Lecture 9 – Logic gates

Dr. Aftab M. Hussain,

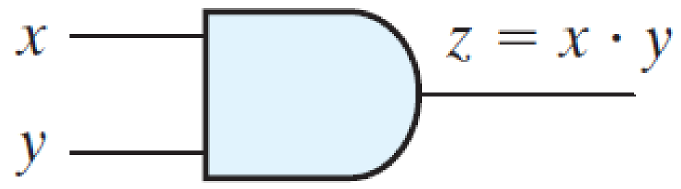
Assistant Professor, PATRIoT Lab, CVEST

Logic gates!

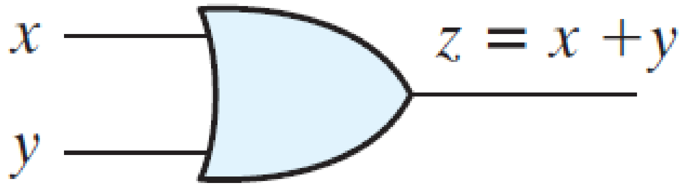
- Logic gates are electronic circuits that operate on one or more input signals to produce an output signal
- Because Boolean world is binary in nature, we represent Boolean values as signals of two distinct voltage levels
- Voltage-operated logic circuits respond to these voltage levels that represent 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 5 V

Logic gates

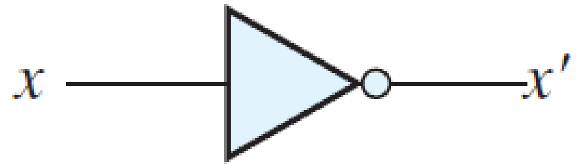
- We represent the logic gates for basic Boolean operations as shown
- Logic gates can have more than two inputs (except for NOT gate of course!)



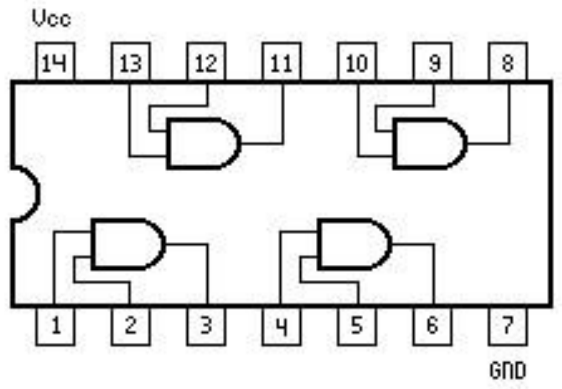
(a) Two-input AND gate



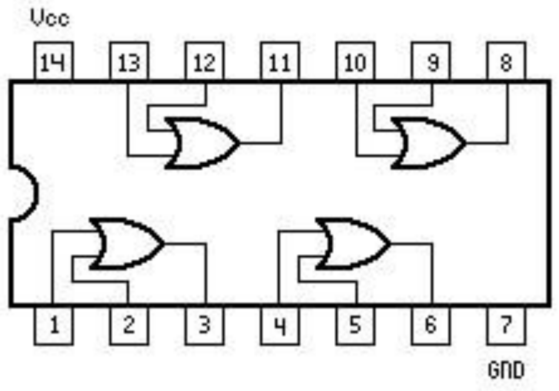
(b) Two-input OR gate



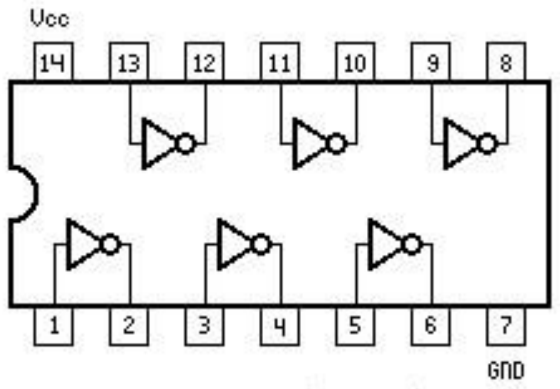
(c) NOT gate or inverter



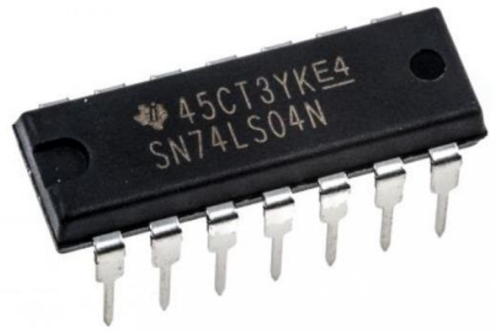
7408 (AND)



7432 (OR)

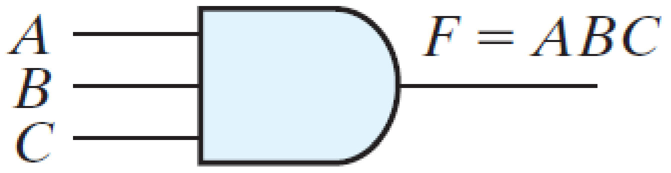


7404 (NOT)

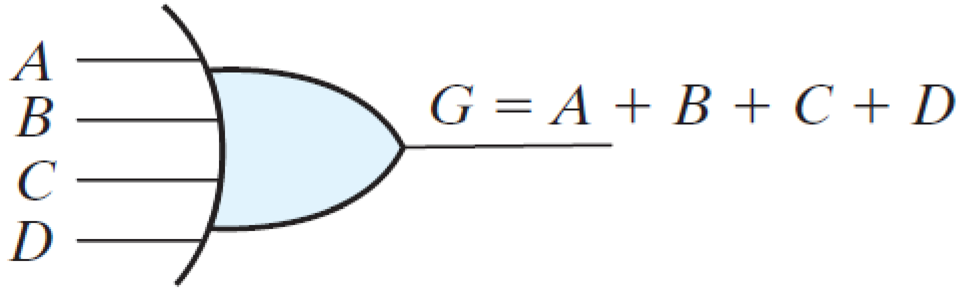


Logic gates

- 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

Logic gates

- Since Boolean functions are expressed in terms of AND, OR, and NOT operations, it is easier to implement a Boolean function with these type of gates
- Still, the possibility of constructing gates for the other logic operations is of practical interest
- Factors to be weighed in considering the construction of other types of logic gates are:

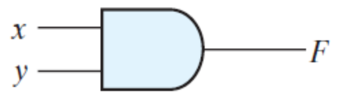
1. The feasibility and economy of producing the gate with physical components
2. The possibility of extending the gate to more than two inputs
3. The basic properties of the binary operator such as commutativity and associativity
4. The ability of the gate to implement Boolean functions alone or in conjunction with other gates

Logic gates

- Of the 16 functions for two variables x and y, two are equal to a constant and four are unary operators (transfer and complement)
- This leaves us with 10 functions for multiple input operations
- Out of these, four functions (two inhibitions and two implications) are not commutative or associative and thus are impractical to use as standard logic gates. Further, their logical definitions cannot be easily extended to multiple inputs
- Hence, the other six—AND, OR, NAND, NOR, exclusive-OR, and equivalence—are used as standard multi-input gates in digital design

Logic gates

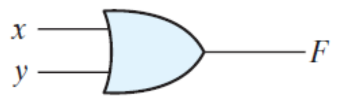
AND



$F = x \cdot y$

x	y	F
0	0	0
0	1	0
1	0	0
1	1	1

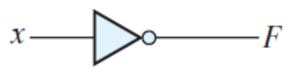
OR



$F = x + y$

x	y	F
0	0	0
0	1	1
1	0	1
1	1	1

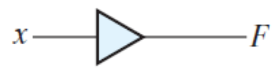
Inverter



$F = x'$

x	F
0	1
1	0

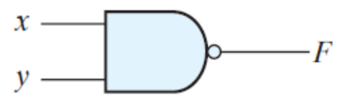
Buffer



$F = x$

x	F
0	0
1	1

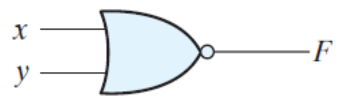
NAND



$F = (xy)'$

x	y	F
0	0	1
0	1	1
1	0	1
1	1	0

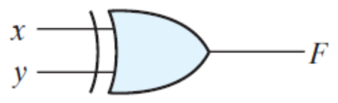
NOR



$F = (x + y)'$

x	y	F
0	0	1
0	1	0
1	0	0
1	1	0

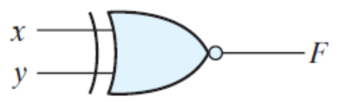
Exclusive-OR
(XOR)



$F = xy' + x'y$
$= x \oplus y$

x	y	F
0	0	0
0	1	1
1	0	1
1	1	0

Exclusive-NOR
or
equivalence



$F = xy + x'y'$
$= (x \oplus y)'$

x	y	F
0	0	1
0	1	0
1	0	0
1	1	1

Multiple inputs

- The exclusive-OR and equivalence gates are both commutative and associative and can be extended to more than two inputs
- However, multiple-input exclusive-OR gates are difficult to make from the hardware standpoint
- The definition of the function must be modified when extended to more than two variables
- Multi-input Exclusive-OR is an *odd* function (i.e., it is equal to 1 if the input variables have an odd number of 1's), and XNOR is its complement

Multiple inputs

- A gate can be extended to have multiple inputs if the binary operation it represents is commutative and associative
- The AND andOR operations, defined in Boolean algebra, possess these two properties
- The NAND and NOR functions are commutative
- The difficulty is that the NAND and NOR operators are not associative

$$(x \downarrow y) \downarrow z \neq x \downarrow (y \downarrow z)$$

- To overcome this difficulty, we define the multiple NOR (or NAND) gate as a complemented OR (or AND) gate. Thus, by definition, we have:

$$x \downarrow y \downarrow z = (x + y + z)'$$
$$x \uparrow y \uparrow z = (xyz)'$$

Boolean functions with gates

- We can represent any Boolean function using logic gates
- This forms the circuit for the function
- Examples:

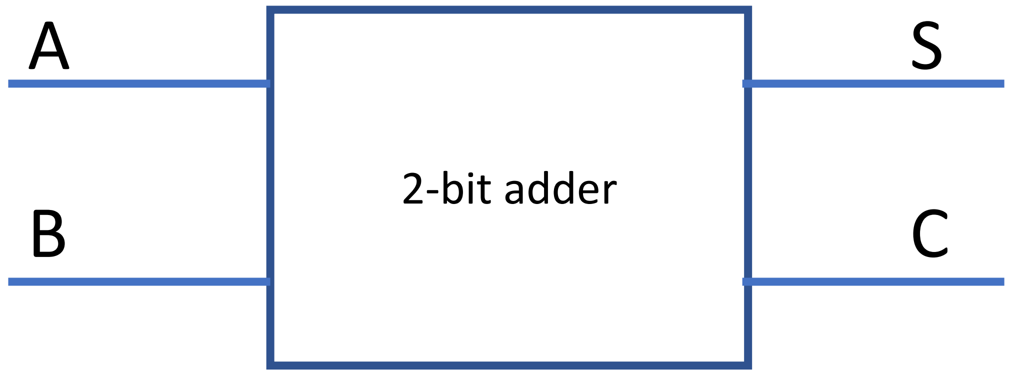
1. $F = x + y.z$
2. $G = ABC + A'B' + AB'C'$

- Simplification of the function leads to a simpler circuit (less gates and smaller gates)

Logic circuits!

- Our first real circuit…
- Logic circuits are combinations of logic gates to make a particular logic function of our choice
- The logic function can be uniquely represented by a truth-table
- However, many algebraic expressions give the same truth-table and the best or most optimum way of making the circuit is to be found by designers
- Consider a statement: design a circuit to add two binary bits
- We go from having a logic function to getting the truth-table, then obtaining the circuit
- In this case, because of its simplicity, we know that

$$S = A \oplus \mathrm{B} \text{ and } C = A \cdot \mathrm{B}$$



A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Gate interconversions

- For a given function, many logic implementations can be done using the gates discussed
- But can we do ALL these implementations with a single logic function implemented over and over?
- These are the NAND and NOR gates –they are referred to as *Universal gates* for this property
- How do we prove this?
- If we can prove that we can get NOT, AND and OR from these gates, we can generate other logic functions using these basic functions

Gate interconversions

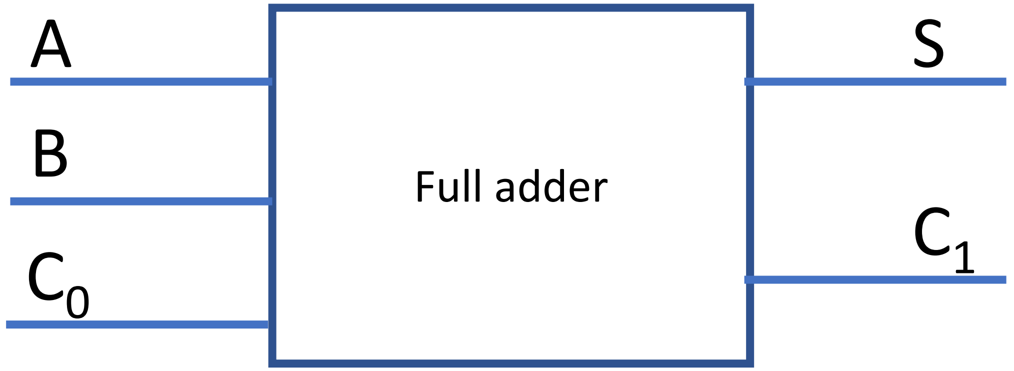
- Obtain NOT, AND and OR from NAND

- Obtain NOT, AND and OR from NOR

- Can we get NOT, AND and OR from NOT, AND or OR gates?

Logic circuits

- Let us say we wanted to continue adding, now we need to take care of the previous carry
- Thus, we need to add A, B and C to get a generic adding machine
- Still from observation, we can deduce that $S = A \oplus B \oplus C_0$
- But what is the algebraic expression for C_1?
- We know the sum of minterms, but is there an better way?



A	B	C_0	C_1	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Gate level minimization

- *Gate-level minimization* is the design task of finding an optimal gate-level implementation of the Boolean functions describing a digital circuit
- This task is well understood, but is difficult to execute by manual methods when the logic has more than a few inputs
- Fortunately, computer-based logic synthesis tools can minimize a large set of Boolean equations efficiently and quickly
- Nevertheless, it is important that a designer understand the underlying mathematical description and solution of the problem