Tutorial 5

Boolean Algebra and Digital Logic

Objectives

- Understand the relationship between Boolean logic and digital computer circuits.
- Learn how to create and use the truth table for a Boolean function.
- Learn how to synthesize an equivalence to simple logic circuits based on a constrained vocabulary.
- Learn different formats to express Boolean functions.

Boolean Algebra

- Boolean algebra is a mathematical system for the manipulation of variables that can have one of two values.
  - In formal logic, these values are “true” and “false.”
  - In digital systems, these values are “on” and “off,” 1 and 0, or “high” and “low.”
- Boolean expressions or ***Combinational Circuits*** are created by performing operations on Boolean variables by combining basic ***common gates*** .
  - Common Boolean operators (gates) include AND, OR, and NOT.

Boolean Function

- A Boolean function specifies what operation we want a combinational circuit to compute.
- There are many implementations of a given Boolean function.
- We need terminology to specify a Boolean function and to describe particular implementations.
- Diagrams describe implementations, and connectives (and formulas) from sentential (propositional) logic specify Boolean functions.

Truth Tables

- ***A mathematical table*** used in connection with Boolean Algebra to determine the functional value of a logical expression for each combination of values taken by its logical variables.
- ***Two logical expressions are equal if they have identical truth tables***.
- In the next 2 slides, we see the most basic truth tables for the AND, OR and NOT functions. These truth tables will be used in building the truth table for more complex logical expressions.
- The truth table for a Boolean function with **n** variables has $\mathbf{2^n}$ rows or possible outcomes.

Boolean Algebra

- A Boolean operator can be completely described using a truth table.
- The truth table for the Boolean operators AND & OR are shown at the right.
- The AND operator is also known as a Boolean product and designated using **/** or **simply nothing** (look at the table).
- The OR operator is the Boolean sum and designated using **\\/ or +**.

X AND Y

X	Y	XY
0	0	0
0	1	0
1	0	0
1	1	1

X OR Y

X	Y	X+Y
0	0	0
0	1	1
1	0	1
1	1	1

Boolean Algebra

- The truth table for the Boolean NOT operator is shown at the right.
- The NOT operation is most often designated by an overbar or as (¬**x**).
- Note that I use {0,1} in truth tables. This same as using {F,T} respectively.

NOT x

x	$\overline{x}$
0	1
1	0

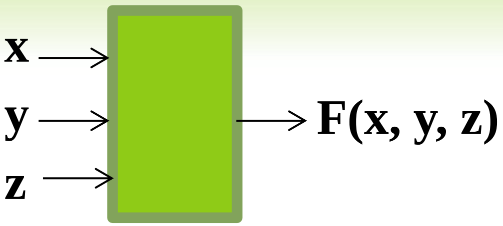
Boolean Algebra

- A Boolean function has:
  - At least one Boolean variable,
  - At least one Boolean operator, and
  - At least one input from the set {0,1}.
- It produces an output that is also a member of the set {0,1}.
- We consider Boolean functions that produce one output for a given a given input combination.

Now you know why the binary numbering system is so handy in digital systems.

Boolean Algebra

- Any combinational circuit F with 3 inputs can be represented as shown.



- On the right, we see the truth table for the Boolean function:

$$F(x,y,z) = x\bar{z}+y$$

same as **F(x, y, z) = (x ∧¬z) ∨ y**

- To make the evaluation of the Boolean function easier, the truth table contains extra (shaded) columns to hold evaluations of subparts of the function.

$F(x,y,z) = x\bar{z}+y$

x	y	z	$\bar{z}$	$x\bar{z}$	$x\bar{z}+y$
0	0	0	1	0	0
0	0	1	0	0	0
0	1	0	1	0	1
0	1	1	0	0	1
1	0	0	1	1	1
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	0	0	1

Boolean Algebra

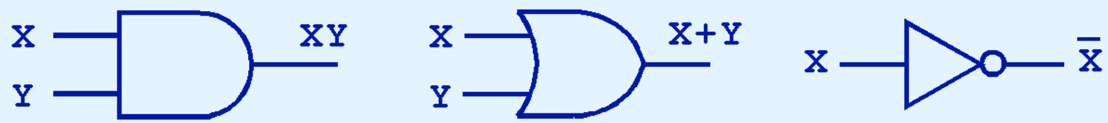
- As with common arithmetic, Boolean operations have rules of precedence.
- The NOT operator has highest priority, followed by AND and then OR.
- This is how we chose the (shaded) function subparts in our table.

$F(x,y,z) = x\bar{z}+y$

x	y	z	$\bar{z}$	$x\bar{z}$	$x\bar{z}+y$
0	0	0	1	0	0
0	0	1	0	0	0
0	1	0	1	0	1
0	1	1	0	0	1
1	0	0	1	1	1
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	0	0	1

Basic Logic Gates

- The three simplest gates are the AND, OR, and NOT gates.



X AND Y

X	Y	XY
0	0	0
0	1	0
1	0	0
1	1	1

X OR Y

X	Y	X+Y
0	0	0
0	1	1
1	0	1
1	1	1

NOT X

X	$\overline{X}$
0	1
1	0

- They correspond directly to their respective Boolean operations, as you can see by their truth tables.

Logic Gates

- Another very useful gate is the exclusive OR (XOR) gate.
- The output of the XOR operation is true only when the values of the inputs differ.

X XOR Y

X	Y	X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0

Note the special symbol ⊕ **used for the XOR operation. You may see a different symbol in other books.**

Using Truth Table

- We will use truth tables to ***synthesize*** or ***find the logical equivalent*** of a Boolean function (**designated |= =|**)using a set of given connectives or vocabulary.
- Two Boolean functions are logically equivalent if they have the same truth table outputs.

Example-1

Let **X** be the ternary connective such that **Xpqr** is equivalent to (p+q) /\ (q<--->r).

(p<--->q) is true iff p and q have the same truth value. '+' is exclusive or. F and T are the constant functions.

A- Using {'X', 'T'}, synthesize: ~p |= =| X__ __ __

B- Using {'X', '~', 'F'), synthesize: p\/q |= =| __X__ __ __

Example-1

We will start by creating the truth table for

Xpqr |= =| (p+q) /\ (q<--->r)

p	q	r	p+q	q<-->r	(p+q)/\(q <--> r)
F	F	F	F	T	F
F	F	T	F	F	F
F	T	F	T	F	F
F	T	T	T	T	T
T	F	F	T	T	T
T	F	T	T	F	F
T	T	F	F	F	F
T	T	T	F	T	F

Example-1 Part-A

A- Using {'X', 'T'}, synthesize: ~p |= =| X__ __ __

Now, the truth table will be used as a lookup table. For this part, the only relevant variable is p. We need to create a truth table to show all possible outcomes for the function "~p" for all possible values of p. Namely, {F,T} The variables q and r are not relevant and are mutated using the constant value "T".

p	~p	XpTT
F	T	T
T	F	F

So, ~p |= =| XpTT

Example-1 Part-B

B- Using {‘X’, ‘~’, ‘F’), synthesize: p⋁q |= =| __X__ __ __

We see that the resulting truth table for the second functional equivalence is the complement of the desired function.

p	q	p⋁q	XpqF	X~pqF
F	F	F	X(FFF)=F	X(TFF)=T
F	T	T	X(FTF)=F	X(TTF)=F
T	F	T		X(FFF)=F
T	T	T		X(FTF)=F
Yes/No			No	Yes

So, p⋁q |= =| ~X ~pqF

EXAMPLE-2

‘M’ is the ternary minority ‘connective’. **Mpqr** is true iff a minority of its arguments is True. ‘#’ is the ternary majority ‘connective’. Thus, #pqr |= =| ~Mpqr. ‘F’ is a constant function.

A- Using {‘M’}, synthesize: ~p |= =| M__ __ __

B- Using {‘#’, ‘F’), synthesize: p/\q |= =| #__ __ __

What do we note about this example. How is it different from the previous one.

Example-2
Part-A

A truth table is not really required for this example. We can simply inspect the input for the number of inputs with true value to determine the outcome of the function.

A- Using {'M'}, synthesize: ~p |= =| M__ __ __

For this part, we can only use the variable p. So there is only one possible solution.

p	~p	Mppp
F	T	T
T	F	F

So, ~p |= =| Mppp

Example-2
Part-B

B- Using {‘#’, ‘F’), synthesize: p/\q |= =| #__ __ __

Similar to previous example, r is not required and can be mutated using the constant value ‘F’ that is given. It can be seen that one single solution is possible.

p	q	p/\q	#pqF
F	F	F	#(FFF)=F
F	T	F	#(FTF)=F
T	F	F	#(TFF)=F
T	T	T	#(TTF)=T
YES/NO			Yes

So, p/\q |= =| #pqF

Boolean Algebra

- In the previous examples, we have seen different exercises for synthesizing the logical equivalence for Boolean functions; *designated using (|= =|).*
- We have seen 2 types of functions:
  - *Functions described by a Boolean algebra expression.*
  - *Functions described in terms of functionality.*
- In both cases, we can use the truth table to determine the logical equivalent of a given function. However, the truth table was not required for the second example.
- ***Remember that Two Boolean functions are equal if they have identical truth tables***.

Boolean Algebra

- There are two canonical forms for Boolean expressions: sum-of-products and product-of-sums.
  - Recall the Boolean product is the AND operation and the Boolean sum is the OR operation.
- In the sum-of-products form, ANDed variables are ORed together.
  - For example: $F(x,y,z) = xy + xz + yz$
- In the product-of-sums form, ORed variables are ANDed together:
  - For example: $F(x,y,z) = (x+y)(x+z)(y+z)$

Boolean Algebra

- It is easy to convert a function to sum-of-products form using its truth table.
- We are interested in the values of the variables that make the function true (=1).
- Using the truth table, we list the values of the variables that result in a true function value.
- Each group of variables is then ORed together.

$$F(x,y,z) = x\bar{z}+y$$

x	y	z	$x\bar{z}+y$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Boolean Algebra

- The sum-of-products form for our function is:

$$F(x,y,z) = (\bar{x}y\bar{z}) + (\bar{x}yz) + (x\bar{y}\bar{z}) + (xy\bar{z}) + (xyz)$$

We note that this function is not in simplest terms. Our aim is only to rewrite our function in canonical sum-of-products form.

$$F(x,y,z) = x\bar{z}+y$$

x	y	z	$x\bar{z}+y$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Boolean Algebra

Example: Use the truth table in Example-1to produce the sum-of-products form for the function **Xpqr |= =| (p+q) /\ (q<--->r)**

p	q	r	p+q	q<-->r	(p+q)/\(q <--> r)
F	F	F	F	T	F
F	F	T	F	F	F
F	T	F	T	F	F
F	T	T	T	T	T
T	F	F	T	T	T
T	F	T	T	F	F
T	T	F	F	F	F
T	T	T	F	T	F

By looking at the two True entries in the truth table, X can be written as:

Xpqr |= =| ~pqr + p~q~r

Note again that this function is not in simplest terms. We ***do not need*** to simplify functions for this course. However, simplifying the function may be useful for synthesizing

Boolean Algebra

- Digital computers contain circuits that implement Boolean functions.
- The simpler that we can make a Boolean function, the smaller the circuit that will result.
  - Simpler circuits are cheaper to build, consume less power, and run faster than complex circuits.
- With this in mind, we always want to reduce our Boolean functions to their simplest form.
- There are a number of Boolean identities that help us to do this.

Boolean Algebra
Boolean Identities

- Most Boolean identities have an AND (product) form as well as an OR (sum) form. We give our identities using both forms. Our first group is rather intuitive:

Identity Name	AND Form	OR Form
Identity Law	$1x = x$	$0 + x = x$
Null Law	$0x = 0$	$1 + x = 1$
Idempotent Law	$xx = x$	$x + x = x$
Inverse Law	$x\overline{x} = 0$	$x + \overline{x} = 1$

Boolean Algebra

- Our second group of Boolean identities should be familiar to you from your study of algebra:

Identity Name	AND Form	OR Form
Commutative Law	xy = yx	x+y = y+x
Associative Law	(xy) z = x (yz)	(x+y) +z = x + (y+z)
Distributive Law	x+yz = (x+y) (x+z)	x (y+z) = xy+xz

Boolean Algebra

- Our last group of Boolean identities are perhaps the most useful.
- If you have studied set theory or formal logic, these laws are also familiar to you.

Identity Name	AND Form	OR Form
Absorption Law DeMorgan's Law	$x(x+y) = x$ $\overline{(xy)} = \overline{x} + \overline{y}$	$x + xy = x$ $\overline{(x+y)} = \overline{x}\,\overline{y}$
Double Complement Law	$\overline{(\overline{x})} = x$	

Boolean Algebra

- We can use Boolean identities to simplify:

$$F(X,Y,Z) = (X+Y)(X+\overline{Y})(\overline{X\overline{Z}})$$

as follows:

$(X + Y)(X + \overline{Y})(\overline{X\overline{Z}})$	
$(X + Y)(X + \overline{Y})(\overline{X} + Z)$	DeMorgan's Law
	Double complement Law
$(XX + X\overline{Y} + YX + Y\overline{Y})(\overline{X} + Z)$	Distributive Law
$((X + Y\overline{Y}) + X(Y + \overline{Y}))(\overline{X} + Z)$	Commutative and Distributive Laws
$((X + 0) + X(1))(\overline{X} + Z)$	Inverse Law
$X(\overline{X} + Z)$	Idempotent and Identity Laws
$X\overline{X} + XZ$	Distributive Law
$0 + XZ$	Inverse Law
XZ	Identity Law

Boolean Algebra

- Sometimes it is more economical to build a circuit using the complement of a function (and complementing its result) than it is to implement the function directly.
- DeMorgan's law provides an easy way for finding the complement of a Boolean function.
- Recall DeMorgan's law states:

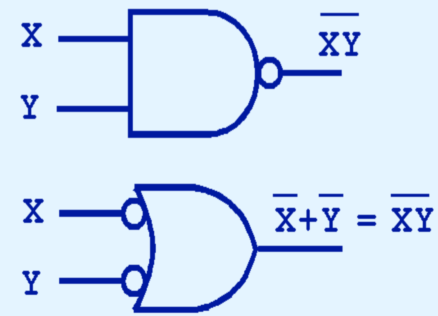
$$\overline{(xy)} = \overline{x} + \overline{y} \quad \text{and} \quad \overline{(x+y)} = \overline{x}\,\overline{y}$$

More Logic Gates

- NAND and NOR are two very important gates. Their symbols and truth tables are shown at the right.

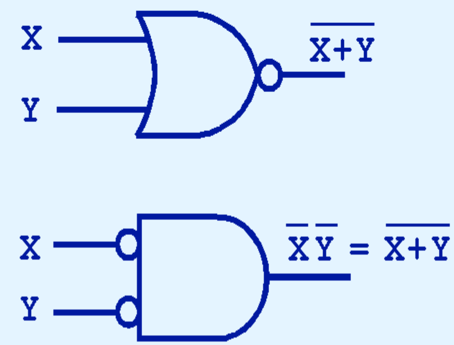
X NAND Y

X	Y	X NAND Y
0	0	1
0	1	1
1	0	1
1	1	0



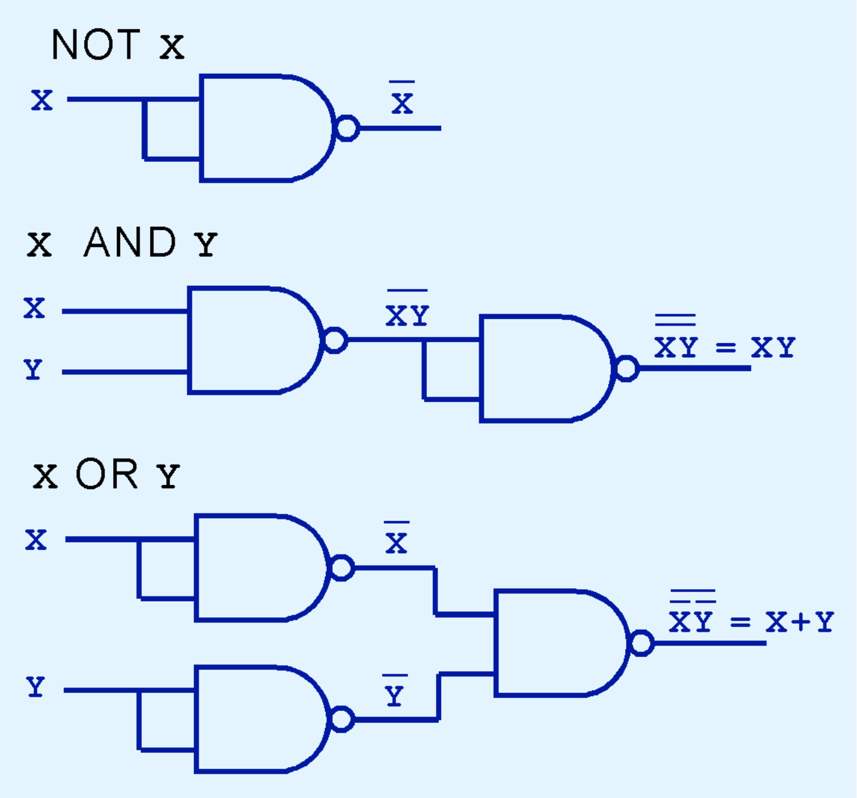
X NOR Y

X	Y	X NOR Y
0	0	1
0	1	0
1	0	0
1	1	0



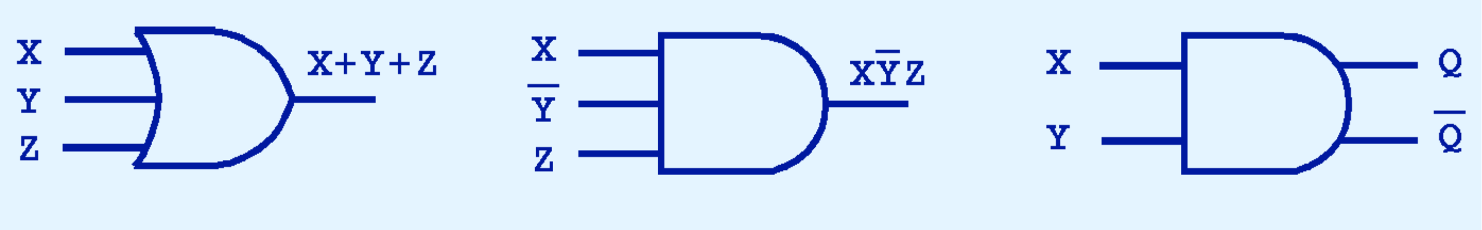
Logic Gates

- NAND and NOR are known as *universal gates* because they are inexpensive to manufacture and any Boolean function can be constructed using only NAND or only NOR gates.



Logic Gates

- Gates can have multiple inputs and more than one output.
  - A second output can be provided for the complement of the operation.
  - We'll see more of this later.



Conclusion

- We have seen the implementation of Boolean logic using combinational circuits.
- Combinational circuits produce different outputs when their inputs change.
- Boolean functions can be completely described by truth tables.
- Logic gates are small circuits that implement Boolean operators.
- The basic gates are AND, OR, and NOT.
  - The XOR gate is also very useful in parity checkers and adders.
- The “universal gates” are NOR, and NAND.