Digital Logic Design CS302

Provide by VUAnswer.com

Lecture No 1

Digital representing of quantities

Digital quantities unlike Analogue quantities are not continuous but represent quantities measured at discrete intervals.

Electronic Processing of Continuous and Digital Quantities

Electronic Processing of the continuous quantities or their Digital representation requires that the continuous signals or the discrete values be converted and represented in terms of voltages. There are basically two types of Electronic Processing Systems.

• **Analogue Electronic Systems:** **Digital Electronic Systems:**

Digital Systems and Digital Values

Digital systems are designed to work with two voltage values. A +5 volts represents a logic high state or logic 1 state and 0 volts represents a logic low state or logic 0 state.

Digital Systems are based on the Binary Number system which allows more than two or multiple values to be represented very conveniently.] Any quantity can be represented through digital values. The number of digits (0s and 1s) that represents a quantity is proportional to the range of values that are to be represented. Computers process information that is represented digitally in the form of Binary Numbers.

Data can be easily and precisely reproduced: The picture quality and the sound quality of digital videos are far more superior to those of analogue videos. The reason being that the digital video stored as digital numbers can be exactly reproduced whereas analogue video is stored as a continuous signal cannot be reproduced with exact precision.

Digital circuits occupy small space: Digital circuits are based on two logical states. Characters used in writing text are represented through yet another standard known as ASCII which allows alphabets, punctuation marks and numbers to be represented through a combination of 0s and 1s. Large and complex digital system such as a computer is built using combinations of these basic Logic Gates. These basic building blocks are available in the form of Integrated Circuit or ICs.

Combinational Logic Circuits and Functional Devices

The logic gates which form the basic building blocks of a digital system are designed to perform simple logic operations. Digital circuits that generate a new output on the basis of some previously stored information and the new input are known as Sequential circuits.

PLD Programmable Logic Devices can be used to implement Combinational and Sequential Digital Circuits.

RAM and ROM are an essential part of a digital system.

Caveman number system

A number system discovered by archaeologists in a prehistoric cave indicates that the caveman used a number system that has 5 distinct shapes Σ, Δ, >, Ω and ↑.

Lecture No 2

Binary to Decimal conversion

Most real world quantities are represented in Decimal Number System. Digital Systems on the other hand are based on the Binary Number System. Therefore, when converting from the Digital Domain to the real-world, Binary numbers have to be represented in terms of their Decimal equivalents.

There are two methods to convert binary to decimal.

1. **Sum-of-Weights Method**
2. **Sum-of-non-zero terms**
   The Sum-of-non-zero terms method is a quicker method to determine decimal equivalents of binary numbers without resorting to writing an expression.

Decimal to Binary conversion:

The **Sum-of-weights** and **repeated division by 2** methods are used to convert a Decimal number to equivalent Binary.

Sum-of-weights

43_{10}

H Weight	64	32	16	8	4	2	1
Binary Num		1	0	1	0	1	1_2

A Binary 1 is marked to represent the bit which contributed its weight in the Sum representing the decimal equivalent. The weight is subtracted from the sum decimal equivalent. The next highest weight included in the sum term is found. A binary 1 is marked to represent the bit which contributed its weight in the sum term and the weight is subtracted from the sum term. This process is repeated until the sum term becomes

Converting Decimal fractions to Binary:

Repeated Multiplication-by-2 Method:

In this method the number to be converted is repeatedly multiplied by the Base Number to which the number is being converted to, in this case 2. A new number having an Integer part and a Fraction part is generated after each multiplication. The Integer part is noted down and the fraction part is again multiplied with the Base number 2. The process is repeated until the fraction term becomes equal to zero.

Binary Arithmetic

Digital systems use the Binary number system to represent numbers. Therefore, these systems should be capable of performing standard arithmetic operations on binary numbers.

Binary Multiplication by shifting left

00011 (3) original binary number
00110 (6) binary number shifted left by 1 bit
01100 (12) binary number shifted left by 2 bits
11000 (24) binary number shifted left by 3 bits

Binary Division by Shifting right

10100 (20) original binary number
01010 (10) binary number shifted right by 1 bit
00101 (5) binary number shifted right by 2 bits

Signed and Unsigned Binary Numbers

Digital systems not only handle positive numbers but both positive and negative numbers. In a digital system which uses the Binary number system, the positive and negative signs cannot be represented as + and -. As mentioned in the Overview all forms of numbers, text, punctuation marks etc. are represented in the form of 1s and 0s. Thus the positive and negative signs are also presented in terms of binary 0 and 1.

MSB set to 1 indicates a negative number **MSB set to 0 indicates a positive number**

Thus +13 and -13 are represented as 01101and 11101 respectively. The bits 1101 represent the number 13 and the MSBs 0 and 1 represent positive and negative signs respectively.

1's & 2's complement

Informing the digital system how to treat a binary number is not very efficient. A better way is to represent negative signed numbers in their 2's complement form.

01101 The number 13
10010 1's complement of 13 is obtained by inverting all the five bits.
+ 1
(The num is -13) 10011 2's complement of 13 is obtained by adding a 1 to its 1's complement.

In a 2's complement number system all negative numbers are represented in their 2's complement form and all positive numbers are represented in their actual form.

Addition and Subtraction Operations with Signed Binary

An additional benefit of using 2's complement representation for signed numbers is that both add and subtract operations can be performed by addition.

Range of Signed and Unsigned Binary numbers

Three different types of Binary representations have been discussed. The Unsigned Binary representation can only represent positive binary numbers. The Sign-Magnitude can represent both positive and negative numbers. The 2's complement signed representation also allows positive and negative numbers to be handled.

Lecture No 3

Range of Numbers and Overflow

When arithmetic operation such as Addition, Subtraction, Multiplication and Division are performed on numbers the results generated may exceed the range of values specified by the Binary representations. The values that exceed the specified range cannot be correctly represented and are considered as Overflow values.

Determining Overflow Conditions for 2's Complement Numbers

The Overflow condition can be easily determined when two numbers represented in 2's Complement form are added together.

Floating-Point Numbers

Modern computers can handle large binary numbers such as 64-bit unsigned number, the maximum decimal number that can be represented using the 64-bit unsigned representation is 2^{64}-1 which is nearly equal to1.84 x 10^{19}.
The number 6918.3125 can be written as 6.9183125 x 10^{3}.

Arithmetic Operations on Floating Point Numbers

Arithmetic operations can be directly performed on floating point numbers by manipulating the mantissa and exponent parts of the floating point numbers.

723 represented in f.p. as exponent 2 mantissa 7.23
+ 134 represented in f.p. as exponent 2 mantissa 1.34

857 Adding together the mantissa part results in exponent 2 mantissa 8.57

Hexadecimal Numbers

The Hexadecimal number system is a base 16 number system and therefore has 16 digits and is used primarily to represent binary strings in a compact manner. Hexadecimal number system is not used by a Digital System. The Hexadecimal number system is for our convenience to long binary strings in a short and concise form. Each Hexadecimal Number digit can represent a 4-bit Binary Number.

Binary to Hexadecimal Conversion

Converting Binary to Hexadecimal is a very simple operation. The Binary string is divided into small groups of 4-bits starting from the least significant bit. Each 4-bit binary group is replaced by its Hexadecimal equivalent.

1101011010111001 0110 Binary Number
1101 0110 1011 1001 0110 Dividing into groups of 4-bits
D 6 B 9 6 Replacing each group by its Hexadecimal equivalent

Hexadecimal to Binary Conversion

Converting from Hexadecimal back to binary is also very simple. Each digit of the Hexadecimal number is replaced by an equivalent binary string of 4-bits.

FD13 Hexadecimal Number
1111 1101 0001 0011 Replacing each Hexadecimal digit by its 4-bit binary equivalent

Decimal to Hexadecimal Conversion

There are two methods

1. Indirect Method

A decimal number can be converted into its Hexadecimal equivalent indirectly by first converting the decimal number into its binary equivalent and then converting the binary to Hexadecimal.

2. Repeated Division-by-16 Method

The decimal number is continuously divided by 16 (base value of the Hexadecimal number system).

Lecture No 4

Octal Numbers

Octal Number system also provides a convenient way to represent long string of binary numbers. The Octal number is a base 8 number system with digits ranging from 0 to 7. Each Octal Number digit can represent a 3-bit Binary Number.

Binary to Octal Conversion

Converting Binary to Octal is a very simple. The Binary string is divided into small groups of 3-bits starting from the least significant bit. Each 3-bit binary group is replaced by its Octal equivalent.

11101011010111001 0110 Binary Number
111 010 110 101 110 010 110 Dividing into groups of 3-bits
7 2 6 5 6 2 6 Replacing each group by its Octal equivalent

Octal to Binary Conversion

Converting from Octal back to binary is also very simple. Each digit of the Octal number is replaced by an equivalent binary string of 3-bits

1726 Octal Number
001 111 010 110 Replacing each Octal digit by its 3-bit binary equivalent

Working with different Binary representations

There are different ways of representing numbers in binary. Four ways of representing binary numbers have been already discussed.

- Unsigned binary
- Signed-Magnitude form
- 2's Complement form
- Floating point notation

Alternate forms of Binary representations

There are many different ways to represent binary numbers, other than the 4 representation
Many of these alternate representations are used to support specific applications and requirements.

The Excess Code

Consider the decimal number range +7 to -8. These positive and negative decimal numbers can be represented by the 2's complement representation. The magnitude of positive and negative numbers cannot be easily compared as the positive and negative numbers represented in 2's complement form are not represented on a uniformly increasing scale.

The BCD Code

Binary Coded Decimal (BCD) code is used to represent decimal digits in binary. BCD code is a 4-bit binary code; the first 10 combinations represent the decimal digits 0 to 9. Remaining six 4-bit combinations 1010, 1011, 1100, 1101, 1110 and 1111 are considered to be invalid and do not exist.
A telephone keypad having the digits 0 to 9 generates BCD codes for the keys pressed.

The Gray Code

The Gray code does not have any weights assigned to its bit positions. The Gray Code is not a positional code. The Gray code is different from the unsigned binary code as successive values of Gray code differ by only one bit.

Alphanumeric Codes

All the representation studied so far allow decimal numbers to be represented in binary. Digital systems also process text information as in editing of documents. Thus each letter of the alphabet, upper case and lower case, along with the punctuation marks should have a representation. Numbers are also written in textual form such as 2nd June 2003. The ASCII Code is a universally accepted code that allows 128 characters and symbols to be represented.

ASCII Code

The ASCII Code (American Standard Code for Information Interchange) is a 7-bit code representing 128 unique codes which represent the alphabet characters A to Z in lower case and upper case, the decimal numbers 0 to 9, punctuation marks and control characters.

Extended ASCII Code

The 7-bit ASCII code only has 128 unique codes which are not enough to represent some graphical characters displayed on Computer screens. An 8-bit code Extended ASCII code gives 256 unique codes. The extended 128 unique codes represent graphic symbols which have become an unofficial standard as vendors use their own interpretation of these graphic codes.

Parity Method

Binary information which can be text or numbers is processed, stored and transmitted. Although digital systems are extremely reliable but still there is a possibility that one bit gets corrupted. That is, a 1 changes to 0 or 0 changes to 1. Many systems use a parity bit to detect errors. A single parity based error detection scheme is not very practically efficient and more elaborate and robust schemes have been designed and implemented to detect and correct multiple bit errors.

Even Parity Method

The information 10001101 is to be transmitted to a remote location. A parity bit error detection method is adopted to indicate if the information has been corrupted when it reaches the other end. In the Even Parity method the number of 1s is counted in the information and depending upon the number of 1s in the message the appended parity bit is either set to 0 or 1 to make the total number of 1s to be even (Even Parity)

Lecture No 5

LOGIC GATES

The Digital Systems should be able to process or perform operations on the numbers that are represented in the Binary Number System.
Digital Logic Gates provide the basic building blocks; these Logic Gates perform different operations on the Binary information. These Logic Gates are used in different combinations to implement large complex systems. Digital Logic Gates are represented and identified by unique symbols. These symbols are used in circuit diagrams to describe the function of a digital circuit.

AND Gate

The AND Gate performs a logical multiplication function. An AND Gate has multiple inputs and a single output.
OR Gate: The OR Gate performs a Boolean add function. An OR Gate has multiple inputs and a single output. Most commonly used OR Gates are two input OR gates.
NOT Gate: NOT Gate is also known as an Inverter. The name indicates that the NOT Gate should be performing an inversion function. The Not Gate has a single input and a single output.
NAND Gate: The NAND Gate performs a function that is equivalent to the function performed by the combination of an AND gate and a NOT gate. A NAND Gate has multiple inputs and a single output.

NAND Gate as a Universal Gate

The NAND gate is also used as a Universal Gate as the NAND Gate can be used in a combination to perform the function of a AND, OR and NOT gates.

1. NOT Gate

Implementation: A NOT gate can be implemented using a NAND gate by connecting both the inputs of the NAND gate together. By connecting the two inputs together, the input combinations where the inputs are dissimilar become redundant.

2. AND Gate Implementation

A NAND Gate performs the AND-NOT function. Removing the NOT gate at the output of the NAND gate results in an AND gate. The effect of the NOT gate at the output of the NAND gate can be cancelled by connecting a NOT gate at the output of the NAND Gate.

3. OR Gate Implementation

An OR Gate can be implemented using a combination of three NAND gates. The implementation is based on the alternate symbolic representation of the OR gate. The OR gate is represented as an AND gate with bubbles at the inputs and outputs.

NOR Gate

The NOR Gate performs a function that is equivalent to the function performed by a combination of an OR gate and a NOT gate.

Lecture No 6

NOR Gate as a Universal Gate

The NOR gate is also used as a Universal Gate as the NOR Gate can be used in a combination to perform the function of a AND, OR and NOT gates.

NOT Gate Implementation

A NOT gate can be implemented using a NOR gate by connecting both the inputs of the NOR gate together. By connecting the two inputs together, the combinations with dissimilar inputs become redundant.

OR Gate Implementation

A NOR Gate performs the OR-NOT function. Removing the NOT gate at the output of the NOR gate results in an OR gate. The effect of the NOT gate at the output of the NOR gate can be cancelled by connecting a NOT gate at the output of the NOR Gate. The two NOT gates cancel each other out.

AND Gate implementation

An AND Gate can be implemented using a combination of three NOR gates. The implementation is based on the alternate symbolic representation of the AND gate. The AND gate is represented as an OR gate with bubbles at the inputs and outputs.

NAND-NOR Universal Gates

NAND and NOR gates are known as Universal Gates as they can be used to implement any of the three fundamental gates, AND, OR and NOT.

NAND gate Implementation using NOR gates

A NAND gate implementation requires addition of an inverter (NOT) gate at the output. The NOT gate is implemented using a NOR gate.

NOR gate Implementation using NAND gates

A NOR gate implementation requires addition of an inverter (NOT) gate at the output. The NOT gate is implemented using a NAND gate.

NAND and NOR Gate Applications

The output of a NAND is 0 when all inputs to the NAND gate are 1s. This property of the NAND gate can be used to activate an operation when any of the inputs to the NAND gate are deactivated. A NOR gate on the other hand generates an output of 1 when all inputs to NOR gate are deactivated. The output is deactivated when any input is activated.

Exclusive-OR Gate

The Exclusive-OR Gate or XOR Gate performs a function that is equivalent to the combination of NOT, AND and OR gates. XOR gates are extensively used in digital applications;

Logical XOR Operation

Inputs Output 0 0 = 0, 0 1 = 1, 1 0 = 1, 1 1 = 0

Exclusive-NOR Gate

The Exclusive-NOR Gate or XNOR Gate performs a function that is equivalent to the combination of NOT, AND and OR gates. XNOR gate is extensively used in digital applications;

Logical XOR Operation

Inputs Output 0 0 = 1, 0 1 = 0, 1 0 = 0, 1 1 = 1

XOR and XNOR Gate Applications

XOR and XNOR gates are used to detect dissimilar and similar inputs. This property of XOR and XNOR gates is used to detect odd and even number of 1s in a Parity Detection Circuit.

TTL/CMOS NOT Gate Operation

Logic Gates are implemented using transistors. These transistors are connected in various combinations to form a switching circuit. The transistor itself is configured to work like a switch. On the application of a bias voltage the transistor is switched on and by removing the bias voltage the transistor is turned off.

Lecture No 7

DC Supply Voltage: TTL based devices work with a dc supply of +5 Volts. TTL offers fast switching speed,

Immunity from damage due to electrostatic discharges. Power consumption is higher than CMOS. The TTL family has six different types of devices characterized by different power dissipation and switching speeds. The series of TTL chips are:

- 74 Standard TTL
- 74S Schottky TTL
- 74AS Advanced Schottky TTL
- 74LS Low-Power Schottky TTL
- 74ALS Advanced Low-Power Schottky TTL
- 74F Fast TTL

The Standard TTL is the slowest and consumes more power and the Advanced low power Schottky has the fastest switching speed and low power requirements.

CMOS technology is the dominant technology today and used in large scale ICs and microprocessors. CMOS technology is characterized by low power dissipation with slow switching speeds. There are two categories of CMOS in terms of the dc supply voltage. The 3.3 v CMOS series is characterized by fast switching speeds and very low power dissipation as compared to the 5 v CMOS series.

- +5 V CMOS
  - o 74HC and 74HCT High-Speed
  - o 74AC and 74ACT Advanced CMOS
  - o 74AHC and 74AHCT Advanced High Speed
- 3.3 V CMOS
  - o 74LV Low voltage CMOS
  - o 74LVC Low-voltage CMOS
  - o 74ALVC Advanced Low voltage CMOS

Logic Levels and Noise Margin: The TTL and CMOS circuit operating at +5 or 3.3 Volts respectively are designed to accept voltages in a certain range as logic 1 and 0.

Noise Margin Noise margin is a measure of the circuit's immunity to noise. The high-level and low level noise margins are represented by V_{NH} and V_{NL} respectively.

The CMOS 5 volts and the 3.3 volts series cannot be mixed.

For CMOS 5 volt series the high-level noise margin is 0.9 volts. That is, the logic high output of the gate would never be below 4.4 volts.

The V_{NH} high-level and V_{NL} low-level noise margins for TTL 5 volt and CMOS 3.3 series are 0.4 volts and 0.4 volts respectively. Therefore in noisy environments, CMOS 5 volt series based digital system perform better.

Power Dissipation: Logic Gates and Logic circuits consume varying amount of power during their operation. Ideally, logic gates and logic circuit should consume minimal power. Advantages of low power consumption are circuits that can be run from batteries instead of mains power supplies. Thus portable devices that run on batteries use Integrated circuits that have low power dissipation. Secondly, low power consumption means less heat is dissipated by the logic devices; this means that logic gates can be tightly packed to reduce the circuit size without having to worry about dissipating the access heat generated by the logic devices. Microprocessors for example generate considerable heat which has to be dissipated by mounting small fans.

Generally, the Power dissipation of TTL devices remains constant throughout their operation. CMOS device on the other hand dissipate varying amount power depending upon the frequency of operation.

Power Dissipation of TTL Devices: When a TTL logic gate output is in a logic high state it draws out a current from the dc power supply. It is said to be sourcing current. The high current is designated by I_{CCH}, typical value for I_{CCH} is 1.5 mA when $V_{CC} = 5$ V. When a TTL logic gate output is in a logic low state it sinks a current designated by $I_{CCL} = 3.0$ mA when $V_{CC} = 5$ V.

Power Dissipation in TTL circuits is constant over its range of operating frequencies. For example, the power dissipation of a LS TTL gate is a constant 2.2 mW.

Power Dissipation of CMOS Devices: The transistors used in CMOS logic present a capacitive load instead of the resistive load in TTL based logic. Each time a CMOS logic gate switches between low and high, current has to be supplied to the capacitive load. The typical supply current is 5 mA for a duration of 20-30 nsec. As the frequency of operation increases, there would be more of these current spikes occurring per second, thus the average current drawn from the voltage source increases. Power Dissipation in CMOS circuits is frequency dependent. It is extremely low under static (dc) conditions and increases as the frequency increases. Total Dynamic Power dissipation of a CMOS circuit is $P_D = P_T + P_L$

Propagation Delay: Whenever a signal passes through a gate it experiences a delay. That is, a signal applied to the input of a gate does not result in an instantaneous response. The output of a gate is delayed with respect to the input. The delay in the output is known as the Propagation Delay.

Speed-Power Product (SPP): An important parameter is the Speed-Power Product which is used as a measure of performance of a logic circuit taking into account the propagation delay and the power dissipation.

Fan-Out and Loading: The fan-out of a logic gate is the maximum numbers of inputs of the same series in an IC family that can be connected to a gate's output and still maintain the output voltage levels within the specified limits. Fan-out parameter is associated with TTL technology. CMOS circuits have very high impedance therefore fan-out of CMOS circuits is very high but depends upon the frequency because of capacitance effects.

Lecture No 8

Any digital circuit no matter how complex can be described by Boolean Expressions. Boolean algebra is the mathematics of Digital Systems. Knowledge of Boolean algebra is indispensable to the study and analysis of logic gates. AND, OR, NOT, NAND and NOR gates perform simple Boolean operations and Boolean expressions represent the Boolean operations performed by the logic gates.

• AND gate F = A.B • OR gate F = A + B • NOT gate F = $\bar{A}$

• NAND gate F = $\overline{A.B}$ • NOR gate F = $\overline{A + B}$

Boolean Algebra expressions are written in terms of variables and literals using laws, rules and theorems of Boolean Algebra. Simplification of Boolean expressions is also based on the Boolean laws, rules and theorems.

Boolean Algebra Definitions

1. Variable: A variable is a symbol usually an uppercase letter used to represent a logical quantity. A variable can have a 0 or 1 value.

2. Complement: A complement is the inverse of a variable and is indicated by a bar over the variable. Complement of variable X is $\bar{X}$. If X = 0 then $\bar{X}$ = 1 and if X = 1 then $\bar{X}$ = 0.

3. Literal: A Literal is a variable or the complement of a variable.

Boolean Addition: Boolean Addition operation is performed by an OR gate. In Boolean algebra the expression defining Boolean Addition is a sum term which is the sum of literals.

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

• A sum term is 1 when any one literal is a 1 • A sum term is 0 when all literals are a 0.

Boolean Multiplication: Boolean Multiplication operation is performed by an AND gate. In Boolean algebra the expression defining Boolean Multiplication is a product term which is the product of literals.

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

• A product term is 1 when all literal terms are a 1 • A product term is 0 when any one literal is a 0.

Laws of Boolean Algebra: The basic laws of Boolean Algebra are the same as ordinary algebra and hold true for any number of variables.

1. Commutative Law for addition and multiplication A + B = B + A A.B = B.A
2. Associative Law for addition and multiplication A + (B + C) = (A + B) + C A.(B.C) = (A.B).C
3. Distributive Law A.(B + C) = A.B + A.C

Rules of Boolean Algebra: Rules of Boolean Algebra can be proved by replacing the literals with Boolean values 0 and 1.

1. A + 0 = A 2. A + 1 = 1 3. A.0 = 0 4. A.1 = A 5. A + A = A 6. A + $\bar{A}$ = 1 7. A.A = A
8. A.$\bar{A}$ = 0 9. $\bar{\bar{A}}$ = A 10. A + A.B = A = A.(1 + B)

Demorgan's Theorems: Demorgan's First Theorem states: The complement of a product of variables is equal to the sum of the complements of the variables.

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

DE Morgan's Second Theorem states: The complement of sum of variables is equal to the product of the complements of the variables.

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

Standard Form of Boolean Expressions: All Boolean expressions can be converted into and represented in one of the two standard forms

• Sum-of-Products form • Product-of-Sums form

1. Sum of Product form: When two or more product terms are summed by Boolean addition, the result is a Sum-of-Product or SOP expression.

• AB + $\bar{A}$BC • ABC + CDE + $\bar{B}\bar{C}\bar{D}$ • AB + $\bar{A}\bar{B}$C + A$\bar{C}$

2. Product of Sums form: When two or more sum terms are multiplied by Boolean multiplication, the result is a Product-of-Sum or POS expression.

• (A + B)($\bar{A}$ + B + C) • (A + $\bar{B}$ + C)(C + D + E)($\bar{B}$ + C + D) • (A + B)($\bar{A}$ + $\bar{B}$ + C)($\bar{A}$ + C)

Conversion of a general expression to SOP form: Any logical expression can be converted into SOP form by applying techniques of Boolean Algebra.

• AB+ B(CD + EF) = AB+ BCD + BEF
• (A + B)(B + C + D) = AB+ AC+ AD+ B + BC + BD = AC+ AD+ B
• (A + B) + C = (A + B)C = (A + B)C = AC + BC

Lecture No 9

BOOLEAN ALGEBRA AND LOGIC SIMPLIFICATION

Boolean Analysis of Logic Circuits, evaluating of Boolean expressions, representing the operation of Logic circuits and Boolean expressions in terms of Function tables and representing Boolean expressions in SOP and POS forms are inter-related. Boolean laws, rules and theorems are used to readily change from one form of representation to the other.

The output of the logic circuit is 1 when X = 0 and Y = 0

- X=0 NOR Y=0 Output = 1
- X=0 NOR Y=1 Output = 0
- X=1 NOR Y=0 Output = 0
- X=1 NOR Y=1 Output = 0

Standard SOP form

A standard SOP form has product terms that have all the variables in the domain of the expression. The SOP expression AC + BC is not a standard SOP as the domain of the expression has variables A, B and C.

A non-standard SOP is converted into a standard SOP by using the rule $A + \bar{A} = 1$

Standard POS form

A standard POS form has sum terms that have all the variables in the domain of the expression. The POS expression (A + B + C)(A + B + D)(A + B + C + D) is not a standard POS as the domain of the expression has variables A, B, C and D.

A non-standard POS is converted into a standard POS by using the rule $A\bar{A} = 0$

Converting to Standard SOP and POS forms

There are several reasons for converting SOP and POS forms into standard SOP and POS forms respectively.
Any logic circuit can be implemented by using either the SOP, AND-OR combination of gates or POS, OR-AND combination of gates. It is very simple to convert from standard SOP to standard POS or vice versa. This helps in selecting an implementation that requires the minimum number of gates. Secondly, the simplification of general Boolean expression by applying the laws, rules and theorems does not always result in the simplest form as the ability to apply all the rules depends on one's experience and knowledge of all the rules.

A simpler mapping method uses Karnaugh maps to simplify general expressions. Mapping of all the terms in a SOP form expression and the sum terms in a POS form can be easily done if standard forms of SOP and POS expressions are used.

Lastly, the PLDs are implemented having a general purpose structure based on ANDOR arrays. A function represented by an expression in Standard SOP form can be readily programmed.

Minterms and Maxterms: The Product terms in the Standard SOP form are known as Minterms and the Sum terms in the Standard POS form are known as Maxterms.

Binary representation of a standard Product term or Minterm: A standard product term is equal to one for only one combination of variable values. For all other variable values the standard product term is equal to zero.

Converting Standard SOP into Standard POS: The binary values of the product terms in a given standard SOP expression are not present in the equivalent standard POS expression. Also, the binary values that are not represented in the SOP expression are present in the equivalent POS expression.

Boolean Expressions and Truth Tables: All standard Boolean expressions can be easily converted into truth table format using binary values for each term in the expression. Standard SOP or POS expressions can also be determined from a truth table.

Converting SOP expression to Truth Table format: A truth table is a list of possible input variable combinations and their corresponding output values. An SOP expression having a domain of 2 variables will have a truth table having 4 combinations of inputs and corresponding output values. To convert an SOP expression in a Truth table format, a truth table having input combinations proportional to the domain of variables in the SOP expression is written. Next the SOP expression is written in a standard SOP form. In the last step all the sum terms present in the standard SOP expression are marked as 1 in the output.

Converting POS expression to Truth Table format: An POS expression having a domain of 2 variables will have a truth table having 4 combinations of inputs and corresponding output values. To convert a POS expression in a Truth table format, a truth table having input combinations proportional to the domain of variables in the POS expression is written. Next the POS expression is written in a standard POS form. In the last step all the product terms present in the standard POS expression are marked as 0 in the output.

$(A + \bar{B})(B + \bar{C})$ has a domain of three variables thus a truth table having 8 input and output combinations is required. The POS expression is converted into standard POS expression.

Lecture No 10

KARNAUGH MAP & BOOLEAN EXPRESSION SIMPLIFICATION

Simplifying Boolean Expressions using the laws, rules and theorems do not guarantee the simplest form of expression as sometimes simplification of certain terms is not so obvious or the person doesn't have the necessary experience in applying the laws and rules. The Karnaugh Map provides a systematic method for simplifying Boolean expressions. A Karnaugh Map is organized in the form of an array. Adjacent cells of the array can be grouped together to result in simplification of a given expression. Karnaugh Maps can be used to simplify expressions of 2, 3, 4 and 5 variables.

The 3-variable Karnaugh Map:

AB\C	0	1
00	0	1
01	2	3
11	6	7
10	4	5

A\BC	00	01	11	10
0	0	1	3	2
1	4	5	7	6

• A 3-variable K-Map has an array of 8 cells. The 8 cells can be arranged in 2 columns and 4 rows representing the column form of the Karnaugh Map.
• Alternately, the 8 cells can be organized in 2 rows and 4 columns representing the row form of the Karnaugh map.
• Any of the two forms of the Karnaugh Map can be used to simplify Boolean expressions. The simplified expressions using either of the two K-maps are identical.
• Considering first the column based 3-variable Karnuagh map. The binary values 00, 01, 11 and 10 in the left most column of the K-map represent the binary values of variables A and B. The binary values 0 and 1 in the top row of the K-map represent the binary values of variable C.
• The 3-variable K-Map based on the row representation is considered next. The binary values 0 and 1 in the left most column of the K-map represent the binary values of variable A. The binary values 00, 01, 11 and 10 in the top row of the K-map represent the binary values of variables B and C
• The numbers in the cells represent the Minterms or Maxterms of an expression that is to be represented using the K-map. The cell marked 0 for example, represents the minterm 0 or the maxterm 0 having binary value of variables A, B and C equal to 000. Similarly cell marked 5 represents the minterm 5 or the maxterm 5 having binary values of variables A, B and C equal to 101.

Grouping and Adjacent Cells: Karnaugh Map Array is considered to be wrapped around were all sides are adjacent to each other. Groups of 2, 4, 8, 16, 32 etc. adjacent cells are formed. Adjacent cells can be
• row wise • column wise • four corner cells • row-column groups of 4, 8, 16, 32 etc
Groups are formed on the basis of 1s (Minterms) or 0s (maxterms). A group is selected to have maximum number of cells of Minterms or Maxterms, keeping in view that the size of the group should be a power of 2. The idea is to form minimal number of largest groups that uniquely cover all the cells, thereby ensuring that all minterms or maxterms are included.

Mapping a standard SOP Expression: The first step in simplification of Boolean expressions is to map the expressions to the Karnaugh maps. For a Standard SOP expression, a 1 is placed in the cell corresponding to the product term (Minterm) present in the expression. The cells that are not filled with 1s have 0s. The Standard SOP expression having a Domain of three variables $AB\bar{C} + A\bar{B}C + \bar{A}BC$ is mapped to a 3-Variable Karnaugh Map. The product terms or the Minterms are 2, 4 and 6. The expression is mapped on a K-Map by placing a 1 at Minterm cells 2, 4 and 6 and placing 0 at remaining cells.

Mapping a non-standard SOP Expression: In many practical cases, SOP expressions are not in a standard format. To map them to K-maps they have to be either converted into Standard SOP expressions or they can be directly mapped.

Example 1: The expression $A + B\bar{C}$ is a non-standard SOP expression having a domain of 3 variables. If the expression is converted into a standard SOP expression then there will be four product terms having the variable A. Similarly, there would be two product terms having the variable combination $B\bar{C}$. Two of the product terms $AB\bar{C}$ are identical. The expression $A + B\bar{C}$ can be directly mapped to a K-map without first converting the expression to the standard form.

AB\C	0	1
00	0	0
01	1	0

11	1	1
10	1	1

The term A is mapped first. A '1' is marked in cells where the variable A is present.

A\BC	00	01	11	10
0				
1	**1**	**1**	**1**	**1**

Simplification of SOP expressions using the Karnaugh Map

SOP expressions can be very easily simplified using the K-Map method. In the first step of the simplification process, the SOP expression is mapped on the K-map. In the next step, groups of 1s are formed starting with the largest group of 1s. The group should be of size 2, 4, 8, 16 etc. having adjacent 1s. Multiple (unique) groups of 1s are formed. All the groups formed can either be separate groups or they could share common 1s each having at least a single 1 that is not common to any other group. A single 1 that is not adjacent to any other 1 is considered as a group having only a single cell.

A 3-variable K-map yields

- A product term of three variables for a group of 1 cell
- A product term of two variables for a group of 2 cell
- A product term of one variable for a group of 4 cell
- A group of 8 cells yields a value of 1 for the expression.

A 4-variable K-map yields

- A product term of four variables for a group of 1 cell
- A product term of three variables for a group of 2 cell
- A product term of two variables for a group of 4 cell
- A product term of one variable for a group of 8 cell
- A group of 16 cells yields a value of 1 for the expression.

Mapping Directly from Function Table

Practically, when a digital circuit is to be implemented to perform some operation, its function is first defined using a function table. The information in the function table is directly mapped to a K-map of appropriate variables which is then simplified. The simplified expression obtained from the K-map is directly implemented using logic Gates.

Don't care Conditions: Don't care conditions are marked as x in the output column of the function table corresponding to the don't care conditions. When the function table is mapped to the process for simplification of the SOP expression the x outputs can be considered as 0 or 1. By assigning a 0 or 1 to the cells marked with x, the final expression can be significantly simplified.

Lecture No 11

Mapping a Standard POS Expression For a Standard POS expression, a 0 is placed in the cell corresponding to the product term (maxterm) present in the expression. The cells are not filled with 0s have 1s. The Standard POS expression having a Domain of three variables

$(A + B + \bar{C}).(A + \bar{B} + C).(\bar{A} + B + \bar{C}).(\bar{A} + \bar{B} + \bar{C})$ uses a 3-Variable Karnaugh Map. The sum terms or the Maxterms are 1, 2, 5 and 7. The expression can be represented by a K-Map by placing a 0 at Maxterm locations 1, 2, 5 and 7 and placing 1 at remaining places. Any of the two K-maps can be used.

AB\C	0	1
00	1	0
01	0	1
11	1	0
10	1	0

A\BC	00	0	11	10
0	1	0	1	0
1	1	0	0	1

Karnaugh Map simplification of POS expressions: POS expressions can be easily simplified by use of the K-Map in a manner similar to the method adopted for simplifying SOP expressions. After the POS expression is mapped on the K-map, groups of 0s are marked instead of 1s based on the rules for forming groups used for simplifying SOP. In the next

step minimal sum terms are determined. Each group, including a group having a single cell, represents a sum term having variables that occur in only one form either complemented or un-complemented.

Converting between POS and SOP using the K-map: Converting between the two forms of standard expressions is very simple. If the 1s mapped on the K-map are grouped together they form the product terms of the SOP expression. Similarly, if the 0s mapped on the K-map are grouped together they form the sum terms of the POS expression

7-Segment Display: The 7-segment display digit is shown. Figure 11.10. 7-Segment Display is used to display the decimal numbers 0 to 9. A 7-segment display digit has 7 segments a, b, c, d, e, f and g that are turned on/off by a digital circuit depending upon the number that is to be displayed.

Digit	Segments
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, f, 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

Lecture No 12

COMPARATOR: A comparator circuit compares two numbers and sets one of its three outputs to 1 indicating the result of the comparison operation. A Comparator circuit has multiple inputs and three outputs.

A 2-bit Comparator circuit compares two 2-bit numbers A and B. The comparator circuit has three outputs. It sets the A>B output to 1 if A>B. It sets the A=B output to 1 if A=B and sets A<B output to 1 if A < B.

• The output A>B is set to 1 when the input combinations are 01 00, 10 00, 10 01, 11 00, 11 01 and 11 10

• The output A=B is set to 1 when the input combinations are 00 00, 01 01, 10 10 and 11 11

• The output A<B is set to 1 when the input combinations are 00 01, 00 10, 00 11, 01 10, 01 11 and 10 11

The circuit has 4-bit input, 2-bits represent A and 2-bits represent B and a 3-bit output representing A>B, A=B and A<B. To represent the function of a Comparator circuit, three function tables are required for each of the three outputs. A single function table is drawn with three outputs.

Each of the three outputs, A>B, A=B and A<B are mapped separately using three 4- variable Karnaugh maps.

Quine-McCluskey Simplification Method: Karnuagh map method becomes difficult to manage when numbers of variables exceed 4. Even with a 4-varaiable K-map, grouping of 1s or 0s depends on the ability of the user to detect optimum groups. Sometimes some redundant groups are included which adds a product term or a sum term which is not required and thus the expression is not the simplified version.

Quine-McCluskey is a program based method that is able to carry out the exhaustive search for removing shared variables. The Quine-McCluskey method is a two-step method which comprises of finding Prime Implicants and selecting a minimal set of Prime Implicants.

• Find Prime Implicants: Find by an exhaustive search all the terms that are candidates for inclusion in the simplified function. These terms are known as Prime Implicants.

• Selecting Minimal Set of Prime Implicants: Choose from amongst the Prime Implicants those that give expression with the least number of literals.

Comparator Circuit: A 2-bit Comparator circuit that compares two 2-bit numbers A and B and activates one of its three outputs A>B, A=B and A<B depending upon the magnitudes of the numbers A and B has been discussed earlier. The function outputs of the three outputs A>B, A=B and A<B can easily be represented using truth tables which can then be written in a simplified Boolean expression form after simplifying the three function expressions using 4-variable Karnaugh maps.

A comparator circuit that compares two 3-bit numbers A and B instead of the 2-bit numbers has an input of 6-bits, which represents an input combination of 64. Writing a truth table and simplifying the three expressions using the 6-variable Karnaugh maps becomes unmanageable. A program based Quine-McCluskey method can easily handle expression of 6 variables represented in the Canonical form $(8,16,17,24,.\ldots\ldots)_{A,B,C,D,E,F}\Sigma$

Odd-Prime Number Detector

A circuit that detects Odd Prime numbers between 0 and 9 has been considered earlier. The circuit is to be improved to detect Odd Prime numbers for a decimal number rangerepresented by 5-bit binary numbers or in terms of decimal numbers between the decimal numbers ranges 0 to 31. Writing out a function table to represent the 32 input combinations and their corresponding outputs, and then simplifying the function expression using a 5-varaibale

K-map can take up considerable amount of time.

Quine-McCluskey method can be used to easily simplify the 5-variable Boolean expression represented in Canonical Sum form as $(1,3,5,7,11,13,17,19,23,29,31)_{A,B,C,D,E}\Sigma$. The minterms 1, 3, 5, 7, 11, 13, 17, 19, 23, 29 and 31 represent the 5-bit input combinations (decimal numbers) which are Odd and Prime numbers.

Lecture No 13

Combinational Logic: Individual gates AND, OR and NOT, NAND and NOR Universal Gates and XOR and XNOR gates perform unique functions. These gates in their individual capacity can not perform any useful function. The Logic Gates have to be connected together in different combinations to form Logic Circuits that are able to perform some useful operation like addition , comparison etc. These combinations of gates which results in a circuit used to perform some function are known as Combinational Logic.

The function of any Digital Logic circuit is represented by Boolean expressions. In the examples discussed earlier, Boolean expressions for various functions have been determined. Two forms of representing functions through Boolean expressions are the SOP and POS expressions. These two types of Boolean expressions are implemented using a combination of gates to form Combinational Logic Circuits.

Combinational Circuit Implementation based on SOP form: A standard way to express a Boolean expression is the SOP form. The expression has several product terms which are summed together through a single OR gate. The product terms can have variables and their complemented form. A SOP expression is implemented by using a combinational circuit made up of many AND gates and a single OR gate (AND-OR gate combination). The inputs to the AND gates can be in the complemented form or the un-complemented form, requiring the use of NOT gates.

Combinational Circuit Implementation based on POS form

A standard way to express a Boolean expression is the POS form. The expression has several sum terms which are multiplied together through a single AND gate. The sum terms can have variables and their complemented form. A POS expression is implemented by using a combinational circuit made up of many OR gates and a single AND gate (OR-AND gate combination). The inputs to the OR gates can be in the complemented form or the un-complemented form, requiring the use of NOT gates.

Design and Implementation of Combinational Circuits

The design and implementation of a combinational circuit starts by defining the function of the Combinational circuit. The function of a combinational circuit is defined by a truth table or a function table. Once the function table is defined the combinational circuit can be directly implemented from the function table.

Direct implementation of a combinational circuit from the function table results in a circuit which uses maximum number of gates organized at three levels. This increases the cost, the size of the circuit and the power requirement of the Combinational circuit. The propagation delay of the circuit is of the order of three gates. Therefore, before implementing the circuit the expression is simplified using the manual method by applying rules, laws and theorems of Boolean Algebra or by the Karnaugh map method or the Quine-McCluskey method if the number of variables exceeds 4.

Implementation of an Adjacent 1s Detector Circuit

A circuit that checks an input number and determines if it has two adjacent 1s is considered to explain the entire process of design and implementation of a typical Combinational Logic Circuit. The Adjacent 1s detector circuit is implemented using the standard SOP and POS forms of Boolean expressions. The circuit is also implemented using the simplified Boolean expressions. The alternate form of implementing the circuit using only NAND or NOR gates is also discussed.

Operation of Adjacent 1s detector Circuit

The operation of a Combinational Logic Circuit can be verified by applying varying set of signals at the input of the circuit and comparing the output of the combinational circuit with the corresponding outputs in the Function Table. If the varying set of inputs and the corresponding outputs are plotted over a period of time, the timing diagram thus obtained, describes the operation of the circuit.

Active low/high Inputs and Outputs

The circuits discussed so far have their output set to when to indicate an active state. For example, the output of the BCD to 7-Segment Decoder circuit has its seven segment outputs set to 1 to indicate a segment that has been selected. Similarly, the Comparator circuit's three outputs are normally at binary 0. The appropriate output is set to 1 to indicate the relationship between the two numbers. The Odd-Prime Number detector circuit output normally is set at 0. It is activated to 1 to indicate an Odd-Prime number. The Adjacent 1s

Detector circuit also sets its output to active 1 to indicate detection of Adjacent 1s. All the four circuits have an active-high output. That is, normally the output is at logic 0. The output is set to 1 to indicate an active state.

Combinational circuits can have an active-high output or an active-low output. An active-high or active-low output doesn't effect the operation of the combinational circuit in any manner. To convert a circuit having an active-high output to active low-output requires the inversion of the circuit output by connecting a NOT gate. Symbolically, a bubble is added to the circuit output. Thus, circuits having a bubble at their outputs are considered to have an active-low output.

Lecture No 14

IMPLEMENTATION OF AN ODD-PARITY GENERATOR CIRCUIT

The first step in implementing any circuit is to represent its operation in terms of a Truth or Function table. The function table for an 8-bit data as input has 2_8 has 256 input combinations, which becomes unmanageable. Therefore, for the sake of simplicity a 4-bit data with odd parity is assumed. The receiver circuit is also based on the 4-bit data.

XOR and XNOR Gates

XOR and XNOR gates are used to implement the Odd-Parity Generator Circuit. An XNOR is also used to check for single bit errors at the Receiver end. Both, the XOR and XNOR gates perform simple comparison functions. The XOR gate detects dissimilar inputs, where as the XNOR gate looks for similar inputs. Both, the gates can be considered as functional devices as each gate performs a simple specific function. The XOR and XNOR gates are implemented using a combination of NOT, AND and OR gates. Since the function performed by the XOR and XNOR gate is commonly used in digital circuits therefore XOR and XNOR gates are available in Integrated circuit form which can be readily used instead of implementing an XOR and XNOR circuit based on NOT-ANDOR combination of gates.

Combinational Function Devices

Digital circuits are formed by the combination of Logic Gates. Most Combinationalcircuits perform standard and useful functions such as addition, comparison, decoding and encoding, multiplexing and de-multiplexing, selection and enabling of devices and many more operations. Implementation of these standard functional devices through combination of gates takes up considerable space, therefore these functional devices are implemented as MSI or Medium Scale Integrated Chips.

The simplest of these functional devices can be considered to be the NAND and NOR gates which perform the AND-NOT and OR-NOT functions. The XOR and XNOR Gates are also a combination of NOT-AND-OR gates which perform functions to detect dissimilar and similar inputs.

Half Adder and Full Adder

A single bit binary adder circuit basically adds two bits and a carry bit, generated by the addition of the least significant bits. The output of the single bit adder circuit generates a sum bit and a carry bit. A single digit binary adder circuit therefore has three inputs, one representing single bit number A, the other representing the single bit number B and the third bit represents the single bit carry. The single bit binary adder has two bit output. One bit represents the Sum between numbers A and B. The other bit represents the carry bit generated due to addition. In Digital logic terminology the adder which has been described is known as a full adder. An adder circuit that only has two bit input representing the two single bit numbers A and B and does not have the carry bit input from the least significant digit is regarded as a half-adder. The block diagrams represent a Half-Adder and a Full-Adder.

1. Half-Adder

A Half-Adder can be fully described in terms of its Function table, its Sum and Carry Out Boolean Expressions and the circuit Implementation.

Half-Adder Function Table

The Half-Adder has a 2-bit input and a 2-bit output. The function table of the Half-Adder has two input columns representing the two single bit numbers A and B. The function table also has two output columns representing the Sum bit and Carry Out bit.

2. Full-Adder

A Full-Adder can be fully described in terms of its Function table, its Sum and Carry Out Boolean Expressions and the circuit Implementation.

Full-Adder Function Table

The Full-Adder has a 3-bit input and a 2-bit output. The function table of the Full-Adder has three input columns representing the two single bit numbers A, B and the Carry In bit. The function table also has two output columns representing the Sum bit and Carry Out bit.

Forming a Full-Adder using Half-Adders

A 1-bit Full-Adder cane be implemented by combining together two Half-Adders.

Parallel Binary Adders

Single bit Full or Half Adders do not perform any useful function. To add two 4-bit numbers a 4-bit adder is required. Four single bit Full-Adders are connected together to form a 4-bit Parallel Adder capable of adding two 4-bit binary numbers.

MSI Adders

4-bit parallel Adders are available as Medium Scale Integrated Circuits. These circuits use the Look-Ahead Carry Circuitry to remove the carry ripple. The two ICS are 74LS83A and 74LS283. Both the devices are functionally identical, however they are not pin compatible.

These devices are packaged as 16-pin devices. The division of the 16 pins is

• 4 pins for the 4-bit input A • 4 pins for the 4-bit input B • 4 pins for the 4-bit output Sum
• 1 pin for Carry In • 1 pin for Carry Out • 1 pin for Circuit Power Supply • 1 pin for Circuit GND

Lecture No 15

BCD ADDER: BCD binary numbers represent Decimal digits 0 to 9. A 4-bit BCD code is used to represent the ten numbers 0 to 9. Since the 4-bit Code allows 16 possibilities, therefore the first 10 4-bit combinations are considered to be valid BCD combinations. The latter six combinations are invalid and do not occur.

BCD Code has applications in Decimal Number display Systems such as Counters and Digital Clocks. BCD Numbers can be added together using BCD Addition. BCD Addition is similar to normal Binary Addition except for the case when sum of two BCD digits exceeds 9 or a Carry is generated. When the Sum of two BCD numbers exceeds 9 or a Carry is generated a 6 is added to convert the invalid number into a valid number. The carry generated by adding a 6 to the invalid BDC digit is passed on to the next BCD digit.

Addition of two BCD digits requires two 4-bit Parallel Adder Circuits. One 4-bit Parallel Adder adds the two BCD digits. A BCD Adder uses a circuit which checks the result at the output of the first adder circuit to determine if the result has exceeded 9 or a Carry has been generated. If the circuit determines any of the two error conditions the circuit adds a 6 to the original result using the second Adder circuit. The output of the second Adder gives the correct BCD output. If the circuit finds the result of the first Adder circuit to be a valid BCD number (between 0 and 9 and no Carry has been generated), the circuit adds a zero to the valid BCD result using the second Adder. The output of the second Adder gives the same result.

Connection of Invalid BCD Detector Circuit to second Adder

Adding of 6 when error conditions are detected and adding a zero when error conditions are not detected is implemented by connecting the output of the Invalid BCD Number Detector circuit to bits B_1 and B_2 of the Adder. Bits B_0 and B_3 are permanently connected to 0.

2-digit BCD Adder

Two singe digit BCD Adders can be cascaded together to form a 2-digit BCD Adder. Four, 4-bit 74LS283 MSI chips are used. Two 74LS283s are required to directly add the two 2- digit BCD numbers and the remaining two 74LS283s are required to add a six to the result if any of the two digits add up to invalid BCD digits or generate a Carry. Two invalid BCD detector circuits are used.

Subtraction: Subtraction in Digital Systems is performed by taking the 2's complement of the number to be subtracted (subtrahend) and adding it to the minuend. The example shows the subtraction of 6 represented in 2's complement form from nine also represented in its 2's complement form. Since 9 is a positive number therefore its 2's complement representation is the same. Neglecting the carry bit, the 4-bit number represents decimal 4.

```
  9       1001
- 5       1011
4      1 0100
```

Arithmetic and Logic Unit (ALU)

Microprocessors have Arithmetic and Logic Units, a combinational circuit that can perform any of the arithmetic operations and logic operations on two input values. The operation to be performed is selected by set of inputs known as function select inputs.

There are different MSI ALUs available that have two 4-bit inputs a 4-bit output and three to five function select inputs that allows up to 32 different functions to be performed.

Three commercially available 4-bit ALUS are

• 74XX181: The 4-bit ALU has five function select inputs allowing it to perform 32 different Arithmetic and Logic operations.

• 74XX381: The 4-bit ALU only has three function select inputs allowing only 8 different arithmetic and logic functions. Table 15.6

• 74XX382: The 4-bit ALU is similar to the 74XX381, the only difference is that 74XX 381 provides group-carry look-ahead outputs and 74XX382 provides ripple carry and overflow outputs

Implementing 16-bit ALU

16-bit ALU can be implemented by cascading together four 4-bit ALUs. These 4-bit ALUs have built in Look-Ahead Carry Generator circuits that eliminate the delay caused by carry bit propagating through the Parallel Adder circuit within the 4-bit ALU circut. However, when a number of such units are cascaded together to implement large 16-bit and 32-bit ALU, the carry propagating between one unit to the next gets delayed due to the Carry rippling through multiple 4-bit units. For large 32-bit ALUs, the Carry propagates through 8, 4-bit units delaying the Carry out from the last most significant unit by a factor of 8.The 74XX181 and 74XX381 circumvent the problem by having Group-Carry Look-Ahead.

Lecture No 16

Comparators: The basic function of a Comparator is to compare two binary quantities and to determine if the two quantities are equal. If the quantities are not equal then it has to determine which of the two quantities is greater than the other. Many Integrated Circuit Comparators have three outputs to indicate A=B, A>B and A<B.

Earlier, simplified Boolean expressions for a 2-bit Comparator circuit were determined that compares two 2-bit numbers and sets one of its three outputs to indicate A=B, A>B or A<B. The Booleans expressions representing the three outputs are presented.

The 2-bit Comparator discussed earlier is considered to be a Parallel Comparator as all the bits are compared simultaneously. External Logic has to be used to Cascade together two such Comparators to form a 4-bit Comparator.

The two most significant bits of 4-bit numbers A and B are compared by the Most Significant 2-bit Comparator M and the least significant two bits are compared by the Least Significant 2-bit Comparator L.

Iterative Circuit based Comparator: An Iterative circuit is implemented using identical modules each of which has Primary Inputs and Outputs and Cascading Inputs and Outputs. The Cascading inputs of the least significant module are connected to fixed logic inputs and the Cascading outputs are connected to the Cascading inputs of the next significant module.

MSI 4-bit Comparator: MSI 74HC85 4-bit Iterative Circuit based Comparator allows multiple 74HC85s to be cascaded together to form Comparators N x 4-bit Comparators. Three 74HC85s cascaded together forms a 12-bit Comparator circuit.

Decoders: A Decoder has multiple inputs and multiple outputs. The Decoder device accepts as an input a multi-bit code and activates one or more of its outputs to indicate the presence of the multi-bit code. There are different variations of Decoder devices.

Basic Decoder: Consider an electronic door lock which unlocks the door when a 4-bit code 1011 is entered. The door is locked when another 4-bit combination 1001 is entered. The lock and unlock circuit is implemented using a combination of NOT and AND gates.

Applications of decoders: Decoders have two major uses in Computer Systems.

1. Selection of Peripheral Devices: Computers have different internal and external devices like the Hard Disk, CD Drive, Modem, Printer etc. Each of these different devices is selected by specifying different codes. A decoder similar to the Electronic Door Lock/Unlock circuit is used to uniquely select or deselect the appropriate devices.

2. Instruction Decoder: Computer programs are based on instructions which are decode by the Computer Hardware and implemented. The codes 1100010, 1100011, 1110000 and 1000101 represent Add two numbers, Subtract two numbers, Clear the result and Store the result instructions. These instruction codes are decoded by an Instruction Decoder to generate signals that control different logic circuits like the ALU and memory to perform these operations.

Binary Decoder: The simplest and most commonly used Decoders are the n-to-2_n Decoders. These Decoders have n inputs and 2_n outputs Each, n-bit input selects 1 out of 2_n output code. A 2-to-4 Decoder is represented by the function table. Table16.2.

MSI Decoder: The 74LS139 MSI chip has dual 2-to-4 Decoders.

Lecture No 17

THE 74XX138 3-TO-8 DECODER:

The 3-to-8, 74XX138 Decoder is also commonly used in logical circuits. Similar, to the 2-to-4 Decoder, the 3-to-8 Decoder has active-low outputs and three extra NOT gates connected at the three inputs to reduce the four unit load to a single unit load. The 3-to-8 Decoder has three enable inputs, one of the three enable inputs is active-high and the remaining two are active-low. All three enable inputs have to be activated for the Decoder to work.

The three enable inputs serve to implement to larger Decoders such as 4-to-16 and 5-to-32 by cascading two or four 3-to-8 Decoders respectively.

BCD to 7-Segment Decoder

BCD to 7-Segmnet Decoder is a specific type of decoder that is used to convert a 4-bit BCD Code to a 7-Segment Code. The BCD to 7-Segment Decoder unlike the Binary Decoders activates multiple but unique set of outputs for each 4-bit BCD input combination. Earlier, the seven expressions for activating each of the seven segments were defined. Each of the seven Boolean expressions can be implemented using a combination of NOTAND- OR gates.

MSI Seven-Segment Decoder

The 7-Segment Decoder circuit is available in MSI form, 74LS47. The IC has 4-bit BCD input ABCD and 7-bit active-low outputs for segments a, b, c, d, e, f and g. The Decoder also has three extra active-low inputs.

- LT: Lamp test
- RBI: Ripple Blanking Input
- BI/RBO: Blanking Input/Ripple Blanking Output

When a low is applied to the LT input and the BI/RBO is high, all of the seven segments in the display are turned on to test that no segments are burned out. The Ripple Blanking Input and The Blanking Input/Ripple Blanking Outputs are used to prevent display of leading and trailing zeros.

BCD-to-Decimal Decoder

The operation of the BCD-to-Decimal Decoder is the same as a Binary 4-to-16 decoder, the only difference being that the BCD-to-Decimal Decoder has ten output pins instead of sixteen and the input is a valid BCD number. Thus invalid BCD codes 1010, 1011, 1100, 1101, 1110 and 1111 applied at the input of the Decoder do not activate any of the ten outputs. The commercially available MSI, BCD-to-Decimal Decoder is the 74LS42, which has active-high inputs and active-low outputs.

Encoder

An Encoder functional device performs an operation which is the opposite of the Decoder function. The Encoder accepts an active level at one of its inputs and at its output generates a BCD or Binary output representing the selected input. There are various types of Encoders that are used in Combinational Logic Circuits.

Binary Encoder

The simplest of the Encoders are the 2_n-to-n Encoders. The functional table and the circuit diagram of an 8-to-3 Binary Encoder are shown in table 17.2 and figure 17.6 respectively.

Priority Encoders

Priority Encoders remove the problem highlighted earlier with simple Binary Encoders. Priority Encoders have necessary logic to activate the outputs corresponding to the highest Priority input when multiple inputs are asserted simultaneously.

Cascading Priority Encoders

The 74XX148 Priority Encoder has active-low inputs and active-low outputs. The Encoder also has an active-low enable input E1 which enables or disables the outputs. The

Group Select GS active-low output is asserted when any one of the inputs is asserted. The Enable output EO signal is used to cascade multiple Encoders to form larger Encoders. The EO output is connected to the EI input of the Encoder which handles lower priority inputs. Two 8-input are shown connected together to form a 16-input Priority Encoder.

Decimal-to-BCD Encoder

The Decimal-to-BCD Encoder has ten inputs, for the decimal digits 0 to 9 and four outputs corresponding to the 4-bit BCD output. The 74LS147 is a Decimal-to-BCD Priority Encoder which has active-low input and outputs. The Decimal-to-BCD Priority Encoder is used as a keypad encoder. A telephone keypad has digits 0 to 9. The keypad is connected to the encoder through pull-up resistors that ensure that the inputs to the encoder are logic high when none of the keypad keys is pressed. Whenever a key is pressed the appropriate input of the encoder is connected to logic low and at the output the corresponding BCD code is generated.

Multiplexer

Multiplexer is a digital switch that has several inputs and a single output. The Multiplexer also has select inputs that allow any one of the multiple inputs can be selected to be connected to the output. Multiplexers are also known as Data Selectors. The main use of the Multiplexer is to select data from multiple sources and to route it to a single Destination. In a computer, the ALU combinational circuit has two inputs to allow arithmetic operations to be performed on two quantities. The two quantities are usually stored in different set of registers. The inputs of the two multiplexers are connected to the output of each of the multiple registers. The outputs of the two multiplexers are connected to the two inputs of the ALUs. The Multiplexers are used to route the contents of any two registers to the ALU inputs.

Dual 4-Input Multiplexer

Commercial available 4-input Multiplexer is the 74XX153 IC which has two 4-input multiplexers. The two select inputs of the two 4-input multiplexers are common, however each multiplexer has a separate enable input which allows the two multiplexers to be separately controlled.

Lecture No 18

2-INPUT 4-BIT MULTIPLEXER

The MSI, 74X157 is a 2-input, 4-bit Multiplexer. This multiplexer has two sets of 4-bit inputs. It also has 4-bit outputs. The single select input line allows the first set of four inputs or the second set of 4-inputs to be connected to the output. Thus four-bits of data from two sources are routed to the output.

Expanding Multiplexers

Multiplexers have to be connected together to form larger multiplexer to fulfil specific application requirements.

8-Input Mutliplexer

A single dual, 4-input multiplexer 74X153 can be connected to form an 8-input multiplexer.

16-Input Multiplexer

Two 74XX153 Dual, 4-input multiplexer can be connected to form a 16-input multiplexer.

2-Input, 8-bit Multiplexer

Two 2-input, 4-bit multiplexers 74X157 can be connected to implement a 2-input, 8-bit multiplexer.

Applications of Multiplexers

Multiplexers are used in a wide variety of applications. Their primary use is to route data from multiple sources to a single destination. Other than its use as a Data router, a parallel to serial converter, logic function generator and used for operation sequencing.

1. Data Routnig

A two digit 7-Segment display uses two 7-Segments Display digits connected to two BCD to 7-Segment display circuits. To display the number 29 the BCD number 0010 representing the MSD is applied at the inputs of the BCD to 7-Segment display circuit connected to the MSD 7-Segment Display Digit. Similarly, the BCD input 1001 representing the numbers 9 is applied at the inputs of the LSD display circuit. The circuit uses two BCD to 7-Segment decoder circuits to decode each of the two BCD inputs to the respective 7- Segment display outputs.

Common Anode 7-Segment Display

The Common Anode 7-Segment Display has positive end of each of the seven display segments (LEDs) connected together. To display any segment the Common Anode of the display has to be connected to +5 volts and the other end of each segment has to be connected to 0 volts.

Common Cathode 7-Segment Display

The Common Cathode 7-Segment Display has negative end of each of the seven display segments (LEDs) connected together. To display any segment the Common Cathode of the display has to be connected to 0 volts and the other end of each segment has to be connected to +5 volts.

2. Parallel to Series Conversion

In a Digital System, Binary data is used and represented in parallel. Parallel data is a set of multiple bits. For example, a nibble is a parallel set of 4-bits, a byte is a parallel set of 8 bits. When two binary numbers are added, the two numbers are represented in parallel and the parallel adder works and generates a sum term which is also in parallel.

Transmission of information to remote locations through a piece of wire requires that the parallel information (data) be converted into serial form. In a serial data representation, data is represented by a sequence of single bits. An 8-bit parallel data can be transmitted through a single piece of wire 1-bit at a time. Transmitting 8-bits simultaneously (in parallel form) requires 8 separate wires for the 8-bits. Laying of 8 wires across two remote locations for data transfer is expensive and is therefore not practical. All communication systems set up across remote locations use serial transmission.

An 8-bit parallel data can be converted into serial data by using an 8-to-1 multiplexer such as 74X151 which has 8 inputs and a single output. The 8-bit data which is to be transmitted serially is applied at the 8 inputs $I_{0\text{-}7}$ of the multiplexer.

3. Logic Function Generator

Multiplexers can be used to implement a logic function directly from the function table without the need for simplification. The select inputs of the multiplexer are used as the function variables. The inputs of the multiplexer are connected to logic 1 and 0 to represent the missing and available terms.

4. Operation Sequencing

Many industrial applications have processes that run in a sequence. A paint manufacturing plant might have a four step process to manufacture paint. Each of the four steps runs in a sequence one after the other. The second step can not start before the first step has completed. Similarly, the third and fourth step of the paint manufacturing process can not proceed unless steps two and three have completed. It is not necessary that each of the manufacturing steps is of the same duration. Each manufacturing step can have different time duration and can be variable depending upon the quantity of paint manufactured or other parameters. Normally, the end of each step in the manufacturing process is indicated by a signal which is actuated by some machine which has completed its part of the manufacturing process. On receiving the signal the next step of the manufacturing process is initiated.

Lecture No 19

DEMULTIPLEXER

A Multiplexer has several inputs. It selects one of the inputs and routes the data at the selected input to the single output. Demultiplexer has an opposite function to that of the Multiplexer. It has a single input and several outputs. The Demultiplexer selects one of the several outputs and routes the data at the single input to the selected output. A demultiplexer is also known as a Data Distributor.

Applications of Demultiplexer

Demultiplexer is used to connect a single source to multiple destinations. One use of the Demultiplexer is at the output of the ALU circuit. The output of the ALU has to be stored in one of the multiple registers or storage units. The Data input of the Demultiplexer is connected to the output of the ALU. Each output of the Demultiplexer is connected to each of the multiple registers. By selecting the appropriate output data from the ALU is routed to the appropriate register for storage.

The second use of the Demultiplexer is the reconstruction of Parallel Data from the incoming serial data stream. Serial data arrives at the Data input of the Demultiplexer at fixed time intervals. A counter attached to the Select inputs of the Demultiplexer routes the incoming serial bits to successive outputs where each bit is stored. When all the bits have been stored, data can be read out in parallel.

Programmable Logic Devices

Programmable Logic Devices are used in many applications to replace the Logic gates and MSI chips. PLDs save circuit space and reduce and save the cost of components in a Digital Circuit. PLDS consists of Arrays of AND gates and OR gates that can be programmed to perform specific functions.

Programmable Arrays of AND Gates and OR Gates

The array is essentially a grid of conductors that forms rows and columns with a fuse connecting each column conductor with each row conductor. The fuses can be blown to disconnect a particular column from a particular row. The OR gate array consists of the grid and OR gates. Similarly the AND gate array consists of the grid and AND Gates.

1. Programmable Read-Only Memory (PROM)

The PROM consists of a fixed non-programmable AND array configured as a decoder and a programmable OR array. The PROM is used as a storage device which stores information at addressable locations. It has limited applications and is not used as a logic device.

2. Programmable Logic Array (PLA)

The PLA consists of a programmable AND array and a programmable OR array. It has been designed to overcome the limitations of a PROM. PLA is also known as a Field-Programmable Logic Array as it can be programmed by the user and not by the manufacturer.

3. Programmable Array Logic (PAL)

The PAL has been designed to overcome the longer delays and the complex circuitry associated with the PLA due to two programmable arrays. The PAL has programmable AND array and a fixed OR array.

4. Generic Array Logic (GAL)

The GAL has a reprogrammable AND array and a fixed OR array with programmable output logic.The main difference between GAL and PAL are the reprogrammable AND array which can be programmed again and again, unlike PAL AND array which can be programmed once. GAL uses E^2CMOS technology which is Electrically Erasable CMOS instead of Bipolar technology and fusible links. The other difference is the programmable outputs.

PAL Circuit and Programming

A simplified PAL structure is shown where the AND array has been programmed to generate three product terms which are added together by the OR array.

PALs have many inputs and multiple outputs connected through a large number of AND gates and OR gates. Drawing the circuit diagram of a PAL having multiple gates each having multiple inputs becomes difficult. PALs have Buffers at the inputs which produce the actual variable and its complement. The multiple input lines to an AND gate array are represented by a single line with a slash indicating the number of inputs. The cross indicates the fuses that are intact showing a connection between the vertical line and horizontal line of the AND array.

PAL Outputs

PALs typically have 8 or more inputs to the AND array and 8 or less outputs from the fixed OR array. Some PALs have combined inputs and outputs that can be programmed as either inputs or outputs. PAL output logic can be configured according to the application of the PAL. The modified block diagram representing a PAL showing the output of the OR Array connected to output logic which allows the outputs to be configured is shown in figure 19.11.

The three types of outputs are

- Combinational Output used for an SOP function and is available as an active-high or active-low output. Figure 19.12a
- Combinational Input/Output is used when the output is connected back to the input of the PAL or if the output pin is used as an input only. Figure 19.12b
- Programmable polarity output is used to either select the output function or its complement by programming an XOR gate at the output.

PAL Identification

PALs come in different configurations they are identified by unique number. The numbers begin with the prefix PAL followed by two digits that indicate the number of inputs followed by a letter L active-low, H active-high or P programmable polarity followed by a single or two digits that indicate the number of outputs. In addition to the standard number there may be suffixes which specify the speed, package type and temperature range.

Lecture No 20

Implementing Odd-Prime Number Function

The Odd-Prime Number generator can be implemented by programming the 4 x 3 PLA. Due to the limitations of the PLA which only has six product term (six AND gates), only the first six Odd-Prime numbers 1, 3, 5, 7, 11 and 13 can be detected. Additional two outputs are programmed to detect Odd-Prime multiples of 15 and 39 respectively. The six product terms represented by P1, P2, P3, P4, P5 and P6 are minterms 1, 3, 5, 7, 11 and 13. The first OR gate sums the six minterms (product terms) to give an output of 1 when any one of the first six
Odd-Prime numbers is applied at the inputs I1, I2, I3 and I4 of the PLA respectively. The second OR gate sums the minterms 1, 3 and 5. Thus the output of the second OR gate is a 1 when any of the three minterms is applied at the PLA inputs. Similarly, the third OR gate sums the minterms 1, 3 and 13 and the output is set to logic 1 when any one of the three inputs are detected at the input of the PLA.

GAL Operation: The GAL has a reprogrammable AND gate array and a fixed OR array. GAL can be reprogrammed as instead of fuses E^2CMOS logic is used which can be programmed to connect a column with a row. The E^2CMOS logic at each column–row intersection is known as a cell. The E^2CMOS cell in the 'on' state connects the column with the row and a cell in the 'off' state disconnects the column and row. Appropriate cells are programmed to the 'on' state to allow appropriate literals to be connected to the AND gates which generate product terms. The simplified GAL structure shows the implementation of an SOP function.

Programming of PLDs: PLDs are programmed with the help of computer which runs the programming software. The computer is connected to a programmer socket in which the PLD is inserted for programming. PLDs can also be programmed when they are installed on a circuit board. The programming of a PLD device involves entering the logic function in the form of a Boolean equation, truth table or a state diagram. Any errors during the entry process are corrected. The software compiler processes the information in the input file and translates it into a suitable format. The complier also minimizes the logic. The minimized logic is then tested by using a set of hypothetical inputs known as test vectors. The testing verifies the design of the logic circuit before committing it to the PLD. If any flaws are detected during the testing process the design must be debugged and submitted for recompilation. Once the design has been finalized a documentation file is produced along with a fuse map file which is downloaded to the programmer which programs the PLD device inserted in the programmer socket.

PLDs have In-System Programming (ISP) capability that allows the PLDs to be programmed after they have been installed on a circuit board. A standard 4-wire interface is used for programming the In-System PLD. ISP capability allows systems to be upgraded by reprogramming the PLD.

The GAL22V10

The GAL22V10 is a popular GAL device having twelve inputs and ten inputs/outputs. The device is available as low-voltage 3.3v version. It is also available as an ISP version. The device has ten OLMCs that can be programmed to different output modes. The ten OLMCs receive different number of inputs from the programmable AND gate array.
The four OLMC configurations are

- Combination Mode with active-low output
- Combinational Mode with active-high output
- Registered Mode with active-low output
- Registered Mode with active-high output

OLMC Combinational Mode

When the select inputs S_0 and S_1 are set to 0 and 1 respectively, the 4-to-1 MUX selects the OR gate output and the output is active-low because of the inversion by the tristate buffer. When the select inputs are set to 1 and 1 respectively, the MUX selects the complement of the OR gate output. The output of the OLMC is active-high due to double inversion.

Tri-State Buffers

Tri-State Buffer is a NOT gate with a control line that disconnects the output from the input. When the control line is high the buffer operates like a NOT gate and when the control line is low the output is disconnected from the output and high impedance is seen at the output. Tri-state buffers are used to disconnect the outputs of devices which are connected or share a common output line.

Introduction to ABEL:

ABEL which is an acronym for advanced Boolean expression language is a hardware description language used for implementing logic designs using PLDs. ABEL is a device-independent language and can be used to program any type of PLD.
ABEL is run on a computer connected to a PLD programmer which programs the PLD.
ABEL provides three different text-based methods for describing and entering a logic design.
The three methods are

- Boolean equations
- Truth tables
- State diagrams

The Boolean equations and the truth table method are used for combinational logic circuits. The state diagram is used specifically for sequential logic circuits. The Boolean equations and the truth table method can also be used for describing and entering sequential logic circuits.

Lecture No 21

THE GAL16V8

This device has eight inputs, two special function input pins and eight pins that can be used as inputs or output. The architecture of the GAL16V8 is similar to that of a PAL and it is designed to be programmed in one of the three available modes to emulate most of the existing PALs, thus replacing the PAL. The three modes in which PALs are programmed are

- Simple
- Complex
- Registered

The simple and complex modes are associated with the Combinational Logic whereas the Registered mode is associated with Sequential Logic.

OLMC for GAL16V8

The OLMC of the GAL16V8 is similar to the OLMC of the GAL22V10 with some enhancements. The main aspects of the GAL16V8 OLMC are

Tri-state Buffer and OLMC output pin

The tri-state buffer connecting the output of the OLMC circuit to the output pin is controlled through four different sources. The tri-state buffer control input can be connected in four different ways.

1. Connected to V_{cc}. The output is always enabled.
2. Connected to GND. The output is disabled and the output pin is configured as an input pin.
3. Connected to the external pin (11) which can be connected to V_{cc} or GND. The tri-state buffer is therefore controlled externally by applying an appropriate signal at the pin.
4. Connected to the output of one of the eight AND gates connected to the OLMC. Thus the tri-state buffer is controlled by a logical expression.

The output of the Sum of Product term

The OR gate used to implement the Sum-of-Product term has its output connected to the output pin thorough the tri-state buffer. The tri-state buffer is also connected to the output from the flip-flop. Thus either of the two inputs to the tri-state buffer can be selected. The output of the OR gate can also be programmed for output polarity by configuring the XOR gate connected at the output of the OR gate.

Simple Mode

In the Simple Mode the OLMC is configured as dedicated active combinational output or as dedicated input (limited to six). Three possible combinations of the Simple Mode are

- Combinational Output.
- Combinational Output with feedback to AND Array.
- Dedicated input.

Complex Mode

In this mode the OLMCs can be configured in two ways. In the complex Mode the tristate control is formed by a logical expression, this leaves seven product terms that can be used to form a sum-of product expression. Two possible combinations of the Complex Mode are

- Combinational Output.
- Combinational Input/Output.

Introduction to ABEL

ABEL which is an acronym for Advanced Boolean Expression Language is a hardware description language used for implementing logic designs using PLDs. ABEL is a deviceindependent language and can be used to program any type of PLD. ABEL is run on a computer connected to a PLD programmer which programs the PLD. ABEL provides three different text-based methods for describing and entering a logic design. The three methods are

- Boolean Equations
- Truth Tables
- State Diagrams

The Boolean Equations and the Truth Table method are used for Combinational Logic Circuits. The State Diagram is used specifically for Sequential Logic circuits. The Boolean Equations and the Truth Table method can also be used for describing and entering Sequential Logic Circuits.

Boolean Operations and Boolean Notations

The NOT, AND, OR and XOR operations have special symbols in ABEL as shown in

Logic Operation	ABEL Symbol
NOT	!
AND	&
OR	#
XOR	

The standard Boolean notations in terms of ABEL notations are defined in table 20.2. The operators !, &, # and $ have precedence in the order given in table.

Boolean Notation	ABEL Symbol
$\bar{A}$	!A
A.B	A&B
A +B	A#B
A $\oplus$B	A$B

Boolean Equations

One of the ABEL entry methods uses logic equations. In ABEL any letter or combination of letters and numbers can be used to identify variables. ABEL however is casesensitive, thus variable 'A' is treated separately from variable 'a'. All ABEL equations must end
with ';'.

Boolean expression F = A$\bar{B}$+ AC +$\bar{B}\bar{D}$ABEL expression F = A & !B # A & C # !B & !D;

Multiple Inputs and Outputs

In some cases, multiple input and output variables can be grouped as a set to simplify an equation. Fig 21.7. Thus D_0, D_1 and D_2 input or output variables can be defined by a single variable D using the ABEL notation D = [D0, D1, D2];

The four ABEL notations can be represented by a single notation if variables A3, A2, A1 and A0 are defined as a set A. Similarly, sets B, C and D can be defined.

Truth Table

ABEL accepts a logical design described in the form of a Truth Table. Truth Tables are sometimes more convenient in describing certain logic circuits. The ABEL Truth Table format includes a header and the truth table entries.
TRUTH_TABLE ([A, B, C, D] → [X1, X2]) A, B, C and D are the inputs and XI and X2 are the outputs.
The truth table of an XOR gate is represented by the ABEL Truth Table notation.

Test Vectors

Once the Logic circuit design has been entered its operation is verified by using 'test vectors'. A 'test vector' specifies the inputs and the corresponding outputs. The software simulates the operation of the logic circuit by applying the test vector and checking the outputs. Test vectors are essentially the same as Truth Tables.

The ABEL Input File

When an Input (source) file is created in ABEL a module is created which has three sections. The three sections are

Declarations

The declaration section generally includes the device declaration, pin declarations and set declarations. Device declaration is used to specify the PLD device that is to be programmed. The device is referred to as the target device. Decoder device 'P22V10'; A0, A1, A2, A3, PIN 1, 2, 3, 4; INPUT = [A1, A0, B1, B0];

Logic Descriptions

Logic descriptions include the three methods of describing a logic circuit. Two methods

Test Vectors

The Test Vector format has been described. The Test vector description is used to simulate the logic circuit and verify its operation. the Boolean equation and the Truth Table method already have been discussed.

The Documentation file

After an input file is processed by ABEL a documentation file is generated which provides a hardcopy of the final reduced equations, a JEDEC file and a device pin diagram.

The JEDEC file

The JEDEC file is downloaded to the PLD programmer to program the appropriate PLD device.

Lecture No 22

A Quad 1-of-4 MUX has four Multiplexers, each Multiplexer has four inputs and a single output. Each multiplexer has two select inputs to select one of the four inputs. The two select inputs are common to all the four multiplexers. The function table of the Quad 1-of-4 MUX is shown in table 22.1.

Select Inputs		Outputs			
S_1	S_0	D_{out}	C_{out}	B_{out}	A_{out}

0	0	D_0	C_0	B_0	A_0
0	1	D_1	C_1	B_1	A_1
1	0	D_2	C_2	B_2	A_2
1	1	D_3	C_3	B_3	A_3

Table 22.1 Truth table of a Quad 1-of-4 Multiplexer

```
Module quad_1of4_mux
Title          'Quad 1 of 4 multiplexer in a GAL20V8'
       mux    device          'P20V8';
       A0, A1, A2, A3          pin 1, 2, 3, 4;
       B0, B1, B2, B3          pin 5, 6, 7, 8;
       C0, C1, C2, C3          pin 9, 10, 11, 13;
       D0, D1, D2, D3          pin 14, 15, 16, 17;
       Aout, Bout, Cout, Dout  pin 21, 20, 19, 18;
S0, S1                         pin 22, 23;
Equations
       Aout = !S1 & !S0 & A0 # !S1 & S0 & A1 # S1& !S0 & A2 # S1 & S0 & A3;
       Bout = !S1 & !S0 & B0 # !S1 & S0 & B1 # S1& !S0 & B2 # S1 & S0 & B3;
       Cout = !S1 & !S0 & C0 # !S1 & S0 & C1 # S1& !S0 & C2 # S1 & S0 & C3;
       Dout = !S1 & !S0 & D0 # !S1 & S0 & D1 # S1& !S0 & D2 # S1 & S0 & D3;
Test_vectors
([S1, S0, A0, A1, A2, A3, B0, B1, B2, B3, C0, C1, C2, C3, D0, D1, D2, D3] →
[Aout, Bout, Cout, Dout])
"S  S A  A  A  A  B  B  B  B  C  C  C  C  D  D  D  D   outputs
 "0 1   0  1  2  3 0  1  2 3  0   1 2 3   0  1 2 3          A  B C  D
[0, 0,  1,  0,  0,  0, 0,  1,  0, 0,  0,  0,  1, 0,  0,  0, 0,  1]  →   [1,  0,  0,  0];
[0, 1,  1,  0,  0,  0, 0,  1,  0, 0,  0,  0,  1, 0,  0,  0, 0,  1]  →   [0,  1,  0,  0];
[1, 0,  1,  0,  0,  0, 0,  1,  0, 0,  0,  0,  1, 0,  0,  0, 0,  1]  →   [0,  0,  1,  0];
[1, 1,  1,  0,  0,  0, 0,  1,  0, 0,  0,  0,  1, 0,  0,  0, 0,  1]  →   [0,  0,  0,  1];
[0, 0,  1,  1,  1,  0, 1,  1,  0, 1,  1,  0,  1, 1,  0,  1, 1,  1]  →   [1,  1,  1,  0];
[0, 1,  1,  1,  1,  0, 1,  1,  0, 1,  1,  0,  1, 1,  0,  1, 1,  1]  →   [1,  1,  0,  1];
[1, 0,  1,  1,  1,  0, 1,  1,  0, 1,  1,  0,  1, 1,  0,  1, 1,  1]  →   [1,  0,  1,  1];
[1, 1,  1,  1,  1,  0, 1,  1,  0, 1,  1,  0,  1, 1,  0,  1, 1,  1]  →   [0,  1,  1,  1];
```

Implementation of Quad MUX: The Quad Multiplexer has 16 inputs, 4 inputs for each Multiplexer. Each multiplexer has a single output, therefore a total of 4 outputs are required. To select an appropriate multiplexer input there are two select input lines connected to all the four multiplexers. The Quad Multiplexer has a total of 22 pins through which the device is operated. The GAL16V8 device can not be used as it does not enough pins to implement the quad multiplexer. The GAL20V8 PLD is used for the implementation of the Quad 1-of-4 Multiplexer. The device has 12 inputs, 2 special function inputs and 8 input/output pins. Four input/output pins of the GAL device are configured as inputs to support the fourth multiplexer inputs D1, D2 and D3 and the select input S0.

Sequential Circuits: The combinational digital circuits have no storage element; therefore combinational circuits handle only instantaneous inputs. The outputs of the combinational circuits also can not be stored. The absence of a memory element restricts the use of digital combinational circuits to certain application areas. The use of a memory element which is capable of storing digital inputs and outputs is an important part of all practical digital circuits.

Consider an ALU which performs Arithmetic and Logical operations. An ALU can not perform its operations unless it is connected to memory elements that store the inputs applied at the inputs of the ALU and outputs from the ALU. Consider an ALU that performs addition operation on a set of numbers, 2, 3, 4 and 5. The ALU can add two numbers at a time; therefore the ALU has to add the four numbers two at a time. The four numbers have to be stored temporarily, the partial results after adding two numbers also need to be stored. To add the four numbers, the first two numbers 2 and 3 stored in two separate memory elements are added together, the result (5) has to be added to the next number 4. The result (5) is temporarily stored in one of the two memory elements used to store the numbers 2 and 3. The result (5) is added to the third number 4 to provide another partial sum result 9 which has to be stored and then added with the fourth number 5.

In a parallel-to-serial conversion of byte data using a multiplexer and the conversion from serial-to-parallel using a demultiplexer, memory elements are required that store the byte data at the input of the multiplexer for conversion into serial information and another memory element at the output of the demultiplexer for conversion back to parallel.

The counter circuit used in digital circuits count to the next value because of the memory element which stores and remembers the previous count value. A counter can not operate without a memory element.

Digital circuits that use memory elements for their operation are known as Sequential circuits. Thus Sequential circuits are implemented by combining combinational circuits with memory elements.

Latches and Flip-Flops: A latch is a temporary storage device that has two stable states. A latch output can change from one state to the other by applying appropriate inputs. A latch normally has two inputs, the binary inputcombinations at the latch input allows the latch to change its state. A latch has two outputs Q and its complement $\overline{Q}$. The latch is said to be in logic high state when Q=1 and $\overline{Q}$=0 and it is in the logic low state when Q=0 and $\overline{Q}$=1. When the latch is set to a certain state it retains its state unless the inputs are changed to set the latch to a new state. Thus a latch is a memory element which is able to retain the information stored in it.

The NAND gate based S-R (Set-Reset) Latch: An S-R Latch is implemented by connecting two NAND gates together. The output of each NAND gate is connected to the input of the other NAND gate. The unconnected inputs of the two NAND gates are the Set S and Reset R inputs. The outputs of the two NAND gates are the Q and its complement $\overline{Q}$.

The S-R latch has two inputs, therefore four different combinations of inputs can be applied to control the operation of the S-R latch. The four possible input combinations are

1. Inputs S=0 & R=0

a. Assume that the outputs Q and $\overline{Q}$ are set at logic 1 and logic 0 respectively. Since both the inputs S and R are logic low, therefore both the Q and $\overline{Q}$ outputs are set to 1. The inputs S = 0 and R = 0 are never applied as these inputs result in invalid output states as Q and $\overline{Q}$ should be complements of each other.

b. Assume that the outputs Q and $\overline{Q}$ are set at logic 0 and logic 1 respectively. Since both the inputs S and R are logic low, therefore both the Q and $\overline{Q}$ outputs are set to 1. The inputs S = 0 and R = 0 are never applied as these inputs result in invalid output states as Q and $\overline{Q}$ should be complements of each other.

The input combination S=0 and R=0 is considered to be invalid as it results in an invalid output of Q=1 and $\overline{Q}$=1.

2. Inputs S=0 & R=1

a. Consider that the outputs Q and $\overline{Q}$ have 1 and 0 logic states. The Set input S = 0 sets the output Q to 1. The Q input and the R inputs to gate 2 are both at logic 1, therefore the output $\overline{Q}$ is set to 0.

b. Consider that initially the Q and $\overline{Q}$ outputs are at logic state 0 and 1 respectively. The Set input S = 0 sets the output Q to 1. The Q input and the R inputs to gate 2 are both at logic1, therefore the output $\overline{Q}$ is set to 0.

Thus what ever the initial outputs, setting S to 0 and R to 1 sets the Q and $\overline{Q}$ outputs to 1 and 0 respectively.

3. Inputs S=1 & R=0

a. Initially, the Q and $\overline{Q}$ outputs are at 1 and 0 respectively. The Reset input R=0 sets the output $\overline{Q}$ to 1. The inputs of gate 1, $\overline{Q}$ and S are both at logic 1, therefore the output Q is set to 0. **VUAnswer.com**

b. Initially, if the Q and $\overline{Q}$ outputs are at logic 0 and 1 respectively, setting R to 0sets $\overline{Q}$ to 1. The inputs of gate 1, $\overline{Q}$ and S are both at logic 1, therefore the output Q is set to 0.

Thus, what ever the outputs, setting S to 1 and R to 0 sets the Q and $\overline{Q}$ outputs to 0 and 1 respectively.

4. Inputs S=1 & R=1

a. Initially, the Q and $\overline{Q}$ outputs are at 1 and 0 respectively. The inputs of gate 2 , Q and R are both at logic 1, therefore the output $\overline{Q}$ is set to 0. The inputs of gate 1, $\overline{Q}$ and S are 0 and 1 respectively, therefore the output is set to 1.

b. Initially, the Q and $\overline{Q}$ outputs are at 0 and 1 $\overline{Q}$ respectively. The inputs of gate 2 , Q and R are at logic 0 and 1 respectively, therefore the output is set to 1. The inputs of gate 1, and S are both at logic 1 respectively, therefore the output is set to 0.

Thus, with S and R inputs both set to logic 1, the previous output state is maintained. If initially, the Q and $\overline{Q}$ are at logic 1 and 0 respectively, setting S=1 and R=1 maintains the same outputs. Similarly, if initially Q and $\overline{Q}$ are at logic 0 and 1 respectively, setting S=1 and R=1 maintains the same outputs.

The NOR gate based S-R (Set-Reset) Latch: A NOR based S-R latch is implemented using NOR gates instead of NAND gates. Connections are identical to that of the NAND based latch. The S and R inputs have been switched.

The S-R NOR based latch has two inputs, therefore four different combinations of inputs can be applied to control the operation of the S-R latch. The four possible input combinations are

1. Inputs S=0 & R=0

a. Assume that the outputs Q and $\overline{Q}$ are set at logic 1 and logic 0 respectively. The R and $\overline{Q}$ inputs at gate 1 are both at logic 0, therefore the Q output is set to logic 1. The S and Q inputs at gate 2 are at logic 0 and 1 respectively, therefore the output $\overline{Q}$ is set to logic 0.

b. Assume that the outputs Q and $\overline{Q}$ are set at logic 0 and logic 1 respectively. The S and Q inputs at gate 2 are both at logic 0, therefore the $\overline{Q}$ output is set to logic 1. The R and $\overline{Q}$ inputs at gate 1 are at logic 0 and 1 respectively, therefore the output Q is set to logic 0.

Thus, with S and R inputs both set to logic 0, the previous output state is maintained. If initially, the Q and $\overline{Q}$ are at logic 1 and 0 respectively, setting S=0 and R=0 maintains the same outputs. Similarly, if initially Q and $\overline{Q}$ are at logic 0 and 1 respectively, setting S=0 and R=0 maintains the same outputs.

2. Inputs S=0 & R=1

a. Consider that the outputs Q and $\overline{Q}$ have 1 and 0 logic states. The Reset input R = 1 sets the output Q to 0. The Q input and the S inputs to gate 2 are both at logic 0, therefore the output $\overline{Q}$ is set to 1.

b. Consider that initially the Q and $\overline{Q}$ outputs are at logic state 0 and 1 respectively. The Reset input R = 1 sets the output Q to 0. The Q input and the S inputs to gate 2 are both at logic 0, therefore the output $\overline{Q}$ is set to 1.

Thus what ever the initial outputs, setting S to 0 and R to 1 sets the Q and $\overline{Q}$ outputs to 0 and 1 respectively.

3. Inputs S=1 & R=0

a. Initially, the Q and $\overline{Q}$ outputs are at 1 and 0 respectively. The Set input S=1 sets the output $\overline{Q}$ to 0. The inputs of gate 1, $\overline{Q}$ and R are both at logic 0, therefore the output Q is set to 1.

b. Initially, if the Q and $\overline{Q}$ outputs are at logic 0 and 1 respectively, setting S to 1 sets $\overline{Q}$ to 0. The inputs of gate 1, $\overline{Q}$ and R are both at logic 0, therefore the output Q is set to 1.

Thus, what ever the outputs, setting S to 1 and R to 0 sets the Q and $\overline{Q}$ outputs to 1 and 0 respectively.

4. Inputs S=1 & R=1

a. Initially, the Q and $\overline{Q}$ outputs are at 1 and 0 respectively. Since both the inputs S and R are logic 1, therefore both the Q and $\overline{Q}$ outputs are set to 0. The inputs S = 1 and R = 1 are never applied as these inputs result in invalid output states as Q and $\overline{Q}$ should be complements of each other.

b. Initially, the Q and $\overline{Q}$ outputs are at 0 and 1 respectively. Since both the inputs S and R are logic 1, therefore both the Q and $\overline{Q}$ outputs are set to 0. The inputs S = 1 and R = 1 are never applied as these inputs result in invalid output states as Q and $\overline{Q}$ should be complements of each other.

The input combination S=1 and R=1 is considered to be invalid as it results in an invalid output of Q=0 and $\overline{Q}$=0.

S-R Latch Timing Diagrams: The operation of the active-high and active-low input latches can be understood with the help of timing diagrams. Figure 22.6 shows the timing diagrams of the active high and active-low input latches respectively. In the timing diagram of the NAND based S-R flip-flop, the inputs S=0 and R=0 are not applied as it results in an invalid output state. Similarly, in the timing diagram of the NOR based S-R flip-flop, the inputs S=1 and R=1 are not applied as it results in an invalid output state.