More Boolean Algebra

(from 12.2)

Definition: Because every Boolean function can be represented using the Boolean operators ·, +, and ⁻, we say that the set {·, + , ⁻} is *functionally complete*.

Functionally complete

The set {·, ⁻} is functionally complete since $x + y = \overline{\bar{x}\bar{y}}$.

The set {+, ⁻} is functionally complete since $xy = \overline{\bar{x} + \bar{y}}$.

nand operator

The *nand* operator (not and), denoted by |, is defined by $1|1 = 0$, and $1|0 = 0|1 = 0|0 = 1$. The set consisting of just the one operator nand {|} is functionally complete. Note that $\bar{x} = x \mid x$ and $xy = (x|y)|(x|y)$.

nor operator

The *nor* operator (not or), denoted by ↓, is defined by $0 \downarrow 0 = 1$, and $1 \downarrow 0 = 0 \downarrow 1 = 1 \downarrow 1 = 0$. The set consisting of just the one operator nor {↓} is functionally complete. (*see Exercises* 15 *and* 16)

Adding binary integers

If we add two one bit integers x and y we get a sum for that bit position plus a carry bit.

If we don't consider a carry bit from a lower bit addition we get what's called a half adder.

If we do consider an input carry bit we have a full adder. (see p. 827, 6th ed.765)

Half Adder

Given input bits x and y, the result bit will be x+y unless both x and y are 1, in which case the result is 0.

This means that we can express the result bit as $(x+y)(\overline{xy})$, or $(x+y)(\bar{x} + \bar{y})$.

The carry bit will be xy (we carry if both x and y are 1)

Full Adder

If we add a carry bit c_0 from the previous order bit sum our result for this bit would be 1 if one or three of c_0, x, y are 1, and 0 otherwise.

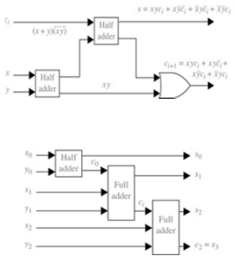
This means
$xyc_0+x\bar{y}\bar{c_0}+\bar{x}y\bar{c_0}+\bar{x}\bar{y}c_0$ would work, with carry bit
$xyc_0+xy\bar{c_0}+x\bar{y}c_0+\bar{x}yc_0$

See p. 827 to check your implementation.

(from page 827)

TABLE 4
Input and Output for the Full Adder.

Input			Output	
x	y	c_i	s	c_{i+1}
1	1	1	1	1
1	1	0	0	1
1	0	1	0	1
1	0	0	1	0
0	1	1	0	1
0	1	0	1	0
0	0	1	1	0
0	0	0	0	0



Minimizing Circuits

A Boolean function can be implemented by many different Boolean expressions.

Disjunctive normal form, the sum-of-products expansion we got from the table of values of the expression, is often not the most efficient.

Minimizing Circuits

For example, the Boolean expression
$x_1\overline{x_2}x_3 + x_1x_2x_3 + \overline{x_1}x_3$
$= x_1(\overline{x_2}+x_2)x_3 + \overline{x_1}x_3$
$= x_1x_3 + \overline{x_1}x_3 = (x_1 + \overline{x_1})x_3 = x_3$

This last expression is a lot easier to compute. No gates required. Much simpler circuit.

Minimizing Circuits

Karnaugh Maps and the Quine-McCluskey Method are used for simplifying Boolean expressions.

See section 12.4.

We'll do some examples on the board.

Karnaugh Maps

	yz	$y\bar{z}$	$\bar{y}\bar{z}$	$\bar{y}z$
x		1	1	
$\bar{x}$	1			1

This Karnaugh map represents the Boolean expression

$xy\bar{z} + x\bar{y}\bar{z} + \bar{x}yz + \bar{x}\bar{y}z$

The idea is to use the picture to help decide which terms can be combined

Karnaugh Maps

The idea is to combine cells which are adjacent horizontally or vertically, since they can be simplified.

The grouped cells can be described by a simpler expression.

Cells on the right edge are adjacent to cells on the left edge, and cells on the top are adjacent to cells on the bottom.

Karnaugh Maps

	yz	$y\bar{z}$	$\bar{y}\bar{z}$	$\bar{y}z$
x		1	1	
$\bar{x}$	1			1

If we group the adjacent cells in the top and bottom row we get the expression

$x\bar{z} + \bar{x}\bar{y}$, which is simpler than

$xy\bar{z} + xy\bar{z} + \bar{x}yz + \bar{x}\bar{y}z$

Karnaugh Maps, 4 variables

	yz	$y\bar{z}$	$\bar{y}\bar{z}$	$\bar{y}z$
wx			1	
$w\bar{x}$	1	1	1	
$\bar{w}\bar{x}$		1	1	
$\bar{w}x$			1	

Here the grouping shows we can express the sum as $\bar{y}\bar{z} + \bar{x}\bar{z} + w\bar{x}y$

Karnaugh Maps, 4 variables

	yz	$y\bar{z}$	$\bar{y}\bar{z}$	$\bar{y}z$
wx			1	
$w\bar{x}$	1	1	1	
$\bar{w}\bar{x}$		1	1	
$\bar{w}x$			1	

Nonoverlapping grouping would give $w\bar{x}yz + \bar{x}y\bar{z} + \bar{y}\bar{z}$, more complex than $\bar{y}\bar{z} + \bar{x}\bar{z} + w\bar{x}y$

Karnaugh Maps

See the book, page 836,

for a description of

"don't care conditions".

Sometimes some cells in a Karnaugh map are irrelevant to our purpose and we can include them or not, which can allow a simpler result.

Quine-McCluskey Method

This method is a little trickier to describe, but can be automated for larger Boolean expressions, a real advantage.

Quine-McCluskey

The idea is to make a table, with a row for each term in the original Boolean expression.

We form a bit string for each term, e.g. 110 for $xy\bar{z}$, and arrange the rows by order of decreasing number of 1s.

Then we combine terms differing in only one position (1 & 0), form another column, and repeat if possible.

Quine-McCluskey

Once we have simplified expressions we make another table to see how many of these we need to actually cover the original terms.

More than one simplification might do the job.

See the book, pp 837-843, and the examples we'll do on the board.

See also

http://en.wikipedia.org/wiki/Karnaugh

http://en.wikipedia.org/wiki/Gray_code

http://en.wikipedia.org/wiki/Quine-McCluskey_algorithm