Lecture 9 – Logic gates

Boolean functions

- When the binary operators AND and OR are placed between two variables, *x* and *y*, they form two Boolean functions, *x.y* and *x* + *y*, respectively
- However, we can have 2^{2^n} functions for *n* binary variables!
- Thus, for two variables, *n* = 2, and the number of possible Boolean functions is 16
- Therefore, the AND and OR functions are only 2 of a total of 16 possible functions formed with two binary variables
- Let us find the other 14 functions and investigate their properties

Boolean functions

x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

- Constant functions: 0 and 1 (F_0 , F_{15})
- AND and OR –the well-known logic functions
- Transfer functions: x and y

Boolean functions

x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

- NAND and NOR –these are complementary functions to the usual AND and OR functions
- Take the AND/OR and then take the complement
- NAND is represented by ↑and NOR is represented by ↓
- $x \uparrow y = (x.y)'$ and $x \downarrow y = (x + y)'$

Boolean functions

x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

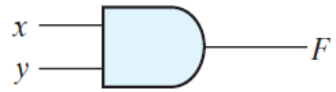
- Exclusive OR (XOR) returns 1 only if one of x or y is 1, it is 0 if both are one
- This is represented by the symbol $\oplus$
- $x \oplus y = x'y + y'x$
- The complement of this is XNOR or Equivalence (is x=y?)

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

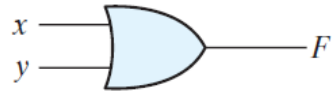
AND



$F = x \cdot y$

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

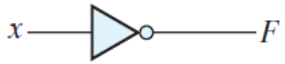
OR



$F = x + y$

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

Inverter



$F = x'$

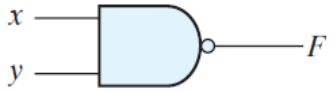
x	F
0	1
1	0

Buffer

$F = x$

x	F
0	0
1	1

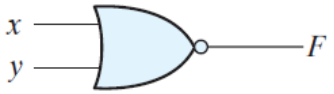
NAND



$F = (xy)'$

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

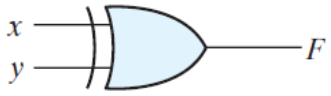
NOR



$F = (x + y)'$

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

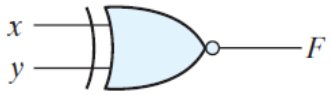
Exclusive-OR (XOR)



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

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

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

Multiple inputs

- A gate can be extended to have multiple inputs if the binary operation it represents is commutative and associative
- The AND and OR 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:

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

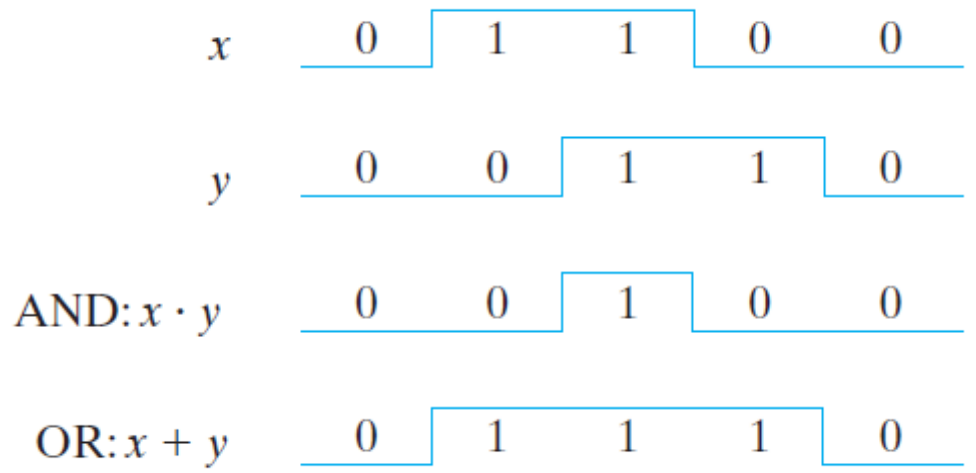
Multiple inputs

- 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

Timing diagrams

- With logic functions, it is always nice to visualize the various conditions giving a particular output
- One way of visualizing is the truth table, another way can be the timing diagram of a particular function
- Timing diagrams are useful to indicate the sequence of events in large combinational circuits

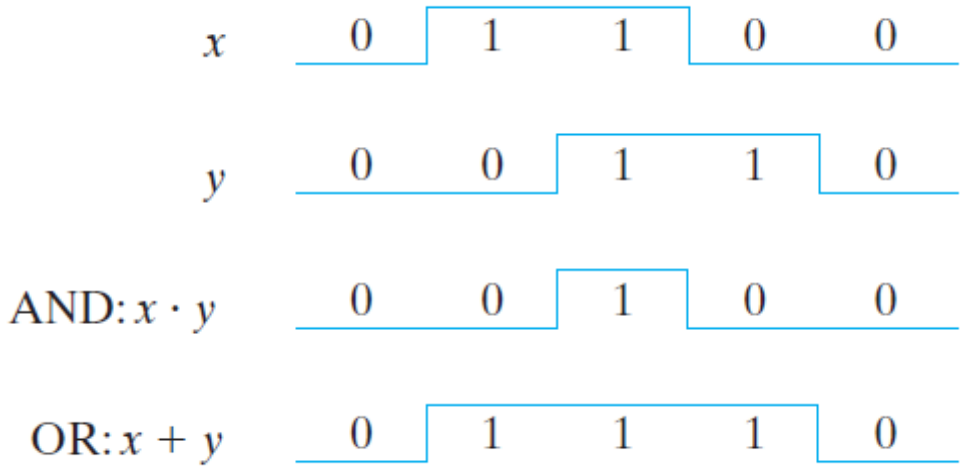
AND			OR		
x	y	$x \cdot y$	x	y	$x + y$
0	0	0	0	0	0
0	1	0	0	1	1
1	0	0	1	0	1
1	1	1	1	1	1



Timing diagrams

- The timing diagrams illustrate the idealized response of each gate to the four input signal combinations
- The horizontal axis of the timing diagram represents the time, and the vertical axis shows the signal as it changes between the two possible voltage levels
- In reality, the transitions between logic values occur quickly, but not instantaneously
- The low level represents logic 0, the high level logic 1

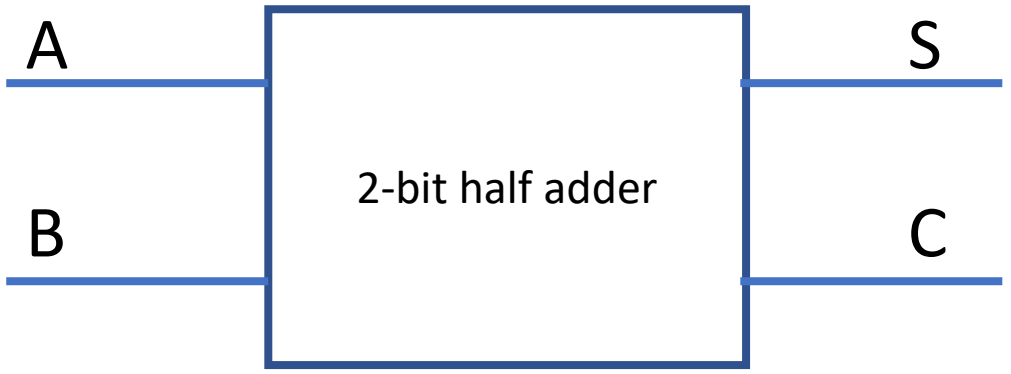
AND			OR		
x	y	$x \cdot y$	x	y	$x + y$
0	0	0	0	0	0
0	1	0	0	1	1
1	0	0	1	0	1
1	1	1	1	1	1



Logic circuits!

- Our first 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 simple 2-bit adder: A + B
- 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

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