MANIPAL UNIVERSITY

INSPIRED BY LIFE

DEPARTMENT OF INFORMATION & COMMUNICATION TECHNOLOGY

MANIPAL INSTITUTE OF TECHNOLOGY MANIPAL

Digital System Design Lab

Second Year B. Tech. (IT/CCE) Degree

(2024-25) Batch

INDEX

Sample lab observation note preparation

Title: VERIFICATION OF BOOLEAN THEOREMS **Date:**

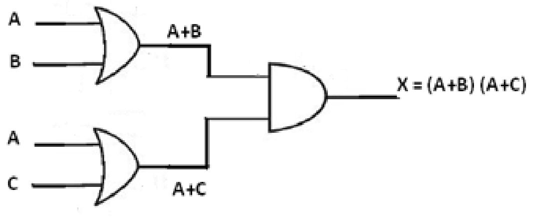
Verify the Boolean theorem i.e., distributive law: A + BC = (A + B) (A + C)

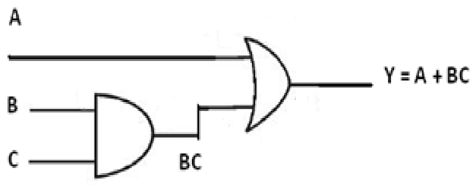
Aim: To prove the Boolean theorem theoretically and verify the same using 2 input AND gate and 2 input OR gates.

Requirements: IC 7408, 7432 [Refer Appendix 1]

Proof: A + BC = A.1 + BC [Since, A.1 = A]
= A(1 + B) + BC [Since, B+1 = 1]
= A.1 + AB + BC
= A(1 + C) + AB + BC[Since, A.A = A.1 = A]
= A (A + C) + B (A+C)
A+BC = (A+B) (A+C)

Circuit Diagram:





Sample input and output:

Input A	Input B	Input C	Output X	Output Y
0	1	0	**0**	**0**
1	0	0	**1**	**1**
1	1	1	**1**	**1**

We infer that X = Y for various input combinations of A, B & C.
Therefore, distributive law holds good for Boolean Expression.

LAB NO: 1 **Date:**

VERIFICATION OF BOOLEAN THEOREMS AND DE'MORGANS LAWS

Objectives:

In this lab, student will be able to:

- Prove Boolean theorems and verify the same using the kit.
- State and explain De' Morgan's laws and verify the same.
- Reduce simple Boolean expressions to simplified form using Boolean theorems and De' Morgan's laws and verify the same.

I. BOOLEAN THEOREMS

Boolean algebra is a mathematical system consisting of a set of two or more distinct elements. The postulates of Boolean algebra are given below.

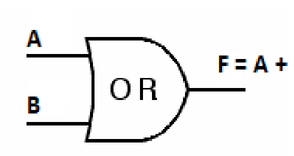
a)	$A+0 = A$	c)	$A+A = A$	e)	$A.1 = A$	g)	$A.A = A$
b)	$A+1 = 1$	d)	$A+\bar{A} = 1$	f)	$A.0 = 0$	h)	$A.\,\bar{A} = 0$

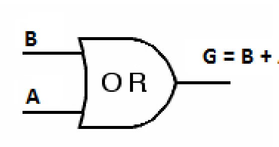
The laws of Boolean algebra are explained below.

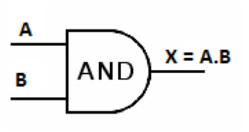
1. **Commutative law:** Using OR operation the Law is given as - A + B = B + A and using AND operation, this law is given as: A.B = B.A. According to this law, order of the OR / AND operations conducted on the variables makes no difference.

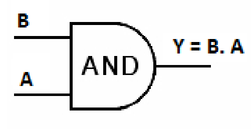
Rig up the circuit for F,G,X and Y as shown in the figure below. Complete the truth table. What do you observe? [Refer Appendix for Pin diagram of relevant IC]

A	B	F = A+B	G = B + A	X = A.B	Y = B.A
0	0				
0	1				
1	0				
1	1				





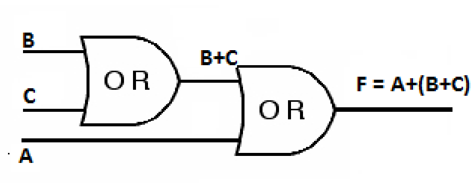


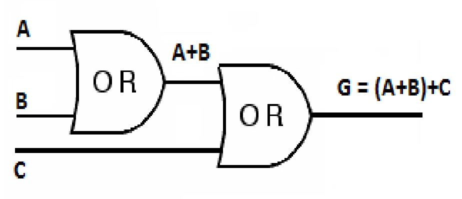


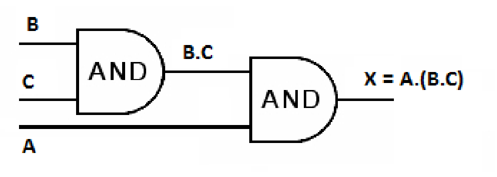
2. **Associative law:** This law is given as - A+(B+C) = (A+B)+C with OR operator and A.(B.C) = (A.B).C with AND operator. According to this law, grouping of Boolean expressions do not make any difference during the OR / AND operation of several variables.

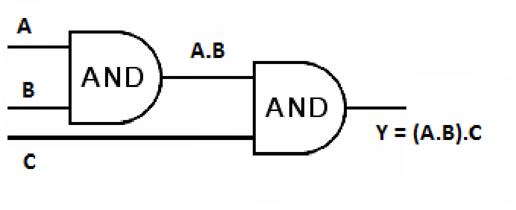
Rig up the circuit for F,G,X and Y as shown in the figure below. Complete the truth table. What do you observe? [Refer Appendix for Pin diagram of relevant IC]

A	B	C	B+ C	F=A + (B+C)	A + B	G=(A+B)+C	B.C	X = A. (BC)	A.B	Y= (A.B).C
0	0	0								
0	0	1								
0	1	0								
0	1	1								
1	0	0								
1	0	1								
1	1	0								
1	1	1								









3. **Distributive law:** The law is A + BC = (A + B)(A + C). This law is composed of logical AND and logical OR operators. Here, AND operation of several variables, followed by OR operation of the result with a single variable, is equivalent to the AND of the OR of single variable to one of the variable of several variables. The proof of this law in Boolean algebra is given below, considering three variables A,B and C:

Proof: A + BC = A.1 + BC [Since, A.1 = A]
= A(1 + B) + BC [Since, B+1 = 1]
= A.1 + AB + BC
= A.(1 + C) + AB + BC [Since, A.A = A.1 = A]
= A (A + C) + B (A+C)
Thus, A+BC = (A+B)(A+C).

A	B	C	B C	F=A+(BC)	A+B	A+C	G=(A+B).(A+C)	B+C	F= A(B+C)	G = AB+AC
0	0	0								
0	0	1								
0	1	0								
0	1	1								
1	0	0								
1	0	1								
1	1	0								
1	1	1								

This law for Boolean multiplication is given as- A.(B + C) = A.B + A.C. Refer sample lab program for verification.

4. Absorption laws: Absorption laws are a group of laws.

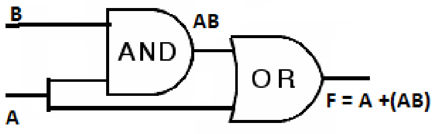
Rig up the circuit as shown in the figure below. Complete the truth table. What do you observe? [Refer Appendix for Pin diagram of relevant IC]

i) A+AB = A

Proof: A+AB = A.1 + AB [A.1 = A]

= A(1+B) [Since, 1 + B = 1]

= A.1 = A



A	B	A.B	F = A +(A.B)
0	0		
0	1		
1	0		
1	1		

ii) A(A+B) = A

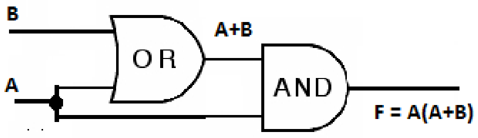
Proof: A(A+B) = A.A + A.B

= A+AB

= A(1+B)

= A.1

= A



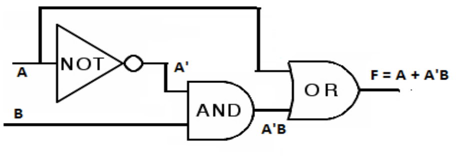
A	B	A + B	F = A.(A+B)
0	0		
0	1		
1	0		
1	1		

iii) A+ĀB = A+B

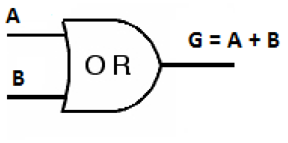
Proof: A+ĀB = (A+Ā) [Since, A+BC = (A+B)(A+C) using distributive law.]

= 1 (A+B) [Since, A+Ā = 1]

= A+B



A	B	A’	A’B	F = A+(A’B)	G =A+B
0	0	1			
0	1	1			
1	0	0			
1	1	0			

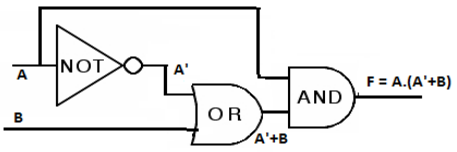


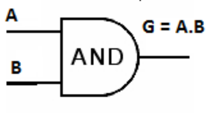
iv) A.(Ā+B) = AB

Proof: A.(Ā+B) = A. Ā+AB

= AB [AĀ = 0]

A	B	A’	A’+B	F = A(A’+B)	G =A.B
0	0	1			
0	1	1			
1	0	0			
1	1	0			





5. **Consensus Laws:** This is other group of laws which are given more priority than the theorems of Boolean algebra.

Rig up the circuit as shown in the figure below. Complete the truth table. What do you observe? [Refer Appendix for Pin diagram of relevant IC]

a) AB + ĀC+BC = AB+ĀC

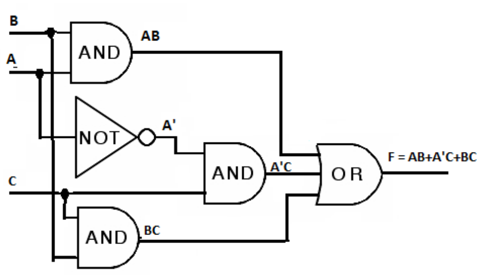
Proof: AB+ĀC+BC = AB + ĀC + BC.1

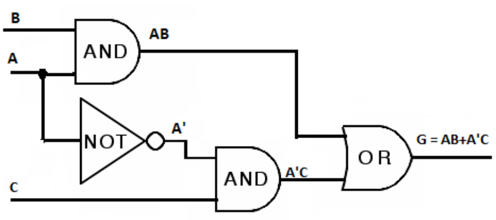
= AB+ĀC+BC(A+Ā) [A+Ā=1]

= AB+ĀC+ABC+ĀBC

= AB(1+C)+ ĀC(1+B) [1+B=1=1+C]
= AB+ĀC

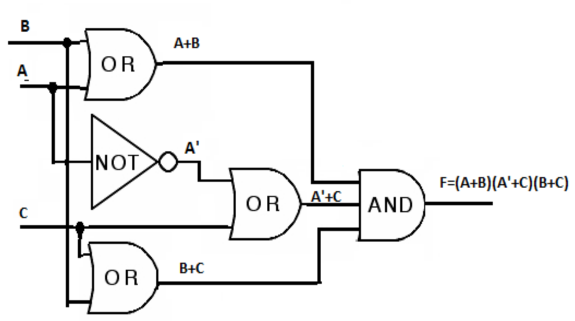
A	B	C	A’	AB	A’C	BC	F= AB+A’C+BC	G=AB+A’C
0	0	0						
0	0	1						
0	1	0						
0	1	1						
1	0	0						
1	0	1						
1	1	0						
1	1	1						

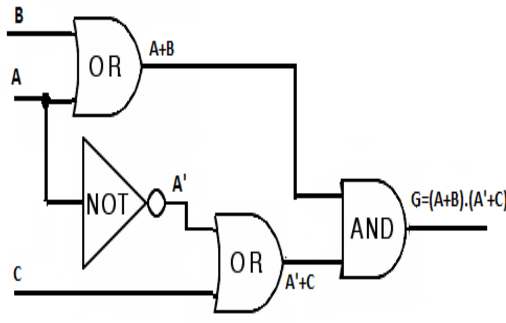




b) (A+B)(Ā+C)(B+C) = (A+B)(Ā+C)
Proof:(A+B)(Ā+C)(B+C) = (A+B)(Ā+C)(B+C+0)
= (A+B)(Ā+C)(B+C+AĀ) [By distributive law]
= (A+B)(A+B+C)(Ā+C)(Ā+C+B)
= (A+B)(Ā+C) [A(A+B)= A]

A	B	C	A’	A+B	A’+C	B+C	F= (A+B)(A’+C)(B+C)	G=(A+B)(A’+C)
0	0	0						
0	0	1						
0	1	0						
0	1	1						
1	0	0						
1	0	1						
1	1	0						
1	1	1						





Thus we have completed the laws of Boolean algebra.

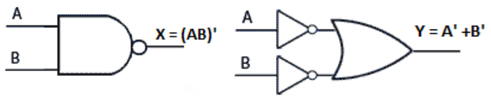
.

II. DE'MORGANS LAWS

Theorem 1:

The complement of product of two variables is equal to the sum of complement of each variable. Thus according to De-Morgan's laws or De-Morgan's theorem, if A and B are the two variables or Boolean numbers then,

$\overline{A.B} = \overline{A} + \overline{B}$

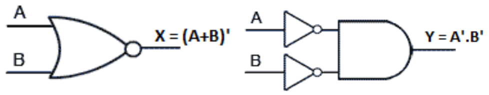


A	B	A'	B'	AB	X=(AB)'	Y=A'+B'
0	0	1				
0	1	1				
1	0	0				
1	1	0				

Theorem 2:

The complement of sum of two variables is equal to the product of complement of each variable. Thus according to De Morgan's theorem, if A and B are the two variables then,

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



A	B	A'	B'	A+B	X=(A+B)'	Y=A'.B'
0	0	1				
0	1	1				
1	0	0				
1	1	0				

These can be verified using truth table. De' Morgan's laws can be generalized for more than two variables as shown below.

$\overline{A.B.C} = \overline{A} + \overline{B} + \overline{C}$

$\overline{(A+B+C)} = \overline{A}.\overline{B}.\overline{C}$

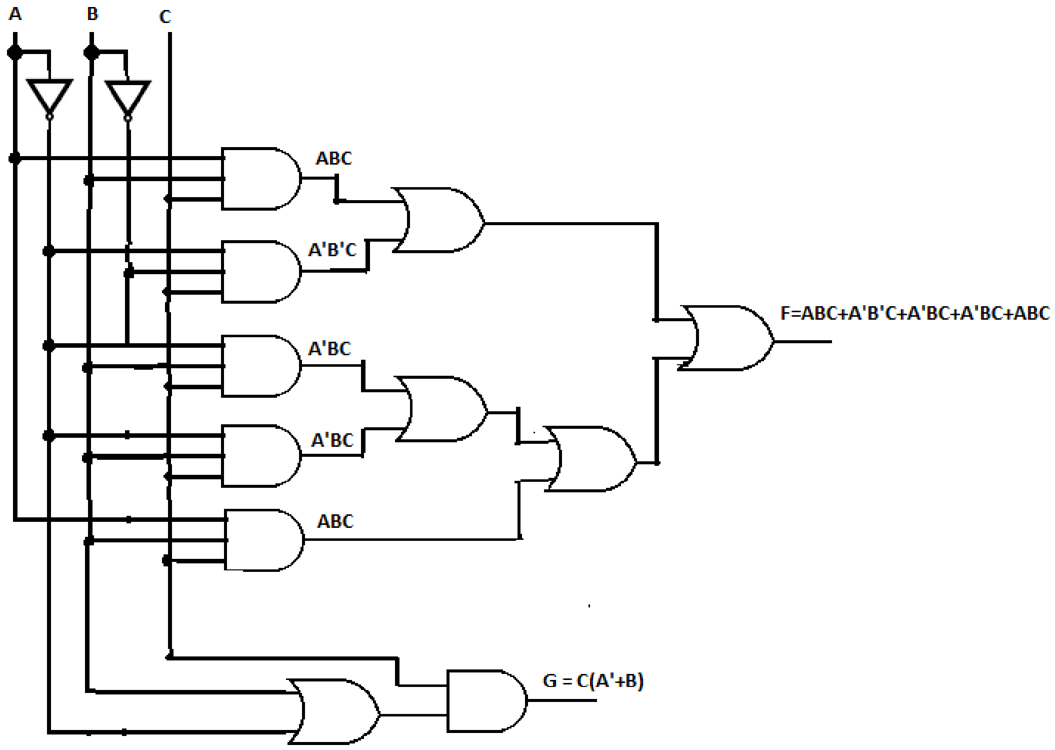
III. REDUCTION OF BOOLEAN EXPRESSIONS USING THEOREMS AND LAWS

Solved Exercise

i. **ABC + A'B'C + A'BC + A'BC + ABC**

= ABC + A'B'C + A'BC	[A + A = A]
= ABC + A'C (B'+B)	[B' + B = 1]
= C (A +A') (B + A')	[A' + A = 1]
= C (B + A')	

Circuit Diagram: Let F = ABC + A'B'C+A'BC+A'BC+ABC and let G = C (B+A')



Observation:

A	**B**	**C**	**A'**	**B'**	**ABC**	**A'B'C**	**A'BC**	**ABC**	**F**	**A'+B**	**G**
0	0	0									
0	0	1									
0	1	0									
0	1	1									
1	0	0									
1	0	1									
1	1	0									
1	1	1									

Note that simplification of Boolean expression help to reduce the fan out and propagation delay of the circuit.

Lab exercise

a. Simplify the following expressions using Boolean theorems and implement using basic gates. Consider the given expression as 'F' and simplified expression as 'G'. [Refer Appendix for pin diagram of ICs]

i. F = (x + y) (x + y')

Hardware Requirements:

Truth Table:

X	**Y**	**X+Y**	**X+Y'**	**F**	**G (Simplified Expression)**
0	0				
0	1				
1	0				
1	1				

ii. F = y (wz’ + wz) + xy

Sol: Simplified Expression:

Hardware Requirements:

Truth Table:

W	**X**	**Y**	**Z**	**F**	**G (Simplified Expression)**
0	0	0	0		
0	0	0	1		
0	0	1	0		
0	0	1	1		
0	1	0	0		
0	1	0	1		
0	1	1	0		
0	1	1	1		
1	0	0	0		
1	0	0	1		
1	0	1	0		
1	0	1	1		
1	1	0	0		
1	1	0	1		
1	1	1	0		
1	1	1	1		

iii. F = [(CD)’ + A] +A +C’D+AB

Sol: Simplified Expression:

Circuit Diagram:

Hardware Requirements:

Truth Table:

A	B	C	D	F	G (Simplified Expression)
0	0	0	0		
0	0	0	1		
0	0	1	0		
0	0	1	1		
0	1	0	0		
0	1	0	1		
0	1	1	0		
0	1	1	1		
1	0	0	0		
1	0	0	1		
1	0	1	0		
1	0	1	1		
1	1	0	0		
1	1	0	1		
1	1	1	0		
1	1	1	1		

LAB NO: 2 **Date:**

SIMPLIFICATION OF BOOLEAN EXPRESSIONS USING K MAP

Objectives:

In this lab, student will be able to:

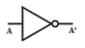
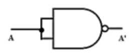
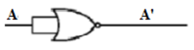
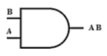
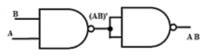
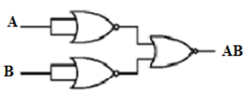
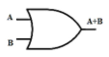
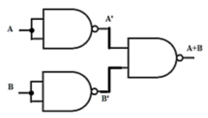
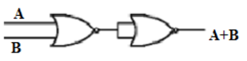
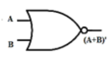
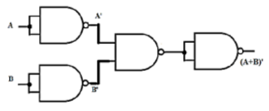
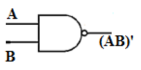
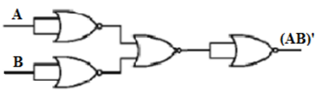
1. Reduce the given Boolean expressions using K-map to simplified form
2. Implement the simplified expression using universal gates
3. Reduce the Boolean expressions using don't care condition to SOP and POS form.

Introduction

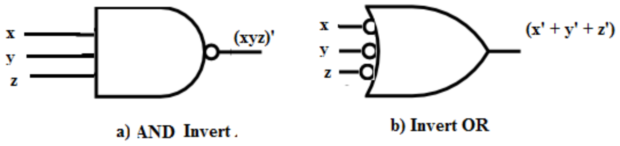
Universal gates

Digital circuits are frequently constructed with NAND or NOR gates rather than with AND or OR gates. NAND and NOR gates are easier to fabricate with electronic components and are used in all IC digital logic families. NAND and NOR gates are called universal gates as any other basic gate can be constructed using these gates.

The basic AND, OR, NOR, and NOT gates can be implemented using NAND gates only or NOR gates only as shown below

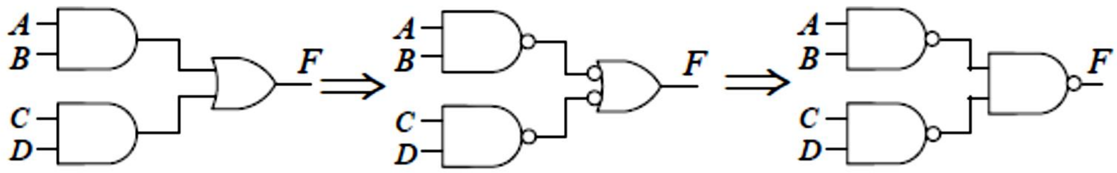
BASIC GATE	EQUIVALENT NAND CIRCUIT	EQUIVALENT NOR CIRCUIT
		
		
		
		
		

Two equivalent graphic symbols for NAND gate are shown below:

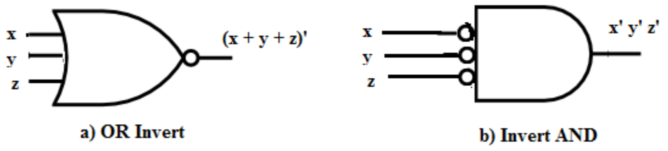


Example 1: Implement F = AB + CD using NAND gates. *The implementation of Boolean functions with NAND gates requires that, the functions be in sum-of-product (SOP) form.*

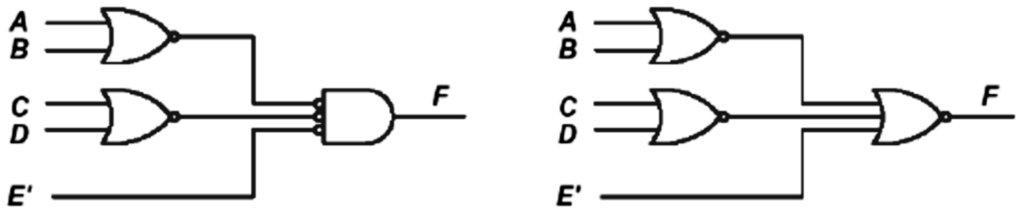
$$F = \overline{\overline{AB + CD}} = \overline{(\overline{AB} \bullet \overline{CD})}$$



Two equivalent graphic symbols for the NOR gate, are shown below:



Example 2: Implement F=(A+B) (C+D) E using OR/AND gates then using NOR gates only. *The implementation of Boolean functions with NOR gates requires that, the functions be in product-of-sum (POS) form.*



Simplification of Boolean expression

A product term is a term with AND-ed literals. Thus, AB, A'B, AB' and A'B' are all product terms. A minterm is a special case of a product term, where all input variables appear in the product term, either in the true or complement form. A sum term is a term with OR-ed literals. Thus, (A+B), (A'+B), (A+B') and (A'+B') are all sum terms. A maxterm is a special case of a sum term where all input variables, either in the true or complement form, are OR-ed together.

The minterm (m_i) list contains the numbers of the rows of the truth table for which the output Q = 1. The maxterm (M_i) list contains the numbers of the rows of the truth table for which the output Q = 0. In general, for n-input variables, the number of Maxterms is equal to the total number of possible input combinations 2^n. Using De-Morgan's theorem, or truth tables, it can be easily shown that $M_i=\overline{m_i};\forall=0, 1, 2\ldots, (2^n-1)$.

The expression in which any function can be represented by OR-ing all minterms (m_i) corresponding to input combinations (i) at which the function has a value of 1 is commonly referred to as the SUM of minterms and is typically expressed as $F = \sum (m_i, m_j\ldots, m_k)$, where '$\sum$' indicates OR-ing of the indicated minterms.

Example3: Function $F = \sum (2, 4, 5, 7) = (m_2 + m_4 + m_5 + m_7)$
Using De-Morgan theorem, $\overline{F}=\overline{m2+m4+m5+m7}=\overline{m2}.\overline{m4}.\ \overline{m5}.\overline{m7} = M_2.M_4.M_5.M_7$.

This represents product of Maxterms for the complement form of function F. In general, $F = \sum (2, 4, 5, 7) = \prod (0, 1, 3, 6)$ where $\prod$ notation signifies product of Maxterms. The sum of minterms (SoM) and the product of maxterms (PoM) forms of Boolean expressions are known as *the canonical forms*. These forms are rarely the ones with the least number of literals, because each minterm or maxterm must contain, by definition all the variables either complemented or un-complemented form. Boolean functions can also be expressed in the form of a Sum of Products (SOP) or in the form of a Product of Sums (POS).In this configuration; the terms that form the function may contain one, two or any number of literals.

The sum of minterms form is a special case of the SOP form where all product terms are minterms. The product of maxterms form is a special case of the POS form where all sum terms are maxterms. *The SOP and POS forms are Standard forms* for representing Boolean functions. Any SOP expression can be implemented in 2-levels of gates. The first level consists of a number of AND gates which equals the number of product terms in the expression. Each AND gate implements one of the product terms in the expression. The second level consists of a SINGLE OR gate whose number of inputs equals the number of product terms in the expression. Similarly, any POS expression can be implemented in 2-levels of gates. The first level consists of a number of OR gates which equals the number of sum terms in the expression, each gate implements one of the sum terms in the expression. The second level consists of a SINGLE AND gate whose number of inputs equals the number of sum terms.

K map: The Karnaugh map, also known as the K-map, is a method to simplify Boolean algebraic expressions. A Karnaugh map provides a pictorial method of grouping together expressions with common factors and therefore eliminating unwanted variables. It is a diagram made up of squares, with each square representing one minterm/maxterm of the function that is to be minimized. The diagram below illustrates the correspondence between the Karnaugh map and the truth table for the general case of a *two variable problem*.

Truth Table:

A	B	F
0	0	A
0	1	B
1	0	C
1	1	D

Kmap:

	0	1
0	a	b
1	c	d

A three-variable K-map is shown below. There are eight minterms for three binary variables (x, y, z); therefore, the map consists of eight squares. The characteristic of sequence of minterms represented in K-map below is that only one bit changes in value from one adjacent column to the next.

x\yz	y'z’	y'z	yz	yz'
x'	x'y’z’	x'y’z	x'yz	x'yz’
x	xy’z’	xy’z	xyz	xyz’

Reduction of Boolean expression using K map
One square represents one minterm giving a term with three literals. (Ex: x’yz)
Two adjacent squares represent a term with two literals. (Ex: xy)
Four adjacent squares represent a term with one literal. (Ex: y)
Eight adjacent squares encompass the entire map producing a function that is always equal to 1

The K-map for Boolean functions of four binary has four input variables (w, x, y, z) and sixteen minterms/maxterms. Therefore there are sixteen squares in a *four variable K-map.*
Reduction of Boolean expression using K map
One square represents one minterm giving a term with four literals.
Two adjacent squares represent a term with three literals.
Four adjacent squares represent a term with two literals.
Eight adjacent squares represent a term with one literal.
Sixteen adjacent squares produce a function that is always equal to 1.

wx\yz	y'z’	y'z	yz	yz'
w'x'	w'x'y’z’	w'x'y’z	w'x'yz	w'x'yz’
w'x	w'xy’z’	w'xy’z	w'xyz	w'xyz’
wx	wxy’z’	wxy’z	wx yz	wxyz’
wx'	wx'y’z’	wx'y’z	wx'yz	wx'yz’

Sometimes, there may be two or more expressions that satisfy the simplification criteria. In this case, selection of simplified expression has to done such that there are no redundant groups. A prime implicant is a product term obtained by combining the maximum possible number of adjacent squares in the map. If a minterm in a square is covered by only one prime implicant, that prime implicant is said to be essential. The prime implicants of a function can be obtained from the map by combining all possible maximum numbers of squares.

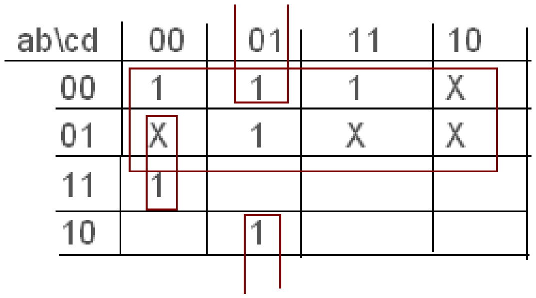
Don’t care conditions

In practice, in some applications the function is not specified for certain combinations of the variables. Such functions are called incompletely specified functions. In these cases, we simply don't care what value is assumed by the function for the unspecified minterms. For this reason, it is customary to call the unspecified minterms of a function as don't care conditions. These don't -care conditions can be used on a map to provide further simplification of the Boolean expression.

A don't-care minterm is a combination of variables whose logical value is not specified. Such a minterm cannot be marked with a 1 in the map because it would require that the function always be a 1 for such a combination. Likewise, putting a 0 on the square requires the function to be 0. To distinguish the don't-care condition from l's and 0's, an ‘X’ or ‘D’ is used. Thus, an ‘X’ inside a square in the map indicates that we don't care whether the value of 0 or 1 is assigned to F for the particular minterm. When simplifying the function, we can choose to include each don't -care minterm with either the 1s or the 0's, depending on which combination gives the simplest expression.

Solved Exercise

Let F =Σ(0,1,3,5,9,12)+Σd(2,4,6,7). The 4 variable K-map for F is shown below



ab\cd	00	01	11	10
00	1	1	1	X
01	X	1	X	X
11	1			
10		1		

Simplified SOP Expression: F=a'+bc'd'+b'c'd. Draw the circuit using NAND gates only

Get the simplified POS expression from the K-map and draw the circuit using NOR gates only.

Hardware Requirements:

Truth table:

a	b	c	d	a’	b’	c’	d’	F for SOP expression	F for POS expression
0	0	0	0	1	1	1	1		
0	0	0	1	1	1	1	0		
0	0	1	0	1	1	0	1		
0	0	1	1	1	1	0	0		
0	1	0	0	1	0	1	1		
0	1	0	1	1	0	1	0		
0	1	1	0	1	0	0	1		
0	1	1	1	1	0	0	0		
1	0	0	0	0	1	1	1		
1	0	0	1	0	1	1	0		
1	0	1	0	0	1	0	1		
1	0	1	1	0	1	0	0		
1	1	0	0	0	0	1	1		
1	1	0	1	0	0	1	0		
1	1	1	0	0	0	0	1		
1	1	1	1	0	0	0	0		

Complete the truth table. Rig up the circuit following the logic diagram and compare the output with the truth table.

Lab exercises

1. Simplify the Boolean function ‘F’ using K maps & implement using NAND gates only
   F = x’z’+y’z’+yz’+xyz

2. Simplify the Boolean function ‘F’ using don’t care conditions in ‘D’ and realize the circuit using only NOR gates
   F = (a’+b+d) (c+d) (c’+d’)
   D = (a+b’+c+d’) (a’+b+c+d’)

Hardware Requirements:

Additional exercises

Simplify the following questions using K maps & implement using

a. NAND gates only
b. NOR gates only.
   i. F(w,x,y,z) = ∑(2,3,12,13,14,15) + D(6,7,8,9)
   ii. F(a,b,c,d) = π(1,4,5,7,9,10,11) + D(14,15)
   iii. F(a,b,c,d) = a’d + bd + b’c + ab’d

LAB NO: 3 **Date:**

DESIGN OF COMBINATIONAL LOGIC CIRCUITS

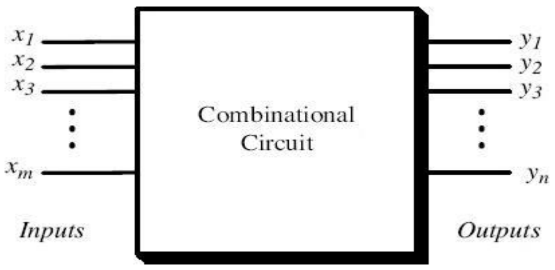
Objectives:

In this lab, student will be able to

1. Differentiate between combinational and sequential logic circuits.
2. Design a combinational logic circuit using basic gates

Introduction:

Combinational logic refers to circuits whose output is a function of the present value of the inputs only. *Sequential logic* circuits are those whose outputs are also dependent upon past inputs, and hence outputs. In other words, the output of a sequential circuit may depend upon its previous outputs and so in effect, has some form of "memory". A combinational circuit consists of input variables, logic gates, and output variables. The logic gates react to the values of the signals at their inputs and produce the value of the output signal, transforming binary information from the given input data to a required output data. The diagram of a combinational circuit has logic gates with no feedback paths or memory elements. A feedback path is a connection from the output of one gate to input to the first gate.



The figure above depicts simple block of combinational logic circuit. The ‘m’ input binary variable come from an external source; the ‘n’ output variables are produced by the internal combinational logic circuit and go to an external destination. For ‘m’ input variables there are 2^m possible binary input combinations. For each possible input combination there is one possible output value.

The design of combinational circuits starts from the specification of the design objective and culminates in a logic circuit diagram or a set of Boolean functions from which the logic diagram can be obtained. The procedure involves the following steps:

1. From the specifications of the circuit determine the required number of inputs and outputs and assign a symbol to each.
2. Derive the truth table that defines the required relationship between inputs and outputs
3. Obtain the simplified Boolean functions for each output as a function of the input variables using any reduction method.
4. Draw the logic diagram and verify the correctness of the design (by simulation).

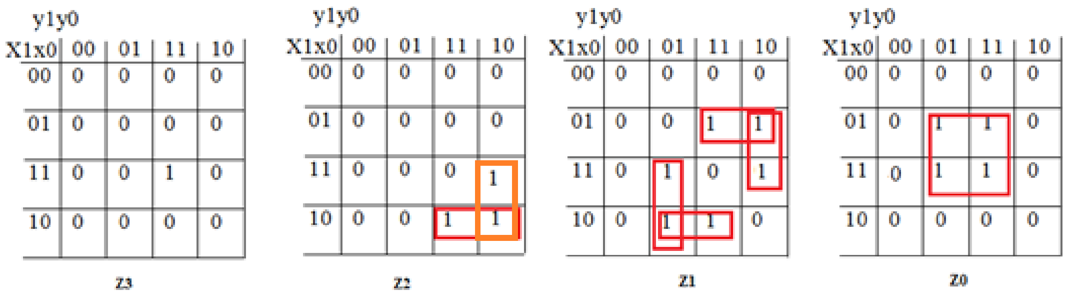
Solved Exercise

Design a combinational circuit that accepts two 2 bit numbers and displays product of two numbers as the result.

Truth Table: X_1 X_0 and Y_1 Y_0 represents two 2 bit numbers. The product is represented as $Z_3Z_2Z_1Z_0$

X_1	X_0	Y_1	Y_0	Z_3	Z_2	Z_1	Z_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	1
0	1	1	0	0	0	1	0
0	1	1	1	0	0	1	1
1	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0
1	0	1	0	0	1	0	0
1	0	1	1	0	1	1	0
1	1	0	0	0	0	0	0
1	1	0	1	0	0	1	1
1	1	1	0	0	1	1	0
1	1	1	1	1	0	0	1

K- maps:



$Z_3 = X_1X_0Y_1Y_0$
$Z_2 = X_1X_0'Y_1 + X_1Y_1Y_0'$
$Z_1 = X_1'X_0Y_1 + X_0Y_1Y_0' + X_1X_0'Y_0 + X_1Y_1'Y_0$
$Z_0 = X_0Y_0$

Hardware Requirements:

Lab exercises

1. Design a combinational circuit that accepts a 3 bit number and generate an output binary numbers equal to square of input number. [Hint : Since the maximum 3 bit number is 7, and square of it is 49, the number of output lines required is 6]. Realize using basic logic gates

   Truth Table:

X	Y	Z	P_5	P_4	P_3	P_2	P_1	P_0
0	0	0						
0	0	1						
0	1	0						
0	1	1						
1	0	0						
1	0	1						
1	1	0						
1	1	1						

   Simplified SOP expressions:

   Hardware Requirements:

2. Design a combinational circuit with 4- input lines that represents a decimal digit in BCD and 4- output lines that generates 2's complement of input digit.[Hint: 2's complement of a given binary number is obtained by adding one to 1's complement of the number, where 1's complement of the binary number is obtained by inverting the bits]

   Truth Table:

A	B	C	D	W	X	Y	Z
0	0	0	0				
0	0	0	1				
0	0	1	0				
0	0	1	1				
0	1	0	0				
0	1	0	1				
0	1	1	0				
0	1	1	1				
1	0	0	0				
1	0	0	1				

Realize using NAND gates only

Hardware Requirements:

3. Design a combinational circuit to check for even parity of 4 bits. A logic '1' output is required when the 4 bits constitute an even parity. [Hint: Parity denotes the total number of 1's in the input sequence. To indicate the given number has even parity, the parity bit will be one if the total number of 1's in the input is an even number. Ex: If input is 1100, the parity bit is 1 and if input is 1101, parity bit is 0]. Realize using Ex-OR gates

Truth Table:

A	B	C	D	P
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

Simplified SOP expression:
Hardware Requirements:

Additional exercises

1. Design a combinational circuit that multiplies by '5' an input decimal digit represented in BCD. The output is also in BCD.
2. Design a 4 input, 1 output combinational circuit whose output is HIGH when majority of inputs are HIGH.

LAB NO: 4 **Date:**

IMPLEMENTATION OF CODE CONVERTERS

Objectives:

In this lab, student will be able to:

1. Differentiate between various representation formats of a given number
2. Convert one representation of code to another representation using universal gate

Introduction:

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 that performs this transformation by means of logic gates is called as a code converter. There are different representation format for a given decimal digit, few of which are shown below.

Digit	BCD in 8421	Gray Code	Excess-3	BCD	8 4 -2 -1	2 4 2 1
0	0000	0000	0011	0000	0000	0000
1	0001	0001	0100	0001	0111	0001
2	0010	0011	0101	0010	0110	0010
3	0011	0010	0110	0011	0101	0011
4	0100	0110	0111	0100	0100	0100
5	0101	0111	1000	0101	1011	1011
6	0110	0101	1001	0110	1010	1100
7	0111	0100	1010	0111	1001	1101
8	1000	1100	1011	1000	1000	1110
9	1001	1101	1100	1001	1111	1111

Don’t care terms:

BCD in 8421	1010, 1011, 1100, 1101, 1110, 1111
Excess – 3	0000, 0001, 0010, 1101, 1110, 1111
8 4 -2 – 1 (self-complementary)	0001, 0010, 0011, 1100, 1101, 1110
2 4 2 1 (self-complementary)	0101, 0110, 0111, 1000, 1001, 1010

In the table, 8 4 2 1, 8 4 -2 -1 and 2 4 2 1 are called weighted binary codes. If the nine's complement of the BCD digit in a code is equal to the one's complement of the code in binary, then it is called a self-complementing code. 8 4 -2 -1 and 2 4 2 1 are called self-complementing weighted binary code and excess-3 is called self-complementing non-weighted binary code.

Solved exercise

Design a combinational logic circuit to convert from gray code to BCD code using basic logic gates.

The truth table of gray and BCD code is given below. In the table D signifies don't care terms. A, B, C and D represent a decimal number represented in Gray code. W, X, Y and Z represents its corresponding BCD number.

Truth Table:

Gray Code				BCD			
A	**B**	**C**	**D**	**W**	**X**	**Y**	**Z**
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	1	0	0	1	0
0	0	1	0	0	0	1	1
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1
0	1	0	1	0	1	1	0
0	1	0	0	0	1	1	1
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	1
1	1	1	1	X	X	X	X
1	1	1	0	X	X	X	X
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	0	0	1	X	X	X	X
1	0	0	0	X	X	X	X

K map for W

	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	X	X
10	X	X	X	X

Simplified expression: W = A
W = ∑ (12,13) + D(8,9,10,11,14,15)

K map for X

	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	X	X
10	X	X	X	X

Simplified expression X = A’B
X = ∑ (4,5,6,7)

K map for Y

	00	01	11	10
00	0	0	1	1
01	1	1	0	0
11	0	0	X	X
10	X	X	X	X

Simplified expression Y = A’BC’ + B’C
Y = ∑ (2,3,4,5) + D(10,11)

K map for Z

	00	01	11	10
00	0	1	0	1
01	1	0	1	0
11	0	1	X	X
10	X	X	X	X

Simplified expression:
Z = (A + BC +B’C’) D + (B’C + A’BC’) D’
To reduce the number of ICs, substitute Y in Z
Z = (A + BC +B’C’) D + (Y) D’

Hardware Requirements:

Lab exercises
Design following code converters
1. BCD to excess-3 using NAND gates only

Truth Table:

Decimal	BCD				Excess -3			
	A	**B**	**C**	**D**	**W**	**X**	**Y**	**Z**
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				

Hardware Requirements:

2. Self-complementary 8 4 -2 -1 to self-complementary 2 4 2 1 using NOR gates only

Truth Table:

Decimal	8	4	-2	-1	2	4	2	1
	A	**B**	**C**	**D**	**W**	**X**	**Y**	**Z**
0	0	0	0	0				
1	0	1	1	1				
2	0	1	1	0				
3	0	1	0	1				
4	0	1	0	0				
5	1	0	1	1				
6	1	0	1	0				
7	1	0	0	1				
8	1	0	0	0				
9	1	1	1	1				

Hardware Requirements:

3. Excess – 3 to 2 4 2 1 using basic logic gates

Truth Table:

Decimal	Excess -3				2	4	2	1
	A	**B**	**C**	**D**	**W**	**X**	**Y**	**Z**
0	0	0	1	1				
1	0	1	0	0				
2	0	1	0	1				
3	0	1	1	0				
4	0	1	1	1				
5	1	0	0	0				
6	1	0	0	1				
7	1	0	1	0				
8	1	0	1	1				
9	1	1	0	0				

Hardware Requirements:

Additional exercises

Design following code converters

1. 2 4 2 1 to 8 4 2 1
2. BCD to 8 4 -2 -1

[SPACE FOR ADDITIONAL EXERCISES]

LAB NO: 5 **Date:**

DESIGN OF ADDERS AND SUBTRACTORS

Objectives:

In this lab, student will be able to:

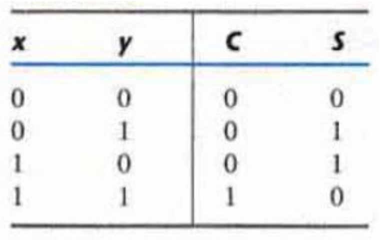
- Design and compare binary full adder and full subtractor

Introduction

Digital computers perform a variety of information-processing tasks. One of the most important tasks performed by a digital computer is the operation of adding two binary numbers. A combinational circuit that performs the addition of two bits is called a half adder and the one that performs the addition of three bits (two significant bits and a previous carry) is a full adder. The third input in a full adder represents the carry from the previous lower significant position. A binary adder- subtractor is a combinational circuit that performs the arithmetic operations of addition and subtraction with binary numbers.

Half adder

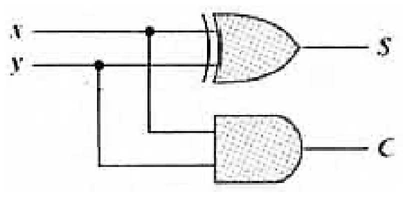
A combinational logic circuit that performs the addition of two data bits, x and y, is called a half-adder. Addition will result in two output bits; one of which is the sum bit S, and the other is the carry bit C. The truth table for the half adder is shown. The C output is I only when both inputs are 1. The S output represents the least significant bit of the sum. The simplified Boolean functions for the two outputs can be obtained directly from the truth table. The simplified sum-of-products expressions are



x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$S = x'y + xy' = x \oplus y$$
$$C = xy$$

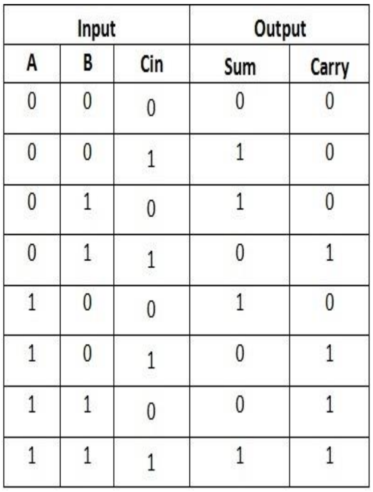
The logic diagram of the half adder implemented in sum of products is shown below.
Rig up the circuit and verify the adder with the truth table.



Full adder

The half-adder does not take the carry bit from its previous stage into account. This carry bit from its previous stage is called carry-in bit. A combinational logic circuit that adds two data bits, A and B, and a carry-in bit, C_{in} , is called a full-adder. The truth table of a full adder with x,y,z as input variables and S, C_{out} as output variables is shown in the table below.

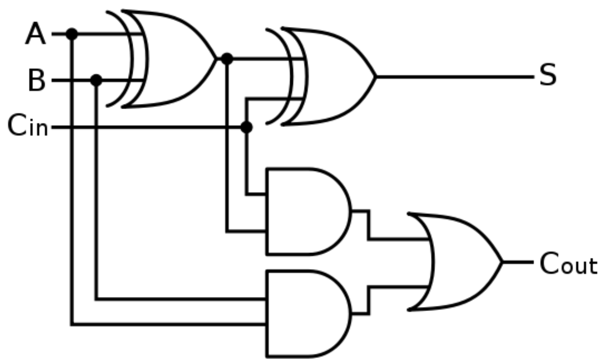
Rig up the circuit and verify the adder with the truth table.



Input			Output	
A	B	Cin	Sum	Carry
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

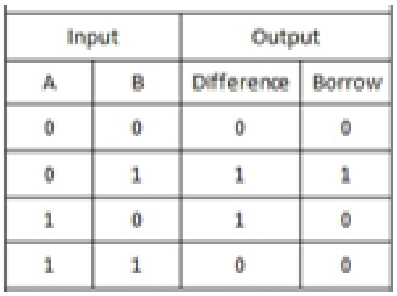
The simplified expression for sum and carry in a full adder is given as: $S = A \oplus (B \oplus C_{in})$

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



Half Subtractor

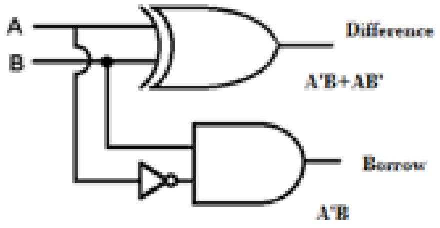
Subtracting a single-bit binary value B from another A (i.e. A -B) produces a Difference bit D and Borrow out bit B_{out}. This operation is called half subtraction and the circuit to realize it is called a half subtractor. The Boolean functions describing the half subtractor are:



Input		Output	
A	B	Difference	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

$D = A \oplus B$

$B_{out} = A'B$



Rig up the circuit and verify with the truth table of half subtractor

Full Subtractor:

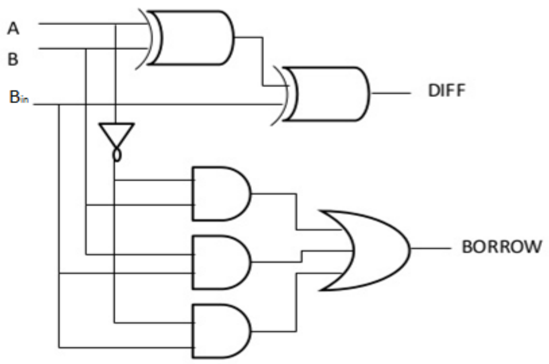
Subtracting two single-bit binary values, B, B_{in} from a single-bit value A, produces a difference bit D and Borrow out bit B_{out}. This is called full subtraction. ***Rig up the circuit and verify with the truth table of Full subtractor.***

The Boolean functions describing the full-subtractor are:

Input			Output	
A	B	Bin	D	Bout
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

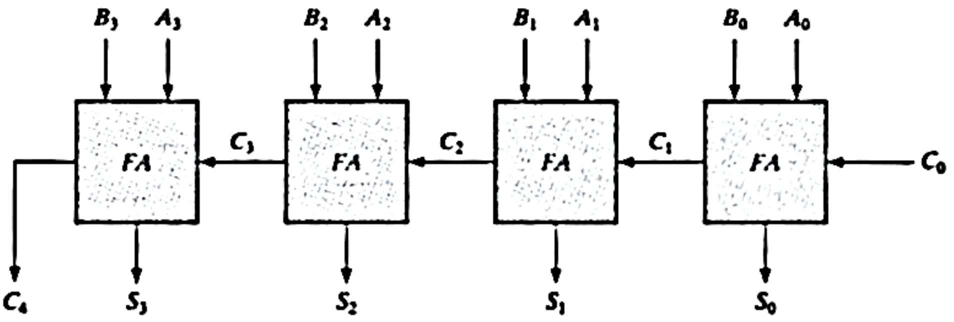
$D = A \oplus B \oplus B_{in}$

$B_{out} = A'B + A'B_{in} + BB_{in}$



Binary adder

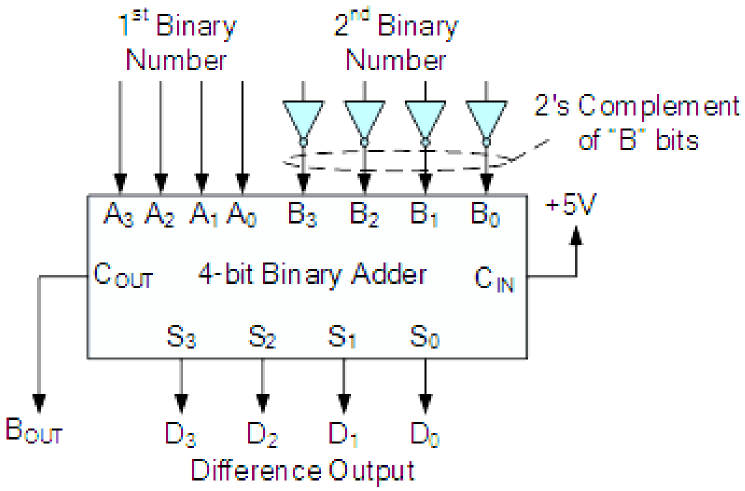
A binary adder is a digital circuit that produces the arithmetic sum of two binary numbers. It can be constructed with full adders connected in cascade with the output carry from each full adder connected to the input carry of the next full adder in the chain. The block diagram of a four-bit adder given below is a typical example of a standard component. It can be used in many applications involving arithmetic operations.



Two n-bit binary numbers are available to the adder with all digits being presented in parallel. The addition is performed by using a full adder to add each corresponding pair of digits, one from each number. The full adders are connected in tandem so that the carry out from one stage becomes the carry into the next stage as shown above. Thus, the carry ripples through each stage. For binary addition, the carry into the first (least significant) stage is 0. The last carry out (the overflow carry) becomes the most significant bit of the (n + 1)-bit sum.

Binary Subtractor:

The subtraction (A – B) can be done by taking the 2's complement of B and adding it to A. The 2's complement can be obtained by taking the 1's complement and adding 1 to the least significant pair of bits. The 1's complement can be implemented with inverters, and 1can be added to the sum through the input carry.
The circuit for subtracting A – B consists of an adder with inverters placed between eachdata input B and the corresponding input of the full adder. The input carry C_{in} must be equal to 1(+5V) when subtraction is performed. The operation thus performed becomes A. plus the 1's complement of B plus 1. This is equal to A plus the 2's complement of B. For unsigned numbers, that gives A - B if A ≥B or the 2's complement of (B - A) if A < B. For signed numbers, the result is A - B, provided that there is no overflow.



Lab exercises

1. Design a full adder using NAND gates only [Refer truth table given in introduction]

   Hardware Requirements:

2. Design a full subtractor using NOR gates only [Refer truth table given in introduction]

   Hardware Requirements:

3. Design a full adder using two half adders and one gate

   Hardware Requirements:

4. Design a 4 bit binary adder/subtractor using 7483 IC and external gates

[Hint: The addition and subtraction operations can be combined into one circuit with one common binary adder by including an Exclusive-OR gate with each full adder. The mode input M controls the operation. When M = 0, the circuit is an adder and when M = 1, the circuit becomes a subtractor].

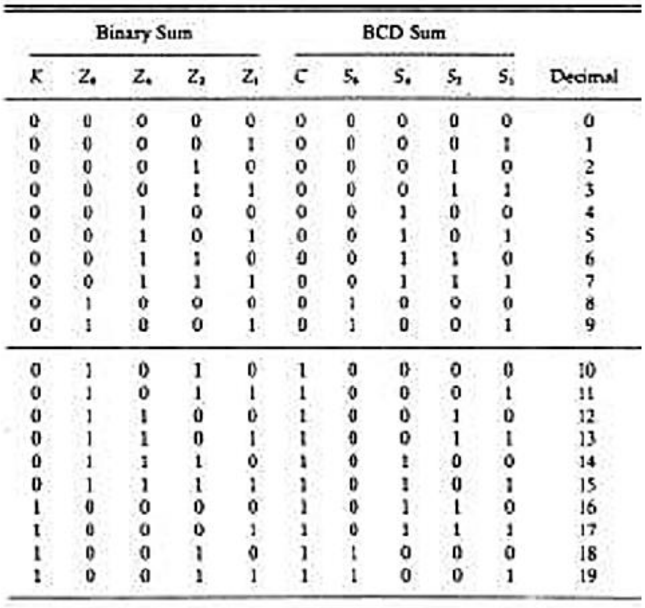
Hardware Requirements:

For the following input combinations, write the output obtained:

i. A = 10, B = 4, A + B = ?
ii. A = 12, B = 3, A – B = ?
iii. A = 7, B = 13, A – B = ?
   Note that when B>A, the difference will be in two's complement form.

5. Design a BCD adder using 7483 ICs and external gates.
   [Hint: Note that when the binary value of sum exceeds 9, the BCD sum is excess by 6]

   Truth table:



Binary Sum					BCD Sum					
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	Decimal
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

It is clear that a correction is needed when binary sum has an output carry K = 1. The other six combinations from 1010 to 1111 which need a correction have a 1 in position Z. When C = 1, we require to add 0110 to binary sum and provide an output-carry for next stage.

Hardware Requirements:

Additional exercises

Design a full subtractor using half subtractors and external gates.

[SPACE FOR ADDITIONAL EXERCISES]

LAB NO: 6 **Date:**

DESIGN OF MUTIPLIERS AND MAGNITUDE COMPARATORS

Objectives:

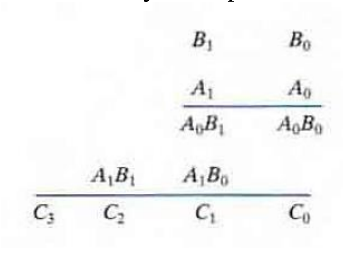
In this lab, student will be able to:

1. Elucidate various applications of 7483 IC
2. Implement a magnitude comparator with cascading inputs
3. Design a magnitude comparator using 7485 IC.

Introduction

Multipliers

Multiplication of binary numbers is performed in the same way as multiplication of decimal numbers. Consider the multiplication of two 2·bit numbers as shown below. The multiplicand bits are B_1 and B_0, the multiplier bits are A_1 and A_0, and the product is $C_3C_2C_1C_0$. The first partial product is formed by multiplying B_1 B_0 by A_0. The multiplication of two bits such as A_0 and B_0 produces a 1 if both bits are 1: otherwise, it produces a 0. This is identical to an AND operation. The second partial product is formed by multiplying B_1B_0 by A_1 and shifting one position to the left. The two partial product s are added with two half-adder (HA) circuits. A combinational circuit binary multiplier with more bits can be constructed in a similar fashion.



Magnitude Comparators

The comparison of two numbers is an operation that determines whether one number is greater than, less than or equal to the other number. A magnitude comparator is a combinational circuit that compares two numbers A and B and determines their relative magnitudes. The outcome of comparison is specified by three binary variables must indicate whether A > B, A = B or A < B. Consider two numbers A and B with four digits each. The coefficients of these numbers are given as: $A = A_3A_2A_1A_0$

$B = B_3B_2B_1B_0$

Each subscripted letter represents one of the digits in the number. The two numbers are equal if all pairs of significant digits are equal: $A_3 =B_3$, $A_2 = B_2$, $A_l =B_1$ and $A_0 = B_0$. When the numbers are binary, the digits are either 1 or 0 and the equality of each pair of bits can be expressed logically with an exclusive-NOR function as $X_i = A_i'B_i'; + A_iB_i$; for i = 0,1, 2, 3 where $X_i = 1$ only if the pair of bits in position i are equal. For equality (A=B) to exist, all Xi variables must be equal to 1. A condition that dictates an AND operation of all variables:

$(A =B) = X_3X_2X_1X_0$

To determine whether A is greater or less than B, we inspect the relative magnitudes of pairs of significant digits starting from the most significant position. If the two digits of a pair are equal, we compare the next lower significant pair of digits. The comparison continues until a pair of unequal digits is reached. If the corresponding digit of A is 1, and that of B is 0, we conclude that A > B. If the corresponding digit of A is 0 and that of B is 1, we have A < B. The sequential comparison can be expressed logically by the two Boolean functions

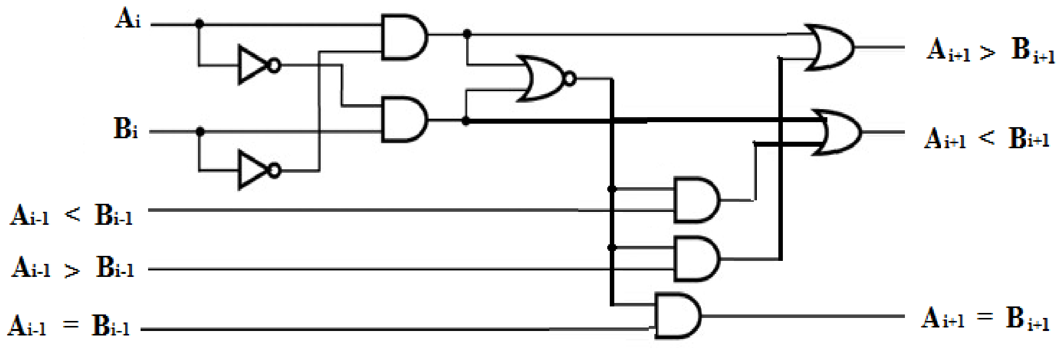
$$(A > B) = A_3B_3' + X_3A_2B_2' + X_3X_2A_1B_1' + X_3X_2X_1A_0B_0'$$

$$(A < B) = A3'B3 + X_3A_2'B_2 + X_3X_2A_1'B_1 + X_3X_2X_1A_0'B_0$$

Solved Exercise

Design a 1 bit magnitude comparator with cascading inputs using basic gates.
Let A and B be the two inputs of one bit comparator. There are three possible output values.
A = B (00 or 11), A < B (01) or A > B (10)

The logic diagram is shown in the figure



Build the circuit and verify the result obtained with truth table given below.

A	B	A>B	A=B	A<B
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

Lab exercises

1. Design a magnitude comparator (4-bit) using 7483 IC and external gates.
   Hardware Requirements:

2. Design 5 bit magnitude comparator using 7485 ICs.
   [Use cascading inputs of 7485 IC to obtain 8 bit comparator]

   Hardware Requirements:

3. Design 2 bit X 3 bit binary multiplier using 7483 ICs and external gates.

   Hardware Requirements:

Additional exercise

Design a 4 bit X 3 bit binary multiplier using 7483 ICs and external gates.

[SPACE FOR ADDITIONAL EXERCISES]

LAB NO: 7 **Date:**

DESIGN OF DECODERS AND IMPLEMENTATION OF COMBINATIONAL LOGIC CIRCUITS USING DECODERS

Objectives:

In this lab, student will be able to:

1. Realize the applications of decoder
2. Construct higher order decoders using lower order decoders
3. Comprehend 74138 IC and its applications.

Introduction

Decoders

A decoder is a combinational circuit that converts binary information from n input lines to a maximum of 2" unique output lines. Furthermore, decoders include one or more enable inputs (E) to control the circuit operation. The decoder is enabled when E is equal to 0 (i.e., active-low enable).The circuit is disabled when E is equal to 1, regardless of the values of the other 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 un-complemented outputs.

Decoders find various applications which include binary to octal conversion. A higher order decoder can be constructed using blocks of lower order decoders with enable pins. Decoders can be used to implement any Boolean expression given in SOP or POS form.

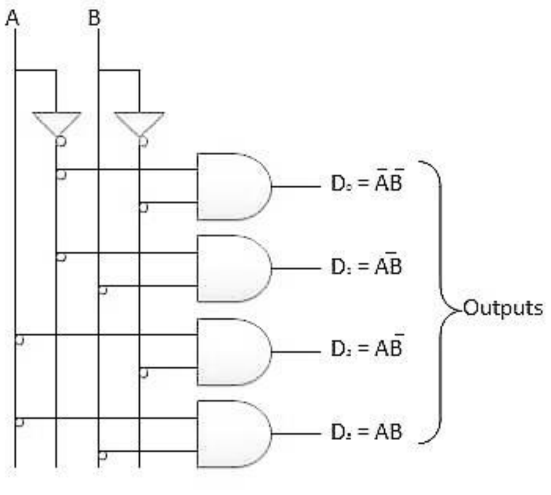
Solved exercise 1

The truth table given below explains the working of a simple 2 to 4 decoder without enable pin, considering active high output lines.

A	**B**	**D_3**	**D_2**	**D_1**	**D_0**
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Observe that, at any instant of time, only one of the output lines will be high.

Circuit Diagram:



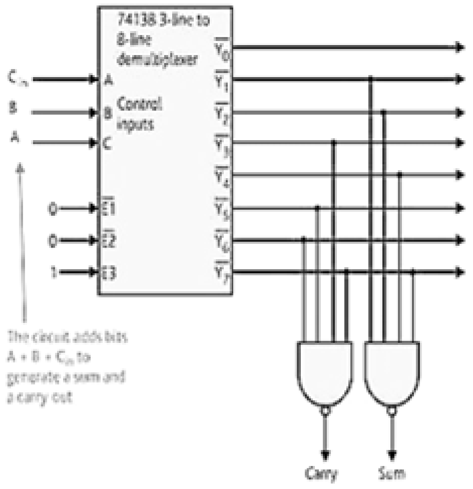
Rig the circuit as shown in the diagram using basic gates and verify the result obtained using truth table.

Combinational Logic Circuit Design using decoders

Solved exercise 2

Implement one bit full adder using 74138 IC and external NAND gates only.
The truth table of 1 bit full adder is given below. The circuit diagram built using the 74138 and NAND gates is shown below.

Input bit for number A	Input bit for number B	Carry bit input C_{IN}	Sum bit output S	Carry bit output 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



From the diagram observe that 74138 provide active low outputs.
From the truth table,

$$S = \sum(1,2,4,7) \text{ and}$$
$$C_{out} = \sum(3,5,6,7)$$

Here S and C are considered to be active high logic output lines. So, to implement S and C using active high output decoder, we need to consider the sum of minterms, where each AND terms are OR-ed to obtain the output. To implement S and C using active low output decoder, we need to invert the output lines obtained at every AND terms. This is then OR-ed. But, we know that invert OR operation is equivalent to NAND operation. Therefore, the active low output lines of the corresponding minterms are NAND-ed to obtain S and C output lines. Similarly, to implement a function F given by product of maxterms, each OR-ed term is AND-ed in active high output decoder and each OR-ed term is NOR-ed in active low output decoder.

Build the circuit as shown above and verify the result obtained using truth table.

Lab exercises

1. Design a combinational circuit using 74138 IC and external gates to implement the following functions.
   F1 = x'y' + xy'z
   F2 = x'+y

   Hardware Requirements:

   For the various input values, X, Y and Z, F1 and F2 obtained is given below.

X	Y	Z	F1	F2
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

2. Design a 4-16 decoder 3-8 decoders [74138 ICs] and external gates
   [Hint: The MSB of 4 bit input is connected to enable line input of 74138 IC to obtain higher order decoder]

Hardware Requirements:

Additional exercises

Design a full adder/subtractor using 74138 and external gates

---

[SPACE FOR ADDITIONAL EXERCISES]

LAB NO: 8 **Date:**

DESIGN OF MULTIPLEXERS AND IMPLEMENTATION OF COMBINATIONAL LOGIC CIRCUITS USING MULTIPLEXERS

Objectives:

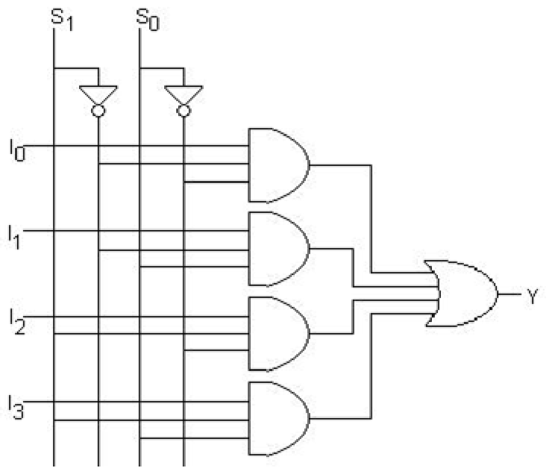
In this lab, student will be able to:

1. Realize various combinational logic circuits using Multiplexers.
2. Build higher order MUX using lower order MUX
3. Understand the applications of 74151, 74153 and 74157 ICs

Introduction

Multiplexers

A multiplexer (MUX) 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. The multiplexers act like an electronic switch that selects one out of many sources. The block diagram and truth table of a 4:1 multiplexer is sometimes depicted by a wedge-shaped symbol as shown below. The expression for output Y is given as $Y = S_1'S_0'I_0 + S_1'S_0I_1 + S_1S_0'I_2 + S_1S_0I_3$. I_0, I_1, I_2 and I_3 may be 0 or 1 at any instant of time. Observe that at any instant of time, any one of the input lines is selected at the output of multiplexer.



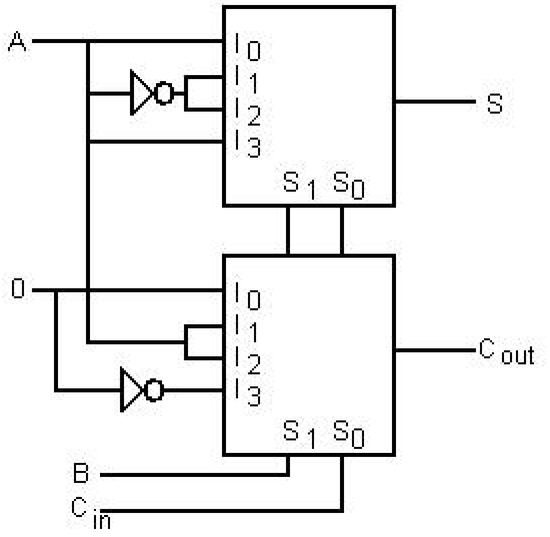
Select Lines		Output (Y)
S_1	S_0	
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Solved exercise
Combinational Logic circuit design using multiplexers

Implement a full adder using 4 X 1 MUX and external gates.

Truth Table:

Input bit for number A	Input bit for number B	Carry bit input C_{IN}	Sum bit output S	Carry bit output 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



Rig the circuit and verify the output obtained using truth table.

Lab exercises

1. Design a combinational circuit using 74151 IC and external gates to implement the following functions
   $F_1(w,x,y,z) = wx'y'+w'xy'z$
   $F_2(w,x,y,z) = xz'+wx'y$

   Hardware Requirements:

2. Design 8 X 1 MUX using 4 X 1 MUX's only
   [Hint: 74153 IC is a dual 4 X 1 MUX. Use three 4 X 1 MUX for implementation.]

   Hardware Requirements:

3. Design a full adder/subtractor using 74153 ICs and external gates . Also write the reduction table.

   Truth Table:

A	B	C_{in}	SUM	CARRY	DIFFERENCE	BORROW
0	0	0				
0	0	1				
0	1	0				
0	1	1				
1	0	0				
1	0	1				
1	1	0				
1	1	1				

Hardware Requirements:

Additional exercises

1. Design a 3 bit magnitude comparator using 74153 ICs and external gates
2. Design a combinational circuit that works as a 4 bit adder when y = 0 and works as a 4 bit subtractor when y = 1, using 74157 ICs, 7483 IC.

[SPACE FOR ADDITIONAL EXERCISE]

LAB NO: 9 **Date:**

CONVERSION OF FLIP FLOPS AND DESIGN OF ASYNCHRONOUS COUNTERS USING FLIP FLOPS

Objectives:

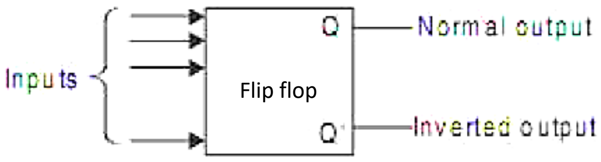
In this lab, student will be able to:

1. Convert one flip flop into another flip flop
2. Construct different circuits of asynchronous counter using flip flops
3. Understand real time applications of 7490 and 7493 ICs

Introduction

Flip flops

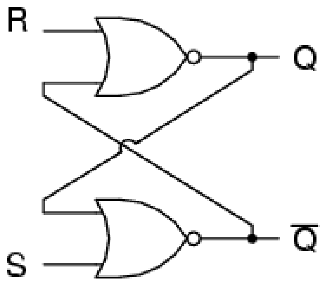
The basic 1-bit digital memory circuit is known as a flip-flop. It can have only two states, either the 1 state or the 0 state. A flip flop is also known as bi-stable multi-vibrator. Flip-flops can be obtained by using NAND or NOR gates. The general block diagram representation of a flip-flop is shown in Figure.



It has one or more inputs and two outputs. The two outputs are complementary to each other. If Q is 1 i.e., Set, then Q' is 0; if Q is 0 i.e., Reset, then Q' is 1. That means Q and Q' cannot be at the same state simultaneously. There are different types of flip-flops depending on how their inputs and clock pulses cause transition between two states i.e. S-R, D, JK and T.

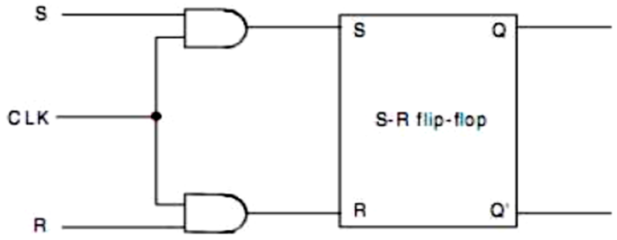
S R flip flop

A S-R flip flop is constructed with NOR gates at ease by connecting the NOR gates back to back as shown in Figure. The cross-coupled connections from the output of gate 1 to the input of gate 2 constitute a feedback path. This circuit is not clocked and is classified as an asynchronous sequential circuit. The truth table for the S-R flip-flop based on a NOR gate is shown in the table.

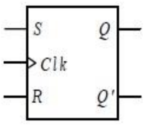
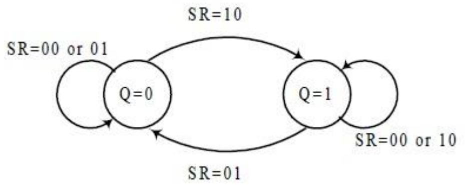


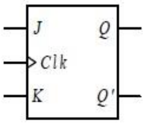
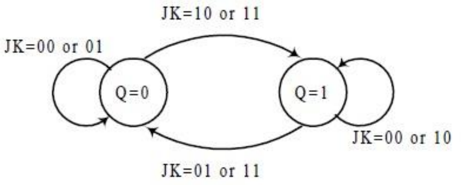
Inputs		Output	Action
S	R	Q_{n+1}	
0	0	Q_n	No Change
0	1	0	Reset
1	0	1	Set
1	1	X	Undefined

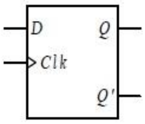
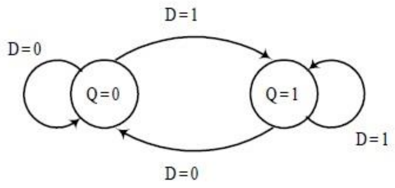
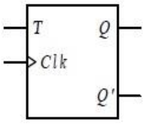
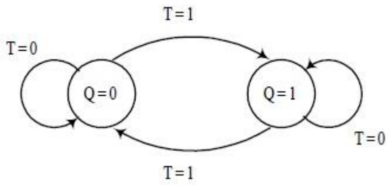
Generally, synchronous circuits change their states only when clock pulses are present. The operation of the basic flip flop can be modified by including an additional input to control the behavior of the circuit. The clock input is connected to both of the AND gates, resulting in LOW outputs when the clock input is LOW. In this situation the changes in S and R inputs will not affect the state (Q) of the flip flop. On the other hand, if the clock input is HIGH, the changes in S and R will be passed over by the AND gates and they will cause changes in the output (Q) of the flip flop. This way, any information, either 1 or 0, can be stored in the flip flop by applying a HIGH clock input and be retained for any desired period of time by applying a LOW at the clock input. This type of flip flop is called a clocked S-R flip flop.



The expression for output $Q_{n+1} = S + R'Q_n$
The characteristic table for clocked JK, D and T flip flop and their corresponding output equation is shown below. All flip flops shown in the table are positive edge triggered. In such flip flops, the output is affected only when the clock edge goes positive.

Name / Symbol	Characteristic (Truth) Table	State Diagram / Characteristic Equations	Excitation Table
SR 	*S R Q Qnext* 0 0 0 0 0 0 1 1 0 1 0 0 0 1 1 0 1 0 0 1 1 0 1 1 1 1 0 × 1 1 1 ×	 $Q_{next} = S + R'Q$ $SR = 0$	*Q Qnext S R* 0 0 0 × 0 1 1 0 1 0 0 1 1 1 × 0

JK 	*J K Q Qnext*
	0 0 0 0
	0 0 1 1
	0 1 0 0
	0 1 1 0
	1 0 0 1
	1 0 1 1
	1 1 0 1
	1 1 1 0
		 JK=10 or 11
		JK=00 or 01
		Q=0
		Q=1
		JK=00 or 10
		JK=01 or 11
		$Q_{next} = J'K'Q + JK' + JKQ'$
		$= J'K'Q + JK'Q + JK'Q' + JKQ'$
		$= K'Q(J'+J) + JQ'(K'+K)$
		$= K'Q + JQ'$
			Q Qnext J K
			0 0 0 ×
			0 1 1 ×
			1 0 × 1
			1 1 × 0

D 	*D Q Qnext* 0 × 0 1 × 1	 $Q_{next} = D$	*Q Qnext D* 0 0 0 0 1 1 1 0 0 1 1 1
T 	*T Q Qnext* 0 0 0 0 1 1 1 0 1 1 1 0	 $Q_{next} = TQ' + T'Q = T \oplus Q$	*Q Qnext T* 0 0 0 0 1 1 1 0 1 1 1 0

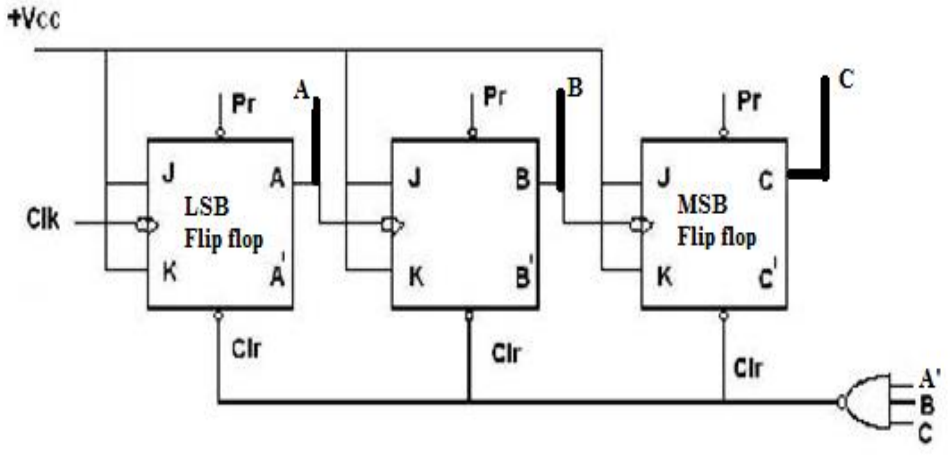
Asynchronous Counters

A sequential circuit whose behavior can be defined from the knowledge of its signal at discrete instants of time is referred to as a synchronous sequential circuit. Here there is a common clock connected to every unit. A sequential circuit whose behavior depends upon the sequence in which the input signals change is referred to as an asynchronous sequential circuit. Counters and shift registers are the major applications of flip flops. Asynchronous counters are also called as ripple counters. To design a Mod N asynchronous counter that counts from 0 to N-1 continuously, where

N <= 2^n, n flip flops are required. External clock input is given as input for the LSB flip flop. Asynchronous counters can be either positive edge triggered or negative edge triggered. In a negative edge triggered counter, if the output Q of LSB flip flop is connected as clock input for adjacent flip flop, the overall system behaves as an asynchronous UP counter. Similarly if Q’ of a flip flop is connected as clock reference for adjacent flip flop, it works as an asynchronous DOWN counter. In positive edge triggered counter, if the output Q of LSB flip flop is connected as clock input for adjacent flip flop, the overall system behaves as an asynchronous DOWN counter. Similarly if Q’ of a flip flop is connected as clock reference for adjacent flip flop, it works as an asynchronous UP counter. To reset the counter to a particular value, there is CLR line (active low). When this line is zero, all the flip flop values will be reset to zero, irrespective of the edge of clock. To set a particular value to the counter, PRE line is available (active low). When this line is zero, the counter can bel loaded with parallel inputs. Note that at any point, either CLR or PRE can to enable. If both the lines (CLR and PRE) are disabled, the flip flops are triggered by the clock input and corresponding output is generated that represents a count.

Solved exercise

Implement MOD 6 asynchronous UP counter using JK flip flop and draw the output waveform. MOD 6 counters have 6 states. It counts from 0 to 5. When the count reaches 6, the counter has to be reset to loop the count from 0 to 5.This is achieved using NAND gates and CLR input of flip flops. The block diagram of negative edge triggered asynchronous MOD 6 UP counter built using JK flip flop is shown below.



Lab exercises

1. Convert a given D flip flop to work as a JK flip flop.

Inputs				Output
J	**K**	**Q_n**	**Q_{n+1}**	**D**

Excitation table of D flip flop

Q_n	Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

K map for D:

	00	01	11	10
00				
01				
11				
10				

Simplified expression for D:

Hardware Requirements:

2. Design a 4 bit binary ripple (asynchronous) UP counter using JK flip flops.

Hardware Requirements:

3. Design a MOD 12 asynchronous UP counter using JK flip flops and external gates

Hardware Requirements:

4. Design a 4 bit binary ripple (asynchronous) DOWN counter using JK flip flops

Hardware Requirements:

5. Design a 4 bit asynchronous UP/DOWN counter using JK flip flops and 74157 IC

Hardware Requirements:

6. Design a counter that performs UP count from 00 to 81 using 7490 IC and external gates

   Hardware Requirements:

7. Design an asynchronous counter that counts from 00h to 24h using 7493 IC and external gates

   Hardware Requirements:

Additional exercises

1. Design a mod 12 asynchronous binary counter to divide the frequency of input clock waveform by a factor of 12 while producing a waveform with 50% duty cycle.
2. Design a MOD 10 asynchronous UP counter using 7493 IC and external gates.

[SPACE FOR ADDITONAL EXERCISE]

LAB NO: 10 **Date:**

DESIGN OF SYNCHRONOUS COUNTERS USING FLIP FLOPS

Objectives:

In this lab, student will be able to:

1. Design a synchronous counter using flip flop
2. Illustrate the applications of 74193 ICs

Introduction

A synchronous counter, in contrast to an asynchronous counter, is one whose output bits change state simultaneously, with no ripple. The only way we can build such a counter circuit from J-K flip-flops is to connect all the clock inputs together, so that each and every flip-flop receives the exact same clock pulse at the exact same time. The result of this synchronization is that all the individual output bits changing state at exactly the same time in response to the common clock signal with no ripple effect and therefore, no propagation delay.

The steps to design a synchronous counter are listed below.

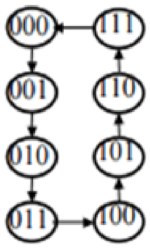
1. Determine the number of flip flops needed to support the counting sequence's highest number
2. Build a State Transition Diagram by including all states.
3. Build a State/Excitation Truth Table.
4. Simplify expressions for flip flop inputs (JK/T/D) for each flip flop on K-Maps.
5. Implement the Synchronous Counter/State Machine Circuit.
6. Draw the Timing Diagram.

Solved exercise

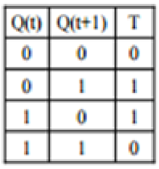
Design a 3 bit binary synchronous UP counter using T flip flops and external gates

The state diagram, state transition table and simplified expression for flip flop input are shown below. [Refer excitation table of T flip flop]

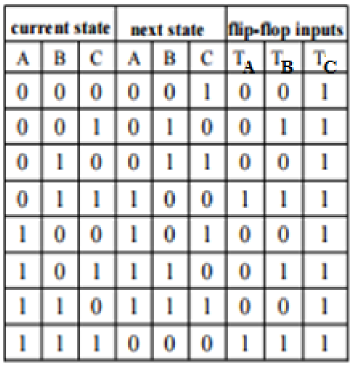
Rig up the circuit and verify the output against the state table.



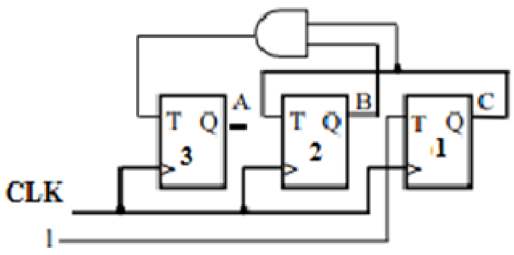
$$\begin{tikzcd}
000 \arrow[d] & 111 \arrow[l] \\
001 \arrow[d] & 110 \arrow[u] \\
010 \arrow[d] & 101 \arrow[u] \\
011 \arrow[r] & 100 \arrow[u]
\end{tikzcd}$$



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



current state			next state			flip-flop inputs		
A	B	C	A	B	C	T_A	T_B	T_C
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



Expression for flip flop inputs obtained from state transition table:

$$T_A = BC$$
$$T_B = C$$
$$T_C = 1$$

Note: Use 7473ICs to obtain T flip flops

Lab exercises

1. Design a 3 bit binary synchronous counter to count the sequence 0 -> 4 -> 2 ->7 -> 6 -> 0 using JK flip flop and external gates

   State Table with JK excitation

Present State			Next State			J,K inputs					
Q_C	**Q_B**	**Q_A**	**Q_C^+**	**Q_B^+**	**Q_A^+**	**J_C**	**K_C**	**J_B**	**K_B**	**J_A**	**K_A**

Hardware Requirements:

2. Design a 2 bit binary UP/DOWN counter such that when control input Y=0, it performs BINARY UP count and when Y=1, it performs BINARY DOWN count using D flip flops and external gates.

State Table with D excitation

Control input	Present State		Next State		D inputs	
Y	**Q_B**	**Q_A**	**Q_B^+**	**Q_A^+**	**D_B**	**D_A**
0						
0						
0						
0						
1						
1						
1						
1						

Hardware Requirements:

3. Design a 2 digit hexadecimal UP counter using 74193 IC

Hardware Requirements:

4. Design a 2 digit hexadecimal UP counter that counts between 28h to 82h using 74193 IC and external gates.

Hardware Requirements:

5. Design a 2 digit hexadecimal DOWN counter using 74193 IC

   Hardware Requirements:

6. Design a 2 digit hexadecimal DOWN counter that counts between 82h to 28h using 74193 IC and external gates.

   Hardware Requirements:

Additional exercises

1. Design a 2 digit synchronous BCD DOWN counter using 74193 IC and external gates
2. Design a 3 bit pre-settable synchronous counter (UP DOWN) using7474ICs and external gates to count from N1 to N2.

[SPACE FOR ADDITIONAL EXERCISES]

LAB NO: 11 **Date:**

IMPLEMENTATION OF SHIFT REGISTERS AND SEQUENCE GENERATORS

Objectives:

In this lab, student will be able to:

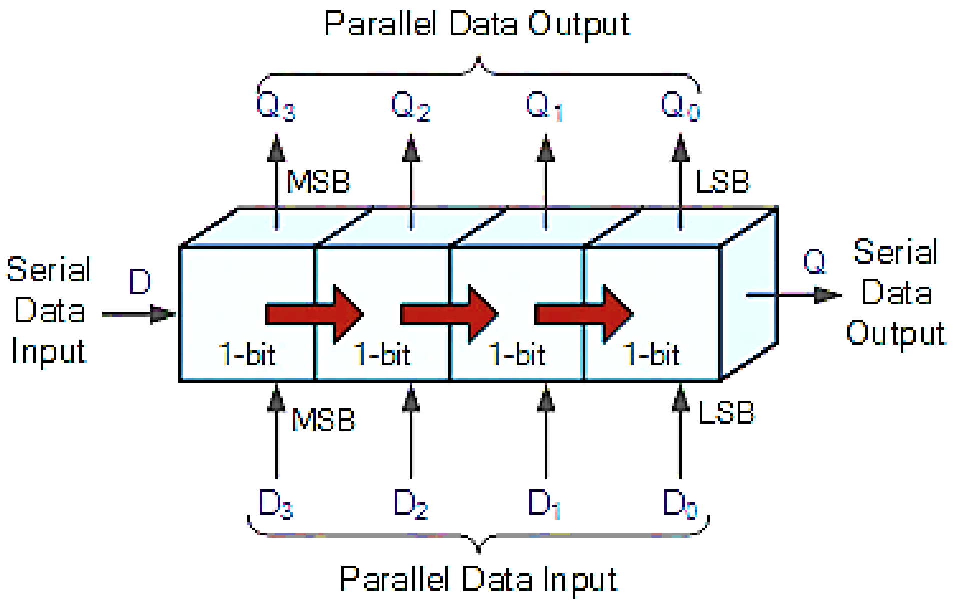
1. Realize the function of universal shift register
2. Differentiate between Johnson and Ring counter
3. Design a sequential logic circuit to generate a binary sequence

Introduction

Shift registers are a type of sequential logic circuit, mainly for storage of digital data. They are a group of flip-flops one for each data bit, either a logic "0" or a "1", connected in a chain so that the output from one flip-flop becomes the input of the next flip-flop. Shift Registers are used for data storage or for the movement of data and are therefore commonly used inside calculators or computers to store data such as two binary numbers before they are added together, or to convert the data from either a serial to parallel or parallel to serial format. Shift register IC's are generally provided with a clear or reset connection so that they can be "SET" or "RESET" as required. Generally, shift registers operate in one of four different modes with the basic movement of data through a shift register being:

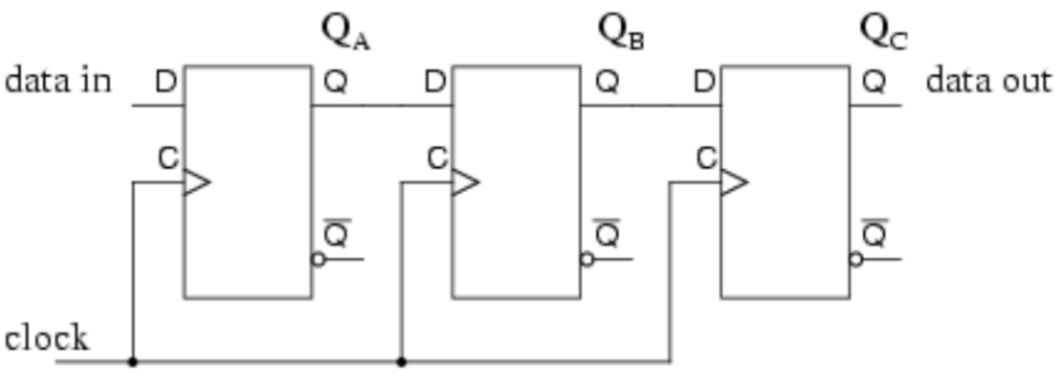
- Serial-in to Parallel-out (SIPO) - the register is loaded with serial data, one bit at a time, with the stored data being available at the output in parallel form.
- Serial-in to Serial-out (SISO) - the data is shifted serially "IN" and "OUT" of the register, one bit at a time in either a left or right direction under clock control.
- Parallel-in to Serial-out (PISO) - the parallel data is loaded into the register simultaneously and is shifted out of the register serially one bit at a time under clock control.
- Parallel-in to Parallel-out (PIPO) - the parallel data is loaded simultaneously into the register, and transferred together to their respective outputs by the same clock pulse.

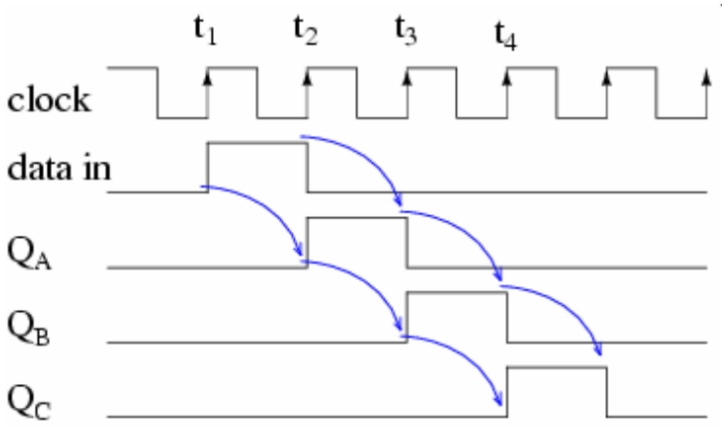
A universal shift register can perform all four types of shifting mentioned above.
The effect of data movement from left to right through a shift register can be presented graphically as:



Also, the directional movement of the data through a shift register can be either to the left, (left shifting) to the right, (right shifting) left-in but right-out, (rotation) or both left and right shifting within the same register thereby making it bidirectional. In this tutorial it is assumed that all the data shifts to the right, (right shifting).

Solved exercise
Design a 3 bit Serial-in Serial out shift register using D flip flops.
The three bit shift register shifts one bit to the right for every clock cycle. Logic diagram and output waveform is shown below.





Rig up the circuit and give data in as 1010. Observe the output waveform.

Lab exercises

1. Design a 4 bit universal shift register using D flip flops and 74153 ICs
Function Table of Universal Shift Register:

Select Lines		Function performed
S_1	S_0	F
0	0	No change
0	1	Left Shift
1	0	Right Shift
1	1	Parallel Load

Hardware Requirements:

2. Design a 4 bit ring counter using JK flip flops

Hardware Requirements:

Q_D	Q_C	Q_B	Q_A

3. Design a 4 bit Johnson (twisted ring) counter using JK flip flops. Decode the output of counter using 2 input AND gates only.

Hardware Requirements:

Output Table:

Q_D	Q_C	Q_B	Q_A

4. Design a sequential logic circuit to generate a binary sequence 110010 using Johnson counter

Counter State			Output Sequence
Q_C	Q_B	Q_A	**Y**

Hardware Requirements:

5. Design a sequential logic circuit to generate a binary sequence 0111101 using synchronous counter (designed using JK flip flop) and external gates

State Table with SR excitation

Present State			Next State			Output	J, K inputs					
Q_C	Q_B	Q_A	Q_C^+	Q_B^+	Q_A^+	Y	J_C	K_C	J_B	K_B	J_A	K_A

Hardware Requirements:

Additional exercises

1. Design a sequential logic circuit to generate a binary sequence 101110 using synchronous counter IC

[SPACE FOR ADDITIONAL EXERCISE]

LAB NO: 12 **Date:**

DESIGN AND IMPLEMENTATION OF BINARY SEQUENCE DETECTOR

Objectives:

In this lab, student will be able to:

- Implement sequential model to detect a binary sequence.

Introduction

A sequence detector accepts as input a string of bits: either 0 or 1.Its output goes to 1 when a target sequence has been detected. There are two basic types: overlap and non-overlap. In a sequence detector that allows overlap, the final bits of one sequence can be the start of another sequence.

```
11011 detector with overlap        X 11011011011
                                   Z 00001001001
11011 detector with no overlap Z 00001000001
```

Steps to design a sequence detector are as follows:

1. Identify the number of states required to detect the entire sequence. Each state can determine one bit of the sequence.
2. Write a state transition machine for the required sequence considering all possible combinations of input and its corresponding outputs with overlap or without overlap.
3. Generate a state/excitation table for the given state machine using required flip flops.
4. Simplify expressions for flip flop(JK/T/D) inputs for each flip flop on K-Maps.
5. Implement the Synchronous Counter/State Machine Circuit.
6. Draw the Timing Diagram.

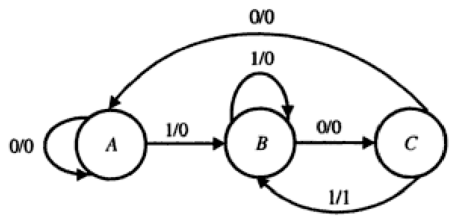
Solved exercise

Design a sequential model that detects the sequence 101 allowing overlapping of sequence

The sample input sequence and expected output is given below.

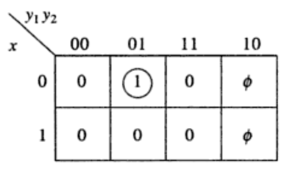
```
Input sequence:   1 0 1 0 1 1 0 1 0 0 1 1 0 1 0 1
                      ↓   ↓     ↓           ↓   ↓
Output sequence:      1   1     1           1   1
```

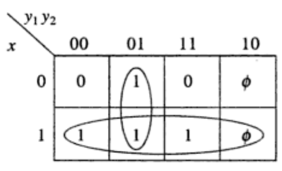
Since there are 3 bits to be detected, 3 flip flops are necessary.
State diagram and state table is given below. Here A, B, C represents any count value.

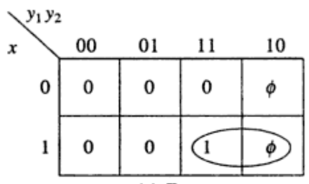


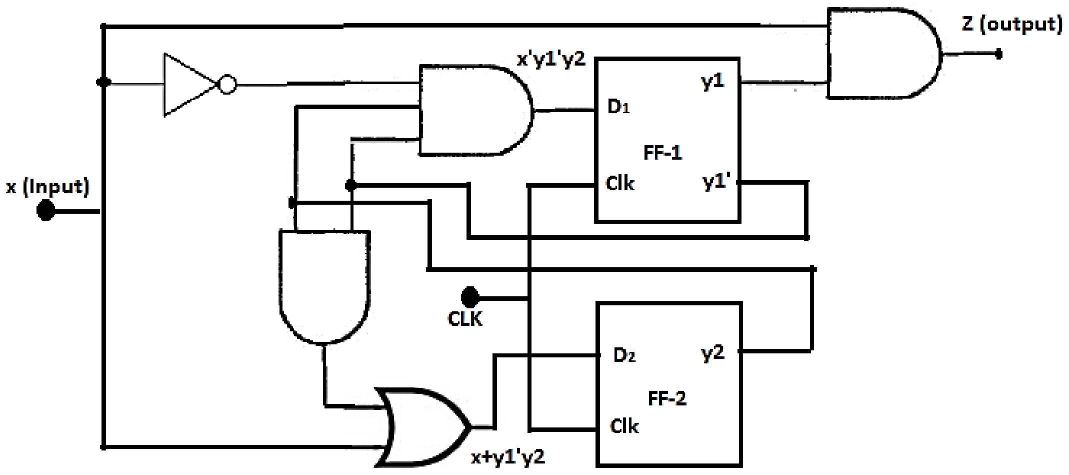
	Binary code	Next state, Y/Output, Z			
Present state	y_1y_2	$x = 0$	Y_1Y_2/Z	$x = 1$	Y_1Y_2/Z
A	00	*A*/0	00/0	*B*/0	01/0
B	01	*C*/0	11/0	*B*/0	01/0
C	11	*A*/0	00/0	*B*/1	01/1

Using D flip flop, we get the excitation table to be equivalent to next state output values. Thus y1 (MSB) and y2 (LSB) represent D flip flop outputs. X is the external input sequence. Z represents the sequence detector output. To draw the circuit diagram, obtain the simplified expression for flip flop inputs using K Map.


(a) $Y_1 = \bar{x}\bar{y}_1y_2$


(b) $Y_2 = x + \bar{y}_1y_2$


(c) $Z = xy_1$



Lab exercises

1. Design a sequential model that detects the sequence 1110 using T flip flops and external gates. Overlapping of the sequence is allowed.

   State Table with T flip flop excitation

Present State			Input	Next State			D flip flop input			Output
				X = 0			X = 0			
Q_C	**Q_B**	**Q_A**	**X**	**Q_C^+**	**Q_B^+**	**Q_A^+**	**T_C**	**T_B**	**T_A**	**Y**
0	0	0	0							
0	0	0	1							
0	0	1	0							
0	0	1	1							
0	1	0	0							
0	1	0	1							
0	1	1	0							
0	1	1	1							
1	0	0	0							
1	0	0	1							
1	0	1	0							
1	0	1	1							
1	1	0	0							
1	1	0	1							
1	1	1	0							
1	1	1	1							

   Hardware Requirements:

2. Design a sequential model that detects the sequence 10001. Assume non overlapping sequence is to be detected

State Table with D flip flop excitation

Present State			Input	Next State			D flip flop input			Output
				X = 0			X = 0			
Q_C	**Q_B**	**Q_A**	**X**	**Q_C^+**	**Q_B^+**	**Q_A^+**	**D_C**	**D_B**	**D_A**	**Y**
0	0	0	0							
0	0	0	1							
0	0	1	0							
0	0	1	1							
0	1	0	0							
0	1	0	1							
0	1	1	0							
0	1	1	1							
1	0	0	0							
1	0	0	1							
1	0	1	0							
1	0	1	1							
1	1	0	0							
1	1	0	1							
1	1	1	0							
1	1	1	1							

Hardware Requirements:

REFERENCES

1. M. Morris Mano, “Digital Design”, Prentice Hall India, 2013.

2. Ronald J. Tocci and Neal S. Widmer, “Digital Systems”, 9th Edition, Pearson Education, 2007.

3. J.F.Wakerly, “Digital Design Principles and Practices”, 3rd Edition, Pearson Education, 2003

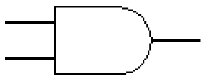
APPENDIX

BASIC GATES

In electronics, a logic gate is an idealized or physical device implementing a Boolean function; that is, it performs a logical operation on one or more logical inputs, and produces a single logical output. Logic gates are primarily implemented using diodes or transistors acting as electronic switches.

AND gate

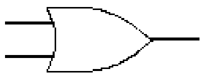
The AND gate is so named because, if 0 is called "false" and 1 is called "true," the gate acts in the same way as the logical "and" operator.



Input	Input	Output
0	0	0
0	1	0
1	0	0
1	1	1

OR GATE

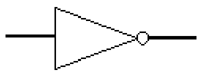
The OR gate gets its name from the fact that it behaves after the fashion of the logical inclusive "or." The output is "true" if either or both of the inputs are "true." If both inputs are "false," then the output is "false."



Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	1

NOT GATE

A logical inverter, sometimes called a NOT gate to differentiate it from other types of electronic inverter devices, has only one input. It reverses the logic state.

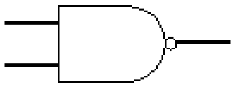


Input	Output
1	0
0	1

UNIVERSAL GATES

NAND GATE

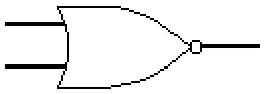
The NAND gate operates as an AND gate followed by a NOT gate. It acts in the manner of the logical operation "and" followed by negation. The output is "false" if both inputs are "true." Otherwise, the output is "true."



Input 1	Input 2	Output
0	0	1
0	1	1
1	0	1
1	1	0

NOR GATE

The NOR gate is a combination OR gate followed by an inverter. Its output is "true" if both inputs are "false." Otherwise, the output is "false."

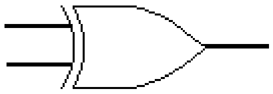


Input 1	Input 2	Output
0	0	1
0	1	0
1	0	0
1	1	0

OTHER GATES

XOR GATE

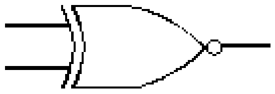
The XOR (exclusive-OR) gate acts in the same way as the logical "either/or." The output is "true" if either, but not both, of the inputs are "true." The output is "false" if both inputs are "false" or if both inputs are "true."



Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	0

XNOR GATE

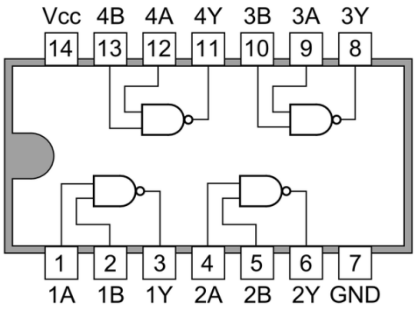
The XNOR (exclusive-NOR) gate is a combination XOR gate followed by an inverter. Its output is "true" if the inputs are the sameand “false" if the inputs are different.



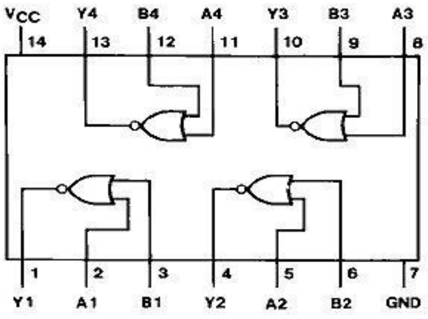
Input 1	Input 2	Output
0	0	1
0	1	0
1	0	0
1	1	1

IC PIN DIAGRAMS

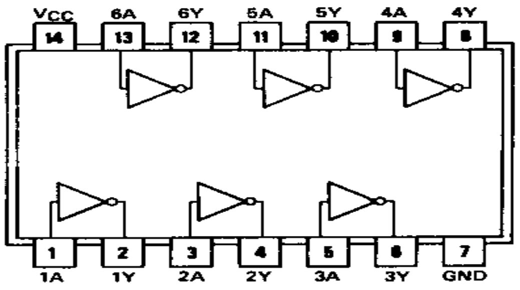
1. 7400 : QUAD 2 input NAND gates



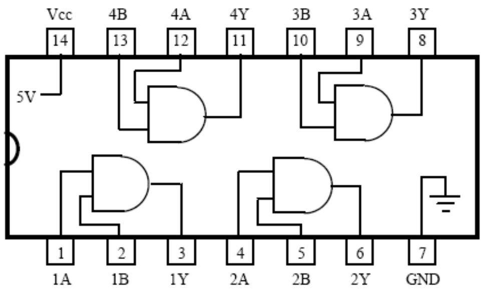
2. 7402 : QUAD 2 input NOR gates



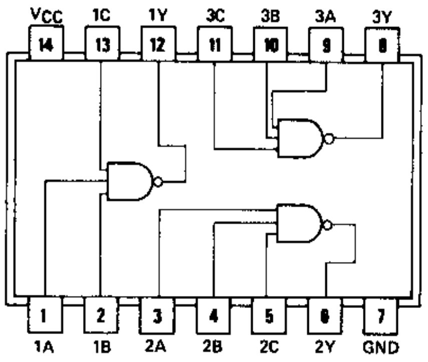
3. 7404 : HEX NOT gates



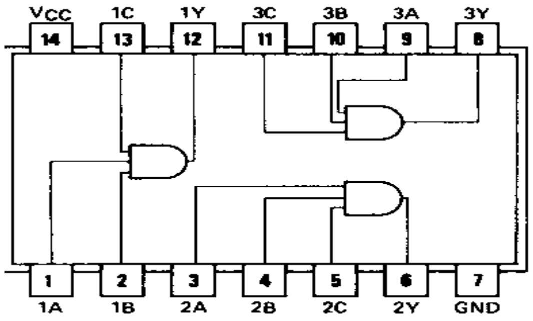
4. 7408 : QUAD 2 input AND gates



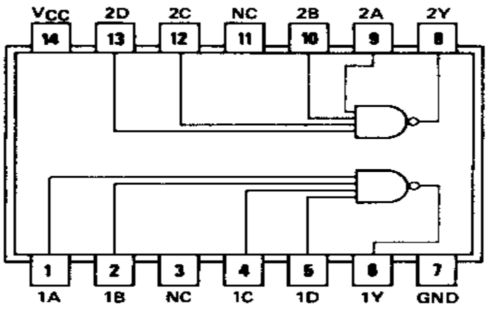
5. 7410: TRIPLE 3 input NAND gates



6. 7411 : TRIPLE 3 input AND gates

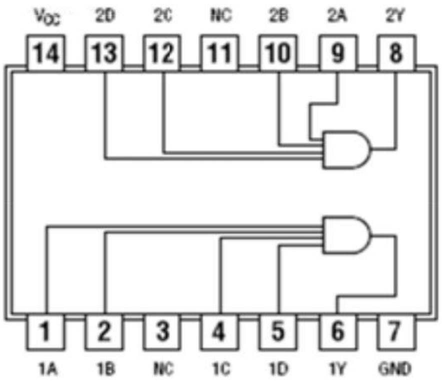


7. 7420:DUAL 4 input NAND gates

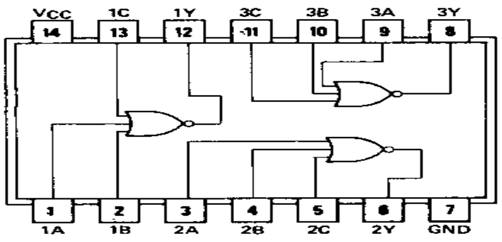


$

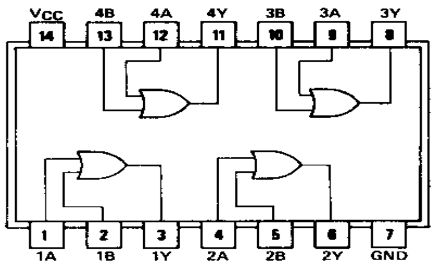
8. 7421:DUAL 4 input AND gates



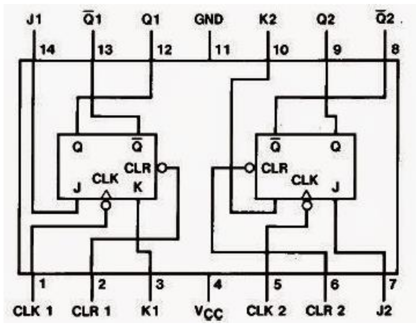
9. 7427 : TRIPLE 3 input NOR gates



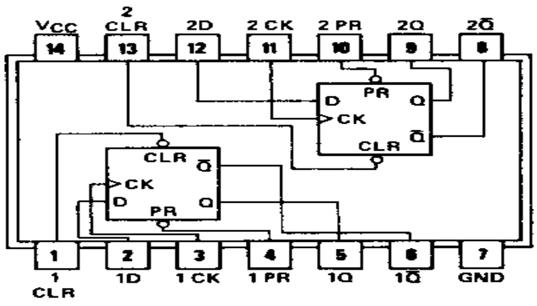
10. 7432 : QUAD 2 input OR gates



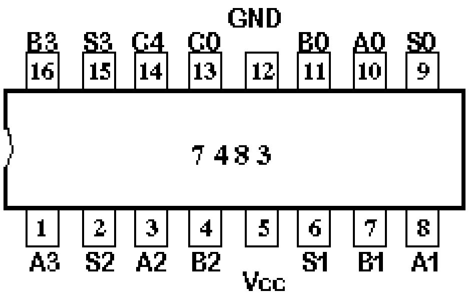
11. 7473: DUAL edge triggered JK flip flop



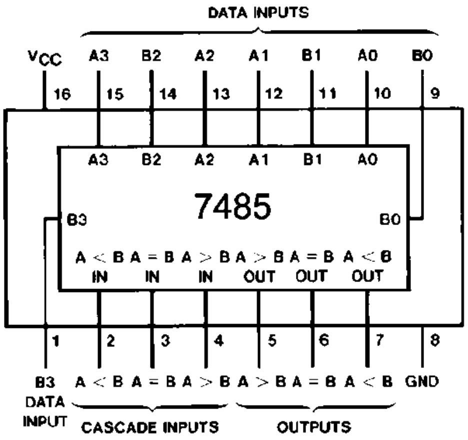
12. 7474: DUAL edge triggered D flip flops



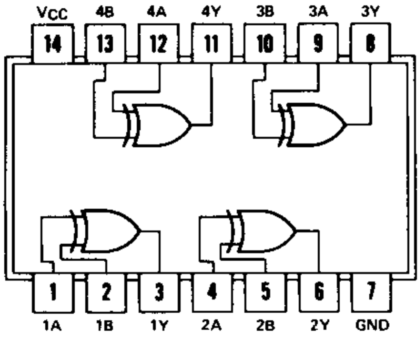
13. 7483 : 4 bit binary full adder



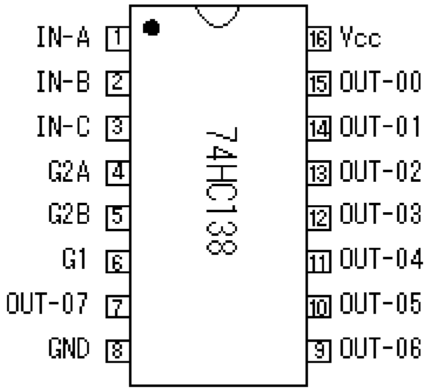
14. 7485:4 bit Magnitude comparator



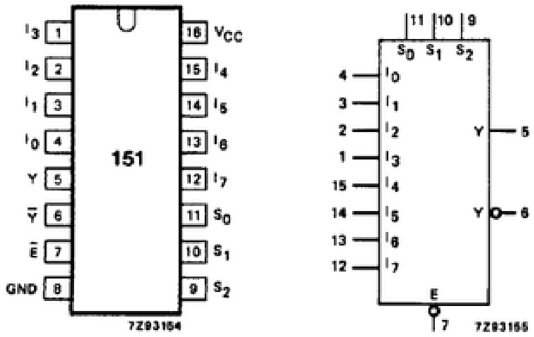
15. 7486: QUAD 2 input EX-OR gates



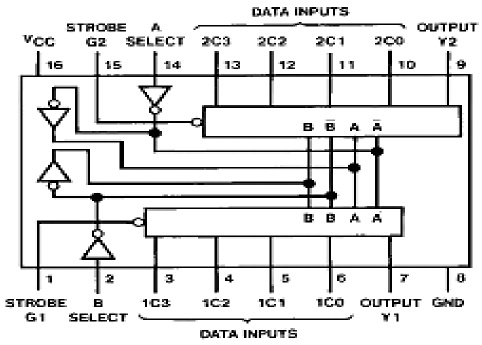
16. 74138: 3 to 8 active low output decoder



17. 74151: 8 input multiplexer

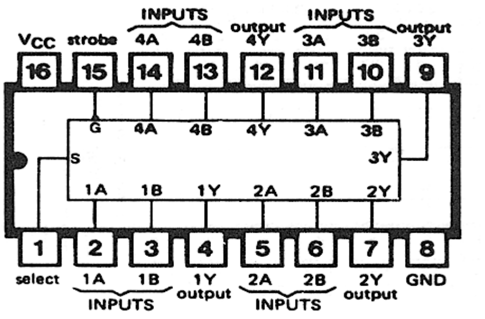


18. 74153: DUAL 4 input multiplexer



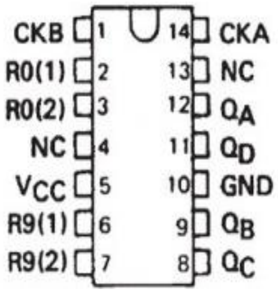
Note: Strobe G1, G2 are to be connected to GND
Pin number 14 is LSB

19. 74157: QUAD 2 input multiplexer



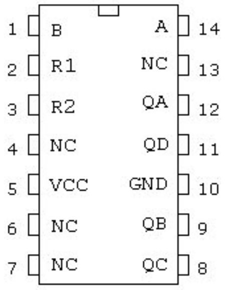
Note: Strobe pin (15) is to be connected to GND

20. 7490 : Asynchronous Decade counter



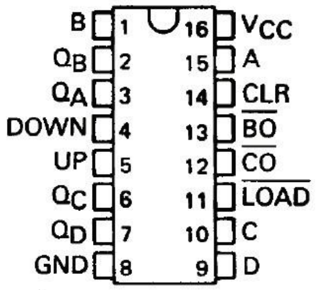
Note: Connect pin number 6 & 7 to GND

21. 7493 : Asynchronous MOD 16 UP counter



Note: Pin 6, 7 represent NO connection

22. 74193 : Synchronous UP/DOWN counter



Note: A(LSB), B, C, D(MSB) are parallel inputs
$Q_{A\ (LSB)}$, Q_B, Q_C, $Q_{D(MSB)}$ are outputs
UP, DOWN are clock inputs
CLR is active high input
B0 is active low output for borrow
C0 is active low output for carry