

# Digital Fundamentals



Sub Code : 3130704

AS PER NEW SYLLABUS OF  
GURU NANAK DEV TECHNOLOGICAL UNIVERSITY

- EASY LEARNING
- SIMPLIFIED APPROACH
- CHAPTERWISE SOLVED GTU QUESTIONS  
WINTER 2011 to WINTER 2018
- ORAL QUESTIONS & ANSWERS

**MODEL PAPER**

AS PER NEW QUESTION PAPER PATTERN

 **TECHNICAL**  
**PUBLICATIONS**™

An Up-Thrust for Knowledge

**A. P. Godse**  
**Dr. D. A. Godse**

# TABLE OF CONTENTS

<b>Chapter - 1 Fundamentals of Digital Systems and Logic Families (1 - 1) to (1 - 74)</b>	
1.1 Digital signals .....	1 - 2
1.2 Number Systems .....	1 - 3
1.3 Representation of Numbers of Different Radix .....	1 - 4
1.3.1 Decimal Number System .....	1 - 4
1.3.2 Binary Number System .....	1 - 5
1.3.3 Octal Number System .....	1 - 5
1.3.4 Hexadecimal Number System .....	1 - 6
1.3.5 Usefulness of Octal and Hexadecimal System .....	1 - 6
1.3.6 Format of a Binary Number .....	1 - 7
1.3.7 Counting in Radix (Base) $r$ .....	1 - 7
1.4 Conversion of Numbers from One Radix to Another Radix .....	1 - 8
1.4.1 Binary to Octal Conversion .....	1 - 8
1.4.2 Octal to Binary Conversion .....	1 - 9
1.4.3 Binary to Hexadecimal Conversion .....	1 - 9
1.4.4 Hexadecimal to Binary Conversion .....	1 - 10
1.4.5 Octal to Hexadecimal Conversion .....	1 - 10
1.4.6 Hexadecimal to Octal Conversion .....	1 - 11
1.4.7 Converting any Radix to Decimal .....	1 - 11
1.4.8 Conversion of Decimal Number to Any Radix Number .....	1 - 12
1.5 Complement of Numbers .....	1 - 23
1.5.1's Complement Representation .....	1 - 23
1.5.2 2's Complement Representation .....	1 - 23
1.6 Signed Binary Numbers .....	1 - 24

1.7 Binary Arithmetic .....	1-26
1.7.1 Binary Addition .....	1-26
1.7.2 Binary Subtraction .....	1-28
1.7.3 Binary Subtraction using 1's Complement Method .....	1-28
1.7.4 Binary Subtraction using 2's Complement Method .....	1-29
1.8 Octal Arithmetic .....	1-34
1.8.1 Octal Addition .....	1-34
1.8.2 Octal Subtraction using 8's Complement .....	1-35
1.9 Hexadecimal Arithmetic .....	1-37
1.9.1 Hexadecimal Addition .....	1-37
1.9.2 Hexadecimal Subtraction using 16's Complement .....	1-39
1.10 Binary Codes .....	1-40
1.10.1 Classification of Binary Codes .....	1-41
1.10.2 BCD (Binary Coded Decimal) Codes .....	1-43
1.10.2.1 BCD Addition .....	1-44
1.10.2.2 BCD Subtraction using 9's Complement Method .....	1-48
1.10.2.3 BCD Subtraction using 10's Complement Method .....	1-50
1.10.3 Other 4-bit BCD Codes .....	1-52
1.10.3.1 2-4-2-1 Codes .....	1-52
1.10.3.2 Other Weighted BCD Codes .....	1-53
1.10.4 Excess-3 Code .....	1-54
1.10.4.1 Excess-3 Addition .....	1-56
1.10.4.2 Excess-3 Subtraction .....	1-56
1.10.5 Gray Code .....	1-58
1.10.5.1 Application of Gray Code .....	1-58
1.10.5.2 Gray to Binary Conversion .....	1-60
1.10.5.3 Binary to Gray Code Conversion .....	1-61
1.10.6 Alphanumeric Codes .....	1-62
1.10.6.1 ASCII .....	1-62
1.11 Error Detecting and Correcting Codes .....	1-65
1.11.1 Parity Bit .....	1-65

1.11.2 Hamming Code .....	1-66
1.11.3 Detecting and Correcting an Error .....	1-70
1.11.4 Single Error Correction Plus Double Error Detection .....	1-72
Oral Questions and Answers .....	1-73
<b>Chapter - 2 Logic Gates and Boolean Algebra</b> .....	<b>(2 - 1) to (2 - 20)</b>
2.1 Logic Gates .....	2-2
2.1.1 Positive and Negative Logic .....	2-5
2.2 Boolean Algebra .....	2-6
2.2.1 Boolean Algebra Terminology .....	2-7
2.2.2 Axiomatic Definitions of Boolean Algebra .....	2-7
2.2.3 Basic Theorems and Properties of Boolean Algebra .....	2-9
2.2.4 DeMorgan's Theorems .....	2-10
2.2.5 Principle of Duality .....	2-11
Oral Questions and Answers .....	2-18
<b>Chapter - 3 Digital Logic Families</b> .....	<b>(3 - 1) to (3 - 44)</b>
3.1 Introduction .....	3-2
3.2 Characteristics of Digital ICs .....	3-3
3.3 Transistor-Transistor Logic (TTL) .....	3-9
3.3.1 2-Input TTL NAND Gates .....	3-9
3.3.2 3-Input TTL NAND Gate .....	3-10
3.3.3 Totem-Pole Output / Active Pull-Up .....	3-11
3.3.4 Input and Output Currents .....	3-14
3.3.5 Wired Logic - Open Collector Output .....	3-18
3.3.6 Comparison between Totem-Pole and Open-Collector Outputs .....	3-20
3.3.7 Standard TTL Characteristics .....	3-21
3.3.8 Advantages and Disadvantages of TTL Family .....	3-23
3.3.9 Unconnected Inputs .....	3-23
3.3.10 Tri-State TTL Inverter .....	3-24
3.3.11 Schottky TTL .....	3-25

3.4 CMOS Logic .....	3 - 27
3.4.1 CMOS Inverter.....	3 - 27
3.4.2 CMOS NAND Gate.....	3 - 29
3.4.3 CMOS NOR Gate .....	3 - 32
3.4.4 CMOS Characteristics .....	3 - 33
3.4.5 Wired Logic .....	3 - 35
3.4.6 Open Drain Outputs .....	3 - 36
3.4.7 Advantages and Disadvantages of CMOS Family .....	3 - 36
3.5 Interfacing of CMOS to TTL and TTL to CMOS .....	3 - 37
3.5.1 TTL Driving CMOS .....	3 - 38
3.5.2 CMOS Driving TTL .....	3 - 39
3.6 Comparison between TTL and CMOS .....	3 - 41
Oral Questions and Answers .....	3 - 42

**Chapter - 4 Combinational Digital Circuits (4 - 1) to (4 - 188)**

4.1 Standard Representation for Logic Functions .....	4 - 3
4.1.1 Sum of Product Form .....	4 - 4
4.1.2 Product of Sum Form .....	4 - 5
4.1.3 Standard SOP and Standard POS Forms .....	4 - 5
4.1.4 Converting Expressions in Standard SOP or POS Form .....	4 - 6
4.1.4.1 Steps to Convert SOP to Standard SOP Form .....	4 - 6
4.1.4.2 Steps to Convert POS to Standard POS Form .....	4 - 8
4.1.5 M Notations : Minterms and Maxterms .....	4 - 10
4.1.6 Complements of Standard Forms .....	4 - 12
4.2 Karnaugh-Maps (K-Map) Representation .....	4 - 15
4.2.1 One-Variable, Two-Variable, Three-Variable and Four-Variable Maps .....	4 - 15
4.2.2 Plotting a Karnaugh Map .....	4 - 18
4.2.2.1 Representation of Truth Table on Karnaugh Map .....	4 - 18
4.2.2.2 Representing Standard SOP on K-Map .....	4 - 19
4.2.2.3 Representing Standard POS on K-Map .....	4 - 20

4.2.3 Grouping Cells for Simplification .....	4 - 21
4.2.3.1 Grouping Two Adjacent Ones (Pair) .....	4 - 22
4.2.3.2 Grouping Four Adjacent Ones (Quad) .....	4 - 24
4.2.3.3 Grouping Eight Adjacent Ones (Octet) .....	4 - 25
4.2.4 Illegal Grouping .....	4 - 26
4.3 Simplification SOP Functions using K-Maps .....	4 - 27
4.3.1 Essential Prime Implicants .....	4 - 35
4.3.2 Incompletely Specified Functions (Don't Care Terms) .....	4 - 35
4.3.2.1 Describing Incomplete Boolean Function .....	4 - 35
4.3.2.2 Don't Care Conditions in Logic Design .....	4 - 36
4.3.2.3 Minimization of Incompletely Specified Functions .....	4 - 36
4.4 Simplification of POS Functions using K-Maps .....	4 - 41
4.5 Summary of Rules for K-Map Simplification .....	4 - 45
4.6 Limitation of Karnaugh Map .....	4 - 46
4.7 Realizing Logic Function with Gates .....	4 - 46
4.7.1 Implementation of SOP Boolean Expression .....	4 - 46
4.7.2 Implementation of POS Boolean Expression .....	4 - 47
4.7.3 Universal Gates .....	4 - 52
4.7.3.1 NAND Gate .....	4 - 52
4.7.3.2 NOR Gate .....	4 - 54
4.7.4 NAND-NAND Implementation .....	4 - 57
4.7.5 NOR-NOR Implementation .....	4 - 62
4.7.6 The EX-OR Gate in Function Realization .....	4 - 70
4.8 Combinational Design Examples .....	4 - 72
4.8.1 Introduction .....	4 - 72
4.8.2 Design Procedure .....	4 - 73
4.8.3 A Combinational Function Generator .....	4 - 73
4.9 Quine McCluskey Method of Function Realization .....	4 - 80
4.9.1 Algorithm for Generating Prime Implicants .....	4 - 80
4.9.2 Quine McCluskey using Don't Care Terms .....	4 - 84

4.9.3 Prime Implicant Table and Redundant Prime Implicants .....	4 - 87
4.9.4 Advantages and Disadvantages of Quine McCluskey Method .....	4 - 90
<b>4.10 Multiplexers.....</b>	<b>4 - 91</b>
4.10.1 2 : 1 Multiplexer .....	4 - 92
4.10.2 4 : 1 Multiplexer .....	4 - 93
4.10.3 8 : 1 Multiplexer .....	4 - 93
4.10.4 Quadruple 2 to 1 Multiplexer .....	4 - 94
4.10.5 The 74151 Multiplexer .....	4 - 95
4.10.6 The 740X153 Dual 4 to 1 Multiplexer .....	4 - 96
4.10.7 Expanding Multiplexers .....	4 - 96
4.10.8 Implementation of Combinational Logic using MUX .....	4 - 98
4.10.9 Implementation of Logic Function using IC 74153 .....	4 - 103
4.10.10 Applications of Multiplexer .....	4 - 107
4.10.11 Multiplexer ICs .....	4 - 107
<b>4.11 Demultiplexers.....</b>	<b>4 - 108</b>
4.11.1 Types of Demultiplexers .....	4 - 109
4.11.1.1 1 : 4 Demultiplexer .....	4 - 109
4.11.1.2 1 : 8 Demultiplexer .....	4 - 109
4.11.2 Expanding Demultiplexers .....	4 - 111
4.11.3 Implementation of Combinational Logic using Demultiplexer .....	4 - 113
4.11.4 Applications of Demultiplexer .....	4 - 114
4.11.5 Demultiplexer ICs .....	4 - 115
<b>4.12 Decoders .....</b>	<b>4 - 115</b>
4.12.1 Binary Decoder .....	4 - 115
4.12.2 The 74X138 3-to-8 Decoder .....	4 - 117
4.12.3 Expanding Cascading Decoders .....	4 - 118
4.12.4 Realization of Boolean Function using Decoder .....	4 - 120
4.12.5 Implementation of Logic Function using IC 74138 .....	4 - 124
4.12.6 Applications of Decoder .....	4 - 124
4.12.7 Decoder ICs .....	4 - 125

4.12.8 Seven Segment Decoder .....	4 - 125
4.12.8.1 Basic Connection for Driving 7-Segment Displays .....	4 - 128
4.12.8.2 IC 7446A, 7447A and 74LS47 .....	4 - 130
4.12.8.3 IC 74X49 Seven Segment Decoder .....	4 - 136
<b>4.13 Encoders .....</b>	<b>4 - 138</b>
4.13.1 Decimal to BCD Encoder .....	4 - 138
4.13.2 Priority Encoder .....	4 - 140
4.13.3 Priority Encoder IC (74XX148) .....	4 - 141
4.13.4 Octal to Binary Encoder .....	4 - 142
4.13.5 Encoder ICs .....	4 - 143
<b>4.14 Adders .....</b>	<b>4 - 143</b>
4.14.1 Half Adder .....	4 - 144
4.14.2 Full Adder .....	4 - 145
<b>4.15 Subtractors .....</b>	<b>4 - 147</b>
4.15.1 Half Subtractor .....	4 - 147
4.15.2 Full-Subtractor .....	4 - 148
<b>4.16 BCD Arithmetic .....</b>	<b>4 - 151</b>
<b>4.17 Carry Look Ahead Adder .....</b>	<b>4 - 153</b>
<b>4.18 Digital Comparator .....</b>	<b>4 - 154</b>
<b>4.19 Parity Generator / Checker .....</b>	<b>4 - 160</b>
<b>4.20 Code Converters .....</b>	<b>4 - 166</b>
<b>4.21 ALU and Elementary ALU Design .....</b>	<b>4 - 177</b>
4.21.1 Design of Arithmetic Unit .....	4 - 177
4.21.2 Design of Logic Circuit .....	4 - 181
4.21.3 Combining Arithmetic and Logic Unit .....	4 - 182
<b>Oral Questions and Answers .....</b>	<b>4 - 185</b>
<b>Chapter - 5 Sequential Circuits and Systems</b>	<b>(5 - 1) to (5 - 148)</b>
<b>5.1 Introduction .....</b>	<b>5 - 2</b>

5.1.1 Comparison between Combinational and Sequential Logic Circuits.....	5 - 2
5.1.2 Clock.....	5 - 3
<b>5.2 One-Bit Memory and the Circuit Properties of Bistable Latch.....</b>	<b>5 - 4</b>
<b>5.3 Latches.....</b>	<b>5 - 5</b>
5.3.1 SR Latch.....	5 - 6
5.3.2 Gated SR Latch.....	5 - 7
5.3.3 Gated D Latch.....	5 - 9
<b>5.4 Clocked Flip-Flops.....</b>	<b>5 - 9</b>
5.4.1 Latches Vs Flip-Flops.....	5 - 9
5.4.2 Level and Edge Triggering.....	5 - 10
5.4.3 SR Flip-Flop.....	5 - 13
5.4.4 D Flip-Flop.....	5 - 15
5.4.5 JK Flip-Flop.....	5 - 17
5.4.5.1 JK Flip-Flop using NAND Gates .....	5 - 18
5.4.5.2 Race-around Condition .....	5 - 19
5.4.6 Master-Slave SR Flip-Flop.....	5 - 20
5.4.7 Master-Slave JK Flip-Flop.....	5 - 21
5.4.8 T Flip-Flop.....	1 - 22
5.4.9 Preset and Clear.....	1 - 29
<b>5.5 Excitation Tables.....</b>	<b>1 - 29</b>
5.5.1 SR Flip-Flop.....	1 - 29
5.5.2 JK Flip-Flop.....	1 - 30
5.5.3 D Flip-Flop.....	1 - 31
5.5.4 T Flip-Flop.....	1 - 31
<b>5.6 Conversion from One Type to another Type of Flip-Flop.....</b>	<b>1 - 31</b>
5.6.1 SR Flip-Flop to D Flip-Flop.....	5 - 32
5.6.2 SR Flip-Flop to JK Flip-Flop .....	5 - 32
5.6.3 SR Flip-Flop to T Flip-Flop.....	5 - 33
5.6.4 JK Flip-Flop (OR MSJK) to T Flip-Flop .....	5 - 34
5.6.5 JK Flip-Flop (OR MSJK) to D Flip-Flop .....	5 - 35

5.6.6 D Flip-Flop to T Flip-Flop .....	5 - 35
5.6.7 T Flip-Flop to D Flip-Flop .....	5 - 37
5.6.8 JK Flip-Flop (OR MSJK) to SR Flip-Flop .....	5 - 37
5.6.9 D Flip-Flop to SR Flip-Flop .....	5 - 38
5.6.10 T Flip-Flop to SR Flip-Flop .....	5 - 39
5.6.11 D Flip-Flop to JK Flip-Flop .....	5 - 39
<b>5.7 Applications of Flip-Flops .....</b>	<b>5 - 40</b>
5.7.1 Bounce Elimination Switch .....	5 - 41
5.7.2 Registers .....	5 - 42
5.7.3 Counters .....	5 - 43
5.7.4 Frequency Division .....	5 - 44
5.7.5 Clock System with Delayed Clock Signal .....	5 - 45
5.7.6 A Two Phase Clock Circuit .....	5 - 46
<b>5.8 Registers and Shift Registers .....</b>	<b>5 - 47</b>
5.8.1 Shift Registers .....	5 - 48
5.8.2 Types of Shift Registers .....	5 - 49
5.8.2.1 Serial In Serial Out (SISO) Shift Register .....	5 - 49
5.8.2.2 Serial In Parallel Out (SPIO) Shift Register .....	5 - 54
5.8.2.3 Parallel In Serial Out (PISO) Shift Register .....	5 - 54
5.8.2.4 Parallel In Parallel Out (PIPO) Shift Register .....	5 - 55
5.8.2.5 Bidirectional Shift Register .....	5 - 55
5.8.3 Universal Shift Register .....	5 - 57
5.8.4 Applications of Shift Registers .....	5 - 58
5.8.4.1 Delay Line .....	5 - 58
5.8.4.2 Serial-to-Parallel Converter .....	5 - 59
5.8.4.3 Parallel-to-Serial Converter .....	5 - 59
5.8.4.4 Shift Register Counters .....	5 - 59
5.8.4.5 Pseudo-Random Binary Sequence (PRBS) Generator .....	5 - 60
5.8.4.6 Sequence Generator .....	5 - 60
5.8.4.7 Sequence Detector .....	5 - 60

5.9	Counters.....	5 - 62
5.9.1	Ripple / Asynchronous Counters .....	5 - 64
5.9.1.1	Asynchronous / Ripple Down Counter .....	5 - 68
5.9.1.2	Asynchronous Up / Down Counter .....	5 - 70
5.9.1.3	Decoding Gates .....	5 - 73
5.9.1.4	Problem Faced by Ripple Counters (Glitch Problem) .....	5 - 74
5.9.2	Design of Ripple (Asynchronous) Counters .....	5 - 79
5.9.3	Synchronous Counters .....	5 - 79
5.9.3.1	2-bit Synchronous Binary Up Counter .....	5 - 79
5.9.3.2	3-bit Synchronous Binary Up Counter .....	5 - 80
5.9.3.3	4-bit Synchronous Binary Up Counter .....	5 - 81
5.9.3.4	Synchronous Down and Up / Down Counters .....	5 - 83
5.9.4	Design of Synchronous Counters .....	5 - 83
5.9.5	Presettable Counters .....	5 - 95
5.9.6	Design of Skipping State Counter .....	5 - 96
5.10	Shift Register Counters .....	5 - 103
5.10.1	Ring Counter .....	5 - 103
5.10.2	Johnson or Twisting Ring or Switch Tail Counter .....	5 - 104
5.11	Sequence Generator .....	5 - 107
5.11.1	Sequence Generator using Counters .....	5 - 107
5.11.2	Sequence Generator using Shift Register .....	5 - 124
5.12	Design with Counter ICs .....	5 - 132
5.12.1	IC 7490 (Decade Binary Counter) .....	5 - 132
5.12.2	IC 7493 (4-bit Ripple Counter) .....	5 - 136
5.12.3	IC 74192/74193 (Up/Down - BCD/Binary Counters) .....	5 - 139
	Oral Questions and Answers .....	5 - 145
<b>Chapter - 6</b>	<b>A / D and D / A Converters</b>	<b>(6 - 1) to (6 - 50)</b>
6.1	Introduction .....	6 - 2
6.2	D/A Converters .....	6 - 3
6.2.1	Performance Parameters (Specifications) of DAC .....	6 - 4

6.2.2	Basic Conversion Techniques .....	6 - 7
6.2.2.1	Binary Weighted Resistor D/A Converter .....	6 - 8
6.2.2.2	R/2R Ladder D/A Converter .....	6 - 9
6.3	D/A Converter IC .....	6 - 16
6.4	Applications of DAC .....	6 - 20
6.5	Sample and Hold Circuit .....	6 - 20
6.5.1	Basic Sample and Hold Circuit .....	6 - 22
6.5.2	Performance Parameters of S/H Circuits .....	6 - 22
6.5.3	Advantages of Sample and Hold Circuits .....	6 - 24
6.5.4	Applications of Sample and Hold circuits .....	6 - 25
6.6	A/D Converters .....	6 - 25
6.7	Performance Parameters (Specifications of ADC) .....	6 - 26
6.8	Types of A/D Converters .....	6 - 31
6.8.1	Dual Slope ADC .....	6 - 31
6.8.2	Successive Approximation ADC .....	6 - 34
6.8.3	Flash ADC (Parallel Comparators) .....	6 - 37
6.8.4	Comparison between ADCs .....	6 - 38
6.8.5	Counter Type A/D Converter .....	6 - 39
6.8.6	Tracking Type Continuous A/D Conversion .....	6 - 41
6.9	A/D Converter using V/F Converter .....	6 - 43
6.10	A/D Converter using V/T Conversion .....	6 - 44
6.11	Example of A/D Converter IC .....	6 - 45
6.11.1	Features .....	6 - 45
6.11.2	Pin Diagram .....	6 - 45
6.11.3	Operation .....	6 - 45
6.11.4	Interfacing .....	6 - 46

<b>Chapter - 7 Semiconductor Memories and Programmable Logic Devices</b>	<b>(7 - 1) to (7 - 76)</b>
7.1 Introduction .....	7 - 2
7.1.1 Computer Memory .....	7 - 3
7.2 Characteristics of Memories .....	7 - 4
7.3 Classification of Memories .....	7 - 6
7.4 Serial (Sequential) Random Access Memories .....	7 - 8
7.5 Read Only Memory (ROM) .....	7 - 9
7.5.1 PROM (Programmable Read Only Memory) .....	7 - 9
7.5.2 EPROM (Erasable Programmable Read Only Memory) .....	7 - 11
7.5.3 EEPROM (Electrically Erasable Programmable Read Only Memory) .....	7 - 11
7.5.4 EEPROM .....	7 - 12
7.6 Read and Write Memory (RAM) .....	7 - 13
7.6.1 Static RAM (SRAM) .....	7 - 13
7.6.1.1 Static RAM Cell .....	7 - 13
7.6.2 Dynamic RAM (DRAM) .....	7 - 14
7.6.2.1 Dynamic RAM Cell .....	7 - 14
7.6.2.2 Comparison between SRAM and DRAM .....	7 - 14
7.6.3 Comparison between RAM and ROM .....	7 - 15
7.7 Memory Organization .....	7 - 15
7.8 Expanding Memory Size .....	7 - 18
7.9 Content Addressable Memory (CAM) .....	7 - 23
7.9.1 Hardware Organization .....	7 - 23
7.9.2 Read Operation .....	7 - 26
7.9.3 Write Operation .....	7 - 26
7.10 Charge - Coupled Devices (CCD) Memory .....	7 - 27
7.11 Programmable Logic Devices .....	7 - 28
7.12 ROM / PROM as a PLD .....	7 - 29
7.12.1 AND Matrix .....	7 - 31

7.12.2 OR Matrix .....	7 - 33
7.12.3 Invert / Non-invert Matrix .....	7 - 34
7.12.4 Combinational Logic Implementation using PROM .....	7 - 34
7.13 Programmable Logic Array (PLA) .....	7 - 41
7.13.1 Input Buffer .....	7 - 42
7.13.2 Output Buffer .....	7 - 42
7.13.3 Output through Flip-Flops .....	7 - 43
7.13.4 Implementation of Combination Logic Circuit using PLA .....	7 - 43
7.14 Programmable Array Logic (PAL) .....	7 - 54
7.14.1 Implementation of Combinational Logic Circuit using PAL .....	7 - 55
7.15 Comparison between PROM, PLA and PAL .....	7 - 63
7.16 Complex Programmable Logic Devices (CPLDs) .....	7 - 63
7.16.1 Block Diagram .....	7 - 64
7.16.2 Architecture of XC9572 CPLD .....	7 - 65
7.17 Field Programmable Gate Array (FPGA) .....	7 - 67
7.18 Comparison between CPLDs and FPGAs .....	7 - 71
Oral Questions and Answers .....	7 - 72

# 1

## Fundamentals of Digital Systems and Logic Families

### Contents

1.1	Digital Signals	
1.2	Number Systems	
1.3	Representation of Numbers of Different Radix	
1.4	Conversion of Numbers from One Radix to Another Radix	Dec.-12, 13, May-12, 13, 14, Winter-15, 16, 17, Summer-16, 17, 18, Marks 7
1.5	Complement of Numbers	
1.6	Signed Binary Numbers	
1.7	Binary Arithmetic	May-13, Summer-15, Winter-17, Marks 7
1.8	Octal Arithmetic	Summer-15, Marks 1
1.9	Hexadecimal Arithmetic	Summer-16, Marks 4
1.10	Binary Codes	May-14, Summer-15, 16, 18, Winter-16, Marks 7
1.11	Error Detecting and Correcting Codes	
	Oral Questions and Answers	

### 1.1 Digital Signals

#### Analog versus Digital

Analog systems process information that varies continuously i.e. they process time varying signals that can take on any value across a continuous range of voltage, current or any physical parameter. On the other hand, the digital systems use discrete quantities to represent information. Discrete means distinct or separated as opposed to continuous or connected. The Fig. 1.1.1 shows the difference between two signals : continuous and discrete.

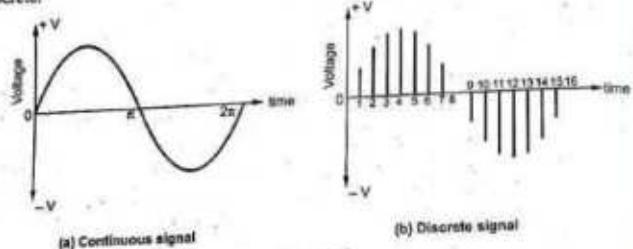


Fig. 1.1.1

**Digital / discrete signal :** A digital/discrete signal is a signal that can have one of a finite set of possible values at any time.

**Analog / continuous signal :** An analog/continuous signal is a signal that can have one of an infinite number of possible values.

**Binary signal :** In digital systems, the most common digital signal is the signal that has one of two possible values, like on or off (often represented as 1 or 0). Such signal is known as binary signal. In digital systems number formed by binary values are used to represent the levels of discrete signal.

The reasons for widespread use of digital systems are :

1. They are easy to design.
2. They are more flexible in design.
3. They provide higher accuracy.

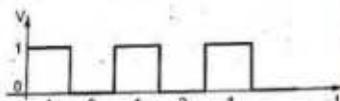


Fig. 1.1.2 Binary signal

4. They are less affected by component aging and noise.
5. They are also less affected by variations in power-supply voltage and temperature.
6. They can have memory and hence they are highly suitable for computers, calculators, watches, etc.
7. It is easy to provide added functionality in digital systems than analog systems.
8. They always produce exactly same results with same set of inputs and circuit components.
9. In digital systems, display of information is very convenient, accurate and elegant.
10. They are cost effective.
11. HDL (Hardware Description Languages) are available; they make digital system design simple.
12. Communication between digital systems is much more easier.

#### Advantage of Binary

- \* The base for all digital system operations is the binary number system. Though many systems use octal, decimal and hexadecimal number system, when the data is provided to a digital system it is in the form of binary numbers. Thus, all computers represent numbers and alphabetic characters in the binary form.
- \* In binary number system there are only two logic values viz. logic 0 and logic 1. These two logic values are also known as low level logic/OFF and high level logic/ON. Thus, binary number system is easy, highly reliable and accurate.
- \* The binary number system can produce two different states easily. For example, a flip-flop has output either '1' or '0'. This makes the use of binary system a very reasonable choice, in case of various electronic circuits.
- \* Though multilevel circuits having more than two output levels have been studied since long time, binary circuits are universal in computer systems.

#### Review Questions

1. Define digital signal and analog signal.
2. Compare digital signal with analog signal.
3. What is binary signal?
4. State the advantages of digital systems/digital circuits.
5. Give the advantages of binary number system.

### 1.2 Number Systems

- \* Number system is a basis for counting various items.

- The decimal number system has 10 digits : 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9.
  - Modern computers communicate and operate with binary numbers which use only the digits 0 and 1.
  - When decimal quantities are represented in the binary form, they take more digits.
  - For large decimal numbers people have to deal with very large binary strings and therefore, they do not like working with binary numbers. This fact gave rise to three new number systems : Octal, Hexadecimal and Binary Coded Decimal (BCD).

### **Review Questions**

1. Explain various number systems.
  2. Name the number system used in computers.

### 1.3 Representation of Numbers of Different Radix

### 1.3.1 Decimal Number System

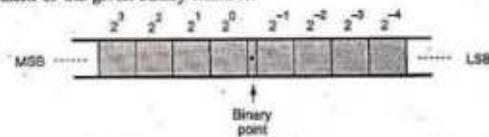
- In decimal number system we can express any decimal number in units, tens, hundreds, thousands and so on.
  - When we write a decimal number say, 5678.9, we know it can be represented as  $5000 + 600 + 70 + 8 + 0.9 = 5678.9$
  - The decimal number 5678.9 can also be written as  $5678.9_{10}$ , where the 10 subscript indicates the radix or base.
  - The position of a digit with reference to the decimal point determines its value/weight. The sum of all the digits multiplied by their weights gives the total number being represented.
  - The leftmost digit, which has the greatest weight is called the most significant digit and the rightmost digit, which has the least weight, is called the least significant digit.
  - Fig. 1.3.1 shows decimal digit and its weights expressed as a power of 10.

	$10^{-3}$	$10^{-2}$	$10^{-1}$	$10^0$	$10^1$
In power of 10	5	6	7	8	+
	$5 \times 10^3$	$6 \times 10^7$	$7 \times 10^1$	$8 \times 10^6$	+
MED					LSD

Fig. 1.3.1 Representation of a decimal number

### 1.3.2 Binary Number System

- Binary system with its two digits is a **base-two system**.
  - The two binary digits (bits) are 1 and 0.
  - In binary system, weight is expressed as a power of 2.
  - The Fig. 1.3.2 (a) shows representation of binary number 1101.101 in power of 2.
  - By adding each digit of a binary number in a power of 2 we can find the decimal equivalent of the given binary number.



**Fig. 1.3.2** Binary position values as a power of 2

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

$$N = \frac{1 \times 2^2 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0}{1 \times 2^{2+} + 0 \times 2^{2+} + 1 \times 2^{-3}} = (13.625)_{10}$$

**Fig. 1.3.2 (a)**

### 1.3.3 Octal Number System

- The octal number system uses first eight digits of decimal number system : 0, 1, 2, 3, 4, 5, 6 and 7. As it uses 8 digits, its base is 8.
  - For example, the octal number 5632.471 can be represented in power of 8 as shown in Fig. 1.3.2 (b).

$N =$	$g^3$	$g^2$	$g^1$	$g^0$	$g^{-1}$	$g^{-2}$	$g^{-3}$	
	5	6	-3	-2	-	4	7	1
	$5 \times g^3$	$6 \times g^2$	$-3 \times g^1$	$-2 \times g^0$	-	$4 \times g^{-1}$	$7 \times g^{-2}$	$1 \times g^{-3}$

$$N = 5 \times 8^3 + 5 \times 8^2 + 3 \times 8^1 + 2 \times 8^0 + 4 \times 8^{-1} + 7 \times 8^{-2} + 1 \times 8^{-3} = (2970.811328)_{10}$$

Fig. 1.3.2 (b)

- By adding each digit of an octal number in a power of 8 we can find the decimal equivalent of the given octal number.

**1.3.4 Hexadecimal Number System**

- The hexadecimal number system has a base of 16 having 16 characters: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F.
- Table 1.3.1 shows the relationship between decimal, binary, octal and hexadecimal.

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

Table 1.3.1 Relation between decimal, binary, octal and hexadecimal numbers

- For example, 3FD.84 can be represented in power of 16 as shown below.
- By adding each digit of a hexadecimal number in a power of 16 we can find decimal equivalent of the given hexadecimal number.

$$\begin{aligned}
 N &= 3 \times 16^2 + F \times 16^1 + D \times 16^0 + 8 \times 16^{-1} + 4 \times 16^{-2} \\
 N &= 3 \times 16^2 + F \times 16^1 + D \times 16^0 + 8 \times 16^{-1} + 4 \times 16^{-2} \\
 &= 3 \times 16^2 + 15 \times 16^1 + 13 \times 16^0 + 8 \times 16^{-1} + 4 \times 16^{-2} = (1021.515625)_{10}
 \end{aligned}$$

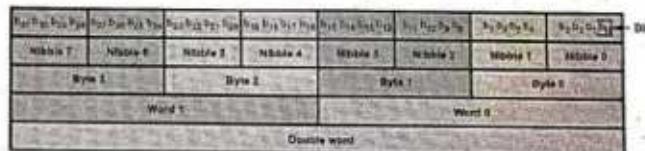
**1.3.5 Usefulness of Octal and Hexadecimal System**

- When dealing with a large quantity of binary numbers of many bits, it is convenient to use octal and hexadecimal numbers as a "shorthand" means of expressing large binary numbers.

- But it is necessary to keep in mind that the digital circuits and systems work strictly in binary; we are using octal and hexadecimal only as a convenience for the operators of the system.

**1.3.6 Format of a Binary Number**

- A single digit in the binary number is called bit.



The following figure shows the format of binary number. Four binary digits form a nibble, eight binary digits form a byte, sixteen binary digits form a word and thirty-two binary digits form a double-word.

Nibble : 4-bits can represent  $2^4 = 16$  distinct values

Byte : 8-bits can represent  $2^8 = 256$  distinct values

Word : 16-bits can represent  $2^{16} = 65536$  distinct values

Double word : 32-bits can represent  $2^{32} = 4294967296$  distinct values

**1.3.7 Counting in Radix (Base)  $r$** 

- Each number system has  $r$  set of characters. For example, in decimal number system  $r$  equals to 10 has 10 characters from 0 to 9, in binary number system  $r$  equals to 2 has 2 characters 0 and 1 and so on.
- In general we can say that, a number represented in radix  $r$ , has  $r$  characters in its set and  $r$  can be any value. This is illustrated in Table 1.3.2.

Radix (Base) $r$	Characters in set
2 (Binary)	0, 1
3	0, 1, 2
4	0, 1, 2, 3
5	0, 1, 2, 3, 4
6	0, 1, 2, 3, 4, 5
7	0, 1, 2, 3, 4, 5, 6
8 (Octal)	0, 1, 2, 3, 4, 5, 6, 7
9	0, 1, 2, 3, 4, 5, 6, 7, 8
10 (Decimal)	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
16 (Hexadecimal)	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Table 1.3.2 Radix and character set

**Example 1.3.1** Find the decimal equivalent of  $(231.23)_4$ .

$$\begin{aligned} \text{Solution : } N &= 2 \times 4^2 + 3 \times 4^1 + 1 \times 4^0 + 2 \times 4^{-1} + 3 \times 4^{-2} \\ &= 32 + 12 + 1 + 0.5 + 0.1875 \\ &= 45.6875 \end{aligned}$$

**Example 1.3.2** Count from 0 to 9 in radix 5.

**Solution :** The Table 1.2.2 indicates that radix 5 has 5 characters. A count sequence from 0 decimal to 9 decimal is

00, 01, 02, 03, 04, 10, 11, 12, 13, 14

**Example 1.3.3** What is the largest binary number that can be expressed with 12-bits? What is the equivalent decimal and hexadecimal?

**Solution :**  $(1111\ 1111\ 1111)_2$ ,  $(4095)_{10}$ ,  $(FFF)_{16}$

**1.4 Conversion of Numbers from One Radix to Another Radix**

GTU Dec.-12, 13, May-12, 13, 14, Winter-15, 16, 17, Summer-16, 17, 18

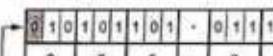
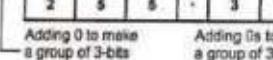
**1.4.1 Binary to Octal Conversion**

- The base for octal number is 8 and the base for binary number is 2.
- The base for octal number is the third power of the base for binary numbers. Therefore, by grouping 3 digits of binary numbers and then converting each group digit to its octal equivalent we can convert binary number to its octal equivalent.

**Example 1.4.1** Convert  $10101101.0111$  to octal equivalent.

**Solution :**

**Step 1 :** Make group of 3-bits starting from LSB for integer part and MSB for fractional part, by adding 0s at the end, if required.

Step 1.		Binary (Base 2)
Step 2.		Octal (Base 8)

Adding 0s to make a group of 3-bits      Adding 0s to make a group of 3-bits

**Step 2 :** Write equivalent octal number for each group of 3-bits.

$$(10101101.0111)_2 = (255.34)_8$$

**1.4.2 Octal to Binary Conversion**

- Conversion from octal to binary is a reversal of the process explained in the previous section. Each digit of the octal number is individually converted to its binary equivalent to get octal to binary conversion of the number.

**Example 1.4.2** Convert  $(125.62)_8$  to binary.

**Solution :**

**Step 1 :** Write equivalent 3-bit binary number for each octal digit.

**Step 2 :** Remove any leading or trailing zeros.

1	2	5	.	6	2	Octal (Base 8)
0	0	1	0	1	0	1
1	0	1	0	1	0	1

Step 1.      Step 2.      Leading zeros      Trailing zero

$$(125.62)_8 = (1010101.11001)_2$$

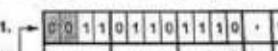
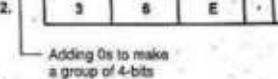
**1.4.3 Binary to Hexadecimal Conversion**

- The base for hexadecimal numbers is 16 and the base for binary numbers is 2.
- The base for hexadecimal number is the fourth power of the base for binary numbers. Therefore, by grouping 4 digits of binary numbers and then converting each group digit to its hexadecimal equivalent we can convert binary number to its hexadecimal equivalent.

**Example 1.4.3** Convert  $1101101110.1001101$  to hexadecimal equivalent.

**Solution :**

**Step 1 :** Make group of 4-bits starting from LSB for integer part and MSB for fractional part, by adding 0s at the end, if required.

Step 1.		Binary (Base 2)
Step 2.		Hexadecimal (Base 16)

Adding 0s to make a group of 4-bits      Adding 0s to make a group of 4-bits

**Step 2 :** Write equivalent hexadecimal number for each group of 4-bits.

$$(1101101110.1001101)_2 = (36E.9A)_{16}$$

**1.4.4 Hexadecimal to Binary Conversion**

- Conversion from hexadecimal to binary is a reversal of the process explained in the previous section. Each digit of the hexadecimal number is individually converted to its binary equivalent to get hexadecimal to binary conversion of the number.

**Example 1.4.4** Convert  $(8A9.B4)_{16}$  to binary.**Solution :****Step 1 :** Write equivalent 4-bit binary number of each hexadecimal digit.**Step 2 :** Remove any leading or trailing zeros.

B	A	9	.	B	4
Step 1. 1 0 0 0 1 0 1 0 1 0 0 1				1 0 1 1 0 1 0 1 0 1	
Step 2. 1 0 0 0 1 0 1 0 1 0 0 1				1 0 1 1 0 1 0 1	

Trailing zeros

$$(8A9.B4)_{16} = (1000 1010 1001.101101)_2$$

**1.4.5 Octal to Hexadecimal Conversion**

- The easiest way to convert octal number to hexadecimal number is given below.

  - Convert octal number to its binary equivalent.
  - Convert binary number to its hexadecimal equivalent.

**Example 1.4.5** Convert  $(615.25)_8$  to its hexadecimal equivalent.**Solution :****Step 1 :** Write equivalent 3-bit binary number for each octal digit.**Step 2 :** Make group of 4-bits starting from LSB for integer part and MSB for fractional part by adding 0s at the end, if required.**Step 3 :** Write equivalent octal number for each group to 4-bits.

6	7	5	.	2	5
Step 1. 1 1 0 0 0 1 1 0 1				0 1 0 1 0 1 0 1	
Step 2. 0 0 0 1 1 0 0 0 1 1 0 1				0 1 0 1 0 1 0 1 0 0	
Step 3. 1 8 D . 5 4					

Adding 0s to make a group of 4-bits      Adding 0s to make a group of 4-bits

TECHNICAL PUBLICATIONS™ An up beat for knowledge

**1.4.6 Hexadecimal to Octal Conversion**

- The easiest way to convert hexadecimal number to octal number is given below.

  - Convert hexadecimal number to its binary equivalent.
  - Convert binary number to its octal equivalent.

**Example 1.4.6** Convert  $(BC6.AF)_{16}$  to its octal equivalent.**Solution :****Step 1 :** Write equivalent 4-bit binary number for each hexadecimal digit.**Step 2 :** Make group of 3-bits starting from LSB for integer part and MSB for fractional part by adding 0s at the end, if required.**Step 3 :** Write equivalent octal number for each group of 3-bits.

B	C	6	8	.	A	F
Step 1. 1 0 1 1 1 1 0 0 0 1 1 1 0 0 0 1 1 0					1 0 1 0 1 1 1 1 1	
Step 2. 0 0 1 0 1 1 1 1 0 0 0 0 1 1 1 0 0 0 1 1 0					1 0 1 0 1 1 1 1 1 0	
Step 3. 1 3 6 1 4 8 . 5 3 6						

Adding 0s to make a group of 3-bits      Adding 0 to make a group of 3-bits

**1.4.7 Converting any Radix to Decimal**

- In general, numbers can be represented as

$$N = A_{n-1} r^{n-1} + A_{n-2} r^{n-2} + \dots + A_1 r^1 + A_0 r^0 + A_{-1} r^{-1} + A_{-2} r^{-2} + \dots C_{-m} r^{-m}$$

where  $N$  = Number in decimal $A_i$  = Digit $r$  = Radix or base of a number system $n$  = The number of digits in the integer portion of number $m$  = The number of digits in the fractional portion of number

- From this general equation we can convert number with any radix into its decimal equivalent. This is illustrated using following example.

**Example 1.4.7** Convert  $(3102.12)_4$  to its decimal equivalent.

$$\begin{aligned} \text{Solution : } N &= 3 \times 4^3 + 1 \times 4^2 + 0 \times 4^1 + 2 \times 4^0 + 1 \times 4^{-1} + 2 \times 4^{-2} \\ &= 192 + 16 + 0 + 2 + 0.25 + 0.125 = 210.375_{10} \end{aligned}$$

**Example 1.4.8** Determine the value of base  $x$ , if  $(193)_x = (623)_8$ .

$$\text{Solution : } (193)_x = (623)_8$$

TECHNICAL PUBLICATIONS™ An up beat for knowledge

Converting octal into decimal :  $6 \times 8^2 + 2 \times 8 + 3 = (403)_{10} = (623)_8$

$$\begin{aligned} A. \quad (193)_8 &= 1 \times x^2 + 9 \times x + 3 \times x^0 = (403)_{10} \\ A. \quad x^2 + 9x + 3 &= 403 \quad \therefore x = 16 \text{ or } x = -25 \\ \text{Negative is not applicable} & \quad \therefore x = 16 \\ A. \quad (193)_{10} &= (623)_8 \end{aligned}$$

#### 1.4.8 Conversion of Decimal Number to Any Radix Number

**Step 1 :** Convert integer part.

**Step 2 :** Convert fractional part.

- The conversion of integer part is accomplished by successive division method and the conversion of fractional part is accomplished by successive multiplication method.

##### Steps in Successive Division Method

- Divide the integer part of decimal number by desired base number, store quotient (Q) and remainder (R).
- Consider quotient as a new decimal number and repeat step 1 until quotient becomes 0.
- List the remainders in the reverse order.

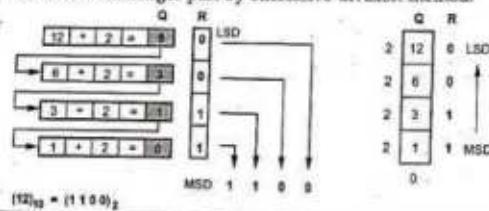
##### Steps in Successive Multiplication Method

- Multiply the fractional part of decimal number by desired base number.
- Record the integer part of product as carry and fractional part as new fractional part.
- Repeat steps 1 and 2 until fractional part of product becomes 0 or until you have many digits as necessary for your application.
- Read carries downwards to get desired base number.

**Example 1.4.9** Convert 12.125 decimal into binary

Solution :

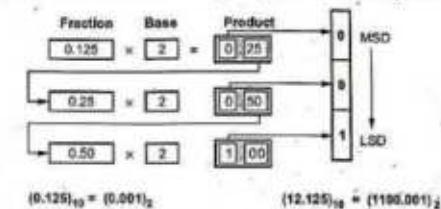
**Integer part :** Conversion of integer part by successive division method.



$$(12)_{10} = (1100)_2$$

TECHNICAL PUBLICATIONS® - An up beat for knowledge

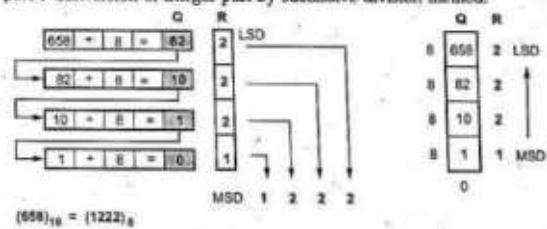
**Fractional part :** Conversion of fractional part by successive multiplication method.



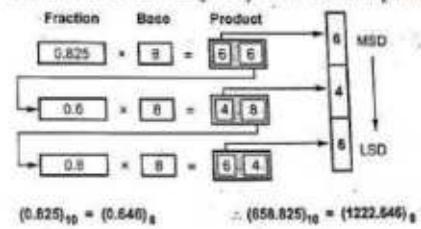
**Example 1.4.10** Convert 658.825 decimal into octal.

Solution :

**Integer part :** Conversion of integer part by successive division method.



**Fractional part :** Conversion of fractional part by successive multiplication method.

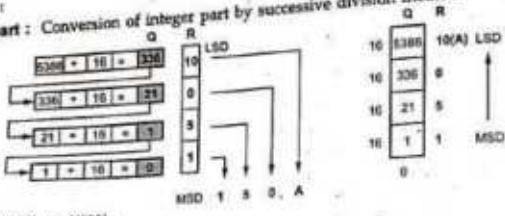


In this example, we have restricted fractional part up to 3 digits. This answer is an approximate answer. To get more accurate answer we have to continue multiplying by 8 until we have as many digits as necessary for our application.

**Example 1.4.11** Convert 5386.345 decimal into hexadecimal.

**Solution :**

Integer part : Conversion of integer part by successive division method.



Fractional part : Conversion of fractional part by successive multiplication method.

In this example, we have restricted fractional part up to 3 digits. This answer is an approximate answer. To get more accurate answer we have to continue multiplying by 16 until we have as many digits as necessary for our application.

**Example 1.4.12** The solution to the quadratic equation " $x^2 - 11x + 22 = 0$ " are  $x = 3$  and  $x = 6$ . What is the base of the number system used?

**Solution :** Let us assume that the base is b. We have,

$$(x-3)(x-6) = x^2 - 11x + 22$$

But 3 and 6 are same in any base, whose value is more than 6.

$$\text{i.e. } (3)_b = (3)_{10} \quad \text{and} \quad (6)_b = (6)_{10}$$

$$(x^2 - 9x + 18)_{10} = (x^2 - 11x + 22)_b$$

Comparing coefficients of x we have

$$(9)_{10} = (11)_b$$

$$9 = b^1 + b^0 = b^1 + 1$$

$$\therefore b = 8$$

**OR**

Comparing coefficients of  $x^0$  we have

$$(18)_{10} = (22)_b$$

$$\therefore 18 = 2 \times b^1 + 2 \times b^0 = 2 \times b^1 + 2$$

$$\therefore 16 = 2 \times b$$

$$\therefore b = 8$$

**Example 1.4.13** Convert the base - 5 number  $(4433214)_5$  directly to base - 12.

**Solution :**  $(4433214)_5 = 4 \times 5^6 + 4 \times 5^5 + 3 \times 5^4 + 3 \times 5^3 \times 2 \times 5^2 + 1 \times 5^1 \times + 4 \times 5^0$

$$= 4 \times 15625 + 4 \times 3125 + 3 \times 625 + 3 \times 125 + 2 \times 25 + 1 \times 5 + 4 \times 1$$

$$= 62500 + 12500 + 1875 + 375 + 50 + 5 + 4$$

$$= 77309$$

12	77309	5
12	6442	10 → A
12	536	8
12	44	6
12	3	3
12	0	

$$\therefore (4433214)_5 = (388A5)_{12}$$

**Example 1.4.14** Convert the base - 7 number  $(35614)_7$  to base - 12.

**Solution :**  $(35614)_7 = 3 \times 7^4 + 5 \times 7^3 + 6 \times 7^2 + 1 \times 7^1 + 4 \times 7^0$

$$= 2103 + 1715 + 294 + 7 + 4$$

$$= 9223$$

12	9223	7
12	768	0
12	64	4
12	5	5
12	0	

$$\therefore (35614)_7 = (5407)_{12}$$

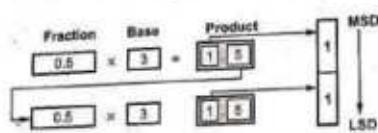
**Example 1.4.15** Convert the decimal number 250.5 to base 3, base 4, base 7 and base 16.

**Solution :** i)  $(250.5)_{10} = (\ )_3$

**Integer part :**

Q	R	LSD
3	250	1
3	83	2
3	27	0
3	9	0
3	3	1
3	1	1
0	0	

$$(250)_{10} = (110021)_3$$

**Fractional part :**

$$(0.5)_{10} = (0.11)_3$$

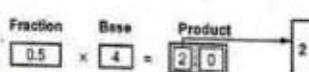
$$(250.5)_{10} = (110021.11)_3$$

$$\text{iv)} \quad (250.5)_{10} = (?)_4$$

**Integer part :**

Q	R	LSD
4	250	2
4	62	2
4	15	3
4	3	3
0	0	

$$(250)_{10} = (110021)_2$$

**Fractional part :**

$$\text{i)} \quad (0.5)_{10} = (0.2)_4$$

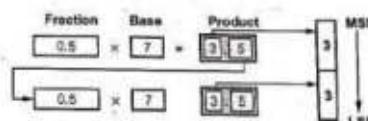
$$\text{ii)} \quad (250.5)_{10} = (3322.2)_4$$

$$\text{iii)} \quad (250.5)_{10} = (?)_7$$

**Integer part :**

Q	R	LSD
7	250	2
7	35	6
7	5	5
0	0	

$$(250)_{10} = (505)_7$$

**Fractional part :**

$$(0.5)_{10} = (0.33)_7$$

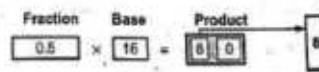
$$(250.5)_{10} = (505.33)_7$$

$$\text{iv)} \quad (250.5)_{10} = (?)_{16}$$

**Integer part :**

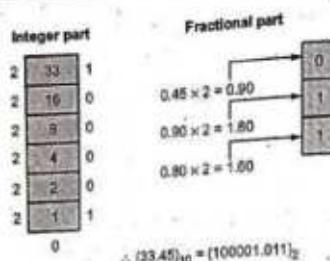
Q	R	LSD
16	250	10 → A
16	15	15 → F
0	0	

$$(250)_{10} = (FA)_{16}$$

**Fractional part :**



**Example 1.4.19** Convert  $(33.45)_{10}$  to binary. Result should be accurate to within  $0.01_{10}$ .  
GTU : Summer-18, Marks 1



Solution :  $\therefore (33.45)_{10} = (100001.011)_2$

**Example 1.4.20**  $(56)_{10} = (?)_2$

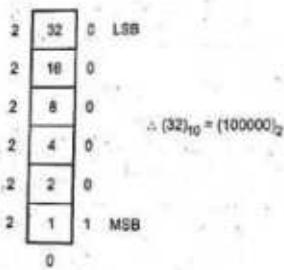
GTU : Winter-16, Mark 1

Solution :  $(56)_{10} = 5 \times 10^1 + 6 \times 10^0 = 80 + 6 = (86)_{10}$

**Example 1.4.21**  $(32)_{10} = (?)_2$

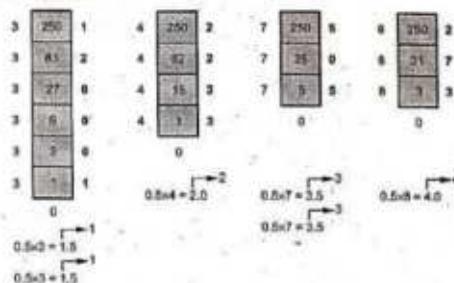
GTU : Winter-16, Mark 1

Solution :



**Example 1.4.22** Convert the decimal number 250.5 to base 3, base 4, base 7 and base 8.  
GTU : Summer-17, Marks 4

**Solution :**



$$(250.5) = (100021.11)_3 = (3322.2)_4 = (505.33)_7 = (372.4)_8$$

**Example 1.4.23** Convert decimal number  $(0.252)_{10}$  to binary with an error less than  $1\%$ .  
GTU : Winter-15, Marks 3

**Solution :** Error allowable is  $0.01 \times 0.252$

$$= 0.00252$$

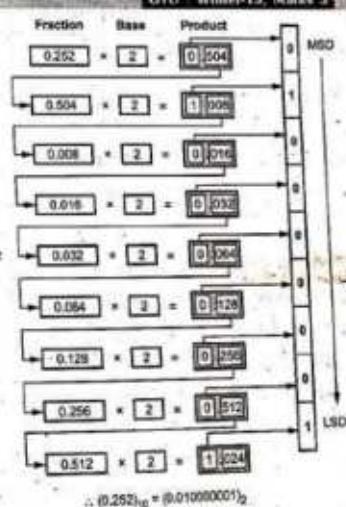
$$2^{-n} < 0.00252 \text{ or } 2^n > 397$$

$$n = \frac{\log 397}{\log 2} = 8.63$$

$$\therefore n = 8.63 = 9$$

(nearest integer)

Number of digits after decimal are nine.



$$\therefore (0.252)_{10} = (0.01000001)_2$$

**Example 1.4.24** Convert the following numbers.

$$\text{i)} (52)_{10} = (?)_2 \quad \text{ii)} (436)_{10} = (?)_{16} \quad \text{iii)} (5C7)_{16} = (?)_{10} \quad \text{iv)} (11011101)_2 = (?)_{10}$$

GTU Winter-17, Marks 5

**Solution :**

2	52	0
2	26	0
2	13	1
2	6	0
2	3	1
2	1	1
	0	

$$\therefore (52)_{10} = (110100)_2$$

ii)

	4	3	2	1	0	Octal number
0	0	0	1	0	0	Binary number
1	1	1	1	1	0	Hex number

$$\therefore (436)_{10} = (11E)_{16}$$

$$\text{iii)} (5C7)_{16} = 5 \times 16^2 + 12 \times 16^1 + 7 \times 16^0 = 1280 + 192 + 7 = (1479)_{10}$$

$$\text{iv)} (11011101)_2 = 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}$$

$$= 16 + 8 + 0 + 2 + 1 + 0.5 + 0.125 = 27.625$$

**Example 1.4.25** Do as directed : I. Given that  $(16)_{10} = (100)_2$ , find the value of  $x$ .

$$\text{II. } (4433)_5 = (?)_{10} = (?)_2 \quad \text{GTU Summer-16, Marks 2}$$

**Solution :** I.  $(16)_{10} = 1 \times x^2 + 0 \times x + 0 = x^2 \quad \therefore x = 4$ .

III)

$$(4433)_5 = 4 \times 5^3 + 4 \times 5^2 + 3 \times 5^1 + 3 \times 5^0 = 500 + 100 + 15 + 3 = (618)_{10}$$

2	818	0	LSD
2	309	1	
2	154	0	
2	77	1	
2	38	0	
2	19	1	
2	9	1	
2	4	0	
2	2	0	
2	1	1	MSD
	0		

$$\therefore (818)_{10} = (1001101010)_2$$

**Example 1.4.26** Do as directed :  $(1011011101101110)_2 = (?)_{10}$  GTU Summer-16, Mark 1

**Solution :**  $\begin{array}{cccc} 1011 & 0111 & 0110 & 1110 \\ \text{B} & 7 & 6 & \text{E} \end{array}$

$$\therefore (1011011101101110)_2 = (B76E)_{16}$$

## 1.5 Complement of Numbers

- In this section, we see the complement numbers used in different number systems. These complement numbers are used in case of subtraction of two numbers in corresponding number systems.

### 1.5.1 1's Complement Representation

- The 1's complement of a binary number is the number that results when we change all 1's to zeros and the zeros to ones.

**Example 1.5.1** Find 1's complement of  $(11010100)_2$ .

**Solution :**

1	1	0	1	0	1	0	0	Number
▽	▽	▽	▽	▽	▽	▽	▽	NOT operation
0	0	1	0	1	0	1	1	1's complement of number

### 1.5.2 2's Complement Representation

- The 2's complement is the binary number that results when we add 1 to the 1's complement. It is given as,

$$2^{\text{nd}} \text{ complement} = 1^{\text{st}} \text{ complement} + 1$$

- The 2's complement form is used to represent negative numbers.

**Example 1.5.2** Find 2's complement of  $(11000100)_2$ .

**Solution :**

Number							
			1	1			
0	0	1	1	1	0	1	1
						1	
+ 1							
0	0	1	1	1	1	0	0

Carry  
1's complement of number  
Add 1  
2's complement of number

**Example 1.5.3** Represent the decimal number  $-200$  and  $200$  using 2's complement binary form.

**Solution :** To represent  $200$  and  $-200$  we need 9-bit number system.

$+200 =$	0 1 1 0 0 1 0 0 0	
$-200 =$	1 0 0 1 1 0 1 1 1	1's complement
	*	Add 1
	1 0 0 1 1 1 0 0 0	2's complement

Fig. 1.5.1

## 1.6 Signed Binary Numbers

- In practice, we use plus sign to represent positive number and minus sign to represent negative number.
- However, because of hardware limitations, in computers, both positive and negative numbers are represented with only binary digits.
- The leftmost bit (sign bit) in the number represents sign of the number.
- The sign bit is 0 for positive numbers and it is 1 for negative numbers.
- Fig. 1.6.1 shows the sign magnitude format for 8-bit signed number.
- Here are some examples of sign-magnitude numbers.

- $+6 = 0000\ 0110$
- $-14 = 1000\ 1110$
- $+24 = 0001\ 1000$
- $-64 = 1100\ 0000$

Maximum positive number: 0 1 1 1 1 1 1 1 = +127  
 Maximum negative number: 1 1 1 1 1 1 1 1 = -127

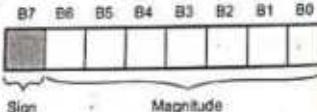


Fig. 1.6.1

- Let us see how  $-6$  is represented in these three formats. Consider the number  $6$  represented in binary with eight bits.

Signed-magnitude representation:	1 0 0 0 0 1 1 0
Signed-1's complement representation:	1 1 1 1 1 0 0 1
Signed-2's complement representation:	1 1 1 1 1 1 0 1 0

- In signed-magnitude,  $-6$  is obtained from  $+6$  by changing the sign bit in the leftmost position from 0 to 1. In signed-1's complement,  $-6$  is obtained by complementing all the bits of  $+6$ , including the sign bit. The signed-2's complement representation of  $-6$  is obtained by taking 2's complement of positive number, including the sign bit.
- The Table 1.6.1 lists all possible 4-bit signed binary numbers in the three representations. Looking at the Table 1.6.1 we understand following points :
  - Positive numbers in all three representations are identical and have 0 in the leftmost position.
  - All negative numbers have a 1 in the leftmost bit position.
  - The signed-2's complement system has only one representation for 0, which is always positive.
  - The signed-magnitude and 1's complement systems have either a positive 0 or a negative 0.
  - With four bits, we can represent 16 binary numbers.
  - The sign-magnitude representation requires separate handling for sign and magnitude during arithmetic operations and hence it is suitable in computer arithmetic. Therefore, the signed complement numbers are normally used in computer arithmetic. The 1's complement imposes some difficulties and is seldom used for arithmetic operations. It is used as a logical operation since the change of 1 to 0 or 0 to 1 is equivalent to a logical complement operation. The signed-2's complement system is commonly used in computer arithmetic.

Table 1.6.1

**Example 1.6.1** Represent numbers + 9 and - 9 signed-magnitude, signed 1's complement and signed 2's complement 8-bit form.

**Solution :** Signed magnitude representation of a number :

In this representation most significant bit is used to code sign of a number (1 = negative, 0 = + ve) and remaining bits are used to represent magnitude with 'n' bits representation, number represented will be from  $+2^{n-1}$  to  $-2^{n-1}$ .

Number	Signed magnitude representation	1's complement representation	2's complement representation
+ 9	0000 1001	0000 1001	0000 1001
- 9	1000 1001	1111 0110	1111 0111

**Example 1.6.2** Represent decimal number  $-13^*$  in all three methods of negative binary number representation using eight bits.

**Solution :**

1 0 0 0 0 1 1 0 1	- 13 in sign magnitude representation
1 1 1 1 0 0 1 0	- 13 in 1's complement representation
1 1 1 1 0 0 1 1	- 13 in 2's complement representation

#### Example for Practice

**Example 1.6.3 :** Write the sign-magnitude, 1's complement and 2's complement forms of  $-36$  in a 8-bit number system [Ans.: Sign-magnitude: 10100100  
1's complement: 11011011, 2's complement: 11011100]

#### Review Questions

- What are 1's complement and 2's complement numbers?
- State advantages of 2's complement number representation.
- State the different ways for representing the signed binary numbers.

### 1.7 Binary Arithmetic

GTU : May-13, Summer-15, Winter-17

#### 1.7.1 Binary Addition

- The addition consists of four possible elementary operations, as shown in Table 1.7.1.

- The first three operations produce a sum whose length is one digit, but when the last operation is performed sum is two digits.
- The higher significant bit of this result is called a carry and lower significant bit is called sum.
- Such a operation is known as half addition.
- The operation which performs addition of three bits (two significant bits and a previous carry) is called a full addition.
- The Table 1.7.2 shows the truth table for full addition.

#### Addition Method

- Add bits column-wise starting from LSB with carry if any.
- Put the sum at the bottom of the same column.
- Put the carry, if any, on the top of next column.

St. No.	Operations
0.	$0 + 0 = 0$
1.	$0 + 1 = 1$
2.	$1 + 0 = 1$
3.	$1 + 1 = 10_2$

Table 1.7.1 Addition operations

Inputs			Outputs	
A	B	C <sub>in</sub>	C <sub>out</sub>	Sum (S)
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Table 1.7.2 Truth table for full-adder

**Example 1.7.1** Perform addition of  $(11001100)_2$  and  $(11011010)_2$ .

**Solution :** In the above example, we have seen the addition of two unsigned binary numbers. In case of the signed numbers, we have to consider the sign of the number and we may require sign extension to avoid possible overflow.

1 1	1 1			Carry
1 1 0 0 1 1 0 0				Number 1
1 1 0 1 1 0 1 0				Number 2
1 1 0 1 0 0 1 1 0				Result (S)

**Example 1.7.2** Add  $(28)_{10}$  and  $(15)_{10}$  by converting them into binary.

**Solution :** Using decimal to binary conversion technique discussed in section 1.4.8 we have,

Sign Extension →	+	1 1 1		Carry
		0 0 1 1 1 0 0		Binary equivalent of $(28)_{10}$
		0 0 0 1 1 1 1		$\times (15)_{10}$
		0 1 0 1 0 1 1		Result : Binary equivalent of $(43)_{10}$
				$(43)_{10}$

$$(28)_{10} = (011100)_2 \text{ and } (15)_{10} = (01111)_2$$

### 1.7.2 Binary Subtraction

- The subtraction consists of four possible elementary operations, as shown in Table 1.7.3.
- In all operations, each subtrahend bit is subtracted from the minuend bit. In case of second operation the minuend bit is smaller than the subtrahend bit, hence 1 is borrowed.

Sr. No.	Operations
1.	$0 - 0 = 0$
2.	$0 - 1 = 1$ with 1 borrow
3.	$1 - 0 = 1$
4.	$1 - 1 = 0$

#### Subtraction Method

- Subtract bits column-wise starting from LSB with borrow if any.
- Put the difference at the bottom of the same column.
- Take the borrow, if required from the next column.

Table 1.7.3 Subtraction operations

#### Example 1.7.3 Perform $(11101100)_2 - (00110100)_2$ .

Solution :

Number 1									
0 1 0 1 1 1 0 0					0 1 0 1 1 0 0				
$-$					$-$				
0 0 1 1 0 0 1 0					0 0 1 1 0 0 1 0				
1 0 1 1 1 0 1 0					1 0 1 1 1 0 1 0				

Note :  $(10)_2 - (1)_2 = (1)_2$

- However, the subtraction technique we have just seen is difficult to implement in a computer. Thus, computer uses 2's complement representation to implement negative number.

### 1.7.3 Binary Subtraction using 1's Complement Method

- In a 1's complement subtraction, negative number is represented in the 1's complement form and actual addition is performed to get the desired result.
- For example, operation  $A - B$  is performed using following steps :
  - Take 1's complement of B.
  - Result  $\leftarrow A + 1$ 's complement of B.
  - If carry is generated then the result is positive and in the true form. Add carry to the result to get the final result.
  - If carry is not generated then the result is negative and in the 1's complement form.

TECHNICAL PUBLICATIONS™ An up thrust for knowledge

#### Example 1.7.4 Perform $(28)_{10} - (15)_{10}$ using 6-bit 1's complement representation.

Solution :  $(28)_{10} = (011100)_2$ 

$$(15)_{10} = (001111)_2$$

$$\begin{array}{ccccccc} 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ \text{Carry} & & & & & & (15)_{10} \end{array}$$

$$\begin{array}{ccccccc} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ \text{1's complement of } (15)_{10} & & & & & & \end{array}$$

Sign Extension	$\begin{array}{ccccc} 1 & 1 & & & \\ 0 & 1 & 1 & 1 & 0 & 0 \\ \xrightarrow{-} & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \end{array}$	Carry	$(28)_{10}$
	$\begin{array}{ccccc} 1 & 1 & & & \\ 0 & 1 & 1 & 1 & 0 & 0 \\ \xrightarrow{-} & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \end{array}$		$(-15)_{10}$
		Result	$(13)_{10}$
		Add end around carry	
		Final result : Binary equivalent of $(13)_{10}$	

#### Example 1.7.5 Perform $(15)_{10} - (28)_{10}$ using 6-bit 1's complement representation.

Solution :  $(15)_{10} = (001111)_2$ 

$$(28)_{10} = (011100)_2$$

$$\begin{array}{ccccccc} 0 & 1 & 1 & 1 & 0 & 0 & \\ \text{Binary equivalent of } (28)_{10} & & & & & & \end{array}$$

$$\begin{array}{ccccccc} 1 & 1 & & & & & \\ 1 & 0 & 0 & 0 & 1 & 1 & \\ \text{Carry} & & & & & & \text{1's complement of } (28)_{10} \end{array}$$

Verification	$\begin{array}{ccccc} 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ \xrightarrow{+} & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \end{array}$	Carry	$(15)_{10}$
	$\begin{array}{ccccc} 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ \xrightarrow{+} & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \end{array}$		$(-25)_{10}$
		Result = Binary equivalent of $(-13)_{10}$	$(-13)_{10}$

Verification :  $0 \ 0 \ 1 \ 1 \ 0 \ 1$  is 1's complement of result (Binary equivalent of  $(13)_{10}$ )

### 1.7.4 Binary Subtraction using 2's Complement Method

- In a 2's complement subtraction, negative number is represented in the 2's complement form and actual addition is performed to get the desired result.
- For example, operation  $A - B$  is performed using following steps :
  - Take 2's complement of B.
  - Result  $\leftarrow A + 2$ 's complement of B.

TECHNICAL PUBLICATIONS™ An up thrust for knowledge

3. If carry is generated then the result is positive and in the true form. In this case, carry is ignored.
4. If carry is not generated then the result is negative and in the 2's complement form.

**Example 1.7.6** Perform  $(28)_{10} - (15)_{10}$  using 6-bit 2's complement representation.

**Solution :**  $(28)_{10} = (011100)_2$

$$(15)_{10} = (001111)_2$$

$$\begin{array}{r} 0 \ 0 \ 1 \ 1 \ 1 \ 1 \\ + 1 \ 1 \ 0 \ 0 \ 0 \ 1 \\ \hline \end{array} \quad (15)_{10}$$

Carry

1's complement of  $(15)_{10}$ 

Add 1

2's complement of 15, i.e.,  $(-15)_{10}$ 

$$\begin{array}{r} 1 \ 1 \ 0 \ 0 \ 0 \ 0 \\ + 1 \ 1 \ 0 \ 0 \ 0 \ 1 \\ \hline \end{array} \quad \text{Binary equivalent of } (28)_{10}$$
  

$$\begin{array}{r} 0 \ 1 \ 1 \ 1 \ 0 \ 0 \\ + 1 \ 1 \ 0 \ 0 \ 0 \ 1 \\ \hline \end{array} \quad \text{2's complement of } (15)_{10}$$
  

$$\begin{array}{r} \text{Carry} \\ \hline \end{array}$$

$$\begin{array}{r} 1 \ 1 \ 1 \ 1 \ 0 \ 0 \\ + (-15)_{10} \\ \hline (13)_{10} \end{array} \quad \text{Result: Binary equivalent of } (13)_{10}$$

Sign Extension

Ignore Carry

Discard carry

$$\begin{array}{r} 1 \ 1 \ 1 \ 1 \ 0 \ 0 \\ + 0 \ 0 \ 0 \ 1 \ 1 \ 1 \\ \hline \end{array} \quad \text{Subtrahend}$$
  

$$\begin{array}{r} 0 \ 0 \ 0 \ 1 \ 1 \ 1 \\ + 1 \ 1 \ 1 \ 0 \ 0 \ 0 \\ \hline \end{array} \quad \text{1's complement of subtrahend}$$
  

$$\begin{array}{r} 1 \ 1 \ 1 \ 1 \ 0 \ 0 \\ + 1 \ 1 \ 1 \ 1 \ 0 \ 1 \\ \hline \end{array} \quad \text{2's complement of subtrahend}$$

No carry, result is negative and in 2's complement form

**Example 1.7.7** Perform  $(15)_{10} - (28)_{10}$  using 6-bit 2's complement representation.

**Solution :**  $(15)_{10} = (001111)_2$

$$(28)_{10} = (011100)_2$$

Binary equivalent of  $(28)_{10}$ 

Carry

1's complement of  $(28)_{10}$ 

Add 1

2's complement of  $(28)_{10}$ , i.e.,  $(-28)_{10}$ 

$$\begin{array}{r} 0 \ 1 \ 1 \ 1 \ 0 \ 0 \\ + 1 \ 0 \ 0 \ 0 \ 1 \ 1 \\ \hline \end{array} \quad \text{Binary equivalent of } (28)_{10}$$
  

$$\begin{array}{r} 1 \ 1 \ 1 \ 1 \ 0 \ 0 \\ + 1 \ 1 \ 1 \ 0 \ 0 \ 1 \\ \hline \end{array} \quad \text{2's complement of } (28)_{10}$$
  

$$\begin{array}{r} \text{Carry} \\ \hline \end{array}$$

$$\begin{array}{r} 1 \ 1 \ 1 \ 1 \ 0 \ 0 \\ + (-28)_{10} \\ \hline (-13)_{10} \end{array} \quad \text{No carry, thus result is negative and in 2's complement form}$$

Verification

1's complement of result

$$\begin{array}{r} 0 \ 0 \ 1 \ 1 \ 0 \ 0 \\ + 0 \ 0 \ 1 \ 1 \ 0 \ 1 \\ \hline \end{array} \quad \text{- Result = Binary equivalent of } (13)_{10}$$
  

$$\begin{array}{r} 1 \ 1 \ 1 \ 1 \ 0 \ 0 \\ + 1 \ 0 \ 1 \ 0 \ 1 \ 0 \\ \hline \end{array} \quad \text{Add 1}$$

**Example 1.7.8** Perform the following subtraction using 2's complement method:

i)  $01000 - 01001$ ii)  $0011001 - 0001110$ 

**Solution : i)  $01000 - 01001$**

$$\begin{array}{r} 0 \ 1 \ 0 \ 0 \ 0 \ 1 \\ + 1 \ 0 \ 1 \ 1 \ 0 \ 0 \\ \hline \end{array} \quad \text{Subtractor}$$

1's complement of subtractor  
Add 1  
2's complement of subtractor

$$\begin{array}{r} 0 \ 1 \ 0 \ 0 \ 0 \ 0 \\ + 1 \ 0 \ 1 \ 1 \ 0 \ 1 \\ \hline \end{array} \quad \text{Subtractor}$$

2's complement of subtractor  
No carry, result is negative and in 2's complement form

**Solution : ii)  $00111001 - 00011100 = 00011011$**

$$\begin{array}{r} 0 \ 0 \ 1 \ 1 \ 1 \ 0 \ 1 \\ + 1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0 \\ \hline \end{array} \quad \text{Subtractor}$$

1's complement of subtractor

Add 1  
2's complement of subtractor

$$\begin{array}{r} 0 \ 0 \ 1 \ 1 \ 1 \ 0 \ 0 \\ + 1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 1 \\ \hline \end{array} \quad \text{Subtractor}$$

2's complement of subtractor  
Carry

2's complement of subtractor  
No carry, result is negative and in 2's complement form

**Example 1.7.9** Perform following arithmetic operation in binary using signed 2's complement representation:  $(-42) - (-13)$

**Solution :**

$$-42 - (-13) = -42 + 13$$

$$(+13)_{10} = (0001101)_2$$

$$(+42)_{10} = (0101010)_2$$

$$(15)_{10}$$

$$(-28)_{10}$$

$$(-13)_{10}$$

$$(-42)_{10}$$

$$(+42)_{10}$$

$$(+13)_{10}$$

$$(-42)_{10}$$

$$(+13)_{10}$$

$$(-42)_{10}$$

TECHNICAL PUBLICATIONS™ - An up must for knowledge

**Example 1.7.10** Perform the subtraction with the following decimal numbers using 1's complement and 2's complements  
 a)  $11010 - 1101$  b)  $10010 - 10011$

GTU : May-13, Marks 7

Solution : It is assumed that given numbers are signed numbers.

$$a) 11010 - 1101$$

Subtraction using 1's complement

1	1	0	1	(13) <sub>10</sub>
0	0	1	0	1's complement of (13) <sub>10</sub>

Carry				
1	1	0	1	0
(26) <sub>10</sub>				
0	0	0	1	0
				1's complement of (13) <sub>10</sub>
1	1	1	0	0
				Result

Fig. 1.7.1

There is no end around carry. Therefore, the answer is,

$$-(1's \text{ complement of } 11100) = -(00011)$$

Subtraction using 2's complement

1	1	0	1	(13) <sub>10</sub>
0	0	1	0	1's complement of (13) <sub>10</sub>
*			1	Add 1
0	0	1	1	2's complement of (13) <sub>10</sub>

Carry				
1	1	0	1	0
(26) <sub>10</sub>				
+ 0	0	0	1	1
1	1	1	0	1
				2's complement of (13) <sub>10</sub>
1	1	1	0	1
				Result

Fig. 1.7.2

There is no end carry. Therefore, the answer is,

$$-(2's \text{ complement of } 11101) = -(00011)$$

$$b) 10010 - 10011$$

Subtraction using 1's complement

1	0	0	1	1	(19) <sub>10</sub>
0	1	1	0	0	1's complement of (19) <sub>10</sub>
*				1	
1	0	0	1	0	(18) <sub>10</sub>
0	1	1	0	0	1's complement of (19) <sub>10</sub>
1	1	1	1	0	Result

Fig. 1.7.3

There is no end carry. Therefore, the answer is,

$$-(1's \text{ complement of } 11110) = -(00001)$$

Subtraction using 2's complement

1	0	0	1	1	(19) <sub>10</sub>
0	1	1	0	0	1's complement of (19) <sub>10</sub>
*			1	Add 1	
0	1	1	0	1	2's complement of (19) <sub>10</sub>

Fig. 1.7.4

There is no end carry. Therefore, the answer is,

$$-(2's \text{ complement of } 11111) = -(00001)$$

**Example 1.7.11** Perform subtraction of  $(78)_{10} - (58)_{10}$  using 2's complement method.

GTU : Summer-15, Marks 3

Solution :  $(78)_{10} = (01001110)_2$

$$(58)_{10} = (0111010)_2$$

0	1	1	1	0	1	0
1	0	0	0	1	0	1
*				1		
1	0	0	0	1	1	0
					2's complement of 58	
					Carry	
					(78) <sub>10</sub>	
					2's complement of 58	
					(20) <sub>10</sub>	

**Example 1.7.12** Perform following subtraction using 2's complement method.  
 $(11010)_2 - (10000)_2$

Solution :

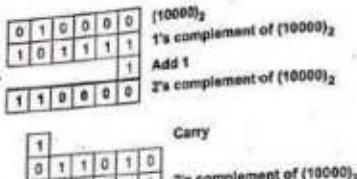
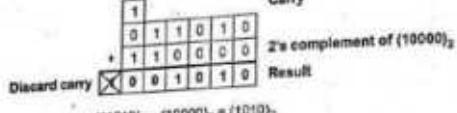
 $(11010)_2$ 1's complement of $(10000)_2$ Add 1 2's complement of $(10000)_2$
<b>Carry</b>  Discard carry <b>Result</b> $\Delta (11010)_2 - (10000)_2 = (10110)_2$

Fig. 1.7.5

GTU : Summer-10

## 1.8 Octal Arithmetic

### 1.8.1 Octal Addition

- The sum of two octal digits is the same as their decimal sum, provided the decimal sum is less than 8. If the decimal sum is 8 or greater, subtract 8 to obtain the octal digit.
- A carry of 1 is produced when the decimal sum is corrected this way, as illustrated in the following examples.

**Example 1.8.1** a)  $4_8 + 2_8$ ; b)  $6_8 + 7_8$ ; c)  $1_8 + 7_8$ .

Solution : a)  $4_8 + 2_8 = (6)_8$ b)  $6_8 + 7_8 = (13 - 8) 5_8$  carry 1c)  $1_8 + 7_8 = (8 - 8) 0_8$  carry 1

**Notes** Subscript 8 denotes it is an octal number.

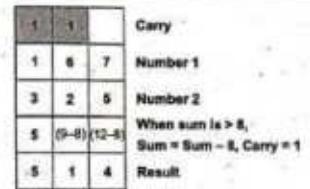
- To obtain the sum of multi-digit octal numbers, the procedure just described is applied to each column of digits as illustrated in the following example.

**Example 1.8.2** Add  $(167)_8$  and  $(325)_8$ .

Solution :

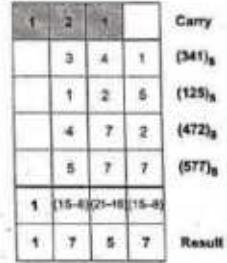
- If the decimal sum of several octal digits is 16 or greater, subtract 16 and set carry equal to 2.

- In general, we can express any decimal sum in octal by repeatedly subtracting 8 until the result is one of the octal digits through 0 to 7.
- Each time 8 is subtracted, the amount of the carry is increased by 1. This procedure is illustrated in the following example.

 <b>Number 1</b> <b>Number 2</b> $5 \quad (9-8) \quad (12-8)$ <b>Sum = Sum - 8, Carry = 1</b> <b>Result</b> <b>Carry</b>
---

**Example 1.8.3** Add the octal numbers  $(341)_8$ ,  $(125)_8$ ,  $(472)_8$  and  $(577)_8$ .

Solution :

 <b>Carry</b> $(341)_8$ $(125)_8$ $(472)_8$ $(577)_8$ <b>Result</b>
---

$$(341)_8 + (125)_8 + (472)_8 + (577)_8 = (1757)_8$$

### 1.8.2 Octal Subtraction using 8's Complement

- We have seen how binary subtraction is performed using 2's complement method. Similarly, octal subtraction is performed using 8's complement method.
- The 8's complement of an octal number is found by adding a 1 to the least significant bit of the 7's complement of an octal number, as illustrated in the Example 1.8.2.

**Example 1.8.4** Find the 8's complement of  $(346)_8$ .

Solution :

**Step 1 :** Subtract each digit of a number from 7 to get the 7's complement of the number.

$7 \quad 7 \quad 7$ $3 \quad 4 \quad 6$ $4 \quad 3 \quad 1$	$[346]_8$ 7's complement of $(346)_8$
$+ \quad \quad 1$ $4 \quad 3 \quad 2$	Add 1 <b>8's complement of <math>(346)_8</math></b>

**Step 2 :** Add 1 to it to get the 8's complement of the number.

The steps for octal subtraction using 8's complement method are as given below.

**Step 1 :** Find 8's complement of subtrahend.

**Step 2 :** Add two octal numbers (first number and 8's complement of the second number).

**Step 3 :** If carry is produced in the addition it is ignored; otherwise find 8's complement of the sum as a result with negative sign.

**Example 1.8.5** Use 8's complement method of subtraction to compute  $(516)_8 - (413)_8$ .

**Solution :**

Step 1:	<table border="1"> <tr><td>7</td><td>7</td><td>7</td></tr> <tr><td>4</td><td>1</td><td>3</td></tr> <tr><td>3</td><td>6</td><td>4</td></tr> <tr><td>1</td><td></td><td></td></tr> <tr><td>3</td><td>6</td><td>5</td></tr> </table>	7	7	7	4	1	3	3	6	4	1			3	6	5	$(413)_8$
7	7	7															
4	1	3															
3	6	4															
1																	
3	6	5															
		7's complement of $(413)_8$															

Step 2:	<table border="1"> <tr><td>1</td><td>1</td><td>0</td></tr> <tr><td>3</td><td>1</td><td>0</td></tr> <tr><td>3</td><td>6</td><td>5</td></tr> <tr><td>1</td><td>(9-8)</td><td>(9-8)</td></tr> <tr><td>1</td><td>0</td><td>3</td></tr> </table>	1	1	0	3	1	0	3	6	5	1	(9-8)	(9-8)	1	0	3	Carry
1	1	0															
3	1	0															
3	6	5															
1	(9-8)	(9-8)															
1	0	3															
		$(316)_8$															
		8's complement of $(316)_8$															
		Add 1															
		Ignore carry → $\times$															
		Result															

Step 3:

Step 3:	<table border="1"> <tr><td>1</td><td>0</td><td>3</td></tr> <tr><td>1</td><td>0</td><td>3</td></tr> </table>	1	0	3	1	0	3	Result
1	0	3						
1	0	3						

$$\therefore (516)_8 - (413)_8 = (103)_8$$

**Example 1.8.6** Use the 8's complement method of subtraction to compute  $316_8 - 451_8$ .

**Solution :**

Step 1:	<table border="1"> <tr><td>7</td><td>7</td><td>7</td></tr> <tr><td>4</td><td>5</td><td>1</td></tr> <tr><td>3</td><td>2</td><td>6</td></tr> <tr><td>*</td><td>1</td><td></td></tr> <tr><td>3</td><td>2</td><td>7</td></tr> </table>	7	7	7	4	5	1	3	2	6	*	1		3	2	7	$(451)_8$
7	7	7															
4	5	1															
3	2	6															
*	1																
3	2	7															
		7's complement of $(451)_8$															

Step 2:	<table border="1"> <tr><td>1</td><td>1</td><td>0</td></tr> <tr><td>3</td><td>1</td><td>0</td></tr> <tr><td>3</td><td>2</td><td>7</td></tr> <tr><td>1</td><td>(9-8)</td><td>(9-8)</td></tr> <tr><td>6</td><td>4</td><td>(3-8)</td></tr> <tr><td>6</td><td>4</td><td>5</td></tr> </table>	1	1	0	3	1	0	3	2	7	1	(9-8)	(9-8)	6	4	(3-8)	6	4	5	Carry
1	1	0																		
3	1	0																		
3	2	7																		
1	(9-8)	(9-8)																		
6	4	(3-8)																		
6	4	5																		
		$(316)_8$																		
		8's complement of $(316)_8$																		
		Add 1																		
		Ignore carry → $\times$																		
		Result																		

Step 3 : No carry, hence take 8's complement.

	<table border="1"> <tr><td>7</td><td>7</td><td>7</td></tr> <tr><td>6</td><td>4</td><td>5</td></tr> <tr><td>1</td><td>3</td><td>2</td></tr> <tr><td>*</td><td>1</td><td></td></tr> <tr><td>1</td><td>3</td><td>2</td></tr> </table>	7	7	7	6	4	5	1	3	2	*	1		1	3	2	7's complement of result
7	7	7															
6	4	5															
1	3	2															
*	1																
1	3	2															
		Add 1															

$$\therefore (316)_8 - (451)_8 \rightarrow (-133)_8$$

**Example 1.8.7** i. Subtract  $(45)_8$  from  $(66)_8$

GTU Summer-16, Mark 1

**Solution :** Let us perform octal subtraction using 8's complement

8's complement of 45 =  $77 - 45 + 1 = 33$

$$\therefore (66)_8 - (45)_8 = (21)_8$$

1	1		Carry
		6	6
		3	3
		(10-8)	(9-8)
		2	1

Ignore carry  $\times$  Result  
 $\therefore (66)_8 - (45)_8 = (21)_8$

Fig. 1.8.1

## 1.9 Hexadecimal Arithmetic

GTU Summer-16

### 1.9.1 Hexadecimal Addition

- The sum of two hexadecimal digits is the same as their equivalent decimal sum (with any sum from 10 through 16 is expresses a letter from A through F), provided the decimal equivalent is less than 16.
- If the decimal sum is 16 or greater, subtract 16 to obtain the hexadecimal digit.
- A carry of 1 is produced when the decimal sum is corrected this way, as illustrated in the following examples.

**Example 1.9.1** a)  $3_{16} + 9_{16}$  b)  $9_{16} + 7_{16}$  c)  $A_{16} + 8_{16}$

**Solution :** a)  $3_{16} + 9_{16} = C_{16}$

$$b) 9_{16} + 7_{16} = (15 - 16) \quad 0_{16} \text{ carry } 1 = (10)_{16}$$

$$c) A_{16} + 8_{16} = (18 - 16) \quad 2_{16} \text{ carry } 1 = (12)_{16}$$

**Note :** Sometimes instead of subscript 16, subscript H is used to denote the hexadecimal number.

- To obtain the sum of multi-digit hexadecimal numbers, the procedure just described is applied to each column of digits as illustrated in the following example.

**Example 1.9.2** Add  $3F8_{16}$  and  $5B3_{16}$ 

Solution :

- If the decimal sum of several hex digits is 32 or greater, subtract 32 and set carry equal to 2.
- In general, we can express any decimal sum in hexadecimal form by repeatedly subtracting 16 until the result is one of the hex digits through 0 to F.
- Each time 16 is subtracted the amount of the carry is increased by 1.

1	F	8
3	F	8
5	B	3
9	(20-16)	B
9	A	B

Carry  
 $(3F8)_{16}$   
 $(5B3)_{16}$   
 When Sum > 16, Sum = Sum - 16, Carry + 1

**Example 1.9.3** Add the hexadecimal numbers  $(892)_{16}$ ,  $(58F)_{16}$ ,  $(178)_{16}$ , and  $(48E)_{16}$ 

Solution :

1	2	2	
	8	9	2
	5	8	F
	1	7	8
	4	8	E
	1	(20-16)	(34-32)
	1	4	2

Carry  
 $(892)_{16}$   
 $(58F)_{16}$   
 $(178)_{16}$   
 $(48E)_{16}$   
 Result

$$\therefore (892)_{16} + (58F)_{16} + (178)_{16} + (48E)_{16} = (1427)_{16}$$

**Example 1.9.4** Do as directed. Add  $(6E)_{16}$  and  $(C5)_{16}$ 

Solution :

1	1	
	6	E
*	C	5
	(10-10)	(10-10)
	1	3

Carry

$$\therefore (6E)_{16} + (C5)_{16} = (133)_{16}$$

Fig. 1.9.1

**1.9.2 Hexadecimal Subtraction using 16's Complement**

- Hexadecimal subtraction is best accomplished using the 16's complement method like 2's complement for binary subtraction.
- The 16's complement of a hexadecimal number is found by adding a 1 to the least significant bit of the 15's complement of a hexadecimal number, as illustrated in the example 1.8.4.

**Example 1.9.5** Find the 16's complement of  $(ABC)_{16}$ 

Solution :

**Step 1 :** Subtract each digit of a number from 15 to get the 15's complement of the number.

**Step 2 :** Add 1 to it to get the 16's complement of the number.

- The steps for hexadecimal subtraction using 16's complement method are as given follows.

**Step 1 :** Find 16's complement of subtrahend.

**Step 2 :** Add two hexadecimal numbers (first number and 16's complement of the second number).

**Step 3 :** If carry is produced in the addition it is discarded; otherwise find 16's complement of the sum as a result with negative sign.

**Example 1.9.6** Use the 16's complement method of subtraction to compute  $(CB2)_{16} - (972)_{16}$ 

Solution :

Step 1 :	15	15	15
-	9	7	2
	6	8	D
*			1
	6	8	E

Step 2 :	15	15	15
-	C	B	2
	8	8	E
*			1
	3	4	9

Step 3 :	15	15	15
-	CB2	972	16
	15-10	15-10	15-10
*	5	5	5
	3	4	9

$$\therefore (CB2)_{16} - (972)_{16} \rightarrow (349)_{16}$$

1 - 40 Fundamentals of Digital Systems and Logic Families

**Digital Fundamentals**

**Example 1.9.7** Use the 16's complement method of subtraction to compute  $3B7_{16} - 854_{16}$ .

**Solution :**

Step 1 :  $\begin{array}{r} 15 \ 10 \ 15 \\ - 8 \ 5 \ 4 \\ \hline 7 \ A \ B \end{array}$  15's complement of  $(854)_{16}$

Step 2 :  $\begin{array}{r} 1 \ 1 \\ 3 \ B \ T \\ + 7 \ A \ C \\ \hline 8 \ 6 \ 3 \end{array}$  Carry  $(3B7)_{16}$   
16's complement of  $(854)_{16}$   
Result

Step 3 : No carry, hence take 16's complement  
 $\begin{array}{r} 15 \ 15 \ 15 \\ - 8 \ 6 \ 3 \\ \hline 4 \ 9 \ C \end{array}$  15's complement of result  
 Add 1  
 $\begin{array}{r} 4 \ 9 \ D \\ + 1 \\ \hline \end{array}$  16's complement of result

$\Delta (3B7)_{16} - (854)_{16} \rightarrow (-49D)_{16}$

**Examples for Practice**

**Example 1.9.8 :** Perform addition of  $(FA6)_{16}$  and  $(BC4)_{16}$  [Ans. :  $(1B6A)_{16}$ ]

**Example 1.9.9 :** Perform  $(E8D)_{16} - (1F8)_{16}$  [Ans. :  $(C95)_{16}$ ]

**Example 1.9.10 :** Perform addition of  $(1A6)_{16}$  and  $(474)_{16}$  [Ans. :  $(2E2)_{16}$ ]

**Example 1.9.11 :** Perform  $(EAD)_{16} - (10101010)_2$  [Ans. :  $(DE3)_{16}$ ]

**Example 1.9.12 :** Add the following numbers :  $(ABC)_{16} + (CDE)_{16}$  [Ans. :  $(179A)_{16}$ ]

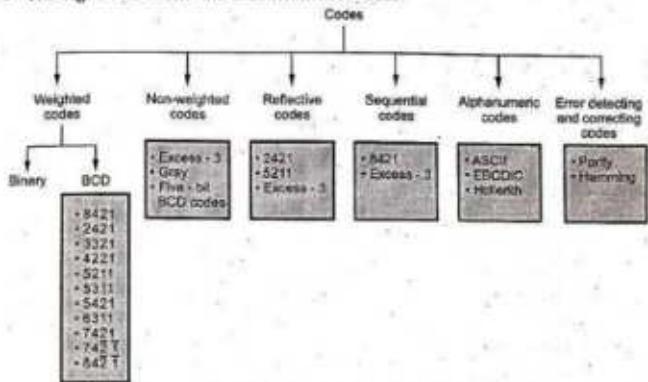
### 1.10 Binary Codes

GTU : May-14, Summer-15, 16, 18, Winter-16

- When numbers, alphabets or words are represented by a specific group of symbols, we can say that they are encoded.
- The group of symbols used to encode them are called codes. The digital data is represented, stored and transmitted as groups of binary digits (bits).
- The group of bits, also known as binary code, represent both numbers and letters of the alphabets as well as many special characters and control functions. They are classified as numeric or alphanumeric.
- Numeric codes are used to represent numbers.
- Alphanumeric codes are used to represent characters : alphabetic letters and numerals.

### 1.10.1 Classification of Binary Codes

- The Fig. 1.10.1 shows the classification of codes.



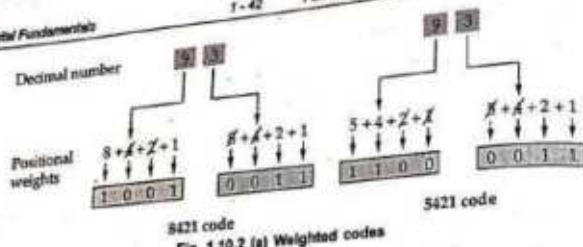


Fig. 1.10.2 (a) Weighted codes

**Reflective codes**

- A code is said to be reflective when the code for 9 is the complement for 0, the code for 8 is complement for 1, 7 for 2, 6 for 3 and 5 for 4. This is illustrated in Fig. 1.10.2 (b).

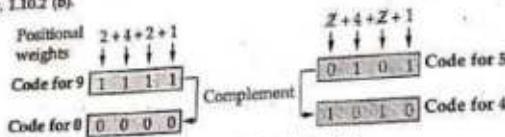


Fig. 1.10.2 (b) Reflective codes

- Like 2421, codes 5211 and excess-3 are also reflective. The 8421 code is not reflective.
- Reflectivity is desirable in a code when the nine's complement must be found, such as in nine's complement subtraction.

**Sequential codes**

- In sequential codes each succeeding code is one binary number greater than its preceding code.
- This greatly aids mathematical manipulation of data.
- The 8421 and excess-3 are sequential, whereas the 2421 and 5211 codes are not.

**Alphanumeric codes**

- The codes which consists of both numbers and alphabetic characters are called alphanumeric codes.
- Most of these codes, however, also represent symbols and various instructions necessary for conveying intelligible information.
- The most commonly used alphanumeric codes are : ASCII (American Standard Code for Information Interchange), EBCDIC (Extended Binary Coded Decimal Interchange Code) and Hollerith code.

TECHNICAL PUBLICATIONS™ - An up thrust for knowledge

**Error detecting and correcting codes**

- When the digital information in the binary form is transmitted from one circuit or system to another circuit or system an error may occur. This means a signal corresponding to 0 may change to 1 or vice versa due to presence of noise.
- To maintain the data integrity between transmitter and receiver, extra bit or more than one bit are added in the data.
- These extra bits allow the detection and sometimes correction of error in the data.
- The data along with the extra bit/bits forms the code. Codes which allow only error detection are called error detecting codes and codes which allow error detection and correction are called error detecting and correcting codes.

**1.10.2 BCD (Binary Coded Decimal) Codes**

- BCD is an abbreviation for binary coded decimal.
- BCD is a numeric code in which each digit of a decimal number is represented by a separate group of 4-bits. The most common BCD code is 8421 BCD.
- It is called 8-4-2-1 BCD because the weights associated with 4 bits are 8-4-2-1 from left to right. This means that, bit-3 has weight 8, bit-2 has weight 4, bit-1 has weight 2 and bit-0 has weight 1.
- The Table 1.10.1 shows the 4-bit 8-4-2-1 BCD code used to represent a single decimal digit.

Decimal Digit	BCD code			
	8	4	2	1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

Table 1.10.1 8-4-2-1 BCD code

TECHNICAL PUBLICATIONS™ - An up thrust for knowledge

- In multidigit coding, each decimal digit is individually coded with 8-4-2-1 BCD code, as shown in the Fig. 1.10.3. Total 8-bits are required to encode  $58_{10}$  in 8-4-2-1 BCD.

**Advantage**

- Easy to convert between it and decimal.

**Disadvantages**

- Since it requires more binary digits to represent multi-digit number than binary number system. It is less efficient. For example, to represent the same number (58) in binary :  $111010_2$ , we require only 6 digits.

- Arithmetic operations are more complex.

**Example 1.10.1** Convert  $1000\ 0110$  (BCD) to decimal, binary and octal.

GTU : Summer-18, Marks 3

Solution :  $(1000\ 0110)_{BCD} = 85_{10}$

2	80	0	0
2	40	1	0
2	20	1	1
2	10	0	0
2	5	1	0
2	2	0	1
2	1	1	0
0			

$\therefore (85)_{10} = (100110)_2$

0	0	1	0	1	0	1	0
Binary							
1	2	8	Octal				

$\therefore (85)_{10} = (100110)_2 = (125)_8$

Fig. 1.10.4

**1.10.2.1 BCD Addition**

- The addition of two BCD numbers can be best understood by considering the three cases that occur when two BCD digits are added.

**Case 1 : Sum equals 9 or less with carry 0**

Let us consider additions of 3 and 6 in BCD.

Decimal	3	6
8-4-2-1 BCD	0 1 1 0	1 0 0 0

Fig. 1.10.3

+ 6	0 1 1 0	Carry	BCD for 6
3	0 0 1 1		BCD for 3
	0 1 0 0		BCD for 9
9	0 1 0 1	Carry	BCD for 5

Valid BCD numbers

- The addition is carried out as in normal binary addition and the sums are 1 0 0 1 and 0 1 1 1 which are valid BCD codes.

**Case 2 : Sum greater than 9 with carry 0**

- Let us consider addition of 6 and 8 in BCD.

+ 8	0 1 1 0	BCD for 6	
8	1 0 0 0	BCD for 3	
14	0 1 1 0	Invalid BCD number (1110) > 9	

Carry

- The sum 1 1 1 0 is an invalid BCD number. This has occurred because the sum of the two digits exceeds 9. Whenever this occurs the sum has to be corrected by the addition of six (0110) in the invalid BCD number, as shown follows.

+ 6	0 1 1 0	BCD for 6	
6	1 0 0 0	BCD for 3	
14	1 1 1 0	Invalid BCD number	
	0 1 1 0	Add 6 for correction	
Carry	1 0 1 0		
	0 0 0 1 0 1 0 0	BCD for 14	
	1 4	Decimal	

- After addition of 6 carry is produced into the second decimal position.

**Case 3 : Sum equals 9 or less with carry 1**

- Let us consider addition of 8 and 9 in BCD.

+ 8	1	0	0	0	Carry
9	1	0	0	1	BCD for 8
17	1	0	0	1	BCD for 9
	1	0	0	0	
Carry	1	0	0	0	Incorrect BCD result
	0	0	0	1	
	1	1			





Solution :

$$\begin{array}{r} 893 = \\ 748 = \\ -748 = \end{array} \begin{array}{c} 1 0 0 1 1 0 0 0 0 0 1 1 \\ 0 1 1 1 0 1 0 0 1 0 0 0 \\ 0 0 1 0 0 1 0 1 0 0 0 1 \end{array}$$

BCD for 893  
BCD for 748  
9's complement of 748

$$\begin{array}{r} 893 \\ 748 \\ -235 \\ \hline \end{array} \begin{array}{c} 1 0 0 1 1 0 0 0 0 0 1 1 \\ + 0 0 1 0 0 1 0 1 0 0 0 1 \\ \hline 1 0 1 1 1 1 0 1 0 1 0 0 \\ + 0 1 1 0 0 1 1 0 0 0 0 0 \\ \hline 1 1 1 1 1 1 \\ \hline 1 0 0 1 0 0 0 1 1 0 1 0 0 \\ \hline 0 0 1 0 0 0 1 1 0 1 0 1 \\ \hline 2 \quad 3 \quad 5 \end{array}$$

BCD for 893  
9's complement of 748  
Add 660

Add end around carry  
BCD for 235

**Example 1.10.11** Perform the following arithmetic operations in BCD numbers :

$$893 - 647.$$

Solution :

$$\begin{array}{r} 893 = \\ 647 = \\ -246 = \end{array} \begin{array}{c} 1 0 0 0 1 0 0 1 0 0 1 1 \\ 0 1 1 0 0 1 0 0 0 1 1 1 \\ 0 0 1 1 0 1 0 1 0 0 1 0 \end{array}$$

BCD for 893  
BCD for 647  
9's complement of 647

$$\begin{array}{r} 893 \\ 647 \\ -246 \\ \hline \end{array} \begin{array}{c} 1 0 0 0 1 0 0 1 0 0 1 1 \\ + 0 0 1 1 0 1 0 1 0 0 1 0 \\ \hline 1 0 1 1 1 1 0 0 1 0 1 0 1 \\ 0 1 1 0 0 1 1 0 0 0 0 0 0 \\ \hline 1 1 1 1 \\ \hline 1 0 0 1 0 0 1 0 0 0 1 0 1 \\ \hline 0 0 1 0 0 1 0 0 0 1 1 0 \end{array}$$

BCD for 893  
9's complement of 647

Add 660

Add end around carry  
BCD for 246

### 1.10.2.3 BCD Subtraction using 10's Complement Method

- Subtraction of BCD numbers can also be performed by representing negative numbers in the 10's complement form.
- The 10's complement of a decimal number is equal to the 9's complement plus 1.

**Example 1.10.12** Find the 10's complement of (3497)<sub>BCD</sub>.

Solution :

Steps for subtraction of BCD numbers using 10's complement method

$$\begin{array}{c} 9 9 9 9 \\ 3 4 9 7 \\ 6 5 8 2 \\ + 1 \\ \hline 6 5 0 3 \end{array}$$

(3497)<sub>BCD</sub>  
9's complement of (3497)<sub>BCD</sub>  
Add 1  
10's complement of (3497)<sub>BCD</sub>

**Step 1 :** Find the 10's complement of a negative number.

**Step 2 :** Add two numbers using BCD addition (Minuend + 10's complement of subtrahend).

**Step 3 :** If carry is not generated result is negative and find the 10's complement of the result, otherwise result is positive and discard carry.

**Example 1.10.13** Find the 10's complement of the following

$$1) (935)_{11} \quad 2) (6106)_{10}$$

GTU : Winter 16, Marks 4

Solution : 1)  $(935)_{11} = 9 \times 11^2 + 3 \times 11^1 + 5 \times 11^0 = 1089 + 33 + 5 = (1127)_{10}$

$$9999 - 1127 + 1 = 8873$$

... 10's complement of (935)<sub>11</sub>

$$2) 9999 - 6106 + 1 = 3894$$

... 10's complement of (6106)<sub>10</sub>

**Example 1.10.14** Perform  $(45)_{10} - (22)_{10}$  in BCD using 10's complement.

Solution :

**Step 1 :** Find 10's complement of 22.

$$\begin{array}{r} 10's \text{ complement of } 22 = 9's \text{ complement of } 22 + 1 \\ = (99 - 22) + 1 = 78 \end{array}$$

**Step 2 :** Add 46 and 10's complement of 22.

$$\begin{array}{r} + 46 \\ - 22 \\ \hline 24 \end{array} \quad \begin{array}{r} 1 \\ 0 1 0 0 0 1 1 0 \\ 0 1 1 1 1 0 0 0 \\ 1 0 1 1 1 1 1 1 0 \\ + 0 1 1 0 0 1 1 0 \\ \hline 0 0 1 0 0 1 0 0 0 \end{array}$$

Discard carry  X

Carry (46)<sub>BCD</sub>  
(78)10's complement of (22)<sub>BCD</sub>

Invalid BCD numbers  
Add 6 in each digit

Result 2      4

Since there is a carry the result is positive and true.

$$(46)_{BCD} - (22)_{BCD} = (24)_{BCD}$$

**Example 1.10.15** Perform  $(24)_{10} - (56)_{10}$  in BCD using 10's complement.

Solution :

Step 1 : Find 10's complement of 56.

$$\begin{aligned} 10's \text{ complement of } 56 &= 9's \text{ complement of } 56 + 1 \\ &= (99 - 56) + 1 = 44 \end{aligned}$$

Step 2 : Add  $(24)_{10}$  and 10's complement of 56.

24	1				Carry (24)BCD
	0	0	1	0	
- 56	*	0	1	0	0
- 32		0	0	1	1

10's complement of  $(56)_{BCD}$   
 $(68)_{BCD}$

Since carry is 0 the answer is negative.

Step 3 : Take 10's complement of answer.

$$\begin{aligned} 10's \text{ complement of } 68 &= 9's \text{ complement of } 68 + 1 \\ &= (99 - 68) + 1 = 32 \end{aligned}$$

$$(24)_{10} - (56)_{10} = -(32)_{10}$$

#### Examples for Practice

**Example 1.10.16** Represent the decimal number 6 in BCD code.

[Ans. : 0110]

**Example 1.10.17** Perform the following subtractions of BCD numbers using 9's complement : i)  $68 - 24$  ii)  $24 - 29$

#### 1.10.3 Other 4-bit BCD Codes

##### 1.10.3.1 2-4-2-1 Codes

The 2-4-2-1 BCD is another self complementing code or reflective code. Its 4-bit code groups are weighted. In this case, the weights are 2-4-2-1, meaning that bit 1 and bit 3 have the same weight (2). It is sometimes referred to as 2'-4-2-1 code, where the asterisk simply distinguishes one position with weight 2 from the other. Since two positions have the same weight, there are two possible

Decimal digit	2-4-2-1 Code
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1

Table 1.10.3 2-4-2-1 code

TECHNICAL PUBLICATIONS™ An up thrust for knowledge

bit patterns that could be used to represent some decimal digits, but only one of those patterns is actually assigned. Table 1.10.3 shows the code assignments for 2-4-2-1 code.

##### Reflective Property of 2421 Code

The Table 1.10.4 shows the reflective or self complementing property of 2421 code. It shows that, in 2421 code, code for 9 is complement of code 0, code for 8 is complement of code 1 and so on.

##### 1.10.3.2 Other Weighted BCD Codes

There are various other weighted 4-bit BCD codes, each developed to have certain properties useful for special applications. Table 1.10.5 shows these codes, identified by the weights assigned to their bit positions. The 7421 code is somewhat different than the other codes. When 1 occurs in either of the two rightmost positions means that the weight of that position is subtracted, rather than added, to determine the decimal value.

Decimal	Complement relation	Decimal
0	→ 0 0 0 0	9
	1 1 1 1 ←	
1	→ 0 0 0 1	8
	1 1 1 0 ←	
2	→ 0 0 1 0	7
	1 1 0 1 ←	
3	→ 0 0 1 1	6
	1 1 0 0 ←	
4	→ 0 1 0 0	5
	1 0 1 1 ←	

Table 1.10.4 Reflective property of 2421 code

Decimal	3 3 2 1	4 2 2 1	5 2 1 1	5 3 1 1	5 4 2 1	6 3 1 1	7 4 2 1	7 4 2 1
0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1	0 0 0 1	0 0 0 1	0 0 0 1	0 0 0 1	0 0 0 1	0 1 1 1
2	0 0 1 0	0 0 1 0	0 0 1 1	0 0 1 1	0 0 1 0	0 0 1 1	0 0 1 0	0 1 1 0
3	0 0 1 1	0 0 1 1	0 1 1 0	0 1 0 0	0 0 1 1	0 1 0 0	0 0 1 1	0 1 0 1
4	0 1 0 1	1 0 0 0	0 1 1 1	0 1 0 1	0 1 0 0	0 1 0 1	0 1 0 0	0 1 0 0
5	1 0 1 0	0 1 1 1	1 0 0 0	1 0 0 0	1 0 0 0	0 1 1 1	0 1 0 1	1 0 1 0
6	1 1 0 0	1 1 0 0	1 0 0 1	1 0 0 1	1 0 0 1	1 0 0 0	0 1 1 0	1 0 0 1
7	1 1 0 1	1 1 0 1	1 1 0 0	1 0 1 1	1 0 1 0	1 0 0 1	1 0 0 0	1 0 0 0
8	1 1 1 0	1 1 1 0	1 1 1 0	1 1 0 0	1 0 1 1	1 0 1 1	1 0 0 1	1 1 1 1
9	1 1 1 1	1 1 1 1	1 1 1 1	1 1 0 1	1 1 0 0	1 1 0 0	1 0 1 0	1 1 1 0

Table 1.10.5 Weighted 4-bit BCD codes

TECHNICAL PUBLICATIONS™ An up thrust for knowledge

**Example 1.10.18** Represent  $(7)_{10}$  using all the weighted 4-bit BCD codes listed in the Table 1.10.5.

**Solution :**

1. 3211 code $(7)_{10} = 3 + 3 + 2 + 1$ $\begin{array}{cccc} \downarrow & \downarrow & \downarrow & \downarrow \\ 1 & 1 & 0 & 1 \end{array} = (1101)_{3211\text{BCD}}$	2. 4221 code $(7)_{10} = 4 + 2 + 2 + 1$ $\begin{array}{cccc} \downarrow & \downarrow & \downarrow & \downarrow \\ 1 & 0 & 1 & 1 \end{array} = (1011)_{4221\text{BCD}}$
3. 5221 code $(7)_{10} = 5 + 2 + 2 + 1$ $\begin{array}{cccc} \downarrow & \downarrow & \downarrow & \downarrow \\ 1 & 1 & 0 & 0 \end{array} = (1100)_{5221\text{BCD}}$	4. 5311 code $(7)_{10} = 5 + 3 + 1 + 1$ $\begin{array}{cccc} \downarrow & \downarrow & \downarrow & \downarrow \\ 1 & 0 & 1 & 1 \end{array} = (1011)_{5311\text{BCD}}$
5. 5421 code $(7)_{10} = 5 + 4 + 2 + 1$ $\begin{array}{cccc} \downarrow & \downarrow & \downarrow & \downarrow \\ 1 & 1 & 0 & 1 \end{array} = (1010)_{5421\text{BCD}}$	6. 6311 code $(7)_{10} = 6 + 3 + 1 + 1$ $\begin{array}{cccc} \downarrow & \downarrow & \downarrow & \downarrow \\ 1 & 0 & 0 & 1 \end{array} = (1001)_{6311\text{BCD}}$
7. 7421 code $(7)_{10} = 7 + 4 + 2 + 1$ $\begin{array}{cccc} \downarrow & \downarrow & \downarrow & \downarrow \\ 1 & 0 & 0 & 0 \end{array} = (1000)_{7421\text{BCD}}$	8. 7421 code $(7)_{10} = 7 + 4 - 2 - 1$ $\begin{array}{cccc} \downarrow & \downarrow & \downarrow & \downarrow \\ 1 & 0 & 0 & 0 \end{array} = (1000)_{7421\text{BCD}}$

#### Example for Practice

**Example 1.10.19 :** Represent the decimal number 6 in 8421 and 2421 codes.

[Ans. : 1010, 1100]

#### 1.10.4 Excess-3 Code

- The excess-3 code can be derived from the natural BCD code by adding 3 to each coded number.
- For example, decimal 12 can be represented in BCD as 0001 0010. Now adding 3 to each digit we get Excess-3 code as 0100 0101 (12 in decimal). It is a non-weighted code.

- Table 1.10.6 shows excess-3 codes to represent single decimal digit. It is sequential code because we get any code word by adding binary 1 to its previous code word as shown in the Table 1.10.6.
- In excess-3 code we get 9's complement of a number by just complementing each bit. Due to this excess-3 code is called self-complementing code or reflective code.

Decimal digit	Excess-3 code
0	0 0 1
1	0 1 0
2	0 1 1
3	0 0 0
4	1 0 0
5	1 0 1
6	1 1 0
7	1 1 1
8	1 0 0
9	1 0 1

Table 1.10.6 Excess-3 code

**Example 1.10.20** Find the XS-3 code of 37.

GTU - Summer 16, Mat. 1

**Solution :**

3	7
0 0 1 1 0 1 1 1	BCD
0 1 1 0 1 0 1 0	BCD + 3 = XS-3 code

$$(37)_{10} = (0110\ 1010)_{XS-3}$$

Fig. 1.10.5

**Example 1.10.21** Find the excess-3 code and its 9's complement for following decimal numbers. a) 592 b) 403

**Solution :**

- a) By referring Table 1.10.6.

$592_{10} =$	$1\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 1\ 0\ 1$	Complement of each other
9's complement of $592_{10} =$	$0\ 1\ 1\ 1\ 0\ 0\ 1\ 1\ 0\ 1\ 0$	

## b) By referring Table 1.10.6.

$$403_{10} = \begin{array}{ccccccccc} 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \end{array}$$

Complement of each other  
9's complement of 403<sub>10</sub>

## 1.10.4.1 Excess-3 Addition

To perform excess-3 addition we have to

- Add two excess-3 numbers using binary addition.

- If carry = 1 → add 3 ( $0\ 0\ 1\ 1$ )<sub>2</sub> to the sum of two digits.
- = 0 → subtract 3 ( $0\ 0\ 1\ 1$ )<sub>2</sub> from the sum.

**Example 1.10.22** Perform the excess-3 addition of a) 8, b) 1, 2.

Solution :

a) 8 + 6

1	1	1	1
+	1	0	1
1	0	0	1

Carry      Excess-3 for 8  
Excess-3 for 6  
+      Add 3  
Result in decimal

b) 1 + 2

1	1	1	1
+	0	1	0
0	1	0	1

Carry      Excess-3 for 1  
Excess-3 for 2  
+      Sub 3  
Result in decimal

## 1.10.4.2 Excess-3 Subtraction

- To perform excess-3 subtraction we have to
  - Complement the subtrahend.
  - Add complemented subtrahend to minuend.
  - If carry = 1 Result is positive. Add 3 ( $0\ 0\ 1\ 1$ )<sub>2</sub> and end-around carry.
  - If carry = 0 Result is negative. Subtract 3 ( $0\ 0\ 1\ 1$ )<sub>2</sub>.

**Example 1.10.23** Perform the excess-3 subtraction of a) 8 - 5, b) 5 - 8.

Solution : a) 8 - 5

1	1	1	1
+	1	0	1
1	0	1	1

Carry      Excess-3 for 8  
Complement of 5 in excess-3  
Add 3  
Add end-around carry  
Excess-3 for 3

TECHNICAL PUBLICATIONS® - An up trend for knowledge

**Note** In excess-3, 9's complement and complement of any number is same. So this excess-3 subtraction method is also known as excess-3 subtraction using 9's complement.

b) 5 - 8

1	0	0	0
+	0	1	0
0	1	1	0

Carry      Excess-3 for 5  
Complement of 8 in excess-3  
Subtract 3  
Excess-3 for -3

**Example 1.10.24** Perform each of the following operations in excess-3 code.

a)  $\begin{array}{r} 16 \\ - 29 \\ \hline \end{array}$

b)  $\begin{array}{r} 21 \\ - 12 \\ \hline \end{array}$

Solution :

a) 16      + 29      3

1	1	1	1
+	0	1	0
0	1	0	1

Carry      Excess-3 for 16  
Excess-3 for 29  
Propagates carry  
Carry  
Add 3 to correct 0 1 0 1  
Subtract 3 to correct 1 0 1 0  
Excess-3 for 45  
Result

1	1	1	1
+	0	1	0
0	1	0	1

b) 21      - 12      9

1	1	1	1
-	1	0	1
0	1	0	0

Carry      Excess-3 for 21  
Complement of 12 in excess-3  
Add 3 and subtract 3  
Add end-around carry  
Excess-3 for 9

## Examples for Practice

**Example 1.10.25** Represent decimal number 327 in Excess-3 code. [Ans.: 0110 0101 1010]

TECHNICAL PUBLICATIONS® - An up trend for knowledge

**Example 1.10.26 :** What is Excess-3 code of binary numbers :  
0010 B, 0110 B and 0111 B ? [Ans. : 1001, 1001, 1010]

**Example 1.10.27 :** Represent the decimal number 6 in Excess-3 code. [Ans. : 1001]

### 1.10.5 Gray Code

- Gray code is a non-weighted code and is a special case of unit-distance code.
- In unit-distance code, bit patterns for two consecutive numbers differ in only one bit position. These codes are also called cyclic codes.
- The Table 1.10.7 shows the bit patterns assigned for gray code from decimal 0 to decimal 15.
- As shown in the Table 1.10.7 for gray code any two adjacent code groups differ only in one bit position.
- Reflective Property :** The gray code is also called reflected code. Notice that the two least significant bits for  $4_{10}$  through  $7_{10}$  are the mirror images of those for  $0_0$  through  $3_{10}$ . Similarly, the three least significant bits for  $8_{10}$  through  $15_{10}$  are the mirror images of those for  $0_{10}$  through  $7_{10}$ .
- In general, the  $n$  least significant bits for  $2^n$  through  $2^{n+1}-1$  are the mirror images of those for 0 through  $2^n-1$ .

Decimal code	Gray code
0	000
1	001
2	011
3	010
4	011
5	111
6	010
7	010
8	110
9	110
10	111
11	111
12	101
13	101
14	100
15	100

Table 1.10.7 Gray code

### 1.10.5.1 Application of Gray Code

- Let us consider an application where 3-bit binary code is provided to indicate position of the rotating disk with the help of brushes.
- When brushes are on the black portion, they output a 1. When on the white portion, they output a 0.
- Now consider what happens when the brushes are on the 111 sector and almost ready to enter the 000 sector.

- If one brush is slightly ahead of the other, say the 3rd brush, then the position is indicated by a 011 instead of a 111 or 000. This results, a 180° error in the disk position.
- Since it is physically impossible to have all the brushes precisely aligned, some error will always be present at the edges of the sectors.
- If we use gray code to represent disk position then error due to improper brush alignment can be reduced. This is because the gray code assures that only one bit will change each time the decimal number is incremented.

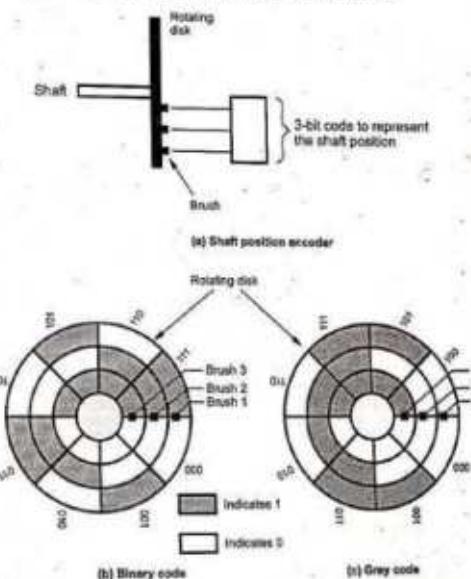


Fig. 1.10.8 Position indication using binary and gray codes

- So in 3-bit code, error may occur due to one bit position. Other two bit positions of two adjacent sectors are always same and hence there is no possibility of error.
- Therefore in 3-bit code probability of error is reduced up to 66 %. In case of 4-bit code it is reduced up to 75 %. This is an important advantage of gray code.

**1.10.5.2 Gray to Binary Conversion**

**Steps for gray to binary code conversion**

1. The Most Significant Bit (MSB) of the binary number is the same as the Most Significant Bit (MSB) of the gray code number. So write down MSB as it is.
2. To obtain the next binary digit, perform an exclusive-OR-operation between the bit just written down and the next gray code bit. Write down the result.
3. Repeat step 2 until all gray code bits have been exclusive-ORed with binary digits.

A	B	A ⊕ B Exclusive OR operation
0	0	0
0	1	1
1	0	1
1	1	0

Table 1.10.8 Exclusive OR operation

**Example 1.10.26** Convert gray code 101011 into its binary equivalent.

**Solution :**

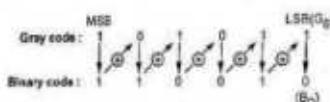


Fig. 1.10.7

$$(101011)_{\text{gray}} = (111001)_{\text{binary}}$$

In general we can perform gray code to binary code conversion as shown below :

$$\begin{aligned} B_n (\text{MSB}) &= G_n (\text{MSB}) & B_2 &= B_3 \oplus G_2 \\ B_{n-1} &= B_n \oplus G_{n-1} & \dots & B_1 = B_2 \oplus G_1 \\ B_{n-2} &= B_{n-1} \oplus G_{n-2} & B_0 &= B_1 \oplus G_0 \end{aligned}$$

**Example 1.10.29** Convert the gray code 1101 to binary.

GTU : Summer-16, Mark 1

**Solution :**

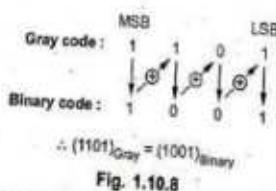


Fig. 1.10.8

TECHNICAL PUBLICATIONS™ - An up thrust for knowledge

**1.10.5.3 Binary to Gray Code Conversion**

**Steps for binary to gray code conversion**

1. The MSB of the gray code is the same as the MSB of the binary number. So write down MSB as it is.
2. To obtain the next gray digit, perform an exclusive-OR-operation between previous and current binary bit. Write down the result.
3. Repeat step 2 until all binary bits have been exclusive-ORed with their previous ones.

**Example 1.10.30** Convert 10111011 in binary into its equivalent gray code.

**Solution :**

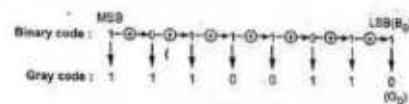


Fig. 1.10.9

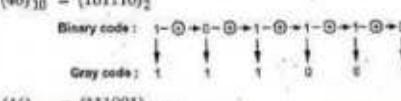
$$(10111011)_2 = (11100110)_{\text{gray}}$$

In general we can perform binary code to gray code conversion as shown below :

$$\begin{aligned} G_n (\text{MSB}) &= B_n (\text{MSB}) & G_2 &= B_3 \oplus B_2 \\ G_{n-1} &= B_n \oplus B_{n-1} & \dots & G_1 = B_2 \oplus B_1 \\ G_{n-2} &= B_{n-1} \oplus B_{n-2} & G_0 &= B_1 \oplus B_0 \end{aligned}$$

**Example 1.10.31** Encode the decimal number 46 to Gray code.

**Solution :**  $(46)_{10} = (101110)_2$

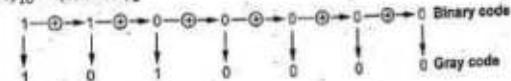


$$(46)_{10} = (11100110)_{\text{gray}}$$

**Example 1.10.32** Convert  $(96)_{10}$  to its equivalent gray code and Ex-3 code.

GTU : Summer-15, Marks 4

**Solution :**  $(96)_{10} = (1100000)_2$



$$(96)_{10} = (1010000)_\text{Gray} = (1100\ 1001)_{\text{Ex-3}}$$

TECHNICAL PUBLICATIONS™ - An up thrust for knowledge

**Example for Practice**

**Example 1.10.33 :** Represent the decimal number 6 in Gray code. [Ans. : 0101]

**1.10.5 Alphanumeric Codes**

- The codes which consists of both numbers and alphabetic characters are called alphanumeric codes.
- The most commonly used alphanumeric codes are : ASCII (American Standard Code for Information Interchange) and EBCDIC (Extended Binary Coded Decimal Interchange Code).
- These codes consists of symbols to represent -
  - 26 alphabets with capital and small letters.
  - Numbers from 0 to 9.
  - Punctuation marks and other symbols.

**1.10.6.1 ASCII**

- ASCII code is a seven-bit code in which the decimal digits are represented by the BCD code preceded by 011.
- Since it is a 7-bit code, it represent  $2^7 = 128$  symbols.
- The letter of the alphabet and other symbols and instructions are represented by other code combination as shown in Table 1.10.9. For example, ASCII code for  $N = (4E)_H = (1001110)_2$ .

	MSBs							
	000 (0)	001 (1)	010 (2)	011 (3)	100 (4)	101 (5)	110 (6)	111 (7)
0000 (0)	NUL	DLE	SP	0	@	P		P
0001 (1)	SOH	DC <sub>1</sub>	!	1	A	Q	a	q
0010 (2)	STX	DC <sub>2</sub>	*	2	B	R	b	r
0011 (3)	ETX	DC <sub>3</sub>	#	3	C	S	c	s
0100 (4)	EOT	DC <sub>4</sub>	\$	4	D	T	d	t
0101 (5)	ENQ	NAK	%	5	E	U	e	u
0110 (6)	ACK	SYN	&	6	F	V	f	v
0111 (7)	BEL	ETB	*	7	G	W	g	w
1000 (8)	BS	CAN	(	8	H	X	h	x

TECHNICAL PUBLICATIONS® - An up thrust for knowledge

1001 (9)	HT	EM	)	9	L	Y	I	Y
1010 (A)	LF	SUB	*	:	Z	I	z	
1011 (B)	VI	ESC	*	:	K	I	k	I
1100 (C)	PF	PS	,	L	X	I	l	
1101 (D)	CR	GS	-	M	J	m	j	
1110 (E)	SO	RS	>	N	t	n	-	
1111 (F)	SI	US	/	O	-	o	DEL	

Table 1.10.9 American standard code for information Interchange

**Definition of control abbreviations :**

ACK	Acknowledge	PS	Form separator
BEL	Bell	GS	Group separator
BS	Backspace	HT	Horizontal tab
CAN	Cancel	LF	Line feed
CR	Carriage return	NAK	Negative acknowledge
DC <sub>1</sub> -DC <sub>4</sub>	Direct control	NUL	Null
DEL	Delete idle	RS	Record separator
DLE	Data link escape	SI	Shift in
EM	End of medium	SO	Shift out
ENQ	Enquiry	SOH	Start of heading
EOT	End of transmission	STX	Start text
ESC	Escape	SUB	Substitute
ETB	End of transmission block	SYN	Synchronous idle
ETX	End text	US	Unit separator
FF	Form feed	VT	Vertical tab

**Note** The hexadecimal digit representing each bit pattern is shown in parentheses.**Example 1.10.34** Write the ASCII code for 'ELECTRONICS'.**Solution :**

45, 4C, 45, 43, 54, 52, 4F, 4E, 49, 43, 53 i.e.,

1000101 1001100 1000101 1000011 1010100 1010010

1001111 1001110 1001001 1000011 1010011

**Example 1.10.35** Encode the word 'BINARY' in ASCII form.

Solution : 42, 49, 4E, 41, 52, 59 i.e.

1000010 1001001 1001110 1000001 1010010 1011001

**Example 1.10.36** A line printer is capable of printing 132 characters in a single-line and each character is represented by ASCII code. How many bits are required to print each line?

Solution : ASCII code is a 7-bit code. Therefore, to print one line we require

$$132 \times 7 = 924 \text{ bits.}$$

**Example 1.10.37** How many bits of memory are required for storing 100 names of a group of people assuming that no name occupies more than 20 characters (including space)? Assume 7-bit ASCII code with parity bit.

Solution : Space reserved for one name =  $20 \times (7+1) = 160$  bits. Bits required for storing 100 names =  $160 \times 100 = 16000$  bits.

#### Examples for Practice

**Example 1.10.38 :** Find gray codes for the following binary numbers :

i) 11001100    ii) 01011110    [Ans. : i) 10101010 ii) 01110001]

**Example 1.10.39 :** What will be the gray code of binary number 1100 B, 0111 B and 1101 B ?

[Ans. : 1010, 0100, 1011]

#### Review Questions

- Give classification of codes.
- Define code. List various codes.
- What is meant by weighted and non-weighted coding?
- Define reflective and sequential codes?
- Write a note on BCD code.
- What are the rules for BCD addition? Illustrate with the help of example.
- Explain the steps for BCD subtraction using 9's complement method.
- Explain the steps for BCD subtraction using 10's complement method.
- What is gray code? What is the advantage of gray codes over the binary number sequence?
- Show that the excess-3 code is self-complementing.
- Write a note on excess-3 code.
- Write a brief note on gray codes. Also discuss methods for conversion from gray to binary code and vice versa.
- State the application of gray code.
- Explain ASCII code with the help of example.
- What is self complementing code? Represent (472)<sub>10</sub> in 2421 self complementing code.

GTU : May-14, Marks 7

GTU : Summer-15, Marks 2

#### 1.11 Error Detecting and Correcting Codes

- When the digital information in the binary form is transmitted from one circuit or system to another circuit or system an error may occur. This means a signal corresponding to 0 may change to 1 or vice-versa due to presence of noise. To maintain the data integrity between transmitter and receiver, extra bit or more than one bit are added in the data. These extra bits allow the detection and sometimes correction of error in the data. The data along with the extra bit/bits forms the code. Codes which allow only error detection are called error detecting codes and codes which allow error detection and correction are called error detecting and correcting codes.

##### 1.11.1 Parity Bit

- A parity bit is used for the purpose of detecting errors during transmission of binary information. A parity bit is an extra bit included with a binary message to make the number of 1s either odd or even. The message, including the parity bit is transmitted and then checked at the receiving end for errors. An error is detected if the checked parity does not correspond with the one transmitted. The circuit that generates the parity bit in the transmitter is called a parity generator and the circuit that checks the parity in the receiver is called a parity checker.
- In even parity the added parity bit will make the total number of 1s an even amount. In odd parity the added parity bit will make the total number of 1s an odd amount.

\* Table 1.11.1 shows the 3-bit message with even parity and odd parity.

3-bit Message			Message with Odd Parity		Message with Even Parity	
A	B	C	Message	Parity	Message	Parity
0	0	0	000	1	000	0
0	0	1	001	0	001	1
0	1	0	010	0	010	1
0	1	1	011	1	011	0
1	0	0	100	0	100	1
1	0	1	101	1	101	0
1	1	0	110	1	110	0
1	1	1	111	0	111	1

Table 1.11.1 Message with even and odd parity

**Example 1.11.1** Write a ASCII code for the decimal digit 9 with an even parity. Place parity bit in the most significant position.

**Solution :** The 7-bit ASCII code for the decimal digit 9 is 0111001. This requires the addition of a 0 in the most significant place to give even parity as shown.

$$\text{Added parity bit} \rightarrow 0111001$$

**Example 1.11.2** Write a ASCII code for the alphabet 'A' with an odd parity. Place parity bit in the most significant position.

**Solution :** The 7-bit ASCII code for the alphabet 'A' is 1000001. This requires the addition of a 1 in the most significant place to give odd parity, as shown.

$$\text{Added parity bit} \rightarrow 1100001$$

At the receiving end, message with parity bit is received. Every time it is checked for parity. When parity error is detected, receiver requests for transmitter to re-transmit the message.

**Example 1.11.3** The received code is 10000001. Check whether code is correctly received or not if odd parity is used.

**Solution :** The received code has even parity hence the code is not received correctly.

#### More about parity error detection

As a general rule in a digital system where the transmission system is relatively short, it may be assumed that the probability of a single-bit error is small and that of a double-bit error and higher order errors is extremely small. The parity error detection system just described detects any odd number of errors. However, it cannot detect an even number of errors because such errors will not destroy the parity of the transmitted group of bits.

#### 1.11.2 Hamming Code

- Hamming code not only provides the detection of a bit error, but also identifies which bit is in error so that it can be corrected. Thus Hamming code is called error detecting and correcting code. The code uses a number of parity bits (dependent on the number of information bits) located at certain positions in the code group. Follows sections describe how Hamming code can be constructed for single error correction.

#### Number of Parity Bits

- As mentioned earlier, number of parity bits depend on the number of information bits. If the number of information bits is designed  $x$ , then the number of parity bits,  $P$  is determined by the following relationship :

$$2^P \geq x + P + 1$$

... (1.11.1)

For example, if we have four information bit, i.e.  $x = 4$ , then  $P$  is found by trial and error using equation 1. Let  $P = 2$ . Then

$$2^P = 2^2 = 4$$

$$\text{and } x + P + 1 = 4 + 2 + 1 = 7$$

- Since  $2^P$  must be equal to or greater than  $x + P + 1$ , the relationship in equation (1.9.1) is not satisfied. Hence we have to try with next value of  $P$ . Let  $P = 3$ .

$$\text{Then } 2^P = 2^3 = 8$$

$$\text{and } x + P + 1 = 4 + 3 + 1 = 8$$

- This value of  $P$  satisfies the relationship given in equation (1.9.1), and therefore we can say that three parity bits are required to provide single error correction for four information bits.

#### Locations of the Parity Bits in the Code

- Now we know that how to calculate the number of parity bits required to provide single error correction for given number of information bits. In our example we have four information bits and three parity bits. Therefore, the code is of seven bits. The right-most bit is designated bit 1, the next bit is bit 2, and so on, as shown below :

Bit 7, Bit 6, Bit 5, Bit 4, Bit 3, Bit 2, Bit 1

- The parity bits are located in the positions that are numbered corresponding to ascending powers of two (1, 2, 4, 8 ...). Therefore, for 7-bit code, locations for parity bits and information bit are as shown below :

$D_7, D_6, D_5, P_4, D_3, P_2, P_1$

where symbol  $P_n$  designates a particular parity bit,  $D_n$  designates a particular information bit, and  $n$  is the location number.

#### Assigning Values to Parity Bits

- Now we know the format of the code. Let us see how to determine 1 or 0 value to each parity bit. In Hamming code, each parity bit provides a check on certain other bits in the total code, therefore, we must know the value of these others in order to assign the parity bit value. To do this, we must write the binary number for each decimal location number as shown in the third row of Table 1.9.2.

Bit designation	$D_7$	$D_6$	$D_5$	$P_4$	$D_3$	$P_2$	$P_1$
Bit location	7	6	5	4	3	2	1
Binary location number	111	110	101	100	011	010	001

Information bits ( $D_i$ )	1	0	1	1
Parity bits ( $P_j$ )			0	0

Table 1.11.2 Bit position table for a seven bit error correcting code

**Assignment of  $P_1$ :**

- Looking at the Table 1.11.2 we can see that the binary location number of parity bit  $P_1$  has a 1 for its right-most digit. This parity bit checks all bit locations, including itself, that have 1s in the same location in the binary location numbers. Therefore, parity bit  $P_1$  checks bit locations 1, 3, 5 and 7, and assigns  $P_1$  according to even or odd parity. For even parity Hamming code, it assigns  $P_1$  such that bit locations 1, 3, 5, and 7 will have even parity.

**Assignment of  $P_2$ :**

- Looking at the Table 1.11.2 we can see that the binary location number of parity bit  $P_2$  has a 1 for its middle bit. This parity bit checks all bit locations, including itself, that have 1s in the middle bit. Therefore, parity bit  $P_2$  checks bit locations 2, 3, 6 and 7 and assigns  $P_2$  according to even or odd parity.

**Assignment of  $P_4$ :**

- Looking at the Table 1.11.2 we can see that the binary location number of parity bit  $P_4$  has a 1 for its left-most digit. This parity bit checks all bit locations, including itself, that have 1s in the left-most bit. Therefore, parity bit  $P_4$  checks bit locations 4, 5, 6 and 7 and assigns  $P_4$  according to even and odd parity.

**Example 1.11.4** Encode the binary word 1011 into seven bit even parity Hamming code.

Solution :

**Step 1 :** Find the number of parity bits required. Let  $P = 3$ , then

$$2^P = 2^3 = 8$$

$$x + P + 1 = 4 + 3 + 1 = 8$$

Three parity bits are sufficient.

$$\therefore \text{Total code bits} = 4 + 3 = 7$$

**Step 2 :** Construct a bit location table

Bit designation	$D_7$	$D_6$	$D_5$	$P_4$	$D_3$	$P_2$	$P_1$
Bit location	7	6	5	4	3	2	1
Binary location number	111	110	101	100	011	010	001

Information bits	1	0	1	1
Parity bits			0	0

**Step 3 :** Determine the parity bits

For  $P_1$  : Bit locations 3, 5 and 7 have three 1s and therefore to have an even parity  $P_1$  must be 1.

For  $P_2$  : Bit locations 3, 6 and 7 have two 1s and therefore to have an even parity  $P_2$  must be 0.

For  $P_4$  : Bit locations 5, 6 and 7 have two 1s and therefore to have an even parity  $P_4$  must be 0.

**Step 4 :** Enter the parity bits into the table to form a seven bit Hamming code = 1010101.

**Example 1.11.5** Determine the single error-correcting code for the information code 1011111 for odd parity.

Solution :

**Step 1 :** Find the number of parity bits required. Let  $P = 3$ .

$$\text{Then } 2^P = 2^3 = 8$$

$$\therefore x + P + 1 = 5 + 3 + 1 = 9$$

This will not work. Try  $P = 4$ . Then

$$2^P = 2^4 = 16$$

$$x + P + 1 = 5 + 4 + 1 = 10$$

Equation (1.9.1) is satisfied and hence four bits are sufficient.

$$\therefore \text{Total code bit} = 5 + 4 = 9$$

**Step 2 :** Construct a bit location table

Bit designation	$D_9$	$D_8$	$D_7$	$D_6$	$D_5$	$P_4$	$D_3$	$P_2$	$P_1$
Bit location	9	8	7	6	5	4	3	2	1
Binary location number	1001	1000	0111	0110	0101	0100	0011	0010	0001
Information bits	1		0	1	1			1	
Parity bits		0				1		1	0

**Step 3 : Determine the Parity Bits**

For  $P_1$  : Bit locations 3, 5, 7 and 9 have three 1s and therefore to have odd parity  $P_1$  must be 0.

For  $P_2$  : Bit locations 3, 6, 7 have two 1s and therefore to have an odd parity  $P_2$  must be 1.

For  $P_3$  : Bit locations 5, 6 and 7 have two 1s and therefore to have an odd parity  $P_3$  must be 1.

For  $P_4$  : Bit locations 8 and 9 and must be 0 to have an odd parity.

For  $P_8$  : Bit  $P_8$  checks bit locations 8 and 9 and must be 0 to have an odd parity.

**Step 4 :** Enter the parity bits into the table to form a nine bit Hamming code = 100111110.

**1.11.3 Detecting and Correcting an Error**

- In the last section we have seen how to construct Hamming code for given number of information bits. Now we will see how to use it to locate and correct an error. To do this, each parity bit, along with its corresponding group of bits must be checked for proper parity. The correct result of individual parity check is marked by 0 whereas wrong result is marked by 1. After all parity checks, binary word is formed taking resulting bit for  $P_1$  as LSB. This word gives bit location where error has occurred. If word has all bits 0 then there is no error in the Hamming code.

**Example 1.11.6** Assume that the even parity Hamming code in example (0 1 1 0 0 1 1) is transmitted and that 0 1 0 0 0 1 1 is received. The receiver does not know what was transmitted. Determine bit location where error has occurred using received code.

**Solution :**

**Step 1 :** Construct the bit location table.

Bit designation	$D_7$	$D_4$	$D_5$	$P_4$	$D_3$	$P_2$	$P_1$
Bit location	7	6	5	4	3	2	1
Binary location number	111	110	101	100	011	010	001
Received code	0	1	0	0	0	1	1

**Step 2 :** Check for parity bits

For  $P_1$  :  $P_1$  checks locations 1, 3, 5, and 7.

There is one 1 in the group.

∴ Parity check for even parity is wrong → 1 (LSB)

For  $P_2$  :  $P_2$  checks locations 2, 3, 6 and 7.

There are two 1s in the group

∴ Parity check for even parity is correct → 0

For  $P_3$  :  $P_3$  checks locations 4, 5, 6 and 7.

There are one 1 in the group

∴ Parity check for even parity is wrong → 1

The resultant word is 1 0 1. This says that the bit in the number 5 location is in error. It is 0 and should be a 1. Therefore, the correct code is 0 1 1 0 0 1 1, which agrees with the transmitted code.

**Example 1.11.7** The Hamming code 1 0 1 1 0 1 1 is received. Correct it if any errors.

There are four parity bits and odd parity is used.

**Solution :**

**Step 1 :** Construct a bit location table

Bit designation	$D_9$	$P_8$	$D_7$	$D_6$	$D_5$	$P_4$	$D_3$	$P_2$	$P_1$
Bit location	9	8	7	6	5	4	3	2	1
Binary location number	1001	1000	0111	0110	0101	0100	0011	0010	0001
Received code	1	0	1	1	0	1	1	0	1

**Step 2 :** Check for parity bits

For  $P_1$  :  $P_1$  checks locations 1, 3, 5, 7 and 9.

There are four 1s in the group

∴ Parity check for odd parity is wrong → 1 (LSB)

For  $P_2$  :  $P_2$  checks locations 2, 3, 6 and 7.

There are three 1s in the group

∴ Parity check for odd parity is correct → 0

For  $P_3$  :  $P_3$  checks locations 4, 5, 6 and 7.

There are three 1s in the group

∴ Parity check for odd parity is correct → 0

For  $P_4$  :  $P_4$  checks locations 8 and 9.

There is one 1 in the group

∴ Parity check for odd parity is correct → 0

The resultant word is 0 0 0 1. This says that the bit in the number 1 location is in error. It is 1 and should be 0. Therefore, the correct code is 1 0 1 1 0 1 1 0 0.

#### 1.11.4 Single Error Correction Plus Double Error Detection

A hamming code as explained above provides for the detection and correction of only a single error. With a slight modification, it is possible to construct hamming code for single-error correction and double-error detection. A one more parity bit is added in the hamming code to ensure hamming code (including all parity bits) contains an even number of 1's. The added parity bit is not used in determining the values of the other parity bits. The resulting hamming code enables single error correction and double error detection.

When overall parity bit is correct, there is no single error during the transmission of the code. If overall parity bit is incorrect, then there is single error and the bit position of the error can be indicated by binary number formed after checking the parity bits. Hence, single error correction can be achieved. If overall parity bit is correct and binary number formed after checking the parity bits is other than 0-0-0, there are two errors. In this case, double error detection is achieved. However, no correction is possible.

#### Examples for Practice

**Example 1.11.8 :** Construct hamming codes for the decimal numbers 1, 4, 8.  
 [Ans. : Considering odd parity 1 : 0001100, 4 : 0100001, 8 : 1000000]

**Example 1.11.9 :** Generate the even parity Hamming codes for following binary data.  
 a) 1 1 0 1      b) 1 0 0 1      [Ans. : a) 1100 110 b) 1001 100]

**Example 1.11.10 :** A seven bit Hamming code is received as 1 1 1 1 1 0 1. What is the correct code ?  
 [Ans. : 1111111]

**Example 1.11.11 :** Generate Hamming code for the following binary data. i) 1101 ii) 1001  
 [Ans. : i) 1100110 ii) 1001100]

**Example 1.11.12 :** Determine whether single error has occurred and if so correct the error using Hamming code for 1100010.  
 [Ans. : Corrected code = 1100110]

#### Review Questions

1. What is parity bit ? Explain its use.
2. Explain the 7 bit hamming code.
3. Explain the procedure of generating hamming code from the information.
4. Explain how Hamming code is useful for correcting and detecting the errors using an example.

5. Explain error detection and correction.  
 6. What is hamming code ? How is the hamming code word tested and corrected ?

#### Oral Questions and Answers

- Q.1 Which system is used in digital computers because all electrical and electronic circuits can be made to respond to the states concept ?  
 Ans. : Binary number.
- Q.2 Which numbers are used to represent decimal numbers in computer ?  
 Ans. : BCD numbers.
- Q.3 Which number system uses binary numbers in most compressed form ?  
 Ans. : Hexadecimal numbers.
- Q.4 In which numeral every position has a value 2 times the value of the position to its right ?  
 Ans. : Binary.
- Q.5 Which number system has a base of 16 ?  
 Ans. : Hexadecimal number system.
- Q.6 What is meant by weighted and non-weighted coding ?  
 Ans. : Weighted codes : In weighted codes, each digit position of the number represents a specific weight. For example, in decimal code, if number is 567 then weight of 5 is 100, weight of 6 is 10 and weight of 7 is 1. In weighted binary codes each digit has a weight 8, 4, 2 or 1.

The codes 8421, 2421 and 5211 are all weighted codes. (Refer Tables 1.10.1, 1.10.3 and 1.10.5)

Non-weighted codes : Non-weighted codes are not assigned with any weight to each digit position, i.e., each digit position within the number is not assigned fixed value. Excess-3 and gray codes are the non-weighted codes.

- Q.7 In which type of codes the positional weights are not assigned ?  
 Ans. : Non-weighted code.



# 2

## Logic Gates and Boolean Algebra

### Contents

2.1 Logic Gates	.....	Dec-12, 13, May-13, 14,
2.2 Boolean Algebra	.....	Summer-15, 16, 17, 18,
	.....	Winter-14, 16, 18, ..... Marks 6

Oral Questions and Answers

(2 - 1)

**2.1 Logic Gates**

- The three primitive logic functions are represented by distinctive logic symbols known as basic gates : AND gate, OR gate and NOT gate.
- From these primitive logic functions we have derived logic functions and hence derived logic gates. There are NAND gate, NOR gate, EX-OR gate and EX-NOR gate.

**NOT gate (Inverter) :** The output is a complement of input.

Logic Diagram (Symbol)	Switch Equivalent	Truth Table												
 <b>Pin Diagram</b>	 <b>Input Output</b> <table border="1"> <tr><td>Switch open (Low)</td><td>Lamp ON (High)</td></tr> <tr><td>Switch close (High)</td><td>Lamp OFF (Low)</td></tr> </table>	Switch open (Low)	Lamp ON (High)	Switch close (High)	Lamp OFF (Low)	<b>Truth Table</b> <table border="1"> <thead> <tr><th>Input</th><th>Output</th></tr> </thead> <tbody> <tr><td>A</td><td>Y</td></tr> <tr><td>0</td><td>1</td></tr> <tr><td>1</td><td>0</td></tr> </tbody> </table> <b>Boolean Expression</b> $Y = \bar{A}$	Input	Output	A	Y	0	1	1	0
Switch open (Low)	Lamp ON (High)													
Switch close (High)	Lamp OFF (Low)													
Input	Output													
A	Y													
0	1													
1	0													

**AND gate :** The output is high only when all inputs are high.

Logic Diagram (Symbol)	Switch Equivalent	Truth Table																											
 <b>Pin Diagram</b>	 <b>Input Output</b> <table border="1"> <tr><td>S<sub>1</sub></td><td>S<sub>2</sub></td><td>Output</td></tr> <tr><td>Open (Low)</td><td>Open (Low)</td><td>Lamp OFF (Low)</td></tr> <tr><td>Open (Low)</td><td>Close (High)</td><td>Lamp OFF (Low)</td></tr> <tr><td>Close (High)</td><td>Open (Low)</td><td>Lamp OFF (Low)</td></tr> <tr><td>Close (High)</td><td>Close (High)</td><td>Lamp ON (High)</td></tr> </table>	S <sub>1</sub>	S <sub>2</sub>	Output	Open (Low)	Open (Low)	Lamp OFF (Low)	Open (Low)	Close (High)	Lamp OFF (Low)	Close (High)	Open (Low)	Lamp OFF (Low)	Close (High)	Close (High)	Lamp ON (High)	<b>Truth Table</b> <table border="1"> <thead> <tr><th>Input</th><th>Output</th></tr> </thead> <tbody> <tr><td>A    B</td><td>Y</td></tr> <tr><td>0    0</td><td>0</td></tr> <tr><td>0    1</td><td>0</td></tr> <tr><td>1    0</td><td>0</td></tr> <tr><td>1    1</td><td>1</td></tr> </tbody> </table> <b>Boolean Expression</b> $Y = A \cdot B$	Input	Output	A    B	Y	0    0	0	0    1	0	1    0	0	1    1	1
S <sub>1</sub>	S <sub>2</sub>	Output																											
Open (Low)	Open (Low)	Lamp OFF (Low)																											
Open (Low)	Close (High)	Lamp OFF (Low)																											
Close (High)	Open (Low)	Lamp OFF (Low)																											
Close (High)	Close (High)	Lamp ON (High)																											
Input	Output																												
A    B	Y																												
0    0	0																												
0    1	0																												
1    0	0																												
1    1	1																												

**Application :** Used to implement logical AND operation.

**NOT gate (Inverter) :** The output is a complement of input.

**OR gate :** The output is high when any of the inputs is high.

Logic Diagram (Symbol)	Switch Equivalent	Truth Table																											
 <b>Pin Diagram</b>	 <b>Input Output</b> <table border="1"> <tr><td>S<sub>1</sub></td><td>S<sub>2</sub></td><td>Output</td></tr> <tr><td>Open (Low)</td><td>Open (Low)</td><td>Lamp OFF (Low)</td></tr> <tr><td>Open (Low)</td><td>Close (High)</td><td>Lamp ON (High)</td></tr> <tr><td>Close (High)</td><td>Open (Low)</td><td>Lamp ON (High)</td></tr> <tr><td>Close (High)</td><td>Close (High)</td><td>Lamp ON (High)</td></tr> </table>	S <sub>1</sub>	S <sub>2</sub>	Output	Open (Low)	Open (Low)	Lamp OFF (Low)	Open (Low)	Close (High)	Lamp ON (High)	Close (High)	Open (Low)	Lamp ON (High)	Close (High)	Close (High)	Lamp ON (High)	<b>Truth Table</b> <table border="1"> <thead> <tr><th>Input</th><th>Output</th></tr> </thead> <tbody> <tr><td>A    B</td><td>Y</td></tr> <tr><td>0    0</td><td>0</td></tr> <tr><td>0    1</td><td>1</td></tr> <tr><td>1    0</td><td>1</td></tr> <tr><td>1    1</td><td>1</td></tr> </tbody> </table>	Input	Output	A    B	Y	0    0	0	0    1	1	1    0	1	1    1	1
S <sub>1</sub>	S <sub>2</sub>	Output																											
Open (Low)	Open (Low)	Lamp OFF (Low)																											
Open (Low)	Close (High)	Lamp ON (High)																											
Close (High)	Open (Low)	Lamp ON (High)																											
Close (High)	Close (High)	Lamp ON (High)																											
Input	Output																												
A    B	Y																												
0    0	0																												
0    1	1																												
1    0	1																												
1    1	1																												

**Application :** Used to implement logical OR operation.

**Buffer :** The output is same as input.

Symbol	Boolean Expression	Truth Table
	$Y = A$	<b>Pin Diagram</b>

**Application :** It is used to increase output driving capacity.

Symbol	Boolean Expression $Y = \overline{A} \cdot \overline{B}$	Truth Table		
		Input	Output	
		A	B	Y
Application : It can be used to implement any digital circuit.		0	0	1
		0	1	1
		1	0	1
		1	1	0

NAND gate : The output is high only when one of the inputs is low.

Symbol	Boolean Expression $Y = A + B$	Truth Table		
		Input	Output	
		A	B	Y
Application : It can be used to implement any digital circuit.		0	0	1
		0	1	0
		1	0	0
		1	1	0

NOR Gate : The output is high when all the inputs are low.

Symbol	Boolean Expression $Y = A \oplus B$	Truth Table		
		Input	Output	
		A	B	Y
Application : It is used to implement magnitude comparator, gray code converter, adder/subtractor circuits, parity generator, modulo-2 adder, etc.		0	0	0
		0	1	1
		1	0	1
		1	1	0

Exclusive OR (EX-OR) gate : The output is high only when odd number of inputs are high.

Symbol	Boolean Expression $Y = A \oplus B$	Truth Table		
		Input	Output	
		A	B	Y
Application : It is used to implement even parity generator, comparator, even parity checker, etc.		0	0	1
		0	1	0
		1	0	0
		1	1	1

Exclusive NOR (EX-NOR) gate : The output is high only when even number of ones at the input or all inputs are high.				
Symbol	Boolean Expression $Y = A \oplus B$	Truth Table		
		Input	Output	
		A	B	Y
Applications : It is used to implement even parity generator, comparator, even parity checker, etc.		0	0	1
		0	1	0
		1	0	0
		1	1	1

Note EX-OR and EX-NOR gates are also known as mutually exclusive gates.

#### Wave-forms / Timing diagrams of all the gates

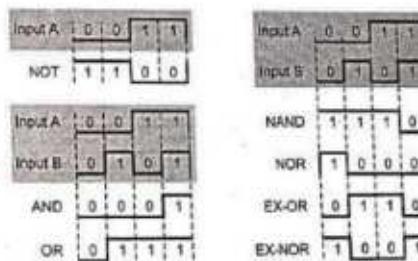


Fig. 2.1.1

Note : EX-OR and EX-NOR gates are also known as mutually exclusive gates.

#### 2.1.1 Positive and Negative Logic

- \* The binary logic used in the digital systems assumes only two values, either HIGH or LOW. The HIGH and LOW voltage levels are used to denote these two values.

- The two levels or states, of a signal variable, can be considered to represent the two numerals, viz. 1 and 0 of the binary number system or the two logic states, viz. TRUE and FALSE in logic operations.
  - Two voltage levels used to represent two logic states are also called logic levels.
  - In binary logic, two voltage levels represent the two binary digits, 1 and 0.
  - If the higher of the two voltages represents a 1 and the lower voltage represents 0, the system is called a positive logic system.
  - If the lower voltage represents a 1 and the higher voltage represents a 0, we have a negative logic system.
  - As an example, assume that we have positive 5 V and 0 V as logic-level voltages.
  - Then positive and negative logic can be defined as:
- Positive Logic :** HIGH = 1 (i.e. +5 V)  
LOW = 0 (i.e. 0 V)
- Negative Logic :** HIGH = 0 (i.e. +5 V)  
LOW = 1 (i.e. 0 V)
- Both positive and negative logic are used in digital systems, but positive logic is the more common.
  - In practice, the voltages at different nodes in the digital circuit may differ slightly due to internal resistances, parasitic effects and loading effects. Therefore, to define a logic level, a range of voltage is assigned instead of a particular voltage level. This is illustrated in Fig. 2.1.2.

**Review Question**

2. Draw the logic symbol, expression and truth table for following logic gates :  
1. AND 2. OR 3. NOT 4. NAND 5. NOR 6. EX-OR 7. EX-NOR

**2.2 Boolean Algebra**

GTU : Dec-12, 13, May-13, 14, Summer-15, 16, 17, 18  
Winter-14, 16, 18

- Boolean algebra is a mathematical system that defines a series of logical operations (AND, OR, NOT) performed on sets of variables (a, b, c, ...). When stated in this form, the expression is called a Boolean equation or switching equation.

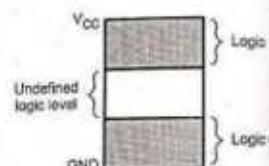


Fig. 2.1.2 Voltage range for positive logic system

**2.2.1 Boolean Algebra Terminology**

**Variable :** The symbol which represent an arbitrary elements of an Boolean algebra is known as variable. Any single variable or a function of several variables can have either a 1 or 0 value. For example, in expression  $Y = A + BC$ , variables A, B and C can have either a 1 or 0 value, and function Y also can have either a 1 or 0 value; however its value depends on the value of Boolean expression.

**Constant :** In expression  $Y = A + 1$ , the first term A is a variable and have value either a 1 or 0. The second term has a fixed value 1. So 1 is a constant here. The constant term may be 0 or 1.

**Complement :** A complement of a variable is represented by a "bar" over the letter. For example, the complement of a variable A will be denoted by  $\bar{A}$ . So if  $A = 1$ ,  $\bar{A} = 0$  and if  $A = 0$ ,  $\bar{A} = 1$ . Sometimes a prime symbol ('') is used to denote the complement. For example, the complement of A can be written as  $A'$ .

**Literal :** Each occurrence of a variable in Boolean function either in a complemented or uncomplemented form is called a literal.

**Boolean function :** Boolean expressions are constructed by connecting the Boolean constants and variables with the Boolean operations. These Boolean expressions are also known as Boolean formulae.

- We use Boolean expressions to describe Boolean functions.
- For example, if the Boolean expression  $(A + \bar{B})C$  is used to describe the function f, then Boolean function is written as  
 $f(A, B, C) = (A + \bar{B})C$  or  $f = (A + \bar{B})C$

**2.2.2 Axiomatic Definitions of Boolean Algebra**

- The postulates of a mathematical system form the basic assumption from which it is possible to deduce the theorems, laws and properties of the system. Boolean algebra is formulated by a defined set of elements, together with two binary operators, + and ., provided that the given postulates are satisfied.

**Closure Property**

- Closure (a) :** Closure with respect to the operator + : When two binary elements are operated by operator + the result is a unique binary element.
- Closure (b) :** Closure with respect to the operator .(dot) : When two binary elements are operated by operator .(dot), the result is a unique binary element.

**Identity Property**

- An identity element with respect to  $\cdot$ , designated by  $1 : A \cdot 1 = 1 \cdot A = A$
- 
- a) AND identity      b) OR identity

Fig. 2.2.1 Identity elements for the AND and OR functions

**Commutative Property**

- Commutative with respect to  $+$  :  $A + B = B + A$
- Commutative with respect to  $\cdot$  :  $A \cdot B = B \cdot A$

**Distributive Property**

- Distributive property of  $\cdot$  over  $+$  :  $A \cdot (B + C) = (A \cdot B) + (A \cdot C)$
- Distributive property of  $+$  over  $\cdot$  :  $A + (B \cdot C) = (A + B) \cdot (A + C)$

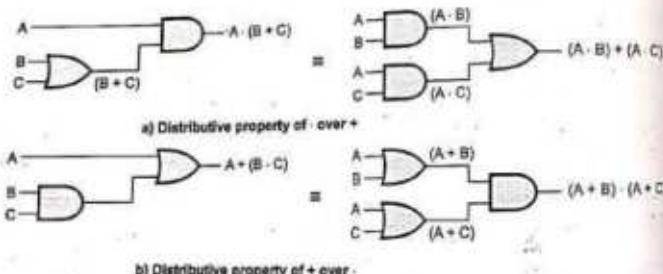


Fig. 2.2.2 Distributive property

**Associative Property**

- Associative property of  $+$  :  $A + (B + C) = (A + B) + C$
- Associative property of  $\cdot$  :  $(A \cdot B) \cdot C = A \cdot (B \cdot C)$

**Complement Property**

- For every binary element  
 $A = 1, \bar{A} = 0$
- Complementary property of  $\cdot$  :  $A \cdot \bar{A} = 0$
- Complementary property of  $+$  :  $A + \bar{A} = 1$

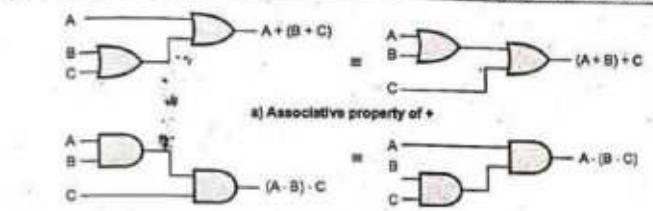


Fig. 2.2.3 Associative property

**Idempotency Property**

- Idempotency refers to the property of sameness :  $A + A = A$  or  $A \cdot A = A$
- Any variable ANDed or ORed with itself results in the original variable.

**Absorption Property**

- We know that, in OR function if any one input is logic 1, output is logic 1. Therefore, we have an important relationship :  $A + 1 = 1$
  - Using this relationships we can derive two absorption properties as follows :
- $A + AB = A (1 + B) = A$  ... Distributive property  
 $\because 1 + B = 1$
  - $A (A + B) = AA + AB = A + AB$  ... Distributive property  
 $\therefore A (A + B) = A$  ... Idempotency property  
... Absorption property

**Involution Property**

- Double inversion of any variable results in the original variable  
 $\bar{\bar{A}} = A$

**2.2.3 Basic Theorems and Properties of Boolean Algebra**

- From the above discussion we can summarize the postulates and theorems of Boolean algebra as shown in Table 2.2.1.

Postulates	(a)	(b)
Postulate 1	Result of each operation is either 0 or 1 i.e. 0	$A \cdot 1 = A$
Postulate 2 (Identity)	$A + 0 = A$	$AB = BA$
Postulate 3 (Commutative)	$A + B = B + A$	

Digital Fundamentals		Logic Gates and Boolean Algebra	
	2-10		2-11
Postulate 4 (Distributive)	$A(B+C) = AB + AC$	$A + BC = (A+B)(A+C)$	
Postulate 5 (Complement)	$A + \bar{A} = 1$	$A \cdot \bar{A} = 0$	
Theorems	(a)	(b)	
Theorem 1 (Idempotency)	$A + A = A$	$A \cdot A = A$	
Theorem 2	$A + 1 = 1$	$A \cdot 0 = 0$	
Theorem 3 (Involution)		$\bar{\bar{A}} = A$	
Theorem 4 (Absorption)	$A + AB = A$	$A(A+B) = A$	
Theorem 5	$A + \bar{A}B = A + B$	$A \cdot (\bar{A} + B) = AB$	
Theorem 6 (Associative)	$A + (B + C) = (A + B) + C$	$A(BC) = (AB)C$	

Table 2.2.1 Postulates and basic theorems of Boolean algebra

**Example 2.2.1** Prove that  $A + \bar{A}B = A + B$ 

GTU : Winter-18, Marks 2

Solution : 
$$\begin{aligned} A + \bar{A}B &= A + AB + \bar{A}B && \text{by Theorem : 4(a)} \\ &= A + B \cdot (A + \bar{A}) && \text{by Postulate : 4(a)} \\ &= A + B \cdot 1 && \text{: 5(a)} \\ &= A + B && \text{: 2(b)} \end{aligned}$$

**Example 2.2.2** Prove that  $A \cdot (\bar{A} + B) = AB$ 

GTU : Winter-18, Marks 2

Solution : 
$$\begin{aligned} A \cdot (\bar{A} + B) &= (A + AB) \cdot (\bar{A} + B) && \text{by Theorem : 4(a)} \\ &= \bar{A}A + AB + ABB && \text{by Postulate : 4(a)} \\ &= AB + ABB && \text{: 5(b)} \\ &= AB + AB && \text{: 2(b)} \\ &= AB && \text{by Theorem : 1(a)} \end{aligned}$$

**2.2.4 DeMorgan's Theorems**

- DeMorgan suggested two theorems that form an important part of Boolean algebra. In the equation form, they are : 1.  $\bar{AB} = \bar{A} + \bar{B}$
- 2.  $\bar{A+B} = \bar{A} \cdot \bar{B}$

1.  $\bar{AB} = \bar{A} + \bar{B}$  : The complement of a product is equal to the sum of the complements. This is illustrated by truth Table 2.2.2.

A	B	$\bar{AB}$	$\bar{A} + \bar{B}$
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0

Table 2.2.2 Truth table

**Digital Fundamentals**

2-11

**Logic Gates and Boolean Algebra**

2.  $\bar{A+B} = \bar{A} \cdot \bar{B}$  : The complement of a sum is equal to the product of the complements. The truth Table 2.1.3 illustrates this law.

**2.2.5 Principle of Duality**

- The principle of duality theorem says that, starting with a Boolean relation, you can derive another Boolean relation by

1. Changing each OR sign to an AND sign
2. Changing each AND sign to an OR sign and
3. Complementing any 0 or 1 appearing in the expression.

For example : Dual of relation  $A + \bar{A} = 1$  is  $A \cdot \bar{A} = 0$

**Illustrative Examples****Example 2.2.3** Apply DeMorgan's theorem to simplify  $A + \bar{B}C$ 

Solution : 
$$\begin{aligned} A + \bar{B}\bar{C} &= \bar{A} \cdot \bar{\bar{B}\bar{C}} \\ &= \bar{A} \cdot (\bar{B} + \bar{C}) = \bar{A}\bar{B} + \bar{A}\bar{C} \end{aligned}$$

**Example 2.2.4** Simplify  $\bar{ABC} + \bar{BCD} + \bar{BCD} + \bar{BCD}$ 

Solution : 
$$\begin{aligned} &= \bar{BCD}(\bar{A} + 1) + \bar{BCD} + \bar{BCD} && \text{Distributive} \\ &= \bar{BCD} + \bar{BCD} + \bar{BCD} && \text{Theorem 2(a) : } [\bar{A} + 1 = 1] \\ &= \bar{BD}(\bar{C} + \bar{C}) + \bar{BCD} && \text{Distributive} \\ &= \bar{BD} + \bar{BCD} && \text{Postulate 5(a) : } [\bar{A} + \bar{A} = 1] \\ &= B(\bar{D} + \bar{C}D) && \text{Distributive} \\ &= B(\bar{D} + \bar{C}) && \text{Theorem 5(a) : } [A + \bar{AB} = A + B] \end{aligned}$$

**Example 2.2.5** Simplify  $AB + \bar{AC} + \bar{AB}BC + \bar{AB}CC$ 

$$\begin{aligned} &= AB + \bar{AC} + A\bar{B}BC + \bar{AB}CC && \text{Distributive} \\ &= AB + \bar{AC} + \bar{AB}CC && \text{Postulate 5(b) : } [A \cdot \bar{A} = 0] \\ &= AB + \bar{AC} + \bar{AB}C && \text{Theorem 1(b) : } [A \cdot A = A] \\ &= AB + \bar{A} + \bar{C} + A\bar{B}C && \text{DeMorgan's Theorem 1 : } [\bar{AB} = \bar{A} + \bar{B}] \\ &= \bar{A} + B + \bar{C} + \bar{AB}C && \text{Theorem 5(a) : } [A + \bar{AB} = A + B] \\ &= \bar{A} + \bar{AB}C + B + \bar{C} && \text{Commutative} \\ &= \bar{A} + \bar{B}C + B + \bar{C} && \text{Theorem 5(a) : } [A + \bar{AB} = A + B] \\ & \text{Here } B = \bar{B}C && \end{aligned}$$

A	B	$\bar{A} + \bar{B}$	$A \cdot \bar{B}$
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	0

Table 2.2.3 Truth table

$$\begin{aligned} &= \bar{A} + B + \bar{C} + \bar{B}C \\ &= \bar{A} + B + \bar{C} + \bar{B} \\ &= \bar{A} + \bar{C} + 1 \\ &= 1 \end{aligned}$$

Commutative  
Theorem 5(a) :  $[A + \bar{A}B = A + B]$   
Postulate 5(a) :  $[A + \bar{A} = 1]$   
Theorem 2(a) :  $[A + 1 = 1]$

**Example 2.2.5** Simplify  $\bar{A}B + \bar{A} + AB$ 

$$\begin{aligned} \text{Solution: } &= \bar{A} + \bar{B} + A + AB \quad \text{DeMorgan's Theorem 1: } [\bar{AB} = \bar{A} + \bar{B}] \\ &= \bar{A} + B + \bar{A}B \quad \text{Theorem 5(a): } [\bar{A} + \bar{AB} = \bar{A} + B] \\ &= \bar{A} + 1 \quad \text{Theorem 1(a): } [A + A = A] \text{ and} \\ &\quad \text{Postulate 5(a): } [\bar{A} + \bar{A} = 1] \\ &= 1 \quad \text{Theorem 2(a): } [A + 1 = 1] \\ &= 0 \end{aligned}$$

**Example 2.2.7** Realize expression using minimum NAND gates only.

Solution :

$$\begin{aligned} y &= A\bar{B} + A\bar{C} + C + AD + A\bar{B}\bar{C} + ABC \\ &= A\bar{B}(1+C) + A\bar{C} + C(1+AD) + AD \\ &= A\bar{B} + A\bar{C} + C + AD \quad \because 1+C = 1 \text{ and } 1+AD = 1 \\ &= A\bar{B} + A + C + AD \quad \because A + \bar{A}B = A + B \\ &= A(\bar{B} + 1 + D) + C \quad \because 1 + A = 1 \\ &= A + C \\ &= \bar{A} + \bar{C} \quad \because A = \bar{A} \\ &= \bar{A} \cdot \bar{C} \end{aligned}$$

Realization using NAND gates

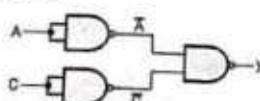


Fig. 2.2.4

**Example 2.2.8** Prove the following using Boolean theorems.

$$\begin{aligned} i) & [(C + \bar{C}D)(C + \bar{C}\bar{D})] [AB + \bar{A}\bar{B} + A \oplus B] = C \\ ii) & \bar{ABC} + \bar{ABC} + \bar{ABC} + \bar{ABC} = AB + AC + BC \end{aligned}$$

Solution :

$$\begin{aligned} i) & [(C + \bar{C}D)(C + \bar{C}\bar{D})] [AB + \bar{A}\bar{B} + A \oplus B] \\ &= [(C + D)(C + \bar{D})] [AB + \bar{A}\bar{B} + A\bar{B} + \bar{A}B] \quad \dots A + \bar{A}B = A + B \\ &= [CC + C\bar{D} + CD + D\bar{D}] [A(B + \bar{B}) + \bar{A}(\bar{B} + B)] \\ &= [C + C\bar{D} + CD] [A + \bar{A}] \quad \because A\bar{A} = 0 \text{ and } B + \bar{B} = 1 \\ &= [C(1 + \bar{D} + D)] \quad \because A + \bar{A} = 1 \\ &= C \quad \dots \text{Proved} \\ &\quad \because 1 + A = 1 \\ ii) & \bar{ABC} + \bar{ABC} + \bar{ABC} + \bar{ABC} \\ &= BC(\bar{A} + A) + AC(\bar{B} + B) + AB(\bar{C} + C) \\ &= BC + AC + AB \quad \because A + \bar{A} = 1 \\ &= AB + AC + BC \quad \dots \text{Proved} \end{aligned}$$

**Example 2.2.9** Simplify : 1)  $AB + \bar{ABC} + \bar{ABCD} + \bar{ABCDE}$   
2)  $(P + Q + R)(\bar{P} + \bar{Q} + \bar{R})P$ 

GTU : May-14, Marks 8

$$\begin{aligned} \text{Solution : 1)} & \bar{AB} + \bar{ABC} + \bar{ABCD} + \bar{ABCDE} \\ &= \bar{AB}(1 + \bar{C} + \bar{CD} + \bar{CDE}) \\ &= \bar{AB} \quad \because 1 + A = 1 \\ 2) & (P + Q + R)(\bar{P} + \bar{Q} + \bar{R})P \\ &= (P\bar{P} + P\bar{Q} + P\bar{R} + \bar{P}Q + \bar{P}R + \bar{Q}P + Q\bar{R} + \bar{P}R + \bar{Q}R + R\bar{P} + R\bar{Q})P \\ &= (P\bar{Q} + P\bar{R} + \bar{P}Q + Q\bar{R} + PR + \bar{Q}R)P \quad \dots \because A \cdot \bar{A} = 0 \\ &= P\bar{P}Q + P\bar{P}R + P\bar{Q}R + P\bar{Q}Q + P\bar{Q}R + P\bar{R}R + P\bar{R}Q \quad \dots \because A \cdot A = A \text{ and } A \cdot \bar{A} = 0 \\ &= P\bar{Q}(1 + R) + P\bar{R}(1 + Q) \\ &= P\bar{Q} + P\bar{R} \quad \dots \because 1 + A = 1 \end{aligned}$$

**Example 2.2.10** Simplify the following Boolean functions to a minimum number of literals.  
a)  $xyz + \bar{x}y + xy\bar{z}$  and b)  $(A + B)(\bar{A} + B)$ 

GTU : May-15, Marks 7

Solution : a)  $xyz + \bar{x}y + xy\bar{z}$ 

$$\begin{aligned} &= xy(z + \bar{z}) + \bar{x}y \\ &= xy + \bar{x}y \quad \dots \because A + \bar{A} = 1 \\ &= y(x + \bar{x}) \\ &= y \quad \dots \because A + \bar{A} = 1 \end{aligned}$$

$$\begin{aligned}
 b) & (A + B)(\overline{A} + \overline{B}) \\
 &= \overline{A} \cdot \overline{B} + A \cdot \overline{B} \\
 &= \overline{B}(A + \overline{A}) \\
 &= \overline{B}
 \end{aligned}
 \quad \dots \text{De Morgan's theorem}$$

**Example 2.2.11** Prove that : 1)  $((AB + ABC) + A(B + A\bar{B})) = 0$   
 2)  $ABC + \overline{ABC} + ABC = AC + BC$

GTU : May-14, Marks 7

$$\begin{aligned}
 \text{Solution : } 1) & ((A\bar{B} + ABC) + A(\bar{B} + A\bar{B})) \\
 &= (\overline{A}\bar{B} + ABC)(A(\bar{B} + A\bar{B})) \\
 &= (A\bar{B} + ABC)(A\bar{B} + A\bar{B}) \\
 &= (A\bar{B} + ABC)(\overline{A} + \overline{A}\bar{B}) \\
 &= A(\bar{B} + BC)(\overline{A} + \overline{B}) \\
 &= A(\bar{B} + C)\cdot \overline{A} \\
 &= 0
 \end{aligned}
 \quad \dots \because A \cdot \overline{A} = 0, \text{ hence proved}$$

$$\begin{aligned}
 2) \quad \text{L.H.S.} &= A\bar{B}C + \overline{ABC} + ABC \\
 &= A\bar{B}C + BC(\overline{A} + A) \\
 &= A\bar{B}C + BC \\
 &= C(A\bar{B} + B) \\
 &= C(A + B) \\
 &= AC + BC
 \end{aligned}
 \quad \dots \because A + \overline{A}B = A + B$$

**Example 2.2.12** Find the complement of the following Boolean function and reduce to a minimum number of literals.  
 $BD + ABC + ACD + ABC$

GTU : Dec.-12, Marks 7

$$\begin{aligned}
 \text{Solution : } Y &= \overline{BD} + \overline{ABC} + ACD + \overline{ABC} \\
 Y &= \overline{BD} + \overline{AB} + ACD \\
 \overline{Y} &= (B + \overline{D})(A + \overline{B})(\overline{A} + \overline{C} + \overline{D}) \\
 &= (AB + A\overline{D} + B\overline{D})(\overline{A} + \overline{C} + \overline{D}) \\
 &= ABC\bar{C} + AB\bar{D} + A\overline{C}\bar{D} + A\bar{D} + \overline{A}\overline{B}\overline{D} + \overline{B}\overline{C}\overline{D} + \overline{B}\overline{D} \\
 &= AB\bar{C} + A\bar{D}(B + \overline{C} + 1) + \overline{B}\overline{D}(\overline{A} + \overline{C} + 1) \\
 &= AB\bar{C} + A\bar{D} + \overline{B}\overline{D}
 \end{aligned}$$

**Example 2.2.13** Show that the dual of the EX-OR is equal to its complement.

GTU : May-13, Marks 7

Solution :  $A \oplus B = A\bar{B} + \overline{A}\bar{B}$

... equation of EX-OR

According to duality property,

$$\overline{A \oplus B} = (\overline{A} + B) \cdot (\overline{A} + \overline{B}) \quad \dots (i)$$

According to complement property,

$$\begin{aligned}
 \overline{A \oplus B} &= \overline{A\bar{B} + \overline{A}\bar{B}} \\
 &= (\overline{A} + B) \cdot (\overline{A} + \overline{B})
 \end{aligned} \quad \dots (ii)$$

From equation (i) and (ii) it is proved that duality of EX-OR is equal to its complement.

**Example 2.2.14** Show that  $A \oplus B = AB + A'B' = (A \oplus B)' = (\overline{AB} + \overline{A'B})$ . Also construct the corresponding logic diagrams.

GTU : Summer-16, Marks 7

Solution :  $A \oplus B = \overline{A\bar{B} + \overline{A}\bar{B}} = (\overline{A}\bar{B}) \cdot (\overline{A}\bar{B})$

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

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

$$= A \oplus B$$

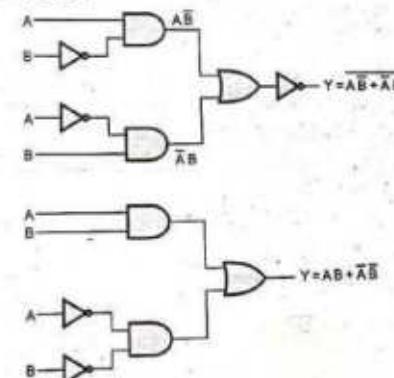


Fig. 2.2.5

**Example 2.2.15** Design a NOT gate using a two input EX-OR gate.

GTU : Winter-16, Mark 1

Solution : EX-OR gate :  $A \bar{B} + \bar{A} B$  considering one input say  $B = 1$  we have,

$$Y = A \cdot (\bar{1}) + \bar{A} \cdot (1) = A \cdot 0 + \bar{A} = \bar{A}$$

Thus by connecting any one input to logic 1 two input EX-OR gate can be used as a NOT gate.

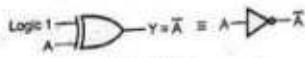


Fig. 2.2.6

**Example 2.2.16** Show that  $(A+C)(A+D)(B+C)(B+D) = AB+CD$ .

GTU : Winter-16, Marks 3

$$\begin{aligned} \text{Solution : L.H.S} &= (A+C)(A+D)(B+C)(B+D) \\ &= [AA + AD + AC + CD] [BB + BD + BC + CD] \\ &= [A + AD + AC + CD] \cdot [B + BD + BC + CD] \\ &= [A(1 + D + C) + CD] \cdot [B(1 + D + C) + CD] \\ &= (A + CD) \cdot (B + CD) \\ &= AB + ACD + BCD + CD \\ &= AB + ACD + BCD + CD \\ &= AB + CD(A + B + 1) \\ &= AB + CD \end{aligned}$$

**Example 2.2.17** Simplify using boolean laws and draw the logic diagram for the simplified expression.  $F = (ABC)' + (AB'C) + A'BC' + A(BC)' + AB'C$

GTU : Winter-14, Summer-17, Marks 4

**Solution :**

$$\begin{aligned} F &= \overline{ABC} + \overline{ABC} + \overline{ABC} + \overline{ABC} + \overline{ABC} \\ &= \overline{A} + \overline{B} + \overline{C} + \overline{AC} + \overline{BC} + \overline{ABC} + \overline{AB} + \overline{A} \overline{C} + \overline{ABC} \\ &= \overline{A}(1 + C + \overline{B} \overline{C}) + \overline{B}(1 + C + A + AC) + \overline{C}(1 + A) \\ &= \overline{A} + \overline{B} + \overline{C} = ABC \end{aligned}$$

**Logic Diagram :**

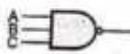


Fig. 2.2.7

**Example 2.2.18** State and prove DeMorgan's theorem for three variables.

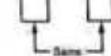
Solution : 1)  $\overline{ABC} = \overline{A} + \overline{B} + \overline{C}$  DeMorgan's Theorem for 3 - variables

$$2) \overline{A + B + C} = \overline{A} \overline{B} \overline{C}$$

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



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



**Example 2.2.19** Using laws of Boolean algebra prove that  $AB + BC + A'C = AB + A'C$ .

GTU : Summer-18, Marks 7

**Solution :**

$$\begin{aligned} AB + BC + \overline{A}C &= AB + BC(A + \overline{A}) + \overline{A}C(B + \overline{B}) \\ &= AB + ABC + \overline{ABC} + \overline{ABC} + \overline{A}C \\ &= AB(1+C) + \overline{A}C(B+\overline{B}) \\ &= AB + \overline{A}C \end{aligned}$$

...proved

**Example 2.2.20** Simplify the following Boolean functions to a minimum numbers of literals.

GTU : Winter-18, Marks 4

**Solution :**

$$\begin{aligned} 1. \quad \overline{xy}z + \overline{x}yz + xy &= \overline{x}z(\overline{y} + y) + x\overline{y} = \overline{x}z + x\overline{y} \\ 2. \quad xy + \overline{x}z + yz &= xy + \overline{x}z + yz(x + \overline{x}) \\ &= xy + \overline{x}z + xyz + \overline{x}yz = xy(1+z) + \overline{x}z(1+y) \\ &= xy + \overline{x}z \end{aligned}$$

## Review Questions

1. Define the following terms : Boolean variable, complement, literal.  
 2. State the fundamental postulates of Boolean algebra.  
 3. State various laws of Boolean algebra.  
 4. State the associative law of Boolean algebra.  
 5. State and prove DeMorgan's theorem with help of truth tables.  
 6. Explain the principle of duality with the help of example.  
 7. State and prove the postulates of Boolean algebra.  
 8. State and prove idempotent laws of Boolean algebra.  
 9. State De Morgan's theorems and prove with the help of truth table. **GTU : Dec.-13, Marks 1**  
**GTU : Dec.-13, Marks 2**  
**Summer-15, 18, Marks 4**

## Oral Questions and Answers

**Q.1** What is a logic gate?

**Ans. :** The logic gate is an electronic circuit that has one or more input binary variables but only one output. It is called logic gate because of its ability to operate on a number of binary inputs to perform a logical function, i.e. its output is a logical function of inputs.

**Q.2** What are the basic digital logic gates?**Ans. :** The three basic logic gates are

- AND gate
- OR gate
- NOT gate

**Q.3** Draw the logic symbol and construct the truth table of the following gates :

- a) Three input OR gate b) Three input EX-NOR gate

**Ans. :** a) Three input OR gate :

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



Fig. 2.1 Three input OR gate

Table 2.1 Truth table for three input OR gate

## b) Three input EX-NOR gate :

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



Fig. 2.2 Three input EX-NOR gate

Table 2.2 Truth table for three input EX-NOR

**Q.4** Give the Boolean expression used for following gates

- a) AND b) NOR c) EX-OR d) OR e) NOT

**Ans. :**

- a) AND :  $Y = AB$   
 b) NOR :  $Y = \overline{A+B}$   
 c) EX-OR :  $Y = \overline{AB} + \overline{AB} = A \oplus B$   
 d) OR :  $Y = A + B$   
 e) NOT :  $Y = \overline{A}$

**Q.5** Which gate is equal to AND-invert gate?**Ans. :** NAND gate.**Q.6** Which gate is equal to OR-invert gate?**Ans. :** NOR gate.**Q.7** Show that a bubbled AND gate works like a NOR gate.**Ans. :**

Fig. 2.3

A	B	$\bar{A}$	$\bar{B}$	Y
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	0

Here, output Y presents the output of NOR gate.

**Q.8** What do you mean by literal?

**Ans.:** Definition : In Boolean Function, the total number of variables in complemented or uncomplemented form are called literals.

For example : The function  $F(A, B, C, D) = A + \bar{B}C + A\bar{C}\bar{D}$  contains 4 literals.

**Q.9** State two absorption properties of Boolean algebra.

**Ans.:** 1.  $A + AB = A$ , and  $A(A + B) = A$

**Q.10** State the associative law of Boolean algebra.

**Ans.:** Associative laws are as follows :

1. Law 1 (The associative law of addition) :

In the ORing of several variables, the result is the same regardless of the grouping of the variables. For three variables, A ORed with B OR C is the same as A OR B ORed with C.

$$\text{i.e. } A + (B + C) = (A + B) + C$$

2. Law 2 (The associative law of multiplication) :

It makes no difference in what order the variables are grouped when ANDing several variables. For three variables, A AND B ANDed with C is the same as A ANDed with B and C.

$$\text{i.e. } (AB)C = A(BC)$$

**Q.11** Explain the De Morgan's theorem in Boolean algebra.

**Ans.:** Refer section 2.2.4.

**Q.12** Explain the principle of duality with the help of example.

**Ans.:** The duality theorem says that, starting with a Boolean relation, you can derive another Boolean relation by -

1. Changing each OR sign to an AND sign

2. Changing each AND sign to an OR sign and

3. Complementing any 0 to 1 appearing in the expression.

For example :  $A + 0 = A$ . Using duality theorem, we can say that,  $A \cdot 1 = A$ .

# 3

## Digital Logic Families

### Contents

3.1	Introduction	Summer-18, ... Marks 2
3.2	Characteristics of Digital ICs	Summer-15, 18, ... Winter-15, 18, ... Marks 4
3.3	Transistor-Transistor Logic (TTL)	Winter-18, ... Marks 3
3.4	CMOS Logic	
3.5	Interfacing of CMOS to TTL and TTL to CMOS	
3.6	Comparison between TTL and CMOS	Winter-15, Summer-18, ... Marks 7
Oral Questions and Answers		

### 3.1 Introduction

A digital logic family is a group of compatible devices with the same logic levels and supply voltages. According to components used in the logic family, digital logic families are classified as shown in the Fig. 3.1.1.

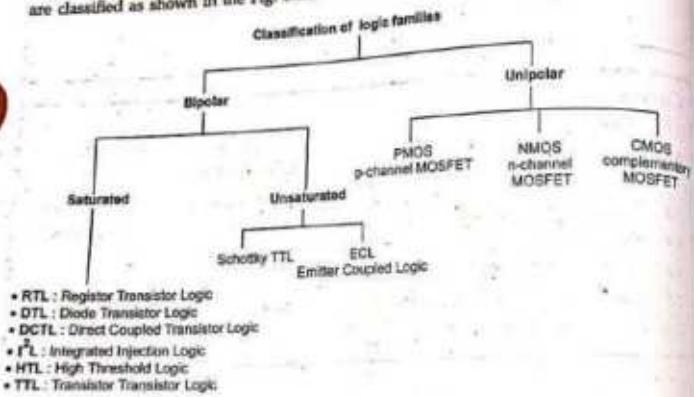


Fig. 3.1.1 Classification of logic families

#### Advantages of digital integrated circuits

1. ICs pack a lot more circuitry in a small package, so that the overall size of almost any digital system is reduced.
2. The cost of ICs is very low, which makes them economical to operate.
3. They have high reliability against failure, so the digital system needs less repair.
4. Their reduced power consumption makes the digital system more economical to operate.
5. The operating speed is higher, which makes them suitable for high-speed operations.
6. The use of ICs reduces the number of external wiring connections because many of them are internal to the package.
- With the widespread use of ICs, it becomes necessary to know and understand the electrical characteristics of the IC logic families such as TTL and CMOS.

### Review Questions

1. What is logic family ? List out various logic families.
2. What are the advantages of digital integrated circuits ?

### 3.2 Characteristics of Digital ICs

#### Propagation Delay

The propagation delay of a gate is basically the time interval between the application of an input pulse and the occurrence of the resulting output pulse.

- The propagation delay is a very important characteristic of logic circuits because it limits the speed at which they can operate.
- The shorter the propagation delay, the higher the speed of the circuit and vice-versa.
- The propagation delay is determined using two basic time intervals :
  1.  $t_{PLH}$  : It is the delay time measured when output is changing from logic 0 to logic 1 state (LOW to HIGH).
  2.  $t_{PHL}$  : It is the delay time measured when output is changing from logic 1 to logic 0 state (HIGH to LOW).
- When  $t_{PHL}$  and  $t_{PLH}$  are not equal, the larger value is considered as a propagation delay time for that logic gate, i.e.

$$t_p = \max(t_{PLH}, t_{PHL})$$

#### Power Dissipation

The amount of power that an IC dissipates is determined by the average supply current,  $I_{CC}$ , that it draws from the  $V_{CC}$  supply. It is the product of  $I_{CC}$  and  $V_{CC}$ .

- For ICs, the value of  $I_{CC}$  for a LOW gate output ( $I_{CCL}$ ) is different from a HIGH output ( $I_{CHL}$ ).
- Average  $I_{CC}$  is determined based on the 50 % duty cycle operation of the gate (LOW half of the time and HIGH half of the time).
 
$$I_{CC(\text{avg})} = \frac{I_{CCH} + I_{CCL}}{2}$$
- Average power dissipation as,
 
$$P_{D(\text{avg})} = I_{CC(\text{avg})} \times V_{CC}$$

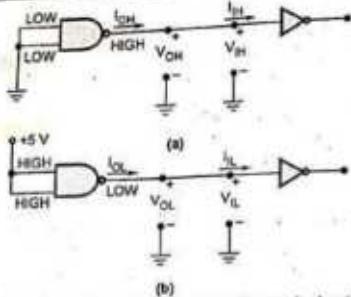


Fig. 3.2.1 Currents and voltages in the two logic states

## Current and Voltage Parameters

$V_{IH(\min)}$  - High-Level Input Voltage : It is the minimum voltage level required for a logical 1 at an input. Any voltage below this level will not be accepted as a HIGH by the logic circuit.

$V_{IL(\max)}$  - Low-Level Input Voltage : It is the maximum voltage level required for a logic 0 at an input. Any voltage above this level will not be accepted as a LOW by the logic circuit.

$V_{OH(\min)}$  - High-Level Output Voltage : It is the minimum voltage level at a logic circuit output in the logical 1 state under defined load conditions.

$V_{OL(\max)}$  - Low-Level Output Voltage : It is the maximum voltage level at a logic circuit output in the logical 0 state under defined load conditions.

$I_{IH}$  - High-Level Input Current : It is the current that flows into an input when a specified high-level voltage is applied to that input.

$I_{IL}$  - Low-Level Input Current : It is the current that flows into an input when a specified low-level voltage is applied to that input.

$I_{OH}$  - High-Level Output Current : It is the current that flows from an output in the logical 1 state under specified load conditions.

$I_{OL}$  - Low-Level Output Current : It is the current that flows from an output in the logical 0 state under specified load conditions.

**Note** The current directions shown in the Fig. 3.2.1 may be opposite to those shown depending on the logic family.

## Noise Margin and Logic Voltages Levels

- In digital circuits, the binary 0 and 1 are represented by a pair of voltage levels.
- Each logic family has a different standard. The Table 3.2.1 shows the voltages used by several families.
- These are the ideal voltage levels. But in practice, it is difficult to get these ideal voltages.
- Stray electric and magnetic field can induce voltages on the connecting wires between logic circuits. These unwanted signals are called noise and can sometimes cause the voltage at the input to a logic circuit to drop below  $V_{IH(\min)}$  or rise above  $V_{IL(\max)}$ , which could produce unpredictable operation.
- The noise immunity of a logic circuit refers to the circuit's ability to tolerate the noise without causing spurious changes in the output voltage. To avoid this problem due to noise, voltage level  $V_{IH(\min)}$  is kept at a few fraction of volts below  $V_{OH(\min)}$  and voltage level  $V_{IL(\max)}$  is kept above  $V_{OL(\max)}$ , at the design time. This is illustrated in Fig. 3.2.2.

Table 3.2.1

Family	Logic 0	Logic 1
TTL	0 V	+5 V
BCI	-1.7 V	-0.9 V
CMS	0 V	3 - 15 V

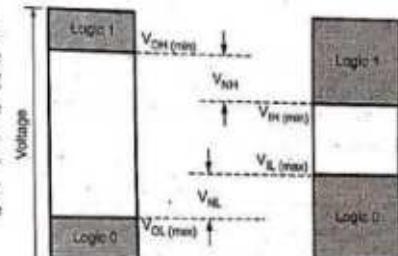


Fig. 3.2.2 Noise margins

- As shown in Fig. 3.2.2,  $V_{NH}$  is the difference between the lowest possible HIGH output,  $V_{OH(\min)}$  and the minimum voltage,  $V_{IH(\min)}$  required for a HIGH input. This voltage difference,  $V_{NH}$  is called high-state noise margin.
- Similarly, we have low-state noise margin. It is the voltage difference between the largest possible low output,  $V_{OL(\max)}$  and the maximum voltage,  $V_{IL(\max)}$  required for a LOW input.
- In short we can write as,

$$V_{NH} = V_{OH(\min)} - V_{IH(\min)} \text{ and}$$

$$V_{NL} = V_{IL(\max)} - V_{OL(\max)}$$

- The noise margin allows the digital circuit to function properly if noise voltages are within the limits of  $V_{NH}$  and  $V_{NL}$  for a particular logic family.

**Fan-In and Fan-out**

- In a digital system, we typically find many types of digital ICs interconnected to perform various functions. In these situations, the output of a logic gate may be connected to the inputs of several other similar gates. The maximum number of inputs of several gates that can be driven by the output of a logic gate is decided by the parameter called fan-out. In general, the fan-out is defined as the maximum number of inputs of the same IC family that the gate can drive maintaining its output levels within the specified limits. For example, a logic gate with fan-out 10 can drive maximum 10 logic inputs from the same family. It depends on current sourcing and sinking capacity of input and output signals of same IC family.
- The fan-in of a digital logic gate refers to the number of inputs. For example, an inverter has a fan-in of 1, a 2-input NOR gate has a fan-in of 2, a 4-input NAND gate has a fan-in of 4 and so on. A logic designer has to select the fan-in of the gate to accommodate the number of inputs. At the hardware level, however, the fan-in provides information about the intrinsic speed of the gate itself. In general, the propagation delay increases with the fan-in. This means that 2-input NAND gate is faster than the 4-input NAND if both are from same logic family.

**Current Sinking**

- A device output is said to sink current when current flows from the power supply, through the load and through the device output to ground. This is illustrated in Fig. 3.2.3 (a).

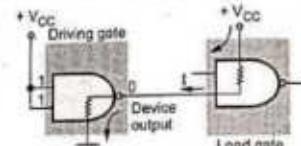


Fig. 3.2.3 (a) Current sinking

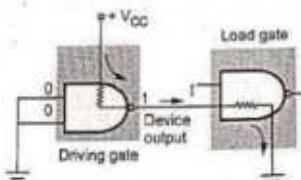


Fig. 3.2.3 (b) Current sourcing

**Speed Power Product (Figure of Merit)**

- In general, for any digital IC, it is desirable to have shorter propagation delays (higher speed) and lower values of power dissipation. There is usually a trade-off between switching speed and power dissipation in the design of a logic circuit i.e. speed is gained at the expense of increased power dissipation. Therefore, a common means for measuring and comparing the overall performance of an IC family is the Speed-Power Product (SPP). It is also called Figure of Merit.

**Illustrative Examples**

**Example 3.2.1** If a manufacturer specifies the minimum logical 1 at a gate output as 4.0 V and also specifies that any voltage down up to 3.6 V will be considered as logical 1, find the noise margin.

Solution : Given :  $V_{OH(min)} = 4.0 \text{ V}$  and  $V_{IH(max)} = 3.6 \text{ V}$

$$\text{Noise margin} = V_{OH(min)} - V_{IH(max)} = 4.0 \text{ V} - 3.6 \text{ V} = 0.4 \text{ V}$$

**Example 3.2.2** Determine the fan-out given  $I_{OL(max)} = 40 \mu\text{A}$  and  $I_{OH(max)} = 400 \mu\text{A}$ .

$$\text{Solution : Fan-out} = \frac{I_{OH(max)}}{I_{OL(max)}} = \frac{400 \mu\text{A}}{40 \mu\text{A}} = 10$$

**Example 3.2.3** A certain gate draws 2 mA when its output is high and 3.6 mA when its output is low. What is the average power dissipation if  $V_{CC}$  is 5 V and it is operated on a 50% duty cycle?

Solution : Given  $I_{OH} = 2 \text{ mA}$

$$I_{OL} = 3.6 \text{ mA}$$

$$P_D = \frac{2 \text{ mA} + 3.6 \text{ mA}}{2} \times 5 = 2.8 \text{ mA} \times 5 = 14 \text{ mW}$$

**Example 3.2.4** For a certain IC family, propagation delay is 10 ns with an average power dissipation of 6 mW. What is its speed-power product?

Solution : The speed-power product is given as,

$$\begin{aligned} \text{SPP} &= \text{Propagation delay} \times \text{Average power dissipation} \\ &= 10 \text{ ns} \times 6 \text{ mW} = 60 \text{ pico joules (pJ)} \end{aligned}$$

**Example 3.2.5** If  $V_{CC} = 5 \text{ V}$ ,  $I_{CE(0)} = 22 \text{ mA}$ ,  $I_{CEH} = 5 \text{ mA}$ ,  $V_{CH} = 2.4 \text{ V}$ ,  $V_{IH} = 2 \text{ V}$ ,

$$V_{OL} = 0.45 \text{ V}$$

$$V_{H_L} = 0.8 \text{ V}$$

$$t_{PHL} = 15 \text{ nsec}$$

$$t_{PLH} = 22 \text{ nsec}$$

$$I_{OH} = 40 \mu\text{A}$$

$$I_{OL} = 5.2 \text{ mA}$$

$$i) \text{ Noise margin logic 1 and logic 0. } ii) \text{ Power dissipation. }$$

$$iii) \text{ Propagation delay and figure of merit. } iv) \text{ Fan-out. }$$

Solution :

i) Noise margin logic 1

$$V_{NH} = V_{OH(min)} - V_{IH(max)} = 2.4 \text{ V} - 2 \text{ V}$$

$$V_{NH} = 0.4 \text{ V}$$

## Noise margin logic 0

$$V_{NL} = V_{IL(\max)} - V_{OL(\max)} = 0.8 \text{ V} - 0.45 \text{ V}$$

$$V_{NL} = 0.35 \text{ V}$$

iii) Power dissipation =  $I_{CC(\text{avg})} \times V_{CC}$

$$I_{CC(\text{avg})} = \frac{I_{CC(0)} + I_{CC(1)}}{2} = 15 \text{ mA}$$

$$\therefore \text{Power dissipation} = 15 \text{ mA} \times 5 \text{ V} = 75 \text{ mW}$$

## iii) Propagation delay

$$t_p = \max(t_{PLH}, t_{PHL})$$

$$t_p = t_{PLH} = 22 \text{ nsec}$$

OR  $t_p = \frac{1}{2}(t_{PLH} + t_{PHL}) = \frac{1}{2}(22 + 15) \text{ nsec} = 18.5 \text{ nsec}$

Figure of merit = SPP = Delay  $\times$  Power dissipation =  $18.5 \text{ nsec} \times 75 \text{ mWatt}$

$$= 1387.5 \text{ pico joule}$$

iv) Fan-out =  $\left| \frac{I_{OH(\max)}}{I_{IH(\min)}} \right| = \frac{52 \text{ mA}}{40 \mu\text{A}} = 130$

## Review Questions

1. List various characteristics of digital ICs and explain their significance in brief.
2. Define 1)  $V_{OH}$ ,  $V_{IL}$ , 2)  $V_{OL}$ ,  $V_{IL}$ , 3) Noise margin.
3. Define fan-out.
4. Define fan-in.
5. What is propagation delay?
6. What is noise margin?
7. Define current sinking and current sourcing.
8. What is speed power product? List the characteristics of digital ICs.

GTU : Summer-18, Marks 2

9. Explain following characteristics of logic families with examples:
  - i) Figure of merit ii) Fan-out iii) Voltage parameters iv) Speed v) Noise margin.
10. Explain parameter's of logic families.
11. Explain the current parameters of logic families.
12. Define followings with respect to logic families.
  - i) Fan-in ii) Fan out iii) Noise Margin iv) Propagation delay

GTU : Summer-15, Marks 4

GTU : Winter-15, Mark 1

13. Define fan-out.
14. Explain following terms w.r.t. Digital Logic Family : 1)Fan - in 2) Noise Margin 3) Power Dissipation.

GTU : Winter-18, Marks 3

## 3.3 Transistor-Transistor Logic (TTL)

GTU : Winter-18

- Transistor-transistor logic, TTL, is named for its dependence on transistors alone to perform basic logic operations. The first version, which is now known as standard TTL, was developed in 1965 and is rarely used in today's systems. Through the years, the basic design has been modified to improve its performance in several respects and as a consequence, a number of subfamilies have evolved.

## 3.3.1 2-input TTL NAND Gates

- The Fig. 3.3.1 (a) shows the circuit diagram of 2-input NAND gate.
- Its input structure consists of multiple-emitter transistor and output structure consists of totem-pole output.
- $Q_1$  is an NPN transistor having two emitters, one for each input to the gate.
- Although this circuit looks complex, we can simplify its analysis by using the diode equivalent of the multiple-emitter transistor  $Q_1$ , as shown in Fig. 3.3.1 (b).

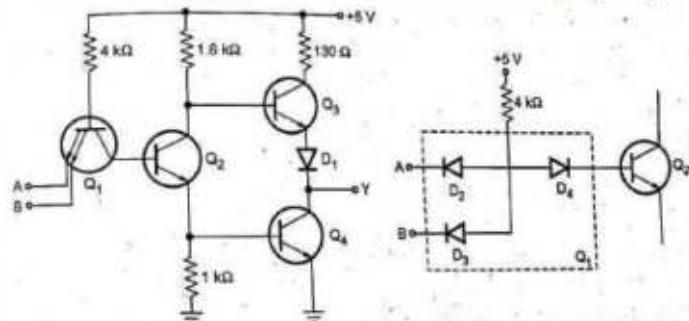


Fig. 3.3.1 (a) Two input TTL NAND gate

Fig. 3.3.1 (b) Diode equivalent for  $Q_1$ 

- Diodes  $D_2$  and  $D_3$  represent the two E-B junctions of  $Q_1$  and  $D_4$  is the collector-base (C-B) junction.
- The input voltages A and B are either LOW (ideally grounded) or HIGH (ideally +5 volts).
- If either A or B or both are low, the corresponding diode conducts and the base of  $Q_1$  is pulled down to approximately 0.7 V. This reduces the base voltage of  $Q_2$  to almost zero. Therefore,  $Q_2$  cuts off. With  $Q_2$  open,  $Q_4$  goes into cut-off and the

$Q_3$  base is pulled HIGH. Since  $Q_3$  acts as an emitter follower, the  $Y$  output is pulled up to a HIGH voltage.

- When A and B both are HIGH, the emitter diode of  $Q_1$  are reversed biased making them off. This causes the collector diode  $D_4$  to go into forward conduction. This forces  $Q_2$  base to go HIGH. In turn,  $Q_4$  goes into saturation, producing a low output.
- Table 3.3.1 summarizes all input and output conditions.

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

Table 3.3.1 Truth table for 2-input NAND gate

- Without diode  $D_1$  in the circuit,  $Q_3$  will conduct slightly when the output is low. To prevent this, the diode is inserted; its voltage drop keeps the base-emitter diode of  $Q_3$  reverse-biased. In this way, only  $Q_4$  conducts when the output is low.

### 3.3.2 3-Input TTL NAND Gate

- The Fig. 3.3.2 shows the three input TTL NAND gate. It is same as that of two input TTL NAND gate except that its  $Q_1$ (NPN) transistor has three emitters instead of two.
- For three input NAND gate if all the inputs are logic 1 then and then only output is logic 0; otherwise output is logic 1. The operation is similar to the 2-input NAND gate.
- The Table 3.3.2 shows the truth table for 3-input NAND gate.  
(See Table 3.3.2 on next page)

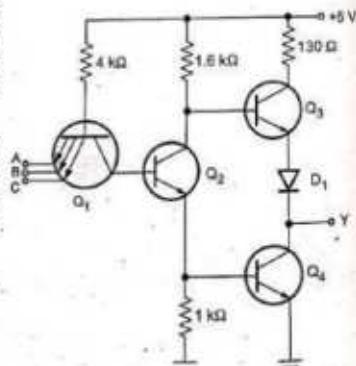


Fig. 3.3.2 Three input TTL NAND gate

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

Table 3.3.2 Truth table of 3-input NAND gate

### 3.3.3 Totem-Pole Output / Active Pull-Up

- Fig. 3.3.3 shows an highlighted output configuration. Transistor  $Q_3$  and  $Q_4$  form a totem-pole. Such a configuration is known as active pull-up or totem pole output.

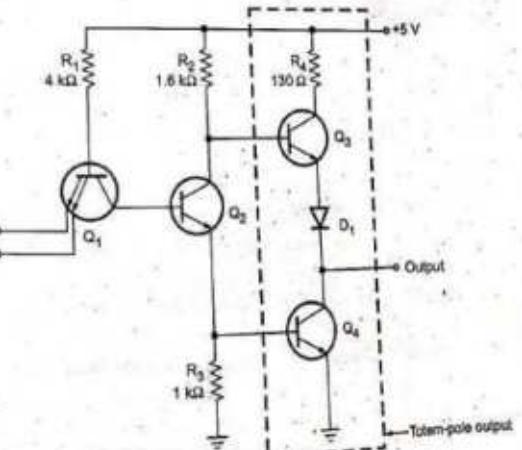


Fig. 3.3.3 Two input NAND gate with totem-pole output

- The active pull-up formed by  $Q_3$  and  $Q_4$  has specific advantage. Totem-pole transistors are used because they produce a LOW output impedance. Either  $Q_3$  acts as an emitter follower (HIGH output), or  $Q_4$  is saturated (LOW output). When  $Q_3$  is conducting, the output impedance is approximately  $70\ \Omega$ ; when  $Q_4$  is saturated, the output impedance is only  $12\ \Omega$ . Either way, the output impedance is low.
- The output voltage can change quickly from one state to the other because any stray output capacitance is rapidly charged or discharged through the low output impedance. Thus the propagation delay is low in totem-pole TTL logic.
- Fig. 3.3.4 shows static analysis of one gate in a 7400 quad 2-input NAND gate, including the totem-pole output.

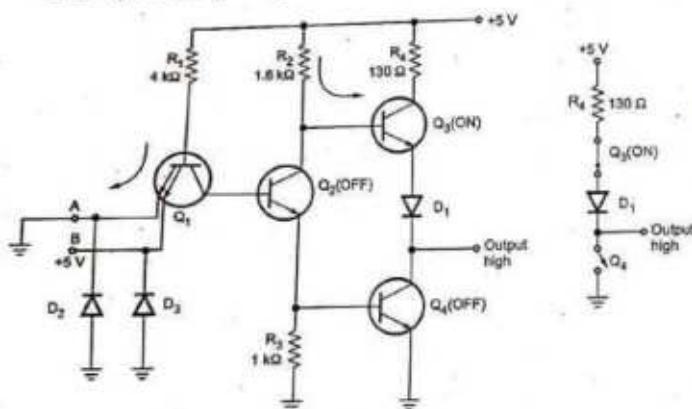


Fig. 3.3.4 Static analysis when output is HIGH

- Diodes  $D_2$  and  $D_3$  have been added at the input terminals. These diodes protect the circuit from large negative transients on input lines. If an input attempts to go more than about 1 V negative, the protective diode conducts like a short circuit to ground.
- $A = 0\text{ V}$  and  $B = +5\text{ V}$ , this makes  $Q_1$  to conduct and  $Q_2$  to switch off. Since  $Q_2$  is like an open switch, no current flows through it. Instead, current flows through the  $1.6\text{ k}\Omega$  resistor and into the base of  $Q_3$ , turning it ON.  $Q_4$  remains off because there is no path through which it can receive base current.

- The equivalent switches in the totem-pole under these conditions are shown in Fig. 3.3.5. The output current,  $I_L$  flows through  $R_4(130\ \Omega)$  and diode  $D_1$ . Therefore, the HIGH output voltage is given as,
- $$V_{OH} = V_{CC} - V_{CE(on)} - V_D - I_L \times (130\ \Omega)$$
- where  $V_D$  is the forward drop across diode  $D_1$ , about  $0.7\text{ V}$ .
- $V_{CE(on)}$  is the saturation voltage of  $Q_3$ , about  $0.1\text{ V}$ .

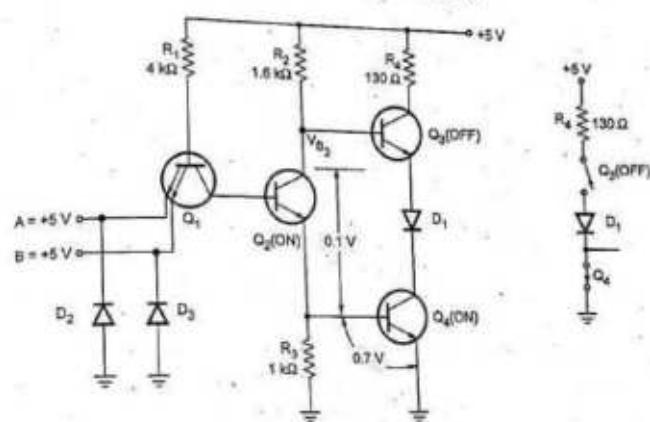


Fig. 3.3.5 Static analysis when output is LOW

- We have seen that when both inputs are HIGH  $Q_2$  is ON and it drives  $Q_4$  turning it ON. Note that in this condition the voltage at the base of  $Q_3$  equals the sum of the base-to-emitter drop of  $Q_4$  and the  $V_{CE(on)}$  of  $Q_2$ .
- $$V_{B3} = V_{BE}(Q_4) + V_{CE(on)}(Q_2) = 0.7\text{ V} + 0.1\text{ V}$$
- $$= 0.8\text{ V}$$
- Now we can easily understand the purpose of diode  $D_1$ . It does not allow base-emitter junction of  $Q_3$  to be forward-biased and thus ensures that  $Q_3$  remains OFF when  $Q_4$  is ON.

**Example 3.3.1** A two input NAND gate has  $V_{CC} = +5\text{ V}$  and  $1\text{ k}\Omega$  load connected to its output. Calculate the output voltage

a) When both inputs are LOW b) When both inputs are HIGH.

Solution : a) When both inputs of NAND gate are LOW, the output is HIGH and it is given as,

$$\begin{aligned} V_{OH} &= V_{CC} - V_{CE(sat)} - V_D - I_L \times (130 \Omega) = 5 - 0.1 - 0.7 - I_L (130 \Omega) \\ &= 4.2 \text{ V} - I_L (130 \Omega) \end{aligned} \quad \dots (1)$$

where the load current is,

$$I_L = \frac{V_{OH}}{R_L} = \frac{V_{OH}}{1 \text{ k}\Omega}$$

Substituting for  $I_L$  in the equation (1) we get,

$$V_{OH} = 4.2 \text{ V} - \frac{V_{OH}}{1 \text{ k}\Omega} (130 \Omega)$$

$$\therefore V_{OH} + \frac{130 V_{OH}}{1000} = 4.2$$

$$\therefore 1130 V_{OH} = 4200$$

$$\therefore V_{OH} = \frac{4200}{1130} = 3.716 \text{ V}$$

b) When both inputs of NAND gate are HIGH, the output is LOW and it is given as,

$$V_{OL} = V_{CE(sat)} = 0.1 \text{ V}$$

### 3.3.4 Input and Output Currents

- We have seen that TTL output stage is a totem pole. It consists of two transistors  $Q_3$  and  $Q_4$ . When  $Q_3$  is ON, the output is HIGH and it supplies current  $I_{OH}$  to drive the load. Hence  $Q_3$  is called a pull-up transistor. When  $Q_4$  is ON, the output is low and it draws current from the load to pull the load voltage down. Hence transistor  $Q_4$  is called a pull-down transistor.
- Since current flows out of the totem pole when the output is HIGH,  $Q_3$  acts as a current source to a load. When the output is low, current flows into  $Q_4$ , and we say that  $Q_4$  is a current sink.
- Fig. 3.3.6 shows the output stage of the TTL driver connected to the input stage of the TTL load.
- In Fig. 3.3.6 (a) the driver output is low and  $Q_4$  sinks current from the forward biased base-emitter junction of the input transistor of the load. This current is approximately 1.6 mA, and is designated  $I_{OL}$  at the output and  $I_{IL}$  at the input.
- In Fig. 3.3.6 (b) the driver output is HIGH and  $Q_3$  source current to the load. It is small leakage current supplied to the reverse biased emitter base junction of the

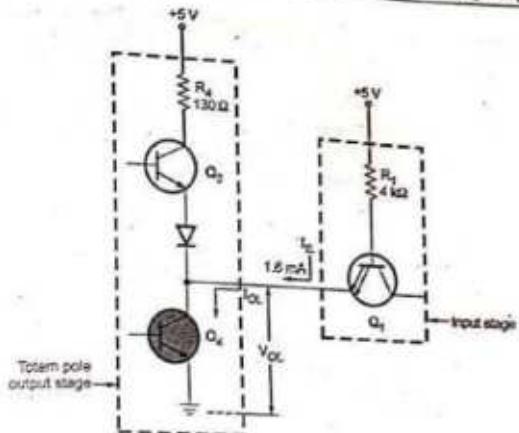


Fig. 3.3.6 (a)  $Q_4$  acting as a current sink when output is LOW

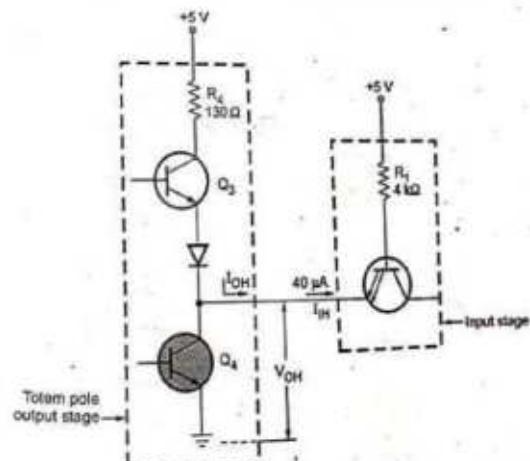


Fig. 3.3.6 (b)  $Q_3$  acting as a current source when output is HIGH

input transistor of the load. It is approximately  $40 \mu A$ . This current is designated as  $I_{OH}$  at the output and  $I_{IH}$  at the input. By convention current flowing into a device is positive and current flowing out is negative. Therefore, manufacturers specify negative values for  $I_{OH}$  and  $I_{IH}$ .

- Fig. 3.3.7 shows a TTL output connected to the several TTL loads. When the output is HIGH, it is necessary to supply load current ( $I_{IH}$ ) to each TTL load, so  $Q_3$  must be capable of sourcing the sum of these currents.

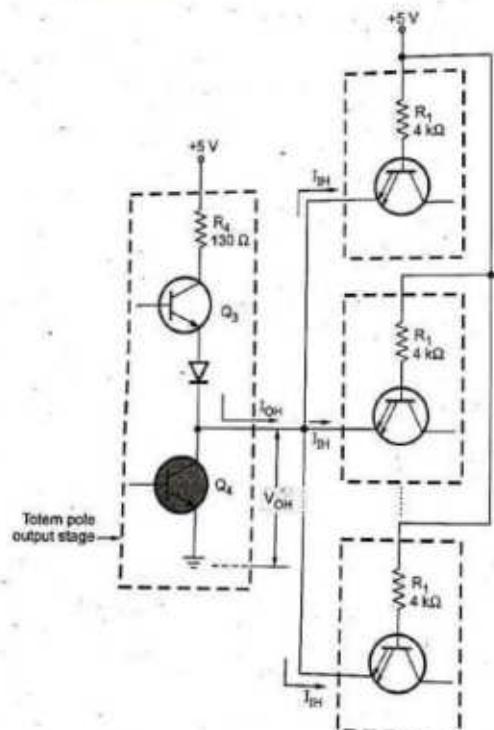


Fig. 3.3.7 (a) When output is HIGH

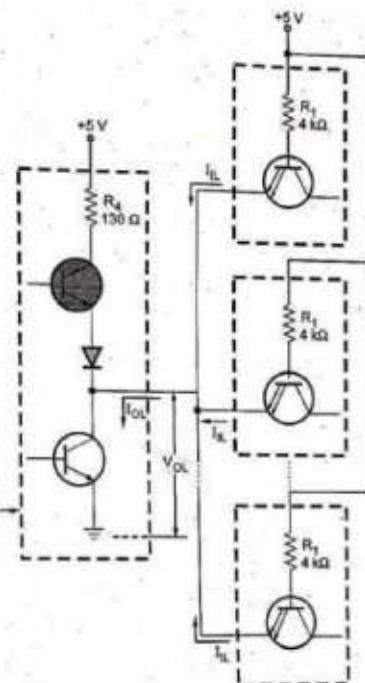


Fig. 3.3.7 (b) When output is LOW

- Fan-out is the maximum number of loads belonging to same family that a logic gate can drive,

$$\text{So } \text{Fan-out} = \frac{|I_{OH(\max)}|}{I_{IH(\max)}}$$

- For standard TTL,  $I_{OH(\max)} = -400 \mu A$  and  $I_{IH(\max)} = 40 \mu A$ .

Therefore,

$$\text{Fan-out} = \frac{|-400 \mu A|}{40 \mu A} = 10$$

- When the output is low, each load supplies current to the totem pole, so  $Q_4$  must be capable of sinking the sum of these currents. In this case, fan-out is defined as,

$$\text{Fan-out} = \frac{|I_{OL(\max)}|}{|I_{IL(\max)}|}$$

- For standard TTL,

$$I_{OL(\max)} = 16 \text{ mA} \text{ and } I_{IL(\max)} = -1.6 \text{ mA}$$

Therefore,

$$\text{Fan-out} = \frac{16 \text{ mA}}{|-1.6 \text{ mA}|} = 10$$

- We have seen that fan-out can be determined in two ways. Here, in both cases fan-out is 10. But if they differ, the actual fan-out is always the smaller of the two computations.

### 3.3.5 Wired Logic - Open Collector Output

- One problem with totem pole output is that two outputs cannot be tied together.
- See the Fig. 3.3.8, where the totem pole outputs of two separate gates are connected together at point X.

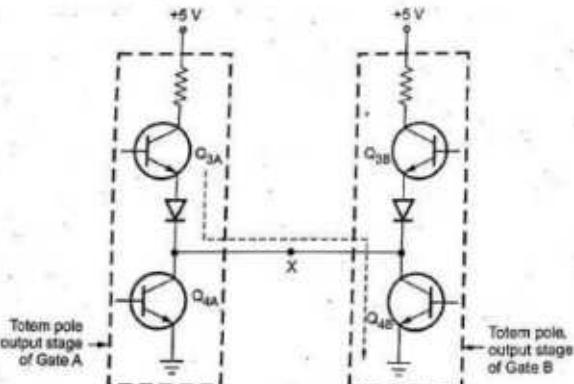


Fig. 3.3.8 Totem-pole outputs tied together can produce harmful current

- Suppose that the output of gate A is high ( $Q_{3A}$  ON and  $Q_{4A}$  OFF) and the output of gate B is low ( $Q_{3B}$  OFF and  $Q_{4B}$  ON). In this situation transistor  $Q_{4B}$  acts as a load for  $Q_{3A}$ .

- Since  $Q_{4B}$  is a low resistance load, it draws high current around 55 mA. This current might not damage  $Q_{3A}$  or  $Q_{4B}$  immediately, but over a period of time can cause overheating and deterioration in performance and eventual device failure.

- Some TTL devices provide another type of output called open collector output. The outputs of two different gates with open collector output can be tied together. This is known as wired logic.

- Fig. 3.3.9 shows a 2-input NAND gate with an open-collector output. It eliminates the pull-up transistor  $Q_3$ ,  $D_1$  and  $R_3$ . The output is taken from the open collector terminal of transistor  $Q_4$ .

- Because the collector of  $Q_4$  is open, a gate like this will not work properly until you connect an external pull-up resistor, as shown in Fig. 3.3.10.
- When  $Q_4$  is ON, output is low and when  $Q_4$  is OFF output is tied to  $V_{CC}$  through an external pull up resistor.
- As mentioned earlier, the open collector outputs of two or more gates can be connected together, as shown in the Fig. 3.3.11 (a). The connection is called a wired-AND and represented schematically by the special AND gate symbol as shown in Fig. 3.3.11 (b). (See Fig. 3.3.11 on next page)

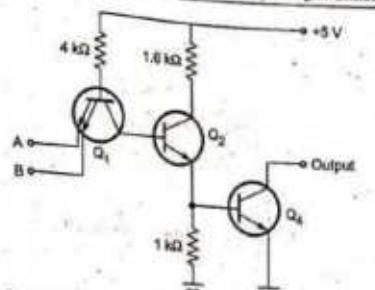


Fig. 3.3.9 Open collector 2-input TTL NAND gate

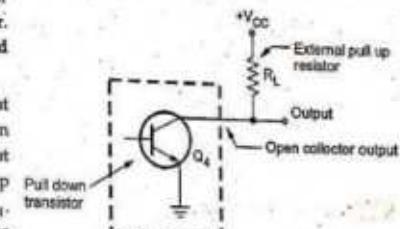


Fig. 3.3.10 Open collector output with pull-up resistor

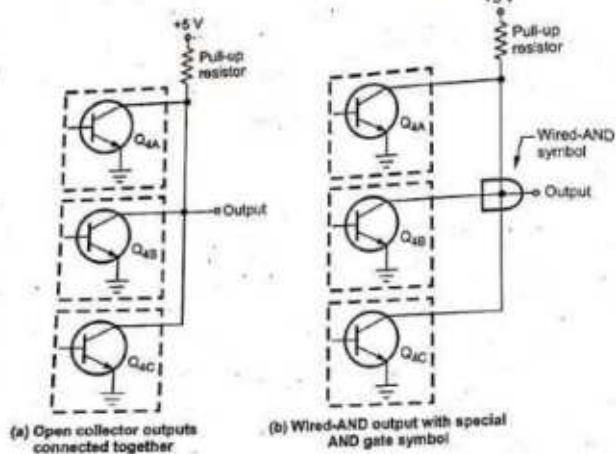


Fig. 3.3.11

### 3.3.6 Comparison between Totem-Pole and Open-Collector Outputs

Table 3.3.3 summarizes the difference between totem-pole and open collector outputs.

Sr. No.	Totem-pole	Open collector
1.	Output stage consists of pull-up transistor ( $Q_3$ ), diode resistor and pull-down transistor ( $Q_4$ ).	Output stage consists of only pull-down transistor.
2.	External pull-up resistor is not required.	External pull-up resistor is required for proper operation of gate.
3.	Output of two gates cannot be tied together.	Output of two gates can be tied together using wired AND technique.
4.	Operating speed is high.	Operating speed is low.

Table 3.3.3 Comparison of totem-pole and open collector output

### 3.3.7 Standard TTL Characteristics

Let us see the characteristics of standard TTL family.

#### Supply voltage and temperature range

- Both the 74 series and 54 series operate on supply voltage of 5 V.
- The 74 series works reliably over the range 4.75 V to 5.25 V, while the 54 series can tolerate a supply variation of 4.5 to 5.5 V.
- The 74 series devices are guaranteed to work reliably over a temperature range of 0 to 70°C where as 54 series devices can handle temperature variations from -55 to +125°C.
- From the above values we can say that 54 series devices have greater tolerance of voltage and temperature variations. Hence, these devices are used where it is necessary to maintain reliable operation over an extreme range of conditions. For example, in military and space application.
- The only disadvantage of these devices is that they are expensive.

#### Voltage levels and noise margin

Table 3.3.4 shows the input and output logic voltage levels for the standard 74 series. The minimum and maximum values shown in the Table 3.3.4 are for worst case conditions of power supply, temperature and loading conditions.

- In the worst case, there is a difference of 0.4 V between the driver output voltages and the required load input voltages. For instance, the worst-case low values are

$$V_{OL(max)} = 0.4 \text{ V driver output}$$

$$V_{IL(max)} = 0.8 \text{ V load input}$$

Similarly, the worst-case high values are

$$V_{OH(min)} = 2.4 \text{ V driver output}$$

$$V_{IH(min)} = 2 \text{ V load input}$$

Table 3.3.4 Voltage levels

- In either case, the difference is 0.4 V. This difference is called noise margin.
- For TTL, Low state noise margin,  $V_{NL}$ , and high state noise margin,  $V_{NH}$ , both are equal and 0.4 V. This is illustrated in Fig. 3.3.12. It provides built-in protection against noise. It ensures reliable operation of the device for induced noise voltages less than 0.4 V. (See Fig. 3.3.12 on next page)

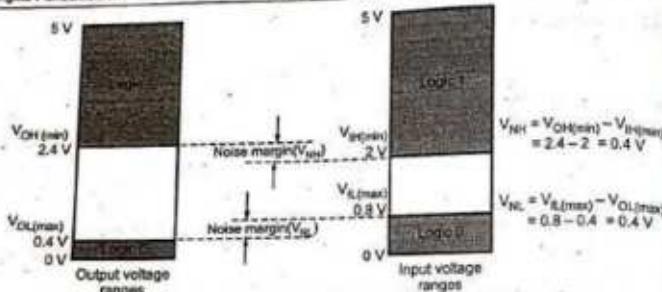


Fig. 3.3.12 TTL logic levels and noise margin

**Power dissipation**

A standard TTL gate has an average power dissipation of about 10 mW. It may vary from this value because of signal levels, tolerances etc.

**Propagation delay**

The propagation delay time is the time it takes for the output of a gate to change after the inputs have changed. The propagation delay time of a TTL gate is approximately 10 nanoseconds.

**Fan-out**

A standard TTL output can typically drive 10 standard TTL inputs. Therefore, standard TTL has fan-out 10.

Table 3.3.5 summarizes the characteristics of standard TTL (74XX).

Characteristics	Values
Supply voltage	For 74 series - (4.75 to 5.25) units For 54 series - (4.5 to 5.5) units
Temperature range	For 74 series - (0 °C to 70 °C) For 54 series - (-55 °C to 125 °C)
Voltage levels	$V_{OL(\max)} = 0.4 \text{ V}$ $V_{OH(\min)} = 2.4 \text{ V}$ $V_{IL(\max)} = 0.8 \text{ V}$ $V_{IH(\min)} = 2.0 \text{ V}$
Noise margin	0.4 V
Power dissipation	10 mW per gate
Propagation delay	Typically 10 ns
Fan-out	10

Table 3.3.5 Standard TTL characteristics

**3.3.6 Advantages and Disadvantages of TTL Family****Advantages of TTL**

1. High speed operation. Fastest among the saturated logic families. The propagation delay time is about 10 ns.
2. Moderate power dissipation.
3. Available in commercial and military versions.
4. Available for wide range of functions.
5. Low cost.
6. Moderate packaging density.

**Disadvantages of TTL**

1. Higher power dissipation than CMOS.
2. Lower noise immunity than CMOS.
3. Less fan-out than CMOS.

**3.3.9 Unconnected Inputs**

- If any input of TTL gate is left open or unconnected, then the corresponding base emitter junction of the input transistor  $Q_1$  is reverse biased. See Fig. 3.3.13 (a). Therefore, the open or unconnected input behaves exactly same way as if a logic 1 is applied to that input. Thus, in TTL ICs all the unconnected inputs are treated as logic 1. However, it is recommended that the unused inputs should be either connected to some used input or tied to  $V_{CC}$  through pull-up resistor, as shown in the Fig. 3.3.13 (b).

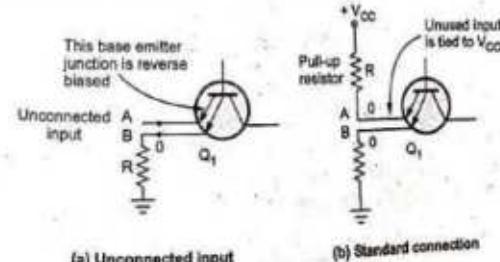


Fig. 3.3.13.

**3.3.10 Tri-State TTL Inverter**

- The tristate configuration is a third type of TTL output configuration.
- It utilizes the high-speed operation of the totem-pole arrangement while permitting outputs to be wired-ANDed (connected together).
- It is called tristate TTL because it allows three possible output stages : HIGH, LOW and high-impedance.
- Transistor  $Q_3$  is ON when output is HIGH and  $Q_4$  is ON when output is LOW.
- In the high impedance state both transistors, transistors  $Q_3$  and  $Q_4$  in the totem-pole arrangement are turned OFF. As a result, the output is open or floating, it is neither LOW nor HIGH.
- Fig. 3.3.14 shows the simplified circuit for tristate inverter. It has two inputs A and E.

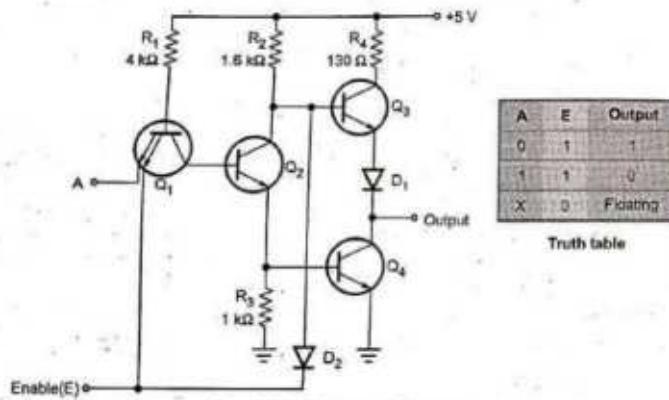


Fig. 3.3.14 Tristate TTL Inverter

- A is the normal logic input whereas E is an ENABLE input.
- When ENABLE input is HIGH, the circuit works as a normal inverter. Because when E is HIGH, the state of the transistor  $Q_1$  (either ON or OFF) depends on the logic input A.
- When ENABLE input is LOW, regardless of the state of logic input A, the base-emitter junction of  $Q_1$  is forward biased and as a result it turns ON.
- This shunts the current through  $R_1$  away from  $Q_2$  making it OFF.

- As  $Q_2$  is OFF, there is no sufficient drive for  $Q_4$  to conduct and hence  $Q_4$  turns OFF.
- The LOW at ENABLE input also forward-biases diode  $D_2$ , which shunt the current away from the base of  $Q_3$ , making it OFF.
- In this way, when ENABLE input is LOW, both transistors are OFF and output is at high impedance state.

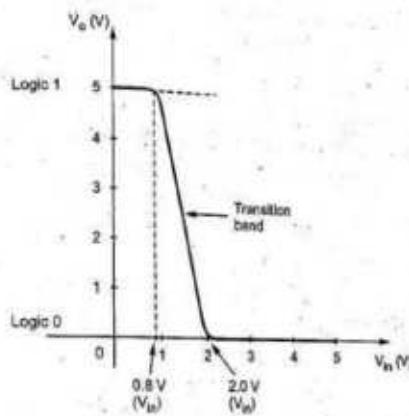
**Transfer characteristics**

Fig. 3.3.15 Transfer characteristics of TTL Inverter

- Fig. 3.3.15 shows transfer characteristics of TTL inverter.
- For  $V_{in} \leq V_{IL\text{ (max)}} (V_{in} \leq 0.8 \text{ V})$ , the output voltage is High (5 V).
- For  $V_{IL\text{ (max)}} < V_{in} \leq V_{IH\text{ (min)}}$  the output voltage is undefined.
- For  $V_{in} \geq V_{IH\text{ (min)}} (V_{in} \geq 2.0 \text{ V})$ , the output voltage is low (0 V).

**3.3.11 Schottky TTL**

With standard TTL, high-speed TTL and low-power TTL, when transistors are ON, they are driven into hard saturation. This causes a surplus of carriers to be stored in the base. This excess charge carriers in the base region must be removed before the transistors is switched from ON to OFF. The time required to remove these carriers, called storage time, is responsible for high propagation delays or low switching times.

One way to counter this problem is to prevent the transistor from going into deep saturation when it is turned ON i.e. to restrict the forward bias of its base-to-collector junction to few tenths of a volt. This can be accomplished by connecting a diode across the base-to-collector junction, as shown in the Fig. 3.3.16 (a). The diode does not allow to increase the forward bias voltage at base-to-collector junction above its cut-in voltage. In other words, it clamps base-to-collector voltage upto its small cut-in voltage. As a result the junction cannot be heavily forward biased and the transistor is kept out of deep saturation.

We know that germanium diodes have less cut-in voltage but they cannot be fabricated in silicon integrated circuits. The Schottky diode, consisting of a silicon-metal junction with a cut-in voltage of 0.3 to 0.4 volts is ideal for this purpose. Fig. 3.3.16 (b) and (c) show the construction, and Fig. 3.3.16 (d) and (e) show Schottky clamped NPN transistor and its symbol.

Another important parameter of Schottky diode is that it has very little capacitance and fast recovery time. So it can be switched rapidly without storage time delays.

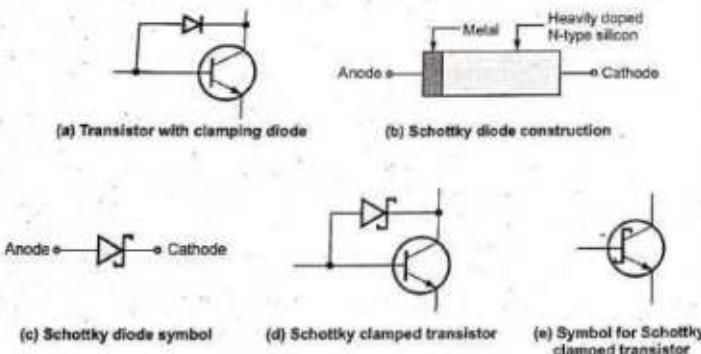


Fig. 3.3.16

## Review Questions

- With neat circuit diagram explain the operation of two-input TTL NAND gates.
- Draw circuit of TTL with totem-pole output and justify speed of operation is improved.
- Prove that TTL gates have a typical fan-out of 10.
- With the help of diagrams explain the current sinking and current sourcing actions for TTL gate output.
- Explain the wired logic output of TTL with neat diagram.

- Does totem-pole output support wired-logic connectors? Justify your answer.
- Why totem-pole output stages cannot be connected together except open collector logic?
- Define wired-AND connection.
- Give the advantages and disadvantages of totem-pole output stage arrangement.
- Explain the advantages of open collector output.
- Draw three input standard TTL NAND gate circuit and explain its operation.
- Explain the following characteristics of TTL logic families :
  - i) Power dissipation
  - ii) Noise margin
  - iii) Propagation delay
  - iv) Fan out
- Comment on unconnected inputs in TTL.
- Draw and explain the operation of Tri-state TTL inverter.
- Draw and explain the transfer characteristics of TTL inverter.
- Draw standard TTL inverter, discuss its operation and draw its transfer characteristics.
- Discuss the advantages and disadvantages of TTL Logic Family.
- Write a short note on TTL schottky family.
- What is the advantage of schottky TTL family.

GTU : Winter-18, Marks 3

## 3.4 CMOS Logic

## 3.4.1 CMOS Inverter

- Fig. 3.4.1 shows the basic CMOS inverter circuit. It consists of two MOSFETs in series in such a way that the P-channel device has its source connected to  $+V_{DD}$  (a positive voltage) and the N-channel device has its source connected to ground. The gates of the two devices are connected together as the common input and the drains are connected together as the common output.

- When input is HIGH, the gate of  $Q_1$  (P-channel) is at 0 V relative to the source of  $Q_1$  i.e.  $V_{GS1} = 0$  V. Thus,  $Q_1$  is OFF. On the other hand, the gate of  $Q_2$  (N-channel) is at  $+V_{DD}$  relative to its source i.e.  $V_{GS2} = +V_{DD}$ . Thus,  $Q_2$  is ON. This will produce  $V_{OUT} = 0$  V, as shown in the Fig. 3.4.2 (a). (See Fig. 3.4.2 on next page).

- When input is LOW, the gate of  $Q_1$  (P-channel) is at a negative potential relative to its source while  $Q_2$  has  $V_{GS} = 0$  V. Thus,  $Q_1$  is ON and  $Q_2$  is OFF. This produces output voltage approximately  $+V_{DD}$ , as shown in the Fig. 3.4.2 (b).

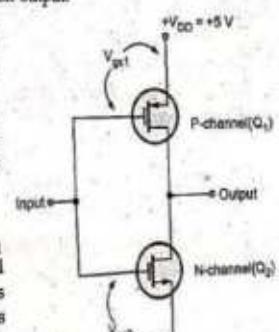


Fig. 3.4.1 CMOS inverter circuit

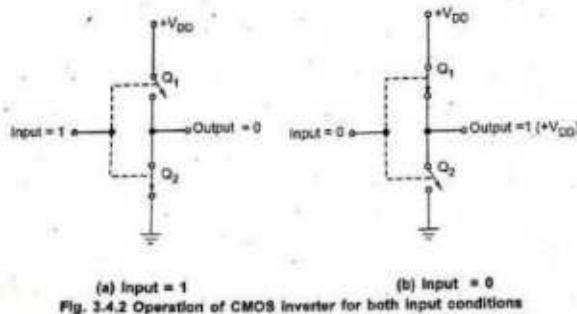


Table 3.5.1 summarizes the operation of CMOS inverter circuit

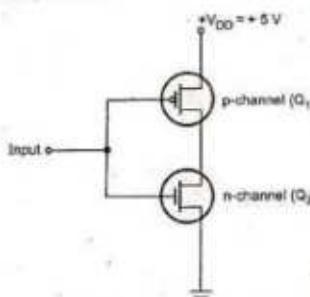
A	Q <sub>1</sub>	Q <sub>2</sub>	Output
0	ON	OFF	1
1	OFF	ON	0

Table 3.4.1 Truth table of Inverter

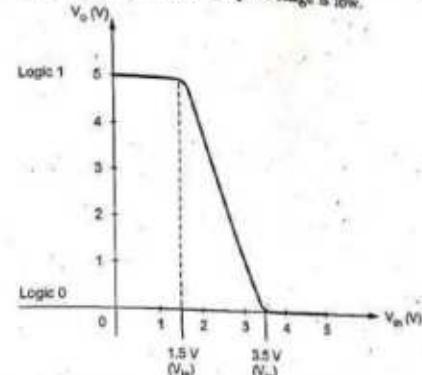
- The Fig. 3.4.3 shows different symbols used for the p-channel and n-channel transistors to reflect their logical behaviour.
- The n-channel transistor ( $Q_2$ ) is switched 'ON' when a HIGH voltage is applied at the input.
- The p-channel transistor ( $Q_1$ ) has the opposite behaviour; it is switched ON when a LOW voltage is applied at the input. It is indicated by placing bubble in the symbol.

#### Transfer Characteristics of CMOS Inverter

- Fig. 3.4.4 shows transfer characteristics of CMOS inverter. (See Fig. 3.4.4 on next page).
- For CMOS family  $V_{IH(\min)} = 3.5 \text{ V}$  and  $V_{IL(\max)} = 1.5 \text{ V}$ .
- For  $V_{in} < V_{IL(\max)}$  ( $V_{in} \leq 1.5 \text{ V}$ ), the output voltage is high.



- For  $V_{IL(\max)} \leq V_{in} \leq V_{IH(\min)}$ , the output voltage is in the transition phase.
- For  $V_{in} \geq V_{IH(\min)}$  ( $V_{in} \geq 3.5 \text{ V}$ ), the output voltage is low.



#### 3.4.2 CMOS NAND Gate

- Fig. 3.4.5 shows CMOS 2-input NAND gate. It consists of two P-channel MOSFETs,  $Q_1$  and  $Q_2$ , connected in parallel and two N-channel MOSFETs,  $Q_3$  and  $Q_4$  connected in series. (See Fig. 3.4.5 on next page)
- Fig. 3.4.5 (a) shows the equivalent switching circuit when both inputs are low. Here, the gates of both P-channel MOSFETs are negative with respect to their sources, since the sources are connected to  $+V_{DD}$ . Thus,  $Q_1$  and  $Q_2$  are both ON. Since the gate-to-source voltages of  $Q_3$  and  $Q_4$  (N-channel MOSFETs) are both 0 V, those MOSFETs are OFF. The output is therefore connected to  $+V_{DD}$  (HIGH) through  $Q_1$  and  $Q_2$  and is disconnected from ground, as shown in the Fig. 3.4.5 (b).
- Fig. 3.4.5 (c) shows the equivalent switching circuit when  $A = 0$  and  $B = +V_{DD}$ . In this case,  $Q_1$  is on because  $V_{GS1} = -V_{DD}$  and  $Q_4$  is off because  $V_{GS4} = +V_{DD}$ . MOSFETs  $Q_2$  and  $Q_3$  are off because their gate-to-source voltages are 0 V. Since  $Q_1$  is ON and  $Q_3$  is OFF, the output is connected to  $+V_{DD}$  and it is disconnected from ground. When  $A = +V_{DD}$  and  $B = 0$  V, the situation is similar (not shown); the output is connected to  $+V_{DD}$  through  $Q_2$  and it is disconnected from ground because  $Q_4$  is OFF. Finally, when both inputs are high ( $A = B = +V_{DD}$ ), MOSFETs  $Q_1$  and  $Q_2$  are both OFF and  $Q_3$  and  $Q_4$  are both ON. Thus, the output is connected to the ground through  $Q_3$  and  $Q_4$  and it is disconnected from  $+V_{DD}$ .

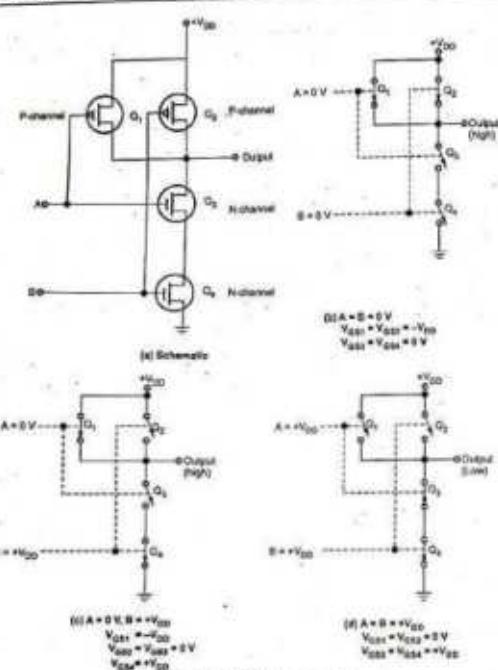


Fig. 3.4.5 CMOS NAND gate

P-channel MOSFET is ON when its gate voltage is negative with respect to its source whereas N-channel MOSFET is ON when its gate voltage is positive with respect to its source.

- The Table 3.4.2 summarizes the operation of 2-input CMOS NAND gate.
- The Fig. 3.4.6 shows the circuit diagram, function table and logic symbol of CMOS 3-input NAND gate.

A	B	$Q_1$	$Q_2$	$Q_3$	$Q_4$	Output
0	0	ON	ON	OFF	OFF	1
0	1	ON	OFF	OFF	ON	1
1	0	OFF	ON	ON	OFF	1
1	1	OFF	OFF	ON	ON	0

Table 3.4.2 Truth table of NAND gate

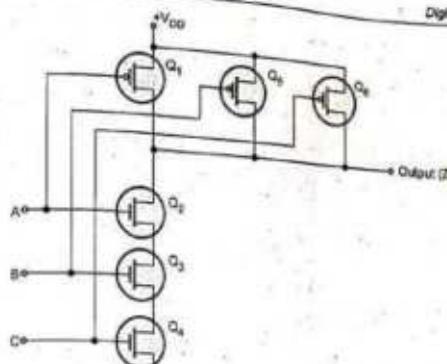


Fig. 3.4.6 (a) Circuit diagram

A	B	C	$Q_1$	$Q_2$	$Q_3$	$Q_4$	$Q_5$	$Q_6$	Z
0	0	0	ON	OFF	OFF	OFF	ON	ON	1
0	0	1	ON	OFF	OFF	ON	ON	OFF	1
0	1	0	ON	OFF	ON	OFF	OFF	ON	1
0	1	1	ON	OFF	ON	ON	OFF	OFF	1
1	0	0	OFF	ON	OFF	OFF	ON	ON	1
1	0	1	OFF	ON	OFF	ON	ON	OFF	1
1	1	0	OFF	ON	ON	OFF	OFF	ON	1
1	1	1	OFF	ON	ON	ON	OFF	OFF	0

Fig. 3.4.6 (b) Function table

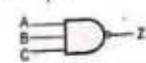


Fig. 3.4.6 (c) Logic symbol

**3.4.3 CMOS NOR Gate**

- Fig. 3.4.7 shows 2-input CMOS NOR gate. Here, P-channel MOSFETs  $Q_1$  and  $Q_2$  are connected in series and N-channel MOSFETs  $Q_3$  and  $Q_4$  are connected in parallel.
- Like NAND circuit, this circuit can be analyzed by realizing that a LOW at any input turns ON its corresponding P-channel MOSFET and turns OFF its corresponding N-channel MOSFET, and vice versa for a HIGH input. This is illustrated in Fig. 3.4.7.

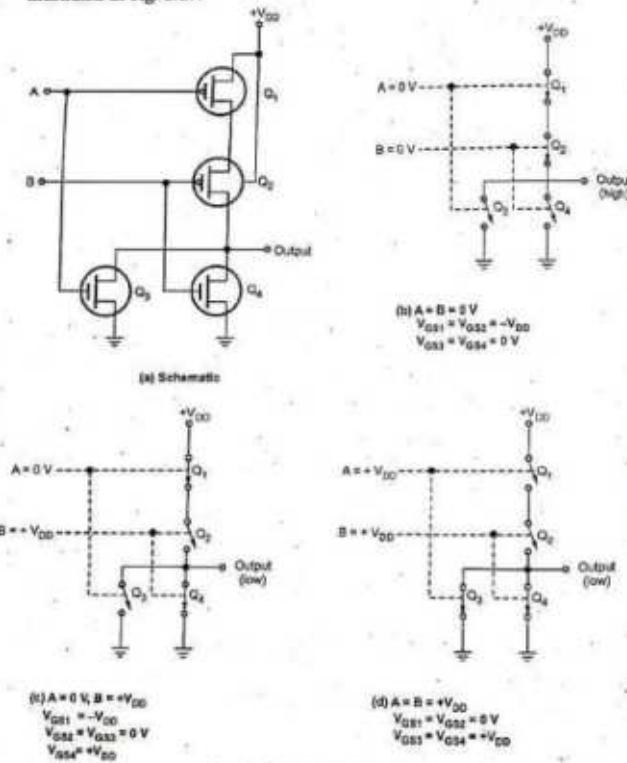


Fig. 3.4.7 CMOS NOR gate

TECHNICAL PUBLICATIONS™ An up beat for knowledge

- The Table 3.4.3 summarizes the operation of 2-input NOR gate.

A	B	$Q_1$	$Q_2$	$Q_3$	$Q_4$	Output
0	0	ON	ON	OFF	OFF	1
0	1	ON	OFF	OFF	ON	0
1	0	OFF	ON	ON	OFF	0
1	1	OFF	OFF	ON	ON	0

Table 3.4.3 Truth table for NOR gate

**3.4.4 CMOS Characteristics****Operating Speed :**

- Slower than TTL series.
- Approximately 25 to 100 ns depending on the subfamily of CMOS.
- It also depends on the power supply voltage.

**Voltage Levels and Noise Margins :**

- The voltage levels for CMOS varies according to their subfamilies. These are listed in Table 3.4.4.
  - Noise margins in table are calculated as follows.
- $$V_{NH} = V_{OH(\min)} - V_{IH(\min)}$$
- $$V_{NL} = V_{IL(\max)} - V_{CL(\max)}$$

**Fan-out :**

- The CMOS inputs have an extremely large resistance ( $10^{12} \Omega$ ) that draws essentially no current from the signal source. Each CMOS input, however, typically presents a 5 pF load to ground as shown in the Fig. 3.4.8. This input capacitance limits the number of CMOS inputs that one CMOS output can drive.
- The CMOS output has to charge and discharge the parallel combination of all the input capacitances. This charging and discharging time increases as we increase input capacitances.

Table 3.4.4

Parameter	CMOS series				
	4000 B	74 HC	74 HCT	74 AC	74 ACT
$V_{OH(\min)}$	3.5	3.5	2.0	3.5	2.0
$V_{OL(\max)}$	1.5	1.0	0.8	1.5	0.8
$V_{IH(\max)}$	4.95	4.9	4.9	4.9	4.9
$V_{IL(\max)}$	0.05	0.1	0.1	-0.1	0.1
$V_{VIH}$	1.45	1.4	2.9	1.4	2.9
$V_{VIL}$	1.45	0.9	0.7	1.4	0.7

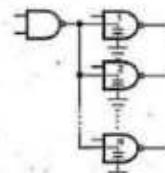


Fig. 3.4.8 One CMOS output driving several CMOS inputs

TECHNICAL PUBLICATIONS™ An up beat for knowledge

number of loads. Typically, each CMOS load increases the driving circuit's propagation delay by 3 ns. Thus, fan-out for CMOS depends on the permissible maximum propagation delay.

3. Typically, CMOS outputs are limited to a fan-out of 50 for low-frequency operation ( $\leq 1$  MHz). Of course, for high-frequency operation the fan-out would have to be less.

#### Power Dissipation ( $P_D$ ):

1. The power dissipation of a CMOS IC is very low as long as it is in a d.c. condition.
2. Unfortunately, power dissipation of CMOS IC increases in proportion to the frequency at which the circuits are switching states.
3. For example, a CMOS NAND gate that has  $P_D = 10$  nW under d.c. conditions will have  $P_D = 0.1$  mW at a frequency of 100 kHz and 1 mW at 1 MHz.
4. When CMOS output switches from LOW to HIGH, a transient charging current has to be supplied to the load capacitance. Therefore, as the switching frequency increases, the average current drawn from  $V_{DD}$  also increases, resulting increase in power dissipation.

#### Propagation Delay :

1. The propagation delay in CMOS is the sum of delay due to internal capacitance and due to load capacitance.
2. The delay due to internal capacitance is called the intrinsic propagation delay.
3. The delay due to load capacitance can be approximated as follows:

$$t_p(C_L) = 0.5 R_o C_L \text{ seconds}$$

where  $t_p(C_L)$  is either  $t_{PLH}$  or  $t_{PHL}$ .

$R_o$  is the output resistance of the gate and  $C_L$  is the total load capacitance.

The  $R_o$  depends on the supply voltage and it can be approximated as

$$R_o = \frac{V_{CC}}{I_{os}}$$

where  $I_{os}$  is the short circuit output current.

#### Unconnected Inputs :

1. CMOS inputs should never be left disconnected. All CMOS inputs have to be tied either to a fixed voltage level (0 V or  $V_{DD}$ ) or to another input. This rule applies even to the inputs of extra unconnected logic gates on a chip.

2. An unconnected CMOS input is susceptible to noise and static charges that could easily bias both the P and N-channel MOSFETs in the conductive state, resulting in increased power dissipation and possible overheating.

#### static-charge Susceptibility (CMOS Hazards) :

1. Every CMOS device is vulnerable to the building up of electrical charge on its insulated gate. Recall that the relationship between charge  $Q$  and voltage  $V$  on a capacitor having capacitance  $C$  is

$$V = \frac{Q}{C}$$

2. Since the input capacitance at the gate is usually quite small (a few picofarads), a relatively small amount of charge can create a large voltage which may be greater than the breakdown voltage of a MOS gate (typically 100 V).

3. The primary source of charge is "static" electricity, usually produced by handling and the motion of various kinds of plastics and textiles.

4. The CMOS devices are protected against this static charge by on-chip diode-resistor network, as shown in the Fig. 3.4.9. These diodes are designed to turn ON and limit the size of the input voltage to well below any damaging value.

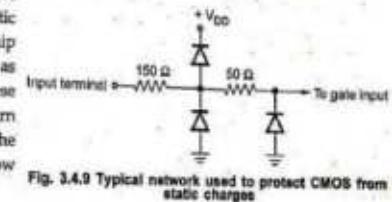


Fig. 3.4.9 Typical network used to protect CMOS from static charges

#### Latch-up :

1. CMOS integrated circuits contain parasitic PNP and NPN transistors : Transistors that exist because of the proximity of P and N materials embedded in the substrate. Their existence is not intentional but is unavoidable. Because of conducting paths between a pair of such transistors, a device can be triggered into a heavy conduction mode, known as latch-up.
2. This heavy conduction mode, results large current flow which can destroy IC.
3. Most CMOS circuits contain protective measures to prevent latch-up, but it can still occur if the manufacturers specified maximum ratings are exceeded.

#### 3.4.5 Wired Logic

- \* Fig. 3.4.10 shows two CMOS inverters with their outputs connected together.

- This circuit does not work properly when B input is logic 0 and A input is logic 1. In this situation,  $Q_1$  and  $Q_2$  are ON and large current flows through  $Q_3$  and  $Q_4$  damaging these transistors. Therefore, wired-logic must not be used for CMOS logic circuits.

#### 3.4.6 Open Drain Outputs

- CMOS gates are available with open drain outputs, as shown in Fig. 3.4.11.
- In open drain outputs, PMOS transistor is replaced by a diode  $D_1$  which provides protection from electrostatic discharge.
- Open drain gates can be used with external pull-up resistors to perform wired-AND operation, as discussed in TTL logic.

#### 3.4.7 Advantages and Disadvantages of CMOS Family

##### Advantages

- Consumes less power.
- Can be operated at high voltages, resulting in improved noise immunity.
- Fan-out is more.
- Better noise margin.

##### Disadvantages

- Susceptible to static charge.
- Switching speed low.
- Greater propagation delay.

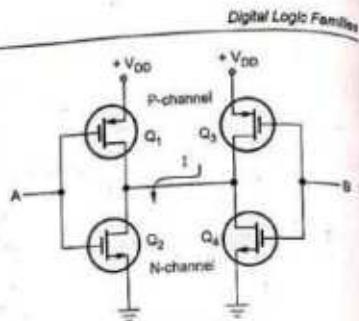


Fig. 3.4.10

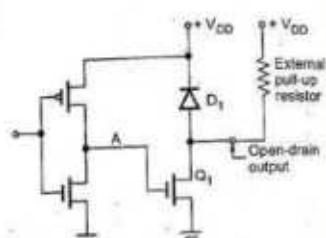


Fig. 3.4.11

#### Review Questions

- What are salient features of CMOS logic family?
  - Draw the structure of CMOS inverter gate. Explain its working.
  - Draw and explain the transfer characteristics of CMOS inverter.
  - Explain with neat diagram two input CMOS NAND gate.
  - Explain with neat diagram two input CMOS NOR gate.
  - Define the following parameters and give typical values of these parameters w.r.t. CMOS logic family : (i) Speed of operation (ii) Power dissipation (iii) Sourcing current (iv) Fan-in (v) Sinking current.
  - Explain, why wired-logic is not used for CMOS logic circuits.
  - What do you mean by open drain output? Where is it used?
  - List differences between open drain and wired logic CMOS.
  - State merits and demerits of CMOS logic family.
  - Explain with a neat diagram interfacing of TTL gate driving CMOS gates and vice-versa.
  - Which parameters are significant while interfacing TTL and CMOS? Draw and explain TTL driving CMOS gate.
  - Compare TTL and CMOS logic families w.r.t. :
    - Power dissipation per gate
    - Propagation delay (iii) Figure of merit (iv) Fan-out.
  - List differences between CMOS and TTL.
  - Explain OR gate using CMOS logic.
- [Hints : Connect CMOS inverter at the output of CMOS NOR gate]
16. Write a note on CMOS logic family.

#### 3.5 Interfacing of CMOS to TTL and TTL to CMOS

- Interfacing means connecting the output(s) of one circuit or system to the input(s) of another circuit or system that may have different electrical characteristics. When two circuits have different electrical characteristics direct connection cannot be made. In such cases driver and load circuits are connected through interface circuit. Its function is to take the driver output signal and condition it so that it is compatible with requirements of the load.
- One must consider following important points while interfacing two circuits or systems.
  - The driver output must satisfy the voltage and current requirements of the load circuit.
  - The driver and load circuit may require different power supplies. In such cases the output of both circuit must swing between its specified voltage ranges.

### 3.5.1 TTL Driving CMOS

- Here, TTL is a driver circuit and CMOS is a load circuit. The two circuits are from different families with different electrical characteristics. Therefore, we must check that the driving device can meet the current and voltage requirements of the load device.

CMOS			TTL	
4000B	74HC/HCT	74	74LS	74AS
$I_{OH(\text{max})}$	1 $\mu\text{A}$	1 $\mu\text{A}$	40 $\mu\text{A}$	20 $\mu\text{A}$
$I_{OL(\text{max})}$	1 $\mu\text{A}$	1 $\mu\text{A}$	1.6 mA	0.4 mA
$I_{OL(\text{max})}$	0.4 mA	4 mA	0.4 mA	0.4 mA
$I_{OL(\text{max})}$	0.4 mA	4 mA	16 mA	8 mA

Table 3.5.1 Input/output currents for standard devices with supply voltage of 5 V

- Table 3.5.1 indicates that the input current values for CMOS are extremely low compared with the output current capabilities of any TTL series. Thus, TTL has no problem meeting the CMOS input current requirements.
- When we compare the TTL output voltages with the CMOS input voltage requirements we find that:

$V_{OH(\text{max})}$  for TTL  $\ll V_{IH(\text{min})}$  for CMOS for these situations TTL output must be raised to an acceptable level for CMOS. This can be done by connecting pull-up resistor at the output of TTL, as shown in the Fig. 3.5.1. The pull-up resistor causes the TTL output to rise to approximately 5 V in the HIGH state, thereby providing an adequate CMOS input voltage level.

**TTL Driving HIGH Voltage CMOS :** When output CMOS circuit is operating with  $V_{DD} > 5$  V, the situation becomes more difficult. The outputs of many TTL devices cannot be operated at more than 5 V. In such cases some alternative arrangements are made. Two of them are discussed below :

- When the TTL output cannot be pulled up to  $V_{DD}$ , one can use open collector buffer as an

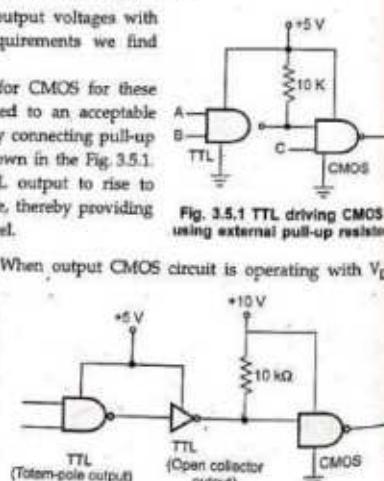


Fig. 3.5.1 TTL driving CMOS using external pull-up resistor

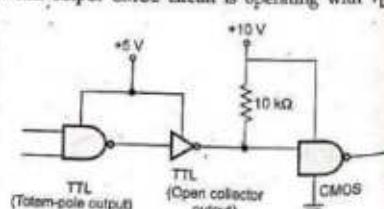


Fig. 3.5.2 Open collector buffer used as interface circuit

interface between totem-pole TTL output and CMOS operating at  $V_{DD} > 5$  V, as shown in the Fig. 3.5.2.

- The second alternative is to use level translator circuit, such as the 40104. This is a CMOS chip that is designed to take a low-voltage input (e.g. from TTL) and translate it to high voltage output for CMOS. Fig. 3.5.3 shows the circuit arrangement.

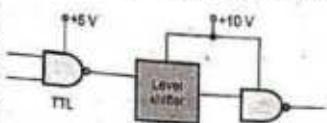


Fig. 3.5.3 Level shifter used as interface circuit

### 3.5.2 CMOS Driving TTL

- Before we consider the problem of interfacing CMOS outputs to TTL inputs, it will be helpful to review the CMOS output and TTL input characteristics for the two logic states.

For CMOS (4000B)	For TTL
$V_{OH(\text{max})}$ : 4.95 V	$V_{IH(\text{min})}$ : 2.0 V
$V_{OL(\text{max})}$ : 0.05 V	$V_{IL(\text{max})}$ : 0.8 V
$I_{OH(\text{max})}$ : 0.4 mA	$I_{IH(\text{min})}$ : 40 $\mu\text{A}$
$I_{OL(\text{max})}$ : 0.4 mA	$I_{IL(\text{max})}$ : 1.6 mA

Table 3.5.2

#### CMOS Driving TTL in the HIGH state :

- Above voltage parameters show that CMOS outputs can easily supply enough voltage ( $V_{OH}$ ) to satisfy the TTL input requirement in the HIGH state ( $V_{Ip}$ ).
- The parameters also show that CMOS outputs can supply more than enough current ( $I_{OH}$ ) to meet the TTL input current requirements ( $I_{Ip}$ ). Thus no special consideration is required for CMOS driving TTL in the HIGH state.

#### CMOS Driving TTL in LOW state :

- The parameters in the Table 3.5.2 show that CMOS output voltage ( $V_{OL}$ ) satisfies TTL input requirement in the LOW state ( $V_{IL}$ ). However, the current requirements in the LOW state are not satisfied.
- The TTL input has a relatively high input current in the LOW state (1.6 mA) and CMOS output current at LOW state ( $I_{OL}$ ) is not sufficient to drive even one input of the TTL. In such situations some type of

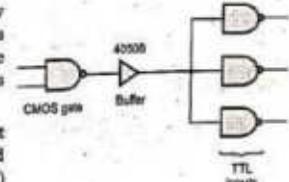


Fig. 3.5.4 CMOS driving TTL in LOW state using buffer

interface circuit is needed between the CMOS and TTL devices. Fig. 3.5.4 shows the possible interface circuit.

3. In Fig. 3.5.4 the CMOS 4050B, non-inverting buffer is used as an interfacing circuit. It has an output current rating of  $I_{OL(\text{max})} = 3 \text{ mA}$  which satisfies the TTL input current requirement.

#### HIGH Voltage CMOS Driving TTL :

- Some IC manufacturers have provided several 74LS TTL devices that can withstand input voltages as high as 15 V. These devices can be driven directly from CMOS outputs operating at  $V_{DD} = 15 \text{ V}$ .

- Most TTL inputs cannot handle more than 7 V and so interface is necessary if they are to be driven from high-voltage CMOS. In such situations voltage level translators are used. They convert the high-voltage input to a 5 V output that can be connected to TTL.

- Fig. 3.5.5 shows how the 4050B can be used to perform this level translation between 15 V and 5 V.

#### Example 3.5.1 Why is a buffer required between some CMOS outputs and TTL inputs ?

**Solution :** The current requirement of CMOS output at LOW state are not satisfied by TTL input. The TTL input has a relatively high input current in the LOW state (1.6 mA) and CMOS output current at LOW state ( $I_{OL}$ ) is not sufficient to drive even one input of the TTL. Hence buffer is used for interfacing.

#### Example 3.5.2 Which CMOS series can have its inputs driven directly from a TTL output ? Justify your answer.

**Solution :** VHC and VHCT CMOS families are TTL compatible.

#### Review Question

- Explain interfacing of TTL and CMOS logic families.

### 3.6 Comparison between TTL and CMOS

GTU : Winter-15

Sr. No.	Parameter	CMOS	TTL
1.	Device used	n-channel and P-channel MOSFET	Bipolar junction transistor
2.	$V_{IH(\text{min})}$	3.5 V	2 V
3.	$V_{IL(\text{max})}$	1.5 V	0.8 V
4.	$V_{OH(\text{min})}$	4.95 V	2.7 V
5.	$V_{OL(\text{max})}$	0.05 V	0.4 V
6.	High level noise margin	$V_{NH} \approx 1.45 \text{ V}$	0.4 V
7.	Low level noise margin	$V_{NL} \approx 1.45 \text{ V}$	0.4 V
8.	Noise immunity	Better than TTL	Less than CMOS
9.	Propagation delay	70 ns	10 ns
10.	Switching speed	Less than TTL	Faster than CMOS
11.	Power dissipation per gate	0.1 mW	10 mW
12.	Speed power product	0.7 pJ	100 pJ
13.	Fan-out	50	10
14.	Power supply voltage	3 - 15 V	Fixed 5 V
15.	Power dissipation	Increase with frequency	Increase with frequency
16.	Application	Portable instrument where battery supply is used.	Laboratory instruments

Table 3.6.1 Comparison between TTL and CMOS families

#### Review Question

- Compare TTL and CMOS logic families.

GTU : Winter-15, Summer-18, Marks 7

**Oral Questions and Answers****Q.1** What are the types of TTL logic ?

Ans. : 1. Open collector output 2. Totem-pole output 3. Tri-state output.

**Q.2** List the different versions of TTL.Ans. : 1. TTL (Std.TTL) 2. LTTL (Low Power TTL) 3. HTTL (High Speed TTL)  
4. STTTL (Schottky TTL) 5. LSTTL (Low power Schottky TTL).**Q.3** Distinguish between 7400 series and 5400 series.

Ans. :

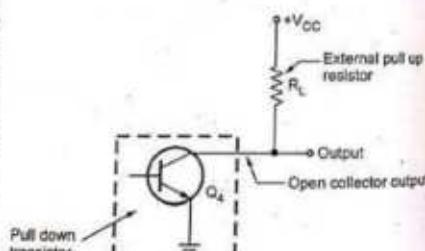
Supply voltage	For 74 series - (4.75 to 5.25) V For 54 series - (4.5 to 5.5) V
Temperature range	For 74 series - (0 °C to 70 °C) For 54 series - (-55 °C to 125 °C)

**Q.4** What is meant by tristate capability ?

Ans. : Tristate capability is the capability of output level to take three stages (output levels) as HIGH, LOW and High-impedance. For example, tristate TTL.

**Q.5** Explain open collector output.

Ans. : When the collector terminal of a transistor is kept open without any pull up transistor the arrangement is called open collector output. The output is taken directly from the open collector terminal of a transistor at the output. But, a gate with open collector will not work properly until an external resistor is connected.



When  $Q_4$  is ON, output is low and when  $Q_4$  is OFF, output is tied to  $V_{CC}$  through an external pull up resistor.

Fig. 3.1 Open collector output with pull-up resistor

**Q.6** Differentiate : Source and sink current.

Ans. :

Sr. No.	Current sinking	Current sourcing
1.	When the output of the logic circuit is low, the current flows from load into the pull down transistor of the totem-pole. This is known as current sinking.	When the output of the logic circuit is high, the current flows from pull up transistor of the totem-pole to the load. This is known as current sourcing.
2.	During current sinking base-emitter diode of the load is forward biased. Hence sinking current is in milliamperes.	During current sourcing only leakage current flows hence sourcing current is in microamperes.

**Q.7** How Schottky transistors are formed and state its use ?

Ans. : A Schottky diode is formed by the combination of metal and semiconductor. The presence of schottky diode between the base and the collector prevents the transistor from going into saturation. The resulting transistor is called Schottky transistor.

The use of Schottky transistor in TTL decreases the propagation delay without a sacrifice of power dissipation.

**Q.8** Why totem pole outputs cannot be connected together ?

Ans. : Totem pole outputs cannot be connected together because such a connection might produce excessive current and may result in damage to the devices.

**Q.9** What happens to output when a tristate circuit is selected for high impedance.

Ans. : Output is disconnected from rest of the circuits by internal circuitry.

**Q.10** State advantages and disadvantages of totem-pole output.

Ans. : Advantages of totem-pole output are :

1. External pull up resistor is not required.
2. Operating speed is high.

Disadvantages of totem-pole output :

1. Output of two gates cannot be tied together.

**Q.11** Explain the wired-AND connection.

Ans. : When the open collector outputs of two or more gates can be connected together, the connection is called a wired AND. In wired-AND connection, the output is high only when all switches are open i.e. when output of each stage is high.

Hence, the output is equivalent to the logical AND operation of the logic function performed by the gates.

**Q.12** Give any two applications of open collector logic.

**Ans. :** 1. It is used when we require to connect outputs of two or more gates.  
2. It is used when sourcing current requirements is larger than the standard totem-pole output.

**Q.13** State advantages and disadvantages of TTL.

**Ans. :** Refer section 3.3.8.

**Q.14** What is the effect of increasing the supply voltage on the propagation delay of the CMOS gates?

**Ans. :** Increasing the supply voltage reduce the propagation delay of the CMOS gates.

**Q.15** Which is the fastest logic family?

**Ans. :** ECL.

**Q.16** Which TTL logic gate is used for wired ANDing?

**Ans. :** The output of open collector TTL logic gates can be tied together using wired AND technique.



# 4

## Combinational Digital Circuits

### Contents

4.1	Standard Representation for Logic Functions	May-12, 13, 14, Dec.-13, Winter-17, Marks 7
4.2	Karnaugh-Maps (K-Map) Representation	May-14, Marks 7
4.3	Simplification SOP Functions using K-Maps	May-12, 13, Dec.-12, 13, Summer-17, 18, Marks 7
4.4	Simplification of POS Functions using K-Maps	Winter-17, Marks 7
4.5	Summary of Rules for K-Map Simplification	Marks 7
4.6	Limitation of Karnaugh Map	
4.7	Realizing Logic Function with Gates	Dec.-11, 12, 13, May-12, 13, Summer-15, 16, 18, Winter-14, 15, 18, Marks 10
4.8	Combinational Design Examples	Winter-14, 18, Marks 7
4.9	Quine McCluskey Method of Function Realization	
4.10	Multiplexers	Dec.-13, Winter-14, 15, 18, Summer-15, 17, 18, Marks 7
4.11	Demultiplexers	
4.12	Decoders	May-12, 14, Dec-13, Winter-15, 18, Summer-15, 18, Marks 7
4.13	Encoders	
4.14	Adders	May-13, Dec-13, Marks 7
4.15	Subtractors	
4.16	BCD Arithmetic	
4.17	Carry Look Ahead Adder	Summer-15, Marks 7

4.18 Digital Comparator	Dec.-13, Summer-15, 18,	Marks 7
4.19 Parity Generator/Checker	Winter-15, 16,	Marks 7
4.20 Code Converters	May-14, Summer-15, 16,	Marks 7
4.21 ALU and Elementary ALU Design	May-12, 13,	Marks 7

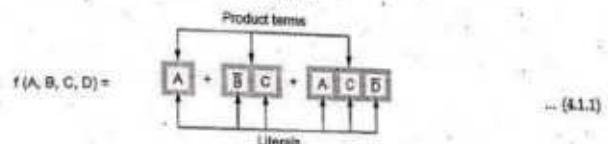
## Oral Questions and Answers

## 4.1 Standard Representation for Logic Functions

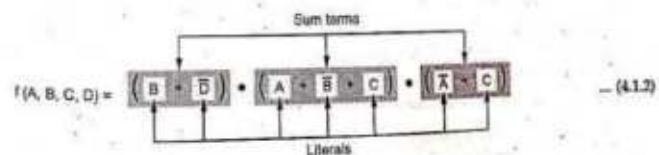
GTU : May-12, 13, 14, Dec-13, Winter-17

- Boolean expressions are constructed by connecting the Boolean constants and variables with the Boolean operations. These Boolean expressions are also known as Boolean formulae.
- We use Boolean expressions to describe switching function or Boolean functions. For example, if the Boolean expression  $(A + \bar{B})C$  is used to describe the function  $f$ , then Boolean function is written as  

$$f(A, B, C) = (A + \bar{B})C \quad \text{or} \quad f = (A + \bar{B})C$$
- Based on the structure of Boolean expression, it can be categorized in different formulae. One such categorization are the normal formulae.
- Consider the four-variable Boolean function.



- In this Boolean function the variables are appeared either in a complemented or an uncomplemented form.
- Each occurrence of a variable in either a complemented or an uncomplemented form is called a literal. Thus, the above Boolean function 4.1.1 consists of six literals. They appear in the product terms.
- A product term is defined as either a literal or a product (also called conjunction) of literals. Function 4.1.1 contains three product terms, namely,  $A$ ,  $\bar{B}'C$  and  $A\bar{C}'D$ .
- Consider another four variable Boolean function.

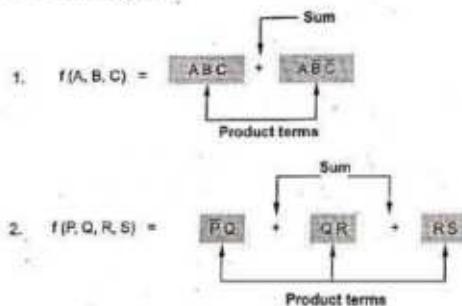


- The above Boolean function consists of seven literals. Here, they appear in the sum terms.

- A sum term is defined as either a literal or a sum (also called disjunction) of literals. Function 4.1.2 contains three sum terms, namely,  $(B + \bar{D})$ ,  $(A + \bar{B} + C)$  and  $(\bar{A} + C)$ . These literals and terms are arranged in one of the two forms :
  - Sum Of product form (SOP) and
  - Product Of Sum form (POS).

#### 4.1.1 Sum of Product Form

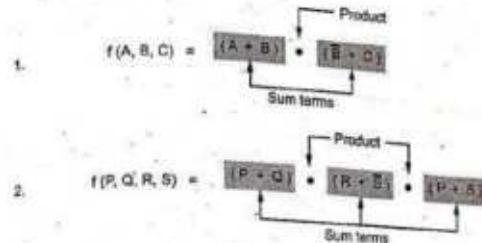
- The words sum and product are derived from the symbolic representations of the OR and AND functions by  $+$  and  $\cdot$  (addition and multiplication), respectively. But we realize that these are not arithmetic operators in the usual sense. A product term is any group of literals that are ANDed together.
- For example, ABC, XY and so on. A sum term is any group of literals that are ORed together such as  $A + B + C$ ,  $X + Y$  and so on. A Sum Of Products (SOP) is a group of product terms ORed together. Some examples of this form are :
- Each of these sum of products expressions consist of two or more product terms (AND) that are ORed together.



- Each product term consists of one or more literals appearing in either complemented or uncomplemented form. For example, in the sum of products expression  $ABC + A\bar{B}\bar{C}$ , the first product term contains literals A, B and C in their uncomplemented form. The second product term contains B and C in their complemented (inverted) form.
- The sum of product form is also known as disjunctive normal form or disjunctive normal formula.

#### 4.1.2 Product of Sum Form

- A product of sums is any groups of sum terms ANDed together. Some examples of this form are :



- Each of these product of sums expressions consist of two or more sum terms (OR) that are ANDed together.
- Each sum term consists of one or more literals appearing in either complemented or uncomplemented form.
- The product of sum form is also known as conjunctive normal form or conjunctive normal formula.

#### 4.1.3 Standard SOP and Standard POS Forms

- We can realize that in the SOP form, all the individual terms do not involve all literals.
- For example, in expression  $AB + ABC$  the first product term do not contain literal C.
- If each term in SOP form contains all the literals then the SOP form is known as standard or canonical SOP form.
- Each individual term in the standard SOP form is called minterm.
- One standard sum of products expression is as shown in Fig. 4.1.1.
- If each term in POS form contains all the literals then the POS form is known as standard or canonical POS form.

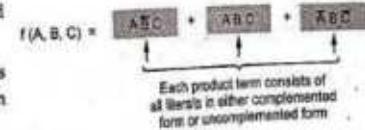


Fig. 4.1.1 Standard SOP form

- Each individual term in the standard POS form is called maxterm.
- One standard product of sums expression is as shown in Fig. 4.1.2.

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

Each sum term consists of all literals in either complemented form or uncomplemented form

Fig. 4.1.2 Standard POS form

**4.1.4 Converting Expressions in Standard SOP or POS Form**

- Sum of product form can be converted to standard sum of products by ANDing the terms in the expression with terms formed by ORing the literal and its complement which are not present in that term.
- For example for a three literal expression with literals A, B and C, if there is a term AB, where C is missing, then we form term ( $C + \bar{C}$ ) and AND it with AB. Therefore, we get  $AB(C + \bar{C}) = ABC + A\bar{B}C$ .

**4.1.4.1 Steps to Convert SOP to Standard POS Form**

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

**Step 2 :** AND each product term having missing literal/s with term/s form by ORing the literal and its complement.

**Step 3 :** Expand the terms by applying distributive law and reorder the literals in the product terms.

**Step 4 :** Reduce the expression by omitting repeated product terms if any. Because  $A + A = A$ .

**Illustrative Examples**

**Example 4.1.1** Convert the given expression in standard SOP form.

$$f(A, B, C) = AC + AB + BC$$

Solution :

**Step 1 :** Find the missing literal/s in each product term.

$$f(A, B, C) = \boxed{AC} + \boxed{AB} + \boxed{BC}$$

Literal A is missing.  
Literal C is missing.  
Literal B is missing.

TECHNICAL PUBLICATIONS™ An up beat for knowledge

**Step 2 : AND product term with (missing literal + its complement).**

$$f(A, B, C) = \boxed{AC} \cdot (B + \bar{B}) + \boxed{AB} \cdot (C + \bar{C}) + \boxed{BC} \cdot (A + \bar{A})$$

Original product terms  
Missing terms and their complements

**Step 3 : Expand the terms and reorder literals.**

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

**Note** After having sufficient practice student should expand product term and reorder literals in it in a single step.

**Step 4 : Omit repeated product terms.**

$$\begin{aligned} f(A, B, C) &= ABC + A\bar{B}C + \boxed{ABC} + A\bar{B}C + \boxed{ABC} + \bar{A}BC \\ f(A, B, C) &= ABC + A\bar{B}C + A\bar{B}C + \bar{A}BC \end{aligned}$$

**Example 4.1.2** Convert the given expression in standard SOP form.  $f(A, B, C) = A + ABC$

Solution :

**Step 1 : Find the missing literal/s in each product term.**

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

Literal B and C are missing

**Step 2 : AND product term with (missing literal + its complement).**

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

Original terms  
Missing literals and their complements

**Step 3 : Expand the terms and reorder literals.**

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

**Step 4 : Omit repeated product term**

$$\begin{aligned} f(A, B, C) &= ABC + A\bar{B}C + \bar{A}BC + \bar{A}\bar{B}C \\ f(A, B, C) &= ABC + A\bar{B}C + A\bar{B}C + \bar{A}BC \end{aligned}$$

Repeated product term

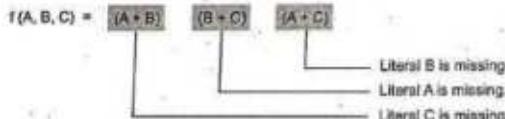
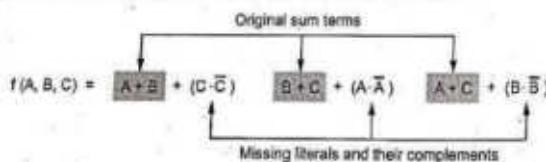
TECHNICAL PUBLICATIONS™ An up beat for knowledge

**Examples for Practice****Example 4.1.3 :** Express the following function in standard SOP form

$$F_1 = AB + \bar{C}D + A\bar{B}C$$

[Ans. :  $\bar{A}\bar{B}CD + \bar{A}\bar{B}\bar{C}D + A\bar{B}CD + A\bar{B}\bar{C}D + A\bar{B}CD + A\bar{B}C\bar{D} + AB\bar{C}\bar{D} + AB\bar{C}D + ABC\bar{D} + ABCD$ ]**Example 4.1.4 :** Determine the canonical SOP form of  $f(x, y, z) = (x\bar{y}z + \bar{x}\bar{y}z)(\bar{y} + x\bar{z})$ [Ans. :  $x\bar{y}z + x\bar{y}z + \bar{x}yz + x\bar{z}y$ ]**4.1.4.2 Steps to Convert POS to Standard POS Form****Step 1 :** Find the missing literals in each sum term if any.**Step 2 :** OR each sum term having missing literal/s with term/s from by ANDing the literal and its complement.**Step 3 :** Expand the terms by applying distributive law and reorder the literals in the sum terms.**Step 4 :** Reduce the expression by omitting repeated sum terms if any. Because  $A \cdot A = A$ .**Illustrative Examples****Example 4.1.5** Convert the given expression in standard POS form.

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

**Solution :****Step 1 :** Find the missing literal/s in each sum term.**Step 2 :** OR sum term with (missing literal + its complement).**Step 3 :** Expand the terms and reorder literals.**Expand :**Since  $A + BC = (A + B)(A + C)$  we have,

$$\begin{aligned} f(A, B, C) &= (A + B + C)(A + B + \bar{C})(B + C + A)(B + C + \bar{A}) \\ &\quad (A + C + B)(A + C + \bar{B}) \end{aligned}$$

**Reorder :**

$$\begin{aligned} f(A, B, C) &= (A + B + C)(A + B + \bar{C})(A + B + C)(\bar{A} + \bar{B} + C) \\ &\quad (A + B + C)(A + \bar{B} + C) \end{aligned}$$

**Step 4 :** Omit repeated sum terms.

Repeated sum terms

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

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

**Example 4.1.6** Convert the given expression in standard POS form.

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

**Solution :****Step 1 :** Find the missing literal/s in each sum term.

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

↑ Literals B and C are missing

**Step 2 :** OR sum term with (missing literal + its complement).

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

**Step 3 :** Expand the terms and reorder literals.Since  $A + BC = (A + B)(A + C)$  we have,

$$\begin{aligned} f(A, B, C) &= (A + B \cdot \bar{B} + C)(A + B \cdot \bar{B} + \bar{C})(A + B + C) \\ &= (A + B + C)(A + \bar{B} + C)(A + B + \bar{C})(A + \bar{B} + \bar{C})(A + B + C) \end{aligned}$$

**Step 4 : Omit repeated sum terms.**

$$\begin{aligned} f(A, B, C) &= (A + B + C)(A + \bar{B} + C)(A + B + \bar{C})(A + \bar{B} + \bar{C}) \\ f(A, B, C) &= (A + B + C)(A + \bar{B} + C)(A + B + \bar{C})(A + \bar{B} + \bar{C}) \end{aligned}$$

Repeated sum term

#### Example for Practice

**Example 4.1.7 :** Convert the given expression in standard POS form  
 $f(P, Q, R) = (P + \bar{Q})(P + R)$

$$(\text{Ans. : } (P + \bar{Q} + R)(P + \bar{Q} + \bar{R})(P + Q + R))$$

#### 4.15 M Notations : Minterms and Maxterms

- Each individual term in standard SOP form is called minterm and each individual term in standard POS form is called maxterm.
- The concept of minterms and maxterms allows us to introduce a very convenient shorthand notations to express logical functions.
- Table 4.1.1 gives the minterms and maxterms for a three literal/variable logical function where the number of minterms as well as maxterms is  $2^3 = 8$ . In general, for an n-variable logical function there are  $2^n$  minterms and an equal number of maxterms.
- As shown in Table 4.1.1 each minterm is represented by  $m_i$  and each maxterm is represented by  $M_i$ , where the subscript  $i$  is the decimal number equivalent of the natural binary number. With these shorthand notations logical function can be represented as follows :

$$\begin{aligned} 1. \quad f(A, B, C) &= \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} \\ &= m_0 + m_1 + m_3 + m_6 \\ &= \sum m(0, 1, 3, 6) \end{aligned}$$

TECHNICAL PUBLICATIONS™ - An up beat for knowledge

Variables	Minterms	Maxterms
A B C	$m_0$	$M_0$
0 0 0	$\bar{A}\bar{B}\bar{C} = m_0$	$A + B + C = M_0$
0 0 1	$\bar{A}\bar{B}C = m_1$	$A + B + \bar{C} = M_1$
0 1 0	$\bar{A}B\bar{C} = m_2$	$A + \bar{B} + C = M_2$
0 1 1	$\bar{A}B{C} = m_3$	$A + \bar{B} + \bar{C} = M_3$
1 0 0	$A\bar{B}\bar{C} = m_4$	$\bar{A} + B + C = M_4$
1 0 1	$A\bar{B}C = m_5$	$\bar{A} + B + \bar{C} = M_5$
1 1 0	$AB\bar{C} = m_6$	$\bar{A} + \bar{B} + C = M_6$
1 1 1	$ABC = m_7$	$\bar{A} + \bar{B} + \bar{C} = M_7$

Table 4.1.1 Minterms and maxterms for three variables

$$\begin{aligned} 2. \quad f(A, B, C) &= (A + B + \bar{C})(A + \bar{B} + \bar{C})(\bar{A} + \bar{B} + C) \\ &= M_1 + M_3 + M_6 \\ &= \prod M(1, 3, 6) \end{aligned}$$

where  $\Sigma$  denotes sum of product while  $\Pi$  denotes product of sum.

- Logical expression can be represented in the truth table form.
- It is possible to write logic expression in standard SOP or POS form corresponding to a given truth table.
- The logic expression corresponding to a given truth table can be written in a standard sum of products form by writing one product term for each input combination that produces an output of 1.
- These product terms are ORed together to create the standard sum of products. The product terms are expressed by writing complement of a variable when it appears as an input 0, and the variable itself when it appears as an input 1.
- Consider, for example, the truth Table 4.1.2.
- The product corresponding to input combination 010 is  $\bar{A}BC$ , the product corresponding to input combination 011 is  $\bar{A}BC$  and product corresponding to input combination 110 is  $ABC$ . Thus the standard sum of products form is

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

$$= m_2 + m_3 + m_6$$

- The logic expression corresponding to a truth table can also be written in a standard product of sums form by writing one sum term for each output 0.

- The sum terms are expressed by writing complement of a variable when it appears as an input 1 and the variable itself when it appears as an input 0.

- Consider, for example, the truth Table 4.1.3.

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

$\leftarrow \bar{A}BC$   
 $\leftarrow \bar{A}BC$   
 $\leftarrow ABC$

Table 4.1.2

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

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

TECHNICAL PUBLICATIONS™ - An up beat for knowledge

TECHNICAL PUBLICATIONS™ - An up beat for knowledge

- The sum corresponding to input combinations 010 is  $A + \bar{B} + C$ , and the sum corresponding to input 101 is  $\bar{A} + B + \bar{C}$ . Thus, the standard product of sum form is
- $$f(A, B, C) = (A + \bar{B} + C)(\bar{A} + B + \bar{C}) \\ = M_2 \cdot M_5$$

#### 4.1.6 Complements of Standard Forms

- The POS and SOP functions derived from the same truth table are logically equivalent.
  - In terms of minterms and maxterms we can then write
- $$f(A, B, C) = m_0 + m_1 + m_3 + m_4 + m_6 + m_7 = M_2 + M_5$$
- $\therefore f(A, B, C) = \sum m(0, 1, 3, 4, 6, 7) = \prod M(2, 5)$

From the above expressions we can easily notice that there is a complementary type of relationship between a function expressed in terms of minterms. Using the complementary relationship we can find logical function in terms of maxterms if function in minterms is known or vice-versa. For example, for a four variables if

$$f(A, B, C, D) = \sum m(0, 2, 4, 6, 8, 10, 12, 14)$$

then  $f(A, B, C, D) = \pi M(1, 3, 5, 7, 9, 11, 13, 15)$

#### Example 4.1.8 Express $F = A + B'C$ as sum of minterms.

Solution :

$$\begin{aligned} A + \bar{B}C &= A(B + \bar{B})(C + \bar{C}) + (A + \bar{A})\bar{B}C \\ &= (AB + A\bar{B})(C + \bar{C}) + (A\bar{B}C) + \bar{A}\bar{B}C \\ &= A BC + A \bar{B}C + A \bar{B}\bar{C} + A \bar{B}C + \bar{A}\bar{B}C \\ &= A BC + A \bar{B}C + A \bar{B}\bar{C} + \bar{A}\bar{B}C \\ \therefore F &= \sum m(1, 4, 5, 6, 7) \end{aligned}$$

#### Example 4.1.9 Express the Boolean function $F = XY + \bar{X}Z$ in product of maxterm.

Solution :  $F = XY + \bar{X}Z$

$$= XY(Z + \bar{Z}) + \bar{X}Z(Y + \bar{Y}) = XYZ + XY\bar{Z} + \bar{X}YZ + \bar{X}\bar{Y}Z$$

$$F = \sum m(7, 6, 3, 1)$$

$$= \Pi M(0, 2, 4, 5)$$

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

#### Example 4.1.10 Express the Boolean function as

- POS form
  - SOP form
- $$D = (A' + B)(B' + C)$$

Solution :

$$\begin{aligned} 1. \text{ POS form} : D &= (\bar{A} + B)(\bar{B} + C) \\ 2. \text{ SOP form} : D &= \bar{A}\bar{B} + \bar{A}C + B\bar{B} + BC \\ &= \bar{A}\bar{B} + \bar{A}C + BC \quad \because B\bar{B} = 0 \end{aligned}$$

#### Example 4.1.11 Express the following function in product of maxterms.

$$F(x, y, z) = (xy + z)(y + xz)$$

Solution :  $F(x, y, z) = (xy + z)(y + xz)$

$$\begin{aligned} &= xy + xyz + yz + xz \\ &= xy(1 + z) + yz + xz \\ &= xy + yz + xz \\ &= xy(z + \bar{z}) + yz(x + \bar{x}) + xz(y + \bar{y}) \\ &= xyz + xy\bar{z} + xyz + \bar{y}yz + xyz + x\bar{y}z \end{aligned}$$

$$F(x, y, z) = xyz + xy\bar{z} + \bar{y}yz + x\bar{y}z$$

$$\therefore F(x, y, z) = \sum m(3, 5, 6, 7) = \prod M(0, 1, 2, 4)$$

$$\therefore F(x, y, z) = (x + y + z)(x + y + \bar{z})(x + \bar{y} + z)(\bar{x} + y + z)$$

#### Example 4.1.12 Obtain the truth table of the function $F = xy + x\bar{y} + yz$ .

Solution :  $F = xy + x\bar{y} + yz$

$$\begin{aligned} &= xy(z + \bar{z}) + x\bar{y}(z + \bar{z}) + \bar{y}z(x + \bar{x}) \\ &= xyz + xy\bar{z} + x\bar{y}z + x\bar{y}z + \bar{y}xz \\ F &= xyz + xy\bar{z} + x\bar{y}z + x\bar{y}z + \bar{y}xz \end{aligned}$$

x	y	z	o/p
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0

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

Truth table of the function F

**Example 4.1.13** Attempt following : 1) Convert into sum-of-minterms :  $A + B + CA$ 2) Convert into product-of-maxterms :  $A(\bar{A} + B)\bar{C}$ 

GTU : May-14, Marks 7

**Solution :** 1)  $\bar{A} + B + CA$ 

$$\begin{aligned}
 &= \bar{A} \cdot (\bar{B} + \bar{B}) \cdot (\bar{C} + \bar{C}) + B \cdot (\bar{A} + \bar{A}) \cdot (\bar{C} + \bar{C}) + CA(\bar{B} + \bar{B}) \\
 &= \bar{A} \cdot (\bar{B}C + \bar{B}\bar{C} + \bar{B}\bar{C} + \bar{B}\bar{C}) + B \cdot (\bar{A}C + \bar{A}\bar{C} + \bar{A}\bar{C} + \bar{A}\bar{C}) + ABC + A\bar{B}\bar{C} \\
 &= \bar{A}\bar{B}C + \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}\bar{C} + ABC + A\bar{B}\bar{C} + \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}\bar{C} + ABC + A\bar{B}\bar{C} \\
 &= \bar{A}\bar{B}C + \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}\bar{C} + ABC + A\bar{B}\bar{C} + \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}\bar{C}
 \end{aligned}$$

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

$$\begin{aligned}
 &= (A + B \cdot \bar{B} + C \cdot \bar{C})(\bar{A} + B + C \cdot \bar{C})(A \cdot \bar{A} + B \cdot \bar{B} + \bar{C}) \\
 &= (A + B \cdot \bar{B} + C)(A + B \cdot \bar{B} + \bar{C})(\bar{A} + B + C)(\bar{A} + B + \bar{C}) \\
 &\quad (A + B \cdot \bar{B} + \bar{C})(\bar{A} + B + \bar{B} + \bar{C}) \\
 &= (A + B \cdot \bar{B} + C)(A + B \cdot \bar{B} + \bar{C})(\bar{A} + B + C)(\bar{A} + B + \bar{C})(\bar{A} + B + \bar{B} + \bar{C}) \\
 &\quad \because A \cdot \bar{A} = 0 \\
 &= (A + B + C)(A + \bar{B} + C)(A + B + \bar{C})(A + \bar{B} + \bar{C})(\bar{A} + B + C) \\
 &\quad (\bar{A} + B + \bar{C})(\bar{A} + B + \bar{C})(\bar{A} + \bar{B} + \bar{C})
 \end{aligned}$$

**Example 4.1.14** Express the Boolean function  $F = AB + AC$  in a product of maxterm.

GTU : Winter-17, Marks 4

**Solution :**  $F = AB(C + \bar{C}) + \bar{A}C(B + \bar{B})$ 

$$\begin{aligned}
 &= ABC + AB\bar{C} + \bar{A}BC + \bar{A}\bar{B}C \\
 &= \sum m(7, 6, 3, 1) \\
 &= \sum m(1, 3, 6, 7) \\
 &= \pi M(0, 2, 4, 5) \\
 &= (A + B + C)(A + \bar{B} + C)(\bar{A} + B + C)(\bar{A} + B + \bar{C})
 \end{aligned}$$

TECHNICAL PUBLICATIONS™ - An up thrust for knowledge

## Review Questions

1. Define the following terms : 1) Minterm 2) Maxterm.  
 2. Define switching function.  
 3. Define literal, product term and sum term.  
 4. Explain sum of product form.  
 5. What do you mean by SOP and POS form also convert the expression  $y = (A + BC)(B + \bar{AC})$  into SOP and POS form.  
 6. Explain how to convert SOP or POS expressions in their standard forms.  
 7. What do you mean by minterms and maxterms ?

GTU : Dec-13, Marks 2

## 4.2 Karnaugh-Maps (K-Map) Representation

GTU : May-14

- During the process of simplification of Boolean expression we have to predict each successive step.
- We can never be absolutely certain that an expression simplified by Boolean algebra alone is the simplest possible expression.
- On the other hand, the map method gives us a systematic approach for simplifying a Boolean expression.
- The map method, first proposed by Veitch and modified by Karnaugh, hence it is known as the Veitch diagram or the Karnaugh map.

## 4.2.1 One-Variable, Two-Variable, Three-Variable and Four-Variable Maps

- The basis of this method is a graphical chart known as Karnaugh map (K-map).
- It contains boxes called cells.
- Each of the cell represents one of the  $2^n$  possible products that can be formed from  $n$  variables. Thus, a 2-variable map contains  $2^2 = 4$  cells, a 3-variable map contains  $2^3 = 8$  cells and so forth.
- Fig. 4.2.1 shows outlines of 1, 2, 3 and 4-variable maps.

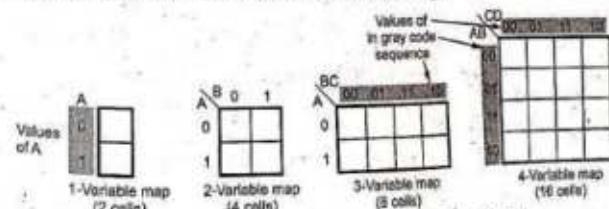


Fig. 4.2.1 Outlines of 1, 2, 3 and 4-variable Karnaugh maps

TECHNICAL PUBLICATIONS™ - An up thrust for knowledge

- Product terms are assigned to the cells of a Karnaugh map by labeling each row and each column of the map with a variable, with its complement, or with a combination of variables and complements.
- The product term corresponding to a given cell is then the product of all variables in the row and column where the cell is located.
- Fig. 4.2.2 shows the way to label the rows and columns of a 1, 2, 3 and 4-variable maps and the product terms corresponding to each cell.

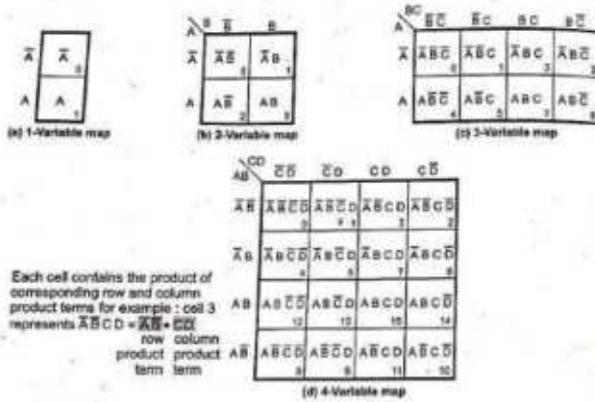


Fig. 4.2.2 1, 2, 3 and 4-variable maps with product terms

- When we move from one cell to the next along any row or from one cell to the next along any column, one and only one variable in the product term changes (to a complemented or to an uncomplemented form).
- For example, in Fig. 4.2.2 (b) the only change that occurs in moving along the bottom row from  $\bar{A}B$  to  $AB$  is the change from  $\bar{B}$  to  $B$ . Similarly, the only change that occurs in moving down the right column from  $\bar{A}B$  to  $AB$  is the change from  $A$  to  $\bar{A}$ . Irrespective of number of variables the labels along each row and column must conform to the single-change rule.
- The gray code has same properties (only one variable change when we proceed to next number or previous number) hence gray code is used to label the rows and columns of K-map as shown in Fig. 4.2.3.

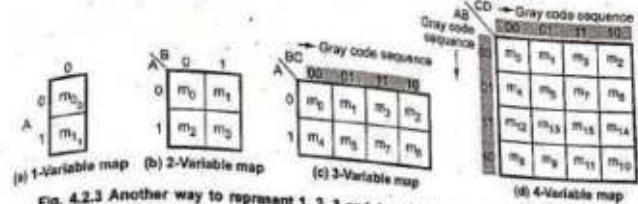


Fig. 4.2.3 Another way to represent 1, 2, 3 and 4-variable maps for SOP expressions

- The Fig. 4.2.3 shows label of the rows and columns of a 1, 2, 3 and 4-variable maps using gray code and the product terms corresponding to each cell.
- Instead of writing actual product terms, corresponding shorthand minterm notations are written in the cell and row and columns are marked with gray code instead of variables.
- In case of POS expressions we assign maxterms (sum terms) to the cells of a Karnaugh map.
- The Fig. 4.2.4 shows the way to label the rows and columns of a 1, 2, 3 and 4-variable maps using gray code and the sum terms corresponding to each cell.

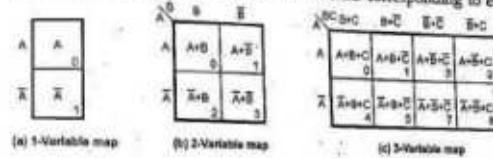


Fig. 4.2.4 1, 2, 3 and 4-variable maps with sum terms

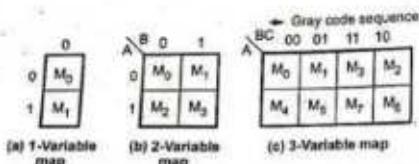


Fig. 4.2.5 Another way to represent 1, 2, 3 and 4-variable map for POS expressions

- Instead of writing actual sum terms, corresponding shorthand maxterm notations are written in the cell and row and columns are marked with gray code instead of variables.

#### 4.2.2 Plotting a Karnaugh Map

- Logic function can be represented in various forms such as truth table, SOP Boolean expression and POS Boolean expression. In this section we will see the procedures to plot the given logic function in any form on the Karnaugh map.

##### 4.2.2.1 Representation of Truth Table on Karnaugh Map

**Cell :** The smallest unit of a Karnaugh map, corresponding to one line of a truth table. The input variables are the cell's co-ordinates and the output variable is the cell's contents.

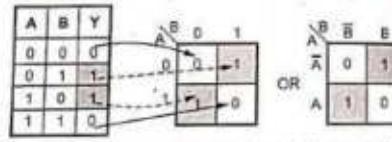


Fig. 4.2.6 (a) Representation of 2-variable truth table on K-map

- Fig. 4.2.6 shows K-maps plotted from truth tables with 2, 3 and 4-variables.

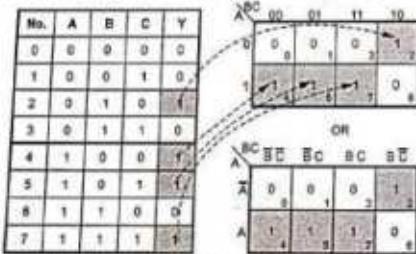


Fig. 4.2.6 (b) Representation of 3-variable truth table on K-map.

- The terms which are having output 1, have the corresponding cells marked with 1s. The other cells are marked with zeros.

No.	A	B	C	D	Y
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	0
11	1	0	1	1	0
12	1	1	0	0	0
13	1	1	0	1	0
14	1	1	1	0	1
15	1	1	1	1	1

(a) Representation of 4-variable truth table on K-map

Fig. 4.2.6 Plotting truth table on K-map

Note The student can verify the data in each cell by checking the data in the column Y for particular row number and the data in the same cell number in the K-map.

#### 4.2.2.2 Representing Standard SOP on K-Map

- A Boolean expression in the sum of products form can be plotted on the Karnaugh map by placing a 1 in each cell corresponding to a term (minterm) in the sum of products expression. Remaining cells are filled with zeros. This is illustrated in the following examples.

#### Illustrative Examples

##### Example 4.2.1 Plot Boolean expression $Y = ABC + ABC + \bar{A}BC$ on the Karnaugh map.

Solution : The expression has 3-variables and hence it can be plotted using 3-variable as in Fig. 4.2.7.

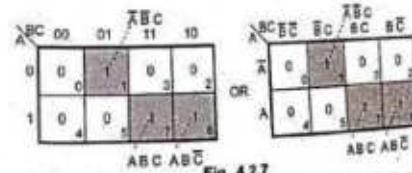


Fig. 4.2.7

**Example 4.2.2** Plot Boolean expression

$$Y = ABC\bar{D} + A\bar{B}CD + \bar{A}BCD + A\bar{E}CD + A\bar{B}\bar{C}D$$

Solution : The expression has 4-variables and hence it can be plotted using 4-variable map as shown in Fig. 4.2.8.

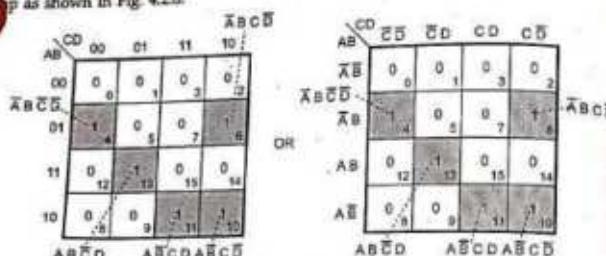


Fig. 4.2.8

**4.2.3 Representing Standard POS on K-Map**

- A Boolean expression in the product of sums can be plotted on the Karnaugh map by placing a 0 in each cell corresponding to a term (maxterm) in the expression. Remaining cells are filled with ones. This is illustrated in the following example.

**Illustrative Examples****Example 4.2.3** Plot Boolean expression  $Y = (A + \bar{B} + C)(A + \bar{B} + \bar{C})(A + \bar{B} + \bar{C})$  ( $\bar{A} + B + C$ ) on the Karnaugh map.

Solution : The expression has 3-variables and hence it can be plotted using 3-variable map as shown in Fig. 4.2.9.

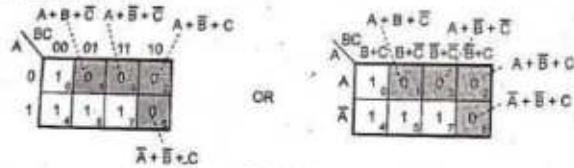


Fig. 4.2.9

$$(\bar{A} + \bar{B} + C) = M_2, (\bar{A} + \bar{B} + \bar{C}) = M_3, (\bar{A} + \bar{B} + C) = M_4, (\bar{A} + B + \bar{C}) = M_1$$

**Example 4.2.4** Plot Boolean expression:

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

Solution : The expression has 4-variables and hence it can be plotted using 4-variable map as shown in Fig. 4.2.10.

AB	CD	00	01	11	10
00	00	0	0	0	0
01	01	1	1	1	0
11	10	1	0	1	0
10	11	0	1	1	1

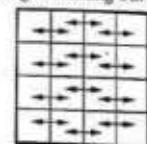
$A + B + C + \bar{D}$	$A + B + \bar{C} + \bar{D}$
$A + B + C + \bar{D}$	$\bar{A} + B + \bar{C} + \bar{D}$
$A + B + C + \bar{D}$	$A + B + \bar{C} + D$
$A + B + C + \bar{D}$	$\bar{A} + B + \bar{C} + D$
$\bar{A} + B + C + \bar{D}$	$\bar{A} + B + \bar{C} + D$

Fig. 4.2.10

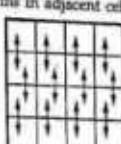
$$(A + B + C + \bar{D}) = M_1, (A + \bar{B} + \bar{C} + D) = M_6, (A + B + \bar{C} + \bar{D}) = M_3, \\ (\bar{A} + \bar{B} + C + \bar{D}) = M_{13}, (\bar{A} + \bar{B} + \bar{C} + D) = M_{14}$$

**4.2.3 Grouping Cells for Simplification**

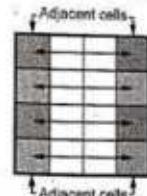
- Once the Boolean function is plotted on the Karnaugh map we have to use grouping technique to simplify the Boolean function.
- The grouping is nothing but combining terms in adjacent cells.



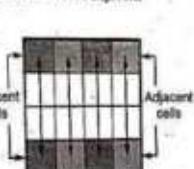
(a) Neighbouring cells in the row are adjacent



(b) Neighbouring cells in the column are adjacent



(c) Leftmost and corresponding rightmost cells are adjacent



(d) Top and corresponding bottom cells are adjacent

Fig. 4.2.11 Adjacent cells

- Two cells are said to be adjacent if they conform the single change rule, i.e. there is only one variable difference between co-ordinates of two cells. For example, the cells for minterms ABC and  $\bar{A}BC$  are adjacent.
- The Fig. 4.2.11 shows the adjacent cells.
- The simplification is achieved by grouping adjacent 1s or 0s in groups of  $2^i$ , where  $i = 1, 2, \dots, n$  and  $n$  is the number of variables.
- When adjacent 1s are grouped then we get result in the sum of products form; otherwise we get result in the product of sums form. Let us see the various grouping rules.

#### 4.2.11 Grouping Two Adjacent Ones (Pair)

Fig. 4.2.12 (a) shows the Karnaugh map for a particular three variable truth table. This K-map contains a pair of 1s that are horizontally adjacent to each other; the first represents  $\bar{A}\bar{B}C$  and the second represents  $\bar{A}BC$ . Note that in these two terms only the B variable appears in both normal and complemented form ( $\bar{A}$  and C remain unchanged). Thus these two terms can be combined to give a result that eliminates the B variable since it appears in both uncomplemented and complemented form. This is easily proved as follows :

$$\begin{aligned} Y &= \bar{A}\bar{B}C + \bar{A}BC \\ &= \bar{A}C(\bar{B} + B) \quad \text{Rule 6 : } [\bar{A} + A = 1] \\ &= \bar{A}C \end{aligned}$$

This same principle holds true for any pair of vertically or horizontally adjacent 1s.

- Fig. 4.2.12 (b) shows an example of two vertically adjacent 1s. These two can be combined to eliminate A variable since it appears in both its uncomplemented and complemented forms. This gives result

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

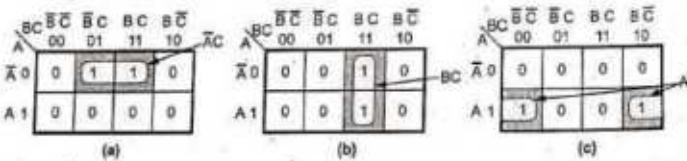


Fig. 4.2.12

- In a Karnaugh map the corresponding cells in the leftmost column and rightmost column are considered to be adjacent. Thus, the two 1s in these columns with a common row can be combined to eliminate one variable. This is illustrated in Fig. 4.2.12 (c).

Here variable B has appeared in both its complemented and uncomplemented forms and hence eliminated as follows :

$$\begin{aligned} Y &= A\bar{B}\bar{C} + A\bar{B}C \\ &= A\bar{C}(\bar{B} + B) \\ &= A\bar{C} \end{aligned}$$

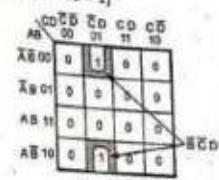
Rule 6 :  $[\bar{A} + A = 1]$ 

Fig. 4.2.12 (d)

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

$$\begin{aligned} &= \bar{B}\bar{C}D(\bar{A} + A) \\ &= \bar{B}\bar{C}D \end{aligned}$$

- Fig. 4.2.12 (e) shows a Karnaugh map that has two overlapping pairs of 1s. This shows that we can share one term between two pairs.

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

$$\begin{aligned} &= \bar{A}\bar{B}C + \bar{A}BC + \bar{A}BC + A\bar{B}C \\ &= \bar{A}C(\bar{B} + B) + B\bar{C}(\bar{A} + A) \\ &= \bar{A}C + B\bar{C} \end{aligned}$$

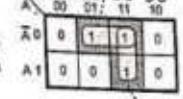
Group 1  $\rightarrow \bar{A}C$ Group 2  $\rightarrow BC$ Rule 5 :  $[A + A = A]$ 

Fig. 4.2.12 (e)

- Fig. 4.2.12 (f) shows a K-map where three group of pairs can be formed. But only two pairs are enough to include all 1s present in the K-map. In such cases third pair is not required.

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

$$\begin{aligned} &= \bar{A}C(\bar{B} + B) + A\bar{B}(C + \bar{C}) \\ &= \bar{A}C + A\bar{B} \end{aligned}$$

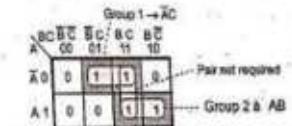
Rule 6 :  $[\bar{A} + A = 1]$ 

Fig. 4.2.12 (f) Examples of combining pairs of adjacent ones

A pair is a group of two adjacent cells in a Karnaugh map. It cancels one variable in a K-map simplification.

**4.2.13 Grouping Four Adjacent Ones (Quad)**

- In a Karnaugh map we can group four adjacent 1s. The resultant group is called Quad.
- Fig. 4.2.13 shows several examples of quads.

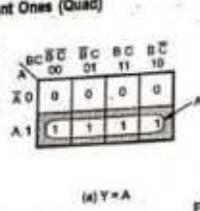
(a)  $Y = A$ 

Fig. 4.2.13

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

(b)  $Y = CD$ 

- Fig. 4.2.13 (a) shows the four 1s are horizontally adjacent.

- Fig. 4.2.13 (b) shows the four 1s are vertically adjacent.

- A K-map in Fig. 4.2.13 (c) contains four 1s in a square and they are considered adjacent to each other.

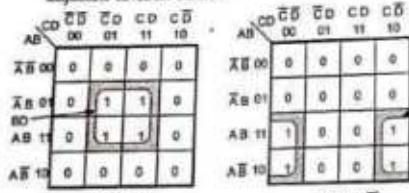
(c)  $Y = BD$ 

Fig. 4.2.13

(d)  $Y = AD$ (e)  $Y = \bar{BD}$ 

- The four 1s in Fig. 4.2.13 (d) are also adjacent.
- In Fig. 4.2.13 (e) the top and bottom rows are considered to be adjacent to each other and the leftmost and rightmost columns are also adjacent to each other.

- When a quad is combined, two variables are eliminated. For example, in Fig. 4.2.13 (c) we have following terms with 4 variables :

$$\begin{aligned}
 Y &= \bar{A}B\bar{C}D + \bar{A}BC\bar{D} + A\bar{B}\bar{C}D + A\bar{B}CD \\
 &= \bar{A}BD(\bar{C} + C) + ABD(\bar{C} + C) \\
 &= \bar{A}BD + ABD \\
 &= BD(\bar{A} + A) \\
 &= BD
 \end{aligned}$$

(Only two variables in the result variables A and C are eliminated)

- Fig. 4.2.13 (f) shows overlapping groups. As mentioned earlier one term can be shared between two or more groups.

Quad is a group of four adjacent cells in a Karnaugh map. It cancels two variables in a K-map simplification.

Fig. 4.2.13 Examples of combining quads of adjacent ones

**4.2.14 Grouping Eight Adjacent Ones (Octet)**

- In a Karnaugh map we can group eight adjacent 1s. The resultant group is called octet. Fig. 4.2.14 shows several examples of octets.
- Fig. 4.2.14 (a) shows the eight 1s are horizontally adjacent.
- Fig. 4.2.14 (b) shows they are vertically adjacent.
- From the Fig. 4.2.14 we can easily observe that when an octet is combined in a four variable map, three of the four variables are eliminated because only one variable remains unchanged.

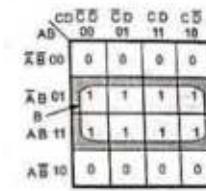
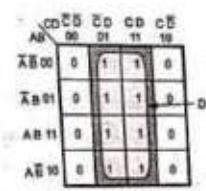
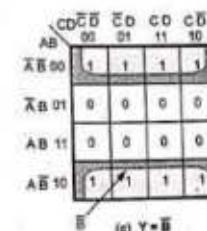
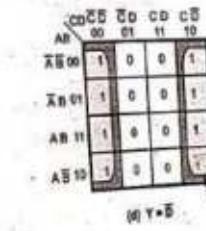
(a)  $Y = B$ (b)  $Y = D$ (e)  $Y = B$ (d)  $Y = \bar{B}$ 

Fig. 4.2.14 Examples of combining octets of adjacent ones

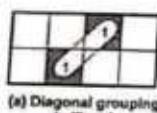
- For example, in K-map shown in Fig. 2.3.14 (a) we have following terms :

$$\begin{aligned}
 Y &= \bar{A}B\bar{C}\bar{D} + A\bar{B}\bar{C}\bar{D} + \bar{A}B\bar{C}D + \bar{A}B\bar{C}\bar{D} \\
 &\quad + AB\bar{C}\bar{D} + A\bar{B}\bar{C}D + ABC\bar{D} + ABC\bar{D} \\
 &= \bar{A}B\bar{C}(\bar{D} + D) + \bar{A}B\bar{C}(D + \bar{D}) + A\bar{B}\bar{C}(\bar{D} + D) + A\bar{B}\bar{C}(D + \bar{D}) \\
 &= \bar{A}B\bar{C} + \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC = \bar{A}B(\bar{C} + C) + AB(\bar{C} + C) \\
 &= \bar{A}B + AB \\
 &= B(\bar{A} + A) = B
 \end{aligned}$$

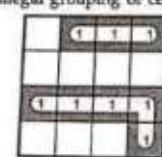
Octet is a group of eight adjacent cells in a Karnaugh map. It cancels three variables in a K-map simplifications.

#### 4.2.4 Illegal Grouping

- The Fig. 4.2.15 shows the examples of illegal grouping of cells.



(a) Diagonal grouping is illegal



(b) Grouping of odd number of cells is illegal

Fig. 4.2.15

**Example 4.2.5** Using K-map find the boolean function and its complement for the following  
 $F(A, B, C, D) = \sum(1, 2, 3, 4, 6, 8, 9, 10, 11, 12, 14)$

GTU - May-14, Marks 7

Solution :

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

$$\text{and } \bar{F} = \bar{C}\bar{D} + \bar{B}\bar{D} + A\bar{B} + BD$$

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

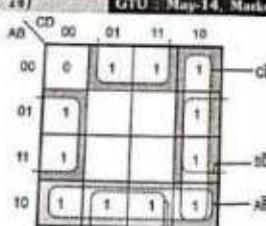


Fig. 4.2.16

#### Review Question

- Explain SOP and POS forms of K-map for three variables.

#### 4.3 Simplification SOP Functions using K-Maps

GU - May-12, 13, Dec. 12, 13, Summer 17, 18

- A pair of 1s eliminates one variable, a quad of 1s eliminates two variables and an octet of 1s eliminates three variables.
- In general, when a variable appears in both complemented and uncomplemented form within a group, that variable is eliminated from the resultant expression.
- Variables that are same in all with the group must appear in the final expression.
- Each group gives us a product term and summation of all product term gives us a Boolean expression.
- Each product term implies the function and, hence is an implicant of the function.
- All the implicants of a function determined using a Karnaugh map are the prime implicants.
- From the above discussion we can outline generalized procedure to simplify Boolean expressions as follows :

  - Plot the K-map and place 1s in those cells corresponding to the 1s in the truth table or sum of product expression. Place 0s in other cells.
  - Check the K-map for adjacent 1s and encircle those 1s which are not adjacent to any other 1s. These are called isolated 1s.
  - Check for those 1s which are adjacent to only one other 1 and encircle such pairs.
  - Check for quads and octets of adjacent 1s even if it contains some 1s that have already been encircled. While doing this make sure that there are minimum number of groups.
  - Combine any pairs necessary to include any 1s that have not yet been grouped.
  - Form the simplified expression by summing product terms of all the groups.

#### Illustrative Examples

**Example 4.3.1** Minimize the expression  $Y = \bar{AB}C + \bar{A}\bar{B}C + ABC + AB\bar{C} + A\bar{B}C$ .

Solution :

**Step 1 :** Fig. 4.3.1 (a) shows the K-map for three variables and it is plotted according to the given expression.

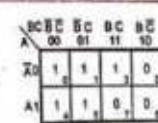


Fig. 4.3.1 (a)

**Step 2 :** There are no isolated 1s.

**Step 3 :** 1 in the cell 3 is adjacent only to 1 in the cell 1. This pair is combined and referred to as group 1.

**Step 4 :** There is no octet, but there is a quad. Cells 0, 1, 4 and 5 form a quad. This quad is combined and referred to as group 2.

**Step 5 :** All 1s have already been grouped.

**Step 6 :** Each group generates a term in the expression for Y. In group 1, B variable is eliminated and in group 2, variables A and C are eliminated and we get,

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

**Example 4.3.2** Minimize the expression

$$Y = ABCD + ABC\bar{D} + A\bar{B}CD + A\bar{B}C\bar{D} + ABC\bar{D} + A\bar{B}C\bar{D}$$

**Solution :**

**Step 1 :** Fig. 4.3.2 (a) shows the K-map for four variables and it is plotted according to the given expression.



Fig. 4.3.2 (b)

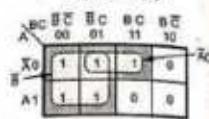


Fig. 4.3.2 (c)

**Step 2 :** Cell 2 is the only cell containing a 1 that is not adjacent to any other 1. It is referred to separately as group 1.



Fig. 4.3.2 (a)

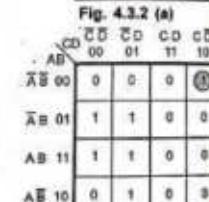


Fig. 4.3.2 (b)

**Step 3 :** 1 in the cell 9 is adjacent only to 1 in the cell 13. This pair is combined and referred to as group 2.

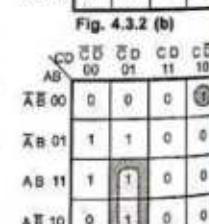


Fig. 4.3.2 (c)

**Step 4 :** There is no octet, but there is a quad. Cells 0, 1, 4 and 5 form a quad. This quad is combined and referred to as group 3.

**Step 5 :** All 1s have already been grouped.

**Step 6 :** Each group generates a term in the expression for Y. In group 1 variable A is not eliminated. In group 2 variable B is eliminated and in group 3 variables A and D are eliminated and we get,

$$Y = \overline{A}\overline{B}CD + A\overline{C}D + B\overline{C}$$

**Example 4.3.3** Reduce the following four variable function to its minimum sum of products form:

$$Y = ABC\bar{D} + ABCD + A\bar{B}CD + AB\bar{C}D + ABC\bar{D} + ABCD + A\bar{B}CD + ABCD$$

**Solution :**

**Step 1 :** Fig. 4.3.3 (a) shows the K-map for four variables and it is plotted according to the given expression.

**Step 2 :** There are no isolated 1s.

**Step 3 :** There are no such 1s which are adjacent to only one other 1.

**Step 4 :** There are three quads formed by cells 0, 2, 8, 10, cells 8, 10, 12, 14 and cells 2, 3, 10, 11. These quads are combined and referred to as group 1, group 2 and group 3 respectively.

**Step 5 :** All 1s have already been grouped.

**Step 6 :** Each group generates a term in the expression for Y. In group 1 variables A and C are eliminated, in group 2 variables B and C are eliminated and in group 3 variables A and D are eliminated and we get,

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

**Example 4.3.4** Reduce the following function to its minimum sum of products form:

$$Y = ABCD + A\bar{B}CD + ABC\bar{D} + A\bar{B}CD + ABCD + A\bar{B}CD + A\bar{B}CD + ABCD$$

Fig. 4.3.3 (b)

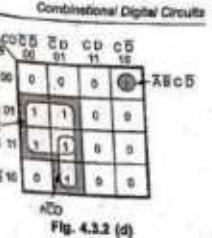


Fig. 4.3.3 (d)

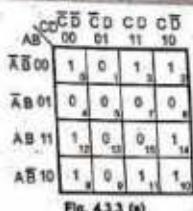


Fig. 4.3.3 (e)

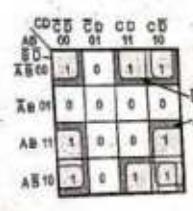


Fig. 4.3.3 (f)

**Solution :**

**Step 1 :** Fig. 4.3.4 (a) shows the K-map for four variables and it is plotted according to the given expression.

**Step 2 :** There are no isolated 1s.

**Step 3 :** The 1 in the cell 1 is adjacent only to 1 in the cell 5, the 1 in the cell 6 is adjacent only to the 1 in the cell 7, the 1 in the cell 12 is adjacent only to the 1 in the cell 13 and the 1 in the cell 11 is adjacent only to the 1 in the cell 15. These pairs are combined and referred to as group 1-4 respectively.

**Step 4 :** There is no octet, but there is a quad. However, all 1s in the quad have already been grouped. Therefore this quad is ignored.

**Step 5 :** All 1s have already been grouped.

**Step 6 :** Each group generates a term in the expression for Y. In group 1 variable B is eliminated. Similarly, in group 2-4 variables D, D and B are eliminated one in each group. We finally get minimum sum of products form as,

$$Y = \bar{A} \bar{C}D + \bar{A}BC + ABC + ACD$$

**Example 4.3.5** Simplify the logic function specified by the truth Table 4.3.1 using the Karnaugh map method. Y is the output variable, and A, B and C are the input variables.

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

Table 4.3.1

AB	CD	$\bar{C}D$	$\bar{D}$	CD	$C\bar{D}$
$\bar{A}\bar{B}\ 00$	0	1	0	0	0
$\bar{A}\bar{B}\ 01$	0	1	1	1	0
$\bar{A}\bar{B}\ 11$	1	1	1	0	0
$\bar{A}\bar{B}\ 10$	0	0	1	1	0

Fig. 4.3.4 (a)

AB	CD	$\bar{C}D$	$\bar{D}$	CD	$C\bar{D}$
$\bar{A}\bar{B}\ 00$	0	1	0	0	0
$\bar{A}\bar{B}\ 01$	0	1	1	1	0
$\bar{A}\bar{B}\ 11$	1	1	1	0	0
$\bar{A}\bar{B}\ 10$	0	0	1	0	0

Fig. 4.3.4 (b)

**Solution :**

**Step 1 :** Fig. 4.3.5 (a) shows the K-map for three variables and it is plotted according to given truth table.

**Step 2 :** There are no isolated 1s.

**Step 3 :** The 1 in the cell 0 is adjacent only to 1 in the cell 4 and the 1 in the cell 3 is adjacent only to 1 in the cell 7. These two pairs are grouped and referred to as group 1 and group 2.

**Step 4 :** There is no octet and quad.

**Step 5 :** All 1s have already been grouped.

**Step 6 :** In group 1 and group 2 variable A is eliminated and we get,

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

AC	$\bar{C}$	BC	$\bar{B}C$
A0	1	0	1
A1	1	0	0

Fig. 4.3.5 (a)

AC	$\bar{C}$	BC	$\bar{B}C$
A0	1	0	1
A1	1	0	0

Fig. 4.3.5 (b)

**Example 4.3.6** Reduce the following function using Karnaugh map technique and implement using basic gates

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

**Solution :** The given function is not in the standard sum of products form. It is converted into standard SOP form as given below.

$$\begin{aligned} f(A, B, C, D) &= \bar{A}\bar{B}D + ABCD + \bar{A}BD + ABC\bar{D} \\ &= \bar{A}\bar{B}D(C + \bar{C}) + ABCD + \bar{A}BD(C + \bar{C}) + ABC\bar{D} \\ &= \bar{A}\bar{B}CD + \bar{A}BC\bar{D} + ABCD + \bar{A}BCD + \bar{A}B\bar{C}D + ABC\bar{D} \end{aligned}$$

**Step 1 :** Fig. 4.3.6 (a) shows the K-map for four variables and it is plotted according to expression in standard SOP form.

**Step 2 :** There are no isolated 1s.

AB	CD	$\bar{C}D$	$\bar{D}$	CD	$C\bar{D}$
$\bar{A}\bar{B}\ 00$	0	1	1	0	0
$\bar{A}\bar{B}\ 01$	0	1	0	1	0
$\bar{A}\bar{B}\ 11$	1	0	0	1	1
$\bar{A}\bar{B}\ 10$	0	0	1	0	0

Fig. 4.3.6 (a)

AB	CD	$\bar{C}D$	$\bar{D}$	CD	$C\bar{D}$
A0	0	1	1	0	0
A1	0	1	0	1	0
AB11	1	0	0	1	1
AB10	0	0	1	0	0

Fig. 4.3.6 (b)

**Step 3 :** The 1 in the cell 12 is adjacent only to the 1 in the cell 14. This pair is combined and referred to as group 1.

**Step 4 :** There is a quad. Cells 1, 3, 5 and 7 form a quad. This quad is referred to as group 2.

**Step 5 :** All 1s have already been grouped.

**Step 6 :** In group 1 variable C is eliminated and in group 2 variables B and C are eliminated. We get simplified equation as,

$$Y = ABD + \bar{A}D$$

**Example 4.3.7** Reduce the following function using K-map technique.  
 $f(A, B, C, D) = \sum m(0, 1, 4, 8, 9, 10)$

**Solution :**

**Step 1 :** Fig. 4.3.7 (a) shows the K-map for four variables and it is plotted according to given minterms.

**Step 2 :** There are no isolated 1s.

Combinational Digital Circuits			
AB	CD	$\bar{C}\bar{D}$	$\bar{C}D$
00	1	1	0
01	0	1	0
11	1	0	0
10	0	0	0

Fig. 4.3.8 (c)

Combinational Digital Circuits			
AB	CD	$\bar{C}\bar{D}$	$\bar{C}D$
00	1	1	0
01	1	0	0
11	0	0	0
10	1	1	0

Fig. 4.3.7 (a)

Combinational Digital Circuits			
AB	CD	$\bar{C}\bar{D}$	$\bar{C}D$
00	1	1	0
01	1	0	0
11	0	0	0
10	1	1	0

Fig. 4.3.7 (b)

Combinational Digital Circuits			
AB	CD	$\bar{C}\bar{D}$	$\bar{C}D$
00	1	1	0
01	1	0	0
11	0	0	0
10	1	1	0

Fig. 4.3.7 (c)

**Step 3 :** Cell 4 is adjacent only to cell 0 and cell 8 is adjacent only to cell 9. These two pairs are combined and referred to as group 1 and group 2 respectively.

**Step 4 :** There is a quad. Cells 0, 1, 8 and 9 form a quad. This quad is referred to as group 3.

**Step 5 :** All 1s have already been grouped.

**Step 6 :** In group 1, B and in group 2, C are eliminated respectively. In group 3, A and D are eliminated and finally we get

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

Fig. 4.3.7 (d)

**Example 4.3.8** Obtain the simplified expression in sum of product for the following boolean functions:  
a)  $F = \sum (0, 1, 4, 5, 10, 11, 12, 14)$  b)  $F = \sum (11, 12, 13, 14, 15)$

(GTU : May-13, Marks 7)

**Solution :** a)  $F = \sum (0, 1, 4, 5, 10, 11, 12, 14)$

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

b)  $F = \sum (11, 12, 13, 14, 15)$

$$F = AB + ACD$$

Combinational Digital Circuits			
AB	CD	$\bar{C}\bar{D}$	$\bar{C}D$
00	1	1	0
01	1	0	0
11	0	0	0
10	1	1	0

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

(a)

Combinational Digital Circuits			
AB	CD	$\bar{C}\bar{D}$	$\bar{C}D$
00	0	0	0
01	0	0	0
11	1	1	1
10	0	0	0

$$F = AB + ACD$$

(b)

Fig. 4.3.8

**Example 4.3.9** Simplify the boolean function with Karnaugh map.  
 $F(w, x, y, z) = \sum (0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$

(GTU : Dec-13, Marks 3)

**Solution :**  $F(w, x, y, z) = \sum (0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$

$$F(w, x, y, z) = \bar{y} + w\bar{z} + x\bar{z}$$

Combinational Digital Circuits						
wx	yz	00	01	11	10	wz
00	1	1	1	1	1	0
01	1	1	1	1	1	0
11	1	1	1	1	1	1
10	1	1	1	1	1	0

$$F(w, x, y, z) = \bar{y} + w\bar{z} + x\bar{z}$$

Fig. 4.3.9

**Example 4.3.10** Simplify the boolean function with K-map  
 $F = ABC + ACD + ABCD + ABD$

Solution :  $F = \bar{A}BC + BCD + \bar{A}BC\bar{D} + A\bar{B}C$   
 $F = \bar{B}\bar{C} + \bar{B}D + \bar{A}CD$

Combinational Digital Circuits				
GTU : May-12, Dec-13, Marks 4				
AB	00	01	11	
CD	00	1	1	1
	01			1
	11			
	10	1	1	1

BC

Fig. 4.3.10

**Example 4.3.11** Reduce to simplest form using K-map :  $F(A, B, C, D) = \sum(0, 1, 2, 5, 8, 9, 10)$   
[Ans. :  $F(A, B, C, D) = \bar{B}\bar{C} + \bar{B}\bar{D} + \bar{A}CD$ ] GTU : Summer-18, Marks 4

Solution :

CD	00	01	11	10
AB	1	1		1
		1		

$\bar{B}\bar{D}$

$\bar{A}CD$

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

$\therefore F = \bar{B}\bar{C} + \bar{B}\bar{D} + \bar{A}CD$

Fig. 4.3.11

**Examples for Practice**

**Example 4.3.12** Simplify following logical expression using Karnaugh maps  
 $Y = \bar{A}\bar{B}\bar{C} + A\bar{B}C + A\bar{B}\bar{C} + \bar{A}\bar{B}C + A\bar{B}\bar{C}$  [Ans. :  $Y = \bar{A}\bar{B} + \bar{C}$ ]

**Example 4.3.13** Simplify the following function  
 $f_1(A, B, C, D) = \sum m(0, 3, 5, 6, 9, 10, 12, 15)$   
[Ans. :  $(A \oplus B) \oplus (C \oplus D)$ ]

**Example 4.3.14** Simplify the following function  
 $f_2(A, B, C, D) = \sum m(0, 1, 2, 3, 11, 12, 14)$   
[Ans. :  $\bar{A}B + A\bar{B}\bar{D} + \bar{B}CD$ ]

**Example 4.3.15** Simplify the following using K-map.  
 $X = A'B + A'B'C + ABC' + AB'C'$  [Ans. :  $X = \bar{A}C + B'$ ]

**Example 4.3.16** Solve the following using minimization technique  
 $Z = f(A, B, C, D) = \sum(0, 2, 4, 7, 11, 13, 15)$   
[Ans. :  $Z = \bar{A}B\bar{D} + \bar{A}C\bar{D} + BCD + ABD + ACD$ ]

**4.3.1 Essential Prime Implicants**

- After grouping the cells, the sum terms which appear in the K-map are called prime implicants groups.
- It is observed that some cells may appear in only one prime implicants group; while other cells may appear in more than one prime implicants group.
- In Fig. 4.3.7 (c), cells 1, 4, 9 and 10 appear in only one prime implicants group. These cells are called essential cells and corresponding prime implicants are called essential prime implicants.

**4.3.2 Incompletely Specified Functions (Don't Care Terms)**

- In some logic circuits, certain input conditions never occur, therefore the corresponding output never appears. In such cases the output level is not defined, it can be either HIGH or LOW. These output levels are indicated by 'X' or 'd' in the truth tables and are called don't care outputs or don't care conditions or incompletely specified functions.
- As shown in truth table given by Table 4.3.2, outputs are defined for input conditions from 0 0 0 to 1 0 1. For remaining two conditions of input, output is not defined, hence these are called don't care conditions for this truth table.
- A circuit designer is free to make the output for any "don't care" condition either a '0' or a '1' in order to produce the simplest output expression.

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

Table 4.3.2

**4.3.2.1 Describing Incomplete Boolean Function**

- We can describe the Boolean function using either a minterm canonical formula or a maxterm canonical formula.
- In order to obtain similar-type expressions for incomplete Boolean functions we use additional term to specify don't care conditions in the original expression. This is illustrated in the following examples.

In expression,

$$f(A, B, C) = \sum m(0, 2, 4) + d(1, 5)$$

- minterms are 0, 2 and 4. The additional term  $d(1, 5)$  is introduced to specify the don't care conditions.
- This term specifies that outputs for minterms 1 and 5 are not specified and hence these are don't care conditions.
  - Letter d is used to indicate don't care conditions in the expression.
  - The above expression indicates how to represent don't care conditions in the minterm canonical formula. In the similar manner, we can specify the don't care conditions in the maxterm canonical formula. For example,
- $$f(A, B, C) = \prod M(2, 5, 7) + d(1, 3)$$

#### 4.3.2.2 Don't Care Conditions in Logic Design

- Consider the logic circuit for an even parity generator for 4-bit BCD number. The Table 4.3.3 shows the truth table for even-parity generator.
- The truth table shows that the output for last six input conditions cannot be specified, because such input conditions does not occur when input is in the BCD form.
- The Boolean function for even parity generator with 4-bit BCD input can be expressed in minterm canonical formula as,

$$f(A, B, C, D) = \sum m(1, 2, 4, 7, 8) + d(10, 11, 12, 13, 14, 15)$$

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

Table 4.3.3 Truth table for even parity generator with 4-bit BCD input

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

Table 4.3.4

output for cell ABC is taken as 1 to form a quad and don't care output for cell ABC is taken as 0, since it is not helping any way to reduce an expression. Using don't care conditions in this way we get the simplified Boolean expression as

$$Y = C$$

- It is important to decide which don't cares to change to 0 and which to 1 to produce the best K-map grouping (i.e. the simplest expression).

#### Illustrative Examples

**Example 4.3.17** Find the reduced SOP form of the following function.  
 $f(A, B, C, D) = \sum m(1, 3, 7, 11, 15) + \sum d(0, 2, 4)$

Solution :

AB	$\bar{C}\bar{D}$	$\bar{C}D$	$CD$	$C\bar{D}$
AB 00	X	1	1	X
AB 01	X	0	1	0
AB 11	0	0	1	0
AB 10	0	0	1	1

(a)

AB	$\bar{C}\bar{D}$	$\bar{C}D$	$CD$	$C\bar{D}$
AB 00	1	1	1	1
AB 01	0	0	1	0
AB 11	0	0	1	0
AB 10	0	0	1	0

(b)

Fig. 4.3.13

- To form a quad of cells 0, 1, 2 and 3 the don't care conditions 0 and 2 are replaced by 1s.
- The remaining don't care condition is replaced by 0 since it is not required to form any group. With these replacements we get the simplified equation as

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

Group 1      Group 2

**Example 4.3.18** Find the prime implicants for the following function and determine which are essential.  $F(w, x, y, z) = \sum (0, 2, 4, 5, 6, 7, 8, 10, 13, 15)$

**Solution :**

- The minterms of the given function are marked with 1's in the map of Fig. 4.3.14.

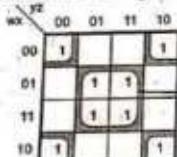
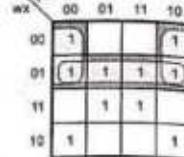
(a) Essential prime implicants  
 $\bar{x}z$  and  $xz$ (b) Prime implicants  
 $wx$  and  $w\bar{z}$ 

Fig. 4.3.14

- The Fig. 4.3.14 (a) shows two essential prime implicants.
- The term  $xz$  is essential because there is only one way to include minterms  $m_2$  and  $m_{10}$  within four adjacent squares.
- There is only one way that minterms  $m_4$  and  $m_{10}$  can be combined with four adjacent squares and this gives second term  $\bar{x}z$ .
- The two essential prime implicants cover eight minterms.
- The remaining two minterms,  $m_4$  and  $m_6$  must be considered next.
- The Fig. 4.3.14 shows the two possible ways of including remaining two minterms with prime implicants.
- Minterms  $m_4$  and  $m_6$  can be included with either  $wx$  or  $w\bar{z}$ .
- The simplified expression is obtained from the logical sum of the two essential prime implicants and any one prime implicant that includes minterms  $m_4$  and  $m_6$ .
- There are two possible ways that the function can be expressed in the simplified form :

$$F = \bar{x}z + xz + \bar{w}x = \bar{x}z + xz + w\bar{z}$$

**Example 4.3.19** Express the following function as the minimal sum of products using K-map.  $f(a,b,c,d) = \Sigma(0,2,4,5,6,8,10,15) + \Sigma_0(7,13,14)$

**Solution :**

- As shown in Fig. 4.3.15 the given example has three solutions and all are correct. Students are expected to give any one solution.

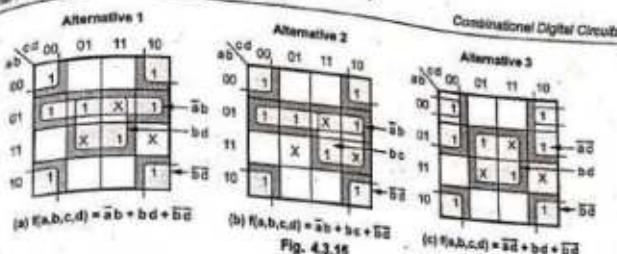
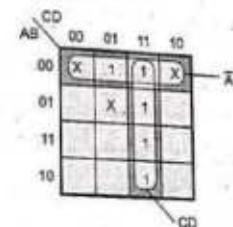


Fig. 4.3.15

**Example 4.3.20** Implement the function  $F = \sum(0, 3, 7, 11, 15)$  with don't care conditions  $d = \sum(0, 2, 5)$ . GTU : Mar-13, Dec-13, Marks 5

$$\text{Solution : } F = \sum(1, 3, 7, 11, 15) + \sum(0, 2, 5)$$



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

Fig. 4.3.16

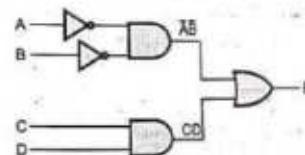


Fig. 4.3.17

**Example 4.3.21** Obtain the simplified expression in sum of product using K-map.  $\Sigma_0(2,4,5,6,8,10,15) + \Sigma_1(7,13,14)$ . GTU : Dec-12, Marks 7

$$\text{Solution : } F = \bar{x}z + \bar{w}x\bar{y} + w(\bar{x}y + x\bar{y})$$

$$F = \bar{x}z + \bar{w}x\bar{y} + w\bar{x}y + wxy$$

TECHNICAL PUBLICATIONS™ An up beat for knowledge

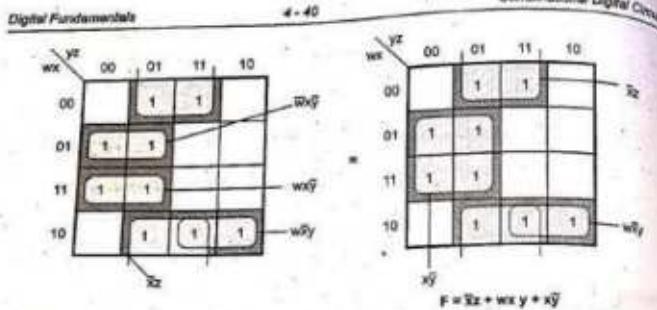


Fig. 4.3.18

$$F = \bar{w}z + w\bar{x}y + wxy$$

Example 4.3.22 Simplify Boolean function using K-map

$$F(w, x, y, z) = \sum(1, 3, 5, 8, 9, 11, 15)$$

$$d(w, x, y, z) = \sum(2, 13)$$

Solution :  $F = w\bar{x}y + \bar{y}z + wz + \bar{w}z$

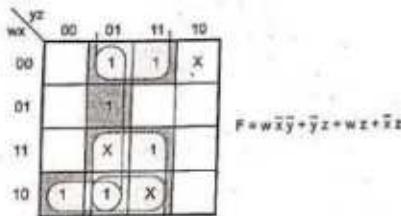


Fig. 4.3.19

#### Examples for Practice

Example 4.3.23 : Simplify the following switching function using Karnaugh map  
 $F(A, B, C, D) = \sum(0, 5, 7, 8, 9, 10, 11, 14, 15) + \phi(1, 4, 13)$

[Ans. :  $\bar{B}\bar{C} + BD + AC$ ]

Example 4.3.24 : Determine the minimal sum of product form of

$$F(w, x, y, z) = \sum m(4, 5, 7, 12, 14, 15) + d(3, 8, 10)$$

[Ans. :  $w\bar{x}y + xyz + w\bar{z}$ ]

TECHNICAL PUBLICATIONS® - An up beat for knowledge

#### Review Questions

- Give the steps for simplification of SOP expression.
- What do you mean by essential prime implicants?
- What is don't care condition?

#### 4.4 Simplification of POS Functions using K-Maps

GTU : Summer-17

- In practice, the designer should examine both the sum of products and product of sums reductions to ascertain which is more simplified.
- Once the expression is plotted on the K-map instead of making the groups of ones, we have to make groups of zeros.
- Each group of zero results a sum term and it is nothing but the prime implicant. The technique for using maps for POS reductions is a simple step by step process and it is similar to the one used earlier.
  - Plot the K-map and place 0s in those cells corresponding to the 0s in the truth table or maxterms in the product of sums expression.
  - Check the K-map for adjacent 0s and encircle those 0s which are not adjacent to any other 0s. These are called isolated 0s.
  - Check for those 0s which are adjacent to only one other 0 and encircle such pairs.
  - Check for quads and octets of adjacent 0s even if it contains some 0s that have already been encircled. While doing this make sure that there are minimum number of groups.
  - Combine any pairs necessary to include any 0s that have not yet been grouped.
  - Form the simplified POS expression for F by taking product of sum terms of all the groups.

To get familiar with these steps we will solve some examples.

#### Illustrative Examples

Example 4.4.1 Minimize the expression

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

Solution :  $(\bar{A} + B + C) = M_1, (A + \bar{B} + \bar{C}) = M_3,$

$$(\bar{A} + \bar{B} + \bar{C}) = M_7, (\bar{A} + B + C) = M_4,$$

$$(A + B + C) = M_0$$

TECHNICAL PUBLICATIONS® - An up beat for knowledge

**Step 1 :** Fig. 4.4.1 (a) shows the K-map for three variable and it is plotted according to given maxterms.

**Step 2 :** There are no isolated 0s.

**Step 3 :** 0 in the cell 4 is adjacent only to 0 in the cell 0 and 0 in the cell 7 is adjacent only to 0 in the cell 3. These two pairs are combined and referred to as group 1 and group 2 respectively.

**Step 4 :** There are no quads and octets.

**Step 5 :** The 0 in the cell 1 can be combined with 0 in the cell 3 to form a pair. This pair is referred to as group 3.

**Step 6 :** In group 1 and in group 2, A is eliminated, whereas in group 3 variable B is eliminated and we get,

$$Y = (B+C)(\bar{B}+\bar{C})(A+\bar{C}) \quad \text{Group 1} \rightarrow B+C$$

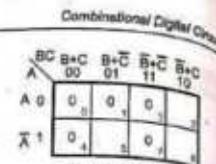


Fig. 4.4.1 (a)

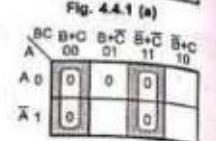


Fig. 4.4.1 (b)

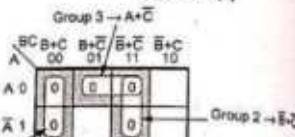


Fig. 4.4.1 (c)

#### Example 4.4.2 Minimize the following expression in the POS form

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

**Solution :**

$$\begin{aligned}(\bar{A} + \bar{B} + C + D) &= M_{12}, (\bar{A} + \bar{B} + \bar{C} + D) = M_{14}, (\bar{A} + \bar{B} + \bar{C} + \bar{D}) = M_{15} \\(\bar{A} + B + C + D) &= M_8, (A + \bar{B} + \bar{C} + D) = M_5, (A + \bar{B} + \bar{C} + \bar{D}) = M_7 \\(A + B + C + D) &= M_0 \text{ and } (\bar{A} + \bar{B} + C + \bar{D}) = M_{13}\end{aligned}$$

**Step 1 :** Fig. 4.4.2 (a) shows the K-map for four variable and it is plotted according to given maxterms.

**Step 2 :** There are no isolated 0s.

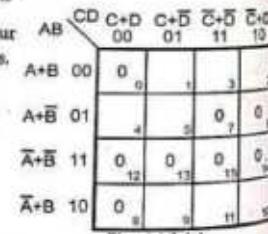


Fig. 4.4.2 (a)

**Step 3 :** 0 in the cell 0 is adjacent only to 0 in the cell 8. This pair is combined and referred to as group 1.

**Step 4 :** There are two quads. Cells 12, 13, 14 and 15 forms a quad 1 and cells 6, 7, 14, 15 forms a quad 2. These two quads are referred to as group 2 and group 3, respectively.

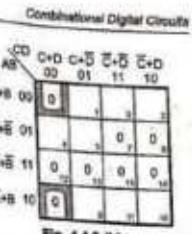


Fig. 4.4.2 (b)

**Step 5 :** All 0s have already been grouped.

**Step 6 :** In group 1, variable A is eliminated. In group 2, variable C and D are eliminated and in group 3 variables A and D are eliminated. Therefore we get simplified POS expression as,

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

#### Example 4.4.3 Reduce the following function using K-map technique

$$f(A, B, C, D) = \prod M(0, 2, 3, 8, 9, 12, 13, 15)$$

**Solution :**

**Step 1 :** Fig. 4.4.3 (a) shows the K-map for four variables and it is plotted according to given maxterms.

**Step 2 :** There are no isolated 0s.

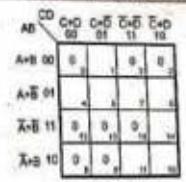


Fig. 4.4.3 (a)

**Step 3 :** The 0 in the cell 15 is adjacent only to 0 in the cell 13 and 0 in the cell 3 is adjacent only to 0 in the cell 2. These two pairs are combined and referred to as group 1 and group 2, respectively.

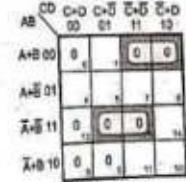


Fig. 4.4.3 (b)

**Step 4 :** The cells 8, 9, 12 and 13 form a quad which is referred to as group 3.

**Step 5 :** The remaining 0 in the cell 0 is combined with the 0 in the cell 2 to form a pair, which is referred to as group 4.

**Step 6 :** In group 1 and in group 4 variable C is eliminated. In group 2 variable D is eliminated and in group 3 variables B and D are eliminated. Therefore, we get simplified expression in POS form as,

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

**Example 4.4.4** Reduce the expression in SOP and POS form using K-map.

$$F(A, B, C, D) = \sum m(1, 5, 6, 12, 13, 14) + d(2, 4)$$

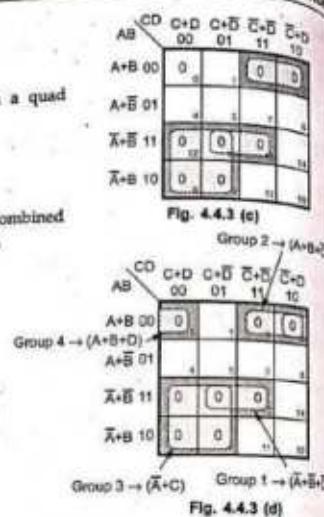


Fig. 4.4.3 (c)

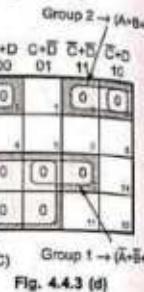


Fig. 4.4.3 (d)

Solution :

		CD		AB	
		00	01	11	10
					X
				1	
		X	1		
				1	
					1

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

Fig. 4.4.4

		CD		AB	
		00	01	11	10
					X
				0	
		X		0	
				0	
				0	0
				0	0

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

**Examples for Practice**

**Example 4.4.5 :** Simplify the following Boolean function for minimal POS form  
 $F(w, x, y, z) = \pi M(4, 5, 6, 7, 8, 12) + d(1, 2, 3, 9, 11, 14)$

**Example 4.4.6 :** Reduce the following function using K-map  
 $F(A, B, C) = \pi M(0, 1, 2, 3, 4, 7)$  [Ans. :  $F = A(\bar{B} + \bar{C}) \cdot (B + C)$ ]

**Example 4.4.7 :** Solve the following using minimization technique  
 $Z = f(A, B, C, D) = \pi(1, 2, 3, 6, 8, 11, 14, 15)$  [Ans. :  $Z = (A + B + D)(A + B + \bar{C})(\bar{B} + \bar{C} + D)(\bar{A} + \bar{C} + D)(\bar{A} + B + C + D)$ ]

**Review Question**

1. Give the steps for simplification of POS expression.

**4.5 Summary of Rules for K-Map Simplification**

Rules for simplifying logic function using K-map are :

1. Group should not include any cell containing a zero.
2. The number of cells in a group must be a power of 2, such as 1, 2, 4, 8 or 16.
3. Group may be horizontal, vertical but not diagonal.
4. Cell containing 1 must be included in at least one group.
5. Groups may overlap.
6. Each group should be as large as possible to get maximum simplification.
7. Groups may be wrapped around the map. The leftmost cell in a row may be grouped with the rightmost cell and the top cell in a column may be grouped with the bottom cell.
8. A cell may be grouped more than once. The only condition is that every group must have at least one cell that does not belong to any other group. Otherwise, redundant terms will result.
9. We need not group all don't care cells, only those that actually contribute to a maximum simplification.
10. All above rules are stated considering the SOP simplification. In case of POS simplification, all rules are same except 0 (zero) takes place of 1 (one).

**Review Question**

1. State the rules for K-map simplification.

**4.6 Limitation of Karnaugh Map**

- The map method of simplification is convenient as long as the number of variables does not exceed five or six. As the number of variables increases it is difficult to make judgements about which combinations form the minimum expression. In case of complex problem with 7, 8 or even 10 variables it is almost an impossible task to simplify expression by the mapping method.
- The K-map simplification is manual technique and simplification process is heavily depends on the human abilities.

To meet this need, W. V. Quine and E. J. McCluskey developed an exact tabular method to simplify the Boolean expression. This method is called the Quine McCluskey or tabular method.

**Review Question**

- Explain the limitations of Karnaugh map.

**4.7 Realizing Logic Function with Gates**

GTU : Dec-11, 12, 13, May-12, 13, Summer-15, 16, 18, Winter-14, 15, 18

- The Boolean algebra is used to express the output of any combinational network. Such a network can be implemented using logic gates.

**4.7.1 Implementation of SOP Boolean Expression**

- Consider the Boolean expression

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

- In this expression, we have three product terms with 2 literals in each product term.

- We can implement these product terms by using three 2-input AND gates, as shown in the Fig. 4.7.1.

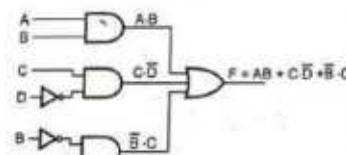


Fig. 4.7.1 Implementation of SOP Boolean expression

- Expression tells that these product terms should be ORed to get the output F.
- We have three product terms so we have to use 3-input OR gate to obtain the sum of products.

- Literals are complemented using NOT gates.

**4.7.2 Implementation of POS Boolean Expression**

- Consider the Boolean expression

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

- In this expression, we have three sum terms with 2 literals in two terms and 3 literals in one term.

- We can implement these sum terms by using two 2-input OR gates and one 3-input OR gate, as shown in the Fig. 4.7.2.

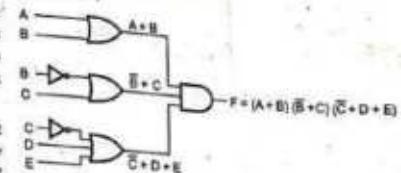


Fig. 4.7.2 Implementation of POS Boolean expression

- Expression tells that these product terms should be ANDed to get the output F.
- We have three sum terms so we have to use 3-input AND gate to obtain the product of sums.
- Literals are complemented using NOT gates.
- In the previous examples we have seen that simplified SOP Boolean expression can be implemented using AND-OR gates.
- The AND-OR implementation is a two level implementation. In the first level we implement all product terms using AND gates and in the second level all product terms are logically ORed using OR gate.
- In case of POS expression we use OR-AND implementation. Here, we implement all sum terms using OR gates in the first level and all sum terms are logically ANDed using AND gate, to get product of sum, in the second level.
- The logic gates are available in the Integrated Circuit (IC) packages. When we implement logic circuit using basic gates, we require ICs for AND, OR and NOT gates.
- Many times it may happen that all gates from the IC packages are not required to build the circuit and thus remaining gates are unused.
- Consider a combinational circuit which requires two 2-input AND gates and one 2-input OR gate as shown in Fig. 4.7.3.
- To implement such a circuit we require IC 7408 (Four 2-input AND gates) and IC 7432 (Four 2-input OR gate).

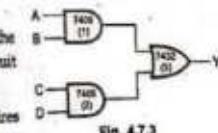


Fig. 4.7.3

- When we use these two ICs we find that two 2-input AND gates are unused and three 2-input OR gates are unused. Thus, the utility factor is very poor.
- The utility factor can be increased by using universal gates to implement logic functions.

**Example 4.7.1** For the following function implement the SOP and POS circuit.

$$F(A, B, C, D) = \Sigma m(2, 3, 5, 7, 12) + \Sigma d(6, 13, 14, 15) \quad \text{GTU : Dec-11, Marks 10}$$

Solution :

$$F = AB + BD + \overline{AC}$$

	CD	AB	00	01	11	10
			1	1		
				1	1	X
			1	X	X	X

Implement using SOP circuit



Fig. 4.7.4

Implementation using POS circuit

$$F = \overline{F} = \overline{AB + BD + \overline{AC}} = \overline{\overline{AB} \cdot \overline{BD} \cdot \overline{\overline{AC}}} = (\overline{A} + \overline{B}) \cdot (\overline{B} + \overline{D}) \cdot (\overline{A} + \overline{C})$$

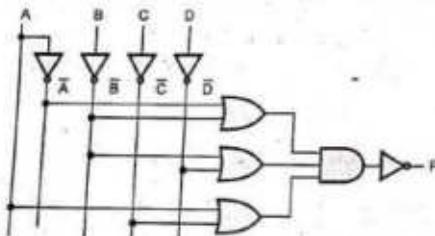


Fig. 4.7.5

**Example 4.7.2** Given Boolean function  $F = xy + \overline{xy} + \overline{yz}$ .

- Implement it with only OR and NOT gates.
- Implement it with only AND and NOT gates.

GTU : May-12, Marks 5

Solution : 1) The given Boolean function is

$$\begin{aligned} \overline{F} &= \overline{xy + \overline{xy} + \overline{yz}} \\ &= \overline{(xy) \cdot (\overline{xy}) \cdot (\overline{yz})} \\ &= \overline{(\overline{x} + \overline{y}) \cdot (x + y) \cdot (y + z)} \\ &= (\overline{x} + \overline{y}) + (x + y) + (y + z) \end{aligned}$$

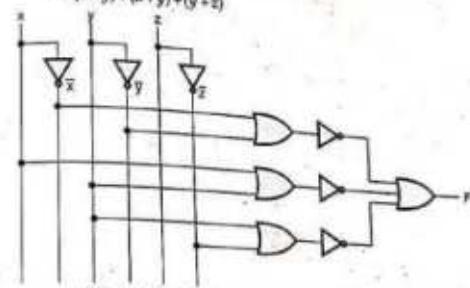


Fig. 4.7.6 Logic diagram using OR and NOT gates

2)

$$\begin{aligned} F &= xy + \overline{xy} + \overline{yz} \\ &= xy + \overline{xy} + \overline{yz} = (xy) \cdot (\overline{xy}) \cdot (\overline{yz}) \end{aligned}$$

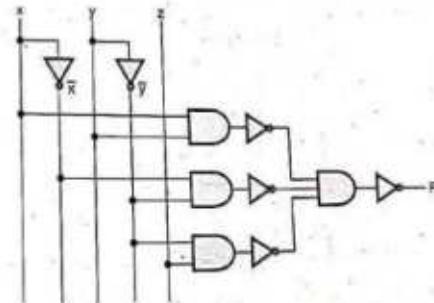


Fig. 4.7.7 Logic diagram using AND and NOT gates

**Example 4.7.3** Implement the Boolean functions a)  $xyz + \bar{x}y + xy\bar{z}$ , b)  $(A+B)(\bar{A}+B)$   
c)  $F = xy + x\bar{y} + y\bar{z}$  with logic gates.

GTU : May-13, Marks 7

**Solution :** a)  $xyz + \bar{x}y + xy\bar{z}$

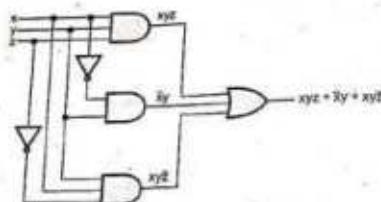


Fig. 4.7.8

b)  $(A+B)(\bar{A}+B)$

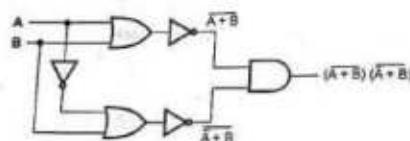


Fig. 4.7.9

c)  $F = xy + x\bar{y} + y\bar{z}$

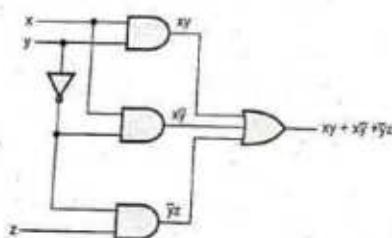


Fig. 4.7.10

**Example 4.7.4** Reduce the given function using K-map and implement the same using gates.  
 $F(A, B, C, D) = \sum m(0, 1, 3, 7, 11, 15) + \sum d(2, 4)$

GTU : Summer-13, Marks 7

**Solution :**

CD	00	01	11	10
AB	00	1	1	X
	01	X	1	
	11		1	
	10			1

Fig. 4.7.11

**Implementation**

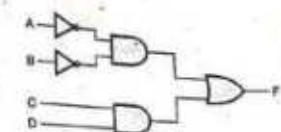


Fig. 4.7.12

**Example 4.7.5** Solve the following Boolean by using K-map. Implement the simplified function by using logic gates.  $F = (w,x,y,z) = \sum m(0,1,4,5,6,8,9,10,11,13,14)$

GTU : Winter-13, Marks 7

**Solution :**

wz	00	01	11	10
w	00	1	1	X
	01	1	1	1
	11	1	1	1
	10	1	1	1

$$F = \bar{y} + x\bar{z} + w\bar{z}$$

**Implementation**

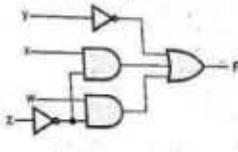


Fig. 4.7.13

**Definition :** A state machine is a device which can be in one of a set number of stable conditions depending on its previous condition and on the present values of its inputs.

**Examples for Practice**

**Example 4.7.6 :** Minimize the following function using K-map and implement using logic gates.

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

[Ans. :  $\bar{A}D + \bar{B}C + \bar{B}D + \bar{A}\bar{B} + ABC\bar{D}$ ]

**Example 4.7.7 :** Reduce the following function using K-map technique and implement using gates :

$$F(A, B, C, D) = \sum m(0, 1, 4, 5, 9, 10) + d(2, 11)$$

[Ans. :  $\bar{B}\bar{C} + \bar{A}\bar{C}D + A\bar{B}$ ]

**Example 4.7.8 :** Minimize the following expression using K-map and implement it using basic gates.

$$i) F(A, B, C, D) = \sum m(1, 4, 8, 12, 13, 15) + d(3, 14)$$

$$ii) F(A, B, C, D) = \sum m(1, 7, 9, 10, 11, 15) + d(2, 3, 5)$$

[Ans. : i)  $\bar{A}\bar{B}D + \bar{B}CD + \bar{A}CD + AB$ , ii)  $\bar{B}C + \bar{B}D + CD$ ]

**Example 4.7.9 :** For the following function implement the SOP and POS circuit.

$$F(A, B, C, D) = \sum m(2, 3, 5, 7) + \sum d(6, 13, 14, 15)$$

GTU : Dec.-12, Marks 10

[Ans. :  $F = BD + \bar{A}C$  and  $\bar{F} = (\bar{A})(C + D)(A + B + C)$ ]

**4.7.3 Universal Gates**

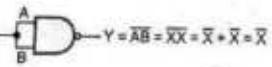
- The NAND and NOR gates are known as universal gates, since any logic function can be implemented using NAND or NOR gates.

**4.7.3.1 NAND Gate**

- The NAND gate can be used to generate the NOT function, the AND function, the OR function, and the NOR function.

**NOT Function :**

An inverter can be made from a NAND gate by connecting all of the inputs together and creating, in effect, a single common input, as shown in Fig. 4.7.14, for a two-input gate.



A	B	AB
0	0	1
0	1	0
1	0	0
1	1	0

Fig. 4.7.14 NOT function using NAND gate

**AND Function :**

An AND function can be generated using only NAND gates. It is generated by simply inverting output of NAND gate; i.e.  $\bar{A}\bar{B} = AB$ . Fig. 4.7.15 shows the two input AND gate using NAND gates.

A	B	AB	AB-bar
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

Table 4.7.1 Truth table

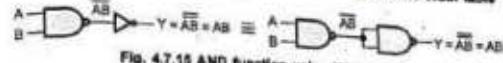


Fig. 4.7.15 AND function using NAND gates

**OR Function :**

OR function is generated using only NAND gates as follows : We know that Boolean expression for OR gate is

$$\begin{aligned} Y &= A + B \\ &= \bar{A} + \bar{B} \\ &= A \cdot \bar{B} \end{aligned}$$

$\bar{A} = A$

DeMorgan's theorem 1

The above equation is implemented using only NAND gates as shown in the Fig. 4.7.16.

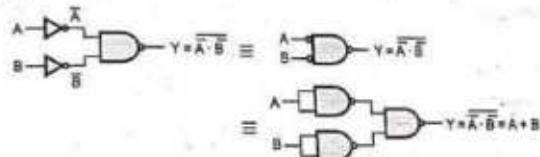


Fig. 4.7.16 OR function using only NAND gates

Note: Bubble at the input of NAND gate indicates inverted input.

A	B	A + B	A + B-bar
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

Table 4.7.2 Truth table

**NOR Function :**

NOR function is generated using only NAND gates as follows : We know that Boolean expression for NOR gate is

$$\begin{aligned} Y &= \overline{A+B} = \overline{\overline{A} \cdot \overline{B}} \\ &= A \cdot B \end{aligned} \quad \text{DeMorgan's theorem 2}$$

The above equation is implemented using only NAND gates, as shown in Fig. 4.7.17.

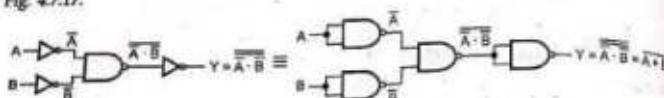


Fig. 4.7.17 NOR function using only NAND gates

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

A	B	$\overline{A} \cdot \overline{B}$	$\overline{A+B}$	$\overline{A \cdot B}$
0	0	1	0	1
0	1	0	1	0
1	0	0	1	0
1	1	0	1	0

**4.7.3.2 NOR Gate**

- The NOR gate is also a universal gate, since it can be used to generate the NOT, AND, OR and NAND functions.

**NOT Function :**

An inverter can be made from a NOR gate by connecting all of the inputs together and creating, in effect, a single common input, as shown in Fig. 4.7.18.

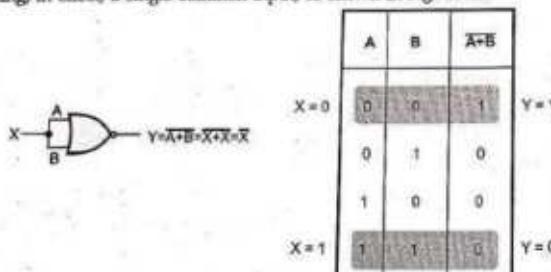


Fig. 4.7.18 NOT function using NOR gate

TECHNICAL PUBLICATIONS™ - An up thrust for knowledge

**OR Function :**

An OR function can be generated using only NOR gates. It can be generated by simply inverting output of NOR gate; i.e.  $\overline{A+B} = A+B$ . Fig. 4.7.19 shows the two input OR gate using NOR gates.



Fig. 4.7.19 OR function using NOR gates

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

Table 4.7.3 Truth table

**AND Function :**

AND function is generated using only NOR gates as follows : We know that Boolean expression for AND gate is

$$\begin{aligned} Y &= A \cdot B \\ &= \overline{\overline{A} \cdot \overline{B}} \\ &= \overline{\overline{A} + \overline{B}} \end{aligned} \quad \text{De-Morgan's theorem 2}$$

The above equation is implemented using only NOR gates as shown in the Fig. 4.7.20.

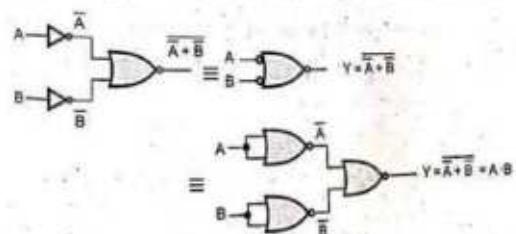


Fig. 4.7.20 AND function using NOR gates

TECHNICAL PUBLICATIONS™ - An up thrust for knowledge

**NOTE:** Bubbling at the input of NOR gate indicates inverted input.

A	B	$A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1

A	B	$\bar{A} + \bar{B}$	$\bar{A} \cdot \bar{B}$
0	0	1	0
0	1	1	0
1	0	1	0
1	1	0	1

Table 4.7.4 Truth table

**NAND Function :**

NAND function is generated using only NOR gates as follows : We know that Boolean expression for NAND gate is

$$\begin{aligned} Y &= \bar{A} \cdot \bar{B} \\ &= \bar{A} + \bar{B} \quad \text{DeMorgan's theorem} \\ &= \overline{\bar{A} + \bar{B}} \quad [\bar{\bar{A}} = A] \end{aligned}$$

The above equation is implemented using only NOR gates, as shown in Fig. 4.7.21.

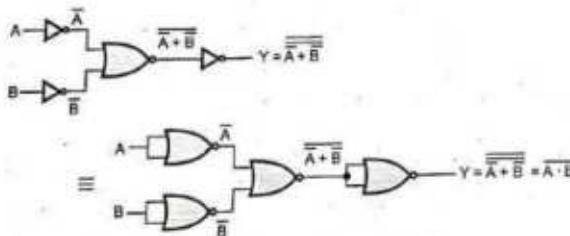


Fig. 4.7.21 NAND function using only NOR gates

A	B	$A \cdot B$
0	0	1
0	1	1
1	0	1
1	1	0

Table 4.7.5 Truth table

**4.7.4 NAND-NAND Implementation**

- The implementation of a Boolean function with NAND-NAND logic requires that the function be simplified in the sum of product form.
- The relationship between AND-OR logic and NAND-NAND logic is explained using following example.
- Consider the Boolean function :  $Y = ABC + DE + F$ .
- This Boolean function can be implemented using AND-OR logic, as shown in Fig. 4.7.22 (a).

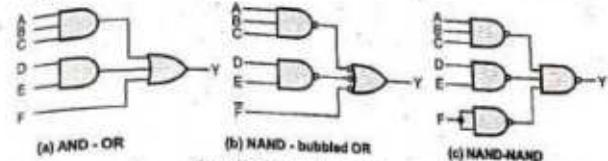


Fig. 4.7.22 NAND-NAND implementation

- Fig. 4.7.22 (b) shows the AND gates are replaced by NAND gates and the OR gate is replaced by a bubbled OR gate. The implementation shown in Fig. 4.7.22 (b) is equivalent to implementation shown in Fig. 4.7.22 (a), because two bubbles on the same line represent double inversion (complementation) which is equivalent to having no bubble on the line.
- In case of single variable, F, the complemented variable is again complemented by bubble to produce the normal value of F.
- In Fig. 4.7.22 (c), the output NAND gate is redrawn with the conventional symbol. The NAND gate with same inputs gives complemented result, therefore  $\bar{F}$  is replaced by NAND gate with F input to its both inputs. Thus all the three implementations of Boolean function are equivalent.

**Rules for obtaining the NAND-NAND logic diagram**

- Simplify the given Boolean function and express it in sum of product form (SOP form).
- Draw a NAND gate for each product term of the function that has two or more literals. The inputs to each NAND gate are the literals of the term. This constitutes a group of first level gates.
- If Boolean function includes any single literal or literals draw NAND gate for each single literal and connect corresponding literal as an input to the NAND gate.

4. Draw a single NAND gate in the second level, with inputs coming from outputs of first level gates.

#### Illustrative Examples

**Example 4.7.10** Implement the following Boolean function with NAND-NAND logic  

$$Y = A'C + ABC + \bar{A}BC + AB + D$$

Solution : Step 1 : Simplify the given Boolean function.

$$\begin{aligned} Y &= A'C + A'BC + \bar{A}BC + AB + D \\ &= A'C + BC(A + \bar{A}) + AB + D = A'C + BC + AB + D \end{aligned}$$

Step 2 : Implement using AND-OR logic. Step 3 : Convert AND-OR logic to NAND-NAND logic.

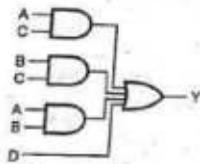


Fig. 4.7.23 (a)

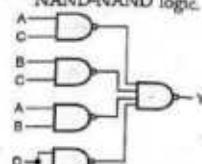


Fig. 4.7.23 (b)

**Example 4.7.11** Implement the following Boolean function with NAND-NAND logic  

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

Solution :

Step 1 : Implement Boolean function with AND-OR logic.

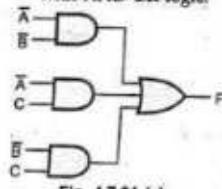


Fig. 4.7.24 (a)

Step 2 : Convert AND-OR logic to NAND-NAND logic.

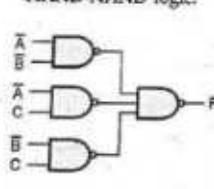


Fig. 4.7.24 (b)

Note : It is possible to directly go to step 2 skipping step 1. Here, step 1 is included for clear understanding.

**Example 4.7.12** Implement the following Boolean function with NAND-NAND logic.  

$$F = (A, B, C) = \sum m(0, 1, 3, 5)$$

Solution : Step 1 : Simplify the given Boolean function.

A	B	C	00	01	11	10
0	1	1	1	1	1	0
1	0	1	0	1	0	0

Fig. 4.7.25

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

Step 2 : Implement Boolean function with AND-OR logic.

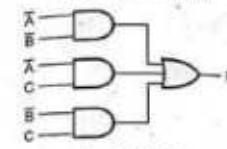


Fig. 4.7.26 (a)

Step 3 : Convert AND-OR logic to NAND-NAND logic.



Fig. 4.7.26 (b)

Note : It is possible to directly go to step 3 skipping step 2. Here, step 2 is included for clear understanding.

**Example 4.7.13** Implement EX-OR gate using only NAND gates.

Solution : The Boolean expression for EX-OR gate is :  $Y = A\bar{B} + \bar{A}B$

We can implement AND-OR logic by using NAND-NAND logic as shown in Fig. 4.7.27 (b).



Fig. 4.7.27 (a)

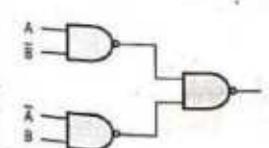


Fig. 4.7.27 (b)

**Example 4.7.14** Implement EX-NOR gate using only NAND gates.

**Solution :** The Boolean expression for EX-NOR gate is  $y = AB + \bar{A}\bar{B}$ . We can implement AND-OR logic by using NAND-NAND logic as shown in Fig. 4.7.28.

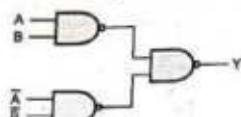


Fig. 4.7.28

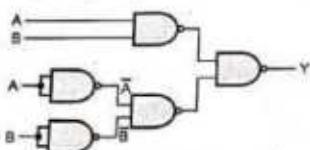
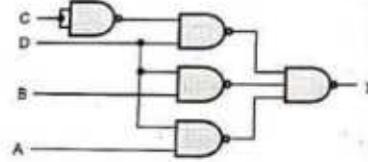
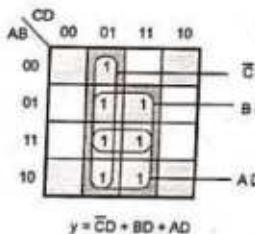


Fig. 4.7.28 (a)

**Example 4.7.15** Minimize  $y = \sum m(1, 5, 7, 9, 11, 13, 15)$  and realize using universal gates.

**Solution :**



$$y = \overline{CD} + \overline{BD} + \overline{AD}$$

**Example 4.7.16** Draw the logic circuit for following function using only NAND gates:  
 $F = ABC + \bar{A}B + \bar{A}CD$ .

GTU : Dec.-13, Marks 2

**Solution :**

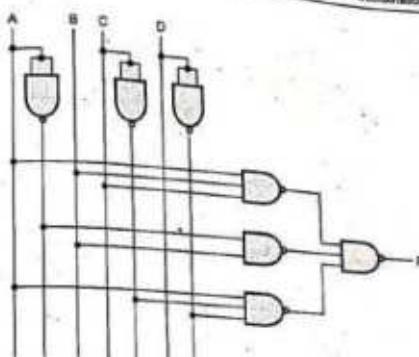
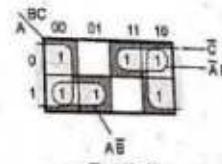


Fig. 4.7.29

**Example 4.7.17** Reduce the expression  $F = \sum m(0, 2, 3, 4, 5, 6)$  using K-map and implement using NAND gates only.  
 GTU : Summary 16, Marks 2

**Solution :**



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

Fig. 4.7.30

We can implement AND-OR logic with NAND-NAND logic.

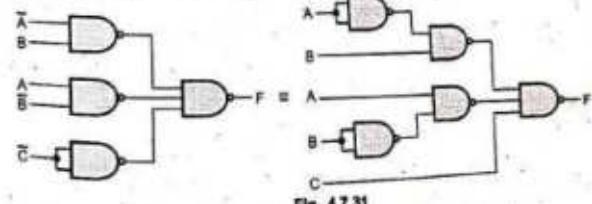


Fig. 4.7.31

**Note :** For C input two inversions get cancelled.

## Examples for Practice

**Example 4.7.18 :** Simplify the following Boolean function using K-map and implement the same using only NAND gates :  
 $F(A, B, C, D) = \overline{D} + \overline{ABC} + ABC + \overline{AB}\overline{CD}$

$$[\text{Ans. : } Y = D(\overline{B}C)(\overline{A}C)]$$

**Example 4.7.19 :** Minimize the following equation using K-map and realise it using NAND gates only :  
 $Y = \sum m(4, 5, 8, 9, 11, 12, 13, 15)$

$$[\text{Ans. : } Y = \overline{BC} + \overline{AB} + \overline{AC}]$$

**Example 4.7.20 :** Minimize the given function using K-map and implement using NAND gates only.

i)  $F(A, B, C, D) = \sum m(1, 3, 7, 11, 15) + d(0, 2, 5, 14)$

ii)  $F(A, B, C, D) = \sum m(0, 1, 2, 3, 6, 7, 13, 15)$

$$[\text{Ans. : i) } \overline{A}\overline{B} + CD, \text{ ii) } \overline{A}\overline{B} + \overline{AC} + ABD]$$

## 4.7.5 NOR-NOR Implementation

- The NOR function is a dual of the NAND function. For this reason, the implementation procedures and rules for NOR-NOR logic are the duals of the corresponding procedures and rules developed for NAND-NAND logic.
- The implementation of a Boolean function with NOR-NOR logic requires that the function be simplified in the product of sum form.
- In product of sum form, we implement all sum terms using OR gates. This constitutes the first level.
- In the second level all sum terms are logically ANDed using AND gate. The relationship between OR-AND logic and NOR-NOR is explained using following example.
- Consider the Boolean function :  $Y = (A + B + C)(D + E)F$
- This Boolean function can be implemented using OR-AND logic, as shown in the Fig. 4.7.32 (a).
- Fig. 4.7.32 (b) shows the OR gates are replaced by NOR gates and the AND gate is replaced by a bubbled AND gate. The implementation shown in Fig. 4.7.32 (b)

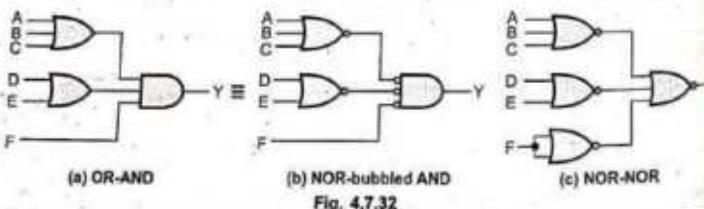


Fig. 4.7.32

is equivalent to implementation shown in Fig. 4.7.32 (a), because two bubbles on the same line represent double inversion (complementation) which is equivalent to having no bubble on the line.

- In case of single variable, F, the complemented variable again complemented by bubble to produce the normal value of F.
- In Fig. 4.7.32 (c), the output NOR gate is redrawn with the conventional symbol replaced by NOR gate with F input to its both inputs. Thus all the three implementations of Boolean function are equivalent.

## Rules for obtaining the NOR-NOR logic diagram

- Simplify the given Boolean function and express it in product of sum form (POS form).
- Draw a NOR gate for each sum term of the function that has two or more literals. The inputs to each NOR gate are the literals of the term. This constitute a group of first level gates.
- If Boolean function includes any single literal or literals, draw NOR gate for each single literal and connect corresponding literal as an input to the NOR gate.
- Draw a single NOR gate in the second level, with inputs coming from outputs of first level gates.

## Illustrative Examples

**Example 4.7.21** Implement the following Boolean function with NOR-NOR logic

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

Solution : Step 1 : Express Boolean function in POS form.

Using duality theorem we get,

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

Step 2 : Implement Boolean function with OR-AND logic.

Step 3 : Convert OR-AND logic to NOR-NOR logic.

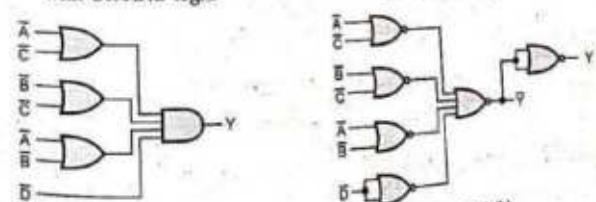


Fig. 4.7.33 (a)

**Example 4.7.22** Implement the following Boolean function with NOR-NOR logic.  
 $F = (A, B, C) = \Sigma m(0, 2, 4, 5, 6)$

**Solution :** Step 1 : Simplify the given Boolean function:

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

A	B	C	00	01	11	10
0	0	0	0	0	0	0
1	0	0	0	0	0	0

Fig. 4.7.34

Step 2 : Implement Boolean function with OR-AND logic.

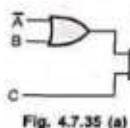


Fig. 4.7.35 (a)

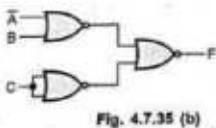


Fig. 4.7.35 (b)

**Note** It is possible to directly go to step 3 skipping step 2. Here, step 2 is included for clear understanding.

**Example 4.7.23** Implement EX-NOR gate using only NOR gates.

**Solution :** The Boolean expression for EX-NOR gate is :

$$\begin{aligned} Y &= AB + \bar{A}\bar{B} = \overline{\bar{A}\bar{B}} \\ &= \overline{AB} \cdot \overline{\bar{A}\bar{B}} = (\bar{A} + B) \cdot (\bar{A} + \bar{B}) \end{aligned}$$

We can implement OR-AND logic by using NOR-NOR logic, as shown in Fig. 4.7.36 (b).

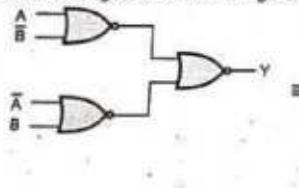


Fig. 4.7.36 (a)

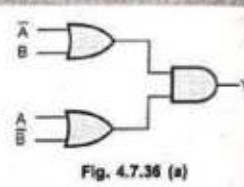


Fig. 4.7.36 (b)

**Example 4.7.24** Implement EX-OR gate using only NOR gates.

**Solution :** Boolean expression of EX-OR gate  $Y = A\bar{B} + \bar{A}B = \overline{AB} + \overline{\bar{A}\bar{B}}$

$$\begin{aligned} &= \overline{AB} \cdot \overline{\bar{A}\bar{B}} \\ &= (\bar{A} + \bar{B})(A + B) \end{aligned}$$

**Note** We can implement OR-AND logic by NOR-NOR logic

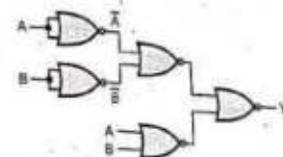
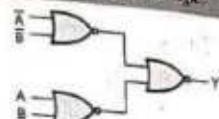


Fig. 4.7.37

**Example 4.7.25** For the following function implement the logic circuit using (i) Only NOR gates (ii) Only NAND gates.

$$F(A, B, C, D) = \Sigma m(1, 3, 4, 6, 9, 11, 12, 14) + \Sigma d(2, 5, 8, 15)$$

**Solution :** i)

CD	00	01	11	10	
AB	00	1	1	X	$\overline{ED}$
AB	01	1	X	1	$\overline{ED}$
AB	11	1	X	1	
AB	10	X	1	1	

$$\therefore F = B\overline{D} + \overline{ED}$$

Implementation using NAND gate

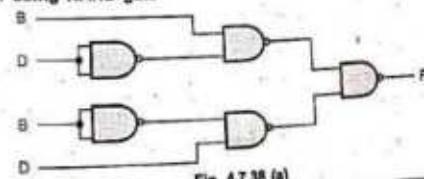


Fig. 4.7.38 (a)

ii) We have,

$$F = BD + \bar{BD}$$

$$\bar{F} = \overline{BD + \bar{BD}}$$

$$\bar{F} = \overline{BD \cdot \bar{BD}} = \overline{(B+D) \cdot (\bar{B}+\bar{D})}$$

Implementation using NOR gates

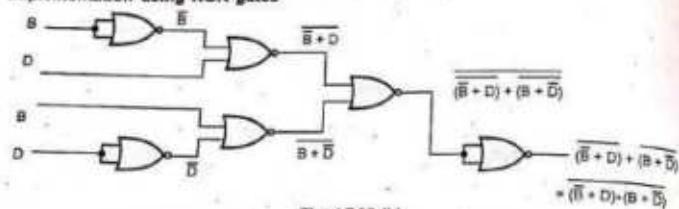


Fig. 4.7.38 (b)

**Example 4.7.26** Draw the logic circuit for following function using only NOR gates

$$F = ABC + AB(C+D)$$

GTU : Dec.-13, Marks 4

Solution :  $F = ABC + AB(C+D)$   
 $= ABC + ABC + ABD$

Express the function  $F$  in POS form. Using duality theorem, we get,

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

Logic diagram

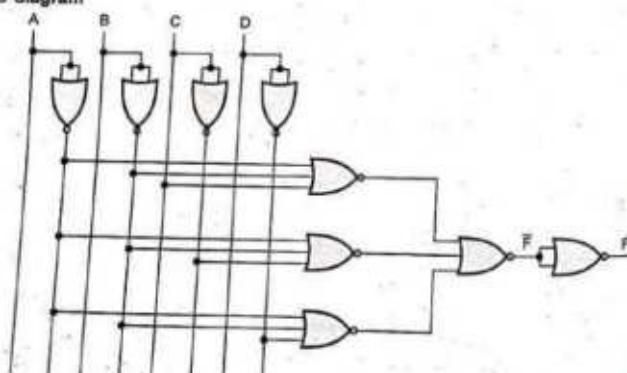


Fig. 4.7.39

TECHNICAL PUBLICATIONS™ - An up beat for knowledge

**Example 4.7.27** Minimize the following logic function

$$F(A, B, C, D) = \sum m(1, 2, 3, 8, 9, 10, 11, 14) + \sum d(15)$$

Use Karnaugh map. Draw the logic circuit for the simplified function using NOR gates only.

Solution :

K-map Simplification

AB	C+D	C+D̄	D+D̄	D̄+D
A+B	0	0	0	0
A+B̄		X		
A+B̄			X	0
A+B	0	0	0	0

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

Fig. 4.7.40

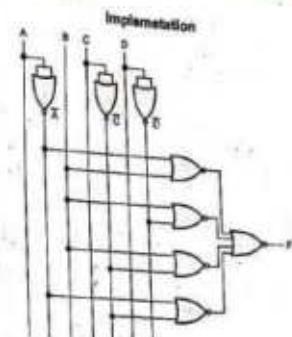


Fig. 4.7.41

**Example 4.7.28** Reduce following Boolean function and then realize the reduced one using NOR gate only.  $X = A(B' + C')(A + D)$

GTU : Winter-15, Marks 4

Solution : Implementation using NOR gates

CD	C+D	C+D̄	D+D̄	D̄+D
AB	0	0	0	0
A+B	0	0	0	0
A+B̄		X		
A+B̄			X	0
A+B	0	0	0	0

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

Note : POS function can be implemented by NOR-NOR logic

Fig. 4.7.42

TECHNICAL PUBLICATIONS™ - An up beat for knowledge

**Example 4.7.29** Implement the following function with NAND and NOR Gate.

$$F(a, b, c) = \sum(0, 6)$$

$$\text{Solution : } F(a, b, c) = \sum(0, 6) = \bar{a}\bar{b}\bar{c} + a\bar{b}\bar{c}$$

Implementation using NAND gates

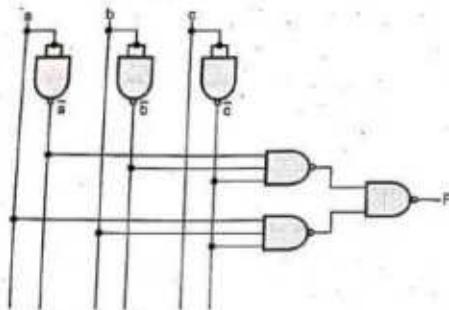


Fig. 4.7.43

Note : AND -OR logic can be directly implemented by NAND-NAND logic.

Implementation using NOR gates

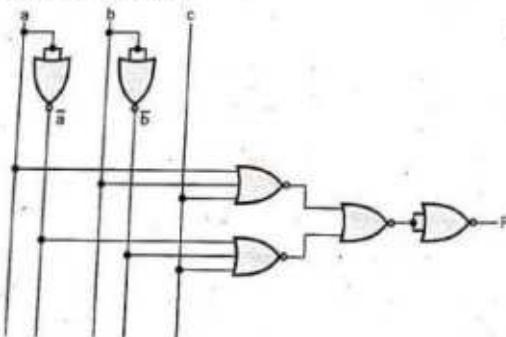


Fig. 4.7.44

$$\begin{aligned} F &= \bar{a}\bar{b}\bar{c} + a\bar{b}\bar{c} \\ &= \overline{\bar{a}\bar{b}\bar{c}} + \overline{a\bar{b}\bar{c}} \\ &= \overline{(a+b+c)}(\overline{a+b+c}) \\ &= (a+b+c)(\bar{a}+\bar{b}+\bar{c}) \end{aligned}$$

**Example 4.7.30** Minimize the following logic function using K-maps and realize using NAND and NOR gates

$$F(A, B, C, D) = \sum m(1, 3, 5, 8, 9, 11, 15) + d(2, 13)$$

Solution :

$$\begin{aligned} F(A, B, C, D) &= \sum m(1, 3, 5, 8, 9, 11, 15) + d(2, 13) \\ &= \pi M(0, 4, 6, 7, 10, 12, 14) + d(2, 13) \end{aligned}$$

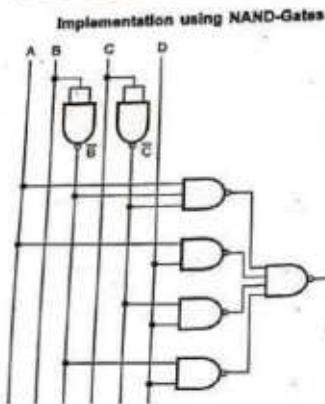
AB	CD	00	01	11	10
00		1	1	X	
01			1		
11		X	1		
10		1	1	1	

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

AB	CD	C+D	D+C	C+D
A+B		0		X
A+B		0	0	0
A+B		0	X	0
A+B				0

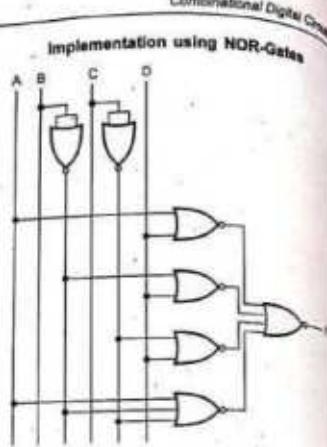
$$F = (A+D)(B+D)(\bar{C}+D)(A+B+\bar{D})$$

Fig. 4.7.45



Note : SOP function can be implemented using NAND - NAND logic

Fig. 4.7.46



Note : POS function can be implemented using NOR - NOR logic

Fig. 4.7.47

#### 4.7.6 The EX-OR Gate in Function Realization

- In Ex-OR gate, the output goes high only if ODD number of inputs are high.
- Ex-OR gate can be used to implement the boolean functions presented in following truth tables and K-maps.
- Two-input Ex-OR gate :

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

Truth table

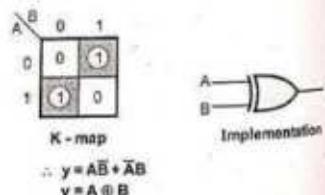


Fig. 4.7.48

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

$Y = A \oplus B$

Three input Ex-OR gate :

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

Truth table

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

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

$$= A \oplus B \oplus C$$

Four input Ex-OR gate :

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

Table 4.7.8 Truth table

**K-map simplification**

$$\begin{aligned}
 Y &= \bar{A}\bar{B}(\bar{C}D + C\bar{D}) + \bar{A}B(\bar{C}\bar{D} + CD) \\
 &\quad + A\bar{B}(\bar{C}D + C\bar{D}) + AB(\bar{C}\bar{D} + CD) \\
 &= \bar{A}\bar{B}(C\oplus D) + \bar{A}B(C\oplus D) \\
 &\quad + A\bar{B}(C\oplus D) + AB(C\oplus D) \\
 &= (\bar{A}\bar{B} + A\bar{B})(C\oplus D) + (\bar{A}B + AB)(C\oplus D) \\
 &= (\bar{A}\oplus B)(C\oplus D) + (A\oplus B)(C\oplus D) \\
 &= (A\oplus B)\oplus(C\oplus D)
 \end{aligned}$$

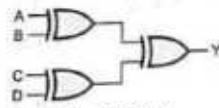
**Logic diagram**

Fig. 4.7.50 (b)

**Review Questions**

- Define the following term : Universal gate.
  - Why NAND and NOR gates are called universal gates.
  - Using NOR gates draw inverters, AND and OR gates.
  - Discuss NAND gate as universal gate (implement NOT, AND, OR and NOR gate using NAND gate).
  - Implement NOT, AND and OR gates NAND gates only.
- GTU : Dec-13, Marks 1  
Summer-15, Marks 1  
GTU : Summer-18, Marks 1

**4.8 Combinational Design Examples**

GTU : Winter-14, II

**4.8.1 Introduction**

- When logic gates are connected together to produce a specified output for certain specified combinations of input variables, with no storage involved, the resulting circuit is called **combinational logic circuit**.
- In combinational logic circuit, the output variables are at all times dependent on the combination of input variables.

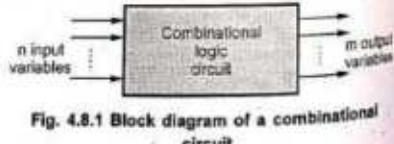


Fig. 4.8.1 Block diagram of a combinational circuit

- Fig. 4.8.1 shows the block diagram of a combinational circuit.

**4.8.2 Design Procedure**

The design of combinational circuits starts from the outline of the problem statement and ends in a logic circuit diagram or a set of Boolean functions from which the logic diagram can be easily obtained. The design procedure of the combinational circuit involves following steps :

1. The problem definition.
2. The determination of number of available input variables and required output variables.
3. Assigning letter symbols to input and output variables.
4. The derivation of truth table indicating the relationships between input and output variables.
5. Obtain simplified Boolean expression for each output.
6. Obtain the logic diagram.

**4.8.3 A Combinational Function Generator**

- Here, the combinational circuit is represented by the relationship between its input and an output variable or variables. The following examples illustrate the procedure to design such combinational circuits.

**Example 4.8.1** Design a combinational logic circuit with three input variables that will produce a logic 1 output when more than one input variable are logic 1.

Solution :

Step 1 : Derive the truth table for given statement.

Given problem specifies that there are three input variables and one output variable. We assign A, B and C letter symbols to three input variables and assign Y letter symbol to one output variable. The relationship between input variables and output variable can be tabulated as shown in truth Table 4.8.1.

Step 2 : Obtain simplified Boolean expression.

Now we obtain the simplified Boolean expression for output variable Y using K-map simplification.

$$Y = AC + BC + AB$$

A	B	C	Y
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

Table 4.8.1 Truth table

BC	00	01	11	10
A	0	0	1	0
	0	1	1	1

Fig. 4.8.2 K-map simplification

## Combinational Digital Circuits

**Step 3 :** Draw logic diagram.

In this chapter we are going to study various combinational circuits using above illustrated design method.



Fig. 4.8.3 Logic diagram

**Example 4.8.2** Design a circuit that has one control line C and three data lines  $D_1, D_2, D_3$ . When the control line is high, the circuit is to detect when one of the data lines has a 1 on it. No more than one data line will ever have a 1 on it. When the control line is low, the circuit will output a 0, regardless of what is on the data lines.

**Solution :** The truth table for the given problem is as shown below.

C	$D_3$	$D_2$	$D_1$	Output
0	x	x	x	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	1	0	0	1

Table 4.8.2 Truth Table for given problem

$$Y = CD_3 + CD_2 + CD_1$$

## K-map Simplification

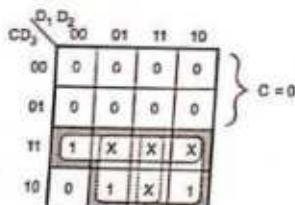


Fig. 4.8.4

## Logic diagram

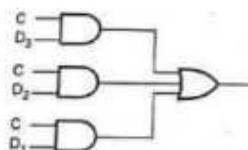


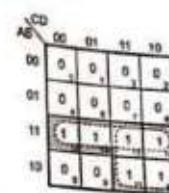
Fig. 4.8.5

**Example 4.8.3** Design circuit to detect invalid BCD number and implement using NAND gate only.

## Solution : Truth table

Dec	A	B	C	D	Output Y
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	1
12	1	1	0	0	1
13	1	1	0	1	1
14	1	1	1	0	1
15	1	1	1	1	1

## K-map Simplification



$$Y = AB + AC$$

## Logic diagram

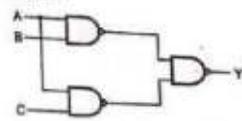


Fig. 4.8.6

AND-OR logic can be directly implemented by NAND-NAND logic.

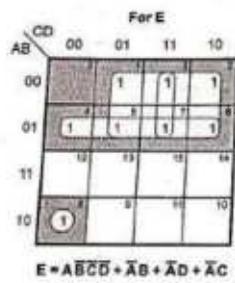
**Example 4.8.4** Design a combinational circuit to produce the 2's complement of a 4-bit binary number as a input.

**Solution :**

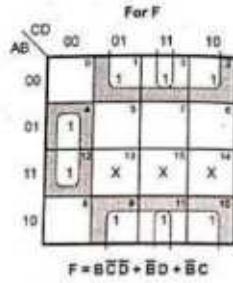
Input	2's complement output							
	A	B	C	D	E	F	G	H
0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	1
0	0	0	1	0	1	1	1	0
0	0	1	1	1	1	1	0	1
0	1	0	0	1	1	1	0	1
0	1	0	1	1	0	1	1	0
0	1	1	0	1	0	1	0	1

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

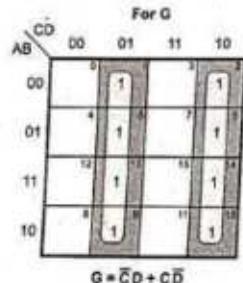
## K-map simplification



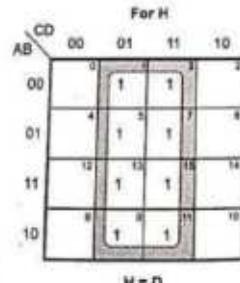
$$E = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B} + \bar{A}\bar{D} + \bar{A}\bar{C}$$



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



$$G = \bar{C}\bar{D} + \bar{C}\bar{D}$$



$$H = D$$

## Logic diagram

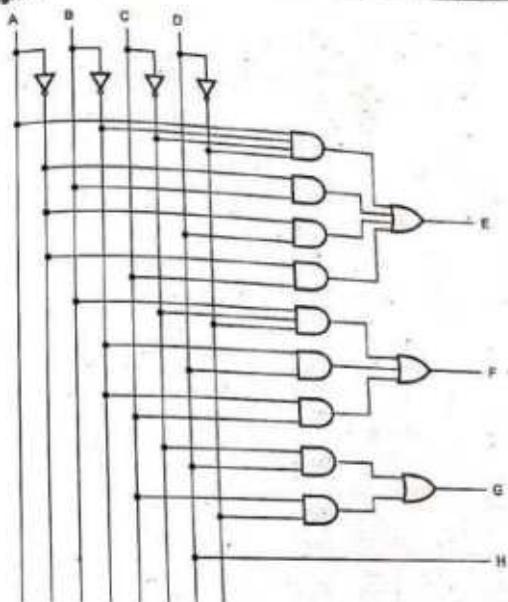


Fig. 4.8.7

**Example 4.8.5** A combinational circuit has 3 inputs A, B, C and output E. E is true for following input combinations

A is False, B is True

A is False, C is True

A, B, C are False

A, B, C are True

i) Write the truth table for E. Use the convention True = 1 and False = 0.

ii) Write the simplified expression for E in SOP form.

iii) Write the simplified expression for E in POS form.

iv) Draw logic circuit using minimum number of 2-input NAND gates.

GTU : Winter-14, Marks 7

**Solution :**

Truth table

A	B	C	F
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

Truth table

A	BC	00	01	11	10
0	1	1	1	1	1
1	0	0	1	0	0

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

Fig. 4.8.8

A	BC	$B+C$	$B+\overline{C}$	$\overline{B}+C$	$\overline{B}+\overline{C}$
0	1	1	1	1	0
1	0	0	1	0	0

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

- ii) Simplified SOP expression :  $F = \overline{A} + BC$   
 iii) Simplified POS expression :  $F = (\overline{A} + B)(\overline{A} + C)$   
 iv) Implementation used 2-input NAND gates.

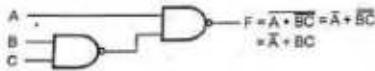


Fig. 4.8.9

**Example 4.8.6** Design a combinational logic circuit output is high only when majority of inputs (A, B, C, D) are low.

GTU : Winter-18, Marks 1

**Solution :** Step 1 : Derive the truth table for given statement

Input				Output
A	B	C	D	Y
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0

TECHNICAL PUBLICATIONS™ - An up beat for knowledge

Step 2 : Obtain simplified Boolean expression.

CD	00	01	11	10
00	1	1	0	1
01	1	0	0	0
11	0	0	0	0
10	1	0	0	0

$$Y = ABC + AC\bar{B} + \bar{A}\bar{B}D + \bar{B}\bar{C}D$$

Fig. 4.8.10

Step 3 : Draw Logic diagram

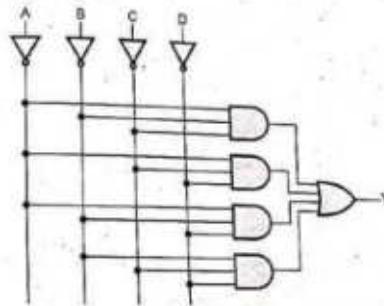


Fig. 4.8.11

## Example for Practice

**Example 4.8.7** : The inputs to a circuit are the 4 bits of the binary number  $D_3D_2D_1D_0$ . The circuit produces a 1 if and only if all of the following conditions hold.  
 1) MSB is '1' or any of the other bits are a '0'.  
 2)  $D_2$  is a 1 or any of the other bits are a '0'.  
 3) Any of the 4 bits are a 0.  
 Obtain a minimal expression for the output. [Ans. :  $Y = \overline{D}_1 + \overline{D}_0 + \overline{D}_3\overline{D}_2$ ]

## Review Question

- I. Explain the design procedure of combinational circuits.

TECHNICAL PUBLICATIONS™ - An up beat for knowledge

#### 4.9 Quine McCluskey Method of Function Realization

In simplification of Boolean expression we observed that the adjacent minterms can be reduced. These minterms are reduced because they differ by only one literal. For example,  $A\bar{B}C$  and  $A\bar{B}\bar{C}$  can be reduced because only the B literal differs. The binary numbers equivalent to these minterms are 110 and 100. Note that the 2's place indicates precisely the same information that the literal represented, namely, that the B is the only literal that differs. Thus we can say that the minterms whose binary equivalent differ only in one place can be combined to reduce the minterms. This is the fundamental principle of the Quine McCluskey method.

Like in other methods, in this method minterms are listed to specify the given function. However, the minterms are written in their binary equivalents. These minterms are grouped according to number of 1s contained and separated by a horizontal line between each number of 1s category, as shown in the Table 4.9.1 (column (b)). This separation of minterms helps in searching the binary minterms that differ only in one place. Once the separation is over, each binary number is compared with every term in the next higher category and if they differ by only one position, a check mark is placed beside each of the two terms and then the term is copied in the second column with a '-' in the position that they differed.

For example, if the terms are 0000 and 0010 then the resultant term will be 0 0 - 0. This term, 00-0 is called an **implicant**, it implies the  $\bar{A}\bar{B}\bar{D}$  could be a term in the final expression, covering minterms  $\bar{A}\bar{B}\bar{C}\bar{D}$  and  $\bar{A}\bar{B}C\bar{D}$ . This process of comparison is repeated for every minterm. Once this process is completed the same process is applied to the new resultant terms which are placed in the Table 4.9.2 column (c). These cycles are continued until a single pass through a cycle yields no further elimination of literals. The remaining terms and all the terms that did not match during the process are called the **prime implicants**. Summing one or more prime implicants gives the simplified Boolean expression.

##### 4.9.1 Algorithm for Generating Prime Implicants

1. List all minterms in the binary form.
2. Arrange the minterms according to number of 1s.
3. Compare each binary number with every term in the adjacent next higher category and if they differ only by one position, put a check mark and copy the term in the next column with '-' in the position that they differed.
4. Apply the same process described in step 3 for the resultant column and continue these cycles until a single pass through cycle yields no further elimination of literals.
5. List all prime implicants.

6. Select the minimum number of prime implicants which must cover all the minterms.  
See the following examples to illustrate the algorithm.

##### Illustrative Examples

**Example 4.9.1** Simplify the following Boolean function by using a Quine McCluskey method.  $F(A, B, C, D) = \sum m(0, 2, 3, 5, 6, 7, 8, 10, 12, 13)$

**Solution :**  
**Step 1 :** List all minterms in the binary form as shown in Table 4.9.1 (column (a)).  
**Step 2 :** Arrange the minterms according to number of 1s, as shown in the Table 4.9.1 (column (b)).

Minterm	Binary representation	Minterm	Binary representation
$m_0$	0 0 0 0	$m_4$	0 0 0 0 ✓
$m_2$	0 0 1 0	$m_5$	0 0 1 0 ✓
$m_3$	0 0 1 1	$m_6$	1 0 0 0 ✓
$m_4$	0 1 1 0	$m_7$	0 0 1 1 ✓
$m_5$	0 1 1 1	$m_8$	0 1 1 0 ✓
$m_6$	1 0 0 0	$m_{10}$	1 0 1 0 ✓
$m_{10}$	1 0 1 0	$m_{12}$	1 1 0 0 ✓
$m_{12}$	1 1 0 0	$m_7$	0 1 1 1 ✓
$m_{13}$	1 1 0 1	$m_{13}$	1 1 0 1 ✓

Table 4.9.1

**Step 3 :** Compare each binary number with every term in the adjacent next higher category and if they differ only by one position, put a check mark and copy the term in the next column with '-' in the position that they differed.

**Step 4 :** Apply the same process described in step 3 for the resultant column and continue these cycles until a single pass through cycle yields no further elimination of literals.

Minterms	Binary representation	Minterms	Binary representation
0, 2	0 0 - 0 ✓	0, 2, 8, 10	- 0 - 0
0, 8	- 0 0 0 ✓	2, 3, 5, 7	0 - 1 -
2, 3	0 0 1 - ✓		
2, 6	0 - 1 0 ✓		
2, 10	- 0 1 0 ✓		
8, 10	1 0 - 0 ✓		
8, 12	1 - 0 0 ✓		
3, 7	0 - 1 1 ✓		

TECHNICAL PUBLICATIONS™ - An up beat for knowledge

6, 7	0 1 1 -	✓
12, 13	1 1 0 -	
		Column (c)

Table 4.9.2

Column (d)

**Step 5 :** List the prime implicants.

Prime implicants	Binary representation
$A\bar{C}D$	0, 12
$A\bar{B}C$	12, 13
$\bar{B}\bar{D}$	0, 2, 8, 10
$\bar{A}C$	2, 3, 6, 7

Table 4.9.3

**Step 6 :** Select the minimum number of prime implicants which must cover all the minterms.

Table 4.9.4 shows prime implicant selection chart. Each prime implicant is represented in a row and each minterm in a column. Dots are placed in each row to show the composition of minterms that make the prime implicants. From this chart we have to select the minimum number of prime implicants which must cover all the minterms. The selection procedure is as follows.

Search for single dot columns and select the prime implicants corresponding to the dot by putting the check mark in front of it.

Prime implicants	$m_0$ (Col 1)	$m_1$ (Col 2)	$m_2$ (Col 3)	$m_3$ (Col 4)	$m_4$ (Col 5)	$m_5$ (Col 6)	$m_6$ (Col 7)	$m_{10}$ (Col 8)	$m_{12}$ (Col 9)
$A\bar{C}D$	0, 12								
$A\bar{B}C$	12, 13 ✓							○	○
$\bar{B}\bar{D}$	0, 2, 8, 10 ✓	○	○			○	○		
$\bar{A}C$	2, 3, 6, 7 ✓	○	○	○	○				

Table 4.9.4 Prime implicant selection chart

Search for multiple dot columns one by one. If the corresponding minterm is already included in the final expression ignore the minterm and go to next multi-dot column; otherwise include the corresponding prime implicant in the final expression.

In our case, column 1 ( $m_0$ ) has single dot, so include corresponding prime implicant (0, 2, 8, 10). Column 3 ( $m_3$ ) has single dot so include corresponding prime implicant (2, 3, 6, 7). Columns 4, 5 and 7 have single dots but the corresponding minterms are already included in the final expression.

Column 9 has single dot so include corresponding prime implicant (12, 13) in the final expression. Now search for multi-dot columns. Columns 2, 6 and 8 have multi-dots but their minterms are already included in the final expression. Therefore, the final expression is

$$F(A, B, C, D) = (1 \ 1 \ 0 \ -) + (- \ 0 \ - \ 0) + (0 \ 1 \ -)$$

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

**Example 4.9.2**

Minimize the expression using Quine-McCluskey method.

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

**Solution :****Step 1 :** List all minterms in the binary form, as shown in Table 4.9.5 (column (a)).**Step 2 :** Arrange minterms according to categories of 1s, as shown in the Table 4.9.5 (column (b)).

Minterm	Binary representation	Minterm	Binary representation
$m_4$	0 1 0 0	$m_2$	0 0 1 0
$m_5$	0 1 0 1	$m_4$	0 1 0 0 ✓
$m_{12}$	1 1 0 0	$m_2$	0 1 0 1 ✓
$m_{13}$	1 1 0 1	$m_9$	1 0 0 1 ✓
$m_7$	1 0 0 1	$m_{12}$	1 1 0 0 ✓
$m_2$	0 0 1 0	$m_{11}$	1 1 0 1 ✓

Column (a)                                    Column (b)

Table 4.9.5

**Step 3 :** Compare each binary number with every term in the next higher category and if they differ by only one position put a check mark and copy the term in the next column with '-' in the position that they differed.

Minterms	Binary representation	Minterms	Binary representation
4, 5	0 1 0 -	✓	4, 5, 12, 13 - 1 0 -
4, 12	- 1 0 0	✓	
5, 13	- 1 0 1	✓	
9, 13	1 - 0 1		
12, 13	1 1 0 -	✓	

Table 4.9.6

**Step 4 :** Apply the same process described in step 3 for the resultant column and continue this cycles until a single pass through cycle yields no further elimination of literals.

**Step 5 :** List the prime implicants.

Prime implicants	Binary representation
$\bar{A} \bar{B} C \bar{D}$	0 0 1 0
$A \bar{C} D$	1 - 0 1
$B \bar{C}$	1 0 -

Table 4.9.7

**Step 6 :** Select the minimum number of prime implicants which must cover all  $S_0$  minterms.

Prime implicants	$m_2$ (Col 1)	$m_4$ (Col 2)	$m_5$ (Col 3)	$m_9$ (Col 4)	$m_{10}$ (Col 7)	$m_{12}$ (Col 5)	$m_{13}$ (Col 9)
$\bar{A} \bar{B} C \bar{D}$	✓	○					
$A \bar{C} D$	✓			○	○	○	
$B \bar{C}$	✓	○	○	○	○	○	

Table 4.9.8 Prime implicant selection chart

The final expression is  $Y = (0 \ 0 \ 1 \ 0) + (1 \ - \ 0 \ 1) + (-1 \ 0 \ -) = \bar{A} \bar{B} C \bar{D} + A \bar{C} D + B \bar{C}$

#### 4.9.2 Quine McCluskey using Don't Care Terms

The same rules that applied to using don't care terms with Karnaugh map are appropriate for Quine McCluskey. Here, don't care terms are never included as prime implicants by themselves. Let us consider the following example.

##### Illustrative Example

**Example 4.9.3** Simplify the following using tabulation methods.

$$Y(w, x, y, z) = \sum m(1, 2, 3, 5, 9, 12, 14, 15) + \sum d(4, 6, 11)$$

**Solution :** It is important to note that don't care conditions are used to find the prime implicant but it is not compulsory to include don't care terms in the final expression.

**Step 1 :** List all minterms in the binary form, as shown in the Table 4.9.9 (Column (a)).

Minterm	Binary representation	Minterm	Binary representation
$m_2$	0 0 0 1	$m_3$	0 0 0 1 ✓
$m_4$	0 0 1 0	$m_5$	0 0 1 0 ✓
$m_9$	0 0 1 1	$m_6$	0 1 0 0 ✓
$m_{10}$	0 1 0 1	$m_7$	1 0 0 0 ✓
$m_{12}$	1 0 0 1	$m_8$	0 0 1 1 ✓
$m_{13}$	1 1 0 0	$m_{14}$	0 1 0 1 ✓
$m_{14}$	1 1 1 0	$m_{15}$	1 0 0 1 ✓
$m_{15}$	1 1 1 1	$m_{16}$	1 1 0 0 ✓
$d_{m_4}$	0 1 0 0	$m_{17}$	1 0 1 1 ✓
$d_{m_5}$	1 0 0 0	$m_{18}$	1 1 1 0 ✓
$d_{m_{12}}$	1 0 1 1	$m_{19}$	1 1 1 1 ✓

Table 4.9.9

**Step 2 :** Arrange the minterms according to categories of 1s as shown in the Table 4.9.9 (Column (b)).

**Step 3 :** Compare each binary number with every term in the adjacent next higher category and if they differ by only one position put a check mark and copy the term in the next column with '-' in the position that they differed.

**Step 4 :** Apply the same process described in step 3 for the resultant column and continue these cycles until a single pass through cycle yields no further elimination of literals.

Minterms	Binary representation	Minterms	Binary representation
1, 3	0 0 - 1	1, 3, 9, 11	- 0 - 1
1, 5	0 - 0 1		
1, 9	- 0 0 1		
2, 3	0 0 1 -		
4, 5	0 1 0 -		

TECHNICAL PUBLICATIONS™ - An up beat for knowledge

4, 12	-	1	0	0
8, 9	1	0	0	-
8, 12	1	-	0	0
3, 11	-	0	1	1
9, 11	1	0	-	1
12, 14	1	1	-	0
13, 15	1	-	1	1
14, 15	1	1	1	-

Table 4.9.10

**Step 5 :** List the prime implicants.

**Step 6 :** Select the minimum number of prime implicants which must cover all the minterms, except don't care minterms.

Only column 2 has single dot i.e. it is essential prime implicant and hence the prime implicant corresponding to it ( $m_{2,3}$ ) is included in the final expression. Now search for multi-dot columns. Column 1 has multi-dot and the corresponding terms are not included in the final expression, so we can include either  $m_{1,5}$  or  $m_{1,3,9,11}$ . Let us include prime implicant which has more minterms. Thus minterm 1, 3, 9, 11 is included. Now minterm for column 3 is already included and minterm for column 4 is don't care. Column 5 has a minterm which is not included yet, therefore prime implicant 1, 5 is included in the final expression. The minterms from column 6 and 8 are don't care and the minterm of column 7 ( $m_9$ ) is already included in the final expression. Column 9 has minterm 12 which is not included yet.

To include this term we have 3 options. We include prime implicant 12, 14 because with this inclusion we can also cover minterm 14 in the final expression. The minterm from the remaining column 11 ( $m_{15}$ ) can be included in the final expression by including prime implicant 14, 15. Therefore, the final expression is

$$\begin{aligned} Y &= (0 - 0 1) + (0 0 1 -) + (1 1 - 0) + (1 1 1 -) + (- 0 - 1) \\ &= \bar{A}\bar{C}D + \bar{A}\bar{B}C + A\bar{B}\bar{D} + ABC + \bar{B}D \end{aligned}$$

Prime implicants	Binary representation
A C D	1, 5
A B C	1, 3
A B C	4, 5
B C D	4, 12
A B C	8, 9
A C D	8, 12
A B D	12, 14
A C D	11, 15
A B C	14, 15
B D	1, 3, 9, 11

Table 4.9.11

Prime implicants	$m_0$ (Col 1)	$m_1$ (Col 2)	$m_2$ (Col 3)	$m_3$ (Col 4)	$m_4$ (Col 5)	$m_5$ (Col 6)	$m_6$ (Col 7)	$m_7$ (Col 8)	$m_8$ (Col 9)	$m_9$ (Col 10)	$m_{10}$ (Col 11)
A C D	1, 5	○			○						
A B C	1, 3	○	○								
A B C	4, 5			•	•						
B C D	4, 12			•							
A B C	8, 9										
A C D	8, 12										
A B D	12, 14									○	○
A C D	11, 15									•	
A B C	14, 15									○	○
B D	1, 3, 9, 11	○	○						○	○	

Table 4.9.12 Prime implicant selection table

### 4.9.3 Prime Implicant Table and Redundant Prime Implicants

We have seen that the prime implicant selection table displays pictorially the covering relationships between the prime implicants and the minterms of the function. It consists of an array of  $u$  columns and  $v$  rows, where  $u$  and  $v$  designate the number of minterms for which the function takes on the value 1 and the number of prime implicants, respectively. The entries of the  $i^{\text{th}}$  row in the chart consists of dots placed at its intersections with the columns, corresponding to minterms covered by the  $i^{\text{th}}$  prime implicant. For example, the prime implicant chart of  $f(ABC) = \sum m(0, 1, 2, 5, 6, 7)$  is shown in Table 4.9.13 (b). It consists of 6 columns corresponding to the minterms of  $f$  and six rows which correspond to the prime implicant generated in Table 4.9.13 (a).

Minterm	Binary representation	Minterms	Binary representation
0	0 0 0	0, 1	0 0 -
1	0 0 1	0, 2	0 - 0
2	0 1 0	1, 5	- 0 1
5	1 0 1	2, 6	- 1 0
6	1 1 0	5, 7	1 - 1
7	1 1 1	6, 7	1 1 -

Table 4.9.13 (a)

The problem now is to select a minimal subset of prime implicants such that column contains at least one dot in the rows corresponding to the selected subset and the total number of literals in the prime implicants selected as small as possible. In our example, each column has two dots. It is a special case. When each column in the prime implicants selection chart has two or more dots, the chart is known as cyclic prime implicant table.

Since all columns have two dots, we have to proceed by trial and error. Both (0, 1) and (0, 2) cover column 0, so we will try (0, 1). After selecting row (0, 1) and column 0, and 1 we examine column 2, which is covered by (0, 2) and (2, 6). The best choice is (2, 6) because it covers two of the remaining columns while (0, 2) covers only one of the remaining columns. After selecting row (2, 6) and columns 2 and 6, we see that (3, 7) covers the remaining columns and completes the solution. Therefore, one of the solution is  $f = \bar{A}\bar{B} + \bar{B}\bar{C} + A\bar{C}$ . The prime implicants which are not selected to get the solution  $\{(0, 2), (1, 5), (6, 7)\}$  are called redundant prime implicants.

Prime Implicants		$m_0$	$m_1$	$m_2$	$m_3$	$m_4$	$m_5$	$m_6$	$m_7$
$\bar{A}\bar{B}$	0, 1 ✓	◎	◎						
$\bar{A}\bar{C}$	0, 2	*		*					
$\bar{B}\bar{C}$	1, 5		*		*				
$B\bar{C}$	2, 6 ✓			◎		◎			
$A\bar{C}$	5, 7 ✓				◎		◎		
$AB$	6, 7					*		*	

Table 4.9.13 (b) Prime implicant selection table

**Example 4.9.4** Minimize the following function by tabular method.

$$f(A, B, C, D) = \sum m(0, 2, 3, 6, 7, 8, 9, 10, 13)$$

Solution :

Minterms	Binary representation	Minterm	Binary representation	Minterm	Binary representation
$m_0$	0 0 0 0	$m_0$ ✓	0 0 0 0	0, 2 ✓	0 0 - 0
$m_2$	0 0 1 0	$m_2$ ✓	0 0 1 0	0, 8 ✓	- 0 0 0
$m_3$	0 0 1 1	$m_3$ ✓	1 0 0 0	2, 3 ✓	0 0 1 -
$m_6$	0 1 1 0	$m_6$ ✓	0 0 1 1	2, 6 ✓	0 - 1 0
$m_7$	0 1 1 1	$m_5$ ✓	0 1 1 0	2, 10 ✓	- 0 1 0
$m_8$	1 0 0 0	$m_9$ ✓	1 0 0 1	8, 9	1 0 0 -

TECHNICAL PUBLICATIONS™ - An up trust for knowledge

Digital Fundamentals		4 - 89		Combinational Digital Circuits	
$m_0$	1 0 0 1	$m_{10}$ ✓	1 0 1 0	8, 10 ✓	1 0 - 0
$m_10$	1 0 1 0	$m_7$ ✓	0 1 1 1	3, 7 ✓	0 - 1 1
$m_{13}$	1 1 0 1	$m_{15}$ ✓	1 1 0 1	6, 7 ✓	0 1 1 -
				9, 13	1 - 0 1

Minterm	Binary representation
0, 2, 8, 10	- 0 - 0
2, 3, 6, 7	0 - 1 -

prime implicant selection chart

Prime implicants	$m_0$	$m_1$	$m_2$	$m_3$	$m_4$	$m_5$	$m_6$	$m_7$	$m_8$	$m_9$	$m_{10}$	$m_{11}$
$\bar{B}\bar{D}$	0, 2, 8, 10 ✓	◎	◎						◎		◎	
$\bar{A}C$	2, 3, 6, 7 ✓		◎	◎	◎	◎						
$A\bar{B}\bar{C}$	8, 9								*	*		
$A\bar{C}D$	9, 13 ✓									◎	◎	

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

**Example 4.9.5** Minimize the following function by tabular method :

$$f(w, x, y, z) = \sum m(1, 4, 8, 9, 13, 14, 15) + d(2, 3, 11, 12)$$

Solution :

Minterms	Binary representation	Minterms	Binary representation	Minterms	Binary representation
$m_1$	0 0 0 1	$m_4$	0 0 1 1	1 3 ✓	0 0 - 1
$m_4$	0 0 1 0	$m_5$	0 0 1 0	1 9 ✓	- 0 0 1
$m_5$	0 0 1 1	$m_6$	0 1 0 0	2, 3	0 0 1 -
$m_6$	0 1 0 0	$m_8$	1 0 0 0	4, 12	- 1 0 0
$m_8$	1 0 0 0	$m_9$	0 0 1 1	8, 9 ✓	1 0 0 -
$m_9$	1 0 0 1	$m_{10}$	1 0 0 1	8, 12 ✓	- 1 0 0
$m_{10}$	1 0 1 1	$m_{11}$	1 1 0 0	3, 11 ✓	- 0 1 1
$m_{11}$	1 1 0 0	$m_{12}$	1 0 1 1	9, 11 ✓	1 0 - 1
$m_{12}$	1 1 0 1	$m_{13}$	1 1 0 1	9, 13 ✓	1 - 0 1
$m_{13}$	1 1 1 0	$m_{14}$	1 1 1 0	12, 13 ✓	1 1 0 -
$m_{14}$	1 1 1 1	$m_{15}$	1 1 1 1	12, 14 ✓	1 1 0 0
$m_{15}$	1 1 1 1			1 1, 15 ✓	1 - 1 1
				13, 15 ✓	1 1 - 1
				14, 15 ✓	1 1 1 -

TECHNICAL PUBLICATIONS™ - An up trust for knowledge

Minterms	Binary representation
1, 3, 9, 11	-0-1
8, 9, 12, 13	1-0-
9, 11, 13, 15	* 1 -- 1
12, 13, 14, 15	1 1 --

Prime Implicant selection chart

Prime implicants	$m_1$	$m_2$	$m_3$	$m_4$	$m_5$	$m_6$	$m_{11}$	$m_{12}$	$m_{13}$	$m_{14}$	$m_{15}$
$\bar{x}z$	1, 3, 9, 11 ✓	◎	◎			◎	◎				
$w\bar{y}$	8, 9, 12, 13 ✓				◎	◎		◎	◎		
$wz$	9, 11, 13, 15				*	*			*		
$wx$	12, 13, 14, 15 ✓							◎	◎	◎	◎
$x\bar{y}z$	4, 12 ✓			◎				◎			

$$f(w, x, y, z) = \bar{B}D + A\bar{C} + AB + B\bar{C}\bar{D}$$

#### 4.9.4 Advantages and Disadvantages of Quine McCluskey Method

The Quine McCluskey method of simplification can be applied to any number of variables and is algorithmic in nature. The simplification process of Quine-McCluskey becomes lengthy and time consuming as number of variables increase; however the algorithm can be easily programmed to run on a computer to identify prime implicants and implicants of any function.

#### Examples for Practice

**Example 4.9.6 :** Find prime implicants for the Boolean expression by using Quine McCluskey method.  $f(A, B, C, D) = \sum (1, 3, 6, 7, 8, 9, 10, 12, 14, 15) + d(11, 13)$   
Ans. :  $F(A, B, C, D) = \bar{B}D + A + BC$

**Example 4.9.7 :** Give simplified logic equation using Quine-McCluskey method for the following Boolean function  $f(A, B, C, D) = \sum m(0, 1, 2, 3, 10, 11, 12, 13, 14, 15)$   
Ans. :  $F(A, B, C, D) = \bar{A}\bar{B} + AC + AB$

**Example 4.9.8 :** Minimize the following using Quine-McCluskey method.  
 $\sum m(0, 1, 2, 8, 9, 15, 17, 21, 24, 25, 27, 31)$

$$\text{Ans. : } \bar{A}\bar{C}\bar{D} + B\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}\bar{E} + A\bar{B}\bar{C}E + BCDE + A\bar{B}\bar{D}\bar{E}$$

**Example 4.9.9 :** Minimize the given terms  $\pi M(0, 1, 4, 11, 13, 15) + \pi d(5, 7, 8)$  using Quine-McCluskey methods and verify the results using K-map method.  
Ans. :  $F(A, B, C, D) = \bar{A}C + \bar{A}\bar{D} + A\bar{B}\bar{C}$

**Example 4.9.10 :** Using QM method simplify the Boolean expression  
 $f(x_1, x_2, x_3, x_4, x_5) = \sum (0, 1, 4, 5, 16, 17, 21, 25, 29)$   
Ans. :  $f = \bar{x}_1\bar{x}_2\bar{x}_4 + x_2\bar{x}_3\bar{x}_4 + x_1\bar{x}_4\bar{x}_5$

**Example 4.9.11 :** Minimize the following function using Quine Mc Cluskey method :  
 $f(A, B, C, D) = \sum m(0, 2, 3, 6, 7, 8, 9, 10, 13)$

**Example 4.9.12 :** Minimize the following function by tabular method and implement the result using NAND gate only :  
 $f(w, x, y, z) = \sum m(1, 4, 8, 9, 13, 14, 15) + d(2, 3, 11, 12)$

#### 4.10 Multiplexers

In digital systems, many times it is necessary to select single data line from several data-input lines, and the data from the selected data line should be available on the output. The digital circuit which does this task is a multiplexer.

It is a digital switch. It allows digital information from several sources to be routed onto a single output line, as shown in the Fig. 4.10.1. The basic multiplexer has several data-input lines and a single output line. The selection of a particular input line is controlled by a set of selection lines. Since multiplexer selects one of the input and routes it to output, it is also known as data selector. Normally, there are  $2^n$  input lines and  $n$  selection lines whose bit combinations determine which input is selected. Therefore, multiplexer is 'many into one' and it provides the digital equivalent of an analog selector switch.

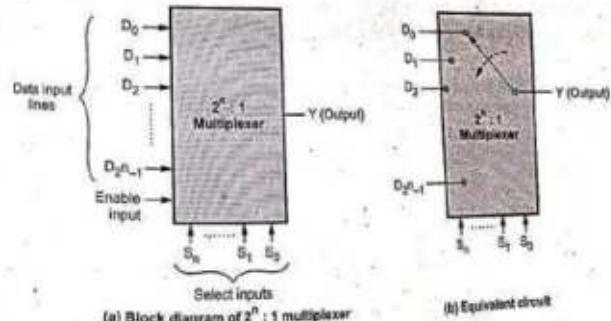
(a) Block diagram of  $2^n : 1$  multiplexer

Fig. 4.10.1

**4.10.1 2 : 1 Multiplexer**

Fig. 4.10.2 (a) shows 2 : 1 multiplexer.  $D_0$  is applied as an input to one AND gate and  $D_1$  is applied as an input to another AND gate. Enable input is applied to both AND gates as one input. Selection line  $S$  is connected as second input to second AND gate. An inverted  $S$  is applied as second input to first AND gate. Outputs of both AND gates are applied as inputs to OR gate.

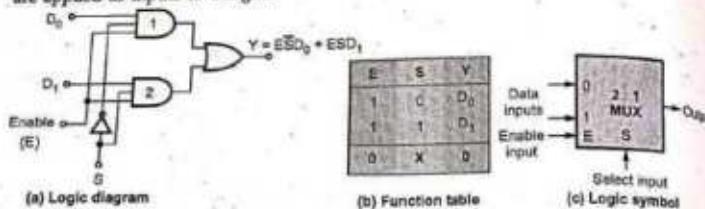


Fig. 4.10.2

**Working**

When  $E = 0$ , output is 0, i.e.  $Y = 0$  irrespective of any input condition. When  $E = 1$ , the circuit works as follows :

When  $S = 0$ , the inverted  $S$ , that is 1 gets applied as second input to first AND gate. Since  $S$  is applied directly as input to second AND gate, its output goes zero irrespective of first input. Since the second input of first AND gate is 1, its output is equal to its first input, that is  $D_0$ . Hence  $Y = D_0$ .

Exactly opposite is the case when  $S = 1$ . In this case, second AND gate output is equal to its first input  $D_1$  and first AND gate output is 0. Hence  $Y = D_1$ . Both these cases are summarized in truth table shown in Fig. 4.10.2 (b).

**Deriving realization expression**

The Table 4.10.1 shows the truth table for 2 : 1 multiplexer. From the truth table it is clear that  $Y = 1$  when  $E \bar{S} D_0 = 1$  or  $ESD_1 = 1$  as indicated by shaded rows.

$$\therefore Y = E\bar{S}D_0 + ESD_1$$

Enable (E)	Select (S)	$D_1$	$D_0$	Output Y
1	0	X	0	0
1	0	X	1	1
1	1	0	X	0
1	1	1	X	1
0	X	X	X	0

Table 4.10.1 Truth table for 2 : 1 multiplexer

**4.10.2 4 : 1 Multiplexer**

Fig. 4.10.3 (a) shows 4-to-1 line multiplexer. Each of the four lines,  $D_0$  to  $D_3$ , is applied to one input of an AND gate. Selection lines are decoded to select a particular AND gate.

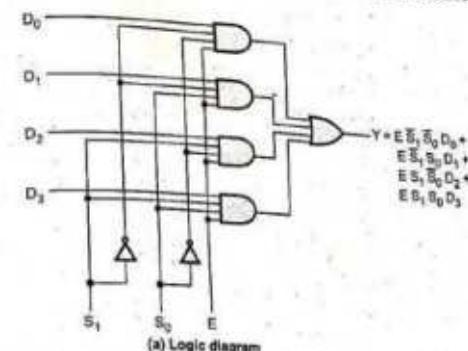


Fig. 4.10.2

E	$S_1$	$S_0$	Y
1	0	0	$D_0$
1	0	1	$D_1$
1	1	0	$D_2$
1	1	1	$D_3$
0	X	X	0

(b) Function table

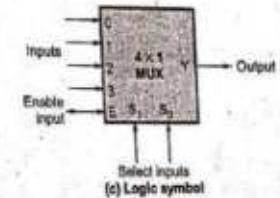


Fig. 4.10.3 4 to 1 line multiplexer

For example, when  $S_1 S_0 = 01$ , the AND gate associated with data input  $D_1$  has two of its inputs equal to 1 and the third input connected to  $D_1$ . The other three AND gates have at least one input equal to 0, which makes their outputs equal to 0. The OR gate output is now equal to the value of  $D_1$ , thus we can say data bit  $D_1$  is routed to the output when  $S_1 S_0 = 01$ .

**4.10.3 8 : 1 Multiplexer**

Fig. 4.10.4 shows 8 : 1 multiplexer.

There are eight input lines one output line and three select lines. As shown in the function table, the selection of a particular input line is controlled by three selection lines.

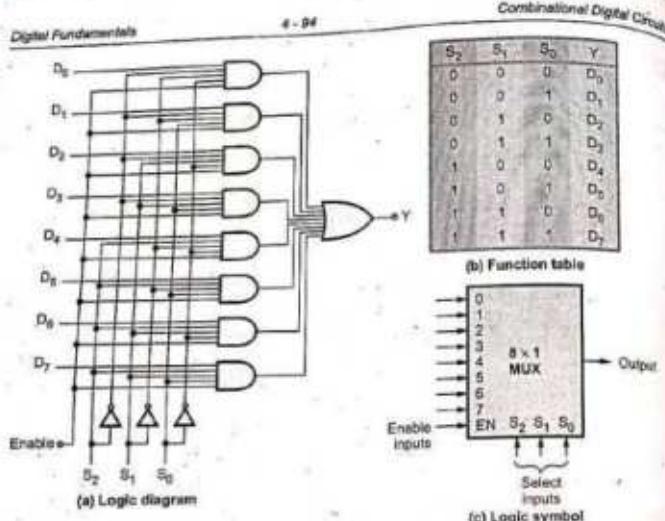


Fig. 4.10.4 8 : 1 Multiplexer

#### 4.10.4 Quadruple 2 to 1 Multiplexer

In some cases, two or more multiplexers are enclosed within one IC package, as shown in the Fig. 4.10.5. The Fig. 4.10.5 shows quadruple 2-to-1 line multiplexer, i.e. four multiplexers, each capable of selecting one of two input lines. Output  $Y_1$  can be selected to be equal to either  $A_1$  or  $B_1$ . Similarly output  $Y_2$  may have the value of  $A_2$  or  $B_2$ , and so on. The selection line  $S$  selects one of two lines in all four multiplexers.

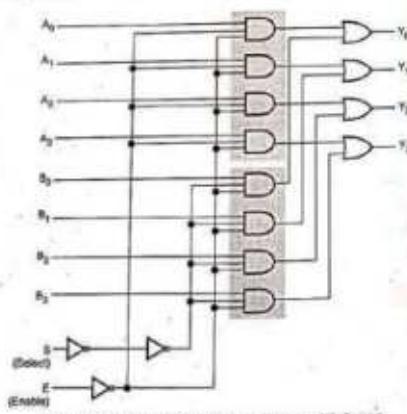


Fig. 4.10.5 Quadruple 2-to-1 line multiplexer

TECHNICAL PUBLICATIONS™ - An up there for knowledge

Digital Fundamentals			4 - 94	Combinational Digital Circuits					
Digital Fundamentals			4 - 95	Combinational Digital Circuits					
The control input E enables the multiplexers in the 0 state and disables them in the 1 state. When E = 1, outputs have all 0's, regardless of the value of S.									
<b>Function Table</b>									
E	S	Output Y							
1	X	All 0's							
0	0	Select A							
0	1	Select B							

#### 4.10.5 The 74151 Multiplexer

The 74XX151 is a 8-to-1 multiplexer. It has eight inputs. One is active high, the other is active low. The Fig. 4.10.6 shows the logic symbol for 74XX151. As shown in the logic symbol, there are three select inputs C, B and A which select one of the eight inputs. The 74XX151 is provided with active low enable input.

The Table 4.10.2 shows the truth table for 74XX151. In this truth table for each input combination, output is not specified in 1s and 0s. Because, we know that, multiplexer is a data switch, it does not generate any data of its own, but it simply passes external input data from the selected input to the output. Therefore, the two output column represent data by  $D_n$  and  $\bar{D}_n$ .

Input		Outputs	
Select		Enable	
C	B	A	EN
x	x	x	1
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	0

Table 4.10.2 Truth table for 74XX151, 8 to 1 multiplexer

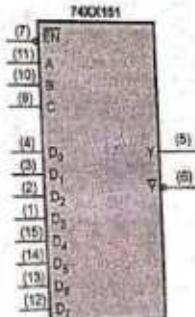


Fig. 4.10.6 Logic symbol for 74XX151, 8 to 1 multiplexer

TECHNICAL PUBLICATIONS™ - An up there for knowledge

#### 4.10.6 The 74XX153 Dual 4 to 1 Multiplexer

The 74XX153 is a dual 4-to-1 multiplexer. Fig. 4.10.7 shows the logic symbol for 74XX153. It contains two identical and independent 4-to-1 multiplexers. Each multiplexer has separate enable inputs. The Table 4.10.3 shows the truth table for 74XX153.

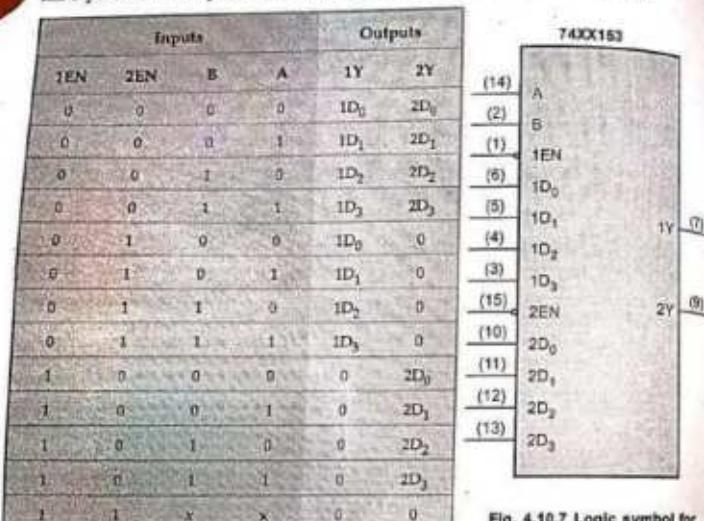


Fig. 4.10.7 Logic symbol for 74XX153

Table 4.10.3 Truth table for 74XX153, dual 4-to-1 multiplexer

#### 4.10.7 Expanding Multiplexers

It is possible to expand range of inputs for multiplexer beyond the available range by interconnecting several multiplexers in cascade. The circuit with two or more multiplexers connected to obtain the multiplexer with more number of inputs is known as multiplexer tree.

#### Illustrative Examples

**Example 4.10.1** Design 16 : 1 multiplexer using 8 : 1 multiplexer.

Solution :

Step 1 : Connect the select lines ( $S_2, S_1$  and  $S_0$ ) of two multiplexers in parallel.

Step 2 : Connect most significant select line ( $S_3$ ) such that when  $S_3 = 0$  MUX 1 is enabled and when  $S_3 = 1$ , MUX 2 is enabled.

Step 3 : Logically OR the outputs of two multiplexers to obtain the final output Y.

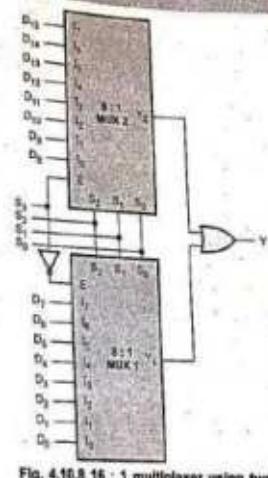


Fig. 4.10.8 16 : 1 multiplexer using two 8 : 1 multiplexers

**Example 4.10.2** Design 16 : 1 multiplexer using 4 : 1 multiplexers.

Solution : Since there are 16-inputs for the multiplexers we require four 4 : 1 multiplexers to satisfy input needs. The four outputs of 4 : 1 multiplexers are again multiplexed by 4 : 1 multiplexer to generate final output.

Step 1 : Connect the select lines ( $S_3$  and  $S_2$ ) of four multiplexers in parallel.

Step 2 : Connect the most significant select lines ( $S_1$  and  $S_0$ ) to the MUX 5.

Step 3 : Connect the outputs  $Y_0, Y_1, Y_2$  and  $Y_3$  of four multiplexers as data inputs for the MUX 5, as shown in the Fig. 4.10.2.

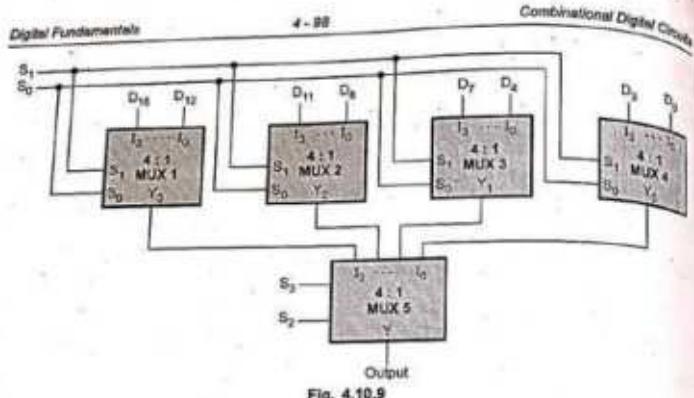


Fig. 4.10.9

#### Examples for Practice

**Example 4.10.3** Draw the block diagram of a 4 : 1 multiplexer using 2 : 1 MUX.

**Example 4.10.4** Design 32 : 1 MUX using 8 : 1 MUX.

**Example 4.10.5** Design 14 : 1 mux using 4 : 1 mux (with enable inputs). Explain the truth table of your circuit in short.

**Example 4.10.6** Design 28 : 1 mux using 8 : 1 mux (with enable inputs). Explain truth table of your design in short. [Hint : You can use separate mux for enable of respective IC's]

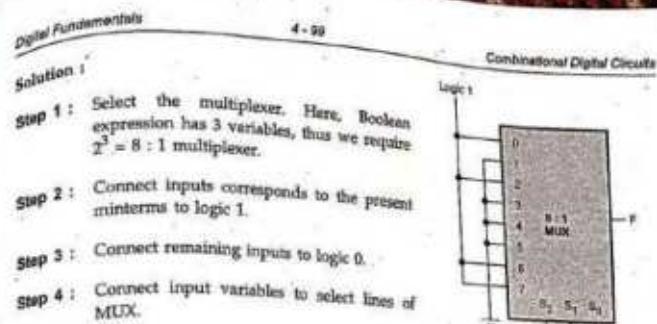


Fig. 4.10.10

**Example 4.10.8** Implement the Boolean function represented by the given truth table using multiplexer.

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

**Solution :**

**Step 1 :** Select the multiplexer. Here, there are three input variables, thus we require  $2^3 = 8 : 1$  multiplexer.

**Step 2 :** Find the minterm expression.

Minterm expression for given truth table is  $\sum m(1, 2, 5, 7)$ .

**Step 3 :** Connect inputs corresponds to the present minterms to logic 1.

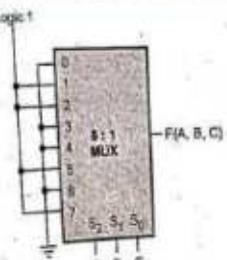


Fig. 4.10.11

**Step 4 :** Connect remaining inputs to logic 0.

**Step 5 :** Connect input variables to select lines of MUX.

In the above example, we have seen the method for implementing Boolean function of 3 variables with  $2^3$ -to-1 multiplexer. Similarly, we can implement any Boolean function of  $n$  variables with  $2^n$ -to-1 multiplexer. However, it is possible to do better than this. If we have Boolean function of  $n+1$  variables, we take  $n$  of these variables and connect them to the selection lines of a multiplexer. The remaining single variable of the function is used for the inputs of the multiplexer. In this way we can implement any Boolean function of  $n$  variables with  $2^{n-1}$ -to-1 multiplexer. Let us see some examples.

**Example 4.10.9** Implement the following Boolean function using 4 : 1 multiplexer.

$$F(A, B, C) = \sum m(1, 3, 5, 6)$$

**Solution :**

**Step 1 :** Connect least significant variables as select inputs of multiplexer. Here, connect  $C$  to  $S_0$  and  $B$  to  $S_1$ .

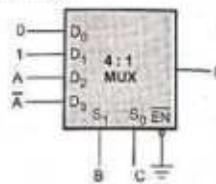
**Step 2 :** Derive inputs for multiplexer using implementation table.

	D <sub>0</sub>	D <sub>1</sub>	D <sub>2</sub>	D <sub>3</sub>
Ā	0	①	2	③
A	4	⑤	⑥	7

row 1  
row 2  
Most significant variable ↑

(a) Implementation table

Fig. 4.10.12



(b) Multiplexer implementation

As shown in the Fig. 4.10.12 (a) the implementation table is nothing but the list of the inputs of the multiplexer and under them list of all the minterms in two rows. The first row lists all those minterms where  $A$  is complemented, and the second row lists all the minterms with  $A$  uncomplemented. The minterms given in the function are circled and then each column is inspected separately as follows :

- If the two minterms in a column are not circled, 0 is applied to the corresponding multiplexer input (see column 0).
- If the two minterms in a column are circled, 1 is applied to the corresponding multiplexer input (see column 1).
- If the minterm in the second row is circled and minterm in the first row is not circled,  $A$  is applied to the corresponding multiplexer input (see column 2).

- If the minterm in the first row is circled and minterm in the second row is not circled,  $\bar{A}$  is applied to the corresponding multiplexer input (see column 3).

**Example 4.10.10** Implement the following Boolean function using 8 : 1 multiplexer

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

**Solution :**

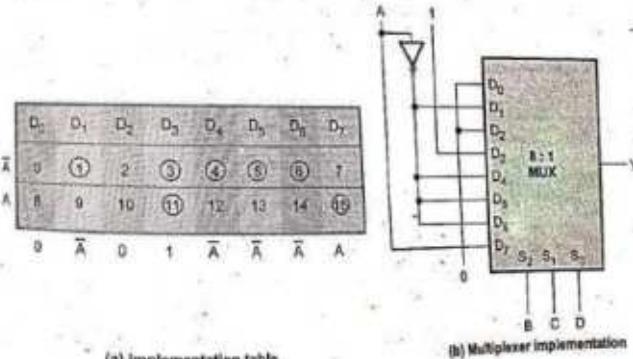
**Step 1 :** Express Boolean function in the minterm form.

The given Boolean expression is not in standard SOP form. Let us first convert this in standard SOP form

$$\begin{aligned} F(A, B, C, D) &= \bar{A} \cdot B \cdot \bar{D} \cdot (C + \bar{C}) + A \cdot C \cdot D \cdot (B + \bar{B}) + \\ &= \bar{A} \cdot B \cdot C \cdot \bar{D} + \bar{A} \cdot B \cdot \bar{C} \cdot \bar{D} + A \cdot B \cdot C \cdot D + A \cdot \bar{B} \cdot C \cdot D \\ &+ A \cdot B \cdot C \cdot \bar{D} + \bar{A} \cdot B \cdot \bar{C} \cdot D + A \cdot B \cdot \bar{C} \cdot \bar{D} + \bar{A} \cdot \bar{B} \cdot C \cdot D + \\ &= \sum m(6, 4, 15, 11, 3, 5, 1) \\ &= \sum m(1, 3, 4, 5, 6, 11, 15) \end{aligned}$$

**Step 2 :** Implement it using implementation table.

From the Boolean function in the minterm form can be implemented using 8 : 1 multiplexer as follows :



(a) Implementation table

Fig. 4.10.13

**Example 4.10.11** Implement the following Boolean function with 8 : 1 multiplexer

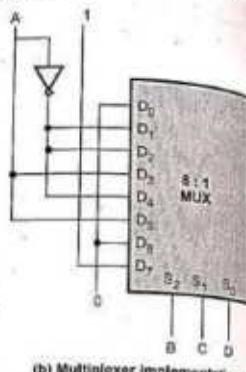
$$F(A, B, C, D) = \sum m(0, 3, 5, 6, 8, 9, 10, 12, 14)$$

**Solution :** Here, instead of minterms, maxterms are specified. Thus, we have to find maxterms which are not included in the Boolean function. Fig. 4.10.14 shows the implementation of Boolean function with 8 : 1 multiplexer.

	D <sub>0</sub>	D <sub>1</sub>	D <sub>2</sub>	D <sub>3</sub>	D <sub>4</sub>	D <sub>5</sub>	D <sub>6</sub>	D <sub>7</sub>
Ā	0	1	2	3	4	5	6	7
A	0	9	10	11	12	13	14	15
0	Ā	Ā	A	Ā	A	A	0	1

(a) Implementation table

Fig. 4.10.14



(b) Multiplexer implementation

**Example 4.10.12** Implement the following Boolean function with 8 : 1 multiplexer.

$$F(A, B, C, D) = \sum m(0, 2, 6, 10, 11, 12, 13) + d(3, 8, 14)$$

**Solution :** In the given Boolean function three don't care conditions are also specified. We know that don't care conditions can be treated as either 0s or 1s. Fig. 4.10.15 shows the implementation of given Boolean function using 8 : 1 multiplexer.

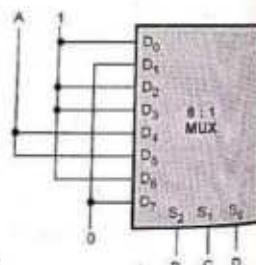
In this example, by taking don't care conditions 8 and 14 as 1s we have eliminated 1 term and hence the inverter.

	D <sub>0</sub>	D <sub>1</sub>	D <sub>2</sub>	D <sub>3</sub>	D <sub>4</sub>	D <sub>5</sub>	D <sub>6</sub>	D <sub>7</sub>
Ā	0	1	2	3	4	5	6	7
A	0	9	10	11	12	13	14	15
1	0	1	1	1	A	A	1	0

Here, don't cares are treated as 1s

(a) Implementation table

Fig. 4.10.15



(b) Implementation

**Examples for Practice**

**Example 4.10.13** Implement the following expression using 8 : 1 multiplexer :  $f(A, B, C, D) = \sum m(2, 4, 6, 7, 9, 10, 11, 12, 15)$ .

**Example 4.10.14** Implement Boolean function  $f = AB + \bar{C}D + \bar{A}\bar{B}C$  using 8 : 1 multiplexer.

**4.10.9 Implementation of Logic Function using IC 74153**

The 74XX153 is a dual 4 to 1 multiplexer. We have already seen how to implement combinational circuits using multiplexers in section 4.10.8. Let us see some examples of logic function implementation using IC 74153.

**Example 4.10.15** Implement full adder using IC 74153.

**Solution :** The Table 4.10.4 shows the truth table for full adder.

Inputs			Outputs	
A	B	C <sub>in</sub>	Carry	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

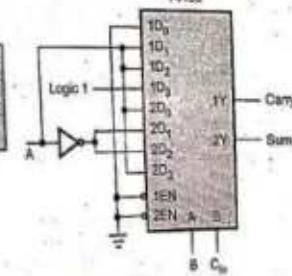
Table 4.10.4 Truth table for full-adder

For carry			
1D <sub>0</sub>	1D <sub>1</sub>	1D <sub>2</sub>	1D <sub>3</sub>
0	1	2	3
Ā	A	A	A

For sum			
2D <sub>0</sub>	2D <sub>1</sub>	2D <sub>2</sub>	2D <sub>3</sub>
0	3	2	3
Ā	A	Ā	A

(a) Implementation tables

Fig. 4.10.16



(b) Implementation

Since 74153 is a dual 4 : 1 multiplexer, we can use one 4 : 1 multiplexer to implement carry function and other 4 : 1 multiplexer to implement sum function.

**Example 4.10.16** Implement full subtractor using IC 74153.

**Solution :** The Table 4.10.5 shows the truth table for 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

Table 4.10.5 Truth table for full-subtractor

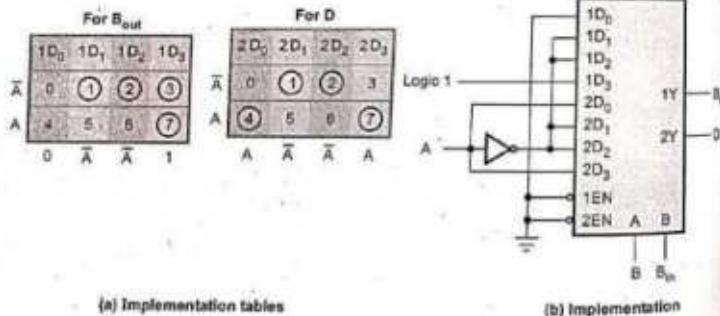


Fig. 4.10.17

TECHNICAL PUBLICATIONS™ - An up thrust for knowledge

Since 74153 is a dual 4 : 1 multiplexer, we can use one 4 : 1 multiplexer to implement  $B_{out}$  and other 4 : 1 multiplexer to implement sum function.

**Example 4.10.17** Design a 8 to 1 multiplexer by using the four minterms given by  $F(A, B, C, D) = \sum m(0, 1, 3, 4, 5, 8, 10, 15)$ .

**Solution :**

Implementation table

$\bar{A}$	$D_0$	$D_1$	$D_2$	$D_3$	$D_4$	$D_5$	$D_6$	$D_7$
0	0	1	2	3	4	5	6	7
1	9	10	11	12	13	14	15	

Fig. 4.10.18

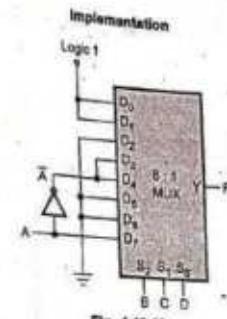


Fig. 4.10.19

**Example 4.10.18** Implement the given function using 8 × 1 Multiplexer  
 $F(A, B, C, D) = \sum m(0, 1, 2, 3, 5, 8, 9, 11, 14)$

GTU : Summer-15, Marks 7

**Solution :** Implementation Table

$\bar{A}$	$D_0$	$D_1$	$D_2$	$D_3$	$D_4$	$D_5$	$D_6$	$D_7$
0	0	1	2	3	4	5	6	7
1	9	10	11	12	13	14	15	

Fig. 4.10.20

Implementation

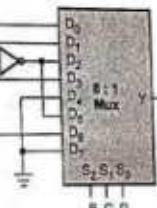


Fig. 4.10.21

**Example 4.10.19** Implement following logic function using 8 × 1 MUX

$$F = \sum m(0, 1, 3, 5, 7, 11, 13, 14, 15)$$

GTU : Winter-15, Marks 7

TECHNICAL PUBLICATIONS™ - An up thrust for knowledge

## Solution : Implementation table

$\bar{A}$	$A$	$D_0$	$D_1$	$D_2$	$D_3$	$D_4$	$D_5$	$D_6$	$D_7$
0	0	0	1	2	3	4	5	6	7
1	1	8	6	10	11	12	13	14	15

Fig. 4.10.22

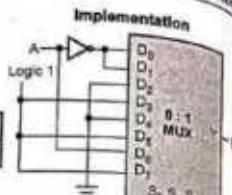


Fig. 4.10.23

**Example 4.10.20** Implement following Boolean function using 8:1 multiplexer.

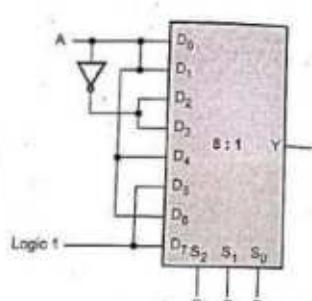
$$F(A, B, C, D) = \sum(2, 3, 5, 7, 8, 9, 12, 13, 14, 15)$$

GTU : Summer-17, Marks 7

## Solution : Implementation Table

$\bar{A}$	$A$	$D_0$	$D_1$	$D_2$	$D_3$	$D_4$	$D_5$	$D_6$	$D_7$
0	0	0	1	2	3	4	5	6	7
1	1	8	9	10	11	12	13	14	15

(a) Implementation table



(b) Implementation

Fig. 4.10.24

**Example 4.10.21** Implement the following Boolean functions with a multiplexer.

$$F(w, x, y, z) = \sum(1, 3, 5, 6, 11, 14, 15)$$

GTU : Winter-18, Marks 7

## Solution : Implementation using Multiplexer

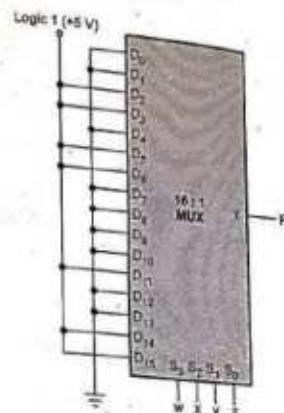


Fig. 4.10.25

## 4.10.10 Applications of Multiplexer

- They are used as a data selector to select one out of many data inputs.
- They can be used to implement combinational logic circuit.
- They are used in time multiplexing systems.
- They are used in frequency multiplexing systems.
- They are used in A/D and D/A converter.
- They are used in data acquisition systems.

## 4.10.11 Multiplexer ICs

IC number	Function
74150	16:1 multiplexer
74151	8:1 multiplexer
74153	Dual 4:1 multiplexer
74157	Quad 2-input multiplexer

Table 4.10.6 Multiplexer ICs

## Review Questions

- Draw logic circuit of 4 : 1 MUX.
- What is multiplexer?
- Explain the working of 8 : 1 multiplexer.
- Obtain an 8 : 1 multiplexer with a dual 4-line to 1-line multiplexers having separate enable inputs but common selection lines.
- Design an 8 : 1 multiplexer using two 4 : 1 multiplexers. Explain with the help of the truth table. Implement the function  $f(A, B, C) = \sum m(1, 3, 7)$  using the same.
- Realize the expression  $Y(A, B, C, D) = \sum (15, 7, 4, 6, 8, 9, 12, 14)$  using an 8:1 MUX.

GTU : Summer-18, Marks 3

GTU : Dec-13, Marks 3

GTU : Summer-15, Marks 2

## Differentiate between Multiplexer and Demultiplexer

Parameter	Multiplexer	Demultiplexer
Definition	Multiplexer is a digital switch which allows digital information from several sources to be routed onto a single output line.	Demultiplexer is a circuit that receives information on a single line and transmits this information on one of $2^n$ possible output lines.
Number of data inputs	$2^n$	1
Number of data outputs	1	$2^n$
Relationship of input and output	Many to one	One to many
Applications	<ul style="list-style-type: none"> <li>Used as a data selector</li> <li>In time division multiplexing at the transmitting end</li> </ul>	<ul style="list-style-type: none"> <li>Used as a data distributor</li> <li>In time division multiplexing at the receiving end</li> </ul>

## 4.11 Demultiplexers

A demultiplexer is a circuit that receives information on a single line and transmits this information on one of  $2^n$  possible output lines. The selection of specific output line is controlled by the values of  $n$  selection lines.

The Fig. 4.11.1 shows the block diagram of a demultiplexer. It has one input data line,  $2^n$  output lines,  $n$  select lines and one enable input.

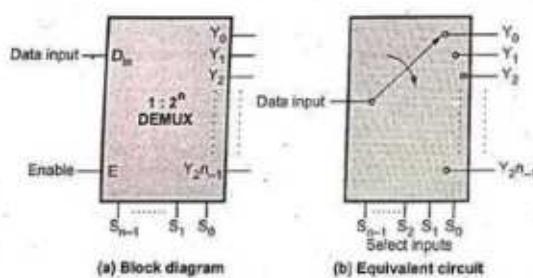


Fig. 4.11.1

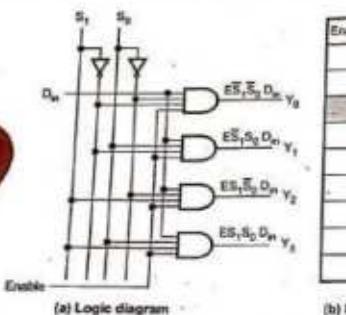
## 4.11.1 Types of Demultiplexers

## 4.11.1.1 1 : 4 Demultiplexer

Fig. 4.11.2 shows 1 : 4 demultiplexer. The single input variable  $D_m$  has a path to all four outputs, but the input information is directed to only one of the output lines depending on the select inputs. Enable input should be high to enable demultiplexer.

## 4.11.1.2 1 : 8 Demultiplexer

The Fig. 4.11.3 shows 1 : 8 demultiplexer. The single input data  $D_m$  has a path to all eight outputs, but the input information is directed to only one of the output lines depending on the select inputs.

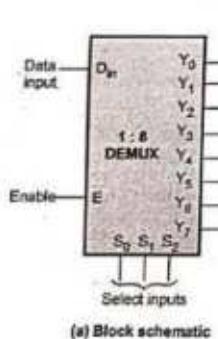


(a) Logic diagram

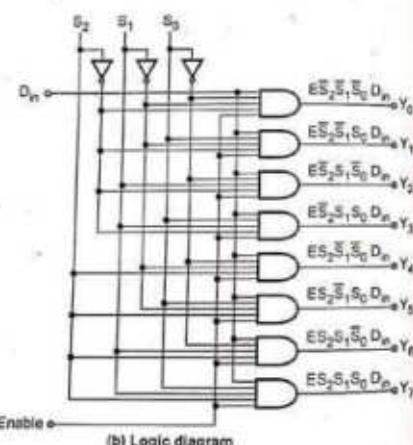
Enable (E)	S <sub>1</sub>	S <sub>2</sub>	D <sub>in</sub>	Y <sub>0</sub>	Y <sub>1</sub>	Y <sub>2</sub>	Y <sub>3</sub>
0	X	X	X	0	0	0	0
1	0	0	0	0	0	0	0
-1	0	0	1	1	0	0	0
1	0	1	0	0	0	0	0
1	0	1	1	0	1	0	0
1	1	0	0	0	0	0	0
1	1	0	1	0	0	1	0
1	1	1	0	0	0	0	0
1	1	1	1	0	0	0	1

(b) Function table for 1:4 demultiplexer

Fig. 4.11.2



(a) Block schematic



(b) Logic diagram

Enable	Select inputs			Outputs								
	E	S <sub>2</sub>	S <sub>1</sub>	S <sub>0</sub>	Y <sub>0</sub>	Y <sub>1</sub>	Y <sub>2</sub>	Y <sub>3</sub>	Y <sub>4</sub>	Y <sub>5</sub>	Y <sub>6</sub>	Y <sub>7</sub>
0	X	X	X	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	1	0	1	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0	0
1	0	1	1	0	1	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	1	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0	0	0

(c) Function table

Fig. 4.11.3 1 : 8 demultiplexer

**4.11.2 Expanding Demultiplexers**

To provide larger output needs we can cascade two or more demultiplexers to get demultiplexer with more number of output lines. Such a connection is known as demultiplexer tree.

**Example 4.11.1** Design 1 : 8 demultiplexer using two 1 : 4 demultiplexers.

**Solution :**

**Step 1 :** Connect D<sub>in</sub> signal to D<sub>in</sub> input of both the demultiplexers.

**Step 2 :** Connect select lines B and C to select lines S<sub>1</sub> and S<sub>0</sub> of the both demultiplexers, respectively.

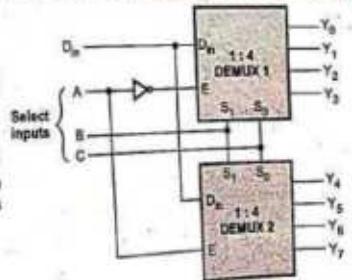


Fig. 4.11.4 Cascading of demultiplexers

**Step 3 :** Connect most significant select line (A) such that when  $A = 0$  DEMUX 1 is enabled and when  $A = 1$  DEMUX 2 is enabled.

**Example 4.11.2** Implement 1 : 16 demultiplexer using 1 : 4 demultiplexers.

**Solution :** The 1 : 16 demultiplexer has 16 outputs. To select one of the 16 output, the circuit needs 4 ( $: 2^4 = 16$ ) select lines. Each 1 : 4 demultiplexer requires 2 select lines.

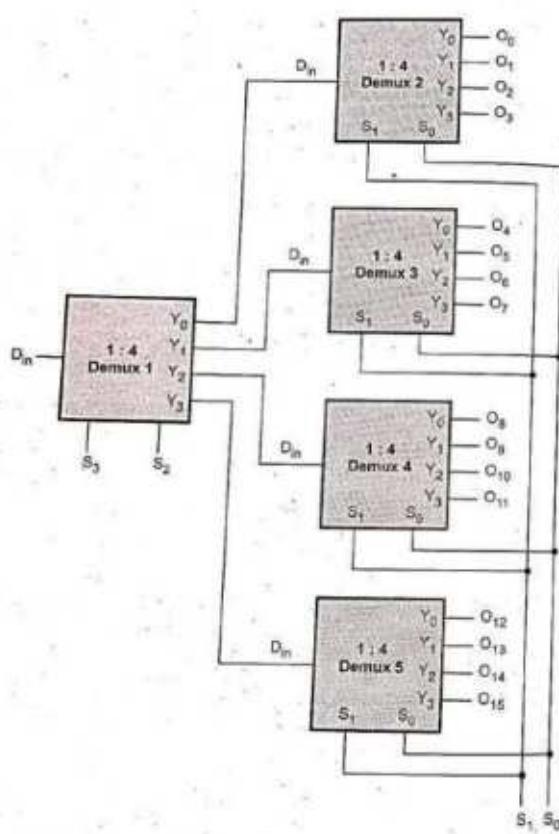


Fig. 4.11.5 1 : 16 Demux using 1 : 4 Demux

**Step 1 :** Connect two least significant select lines ( $S_1, S_0$ ) to select lines of four 4 : 1 demultiplexers.

**Step 2 :** Connect one more 4 : 1 demultiplexer such that its four outputs are routed to the data inputs of the four demultiplexers. Connect higher select lines ( $S_2, S_3$ ) to the select lines of this demultiplexer. (See Fig. 4.125 on previous page).

**Examples for Practice**

**Example 4.11.3** Draw 1 : 64 demultiplexer tree using 1 : 16 demultiplexer.  
**Example 4.11.4** Draw 1 : 64 demultiplexer tree using 1 : 8 demultiplexer.

**4.11.3 Implementation of Combinational Logic using Demultiplexer**

**Example 4.11.5** Implement full subtractor using demultiplexer

**Solution :**

**Step 1 :** Write the truth table of full subtractor.

**Step 2 :** Represent output of full-subtractors in minterm form.

For full subtractor difference D function can be written as  $D = f(A, B, C) = \sum m(1, 2, 4, 7)$  and  $B_{out}$  function can be written as,

$$B_{out} = F(A, B, C) = \sum m(1, 2, 3, 7)$$

**Step 3 :** Logically OR the outputs corresponding to minterms.

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

Table 4.11.1 Truth table of full subtractor

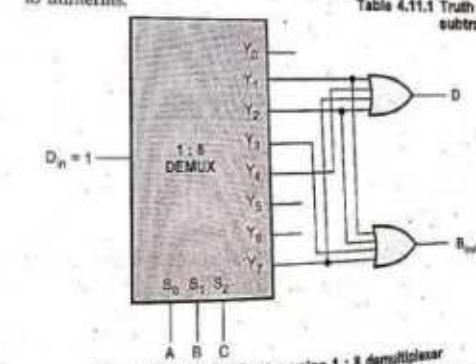


Fig. 4.11.6 Full subtractor using 1 : 8 demultiplexer

With  $D_m$  input 1, demultiplexer gives minterms at the output so by logically ORing required minterms we can implement Boolean functions for full subtractor. Fig. 4.11.6 shows the implementation of full subtractor using demultiplexer.

**Example 4.11.6** Implement the following functions using demultiplexer :

$$f_1(A, B, C) = \sum m(0, 3, 7)$$

$$f_2(A, B, C) = \sum m(1, 2, 5)$$

**Solution :**

$$f_1(A, B, C) = \sum m(0, 3, 7)$$

$$f_2(A, B, C) = \sum m(1, 2, 5)$$

Implementation using 1 : 8 demultiplexer.

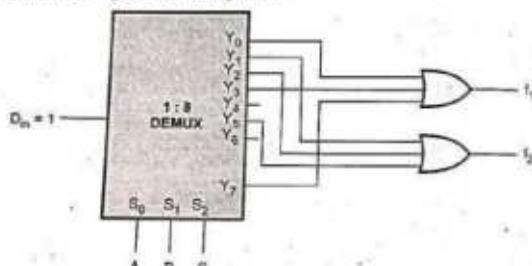


Fig. 4.11.7

**Examples for Practice**
**Example 4.11.7** Implement full adder using demultiplexer.

**Example 4.11.8** Implement the following functions using demultiplexer

$$f_1(A, B, C) = \sum m(1, 3, 5, 7), \quad f_2(A, B, C) = \sum m(3, 6, 7)$$

**4.11.4 Applications of Demultiplexer**

1. It can be used as a decoder.
2. It can be used as a data distributor.
3. It is used in time division multiplexing at the receiving end as a data separator.
4. It can be used to implement Boolean expressions.

IC Number	Function
74154	1 : 16 Demultiplexer
74153	Dual 1 : 4 Demultiplexer

Refer Appendix A for details

Table 4.11.2 Demultiplexer ICs

**Review Questions**

1. Define demultiplexer.
2. Differentiate between multiplexer and demultiplexer.
3. State the applications of demultiplexer.

**4.12 Decoders**

A decoder is a multiple-input, multiple-output logic circuit which converts coded inputs into coded outputs, where the input and output codes are different.

The Fig. 4.12.1 shows the general structure of the decoder circuit. As shown in the Fig. 4.12.1, the encoded information is presented as  $n$  inputs producing  $2^n$  possible outputs. The  $2^n$  output values are from 0 through  $2^n - 1$ .

Usually, a decoder is provided with enable inputs to activate decoded output based on data inputs. When any one enable input is asserted, all outputs of decoder are disabled.



Fig. 4.12.1 General structure of decoder

**4.12.1 Binary Decoder**

A decoder which has an  $n$ -bit binary input code and a one activated output out of  $2^n$  output code is called **binary decoder**. A binary decoder is used when it is necessary to activate exactly one of  $2^n$  outputs based on an  $n$ -bit input value.

Fig. 4.12.2 shows 2-to-4 decoder. Here, 2 inputs are decoded into four outputs, each output representing one of the

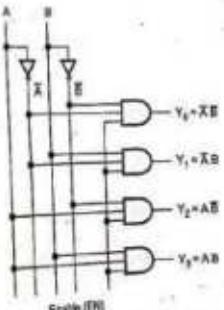


Fig. 4.12.2 2 to 4 line decoder

minterms of the 2 input variables. The two inverters provide the complement of the inputs, and each one of four AND gates generates one of the minterms.

The Table 4.12.1 shows the truth table for a 2-to-4 decoder. As shown in the truth table, if enable input is 1 (EN = 1), one, and only one, of the outputs  $Y_0$  to  $Y_3$  is active for a given input. The output  $Y_0$  is active, i.e.  $Y_0 = 1$  when inputs  $A = B = 0$ , the output  $Y_1$  is active when inputs  $A = 0$  and  $B = 1$ . If enable input is 0, i.e. EN = 0, then all the outputs are 0.

Inputs			Outputs				
EN	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	

Table 4.12.1 Truth table for a 2 to 4 decoder

**Example 4.12.1** With logic circuit and truth table explain the working of 3 : 8 line decoder.

[GTU : Dec-13, Marks 7]

**Solution :** Fig. 4.12.3 shows 3-to-8 line decoder. Here, 3 inputs are decoded into eight outputs, each output represent one of the minterms of the 3 input variables. The three inverters provide the complement of the inputs, and each one of the eight AND gate generates one of the minterms. Enable input is provided to activate decoded output based on data inputs A, B, and C. The table shows the truth table for 3-to-8 decoder.

Inputs			Outputs								
EN	A	B	C	$Y_7$	$Y_6$	$Y_5$	$Y_4$	$Y_3$	$Y_2$	$Y_1$	$Y_0$
0	X	X	X	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

Table 4.12.2 Truth table for a 3 to 8 decoder

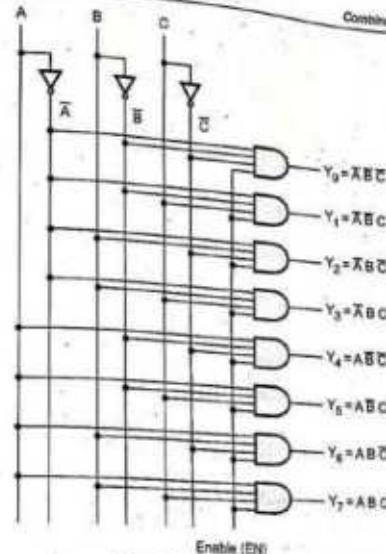


Fig. 4.12.3 3 : 8 line decoder

#### 4.12.2 The 74X138 3-to-8 Decoder

The 74X138 is a commercially available 3-to-8 decoder. It accepts three binary inputs (A, B, C) and when enabled, provides eight individual active low outputs ( $Y_0$  -  $Y_7$ ). The device has three enable inputs : Two active low ( $G_{2A}$ ,  $G_{2B}$ ) and one active high ( $G_1$ ). Fig. 4.12.4 and Table 4.12.3 show logic symbol and function table respectively.

Inputs			Outputs										
$G_{2B}$	$G_{2A}$	$G_1$	C	B	A	$\bar{Y}_7$	$\bar{Y}_6$	$\bar{Y}_5$	$\bar{Y}_4$	$\bar{Y}_3$	$\bar{Y}_2$	$\bar{Y}_1$	$\bar{Y}_0$
1	X	X	X	X	X	1	1	1	1	1	1	1	1
X	1	X	X	X	X	1	1	1	1	1	1	1	1
X	X	0	X	X	X	1	1	1	1	1	1	1	1
0	0	1	0	0	1	1	1	1	1	1	1	1	1
0	0	1	0	0	1	1	1	1	1	1	1	1	1
0	0	1	0	1	0	1	1	1	1	1	1	1	1

TECHNICAL PUBLICATIONS™ An up beat for knowledge

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

Table 4.12.3 Function table

**4.12.3 Expanding Cascading Decoders**

Binary decoder circuits can be connected together to form a larger decoder circuit. Fig. 4.12.5 shows the 4 × 16 decoder using two 3 × 8 decoders.

Here, one input line (D) is used to enable/disable the decoders. When D = 0, the top decoder is enabled and the other is disabled. Thus the bottom decoder outputs are all 1s, and the top eight outputs generate minterms 0 to 8. When D=1, the enable conditions are reversed and thus bottom decoder outputs generate minterms 1000 to 1111, while the outputs of the top decoder are all 1s.

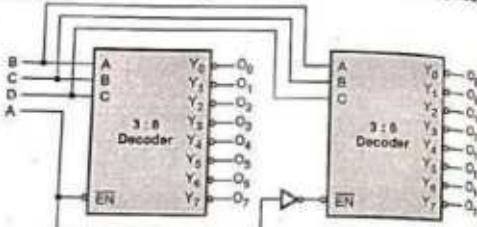


Fig. 4.12.5



Fig. 4.12.4 Logic symbol

**Example 4.12.2 Design 5-to-32 decoder using one 2-to-4 and four 3-to-8 decoder ICs.**

**Solution :** The Fig. 4.12.6 shows the construction of 5-to-32 decoder using four 74LS138 and half 74LS139 IC. The half section of 74LS139 IC is used as a 2-to-4 decoder to decode the two higher order inputs, D and E. The four outputs of this decoder are used to enable one of the four 3-to-8 decoders. The three lower order inputs A, B and C are connected in parallel to four 3-to-8 decoders. This means that the same output pin of each of the four 3-to-8 decoders is selected but only one is enabled. The remaining enable signals of four 3-to-8 decoder ICs are connected in parallel to construct enable signals for 5-to-32 decoder.

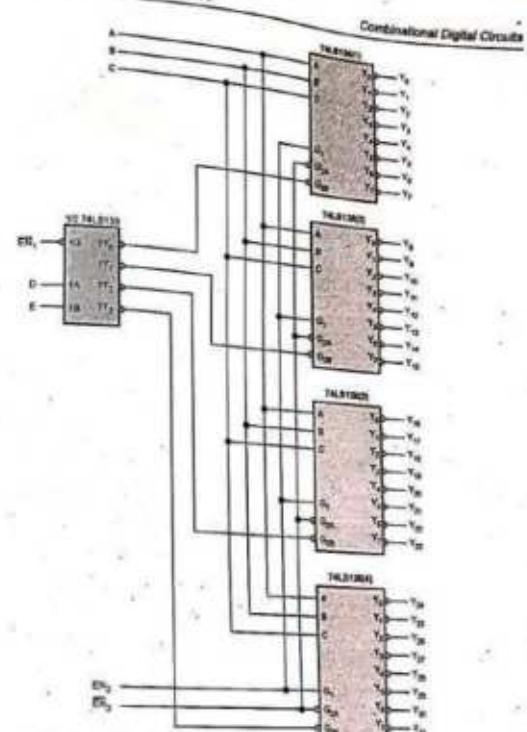


Fig. 4.12.6 5-to-32 decoder using 74LS138 and 74LS139

**Example 4.12.3 Design 4 line to 16 line decoder using 2 line to 4 line decoders.**

GTU : May-12, 14, Marks 7

**Solution :** 4 line to 16 line decoder using 1 line to 4 line decoder.

As shown in Fig. 4.12.7 five 2 : 4 decoders are required to design 4 : 16 decoder. Decoder 1 is used to enable one of the decoder 2, 3, 4 and 5. Inputs of first decoder are the A and B, MSB inputs of 4 : 16 decoder. The inputs of decoder 2, 3, 4 and 5 are connected together forming C and D LSB inputs of 4 : 16 decoder.

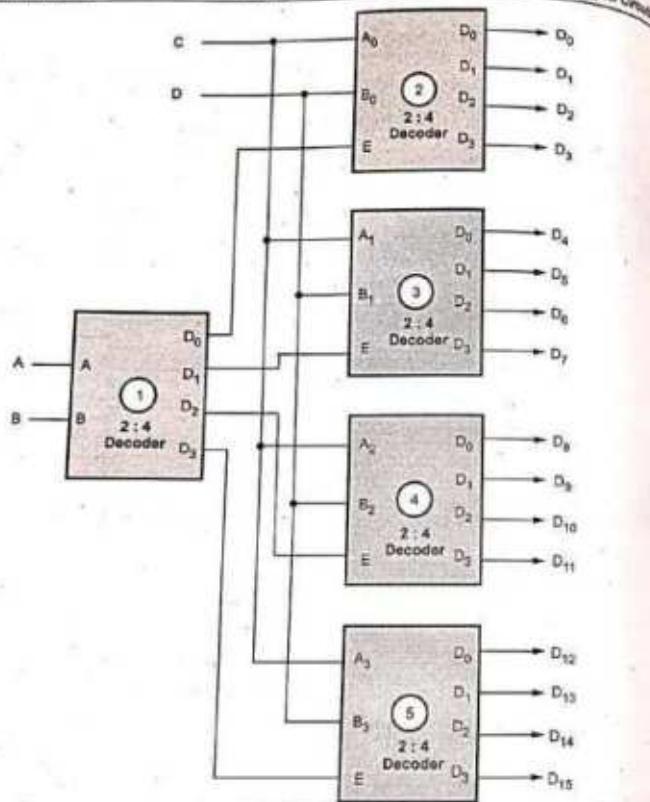


Fig. 4.12.7

When A and B are 0 0, decoder 1 is enabled, for AB = 0 1 decoder 2 is enabled, for AB = 1 0 decoder 3 is enabled and for A = B = 1, decoder 4 is enabled.

#### 4.12.4 Realization of Boolean Function using Decoder

The combination of decoder and external logic gates can be used to implement single or multiple output functions. We know that decoder can have one of the two output

states; either active low or active high. Let us see the significance of these output states in the implementation of binary function.

When decoder output is active high, it generates minterms (product terms) for input variables; i.e. it makes selected output logic 1. In such case to implement SOP function we have to take sum of selected product terms generated by decoder.

#### Illustrative Examples

**Example 4.12.4** Implement Boolean function  $F = \sum m(1, 2, 3, 7)$  using 3 : 8 decoder.

Solution :

Step 1 : Connect function variables as inputs to the decoder.

Step 2 : Logically OR the outputs correspond to present minterms to obtain the output.

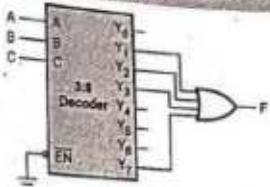


Fig. 4.12.8

**Example 4.12.5** Design and implement a full adder circuit using a 3 : 8 decoder.

GTU Winter-15, Marks 7

Solution : Truth table for full adder is as shown in the Table 4.12.4.

Inputs			Outputs	
A	B	C <sub>in</sub>	Carry	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Table 4.12.4 Truth table for full-adder

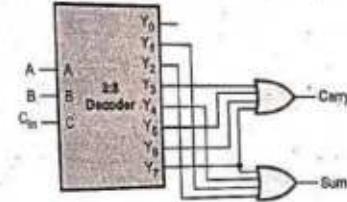


Fig. 4.12.9

**Example 4.12.6** Implement the following Boolean functions with a decoder.

$$F(w, x, y, z) = \sum m(1, 3, 5, 6, 11, 14, 15)$$

GTU : Winter-18, Marks 7

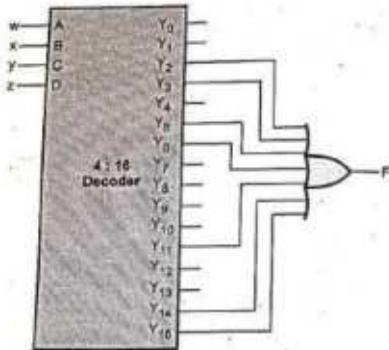
**Solution :**

Fig. 4.12.10

**Example 4.12.7** Using suitable decoder and OR gates, design 4-bit binary to gray code converter.  
GTU : Summer-18, Marks 7

**Solution :****Step 1 :** Form the Truth table relating 8421 binary code and Gray codeInput code : Binary code : B<sub>3</sub> B<sub>2</sub> B<sub>1</sub> B<sub>0</sub>Output code : Gray code : G<sub>3</sub> G<sub>2</sub> G<sub>1</sub> G<sub>0</sub>

Decimal	Binary code				Gray code			
	B <sub>3</sub>	B <sub>2</sub>	B <sub>1</sub>	B <sub>0</sub>	G <sub>3</sub>	G <sub>2</sub>	G <sub>1</sub>	G <sub>0</sub>
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	0	1	0
4	0	1	0	0	0	1	1	0
5	0	1	0	1	0	1	1	1
6	0	1	1	0	0	1	0	1
7	0	1	1	1	0	1	0	0
8	1	0	0	0	1	1	0	0
9	1	0	0	1	1	1	0	1

10	1	0	1	0	1	1	1	1
11	1	0	1	1	2	2	2	0
12	1	1	0	0	2	2	1	0
13	1	1	0	1	1	0	1	0
14	1	1	1	0	1	0	1	1
15	1	1	1	1	1	0	0	1

Table 4.12.5

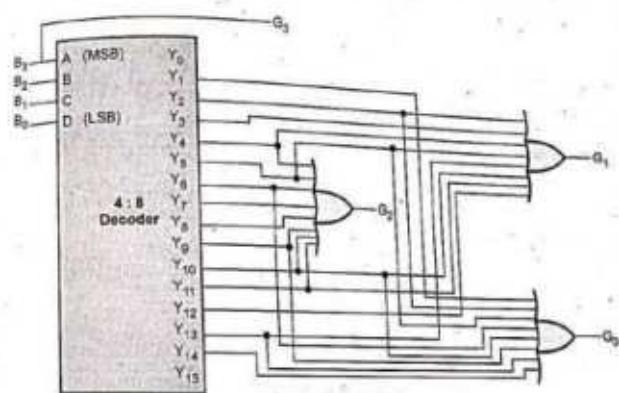
**Step 2 : implement Truth Table using decoder**

Fig. 4.12.11

**Examples for Practice**

**Example 4.12.8 :** Implement the following Boolean functions using decoder and OR gates :  
 $F_1(A, B, C, D) = \sum(2, 4, 7, 9)$   
 $F_2(A, B, C, D) = \sum(10, 13, 14, 15)$

**Example 4.12.9 :** Implement the logic circuit for full subtractor using decoder.**Example 4.12.10 :** Explain decoder (1 : 8) as a binary to gray code converter. Show your design.

#### 4.12.5 Implementation of Logic Function using IC 74138

IC 74138 is a 3:8 decoder with active-low outputs. In section 4.12.4 we have seen how to implement combinational logic function using decoder with active-high outputs. For active-high outputs, we use OR gate to implement the function. We know that bubbled-NAND is OR. Thus, for active-low outputs we have to use NAND gate to implement the function.

**Example 4.12.11** Implement full subtractor using a decoder (IC 74138) and write its truth table.

**Solution :** The truth table for full subtractor is as shown in Table 4.12.6.

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

Table 4.12.6 Truth table for full subtractor

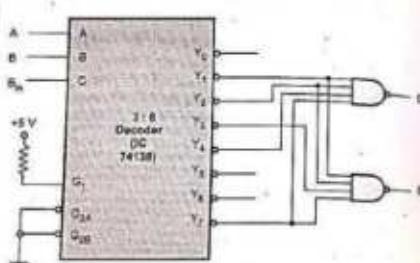


Fig. 4.12.12 Implementation of full subtractor using 3:8 decoder (IC 74138)

#### 4.12.6 Applications of Decoder

The uses of decoders are :

- Code converters
- Implementation of combinational circuits
- Address decoding
- BCD to 7-segment decoder

#### 4.12.7 Decoder ICs

IC Number	Function
74138	3:8 Decoder
74139	Dual 2:4 Decoder
7442	BCD to decimal decoder
7447	BCD to 7-segment decoder

Table 4.12.7 Decoder ICs

#### 4.12.8 Seven Segment Decoder

In most practical applications, seven segment displays are used to give a visual indication of the output states of digital ICs such as decade counters, latches etc. These outputs are usually in four bit BCD (binary coded decimal) form, and are thus not suitable for directly driving seven segment displays. The special BCD to seven segment decoder/driver ICs are used to convert the BCD signal into a form suitable for driving these displays. In this sections, we are going to study LED and LCD decoders/drivers for seven segment displays. Let us tabulate the segments activated during each digit display.

Digit	Segments Activated	Display
0	a, b, c, d, e, f	
1	b, c	
2	a, b, d, e, g	
3	a, b, c, d, g	
4	b, c, e, g	

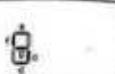
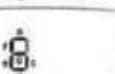
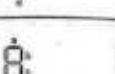
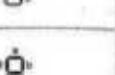
5	a, c, d, f, g	
6	a, c, d, e, f, g	
7	a, b, c	
8	a, b, c, d, e, f, g	
9	a, b, c, d, f, g	

Table 4.12.8

From the above Table 4.12.8 we can determine the truth table for BCD-to-7 segment decoder/driver. This truth table also depends on the construction of 7-segment display. If 7-segment display is common anode, the segment driver output must be active low to glow the segment. In case of common cathode type 7-segment display, the segment driver output must be active high to glow the segment. Table 4.12.9 and 4.12.10 show the truth tables for both BCD to 7 segment decoder/driver with common cathode display and with common anode display respectively.

Digit	A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1

Table 4.12.9 Truth table for BCD-to-common cathode 7-segment decoder/driver

Digit	A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	1	0	0	0	0	0	1
2	0	0	1	0	0	0	0	1	1	1	1
3	0	0	1	1	0	0	0	1	0	1	1
4	0	1	0	0	0	1	0	0	0	1	0
5	0	1	0	1	1	0	1	0	1	1	0
6	0	1	1	0	1	0	1	1	1	0	0
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	0

Table 4.12.10 Truth table for BCD-to-common anode 7-segment decoder/driver

Let us design the combinational circuit for common cathode 7-segment display/decoder.

## K-map simplification

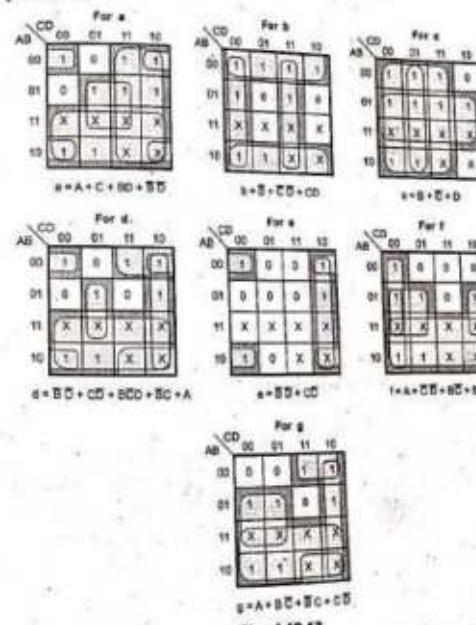


Fig. 4.12.13

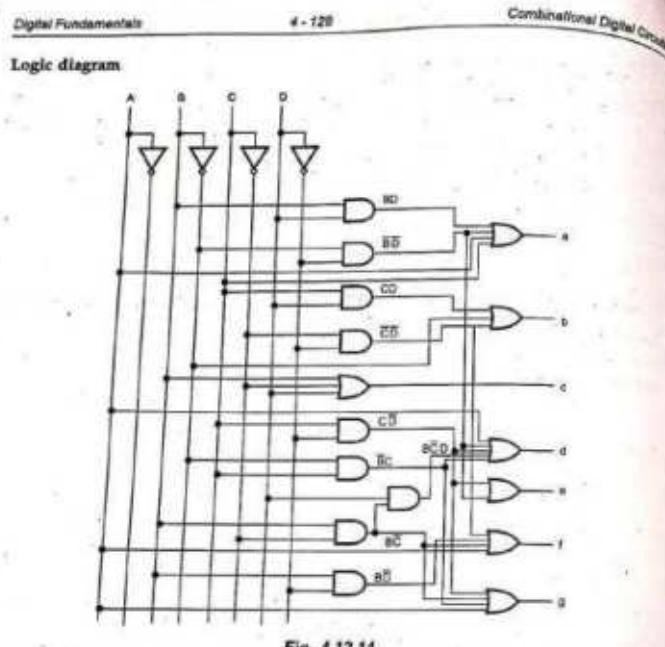


Fig. 4.12.14

**Example 4.12.12** Implement BCD-to-7 segment decoder for common anode using 4-to-16 decoder.

**Solution :** Referring the truth table from Table 4.12.9 we can implement BCD-to-7 segment decoder for common anode as shown in the Fig. 4.12.15.

#### 4.12.8.1 Basic Connection for Driving 7-Segment Displays

Fig. 4.12.16 and 4.12.17 show the basic connections of BCD to seven segment decoder/driver for common-anode and common-cathode displays, respectively. In both the circuits, current limiting resistors are placed in series with each display segment. Looking at the figures, we can observe that common anode decoder/driver sinks current whereas common-cathode decoder/driver source the current to each display segment.

Now we will see the practical decoder/driver ICs, their pin connections, functional description and features.

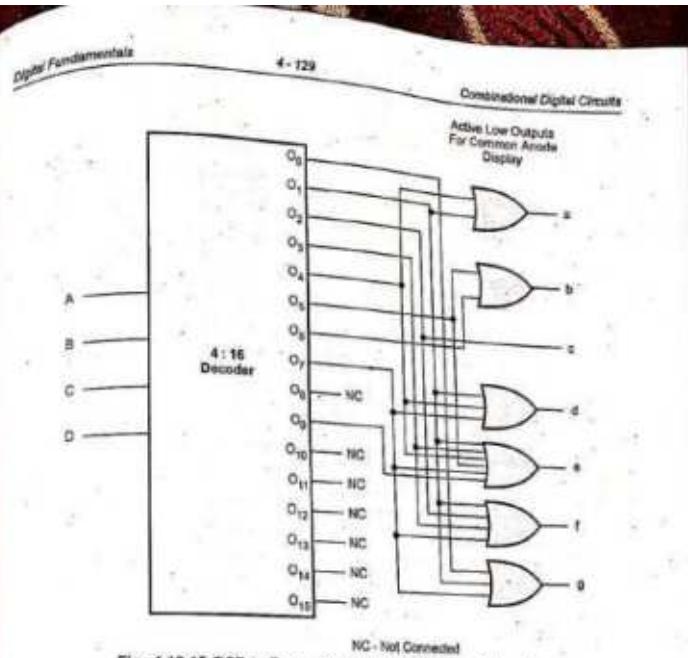


Fig. 4.12.15 BCD-to-7 segment decoder using 4 : 16 decoder

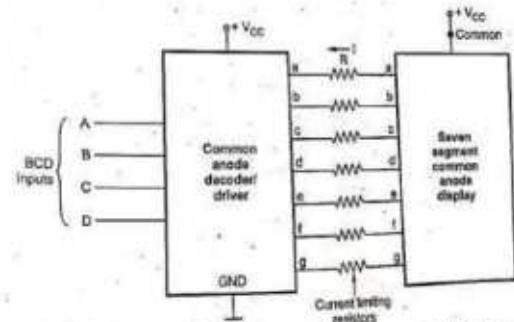


Fig. 4.12.16 Basic connections for driving common anode display

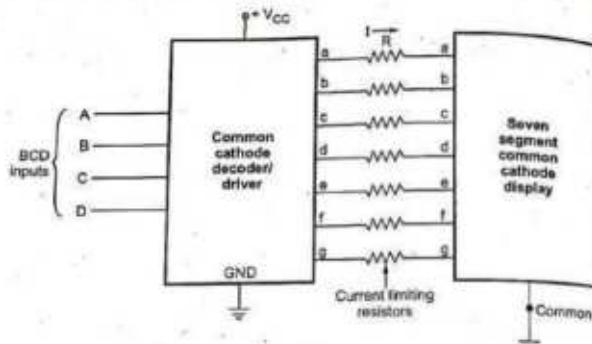


Fig. 4.12.17 Basic connections for driving common cathode display

## 4.12.8.2 IC 7446A, 7447A and 74LS47

The 7446A, 7447A and 74LS47 ICs accept four lines of BCD input data and give open collector outputs to drive the individual segments directly. Each segment output is guaranteed to sink 40 mA (24 mA for 74LS47) in the ON (LOW) state. As the outputs of these ICs sink the current, it is suitable to drive common anode seven segment displays.

Fig. 4.12.18 shows the pin diagram for 7446A, 7447A and 74LS47.

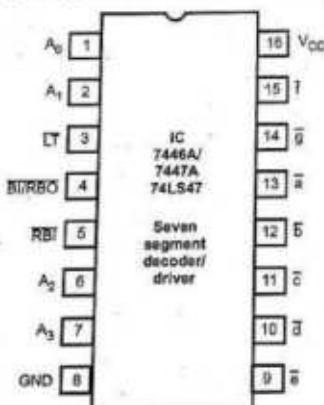


Fig. 4.12.18 Pin diagram for 7446A, 7447A and 74LS47

Pins  $A_0$ ,  $A_1$ ,  $A_2$  and  $A_3$  represent BCD inputs with  $A_0$  as a least significant bit (LSB) and  $A_3$  as a most significant bit (MSB). Pins  $\bar{a}$  through  $\bar{g}$  are the seven segment outputs. These are active low outputs, i.e. when segment output goes low (active state), segment is made 'ON'. The test lamp pin (LT) is provided to test whether all segments are working properly or not. When LT pin is held low with BIBO pin open or at logic high, IC drives all display terminals ON (active low). When the BL/RBI pin is pulled low, all outputs are blanked ; this pin also functions as a ripple-blanking output terminal. BL/RBI along with RBI can be used to provide ripple blanking feature discussed later.

Fig. 4.12.19 shows a circuit to drive a single, seven segment, common anode LED display. For common anode, when anode is connected to positive supply, a low voltage is applied to a cathode to turn it on. Here, BCD to seven segment decoder, IC 7447 is used to apply low voltages at cathodes according to BCD input applied to 7447. To limit the current through LED segments, resistors are connected in series with the segments. This circuit connection is referred to as a static or non-multiplex display because current is being passed through the display at all times.

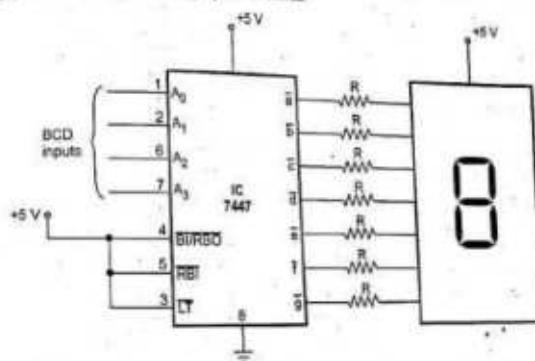


Fig. 4.12.19 Circuit for driving single seven segment display using 7446/7447

The value of the resistor in series with the segment can be calculated as follows :

We know,  $V_{CC} - \text{drop across LED segment} - IR = 0$

Drop across LED segment is nearly 1.5 V.

$$\begin{aligned} IR &= V_{CC} - 1.5 \text{ V} \\ &= 5 - 1.5 \text{ V} \\ &= 3.5 \text{ V} \end{aligned}$$

Each LED segment requires a current of between 5 and 30 mA to light. Let us assume that current through LED segment is 15 mA

$$\begin{aligned} R &= \frac{3.5 \text{ V}}{1.5 \text{ mA}} \\ &= 233 \Omega \end{aligned}$$

In practice, the voltage drop across the LED and the output of 7447 are not exactly predictable and the exact current through the LED is not critical as long as we do not exceed its maximum current rating. Therefore, a standard value  $220 \Omega$  can be used.

#### (a) Cascaded Non Multiplexed Displays

Several sets of seven segment displays and associated decoder/drivers ICs can be cascaded to make multi-digit display system. Fig. 4.12.20 shows the connection for cascaded multi-digit display system.

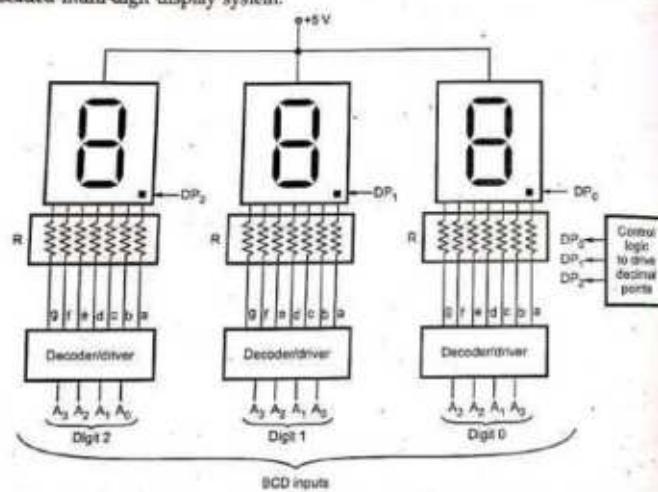


Fig. 4.12.20 Cascaded multi-digit non multiplexed display system

#### (b) Ripple Blanking in Multi-digit Displays

If the display system shown in Fig. 4.12.20 is used to display count of 8, then it actually gives the reading 008. Similarly, if it is used to display 0.2, it actually gives reading 0.20. These leading and trailing zeroes can be automatically blanking using  $\overline{\text{RBI}}$  and  $\overline{\text{RBO}}$  signals of the IC 7447 decoder/driver. The technique of blanking leading and trailing zeroes is called ripple blanking.

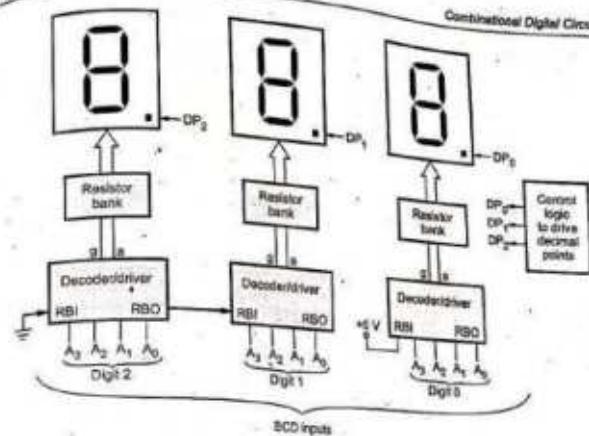


Fig. 4.12.21 Circuit for blanking leading zeroes

Fig. 4.12.21 and 4.12.22 show circuits to provide ripple blanking for leading zeroes and trailing zeroes.

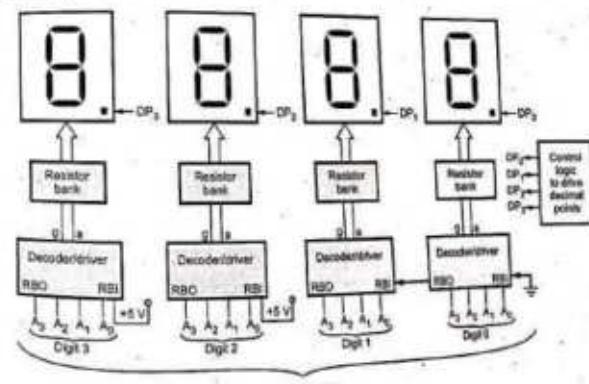


Fig. 4.12.22 Circuit for blanking trailing zeroes

To understand the working of signals RBI and RBO we will see the internal circuit for these two signals, as shown in Fig. 4.12.23.

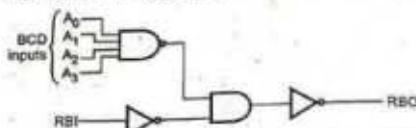


Fig. 4.12.23 Internal circuit for RBI and RBO signals

When RBI is low (activated) and all the BCD inputs are zero, then RBO goes low. Looking at Fig. 4.12.23 we can observe that when there is leading zero, digit 0 is blanked and it sets RBI zero for adjacent digit (digit 1), so that if BCD inputs for adjacent digit (digit 1) are zero, the digit is blanked.

In case of trailing zero the process of blanking starts from the right most digit. Here, digit 0 is blanked if all BCD inputs to the digit 0 are zero. If so, RBO for digit 0 goes low and therefore RBI input for digit 1 is low. Now if all BCD inputs of the digit 1 are zero then digit 1 is blanked.

#### (c) Multiplexed Displays

Until now we have seen static or non-multiplexed display circuits. These circuits work well for driving up to four LED digits. However, these circuits are not suitable for driving more LED digits, say 8 digits. When there are more number of digits, the first problem is a power consumption. For worst-case calculations, assume that all eight digits with all segments are lit. Therefore, worst case current required is

$$\begin{aligned} I &= 8 \text{ (digits)} \times 7 \text{ (segments)} \times 15 \text{ mA (current per segment)} \\ &= 840 \text{ mA} \end{aligned}$$

A second problem of the static approach is that each display digit requires a separate BCD to 7 segment decoder. To solve the problems of the static non-multiplexed display approach multiplexed display method is used. Fig. 4.12.24 shows the 4 seven segment displays connected using multiplexed method. Here, common anode seven segment LEDs are used.

Anodes are connected to + 5 V through transistors. Cathodes of all seven segments are connected in parallel and then to the output of 7447 IC through resistors. Looking at the Fig. 4.46, the question may occur in our mind that, "Aren't all of the digits going to display the same number?" The answer is that they would show the same number only if all the digits are turned-on at the same time. However, in multiplexed display the segment information is sent for all digits on the common lines (output lines of 7447), but only one display digit is turned on at a time. The PNP transistors connected in series with the common anode of each digit acts as an ON and OFF switch for that digit. Here is how the multiplexing process works.

The BCD code for digit 0 is first applied to the 7447. The 7447, BCD to seven segment decoder outputs the corresponding seven segment code on the segment bus lines. The transistor Q<sub>0</sub> connected to digit 0 is then turned on by corresponding control turned on. After 2 ms, digit 0 is turned-off by making all control inputs high. The BCD ON. After next 2 ms, digit 1 is turned-off and the process is repeated for digit 2 and digit 4. After completion of turn for each digit, all the digits are lit again in turn.

With 4 digits and 2 ms per digit we get back to digit 1 every 8 ms or about 125 times a second. This refresh rate is fast enough that, to our eyes and due to persistence of all digits will appear to be lit all the time.

In multiplexed display, the segment current is kept in between 40 mA to 60 mA so that they will appear as bright as they would, if not multiplexed. Even with this increased segment current, multiplexing gives a large saving in power and hardware components.

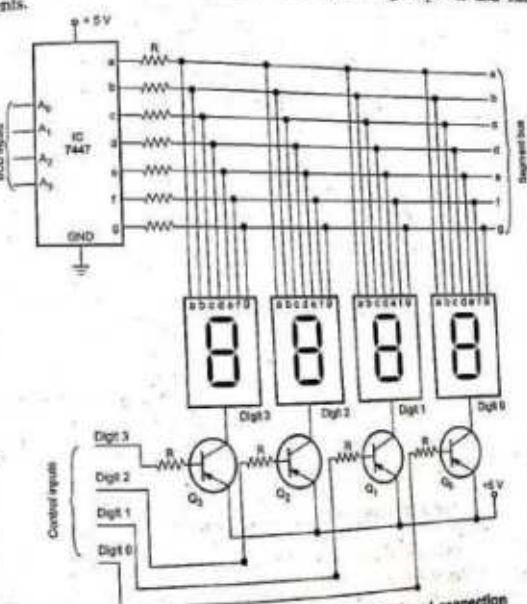


Fig. 4.12.24 Seven segment display in multiplexed connection

## 4.12.3 IC 74X49 Seven Segment Decoder

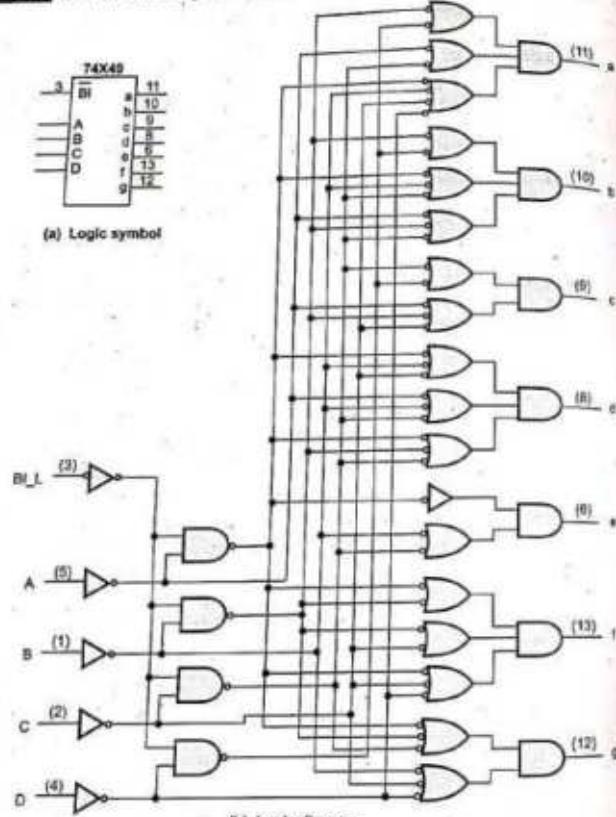


Fig. 4.12.25 The 74X49 seven segment decoder

The Fig. 4.12.25 shows the pinouts and the logic diagram for IC 74X49, 4-bit BCD to seven segment decoder. It has active-low blanking BL input. When BL input is activated, the outputs are logic 0. The Table 4.12.11 shows the truth table for IC 74X49.

Bi D C B A	Inputs							Outputs						
	a	b	c	d	e	f	g	a	b	c	d	e	f	g
0	X	X	X	X	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0	0	0	0
1	0	0	1	0	1	1	0	1	1	0	1	1	0	1
1	0	0	1	1	1	1	1	1	1	1	1	0	0	1
1	0	1	0	0	0	1	1	0	1	1	0	0	1	1
1	0	1	0	1	1	0	1	0	1	1	0	1	1	1
1	0	1	1	0	0	0	1	1	1	1	1	1	1	1
1	0	1	1	1	1	1	1	1	1	0	0	0	0	0
1	1	0	0	0	0	1	1	1	1	1	1	1	1	1
1	1	0	0	1	0	0	0	0	1	1	0	0	1	1
1	1	0	1	0	0	0	1	0	0	1	0	0	1	1
1	1	1	0	0	1	0	0	0	1	0	0	1	0	1
1	1	1	0	1	0	0	1	0	0	0	1	0	1	1
1	1	1	1	0	0	0	0	0	1	0	0	1	0	1
1	1	1	1	1	0	0	0	0	0	1	1	1	1	1

Table 4.12.11 Truth table for IC 74X49

## Review Questions

- What is decoder?
- Draw logic diagram of 3-line to 8-line decoder.
- What is difference between demultiplexer and decoder?
- Compare and contrast : Multiplexer and decoder.
- Give applications of decoder.
- Design full adder using suitable decoder.
- Design 4 × 16 decoder using two 3 × 8 decoder.

GHU : Semester-1B, Marks 4

GII : Semester-1B, Marks 7

## 4.13 Encoders

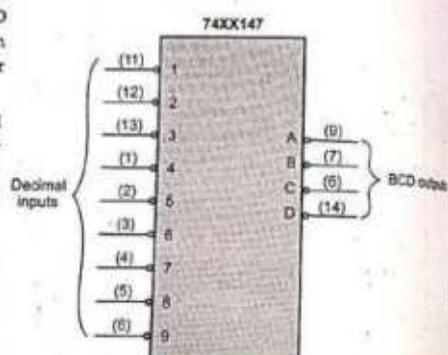
- An encoder is a digital circuit that performs the inverse operation of a decoder.
  - An encoder has  $2^n$  (or fewer) input lines and  $n$  output lines.
  - In encoder the output lines generate the binary code corresponding to the input value.
  - The Fig. 4.13.1 shows the general structure of the encoder circuit. As shown in the Fig. 4.13.1 the decoded information is presented as  $2^n$  inputs producing  $n$  possible outputs.

#### 4.13.1 Decimal to BCD Encoder

- The decimal to BCD encoder, usually has ten input lines and four output lines.
  - The decoded decimal data acts as an input for encoder and encoded BCD output is available on the four output lines.
  - The Fig. 4.13.2 shows the logic symbol for decimal to BCD encoder IC, IC 74XX147.
  - It has nine input lines and four output lines.
  - Both input and output lines are asserted active low.
  - It is important to note that there is no input line for decimal zero. When this condition occurs, all output lines are 1.

The function table for the 74XX147 is shown in Table 4.13.1.

Fig. 4.13.2 Logic symbol for 74XX147 (Decimal to BCD encoder)



Decimal value	Inputs	Outputs
0	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	D C B A
1	0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	
2	x 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1	
3	x x 0 1 1 1 1 1 1 1 1 1 1 1 1 1	
4	x x x 0 1 1 1 1 1 1 1 1 1 1 1 1	
5	x x x x 0 1 1 1 1 1 1 1 1 1 1 1	
6	x x x x x 0 1 1 1 1 1 1 1 1 1 1	
7	x x x x x x 0 1 1 1 1 1 1 1 1 1	
8	x x x x x x x 0 1 1 1 1 1 1 1 1	
9	x x x x x x x x 0 1 1 1 1 1 1 1 1	

\* indicates don't care condition

**Table 4.13.1** Truth table for  $d = 1$

**Example 4.13.1** Draw the interfacing diagram of ten key keypad interface to a digital system.

**Solution:**

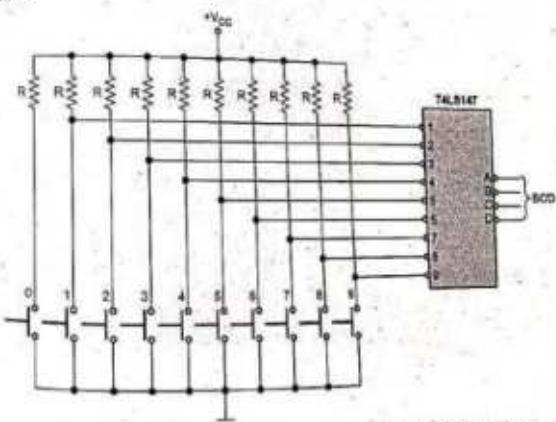


Fig. 4.100-7. An improved interface using decimal-to-BCD encoder

**4.13.2 Priority Encoder**

- A priority encoder is an encoder circuit that includes the priority function. In priority encoder, if two or more inputs are equal to 1 at the same time, the input having the highest priority will take precedence.
- Table 4.13.2 shows truth table of 4-bit priority encoder.
- Table 4.13.2 shows  $D_3$  input with highest priority and  $D_0$  input with lowest priority.
- When  $D_3$  input is high, regardless of other inputs output is ( $Y_1 Y_0 = 11$ ) 11.
- The  $D_2$  has the next priority. Thus, when  $D_3 = 0$  and  $D_2 = 1$ , regardless of other two lower priority input, output is 10.
- The output for  $D_1$  is generated only if higher priority inputs are 0, and so on.
- The output  $V$  (a valid output indicator) indicates, one or more of the inputs are equal to 1. If all inputs are 0,  $V$  is equal to 0, and the other two outputs ( $Y_1$  and  $Y_0$ ) of the circuit are not used.

**Example 4.13.2** Design a priority encoder circuit for 4-bit input.

Solution : Refer Table 4.13.2.

**K-map simplification**

		For $Y_1$				
		$D_2 D_3$	00	01	11	10
$D_0 D_1$	00	X	1	1	1	1
01	0	1	1	1	1	1
11	0	1	1	1	1	1
10	0	1	1	1	1	1

$$Y_1 = D_2 + D_3$$

		For $Y_0$				
		$D_2 D_3$	00	01	11	10
$D_0 D_1$	00	X	1	1	0	0
01	1	1	1	1	0	0
11	1	1	1	1	0	0
10	0	1	1	1	0	0

$$Y_0 = D_3 + D_1 \bar{D}_2$$

		For $V$				
		$D_2 D_3$	00	01	11	10
$D_0 D_1$	00	0	1	1	1	1
01	1	1	1	1	1	1
11	1	1	1	1	1	1
10	1	1	1	1	1	1

$$V = D_0 + D_1 + D_2 + D_3$$

Fig. 4.13.4

Inputs		Outputs				
$D_0$	$D_1$	$D_2$	$D_3$	$Y_1$	$Y_0$	$V$
0	0	0	0	0	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

Table 4.13.2 Truth table of 4-bit priority encoder

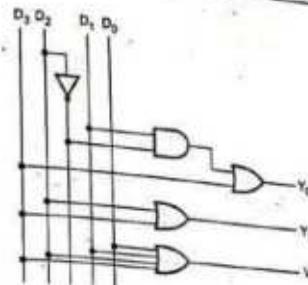
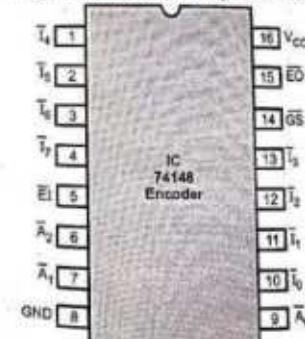
**Logic diagram**

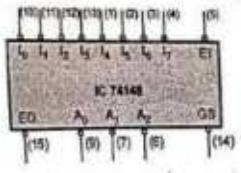
Fig. 4.13.5

**4.13.3 Priority Encoder IC (74XX148)**

- The IC 74XX148 is an 8-input priority encoder. It accepts data from eight active low inputs and provides a binary representation on the three active-low outputs.
- A priority is assigned to each input so that when two or more inputs are simultaneously active, the input with the highest priority is represented on the output.
- Input  $I_0$  has least priority and input  $I_7$  has highest priority.
- Fig. 4.13.6 shows pin diagram and logic symbol for IC 74XX148.



(a) Pin diagram



(b) Logic symbol

- $\bar{E}I$  is the active low Enable Input. A high on the  $\bar{E}I$  will force all outputs to the inactive (high) state and allow new data to settle without producing erroneous information at the outputs.
- A Group Signal ( $\bar{GS}$ ) is asserted when the device is enabled and one or more of the inputs to the encoder are active.
- Enable Output ( $\bar{EO}$ ) is an active low signal that can be used to cascade several priority encoder devices to form a larger priority encoding system.
- When all inputs are high, i.e., none of the input is active, the  $\bar{EO}$  goes low indicating that no priority event connected to the IC is present.
- In cascaded priority encoders,  $\bar{EO}$  is connected to the  $\bar{EI}$  input of the next lower priority encoder.
- The Table 4.13.3 shows the truth table for 74XX148 priority encoder.

EI	Inputs								Outputs			GS	EO
	0	1	2	3	4	5	6	7	$A_2$	$A_1$	$A_0$		
1	x	x	x	x	x	x	x	x	1	1	1	1	1
0	1	1	1	1	1	1	1	1	1	1	1	0	0
0	0	1	1	1	1	1	1	1	1	1	1	0	1
0	x	0	1	1	1	1	1	1	1	1	0	0	1
0	x	x	0	1	1	1	1	1	1	0	1	0	1
0	x	x	x	0	1	1	1	1	1	0	0	0	1
0	x	x	x	x	0	1	1	1	0	1	1	0	1
0	x	x	x	x	x	0	1	1	0	0	0	0	1
0	x	x	x	x	x	x	0	1	0	0	0	0	1

Table 4.13.3 Truth table for the 74XX148 priority encoder

#### 4.13.4 Octal to Binary Encoder

- Fig. 4.13.7 shows octal to binary encoder. It has eight inputs, one for each octal digit, and three outputs that generate the corresponding binary code.
- In encoders it is assumed that only one input has a value of 1 at any given time; otherwise the circuit is meaningless.
- Table 4.13.4 shows the truth table of octal-to-binary converter.

$D_7$	Inputs								Outputs		
	$D_6$	$D_5$	$D_4$	$D_3$	$D_2$	$D_1$	$D_0$	$A$	$B$	$C$	
1	0	0	0	0	0	0	0	0	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	1	0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0	1	0	0	1
0	0	0	0	0	1	0	0	0	1	0	1
0	0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	0	1	1	1	0	1

Table 4.13.4 Truth table of octal-to-binary encoder

TECHNICAL PUBLICATIONS™ - An up thrust for knowledge

TECHNICAL PUBLICATIONS™ - An up thrust for knowledge

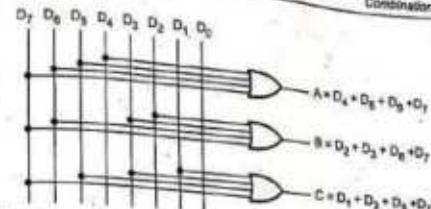


Fig. 4.13.7 Octal to binary encoder

The above circuit has one more ambiguity that when all inputs are 0s the outputs are 0s. The zero output can also be generated when  $D_0 = 1$ . This ambiguity can be resolved by providing an additional output that specifies the valid condition.

#### 4.13.5 Encoder ICs

IC Number	Function
74147	Decimal to BCD encoder
74148	8-input priority Encoder

Table 4.13.5 Encoder ICs

#### Review Questions

- What is encoder?
- Define priority encoder.

#### 4.14 Adders

Digital computers perform various arithmetic operations. The most basic operation, no doubt, is the addition of two binary digits. This simple addition consists of four possible elementary operations, namely,

$$\begin{aligned}0 + 0 &= 0 \\0 + 1 &= 1 \\1 + 0 &= 1 \\1 + 1 &= 10_2\end{aligned}$$

The first three operations produce a sum whose length is one digit, but when the last operation is performed sum is two digits. The higher significant bit of this result is called a carry, and lower significant bit is called sum. The logic circuit which performs

this operation is called a **half-adder**. The circuit which performs addition of three bits (two significant bits and a previous carry) is a **full-adder**.

#### 4.14.1 Half Adder

The half-adder operation needs two binary inputs : augend and addend bits; and two binary outputs : sum and carry. The truth table shown in Table 4.14.1 gives the relation between input and output variables for half-adder operation.

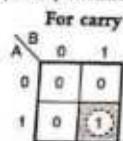
Inputs		Outputs	
A	B	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Table 4.14.1 Truth table for half-adder

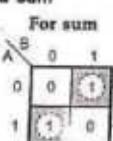


Fig. 4.14.1 Block schematic of half-adder

#### K-map simplification for carry and sum



$$\text{Carry} = AB$$



$$\text{Sum} = AB' + A'B \\ = A \oplus B$$

Fig. 4.14.2 Maps for half-adder

#### Logic diagram

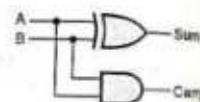


Fig. 4.14.3 Logic diagram for half-adder

#### Limitations of Half-Adder :

In multidigit addition we have to add two bits along with the carry of previous digit addition. Effectively such addition requires addition of three bits. This is not possible with half-adder. Hence half-adders are not used in practice.

#### Example 4.14.1 Draw half adder using NAND gates.

**Solution :** For half adder :

$$\begin{aligned} \text{Sum} &= AB' + A'B \\ &= \overline{AB} + \overline{A'B} \\ &= \overline{AB} \cdot \overline{A'B} \\ &= \overline{AB} \cdot \overline{A} \cdot \overline{B} \end{aligned}$$

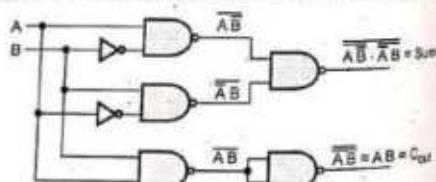


Fig. 4.14.4

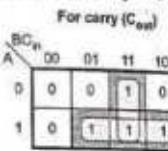
#### 4.14.2 Full Adder

A full-adder is a combinational circuit that forms the arithmetic sum of three input bits. It consists of three inputs and two outputs. Two of the input variables, denoted by A and B, represent the two significant bits to be added. The third input  $C_{in}$  represents the carry from the previous lower significant position. The truth table for full-adder is shown in Table 4.14.2.

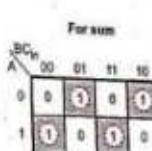
Inputs		Outputs		
A	B	$C_{in}$	Carry	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Table 4.14.2 Truth table for full-adder

#### K-map simplification for carry and sum



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



$$\text{Sum} = ABC_{in} + A'BC_{in} + AB'C_{in} + A'B'C_{in}$$

Fig. 4.14.5 Maps for full-adder

#### Logic diagram

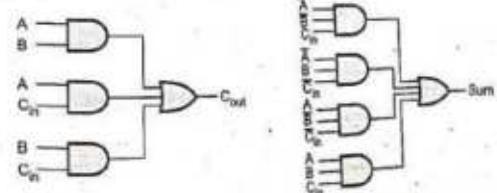


Fig. 4.14.7 Sum of product implementation of full-adder

The Boolean function for sum can be further simplified as follows :

$$\begin{aligned} \text{Sum} &= \overline{A} \overline{B} C_{in} + \overline{A} B \overline{C}_{in} + A \overline{B} \overline{C}_{in} + A B C_{in} \\ &= C_{in} (\overline{A} \overline{B} + AB) + \overline{C}_{in} (\overline{A} B + A \overline{B}) \\ &= C_{in} (A \oplus B) + \overline{C}_{in} (A \oplus B) \\ &= C_{in} (A \oplus B) + \overline{C}_{in} (A \oplus B) \\ &= C_{in} \oplus (A \oplus B) \end{aligned}$$

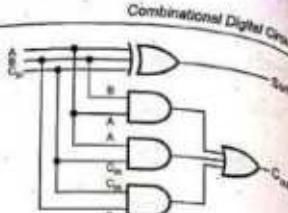


Fig. 4.14.8 Implementation of full-adder

With this simplified Boolean function circuit for full-adder can be implemented as shown in the Fig. 4.14.8.

#### Full adder using two half adders

A full-adder can also be implemented with two half-adders and one OR gate, as shown in the Fig. 4.14.9. The sum output from the second half-adder is the exclusive-OR of C<sub>in</sub> and the output of the first half-adder, giving

$$\begin{aligned} C_{out} &= AB + A C_{in} + BC_{in} = AB + A C_{in} (B + \overline{B}) + BC_{in} (A + \overline{A}) \\ &= AB + ABC_{in} + A \overline{B} C_{in} + ABC_{in} + \overline{A} BC_{in} = AB (1 + C_{in} + C_{in}) + A \overline{B} C_{in} + \overline{A} BC_{in} \\ &= AB + A \overline{B} C_{in} + \overline{A} BC_{in} = AB + C_{in} (A \overline{B} + \overline{A} B) = AB + C_{in} (A \oplus B) \end{aligned}$$

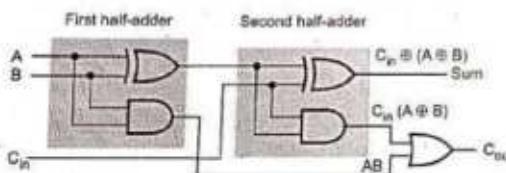


Fig. 4.14.9 Implementation of a full-adder with two half-adders and an OR gate

#### Review Questions

- Design a full adder.
- Define half adder and full adder.
- Draw a block diagram of half adder. Write truth table. Draw logic diagram.
- Write a truth table for half adder, reduce the equation using K-map and design half adder using logic gates.
- Define full adder. Draw logic circuit and truth table of full adder.
- Implement full adder using two half adders.
- Explain half and full adders in detail.

GTU : May-13, Dec-13, Marks 7

#### 4.15 Subtractors

The subtraction consists of four possible elementary operations, namely,

$$\begin{aligned} 0 - 0 &= 0 \\ 0 - 1 &= 1 \text{ with } 1 \text{ borrow} \\ 1 - 0 &= 1 \\ 1 - 1 &= 0 \end{aligned}$$

In all operations, each subtrahend bit is subtracted from the minuend bit. In case of second operation the minuend bit is smaller than the subtrahend bit, hence 1 is borrowed. Just as there are half and full-adders, there are half and full-subtractors.

#### 4.15.1 Half Subtractor

A half-subtractor is a combinational circuit that subtracts two-bits and produces their difference. It also has an output to specify if a 1 has been borrowed. Let us designate minuend bit as A and the subtrahend bit as B. The result of operation A - B for all possible values of A and B is tabulated in Table 4.15.1.

As shown in the Table 4.15.1, half-subtractor has two input variables and two output variables. The Boolean expression for the outputs of half-subtractor can be determined as follows.

#### K-map simplification for half-subtractor

##### For difference

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

$$\text{Difference} = \overline{AB} + \overline{A}B = A \oplus B$$

##### For borrow

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

$$\text{Borrow} = \overline{AB}$$

Fig. 4.15.1

**Limitations of half-subtractor :**

In multidigit subtraction, we have to subtract two bits along with the borrow of the previous digit subtraction. Effectively such subtraction requires subtraction of three bits. This is not possible with half-subtractor.

Logic diagram

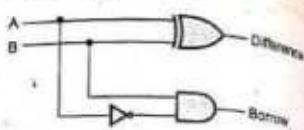


Fig. 4.15.2 Implementation of half-subtractor

**Example 4.15.1** Draw half-subtractor using NAND gates**Solution :** For half-subtractor :

$$\begin{aligned} \text{Difference} &= A \bar{B} + \bar{A} B \\ &= A \bar{B} + \bar{A} B \\ &= \overline{A B \cdot A B} \end{aligned}$$

$$\begin{aligned} \text{Borrow} &= \bar{A} B \\ &= \overline{\bar{A} B} \end{aligned}$$

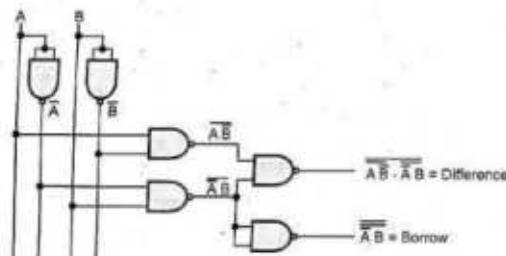
**Implementation :**

Fig. 4.15.3

**4.15.2 Full-Subtractor**

A full-subtractor is a combinational circuit that performs a subtraction between two bits, taking into account borrow of the lower significant stage. This circuit has three inputs and two outputs. The three inputs are  $A$ ,  $B$  and  $B_{in}$  denote the minuend, subtrahend, and previous borrow, respectively. The two outputs,  $D$  and  $B_{out}$ , represent the difference and output borrow, respectively. The Table 4.15.2 shows the truth table for full-subtractor.

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

Table 4.15.2 Truth table for full-subtractor

Logic diagram

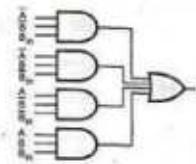
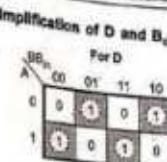


Fig. 4.15.5 Sum of product implementation of full-subtractor

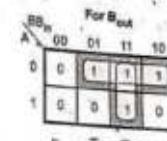
The Boolean function for  $D$  (difference) can be further simplified as follows :

$$\begin{aligned} D &= \bar{A} \bar{B} B_{in} + \bar{A} B \bar{B}_{in} + A \bar{B} \bar{B}_{in} + A B B_{in} \\ &= B_{in} (\bar{A} \bar{B} + A B) + \bar{B}_{in} (\bar{A} B + A \bar{B}) \\ &= B_{in} (A \oplus B) + \bar{B}_{in} (A \oplus B) \\ &= B_{in} \oplus (A \oplus B) \\ &= B_{in} \oplus (A \oplus B) \end{aligned}$$

With this simplified Boolean function Fig. 4.15.6 implementation of full-subtractor circuit for full-subtractor can be implemented as shown in the Fig. 4.15.6.

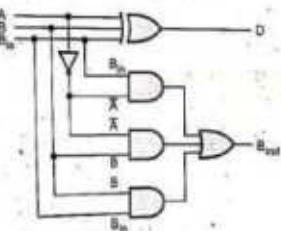


$$D = \bar{A} B B_{in} + \bar{A} \bar{B} B_{in} + A \bar{B} B_{in} + A B B_{in}$$



$$B_{out} = \bar{A} B_{in} + \bar{A} B + B B_{in}$$

Fig. 4.15.4 Maps for full-subtractor



A full subtractor can also be implemented with two half-subtractors and one OR gate, as shown in the Fig. 4.15.7. The difference output from the second half-subtractor is the exclusive-OR of  $B_{in}$ , and the output of the first half-subtractor, which is same as difference output of full-subtractor.

The borrow output for circuit shown in Fig. 4.15.6 can be given as,

$$\begin{aligned} B_{out} &= \overline{A} B_{in} + \overline{A} B + B \overline{B}_{in} \\ &= \overline{A} B_{in} (\overline{B} + \overline{B}) + \overline{A} B + B \overline{B}_{in} (A + \overline{A}) \\ &= \overline{A} B B_{in} + \overline{A} \overline{B} B_{in} + \overline{A} B + A B B_{in} + \overline{A} B B_{in} \\ &= \overline{A} B (B_{in} + 1 + B_{in}) + \overline{A} \overline{B} B_{in} + A B B_{in} \\ &= \overline{A} B + \overline{A} \overline{B} B_{in} + A B B_{in} \\ &= \overline{A} B + B_{in} (\overline{A} \overline{B} + A B) \\ &= \overline{A} B + B_{in} (A \oplus B) \end{aligned}$$

This Boolean function is same as borrow out of the full-subtractor. Therefore, we can implement full-subtractor using two half-subtractors and OR gate.

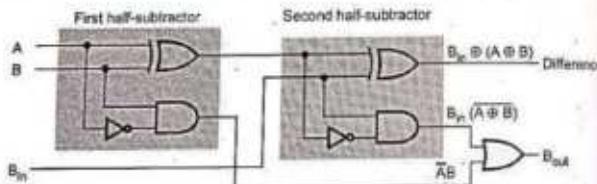


Fig. 4.15.7 Implementation of a full-subtractor with two half-subtractors and an OR gate

#### Review Questions

- Define half subtractor and full subtractor.
- Design a half-subtractor combinational circuit to produce the outputs, Difference and borrow.
- Explain the operation of a half subtractor with the help of logic diagram and truth table.
- Write the logic expressions for the difference and borrow of a half subtractor.
- Realize full-subtractor using K-map.
- Write an expression for borrow and difference in a full subtractor circuit.
- Design full-subtractor circuit and draw necessary truth tables.
- Explain how full subtractor can be designed by using two half subtractor circuits. Draw the circuit diagram.

#### 4.16 BCD Arithmetic

The digital systems handle the decimal number in the form of Binary Coded Decimal numbers (BCD). A BCD adder is a circuit that adds two BCD digits and produces a sum digit also in BCD. Here, we will see the implementation of addition of BCD numbers.

To implement BCD adder we require :

- 4-bit binary adder for initial addition
- Logic circuit to detect sum greater than 9 and
- One more 4-bit adder to add  $011_2$  in the sum if sum is greater than 9 or carry is 1.

The logic circuit to detect sum greater than 9 can be determined by simplifying the boolean expression of given truth table.

$Y = 1$  indicates sum is greater than 9. We can put one more term,  $C_{out}$  in the above expression to check whether carry is one. If any one condition is satisfied we add 6 ( $0110$ ) in the sum.

With this design information we can draw the block diagram of BCD adder, as shown in the Fig. 4.16.1 (b).

$S_3 S_2$	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	0	0	1	1

$$Y = S_3 S_2 + S_3 S_1$$

(a) K-map simplification

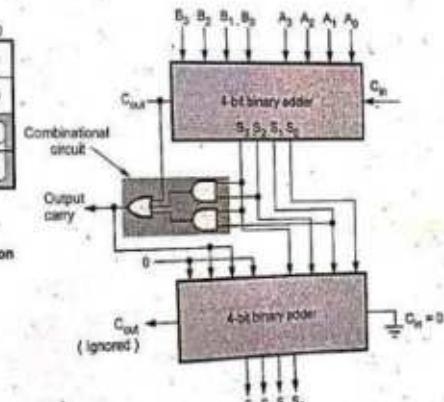


Fig. 4.16.1  
(b) Block diagram of BCD adder

As shown in the Fig. 4.16.1 (b), the two BCD numbers, together with input carry, are first added in the top 4-bit binary adder to produce a binary sum.

When the output carry is equal to zero (i.e. when sum  $\leq 9$  and  $C_{out} = 0$ ) nothing (zero) is added to the binary sum. When it is equal to one (i.e. when sum  $> 9$  or  $C_{out} = 1$ ), binary 0110 is added to the binary sum through the bottom 4-bit binary adder.

The output carry generated from the bottom binary adder can be ignored, since it supplies information already available at the output-carry terminal.

#### Example 4.16.1 Design an 8-bit BCD adder using 4-bit binary adder.

**Solution :** To implement 8-bit BCD adder we have to cascade two 4-bit BCD adders. In cascade connection carry output of the lower position (digit) is connected as a carry input of the higher position (digit). Fig. 4.16.2 shows the block diagram of 8-bit BCD adder.

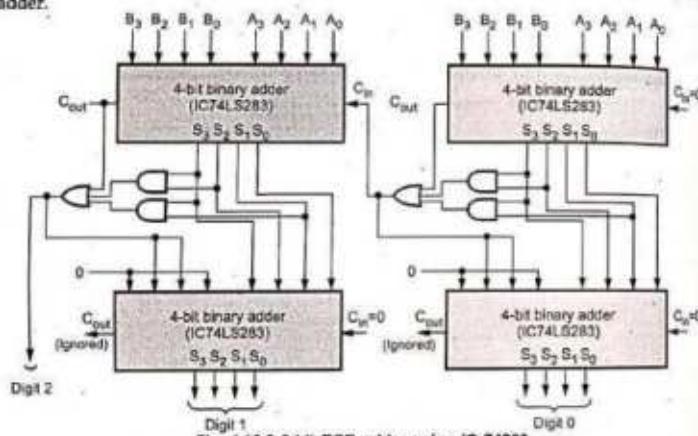


Fig. 4.16.2 8-bit BCD adder using IC 74283

#### Example 4.16.2 Design a BCD to Ex-3 code converter using binary parallel adder.

**Solution :**

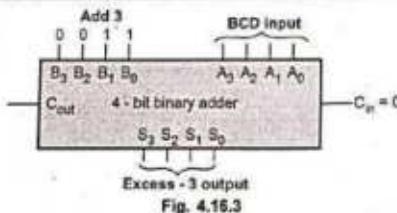


Fig. 4.16.3

TECHNICAL PUBLICATIONS™ An up beat for knowledge

#### Review Question

- With a suitable block diagram explain the operation of BCD adder.

#### 4.17 Carry Look Ahead Adder

GTU : Summer-15

One method of speeding up parallel addition process by eliminating inter stage carry delay is called look ahead carry addition. This method utilises logic gates to look at the lower-order bits of the augend and addend to see if a higher-order carry is to be generated. It uses two functions: carry generate and carry propagate.

Consider the circuit of the full-adder shown in Fig. 4.17.1. Here, we define two functions: carry generate and carry propagate.

$$\begin{aligned}P_1 &= A_1 \oplus B_1 \\G_1 &= A_1 B_1\end{aligned}$$

The output sum and carry can be expressed as

$$\begin{aligned}S_1 &= P_1 B_1 \\C_{i+1} &= G_1 + P_1 C_i\end{aligned}$$

$G_i$  is called a carry generate and it produces on carry when both  $A_i$  and  $B_i$  are one, regardless of the input carry.  $P_i$  is called a carry propagate because it is term associated with the propagation of the carry from  $C_i$  to  $C_{i+1}$ .

Now the Boolean function for the carry output of each stage can be written as follows.

$$\begin{aligned}C_2 &= G_1 + P_1 C_1 \\C_3 &= G_2 + P_2 C_2 = G_2 + P_2 (G_1 + P_1 C_1) = G_2 + P_2 G_1 + P_2 P_1 C_1 \\C_4 &= G_3 + P_3 C_3 = G_3 + P_3 (G_2 + P_2 G_1 + P_2 P_1 C_1) \\&= G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 C_1\end{aligned}$$

From the above Boolean function it can be seen that  $C_4$  does not have to wait for  $C_3$  and  $C_2$  to propagate; in fact  $C_4$  is propagated at the same time as  $C_2$  and  $C_3$ .

#### Review Question

- Describe the working of look-ahead carry adder.

GTU : Summer-15, Marks 7

TECHNICAL PUBLICATIONS™ An up beat for knowledge

**4.18 | Digital Comparator**

GTU : Dec.-13, Summer-15, 18, Winter-15, 18

- Here, the combinational circuit compares numerical values of two binary numbers and result is presented by the outputs of combinational circuits.

**Example 4.18.1** Design 2-bit comparator using gates.

GTU : Summer-15, Marks 2

Solution : The truth table for 2-bit is given in Table 4.18.1.

Inputs				Outputs		
$A_1$	$A_0$	$B_1$	$B_0$	$A > B$	$A = B$	$A < B$
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

Table 4.18.1

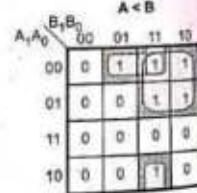
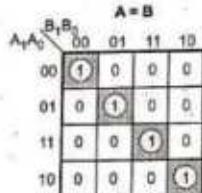
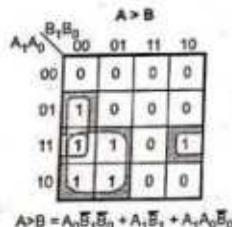
**K-map simplification**

Fig. 4.18.1

TECHNICAL PUBLICATIONS™ - An up beat for knowledge

TECHNICAL PUBLICATIONS™ - An up beat for knowledge

Fig. 4.18.2

**Note** We can also use outputs  $A = B$  and  $A < B$  to generate the third output  $A > B$ . This is illustrated in Fig. 4.18.3.

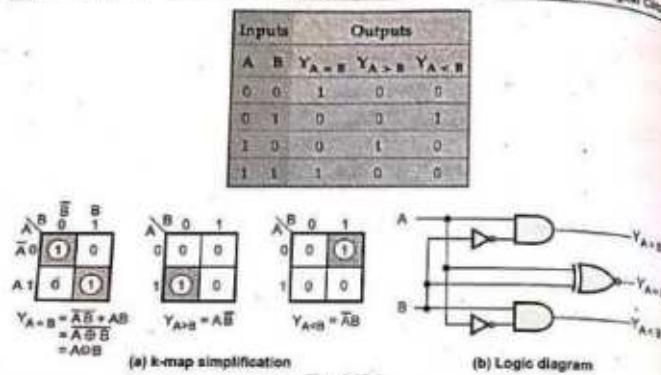


Fig. 4.18.3

**Example 4.18.2** Design a 7-bit comparator using basic gates.

GTU : Winter-16, Summer-16, Marks 3

Solution : Consider two one bit number A and B. The truth table is as shown.



Example 4.18.3 Design a 4-bit magnitude comparator.

Solution :

## Comparison Algorithm

 $(A = B)$ 

- Consider two 4-bit binary numbers A and B such that  $A = A_3 A_2 A_1 A_0$  and  $B = B_3 B_2 B_1 B_0$
  - Two numbers are said to be equal when  $A_3 = B_3, A_2 = B_2, A_1 = B_1$  and  $A_0 = B_0$
  - In binary, the equality of each pair of bits can be expressed logically with an exclusive-NOR function as
- $$x_i = A_i B_i + \bar{A}_i \bar{B}_i \quad \text{for } i = 0, 1, 2, 3$$
- where  $x_i = 1$  only if the pair of bits in position  $i$  are equal (i.e. if both are 1 or both are 0).
- Therefore, we can say that  $A = B$ , if  $x_0 \cdot x_1 \cdot x_2 \cdot x_3 = 1$

 $(A > B)$  or  $(A < B)$ 

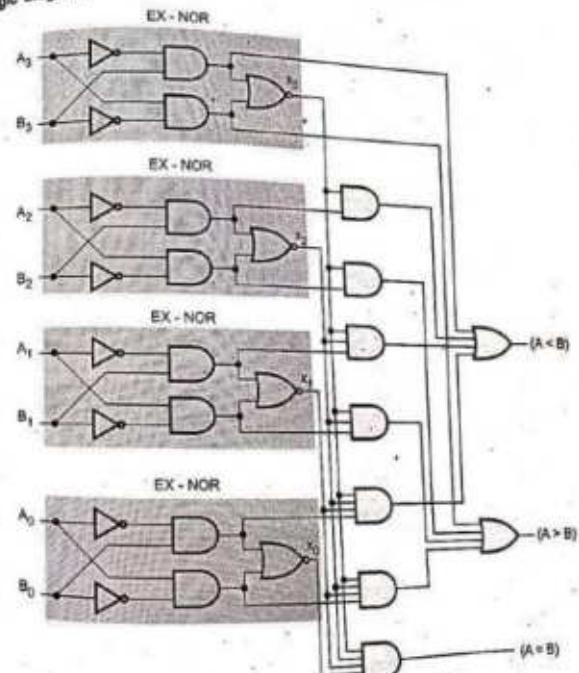
- To determine whether A is greater or less than B, it is necessary to inspect the relative magnitudes of pairs of significant digits, starting from the most significant position.

- If the two digits of a pair are equal, we compare the next lower significant pair of digits. The comparison continues until a pair of unequal digits is reached.
- If corresponding digits of A is 1 and that of B is 0, we say that  $A > B$ .
- If corresponding digits of A is 0 and that of B is 1, we say that  $A < B$ .
- The sequential expression to determine whether  $A > B$  or  $A < B$  are as given below :

$$(A > B) = A_3 \bar{B}_3 + x_3 A_2 \bar{B}_2 + x_3 x_2 A_1 \bar{B}_1 + x_3 x_2 x_1 A_0 \bar{B}_0$$

$$(A < B) = \bar{A}_3 B_3 + x_3 \bar{A}_2 B_2 + x_3 x_2 \bar{A}_1 B_1 + x_3 x_2 x_1 \bar{A}_0 B_0$$

## Logic diagram



Digital Fundamentals

4-153

Fig. 4.15.6 shows the block diagram of an  $n$ -bit comparator. It receives two  $n$ -bit numbers  $A$  and  $B$  as inputs and the outputs are  $A > B$ ,  $A = B$  and  $A < B$ . Depending upon the relative magnitudes of the two numbers, one of the outputs will be high.

IC 7485 is a 4-bit comparator. It can be used to compare two 4-bit binary words by grounding I ( $A < B$ ), and I ( $A > B$ ), and connector input I ( $A = B$ ) to  $V_{CC}$ . These ICs, can be cascaded to compare words of almost any length. Its 4-bit inputs are weighted ( $A_0 - A_3$ ) and ( $B_0 - B_3$ ), where  $A_3$  and  $B_3$  are the most significant bits.

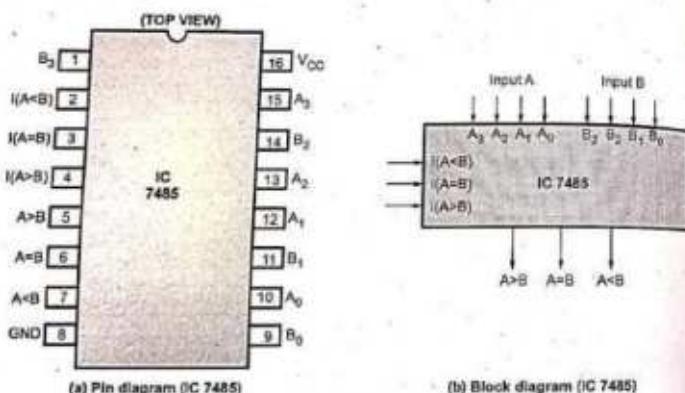


Fig. 4.18.7

Fig. 4.18.7 shows the logic symbol and pin diagram of IC 7485. The operation of IC 7485 is described in the function table, showing all possible logic conditions.

1

TECHNICAL SPECIFICATIONS<sup>2</sup>: An up-front breakdown

#### *International Classification*

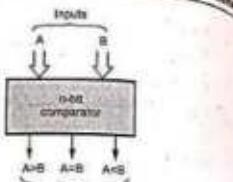


Fig. 4.18.6 Block diagram of n-bit comparison

卷之三

Comparing Inputs		Cascading Inputs			Outputs		
A	B	$(A > B)$	$(A = B)$	$(A < B)$	$A > B$	$A = B$	$A < B$
$A > B$	X	X	X				
$A = B$	1	0	0	1	0	0	0
	X	1	X	2	0	0	0
	0	0	1	0	1	0	0
	0	0	0	1	0	0	1
	1	0	1	0	0	0	0
$A < B$	X	X	X	0	0	0	1

Table 4.18.2 Function table for IC 7400

**Example 4.18.4** Design an 8-bit comparator using two 7485 IC's.

**Solution :** Fig. 4.18.5 shows an 8-bit comparator using two 74HC105 ICs.

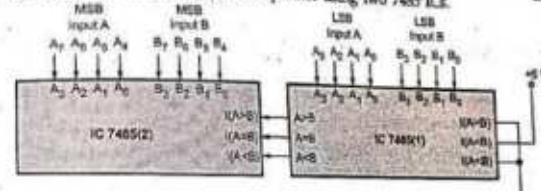


Fig. 4.18.3 8-bit comparator

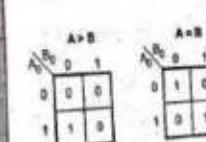
**Example 4.11.5** Design a 5-bit magnitude comparator using comparator IC 7485.

**Solution :**

### Truth Table

$A_0$	$B_0$	$I(A > B)$	$I(A = B)$	$I(A < B)$
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

### K-map simplification



10

$$\begin{aligned} (A > B) &= A_0 \bar{B}_0 \\ (A = B) &= \bar{A}_0 \bar{B}_0 + A_0 B_0 = A_0 \oplus B_0 \\ (A < B) &= \bar{A}_0 B_0 \end{aligned}$$

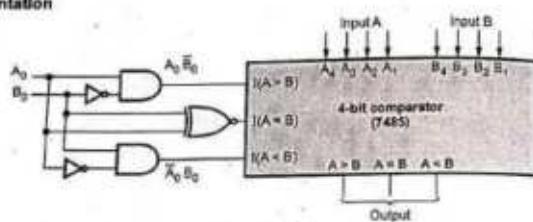
**Implementation**

Fig. 4.18.9

**Example 4.18.6** Explain how you will build a 24-bit magnitude comparator using 7485s.

**Solution :** By cascading six 4-bit comparator ICs as shown in Fig. 4.18.10 we can build a 24-bit magnitude comparator. (Refer Fig. 4.18.10 on next page).

**Review Questions**

- What is magnitude comparator ?
- Explain a 2-bit magnitude comparator with gate level circuit and truth table.
- Draw the schematic of a magnitude comparator and give its truth table.
- Write a note on IC 7485.
- With logic circuit explain the working of 4-bit magnitude comparator. **GTU : Dec-13, Marks 7**
- Draw and explain in brief pin diagram of 7485 four-bit magnitude comparator. **GTU : Winter-15, Marks 3**

**4.19 Parity Generator / Checker**

**GTU : May-14, Summer-15, 18**

- A parity bit is used for the purpose of detecting errors during transmission of binary information.
- A parity bit is an extra bit included with a binary message to make the number of 1s either odd or even.
- The message, including the parity bit is transmitted and then checked at the receiving end for errors. An error is detected if the checked parity does not correspond with the one transmitted.

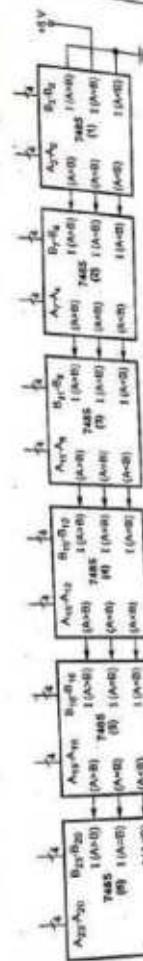


Fig. 4.18.10 24-bit comparator

- The circuit that generates the parity bit in the transmitter is called a parity generator and the circuit that checks the parity in the receiver is called a parity checker.
- In even parity, the added parity bit will make the total number of 1s an even amount.
- In odd parity, the added parity bit will make the total number of 1s an odd amount.
- Table 4.19.1 shows the 3-bit message with even parity and odd parity.

3-bit message			Odd parity bit	Even parity bit
A	B	C		
0	0	0	1	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	0	1

Table 4.19.1 Parity generator truth table for even and odd parity

**Example 4.19.1** Design an even parity generator with 3 message bits.

GTU : Summer-15, 16, Marks 7

Solution : Refer Table 4.19.1.

$$\begin{aligned}
 P &= \overline{ABC} + \overline{A}B\bar{C} + A\overline{B}\bar{C} + ABC = \overline{A}(\overline{B}C + B\bar{C}) + A(\bar{B}\bar{C} + BC) \\
 &= \overline{A}(B \oplus C) + A(B \otimes C) \\
 &= \overline{A}(B \oplus C) + A(\overline{B} \oplus \overline{C}) = A \oplus (B \oplus C)
 \end{aligned}$$

## K-map simplification

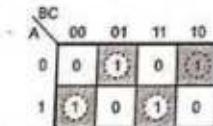


Fig. 4.19.1 (a) Simplification of even parity bit

## Logic diagram



Fig. 4.19.1 (b)

## Parity checker

- The three bits in the message together with the parity bit are transmitted to their destination, where they are applied to the parity checker circuit.
- The parity checker circuit checks for possible errors in the transmission.
- Since the information was transmitted with even parity, the four bits received must have an even number of 1s. An error occurs during the transmission if the four bits received have an odd number of 1s, indicating that one bit has changed in value during transmission.

Decimal equivalent	Four bits received				Parity error check
	P	A	B	C	
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	1
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	0
10	1	0	1	0	0
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	1
15	1	1	1	1	0

Table 4.19.2 Truth table for even parity checker

$$PEC = \overline{P} \overline{A} (\overline{B}C + BC) + \overline{P}A(\overline{B}\bar{C} + BC)$$

$$+ PA(\overline{B}C + B\bar{C}) + PA(\overline{B}\bar{C} + BC)$$

$$= \overline{P}A(B \oplus C) + \overline{P}A(\overline{B} \oplus C) + PA(B \oplus C) + PA(\overline{B} \oplus C)$$

$$= (\overline{P} \overline{A} + PA)(B \oplus C)$$

$$+ (\overline{P}A + PA)(\overline{B} \oplus C)$$

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

## K-map simplification

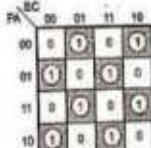


Fig. 4.19.2 (a)

## Logic diagram

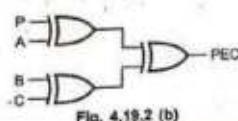


Fig. 4.19.2 (b)

**Example 4.19.2** Implement an even parity checker for a 4-bit data, using 8 + 1 mrs ad inverters.

**Solution :** Refer Table 4.19.2 for the truth table of 4-bit even parity checker.

## Implementation Table

	D <sub>0</sub>	D <sub>1</sub>	D <sub>2</sub>	D <sub>3</sub>	D <sub>4</sub>	D <sub>5</sub>	D <sub>6</sub>	D <sub>7</sub>	
P	0	1	2	3	4	5	6	7	
P	8	9	10	11	12	13	14	15	

Implementation

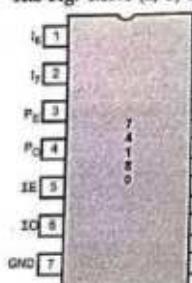
The implementation diagram shows a 5-to-1 multiplexer (MUX) with enable inputs P<sub>E</sub> and P<sub>O</sub>. The data inputs are D<sub>0</sub>, D<sub>1</sub>, D<sub>2</sub>, D<sub>3</sub>, and D<sub>4</sub>. The enable inputs are P<sub>E</sub> and P<sub>O</sub>. The outputs are labeled S<sub>0</sub>, S<sub>1</sub>, S<sub>2</sub>, Y = P<sub>E</sub>, and Y = P<sub>O</sub>. The enable inputs P<sub>E</sub> and P<sub>O</sub> are connected to the MUX's enable inputs through inverters.

Fig. 4.19.3

## IC 74180 : Parity Generator/Checker

The 74180 is a 9-bit parity generator or checker commonly used to detect errors in high speed data transmission or data retrieval systems. Both even and odd parity enable inputs and parity outputs are available for generating or checking parity on 8-bits.

The Fig. 4.19.4 (a, b) shows pin diagram and logic diagram for 74180.



(a) Pin diagram

(b) Logic diagram

Fig. 4.19.4

Table 4.19.3 shows the function table for 74180

Number of HIGH data Inputs (I <sub>0</sub> - I <sub>7</sub> )	Inputs		Outputs	
	P <sub>E</sub>	P <sub>O</sub>	$\Sigma E$	$\Sigma O$
Even	1	0	1	0
Odd	1	1	0	1
Even	0	1	0	1
Odd	0	0	1	0
X	1	1	1	1
X	0	0	0	0

Table 4.19.3 Function table for 74180

As shown in the Table 4.19.3, true active-HIGH or true active-LOW parity can be generated at both the even or odd outputs. True active-HIGH parity is established with Even parity enable input (P<sub>E</sub>) set HIGH and the Odd parity enable input (P<sub>O</sub>) set LOW. True active LOW parity is established when P<sub>E</sub> is LOW and P<sub>O</sub> is HIGH. When both are enable inputs are at the same logic level, both outputs will be forced to the opposite logic level.

Parity checking of a 9-bit word (8-bits plus parity) is possible by using the two enable inputs plus an inverter as the ninth data input. To check for true active-HIGH parity, the ninth data input is tied to the P<sub>O</sub> input and an inverter is connected between the P<sub>O</sub> and P<sub>E</sub> inputs as shown in the Fig. 4.19.5 (a). To check for true active-LOW parity, the ninth data input is tied to the P<sub>E</sub> input and an inverter is connected between the P<sub>E</sub> and P<sub>O</sub> inputs, as shown in the Fig. 4.19.5 (b).

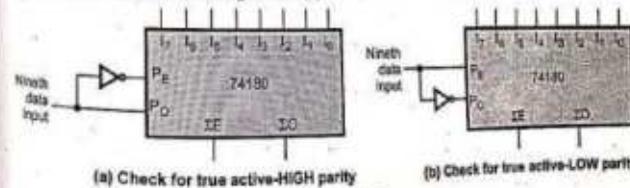


Fig. 4.19.5

Expansion to larger word sizes is accomplished by serially cascading the 74180 in 8-bit increments. The even and odd parity outputs of the first stages are connected to the corresponding P<sub>E</sub> and P<sub>O</sub> inputs, respectively, of the succeeding stage, as shown in Fig. 4.19.6.

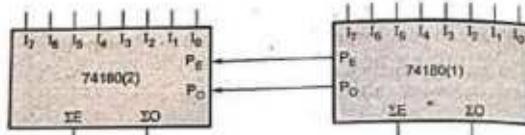


Fig. 4.19.6 Expansion to the larger word.

**Review Questions**

- Write a note on parity generator / parity checker. GTU : May-14, Marks 2
- Implement an odd parity checker for a 4-bit date, using 8 : 1 mux and inverters.
- Design 16-bit odd parity checker using 74180 ICs.
- Design 3-bit even parity generator circuit. GTU : Summer-15, 18, Marks 7

**4.20 Code Converters**GTU : May-12, 11

There is a wide variety of binary codes used in digital systems. Some of these code are Binary-Coded Decimal (BCD), Excess-3, Gray and so on. Many times it is required to convert one code to another.

Let us see the procedure to design code converters :

- Step 1 :** Write the truth table showing the relationship between input code and output code.
- Step 2 :** For each output code bit determine the simplified Boolean expression using K-map.
- Step 3 :** Realize the code converter using logic gates.

**Illustrative Examples**

**Example 4.20.1** Design a 4-bit binary to BCD converter.

**Step 1 :** Form the truth table relating binary and BCD code.

Input code : Binary code :  $B_3 B_2 B_1 B_0$  ( $B_0$  LSB)

Output code : BCD (Decimal) code :  $D_4 D_3 D_2 D_1 D_0$  ( $D_0$  LSB)

**Step 2 :** K-map simplification for each BCD output

		Binary code				BCD code			
$B_3$	$B_2$	$B_1$	$B_0$	$D_4$	$D_3$	$D_2$	$D_1$	$D_0$	
0	0	0	0	0	0	0	0	0	
0	0	0	1	0	0	0	0	1	
0	0	1	0	0	0	0	1	0	
0	0	1	1	0	0	0	1	1	
0	1	0	0	0	0	1	0	0	
0	1	0	1	0	0	1	0	1	
0	1	1	0	0	0	1	1	0	
0	1	1	1	0	0	1	1	1	
1	0	0	0	0	1	0	0	0	
1	0	0	1	0	1	0	0	1	
1	0	1	0	1	0	0	0	0	
1	0	1	1	1	0	0	0	1	
1	1	0	0	1	0	0	1	0	
1	1	0	1	1	0	0	1	1	
1	1	1	0	1	0	1	0	0	
1	1	1	1	1	1	0	1	1	

		For $D_0$				For $D_1$			
$B_3 B_2$	$B_1 B_0$	00	01	11	10	00	01	11	10
00	00	0	1	1	0	00	0	0	1
01	01	0	1	1	0	01	0	0	1
11	01	0	1	1	0	11	1	1	0
10	00	0	1	1	0	10	1	0	0

		For $D_2$				For $D_3$			
$B_3 B_2$	$B_1 B_0$	00	01	11	10	00	01	11	10
00	00	0	0	0	0	00	0	0	0
01	11	1	1	1	1	01	0	0	0
11	00	0	1	1	1	11	0	0	0
10	00	0	0	0	0	10	1	1	0

		For $D_4$			
$B_3 B_2$	$B_1 B_0$	00	01	11	10
00	00	0	0	0	0
01	01	0	0	0	0
11	11	1	1	1	1
10	00	0	0	1	1

Table 4.20.1 Truth table for binary to BCD conversion

$$D_2 = \overline{B}_3 B_2 + B_3 B_1$$

Fig. 4.20.1

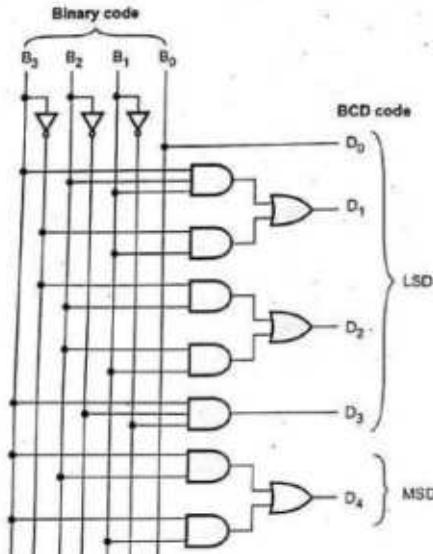
**Step 3 :** Realization of code converter

Fig. 4.20.2 Binary to BCD converter

**Example 4.20.2** Design a logic circuit to convert the 8421 BCD to Excess-3 code.

GTU : May-13, Marks 1

**Solution :**

**Step 1 :** Form the truth table relating BCD and Excess-3 code

Excess-3 code is a modified form of a BCD number. The Excess-3 code can be derived from the natural BCD code by adding 3 to each coded number. For example, decimal 12 can be represented in BCD as 0001 0010. Now adding 3 to each digit we get Excess-3 code as 0100 0101 (12 in decimal). With this information the truth table for BCD to Excess-3 code converter can be determined as shown in Table 4.20.2.

**Input code :** BCD code :  $D_3 D_2 D_1 D_0$  ( $D_0$  LSB)

**Output code :** Excess-3 code :  $E_3 E_2 E_1 E_0$  ( $E_0$  LSB)

Decimal	$D_3$	$D_2$	$D_1$	$D_0$	$E_2$	$E_1$	$E_0$
0	0	0	0	0	0	0	1
1	0	0	0	1	0	0	1
2	0	0	1	0	1	0	0
3	0	0	1	1	0	1	0
4	0	1	0	0	0	1	1
5	0	1	0	1	1	1	1
6	0	1	1	0	1	0	0
7	0	1	1	1	0	0	1
8	1	0	0	0	1	0	1
9	1	0	0	1	1	1	0

Table 4.20.2

**Step 2 :** K-map simplification for each Excess-3 code output.

$D_3 D_2$		For $E_3$			
$D_3$	$D_2$	00	01	11	10
00	0	0	0	0	0
01	0	1	1	1	1
11	X	X	X	X	X
10	1	1	X	X	X

$$\therefore E_3 = D_3 + D_2(D_0 + D_1)$$

$D_3 D_2$		For $E_2$			
$D_3$	$D_2$	00	01	11	10
00	0	1	1	1	1
01	1	0	0	0	0
11	X	X	X	X	X
10	0	1	X	X	X

$$\therefore E_2 = D_2 \bar{D}_1 \bar{D}_0 + \bar{D}_2 D_1 D_0 + D_1$$

$D_3 D_2$		For $E_1$			
$D_3$	$D_2$	00	01	11	10
00	1	0	1	0	0
01	1	0	1	0	0
11	X	X	X	X	X
10	1	0	X	X	X

$$E_1 = \bar{D}_1 \bar{D}_0 + D_1 D_0 \\ = D_1 \oplus D_0$$

$D_3 D_2$		For $E_0$			
$D_3$	$D_2$	00	01	11	10
00	1	0	0	1	0
01	1	0	0	0	1
11	X	X	X	X	X
10	1	0	X	X	X

$$E_0 = \bar{D}_3$$

Fig. 4.20.3

**Step 3 :** Realization of code converter.

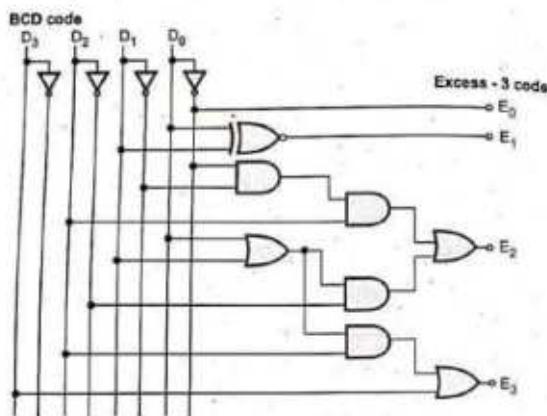


Fig. 4.20.4 BCD to excess-3 code converter

**Example 4.20.3** Design a logic circuit to convert BCD to gray code.

**Solution :**

**Step 1 :** Form the truth table relating BCD and gray code.

Input code : BCD code :  $D_3 D_2 D_1 D_0$  ( $D_0$ , LSB)

Output code : Gray code :  $G_3 G_2 G_1 G_0$  ( $G_0$ , LSB)

Table 4.20.3 shows truth table for BCD to gray code converter.

BCD code				Gray code			
$D_3$	$D_2$	$D_1$	$D_0$	$G_3$	$G_2$	$G_1$	$G_0$
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0

Table 4.20.3 Truth table for BCD to gray code converter

**Step 2 :** K-map simplification

For $G_3$				
$D_3 D_2$	00	01	11	10
00	0	1	0	1
01	0	1	0	1
11	X	X	X	X
10	0	1	X	X

$G_3 = \overline{D}_1 D_0 + D_2 \overline{D}_0$   
 $= D_1 \oplus D_0$

For $G_2$				
$D_3 D_2$	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	X	X	X	X
10	1	1	X	X

$G_2 = D_2 + D_3$

For $G_1$				
$D_3 D_2$	00	01	11	10
00	0	0	1	1
01	1	1	0	0
11	X	X	X	X
10	0	0	X	X

$G_1 = D_2 D_1 + \overline{D}_2 D_1$   
 $= D_2 \oplus D_1$

For $G_0$				
$D_3 D_2$	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	X	X	X	X
10	1	1	X	X

$G_0 = D_3$

**Step 3 :** Realization of code converter

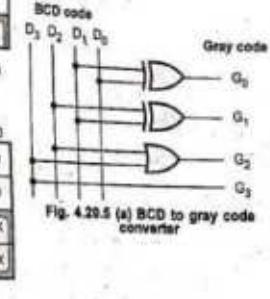


Fig. 4.20.5 (a) BCD to gray code converter

Fig. 4.20.5

**Example 4.20.4** Design and implement a 8421 to Gray code converter. Realize the converter using only NAND gates.

GTU : May-12, Marks 5

**Solution :**

**Step 1 :** Form the Truth table relating 8421 binary code and Gray code

Input code : Binary code :  $B_3 B_2 B_1 B_0$

Output code : Gray code :  $G_3 G_2 G_1 G_0$

Decimal	Binary code				Gray code			
	$B_3$	$B_2$	$B_1$	$B_0$	$G_3$	$G_2$	$G_1$	$G_0$
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	0
3	0	0	1	1	1	0	1	1
4	0	1	0	0	0	0	1	1
5	0	1	0	1	0	1	0	1
6	0	1	1	0	0	1	0	0

TECHNICAL PUBLICATIONS™ - An up thrust for knowledge

7	0	1	1	1	0	1	0	0
8	1	0	0	0	1	1	0	0
9	1	0	0	1	1	1	0	1
10	1	0	1	0	1	1	1	1
11	1	0	1	1	1	1	1	0
12	1	1	0	0	1	0	1	0
13	1	1	0	1	1	0	1	1
14	1	1	1	0	1	0	0	1
15	1	1	1	1	0	0	0	0

Table 4.20.4

**Step 2 :** K-map simplification for each gray code output

		For $G_0$				For $G_1$			
		$B_3B_2$		00 01 11 10		$B_3B_2$		00 01 11 10	
00	0	0	1	0	1	00	0	0	11
01	0	1	0	0	1	01	1	1	00
11	0	1	0	1	1	11	1	1	00
10	0	1	0	1	0	10	0	0	11

		For $G_2$				For $G_3$			
		$B_3B_2$		00 01 11 10		$B_3B_2$		00 01 11 10	
00	0	0	0	0	0	00	0	0	0
01	1	1	1	1	0	01	0	0	0
11	0	0	0	0	0	11	1	1	1
10	1	1	1	1	1	10	1	1	1

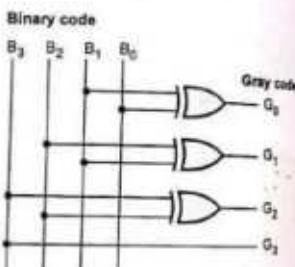
$$G_0 = B_1\bar{B}_0 + \bar{B}_1B_0 \\ = B_1 \oplus B_0$$

		For $G_2$				For $G_3$			
		$B_3B_2$		00 01 11 10		$B_3B_2$		00 01 11 10	
00	0	0	0	0	0	00	0	0	0
01	1	1	1	1	0	01	0	0	0
11	0	0	0	0	0	11	1	1	1
10	1	1	1	1	1	10	1	1	1

$$G_2 = \bar{B}_3B_2 + B_3\bar{B}_2 = B_3 \oplus B_2$$

Fig. 4.20.6

Fig. 4.20.7 Binary to gray code converter



**Step 3 : Realization of code converter using XOR-gates**

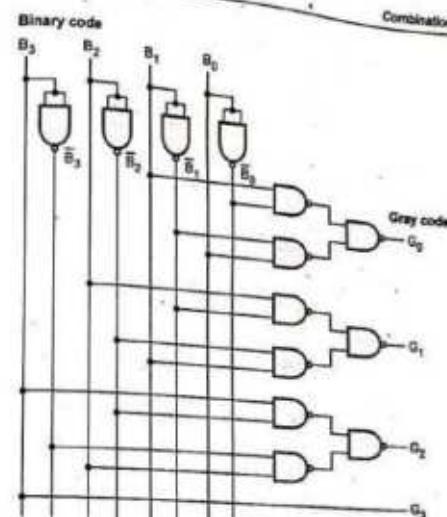
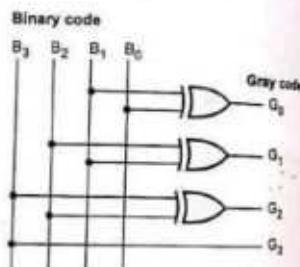


Fig. 4.20.8 Binary to Gray code converter

#### Example 4.20.5 Design a Gray to BCD code converter.

**Solution :** The Table 4.20.5 shows truth table for gray to BCD code converter.

Gray code				BCD code			
$G_3$	$G_2$	$G_1$	$G_0$	$D_3$	$D_2$	$D_1$	$D_0$
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	1	1	0	0	1
0	0	1	0	0	0	0	1
0	1	1	0	0	0	1	0
0	1	1	1	0	0	1	1
0	1	0	1	0	1	0	1
0	1	0	0	0	0	1	1

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

Table 4.20.5 Gray to BCD code converter

**Kmap-simplification**

		For D <sub>0</sub>					
		G <sub>3</sub>	G <sub>2</sub>	00	01	11	10
00		0	1	0	1		
01		1	0	1	0		
11		0	1	0	1		
10		1	0	1	0		

$$\begin{aligned}D_0 &= \overline{G}_3 \overline{G}_2 \overline{G}_1 G_0 + \overline{G}_3 \overline{G}_2 G_1 \overline{G}_0 + \\&\quad \overline{G}_3 G_2 \overline{G}_1 \overline{G}_0 + \overline{G}_3 G_2 G_1 G_0 + \\&\quad G_3 \overline{G}_2 \overline{G}_1 \overline{G}_0 + G_3 \overline{G}_2 G_1 \overline{G}_0 + \\&\quad G_3 G_2 \overline{G}_1 \overline{G}_0 + G_3 G_2 G_1 G_0 = \\&= \overline{G}_3 \overline{G}_2 (G_1 \oplus G_0) + \overline{G}_3 G_2 (\overline{G}_1 \oplus G_0) + \\&\quad G_3 \overline{G}_2 (G_1 \oplus G_0) + G_3 G_2 (G_1 \oplus G_0) \\&= (\overline{G}_3 \oplus G_2) (G_1 \oplus G_0) + (G_3 \oplus G_2) (\overline{G}_1 \oplus G_0) \\&= G_3 \oplus G_2 \oplus G_1 \oplus G_0\end{aligned}$$

		For D <sub>1</sub>					
		G <sub>3</sub>	G <sub>2</sub>	00	01	11	10
00		0	0	0	1	1	
01		1	1	0	0	0	
11		0	0	0	0	0	
10		0	0	1	1		

$$D_1 = \overline{G}_3 G_2 \overline{G}_1 + \overline{G}_2 G_1$$

		For D <sub>2</sub>					
		G <sub>3</sub>	G <sub>2</sub>	00	01	11	10
00		0	0	0	0	0	
01		1	1	0	0	1	
11		0	0	0	0	0	
10		1	1	0	0	0	

$$D_2 = \overline{G}_3 G_2 + G_3 \overline{G}_2$$

		For D <sub>3</sub>					
		G <sub>3</sub>	G <sub>2</sub>	00	01	11	10
00		0	0	0	0	0	
01		0	0	0	0	0	
11		1	1	0	0	0	
10		0	0	0	0	0	

$$D_3 = G_3 G_2 \overline{G}_1$$

		For D <sub>4</sub>					
		G <sub>3</sub>	G <sub>2</sub>	00	01	11	10
00		0	0	0	0	0	
01		0	0	0	0	0	
11		0	0	1	1		
10		1	1	1	1		

$$D_4 = G_3 \overline{G}_2 + G_3 G_1$$

Fig. 4.20.9

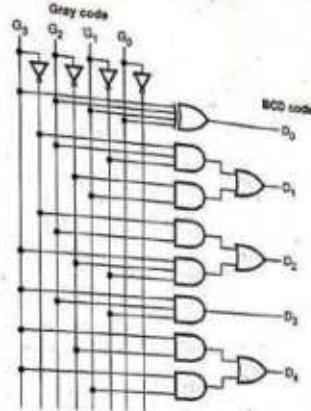


Fig. 4.20.10

**Example 4.20.6** Design 4 bit excess-3 to BCD code converter and implement using logic gates.

**Solution :** 4-bit excess - 3 to BCD code converter :

Decoder	Inputs				Outputs			
	E <sub>3</sub>	E <sub>2</sub>	E <sub>1</sub>	E <sub>0</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>
3	0	0	1	1	0	0	0	0
4	0	1	0	0	0	0	0	1
5	0	1	0	1	0	0	1	0
6	0	1	1	0	0	1	0	0
7	0	1	1	1	0	1	0	1
8	1	0	0	0	0	0	1	0
9	1	0	0	1	0	1	1	1
10	1	0	1	0	0	1	0	0
11	1	0	1	1	1	0	0	1
12	1	1	0	0	1	0	0	0

## K-map simplification

$E_3 E_2$	00	01	11	10
00	X	X	0	X
01	0	0	0	0
11	1	X	X	X
10	0	0	1	0

$$D_1 = E_3 E_2 + E_3 \bar{E}_2$$

$E_3 E_2$	00	01	11	10
00	X	X	0	X
01	0	0	1	0
11	0	X	X	X
10	1	1	0	1

$$D_2 = E_3 E_2, E_3 + \bar{E}_3 \bar{E}_2 + E_3 E_2$$

$E_3 E_2$	00	01	11	10
00	X	X	0	X
01	0	1	0	1
11	0	X	X	X
10	1	1	0	1

$$D_3 = E_3 E_2 + \bar{E}_3 E_2$$

$E_3 E_2$	00	01	11	10
00	X	X	0	X
01	1	0	0	1
11	1	X	X	X
10	1	0	0	1

$$D_4 = \bar{E}_3$$

Fig. 4.20.11

## Implementation using logic gates :

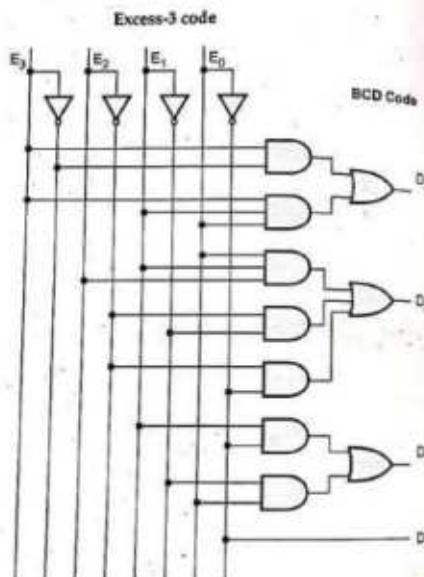


Fig. 4.20.12

## Example for Practice

**Example 4.20.7 :** Design a logic circuit to convert gray code to binary code.

$$\begin{aligned} \text{(Ans.) } B_0 &= G_3 \oplus G_2 \oplus G_1 \oplus G_0 & B_1 &= G_3 \oplus G_2 \oplus G_1 \\ B_2 &= G_3 \oplus G_2 & B_3 &= G_3 \end{aligned}$$

## 4.21 ALU and Elementary ALU Design

## 4.21.1 Design of Arithmetic Unit

The Arithmetic and Logic Unit (ALU) performs all the necessary adding, subtracting, shifting, and logical operations. It requires one or two operands upon which to operate and produces a result. The complexity of an ALU is determined by the way in which its subtraction, as well as word based logic operations, can be realized. Simple ALUs that perform fixed point addition and circuits. ALUs that also perform multiplication and division can be constructed using combinational circuits discussed in preceding sections for them. Let us see the simple ALU design with combinational circuits.

The basic component of the arithmetic section of a simplest ALU is a parallel adder. A parallel adder can be constructed using number of full-adder circuits connected in cascade. Let us see the implementation of various functions of arithmetic unit using parallel adder. Table 4.21.1 shows these implementations.

Sr. No	Function	Implementation
1.	$Y \leftarrow A$	<p>A: Input D: Input <math>C_{out} \leftarrow</math> Parallel Adder <math>C_n = 0</math> <math>Y = A</math></p>
2.	$Y \leftarrow A + 1$	<p>A: Input D: Input <math>C_{out} \leftarrow</math> Parallel Adder <math>C_n = 1</math> <math>Y = A + 1</math></p>
3.	$Y \leftarrow A + B$	<p>A: Input B: Input <math>C_{out} \leftarrow</math> Parallel Adder <math>C_n = 0</math> <math>Y = A + B</math></p>

		Combinational Digital Circuits
4.	$Y \leftarrow A + B + 1$	
5.	$Y \leftarrow A + \bar{B}$	
6.	$Y = A + \bar{B} + 1$	
7.	$Y = A - 1$	
8.	$Y = A$	

Table 4.21.1 Implementation of basic arithmetic operations

Looking at Table 4.21.1 we can realize that by selecting proper B and  $C_{in}$  inputs we can implement basic arithmetic operations. There are four possible B inputs: true complement, all 1s and all 0s and two possible  $C_{in}$  inputs 0 and 1.

TECHNICAL PUBLICATIONS™ - An up front for knowledge

Digital Fundamentals  
4 - 179  
Combinational Digital Circuits

Referring Table 4.21.2, we can write truth table for B input with respect to  $S_0$  and  $S_1$  signals as follows :

$S_1$	$S_0$	$B'$
0	0	0
0	1	1
1	0	1
1	1	1

Table 4.21.2 Truth table

Note -  $B'$  indicates modified B input

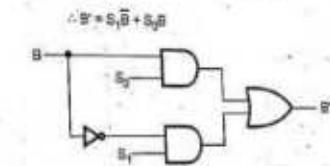
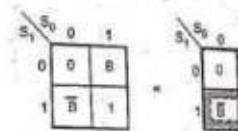


Fig. 4.21.1 K-map simplification and logic diagram of modified B input

#### K-map simplification

Using the logic diagram of modified B input we can draw the logic diagram of arithmetic circuit, as shown in Fig. 4.21.2.

TECHNICAL PUBLICATIONS™ - An up front for knowledge

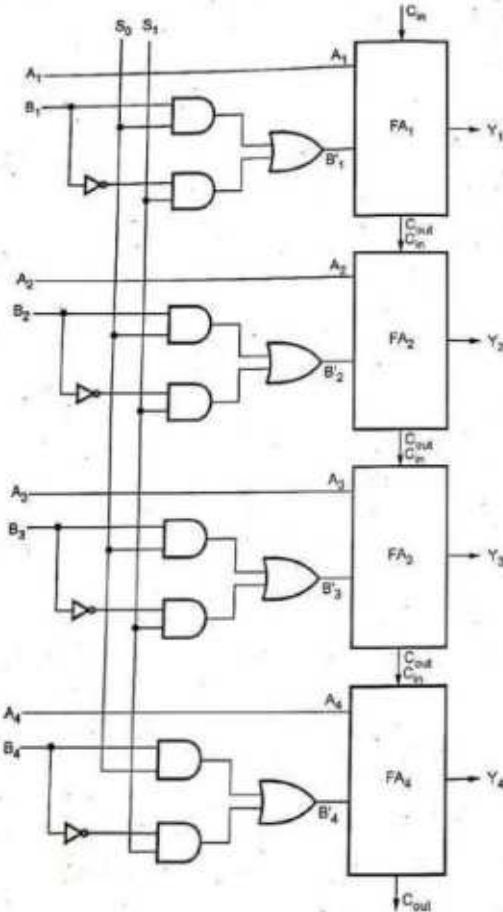


Fig. 4.21.2 Logic diagram of arithmetic circuit

TECHNICAL PUBLICATIONS™ - An up thrust for knowledge

## 4.21.2 Design of Logic Circuit

The design of logic circuit is comparatively simple than the design of arithmetic circuit. Fig. 4.21.3 shows the function table and the logic diagram for logic circuit.

S <sub>1</sub>	S <sub>0</sub>	Output	Function
0	0	Y = A ∨ B	OR
0	1	Y = A ⊕ B	XOR
1	0	Y = A ∧ B	AND
1	1	Y = Ā	NOT

a) Function table

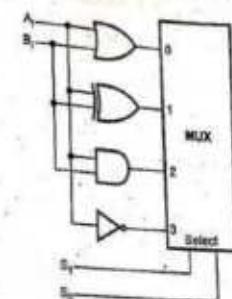


Fig. 4.21.3 Logic circuit

Note : The logic diagram shown in Fig. 4.21.3 (b) is repeated n times for n-bit logic circuit. Therefore, inputs A and B, and output Y are shown with subscript i, where i = 1, 2, ..., n.

We have seen the design of arithmetic and logic circuits for a simple ALU. By combining both circuits with the help of multiplexer we can get the arithmetic and logic circuit, as shown in the Fig. 4.21.4. When S (mode select) is 0, the output of arithmetic circuit is transferred as a final output; otherwise the output of logic circuit is transferred as a final output.

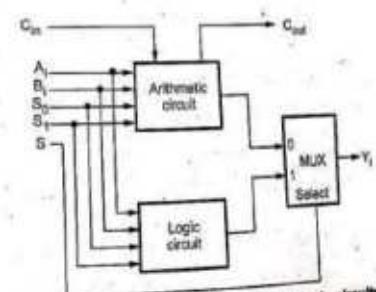


Fig. 4.21.4 Combining arithmetic and logic circuits

### 4.21.3 Combining Arithmetic and Logic Unit

The design of ALU in above mentioned way is the simplest and straight forward. A more efficient ALU can be implemented if we investigate the possibility of generating logic operations in an already available arithmetic circuit.

We know that carry input is not required in the logic circuits. Hence, when logic operation is selected ( $S = 1$ ), the carry input must be zero. This gives us the output sum in full-adder circuit as

$$\begin{aligned} Y_1 &= A_1 \oplus B_1 \oplus 0 \\ &= A_1 \oplus B_1 \end{aligned}$$

Using this relation we have to change the outputs as shown in Table 4.21.3.

S	$S_1$	$S_0$	$A'_1$	$B'_1$	$C_1$	$F_1 = A_1 \oplus B_1$	Operation	Required operation
1	0	0	$A_1$	0	0	$Y_1 = A_1$	Transfer A	OR
1	0	1	$A_1$	$B_1$	0	$Y_1 = A_1 \oplus B_1$	XOR	XOR
1	1	0	$A_1$	$\bar{B}_1$	0	$Y_1 = A_1 \oplus \bar{B}_1$	X-NOR	AND
1	1	1	$A_1$	1	0	$Y_1 = \bar{A}_1$	NOT	NOT

Table 4.21.3 Excitation table for ALU

Let us see, how we can change these outputs.

#### Case 1 : $S_1, S_0 = 00$ Required operation OR

Here,  $Y_1 = A_1$ . To make  $Y_1 = A_1 \vee B_1$ , we must change the input  $A'_1$  to each full adder circuit from  $A_1$  to  $A_1 + B_1$ .

#### Case 2 : $S_1, S_0 = 01$ Required operation XOR

Here,  $Y_1 = A_1 \oplus B_1$ . Hence change is not required.

#### Case 3 : $S_1, S_0 = 10$ Required operation AND

Here,  $Y_1 = A_1 \odot B_1$  and we want  $Y_1 = A_1 B_1$ . Let us investigate the possibility of ORing each input  $A_1$  with some boolean function  $K_1$ . Therefore, now

$$\begin{aligned} Y_1 &= (A_1 + K_1) \oplus \bar{B}_1 = (A_1 + K_1) B_1 + (\bar{A}_1 + K_1) \bar{B}_1 \\ &= A_1 B_1 + K_1 B_1 + (\bar{A}_1 \cdot \bar{K}_1) \bar{B}_1 = A_1 B_1 + K_1 B_1 + \bar{A}_1 \bar{K}_1 \bar{B}_1 \end{aligned}$$

To have  $Y_1 = A_1 B_1$ , terms 2 and 3 must be 0. i.e.  $K_1 B_1 + \bar{A}_1 \bar{K}_1 \bar{B}_1 = 0$ . This result can be obtained when  $K_1 = \bar{B}_1$ . This is illustrated as follows :

$$K_1 = \bar{B}_1$$

$$\begin{aligned} Y_1 &= A_1 B_1 + \bar{B}_1 B_1 + \bar{A}_1 \bar{B}_1 \bar{B}_1 \\ &= A_1 B_1 + \bar{B}_1 B_1 + \bar{A}_1 B_1 \bar{B}_1 \\ &= A_1 B_1 \end{aligned}$$

$$\because \bar{B}_1 \bar{B}_1 = 0$$

#### Case 4 : $S_1, S_0 = 11$ Required operation NOT

Here,  $Y_1 = \bar{A}_1$ . Hence change is not required. With above changes we can write excitation table for ALU as shown in Table 4.21.4.

S	$S_1$	$S_0$	$C_1'$	$A'_1$	$B'_1$
0	0	0	$C_1$	$A_1$	0
0	0	1	$C_1$	$A_1$	$B_1$
0	1	0	$C_1$	$A_1$	$\bar{B}_1$
0	1	1	$C_1$	$A_1$	1
1	0	0	0	$A_1 + B_1$	0
1	0	1	0	$A_1$	$B_1$
1	1	0	0	$A_1 + \bar{B}_1$	$\bar{B}_1$
1	1	1	0	$A_1$	1

Table 4.21.4 Excitation table for ALU

#### K-map simplification for $A'_1, B'_1$ and $C_1'$

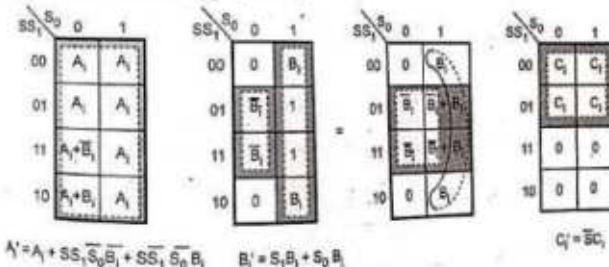


Fig. 4.21.5

Therefore, Boolean expressions for ALU inputs can be given as

$$A'_1 = A_1 + SS_1S_0B_1 + SS_1S_0B_1$$

$$B'_1 = S_1\bar{B}_1 + S_0B_1$$

$$C_1' = \bar{S}C_1$$

When  $S = 1$ , these equations reduce to

$$\begin{aligned} A'_1 &= A_1 \\ B'_1 &= S_1 \bar{B}_1 + S_0 B_1 \\ C'_1 &= 0 \end{aligned}$$

which are the functions of arithmetic circuits. Fig. 4.21.6 shows the final ALU circuit. It shows only first two stages, but can easily be extended to more stages.

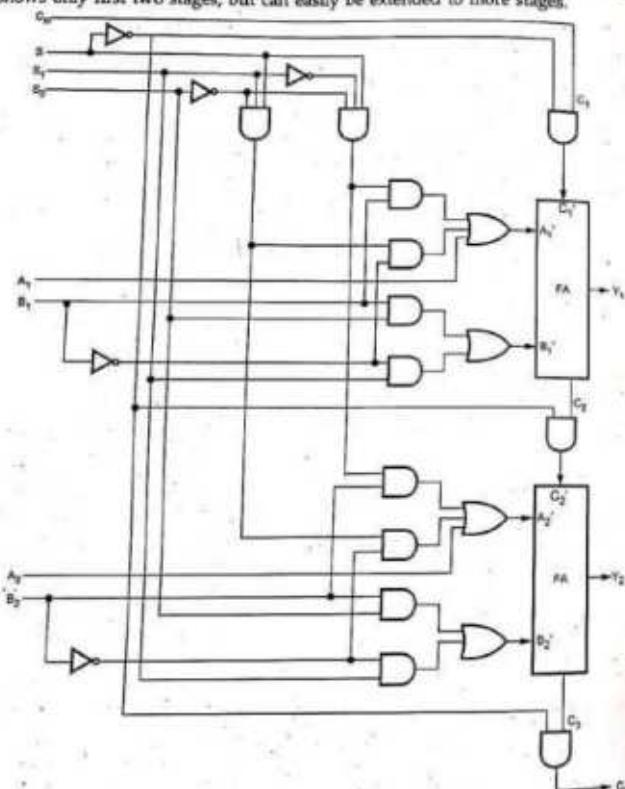


Fig. 4.21.6 Logic diagram of arithmetic logic unit (ALU)

## Review Questions

1. What is ALU? What is its function?
2. Explain the arithmetic logic unit.

## Ques. and Answers

- Q.1** Name the two basic types of Boolean expressions.  
**Ans.:** There are two basic types of Boolean expressions are :  
  - Sum of product Form (SOP) and
  - Product of Sum Form (POS)**Q.2** Name the two canonical forms for Boolean algebra.  
**Ans.:** Standard SOP and standard POS forms.
- Q.3** Express  $F = BC' + AC$  in a canonical SOP form.  
**Ans.:** 
$$\begin{aligned} F &= B\bar{C} + A\bar{C} = (\bar{A} + \bar{B})B\bar{C} + A\bar{C}(B + \bar{B}) \\ &= A\bar{B}\bar{C} + \bar{A}B\bar{C} + A\bar{B}C + A\bar{B}C \end{aligned}$$
- Q.4** What is variable mapping?  
**Ans.:** Representing the minterms of the sum of product expression in the Karnaugh-map of appropriate variables is known as variable mapping.
- Q.5** What code is used to label the row headings and column heading of K-map and why?  
**Ans.:**
- Gray code is used to label the rows and columns of K-map.
  - In case of Gray code, only one variable changes between two consecutive number. This is useful in grouping pair, quad, or octets in k-maps and thus eliminating variables in the final expression. Hence, Gray code is used to label the rows and columns of K-map.
- Q.6** What are don't care conditions and incompletely specified functions?  
**Ans.:** Sometimes, while designing a digital circuit, certain conditions of input are of no use. Consider an example of BCD input for a seven segment display. In this case, inputs from 0 through 9 are valid inputs. Thus, rest of the inputs from 10 through 15 do not signify anything at the output side. The outputs for which inputs are not specified are called don't care outputs. The Boolean function in which don't care outputs exists is called incomplete Boolean function or incompletely specified functions.

**Q.7** Distinguish between completely specified function and incompletely specified function.

**Ans. :**

Sr. No.	Completely specified function	Incompletely specified function
1.	All input conditions occur.	Certain input conditions never occur.
2.	Each output corresponding to input conditions is defined.	Outputs corresponding to non-occurring input conditions are undefined.
3.	Each output has certain logical value, either 0 or 1.	Undefined output doesn't have certain logical value, but it is indicated by 'Y' or 'd' in truth tables.
4.	In expression, $f(A, B, C) = \sum m(0, 1, 3, 5, 6)$ $= \pi M(1, 4, 7)$	In expression, $f(A, B, C) = \sum m(0, 1, 3) + d(5, 6)$ $= \pi M(2, 4, 7) + d(5, 6)$ Here outputs corresponding to terms 5 and 6 are assumed to be undefined ('x' or 'd') called as 'don't care'.

**Q.8** What are prime implicants?

**Ans. :** All the implicants of a function determined using a Karnaugh map are called prime implicants.

**Q.9** Write the names of universal gates.

**Ans. :** Universal gates are : NAND gate and NOR gate.

**Q.10** Why are NAND and NOR gates known as Universal gates?

**Ans. :** NAND and NOR are the gates that can be used alone to generate remaining gates such as NOT, AND and OR. Thus, with only any of the two gates, we can implement the logic circuit. Hence, they are called Universal gates.

**Q.11** How can a NAND gate be used as an inverter?

**Ans. :**



Fig. 4.1

**Q.12** Define a combinational logic circuit. Give an example.

**Ans. :** When logic gates are connected together to produce a specified output for certain specified combinations of input variables, with no storage involved, the resulting circuit is called 'combinational logic circuit'.

A combinational circuit consists of input variables, logic gates and output variables. For example, consider following Boolean expression.

$$Y = AB + BC + AC$$

The combinational logic circuit for this would require 3 AND gates and 1 OR gate as shown in Fig. 4.2.

**Q.13** Define half adder and full adder.

**Ans. :**

1) Half adder : The logic circuit which performs the arithmetic sum of two bits is called a half adder.

2) Full adder : The logic circuit which performs the arithmetic sum of 3 bits (bit 1 : input 1, bit 2 : input 2, bit 3 : carry from the previous addition) is called a full adder.

**Q.14** Define half subtractor and full subtractor.

**Ans. :**

1) Half subtractor : It is a combinational circuit that subtracts two bits and produces their difference and borrow.

2) Full subtractor : It is a combinational circuit that performs a subtraction between 2 bits. It also takes into account borrow of the lower significant stage.

**Q.15** What is a data selector ? or What is multiplexer ? or Why is MUX called as data detector ?

**Ans. :**

Multiplexer is a digital switch. Particularly, it has  $2^n$  input lines and n selection lines whose bit combinations determine which input line is selected and routed onto available only single output line.

Hence, multiplexer is a selector of one out of several data sources available at its input lines, to connect it to output line. Simply it is a 'many into one' device and also called 'data selector'.

**Q.16** What is decoder ?

**Ans. :** A decoder is a multiple - input, multiple-output logic circuit which converts coded inputs into coded outputs, where the input and output codes are different. In a binary decoder n-inputs produce  $2^n$  outputs. Usually, a decoder is provided with enable inputs to activate decoded output.



Fig. 4.2

**Q.17** What is binary decoder?

**Ans.:** A decoder is a combinational circuit that converts binary information from input lines to a maximum of  $2^n$  outputs lines.

**Q.18** What do you mean by encoder?

**Ans.:** An encoder is a digital circuit that performs the inverse operation of a decoder. Encoder has  $2^m$  (or fewer) input lines and n output lines. Encoder has enable inputs to activate encoded outputs.

**Q.19** Write a short note on priority encoder. OR What is a priority encoder?

**Ans.:** A priority encoder is an encoder circuit in which if two or more inputs are equal to 1 at the same time, the input having the highest priority will take precedence.

Inputs				Outputs		
D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>	Y <sub>1</sub>	Y <sub>0</sub>	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

Table 4.1 Truth table of 4-bit priority encoder

- As per the table, when D<sub>3</sub> is high, regardless of other inputs, the output is 11.
- D<sub>2</sub> has next priority, thus, with D<sub>2</sub> = 1 and D<sub>3</sub> = 0, output will be 10.
- Output for D<sub>1</sub> is generated only if higher priority inputs are 0, and so on.
- If all inputs are 0, two outputs of the circuit are not used.

**Q.20** What do you mean by comparator? OR Write a short note on 1-bit comparator.

**Ans.:** It is a special combinational circuit designed primarily to compare the relative magnitudes of two binary numbers. An n-bit comparator receives two n-bit numbers A and B, outputs are : A > B, A = B and A < B. As per the magnitudes of the two numbers, one of the outputs will be high.

## 5

## Sequential Circuits and Systems

### Contents

5.1	Introduction	Winter-15, ... Marks 3
5.2	One-Bit Memory and the Circuit Properties of Bistable Latch	Dec.-13, May-12, 12, 14, Summer-15, 17, 18, ... Marks 3
5.3	Latches	Winter-15, Summer-18, ... Marks 3
5.4	Clocked Flip-Flops	Summer-15, 17, 18, ... Marks 3
5.5	Excitation Tables	Winter-15, 17, 18, ... Marks 7
5.6	Conversion from One Type to another Type of Flip-Flop	Summer-15, 18, ... Marks 7
5.7	Applications of Flip-Flops	Summer-15, ... Marks 7
5.8	Registers and Shift Registers	Dec.-13, May-13, 14, Summer-15, 18, Winter-18 - Marks 7
5.9	Counters	May-12, 14, Dec-12, 13, Summer-15, 16, 18, ... Marks 7
5.10	Shift Register Counters	Winter-14, 15, ... Marks 7
5.11	Sequence Generator	May-12, ... Marks 7
5.12	Special Counter ICs	Summer-18, Winter-18, ... Marks 7
GATE Questions and Answers		

**5.1 Introduction**

- Fig. 5.1.1 shows the block diagram of sequential circuit/Finite State Machine (FSM).
- Memory elements are connected to the combinational circuit as a feedback path.

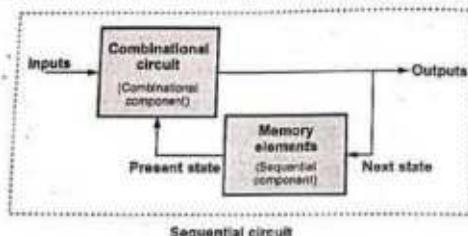


Fig. 5.1.1 Block diagram of sequential circuit / FSM

- The information stored in the memory elements at any given time defines the present state of the sequential circuit.
- The present state and the external inputs determine the outputs and the next state of the sequential circuit.
- Thus we can specify the sequential circuit by a time sequence of external inputs, internal states (present states and next states), and outputs.
- The counters and registers are the common examples of sequential circuits.

The memory element used in sequential circuits is a flip-flop which is capable of storing 1-bit binary information.

**5.1.1 Comparison between Combinational and Sequential Logic Circuits**

Sr. No.	Combinational circuits	Sequential circuits
1.	In combinational circuits, the output variables are at all times dependent on the combination of input variables.	In sequential circuits, the output variables depend not only on the present input variables but they also depend upon the past history of these input variables.
2.	Memory unit is not required in combinational circuits.	Memory unit is required to store the past history of input variables in the sequential circuit.

1.	Combinational circuits are faster in speed because the delay between input and output is due to propagation delay of gates.	Sequential circuits are slower than the combinational circuits.
4.	Combinational circuits are easy to design.	Sequential circuits are comparatively harder to design.
5.	Parallel adder is a combinational circuit.	Serial adder is a sequential circuit.

Table 5.1.1 Comparison between combinational and sequential circuits

**5.1.2 Clock**

- A clock signal is a particular type of signal that oscillates between a high and a low state and is utilized to co-ordinate actions of circuits.
- It is produced by a clock generator.
- The most common clock signal is in the form of a square wave with a 50 % duty cycle, usually with a fixed, constant frequency as shown in Fig. 5.1.2.
- Circuits using the clock signal for synchronization may become active at either the rising edge, falling edge or in the case of double data rate, both in the rising and in the falling edges of the clock cycle.
- The time required to complete one cycle is called 'clock period' or 'clock cycle'.
- Ideally, the clock signal should have sharp transitions from one level to other as shown in Fig. 5.1.2.



Fig. 5.1.2 Clock signal

**Review Questions**

- Define sequential logic circuit.
- What is flip-flop?
- Give the comparison between combinational and sequential logic circuits.
- What is clock? State its use.
- Draw a general model for a sequential or state machine. Also list out various types of FSMs.

### 5.2 One-Bit Memory and the Circuit Properties of Bistable Latch

- The Fig. 5.2.1 shows the basic bistable element used in latches and flip-flops.

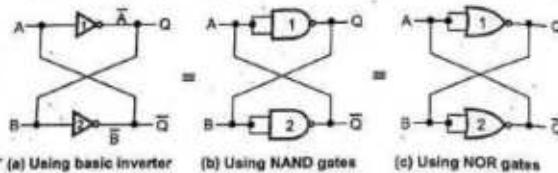


Fig. 5.2.1 Basic bistable element (1-bit memory cell)

- The basic bistable element has two outputs  $Q$  and  $\bar{Q}$ .
- It has two cross-coupled inverters, i.e., the output of the first inverter is connected as an input to the second inverter and the output of second inverter is connected as an input to the first inverter.
- The basic bistable element circuit has two stable states logic 0 and logic 1, hence the name 'bistable'.
- When  $A = 0$ , the output of inverter 1 is 1 ( $\bar{A}$ ), i.e.,  $Q = 1$ .
- Since the output of inverter 1 is the input to the inverter 2,  $\bar{A} = B = 1$ . Consequently, the output of inverter 2, i.e.,  $\bar{B}$  is 0.
- Since the output of the inverter 2 is connected to the input of the inverter 1,  $\bar{Q} = \bar{B} = A = 0$ .
- We have assumed same value for  $A$ . Thus, the circuit is stable with  $\bar{Q} = A = \bar{B} = 0$  and  $Q = \bar{A} = B = 1$ .
- Using similar explanation it is easy to show that if it is assumed that  $A = 1$ , the basic bistable element is stable with  $\bar{Q} = A = \bar{B} = 1$  and  $Q = \bar{A} = B = 0$ . This is second stable condition of the basic bistable element.
- The two stable states of basic bistable elements are used to store two binary elements, 0 and 1.
- In positive logic system, state  $Q = 1$  is used to store logic 1, and state  $Q = 0$  is used to store logic 0.
- Two outputs are complementary. That is when  $Q = 0$ ,  $\bar{Q} = 1$ ; and when  $Q = 1$ ,  $\bar{Q} = 0$ .

### Important Points

- The outputs  $Q$  and  $\bar{Q}$  are always complementary.
- The circuit has two stable states. The state corresponds to  $Q = 1$  is referred to as 1 state or set state and state corresponds to  $Q = 0$  is referred to as 0 state or reset state.
- If the circuit is in the set (1) state, it will remain in the set state and if the circuit is in the reset (0) state, it will remain in the reset state. This property of the circuit shows that it can store 1-bit of digital information. Therefore, the circuit is called a 1-bit memory cell.
- The 1-bit information stored in the circuit is locked or latched in the circuit. Therefore, this circuit is also referred to as a latch.

### Review Question

- Explain the operation of one-bit memory cell.

### 5.3 Latches

GTU : Winter-15, Summer-16

#### 5.3.1 SR Latch

- Fig. 5.3.1 shows SR latch which is 1-bit memory cell.
- Two inverters 3 and 4 are connected to enter the digital information.
- Input for gate 3 is  $S$  and input for gate 4 is  $R$ . This latch is also called RS latch.
- For understanding the circuit operation, we must first determine the output of NAND gate whose one of the input is logic 0 and accordingly we have to determine the output of other NAND gate in the cross coupled circuit.
- Because the output of NAND gate is 1 if any one input is 0.
- The circuit operation is as follows. In Fig. 5.3.2, the output of shaded NAND gate is determined first, and the 0 input that decides the output of shaded NAND as 1 is shown in bold.

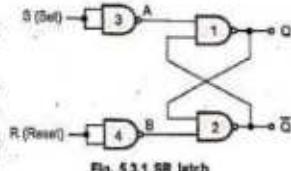


Fig. 5.3.1 SR latch

#### Case 1 : $S = R = 0$

In this case,  $S = R = 1$ . If  $Q$  is 1,  $Q$  and  $\bar{R}$  inputs for NAND gate 2 are both 1 and hence output  $\bar{Q} = 0$ . Since  $\bar{Q} = 0$  and  $\bar{S} = 1$ , the output of NAND gate 1 is 1, i.e.  $Q = 1$ .

If  $Q = 0$ ,  $Q$  and  $\bar{R}$  inputs for NAND gate 2 are 0 and 1, and hence output  $\bar{Q} = 1$ . Since  $\bar{Q} = 1$  and  $S = 1$ , the output of NAND gate 1 is 0, i.e.,  $Q = 0$ .

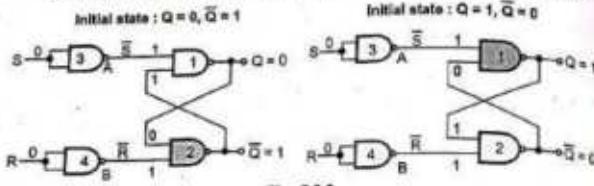


Fig. 5.3.2

This shows that when  $S = R = 0$ , the outputs do not change.

#### Case 2 : $S = 1$ and $R = 0$

In this case,  $\bar{S} = 0$  and  $\bar{R} = 1$ . Since  $\bar{S} = 0$ , the output of NAND gate 1,  $Q = 1$  (Recall that, for NAND any one or more input is 0, the output is 1). For NAND gate 2, both inputs  $Q$  and  $\bar{R}$  are 1, thus output  $\bar{Q} = 0$ .

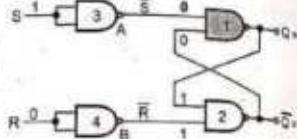


Fig. 5.3.3

The inputs  $S = 1$  and  $R = 0$ , makes  $Q = 1$ , i.e., set state.

#### Case 3 : $S = 0$ and $R = 1$

In this case,  $\bar{S} = 1$  and  $\bar{R} = 0$ . Since  $\bar{R} = 0$ , the output of NAND gate 2,  $\bar{Q} = 1$ . For NAND gate 1, both inputs  $Q$  and  $\bar{S}$  are 1, thus output  $Q = 0$ .

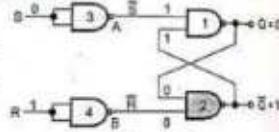


Fig. 5.3.4

The inputs  $S = 0$  and  $R = 1$ , makes  $Q = 0$ , i.e., reset state.

#### Case 4 : $S = 1$ and $R = 1$

When  $S = R = 1$ , both the outputs  $Q$  and  $\bar{Q}$  try to become 1 which is not allowed and therefore, this input condition is prohibited.

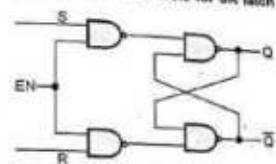
### 5.3.2 Gated SR Latch

- In the SR latch output changes occur immediately after the input changes i.e. the latch is sensitive to its  $S$  and  $R$  inputs at all times.
- It can easily be modified to create a latch that is sensitive to these inputs only when an enable input is active. Such a latch with enable input is known as gated SR latch. It is as shown in the Fig. 5.3.5.

- The Table 5.3.1 shows the truth table for gated latch.

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

Table 5.3.1 Truth table for SR latch with enable input



(a) SR latch with enable input using NAND gates

(b) Logic symbol

- As shown by truth table, the circuit behaves like a SR latch when  $EN = 1$ , and retains its previous state when  $EN = 0$ .

### 5.3.3 Gated D Latch

- For SR latch, when both inputs are same the output either does not change or it is invalid (Inputs  $\rightarrow 00$ , no change and inputs  $\rightarrow 11$ , invalid).
- In many practical applications, these input conditions are not required.
- These input conditions can be avoided by making them complement of each other. This modified SR latch is known as D latch.
- Fig. 5.3.6 shows the D latch.
- The NAND gates 1, 2, 3 and 4 form the basic SR latch with enable input.
- The fifth NAND gate is used to provide the complemented inputs.
- As shown in the Fig. 5.3.6, D input goes directly to the S input, and its complement is applied to the R input, through gate 5. Therefore, only two input conditions exists, either  $S = 0$  and  $R = 1$  or  $S = 1$  and  $R = 0$ .

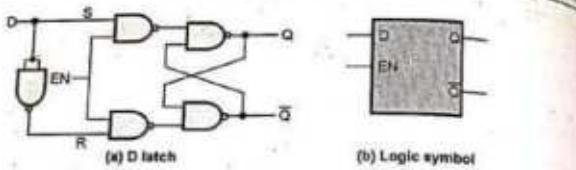


Fig. 5.3.6

- The truth table for D latch is as shown in the Table 5.3.2.

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

Table 5.3.2 Truth table for D latch

- As shown in the truth table, the Q output follows the D input. For this reason D latch is sometimes called transparent latch.
- Looking at the truth table for D latch with enable input and simplifying  $Q_{n+1}$  function by k-map we get the characteristic equation for D latch with enable input as

$Q_{n+1} = EN \cdot D + \overline{EN} \cdot Q_n$ . This is illustrated in Fig. 5.3.7.

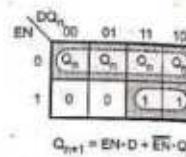


Fig. 5.3.7 Characteristic equation

#### Review Questions

- What is SR latch? Explain its operation.
- What is gated SR latch?
- Explain the working of gated D latch with truth table and characteristic equation.
- Draw gated SR latch using NAND gates only.
- Draw and explain in brief a high assertion input SR latch.
- Which latch is also known as transparent latch?
- Draw high assertion and low assertion input SR latches.
- Draw gated SR latch using NAND gates only.

GTU : Summer-18, Marks 3

GTU : Winter-15, Marks 3

GTU : Winter-15, Mark 1

GTU : Summer-18, Marks 3

GTU : Summer-18, Marks 3

#### 5.4 Clocked Flip-Flops

GTU : Dec-13, May-12, 13, 14, Session 15, 17, 18, Winter-15, 17, 18

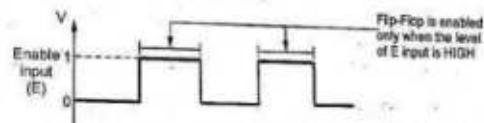
##### 5.4.1 Latches Vs Flip-Flops

- Latches and flip-flops are the basic building blocks of the most sequential circuits.
- The main difference between latches and flip-flops is in the method used for changing their state.
- A simple latch forms the basis for the flip-flop.
- Latches are controlled by enable signal, and they are level triggered, either positive level triggered or negative level triggered.
- The output state is free to change according to the S and R input values, when active level is maintained at the enable input.
- Flip-flops are pulse or clock edge triggered instead of level triggered.

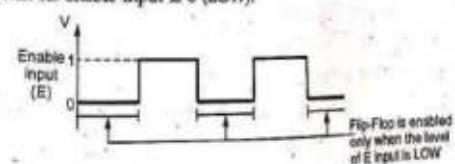
##### 5.4.2 Level and Edge Triggering

###### Level Triggering

- In the level triggering, the output state is allowed to change according to input(s) when active level (either positive or negative) is maintained at the enable input.
- There are two types of level triggered latches :
- Positive level triggered : The output of flip-flop responds to the input changes only when its enable input is 1 (HIGH).



- Negative level triggered : The output of flip-flop responds to the input changes only when its enable input is 0 (LOW).



**Edge Triggering**

- In the edge triggering, the output responds to the changes in the input only at the positive or negative edge of the clock pulse at the clock input.
- There are two types of edge triggering.
- Positive edge triggering :** Here, the output responds to the changes in the input only at the positive edge of the clock pulse at the clock input.

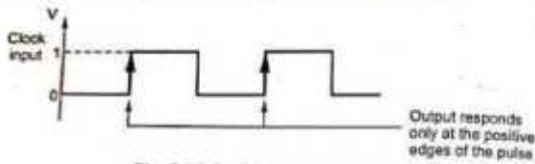


Fig. 5.4.3 Positive edge triggering

- Negative edge triggering :** Here, the output responds to the changes in the input only at the negative edge of the clock pulse at the clock input.

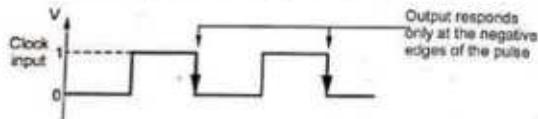


Fig. 5.4.4 Negative edge triggering

**5.4.3 SR Flip-Flop****Positive Edge Triggered SR Flip-Flop**

- The Fig. 5.4.5 shows the positive edge triggered clocked SR flip-flop.

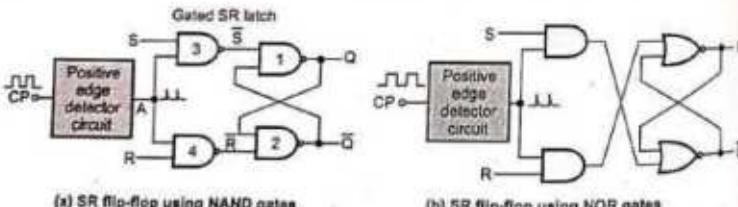


Fig. 5.4.5 Clocked SR flip-flop

- The circuit is similar to SR latch except enable signal is replaced by the Clock Pulse (CP) followed by the positive edge detector circuit.

- The edge detector circuit is a differentiator.
- The Fig. 5.4.7 shows input and output waveforms for positive edge triggered clocked SR flip-flop.
- The circuit output responds to the S and R inputs only at the positive edges of the clock pulse. At any other instants of time, the SR flip-flop will not respond to the changes in input.

The Fig. 5.4.6 shows the logic symbol and truth table of clocked SR flip-flop.

(a) Logic symbol	(b) Truth table for positive edge clocked SR flip-flop	(c) Characteristic equation																																																						
	<table border="1"> <thead> <tr> <th>CP</th> <th>S</th> <th>R</th> <th><math>Q_n</math></th> <th><math>Q_{n+1}</math></th> <th>Notes</th> </tr> </thead> <tbody> <tr> <td>T</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>No Change(C)</td> </tr> <tr> <td>T</td> <td>0</td> <td>0</td> <td>1</td> <td>1</td> <td>Reset</td> </tr> <tr> <td>T</td> <td>0</td> <td>1</td> <td>0</td> <td>0</td> <td>Set</td> </tr> <tr> <td>T</td> <td>1</td> <td>0</td> <td>0</td> <td>1</td> <td>Set</td> </tr> <tr> <td>T</td> <td>1</td> <td>0</td> <td>1</td> <td>X</td> <td>Indeterminate</td> </tr> <tr> <td>T</td> <td>1</td> <td>1</td> <td>0</td> <td>X</td> <td>Indeterminate</td> </tr> <tr> <td>0</td> <td>X</td> <td>X</td> <td>0</td> <td>0</td> <td>No Change(C)</td> </tr> <tr> <td>0</td> <td>X</td> <td>X</td> <td>1</td> <td>1</td> <td>No Change(C)</td> </tr> </tbody> </table>	CP	S	R	$Q_n$	$Q_{n+1}$	Notes	T	0	0	0	0	No Change(C)	T	0	0	1	1	Reset	T	0	1	0	0	Set	T	1	0	0	1	Set	T	1	0	1	X	Indeterminate	T	1	1	0	X	Indeterminate	0	X	X	0	0	No Change(C)	0	X	X	1	1	No Change(C)	$Q_{n+1} = S + \bar{R} Q_n$
CP	S	R	$Q_n$	$Q_{n+1}$	Notes																																																			
T	0	0	0	0	No Change(C)																																																			
T	0	0	1	1	Reset																																																			
T	0	1	0	0	Set																																																			
T	1	0	0	1	Set																																																			
T	1	0	1	X	Indeterminate																																																			
T	1	1	0	X	Indeterminate																																																			
0	X	X	0	0	No Change(C)																																																			
0	X	X	1	1	No Change(C)																																																			

Fig. 5.4.6

**Case 1 :** If  $S = R = 0$  and the clock pulse is applied, the output do not change, i.e.  $Q_{n+1} = Q_n$ . This is indicated in the first row of the truth table.

**Case 2 :** If  $S = 0, R = 1$  and the clock pulse is applied,  $Q_{n+1} = 0$ . This is indicated in the second row of the truth table.

**Case 3 :** If  $S = 1, R = 0$  and the clock pulse is applied,  $Q_{n+1} = 1$ . This is indicated in the third row of the truth table.

**Case 4 :** If  $S = R = 1$  and the clock pulse is applied, the state of the flip-flop is undefined and therefore is indicated as indeterminate in the fourth row of the truth table.

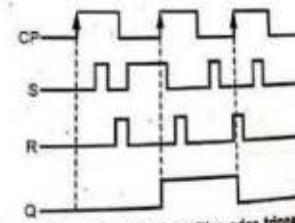
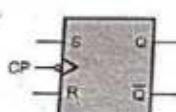


Fig. 5.4.7 Input and output waveforms for positive edge triggered clocked SR flip-flop

**Negative Edge Triggered SR Flip-Flop**

- In the negative edge triggered SR flip-flop, the negative edge detector circuit is used and the circuit output responds at the negative edges of the clock pulse.
- The Fig. 5.4.8 and Fig. 5.4.9 shows the logic symbol, truth table, and input and output waveforms for negative edge triggered SR flip-flop.
- The bubble at the clock input indicates that the flip-flop is negative edge triggered.



(a) Logic symbol

CP	S	R	Q <sub>in</sub>	Q <sub>out</sub>	State
1	0	0	0	0	No change(NC)
1	0	0	1	1	
1	0	1	0	0	
1	0	1	1	0	Reset
1	1	0	0	1	Set
1	1	0	1	1	
1	1	1	0	X	Indeterminate
1	1	1	1	X	
0	X	X	0	0	
0	X	X	1	1	No change(NC)

(b) Truth Table for negative edge clocked SR flip-flop

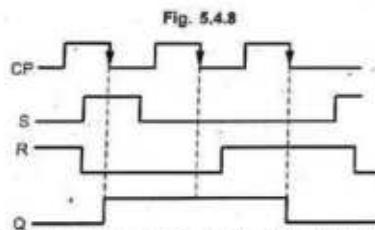


Fig. 5.4.9 Input and output waveforms for negative edge triggered clocked SR flip-flop

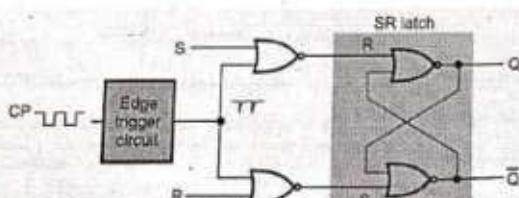
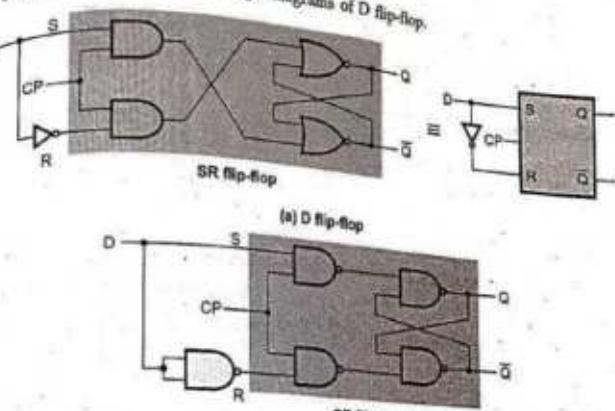
**Example 5.4.1** Realize SR flip-flop using NOR gates.**Solution :**

Fig. 5.4.10 Negative edge-triggered SR flip-flop using only NOR gates

**5.4.4 D Flip-Flop**

- The Fig. 5.4.11 shows the logic diagrams of D flip-flop.

(b) D flip-flop using NAND gates  
Fig. 5.4.11

- The basic building block of D flip-flop is a SR flip-flop.
- The SR flip-flop has two data inputs S and R.
- The S input is made high to store 1 in the flip-flop and R input is made high to store 0 in the flip-flop.
- When both inputs are same the output either does not change or it is invalid (Inputs → 00, no change and Inputs → 11, invalid).
- In many practical applications, these input conditions are not required.
- These input conditions can be avoided by making them complement of each other. This modified SR flip-flop is known as D flip-flop.
- The D input goes directly to the S input, and its complement is applied to the R input. Due to these connections, only two input conditions exists, either S = 0 and R = 1 or S = 1 and R = 0.

**Truth Table**

- The truth table for D flip-flop consider only these two conditions and it is as shown in the Fig. 5.4.12 (b).

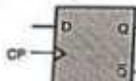


Fig. 5.4.12 (a) Logic symbol

CP	D	$Q_{out}$
↑	0	0
↑	1	1
0	X	$Q_n$

Fig. 5.4.12 (b) Truth table of D flip-flop

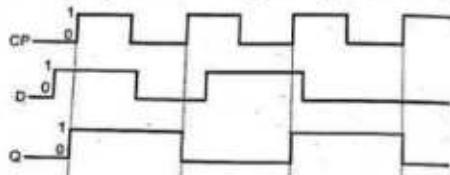


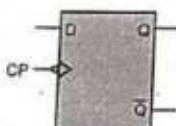
Fig. 5.4.12 (c) Input and output waveforms of clocked D flip-flop

- $Q_{n+1}$  function follows D input at the positive going edges of the clock pulses. Hence the characteristic equation for D flip-flop is  $Q_{n+1} = D$ .
- The output  $Q_{n+1}$  is delayed by one clock period. Thus, D flip-flop is also known as delay flip-flop.
- If we connect the  $\bar{Q}$  output of D flip-flop to its D input as shown in Fig. 5.4.13, the output of D flip-flop will change either from 0 to 1 or from 1 to 0 at every positive edge of the D flip-flop.

Such change in the output is known as toggling of the flip-flop output.

#### Negative Edge Triggered D Flip-Flop

- In case of negative edge triggering, the output is sensitive at the negative edge of the clock input.
- The Fig. 5.4.14 shows the logic symbol and truth table for negative edge triggered D flip-flop.



(a) Logic symbol

CP	D	$Q_{n+1}$
↓	0	0
↓	1	1
0	X	$Q_n$

(b) Truth table of D flip-flop

Fig. 5.4.14

- Fig. 5.4.15 shows input and output waveforms for negative edge triggered D flip-flop.
- The bubble at the clock input indicates that the flip-flop is negative edge triggered.

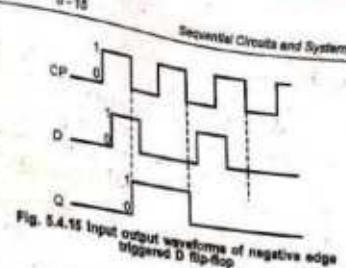


Fig. 5.4.15 Input output waveforms of negative edge triggered D flip-flop

#### JK Flip-Flop

- The uncertainty in the state of an SR flip-flop when  $S = R = 1$  can be eliminated by converting it into a JK flip-flop.

- The data inputs are J and K which are ANDed with Q and  $\bar{Q}$ , respectively, to obtain S and R inputs, as shown in Fig. 5.4.16. Thus,  $S = J \cdot \bar{Q}$  and  $R = K \cdot Q$ .

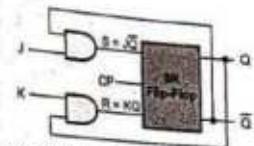


Fig. 5.4.16 JK flip-flop using SR flip-flop

#### Operation of JK Flip-Flop

##### Case 1 : $J = K = 0$

When  $J = K = 0$ ,  $S = R = 0$  and according to truth table of SR flip-flop there is no change in the output.

When inputs  $J = K = 0$ , output does not change.

##### Case 2 : $J = 1$ and $K = 0$

$Q = 0, \bar{Q} = 1$  : When  $J = 1, K = 0$  and  $Q = 0, S = 1$  and  $R = 0$ . According to truth table of SR flip-flop it is set state and the output Q will be 1.

$Q = 1, \bar{Q} = 0$  : When  $J = 1, K = 0$  and  $Q = 1, S = 0$  and  $R = 0$ . Since SR = 00, there is no change in the output and therefore,  $Q = 1$  and  $\bar{Q} = 0$ .

The inputs  $J = 1$  and  $K = 0$ , makes  $Q = 1$ , i.e., set state.

##### Case 3 : $J = 0$ and $K = 1$

$Q = 0, \bar{Q} = 1$  : When  $J = 0, K = 1$  and  $Q = 0, S = 0$  and  $R = 1$ . Since SR = 01, there is no change in the output and therefore,  $Q = 0$  and  $\bar{Q} = 1$ .

$Q = 1, \bar{Q} = 0$  : When  $J = 0, K = 1$  and  $Q = 1, S = 0$  and  $R = 1$ . According to truth table of SR flip-flop it is a reset state and the output Q will be 0.

The inputs  $J = 0$  and  $K = 1$ , makes  $Q = 0$ , i.e., reset state.

**Case 4 :  $J = K = 1$** 

$Q = 0, \bar{Q} = 1$  : When  $J = K = 1$  and  $Q = 0, S = 1$  and  $R = 0$ . According to truth table of SR flip-flop it is a set state and the output Q will be 1.

$Q = 1, \bar{Q} = 0$  : When  $J = K = 1$  and  $Q = 1, S = 0$  and  $R = 1$ . According to truth table of SR flip-flop it is a reset state and the output Q will be 0.

The input  $J = K = 1$ , toggles the flip-flop output.

- Fig. 5.4.17 shows the logic symbol, truth table and timing diagram of positive edge triggered JK flip-flop.

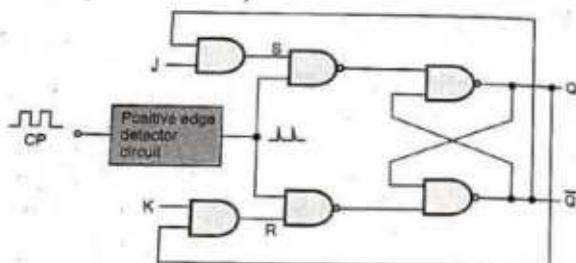


Fig. 5.4.17 (a) Clocked JK flip-flop

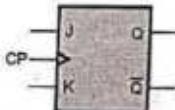


Fig. 5.4.17 (b) Logic symbol

$Q_n$	$J$	$K$	$Q_{n+1}$	$J$	$K$	$Q_{n+1}$
0	0	0	0	0	0	0
0	0	1	0	0	1	1
0	1	0	1	0	1	0
1	1	1	1	1	0	0
1	0	0	1	1	0	1
1	0	1	0	1	1	1
1	1	0	1	1	1	0
1	1	1	0			

Fig. 5.4.17 (c) Truth table

$Q_n$	00	01	11	10
0	0	0	1	1
1	1	0	0	1

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

Fig. 5.4.17 (d) Characteristics equation

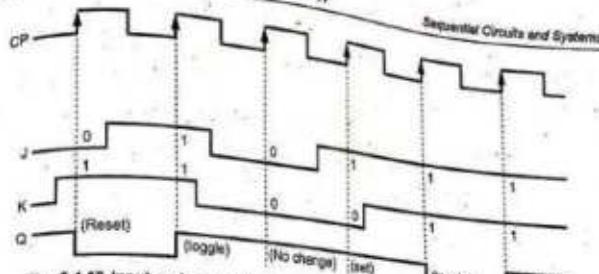


Fig. 5.4.18 Input and output waveforms for positive edge triggered JK flip-flop

**Example 5.4.2** Construct a clocked JK flip-flop which is triggered at the positive edge of the clock pulse from a clocked SR flip-flop consisting of NOR gates.

Solution :

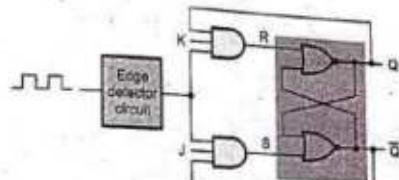


Fig. 5.4.19

**5.4.1 JK Flip-Flop using NAND Gates**

- The Fig. 5.4.20 shows the modified circuit of JK flip-flop which has only NAND gates.

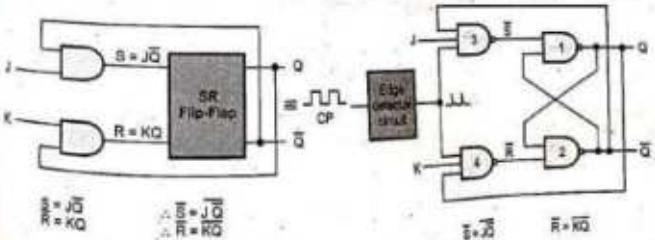


Fig. 5.4.20 JK flip-flop using NAND gates

**5.4.5.2 Race-around Condition**

- In JK flip-flop, when  $J = K = 1$ , the output toggles (output changes either from 0 to 1 or from 1 to 0).
- Consider that initially  $Q = 0$  and  $J = K = 1$ . After a time interval  $\Delta t$  equal to the propagation delay through two NAND gates in series, the output will change to  $Q = 1$  and after another time interval of  $\Delta t$  the output will change back to  $Q = 0$ . This toggling will continue until the flip-flop is enabled and  $J = K = 1$ . At the end of clock pulse the flip-flop is disabled and the value of  $Q$  is uncertain. This situation is referred to as the race-around condition. This is illustrated in Fig. 5.4.21.
- This condition exists when  $t_p \geq \Delta t$ . Thus by keeping  $t_p < \Delta t$  we can avoid race-around condition.
- We can keep  $t_p < \Delta t$  by keeping the duration of edge less than  $\Delta t$ .
- A more practical method for overcoming this difficulty is the use of the Master-Slave (MS) configuration.

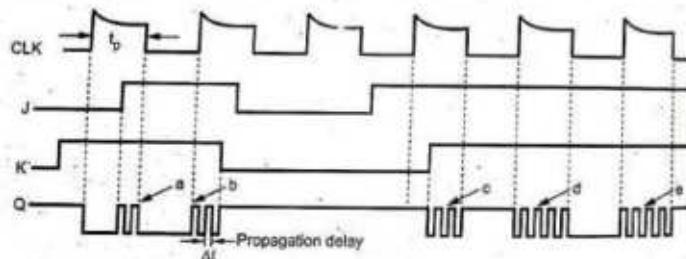


Fig. 5.4.21 Input and output waveforms for clocked JK flip-flop

**Example 5.4.3** Realize a JK flip-flop using only NOR gates.

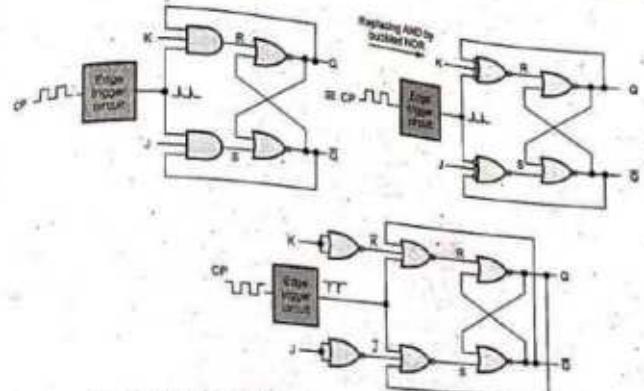


Fig. 5.4.22 Negative edge triggered JK flip-flop using only NOR gates

**5.4.6 Master-Slave SR Flip-Flop**

- A master-slave flip-flop is constructed from two flip-flops.
- One circuit serves as a master and the other as a slave, and the overall circuit is referred to as a master-slave flip-flop.
- Fig. 5.4.23 shows SR master-slave flip-flop.

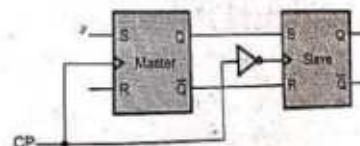


Fig. 5.4.23 Master-Slave SR flip-flop

- It consists of a master flip-flop, a slave flip-flop, and an inverter.
- Both the flip-flops are positive level triggered, but inverter connected at the clock input of the slave flip-flop forces it to trigger at the negative level.

- The output state of the master flip-flop is determined by the S and R inputs at the positive clock pulse.
- The output state of the master is then transferred as an input to the slave flip-flop. The slave flip-flop uses this input at the negative clock pulse to determine its output state.
- Fig. 5.4.24 illustrates the operation of the master-slave flip-flop.

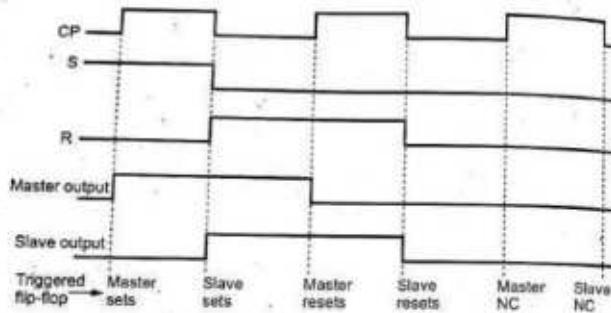


Fig. 5.4.24 Input and output waveforms for master-slave flip-flop

#### 5.4.7 Master-Slave JK Flip-Flop

- Fig. 5.4.25 shows the master-slave JK flip-flop. Positive clock pulses are applied to first flip-flop and inverted (negative) clock pulses are applied to second flip-flop. (See Fig. 5.4.25 on next page)
- When CK = 1, the first flip-flop is enabled and the outputs  $Q_M$  and  $\bar{Q}_M$  respond to the inputs of J and K according to the Table 1. At this time, the second flip-flop is inhibited because its clock is low,  $\bar{CK} = 0$ .
- When CK goes Low ( $\bar{CK} = 1$ ), the first flip-flop is inhibited and second flip-flop is enabled. At this time, the output of second flip-flop (Q and  $\bar{Q}$ ) follow the output  $Q_M$  and  $\bar{Q}_M$ , respectively.
- Since the second flip-flop follows the first one, it is referred to as the slave and the first one as the master.
- In master-slave JK flip-flop state change occurs when flip-flop goes through both positive transition (first half) of clock and negative transition of the clock (second half). Thus, race-around condition does not exist in the master-slave JK flip-flop.

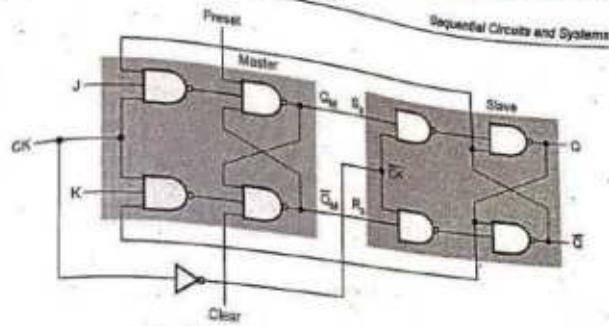


Fig. 5.4.25 Master - slave JK flip - flop

$Q_n$	J	K	$Q_{n+1}$
0	0	0	0
0	0	1	0
1	0	0	1
1	1	0	1
0	1	1	0
1	0	1	0
1	1	0	1
1	1	1	0

J	K	$Q_{n+1}$
0	0	Q <sub>n</sub>
0	1	0
1	0	1
1	1	0

Table 5.4.1 Truth table

#### 5.4.8 T Flip-Flop

- T flip-flop is also known as 'Toggle flip-flop'.
- The T flip-flop is a modification of the JK flip-flop.
- As shown in the Fig. 5.4.27, the T flip-flop is obtained from a JK flip-flop by connecting both inputs, J and K together.

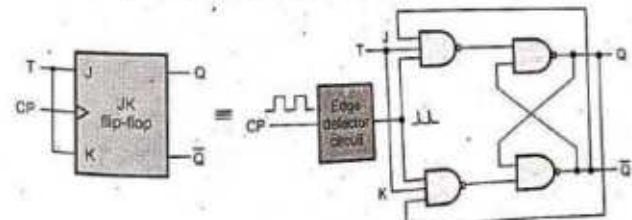


Fig. 5.4.26 T flip-flop using NAND gates

- When  $T = 0, J = K = 0$  and hence there is no change in the output. When  $T = 1, J = K = 1$  and hence output toggles.
- The Fig. 5.4.27 shows logic symbol, truth table and the characteristic equation for T flip-flop.

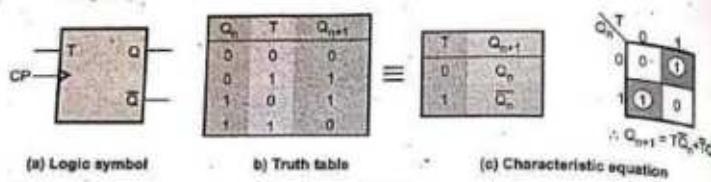


Fig. 5.4.27

**Example 5.4.4** Refer Fig. 5.4.28 and determine the Q output waveform if the flip-flop starts out RESET.

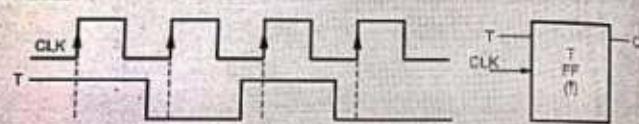


Fig. 5.4.28

Solution :

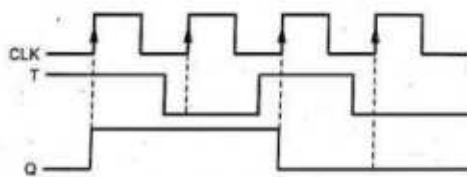


Fig. 5.4.29

#### 5.4.9 Preset and Clear

- For the flip-flops discussed so far, the SR, D, JK, and T, the inputs are called synchronous inputs because data on these inputs are transferred to the flip-flop's output only on the triggering edge of the clock pulse; that is, the data are transferred synchronously with the clock.
- When power is turn ON, the state of the flip-flop is uncertain. It may come to set ( $Q = 1$ ) or reset ( $Q = 0$ ) state.

- In many applications, it is necessary to initially set or reset the flip-flop. Such initial state of flip-flop can be accomplished by using the direct or asynchronous inputs of the flip-flop. These inputs are : Preset ( $\bar{P}$ ) and Clear ( $\bar{C}$ ). They can be applied at any time between clock pulses and are not in synchronism with the clock.
- Fig. 5.4.30 shows the SR and D flip-flops with preset and clear inputs. These are active-low inputs.

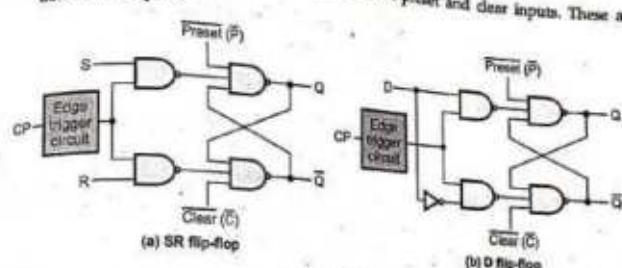


Fig. 5.4.30

- When  $\bar{P} = \bar{C} = 1$ , the circuit operates in accordance with the truth table of SR flip-flop.
- If  $\bar{P} = 1$  and  $\bar{C} = 0$ , the flip-flop is reset and  $\bar{P} = 0$  and  $\bar{C} = 1$ , the flip-flop is set.

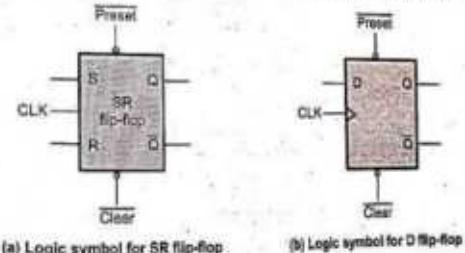
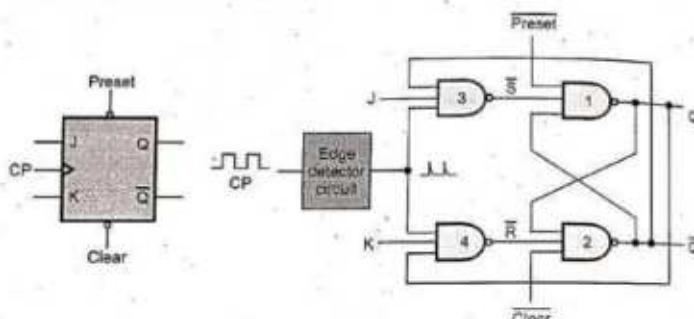


Fig. 5.4.31 Logic symbol

**Note** Condition  $P = C = 0$  must not be used, since this leads to an uncertain state.

- The JK flip-flop with preset and clear is shown in Fig. 5.4.32.
- If preset and clear inputs are 1, the circuit operates in accordance with the truth table of JK flip-flop given in the Fig. 5.4.32 (c).

- If preset = 0 and clear = 1, the output of NAND gate 1 will certainly be 1. Consequently, all the three inputs to NAND gate 2 will be 1 which will make  $\bar{Q} = 0$ .
- Making preset = 0 sets the flip-flop.
- Preset signal is active when it is low, hence it is active low signal.
- Low (0) on the clear input resets the flip-flop making  $\bar{Q} = 1$ .
- A logic symbol for a JK flip-flop with active low preset and clear inputs is shown in the Fig. 5.4.32 (a). These inputs are active low, therefore they must both be kept high for synchronous operation.



(a) Logic symbol

(b) JK flip-flop with active high preset and clear  
Fig. 5.4.32

- The Fig. 5.4.33 illustrates the operation of active low preset and clear inputs.

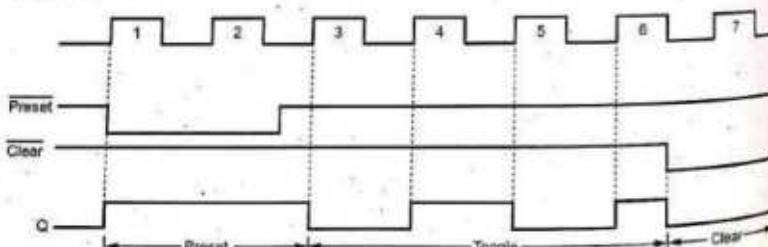
Assume  $J = K = 1$ 

Fig. 5.4.33 Input and output waveforms showing operation of active low preset and clear inputs

- During clock pulse 1 and 2 the Preset is low, keeping the flip-flop set regardless of the synchronous inputs.
- For clock pulses 3, 4, 5 and 6, toggle operation occurs because J and K both are high with preset and clear inactive.
- For clock pulse 7, the Clear input is low, keeping the flip-flop reset regardless of the synchronous inputs.
- In some IC packages preset and clear inputs are active high. In those cases preset and clear input must both be kept low for synchronous operation.

**Example 5.4.5** 1. For the Fig. 5.4.34, 5.4.35 and 5.4.36, plot the output waveforms referenced to the clock signal assuming the initial contents of all FFs is  $Q = 0$ . Assume all FFs are edge triggered.

GTU : Winter-15, Marks 7

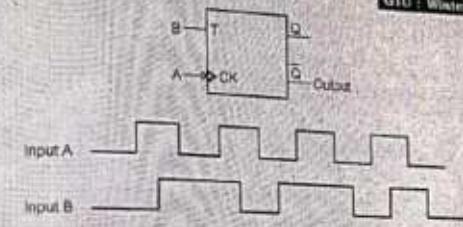


Fig. 5.4.34



Fig. 5.4.35

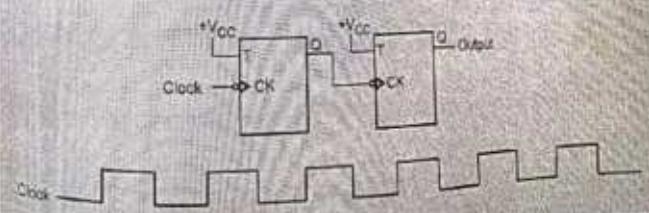


Fig. 5.4.36

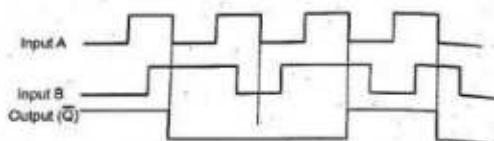
**Solution :**Note : In T flip-flop output toggles when  $T = 1$ 

Fig. 5.4.34 (a)

2.

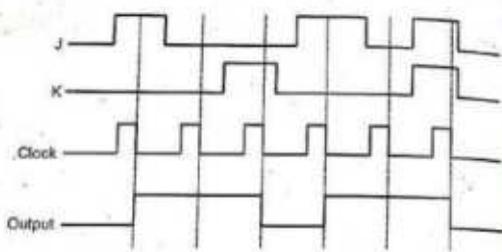


Fig. 5.4.35

3.

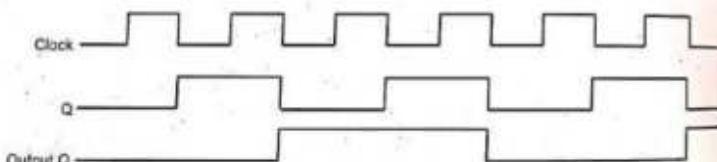


Fig. 5.4.36

**Example 5.4.5** 1. Fill in values for S and R to cause the Q values of the SR FF given in Fig. 5.4.37.

	$t_0$	$t_1$	$t_2$	$t_3$
S	0			
R	0			
Q	1	0	0	1

Fig. 5.4.37

GTU : Winter-15, Marks 3

**Solution :**

- $t_1 : S=0, R=1 \quad X : \text{don't care}$
- $t_2 : S=0, R=X$
- $t_3 : S=1, R=0$

**Example 5.4.7** Plot the output waveform for the inputs shown in Fig. 5.4.38, assuming the initial contents of the FF is  $Q = 0$ .

GTU : Winter-15, Marks 3

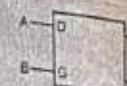


Fig. 5.4.38

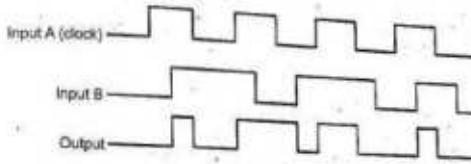
**Solution :**

Fig. 5.4.38 (a)

**Example 5.4.8** Plot the out waveform referenced to the clock signal assuming the initial contents of the flip-flops is  $q = 0$ . Assume all flip-flops are edge triggered.

GTU : Summer-17, Marks 3

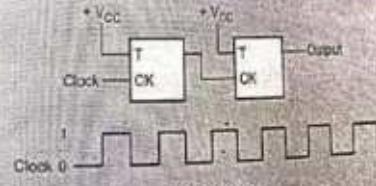


Fig. 5.4.39

**Solution : Assuming flip-flops are positive edge trigger :**

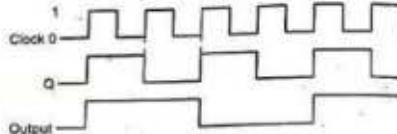


Fig. 5.4.39 (a)

#### Review Questions

1. Define the following term : Flip Flop. GTU : Dec-13, Marks 1
2. Differentiate between latches and flip-flops.
3. Explain the different types of triggering methods used for flip-flops.
4. What is S-R flip-flop ? Explain working of clocked SR flip-flop. What is edge triggering ?
5. Explain the clocked SR flip-flop using NAND gates.
6. Explain D type positive edge triggered flip flop. GTU : May-13, Marks 2
7. If Q output of a D-type flip-flop is connected to D input, it acts as a toggle switch. State whether true or false ? justify your answer.
8. Write a brief note on edge-triggered SR and JK flip-flops. GTU : May-14, Marks 2
9. Justify name delay for D flip-flop.
10. Explain the application of D flip-flop.
11. Explain the operation of JK flip-flop with truth table.
12. Draw the JK flip-flop using NAND gates.
13. Explain race around condition.
14. Discuss method to avoid race around condition in JK flip flop.
15. Explain MS J-K flip-flop. GTU : May-14, Dec-13, May-12, Winter-17, Marks 1
16. Draw and explain the operation of T flip-flop.
17. Explain the application of T flip-flop.
18. Explain the use of reset and preset inputs.
19. Draw the truth table of full adder and implement using minimum number of logic gates. GTU : Summer-15, Marks 2
20. With the help of function table and circuit diagram explain the working of clocked SR flip flop. GTU : Summer-15, Marks 2
21. Draw the truth tables for JK and T flip-flop. STU : Summer-18, Marks 4
22. Explain working of master - slave JK flip-flop with necessary logic diagram, state equation and state diagram. GTU : Winter-17, Marks 7

23. Draw the truth table for JK and TFF. Using these truth tables, derive and explain the excitation tables of JK and T FF. GTU : Summer-18, Marks 3
24. Discuss Clocked R-S Flip-flop with Logic diagram, Symbol, Characteristic z-table and characteristic equation. GTU : Winter-18, Marks 7
25. Draw the characteristic table of following Flip-flop.  
(i) R-S (ii) J - K (iii) E

GTU : Winter-18, Marks 3

GTU : Summer-15, 18

#### 5.5 Excitation Tables

- During the design process we know, from the transition table, the sequence of states, i.e., the transition from each present state to its corresponding next state. From this information we wish to find the flip-flop input conditions that will cause the required transition. For this reason, we need a table that lists the required inputs for a given change of state. Such a table is known as an excitation table of the flip-flop.
- We can derive the excitation tables for flip-flops from their truth tables.
- The excitation table consists of two columns  $Q_n$  and  $Q_{n+1}$ , and a column for each input to show how the required transition can be achieved.

##### 5.5.1 SR Flip-Flop

- Table 5.5.1 (a) and (b) show the truth table and excitation tables for SR flip-flop, respectively.
- There are four possible transitions from the present state to the next state.
- For each transition, the required input condition is derived from the information available in the truth table.

S	R	$Q_{n+1}$
0	0	$Q_n$
1	0	1
0	1	0
1	1	-

(a) SR truth table

$Q_n$	$Q_{n+1}$	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

(b) SR excitation table

Table 5.5.1

- Note** The symbol "X" in the table represents a don't care condition, i.e., it indicates that to get required output it does not matter whether the input is either 1 or 0.
- **0 → 0 Transition :** The present state of the flip-flop is 0 and is to remain 0 when a clock pulse is applied. Looking at truth table of SR flip-flop we can understand that, this can happen either when  $R = S = 0$  (no-change condition) or when  $R = 1$  and  $S = 0$ . Thus, S has to be at 0, but R can be at either level. The table indicates this with a "0" under S and an "X" (don't care) under R.

- 0 → 1 Transition :** The present state is 0 and is to change to 1. This can happen only when S = 1 and R = 0 (set condition). Therefore, S has to be 1 and R has to be 0 for this transition to occur.
- 1 → 0 Transition :** The present state is 1 and is to change to a 0. This can happen only when S = 0 and R = 1 (reset condition). Therefore, S has to be 0 and R has to be 1 for this transition to occur.
- 1 → 1 Transition :** The present state is 1 and is to remain 1. This can happen either when S = 1 and R = 0 (set condition) or when S = 0 and R = 0 (no change condition). Thus R has to be 0, but S can be at either level. The table indicates this with a "X" under S and "0" under R.

### 5.5.2 JK Flip-Flop

- The truth table and excitation table for JK flip-flop are shown in Table 5.5.2 (a) and (b) respectively.
- 0 → 0 Transition :** When both present state and next state are 0, the J input must remain at 0 and the K input can be either 0 and 1.
- 0 → 1 Transition :** The present state is 0 and is to change to 1. This can happen either when J = 1 and K = 0 (set condition) or when J = K = 1 (toggle condition). Thus, J has to be 1, but K can be at either level for this transition to occur.
- 1 → 0 Transition :** The present state is 1 and is to change to 0. This can happen either when J = 0 and K = 1 or when J = K = 1. Thus, K has to be 1 but J can be at either level.
- 1 → 1 Transition :** When both present state and next are 1, the K input must remain at 0 while the J input can be 0 or 1.
- The excitation table for JK flip-flop has more don't care conditions than the excitation table for RS flip-flop.
- The don't care terms usually simplify the function. Therefore, the combinational circuits using JK flip-flops for the input functions are likely to be simpler than those using RS flip-flops.

J	K	$Q_{n+1}$
0	0	$Q_n$
0	1	0
1	0	1
1	1	$Q_n$

(a) JK truth table

$Q_n$	$Q_{n+1}$	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

(b) JK excitation table

Table 5.5.2

### 5.5.3 D Flip-Flop

- The Table 5.5.3 (a) and (b) show the truth table and excitation table for D flip-flop, respectively.

In D flip-flop, the next state is always equal to the D input and it is independent of the present state. Therefore, D must be 0 if  $Q_{n+1}$  has to be 0, and 1 if  $Q_{n+1}$  has to be 1, regardless of the value of  $Q_n$ .

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

(a) D truth table

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

(b) D excitation table

Table 5.5.3

### 5.5.4 T Flip-Flop

- The Table 5.5.4 (a) and (b) show the truth table and the excitation table for T flip-flop, respectively.

When input T = 1, the state of the flip-flop is complemented; when T = 0, the state of the flip-flop remains unchanged. Therefore, for 0 → 0 and 1 → 1 transitions T must be 0 and for 0 → 1 and 1 → 0 transitions T must be 1.

T	$Q_{n+1}$
0	$Q_n$
1	$\bar{Q}_n$
1	$\bar{Q}_n$

Table 5.5.4

### Review Questions

- Derive the excitation tables for SR, D, JK and T flip-flop.
- Draw the truth table of full subtractor and implement using minimum number of logic gates.
- Derive and explain the excitation tables of JK and T FF.

GTU : Summer-13, Marks 7

GTU : Summer-18, Marks 4

### 5.6 Conversion from One Type to another Type of Flip-Flop

GTU : Summer-15

- It is possible to convert one flip-flop into another flip-flop with some additional gates or simply doing some extra connection.

**5.6.1 SR Flip-Flop to D Flip-Flop**

- The excitation table for above conversion is as shown in Table 5.6.1.

Input		Present state	Next state	Flip-flop inputs	
D	Q <sub>n</sub>	Q <sub>n+1</sub>	S	R	
0	0	0	0	X	
0	1	0	0	1	
1	0	1	1	0	
1	1	1	X	0	

Table 5.6.1

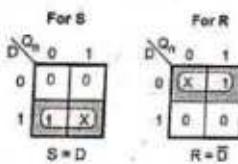
**K-map simplification**

Fig. 5.6.1

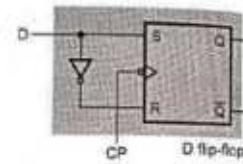
**Logic diagram**

Fig. 5.6.2 SR to D flip-flop conversion

**5.6.2 SR Flip-Flop to JK Flip-Flop**

- The excitation table for above conversion is as shown in Table 5.6.2.

Inputs		Present state	Next state	Flip-flop inputs	
J	K	Q <sub>n</sub>	Q <sub>n+1</sub>	S	R
0	0	0	0	0	X
0	0	1	1	X	0
0	1	0	0	0	X
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	1	X	0
1	1	0	1	1	0
1	1	1	0	0	1

Table 5.6.2

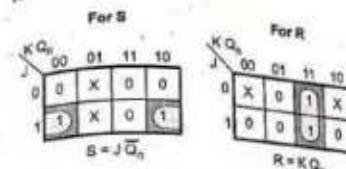
**K-map simplification**

Fig. 5.6.3

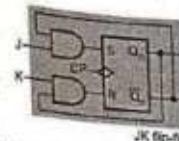
**Logic diagram**

Fig. 5.6.4 SR to JK flip-flop conversion

**5.6.3 SR Flip-Flop to T Flip-Flop**

- The excitation table for above conversion is as shown in the Table 5.6.3.

Input		Present state	Next state	Flip-flop inputs	
T	Q <sub>n</sub>	Q <sub>n+1</sub>	S	R	
0	0	0	0	0	X
0	1	1	1	X	0
1	0	0	1	1	0
1	1	0	0	0	1

Table 5.6.3

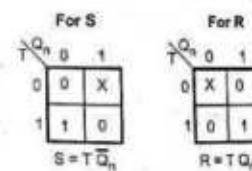
**K-map simplification**

Fig. 5.6.5

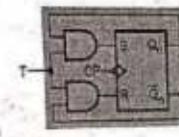
**Logic diagram**

Fig. 5.6.6 SR to T flip-flop conversion

- If we apply clock pulses to the circuit, the circuit output will toggle from 0 to 1 and 1 to 0. Thus, we can build 1-bit counter using SR flip-flop by converting it to T flip-flop.

**Example 5.6.1** Prepare the truth table for the circuit of Fig. 5.6.7 and show that it acts as a T-type flip-flop.

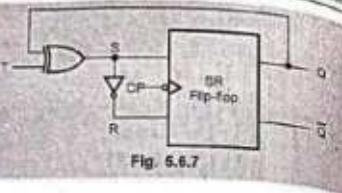


Fig. 5.6.7

**Solution :** For SR flip-flop,

$$\begin{aligned} Q_{n+1} &= S + \bar{R}Q_n \\ &= S + S Q_n \\ &= S(1 + Q_n) = S \end{aligned} \quad \text{... Characteristics equation}$$

We have,  $S = Q_n \oplus T$

$$\begin{aligned} \therefore Q_{n+1} &= Q_n \oplus T \\ &= T\bar{Q}_n + \bar{T}Q_n \end{aligned} \quad \text{... Characteristic equation of T flip-flop}$$

C <sub>P</sub>	T	Q <sub>n</sub>	S = Q <sub>n</sub> ⊕ T	R = S	Q <sub>n+1</sub> = S
↓	0	0	0	1	0
↓	0	1	1	0	1
↓	1	0	1	0	1
↓	1	1	0	1	0

Table 5.6.4 Truth table for the given circuit

- Looking at column 1 and column 5 of the Table 5.6.4 we can conclude that when  $T = 0$ , the output does not change and when  $T = 1$ , the output toggles. Thus, the given circuit acts as a T flip-flop. This is another way of implementing T flip-flop using SR flip-flop.

#### 5.6.4 JK Flip-Flop (OR MSJK) to T Flip-Flop

- The excitation table for above conversion is as shown in Table 5.6.5.

Input	Present state	Next state	Flip-flop inputs
T	Q <sub>n</sub>	Q <sub>n+1</sub>	J <sub>A</sub> K <sub>A</sub>
0	0	0	0 X
0	1	1	X 0
1	0	1	1 X
1	1	0	X 1

Table 5.6.5

#### K-map simplification

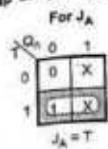
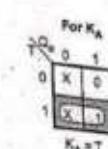
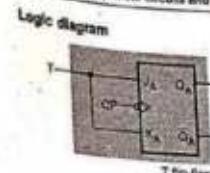
J<sub>A</sub> = TK<sub>A</sub> = T

Fig. 5.6.9 JK to T flip-flop conversion

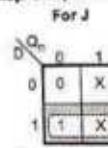
#### 5.6.5 JK Flip-Flop (OR MSJK) to D Flip-Flop

- The excitation table for above conversion is as shown in the Table 5.6.6.

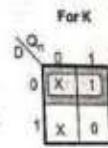
Input	Present state	Next state	Flip-flop inputs
D	Q <sub>n</sub>	Q <sub>n+1</sub>	J K
0	0	0	0 0
0	1	1	0 X
1	0	1	1 1
1	1	0	X 0

Table 5.6.6

#### K-map simplification



J = D



K = D-bar

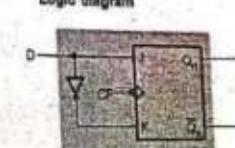


Fig. 5.6.10

Fig. 5.6.11 JK to D flip-flop conversion

#### 5.6.6 D Flip-Flop to T Flip-Flop

- The excitation table for above conversion is as shown in the Table 5.6.7.

Input	Present state	Next state	Flip-flop input
T	Q <sub>n</sub>	Q <sub>n+1</sub>	D
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

Table 5.6.7

## K-map simplification

For D	
T	Q <sub>n</sub>
0	0
1	1

Fig. 5.6.12

## Logic diagram

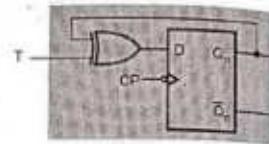


Fig. 5.6.13 D to T flip-flop conversion

$$D = \bar{T}Q_n + T\bar{Q}_n = T \oplus Q_n$$

**Example 5.6.2** Analyze the circuit and prove that it is equivalent to T flip-flop.

Solution : To analyze the circuit means to derive the truth table for it.

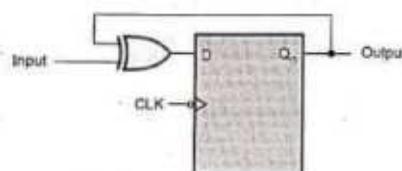


Fig. 5.6.14

We have,  $D = \text{Input} \oplus Q_n$

CLK	Input	Q <sub>n</sub>	D = Input $\oplus$ Q <sub>n</sub>	Q <sub>n+1</sub>
↓	0	0	0	0
↓	0	1	1	1
↓	1	0	1	1
↓	1	1	0	0

Table 5.6.8 Truth table for given circuit

- In the above circuit, output does not change when input is 0 and it toggles when input is 1. This is the characteristics of T flip-flop. Hence, the given circuit is T flip-flop constructed using D flip-flop.

## 5.6.7 T Flip-Flop to D Flip-Flop

The excitation table for above conversion is as shown in Table 5.6.9.

Input	Present state	Next state	Flip-flop input
D	Q <sub>n</sub>	Q <sub>n+1</sub>	T
0	0	0	0
0	1	0	0
1	0	1	1
1	1	1	0

Table 5.6.9

## K-map simplification

For T	
D	Q <sub>n</sub>
0	0
1	0

$$T = D \bar{Q}_n + \bar{D} Q_n$$

## Logic diagram



Fig. 5.6.15

Fig. 5.6.16 T to D flip-flop conversion

## 5.6.8 JK Flip-Flop (OR MSJK) to SR Flip-Flop

The excitation table for above conversion is as shown in Table 5.6.10.

Inputs	Present state	Next state	Flip-flop inputs
S R	Q <sub>n</sub>	Q <sub>n+1</sub>	J K
0 0	0	0	0 0 X
0 0	1	1	X 0 0
0 1	0	0	0 0 X
0 1	1	0	0 X 1
1 0	0	1	1 1 X
1 0	1	1	1 X 0
1 1	0	X	X X X
1 1	1	X	X X X

Table 5.6.10 Excitation table for JK to SR conversion

**K-map simplification**

		For J		For K		
		SR	Q <sub>n</sub>	SR	Q <sub>n</sub>	
00	0	0	X	00	X	0
01	0	X	X	01	X	1
11	X	X	X	11	X	X
10	1	X	X	10	0	X

J = S  
K = R

Fig. 5.6.17

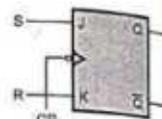
**Logic diagram**

Fig. 5.6.18 JK to SR flip-flop conversion

**5.6.9 D Flip-Flop to SR Flip-Flop**

- The excitation table for above conversion is as shown in the Table 5.6.11.

		Present state		Flip-flop input
S	R	Q <sub>n</sub>	Q <sub>n+1</sub>	D
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1
1	1	0	X	X
1	1	1	X	X

Table 5.6.11 Excitation table for D to SR conversion

**K - map simplification**

		For D			
		SR	Q <sub>n</sub>	SR	Q <sub>n</sub>
0	0	00	01	11	10
1	1	11	X	X	X

$D = \overline{R} Q_n + S$

Fig. 5.6.19

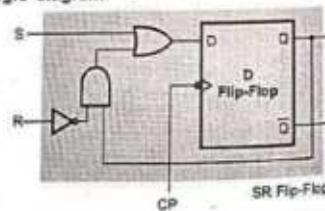
**Logic diagram**

Fig. 5.6.20 D to SR flip-flop conversion

TECHNICAL PUBLICATIONS™ - An up thrust for knowledge

**5.6.10 T Flip-Flop to SR Flip-Flop**

- The excitation table for conversion of T FF into an SR FF is as shown in the Table 5.6.12.

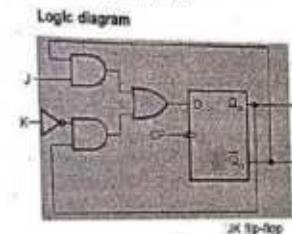
		Present state		Flip-flop input
S	R	Q <sub>n</sub>	Q <sub>n+1</sub>	T
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
1	0	0	1	1
1	0	1	1	1
1	1	0	1	0
1	1	1	X	X

Table 5.6.12 Excitation table for T to SR conversion

**K - map simplification**

		For D					
		J	K	00	01	11	10
0	0	0	1	0	0	0	0
1	1	1	1	1	0	1	1

(a)



(b)

Fig. 5.6.21

**5.6.11 D Flip-Flop to JK Flip-Flop**

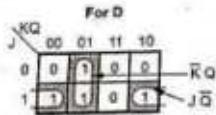
- The excitation table for conversion of D flip-flop to JK flip-flop :

		Present state		Flip-flop input
J	K	Q <sub>n</sub>	Q <sub>n+1</sub>	D
0	0	0	0	0
0	0	1	1	1

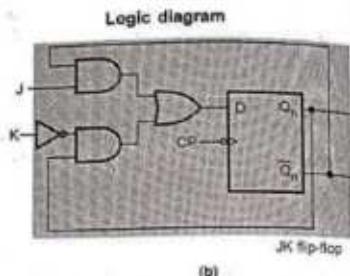
TECHNICAL PUBLICATIONS™ - An up thrust for knowledge

C	I	Q	Q̄
0	0	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

K-map simplification



(a)



(b)

Fig. 5.6.22

## Review Questions

- Convert SR flip-flop to D flip-flop.
- Convert SR flip-flop to JK flip-flop.
- Convert SR flip-flop to T flip-flop.
- Convert JK flip-flop to T flip-flop.
- Convert JK flip-flop to D flip-flop.
- Convert D flip-flop to T flip-flop.
- Convert T flip-flop to D flip-flop.
- Convert JK flip-flop to SR flip-flop.
- Convert D flip-flop to SR flip-flop.
- Convert D flip flop into SR flip flop.

GTU Summer-15, Marks 7

- It is used as a basic building block in sequential circuits such as counters and registers.
- It can be used as a delay element.

## 5.7.1 Bounce Elimination Switch

- For interfacing keys to the digital systems, usually push button keys are used.
- Push button keys when pressed bounce a few times, closing and opening the contacts before providing a steady reading, as shown in the Fig. 5.7.1.

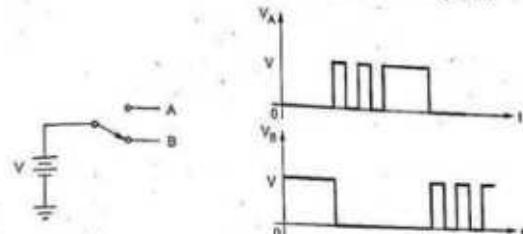


Fig. 5.7.1 Effect of key debounce

- Reading taken during bouncing period may be faulty. This problem is known as key debounce.
- The problem of key debounce is undesirable and it must be avoided.
- One way to avoid key debounce problem is to use SR latch.
- The circuit used to avoid keybounce with SR latch is called a switch or contact debouncer.
- The Fig. 5.7.2 shows the switch debouncer circuit and its waveforms.
- When key is at position A, the output of SR latch is logic 1, and when key is at position B, the output of SR latch is logic 0.
- When key is in between A and B, SR inputs are 00 and hence output does not change, preventing debouncing of key output.

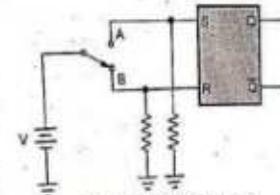


Fig. 5.7.2 (i) Switch debouncer

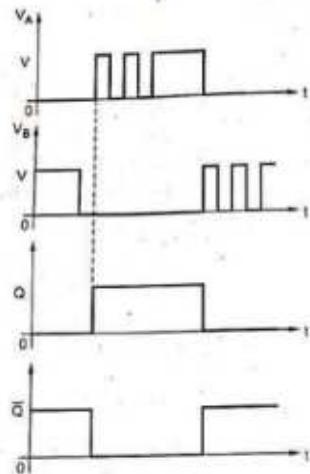


Fig. 5.7.2 Waveforms of switch debouncer

- We can say that the output does not change during transition period, eliminating key debounce.

### 5.7.2 Registers

- A group of flip-flops connected together forms a register.
- A register is used solely for storing and shifting data which is in the form of 1s and/or 0s, entered from an external source.
- Fig. 5.7.3 shows 3-bit register using three D flip-flops.
- Preset, reset and clock inputs are connected in parallel and 3-bit input is applied to D inputs of the flip-flop.
- During positive edge of the clock, D inputs of the flip-flops are stored within the flip-flop and are available at the outputs of D flip-flops.

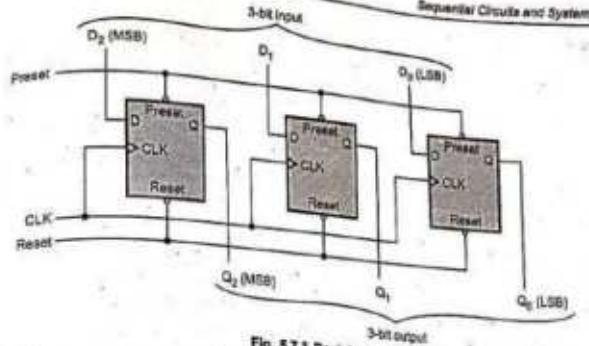


Fig. 5.7.3 Register

### 5.7.3 Counters

- A counter is a register capable of counting the number of clock pulses arriving at its clock input.
- Count represents the number of clock pulses arrived.
- On arrival of each clock pulse, the counter is incremented by one.
- In case of down counter, it is decremented by one.
- Fig. 5.7.4 shows 3-bit counter.
- It consists of three flip-flops.
- A counter with n flip-flops has  $2^n$  possible states.
- The 3-bit counter can count from decimal 0 to 7.

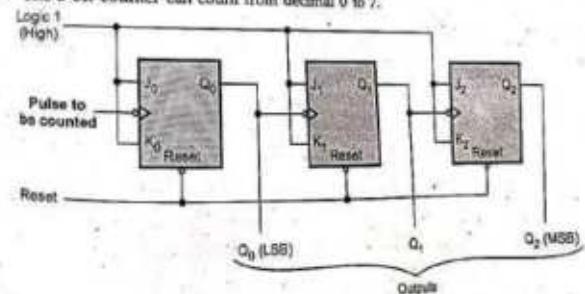


Fig. 5.7.4 Counter

**5.7.4 | Frequency Division**

- A frequency divider, also called a clock divider or scaler or prescaler, is a circuit that takes an input signal of a frequency,  $f_{in}$ , and generates an output signal of a frequency :

$$f_{out} = \frac{f_{in}}{n}$$

where  $n$  is an integer.

- For power-of-2 integer division, a simple binary counter can be used, clocked by the input signal. The least-significant output bit alternates at 1/2 the rate of the input clock, the next bit at 1/4 the rate, the third bit at 1/8 the rate, etc.
- An arrangement of flip-flops are a classic method for integer- $n$  division.
- The easiest configuration is a series where each flip-flop is a divide-by 2. For a series of three of these, such system would be a divide-by-8.
- By adding additional logic gates to the chain of flip-flops, other division ratios can be obtained.
- The Fig. 5.7.6 shows frequency divider using D flip-flops. Here, D flip-flops are used as toggle flip-flops. After each stage frequency is divided by 2.

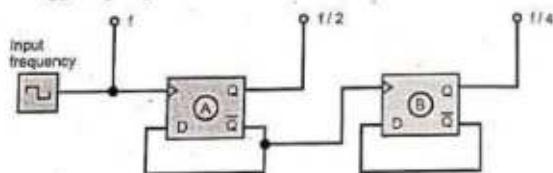


Fig. 5.7.5 Frequency divider using D flip-flops

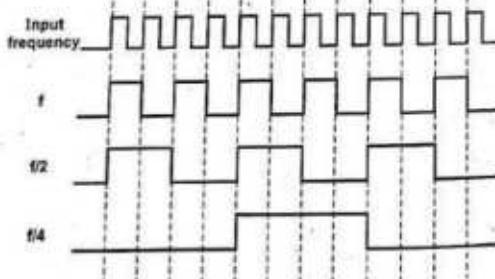


Fig. 5.7.6 Frequency divider waveform

TECHNICAL PUBLICATIONS™ An up beat for knowledge

**5.7.5 | Clock System with Delayed Clock Signal**

- In many digital systems, a clock signal with 50% duty cycle is used to drive the digital system and in turn it initiates actions in various components. As the transition occurs only every half-period for a 50% duty cycle clock, it helps to generate a second clock signal with a one-quarter delay of clock period. Thus, the occurrence of transition increases due to both the clock signal and delayed clock signal.

- Fig. 5.7.7 shows the clock system with delayed clock signal. The oscillator generates a clock signal C and its delayed version ( $C_D$ ) is obtained by inverting the clock signal C. As shown in Fig. 5.7.8 the signal  $Q_A$  generates clock transition at the beginning and at the one-half point of the period and signal  $Q_B$  generates clock transition at the beginning and at the one-quarter point of the period.
- The main disadvantage of this technique is that the oscillator need to generate an output frequency that is twice the required frequency.

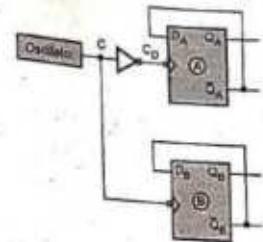


Fig. 5.7.7 Clock system with delayed clock signal

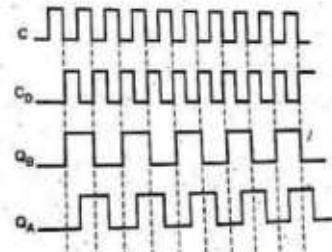


Fig. 5.7.8 Waveforms of clock system with delayed clock signal

TECHNICAL PUBLICATIONS™ An up beat for knowledge

**5.7.6 A Two Phase Clock Circuit**

- Fig. 5.7.9 shows the two phase clock circuit that consists of two separate signals and each of them go high for a portion of the clock cycle but not simultaneously.
- The glitches at point a on  $\phi_1$  and at point b on  $\phi_2$  can be eliminated by the use of delaying inverters.
- The signal  $C_D$  must go low before signal C goes high so as to avoid glitch at point a on  $\phi_1$ . Same is the case at point b on  $\phi_2$ .
- The delay introduced by the inverters must exceed introduced by flip-flops.

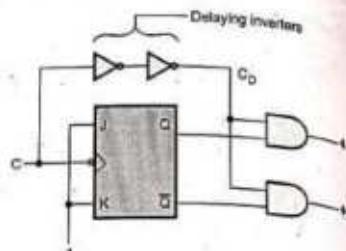


Fig. 5.7.9 Two phase clock circuit

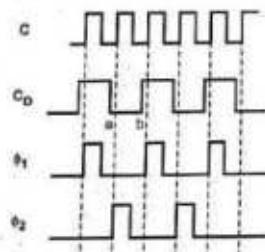


Fig. 5.7.10 Waveforms of two phase clock circuit

**Review Questions**

- List the applications of flip-flops.
- How keyboard debouncing is eliminated using flip-flop? Explain with suitable circuit diagram.
- Write a short note on frequency division.
- Explain in detail clock system with delayed clock signal.
- Write a note on two phase clock circuit.

**5.8 Registers and Shift Registers**

GTU : Drx-13, May-13, 14, Summer-13, 14, Winter-14

- A group of flip-flops can be used to store a word, which is called register.
- A flip-flop can store 1-bit information. So an n-bit register has a group of n flip-flops and is capable of storing any binary information/number containing n-bits.

**Buffer Register**

- Fig. 5.8.1 shows the simplest register constructed with four D flip-flops. This register is also called buffer register.

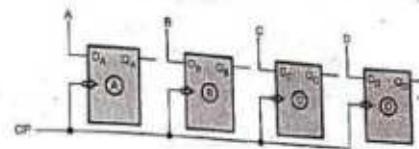


Fig. 5.8.1 Buffer register

- Each D flip-flop is triggered with a common negative edge clock pulse.
- The input bits set up the flip-flops for loading.
- When the first negative clock edge arrives, the stored binary information becomes,  $Q_AQ_BQ_CQ_D = ABCD$
- In this register, four D flip-flops are used. So it can store 4-bit binary information.
- The number of flip-flop stages in a register determines its total storage capacity.

**Controlled Buffer Register**

- We can control input and output of the register by connecting tri-state devices at the input and output sides of register as shown in Fig. 4.8.2. So this register is called 'controlled buffer register'.
- Here, tri-state switches are used to control the operation.
- When you want to store data in the register, you have to make LOAD or WR signal low to activate the tri-state buffers.
- When you want the data at the output, you have to make RD signal low to activate the buffers.
- Controlled buffer registers are commonly used for temporary storage of data within a digital system.

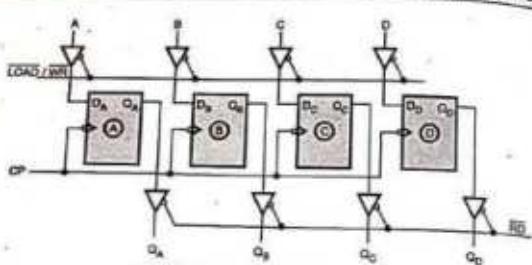


Fig. 5.8.2 Controlled buffer register

- As seen above the 4-bit register can store 4-bit binary information. In general,  $n$ -bit register can store  $n$ -bit binary information.

**Example 5.8.1** Determine the number of flip-flops needed to construct a register capable of storing.

- A 6-bit binary number
- Decimal numbers up to 32.
- Hexadecimal numbers up to F
- Octal numbers up to 10.

Solution :

- A 6-bit binary number requires register with 6 flip-flops.
- $(32)_{10} = (100000)_2$ . The number of bits required to represent 32 in binary are six, therefore, 6 flip-flops are needed to construct a register capable of storing 32 decimal.
- $(F)_{15} = (1111)_2$ . The number of bits required to represent  $(F)_{15}$  in binary are four, therefore four flip-flops are needed to construct a register capable of storing  $(F)_{15}$ .
- $(10)_8 = (1000)_2$ . The number of bits required to represent  $(10)_8$  in binary are four, therefore, four flip-flops are needed to construct a register capable of storing  $(10)_8$ .

### 5.8.1 Shift Registers

- The binary information (data) in a register can be moved from stage to stage within the register or into or out of the register upon application of clock pulses. This type of bit movement or shifting is essential for certain arithmetic and logic operations used in microprocessors. This gives rise to a group of registers called 'shift registers'.

- They are very important in applications involving the storage and transfer of data in a digital system.
- Fig. 5.8.3 gives the symbolical representation of the different types of data movement in shift register operations.

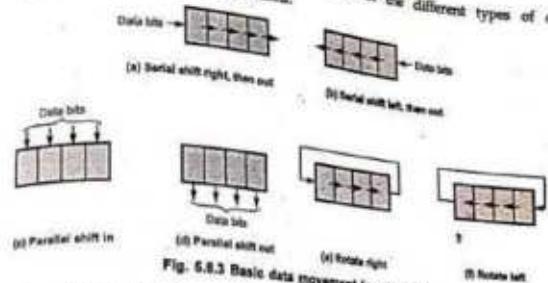


Fig. 5.8.3 Basic data movement in registers

- According to the data movement in a register, there are different types of shift registers.

### 5.8.2 Types of Shift Registers

#### 5.8.2.1 Serial In Serial Out (SISO) Shift Register

##### Shift Left Mode

- Fig. 5.8.4 shows serial-in serial-out shift-left register.

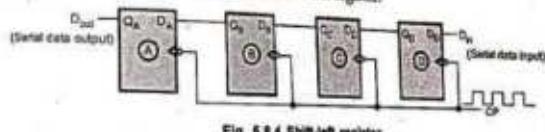
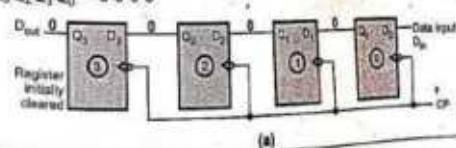


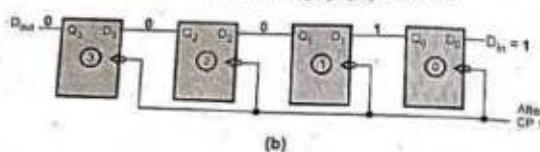
Fig. 5.8.4 Shift-left register

- We will illustrate the entry of the four bit binary number 1111 into the register, beginning with the left-most bit. Initially, register is cleared. So  $Q_3Q_2Q_1Q_0 = 0\ 0\ 0\ 0$



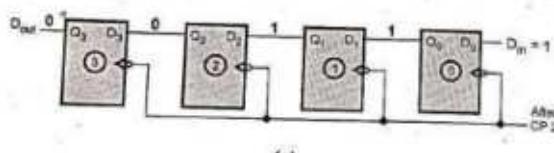
(a)

- a) When data 1 1 1 1 is applied serially, i.e. left-most 1 is applied as  $D_{in}$ ,  $D_{in} = 1$ ,  $Q_3 Q_2 Q_1 Q_0 = 0 0 0 0$ . The arrival of the first falling clock edge sets the right-most flip-flop, and the stored word becomes,  $Q_3 Q_2 Q_1 Q_0 = 0 0 0 1$



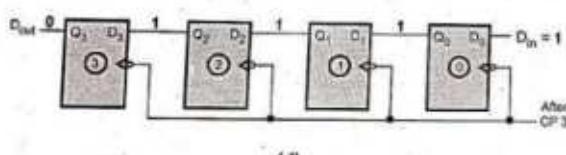
(b)

- b) When the next negative clock edge hits, the  $Q_1$  flip-flop sets and the register contents become,  $Q_3 Q_2 Q_1 Q_0 = 0 0 1 1$



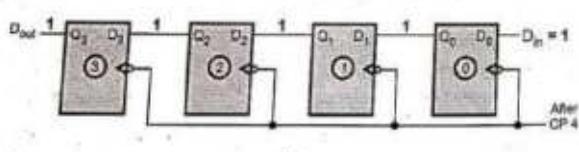
(c)

- c) The third negative clock edge result in,  $Q_3 Q_2 Q_1 Q_0 = 0 1 1 1$



(d)

- d) The fourth falling clock edge gives,  $Q_3 Q_2 Q_1 Q_0 = 1 1 1 1$ .



(e)

Fig. 5.8.5 Four bits 1111 being serially entered into shift-left register

- The Table 5.8.1 summarizes the shift left operation.

CP	$Q_3$	$Q_2$	$Q_1$	$Q_0$	$D_{in}$
Initial	0	0	0	0	-
t <sub>1</sub> <sup>↑</sup>	0	0	0	1	1
t <sub>2</sub> <sup>↑</sup>	0	0	1	1	-
t <sub>3</sub> <sup>↑</sup>	0	1	1	1	-
t <sub>4</sub> <sup>↑</sup>	1	1	1	1	-

Table 5.8.1 Shift left operation

- Fig. 5.8.6 shows waveforms for shift left operation

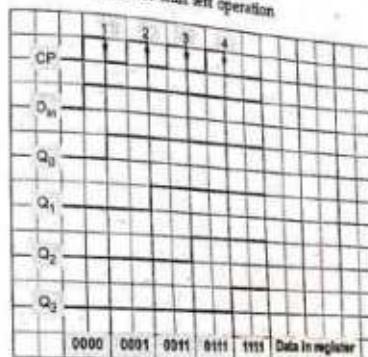


Fig. 5.8.6 Waveforms for shift left register

#### Shift Right Mode

- Fig. 5.8.7 shows serial-in serial-out shift-right register.

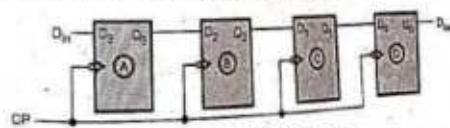
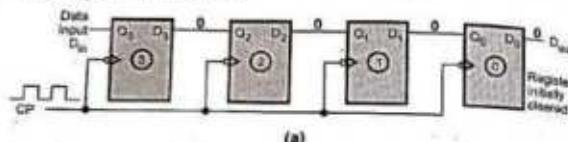


Fig. 5.8.7 Shift-right register

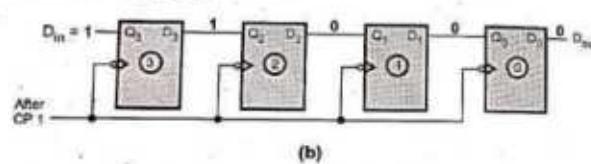
- We will illustrate the entry of the four bit binary number 1111 into the register, beginning with the left-most bit.



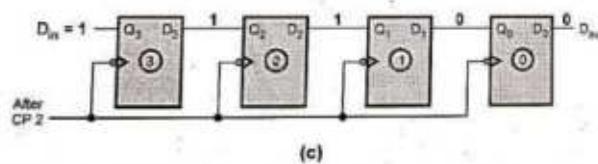
Initially, register is cleared. So  $Q_3Q_2Q_1Q_0 = 0\ 0\ 0\ 0$

- a) When data 1 1 1 1 is applied serially, i.e. left-most 1 is applied as  $D_{in}$   
 $D_{in} = 1, Q_3Q_2Q_1Q_0 = 0\ 0\ 0\ 0$

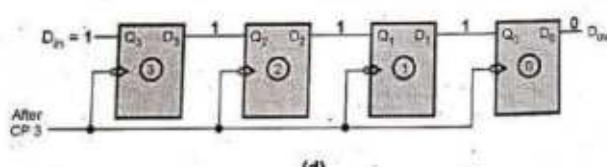
- The arrival of the first falling clock edge sets the left-most flip-flop, and the word becomes,  $Q_3Q_2Q_1Q_0 = 1\ 0\ 0\ 0$



- b) When the next falling clock edge hits, the  $Q_2$  flip-flop sets and the register contents become,  $Q_3Q_2Q_1Q_0 = 1100$



- c) The third falling clock edge results in,  $Q_3Q_2Q_1Q_0 = 1110$ .



- d) The fourth falling clock edge gives,  $Q_3Q_2Q_1Q_0 = 1111$ .

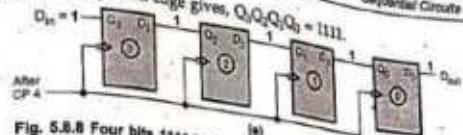


Fig. 5.8.8 Four bits 1111 being serially entered into shift-right register

- Table 5.8.2 summarizes the shift right operation.

CP	$D_{in}$	$Q_3$	$Q_2$	$Q_1$	$Q_0$
Initially	1	0	0	0	0
1	1	1	0	0	0
2	1	1	1	0	0
3	1	1	1	1	0
4	1	1	1	1	1

Table 5.8.2 Shift right operation

The Fig. 5.8.9 shows waveforms for shift right operation.

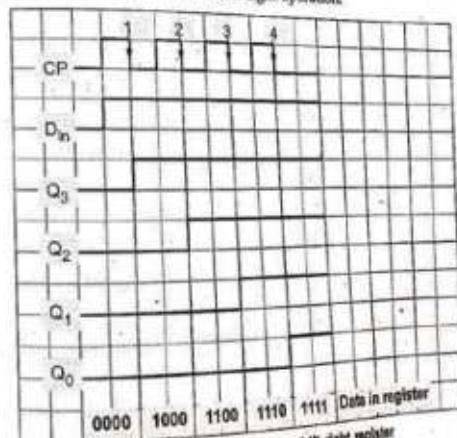


Fig. 5.8.9 Waveforms for shift right register

**5.8.2.2 Serial In Parallel Out (SiPO) Shift Register**

- The data bits are entered serially into the register but the output is taken in parallel.
- Once the data are stored, each bit appears on its respective output line and all bits are available simultaneously as shown in Fig. 5.8.10.

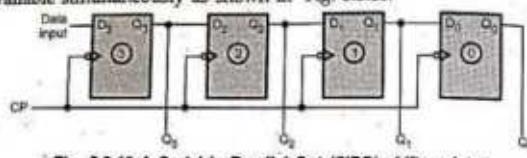


Fig. 5.8.10 A Serial In Parallel Out (SiPO) shift register

Table 5.8.3 Truth table

CP	$Q_3$	$Q_2$	$Q_1$	$Q_0$
-	NC	NC	NC	NC
↓	$D_3$	$D_2$	$D_1$	$D_0$

**5.8.2.3 Parallel In Serial Out (PISO) Shift Register**

- In this type, the bits are entered in parallel i.e. simultaneously into their respective stages on parallel lines.
- Fig. 5.8.11 illustrates a four-bit parallel in serial out register.

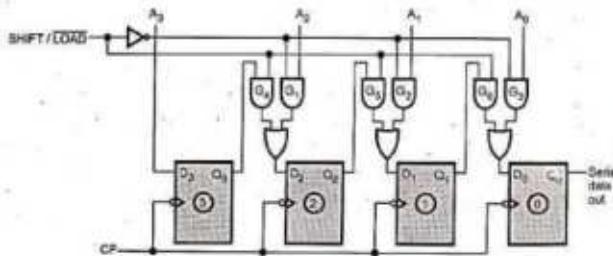


Fig. 5.8.11 Parallel In Serial Out (PISO) shift register

- There are four input lines  $A_3, A_2, A_1, A_0$  for entering data in parallel into the register.
- SHIFT/LOAD is the control input which allows shift or loading data operation of the register.
- When SHIFT/LOAD is low, gates  $G_1, G_2, G_3$  are enabled, allowing each input data bit to be applied to D input of its respective flip-flop.
- When a clock pulse is applied, the flip-flops with  $D = 1$  will SET and those with  $D = 0$  will RESET.
- All four bits are stored simultaneously.

- When SHIFT/LOAD is high, gates  $G_1, G_2, G_3$  are disabled and gates  $G_4, G_5, G_6$  are enabled. This allows the data bits to shift right from one stage to the next.
- The OR gates at the D-inputs of the flip-flops allow either the parallel data entry level or shift operation, depending on which AND gates are enabled by the level on the SHIFT/LOAD input.

**5.8.2.4 Parallel In Parallel Out (PIPO) Shift Register**

- In 'parallel in parallel out register', there is simultaneous entry of all data bits and the bits appear on parallel outputs simultaneously.
- Fig. 5.8.12 shows this type of register.

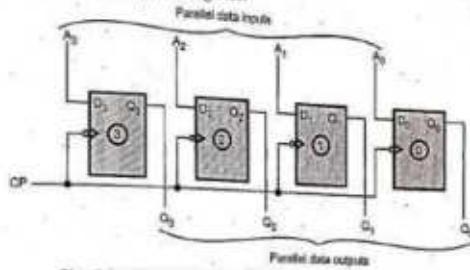


Fig. 5.8.12 Parallel In Parallel Out (PIPO) shift register

**5.8.2.5 Bidirectional Shift Register**

- This type of register allows shifting of data either to the left or to the right side. It can be implemented by using logic gate circuitry that enables the transfer of data from one stage to the next stage to the right or to the left, depending on the level of a control line.
- Fig. 5.8.13 illustrates a four-bit bidirectional register.

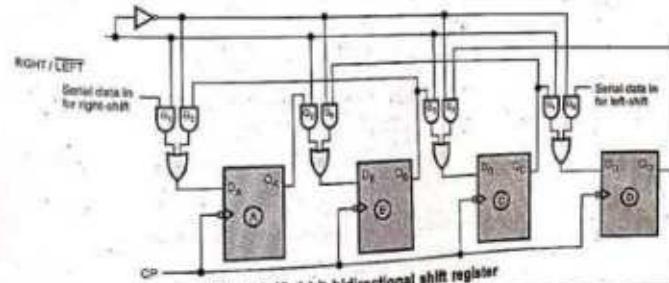


Fig. 5.8.13 4-bit bidirectional shift register

- The RIGHT/LEFT is the control input signal which allows data shifting either towards right or towards left.
- A high on this line enables the shifting of data towards right and a low enables it towards left.
- When RIGHT/LEFT signal is high, gates G<sub>1</sub>, G<sub>2</sub>, G<sub>3</sub>, G<sub>4</sub> are enabled.
- The state of the Q output of each flip-flop is passed through the D input of Q<sub>0</sub> following flip-flop.
- When a clock pulse arrives, the data are shifted one place to the right.
- When the RIGHT/LEFT signal is low, gates G<sub>5</sub>, G<sub>6</sub>, G<sub>7</sub>, G<sub>8</sub> are enabled.
- The Q output of each flip-flop is passed through the D input of the preceding flip-flop.
- When clock pulse arrives, the data are shifted one place to the left.

#### Bidirectional Shift Register with Parallel Load

- When parallel load capability is added to the shift register, the data entered in parallel can be taken out in serial fashion by shifting the data stored in the register. Such a register is called bidirectional shift register with parallel load.
- Fig. 5.8.14 shows bidirectional shift register with parallel load.
- As shown in the Fig. 5.8.14, the D input of each flip-flop has three sources: Output of left adjacent flip-flop, output of right adjacent flip-flop and parallel input. Out of these three sources one source is selected at a time and it is done with the help of decoder. The decoder select lines (SL<sub>1</sub> and SL<sub>0</sub>) select the one source out of three as shown in the Table 5.8.4.

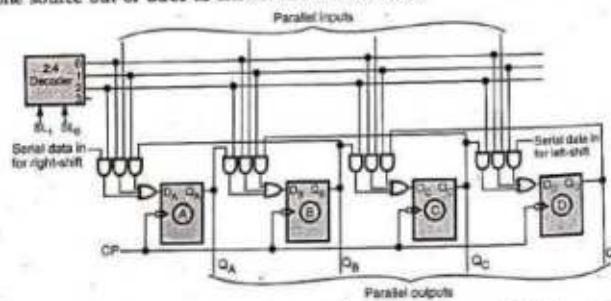


Fig. 5.8.14 4-bit bidirectional shift register with parallel load

- When select lines are 00 (i.e. SL<sub>1</sub> = 0 and SL<sub>0</sub> = 0), data from the parallel inputs is loaded into the 4-bit register.

TECHNICAL PUBLICATIONS™ - An up front for knowledge

- When select lines are 01 (i.e. SL<sub>1</sub> = 0 and SL<sub>0</sub> = 1), data within the register is shifted 1-bit left.
- When select lines are 10 (i.e. SL<sub>1</sub> = 1 and SL<sub>0</sub> = 0), data within the register is shifted 1-bit right.

SL <sub>1</sub>	SL <sub>0</sub>	Selected source
0	0	Parallel input
0	1	Output of right adjacent FF
1	0	Output of left adjacent FF

Table 5.8.4

#### Universal Shift Register

- A register capable of shifting in one direction only is a unidirectional shift register.
- A register capable of shifting in both directions is a bidirectional shift register.
- If the register has both shifts (right shift and left shift) and parallel load capabilities, it is referred to as universal shift register.
- The Fig. 5.8.15 shows the 4-bit universal shift register.

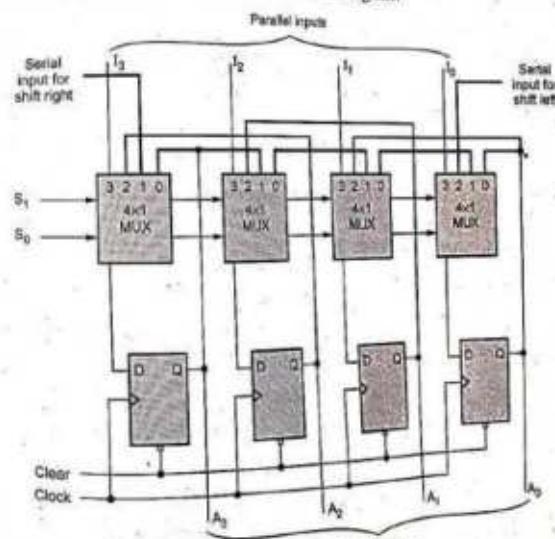


Fig. 5.8.15 4-bit universal shift register  
TECHNICAL PUBLICATIONS™ - An up front for knowledge

- It consists of four flip-flops and four multiplexers.
- The four multiplexers have two common selection inputs  $S_1$  and  $S_0$ , and they select appropriate input for D flip-flop.
- The Table 5.8.5 shows the register operation depending on the selection inputs of multiplexers.
- When  $S_1S_0 = 00$ , input 0 is selected and the present value of the register is applied to the D inputs of the flip-flops. This results no change in the register value.
- When  $S_1S_0 = 01$ , input 1 is selected and circuit connections are such that it operates as a right shift register.
- When  $S_1S_0 = 10$ , input 2 is selected and circuit connections are such that it operates as a left shift register.
- Finally, when  $S_1S_0 = 11$ , the binary information on the parallel input lines is transferred into the register simultaneously and it is a parallel load operation.

**Example 5.8.2** Draw the logic diagram of a 4-bit shift register with four D flip-flops and four  $4 \times 1$  multiplexer with mode selection inputs  $S_1$  and  $S_0$ . The register operates as follows:

$S_1$	$S_0$	Register operation
0	0	No change
0	1	Complement
1	0	Clear to 0
1	1	Load parallel data

**Solution :** The Fig. 5.8.16 shows the register that does the given operations.

(See Fig. 5.8.16 on next page)

#### 5.8.4 Applications of Shift Registers

- Primary use of shift register is temporary data storage and bit manipulations.
- Some of the common applications of shift registers are as discussed below.

##### 5.8.4.1 Delay Line

- A Serial-In-Serial-Out (SISO) shift register can be used to introduce time delay in digital signals. The time delay can be given as

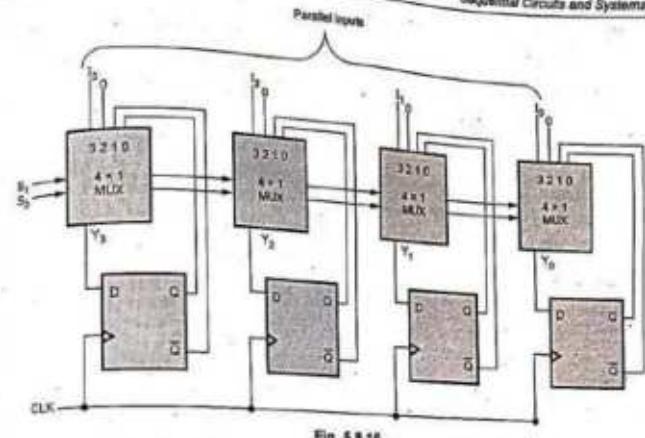


Fig. 5.8.16

$$\Delta t = N \times \frac{1}{f_c}$$

where  $N$  is the number of stages (i.e. flip-flops) and  $f_c$  is the clock frequency.

- An input pulse train appears at the output delayed by  $\Delta t$ .
- The amount of delay can be controlled by the clock frequency or by the number of flip-flops in the shift register.

##### 5.8.4.2 Serial-to-Parallel Converter

- A Serial-In-Parallel-Out (SIPO) shift register can be used to convert data in the serial form to the parallel form.

##### 5.8.4.3 Parallel-to-Serial Converter

- A Parallel-In-Serial-Out (PISO) shift register can be used to convert data in the parallel form to the serial form.

##### 5.8.4.4 Shift Register Counters

- A shift register with the serial output connected back to the serial input is called shift register counter.
- The most common shift register counters are the ring counter and the Johnson counter.

**5.8.4.5 Pseudo-Random Binary Sequence (PRBS) Generator**

- A shift register can be used as a pseudo-random binary sequence generator.
- A suitable feedback is used to generate pseudo-random sequence.
- The term random here means that the outputs do not cycle through a normal binary count sequence.
- The term pseudo here refers to the fact that the sequence is not truly random because it does cycle through all possible combinations once every  $2^n - 1$  clock cycles, where  $n$  represents the number of shift register stages (number of flip-flops).

**5.8.4.6 Sequence Generator**

- The shift register can be used to generate a particular bit pattern repetitively.
- The Fig. 5.8.17 shows the basic block diagram of a sequence generator.
- Left most flip-flop input accepts the serial input and the right most flip-flop gives serial data output.
- The serial data output signal is connected as a serial data in.
- On every clock pulse the data shift operation takes place.
- The loaded bit pattern at the serial output is in a sequence.
- Same bit pattern is again loaded in the register since serial output is connected serial in of the register. Thus, the circuit generates a particular bit pattern repetitively.

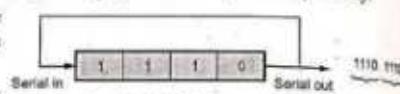


Fig. 5.8.17 4-bit sequence generator

**5.8.4.7 Sequence Detector**

- The shift register can be used to detect the desired sequence.
- The detection process requires two registers : one register stores the bit pattern to be detected i.e.  $R_1$  and other register accepts the input data stream i.e.  $R_2$ .
- Input data stream enters a shift register as serial data in and leaves as serial out.
- In every clock cycle, bit-wise comparisons of these two registers are done using EX-NOR gates as shown in the Fig. 5.8.18. The two-input EX-NOR gate gives logic high output when both inputs are either low or high, i.e. when both are equal. When outputs of all the EX-NORs gates are logic high we can say that all bits are matched and hence the desired bit pattern is detected. The final output which indicates that the pattern is detected is taken from four-input AND gate.

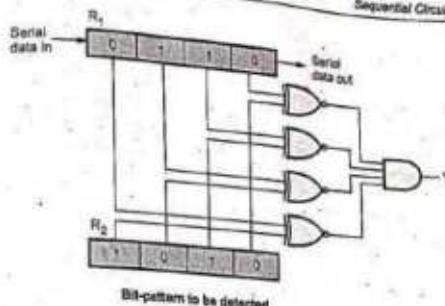


Fig. 5.8.18 4-bit sequence detector

- The 4-bit sequence detector shown in Fig. 5.8.18 can be made programmable by loading the desired 4-bit data in the register  $R_1$ .

**Review Questions**

- Define register.
- What is buffer register?
- Draw and explain 4-bit controlled buffer register.
- List the basic types of shift registers in terms of data movements.
- Draw the logical diagram of a 4-bit shift register. Explain how shift left and shift right operations are performed.
- What is the difference between serial and parallel transfer? What type of registers are used in each case?
- Explain modes of operation of shift register.
- Explain with a neat diagram working of parallel in serial out 4-bit shift register. Draw necessary timing diagram.
- Design a circuit for 4-bits parallel register with load with D flip-flops. Load input decides whether to load new input or to apply no change condition.
- Explain the operation of 4-bit SISO shift register using D flip-flops.
- Draw and explain SISO and PISO for the shift register.
- Draw and explain the circuit diagram of 3-bit register with the following facility  
i) Parallel in serial output ii) Reset.
- Explain how shift registers are used as serial to parallel converter.

GTU : Dec. 13, Marks 1

GTU : May 13, Marks 7

GTU : May 14, Marks 7

GTU : Summer 16, Marks 4

14. Draw and explain 5-bit shift register having parallel in and serial in (left to right) facilities. Explain any one application of such register.  
 15. Explain serial to parallel shift register with neat circuit diagram and timing diagram.  
 16. Explain with the help of neat diagram, the operation of 3-bit bidirectional shift register.  
 17. Draw the circuit and explain the function of 4-bit bidirectional shift register.  
 18. Explain with the help of neat diagram the operation of 4-bit universal shift register.  
 19. Draw and explain 4-bit shift register having shift left, shift right and parallel in facilities like suitable multiplexers in your circuit. Explain any one application of such register.  
 20. Design 4-bit shift register with the following facilities :  
     (i) Shift-left (if  $L1 = 0$  and  $L0 = 0$ )  
     (ii) Shift-right (if  $L1 = 0$  and  $L0 = 1$ )  
     (iii) Parallel-in (if  $L1 = 1$  and  $L0 = 1$ )  
     Control signals 'L1 L0' will select one of the possible operation.  
 21. State applications of shift registers.  
 22. With neat sketch design 4-bit bidirectional shift register. GTU : Summer-15, Marks 7  
 23. Explain serial transfer w.r.t. shift register with suitable example. GTU : Winter-18, Marks 4

**5.9 Counters**

GTU : May-12, 14, Dec-12, 13, Summer-15, 16, 18, Winter-14, 15

- A counter is a register capable of counting the number of clock pulses arriving at its clock input.
- Count represents the number of clock pulses arrived.
- On arrival of each clock pulse, the counter is incremented by one.
- In case of down counter, on arrival of each clock pulse, it is decremented by one.
- The Fig. 5.9.1 shows the logic symbol of a binary counter.

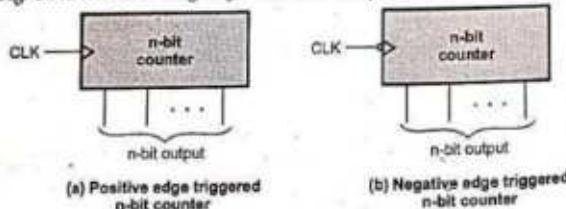


Fig. 5.9.1 Logic symbol of counter

- External clock is applied to the clock input of the counter.
- The counter can be positive edge triggered or negative edge triggered.
- The n-bit binary counter has n flip-flops and it has  $2^n$  distinct states of outputs.

TECHNICAL PUBLICATIONS® - An up thrust for knowledge

- For example, 2-bit counter has 2 flip-flops and it has  $4(2^2)$  distinct states : 00, 01, 10 and 11.
- The 3-bit binary counter has 3 flip-flops and it has  $8(2^3)$  distinct states : 000, 001, 010, 011, 100, 101 110 and 111.
- The maximum count that the binary counter can count is  $2^n-1$ .
- For example, in 2-bit binary counter, the maximum count is  $2^2-1=3$  (11 in binary).
- After reaching the maximum count the counter resets to 0 on arrival of the next clock pulse and it starts counting again.

**Synchronous Counter**

- When counter is clocked such that each flip-flop in the counter is triggered at the same time, the counter is called synchronous counter.

**Asynchronous Counter / Ripple Counter**

- A binary asynchronous/ripple counter consists of a series connection of complementing flip-flops, with the output of each flip-flop connected to the clock input of the next higher-order flip-flop.
- The flip-flop holding the least significant bit receives the incoming clock pulses.
- The Table 5.9.1 shows the comparison between synchronous and asynchronous counters.

St. No.	Asynchronous counters	Synchronous counters
1.	In this type of counter flip-flops are connected in such a way that output of first flip-flop drives the clock for the next flip-flop.	In this type there is no connection between output of first flip-flop and clock input of the next flip-flop.
2.	All the flip-flops are not clocked simultaneously.	All the flip-flops are clocked simultaneously.
3.	Logic circuit is very simple even for more number of states.	Design involves complex logic circuit as number of states increases.
4.	Main drawback of these counters is their low speed as the clock is propagated through number of flip-flops before it reaches last flip-flop.	As clock is simultaneously given to all flip-flops there is no problem of propagation delay. Hence they are high speed counters and are preferred when number of flip-flops increases in the given design.

Table 5.9.1 Synchronous Vs Asynchronous counters

**Modulus of Counter**

- The total number of counts or stable states a counter can indicate is called 'Modulus'.

TECHNICAL PUBLICATIONS® - An up thrust for knowledge

- The modulus of a four-stage counter would be  $16_{10}$ , since it is capable of indicating  $0000_2$  to  $1111_2$ .
- The term 'modulo' is used to describe the count capability of counters.
- For example, mod 6 counter goes through states 0 to 5 and mod 4 counter goes through states 0 - 3.

**Example 5.9.1** Draw the state diagram of MOD-10 counter.

**Solution :**

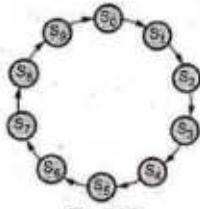


Fig. 5.9.2

**Example 5.9.2** Assume that the 5-bit binary counter starts in the  $00000$  state. What will be the count after 144 input pulses?

**Solution :**  $(144)_{10} = (1001\ 0000)_2$

Since counter is a 5-bit counter, it resets after  $2^5 = 32$  clock pulses.

$$\begin{array}{r} 4 \\ 32 \overline{) 144} \\ - 128 \\ \hline 16 \end{array}$$

Therefore, counter resets four times and then it counts remaining 16 clock pulses. Thus, the count will be  $(10000)_2$ , i.e., 16 in decimal.

In general,

$$\text{Count} = \text{Remainder of } \left[ \frac{\text{Number of input pulses}}{2^n} \right] \text{ in binary}$$

where  $n$  = Number of counter bits

#### 5.9.1 Ripple / Asynchronous Counters

- A binary ripple/asynchronous counter consists of a series connection of complementing flip-flops, with the output of each flip-flop connected to the clock input of the next higher-order flip-flop.

- The flip-flop holding the least significant bit receives the incoming clock pulses.
- A complementing flip-flops can be obtained from a JK flip-flop with the J and K inputs tied together as shown in the Fig. 5.9.3 or from a T flip-flop.
- A third alternative is to use a D flip-flop with the complement output connected to the D input.
- The D input is always the complement of the present state and the next clock pulse will cause the flip-flop to complement.
- Fig. 5.9.3 (a) shows 2-bit asynchronous counter using JK flip-flops.
- The clock signal is connected to the clock input of only first stage flip-flop.
- The clock input of the second stage flip-flop is triggered by the  $Q_A$  output of the first stage.
- Because of the inherent propagation delay time through a flip-flop, a transition of the input clock pulse and a transition of the  $Q_A$  output of first stage can never occur at exactly the same time. Therefore, the two flip-flops are never simultaneously triggered, which results in asynchronous counter operation.

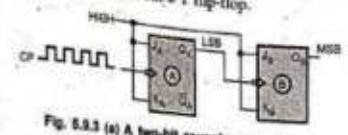


Fig. 5.9.3 (a) A two-bit asynchronous binary counter

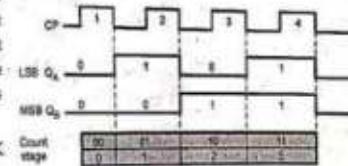


Fig. 5.9.3 (b) Timing diagram for the counter of Fig. 5.9.3 (a)

**Example 5.9.3** Extend the counter shown in Fig. 5.9.3 (a) for 3-stages, and draw its waveform.

**Solution :** In Fig. 5.9.4 (b), timing diagram for 3-bit asynchronous counter we have not considered the propagation delays of flip-flops, for simplicity. If we consider the propagation delays of flip-flops we get timing diagram as shown in Fig. 5.9.5.

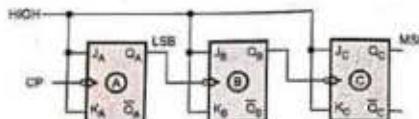


Fig. 5.9.4 (a) Logic diagram

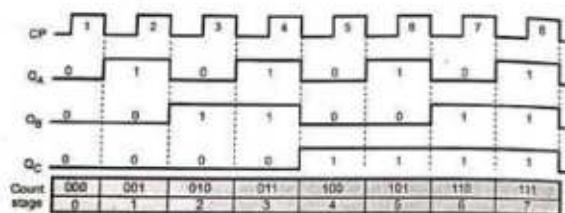
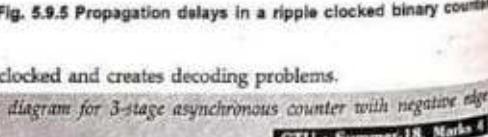


Fig. 5.9.4 (b) Output waveforms for 3-bit asynchronous counter

The timing diagram shows propagation delays. We can see that propagation delay of the first stage is added in the propagation delay of second stage to decide the transition time for third stage. This cumulative delay of an asynchronous counter is a major disadvantage in many applications because it limits the rate at which the counter can be clocked and creates decoding problems.

Fig. 5.9.5 Propagation delays in a ripple clocked binary counter



**Example 5.9.4** Draw the logic diagram for 3-stage asynchronous counter with negative edge triggered flip-flops.

**GTU : Summer-18, Marks 6**

**Solution :** When flip-flops are negatively edge triggered, the Q output of previous stage is connected to the clock input of the next stage. Fig. 5.9.6 shows 3-stage asynchronous counter with negative edge triggered flip-flops.

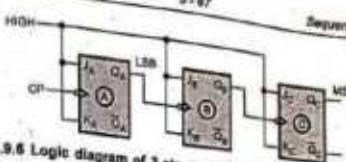


Fig. 5.9.6 Logic diagram of 3-stage negative edge triggered counter

**Example 5.9.5** A counter has 14 stable states 1000 through 1101. If the input frequency is 50 kHz what will be its output frequency?

$$\text{Solution : } \frac{50 \text{ kHz}}{14} = 3.57 \text{ kHz}$$

**Example 5.9.6** The  $t_{PD}$  for each flip-flop is 50 ns, determine the maximum operating frequency for MOD-32 ripple counter.

**Solution :** We know that MOD-32 uses five flip-flops. With  $t_{PD} = 50 \text{ ns}$ , the  $f_{max}$  for ripple counter can be given as,

$$f_{max(\text{ripple})} = \frac{1}{5 \times 50 \text{ ns}} = 4 \text{ MHz}$$

**Example 5.9.7** Draw the logic diagram for 4-stage synchronous counter with positive edge triggered flip-flops. Also draw necessary timing diagram. Is there any frequency division concept in it.

**Solution :** When flip-flops are positive edge triggered, the  $\bar{Q}$  output of previous stage is connected to the clock input of the next stage. Fig. 5.9.7 shows 4-stage asynchronous counter with positive edge triggered flip-flops.

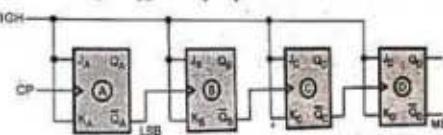


Fig. 5.9.7 Logic diagram of 4-stage positive edge triggered ripple counter

The Fig. 5.9.8 shows the timing diagram for 4-bit ripple up counter using positive edge triggered JK-FFs.

From the timing diagram we can observe that

- Frequency at output  $Q_A = \frac{F_{CLK}}{2}$

- Frequency at output  $Q_B = \frac{Q_A}{2} = \frac{F_{CLK}}{4}$

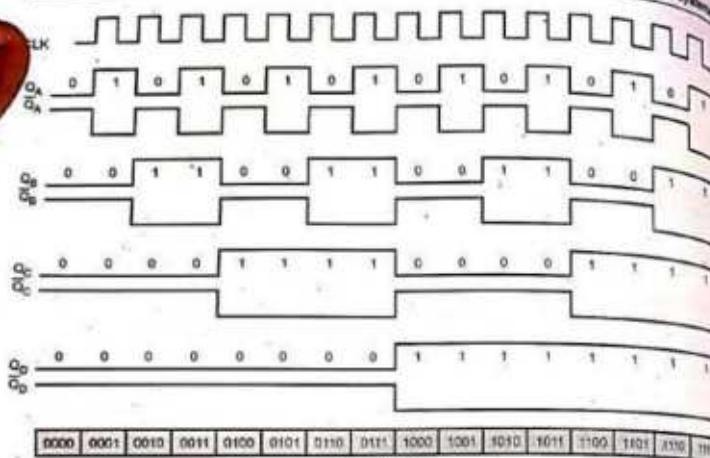


Fig. 5.9.8

- Frequency at output  $Q_C = \frac{Q_B}{2} = \frac{Q_A}{4} = \frac{F_{CLK}}{8}$
- Frequency at output  $Q_D = \frac{Q_C}{2} = \frac{Q_B}{4} = \frac{Q_A}{8} = \frac{F_{CLK}}{16}$

In general we can say that the frequency at the MSB output of counter is  $\frac{F_{CLK}}{2^N}$  where N represents number of stages/bits of the counter.

#### 5.9.1.1 Asynchronous / Ripple Down Counter

- The down counter will count downward from a maximum count to zero.
- The Fig. 5.9.9 shows the 4-bit asynchronous down counter using JK flip-flops.
- The clock signal is connected to the clock input of only first flip-flop.
- The clock input of the remaining flip-flops is triggered by the  $\bar{Q}_A$  output of the previous stage instead of  $Q_A$  output of the previous stage.

TECHNICAL PUBLICATIONS™ - An up thrust for knowledge

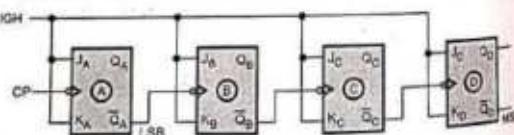


Fig. 5.9.9 4-bit asynchronous down counter

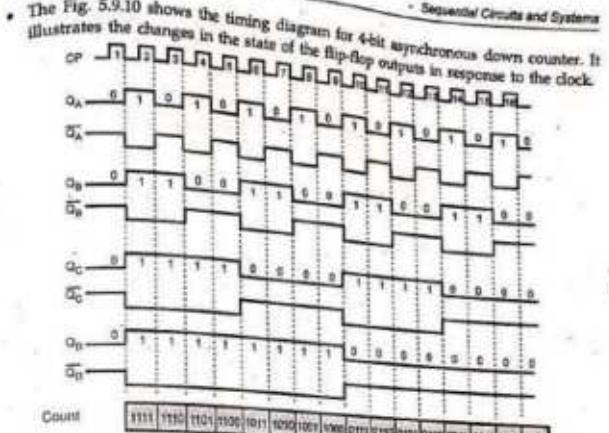


Fig. 5.9.10 Timing diagram of 4-bit asynchronous down counter

- The Fig. 5.9.10 shows the timing diagram for 4-bit asynchronous down counter. It illustrates the changes in the state of the flip-flop outputs in response to the clock.
- The J and K inputs of JK flip-flops are tied to logic HIGH hence output will toggle for each negative edge of the clock input.
- Down counters are not as widely used as up counters. They are used in situations where it must be known when a desired number of input pulses has occurred. In these situations the down counter is preset to the desired number and then allowed to count down as the pulses are applied. When the counter reaches the zero state it is detected by a logic gate whose output then indicates that the preset number of pulses have occurred.

**Example 5.9.8** For the ripple counter shown in Fig. 5.9.11, show the complete timing diagram for eight clock pulses, showing the clock,  $Q_0$  and  $Q_1$  waveforms.

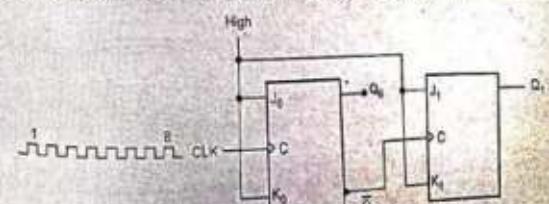
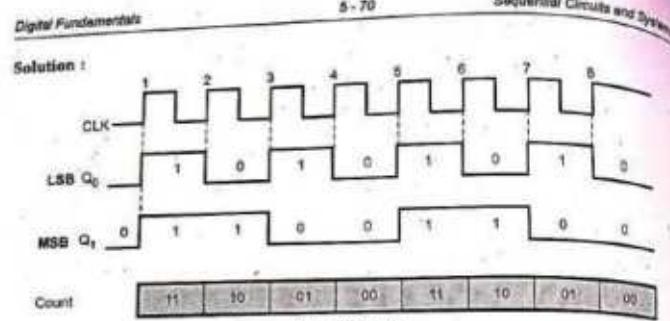


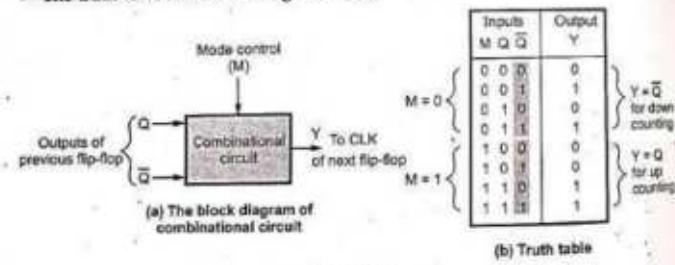
Fig. 5.9.11

TECHNICAL PUBLICATIONS™ - An up thrust for knowledge



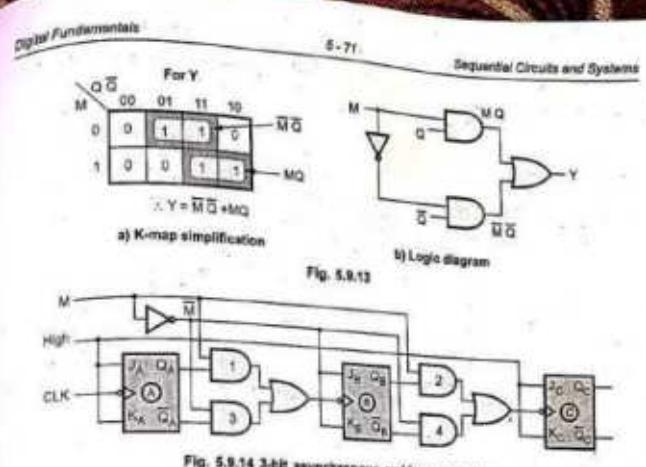
### 5.9.12 Asynchronous Up / Down Counter

- To form an asynchronous up/down counter one control input say M is necessary to control the operation of the up/down counter.
- When M = 0, the counter will count up and when M = 1, the counter will count down. To achieve this the M input should be used to control whether the normal flip-flop output (Q) or the inverted flip-flop output ( $\bar{Q}$ ) is fed to drive the clock signal of the successive stage flip-flop, as shown in Fig. 5.9.12 (a).
- The truth table is shown in Fig. 5.9.12 (b).



- Fig. 5.9.14 shows the 3-bit up/down counter that will count from 000 up to 111 when the mode control input M is 1 and from 111 down to 000 when mode control input M is 0.
- A logic 1 on M enables AND gates 1 and 2 and disables AND gates 3 and 4. This allows the  $Q_A$  and  $Q_B$  outputs to drive the clock inputs of their respective next stages. So that counter will count up.

TECHNICAL PUBLICATIONS™ - An up beat for knowledge

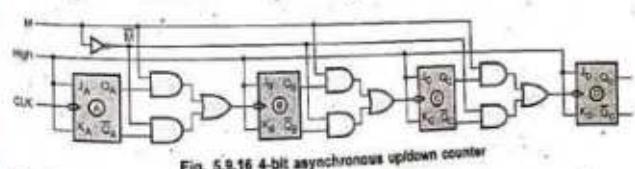


- When M is logic 0, AND gates 1 and 2 are disabled and AND gate 3 and 4 are enabled. This allows the  $Q_A$  and  $Q_B$  outputs to drive the clock inputs of their respective next stages so that counter will count down.
- Fig. 5.9.15 shows the timing diagram for 3-bit up/down ripple counter. (See Fig. 5.9.15 on next page)

### Example 5.9.9 Design a 4-bit updown ripple counter with a control for up/down counting.

**Solution :** The 4-bit counter needs four flip-flops. The circuit for 4-bit up/down ripple counter is similar to 3-bit up/down ripple counter except that 4-bit counter has one more flip-flop and its clock driving circuiting.

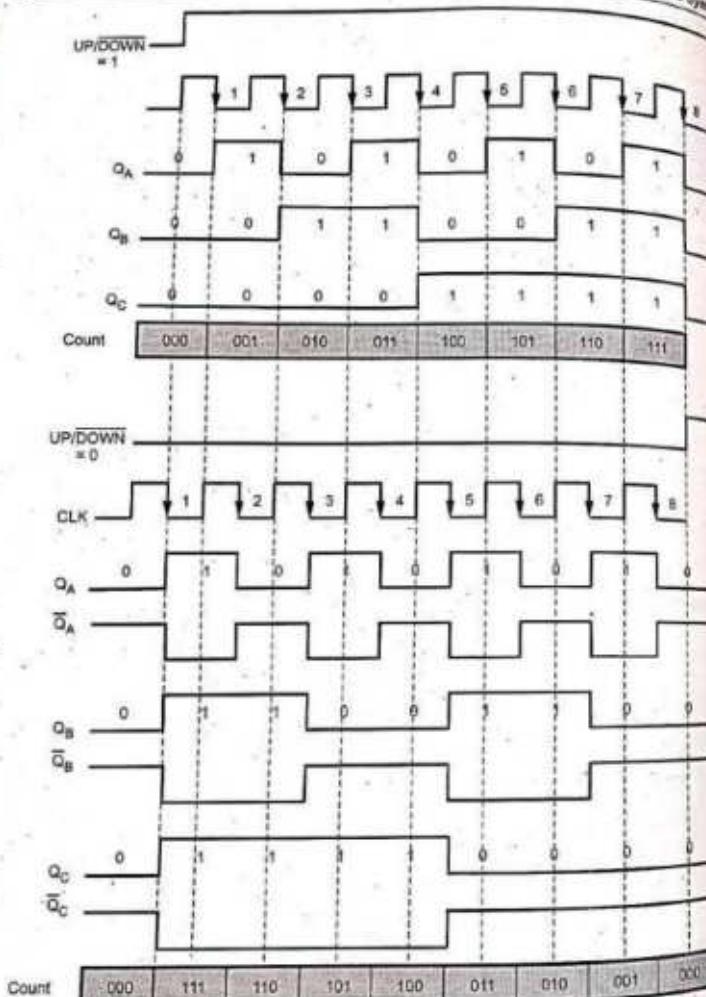
The Fig. 5.9.16 shows the 4-bit up/down ripple counter.



### 5.9.13 Decoding Gates

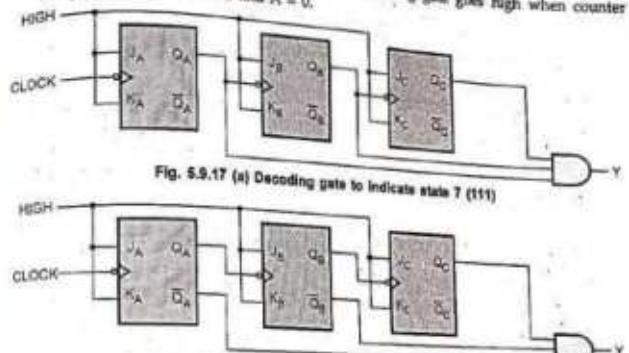
- Decoding gates are used to indicate whether counter has reached to particular state.

TECHNICAL PUBLICATIONS™ - An up beat for knowledge



TECHNICAL PUBLICATIONS™ - An up beat for knowledge

- The outputs of the counter are connected to the AND gate as inputs and the output of the AND gate goes high for particular state.
- As shown in Fig. 5.9.17 (a), the output of decoding gate goes high when counter outputs are  $C = 1$ ,  $B = 1$  and  $A = 1$ .
- As shown in Fig. 5.9.17 (b) the output of decoding gate goes high when counter outputs are  $C = 1$ ,  $B = 0$  and  $A = 0$ .



- We can connect corresponding outputs to decoding gate inputs to indicate desired state.
- The Fig. 5.9.18 gives these connections for all possible state detection for 3-bit counter.

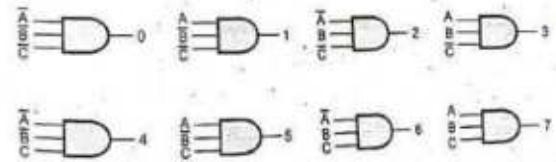


Fig. 5.9.18 Decoding gate connections

**5.9.14 Problem Faced by Ripple Counter (Glitch Problem)**

- Due to the propagation delay the output flip-flop is delayed by time  $t_p$ .
- Fig. 5.9.19 shows the waveform of the circuit which decodes state 7.

TECHNICAL PUBLICATIONS™ - An up beat for knowledge

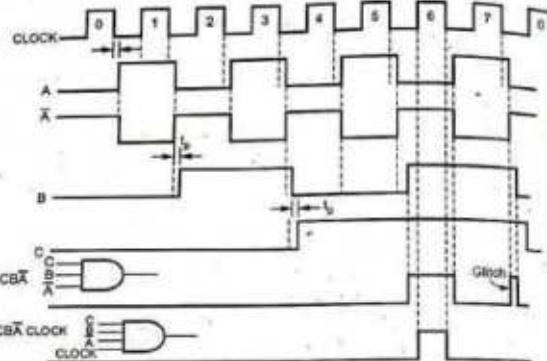


Fig. 5.9.19 Elimination of glitch

- Here, the output of flip-flop A triggers the flip-flop B, hence the B waveform is delayed by one flip-flop delay time ( $t_p$ ) from the negative transition of A.
- The C waveform is delayed by  $t_p$  from each negative transition of B.
- At point X, A goes low ( $\bar{A}$  goes high); however, because of flip-flop delay time, it does not go low until point Y. Thus between points X and Y we have the condition  $C = 1, B = 1$  and  $A = 1$ . As a result, the output is high between points X and Y. This undesirable output is known as glitch.
- We can avoid the glitch on the output waveform by connecting clock as a fourth input to the decoding gate along with inputs A, B and C. This is illustrated in Fig. 5.9.19.

### 5.9.2 Design of Ripple (Asynchronous) Counters

Steps involved in the design of asynchronous counter

- Determine the number of flip-flops needed.
- Choose the type of flip-flops to be used : T or JK. If T flip-flops are used connect input of all flip-flops to logic 1. If JK flip-flops are used connect both J and K inputs of all flip flops to logic 1.  
Such connection toggles the flip-flop output on each clock transition.
- Write the truth table for the counter.
- Derive the reset logic by K-map simplification.
- Draw the logic diagram.

**Example 5.9.10** Design BCD (mod-10) ripple counter using JK flip-flop

GU : Map 14, Marks 7

Solution :

Step 1 : Determine the number of flip-flops needed. The BCD counter goes through states 0-9, i.e. total 10 states. Thus,  $N = 10$  and for  $2^n \geq N$ , we need  $n = 4$ .

Step 2 : Type of flip-flops to be used : JK

Step 3 : Write the truth table for the counter

CLK	A	B	C	D	Output of reset logic Y
0	0	0	0	0	1
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	1
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	1
-	1	0	1	0	0
-	1	0	1	1	0
-	1	1	0	0	0
-	1	1	0	1	0
-	1	1	1	0	0
-	1	1	1	1	0

Table 5.9.2 Truth table for BCD counter

Step 4 : Derive reset logic

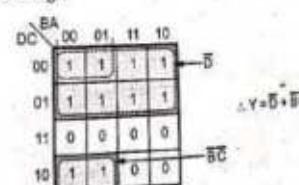


Fig. 5.9.20

Note : The reset input (CLR) of each Flip-Flop is active-low input. By making CLR input of all Flip-Flops logic 0, we can reset the counter. Thus reset logic is designed such a way that for invalid states,  $Y = 0$  and counter resets.

Valid states

Invalid states

**Step 5 :** Draw logic diagram.

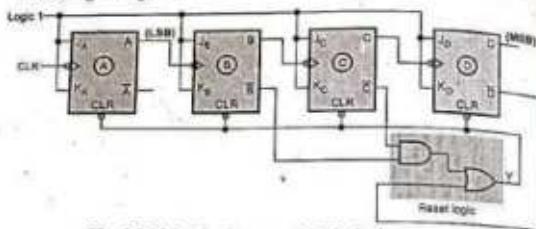


Fig. 5.9.21 Logic diagram of BCD ripple counter

**Example 5.9.11** Design a 3-bit asynchronous ripple counter using T flip-flops and explain its operation.

**Solution :** The Fig. 5.9.22 shows a 3-bit asynchronous ripple counter using T flip-flops. As shown in Fig. 5.9.22, the clock input of only first stage flip-flop. The clock input of the second stage flip-flop is triggered by the  $Q_A$  output of the first stage and the third stage flip-flop is triggered by the  $Q_B$  output of second stage. Because of the inherent propagation delay time through a flip-flop, a transition of the input clock pulse and a transition of the  $Q_A$  output of previous stage can never occur at exactly the same time. Therefore, flip-flops are never simultaneously triggered, which results in asynchronous counter operation.

- Since, T input is connected to logic 1 each flip-flop toggles at clock input. Fig. 5.9.23 shows the timing diagram for 3-bit asynchronous counter.

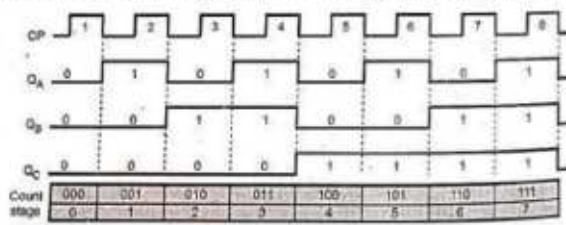


Fig. 5.9.23 Output waveforms for 3-bit asynchronous counter

TECHNICAL PUBLICATIONS™ - An up beat for knowledge

**Example 5.9.12** Design mod 6 ripple counter using T flip-flops.

**Solution :**

**Step 1 :** Determine the number of flip-flop required. Here, counter goes through 0 - 5 i.e. total 6 states. Thus  $N = 6$  and for  $2^n \geq N$  we need  $n = 3$ .

**Step 2 :** Type of flip-flops to be used : T

**Step 3 :** Write the truth table for counter

CLK	C	B	A	Output of reset logic Y
0	0	0	0	1
1	0	0	1	0
2	0	1	0	1
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
-	1	1	0	0
-	1	1	1	0

**Step 4 :** Derive reset logic

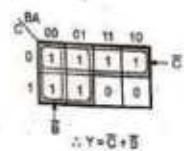


Fig. 5.9.24

Fig. 5.9.25

**Step 5 :** Draw logic diagram

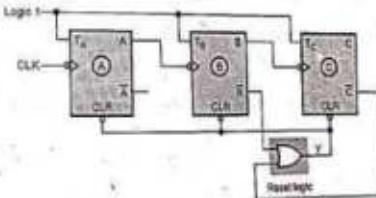


Fig. 5.9.26

**Example 5.9.13** Design ripple counter for state diagram shown.

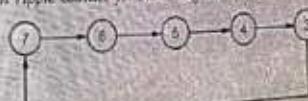


Fig. 5.9.27

TECHNICAL PUBLICATIONS™ - An up beat for knowledge

**Solution :****Step 1 :** Determine the number of flip-flops needed.We know that  $2^n \geq N$ . Here,  $N = 8 \therefore n = 3$  i.e. 3 flip-flops needed.**Step 2 :** Type of flip-flops to be used : JK**Step 3 :** Write truth table for counter.

CLK	C	B	A	Output of reset logic Y
0	1	1	1	1
1	1	1	0	1
2	1	0	1	1
3	1	0	0	1
4	0	1	1	1
5	0	1	0	0
6	0	0	1	0
7	0	0	0	0

Table 5.9.3 Truth table

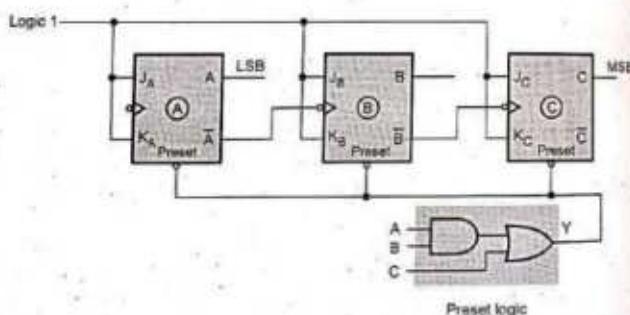
**Step 5 :** Draw logic diagram.

Fig. 5.9.29

- Since it is a down counter, the clock of the next flip-flop is given by the  $\bar{Q}$  output of the previous flip-flop.

**Step 4 :** Derive preset logic. Since it is a down counter we need to derive preset logic instead of set logic.

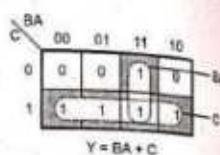


Fig. 5.9.28

**5.9.3 Synchronous Counters**

- When counter is clocked such that each flip-flop in the counter is triggered at the same time, the counter is called as synchronous counter.

**5.9.3.1 2-bit Synchronous Binary Up Counter**

- Fig. 5.9.30 shows two stage synchronous counter.
- Clock signal is connected in parallel to clock inputs of both the flip-flops.
- The  $Q_A$  output of first stage is used to drive the J and K inputs of the second stage.
- Initially, we assume that the  $Q_A = Q_B = 0$ .
- When positive edge of the first clock pulse is applied, flip-flop A will toggle because  $J_A = K_A = 1$ , whereas flip-flop B output will remain zero because  $J_B = K_B = 0$ .
- After first clock pulse  $Q_A = 1$  and  $Q_B = 0$ .
- At negative going edge of the second clock pulse both flip-flops will toggle because they both have a toggle condition on their J and K inputs ( $J_A = K_A = J_B = K_B = 1$ ). Thus after second clock pulse,  $Q_A = 0$  and  $Q_B = 1$ .
- At negative going edge of the third clock pulse flip-flop A toggles making  $Q_A = 1$ , but flip-flop B remains set i.e.  $Q_B = 1$ .
- Finally, at the leading edge of the fourth clock pulse both flip-flops toggle as their JK inputs are at logic 1. This results  $Q_A = Q_B = 0$  and counter recycled back to its original state.
- The timing details of above operation is shown in Fig. 5.9.31.

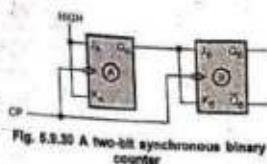


Fig. 5.9.30 A two-bit synchronous binary counter

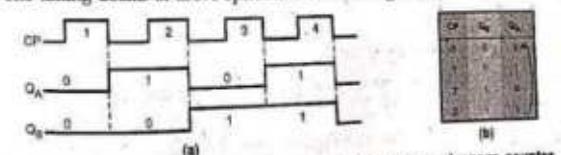


Fig. 5.9.31 Timing diagram and state sequence for the 2-bit synchronous counter

**5.9.3.2 3-bit Synchronous Binary Up Counter**

- Fig. 5.9.32 (a) shows 3-bit synchronous binary counter and its timing diagram.

- The state sequence for this counter is shown in Table 5.9.4.
- Looking at Fig. 5.9.32 (b), we can see that  $Q_A$  changes on each clock pulse as we progress from its original state to its final state and then back to its original state.
- Flip-flop A is held in the toggle mode by connecting J and K inputs to HIGH.
- Flip-flop B toggles, when  $Q_A$  is 1.
- When  $Q_A$  is 0, flip-flop B is in the no-change mode and remains in its present state.
- Looking at the Table 5.9.4 we can notice that flip-flop C has to change its state only when  $Q_B$  and  $Q_A$  both are at logic 1. This condition is detected by AND gate and applied to the J and K inputs of flip-flop C. Whenever both  $Q_A$  and  $Q_B$  are HIGH, the output of the AND gate makes the J and K inputs of flip-flop C HIGH and flip-flop C toggles on the following clock pulse. At all other times, the J and K inputs of flip-flop C are held LOW by the AND gate output and flip-flop does not change state.

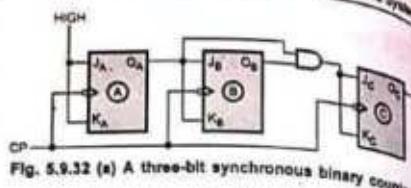


Fig. 5.9.32 (a) A three-bit synchronous binary counter

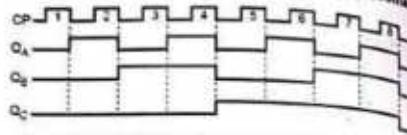


Fig. 5.9.32 (b) Timing diagram for 3-bit synchronous binary counter

CP	$Q_C$	$Q_B$	$Q_A$
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

Table 5.9.4 State sequence for 3-bit binary counter

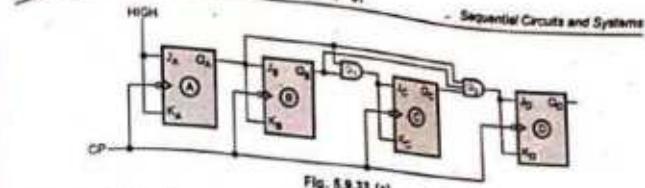


Fig. 5.9.33 (a)

- For the fourth stage, flip-flop has to change the state when  $Q_A = Q_B = Q_C = 1$ . This condition is decoded by 3-input AND gate  $G_2$ . Therefore, when  $Q_A = Q_B = Q_C = 1$ , flip-flop D toggles and for all other times it is in no change condition.

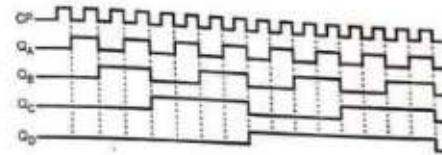


Fig. 5.9.33 (b) A four-bit synchronous binary counter and timing diagram

**Example 5.9.14** Determine  $f_{max}$  for the 4-bit synchronous counter if  $t_{pd}$  for each flip-flop is 50 ns and  $t_{pd}$  for each AND gate is 20 ns. Compare this with  $f_{max}$  for a MOD-16 ripple counter.

**Solution :** For a synchronous counter the total delay that must be allowed between input clock pulses is equal to flip-flop  $t_{pd}$  + AND gate  $t_{pd}$ . Thus  $T_{clock} \geq 50 + 20 = 70$  ns and so the counter has

$$f_{max} = \frac{1}{70\text{ ns}} = 14.3 \text{ MHz}$$

We know that MOD-16 ripple counter used four flip-flops. With flip-flop  $t_{pd} = 50$  ns, the  $f_{max}$  for ripple counter can be given as,

$$f_{max(\text{ripple})} = \frac{1}{4 \times 50\text{ ns}} = 5 \text{ MHz}$$

#### 5.9.3.4 Synchronous Down and Up / Down Counters

- A parallel/synchronous down counter can be constructed by using the inverted FF outputs to drive the following JK inputs.

- For example, the parallel up counter of Fig. 5.9.33 (a) can be converted to a down counter by connecting the  $\bar{Q}_A$ ,  $\bar{Q}_B$ ,  $\bar{Q}_C$  and  $\bar{Q}_D$  outputs in place of  $Q_A$ ,  $Q_B$ ,  $Q_C$  and  $Q_D$  respectively.
- The counter will then proceed through the following sequence as input pulses are applied :
- To form a parallel up/down counter the control input (UP/DOWN) is used to control whether the normal flip-flop outputs or the inverted flip-flop outputs are fed to the J and K inputs of the following flip-flops.
- A logic 1 on the Up/Down enables AND gates 1 and 2 and disables AND gates 3 and 4. This allows the  $Q_A$  and  $Q_B$  outputs through to the J and K inputs of the next flip-flops so that the counter will count up as pulses are applied.

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

Fig. 5.9.34

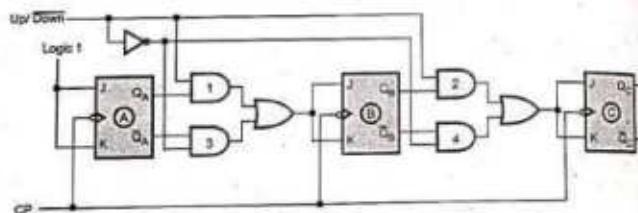


Fig. 5.9.35 3-bit synchronous / parallel up / down counter

- When Up/Down line is logic 0, AND gates 1 and 2 are disabled and AND gates 3 and 4 are enabled. This allows the  $\bar{Q}_A$  and  $\bar{Q}_B$  outputs through to the J and K inputs of the next flip-flops so that the counter will count down as pulses are applied.
- Fig. 5.9.36 shows the timing diagram for 3-bit up-down counter.

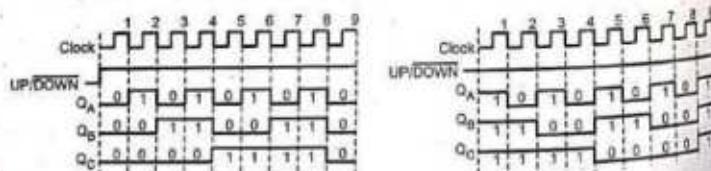


Fig. 5.9.36 Timing diagram

**5.9.4 Design of Synchronous Counters**

- Determine the number of flip-flops needed. If  $n$  represents number of flip-flops  $2^n \geq N$  where  $N$  is number of states in the counter.
- Choose the type of flip-flops to be used.
- Using excitation table for selected flip-flop determine the excitation table for the counter.
- Use K-map or any other simplification method to derive the flip-flop input functions.
- Draw the logic diagram.

**Example 5.9.15** Design a MOD-5 synchronous counter using JK flip-flops and implement it. Also draw the timing diagram.

**Solution :**

**Step 1 :** Determine the number of flip-flop needed

Flip-flops required are

$$2^n \geq N$$

Here  $N = 5 \therefore n = 3$  i.e. three flip-flops are required.

**Step 2 :** Type of flip-flop to be used : JK

**Step 3 :** Determine the excitation table for the counter.

$Q_2$	$Q_{2+1}$	J	K
0	0	1	X
0	1	1	X
1	0	X	1

Table 5.9.5 Excitation table for JK flip-flop

Present state	Next state	Flip-flop inputs							
		$Q_2$	$Q_2+1$	$Q_{2+1}$	$Q_{2+1}$	$J_2$	$K_2$	$J_1$	$K_1$
0 0 0	0 0 0	0	0	0	0	1	0 X 0 X 1 X		
0 0 1	0 0 1	0	0	1	0	0	0 X 1 X X 1		
0 1 0	0 1 0	0	1	0	1	1	0 X X 0 1 X		
0 1 1	0 1 1	1	1	1	0	0	1 X X 1 X 1		

1	0	0	0	0	0	X	1	0	X	0	X
1	0	1	X	X	X	X	X	X	X	X	X
1	1	0	X	X	X	X	X	X	X	X	X
1	1	1	X	X	X	X	X	X	X	X	X

Table 5.9.6 Excitation table for counter.

Step 4 : K-map simplification

		For $J_C$				For $K_C$				For $J_B$				For $K_B$					
		Q <sub>C</sub>	Q <sub>A</sub>	00	01	11	10	Q <sub>C</sub>	Q <sub>A</sub>	00	01	11	10	Q <sub>C</sub>	Q <sub>A</sub>	00	01	11	10
0	0	0	0	1	0			0	X	X	X	X	0	0	1	X	X	X	
1	X	X	X	X	X			1	1	X	X	X	1	0	X	X	X	X	

$J_C = Q_B Q_A$

$K_C = 1$

$J_B = Q_A$

		For $K_B$				For $J_A$				For $K_A$				For $J_A$				For $K_A$			
		Q <sub>C</sub>	Q <sub>A</sub>	00	01	11	10	Q <sub>C</sub>	Q <sub>A</sub>	00	01	11	10	Q <sub>C</sub>	Q <sub>A</sub>	00	01	11	10		
0	X	X	1	0				0	1	X	X	X		0	X	1	1	X			
1	X	X	X	X	X			1	0	X	X	X		1	X	X	X	X			

$K_B = Q_A$

$J_A = \bar{Q}_C$

$K_A = 1$

Fig. 5.9.37

Step 5 : Draw the logic diagram

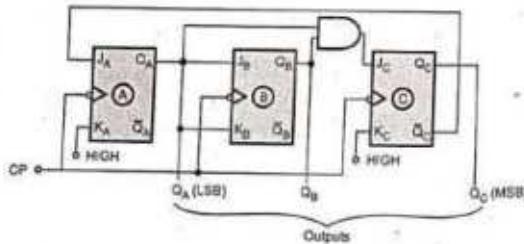


Fig. 5.9.37 (a) MOD-5 synchronous counter

## Timing diagram

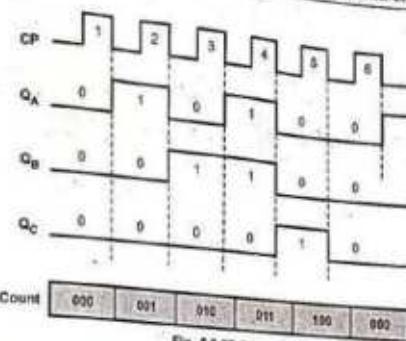


Fig. 5.9.37 (b)

**Example 5.9.16** Design divide by 6 counter using T-flip-flops. Write state table and reduce the expression using K-map.

Solution : Step 1 : Determine the number of flip-flops needed.

For designing mod 6 counter using the formula

$$2^n \geq N$$

Here  $N = 6 \therefore n = 3$  i.e. 3 flip-flops are required.

Step 2 : Type of flip-flops to be used : T

Step 3 : Determine the excitation table for counter.

Q <sub>n</sub>	Q <sub>n+1</sub>	T	CP	Q <sub>C</sub>	Q <sub>B</sub>	Q <sub>A</sub>	Q <sub>n+1</sub>	Q <sub>B+1</sub>	Q <sub>A+1</sub>	T <sub>C</sub>	T <sub>B</sub>	T <sub>A</sub>	
0	0	0	0	0	0	0	0	1	0	0	0	0	1
1	0	1	1	0	0	0	1	0	1	0	0	1	1
0	1	1	2	0	1	0	1	1	1	0	1	0	2
1	0	1	3	0	1	1	0	0	1	1	0	1	1
0	1	0	4	1	0	0	0	1	0	1	0	0	1
1	1	0	5	1	0	1	0	0	0	1	0	1	0

Table 5.9.7 Excitation table for T-flip-flop

Table 5.9.8 Excitation table for counter

**Step 4 : K-map simplification.**

		For $T_C$				For $T_B$				For $T_A$						
		Q <sub>B</sub> Q <sub>A</sub>	00	01	11	10	Q <sub>B</sub> Q <sub>A</sub>	00	01	11	10	Q <sub>B</sub> Q <sub>A</sub>	00	01	11	10
Q <sub>B</sub>	Q <sub>A</sub>	0 0	0 1	1 1	1 0	0 0	0 1	1 1	1 0	0 1	1 1	0 0	0 1	1 1	1 0	
0	0	0 0	0 1	X	X	0	0	1 1	1 0	0 1	1 1	0 0	0 1	1 1	1 0	
1	0	1 0	X	X	X	1	0	0 X	X	1 1	X	1 0	0	1	X	X

$$T_C = Q_B Q_A + Q_B Q_A$$

		For $T_B$				
		Q <sub>B</sub> Q <sub>A</sub>	00	01	11	10
Q <sub>B</sub>	Q <sub>A</sub>	0 0	0 1	1 1	1 0	
0	0	0 0	0 1	1 1	1 0	
1	0	0 1	X	X	X	

$$T_B = \bar{Q}_B Q_A$$

		For $T_A$				
		Q <sub>B</sub> Q <sub>A</sub>	00	01	11	10
Q <sub>B</sub>	Q <sub>A</sub>	0 0	0 1	1 1	1 0	
0	0	0 1	1 1	X	X	
1	0	1 1	X	X	X	

$$T_A = 1$$

**Step 5 : Draw the logic diagram.**

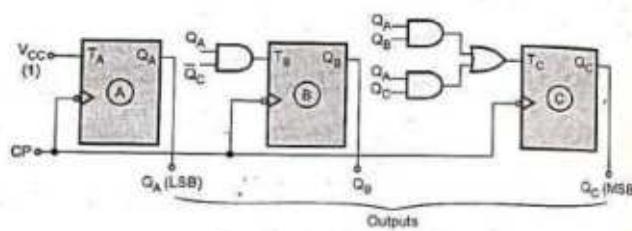


Fig. 5.9.38 (a) Logic diagram

**Example 5.9.17** Design a synchronous decade counter using D flip-flop.

Solution : The decade counter is a mod-10 counter. It has ten states : 0 - 9.

**Step 1 :** Determine the number of flip-flops needed.

We know that  $2^n \geq N$ . Here,  $N = 10 \therefore n = 4$  i.e. 4 flip-flops needed.

**Step 2 :** Types of flip-flops to be used : D

**Step 3 :** Determine the excitation table for counter.

Present state				Next state			
Q <sub>D</sub>	Q <sub>C</sub>	Q <sub>B</sub>	Q <sub>A</sub>	Q <sub>D+1</sub>	Q <sub>C+1</sub>	Q <sub>B+1</sub>	Q <sub>A+1</sub>
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	0	1	1

**Step 4 : K-map simplification**

		For $Q_{D+1}$				
		Q <sub>B</sub> Q <sub>A</sub>	00	01	11	10
Q <sub>B</sub>	Q <sub>A</sub>	0 0	0 1	0 0	0 0	
0	0	0 0	0 1	0 0	0 0	
1	0	0 1	X	X	X	

$$Q_{D+1} = Q_B \bar{Q}_A + Q_B Q_A$$

		For $Q_{C+1}$				
		Q <sub>B</sub> Q <sub>A</sub>	00	01	11	10
Q <sub>B</sub>	Q <sub>A</sub>	0 0	0 1	1 1	1 0	
0	0	0 0	0 1	1 1	1 0	
1	0	0 1	X	X	X	

$$Q_{C+1} = Q_B \bar{Q}_A + Q_B Q_A + \bar{Q}_C Q_A$$

		For $Q_{B+1}$				
		Q <sub>B</sub> Q <sub>A</sub>	00	01	11	10
Q <sub>B</sub>	Q <sub>A</sub>	0 0	0 1	0 0	1 1	
0	0	0 0	0 1	0 0	1 1	
1	0	0 1	X	X	X	

$$Q_{B+1} = \bar{Q}_B \bar{Q}_A Q_A + Q_B \bar{Q}_A$$

		For $Q_{A+1}$				
		Q <sub>B</sub> Q <sub>A</sub>	00	01	11	10
Q <sub>B</sub>	Q <sub>A</sub>	0 0	0 1	0 0	0 1	
0	0	0 0	0 1	0 0	0 1	
1	0	0 1	X	X	X	

$$Q_{A+1} = \bar{Q}_A$$

**Step 5 :** Draw the logic diagram.

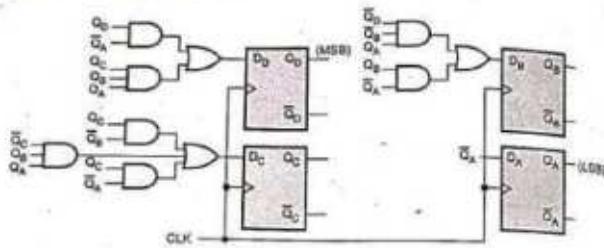


Fig. 5.9.39 (a) Logic diagram

**Example 5.9.18** Design a BCD up/down counter using SR flip-flops.

**Solution :**

**Step 1 :** Number of flip-flops needed = 4

**Step 2 :** Flip-flops to be used = SR

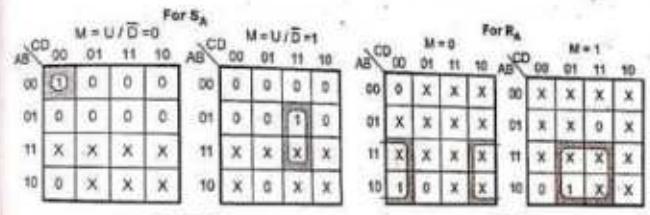
**Step 3 :** Excitation table for counter

UP	Present state	Next state	Flip-flop inputs													
U/D	A	B	C	D	A+	B+	C+	D+	S <sub>A</sub>	R <sub>A</sub>	S <sub>B</sub>	R <sub>B</sub>	S <sub>C</sub>	R <sub>C</sub>	S <sub>D</sub>	R <sub>D</sub>
0	0	0	0	0	1	0	0	1	1	0	0	x	0	x	1	0
0	0	0	0	1	0	0	0	0	0	x	0	x	0	1	0	1
0	0	0	1	0	0	0	0	1	0	1	0	x	0	1	1	0
0	0	0	1	1	0	0	1	0	0	1	0	x	1	0	0	1
0	0	1	0	0	0	0	1	1	0	1	0	1	1	0	1	0
0	0	1	0	1	0	1	0	0	0	1	1	0	0	1	0	1
0	0	1	1	0	0	1	0	1	0	1	1	1	0	0	1	0
0	0	1	1	1	0	1	1	0	0	1	1	1	0	0	1	0
0	1	0	0	0	0	1	1	1	0	1	1	0	1	0	1	0
0	1	0	0	1	1	0	0	0	1	1	1	0	1	0	1	0
0	1	0	0	1	1	0	0	1	1	0	1	0	1	0	1	0

AB	CD	M = U / $\bar{D}$ = 0	AB	CD	M = U / $\bar{D}$ = 1	AB	CD	M = 0	AB	CD	M = 1
00	00	0 0 0 0	00	00	0 0 0 0	00	00	0 0 0 0	00	00	0 0 0 0
01	01	0 0 0 0	01	01	0 0 0 0	01	01	0 0 0 0	01	01	0 0 0 0
11	11	X X X X	11	X X X X	X X X X	11	X X X X	X X X X	11	X X X X	X X X X
10	10	0 X X X	10	X 0 X X	0 X X X	10	X 0 X X	0 X X X	10	0 1 X X	0 X X X

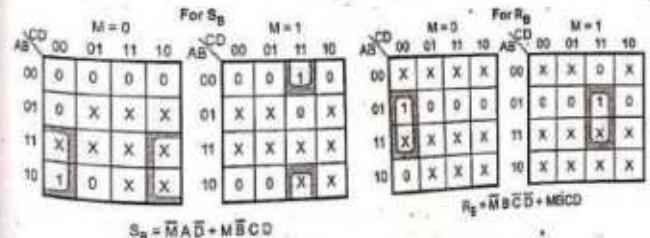
Table 5.9.10

**Step 4 : K-map simplification**



$$S_A = \overline{M} \overline{A} \overline{B} \overline{C} \overline{D} + MBCD$$

$$R_A = \overline{M} \overline{A} \overline{D} + MAD$$



$$S_B = MAD + M\overline{B}CD$$

$$R_B = \overline{M} B \overline{C} \overline{D} + M\overline{C}D$$

For $S_C$				For $R_C$			
M = 0				M = 1			
AB <sup>CD</sup> 00 01 11 10							
00 0 0 X 0	00 0 1 0 X	01 1 0 X 0	01 0 1 0 X	00 X X 0 1	00 0 1 0 1	01 0 X 0 1	01 0 0 1 0
01 1 0 X 0	01 0 1 0 X	11 X X X X	11 X X X X	01 X X X X	01 X X X X	11 X X X X	10 X X X X
11 X X X X	10 0 0 X X	10 0 0 X X	10 0 0 X X	10 0 X X X	10 0 X X X	10 X X X X	10 X X X X
10 1 0 X X							

$S_C = \bar{M}B\bar{C}\bar{D} + M\bar{A}\bar{C}\bar{D}$

For $S_D$				For $R_D$			
M = 0				M = 1			
AB <sup>CD</sup> 00 01 11 10							
00 1 0 0 1	00 1 0 0 1	01 1 0 0 1	01 1 0 0 1	00 0 1 1 0	00 0 1 1 0	01 0 1 1 0	01 0 1 1 0
01 1 0 0 1	01 1 0 0 1	X X X X	X X X X	11 X X X X	11 X X X X	11 X X X X	10 X X X X
X X X X	10 1 0 X X	10 1 0 X X	10 1 0 X X	10 1 X X X	10 1 X X X	10 X X X X	10 X X X X
10 1 0 X X							

$S_D = \bar{D}$

$R_D = D$

Fig. 5.9.40

Step 5 : Logic diagram

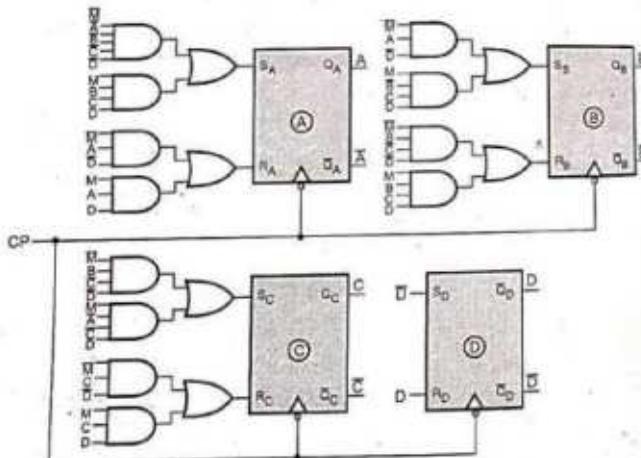


Fig. 5.9.41

Example 5.9.19 Design a mod-12 synchronous up counter using D flip-flop.

GIU : Winter-14, Marks 2

Solution : Mod-12 synchronous counter using D flip-flop :

Number of flip-flop required = n

$$2^n \geq 12$$

$$n = 4$$

Let,

Excitation table

Q <sub>D</sub>	Q <sub>C</sub>	Q <sub>B</sub>	Present state				Next state			
			Q <sub>A</sub>	Q <sub>D1</sub>	Q <sub>C1</sub>	Q <sub>B1</sub>	Q <sub>A1</sub>	Q <sub>D2</sub>	Q <sub>C2</sub>	Q <sub>B2</sub>
0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1	0	0	1
0	0	1	0	0	0	0	1	0	1	0
0	0	1	1	0	0	0	1	0	1	1
0	1	0	0	0	0	0	0	1	0	0
0	1	0	1	0	0	0	1	0	1	0
0	1	1	1	0	0	0	1	1	1	0
1	0	0	0	0	0	0	1	0	0	1
1	0	0	1	1	0	0	1	1	0	1
1	0	1	0	1	1	0	0	1	0	1
1	0	1	1	1	1	0	0	0	0	0
1	1	0	0	0	0	1	0	0	1	0
1	1	0	1	0	1	0	1	0	1	1
1	1	1	1	1	1	0	0	0	0	0

K-map simplification

For $D_A$				For $D_B$			
$Q_0 Q_1$	$Q_0 Q_2$	$Q_0 Q_3$	$Q_1 Q_2$	$Q_1 Q_3$	$Q_2 Q_3$	$Q_0 Q_1 Q_2$	$Q_0 Q_1 Q_3$
00 1 0 0 1	00 1 0 0 1	01 1 0 0 1	01 1 0 0 1	00 0 1 1 0	00 0 1 1 0	01 0 1 1 0	01 0 1 1 0
01 1 0 0 1	01 1 0 0 1	X X X X	X X X X	11 X X X X	11 X X X X	11 X X X X	10 X X X X
X X X X	10 1 0 X X	10 1 0 X X	10 1 0 X X	10 1 X X X	10 1 X X X	10 X X X X	10 X X X X
10 1 0 X X							

$D_A = \bar{Q}_A$

$D_B = \bar{Q}_B + \bar{Q}_A Q_B$

$= Q_1 \oplus Q_2$

For $D_C$			
$Q_0 Q_1$	00	01	11
00	0	0	1
01	1	1	0
11	X	X	X
10	0	0	0

$$D_C = Q_0 \bar{Q}_B + Q_0 \bar{Q}_A + \bar{Q}_0 \bar{Q}_C Q_B Q_A$$

For $D_D$			
$Q_0 Q_1$	00	01	11
00	0	0	0
01	0	0	1
11	X	X	X
10	1	1	0

$$D_D = Q_0 \bar{Q}_B + Q_0 Q_B Q_A + Q_0 \bar{Q}_A$$

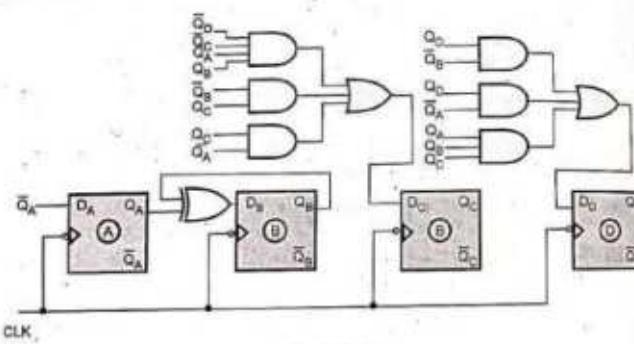
**Logic diagram**

Fig. 5.9.42

**Example 5.9.20** Design a synchronous BCD counter with JK flip-flops.

GTU : Summer-16, Marks 7

**Solution :** The mod-10 counter has ten states : 0 to 9.

**Step 1 :** Determine the number of flip-flops needed.

We know that  $2^n \geq N$ . Here  $N = 10 \therefore n = 4$  i.e. 4 flip-flops needed.

**Step 2 :** Types of flip-flops to be used : JK.

**Step 3 :** Determine the excitation table for counter.

Present state		Next state								Flip-flop inputs					
$Q_0$	$Q_1$	$Q_2$	$Q_3$	$Q_{4+1}$	$Q_{5+1}$	$Q_{6+1}$	$J_D$	$K_D$	$J_E$	$K_E$	$J_F$	$K_F$	$J_G$	$K_G$	
0	0	0	0	0	0	0	1	0	X	0	X	0	X	1	X
0	0	0	1	0	0	1	0	0	X	0	X	1	X	X	1
0	0	1	0	0	0	1	1	1	0	X	0	X	0	1	X
0	0	1	1	0	1	0	0	0	X	1	X	1	X	1	X
0	1	0	0	0	1	0	0	1	0	X	1	X	1	X	1
0	1	0	1	0	1	1	0	0	X	0	0	X	1	1	X
0	1	1	0	0	1	1	1	0	X	0	1	X	X	1	1
0	1	1	1	1	0	0	0	1	X	0	0	X	0	1	X
1	0	0	0	1	0	0	1	X	0	0	1	X	1	X	1
1	0	0	1	0	0	0	0	0	X	1	0	X	1	0	X

Excitation table for counter

**Step 4 : K-map simplification**

For $J_B$			
$Q_0 Q_1$	00	01	11
00	0	0	0
01	0	1	0
11	X	X	X
10	0	0	X

$$J_B = Q_A Q_B Q_C$$

For $J_E$			
$Q_0 Q_1$	00	01	11
00	0	0	0
01	0	1	0
11	X	X	X
10	0	0	X

$$J_E = Q_A Q_B$$

For $J_D$			
$Q_0 Q_1$	00	01	11
00	0	0	1
01	X	X	X
11	X	X	X
10	0	0	X

$$J_D = Q_A Q_B Q_C$$

For $J_G$			
$Q_0 Q_1$	00	01	11
00	0	1	X
01	0	1	X
11	X	X	X
10	0	0	X

$$J_G = Q_A \bar{Q}_B$$

For $J_F$			
$Q_0 Q_1$	00	01	11
00	0	0	0
01	0	1	0
11	X	X	X
10	0	0	X

$$J_F = Q_A Q_B$$

For $J_C$			
$Q_0 Q_1$	00	01	11
00	0	0	0
01	0	1	0
11	X	X	X
10	0	0	X

$$J_C = Q_A$$

		For $J_A = 1$			
$Q_0 Q_1$	$Q_2$	00	01	11	10
00	1	x	x	1	
01	1	x	x	1	
11	x	x	x	x	
10	1	x	x	x	x

		For $K_A = 1$			
$Q_0 Q_1$	$Q_2$	00	01	11	10
00	x	1	1	x	
01	x	1	1	x	
11	x	x	x	x	
10	x	1	x	x	x

Step 5 : Logic diagram

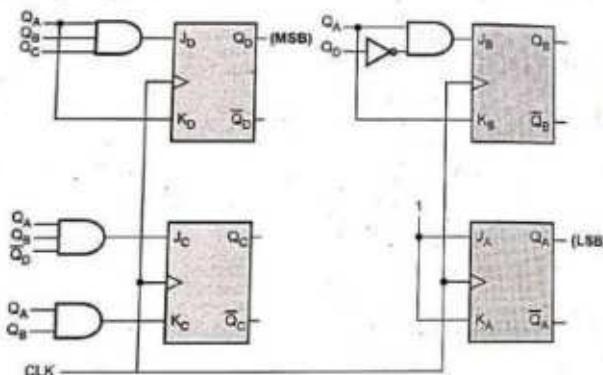


Fig. 5.9.43

**Examples for Practice****Example 5.9.21 :** Design a 3 bit synchronous counter using T flip-flop.**Example 5.9.22 :** The following sequence is to be realized by a counter consisting of 3 JK FF's.

A <sub>1</sub>	0	0	0	0	1	1	0
A <sub>2</sub>	0	1	1	0	0	1	0
A <sub>3</sub>	0	1	0	1	1	0	0

Design the counter.

**Example 5.9.23 :** Design MOD-2 synchronous counter.

- Example 5.9.24 :** Design 3-bit up/down synchronous counter with directional mode control.
- Example 5.9.25 :** Design MOD-5 synchronous counter using T-flip flops.
- Example 5.9.26 :** Design 4-bit synchronous counter and explain its operation with truth table.
- Example 5.9.27 :** Design MOD-10 synchronous counter using J-K flip-flop and explain its operation.

**5.9.5 Presettable Counters**

- Presettable Counters can be preset to any desired starting count either asynchronously (independent of the clock signal) or synchronously (on the active transition of the clock signal).
- This presetting operation is also referred to as parallel loading the counter.
- Fig. 5.9.44 shows the logic circuit for a three-bit presettable parallel up counter.
- The J, K and CLK inputs are wired for operation as a parallel up counter. The asynchronous PRESET and CLEAR inputs are wired to perform asynchronous presetting.

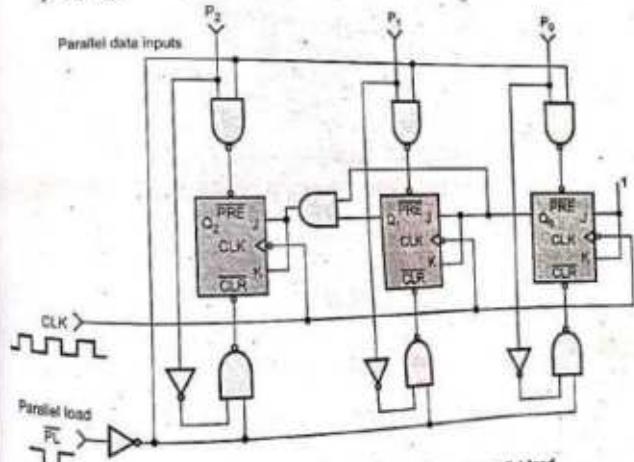


Fig. 5.9.44 Synchronous counter with asynchronous parallel load

- The counter is loaded with any desired count at any time by performing following steps :
  - Apply the desired count to the parallel data inputs,  $P_2$ ,  $P_1$  and  $P_0$ .
  - Apply a LOW pulse to the PARALLEL LOAD input,  $\overline{PL}$ .
- The effect of the CLK input will be disabled as long as  $\overline{PL}$  is in its active-LOW state because each flip-flop will have one of its asynchronous inputs activated while  $\overline{PL} = 0$ .
- Once  $\overline{PL}$  returns HIGH, the flip-flops can respond to their CLK inputs and can resume the counting-up operation starting from the count that was loaded into the counter.
- For example, let's say that  $P_2 = 0$ ,  $P_1 = 1$  and  $P_0 = 0$ . While  $\overline{PL}$  is HIGH, these parallel data inputs have no effect.
- If clock pulses are present, the counter will perform the normal count-up operation.
- Now let's say that  $\overline{PL}$  is pulsed LOW when the counter is at the 100 count (i.e.  $Q_2 = 1$ ,  $Q_1 = 0$  and  $Q_0 = 0$ ). This LOW at  $\overline{PL}$  will produce LOWs at the PRE inputs of  $Q_1$  and at the CLR inputs of  $Q_2$  and  $Q_0$  so that the counter will go to the 010 count regardless of what is occurring at the CLK input. The count will hold at 010 until  $\overline{PL}$  is deactivated (returned HIGH); at that time the counter will resume counting up at each clock pulse from the count of 010.
- The asynchronous presetting is used by several counter ICs such as TTL 74ALS190, 74ALS191, 74ALS192 and 74ALS193 and the CMOS equivalents, 74HC190, 74HC191, 74CH192 and 74HC193.

#### Synchronous Presetting

- Some parallel counter ICs use synchronous presetting whereby the counter is preset on the active transition of the same clock signal that is used for counting.
- The logic level on the parallel load control input determines if the counter is preset with the applied input data at the next active clock transition.
- Examples of IC counters that use synchronous presetting include the TTL 74ALS160, 74ALS161, 74ALS162 and 74ALS163 and their CMOS equivalents, 74HC160, 74HC161, 74HC162 and 74HC163.

#### 5.9.6 Design of Skipping State Counter

- The skipping state counter is a counter in which some states are missing in the sequence. For example, if counter goes through states : 0, 2, 4, 6 and 0 we can say that states 1, 3, 5 and 7 are missing, i.e. counter skips states 1, 3, 5 and 7. The skipping states are also known as unused states.

- In a skipping state counter, if the next state of some unused state is again an unused state and if by chance the counter happens to find itself in the unused states and never arrived at a used state then the counter is said to be in the lockout conditions. This is illustrated in Fig. 5.9.45.
- The counter which never goes in lockout condition is called self starting counter.
  - To make sure that the counter will come to the initial state from any unused state, the additional logic circuit is necessary.
  - To ensure that the lock-out does not occur, the counter should be designed by forcing the next state to be the initial state from the unused states as shown in Fig. 5.9.46.
  - For example, as shown in Fig. 5.9.46, actually it is not necessary to force all unused states into initial state. Forcing any one state is sufficient. Because if counter initially goes to unused state which is not forced, it will go to another unused state. This will continue until it reaches the forced unused state. Once forced unused state is reached next state is used state, and circuit is lock free circuit. This is illustrated in Fig. 5.9.46 (b).

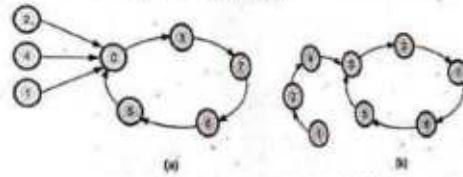


Fig. 5.9.46 State diagram for removing lockout

**Example 5.9.28** Design a synchronous counter for $4 \rightarrow 6 \rightarrow 7 \rightarrow 3 \rightarrow 1 \rightarrow 4$ 

Avoid lockout condition. Use JK type design

**Solution :****Step 1 :** State diagram

Here, states 5, 2 and 0 are forced to go into 6, 3 and 1 state, respectively to avoid lockout condition.

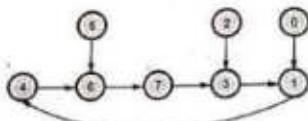


Fig. 5.9.47

**Step 2 : Excitation table**

Present states			Next states			Flip-flop inputs					
A	B	C	$A_{+1}$	$B_{+1}$	$C_{+1}$	$J_A$	$K_A$	$J_B$	$K_B$	$J_C$	$K_C$
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	1	0	0	1	X	0	X	X	1
0	1	0	0	1	1	0	X	X	0	1	X
0	1	1	0	0	1	0	X	X	1	X	0
1	0	0	1	1	0	X	0	1	X	0	X
1	0	1	1	1	0	X	0	1	X	X	1
1	1	0	1	1	1	X	0	X	0	1	X
1	1	1	0	1	1	X	1	X	0	X	0

Table 5.9.11

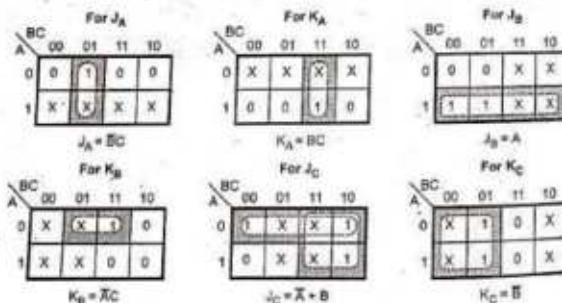
**Step 3 : K-map simplification**

Fig. 5.9.48

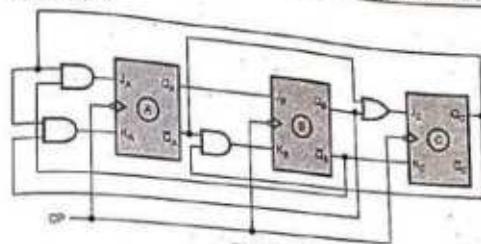
**Step 4 : Logic diagram**

Fig. 5.9.49

**Example 5.9.29** Design synchronous counter which will go through the following step, using 3 flip-flop. (Avoid lock out condition).



**Solution :** The Fig. 5.9.50 shows the state diagram for the given counter. To avoid lock-out condition states 1, 4 and 6 are forced to enter into state 3.

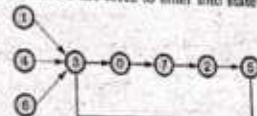


Fig. 5.9.50

- Flip-flop excitation table is as shown below.

Present state			Next state		
A	B	C	$A_{+1}$	$B_{+1}$	$C_{+1}$
0	0	0	1	1	1
0	0	1	0	1	1
0	1	0	1	0	0
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	0	0	1
1	1	0	1	0	1
1	1	1	0	1	0

Table 5.9.12

## K-map simplification

For  $D_A$

	BC	00	01	11	10
0	A	0	0	1	0
1	$\bar{A}$	0	0	0	0

$$D_A = \bar{A} \bar{C}$$

For  $D_B$

	BC	00	01	11	10
0	B	0	1	0	0
1	$\bar{B}$	1	1	1	1

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

For  $D_C$

	BC	00	01	11	10
0	C	0	1	0	1
1	$\bar{C}$	1	1	0	1

$$D_C = \bar{B} + \bar{C}$$

Fig. 5.9.51 (a)

## Logic diagram

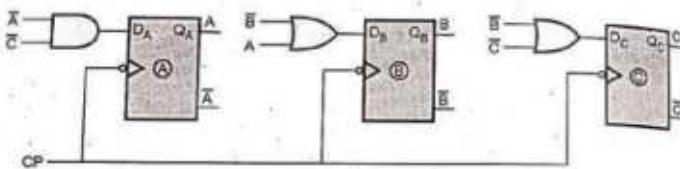


Fig. 5.9.51 (b)

**Example 5.9.30** Design a counter with the following binary sequence : 0, 1, 3, 7, 6, 4 and repeat. Use T-flip flop.

GTU : Dec-12, Marks 7

Solution :

## Step 1 : State diagram

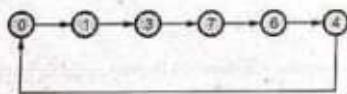


Fig. 5.9.52 (a)

## Step 2 : Truth table

Present state			Next state			Flip-Flop inputs		
$Q_A$	$Q_B$	$Q_C$	$Q'_A$	$Q'_B$	$Q'_C$	$T_A$	$T_B$	$T_C$
0	0	0	0	0	1	0	0	1
0	0	1	0	1	1	0	1	0
0	1	1	1	1	1	1	0	0

TECHNICAL PUBLICATIONS™ - An up thrust for knowledge

Table 5.9.51

## Step 3 : K-map simplification

For  $T_A$

	$Q_B Q_C$	00	01	11	10
0	0	0	1	X	
1	1	X	0	0	

$$T_A = Q_A \bar{Q}_B + \bar{Q}_A Q_B$$

For  $T_B$

	$Q_B Q_C$	00	01	11	10
0	0	0	1	X	
1	0	X	0	1	

$$T_B = \bar{Q}_B Q_C + Q_B \bar{Q}_C$$

For  $T_C$

	$Q_B Q_C$	00	01	11	10
0	0	1	0	0	X
1	0	X	1	0	

$$T_C = \bar{Q}_A \bar{Q}_B + Q_A Q_B$$

Fig. 5.9.52 (b)

## Step 4 : Logic diagram

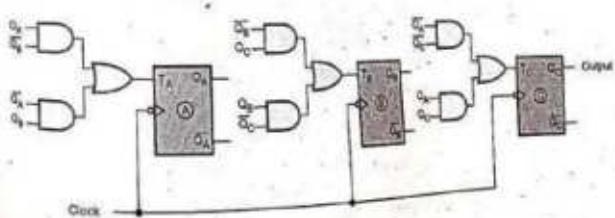


Fig. 5.9.52 (c)

TECHNICAL PUBLICATIONS™ - An up thrust for knowledge

**Review Questions**

1. Define counters.
2. State types of counters.
3. Compare synchronous and asynchronous counter.
4. What is MOD counter?
5. Give comparison between synchronous and asynchronous counters.
6. Write a note on binary ripple counter.
7. Draw and explain 3-bit asynchronous up counter using flip-flops and explain with output waveforms.
8. With logic diagram explain the operation of 4-bit binary ripple counter. How up counter can be converted into down counter?
9. Draw 4-bit asynchronous counter. Also explain timing diagram for the same.
10. Design 3-bit up-down asynchronous counter.
11. What do you mean by decoding gates?
12. Explain the problem of glitches in asynchronous counter circuits and solution for the same.
13. Explain using suitable waveforms : Problems faced by ripple counter.
14. Write a note on asynchronous counters.
15. What are decoding glitches and how these can be eliminated?
16. Design mod-8 ripple counter using J-K flip-flop. Also draw its waveforms.
17. Design modulo-5 ripple counter using 3-bit ripple counter. Draw waveforms also.
18. What do you mean by ripple counter?
19. Draw and explain 3-bit asynchronous up counter using flip-flops and explain with output waveforms.
20. Explain 4-bit up-down binary synchronous counter.
21. Design 4-bit ripple counter using negative edge triggered JK flip flop.
22. Draw a frequency divider using JK FFs to divide input clock frequency by a factor of 8.
23. Design a 3-bit synchronous up counter using K-maps and positive edge-triggered JK FFs.
24. Explain the steps in design of synchronous counter with the help of example.
25. Write a short note on up-down synchronous counter.
26. Design mod-8 ripple counter using J-K flip-flop. Also draw its waveforms.

GTU : Dec-13, Marks 1

GTU : May-14, Marks 1

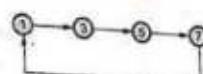
GTU : Dec-13, Marks 2

GTU : Summer-15, Marks 1

GTU : Winter-15, Marks 3

GTU : Winter-15, Marks 7

27. Design modulo-5 ripple counter using 3-bit ripple counter. Draw waveforms also.
28. What is meant when we say that a counter is presettable?
29. Describe the difference between asynchronous and synchronous presetting.
30. Write a note on skipping state counter.
31. What is lockout condition?
32. What is a self starting counter?
33. How to avoid lockout condition?



34. Design sequence generator using JK FFs. Avoid lockout condition.

**5.10 Shift Register Counters**

GTU : May-12

**5.10.1 Ring Counter**

- Fig. 5.10.1 shows the logic diagram for four-bit ring counter.

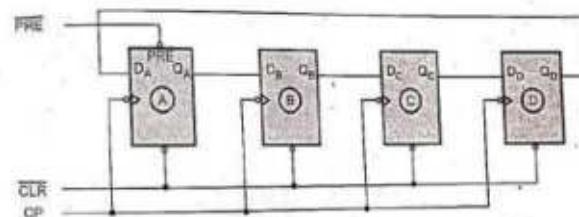


Fig. 5.10.1 Four-bit ring counter

- The Q output of each stage is connected to the D input of the next stage and the output of last stage is fed back to the input of first stage.
- The CLR followed by PRE makes the output of first stage to '1' and remaining outputs are zero, i.e.  $Q_A$  is one and  $Q_B, Q_C, Q_D$  are zero.
- The first clock pulse produces  $Q_B = 1$  and remaining outputs are zero.
- According to the clock pulses applied at the clock input CP, a sequence of four states is produced. These states are listed in Table 5.M1.

- 1 is always retained in the counter and simply shifted 'around the ring', advancing one stage for each clock pulse. In this case four stages of flip-flops are used. So a sequence of four states is produced and repeated.
- Fig. 5.10.1 gives the timing sequence for a four-bit ring counter.
- The ring counter can be used for counting the number of pulses.
- The number of pulses counted is read by noting which flip-flop is in state 1.
- No decoding circuitry is required.
- Since there is one pulse at the output for each of the N clock pulses, this circuit is also referred to as a divide-by-N-counter or an N : 1 scalar.
- Ring counters can be instructed for any desired MOD number, that is MOD-N ring counter requires N flip-flops.

Clock pulse	$Q_A$	$Q_B$	$Q_C$	$Q_D$
0	1	0	0	0
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1
4	1	0	0	0

Table 5.10.1 Ring counter sequence 4-bits

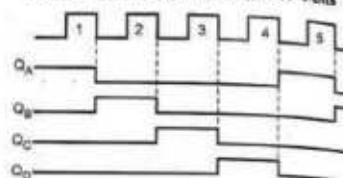


Fig. 5.10.2 Timing sequence for a four-bit ring counter

### 5.10.2 Johnson or Twisting Ring or Switch Tail Counter

- In a Johnson counter, the Q output of each stage of flip-flop is connected to the D input of the next stage.
- The single exception is that the complement output of the last flip-flop is connected back to the D-input of the first flip-flop as shown in Fig. 5.10.3.

**Note** Johnson counter can be implemented with SR or JK flip-flops as well.

- As shown in Fig. 5.10.3 there is a feedback from the rightmost flip-flop complement output to the leftmost flip-flop input. This arrangement produces a unique sequence of states.

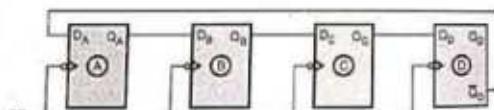


Fig. 5.10.3 Four-bit Johnson counter

- Initially, the register (all flip-flops) is cleared. So all the outputs,  $Q_A$ ,  $Q_B$ ,  $Q_C$ ,  $Q_D$  are zero.
- The output of last stage,  $Q_D$  is zero. Therefore complement output of last stage,  $\bar{Q}_D$  is one. This is connected back to the D input of first stage. So  $D_A$  is one.
- The first falling clock edge produces  $Q_A = 1$  and  $Q_B = 0$ ,  $Q_C = 0$ ,  $Q_D = 0$  since  $D_B$ ,  $D_C$ ,  $D_D$  are zero.
- The next clock pulse produces  $Q_A = 1$ ,  $Q_B = 1$ ,  $Q_C = 0$ ,  $Q_D = 0$ .
- The sequence of states is summarized in Table 5.10.2.

Clock pulse	$Q_A$	$Q_B$	$Q_C$	$Q_D$
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1

Table 5.10.2 Four-bit Johnson sequence

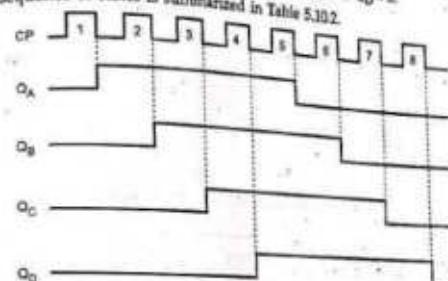


Fig. 5.10.4 Timing sequence for a four-bit Johnson counter

- After 8 states the same sequence is repeated.
- In this case, four-bit register is used. So the four-bit sequence has a total of eight states.
- Fig. 5.10.4 gives the timing sequence for a four-bit Johnson counter.
- If we design a counter of five-bit sequence, it has a total of ten states, as shown in Table 5.10.3.
- An n-stage Johnson counter will produce a modulus of  $2 \times n$ , where n is the number of stages (i.e. flip-flops) in the counter.

Clock pulse	$Q_A$	$Q_B$	$Q_C$	$Q_D$	$Q_E$
0	0	0	0	0	0
1	1	0	0	0	0
2	1	1	0	0	0
3	1	1	1	0	0
4	1	1	1	1	0
5	1	1	1	1	1
6	0	1	1	1	1
7	0	0	1	1	1
8	0	0	0	1	1
9	0	0	0	0	1

Table 5.10.3 Five-bit Johnson sequence

- Johnson counter requires only half the number of flip-flops compared to standard ring counter. However, it requires more flip-flop than binary counter.
- As shown in tables, the counter will 'fill up' with 1s from left to right and then will 'fill up' with 0s again.
- Another advantage of this type of sequence is that it is readily decoded with input AND gates.
- Table 5.10.4 gives the count sequence and required decoding.

Clock pulse	$Q_A$	$Q_B$	$Q_C$	$Q_D$	AND Gate required for output
0	0	0	0	0	$\bar{Q}_A \bar{Q}_D$
1	1	0	0	0	$Q_A \bar{Q}_B$
2	1	1	0	0	$Q_B \bar{Q}_C$
3	1	1	1	0	$Q_C \bar{Q}_D$
4	1	1	1	1	$Q_A Q_D$
5	0	1	1	1	$\bar{Q}_A Q_B$
6	0	0	1	1	$Q_B Q_C$
7	0	0	0	1	$Q_C Q_D$

Table 5.10.4 Count sequence and required decoding

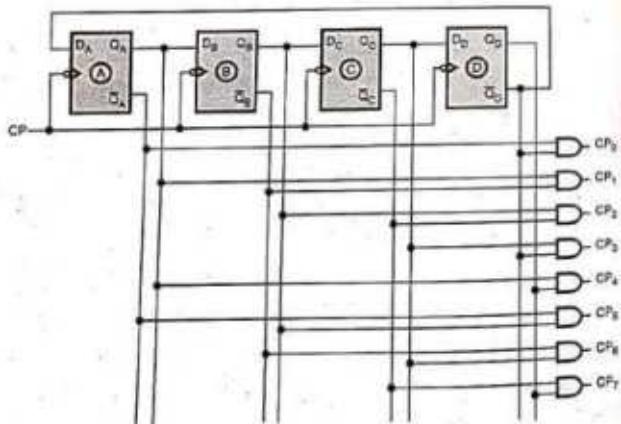


Fig. 5.10.5 Johnson counter with decoder

## Review Questions

- Draw and explain the working of 4-bit ring counter.
- Describe the operation of shift register as ring counter.
- Explain ring counter with design having initial state '1011', from initial state explain all possible states in that ring.
- Explain ring counter design for the initial condition '10110' also explain twisted ring counter in brief.
- Explain Johnson counters.
- What is the difference between ring counter and Johnson's counter?
- Draw the next circuit diagram of 3-bit Johnson's counter. Draw the relevant output waveforms.
- Explain the use of shift register as twisted ring counter with diagram.

GTU - May-12, Marks 7

GTU - Summer-18, Winter-18

## 5.11 Sequence Generator

## 5.11.1 Sequence Generator using Counters

- A sequential circuit which generates a prescribed sequence of bits, in synchronism with a clock, is referred to as a sequence generator. Fig. 5.11.1 shows the basic structure of a sequence generator using counters.

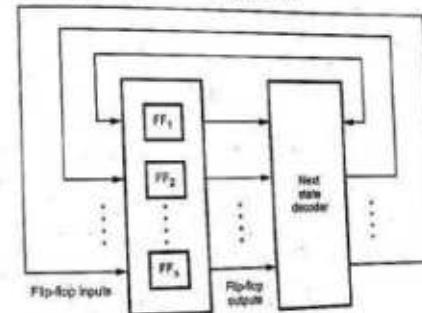


Fig. 5.11.1 Basic structure of a sequence generator

- For the design of sequence generator, we must determine the required number of flip-flops and the logic circuit for next state decoder.

## Number of flip-flops required

- Number of flip-flops required to generate particular sequence can be determined as follows :

- Find the number of 1s in the sequence.
- Find the number of 0s in the sequence.
- Take the maximum out of two.
- If  $N$  is the required number of flip-flops, choose minimum value of  $n$  to satisfy equation given below.  
 $\max(0s, 1s) \leq 2^n - 1$

**Examples with Solutions****Example 5.11.1** Find the number of flip-flops required to generate the sequence 1101011.

**Solution :** The given sequence number of 0s are 2 and number of 1s are 5. Therefore equation becomes,

$$\begin{aligned} \therefore \max(2, 5) &\leq 2^n - 1 \\ 5 &\leq 2^n - 1 \\ n &= 4 \end{aligned}$$

- Once the number of flip-flops are decided, we have to assign unique states corresponding to each bit in the given sequence such that flip-flop representing least significant bit generates the given sequence. (Usually, the output of the flip-flop representing least significant bit is used to generate the given sequence).

**Example 5.11.2** Find the state assignments for sequence 1101011.

**Solution :** We have already seen that this sequence requires four flip-flops. Assuming the output of D flip-flop as a desired sequence state assignments are shown in Table 5.11.1.

A	B	C	D	States
0	0	0	1	1
0	0	1	1	3
0	0	0	0	0
0	1	0	1	5
0	0	1	0	2
0	1	1	1	7
1	0	0	1	9

Table 5.11.1

**Example 5.11.3** Referring Table 5.11.1, draw state diagram and implement the sequence generator.

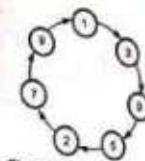
**Solution :**

Fig. 5.11.2 State diagram

Present state	Next state								Flip-Flop inputs							
	Q <sub>A</sub>	Q <sub>B</sub>	Q <sub>C</sub>	Q <sub>D</sub>	Q <sub>A+1</sub>	Q <sub>B+1</sub>	Q <sub>C+1</sub>	Q <sub>D+1</sub>	J <sub>A</sub>	K <sub>A</sub>	J <sub>B</sub>	K <sub>B</sub>	J <sub>C</sub>	K <sub>C</sub>	J <sub>D</sub>	K <sub>D</sub>
0 0 0 0	0	1	0	1	1	0	1	0	X	1	X	0	X	1	X	1
0 0 0 1	0	0	1	0	0	1	1	1	0	X	0	X	1	X	X	0
0 0 1 0	0	1	1	0	1	1	1	1	0	X	1	X	X	0	1	X
0 0 1 1	0	0	0	0	0	0	0	0	0	X	0	X	X	1	X	1
0 1 0 0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
0 1 0 1	0	0	1	0	0	1	0	0	0	X	X	1	1	X	X	1
0 1 1 0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
0 1 1 1	1	0	0	0	0	1	1	1	X	X	1	X	1	X	0	0
1 0 0 0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
1 0 0 1	0	0	0	1	0	0	1	1	X	1	0	X	0	X	X	0
1 0 1 0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
1 0 1 1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
1 1 0 0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
1 1 0 1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
1 1 1 0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
1 1 1 1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Table 5.11.2

## K-map simplification.

		For $J_A$			
$Q_A Q_B$		00	01	11	10
00	0	0	0	0	0
01	X	0	1	X	
11	X	X	X	X	
10	X	X	X	X	

$J_A = Q_B \bar{Q}_C$

		For $K_A$			
$Q_A Q_B$		00	01	11	10
00	X	X	X	X	X
01	X	0	1	X	
11	X	X	X	X	
10	X	1	X	X	

$K_A = 1$

		For $J_B$			
$Q_A Q_B$		00	01	11	10
00	1	0	0	0	1
01	X	X	X	X	X
11	X	X	X	X	X
10	X	1	X	X	X

$J_B = \bar{Q}_B$

		For $K_B$			
$Q_A Q_B$		00	01	11	10
00	X	X	X	X	X
01	X	1	1	X	
11	X	X	X	X	
10	X	X	X	X	

$K_B = 1$

		For $J_C$			
$Q_A Q_B$		00	01	11	10
00	0	1	X	X	X
01	X	1	X	X	X
11	X	X	X	X	X
10	X	0	X	X	X

$J_C = \bar{Q}_A \bar{Q}_B$

		For $K_C$			
$Q_A Q_B$		00	01	11	10
00	X	X	1	0	
01	X	X	1	0	X
11	X	X	X	X	X
10	X	0	X	X	X

$K_C = Q_B$

		For $J_D$			
$Q_A Q_B$		00	01	11	10
00	1	X	X	1	1
01	X	X	X	X	X
11	X	X	X	X	X
10	X	X	X	X	X

$J_D = 1$

		For $K_D$			
$Q_A Q_B$		00	01	11	10
00	X	0	1	X	X
01	X	1	0	X	X
11	X	X	X	X	X
10	X	0	X	X	X

$K_D = Q_B \bar{Q}_C + \bar{Q}_B Q_C$

Fig. 5.11.3

## Logic diagram

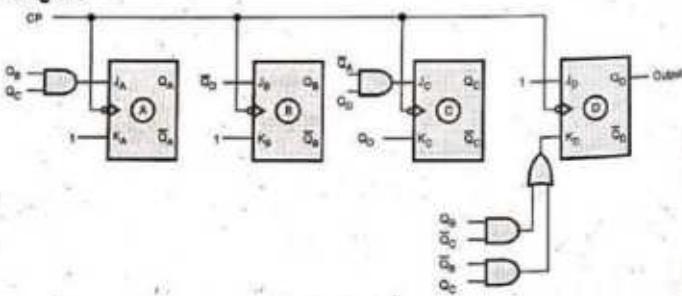


Fig. 5.11.4

TECHNICAL PUBLICATIONS™ - An up thrust for knowledge

TECHNICAL PUBLICATIONS™ - An up thrust for knowledge

Table 5.11.3

## K-map simplification

		For $D_A$			
$A \quad BC$		00	01	11	10
0	1	1	0	0	0
1	0	0	0	0	0

$D_A = \bar{A} \bar{C}$

		For $D_B$			
$A \quad BC$		00	01	11	10
0	1	1	1	0	0
1	1	1	1	1	1

$D_B = \bar{B} + A$

		For $D_C$			
$A \quad BC$		00	01	11	10
0	1	1	0	0	1
1	1	0	0	1	0

$D_C = \bar{B} + \bar{C}$

Fig. 5.11.5

TECHNICAL PUBLICATIONS™ - An up thrust for knowledge

TECHNICAL PUBLICATIONS™ - An up thrust for knowledge

## Logic diagram

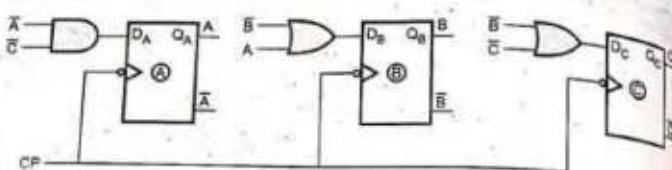


Fig. 5.11.7

**Example 5.11.5** Using J-K flip-flops, design a synchronous counter that has the following sequence:  
 $0 \rightarrow 2 \rightarrow 5 \rightarrow 6 \rightarrow 0$   
undesired states 1, 3, 4, 7 must always go to 0 on the next clock pulse.

Solution : Excitation table

Present state			Next state			Flip-flop inputs					
A	B	C	A+1	B+1	C+1	J <sub>A</sub>	K <sub>A</sub>	J <sub>B</sub>	K <sub>B</sub>	J <sub>C</sub>	K <sub>C</sub>
0	0	0	0	1	0	0	X	1	X	0	X
0	0	1	0	0	0	0	X	0	X	X	1
0	1	0	1	0	1	1	X	X	1	1	X
0	1	1	0	0	0	0	X	X	1	X	1
1	0	0	0	0	0	X	1	0	X	0	X
1	0	1	1	1	0	X	0	1	X	X	1
1	1	0	0	0	0	X	1	X	1	0	X
1	1	1	0	0	0	X	1	X	1	X	1

Table 5.11.4

## K-map simplification

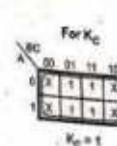
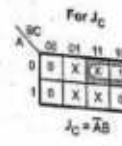
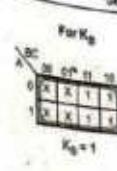
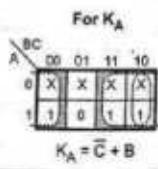
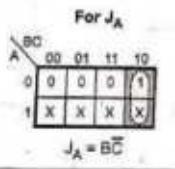


Fig. 5.11.8

## Logic diagram

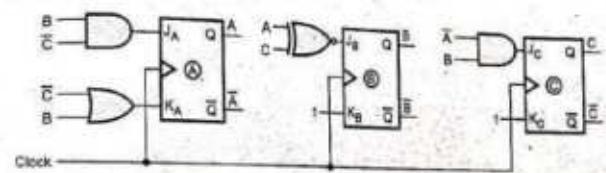


Fig. 5.11.9

**Example 5.11.6** Design and implement 4-bit binary counter (using D flip-flops) which counts all possible odd numbers only.

Solution : The state diagram for given problem is as shown in the Fig. 5.11.10.

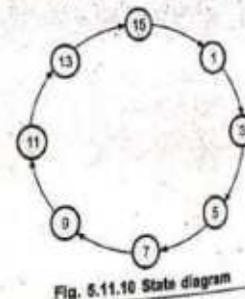
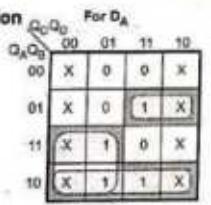


Fig. 5.11.10 State diagram

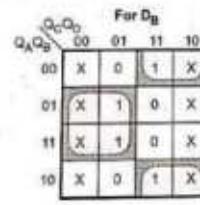
## Excitation table

Present state				Next state			
$Q_A$	$Q_B$	$Q_C$	$Q_D$	$Q_{A+}$	$Q_{B+}$	$Q_{C+}$	$Q_{D+}$
0	0	0	0	X	X	X	X
0	0	0	1	0	0	1	1
0	0	1	0	X	X	X	X
0	0	1	1	0	1	0	1
0	1	0	0	X	X	X	X
0	1	0	1	0	1	1	1
0	1	1	0	X	X	X	X
0	1	1	1	1	0	0	1
1	0	0	0	X	X	X	X
1	0	0	1	1	0	1	1
1	0	1	0	X	X	X	X
1	0	1	1	1	1	0	1
1	1	0	0	X	X	X	X
1	1	0	1	1	1	1	1
1	1	1	0	X	X	X	X
1	1	1	1	0	0	0	1

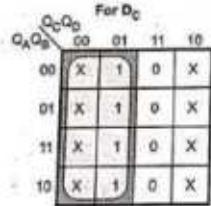
## K-map simplification



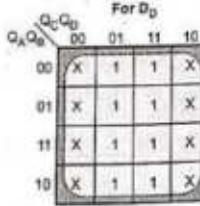
$$D_A = Q_A \bar{Q}_C + Q_A \bar{Q}_B + \bar{Q}_A Q_B Q_C$$



$$D_B = Q_B \bar{Q}_C + \bar{Q}_B Q_C$$



$$D_C = \bar{Q}_C$$



$$D_D = 1$$

Fig. 5.11.11

## Logic diagram

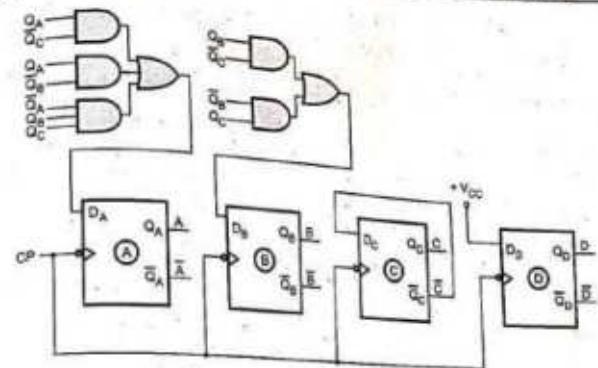


Fig. 5.11.12

**Example 5.11.7** Design a pulse-train generator to generate a pulse train 110011... using 'D' flip-flop.

Solution : The minimum number of flip-flops can be given as  $N \leq 2^6 - 1$ .

Here  $N = 6$ , therefore  $n = 3$ . Table 5.11.5 shows state encoding for given sequence using 3 flip-flops.

CP	FF O/Ps			States
	$Q_A$	$Q_B$	$Q_C$	
1	1	1	1	7
2	1	1	0	6
3	0	0	1	1
4	0	0	0	0
5	1	0	0	4
6	1	0	1	5

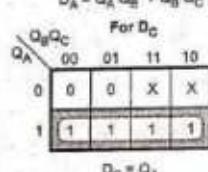
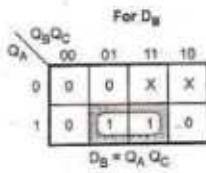
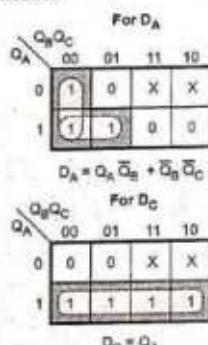
Table 5.11.5

Therefore, design a circuit to get above state. In case of D flip-flop, flip-flop inputs are same as next states.

## Excitation table

Present state			Next state/Flip-flop inputs		
$Q_A$	$Q_B$	$Q_C$	$Q'_A$	$Q'_B$	$Q'_C$
0	0	0	1	0	0
0	0	1	0	0	0
0	1	0	x	x	x
0	1	1	x	x	x
1	0	0	1	0	1
1	0	1	1	1	1
1	1	0	0	0	1
1	1	1	0	1	1

## K-map simplification



The required sequence is generated at the output of flip-flop A.

## Logic diagram

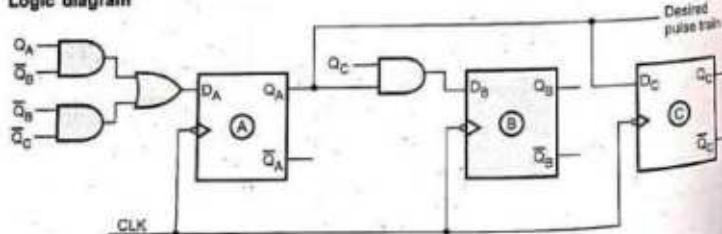


Fig. 5.11.13

Example 5.11.8 Design a non-sequential counter using JK flip-flops as per following state diagram.

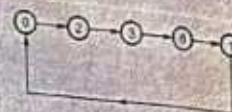
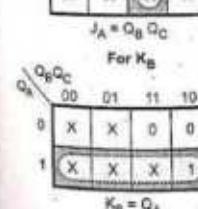
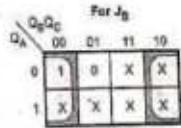
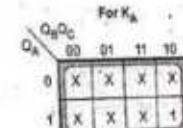
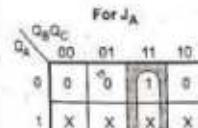


Fig. 5.11.14

Solution : Excitation table

Present state			Next state			Flip-flop inputs					
$Q_A$	$Q_B$	$Q_C$	$Q'_A$	$Q'_B$	$Q'_C$	$J_A$	$K_A$	$J_B$	$K_B$	$J_C$	$K_C$
0	0	0	0	1	0	1	x	1	x	0	x
0	0	1	0	0	0	0	x	0	x	x	1
0	1	0	0	1	1	0	x	x	0	1	x
0	1	1	1	1	0	1	x	0	x	1	1
1	0	0	x	x	x	x	x	x	x	x	x
1	0	1	x	x	x	x	x	x	x	x	x
1	1	0	0	0	1	x	1	x	1	1	x
1	1	1	x	x	x	x	x	x	x	x	x

## K-map simplification



## Logic diagram

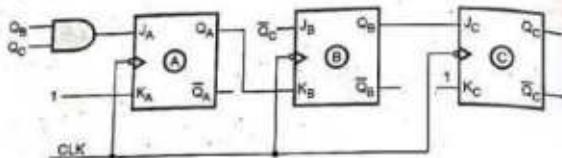


Fig. 5.11.14 (a)

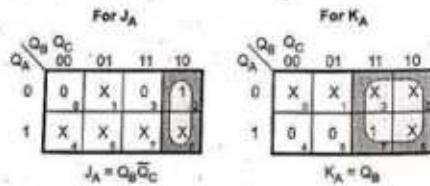
**Example 5.11.9** Design the circuit to generate the sequence

$0 \rightarrow 2 \rightarrow 5 \rightarrow 4 \rightarrow 7 \rightarrow 3$

**Solution : Excitation table**

Present state			Next state			Flip-flop Inputs					
$Q_A$	$Q_B$	$Q_C$	$Q_{A-1}$	$Q_{B-1}$	$Q_{C-1}$	$J_A$	$K_A$	$J_B$	$K_B$	$J_C$	$K_C$
0	0	0	0	1	0	0	X	1	X	0	X
0	0	1	X	X	X	X	X	X	X	X	X
0	1	0	1	0	1	1	X	X	1	1	X
0	1	1	0	0	0	0	X	X	1	X	1
1	0	0	1	1	1	X	0	1	X	1	X
1	0	1	1	0	0	X	0	0	X	X	1
1	1	0	X	X	X	X	X	X	X	X	X
1	1	1	0	1	1	X	1	X	0	X	0

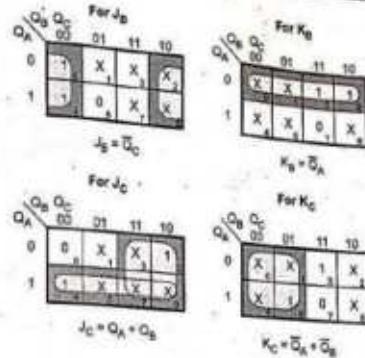
## K-map simplification



$$J_A = Q_B \bar{Q}_C$$

$$K_A = Q_B$$

TECHNICAL PUBLICATIONS™ - An up beat for knowledge



$$J_B = \bar{Q}_C$$

$$K_B = \bar{Q}_A$$

$$J_C = Q_A + Q_B$$

$$K_C = \bar{Q}_A + \bar{Q}_B$$

Fig. 5.11.15 (a)

## Logic diagram

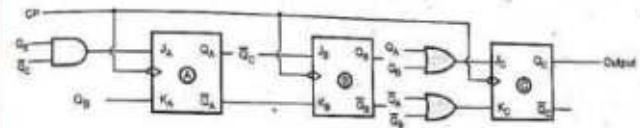


Fig. 5.11.15 (b)

**Example 5.11.10** Design sequence generator to generate sequence  $1 \rightarrow 2 \rightarrow 5 \rightarrow 4 \rightarrow 7 \rightarrow 3$ , using JK flip-flops.

$0 \rightarrow 2 \rightarrow 5 \rightarrow 4 \rightarrow 7 \rightarrow 3$

**Solution : Excitation table**

Present state			Next state			Flip-flop Inputs									
$Q_A$	$Q_B$	$Q_C$	$Q_D$	$Q_{A+1}$	$Q_{B+1}$	$Q_{C+1}$	$Q_{D+1}$	$J_A$	$K_A$	$J_B$	$K_B$	$J_C$	$K_C$	$J_D$	$K_D$
0	0	0	0	0	X	X	X	X	X	X	X	X	X	X	X
0	0	0	1	1	0	0	1	1	X	0	X	0	X	0	0
0	0	1	0	1	1	1	0	0	X	1	X	1	X	1	X

TECHNICAL PUBLICATIONS™ - An up beat for knowledge

0	0	1	1	0	1	1	0	X	1	X	X	0	X	1
0	1	0	0	X	X	X	X	X	X	X	X	X	X	X
0	1	0	1	X	X	X	X	X	X	X	X	X	X	X
0	1	1	0	0	0	1	0	X	X	1	X	X	1	X
0	1	1	1	0	0	1	1	0	X	X	1	X	0	X
1	0	0	0	X	X	X	X	X	X	X	X	X	0	X
1	0	0	1	0	0	1	0	X	1	0	X	1	X	X
1	0	1	0	X	X	X	X	X	X	X	X	X	X	1
1	0	1	1	X	X	X	X	X	X	X	X	X	X	X
1	1	0	0	X	X	X	X	X	X	X	X	X	X	X
1	1	0	1	X	X	X	X	X	X	X	X	X	X	X
1	1	1	0	X	X	X	X	X	X	X	X	X	X	X
1	1	1	1	X	X	X	X	X	X	X	X	X	X	X

**K-map simplification**

For $J_A$				
$Q_A Q_B$	00	01	11	10
X	1	0	0	
X	X	0	0	
X	X	X	X	X
X	X	X	X	X

For $K_A$				
$Q_A Q_B$	00	01	11	10
X	X	X	X	X
X	X	X	X	X
X	X	X	X	X
X	1	X	X	

$$J_A = \bar{Q}_C$$

For $J_B$				
$Q_A Q_B$	00	01	11	10
X	0	1	1	
X	X	X	X	
X	X	X	X	
X	0	X	X	

$$J_B = Q_C$$

For $K_B$				
$Q_A Q_B$	00	01	11	10
X	X	X	X	X
X	X	1	1	
X	X	X	X	
X	X	X	X	

$$K_B = 1$$

For $J_C$				
$Q_A Q_B$	00	01	11	10
X	0	X	X	X
X	X	X	X	X
X	X	X	X	X
X	1	X	X	X

$$J_C = Q_A$$

For $K_D$				
$Q_A Q_B$	00	01	11	10
X	X	0	1	X
X	X	0	X	
X	X	X	X	
X	1	X	X	X

$$K_D = Q_A \oplus \bar{Q}_B$$

For $J_D$				
$Q_A Q_B$	00	01	11	10
X	X	X	1	
X	X	X	1	
X	X	X	X	
X	X	X	X	

$$J_D = 1$$

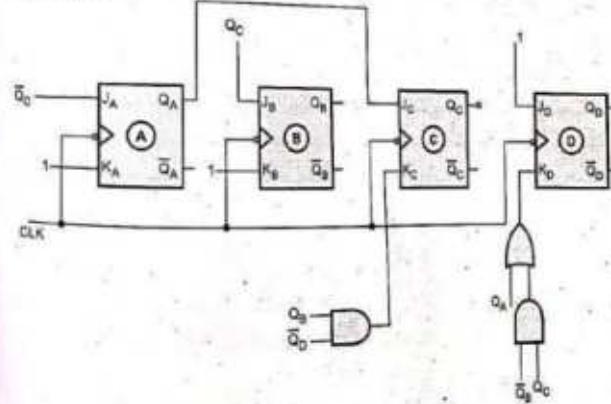
**Logic diagram**

Fig. 5.11.16

**Example 5.11.11** Design a counter to generate the repetitive sequence 0, 3, 5, 7, 4 using DFF's.

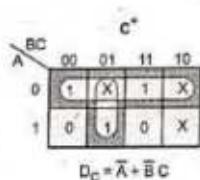
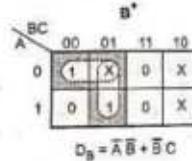
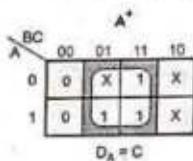
GTU : Summer-18 Marks 7

**Solution :** Step 1 : Determine the number of flip-flops needed we know that  $2^n \geq N$ .  
Here,  $N = 7 \therefore n = 3$  i.e. 3 flip-flops needed.

**Step 2 :** Determine the excitation table for counter

Present State			Next State		
A	B	C	$A'$	$B'$	$C'$
0	0	0	0	1	1
0	0	1	x	x	x
0	1	0	x	x	x
0	1	1	1	0	1
1	0	0	0	0	0
1	0	1	1	1	1
1	1	0	x	x	x
1	1	1	1	0	0

**Step 3 :** K-map simplification.



**Step 4 :** Draw logic diagram

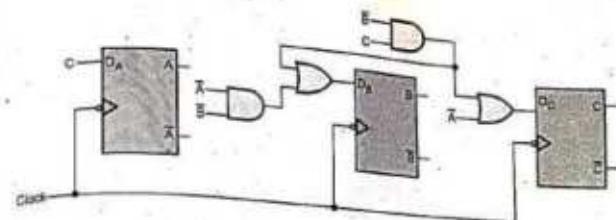


Fig. 5.11.17

**Example 5.11.12** Design a counter to generate the following binary sequence 0-1-3-4-5-0.

GTU : Winter-18 Marks 7

**Solution :** Step 1 : Determine the number of flip-flops needed we know that  $2^n \geq N$ .  
Here  $N = 6 \therefore n = 3$  i.e. 3 flip-flops needed.

**Step 2 :** Determine the excitation table for counter

Present State			Next state		
A	B	C	$A'$	$B'$	$C'$
0	0	0	0	0	1
0	0	1	0	1	1
0	1	0	x	x	x
0	1	1	1	0	0
1	0	0	1	1	0
1	0	1	x	x	x
1	1	0	0	0	0
1	1	1	x	x	x

## Step 3 : K-map simplification.

		A*			
		00	01	11	10
0		0	0	1	X
1		1	X	X	0

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

		B*			
		00	01	11	10
0		0	0	1	X
1		1	X	X	0

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

		C*			
		00	01	11	10
0		1	1	0	X
1		0	X	X	0

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

## Logic Diagram

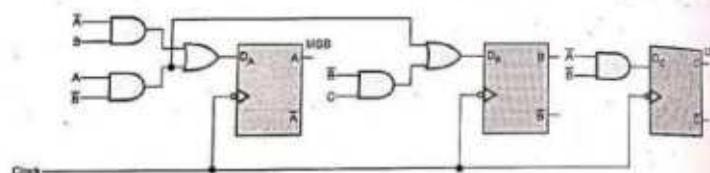


Fig. 5.11.18

## 5.11.2 Sequence Generator using Shift Register

- The simplest way of designing sequence generator using shift register is to take shift register of  $n$ -bits where  $n$  is equal to the length of sequence. Then load the bit sequence in the shift register by parallel load operation and apply the clock signal. The Fig. 5.11.19 illustrates the operation of such a sequence generator. Here, the sequence to be generated is 11001.
- Another design approach is used for sequence generator to reduce the required number of flip-flop stages in the shift register. In this approach a shift register with a next state decoder and preset logic is used. The Fig. 5.11.20 shows the block diagram of this approach. Here, the output of the next state decoder is a function of  $Q_A, Q_B, \dots, Q_n$  and it is used to determine the  $D_{in}$  input for the shift right

register. Initially, start button is pressed to activate parallel load operation. This loads initial value in the shift register. Then at each clock pulse shift register contents are shifted right by 1 bit position. The next state decoder circuit decodes the output and generates  $D_{in}$  input for the shift register such that output  $Q_A$  generates the desired sequence.

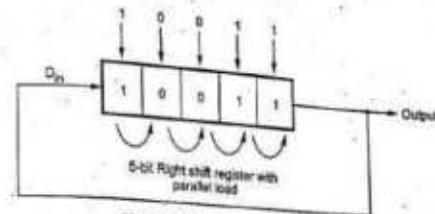


Fig. 5.11.19 (a) Sequence generator

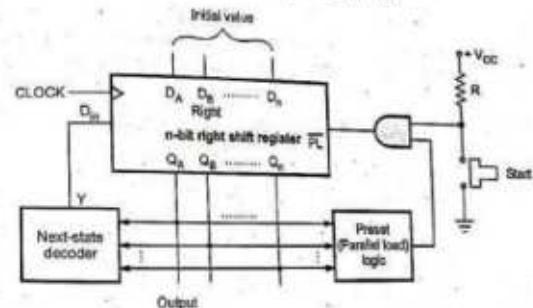


Fig. 5.11.19 (b) Sequence generator using shift register

After completion of one complete sequence, the register is again loaded with initial value to start the next train of sequence.

## Examples with Solutions

**Example 5.11.13** Design a sequence generator to generate the sequence 1101011 by shift register method.

**Solution :** In this approach, the minimum number of flip-flops  $n$ , required to generate a sequence of length  $N$  is given by

$$N \leq 2^n - 1$$

- In this example,  $N = 7$ , therefore, the minimum value of  $n$ , which may generate this sequence is 3. However, it is not guaranteed to lead to a solution. Let us try with 3 flip-flops. The Table 5.11.6 shows sequence generation with three flip-flops.

CP	Flip-flop outputs			$D_{in}$	States
	$Q_A$	$Q_B$	$Q_C$		
1	1	0	0	1	4
2	1	1	0	0	6
3	0	1	1	1	3
4	1	0	1	0	5
5	0	1	0	1	2
6	1	0	1	-	5
-	-	-	-	-	-

State is repeated →

Table 5.11.6

- As shown in the Table 5.11.7, the state 6 is repeated. This means that  $n = 3$  is not sufficient. Let us try with 4 flip-flops. The table shows sequence generation with four flip-flops.

CP	Flip-flop outputs				$D_{in}$	Preset	States
	$Q_A$	$Q_B$	$Q_C$	$Q_D$			
Initial value	1	0	0	0	1	1	8
2	1	1	0	0	0	1	12
3	0	1	1	0	1	1	6
4	1	0	1	1	0	1	11
5	0	1	0	1	1	1	5
6	1	0	1	0	1	1	10
7	1	1	0	1	1	1	13
1	1	1	1	0	X	0	14
1	1	0	0	0	1	1	8

Table 5.11.7

- As shown in the Table 5.11.7, states are not repeated. After completion of one complete sequence register is preset to value 1000 to start the next train of sequence. Fig. 5.11.20 (b) shows the logic diagram.

$Q_3 Q_2$	00	01	11	10
00	X	X	X	X
01	X	1	3	1
11	0	1	X	X
10	1	X	0	1

Fig. 5.11.20 (a)

Logic diagram

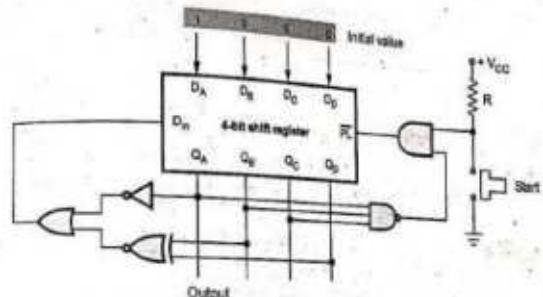


Fig. 5.11.20 (b)

**Example 5.11.14** Design and implement the following sequence generator using shift register  
1010.....

Solution : Here the sequence is 10 and it is repeated. Thus the length of the sequence  $N = 2$ . Let the number of flip-flops required  $n$  be.

$$N \leq 2^n - 1$$

$$2 \leq 2^n - 1$$

$$n = 2$$

CP	Flip-flop outputs - $D_{in}$		States
	$Q_A$	$Q_B$	
1	1	1	0
2	0	1	1
1	1	0	0

K-map simplification for  $D_{in}$ 

$Q_A$	0	1
0	X	1
1	0	X

Fig. 5.11.21 (a)

## Logic diagram

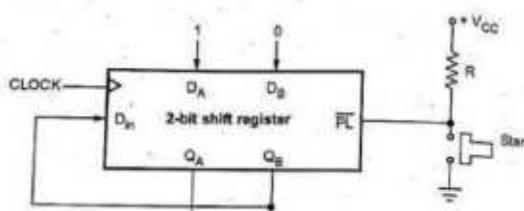


Fig. 5.11.21 (b)

In the above circuit preset/parallel load logic is only required to load the initial value. After completion of one sequence, the shift register have the same initial value and hence it is not required to reload again.

**Example 5.11.15** Design a pulse train generator using shift register for the following pulse train. .... 1 1 1 0 1 0 .....

**Solution :** The minimum number of flip-flops  $n$  can be given as

$$N \leq 2^n - 1$$

Here  $N = 6$ , therefore  $n = 3$ . The Table 5.11.8 shows sequence generation with three flip-flops.

CP	Flip-flop outputs			$D_{in}$	Preset	State
	$Q_A$	$Q_B$	$Q_C$			
1	1	0	0	1	1	4
2	1	1	0	1	1	5
3	1	1	1	1	0	6
4	0	1	1	1	1	7
5	1	0	1	0	1	8

## J-K flip-flop

Table 5.11.8

6	0	1	2	3	4	5	6	7
7	0	1	0	1	0	1	0	1
1	0	0	1	0	1	0	1	0

K-map simplification for  $D_{in}$ 

$Q_A$	0	1	1	0
0	X	0	1	0
1	0	0	0	1

Fig. 5.11.22 (a)

$$D_{in} = \bar{Q}_A Q_C + Q_A \bar{Q}_C = Q_A \oplus Q_C$$

## Logic diagram

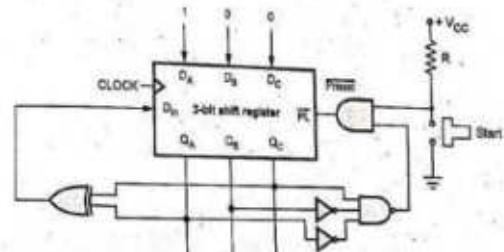


Fig. 5.11.22 (b)

**Example 5.11.16** Design a pulse train generator circuit using shift register for the following pulse train. .... 1 0 0 0 1 1 0 .....

**Solution :** The minimum number of flip-flops  $n$  required to generate sequence of length  $N$  is given by

$$N \leq 2^n - 1$$

Here,  $N = 7$ , therefore  $n = 3$ . The Table 5.11.9 (a) shows sequence generation with three flip-flops.

CP	Flip-flop outputs			$D_{in}$	States
	$Q_A$	$Q_B$	$Q_C$		
1	1	1	1	0	7
2	0	1	1	0	3
3	0	0	1	0	1
4	0	0	0	1	0
5	1	0	0	1	4
6	1	1	0	0	6
7	0	1	1	-	3

State is repeated

Table 5.11.9 (a)

As seventh state is repeated,  $n = 3$  is not sufficient. Let us try with 4 flip-flops. The Table 5.11.9 (b) shows sequence generation with four flip-flops.

CP	Flip-flop outputs				$D_{in}$	Preset	States
	$Q_A$	$Q_B$	$Q_C$	$Q_D$			
1	1	1	1	1	0	1	15
2	0	1	1	1	0	1	7
3	0	0	1	1	0	1	3
4	0	0	0	1	1	1	1
5	1	0	0	1	1	1	9
6	1	1	0	0	0	1	12
7	0	1	1	0	1	1	6
Preset flip-flop	1	1	0	1	1	X	0
	1	1	1	1	0	1	15

Table 5.11.9 (b)

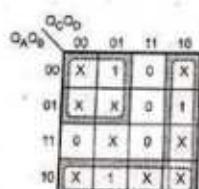
K-map simplification for  $D_{in}$ 

Fig. 5.11.23 (a)

## Logic diagram

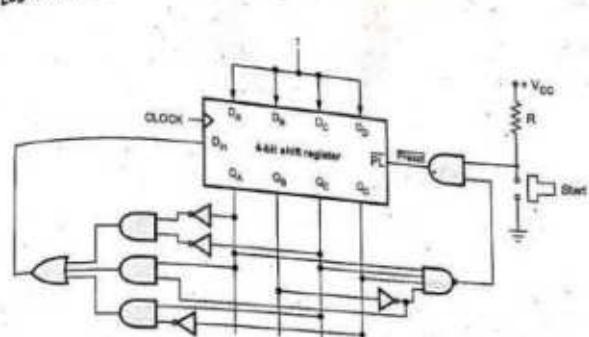


Fig. 5.11.23 (b)

**Example 5.11.17** Design sequence generator using shift register to generate sequence 1101.

Solution : The minimum number of flip-flops  $n$  can be given as

$$N \leq 2^n - 1$$

Here  $N = 4$ , therefore  $n = 3$ . The Table 5.11.10 shows sequence generation with three flip-flops.

CP	Flip-flop inputs			$D_{in}$	PL	States
	$Q_A$	$Q_B$	$Q_C$			
1	1	0	1	1	1	4
2	1	1	0	0	0	6
3	0	1	1	1	1	2
4	1	0	0	0	0	1
Preset flip-flop	0	1	0	-	-	2
	1	1	0	1	1	4

Table 5.11.10

K-map simplification for  $D_{in}$ 

$Q_B Q_C$	00	01	11	10
0	X	X	1	X
1	1	0	X	0

$D_{in} = \overline{Q}_B \overline{Q}_C + \overline{Q}_A$

Fig. 5.11.24 (a)

## Logic diagram

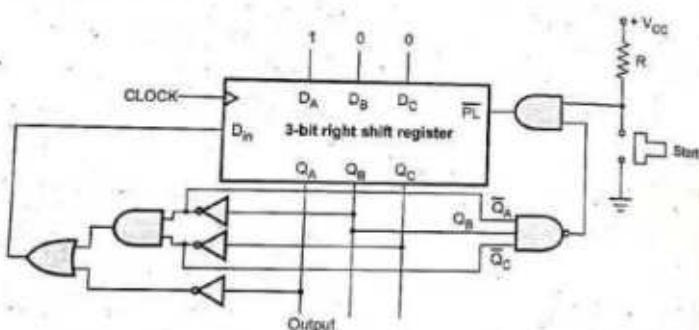


Fig. 5.11.24 (b)

## 5.12 Special Counter ICs

## 5.12.1 IC 7490 (Decade Binary Counter)

- IC 7490 is a decade binary counter. It consists of four master-slave flip-flops and additional gating to provide a divide-by-two counter and a three stage binary counter for which the count cycle length is divide by five.
- Since the output from the divide-by-two section is not internally connected to the succeeding stages, the devices may be operated in various counting modes.
- 1. BCD Decade (8421) Counter :** The B input must be externally connected to the  $Q_A$  output and A input receives the incoming count and a BCD count sequence is produced.

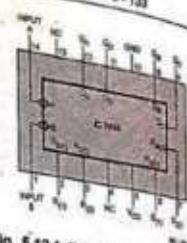


Fig. 5.12.1 Connection diagram for 7490

- Symmetrical Bi-quinary Divide-by-Ten Counter :** The  $Q_D$  output must be externally connected to the A input. The input count is then applied to the B input and a divide-by-ten square wave is obtained at output  $Q_A$ .
- Divide-by-Two and Divide-by-Five Counter :** No external interconnections are required. The first flip-flop is used as a binary element for the divide-by-two function (A as the input and  $Q_A$  as the output). The B input is used to obtain binary divide-by-five operation at the  $Q_D$  output.
- Table 5.12.1 shows function tables and Fig. 5.12.2 shows logic diagram for IC 7490. (See Fig. 5.12.2 on next page)

Count	Outputs			
	$Q_D$	$Q_C$	$Q_B$	$Q_A$
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

Table 5.12.1 (a) BCD count sequences  
(Note 1)Table 5.12.1 (b) BCD Bi-quinary (5-2) (Note 2)  
sequences (Note 1)

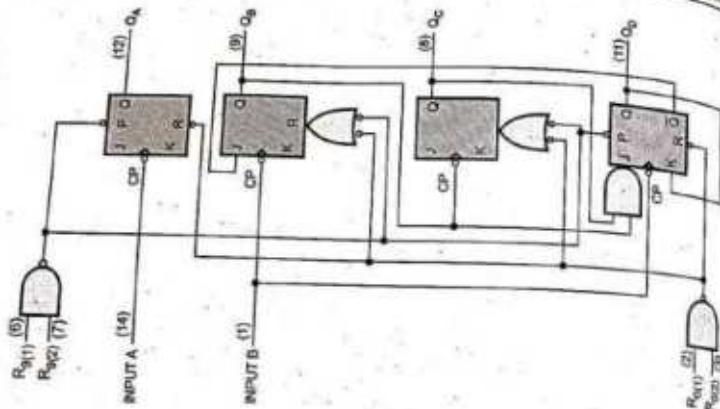


Fig. 5.12.2

Note 1 : Output  $Q_A$  is connected to input B for BCD count.

Note 2 : Output  $Q_D$  is connected to input A for bi-quinary count.

**Example 5.12.1** Design a divide-by-96 counter using 7490 ICs.

**Solution :** IC 7490 is a decade counter. When two such ICs are cascaded, it becomes a divide by 100 counter. To get a divide-by-96 counter, the counter is reset as soon as it becomes 1001 0110. The diagram is shown in Fig. 5.12.3.

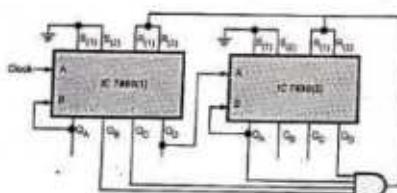


Fig. 5.12.3 Divide-by-96 counter

**Example 5.12.2** Design MOD-78 counter using IC 7490.

**Solution :** IC 7490 is a decade counter. When two such ICs are cascaded, it becomes a divide-by-100 counter. To get a divide by 78 or MOD-78 counter, the counter is reset as soon as it becomes 0111 1000 as shown in Fig. 5.12.4.

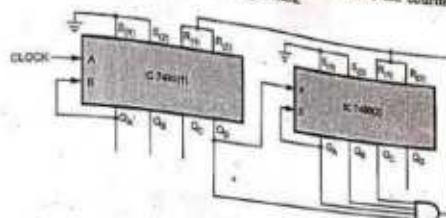


Fig. 5.12.4

**Example 5.12.3** Design Mod 8 counter using 7490.

**Solution :** MOD 8 ripple counter using 7490 :

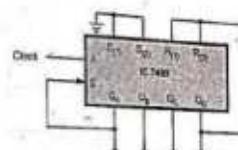


Fig. 5.12.5

**Example 5.12.4** In 7490, if  $Q_D$  output is connected to A input and pulses of B input. Find COUNT sequence and waveform of output  $Q_A$ .

**Solution :** If the  $Q_D$  output is connected to A input of 7490 IC and, input clock is applied to B input divide by ten square wave is obtained at output  $Q_A$ .

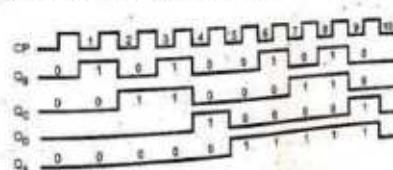


Fig. 5.12.6 Waveform

Clock	Outputs			
	Q <sub>A</sub>	Q <sub>B</sub>	Q <sub>C</sub>	Q <sub>D</sub>
0	L	L	L	L
1	L	L	L	H
2	L	L	H	L
3	L	L	H	H
4	L	H	L	L
5	H	L	L	L
6	H	L	L	H
7	H	L	H	L
8	H	L	H	H
9	H	H	L	L

Table 5.12.2

**Example 5.12.5** Design a mod 25 counter using IC 7490.

**Solution :** IC 7490 is a decade counter. When two such ICs are cascaded, it becomes a divide-by-100 counter. To get a divide-by-25 counter, the counter is reset as soon as it becomes 0010 0101. The diagram is shown in Fig. 5.12.7.

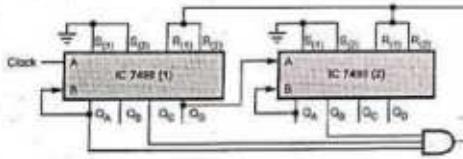


Fig. 5.12.7

### 5.12.2 IC 7493 (4-bit Ripple Counter)

- The 7493 is a high speed 4-bit ripple type counters partitioned into two sections. The counter has a divide-by-two section and divide-by-eight (7493) section which are triggered by a HIGH to LOW transition on the clock inputs. Each section can be used separately or tied together to form divide-by-sixteen counters.
  - The 7493 is 4-bit binary counter. It consists of four master slave flip-flops which are internally connected to provide a divide-by-two section. Since the output from the divide-by-two section is not internally connected to the succeeding stages, the device may be operated in various counting modes.

Digital Fundamentals  
Connection Diagram of 7493      5-137      Sequential Circuits and Systems

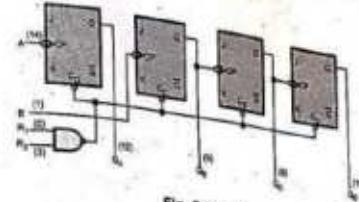


Fig. 5.12.8 (a)

Modes of 7493

- 1. 4-Bit Ripple Counter :** The output  $Q_4$  must be externally connected to input B. The input count pulses are applied to the input A. Simultaneously divisions of 2, 4, 8 and 16 are performed at the  $Q_3$ ,  $Q_2$ ,  $Q_1$  and  $Q_0$  outputs as shown in the truth table.

**2. 3-Bit Ripple Counter :** The input count pulses are applied to input A. Simultaneous frequency divisions of 2, 4 and 8 are available at the Q<sub>0</sub>, Q<sub>1</sub> and Q<sub>2</sub> outputs. Independent use of the first flip-flop is available if the reset function coincides with reset of the 3-bit ripple through counter.

- Table 5.12.3 shows the truth tables for 7493.

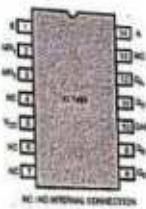


Fig. 5.12.8 (b)

Count	Output			
	Q <sub>A</sub>	Q <sub>B</sub>	Q <sub>C</sub>	Q <sub>D</sub>
0	0	0	0	0
1	1	0	0	0
2	0	1	0	0
3	1	1	0	0
4	0	0	1	0
5	1	0	-1	0
6	0	1	1	0
7	1	1	1	1
8	0	0	0	1

9	1	0	0	1
10	0	1	0	1
11	1	1	0	1
12	0	0	1	1
13	1	0	1	1
14	0	1	1	1
15	1	1	1	1

Table 5.12.3 Truth table

Note Output  $Q_A$  is connected to input B.

#### Example 5.12.6 Design a divide-by-128 counter using 7493 ICs.

Solution : Since  $128 = 16 \times 8$ , a divide-by-16 counter followed by a divide-by-8 counter will become a divide-by-128 counter. IC 7493 is a 4-bit binary counter (i.e. mod-16 or divide-by-16), therefore, two IC packages will be required.

- The circuit diagram is as shown in the Fig. 5.12.9.

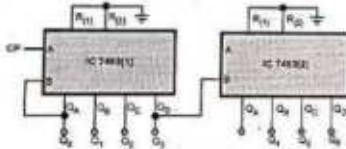


Fig. 5.12.9 Divide-by-128 counter

#### Example 5.12.7 Design divide-by-6 counter using IC 7493.

Solution : The Fig. 5.12.10 shows divide-by-6 (MOD 6) counter using 7493. As shown in the Fig. 5.12.10, the clock is applied to input B of IC 7493 and the output count sequence is taken from  $Q_D$ ,  $Q_C$  and  $Q_B$ . As soon as count is 110, i.e.  $Q_D$  and  $Q_C = 1$ , the internal NAND gate output goes low and it resets the counter.

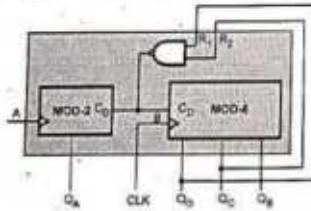


Fig. 5.12.10 Divide-by-6 counter using 7493

#### 5.12.3 IC 74192/74193 (Up/Down - BCD/Binary Counters)

- The 74192 is an UP/DOWN BCD decade counter and the 74193 is an UP/DOWN modulo-16 binary counter. Separate count up and count down clocks are used and in either counting mode the circuits operate synchronously. The outputs change state in synchronous manner with the LOW-to-HIGH transitions on the clock inputs.

- Separate terminal count up and terminal count down outputs are provided which are used as the clocks for a subsequent stage without extra logic, thus simplifying multistage counter designs. Individual preset inputs allow the circuits to be used as programmable counters. Both the Parallel Load (PL) and the Master Reset (MR) inputs asynchronously override the clocks.

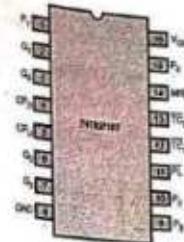


Fig. 5.12.11

#### Features :

- Low Power ... 95 mW Typical Dissipation
- High Speed ... 40 MHz Typical Count Frequency
- Synchronous Counting
- Asynchronous Master Reset and Parallel Load
- Individual Preset Inputs
- Cascading Circuitry Internally Provided
- Input Clamp Diodes Limit High Speed Termination Effects
- Fig. 5.12.11 shows the pin diagram for 74192 and 74193.

#### PIN NAMES

- |        |   |
|--------|---|
| $CP_U$ | : Count Up Clock Pulse Input                    |
| $CP_D$ | : Count Down Clock Pulse Input                  |
| MR     | : Asynchronous Master Reset (Clear) Input       |
| PL     | : Asynchronous Parallel Load (Active LOW) Input |
| $P_n$  | : Parallel Data Inputs                          |
| $Q_n$  | : Flip-Flop Outputs                             |

$\overline{TC}_D$  : Terminal Count Down (Borrow) Output  
 $\overline{TC}_U$  : Terminal Count Up (Carry) Output

#### State Equations for 74192 :

$$\begin{aligned}\overline{TC}_U &= Q_0 \cdot Q_1 \cdot \overline{CP}_U \\ \overline{TC}_D &= \bar{Q}_0 \cdot \bar{Q}_1 \cdot \bar{Q}_2 \cdot \bar{Q}_3 \cdot \overline{CP}_D\end{aligned}$$

#### State Equations for 74193 :

$$\begin{aligned}\overline{TC}_U &= Q_0 \cdot Q_1 \cdot Q_2 \cdot Q_3 \cdot \overline{CP}_U \\ \overline{TC}_D &= \bar{Q}_0 \cdot \bar{Q}_1 \cdot \bar{Q}_2 \cdot \bar{Q}_3 \cdot \overline{CP}_D\end{aligned}$$

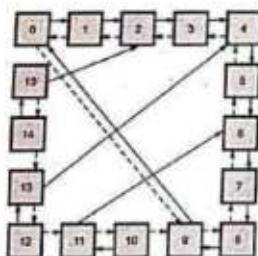


Fig. 5.12.12 (a) State diagram for 74192

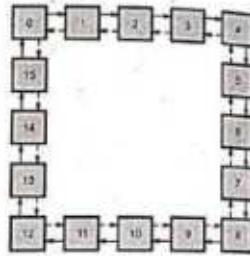


Fig. 5.12.12 (b) State diagram for 74193

#### Functional Description

- The LS192 and LS193 are asynchronously presettable decade and 4-bit binary synchronous UP/DOWN (reversible) counters. The operating modes of the LS192 decade counter and the LS193 binary counter are identical, with the only difference being the count sequences as noted in the state diagrams. Each circuit contains four master-slave flip-flops, with internal gating and steering logic to provide master reset, individual preset, count up and count down operations.
- Each flip-flop contains feedback from slave to master such that a LOW-to-HIGH transition on its T input causes the slave, and thus the Q output to change state. Synchronous switching, as opposed to ripple counting, is achieved by driving the steering gates of all stages from a common count up line and a common count down line, thereby causing all state changes to be initiated simultaneously. A LOW-to-HIGH transition on the count up input will advance the count by one; a similar transition on the count down input will decrease the count by one. While counting with one clock input, the other should be held HIGH. Otherwise, the

TECHNICAL PUBLICATIONS® - An up beat for knowledge

circuit will either count by two or not at all, depending on the state of the first flip-flop, which cannot toggle as long as either clock input is LOW.

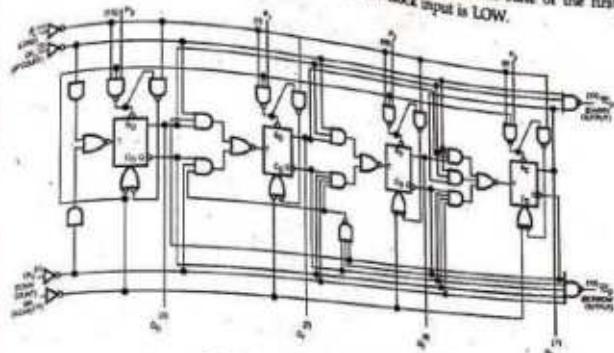


Fig. 5.12.13 Logic diagram for 74192

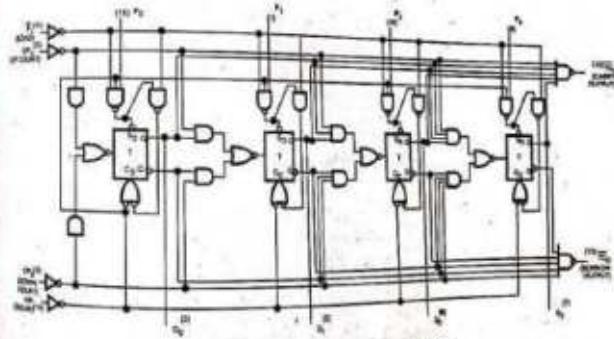


Fig. 5.12.14 Logic diagram for 74193

- The Terminal Count Up ( $\overline{TC}_U$ ) and Terminal Count Down ( $\overline{TC}_D$ ) outputs are normally HIGH. When a circuit has reached the maximum count state (9 for the LS192, 15 for the LS193), the next HIGH-to-LOW transition of the count up clock will cause  $\overline{TC}_U$  to go LOW.  $\overline{TC}_U$  will stay LOW until  $CP_U$  goes HIGH again, thus

TECHNICAL PUBLICATIONS® - An up beat for knowledge

effectively repeating the count up clock, but delayed by two gate delays. Similarly, the  $\overline{TC}$  output will go LOW when the circuit is in the zero state and the count down clock goes LOW. Since the TC outputs repeat the clock waveforms, they can be used as the clock input signals to the next higher order circuit in a multistage counter.

- Each circuit has an asynchronous parallel load capability permitting the counter to be preset. When the Parallel Load ( $\overline{PL}$ ) and the Master Reset (MR) inputs are LOW, information present on the parallel data inputs ( $P_0$ ,  $P_5$ ) is loaded into the counter and appears on the outputs regardless of the conditions of the clock inputs. A HIGH signal on the master reset input will disable the preset gates, override both clock inputs, and latch each Q output in the LOW state. If one of the clock inputs is LOW during and after a reset or load operation, the next LOW-to-HIGH transition of that clock will be interpreted as a legitimate signal and will be counted.

MR	$\overline{PL}$	$CP_U$	$CP_D$	MODE
H	X	X	X	Reset (Asyn.)
L	L	X	X	Preset (Asyn.)
L	H	H	H	No change
L	H	↑	H	Count up
L	H	H	↑	Count down

Table 5.12.4 Mode select table

L : Low voltage level

H : High voltage level

X : Don't care

**Example 5.12.3** Design a divide-by-7 DOWN counter using 74192 IC. Make use of the borrow output.

**Solution :** For the DOWN counter, the clock pulses are applied at  $CP_D$  input. When the output becomes 0, the counter is loaded with preset inputs 0111 and the states will be : 0111, 0110, 0101, 0100, 0011, 0010, and 0001. The circuit is shown in Fig. 5.12.15.

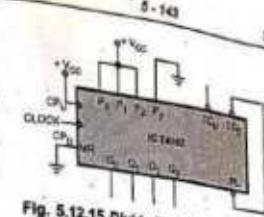


Fig. 5.12.15 Divide-by-7 down counter

**Example 5.12.4** Design a mod-12 Up counter using 74193 IC.

**Solution :** This can be done in two ways :

- Connect the circuit shown in Fig. 5.12.15. The  $Q_2$  and  $Q_3$  outputs are ANDed and are connected to clear input (with  $\overline{PL} = 1$ ). As soon as count becomes 1100, the counter is cleared.
- Connect the circuit shown in Fig. 5.12.16. The  $Q_2$  and  $Q_3$  outputs are NANDed and are connected to load input (with  $MR = 1$ ,  $P_0 = P_1 = P_2 = P_3 = 0$ ). As soon as the count reaches 1100, the counter is loaded with  $P$  inputs.

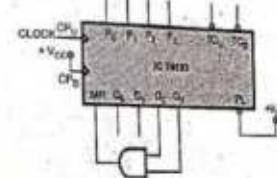


Fig. 5.12.16 Mod-12 up counter

**Example 5.12.10** Draw the timing diagram for IC 74193 showing all signals.

**Solution :** Fig. 5.12.17 shows the timing diagram for 74193.

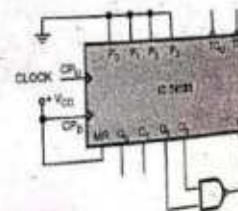


Fig. 5.12.17 Mod-12 up counter

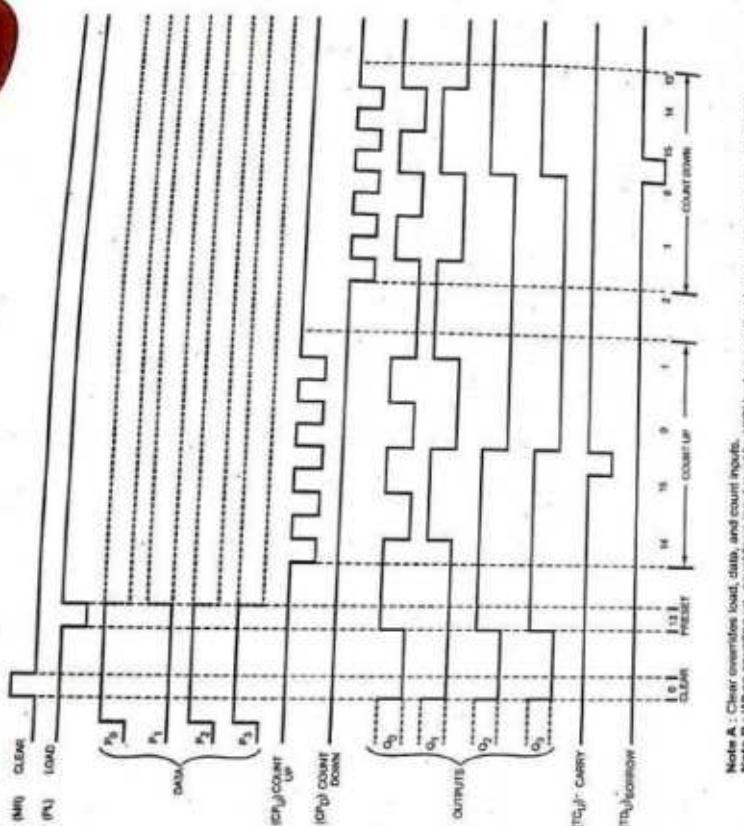


Fig. 5.12.18 Timing diagram

**Q.1** Define a sequential logic circuit. Give an example.  
**Ans. :** The circuits in which the output variables depend not only on the present input but they also depend upon the past history of these input variables are known as sequential logic circuits. Flip-flops, counters, registers are the examples of sequential logic circuit.

**Q.2** What are synchronous sequential circuits?

**Ans. :** Synchronous sequential circuits are those in which signals can affect the memory elements only at discrete instants of time. Clocked flip-flops are examples of synchronous sequential circuits.

**Q.3** Differentiate between flip-flop and latch.

Sr. No.	Latch	Flip-flop
1.	A simple latch is the basis for flip-flop building.	Flip flop is built by connecting some additional components around a basic latch.
2.	Latch is level triggered either positive level or negative level triggered.	Flip flop is pulse or clock-edge triggered either positive edge or negative edge triggered.
3.	The latch output responds to inputs, until active level is maintained at the enable input.	Flip-flop output responds to inputs only at the specified (positive or negative) edge of the clock pulse.

**Q.4** Give the meaning for edge triggering in flip-flops.

**Ans. :** In the edge triggering, the output responds to the changes in the input only at the positive or negative edge of the clock pulse at the clock input. There are two types of edge triggering.

- Positive edge triggering : Here, the output responds to the changes in the input only at the positive edge of the clock pulse at the clock input.

- Negative edge triggering : Here, the output responds to the changes in the input only at the negative edge of the clock pulse at the clock input.

**Q.5** What is the operation of D flip-flop?

**Ans. :** In D flip-flop during the occurrence of clock pulse if  $D = 1$ , the output Q is set and if  $D = 0$ , the output is reset.

**Q.6** What is the operation of JK flip-flop?

**Ans. :**

- When K input is low and J input is high the Q output of flip-flop is set.
- When K input is high and J input is low the Q output of flip-flop is reset.
- When both the inputs K and J are low the output does not change.

- When both the inputs K and J are high the output toggle on the next positive clock edge.

**Q.7 What is the operation of T flip-flop?**

**Ans. :** T flip-flop is also known as Toggle flip-flop.

- When T = 0 there is no change in the output.

- When T = 1 the output switch to the complement state (i.e.) the output toggles.

**Q.8 What is a master-slave flip-flop?**

**Ans. :** A master-slave flip-flop consists of two flip-flops where one circuit serves as a master and the other as a slave. The output of the master flip-flop is fed as an input to the slave flip-flop. The master flip-flop is triggered at the positive edge of the clock and slave flip-flop is triggered at the negative edge of the clock.

**Q.9 Give the characteristic equation and state diagram of JK flip-flop.**

**Ans. :**

- Characteristic equation :  $Q_{n+1} = J \bar{Q}_n + K$

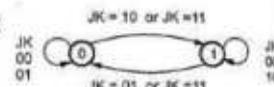


Fig. 5.1

**Q.10 What is the minimum number of flip-flops needed to design a counter of modulus 60?**

**Ans. :**  $2^n \geq 60 \therefore n = 6$

**Q.11 What is the minimum number of flip-flops required to implement a modulo 21 synchronous counter?**

**Ans. :**  $2^n \geq 21 \therefore n = 5$

**Q.12 What is meant by programmable counter? Mention its application.**

**Ans. :** A counter that divides an input frequency by a number which can be programmed, is called programmable counter.

**Applications of programmable counter**

- Frequency division
- Digital clock
- Stop watch
- Programmable logic controllers.

**Q.13 Name two sequential switching circuits.**

**Ans. :** Counter, shift register.

**Q.14 Define synchronous counter.**

**Ans. :** When counter is clocked such that each flip-flop in the counter is triggered at the same time, the counter is called synchronous counter.

**Q.15 Define asynchronous counter.**

**Ans. :** The counter in which the output of the current Flip-Flop drives the clock input of the next higher-order Flip-Flop is called asynchronous counter.

**Q.16 Draw the state diagram of MOD-10 counter.**

**Ans. :**

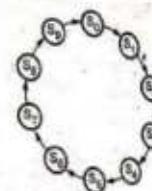


Fig. 5.2

**Q.17 Define registers.**

**Ans. :** A register is a group of flip-flops. So an n-bit register has a group of n flip-flops and is capable of storing any binary information/number containing n-bits.

**Q.18 Define shift registers.**

**Ans. :** The binary information in a register can be moved from stage to stage within the register or into or out of the register upon application of clock pulses. This type of bit movement or shifting is essential for certain arithmetic and logic operations used in microprocessors. This gives rise to group of registers called shift registers.

**Q.19 What are the different types of shift type? or Classify the registers with respect to serial and parallel input output.**

**Ans. :** There are five types. They are,

- Serial In Serial Out Shift Register
- Serial In Parallel Out Shift Register
- Parallel In Serial Out Shift Register
- Parallel In Parallel Out Shift Register
- Bidirectional Shift Register

**Q.20 What is the difference between serial and parallel transfer? What type of register is used in each case?**

**Ans. :** When data is transferred one bit at a time, the process of transfer is known as serial transfer. When multiple bits of data are transferred at a time, the process is known as parallel transfer. For parallel transfer, we can use parallel in and parallel out register. For serial transfer we can use left shift or right shift register.

**Q.21** If a serial-in-serial-out shift register has  $N$  stages and if the clock frequency is  $f$ , what will be time delay between input and output?

**Ans. :** The time delay between input and output is  $T_D = \frac{N}{f}$ .

**Q.22** Define universal shift register.

**Ans. :** The register which has both shifts (right-shift and left-shift) and parallel load capabilities is referred to as universal shift register.

**Q.23** Mention applications of shift registers.

**Ans. :** 1. Delay line 2. Serial to parallel converter 3. Parallel to serial converter  
4. Pseudo-random binary sequence generator 5. Sequence detector.

**Q.24** What is shift register counter?

**Ans. :** A shift register can also be used as a counter. A shift register with the serial output connected back to the serial input is called shift register counter.

□□□

# 6

## A / D and D / A Converters

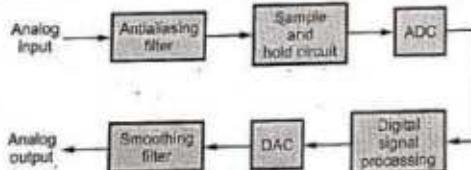
### Contents

Introduction	
6.1 D/A Converters	Winter-17, 18, Summer-18 - Marks 7
6.2 D/A Converter IC	Winter-18,
6.3 Applications of DAC	Summer-18, 19, 20 - Marks 7
6.4 Sample and Hold Circuit	
6.5 A/D Converters	
6.6 Performance Parameters (Specifications of ADC)	
6.7 Types of A/D Converters	
6.8 A/D Converter using VF Converter	Winter-19,
6.9 A/D Converter using VT Conversion	Summer-19, 20 - Marks 7
6.10 Example of A/D Converter IC	

(6 - 1)

### 6.1 Introduction

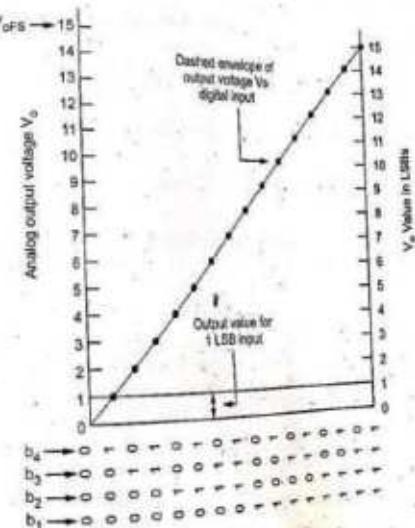
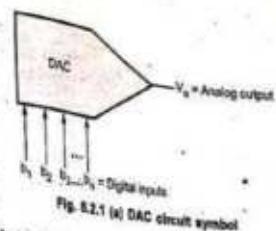
- Most of the information carrying signals such as voltage, current, charge, temperature, pressure and time are available in the analog form.
- However, for processing, transmission and storage purposes, it is often more convenient to express such signals in the digital form. When expressed in the digital form, they provide better accuracy and reduce noise.
- The development in the microprocessor technology has made it compulsory to process data in the digital form. Since digital systems such as microprocessor use a binary system of ones and zeros, we have to convert signal from analog form to digital form.
- The circuit that performs this conversion is called an analog to digital (A/D) converter.
- On the other hand, a digital to analog (D/A) converter is used when a binary output from a digital system must be converted to some equivalent analog voltage or current.
- For example, if in a particular system a computer is used as a controller, the controlling signal produced by the computer is always digital. The system to be controlled requires the analog signal. Hence in between the computer and the system to be controlled the digital to analog converter is used.
- Fig. 6.1.1 shows a typical A/D and D/A converter application. As shown in the Fig. 6.1.1, the analog signal is converted in digital form, which is then processed or perhaps just transmitted or recorded. Once processed, received or retrieved the signal is D-A converted to be reused in analog form.



**Fig. 6.1.1 Typical A/D and D/A converter application**

### 6.2 D/A Converters

- A DAC (Digital to Analog Converter) accepts an n-bit input word  $b_1, b_2, b_3, \dots, b_n$  in binary and produce an analog signal proportional to it.
- Fig. 6.2.1 (a) shows circuit symbol and input-output characteristics of a 4-bit DAC. There are four digital inputs, indicating 4-bit DAC. Each digital input requires an electrical signal representing either a logic 1 or a logic 0. The  $b_4$  is the least significant bit, LSB, whereas  $b_1$  is the most significant bit, MSB.
- Fig. 6.2.1 (b) shows analog output voltage  $V_o$  is plotted against all 16 possible digital input words.



#### Review Question

- What is difference between A/D and D/A converters?

### 6.2.1 Performance Parameters (Specifications) of DAC

#### Resolution

- Resolution is defined in two ways.
- Resolution is the number of different analog output values that can be provided by a DAC. For an n-bit DAC
 
$$\text{Resolution} = 2^n \quad \dots (6.2.1)$$
- Resolution is also defined as the ratio of a change in output voltage resulting from a change of 1 LSB at the digital inputs. For an n-bit DAC it can be given as
 
$$\text{Resolution} = \frac{V_{FS}}{2^n - 1} \quad \dots (6.2.2)$$
 where,  $V_{FS}$  = Full scale output voltage
- From equation (6.2.1), we can say that, the resolution can be determined by the number of bits in the input binary word. For an 8-bit DAC resolution can be given as
 
$$\text{resolution} = 2^8 = 2^8 = 256$$
- If the full scale output voltage is 10.2 V then by second definition the resolution for an 8-bit DAC can be given as
 
$$\text{Resolution} = \frac{V_{FS}}{2^n - 1} = \frac{10.2}{2^8 - 1} = \frac{10.2}{255} = 40 \text{ mV/LSB}$$

Therefore, we can say that an input change of 1 LSB causes the output to change by 40 mV.

- From the resolution, we can obtain the input-output equation for a DAC.  
Thus  $V_o = \text{Resolution} \times D$   
where,  $D$  = Decimal value of the digital input  
and  $V_o$  = Output voltage
- The resolution takes care of changes in the input.

#### Accuracy

- It is a comparison of actual output voltage with expected output. It is expressed in percentage. Ideally, the accuracy of DAC should be, at worst,  $\pm \frac{1}{2}$  of its LSB. If the full scale output voltage is 10.2 V then for an 8-bit DAC accuracy can be given as

$$\text{Accuracy} = \frac{V_{FS}}{(2^n - 1) 2} = \frac{10.2}{255 \times 2} = 20 \text{ mV} \quad \dots (6.2.3)$$

#### Settling Time (Conversion Time)

- The operating conversion speed of a DAC is usually specified by giving its settling time. The settling time is the time required for the DAC output to go from zero to full scale as the binary input is changed from all 0s to all 1s.

- Actually, the settling time is measured as the time for the DAC output to settle within  $\pm 1/2$  step size ( $1/2$  LSB) of its final value.
- Typical values for settling time range from 50 ns to 10  $\mu$ s. It depends on word length, circuit technology and architecture.
- Generally, DAC's with a current output will have shorter settling times than those with voltage outputs.

#### Dynamic Range

- The dynamic range of DAC is defined as the ratio of the largest output to the smallest output, excluding zero. It is expressed in dB. For linear DACs it is given by,

$$\text{Dynamic range} = 20 \log 2^n = 6n$$

- For companding type DAC, the dynamic range is more. It is typically 66 to 72 dB.

#### Offset Error

- The offset error is defined as the nonzero level of the output voltage when all inputs are zero.
- It adds a constant value to all output values, as shown in Fig. 6.2.2.
- It is due to the presence of offset voltage in op-amp and leakage currents in the current switches.

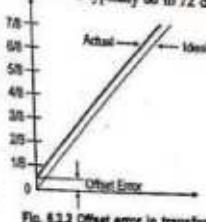


Fig. 6.2.2 Offset error in transfer characteristics of DAC

#### Gain Error

- The gain error is defined as the difference between the calculated gain of the current to voltage converter and the actual gain achieved. It is due to the errors in the feedback resistor on the current to voltage converter op-amp.
- Fig. 6.2.3 shows the gain error in transfer characteristics of DAC.

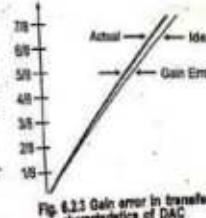


Fig. 6.2.3 Gain error in transfer characteristics of DAC

#### Non-Linearity (Linearity Error)

- An ideal DAC should be linear. For an ideal DAC, the output voltage would be a linear function of the input code. But it is fact that all DAC departs somewhat from the ideal linearity. The typical factors responsible for introducing non-linearity are :

- Non-exact values of resistors, and
- Non-ideal electronic switches that introduce extra resistance to the circuit.
- The non-linearity error is the amount by which the actual output differs from the ideal straight-line output. The different types of non-linearity are :
  - Integral non-linearity,
  - Differential non-linearity.

**Stability**

- The performance of converter changes with temperature, age and power supply variations. So all the relevant parameters such as offset, gain, linearity error and monotonicity must be specified over the full temperature and power supply ranges. These parameters represent the stability of the converter.

**Example 6.2.1** An 8-bit DAC has an output voltage range of 0 - 2.55 V. Define its resolution in two ways.

**Solution :** For the given DAC,

$$n = \text{Number of bits} = 8$$

i) Resolution =  $2^n = 2^8 = 256$

i.e. the output voltage can have 256 different values including zero.

ii)  $V_{oFS} = \text{Full scale output voltage} = 2.55 \text{ V}$

$$\therefore \text{Resolution} = \frac{V_{oFS}}{2^n - 1} = \frac{2.55}{2^8 - 1} = \frac{10 \text{ mV}}{1 \text{ LSB}}$$

Thus an input change of 1 LSB causes the output to change by 10 mV.

**Example 6.2.2** An 8-bit D/A converter has resolution of 10 mV/bit. Find the analog output voltage for the following digital inputs.

i) 10001010 ii) 00010000

**Solution :** i) 10001010

$$D = \text{Equivalent of } (10001010)_2 = 138$$

$$V_o = \text{Resolution} \times D = 10 \times 10^{-3} \times 138 = 1.38 \text{ V}$$

ii) 00010000

$$D = \text{Equivalent of } (00010000)_2 = 16$$

$$V_o = \text{Resolution} \times D = 10 \times 10^{-3} \times 16 = 0.16 \text{ V}$$

**Example 6.2.3** An 8-bit DAC has resolution of 20 mV/LSB. Find  $V_{oFS}$  and  $V_o$  if the input is (10000000)<sub>2</sub>.

$$\text{Solution : Resolution} = \frac{V_{oFS}}{2^n - 1} \therefore 20 = \frac{V_{oFS}}{2^8 - 1}$$

$$V_{oFS} = 5.1 \text{ V}$$

$$D = \text{Equivalent of } (10000000)_2 = 128$$

$$V_o = \text{Resolution} \times D = 20 \times 10^{-3} \times 128 = 2.56 \text{ V}$$

**Example 6.2.4** Find out step size and analog output for 4-bit R/2R ladder DAC when input is (1000)<sub>2</sub> and (1111)<sub>2</sub>. Assume  $V_{oFS} = +5 \text{ V}$ .

**Solution :** For given DAC,  $n = 4$ ,  $V_{oFS} = +5 \text{ V}$

$$\text{Resolution} = \frac{V_{oFS}}{2^n - 1} = \frac{5}{2^4 - 1} = \frac{1}{3} \text{ V/LSB}$$

$$V_o = \text{Resolution} \times D$$

$$\text{For } D = \text{Decimal of } (1000)_2 = 8$$

$$V_o = \frac{1}{3} \times 8 = 2.6667 \text{ V}$$

$$\text{For } D = \text{Decimal of } (1111)_2 = 15$$

$$V_o = \frac{1}{3} \times 15 = 5 \text{ V}$$

**Example 6.2.5** A 12-bit DAC has a step size of 8 mV. Determine the full scale output voltage and percentage resolution. Also find the output voltage for the input of (010101010101)<sub>2</sub>.

**Solution :** For 12-bit DAC, step size is 8 mV.

$$V_{oFS} = 8 \text{ mV} \times 2^{12} - 1 = 32.76 \text{ V}$$

$$\% \text{ Resolution} = \frac{8 \text{ mV}}{32.76 \text{ V}} \times 100 = 0.02442$$

The output voltage for the input (010101010101)<sub>2</sub> is  $8 \text{ mV} \times (1389)_2 = 11.112 \text{ V}$

**6.2.2 Basic Conversion Techniques**

- There are mainly two techniques used for digital to analog conversion :

- Binary weighted resistor D/A converter
- R/2R ladder D/A converter

- In these techniques, the shunt resistors are used to generate n binary weighted currents. These currents are added according to switch positions controlled by the digital input and then converted into voltage to give analog voltage equivalent to the digital input. Therefore, such digital to analog converters are called current driven DACs.

#### 6.2.2.1 Binary Weighted Resistor D/A Converter

- The binary weighted resistor DAC uses an op-amp to sum n binary weighted currents derived from a reference voltage  $V_R$  via current scaling resistors  $2R, 4R, \dots, 2^n R$ , as shown in the Fig. 6.2.4.

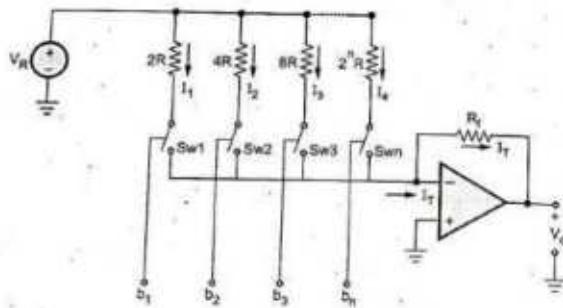


Fig. 6.2.4 Binary weighted resistor DAC

- As shown in the Fig. 6.2.4, switch positions are controlled by the digital inputs. When digital input is logic 1, it connects the corresponding resistance to the reference voltage  $V_R$ ; otherwise it leaves resistor open. Therefore,

$$\text{For ON-switch, } I = \frac{V_R}{R} \quad \text{and}$$

$$\text{For OFF-switch, } I = 0$$

- The operational amplifier is used as a summing amplifier. Due to high input impedance of op-amp, summing current will flow through  $R_f$ . Hence the total current through  $R_f$  can be given as

$$I_T = I_1 + I_2 + I_3 + \dots + I_n \quad \dots (6.2.4)$$

- The output voltage is the voltage across  $R_f$  and it is given as

$$V_o = -I_T R_f = -(I_1 + I_2 + I_3 + \dots + I_n) R_f \quad \dots (6.2.5)$$

$$= - \left( b_1 \frac{V_R}{2R} + b_2 \frac{V_R}{4R} + b_3 \frac{V_R}{8R} + \dots + b_n \frac{V_R}{2^n R} \right) R_f \quad \dots (6.2.6)$$

$$\text{When } R_f = R, V_o \text{ is given as} \quad \dots (6.2.7)$$

$$V_o = -V_R (b_1 2^{-1} + b_2 2^{-2} + b_3 2^{-3} + \dots + b_n 2^{-n}) \quad \dots (6.2.8)$$

- The equation (6.2.7) indicates that the analog output voltage is proportional to the input digital word.
- The simplicity of the binary weighted DAC is offset by drawbacks associated with it.

#### Drawbacks :

- Wide range of resistor values are required. For 8-bit DAC, the resistors required are  $2^1 R, 2^2 R, 2^3 R, \dots$  and  $2^8 R$ . Therefore, the largest resistor is 128 times the smallest.
- This wide range of resistor values has restrictions on both, higher and lower ends. It is impractical to fabricate large values of resistor in IC, and voltage drop across such a large resistor due to the bias current also affects the accuracy. For smaller values of resistors, the loading effect may occur.
- The finite resistance of the switches disturbs the binary-weighted relationship among the various currents, particularly in the most significant bit positions, where the current setting resistances are smaller.
- All these drawbacks, especially the requirement of wide range of resistors restricts the use of binary weighted resistor DACs below 8-bits.

#### 6.2.2.2 R/2R Ladder D/A Converter

##### Inverter R/2R Ladder (Current Steering Mode) D/A Converter

- R/2R ladder D/A converter uses only two resistor values. This avoids resistance spread drawback of binary weighted D/A converter.
- Fig. 6.2.5 shows R/2R ladder DAC. Like binary weighted resistor DAC, it also uses shunt resistors to generate n binary weighted currents; however it uses voltage scaling and identical resistors instead of resistor scaling and common voltage reference used in binary weighted resistor DAC. Voltage scaling requires an additional set of voltage dropping series resistances between adjacent nodes, as shown in the Fig. 6.2.5.

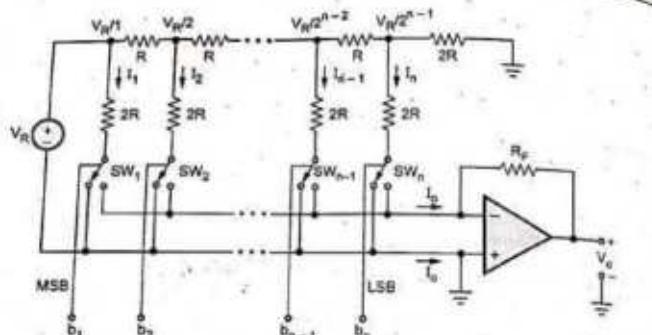


Fig. 6.2.5

- Here, each bit of the binary word connects the corresponding switch either to ground or to the inverting input terminal of the op-amp which is at the virtual ground. Since both the positions of switches are at ground potential, the current flowing through resistances is constant and it is independent of switch position. These currents can be given as,

$$I_1 = \frac{V_R}{2R} \quad \dots (6.2.9)$$

$$I_2 = \frac{V_R/2}{2R} = \frac{V_R}{4R} = \frac{I_1}{2}$$

$$I_3 = \frac{V_R/4}{2R} = \frac{V_R}{8R} = \frac{I_1}{4}$$

$$I_n = \frac{V_R / 2^{(n-1)}}{2R} = \frac{I_1}{2^{(n-1)}} \quad \dots (6.2.10)$$

- We know that,  $V_o$  is given as,

$$V_o = -I_f R_f \quad \dots (6.2.11)$$

$$V_o = -R_f (I_1 + I_2 + I_3 + \dots + I_n)$$

$$= -R_f \left( b_1 \frac{V_R}{2R} + b_2 \frac{V_R}{4R} + b_3 \frac{V_R}{8R} + \dots + b_n \frac{V_R}{2^n R} \right)$$

$$= \frac{-V_R R_f}{R} (b_1 2^{-1} + b_2 2^{-2} + b_3 2^{-3} + \dots + b_n 2^{-n}) \quad \dots (6.2.12)$$

- When  $R_f = R$ ,  $V_o$  is given as,

$$V_o = -V_R (b_1 2^{-1} + b_2 2^{-2} + b_3 2^{-3} + \dots + b_n 2^{-n}) \quad \dots (6.2.13)$$

- Let us consider 4-bit binary DAC with binary input 1001 and  $R_f = R$ , as shown in the Fig. 6.2.6.

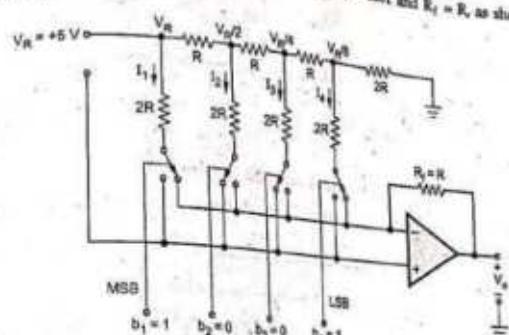


Fig. 6.2.6 Inverted 4-bit R/2R ladder DAC

- Here, output voltage is given as

$$V_o = -V_R (1 \times 2^{-1} + 0 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4})$$

$$= -V_R \left( \frac{1}{2} + 0 + 0 + \frac{1}{16} \right) = -0.5625 V_R = 2.8125 V$$

- The inverting R/2R ladder DAC works on the principle of summing currents and it is also said to operate in the current steering mode.
- An important advantage of the current mode is that all ladder node voltages remain constant with changing input codes, thus avoiding any shutdown effects by stray capacitances.

#### R2R Ladder (Voltage Switching Mode) D/A Converter

- In this type, reference voltage is applied to one of the switch positions, and other switch position is connected to ground, as shown in the Fig. 6.2.7.

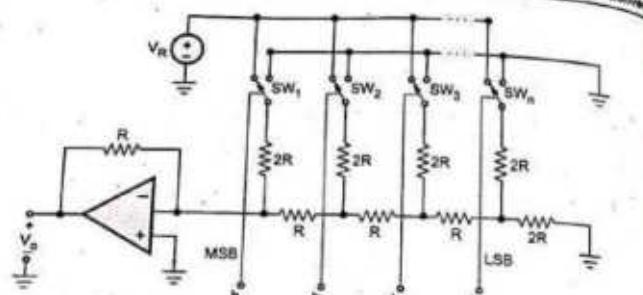


Fig. 6.2.7 R/2R ladder DAC converter

- Let us consider 3-bit R/2R ladder DAC with binary input 001, as shown in the Fig. 6.2.8.

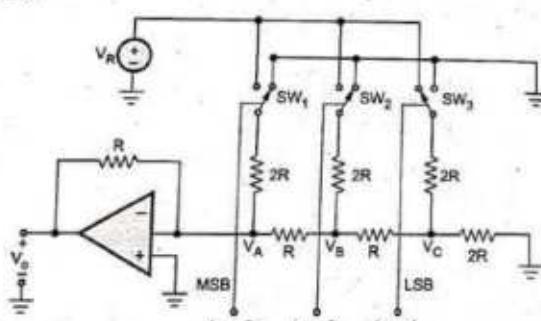


Fig. 6.2.8 3-bit R/2R ladder DAC

- Reducing above network to the left by Thevenin's theorem we get,

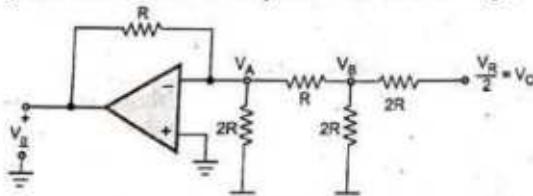


Fig. 6.2.9 (a)

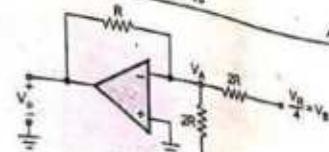


Fig. 6.2.9 (b)

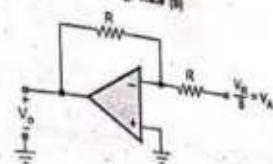


Fig. 6.2.9 (c)

Therefore, the output voltage is  $V_R/8$  which is equivalent to binary input 001.

- For binary input 100 the network can be reduced as follows :

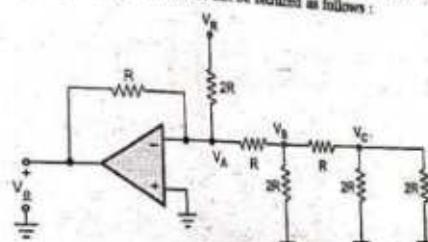


Fig. 6.2.10 (a)

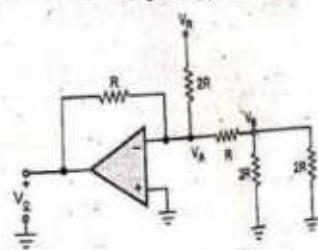


Fig. 6.2.10 (b)

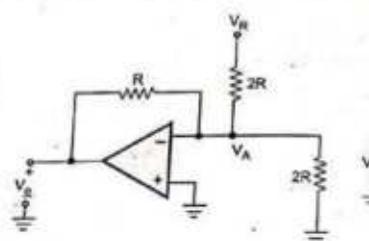


Fig. 6.2.10 (c)

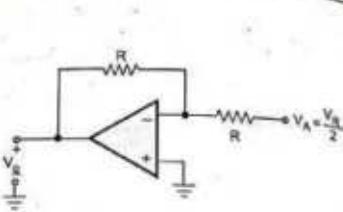


Fig. 6.2.10 (d)

Therefore, the output voltage is  $V_R/2$ , which is equivalent to binary input 100.

- In general, the voltage is given by,

$$V_o = -V_R(b_1 2^{-1} + b_2 2^{-2} + b_3 2^{-3} + \dots + b_n 2^{-n})$$

#### Advantages of R/2R ladder DACs

- Easier to build accurately as only two precision metal film resistors are required.
- Number of bits can be expanded by adding more sections of same R/2R values.
- In inverted R/2R ladder DAC, node voltages remain constant with changing input binary words. This avoids any slowdown effects by stray capacitances.

#### Expression for output voltage

- The expression for  $V_o$  can be obtained as,

Let  $I_{out}$  = Output current

$R_f$  = feedback resistance of op-amp

$$\therefore V_o = -I_{out} R_f$$

Now  $I_{out}$  = current resolution  $\times$  D

$$\therefore V_o = -(current\ resolution \times D) R_f$$

$$\therefore V_o = -(current\ resolution \times R_f) \times D \quad \dots (6.2.14)$$

- The coefficient of D is the voltage resolution and can be called simple resolution.

$$\therefore V_o = -Resolution \times D \quad \dots (6.2.15)$$

- In terms of actual circuit elements, output can be written as,

$$V_o = -\left(\frac{V_R}{R} \times \frac{1}{2^n} R_f\right) \times D \quad \dots (6.2.16)$$

#### Solution

- The resolution of R/2R ladder type DAC with current output is,

$$\text{Resolution} = \frac{1}{2^n} \times \frac{V_R}{R}$$

while the resolution of R/2R ladder type DAC with voltage output is,

$$\text{Resolution} = \left(\frac{1}{2^n} \times \frac{V_R}{R}\right) \times R_f \quad \dots (6.2.17)$$

**Example 6.2.6** Suggest the values of resistors and reference voltage if resolution required is 0.5 V for 4 bit R/2R ladder type DAC

$$\text{Solution : Resolution} = \left(\frac{1}{2^n} \times \frac{V_R}{R}\right) \times R_f$$

Let  $V_R = 10$  V,  $n = 4$  and resolution = 0.5

$$0.5 = \frac{1}{2^4} \times \frac{10}{R} \times R_f \therefore \frac{R}{R_f} = 1.25$$

Choose  $R_f = 10$  kΩ

$$R = 12.5$$
 kΩ

**Example 6.2.7** Calculate the output voltage and conversion resolution of 8-bit R-2R ladder DAC for the input data of 11011101 with reference voltage of 10 V.

Solution :

$$\text{Resolution} = \frac{V_{oPS}}{2^n - 1} = \frac{10}{2^8 - 1} = 0.0392$$

$$V_o = \text{Resolution} \times D$$

$$D = \text{Equivalent of } (11011101)_2 = (221)_{10}$$

$$V_o = 0.0392 \times 221 = 8.663$$

**Example 6.2.8** A 5-bit R-2R ladder network with reference voltage of 10 V.

Find :

- Analog output due to LSB change w.r.t Full scale output voltage
- Analog output for digital input 11001

Solution : i) Analog output due to LSB change :

$$V_o = -V_R \left( \frac{1}{2^5} \right) = -10 \times \frac{1}{32} = -0.3125$$

ii) Full scale output voltage :  $V_{oPS} = 9.8875$  V

iii) Analog output for digital input 11001 :

$$(11001)_2 = 25$$

$$25 \times -0.3125 = -7.8125 \text{ V}$$

#### Review Questions

1. What is D/A converter? Explain any two types of D/A converters along with characteristic features.
2. Define the following terms for D/A converters : i) Resolution ii) Accuracy iii) Monotonicity iv) Conversion time
3. Draw a 2-bit-D/A converter with R-2R resistors and explain its operation? State its advantages?
4. Explain the R/2R ladder techniques of D/A conversion.
5. Explain the binary weighted resistor technique of D/A conversion.
6. Explain the advantages of R/2R ladder technique over binary weighted resistor technique.
7. Draw the circuit diagram of voltage mode R-2R ladder DAC and explain its working.
8. List the various methods of D/A conversion, state the advantages and disadvantages of each.
9. State the specifications of DAC. Also explain the applications of DAC.
10. Explain the following with reference to DAC :
  - i) Linearity
  - ii) Accuracy
  - iii) Settling time.
11. List various specifications of DAC.

#### 6.3 D/A Converter IC

- The 1408/DAC0808 is an 8-bit R/2R ladder type D/A converter compatible with TTL and CMOS logic. It is designed to use where the output current is linear product of an eight-bit digital word.
- Fig. 6.3.1 shows the pin diagram and block diagram for IC 1408/DAC0808.
- The IC 1408/DAC0808 consists of a reference current amplifier, an R/2R ladder and eight high speed current switches. It has eight

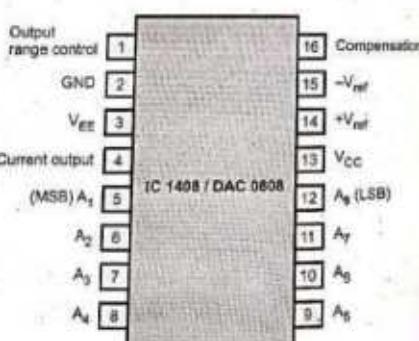


Fig. 6.3.1 (a) Pin diagram

- input data lines  $A_1$  (MSB) through  $A_8$  (LSB) which control the positions of current switches.
- It requires 2 mA reference current for full scale input and two power supplies  $V_{CC} = +5 \text{ V}$  and  $V_{EE} = -15 \text{ V}$  ( $V_{EE}$  can range from  $-5 \text{ V}$  to  $-15 \text{ V}$ ).

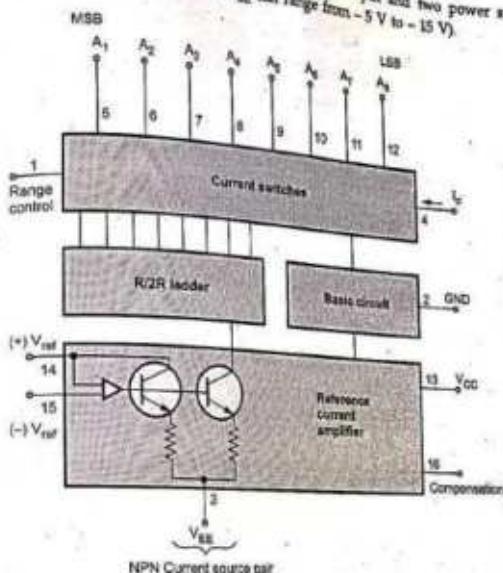


Fig. 6.3.1 (b) Block diagram

- The voltage  $V_{ref}$  and resistor  $R_{14}$  determines the total reference current source and  $R_{15}$  is generally equal to  $R_{14}$  to match the input impedance of the reference current amplifier.
- Fig. 6.3.2 shows a typical circuit for IC 1408/DAC0808. The output current  $I_o$  can be given as,

$$I_o = \frac{V_{ref}}{R_{14}} \left( \frac{A_1}{2} + \frac{A_2}{4} + \frac{A_3}{8} + \frac{A_4}{16} + \frac{A_5}{32} + \frac{A_6}{64} + \frac{A_7}{128} + \frac{A_8}{256} \right) \quad \dots(6.3.1)$$

**Note** Input  $A_1$  through  $A_8$  can be either 0 or 1. Therefore, for typical circuit full scale current can be given as,

$$I_o = \frac{3}{2.5K} \left( \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \frac{1}{64} + \frac{1}{128} + \frac{1}{256} \right) = \frac{2 \text{ mA} \times 155}{256} = 1.992 \text{ mA}$$

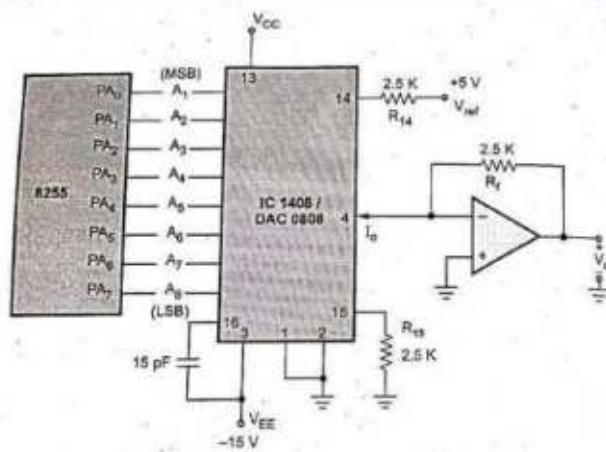


Fig. 6.3.2 Typical circuit for IC 1408/DAC0808

It shows that the full scale output current is always 1 LSB less than the reference current source of 2 mA. This output current is converted into voltage by I to V converter. The output voltage for full scale input can be given as,

$$V_o = 1.992 \times 2.5 \text{ K} = 4.98 \text{ V}$$

**Note** The arrow on the pin 4 shows the output current direction. It is inward. This means that IC 1408/DAC0808 sinks current. At (0000 0000)<sub>2</sub> binary input it sinks zero current and at (1111 1111)<sub>2</sub> binary input it sinks 1.992 mA.

- The circuit shown in the Fig. 6.3.3 gives output in the unipolar range. When digital input is 00H, the output voltage is 0 V and when digital input is FFH (1111 1111)<sub>2</sub>, the output voltage is + 5 V. This circuit can be modified to give bipolar output.

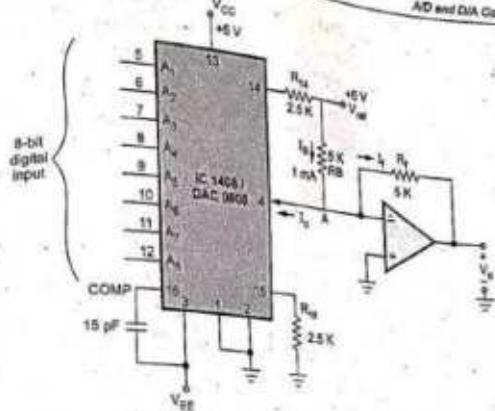


Fig. 6.3.3 Interfacing DAC in the bipolar range

- Fig. 6.3.3 shows the circuit for giving output in the bipolar range. Here, resistor  $R_B$  (5 K) is connected between  $V_{ref}$  and the output terminal of IC 1408/DAC0808. This gives a constant current source of 1 mA.

#### Operation

- Condition 1 :** For binary input 0000 0000 (00H)

When binary input is 00H, the output current  $I_o$  at pin 4 is zero. Due to this current flowing through  $R_B$  (1 mA) flows through  $R_t$  giving  $V_o = -5 \text{ V}$ .

- Condition 2 :** For binary input 1000 0000 (80H).

When binary input is 80H, the output current  $I_o$  at pin 4 is 1 mA. By applying KCL at node A we get,

$$-I_B + I_o + I_f = 0$$

Substituting values of  $I_B$  and  $I_o$  we get,

$$-(1 \text{ mA}) + (1 \text{ mA}) + I_f = 0$$

$$I_f = 0$$

and therefore the output voltage is zero.

**Condition 3 :** For binary input 1111 1111 (FFH).

When binary input is FFH, the output current  $I_o$  at pin 4 is 2 mA. By applying KCL at node A we get,

$$-I_B + I_o + I_f = 0$$

Substituting values of  $I_B$  and  $I_o$  we get,

$$-(1 \text{ mA}) + (2 \text{ mA}) + I_f = 0$$

$$I_f = -1 \text{ mA}$$

Therefore, the output voltage is + 5 V. In this way, circuit shown in the Fig. 6.3.2 gives output in the bipolar range.

**Important electrical characteristics for IC 1408/DAC0808**

- Reference current : 2 mA
- Supply voltage : + 5 V  $V_{CC}$  and - 15 V  $V_{EE}$
- Setting time : 300 ns
- Full scale output current : 1.992 mA
- Accuracy : 0.19 %

**Review Questions**

1. Interface a typical 8 bit DAC with 8255.
2. Draw and explain the typical interfacing circuit for DAC 1408/0808.
3. Draw and explain the typical interfacing circuit for DAC 1408/0808 in the bipolar range.

**6.4 Applications of DAC**

- DACs are used with a Digital Signal Processor (DSP) to convert a single into analog for transmission in the mixer circuit.
- They are used in waveform generators.
- They are used in ADC converters.

**Review Question**

1. State the applications of DAC.

**6.5 Sample and Hold Circuit**

- The sample and hold (S/H) circuit samples the value of the input signal in response to a sampling command and hold it at the output until arrival of the next command.

- It samples an analog input voltage in a very short period, generally in the range of 1 to 10  $\mu\text{s}$  and holds the sampled voltage level for an extended period, which can range from a few millisecond to several seconds.
- The Fig. 6.5.1 shows input and output response of the sample and hold circuit.

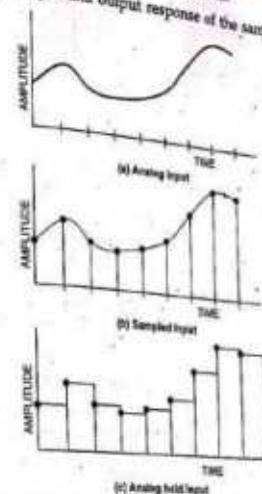


Fig. 6.5.1 Input and output response of sample and hold circuit

- The sample and hold circuit uses basic components analog switch and capacitor.
- The Fig. 6.5.2 shows the basic sample and hold circuit.

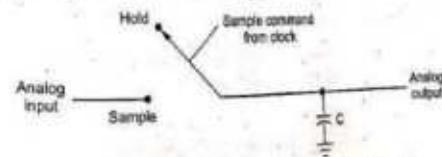


Fig. 6.5.2 Principle diagram for sample and hold circuit

- The circuit tracks the analog signal until the sample command causes the digital switch to isolate the capacitor from the signal and the capacitor holds this analog voltage during A/D conversion.

### 6.5.1 Basic Sample and Hold Circuit

- Fig. 6.5.3 shows the basic sample and hold circuit. Here JFET is used as a switch.

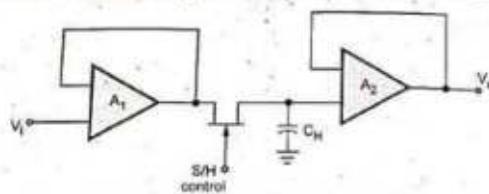


Fig. 6.5.3 Basic sample and hold circuits

- During the sampling time the JFET switch is turned on, and the holding capacitor charges up to the level of the analog input voltage.
- At the end of this short sampling period, the JFET switch is turned off. This isolates the holding capacitor  $C_H$  from the input signal.
- As a result, the voltage across capacitor  $C_H$  and hence the output voltage will remain essentially constant at the value of the input voltage at the end of the sampling time.
- However, there will be a small drop-off or drop of the capacitor voltage during the hold period due to the various leakage currents. To avoid this, input and output buffers (voltage follower) circuits are used.

### 6.5.2 Performance Parameters of S/H Circuits

- The S/H circuit performance can be characterised by specifications of the ordinary amplifier such as input offset voltage, gain error, nonlinearity etc.
- However, during the transitions from one mode to other (sample to hold or hold to sample), as well as in hold mode, circuit performance is characterised by specifications peculiar to S/H circuits.
- These specifications are explained below with the help of S/H circuit waveform, shown in Fig. 6.5.4. (See Fig. 6.5.4 on next page)

#### 1. Acquisition time ( $t_{ac}$ )

- It is the time required for the holding capacitor  $C_H$  to charge up to a level close to the input voltage during sampling. It depends on three factors :
  - RC time constant
  - Maximum output current of op-amp
  - Slew rate of op-amp

#### 2. Aperture time ( $t_{ap}$ )

- Because of propagation delays through the driver and switch,  $V_o$  will keep tracking  $V_i$  some time after the inception of the hold command. This is the aperture time.
- To get the precise timing, it is necessary to advance hold command by this amount.

#### 3. Aperture uncertainty ( $\Delta t_{ap}$ )

- It is the variation in aperture time from sample to sample. Due to aperture uncertainty it is difficult to compensate aperture time by advancing hold command.

#### 4. Hold mode settling time ( $t_s$ )

- After the application of hold command, it takes a certain amount of time for  $V_o$  to settle within a specified error band, such as 1%, 0.1% or 0.01%. This is the hold mode settling time.

#### 5. Hold step

- Because of the parasitic switch capacitances, at the time of switching between sample to hold mode, there is an unwanted transfer of charge between the switch driver and  $C_H$ .
- This changes the capacitor voltage and hence the output voltage, as shown in the Fig. 6.5.4.
- This change in output voltage are referred to as hold step, pedestal error and sample to hold offset.

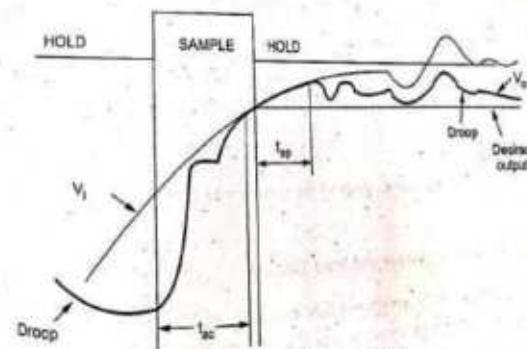


Fig. 6.5.4 S/H circuit waveforms

- The change in the output voltage can be related as given below,

$$\Delta V_o = \frac{\Delta Q}{C_H}$$

— (6.5.1)

#### 6. Feedthrough

- In the hold mode, because of stray capacitance across the switch, there is a small amount of ac coupling between  $V_o$  and  $V_i$ .
- This a.c. coupling causes output voltage to vary with variation in the input voltage.
- This is referred to as feedthrough and it is given as,

$$\Delta V_o \approx \frac{C_{ds}}{C_H} \Delta V_i$$

— (6.5.2)

where  $C_{ds}$  is the stray capacitance between drain and source of JFET.

- Feedthrough is usually expressed in terms of the Feedthrough Rejection Ratio (FRR) and it is given as,

$$FRR = 20 \log_{10} \frac{\Delta V_i}{\Delta V_o}$$

— (6.5.3)

**Example 6.5.1** Calculate the change in the output voltage if input changes by 5 V with FRR for S/H circuit is 80 dB.

Solution :  $FRR = 20 \log_{10} \frac{\Delta V_i}{\Delta V_o}$

$$80 = 20 \log_{10} \frac{5}{\Delta V_o} \quad \text{i.e. } \Delta V_o = 0.5 \text{ mV}$$

#### 7. Voltage droop

- The leakage current causes voltage of the capacitor to drop down. This is referred to as droop.

#### 6.5.3 Advantages of Sample and Hold Circuits

- The primary use of the sample and hold circuit to hold the sampled analog input voltage constant during conversion time of A/D converter.

- In case of multichannel ADCs, synchronization can be achieved by sampling signals from all channels at the same time.
- It also reduces the crosstalk in the multiplexer.

#### 6.5.4 Applications of Sample and Hold circuits

- The applications of such sample and hold circuit are :
  - Digital interfacing.
  - Analog to digital converter circuits.
  - Pulse modulation systems.
  - In storage of outputs of a multiplexer between updates in data distribution systems.
  - In reset-stabilised op-amps.
  - In analog demultiplexers.

#### Review Questions

- What is sample and hold circuit ? Explain its operation with the help of circuit diagram.
- Define : i) Acquisition time ii) Aperture time.
- List and define the performance parameters of a sample and hold circuit.
- State the advantages and applications of sample and hold circuit.

#### 6.6 A/D Converters

- The A/D conversion is a quantizing process whereby an analog signal is converted into equivalent binary word.
- The A/D converter is exactly opposite function that of the D/A converter.
- Fig. 6.6.1 shows symbol for A/D converter.

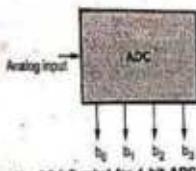


Fig. 6.6.1 Symbol for 4 bit ADC

#### Review Question

- What is A/D converter ?

### 6.7 Performance Parameters (Specifications of ADC)

GTU : Winter-17, 18, Summer-18

- The Fig. 6.7.1 shows the digital output of an ideal 3 bit ADC plotted against the analog input voltage.

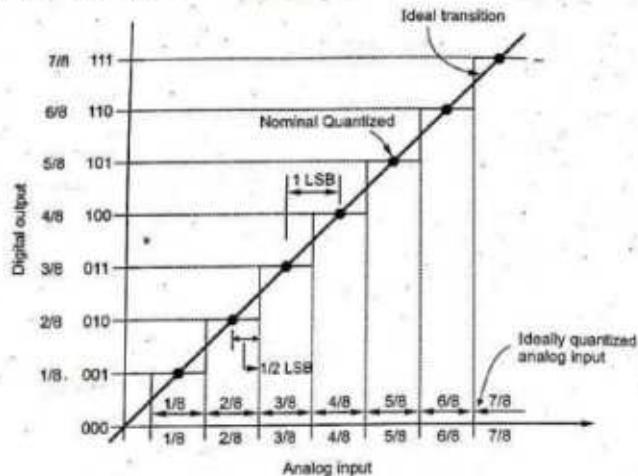


Fig. 6.7.1 Analog Input Vs digital output

#### Resolution

- Resolution is also defined as the ratio of a change in value of input voltage,  $V_i$ , needed to cause the digital output by 1 LSB. If the full scale input voltage required to cause a digital output of all 1's is  $V_{FS}$ , then resolution can be given as,

$$\text{Resolution} = \frac{V_{FS}}{2^n - 1} \quad (6.7.1)$$

#### Quantization Error and Accuracy

- Fig. 6.7.1 shows that the binary output is 011 for all values of  $V_i$  between  $\frac{1}{4}V$  and  $\frac{3}{4}V$ . There is an unavoidable uncertainty about the exact value of  $V_i$  when the output is 011. This uncertainty is specified as quantization error. Its value is  $\pm \frac{1}{2}$  LSB.

- It is given as,

$$Q_E = \frac{V_{FS}}{(2^n - 1)2} \quad (6.7.2)$$

- Increasing the number of bits results in a finer resolution, a smaller quantization error and greater the accuracy. The quantization error can be observed by continuously sampling a time-varying analog signal with an ADC, converting it back to analog with a DAC, and taking the difference between the two. The resulting sawtooth-like signal, called quantization noise.
- The root mean square value of such signal,  $E_n$  can be given as,

$$E_n = \frac{V_{FS}}{2^n \sqrt{12}}$$

- This is related to the resolution of the system. For each additional bit of resolution  $E_n$  is cut in half, that is reduced by 6 dB.

#### Gain and Offset Errors

- The offset error is the difference between the actual and ideal first transition voltages. The term offset; however, implies that all conversions are off by an equal amount. On the other hand, the gain error is the difference between the actual full scale transition voltage and the ideal full-scale transition voltage. This is illustrated in Fig. 6.7.2.



#### Gain and Offset Drifts

- The gain drift is defined as a change in the full-scale transition voltage measured over the entire operating temperature range. It is usually expressed in parts per million per degree Celsius (ppm/°C).
- The offset drift is defined as a change due to temperature in the analog zero for an A/D converter operating in bipolar mode. It is also expressed in parts per million per degree Celsius (ppm/°C).

#### Sampling Frequency and Aliasing Phenomenon

- According to sampling theorem, the minimum rate at which the analog signal should be sampled twice the highest frequency in the analog signal. Thus

$$f_s \geq 2f_{max}$$

- With this frequency, it is possible to reproduce analog signal faithfully by using a suitable interpolation algorithm.
- If the sampling frequency is less than the Nyquist rate ( $2 f_{\max}$ ), then the faithful reproduction of original signal is not possible. However, in such cases, spurious signals called aliases are produced. The frequency of aliased signal is the difference between the signal frequency and the sampling frequency. This problem is called aliasing and it is removed in all practical A/D converters by using anti-aliasing filters.

#### Non Linearity or Integral Non-Linearity (INL)

- It is a measure of the maximum deviation of the actual ADC transfer function from a straight line drawn through the first and last code transition after correction for offset and gain errors. It gives a more pessimistic but useful estimation of the non-linearity than referring the linearity of the ADC characteristics to an arbitrary best fit curve drawn through the output codes.

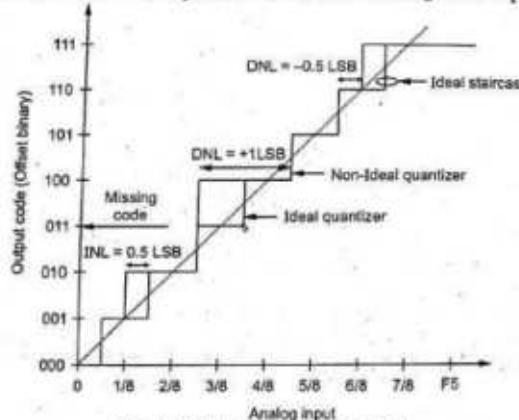


Fig. 6.7.3 INL and DNL analog input

#### Differential Non-Linearity (DNL)

- Differential Non-Linearity (DNL) is the maximum of the difference in the each conversion's Current Code Width (CCW) and the Ideal Code Width (ICW).
- DNL is the most critical of the measures of an ADC's performance for many control applications since it represents the ADC's ability to relate a small change in input voltage to the correct change in code conversion. DNL is defined as :

$$\text{Code DNL} = \text{CCW} - \text{ICW}$$

$$\text{DNL} = \text{Max}(\text{Code DNL})$$

#### Conversion Time (Settling Time)

- It is an important parameter for ADC. It is defined as the total time required to convert an analog signal into its digital output. It depends on the conversion technique used and the propagation delay of circuit components.

#### Aperture and Acquisition Times

- For accurate analog to digital conversion the analog input voltage should be held constant during the conversion cycle. If the analog input voltage changes by more than  $\pm 1/2$  LSB an error can occur in the digital output code. To minimize the occurrence of these errors it is necessary to hold the value of the analog input voltage constant during the conversion process. The sample and hold circuit does this task. Aperture and acquisition times are the specifications of sample and hold circuits.
- These specifications are explained below with the help of S/H circuit waveform, shown in Fig. 6.7.4.

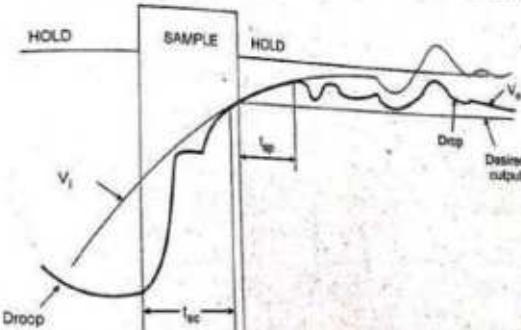


Fig. 6.7.4 S/H circuit waveforms

- Acquisition time ( $t_{ac}$ ) : It is the time required for the holding capacitor  $C_H$  to charge up to a level close to the input voltage during sampling. It depends on three factors :

- RC time constant
  - Maximum output current of op-amp
  - Slew rate of op-amp
- Aperture time ( $t_{ap}$ ) : Because of propagation delays through the driver and switch.  $V_o$  will keep tracking  $V_i$  some time after the inception of the hold command. This is the aperture time. To get the precise timing, it is necessary to advance hold command by this amount.

3. Aperture uncertainty ( $\Delta t_{ap}$ ) : It is the variation in aperture time from sample to sample. Due to aperture uncertainty it is difficult to compensate aperture time by advancing hold command.

#### Code Width

- The code width is the change in input voltage that occurs between the output code transitions expressed in LSBs of full scale. Code width uncertainty is the dynamic variation or jitter in the code width causing noise.

Note : The specifications discussed above are to be considered for selection of ADC.

**Example 6.7.1** An 8-bit ADC outputs all 1's when  $V_i = 5.1 \text{ V}$ . Find its a) Resolution and b) Digital output when  $V_i = 1.28 \text{ V}$ .

Solution :

From equation (6.7.1) we have,

$$\text{Resolution} = \frac{5.1 \text{ V}}{2^8 - 1} = 20 \text{ mV/LSB}$$

Therefore, we can say that to change output by 1 LSB we have to change input by 20 mV.

b) For 1.28 V analog input, digital output can be calculated as,

$$D = \frac{1.28 \text{ V}}{20 \text{ mV / LSB}} = 64 \text{ LSBs}$$

The binary equivalent of 64 is  $0100\ 0000_2$

**Example 6.7.2** Calculate the quantizing error for 12-bit ADC with full scale input voltage  $4.095 \text{ V}$ .

Solution : From equation (6.7.2) we get

$$Q_E = \frac{4.095}{(2^{12} - 1) \times 2} = \frac{4.095}{(4096 - 1) \times 2} = 0.5 \text{ mV}$$

#### Review Questions

- Explain the performance parameters of ADC.
- Explain in detail specifications of ADCs.
- State the specifications and errors associated with ADC. Also state the applications of ADC.
- List various specifications of ADC.
- Explain resolution and quantization error in reference to ADC. **GTU : Winter-17, Marks 4**
- Give the points to be considered for selection of ADC. **GTU : Summer-18, Marks 3**
- Define quantization error for ADC. **GTU : Summer-18, Marks 2**
- Define following specification of ADC (i) Resolution (ii) Conversion time (iii) Quantization error. **GTU : Winter-18, Marks 3**

#### 6.8 Types of A/D Converters

- GTU : Winter-16, Summer-15, 16, 18
- Analog to digital converters are classified into two general groups based on the conversion techniques.
  - One technique involves comparing a given analog signal with the internally generated reference voltages. This group includes successive approximation and flash type converters.
  - The another technique involves changing an analog signal into time or frequency integrator converters and voltage-to-frequency converters. This group includes

##### 6.8.1 Dual Slope ADC

- Dual slope conversion is an indirect method for A/D conversion where an analog voltage and a reference voltage are converted into time periods by an integrator and then measured by a counter. The speed of this conversion is slow but the accuracy is high.
- Fig. 6.8.1 shows a typical dual slope converter circuit. It consists of integrator (ramp generator), comparator, binary counter, output latch and reference voltage.
- The ramp generator input is switched between the analog input voltage  $V_i$  and a negative reference voltage,  $-V_{REF}$ .
- The analog switch is controlled by the MSB of the counter. When the MSB is a logic 0, the voltage being measured is connected to the ramp generator input. When MSB is logic 1, the negative reference voltage is connected to the ramp generator.

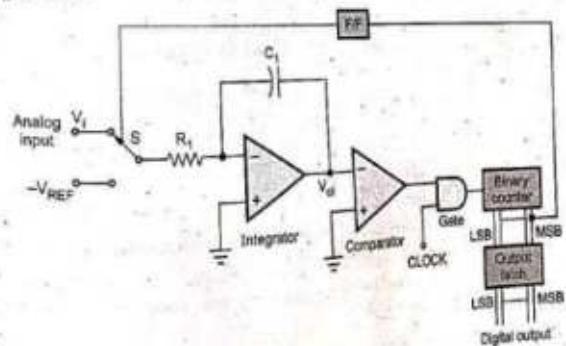


Fig. 6.8.1 Dual slope ADC converter

- At time  $t = 0$ , analog switch S is connected to the analog input voltage  $V_i$ , so that the analog input voltage integration begins. The output voltage of the integrator can be given as,

$$V_{oi} = \frac{-1}{R_1 C_1} \int_0^t V_i dt = \frac{-V_i t}{R_1 C_1}$$

where  $R_1 C_1$  is the integrator time constant and  $V_i$  is assumed constant over the integration time period.

- At the end of  $2^N$  clock periods MSB of the counter goes high. As a result the output of the flip-flop goes high, which causes analog switch S to be switched from  $V_i$  to  $-V_R$ . At this very same time the binary counter which has gone through its entire count sequence is reset.
- The negative input voltage ( $-V_R$ ) connected to the input of integrator causes the integrator output to ramp positive. When integrator output reaches zero, the comparator output voltage goes low, which disables the clock AND gate. This stops the clock pulses reaching the counter, so that the counter will be stopped at a count corresponding to the number of clock pulses in time  $t_2$ .
- The integrator output ramp down to a voltage  $V$  and get back upto 0. Therefore, the charge voltage is equal to discharge voltage and we can write,

$$\frac{V_i t_1}{R_1 C_1} = \frac{V_R t_2}{R_1 C_1}$$

$$V_i t_1 = V_R t_2$$

$$t_2 = \frac{V_i t_1}{V_R}$$

- The above equation shows that  $t_2$  is directly proportional only to the  $V_i$  since  $V_R$  and  $t_1$  are constants. The binary digital output of the counter gives corresponding digital value for time period  $t_2$  and hence it is also directly proportional to input signal  $V_i$ .
- The actual conversion of analog voltage  $V_{in}$  into a digital count occurs during  $t_2$ . The control circuit connects the clock to the counter at the beginning of  $t_2$ .

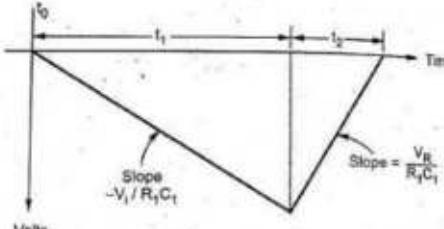


Fig. 6.8.2 Integrator output voltage

clock is disconnected at the end of  $t_2$ . Thus the counter contents is digital output. Hence we can write,

$$\text{Digital output} = \left( \frac{\text{counts}}{\text{second}} \right) t_2$$

But from equation of  $t_2$  we can write,

$$\text{Digital output} = \left( \frac{\text{counts}}{\text{second}} \right) t_1 \left( \frac{V_i}{V_R} \right)$$

The counter output can then be connected to an appropriate digital display.

- The advantages of dual slope ADC are
  - i) It is highly accurate.
  - ii) Its cost is low.
  - iii) It is immune to temperature caused variations in  $R_1$  and  $C_1$ .
- The only disadvantage of this ADC is its speed which is low.

**Example 6.8.1** For a particular dual slope ADC,  $t_1 = 83.33$  ms and the reference voltage is 100 mV. Calculate  $t_2$  if i)  $V_i = 100$  mV and ii) 200 mV.

Solution : We know that,

$$t_2 = \left( \frac{V_i}{V_R} \right) t_1$$

$$\text{i)} \quad t_2 = \left( \frac{100}{100} \right) (83.33) = 83.33 \text{ ms}$$

$$\text{ii)} \quad V_i = 200 \text{ mV}$$

$$\therefore \quad t_2 = \left( \frac{200}{100} \right) (83.33) = 166.6 \text{ ms}$$

**Example 6.8.2** Find the digital output of an ADC having  $t_1 = 83.33$  ms and  $V_R = 100$  mV for an input voltage of +100 mV. The clock frequency is 12 kHz.

Solution : The digital output is given as,

$$\text{Digital output} = \left( \frac{\text{counts}}{\text{second}} \right) t_1 \left( \frac{V_i}{V_R} \right)$$

Now clock frequency = 12 kHz

$$\text{i.e. } = 12000 \frac{\text{counts}}{\text{second}}$$

$$\text{Digital output} = 12000 \times 83.33 \times \left( \frac{100}{100} \right) \times 10^{-3} = 1000 \text{ counts}$$

### 6.8.2 Successive Approximation ADC

- In this technique, the basic idea is to adjust the DAC's input code such that its output is within  $\pm \frac{1}{2}$  LSB of the analog input  $V_i$  to be A/D converted. The code that achieves this represents the desired ADC output.
- The successive approximation method uses very efficient code searching strategy called binary search. It completes searching process for n-bit conversion in just n clock periods.
- Fig. 6.8.3 shows the block diagram of successive approximation A/D converter. It consists of a DAC, a comparator and a successive approximation register (SAR).

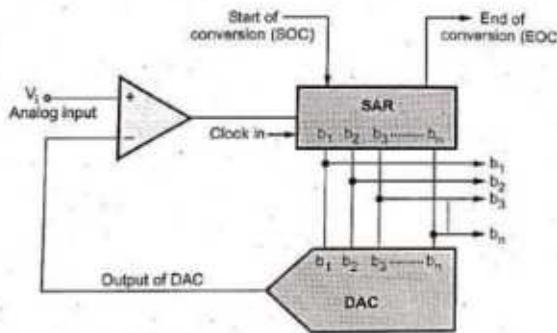


Fig. 6.8.3 Block diagram of successive approximation A/D converter

- The external clock input sets the internal timing parameters. The control signal start of conversion (SOC) initiates an A/D conversion process and end of conversion signal is activated when the conversion is completed.

#### Operation :

- The searching code process in successive approximation method is similar to weighing an unknown material with a balance scale and a set of standard weights.
- Let us assume that we have 1 kg, 2 kg and 4 kg weights (SAR) plus a balance scale (comparator and DAC). Now we will see the successive approximation analogy for 3-bit ADC.

- Refer Fig. 6.8.3 and 6.8.4. The analog voltage  $V_i$  is applied at one input of comparator. On receiving start of conversion signal (SOC) successive approximation register sets 3-bit binary code  $100_2$  ( $b_2 = 1$ ) as an input of DAC. This is similar process of placing the unknown weight on one platform of the balance and 4 kg

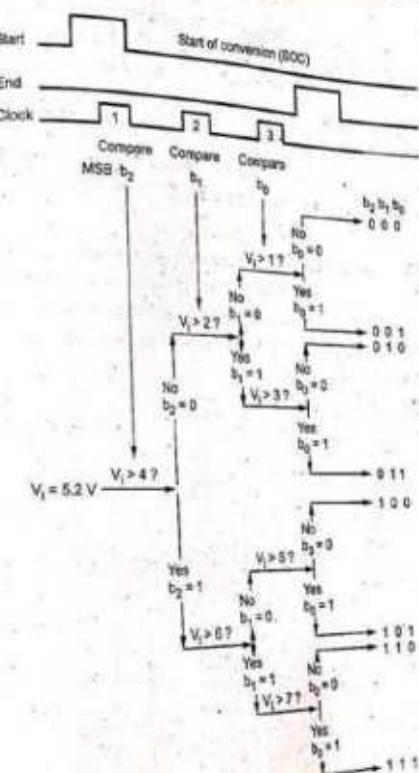


Fig. 6.8.4 Illustration of conversion process

- The DAC converts the digital word 100 and applies its equivalent analog output at the second input of the comparator.
- The comparator then compares two voltages just like comparing unknown weight with 4 kg weight with the help of balance scale.
- If the input voltage is greater than the analog output of DAC, successive approximation register keeps  $b_2 = 1$  and makes  $b_1 = 1$  (addition of 2 kg weight to have total 6 kg weight) otherwise it resets  $b_2 = 0$  and makes  $b_1 = 1$  (replacing 2 kg weight). The same process is repeated for  $b_1$  and  $b_0$ . The status of  $b_0$ ,  $b_1$  and  $b_2$  bits gives the digital equivalent of the analog input.
- Fig. 6.8.4 illustrates the process we have just discussed. The dark lines in the Fig. 6.8.4 shows setting and resetting actions of bits for input voltage 5.2 V, on the basis of comparison.
- It can be seen from the Fig. 6.8.4 that one clock pulse is required for the successive approximation register to compare each bit. However an additional clock pulse is usually required to reset the register prior to performing a conversion. The time for one analog to digital conversion must depend on both the clock's period  $T$  and number of bits  $n$ . It is given as,

$$T_c = T(n+1)$$

where

 $T_c$  = Conversion time $T$  = Clock period $n$  = Number of bits

**Example 6.8.3** An 8-bit successive approximation ADC is driven by a 1 MHz clock. Find its conversion time.

**Solution :**  $f = 1 \text{ MHz}$

$$\begin{aligned} T &= \frac{1}{f} \\ &= \frac{1}{1 \times 10^6} = 1 \mu\text{sec} \end{aligned}$$

$$n = 8$$

$$T_c = T(n+1) = 1(8+1) = 9 \mu\text{sec}$$

### 6.8.3 Flash ADC (Parallel Comparators)

When system designs call for the highest speed available, flash-type A/D converters (ADCs) are likely to be the right choice. They get their names from their ability to do the conversion very rapidly. Flash A/D converter, also known as a simultaneous or parallel comparator ADC, because the fast conversion speed is accomplished by providing  $2^n - 1$  comparators and simultaneously comparing the input signal with unique reference levels spaced 1 LSB apart.

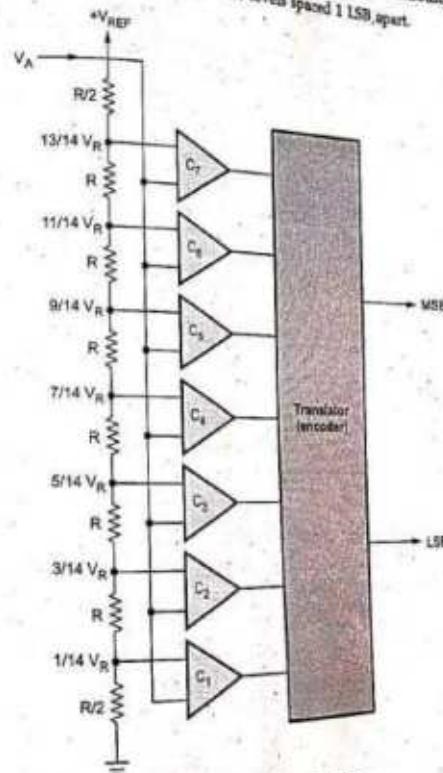


Fig. 6.8.5 3-bit flash converter

- Fig. 6.8.5 shows 3-bit flash A/D converter. For this ADC, seven ( $2^3 - 1$ ) comparators are required. As shown in the Fig. 6.8.5, one input of each comparator is connected to the input signal and other input to the reference voltage level generated by the reference voltage divider. The reference voltage ( $V_{REF}$ ) is equal to the full scale input signal voltage. The manner in which the flash A/D converter performs a quantization is relatively simple.
- The comparators give output "1" or "0" state depending on whether the input signal is above or below the reference level at that instant. Those comparators referred above the input signal, remain turned-off, representing a "0" state. The comparators at or below the input signal conversely become a "1" state. The code resulting from this comparator is converted to a binary code by the encoder.
- The number of comparators required for  $n$ -bit resolution is,

$$\text{Number of comparators} = 2^n - 1$$

- As seen earlier the quantization error is  $\pm \frac{1}{2}$  LSB. Thus for an ADC, the maximum frequency for a sine wave  $V_{in}$  to be digitized within an accuracy of  $\pm \frac{1}{2}$  LSB is,

$$f_{max} = \frac{1}{2\pi(T_c)2^n}$$

where  $f_{max}$  = Maximum input frequency

$T_c$  = Conversion time

$n$  = Number of bits

**Example 6.8.4** For a particular 8-bit ADC, the conversion time is 9  $\mu$ s. Find the maximum frequency of an input sine wave that can be digitized.

**Solution :** The maximum frequency is given by,

$$f_{max} = \frac{1}{2\pi(T_c)2^n} = \frac{1}{2\pi \times 9 \times 10^{-6} \times 2^8} = 69.07 \text{ Hz}$$

#### 6.8.4 Comparison between ADCs

- The comparison of ADCs is given in the tabular form as below.

Parameter	Flash (Parallel comparators)	Successive approximation	Dual slope
Speed	Fastest	Fast	Slow
Accuracy	Less	Medium	More

Resolution	Up to $2^n$	Up to $2^n$	Up to $2^n$	Up to $2^n$	Up to $2^n$	Up to $2^n$	Up to $2^n$
Input hold time	Very less	Very less	Depends on number of bits. If it is more than 8 bits, it is minimum, hence sample and hold circuit is required.	Depends on number of bits. If it is more than 8 bits, it is minimum, hence sample and hold circuit is required.	Depends on number of bits. If it is more than 8 bits, it is minimum, hence sample and hold circuit is required.	Depends on number of bits. If it is more than 8 bits, it is minimum, hence sample and hold circuit is required.	Depends on number of bits. If it is more than 8 bits, it is minimum, hence sample and hold circuit is required.
Cost	Very costly	Very costly	Medium	Medium	Medium	Medium	Medium
Applications	High speed fiber optic communication, digital storage oscilloscope, imaging and many more where high speed A/D conversion is required.	The successive approximation A/D converter has the disadvantage of requiring D/A converter, but it has the advantage of high speed with excellent resolution. Hence these are most popular and used in data acquisition systems.	These are used when high accuracy and resolution is required and speed is not the important criteria.	These are used when high accuracy and resolution is required and speed is not the important criteria.	These are used when high accuracy and resolution is required and speed is not the important criteria.	These are used when high accuracy and resolution is required and speed is not the important criteria.	These are used when high accuracy and resolution is required and speed is not the important criteria.
ICs	ADC0809KD AD90095D TM1578 TDC19193	ADC1011 ADC1011 ADC1011 ADC1012	10 10 10 10	100 KHz 175 KHz 100 KHz 25 KHz	100 KHz 100 KHz 100 KHz 25 KHz	10 10 10 10	25 Hz 30 Hz 25 Hz 25 Hz

#### 6.8.5 Counter Type A/D Converter

- This ADC uses DAC for A to D conversion. The output of the DAC is continuously compared with the analog input to the ADC which is to be converted into digital output. When the output of the DAC becomes greater than this analog input, the corresponding digital input to the DAC is noted which represents the analog input to the ADC.
- Fig. 6.8.6 shows the circuit diagram of counter type ADC.
- As shown in the Fig. 6.8.6, the counter type ADC consists of a binary counter, DAC, comparator and AND gate. The operation of the circuit is explained below.
  - Initially, the counter is reset, i.e. its output is set to zero by applying a reset pulse. The output of the counter is given as digital input to DAC. Since input to DAC is zero, its output  $V_D$  is zero.
  - When the analog input voltage  $V_A$  is applied to ADC, it becomes greater than  $V_D$ .  $V_A$  acts as input voltage for noninverting terminal and  $V_D$  acts as input voltage for inverting terminal of the comparator. Since  $V_A$  is greater than  $V_D$ , the comparator output goes high.

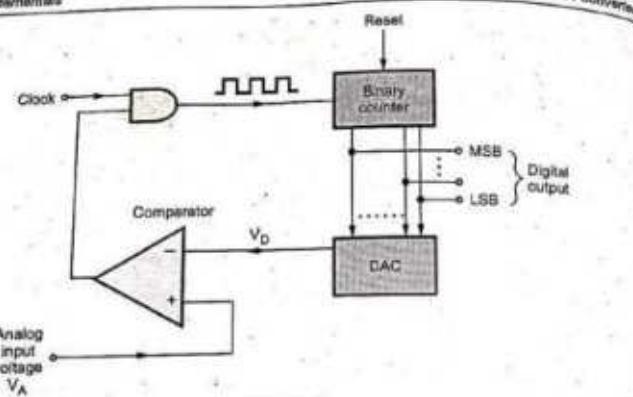


Fig. 6.8.6

- iii) For an AND gate, one input is clock pulses and another input is the output of the comparator. Because of the high output of the comparator, the clock pulses are allowed to pass through the AND gate.
- iv) The counter starts counting these clock pulses. According to the number of clock pulses, the output of the counter goes on increasing. This increases the output of the DAC.

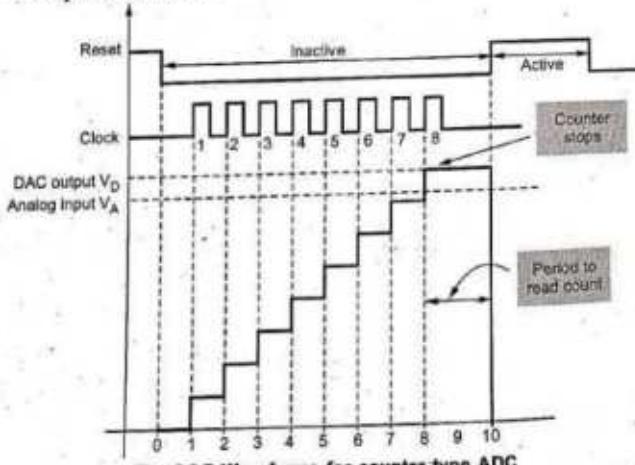


Fig. 6.8.7 Waveforms for counter type ADC.

TECHNICAL PUBLICATIONS™ - An up beat for knowledge

- v) The above steps are continued till  $V_D$  is less than  $V_A$ .
- vi) As soon as DAC output  $V_D$  becomes greater than  $V_A$ , the comparator output goes low. This disables AND gate. So the clock pulses are not allowed to pass through the AND gate. The counting process of the binary counter is stopped.
- vii) The output of the binary counter which is in digital form is noted which represents the digital equivalent of the analog input voltage  $V_A$ .
- viii) For the next A to D conversion, the input voltage to ADC,  $V_A$  changes. The binary counter is cleared by applying a second reset pulse and all the above steps are repeated to obtain the digital equivalent of  $V_A$ .

#### Disadvantages

- It is necessary to give enough time for DAC conversion and comparator to respond. Therefore, there is a limitation on the clock frequency. As clock frequency is low, the speed of conversion is less.
- Conversion time is not constant. It increases with increase in input voltage. In other words, we can say that conversion time is high at high input voltage.

#### 6.8.6 Tracking Type Continuous A/D Conversion

- The tracking type converter is similar to counter type converter; however in this converter the counter starts counting from the most recent code rather than restarting from zero. Such an arrangement increases conversion speed, if signal changes in the input signal are minor since the last conversion. Increase in conversion speed is achieved because fewer counts are required for the DAC output to catch up with  $V_i$  when input changes are minor.
- The tracking type converter is also called servo converter or continuous A/D converter. The Fig. 6.8.8 shows the tracking type ADC. It consists of up/down counter, a comparator, DAC and the control circuitry. Here, comparator output controls the direction of count. Counting will be up or down depending on whether the DAC output is below or above  $V_i$  i.e. comparator output is logic 0 or logic 1. Whenever, the DAC output crosses  $V_i$ , the comparator changes state and generates positive going or negative going edge at the output. This edge can be taken as EOC command, so that counter stops counting when comparator changes its state. The edge detection is achieved by using two D flip-flops. Flip-flop 1 is used to detect positive edge whereas flip-flop 2 is used to detect negative edge. Initially, with the activation of start of conversion (SOC) signal the outputs of both flip-flops are set to logic 1. Since, the D input of both the flip-flops is connected to logic 0, upon detection of the edge, the flip-flop output goes to logic 0 and disables the clock. As a result, the counter stops counting. When comparator output goes from logic 1 to logic 0, negative edge is generated and it resets

TECHNICAL PUBLICATIONS™ - An up beat for knowledge

flip-flop 2 whereas when comparitor output goes from logic 0 to logic 1, positive edge is generated and it resets flip-flop 1. When either of the two flip-flop gets reset, the clock is disabled.

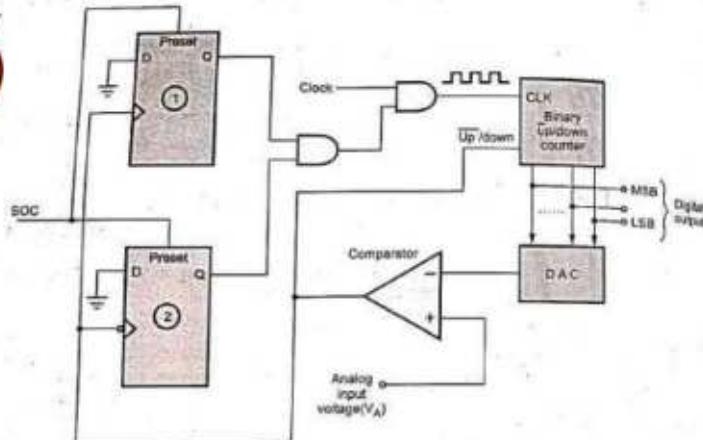


Fig. 6.8.8 Tracking type ADC

### **Review Questions**

2. Which are different methods for A/D conversion.
  2. Explain the dual slope A/D conversion technique with the help of block diagram.
  3. Explain the operation of flash A/D converter.
  4. Compare three A/D conversion techniques. GTU : Winter-16 Marks 4
  5. Write a note on counter type A/D converter.
  6. Explain the tracking type continuous A/D converter.
  7. Explain working of successive approximation type ADC. GTU : Summer-16 Marks 7
  8. Explain the working of analog to digital converter with necessary diagram. GTU : Summer-15 Marks 7
  9. Write down various ADC networks and explain any one in brief. What is the best ADC ? GTU : Summer-16 Marks 7

## A/D Converter using V/F Converter

- An A/D converter using the voltage to frequency is shown in Fig. 6.9.1. The voltage to frequency converter section consists of an integrator, a comparator and a monostable multivibrator. The analog input voltage is applied to the integrator whose output is applied to the inverting input terminal of the comparator.
  - The comparator output activates a monostable multivibrator to produce a short pulse to control the active switch S of the integrator. The comparator generates a pulse train and whose frequency  $F$  can be derived as
$$F = \frac{1}{RC} \left( \frac{V_s}{V_r} \right)$$
  - The equation shows that comparator output signal frequency  $F$  is proportional to the analog input voltage  $V_s$ .
  - The output of the voltage to frequency converter is applied to the clock input of a N-bit binary counter through an AND gate. So we can apply the clock pulses to the binary counter for specific time period, say  $T_1$  and the counter will count the number of pulses for that duration. If the number of pulses recorded at counter in time  $T_1$  is  $n$ , then  $T_1 \times F = n$
$$n = \frac{1}{RC} \frac{V_s}{V_r} \cdot T_1$$
  - From above expression we can observe that number of pulses counted in the counter is proportional to the analog input voltage and thus we obtain the equivalent digital form of the analog voltage.

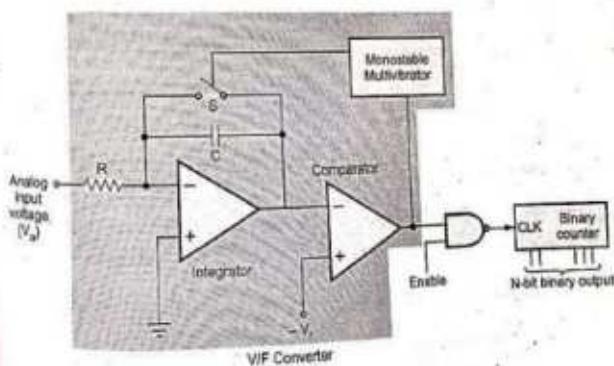


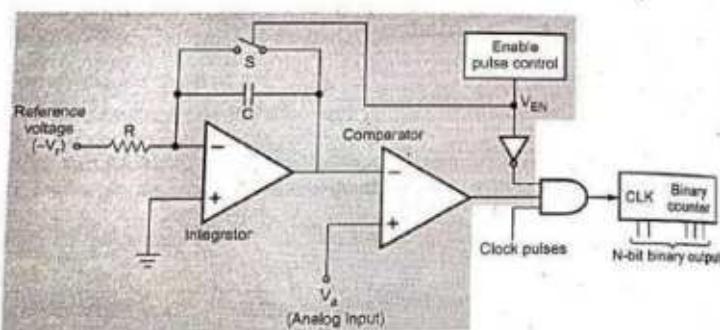
Fig. 6.9.1 A/D converter using V/F converter

**Review Question**

1. Explain the operation of A/D converter using V/F converter.

**6.10 A/D Converter using V/T Conversion**

- The concept of A/D converter using voltage to time conversion is very much similar to the A/D converter with voltage to frequency conversion, as the digital output for both the types of converter is derived by counting the number of pulses that is proportional to the analog input voltage. The difference is that on A/D converter using V/F conversion is based on the counting of pulses of variable frequency for fixed time duration, whereas the A/D converter using voltage to time conversion counts the pulses of fixed frequency but of variable time.
- Fig. 6.10.1 shows the schematic diagram of the A/D converter using voltage to time conversion.



**Fig. 6.10.1 A/D converter using voltage to time conversion**

- The comparator output is applied to one of the inputs of a 3-input AND gate, where other two inputs of AND gates are complement of  $V_{EN}$  and clock pulses of time period  $T_c$ . As long as comparator output is high and  $V_{EN}$  is low, the clock pulses are applied to the binary counter for count operation.
- If the counter records the counting of  $n$  number of pulses with pulse time period is  $T_c$ , then  $T = nT_c$ .

$$\text{where } T = \frac{V_a RC}{V_r}$$

$$n = \frac{V_a RC}{V_r T_c}$$

- From this expression we can observe that the counter reading is proportional to the analog input voltage.

**Review Question**

1. Explain the working of A/D converter using voltage to time conversion.

**6.11 Example of A/D Converter IC**

- The ADC 0808 and ADC 0809 are monolithic CMOS devices with an 8-channel multiplexer.
- These devices are also designed to operate from common microprocessor control buses, with tri-state output latches driving the data bus.

**6.11.1 Features**

- 8-bit successive approximation ADC.
- 8-channel multiplexer with address logic.
- Conversion time 100  $\mu$ s.
- It eliminates the need for external zero and full-scale adjustments.
- Easy to interface to all microprocessors.
- It operates on single 5 V power supply.
- Output meet TTL voltage level specifications.

**6.11.2 Pin Diagram**

Fig. 6.11.1 shows pin diagram of 0808/0809 ADC. (See Fig. 6.11.1 on next page)

**6.11.3 Operation**

- ADC 0808/0809 has eight input channels, so to select desired input channel, it is necessary to send 3-bit address on A, B and C inputs.
- The address of the desired channel is sent to the multiplexer address inputs through port pins. After atleast 50 ns, this address must be latched. This can be achieved by sending ALE signal. After another 2.5  $\mu$ s, the start of conversion (SOC) signal must be sent high and then low to start the conversion process. To indicate end of conversion ADC 0808/0809 activates EOC signal.
- The microprocessor system can read converted digital word through data bus by enabling the output enable signal after EOC is activated. This is illustrated in Fig. 6.11.2.

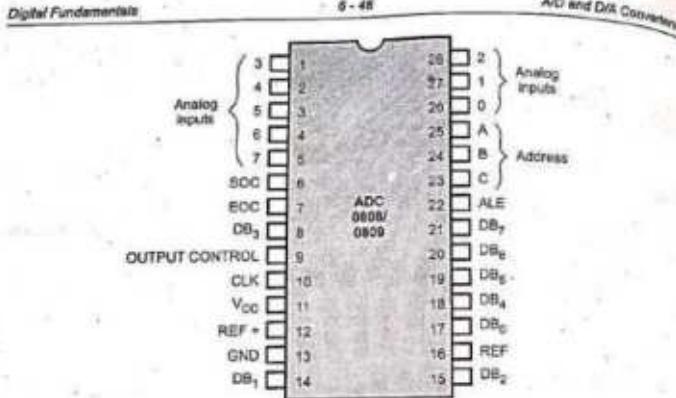


Fig. 6.11.1 Pin diagram of 0808/0909

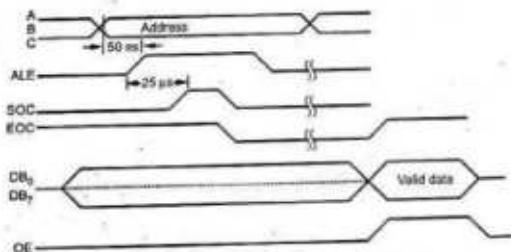


Fig. 6.11.2 Timing waveforms for ADC 0808

#### 6.11.4 Interfacing

- Fig. 6.11.3 shows typical interfacing circuit for ADC 0808 with microprocessor system.
  - The zener diode and LM 308 amplifier circuitry is used to produce a  $V_{CC}$  an  $+V_{REF}$  of 5.12 V for the A/D converter. With this reference voltage the A/D converter will have 256 steps of 20 mV each.

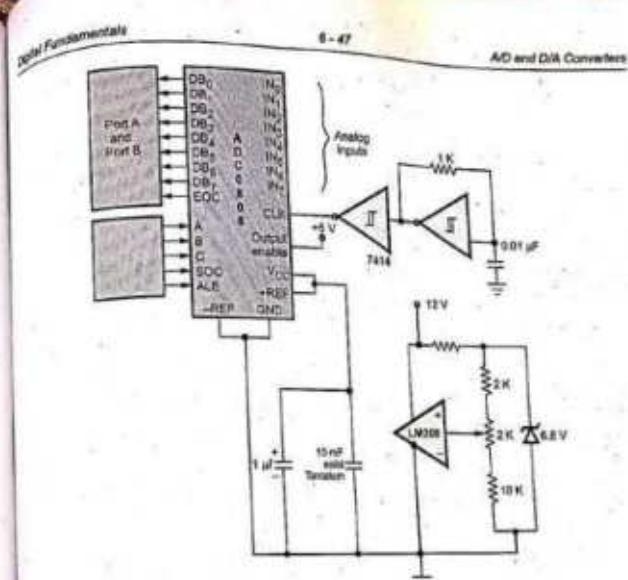


Fig. 6.11.3 Typical interface for 0808/0809

**Example 6.11.1** An 8-bit digital ramp ADC with a 40 mV resolution uses a clock frequency of 2.5 MHz. A compensator with  $V_o = 1 \text{ mV}$ . Determine the following values.

- i) The digital output for  $V_{in} = 6.000 \text{ V}$ .  
ii) The digital output for  $V_{in} = 6.035 \text{ V}$ .

Solution : The digital output is given by,

$$\text{Count} = \frac{V_{in}}{\text{Resolution}}$$

$V_s = 5000$  V

$$\text{Count} = \frac{6,000}{10 \times 10^{-3}} = 150$$

$$V = 6015 \text{ V}$$

$$\therefore \text{Count} = \frac{6.035}{40 \times 10^{-3}} \approx 150$$

$$\text{III} \quad (T_e)_{\text{max}} = (\text{Maximum count}) \times T \\ = (2^8 - 1) \times \frac{1}{f} = \frac{(255)}{2.5 \times 10^6} = 102 \mu\text{s}$$

$$(T_e)_{\text{average}} = \frac{(T_e)_{\text{max}}}{2} = 51 \mu\text{s}$$

**Example 6.11.2** Use successive approximation type ADC to convert the analog voltage 7.28 V to 8-bit binary. If  $V_R = 10$  V, determine the final binary answer and the error present. Show the timing waveforms that would occur in the successive approximation ADC process.

Note :  $V_R$  = Full scale input voltage to the DAC

Solution : The reference voltage  $V_R$  is full scale input voltage to the DAC.

Hence the 10 V reference corresponds to  $2^n$  levels i.e.  $2^8 = 256$  levels.

Hence one level corresponds to,

$$= \frac{10}{256} = 0.03906 \text{ V}$$

This is nothing but the resolution of the ADC.

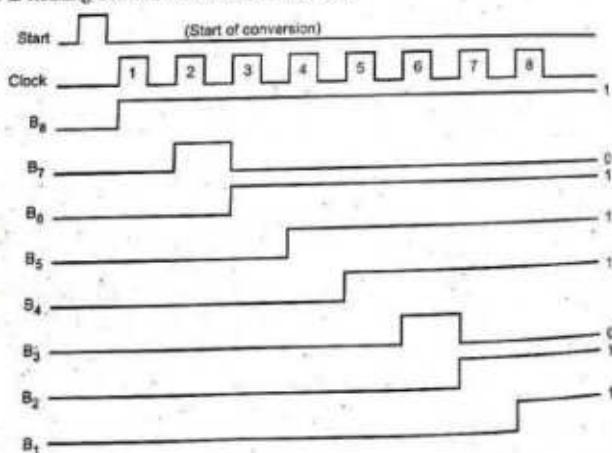


Fig. 6.11.4

The timing waveforms are shown in the Fig. 6.11.4.

Hence the final binary number is,

$$= (10111011)_2$$

Hence the total output count is,

$$= (1 \times 128) + (0 \times 64) + (1 \times 32) + (1 \times 16) + (1 \times 8) + (0 \times 4) + (1 \times 2) + (1 \times 1) \\ = 187$$

$$\text{Actual output} = \text{Count} \times \text{Resolution} = 187 \times 0.03906 = 7.3046 \text{ V}$$

$$\text{Exact value} = 7.28 \text{ V}$$

$$\text{Error} = 7.3046 - 7.28 = 0.024 \text{ V}$$

#### Review Questions

1. List the features of ADC 0808.
2. Explain the interfacing of ADC 0808 to the microprocessor.



## 7

**Semiconductor Memories and Programmable Logic Devices****Contents**

7.1	Introduction	Summer-18, .....	Marks 7
7.2	Characteristics of Memories		
7.3	Classification of Memories		
7.4	Serial (Sequential) Random Access Memories		
7.5	Read Only Memory (ROM)	Winter-14, 18, .....	Marks 4
7.6	Read and Write Memory (RAM)	Winter-14, .....	Marks 7
7.7	Memory Organization		
7.8	Expanding Memory Size		
7.9	Content Addressable Memory (CAM)		
7.10	Charge - Coupled Devices (CCD) Memory		
7.11	Programmable Logic Devices		
7.12	ROM / PROM as a PLD	Winter-15, 18, Summer-18, Marks 7	
7.13	Programmable Logic Array (PLA)	Dec-12, May-13, Winter-16 .....	Marks 7
7.14	Programmable Array Logic (PAL)		
7.15	Comparison between PROM, PLA and PAL		
7.16	Complex Programmable Logic Devices (CPLDs)		
7.17	Field Programmable Gate Array (FPGA)	Winter-15, .....	Marks 3
7.18	Comparison between CPLDs and FPGAs		

Oral Questions and Answers

(7 - 1)

## 7.1 Introduction

- Memories are made up of registers. Each register in the memory is one storage location also called **memory location**. Each memory location is identified by an **address**. The number of storage locations can vary from a few in some memories to hundreds of thousand in others. Each location can accommodate one or more bits. Generally, the total number of bits that a memory can store is its **capacity**. Most of the types the capacity is specified in terms of bytes (group of eight bits).
  - Each register consists of storage elements (flip-flops or capacitors in semiconductor memories and magnetic domain in magnetic storage), each of which stores one bit of data. A storage element is called a **cell**.
  - The data stored in a memory by a process called **writing** and are retrieved from the memory by a process called **reading**. Fig. 7.1.1 illustrates in a very simplified way the concept of write, read, address and storage capacity for a generalized memory.
  - As shown in the Fig. 7.1.1 a memory unit stores binary information in groups of bits called **words**. A word in memory is an entity of bits that moves in and out of storage as a unit. A word having group of 8-bits is called a **byte**. Most computer memories use words that are multiples of eight bits in length. Thus, a 16-bit word contains two bytes, and a 32-bit word is made of 4-bytes.

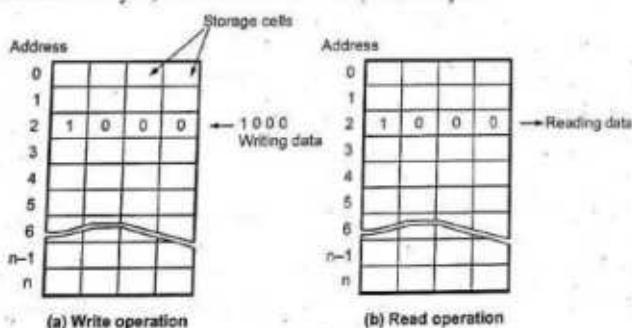


Fig. 7.1.1

- The communication between a memory and its environment is achieved through data lines, address selection lines, and control lines that specify the direction of transfer. The Fig. 7.1.2 shows the block diagram of memory unit. Then data lines provide the information to be stored in memory and the  $k$  address lines specify the particular word chosen among the many available. The two control inputs specify the direction transfer.

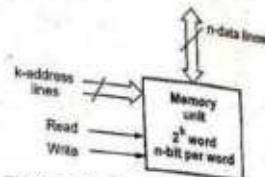


Fig. 7.1.2 Block diagram of  $\omega_{\text{ext}}$

When there are  $k$  address lines we can access  $2^k$  memory words. For example, if  $k = 10$  we can access  $2^{10} = 1024$  memory words.

### **Illustrative Examples**

**Example 7.1.1** A bipolar RAM chip is arranged as 16 words. How many bits are stored in the chip?

solution:  $16 \times 8 = 128$  bits  $\therefore$  one word = 8 bits

**Example 7.1.2** How many address bits are needed to provide a  $2^k \times 8$  ROM?

~~Section : 2K memory locations = 2048 locations~~

Since  $2^{11} = 2048$ , we need 11 address lines.

**Example 7.1.3** How many locations are addressed using 16 address bits?

station. The number of locations addressed will increase.

Solution: The number of locations addressed =  $2^{10} = 2048$

## 7.1.1 Computer Memory

Memory is internal storage areas in the computer system. It is used to store data and instructions. Computer memory is the storage space in the computer, where data is to be processed and instructions required for processing are stored. The memory is divided into large number of small parts called cells. Each location or cell has a unique address, which varies from zero to memory size minus one. For example, if the computer has 64K words, then this memory unit has  $64 \times 1024 = 65536$  memory locations. The address of these locations varies from 0 to 65535.

Memory is primarily of three types:

- Primary memory/Main memory
  - Cache memory

- Secondary memory
- Primary memory (main memory)**: Primary memory holds only those data and instructions on which the computer is currently working. It has a limited capacity and data is lost when power is switched off (i.e. it is volatile memory). It is generally made up of semiconductor device. These memories are not as fast as registers. The data and instruction required to be processed resides in the main memory. It is divided into two subcategories RAM and ROM.
- Cache memory**: Cache memory is a very high speed semiconductor memory which can speed up the CPU. It acts as a buffer between the CPU and the main memory. It is used to hold those parts of data and program which are most frequently used by the CPU. The parts of data and programs are transferred from the disk to cache memory by the operating system, from where the CPU can access them.
- Secondary memory**: This type of memory is also known as external memory or non-volatile. It is slower than the main memory. These are used for storing data/information permanently. CPU directly does not access these memories, instead they are accessed via input-output routines. The contents of secondary memories are first transferred to the main memory and then the CPU can access it. For example, disk, CD-ROM, DVD, etc.

**Review Questions**

- Define a memory cell. Give an example.
- Define a 'memory location' and a 'cell'.
- Write a descriptive note on memories.
- What is the memory capacity of random access memory if it has 10 bit address lines?
- Write a short note on memory.

GTU Summer-16, Marks 7

**7.2 Characteristics of Memories**

Location :	CPU
	Internal (main)
	External (Secondary)
Capacity :	Word size
	Number of words
Unit of transfer :	Word
	Block

<b>Access method :</b>	Sequential access Direct access Random access Associative access
<b>Performance :</b>	Access time, Cycle time, Transfer rate
<b>Physical type :</b>	Semiconductor Magnetic surface
<b>Physical characteristics :</b>	Volatile / non-volatile
<b>Organization</b>	Erasable / Non-erasable
<b>Location :</b>	The computer memory is placed in three different locations:
CPU	It is in the form of CPU registers and its internal cache memory (16 kbytes in case of Pentium)
internal	It is the main memory of the system which CPU can access directly.
External	It is in the form of secondary storage devices such as magnetic disk, tapes etc. The CPU accesses this memory with the help of I/O controllers.
<b>Capacity :</b>	It is expressed using two terms : Word size and number of words.
<b>Word size</b>	It is expressed in bytes (8-bit). The common word sizes are 8, 16 and 32-bits.
<b>Number of word</b>	This term specifies the number of words available in the particular memory device. For example, if memory capacity is $4\text{ K} \times 8$ , then its word size is 8 and number of words are $4\text{ K} = 4096$ .
<b>Unit of transfer :</b>	It is the maximum number of bits that can be read or written into the memory at a time. In case of main memory, most of the times it is equal to word size. In case of external memory, unit of transfer is not limited to word size, it is often larger than a word and it is referred to as blocks.
<b>Access methods :</b>	There are two different methods generally used for memory access.

**Sequential access**

Here, memory is organized into units of data, called records. If current record is 1, then in order to read record N, it is necessary to read physical records 1 through  $N - 1$ . A tape drive is an example of sequential access memory.

**Random access**

Here, each addressable location in memory has an unique address. It is possible to access any memory location at random.

**Performance :**

The performance of the memory system is determined using three parameters :

**Access time**

In case of random access memory, it is the time taken by memory to complete read/write operation from the instant that an address is sent to the memory. On the other hand, for non-random access memory, access time is the time it takes to position the read/write mechanism at the desired location.

**Memory cycle time**

This term is used only in concern with random access memory and it is defined as access time plus additional time required before a second access can commence.

**Transfer rate**

It is defined as the rate at which data can be transferred into or out of a memory unit.

**Physical type :**

Two most common physical types used today are *semiconductor memory* and *magnetic surface memory*.

**Physical characteristics :**

**Volatile / Non-volatile** If memory can hold data even if power is turned off, it is called as non-volatile memory; otherwise it is called as volatile memory.

**Erasable / Non-erasable** The memories in which data is once programmed cannot be erased are called as non-erasable memories. On the other hand, if data in the memory is erasable then memory is called as erasable memory.

**Review Question**

1. List the different characteristics of memories.

**7.3 Classification of Memories**

- The Table 7.3.1 shows the classification of semiconductor memory devices. The semiconductor memory devices can be categorized in several ways according to their functional and architectural characteristics.

Classification of semiconductor memories			
Non-volatile memory		Volatile memory	
Read Only Memory (ROM)	Read/Write Memory (NVRAM)	Read/Write Memory (RWM)	
• Mask-Programmable ROM	• EEPROM	Random Access	Non-Random Access
• Programmable ROM	• EEPROM	• SRAM	• FIFO
	• FLASH	• DRAM	• LIFO
		• DSAM	• Shift Register

**Note :**

- ROM : Read Only Memory
- PROM : Programmable ROM
- EPROM : Erasable PROM
- EEPROM : Electrically Erasable PROM
- SRAM : Static Random Access Memory
- DRAM : Dynamic Random Access Memory
- FIFO : First-in First-out
- LIFO : Last-in First-out

Table 7.3.1 Classification of semiconductor memories

- Boardly semiconductor memories are classified as volatile memories and non-volatile memories. Volatile memories can retain their state as long as power is applied. On the other hand, non-volatile memories can hold data even if power is turned off. Read/Write Memories (RWMs) are those memories, which allows both read and write operations. They are used in applications where data has to change continuously. They are also used for temporary storage of data. ROM memories allow only read operation. They are used to store monitor programs and constants used in the program.
- The volatile memories which can hold data as long as power is ON are called static RAMs (SRAMs). Dynamic RAMs (DRAMs) stores the data as a charge on the capacitor and they need refreshing of charge on the capacitor after every few milliseconds to hold the data even if power is ON.
- EEPROM and EEPROM are erasable memories in which the stored data can be erased and new data can be stored.

- The semiconductor memories are also classified as Bipolar and MOS memories depending upon the type of transistors used to construct the individual cell.

**Review Questions**

- Describe the classification of semiconductor memories.
- Which memory is called volatile? Why?
- What is meant by 'static' and 'dynamic' memories?
- Explain static memory.
- Discuss the classification of ROM and RAM memories.

**7.4 Serial (Sequential) Random Access Memories**

The Table 7.4.1 shows the comparison between serial and random access memories.

St. No.	Serial access memories	Random access memories
1.	Use serial access method.	Use random access method.
2.	Memory is organized into units of data, called records. Records are accessed sequentially. If current record is 1, then in order to read record N, it is necessary to read physical records 1 through N-1.	Each storage location in the memory has an unique address and it can be accessed independently of the other locations.
3.	Memory access time is dependent on the position of storage location.	Memory access time is independent of storage location being accessed.
4.	The time required to bring the desired location into correspondence with a read-write head increases effective access time, so serial access tends to be slower than random access.	Memory access time is less.
5.	Cheaper than random access memories.	Random access memories are comparatively costly.
6.	Nonvolatile memories.	May be volatile or nonvolatile depending on physical characteristics.
7.	Magnetic tape is an example of serial access memories.	Semiconductor memories are random access memories.

Table 7.4.1 Comparison between serial and random access memories

**Review Question**

- Give the comparison between serial access and random access memories.

**7.5 Read Only Memory (ROM)**

- We can't write data in read only memories. It is non-volatile memory i.e. it can hold data even if power is turned off. Generally, ROM is used to store the binary codes for the sequence of instructions you want the computer to carry out and change.
- It is important to note that although we give the name RAM to static and dynamic read/write memory devices, that does not mean that the ROMs that we are using are also not random access devices. In fact, most ROMs are accessed randomly with unique addresses.
- The Fig. 7.5.1 shows the typical configuration of a ROM cell. It consists of a transistor T and switch P. The transistor T is driven by the word line. The contents of cell can be read from the cell when word line is logic 1. A logic value 0 is read if the transistor is connected to ground through switch P. If switch P is open, a logic value 1 is read. The bit line is connected through a resistor to the power supply. A sense circuit at the end of the bit line generates the proper output value. Data is stored into a ROM when it is manufactured.

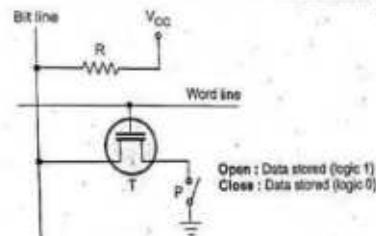


Fig. 7.5.1 ROM cell

- There are four types of ROM : Masked ROM, PROM, EEPROM or E<sup>2</sup>PROM.

**7.5.1 PROM (Programmable Read Only Memory)**

- PROMs are programmed by user. To provide the programming facility, each address select and data line intersection has its own fused MOSFET or transistor. When the fuse is intact, the memory cell is configured as a logic 1 and when fuse is blown (open circuit), the memory cell is logical 0. Logical 0s are programmed by selecting the appropriate select line and then driving the vertical data line with a pulse of high current. The Fig. 7.5.2 shows a PROM fused MOSFET memory cell.

- Fig. 7.5.3 shows four byte PROM. It has diodes in every bit position; therefore, the output is initially all 0s. Each diode, however, has a fusible link in series with it. By addressing bit and applying proper current pulse at the corresponding output, we can blow out the fuse, storing logic 1 at that bit position. The fuse uses material like nichrome and polycrystalline. For blowing the fuse it is necessary to pass around 20 to 50 mA of current for period 5 to 20  $\mu$ s.
- The blowing of fuses according to the truth table is called programming of ROM. The user can program PROMs with special PROM programmer. The PROM programmer selectively burns the fuses according to the bit pattern to be stored. This process is also known as burning of PROM. The PROMs are one time programmable. Once programmed, the information stored is permanent.

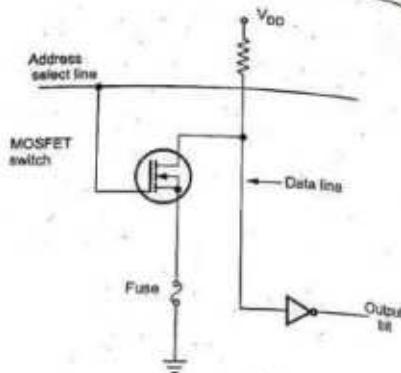


Fig. 7.5.2 Single fused PROM cell

The blowing of fuses according to the truth table is called programming of ROM. The user can program PROMs with special PROM programmer. The PROM programmer selectively burns the fuses according to the bit pattern to be stored. This process is also known as burning of PROM. The PROMs are one time programmable. Once programmed, the information stored is permanent.

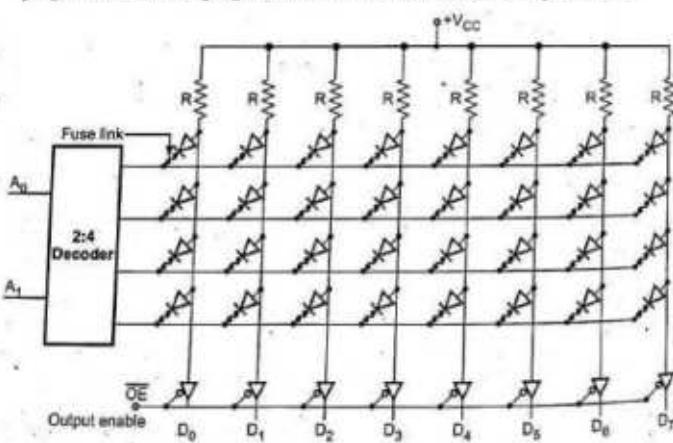


Fig. 7.5.3 Four byte PROM

### 7.5.2 EPROM (Erasable Programmable Read Only Memory)

- Erasable programmable ROMs use MOS circuitry. They store 1's and 0's as a packet of charge in a buried layer of the IC chip. EPROMs can be programmed by the user with a special EPROM programmer. The important point is that we can erase the stored data in the EPROMs by exposing the chip to ultraviolet light through its quartz window for 15 to 20 minutes, as shown in the Fig. 7.5.4.

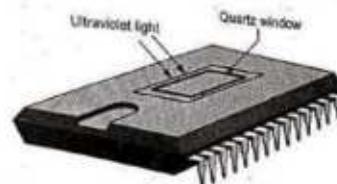


Fig. 7.5.4 EPROM

- It is not possible to erase selective information, when erased the entire information is lost. The chip can be reprogrammed. This memory is ideally suitable for product development, experimental projects and college laboratories, since this chip can be reused many times, over.

#### **EPROM Programming :**

- When erased each cell in the EPROM contains 1. Data is introduced by selectively programming 0's into the desired bit locations. Although only 0's will be programmed, both 1's and 0's can be presented in the data.
- During programming address and data are applied to address and data pins of the EPROM. When the address and data are stable, program pulse is applied to the program input of the EPROM. The program pulse duration is around 50 ms and its amplitude depends on EPROM IC. It is typically 5.5 V to 25 V. In EPROM, it is possible to program any location at any time - either individually, sequentially, or at random.

### 7.5.3 EEPROM (Electrically Erasable Programmable Read Only Memory)

- Electrically erasable programmable ROMs also use MOS circuitry very similar to that of EPROM. Data is stored as charge or no charge on an insulated layer or an insulated floating gate in the device. The insulating layer is made very thin ( $< 200 \text{ \AA}$ ). Therefore, a voltage as low as 20 to 25 V can be used to move charges across the thin barrier in either direction for programming or erasing. EEPROM allows selective erasing at the register level rather than erasing all the information

since the information can be changed by using electrical signals. The EEPROM memory also has a special chip erase mode by which entire chip can be erased in 10 ms. This time is quite small as compared to time required to erase EEPROM and it can be erased and reprogrammed with device right in the circuit. However, EEPROMs are most expensive and the least dense ROMs.

#### 7.5.4 EEPROM

- EEPROM is an acronym for Electrically Alterable Programmable Read - Only Memory. It is a non-volatile read only memory. It is a form of PROM in which the contents of selected memory locations can be changed by applying suitable electrical signals without removing the EEPROM from the printed circuit board.

#### Advantages of EEPROM

- We can selectively program or can selectively erase EEPROM.
- Erasing time is 5 to 10 ms and programming time is 250  $\mu$ s to 1 ms for a particular location. This time is quite low.

#### Disadvantages of EEPROM

- It is expensive.
- It is not available from a variety of source.
- It is not popular as this technology is still not matured.

#### Review Questions

1. Write note on ROM technologies.
2. Give the features of a ROM cell.
3. Briefly explain any four nonvolatile memory in detail.
4. Write a note on ROM.
5. Explain EPROM.
6. Write a note on EEPROM.
7. Mention the two types of erasable PROM.
8. How is individual location in a EEPROM programmed or erased?
9. Elaborate the single fused PROM cell with clear sketch.
10. Write the advantages of EPROM over PROM.
11. Whether ROM is classified as nonvolatile storage device? Why?
12. Write a note on types of ROMs.
13. Explain the various ROM organizations and give the uses for each type.
14. Write a note on EEPROM.
15. Briefly explain the EPROM and EEPROM technology.
16. Write short note on EEPROM.

17. Define the following : EPROM

18. List down the various types of ROMs and discuss any two of them.

GTU : Winter-14, Mark 1

GTU : Winter-14, 18, Marks 4

#### 7.6 Read and Write Memory (RAM)

There are two types of RAMs :

- Static RAM
- Dynamic RAM

##### 7.6.1 Static RAM (SRAM)

- Memories that consists of circuits capable of retaining their state as long as power is applied are known as static memories. These are random access memory (RAM) and hence commonly called static RAM memories.

##### 7.6.1.1 Static RAM Cell

- The Fig. 7.6.1 shows the implementation of static RAM cell. It consists of two cross-coupled inverters as a latch and two transistors  $T_1$  and  $T_2$  which act as a switches.

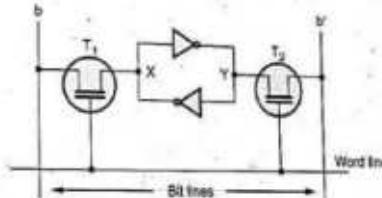


Fig. 7.6.1 Static RAM Cell

- The latch is connected to two bit lines by transistors  $T_1$  and  $T_2$ . The word line controls the opening and closing of transistors  $T_1$  and  $T_2$ . When word line is at logic 0 level (Ground level), the transistors are off and the latch retains its state.

##### Read operation

- For read operation, word line is made logic 1 (high) so that both transistors are ON. Now if the cell is in state 1, the signal on bit line  $b$  is high and the signal on bit line  $b'$  is low. The opposite is true if the cell is in state 0. The  $b$  and  $b'$  are complements of each other. The sense/write circuits connected to the bit lines monitor the states of  $b$  and  $b'$  and set the output accordingly.

**Write operation**

For write operation, the state to be set is placed on the line  $b$  and its complement is placed on line  $b'$  and then the word line is activated. This action forces the cell into the corresponding state and write operation is completed.

**7.6.2 Dynamic RAM (DRAM)****7.6.2.1 Dynamic RAM Cell**

Dynamic RAM stores the data as a charge on the capacitor. Fig. 7.6.2 shows the dynamic RAM cell. A dynamic RAM contains thousands of such memory cells. When COLUMN (Sense) and ROW (Control) lines go high, the MOSFET conducts and charges the capacitor. When the COLUMN and ROW lines go low, the MOSFET opens and the capacitor retains its charge. In this way, it stores 1 bit. Since only a single MOSFET and capacitor are needed, the dynamic RAM contains more memory cells as compared to static RAM per unit area.

The disadvantage of dynamic RAM is that it needs refreshing of charge on the capacitor after every few milliseconds. This complicates the system design, since it requires the extra hardware to control refreshing of dynamic RAMs.

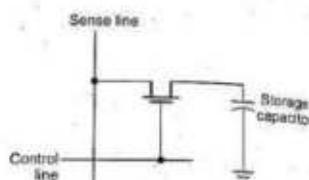


Fig. 7.6.2 Dynamic RAM

**7.6.2.2 Comparison between SRAM and DRAM**

Sl. No.	Static RAM	Dynamic RAM
1.	Static RAM contains less memory cells per unit area.	Dynamic RAM contains more memory cells as compared to static RAM per unit area.
2.	It has less access time hence faster memories.	Its access time is greater than static RAMs.
3.	Static RAM consists of number of flip-flops. Each flip-flop stores one bit.	Dynamic RAM stores the data as a charge on the capacitor. It consists of MOSFET and the capacitor for each cell.
4.	Refreshing circuitry is not required.	Refreshing circuitry is required to maintain the charge on the capacitors after every few milliseconds. Extra hardware is required to control refreshing. This makes system design complicated.
5.	Cost is more.	Cost is less.

**7.6.3 Comparison between RAM and ROM**

Parameter	RAM	ROM
Definition	It is a read / write memory.	It is a read only memory.
Volatility	It is a volatile memory, its contents are lost when the power is turned off.	It is a non-volatile memory, its contents are retained even after the power is switched off.
Types	The two main types of RAM are static RAM and dynamic RAM.	The types of ROM include PROM, EPROM and EEPROM.
Speed	Ram chips can read data faster than ROM.	Data retrieve speed is slower than RAM chips.
Use	RAM allows the computer to read data quickly to run applications. It also allows to store temporary data and intermediate results.	Since it is a read only memory it is used to store program / code and constants. In computers, it is used to store program necessary to boot the computer.

**Review Questions**

1. Draw a RAM cell and explain its working.
2. Draw the basic dynamic memory cell.
3. Write a note on dynamic RAM cell.
4. What is RAM ?
5. Compare static RAMs and dynamic RAMs.
6. What are the advantages of static RAM compared to dynamic RAM ?
7. Differentiate static and dynamic RAM. Draw the circuits of one cell of each and explain its working principle.
8. Compare RAM and ROM.

**7.7 Memory Organization**

- \* Memory cells are organized in the form of an array, in which each cell is capable of storing one bit of information. Let us see the organization of memory chip using single decoder and using two dimensional decoding.

**Organization using single decoder**

- \* The Fig. 7.7.1 shows the organization of  $16 \times 8$  (16 words of 8-bit each) memory. Here, the organization is shown with detail connections of address lines, data lines and control lines. The data input and the data output of each sense/write circuit are connected to a single bidirectional data line that can be connected to the data

- bus of a computer. Two control lines, R/W and  $\overline{CS}$ , are provided in addition to address and data lines. The R/W (Read/Write) input specifies the required operation and the  $\overline{CS}$  input selects a given chip in a multi chip memory system.
- As shown in the Fig. 7.7.1 there are 16 words of 8-bits each. A memory with 16 words needs four address lines. The four address inputs go through a  $4 \times 16$  decoder to select one of the sixteen words. The decoder is enabled with a memory enable input. When the memory enable is 0, all outputs of the decoder are 0 and none of the memory words are selected. With the memory select at 1, one of the sixteen words is selected, dictated by the value in the four address lines. Once a word has been selected, the read/write input determines the operation. During a write operation, the data available in the input lines are transferred into the eight memory cells of the selected word. The memory cells that are not selected are disabled and their previous binary values remain unchanged.

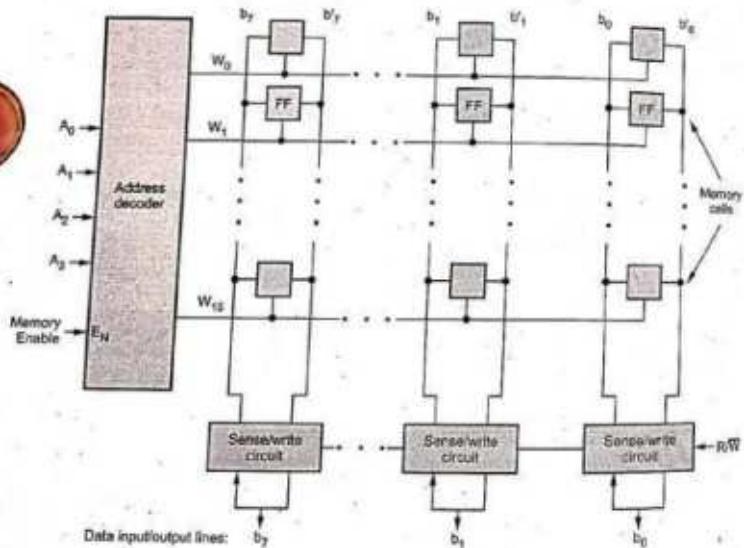


Fig. 7.7.1 Organization of bit cells in a memory chip

## organization using Two dimensional decoding

- A decoder with  $k$  inputs and  $2k$  outputs requires  $2k$  AND gates with  $k$  inputs per gate. We can reduce the total number of gates and the number of inputs per gate by employing two decoders in a two-dimensional selection scheme. The Fig. 7.7.2 shows the two dimensional decoding scheme. Here, two  $k/2$  input decoders are used instead of one  $k$ -input decoder. One decoder performs the row selection and the other the column selection in a two dimensional matrix configuration.

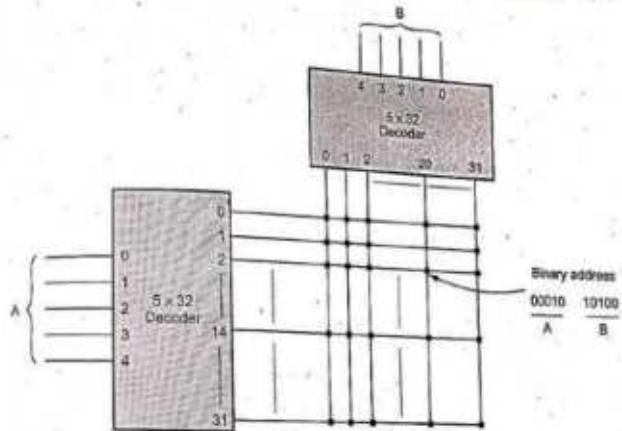


Fig. 7.7.2 Two dimensional decoding scheme

- The two-dimensional selection pattern shown in Fig. 7.7.2 has  $k = 10$ , i.e. 10 decoder inputs. These decoder inputs are divided by 2 to get individual decoder inputs, i.e. 5 input for each decoder. Therefore, here instead of using a single  $10 \times 1024$  decoder, we use two  $5 \times 32$  decoders. With a single decoder we would need 1024 AND gates with 10 inputs in each. In the two - decoder case, we need 64 AND gates with five inputs in each. The five most significant bits of the address go to input A and the five least significant bits go to input B. Each word within goes to one A and one B line. Thus, the memory array is selected by the coincidence of one A and one B line. Thus, each word in memory is selected by the coincidence between 1 of 32 rows and 1 of 32 columns for a total of 1024 words.

## Review Question

1. Write a note on memory organization.

## 7.8 | Expanding Memory Size

- It is possible to expand word size of memory by connecting two or more ICs together. The word size of memory IC can be increased by connecting two memory ICs in such a way that their data bus is in series and address bus in parallel. Both memory ICs are selected simultaneously by common chip select signal to access entire expanded word at a time. The example 7.8.1 illustrates method of word expansion.

**Example 7.8.1** Design  $1 \text{ K} \times 8$  RAM using two 2114 ICs.

Solution :

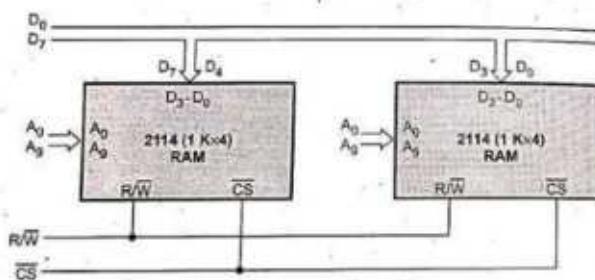


Fig. 7.8.1 Expanding word size

- The memory capacity can be increased by connecting two or more memory ICs in parallel. This means that the address, data and control lines are connected in parallel to all memory ICs. Each IC is selected by the separate chip select signal generated by the address decoder. This is illustrated by the following example.

## Memory Map

$A_9$	$A_8$	$A_7$	$A_6$	$A_5$	$A_4$	$A_3$	$A_2$	$A_1$	$A_0$	Memory address
0	0	0	0	0	0	0	0	0	0	Starting address
1	1	1	1	1	1	1	1	1	1	End address

**Example 7.8.2** Design  $2 \text{ K} \times 4$  RAM using two 2114 ICs.

Solution :

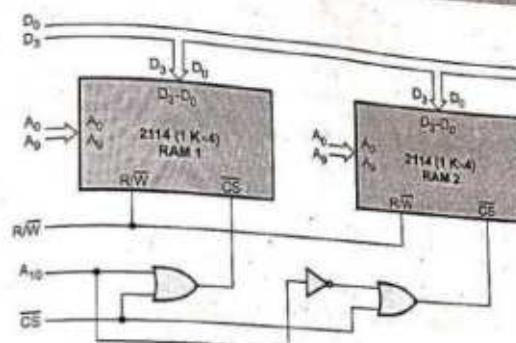


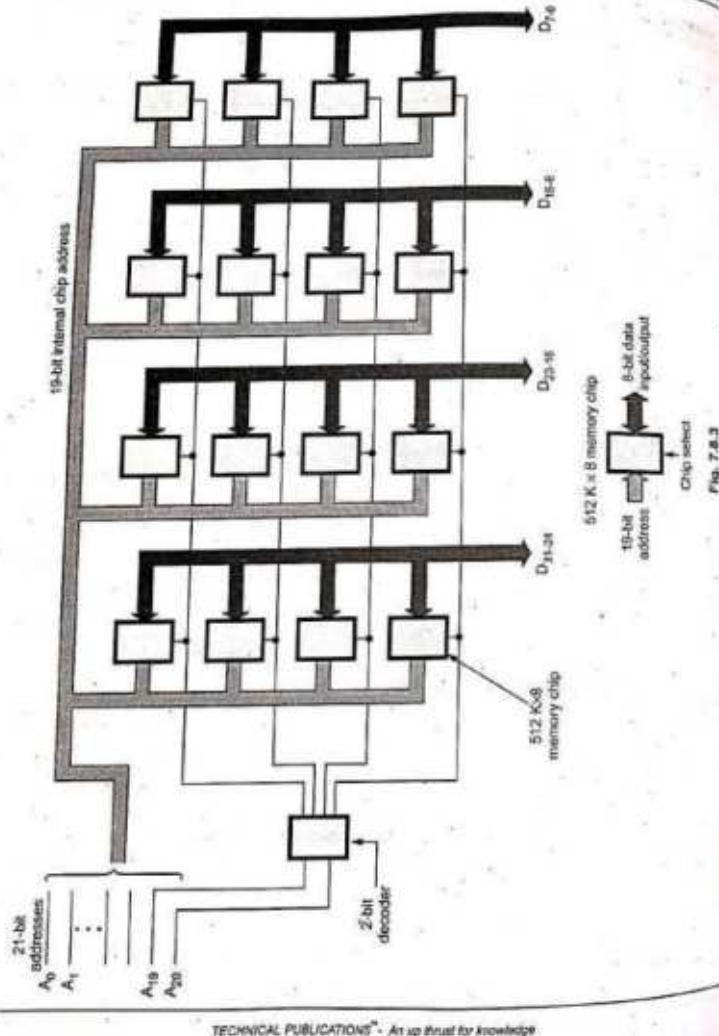
Fig. 7.8.2

## Memory Map

$A_{10}$	$A_9$	$A_8$	$A_7$	$A_6$	$A_5$	$A_4$	$A_3$	$A_2$	$A_1$	$A_0$	Memory ICs
0	0	0	0	0	0	0	0	0	0	0	RAM1
0	1	1	1	1	1	1	1	1	1	1	RAM1
1	0	0	0	0	0	0	0	0	0	0	RAM2
1	1	1	1	1	1	1	1	1	1	1	RAM2

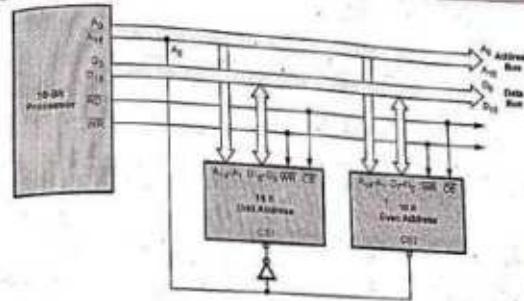
**Example 7.8.3** Implementation of 2M words of 32-bit each using 512 Kx8 static memory chips.

Solution : The Fig. 7.8.3 shows the implementation of 2 M words of 32-bit each using 512 Kx8 static memory chip. In this implementation, we have expand word size as well as capacity of the memory. Since each memory chip has byte organization, we have to use 4 chips in each row to get the word size of 32 bits ( $8 \times 4$ ). Similarly, to increase memory capacity up to 2 M we have to use 4 memory chips of capacity 512 K in each column.



**Example 7.8.4** For a 32 K memory, obtain memory map to interface memory to 16-bit processor.

**Solution :**



#### Memory Map :

Memory Address	A <sub>15</sub>	A <sub>14</sub>	A <sub>13</sub>	A <sub>12</sub>	A <sub>11</sub>	A <sub>10</sub>	A <sub>9</sub>	A <sub>8</sub>	A <sub>7</sub>	A <sub>6</sub>	A <sub>5</sub>	A <sub>4</sub>	A <sub>3</sub>	A <sub>2</sub>	A <sub>1</sub>	A <sub>0</sub>	Address
Even Address	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000H
Odd Address	X	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	FFFFH
Odd Address	X	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0001H
Odd Address	X	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	FF99H

**Note :** A<sub>15</sub> is considered logic 0.

**Example 7.8.5** How many 128 x 8 memory chips are needed to provide a memory capacity of 4096 x 16.

**Solution :** Number of chips required =  $\frac{\text{Required RAM size}}{\text{Available chip capacity}} = \frac{4096 \times 16}{128 \times 8} = 64$

Hence, 64 chips of 128 x 8 size are needed to provide a memory capacity of 4096 x 16.

**Example 7.8.6** Given a 32 x 8 size ROM chip with an enable input. Show the external connections necessary to construct a 128 x 8 ROM with four chips and decoder.

**Solution :** Address bus width =  $\log_2 128 = \log_2 2^7 = 7$ -bits = A<sub>6</sub> - A<sub>0</sub>

Data bus width = 8-bits = D<sub>7</sub> - D<sub>0</sub>  
Address lines required for 32 x 8 ROM =  $\log_2 3^2 = \log_2 9 = 4$  bits = A<sub>5</sub> - A<sub>2</sub>

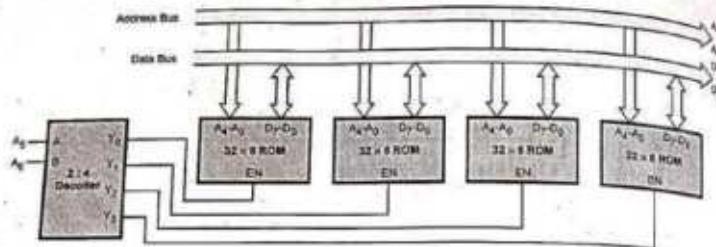


Fig. 7.8.5

**Example 7.8.7** A microprocessor uses RAM chips of  $1024 \times 1$  capacity.

- How many chips are needed and how should their address lines be connected to provide a memory capacity of  $1024$  bytes?
- How many chips are needed to provide a memory capacity of  $16$  kbytes? Explain in words how the chips are to be connected to the address bus.

**Solution :** i) Eight chips are needed with address lines connected in parallel as shown in the Fig. 7.8.6.

$$\text{ii) Number of chips needed} = \frac{\text{Required memory capacity}}{\text{Chip size}}$$

$$= \frac{16 \text{ K} \times 8}{1 \text{ K} \times 1} = 128 \text{ chips}$$

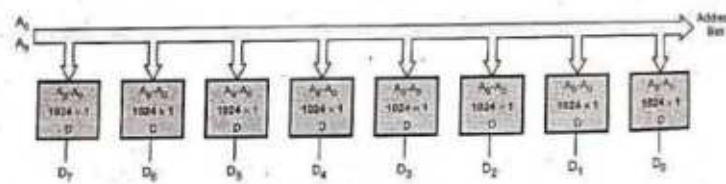


Fig. 7.8.6

- The address lines  $A_9 - A_0$  from the address bus are connected in parallel to all the chips. In all there are 128 chips. These 128 chips are divided into 16 groups consists of 8 chips in each group. The address lines  $A_{13} - A_{10}$  are decoded to generate 16 chip select signals one for each group so that at a time all chips from one group will be selected when corresponding chip select signal is activated.

## Review Questions

- Explain how to expand memory word size with the help of example.
- Explain how to expand memory capacity with the help of example.

## 7.9 Content Addressable Memory (CAM)

Many data-processing applications require the search of items in a table stored in memory. They use object names or number to identify the location of the named or numbered object within a memory space. For example, an account number may be searched in a file to determine the holder's name and account status. To search an object, the number of accesses to memory depends on the location of the object and the efficiency of the search algorithm.

The time required to find an object stored in memory can be reduced considerably if objects are selected based on their contents, not on their locations. A memory unit accessed by the content is called an associative memory or Content Addressable Memory (CAM). This type of memory is accessed simultaneously and in parallel on the basis of data content rather than by specific address or location.

## 7.9.1 Hardware Organization

The Fig. 7.9.1 shows the block diagram of an associative memory. It consists of memory array with match logic for  $m$   $n$ -bit words and associated registers. The argument register (A) and key register (K) each have  $n$ -bits per word. Each word in memory is compared in parallel with the contents of the argument register. The

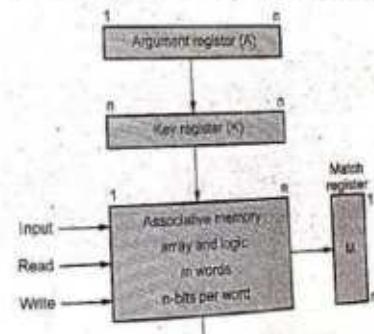


Fig. 7.9.1 Block diagram of associative memory

words that match with the word stored in the argument register set a corresponding bits in the match register. Therefore, reading can be accomplished by a sequential access to memory for those words whose corresponding bits in the match register have been set.

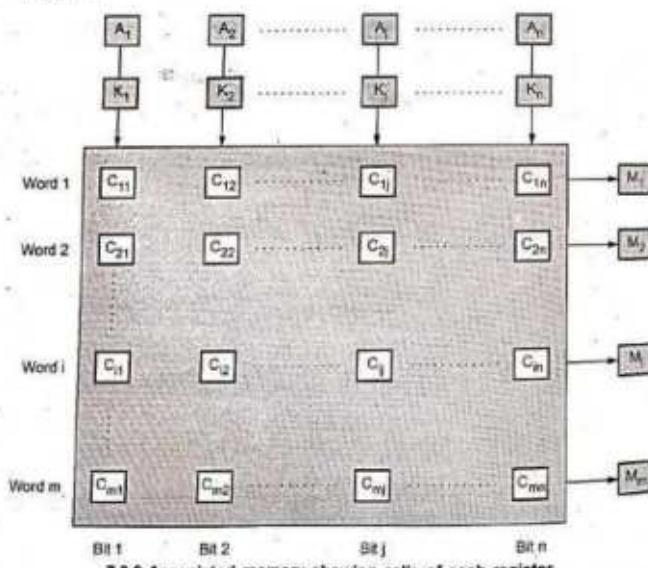
- The key register provides a mask for choosing a particular field or bits in the argument word. Only those bits in the argument register having 1's in their corresponding position of the key register are compared. For example, if argument register A and the key register K have the bit configuration shown below. Only the three rightmost bits of A are compared with memory words because K has 1's in these positions.

A	1 1 0 1 1	0 1 0
K	0 0 0 0 0	1 1 1

Word 1 0 1 0 1 0 0 1 0 match

Word 2 1 1 0 1 1 1 0 0 no match

- The Fig. 7.9.2 shows the associated memory with cells of each register. The cells in the memory array are marked by the letter C with two subscripts. The first subscript gives the word number and the second subscript gives the bit position in the word.



7.9.2 Associated memory showing cells of each register

- The Fig. 7.9.3 shows the internal organization of a typical cell. It consists of a SR flip-flop as a storage element and the circuit for reading, writing and matching into the storage cell. To carry-out read operation read signal is made logic 1. The match logic compares the bit in the storage cell with the corresponding unmasked bit of the argument and provides an output for the decision logic that sets the corresponding bit in the match register.

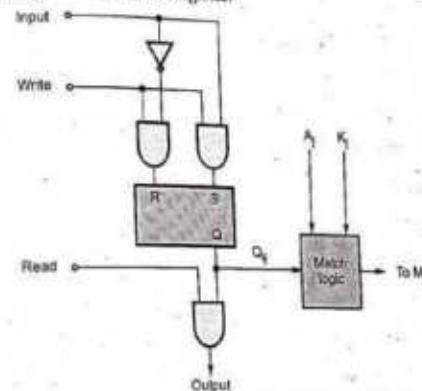


Fig. 7.9.3 Internal organization of typical cell of associative memory

- The Fig. 7.9.4 shows the match logic for each word. The cell bit ( $Q_i$ ) is compared with the corresponding argument bit with EX-NOR gate. EX-NOR gate gives valid output logic 1 only when its inputs are same. The output of EX-NOR gate is valid only when the corresponding bit in key register is logic 1. This condition is

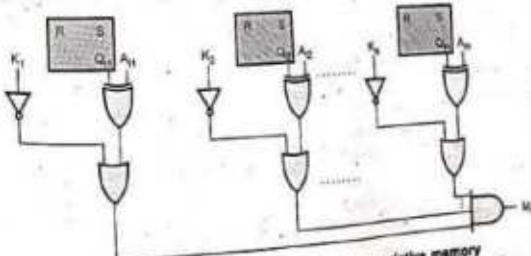


Fig. 7.9.4 Match logic for one word of associative memory

implemented using 2-input OR gate. When corresponding bit in key register is logic 1, the inverter gives output 0 which forces the output of OR gate to follow the second input, i.e. the comparison output; otherwise output of OR-gate is logic 1. The outputs of all OR-gates are then ANDed with n-input AND gate to check whether all bits in the word are matched with the bits in the argument register.

### 7.9.2 Read Operation

- In the read operation, all matched words are read in sequence by applying a read signal to each word line whose corresponding  $M_i$  bit is a logic 1. In applications where no two identical items are stored in the memory, only one word may match the unmasked argument field. In such case, we can use  $M_i$  output directly as a read signal for the corresponding word. The contents of the matched word will be presented automatically at the output lines and no special read signal is needed.

### 7.9.3 Write Operation

- In write operation, we consider two separate cases : 1. It is necessary to load entire memory with new information 2. It is necessary to replace one word with new information. The entire memory can be loaded with new information by addressing each location in sequence. This will make the memory device a random access memory for writing and a content addressable memory for reading. Here, the address for the input can be decoded as in a random access memory. Thus instead of having  $m$  address lines, one for each word, the number of address lines can be reduced by the use of decoder to  $d$  lines, where  $m = 2^d$ .
- To implement the write operation in the second case the tag register is used. This register has as many bits as there are words in the memory for every active (valid) word stored in the memory, the corresponding bit in the tag register is set to 1. When word is deleted from the memory the corresponding tag bit is cleared, i.e. it is set to logic 0. The word is then stored in the memory by scanning the tag register until the first 0 bit is encountered. This gives the first available inactive word and a position for writing a new word. After the new word is stored in the memory it is made active by setting the corresponding tag bit in the tag register.

#### Review Questions

- Write a short note on associative memory.
- Describe by means of block diagram how multiple matched words can be read out from an associative memory.
- What is CAM ? Explain it with block diagram.
- Explain associative memory with its hardware organization. Explain how the data is read and write in the associative memory.

### 7.10 Charge - Coupled Devices (CCD) Memory

- Charge - coupled device (CCD) memory is a type of dynamic memory in which packets of charge are continuously transferred from one MOS device to another. The structure of a MOS charge-coupled device is quite simple, as shown in the Fig. 7.10.1. It consists simply of a P substrate, an insulating oxide layer, and isolated gates.

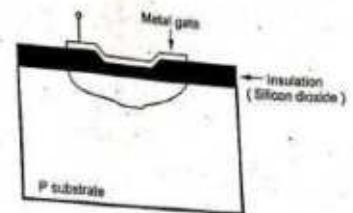


Fig. 7.10.1 MOS charge-coupled device structure

- When a high voltage is applied to the metal gate, holes are repelled from a region beneath the gate in the P-type substrate. This region, called a potential well, is then capable of accepting a packet of negative charged electrons. Therefore, data is stored in a CCD as charge, and it is transferred from one device to an adjacent one by clocking their gates.
- The charge-coupled devices are built as a serial-in, serial-out MOS shift registers, each shift register being a line of charge-coupled devices. By controlling the timing of the clock signals applied to the shift registers, data can be accessed one bit at a time from a single register or several bits at a time from multiple registers.
- The main advantage of CCD is that its simple cell structure. It makes it possible to construct large-capacity memories at low cost. Since CCDs are dynamic in nature, they must be periodically refreshed, and must be driven by rather complex, multi phase clock signals. Another disadvantage of CCDs is due to serial data storage. Serial data storage has a longer access times in comparison to semiconductor RAM memory. It is on the order of 100  $\mu$ s.
- CCDs can be used for digital or analog delay, and as serial data memories. Another exciting application of CCDs is as the light sensitive image sensor in television cameras.

#### Review Question

- Write a note on charged coupled device memories.

**7.11 Programmable Logic Devices**

- Various digital ICs for performing basic digital operations and other functions such as adders, comparators, arithmetic logic unit, multiplexers, demultiplexers, code converters, shift registers, counters etc. are known as fixed function ICs due to their fix function.
- These ICs are designed by their manufacturers and produced in large quantities to satisfy the needs of a wide variety of applic.
- We have seen the design of digital circuits using fixed function ICs.
- There are two more approaches for the design of digital circuits.
  - Use of Application Specific Integrated Circuits (ASICs)
  - Use of Programmable Logic Devices (PLDs)
- In the fixed function IC approach, we have to use various fixed function ICs to implement different functional blocks in the digital circuit.
- In ASIC, a single IC is designed and manufactured to implement the entire circuit.
- In the third approach programmable logic devices are used to implement logic functions. The main advantage of PLD approach is that PLDs can be easily configurable by the individual user for specific applications.
- The Table 7.11.1 shows the comparison between these three design approaches.

Comparison parameter	Fixed-function IC approach	ASIC approach	PLD approach
Development cost	Low	High	Low
Space required	Large	Minimum	Less
Power required	Large	Less	Less
Reliability	Less compared to other two approaches.	Highest	High
Circuit testing	Easy	Specialized testing methods are required which may increase cost and effort.	Easy
Design flexibility	Less	No	More
Modification in design	Possible with change in circuit and/or with change in components.	No	May be possible without any circuit or component changes but only by reconfiguring the device.
Design security	Lack of security i.e. circuit can easily be copied by others	High	High
Design time	Less	More	Less

**Table 7.11.1 Comparison between design approaches**

- PLDs can be reprogrammed in few seconds and hence give more flexibility to experiment with designs. Reprogramming feature of PLDs also make it possible to accept changes/modifications in the previously design circuits. These two main advantages and others discussed in Table 7.11.1 make PLDs very popular in digital design.

- According to architecture, complexity and flexibility in programming PLDs are classified as

1. PROMs : Programmable Read Only Memories
2. PLAs : Programmable Logic Arrays
3. PAL : Programmable Array Logic
4. FPGAs : Field Programmable Gate Arrays
5. CPLDs : Complex Programmable Logic Devices

**7.12 ROM / PROM as a PLD**

- ROM stands for Read Only Memory and PROM stands for Programmable Read Only Memory. The only difference in ROM and PROM is that ROM is programmed by manufacturer while PROM is user programmable.

- Fig. 7.12.1 shows the block diagram of PROM.

- It consists of  $n$ -input lines and  $m$ -output lines.
- Each bit combination of the input variables is called an address.
- Each bit combination that comes out of the output lines is called a word.
- The number of bits per word is equal to the number of output lines,  $m$ .
- The address specified in binary number denotes one of the minterms of  $n$  variables.
- The number of distinct addresses possible with  $n$ -input variables is  $2^n$ .
- An output word can be selected by a unique address and since there are  $2^n$  distinct addresses in PROM, there are  $2^n$  distinct words in the PROM.
- The word available on the output lines at any given time depends on the address value applied to the input lines.
- Let us consider  $64 \times 4$  PROM.
- The PROM consists of 64 words of 4-bits each. This means that there are four output lines and particular word from 64 words presently available on the output lines is determined from the six input lines.

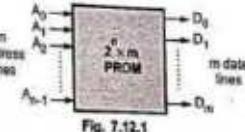


Fig. 7.12.1

- There are only six inputs in a  $64 \times 4$  PROM because  $2^6 = 64$  and with six variables, we can specify 64 addresses or minterms. For each address input, there is a unique selected word. Thus, if the input address is 000000, word number 0 is selected and applied to the output lines. If the input address is 111111, word number 63 is selected and applied to the output lines.
- The Fig. 7.12.2 shows the internal logic construction of a  $64 \times 4$  PROM. The six input variables are decoded in 64 lines by means of 64 AND gates and 6 inverters.

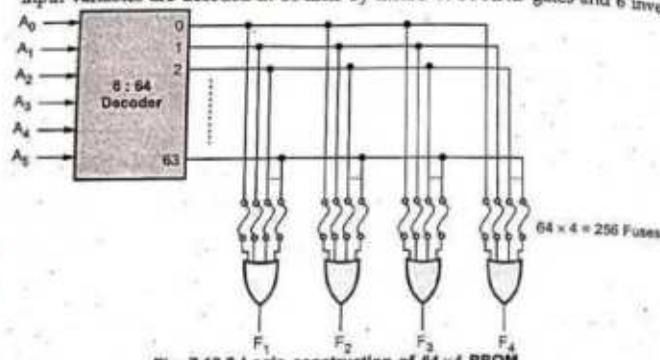


Fig. 7.12.2 Logic construction of 64x4 PROM

- Each output of the decoder represents one of the minterms of a function of six variables.
- The 64 outputs of the decoder are connected through fuses to each OR gate.
- Only four of these fuses are shown in the diagram, but actually each

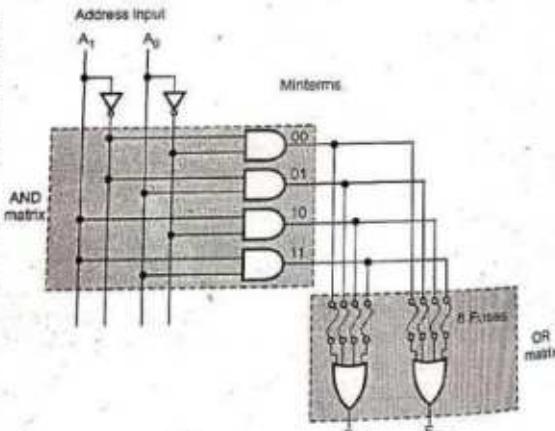


Fig. 7.12.3 (a) 4x2 PROM with AND-OR gates

OR gate has 64 inputs and each input goes through a fuse that can be blown as desired.

- The PROM is a two level implementation in sum of minterms form. Let us see AND-OR and AND-OR-INVERTER implementation of PROM. Fig. 7.12.3 shows the  $4 \times 2$  PROM with AND-OR and AND-OR-INVERTER implementations.

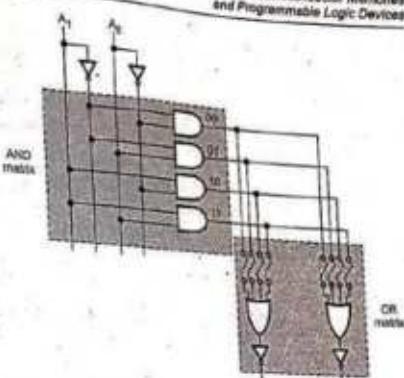


Fig. 7.12.3 (b) 4x2 PROM with AND-OR-INVERTER gates

#### 7.12.4 AND Matrix

- Fig. 7.12.4 shows the AND matrix. It is used to form product terms. It has m AND gates with 2n-inputs and m-outputs, one for each AND gate.

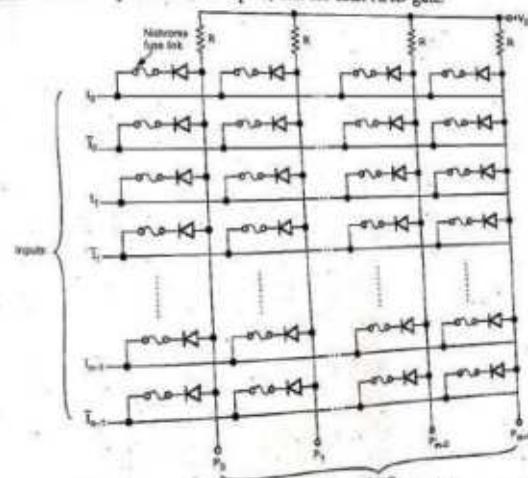


Fig. 7.12.4 Internal structure of AND matrix

- Fig. 7.12.4 shows the AND gates formed by diodes and resistors structure. Each AND gate has all the input variables in complemented and uncomplemented form.
  - There is a nichrome fuse link in series with each diode which can be burn out to disconnect particular input for that AND gate. Before programming, all fuse links are intact and the product term for each AND gate is given by
$$P = I_0 \cdot \bar{I}_0 \cdot I_1 \cdot \bar{I}_1 \cdots I_{n-1} \cdot \bar{I}_{n-1}$$
  - Fig. 7.12.5 shows the simplified and equivalent representation of input connections for one AND gate.
  - The array logic symbol shown in Fig. 7.12.5 (b) uses a single horizontal line connected to the gate input and multiple vertical lines to indicate the individual inputs.

Each intersection between horizontal line and vertical line indicates the fuse connection.

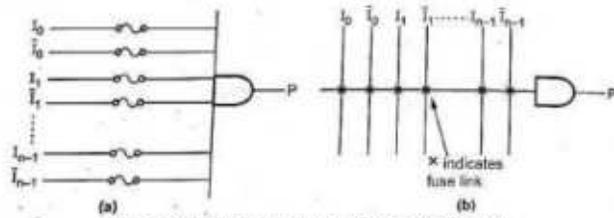


Fig. 7.12.5 Equivalent representation of AND gate

- Fig. 7.12.6 shows the simplified representation of AND matrix with input buffer.

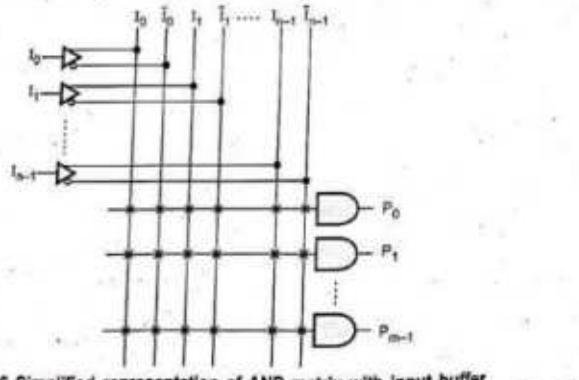


Fig. 7.12.6 Simplified representation of AND matrix with input buffer

- OR matrix is provided to produce the logical sum of the product term outputs of the AND matrix.
  - Fig. 7.12.7 shows the OR gates formed by diodes and resistors structure.

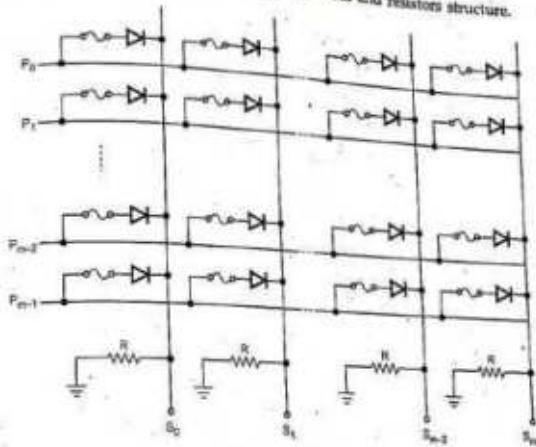


Fig. 7.12.7

- Each OR gate has all the product terms as input variables.
  - There is a nichrome fuse link in series with each diode which can be burn out to disconnect particular product term for that OR gate.
  - Before programming, all fuse link in OR matrix are also intact and the sum term for each OR gate is given by,

$$S = P_0 + P_1 + \dots + P_{m-2} + P_{m-1}$$

- Fig. 7.12.8 shows the simplified and equivalent representation of input connections for one OR gate.
  - Fig. 7.12.9 shows the simplified representation of OR matrix.

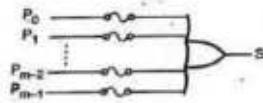


Fig. 7.12.8 Equivalent representation of OR gate

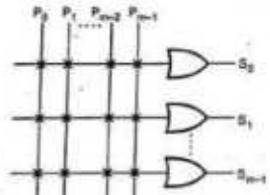


Fig. 7.12.9 Simplified representation of OR matrix

**7.12.3 Invert / Non-invert Matrix**

- Invert/Non-invert matrix provides output in the complement or uncomplemented form.
- The user can program the output in either  $S$  or complement form or uncomplement form as per design requirements.
- The typical circuits for invert/non-invert matrix is as shown in Fig. 7.12.10. In both the cases if fuse is intact the output is in its uncomplemented form; otherwise output is in the complemented form.

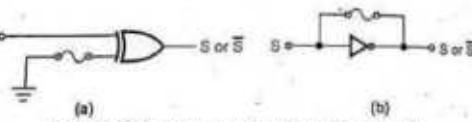


Fig. 7.12.10 Inverting and non-inverting circuits

**7.12.4 Combinational Logic Implementation using PROM**

- Looking at the logic diagram of the PROM, we can realize that each output provides the sum of all the minterms of  $n$ -input variables.
- Any Boolean function can be expressed in sum of minterms form.
- By breaking the links of those minterms not included in the function, each PROM output can be made to represent the Boolean function of one of the output variables in the combinational circuit.
- For an  $n$ -input,  $m$ -output combinational circuit, we need a  $2^n \times m$  PROM.

**Illustrative Examples**

Example 7.12.1 Using PROM realize the following expression.

$$F_1(a, b, c) = \sum m(0, 1, 3, 5, 7)$$

$$F_2(a, b, c) = \sum m(1, 2, 3, 6, 8)$$

Solution : The given functions have three inputs. They generate  $2^3 = 8$  minterms and since there are two functions, there are two outputs. The functions can be realized as shown in Fig. 7.12.11.

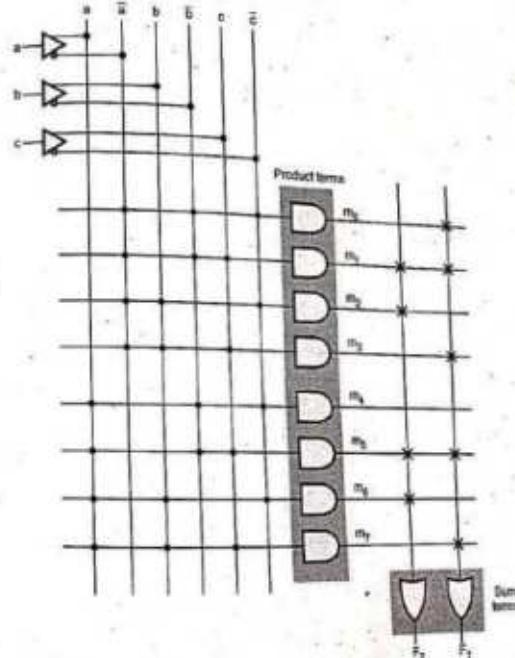


Fig. 7.12.11

The Fig. 7.12.12 shows the block diagram and truth table of PROM.



(a) Block diagram

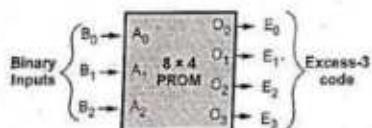
$A_2$	$A_1$	$A_0$	$F_1$	$F_2$
0	0	0	0	1
0	0	1	1	1
0	1	0	1	1
1	1	1	0	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	0	1

(b) PROM truth table

**Example 7.12.2** Design a combinational circuit using a PROM. The circuit accepts 3-bit binary number and generates its equivalent Excess-3 code.

**Solution :** Let us derive the truth table for the given combination circuit. Table 7.12.1 shows the truth table.

In practice when we are designing combinational circuits with PROM, it is not necessary to show the internal gate connections of fuses inside the unit, as shown in the Fig. 7.12.13. (Refer Fig. 7.12.13 on next page). This was shown for demonstration purpose only. The designer has to only specify the PROM (inputs and outputs) and its truth table, as shown in the Fig. 7.12.14.



(a) Block diagram

Fig. 7.12.14

$B_2$	$B_1$	$B_0$	$E_3$	$E_2$	$E_1$	$E_0$
0	0	0	0	0	1	1
0	0	1	0	1	0	0
0	1	0	0	1	0	1
0	1	1	0	1	1	0
1	0	0	0	1	1	1
1	0	1	1	0	0	0
1	1	0	1	0	0	1
1	1	1	1	0	1	0

(b) PROM truth table

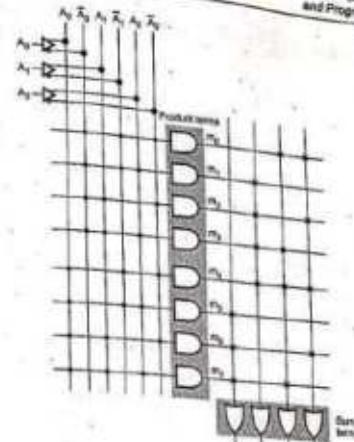
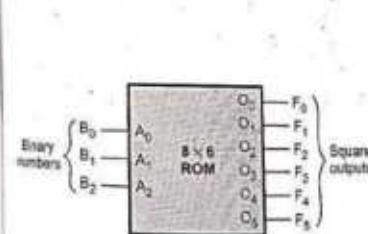


Fig. 7.12.13

**Example 7.12.3** Design a combinational circuit using ROM. The circuit accepts 3-bit number and generates an output binary number equal to square of input number.

**Solution :**



(a) Block diagram

Fig. 7.12.15

Binary numbers			Square of number on data lines					
$B_2$	$B_1$	$B_0$	$F_0$	$F_1$	$F_2$	$F_3$	$F_4$	$F_5$
0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1
0	1	0	0	0	1	0	0	0
0	1	1	0	0	1	1	0	1
1	0	0	0	0	1	1	0	0
1	0	1	1	0	0	0	0	1
1	1	0	1	0	0	0	1	0
1	1	1	1	0	1	0	0	1

(b) ROM truth table

**Example 7.12.4** Implement binary to excess-3 code converter using ROM.

**Solution :**

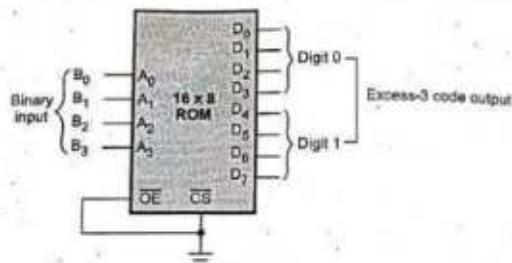


Fig. 7.12.16

Address				Memory contents							
A <sub>3</sub>	A <sub>2</sub>	A <sub>1</sub>	A <sub>0</sub>	D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>
0	0	0	0	0	0	1	1	0	0	1	1
0	0	0	1	0	0	1	1	0	1	0	0
0	0	1	0	0	0	1	1	0	1	0	1
0	0	1	1	0	0	1	1	0	1	1	0
0	1	0	0	0	0	1	1	0	1	1	1
0	1	0	1	0	0	1	1	1	0	0	0
0	1	1	0	0	0	1	1	1	0	0	1
0	1	1	1	0	0	1	1	1	0	1	0
1	0	0	0	0	0	1	1	1	0	1	1
1	0	0	1	0	0	1	1	1	1	0	0
1	0	1	0	0	1	0	0	0	0	1	1
1	0	1	1	0	1	0	0	0	1	0	0
1	1	0	0	0	1	0	0	0	1	1	0
1	1	0	1	0	1	0	0	0	1	1	1
1	1	1	0	0	1	0	0	0	1	1	1
1	1	1	1	0	1	0	0	1	0	0	0

Table 7.12.2 ROM contents

**Example 7.12.5** Implement following functions using ROM.

$$F_1 = \sum m(1, 3, 4, 6), F_2 = \sum m(2, 4, 5, 7), F_3 = \sum m(0, 1, 5, 7)$$

**Solution :**

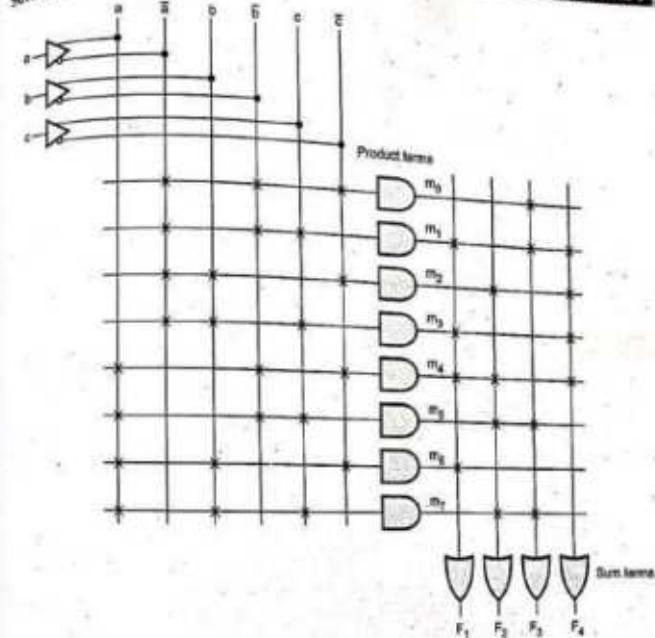


Fig. 7.12.17

**Example 7.12.6** Using 8x4 ROM, realize the expression  $F_1 = AB'C + ABC + A'BC$ ,  $F_2 = A'B'C' + A'BC' + AB'C$ ,  $F_3 = A'B'C' + ABC$ . Show the contents of all locations.

(GTU) Semester 15, Marks 7

**Solution :**  $F_1 = A\bar{B}C + A\bar{B}\bar{C} + \bar{A}BC = \sum m(5, 6, 7)$

$$F_2 = \bar{A}B\bar{C} + \bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C} = \sum m(1, 2, 4)$$

$$F_3 = \bar{A}B\bar{C} + A\bar{B}C = \sum m(0, 7)$$

Location			4 - Memory contents			
Address			D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>
A	B	C	(x)	(F <sub>3</sub> )	(F <sub>2</sub> )	(F <sub>1</sub> )
0	0	0	1	1	0	0
0	0	1	1	0	1	0
0	1	0	1	0	1	0
0	1	1	1	0	0	1
1	0	0	1	0	1	0
1	0	1	1	0	0	1
1	1	0	1	0	0	1
1	1	1	1	1	0	0
Unused						

**Example 7.12.7** Design a 3-bit gray to binary code converter. Implement it using suitable PROM. Draw the necessary diagrams and tables.

GTU : Winter-18, Marks 3

**Solution :** Step 1 : Truth table for 3-bit Gray to binary code converter

	Gray Code			Binary Code		
	G <sub>2</sub>	G <sub>1</sub>	G <sub>0</sub>	B <sub>2</sub>	B <sub>1</sub>	B <sub>0</sub>
m <sub>0</sub>	0	0	0	0	0	0
m <sub>1</sub>	0	0	1	0	0	1
m <sub>2</sub>	0	1	1	0	1	0
m <sub>3</sub>	0	1	0	0	1	1
m <sub>4</sub>	1	1	0	1	0	0
m <sub>5</sub>	1	1	1	1	0	1
m <sub>6</sub>	1	0	1	1	1	0
m <sub>7</sub>	1	0	0	1	1	1

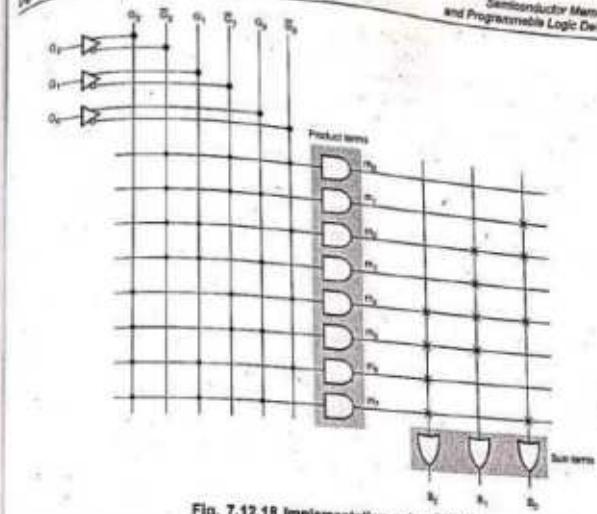


Fig. 7.12.18 Implementation using PROM

#### Examples for Practice

**Example 7.12.8 :** Design a switching circuit that converts a 4 bit binary code into a 4 bit Gray code using ROM array.

**Example 7.12.9 :** Design a PROM which will convert BCD numbers into Gray code.

#### 7.13 Programmable Logic Array (PLA)

GTU : Dec-12, May-13, Winter-16

\* The combinational circuit do not use all the minterms every time. Occasionally, they have don't care conditions. Don't care condition when implemented with a PROM becomes an address input that will never occur. The result is that not all the bit patterns available in the PROM are used, which may be considered a waste of available equipment.

\* For cases where the number of don't care conditions is excessive, it is more economical to use a second type of LSI component called a Programmable Logic Array (PLA).

\* A PLA is similar to a PROM in concept; however it does not provide full decoding of the variables and does not generate all the minterms as in the PROM.

- The PLA replaces decoder by group of AND gates, each of which can be programmed to generate a product term of the input variables.
- In PLA, both AND and OR gates have fuses at the inputs, therefore in PLA both AND and OR gates are programmable.
- Fig. 7.13.1 shows the block diagram of PLA. It consists of  $n$ -inputs, output buffer with  $m$  outputs,  $m$  product terms,  $m$  sum terms, input and output buffers.

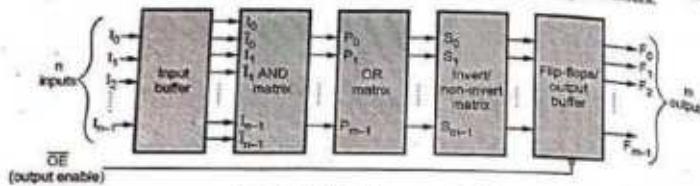


Fig. 7.13.1 Block diagram of a PLA

- The product terms constitute a group of  $m$  AND gates and the sum terms constitute a group of  $m$  OR gates, called OR matrix.
- Fuses are inserted between all  $n$ -inputs and their complement values to each of the AND gates.
- Fuses are also provided between the outputs of the AND gates and the inputs of the OR gates.
- The third set of fuses in the output inverters allows the output function to be generated either in the AND-OR form or in the AND-OR-INVERT form.
- When inverter is bypassed by link we get AND-OR implementation.
- To get AND-OR-INVERTER implementation inverter link has to be disconnected.

### 7.13.1 Input Buffer

- Input buffers are provided in the PLA to limit loading of the sources that drive the inputs.
- They also provide inverted and non-inverted form of inputs at its output.
- The Fig. 7.13.2 shows two ways of representing input buffer for single input line.

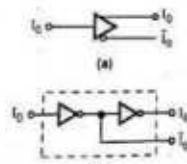


Fig. 7.13.2 Input buffer for single input line

### 7.13.2 Output Buffer

- The driving capacity of PLA is increased by providing buffers at the output. They are usually TTL compatible.
- The Fig. 7.13.3 shows the tri-state, TTL compatible output buffer.

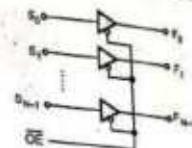


Fig. 7.13.3 Output buffers

- The output buffer may provide totem-pole, open collector or tri-state output.

### 7.13.3 Output through Flip-Flops

- For the implementation of sequential circuits we need memory elements, flip-flops and combinational circuitry for deriving the flip-flop inputs.
- To satisfy both the needs some PLAs are provided with flip-flop at each output, as shown in the Fig. 7.13.4.

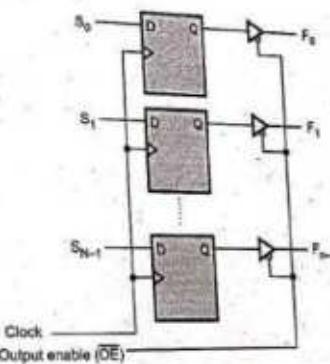


Fig. 7.13.4 PLA with flip-flop at the output

### 7.13.4 Implementation of Combination Logic Circuit using PLA

- Like ROM, PLA can be mask-programmable or field-programmable.
- With a mask-programmable PLA, the user must submit a PLA program table to the manufacturer. This table is used by the vendor to produce a user-made PLA that has the required internal paths between inputs and outputs.

- A second type of PLA available is called a field-programmable logic array or FPLA. The FPLA can be programmed by the user by means of certain recommended procedures. PLAs can be programmed with commercially available programmer units.

**Illustrative Examples**

**Example 7.13.1** A combinational circuit is defined by the functions :

$$F_1 = \sum m(3, 5, 7), F_2 = \sum m(4, 5, 7)$$

Implement the circuit with a PLA having 3 inputs, 3 product terms and two outputs.

Solution :

**Step 1 :** Simplify the given Boolean functions.

- The Boolean functions are simplified, as shown in the Fig. 7.13.5. The simplified functions in sum of products are obtained from the maps are :

$$F_1 = AC + BC, \quad F_2 = A\bar{B} + AC$$

**Step 2 :** Write PLA program table.

- Therefore, there are three distinct product terms : AC, BC and  $A\bar{B}$ , and two sum terms. The PLA program table shown in Table 7.13.1 consists of three columns specifying product terms, inputs and outputs. The first column gives the lists of product terms numerically.

The second column specifies the required paths between inputs and AND gates. The third column specifies the required paths between the AND gates and the OR gates. Under each output variable, we write a T (for true) if the output inverter is to be bypassed, and C (for complement) if the function is to be complemented with the output inverter. The product terms listed on the left of first column are not the part of PLA program table they are included for reference only.

	BC	For $F_1$				For $F_2$			
		00	01	11	10	00	01	11	10
	0	0	0	1	0	0	0	0	0
	1	0	1	1	0	1	1	1	0

Fig. 7.13.5

Product term	Inputs	Outputs				
		A	B	C	$F_1$	$F_2$
AC	1	1	-	1	1	1
BC	2	-	1	1	1	-
$A\bar{B}$	3	1	0	-	-	1
				T	T	T/C

Table 7.13.1 PLA program table

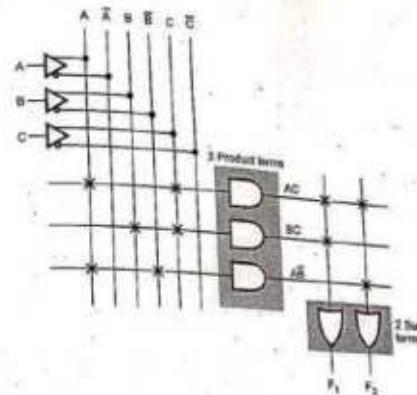
**Step 3 : Implementation**

Fig. 7.13.6

**Example 7.13.2** Draw a PLA circuit to implement the logic functions  $\bar{ABC} + A\bar{B}C + A\bar{C}$  and  $A\bar{B}\bar{C} + BC$ .

Solution :

**Step 1 :** Simplify the Boolean functions

$$\begin{aligned} \bar{ABC} + A\bar{B}C + A\bar{C} &= \bar{ABC} + A(\bar{B} + \bar{C}) \\ &= \bar{ABC} + A\bar{B} + A\bar{C} \quad \because A + \bar{A}B = A + B \\ &= \bar{ABC} + A\bar{B} + A\bar{C} \end{aligned}$$

**Note** The second Boolean function is in simplified form.

**Step 2 :** Implementation (see Fig. 7.13.7 on next page)

**Example 7.13.3** Implement the following multivalue function using  $3 \times 4 \times 2$  PLA PLD.  
 $f_1(a_2, a_1, a_0) = \sum m(0, 1, 3, 5)$  and  $f_2(a_2, a_1, a_0) = \sum m(3, 5, 7)$

Solution :

**Step 1 :** Simplify the Boolean functions.

$$\begin{aligned} f_1 &= \bar{a}_2 \bar{a}_1 + \bar{a}_2 a_0 + \bar{a}_1 a_0 \\ f_2 &= a_2 a_0 + a_1 a_0 \end{aligned}$$

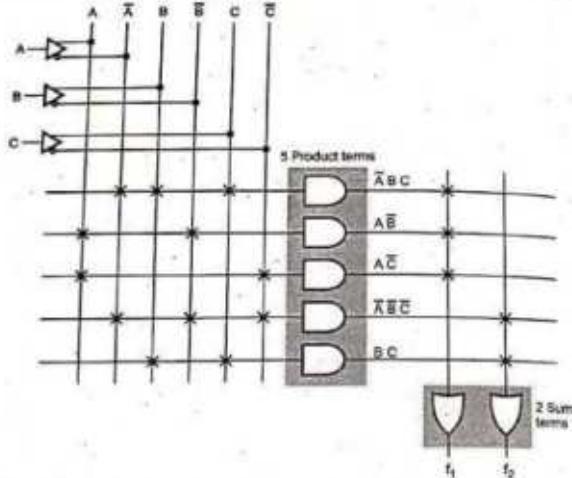


Fig. 7.13.7

- To implement functions f<sub>1</sub> and f<sub>2</sub> we require 3 × 5 × 2 PLA and we have to implement them using 3 × 4 × 2 PLA. Therefore, we have to examine product terms by grouping 0s instead of 1. That is product terms for complement of a function.
- $f_1 = a_2 \bar{a}_0 + a_1 \bar{a}_0 + a_2 a_1$
- $\bar{f}_1 = \bar{a}_2 \bar{a}_1 + a_1 \bar{a}_0 + a_2 \bar{a}_0$

#### Step 2 : Implementation

- Looking at function outputs we can realize that product terms  $a_2 \bar{a}_0$  and  $a_1 \bar{a}_0$  are common in both functions. Therefore, we need only 4 product terms and functions

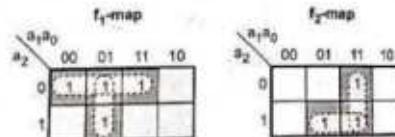


Fig. 7.13.8 K-map simplification

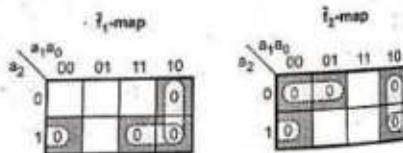


Fig. 7.13.9

can be implemented using a 3 × 4 × 2 PLA as shown in Table 7.13.2 and Fig. 7.13.10.

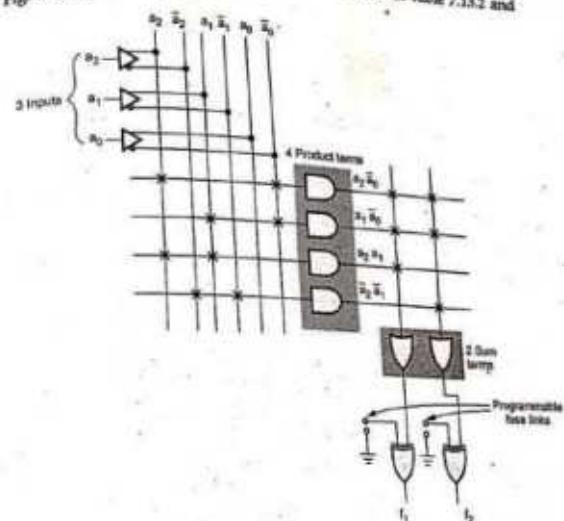


Fig. 7.13.10 Logic diagram

PLA

Product terms	Inputs			Outputs	
	a <sub>2</sub>	a <sub>1</sub>	a <sub>0</sub>	f <sub>1</sub>	f <sub>2</sub>
a <sub>2</sub> a <sub>0</sub>	1	—	—	0	1
a <sub>1</sub> a <sub>0</sub>	—	1	—	1	1
a <sub>2</sub> a <sub>1</sub>	1	—	—	—	1
a <sub>2</sub> a <sub>1</sub>	0	—	—	—	1

Table 7.13.2

As shown in the Fig. 7.13.10 exclusive-OR gate is programmed to invert the function to get the desired function outputs.

**Example 7.13.4** A combinational circuit is defined by the function,

$$f_1 = \sum m(1, 3, 5), \quad f_2 = \sum m(5, 6, 7)$$

Implement the circuit with a PLA having 3 inputs, 3 product terms and two outputs.

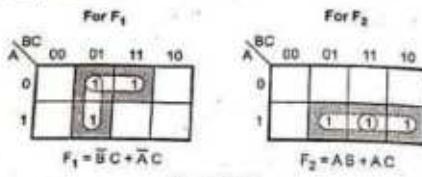
**Solution : K-map simplification**

Fig. 7.13.11

- To implement functions  $F_1$  and  $F_2$  we require  $3 \times 4 \times 2$  PLA and we have to implement them using  $3 \times 3 \times 2$  PLA. Therefore we have to examine product terms by grouping 0s instead of 1s. That is product terms for complement of a function.

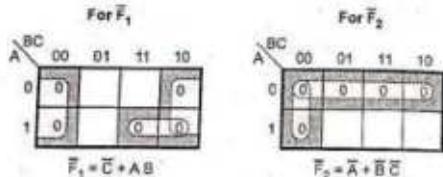


Fig. 7.13.12

- Looking at function outputs, functions  $\bar{F}_1$  and  $\bar{F}_2$  have one common product term. Thus they have total 3 product terms and can be implemented using  $3 \times 3 \times 2$  PLA.

**PLA program table**

Product terms	Inputs			Outputs	
	A	B	C	$F_1$	$F_2$
$\bar{C}$	-	-	0	1	-
$AB$	1	1	-	1	1
$AC$	1	-	1	-	1
				C	T

Table 7.13.3

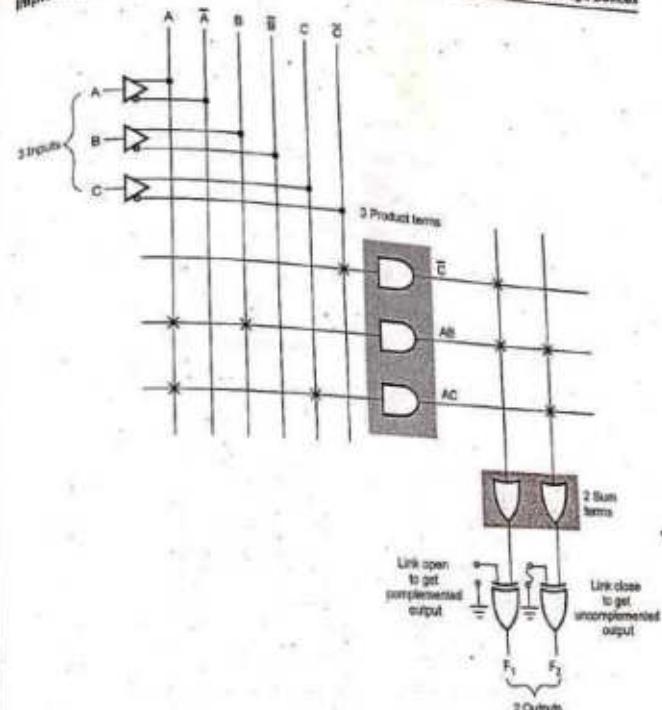
**Implementation**

Fig. 7.13.13

**Example 7.13.5** Design a BCD to Excess-3 code converter and implement using standard PLA.

**Solution :**

**Step 1 :** Derive the truth table of BCD to Excess-3 converter.

Decimal	BCD code				Excess-3 code			
	B <sub>3</sub>	B <sub>2</sub>	B <sub>1</sub>	B <sub>0</sub>	E <sub>3</sub>	E <sub>2</sub>	E <sub>1</sub>	E <sub>0</sub>
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	2	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	1	0	0

Table 7.13.4 Truth table for BCD to Excess-3 code converter

Step 2 : Simplify the Boolean functions for Excess-3 code

For E <sub>3</sub>	
B <sub>3</sub>	B <sub>2</sub>
00	0 0 0 0
01	0 1 1 1
11	X X X X
10	1 1 X X

$$E_3 = B_3 + B_2 B_0 + B_2 B_1$$

For E <sub>2</sub>	
B <sub>3</sub>	B <sub>2</sub>
00	0 0 1 1
01	1 0 0 0
11	X X X X
10	0 1 X X

$$E_2 = \bar{B}_2 \bar{B}_1 \bar{B}_0 + \bar{B}_2 B_0 + \bar{B}_2 B_1$$

For E <sub>1</sub>	
B <sub>3</sub>	B <sub>2</sub>
00	1 0 1 0
01	1 0 1 0
11	X X X X
10	1 0 X X

$$E_1 = \bar{B}_1 \bar{B}_0 + B_1 B_0$$

Fig. 7.13.14

For E <sub>0</sub>	
B <sub>3</sub>	B <sub>2</sub>
00	1 0 0 1
01	1 0 0 1
11	X X X X
10	1 0 X X

$$E_0 = \bar{B}_1 \bar{B}_0 + B_1 \bar{B}_0$$

Step 3 : Write PLA program table

Product terms	Inputs								Outputs			
	B <sub>3</sub>	B <sub>2</sub>	B <sub>1</sub>	B <sub>0</sub>	E <sub>3</sub>	E <sub>2</sub>	E <sub>1</sub>	E <sub>0</sub>	B <sub>3</sub>	B <sub>2</sub>	B <sub>1</sub>	B <sub>0</sub>
B <sub>3</sub>	1	-	-	-	-	-	-	-	1	-	-	-
B <sub>2</sub> B <sub>0</sub>	2	-	-	-	1	-	-	-	1	1	-	-
B <sub>2</sub> B <sub>1</sub>	3	-	-	-	1	1	-	-	1	1	-	-
B <sub>2</sub> B <sub>1</sub> B <sub>0</sub>	4	-	-	-	1	0	0	-	1	-	-	-
B <sub>2</sub> B <sub>0</sub>	5	-	-	-	0	-	-	-	1	-	-	-
B <sub>2</sub> B <sub>1</sub>	6	-	-	-	0	1	-	-	1	-	-	-
B <sub>1</sub> B <sub>0</sub>	7	-	-	-	0	0	-	-	1	-	-	-
B <sub>1</sub> B <sub>0</sub>	8	-	-	-	1	1	-	-	1	-	-	-
B <sub>1</sub> B <sub>0</sub>	9	-	-	-	1	0	-	-	1	-	-	-

Table 7.13.5 PLA program table

Step 4 : Implementation

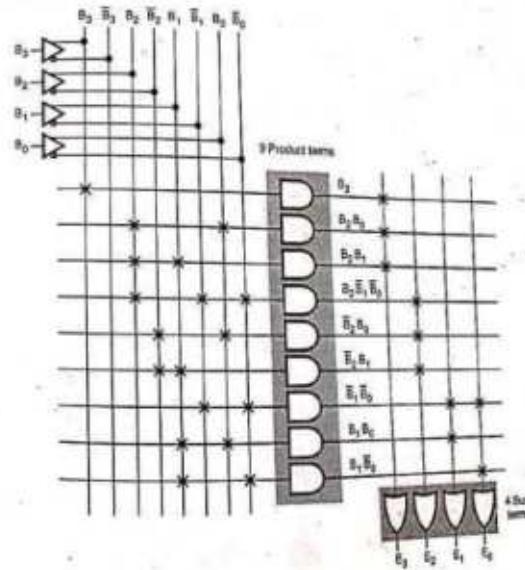


Fig. 7.13.15

**Example 7.13.6** Implement the following function using PLA:

$$f_1 = \sum m(0, 3, 4, 7)$$

$$f_2 = \sum m(1, 2, 5, 7)$$

**Solution : K-map simplification**

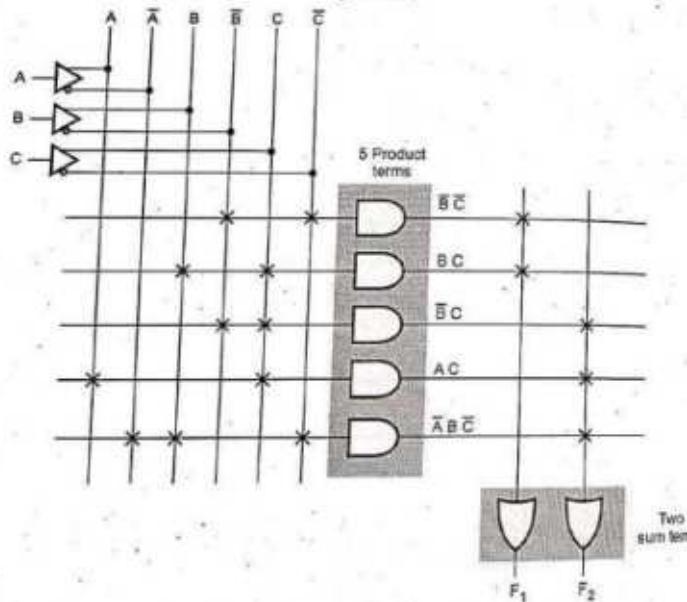
		BC	00	01	11	10
		0	1		1	
		1	1		1	
A	BC	00	01	11	10	
0	1					
1	1					

$$f_1 = \overline{B}\overline{C} + BC$$

		BC	00	01	11	10
		0		1		1
		1		1	1	
A	BC	00	01	11	10	
0						
1						

$$f_2 = \overline{B}C + AC + \overline{A}BC$$

Fig. 7.13.16



**Example 7.13.7** Design using PLD a 3 : 8 decoder.

**Solution :** The Fig. 7.13.18 shows 3 : 8 decoder using PLD.

**Example 7.13.8** A combinational logic circuit is defined by the functions  $F_1 = \sum m(3, 5, 6, 7)$  and  $F_2 = \sum m(0, 2, 4, 7)$ . Implement the circuit with a PLA having three inputs, four product terms and two outputs.

GTU : Winter-16, Marks 7

**Solution : Step 1 : Simplify the Boolean functions**

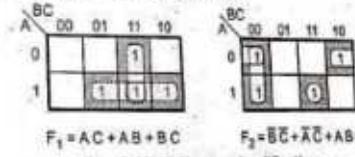


Fig. 7.13.19 K-map simplification

Here, we have 6 product terms so we check for complement functions.

If we take  $f_1$  and  $f_2$  outputs of two functions, we need only four product terms since product terms  $\overline{B}\overline{C}$  and  $\overline{A}\overline{C}$  are common between them.

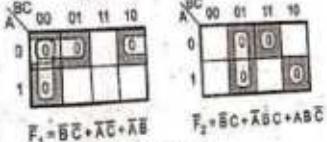


Fig. 7.13.20

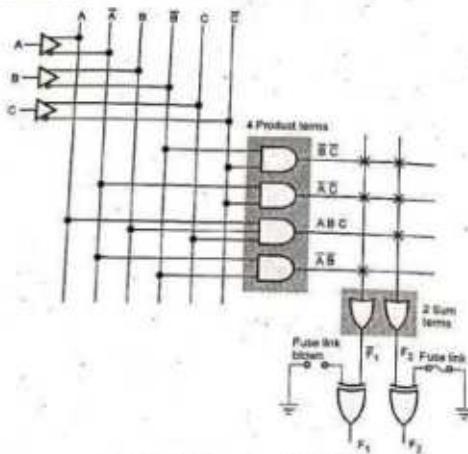
**Step 2 : Implementation**

Fig. 7.13.21 Logic diagram

**Review Questions**

1. Explain PLA in detail.
2. Explain PLA with necessary diagrams.

GTU : May-13, Marks 7

GTU : Dec.-12, Marks 7

**7.14 Programmable Array Logic (PAL)**

- PAL programmable array logic is a programmable logic device with a fixed OR array and a programmable AND array.
- Because only AND gates are programmable, the PAL is easier to program, but is not as flexible as the PLA.
- Fig. 7.14.1 shows the array logic of a typical PAL.
- It has four inputs and four outputs.
- Each input has buffer and an inverter gate.
- Two gates are shown with one composite graphic symbol with normal and complement outputs.
- There are four sections. Each section has three programmable AND gates and one fixed OR gate.

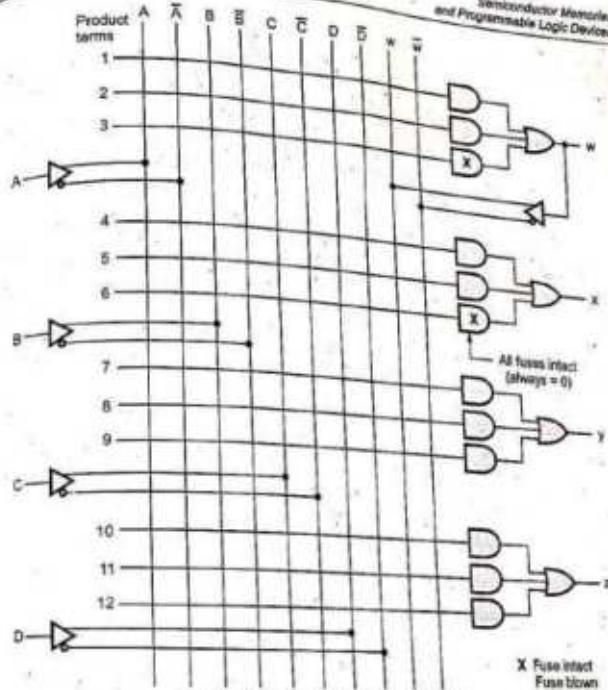


Fig. 7.14.1 Array logic for typical PAL

- The output of section 1 is connected to a buffer-inverter gate and then fed back into the inputs of the AND gates, through fuses. This allows the logic designer to feed an output function back as an input variable to create a new function. Such PALs are referred to as Programmable I/O PALs.
- The commercial PAL devices have more gates than the one shown in Fig. 7.14.1.
- A typical PAL integrated circuit may have eight inputs, eight outputs, and eight sections, each consisting of an eight wide AND-OR array.

**7.14.1 Implementation of Combinational Logic Circuit using PAL**

- Let us see the implementation of a combinational circuit using PAL with the help of examples.

**Illustrative Examples****Example 7.14.1** Implement the following Boolean functions using PAL.

$$w(A, B, C, D) = \sum m(0, 2, 6, 7, 8, 9, 12, 13)$$

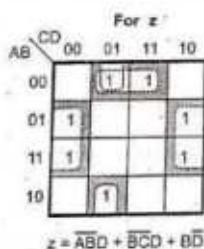
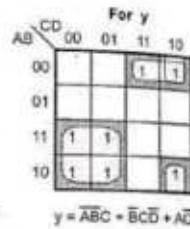
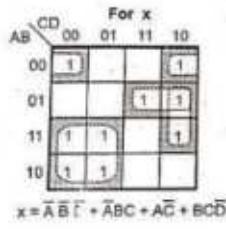
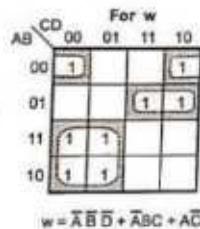
$$x(A, B, C, D) = \sum m(0, 2, 6, 7, 8, 9, 12, 13, 14)$$

$$y(A, B, C, D) = \sum m(2, 3, 8, 9, 10, 12, 13)$$

$$z(A, B, C, D) = \sum m(1, 3, 4, 6, 9, 12, 14)$$

**Solution :****Step 1 :** Simplify the four functions

- Note that function  $x$  has four product terms. Three of them are equal to  $w$ . Therefore we can write  $x = w + BCD$ .

**Fig. 7.14.2 K-map simplification****Step 2 :** Implementation

- In the last section we have seen the PLA program table. The program table for PAL is similar to PLA program table. Table 7.14.1 shows PAL program table with product terms, AND inputs and outputs.

**Table 7.14.1 PAL program table**

Product term	AND Inputs				Output
	A	B	C	D	
1	0	0	-	0	-
2	0	1	1	-	-
3	1	-	0	-	-
4	-	-	-	-	1
5	-	1	1	0	-
6	-	-	-	-	-
7	0	0	1	-	-
8	-	0	1	0	-
9	1	-	0	-	-
10	0	0	-	1	-
11	-	0	0	1	-
12	-	1	-	0	-

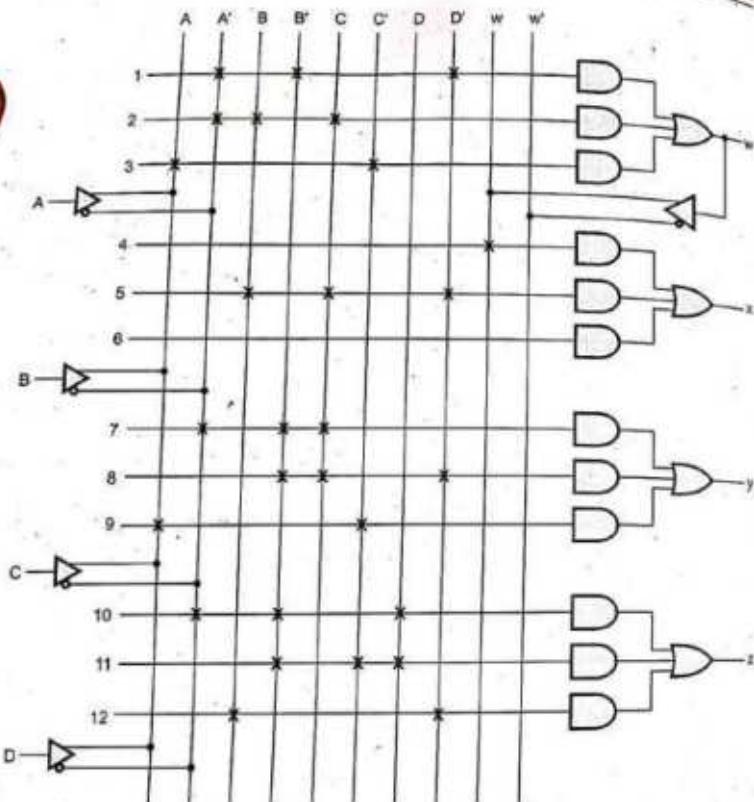


Fig. 7.14.3 Logic diagram

**Example 7.14.2** Design BCD to Excess-3 converter using PAL.

**Solution :**

**Step 1 :** Derive the truth table of BCD to Excess-3 converter

Decimal	BCD code				Excess-3 code			
	B <sub>3</sub>	B <sub>2</sub>	B <sub>1</sub>	B <sub>0</sub>	E <sub>3</sub>	E <sub>2</sub>	E <sub>1</sub>	E <sub>0</sub>
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	0
3	0	0	1	1	0	1	0	1
4	0	1	0	0	1	1	0	0
5	0	1	0	1	0	1	1	1
6	0	1	1	0	1	0	0	0
7	0	1	1	1	1	0	0	1
8	1	0	0	0	0	1	0	0
9	1	0	0	1	1	1	0	0

Table 7.14.2 Truth table for BCD to Excess-3 code converter

**Step 2 :** Simplify the Boolean functions for Excess-3 code outputs.

For E<sub>3</sub>

B <sub>3</sub> B <sub>2</sub>	00	01	11	10
00	0	0	0	0
01	0	1	1	1
11	X	X	X	X
10	1	1	X	X

$$E_3 = B_3 + B_2B_0 + B_2B_1$$

For E<sub>2</sub>

B <sub>3</sub> B <sub>2</sub>	00	01	11	10
00	0	1	1	1
01	1	0	0	0
11	X	X	X	X
10	0	1	(X)	(X)

$$E_2 = B_2\bar{B}_1\bar{B}_0 + \bar{B}_2B_0 + \bar{B}_2B_1$$

For E<sub>1</sub>

B <sub>3</sub> B <sub>2</sub>	00	01	11	10
00	1	0	1	0
01	1	0	1	0
11	X	X	X	X
10	1	0	X	X

$$E_1 = B_1\bar{B}_0 + B_1\bar{B}_1$$

For E<sub>0</sub>

B <sub>3</sub> B <sub>2</sub>	00	01	11	10
00	1	0	0	1
01	1	0	0	1
11	X	X	X	X
10	1	0	X	X

$$E_0 = \bar{B}_1\bar{B}_0 + B_1\bar{B}_1$$

Fig. 7.14.4 K-map simplification

**Step 3 : Implementation**

Product terms	Inputs				Outputs			
	B <sub>3</sub>	B <sub>2</sub>	B <sub>1</sub>	B <sub>0</sub>	E <sub>3</sub>	E <sub>2</sub>	E <sub>1</sub>	E <sub>0</sub>
B <sub>3</sub>	1	1	-	-	1	-	-	-
B <sub>2</sub> B <sub>0</sub>	2	-	1	-	1	1	-	-
B <sub>2</sub> B <sub>1</sub>	3	-	1	1	-	1	-	-
B <sub>2</sub> B <sub>1</sub> B <sub>0</sub>	4	-	1	0	0	-	1	-
B <sub>2</sub> B <sub>0</sub>	5	-	0	-	1	-	1	-
B <sub>2</sub> B <sub>1</sub>	6	-	0	1	-	-	1	-
B <sub>1</sub> B <sub>0</sub>	7	-	-	0	0	-	-	1
B <sub>1</sub> B <sub>0</sub>	8	-	-	1	1	-	-	1
B <sub>1</sub> B <sub>0</sub>	9	-	-	1	0	-	-	1

Table 7.14.3 PAL program table

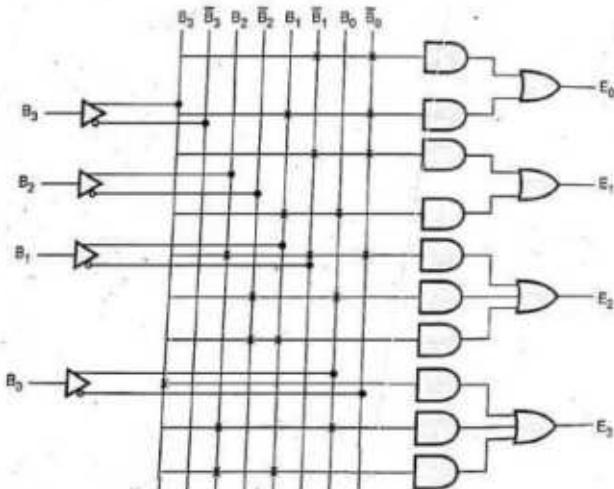


Fig. 7.14.5 Logic diagram

**Example 7.14.3** Generate the following Boolean functions with a PAL with 4 inputs and 4 outputs.

$$Y_3 = \overline{ABCD} + \overline{ABC}\overline{D} + A\overline{BC}\overline{D}$$

$$Y_2 = \overline{ABC}\overline{D} + \overline{ABC} + ABC + A\overline{B}\overline{C}$$

$$Y_1 = \overline{ABC} + \overline{ABC} + A\overline{B}\overline{C} + ABC$$

$$Y_0 = ABCD$$

**Solution 1 :****Step 1 :** Simplify the Boolean functions

$$Y_3 = \overline{AB}\overline{C}\overline{D} + \overline{ABC}\overline{D} + A\overline{B}\overline{C}\overline{D} = (\overline{A} + A)\overline{B}\overline{C}\overline{D} + \overline{ABC}\overline{D}$$

$$= B\overline{C}\overline{D} + \overline{ABC}\overline{D}$$

$$Y_2 = \overline{ABC}\overline{D} + \overline{ABC} + ABCD + \overline{ABC}(\overline{D} + D) + (\overline{A} + A)(BCD)$$

$$= \overline{ABC} + BCD$$

$$Y_1 = \overline{ABC} + \overline{ABC} + A\overline{B}\overline{C} + ABC$$

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

$$Y_0 = ABCD$$

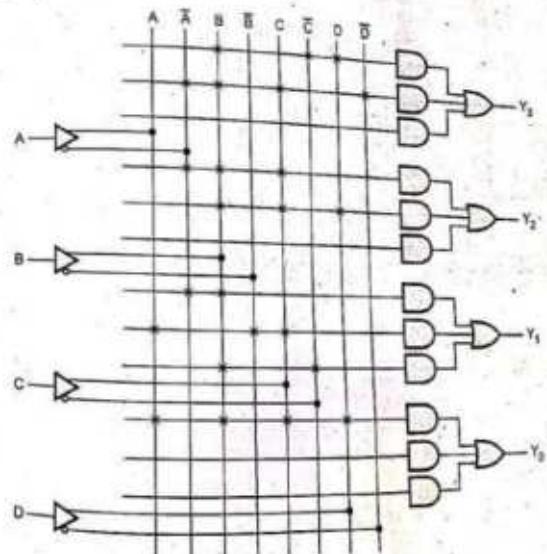
**Step 2 : Implementation**

Fig. 7.14.6 Logic diagram

**Example 7.14.4** Implement 4 : 1 multiplexer using PAL.

**Solution :** Refer section 4.10.2. The Boolean function for 4 : 1 multiplexer is

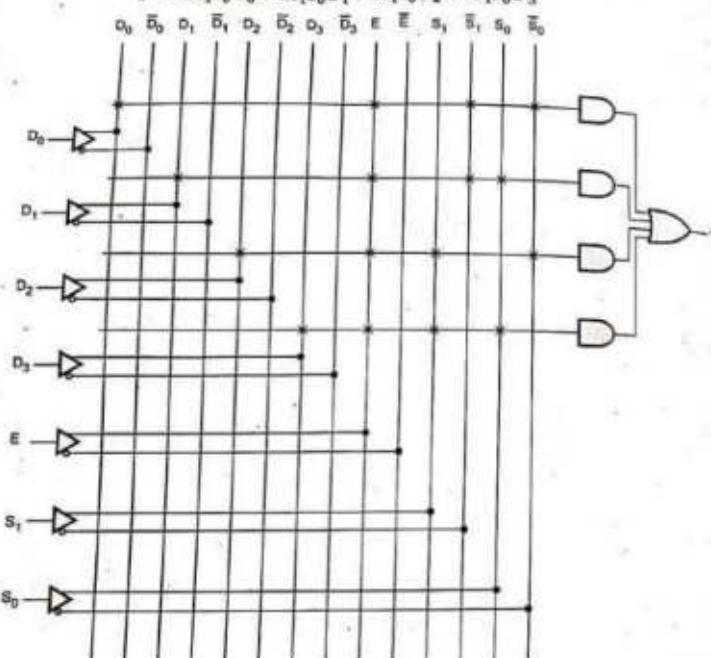
$$Y = E\bar{S}_1\bar{S}_0D_0 + E\bar{S}_1S_0D_1 + ES_1\bar{S}_0D_2 + ES_1S_0D_3$$


Fig. 7.14.7

**Examples for Practice**

**Example 7.14.5 :** A combinational logic circuit is defined by the following function.

$f_1(a, b, c) = S(0, 1, 6, 7)$ ,  $f_2(a, b, c) = S(2, 3, 5, 7)$ . Implement the circuit with a PAL having three inputs, three product terms and two outputs.

**Example 7.14.6 :** Use 5 input, 3 product, 4 output PAL to realize following 3 functions.

$$F_1(A, B, C, D) = \Sigma m(2, 3, 4, 8, 9, 13, 14)$$

$$F_2(A, B, C, D) = \Sigma m(2, 3, 5, 8, 9, 14, 15)$$

$$F_3(A, B, C, D) = \Sigma m(2, 3, 4, 8, 9, 14, 15)$$

**7.15 Comparison between PROM, PLA and PAL**

Sr. No.	PROM	PLA	PAL
1	AND array is fixed and OR array is programmable.	Both AND and OR arrays are programmable.	OR array is fixed and AND array is programmable.
2	Cheaper and simple to use.	Costliest and complex than PAL and PLCDs.	Cheaper and simpler.
3	All minterms are decoded.	AND array can be programmed to get desired minterms.	AND array can be programmed in get desired minterms.
4	Only Boolean functions in standard SOP form can be implemented using PROM.	Any Boolean functions in SOP form can be implemented using PLA.	Any Boolean functions in SOP form can be implemented using PLA.

**7.16 Complex Programmable Logic Devices (CPLDs)**

Simple PLDs (SPLDs) include PLAs, PALs and other similar types of devices.

• SPLDs have limitations of number of input product terms and outputs. For applications which requires more number of inputs or product terms or output we have to expand the capacity of PLDs by cascading them.

• The Complex Programmable Logic Devices (CPLDs) are introduced to solve the above mentioned difficulty of SPLDs. A typical CPLD is merely a collection of multiple PLDs and an interconnection structure, all on the same chip, as shown in the Fig. 7.16.1.

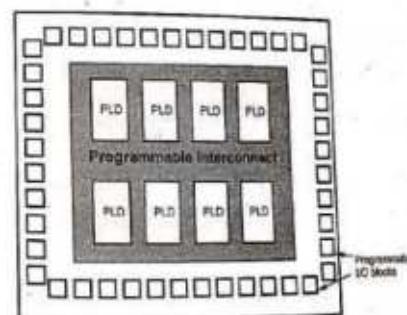


Fig. 7.16.1 General architecture of CPLD

In CPLDs, in addition to the individual PLDs the on-chip interconnection structure is also programmable. Therefore, unlike PLDs, the CPLDs can be scaled to larger sizes by increasing the number of individual PLDs.

**7.16.1 Block Diagram**

- The Fig. 7.16.2 shows the block diagram of a Complex Programmable Logic Device (CPLD).

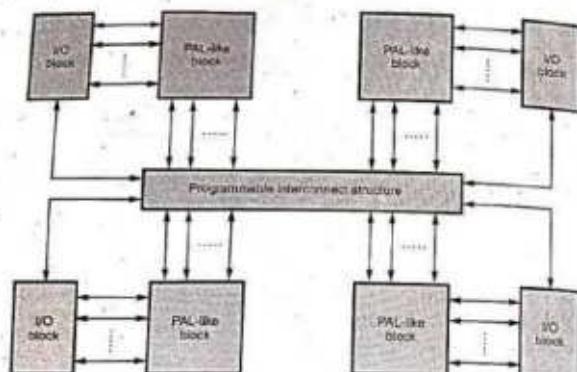


Fig. 7.16.2 Block diagram of CPLD

- It consists of collection of PAL like blocks, I/O blocks and a set of interconnection wires, called programmable interconnection structure.
- The PAL like blocks are connected to the programmable interconnect structure and to the I/O blocks. The chip input-output pins are attached to the I/O blocks.
- A PAL like block in the CPLD usually consists of about 16 macrocells. Like other macrocells, the macrocell in CPLD consists of AND-OR configuration, an EX-OR gate, a flip-flop, a multiplexer, and a tri-state buffer.
- The Fig. 7.16.3 shows the typical macrocell for CPLD. Each AND-OR configuration usually consists of 5-20 AND gates and an OR gate with 5-20 inputs.
- The EX-OR gate provides the output of OR-gate in inverted or non-inverted form as per the fuse link status.
- A D flip-flop stores the output of EX-OR gate.
- A multiplexer selects either the output of the D flip-flop or the output of the EX-OR gate depending upon its select input (either 1 or 0).
- The tri-state buffer acts as a switch which enables or disables the output.

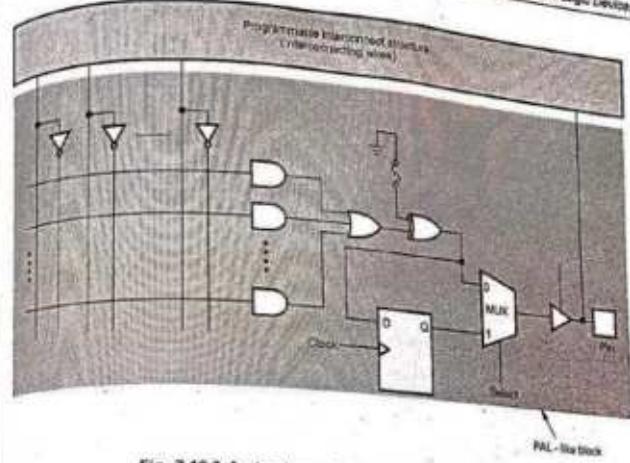


Fig. 7.16.3 A structure of typical macrocell of a CPLD

**7.16.2 Architecture of XC9572 CPLD**

- The XC9572 is a device from XC9500 CPLD family. It consists of 4 functional blocks (FBs), 72 macrocells, 1600 usable gates and 72 registers. The XC9572 device is available in several different packages. Different I/O packages have different number of I/O pins. For example, 44-pin PLCC has 34 I/O pins and 100-pin TQFP has 72 I/O pins.
- In XC9572, the functional blocks and I/O blocks are fully interconnected by the FastCONNECT switch matrix. The IOB provides buffering for device inputs and outputs. Each FB provides programmable logic capacity with 36 inputs and 18 outputs. The Fast CONNECT switch matrix connects all FB outputs and input signals to the FB inputs. For each FB, 12 to 18 outputs (depending on package pin-count) and associated output enable signals drive directly to the IOBs. This is illustrated in Fig. 7.16.4.

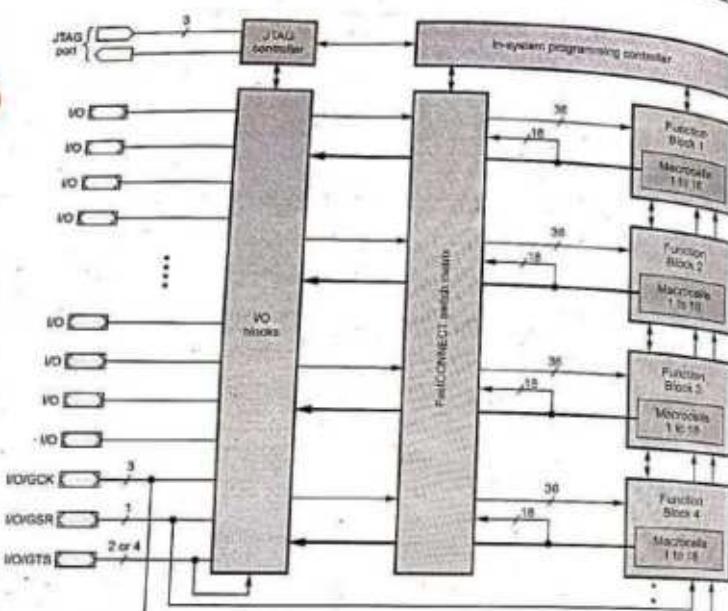


Fig. 7.16.4 XC9500 architecture

**Review Questions**

1. State the limitation of SPLD.
2. Draw and explain the general architecture of CPLD.
3. What it meant by CPLD ? Draw and explain the block diagram of CPLD.
4. Explain the architectural features of XC9572.
5. Draw the basic structure of CPLD. Explain its features in brief.

**Field Programmable Gate Array (FPGA)**

- Field Programmable Gate Arrays (FPGA) provide the next generation in the programmable logic devices.
- The word **field** in the name refers to the ability of the gate arrays to be programmed for a specific function by the user instead of by the manufacturer of the device.
- The word **array** is used to indicate a series of columns and rows of gates that can be programmed by the end user.
- As compared to standard gate arrays, the field programmable gate arrays are larger devices.
- The basic cell structure for FPGA is somewhat complicated than the basic cell structure of standard gate array.
- The programmable logic blocks of FPGAs are called logic blocks or Configurable Logic Blocks(CLBs).
- The basic architecture of FPGA consists of an array of logic blocks with programmable row and column interconnecting channels surrounded by programmable I/O blocks as shown in Fig. 7.17.1.

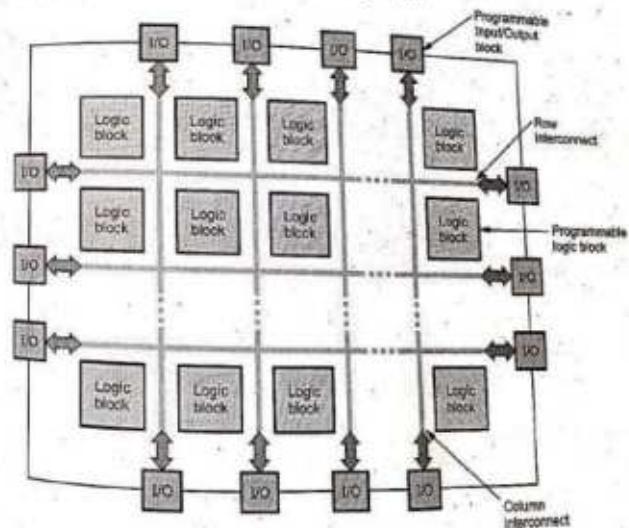


Fig. 7.17.1 Basic architecture of FPGA

- Many FPGA architectures are based on a type of memory called LUT (look-up table) rather than on (sum of product) SOP AND/OR arrays as CPLDs are.
  - Another approach found on some FPGAs is the use of multiplexers to select between different memory blocks.

147

- It is the look-up table used in FPGAs.
  - It is actually a memory device that can be programmed to perform logic functions.
  - The LUT essentially replaces the AND/OR array logic in a CPLD.
  - Fig. 7.17.2 shows a simple diagram of an 8 bit by 1 bit ( $8 \times 1$ ) memory programmed to produce to SOP function  $\overline{ABC} + ABC + ABC$ .
  - When any one of the three product terms appears on the LUT inputs, the corresponding memory cell storing a 1 is selected and the 1 (HIGH) appears on the output.
  - For any product terms that are not part of the SOP function, the LUT output is 0 (LOW).

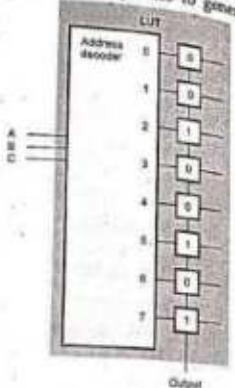
#### **The Logic Block :**

- Each logic block in a generic FPGA contains several logic elements, as shown in Fig. 7.17.3.
  - Generally there can be well over ten thousand logic elements in a single chip.

### The Logic Element :

- A simplified diagram of a typical FPGA logic element is shown in Fig. 7.17.4. It contains an LUT, associated logic, and a flip-flop.

Fig. 7.17.2 An LUT programmed to produce the SOP function



The diagram illustrates a logic block structure. It features a vertical stack of rectangular boxes labeled "Logic element 1", "Logic element 2", "Logic element 3", and "Logic element n". A vertical double-headed arrow on the right side connects these elements to a thick, textured vertical bar labeled "Column interconnects". At the bottom, a horizontal double-headed arrow connects the bottom of the stack to another thick, textured vertical bar labeled "Row interconnects".

TECHNICAL PUBLICATIONS™ - An imprint for professionals

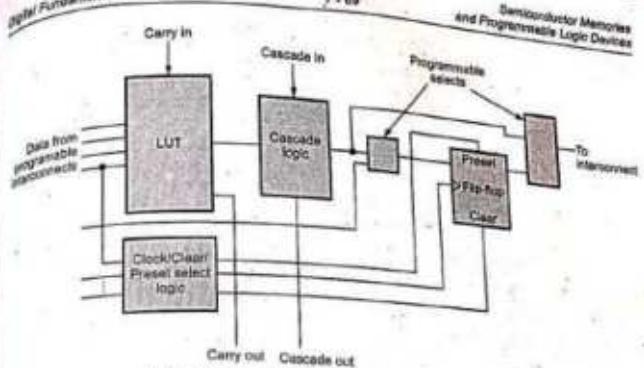


Fig. 7.17.4 A simplified typical FPGA logic slice

- Each logic element contains a 4-input LUT that can be programmed as logic function generator.
  - It can be used to produce SOP functions or logic functions such as adders and comparators.
  - When configured as an adder, the carry in and carry out allow for adder expansion.
  - Using the cascade logic, an LUT can be expanded by cascading with LUTs in other logic elements.
  - The programmable selects let you choose either combinational functions from the LUT output or registered functions from the flip-flop output.

**Example 7.17.1** Explain two input LUT with implementation of the function :

$$F(A, B) = \sum_{m=0}^3$$

Solution : Two input LUT has  $2^2 \times 1$ -bit memory i.e.

The input LUT has  $2 \times 1$ -bit memory, i.e.  $1 \times 1$ -bit memory. We can implement the given function with 2-input LUT as shown in the Fig. 7.175.

The bit corresponding of the two-inputs of one bit wide memory array are available at the output of LUT. For example, if inputs AB are 00, the bit from first location is available at the output. Storing logic 1 at bit 0 and bit 3 positions, and storing logic 0 at bit 1 and bit 2 positions we can implement the given function.

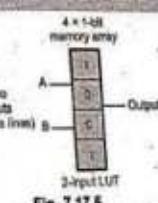
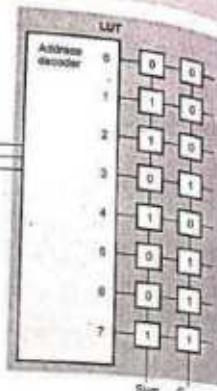


Fig. 7.17.J

**Example 7.17.2** Explain the three input LUT with implementation of a full adder.

**Solution :** Fig. 7.17.6 shows an  $8 \times 2$  LUT is programmed as a full adder. The sum and carry out expressions for a full-adder are as follows :

$$\begin{aligned} \text{Sum} &= (A + B) + C_{in} \\ &= \bar{A} \bar{B} C_{in} + \bar{A} B \bar{C}_{in} + A \bar{B} \bar{C}_{in} + ABC_{in} \\ C_{out} &= AB + (A + B)C_{in} \\ &= \bar{A} BC_{in} + A \bar{B} C_{in} + ABC_{in} \end{aligned}$$

**Fig. 7.17.6** An LUT programmed as full-adder.**Interconnection Technology**

FPGAs use either SRAM or antifuse methods to provide interconnections between logic blocks. The antifuse is normally open and is shorted to create a connection when programmed, compared to the fuse which is normally shorted and is opened to create no connection when programmed.

In SRAM method, a transistor is controlled by the state of an on-chip SRAM cell. When SRAM cell contains zero, the transistor is off and connection is open. When SRAM cell contains one, the transistor is ON and connection is shorted.

**Review Questions**

- What is FPGA ? What are its salient features ?
- Draw and explain the structural block diagram of FPGA.
- What do you mean by FPGA ? Explain the details of internal architecture of FPGA. How will you design any digital circuit with the help of FPGA ?
- With the help of structural block diagram explain FPGA with respect to :
  - Storage cell content in LUT
  - Three input LUT
  - Inclusion of a flip-flop in an FPGA logic.
- What is LUT in FPGA ?
- Write a short note on FPGA.
- Draw and explain basic architecture of FPGA. State difference between PLA and FPGA.
- What are registered programmable devices ?
- Draw and explain in brief general architecture of Xilinx FPGA.

GTU : Winter-15, Marks 3

**J.15 Comparison between CPLDs and FPGAs**

No.	CPLD	FPGA
1.	Fig. 7.16.1 shows basic architecture of CPLD.	Fig. 7.17.1 shows the basic architecture of FPGA.
2.	They consist of PAL like blocks, I/O blocks and programmable interconnect structures.	They consist of Configurable Logic Blocks (CLBs), I/O blocks, row interconnect and column interconnect.
3.	They use AND/OR arrays to generate logic functions.	They use memory called LUT (Look-up Table) or multiplexers to generate logic functions.
4.	They are programmed for a specific function by the manufacturer of the device.	They are programmed for specific function by the user.
5.	It is more suitable in small gate count designs.	It is more suitable in high gate count designs since it has very high logic capacity.
6.	Less complex architecture.	More complex architecture.
7.	Delays are more predictable.	Delays are quite unpredictable.
8.	CPLD contains a few blocks of logic that reaches up to a few thousands.	FPGA contains upto 100 000 of tiny logic blocks.
9.	It is considered as coarse-grain device.	It is considered as a fine-grain device.
10.	CPLDs are made of larger blocks.	FPGAs are made up of tiny logic blocks.
11.	CPLD is EPROM-based digital logic chip.	FPGA is RAM based digital logic chip.
12.	It is much cheaper.	It is more expensive.
13.	Example : XC 9500 family.	Example : Xilinx 4000 family

**Review Question**

- Explain difference between FPGA and CPLD.

## Oral Questions and Answers

**Q.1 Define a memory cell. Give an example.**

**Ans. :** Memories are made up of registers. Each register consists of storage elements (flip-flops or capacitors in semiconductor memories and magnetic domain in magnetic storage), each of which stores one bit of data. Such a storage element is called a memory cell.

**Q.2 Define a memory location.**

**Ans. :** Memories are made up of registers. Each register in the memory is one storage location also called memory location. Each memory location is identified by an address.

**Q.3 Explain 'write operation' with an example. (Refer section 7.1)**

**Q.4 What is a volatile memory? Give example.**

**Ans. :** The memory which cannot hold data when power is turned off is known as volatile memory. The static RAM is a volatile memory.

**Q.5 Name the types of ROM.**

**Ans. :** There are four types of ROM : Masked ROM, PROM, EPROM and EEPROM or E<sup>2</sup>PROM.

**Q.6 What is non-volatile memory? (Refer section 7.2)**

**Q.7 Which memory is called volatile? Why? (Refer section 7.2)**

**Q.8 What is meant by 'static' and 'dynamic' memories? (Section 7.3)**

**Q.9 Explain static memory. (Section 7.6.1)**

**Q.10 Define address and word.**

**Ans. :** In a ROM, each bit combination of the input variable is called an address. Each bit combination that comes out of the output lines is called a word.

**Q.11 Explain ROM.**

**Ans. :** A Read Only Memory (ROM) is a device that includes both the decoder and the OR gates within a single IC package. It consists of  $n$  input lines and  $m$  output lines. Each bit combination of the input variables is called an address. Each bit combination that comes out of the output lines is called a word. The number of distinct addresses possible with  $n$  input variables is  $2^n$ .

**Q.12 How does ROM retain information?**

**Ans. :** ROMs are hardwired devices. It contains memory array of programmed to retain information.

**Q.13 Define PROM.**

**Ans. :** PROM is Programmable Read Only Memory. It consists of a set of fixed AND gates connected to a decoder and a programmable OR array.

**Q.14 How is individual location in a EEPROM programmed or erased?**

**Ans. :** Since it is electrically erasable memory, by activating particular row and column it is possible that individual can be programmed or erased.

**Q.15 Mention the two types of erasable PROM. (Refer sections 7.5.2 and 7.5.3)**

**Q.16 Whether ROM is classified as nonvolatile storage device? Why? (Refer section 7.5)**

**Q.17 Write the advantages of EPROM over PROM. (Refer section 7.5.3)**

**Q.18 What is EEPROM. (Refer section 7.5.4)**

**Q.19 Briefly explain about EEPROM. (Refer section 7.5.3)**

**Q.20 What is RAM?**

**Ans. :** Random Access Memory. Read and write operation can be carried out. It is a volatile memory. It can hold data as power is ON.

**Q.21 Give the advantages of RAM.**

**Ans. :** Advantages of RAM are :

1. RAM is a memory where we can read as well as write the data.

2. It is random access memory i.e. the particular memory location can be accessed using the address of the at memory location.

3. Higher speed.

**Q.22 What is read and write operation?**

**Ans. :** The write operation stores data into a specified address into the memory and the read operation takes data out of a specified address in the memory.

**Q.23 List the two categories of RAM. (Refer section 7.6)**

**Q.24 Define dynamic RAM.**

**Ans. :** Dynamic RAMs use capacitors as storage elements and cannot retain data very long without capacitors being recharged by a process called refreshing.

**Q.25** List the two types of SRAM.**Ans. :** Asynchronous SRAMs and Synchronous Burst SRAMs.**Q.26** List the basic types of DRAMs.**Ans. :** Fast page Mode DRAM, Extended Data Out DRAM (EDO DRAM), Burst EDO DRAM and Synchronous DRAM.**Q.27** Draw the basic dynamic memory cell. (Refer section 7.6)**Q.28** Draw the logic diagram of a memory cell. (Refer section 7.6)**Q.29** Draw the block diagram of dynamic RAM cell. (Refer section 7.6)**Q.30** Define PLD.**Ans. :** Programmable Logic Devices consist of a large array of AND gates and OR gates that can be programmed to achieve specific logic functions.**Q.31** Give the classification of PLDs.**Ans. :** PLDs are classified as :

- PROM(Programmable Read Only Memory),
- Programmable Logic Array(PLA),
- Programmable Array Logic (PAL), and
- Generic Array Logic(GAL).

**Q.32** What is a PLA?**Ans. :** PLA stands for Programmable Logic Array, which is a LSI component. In PLA, both AND and OR gates have fuses at the inputs, therefore in PLA both AND and OR gates are programmable. The outputs from OR gates go through fuses as inputs to output inverters so that final output can be programmed as either AND-OR or AND-OR-INVERT.**Q.33** Why was PAL developed?**Ans. :** PAL is a PLD that was developed to overcome certain disadvantages of PLA, such as longer delays due to additional fusible links that result from using two programmable arrays and more circuit complexity.**Q.34** Why the input variables to a PAL are buffered?**Ans. :** The input variables to a PAL are buffered to prevent loading by the large number of AND gate inputs to which available or its complement can be connected.**Q.35** Mention few applications of PLA and PAL.**Ans. :** Applications of PLA and PAL are :**Q.36** What is FPGA?**Ans. :** FPGA stands for field programmable gate array, which is the next generation in the programmable logic devices. The word field refers to the ability of the gate arrays to be programmed for a specific function by the end user. The word array indicates a series of columns and rows of gates that can be programmed by the end user.**Q.37** List the configurable elements in the FPGA architecture.**Ans. :** The FPGA architecture consists of three types of configurable elements :

1. A perimeter of Input/Output Blocks (IOBs).
2. A core array of Configurable Logic Blocks (CLBs).
3. Resources for interconnection.

**Q.38** How does the architecture of a PLA different from a PROM?**Ans. :** Refer section 7.15.**Q.39** What is PAL? How does it differ from PLA?**Ans. :** Refer sections 7.14 and 7.15.

**SOLVED MODEL QUESTION PAPER**

[As per New Question Paper Pattern]  
 Digital Fundamentals  
 Semester - III (CE/IT)

Time :  $2 \frac{1}{2}$  Hours]

[Total Marks : 70]

## Instructions :

- 1) Attempt all questions.
- 2) Make suitable assumptions wherever necessary.
- 3) Figures to the right indicate full marks.

- Q.1** a) Convert decimal number  $(0.252)_{10}$  to binary with an error less than 1 %. (Refer example 1.4.23) (3)
- b) Convert the decimal number 250.5 to base 3, base 4, base 7 and base 16. (Refer example 1.4.15) (4)
- c) Simplify : 1)  $AB + \overline{ABC} + \overline{ABCD} + \overline{ABCDE}$   
                  2)  $(P + Q + R)(\overline{P} + \overline{Q} + \overline{R})P$  (Refer example 2.2.9) (7)
- Q.2** a) Explain following terms w.r.t. Digital Logic Family : 1) Fan - in 2) Noise Margin 3) Power Dissipation. (Refer section 3.2) (3)
- b) Draw and explain the working of 2-input NAND gate. (Refer section 2.1) (4)
- c) Attempt following : 1) Convert into sum-of-minterms :  $\overline{A} + B + CA$   
                  2) Convert into product-of-minterms :  $A(\overline{A} + B)\overline{C}$  (Refer example 4.1.13) (7)
- OR**
- c) Minimize the following logic function  
 $F(A, B, C, D) = \pi M(1, 2, 3, 8, 9, 10, 11, 14) \cdot d(7, 15)$   
 Use Karnaugh map. Draw the logic circuit for the simplified function using NOR gates only. (Refer example 4.7.27) (7)
- Q.3** a) State the design procedure for combinational circuits. (Refer section 4.5.2) (3)
- b) Implement the following Boolean functions with a decoder.  
 $F(w, x, y, z) = \sum (1, 3, 5, 6, 11, 14, 15)$  (Refer example 4.12.6) (6)

(5 - 1)

- c) Minimize the following function by tabular method :

$$f(w, x, y, z) = \sum m(1, 4, 8, 9, 13, 14, 15) + d(2, 3, 11, 12). \\ (\text{Refer example 4.9.5})$$

**OR**

- Q.3** a) What is encoder ? (Refer Oral Q.18 of Chapter - 4) (3)  
 b) Explain full adder in detail. (Refer section 4.14.2) (4)  
 c) Implement following logic function using  $8 \times 1$  MUX.  
 $F = \sum m(0, 1, 3, 5, 7, 11, 13, 14, 15)$  (Refer example 4.10.19) (7)

- Q.4** a) Compare combinational and sequential circuits. (Refer section 5.1.1) (3)  
 b) Describe working of look-ahead-carry adder. (Refer section 4.17) (4)  
 c) Design 2-bit comparator using gates. (Refer example 4.18.1) (7)

**OR**

- Q.4** a) Draw gated SR latch using NAND gates only. (Refer Fig. 5.4.5 (a)) (3)  
 b) Explain working of master - slave JK flip - flop with necessary logic diagram, state equation and state diagram. (Refer section 5.4.7) (4)  
 c) Design a mod-12 synchronous up counter using D-flipflop.  
 (Refer example 5.9.19) (7)

- Q.5** a) Compare three A/D conversion techniques. (Refer section 6.8.4) (3)  
 b) Explain the operation of 4-bit SISO shift register using D flip-flops.  
 (Refer section 5.8.2.1) (4)  
 c) Explain R/2R ladder technique of D/A conversion. (Refer section 6.2.2.2) (7)

**OR**

- Q.5** a) Write a short note on FPGA. (Refer section 7.17) (3)  
 b) List down the various types of ROMs and discuss any two of them.  
 (Refer section 7.5) (4)  
 c) Design a BCD to Excess-3 code converter and implement using suitable PLA.  
 (Refer example 7.13.5) (7)

