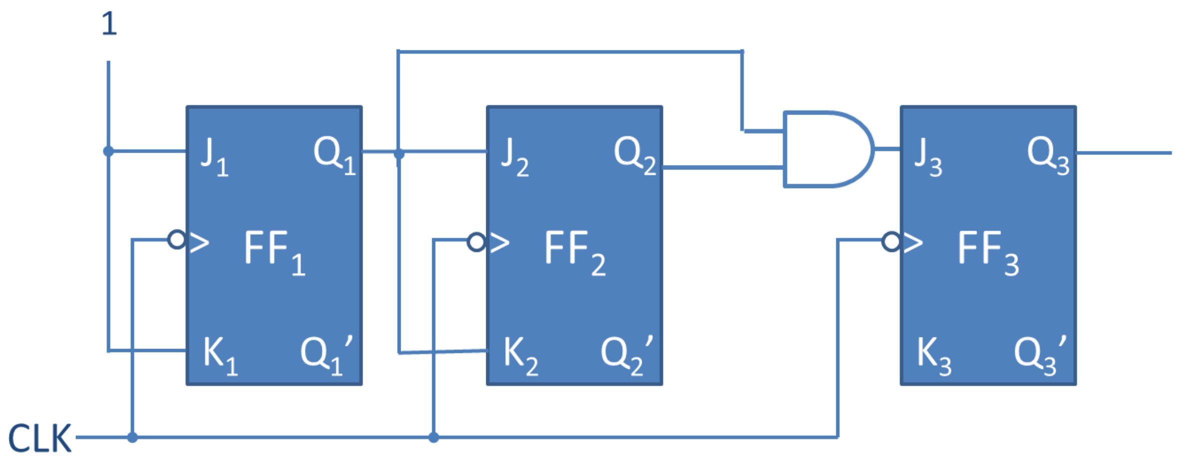
$J_2 = Q_1$



$Q_3 \backslash Q_2Q_1$	00	01	11	10
0	x	x	1	
1	x	x	1	

$K_2 = Q_1$

Step 5. Draw the logic diagram

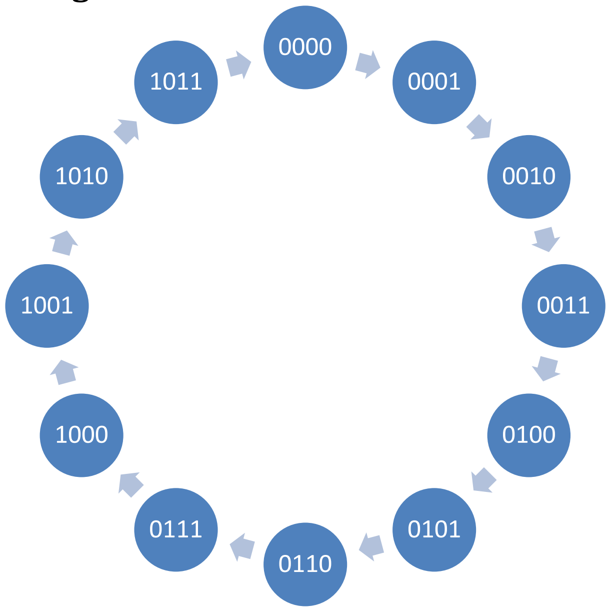


Design a mod-12 Synchronous up counter using D-flipflop.

Step 1. Number of flip-flops:

A mod-12 counter requires 4 flip-flops. The counting sequence is 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 0000 …

Step 2. Draw the state diagram:



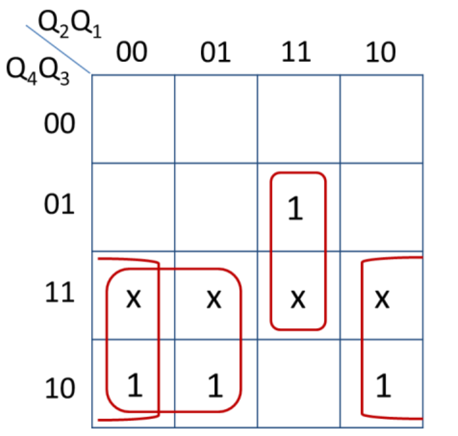
Step 3. Select the type of flip-flops and draw the excitation table:

D flip-flops are selected and the excitation table of a given sequence counter using D flip-flops is shown below.

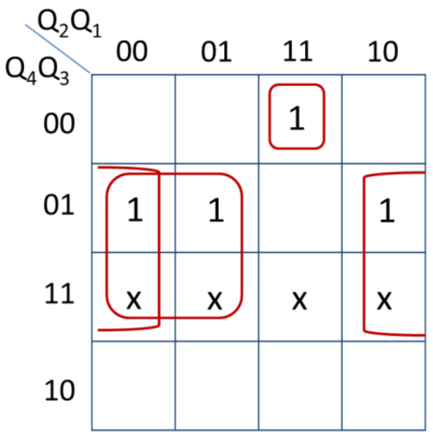
PS				NS				Required excitations			
Q_4	Q_3	Q_2	Q_1	Q_4	Q_3	Q_2	Q_1	D_4	D_3	D_2	D_1
0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	1	0	0	1	0	0	0	1	0
0	0	1	0	0	0	1	1	0	0	1	1
0	0	1	1	0	1	0	0	0	1	0	0
0	1	0	0	0	1	0	1	0	1	0	1
0	1	0	1	0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	1	0	1	1	1
0	1	1	1	1	0	0	0	1	0	0	0
1	0	0	0	1	0	0	1	1	0	0	1
1	0	0	1	1	0	1	0	1	0	1	0
1	0	1	0	1	0	1	1	1	0	1	1
1	0	1	1	0	0	0	0	0	0	0	0

Step 4. Obtain the minimal expressions

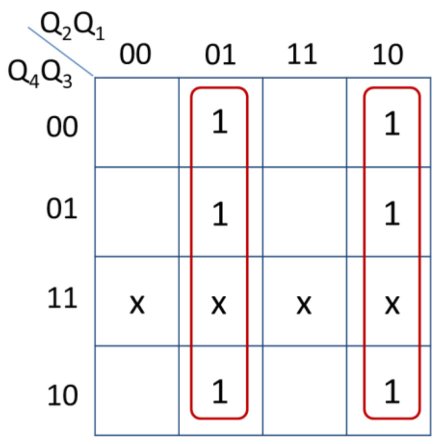
K – Maps for excitations D_4, D_3, D_2, and D_1 and their minimized form are as follows:



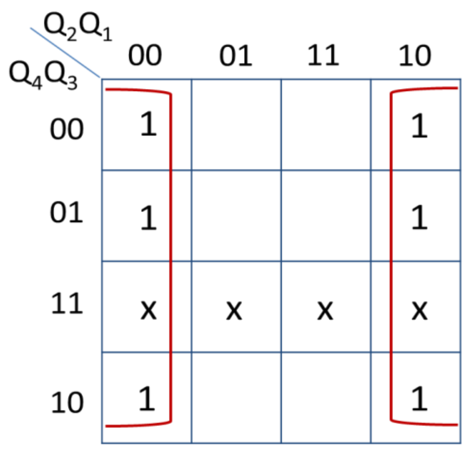
$$D_4 = Q_2'Q_4 + Q_1'Q_4 + Q_1Q_2Q_3$$



$$D_3 = Q_2'Q_3 + Q_1'Q_3 + Q_1Q_2Q_3'Q_4'$$

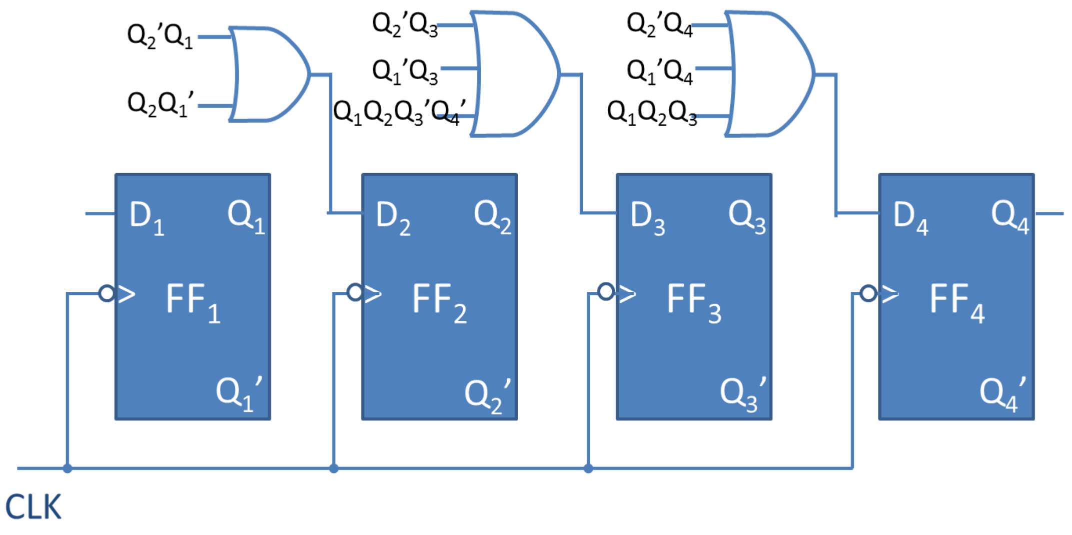


$$D_2 = Q_2'Q_1 + Q_2Q_1'$$



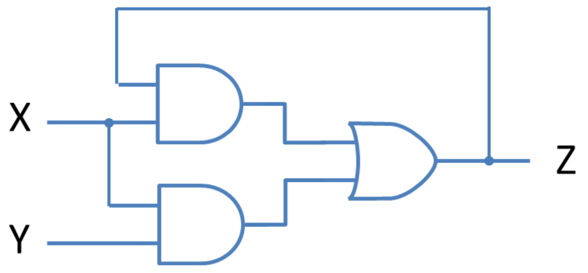
$$D_1 = Q_1'$$

Step 5. Draw the logic diagram

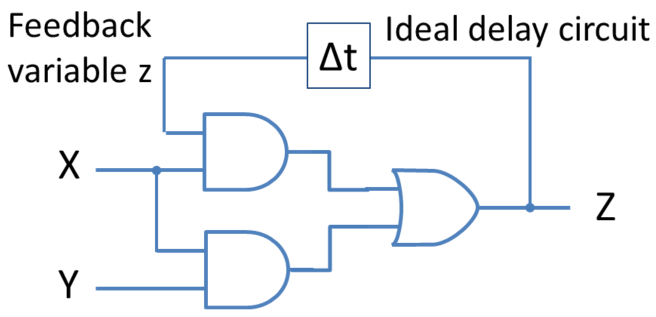

$Q_2'Q_1$
Q_2Q_1'
$Q_2'Q_3$
$Q_1'Q_3$
$Q_1Q_2Q_3'Q_4'$
$Q_2'Q_4$
$Q_1'Q_4$
$Q_1Q_2Q_3$
D_1 Q_1 FF_1 Q_1'
D_2 Q_2 FF_2 Q_2'
D_3 Q_3 FF_3 Q_3'
D_4 Q_4 FF_4 Q_4'
CLK

Fundamental-Mode Model

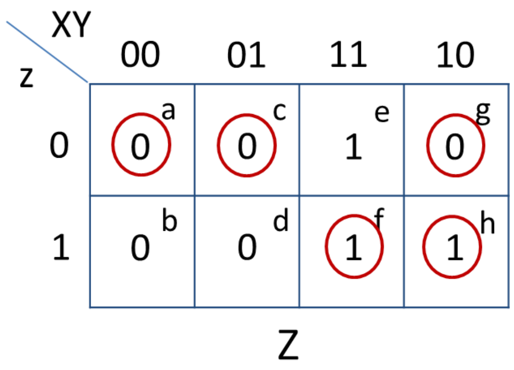
- An asynchronous system uses feedback to produce memory elements as does the synchronous state machine.
- The asynchronous machine generally uses gates rather than flip-flops.
- Figure demonstrates a simple asynchronous circuit.



- X and Y are the system inputs while Z is the system output.
- The signal Z is fed back, however, to a gate input and in this way helps determine its own value.
- When an X or Y change dictates a change in Z, this change occurs only after the cumulative propagation delay time through the gates.
- It is characteristic of asynchronous circuits that the feedback variables along with system inputs determine the values of these same feedback variables.
- An idealized model has been proposed to reflect this behavior. The above circuit is taken for the discussion.



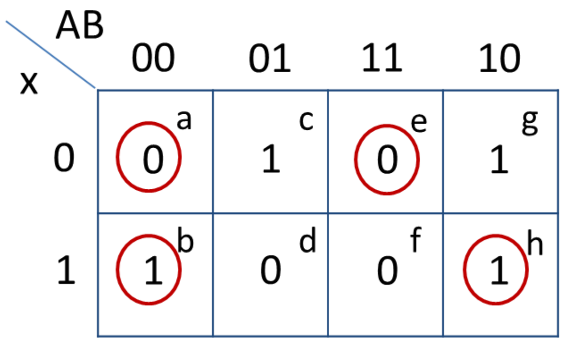
- The gates are considered to have no delay in this model, while the delay element has an output that follows its input after a delay of Δt.
- The variable at the input of the delay element is called the excitation variable, while the feedback variable appears at the output of the delay element.
- In order to characterize the behavior of a circuit, we plot a map of excitation variable as a function of gate inputs.



- It is important to realize that the value Z takes on will also be the value assumed by z after a delay of Δt.
- Thus, the information depicted by the map represents a dynamic situation.
- This can be demonstrated by supposing the system inputs are X = Y = 1 and z = 1, which leads to Z = 1. This is called a stable state since z = Z.
- If X is then changed to 0, the output Z changes to 0 as indicated by the map location corresponding to X = 0, Y = 1, and z = 1.
- This condition will persist for only Δt since z will assume a value of 0 at this time, moving the system to the X = 0, Y = 1, and z = 0 location.
- The location 011 is a transient state, while 010 is a stable state.
- The stable states are normally identified on the map by drawing a circle around the excitation variable such as in figure.

Problems of Asynchronous Circuits

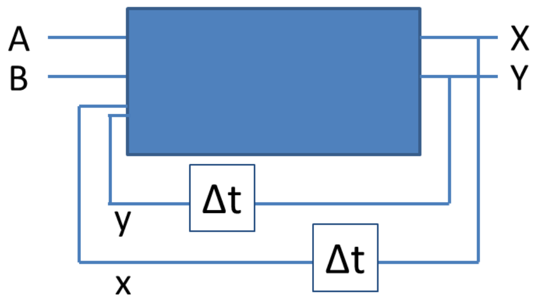
Oscillation Problem



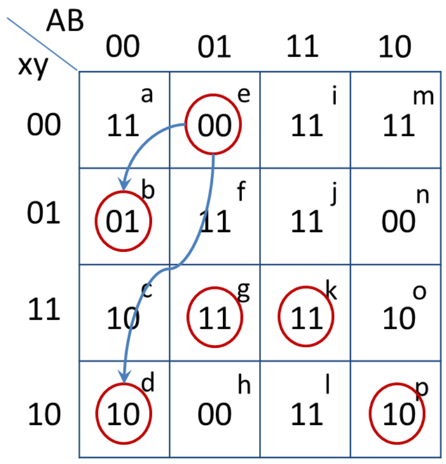
- If the system is in state *a*, a change of input from B = 0 to B = 1 sends the system to state c.
- State c is a transient state, and thus the excitation variable X changes to 1.
- A short time later x changes to 1, moving the system to state d. This state is also a transient state changing X back to 0, followed by a change in x to 0.
- The system now oscillates between states *c* and *d*.
- Of course, this type of situation can be used to advantage in a clock circuit by adding a delaying network to control the delay time Δt to create the desired oscillation frequency.
- In most systems, the oscillation is unacceptable, and the situation depicted by states *c* and *d* of the map must be avoided.

Critical Race

- This situation can occur only when two or more feedback variables are present in the system.



- This system has two external inputs, A and B, and two excitation variables, X and Y, that are fed back to the input of the circuit.



xy \ AB	00	01	11	10
00	11 a	00 e	11 i	11 m
01	01 b	11 f	11 j	00 n
11	10 c	11 g	11 k	10 o
10	10 d	00 h	11 l	10 p

- One critical race occurs if the system starts in state *e* and input B changes from 1 to 0.
- The excitation variables begin to switch from XY = 00 toward XY = 11.
- Due to unequal propagation delays, one of the excitation variables will reach a value of 1 while the other has not changed from a value of 0.
- If the condition XY = 10 is reached, the system moves to stable state d.
- If the condition XY = 01 is reached rather than 10, the system moves to stable state *b*.
- The final stable state reached from this input condition depends on the relative switching speeds of variables X and Y. This situation is referred to as a critical race.

Characteristics of Digital IC

Threshold Voltage
The threshold voltage is defined as that voltage at the input of a gate which causes a change in the state of the output from one logic level to the other.

Propagation Delay
A pulse through a gate takes a certain amount of time to propagate from input to output. This interval of time is known as the propagation delay of the gate.

Power dissipation
The power dissipation of a logic gate is the power required by the gate to operate with 50% duty cycle at a specified frequency and is expressed in mill watts.

Fan-in
The fan-in of a logic gate is defined as the number of inputs that the gate is designed to handle.

Fan-out
The fan-out (loading factor) of a logic gate is defined as the maximum number of standard loads that the output of the gate can drive without impairing its normal operation.

Noise Margin
When the digital circuits operate in noisy environment the gates may malfunction if the noise is beyond certain limits. The noise immunity of a logic circuit refers to the circuit's ability to tolerate noise voltages at its input. A quantitative measure of noise immunity is called noise margin.

Operating temperatures
The IC gates and other circuits are temperature sensitive being semiconductor devices. However they are designed to operate satisfactorily over a specified range of temperatures. The range specified for commercial applications is 0 to 70°C, for industrial it is 0 to 85°C and for military applications it is -55°C to 125°C.

Speed Power Product
A common means for measuring and comparing the overall performance of an IC family is the speed power product which is obtained by multiplying the gate propagation delay by the gate power dissipation. The smaller the product, the better the overall performance.

Transistor-Transistor Logic (TTL)

- The TTL is so named because of its independence on transistors alone to perform basic logic operations.
- The TTL uses transistors operating in saturated mode.
- It is the fastest of saturated logic families.
- The basic TTL logic circuit is the NAND gate.
- Good speed, low manufacturing cost, wide range of circuits and the availability in SSI and MSI are its advantages.
- Tight V_{CC} tolerance, relatively high power consumption, moderate packing density, generation of noise spikes and susceptibility to power transients are its disadvantages.
- TTL logic family consists of several subfamilies such as:

  - Standard TTL
  - High Speed TTL
  - Low Power TTL
  - Schottky TTL
  - Low Power Schottky TTL
  - Advanced Schottky TTL
  - Advanced Low Power Schottky TTL
  - F (Fast) TTL
- For standard TTL,
  - Propagation Delay time = 9 ns
  - Power dissipation per = 10 mW
  - Noise Margin = 0.4 mV
  - Fan-in = 8
  - Fan-out = 10
  - Logic '0' = 0 V to 0.8 V
  - Logic '1' = 2 V to 5 V
  - Indeterminate Range = 0.8 V to 2 V

Emitter-Coupled Logic (ECL)

- This logic family is also called Current Mode Logic or Current Steering Logic.
- It is the fastest of all logic families.
- ECL operates on the principle of current switching, whereby a fixed bias current less than IC switched from one transistor's collector to another.
- Because of this mode operation, this logic form is also referred to as Current Mode Logic (CML).
- It is also called Current Steering Logic (CSL), because current is steered from one device to another.
- The ECL family is used in very high frequency applications where its speed is superior.
- The important characteristics of ECL are:
  - Transistors never saturate. So, speed is high
  - Logic Levels are negative, -0.9 V for Logic 1 and -1.7 V for Logic 0.
  - Noise Margin is less, about 250 mV. This makes ECL, unreliable for use in heavy industrial environment.
  - ECL circuits produce the output and its complement, and therefore, eliminate the need for inverters.
  - Fan-out is large because the output impedance is low. It is about 25.
  - Power dissipation per gate is large.
  - The total current flow in ECL is more or less constant. So, no noise spikes will be internally generated.

MOSFET

- MOSFET family are simpler & inexpensive to fabricate, require much less power, have better noise margin, a greater supply voltage range, a higher fan-out and require much less chip area as compared to other bipolar logic families.
- For MOS logic,
  - Propagation Delay, t_{pd} = 50 ns.
  - Noise Margin, NM = 1.5 V.

- Power Dissipation, PD = 0.1 Mw.
- Fan out = 50 for frequencies greater than 100 Hz and it is virtually unlimited for dc or low frequencies.
- The propagation delay associated with MOS gates is large (50 ns) because of their high output resistance (100 k) and capacitive loading presented by the driven gates.

- There are three types of MOSFET:
  - P-channel MOSFET (PMOS)
    - Enhancement Type PMOS
    - Depletion Type PMOS
  - N-channel MOSFET (NMOS)
    - Enhancement Type NMOS
    - Depletion Type NMOS
  - Complementary MOSFET (CMOS)

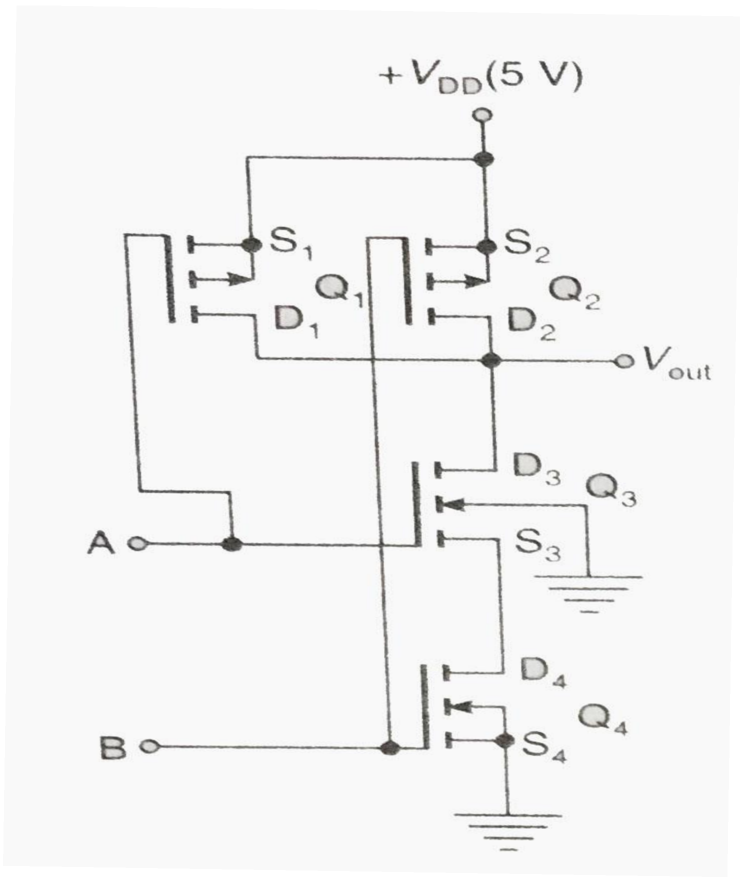
Compare TTL, ECL, & CMOS logic families.

Characteristic	**TTL**	**CMSO**	**ECL**
Power Input	Moderate	Low	Moderate-High
Frequency limit	High	Moderate	Very high
Circuit density	Moderate-high	High-very high	Moderate
Circuit types per family	High	High	Moderate

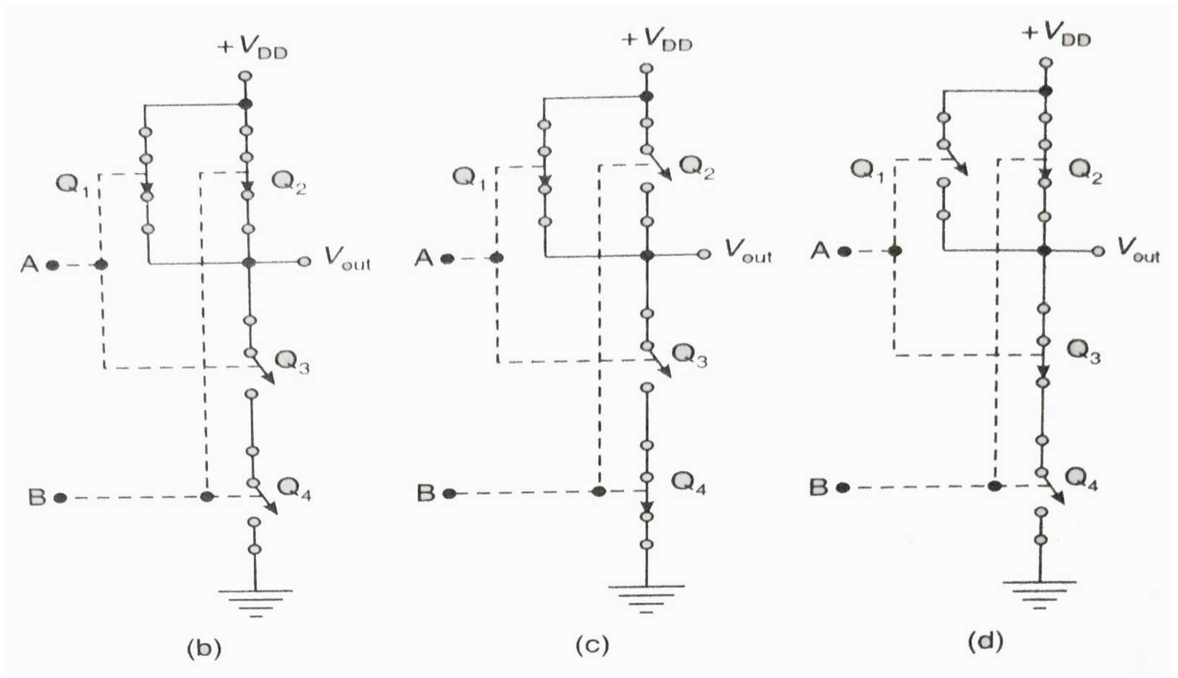
Logic Family	**Propagation delay time (ns)**	**Power dissipation per gate (mW)**	**Noise Margin (V)**	**Fan-in**	**Fan-out**	**Cost**
TTL	9	10	0.4	8	10	Low
CMOS	<50	0.01	5	10	50	Low
ECL	1	50	0.25	5	10	High

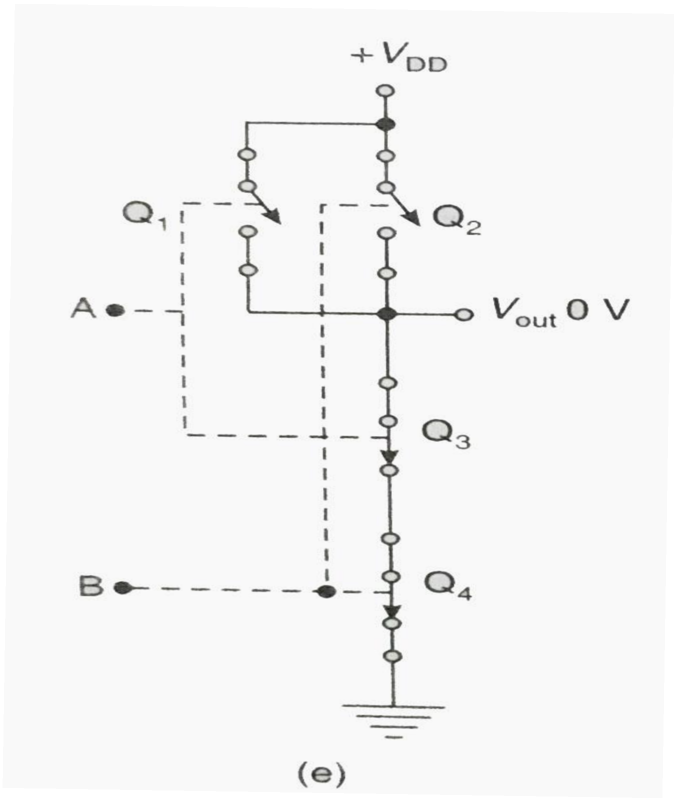
CMOS NAND Gate

- Figure shows a CMOS two-input NAND gate and its equivalent circuits for various input combinations.



- Here Q_1 and Q_2 are parallel-connected PMOS transistors, and Q_3 and Q_4 are series-connected NMOS transistors.





- When A = 0 V and B = 0 V, $V_{GS1} = V_{GS2} = -5$ V, $V_{GS3} = V_{GS4} = 0$ V. So Q_1 is ON, Q_3 is OFF, Q_2 is ON and Q_4 is OFF. Thus, the switching circuit (b) results with $V_{out} = +5$ V.
- When A = 0 V and B = +5 V, $V_{GS1} = -5$ V, $V_{GS2} = 0$ V, $V_{GS3} = 0$ V, $V_{GS4} = 5$ V. So Q_1 is ON, Q_3 is OFF, Q_2 is OFF and Q_4 is ON. Thus, the switching circuit (c) results with $V_{out} = +5$ V.
- When A = +5 V and B = 0 V, $V_{GS1} = 0$ V, $V_{GS2} = -5$ V, $V_{GS3} = 5$ V, $V_{GS4} = 0$ V. So Q_1 is OFF, Q_3 is ON, Q_2 is ON and Q_4 is OFF. Thus, the switching circuit (d) results with $V_{out} = +5$ V.
- When A = +5 V and B = +5 V, $V_{GS1} = V_{GS2} = 0$ V, $V_{GS3} = V_{GS4} = 5$ V. So Q_1 is OFF, Q_3 is ON, Q_2 is OFF and Q_4 is ON. Thus, the switching circuit (e) results with $V_{out} = 0$ V.

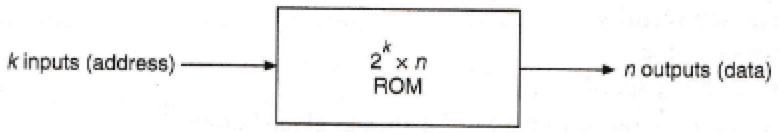
Introduction to Programmable Logic Devices

- A programmable logic device is an IC that is user configurable and is capable of implementing logic functions.
- It is an LSI chip that contains a 'regular' structure and allows the designer to customize it for any specific application, i.e. it is programmed by the user to perform a function required for his application.
- A PLD contains a large number of gates, flip- flops, and registers that are interconnected on the chip. Many of the connections, however, are fusible links that can be broken.
- The IC is said to be programmable because the specific function of the IC for a given application is determined by the selective breaking of some of the interconnections while leaving others intact.
- The 'fuse blowing' process can be done either by the manufacturer in accordance with the customer's instructions, or by the customer himself which is called 'programming' because it produces the desired circuit pattern interconnecting the gates, flip-flops, registers, and so on.
- PLDs can be reprogrammed in a few seconds and hence give more flexibility to experiment with designs.
- The advantages of PLDs over fixed function ICs are as follows:
    1. Low development cost
    2. Less space requirement
    3. Less power requirement
    4. High reliability
    5. Easy circuit testing
    6. Easy design modification
    7. High design security
    8. Less design time
    9. High switching speed

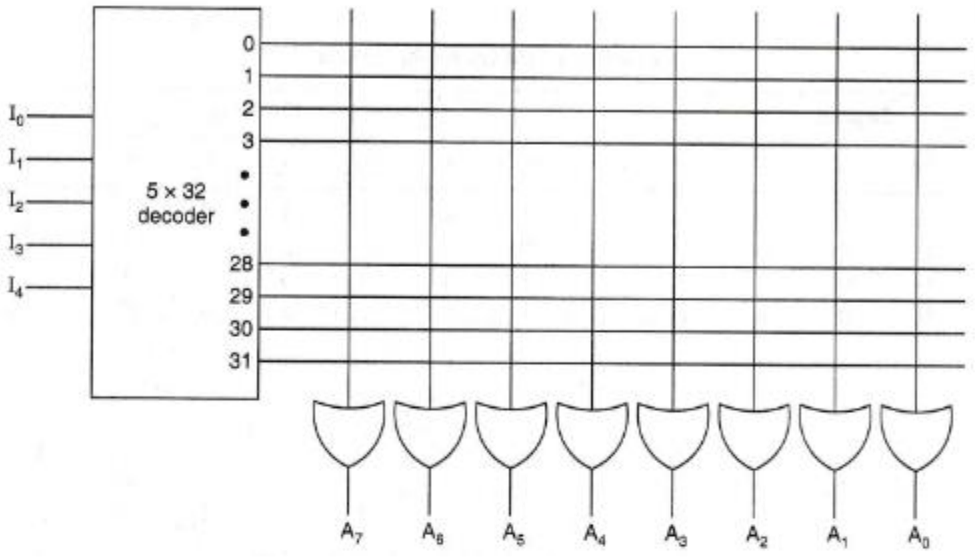
Read Only Memory (ROM)

- A read-only memory (ROM) is essentially a memory device in which permanent binary information is stored.
- Once the pattern is established, it stays within the unit when the power is turned off and on again.
- ROMs are used to store information which is of fixed type, such as tables for various functions, fixed data and instructions.
- The advantages of using a ROM as a PLD are the following:
    1. Ease of design since no simplification or minimization of logic function is required.
    2. Designs can be changed, modified rapidly.
    3. It is usually faster than discrete MSI/SSI circuit.
    4. Cost is reduced.
- There are a few disadvantages also of ROM based circuits, such as:
    1. Non-utilization of complete circuit
    2. Increased power requirement
    3. Enormous increase in size with increase in the number of input variables making it impractical

ROM Organization



- A block diagram of a ROM is shown in the figure. It consists of k inputs and n outputs.
- The inputs provide the address for the memory and the outputs give the data bits of the stored word which is selected by the address.
- The number of words in a ROM is determined from the fact that *k* address input lines are needed to specify *2k* words.
- Consider, for example, a 32 x 8 ROM. The unit consists of 32 words of 8 bits each.



- There are five input lines that form the binary numbers from 0 through 31 for the address.
- The five inputs are decoded into 32 distinct outputs by means of a 5 x 32 decoder.
- ROM is basically a decoder with *k* inputs and *2k* output lines followed by a bank of OR gates.
- Each output of the decoder represents a memory address.
- The 32 outputs of the decoder are connected to each of the 8 OR gates.
- Each OR gate must be considered as having 32 inputs.
- Since each OR gate has 32 input connections and there are 8 OR gates, the ROM contains 32 x 8 = 256 internal connections.
- In general, a *2k* x *n* ROM will have an internal *k* x *2k* decoder and *n* OR gates.
- Each OR gate has 2k inputs, which are connected to each of the outputs of the decoder.

Types of ROM

Maskable programmable read-only memory (MROM)

In this type of read-only memory, the user specifies the data to be stored to the manufacturer of the memory. The data pattern specified by the user are programmed as a part of the fabrication process. Once programmed, the data pattern can never be changed. This type of read-only memory is referred to as ROM. ROMs are highly suited for very high volume usage due to their low cost.

Programmable read-only memory (PROM)

This type of memory comes from the manufacturer without any data stored in it, i.e. empty. The data pattern is programmed electrically by the user using a special circuit known as PROM programmer. It can be programmed only once during its life time. Once programmed, the data cannot be altered. This type of memory is known as PROM. These are highly suited for high volume usage due to their low cost of production.

Erasable programmable read-only memory (EPROM)

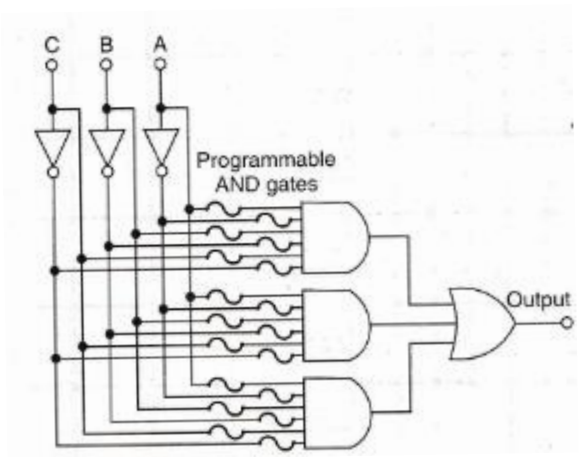
In this type of memory, data can be written any number of times, i.e. they are reprogrammable. Before it is reprogrammed, the contents already stored are erased by exposing the chip to ultraviolet radiation for about 30 minutes. This type of memory is referred to as EPROM. EPROMs are possible only in MOS technology. Programming is done using a PROM programmer.

Electrically erasable and programmable read-only memory (EEPROM or E2PROM)

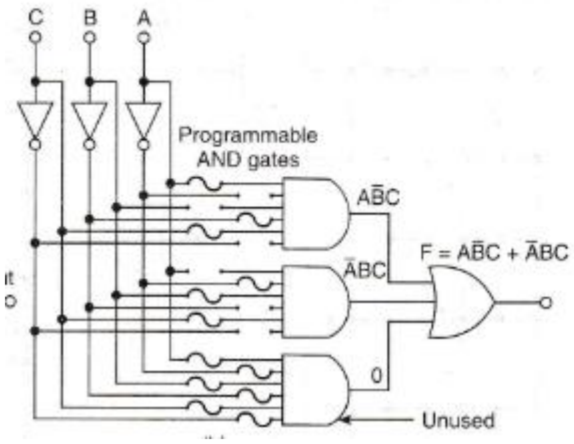
This is another type of reprogrammable memory in which erasing is done electrically rather than exposing the chip to the ultraviolet radiation. It is referred to as EEPROM or electrically alterable ROM (EAROM).

PROGRAMMABLE ARRAY LOGIC (PAL)

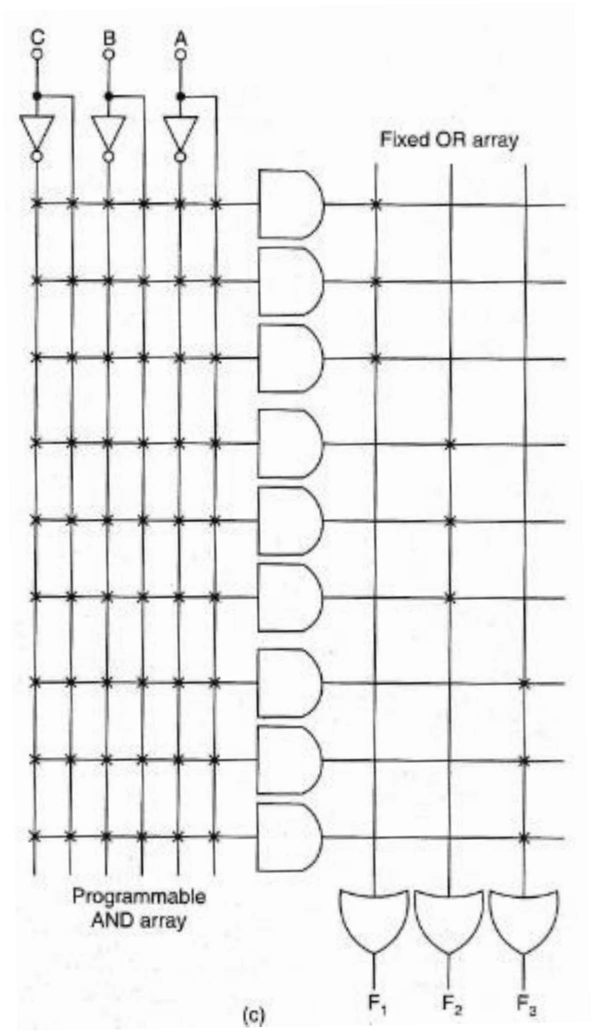
- Programmable array logic (a registered trade mark of Monolithic Memories) is a particular family of programmable logic devices (PLDs) that is widely used and available from a number of manufacturers.
- The PAL circuits consist of a set of AND gates whose inputs can be programmed and whose outputs are connected to an OR gate, i.e. the inputs to the OR gate are hard-wired, i.e. PAL is a PLD with a fixed OR array and a programmable AND array.



- Figure shows a small example of the basic structure. The fuse symbols represent fusible links that can be burned open using equipment similar to a PROM programmer.



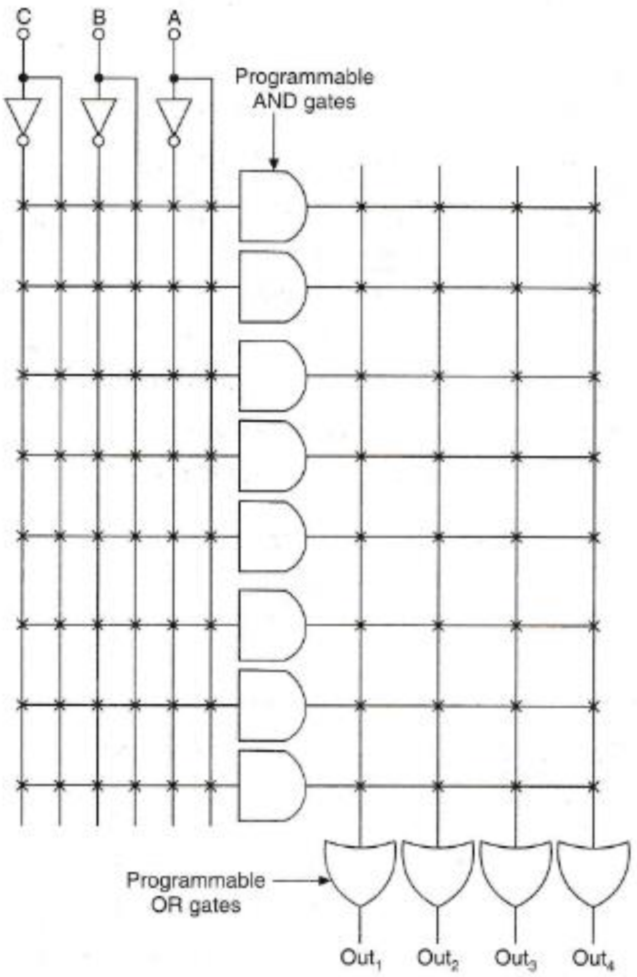
- Figure shows how the circuit is programmed to implement F = A’BC + AB’C.
- All input variables and their complements are left connected to the unused AND gate, whose output is, therefore, AA’BB’CC’ = 0. The 0 has no effect on the output of the OR gate.
- The actual PAL circuits have several groups of AND gates, each group providing inputs to separate OR gates.



- Figure shows an example of how the PAL structure is represented using the abbreviated connections. It is a 3-input 3-wide AND-OR structure.
- Each function can have three minterms or product terms.
- Inputs to the OR gates at the outputs are fixed as shown by 'x' marked on the vertical lines.
- The inputs to the AND gates are marked on the corresponding line by the 'x'.
- Removing the 'x' implies blowing off the corresponding fuse which in turn implies that the corresponding input variable is not applied to the particular AND gate.

Programmable Logic Array (PLA)

- The PLA combines the characteristics of the PROM and the PAL by providing both a programmable OR array and a programmable AND array, i.e. in a PLA both AND gates and OR gates have fuses at the inputs.
- A third set of fuses in the output inverters allows the output function to be inverted if required. Usually X-OR gates are used for controlled inversion.
- However, it has some disadvantages. Because it has two sets of fuses, it is more difficult to manufacture, program and test it than a PROM or a PAL.



- Figure demonstrates the structure of a three-input, four-output PLA with every fusible link intact.
- Like ROM, PLA can be mask programmable or field programmable.

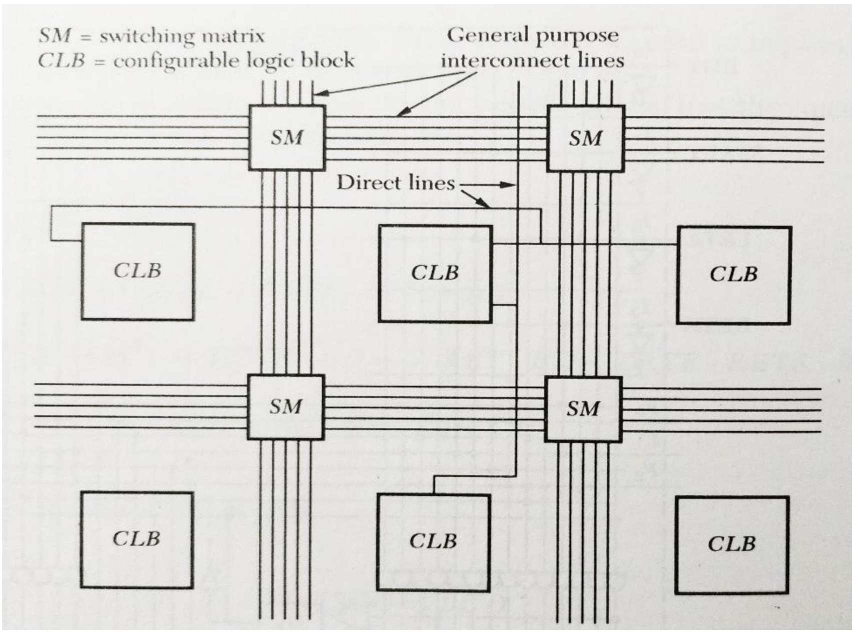
- With a mask programmable PLA, the user must submit a PLA programming table to the manufacturer.
- This table is used by the vender to produce a user made PLA that has the required internal paths between inputs and outputs.
- A second type of PLA available is called a field programmable logic array or FPLA.
- The FPLA can be programmed by the user by means of certain recommended procedures. FPLAs can be programmed with commercially available programmer units.

Compare ROM, PAL and PLA

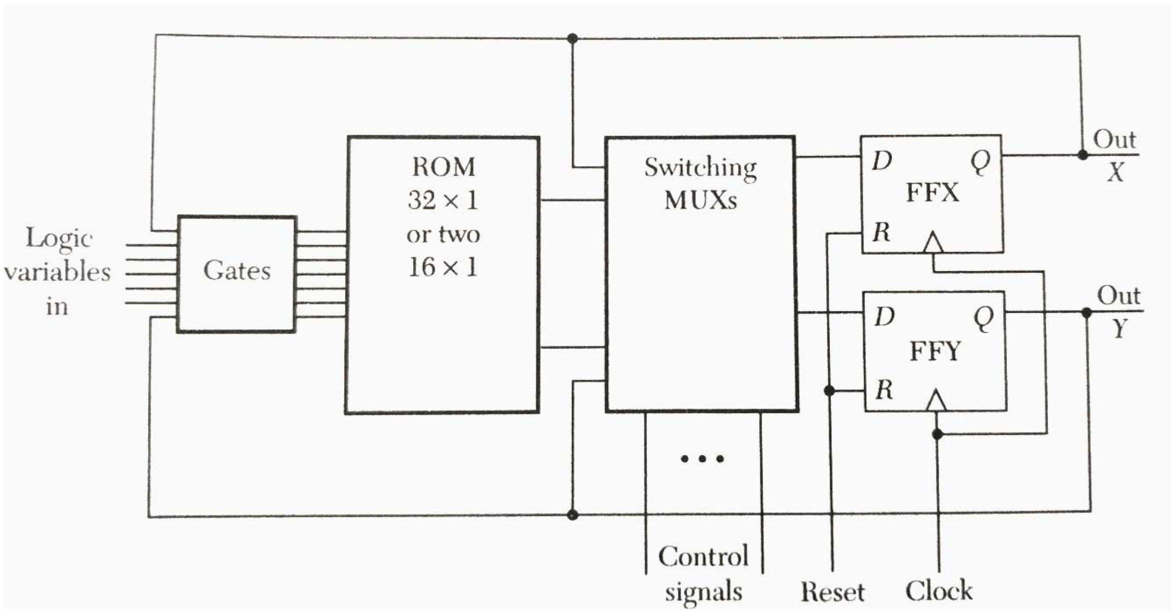
ROM	PAL	PLA
Consist of fixed AND gate array and programmable OR array.	Consist of programmable AND gate array and programmable OR array.	Consist of programmable AND gate array and programmable OR array.
Medium speed	High speed	Slow
Cheap	Intermediate cost	Most expensive
Not flexible	Not flexible	Offering maximum programming flexibility
It is possible to decode any minterms	We can get any desired minterms by programming the AND matrix	We can get any desired minterms by programming the AND & OR matrix
SOP function in the standard form only can be implemented	Any SOP function can be implemented	Any SOP function can be implemented

Xilinx FPGA

- The basic architecture is shown in figure. The logic modules have inputs and outputs that can be connected to metal lines by programmable switches.


SM = switching matrix
CLB = configurable logic block
General purpose interconnect lines
Direct lines
SM
SM
SM
SM
CLB
CLB
CLB
CLB
CLB
CLB

- The direct lines allow signals to be sent to or received from adjacent logic modules.
- These direct lines can also be programmed to connect to the general purpose interconnect lines to allow interconnection of nonadjacent logic modules if required.
- In addition, signals can be switched from one path to another at the intersections of rows and columns of the general purpose interconnect lines.
- A simplified diagram of the configurable logic block or CLB is shown in figure.



- This block is similar to a two flip-flop, RAM -controlled state machine.
- The combinational logic is performed by a RAM that can be used as a 32 x 1 or two 16 x 1 RAMs.