

CSC 213

Digital Logic Design

By Nancy C. Woods

(PhD)

Table of Content

LECTURE ONE	6
BASIC CIRCUIT THEORY	6
1.1 THE BASICS.....	6
1.2 KIRCHHOFF'S LAWS	9
1.2.1 Kirchhoff's First Law - The Current Law, (KCL).....	10
1.2.2 Kirchhoff's Second Law - The Voltage Law, (KVL).....	10
1.4 THEVENIN'S THEOREM.....	13
1.3 NORTON'S THEOREM.....	15
LECTURE TWO	20
SEMICONDUCTORS I	20
2.1 BASIC PARTS OF A COMPUTER SYSTEM.....	20
2.2 ATOMS AND ELECTRIC CHARGES.....	22
2.3 SEMICONDUCTORS	22
2.3.1 Intrinsic Semiconductors.....	22
2.3.2 Extrinsic Semiconductors.....	23
2.3.2.1 N-type Semiconductors.....	24
2.3.2.2 P-Type Semiconductors.....	24
2.4 P-N JUNCTION	24
2.5 DIODES.....	27
LECTURE THREE	32
SEMICONDUCTORS II	32
INTRODUCTION.....	32
3.1 TRANSISTORS.....	32
3.1.1 Bipolar Junction Transistor (BJTs).....	32
The transistor as a switch	35
3.1.2 The Field Effect Transistor (FET).....	38
3.1.2.1 The Junction Field Effect Transistor (JFET).....	38
3.1.2.2 The Metal Oxide Semiconductor Field Effect Transistor (MOSFET).....	41
SUMMARY	43
LECTURE FOUR	46
DIGITAL LOGIC FAMILIES	46
INTRODUCTION.....	46
4.1 INTEGRATED CIRCUITS.....	46
4.2 LOGIC FAMILIES.....	48
4.3 CLASSIFICATION OF LOGIC FAMILIES.....	48
4.3.1 Bipolar Logic Families.....	48
4.3.2 Unipolar Logic Families.....	49
4.4 OPERATING CHARACTERISTICS AND PROPERTIES OF LOGIC FAMILIES	49
4.4.1 DC Voltage Levels	49
4.4.2 Noise Immunity.....	50
4.4.3 Noise Margin.....	51
4.4.4 Propagation Delay.....	51
4.4.5 Fan-out.....	51
4.5 TRANSISTOR-TRANSISTOR LOGIC FAMILY (TTL)	52
4.6 COMPLEMENTARY METAL OXIDE SEMICONDUCTOR LOGIC FAMILY (CMOS).....	54
LECTURE FIVE	56

DIGITAL LOGIC GATES	56
INTRODUCTION	56
OBJECTIVES	56
5.1 DIGITAL LOGIC STATES	56
5.2 DIGITAL LOGIC GATE.....	57
5.2.1 The Logic "AND" Gate.....	57
5.2.2 The Logic "OR" Gate	59
5.2.3 The Logic "NOT" Gate.....	60
5.2.4 The "BUFFER"	61
5.2.5 The Logic "NAND" Gate.....	61
5.2.6 The Logic "NOR" Gate.....	63
5.2.7 The Exclusive-OR Gate (XOR).....	64
5.2.8 The Exclusive-NOR Gate (XNOR).....	65
5.3 UNIVERSAL LOGIC GATES.....	66
5.3.1 NAND gate.....	66
5.3.2 NOR Gate.....	67
SUMMARY	68
LECTURE SIX.....	70
COMBINATIONAL LOGIC DESIGN	70
INTRODUCTION	70
OBJECTIVES	70
6.1 COMBINATIONAL LOGIC CIRCUITS (CLC).....	70
6.1.1 Truth Table.....	72
6.1.2 Boolean Expression.....	74
6.1.3 Logic Circuit diagram.....	77
6.2 EVALUATING LOGIC CIRCUIT OUTPUTS.....	80
SUMMARY	80
LECTURE SEVEN	83
STANDARD FORMS OF EXPRESSION	83
INTRODUCTION	83
OBJECTIVES	83
7.1 PRODUCT TERMS.....	83
7.2 SUM OF PRODUCT.....	84
7.3 FUNDAMENTAL SUM OF PRODUCT EXPRESSION.....	85
7.4 SUM TERMS.....	89
7.5 PRODUCT OF SUM.....	90
7.6 FUNDAMENTAL PRODUCT OF SUM EXPRESSION.....	91
SUMMARY	93
LECTURE EIGHT	94
SWITCHING FUNCTION MINIMIZATION	94
8.1 BOOLEAN ALGEBRA.....	94
8.2 KARNAUGH MAPS (K-MAPS).....	97
8.3 QUINE-McCLUSKEY METHOD	104
SUMMARY	107
LECTURE NINE	109
STANDARD COMBINATIONAL LOGIC CIRCUITS.....	109
9.1 CLASSIFICATION OF COMBINATIONAL LOGIC	109
9.2 COMBINATIONAL LOGIC CIRCUITS FOR ARITHMETIC AND LOGICAL FUNCTIONS	110
9.2.1 The Binary Adder.....	110

The Half Adder Circuit.....	111
The Full Adder Circuit.....	111
The 4-bit Binary Ripple Carry Adder	112
9.2.2 The 4-bit Binary Subtractor.....	113
9.2.3 The Digital Comparator.....	114
9.3 DATA TRANSMISSION COMBINATIONAL LOGIC CIRCUITS	116
9.3.1 The Multiplexer	116
9.3.2 The Demultiplexer.....	119
9.3.3 The Digital Encoder.....	120
9.3.4 Binary Decoder.....	122
9.4 CODE CONVERTERS.....	123
9.4.1 BCD to 7-Segment Display Decoder	123
SUMMARY	125
LECTURE TEN	128
SEQUENTIAL LOGIC CIRCUIT BASICS.....	128
INTRODUCTION.....	128
10.1 THE S-R FLIP-FLOP.....	130
10.2 THE JK FLIP-FLOP	132
10.3 THE T FLIP-FLOP.....	134
10.4 THE D-TYPE FLIP-FLOP.....	134
10.5 ASYNCHRONOUS AND SYNCHRONOUS SEQUENTIAL LOGIC CIRCUITS.....	136
SUMMARY	137
LECTURE ELEVEN.....	139
REGISTERS AND COUNTERS.....	139
11.1 CLOCKS.....	139
11.2 LEVEL AND EDGE TRIGGERING	140
11.3 CLASSIFICATION OF SEQUENTIAL LOGIC CIRCUITS.....	141
11.4 REGISTERS.....	141
11.4.1 Serial-in Parallel-out.....	142
11.4.2 Serial-in-Serial-out.....	143
11.4.3 Parallel-in-Serial-out.....	144
11.4.4 Parallel-in-Parallel-out.....	144
11.5 COUNTERS	145
11.5.1 Binary up-counter.....	147
11.5.2 Binary down-counter.....	149
SUMMARY	150
LECTURE TWELVE	152
COMPUTER CODES	152
12.1 CODES.....	152
12.1.1 Weighted Codes	153
12.1.2 Non-weighted Codes	153
12.1.3 Reflective Codes	153
12.1.4 Sequential Codes	153
12.1.5 Alphanumeric Codes.....	154
12.1.6 Error Detecting and Correcting Codes	154
12.2 BINARY CODES.....	154
12.3 BINARY-CODED DECIMAL (BCD)	154
12.4 EXCESS-3 CODE (XS-3)	157
12.5 GRAY CODE.....	159
SUMMARY	162

LECTURE THIRTEEN	163
ERROR DETECTION AND CORRECTION	163
13.1 DIGITAL ERROR	163
13.2 PARITY BIT.....	164
13.2 HAMMING CODES.....	166
13.3 CYCLIC REDUNDANCY CHECK.....	168
SUMMARY	169

PROPERTIES OF DLC UI, IBADAN

LECTURE ONE

BASIC CIRCUIT THEORY

Introduction

When studying electronic devices and electrical circuits, some basic subjects such as network theorems, electrical circuit analysis, electronic devices and circuits, and so on are usually involved. These network theorems are used to solve electrical circuits and also to calculate different parameters such as voltage, current, etc., of the circuits. Different types of theorems include Norton's theorem, Substitution theorem, Thevenin's theorem, and so on. In this lecture we will discuss the theorems that are relevant to our course.

Objectives

At the end of this lecture, you should be able to:

1. Define what an electric circuit is.
2. Identify and name electric circuit symbols.
3. State Norton's theorem, Thevenin's theorem, Kirchoff's laws
4. Analyze linear circuits and apply any of the theorems to reduce the circuit.

Pre Test

1. State Ohm's law
2. When two or more resistors are connected in series, how do you calculate the equivalent resistance?
3. What is the resistors are connected in parallel, how do calculate the total resistance?

1.1 The Basics

In any electrical circuit, some components are present in varying combinations and connections. These components are resistors, capacitors, batteries, connecting wires, etc. A closed loop through which current can flow is called an electric circuit. Every electric circuit, regardless of where it is or how large or small it is, has four basic parts: a voltage / energy source (AC or DC), a conductor, an electrical load (device), and at least one controller (switch). The voltage in a circuit can be a battery for simple circuit or our electrical sockets in our homes circuit that supply the voltage from an electric power plant. These voltage sources supply electric current

to many homes and businesses in a community. The conductor in most circuits consists of one or more wires. The conductor must form a closed loop from the source of voltage and back again as shown in Figure 1.1. Most circuits have devices such as light bulbs that convert electrical energy to other forms of energy. In the case of a light bulb, electrical energy is converted to light and thermal energy.

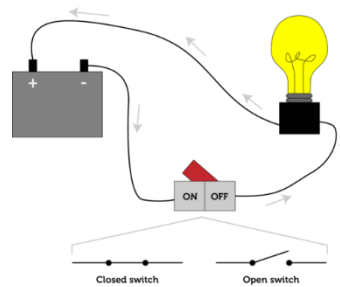


Figure 1.1: A simple electric circuit

Many circuits have switches to control the flow of current. When the switch is turned on, the circuit is closed and current can flow through it. When the switch is turned off, the circuit is open and current cannot flow through it. Different parts of an electric circuit are represented by standard circuit symbols. For simplicity we note that, 1) An ammeter measures the flow of current through a circuit; 2) a voltmeter measures the voltage. A resistor is any device that converts some of the electricity to other forms of energy. For example, a resistor might be a light bulb or doorbell or a device known as resistor itself. Figure 1.2 shows some of the standard symbols used when drawing an electric circuit. .

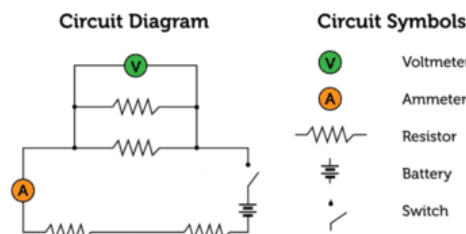


Figure 1.2: Circuit symbols

Activity

Only one of the circuit symbols must be included in every circuit. Which symbol is it?

The standard units of electrical measurement used for the expression of voltage, current and resistance are the Volt (V), Ampere (A) and Ohm (Ω) respectively. These electrical units of measurement are based on the International (metric) System, also known as the SI System with other commonly used electrical units being derived from SI base units. The following table gives a list of a few of the standard electrical units of measure used in electrical formulas and component values.

Electrical Parameter	Measuring Unit	Symbol	Description
Voltage	Volt	V or E	Unit of Electrical Potential $V = I \times R$
Current	Ampere	I or i	Unit of Electrical Current $I = V \div R$
Resistance	Ohm	R or Ω	Unit of DC Resistance $R = V \div I$

Sometimes in electrical or electronic circuits and systems it is necessary to use multiples or fractions of standard electrical measuring units when the quantities being measured are very large or very small. The table below shows the prefixes as well and conversion rate for the quantities.

Prefix	Symbol	Multiplier	Power of Ten
Terra	T	1,000,000,000,000	10^{12}
Giga	G	1,000,000,000	10^9
Mega	M	1,000,000	10^6
kilo	k	1,000	10^3
none	none	1	10^0
centi	c	1/100	10^{-2}
milli	m	1/1,000	10^{-3}
micro	μ	1/1,000,000	10^{-6}
nano	n	1/1,000,000,000	10^{-9}
pico	p	1/1,000,000,000,000	10^{-12}

From the table above we note that :

$$1 \text{ milliampere (mA)} = \frac{1}{1000} A = 10^{-3} A$$

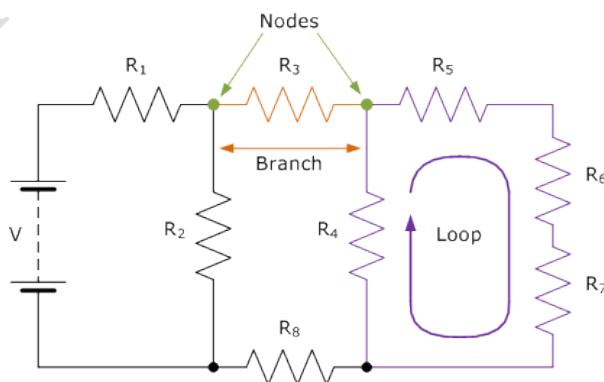
We have just studied the basics with respect to electrical circuits. Next we study the rules that can be applied to analyse a complex electrical circuit so as to make it smaller.

1.2 Kirchhoff's laws

It is possible to calculate a single equivalent resistance, when two or more resistors are connected together in either series, parallel or combinations of both. We also know that the resistors in a network obey Ohm's law. However, for more complex circuits, we cannot simply use Ohm's Law alone to find the voltages or currents circulating within the circuit. For these types of calculations we need some other rules which will allow us to obtain the circuit equations, one of such rules is **Kirchhoff's Laws**. The two rules were developed in 1845, by a German physicist, **Gustav Kirchhoff** and the pair of rules or laws deal with *the conservation of current* and *energy* within an electrical circuits.

When analysing electrical circuits using **Kirchhoff's Circuit Laws** a number of terminologies are used to describe the parts of the circuit being analysed. With reference to the figure, these terms are used frequently in circuit analysis so it is important to understand them.

- **Circuit** - a circuit is a closed loop conducting path in which an electrical current flows.
- **Path** - a line of connecting elements or sources with no elements or sources included more than once.
- **Node** - a node is a junction, connection or terminal within a circuit where two or more circuit elements are connected or joined together giving a connection point between two or more branches. A node is indicated by a dot.
- **Branch** - a branch is a single or group of components such as resistors or a source which are connected between two nodes.
- **Loop** - a loop is a simple closed path in a circuit in which no circuit element or node is encountered more than once.



1.2.1 Kirchhoff's First Law - The Current Law, (KCL)

Kirchhoff's Current Law, also known as Kirchhoff's Junction Law and Kirchhoff's First Law, defines the way that electrical current is distributed when it crosses through a junction - a point where three or more conductors meet., the law specifically states that:

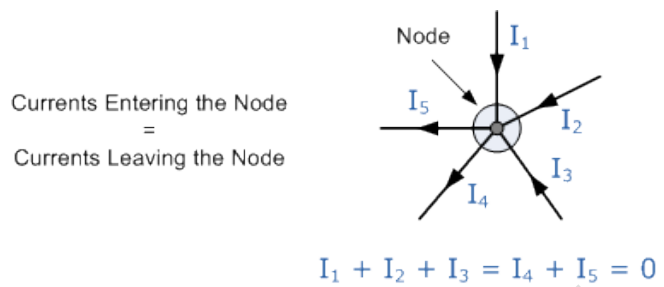


Figure 1.3: Kirchhoff's current law

The algebraic sum of currents in a network of conductors meeting at a point is zero. (Assuming that current entering the junction is taken as positive and current leaving the junction is taken as negative).

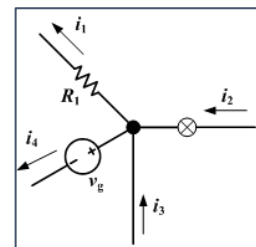
Since current is the flow of electrons through a conductor, it cannot build up at a junction, it must flow. This allows Kirchhoff's Current Law to be restated as:

At any node (junction) in an electrical circuit, the sum of currents flowing into that node is exactly equal to the sum of currents flowing out of that node

This idea by Kirchhoff is known as the **Conservation of Charge**. From figure 1.3 it can be seen that there are 3 currents entering the node, I_1, I_2, I_3 (all positive in value) and then 2 currents leaving the node, I_4 and I_5 (negative in value). Then this means we can also rewrite the equation as; $I_1 + I_2 + I_3 - I_4 - I_5 = 0$

Activity:

Can you derive the Kirchhoff's current equation at the point in this figure?



1.2.2 Kirchhoff's Second Law - The Voltage Law, (KVL)

It is a law referring to the potential field generated by voltage sources. In this potential field, regardless of what electronic components are present, the gain or loss in "energy given by the

potential field" must be zero when a charge completes a closed loop. **Kirchhoff's Voltage Law**

or KVL, states that

"in any closed loop network, the total voltage around the loop is equal to the sum of all the voltage drops within the same loop" which is also equal to zero.

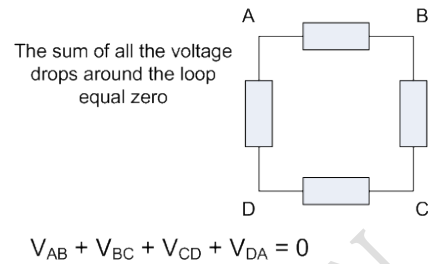


Figure 1.4: Kirchhoff's second law

In other words the algebraic sum of all voltages

within the loop must be equal to zero. This idea by Kirchhoff is

known as the **Conservation of Energy**.

There are some basic steps to follow when using **Kirchhoff's Circuit Laws** to analyse an electrical circuit. These steps are summarised below:

1. Assume all voltage sources and resistances are given. (If not label them $V_1, V_2 \dots, R_1, R_2$ etc.)
2. Label each branch with a branch current. (I_1, I_2, I_3 etc.)
3. Find Kirchhoff's current law equations for each node.
4. Find Kirchhoff's voltage law equations for each of the independent loops of the circuit.
5. Use linear simultaneous equations as required to find the unknown currents.

Example No1

Using Kirchhoff's laws, find the current flowing in the 40Ω Resistor in Figure 1.5.

Note that the 40Ω is R_L

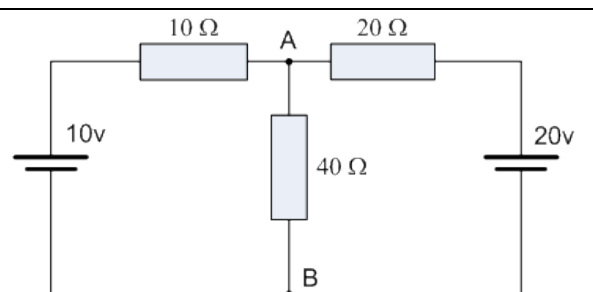


Figure 1.5: Example No 1

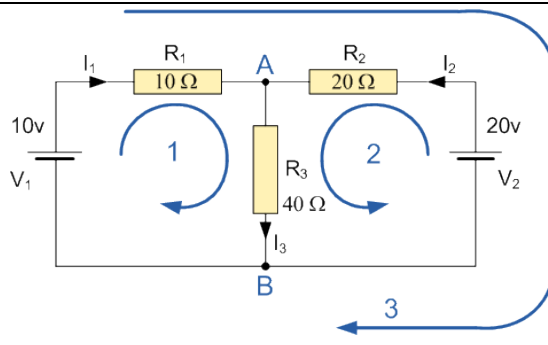
From the terminologies just defined,

we can see that this circuit has 3

branches, 2 nodes (A and B) and 3

loops. Loop 1 and loop 2 are

independent.



Using Kirchhoff's Current Law, the following equations are given as

$$\text{At node A} \quad I_1 + I_2 = I_3 \quad \text{and} \quad \text{At node B} \quad I_3 = I_1 + I_2$$

Using Kirchhoff's Voltage Law, KVL the equations are given as;

$$\text{Loop 1:} \quad 10 = I_1 R_1 + I_3 R_3 = 10I_1 + 40I_3$$

$$\text{Loop 2:} \quad 20 = I_2 R_2 + I_3 R_3 = 20I_2 + 40I_3$$

$$\text{Loop 3:} \quad 10 - 20 = I_1 R_1 - I_2 R_2 = 10I_1 - 20I_2$$

Since $I_3 = I_1 + I_2$ we can rewrite the equations as;

$$10 = 10I_1 + 40(I_1 + I_2) = 10I_1 + 40I_1 + 40I_2 = 50I_1 + 40I_2 \quad \therefore 10 = 50I_1 + 40I_2 \quad \dots \text{Eq 1}$$

$$20 = 20I_2 + 40(I_1 + I_2) = 20I_2 + 40I_1 + 40I_2 = 40I_1 + 60I_2 \quad \therefore 20 = 40I_1 + 60I_2 \quad \dots \text{Eq 2}$$

Equation 1 and 2 are two "**Simultaneous Equations**" that can be solved to give us the value of both I_1 and I_2 as follows: $I_1 = -0.1432 \text{ A}$ and $I_2 = +0.4286 \text{ A}$. You can crosscheck that!

Since $I_3 = I_1 + I_2$, then the current flowing in resistor R_3 is given as:

$$-0.1432 + 0.4286 = 0.2854 \text{ Amps}$$

and the voltage across the resistor R_3 is given as: $0.2854 \times 40 = \mathbf{11.42 \text{ volts}}$

The negative sign for I_1 means that the direction of current flow initially chosen was wrong, but never the less still valid. In fact, the 20v battery is charging the 10v battery.

Activity

Why don't you rework the above problem, but taking the direction the opposite way?

You should get the same answer

1.4 Thevenin's theorem

In the previous section we studied how to solve complex electrical circuits using Kirchhoff's circuit laws (KCL and KVL). There is another circuit analysis theorems available to calculate the currents and voltages at any point in a circuit. In this tutorial we will look at one of the more common circuit analysis theorems (next to Kirchhoff's) that has been developed, Thevenin's Theorem.

Thevenin's Theorem states that *"Any linear circuit containing several voltages and resistances can be replaced by just a Single Voltage in series with a Single Resistor"*.

In other words, it is possible to simplify any "Linear" circuit, no matter how complex, to an equivalent circuit with just a single voltage source in series with a resistance connected to a load as shown in figure 1.6. Thevenin's Theorem is especially useful in analyzing power or battery systems and other interconnected circuits where it will have an effect on the adjoining part of the circuit.

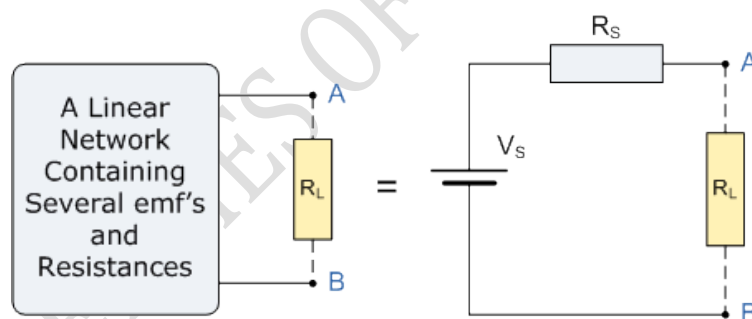


Figure 1.6: Thevenin's equivalent circuit.

As far as the load resistor R_L is concerned, any "one-port" network consisting of resistive circuit elements and energy sources can be replaced by one single equivalent resistance R_S and equivalent voltage V_S , where R_S is the source resistance value looking back into the circuit and V_S is the open circuit voltage at the terminals.

The basic procedure for solving a circuit using Thevenin's Theorem is as follows:

1. Remove the load resistor R_L or component concerned.

- Find R_S by shorting all voltage sources or by open circuiting all the current sources.
- Find the open-circuit V_S which appears across the two terminals from where the resistance was removed.
- Draw the equivalent circuit with the Load resistor included.

Let us take an example.

Example 2:

Find and draw the Thevenin's equivalent circuit for the circuit in Figure 1.7. Then calculate the current flowing in the load resistor R_L .

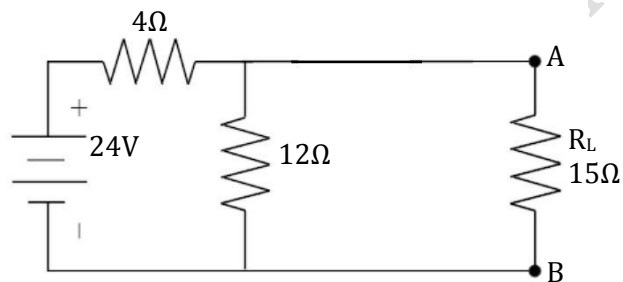
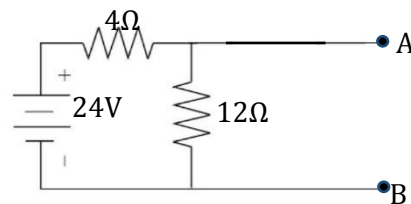


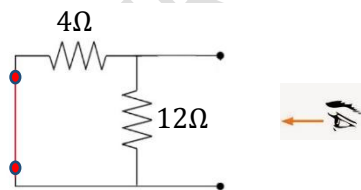
Figure 1.7: Example 2 circuit

Solution

Step 1: Remove load resistor as shown:



Step 2: Find R_S by shorting all voltage sources and looking into the circuit from the point of view of the load



$$R_S = 4\Omega \parallel 12\Omega$$

$$R_S = \frac{12 \times 4}{12 + 4} = \frac{48}{16} = 3\Omega$$

Step 3: Find V_S .

Therefore, using $V = IR$, $I = \frac{V}{R}$

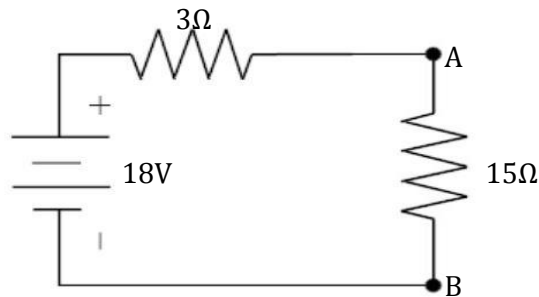
From the circuit shown in step 1, if apply Ohm's law, we can find the current flowing in loop in the circuit. Mind you the point AB is open and the resistors are in series in the loop.

$$I = \frac{24}{12 + 4} = \frac{24}{16} = 1.5A$$

\therefore voltage across terminals $AB = V_S = 1.5 \times 12 = 18V$

Note that since terminal AB is in parallel with the 12Ω resistor, they have the same voltage.

Step 4: The Thevenin's equivalent circuit can then be drawn as shown below:



To calculate the current flowing in the Load resistor, note that from the equivalent circuit, the two resistors are in series now so the same current will run through them.

$$\therefore I = \frac{V}{R} = \frac{18}{3 + 15} = \frac{18}{18} = 1A$$

Activity No 2

Find the Thevenin's equivalent circuit for the circuit in figure 1.5. Also calculate the current flowing in the Load resistor 40Ω. You should get 0.2854 Amps

1.3 Norton's Theorem

This is another useful theorem used to analyse electric circuits like Kirchhoff's laws and Thevenin's theorem. This theorem helps to reduce or simplify any linear electrical complex circuit into a simple circuit that consists of a single current source and a parallel equivalent resistance connected across the load. The Norton's equivalent circuit can be represented as shown in the figure 1.8.

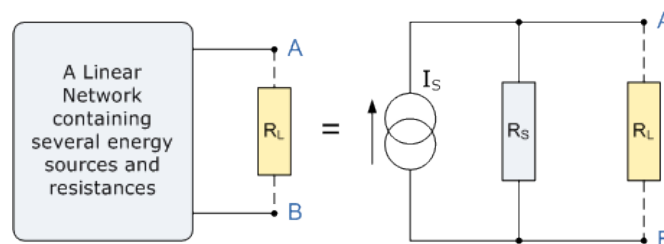


Figure 1.8: Norton's Theorem

Norton's Theorem may be stated as:

Any Linear Electric Network or complex circuit with Current sources, Voltage sources and resistances can be replaced by an equivalent circuit containing of a single independent constant Current Source I_S and a Parallel Resistance R_S .

The value of this "constant current" is one which would flow if the two output terminals were shorted together while the source resistance would be measured looking back into the terminals as in Thevenin's theorem just studied.

The simple Steps to analyse an electric circuit using Norton's Theorem are listed below:

1. Remove the load resistance across the given terminal and place a short-circuit across them.
2. Calculate / measure the Short Circuit Current (I_S)
3. Open Current Sources, Short Voltage Sources and Open Load Resistor.
4. Calculate /measure the Open Circuit Resistance (R_S) looking into the circuit from the terminal
5. Redraw the circuit with measured short circuit Current (I_S) in Step (2) as current Source and measured open circuit resistance (R_S) in step (4) as a parallel resistance and connect the load resistor which we had removed in Step (1).

Example

Find the Norton's equivalent circuit for the circuit in figure 1.9

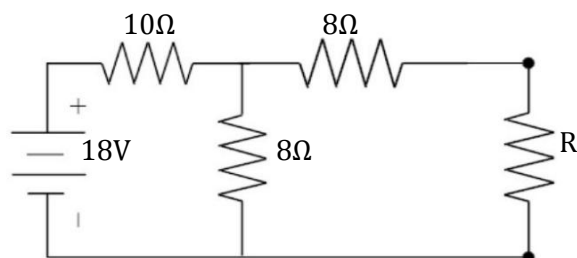
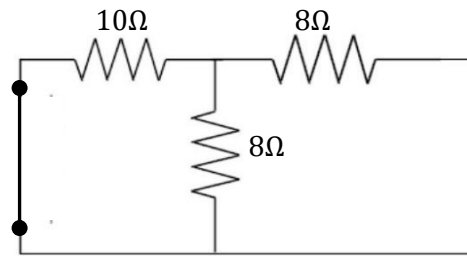


Figure 1.9: Norton example circuit

To find the Norton's equivalent circuit of the circuit in figure 1.9, we first remove the load resistor R_L as shown in the figure and short out the voltage source. We can calculate the Norton equivalent in two steps



1. Find the Equivalent Resistance (R_S)

Replace voltage sources by short circuits and calculate /measure the open circuit resistance between the two terminals where the load was removed

$$R_S = 8\Omega + (10\Omega \parallel 8\Omega)$$

$$R_S = 8 + \left(\frac{10 \times 8}{10 + 8}\right) = 8 + 4.44$$

$$R_S = 12.44\Omega$$

2. Calculate I_S

Replace the voltage source and calculate the short circuit current I_S .

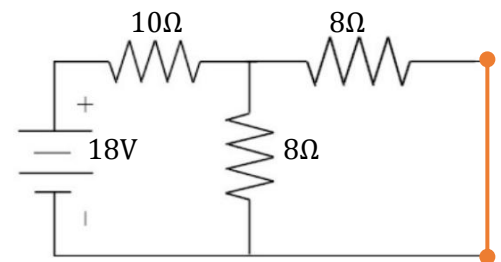
If we short the load terminals to determine the Norton current, I_S . The two 8Ω are then in parallel and this parallel combination are then in series with 10Ω .

So the total resistance of the circuit with respect to the source is:-

$$R_T = 10\Omega + (8\Omega \parallel 8\Omega)$$

$$R_T = 10 + \left(\frac{8 \times 8}{8 + 8}\right) = 10 + 4$$

$$R_T = 14\Omega$$



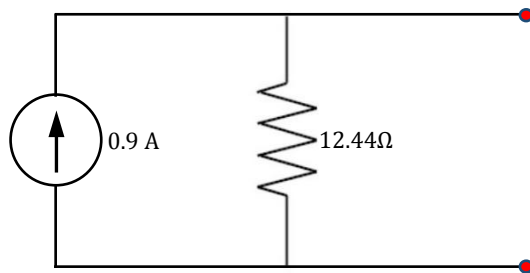
Since we have R_T as 14Ω and we know that the voltage supplied by the cell is $18V$, we can calculate the current flowing through the 10Ω , which is the total current

$$I = \frac{V}{R} = \frac{18}{10} = 1.8A$$

Since the two resistors have the same value, and following Kirchhoff's current law, it means that

$$I_S = 1.8/2 = 0.9A$$

The Norton equivalent circuit is therefore as shown below:



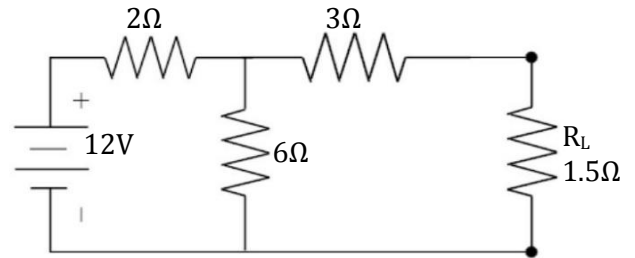
Summary

1. An electric circuit is a closed loop through which current can flow.
2. All electric circuits must have a voltage source, such as a battery, and a conductor, which is usually wire. They may have one or more electric devices as well.
3. An electric circuit can be represented by a circuit diagram, which uses standard symbols to represent the parts of the circuit.
4. Kirchhoff's current law states that the algebraic sum of currents in a network of conductors meeting at a point is zero; while Kirchhoff's voltage law states the algebraic sum of all voltages within a loop must be equal to zero.
5. Thevenin's theorem simplifies any "Linear" circuit, no matter how complex, to an equivalent circuit with just a single voltage source in series with a resistance connected to the Load.
6. Norton's Theorem allows a network consisting of linear resistors and sources to be represented by an equivalent circuit with a single current source in parallel with a single source resistance connected to the Load in Parallel.

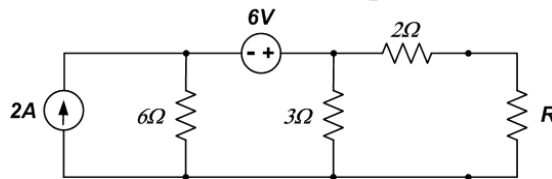
Post-Test

1. What is an electric circuit?
2. Which two parts must all electric circuits contain?
3. Sketch a simple circuit that includes a battery, switch, and light bulb. Then make a circuit diagram to represent your circuit, using standard circuit symbols.

4. Find the Norton's equivalent circuit for this figure



5. a. Study the figure below. Note that R is the *load*.



- i. Find and draw the Thévenin's equivalent circuit for the figure.
- ii. Find the current flowing through the equivalent circuit if $R = 8\Omega$.

References

Duncan, Tom. 1997, Electronics for Today and Tomorrow. Spain: John Murray.

Theraja, B.L. and Theraja, A.K., 2005. A textbook of electrical technology, S. Chand.

<https://www.ck12.org/>

<https://www.hunker.com/>

LECTURE TWO

SEMICONDUCTORS I

Introduction

In this lecture, you will be introduced to the concept of semiconductors, their characteristics and uses. You will also get to know the basic building block of most computer systems and digital devices. However for a start, we will look at the basic parts of computer system as well as their main functions.

Objectives

At the end of this lecture, you should be able to:

1. Define what a semiconductor is.
2. Differentiate between the two types of semiconductors.
3. Identify and state the uses of some diodes
4. Differentiate between the two main types of transistors.

Pre-Test

1. What is an atom?
2. What is referred to as the valence electron of any atom?
3. What is the maximum number of electrons that can be accommodated in the K-shell, L-shell, M-shell and N-shell?
4. Which materials are referred to as conductors?
5. Which materials are referred to as insulators?

2.1 Basic Parts of a Computer System

The parts of any computer system or digital device can be broadly classified into the following:

1. **Input / Output Devices:** These devices enable the user of a system to give commands to the system as well to receive signals from the system. The results that are stored in the memory can be transformed into a form that can be understood by users of a computer system by means of an output device. Some common output devices are monitor, printer,

speaker etc; while some input devices are the keyboard/ keypad, mouse, the touch screen, etc.

2. **Mass storage Device:** These devices allow computer to permanently retain large amounts of data. Common mass storage device include disk drives, flash disk, memory cards, CDs and tape drives.
3. **Central Processing Unit (CPU):** The CPU or processor is the brain of the computer. It is the component that actually executes the instructions issued to any digital device. The CPU itself has three components, namely, arithmetic logic unit (ALU), control unit (CU), and memory unit (MU).

Arithmetic Logic Unit:-

Arithmetic logic unit (ALU) performs two types of operations- arithmetic and logical. Fundamental arithmetic operations include addition, subtraction, multiplication, and division. Logical operations include comparisons like equal to, less than, greater than etc.

Control Unit:-

Control unit (CU) coordinates and controls the operations of a computer system. It controls the activities between memory and ALU and between CPU and input/output devices.

Memory Unit

A memory unit (MU) is also called primary memory or main memory or RAM (random access memory). It holds data for processing, instructions for processing data (program), and information (processed data). The contents of main memory are lost when the computer is turned off.

Most of the parts mentioned above are made up of smaller components. These components include: ***diodes, transistors, capacitors, resistors, inductors***. Some of them are referred to as Semi-conductors but first we will look at the structure of an atom and it's properties.

2.2 Atoms and electric charges

An atom consists of a nucleus that contains protons, which have a positive (+) electric charge, surrounded by an equal number of electrons with a negative (-) charge. The nucleus also contains uncharged neutrons. The charge on a proton is the same as that on an electron. An atom can lose one or more electrons and when that happens, it becomes positively charged because it then has more protons than electrons. It then means that if it gains one or more electrons, it becomes negatively charged.

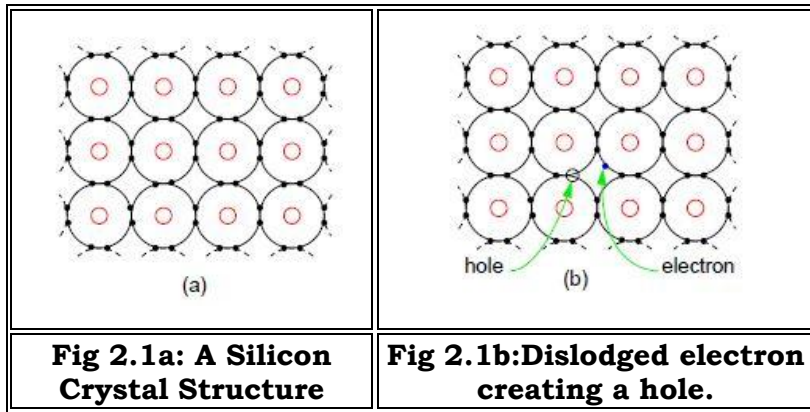
An electric current is produced

2.3 Semiconductors

You can easily tell if a material is a conductor by touch, though that may be dangerous! Pure semiconductors are relatively good insulators as compared with metals, though not nearly as good as a true insulator like glass. Semiconductors are materials whose conductivity lie between that of a conductor and an insulator. There are two main types of semiconductors: Intrinsic and Extrinsic semiconductors.

2.3.1 Intrinsic Semiconductors

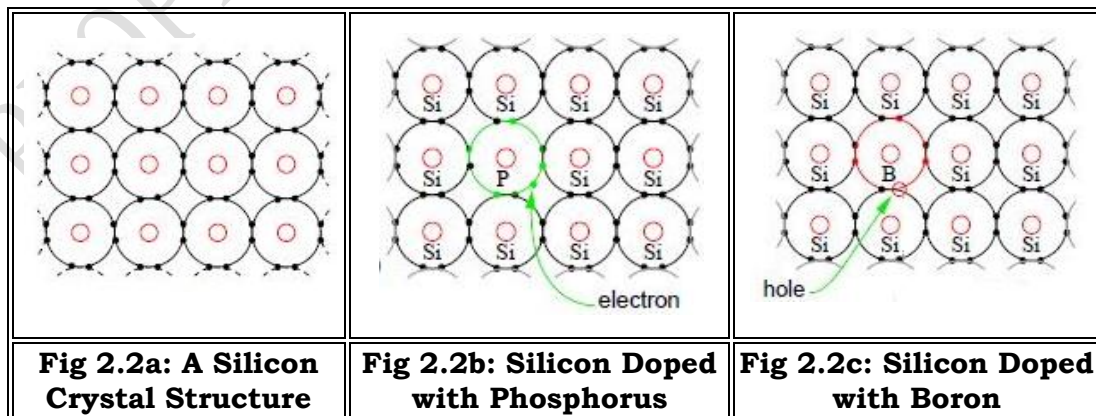
Group IV elements (Silicon, Germanium, Carbon) are good semiconductors, they have 4 valance electrons. Figure 2.1(a) shows four electrons in the valence shell of a semiconductor atom forming covalent bonds to four other atoms. This is a flattened, easier to draw, version. All electrons of an atom are tied up in four covalent bonds, pairs of shared electrons. Electrons are not free to move about the crystal lattice. Thus, intrinsic, pure, semiconductors are relatively good insulators as compared to metals.



Thermal energy may occasionally free an electron from the crystal lattice as in Figure 2.1(b). This electron is free for conduction about the crystal lattice. When the electron was freed, it left an empty spot with a positive charge in the crystal lattice known as a hole. The free electron and hole both contribute to conduction about the crystal lattice. That is, the electron is free until it falls into a hole. This is called *recombination*. Increasing temperature will increase the number of electrons and holes, decreasing the resistance. The number of electrons and holes in an intrinsic semiconductor are equal.

2.3.2 Extrinsic Semiconductors

Pure intrinsic semiconductors, by themselves, are not particularly useful. To make them useful, their conductivity must be increased. As we saw in the previous section, that silicon forms rigid crystals because of its four valence electron structure. Figure 2.2a: shows part of a silicon crystal structure.



Pure silicon is not a conductor because there are no free electrons. To make silicon conducting, producers dope pure silicon by adding very small amounts of impurities which are other elements with *fewer* outer valence electrons like boron, Aluminum or *more* outer valence electrons like phosphorus and arsenic. The addition of a desired impurity to a semiconductor is known as **doping**. Doping increases the conductivity of a semiconductor so that it is more comparable to a metal than an insulator. Depending on the type of doping material used, extrinsic semiconductors can be sub-divided into two classes: N-type and P-Type.

2.3.2.1 N-type Semiconductors

Phosphorus is a pentavalent element, that is, it has five valence electrons. Pentavalent doping atom is known as **donor** atom because it donates or contributes one electron. When silicon is doped with phosphorus (fig 2.2b) there is an extra electron and **a net negative charge**. This type of material is called n-type silicon or N-type semiconductor. The extra electron in the crystal cell is not strongly attached and can be released by normal thermal energy to carry current; the conductivity depends on the amount of phosphorus added to the silicon. For N-type semiconductors, the electron is the majority carrier.

2.3.2.2 P-Type Semiconductors

Boron on the other hand is trivalent, that is, it has only three valence electrons. Trivalent doping atom is known as **acceptor** atom because it accepts one electron. When silicon is doped with boron there is a "hole" where another electron would have been, if the boron atom were silicon; see fig 2.2c. This gives the crystal cell **a net positive charge**. This type of material is referred to as p-type silicon or P-Type semiconductor. It has the ability to pick up an electron easily from a neighboring cell and therefore these materials are sometimes referred to as acceptors. For P-type semiconductors, the hole is the majority carrier.

2.4 P-N Junction

Both p-type and n-type silicon will conduct electricity just like any conductor. However, if a single semiconductor crystal is manufactured with P-type material at one section and N-type

material at the other section as in Figure 2.3, the material has some unique properties. The P-type material has positive majority charge carriers, holes, which are free to move about the crystal lattice. The N-type material has mobile negative majority carriers, electrons. The plane dividing the two zones is called the junction, P-N junction to be precise. Near the junction, the N-type material electrons diffuse across the junction, combining with holes in P-type material. The region of the P-type material near the junction takes on a net negative charge because of the electrons attracted. Since electrons departed the N-type region, it takes on a localized positive charge. The thin layer of the crystal lattice between these charges has been depleted of majority carriers, thus, is known as the **depletion region**.

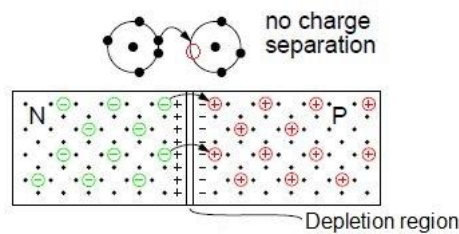


Figure 2.3: A P-N Junction show the depletion region

That region becomes non-conductive and this separation of charges at the P-N junction constitutes a potential barrier that must be overcome by an external voltage source to make the junction conduct. The formation of the junction and potential barrier, happens during the manufacturing process and the magnitude of the potential barrier is a function of the materials used in manufacturing. Silicon P-N junctions have a higher potential barrier than germanium junctions.

Forward Biased junction

What is depicted in figure 2.3 is **an unbiased** on junction, because there is no external voltage source connected to the device. If a battery is arranged such that the negative terminal is connected to the N-type section and thus supplies electrons to the N-type material as shown in figure 2.4a, electrons diffuse toward the junction. The positive terminal removes electrons from the P-type semiconductor, creating holes. If the battery voltage is great enough to overcome the junction potential (0.6V in Si), the N-type electrons and P-holes combine overpowering each

other. This frees up space within the lattice for more carriers to flow toward the junction. Thus, currents of N-type and P-type majority carriers flow toward the junction, thereby reducing the width of the depletion region. The recombination at the junction allows a battery current to flow through the P-N junction, from the p-side to the n-side of the lattice. Such a junction is said to be *forward biased*.

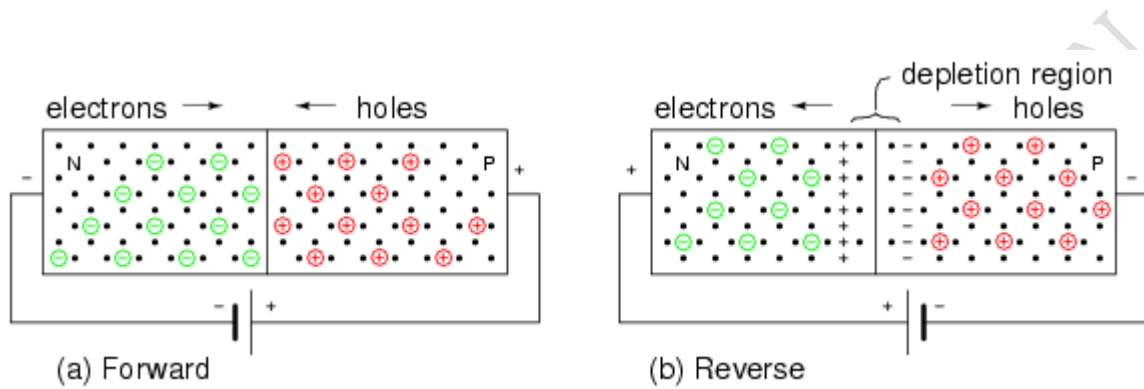


Figure 2.4: Biased P-N junction

Reverse Biased junction

If the battery polarity is reversed as in Figure 2.4(b) above, majority carriers are attracted away from the junction toward the battery terminals. The positive battery terminal attracts N-type majority carriers, electrons, away from the junction. The negative terminal attracts P-type majority carriers, holes, away from the junction. This increases the thickness of the non-conducting depletion region. There is no recombination of majority carriers; thus, no conduction, current does not flow through the junction. This arrangement of battery polarity is called *reverse bias*.

The P-N Junction is fundamental to the operation of diodes, transistors and other solid state devices.

2.5 Diodes

A diode is a *unidirectional* semiconductor device. The symbol of a basic diode or pn diode is depicted in Figure 2.5(b) corresponding to the doped semiconductor bar at figure 2.5(a). Electrons only flows in one direction, from the n-side to the p-side of the bar, this means that current flows in only one direction, from the P-side to the N-side. From our physics in secondary school, we know that the flow of electricity is in the opposite direction of the flow of electrons within the material.

The opposite end of the diode are called cathode and anode. The cathode of the diode corresponds to N-type semiconductor while the anode corresponds to the P-type semiconductor. To remember this relationship, **Not**-pointing (bar) on the symbol corresponds to **N**-type semiconductor. **P**ointing (arrow) corresponds to **P**-type.

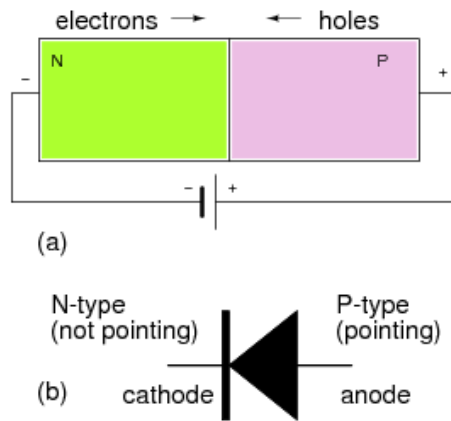


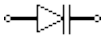
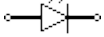



Figure 2.5(a) Forward biased PN junction, (b) Corresponding diode schematic symbol.

If a diode is forward biased, current will increase slightly as voltage is increased from 0 V. In the case of a silicon diode a measurable current flows when the voltage approaches 0.6 V. Increasing the voltage well beyond 0.7 V may result in high enough current to destroy the diode. The forward voltage is a characteristic of the semiconductor: 0.6 to 0.7 V for silicon, 0.2 V for germanium. If the diode is reverse biased, only the leakage current of the intrinsic semiconductor flows.

There are different types of diodes for different functions. The p-n diode can be used as a rectifier so some books may call them rectifier diode. We also have the common light emitting diode used in our torch these days. The table below shows the symbol of some diodes and their description. You may want to read more about their uses and mode of operation.

Symbol	Name	Description
	Zener Diode	Allows current flow in one direction, but also can flow in the reverse direction when above breakdown voltage
	Schottky Diode	Schottky diode is a diode with low voltage drop
	Varactor / Varicap Diode	Variable capacitance diode
	Light Emitting Diode (LED)	LED emits light when current flows through

	Photodiode	Photodiode allows current flow when exposed to light. These are used in solar panels.
---	------------	---

Summary

- Intrinsic semiconductor materials are poor conductors.
- N-type semiconductor is doped with a pentavalent impurity to create free electrons. Such a material is conductive. The electron is the majority carrier.
- P-type semiconductor, doped with a trivalent impurity, has an abundance of free holes. These are positive charge carriers. The P-type material is conductive. The hole is the majority carrier.
- Most semiconductors are based on elements from group IVA of the periodic table, silicon being the most prevalent. Germanium is all but obsolete. Carbon (diamond) is being developed.
- PN junctions are fabricated from a piece of semiconductor with both a P-type and N-type region in proximity at a junction.
- A forward biased PN junction conducts a current once the barrier voltage is overcome. The external applied potential forces majority carriers toward the junction where recombination takes place, allowing current flow.
- A reverse biased PN junction conducts almost no current. The applied reverse bias attracts majority carriers away from the junction. This increases the thickness of the non-conducting depletion region.
- Most modern diodes are based on semiconductor **p-n junctions**
- In a p-n diode, conventional current can flow from the p-doped side (the anode) to the n-doped side (the cathode), but not in the opposite direction.

Post-Test

1. What is referred to as recombination?
2. What is the effect of doping?
3. What is a semiconductor?
4. Differentiate between intrinsic and extrinsic semiconductors?
5. How can silicon be made a p-type semiconductor?
6. How can silicon be made an n-type semiconductor?
7. The majority carriers in an n-type semiconductor differs from that of a p-type semiconductor. What are these majority carriers and how are they created?
8. What is a p-n junction?
9. Describe the three ways of biasing a p-n junction as well as the effects.
10. List any three types of diodes and their description.

PROPERTIES OF DLC UI, IBADAN

References

- Crowe, J., and Barrie Hayes-Gill. 1998. Introduction to Digital Electronics.
- Duncan, Tom. 1997. Electronics for Today and Tomorrow. Spain: John Murray.
- Heuring, Vincent P., and Harry F. Jordan. 2004. Computer Systems Design and Architecture 2nd Edition. Prentice Hall.
- Hwang, Enoch O. 2005. Digital Logic and Microprocessor Design With VHDL. Brooks/ Cole.
- Kuphaldt Tony R., 2009, Lessons In Electric Circuits, Volume III – Fifth Edition Semiconductors
- Rocco, Ronald J., and Neal S. Wilmer. 2001. Digital Systems, Principles and Applications, Eighth Edition. New Jersey: Prentice Hall.
- Theraja, B.L. and Theraja, A.K., 2005. A textbook of electrical technology, S. Chand.

PROPERTIES OF DLC UI, IBADAN

LECTURE THREE

SEMICONDUCTORS II

Introduction

Another type of semiconductor device that is very vital to the operation of any computer system, is the transistor. Microprocessors contain millions of transistors. We will discuss the usefulness of transistors in subsequent lectures but first for this lecture, we will look at Transistors.

Objectives

At the end of this lecture you should be able to

1. Explain what a transistor is
2. Differentiate between the two main types of transistors
3. Discuss how a transistor can be used as a switch
4. Identify the symbols of the different types of transistor

Pre test

1. What is your understanding of the operation of a switch?
2. What are the different types of switches you have come across?
3. What happens at a PN junction?
4. What are the majority charge carriers in semiconductos?

3.1 Transistors

We saw in the previous section that the diode is a two terminal device which allows current to flow in only one direction as soon as the potential difference of the depletion region has been overcome by an external voltage source. The transistor on the other hand is a three terminal semiconductor device. There are two main types of transistor used in the computer systems. These are Bipolar Junction Transistors (BJTs) and Field Effect Transistors (FETs).

3.1.1 Bipolar Junction Transistor (BJTs)

The bipolar junction transistor (BJT) is so called, because its operation involves conduction of electricity by the two charge carriers; that is electrons and holes in the same crystal. However some authors say that the name bipolar was given because the basic construction of the device

consists of two PN-junctions as you can see in figure 3.1 (a). The first bipolar transistor was invented at Bell Labs by William Shockley, Walter Brattain, and John Bardeen so late in 1947 that it was not published until 1948. Brattain fabricated a germanium point contact transistor, bearing some resemblance to a point contact diode. Within a month, Shockley had a more practical junction transistor. They were awarded the Nobel Prize in Physics in 1956 for the transistor.

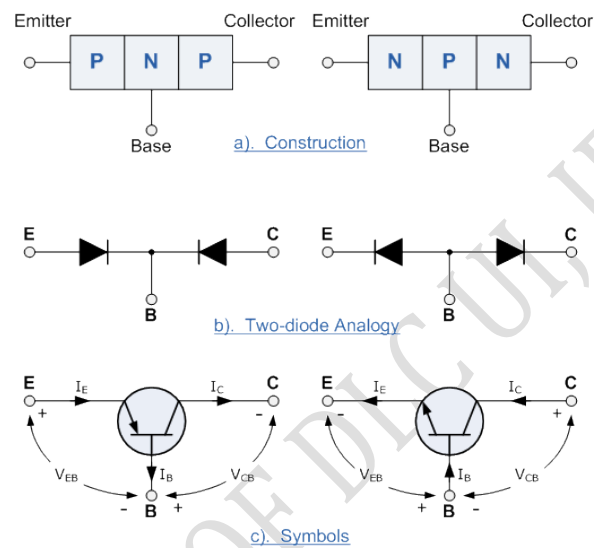


Figure 3.1: NPN vs PNP transistor

There are two basic types of bipolar transistor *construction*, NPN and PNP as shown in figure 3.1. The two different types basically describe the physical arrangement of the P-type and N-type semiconductor materials on the crystal lattice. The principle of operation of the NPN and PNP transistors, is exactly the same the only difference being in the biasing and the polarity of the power supply for each type. The arrow in the circuit symbol always shows the direction of conventional current flow between the base terminal and its emitter terminal, with the direction of the arrow pointing from the positive P-type region to the negative N-type region, exactly the same as for the standard diode symbol. For the rest of this lecture, we will describe the NPN type.

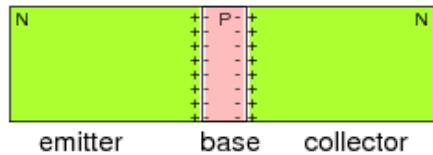


Figure 3.2: The structure of an NPN Bipolar Junction Transistor

As you may have seen from figure 3.1, a BJT is a semiconductor device that has three terminals and three 'layers'. The BJT shown in Figure 3.2 is an NPN semiconductor device, with an emitter and a collector at the ends, and a base in between the two. The key to the fabrication of a bipolar junction transistor is to make the middle layer, the base, as thin as possible without shorting the outside layers, the emitter and collector. Author's usually cannot over emphasize the importance of the thin base region, because this helps with the conduction of current through the device. Basically, the NPN transistor device must meet the following rules:

1. The collector must be more positive than the emitter
2. The base-emitter and base-collector circuits, behave like diodes. Normally, the base-emitter diode is conducting and the base-collector diode is reverse biased as shown in the diode analogy.
3. Any given transistor has maximum values of Collector current (I_C), Base current (I_B), and Voltage between the collector and the emitter (V_{CE}) that cannot be exceeded without costing the exceder the price of a new transistor.
4. I_C is roughly proportional to I_B and can be written as

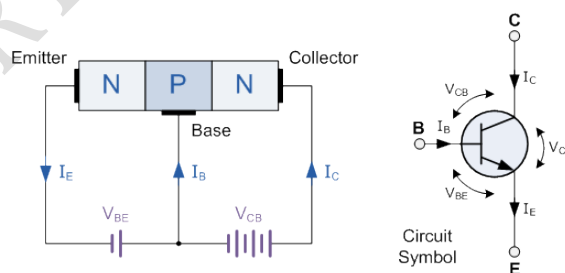


Figure 3.3: NPN transistor and its conventional current flow.

The transistor is a "CURRENT" operated device because a large current (I_C) flows freely through the device between the collector and the emitter terminals. However, this only happens when a small biasing current (I_B) is flowing into the base terminal of the transistor thus allowing the base to act as a sort of current control input. The ratio of these two currents (I_C/I_B) is called the

DC Current Gain of the device and is given the symbol of Beta, (β). Beta has no units as it is a ratio. Also, the current gain from the emitter to the collector terminal, I_C/I_E , is called Alpha, (α), and is a function of the transistor itself. As the emitter current I_E is the product of a very small base current to a very large collector current the value of this parameter α is very close to one, and for a typical low-power signal transistor this value ranges from about 0.950 to 0.999.

$$\text{DC Current Gain} = \frac{\text{Output Current}}{\text{Input Current}} = \frac{I_C}{I_B}$$

$$\beta = \frac{I_C}{I_B} \quad \alpha = \frac{I_C}{I_E}$$

$$I_E = I_C + I_B$$

$$V_{CE} = V_{CB} + V_{BE}$$

One of the most important properties of the **Bipolar Junction Transistor** is that a small base current can control a much larger collector current. One other point to remember about NPN Transistors is that the collector voltage, (V_C) must be greater than the emitter voltage, (V_E) to allow current to flow through the device between the collector-emitter junction.

Example.

An NPN Transistor has a β value of 100. Calculate the base current I_B required to switch a resistive load of 4mA.

$$I_B = \frac{I_C}{\beta} = \frac{4 \times 10^{-3}}{100} = 40\mu A \quad \text{Therefore, } \beta = 100, I_C = 4\text{mA} \quad \text{and} \quad I_B = 40\mu A.$$

For the example above, assuming the Transistor has a β value of 200, find I_B and I_E

The transistor as a switch

Let us see how a BJT can be used as a switch! We just learnt that the base current of a BJT when present allows current to flow through the device. That is to say that because a transistor's collector current is proportionally limited by its base current, it can be used as a sort of current-controlled switch. A relatively small flow of electrons sent through the base of the transistor has the ability to exert control over a much larger flow of electrons through the collector. Suppose

we had a lamp that we wanted to turn on and off with a switch, we could draw such a circuit as extremely simple as in Figure 3.4 (a). For the sake of illustration, let's insert a transistor in place of the switch to show how it can control the flow of electrons through the lamp in Figure 3.4 (b).

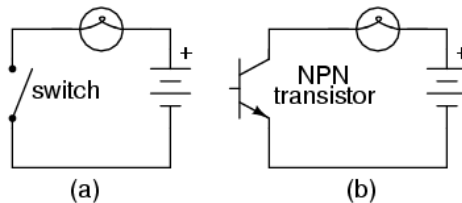


Figure 3.4: (a) mechanical switch, (b) NPN transistor switch.

Note that we need to supply a base current for the transistor to function, so to make the circuit complete, we have to temporarily (we will see why) connect a switch between the base of the transistor and the collector as shown in figure 3.5.

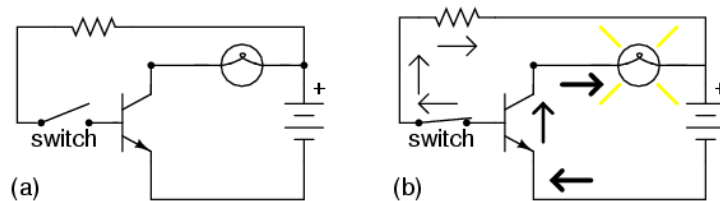


Figure 3.5: Transistor: (a) cutoff, lamp off; (b) saturated, lamp on.

If the switch is open as in Figure 3.5(a), the base wire of the transistor will be left “floating” (not connected to anything) and there will be no current through it. In this state, the transistor is said to be *cutoff*. If the switch is closed as in Figure 3.5 (b), however, electrons will be able to flow from the emitter through to the base of the transistor, through the switch and up to the left side of the lamp, back to the positive side of the battery. This base current will enable a much larger flow of electrons from the emitter through to the collector, thus lighting up the lamp. In this state of maximum circuit current, the transistor is said to be *saturated*.

Of course, it may seem pointless to use a transistor in this capacity to control the lamp. After all, we're still using a switch in the circuit, aren't we? Why not just go back to our original circuit and use the switch directly to control the lamp current? Two points can be made here, actually. First is the fact that when used in this manner, the switch contacts need only handle what little

base current is necessary to turn the transistor on; the transistor itself handles most of the lamp's current. This may be an important advantage if the switch has a low current rating: a small switch may be used to control a relatively high-current load. More important, the current-controlling behaviour of the transistor enables us to use something completely different to turn the lamp on or off. Consider Figure 3.6(a), where a pair of solar cells, that serve as light sensor, provides 1 V to overcome the $0.7 V_{BE}$ of the transistor to cause base current to flow, which in turn controls the lamp. Or Figure 3.6(b) where a thermocouple (many connected in series) is used to provide the necessary base current to turn the transistor on. Or Figure 3.6(c) where even a microphone with enough voltage and current (from an amplifier) output could turn the transistor on, provided its output is rectified from AC to DC so that the emitter-base PN junction within the transistor will always be forward-biased.

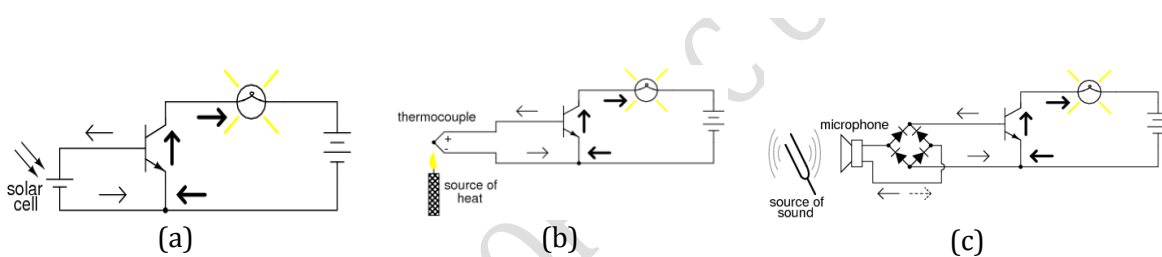


Figure 3.5 (a) Solar cell serves as light sensor. (b) a series of thermocouple (c) a microphone uses sound

The point should be quite apparent by now: *any* sufficient source of DC current may be used to turn the transistor on, and that source of current only need be a fraction of the current needed to energize the lamp. Please note that the actual power for lighting up the lamp comes from the battery to the right of the schematic. It is not as though the small signal current from the solar cell, thermocouple, or microphone is being magically transformed into a greater amount of power. Rather, those small power sources are simply *controlling* the battery's power to light up the lamp by completing the circuit. Now you have seen how a BJT can be used as a switch, we will look at the second type of transistor, the Field Effect Transistor.

3.1.2 The Field Effect Transistor (FET)

In BJT section, we saw that the output Collector current is determined by the amount of current flowing into the Base terminal of the device and thereby making the Bipolar Transistor a CURRENT operated device. The Field Effect Transistor, on the other hand, uses the voltage that is applied to their input terminal to control the output current, since their operation relies on the electric field (hence the name field effect) generated by the input voltage. This then makes the FET a **VOLTAGE** operated device. There are two basic natures of FET, the N-channel and the P-channel which simply signify the type of semiconductor material the main body (called a channel) is made of. The FET is often referred to as a *unipolar* semiconductor device that has very similar properties to those of the *Bipolar Transistor*. "Unipolar" in the sense that the device depends only on the conduction of Electrons (N-channel) or Holes (P-channel). From their physical 'arrangement', only one PN-junction is present. The FET has one main advantage over the BJT, in that their input impedance is very high, making them very sensitive to input signals, but this high sensitivity also means that they can be easily damaged by static electricity. There are two main types of field effect transistor, the **Junction Field Effect Transistor (JFET)** and the **Insulated-gate Field Effect Transistor (IGFET)**, which is more commonly known as the standard **Metal Oxide Semiconductor Field Effect Transistor (MOSFET)**.

3.1.2.1 The Junction Field Effect Transistor (JFET)

The JFET has a narrow "Channel" of N-type or P-type silicon with electrical connections at either end commonly called the DRAIN and the SOURCE respectively (See figure 3.6). If the channel is made of the N-type silicon, then it is an N-channel JFET. Both P-channel and N-channel FET's are available. Within this channel there is a third connection which is called the GATE and this can also be a P or N-type material forming a PN-junction. So if the channel is N-type, then the gate connection is attached to a P-type portion. A comparison can be made between the terminals of the FET and that of the BJT as shown below:

Bipolar Junction Transistor	Field Effect Transistor
-----------------------------	-------------------------

Emitter - (E)	Source - (S)
Base - (B)	Gate - (G)
Collector - (C)	Drain - (D)

The semiconductor "Channel" of the JFET is a resistive path through which a voltage V_{ds} causes a current I_d to flow through the channel. We learnt from our last lecture that at an unbiased PN-junction, there is a depletion region. The FET controls the current flow through them between the drain and source terminals by controlling the voltage applied to the gate terminal. This is achieved by controlling the biasing of the PN-junction. In an N-channel JFET this gate voltage is negative while for a P-channel JFET the gate voltage is positive. We will see why shortly.

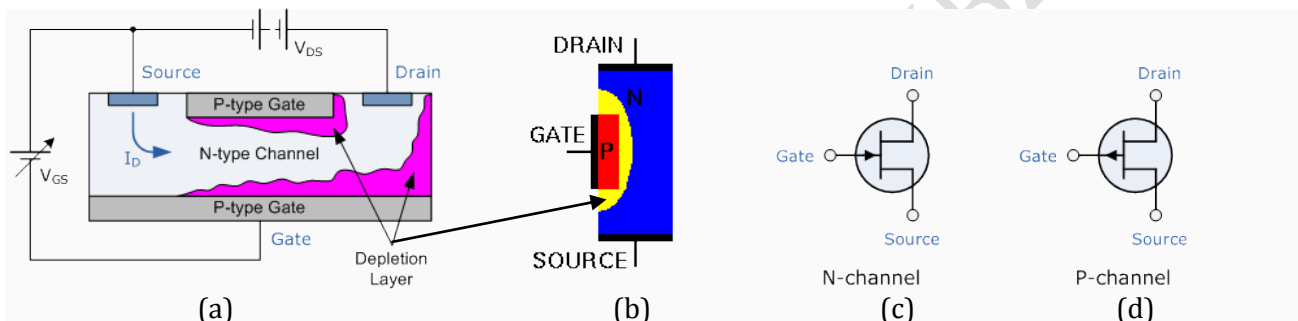


Figure 3.6: Arrangement for an N-channel JFET and corresponding circuit symbols.

Figure 3.6(a) shows the cross sectional diagram of an N-channel JFET with a P-type region called the gate diffused into the N-type channel forming a PN-junction and this junction forms the **depletion layer** around the gate area. Figure 3.6(b) is a simplified diagram of figure 3.6(a) which should make it easy for you to understand how FET works. This depletion layer restricts the current flow through the channel by reducing the effective width of the channel as can be seen in the areas coloured pink (fig 3.6a) and yellow (fig 3.6b) and thus increasing the overall resistance of the channel.

When the gate voltage V_g is equal to 0V and a small external voltage (V_{ds}) is applied between the drain and the source, maximum current (I_d) will flow through the channel slightly restricted by the small depletion layer. If a negative voltage (V_{gs}) is now applied to the gate (that is to reverse

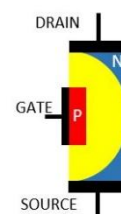


Figure 3.7: FET showing pinched

bias the junction) the size of the depletion layer begins to increase reducing the overall effective area of the channel and thus reducing the current flowing through it. As the gate voltage (V_{gs}) is made more negative, the width of the channel decreases until no more current flows between the drain and the source and the FET is said to be "**pinched-off**" (See figure 3.7). In this pinch-off region the gate voltage, V_{gs} controls the channel current and V_{ds} has little or no effect.

The voltage V_{gs} applied to the gate controls the current flowing between the drain and the source terminals. V_{gs} refers to the voltage applied between the gate and the source while V_{ds} refers to the voltage applied between the drain and the source. Because a **Field Effect Transistor** is a VOLTAGE controlled device, "NO current flows into the gate!" then the source current (I_s) flowing out of the device equals the drain current flowing into it and therefore ($I_d = I_s$).

The are four different regions of operation for a JFET and these are given as:

- Ohmic Region - The depletion layer of the channel is very small and the JFET acts like a variable resistor.
- Cut-off Region - The gate voltage is sufficient to cause the JFET to act as an open circuit as the channel resistance is at maximum.
- Saturation or Active Region - The JFET becomes a good conductor and is controlled by the gate-source voltage, (V_{gs}) while the drain-source voltage, (V_{ds}) has little or no effect.
- Breakdown Region - The voltage between the drain and source, (V_{ds}) is high enough to causes the JFET's resistive channel to break down and pass current.

The control of the drain current by a negative gate potential makes the JFET useful as a switch, just like BJTs. It is essential that the gate voltage is never positive for an N-channel JFET as the channel current will flow to the gate and not the drain resulting in damage to the JFET. The principals of operation for a P-channel JFET are the same as for the N-channel JFET, except that the polarity of the voltages need to be reversed. Next, we study the second type of FET.

3.1.2.2 The Metal Oxide Semiconductor Field Effect Transistor (MOSFET)

The MOSFET is another FET, whose Gate input is electrically insulated from the main current carrying channel and is therefore called an **Insulated Gate Field Effect Transistor**. The **MOSFET** type of FET has a "Metal Oxide" gate (usually silicon dioxide), which is electrically insulated from the main semiconductor N-channel or P-channel. So we can form a thin layer of silicon dioxide along one surface of the channel, and then lay our metal gate region down over the glass. The result is shown in figure 3.8. Silicon dioxide is simply glass, which is a good insulator.

With no voltage applied to the gate (G) electrode, the channel really is just a semiconductor resistance, and will conduct current according to the voltage applied between source (S) and drain (D). There is no PN-junction, so there is no depletion region. As the gate terminal is isolated from the main current carrying channel "NO current flows into the gate" and like the JFET, the **MOSFET** also acts like a voltage controlled resistor. It can easily accumulate large static charges resulting in the **MOSFET** becoming easily damaged unless carefully handled or protected.

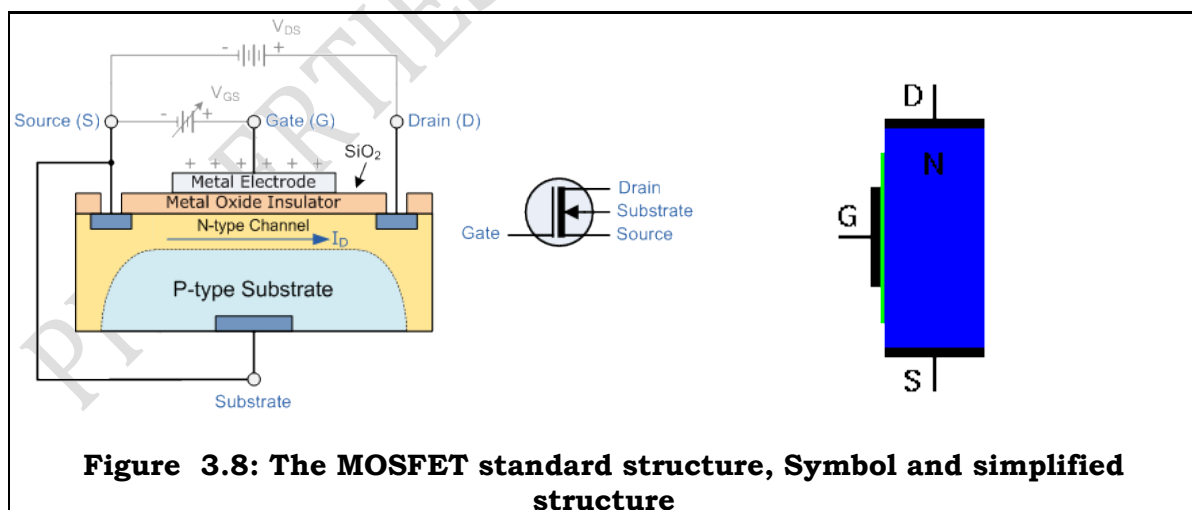


Figure 3.8: The MOSFET standard structure, Symbol and simplified structure

The MOSFET is specially valuable as electronic switches or to make logic gates because with no bias they are normally non-conducting and the high gate resistance means that very little

control current is needed. Both the P-channel and the N-channel MOSFET is available in two basic forms, the **Enhancement** type and the **Depletion** type.

Depletion-mode MOSFET

The **Depletion-mode MOSFET**, which is less common than the enhancement types is normally switched "ON" without a gate bias voltage but requires a gate to source voltage (V_{gs}) to switch the device "OFF". Similar to the JFET types. For N-channel MOSFET's a "Positive" gate voltage widens the channel, increasing the flow of the drain current and decreasing the drain current as the gate voltage goes more negative. The opposite is also true for the P-channel types. The depletion mode MOSFET is equivalent to a "Normally Closed" switch.

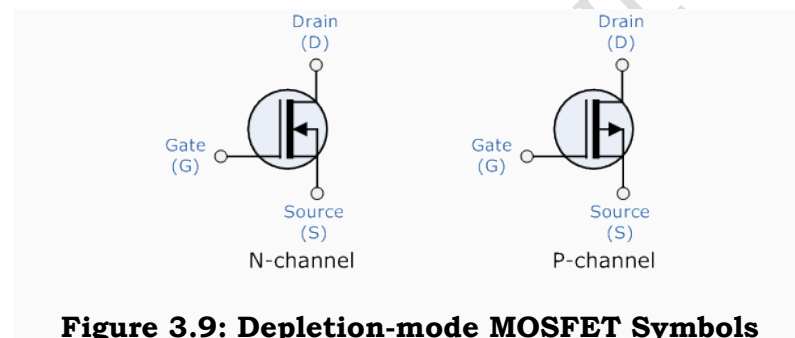


Figure 3.9: Depletion-mode MOSFET Symbols

Depletion-mode MOSFET's are constructed similar to their JFET transistor counterparts where the drain-source channel is inherently conductive with electrons and holes already present within the N-type or P-type channel. This doping of the channel produces a conducting path of low resistance between the drain and source with zero gate bias. ". Because this type of FET operates by creating a depletion region within an existing channel, it is called a *depletion-mode MOSFET*

Enhancement-mode MOSFET

The more common **Enhancement-mode MOSFET** is the reverse of the depletion-mode type. Here the conducting channel is lightly doped or even undoped making it non-conductive. This results in the device being normally "OFF" when the gate bias voltage is equal to zero. A drain current will only flow when a gate voltage (V_{gs}) is applied to the gate terminal. This positive

voltage creates an electrical field within the channel attracting electrons towards the oxide layer and thereby reducing the overall resistance of the channel allowing current to flow. Increasing this positive gate voltage will cause an increase in the drain current, I_d through the channel. Then, the Enhancement-mode device is equivalent to a "Normally Open" switch.

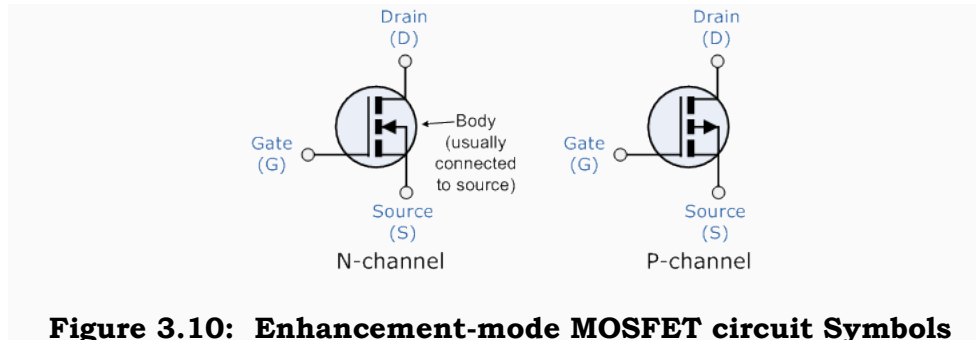


Figure 3.10: Enhancement-mode MOSFET circuit Symbols

Enhancement-mode MOSFET's make excellent electronics switches due to their low "ON" resistance and extremely high "OFF" resistance and extremely high gate resistance. Enhancement-mode MOSFET's are used in integrated circuits to produce CMOS type **Logic Gates** and power switching circuits as they can be driven by digital logic levels.

The **MOSFET** has an extremely high input gate resistance and as such a easily damaged by static electricity if not carefully protected. MOSFET's are ideal for use as electronic switches or common-source amplifiers as their power consumption is very small. Typical applications for MOSFET's are in Microprocessors, Memories, Calculators and Logic Gates etc. Also, notice that the broken lines within the symbol indicates a normally "OFF" Enhancement type showing that "NO" current can flow through the channel when zero gate voltage is applied and a continuous line within the symbol indicates a normally "ON" Depletion type showing that current "CAN" flow through the channel with zero gate voltage.

Summary

- The **Bipolar Junction Transistor** (BJT) is a three layer device constructed from two semiconductor diode junctions joined together, one forward biased and one reverse biased.
- There are two main types of bipolar junction transistors, the NPN and the PNP transistor.
- Transistors are "**Current Operated Devices**" where a much smaller Base current causes a larger Emitter to Collector current, which themselves are nearly equal, to flow.
- A transistor can also be used as an electronic switch to control devices such as lamps, motors and solenoids etc.
- Transistor switches can be used to switch and control lamps, relays or even motors.
- When using bipolar transistors as switches they must be fully "OFF" or fully "ON".
- Transistors that are fully "ON" (fully conducting) are said to be in their **Saturation** region.
- Transistors that are fully "OFF" (fully non-conducting) are said to be in their **Cut-off** region.
- In a transistor switch a small Base current controls a much larger Collector current.
- The NPN transistor requires the Base to be more positive than the Emitter while the PNP type requires that the Emitter is more positive than the Base.
- **Field Effect Transistors**, or FET's are "**Voltage Operated Devices**" and can be divided into two main types: Junction-gate devices called JFET's and Insulated-gate devices called IGFET's or more commonly known as MOSFET's.
- Insulated-gate devices can also be sub-divided into Enhancement types and Depletion types. All forms are available in both N-channel and P-channel versions.
- FET's have very high input resistances so very little or no current (MOSFET types) flows into the input terminal making them ideal for use as electronic switches.

- The input impedance of the MOSFET is even higher than that of the JFET due to the insulating oxide layer and therefore static electricity can easily damage MOSFET devices so care needs to be taken when handling them.
- FET's have very large current gain compared to junction transistors.
- They can be used as ideal switches due to their very high channel "OFF" resistance, low "ON" resistance.

Post-Test

1. Why are BJTs referred to as current operated devices?
2. Draw a comparison between the terminals of the BJT and the FET.
3. Why are FETs said to be voltage operated devices?
4. Draw a tree-like diagram of the transistor family.
5. Why do you think the enhancement mode MOSFET is preferred to the depletion mode MOSFET with respect to electronic switches?
6. When is a BJT said to be saturated?
7. Can you recall the different circuit symbols of all the transistors studied in this lecture?

References

- Crowe, J., and Barrie Hayes-Gill. 1998. Introduction to Digital Electronics.
- Duncan, Tom. 1997. Electronics for Today and Tomorrow. Spain: John Murray.
- Heuring, Vincent P., and Harry F. Jordan. 2004. Computer Systems Design and Architecture 2nd Edition. Prentice Hall.
- Hwang, Enoch O. 2005. Digital Logic and Microprocessor Design With VHDL. Brooks/ Cole.
- Kuphaldt Tony R., 2009, Lessons In Electric Circuits, Volume III – Fifth Edition Semiconductors
- Rocco, Ronald J., and Neal S. Wilmer. 2001. Digital Systems, Principles and Applications, Eighth Edition. New Jersey: Prentice Hall.
- Theraja, B.L. and Theraja, A.K., 2005. A textbook of electrical technology, S. Chand.

LECTURE FOUR

DIGITAL LOGIC FAMILIES

Introduction

In the previous lecture, we studied what transistors are and how they can be used as electronic switches. The main purpose of transistor switches is to enable researchers control electronic circuits using small devices. By controlling the circuits, logical decisions are performed by the circuits. In this lecture, we will study the characteristics of integrated circuits which are in turn built using the two main types of transistors discussed in the previous chapter.

Objectives

At the end of this lecture, you should be able to:

1. Recall and compare the properties of TTL and CMOS integrated circuits
2. Define the terms: Fan out, propagation delay
3. Specify the different logic levels for the logic families
4. Define what an integrated circuit is as well as their groups.

Pre-Test

1. What is a transistor and how is it used?
2. List the two main types of transistors?

4.1 Integrated Circuits

An integrated circuit (IC) is a small semiconductor-based electronic device consisting of fabricated transistors, resistors and capacitors in varying numbers. Integrated circuits are the building blocks of most electronic devices and equipment. An integrated circuit, also known as a chip or microchip, can function as an amplifier, oscillator, timer, counter, computer memory, or microprocessor.

Integrated Circuits can be grouped together into families according to the number of transistors or "gates" that they contain. As we will see in the next lecture, a simple AND gate may contain only a few individual transistors, whereas a more complex microprocessor may contain many thousands of individual transistor gates. Integrated circuits are categorized according to the number of logic gates or the complexity of the circuits within a single chip with the general classification for the number of individual gates given as:

Classification of Integrated Circuits

- Small Scale Integration or (SSI) - Contain up to 10 transistors or a few gates within a single package such as AND, OR, NOT gates that we will study in the next lecture.
- Medium Scale Integration or (MSI) - between 10 and 100 transistors within a single package and perform digital operations such as adders, decoders, counters, flip-flops and multiplexers.
- Large Scale Integration or (LSI) - between 100 and 1,000 transistors or hundreds of gates and perform specific digital operations such as I/O chips, memory, arithmetic and logic units.
- Very-Large Scale Integration or (VLSI) - between 1,000 and 10,000 transistors or thousands of gates and perform computational operations such as processors, large memory arrays and programmable logic devices.
- Super-Large Scale Integration or (SLSI) - between 10,000 and 100,000 transistors within a single package and perform computational operations such as microprocessor chips, micro-controllers and calculators.
- Ultra-Large Scale Integration or (ULSI) - these contain more than 1 million transistors and are used in computers CPUs and other complex devices.
- Giant Scale Integration (GSI): contain much more than 2000000 components per chip.

While the "ultra large scale" ULSI classification is less well used, another level of integration which represents the complexity of the Integrated Circuit is known as the **System-on-Chip** or (**SOC**) for short. Here the individual components such as the microprocessor, memory, peripherals, I/O logic etc., are all produced on a single piece of silicon and which represents a whole electronic system within one single chip, literally putting the word "integrated" into integrated circuit. These chips are generally used in mobile phones, digital cameras, micro-controllers and robotic applications to mention a few, and which can contain up to 100 million individual silicon-transistor gates within one single package.

In Digital Designs, our primary aim is to create an Integrated Circuit (IC). A Circuit configuration or arrangement of the circuit elements in a special manner will result in a particular Logic Family which we will discuss next.

4.2 Logic Families

A logic family is a collection of different IC chips that have similar input, output and internal circuit characteristics i.e. group of compatible ICs with same logic levels and supply voltages but perform different logic functions.

4.3 Classification of Logic Families

Logic families are mainly classified as two types namely the Bipolar logic family and the Unipolar logic family. From our previous lecture, I guess you have an idea what these mean. However we will still study them next.

4.3.1 Bipolar Logic Families

This logic family uses mainly bipolar devices like diodes, transistors as active elements, in addition to passive elements like resistors and capacitors. They include:- Transistor-Transistor

Logic (TTL), Resistor-Transistor Logic (RTL), Direct Coupled Transistor Logic (DCTL), Diode Transistor Logic (DTL), High Threshold Logic (HTL), Integrated Injection Logic (IIL or I2L), Schottky TTL, Emitter Coupled Logic (ECL). These are sub classified as saturated bipolar logic family and unsaturated bipolar logic family. If you would like to know more, then read further.

4.3.2 Unipolar Logic Families

These mainly uses Unipolar devices like MOSFETs as active elements in addition to passive elements like resistors and capacitors. These logic families have the advantages of high speed and lower power consumption than Bipolar families. These are classified as P-Channel MOS Logic Family (PMOS), N-Channel MOS Logic Family (NMOS) and Complementary Metal Oxide Semiconductor Logic Family (CMOS).

Although there are many ways of making ICs and several technologies are available, as discussed above, to the manufacturer, two major 'families' of general-purpose logic ICs remain the most important. The TTL family and the CMOS family (CMOS is pronounced 'sea-moss'). Both families include a very large number of different devices: gates, flip-flops, counters, registers and many other 'building-block' elements. For the rest of this lecture, we will discuss the TTLs and the CMOS, but first we will look at the basic operating characteristics and parameters of logic families.

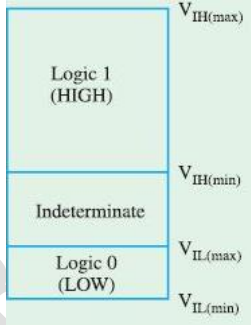
4.4 Operating Characteristics and Properties of Logic Families

When studying logic families, it is important to get familiar with their basic operational properties. These properties specify how each logic family operates. For the purpose of this lecture, we will study the following main characteristics of Logic families: Voltage level, Fan-in, Fan-out, Noise Immunity, Noise margin, Power Dissipation, Propagation delay.

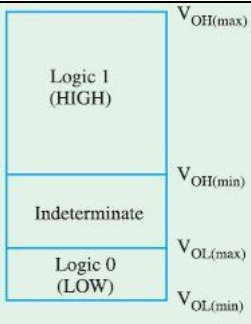
4.4.1 DC Voltage Levels

The voltage levels allowed for any logic family are described here, with reference to Figure 4.1 and Figure 4.2. These values differ for the different logic families. First we define the terms and

later we specify the values for the two main logic families we will discuss. Figure 4.1 shows the range of the allowable input voltage levels. Values outside the given range are not allowed. Therefore, for proper operation the input voltage levels to a logic must be kept outside the indeterminate range. The terms are:

<p>$V_{IH(max)}$ – is the maximum voltage required at an input to be recognized as a logic level “1”.</p> <p>$V_{IH(min)}$ – is the minimum voltage required at an input to be recognized as a logic level “1”.</p> <p>$V_{IL(max)}$ – is the maximum voltage required at an input that will still be recognized as a logic level “0”.</p> <p>$V_{IL(min)}$ is the minimum voltage required at an input that will be recognized as a logic level “0”.</p>	 <p>Figure 4.1: Input Voltage level</p>
---	---

The input voltage level values differ from the output voltage levels shown in figure 4.2 which are defined as:

<p>$V_{OH(max)}$ – This is the maximum voltage level at an output in the logical “1” state under defined load conditions. That is the minimum value of output recognized as a ‘logic 1’</p> <p>$V_{OH(min)}$ – is the minimum value of output recognized as a ‘logic 1’</p> <p>$V_{OL(max)}$ – is the maximum voltage level at an output in the logical “0” state under defined load conditions.</p> <p>$V_{OL(min)}$ – is the minimum voltage level at an output in the logical “0” state</p>	 <p>Figure 4.2: Output Voltage level</p>
--	--

4.4.2 Noise Immunity

Noise is present in all real systems. Noise adds random fluctuations to voltages representing logic levels. *Noise Immunity* is the maximum noise that a circuit can withstand without affecting the output. It refers to the circuit’s ability to tolerate noise without causing a false change in its output voltage. The noise voltage is usually produced by stray electric and magnetic fields on the connecting wires in the circuit. Sometimes, too much noise voltage can cause the voltage at the input to drop below $V_{IH(min)}$ or rise above $V_{IL(max)}$.

4.4.3 Noise Margin

The noise margin of a logic family is a quantitative measure of the circuit's noise immunity (see figure 4.3). It is expressed in volts and there are usually two values of noise margin specified for a given family.

1. The high level noise margin $V_{NH} = V_{OH(min)} - V_{IH(min)}$
2. The low level noise margin $V_{NL} = V_{IL(max)} - V_{OL(max)}$

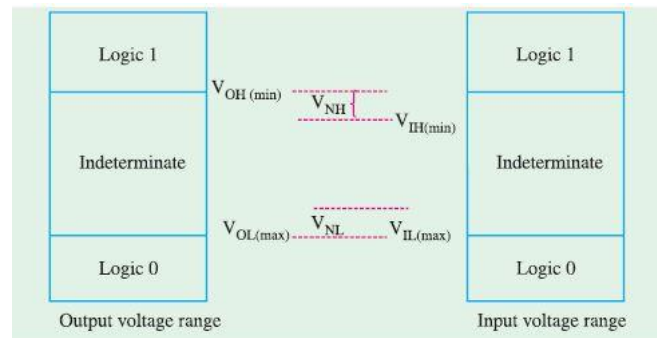


Figure 4.3: Noise Margin

4.4.4 Propagation Delay

The propagation delay of a logic family is the amount of time that it takes for a change in input signal to produce a change in output signal. That is, the propagation delay of a gate is basically the time interval between application of input signal and occurrence of output signal. It is an important characteristic because it limits the speed at which the logic family operates.

4.4.5 Fan-out

The *fan-in* of any logic family determines the number of inputs the logic gate can handle, however, a more important characteristic of a logic family is the *Fan-out*. The Fan-out of a logic family is the maximum number of logic inputs (of the same logic family) that an output can drive reliably. It determines the number of circuits that a gate can drive and it is also known as the *loading factor*. (see figure 4.4).

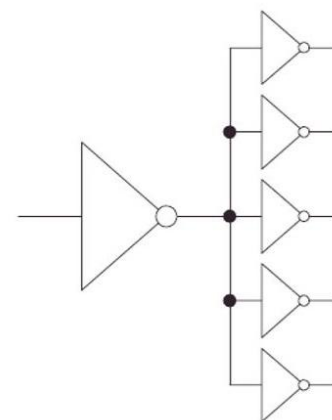


Figure 4.4: Logic Family Loading Factor

There are other characteristics of logic families, but the one discussed above are the very important ones to note. Others include, power dissipation, power consumption, packing density, etc. Next we are going to look at the values for the two main logic families we will be discussing for the purpose of this class. These are the TTLs and CMOS.

4.5 Transistor-Transistor Logic Family (TTL)

There are a large variety of logic gate types in the TTL logic family and these can be identified by their 7400 series code such as 74Lxx, 74LSxx, 74ALSxx, 74HCxx, 74HCTxx. The **TTL** integrated circuits use NPN (or PNP) type BJTs for input and output circuitry, but the **CMOS** family are more common. The TTL logic system uses "Positive logic", in which a logic level "0" or "LOW" is represented by a zero voltage, 0v or ground and a logic level "1" or "HIGH" is represented by a higher voltage such as +5 volts, with the switching from one voltage level to the other, from either a logic level "0" to a "1" or a "1" to a "0" being made as quickly as possible to prevent any faulty operation of the logic circuit. In standard TTL IC's the pre-defined voltage range for the input and output voltage levels which define exactly what is a logic "1" level and what is a logic "0" level are shown in figure 4.5.

TTL Input & Output Voltage Levels

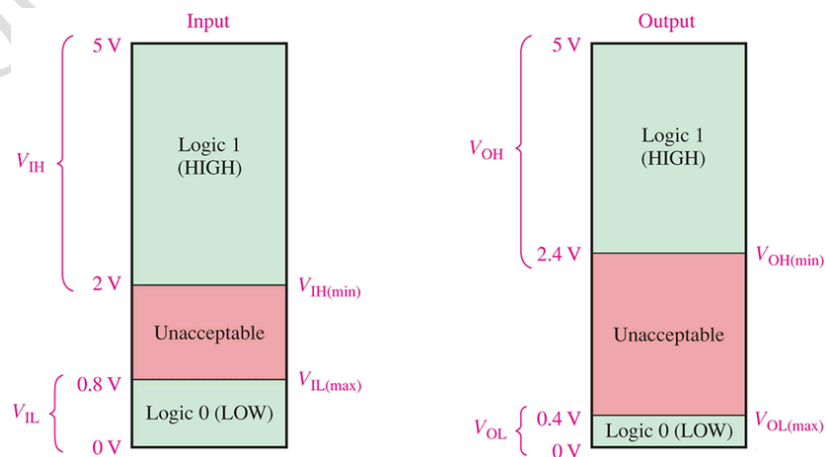


Figure 4.5: TTL input/ output Voltage levels

As shown in figure 4.5, when using a standard +5 volt supply any TTL voltage input between 2.0v and 5v is considered to be a logic "1" or "HIGH" while any voltage input below 0.8v is recognized as a logic "0" or "LOW". The voltage region in between these two voltage levels either as an input or as an output is called the *Indeterminate Region* and is unacceptable. Operating within this region may cause the logic gate to produce a false output.

In addition to the voltage level characteristic discussed above, TTL IC's require a stabilized 5V $\pm 0.25V$ d.c. power supply. The current required is in Milliamperes while the input impedance is low. The high level and low level noise margin for TTL is 0.4V. The fan-out for TTLs, range between 10 and 40 depending on the sub-family. Table 4.1 shows this as well as other characteristics of the various sub-families.

Table 4.1: Characteristics of TTL sub-family

Characteristic	TTL sub-family					
	74	74S	74LS	74AS	74ALS	74F
Propagation delay (ns)	9	3	9.5	1.7	4	3
Power Dissipation (mW)	10	20	2	8	1.2	6
Speed-power product (pJ)	90	60	19	13.6	4.8	18
Fan-out	10	20	20	40	20	33

One of the main disadvantages of the TTL logic series is that the gates are based on bipolar transistor logic technology and as transistors are current operated devices, they consume large amounts of power from a fixed +5 volt power supply. Also, TTL bipolar transistor gates have a limited operating speed when switching from an "OFF" state to an "ON" state and vice-versa called "propagation delay". To overcome these limitations complementary MOS called "CMOS" logic gates using "Field Effect Transistors" or FET's were developed.

4.6 Complementary Metal Oxide Semiconductor Logic Family (CMOS)

CMOS integrated circuits are more common than the TTL IC's. The CMOS ICs use FET's for both their input and output circuitry. The CMOS logic family uses a different level of voltages compared to the TTL types with a logic "1" level operating between 3.0 and 18 volts and a logic "0" level below 1.5 volts. Figure 4.6 shows the acceptable voltage levels for the CMOS family. As shown in figure 4.6, any CMOS voltage input between 3.5 V and 5V is considered to be a logic "1" or "HIGH" while any voltage input below 1.5 V is recognized as a logic "0" or "LOW". The voltage region in between these two voltage levels either as an input or as an output is called the *Indeterminate Region* and is unacceptable. As you can see from the figure, the output voltage levels, for CMOS logic families is very close to both margins. That is, an output logic level 1 ranges from 4.9 V to 5 V, while a logic level '0' is from 0 V to 0.1 V.

In addition to the voltage level characteristic, CMOS IC's will work on any unstabilized d.c. voltage between 3 V and 15 V power supply.. The input impedance of CMOS family is very high because of the FETs, this however makes the current required to be in Microamperes because the FETs are voltage operated devices. The high level and low level noise margin for CMOS is 1.45 V while their fan-out is 50.

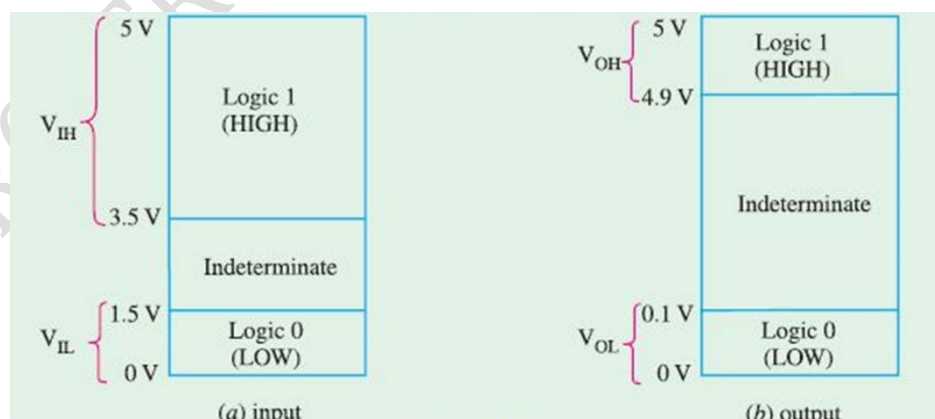


Figure 4.6: CMOS family input/ output voltage levels

Summary

To summarize what we have learned in this lecture, let us consider the table below that shows a comparison the two main logic families discussed in the lecture.

S.No	PARAMETER	CMOS	TTL
1	Device Used	nMOS, pMOS	BJT
2	$V_{IH(min)}$	3.5 V	2 V
	$V_{IL(max)}$	1.5 V	0.8 V
	$V_{OH(min)}$	4.95 V	2.4 V
	$V_{OL(max)}$	0.005 V	0.4 V
3	HIGH level Noise margin	1.45 V	0.4 V
4	LOW level Noise margin	1.45 V	0.4 V
5	Noise immunity	> TTL	< CMOS
6	Propagation Delay	70 ns	10 ns
7	Switching Speed	< TTL	>> CMOS
8	Power Dissipation/Gate	0.1 mW	10 mW
9	Speed-Power Product	0.7 pJ	100 pJ
10	Fan-out	50	10
11	Power Supply Voltage	3-15 V	Fixed 5V

Post-Test

1. Define the terms: Propagation delay, Fan-out, Noise immunity.
2. How are integrated circuits categorized?
3. Differentiate between the TTL logic family and the CMOS logic family.

References

- Duncan, Tom. 1997, Electronics for Today and Tomorrow. Spain: John Murray.
- Hwang, Enoch O. 2005. Digital Logic and Microprocessor Design With VHDL. Brooks/ Cole.
- Theraja, B.L. and Theraja, A.K., 2005. A textbook of electrical technology, S. Chand.
- Watson J. (1990) Logic families. In: Mastering Electronics. Macmillan Master Series. Palgrave, London
www.electronics-tutorials.ws/logic/logic_8.html

LECTURE FIVE

DIGITAL LOGIC GATES

Introduction

Digital electronics is concerned with two-state, switching-type circuits or devices. Their outputs and inputs involve only two levels of voltage referred to as high (logic level '1') and low (logic level '0'). In the fourth lecture, we looked at the two main logic families in use today for building integrated circuits as well as their characteristics that represent the two-state logic levels. In this chapter we will study digital logic gates which are composed of transistors.

Objectives

At the end of this lecture, you should be able to:

1. Recall and draw the logic symbols for the AND, OR, NAND, NOR, ExOR, ExNOR, NOT and BUF logic gates
2. Draw other logic gates using only the NAND or NOR logic gates
3. Recall the truth table for all the logic gates.

Pre-Test

1. When transistors are used as switches, what is their mode of operation?
2. What voltage ranges are regarded as logic '1'?
3. What voltage ranges are regarded as logic '0'?

5.1 Digital Logic States

As written in the introduction, in digital logic design only two voltage levels or states are allowed and these states are generally referred to as Logic "1" and Logic "0", High and Low, True and False and which are represented in Boolean Algebra and Truth Tables by the binary digits of "1" and "0" respectively. A good example of a digital signal is a simple light as it is either "ON" or "OFF" but not both at the same time.

5.2 Digital Logic Gate

A **Digital Logic Gate** is an electronic device that makes logical decisions based on the different combinations of digital signals present on its inputs. The Digital Logic Gate is the basic building block from which all digital electronic circuits and microprocessor based systems are constructed from. Basic digital logic gates perform logical operations of AND, OR and NOT on binary numbers. A digital logic gate may have more than one input *but only has one digital output*. Standard commercially available digital logic gates are available in two basic families, **TTL** such as the 7400 series, and **CMOS** which is the 4000 series of chips. When studying logic gates, it is necessary to specify three things about the gate. (1) the logic symbol, which is a diagram of how that gate is represented in a circuit; (2) the truth table, which is a table that gives all the possible input signal combinations as well as the corresponding output signal. (3) Boolean Expression, which is an expression that represents the gate in Boolean algebra. Figure 5.1 shows a digital logic gate as they are manufactured. It would be interesting to know what the circuit for these logic gates are! These we will study next.



Figure 5.1: Digital Logic Gate

5.2.1 The Logic "AND" Gate

The Logic AND Gate is a digital logic gate that has an output only goes "HIGH" to a logic level "1" when **ALL** of its inputs are at logic level "1". The output of the gate only returns "LOW" again when **any** of its inputs goes to a logic level "0". The Boolean expression for a logic AND gate is that for *Logical Multiplication* which is denoted by a single dot or full stop symbol, (.) giving us the Boolean expression of: $A \cdot B = Q$. Therefore, the operation of a 2-input logic AND gate can be described as:

"If both A and B are true, then Q is true"

A simple 2-input logic AND is shown in figure 5.2 with the inputs connected directly to the transistor bases A and B. Both transistors must be saturated "ON" for an output to be read at Q.

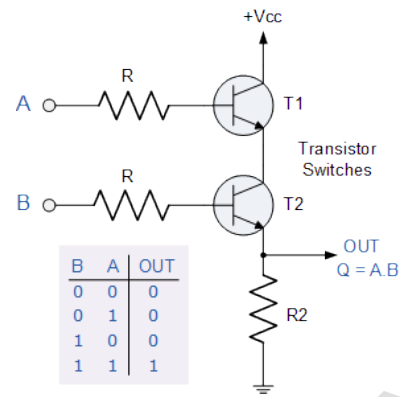




Figure 5.2: 2-input AND gate using TTL

The standard symbol and truth table for a 2-input AND gate is shown below. It is okay to omit the ampersand character inside the symbol.

Symbol	Truth Table		
 <p>2-input AND Gate</p>	A	B	Q
	0	0	0
	0	1	0
	1	0	0
	1	1	1
Boolean Expression Q=A.B	Read as A AND B gives Q		

The 3-input AND Gate symbol and truth table are shown below.

Symbol	Truth Table			
 <p>3-input AND Gate</p>	A	B	C	Q
	0	0	0	0
	0	0	1	0
	0	1	0	0
	0	1	1	0
	1	0	0	0
	1	0	1	0
	1	1	0	0
	1	1	1	1
	Boolean Expression Q=A.B.C	Read as A AND B AND C gives Q		

Commonly available digital logic AND gate IC's include: 74LS08 Quad 2-input and CD4081 Quad 2-input.

5.2.2 The Logic "OR" Gate

A Logic OR Gate or Inclusive-OR gate is a digital logic gate that has an output which only goes "HIGH" to a logic level "1" when ANY of its inputs are at logic level "1". The output of a Logic OR Gate only returns "LOW" again when all of its inputs are at a logic level "0". The Boolean expression for a logic OR gate is denoted by a plus sign, (+) giving us the Boolean expression of: $A+B = Q$. We can then define the operation of a 2-input logic OR gate as:

"If either A or B is true, then Q is true"

Figure 5.3 shows a simple 2-input logic OR gate constructed using transistor switches connected together as shown in the figure with the inputs connected directly to the transistor bases. Either transistor must be saturated "ON" for an output at Q to be high. The symbol of the 2-input OR gate and its truth table are presented below:

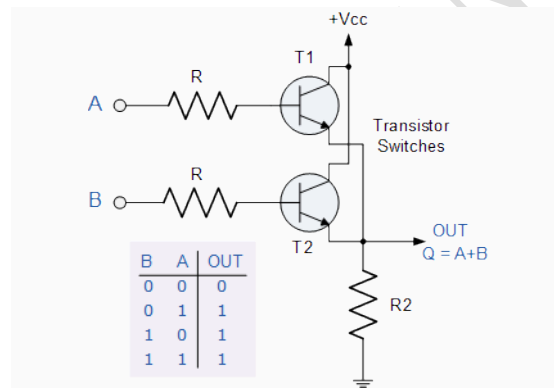
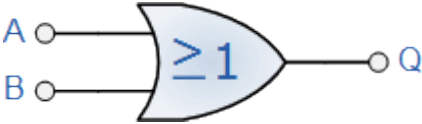
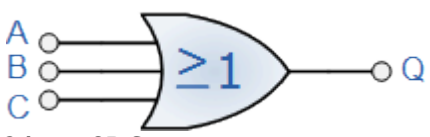


Figure 5.3: A 2-input logic OR gate

Symbol	Truth Table															
 <p>2-input OR Gate</p>	<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>Q</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	A	B	Q	0	0	0	0	1	1	1	0	1	1	1	1
	A	B	Q													
	0	0	0													
	0	1	1													
	1	0	1													
1	1	1														
<p>Boolean Expression $Q = A+B$</p>	<p>Read as A OR B gives Q</p>															

Likewise, the 3-input OR Gate symbol and truth table are below:

Symbol	Truth Table																																				
 <p>3-input OR Gate</p>	<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>C</th> <th>Q</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	A	B	C	Q	0	0	0	0	0	0	1	1	0	1	0	1	0	1	1	1	1	0	0	1	1	0	1	1	1	1	0	1	1	1	1	1
	A	B	C	Q																																	
	0	0	0	0																																	
	0	0	1	1																																	
	0	1	0	1																																	
	0	1	1	1																																	
	1	0	0	1																																	
	1	0	1	1																																	
	1	1	0	1																																	
1	1	1	1																																		
<p>Boolean Expression $Q = A+B+C$</p>	<p>Read as A OR B OR C gives Q</p>																																				

Commonly available OR gate IC's include are 74LS32 Quad 2-input and CD4071 Quad 2-input

5.2.3 The Logic "NOT" Gate

The digital Logic NOT Gate is another logical gate that is sometimes referred to as an Inverting Buffer or simply a Digital Inverter. It is a single input device which has an output at logic level "1" when its single input is at logic level "0", in other words it "inverts" (complements) its input signal. The Boolean expression is $Q = \bar{A}$. Then we can define the operation of a single input logic NOT gate as being: **"If A is NOT true, then Q is true"**. A simple transistor logic NOT gate is shown in figure 5.5, with the input connected directly to the transistor base. The transistor must be saturated, "ON" for an inversed output "OFF" to be read at Q.

The standard NOT gate is given a symbol whose shape is a triangle pointing to the right with a circle at its end as shown below. This circle is known as an "inversion bubble" and is used in NOT, NAND, NOR and XNOR

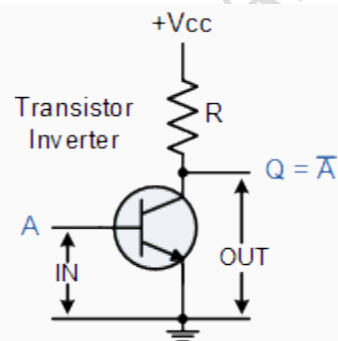


Figure 5.4: The digital NOT gate

symbols at their output to represent the logical operation of the NOT function. This bubble denotes a signal inversion (complementation) of the signal and can be present on either or both the output and/or the input terminals.

Symbol	Truth Table	
<p>Inverter or NOT Gate</p>	A	Q
	0	1
	1	0
Boolean Expression is $Q = \bar{A}$.	Read as Q is NOT A	

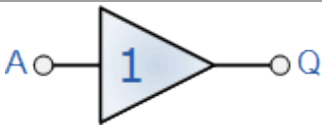
Logic NOT gates provide the complement of their input signal and are so called because when their input signal is "HIGH" their output state will be "LOW". Likewise, when their input signal is "LOW" their output state will be "HIGH".

Commonly available logic NOT gate and Inverter IC's include 74LS04 Hex Inverting NOT Gate and CD4009 Hex Inverting NOT Gate.

5.2.4 The "BUFFER"

The digital Logic Gate referred to as the *buffer*, performs absolutely no logical operation! Surprised! Well, the buffer is the opposite of the NOT gate. It is a single input device with the logic output signal exactly as its input signal. The Boolean expression is $Q = A$. This means that we can define the operation as being: **"If A is true, then Q is true"**. Using Figure 5.4 as a reference, can you figure out what the transistor circuit configuration would look like?

The symbol as well the truth table of the BUF is presented below.

Symbol	Truth Table	
	A	Q
	1	1
	0	0
BUF		
Boolean Expression is $Q = A$.	Read as Q is NOT A	

Some authors argue however, that the Buffer can be used as a time delay device that holds a logical data and gives out the same signal when needed.

5.2.5 The Logic "NAND" Gate

The Logic NAND (Not - AND) Gate can be seen as a cascading of the digital logic AND gate with the NOT gate connected together in series. The NAND gate has an output that is normally at logic level "1" and only goes "LOW" to logic level "0" when **ALL** of its inputs are at logic level "1". This makes the NAND Gate the reverse form of the AND gate we saw previously. The Boolean expression for a logic NAND gate is denoted by a single dot or full stop symbol, (.) with a line or *bar*, ($\bar{\quad}$) over the expression to signify the NOT of the NAND gate giving us the Boolean expression of: $Q = \overline{A \cdot B}$. However, as earlier noted, the dot can be omitted. The operation of a 2-input logic NAND gate is described as: **"If either A or B are NOT true, then Q is true"**.

Figure 5.5 shows a simple 2-input logic NAND gate constructed transistor switches. For a logic level "1" output to be read at Q, then both or at least one of the transistors must be off.

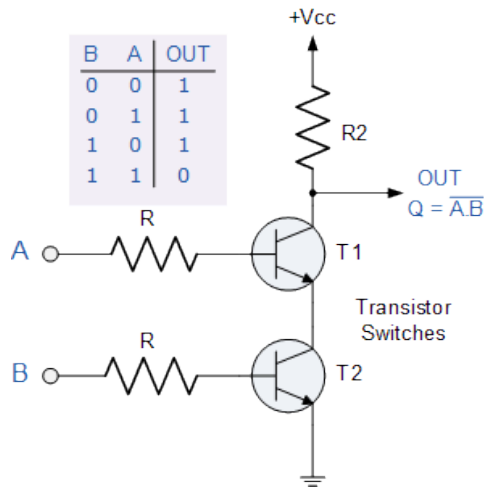


Figure 5.5: The logic NAND gate using transistors

The usual symbol and truth table for a 2-input NAND Gate is shown below.

Symbol	Truth Table															
<p>2-input NAND Gate</p>	<table border="1"> <thead> <tr> <th>B</th> <th>A</th> <th>Q</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	B	A	Q	0	0	1	0	1	1	1	0	1	1	1	0
	B	A	Q													
	0	0	1													
	0	1	1													
	1	0	1													
1	1	0														
<p>Boolean Expression : $Q = \overline{A \cdot B}$</p>	<p>Read as A AND B gives NOT Q</p>															

While the usual symbol and truth table for a 3-input NAND Gate is shown below:

Symbol	Truth Table																																				
<p>3-input NAND Gate</p>	<table border="1"> <thead> <tr> <th>C</th> <th>B</th> <th>A</th> <th>Q</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	C	B	A	Q	0	0	0	1	0	0	1	1	0	1	0	1	0	1	1	1	1	0	0	1	1	0	1	1	1	1	0	1	1	1	1	0
	C	B	A	Q																																	
	0	0	0	1																																	
	0	0	1	1																																	
	0	1	0	1																																	
	0	1	1	1																																	
	1	0	0	1																																	
	1	0	1	1																																	
	1	1	0	1																																	
	1	1	1	0																																	
<p>Boolean Expression : $Q = \overline{ABC}$</p>	<p>Read as A AND B AND C gives NOT Q</p>																																				

Commonly available logic NAND gate IC's include: 74LS00 Quad 2-input, 74LS10 Triple 3-input, CD4011 Quad 2-input.

5.2.6 The Logic "NOR" Gate

The Logic NOR (Not - OR) Gate or sometimes referred to as Inclusive-NOR gate is a combination of the digital logic OR gate with a NOT gate connected together in series. The NOR gate has an output that is normally at logic level "1" and only goes "LOW" to logic level "0" when **ANY** of its inputs are at logic level "1". The Logic NOR Gate is the reverse form of the OR gate we have seen previously. The Boolean expression for a logic NOR gate is denoted by a plus sign, (+) with a line or *Overline*, ($\bar{\quad}$) over the expression to signify the NOT operation of the NOR gate giving us the Boolean expression of: $Q = \overline{A + B}$. We can then define the operation of a 2-input logic NOR gate as being: **"If both A and B are NOT true, then Q is true"**.

Figure 5.6 shows a simple 2-input logic NOR gate constructed using transistor switches with the inputs connected directly to the transistor bases. Both transistors must be cut-off "OFF" for a logic level "1" output to be read at Q. The symbol, truth table and Boolean expression of a 2-input NOR Gate is given below.

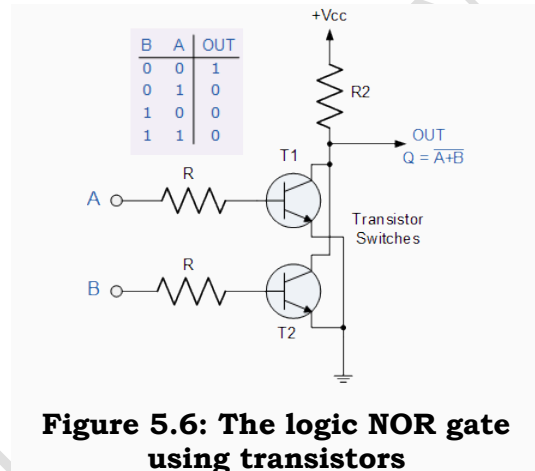


Figure 5.6: The logic NOR gate using transistors

The symbol, truth table and Boolean expression of a 2-input NOR Gate is given below.

Symbol	Truth Table		
<p>2-input NOR Gate</p>	A	B	Q
	0	0	1
	0	1	0
	1	0	0
	1	1	0
Boolean Expression $Q = \overline{A + B}$	Read as neither A NOR B gives Q		

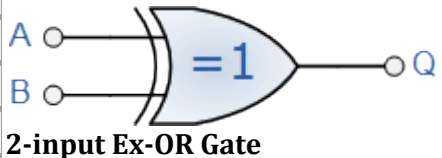
The 3-input NOR Gate symbol and truth table are presented below.

Symbol	Truth Table			
<p>3-input NOR Gate</p>	A	B	C	Q
	0	0	0	1
	0	0	1	0
	0	1	0	0
	0	1	1	0
	1	0	0	0
	1	0	1	0
	1	1	0	0
	1	1	1	0

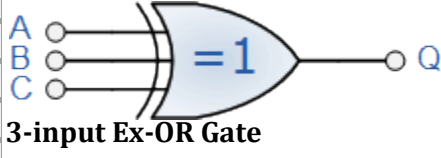
	1	1	1	0
Boolean Expression $Q = \overline{A + B + C}$	Read as A OR B OR C gives NOT Q			

5.2.7 The Exclusive-OR Gate (XOR)

Previously, we saw that for a 2-input OR gate, if A = "1", OR B = "1", OR BOTH A and B are equal to "1" then the output from the gate is also at logic level "1". This is known as an Inclusive-OR function because it *includes* the case of Q = "1" when both A and B = "1". If however, an output "1" is obtained **ONLY** when A = "1" or when B = "1" but **NOT** when both together at the same time are equal to "1", then this type of gate is known as an Exclusive-OR function or an Ex-OR function for short because it *excludes* the "OR BOTH" case of Q = "1" when both A = B = "1". Therefore, the output of a 2-input Exclusive-OR gate **ONLY** goes "HIGH" when its two input terminals are at "DIFFERENT" logic levels with respect to each other, that is when $A \neq B$. Exclusive-OR Gates are used mainly to build circuits that perform arithmetic operations and calculations especially Adders and Half-Adders. The symbol, truth Table and Boolean expression for a 2-input Exclusive-OR gate is given below.

Symbol	Truth Table		
 <p>2-input Ex-OR Gate</p>	A	B	Q
	0	0	0
	0	1	1
	1	0	1
	1	1	0
Boolean Expression $Q = A \oplus B$	Read as A OR B but NOT BOTH gives Q		

In general, an Ex-OR gate will give an output value of logic "1" **ONLY** when there are an **ODD** number of 1's on the inputs to the gate and this description can be expanded to apply to any number of individual inputs as shown below for a 3-input Ex-OR gate.

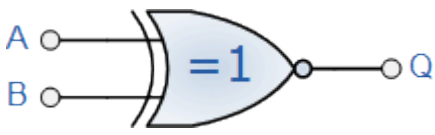
Symbol	Truth Table			
 <p>3-input Ex-OR Gate</p>	A	B	C	Q
	0	0	0	0
	0	0	1	1
	0	1	0	1
	0	1	1	0
	1	0	0	1
	1	0	1	0
	1	1	0	0
	1	1	1	1
	Boolean Expression $Q = A \oplus B \oplus C$	Read as "any ODD number of Inputs" gives Q		

Activity


Can you try and generate the truth table for a 4-input XOR gate?

5.2.8 The Exclusive-NOR Gate (XNOR)

The Exclusive-NOR Gate function or Ex-NOR for short, is a digital logic gate that is the reverse form of the Exclusive-OR function. It is a combination of the Exclusive-OR gate and the NOT gate. From the truth table it can be seen that a logic level "1" output is obtained, when the two inputs are the same, that is when $A = B = "1"$ or when $A = B = "0"$ at the same time. This type of gate gives an output "1" when its inputs are "logically equal" or "equivalent" to each other, which is why an **Exclusive-NOR** gate is sometimes called an **Equivalence Gate**. The logic symbol for an Exclusive-NOR gate is simply an Exclusive-OR gate with a circle or "inversion bubble" at its output to represent the NOT function. The symbol, truth table and Boolean expression for a 2-input XNOR Gate is shown below.

Symbol	Truth Table		
 <p>2-input Ex-NOR Gate</p>	A	B	Q
	0	0	1
	0	1	0
	1	0	0
	1	1	1
Boolean Expression $Q = \overline{A \oplus B}$	Read as if A AND B are the SAME gives Q		

Unlike the XOR gate studied in the previous section, an XNOR gate will give an output value of logic "1" ONLY when there are an **EVEN** number of 1's on the inputs to the gate except when all its inputs are "LOW", and this description can be expanded to apply to any number of individual inputs as shown below for a 3-input XNOR gate.

Symbol	Truth Table			
 <p>3-input Ex-NOR Gate</p>	A	B	C	Q
	0	0	0	1
	0	0	1	0
	0	1	0	0
	0	1	1	1
	1	0	0	0
	1	0	1	1
	1	1	0	1
	1	1	1	0
	Boolean Expression $Q = \overline{A \oplus B \oplus C}$	Read as "any EVEN number of Inputs" gives Q		

XNOR gates are used mainly in electronic circuits that perform arithmetic operations and data checking such as Adders, Subtractors Parity Checkers or Digital Comparator circuits.

5.3 Universal Logic gates

Of all the eight different logic gates that we have studied above, two of them stand out. These gates are referred to as "Universal" logic Gates because all the remaining six logic gates can be produced using only these gates. The two gates are the NAND and the NOR gates.

5.3.1 NAND gate

The **Logic NAND Gate** is usually classified as a "Universal" gate because it is one of the most commonly used logic gate types but more importantly, the NAND gates can also be used to produce any other type of logic gate function. In practice, the NAND gate forms the basis of most practical logic circuits. Figure 5.9 shows the NAND gate and how by connecting them together in various combinations the three basic gate types of AND, OR and NOT function can be formed.

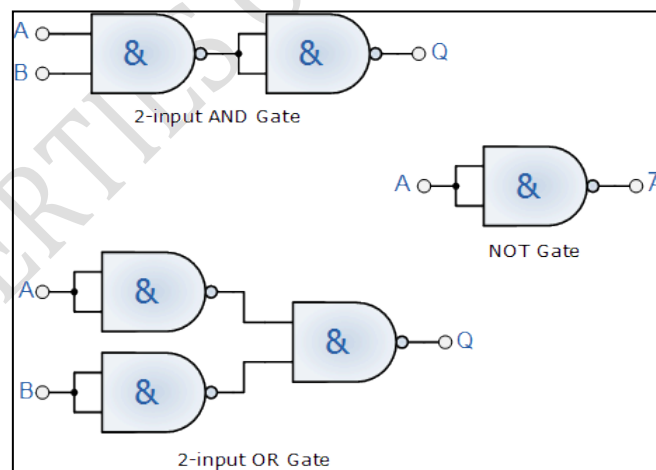
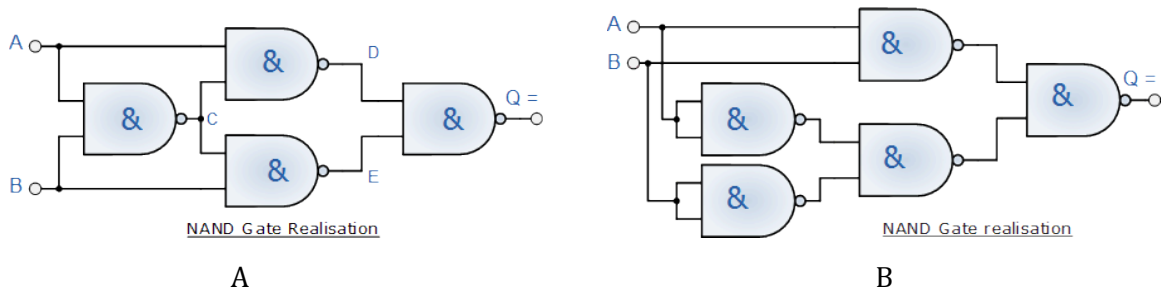


Figure 5.9: Various Logic Gates using only NAND Gates

Activity

Can you try and decipher which logic gates are implemented in A and B below using only NAND gates?



5.3.2 NOR Gate

Similar to the NAND gate seen in the last section, the NOR gate can also be classed as a "Universal" logic gate because they can also be used to produce any other type of logic gate function by connecting them together in various combinations. Figure 5.10 below shows the three basic gate types (AND, OR and NOT) implemented using only NOR gates.

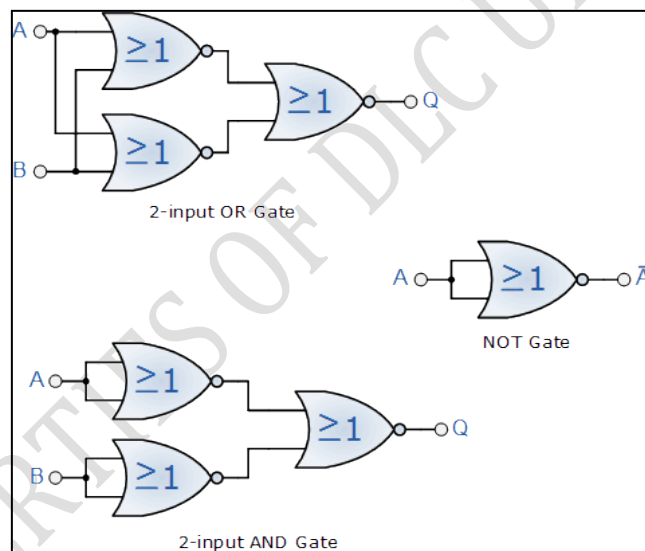


Figure 5.10: Various Logic Gates using only NOR Gates

As you have deciphered from the last activity, apart from the three common types of gate implemented above, the BUF, XOR, XNOR and standard NAND gates can also be formed using just individual NOR gates.

Summary

In this study session on Digital Logic Gates, we have seen that the main basic types of digital logic gates are the AND gate, the OR gate, the NOT gate and the XOR gate. We have also seen that each gate has an opposite or complementary form of itself in the form of the NAND gate, the NOR gate, the Buffer and XNOR gate respectively. These individual gates can be connected together to form more complex Combinational Logic circuits which we will study next. We also studied that each logic gate has a specific symbol, Boolean expression as well as truth table. In summary, the truth tables of the eight standard individual Digital Logic Gates are presented in the next table.

Truth Table Summary

Inputs		Truth Table Outputs for 2-input Logic Gates					
A	B	AND	NAND	OR	NOR	EX-OR	EX-NOR
0	0	0	1	0	1	0	1
0	1	0	1	1	0	1	0
1	0	0	1	1	0	1	0
1	1	1	0	1	0	0	1

Truth Table Output for Single-input Gates		
A	NOT	Buffer
0	1	0
1	0	1

Post-Test

1. What is the function of a NOT gate?
2. Which logic gates are referred to as Universal gates? Show how one of them can be used to implement three other logic gates.
3. Figure 5.11 is a simple 2-input logic gate using transistors as switches:
 - i. Which logic gate does it represent?
 - ii. What is the regular symbol for this logic?
 - iii. Generate the truth table for this logic gate with three inputs.

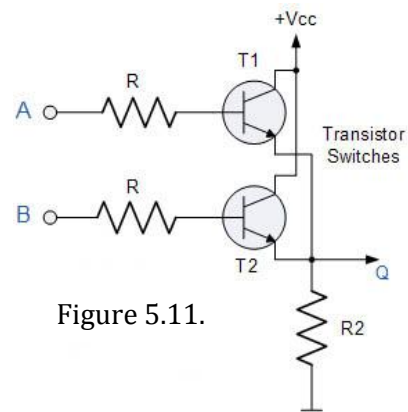


Figure 5.11.

References

Duncan, Tom. 1997, Electronics for Today and Tomorrow. Spain: John Murray.

Hwang, Enoch O. 2005. Digital Logic and Microprocessor Design With VHDL. Brooks/ Cole.

Theraja, B.L. and Theraja, A.K., 2005. A textbook of electrical technology, S. Chand.

Watson J. (1990) Logic families. In: Mastering Electronics. Macmillan Master Series. Palgrave,

London

www.electronics-tutorials.ws

PROPERTIES OF DLC UI, IBADAN

LECTURE SIX

COMBINATIONAL LOGIC DESIGN

Introduction

Having studied the different logic gates, it is pertinent that we study how they can be combined together to form different electronic circuit that would do different things. When these gates are combined to form larger circuits, those circuits are referred to as combinational logic circuits.

Objectives

At the end of this lecture, you should be able to:

1. State and implement the various ways a combination Logic circuit can be represented.
2. Draw a circuit, given its Boolean expression.
3. Generate the Boolean expression of any digital logic circuit.
4. Generate the truth table of any given circuit.

6.1 Combinational Logic Circuits (CLC)

Suppose you want to design a circuit that will send some form of alarm when someone comes close to your front window or your back window, as well as if there is some noise around your house. It means you must have a movement sensor by the two windows and then a noise sensor situated somewhere around the house. The output of these sensors which could be small voltage values (that could represent logic "1" or logic "0") can then be used as input to several logic gates to implement what you want. Let us denote the:

Front window sensor output as F
Back window sensor output as B
Noise sensor output as N

From the scenario depicted above and using plain English language, it means that you what to design a logic circuit that will give an alarm when:

Either <i>Sensor F</i> or <i>Sensor B</i> is one, and <i>Sensor N</i> is one.

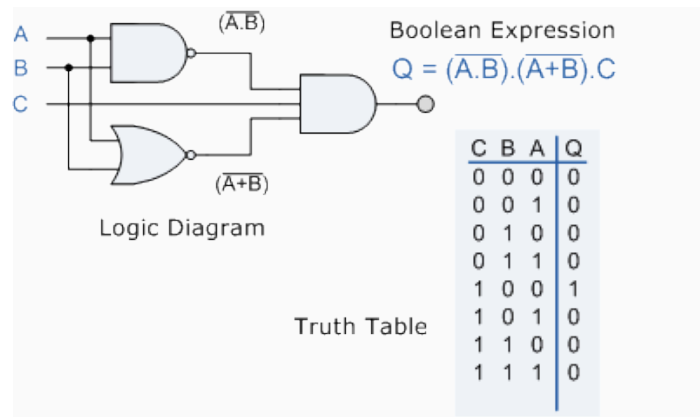
If any of the input signal changes, the output may also change. This is a simple illustration of a CLC.

Combination Logic Circuits are made up from basic logic gates that are "combined" or connected together to produce more complicated switching circuits. The logic gates we studied in the previous lecture are the building blocks of combinational logic circuits. CLC can be very simple or very complicated and any combinational circuit can be implemented with only NAND and NOR gates as these are classed as "universal" gates. The outputs of CLC are only determined by the logical function of their current input state, at any given instant in time because they have no feedback, and any changes to the signals being applied to their inputs will immediately have an effect at the output. In other words, in a CLC, the output is dependant at all times on the combination of its inputs and if one of its inputs condition changes state so does the output.

There are three main ways of specifying the function of a combinational logic circuit. These are:

1. **Truth Table:** which provides a list that shows the output values in tabular form for each possible combination of input variables.
2. **Boolean function / Expression:** this forms an output expression for each input variable that represents a logic "1"
3. **Logic Diagram:** shows the wiring and connections of each individual logic gate that implements the circuit.

An example of the three is shown below.



As combinational logic circuits are made up from individual logic gates only, they can also be considered as "decision making circuits" and combinational logic is about combining logic gates together to process two or more signals in order to produce at least one output signal according to the logical function of each logic gate. You will notice that there is a "dot" on the input line for A as well as for B. This "dot" shows that the lines are connected together. However, notice that when these two lines cross over the C input, there is no "dot", therefore no connection. We will maintain this symbol for the rest of the document. Now let us study what the truth table is briefly.

6.1.1 Truth Table

As written above, a truth table is a table that provides a list of each possible combination of input variables and the corresponding output value(s). There is a trick in generating all the possible combinations as well as the sequence. First of all, you need to know the number of possible combinations and this can be calculated using the total number of inputs. For a 2-input CLC we have 4 (2^2) possible combinations. For a 3-input CLC we have 8 (2^3) possible combinations. Therefore, for an N-input CLC we will have 2^N possible combinations.

Quiz: How many possible combinations would the truth table for your house alarm circuit?

If you said three, then you are on track! Now that we know how to calculate the total number of possible combinations, how do we generate their sequence? That, is also easy. Let us try with a-input CLC. We will use the following steps:

<p>1. Draw a table with the number of row that correspond to the total number of possible combinations and the number of columns that correspond to the total number of inputs. Note that you will need an additional row for labels and an additional column (s) for the output</p>	<table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>F</th> <th>B</th> <th>N</th> <th>alarm (output)</th> </tr> </thead> <tbody> <tr><td> </td><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td><td> </td></tr> </tbody> </table>	F	B	N	alarm (output)																																
F	B	N	alarm (output)																																		
<p>2. Starting with the leftmost input variable (N in this case), write a "0" then a "1" alternating $2^0 = 1$ time(s) in each row. Starting from 0.</p>	<table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>F</th> <th>B</th> <th>N</th> <th>alarm (output)</th> </tr> </thead> <tbody> <tr><td> </td><td> </td><td>0</td><td> </td></tr> <tr><td> </td><td> </td><td>1</td><td> </td></tr> <tr><td> </td><td> </td><td>0</td><td> </td></tr> <tr><td> </td><td> </td><td>1</td><td> </td></tr> <tr><td> </td><td> </td><td>0</td><td> </td></tr> <tr><td> </td><td> </td><td>1</td><td> </td></tr> <tr><td> </td><td> </td><td>0</td><td> </td></tr> <tr><td> </td><td> </td><td>1</td><td> </td></tr> </tbody> </table>	F	B	N	alarm (output)			0				1				0				1				0				1				0				1	
F	B	N	alarm (output)																																		
		0																																			
		1																																			
		0																																			
		1																																			
		0																																			
		1																																			
		0																																			
		1																																			
<p>3. Move to the next column to the left (input variable B) to fill the column. This time write $2^1 = 2$ zeros and ones. Alternating after every set.</p>	<table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>F</th> <th>B</th> <th>N</th> <th>alarm (output)</th> </tr> </thead> <tbody> <tr><td> </td><td>0</td><td>0</td><td> </td></tr> <tr><td> </td><td>0</td><td>1</td><td> </td></tr> <tr><td> </td><td>1</td><td>0</td><td> </td></tr> <tr><td> </td><td>1</td><td>1</td><td> </td></tr> <tr><td> </td><td>0</td><td>0</td><td> </td></tr> <tr><td> </td><td>0</td><td>1</td><td> </td></tr> <tr><td> </td><td>1</td><td>0</td><td> </td></tr> <tr><td> </td><td>1</td><td>1</td><td> </td></tr> </tbody> </table>	F	B	N	alarm (output)		0	0			0	1			1	0			1	1			0	0			0	1			1	0			1	1	
F	B	N	alarm (output)																																		
	0	0																																			
	0	1																																			
	1	0																																			
	1	1																																			
	0	0																																			
	0	1																																			
	1	0																																			
	1	1																																			
<p>4. Move to the next column to the left (input variable F) to fill the column. This time write $2^2 = 4$ zeros and ones. Alternating after every set.</p>	<table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>F</th> <th>B</th> <th>N</th> <th>alarm (output)</th> </tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td><td> </td></tr> <tr><td>0</td><td>0</td><td>1</td><td> </td></tr> <tr><td>0</td><td>1</td><td>0</td><td> </td></tr> <tr><td>0</td><td>1</td><td>1</td><td> </td></tr> <tr><td>1</td><td>0</td><td>0</td><td> </td></tr> <tr><td>1</td><td>0</td><td>1</td><td> </td></tr> <tr><td>1</td><td>1</td><td>0</td><td> </td></tr> <tr><td>1</td><td>1</td><td>1</td><td> </td></tr> </tbody> </table>	F	B	N	alarm (output)	0	0	0		0	0	1		0	1	0		0	1	1		1	0	0		1	0	1		1	1	0		1	1	1	
F	B	N	alarm (output)																																		
0	0	0																																			
0	0	1																																			
0	1	0																																			
0	1	1																																			
1	0	0																																			
1	0	1																																			
1	1	0																																			
1	1	1																																			
<p>5. If you have more columns to fill,</p>	<p>This means that for the next column, we will have a</p>																																				

then repeat step 4 above, however note that you must increase the number in each set to the left to the next power of 2.

set of $2^3 = 8$ zeros followed by a set of 8 ones. As so on!

Once you generate all the possible combinations, then the next thing to do is to specify the output which corresponds to a logic level "1" depending of the input combinations. For our house alarm system mentioned in section 6.1, the complete truth table would look like this:

Row No.	F	B	N	Signal (output)
1	0	0	0	0
2	0	0	1	0
3	0	1	0	0
4	0	1	1	1
5	1	0	0	0
6	1	0	1	1
7	1	1	0	0
8	1	1	1	1

By inspection you can see that the rows 4, 6 and 8 represent cases where either F OR B is "1" AND N is also one. There is another way to fill in the output column. We will look at that in subsequent sections.

Quiz No. 2: Generate the truth table for a CLC that has a total of 5 inputs. Ignore the output column

6.1.2 Boolean Expression

Boolean function forms an output expression for each input variable that represents a logic "1". We can generate this expression, if the description is as simple as our alarm system described in section 6.1:

This basically says (Sensor F or Sensor B) and Sensor N will be on for the alarm to ring. This translates to: $(F \text{ OR } B) \text{ AND } N$ $Alarm = (F + B) \cdot N$

Note that it is important, just like in arithmetic to place a bracket as the appropriate position, so that the logic expression is not misinterpreted. For example, from our expression above, if we had written it without any brackets, such as $Alarm = F + B \cdot N$, it could be interpreted as the alarm will ring if the F sensor gives a signal or if both the B sensor and the N sensor give their signals. This invariably means that the alarm could ring if someone comes close to your front window, despite the fact that there is no noise around your house. So be prudent when generating Boolean expressions and use your brackets well.

Alternatively, given a logic circuit diagram, we can also derive a Boolean function that describes that circuit. We simply write down the Boolean logical expressions at the output of each gate as we trace through the circuit from the primary input to the primary output. Below are some simple examples:

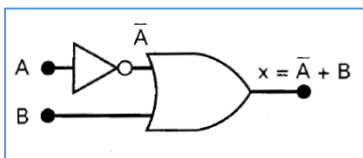


Fig 6.1

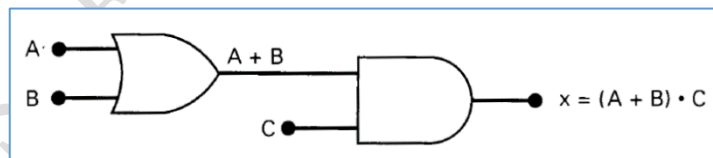


Fig 6.2

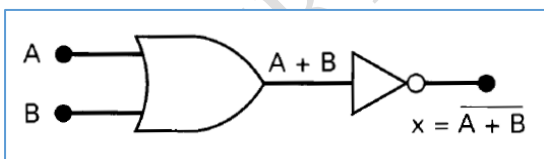


Fig 6.3

Here are more examples that are not so simple!

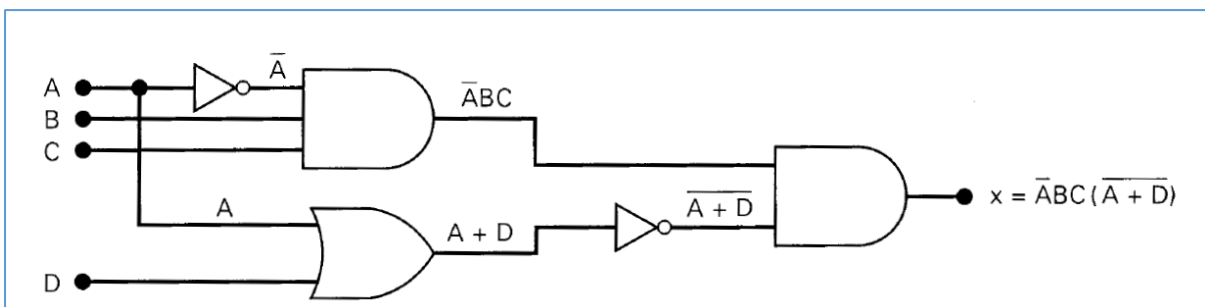


Fig 6.4

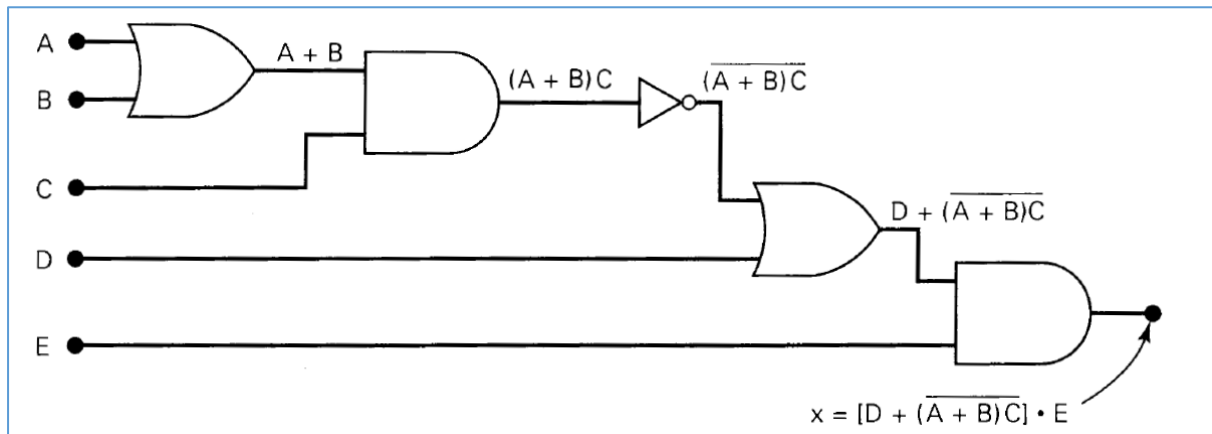


Fig 6.5

Quiz: Generate the Boolean expression for the circuit in fig 6.6

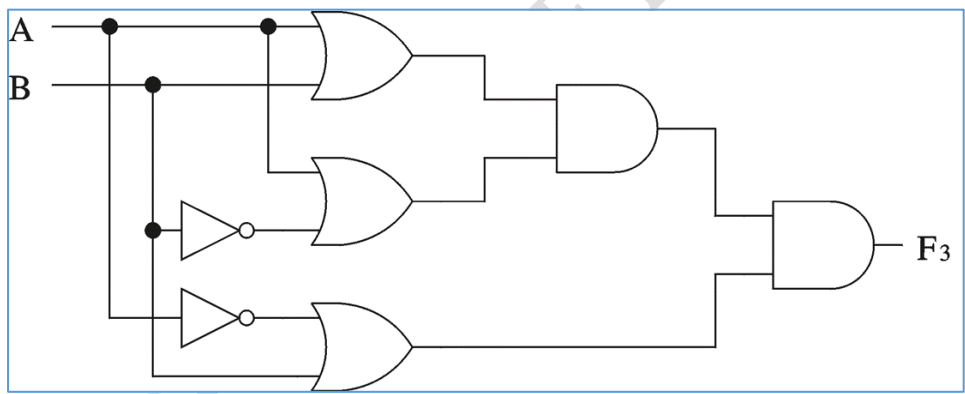


Fig 6.6

It is also possible to generate Boolean expressions, given the truth table. This will be treated in a separate lecture on Sum of Product and Product of Sum.

When dealing with Boolean expressions, it is important to note which gate take precedence over the other just like in arithmetic where we are taught BODMAS. To begin with, AND takes precedence over OR unless overridden by brackets. NOT is a special case in that it can act like a set of brackets. If the bar indicating the NOT function spans a single variable, it takes precedence

over AND and OR. If, however, the NOT bar spans an expression, the expression beneath the bar must be evaluated before the NOT is taken. Figure 5-8 presents two examples of handling precedence with the NOT function.

6.1.3 Logic Circuit diagram

The third way of representing combinational logic circuit is by using the logic diagram itself. This is a diagram that shows the wiring and connections of each individual logic gate that implements the circuit. Going back to our house alarm system in section 6.1, we can clearly identify two distinct logic gates used from the English words OR and AND. Since our alarm system would ring if either F OR B send signals as well as when N sends its own signal. The logic diagram is as shown in figure 6.7

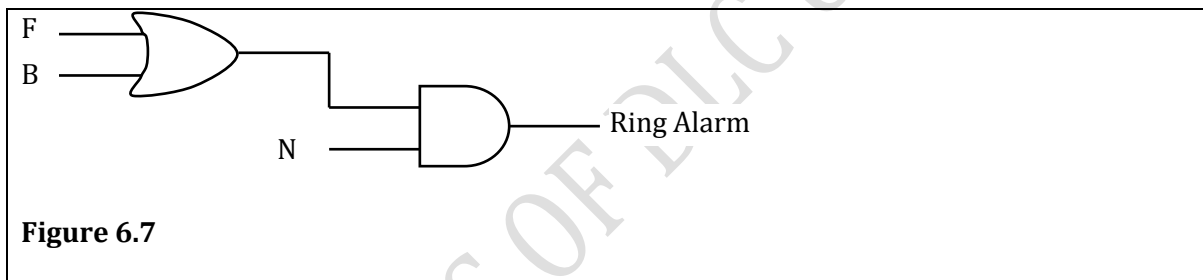


Figure 6.7

Alternatively, given a Boolean expression, you can always draw the logic circuit diagram by taking each gate, drawing it and specifying its inputs. Let us take an example:

Draw the logic circuit diagram for $F = AB + AC$

Looking at the expression we can clearly identify that it contains two AND gates and one OR gate. The A and B inputs are “ANDed” together, then the A and C inputs are also “ANDed” together before their various outputs are “ORed” together. Let us see how this is drawn, using the logic gate symbols.

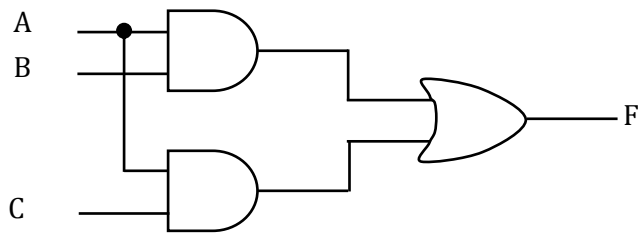
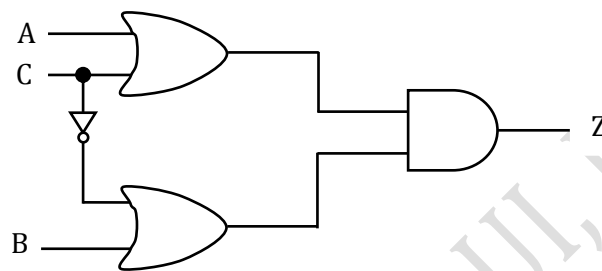


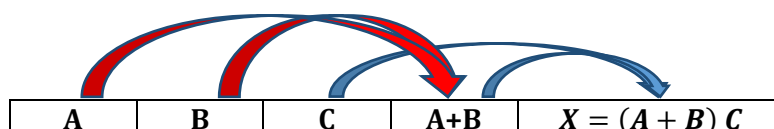
Figure 6.7

Another example: Draw the logic circuit for $Z = (A + C)(B + \bar{C})$



Quiz: Draw the logic circuit for $Z = (B + DC)(A + \bar{BC})$

By now you have seen that given a Boolean expression, you can draw the logic diagram and you have also seen that given a logic diagram, you can generate the Boolean expression. How then can you generate the truth table of a combinational logic circuit, given the logic diagram or the Boolean expression? If you have the Boolean expression, you go ahead and generate the truth table, as we will see shortly. However, if given the logic diagram, the first thing to do is to generate the Boolean expression and then from the Boolean expression, generate the truth table. Let us try that out with Fig 6.2. The Boolean expression generated was $X = (A + B) C$. So following the step on how to create a truth table, we know that this CLC has three inputs (A, B, C) and one output (X). It then means there will be 8 rows in the truth table, the number of columns here will depend on how complex the Boolean expression is. The table below shows the truth table for the Boolean expression $X = (A + B) C$.



0	0	0	0	0
0	0	1	0	0
0	1	0	1	0
0	1	1	1	1
1	0	0	1	0
1	0	1	1	1
1	1	0	1	0
1	1	1	1	1

The first three columns represent the primary inputs (A, B, C). The fourth column draws its values, as depicted by the direction of the red arrows, from the logic operation A OR B. This is a temporary output because it also contributes to the final output X. X on the other hand draws its values, as shown by the blue arrows, from the logic AND operation between C and A+B. As long as you follow this step by step method, you can generate the truth table for any Boolean expression.

Let us take another Boolean expression which is a little bit more complex than the one we just used and also has more primary inputs. We can also generate the truth table for the logic diagram in Figure 6.4. This time around, there are 4 primary inputs (A, B, C, D) and one output X. This means we will have 16 possible combinations for $X = \bar{A}BC(\bar{A} + \bar{D})$

A	B	C	D	\bar{A}	BC	$\bar{A}BC$	A + D	$\bar{A} + \bar{D}$	$X = \bar{A}BC(\bar{A} + \bar{D})$
0	0	0	0	1	0	0	0	1	0
0	0	0	1	1	0	0	1	0	0
0	0	1	0	1	0	0	0	1	0
0	0	1	1	1	0	0	1	0	0
0	1	0	0	1	0	0	0	1	0
0	1	0	1	1	0	0	1	0	0
0	1	1	0	1	1	1	0	1	1
0	1	1	1	1	1	1	1	0	0
1	0	0	0	0	0	0	1	0	0
1	0	0	1	0	0	0	1	0	0
1	0	1	0	0	0	0	1	0	0
1	0	1	1	0	0	0	1	0	0
1	1	0	0	0	0	0	1	0	0
1	1	0	1	0	0	0	1	0	0
1	1	1	0	0	1	0	1	0	0
1	1	1	1	0	1	0	1	0	0

Quiz: Generate the truth table for Figure 6.5

From the truth table above, you would notice that from what seemed to be a complex circuit, only ONE combination of the inputs gives a logic “1” output. That is the shaded row. This brings us to another small topic. Given a Boolean expression, how do you evaluate its output for a particular combination of its input?

6.2 Evaluating Logic circuit outputs

Given any Boolean expression, we can obtain the output logic level for any combination of its inputs. For example, suppose we want to know the logic level for the output $X = \bar{A}BC(\bar{A} + \bar{D})$ when $A=0$, $B=1$, $C=1$ and $D=1$. We achieve this by substituting these logic values for their placeholders as follows:

$$X = \bar{A}BC(\bar{A} + \bar{D}) = \bar{0} \cdot 1 \cdot 1 \cdot (\bar{0} + \bar{1}) = 1 \cdot 1 \cdot 1 \cdot (\bar{1}) = 1 \cdot 1 \cdot 1 \cdot 0 = 0$$

From the evaluation we see that the output will be 0. Why not try to evaluate the same expression with $A=0$, $B=1$, $C=1$ and $D=0$. If you got 1 then you are on track.

Quiz: Evaluate the Boolean expression of figure 6.5. for
A=0 B=1 C=0 D=1 E=0

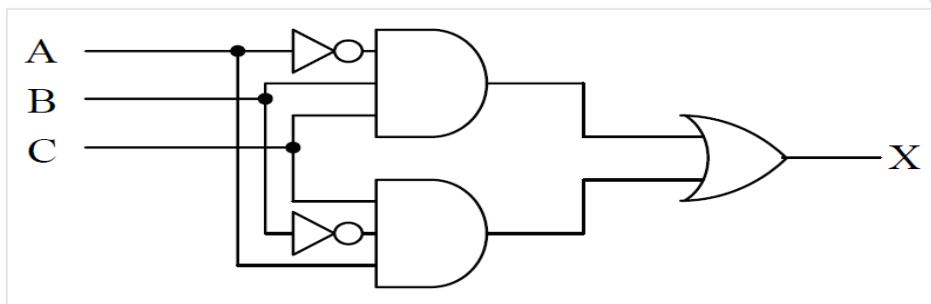
Summary

In this lecture, we learned that Combinational Logic Circuits consist of inputs, two or more basic logic gates and outputs. The logic gates are combined in such a way that the output state depends entirely on the current input states. A combinational circuit performs an operation assigned logically by a Boolean expression or truth table. The three main ways of specifying the function of a Combinational Logic Circuit are Truth Table, Boolean Expression and Circuit diagram. We also learned how to evaluate a Boolean expression for a given combination of inputs. There are some common and standard combinational logic circuits which are made up from individual logic gates to carry out desired application. These include **Multiplexers, De-**

multiplexers, Encoders, Decoders, Full Adders, Half Adders etc. They will be discussed in another lecture.

Post-Test

1. Sketch the logic diagram that represents $F = (\bar{A} + C)(A + \bar{B}C)$.
2. Given that $F = AD(B + C)$, draw its logic diagram and generate the truth table.
3. Generate the Boolean expression and truth table for the figure below.



4. Mr Felix has 5 children, 2 of which are twins. They are named Abiola, Busayo, Carol, Taiye and Kenny. In order to teach his children the value of money, he installs a safe in their reading room. This safe has two doors, one for putting money in (door IN) and the other for taking money out (door OUT). Mr felix alone can open door IN when he wants to put money in. For any of his children to take money out however, the child has to use his/her key as well as their fathers' key. But for the set of twins, both must use their keys as well as their fathers' key for any withdrawal. Using your knowledge of digital electronics
 - A. Design the logic circuit diagram that will open 'door IN'.
 - B. Design the logic circuit diagram that will enable 'door OUT' to be opened.
 - C. Generate the truth table for B above.
5. Mrs Okafor bought three cartons of biscuits named Shortbread, Digestive and Cabin for her pupils to share. There are 30 pupils in her class. She had two options for sharing the biscuits. Option A was that each pupil will take one of each type of biscuit. However, by the time the packets of biscuits in each carton were counted, she knew that Option A would not work. She decided to use Option B, which was this:- A pupil must take either a pack of Shortbread or a pack of Digestive but not both, in addition to that, he must take a packet of Cabin biscuit. Using your knowledge of digital electronics,
 - A. Design the logic circuit that represents Option A.
 - B. Generate the truth table for Option B.
 - C. Design the logic circuit that represents Option B.

References

Ronald J. Rocco and Neal S. Widmer, Digital Systems Principles and Applications, Eighth Edition. Prentice-Hall, 2001.

Watson J. (1990) Logic families. In: Mastering Electronics. Macmillan Master Series. Palgrave, London
www.electronics-tutorials.ws

PROPERTIES OF DLC UI, IBADAN

LECTURE SEVEN

STANDARD FORMS OF EXPRESSION

Introduction

In the previous lecture, we studied combinational logic circuits and the three main ways of specifying the function of a combinational logic circuit. In the design of digital systems, there are some standards that are regularly applied to combinational logic. This chapter outlines two standard representations of combinational logic circuits: Sum-of-Products and Product-of-Sums. Both of these formats represent the fastest possible digital circuitry since, aside from a possible inverter, all of the signals pass through exactly two layers of logic gates.

Objectives

At the end of this lecture, you should be able to:

1. State if a Boolean expression is in sum-of-product (SOP) term
2. State if a Boolean expression is in product-of-sum (POS) term
3. Convert a sum-of-product term to its standard or fundamental form
4. Generate the SOP or POS from truth tables.

7.1 Product Terms

A product term in a Boolean expression that makes use of the AND operator. That is a term that contains inputs that are "ANDed" together. The inputs to the AND gates are either inverted or non-inverted input variables. Some examples of product terms are:

1. AB this represents A AND B
2. $X\bar{Y}Z$ this represents X AND NOT Y AND Z

The following term is not a product term, because the bar covers two inputs, which makes the operator between the two inputs a NAND operator.

1. \overline{CD} this represents NOT C AND D

7.2 Sum of Product

A sum-of-products (SOP) expression is a Boolean expression in a specific format. The term sum-of-products comes from the expression's form: a sum (OR) of one or more products (AND) terms. As a digital circuit, an SOP expression takes the output of one or more AND gates and OR's them together to create the final output. Since the inputs to the AND gates are either inverted or non-inverted input signals, this arrangement limits the number of gates that any input signal passes through before reaching the output to an inverter, an AND gate, and an OR gate. We know from previous studies that each gate causes a delay in the transition from input to output, and since the SOP format forces all signals to go through exactly two gates (not counting the inverters), an SOP expression gives us predictable performance regardless of which input in a combinational logic circuit changes. Below is an example of an SOP expression:

1. $ABC + A\bar{B}C$
2. $AB + C\bar{D} + A\bar{B}\bar{C} + D + ABCD$
3. $\bar{X}Y + \bar{Z} + XYZ + \bar{Y}Z$

Note that each of the expressions above consist of two or more product terms that are ORed together. Each product term consists of one or more inputs *individually* appearing in either complemented or uncomplemented form. As earlier stated, one inversion sign *cannot* cover more than one input in a term and there should be not brackets. The following expression is *not* an SOP expression because it has brackets and the bar covers more than one input.

4. $(AB)\overline{CD} + A\bar{B}\bar{C} + CD$

Quiz: Which of the following expressions is in SOP form?

- a) $AB + CD + \bar{E}$
- b) $XY(C + \bar{D})$
- c) $(A + B)(C + D + \bar{E})$
- d) $PQ + \bar{RS}$
- e) $P\bar{Q} + RST + \bar{P}R$

7.3 Fundamental Sum of Product Expression

Given that $Y = \bar{A}B + \bar{B}C$, we know by inspection that Y is an SOP expression. However Y is not in the *Fundamental* or *Canonical* form, because by inspection we see that Y has three primary inputs (A, B, C) but, none of the product terms in Y has all three inputs. For an SOP expression to be in the Fundamental or Canonical form, each product MUST contain all the input variables in either complemented or un-complemented form. There are two ways of converting an SOP like Y to its Canonical form: (1) from its truth table (2) by Boolean algebra. Let us start with the truth table which is shown below. By evaluating the Boolean expression, $Y = \bar{A}B + \bar{B}C$, we know that $Y = 1$ when either of the product terms $\bar{A}B$ or $\bar{B}C$ is 1. This is due to the fact that any

Row	A	B	C	Y	Product Term
0	0	0	0	0	$\bar{A}\bar{B}\bar{C}$
1	0	0	1	1	$\bar{A}\bar{B}C$
2	0	1	0	1	$\bar{A}B\bar{C}$
3	0	1	1	1	$\bar{A}BC$
4	1	0	0	0	$A\bar{B}\bar{C}$
5	1	0	1	1	$A\bar{B}C$
6	1	1	0	0	$AB\bar{C}$
7	1	1	1	0	ABC

Boolean expression ORed with 1 gives a result of 1.

Now if (A =0) and (B= 1) are necessary for Y= 1 (i.e. the product term $\bar{A}B$ being 1) then the value of C does not matter. Therefore there are two rows of the truth table $\bar{A}B\bar{C}$ (010) and $\bar{A}BC$ (011) [rows 2 and 3] which will give an output of 1 because $\bar{A}B = 1$. Similarly the

two rows corresponding to the product terms $A\bar{B}C$ (101) and $A\bar{B}\bar{C}$ (001) [rows 5 and 1] will also give an output of 1 since for both of these $\bar{B}C = 1$ irrespective of the variable A.

The four rows giving Y= 1 for both $\bar{A}B = 1$ and $\bar{B}C = 1$ correspond to the product terms, $\bar{A}B\bar{C}$, $\bar{A}BC$, $A\bar{B}C$ and $A\bar{B}\bar{C}$ which contain all three input variables. Product terms which contain all of

the input variables are called *fundamental product terms* or sometimes *minterms* (m), because each fundamental product term specifies the input conditions which produce an output of 1 in a single row of the truth table. The minterms are sometimes referenced by the subscript that represents its row in the truth table. And most often that not, the subscript on the minterm is the decimal of the binary value represented. Now, since any Boolean expression OR'd with 1 gives a result of 1, if these fundamental product terms are OR'd together then if any one of them is 1 the result will be 1. Therefore the Boolean expression for Y in the fundamental SOP is:

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

Another way of converting an SOP expression like $Y = \bar{A}B + \bar{B}C$ to its fundamental form is by using Boolean algebra. We know from the OR logic gate that, $0 + 1 = 1 \therefore \bar{C} + C = 1$. To convert each product term, we identify which variable is missing and include it as follows:

$$\begin{aligned} Y &= \bar{A}B + \bar{B}C \\ Y &= \bar{A}B(1) + \bar{B}C(1) \\ Y &= \bar{A}B(\bar{C} + C) + (\bar{A} + A)\bar{B}C \\ Y &= \bar{A}B\bar{C} + \bar{A}BC + \bar{A}\bar{B}C + A\bar{B}C \end{aligned}$$

An alternative notation

As earlier written, the subscript on a minterm is the decimal of the binary value represented. For example $\bar{A}B\bar{C} \equiv 010$, which is the decimal no 2 so the minterm would be written as m_2 we can use the minterms to represent a Boolean expression. Alternatively, since each fundamental product term corresponds to a row in the truth table another way of describing a fundamental sum of products expression is simply to list the rows. To do this all that is required is to decide upon an appropriate code. The obvious choice is to use the decimal equivalent of the binary code held by the input variables. So for a truth table with three inputs we have row 0 ($\bar{A}\bar{B}\bar{C}$), row 1 ($\bar{A}\bar{B}C$), through to row 7 (ABC). Using this notation, the expression

$$Y = \bar{A}B + \bar{B}C = Y = \bar{A}\bar{B}C + \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}C \text{ would be written as:}$$

$$Y(A, B, C) = \sum(m_1, m_2, m_3, m_5) \text{ or simply as } Y(A, B, C) = \sum m(1, 2, 3, 5)$$

The expressions $Y(A, B, C) = \sum(m_1, m_2, m_3, m_5)$ or $Y(A, B, C) = \sum m(1, 2, 3, 5)$ are often referred to as the canonical form of expression.

Quiz: Express the function $Z = \bar{A}C + \bar{A}\bar{B}$ in its canonical form

It is possible to convert any truth table to a fundamental SOP expression and thus it's canonical form. The conversion process is as follows: identify the rows with ones as the output, and then come up with the unique product to put a one in that row. Let us take an example:

Example: Derive the SOP expression and Canonical form for the following truth table.

A	B	C	X
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Solution

First, identify each row that contains a one as the output. That is the shaded rows.

Row	A	B	C	X	Where
0	0	0	0	0	
1	0	0	1	1	A = 0, B = 0, and C = 1
2	0	1	0	0	
3	0	1	1	1	A = 0, B = 1, and C = 1
4	1	0	0	1	A = 1, B = 0, and C = 0
5	1	0	1	0	
6	1	1	0	1	A = 1, B = 1, and C = 0
7	1	1	1	0	

Next we need to make a product for each of these rows such that the output will be 1. For row1 (binary 001 or, decimal No. 1) where A=0, B=0, and C=1, the product that outputs a one must invert A and B in order to have a product of $1 \cdot 1 \cdot 1 = 1$. Therefore, our product term is:

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

Similarly we see that the product that outputs a one for the row3 (binary 011 or decimal No. 3) where A=0, B=1, and C=1 must invert A in order to have a product of $1 \cdot 1 \cdot 1 = 1$. This gives us:

$$\bar{A}BC.$$

The third product term which outputs a one for the row4 (binary 100 or decimal No. 4) where A=1, B=0, and C=0, must invert B and C in order to have a product of $1 \cdot 1 \cdot 1 = 1$. This gives us :

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

The final product outputs a one for the row6 (binary 110 or decimal No. 6) where A=1, B=1, and C=0. This time only C must be inverted, so we have :

$$AB\bar{C}$$

OR'ing all of these product terms together gives us our SOP expression and as you would notice, it is already in the fundamental SOP form.

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

To derive the canonical form, we have: $X(A, B, C) = \sum m(1, 3, 4, 6)$

Quiz: Derive the SOP expression and Canonical form for the following truth tables																																																																																																														
<table border="1" style="margin: auto;"> <thead> <tr><th>A</th><th>B</th><th>C</th><th>X</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>1</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>1</td><td>0</td></tr> </tbody> </table>	A	B	C	X	0	0	0	0	0	0	1	0	0	1	0	0	0	1	1	1	1	0	0	0	1	0	1	1	1	1	0	0	1	1	1	0	<table border="1" style="margin: auto;"> <thead> <tr><th>A</th><th>B</th><th>C</th><th>Y</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>0</td><td>1</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>1</td><td>0</td></tr> </tbody> </table>	A	B	C	Y	0	0	0	1	0	0	1	0	0	1	0	1	0	1	1	0	1	0	0	0	1	0	1	0	1	1	0	1	1	1	1	0	<table border="1" style="margin: auto;"> <thead> <tr><th>A</th><th>B</th><th>C</th><th>Z</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>1</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>1</td><td>1</td></tr> </tbody> </table>	A	B	C	Z	0	0	0	0	0	0	1	1	0	1	0	1	0	1	1	1	1	0	0	0	1	0	1	0	1	1	0	0	1	1	1	1
A	B	C	X																																																																																																											
0	0	0	0																																																																																																											
0	0	1	0																																																																																																											
0	1	0	0																																																																																																											
0	1	1	1																																																																																																											
1	0	0	0																																																																																																											
1	0	1	1																																																																																																											
1	1	0	0																																																																																																											
1	1	1	0																																																																																																											
A	B	C	Y																																																																																																											
0	0	0	1																																																																																																											
0	0	1	0																																																																																																											
0	1	0	1																																																																																																											
0	1	1	0																																																																																																											
1	0	0	0																																																																																																											
1	0	1	0																																																																																																											
1	1	0	1																																																																																																											
1	1	1	0																																																																																																											
A	B	C	Z																																																																																																											
0	0	0	0																																																																																																											
0	0	1	1																																																																																																											
0	1	0	1																																																																																																											
0	1	1	1																																																																																																											
1	0	0	0																																																																																																											
1	0	1	0																																																																																																											
1	1	0	0																																																																																																											
1	1	1	1																																																																																																											

To generate the truth table for a fundamental SOP expression, first inspect the expression to identify the total number of primary input variables. Then draw the appropriate number of rows for the truth table and fill in all the possible combinations. Next, examine each product term to determine when it is equal to a one. Where that product term is a one, a one will also be

outputted from the OR gate. An easier will be to write under each minterm, the logic value of individual input variables, taking \bar{X} as 0 and X as 1. After that, identify the rows with those sequence of combination and put one as their output. Then fill the blank ones with zeros.

Example : Generate the truth table for $Y = \bar{A}BC + \bar{A}\bar{B}C + ABC$

Solution:

By inspection we see that Y has three primary input variable (A, B, C) so the initial truth table will be:

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

Next write out the logic values for each minterm:

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

$$011 + 001 + 111$$

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

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

011 + 001 + 111 This means is that by the time you invert all the zero input variables, their minterm will give a product output of one.

Quiz: Generate the truth table for the Boolean expression $Y = \bar{A}B\bar{C} + \bar{A}\bar{B}C + A\bar{B}C$

7.4 Sum Terms

A sum term is a Boolean expression that is made up of input variables that are ORed together.

That is variables that are derived from the logic OR gate. The inputs to the OR gates are either

inverted or non-inverted input variables. The sum terms cannot have more than one variable combined in a term with an inversion bar. The following terms are sum terms:

1. $(A + B + C)$
2. $(X + \bar{Y})$
3. $(P + Q + \bar{R} + \bar{S})$

However, the following terms are not sum terms:

4. $(\overline{A + B} + C)$
5. $(X + \overline{XY})$
6. $(P + Q + \overline{R + S})$

7.5 Product of Sum

The product-of-sums (POS) format of a Boolean expression is similar to the SOP format with its two levels of logic (not counting inverters). The difference is that the outputs of multiple OR gates are combined with a single AND gate which outputs the final result. So far in this lecture we have approached all topics using positive level logic. In other words we have always considered the output in terms of the input combinations giving an output of 1. We could instead have used a negative level logic approach. The POS approach is sometimes referred to as the negative logic approach because it makes use of the output that is at logic level 0.

The expressions below adhere to the format of a POS expression.

1. $(A + B + C)(\bar{A} + C)$
2. $(X + \bar{Y})(X + Y + Z)$

Quiz: Which of the following expressions is in POS form?

- a) $AB + CD + \bar{E}$
- b) $XY(C + \bar{D})$
- c) $(A + B)(C + D + \bar{E})$
- d) $Q + \bar{RS}$
- e) $(P + \bar{Q})(R + S + T)(\bar{P} + R)$

Since the SOP expression is more used than the POS expression, we will not dwell much on the POS form. However, we will briefly study how to derive the fundamental POS expression.

7.6 Fundamental Product of Sum Expression

As you might have guessed, a Boolean expression is said to be in fundamental POS form if each of the sum terms contain all the input variable. A sum term that contains all the variables in complemented or un-complemented form is called a maxterm (M).

The following expressions are in fundamental POS forms.

$$Y = (\bar{P} + Q)(P + \bar{Q})$$

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

$$Z = (\bar{S} + T + \bar{U} + V)(\bar{S} + \bar{T} + U + V)$$

Converting a POS expression to a truth table follows a similar process as the one used to convert an SOP expression to a truth table. The difference is this: where the SOP conversion focuses on rows with a one output, the POS conversion focuses on rows with a zero output. We do this because the OR gate has an output of zero on exactly one row (See section 5.2.2) while all of the other rows have an output of one. Below are a few examples of this behaviour.

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

By AND'ing the output from these OR gates together, the final output will be zero anytime one of the OR gates outputs a zero. Looking closely at the shaded rows we notice that to have an output of zero, giving that we are using OR gate, then all the inputs MUST be '0'. For example:

$$\bar{A} + B + \bar{C} = 0 \quad \text{when } \bar{A} = 0, \quad B = 0 \text{ and } \bar{C} = 0 \text{ this means}$$

$$\text{When } A = 1, \quad B = 0 \text{ and } C = 1.$$

Quiz: can you work out the combination(s) of the input variables that would give $A + \bar{C} = 0$ for a CLC that has 3 primary input variables.

As you would have noticed, just as with SOP expressions, any truth table can be converted to a POS expression. The conversion process follows these steps: (1) identify the rows with zeros as the output (2) then come up with the unique sum to put a zero in that row, (3) the final group of sums can then be AND'ed together producing the POS expression.

Just like we have for SOP, we can also generate the canonical form for a POS expression. This is achieved by listing the row number for the rows where the function has zero outputs. For example: Derive the Canonical POS form for the truth table below.

Solution

First, identify each row that contains a zero as the output. That is the shaded rows.

Row	A	B	C	X	Where
0	0	0	0	1	
1	0	0	1	0	$A = 0, B = 0, \text{ and } \bar{C} = 0 \quad \therefore A + B + \bar{C} = 0$
2	0	1	0	0	$A = 0, \bar{B} = 0, \text{ and } C = 0 \quad \therefore A + \bar{B} + C = 0$
3	0	1	1	1	
4	1	0	0	0	$\bar{A} = 0, B = 0, \text{ and } C = 0 \quad \therefore \bar{A} + B + C = 0$
5	1	0	1	1	
6	1	1	0	1	
7	1	1	1	1	

$$X = (A + B + \bar{C})(A + \bar{B} + C)(\bar{A} + B + C) \quad \text{or}$$

$$X(A, B, C) = \Pi M(1, 2, 4)$$

Summary

At this point, we should be able to convert any truth table to a Boolean expression and finally to digital circuitry. It should also be clear that no truth table is represented by a unique circuit. In this chapter however, we have studied the two standard forms of expression of Boolean function. We in particular learned how to:

1. State if a Boolean expression in SOP or POS form
2. Convert a sum-of-product expression to its standard or fundamental form
3. Generate the SOP or POS from truth tables.

Post-Test

1. Convert the following expressions to standard sum of products:

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

b. $Y = ABC + B\bar{C}D$

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

2. Generate the truth table for the expressions in 1 above.

3. Derive the SOP expression and Canonical form for the following truth tables

A	B	C	X
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

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

A	B	C	Z
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

4. Derive the POS expression and Canonical form for the truth tables in question 2 above.

References

Ronald J. Rocco and Neal S. Widmer, Digital Systems Principles and Applications, Eighth Edition. Prentice-Hall, 2001.

Essential Electronics

David Tarnoff, Computer Organization and Design Fundamentals, 2007.

LECTURE EIGHT

SWITCHING FUNCTION MINIMIZATION

Introduction

In previous lectures, we studied how to design a digital electronic circuit. We also learned how to generate the truth table as well the Boolean expression for any logic circuit. Sometimes the derived Boolean expression may not be in the minimised form. That is to say, we could generate another Boolean expression with fewer number of gates that would perform the same function. In this chapter, we will look at three ways of minimising a Boolean expression. We would start with the first one, Boolean Algebra, which is how the expressions got its name.

Objectives

At the end of this lecture, you should be able to:

1. Apply the laws of Boolean Algebra to minimize a Boolean expression
2. State De Morgan's Theorem
3. Translate a truth table into a Karnaugh map.
4. Use Karnaugh maps to minimize a Boolean expression
5. Use Quine-McCluskey's Algorithm to minimize a Boolean expression

8.1 Boolean Algebra

The following sections show how Boolean expressions can be used to modify combinational logic in order to reduce complexity or otherwise modify its structure. In 1854, George Boole performed an investigation into the "laws of thought" which were based on a simplified version of the "group" or "set" theory, and from this Boolean or "Switching" algebra was developed. Boolean Algebra deals mainly with the theory that both logic and set operations are either "TRUE" or "FALSE" but not both at the same time. For example, $A + A = A$ and not $2A$ as it would be in normal algebra. Boolean algebra is a simple and effective way of representing the switching action of standard Logic Gates and the basic logic statements which concern us here are given by the logic gate operations of the AND, the OR and the NOT gate functions, that are used mainly in SOP and POS forms of expression. Just like algebra, a set of rules exist that when applied to Boolean expressions can dramatically simplify them. A simpler expression that produces the same output can be realized with fewer logic gates. Boolean algebra uses "Laws of

Boolean" to both reduce and simplify a Boolean expression. Boolean Algebra is therefore a system of mathematics based on logic that has its own set of rules which are used to define and reduce Boolean expressions. The variables used in Boolean Algebra only have one of two possible values, a "0" and a "1" but an expression can have an infinite number of variables all labelled individually to represent inputs to the expression. Following are the laws of Boolean algebra.

Table 8.1: Laws of Boolean algebra/ Boolean Theorem

	Name
<i>Theorem 1a</i> $X + 0 = X$ <i>Theorem 1b</i> $X \cdot 1 = X$	Identity
<i>Theorem 2a</i> $X + 1 = 1$ <i>Theorem 2b</i> $X \cdot 0 = 0$	Annulment
<i>Theorem 3a</i> $X + X = X$ <i>Theorem 3b</i> $X \cdot X = X$	Idempotent
<i>Theorem 4a</i> $\overline{(\overline{X})} = X$ <i>Theorem 4b</i> $\overline{(\overline{X})} = X$	Double Negation
<i>Theorem 5a</i> $X + \overline{X} = 1$ <i>Theorem 5b</i> $X \cdot \overline{X} = 0$	Complement
<i>Theorem 6a</i> $X + Y = Y + X$ <i>Theorem 6b</i> $XY = YX$	Commutative
<i>Theorem 7a</i> $X + (Y + Z) = (X + Y) + Z$ <i>Theorem 7b</i> $X(YZ) = (XY)Z$	Associative
<i>Theorem 8a</i> $(X + Y)(X + Y) = X + Y$ <i>Theorem 8b</i> $XY + XY = XY$	Absorptive
<i>Theorem 9a</i> $X(Y + Z) = XY + XZ$ <i>Theorem 9b</i> $(X + Y)(X + Z) = X + YZ$	Distributive
<i>Theorem 10a</i> $\overline{X + Y + Z + \dots} = \overline{X} \cdot \overline{Y} \cdot \overline{Z} \dots$ <i>Theorem 10b</i> $\overline{X \cdot Y \cdot Z \dots} = \overline{X} + \overline{Y} + \overline{Z} + \dots$	de Morgan's Theorem
<i>Theorem 11a</i> $X(\overline{X} + Y) = XY$ <i>Theorem 11b</i> $X + \overline{X}Y = X + Y$	
<i>Theorem 12a</i> $XY + YZ + \overline{X}Z = XY + \overline{X}Z$ <i>Theorem 12b</i> $(X + Y)(\overline{X} + Z) = XZ + \overline{X}Y$	
<i>Theorem 13a</i> $XY + \overline{X}Y = Y$ <i>Theorem 13b</i> $(X + Y)(\overline{X} + Y) = Y$	
<i>Theorem 14a</i> $X(X + Y) = X$ <i>Theorem 14b</i> $X + XY = X$	

Note that the laws can be used from left to right as well as from right to left. Theorems 11 to 14 can be proved with other Boolean laws or with the truth table.

Now that we have the rules, let us try some examples of reducing a Boolean expression or proving the equality of two Boolean expressions. Bear in mind that to reduce any expression, think of a way to reduce at least one redundant term at a time, using any of the rules as well as any two product terms. You would notice that it is easier to reduce product terms than to reduce sum terms.

Example 8.1: Prove that $(A + B)(A + C) = A + BC$ proving from left to right we have that

$$\begin{aligned}
 &= A(A + C) + B(A + C) && \text{Distributive law} \\
 &= AA + AC + AB + BC && \text{Distributive law} \\
 &= A(1 + C + B) + BC && \text{Distributive law} \\
 &= A(1) + BC && \text{Annulment law} \\
 &= A + BC
 \end{aligned}$$

Example 8.2: Prove that $ABC + A\bar{B}C + AB\bar{C} = A(B + C)$

$$\begin{aligned}
 &= ABC + A\bar{B}C + ABC + AB\bar{C} && \text{Absorptive law} \\
 &= AC(B + \bar{B}) + AB(C + \bar{C}) && \text{Distributive law} \\
 &= AC(1) + AB(1) && \text{Complement law} \\
 &= AC + AB && \text{identity law} \\
 &= A(C + B) && \text{Distributive law} \\
 &= A(B + C) && \text{Commutative law}
 \end{aligned}$$

Example 8.3: Simplify $\bar{B}(A + \bar{A}B)$

$$\begin{aligned}
 &= \bar{B}A + \bar{B}\bar{A}B && \text{Distributive law} \\
 &= \bar{B}A + \bar{A}\bar{B}B && \text{Associative law} \\
 &= \bar{B}A + \bar{A} \cdot 0 && \text{Complement law} \\
 &= \bar{B}A + 0 && \text{Annulment law}
 \end{aligned}$$

$$= A\bar{B}$$

Associative law

Example 8.4: Prove that $(X + Y)(X + \bar{Y})(\bar{X} + Z) = XZ$

$$\begin{aligned} &= (XX + X\bar{Y} + XY + Y\bar{Y})(\bar{X} + Z) \\ &= XX\bar{X} + X\bar{X}\bar{Y} + X\bar{X}Y + \bar{X}Y\bar{Y} + XXZ + X\bar{Y}Z + XYZ + Y\bar{Y}Z \\ &= 0 + 0 + 0 + 0 + XZ + X\bar{Y}Z + XYZ + 0 \\ &= XZ + X\bar{Y}Z + XYZ \\ &= XZ(1 + \bar{Y} + Y) \\ &= XZ \end{aligned}$$

You will notice that by the time you get the hang of it, using Boolean Algebra to minimize Boolean expressions will be easy. So if given a truth table, you generate the SOP expression and then minimize that expression. Now try these:

Quiz: Prove the following equalities

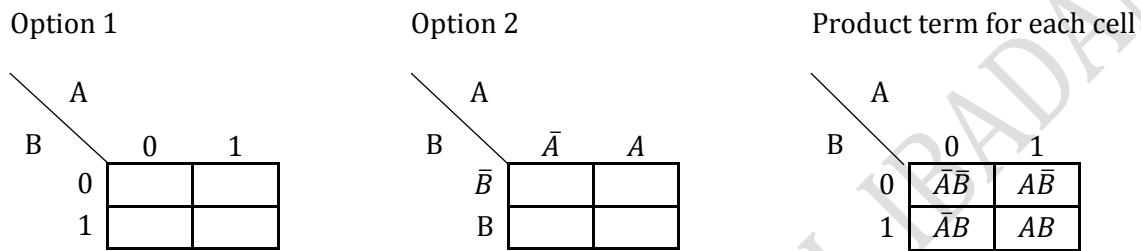
1. $A + B + C = (A + B)(\bar{A} + B) + (C + \bar{D})(C + D) + (A + E)(A + \bar{E})$.
2. $B + D = ABC + \bar{A}BC + B\bar{C} + AD + \bar{A}D$.
3. $B\bar{C} + BD = (A + B)(\bar{C} + D)(\bar{A} + B)$.

8.2 Karnaugh Maps (K-Maps)

Another useful minimization aid is the k-map. This map can be easily generated from the truth table of any digital circuit. K-maps can be used to minimise circuits with multiple inputs, ranging from 3 to 6 but for the purpose of this class, we will limit our discussion to 3 and 4 input circuits. Recall that in the SOP form of a Boolean expression, each row with an output of one corresponded to a product. The OR of all of the products produced an expression that satisfied the truth table, but not necessarily one that was reduced to its simplest form. Karnaugh Maps are graphical representations of truth tables. They consist of a grid with one cell for each row of

the truth table. The grid shown below in Figure 8.1 is the two-by-two Karnaugh map used to represent a truth table with two input variables. A truth table for two input variables has two input columns, one output column and four rows (one for each possible combination of inputs). The corresponding Karnaugh map has four cells arranged as a square, with the two inputs, A and B, used to label the columns and rows as shown.

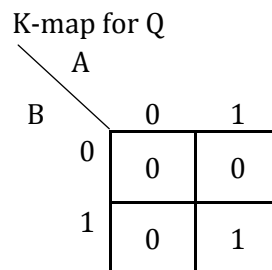
Table 8.2



Above, I have shown two options that could be used in labelling the k-map, I prefer option 1. The third one, shows the product term for each cell. Therefore, to translate a truth table into a k-Map, all you need do is place a '1' in the cell that corresponds to the row in the truth table with a 1 as its output. For example below is a truth table and its corresponding k-Map.

Example 8.5

Truth Table		
A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1



Quiz: Generate the truth table for $R = A\bar{B} + \bar{A}B$. Then draw the K-Map for R.

The purpose of Karnaugh maps is to rearrange truth tables so that adjacent cells can be represented with a single product term and simplified using **Theorem 5a** in the Boolean laws described above. This requires adjacent cells to differ by exactly one of their input values thereby identifying the input that will drop out, when four rows or columns are needed as with a 3- or 4-input Karnaugh map. This requires the rows/columns to be numbered 00-01-11-10 in

order to have only one input change from row to row OR column to column. Example 8.6 is that of a 3-input logic circuit.

Example 8.6: Draw the K-Map for the Truth table below

Truth Table			
A	B	C	Z
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Table 8.3: Option 1 arrangement of 3-input circuit

		AB			
		00	01	11	10
C	0	$\bar{A}\bar{B}\bar{C}$	$\bar{A}B\bar{C}$	$A\bar{B}\bar{C}$	$A\bar{B}C$
	1	$\bar{A}\bar{B}C$	$\bar{A}BC$	ABC	$AB\bar{C}$

K-Map for Z

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

Quiz: Draw the K-Map for these Truth tables

Table 8.4

Truth Table			
A	B	C	X
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Table 8.5

Truth Table			
A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Table 8.6: The Layout for a 4-input logic function.

		AB			
		00	01	11	10
CD	00	$\bar{A}\bar{B}\bar{C}\bar{D}$	$\bar{A}B\bar{C}\bar{D}$	$A\bar{B}\bar{C}\bar{D}$	$AB\bar{C}\bar{D}$
	01	$\bar{A}\bar{B}C\bar{D}$	$\bar{A}BC\bar{D}$	$A\bar{B}C\bar{D}$	$ABC\bar{D}$
	11	$\bar{A}\bar{B}CD$	$\bar{A}BCD$	$A\bar{B}CD$	$ABCD$
	10	$\bar{A}BC\bar{D}$	$\bar{A}BCD$	$ABC\bar{D}$	$ABC\bar{D}$

Quiz: draw the K-Map for $S = A\bar{B}CD + \bar{A}BCD + A\bar{B}C\bar{D} + \bar{A}BC\bar{D} + \bar{A}BCD + \bar{A}B\bar{C}D$

Hint: just put a '1' in the cell that corresponds to the fundamental product term in S .

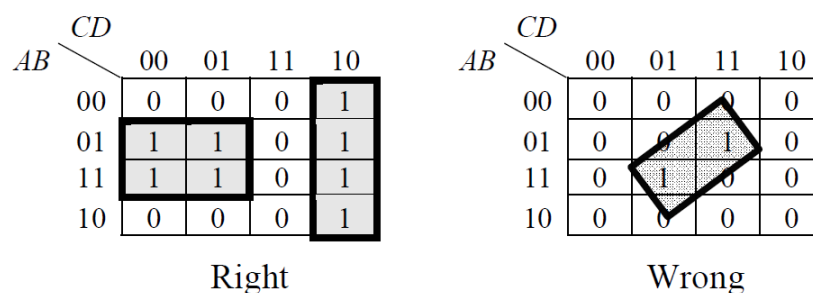
Now that you have understood how to draw the K-Maps for 2 to 4 input logic functions, we will study how to use it for minimisation.

The key to the effectiveness of a Karnaugh map is that each cell represents the output for a specific pattern of ones and zeros at the input, and that to move to an adjacent cell, one and only one of those inputs can change. Tables 8.2, 8.3 and 8.6 show the notation used for drawing Karnaugh maps with two, three and four variables, respectively. Note that each cell represents a Boolean product just as a row in a truth table does. This shows that an SOP expression can be derived from a Karnaugh map just as it would be from a truth table. Below is the SOP for Z in example 8.6

$$Z = \bar{A}BC + ABC + A\bar{B}C$$

So the key to effectively using Karnaugh maps is to find the largest group of adjacent cells containing ones. The larger the group, the fewer products and inputs will be needed to create the Boolean expression that produces the truth table. In order for a group of cells containing ones to be considered adjacent, they must follow some rules, some of which are listed below.

1. The grouping must be in the shape of a rectangle. There are no diagonal adjacencies allowed.



2. All cells in a rectangle must contain ones. No zeros are allowed.

		CD			
		00	01	11	10
AB	00	1	0	0	0
	01	1	0	1	1
	11	1	0	1	1
	10	1	0	0	0

Right

		CD			
		00	01	11	10
AB	00	0	0	0	0
	01	0	1	1	0
	11	0	1	0	0
	10	0	0	0	0

Wrong

3. The number of cells in the grouping must equal a power of two, i.e., only groups of 1, 2, 4, 8, or 16 are allowed.

		CD			
		00	01	11	10
AB	00	1	0	1	1
	01	0	0	1	1
	11	0	1	1	1
	10	0	0	1	1

Right

		CD			
		00	01	11	10
AB	00	1	1	0	0
	01	1	1	0	0
	11	1	1	0	0
	10	0	0	0	0

Wrong

4. Outside edges of Karnaugh maps are considered adjacent, so rectangles may wrap from left to right or from top to bottom.

		CD			
		00	01	11	10
AB	00	0	1	1	0
	01	0	0	0	0
	11	1	0	0	1
	10	0	1	1	0

5. Cells may be contained in more than one rectangle, but every rectangle must have at least one cell unique to it. (In wrong example, the horizontal rectangle is an unnecessary duplicate.)

		CD			
		00	01	11	10
AB	00	0	1	1	0
	01	1	1	1	1
	11	1	1	1	1
	10	0	0	0	1

Right

		CD			
		00	01	11	10
AB	00	0	0	0	0
	01	0	1	0	0
	11	0	1	1	0
	10	0	0	1	0

Wrong

6. Every rectangle must be as large as possible

		CD			
	AB	00	01	11	10
00		1	1	0	0
01		1	1	0	0
11		1	1	0	0
10		1	1	0	0

Right

		CD			
	AB	00	01	11	10
00		1	1	0	0
01		1	1	0	0
11		1	1	0	0
10		1	1	0	0

Wrong

7. Every 1 must be covered by at least one rectangle

		CD			
	AB	00	01	11	10
00		1	0	0	1
01		1	1	1	1
11		1	1	1	1
10		1	0	0	0

Right

		CD			
	AB	00	01	11	10
00		1	0	0	1
01		1	1	1	1
11		1	1	1	1
10		1	0	0	0

Wrong

Example 8.7: Minimize Z with K-Map

Truth Table			
A	B	C	Z
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Solution:

K-Map for Z

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

Looking at the blue group, we see that the only variable that changed is A, which changed from 0 to 1, therefore it is removed from the minimised term, so the blue group minimizes to BC

In the red group on the other hand, the only variable that changed within the grouping is B, which changed from 1 to 0, so it is omitted from the reduced term for that group. The red group is minimised to AC. The minimised expression for $Z = BC + AC$

Example 8.8: Minimize

$$Q = ABC + A\bar{B}C + AB\bar{C}$$

Truth Table			
A	B	C	Q
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

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

Green group resolves to AB Red group resolves to AC

$$\therefore Q = AB + AC$$

You can check this with example 8.2

For the next examples we will look at 4 input variable circuits.

Example 8.9: Use K-Map to prove that:

$$1) \quad Y = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}D + \bar{A}\bar{B}C\bar{D} + \bar{A}\bar{B}CD + \bar{A}BC\bar{D} + ABC\bar{D} = \bar{B}\bar{C} + BC\bar{D}$$

$$2) \quad X = \bar{A}\bar{B}\bar{C}D + \bar{A}B\bar{C}D + \bar{A}BCD + ABCD + A\bar{B}C\bar{D} = BD + A\bar{B}C\bar{D}$$

Solution (1) : K-Map for Y

		AB			
		00	01	11	10
CD	00	1	0	0	1
	01	1	0	0	1
	11	0	0	0	0
	10	0	1	1	0

In the blue group, observe that the variables A and D changed values for 0 to 1, while B and C remained in their complemented state. Therefore, it resolves to $\bar{B}\bar{C}$. Within the red group, only A changed, so that group resolves to $BC\bar{D}$

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

Solution for 2) : K-Map for $X = \bar{A}\bar{B}\bar{C}D + \bar{A}B\bar{C}D + \bar{A}BCD + ABCD + A\bar{B}C\bar{D} = BD + A\bar{B}C\bar{D}$

		AB			
		00	01	11	10
CD	00				
	01				
	11				
	10				

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

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

Quiz: use K-Map to minimize

$$Z = \bar{A}\bar{B}\bar{C}D + A\bar{B}\bar{C}D + \bar{A}BCD + ABCD + \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}C\bar{D} + \bar{A}\bar{B}C\bar{D} + \bar{A}\bar{B}\bar{C}\bar{D} + A\bar{B}\bar{C}D$$

Solution: $Z = \bar{A}\bar{B} + CD + B\bar{C}D$

8.3 Quine-McCluskey Method

The Quine-McCluskey (QM) method is a systematic approach to obtaining minimal expressions when a large number of variables are involved and is of considerable use in multifunctional problems. It is an approach that can be programmed in a computer and is thus of great interest in special large problems.

Essentially the QM method consists of repeated applications of the theorem $XY + \bar{X}Y = Y$. For example, the expression $ABC + \bar{A}BC$ can be reduced to BC by recognizing that A and \bar{A} are the only differences between the two terms. In a like manner the expression $\bar{A}\bar{B}\bar{C}D + \bar{A}\bar{B}C\bar{D}$ can be reduced to $\bar{A}\bar{C}D$ by recognizing that B is redundant. This is the essence of the technique. Just as the Karnaugh map has only one-variable change between adjacent cells, the QM method systematically searches for a one-variable difference between terms.

The first step in applying the QM method is to expand the function into a standard SOP form of canonical terms. Actually, rather than expanding the function in a standard sum form, a listing is made of all the canonical terms in a table.

Example 8.10 $f = \bar{A}\bar{B}\bar{C} + \bar{A}BD + A\bar{B}\bar{C} + A\bar{B}D + \bar{A}BC$

Assume the weight assignments used are $A = 8, B = 4, C = 2, D = 1$; This just means that the A input variable occupies the most significant position, followed by B then C and finally D. This means that Table 8.7 will contain all the canonical terms for the above function in algebra, binary, and decimal. The reason for this will become obvious as the examples are explained. The next step is to compare each canonical term with every other canonical term, looking for a one-variable difference. That is, the first term, $\bar{A}\bar{B}\bar{C}D$, compares with the second term, $\bar{A}\bar{B}C\bar{D}$, with the D variable being redundant. Because these two terms will reduce to $\bar{A}\bar{B}\bar{C}$, a three variable term, this is listed in a new table, Table 8.8, and a check mark is placed alongside each of the two terms to show that they have been used in a reduction. Notice that the binary representation differed only in one variable, and this is identified in Table 8.8 by placing a dash in this position. Furthermore, this comparison was between a decimal 5 and a decimal 4; this is a difference of a decimal 1, which is a binary multiple ($2^0 = 1$) as well as the weight for D.

The next term that compares with $\bar{A}\bar{B}\bar{C}D$ is the third term $\bar{A}B\bar{C}D$. This produces the second term in Table 8.8, $\bar{A}BD$. A check mark is placed beside the $\bar{A}B\bar{C}D$ term in Table 8.7, and the reduced term is listed in Table 8.8. This time C is the redundant variable and a dash is placed in the position C normally occupies. Furthermore, the decimal variable between these terms is 2, a power of 2 ($2^1 = 2$), which is the weight of the variable C . No further comparisons can be made with the first term.

Table 8.7

Canonical terms	Binary	Decimal
$\bar{A}\bar{B}\bar{C}D\checkmark$	0 1 0 1	(5)
$\bar{A}\bar{B}C\bar{D}\checkmark$	0 1 0 0	(4)
$\bar{A}B\bar{C}D\checkmark$	0 1 1 1	(7)
$\bar{A}B\bar{C}\bar{D}\checkmark$	1 0 0 1	(9)
$\bar{A}B\bar{C}D\checkmark$	1 0 0 0	(8)
$\bar{A}B\bar{C}D\checkmark$	1 0 1 1	(11)
$\bar{A}B\bar{C}\bar{D}\checkmark$	0 1 1 0	(6)

Table 8.8

Algebra	Binary	Decimal
$\bar{A}\bar{B}\bar{C}\checkmark$	0 1 0 –	(4,5)
$\bar{A}BD\checkmark$	0 1 – 1	(5,7)
$\bar{A}B\bar{D}\checkmark$	0 1 – 0	(4,6)
$\bar{A}B\bar{C}\checkmark$	0 1 1 –	(6,7)
$\bar{A}\bar{B}D^*$	1 0 – 1	(9,11)
$\bar{A}\bar{B}\bar{C}^*$	1 0 0 –	(8,9)

The second term is now compared with each term below it, and a check mark is placed beside each term that compares according to the rule. Only one term develops, $\bar{A}B\bar{D}$. The process continues with the third term in Table 8.7. The third term, $\bar{A}BCD$, compares with the last term, $\bar{A}B\bar{C}\bar{D}$, to produce $\bar{A}BC$. The process continues until all the possible comparisons have been made. In this example Table 8.7 has been completely checked off and will be of no further use. The comparisons process now continues with Table 8.8. Each term in Table 8.8 is compared with every other term of Table 8.8 in an effort to eliminate redundant variables. The first term, $\bar{A}B\bar{C}$, compares with $\bar{A}BC$ to eliminate the C variable. Table 8.9 now lists these comparisons. The binary form now contains two dashes, one for the C variable and one for the D variable. In addition, the decimal part contains the number represented by the binary form with all possible combinations.

Table 8.9		
Algebra	Binary	Decimal
$\bar{A}B$	0 1 – –	(4,5,6,7)

Thus $\bar{A}B$ – – has 0 for A and 1 for B , giving the decimal 4. The two dashes represent a possible magnitude of 3; therefore, the decimal part shows that the term $\bar{A}B$ represents and contains the decimal numbers 4, 5, 6, and 7.

The next term in Table 8.8, $\bar{A}BD$, compares with $\bar{A}B\bar{D}$ to eliminate \bar{D} . This results in the term $\bar{A}B$. But $\bar{A}B$ is already represented, and so it is not necessary to list it twice. This process repeats just as before until all possible comparisons have been made. In this case no further comparisons can be made. Notice that the last two terms, $A\bar{B}D$ and $A\bar{B}\bar{C}$ have an asterisks placed beside them because it was not possible to eliminate any variable with any of them by comparing them with other product terms in Table 8.8. Then, looking at table 8.9, there can be no further comparisons because the table has just one product term. Therefore, the reduced expression is:

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

Summary

In this lecture we have learned:

1. The three ways of reducing an SOP expression and these are Boolean Algebra, K-Maps and Quine McCluskey.
2. We studied the laws of Boolean Algebra and applied them to minimize a Boolean expression.
3. We also learned how to translate a truth table into a K-map and further use K-maps to minimize a Boolean expression with 2 to 4 input variables.
4. How to use the Quine-McCluskey's Algorithm to minimize a Boolean expression with more than 4 input variables.

Post-Test

1. Using Boolean Algebra, prove that:

- i. $AB(C + D) = (C + D)(\bar{A} + B)(A + \bar{B})(A + B)$
- ii. $ADE + (\bar{A} + \bar{D})G = (AD + G)(\bar{A} + \bar{D} + E)$
- iii. $AC(B + \bar{D}) + AC(\bar{B} + E) = AC$
- iv. $AC + \bar{A}\bar{D} = (\bar{A} + C)(A + \bar{D})$
- v. $\bar{B}C + \bar{A}CD = \bar{B}CD + \bar{B}C\bar{D} + \bar{A}CD$
- vi. $A\bar{B} + \bar{A}B = \overline{AB + \bar{A}\bar{B}}$
- vii. $B\bar{C}D + \bar{A}\bar{B}\bar{C} + AB\bar{D} + \bar{A}\bar{C} = (B + \bar{C})(A + \bar{C})(\bar{A} + \bar{C} + \bar{D})(\bar{A} + B)$

2. Minimize the following functions using either K-map or QM

- i. $f = \sum(5, 6, 8, 9, 12, 13, 14)$
- ii. $f = \sum(0, 4, 5, 6, 7, 10, 11)$

3. Write the minimal expression for the following maps

<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 10%;"></td> <td colspan="4" style="text-align: center;">AB</td> </tr> <tr> <td style="width: 10%;"></td> <td style="width: 15%;"></td> <td style="width: 15%;"></td> <td style="width: 15%;"></td> <td style="width: 15%;"></td> </tr> <tr> <td style="text-align: center;">CD</td> <td style="text-align: center;">00</td> <td style="text-align: center;">01</td> <td style="text-align: center;">11</td> <td style="text-align: center;">10</td> </tr> <tr> <td style="text-align: center;">00</td> <td style="text-align: center;">0</td> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> </tr> </table>		AB									CD	00	01	11	10	00	0	0	1	0	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 10%;"></td> <td colspan="4" style="text-align: center;">AB</td> </tr> <tr> <td style="width: 10%;"></td> <td style="width: 15%;"></td> <td style="width: 15%;"></td> <td style="width: 15%;"></td> <td style="width: 15%;"></td> </tr> <tr> <td style="text-align: center;">CD</td> <td style="text-align: center;">00</td> <td style="text-align: center;">01</td> <td style="text-align: center;">11</td> <td style="text-align: center;">10</td> </tr> <tr> <td style="text-align: center;">00</td> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> </tr> </table>		AB									CD	00	01	11	10	00	0	1	1	0
	AB																																								
CD	00	01	11	10																																					
00	0	0	1	0																																					
	AB																																								
CD	00	01	11	10																																					
00	0	1	1	0																																					

01	0	1	1	1		01	0	1	0	0
11	0	0	0	1		11	0	1	0	0
10	0	0	0	1		10	0	1	1	0

References

Ronald J. Rocco and Neal S. Widmer, Digital Systems Principles and Applications, Eighth Edition. Prentice-Hall, 2001.

David Tarnoff, Computer Organization and Design Fundamentals, 2007.

William E. Wickes, 1968, Logic Design with Integrated Circuits

PROPERTIES OF DLC UI, IBADAN

LECTURE NINE

STANDARD COMBINATIONAL LOGIC CIRCUITS

Introduction

From a previous lecture, we know that Combinational Logic Circuits consist of inputs, two or more basic logic gates and outputs. The logic gates are combined in such a way that the output state depends entirely on the input states. Combinational circuits have "no memory", "timing" or "feedback loops", their operation is instantaneous. A combinational circuit performs an operation assigned logically by a Boolean expression or truth table. Examples of common combinational logic circuits include: half adders, full adders, multiplexers, de-multiplexers, encoders and decoders all of which we will look at in this lecture.

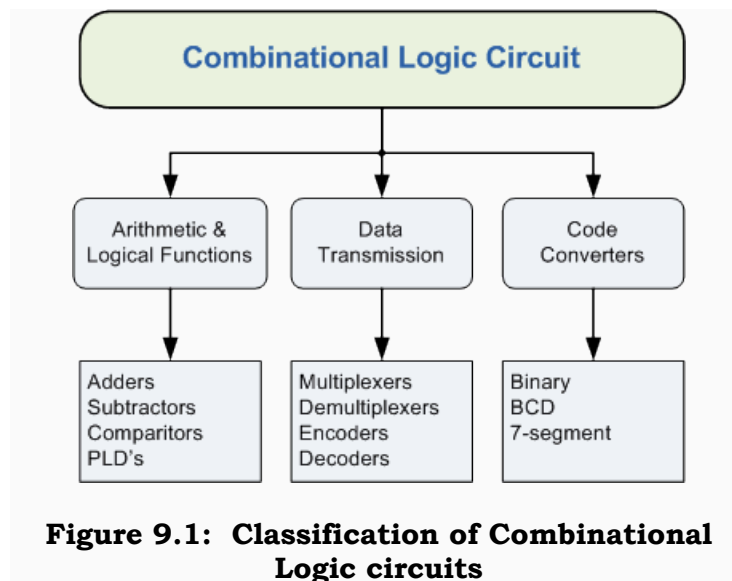
Objectives

At the end of this lecture, you should be able to:

1. Classify combinational logic circuits
2. Describe the operations and functions of arithmetic and logic circuits
3. Describe the operations and functions data transmission circuits
4. State the operations and functions of code converters

9.1 Classification of Combinational Logic

The outputs of **Combinational Logic Circuits** are only determined by the logical function of their current input state, logic "0" or logic "1", at any given instant in time as they have no feedback, and any changes to the signals being applied to their inputs will immediately have an effect at the output. In other words, in a **Combinational Logic Circuit**, the output is dependent at all times on the combination of its inputs and if one of its inputs condition changes state so does the output. There are some standard CLC that are used in building digital systems. These CLC can be classified by the type of function they perform. Figure 9.1 shows this classification as well as the circuits that belong there.



9.2 Combinational Logic Circuits for Arithmetic and Logical functions

These are circuits that perform arithmetic and logical functions. They form part of the circuits of the ALU of a computer or digital devices that support such functions. The standard circuits include the Adders, Subtractors, Comparators and PLDs. For the purpose of lecture, we will treat Adders, Subtractors and Comparators

9.2.1 The Binary Adder

The binary adder circuit is a very useful combinational logic circuit constructed using just a few basic logic gates and adds together binary numbers. This circuit allow a device to "add" together single bit binary numbers, A and B to produce two outputs, the *SUM* of the addition and a *CARRY* called the Carry-out (C_{out}) bit. This circuit is used mainly in arithmetic and counting circuits. **Binary Addition** follows the same basic rules as the decimal addition we were taught in primary school, where addition starts from the least significant bit and progresses leftwards until all the digits are expended. However, in binary addition, there are only two digits and the largest digit is "1", so any "SUM" greater than 1 will result in a "CARRY". This carry 1 is passed over to the next column for addition and so on. Consider the single bit addition below.

0	0	1	1
<u>+0</u>	<u>+1</u>	<u>+0</u>	<u>+1</u>
0	1	1	10

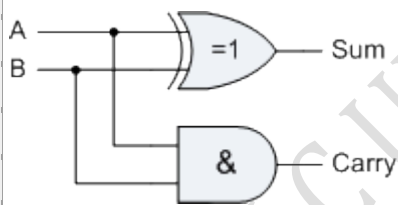
For a simple 1-bit addition problem shown above, if we ignore the resulting carry bit (in red), the result for the sum resembles the output of an Ex OR gate. We know that an Ex-OR gate will only produce a "1" output when "EITHER" input is at logic "1", so we need an additional output to produce a carry output, "1" when "BOTH" inputs "A" and "B" are at logic "1" and a standard AND gate fits the function. Combining the Ex-OR gate with the AND gate results in a simple digital binary adder circuit known commonly as the "**Half Adder**" circuit.

The Half Adder Circuit

From the truth table in Table 9.1, we can see that the SUM output is the result of the Ex-OR gate and the Carry output is the result of the AND gate. One major disadvantage of the Half Adder circuit when

Circuit	Truth Table			
	Input		Output	
A	B	SUM	CARRY	
0	0	0	0	
0	1	1	0	
1	0	1	0	
1	1	0	1	

Boolean Expression: $Sum = A \oplus B$		$Carry = A \cdot B$
--	--	---------------------



used as a binary adder, is that there is no provision for a "Carry-in" from the previous circuit (column) when adding together multiple data bits. The most complicated operation the half adder can do is "1 + 1" but as the half adder has no carry input the resultant added value would be incorrect. One simple way to overcome this problem is to use a **Full Adder** type binary adder circuit.

The Full Adder Circuit

The main difference between the **Full Adder** and the previous seen **Half Adder** is that a full adder has three inputs, the same two single bit binary inputs A and B as before plus an additional *Carry-In* (C-in) input as shown in Table 9.2. The 1-bit **Full Adder** circuit in Table 9.2 is basically two half adders connected together. The truth table for the full adder includes an additional column to take into account the Carry-in input as well as the summed output and carry-output. These full adder circuits are available as standard Integrated Circuit packages in

the form of the TTL 74LS83 which can add together two 4-bit binary numbers and generate a SUM and a CARRY output.

Table 9.2: 1-bit Binary Full Adder with Carry-Out							
Circuit			Truth Table				
			Input		Output		
			C-in	A	B	Sum	C-out
			0	0	0	0	0
			0	0	1	1	0
			0	1	0	1	0
			0	1	1	0	1
			1	0	0	1	0
			1	0	1	0	1
			1	1	0	0	1
			1	1	1	1	1

Boolean Expression: $Sum = A \oplus B \oplus C$ $Carry = (A\bar{B} + \bar{A}B)C_{in} + AB$

However, if we wanted to add together two n-bit numbers, then n 1-bit full adders need to be connected together to produce what is known as the **Ripple Carry Adder**.

Quiz: Generate the Boolean expression for the Full Adder in SOP form.

The 4-bit Binary **Ripple Carry** Adder

The **Ripple Carry Binary Adder** is simply n, full adders cascaded together with each full adder representing a single weighted column in the long addition. The carry signals produce a "ripple" effect through the binary adder from right to left. For example, suppose we want to "add" together two 4-bit numbers, the two outputs of the first full adder will provide the first place digit sum of the addition plus a carry-out bit that acts as the carry-in digit of the next binary adder. The second binary adder in the chain also produces a summed output (the 2nd

bit) plus another carry-out bit and we can keep adding more full adders to the combination to add larger numbers, linking the carry bit output from the first full binary adder to the next full adder, and so forth. An example of a 4-bit adder is given in Figure 9.2. One main disadvantage of "cascading" together 1-bit **binary adders** to add large binary numbers is that if inputs A and B change, the sum at its output will not be valid until any carry-input has "rippled" through every full adder in the chain.

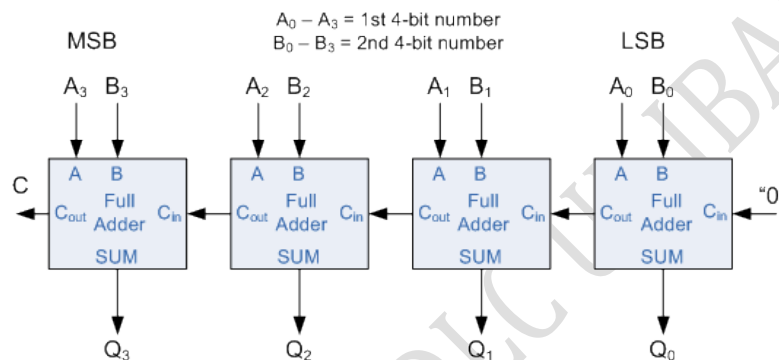


Figure 9.2: A 4-bit Binary Adder

Consequently, there will be a finite delay before the output of an adder responds to a change in its inputs resulting in the accumulated delay especially in large multi-bit binary adders becoming prohibitively large. This delay is called **Propagation delay**. One solution is to generate the carry-input signals directly from the A and B inputs rather than using the ripple arrangement above. This then produces another type of binary adder circuit called a **Carry Look Ahead Binary Adder** where the speed of the parallel adder can be greatly improved using carry-look ahead logic.

9.2.2 The 4-bit Binary Subtractor

When you write a line of code and state $X = A - B$, how would the computer perform

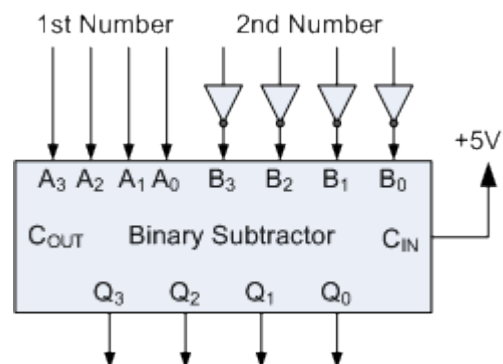


Figure 9.3: 4-bit Binary

that operation? Well the computer will first convert the variables to binary before any further operation. Assuming the two numbers are converted to 4-bits, we can perform the two 4-bit binary number subtraction using the circuit above by using the 2's-complement notation on all the bits in B. This is achieved by inverting all the bits of B using a NOT gate and then adding extra one using the carry-input as shown in Figure 9.3. Also, in the above circuit for the 4-bit binary adder, the first carry-in input is held LOW at logic "0", for the circuit to perform subtraction this input needs to be held HIGH at "1". With this in mind a ripple carry adder can with a small modification be used to perform half subtraction, full subtraction and/or comparison.

9.2.3 The Digital Comparator

This is a combinational logic circuit that compares the digital signals present at their input terminals and produces an output depending upon the condition of those inputs. For example, when you write a line of code and state that, If $A > B$, then ..., the computer circuitry should be able to perform that comparison. The digital comparator accomplishes this using several logic gates that operate on the principles of Boolean algebra. So comparators are circuits used to check whether two digital words are the same. There are two main types of digital comparator:-

1. **Identity Comparator**:-this is a digital comparator that has only one output terminal for when $A = B$, that is when $A = B = 1$ or $A = B = 0$
2. **Magnitude Comparator**:- this is a type of digital comparator that has three output terminals, one for when $A = B$, one for when $A > B$, and the last one for when $A < B$

A Digital Comparator is used to compare a set of variables or unknown numbers, for example A ($A_1, A_2, A_3, \dots, A_n$, etc) against that of a constant or unknown value such as B ($B_1, B_2, B_3, \dots, B_n$, etc)

and produce an output condition or flag depending upon the result of the comparison. For example, a magnitude comparator of two 1-bits, (A and B) inputs would produce the following three output conditions when compared to each other. This is useful if we want to compare two variables and want to produce an output when any of the above three conditions are achieved. Consider the simple 1-bit comparator in Figure 9.4.

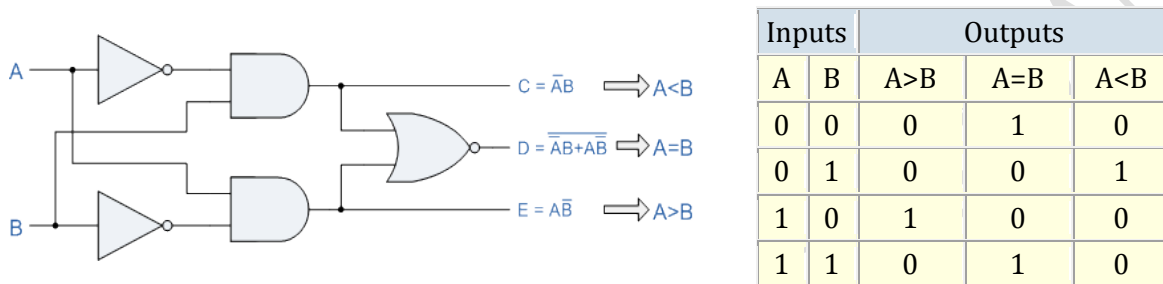


Figure 9.4: 1-bit Comparator

The operation of a 1-bit digital comparator is given in the Truth Table. You may notice two distinct features about the comparator from the above truth table. Firstly, the circuit does not distinguish between either two "0" or two "1"'s as an output A = B is produced when they are both equal, either A = B = "0" or A = B = "1".

Quiz: Can you recall which logic gate has this characteristics?

In addition to comparing individual bits, we can design larger bit comparators by cascading together n – comparators to produce an n -bit comparator just as we did for the n -bit adder. For example, the symbol of a 4-bit Magnitude Comparator is shown in figure 9.5. Here, two 4-bit words are compared to each other to produce the relevant output with one word connected to inputs A and the other to be compared against connected to input B as shown.

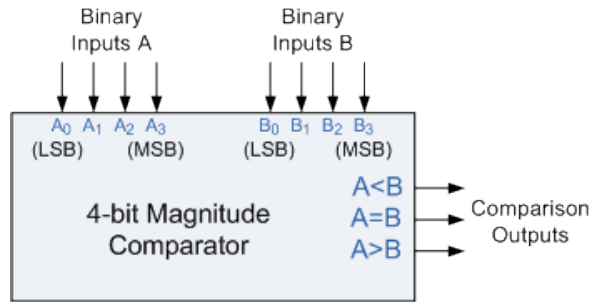


Figure 9.5: A 4-bit Magnitude Comparator

When comparing large binary numbers, to save time the comparator starts by comparing the MSB first. If equality exists, then it compares the next lowest bit and so on until it reaches the LSB. If equality still exists then the two numbers are defined as being equal. If inequality is found, then either $A > B$ or $A < B$.

9.3 Data transmission Combinational Logic circuits

These are circuits that transfer data from one side of a circuit to another side. In some cases, the devices will have to 'choose' which data to transmit, depending on some conditions. Such circuits included the Multiplexers, De-multiplexers, Encoders and Decoders.

9.3.1 The Multiplexer

A Multiplexer is more commonly called data selector and shortened to "MUX" or "MPX". Multiplexers are combinational logic switching devices that operate like very fast-acting multiple position rotary switch. Multiplexers provide a way of selecting one out of many digital signals. A multiplexer will in general have n inputs (either 2, 4, 8 or 16 individual inputs) usually called "channels", and one output, with m control lines which are used to select one of the n inputs. The block diagram of a multiplexer is shown in Figure 9.6.

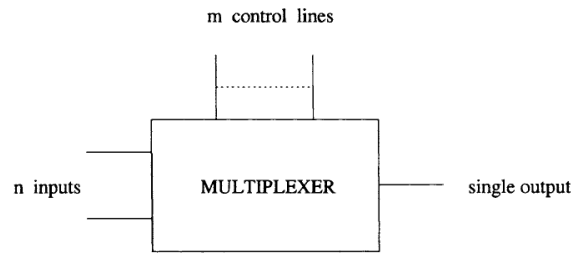


Figure 9.6: A block diagram of a multiplexer

The job of a multiplexer is to allow multiple signals to *share* a single common output. The *n*-input channel that is routed through to the output is determined by the bit pattern on the *m* control lines. The relationship between *n* and *m* is given by $n = 2^m$. Hence, the number of input lines that can be multiplexed is 2^m . Multiplexers are usually referred to as *n*-to-1 or 1-of-*n* multiplexers or data selectors. Figure 9.10 shows the circuit of a 2-to-1 MUX.

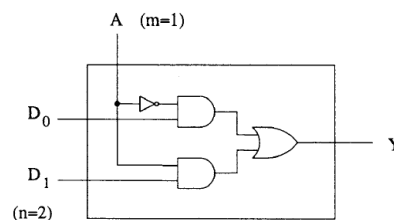


Figure 9.10: a simple 2-to-1 MUX.

Note that it has two inputs ($n= 2$) D_0 and D_1 , with a single control line ($m= 1$). So if $A =0$, then the output from the AND gate with D_1 as an input must be 0, whilst the output from the other AND gate will be $\bar{A} \cdot D_0 = 1 \cdot D_0 = D_0$. So, the output from the multiplexer, $= D_0 + 0 = D_0$. Similarly if $A = 1$ then $Y = D_1$. Therefore, in Boolean expression: $Y = \bar{A}D_0 + AD_1$

One way of thinking of the action of a multiplexer is that only one of the AND gates is ever activated and so allows the input signal fed to it through to the OR gate. Multiplexers are used as one method of reducing the number of logic gates required in a circuit or when a single data line is required to carry two or more different digital signals.

Example 9.1: Draw the circuit diagram and truth table, and give the Boolean equation describing the output, of a 4-to-1 multiplexer.

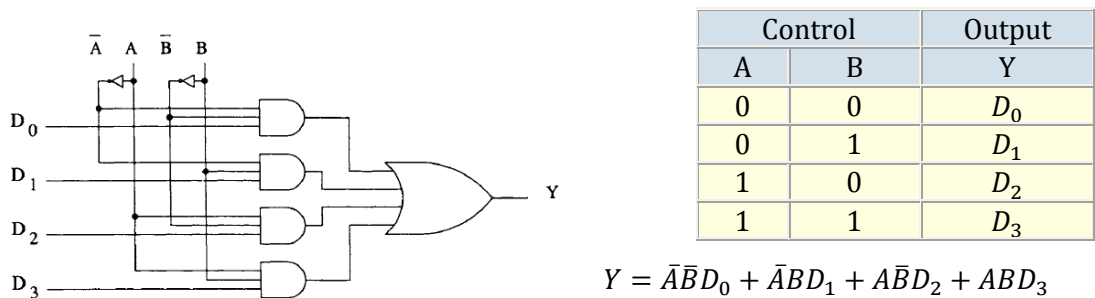


Figure 9.11: 4-to-1 Channel Multiplexer

In this example at any one instant in time only ONE of the four AND gates is activated, connecting only one of the input lines D_0 to D_4 to the single output at Y. As earlier noted, the gate which is active depends upon the addressing/ control input code on lines "A" and "B". The symbol of this 4-to-1 MUX is shown in Figure 9.12.

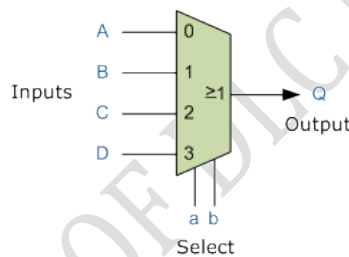


Figure 9.12: Multiplexer Symbol

It is important to note that Multiplexers are not limited to just switching a number of different input lines or channels to one common single output. There are also some multiplexers that can switch their inputs to multiple outputs and have arrangements or 4-to-2, 8-to-3 or even 16-to-4. An example of a simple Dual channel 4 input multiplexer (4 to 2) is given in Figure 9.13.

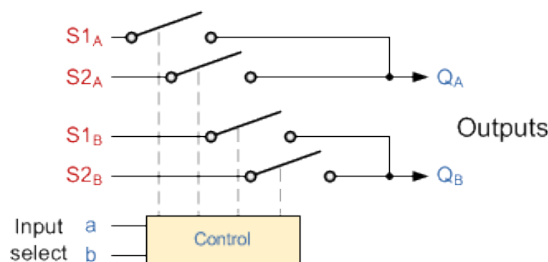


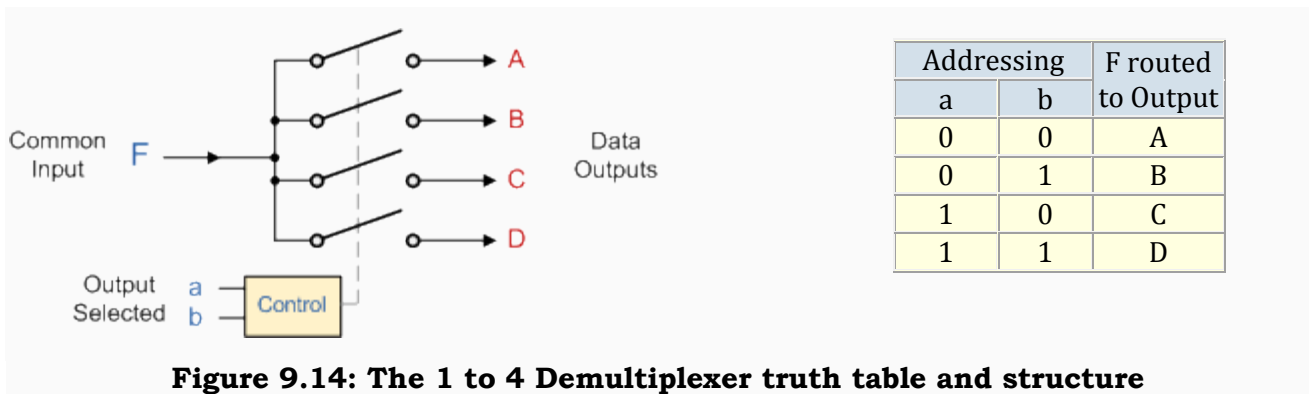
Figure 9.13: A 4-to-2 Multiplexer

In Figure 9.13, the 4 input channels are switched to 2 individual output lines but larger arrangements are also possible. This simple 4 to 2 configuration could be used for example, to switch audio signals for stereo pre-amplifiers or mixers. The Multiplexer is a very useful combinational device that has its uses in many different applications such as signal routing, data communications and data bus control. When used with a demultiplexer, parallel data can be transmitted in serial form via a single data link such as a fibre-optic cable or telephone line. They can also be used to switch either analogue, digital or video signals.

In summary, you can see a multiplexer as a television channel selector. All of the stations are broadcast constantly to the television's input, but only the channel that has been selected is displayed.

9.3.2 The Demultiplexer

A Demultiplexer is often called a data distributor and shortened as "DEMUX". Demultiplexers provide the reverse operation of multiplexers since they allow a single input to be routed to one of n outputs, selected via m control lines ($n = 2^m$). This circuit element is usually referred to as a 1-of- n demultiplexer. The circuit basically consists of n AND gates, one for each of the 2^m possible combinations of the m control inputs, with the single line input fed to all of these gates. Since only one AND gate will ever be active this determines which output the input is fed to. The block, and circuit, diagram of a 1-of-4 demultiplexer is shown in Figure 9.14. So you can say that the demultiplexer converts a serial data signal at the input to a parallel data at its output lines as shown in Figure 9.14.



The function of the Demultiplexer in Figure 9.14 is to switch one common data input line to any one of the 4 output data lines A to D. As with the multiplexer the individual solid state switches are selected by the binary input address code on the output select pins "a" and "b" and by adding more address line inputs it is possible to switch more outputs giving a 1-to-2ⁿ data line outputs. The implementation of the demultiplexer above using individual logic gates would require the use of six individual gates consisting of AND and NOT gates as shown in figure 9.15.

Demultiplexer Symbol

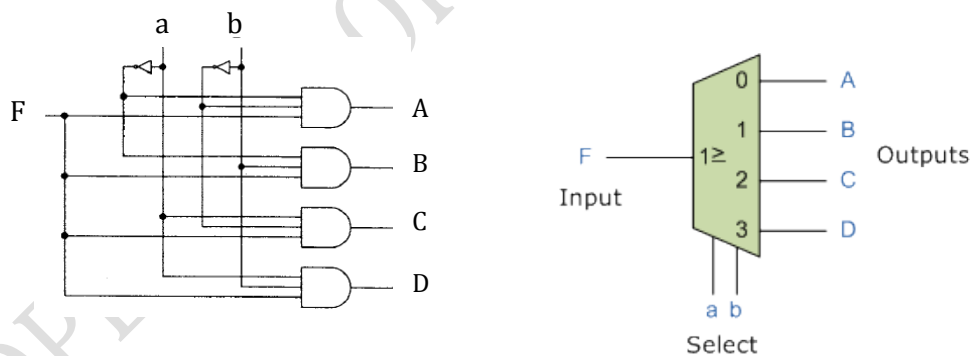


Figure 9.15: The 1 to 4 Demultiplexer Symbol and circuit diagram

9.3.3 The Digital Encoder

Encoders are circuits that convert a single active signal (out of *r* inputs) into a coded binary, *s-bit*, output (this would be referred to as an *r-line- to-s-line* encoder). Unlike a multiplexer that selects one individual data input line and then sends that data to a single output line or switch, a Digital Encoder takes *ALL* its data inputs one at a time and then converts them into a single

encoded output. A digital encoder is sometimes called a binary encoder, which is a multi-input combinational logic circuit that converts the logic level "1" data at its inputs into an equivalent binary code at its output. Generally, digital encoders produce outputs of 2-bit, 3-bit or 4-bit codes depending upon the number of data input lines. An "*s-bit*" binary encoder has 2^s input lines and *s-bit* output lines with common types that include 4-to-2 (Figure 9.16), 8-to-3 and 16-to-4 line configurations. The output lines of a digital encoder generate the binary equivalent of the input line whose value is equal to "1" and are available to encode either a decimal or hexadecimal input pattern to typically a binary or BCD output code.

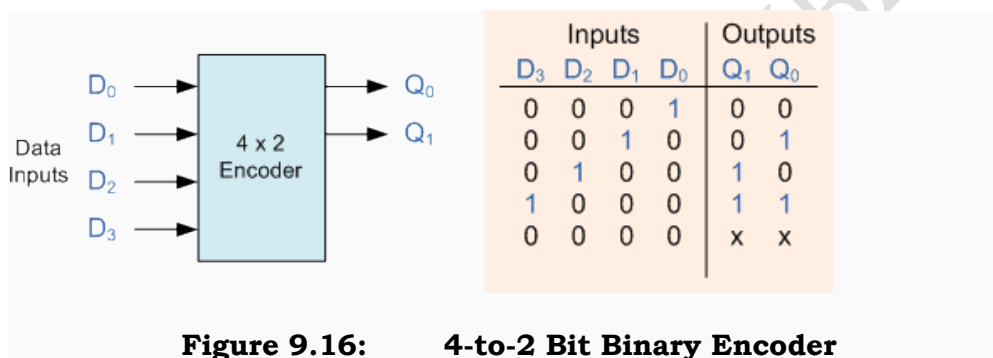


Figure 9.16: 4-to-2 Bit Binary Encoder

One of the main disadvantages of standard digital encoders is that they can generate the wrong output code when there is more than one input present at logic level "1". For example, if we make inputs D₁ and D₂ HIGH at logic "1" at the same time, the resulting output is neither at "01" or at "10" but will be at "11" which is an output binary number that is different to the actual input present. Also, an output code of all logic "0"s can be generated when all of its inputs are at "0" OR when input D₀ is equal to one. A way to overcome this problem is to "Prioritise" the level of each input pin and if there was more than one input at logic level "1" the actual output code would only correspond to the input with the highest designated priority. Then this type of digital encoder is known commonly as a **Priority Encoder**. Priority encoders can be used to reduce the number of wires needed in a particular circuits or application that have multiple inputs. Other applications especially for **Priority Encoders** may include detecting interrupts in microprocessor applications. Here the microprocessor uses interrupts to allow peripheral devices such as the disk drive, scanner, mouse, or printer etc., to communicate with it, but the

microprocessor can only "talk" to one peripheral device at a time. The processor uses "Interrupt Requests" or "IRQ" signals to assign priority to the devices to ensure that the most important peripheral device is serviced first. The order of importance of the devices will depend upon their connection to the priority encoder.

9.3.4 Binary Decoder

A Decoder is the exact opposite of an Encoder. It is basically, a combinational logic circuit that converts the binary code data at its input into one of a number of different output lines, one at a time producing an equivalent decimal code at its output. Binary Decoders have inputs of 2-bit, 3-bit or 4-bit codes depending upon the number of data input lines. An *n-bit* decoder has 2^n output lines. Therefore, if it receives n inputs (usually grouped as a binary or Boolean number) it activates one and only one of its 2^n outputs based on that input with all other outputs deactivated. A decoder output code/ lines normally has more bits than its input code and practical binary decoder circuits include, 2-to-4, 3-to-8 and 4-to-16 line configurations. An example of a 2-to-4 line decoder along with its truth table is given in figure 9.17. It consists of an array of four AND gates, one of which is selected for each combination of the input signals A and B.

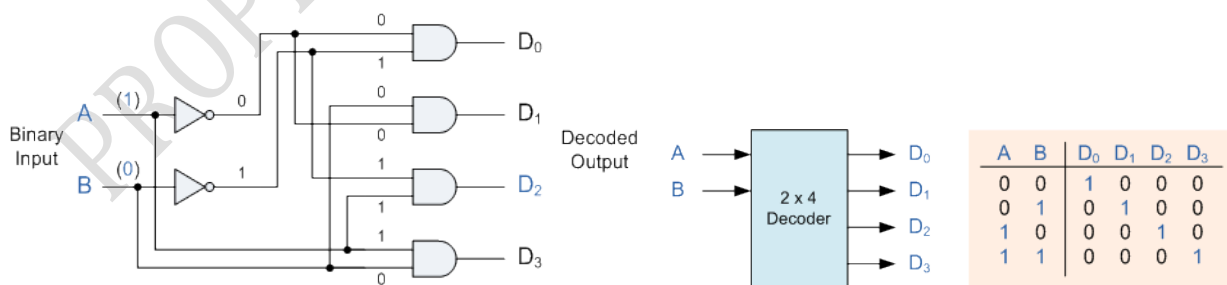


Figure 9.17: 2-to-4 line binary decoder

In this simple example of a 2-to-4 line binary decoder, the binary inputs A and B determine which output line from D₀ to D₃ is "HIGH" at logic level "1" while the remaining outputs are held "LOW" at logic "0" so only one output can be active (HIGH) at any one time. Therefore, whichever output line is "HIGH" identifies the binary code present at the input, in other words it "de-codes" the binary input and these types of binary decoders are commonly used as **Address Decoders** in microprocessor memory applications.

9.4 Code Converters

These are Combinational logic circuits that convert their inputs from one number code to another. The most common one is the BCD to 7-segment decoder.

9.4.1 BCD to 7-Segment Display Decoder

If you ever seen a digital message scrolling along what looks like dotted lines, then you have seen a 7-segment display. The 7-segment **LED** (Light Emitting Diode) or **LCD** (Liquid Crystal) displays, provide a very convenient way of displaying information or digital data in the form of numbers, letters or even alpha-numerical characters and they consist of 7 individual LED's (the segments), within one single display package.

In order to produce the required numbers or HEX characters from 0 to 9 and A to F respectively, on the display, the correct combination of LED segments need to be illuminated and **BCD to 7-segment Display Decoders** do that. A standard 7-segment LED display has 8 input connections, one for each LED segment and one that acts as a common terminal or connection for all the internal segments. Some single displays have an additional input pin for the *decimal point* in their lower right or left hand corner. Figure 9.18 shows the display format for the standard 7-segment LED, While the truth table for the circuit follows.

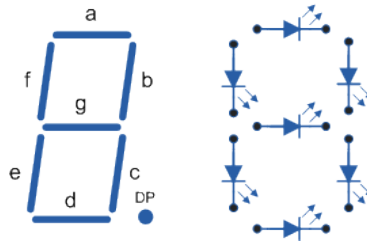
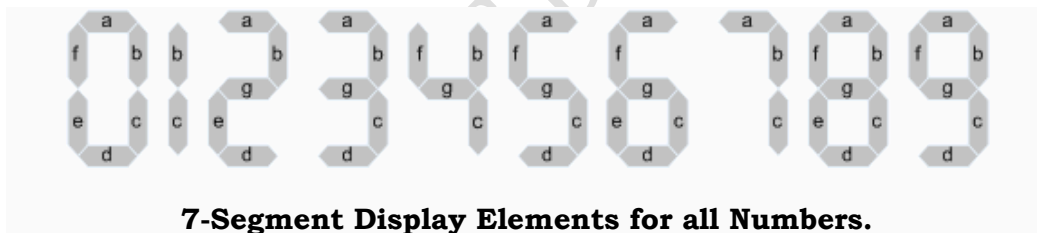


Figure 9.18: 7-Segment Display Format

Display Table for a 7-segment circuit

Individual Segments							Display
a	b	c	d	e	f	g	
x	x	x	x	x	x		0
	x	x					1
x	x		x	x		x	2
x	x	x	x			x	3
	x	x			x	x	4
x		x	x		x	x	5
x		x	x	x	x	x	6
x	x	x					7

Individual Segments							Display
a	b	c	d	e	f	g	
x	x	x	x	x	x	x	8
x	x	x			x	x	9
x	x	x			x	x	A
		x	x	x	x	x	b
x			x	x	x		C
	x	x	x	x		x	d
x			x	x	x	x	E
x				x	x	x	F



7-Segment Display Elements for all Numbers.

It can be seen that to display the number 8, all 7 segments would need to be lit. The BCD to 7-Segment Display Decoder are used to reduce the total number of connections. The decoder sends a signal in BCD format as input while the output it used to drive the various segments, depending on the current input. **Binary Coded Decimal** (BCD) numbers are made up using just 4 data bits and range from 0 to 9.

A binary coded decimal (BCD) to 7-segment display decoder have 4 BCD inputs and 7 output lines, one for each LED segment. This allows a smaller 4-bit binary number to be used to display all the denary numbers from 0 to 9 and by adding two displays together, a full range of numbers from 00 to 99 can be displayed with just a single byte of 8 data bits.

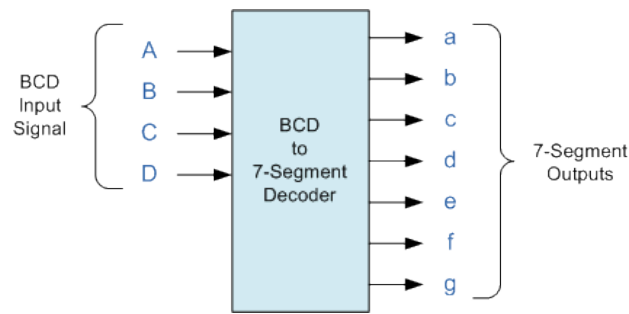
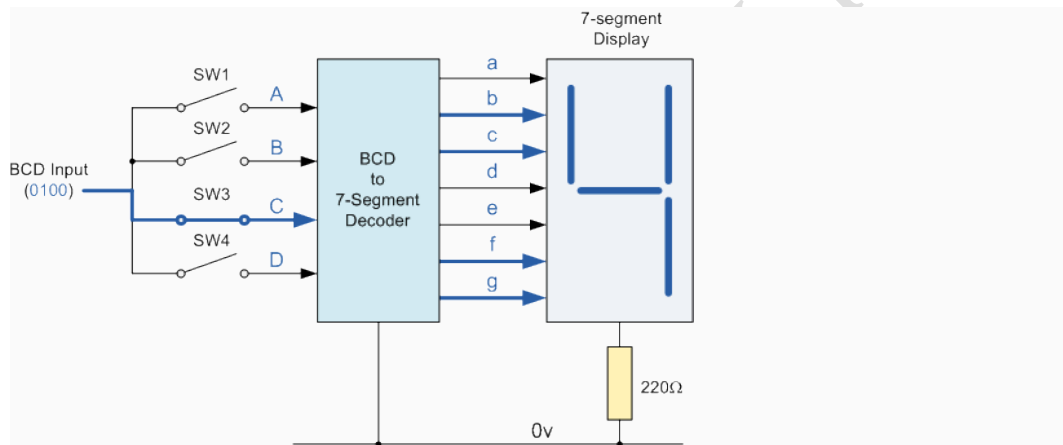


Figure 9.19: The block diagram of a BCD to 7-Segment Decoder

Example: An example of the 4-bit BCD input (0100) representing the number 4 is given below.



In practice current limiting resistors of about 150Ω to 220Ω would be connected in series between the decoder/driver chip and each LED display segment to limit the maximum current flow.

Summary

- Combinational Logic circuits can be classified by the type of function they perform in the microprocessor.
- **Adder** are used for binary arithmetic (addition) and with some modification, can also be used for subtraction.

- **Comparators** are used to compare the magnitude of binary digits and often give three different outputs, depending on its finding.
- The **multiplexer** allows multiple signals to share a single common output, one at a time.
- The **demultiplexer** takes one single input data line and then switches it to any one of a number of individual output lines one at a time.
- The **Digital Encoder** is a combinational circuit that generates a specific code at its outputs such as binary or BCD in response to one or more active inputs. There are two main types of digital encoder. The **Binary Encoder** and the **Priority Encoder**. The **Binary Encoder** converts one of the inputs into an s-bit output. The **Priority Encoder** is another type of combinational circuit similar to a binary encoder, except that it generates an output code based on the highest prioritised input.
- A **binary decoder** converts coded inputs into coded outputs, where the input and output codes are different.

Post-Test

1. What is the function of an 8-to-1 multiplexer?
2. What does a demultiplexer do?
3. Why can a decoder be constructed from a demultiplexer?
4. What three types of combinational logic circuits can an XOR gate be used to construct?
5. What does a full adder do?
6. What type of circuit is a ripple carry adder, what basic unit is it built from?
7. Differentiate between an identity comparator and a magnitude comparator
8. How many control input lines will the following have?
 - a. an 8-to-1 multiplexer
 - b. a 1-to-16 demultiplexer
9. Draw a sketch of a 4-bit Ripple Carry Binary Adder and show what the various outputs will be if the device is used to add $A=1110_2$ to $B=101_2$
10. What is the main difference between Standard Encoders and a Priority Encoders?

References

Essential Electronics

Duncan, Tom. 1997, Electronics for Today and Tomorrow. Spain: John Murray.

www.electronics-tutorials.ws

PROPERTIES OF DLC UI, IBADAN

LECTURE TEN

SEQUENTIAL LOGIC CIRCUIT BASICS

Introduction

In the preceding lectures, we have been studying combinational logic circuits, whose outputs depend solely on their current input. However, once in a while, it would be good a digital system to store some data for later retrieval, at least before the system is shut down. It means there has to be a way to “remember” what has been “stored” and make such data readily available for processing. This brings us to the topic of this lecture, Sequential logic circuits which have some form of memory. In these types of circuits, things happen in a sequence, hence their name.

Objectives

At the end of this lecture, you should be able to:

1. State the main difference between a combinational logic circuit and a sequential logic circuit.
2. Name the four main types of Flip Flops
3. Draw and describe the operation of the S-R flip-flop
4. State what a T-type flip-flop does
5. State what is means to “store” 1 bit in a flip flop.

Introduction

Sequential circuits consist of combinational logic as well as memory elements (used to store certain circuit states). Their outputs depend on *BOTH* current input values and previous input values. Put another way, the output of a sequential circuit may depend upon its previous

outputs and so in effect has some form of 'memory'. This means that sequential circuits are essentially combinational circuits with feedback. A block diagram of a generalised sequential circuit is shown in Figure 10.1. The circuit contains a block of combinational logic which has two sets of inputs and two sets of outputs described as follows:

- A , the present (external) inputs to the circuit;
- y , the inputs fed back from the outputs;
- Z , the present (external) outputs from the combinational circuit;
- Y , the outputs that are fed back into the combinational circuit.

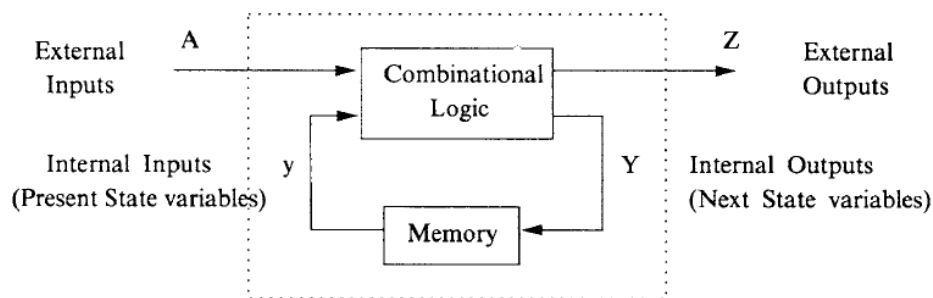


Figure 10.1: The general form of a sequential logic circuit

Note that the outputs, Y , are fed back via the memory block to become the inputs, y , and that y are called the 'present state' variables because they determine the current state of the circuit, with Y the 'next state' variables as they will determine the next state the circuit will enter.

An important concept to note is that sequential circuits can be considered at any time to occupy a certain '**state**'. These 'states' are dependent upon the internal feedback, and sometimes the external inputs as well. This idea of the circuit possessing states is fundamental to sequential circuits since they are often designed and analysed by the manner, or sequence, in which the available states are visited for given sequences of inputs. The sequential logic circuits we will study in this lecture are usually called **Bistables** or **Flip-flops**. Bistables are switching circuits

with two stable states. The states are either a logic level "1" or a logic level "0" and they will remain "latched" indefinitely in a state or condition until some other input trigger pulse or signal is applied which will cause it to change its state once again. They are fundamental components/ devices used in Counters, Shift registers and Memories.

10.1 The S-R Flip-Flop

An S-R Flip-Flop acts as a basic one-bit memory device that has two inputs, one which will "**SET**" the device and another which will "**RESET**" the device back to its original state. It also has two outputs; an output Q that will be either at a logic level "1" or logic "0" depending upon this Set/Reset condition and the other output, it's complement, \bar{Q} . This flip-flop can be constructed using any of the universal logic gates (can you recall them?).

These devices are termed "Flip-flop" because of the actual operation of the device, as it can be "Flipped" into one logic state or "Flopped" back into another. The simplest way to make any simple one-bit Set/Reset (SR) flip-flop is to connect together a pair of cross-coupled 2-input NAND Gates to form a Set-Reset Bistable, so that there is feedback from each output to one of the other NAND Gate inputs. This device consists of two inputs, one called the Set, S and the other called the Reset, R with two outputs Q and its inverse or complement, \bar{Q} as shown in figure 10.2.

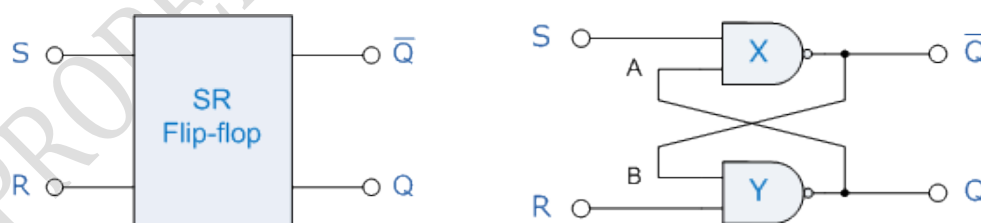


Figure 10.2: NAND gate S-R Bistable / flipflop

Before we look at the mode of operation of this flip-flop, I want you to draw the truth table for a two-input NAND gate and keep it close by. You will need it!

The Set State

Studying figure 10.2, you can see that If the input R is at logic level "0" and input S is at logic level "1", then the Y - NAND Gate has at least one of its inputs at logic "0" therefore, its output Q must be at a logic level "1" (remember I asked you to keep the NAND gate truth table handy!). Output Q is also fed back to input A. This makes both inputs to the X - NAND Gate to be at logic level "1", and therefore its output \bar{Q} must be at logic level "0". If the Reset input R changes state, and now becomes logic "1" with S remaining at logic level "1", the Y- NAND Gate inputs are now $R = "1"$ and $B = "0"$ and since one of its inputs is still at logic level "0" the output at Q remains at logic level "1" and the circuit is said to be "Latched" or "Set" with $Q = "1"$ and $\bar{Q} = "0"$.

Reset State

In this second stable state, Q is at logic level "0", and \bar{Q} is at logic level "1". This is given by $R = "1"$ and $S = "0"$. The X NAND gate has one of its inputs at logic "0" therefore, its output \bar{Q} must equal logic level "1". Output \bar{Q} is fed back to input B, so both inputs to the Y-NAND gate are at logic "1", therefore, $Q = "0"$. If the set input, S now changes state to logic "1" with R remaining at logic "1", output Q still remains at logic level "0" and the circuit's "Reset" state has been latched.

Table 10.1: Characteristic Table for Set-Reset Function

State	S	R	Q	\bar{Q} (Q bar)
Set	1	0	1	0
	1	1	1	0
Reset	0	1	0	1
	1	1	0	1
Indeterminate	0	0	1	1
	1	1	?	?

Indeterminate State

From Table 10.1 you can see that when both inputs $S = "1"$ and $R = "1"$, the outputs Q and Q -bar can be at either logic level "1" or "0", depending on the state before this input condition existed. Therefore things must happen in sequence for the device to function well. Both R and S must not be at logic level "1" simultaneously. The SR Latch is said to be in an "invalid" condition (Meta-stable) if both the Set and Reset inputs are activated simultaneously. Similarly, the input

state $R = "0"$ and $S = "0"$ is an undesirable or invalid condition and must be avoided because this will give both outputs Q and \bar{Q} to be at logic level "1" at the same time!

Figure 10.3 shows how to construct a simple 1-bit SR Flip-flops using two NOR Gates connected the same configuration. The circuit will work in a similar way to the NAND gate circuit above, except that the invalid condition exists when both its inputs are at logic level "1".

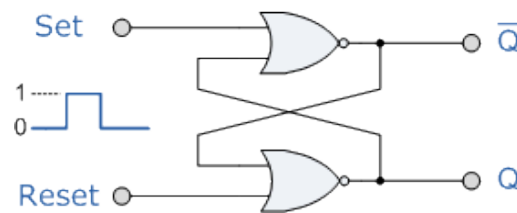


Figure 10.3: The NOR Gate SR Flip-flop

Quiz: Draw the Characteristic Table for the NOR gate implementation of the SR flip-flop.

10.2 The JK Flip-Flop

From the section 10.1, we know that the SR NAND Flip-flop suffers from two basic problems: (1) the $S = 0$ and $R = 0$ condition or $S = R = 0$ must always be avoided, and (2) if S and R ($S=R=1$) are activated simultaneously. To overcome these two problems the JK Flip-Flop was developed.

The JK Flip-Flop is basically an SR Flip-Flop with the addition of clock input circuitry that prevents the illegal or indeterminate output that can occur when both input S and input R equals logic level "1". The symbol and circuit diagram for a JK Flip-flop is shown in figure 10.4.

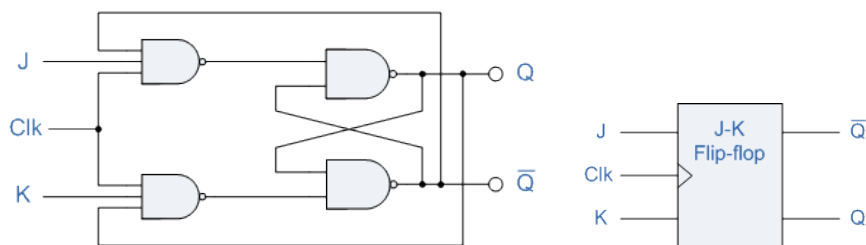


Figure 10.4: The J-K Flip-Flop

From figure 10.4 you will notice that the S and the R inputs of the SR Bistable have been replaced by two inputs called the J and K inputs, respectively. The two 2-input NAND gates of the gated SR Bistable have now been replaced by two 3-input NAND gates with the third input of each gate connected to the outputs Q and Q-bar. This cross coupling allows the previously invalid condition of S = "1" and R = "1" state to be usefully used to turn it into a "**Toggle action**" as the two inputs are now interlocked. The Characteristic table for the JK flip-flop is shown in Table 10.2.

Table 10.2 The Characteristic Table for the JK flip-flop

J	K	Q after clock pulse
0	0	Stays at 0 or 0
0	1	Resets to 0
1	0	Sets to 1
1	1	Toggles

From its circuit diagram, we notice that the JK Flip-flop is basically an SR Flip-flop with feedback and which enables only one of its two input terminals, either Set or Reset at any one time thereby eliminating the invalid condition seen previously in the SR Flip-flop circuit. Also when both the J and the K inputs are at logic level "1" at the same time, and the clock input is applied, the circuit will "Toggle" from a Set state to a Reset state, or visa-versa. This results in the JK Flip-flop acting more like a **T-type Flip-flop** when both terminals are "HIGH". In summary, when clock pulses are applied to Clk input of the JK flip-flop, it:

- (i) Retains it's present state if $J = K = 0$,
- (ii) Acts as a D-type flip-flop if J and K are different,
- (iii) Acts as a T-type flip-flop if $J = K = 1$
- (iv) Sets to 1, if $J=1$
- (v) Resets to 0, if $K= 1$

10.3 The T flip-flop

The T flip-flop gets its name from the ability of the device to toggle or change state with every input clock pulse, as long as the control, T is at logic level "1". That is, regardless of the state of the flip-flop at time t, it will assume the complement state on command of a pulse if the input T is high. Whenever it is desired that the flip-flop remain stable and not change state, the T input must be held to a logic "0" level. T flip-flops are used mainly in counter devices on digital circuits as we will see in the next lecture. The T flip-flop can be achieved using the JK flip-flop by setting $J = K = 1$.

10.4 The D-type flip-flop

The symbol and circuit diagram of the D-type flip-flop is given in figure 10.5. Recall that one of the main disadvantages of the SR NAND Gate flip-flop circuit is that the indeterminate input condition of "SET" = logic "0" and "RESET" = logic "0" is forbidden. That state will force both outputs to be at logic "1", over-riding the feedback latching action and whichever input goes to logic level "1" first will lose control, while the other input still at logic "0" controls the resulting state of the latch. In order to prevent this from happening an inverter can be connected between the "SET" and the "RESET" inputs to produce a D-Type Data Latch or simply Data Latch as it is generally called.

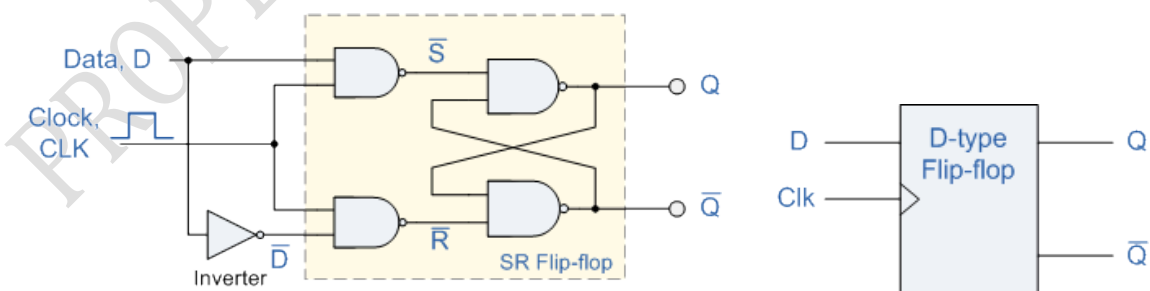


Figure 10.5: The D-Type Flip-flop

By connecting an inverter to the SR flip-flop we can "SET" and "RESET" the flip-flop using just one input as now the two latch inputs are complements of each other. This single input is simply

called the "DATA (D)" input. If a logic 1 is applied to the D input, regardless of what state the flip-flop is in before the pulse, it will assume a 1 state on its normal Q output. This happens however, when a clock or enable input is applied. The flip-flop acts as a buffer. It will store and output whatever logic level is applied to its data terminal so long as the clock input is high. Once the clock input goes low the flip-flop will not change state, therefore it store whatever data was present on its output before the clock transition occurred. In other words the output is "latched" at either logic "0" or logic "1". The Characteristic table for the data-latch is given in Table 10.3.

Table 10.3: Characteristic Table for the D-type Flip-flop

Clk	D	Q	Q bar	OUTPUT
0	x	Q	Q bar	HOLD
1	0	0	1	RESET
1	1	1	0	SET

Suppose you design a circuit and would want to store and hold an n-bit binary number for further processing, how do you imagine the computer achieves this? Well, as we just studied, the data latch will hold and output whatever input signal is applied to its D-input as long as the clock pulse is high. Once the clock pulse is set low, it will hold (store) the last input that was applied. The Data Latch is a very useful devices in electronic and computer circuits, they are used to build registers, which are temporary memory locations in digital devices. Figure 10.6 shows the arrangement of a 4-bit data latch.

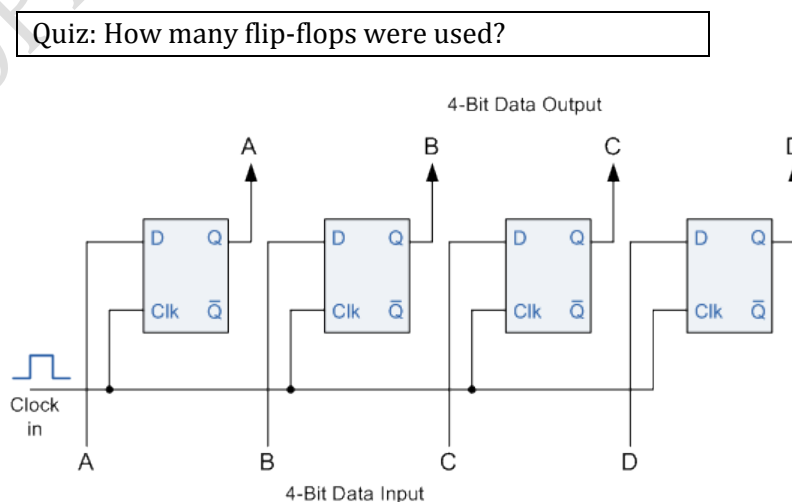


Figure 10.6 A 4-bit Data Latch

10.5 Asynchronous and Synchronous Sequential logic circuits

Sequential logic circuits can be either synchronous or asynchronous in nature. The timing of the operation of *asynchronous circuits*, is not controlled by any external timing mechanism. Rather, as soon as changes are made at the inputs of such a circuit they take effect at the outputs. *Synchronous circuits* are those that possess a clock of some sort which regulates the feedback process. Hence the timing of changes in the outputs, in response to changes at the inputs, are controlled by the 'ticking' of a clock. Consequently, the timing of the operation of sequential circuits can be, and usually is, synchronised to other parts of a larger circuit. Figure 10.7 shows the block diagrams for these two types of sequential circuits. Synchronous sequential circuits are also referred to as clocked circuits, whilst Asynchronous ones are known as unclocked.

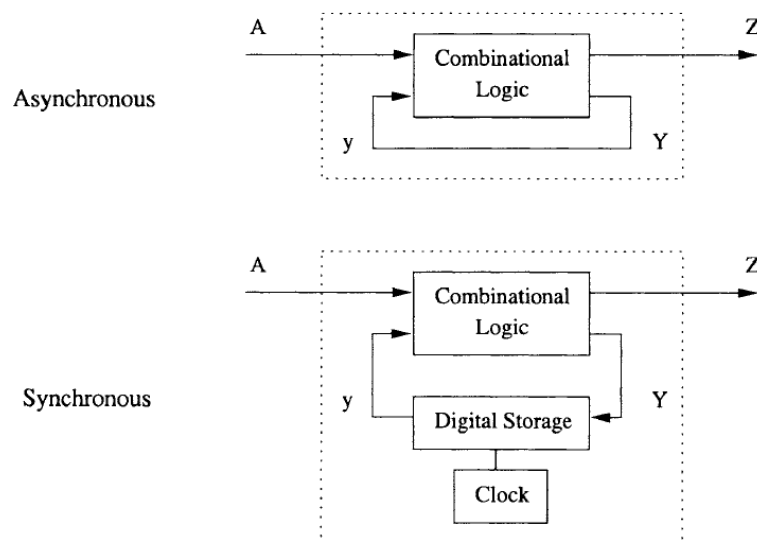


Figure 10.7: General structure of asynchronous and synchronous sequential circuits

Despite the fact that asynchronous circuits are more difficult to design than synchronous circuits, asynchronous circuits do have certain benefits. One of such benefits include the fact

that because they are free running, their speed of operation is limited solely by the characteristics of the components from which they are built and not by the speed at which they are clocked. Consequently, asynchronous circuits have the potential to work at higher speeds than synchronous ones. Also: some systems may require a circuit to respond immediately to changing inputs (i.e. the inputs cannot be synchronised to the rest of the circuit); in very large circuits the unavoidable delays as a signal.

Summary

- The SR Bistable latch is activated or Set by a logic "1" applied to its S input and deactivated or Reset by a logic "1" applied to its R.
- The JK Flip-Flop was invented to eradicate the invalid state of the SR flip-flop. Therefore, when the clock pulse is applied to the Clock input of the JK flip-flop, the flip-flop:
 - (i) Retains its present state if $J = K = 0$,
 - (ii) Acts as a D-type flip-flop if J and K are different,
 - (iii) Acts as a T-type flip-flop if $J = K = 1$
- The T flip-flop toggles or changes state with every clock pulse, as long as the control, T is at logic level "1".
- The Data-Latch, serves as a memory location that will hold whatever signal is applied to its input, as long as the clock pulse is low.
- The operation of asynchronous circuits, is not controlled by any external timing mechanism. Synchronous circuits are those that possess a clock of some sort which regulates the feedback process.

Post-Test

- 1 What is the basic difference between sequential and combinational logic circuits?
- 2 What is the general form of a sequential logic circuit?

- 3 What are the basic differences between asynchronous and synchronous sequential circuits?
- 4 Draw the NAND SR flip-flop and analyse it's mode of operations
- 5 How can a JK flip-flop be converted into a Toggle flip-flop

References

Essential Electronics

Duncan, Tom. 1997, Electronics for Today and Tomorrow. Spain: John Murray.

www.electronics-tutorials.ws

Mark Balch, 2003, Complete Digital Design, A Comprehensive Guide to Digital Electronics and Computer System Architecture, McGraw-Hill

William E. Wickes, 1968, Logic Design with Integrated Circuits

LECTURE ELEVEN

REGISTERS AND COUNTERS

Introduction

The Flip-flops treated in the previous lecture are programmed as counters and registers in one mode or another. They are found in almost all equipment containing digital logic. A computer continuously counts clock pulses, representing time, to control the sequencing of the internal program. A digital washing machine may count pulses that determine the step-by-step operation of a programmed sequence of events. There are binary counters, decade counters, random counters, synchronous counters, ripple counters, ring counters and shift registers, to mention a few. This chapter will treat the most important type of counters and give examples of how they are implemented.

Objectives

At the end of this lecture, you should be able to:

1. Recognize and describe the action of binary up counters
2. State what is meant by the term 'modulo'
3. Draw the block diagram of counters of different modulo
4. Draw the block diagram of an n-bit register
5. State the four modes in which shift registers operate

11.1 Clocks

In the previous lecture we mentioned clock pulses. In this lecture, we will study further how these clock pulses are used to 'control' registers and counters. What then are clock pulses? Clock pulses are supplied by some form of pulse generator, which may be a crystal-controlled

oscillator with a very steady repetition frequency, e.g. 10MHz, or an astable multivibrator or a mechanical switch turning a D.C. supply on and off. The pulse should have fast rise and fall times, that is, be good square waves. Figure 11.1 shows the wave form of clock pulses and the associated properties.

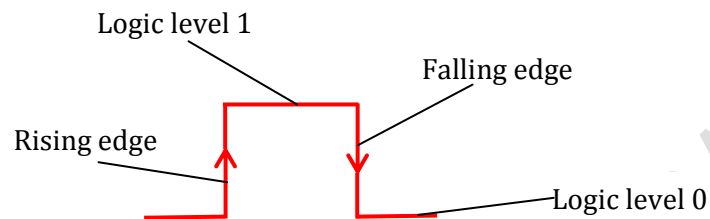


Figure 11.1: The clock pulse symbol

From figure 11.1 you can see that the clock wave form has four sides, the rising edge, falling edge, logic level 1 and logic level 0. Their meaning are self-explanatory from the figure. They determine when a Bistable will change state and ultimately the device which those Bistables are used to implement.

11.2 Level and Edge Triggering

There are two types of clocking or triggering operation in sequential circuits. In level triggering, bistables change state when the logic level of the clock pulse is 1 or 0. Those bistables are referred to as level triggered. In edge triggering, a change in voltage causes switching. If it occurs during the rise of the clock pulse from logic level 0 to 1 it is rising or positive-edge triggering and most modern clocked logic is of this type. In falling or negative-triggering, switching occurs when the clock pulse falls from 1 to 0. The T-type bistable is edge triggered.

In general, edge triggering is more satisfactory than level triggering because in the former, output changes occur at an exact instant during the clock pulse and any further input changes do not affect the output until the next clock rise (or fall). In level triggering, output changes can occur at any time while the clock pulse is 1, which may not be desirable.

11.3 Classification of Sequential Logic circuits

In Sequential Logic circuits, the actual clock signal determines when things will happen next. Simple sequential logic circuits can be constructed from standard Bistable circuits studied in the previous lecture such as Flip-flops and Latches. These Sequential Logic circuits can be divided into 3 main categories:

- 1 **Clock Driven** – These are also called synchronous circuits because they are synchronised to a common clock signal. A synchronous circuit has all of its flip-flops transition at the same time so that they settle at the same time.
- 2 **Event Driven** – Also termed asynchronous circuits as they react or change state when an external event occurs. It also means that the Flip-flops do not react to the same clock pulse.
- 3 **Pulse Driven** - A Combination of Synchronous and Asynchronous.

Although Sequential logic circuits can fall into any of the categories mentioned above, they also have another quality. A Sequential logic circuit that returns back to its original state once reset, is said to be "Cyclic" in nature.

11.4 Registers

Registers are collections of multiple flip-flops arranged in a group with a common function. The most common type being the shift registers. A shift register is a group of flip-flops programmed in such a way that data shifts from one flip-flop to the next in synchrony. A shift register is also recognized as a memory which stores a binary number and shifts it out when required. It consists of several D-type or J-K flip-flops cascaded together, such that one flip-flop stores a bit (0 or 1) in the binary number. Shift registers are used, for example, in calculators to store two binary numbers before they are added. Register arrangement can be for 8-bit, 16-bit, etc

depending on the architecture of the system. Shift registers provide a common clock and clock enable for all flip-flops. The clock enable allows external control of when the flip-flops get reloaded with new D-input values and when they retain their current values.

Shift registers exist in all permutations of serial and parallel inputs and outputs. The role of a shift register is to somehow change the sequence of bits in an array of bits. The bits may be fed in and out serially, i.e. one after the other, or in parallel, i.e. all together. As you soon see, shift registers consists of a number of single bit "D-Type Data Latches" connected together in a chain arrangement so that the output from one data latch becomes the input of the next latch and so on, thereby moving the stored data. They are usually provided with a Clear or Reset connection so that they can be "SET" or "RESET" as required. Generally, Shift Registers operate in one of four different modes: (1) Serial-in, Parallel-out (SIPO), (2) Serial-in, Serial-out (SISO), (3) Parallel-in, Parallel-out (PIPO), (4) Parallel-in, Serial-out (PISO). This will be explained using 4-bit shift registers.

11.4.1 Serial-in Parallel-out

Figure 11.2 depict the arrangement of a 4-bit Serial-in Parallel-out (SIPO) Shift Register. QD denotes the least significant bit of the binary number, while QA denotes the most significant bit.

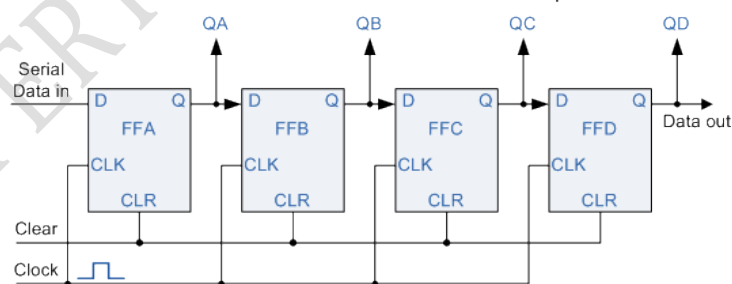


Figure 11.2: 4-bit Serial-in-Parallel-out Shift Register

Lets assume that all the flip-flops (FFA to FFD) have just been reset, this means that all the outputs QA to QD are at logic level "0". If a logic "1" is connected to the D input pin of FFA then on the first clock pulse the output of FFA and the resulting QA will be set to logic "1" with all the other outputs remaining at logic "0". Assume now that the D input pin of FFA has returned to

logic "0", then the next clock pulse will change the output of FFA to logic "0" and the output of FFB and *QB* to logic "1". The logic "1" that was initially applied to the input of FFA, has now moved or been "Shifted" one place along the register to the right. Consequently, when the third clock pulse arrives this logic "1" value moves to the output of FFC (*QC*) and so on until the arrival of the fifth clock pulse which sets all the outputs *QA* to *QD* back again to logic level "0" because the input to FFA has remained at a constant logic level "0".

The effect of each clock pulse is to shift the DATA contents of each stage one place to the right, and this is shown in table 11.1 until the complete DATA is stored, which can now be read directly from the outputs of *QA* to *QD*. Then the DATA has been converted from a Serial Data signal to a Parallel Data word.

Table 11.1 Output per clock pulse

Clock Pulse No	QA	QB	QC	QD
0	0	0	0	0
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1
5	0	0	0	0

Quiz: Suppose, this was an 8-bit register, how many clock pulses would be needed to store/ convert the same data?

11.4.2 Serial-in-Serial-out

This Shift Register is very similar to the one in section 11.4.1 except that this time the DATA is allowed to flow straight through the register. Since there is only one output the DATA leaves the shift register one bit at a time in a serial pattern and hence the name Serial-in-Serial-Out Shift Register. Figure 11.3 shows its arrangement.

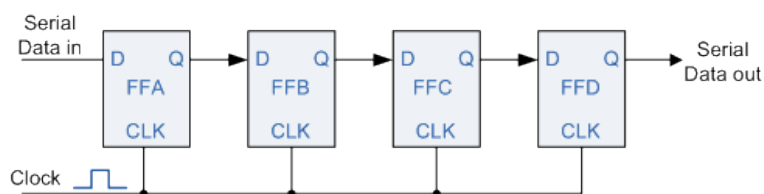


Figure 11.3: 4-bit Serial-in to Serial-out (SISO) Shift Register

This type of Shift Register also acts as a temporary storage device or as a time delay device, with the amount of time delay being controlled by the number of stages in the register, 4, 8, 16 etc or by varying the application of the clock pulses. For example, four clock pulses are needed to enter a four-bit number such as 0101 and another four pulses are need to move the data out serially.

11.4.3 Parallel-in-Serial-out

Parallel-in-Serial-out Shift Registers act in the opposite way to the Serial-in-Parallel-out one. The D input is applied in parallel form to the parallel input pins PA to PD of the register and is then read out sequentially from the register one bit at a time from PA to PD on each clock cycle in a serial format.

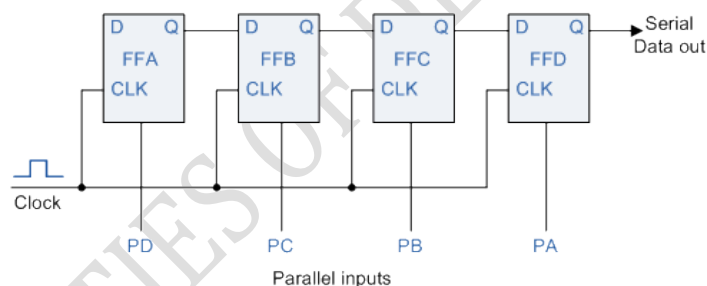


Figure 11.4 4-bit Parallel-in to Serial-out (PISO) Shift Register

As this type of Shift Register converts parallel data, such as an 8-bit data word into serial data it can be used to multiplex many different input lines into a single serial DATA stream which can be sent directly to a computer or transmitted over a communications line.

11.4.4 Parallel-in-Parallel-out

Parallel-in-Parallel-out Shift Registers also act as a temporary storage device or as a time delay device, with the amount of time delay being varied by the frequency of the clock pulses. The data is presented in a parallel format as shown in figure 11.5, to the parallel input pins PA to PD

and gets shifted simultaneously to the corresponding output pins QA to QD when the registers are clocked.

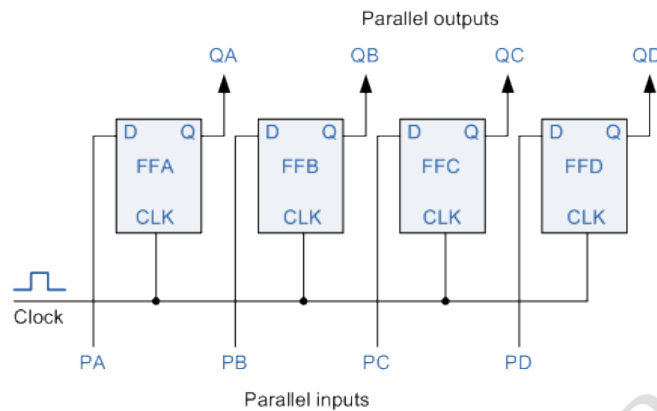


Figure 11.5 4-bit Parallel-in/Parallel-out (PIPO) Shift Register

Quiz: If the data, 1010 are applied to PA, PB, PC, PD for the PIPO shift register, how many clock pulses will be needed to read the outputs at QA, QB, QC, QD.

It is possible to have bi-directional shift registers especially for SIPO, which can shift the data stored in them to the left or to the right. For such devices, you will notice that depending on the data stored, a shift to the left signifies *integer* arithmetic multiplication by 2, while a shift to the right signifies *integer* division by 2. Check this out with the following four bit numbers

0100 0011 0111

11.5 Counters

Counters consist of flip-flops, especially the T-type, connected so that they toggle when the pulses to be counted are applied to their clock input. Counting is done in binary code, the bit 1 and 0 being represented by the 'high' and 'low' states of the bistable's Q output. Counters can either count up that is from zero to the maximum (useful in a digital clock or wrist watch) or

they can count down that is from the maximum to zero (useful in the display of digital microwave ovens and stop watches on you smart phones!). Digital counters have the following important characteristics:

- Maximum number of count
- Up-Down Count
- Asynchronous or Synchronous Operation
- Free-Running or Self-Stopping

Asynchronous counter are commonly referred to as **ripple counter** because the effect of the input clock pulse is first “felt” by first flip-flop (FF₀). That is the LSB flip-flop. So the effect of an input clock pulse “ripples” through the counter, taking some time, due to propagation delays, to reach the last flip-flop. Usually, only the first flip-flop receive clock pulse from the source, other flip-flops receive clock pulse from either Q or Q' of prior flip-flop. In **Synchronous Counters** on the other hand, the flip-flops have a fixed time relationship with each other AND receive clock pulse from a common source.

Modulo of a Counter

The modulus or simply “MOD” of a counter is the number of output states the counter goes through before returning itself back to zero or its starting point, that is, one complete cycle. A counter with *n*-flip-flops (if n=3) will count from 0 to 7 that is, $2^n - 1$. It is called a Modulo-8 or MOD-8 counter. More example are given below:

- A 3-bit Binary Counter = $2^3 = 8$, is a modulo-8 or MOD-8 counter, but counts from 0 to 7 as seen above.
- A 4-bit Binary Counter = $2^4 = 16$, is a modulo-16 or MOD-16 counter and counts from 0 to

15

Quiz: An 8-bit Binary Counter is a modulo _____ counter and counts from 0 to ____

11.5.1 Binary up-counter

A simple sketch of an asynchronous three-bit binary up counter is shown in Figure 11.6, consisting of cascaded T-type flip-flops FF₀, FF₁, FF₂ with the Q output of each feeding the clock input (CK) of the next to the left. The total count is given at any time by the states of Q₀ (the L.S.B.), Q₁ and Q₂ (the M.S.B.), that is the current count is read as, Q₂Q₁Q₀. The counting progresses upwards from 000 to 111 (7 in decimal) as shown in the table 11.2, before resetting to 000.

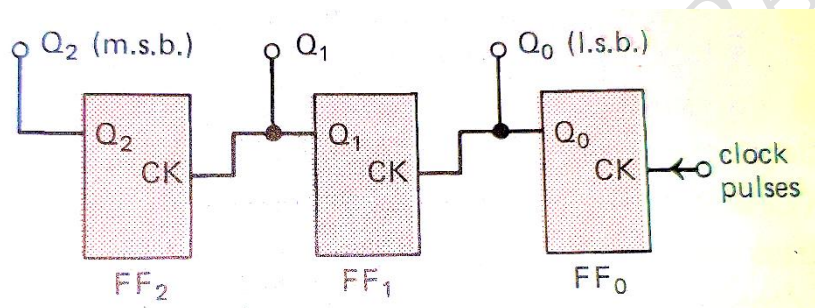


Figure 11.6: A 3-bit asynchronous binary-up counter

Suppose the flip-flops are triggered on the falling edge of a pulse and that initially Q₀, Q₁ and Q₂ are all reset to zero. On the falling edge *ab* of the first clock pulse, shown in Figure 11.7, Q₀ switches from 0 to 1. The resulting rising of edge *AB* of Q₀ is applied to CK of FF₁, which does not change state because *AB* is not a falling edge. Hence the output states are Q₂ = 0, Q₁ = 0 and Q₀ = 1, giving a binary count of 001.

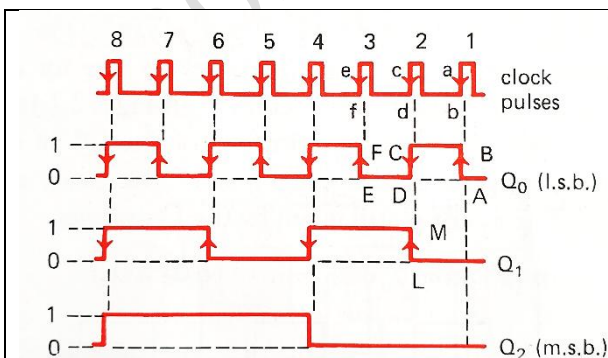


Table 11.2 Outputs for a 3-bit binary up counter

Clock Pulse No	Outputs		
	Q ₂	Q ₁	Q ₀
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0

Figure 11.7 States of a 3-bit binary up counter	<table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td style="text-align: center;">7</td> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> </tr> </table>	7	1	1	1
7	1	1	1		

The falling edge *cd* of the second clock pulse makes FF₀ change state again and Q₀ goes from 1 to 0. The falling edge *CD* of Q₀ switches FF₁ this time, Q₁ = 1. The rising edge *LM* of Q₁ leaves FF₂ unchanged. The count is now Q₂ = 0, Q₁ = 1 and Q₀ = 0, i.e. 010.

The falling edge *ef* of the third clock pulse to FF₀ changes Q₀ from 0 to 1 again but the rising edge *EF* does not switch FF₁ leaving Q₁ = 1, Q₂ = 0 and the count at 011. The action thus ripples along the flip-flops, each one waiting for the previous flip-flop to supply a falling edge at its clock input before changing state.

Quiz: sketch the arrangement of a 4-bit binary up ripple counter

Synchronous counter

The counter described above is a ripple or asynchronous type. In a synchronous counter all flip-flops are clocked simultaneously and the propagation delay time is much less than for a ripple counter with a large number of flip-flops where there is the risk of a 'race condition'. Synchronous counters are therefore used for high-speed counting but their circuits are more complex than those of ripple types. Figure 11.8 shows the arrangement of a 2-bit Synchronous counter.

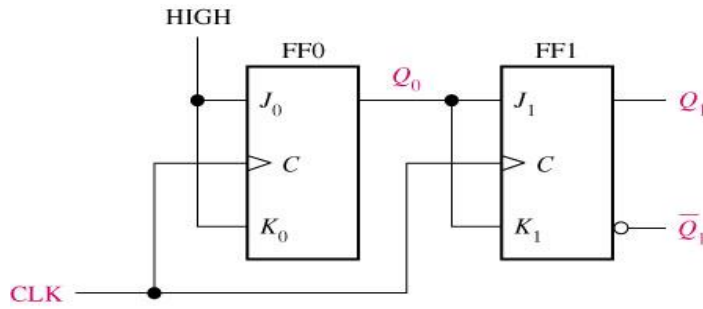


Figure 11.8 a 2-bit Synchronous counter

11.5.2 Binary down-counter

In a binary down-counter, the count decreases by one for each clock pulse, so if it is a 3-bit counter, it would count from 111 down to 000. To convert the up counter in Figure 11.6 into a binary down counter, the \bar{Q} output (instead of the Q) of each flip-flop is coupled to the CK input of the next, as shown in Figure 11.9. The count is still given by the Q outputs.

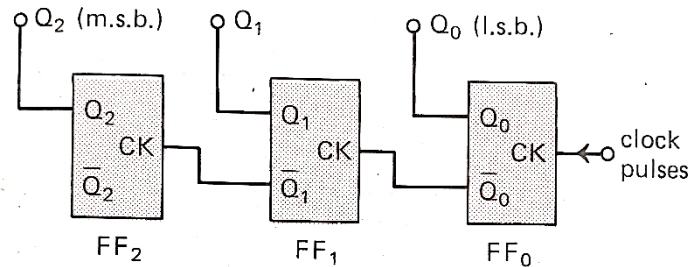


Figure 11.9: A 3-bit binary down counter.

Quiz: is the counter in figure 11.9 synchronous or asynchronous?

Decade counter

A four-bit binary up-counter, modified as in Figure 11.10, acts as a modulo-10 counter, and counts up from 0 to 9 before resetting. When the count is 1010 (decimal 10), $Q_3 = 1$, $Q_2 = 0$, $Q_1 =$

1 and $Q_0 = 0$ and since both inputs to the AND gate are 1s (i.e. Q_3 and Q_1), its output is 1 and this resets all the flip-flops to 0 (otherwise it would be a modulo-16 counter counting from 0 to 15).

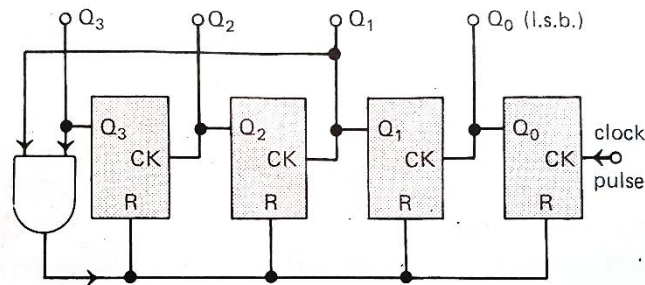


Figure 11.10: A decade counter

Summary

- Clock pulses are supplied by some form of pulse generator
- The two types of clocking or triggering operation are level triggering and edge triggering
- Sequential circuits can be clock driven, event driven or pulse driven
- Registers are collections of multiple flip-flops arranged in a group with a common function to store bits
- The four modes of operation of shift registers are SISO, SIPO, PIPO, PISO
- Counters are digital circuits that count either upwards or downwards.

Post-Test

- 1 Differentiate between *synchronous* and *asynchronous* counters.
- 2 What is referred to as the *modulo* of a counter?
- 3 List the four modes of operation of shift registers.
- 4 With suitable illustrations, differentiate between edge-triggered and level-triggered clock pulses
- 5 Sketch the diagram of a modulo-13 binary-up asynchronous counter
- 6 Walter needs to hold and transfer the data $X=1011_2$ from one point in a digital circuit he designed to another. He has the option of using any of the 4 modes of data transfer for shift registers. He wants to make this transfer as fast as possible so he designs four 4-bit shift register. (i.e. one for each mode.)
 - i. How many flip-flops did he use in each of the design?
 - ii. By tabulating the 4 modes and how long (clock pulses) it will take to read all his data, advise him on the best mode to use for his final implementation.

References

Essential Electronics

Duncan, Tom. 1997, Electronics for Today and Tomorrow. Spain: John Murray.

www.electronics-tutorials.ws

Mark Balch, 2003, Complete Digital Design, A Comprehensive Guide to Digital Electronics and
Computer System Architecture, McGraw-Hill

William E. Wickes, 1968, Logic Design with Integrated Circuits

PROPERTIES OF DLC UI, IBADAN

LECTURE TWELVE

COMPUTER CODES

Introduction

In the preceding lectures, we have studied how the binary digits are generated, represented and stored. We are all too familiar with the binary coding system, however the binary coding system is not the only way data is represented or manipulated in the computer system or digital devices. There are various other computer codes used for different purposes. Some of these will be studied in this lecture.

Objectives

At the end of this lecture, you should be able to:

1. State characteristics of a weighted coding system and give examples
2. State characteristics of unweighted coding system
3. Convert a decimal number to BCD and vice versa
4. Convert a decimal number to XS-3 and vice versa
5. Convert a decimal number to Gray code

12.1 Codes

A code is a systematic way of representing data and sometimes, information. Digital devices, especially computers use a variety of codes and different points to represent, manipulate or correct data. These various codes have different features and can be characterised as follows:

12.1.1. Weighted Codes

Weighted codes obey a positional weight system (or more specifically, positional numbering system) where a specific *weight* is assigned to each *position* of the number. This means that value of each digit depends on the position of that digit. Remember the Hundreds Tens and Units (H T U) we were taught in primary school? For example, given the decimal number 679, in that system we were taught that the value of 6 (6 hundred) is higher than the value of 9 (9 unit). Other example of weighted codes are Binary and BCD. Weighted codes are used, in Data manipulation during arithmetic operation, for input/output operations in digital circuits, to represent the decimal digits in calculators, volt meters etc.

12.1.2. Non-weighted Codes

Non-weighted (un-weighted) codes do not obey positional weight principle, therefore positional weights are not assigned to the digits of the number. In these codes, the digit value does not depend upon their position i.e., each digit position within the number is not assigned fixed value. Example include Excess-3 code, Gray code. Non weighted codes are used, to perform certain arithmetic operations, for error detecting purposes, to mention a few.

12.1.3. Reflective Codes

A code is said to be reflective when it's code for 9 is the complement of it's code for 0. Likewise, its code for 8 is the complement of the code for 1, it's code for 7 is the complement of it's code for 2, it's code for 6 is the complement of it's code for 3 and finally, it's code for 5 is the complement of it's code for 4. The excess-3 (XS-3) code is an example.

12.1.4. Sequential Codes

A code is said to be sequential when each succeeding code is one binary number greater than preceding code. Example Binary, XS-3

12.1.5. Alphanumeric Codes

These are codes designed to represent numbers as well as alphabetic characters. They can also be used to represent symbols as well as instructions. They include ASCII, EBCDIC

12.1.6. Error Detecting and Correcting Codes

These are codes used to detect and correct digital transmission errors, because these special type of codes are capable of detecting and correcting the errors. Examples include: Parity code, Hamming code. Error Detecting and Correcting Codes will be discussed in the next lecture.

Now that we know the different features of computer codes, we will study a few of the examples of these codes.

12.2. Binary Codes

By now you are very familiar with the binary coding system, because we have been using this system through this course. In summary, you know that the binary coding system, also known as base two, uses only two digits 0 and 1. It is possible to represent 2^n different messages in a purely binary code of n bits. The binary code is a straightforward direct conversion of a decimal number (I trust you know how to do the conversion) to the binary. The binary coding system is the most common code used in digital devices because it is a systematic arrangement of the digits. It is a weighted code.

Quiz: Convert 250_{10} to binary

Quiz: Convert 101101_2 to decimal

12.3 Binary-Coded Decimal (BCD)

The next most common code, in digital devices, is the binary-coded decimal code. The need for a code for each of the 10 decimal digits, 0 through 9, arises often, and there is a large variety of

codes from which to choose. In computer and digital systems, binary-coded decimal (BCD) is an encoding for decimal numbers in which each digit is represented by its own binary sequence. It allows for easy conversion to decimal digits for printing or display and faster decimal calculations. In BCD, a digit is usually represented by four bits which, in general, represent the values or digits or characters 0-9. To encode a decimal number in, BCD, each decimal digit is stored in a four-bit nibble. Table 12.1 shows this encoding.

Table 12.1: BCD equivalent of decimal digits

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

From table 12.1 we can see that the BCD code makes use of the first 10 combinations of the 4-bit binary arrangement, that is 4-bit binary equivalent of the decimal numbers 0 to 9. This provides a weighted code in which the LSB has the weight of $2^0 = 1$, the next bit has a weight of $2^1 = 2$, the third bit has the weight of $2^2 = 4$, and the MSB has the weight of $2^3 = 8$. That is why some author refer to it as the 8-4-2-1 BCD code. Thus the decimal equivalent of the binary code is determined by the presence of 1's in the number sequence. From Table 12.1 we see that the decimal numbers 0 to 9 already have their codes. How then do we convert bigger decimal numbers to BCD? Simply express each decimal digit with its equivalent 4-bit BCD code. Hence, the BCD encoding for the decimal number 256 would be:

2	5	6
0010	0101	0110

Giving $256_{10} = 001101010110_{BCD}$

Quiz: what is the BCD code for the decimal number 5682

This means that the smallest number in BCD is 0000 which is 0 and largest is 1001 which is 9. After which decimal numbers with two or more digits will be expressed by combinations, for example $10 = 0001\ 0000$ and this is known as packed BCD.

The conversion of a binary BCD number to its decimal equivalent is done in the reverse manner, as illustrated by the following: Convert $1001001100100111_{BCD} = ?_{10}$

1001	0011	0010	0111
9	3	2	7

Giving 9327_{10}

Quiz: Convert the following BCD code to its decimal equivalent.

001101110000

Since most computers store data in eight-bit bytes, there are two common ways of storing four-bit BCD digits in those bytes:

1. Each digit is stored in one byte, and the other four bits (the most significant 4 bits) are used as zone bits which are then set to all zeros, all ones (as in the EBCDIC code), or to 0011 (as in the ASCII code)
2. Two digits are stored in each byte.

As earlier written, BCD is very common in electronic systems (digital wrist watches, Microwave displays, washing machine displays, ...) where a numeric value is to be displayed, especially in

systems consisting solely of digital logic, and not containing a microprocessor. By utilising BCD, the manipulation of numerical data for display can be greatly simplified by treating each digit as a separate single sub-circuit.

Advantages of BCD codes include (1) it is similar to decimal number system, (2) we need to remember binary equivalents of decimal numbers 0 to 9 only, (3) conversions from decimal to BCD or BCD to decimal is very simple and no calculation is needed! However, its drawbacks are the increased complexity of circuits needed to implement mathematical operations and it is less efficient than binary, since it needs more bits than in binary to store the same decimal number. Even though the importance of BCD has diminished, it is still widely used in financial, commercial, and industrial applications. Note that there are other forms of the weighted BCD codes, where the weights assigned are 6-3-1-1, 5-2-1-1, 7-4-2-1, 5-4-2-1.

12.4 Excess-3 code (XS-3)

Excess-3 code, is a non-weighted, reflective code used to express decimal numbers. Whereas the BCD code makes use of the first 10 of the 16 different combinations of four bits, the excess-3 code adds 3 to decimal number and then converts to binary as shown in table 12.2. This code is called a non-weighted code because the binary 1's do not represent decimal value. The advantage of this type of code is that at least one 1 is present in all states, providing an error-detection ability.

Table 12.2 Excess-3 Code

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

It is another fundamental binary code that is particularly significant for arithmetic operations as it overcomes the shortcomings encountered while using the 8421 BCD code. The Excess-3 codes presented in Table 12.2 for a given decimal number is determined by adding '3' to each decimal digit in the given number and then replacing each digit of the newly found decimal number by its four bit binary equivalent. For example, XS-3 code of 34_{10} is obtained as follows:

$$\begin{array}{r}
 3 4 \\
 +3 +3 \\
 \hline
 6 7 \\
 0110 0111
 \end{array}$$

Thus, XS3 code of 34 is 0110 0111.

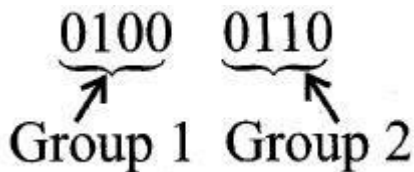
The Excess-3 code is also a reflective code, so it is self-complementing. For example, the XS-3 code for decimal 6 is 1001. The 1's complement of 1001 is 0110, which is the XS-3 code for decimal 3, and 3 is the 9's complement of 6. This property of Excess-3 code makes it useful in some arithmetic operations. Here is another example, the XS-3 code for decimal number 27 is 01011010.

$$\begin{array}{r}
 \underbrace{2}_{0101} \quad \underbrace{7}_{1010} \\
 0101 \quad 1010
 \end{array}$$

Quiz: What is the XS-3 code for 657_{10}

How do we convert a sequence of binary digits in XS-3 to its decimal equivalent? There are two ways. But the first thing to do is to first split the number into groups of four-bits, starting from the LSB for whole numbers or the radix point for floating point numbers. Then (1) look up the decimal value for each group from Table 12.2 or (2) By

subtracting 0011 (3) from each four-bit group, which will give us the 8-4-2-1 BCD equivalent of the given XS-3 code, this can then be converted into the equivalent decimal number. For example; Let us suppose we want to determine the decimal equivalent for the XS-3 code 1000110. First we make group of 4 bits starting from radix point.



Subtracting 0011 from each group, we obtain the new number as 0001 0011. Its decimal equivalent is 13. Therefore, $1000110_{XS-3} = 13_{10}$

Quiz: What is the decimal equivalent of 10110100011_{XS-3}

12.5 Gray Code

The Gray code is an unweighted code that was designed by *Frank Gray* at Bell Labs in 1953. It is also reflective in nature because the lower-order bit sequence is the same when starting from the middle of the count and progressing in either direction. It belongs to a class of codes called the minimum change code. The successive coded characters never differ in more than one-bit. Owing to this feature, the maximum error that can creep into a system using the binary gray code to encode data is much less than the worst -case error encountered in case of straight binary encoding.

Since the Gray code is an unweighted code, the gray code is not suitable for arithmetic operations but finds applications in input/output devices, some of which are: analog-to-digital converters and designation of rows and columns in Karnaugh map. One can easily remember the gray codes because a three-bit gray code can be obtained by merely reflecting the two-bit

code about an axis at the end of the code and assigning a third-bit as 0 above the axis and as 1 below the axis (see figure 12.1). The reflected gray code is nothing but code written in reverse order. By reflecting three-bit code, a four-bit code may be obtained.

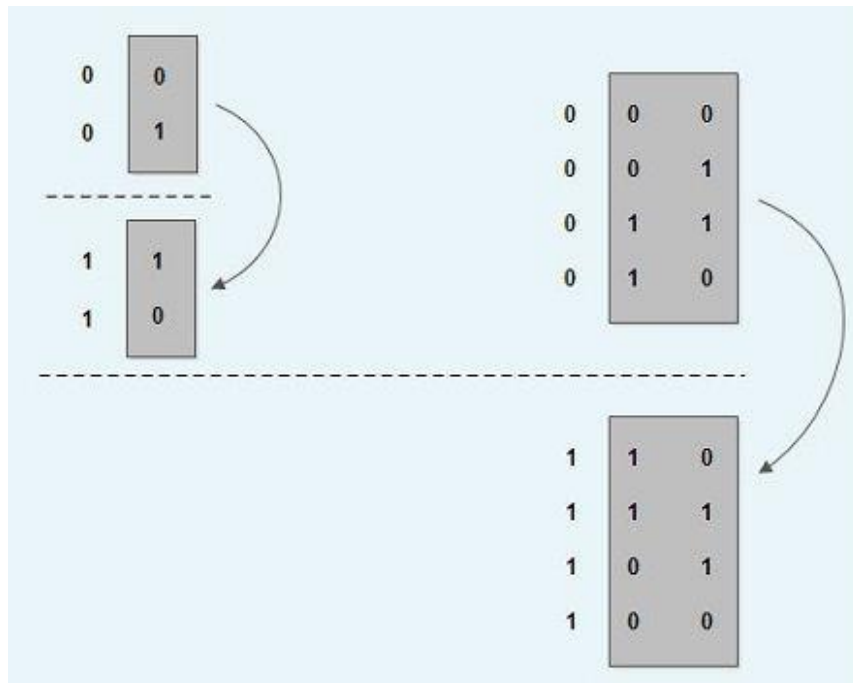


Figure 12.1: The process of obtaining 3-bit gray code by reflecting 2-bit gray code

Table 12.4 3-bit Gray codes

Decimal	3-bit Gray Codes		
0	0	0	0
1	0	0	1
2	0	1	1
3	0	1	0
4	1	0	0
5	1	0	1
6	1	1	1
7	1	1	0

Quiz: Generate the table for the 4-bit Gray codes.

Using the table you just generated, let us take a few examples. The four-bit gray code for decimal number 39 is 00101101

$$\underbrace{\quad\quad\quad}_3 \quad \underbrace{\quad\quad\quad}_9$$

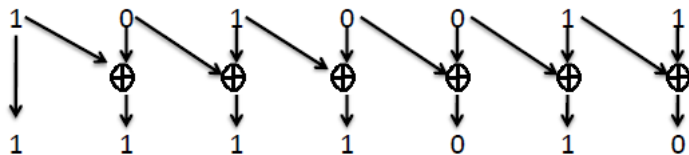
$$0010 \quad 1101$$

Similarly, gray code for $(825.1)_{10} = (1100 \ 0011 \ 0111.0001)_{\text{Gray code}}$

How can we convert from Binary to Gray code? The steps are below:

- 1: Write MSB of given Binary number as it is.
- 2: *Ex-OR* this bit with the next bit of that binary number and write the result.
- 3: *Ex-OR* each successive sum until LSB of that binary number is reached.

Example: Convert $(1010011)_2$ to its equivalent Gray code.

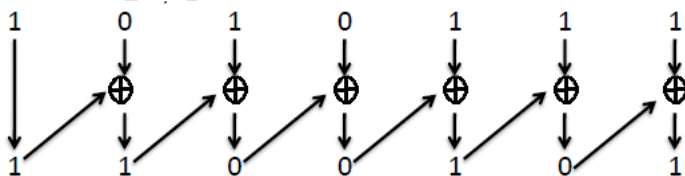


Therefore $(1010011)_2 = (1111010)_{\text{Gray}}$

And to convert from Gray to Binary, we use the following steps:

- 1: Write MSB of given Binary number as it is.
- 2: *Ex-OR* this bit with next bit of that binary number and write the result.
- 3: Continue this process until LSB of that binary number is reached.

Example: Convert $(1010111)_{\text{Gray}}$ to its equivalent Binary number



Therefore $(1010111)_{\text{Gray}} = (1100101)_2$

Summary

- A code is a systematic way of representing data and computer codes can be classified by their characteristics into weighted codes, non-weighted codes, reflective codes, sequential codes, alphanumeric codes and error detecting and correction codes.
- The most common code used by digital devices to store data are binary codes.
- Binary coded decimal codes are used mainly for displays.
- The other codes are Excess-3 code and gray codes.

Post-Test

1. Convert following decimal numbers to BCD:
(a) 394 (b) 4954 (c) 73291
2. Convert following BCD codes to decimal equivalent:
(a) 1001 1000 (b) 0001 0100 0110 (c) 0111 0011 0101
3. Convert following binary numbers to BCD codes: (Hint: convert to decimal first)
(a) 1100 (b) 10001 (c) 1010101
4. Convert following BCD codes to binary equivalent: (Hint: convert to decimal first)
(a) 0010 1000 (b) 1001 0111 (c) 1000 0000
5. Obtain XS-3 equivalent of following numbers:
(a) 437_{10} (b) 146_{10} (c) $0111\ 1000_{BCD}$ (e) 101010_2 (hint: first convert to decimal)
6. Convert the binary codes in question 3 to gray codes.

References

Amraja Shivkar, 2018, lecture notes on CODES, <http://techgurutechnolic.weebly.com/>

Dinesh Thakur, 2018, lecture notes on Excess-3 Codes.

<https://mywbut.com>

William E. Wickes, 1968, Logic Design with Integrated Circuits

LECTURE THIRTEEN

ERROR DETECTION AND CORRECTION

Introduction

In the previous lecture, we studied some regular codes used by the computer and digital devices to store and retrieve data. Regardless of the coding method used, no communications channel or storage medium can be completely error-free. It is physically impossible. As more bits are packed per square millimetre of storage, flux densities increase. Error rates increase in direct proportion to the number of bits per second transmitted, or the number of bits per square millimetre of magnetic storage.

Objectives

At the end of this lecture, you should be able to:

13.1 Digital Error

When a digital information is transmitted, it may not be received correctly by the receiver, because there may be error due to electrical disturbance of circuit, which is also called noise. This noise may force a logic level '1' to change to a logic level '0' or vice versa, thereby altering the original information that was sent. For example, with binary number transmission, usually single-bit errors may occur such that the data **0010** may be erroneously transmitted as **0011**, or **0000**, or **0110**, or **1010**. The error, therefore, has to be detected and corrected or the sender may be asked to resend the data. There are various codes and methods used to detect and correct such errors, some of these will be briefly treated. However, bear in mind, as you study the various methods, that it is impossible to create an error-free medium, it is also impossible to detect or correct 100% of all errors that could occur in a medium.

13.2 Parity Bit

The parity bit error detection method is best suited for detection of single-bit errors in a predefined number of binary digits transmitted or stored, as in the example under section 13.1.

Parity is the most basic of all error detection methods and it is easy to implement in simple devices. In using parity for detection of error an extra bit called the parity bit is attached to code, usually as the most significant bit. This bit is used to specify the number of '1s' in the transmitted/ received data. Therefore it makes sense to use either *even* parity or *odd* parity.

For example: If a 7 bit data 1001110 is to be transmitted, then using even parity code, it will be transmitted as the 8 bit word 01001110 or using odd parity code, it will be transmitted as 11001110.

Notice that the parity is decided by the extra MSB (parity bit) which is introduced in original data. So if the total number of '1's in the transmitted/ received word is even then the parity is even and if total number of '1's in transmitted/ received word is odd then the parity is odd.

Table 13.1 shows the BCD codes for 0 to 9 with odd or even parity bits.

Table 13.1 BCD code with parity bits

BCD code				BCD code with even parity					BCD code with odd parity				
N ₄	N ₃	N ₂	N ₁	P	N ₄	N ₃	N ₂	N ₁	P	N ₄	N ₃	N ₂	N ₁
0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	1	1	0	0	0	1	0	0	0	0	1
0	0	1	0	1	0	0	1	0	0	0	0	1	0
0	0	1	1	0	0	0	1	1	1	0	0	1	1
0	1	0	0	1	0	1	0	0	0	0	1	0	0
0	1	0	1	0	0	1	0	1	1	0	1	0	1
0	1	1	0	0	0	1	1	0	1	0	1	1	0
0	1	1	1	1	0	1	1	1	0	0	1	1	1
1	0	0	0	1	1	0	0	0	0	1	0	0	0
1	0	0	1	0	1	0	0	1	1	1	0	0	1

How then is error detected using parity bit? Suppose that a 7 bit data, say 1011010 is to be transmitted with even parity, then it will be transmitted as 01011010 where the MSB which is the parity bit will be kept as 0 in order to maintain even parity of transmitted word.

If the data is received as 01011010, that is without error then the parity still remains even.

Hence, the data is declared as correct word.

However, what if it is received as 01111010, that is with 1 error then parity becomes odd, therefore it is declared as incorrect word.

As earlier stated, this method of error detection is best suited for single-bit error. One of the disadvantages of this method is that if the data is received with 2 errors, for instance if the 7-bit data above were transmitted as 01110010 then the parity still remains even and declared as correct word even in spite of being incorrect (see figure 13.1). Another disadvantage is that this method cannot detect *where* exactly the error has occurred.

	P	Message bits						Parity	Receivers decision	
Transmitted Code	0	1	0	1	1	0	1	0	Even	-
Received Code (with 0 error)	0	1	0	1	1	0	1	0	Even	Correct word
Received Code (with 1 error)	0	1	1	1	1	0	1	0	Odd	Correct word
Received Code (with 2 errors)	0	1	1	1	0	0	1	0	Even	Correct word

Figure 13.1 Parity bit error detection

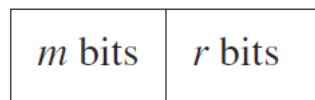
Quiz: Generate the correct bits that would be transferred if the following data are using (1) even parity (2) odd parity.

- (A) 0101100 (B) 1011 (C) 0000111

13.2 Hamming Codes

In digital data communications, it is adequate to have the ability to detect errors in the transmitted data and when a communication device determines that a message contains an erroneous bit, all it has to do is request retransmission. Storage systems and memory do not have this luxury. Consequently, storage devices and memory must therefore have the ability to not only detect but to correct a reasonable number of errors. Another effective error recovery code is the Hamming code. *Hamming codes* are an adaptation of the concept of parity, whereby error detection and correction capabilities are increased in proportion to the number of parity bits added to an information word.

Hamming codes are used in situations where random errors are likely to occur, so in our following discussion, we present Hamming codes in the context of memory bit error detection and correction. As earlier mentioned, Hamming codes use parity bits, also called *check bits* or *redundant bits*. The memory word itself consists of m bits, but r redundant bits are added to allow for error detection and/or correction. Thus, the final word, called a *code word*, is an n -bit unit containing m data bits and r check bits. There exists a unique code word consisting for $n = m + r$ bits for each data word as follows:



The number of bit positions in which two code words differ is called the *Hamming distance* of those two code words. For example, if we have the following two code words:

1	0	0	0	1	0	0	1
1	0	1	1	0	0	0	1
		*	*	*			

we see that they differ in 3 bit positions, so the Hamming distance of these two code words is 3 (we will soon discuss how to create code words). The Hamming distance between two code

words is important in the context of error detection. If two code words are a Hamming distance d apart, d single-bit errors are required to convert one code word to the other, which implies this type of error would not be detected. Therefore, if we wish to create a code that guarantees detection of all single-bit errors (an error in only 1 bit), all pairs of code words must have a Hamming distance of at least 2.

The Hamming distance of a code must be at least $2k + 1$ in order for it to be able to correct k errors. Code words are constructed from information words using r parity bits. Most often than not, the 7-bit Hamming code is commonly used, but this concept can be extended to any number of bits. The arrangement of the data as well as the party bits for the 7-bit Hamming code is shown in figure 13.2

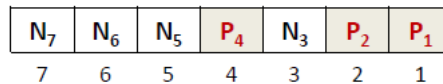


Figure 13.2: Arrangement of 7-bit Hamming code.

In figure 13.2 we note that the portions labelled as N represent the original number/data bits, while P denotes the Parity bits. The humming Parity bits are introduced at each 2^n bit where $n = 0, 1, 2, 3...$ therefore the:

1st parity bit is at $2^0 = 1$ i.e., 1st place and denoted by P₁

2nd parity bit is at $2^1 = 2$ i.e., 2nd place and denoted by P₂

3rd parity bit is at $2^2 = 4$ i.e., 4th place and denoted by P₄

4th parity bit will be at $2^3 = 8$ i.e., 8th place. But since we have only 7 bit code it cannot have this parity bit. So 7 bit Hamming code has only 3 parity bits P₁, P₂, P₄.

Let us take an example of how to construct a 7-bit hamming code for a transmitted data.

A bit word 1 0 1 1 is transmitted. Construct the even parity 7-bit Hamming Code for this data

- **Step 1:** fill the data bits in their respective places (N_7, N_6, N_5, N_3) leaving parity bit places empty as shown

N_7	N_6	N_5	P_4	N_3	P_2	P_1
1	0	1		1		
7	6	5	4	3	2	1

- **Step 2:** Decide P_1 :

P_1 checks parity of bit 1, 3, 5, 7 that means it checks on P_1, N_3, N_5, N_7

N_7	N_6	N_5	P_4	N_3	P_2	P_1
1	0	1		1		1
7	6	5	4	3	2	1

Set $P_1 = 1$ to have even parity for Bits 1,3,5,7

- **Step 3:** Decide P_2 :

P_2 checks parity of bit 2, 3, 6, 7 that means it checks on P_2, N_3, N_6, N_7

N_7	N_6	N_5	P_4	N_3	P_2	P_1
1	0	1		1	0	1
7	6	5	4	3	2	1

Set $P_2 = 0$ to have even parity for Bits 1,3,6,7

- **Step 4:** Decide P_4 :

P_4 checks parity of bit 4, 5, 6, 7 that means it checks on P_4, N_5, N_6, N_7

N_7	N_6	N_5	P_4	N_3	P_2	P_1
1	0	1	0	1	0	1
7	6	5	4	3	2	1

Set $P_4 = 1$ to have even parity for Bits 1,3,5,7

Hence the required 7 bit Hamming code is 1 0 1 0 1 0 1

13.3 Cyclic Redundancy Check

Checksums are self-checking codes that will quickly indicate whether the preceding digits have been misread. Cyclic redundancy check (CRC) is a type of checksum used primarily in data communications that determines whether an error has occurred within a large block or stream of information bytes. The larger the block to be checked, the larger the checksum must be, to provide adequate protection. CRCs are a type of systematic error detection scheme, meaning that the error-checking bits are appended to the original information byte. The group of error-checking bits is called a syndrome. The original information byte is unchanged by the addition of the error-checking bits. The implementation and calculation of the CRC method for error detection is beyond the scope of this class.

Summary

- When a digital data is transmitted, it may be altered by noise and as such introduce error in the data.
- When using parity bit for error detection, an extra bit is added as the MSB of each piece of data. The parity code can either specify even or odd parity. For even parity, the additional bit supplied is to make total number of '1's even while for odd parity, the additional bit supplied is to make total number of '1's odd.
- The Hamming code is another error detection and correction code that constructs the code using the data being transmitted.

Post-Test

1. Generate the correct bits that would be transferred if the following data are transmitted using (1) even parity (2) odd parity.

(A) 0101111 (B) 101100 (C) 0010111

2. Construct the even parity 7-bit Hamming code for the following data

(A) 1011 (B) 1111 (C) 0010

References

Amraja Shivkar, 2018, lecture notes on CODES, <http://techgurutechnolic.weebly.com/>

Dinesh Thakur, 2018, lecture notes on Excess-3 Codes.

<https://mywbut.com>

William E. Wickes, 1968, Logic Design with Integrated Circuits