Lecture 8 – Boolean functions

Dr. Aftab M. Hussain,

Assistant Professor, PATRIoT Lab, CVEST

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

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

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

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

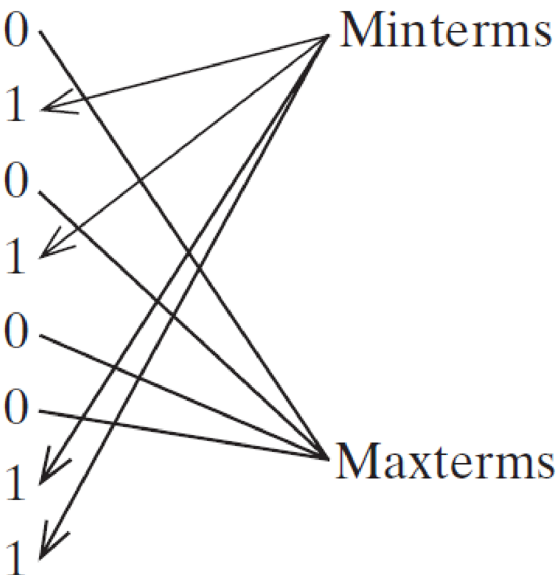
- This shows a second property of Boolean algebra: Any Boolean function can be expressed as a product of maxterms (with “product” meaning the ANDing of terms)
- The procedure for obtaining the product of maxterms directly from the truth table is as follows: Form a maxterm for each combination of the 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

- 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*

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
- AND and OR –the well-known logic functions
- Transfer functions: x and y
- Complement 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 $\uparrow$and NOR is represented by $\downarrow$
- $x \uparrow y = (x.y)'$ 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?)

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

- 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 /
- $x/y=xy'$

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

- 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 form an implication rule between them

Boolean 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