DIGITAL LOGIC AND COMPUTER ORGANIZATION

II B.TECH I SEMESTER - CSE (AR 23)

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

LENDI INSTITUTE OF ENGINEERING AND TECHNOLOGY

(An Autonomous Institute, Approved by AICTE & Permanently Affiliated to JNTU-GV, Vizianagaram)

(Accredited By NAAC with A Grade and Accredited by NBA)

Jonnada (Village), Denkada (Mandal), Vizianagaram District – 535 005

Phone No. 08922-241111, 241112

E-Mail: lendi_2008@yahoo.com website: www.lendi.org

UNIT – II

Combinational circuits-II: Code Converters, Encoders, Decoders, Multiplexers, De-multiplexers.

Sequential logic circuits: Flip-Flops (SR, JK, T, D), Excitation tables, conversion of Flip Flops, Synchronous and Asynchronous counters, Up/Down counters, Modulus Counters Registers: Bi-directional, Universal Shift Register.

INDEX

UNIT – II
Combinational circuits-II

CODE CONVERTERS

The availability of a large variety of codes for the same discrete elements of information results in the use of different codes by different digital systems. It is sometimes necessary to use the output of one system as the input to another. A conversion circuit must be inserted between the two systems if each uses different codes for the same information. Thus, **a code converter is a circuit that makes the two systems compatible even though each uses a different binary code.**

To convert from binary code A to binary code B, the input lines must supply the bit combination of elements as specified by code A and the output lines must generate the corresponding bit combination of code B. A combinational circuit performs this transformation by means of logic gates.

BINARY CODED DECIMAL (BCD) TO THE EXCESS-3 CONVERTER

The design procedure that converts binary coded decimal (BCD) to the excess-3 code for the decimal digits is illustrated here. Since each code uses four bits to represent a decimal digit, there must be four input variables and four output variables. We designate the four input binary variables by the symbols *A, B, C,* and *D,* and the four output variables by *w, x, y* , and *z* . The truth table relating the input and output variables is shown below. Note that four binary variables may have 16 bit combinations, but only 10 are listed in the truth table. The six bit combinations not listed for the input variables are don't-care combinations. These values have no meaning in BCD and we assume that they will never occur in actual operation of the circuit. Therefore, we are at liberty to assign to the output variables either a 1 or a 0, whichever gives a simpler circuit.

Table: Truth table for BCD to Excess-3 converter

Input BCD				Output Excess-3 Code			
A	*B*	*C*	*D*	*w*	*x*	*y*	*z*
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 maps below are plotted to obtain simplified Boolean functions for the outputs. Each one of the four maps represents one of the four outputs of the circuit as a function of the four input variables. The 1's marked inside the squares are obtained from the minterms that make the output equal to 1. The 1's are obtained from the truth table by going over the output columns one at a time. For example, the column under output *z* has five 1's; therefore, the map for *z* has five 1's, each being in a square corresponding to the minterm that makes *z* equal to 1. The six don't-care minterms 10 through 15 are marked with an *X*. One possible way to simplify the functions into sum-of-products form is listed under the map of each variable.

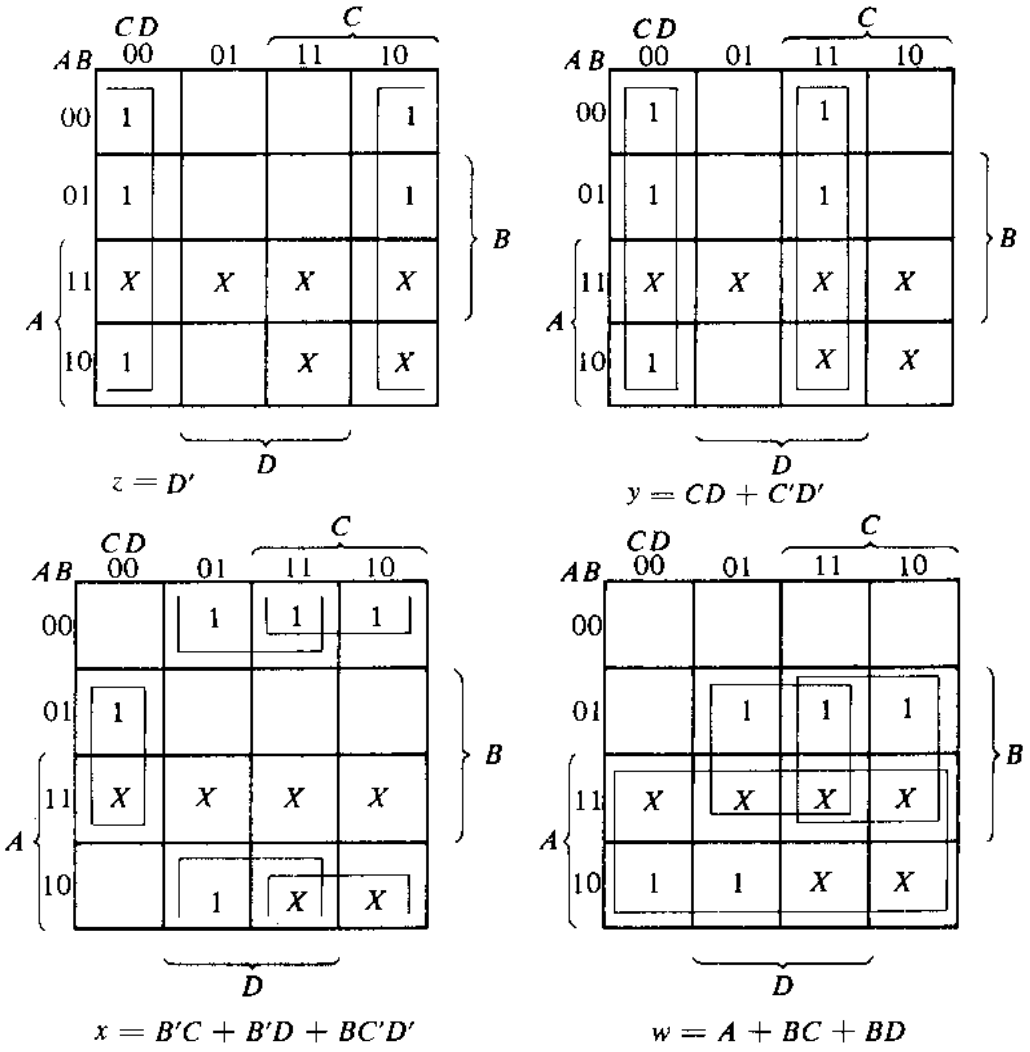


Figure: Maps for BCD to Excess-3 converter

A two-level logic diagram for each output may be obtained directly from the Boolean expressions derived from the maps. There are various other possibilities for a logic diagram that implements this circuit. The expressions obtained above may be manipulated algebraically for the purpose of using common gates for two or more outputs. This manipulation, shown next, illustrates the flexibility obtained with multiple-output systems when implemented with three or more levels of gates:

z = D'
y = CD + C'D' = CD + (C + D)'
x = B'C + B'D + BC'D' = B'(C + D) + BC'D'
= B'(C + D) + B(C + D)'
w = A + BC + BD = A + B(C + D)

The logic diagram that implements these expressions is shown below. Note that the OR gate whose output is *C + D* has been used to implement partially each of three outputs. Not counting input inverters, the implementation in sum-of-products form requires seven AND gates and three OR gates. The implementation of logic diagram requires four AND gates, four OR gates, and one inverter. If only the normal inputs are available, the first implementation will require inverters for variables *B, C,* and *D,* and the second implementation will require

inverters for variables *B* and *D*. Thus, the three-level logic circuit requires fewer gates, all of which in turn require no more than two inputs.

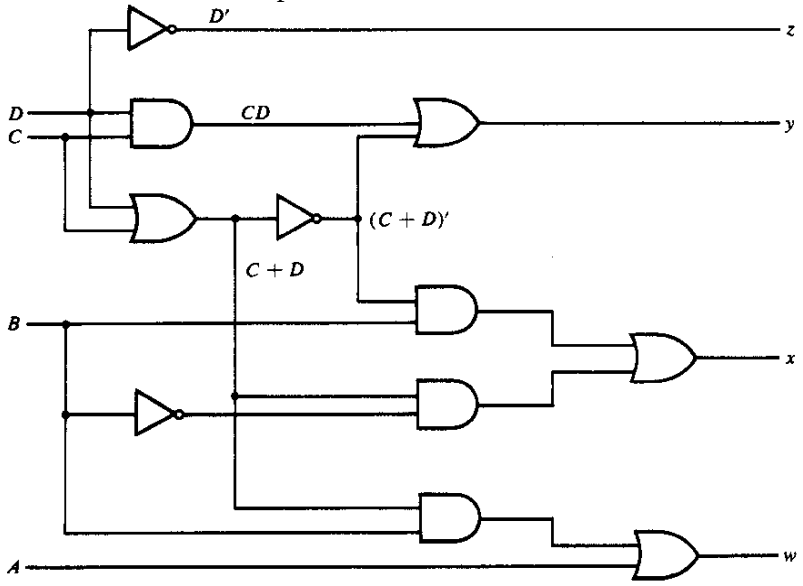


Figure: Logic diagram for BCD to Excess-3 converter

Example: Design and implement 4 bit binary to Gray converter.
Solution:

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

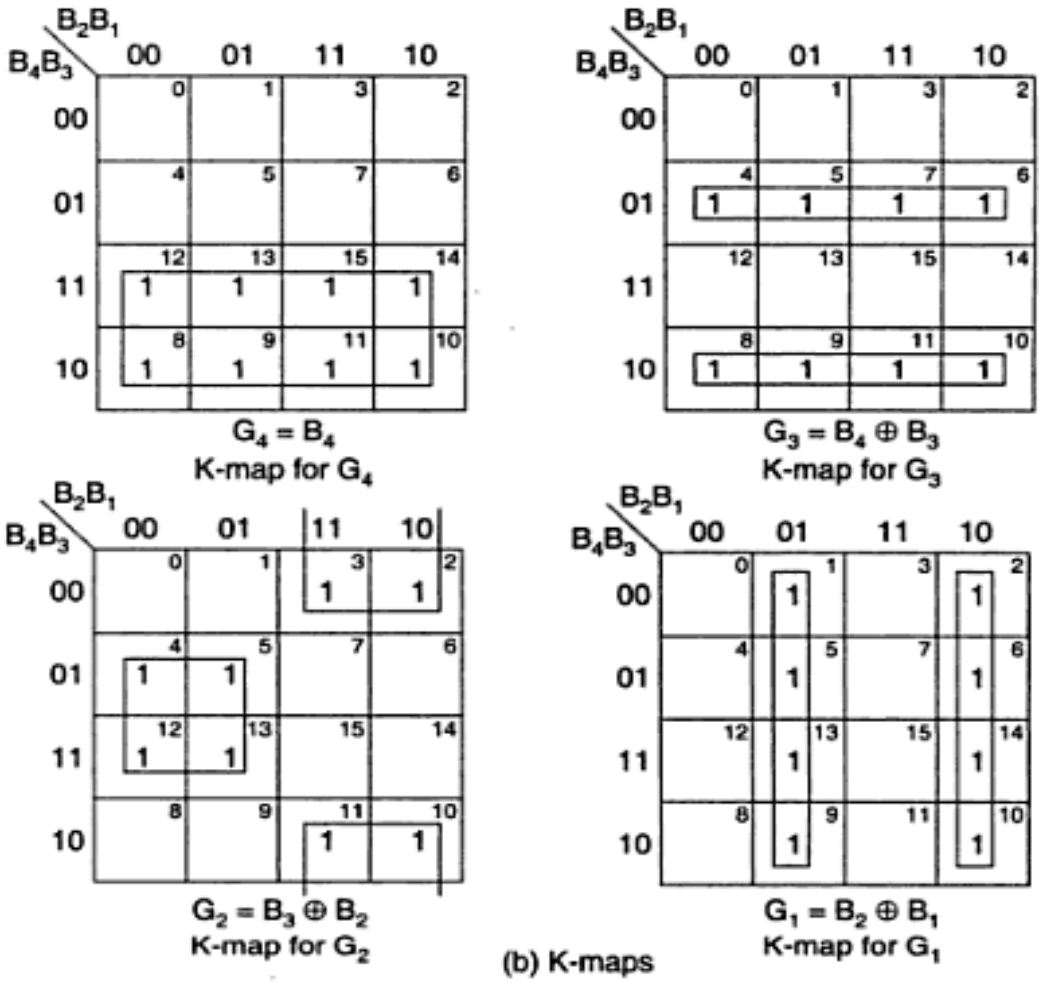
(a) Conversion table

$G_4 = \Sigma m(8, 9, 10, 11, 12, 13, 14, 15)$

$G_3 = \Sigma m(4, 5, 6, 7, 8, 9, 10, 11)$

$G_2 = \Sigma m(2, 3, 4, 5, 10, 11, 12, 13)$

$G_1 = \Sigma m(1, 2, 5, 6, 9, 10, 13, 14)$

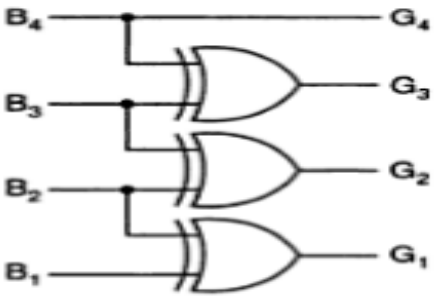


(b) K-maps

4-bit binary-to-Gray code converter.

$$G_4 = B_4$$
$$G_3 = \bar{B}_4 B_3 + B_4 \bar{B}_3 = B_4 \oplus B_3$$
$$G_2 = \bar{B}_3 B_2 + B_3 \bar{B}_2 = B_3 \oplus B_2$$
$$G_1 = \bar{B}_2 B_1 + B_2 \bar{B}_1 = B_2 \oplus B_1$$

Logic Equations



Logic diagram

DECODERS

Discrete quantities of information are represented in digital systems by binary codes. A binary code of n bits is capable of representing up to 2^n distinct elements of coded information. **A decoder is a combinational circuit that converts binary information from n input lines to a maximum of 2^n unique output lines.** If the n -bit coded information has unused combinations, the decoder may have fewer than 2^n outputs.

The decoders presented here are called **n -to- m -line decoders, where $m \leq 2^n$**. Their purpose is to generate the 2^n (or fewer) minterms of n input variables. Each combination of

inputs will assert a unique output. The name decoder is also used in conjunction with other code converters, such as a BCD-to-seven-segment decoder.

As an example, consider **the three-to-eight-line decoder circuit** below. The three inputs are decoded into eight outputs, each representing one of the minterms of the three input variables. The three inverters provide the complement of the inputs, and each one of the eight AND gates generates one of the minterms. **A particular application of this decoder is binary-to-octal conversion.** The input variables represent a binary number, and the outputs represent the eight digits of a number in the octal number system. However, a three-to-eight-line decoder can be used for decoding any three-bit code to provide eight outputs, one for each element of the code. The operation of the decoder may be clarified by the truth table listed below. For each possible input combination, there are seven outputs that are equal to 0 and only one that is equal to 1. The output whose value is equal to 1 represents the minterm equivalent of the binary number currently available in the input lines.

Table: Truth table of 3 to 8 line decoder

Inputs			Outputs							
x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

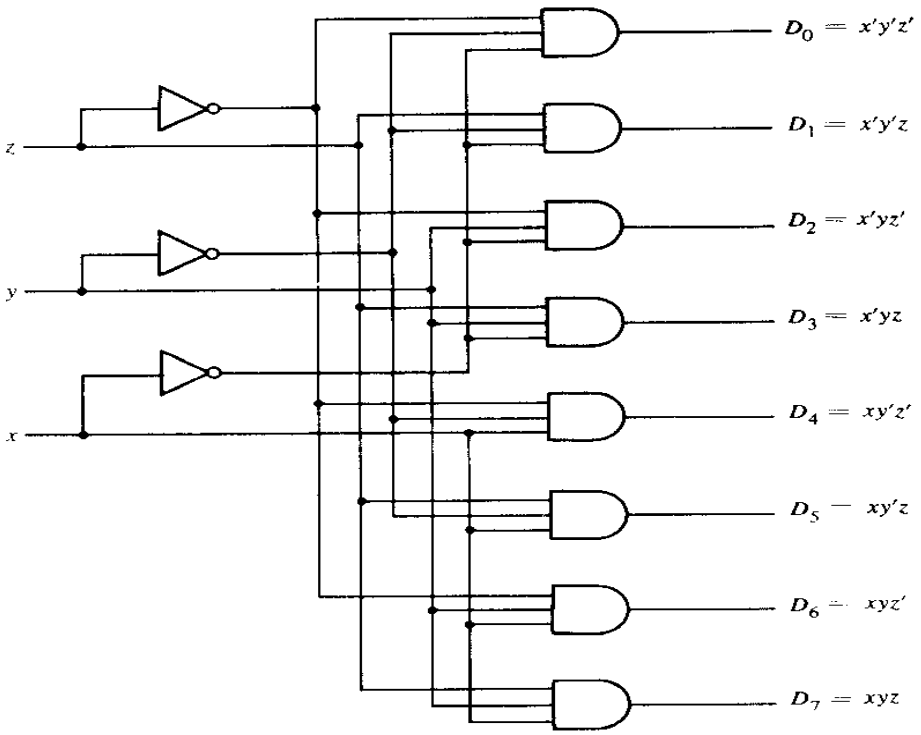


Figure: 3 to 8 line decoder

Example: Implement full adder circuit with a decoder and two OR gates.
Solution: From the truth table of the full adder, we obtain the functions for the combinational circuit in sum-of-minterms form:

$$S(x, y, z) = \Sigma (1, 2, 4, 7)$$
$$C(x, y, z) = \Sigma (3, 5, 6, 7)$$

Since there are three inputs and a total of eight minterms, we need a three-to-eight-line decoder. The implementation is shown below. The decoder generates the eight minterms for x, y, and z. The OR gate for output S forms the logical sum of minterms 1, 2, 4, and 7. The OR gate for output C forms the logical sum of minterms 3, 5, 6, and 7.

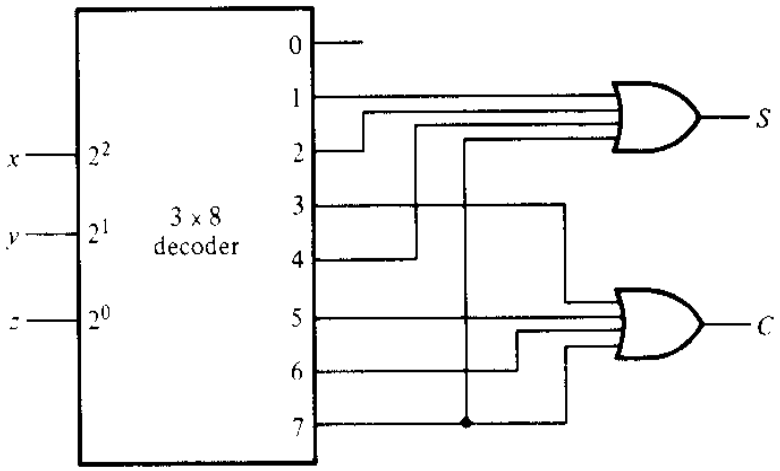
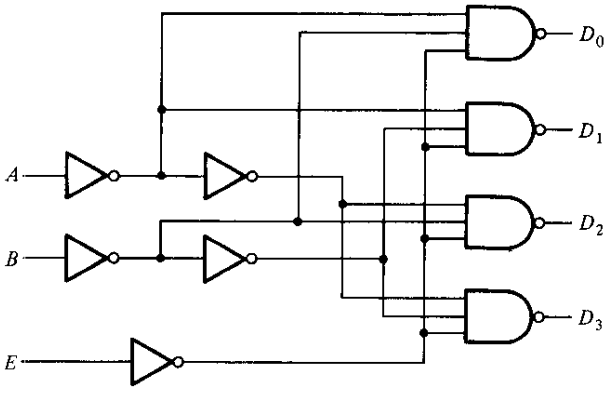


Figure: Implementation of full adder with a decoder

DE-MULTIPLEXERS.

Some decoders are constructed with NAND gates. Since a **NAND gate** produces the AND operation with an inverted output, it becomes more economical to **generate the decoder minterms in their complemented form**. Furthermore, **decoders include one or more enable inputs to control the circuit operation.** A two-to-four-line decoder with an enable input constructed with NAND gates is shown below.



(a) Logic diagram

E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

(b) Truth table

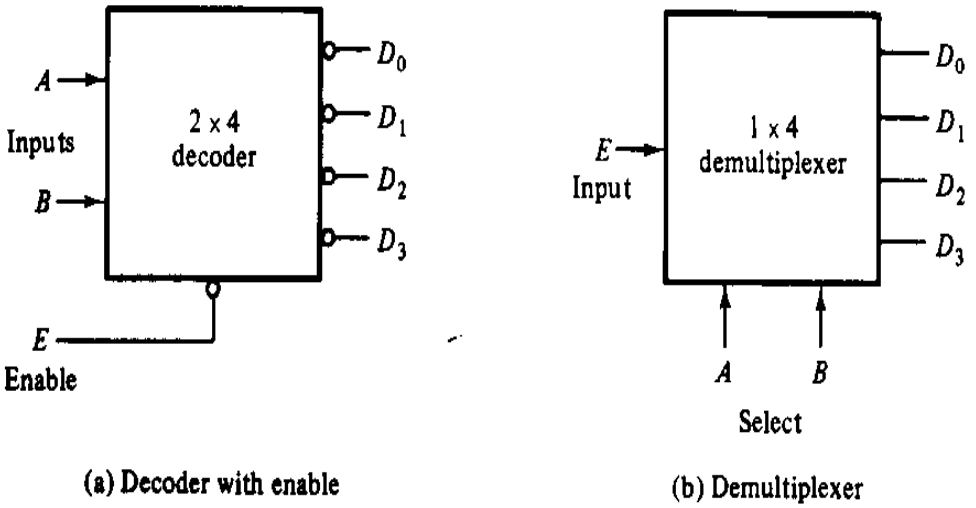
Figure: 2 to 4 line decoder with enable (E) input

The circuit operates with complemented outputs and a complement enable input. The decoder is enabled when E is equal to 0 (i.e., active-low enable). As indicated by the truth table, only one output can be equal to 0 at any given time; all other outputs are equal to 1. The output whose value is equal to 0 represents the minterm selected by inputs A and B. The circuit is

disabled when E is equal to 1, regardless of the values of the other two inputs. When the circuit is disabled, none of the outputs are equal to 0 and none of the minterms are selected. In general, a decoder may operate with complemented or uncomplemented outputs. The enable input may be activated with a 0 or with a 1 signal. Some decoders have two or more enable inputs that must satisfy a given logic condition in order to enable the circuit.

A decoder with enable input can function as a demultiplexer— a circuit that receives information from a single line and directs it to one of 2^n possible output lines. The selection of a specific output is controlled by the bit combination of n selection lines. The below decoder can function as a one-to-four-line demultiplexer when E is taken as a data input line, A and B are taken as the selection inputs. The single input variable E has a path to all four outputs, but the input information is directed to only one of the output lines, as specified by the binary combination of the two selection lines A and B. This feature can be verified from the truth table of the circuit.

For example, if the selection lines AB = 10, output D2 will be the same as the input value E, while all other outputs are maintained at 1. Because decoder and demultiplexer operations are obtained from the same circuit, a decoder with an enable input is referred to as a **decoder – demultiplexer**. It is the enable input that makes the circuit a demultiplexer; the decoder itself can use AND, NAND, or NOR gates.



(a) Decoder with enable (b) Demultiplexer

Figure: Block diagram of decoder with enable and Demultiplexer

Decoders with enable inputs can be connected together to form a larger decoder circuit. The below figure shows two 3-to-8-line decoders with enable inputs connected to form a 4-to-16-line decoder. When w = 0, the top decoder is enabled and the other is disabled. The bottom decoder outputs are all 0's, and the top eight outputs generate minterms 0000 to 0111. When w = 1, the enable conditions are reversed: The bottom decoder outputs generate minterms 1000 to 1111, while the outputs of the top decoder are all 0's.

This example demonstrates the usefulness of enable inputs in decoders and other combinational logic components. In general, **enable inputs are a convenient feature for interconnecting two or more standard components** for the purpose of combining them into a similar function with more inputs and outputs.

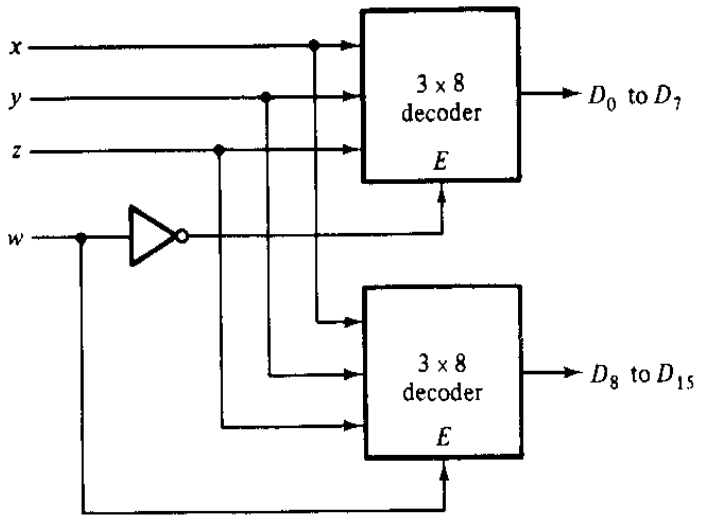


Figure: 4 * 16 decoder constructed with two 3 * 8 decoders

ENCODERS

An encoder is a digital circuit that performs the inverse operation of a decoder. An encoder has 2^n (or fewer) input lines and n output lines. The output lines, as an aggregate, generate the binary code corresponding to the input value. An example of an encoder is the octal-to-binary encoder whose truth table is given below. It has eight inputs (one for each of the octal digits) and three outputs that generate the corresponding binary number. It is assumed that only one input has a value of 1 at any given time.

The encoder can be implemented with OR gates whose inputs are determined directly from the truth table. Output z is equal to 1 when the input octal digit is 1, 3, 5, or 7. Output y is 1 for octal digits 2, 3, 6, or 7, and output x is 1 for digits 4, 5, 6, or 7. These conditions can be expressed by the following Boolean output functions:

$$z = D_1 + D_3 + D_5 + D_7$$
$$y = D_2 + D_3 + D_6 + D_7$$
$$x = D_4 + D_5 + D_6 + D_7$$

Table: Truth table for Octal to binary encoder

Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

The encoder can be implemented with three OR gates as shown below. **The encoder defined in above table has the limitation that only one input can be active at any given time**. If two inputs are active simultaneously, the output produces an undefined combination. **For example, if D3 and D6 are 1 simultaneously, the output of the encoder will be 111 because all three outputs are equal to 1. The output 111 does not represent either binary**

3 or binary 6. To resolve this ambiguity, encoder circuits must establish an input priority to ensure that only one input is encoded. If we establish a higher priority for inputs with higher subscript numbers, and if both D3 and D6 are 1 at the same time, the output will be 110 because D6 has higher priority than D3.

Another ambiguity in the octal-to-binary encoder is that an output with all 0's is generated when all the inputs are 0; but this output is the same as when D0 is equal to 1. The discrepancy can be resolved by providing one more output to indicate whether at least one input is equal to 1.

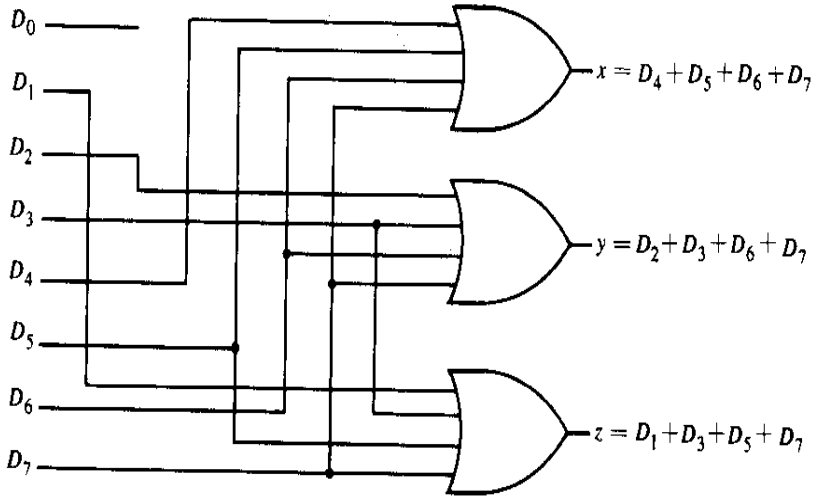


Figure: Octal to binary encoder

PRIORITY ENCODER

A **priority encoder** is an encoder circuit that includes the priority function. The operation of the priority encoder is such that **if two or more inputs are equal to 1 at the same time, the input having the highest priority will take precedence**. The truth table of a four-input priority encoder is given in below table. In addition to the two outputs x and y, the circuit has a **third output** designated by **V**; this is a valid bit indicator that **is set to 1 when one or more inputs are equal to 1**.

If all inputs are 0, there is no valid input and V is equal to 0. The other two outputs are not inspected when V equals 0 and are specified as don't-care conditions. Note that whereas X's in output columns represent don't-care conditions, the X's in the input columns are useful for representing a truth table in condensed form. Instead of listing all 16 minterms of four variables, the truth table uses an X to represent either 1 or 0. For example, X 100 represents the two minterms 0100 and 1100.

Table: Truth table of a priority encoder

Inputs				Outputs		
D_0	D_1	D_2	D_3	x	y	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

According to above truth table, the higher the subscript number, the higher the priority of the input. Input **D_3 has the highest priority**, so, regardless of the values of the other inputs, when this input is 1, the output for xy is 11 (binary 3). D_2 has the next priority level. The output is 10 if $D_2 = 1$, provided that $D_3 = 0$, regardless of the values of the other two lower priority inputs. The output for D_1 is generated only if higher priority inputs are 0, and so on down the priority levels.

The maps for simplifying outputs x and y are shown below. The minterms for the two functions are derived from above truth table. Although the table has only five rows, when each X in a row is replaced first by 0 and then by 1, we obtain all 16 possible input combinations. For example, the fourth row in the table, with inputs XX10, represents the four minterms 0010, 0110, 1010, and 1110.

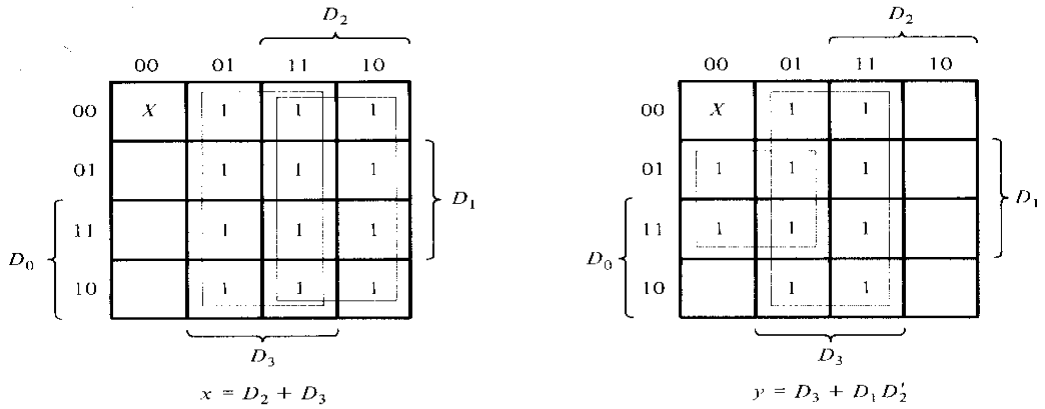


Figure: Maps for a priority encoder

The simplified Boolean expressions for the priority encoder are obtained from the maps. The condition for output V is an OR function of all the input variables. The priority encoder is implemented below according to the following Boolean functions:

x = D_2 + D_3
y = D_3 + D_1 D'$_2$
V = D_0 + D_1 + D_2 + D_3

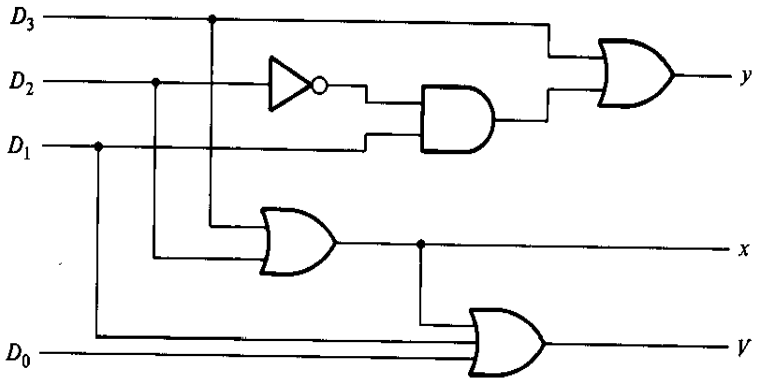


Figure: 4 input priority encoder

MULTIPLEXERS

A **multiplexer** is a combinational circuit that **selects binary information from one of many input lines and directs it to a single output line**. The selection of a particular input line is controlled by a set of selection lines. Normally, there are 2^n input lines and n selection lines whose bit combinations determine which input is selected.

A two-to-one-line multiplexer connects one of two 1-bit sources to a common destination, as shown below. The circuit has two data input lines, one output line, and one selection line S. When S = 0, the upper AND gate is enabled and I_0 has a path to the output. When S = 1, the lower AND gate is enabled and I_1 has a path to the output. The multiplexer acts like an electronic switch that selects one of two sources. The block diagram of a multiplexer is sometimes depicted by a wedge-shaped symbol, as shown in Fig. (b). It suggests visually how a selected one of multiple data sources is directed into a single destination. The multiplexer is often labeled "MUX" in block diagrams.

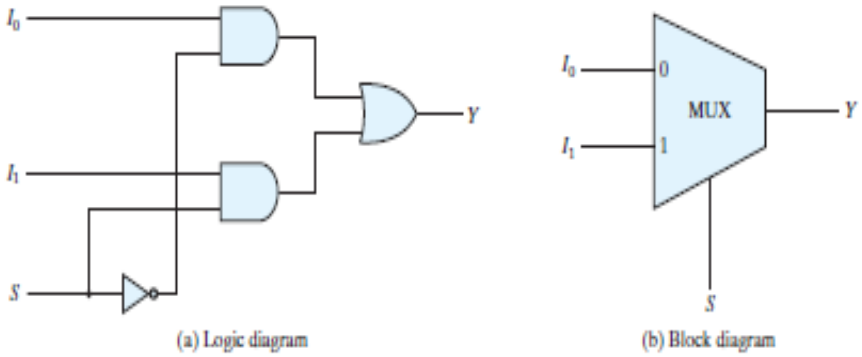


Figure: Two – to - one line multiplexer

A four-to-one-line multiplexer is shown below. Each of the four inputs, I_0 through I_3, is applied to one input of an AND gate. Selection lines S_1 and S_0 are decoded to select a particular AND gate. The outputs of the AND gates are applied to a single OR gate that provides the one-line output. The function table lists the input that is passed to the output for each combination of the binary selection values. To demonstrate the operation of the circuit, consider the case when $S_1S_0 = 10$. The AND gate associated with input I_2 has two of its inputs equal to 1 and the third input connected to I_2. The other three AND gates have at least one input equal to 0, which makes their outputs equal to 0. The output of the OR gate is now equal to the value of I_2, providing a path from the selected input to the output. **A multiplexer is also called a data selector**, since it selects one of many inputs and steers the binary information to the output line.

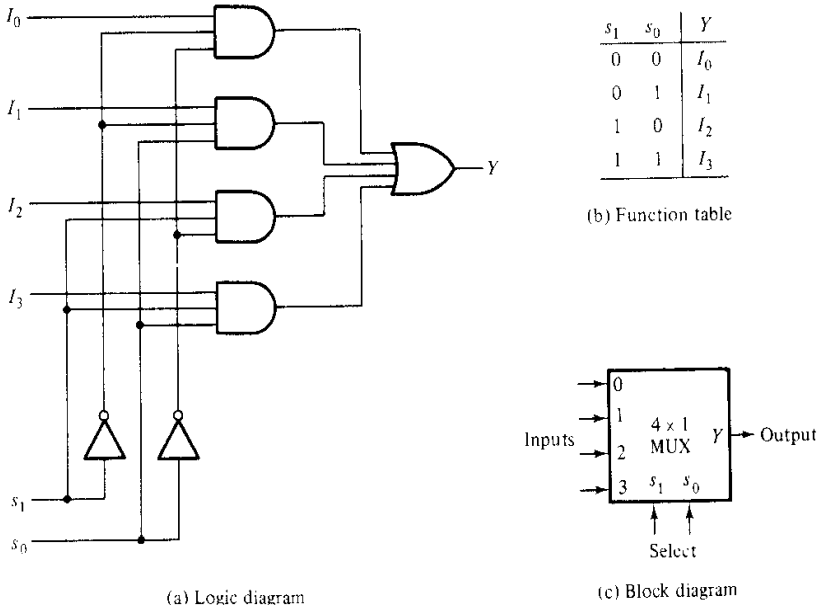


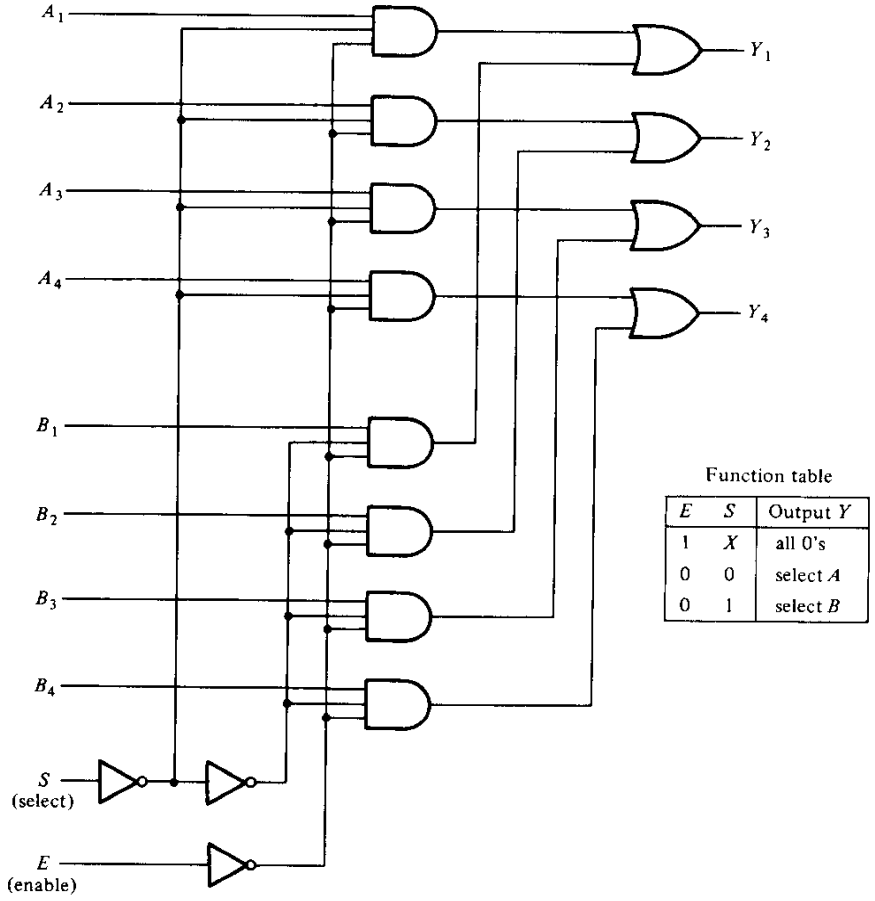
Figure: Four – to - one line multiplexer

The AND gates and inverters in the multiplexer resemble a decoder circuit, and indeed, they decode the selection input lines. In general, **a 2^n -to-1-line multiplexer is constructed from an n -to-2^n decoder by adding 2^n input lines to it, one to each AND gate**. The outputs of the AND gates are applied to a single OR gate. The size of a multiplexer is specified by the number 2^n of its data input lines and the single output line. The n selection lines are implied from the 2^n data lines.

As in decoders**, multiplexers may have an enable input to control the operation of the unit**. When the **enable input is in the inactive state**, the **outputs are disabled**, and when it is in the **active state**, the circuit functions as a **normal multiplexer**.

Multiplexer circuits can be combined with common selection inputs to provide multiple-bit selection logic. As an illustration, a quadruple 2-to-1-line multiplexer is shown below. The circuit has four multiplexers, each capable of selecting one of two input lines. Output Y_0 can be selected to come from either input A_0 or input B_0. Similarly, output Y_1 may have the value of A_1 or B_1, and so on. Input selection line S selects one of the lines in each of the four multiplexers. The enable input E must be active (i.e., asserted) for normal operation.

Although the circuit contains four 2-to-1-line multiplexers, we are more likely to view it as a circuit that selects one of two 4-bit sets of data lines. As shown in the function table, the unit is enabled when E = 0. Then, if S = 0, the four A inputs have a path to the four outputs. If, by contrast, S = 1, the four B inputs are applied to the outputs. The outputs have all 0's when E = 1, regardless of the value of S.



Function table

E	S	Output Y
1	X	all 0's
0	0	select A
0	1	select B

Figure: Quadruple two-to-one-line multiplexer

Boolean Function Implementation

An examination of the **logic diagram of a multiplexer reveals that it is essentially a decoder that includes the OR gate within the unit**. **The minterms of a function are generated in a multiplexer by the circuit associated with the selection inputs**. The individual minterms can be selected by the data inputs, thereby providing a method of implementing a Boolean function of n variables with a multiplexer that has n selection inputs and 2^n data inputs, one for each minterm.

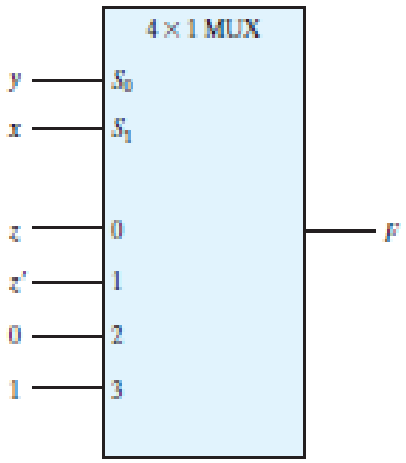
An efficient method for implementing a Boolean function of n variables with a multiplexer that has n - 1 selection inputs is shown here. The first n-1 variables of the function are connected to the selection inputs of the multiplexer. The remaining single variable of the function is used for the data inputs. If the single variable is denoted by z, each data input of the multiplexer will be z, z', 1, or 0.

Example: Implement the Boolean function F (x, y, z) = Σ (1, 2, 6, 7) with a 4 to 1 multiplexer.

Solution: The given function of three variables can be implemented with a four-to-one-line multiplexer as shown below. The two variables **x and y are applied to the selection lines** in that order; x is connected to the S_1 input and y to the S_0 input. The values for the data input lines are determined from the truth table of the function. When xy = 00, output F is equal to z because F = 0 when z = 0 and F = 1 when z = 1. This requires that variable z be applied to data input 0. The operation of the multiplexer is such that when xy = 00, data input 0 has a path to the output, and that makes F equal to z. In a similar fashion, we can determine the required input to data lines 1, 2, and 3 from the value of F when xy = 01, 10, and 11, respectively. This particular example shows all four possibilities that can be obtained for the data inputs.

x	y	z	F	
0	0	0	0	$F = z$
0	0	1	1	
0	1	0	1	$F = z'$
0	1	1	0	
1	0	0	0	$F = 0$
1	0	1	0	
1	1	0	1	$F = 1$
1	1	1	1	

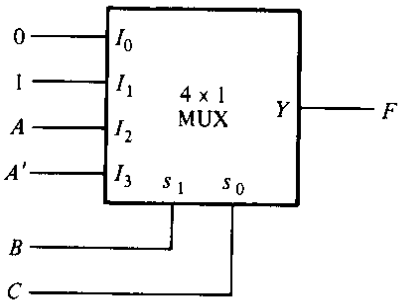
(a) Truth table



(b) Multiplexer implementation

The general procedure for implementing any Boolean function of n variables with a multiplexer with n -1 selection inputs and 2^n-1 data inputs follows from the above example. To begin with, Boolean function is listed in a truth table. Then first n - 1 variables in the table are applied to the selection inputs of the multiplexer. For each combination of the selection variables, we evaluate the output as a function of the last variable. This function can be 0, 1, the variable, or the complement of the variable. These values are then applied to the data inputs in the proper order.

Example: Implement the Boolean function F (x, y, z) = Σ (1, 3, 5, 6) with a 4 to 1 multiplexer.
Solution:



(a) Multiplexer implementation

Minterm	A	B	C	F
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0

(b) Truth table

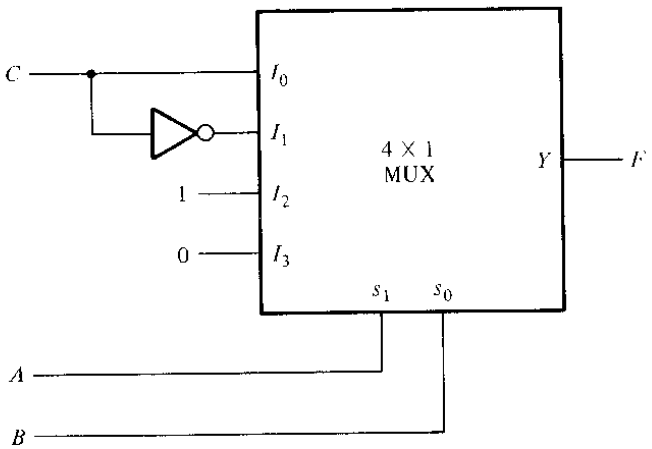
	I_0	I_1	I_2	I_3
A'	0	①	2	③
A	4	⑤	⑥	7
	0	1	A	A'

(c) Implementation table

Example: Implement the Boolean function F (x, y, z) = Σ (1, 2, 4, 5) with a 4 to 1 multiplexer.
Solution:

A	B	C	F	
0	0	0	0	F = C
0	0	1	1	
0	1	0	1	F = C'
0	1	1	0	
1	0	0	1	F = 1
1	0	1	1	
1	1	0	0	F = 0
1	1	1	0	

(a) Truth table



(b) Multiplexer implementation

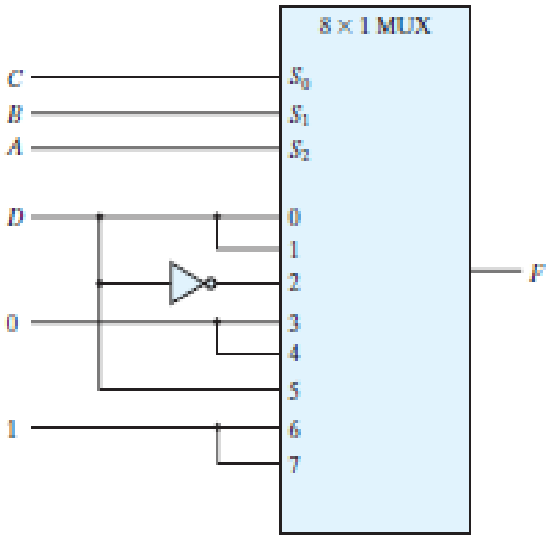
	I_0	I_1	I_2	I_3
C'	0	②	④	6
C	①	3	⑤	7
	C	C'	1	0

(c) Implementation table

Example: Implement the Boolean function F (A, B, C, D) = Σ (1, 3, 4, 11, 12, 13, 14, 15) with 8 to 1 multiplexer.

Solution:

A	B	C	D	F	
0	0	0	0	0	$F = D$
0	0	0	1	1	
0	0	1	0	0	$F = D$
0	0	1	1	1	
0	1	0	0	1	$F = D'$
0	1	0	1	0	
0	1	1	0	0	$F = 0$
0	1	1	1	0	
1	0	0	0	0	$F = 0$
1	0	0	1	0	
1	0	1	0	0	$F = D$
1	0	1	1	1	
1	1	0	0	1	$F = 1$
1	1	0	1	1	
1	1	1	0	1	$F = 1$
1	1	1	1	1	

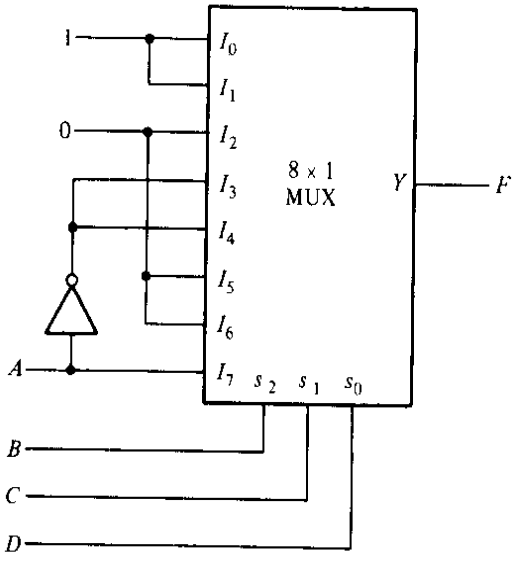


Given function is implemented with a multiplexer with three selection inputs as shown in above. Note that the first variable A must be connected to selection input S_2 so that A, B, and C correspond to selection inputs S_2, S_1, and S_0, respectively. The values for the data inputs are determined from the truth table listed below. The corresponding data line number is determined from the binary combination of ABC. For example, the table shows that when ABC = 101, F = D, so the input variable D is applied to data input 5. The binary constants 0 and 1 correspond to two fixed signal values.

Example: Implement the Boolean function F (A, B, C, D) = Σ (0, 1, 3, 4, 8, 9, 15) with 8 to 1 multiplexer.

Solution:

	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7
A'	⓪	①	2	③	④	5	6	7
A	⑧	⑨	10	11	12	13	14	⑮
	1	1	0	A'	A'	0	0	A



SEQUENTIAL LOGIC CIRCUITS

INTRODUCTION:

Earlier we studied the digital circuits whose output at any instant of time are entirely dependent on the input present at that time. Such circuits are called as combinational circuits on the other hand **sequential circuits are those in which the output at any instant of time is determined by the applied input and past history of these inputs (i.e. present state).** Alternately, **sequential circuits are those in which output at any given time is not only dependent on the input, present at that time but also on previous outputs.** Naturally, such circuits must record the previous outputs. This gives rise to memory. Often, there are requirements of digital circuits whose output remain unchanged, once set, even if the inputs are removed. Such devices are referred as "memory elements", each of which can hold 1-bit of information. These binary bits can be retained in the memory indefinitely (as long as power is delivered) or until new information is feeded to the circuit.

A combinational logic circuit that consists of inputs variable (X), logic gates (Computational circuit), and output variable (Z).

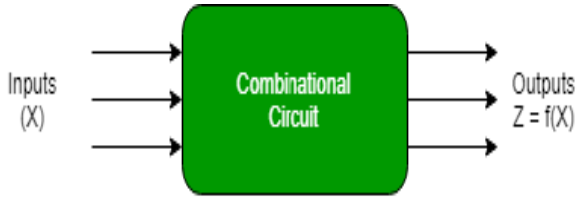


Figure: Block diagram of Combinational circuit

Combinational circuit produces an output based on input variable only, but **Sequential circuit** produces an output based on **current input and previous (past) input variables**. That means sequential circuits include memory elements which are capable of storing binary information. That binary information defines the state of the sequential circuit at that time. A latch capable of storing one bit of information.

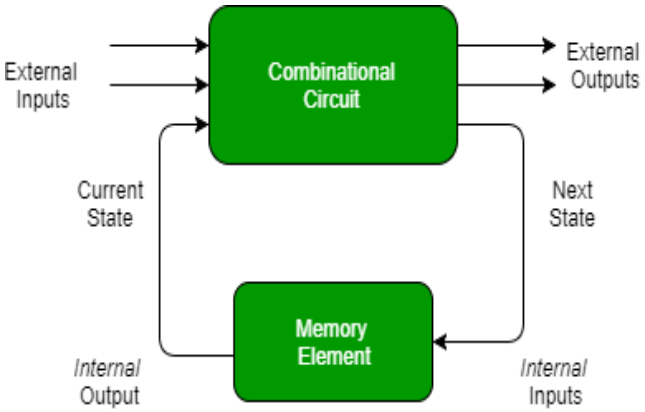


Figure: Block diagram of Sequential circuit

As shown in figure there are two types of input to the combinational logic:
1. External inputs which not controlled by the circuit.
2. Internal inputs which are a function of a previous output states.

Secondary inputs are state variables produced by the storage elements, whereas secondary outputs are excitations for the storage elements.

Combinational Circuits	Sequential Circuits
Outputs depend only on present inputs.	Outputs depend on both present inputs and present state.
Feedback path is not present.	Feedback path is present.
Memory elements are not required.	Memory elements are required.
Clock signal is not required.	Clock signal is required.
Easy to design.	Difficult to design.

Types of Sequential Circuits – There are two types of sequential circuit:

1. **Asynchronous sequential circuit –** These circuit **do not use a clock signal** but uses the pulses of the inputs. These circuits are **faster** than synchronous sequential circuits because there is clock pulse and change their state immediately when there is a change in the input signal. We use asynchronous sequential circuits when speed of operation is important and **independent** of internal clock pulse.

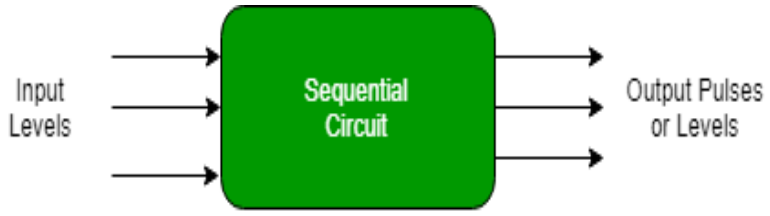


Figure: Asynchronous sequential circuit

But these circuits are more **difficult** to design and their output is **uncertain**.

2. Synchronous sequential circuit – These circuit **uses clock signal** and level inputs (or pulsed) (with restrictions on pulse width and circuit propagation). The output pulse is the same duration as the clock pulse for the clocked sequential circuits. Since they wait for the next clock pulse to arrive to perform the next operation, so these circuits are bit **slower** compared to asynchronous. Level output changes state at the start of an input pulse and remains in that until the next input or clock pulse.

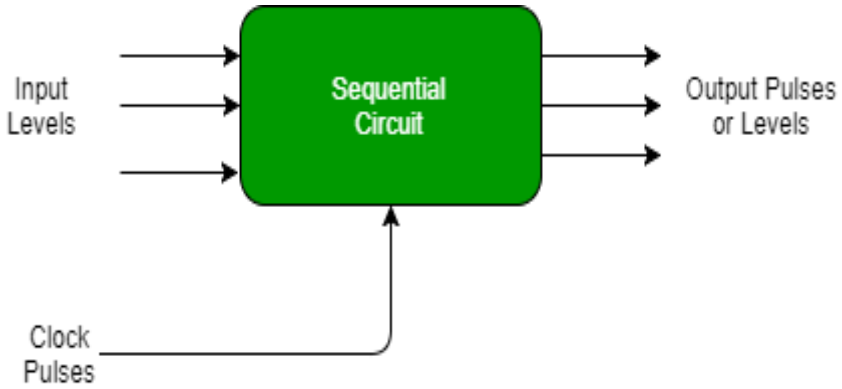


Figure: Synchronous sequential circuit

We use synchronous sequential circuit in synchronous counters, flip flops, and in the design of MOORE-MEALY state management machines. We use sequential circuits to design Counters, Registers, RAM, MOORE/MEALY Machine and other state retaining machines.

Synchronous sequential circuits	Asynchronous sequential circuits
In synchronous circuits, memory elements are clocked FFs.	In asynchronous circuits, memory elements are either un-clocked FFs or time delay elements.
The change in input signals can affect memory elements upon activation of clock signal.	The change in input signals can affect memory elements at any instant of time.
The maximum operating speed of the clock depends on time delays involved.	Because of the absence of the clock, asynchronous circuits can operate faster than synchronous circuits
Easier to design.	More difficult to design.

LATCHES AND FLIP-FLOPS:

There are two types of memory elements based on the type of triggering that is suitable to operate it.

• Latches
• Flip-flops

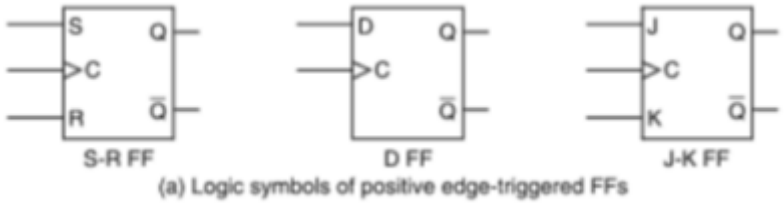
Differences between Flip flops and Latches:

Flip- flops	Latches
Flip flop is a bistable device, it has two states (0 and 1).	Latch is also a bistable device, it has two states (0 and 1).
It checks the inputs but changes the output only at times defined by the clock signal or other control signal.	It checks the inputs continuously and responds to changes in inputs immediately.
It is an edge triggered device.	It is a level triggered device.
Gates like NOR, NAND, NOT, AND are building blocks of flip flops.	These are also made up of logic gates
These are classified in to synchronous and asynchronous flip flops.	There is no such classification in latches.
These forms the building blocks of many sequential circuits like registers and counters.	These can be used for the designing of sequential circuits but are not generally preferred.
Always have a clock signal.	Doesn't have a clock signal.
Flip flops can be built from latches	Latches can be built from gates.
Ex: SR, D, JK, T- flip flops	Ex: SR, D, JK, T- Latches

Latches operate with enable signal, which is **level sensitive**. Whereas, **flip-flops are edge sensitive.**

Edge triggered Flip-Flops:

1. Positive Edge triggered Flip flop
2. Negative Edge triggered Flip flop



(a) Logic symbols of positive edge-triggered FFs

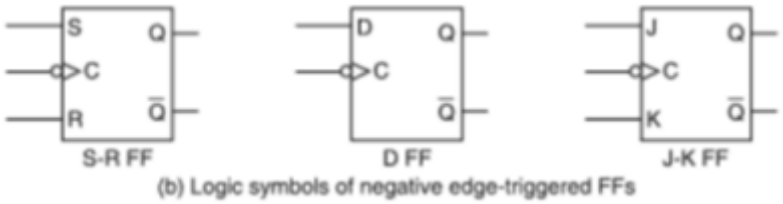


Figure: Logic symbols for positive and negative edge triggered flip-flops

POSITIVE EDGE TRIGGERED SR FLIP FLOP:

Case-1: Set = Reset = 0

This is the normal state of the FF and it has no effect on the output state. Q and Q' will remain in whatever they were prior to the occurrence of this input condition**.**

Case-2: Set = 0, Reset = 1

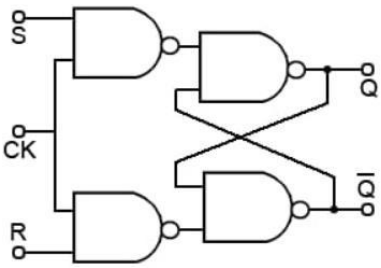
This will always set Q=0, where it will remain even after SET returns to 0, so Reset state occurs.

Case-3: Set = 1, Reset = 0

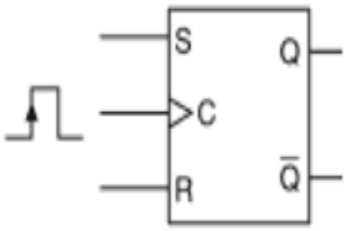
This will always set Q=1, where it will remain even after RESET returns to 1, so Set state occurs.

Case-4: Set = Reset = 1

This condition tries to SET and RESET the FF at the same time, and it produces Q = Q' =0. If the inputs are returned to zero simultaneously, the resulting output state is erratic and unpredictable. This input condition should not be used. It is forbidden.



(a) Logic diagram using NAND gates

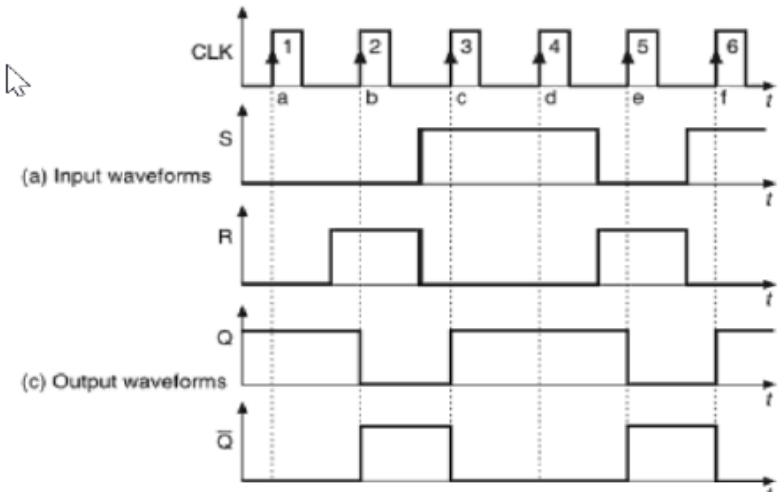


(b) Logic symbol

clk	S	R	Qn+1
0	×	×	Qn
↑	0	0	Qn
↑	0	1	0
↑	1	0	1
↑	1	1	×

clk	S	R	Qn	Qn+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	×	Indeterminate (Invalid)
↑	1	1	1	×	
0	×	×	0	0	No Change(NC)
0	×	×	1	1	

(c) Truth table of positive edge triggered SR-FF



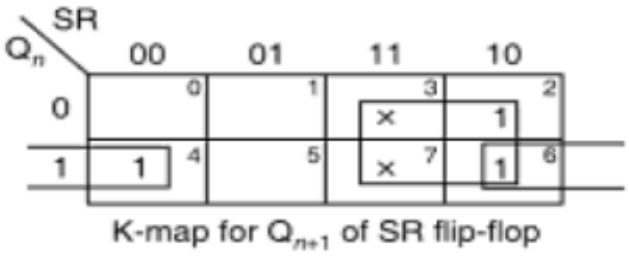
(d) Timing diagram of positive edge triggered SR- FF

Qn	S	R	Qn+1
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	X
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	X

(e) Characteristic table

Qn	Qn+1	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

(f) Excitation table



K-map for Q_{n+1} of SR flip-flop

The characteristic equation of SR flip-flop

$Q_{n+1} = S + Q_n\bar{R}$

(g) Characteristic equation

Figure: Positive edge triggered SR- Flip flop

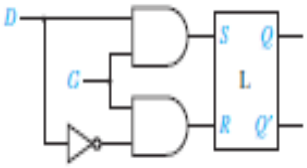
POSITIVE EDGE TRIGGERED D FLIP FLOP:

Case-1: D = 0

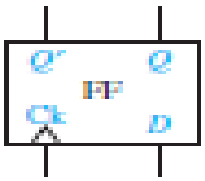
From this condition we get Qn+1' as 1. So the next state Qn+1 is 0. As the resultant is 0 it is known reset condition.

Case-2: D = 1

From this condition we get Qn+1 as 1. So this known as set condition.



(a) Logic diagram



(b) Logic symbol

clk	D	Qn	Qn+1	State
0	×	0	0	NC
0	×	1	1	
↑	0	0	0	Reset
↑	0	1	0	
↑	1	0	1	Set
↑	1	1	1	

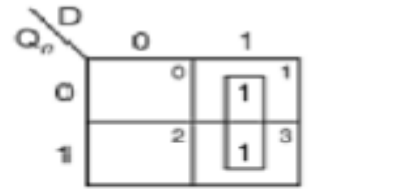
(c) Truth table

Qn	D	Qn+1
0	0	0
0	1	1
1	0	0
1	1	1

(d) Characteristic table

Qn	Qn+1	D
0	0	0
0	1	1
1	0	0
1	1	1

(e) Excitation table



K-map for Q_{n+1} of D flip-flop

The characteristic equation of D flip-flop is $Q_{n+1} = D$

(f) Characteristic equation

Figure: Positive edge triggered D- Flip flop

EDGE TRIGGERED JK FLIP FLOP:

Case 1: J = 0, K = 0

This is the normal state of the FF and it has no effect on the output state. Q and Not Q will remain in whatever they were prior to the occurrence of this input condition**.**

Case 2: J = 0, K = 1

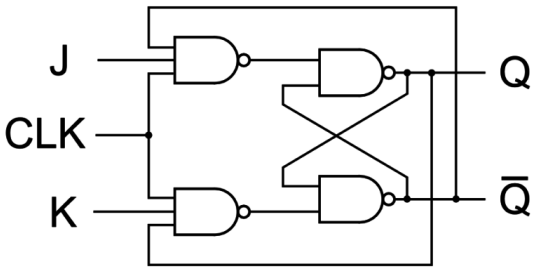
This will always set Q=0, where it will remain even after SET returns to 0

Case 3: J = 1, K = 0

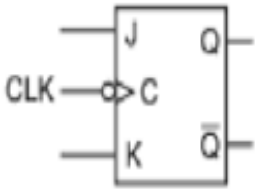
This will always Reset Q=1, where it will remain even after RESET returns to 0

Case 4: J = 1, K = 1

This condition is known as toggle condition. Since the process continues by getting alternating output i.e. frequently changing.



(a) Logic diagram



(b) Logic symbol (Negative edge triggered)

Clk	J	K	Qn	Qn+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	1	Toggle
↑	1	1	1	0	
0	X	X	0	0	No Change
0	X	X	1	1	

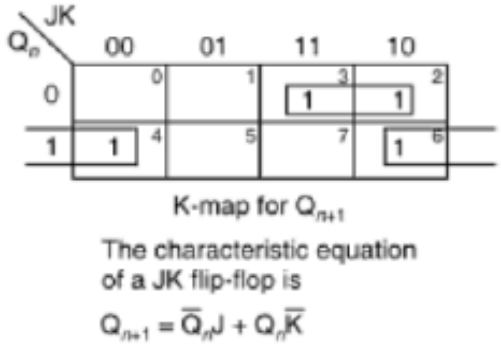
(c) Truth table

Qn	J	K	Qn+1
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

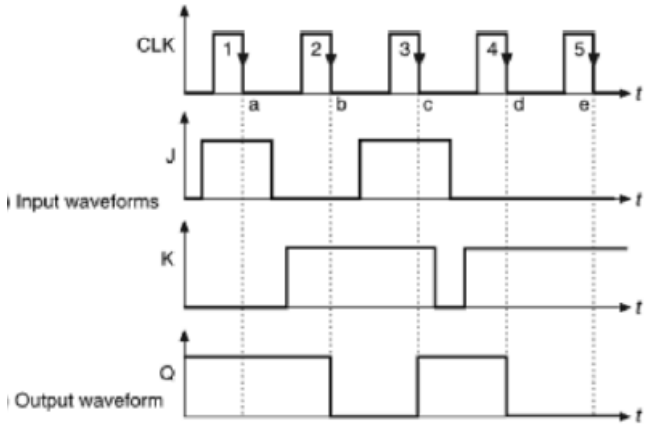
(d) Characteristic table

Qn	Qn+1	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

(e) Excitation table



(f) Characteristic equation



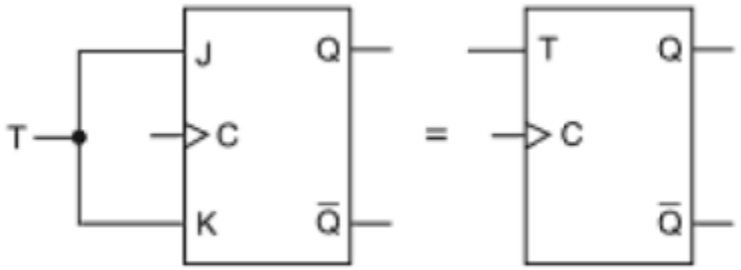
(g) Timing diagram of negative edge triggered JK- Flip flop

Figure: Edge triggered JK- Flip flop

EDGE TRIGGERED T FLIP FLOP:

Case 1: T = 0
The next state is equal to the present state. This is known as present state condition.

Case 2: T = 1
Here the output is complemented to the input. This is known as toggle condition.



(a) T flip-flop from JK flip-flop (b) Logic symbol

T	Qn	Qn+1
0	0	0
0	1	1
1	0	1
1	1	0

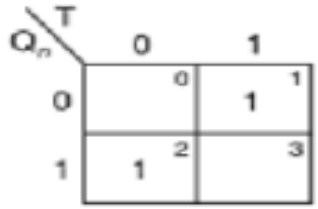
(c) Truth table

T	Qn	Qn+1
0	0	0
0	1	1
1	0	1
1	1	0

(d) Characteristic table

Qn	Qn+1	T
0	0	0
0	1	1
1	0	1
1	1	0

(e) Excitation table



K-map for Q_{n+1} of T flip-flop

The characteristic equation of T flip-flop is

$$Q_{n+1} = \bar{Q}_n T + Q_n \bar{T}$$

(f) Characteristic equation

Characteristic equations for all the FF:

Flip-flop	Characteristic equation
D	$Q_{n+1} = D$
J-K	$Q_{n+1} = J\bar{Q}_n + \bar{K}Q_n$
T	$Q_{n+1} = T\bar{Q}_n + \bar{T}Q_n$
S-R	$Q_{n+1} = S + \bar{R}Q_n$

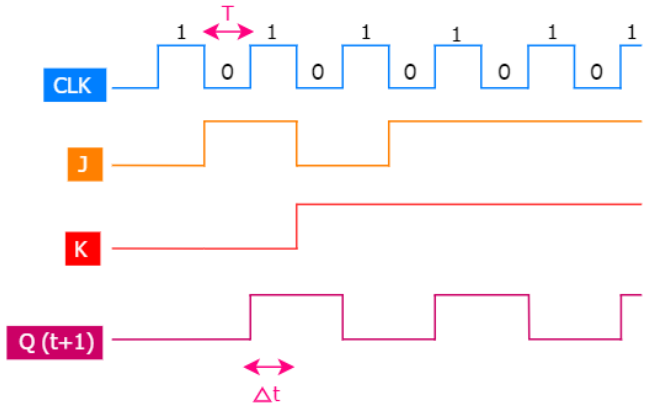
Race around Condition in JK Flip Flop:

Before getting into the race around condition, let us have a look at the JK flip-flop's truth table.

Clock Input	Inputs		Outputs		Comments
	J	K	Q	Q'	
0	X	X	Same as previous	Same as previous	No change
1	0	0	Same as previous	Same as previous	No change
1	0	1	0	1	Reset
1	1	0	1	0	Set
1	1	1	Opposite of previous	Opposite of previous	Toggle

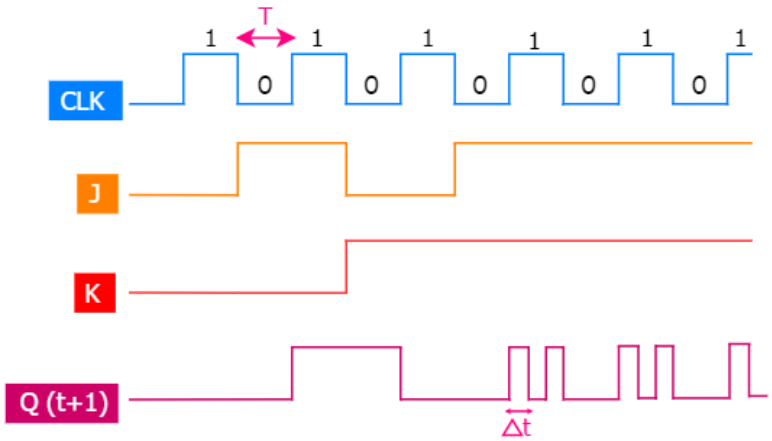
Here, Q is the present state and Q' is the next state. As you can see, when J, K and Clock are equal to 1, toggling takes place, i.e. the next state will be equal to the complement of the present state.

Now, let us look at the timing diagram of JK flip-flop.



Here, **T** is the **time period of the clock** whereas Δ **(Delta) t** is the **propagation delay**. The delay between input and output is called a propagation delay. This is what was expected, but the output may not be like this all the time. This is where **Race around condition** comes into the play.

Let us look at the timing diagram of JK flip-flop when the race around condition is considered.



As you already know, when J, K and Clock are equal to 1, toggling takes place. Here, propagation delay has also been reduced, so the output will be given out at the instant input is given. So there is a toggling again. Therefore, whenever Clock is equal to 1 there are consecutive toggling. This condition is called as **Race around condition**.

For JK flip-flop if J, K and Clock are equal to 1 the state of flip-flop keeps on toggling which leads to uncertainty in determining the output of the flip-flop. This problem is called Race around the condition. This condition also exists in T flip-flop since T flip-flop also has toggling options.

CONVERSION OF FLIP FLOPS:

The four flip-flops, namely SR flip-flop, D flip-flop, JK flip-flop & T flip-flop, can convert one flip-flop into the remaining three flip-flops by including some additional logic. So, there will be total of **twelve flip-flop conversions**.

Step 1: Identify the available and required FF.
Step 2: Make the characteristic Table for required Flip Flop.
Step 3: Make the Excitation table for available Flip Flop.
Step 4: Write Boolean Expression for available Flip Flop.
Step 5: Draw the Corresponding Logic Circuit.

1. SR FLIP FLOP TO JK FLIP FLOP:

Step 1: Identify the available and required FF.
Available→ SR Flip Flop
Required ----------→ JK Flip Flop

Step 2: Make the characteristic Table for required Flip Flop.

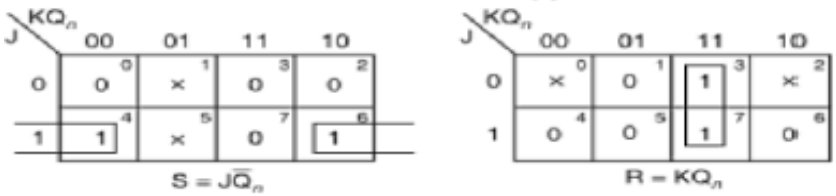
Qn	J	K	Qn+1
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Step 3: Make the Excitation table for available Flip Flop.

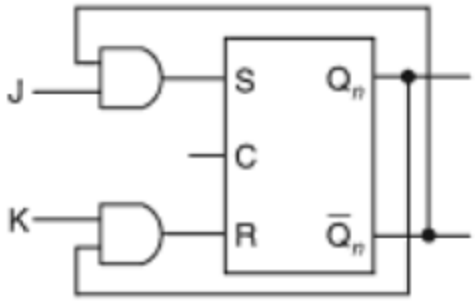
Qn	Qn+1	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

Step 4: Write Boolean Expression for available Flip Flop.

External Inputs		Present State	Next State	Flip-flop Inputs	
J	K	Q_n	Q_{n+1}	S	R
0	0	0	0	0	×
0	0	1	1	×	0
0	1	0	0	0	×
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	1	×	0
1	1	0	1	1	0
1	1	1	0	0	1



Step 5: Draw the Corresponding Logic Circuit.



2. JK FLIP FLOP TO T FLIP FLOP:

Step 1: Identify the available and required FF.
Available → JK Flip Flop
Required → T Flip Flop

Step 2: Make the characteristic Table for required Flip Flop.

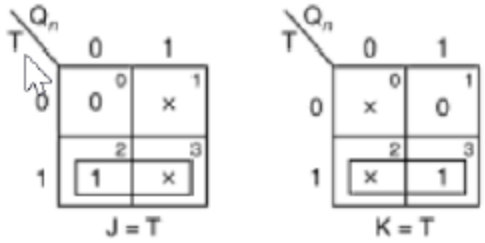
T	Qn	Qn+1
0	0	0
0	1	1
1	0	1
1	1	0

Step 3: Make the Excitation table for available Flip Flop.

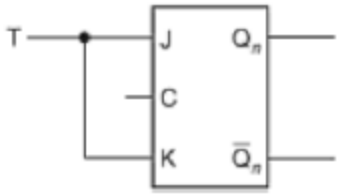
Qn	Qn+1	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

Step 4: Write Boolean Expression for available Flip Flop.

External Input	Present State	Next State	Flip-flop Inputs	
T	Q_n	Q_{n+1}	J	K
0	0	0	0	×
0	1	1	×	0
1	0	1	1	×
1	1	0	×	1



Step 5: Draw the Corresponding Logic Circuit.



SYNCHRONOUS AND ASYNCHRONOUS COUNTERS:

COUNTERS

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

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

• **Asynchronous (ripple) counter** – changing state bits are used as clocks to subsequent state flip-flops.

• **Synchronous counter** – all state bits change under control of a single clock.

• **Decade counter** – counts through ten states per stage.

• **Up/down counter** – counts both up and down, under command of a control input

• **Ring counter** – formed by a shift register with feedback connection in a ring

• **Johnson counter** – a twisted ring counter

• Cascaded counter

• Modulus counter

SYNCHRONOUS COUNTER	ASYNCHRONOUS COUNTER
In synchronous counter, all flip flops are triggered with same clock simultaneously.	In asynchronous counter, different flip flops are triggered with different clock, not simultaneously.
Synchronous counter is faster than asynchronous counter in operation.	Asynchronous counter is slower than synchronous counter in operation.
Synchronous counter does not produce any decoding errors.	Asynchronous counter produces decoding error.
Synchronous counter is also called Parallel Counter.	Asynchronous counter is also called Serial Counter.
Synchronous counter designing as well implementation are complex due to increasing the number of states.	Asynchronous counter designing as well as implementation is very easy.
Synchronous counter will operate in any desired count sequence.	Asynchronous counter will operate only in fixed count sequence (UP/DOWN).
Synchronous counter examples are: Ring counter, Johnson counter.	Asynchronous counter examples are: Ripple UP counter, Ripple DOWN counter.
In synchronous counter, propagation delay is less.	In asynchronous counter, there is high propagation delay.

A counter may be an **up/down** counter.

- An **Up counter** will count numbers from small to high in up direction.
- And **Down counter** will count numbers from high to low in down direction.

Each of the counts of the counter is called the state of the counter.

The number of states through which the counter passes before returning to the starting state is called the **modulus counter**.

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

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

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

ASYNCHRONOUS COUNTERS:

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

TWO-BIT RIPPLE UP-COUNTER USING NEGATIVE EDGE TRIGGERED FLIP FLOP:

The two-bit ripple counter uses two flip-flops. There are four possible states from 2 – bit up- counting i.e. 00, 01, 10 and 11 (counting from 0, 1, 2, 3). For a flip flop we are having bubble at clock it means the negative edge triggered FF. In this First FF output Q1 is connected as clock to the next FF so it is called as **up counter using negative edge triggered.** The counter is initially assumed to be at a reset **state 00**.

When the first clock pulse is applied FF1 toggles at the negative –going edge of this pulse, therefore, Q1 goes from LOW to HIGH. This becomes a positive –going signal at the clock input of FF2 is not affected, and hence the state of the counter after one clock pulse is Q1=1 and Q2=0 i.e., **state 01**.

At the negative –going edge of the second clock pulse, FF1 toggles. So, Q1 goes from HIGH to LOW. This becomes a negative –going signal at the clock input of FF2 and hence Q2 goes from LOW to HIGH. Therefore, the state of the counter after second clock pulse is Q1=0 and Q2=1 i.e., **state 10**.

At the negative –going edge of the third clock pulse, FF1 toggles. So, Q1 goes from LOW to HIGH. This becomes a positive –going signal at the clock input of FF2 and is not affected. Therefore, the state of the counter after third clock pulse is Q1=1 and Q2=1 i.e., **state 11**.

At the negative –going edge of the fourth clock pulse, FF1 toggles. So, Q1 goes from HIGH to LOW HIGH. This becomes a negative –going signal at the clock input of FF2 and toggles. Therefore, the state of the counter after fourth clock pulse is Q1=0 and Q2=0 i.e., **state 00**.

The above process repeats between 00→01→10→11 →00 →01 →10 and so on. So it means that it is like mod 4 counter running from 0 to 3 and again from starting.

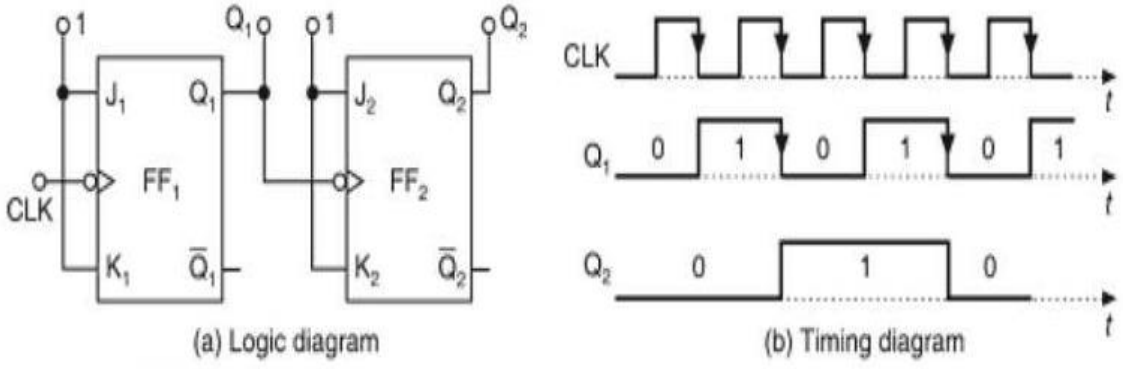


Figure: Asynchronous 2-bit ripple up-counter using negative edge triggered flip flop

TWO-BIT RIPPLE DOWN-COUNTER USING NEGATIVE EDGE TRIGGERED FLIP FLOP:

The two-bit ripple counter uses two flip-flops. There are four possible states from 2 – bit down- counting I.e. 11, 10, 01 and 00 (counting from 3, 2, 1, 0). For a flip flop we are having bubble at clock it means the negative edge triggered FF. In this First FF output Q1 is

connected as clock to the next FF so it is called as **down counter using negative edge triggered.** The counter is initially assumed to be at a reset **state 00**.

When the first clock pulse is applied FF1 toggles at the negative –going edge of this pulse, therefore, Q1 goes from LOW to HIGH and Q1 goes from a 1 to 0. This becomes a negative –going signal Q1 applied at the clock input of FF2 and toggles from 0 to a 1 and hence the state of the counter after one clock pulse is Q2=1 and Q1=1 i.e., **state 11**.

At the negative –going edge of the second clock pulse, FF1 toggles. So, Q1 goes from HIGH to LOW. Therefore, Q2 remains at 1. Hence, the state of the counter after second clock pulse is **state10**.

At the negative –going edge of the third clock pulse, FF1 toggles. So, Q1 goes from LOW to HIGH. And Q1 from a 1 to a 0. This becomes a negative –going signal at the clock input of FF2 so Q2 changes from a 1 to 0. Therefore, the state of the counter after third clock pulse is Q1=1 and Q2=0 i.e., **state 01**.

At the negative –going edge of the fourth clock pulse, FF1 toggles. So, Q1 goes from HIGH to LOW HIGH and Q1 from a 0 to a 1. This becomes a positive –going signal at the clock input of FF2 is not affected. Therefore, the state of the counter after fourth clock pulse is Q1=0 and Q2=0 i.e., **state 00**.

The above process repeats between 00 → 11 → 10 → 01→ 00 → 11 →10 so on. So it means that it is like mod 4 counter running from 3 to 0 and again from starting.

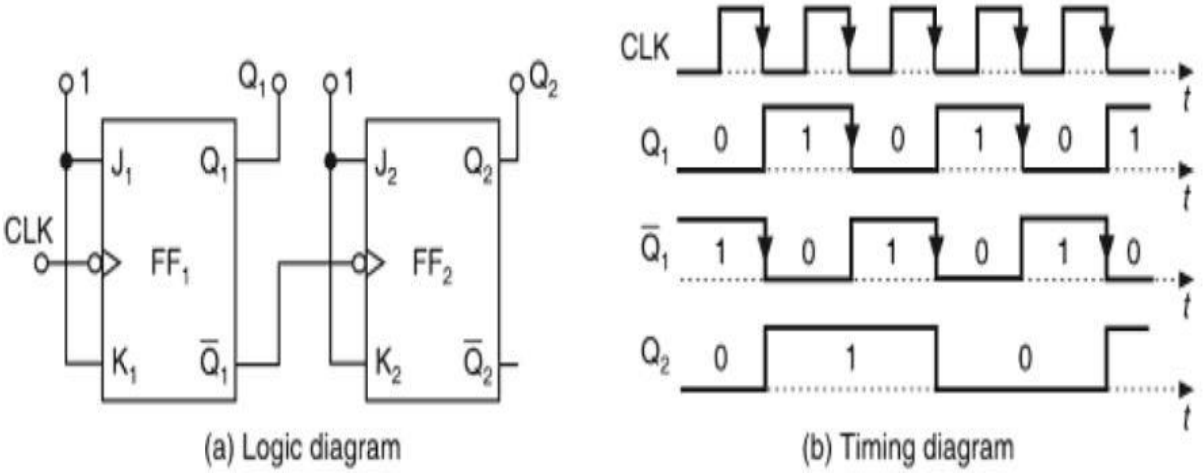


Figure: Asynchronous 2-bit ripple down-counter using negative edge triggered flip flop

TWO-BIT RIPPLE UP-DOWN COUNTER USING NEGATIVE EDGE TRIGGERED FLIP FLOP:

As the name indicates an up-down counter is a counter which can count both in upward and downward directions. An up-down counter is also called a forward/backward counter or a bidirectional counter. So, a control signal or a mode signal M is required to choose the direction of count. When M=1 for up counting, Q1 is transmitted to clock of FF2 and when M=0 for down counting, Q1' is transmitted to clock of FF2. This is achieved by using two AND gates and one OR gates. The external clock signal is applied to FF1.

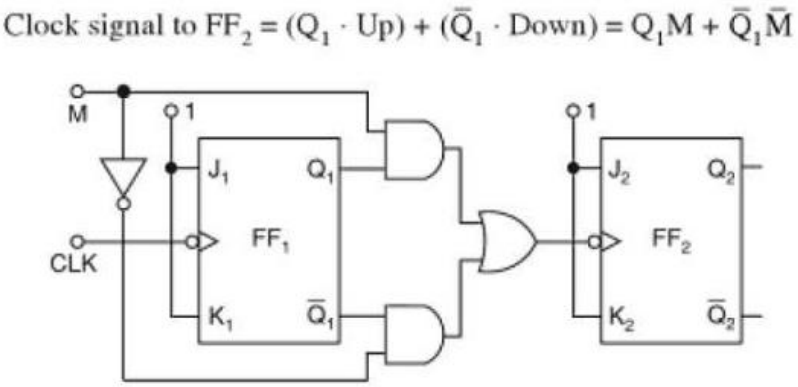


Figure: Asynchronous 2-bit ripple up-down counter using negative edge triggered flip flop

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

The two-bit ripple counter uses two flip-flops. There are four possible states from 2 – bit up- counting i.e. 00, 01, 10 and 11 (counting from 0, 1, 2, 3). For a flip flop we are not having bubble at clock it means the positive edge triggered FF.

In this First FF output Q1 is connected as clock to the next FF so it is called as **up counter using positive edge triggered.** The counter is initially assumed to be at a reset **state 00**. FF1 toggles at the positive going edge of each clock pulse and FF2 toggles whenever Q1' changes from a 0 to a 1. State transitions occur at the positive – going edges of the clock pulses. The counting sequence is 00, 01, 10, 11, 00, 01, 10, 11 etc.

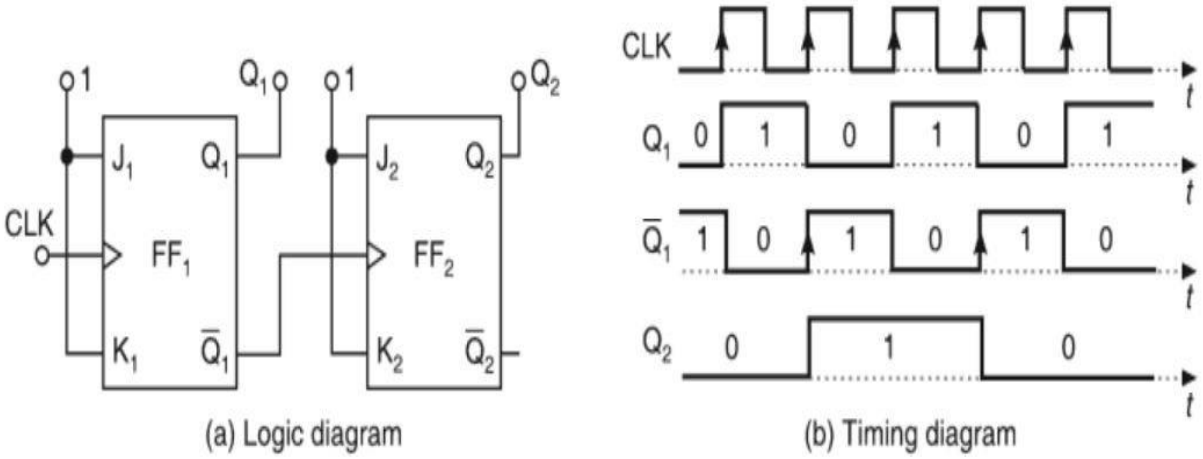


Figure: Asynchronous 2-bit ripple up-counter using positive edge triggered flip flop

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

The two-bit ripple counter uses two flip-flops. There are four possible states from 2 – bit down- counting I.e. 11, 10, 01 and 00 (counting from 3, 2, 1, 0). For a flip flop we are not having bubble at clock it means the positive edge triggered FF.

In this First FF output Q1 is connected as clock to the next FF so it is called as **down counter using positive edge triggered.** The counter is initially assumed to be at a reset **state 00**. FF1 toggles at the positive going edge of each clock pulse and FF2 toggles whenever Q1' changes from a 0 to a 1. State transitions occur at the positive – going edges of the clock pulses. The counting sequence is 00, 11, 10, 01, 00, 11, 10, 11 etc.

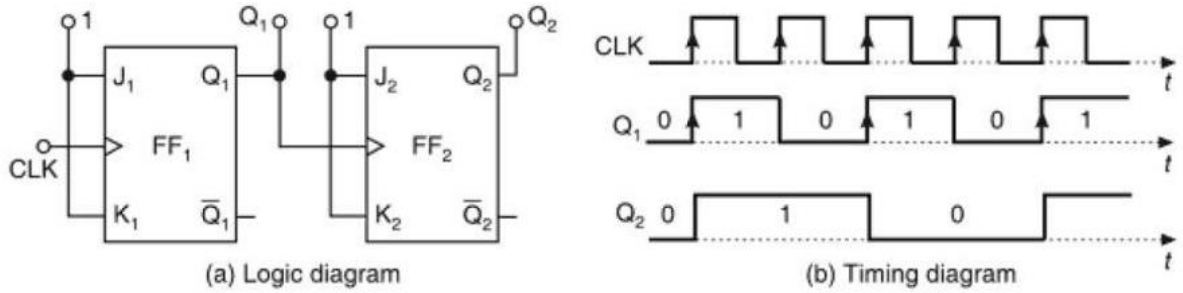


Figure: Asynchronous 2-bit ripple down-counter using positive edge triggered flip flop

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

As the name indicates an up-down counter is a counter which can count both in upward and downward directions. An up-down counter is also called a forward/backward counter or a bidirectional counter. So, a control signal or a mode signal M is required to choose the direction of count. When M=1 for up counting, Q1' is transmitted to clock of FF2 and when M=0 for down counting, Q1 is transmitted to clock of FF2. This is achieved by using two AND gates and one OR gates. The external clock signal is applied to FF1.

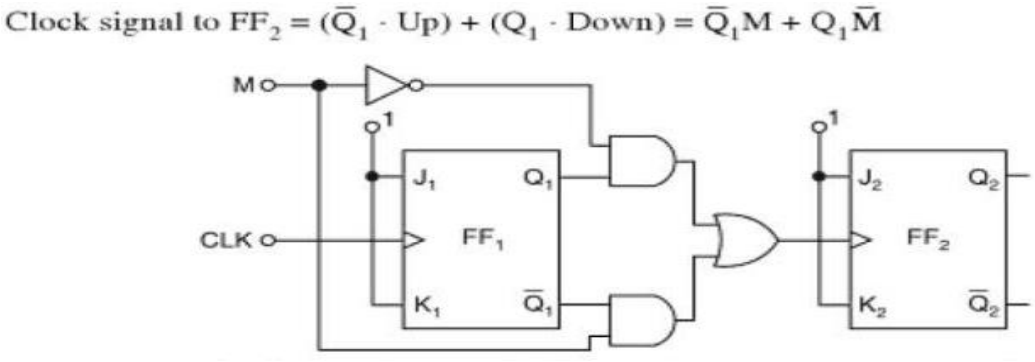


Figure: Asynchronous 2-bit ripple up-down counter using positive edge triggered flip flop

DESIGN OF ASYNCHRONOUS COUNTERS:

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

DESIGN OF A MOD-6 ASYNCHRONOUS COUNTER USING T FFS:

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

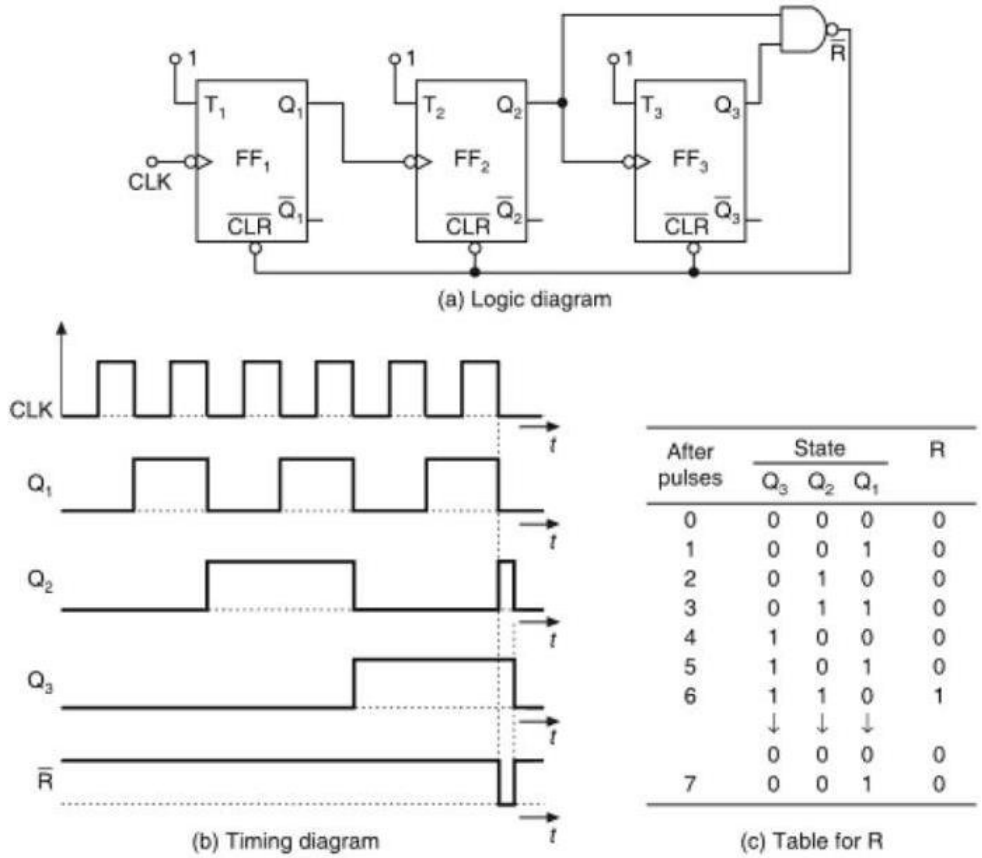
After **s**ixth clock pulse it goes to 000. For the design, write the truth table with present state outputs Q3, Q2 and Q1 as the variables, and reset R as the output and obtain an expression

for R in terms of Q3, Q2, and Q1that decides the feedback into be provided. From the truth table, R=Q3Q2. For active-low Reset, R' is used. The reset pulse is of very short duration, of the order of nanoseconds and it is equal to the propagation delay time of the NAND gate used.

The expression for R can also be determined as follows.
Therefore,

R= 0 for 000 to 101, R=1 for 110, and R =X = for111
R = Q3Q2Q1'+ Q3Q2Q1 = Q3Q2

The logic diagram, timing diagram and truth table of Mod-6 counter is shown in the below figure.



After pulses	State Q_3	Q_2	Q_1	R
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	0
4	1	0	0	0
5	1	0	1	0
6	1	1	0	1
	↓	↓	↓	
	0	0	0	0
7	0	0	1	0

(c) Table for R

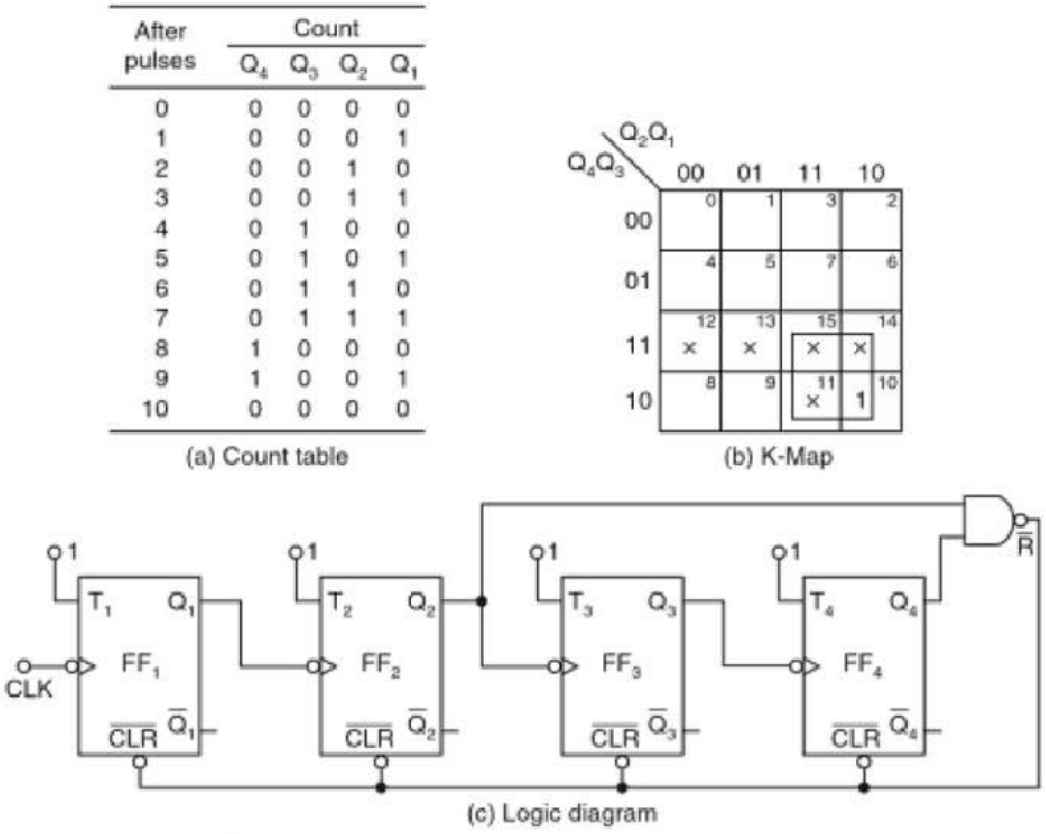
Figure: Asynchronous Mod-6 counter using T- Flip flops

DESIGN OF A MOD-10 ASYNCHRONOUS COUNTER USING T-FLIP-FLOPS:

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

The count table and the K-Map for reset are shown in below figure. From the K-Map R = Q4Q2. So, feedback is provided from second and fourth FFs. For active –HIGH reset,

Q4Q2 is applied to the clear terminal. For active-LOW reset Q4Q2 is connected is of all Flip-flops.



After pulses	Count Q_4	Q_3	Q_2	Q_1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	0	0	0	0

Figure: Asynchronous Mod-10 counter using T- Flip flops

SYNCHRONOUS COUNTERS:

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

Design of synchronous counters:

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

Step 1: Number of Flip-flops: Based on the description of the problem, determine the required number n of the flip-flops, the smallest value of n is such that the number of states $N \leq 2^n$ and the desired counting sequence.

Step 2: State Diagram: Draw the state diagram showing all the possible states state diagram which also be called nth transition diagrams, is a graphical means of depicting the sequence of states through which the counter progresses.

Step 3: Choice of Flip-flops excitation table: Select the type of flip-flop to be used and write the excitation table. An excitation table is a table that lists the present state (PS), the next state (NS) and required excitations.

Step 4: **Minimal expressions for excitations:** Obtain the minimal expressions for the excitations of the FF using K-maps drawn for the excitation of the flip-flops in terms of the present states and inputs.

Step 5: **Logic diagram**: Draw a logic diagram based on the minimal expressions.

Table Excitation tables

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

(a) S-R FF excitation table

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

(b) J-K FF excitation table

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

(c) D FF excitation table

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

(d) T FF excitation table

Figure: Excitation of various Flip-flops

DESIGN OF SYNCHRONOUS 3-BIT UP COUNTER:

Step 1: Determine the number of flip-flops required: A 3-bit up counter requires three FFs. It has 8 states (000, 001, 010, 011, 100, 101, 110, 111, and 000) and all the states are valid.

Step 2: Draw the state diagrams: The state diagram of the 3-bit up counter is drawn as shown in the below figure.

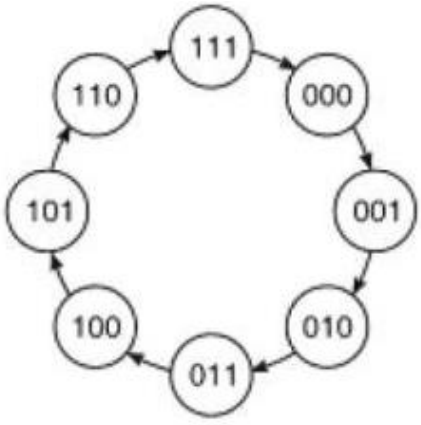


Figure: State diagram of synchronous 3-bit up-counter

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

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	×	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

Figure: Excitation table of synchronous 3-bit up-counter

Step 4: Obtain the minimal expressions: From the excitation table we can conclude that J1=1 and K1=1, because all the entries for J1and K1 are either X or 1. The K-maps for J3, K3, J2 and K2 based on the excitation table and the minimal expression obtained from them are shown in the below figure.

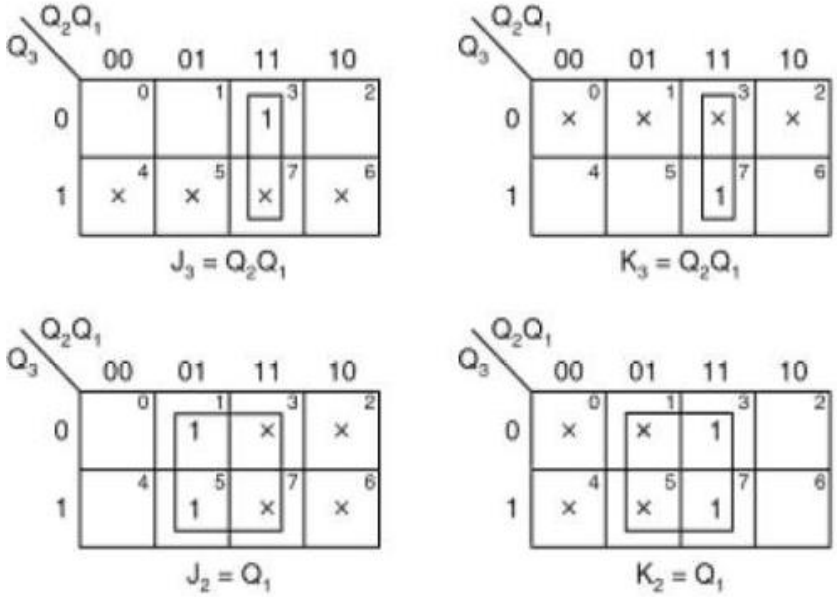


Figure: K-map minimization of synchronous 3-bit up-counter

Step 5: Draw the logic diagram: The logic diagram using those minimal expressions can be drawn as shown in the below figure.

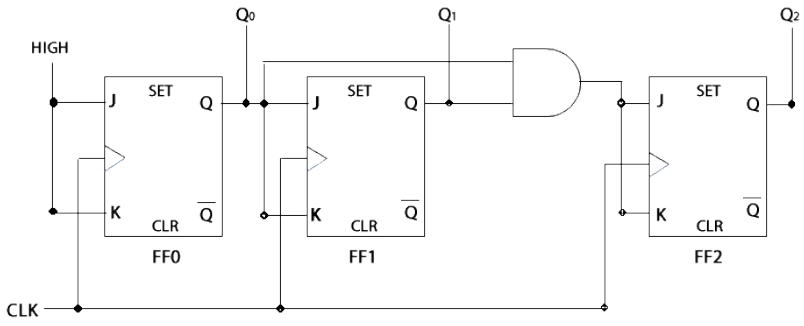


Figure: Logic diagram of synchronous 3-bit up counter

DESIGN OF SYNCHRONOUS 3-BIT DOWN COUNTER:

Step 1: Determine the number of flip-flops required: A 3-bit down counter requires three FFs. It has 8 states (000, 111, 110, 101, 100, 011, 010, 001, 000) and all the states are valid.

Step 2: Draw the state diagrams: the state diagram of the 3-bit down counter is drawn as shown in the below figure.

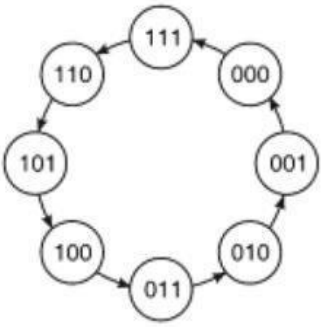


Figure: State diagram of synchronous 3-bit down counter

Step 3: Select the type of flip flop and draw the excitation table: JK flip-flops are selected and the excitation table of a 3-bit down- counter using JK flip-flops is drawn as shown in the below figure.

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	1	1	1	1	×	1	×	1	×
1	1	1	1	1	0	×	0	×	0	×	1
1	1	0	1	0	1	×	0	×	1	1	×
1	0	1	1	0	0	×	0	0	×	×	1
1	0	0	0	1	1	×	1	1	×	1	×
0	1	1	0	1	0	0	×	×	0	×	1
0	1	0	0	0	1	0	×	×	1	1	×
0	0	1	0	0	0	0	×	0	×	×	1

Figure: K-map minimization of synchronous 3-bit down counter

Step 4: Obtain the minimal expressions: From the excitation table we can conclude that J1=1 and K1=1, because all the entries for J1and K1 are either X or 1. The K-maps for J3, K3, J2 and K2 based on the excitation table and the minimal expression obtained from them are shown in the below figure.

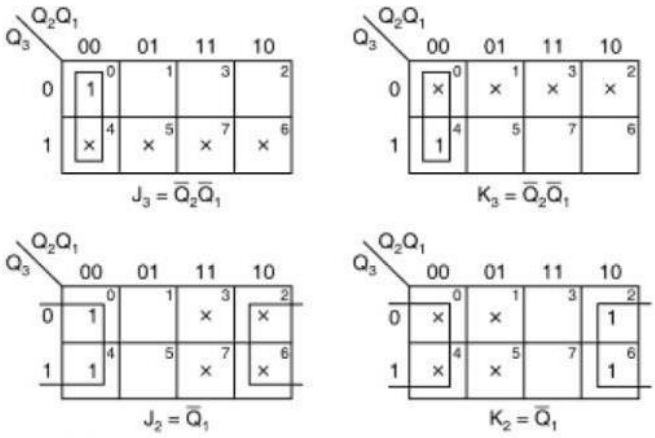
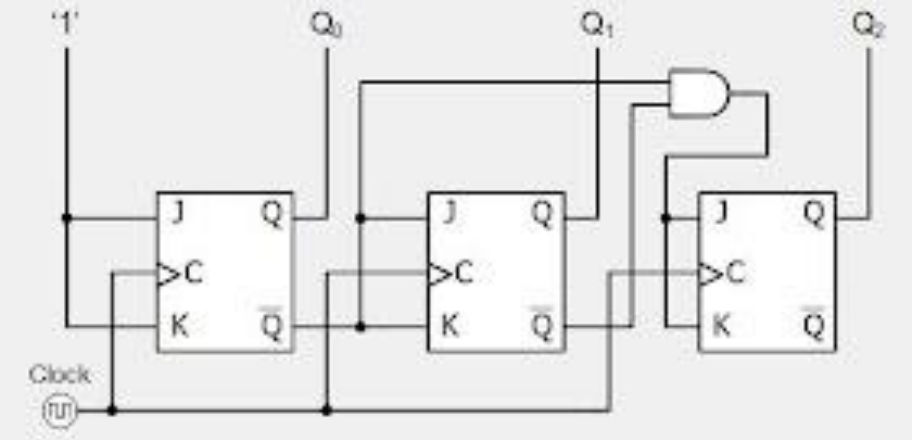


Figure: K-map minimization of synchronous 3-bit down counter

Step 5: Draw the logic diagram: A logic diagram using those minimal expressions can be drawn as shown in the below figure.



DESIGN OF A SYNCHRONOUS 3-BIT UP-DOWN COUNTER USING JK FLIP-FLOPS:

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

Step 2: Draw the state diagrams: The state diagram of the 3-bit up-down counter is drawn as shown in figure.

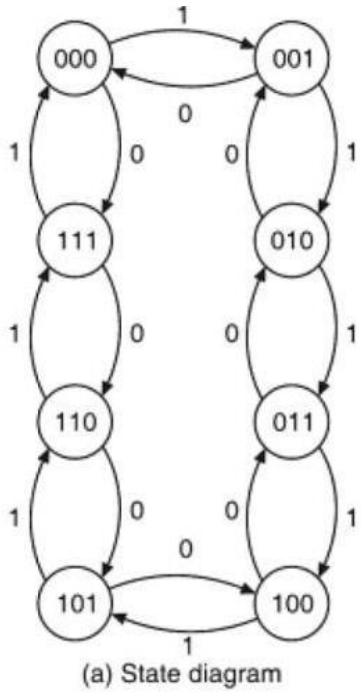


Figure: State diagram of synchronous 3-bit up-down counter

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

PS			Mode	NS			Required excitations					
Q_3	Q_2	Q_1	M	Q_3	Q_2	Q_1	J_3	K_3	J_2	K_2	J_1	K_1
0	0	0	0	1	1	1	1	×	1	×	1	×
0	0	0	1	0	0	1	0	×	0	×	1	×
0	0	1	0	0	0	0	0	×	0	×	×	1
0	0	1	1	0	1	0	0	×	1	×	×	1
0	1	0	0	0	0	1	0	×	×	1	1	×
0	1	0	1	0	1	1	0	×	×	0	1	×
0	1	1	0	0	1	0	0	×	×	0	×	1
0	1	1	1	1	0	0	1	×	×	1	×	1
1	0	0	0	0	1	1	×	1	1	×	1	×
1	0	0	1	1	0	1	×	0	0	×	1	×
1	0	1	0	1	0	0	×	0	0	×	×	1
1	0	1	1	1	1	0	×	0	1	×	×	1
1	1	0	0	1	0	1	×	0	×	1	1	×
1	1	0	1	1	1	1	×	0	×	0	1	×
1	1	1	0	1	1	0	×	0	×	0	×	1
1	1	1	1	0	0	0	×	1	×	1	×	1

(b) Excitation table

Figure: Excitation table of synchronous 3-bit up-down counter

Step 4: Obtain the minimal expressions: From the excitation table we can conclude that J1=1 and K1=1, because all the entries for J1and K1 are either X or 1. The K-maps for J3, K3, J2 and K2 based on the excitation table and the minimal expression obtained from them are shown in below figure.

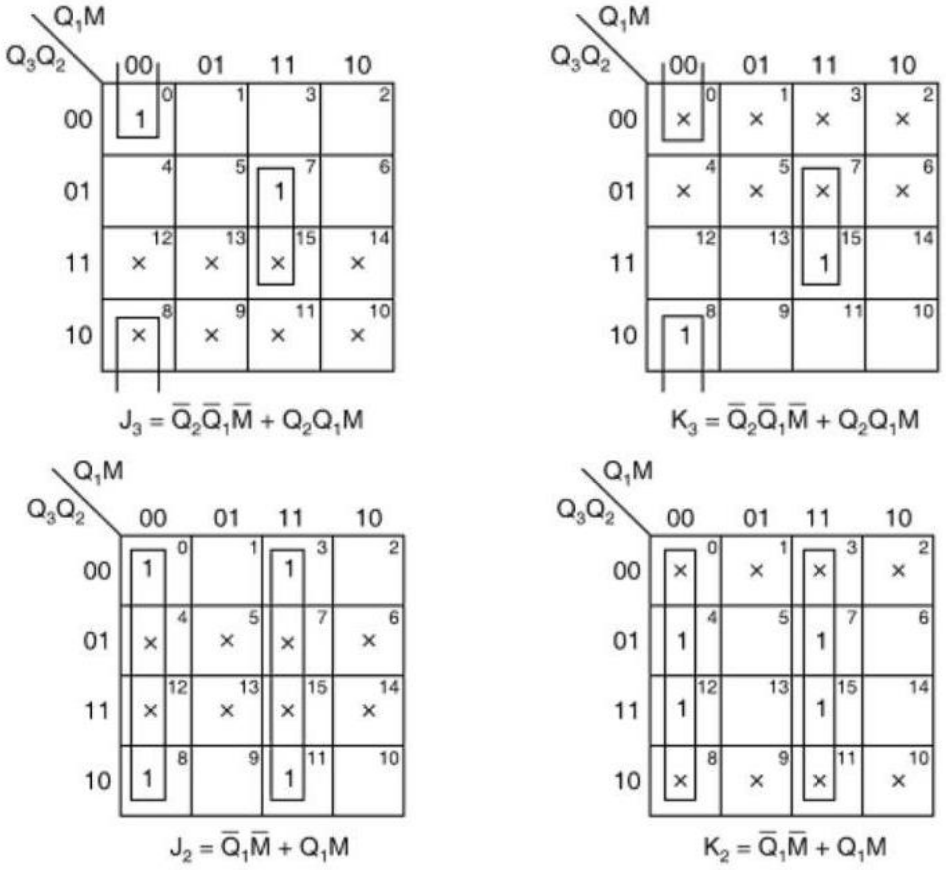


Figure: K-map minimization of synchronous 3-bit up-down counter

Step 5: Draw the logic diagram: Logic diagram using those minimal expressions can be drawn as shown in figure.

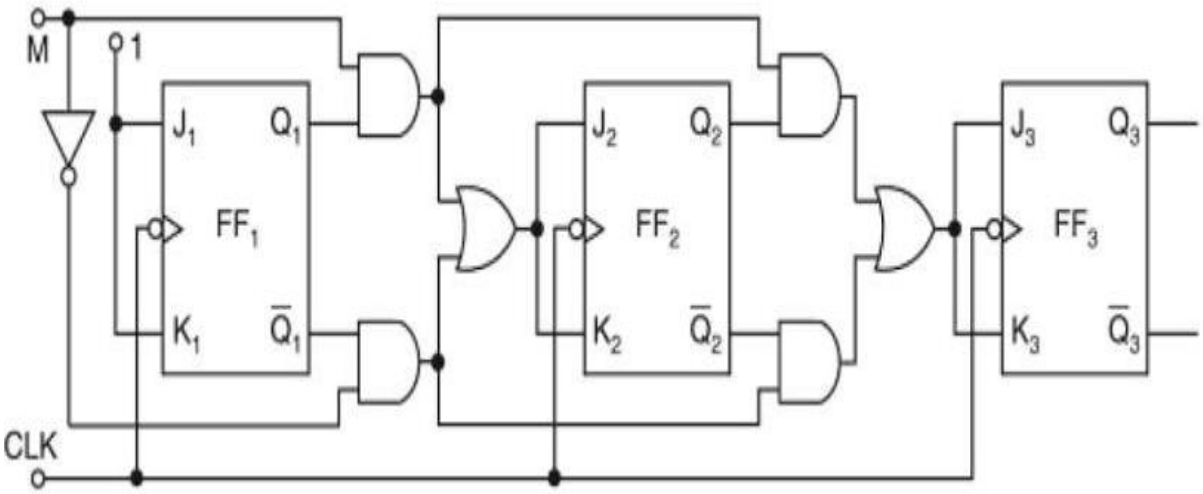


Figure: Logic diagram of synchronous 3-bit up-down counter

DESIGN OF SYNCHRONOUS MODULO-10 UP/DOWN COUNTER USING T FFS:

Step 1: Determine the number of flip-flops required: A modulo-10 has 10 states and so it requires 4 FFs. ($10 \leq 2^4$).

4 FF can have 16 states. So out of 16, six states are invalid (1010 through 1111).

M=1 → Up Counter

M=0 → Down Counter

Step 2: Draw the state diagrams: The state diagram of the mod-10 up/down counter is drawn as shown in the below figure.

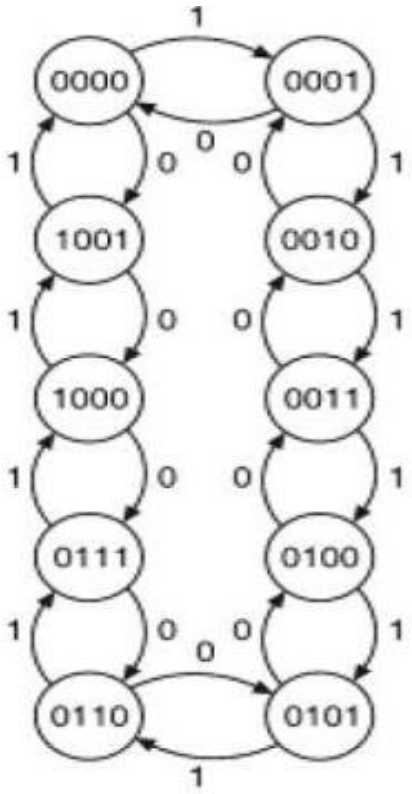


Figure: State diagram of synchronous mod-10 up-down counter

Step 3: Select the type of flip flop and draw the excitation table: T flip-flops are selected and the excitation table of modulo 10 up/down- counter using T flip-flops is drawn as shown in the below figure.

PS				Mode	NS				Required excitations			
Q_4	Q_3	Q_2	Q_1	M	Q_4	Q_3	Q_2	Q_1	T_4	T_3	T_2	T_1
0	0	0	0	0	1	0	0	1	1	0	0	1
0	0	0	0	1	0	0	0	1	0	0	0	1
0	0	0	1	0	0	0	0	0	0	0	0	1
0	0	0	1	1	0	0	1	0	0	0	1	1
0	0	1	0	0	0	0	0	1	0	0	1	1
0	0	1	0	1	0	0	1	1	0	0	0	1
0	0	1	1	0	0	0	1	0	0	0	0	1
0	0	1	1	1	0	1	0	0	0	1	1	1
0	1	0	0	0	0	0	1	1	0	1	1	1
0	1	0	0	1	0	1	0	1	0	0	0	1
0	1	0	1	0	0	1	0	0	0	0	0	1
0	1	0	1	1	0	1	1	0	0	0	1	1
0	1	1	0	0	0	1	0	1	0	0	1	1
0	1	1	0	1	0	1	1	1	0	0	0	1
0	1	1	1	0	0	1	1	0	0	0	0	1
0	1	1	1	1	1	0	0	0	1	1	1	1
1	0	0	0	0	0	1	1	1	1	1	1	1
1	0	0	0	1	1	0	0	1	0	0	0	1
1	0	0	1	0	1	0	0	0	0	0	0	1
1	0	0	1	1	0	0	0	0	1	0	0	1

Figure: Excitation table of synchronous mod-10 up-down counter

Step 4: Obtain the minimal expressions: From the excitation table we can conclude that T=1, because all the entries for T1 are 1. The K-maps T4, T3 and T2 based on the excitation table. In the K-maps, the remaining minterms are don't cares (Σd (20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31))

From the excitation table we can see that T1 = 1 and the expression for T4, T3, and T2 are

T4 = Σm (0, 15, 16, 19) + d (20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31)

T3 = Σm (7, 8, 15, 16) + d (20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31)

T2 = Σm (3, 4, 7, 8, 11, 12, 15, 16) + d (20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31)

The minimal expression for T2 obtained by minimizing the K-map is given below.

$$T2 = Q4\overline{Q1}\overline{M} + \overline{Q4}Q1M + Q2\overline{Q1}M + Q3\overline{Q1}M$$

Step5: Draw the logic diagram: A logic diagram using those minimal expressions can be drawn as shown in the below figure.

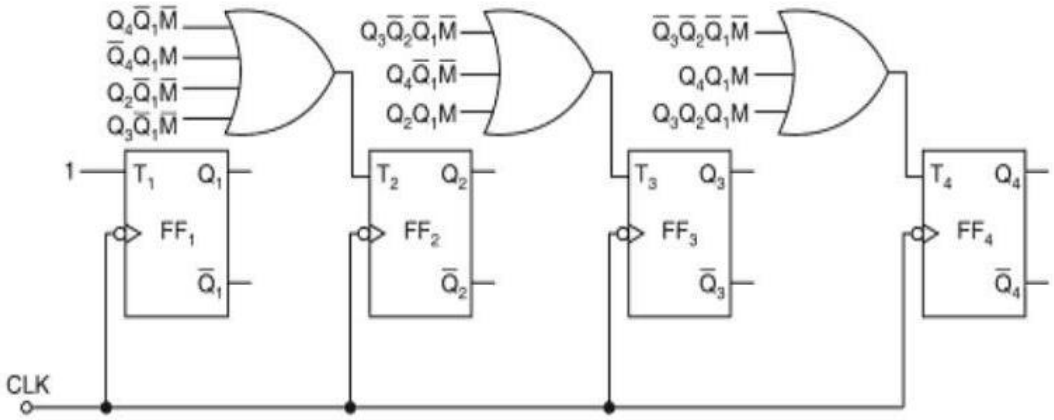


Figure: Logic diagram of synchronous mod-10 up-down counter

DESIGN OF MODULO-9 SYNCHRONOUS COUNTER USING T FFS:

Step 1: Determine the number of flip-flops required: A modulo 9 counter requires four FFs. It has 9 states (0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 0000……) all the states are valid. There are invalid from 1001 to 1111.

Step 2: Draw the state diagrams: the state diagram of the mod-9 counter is drawn as shown in the below figure.

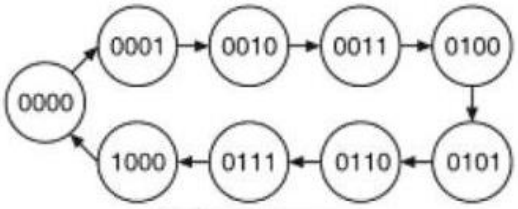


Figure: State diagram of synchronous modulo-9 counter

Step 3: Select the type of flip flop and draw the excitation table: T flip-flops are selected and the excitation table of a mod 9 counter using T flip-flops is drawn as shown in the below figure.

PS				NS				Required Excitations			
Q_4	Q_3	Q_2	Q_1	Q_4	Q_3	Q_2	Q_1	T_4	T_3	T_2	T_1
0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	1	0	0	1	0	0	0	1	1
0	0	1	0	0	0	1	1	0	0	0	1
0	0	1	1	0	1	0	0	0	1	1	1
0	1	0	0	0	1	0	1	0	0	0	1
0	1	0	1	0	1	1	0	0	0	1	1
0	1	1	0	0	1	1	1	0	0	0	1
0	1	1	1	1	0	0	0	1	1	1	1
1	0	0	0	0	0	0	0	1	0	0	0

Figure: Excitation table of synchronous modulo-9 counter

Step 4: Obtain the minimal expressions: The K-maps for excitations T4, T3, T2, and T1 in terms of the outputs of the FFs Q4, Q3, Q2, and Q1 their minimization and the minimal expressions for excitation obtained from them are shown in the below figure.

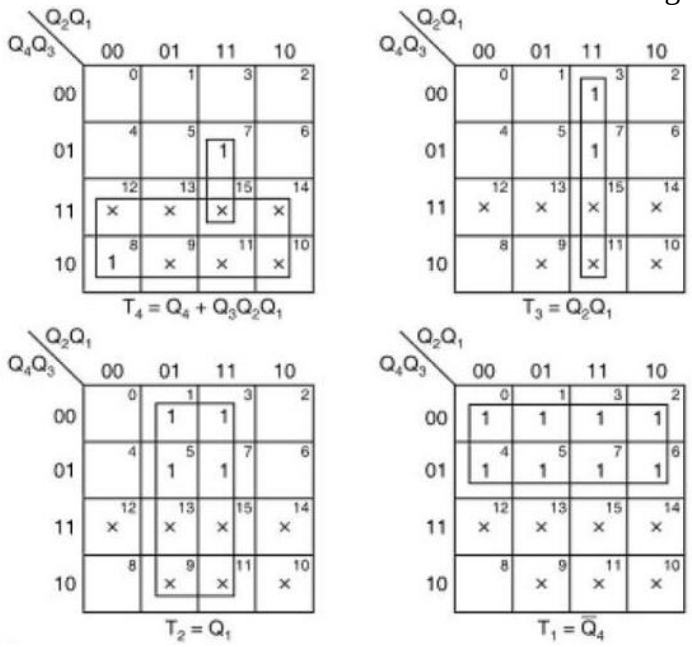


Figure: K-map minimization of synchronous modulo-9 counter using T- flip flops

Step5: Draw the logic diagram: A logic diagram using those minimal expressions can be drawn as shown in the below figure.

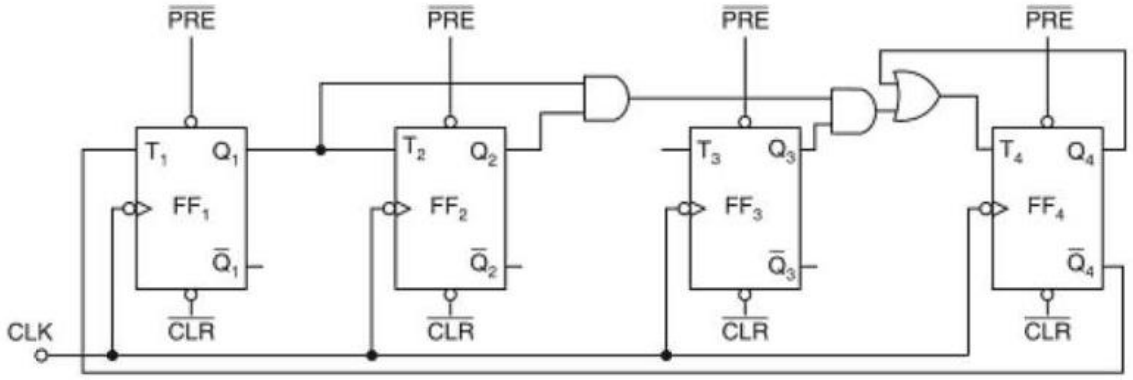


Figure: Logic diagram of synchronous modulo-9 counter using T- flip flops

REGISTERS

An *n*-bit register is a cascade of *n* flip-flops and can store an *n*-bit binary data.

BIDIRECTIONAL SHIFT REGISTER:

A bidirectional shift register is one which the data bits can be shifted from left to right or from right to left. A figure shows the logic diagram of a 4-bit serial-in, serial out, bidirectional shift register. Right/left is the mode signal, **when right /$\overline{\text{left}}$ is a 1**, the logic circuit works as a **shift right-register**. When **right /$\overline{\text{left}}$** is a 0, the logic circuit works as a **shift left-register**.

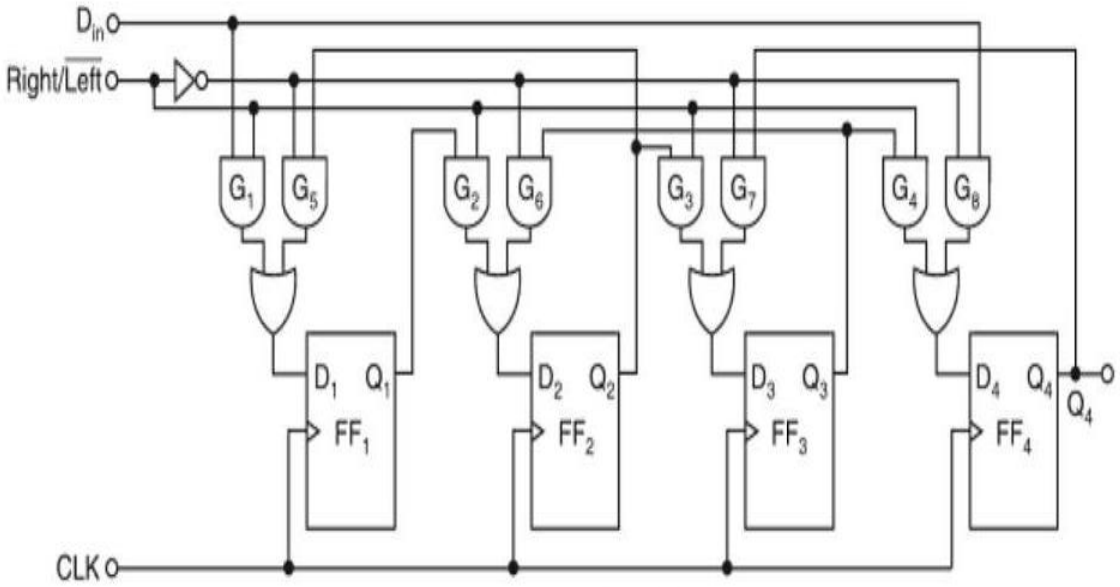


Figure: Logic diagram of 4 bit bidirectional shift register

The bidirectional operation is achieved by using the mode signal and two AND gates and one OR gate for each stage. A **HIGH** on the **right /$\overline{\text{left}}$** control input **enables** the AND gates **G1, G2, G3 and G4** and **disables** the AND gates **G5, G6, G7 and G8**, and the **state of Q output of each FF is passed through the gate to the D input of the following FF.** when a clock pulse occurs**, the data bits** are then effectively **shifted one place to the right**.

A LOW on the **right /$\overline{\textbf{left}}$** control input enables the AND gates G5, G6, G7 and G8 and disables the AND gates G1, G2, G3 and G4 and the Q output of each FF is passed to the D input of the preceding FF. when a clock pulse occurs, the data bits are then effectively shifted one place to the left. Hence, the circuit works as a bidirectional shift register.

UNIVERSAL SHIFT REGISTER:

A register is capable of shifting in one direction only is a **unidirectional shift register**. One that can shift both directions is **a bidirectional shift register**. If the register has both shifts and parallel load capabilities, it is referred to as a **universal shift registers**. Universal shift register is a bidirectional register, whose **input** can be either in **serial form or in parallel form** and whose **output** also can be in **serial form or in parallel form**.

The most general shift register has the following capabilities.

1. A clear control to clear the register to 0.
2. A clock input to synchronize the operations.
3. A shift-right control to enable the shift-right operation and serial input and output lines associated with the shift-right.
4. A shift-left control to enable the shift-left operation and serial input and output lines associated with the shift-left.
5. A parallel loads control to enable a parallel transfer and the n input lines associated with the parallel transfer.
6. N parallel output lines
7. A control state that leaves the information in the register unchanged in the presence of the clock.

A universal shift register can be realized using multiplexers. The below figure shows the logic diagram of a 4-bit universal shift register that has all capabilities.

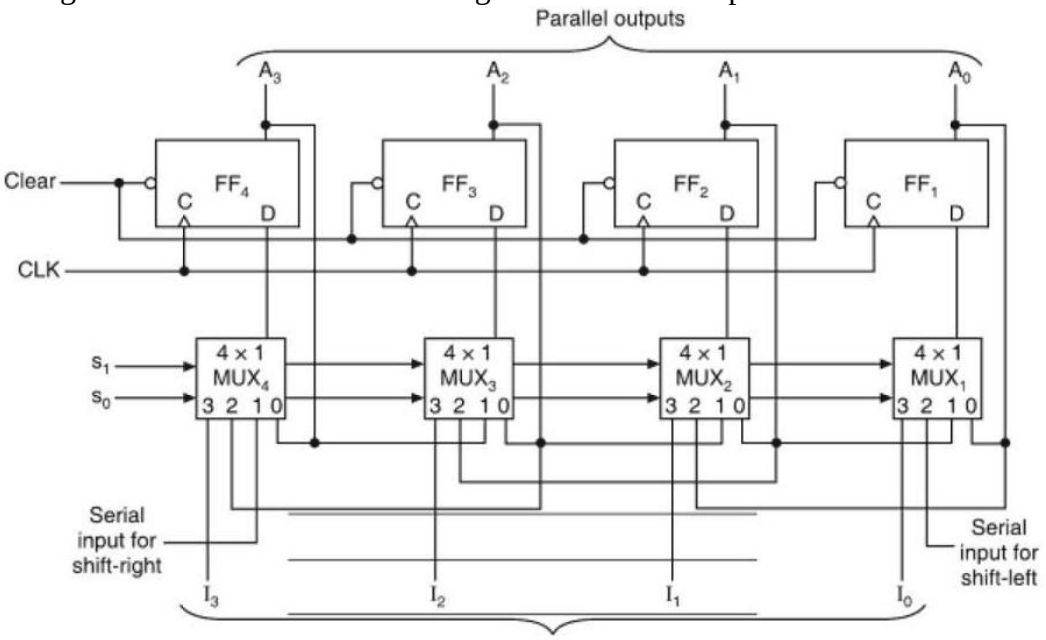


Figure: 4-bit universal shift register

It consists of 4 D flip-flops and four multiplexers. The four multiplexers have two common selection inputs S_1 and S_0. **Input 0** in each multiplexer is selected when **S_1S_0=00**,

input 1 is selected when **S_1S_0=01** and **input 2** is selected when **S_1S_0=10** and **input 3** is selected when **S_1S_0=11**.

The selection inputs control the mode of operation of the register according to the functions entries. **When S_1S_0=0**, the **present value of the register is applied to the D inputs of flip-flops**. The condition forms a path from the output of each flip-flop into the input of the same flip-flop. The next clock edge transfers into each flip-flop the binary value it held previously, and no change of state occurs.

When **S_1S_0=01**, terminal 1 of the multiplexer inputs have a path to the D inputs of the flip-flop. This causes **a shift-right operation**, with serial input transferred into flip-flop4.

When **S_1S_0=10, a shift left operation** results with the other serial input going into flip-flop A1.

Finally, when S_1S_0=11, the binary information on the parallel input lines is transferred into the register simultaneously during the next clock edge.

Mode control		
S_1	S_0	Register operation
0	0	No change
0	1	Shift right
1	0	Shift left
1	1	Parallel load

Applications of Shift Registers:

1. Time Delays
2. Serial / Parallel data conversion
3. Ring Counters
4. UART (Universal asynchronous receiver transmitter)
5. The shift registers are used for temporary data storage.
6. The shift registers are also used for data transfer and data manipulation.
7. The serial-in serial-out and parallel-in parallel-out shift registers are used to produce time delay to digital circuits.
8. The serial-in parallel-out shift register is used to convert serial data into parallel data thus they are used in communication lines where de-multiplexing of a data line into several parallel line is required.
9. A Parallel in Serial out shift register us used to convert parallel data to serial data.