Course code	Digital Logic and Design	L	T	P	J	C
CSE1003		3	0	2	0	4
Pre-requisite	-	Syllabus version				
		v. xx.xx				

Course Objectives:

1. **Introduce the concept of digital and binary systems**
2. **Design and analyze combinational logic circuits.**
3. **Design and analyze sequential logic circuits.**
4. **Design functional blocks of a digital system.**
5. **Reinforce theory and techniques taught in the classroom through experiments in the laboratory.**

Expected Course Outcome:

CO1: Comprehend the different types of number systems.
CO2: Evaluate and simplify logic functions using Boolean Algebra andK-map
CO3: Design minimal combinational logic circuits.
CO4: Analyze the operation of medium complexity standard combinational circuits like the encoder, decoder, multiplexer, demultiplexer.
CO5:Familiar with basic sequential components and analyze and design the FSM
CO6: Design different types of registers and counters using flip flops
CO7:Design Arithmetic and Logic Circuits

Student Learning Outcomes (SLO): 1,2,5,14

Module:1	INTRODUCTION	3 hours	SLO: 1

Number System - Base Conversion- Binary Codes - Complements(Binary and Decimal)

Module:2	BOOLEAN ALGEBRA	8 hours	SLO: 2

Boolean algebra - Properties of Boolean algebra - Boolean functions - Canonical and Standard forms - Logic gates - Universal gates - Karnaugh map - Don't care conditions - Tabulation Method

Module:3	COMBINATIONAL CIRCUIT – I	4 hours	SLO: 2,5

Adder -Subtractor- Code Converter -Analyzing a Combinational Circuit

Module:4	COMBINATIONAL CIRCUIT –II	6 hours	SLO: 2,5

Binary Parallel Adder- Look ahead carry - Magnitude Comparator - Decoders – Encoders - Multiplexers –Demultiplexers

Module:5	SEQUENTIAL CIRCUITS – I	6 hours	SLO: 1,2,5

Flip Flops - Sequential Circuit: Design and Analysis - Finite State Machine: Moore and Mealy model - Sequence Detector

Module:6	SEQUENTIAL CIRCUITS – II	7 hours	SLO: 2,5

Registers - Shift Registers - Counters - Ripple and Synchronous Counters - Modulo counters - Ring and Johnson counters

Module:7	ARITHMETIC LOGIC UNIT	9 hours	SLO: 1,2,5

Bus Organization- ALU-Design of ALU-Status Register-Design of Shifter - Processor Unit- Design of specific Arithmetic Circuits- Accumulator-Design of Accumulator

Module:8	Contemporary issues:	2 hours	SLO: x,x
	Total Lecture hours:	45 hours	

Text Book(s)

1. I.M. Morris Mano – Digital Logic and Computer Design, Pearson Education India – 1st Edition-2016, ISBN: 9789332542525

Reference Books

1. 1.A.P. Malvino, D.P. Leach and GoutamSaha – Digital Principles and Applications(SIE) – Tata McGraw Hill 8th Edition – 2014, ISBN: 9789339203405.
   2.M. Morris Mano and Michael D.Ciletti– Digital Design: With an introduction to Verilog HDL - Pearson Education – 5th Edition- 2014. ISBN:9789332535763
   3.Thomas Floyd – Digital Fundamentals – Pearson Education-10th Edition – 2011, ISBN: 9788131734483.
2. Authors, book title, year of publication, edition number, press, place

Mode of Evaluation:

Lab (Indicative List of Experiments in the areas of)	SLO: 1, 2, 5, 14

Study of Logic Gates

- Logic gates using discreteComponents
- Verification of truthtable for logic gates
- Realization of basic gates using NAND and NOR gates

Implementation of LogicCircuits

- Verification of Boolean laws
- Verification of De Morgan's law

Adder and Subtractor

- Implementation of Half-Adder and Full-Adder
- Implementation of Half-Subtractor and Full-Subtractor

Combinational Circuit Design

- Design of Decoder and Encoder

- Design of Multiplexer and De multiplexer
- Design of Magnitude Comparator
- Design of CodeConverter

Sequential Circuit Design

- Design of Mealy and Moore circuit
- Implementation of Shift registers
- Design of 4-bit Counter
- Design of Ring Counter

List of Challenging Experiments (Indicative)

1.	Consider four seats, numbered 0 to 3, arranged in a circle and described by Boolean variables I0to I3. Boolean variable I0is true if seat 0 is occupied and I0is false if the seat is not occupied, likewise for I1, I2, and I3. Write a Boolean expression that's true if at least two people are sitting next to each other and at least one seat is not occupied and design the circuit using basic gates.	X hours
2.	Controller of a car has three control switches, Accelerator (A), Brake (B) and Clutch (C). A Car runs if A is pressed and either (B and C are not pressed) or C is pressed and B is not pressed. When the switch C alone is pressed then the car will be in running state. If all the three switches are pressed together then we can't predict the result. In all other cases, car will be in off state. Design a circuit to implement this scenario using Universal gates.	X hours
3.	A digitally controlled locker works based on a control switch and two keys which are entered by the user. Each key has a 2-bit binary representation. If the control switch is pressed, the locking system will pass the difference of two keys into the controller unit. Otherwise, the locking system will pass the sum of the two numbers to the controller unit. Design a circuit to determine the input to the controller unit	X hours
4.	The controller unit in the above problem compares the input data with the predefined data to allow access to the locker. Assuming the predefined key to be (01)2, design the circuit that outputs true if the predefined key and the input to the controller unit are similar.	X hours
5.	A bank queuing system has a capacity of 5 customers which serves on first come first served basis. A display unit is used to display the number of customers waiting in the queue. Whenever a customer leaves the queue, the count is reduced by one and the count is increased by one if a customer joins a queue. Two sensors (control signals) are used to sense customers leaving and joining the queue respectively. Design a circuit that displays the number of customers waiting in the queue in binary format using LEDs. Binary '1' is represented by LED glow and '0' otherwise.	X hours
	Total Laboratory Hours	30 hours

Mode of evaluation:

Recommended by Board of Studies	DD-MM-YYYY		
Approved by Academic Council	No. xx	Date	DD-MM-YYYY

CO-PO MAPPING:

	PO 1 SLO 1	PO 2 SLO 2	PO 3 SLO 5	PO 9 SL014	PO 10	PO 11	PO 12	PO 13	PO 15	PO 16	PO 18	PO 20
CO1	*	*										
CO2		*										
CO3		*	*									
CO4		*	*									
CO5		*	*									
CO6			*									
CO7			*	*								

2. Knowledge Areas that contain topics and learning outcomes covered in the course

Knowledge Area	Total Hours of Coverage [Theory]
CE: DIG (Digital Logic)	42
CS: AR Digital Logic and Digital Design	3
Total	45 Hours [45]

2.1 Body of Knowledge coverage

KA	Knowledge Unit	Topics Covered	Hours
CE:	History and overview	Introduction to logic circuits, switching, memory,	1

DIG 0 CS: AR	Digital Logic and Digital Systems	registers, counters and digital systems	
CE: DIG 1	Switching theory	Number System and Codes Binary Arithmetic and Complements Boolean algebra: Properties of Boolean algebra Boolean functions: Canonical and Standard forms Minimization: Karnaugh map, Don't care conditions, Tabulation Method	10
CE: DIG 2 CS:AR	Combinational Logic Circuits Digital Logic and Digital Systems	Logic gates ,Universal gates Realization of switching functions using logic gates Analyzing a Combinational Circuit	2
CE: DIG 3	Modular design of combinational circuits	Adder, Subtractor Binary Parallel Adder, Look ahead carry Magnitude Comparator, Code Converter Decoders, Encoders Multiplexers, Demultiplexers	8
CE: DIG 4 and 5	Memory Elements Sequential Logic Circuits	Flip Flops Sequential Circuit: Design and Analysis Finite State Machine: Moore and Mealy model Sequence Detector	6

This section deals with introduction of number system, binary codes and its arithmetic, Boolean algebra, Boolean function and minimization techniques.

4.2 Part II: Combinational Circuits

This section covers logic gates, universal gates, adder, subtractor, encoder, decoder, multiplexer, de multiplexer and design of a minimal combinational circuit.

4.3 Part III: Sequential Circuits

This section deals with memory elements, design of sequential circuit using memory elements, analysing a sequential circuit, design of finite state machines, registers and counters.

4.4 Part IV: Functional Blocks of a Computer

This section deals with design of functional blocks of a digital computer.

5. What is the format of the course?

This Course is designed with 150 minutes of in-classroom sessions per week, additional video/reading instructional material every week and 100 minutes of lab hours per week. Generally this course has the combination of lectures, in-class discussion, assignments, mandatory off-class reading material, quizzes.

6. How are students assessed?

- Students are assessed on a combination of assignments, continuous and final assessment tests.
- Students will be provided with problem sets for every module.

7. Session wise plan

Sl. No	Topic Covered	Class Hour	Lab Hour	levels of mastery	Text/Reference Book	Remarks
1	Introduction to logic circuits, switching, memory, registers, counters and digital systems Number System and Codes	3	2	Familiarity	1,3	

CE: DIG 6	Digital systems design	Registers, Shift Registers Counters: Ripple and Synchronous Counters, Modulo counters, Ring and Johnson counter	7			
CE: DIG 6	Analyze and design functional building blocks	Bus Organization -ALU –Design of ALU -Status Register –Design of Shifter -Processor Unit- Design of specific Arithmetic Circuits - Accumulator – Design of Accumulator	9			
CE: DIG 6 CS:AR	Recent Trends Digital Logic and Digital Systems		2			
		Total hours	45			

3. Where does the course fit in the curriculum?

This course is a

- Core Course.
- Suitable from 2nd semester onwards.
- Knowledge of basic electrical and electronics is desirable.

4. What is covered in the course?

The digital logic and design covers the digital building blocks and techniques in the design of digital systems. Emphasis is on a building-block approach. This syllabus covers a variety of basic topics, including switching theory, combinational circuits such as adders, subtractors, encoder, decoder, multiplexer, demultiplexer etc. The sequential logic covers memory elements and its application includes design of counters and registers along with functional blocks of a digital computer.

4.1 Part 1: Introduction and Boolean Algebra

	Binary Arithmetic and Complements			Usage		
2	Boolean algebra: Properties of Boolean algebra Boolean functions: Canonical and Standard forms , Simplification	3	2	Usage	1,4	
3	Minimization: Karnaugh map, Don't care conditions Tabulation Method	3		Usage	1,4	
4	Tabulation Contd., Logic gates ,Universal gates, Realization of switching functions using logic gates, Adder, Subtractor	3	2	Usage	1,4	
5	Analyzing a Combinational Circuit,Magnitude Comparator, Code Converter, Binary Parallel Adder	3	4	Assessment	1,3	
6	Look ahead carry, Decoders, Encoders Multiplexers,Demultiplexers	3	2	Usage	1,3	
7	Flip Flops Sequential Circuit: Design and Analysis	3	2	Assessment	1,3	
8	Finite State Machine: Moore and Mealy model, Sequence Detector	3	2	Usage	1,5	

10	Registers, Shift Registers , Ripple Counters	3	2	Assessment	1,5	
11	Synchronous counters, Modulo counters, Ring and Johnson counters	3	4	Usage	1	
12	BusOrganization, DesignofALU	3		Assessment	1	
13	Status Register DesignofShifter	3	2	Usage	1	
14	Processor Unit DesignofspecificArithmeticCircuits	3		Familiarity	1	
15	DesignofAccumulator Recent Trends	3		Familiarity	1	
Total hours covered		45 Hours (3 Credit hours /week 15 Weeks schedule)	30 Hours (1 Credit hours / week)			

CSE1003
Digital Logic and Design

Module 1 Introduction
Lecture 1

Dr. S. Hemamalini
Professor
School of Electrical Engineering
VIT Chennai

Module 1 Introduction 3 hours

- Number System
- Base Conversion
- Binary Codes
- Complements (Binary and Decimal)

Digital Systems

- Digital age and information age
- Digital computers
  - general purposes
  - many scientific, industrial and commercial applications
- Digital systems
  - telephone switching exchanges
  - digital camera
  - electronic calculators, PDA's
  - digital TV
- Discrete information-processing systems
  - manipulate discrete elements of information

Signal

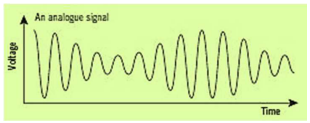
- An information variable represented by physical quantity
- For digital systems, the variable takes on discrete values
  - Two level, or binary values are the most prevalent values
- Binary values are represented abstractly by:
  - digits 0 and 1
  - words (symbols) False (F) and True (T)
  - words (symbols) Low (L) and High (H)
  - words On and Off
- Binary values are represented by values or ranges of values of physical quantities

Signals

$V(t) = 1 \quad 0 < t < t_1$
$= 0 \quad t_1 < t < t_2$

Analog signal

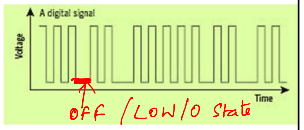
- Is a continuous signal
- Any voltage level is possible at any time
- Explicit formula
  - E.g V(t)=f(t, parameters)
- Graphical representation of the signal



$V(t) = L \frac{di(t)}{dt}$

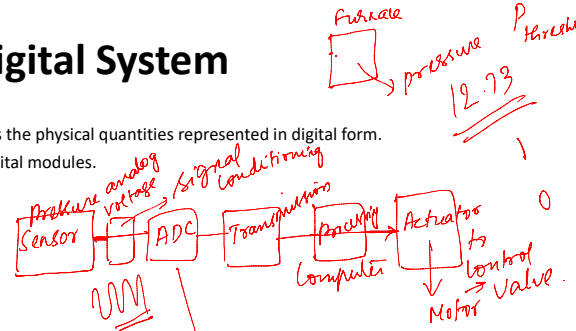
Digital signal

- Is a discrete time signal
- Discrete number of voltage levels are possible at specified time
- conveyed by the **on /off** *states* of pulses in a pulse train
- **ON** or the **off** state is a *bit,* and the time interval for the **on** or **off** state is called a *bit interval.*
- Algorithm
  - Set of conditions and operations
- V(t) can be 1 or 0



Digital System

- Digital systems contain devices that process the physical quantities represented in digital form.
- A digital system is an interconnection of digital modules.
- Digital techniques and systems -advantages
  - easier to design
  - have higher accuracy
  - easy programmability
  - noise immunity
  - easier storage of data
  - ease of fabrication in integrated circuit form, leading to availability of more complex functions in a smaller size.
- The real world, however, is analogue.
- Analogue variables are digitized at the input with the help of an analogue-to-digital converter block and reconverted back to analogue form at the output using a digital-to-analogue converter block.
- **To understand the operation of each digital module, it is necessary to have a basic knowledge of digital circuits and their logical function.**



Number Systems

- Representation of numbers
- Radix: "base", the primitive unit for group of numbers, e.g. for decimal arithmetic radix=10 ("base" 10)
- For every system, we need arithmetic operations (addition, subtraction, multiplication)
- Also, conversion from one base to the other

23-Feb-21

Common Number Systems

Decimal Number System
- Radix or Base 10
- Digits: 0,1,2,3,4,5,6,7,8,9
- Example: 1045_{10}
- First 17 positive integers
- 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16

Octal Number System
- Radix or Base 8
- Digits: 0,1,2,3,4,5,6,7
- Example: 137.21_8
- First 17 positive integers
- 0,1,2,3,4,5,6,7, 10,11,12,13,14,15,16,17,20

Binary Number System
- Radix or Base 2
- Digits: 0, 1
- Example: 1010110_2
- First 17 positive integers
- 0,1,10,11,100,101,110,111, 1000,1001,1010,1011,1100 ,1101,1110,1111,10000

Hexadecimal Number System
- Radix or Base 16
- Digits: 0,1,2,3,4,5,6,7,8,9,A,B,C,D, E,F
- Example: $EF56_{16}$
- First 17 positive integers
- 0,1,2,3,4,5,6,7,8,9,A,B,C,D, E,F,10

Introduction to Number Systems

Positional Notation
- A positive number N can be written as:

$$N = (a_{n-1}a_{n-2}a_{n-3} \ldots . a_1a_0 . a_{-1}a_{-2} \ldots . a_{-m})_r$$

. = radix point
r = radix or base of number system
n = number of integer digits to the left of radix point
m = number of fractional digits to the right of radix point
a_i = integer digit i
a_j = fractional digit j
a_{n-1} = most significant digit
a_{-m} = least significant digit

Polynomial Notation
- A positive number N can also be written as:

$$N = \sum_{i=-m}^{n-1} a_i r^i$$

where a_i is a coefficient between 0 to 9, and i denotes the weight (=10^i) of a_i

$1325_{10} = 1\times10^3 + 3\times10^2 + 2\times10^1 + 5\times10^0$
$1000 + 300 + 20 + 5 = 1325$

Summary

Dec	Bin	Hex	Octal
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7

Dec	Bin	Hex	Octal
8	1000	8	10
9	1001	9	11
10	1010	A	12
11	1011	B	13
12	1100	C	14
13	1101	D	15
14	1110	E	16
15	1111	F	17

11

Binary Arithmetic

Binary Addition

- Single Bit Addition Table

0 + 0 = 0

0 + 1 = 1

1 + 0 = 1

1 + 1 = 10 Note "carry"

```
 1 1 1
 1 1 0 1
+0 1 1 1
 1 0 1 0 0
```

carry, sum

Binary Subtraction

- Single Bit Subtraction Table

0 - 0 = 0

1 - 0 = 1

1 - 1 = 0

0 - 1 = 1 with a "borrow"

```
  0 0
  1 1 0 1
- 0 1 1 1
  0 1 1 0
```


Conversion Methods - Series substitution

Expand number in original base using

$$N = \sum_{i=-m}^{n-1} a_i r^i$$

convert from

$$N = a_{n-1}r^{n-1} + a_{n-2}r^{n-2} + \ldots + a_1 r^1 + a_0 + \cdots + a_{-m}r^{-m}$$

Most Significant Bit (MSB)

Least Significant Bit (MSB)

Binary to Decimal

$(1001.0101)_2 = 9.3125_{10}$

integer, fractional

$1001 = 1\times2^3 + 0\times2^2 + 0\times2^1 + 1\times2^0$

$= 8 + 0 + 0 + 1 = 9$

$.0101 = 0\times2^{-1} + 1\times2^{-2} + 0\times2^{-3} + 1\times2^{-4} = 0 + 0.25 + 0 + 0.0625 = 0.3125$

Binary Arithmetic

Binary Multiplication

•Single Bit Multiplication Table

0 x 0 = 0

0 x 1 = 0

1 x 0 = 0

1 x 1 = 1

```
  1 1 0 1
x 0 1 1 1
  1 1 0 1
  1 1 0 1
  1 1 0 1
  0 0 0 0
-------------
1 0 1 1 0 1 1
```

Binary Division

•Single Bit Division Table

0 / 0 = N/A

0 / 1 = 0

1 / 0 = N/A

1 / 1 = 1

$110 \div 11$

$$11\overline{)110}\ \ 10,\quad 11,\quad 000$$

CSE1003
Digital Logic and Design

Module 1 Introduction
Lecture 2

Dr. S. Hemamalini
Professor
School of Electrical Engineering
VIT Chennai

Module 1 Introduction 3 hours

- Number System
- Base Conversion
- Binary Codes
- Complements (Binary and Decimal)

Decimal to Binary Conversion - Radix Divide Method

Converting decimal integer to binary

- To convert a decimal integer to binary, repeatedly divide by 2 (successive division) until the quotient is zero - Double-Dabble method.
- Last reminder is the MSB. Read the reminders in reverse order (bottom to top) to get the binary form of the number.

Converting decimal fraction to binary

- To convert a decimal fraction to binary, repeatedly multiply the fractional part by 2 till the fraction part of the product is zero or till the desired accuracy is obtained.
- The first integer obtained is the MSB. Read the integers from top to bottom to get the binary equivalent fraction.

Base Conversion Methods - Series substitution

Positional Weights Method

Each binary digit of the number is multiplied by its position weight and the product terms are added to obtain the decimal number.

Expand number in original base using

$$N = \sum_{i=-m}^{n-1} a_i r^i$$

$$N = a_{n-1}r^{n-1} + a_{n-2}r^{n-2} + \ldots + a_1 r^1 + a_0 + \cdots + a_{-m}r^{-m}$$

Most Significant Bit (MSB) — Least Significant Bit (MSB)

Binary to decimal conversion

Convert 11011.101_2 to decimal.

$11011.101 = 1\times2^4 + 1\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0 + 1\times2^{-1} + 0\times2^{-2} + 1\times2^{-3}$

$= 16 + 8 + 0 + 2 + 1 + 0.5 + 0 + 0.125 = 27.625_{10}$

Decimal to Binary Conversion - Radix Divide Method

Convert 163.875_{10} to binary.

Integer part conversion

163	Quotient	Remainder	
163/2	81	1	LSB
81/2	40	1	
40/2	20	0	
20/2	10	0	
10/2	5	0	
5/2	2	1	
2/2	1	0	
1/2			MSB

Fractional part conversion

0.875

0.875×2	1.75	MSB
0.75×2	1.5	
0.5×2	1.0	LSB

$163.875_{10} = 10100011.111_2$

Practice problem: Convert 105.15_{10} to binary

Most Significant Bit

Octal to Binary Conversion

Octal to Binary Conversion

Replace each octal digit by its 3-bit binary equivalent.

Convert an octal to binary by substituting the binary equivalents for each digit.

$317.2_8 = 011\ 001\ 111.010_2$

Binary to Octal Conversion

- To convert binary to octal, arrange the bits in groups of three, starting from the binary point and working outward
- Insert leading or trailing zeros to complete the groups.
- Convert each group of three bits into its octal equivalent.

$011110110.1_2 = 011\ 110\ 110.100 = 366.4_8$

Octal to decimal conversion

Multiply each digit in the octal number by the weight of its position and add all the product terms.

Convert 4057.06_8 to decimal.

$4057.06_8 = 4\times8^3 + 0\times8^2 + 5\times8^1 + 7\times8^0 + 0\times8^{-1} + 6\times8^{-2}$

$= 2095.09375_{10}$

Decimal to Octal conversion

To convert the given decimal integer number to octal, successively divide the given number by 8 till the quotient is 0.
The last reminder is the MSD. Reminders read upwards give the equivalent octal integer number.

572.7341_8

Convert 378.93_{10} to octal.

Integer

	Q	R	
378/8	47	2	LSB
47/8	5	7	
5/8	0	5	MSB

Fraction

0.93 x 8	7.44	7 MSB
0.44 x 8	3.52	3
0.52 x 8	4.16	4
0.16 x 8	1.28	1 LSB.

Octal to Binary Conversion

Convert 367.52_8 to binary.

$011\ 110\ 111.\ 101\ 010_2$

Convert $010\ 101\ 111\ 001.\ 011\ 100_2$ to octal.

$010\ 101\ 111\ 001.\ 011\ 100$

$2\ \ 5\ \ 7\ \ 1.\ 3\ 4_8$

Hex to Binary Conversion
Binary to Hex Conversion

Use the conversion table

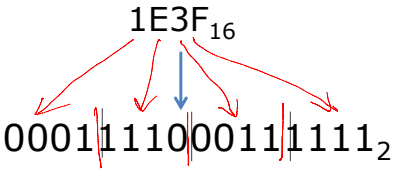
Bin	Hex	Bin	Hex
0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
0011	3	1011	B
0100	4	1100	C
0101	5	1101	D
0110	6	1110	E
0111	7	1111	F

Binary to Hex Conversion

1. Divide binary number into 4-bit groups.
2. Substitute hex digit for each group

$01\ 1\ 0\ 0\ 0\ 1\ 1\ 1\ 0\ 0 \rightarrow 61C_{16}$

Hex to Binary Conversion



Hexadecimal to Decimal Conversion

Multiply each digit in the hexadecimal number by its position weight and add all those product terms.

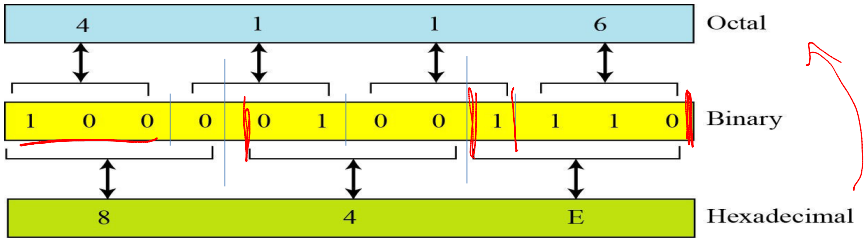
$5C7_{16} = 5\times16^2 + C\times16^1 + 7\times16^0$

$= 5\times16^2 + 12\times16^1 + 7\times16^0$

$= 1479_{10}$

$A0F9.0EB_{16} = 41209.0572_{10}$

Octal to hexadecimal and hexadecimal to octal conversion



Decimal to Hexadecimal Conversion
Radix Divide Method

- Successively divide the integer by 16 till the quotient is zero. Last reminder is MSB.
- Read the reminder from bottom to top and that gives the equ. Hexadecimal integer.
- Multiply the given fraction number by 16 till the product is zero or till the required accuracy is reached.
- First integer is the MSB. Read top to bottom to get the equ. fractional part.

Convert 2598.675_{10} to hexadecimal.

	Q	R	R
2598/16	162	6	6 LSB
162/16	10	2	2
10/16	0	10	A MSB

Decimal Hex.

$0.675\times16 = 10.8$ A

$0.800\times16 = 12.8$ C

$0.800\times16 = 12.8$ C

$0.800\times16 = 12.8$ C

$2598.675_{10} = A26.ACCC_{16}$

Practice problems

1. Convert 756.603_8 to hexadecimal.
2. Convert $B9F.AE_{16}$ to octal.
3. Convert 10110111011_2 to hexadecimal.
4. Convert $4BAC_{16}$ to binary.

CSE1003
Digital Logic and Design

Module 1 Introduction
Lecture 2

Dr. S. Hemamalini
Professor
School of Electrical Engineering
VIT Chennai

Module 1 Introduction 3 hours

- Number System
- Base Conversion
- Binary Codes
- Complements (Binary and Decimal)

SIGNED BINARY NUMBERS

- Sign-magnitude representation
- 1's complement representation
- 2's complement representation

Decimal
+9 −9
+1001 −1001

1 for minus sign
0 for plus sign

Sign-magnitude Representation

- An additional bit is used as the *sign bit*.
- This sign bit is usually placed as the MSB.
- A 0 is reserved for a positive number and a 1 is reserved for a negative number.
- 8-bit signed binary number

+105
−105

−(01101001)$_2$ = +(105)$_{10}$
MSB
+ve sign
Sign
magnitude 7 bits

−11101001 = −(105)$_{10}$
MSB
1 − ve no.

(+44)$_{10}$ ⇓ 1101100$_2$

Eg. 0 1 1 0 0 1 0 0 (2^5 2^4 2^3 2^2 2^1 2^0)
MSB
+ve
(+44)$_{10}$
find the decimal equivalent

$1\times2^5+0\times2^4+1\times2^3+1\times2^2+0\times2^1+0\times2^0$
$= 44_{10}$

Find the decimal equivalent of the following binary numbers assuming the binary numbers have been represented in sign-magnitude form.

(a) 0101100 (b) 101000 (c) 1111 (d) 011011

Solution.

(a) Sign bit is 0, which indicates the number is positive.
Magnitude $101100 = (44)_{10}$
Therefore $(0101100)_2 = (+44)_{10}$.

(b) Sign bit is 1, which indicates the number is negative.
Magnitude $01000 = (8)_{10}$
Therefore $(101000)_2 = (-8)_{10}$.

(c) Sign bit is 1, which indicates the number is negative.
Magnitude $111 = (7)_{10}$
Therefore $(1111)_2 = (-7)_{10}$.

(d) Sign bit is 0, which indicates the number is positive.
Magnitude $11011 = (27)_{10}$
Therefore $(011011)_2 = (+27)_{10}$.

1's Complement Representation

- In 1's complement representation, both numbers are a complement of each other.
- If one of the numbers is positive, then the other will be negative with the same magnitude and vice versa.

Binary Decimal
0111_2 $(+7)_{10}$
1000_2 → 1's complement representation
→ $(-7)_{10}$

Represent +5 & −5 in 1's complement form
$(+5)_{10} \rightarrow (0101)_2$ $(-5) \rightarrow (1010)_2$

2's Complement Representation

- If 1 is added to 1's complement of a binary number, the resulting number is 2's complement of that binary number.

0110_2 $(+6)_{10}$ ⇒
0110 Given no.
1001 1's complement
+ 1
1010 2's complement
→ $(-6)_{10}$.

Binary group — 1's comple 2's comple
0000 +0 +0.
1111 −0 +1

1010 $(+5)_{10} \rightarrow (0101)_2$ 1010 → (−5)
1011 → (−5)

Represent (–19) in
(a) Sign-magnitude,
(b) one's complement, and
(c) two's complement representation.

Binary form

19/2 9 1
9/2 4 1
4/2 2 0
2/2 1 0

(a) 110011 sign magnitude representation
sign | magnitude

(b) 101100 1's complement representation

(c) 101101 2's complement representation

$(-19) \rightarrow (110011)_2$

COMPLEMENTS

- Complements are used in digital computers for simplifying the subtraction operation and for logical manipulations.
- There are two types of complements for each number system of base-*r*: the *r*'s complement and the (*r* – 1)'s complement.
- For a binary system, the value of *r* is 2 and hence the complements are 2's and 1's complements.
- For a decimal system the value of *r* is 10 and the complements are 10's and 9's complements.
- With the same logic if the number system is octal the complements are 8's and 7's complement, while it is 16's and 15's complements for hexadecimal system.

The *r*'s Complement

- If a positive number *N* is given in base *r* with an integer part of *n* digits, the *r*'s complement of *N* is given as $r^n - N$ for $N \neq 0$ and 0 for $N = 0$.

(*r*–1)'s Complement

- If a positive number *N* is given in base *r* with an integer part of *n* digits and a fraction part of *m* digits, then the (*r* – 1)'s complement of *N* is given as $(r^n - r^{-m} - N)$ for $N \neq 0$ and 0 for $N = 0$.

1'S AND 2'S COMPLEMENT ARITHMETIC

Subtraction Using 1's Complement

- Binary subtraction can be performed by adding the 1's complement of the subtrahend to the minuend.
- If a carry is generated, remove the carry, add it to the result. This carry is called the end-around carry.
- If the subtrahend is larger than the minuend, then no carry is generated.
- The answer obtained is 1's complement of the true result and opposite in sign.

Subtract binary 10 from binary 101 using 1's complement. $(10)_2 \rightarrow 2_{10}$ $(101)_2 \rightarrow 5_{10}$

5 Minuend 101 101 Minuend
−2 Subtrahend −010 101 1's complement of 010
+3 ? 1010
End of carry (EAC) → (1)010
+1 EAC added
011
↑
+3

Subtraction Using 2's Complement

- Binary subtraction can be performed by adding the 2's complement of the subtrahend to the minuend.
- If a carry is generated, discard the carry.
- Now if the subtrahend is larger than the minuend, then no carry is generated.
- The answer obtained is in 2's complement and is negative.
- To get a true answer take the 2's complement of the number and change the sign.
- The advantage of the 2's complement method is that the end-around carry operation present in the 1's complement method is not present here.

Subtract binary 110 from binary 101 using 1's complement. $(110)_2 \rightarrow 6_{10}$ $101 \rightarrow 5_{10}$

Decimal Binary
5 Minuend 101 101 Minuend
−6 Subtrahend −110 001 1's complement of 110
−1 ? 110 SUM NO end of carry
↑
1's complement form of −1

Subtract binary 10 from binary 101 using 2's complement.

5 Minuend 101 101 Minuend
−2 Subtrahend −010 +110 2's complement of 010
+3 Difference ? (1)011 SUM
Extra carry ignored
011
↑
+3

Subtract binary 011 from 001 using 2's complement.

1	001	001	Minuend
−3	−011	+101	2's complement
−2	?	110	sum

No carry

2's complement form of 110 → −2

CSE1003
Digital Logic and Design

Module 2
BOOLEAN ALGEBRA L1

Dr. S. Hemamalini
Professor
School of Electrical Engineering
VIT Chennai

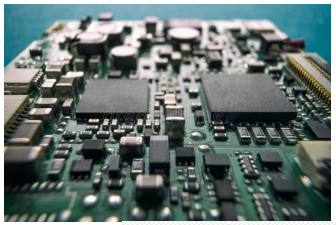
Module 2
BOOLEAN ALGEBRA 8 hrs

- Boolean algebra
- Properties of Boolean algebra
- Boolean functions
- Canonical and Standard forms
- Logic gates - Universal gates
- Karnaugh map - Don't care conditions
- Tabulation Method

Logic Gates

- The logic gate is the most basic building block of any digital system, including computers.
- Each one of the basic logic gates is a piece of hardware or an electronic circuit that can be used to implement some basic logic expression, also known as Boolean expressions.
- There are three basic logic gates, namely the OR gate, the AND gate and the NOT gate.
- Other logic gates that are derived from these basic gates are the NAND gate, the NOR gate, the EXCLUSIVEOR gate and the EXCLUSIVE-NOR gate.

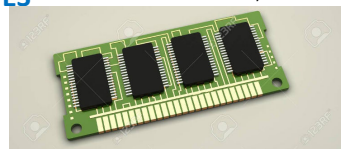
APPLICATIONS OF LOGIC GATES



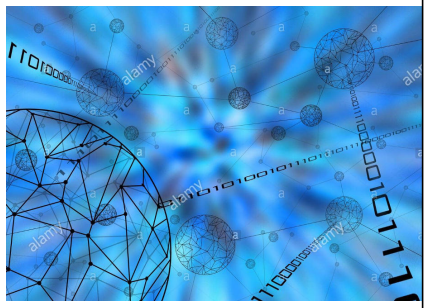
Microprocessors

Flash Memories

Random access memory

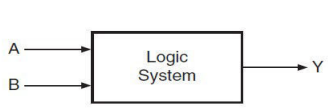


Digital Data Transmission and Internet



Truth Table

- A truth table lists all possible combinations of input binary variables and the corresponding outputs of a logic system.
- The logic system output can be found from the logic expression, often referred to as the Boolean expression, that relates the output with the inputs of that very logic system.



A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

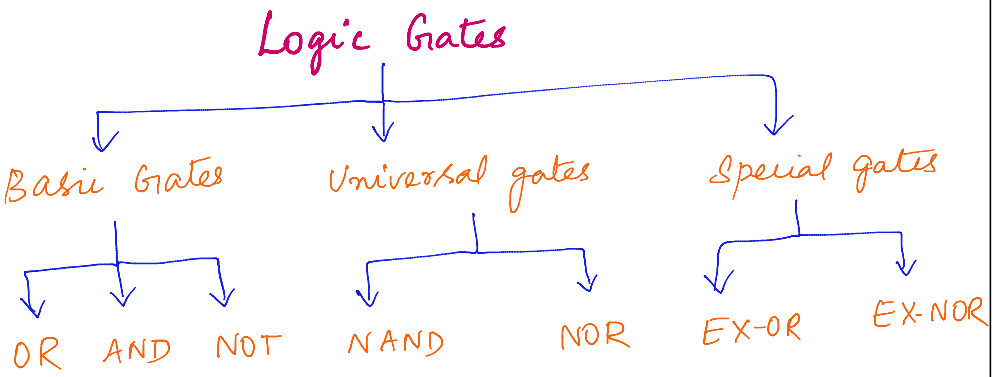
Two-input logic system

A	B	C	Y
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

Truth table of a three-input logic system

Positive and Negative Logic

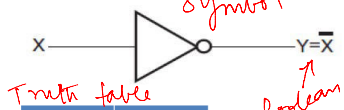
- The binary variables, can have either of the two states, i.e. the logic '0' state or the logic '1' state.
- These logic states in digital systems such as computers, for instance, are represented by two different voltage levels or two different current levels.



NOT Gate

Inverter gate

Symbol

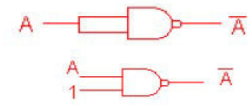


Truth table

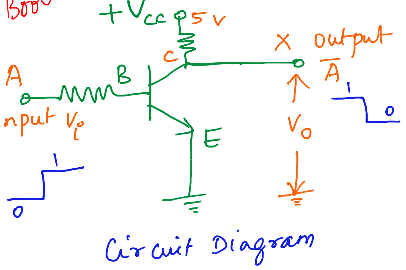
Input A	Output $\bar{A}$
0	1
1	0

A NOT gate or *inverter* outputs a logic level that's the opposite (complement) of the input logic level.

Boolean form. X'

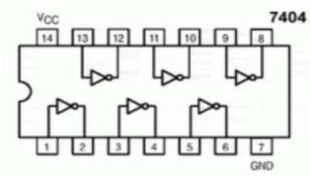


Inverter

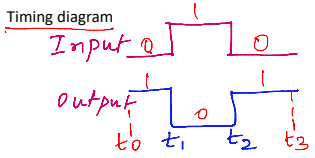


Circuit Diagram

PIN DIAGRAM:

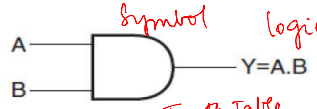


Timing diagram



AND GATE

Symbol

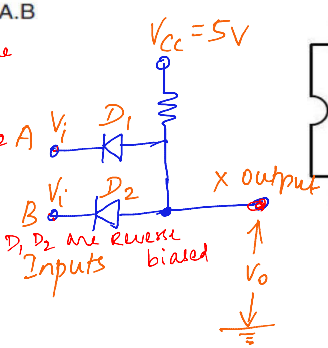


Logical Multiplication

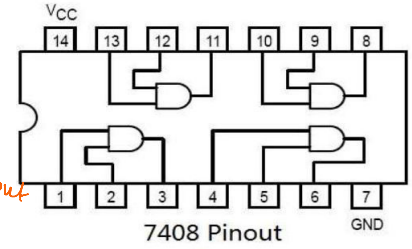
Truth Table

Input A	Input B	Output A.B
0	0	0
0	1	0
1	0	0
1	1	1

The output of an AND gate is HIGH only when both inputs are HIGH.



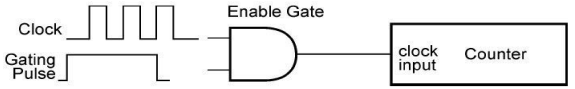
PIN DIAGRAM:



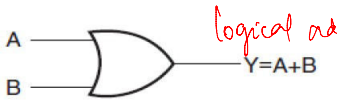
7408 Pinout

Application of an AND gate

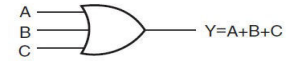
- An AND gate is commonly used as an ENABLE or INHIBIT gate to allow or disallow passage of data from one point in the circuit to another.
- One such application of enabling operation, for instance, is in the measurement of the frequency of a pulsed waveform or the width of a given pulse with the help of a counter.
- In the case of frequency measurement, a gating pulse of known width is used to enable the passage of the pulse waveform to the counter's clock input.
- In the case of pulse width measurement, the pulse is used to enable the passage of the clock input to the counter.



OR GATE

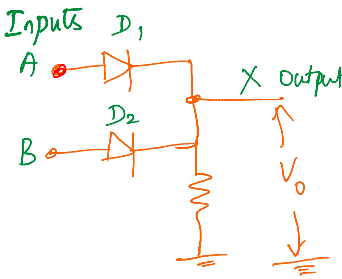


Logical addition

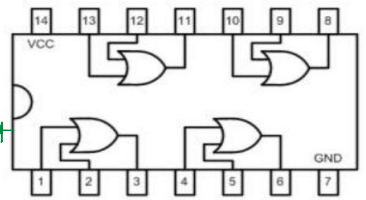


Input A	Input B	Output A+B
0	0	0
0	1	1
1	0	1
1	1	1

The output of an OR gate will go HIGH if one or both inputs goes HIGH. The output only goes LOW when both inputs are LOW.



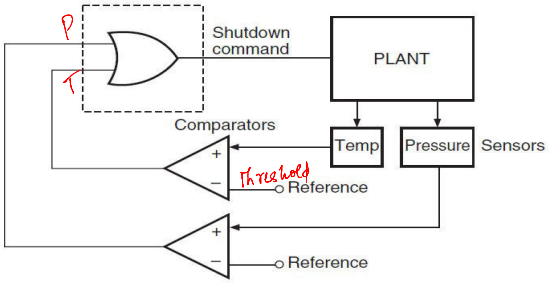
PIN DIAGRAM:



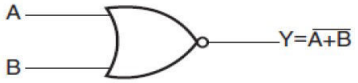
7432 Quad 2 Input OR

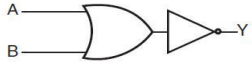
Application of an OR gate

- An OR gate can be used in all those situations where the occurrence of any one or more than one event needs to be detected or acted upon.

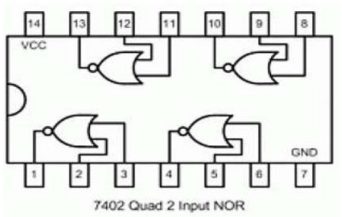


NOR Gate





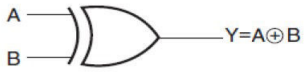
PIN DIAGRAM :



Input A	Input B	Output $\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

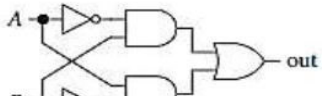
Combines the NOT function with an OR gate; output goes LOW if one or both inputs are LOW, output goes HIGH when both inputs are LOW.

X-OR Gate



Input A	Input B	Output $\overline{A}B + A\overline{B}$
0	0	0
0	1	1
1	0	1
1	1	0

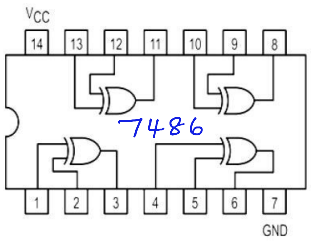
The output of an XOR gate goes HIGH if the inputs are different from each other. XOR gates only come with two inputs.



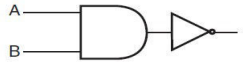
equivalent circuit

$$Y = (A \oplus B) = \overline{A}B + A\overline{B}$$

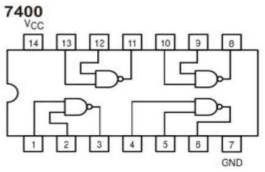
PIN DIAGRAM :



NAND Gate

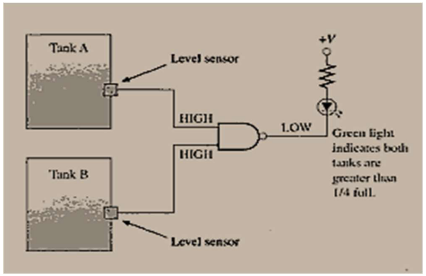


PIN DIAGRAM :

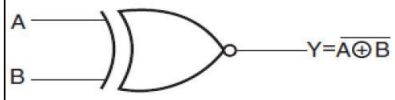


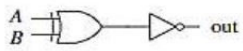
Input A	Input B	Output $\overline{A.B}$
0	0	1
0	1	1
1	0	1
1	1	0

Combines the NOT function with an AND gate; output only goes LOW when both inputs are HIGH.



X-NOR Gate





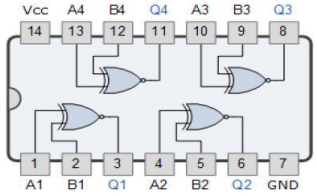
equivalent circuit

Input A	Input B	Output $A.B + \bar{A}.\bar{B}$
0	0	1
0	1	0
1	0	0
1	1	1

Combines the NOT function with an XOR gate; output goes HIGH if the inputs are the same.

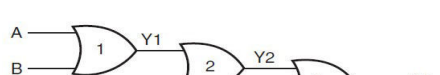
$$Y = (\overline{A \oplus B}) = (A.B + \overline{A}.\overline{B})$$

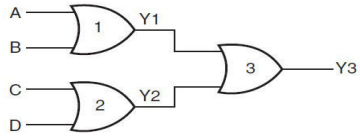
PIN DIAGRAM :



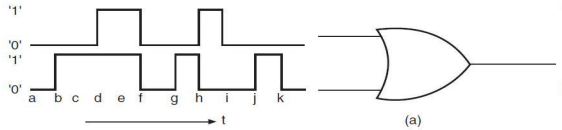
OR GATE

- ***How would you hardware-implement a four-input OR gate using two-input OR gates only?***



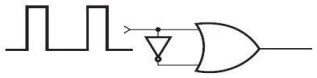


- ***Draw the output waveform for the OR gate and the given pulsed input waveforms of Fig.***

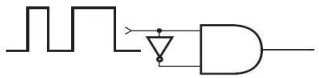


NOT Gate

- *For the logic circuit arrangements of Figs (a) and (b), draw the output waveform.*

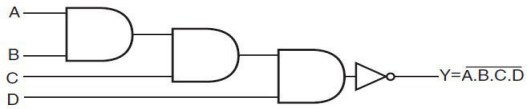


In the case of the OR gate arrangement of Fig. (a), the output will be permanently in logic '1' state as the two inputs can never be in logic '0' state together owing to the presence of the inverter.

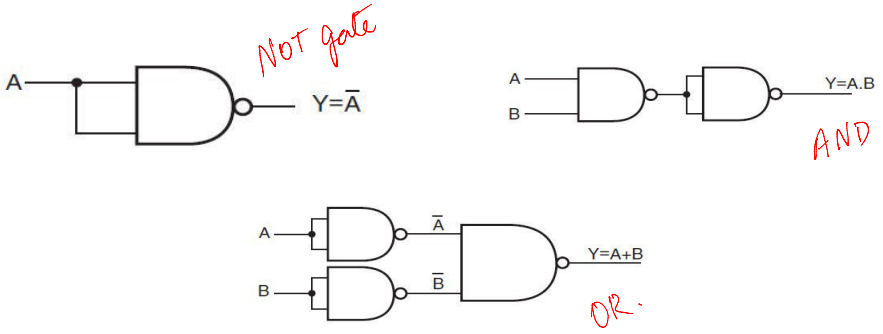


In the case of the AND gate arrangement of Fig. (b), the output will be permanently in logic '0' state as the two inputs can never be in logic '1' state together owing to the presence of the inverter.

Show the logic arrangements for implementing:
(a) a four-input NAND gate using two-input AND gates and NOT gates

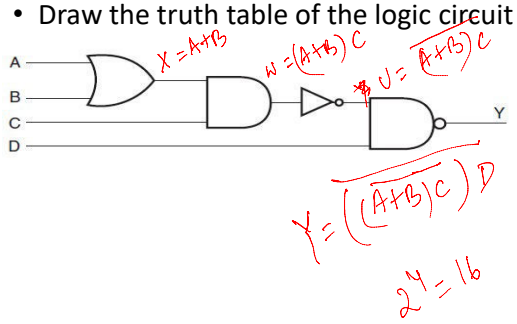


Implementation of basic logic gates using only NAND gates



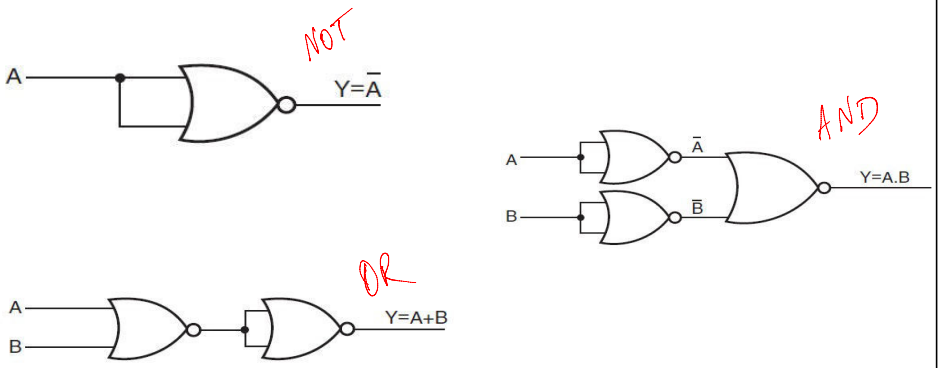
Do it by yourself

- Draw the truth table of the logic circuit shown in



A	B	C	D	Y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

Implementation of basic logic gates using only NOR gates



CSE1003
Digital Logic and Design

Module 2
BOOLEAN ALGEBRA L2

Dr. S. Hemamalini
Professor
School of Electrical Engineering
VIT Chennai

Module 2
BOOLEAN ALGEBRA 8 hrs

Boolean algebra

- Properties of Boolean algebra
- Boolean functions
- Canonical and Standard forms
- Logic gates - Universal gates
- Karnaugh map - Don't care conditions
- Tabulation Method

Boolean algebra

Boolean function

- Is an expression formed with binary variables, Boolean operators and the equality sign.
- Can also be represented in the form of a truth table.

Expression: a set of literals (possibly with repeats) combined with logic operations (and possibly ordered by parentheses)

$AB + C$ $\quad (\bar{A}+B)C$

Equation: expression1 = expression2

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

Function of (possibly several) variables: an equation where the left hand side is defined by the right hand side

$f(A,B,C) = ((\bar{A})+B)C$

Boolean Algebra

Logic 0	Logic 1
False	True
Off	On
Low	High
No	Yes
Open switch	Closed switch

1854
George Boole

- Boolean algebra is mathematics of logic.
- The algebra which deals with the logical operations of binary variables is called Boolean Algebra.
- **Boolean algebra** is a means for expressing the relationship between a logic circuit's inputs and outputs.
- The inputs are considered **logic variables** whose logic levels at any time determine the output levels.
- It is one of the most basic tools available to the logic designer and thus can be effectively used for simplification of complex logic expressions.

Boolean algebra

Variables, Literals and Terms in Boolean Expressions

<5 1
>2
0.8 } 0
0

- ***Variables*** are the letters in a Boolean expression which represents a physical quantity such as a voltage signal. They may take on the value '0' or '1'.
- The complement of a variable is not considered as a separate variable. Each occurrence of a variable or its complement is called a ***literal***.
- A **term** is the expression formed by literals and operations at one level.

$\bar{A} + A.B + A.\bar{C} + \bar{A}.B.C$

$(\bar{P}+Q).(R+\bar{S}).(P+\bar{Q}+R)$

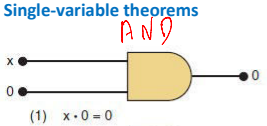
A B C
$\bar{A}$ $\bar{B}$ $\bar{C}$

BOOLEAN THEOREMS

- help to simplify logic expressions and logic circuits

Single-variable theorems

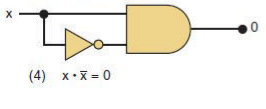
AND



(1) $x \cdot 0 = 0$

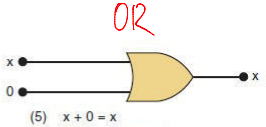
(2) $x \cdot 1 = x$

(3) $x \cdot x = x$

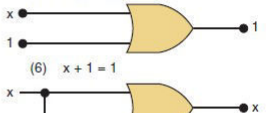


(4) $x \cdot \bar{x} = 0$

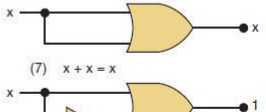
OR



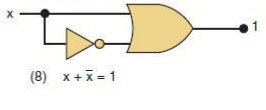
(5) $x + 0 = x$



(6) $x + 1 = 1$



(7) $x + x = x$



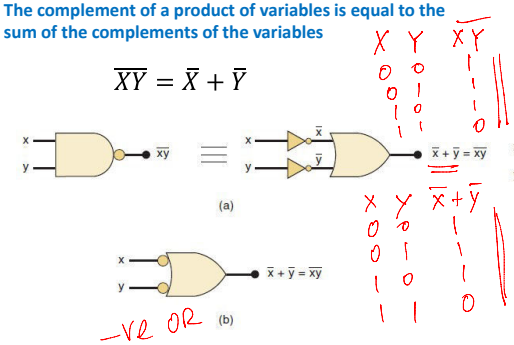
(8) $x + \bar{x} = 1$

DeMorgan's Theorem

DeMorgan's theorems are extremely useful in simplifying expressions in which a product or sum of variables is inverted.

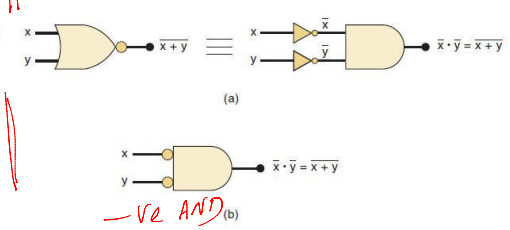
The complement of a product of variables is equal to the sum of the complements of the variables

$$\overline{XY} = \bar{X} + \bar{Y}$$



The complement of a sum of variables is equal to the product of the complements of the variables.

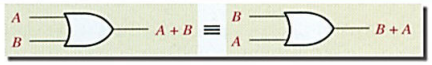
$$\overline{X+Y} = \bar{X}.\bar{Y}$$



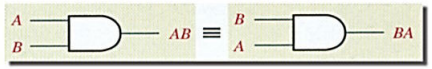
Laws of Boolean Algebra

- Commutative Laws

$A + B = B + A$

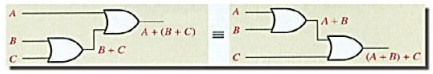


$A \cdot B = B \cdot A$

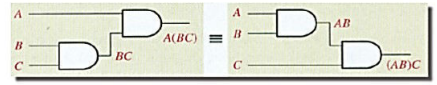


- Associative Laws

$A + (B + C) = (A + B) + C$



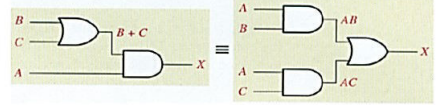
$A \cdot (B \cdot C) = (A \cdot B) \cdot C$



- Distributive Law

$A \cdot (B + C) = A \cdot B + A \cdot C$

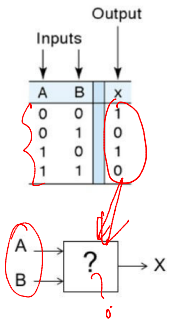
$A\,(B + C) = A\,B + A\,C$



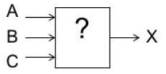
Boolean Logic

- Truth Table: all combinations of input variables
- k variables ➜ 2^k input combinations

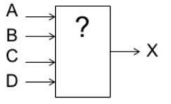
Truth Tables specifies how a logic circuit's output depends on the logic levels present at the inputs.



A	B	C	x
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1



A	B	C	D	x
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1



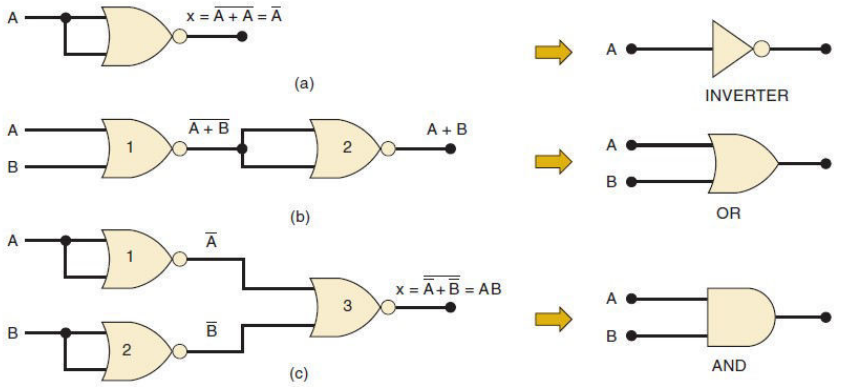
Laws of Boolean Algebra used for simplifying logical expressions

($A^I = \bar{A}$ denotes the complement/inverse/NOT of A)

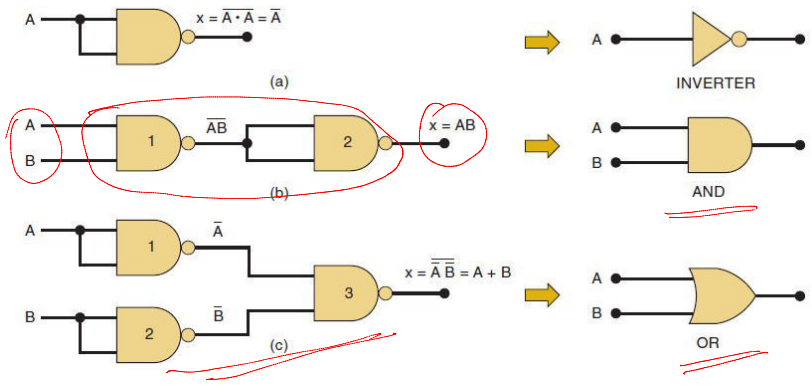
$A+0=A$	$(A^I)^I = A$
$A+1=A$	$A+AB=A$
$A\cdot 0=0$	$A+A^IB=A+B$
$A\cdot 1=A$	$A+B=B+A$
$A+A=A$	$A\cdot B=B\cdot A$
$A+A^I=1$	$\overline{A+B}=\bar{A}\cdot\bar{B}$
$A\cdot A=A$	$\overline{A\cdot B}=\bar{A}+\bar{B}$
$A\cdot A^I=0$	$(A+B)(A+C)=A+BC$

A.B.1=A.B
A+B+1=1
A.B.0=0
A+B+0=A+B

UNIVERSALITY OF NOR GATES



UNIVERSALITY OF NAND GATES



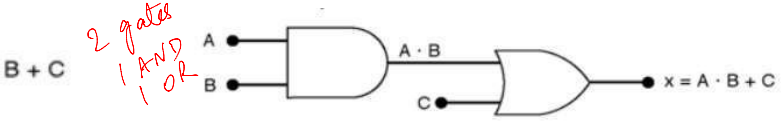
Precedence rules in Boolean algebra

- Scan the expression from left to right.
- First evaluate expression enclosed in parentheses.
- Perform all the complement operations.
- Perform all the AND operations in the order.
- Perform all the OR operations.

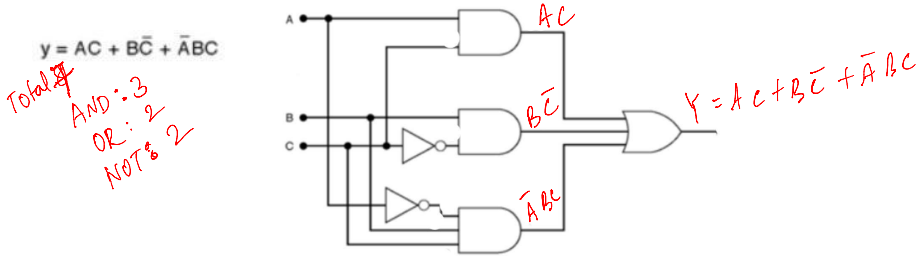
DESCRIBING LOGIC CIRCUITS ALGEBRAICALLY FROM BOOLEAN EXPRESSIONS

A Boolean function from an algebraic expression can be realized to a logic diagram composed of logic gates.

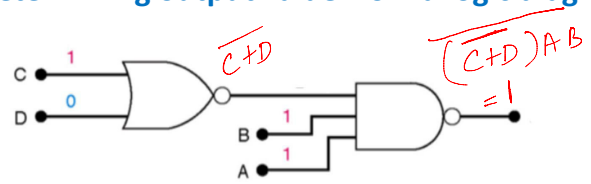
$x = A \cdot B + C$

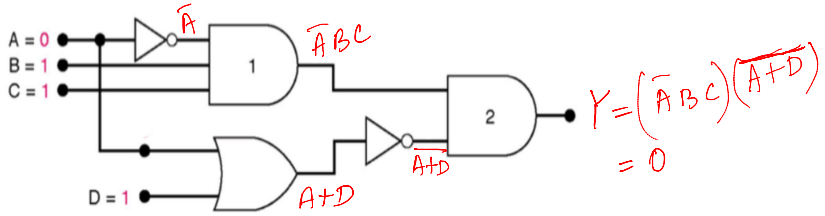


$y = AC + B\bar{C} + \bar{A}BC$



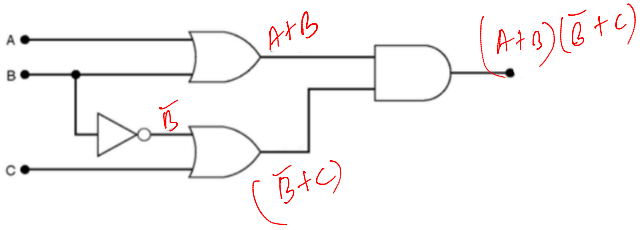
Determining output value from a logic diagram





DESCRIBING LOGIC CIRCUITS ALGEBRAICALLY FROM BOOLEAN EXPRESSIONS

$x = (A + B)(\bar{B} + C)$



Simplification Techniques

- The primary objective of all simplification procedures is to obtain an expression that has the minimum number of terms.
- Obtaining an expression with the minimum number of literals is usually the secondary objective.
- If there is more than one possible solution with the same number of terms, the one having the minimum number of literals is the choice.

Algebraic Method
K-Map
Tabulation

$Y = ABC + \bar{A}B\bar{C} + A\bar{C} + B\bar{C}A$
4 terms

A+C

Simplify using Boolean Algebra

$y = A\bar{B}D + A\bar{B}\bar{D} \rightarrow y = A\bar{B}$

$z = (\bar{A} + B)(A + B) \rightarrow z = B$

$x = ACD + \bar{A}BCD \rightarrow x = ACD + BCD$

$z = (\bar{A} + C)\cdot(B + \bar{D}) \rightarrow z = A\bar{C} + \bar{B}D$

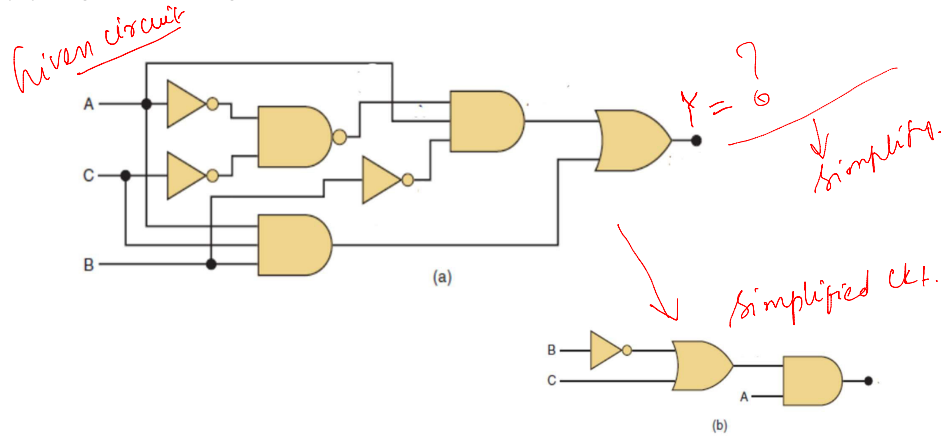
$A\bar{B}(D+\bar{D}) = A\bar{B}$ $\quad D+\bar{D}=1$

$\bar{A}A + AB + \bar{A}B + BB = 0 + AB + \bar{A}B + B = 0 + (A+\bar{A})B + B = B + B = B$

$(A + \bar{A}B)CD = (A+B)CD = ACD + BCD$

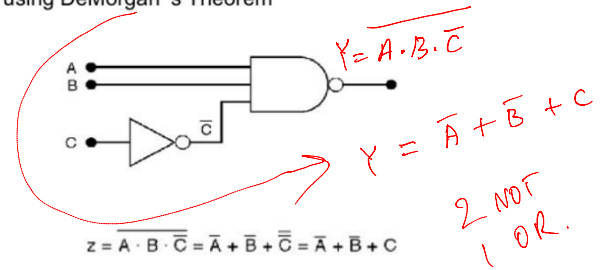
$(\bar{A}+C) + (\bar{B}+\bar{D}) = \bar{A}\cdot\bar{C} + \bar{B}\cdot\bar{D} = A\bar{C} + \bar{B}D$

Simplify the logic circuit shown in Figure



Simplify using Boolean Algebra

Determine the output expression for the circuit below and simplify it using DeMorgan's Theorem



$Y = \overline{A\cdot B\cdot \bar{C}}$

$Y = \bar{A} + \bar{B} + C$

2 NOT
1 OR.

$z = \overline{A\cdot B\cdot \bar{C}} = \bar{A} + \bar{B} + \bar{\bar{C}} = \bar{A} + \bar{B} + C$

CSE1003
Digital Logic and Design

Module 2
BOOLEAN ALGEBRA L3

Dr. S. Hemamalini
Professor
School of Electrical Engineering
VIT Chennai

Module 2
BOOLEAN ALGEBRA 8 hrs

Boolean algebra

- Properties of Boolean algebra
- Boolean functions
- Canonical and Standard forms
- Logic gates - Universal gates
- Karnaugh map - Don't care conditions
- Tabulation Method

CANONICAL AND STANDARD FORMS

- ***Sum of Products* (SOP):** The logical sum of two or more logical product terms is referred to as a sum of products expression. It is basically an OR operation on AND operated variables. This form is also called the Disjunctive Normal Form (DNF).

Y = AB + BC + AC
Y = A'B + BC + AC'

- In a sum-of-products form, a single overbar cannot extend over more than one variable, although more than one variable in a term can have an overbar.
- For example, $\bar{A}\,B\bar{C}$ can be a term in sum-of-products expression but not $\overline{ABC}$.

$Y = A'B + B'A$

CANONICAL AND STANDARD FORMS

- An arbitrary logic function can be expressed in the following forms.

(*i*) Sum of the Products (SOP)

(*ii*) Product of the Sums (POS)

- ***Product Term*:** In Boolean algebra, the logical product of several variables on which a function depends is considered to be a product term. In other words, the AND function is referred to as a product term or standard product.
- ***Sum Term*:** The logical sum of several variables on which a function depends is considered to be a sum term. An OR function is referred to as a sum term.

$f(A, B, C) = A + B + C$

Product of Sums (POS)

- The logical product of two or more logical sum terms is called a product of sums expression.
- It is an AND operation on OR operated variables.
- This form is also called the Conjunctive Normal Form (CNF).

Y = (A + B + C)(A + B' + C)(A + B + C')
Y = (A + B + C)(A' + B' + C')

- In the product-of-sums (POS) expression, a single overbar cannot extend over more than one variable, although more than one variable in a term can have an overbar.
- For example, a product-of-sums (POS) expression can have the term $\bar{A} + B + \bar{C}$ but not $\overline{A + B + C}$.

CANONICAL AND STANDARD FORMS

***Standard form*:** The standard form of the Boolean function is when it is expressed in sum of the products or product of the sums fashion.

Standard or Canonical Sum-of-products (SOP) Form

- If each term in the sum of products form contains all the variables (literals), then the expression is known as standard sum of products form or canonical sum of products form.
- In this form each product term contains all the variables of the function either in complemented or uncomplemented form.

$$f(A,B,C,D) = AB\overline{C}D + \overline{A}\,\overline{B}C\overline{D} + \overline{A}B\overline{C}D$$

- **MINTERM**
- Each of the product terms in the standard SOP form is called a minterm i.e., a product term which contains all the variables of the function either in complemented or uncomplemented form is called a minterm.

Minterms for 3 variables and their designation

A	B	C	Minterm	Designation
0	0	0	$\overline{A}\,\overline{B}\,\overline{C}$	m_0
0	0	1	$\overline{A}\,\overline{B}C$	m_1
0	1	0	$\overline{A}B\overline{C}$	m_2
0	1	1	$\overline{A}BC$	m_3
1	0	0	$A\overline{B}\,\overline{C}$	m_4
1	0	1	$A\overline{B}C$	m_5
1	1	0	$AB\overline{C}$	m_6
1	1	1	ABC	m_7

Σ Notation

Σ notation is used to represent sum-of-products Boolean expressions.

Standard SOP form:

$$f(A,B,C) = \overline{A}\,\overline{B}C + \overline{A}B\overline{C} + \overline{A}BC + A\overline{B}C$$

Another way of representing the function in canonical SOP form is by showing the sum of minterms for which the function value equals 1.

$$f(A,B,C) = m_1 + m_2 + m_3 + m_5$$

Another way of representing the function in canonical form is by listing the decimal equivalents of the minterms for which $f = 1$.

$$f(A,B,C) = \sum m(1,2,3,5)$$

MINTERM

1. An n variable function can have in all 2^n minterms. A minterm is equal to 1 for only one combination of variables.

For example the term $A\bar{B}C\bar{D}$ is equal to 1 only when $A = 1, B = 0, C = 1$ and $D = 0$.

2. The sum of the minterms whose value is equal to 1 is the standard sum of products form of the function.

3. The minterms are often denoted as m_0 , m_1, m_2,..., where the subscripts are the decimal equivalent of the binary number of the minterms.

4. For minterms, the binary words are formed by representing each non-complemented variable by a 1 and each complemented variable by a 0.

For example, for the minterm $A\bar{B}\bar{C}D$ binary number is 1001 and decimal equivalent is 9. Hence it is represented as $m9$

Standard product of sum (POS) form

- **Maxterm:** A sum term which contains each of the n variables in either complemented or uncomplemented form is called a maxterm.

Maxterms for 3 variables and their designation

A	B	C	Maxterm	Designation
0	0	0	$A+B+C$	M_0
0	0	1	$A+B+\overline{C}$	M_1
0	1	0	$A+\overline{B}+C$	M_2
0	1	1	$A+\overline{B}+\overline{C}$	M_3
1	0	0	$\overline{A}+B+C$	M_4
1	0	1	$\overline{A}+B+\overline{C}$	M_5
1	1	0	$\overline{A}+\overline{B}+C$	M_6
1	1	1	$\overline{A}+\overline{B}+\overline{C}$	M_7

Maxterms

1. For an *n*-variable function, there will be at the most 2^n maxterms. A maxterm assumes the value 0 only for one combination of the variables.

For example, the term $A + \overline{B} + C + \overline{D}$ is 0 only when $A = 0, B = 1, C = 0$ and $D = 1$. For all other combinations it will be 1.

2. The product of maxterms whose value is 0 gives the standard or canonical product of sums form of the function.

3. Maxterms are often represented as M_0 , M_1, M_2,....,where the subscripts denote decimal equivalent of the binary number of the maxterms.

4. For maxterms, the binary words are formed by representing each non-complemented variable by a 0 and each complemented variable by a 1.

For example, for the maxterm $A + \overline{B} + \overline{C} + D$ binary number is 0110 and decimal equivalent is 6. Hence, it is represented as M_6 .

CONVERTING EXPRESSIONS TO STANDARD SOP OR POS FORMS

Converting SOP Form to Standard SOP Form

1. Find the missing literal in each product term if any.

2. If one or more variables are missing in any term, expand that term by multiplying it with the sum of each one of the missing variables and its complement.

For example, in a three variable SOP function consider a term AB' . The third variable C is missing. So, to convert it into standard form multiply the term by $(C + C')$ and expand it as $AB'(C + C') = AB'C + AB'C'$

3. Remove repeated product terms if any.

Standard product of sum (POS) form

- **Π Notation**

Standard POS form:

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

Another way of representing the function in canonical POS form is by showing the product of maxterms for which the function value equals 0.

$$f(A,B,C) = M_0 \cdot M_4 \cdot M_6 \cdot M_7$$

Another way of representing the function in canonical POS form is by listing the decimal equivalents of the maxterms for which $f = 0$.

$$f(A,B,C) = \Pi(0,4,6,7)$$

Converting POS Form to standard POS Form

1. Find the missing literals in each sum term if any.

2. If one or more variables are missing in any sum term, expand that term by adding the products of each of the missing variable and its complement.

3. For expanding, apply rule $A + BC = (A + B)(A + C)$. For example, in a three variable POS function consider a term $A + B'$. The third variable C is missing. So, to convert it into standard form add the term (CC') and expand it as $A + B' + CC' = (A + B' + C)(A + B' + C')$.

4. Remove repeated sum terms if any.

Simplifying logic circuits

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

$= ABC + A\bar{B}(A+C) = ABC + A\bar{B}A + A\bar{B}C$

$= ABC + A\bar{B} + A\bar{B}C = AC(B+\bar{B}) + A\bar{B}$

$= AC + A\bar{B} = A(C+\bar{B})$

CSE1003
Digital Logic and Design

Module 2
BOOLEAN ALGEBRA L4

Dr. S. Hemamalini
Professor
School of Electrical Engineering
VIT Chennai

Module 2
BOOLEAN ALGEBRA 8 hrs

Boolean algebra

- Properties of Boolean algebra
- Boolean functions
- Canonical and Standard forms
- Logic gates - Universal gates
- Karnaugh map - Don't care conditions
- Tabulation Method

Minimization of Boolean expressions

- The minimization will result in reduction of
  - the number of gates (resulting from less number of terms)
  - the number of inputs per gate (resulting from less number of variables per term)
- Reduced cost
- Improved efficiency
- Reduced power consumption

Simplification Techniques

- Primary objective - to obtain an expression that has the minimum number of terms.
- Secondary objective - obtaining an expression with the minimum number of literals
- If there is more than one possible solution with the same number of terms, the one having the minimum number of literals is the choice.

Sum-of-Products Boolean Expressions

- A sum-of-products expression contains the sum of different terms, with each term being either a single literal or a product of more than one literal.
- It can be obtained from the truth table directly by considering those input combinations that produce a logic '1' at the output.
- Each such input combination produces a term.
- Different terms are given by the product of the corresponding literals.
- The sum of all terms gives the expression.

A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

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

Minimum SOP

- The minimum sum of products (MSOP) of a function, f, is a SOP representation of f that contains the fewest number of product terms and fewest number of literals of any SOP representation of f.
- f= (xyz +x'yz+ xy'z+ .....) is called sum of products.
- The + is sum operator which is an OR gate.
- The product such as *xy* is an AND gate for the two inputs x and y.
- A minimum SOP expression can be implemented with fewer logic gates than a standard expression.

Minimum product of sums (MPOS)

- The minimum product of sums (MPOS) of a function, *f*, is a POS representation of *f* that contains the fewest number of sum terms and the fewest number of literals of any POS representation of *f*.
- The zeros are considered exactly the same as ones in the case of sum of product (SOP)

Simplification using Boolean Algebra

- **Algebraic method**
- **Karnaugh Map**
- **Tabulation Method**

- **Algebraic method**
  - Put the given expression in the SOP form repeatedly using DeMorgan's theorems and multiplication of terms.
  - Check the product terms for common variables and perform factoring wherever possible. This mostly results in elimination of one or more terms.
  - No guarantee that the reduced expression will be a minimal.

Karnaugh Map (K-Map)

- Used to simplify a Boolean expression to their minimal form.
- K-Map method is a pictorial arrangement of the truth table which allows minimal literals to express the function algebraically.
- K-Map method can be applied to functions up to 6 variables only.
- For more than 6 variables, Quine- McClusky method is used.
- K-Map is an array of cells in which each cell represents a binary value of the input variables.
- The cells are arranged in such a way that simplification of a given expression is based on properly grouping the cells.
- K-Map for an n-variable Boolean function is a rectangular diagram in which the 2^n minterms of the given function are arranged in 2^n squares in such an order that the minterms in the adjacent squares differ only by one variable, which is complemented in one square and uncomplemented in the other.

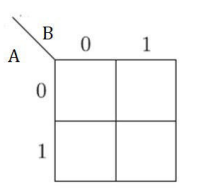
Karnaugh Map

- Is a graphical or pictorial representation of a truth table.
- A ***minterm*** of *n* variables is a product of *n* literals in which each variable appears exactly once in either true or complemented form, but not both.
- A **maxterm** of *n* variables is a sum of *n* literals in which each variable appears exactly once in either true or complemented form, but not both.

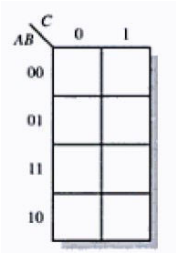
Row No.	A B C	Minterms	Maxterms
0	0 0 0	$A'B'C' = m_0$	$A + B + C = M_0$
1	0 0 1	$A'B'C = m_1$	$A + B + C' = M_1$
2	0 1 0	$A'BC' = m_2$	$A + B' + C = M_2$
3	0 1 1	$A'BC = m_3$	$A + B' + C' = M_3$
4	1 0 0	$AB'C' = m_4$	$A' + B + C = M_4$
5	1 0 1	$AB'C = m_5$	$A' + B + C' = M_5$
6	1 1 0	$ABC' = m_6$	$A' + B' + C = M_6$
7	1 1 1	$ABC = m_7$	$A' + B' + C' = M_7$

- If m_i is a minterm of *f*, then place a 1 in cell i of the K-map.
- If M_i is a maxterm of *f*, then place a 0 in cell i.

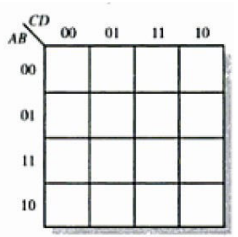
General structure of K-maps



(a) Two variable K-map



(b) Three variable K-Map

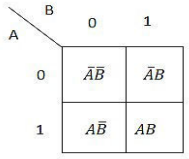


(c) Four variable K-map

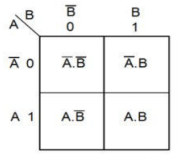
K-Map Procedure

- Prepare a blank map with 2^n cells (n- no. of variables) labelled with the variables appropriately.
- Enter a '1' in each cell (or square) of those minterms for which F=1.
- Enter a '0' in each of the remaining cells.
- Identify the possible pairs of adjacent 1's (horizontally or vertically) and group them together, so that maximum 1's can be grouped into pairs.
  - Bigger groups with eight 1's called octets
  - Bigger groups with four 1's called quads
  - Leave the isolated 1's
  - Any 1 can be reused in this process and the groups can overlap.
- Write down the terms related with each group of 1's as well as each isolated 1, by multiplying the variables and ordinate of that group or isolated 1 and ignoring the variable which appear in both the complemented and uncomplemented forms (A & A')
- Sum up all these terms to get the minimal expression of the function.

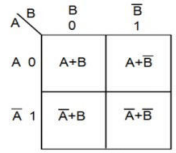
K-Map for 2 variables



SOP form

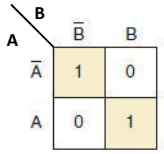


POS form



Two variable K-map f(A,B)=Σm(0,1,3)=A'B'+A'B+AB

A	B	X
0	0	1 → $\bar{A}\bar{B}$
0	1	0
1	0	0
1	1	1 → AB



$x = \bar{A}\bar{B} + AB$

K-Map for 3 variables

SOP

AB \ C	$\overline{C}$ 0	C 1
$\overline{A}\overline{B}$ 00	$\overline{A}\overline{B}\overline{C}$ 0	$\overline{A}\overline{B}C$ 1
$\overline{A}B$ 01	$\overline{A}B\overline{C}$ 2	$\overline{A}BC$ 3
AB 11	$AB\overline{C}$ 6	ABC 7
$A\overline{B}$ 10	$A\overline{B}\overline{C}$ 4	$A\overline{B}C$ 5

A	B	C	X	
0	0	0	1	$\rightarrow \overline{A}\overline{B}\overline{C}$
0	0	1	1	$\rightarrow \overline{A}\overline{B}C$
0	1	0	1	$\rightarrow \overline{A}B\overline{C}$
0	1	1	0	
1	0	0	0	
1	0	1	0	
1	1	0	1	$\rightarrow AB\overline{C}$
1	1	1	0	

$$X = \overline{A}\overline{B}\overline{C} + \overline{A}B\overline{C} + \overline{A}BC + AB\overline{C}$$

AB \ C	$\overline{C}$	C
$\overline{A}\overline{B}$	1	1
$\overline{A}B$	1	0
AB	1	0
$A\overline{B}$	0	0

K-Map for 4 variables

A	B	C	D	X	
0	0	0	0	0	
0	0	0	1	1	$\rightarrow \overline{A}\overline{B}\overline{C}D$
0	0	1	0	0	
0	0	1	1	0	
0	1	0	0	0	
0	1	0	1	1	$\rightarrow \overline{A}B\overline{C}D$
0	1	1	0	0	
0	1	1	1	0	
1	0	0	0	0	
1	0	0	1	0	
1	0	1	0	0	
1	0	1	1	0	
1	1	0	0	0	
1	1	0	1	1	$\rightarrow AB\overline{C}D$
1	1	1	0	0	
1	1	1	1	1	$\rightarrow ABCD$

$$X = \overline{A}\overline{B}\overline{C}D + \overline{A}B\overline{C}D + AB\overline{C}D + ABCD$$

AB \ CD	$\overline{C}\overline{D}$	$\overline{C}D$	CD	$C\overline{D}$
$\overline{A}\overline{B}$	0	1	0	0
$\overline{A}B$	0	1	0	0
AB	0	1	1	0
$A\overline{B}$	0	0	0	0

K-Map for 4 variables

AB \ CD	00	01	11	10
00	**A'B'C'D'**	A'B'C'D	A'B'CD	A'B'CD'
01	A'BC'D'	A'BC'D	A'BCD	A'BCD'
11	ABC'D'	ABC'D	ABCD	ABCD'
10	AB'C'D'	AB'C'D	AB'CD	AB'CD'

AB \ CD	00	01	11	10
00	m_0	m_1	m_2	m_3
01	m_4	m_5	m_6	m_7
11	m_{12}	m_{13}	m_{14}	m_{15}
10	m_8	m_9	m_{10}	m_{11}

Simplification Guidelines for K-maps

- Always combine as many cells in a group as possible. This will result in the fewest number of literals in the term that represents the group.
- Make as few groupings as possible to cover all minterms. This will result in the fewest product terms.
- Always begin with the largest group, which means if you can find eight members group is better than two four groups and one four group is better than pair of two-group.

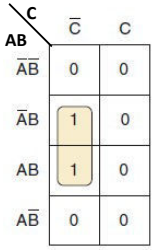
Adjacency

- Cell adjacency
  - Only one variable changes from complemented to uncomplemented form or vice-versa moving from one square to the other in the rows or columns.
- Wrap around adjacency
  - The cells in the top row are adjacent to the corresponding cells in the bottom row and the cells in the outer left column are adjacent to the corresponding cells in the outer right column.

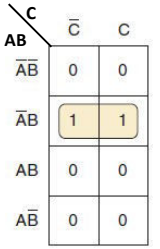
K-map method for simplifying a Boolean expression – step by step procedure

1. Construct the K-map as discussed. Enter 1 in those cells corresponding to the minterms for which function value is 1. Place 0's in other cells.
2. Form the groups of possible 1s as pair, quad and octet. There can be overlapping of groups if they include common cells. While doing this make sure that there are minimum number of groups.
3. Encircle the cells which contain 1s and are not adjacent to any other cell. These are known as isolated minterms and they appear in the expression in same form.
4. Avoid any redundant group.
5. Write the Boolean term for each group and obtain the minimized expression by summing product terms of all the groups.

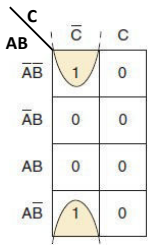
Grouping Cells (Pair)



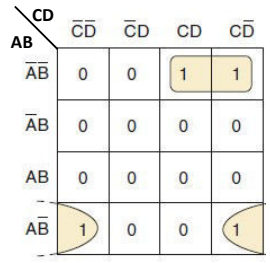
$X = \bar{A}B\bar{C} + AB\bar{C} = B\bar{C}$



$X = \bar{A}B\bar{C} + \bar{A}BC = \bar{A}B$



$X = \overline{ABC} + A\bar{B}\bar{C} = \bar{B}\bar{C}$

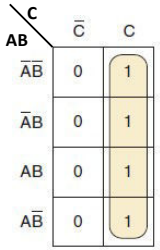


$X = \bar{A}\bar{B}CD + \bar{A}B\bar{C}D + A\bar{B}\bar{C}D + A\bar{B}C\bar{D} = \bar{A}BC + A\bar{B}\bar{D}$

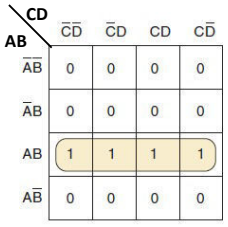
Looping a pair of adjacent 1s in a K map eliminates the variable that appears in complemented and uncomplemented form.

Groups of Four (Quads)

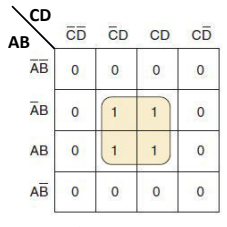
A K map may contain a group of four 1s that are adjacent to each other. This group is called a *quad.*



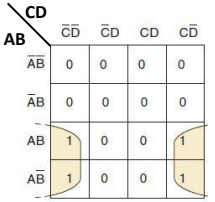
$X = C$



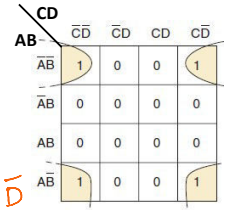
$X = AB$



$X = B\bar{D}$



$X = A\bar{D}$

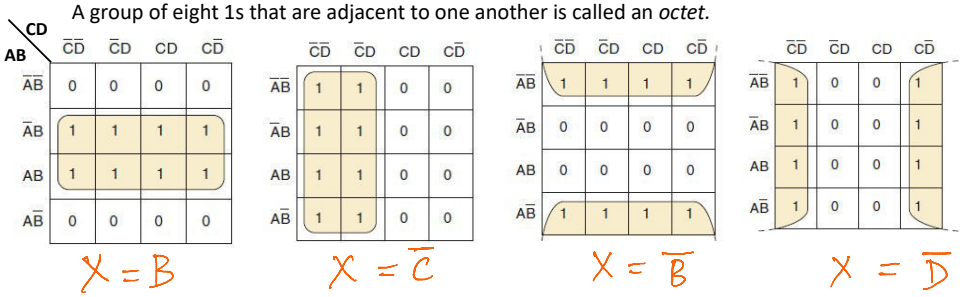


$X = \bar{B}\bar{D}$

Looping a quad of adjacent 1s eliminates the two variables that appear in both complemented and uncomplemented form.

Groups of Eight (Octets)

A group of eight 1s that are adjacent to one another is called an *octet*.



Looping an octet of adjacent 1s eliminates the three variables that appear in both complemented and uncomplemented form.

When a variable appears in both complemented and uncomplemented form within a loop, that variable is eliminated from the expression.
Variables that are the same for all squares of the loop must appear in the final expression.

Module 2
BOOLEAN ALGEBRA 8 hrs

Boolean algebra

- Properties of Boolean algebra
- Boolean functions
- Canonical and Standard forms
- Logic gates - Universal gates
- Karnaugh map - Don't care conditions
- Tabulation Method

CSE1003
Digital Logic and Design

Module 2
BOOLEAN ALGEBRA L5

Dr. S. Hemamalini
Professor
School of Electrical Engineering
VIT Chennai

K-map method for simplifying a Boolean expression – step by step procedure

1. Construct the K-map as discussed. Enter 1 in those cells corresponding to the minterms for which function value is 1. Place 0's in other cells.

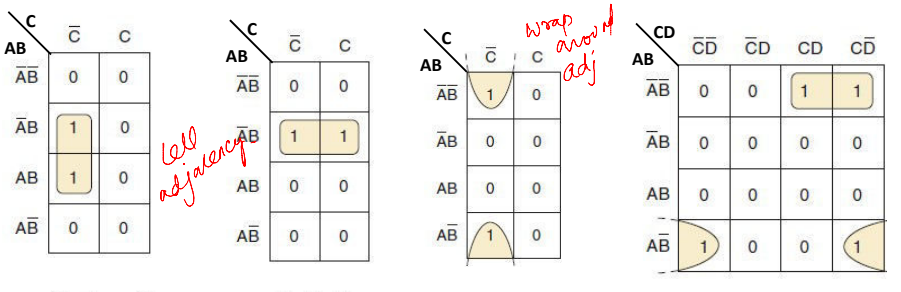
2. Form the groups of possible 1s as pair, quad and octet. There can be overlapping of groups if they include common cells. While doing this make sure that there are minimum number of groups.

3. Encircle the cells which contain 1s and are not adjacent to any other cell. These are known as isolated minterms and they appear in the expression in same form.

4. Avoid any redundant group.

5. Write the Boolean term for each group and obtain the minimized expression by summing product terms of all the groups.

Grouping Cells (Pair)



$X = \bar{A}B\bar{C} + AB\bar{C} = B\bar{C}$

$X = \bar{A}B\bar{C} + \bar{A}BC = \bar{A}B$

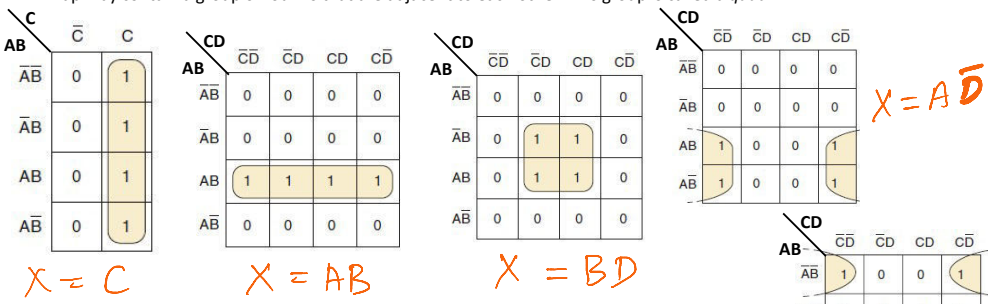
$X = \bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C} = \bar{B}\bar{C}$

$X = \bar{A}\bar{B}CD + \bar{A}B\bar{C}D + A\bar{B}\bar{C}D + AB\bar{C}D = \bar{A}BC + A\bar{B}D$

Looping a pair of adjacent 1s in a K map eliminates the variable that appears in complemented and uncomplemented form.

Groups of Four (Quads)

A K map may contain a group of four 1s that are adjacent to each other. This group is called a *quad.*



$X = C$

$X = AB$

$X = BD$

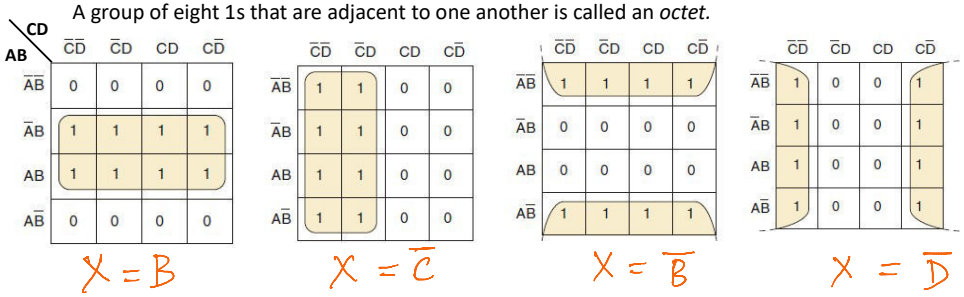
$X = A\bar{D}$

$X = \bar{B}\bar{D}$

Looping a quad of adjacent 1s eliminates the two variables that appear in both complemented and uncomplemented form.

Groups of Eight (Octets)

A group of eight 1s that are adjacent to one another is called an *octet.*



$X = B$

$X = \bar{C}$

$X = \bar{B}$

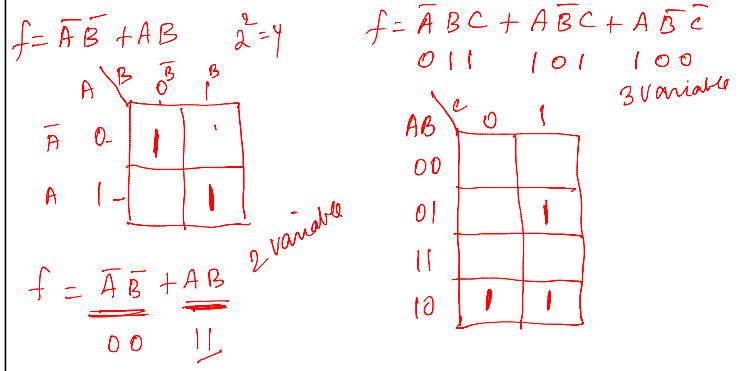
$X = \bar{D}$

Looping an octet of adjacent 1s eliminates the three variables that appear in both complemented and uncomplemented form.

When a variable appears in both complemented and uncomplemented form within a loop, that variable is eliminated from the expression.
Variables that are the same for all squares of the loop must appear in the final expression.

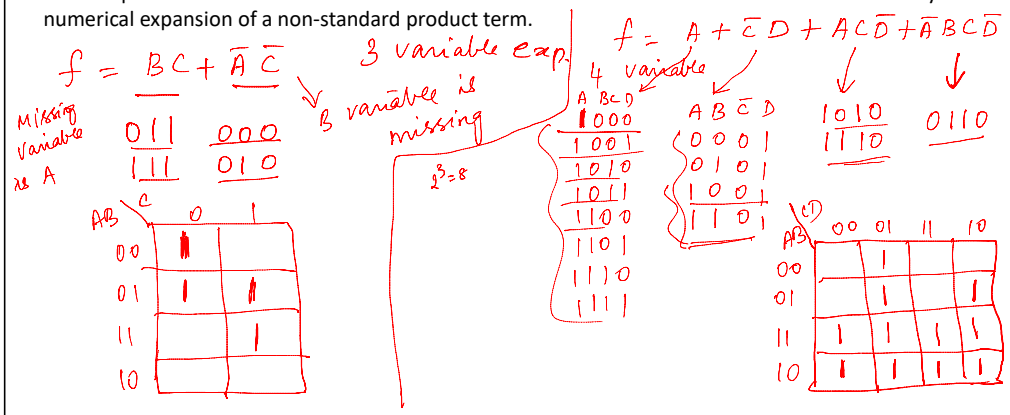
Mapping a standard SOP expression

- Determine the binary value of each product term in the standard SOP expression.
- Place a 1 on the K-map in the cell having the same value as the product term.



Mapping a non-standard SOP expression

If an expression is not in standard form then it must be converted into standard form by numerical expansion of a non-standard product term.

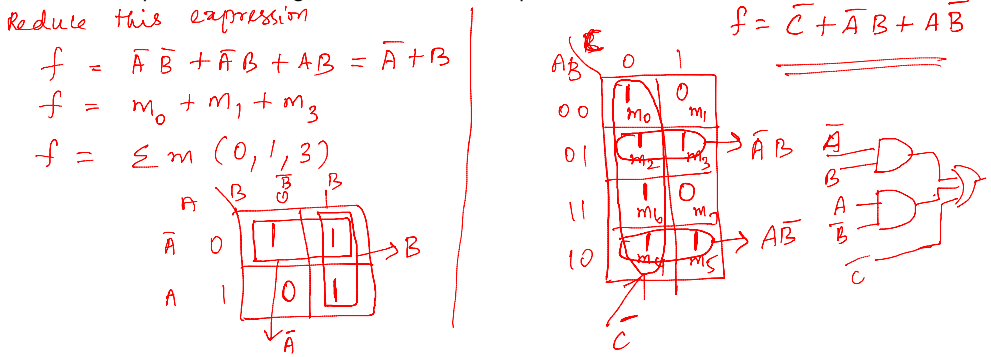


K-map simplification of SOP expressions

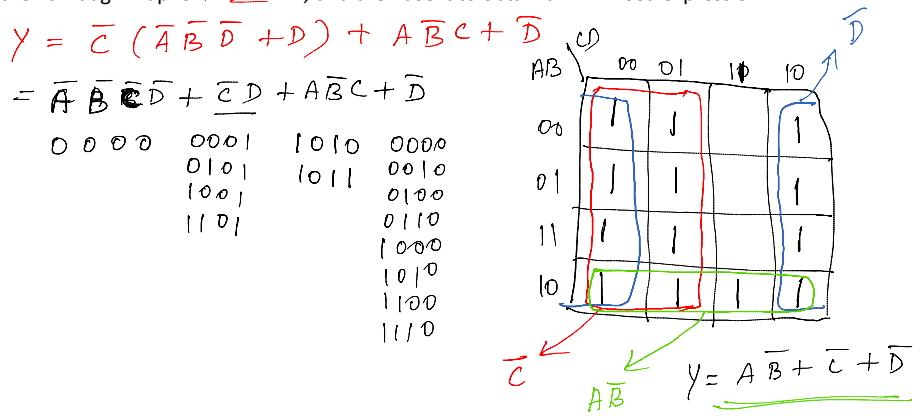
- Grouping the cells that have 1s.
- Determine the minimum product term for each group.
- Sum the product term to get the minimum SOP expression.

Eg. $f = \Sigma m(0,2,3,4,5,6)$
3 Variables

$f = \bar{C} + \bar{A}B + A\bar{B}$



Draw the Karnaugh map for ~~$Y = AB + \bar{A}B$~~, and then use it to obtain a minimised expression.



CSE1003
Digital Logic and Design

Module 2
BOOLEAN ALGEBRA L6

Dr. S. Hemamalini
Professor
School of Electrical Engineering
VIT Chennai

Module 2
BOOLEAN ALGEBRA 8 hrs

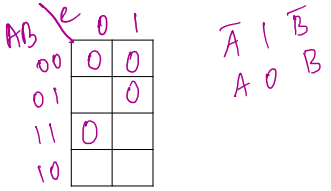
Boolean algebra

- Properties of Boolean algebra
- Boolean functions
- Canonical and Standard forms
- Logic gates - Universal gates
- Karnaugh map - Don't care conditions
- Tabulation Method

K-map For POS Expression

A truth table and corresponding POS K-map

Inputs A	B	C	Output Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

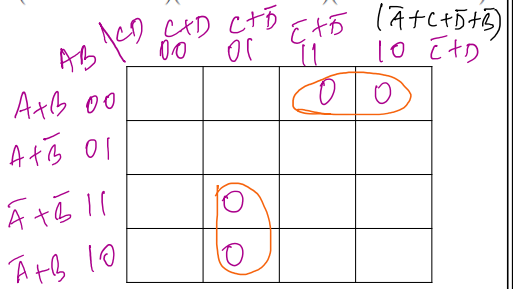


Represent the following Boolean expression by K-map:

$Y(A,B,C,D) = (A+B+\overline{C})(\overline{A}+C+\overline{D})$ — non standard form

$Y(A,B,C,D) = (A+B+\overline{C}+D\overline{D})(\overline{A}+C+\overline{D}+B\overline{B})$

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

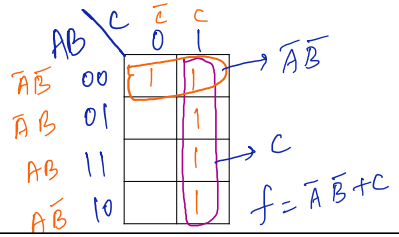


$Y(A,B,C,D) = (\overline{A}+C+\overline{D})(A+B+\overline{C})$

Plotting a Truth Table on K-map

A truth table and corresponding SOP K-map

Inputs A	B	C	Output Y
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1



Represent the following Boolean function by K-map:

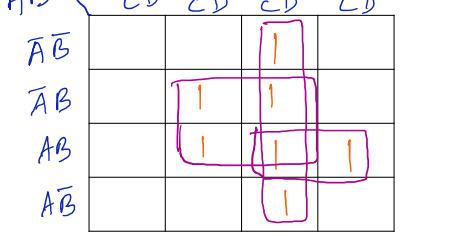
$F(A,B,C,D) = ABC + \overline{B}CD + BD$ → non standard form

$F(A,B,C,D) = ABC(D+\overline{D}) + \overline{B}CD(A+\overline{A}) + BD(A+\overline{A})(C+\overline{C})$

$= ABCD + ABC\overline{D} + A\overline{B}CD + \overline{A}\,\overline{B}CD + ABCD$

$+\overline{A}BCD + AB\overline{C}D + \overline{A}B\overline{C}D$

Standard form →



$f = CD + BD + ABC$

Using K-Map

Minimize the following expression in the POS form

$F(A,B,C,D) = (\overline{A}+\overline{B}+C+D)(\overline{A}+\overline{B}+\overline{C}+D)$

$(\overline{A}+\overline{B}+\overline{C}+\overline{D})(\overline{A}+B+C+D)$

$(A+\overline{B}+\overline{C}+D)(A+\overline{B}+\overline{C}+\overline{D})$

$(A+B+C+D)(\overline{A}+\overline{B}+C+\overline{D})$

1100 $\bar{A}+\bar{B}+C+D$

1110 $\bar{A}+\bar{B}+\bar{C}+D$

1111 $\bar{A}+\bar{B}+\bar{C}+\bar{D}$

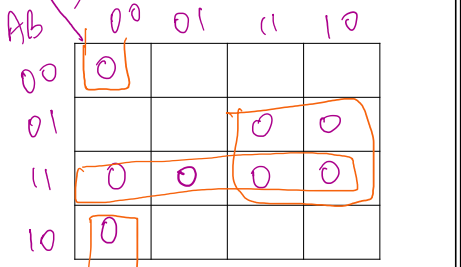
1000 $\bar{A}+B+C+D$

0110 $A+\bar{B}+\bar{C}+D$

0111 $A+\bar{B}+\bar{C}+\bar{D}$

0000 $A+B+C+D$

1101



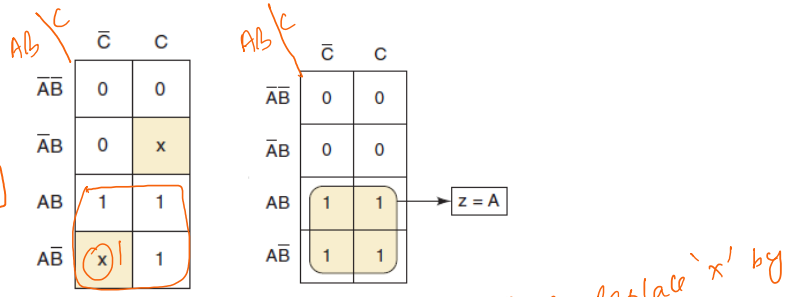
$F(A,B,C,D) = (B+C+D)(\overline{B}+\overline{C})(\overline{A}+\overline{B})$

Don't-Care Conditions

- Some logic circuits can be designed so that there are certain input conditions for which there are no specified output levels, usually because these input conditions will never occur.
- There will be certain combinations of input levels where we "don't care" whether the output is HIGH or LOW.
- A circuit designer is free to make the output for any don't-care condition either a 0 or a 1 to produce the simplest output expression.

A	B	C	z
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	x
1	0	0	x
1	0	1	1
1	1	0	1
1	1	1	1

"don't care"



SOP – Replace 'x' by 1 POS – Replace 'x' by 0

K-Map

Advantages:

- K mapping is a more orderly process with well-defined steps compared with the trial-and-error process sometimes used in algebraic simplification.
- K mapping usually requires fewer steps, especially for expressions containing many terms, and it always produces a minimum expression.

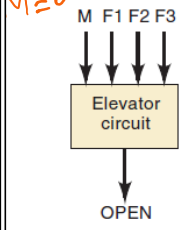
Limitations:

- K-maps are not suitable when the number of variables involved exceed four.
- It may be used with difficulty up to five and six variable systems. But, beyond 'six variables' K-maps cannot be physically visualized.
- K-map simplification is a manual technique and simplification process is heavily dependent on the abilities of the designer. It cannot be programmed.

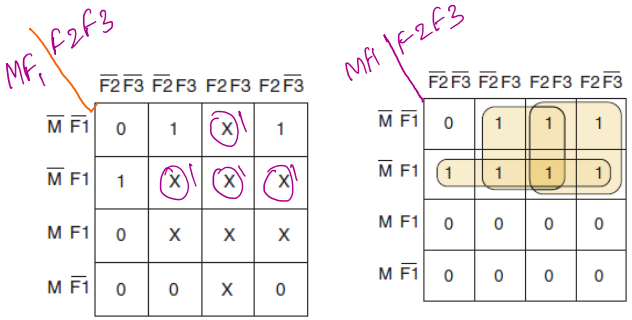
Example for Don't-Care Conditions

Design a logic circuit that controls an elevator door in a three-storey building. The circuit in Figure has four inputs. *M* is a logic signal that indicates when the elevator is moving (*M* = 1) or stopped (*M* = 0). *F1*, *F2*, and *F3* are floor indicator signals that are normally LOW, and they go HIGH only when the elevator is positioned at the level of that particular floor. For example, when the elevator is lined up level with the second floor, *F2*= 1 and *F1*= *F3*= 0. The circuit output is the *OPEN* signal, which is normally LOW and will go HIGH when the elevator door is to be opened.

M = 1 elevator Moving
M = 0 " stopped



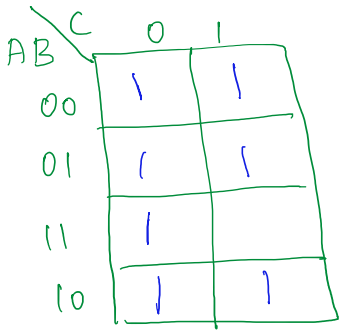
M	F1	F2	F3	OPEN
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	X
0	1	0	0	1
0	1	0	1	X
0	1	1	0	X
0	1	1	1	X
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	X
1	1	0	0	0
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X



OPEN = M̄ (F1 + F2 + F3)

Map the following expression on a K-Map.

$F = \bar{A} + A\bar{B} + AB\bar{C}$



I term = $\bar{A}$ → B C are not given. They may take values as 00, 01, 10, 11

For $\bar{A}$ → possible combinations for which F can be 1

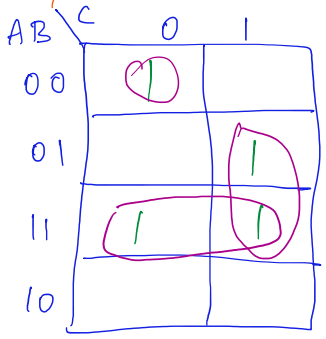
000
001
010
011

} place a '1' in the respective cells

II term $A\bar{B}$ → C can be either '0' or 1

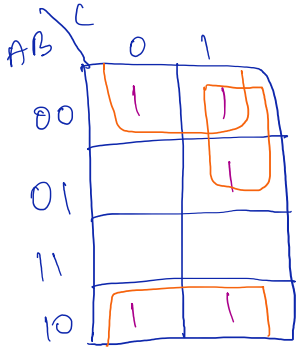
possible combinations : 100 or 101

Determine the product terms for the K-Map in fig & write the resulting minimum SOP expression.



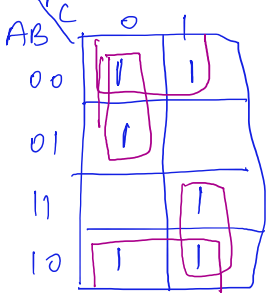
$f = \bar{A}\bar{B}\bar{C} + BC + AB$

Use a K-Map to minimize the following SOP expression. $F = A\bar{B}C + \bar{A}BC + \bar{A}\bar{B}C + \bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C}$



$f = \bar{B} + \bar{A}C$

Determine the product terms for the K-Map in fig & write the resulting minimum SOP expression.



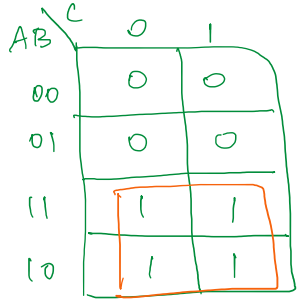
$f = \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + \bar{A}B\bar{C} + ABC + A\bar{B}\bar{C} + A\bar{B}C$

Minimum SOP expression

$f = \bar{B} + AC + \bar{A}\bar{C}$

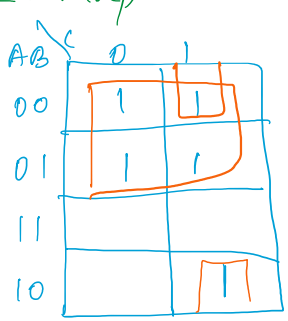
Practice problems

① Obtain the simplified Boolean expression from the truth table in the SOP form.

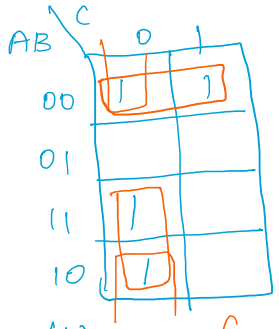


$f = A$

② Obtain the standard SOP expression and simplified SOP expression from the K-Map.



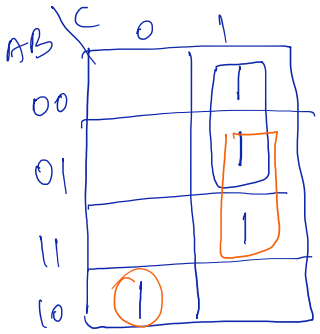
(a) $f = \bar{A} + \bar{B}C$



(b) $f = \bar{A}\bar{B} + A\bar{C} + \bar{B}\bar{C}$

③ Simplify the logic function F in the following cases:

a) $F(A, B, C) = \Sigma m(1, 3, 4, 7)$



$f = A\bar{B}\bar{C} + BC + \bar{A}C$

$f(A, B, C) = \Sigma m(0, 1, 3, 5, 7)$ minimize this by K map.

Ans $f(A, B, C) = C + \bar{A}\bar{B}$

CSE1003
Digital Logic and Design

Module 2
BOOLEAN ALGEBRA L7

Dr. S. Hemamalini
Professor
School of Electrical Engineering
VIT Chennai

Module 2
BOOLEAN ALGEBRA 8 hrs

Boolean algebra

- Properties of Boolean algebra
- Boolean functions
- Canonical and Standard forms
- Logic gates - Universal gates
- Karnaugh map - Don't care conditions
- Tabulation Method

Tabulation Method

- Step-by-step procedure for simplification by the tabular or Quine-McCluskey method.
  - Identify the ***prime implicants***.
  - **Essential prime implicants** are determined by preparing a **prime implicants chart**.
  - The minterms that are not covered by the essential prime implicants, are taken into consideration by selecting some more prime implications to obtain an optimized Boolean expression.

Prime implicants are expressions with least number of literals that represents all the terms given in a truth table.

Tabulation Method

- Also known as the *Quine-McCluskey method.*
- *It is a straightforward, systematic method for producing a minimal function that is less dependent on the designer's ability to recognize patterns than the K-map method.*
- *It is a viable scheme for handling a large number of variables as opposed to the K-map (limited to 5 or 6 variables).*
- *Can also be extended to functions with multiple outputs.*

Determination of Prime Implicants

1. Each minterm of the function is expressed by its binary representation.
2. The minterms are arranged according to increasing index (index is defined as the number of 1s in a minterm). Each set of minterms possessing the same index are separated by lines.
3. Now each of the minterms is compared with the minterms of a higher index. For each pair of terms that can combine, the new terms are formed. If two minterms are differed by only one variable, that variable is replaced by a '-' (dash) to form the new term with one less number of literals. A line is drawn in when all the minterms of one set is compared with all the minterms of a higher index.
4. The same process is repeated for all the groups of minterms. A new list of terms is obtained after the first stage of elimination is completed.
5. At the next stage of elimination two terms from the new list with the '-' of the same position differing by only one variable are compared and again another new term is formed with a less number of literals.
6. The process is to be continued until no new match is possible.
7. All the terms that remain unchecked *i.e.*, where no match is found during the process, are considered to be the prime implicants.

Prime Implicant Chart

1. After obtaining the prime implicants, a chart or table is prepared where rows are represented by the prime implicants and the columns are represented by the minterms of the function.

2. Crosses are placed in each row to show the composition of the minterms that makes the prime implicants.

3. A completed prime implicant table is to be inspected for the columns containing only a single cross. Prime implicants that cover the minterms with a single cross are called the essential prime implicants.

Quine-McCluskey method

Step 2. Perform an exhaustive search between neighboring groups for adjacent minterms and combine them into a column of $(n-1)$-variable implicants, checking off each minterm that is combined. The binary representation of each new implicant contains a dash in the position of the eliminated variable. Repeat for each column, combining $(n-1)$-varible implicants into $(n-2)$-variable implicants, and so on, until no further implicants can be combined. Any term not checked off represents a prime implicant of the function, since it is not covered by a larger implicant. The final result is a list of prime implicants of the switching function.

Quine-McCluskey method

Step 1. List in a column all the minterms of the function to be minimized in their binary representation. Partition them into groups according to the number of 1 bits in their binary representations. This partitioning simplifies identification of logically adjacent minterms since, to be logically adjacent, two minterms must differ in exactly one literal, and therefore the binary representation of one minterm must have either one more or one fewer 1 bit than the other.

Quine-McCluskey method

Step 3. Construct a prime implicant chart that lists minterms along the horizontal and prime implicants along the vertical, with an $\times$ entry placed wherever a certain prime implicant (row) covers a given minterm (column).

Step 4. Select a minimum number of prime implicants that cover all the minterms of the switching function.

Quine-McCluskey method

Minimize the following expression using Q-M method. $f(A,B,C,D) = \sum m(2,4,6,8,9,10,12,13,15)$

Minterms	$ABCD$	
2	0010	
4	0100	Group 1 (a single 1)
8	1000	
6	0110	
9	1001	Group 2 (two 1's)
10	1010	
12	1100	
13	1101	Group 3 (three 1's)
15	1111	Group 4 (four 1's)

	List 1			List 2			List 3		
	Minterm	$ABCD$		Minterms	$ABCD$		Minterms	$ABCD$	
	2	0010	√	2, 6	0-10	PI_2	8, 9, 12, 13	1-0-	PI_1
	4	0100	√	2, 10	-010	PI_3			
	8	1000	√	4, 6	01-0	PI_4			
	6	0110	√	4, 12	-100	PI_5			
	9	1001	√	8, 9	100-	√			
	10	1010	√	8, 10	10-0	PI_6			
	12	1100	√	8, 12	1-00	√			
	13	1101	√	9, 13	1-01	√			
	15	1111	√	12, 13	110-	√			
				13, 15	11-1	PI_7			

	2	4	6	√ 8	√ 9	10	√ 12	√ 13	√ 15
** PI_1				×	⊗		×	×	
PI_2	×		×						
PI_3	×					×			
PI_4		×	×						
PI_5		×					×		
PI_6				×		×			
** PI_7								×	⊗

$f(A, B, C, D) = PI_1 + PI_3 + PI_4 + PI_7$
$= 1\text{-}0\text{-} + \text{-}010 + 01\text{-}0 + 11\text{-}1$
$= A\bar{C} + \bar{B}C\bar{D} + \bar{A}B\bar{D} + ABD$

	√ 2	√ 4	√ 6	√ 10
PI_2	×		×	
* PI_3	×			×
* PI_4		×	×	
PI_5		×		
PI_6				×

Rules for PI chart reduction

Rule 1. A row that *is covered* by another row may be eliminated from the chart. When identical rows are present, all but one of the rows may be eliminated.

Rule 2. A column that *covers* another column may be eliminated. All but one column from a set of identical columns may be eliminated.

	5	10	11	13
PI_2	×			×
PI_3	×			×
PI_4		×	×	
PI_5			×	×
PI_6		×	×	

	√ 5	√ 10
* PI_2	×	
* PI_4		×

Simplify the following function using Quine-McCluskey method.

$$Y(A,B,C,D) = \Sigma m(0,2,3,5,7,8,12,13)$$

STEP 1:

- The minterms of the function are represented in binary form as shown in Table 1.
- The binary representation are grouped into a number of sections in terms of the number of 1's index as shown in Table 2.

Minterms	Binary *ABCD*	No. of 1's
m_0	0000	0
m_2	0010	1
m_3	0011	2
m_5	0101	2
m_7	0111	3
m_8	1000	1
m_{12}	1100	2
m_{13}	1101	3

Minterms Group	Index	Binary *ABCD*
0	0	0000 ✓
2	I	0010 ✓
8		1000 ✓
3	II	0011 ✓
5		0101 ✓
12		1100 ✓
7	III	0111 ✓
13		1101 ✓

STEP 2:

- Compare every term of the lowest group with each term in the adjacent group.
- If two minterms differ in only one variable, that variable should be removed and a dash (–) is placed at the position, thus a new term with one less literal is formed.
- If such a situation occurs, a check mark (V) is placed next to both minterms.
- After all pairs of terms have been considered, a horizontal line is drawn under the last terms as depicted in Table.

Step 3:

- Now repeat step-2 with newly formed groups i.e. combine four minterms of adjacent groups if possibilities exist.
- In this case, dashes (–) exist in same position of two groups and only one position will be different.
- No four minterm group is possible here.

Minterms Group	Index	Binary *ABCD*
0	0	0000 ✓
2	I	0010 ✓
8		1000 ✓
3	II	0011 ✓
5		0101 ✓
12		1100 ✓
7	III	0111 ✓
13		1101 ✓

The combinations of two minterms

Minterms Group	Binary *A*	*B*	*C*	*D*	
0, 2	0	0	–	0	PI
0, 8	–	0	0	0	PI
2, 3	0	0	1	–	PI
8, 12	1	–	0	0	PI
3, 7	0	–	1	1	PI
5, 7	0	1	–	1	PI
5, 13	–	1	0	1	PI
12, 13	1	1	0	–	PI

STEP 4:

- All terms which remain unchecked are the PIs.
- Remove repeated prime implicants as shown in Table.
- Now, construct the prime implicant chart as shown in table.

The combinations of two minterms

Minterms Group	Binary *A*	*B*	*C*	*D*	
0, 2	0	0	–	0	PI
0, 8	–	0	0	0	PI
2, 3	0	0	1	–	PI
8, 12	1	–	0	0	PI
3, 7	0	–	1	1	PI
5, 7	0	1	–	1	PI
5, 13	–	1	0	1	PI
12, 13	1	1	0	–	PI

Prime implicants chart

Prime Implicants	Minterms 0	2	3	5	7	8	12	13
0, 2•	×	×						
0, 8	×					×		
2, 3		×	×					
8, 12•						×	×	
3, 7•			×		×			
5, 7				×	×			
5, 13•				×				×
12, 13							×	×

Step 5:

- Here the function does not have any essential prime implication.
- So choose the prime implicants such that all minterms are covered.

Prime Implicants	Binary Representation	Product Term
0, 2	0 0 – 0	$\overline{A}\,\overline{B}\,\overline{D}$
8, 12	1 – 0 0	$A\overline{C}\,\overline{D}$
3, 7	0 – 1 1	$\overline{A}CD$
5, 13	– 1 0 1	$B\overline{C}D$

Minimized SOP expression

$$f(A,B,C,D) = \overline{A}\,\overline{B}\,\overline{D} + A\overline{C}\,\overline{D} + \overline{A}CD + B\overline{C}D$$

Simplify the following function using Quine-McCluskey method.

$$f(A,B,C,D) = \Sigma m(7,9,12,13,14,15) + d(4,11)$$

Minterms	Binary *ABCD*	No. of 1's
m_7	0111	3
m_9	1001	2
m_{12}	1100	2
m_{13}	1101	3
m_{14}	1110	3
m_{15}	1111	4
m_4	0100	1
m_{11}	1011	3

Minterm Group	Index	Binary *ABCD*
m_4	I	0100 ✓
m_9	II	1001 ✓
m_{12}		1100 ✓
m_7	III	0111 ✓
m_{11}		1011 ✓
m_{13}		1101 ✓
m_{14}		1110 ✓
m_{15}	IV	1111 ✓

Minterm Group	Index	Binary *ABCD*
m_4	I	0100 ✓
m_9	II	1001 ✓
m_{12}		1100 ✓
m_7	III	0111 ✓
m_{11}		1011 ✓
m_{13}		1101 ✓
m_{14}		1110 ✓
m_{15}	IV	1111 ✓

Combination of two minterms

Minterm Group	Binary *A*	*B*	*C*	*D*	
4, 12	–	1	0	0	PI
9, 11	1	0	–	1 ✓	
9, 13	1	–	0	1 ✓	
12, 13	1	1	0	– ✓	
12, 14	1	1	–	0 ✓	
7, 15	–	1	1	1	PI
11, 15	1	–	1	1 ✓	
13, 15	1	1	–	1 ✓	
14, 15	1	1	1	– ✓	

Combination of four minterms

Minterm Group	Binary *A*	*B*	*C*	*D*	
9, 11, 13, 15	1	–	–	1	PI
9, 13, 11, 15	1	–	–	1	PI Eliminated
12, 13, 14, 15	1	1	–	–	PI
12, 14, 13, 15	1	1	–	–	PI Eliminated

Prime implicant chart

Prime Implicants	Binary Representation	Variable Representation
4, 12	– 1 0 0	$B\overline{C}\,\overline{D}$
7, 15	– 1 1 1	BCD
9, 11, 13, 15	1 – – 1	AD
12, 13, 14, 15	1 1 – –	AB

CSE1003 Digital Logic and Design
Module 3
Combinational Circuits I
L1

Dr. S.Hemamalini
Professor
School of Electrical Engineering
VIT Chennai

Contents

4 hrs

- Adder
- Subtractor
- Code converter
- Analyzing a combinational circuit

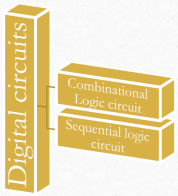
Digital Circuits

Combinational Logic Circuits

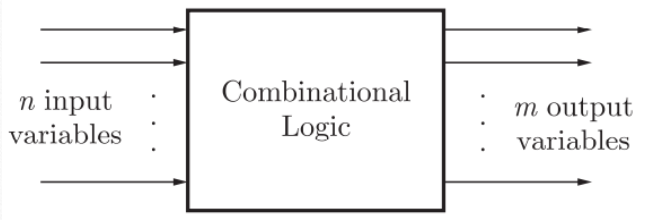
- Output at any instant of time depends only on the inputs present at that instant of time.
- The logic gate is the most basic building block of combinational logic.
- Combinational logic circuits do not have memory elements (storage device).
- It can be designed using gates or available ICs.
- Adder, subtractor, ALU comparators, parity generator and checker, multiplexer, demultiplexer, encoder, and code converters are the examples of combinational logic circuits.

Sequential Logic Circuits

- The output depends upon not only the present but also the past state of inputs.
- Sequential circuits comprise both logic gates and memory elements such as flip-flops.

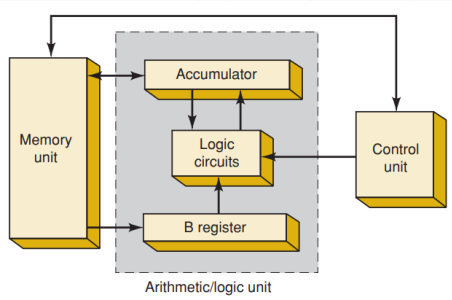


Block diagram of a combinational circuit



ARITHMETIC CIRCUITS
Arithmetic/Logic Unit

- All arithmetic operations take place in the arithmetic/logic unit (ALU) of a computer.
- The arithmetic/logic unit contains at least two flip-flop registers: the B register and the accumulator register.
- It also contains combinational logic, which performs the arithmetic and logic operations on the binary numbers that are stored in the B register and the accumulator.



DESIGN PROCEDURE

Any combinational circuit can be designed by the following steps of design procedure.

1. The problem is stated.
2. Identify the input variables and output functions.
3. The input and output variables are assigned letter symbols.
4. The truth table is prepared that completely defines the relationship between the input variables and output functions.
5. The simplified Boolean expression is obtained by any method of minimization—algebraic method, Karnaugh map method, or tabulation method.
6. A logic diagram is realized from the simplified expression using logic gates.

Design Constraints

(1) minimum number of gates
(2) Minimum number of outputs
(3) minimum propagation time of the signal through a circuit
(4) minimum number of interconnections
(5) limitations of the driving capabilities of each logic gate

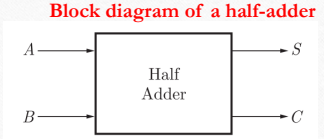
Half-Adder

- The logic circuit that performs the addition of two 1-bit numbers is called as half-adder.
- It is the basic building block for addition of two single bit numbers.
- This circuit has two outputs namely Carry (C) and Sum (S).

Truth table for half-adder

Inputs		Outputs	
A	B	Sum (S)	Carry (C)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Block diagram of a half-adder



ADDERS

- The most basic arithmetic operation is the addition of two binary digits.

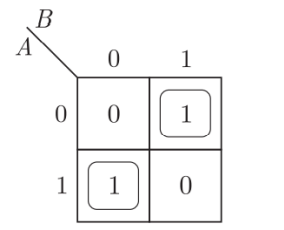
$$0+0=0 \qquad 0+1=1 \qquad 1+0=0 \qquad 1+1=10$$

Carry Sum

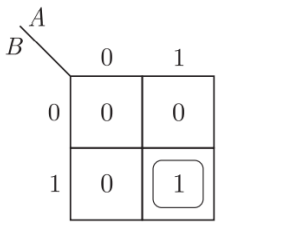
- A combinational circuit that performs the addition of two 1-bit numbers is called as **half-adder**.
- The logic circuit that adds three 1-bit numbers is called as **full-adder**.

Half-Adder

- **K-map Simplification for Carry and Sum**



K-map for sum output

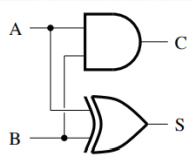


K-map for carry output

$S = \overline{A}B + A\overline{B} = (A \oplus B)$

$C=A.B$

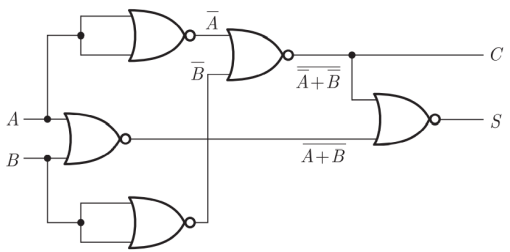
Logic circuit



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

Sum, $S = A\overline{B} + \overline{A}B = A\overline{B} + A\overline{A} + \overline{A}B + B\overline{B}$
$= A(\overline{A} + \overline{B}) + B(\overline{A} + \overline{B})$
$= (A + B)(\overline{A} + \overline{B})$
$= \overline{\overline{(A + B)(\overline{A} + \overline{B})}}$
$= \overline{\overline{A + B} + \overline{\overline{A} + \overline{B}}}$

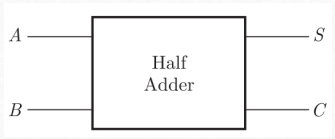
Carry, $C = AB = \overline{\overline{AB}} = \overline{\overline{A} + \overline{B}}$

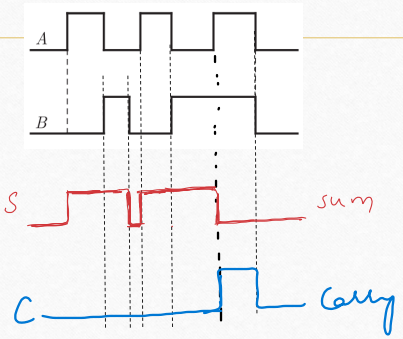


Full-Adder

- A half-adder has two 1-bit inputs and there is no provision to add a carry which could have been generated from lower bit order additions. This limitation of half-adder is overcome in full-adder.
- A full adder circuit is an arithmetic circuit block that can be used to add three bits to produce a sum and a carry output.

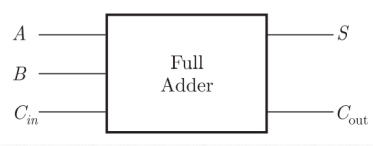
For the half-adder circuit of Figure (a), the inputs applied at A and B are as shown in Figure (b). Plot the corresponding SUM and CARRY outputs for the half-adder.

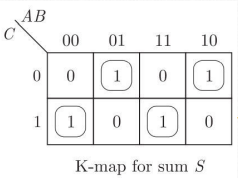




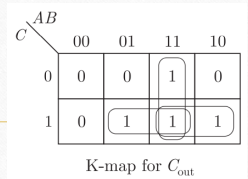
Full-Adder

Inputs			Outputs	
A	B	C_{in}	Sum (S)	Carry 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





K-map for sum S



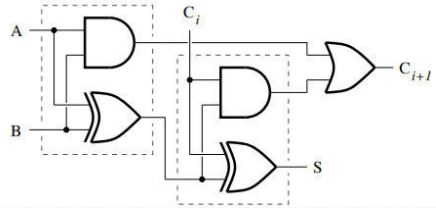
K-map for C_{out}

Sum, $S = \overline{A}\,\overline{B}C_{in} + \overline{A}B\overline{C}_{in} + AB\overline{C}_{in} + A\overline{B}\,\overline{C}_{in}$
$= C_{in}(\overline{A}\,\overline{B} + AB) + \overline{C}_{in}(\overline{A}B + A\overline{B})$
$= C_{in}(A \odot B) + \overline{C}_{in}(A \oplus B)$
$= C_{in}(\overline{A \oplus B}) + \overline{C}_{in}(A \oplus B)$
$S = C_{in} \oplus A \oplus B$

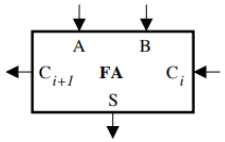
Carry, $C_{out} = AB + AC_{in} + BC_{in}$

Full-Adder

The full-adder circuit can be formed with two half-adders and one OR gate.

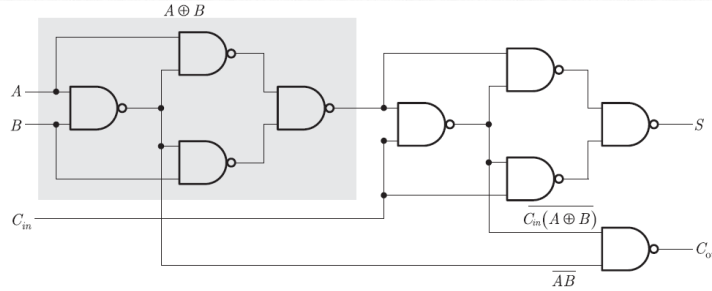


Logic circuit



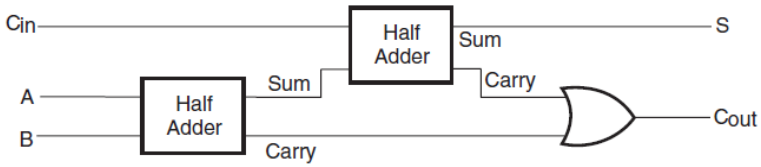
Symbol

Full-Adder using NAND Gates



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

Full-Adder



SUBTRACTORS

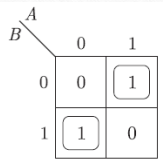
- The logic circuit of subtraction of two 1-bit numbers is called as **half-subtractor.**
- The logic circuit, which performs the subtraction of two 1-bit numbers, taking into account the borrow of the pervious stage, is called as **full-subtractor.**

$0 - 0 = 0$
$1 - 0 = 1$
$1 - 1 = 0$
$0 - 1 = 1$ (with borrow 1)

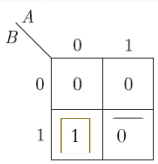
Half-Subtractor

Truth Table of Half-subtractor

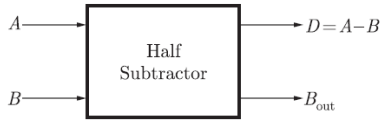
Inputs		Outputs	
A	B	D	B_{out}
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0



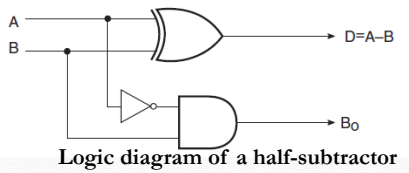
K-map for difference output



K-map for borrow output



Block diagram of a half-subtractor



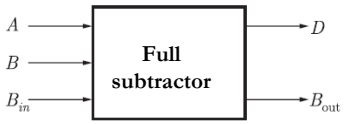
Logic diagram of a half-subtractor

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

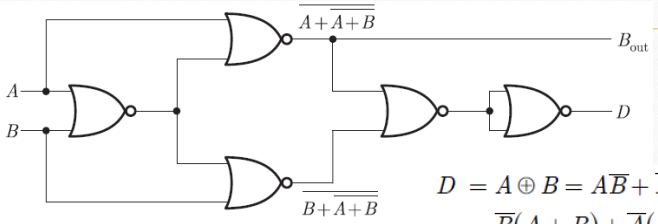
$$B_o = \overline{A}.B$$

Full-Subtractor

- A full subtractor performs subtraction operation on two bits, a minuend and a subtrahend.
- It also takes into consideration whether a '1' has already been borrowed by the previous adjacent lower minuend bit or not.
- Therefore, there are three input bits, namely the two bits to be subtracted and a borrow bit designated as *Bin* .
- There are two outputs, namely the difference output *D* and the borrow output *B*out.
- The BORROW output bit tells whether the minuend bit needs to borrow a '1' from the next possible higher minuend bit.



Half-Subtractor using NOR Gates



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

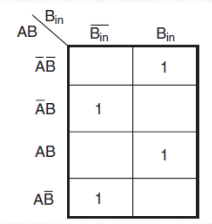
$$D = A \oplus B = A\overline{B} + \overline{A}B = A\overline{B} + B\overline{B} + \overline{A}B + A\overline{A}$$
$$= \overline{B}(A+B) + \overline{A}(A+B)$$
$$D = \overline{B + \overline{A+B}} + \overline{A + \overline{A+B}}$$

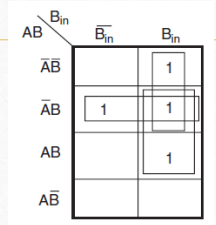
$$B = \overline{A}B = \overline{A}(A+B) = \overline{\overline{A}(A+B)} = \overline{A + (\overline{A+B})}$$

Full-Subtractor

Truth table of Full-Subtractor

Inputs			Outputs	
A	B	B_{in}	D	B_{out}
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





Karnaugh maps for difference and borrow outputs

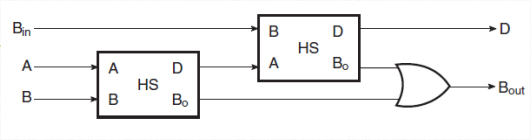
$$D = \overline{A}.\overline{B}.B_{in} + \overline{A}.B.\overline{B}_{in} + A.\overline{B}.\overline{B}_{in} + A.B.B_{in}$$

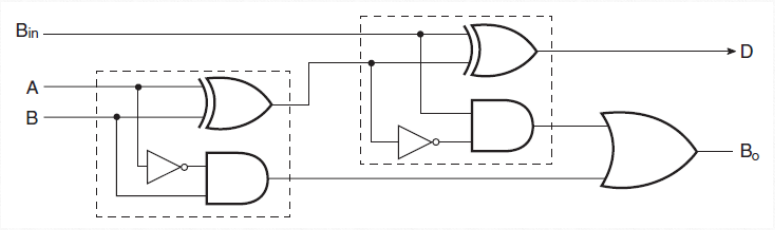
$$B_o = \overline{A}.\overline{B}.B_{in} + \overline{A}.B.\overline{B}_{in} + \overline{A}.B.B_{in} + A.B.B_{in}$$

$$B_o = \overline{A}.B + \overline{A}.B_{in} + B.B_{in}$$

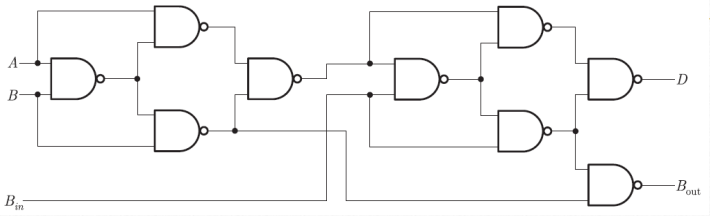
Full-Subtractor

A full subtractor using two half-subtractors





Full Subtractor Using NAND Gates



Difference,

$$D = A \oplus B \oplus B_{in} = \overline{(A \oplus B) \oplus B_{in}}$$

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

where,

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

Borrow,

$$B_{out} = \overline{A}B + B_{in}\overline{(A \oplus B)} = \overline{\overline{\overline{A}B + B_{in}\overline{(A \oplus B)}}}$$

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

$$= \overline{\overline{B(\overline{A} + \overline{B})} \cdot \overline{B_{in}(\overline{B}_{in} + (\overline{A \oplus B}))}}$$

$$B_{out} = \overline{[B \cdot \overline{AB}] \cdot \overline{[B_{in} \cdot (A \oplus B)]}}$$

CSE1003 Digital Logic and Design
Module 3
Combinational Circuits I
L3

DR. S.HEMAMALINI
PROFESSOR
SCHOOL OF ELECTRICAL ENGINEERING
VIT CHENNAI

Contents

4 hrs

Codes

- Codes are used extensively with computers to define alphanumeric characters and other information.

The commonly used binary codes are classified as:

1. Weighted and non-weighted codes
2. Numeric and alphanumeric codes
3. Error detecting and correcting codes
4. Self-complementary codes
5. Unit distance codes (Cyclic codes)
6. Sequential Codes
7. Reflective Codes

Binary-Coded-Decimal Code (8421 Code)

- The four-bit binary numbers from 0000 through 1001 are used.
- The BCD code does not use the numbers 1010, 1011, 1100, 1101, 1110, and 1111.
- If any of the "forbidden" four-bit numbers ever occurs in a machine using the BCD code, it is usually an indication that an error has occurred.

Decimal Numbers	BCD Bit encoding
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Valid BCD codes

Invalid BCD Code

Binary-Coded-Decimal Code (8421 Code)

- If each digit of a decimal number is represented by its binary equivalent, the result is a code called binary-coded decimal.
- The 10 decimal digits 0 through 9 can be represented by their corresponding 4-bit binary numbers.
- It is a weighted code, with 8, 4, 2 and 1 representing the weights of different bits in the four-bit groups, starting from MSB and going towards LSB.
- BCD code is also known as 8421 code or natural binary code.
- It is also sequential. Therefore, it is useful for mathematical operations.

8 1000

12 → 0001 0010

Binary-Coded-Decimal Code (8421 Code)

8	7	4	(decimal)
↓	↓	↓	
1000	0111	0100	(BCD)

9	4	3	(decimal)
↓	↓	↓	
1001	0100	0011	(BCD)

$874_{10} \rightarrow 100001110100_{BCD}$

23.15 (decimal)

0010 0011.0001 0101 (BCD)

Binary-Coded-Decimal Code (8421 Code)

Conversion of BCD code to decimal

- Start at the rightmost bit and break the code into groups of 4 bits.
- Write the decimal digit represented by each 4-bit group.

Convert the following BCD codes to decimal.

10000110 → 8 6 $_{10}$

001101010001 → 351 $_{10}$

1001010001110000 → 9470 $_{10}$

BCD-to-Binary Conversion

- First, write the decimal equivalent of given BCD number and then convert it into binary equivalent.

Convert the following BCD code into its equivalent binary. $[00101001.01110101]_{BCD}$

Given BCD number :	0010	1001	0111	0101
Decimal equivalent :	2	9 .	7	5

$(29.75)_{10} = (11101.11)_2$

Conversion of integer part:

	Quotient	Remainder	
$29 \div 2$	14	1	↑ • LSB
$14 \div 2$	7	0	
$7 \div 2$	3	1	
$3 \div 2$	1	1	
$1 \div 2$	0	1	• MSB

Conversion of fractional part:

Multiplication	Integer part	
$0.75 \times 2 = 1.5$	1	↓ • MSB
$0.5 \times 2 = 1.0$	1	• LSB

Binary-Coded-Decimal Code (8421 Code)

Convert 0110100000111001 (BCD) to its decimal equivalent.

0110 1000 0011 1001
6 8 3 9

Convert the BCD number 011111000001 to its decimal equivalent.

0111 1100 0001
7 ↓ 1

The forbidden code group indicates an error in the BCD number.

Drawbacks of BCD code

- BCD code is less efficient than pure binary-requires more bits.
- An N digit decimal number is represented by 4 x N bits in BCD code.
- The BCD code of 137 is 12 bits and the binary code of 137 is 8 bits; it shows that the BCD code is not efficient as compared to binary.
- The BCD code requires more space and time to transmit the information.
- Arithmetic operations are more complex than they are in pure binary form.

Comparison of BCD and binary

$137_{10} = 10001001_2$ (binary) — 8 bits

$137_{10} = 0001\ 0011\ 0111$ (BCD) — 12 bits

Excess-3 code

- Excess-3 code is non-weighted and sequential code and it is useful for arithmetic operations.
- The excess-3 code for a given decimal number is determined by adding '3' to each decimal digit in the given number and then replacing each digit of the newly found decimal number by its four-bit binary equivalent.
- The XS-3 code has six invalid states 0000, 0001, 0010, 1101, 1110 and 1111.

Decimal	Excess-3
0	0011
1	0100
2	0101
3	0110
4	0111
5	1000
6	1001
7	1010
8	1011
9	1100

Binary codes for decimal digits

Decimal digit	(BCD) 8421	84-2-1	2421	Excess-3
0	0000	0000	0000	0011
1	0001	0111	0001	0100
2	0010	0110	0010	0101
3	0011	0101	0011	0110
4	0100	0100	0100	0111
5	0101	1011	1011	1000
6	0110	1010	1100	1001
7	0111	1001	1101	1010
8	1000	1000	1110	1011
9	1001	1111	1111	1100

Excess-3 code

0 1 0 1	0 1 1 0	• *BCD* code of 56
0 0 1 1	0 0 1 1	• Add 3 to each digit
1 1 1	1 1	• Carry
1 0 0 0	1 0 0 1	• Excess-3 code of (56)

- In excess-3 code, the N digit decimal is represented by 4 x N bits.
- For example, excess-3 code of 12 is 01000101 and there are 8 bits.
- The binary code of 12 is 1100 and there are 4 bits.
- This shows that the excess-3 code is not efficient as compared to binary.
- It requires more space and time to transmit the information.

Gray Codes

- Gray code belongs to a class of code known as minimum change code, in which a number changes by only one bit as it proceeds from one number to the next.
- Hence this code is not useful for arithmetic operations.
- This code finds extensive use for shaft encoders, in some types of analog-to-digital converters, etc.
- Gray code is reflected code.
- The Gray code is not a weighted code.

Decimal numbers	Binary code	Gray code
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Conversion of a Binary Number into Gray Code

(i) the MSB of the Gray code is the same as the MSB of the binary number;

(ii) the second bit next to the MSB of the Gray code equals the Ex-OR of the MSB and second bit of the binary number; it will be 0 if there are same binary bits or it will be 1 for different binary bits;

(iii) the third bit for Gray code equals the exclusive-OR of the second and third bits of the binary number, and similarly all the next lower order bits follow the same mechanism.

$$(74)_{10} = (1001010)_2$$

Binary	1	0	0	1	0	1	0
	⇓	⇓	⇓	⇓	⇓	⇓	⇓
Gray	1	1	0	1	1	1	1

Gray code of 74 = 1101111

Gray-to-Binary Code Conversion

1. The most significant bit (left most bit) of the equivalent binary code is the same as the MSB of the given Gray code.

2. Add the MSB of the binary to the next significant bit of the Gray code, note the sum and ignore the carry.

3. Add the 2nd bit of the binary to the 3rd bit of the Gray; the 3rd bit of the binary to the 4th bit of the Gray code, and so on, each time note the sum and ignore the carry.

4. Continue above step till all Gray bits are used. This sequence of bits is the binary equivalent of the Gray code number.

For example the conversion of Gray code 11011 is shown as below.

Gray	1	1	0	1	1
	⇓	⇓	⇓	⇓	⇓
Binary	1	0	0	1	0

Alphanumeric Codes

- A computer must be capable of handling nonnumeric information like numbers, letters, and special characters. These codes are classified as alphanumeric or character codes.
- An alphanumeric code is a binary code of a group of elements consisting of ten decimal digits, the 26 letters of the alphabet (both in uppercase and lowercase), and a certain number of special symbols such as #, /, &, %, etc.

ASCII code

- American Standard Codes for Information Interchanging (ASCII) is the most widely used alphanumeric code.
- This is basically a 7-bit code and so, it has $2^7 = 128$ possible code groups.
- The ASCII code can be used to encode both the lowercase and uppercase characters of the alphabet (52 symbols) and some special symbols as well, in addition to the 10 decimal digits.
- This code is used to exchange the information between input/output device and computers, and stored into the memory.

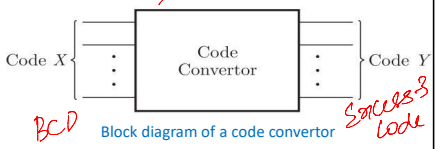
ASCII code

American Standard Code for Information Interchange (ASCII)

$b_4b_3b_2b_1$	$b_7b_6b_5$ = 000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	-	o	DEL

CODE CONVERTERS

- A code converter is a combinational logic circuit which accepts the input information in one binary code, converts it and produces an output into another binary code.
- To convert from one binary code A to binary code B, the input lines must provide the bit combination of elements as specified by A and the output lines must generate the corresponding bit combinations of code B.
- A combinational circuit consisting of logic gates performs this transformation operation.



Block diagram of a code convertor

Common conversions

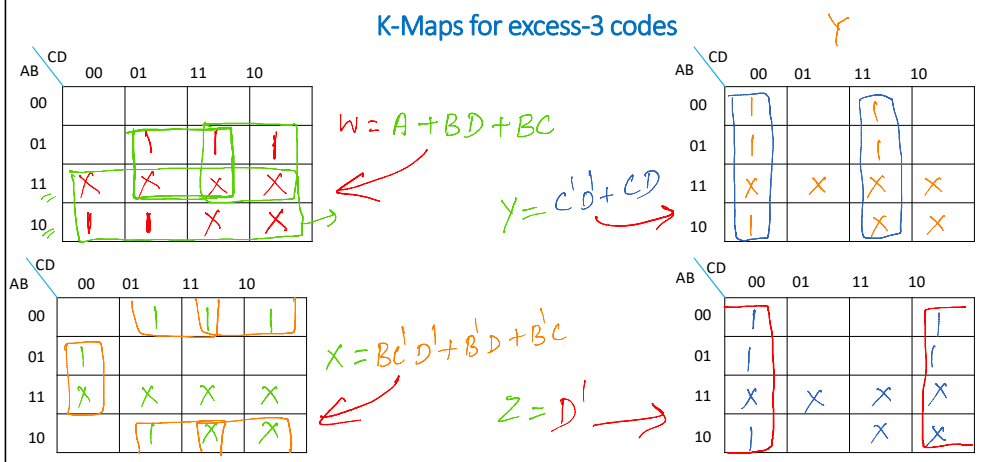
- BCD to 7-segment
- BCD to binary
- Binary to BCD
- Binary to Gray code
- Gray code to binary

BCD-to-Excess-3 Code converter

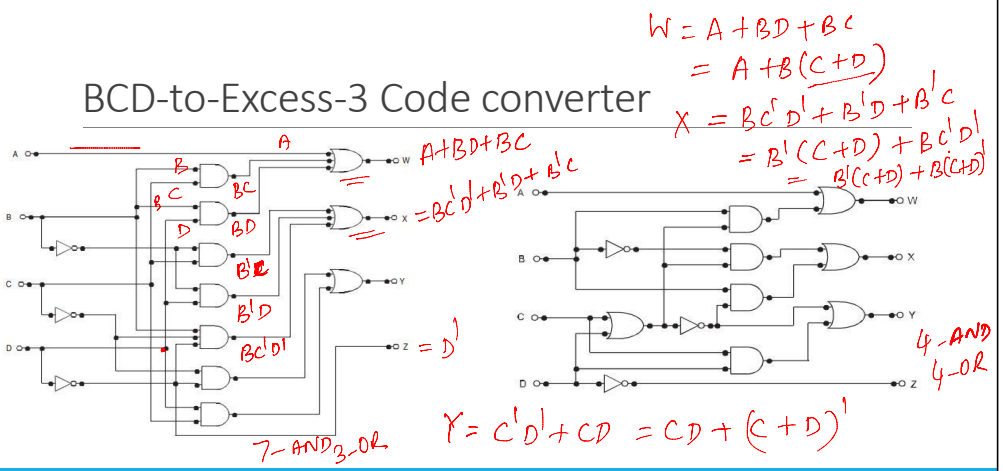
- Excess-3 code is a modified BCD code. It is obtained by adding 3 to each BCD code.
- Note that the input combinations 1010, 1011, 1100, 1101, 1110, and 1111 are invalid in BCD. So they are treated as don't cares.

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

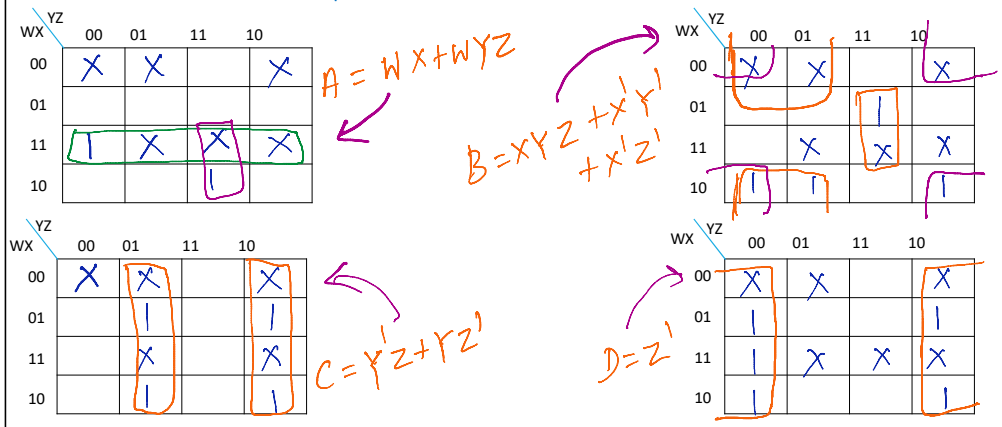
K-Maps for excess-3 codes



BCD-to-Excess-3 Code converter



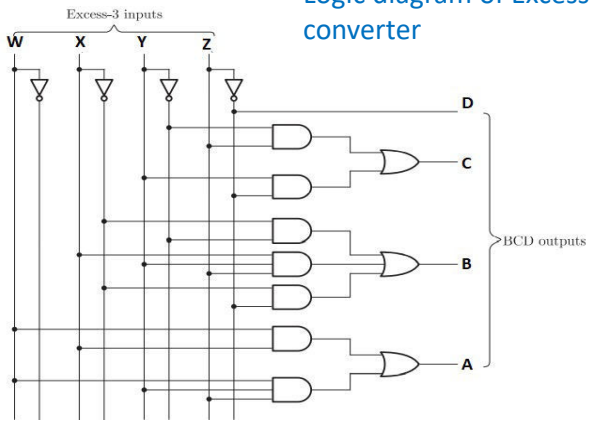
K-Maps for Excess-3-to-BCD Code Converter



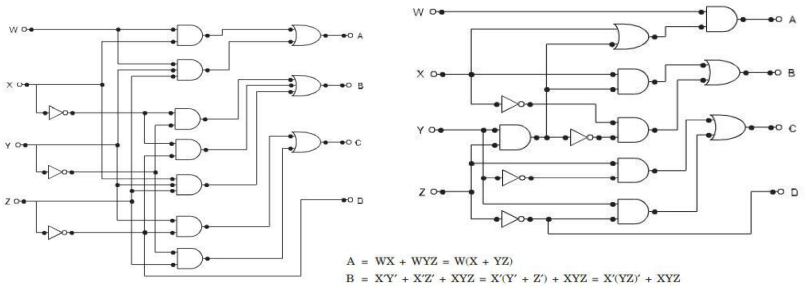
Excess-3-to-BCD Code Converter

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

Logic diagram of Excess-3 to-BCD code converter



Logic diagram of an Excess-3-to-BCD converter



A = WX + WYZ = W(X + YZ)
B = X'Y' + X'Z' + XYZ = X'(Y' + Z') + XYZ = X'(YZ)' + XYZ
C = Y'Z + YZ'
D = Z'

Contents

6 hrs

Binary Parallel Adder - Look ahead carry

Magnitude Comparator

Decoders

Encoders

Multiplexers

Demultiplexers

CO4: Analyze the operation of medium complexity standard combinational circuits like the encoder, decoder multiplexer, demultiplexer.

CSE1003 Digital Logic and Design
Module 4
Combinational Circuits II
L1

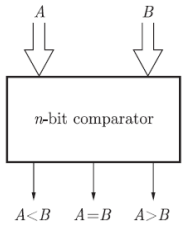
DR. S.HEMAMALINI
PROFESSOR
SCHOOL OF ELECTRICAL ENGINEERING
VIT CHENNAI

Applications – Magnitude Comparators

- Magnitude comparators are useful in control applications where a binary number representing the physical variable being controlled (e.g., position, speed, or temperature) is compared with a reference value. The comparator outputs are used to actuate circuitry to drive the physical variable toward the reference value.
- Comparators are used in central processing units (CPUs) and microcontrollers (MCUs).
- Comparators are also used as process controllers and for Servo motor control.
- Used in password verification and biometric applications.

Magnitude Comparator

- The comparator is a 2n-input, 3-output combinational logic circuit.
- It compares the magnitude of two n-bit numbers and provides the relative result as the output.



$A = (A_{n-1}A_{n-2}\dots A_0)_2$
$B = (B_{n-1}B_{n-2}\dots B_0)_2$

$f_1 = 1$, if $A < B$
$f_2 = 1$, if $A = B$
$f_3 = 1$, if $A > B$

Design of 2-bit Magnitude Comparator

- A comparator used to compare two binary numbers each of two bits is called a 2-bit magnitude comparator.
- It consists of four inputs and three outputs to generate less than, equal to and greater than between two binary numbers.
- Let the two 2-bit binary numbers be $A = A_1A_0$ and $B = B_1B_0$.
- Here each subscript represents one of the digits in the numbers.
- The binary numbers A and B will be equal if all the pairs of significant digits of both numbers are equal, i.e., $A_1 = B_1$ and $A_0 = B_0$.

Truth Table of 2-bit Magnitude Comparator

INPUT				OUTPUT		
A1	A0	B1	B0	A < B	A = B	A > B
0	0	0	0	0	1	0
0	0	0	1	1	0	0
0	0	1	0	1	0	0
0	0	1	1	1	0	0
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	0	1	0
1	0	1	1	1	0	0
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	0	1	0

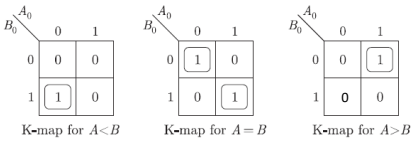
Procedure to compare 2 2bit binary numbers:

1. First compare the two most significant bits (A_1 and B_1).
   If $A_1 > B_1$, then A >B;
   if $A_1 < B_1$, then A< B.
   If $A_1 = B_1$, then the next pair of bits (A_0 and B_0) must be compared.
2. If $A_1 = B_1$ and $A_0 > B_0$, then A> B;
   if $A_1 = B_1$ and $A_0 < B_0$, then A< B.
   Again, if $A_1 = B_1$ and $A_0 = B_0$, then A=B.

1-bit Magnitude Comparator

- The one-bit comparator is a combinational logic circuit with two inputs A and B and three outputs namely A < B, A= B and A > B.
- It compares the two single bit numbers A and B and produces an output that indicates the result of the comparison.
- Let the 1-bit numbers be $A = A_0$ and $B = B_0$.

Design of 1-bit Magnitude Comparator

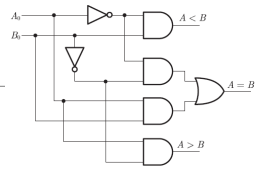


Truth table of a one-bit comparator

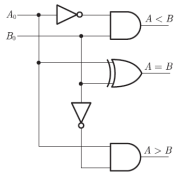
Inputs		Outputs		
A	B	X(A < B)	Y(A = B)	Z(A > B)
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

For $(A < B)$, $X = \overline{A_0}B_0$
For $(A = B)$, $Y = \overline{A_0}\,\overline{B_0} + A_0B_0 = \overline{A_0 \oplus B_0}$
For $(A > B)$, $Z = A_0\overline{B_0}$

Logic diagram of 1-bit comparator



Using Basic Gates

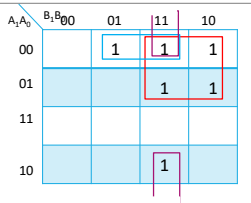


Using AND and EX-NOR gates

Design of 2-bit Magnitude Comparator

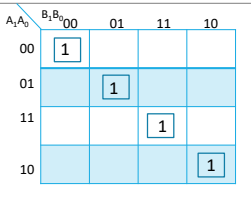
Truth Table of 2-bit Magnitude Comparator

INPUT				OUTPUT		
A1	A0	B1	B0	A < B	A = B	A > B
0	0	0	0	0	1	0
0	0	0	1	1	0	0
0	0	1	0	1	0	0
0	0	1	1	1	0	0
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	0	1	0
1	0	1	1	1	0	0
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	0	1	0

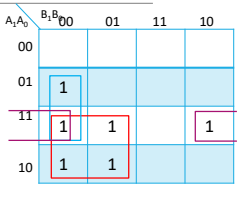


For A<B

$X = \overline{A_1}B_1 + \overline{A_1}B_0\overline{A_0} + \overline{A_0}\,B_1B_0$



For A=B



For A>B

$Z = A_0\overline{B_1}\,\overline{B_0} + A_1A_0\overline{B_0} + A_1\overline{B_1}$

$Y = \overline{A_1}\,\overline{B_1}\,\overline{A_0}\,\overline{B_0} + \overline{A_1}A_0\overline{B_1}B_0 + A_1A_0B_1B_0 + A_1\overline{A_0}B_1\overline{B_0}$
$= \overline{A_1}\,\overline{B_1}(\overline{A_0}\,\overline{B_0} + A_0B_0) + A_1B_1(A_0B_0 + \overline{A_0}\,\overline{B_0})$
$= \overline{A_1}\,\overline{B_1}(A_0 \odot B_0) + A_1B_1(A_0 \odot B_0)$
$= (A_0 \odot B_0)(\overline{A_1}\,\overline{B_1} + A_1B_1)$
$= (A_0 \odot B_0)(A_1 \odot B_1)$

Design of 2-bit Magnitude Comparator

For A<B

$$X = \overline{A_1}B_1 + \overline{A_1}B_0\overline{A_0} + \overline{A_0}\,B_1B_0$$

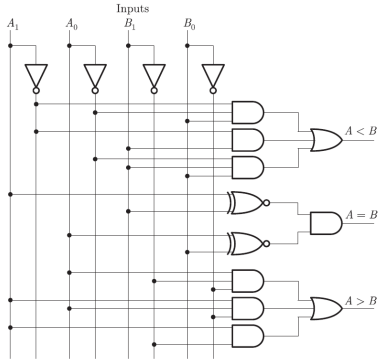
For A=B

$$\begin{aligned} Y &= \overline{A_1}\,\overline{B_1}\,\overline{A_0}\,\overline{B_0} + \overline{A_1}A_0\,\overline{B_1}B_0 + A_1A_0B_1B_0 + A_1\overline{A_0}B_1\overline{B_0} \\ &= \overline{A_1}\,\overline{B_1}(\overline{A_0}\,\overline{B_0} + A_0B_0) + A_1B_1(A_0B_0 + \overline{A_0}\,\overline{B_0}) \\ &= \overline{A_1}\,\overline{B_1}(A_0 \odot B_0) + A_1B_1(A_0 \odot B_0) \\ &= (A_0 \odot B_0)(\overline{A_1}\,\overline{B_1} + A_1B_1) \\ &= (A_0 \odot B_0)(A_1 \odot B_1) \end{aligned}$$

For A>B

$$Z = A_0\,\overline{B_1}\,\overline{B_0} + A_1A_0\,\overline{B_0} + A_1\,\overline{B_1}$$

Logic diagram of 2-bit Magnitude Comparator



Design of 4-bit Magnitude Comparator

A3B3	A2B2	A1B1	A0B0	A>B	A<B	A=B
A3>B3	x	x	x	1	0	0
A3<B3	x	x	x	0	1	0
A3=B3	A2>B2	x	x	1	0	0
A3=B3	A2<B2	x	x	0	1	0
A3=B3	A2=B2	A1>B1	x	1	0	0
A3=B3	A2=B2	A1<B1	x	0	1	0
A3=B3	A2=B2	A1=B1	A0>B0	1	0	0
A3=B3	A2=B2	A1=B1	A0<B0	0	1	0
A3=B3	A2=B2	A1=B1	A0=B0	0	0	1

The output A > B logic expression can be written as

G = A3 $\overline{B3}$ + (A3 Ex-NOR B3) A2 $\overline{B2}$ + (A3 Ex-NOR B3) (A2 Ex-NOR B2) A1 $\overline{B1}$ + (A3 Ex-NOR B3) (A2 Ex-NOR B2) (A1 Ex-NOR B1) A0 $\overline{B0}$

The logical expression for A<B output can be written as

L = $\overline{A3}$ B3 + (A3 Ex-NOR B3) $\overline{A2}$ B2 + (A3 Ex-NOR B3) (A2 Ex-NOR B2) $\overline{A1}$ B1 + (A3 Ex-NOR B3) (A2 Ex-NOR B2) (A1 Ex-NOR B1) $\overline{A0}$ B0

The logical expression for A=B output can be written as

E = (A3 Ex-NOR B3) (A2 Ex-NOR B2) (A1 Ex-NOR B1) (A0 Ex-NOR B0)

Design of 4-bit Magnitude Comparator

- Let the two 4-bit numbers be $A = A_3A_2A_1A_0$ and $B = B_3B_2B_1B_0$
- **Steps used in comparing two 4-bit numbers:**

1. First compare the two most significant bits (A_3 and B_3).
   If $A_3 > B_3$, then $A > B$; if $A_3 < B_3$, then $A < B$.
   If $A_3 = B_3$, the next pair of bits ($A2$ and $B2$) must be compared.
2. If $A_3 = B_3$ and $A_2 > B_2$, then $A > B$;
   if $A_3 = B_3$ and $A_2 < B_2$, then $A < B$
   if $A_3 = B_3$ and $A_2 = B_2$, the next pair of bits (A_1 and B_1) will be compared.
3. If $A_3 = B_3$, $A_2 = B_2$ and $A_1 > B_1$; then $A > B$;
   if $A_3 = B_3$, $A_2 = B_2$ and $A_1 < B_1$, then $A < B$.
   if $A_3 = B_3$, $A_2 = B_2$ and $A_1 = B_1$, compare the LSBs (A_0 and B_0).
4. If $A_3 = B_3$, $A_2 = B_2$, $A_1 = B_1$ and $A_0 > B_0$, then $A > B$;
   if $A_3 = B_3$, $A_2 = B_2$, $A_1 = B_1$ and $A_0 < B_0$, then $A < B$.
5. If $A_3 = B_3$, $A_2 = B_2$, $A_1 = B_1$, $A_0 = B_0$, then $A = B$.

Design of 4-bit Magnitude Comparator

The output A > B logic expression can be written as

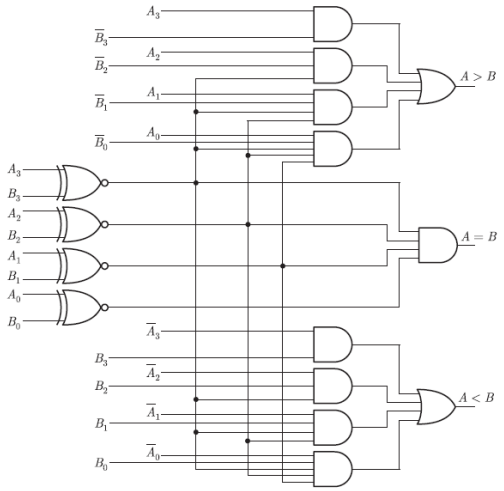
G = A3 $\overline{B3}$ + (A3 Ex-NOR B3) A2 $\overline{B2}$ + (A3 Ex-NOR B3) (A2 Ex-NOR B2) A1 $\overline{B1}$ + (A3 Ex-NOR B3) (A2 Ex-NOR B2) (A1 Ex-NOR B1) A0 $\overline{B0}$

The logical expression for A<B output can be written as

L = $\overline{A3}$ B3 + (A3 Ex-NOR B3) $\overline{A2}$ B2 + (A3 Ex-NOR B3) (A2 Ex-NOR B2) $\overline{A1}$ B1 + (A3 Ex-NOR B3) (A2 Ex-NOR B2) (A1 Ex-NOR B1) $\overline{A0}$ B0

The logical expression for A=B output can be written as

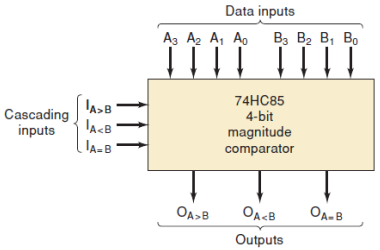
E = (A3 Ex-NOR B3) (A2 Ex-NOR B2) (A1 Ex-NOR B1) (A0 Ex-NOR B0)



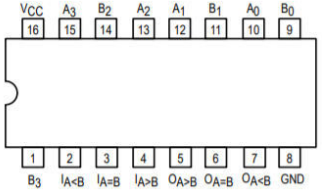
Logic diagram of 4-bit Comparator

MAGNITUDE COMPARATOR

- Magnitude Comparators are available in IC form.
- 7485 is a 4-bit magnitude comparator of the TTL logic family.



Logic Symbol



Pin Configuration

Data Inputs

$$A = (A_3, A_2, A_1, A_0)_2$$
$$B = (B_3, B_2, B_1, B_0)_2$$

Cascade Inputs

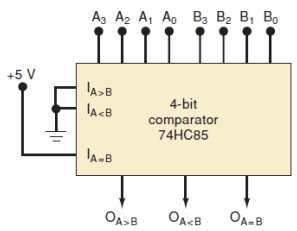
$$C1 \rightarrow A < B$$
$$C2 \rightarrow A = B$$
$$C3 \rightarrow A > B$$

Outputs

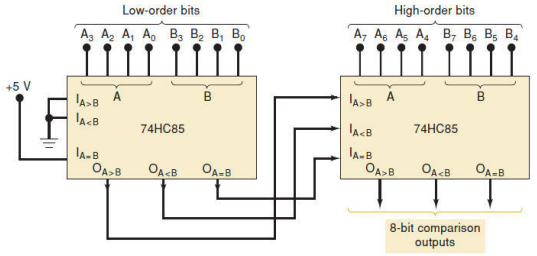
$$O_{A<B}$$
$$O_{A>B}$$
$$O_{A=B}$$

Cascading inputs provide a means for expanding the comparison operation to more than four bits by cascading two or more four-bit comparators.

MAGNITUDE COMPARATOR



(a) 74HC85 wired as a four-bit comparator



(b) Two 74HC85s cascaded to perform an eight-bit comparison

MAGNITUDE COMPARATOR

TRUTH TABLE

COMPARING INPUTS				CASCADING INPUTS			OUTPUTS		
A_3, B_3	A_2, B_2	A_1, B_1	A_0, B_0	$I_{A>B}$	$I_{A<B}$	$I_{A=B}$	$O_{A>B}$	$O_{A<B}$	$O_{A=B}$
$A_3>B_3$	X	X	X	X	X	X	H	L	L
$A_3<B_3$	X	X	X	X	X	X	L	H	L
$A_3=B_3$	$A_2>B_2$	X	X	X	X	X	H	L	L
$A_3=B_3$	$A_2<B_2$	X	X	X	X	X	L	H	L
$A_3=B_3$	$A_2=B_2$	$A_1>B_1$	X	X	X	X	H	L	L
$A_3=B_3$	$A_2=B_2$	$A_1<B_1$	X	X	X	X	L	H	L
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0>B_0$	X	X	X	H	L	L
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0<B_0$	X	X	X	L	H	L
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	H	L	L	H	L	L
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	L	H	L	L	H	L
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	X	X	H	L	L	H
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	L	L	H	H	L	L
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	H	H	L	L	L	L

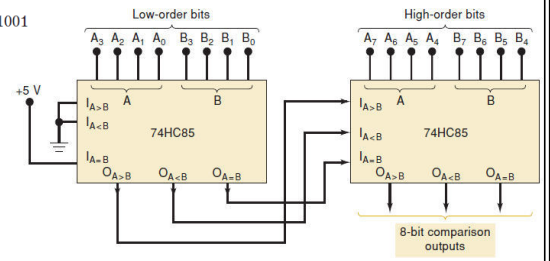
H = HIGH Voltage Level
L = LOW Voltage Level
X = Immaterial

MAGNITUDE COMPARATOR

Describe the operation of the eight-bit comparison arrangement in Figure for the following cases:

(a) $A_7A_6A_5A_4A_3A_2A_1A_0 = 10101111; B_7B_6B_5B_4B_3B_2B_1B_0 = 10110001$

(b) $A_7A_6A_5A_4A_3A_2A_1A_0 = 10101111; B_7B_6B_5B_4B_3B_2B_1B_0 = 10101001$



Solution:

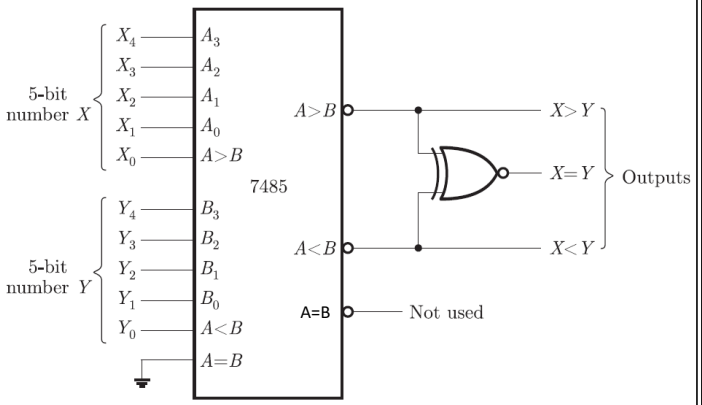
a) $O_{A<B}$=1

b) $O_{A>B}$=1

Design a 5- bit comparator using a single 7485 and one gate.

The two 5-bit numbers to be compared are

$X_4 X_3 X_2 X_1 X_0$ and $Y_4 Y_3 Y_2 Y_1 Y_0$



Contents

6 hrs

- Binary Parallel Adder - Look ahead carry
- Magnitude Comparator
- Decoders
- Encoders
- Multiplexers
- Demultiplexers

- CO4: Analyze the operation of medium complexity standard combinational circuits like the encoder, decoder, multiplexer, demultiplexer.

CSE1003 Digital Logic and Design
Module 4
Combinational Circuits II
L2

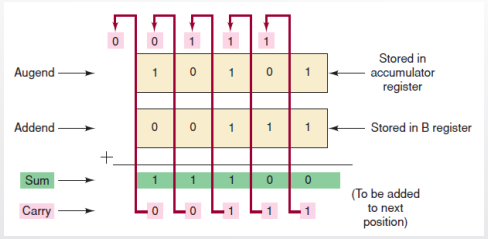
Dr. S.Hemamalini
Professor
School of Electrical Engineering
VIT Chennai

Binary Parallel Adder

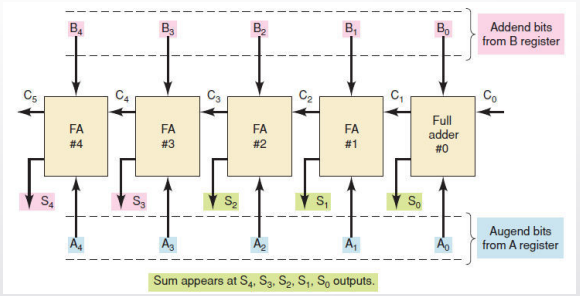
- **A binary parallel adder is a digital circuit that adds two binary numbers in parallel form and produces the arithmetic sum of those numbers in parallel form.**
- **It consists of full adders connected in a chain, with the output carry from each full-adder connected to the input carry of the next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

1. It is used in Digital Processors
2. ALU in computers and in calculators
3. Different IC and microprocessor chips in PCs and laptops
4. In Ripple counters
5. Important tool in DSP (Digital Signal Processing)

Binary Parallel Adder

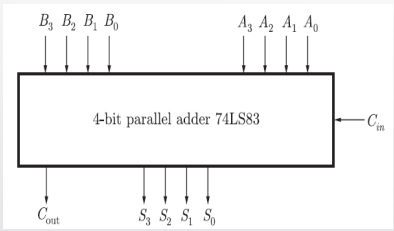


Typical binary addition process

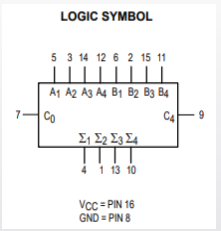


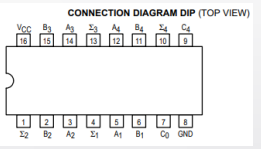
Block diagram of a five-bit parallel adder circuit using full adders.

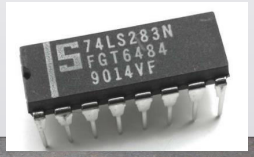
Binary Parallel Adder



Logic symbol of 74LS283



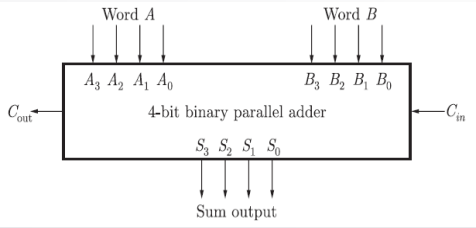




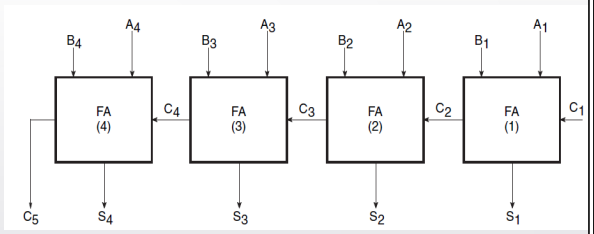
PIN NAMES

A_1–A_4	Operand A Inputs
B_1–B_4	Operand B Inputs
C_0	Carry Input
Σ_1–Σ_4	Sum Outputs
C_4	Carry Output

Binary Parallel Adder

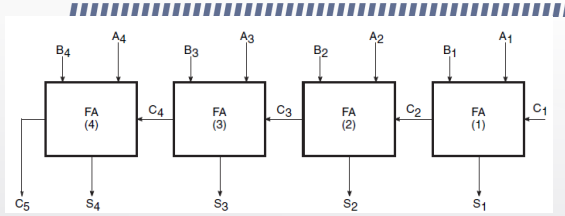


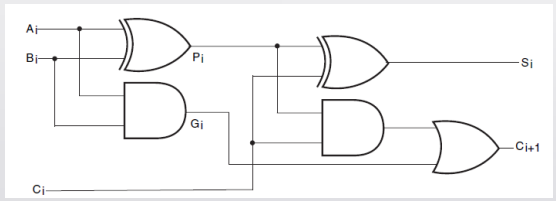
Block diagram of 4-bit binary adder



Logic diagram of a 4-bit binary parallel adder

Binary Parallel Adder



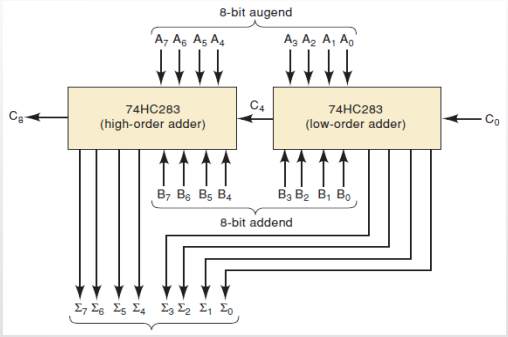


Logic diagram of a full adder

- Parallel adders suffer from propagation delay problem because higher bit additions depend on the carry generated from lower bit addition.
- The **carry propagation delay** for each full adder is the time between the application of the carry-in and the occurrence of the carry out.
- For an *n*-bit parallel adder, the total delay time is equal to nt_p , *where* t_p is the propagation delay of full-adder.
- The parallel adder in which the carry-out of each full adder is the carry-in to the next most significant adder is called a **ripple carry adder**.

The speed of parallel adder depends on the time required for the carries to propagate or ripple through all of the stages of the adder.

Binary Parallel Adder



Block symbol for cascading two 74HC283s

Look ahead carry Adder

- The look-ahead-carry adder speeds up the operation by eliminating the ripple carry delay.
- It examines all the input bits simultaneously and also generates the carry-in bits for all the stages simultaneously.
- It looks ahead and generate the carry for a certain given addition operation.
- The method of speeding up the addition process is based on the two additional functions of the full-adder, called the carry generate and carry propagate functions.

Binary Parallel Adder

- How many inputs does a full adder have? How many outputs?

Ans: 3,2

- Assume the following input levels in Figure.

$A_4A_3A_2A_1A_0$ = 01001; $B_4B_3B_2B_1B_0$ = 00111; C_0 = 0

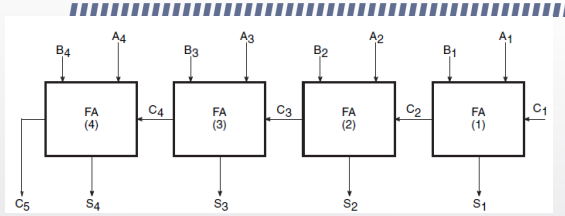
(a) What are the logic levels at the outputs of FA #2?

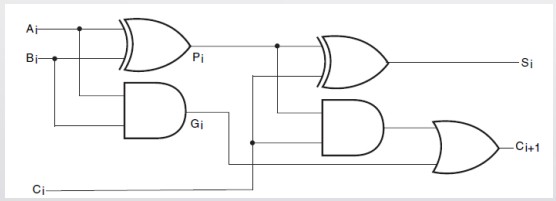
Ans: S_2=0, C_3=1

(b) What is the logic level at the C_5 output?

Ans: C_5=0

Look ahead carry Adder





Logic diagram of a full adder

$$P_i = A_i \oplus B_i$$

$$G_i = A_i.B_i$$

$$S_i = P_i \oplus C_i$$

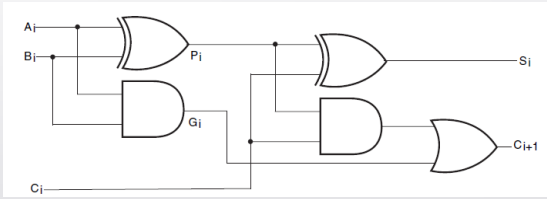
$$C_{i+1} = P_i.C_i + G_i$$

Look ahead carry Adder

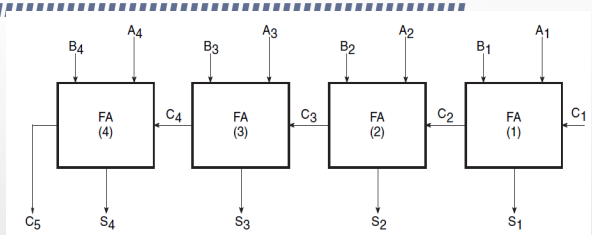
Carry Generation

- Carry is generated only if both the input bits are 1, that is, if both the bits *A* and *B* are 1's, a carry has to be generated in this stage regardless of whether the input carry C_{in} is a 0 or a 1. Otherwise carry will not be generated.

Carry-generation function, $G_i = A_iB_i$

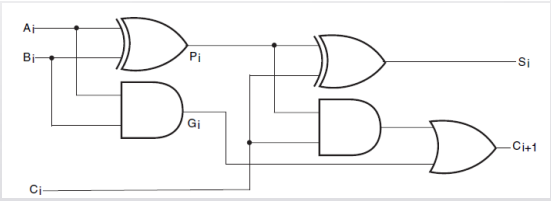


Look ahead carry Adder



CARRY output of each full adder stage in the four-bit binary adder

$$C_2 = G_1 + P_1.C_1$$
$$C_3 = G_2 + P_2.C_2 = G_2 + P_2.(G_1 + P_1.C_1) = G_2 + P_2.G_1 + P_1.P_2.C_1$$
$$C_4 = G_3 + P_3.C_3 = G_3 + P_3.(G_2 + P_2.G_1 + P_1.P_2.C_1)$$
$$C_4 = G_3 + P_3.G_2 + P_3.P_2.G_1 + P_1.P_2.P_3.C_1$$



Look ahead carry Adder

Carry Propagation

- A carry is propagated if any one of the two input bits *A* or *B* is 1.
- If both *A* and *B* are 0, a carry will never be propagated.
- On the other hand, if both *A* and *B* are 1, then it will not propagate the carry but will generate the carry.
- Carry-propagation function

$$P_i = A_i \oplus B_i$$

Carry-generation and carry-propagation

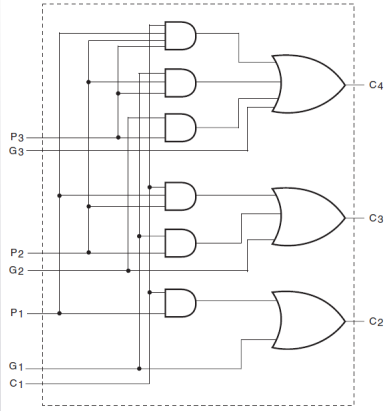
Inputs			Outputs	
A	*B*	C_{in}	Sum	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1 □
1	0	0	1	0
1	0	1	0	1 □
1	1	0	0	1 ○
1	1	1	1	1 ○

□: Indicates carry propagated
○: Indicates carry generated

Look ahead carry Adder

Four-bit full-adder with a look-ahead carry generator

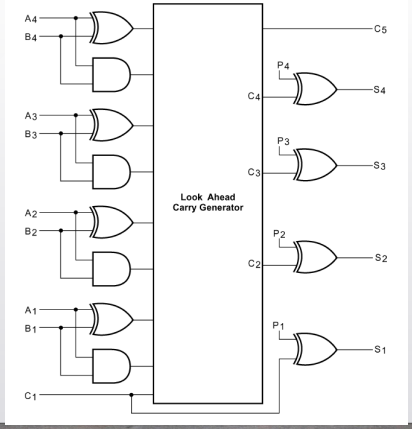
$$C_2 = G_1 + P_1.C_1$$
$$C_3 = G_2 + P_2.C_2 = G_2 + P_2.(G_1 + P_1.C_1) = G_2 + P_2.G_1 + P_1.P_2.C_1$$
$$C_4 = G_3 + P_3.C_3 = G_3 + P_3.(G_2 + P_2.G_1 + P_1.P_2.C_1)$$
$$C_4 = G_3 + P_3.G_2 + P_3.P_2.G_1 + P_1.P_2.P_3.C_1$$



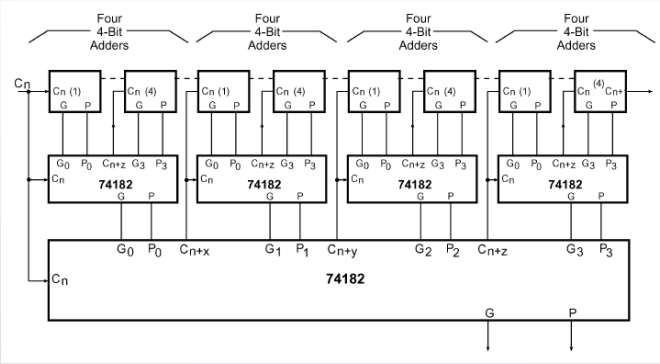
The circuit for look-ahead-carry introduces a delay corresponding to two gate levels.

Look ahead carry Adder

Four-bit full adder with a look-ahead carry generator

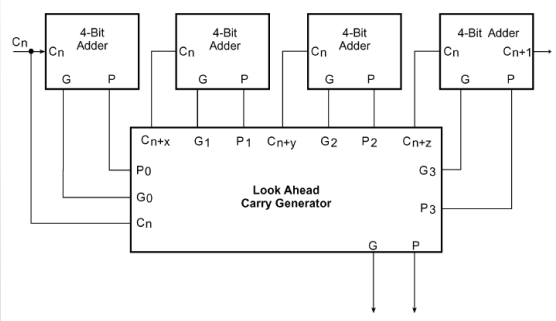


Look ahead carry Adder



Look-ahead carry generation for adding 64-bit numbers

Look ahead carry Adder



IC 74182 interfaced with four four-bit adders

CSE1003 Digital Logic and Design
Module 4
Combinational Circuits II
L2

Dr. S.Hemamalini
Professor
School of Electrical Engineering
VIT Chennai

Contents

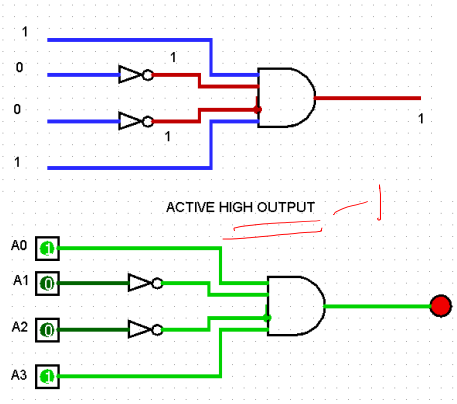
6 hrs

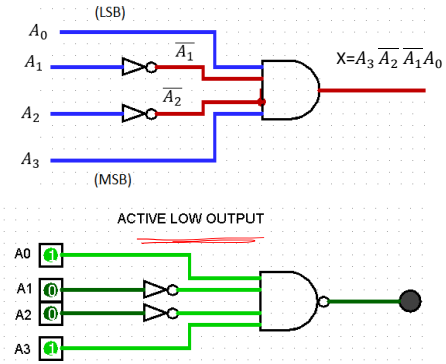
- Binary Parallel Adder - Look ahead carry
- Magnitude Comparator
- **Decoders**
- Encoders
- Multiplexers
- Demultiplexers

- CO4: Analyze the operation of medium complexity standard combinational circuits like the encoder, decoder, multiplexer, demultiplexer.

Decoders

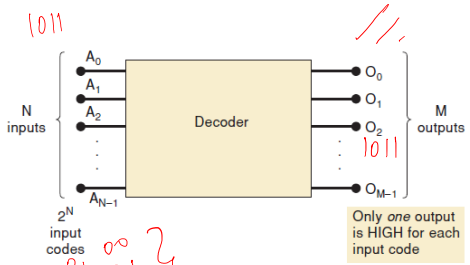
BASIC BINARY DECODER





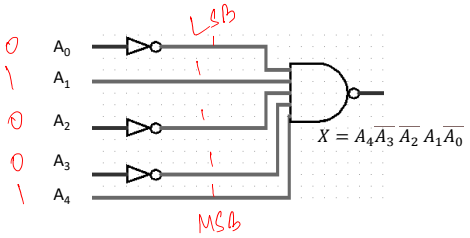
Decoders

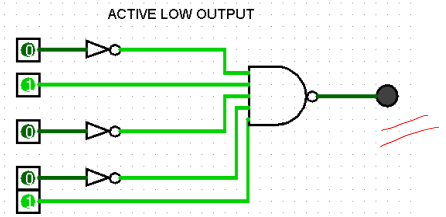
- A **decoder** is a logic circuit that accepts a set of inputs that represents a binary number and activates only the output that corresponds to that input number.
- *N* inputs and *M* outputs
- 2^N possible input combinations or codes



Decoders

- Develop the logic required to detect the binary code 10010 and produce an active LOW output.

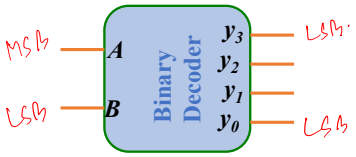




Decoders

2-to-4 Line Decoder

Block Diagram



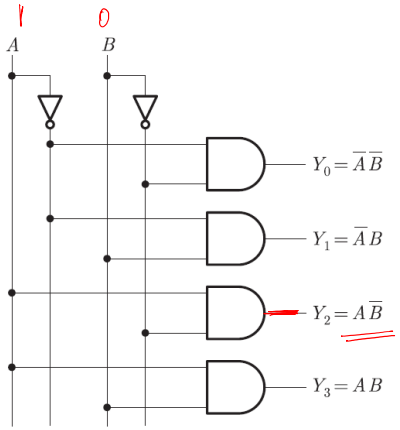
Truth Table

A B	Y_3 Y_2 Y_1 Y_0	
0 0	0 0 0 1	m_0
0 1	0 0 1 0	m_1
1 0	0 1 0 0	m_2
1 1	1 0 0 0	m_3

Logical expressions

$Y_0 = \overline{A}\,\overline{B}$ and $Y_1 = \overline{A}B$

$Y_2 = A\overline{B}$ and $Y_3 = AB$

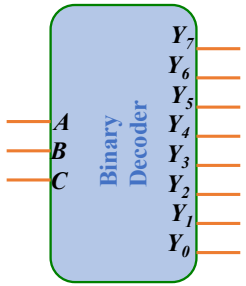


Logic Diagram

Decoders

3-to-8 Line Decoder

Block Diagram



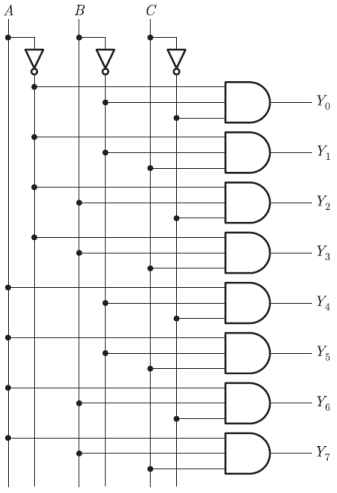
Truth Table

Inputs			Outputs								
A	B	C	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7	
0	0	0	1	0	0	0	0	0	0	0	m_0
0	0	1	0	1	0	0	0	0	0	0	m_1
0	1	0	0	0	1	0	0	0	0	0	m_2
0	1	1	0	0	0	1	0	0	0	0	m_3
1	0	0	0	0	0	0	1	0	0	0	m_4
1	0	1	0	0	0	0	0	1	0	0	m_5
1	1	0	0	0	0	0	0	0	1	0	m_6
1	1	1	0	0	0	0	0	0	0	1	m_7

Logical expressions

$Y_0 = \overline{A}\,\overline{B}\,\overline{C}$; $Y_1 = \overline{A}\,\overline{B}C$; $Y_2 = \overline{A}B\overline{C}$; $Y_3 = \overline{A}BC$

$Y_4 = A\overline{B}\,\overline{C}$; $Y_5 = A\overline{B}C$; $Y_6 = AB\overline{C}$; $Y_7 = ABC$

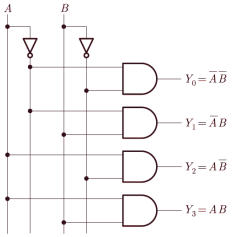


Logic Diagram

Decoders

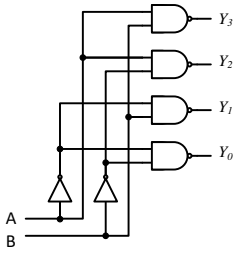
Active-High

A B	Y_3 Y_2 Y_1 Y_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



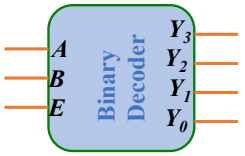
Active-Low

A B	Y_3 Y_2 Y_1 Y_0
0 0	1 1 1 0
0 1	1 1 0 1
1 0	1 0 1 1
1 1	0 1 1 1

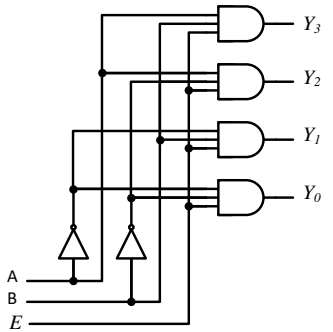


Decoder with ENABLE Input

"*Enable*" Control



E	A B	Y_3 Y_2 Y_1 Y_0
0	x x	0 0 0 0
1	0 0	0 0 0 1
1	0 1	0 0 1 0
1	1 0	0 1 0 0
1	1 1	1 0 0 0

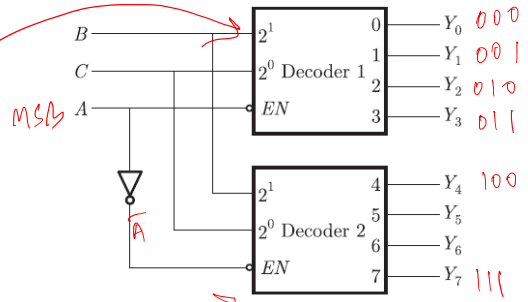


3-to-8 decoder using 2-to-4 decoder

74LS139

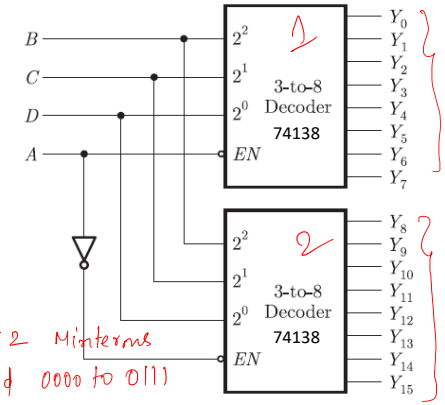
Expansion

A	B	C	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0



4-to-16 Decoder using 3-to-8 Decoder

Binary Inputs				Decimal Output
A	B	C	D	Active Low
0	0	0	0	Y0
0	0	0	1	Y1
0	0	1	0	Y2
0	0	1	1	Y3
0	1	0	0	Y4
0	1	0	1	Y5
0	1	1	0	Y6
0	1	1	1	Y7
1	0	0	0	Y8
1	0	0	1	Y9
1	0	1	0	Y10
1	0	1	1	Y11
1	1	0	0	Y12
1	1	0	1	Y13
1	1	1	0	Y14
1	1	1	1	Y15



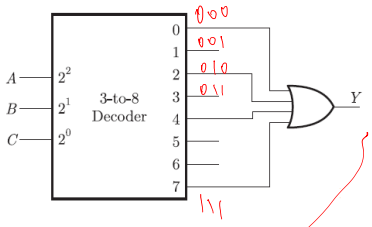
Enable
A — Decoder 1 — Decoder 2 — Minterms
0 — Enabled — Disabled — 0000 to 0111
1 — Disabled — Enabled — 1000 to 1111

IMPLEMENTATION OF LOGIC EXPRESSIONS USING DECODERS

- Any combinational circuit with *n*-inputs and *m* outputs can be implemented with an *n*-to-2^n decoder and OR gates.

Procedure:

1. Express the given Boolean function in sum of minterms.
2. A decoder that generates all the minterms of the input variables is then chosen.
3. The inputs to each OR gate are selected from the decoder outputs according to the list of minterms of each function.



$Y = A \cdot \overline{B} \cdot \overline{C} + \overline{A} \cdot B \cdot \overline{C} + A \cdot B \cdot C + \overline{A} \cdot \overline{B} \cdot \overline{C}$

100 010 111 000

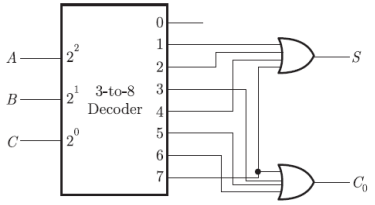
4 2 7 0

$Y = \Sigma m(0, 2, 4, 7)$

Implement a full adder circuit using a 3-to-8 line decoder.

Each output is a minterm
All minterms are produced
Sum the required minterms

Inputs		Cin	Sum	Carry
A	B	C	S	C_0
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



Sum output, $S = \Sigma 1,2,4,7$

Carry output, $C_0 = \Sigma 3,5,6,7$

CSE1003 Digital Logic and Design
Module 4
Combinational Circuits II
L4

Dr. S.Hemamalini
Professor
School of Electrical Engineering
VIT Chennai

Contents

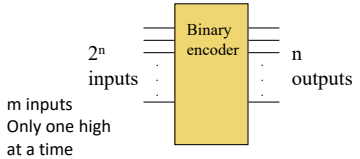
6 hrs

- Binary Parallel Adder - Look ahead carry
- Magnitude Comparator
- **Decoders**
- **Encoders**
- Multiplexers
- Demultiplexers

- CO4: Analyze the operation of medium complexity standard combinational circuits like the encoder, decoder, multiplexer, demultiplexer.

Encoders

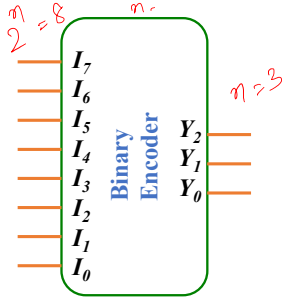
- An encoder is a combinational logic circuit that performs the inverse operation of a decoder.
- The process of converting various numbers (numerals, alphabets, special characters) into a binary format is called encoding.
- An encoder has 2^n (or fewer) input lines and n output lines.
- The opposite of the decoding process is called encoding, i.e. encoding is a process of converting familiar numbers or symbols into a coded format.
- Binary encoders
  - Converts one of 2^n inputs to an n-bit binary output
  - Useful for compressing data
  - Can be developed using AND/OR gates
- The simplest encoder is a 2^n-to-n binary encoder
  - One of 2^n inputs = 1
  - Output is an n-bit binary number
- Commonly used encoders
  - Octal-to-binary encoder
  - Decimal-to-BCD encoder
  - Hexadecimal-to-binary encoder



Block diagram of encoder

8-to-3 Binary Encoder or Octal-to-Binary Encoder

- Have eight input lines, each representing an octal digit, and three output lines representing the three-bit binary equivalent.
- At any one time, only one input line has a value of 1.

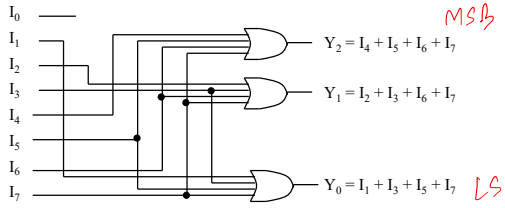


Block diagram of an octal to binary encoder

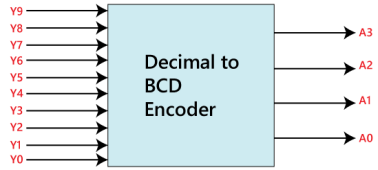
$$Y_2 = I_7 + I_6 + I_5 + I_4$$
$$Y_1 = I_7 + I_6 + I_3 + I_2$$
$$Y_0 = I_7 + I_5 + I_3 + I_1$$

Truth Table of an Octal to Binary Encoder

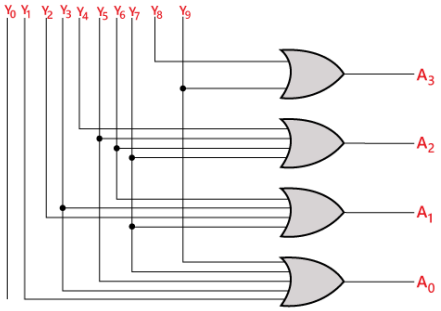
Inputs								Outputs		
I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7	Y_2	Y_1	Y_0
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1



Decimal-to-BCD Encoder



INPUTS										OUTPUTS			
Y_9	Y_8	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0	A_3	A_2	A_1	A_0
0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	1	0
0	0	0	0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0	0	0	0	1	0	1
0	0	0	1	0	0	0	0	0	0	0	1	1	0
0	0	1	0	0	0	0	0	0	0	0	1	1	1
0	1	0	0	0	0	0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	0	0	0	1	0	0	1



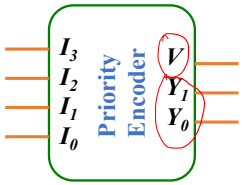
Logic diagram of decimal-to-BCD encoder

$A_3 = Y_8 + Y_9$
$A_2 = Y_4 + Y_5 + Y_6 + Y_7$
$A_1 = Y_2 + Y_3 + Y_6 + Y_7$
$A_0 = Y_1 + Y_3 + Y_5 + Y_7 + Y_9$

Priority Encoders

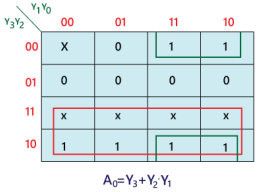
4-to-2 line Priority Encoder

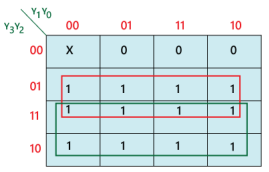
- the higher the subscript number, the higher the priority of the input.



V - Valid bit indicator
V=1 more than 1 i/p line goes HIGH
V=0 when all i/ps are zero

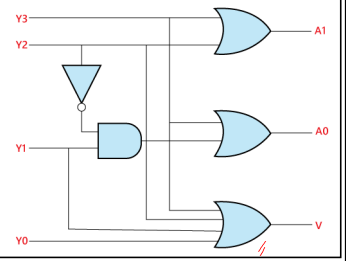
INPUTS				OUTPUTS		
Y_3	Y_2	Y_1	Y_0	A_1	A_0	V
0	0	0	0	X	x	0
0	0	0	1	0	0	1
0	0	1	X	0	1	1
0	1	x	X	1	0	1
1	X	x	X	1	1	1





$A_0=Y_3+Y_2'.Y_1$

$V = Y_3+Y_2+Y_1+Y_0$



Priority Encoders

- A priority encoder generates a binary code corresponding to the number of the active input with highest priority (or, most often, the highest number).
- It can be used in the following applications:
  - – keyboard encoder: when several keys are pressed simultaneously, only the key with the highest number is taken into consideration;
  - – unit processing interrupt requests in a microprocessor: in case of simultaneous interrupt requests, only the request with the highest priority is accepted.

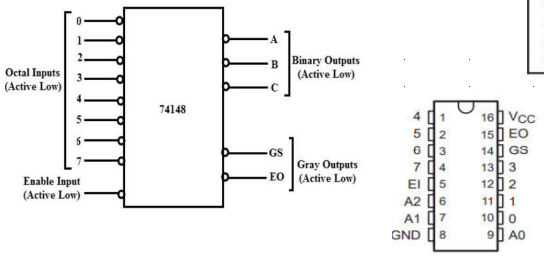
8-to-3 Priority Encoder

- What if more than one input line has a value of 1?
- Ignore "lower priority" inputs.

74148, 74LS148, and 74HC148 are all octal-to-binary priority encoders.

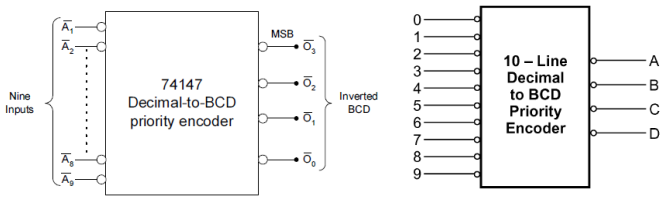
FUNCTION TABLE

INPUTS									OUTPUTS				
EI	0	1	2	3	4	5	6	7	A2	A1	A0	GS	EO
H	X	X	X	X	X	X	X	X	H	H	H	H	H
L	H	H	H	H	H	H	H	H	H	H	H	H	L
L	X	X	X	X	X	X	X	L	L	L	L	L	H
L	X	X	X	X	X	X	L	H	L	L	H	L	H
L	X	X	X	X	X	L	H	H	L	H	L	L	H
L	X	X	X	X	L	H	H	H	L	H	H	L	H
L	X	X	X	L	H	H	H	H	H	L	L	L	H
L	X	X	L	H	H	H	H	H	H	L	H	L	H
L	X	L	H	H	H	H	H	H	H	H	L	L	H
L	L	H	H	H	H	H	H	H	H	H	H	L	H



8-to-3 priority encoder IC 74148

Decimal-to-BCD Encoder

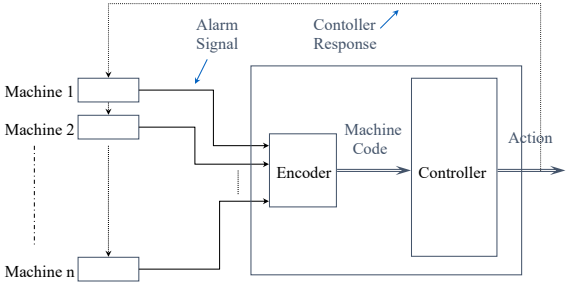


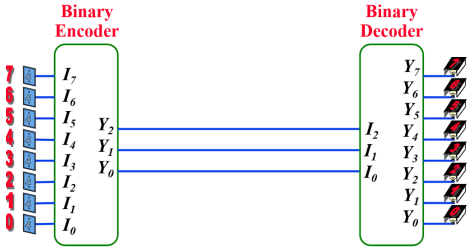
$\overline{A}_1$	$\overline{A}_2$	$\overline{A}_3$	$\overline{A}_4$	$\overline{A}_5$	$\overline{A}_6$	$\overline{A}_7$	$\overline{A}_8$	$\overline{A}_9$	$\overline{O}_3$	$\overline{O}_2$	$\overline{O}_1$	$\overline{O}_0$
1	1	1	1	1	1	1	1	1	1	1	1	1
X	X	X	X	X	X	X	X	0	0	1	1	0
X	X	X	X	X	X	X	0	1	0	1	1	1
X	X	X	X	X	X	0	1	1	1	0	0	0
X	X	X	X	X	0	1	1	1	1	0	0	1
X	X	X	X	0	1	1	1	1	1	0	1	0
X	X	X	0	1	1	1	1	1	1	0	1	1
X	X	0	1	1	1	1	1	1	1	1	0	0
X	0	1	1	1	1	1	1	1	1	1	0	1
0	1	1	1	1	1	1	1	1	1	1	1	0

Encoder Application (Monitoring Unit)

Encoder identifies the requester and encodes the value

Controller accepts digital inputs.





Encoder Applications

1. Keyboard encoder for computers
2. Optical encoders –linear or rotary
3. Interfacing peripherals to microprocessors
4. Audio/video coding and transmission

CSE1003 Problems

Design a logic circuit that has three inputs, *A*, *B*, and *C*, and whose output will be HIGH only when a majority of the inputs are HIGH.

Step 1. Set up the truth table.

A	B	C	x
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Step 2. Write the AND term for each case where the output is a 1.

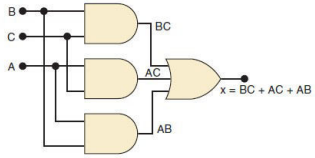
Step 3. Write the sum-of-products expression for the output.

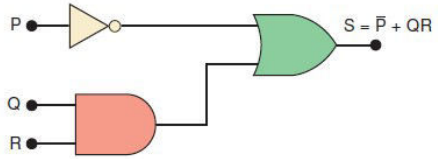
$$x = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$$

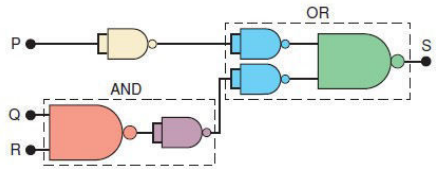
Step 4. Simplify the output expression.

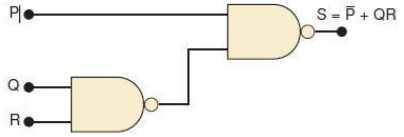
$$x = BC + AC + AB$$

Step 5. Implement the circuit for the final expression.

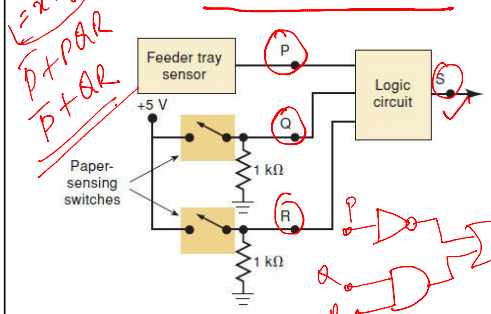








- In a simple copy machine, a stop signal, *S*, is to be generated to stop the machine operation and energize an indicator light whenever either of the following conditions exists: (1) there is no paper in the paper feeder tray; or (2) the two microswitches in the paper path are activated, indicating a jam in the paper path. The presence of paper in the feeder tray is indicated by a HIGH at logic signal *P*. Each of the microswitches produces a logic signal (*Q* and *R*) that goes HIGH whenever paper is passing over the switch to activate it. Design the logic circuit to produce a HIGH at output signal *S* for the stated conditions, and implement it using the 74HC00 CMOS quad two-input NAND chip.



P	Q	R	S
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

$S = \overline{P}\,\overline{Q}\,\overline{R} + \overline{P}\,\overline{Q}R + \overline{P}Q\overline{R} + \overline{P}QR + PQR$

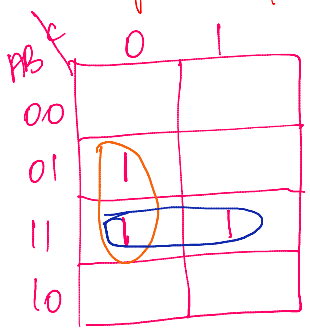
Simplified Boolean expression

$S = \overline{P} + QR$

Simplify $f = A'BC' + AB'C' + ABC$ using maxterms, using K-Map method.

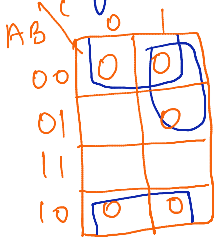
$f = A'BC' + AB'C' + ABC$

010 110 111



Using minterms

$f = BC' + AB$ SOP form



Using maxterms - place 0's in the remaining squares

The group of 0's provides the complement of the function

$$\bar{F} = \bar{B} + \bar{A}C$$

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

$$f = B(A + \bar{C}) \quad \text{POS form}$$

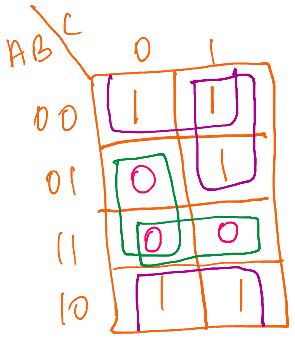
Simplify the following Boolean function into SOP & POS forms.

$$F(A,B,C) = \Sigma(0,1,3,4,5)$$

Fill in 1's in the squares of minterms for which F=1 & fill in 0's in the remaining squares for which F=0

Group of 1's provide the SOP form

Group of 0's provide the complement of the function



Group of 1's — SOP form

$$F = B\bar{C} + A\bar{B}$$

Group of 0's — Complement of F

$$\bar{F} = BC + \bar{A}\bar{B}$$

$$F = \bar{\bar{F}} = \overline{BC + \bar{A}\bar{B}} = \overline{BC} \cdot \overline{\bar{A}\bar{B}}$$

$$= (\bar{B} + \bar{C}) \cdot (A + B)$$

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

Give simplified logic equation & Table by QM method.

	A	B	C	Y
m_0	0	0	0	0
m_1	0	0	1	0
m_2	0	1	0	1
m_3	0	1	1	0
m_4	1	0	0	0
m_5	1	0	1	0
m_6	1	1	0	1
m_7	1	1	1	1

Stage 1

Minterms	Binary A B C	Group
m_2	0 1 0	I
m_6	1 1 0	II
m_7	1 1 1	II

Stage 2

Minterms	Binary A B C	
2, 6	− 1 0	PI1
6, 7	1 1 −	PI2

$Y = B\bar{C} + AB$

Prime Implicant chart

	2	6	7	
2,6 PI1	✓	✓		EPI
6,7 PI2		✓	✓	EPI

CSE1003 problems