

ANALOG AND DIGITAL ELECTRONICS

PROF. AHER VIJAYA NAVNATH

vijaya.aher@viit.ac.in

Department of E&TC Engineering



BRACT'S, Vishwakarma Institute of Information Technology, Pune-48

(An Autonomous Institute affiliated to Savitribai Phule Pune University)
(NBA and NAAC accredited, ISO 9001:2015 certified)

Analog and Digital Electronics

(S.Y. B. Tech.- I.T.)
(2020-21)
(Faculty: AHER V. N.)

Analog and Digital Electronics (ITUA21181)

Teaching Scheme

Credits: 3
Lectures / Week: 3Hrs/week

Examination Scheme

Formative Assessment: 50
Summative Assessment: 50

Prerequisites : Basic Electronics Engineering

Course Objectives :

- To learn basic digital circuit design techniques.
- To study the implementation of digital circuits using combinational logic.
- To explain and implement circuits using sequential logic.
- To illustrate the concept of PLD's & ASM.
- To show the implementation of digital circuits using VHDL.
- To explain the basics of Logic Families.

Course Outcomes :

After studying this course, students will be able to:

1. Simplify Boolean algebraic expressions for designing digital circuits using K-Maps. (Analyzing)
2. Apply digital concepts in designing combinational circuits. (Applying)
3. Apply digital concepts in designing sequential circuits. (Applying)
4. Design digital circuits using PLA and PAL. (Creating)
5. Develop digital circuits using VHDL. (Creating)
6. Design and implement Mini digital circuits. (Creating)

Unit I – Number System and Logic Minimization Techniques

Number System: Introduction, Binary number System, Sign-Magnitude representation, One's and Two's complement representation, Binary arithmetic, 2's complement arithmetic, Octal number System, Hexadecimal number System, Basic theorems and properties.

Logic minimization: Representation of truth-table, Sum of Product (SOP) form, Product of Sum (POS) form, Simplification of logical functions, Minimization of SOP and POS forms using K-Maps up to 4 variables and Quine-McCluskey Technique, realization of logic gates.

Unit II - Combinational Logic

Design of Combinational Logic: Code converter - BCD, Excess-3, Gray code, Binary Code. Half- Adder, Full Adder, Half Subtractor, Full Subtractor, Binary Adder (IC 7483), BCD adder, Look ahead carry generator, Multiplexers (MUX): MUX (IC 74153, 74151), MUX tree, DE multiplexers (DEMUX)- Decoder. (IC 74138, IC 74154). DMUX Tree, Implementation of SOP and POS using MUX, DMUX, Comparators, Parity generators and Checker, One bit, Two bit , 4-bit Magnitude Comparator

Unit III – Sequential Logics

Flip-flop: SR, JK, D, T; Preset & Clear, Master and Slave Flip Flops, Truth Tables and Excitation tables, Conversion from one type to another type of Flip Flop.

Registers: Buffer register, shift register (SISO, SIPO, PISO& PIPO), Applications of shift registers.

Unit IV - Counters

Counters: Asynchronous counter. Synchronous counter, ring counters, Johnson Counter, Modulus of the counter (IC 7490).

Synchronous Sequential Circuit Design: Models – Moore and Mealy, State diagram and State Tables, Design Procedure, Sequence generator and detector.

Asynchronous Sequential Circuit Design: Difference with synchronous circuit design, design principles and procedure, applications.

Unit V – Logic Families

Classification of logic families: Unipolar and Bipolar Logic Families, Characteristics of Digital ICs: Speed, power dissipation, figure of merits, fan-out, Current and voltage parameters, Noise immunity, operating temperature range, power supply requirements.

Transistor-Transistor Logic: Operation of TTL, TTL characteristics.

CMOS: CMOS Inverter, CMOS characteristics, CMOS configurations- Wired Logic, Open drain outputs.

Interfacing: TTL to CMOS and CMOS to TTL.

Digital Circuit Implementation using IOT: Introduction to soldering techniques and its types, Mounting of electronic components on PCB. Introduction and applications of Raspberry pi and Arduino architecture.

Unit VI - Introduction to PLD's & VHDL

Introduction to PLD's: - PAL, PLA, Applications of PLAs to implement combinational and sequential logic circuits, FPGA and CPLD. VHDL: Introduction to HDL, Data Objects & Data Types, Attributes., VHDL- Library, Design Entity, Architecture, Modeling Styles, Concurrent and Sequential Statements. Design Examples: VHDL for Combinational Circuits-Adder, MUX, VHDL for Sequential Circuits, Synchronous and Asynchronous Counter.

Text books :	<ol style="list-style-type: none"> 1. R. P. Jain, "Modern Digital Electronics", 3rd Edition, Tata McGraw-Hill, 2003, ISBN0 – 07 – 049492 – 4. 2. J. Bhaskar, "VHDL Primer" 3rd Edition, Pearson Edition. 3. Stephen Brown, Zvonko Vranesic, "Fundamentals of Digital Logic with VHDL Design", McGraw Hill, ISBN-13:978-1-25-902597-6. 4. G.K.Kharate, "Digital Electronics", Oxford University Press.
Reference Books :	<ol style="list-style-type: none"> 1. John Yarbrough, "Digital Logic applications and Design" Thomson. 2. Flyod "Digital Principles", Pearson Education. 3. Malvino, D.Leach "Digital Principles and Applications", 5th edition, Tata Mc- Graw Hill. 4. Douglas L. Perry, "VHDL Programming by Example", Tata McGraw Hill. 5. G. K. Kharate, "Digital Electronics", Oxford University Press.

Topics To Be Covered

Unit-1: Number System & Logic Minimization Techniques.

Unit-2: Combinational Logic.

Unit-3: Sequential Logic.

Unit-4: Counters.

Unit-5: Logic Families.

Unit-6: Introduction to PLD's & VHDL.

Number System & Logic Minimization Techniques

Unit-I

Bits, Bytes, Nybbles

- The most important base you need to know for computer organization is base 2. We also call this **binary**, since binary means two valued.
- The second most important base is base 16 (or hexadecimal), and the third is base 8 (octal). Beyond that, you don't need to know the other bases.
- Let's start with some definitions first.

Bit : A single binary digit, that can have either value 0 or 1.

Byte : 8 bits.

Nybble: 4 bits.

Word : 32 bits

Halfword: 16 bits

Doubleword: 64 bits

Analog: A way of representing some physical quantity in such a way that physical quantity varies continuously. An analog voltage or current have any value within a defined range.

➤ **Digital: A way of representing a physical quantity by a specific discrete values.**

Difference between Analog & Digital Signals:

Analog	Digital
1. Continuously variable	1. Discrete Steps
2. Amplification	2. Switching
3. Voltages	3. Numbers

Number System

- In any number system there is an ordered set of symbols known as digits with rules defined for performing arithmetic operations like addition, multiplication etc.
- A collection of these digits makes a number which in general has two parts- integer and fractional, set apart by a radix point(.).

Binary Number System

The binary number system contains only

- two symbols: 1 and 0
- In place of electrical current we can use these symbols to abstractly represent some internal state of the computer achieved through combinations of ON/OFF values
- We can represent
 - data
 - somewhere to store the data
 - instructions by combining individual bits

Binary Number System

- The binary number system is a positional number system
- We use the decimal number system which is also positional
- eg in the number 333 the digits are the same but are interpreted based on their position

$$(3 \times 100) + (3 \times 10) + (3 \times 1)$$

or

$$(3 \times 10^2) + (3 \times 10^1) + (3 \times 10^0) \text{ based on}$$

100000 10000 1000 100 10 1

10^5 10^4 10^3 10^2 10^1 10^0

- Whereas the decimal system has ten digits 0,1,2,3,4,5,6,7,8,9 binary only has two 0 and 1
- With positional values

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1

- so the binary number 10101 can be converted to a decimal value

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
0	0	0	1	0	1	0	1

- $(1 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0)$
- $= 16 + 0 + 4 + 0 + 1$
- $= 21$

Signed Binary Numbers

Sign-Magnitude Representation:

- In decimal number system a plus(+) sign is used to denote a positive number & a minus(-) sign for denoting a negative number.
- This representation of numbers is known as signed number.
- Digital circuits understand only two symbols, 0 or 1; so we must use the same symbols to indicate the sign of the number.
- An additional bit is used as the sign bit & is placed as the MSB (Most Significant Bit).
- A 0 is used to represent a positive number & a 1 is used to represent a negative number.
- E.g. An 8-bit signed number (01000100) represents a positive number & its value (magnitude) is (1000100) = (68). The MSB indicates that the number is positive.
- E.g. In signed binary form, 11000100 represents a negative number with magnitude (1000100) = (68). The 1 in the left most position (MSB) indicates that the number is negative & the other seven bits give its magnitude. This representation is known as sign-magnitude representation.

Negative integers

So far we have only non-negative (*unsigned*) integers

In base 10 we use a minus sign

-213

In binary we use the most significant bit (MSB)

- if MSB is 1 number is negative
- if MSB is 0 number is positive
- other 7 bits represent magnitude

2^6	2^5	2^4	2^3	2^2	2^1	2^0
64	32	16	8	4	2	1
0	0	1	1	0	0	1

- if this is 0 value is 25
- if this is 1 value is -25

Called "sign-magnitude" form

Can represent -127 to 127

Sign-magnitude problems

Representing zero

$$00000000 = ?$$

$$10000000 = ?$$

Adding negative to positive

	00101101		45
+	10011000	+	-24
	<hr/>		<hr/>
	11000101		-69

Need a better solution.....

1's Complement Representation

- In a binary number, if each 1 is replaced by 0 & each 0 by 1, the resulting number is known as the 1's complement of the first number.
- Thus both the numbers are complement of each other.
- If one of these numbers is positive, then the other number will be negative with the same magnitude.
- E.g. (0101) binary no. represents (+5) decimal no. whereas (1010) binary no. represents (-5) decimal no.
- It is widely used for representing signed numbers, where MSB is 0 for positive numbers & 1 is for negative numbers.
- E.g. (+7) & (-7) in 1's complement is (0111) & (1000).

Note: The 1's complement of a binary number is the number that results when we change all 1's to 0's & the 0's to 1's.

1's Complement Representation

Steps:

1. Find 1's complements of Subtrahend.
2. Add two nos. using binary addition.
3. Check carry if carry is generated, add the carry in LSB position.
4. If MSB is 1 then the result is negative & it is in 1's Complement form.
5. If MSB is 0 then the result is positive & is true form.

E.g.

1. $8 - 9 = -1$

step 1: $8 = 1000$

$9 = 1001$ 1's complement of 9 is = 0110

step2: $1000 \leftarrow 8$

$+ 0110 \leftarrow 1's \text{ complement of } 9$

1110

step3: The MSB is 1, so the result is negative & in 1's complement form. So result is = $-(0001) = -1$

2. $9 - 8 = 1$ Ans. $(0001) = 1$.

Advantages of 1's Complement

1. The 1's complement subtraction can be accomplished with an binary adder, therefore this method is useful in arithmetic logic circuits.
2. The 1's complement of a number is easily obtained by inverting each bit in the number.

Binary Arithmetic

Binary Addition: Rules for binary addition are as follows:

Augend	Addend	Sum	Carry	Result	
0		0	0	0	0
0		1	1	0	1
1		0	1	0	1
1		1	0	1	10

Binary Subtraction: Rules of Binary Subtraction are as follows:

Minuend	Subtrahend	Difference	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Binary Arithmetic

Binary Addition:

Add the binary numbers:

- 1011 & 1100 Ans. 1 0111
- 0101 & 1111 Ans. 1 0100
- 01101010, 00001000, 10000001, 11111111 Ans. 111110010

Binary addition

What happens when 2 numbers are added together?

- shall we convert to decimal, add them then convert back to binary?
- no need
- binary addition follows same principles as decimal, using carry digits/bits

eg carry when sum > 9

$$\begin{array}{r}
 163 \\
 + 159 \\
 \hline
 322
 \end{array}$$

eg carry when sum > 1

$$\begin{array}{r}
 01010011 \\
 + 00010101 \\
 \hline
 01101000
 \end{array}$$

2's Complement Representation

- Digital Circuits are used for binary arithmetic operations.
- It is possible to use the circuits designed for binary addition to perform the binary subtraction, if we can change the problem of subtraction to that of an addition.

Subtraction Using 2's Complement:

- Binary Subtraction can be performed by adding 2's complement of the subtrahend to the minuend.
- If a final carry is generated, discard the carry and the answer is given by the remaining bits which is +ve (the minuend is greater than the subtrahend).
- If final carry is 0, the answer is -ve (the minuend is smaller than the subtrahend) and is in 2's complement form.

2's Complement Representation

- If 1 is added to 1's complement of a binary number, the resulting number is known as the two's complement of the binary number.

E.g. 2's complement of 0101 is 1011.

- In this representation also if MSB is 0 the number is positive, whereas if the MSB is 1 the number is negative.

Steps:

1. Find 2's complement of a subtrahend.
2. Add two nos. using binary addition.
3. Check carry, if carry is generated, discard the carry. If MSB is 1 then the answer is negative & it is in 2's complement form. If MSB is 0, then the result is positive & in true form.
4. E.g. $8-9 = -1$

step 1 : 8 = 1000 (1's Comp.) (Add 1)

 9 = 1001 2's complement = 0110 = 0111

step 2: 1000 + 0111 = 1111 (No carry generated)

Thus MSB is 1, so result is negative & it is in 2's complement form. So we have $-(0001)$ 2's complement of result.

(H. W. -> Solve Eg.1) 9- 8 =1, 2) 7- 5 = +2 3) 5 - 7 = - 2

2's Complement Representation

Steps for direct conversion:

- 1.If LSB of number is 1, its 2's complement is obtained by changing each 0 to 1 and 1 to 0 except the Least-Significant Bit.**
- 2.If LSB of number is 0, its 2's complement is obtained by scanning the number from the LSB to MSB bit by bit & retaining the bits as they are up to and including the occurrence of the first 1 & complement all other bits.**

E.g.

- | | | |
|------------|-----------------------|-----------------|
| 1 . | Number | 01100100 |
| | 2's Complement | 10011100 |
| 2. | Number | 01100111 |
| | 2's complement | 10011001 |

2's Complement Representation

Addition/Subtraction in 2's Complement Representation:

- It can conveniently be performed using 2's complement representations for both the operands.
- It is most commonly used method when these operations are performed using digital circuits and microprocessors.

E.g. Perform following operations using 2's complement method:

- 1) $48 - 23$ 2) $23 - 48$ 3) $48 - (-23)$ 4) $-48 - 23$

Using 8-bit representation of numbers.

Sol:

1) 2's complement representation of $+48 = 00110000$

2) 2's complement representation of $-23 = 11101001$

$$\begin{array}{r}
 48 + (-23) \qquad 48 \qquad \qquad \qquad 00110000 \\
 \qquad \qquad \qquad +(-23) \qquad (+)11101001 \\
 \hline
 \qquad \qquad \qquad +25 \qquad \qquad 1 \quad 00011001 \qquad \qquad = +25 \\
 \qquad \qquad \qquad \qquad \qquad \qquad \text{(Discard Carry)}
 \end{array}$$

Ans. 2) -25 3) +71 4) -71 (See Pg. 34 R. P. Jain)

2's Complement Representation

1. If two operands are of the opposite sign, the result is to be obtained by the rule of subtraction using 2's complement.
2. If the two operands are of the same sign, the sign bit of the result (msb) is to be compared with the sign bit of the operands.
3. In case sign bits are same, the result is correct and is in 2's complements form.
4. If the sign bits are not same there is a problem of overflow, i.e. the result can not be accommodated using eight bits and the result is to be interpreted suitably.
5. The result in this case will consist of nine bits i.e. carry and eight bits, and the carry bit will give the sign of the number.

Interpreting 2's complement

If MSB is 0 number is positive, interpret normally

If MSB is 1 number is negative

- complement all bits
- add 1
- interpret as unsigned integer but remember value is negative

Try

4 - 7

10 - 18

2's complement

positive integers are in sign-magnitude form

negative numbers are equivalent positive number with all bits "flipped" (complemented), and 1 added to result

eg

0 0 0 0 0 1 1 1 +7

flip 1 1 1 1 1 0 0 0

add 1 1 1 1 1 1 0 0 1 -7

2's complement

45 + (-24)

+24 0 0 0 1 1 0 0 0

flip 1 1 1 0 0 1 1 1

add 1 1 1 1 0 1 0 0 0 -24

$$\begin{array}{r}
 00101101 45 \\
 + 11101000 -24 \\
 \hline
 00010101 21
 \end{array}$$

How do we know if result is negative or positive?

2's complement

45 + (-24)

+24 0 0 0 1 1 0 0 0

flip 1 1 1 0 0 1 1 1

add 1 1 1 1 0 1 0 0 0 -24

	0 0 1 0 1 1 0 1		45
+	<u>1 1 1 0 1 0 0 0</u>	+	<u>- 24</u>
	0 0 0 1 0 1 0 1		21

How do we know if result is negative or positive?

Number System and Base Conversions

- Electronic and Digital systems may use a variety of different number systems, (e.g. Decimal, Hexadecimal, Octal, Binary).
- A number N in base or radix b can be written as:
$$(N)_b = d_{n-1} d_{n-2} \text{ --- } d_1 d_0 . d_{-1} d_{-2} \text{ --- } d_{-m}$$
- In the above, d_{n-1} to d_0 is integer part, then follows a radix point, and then d_{-1} to d_{-m} is fractional part.
- d_{n-1} = Most significant bit (MSB)
- d_{-m}

Base	Representation
2	Binary
8	Octal
10	Decimal
16	Hexadecimal

Binary to Decimal

$$(1010.01)_2$$

$$1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2}$$

$$= 8 + 0 + 2 + 0 + 0 + 0.25$$

$$= 10.25$$

$$(1010.01)_2 = (10.25)_{10}$$

Conversions

Binary To Decimal:

Binary Number

Decimal Number

1. $(11011)_2$:

$(27)_{10}$

2. $(0.1101)_2$:

$(0.8125)_{10}$

H.W. : (Binary To Decimal)

1. $(1101.1101)_2$ $(13.8125)_{10}$

2. $(1101.11)_2$

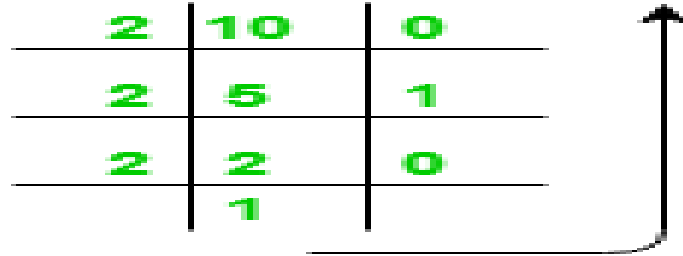
3. $(10101.101)_2$

Decimal to Binary

$(10.25)_{10}$

Integer part :

2	10	0
2	5	1
2	2	0
	1	



$$(10)_{10} = (1010)_2$$

Fractional part

$$0.25 \times 2 = 0.50$$

$$0.50 \times 2 = 1.00$$



$$(0.25)_{10} = (0.01)_2$$

Note: Keep multiplying the fractional part with 2 until decimal part 0.00 is obtained.

$$(0.25)_{10} = (0.01)_2$$

$$\text{Answer: } (10.25)_{10} = (1010.01)_2$$

Conversions

Decimal to Binary:

1. $(46)_{10} = (101110)_2$

2. $(115)_{10} = (\quad)_2$

3. $(0.2)_{10} = (0011)_2$

4. $(25.75)_{10} = (\quad)_2$

Result:

1. $(101110)_2$

2. $(1110011)_2$

3. $(0011)_2$

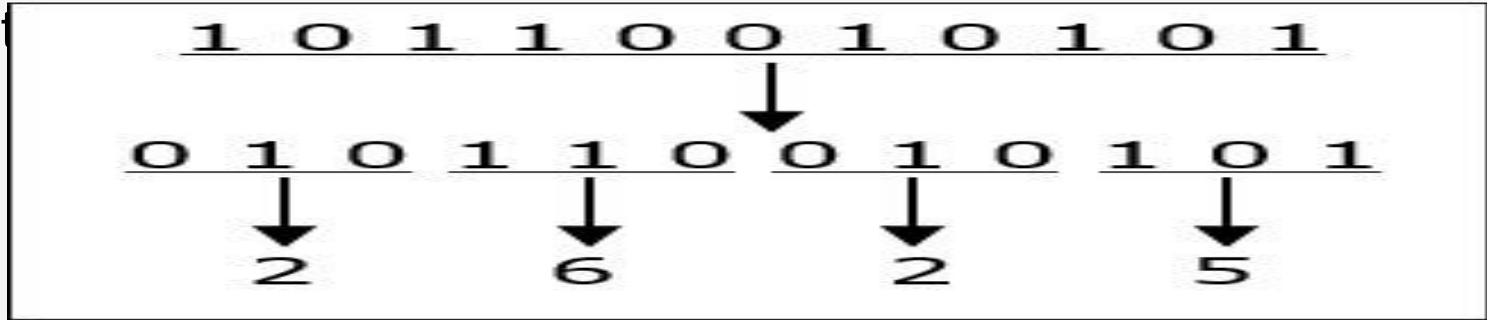
4. $(11001.110)_2$

H.W. : 1. $(268)_{10}$ 2. $(39.50)_{10}$ 3. $(1120.10)_{10}$ 4. $(545646)_{10}$

Binary to Octal

To convert a binary number to octal number, these steps are followed :

- Starting from the least significant bit, make groups of three bits.
- If there are one or two bits less in making the groups, 0s can be added after the most significant bit
- Convert each group into its equivalent octal number
- Let's



$$0101100101_2 = 2625_8$$

Octal to Binary

To convert an octal number to binary, each octal digit is converted to its 3-bit binary equivalent according to this table.

Octal Digit	0	1	2	3	4	5	6	7
Binary Equivalent	000	001	010	011	100	101	110	111

$$54673_8 = 101100110111011_2$$

Conversions

Octal To Binary: (Make combination of 3 bits for conversion since 0 to 7)

	Octal		Binary
1.	$(155.52)_8$	=	$(001101101.101010)_2$
2.	$(16.70)_8$	=	$(001110.111000)_2$

H.W. : (Octal To Binary)

1. $(762)_8$
2. $(0.231)_8$
3. $(762.301)_8$

Binary To Octal:

	Binary	Octal
1.	$(100011110.010111101)_2$	$(436.275)_8$
2.	$(1001.00110)_2$	$(11.14)_8$

Octal to Decimal

$$(12.2)_8$$

$$1 \times 8^1 + 2 \times 8^0 + 2 \times 8^{-1} = 8 + 2 + 0.25 = 10.25$$

$$(12.2)_8 = (10.25)_{10}$$

Decimal to Octal

$$(10.25)_{10}$$

$$(10)_{10} = (12)_8$$

Fractional part:

$$0.25 \times 8 = 2.00$$

Note: Keep multiplying the fractional part with 8 until decimal part .00 is obtained.

$$(.25)_{10} = (.2)_8$$

Answer: $(10.25)_{10} = (12.2)_8$

Conversions

Octal To Decimal:

Octal Number

Decimal

- | | | |
|----|-----------------|-----------------------|
| 1. | $(427.35)_8$ | $(279.4531)_{10}$ |
| 2. | $(6327.4051)_8$ | $(3287.5100098)_{10}$ |

Decimal To Octal:

Decimal Number

Octal Number

- | | | |
|----|-----------------------|-----------------|
| 1. | $(266)_{10}$ | $(412)_8$ |
| 2. | $(0.256)_{10}$ | $(0.20304)_8$ |
| 3. | $(247)_{10}$ | $(367)_8$ |
| 4. | $(0.6875)_{10}$ | $(0.54)_8$ |
| 5. | $(3287.5100098)_{10}$ | $(6327.4051)_8$ |

Conversions

- Hexadecimal Number System:

Hexadecimal	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Decimal to Hexadecimal

Here is an example of using repeated division to convert 1792 decimal to hexadecimal:

Decimal Number	Operation	Quotient	Remainder	Hexadecimal Result
1792	÷ 16 =	112	0	0
112	÷ 16 =	7	0	00
7	÷ 16 =	0	7	700
0	done.			

The only addition to the algorithm when converting from decimal to hexadecimal is that a table must be used to obtain the hexadecimal digit if the remainder is greater than decimal 9.

Decimal:	0	1	2	3	4	5	6	7
Hexadecimal:	0	1	2	3	4	5	6	7
Decimal:	8	9	10	11	12	13	14	15
Hexadecimal:	8	9	A	B	C	D	E	F

Conversions

Hexadecimal To Octal:

Hexadecimal Number

Octal Number

- | | | |
|----|------------|---------------|
| 1. | (4ECE.43F) | (047316.2077) |
| 2. | (A72E) | (123456) |
| 3. | (0.BF85) | (0.577024) |

Decimal To Hexadecimal:

Decimal Number

Hexadecimal Number

- | | | |
|----|-----------|---------|
| 1. | (214) | (D6) |
| 2. | (0.35) | (0.599) |
| 3. | (95.5) | (5F.8) |
| 4. | (675.625) | (2A3.A) |

Hexadecimal To Decimal:

- | | | |
|----|-----------|-------------|
| 1. | (6ABC.2A) | (27324.164) |
| 2. | (3A.2F) | (58.1836) |

Hexadecimal to Binary

To convert from Hexadecimal to Binary, write the 4-bit binary equivalent of hexadecimal.

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

$$(3A)_{16} = (00111010)_2$$

To convert from Binary to Hexadecimal, group the bits in groups of 4 and write the hex for the 4-bit binary. Add 0's to adjust the groups.

1111011011

Conversions

Binary To Hexadecimal:

	Binary Number	Hexadecimal Number
1.	$(110111.10101111)_2$	$(37.AF)_{16}$
2.	$(10100110101111)_2$	$(29AF)_{16}$
3.	$(0.00011110101101)_2$	$(0.1EB4)_{16}$

Hexadecimal To Binary:

	Hexadecimal Number	Binary Number
1.	$(23)_{16}$	$(00100011)_2$
2.	$(F23.45B)_{16}$	$(111100100011.010001011011)_2$
3.	$(2F9A)$	$(0010111110011010)_2$

Logic Minimization:

For two binary variables (taking values 0 and 1) there are 16 possible functions. The functions involve only three operations which make up Boolean algebra: **AND**, **OR**, and **COMPLEMENT**. They are symbolically represented as follows:

AND: $A \cdot B$ OR: $A + B$ COMPLEMENT: $\bar{A} = \text{NOT } A$

These operations are like ordinary algebraic operations in that they are commutative, associative, and distributive. There is a group of useful theorems of Boolean algebra

Boolean algebra (a)

- ◆ In 1854 George Boole introduced a systematic treatment of logic and developed for this purpose an algebraic system called *Boolean algebra*

TABLE 2-1
Postulates and Theorems of Boolean Algebra

Postulate 2	(a) $x + 0 = x$	(b) $x \cdot 1 = x$
Postulate 5	(a) $x + x' = 1$	(b) $x \cdot x' = 0$
Theorem 1	(a) $x + x = x$	(b) $x \cdot x = x$
Theorem 2	(a) $x + 1 = 1$	(b) $x \cdot 0 = 0$
Theorem 3, involution	$(x')' = x$	
Postulate 3, commutative	(a) $x + y = y + x$	(b) $xy = yx$
Theorem 4, associative	(a) $x + (y + z) = (x + y) + z$	(b) $x(yz) = (xy)z$
Postulate 4, distributive	(a) $x(y + z) = xy + xz$	(b) $x + yz = (x + y)(x + z)$
Theorem 5, DeMorgan	(a) $(x + y)' = x'y'$	(b) $(xy)' = x' + y'$
Theorem 6, absorption	(a) $x + xy = x$	(b) $x(x + y) = x$

Boolean Algebra Theorems:

The applications of digital logic involve functions of the AND, OR, and NOT operations. These operations are subject to the following identities:

$ABC = (AB)C = A(BC)$, $A+B+C = (A+B)+C = A+(B+C)$: AND, OR are **associative**.

$AB = BA$, $A+B = B+A$: AND and OR operations are **commutative**.

$A + BC = (A+B)(A+C)$, $A(B+C) = AB + AC$: forms of the **distributive property**.

$\overline{A + B} = \overline{A} \overline{B}$: a form of **DeMorgan's Theorem**.

$\overline{AB} = \overline{A} + \overline{B}$: a form of **DeMorgan's Theorem**.

$AA = A$, $A+A = A$, $A + \overline{A} = 1$, $A\overline{A} = 0$, $A = \overline{\overline{A}}$: **Single Variable Theorems**.

$A + AB = A$, $A + \overline{A}B = A+B$: More **two-variable** theorems.

$A1 = A$, $A + 1 = 1$, $A + 0 = A$, $A0=0$, $\overline{1}=0$, $\overline{0}=1$: **Identity and Null** operations.



Theorems of Boolean Algebra (I)



■ Elementary

1. $X + 0 = X$

2. $X + 1 = 1$

3. $X + X = X$

4. $\overline{\overline{X}} = X$

5. $X + \overline{X} = 1$

1D. $X \cdot 1 = X$

2D. $X \cdot 0 = 0$

3D. $X \cdot X = X$

5D. $X \cdot \overline{X} = 0$

■ Commutativity:

6. $X + Y = Y + X$

6D. $X \cdot Y = Y \cdot X$

■ Associativity:

7. $(X + Y) + Z = X + (Y + Z)$

7D. $(X \cdot Y) \cdot Z = X \cdot (Y \cdot Z)$

■ Distributivity:

8. $X \cdot (Y + Z) = (X \cdot Y) + (X \cdot Z)$

8D. $X + (Y \cdot Z) = (X + Y) \cdot (X + Z)$

■ Uniting:

9. $X \cdot Y + X \cdot \overline{Y} = X$

9D. $(X + Y) \cdot (X + \overline{Y}) = X$

■ Absorption:

10. $X + X \cdot Y = X$

10D. $X \cdot (X + Y) = X$

11. $(X + \overline{Y}) \cdot Y = X \cdot Y$

11D. $(X \cdot \overline{Y}) + Y = X + Y$



Theorems of Boolean Algebra (II)



- **Factoring:**

$$12. (X \cdot Y) + (X \cdot Z) = \\ X \cdot (Y + Z)$$

$$12D. (X + Y) \cdot (X + Z) = \\ X + (Y \cdot Z)$$

- **Consensus:**

$$13. (X \cdot Y) + (Y \cdot Z) + (\bar{X} \cdot Z) = \\ X \cdot Y + \bar{X} \cdot Z$$

$$13D. (X + Y) \cdot (Y + Z) \cdot (\bar{X} + Z) = \\ (X + Y) \cdot (\bar{X} + Z)$$

- **De Morgan's:**

$$14. \overline{(X + Y + \dots)} = \bar{X} \cdot \bar{Y} \cdot \dots$$

$$14D. \overline{(X \cdot Y \cdot \dots)} = \bar{X} + \bar{Y} + \dots$$

- **Generalized De Morgan's:**

$$15. \overline{f(X_1, X_2, \dots, X_n, 0, 1, +, \cdot)} = f(\bar{X}_1, \bar{X}_2, \dots, \bar{X}_n, 1, 0, \cdot, +)$$

- **Duality**

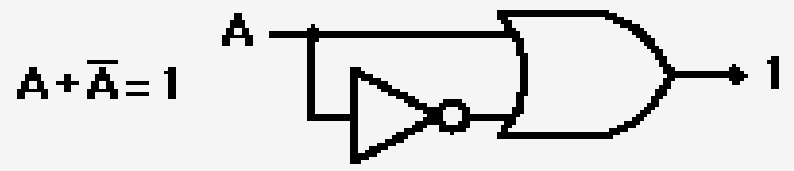
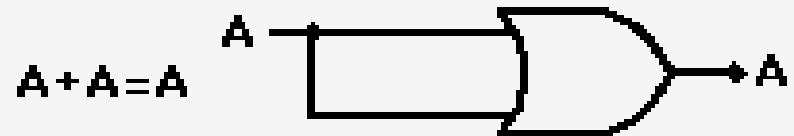
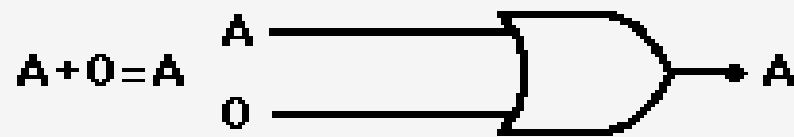
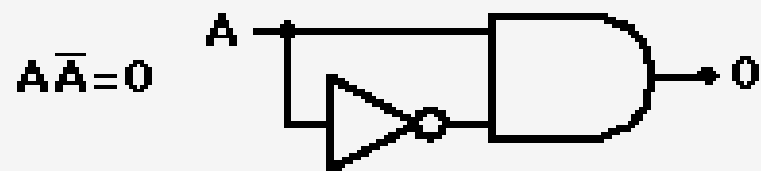
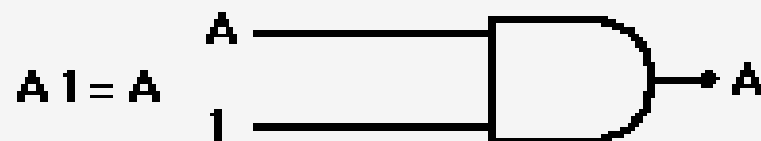
- Dual of a Boolean expression is derived by replacing \cdot by $+$, $+$ by \cdot , 0 by 1, and 1 by 0, and leaving variables unchanged
- $f(X_1, X_2, \dots, X_n, 0, 1, +, \cdot) \Leftrightarrow f(X_1, X_2, \dots, X_n, 1, 0, \cdot, +)$

Binary Functions of Two Variables

Digital logic involves combinations of the three types of operations for two variables: AND, OR, and NOT. There are sixteen possible functions:

1. Null	0	9. NOT OR	$\overline{A + B}$
2. AND	AB	10. EXCLUSIVE NOR	$AB + \overline{A}\overline{B}$
3. A AND NOT B	$A\overline{B}$	11. NOT B	\overline{B}
4.	A	12. A OR NOT B	$A + \overline{B}$
5. NOT A AND B	$\overline{A}B$	13. NOT A	\overline{A}
6.	B	14. NOT A OR B	$\overline{A} + B$
7. EXCLUSIVE OR	$A\overline{B} + \overline{A}B$	15. NOT A AND B	\overline{AB}
8. OR	$A + B$	16. IDENTITY	1

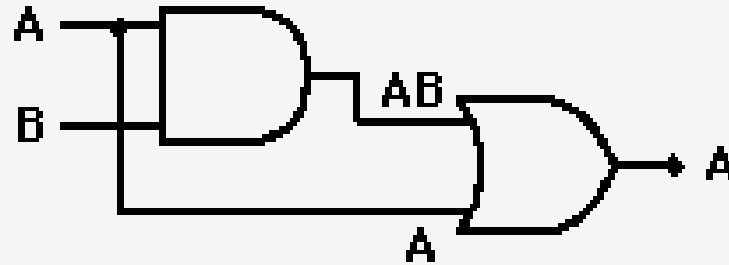
Single Variable Theorems



Two-Variable Theorems

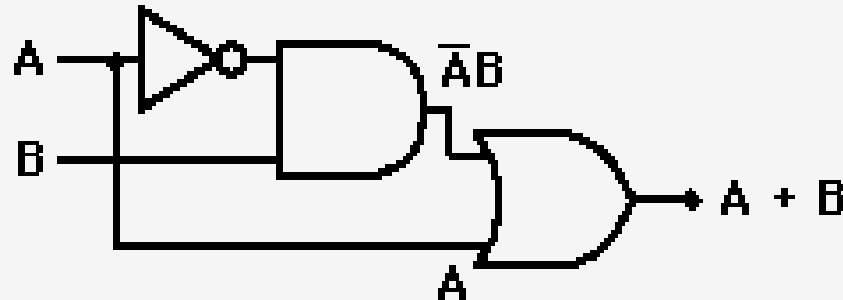
Besides the important De-Morgan's Theorem, the theorems below have utility in digital circuits. They have no direct counterparts in ordinary algebra.

$$A + AB = A$$



By the distributive property,
 $A + AB = A(1+B) = A$

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

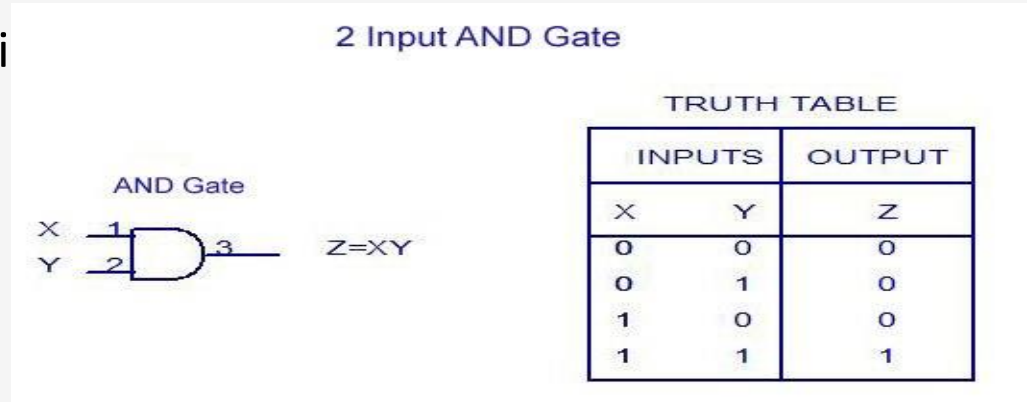


Representation of Truth-Table

- Truth table gives the values of the output variables for all the possible combinations of the input variables.
- Consider a Boolean function $F = A \cdot B$ of the logic AND operation. In this function A & B are the two input independent variables and will have 2^N ($2^2 = 4$) possible input combinations, where N is the number of input variables. Each input combination gives rise an output.
- All possible values of input and output variables listed in the form of a table is known as truth table.

➤

<u>Input Variables</u>	
A	B
0	0
0	1
1	0
1	1



Representation of Truth-Table

Draw the truth table of a Boolean function given: $F = A \cdot B + C$

Solution: The given expression has the three independent variables, so it will have 8 different horizontal rows (as $2^3 = 8$). Putting all possible values of the independent variables in the binary progression and evaluated values of the dependent variable F from the given expression $F = A \cdot B + C$, the required truth table is obtained which is shown in table below.

Input variables		
A	B	C
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Canonical Forms for Boolean Function

- Two basic forms of Boolean function corresponding to a given truth table are Canonical SP form (Sum of Products) and Canonical PS form (Product of Sums).

Canonical SP (or SOP) Form:

- In this form boolean function of the truth table are obtained by summing (ORing) the product (ANDed) terms corresponding to the 1's entry in the output column of the truth table.
- The product terms also known as minterms are formed by ANDing the complemented and uncomplemented variables in such a way that the complement of variable is taken for the 0's entry to the input variable and the variable itself is taken for 1's entry in the input variable.
- The minterms (ANDed terms or products) for the two variables A and B are shown in table:

Input variables		Minterms
A	B	
0	0	$\overline{A} \cdot \overline{B}$
0	1	$\overline{A} \cdot B$
1	0	$A \cdot \overline{B}$
1	1	$A \cdot B$

Canonical Forms for Boolean Function

Canonical PS (or POS) Form:

- In this form boolean function of the truth table are obtained by taking the product (ANDing) of the sum (ORed) terms corresponding to the 0's entry in the output column of the truth table.
- The ORed terms are called as maxterms.
- Maxterms are formed by ORing the complemented and uncomplemented variables present in a row of the truth table in such a way that the complement of variable is taken for the 1's entry to the input variable and the variable itself is taken for 0's entry in the

➤ The maxterm

Input variables			Maxterms	Maxterm notation
A	B	C		
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

Simplification of Logical functions

- **Boolean** algebra is used to **simplify Boolean** expressions which represent combinational logic circuits.
- It reduces the original expression to an equivalent expression that has fewer terms which means that less logic gates are needed to implement the combinational logic circuit.

➤ Simplification Rules:

The Idempotent Laws

$$AA = A$$

$$A+A = A$$

The Associative Laws

$$(AB)C = A(BC)$$

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

The Commutative Laws

$$AB = BA$$

$$A+B = B+A$$

The Distributive Laws

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

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

The Identity Laws

$$AF = F$$

$$AT = A$$

$$A+F = A$$

$$A+T = T$$

The Complement Laws

$$A\overline{A} = F$$

$$A+\overline{A} = T$$

$$\overline{\overline{F}} = F$$

$$\overline{\overline{T}} = T$$

The Involution Law

$$\overline{\overline{A}} = A$$

DeMorgan's Law

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

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

Simplification of Logical functions

- Simplify: $C + \overline{BC}$:

Expression

$$C + \overline{BC}$$

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

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

$$1 + \overline{B}$$

$$1$$

Rule(s) Used

Original Expression

DeMorgan's Law.

Commutative, Associative Laws.

Complement Law.

Identity Law.

Simplification of Logical functions

- Simplify: $\overline{A}B(\overline{A} + B)(\overline{B} + B)$:

Expression

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

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

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

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

$$\overline{A}$$

Rule(s) Used

Original Expression

Complement law, Identity law.

DeMorgan's Law

Distributive law. This step uses th

Complement, Identity.

Simplification of Logical functions

- Simplify: $(A + C)(AD + A\bar{D}) + AC + C$:

Expression

$$(A + C)(AD + A\bar{D}) + AC + C$$

$$(A + C)A(D + \bar{D}) + AC + C$$

$$(A + C)A + AC + C$$

$$A((A + C) + C) + C$$

$$A(A + C) + C$$

$$AA + AC + C$$

$$A + (A + C)C$$

$$A + C$$

Rule(s) Used

Original Expression

Distributive.

Complement, Identity.

Commutative, Distributive.

Associative, Idempotent.

Distributive.

Idempotent, Identity, Distributive.

Identity, twice.

Methods to Simplify Boolean Functions

1. **Algebraic Method.**
2. **Karnaugh-Map Technique**
3. **Quine-McCluskey Method**

Karnaugh-Map Technique:

- **It is the simplest and most commonly used method.**
- **It is a manual method and depends to a great extent upon human intuition.**
- **It can be used conveniently up to six variables beyond which it is very cumbersome.**

Quine-McCluskey Method:

- **It is suitable for computer mechanization and is basically used by logic designers manually.**

Karnaugh Map Representation of Logical Functions

Standard Representations for Logical Functions:

Arbitrary logic function can be expressed in following terms:

1. Sum of Products form (SOP) and

2. Product of Sums form (POS)

This does not mean that logic functions cannot be written in any other form. But these two forms are standard methods for designing the circuits.

e.g.

Sum of Product can be represented as:

$$Y = \sum m(0, 3, 6, 7, 10, 12, 15)$$

$$Y = \prod M(1, 2, 4, 5, 8, 9, 11, 13, 14)$$

Karnaugh Map Representation of Logical Functions

- Some times it is difficult to be sure that a logical expression can be simplified using algebraic method of simplification using boolean algebraic theorms directly.
- There is a graphical Technique known as karnaugh map technique which provides a systematic method for simplifying and manipulating boolean expressions.
- In this technique, the information contained in a truth table or available in POS or SOP or form is represented on Karnaugh map (K-map).
- It can be used for any number of variables, but it is used upto six variables, beyound which it becomes very cumbersome.
- There are K-maps for two, three and four variables. Thus in an n variable K-map there are 2^n cells. Each cell corresponds to one of the combinations of n variables, since there are 2^n combinations of n variables.

Karnaugh Map Representation of Logical Functions

- Truth Table of a 3-variable function:

Row no.	Inputs			Output			
	A	B	C	Y			
0	0	0	0	0			
1	0	0	1	1			
2	0	1	0	1			
3	0	1	1	0			
4	1	0	0	1			
5	1	0	1	0			
6	1	1	0	-	-	-	-
7	1	1	1	1			

Karnaugh Map Representation of Logical Functions

- In above truth table logical 0 corresponds to the rows 0,3,5 & 6 and the output Y can be represented in terms of standard POS form as given below:
- $Y = (A+B+C)(A+B+C)(A+B+C)(A+B+C) - -$

Karnaugh-Map Technique

The K-map technique is not suitable for handling the design of complex digital systems because of the following disadvantages:

- 1. Minimization of logic functions involving more than six variables is unwieldy.**
- 2. Recognition of prime- implicants that may form part of the simplified function relies on the ability of the human user making it difficult to be sure whether the best selection has been made.**

Quine-McCluskey Method

The Quine-McCluskey minimization technique satisfies the following characteristics:

1. It has the capability of handling large number of variables.
2. It does not depend on the ability of a human user for recognising prime-implicants.
3. It ensures minimized expression.
4. It is suitable for computer solution.

Thus Quine-McCluskey method consists of two parts:

1. To find by an exhaustive search all the prime-implicants that may form part of the simplified function.
2. To identify essential prime-implicants obtained from part 1 and choose among the remaining prime-implicants those that give an expression with the least number of literals. This method is also known as tabular method.

Quine-McCluskey Method

E.g. $Y(A,B,C,D) = \sum m(0,1,3,7,8,9,11,15)$

Step 1. Check whether the logic function to be minimized is in the minterm form, if not then convert it in minterm form.

Step 2. Arrange all minterms of the function in binary representation form in a table according to the number of ones contained and form the groups containing no ones, 1 ones, 2 ones, 3 ones and so on.

The groups are separated by horizontal lines.

Step 3. Apply boolean algebraic theorem $A + A = 1$ to pairs of minterms in which only one variable is different and all the other variables are same. This relationship is applicable only to the minterms belonging to adjacent groups of minterms. Compare each minterm in group n with each minterm in group

Quine-McCluskey Method

Step 4. Compare the adjacent groups to see if groups of four can be made by matching. For this match the dashes must be in the same bit position in the groups of two and only one variable must differ.

Step 5. Repeat the process of grouping of eight minterms.

Step 6. All nonchecked minterm groups in tables are the prime implicants of the function.

Step 7. Next a prime-implicant table is prepared listing each of the minterms contained in the original function, PI terms, and the decimal numbers of minterms that make up the PI. A cross is marked in the table in each row under the minterms contained in that PI. Now find the minterms that contain only

Quine-McCluskey Method

E.g. $Y(A,B,C,D) = \sum m(0,1,3,7,8,9,11,15)$

Table 1. Grouping minterms according to the number of 1's.

Group	Minterm	Variables	Steps Checked
		A B C D	
0	0	0 0 0 0	Checked
1	1	0 0 0 1	Checked
	8	1 0 0 0	Checked
2	3	0 0 1 1	Checked
	9	1 0 0 1	Checked

Quine-McCluskey Method

E.g. $Y(A,B,C,D) = \sum m(0,1,3,7,8,9,11,15)$

Table 2. Combination of minterms groups of two.

Group	Minterm	Variables	Steps Checked
A B C D			
0	0,1	0 0 0 -	Checked
	0,8	- 0 0 0	Checked
1	1,3	0 0 - 1	Checked
	1,9	- 0 0 1	Checked
	8,9	1 0 0 -	Checked
2	3,7	0 - 1 1	Checked

Quine-McCluskey Method

E.g. $Y(A,B,C,D) = \sum m(0,1,3,7,8,9,11,15)$

Table 3. Combination of minterms groups of four.

Group	Minterm	Variables				Steps	Checked
		A	B	C	D		
0	0,1,8,9	-	0	0	-		Checked
	0,8,1,9	-	0	0	-		Checked
1	1,3,9,11	-	0	-	1		Checked
	1,9,3,11	-	0	-	1		Checked
2	3,7,11,15	-	-	1	1		Checked

Quine-McCluskey Method

E.g. $Y(A,B,C,D) = \sum m(0,1,3,7,8,9,11,15)$

Table 4. Prime Implicant Table (Essential PI)

PI Terms	Decimal Nos.	Minterms							
<div> <div>— —</div> <div>—</div> </div>		0	1	3	7	8	9	11	15
BC	0,1,8,9			X	X			X	X
BD	1,3,9,11					X	X		X
CD	3,7,11,15			X	X			X	X

E.g.

Minimize the four-variable logic function using k-map:

$$f(A,B,C,D) = \sum m(0,1,2,3,5,7,8,9,11,14)$$

Q.1) a) Quine McCluskey method. $f(A, B, C, D) = \sum m(0,1,2,3,5,7,8,10,12,13,15)$ [6 marks]

Solution: $\sum m$ (0000,0001,0010,0011,0101,0111,1000,1010,1100,1101,1111) in binary form.

First List					
	A	B	C	D	
0	0	0	0	0	✓
1	0	0	0	1	✓
2	0	0	1	0	✓
8	1	0	0	0	✓
3	0	0	1	1	✓
5	0	1	0	1	✓
10	1	0	1	0	✓
12	1	1	0	0	✓
7	0	1	1	1	✓
13	1	1	1	1	✓
15	1	1	1	1	✓

Second List					
	A	B	C	D	
0,1	0	0	0	-	✓
0,2	0	0	-	0	✓
0,8	-	0	0	0	✓
1,3	0	0	-	1	✓
1,5	0	-	0	1	✓
2,3	0	0	1	-	✓
2,10	-	0	1	0	✓
8,10	1	0	-	0	✓
8,12	1	-	0	0	$\bar{A} \bar{C} \bar{D}$
3,7	0	-	1	1	✓
5,7	0	1	-	1	✓
5,13	-	1	0	1	✓
12,13	1	1	0	-	$A B \bar{C}$
7,15	-	1	1	1	✓
13,15	1	1	-	1	✓

Third List					
	A	B	C	D	
0, 1 2 3	0	0	-	-	$\bar{A} \bar{B}$
0, 2 1 3	0	0	-	-	
0, 2, 8, 10	-	0	-	0	$\bar{B} \bar{D}$
0, 8, 2, 10	-	0	-	0	
1, 3, 5, 7	0	-	-	1	$\bar{A} D$
1, 5, 3, 7	0	-	-	1	
5, 7, 13, 15	-	1	-	1	$B D$
5, 13, 7, 15	-	1	-	1	

The prime implicants are: $\bar{A} \bar{B} + \bar{B} \bar{D} + \bar{A} D + B D + A \bar{C} \bar{D} + A B \bar{C}$

Quine-McCluskey Method

	0	1	2	3	5	7	8	10	12	13	15
$\overline{A}\overline{B}$	*	*	*	*							
$\overline{B}\overline{D}$	*		*				*	(*)			
$A\overline{D}$		*		*	*	*					
$B\overline{D}$					*	*				*	(*)
$A\overline{C}\overline{D}$							*		*		
$A\overline{B}C$									*	*	
Essential	X		X		X	X	X	(X)		X	(X)

Thus, one minimal solution is: $Z = \overline{B}\overline{D} + BD + \overline{A}\overline{B} + A\overline{C}\overline{D}$

Quine-McCluskey Method

E.g.

$$f(A,B,C,D) = \sum m(1,3,7,11,15) + d(0,2,5)$$

Definations

Natural BCD Code: Decimal digits 0 through 9 are represented by their natural binary equivalents using 4 bits.

Excess 3 code: Another BCD code in 4 bit binary code. The decimal digit is obtained by adding 3 to the natural BCD code.

Gray code: It is a code in which each gray code number differs from the preceding & succeeding number by a single bit.

Construction of Gray Code:

1. A 1 bit gray code has two code words 0 and 1 representing decimal numbers 0 & 1 respectively.
2. An n bit ($n \geq 2$) gray code will have first 2^{n-1} gray codes of $(n-1)$ bits written

Continue....

E.g. :

1. 1 bit gray code

Decimal number

Gray Code

0

0

1

1

2. 2 bit gray code

Decimal number

Gray Code

0

0

0

1

0

1

2

1

1

3

1

0

3. 3 bit Gray Code

Decimal number

Gray Code

0

0

0

0

1

0

0

1

End
of
Unit-1

Thank You