EC2.101 – Digital Systems and Microcontrollers

Lecture 5 – Other Logic Functions

Dr. Aftab M. Hussain,

Assistant Professor, PATRIoT Lab, CVEST

Chapter 2

Timing diagrams

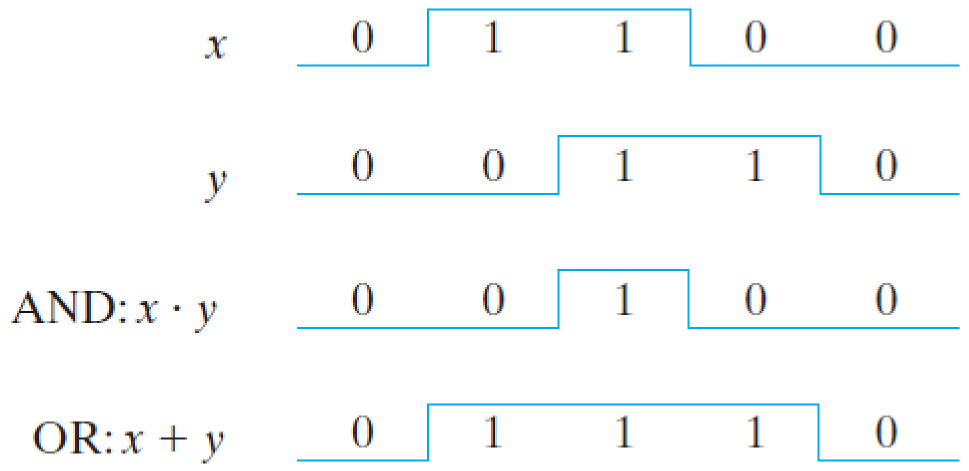
- With logic functions, it is always nice to visualize the various conditions giving a particular output
- One way of visualizing is the truth table, another way can be the timing diagram of a particular function
- Timing diagrams are useful to indicate the sequence of events in large combinational circuits

AND

x	y	$x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

OR

x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1



Timing diagrams

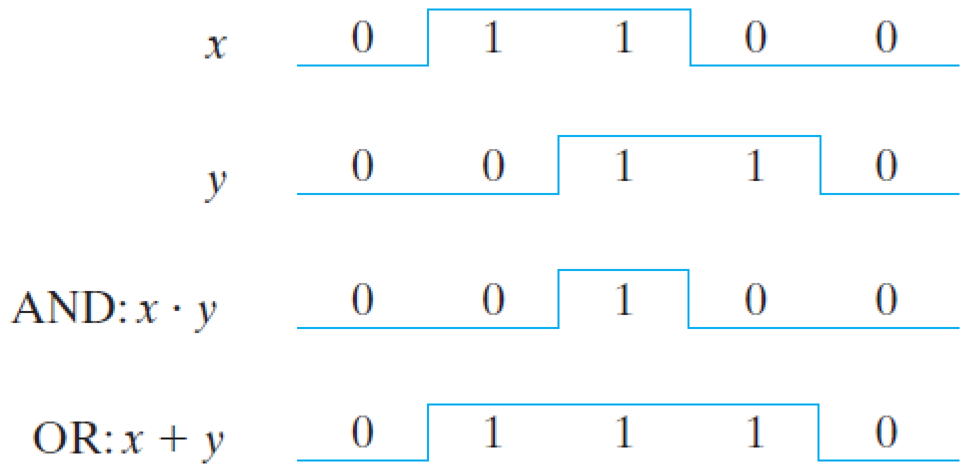
- The timing diagrams illustrate the idealized response of each gate to the four input signal combinations
- The horizontal axis of the timing diagram represents the time, and the vertical axis shows the signal as it changes between the two possible voltage levels
- In reality, the transitions between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

AND

x	y	$x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

OR

x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1



Other logic 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, previously we stated that there are 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

Other logic 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
- AND and OR – the well known logic functions
- Transfer functions: x and y
- Complement functions: x' and y'

Other logic 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 $x \downarrow y = (x + y)'$

Other logic 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?)

Other logic 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

- Inhibition function: x but not y (F2), and y but not x (F4)
- x but not y: If y is LOW then what is x?
- It is represented by a /
- $x/y = xy'$

Other logic 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

- Implications: x implies y (F13), or y implies x (F11)
- This tells us whether the variables x and y are following the *given* implication rule
- It is not for determining whether the two variables for an implication rule between them

Other logic functions

Boolean Functions	Operator Symbol	Name	Comments
$F_0 = 0$		Null	Binary constant 0
$F_1 = xy$	$x \cdot y$	AND	x and y
$F_2 = xy'$	x/y	Inhibition	x, but not y
$F_3 = x$		Transfer	x
$F_4 = x'y$	y/x	Inhibition	y, but not x
$F_5 = y$		Transfer	y
$F_6 = xy' + x'y$	$x \oplus y$	Exclusive-OR	x or y, but not both
$F_7 = x + y$	$x + y$	OR	x or y
$F_8 = (x + y)'$	$x \downarrow y$	NOR	Not-OR
$F_9 = xy + x'y'$	$(x \oplus y)'$	Equivalence	x equals y
$F_{10} = y'$	y'	Complement	Not y
$F_{11} = x + y'$	$x \subset y$	Implication	If y, then x
$F_{12} = x'$	x'	Complement	Not x
$F_{13} = x' + y$	$x \supset y$	Implication	If x, then y
$F_{14} = (xy)'$	$x \uparrow y$	NAND	Not-AND
$F_{15} = 1$		Identity	Binary constant 1

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:

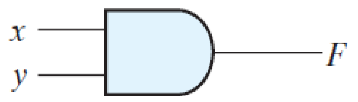
1. The feasibility and economy of producing the gate with physical components
2. The possibility of extending the gate to more than two inputs
3. The basic properties of the binary operator such as commutativity and associativity
4. The ability of the gate to implement Boolean functions alone or in conjunction 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
- The other six—AND, OR, NAND, NOR, exclusive-OR, and equivalence—are used as standard multi-input gates in digital design

Logic gates

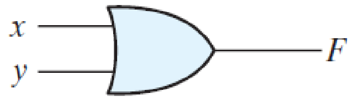
AND



$F = x \cdot y$

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

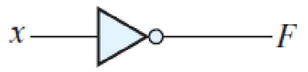
OR



$F = x + y$

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

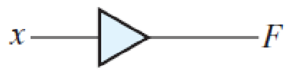
Inverter



$F = x'$

x	F
0	1
1	0

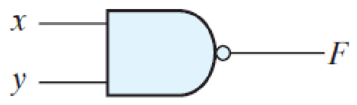
Buffer



$F = x$

x	F
0	0
1	1

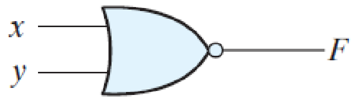
NAND



$F = (xy)'$

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

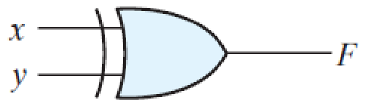
NOR



$F = (x + y)'$

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

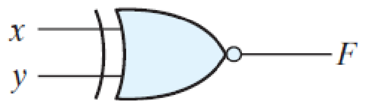
Exclusive-OR
(XOR)



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

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

- 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 uncommon from the hardware standpoint
- In fact, even a two-input function is usually constructed with other types of gates
- Moreover, the definition of the function must be modified when extended to more than two variables
- Exclusive-OR is an *odd* function (i.e., it is equal to 1 if the input variables have an odd number of 1's)

Negative logic

- The binary signal at the inputs and outputs of any gate has one of two values, except during transition
- One signal value represents logic 1 and the other logic 0
- Since two signal values are assigned to two logic values, there exist two different assignments of signal level to logic value
- Choosing the high-level *H* to represent logic 1 defines a positive logic system
- Choosing the low-level *L* to represent logic 1 defines a negative logic system
- The terms *positive* and *negative* are somewhat misleading, since both signals may be positive or both may be negative
- It is not the actual values of the signals that determine the type of logic, but rather the assignment of logic values to the relative amplitudes of the two signal levels

Negative logic

- Hardware digital gates are defined in terms of signal values such as *H* and *L*
- It is up to the user to decide on a positive or negative logic polarity
- Consider an AND gate
- Now consider the negative logic assignment for the same physical gate with *L* = 1 and *H* = 0
- This table represents the OR operation, even though the entries are reversed
- The graphic symbol for the negative logic OR gate is with small triangles in the inputs and output to designate the presence of negative logic

x	y	z
L	L	L
L	H	L
H	L	L
H	H	H

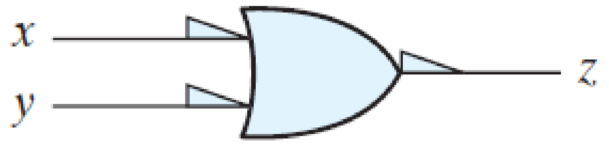
(a) Truth table with *H* and *L*

x	y	z
0	0	0
0	1	0
1	0	0
1	1	1

(c) Truth table for positive logic

x	y	z
1	1	1
1	0	1
0	1	1
0	0	0

(e) Truth table for negative logic



(f) Negative logic OR gate

Negative logic

- The conversion from positive logic to negative logic and vice versa is essentially an operation that changes 1's to 0's and 0's to 1's in both the inputs and the output of a gate
- Since this operation produces the dual of a function, the change of all terminals from one polarity to the other results in taking the dual of the function
- The upshot is that all AND operations are converted to OR operations (or graphic symbols) and vice versa
- In addition, one must not forget to include the polarity-indicator triangle in the graphic symbols when negative logic is assumed

x	y	z
L	L	L
L	H	L
H	L	L
H	H	H

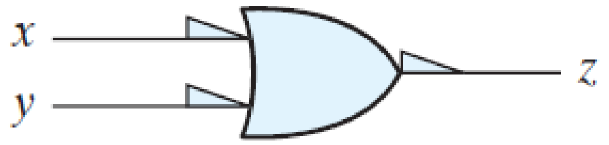
(a) Truth table with H and L

x	y	z
0	0	0
0	1	0
1	0	0
1	1	1

(c) Truth table for positive logic

x	y	z
1	1	1
1	0	1
0	1	1
0	0	0

(e) Truth table for negative logic



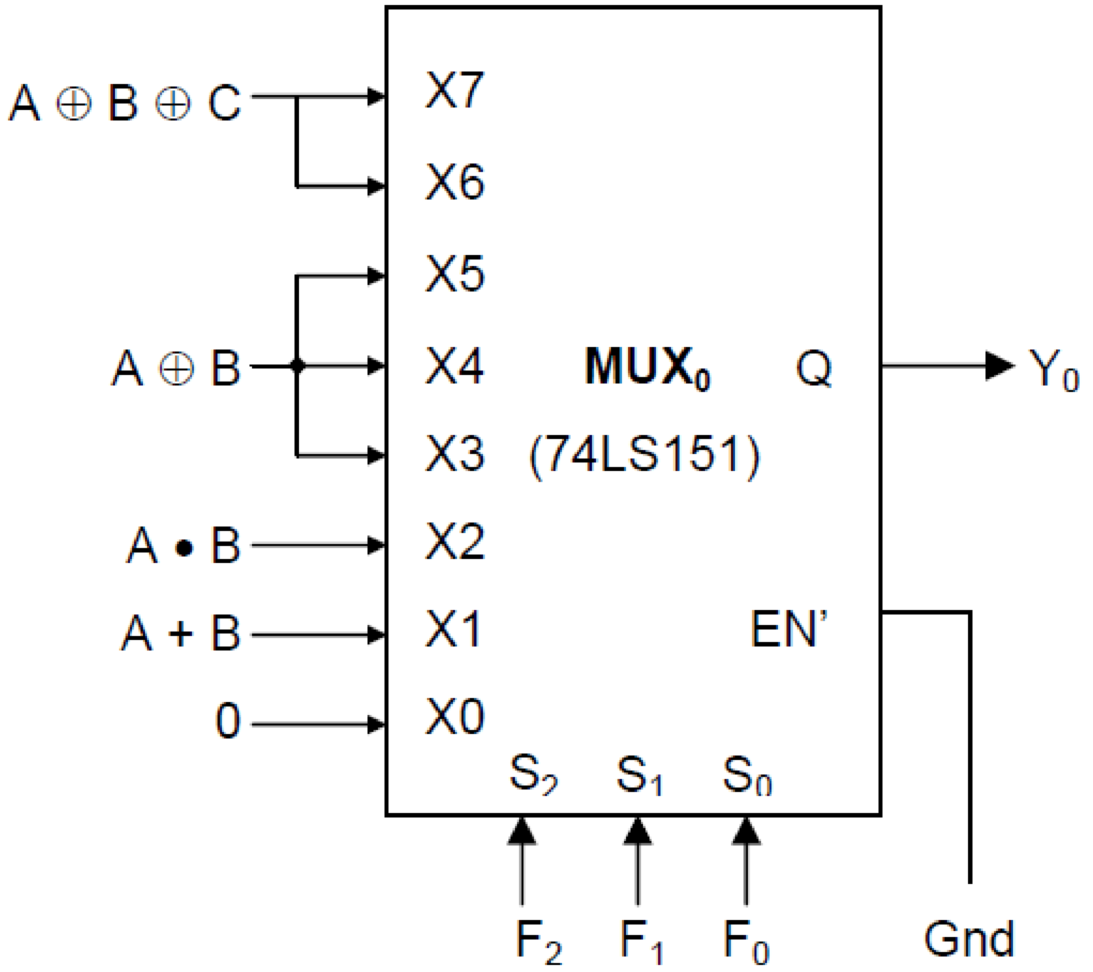
(f) Negative logic OR gate

Gate interconversions

- Many logic implementations can be done using the gates discussed
- Interestingly, we can actually 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
- 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?
- How about XOR gate?

Next Week's Lab

$F_2F_1F_0$	ALU Function	Y_1	Y_0
000	0 (Zero)	-	0
001	A OR B	-	$A + B$
010	A AND B	-	$A \bullet B$
011	A EXOR B	-	$A \oplus B$
100	A PLUS B	Carry	Sum
101	A MINUS B	Borrow	Difference
110	A PLUS B PLUS C	Carry	Sum
111	A MINUS B MINUS C	Borrow	Difference



A ⊕ B ⊕ C
A ⊕ B
A • B
A + B
0
X7
X6
X5
X4
X3
X2
X1
X0
MUX0
(74LS151)
Q
Y0
EN'
S2
S1
S0
F2
F1
F0
Gnd