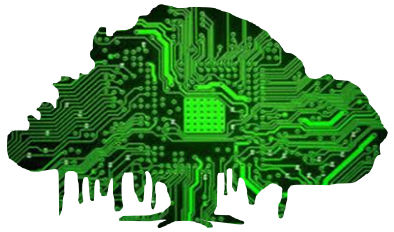


Lecture 4 – Binary Functions

Dr. Aftab M. Hussain,

Assistant Professor, PATRIoT Lab, CVEST

Chapter 2

Logic gates!

- Logic gates are electronic circuits that operate on one or more input signals to produce an output signal

- Electrical signals such as voltages or currents exist as analog signals having values over a given continuous range, say, 0 to 3 V, but in a digital system these voltages are interpreted to be either of two recognizable values, 0 or 1

- Voltage-operated logic circuits respond to two separate voltage levels that represent a binary variable equal to logic 1 or logic 0

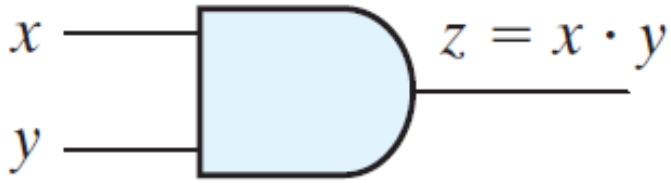
- For example, a particular digital system may define logic 0 as a signal equal to 0 V and logic 1 as a signal equal to 3 V

Logic gates

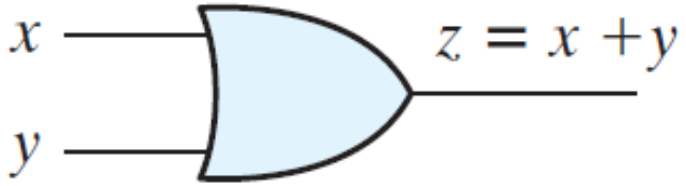
- In practice, each voltage level has an acceptable range
- The input terminals of digital circuits accept binary signals within the allowable range and respond at the output terminals with binary signals that fall within the specified range
- The intermediate region between the allowed regions is crossed only during a state transition
- For example, Arduino reads 3-5 V at its digital input pins as HIGH and 0-1.5 V as LOW
- Any desired information for computing or control can be operated on by passing binary signals through various combinations of logic gates, with each signal representing a particular binary variable (this is how microprocessors work)

Logic gates

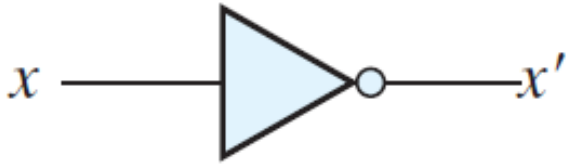
- We represent the logic gates for basic Boolean operations as shown
- Logic gates can have more than two inputs (expect for NOT gate of course!)



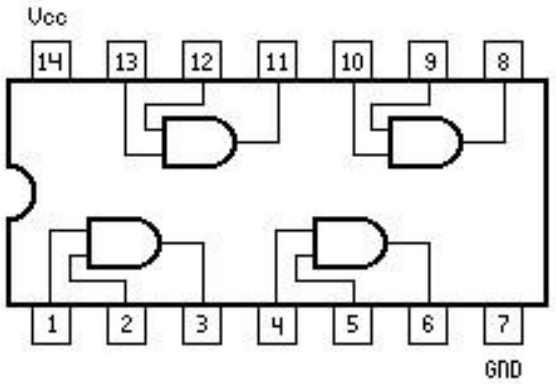
(a) Two-input AND gate



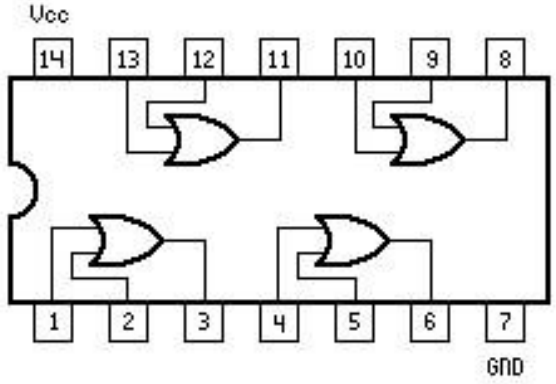
(b) Two-input OR gate



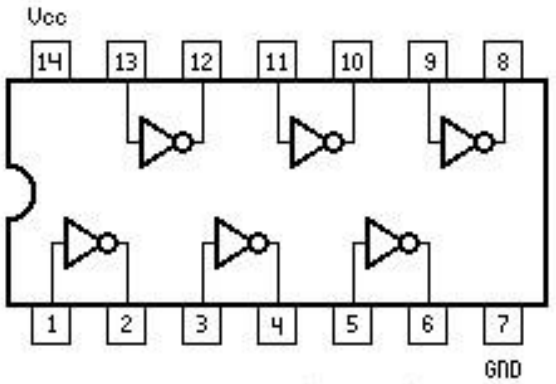
(c) NOT gate or inverter



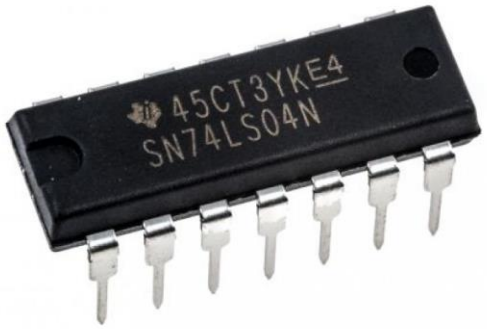
7408 (AND)



7432 (OR)

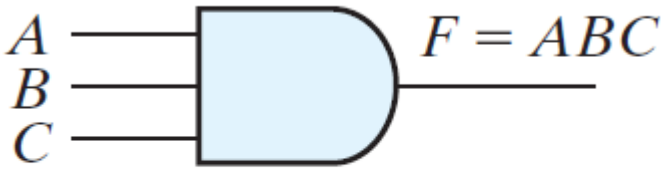


7404 (NOT)

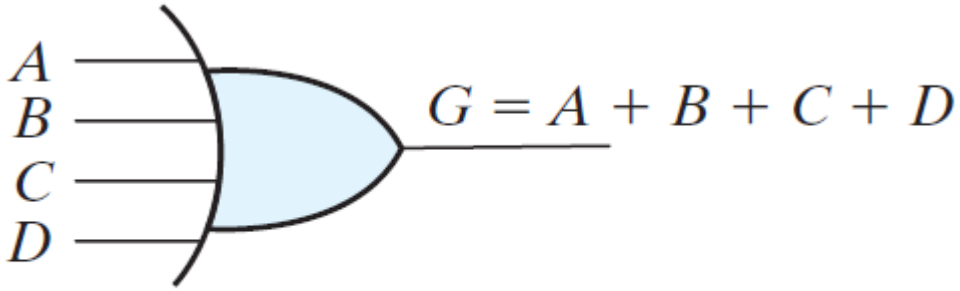


Logic gates

- AND and OR gates may have more than two inputs
- The three-input AND gate responds with logic 1 output if all three inputs are logic 1. The output produces logic 0 if any input is logic 0
- The four-input OR gate responds with logic 1 if any input is logic 1; its output becomes logic 0 only when all inputs are logic 0



(a) Three-input AND gate



(b) Four-input OR gate

Boolean functions

- In normal algebra, we define functions on real number variables using addition, subtraction, exponents, etc.
- A Boolean function described by a Boolean expression consists of binary variables, the constants 0 and 1, and the logic operation symbols
- For a given value of the binary variables, the function can be equal to either 1 or 0 (closure on the Boolean set)
- As an example, consider the Boolean function: $F = x + y \cdot z$
- A Boolean function expresses the logical relationship between binary variables and is evaluated by determining the binary value of the expression for all possible values of the variable (truth table of the function)

Boolean functions

- A Boolean function can be represented in a truth table
- The number of rows in the truth table is 2^n, where n is the number of variables in the function
- The binary combinations for the truth table are obtained from the binary numbers by counting from 0 through $(2^n – 1)$
- For example, there are eight possible binary combinations for assigning bits to the three variables x, y, and z

Boolean functions

- There is only one way that a Boolean function can be represented in a truth table
- However, when the function is in algebraic form, it can be expressed in a variety of ways, all of which have equivalent logic
- Here is a key fact that motivates our use of Boolean algebra: By manipulating a Boolean expression according to the rules of Boolean algebra, it is sometimes possible to obtain a simpler expression for the same logic function, and thus reduce the complexity of the circuit representing that logic
- Consider, for example, the following Boolean functions:

$$F_1 = x'yz + xy'z + xyz \qquad F_2 = x'yz + yz$$

- Can we simply these?

Boolean functions

- A simplified function also has a simplified logic gate implementation
- Therefore, the two circuits have the same outputs for all possible binary combinations of inputs (truth table)
- Thus, each circuit implements the same identical function, but the one with fewer gates and fewer inputs to gates is preferable because it requires fewer wires and components
- In general, there are many equivalent representations of a logic function
- Finding the most economic representation of the logic is an important design task

Boolean functions

- The manipulation of Boolean algebra consists mostly of reducing an expression for the purpose of obtaining a simpler circuit
- Functions of up to five variables can be simplified by the map method which will be discussed later
- For complex Boolean functions and many different outputs, designers of digital circuits use computer minimization programs that are capable of producing optimal circuits with millions of logic gates
- The only manual method available is a procedure employing the basic relations and other manipulation techniques that become familiar with use, but remain, nevertheless, subject to human error
- Example: Simplify $F = xy + x'z + yz$ (consensus theorem)

Complement of a function

- The complement of a function *F* is *F'* and is obtained from an interchange of 0's for 1's and 1's for 0's in the value of *F* (truth table method)
- The complement of a function may be derived algebraically through DeMorgan's theorems
- Example: $F = x + y'z$ what is F'?
- But what if the function has more terms?
- DeMorgan's theorems can be extended to three or more variables
- The three-variable form of the first DeMorgan's theorem:

$$(x + y + z)' = x'y'z'$$
$$(xyz)' = x' + y' + z'$$

- The generalized form of DeMorgan's theorems states that the complement of a function is obtained by interchanging AND and OR operators and complementing each literal

Minterms

- A binary variable may appear either in its normal form (*x*) or in its complement form (x)
- Now consider two binary variables *x* and *y* combined with an AND operation
- Since each variable may appear in either form, there are four possible combinations: $xy, x'y, xy', x'y'$
- Each of these four AND terms is called a *minterm*, or a *standard product*
- In a similar manner, *n* variables can be combined to form 2^n minterms
- The binary numbers from 0 to 2^n - 1 are listed under the *n* variables. Each minterm is obtained from an AND term of the *n* variables, with each variable being primed if the corresponding bit of the binary number is a 0 and unprimed if a 1
- A symbol for each minterm is m_j, where the subscript *j* denotes the decimal equivalent of the binary number of the minterm designated

Maxterms

- In a similar fashion, *n* variables forming an OR term, with each variable being primed or unprimed, provide 2^n possible combinations, called *maxterms*, or *standard sums*
- Each maxterm is obtained from an OR term of the *n* variables, with each variable being unprimed if the corresponding bit is a 0 and primed if a 1, and
- Maxterms are denoted by M_j
- It is important to note that:

1. Each maxterm is the complement of its corresponding minterm and vice versa
2. Minterms are 1 for a unique combination of the variables, ie, $x'y$ is only one when x is 0 and y is 1, in all other cases, it is zero
3. Maxterms are 0 for a single unique combination of variables

Minterms and Maxterms

Minterms and Maxterms for Three Binary Variables

			Minterms		Maxterms	
x	**y**	**z**	**Term**	**Designation**	**Term**	**Designation**
0	0	0	$x'y'z'$	m_0	$x + y + z$	M_0
0	0	1	$x'y'z$	m_1	$x + y + z'$	M_1
0	1	0	$x'yz'$	m_2	$x + y' + z$	M_2
0	1	1	$x'yz$	m_3	$x + y' + z'$	M_3
1	0	0	$xy'z'$	m_4	$x' + y + z$	M_4
1	0	1	$xy'z$	m_5	$x' + y + z'$	M_5
1	1	0	xyz'	m_6	$x' + y' + z$	M_6
1	1	1	xyz	m_7	$x' + y' + z'$	M_7

Boolean functions

- Any Boolean function can be expressed algebraically from a given truth table by forming a minterm for each combination of the variables that produces a 1 in the function and then taking the OR of all those terms
- Example: $f_1 = m_1 + m_4 + m_7$

$$f_1 = x'y'z + xy'z' + xyz$$

- Thus, any Boolean function can be expressed as a sum of minterms (with "sum" meaning the ORing of terms)

x	y	z	Function f_1
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	0
1	1	1	1

Boolean functions

- Now consider the complement of a Boolean function
- It may be read from the truth table by forming a minterm for each combination that produces a 0 in the function and then ORing those terms

$$f_1' = m_0 + m_2 + m_3 + m_5 + m_6$$

- If we again take a complement, we get f_1 back:

$$f_1 = (m_0 + m_2 + m_3 + m_5 + m_6)'$$

$$f_1 = m_0' m_2' m_3' m_5' m_6'$$

$$f_1 = M_0 M_2 M_3 M_5 M_6$$

x	y	z	Function f_1
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	0
1	1	1	1

Boolean functions

- This shows a second property of Boolean algebra: Any Boolean function can be expressed as a product of maxterms (with “product” meaning the ANDing of terms)
- The procedure for obtaining the product of maxterms directly from the truth table is as follows: Form a maxterm for each combination of the variables that produces a 0 in the function, and then form the AND of all those maxterms
- Boolean functions expressed as a sum of minterms or product of maxterms are said to be in *canonical form*

x	y	z	Function f_1
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	0
1	1	1	1

Canonical form

- Previously, we stated that, for *n* binary variables, one can obtain 2^n distinct minterms and that any Boolean function can be expressed as a sum of minterms
- The minterms whose sum defines the Boolean function are those which give the 1's of the function in a truth table
- Since the function can be either 1 or 0 for each minterm, and since there are 2^n minterms, one can calculate all the functions that can be formed with *n* variables to be 2^{2n}
- If a function is not in a canonical form, it can be made so by first expanding the expression into a sum of AND terms
- Each term is then inspected to see if it contains all the variables. If it misses one or more variables, it is ANDed with an expression such as $x + x'$, where x is one of the missing variables
- Example: $f = x + y'z$

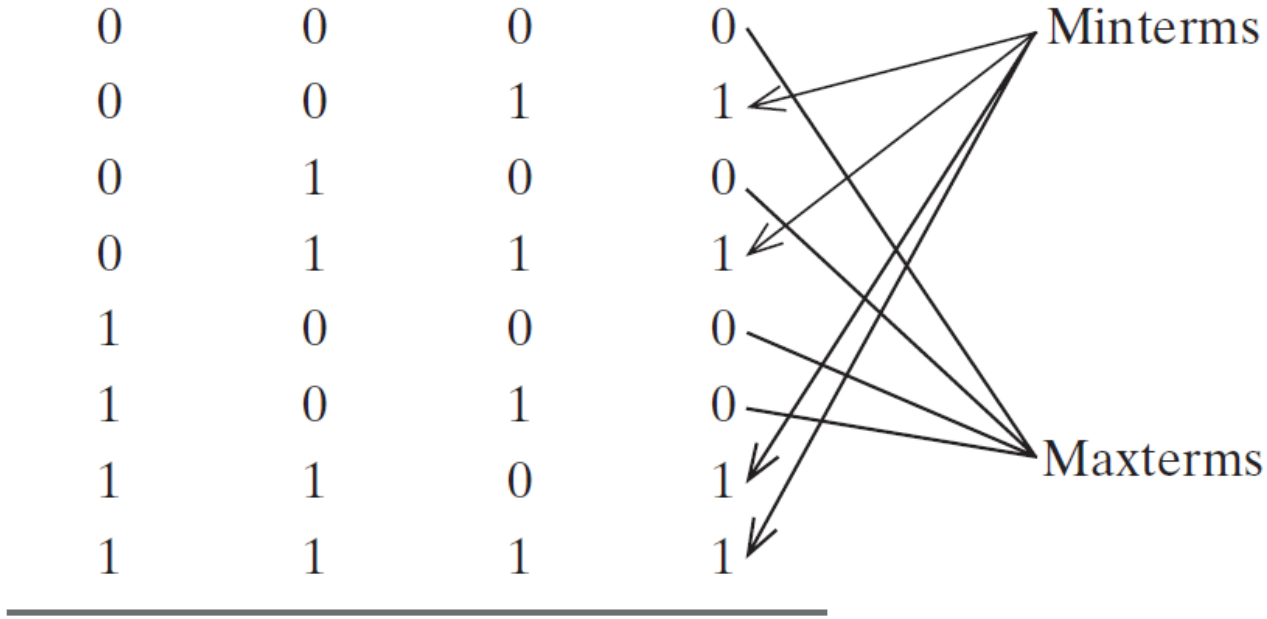
Canonical form

- Each of the 2^{2n} functions of *n* binary variables can be also expressed as a product of maxterms
- To express a Boolean function as a product of maxterms, it must first be brought into a form of OR terms
- This may be done by using the distributive law, $x + yz = (x + y)(x + z)$
- Then any missing variable *x* in each OR term is ORed with *xx'*
- Example: $f = x + y'z$
- Brief notation for sum of minterms: $F(A, B, C) = \sum(1, 4, 5, 6, 7)$
- Brief notation for product of maxterms: $F(A, B, C) = \prod(0,2,3)$

Canonical form

- To convert from one canonical form to another, interchange the symbols $\sum$ and $\prod$ and list those numbers missing from the original form
- In order to find the missing terms, one must realize that the total number of minterms or maxterms is 2^n, where *n* is the number of binary variables in the function
- This is because the function can either have 0 (maxterm) or 1 (minterm) as the output

x	y	z	F
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	1
1	1	1	1



Standard form

- Another way to express Boolean functions is in *standard* form
- In this configuration, the terms that form the function may contain one, two, or any number of literals
- There are two types of standard forms: the sum of products and products of sums

$$F = x' + y'z + xz \qquad\qquad G = (x' + y)(y' + z)(x + z)$$

- The logic diagram of a sum-of-products expression consists of a group of AND gates followed by a single OR gate
- Each product term requires an AND gate, except for a term with a single literal
- The logic sum is formed with an OR gate whose inputs are the outputs of the AND gates
- It is assumed that the input variables are directly available in their complements, so inverters are not included in the diagram
- This circuit configuration is referred to as a *two-level implementation*