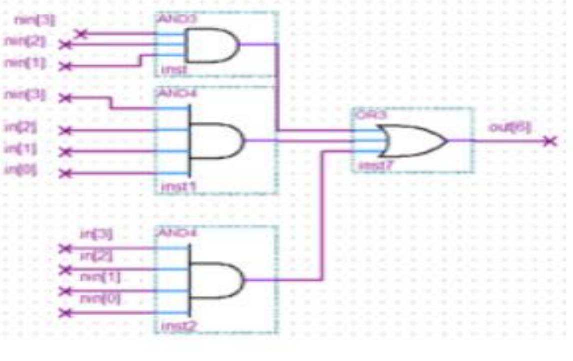
Jimma University
College of Natural Sciences
Department of Physics

Lecture Notes : Electronics I (Phys 2062)

Chapter Six: Digital Circuits

By: Mrs. Hiwot Tegegn (lecturer)



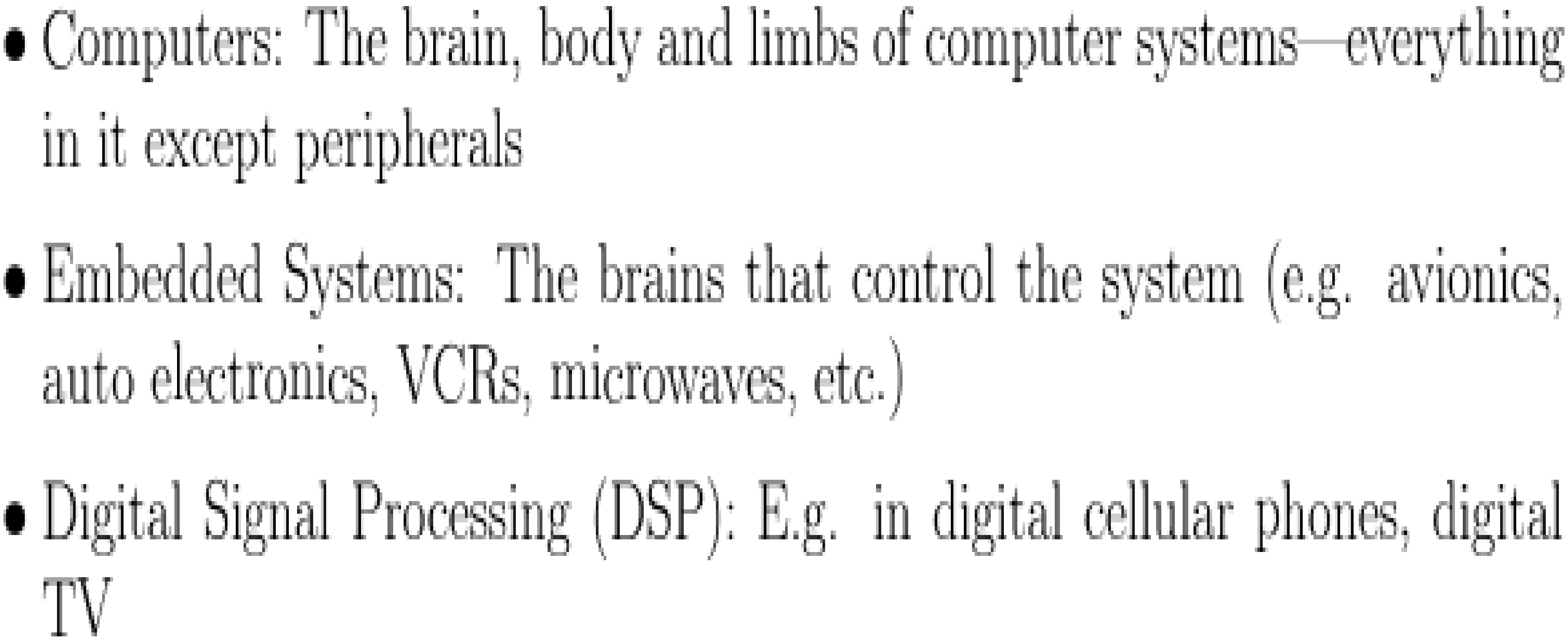
Outline of the Chapter

- The junction field effect transistor (JFET), JFET Common Source Amplifier, JFET
- Common Drain amplifier
- Insulated-Gate Field Effect Transistor. Power,
- Multiple Transistor Circuit

Chapter Six: Digital Circuits

- In analog electronics, voltage is a continuous variable. This is useful because most physical quantities we encounter are continuous: sound levels, light intensity, temperature, pressure, etc.
- Digital electronics, in contrast, is characterized by only two distinguishable voltages. These two states are called by various names: on/off, true/false, high/low, and 1/0
- In practice, these two states are defined by the circuit voltage being above or below a certain value
- By converting continuous analog signals into a finite number of discrete states, a process called *digitization*,
- This is the primary advantage of digital electronics: it is relatively immune to the noise that is ubiquitous in electronic circuits
- Although digital circuits have excellent noise immunity, they also are limited to producing only two levels. This does not appear to be very helpful in representing the continuous signals we so frequently encounter.
- The explosion in digital techniques and technology has been made possible by the incredible increase in the density of digital circuitry, its robust performance, its relatively low cost, and its speed. The requirement of using many bits in reproduction is no longer an issue:
- This circuitry is based upon the transistor, which can be operated as a switch with two states. Hence, the digital information is intrinsically *binary*. So in practice, the terms digital and binary are used interchangeably. In the following sections we summarize some conventions for defining the binary states and for doing binary arithmetic.

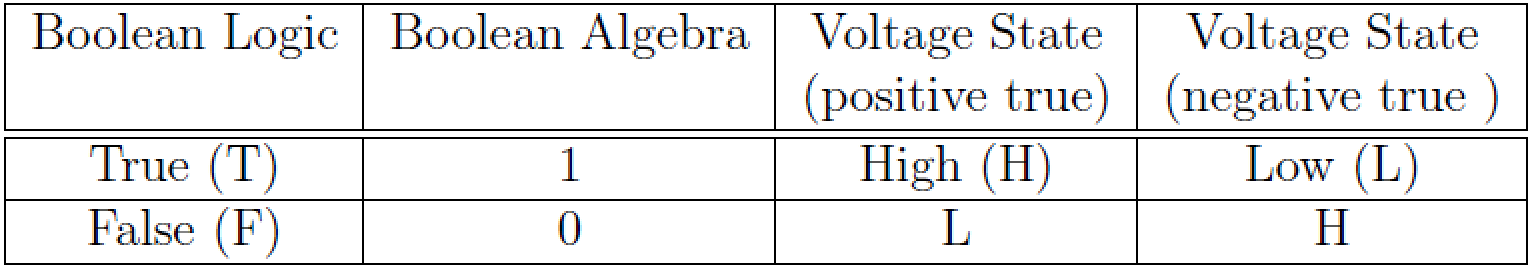
Application of logic circuit



- Computers: The brain, body and limbs of computer systems—everything in it except peripherals
- Embedded Systems: The brains that control the system (e.g. avionics, auto electronics, VCRs, microwaves, etc.)
- Digital Signal Processing (DSP): E.g. in digital cellular phones, digital TV

Binary Logic States

- Each digit in binary is a 0 or a 1 and is called a *bit*, which is an abbreviation of *binary digit*.
- There are several common conventions for representation of numbers in binary.
- The most familiar is *unsigned binary*
- The following table attempts to make correspondences between conventions for defining



Boolean Logic	Boolean Algebra	Voltage State (positive true)	Voltage State (negative true)
True (T)	1	High (H)	Low (L)
False (F)	0	L	H

Numbers systems

Binary Numbers

- It is important to be able to represent numbers in digital logic circuits
  - for example, the output of a analogue to digital converter (ADC) is an n-bit number, where n is typically in the range from 8 to 16
- Various representations are used, e.g.,
  - unsigned integers
  - 2's complement to represent negative numbers

Numbers systems

Binary Numbers

- Binary is base 2. Each digit (known as a bit) is either 0 or 1.
- Consider these 6-bit unsigned numbers

1	0	1	0	1	0	$= 42_{10}$
32	16	8	4	2	1	Binary coefficients
2^5	2^4	2^3	2^2	2^1	2^0	
MSB					LSB	

0	0	1	0	1	1	$= 11_{10}$
32	16	8	4	2	1	Binary coefficients
2^5	2^4	2^3	2^2	2^1	2^0	
MSB					LSB	

MSB – most significant bit

LSB – least significant bit

Number systems

Octal: Base 8

- We have seen base 2 uses 2 digits (0 & 1), not surprisingly base 8 uses 8 digits : 0, 1, 2, 3, 4, 5, 6, 7.

0	5	2	$= 42_{10}$
64	8	1	Octal coefficients
8^2	8^1	8^0	
MSB		LSB	

- To convert from decimal to base 8 either use successive division, i.e.,

$$42/8 = 5 \quad \text{remainder} = 2$$

$$5/8 = 0 \quad \text{remainder} = 5$$

- So the answer is 52_8 (reading upwards)

Number systems

Octal: Base 8

- Or alternatively, convert to binary, divide the binary number into 3-bit groups and work out the octal digit to represent each group. We have shown that

$$42_{10} = 101010_2$$

- So,

$$\begin{array}{|ccc|ccc|} 1 & 0 & 1 & 0 & 1 & 0 \\ \hline \multicolumn{3}{|c|}{5} & \multicolumn{3}{c|}{2_8} \end{array} = 42_{10}$$

MSB LSB

Number systems

Hexadecimal: Base 16

- For base 16 we need 16 different digits. Consequently we need new symbols for the digits to represent 10-15

$$1010_2 = 10_{10} = A_{16} \qquad 1101_2 = 13_{10} = D_{16}$$

$$1011_2 = 11_{10} = B_{16} \qquad 1110_2 = 14_{10} = E_{16}$$

$$1100_2 = 12_{10} = C_{16} \qquad 1111_2 = 15_{10} = F_{16}$$

$$0 \quad 2 \quad A_{16} \quad = 42_{10}$$

256	16	1	Hex coefficients
16^2	16^1	16^0	
MSB		LSB	

Number systems

Hex: Base 16

- To convert from decimal to base 16 use either use successive division by 16, i.e.,

$$42/16=2 \quad \text{remainder}=A$$

$$2/16=0 \quad \text{remainder}=2$$

- So the answer is $2A_8$ (reading upwards)

Number systems

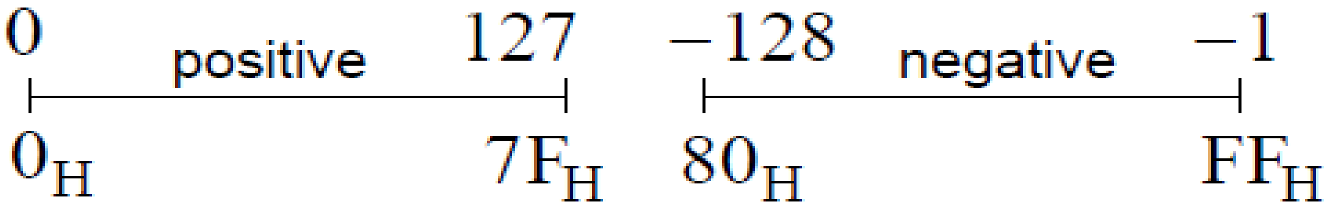
Negative numbers

- So far we have only been able to represent positive numbers. For example, we have seen an 8-bit byte can represent from 0 to 255, i.e., $2^8 = 256$ different combinations of bits in a byte
- If we want to represent negative numbers, we have to give up some of the range of positive numbers we had before
  - A popular approach to do this is called *2's complement*

Number systems

2's Complement

- For 8-bit numbers:



- Note all negative numbers have the MSB set
- The rule for changing a positive 2's complement number into a negative 2's complement number (or vice versa) is:

Complement all the bits and add 1.

Logic Gates and Combinational Logic

- ❖ Logic gates are the building blocks of digital circuits
- ❖ Basic logic circuits with one or more inputs and one output are known as *gates*
- ❖ *Gates* are used as the building blocks in the design of more complex digital logic circuits

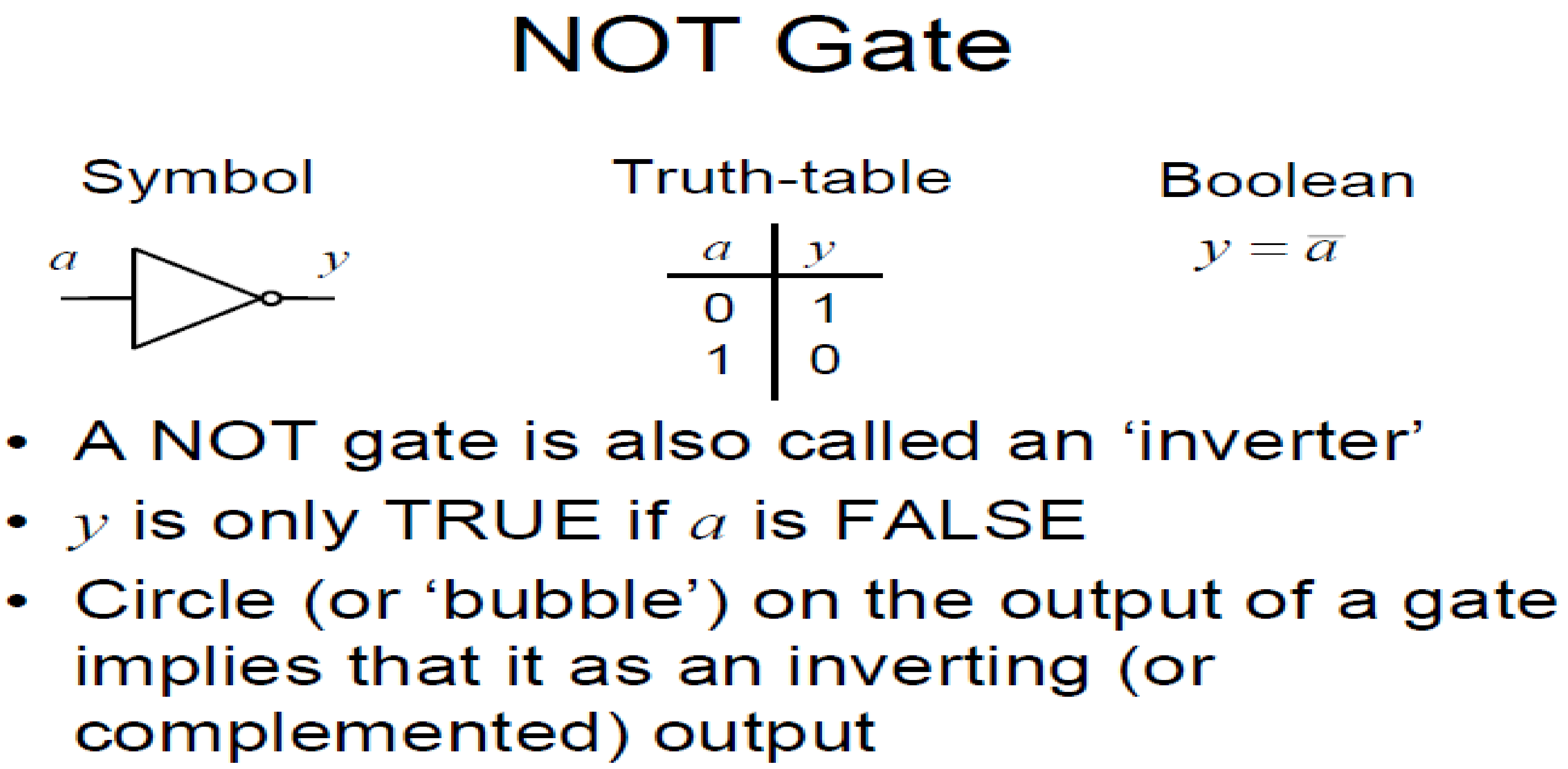
Representing Logic Functions

- ❖ There are several ways of representing logic functions:
  - ✓ Symbols to represent the gates
  - ✓ Truth tables
  - ✓ Boolean algebra

- ❖ The basic circuit element for manipulating digital signals is the *logic gate*

Logic Gates

- There are several types of logic gate, and each performs a particular logical operation on the input signals.
- The logical operation of the gate is defined by its *truth table* which gives the output state for all possible combinations of the inputs



NOT Gate

Symbol

Truth-table

a	y
0	1
1	0

Boolean

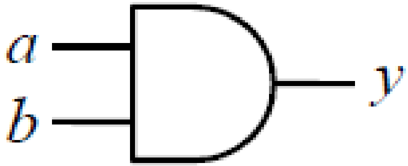
$y = \overline{a}$

- A NOT gate is also called an ‘inverter’
- y is only TRUE if a is FALSE
- Circle (or ‘bubble’) on the output of a gate implies that it as an inverting (or complemented) output

Logic Gates

AND Gate

Symbol



Truth-table

a	b	y
0	0	0
0	1	0
1	0	0
1	1	1

Boolean

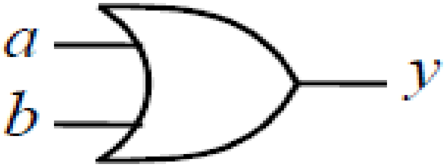
$y = a.b$

- y is only TRUE only if a is TRUE and b is TRUE
- In Boolean algebra AND is represented by a dot .

Logic Gates

OR Gate

Symbol



Truth-table

a	b	y
0	0	0
0	1	1
1	0	1
1	1	1

Boolean

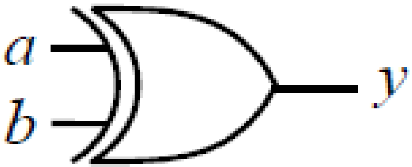
$y = a + b$

- y is TRUE if a is TRUE or b is TRUE (or both)
- In Boolean algebra OR is represented by a plus sign $+$

Logic Gates

EXCLUSIVE OR (XOR) Gate

Symbol



Truth-table

a	b	y
0	0	0
0	1	1
1	0	1
1	1	0

Boolean

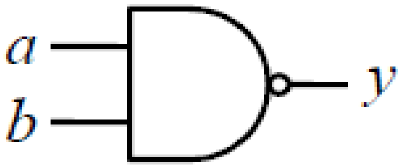
$y = a \oplus b$

- y is TRUE if a is TRUE or b is TRUE (but not both)
- In Boolean algebra XOR is represented by an $\oplus$ sign

Logic Gates

NOT AND (NAND) Gate

Symbol



Truth-table

a	b	y
0	0	1
0	1	1
1	0	1
1	1	0

Boolean

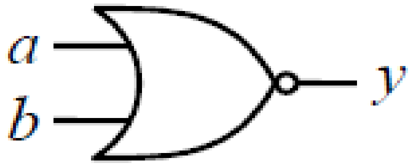
$$y = \overline{a.b}$$

- y is TRUE if a is FALSE or b is FALSE (or both)
- y is FALSE only if a is TRUE and b is TRUE

Logic Gates

NOT OR (NOR) Gate

Symbol



Truth-table

a	b	y
0	0	1
0	1	0
1	0	0
1	1	0

Boolean

$$y = \overline{a + b}$$

- y is TRUE only if a is FALSE and b is FALSE
- y is FALSE if a is TRUE or b is TRUE (or both)

Boolean Algebra

- An *algebra* is a statement of rules for manipulating members of a set.
- There are rules for doing mathematical manipulations with integers, real numbers, and complex numbers.
- There is also a special algebra for logical operations. It is called *Boolean algebra*.
- They consist of definitions for the AND, OR, and NOT (or inversion) operations, and several theorems
- Boolean algebra can be used to find alternative ways of expressing a logical function

Thank you