EE 224 Digital Systems – IIT-Bombay

Lecture Notes by Professor Janak H. Patel, University of Illinois at Urbana-Champaign, USA

Self Dual Functions:

We had earlier defined dual function with a descriptive procedure and also with Boolean identity. The descriptive definition applies only to algebraic expression – namely – To obtain a Dual of Boolean Expression, exchange AND's and OR's; and exchange 0's and 1's. The functional definition is:

Dual of $f(x_1, x_2, \ldots, x_n) = Complement\ of\ f(x_1', x_2', \ldots, x_n')$

A Self Dual Function is a dual of itself.

Dual of $f(x_1, x_2, \ldots, x_n) = f(x_1, x_2, \ldots, x_n)$

Dual in the Boolean identity form is –

If $f(x_1, x_2, \ldots, x_n)' = f(x_1', x_2', \ldots, x'_n)$ then f is a self-dual function.

In words, a function f is *self-dual* iff when complementing its input variables, the output becomes f'.

Examples:

$f(a, b, c) = ab + ac + bc$

$dual\ of\ f = (a + b)(a + c)(b + c) = (a + ac + ab + bc)(b + c) = (a + bc)(b + c) = ab + ac + bc$

It must also satisfy: $f(a, b, c)' = f(a', b', c')$

$f(a, b, c)' = (ab + ac + bc)' = (a' + b')(a' + c')(b' + c') = (a' + b'c')(b' + c') = a'b' + a'c' + b'c' = f(a', b', c')$

$f(x, y, z) = x \oplus y \oplus z$ *is a self-dual, since we know that* $(x \oplus y \oplus z)' = x' \oplus y' \oplus z'$

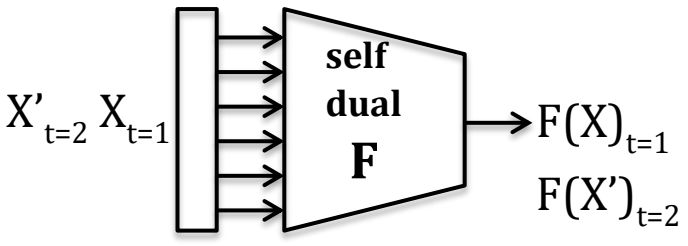
Verify on you own that $x \oplus y$ is not a self-dual!

It is relatively easy to create many self-dual functions from the definition above. Here we give an example of three variable function.

a b c	f(a,b,c)
0 0 0	f_0
0 0 1	f_1
0 1 0	f_2
0 1 1	f_3
1 0 0	f_3'
1 0 1	f_2'
1 1 0	f_1'
1 1 1	f_0'

For arbitrary 3 variable functions we have eight free cells to fill in the output with either 0 or 1, giving us 2^8 functions. However, self-dual functions put a restriction on possible outputs. For example for a self-dual function each minterm must satisfy that when its variables are complemented, its output must also be complemented. For example, minterm *a'b'c* has output assigned f_1, it follows that minterm *abc*' must have output f_1', and so on. This gives us four free cells to assign arbitrary outputs. So the number of self-dual functions of 3 variables is simply 2^4.

In general, for n-variable functions, there are: 2^{2^n} Boolean Functions, while the number of self-dual n-variable functions are: $2^{2^{n-1}}$



One application of Self-Dual function was for error detection in logic circuits. The idea is that if you compute F(X) and then F(X'), you should get complementary outputs when no defects are present in the circuit. It is easy to see that if an input wire is shorted or open, this method will detect it. The figure shows the arrangement. At time t=1, true input X is fed, output $F(x)_{t=1}$ is captured and stored. In the second cycle at time t=2, complemented input X' is fed, output $F(X')_{t=2}$ is compared with the stored value. If they are complement of each other the result is good. This organization is called *Alternating Logic*. For more information see:

D. A. Reynolds and G. Metze, "Fault Detection Capabilities of Alternating Logic," in *IEEE Transactions on Computers*, vol. C-27, no. 12, pp. 1093-1098, Dec. 1978. doi: 10.1109/TC.1978.1675011

Symmetric Functions

A Boolean function that does not change under any permutation of its input variables is called a *Totally Symmetric Function.*
For example: $F(a, b, c) = ab + ac + bc$ and $F(x, y, z) = x \oplus y \oplus z$ are both totally Symmetric. All basic gates, AND, NAND, OR, NOR are also totally symmetric.

When a function does not change under any permutation of a subset of its variables is called a *Partially Symmetric Function*.
For example: $F(x, y, z) = x'y'z + x'yz' + xyz$ is Partially Symmetric in variables y and z.

It is customary to just use "Symmetric function" to mean "Totally Symmetric Function".

Symmetry Theorem: A function $f(x_1, x_2, \ldots, x_n)$ is totally symmetric iff it can be specified by a list of integers $A = \{a_1, a_2, \ldots, a_m\}$, $0 \le a_i \le n$ so that $f = 1$ iff exactly a_i of the n variables are 1.
For example the above function $F = ab + ac + bc$ can be specified by A ={2, 3}.
A parity function $F = w \oplus x \oplus y \oplus z$ can be specified by A = {1, 3}.

Customary notation for symmetric functions uses the symbol S with a subscript of the set A.
Definition: A symbol S_A indicates a symmetric function with the set A. For example the above majority function $S_{\{2, 3\}}(a, b, c) = ab + ac + bc$ and the parity function is $S_{\{1, 3\}}(w, x, y, z) = w \oplus x \oplus y \oplus z$

Boolean Operations on Symmetric Functions:
Let the set M be a set of a-numbers and another set N be another set of a-numbers and let A be set of All a-numbers, then
$S_M + S_N = S_{M \cup N}$

$S_M \bullet S_N = S_{M \cap N}$
$S'_M = S_{A-M}$ where A-M is the set difference

How many Symmetric Functions of n-variables? It is easy to compute this formula from the definition of a-numbers. For a function to be symmetric, we can choose any subset of a-numbers from the set of (n+1) integers {0, 1, 2, …, n}. Number of different subsets of n+1 integers is 2^{n+1}. So for n-variables functions there are 2^{n+1} distinct symmetric functions.

XOR Gate:
Exclusive -OR of (X, Y) in logic is defined as *X OR Y, but not Both*. In Boolean Algebra
$X \oplus Y = X'Y + XY' = (X + Y)(X' + Y')$
Sometimes XNOR is also used as gate, which is simply complement of XOR gate.
$(X \oplus Y)' = XY + X'Y' = (X + Y')(X' + Y) = X' \oplus Y = X \oplus Y'$
From the above definitions it is easy to prove Commutative and Associative properties:
$X \oplus Y = Y \oplus X$ and $X \oplus (Y \oplus Z) = (X \oplus Y) \oplus Z$

Useful Identities involving XOR:
$X \oplus 0 = X$ $X \oplus 1 = X'$ $X \oplus X = 0$ $X \oplus X' = 1$
$X(Y \oplus Z) = XY \oplus XZ$ *(AND distributes over XOR, but XOR does not distribute over AND)*
$X'Y = XY \oplus Y$ Useful for removing complemented variables in an expression
$X + Y = X \oplus Y$ if $XY=0$ In words, "Disjoint Sum of Products can be rewritten as XOR Sum of Products." This is an useful property in forming RMC Network

$X \oplus Y$ = modulo 2 sum of X and Y
$X_1 \oplus X_2 \oplus X_3 \oplus \ldots \oplus X_n$ = modulo 2 sum of n binary variables X_1 to X_n

Reed-Muller Canonical (RMC) Network:
We illustrate RMC network with a small example: The function $F(x_1, x_2, x_3)$ can be uniquely expressed as:
$g_0 \oplus g_1x_1 \oplus g_2x_2 \oplus g_3x_3 \oplus g_{12}x_1x_2 \oplus g_{13}x_1x_3 \oplus g_{23}x_2x_3 \oplus g_{123}x_1x_2x_3$
where, g_i are constants 0 or 1, uniquely determined for a Boolean Function.
Procedure to Obtain RMC Network from K-Maps:

1. Form **disjoint** (non-overlapping) product terms on the K-map.
2. Write the XOR Sum of Products of the circled product terms
3. If there is a complemented variable in a product, say *x'yz,* replace it with *xyz*⊕yz, if complemented variable appears alone, say x', replace it with x⊕1.
4. If two terms are identical, remove them
5. Repeat steps 3 and 4, until all complemented variables are gone.

Example: f(a, b, c) = ∑(0. 1. 2. 6. 7)
Minimal sum with three prime implicants is: *a'b' + bc'+ ab*
There is an overlap between *bc'* and *ab,* namely the minterm *abc'*
A disjoint sum is: *a'b' + bc' + abc*
f(a, b, c) = a'b'⊕bc'⊕abc
= (ab'⊕b')⊕(bc⊕b)⊕abc where a'b' is replaced by ab'⊕b' and bc' is replaced by bc⊕b
= (ab⊕a)⊕(b⊕1) ⊕bc⊕b⊕abc where ab' is replaced by ab⊕a and b' is replaced by b⊕1
=1⊕a⊕ab⊕bc⊕abc

Iterative form of getting RMC from any Boolean Function for computer coding.
Use Shannon's expansion - $F(X_1, X_2, \dots, X_n) = X_1F(1, X_2 \dots, X_n) + X_1'F(0, X_2, \dots, X_n)$
Since the two terms are clearly disjoint, we can substitute XOR for OR.
$F(X_1, X_2, \dots, X_n) = X_1F(1, X_2 \dots, X_n) \oplus X_1'F(0, X_2, \dots, X_n)$ *replace X'F with XF⊕F.*
$F(X_1, X_2, \dots, X_n) = X_1F(1, X_2 \dots, X_n) \oplus X1F(0, X_2, \dots, X_n) \oplus F(0, X_2, \dots, X_n)$
Iterate on each of the n variables using the above form to get the RMC form.

Canonical forms are useful in comparing different implementations of the same Boolean function. Since the canonical form is *unique for a function*, independent of the implementation, we can use it to verify if a particular implementation of the function is error-free. The verification procedure is as follows:

1. Derive the Canonical Form from the Functional Specification
2. Derive the Canonical Form from the implementation without the use of function.

Compare the two Canonical forms – any discrepancy indicates errors in the implementation.

There are several Canonical Forms that we will study:

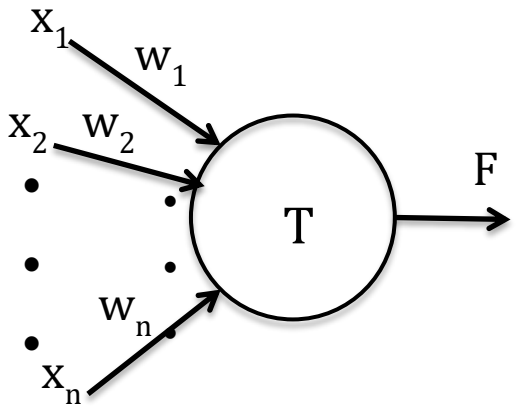
1. Canonical Sum: Sum of all Minterms, also known as *Fundamental SOP*
2. Canonical Product: Product of Maxterms, also known as *Fundamental POS*
3. Reed-Muller Canonical form - RMC
4. Reduced Ordered Binary Decision Diagrams – BDD
5. Complete Sum: Sum of all Prime Implicants

Threshold Functions:

A Threshold Function is a Boolean function that can be expressed as a system of inequality:

$F(x_1, x_2, \ldots, x_n) = 1$ *iff* $w_1x_1 + w_2x_2 + \ldots + w_nx_n \geq T$

where, w_i's and T are real numbers, w_ix_i are arithmetic products and the + is arithmetic sum.
w_i's are called ***weights*** and T the ***threshold***.



Examples: AND gate: $x_1x_2x_3$: $w_1 = w_2 = w_3 = 1$ and $T = 3$. (any number $2 < T \leq 3$ works)
NAND gate: $(x_1x_2x_3)'$: $w_1 = w_2 = w_3 = -1$ and $T = -2.5$. (any number $-3 < T \leq -2$ works)
OR gate: $x_1 + x_2 + x_3$: $w_1 = w_2 = w_3 = 1$ and $T = 1$. (any number $0 < T \leq 1$ works)
NOR gate $(x_1 + x_2 + x_3)'$: $w_1 = w_2 = w_3 = -1$ and $T = -0.5$
Majority: $x_1x_2 + x_1x_3 + x_2x_3$: $w_1 = w_2 = w_3 = 1$ and $T = 2$
Majority function of 2N+1 variables: all $w_i = 1$ and $T = N+1$

Derive weights and threshold for the function $F = x_1 + x_2x_3$

X_1 X_2 X_3	F	Inequality
0 0 0	0	$0 < T$
0 0 1	0	$w_3 < T$
0 1 0	0	$w_2 < T$
0 1 1	1	$w_2 + w_3 \geq T$
1 0 0	1	$w_1 \geq T$
1 0 1	1	$w_1 + w_3 \geq T$
1 1 0	1	$w_1 + w_2 \geq T$
1 1 1	1	$w_1 + w_2 + w_3 \geq T$

These inequalities can be solved by a little bit of trial and error. One possible solution is:
$w_1 = 2$, $w_2 = w_3 = 1$ and $T = 1.5$

Not all Boolean Functions can be implemented as Threshold Functions. The set of inequalities may not have a solution making it impossible to implement a Threshold Function. The following function of two variables shows such a situation.

Derive weights and threshold for the function $F(x_1, x_2) = x_1x_2' + x_1'x_2$

X_1 X_2	F	Inequality
0 0	0	$0 < T$
0 1	1	$w_2 \geq T$
1 0	1	$w_1 \geq T$
1 1	0	$w_1 + w_2 < T$

These set of inequalities are contradictory and hence no solution can exist.

There is no a simple test for Boolean functions to tell us if it can be implemented as Threshold Function. However a necessary condition does exist that can be used to rule out many Boolean Functions. It is known that all threshold functions are unate. However, not all unate functions can be implemented as threshold functions. Here are things we know about threshold functions are listed here.

1. All Threshold Functions are Unate
2. Threshold functions don't have unique weights and threshold. Any threshold function can be implemented using only integer weights and threshold.
3. Complement and Dual of a Threshold Function are also Threshold Functions.

An Unate function has the property that it can be expressed as sum-of-products such that no single variable appears both in true and complemented form. More formally, a function is Unate iff it is positive unate or negative unate in each of its variables.
If F(X1,X2, … , Xi=1, …, Xn) ≥ F(X1,X2, … , Xi=0, … , Xn) then F is Positive Unate in Xi.
If F(X1,X2, … , Xi=0, …, Xn) ≥ F(X1,X2, … , X=1, … , Xn) then F is Negative Unate in Xi.