

2.1 Primitives

Friday, February 9, 2001

AND:	x	y	$x * y$
	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

NOT:	x	\bar{x}
	0	1
	1	0

2.2 DeMorgan's Theorem

Proof of DeMorgan's Theorem:

x	y	$x + y$	$\overline{(x + y)}$	\bar{x}	\bar{y}	$\bar{x} * \bar{y}$
0	0	0	1	1	1	1
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	0

x	y	$x * y$	$\overline{(x * y)}$	\bar{x}	\bar{y}	$\bar{x} + \bar{y}$
0	0	0	1	1	1	1
0	1	0	1	1	0	1
1	0	0	1	0	1	1
1	1	1	0	0	0	0

2.3 Identities

Elementary:

$A * 0 = 0$	$A + 1 = 1$
$A * 1 = A$	$A + 0 = A$
$A * A = A$	$A + A = A$
$A * \bar{A} = 0$	$A + \bar{A} = 1$

Commutative:

$A * B = B * A$	$A + B = B + A$
-----------------	-----------------

Distributive:

$A * (B + C) = A * B + A * C$	$A + (B * C) = (A + B) * (A + C)$
-------------------------------	-----------------------------------

Absorption:

$A * (A + B) = A$	$A + (A * B) = A$
-------------------	-------------------

Nameless:

$A * (\bar{A} + B) = A * B$	$A + (\bar{A} * B) = A + B$
-----------------------------	-----------------------------

Consensus:

$(A + B) * (\bar{A} + C) * (B + C)$	$A * B + \bar{A} * C + B * C$
$= (A + B) * (\bar{A} + C)$	$= A * B + \bar{A} * C$

2.4 Duality and DeMorgan

Equals is by DeMorgan's Theorem

$$F(A, B, 0, 1, *, +) = \bar{F}(\bar{A}, \bar{B}, 1, 0, +, *)$$

Duality

Duality

$$F_d(A, B, 1, 0, +, *) = \bar{F}_d(\bar{A}, \bar{B}, 0, 1, *, +)$$

2.5 Mass. Stoplight Check

Massachusetts Stoplight Check Function
(F = 1 implies legal state.)

	r	y	g	F
	0	0	0	0
Truth Table:	0	0	1	1
Allowed combinations	0	1	0	1
of lights lit are red,	0	1	1	0
yellow, green, and red	1	0	0	1
+ yellow.	1	0	1	0
	1	1	0	1
	1	1	1	0

2.6 Standard Sum of Products

Standard Sum of Products (Or of Ands)

$$\begin{aligned}
 F &= \bar{r} * \bar{y} * g + \bar{r} * y * \bar{g} + r * \bar{y} * \bar{g} + r * y * \bar{g} \\
 &= \bar{r} * \bar{y} * g + \bar{g} * (\bar{r} * y + r * \bar{y} + r * y) \\
 &= \bar{r} * \bar{y} * g + \bar{g} * (\bar{r} * y + r * (\bar{y} + y)) \\
 &= \bar{r} * \bar{y} * g + \bar{g} * (\bar{r} * y + r) \\
 &= \bar{r} * \bar{y} * g + \bar{g} * (r + y) \\
 &= \bar{r} * \bar{y} * g + r * \bar{g} + y * \bar{g}
 \end{aligned}$$

Or, using DeMorgan:

$$\begin{aligned}
 F &= \overline{(\bar{r} * \bar{y} * g)} * \overline{(\bar{g} * (r + y))} \\
 &= (r + y + \bar{g}) * (g + r * y)
 \end{aligned}$$

2.7 Standard Product of Sums

Standard Product of Sums (And of Ors)

$$\begin{aligned}
 F &= (r + y + g) * (r + \bar{y} + \bar{g}) * (\bar{r} + y + \bar{g}) * (\bar{r} + \bar{y} + \bar{g}) \\
 &= (r + y + g) * (\bar{g} + (r + \bar{y}) * (\bar{r} + y) * (\bar{r} + \bar{y})) \\
 &= (r + y + g) * (\bar{g} + (r + \bar{y}) * \bar{r}) \\
 &= (r + y + g) * (\bar{g} + \bar{r} * \bar{y}) \\
 &= (r + y + g) * (\bar{g} + \bar{r}) * (\bar{g} + \bar{y})
 \end{aligned}$$

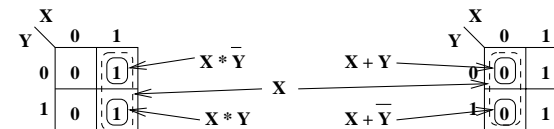
By DeMorgan:

$$F = \bar{r} * \bar{y} * \bar{g} + g * (r + y)$$

2.8 Karnaugh Maps

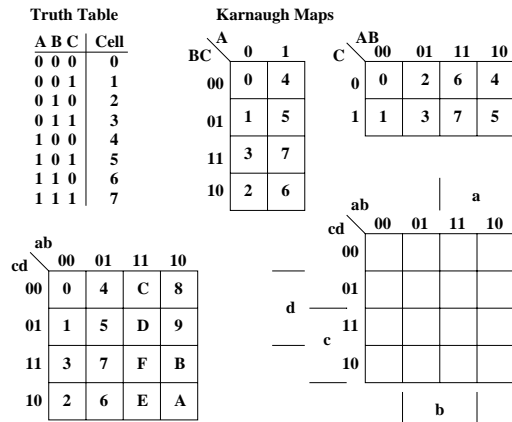
Karnaugh Maps are:

a simple remapping of truth tables and
a graphical means of reducing logic equations.

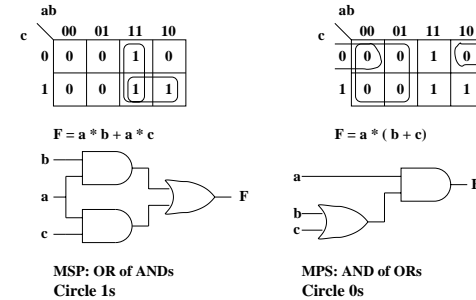


2.9 Use of K Maps

Karnaugh Maps are useful for 3–6 variables.
(Though HARD for > 4 variables.)
Adjacent cells have a one bit change, like a Gray Code.

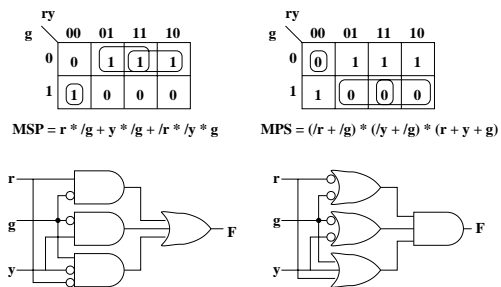


2.10 Circling Groups



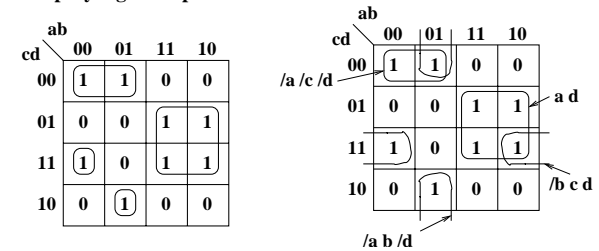
2.11 Stoplight Check Function

Massachusetts Stoplight Check Function

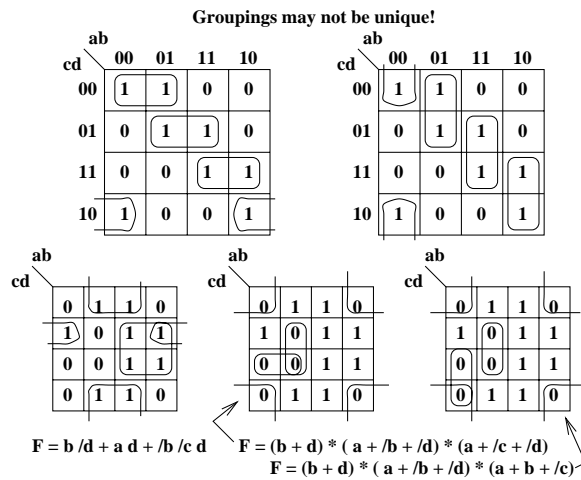


2.12 Simple Circles

The simplest groups are the largest: this is how we can use K-maps to simplify logical expressions.

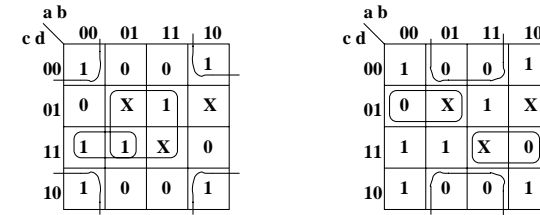


2.13 Uniqueness?



2.14 Don't Cares

Don't Cares can simplify things: (impossible inputs, for example).



$$MSP = /b /d + b d + /a c d \quad MPS = (/b + d) * (/a + /c + /d) * (a + c + /d)$$

Here abcd = 0101, 1111, and 1001 are don't cares.

Note that MSP != MPS.

2.15 XOR

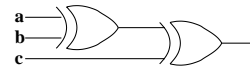
Now, there are some functions you can't do very much with:

Like this one: a parity function

	ab	00	01	11	10
c	0	0	1	0	1
1	1	1	0	1	0

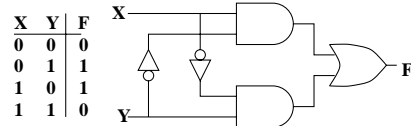
$$\begin{aligned}
 F &= \bar{a} \bar{b} \bar{c} + a \bar{b} \bar{c} + \bar{a} \bar{b} c + a \bar{b} c \\
 &= (\bar{a} \bar{b} + a \bar{b}) * \bar{c} + (\bar{a} \bar{b} + a \bar{b}) * c \\
 &= (\bar{a} \oplus b) * \bar{c} + (\bar{a} \oplus b) * c \\
 &= (\bar{a} \oplus b) \oplus c
 \end{aligned}$$

It can be implemented with this (new) function, the exclusive OR.

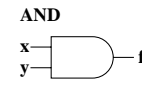


Exclusive OR

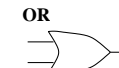
$$F = X \oplus Y$$



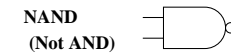
2.16 Gate Symbols



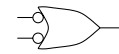
x	y	f
0	0	0
0	1	0
1	0	0
1	1	1



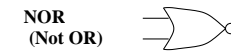
x	y	f
0	0	0
0	1	1
1	0	1
1	1	1



NAND
(Not AND)



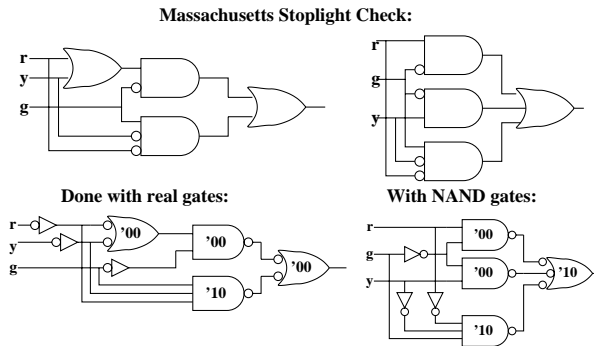
x	y	f
0	0	1
0	1	1
1	0	1
1	1	0



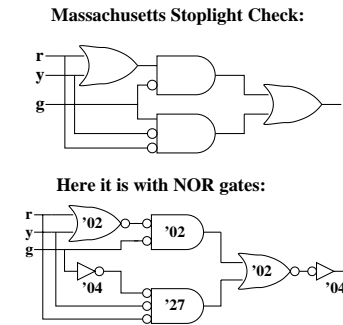
NOR
(Not OR)

x	y	f
0	0	1
0	1	0
1	0	0
1	1	0

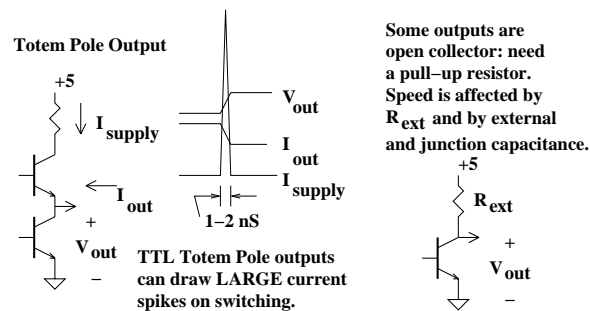
2.17 Mass. Stoplight with NANDS



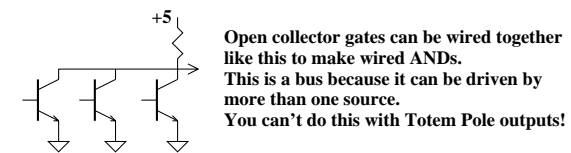
2.18 Mass. Stoplight with NORS



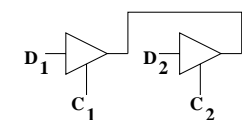
2.19 Totem Pole Output



2.20 Busses



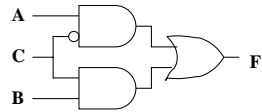
By controlling the gates on both transistors of a Totem Pole to be open, a high impedance is created (this is a tri-state). Control inputs C_1 and C_2 are output enables.



2.21 Hazards

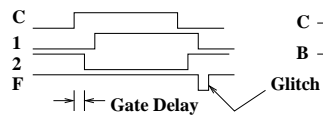
Static Hazards: Consider this function:

$$F = A * \bar{C} + B * C$$

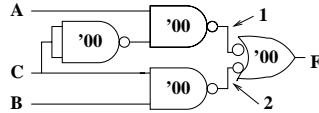


C	AB			
	00	01	11	10
0	0	0	1	1
1	0	1	1	0

$$A = B = 1$$

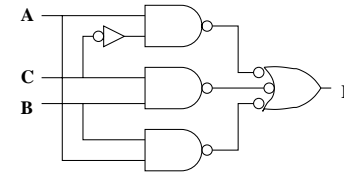


Implemented with MSI gates:



2.22 Fixing Hazards

The glitch is the result of timing differences in parallel data paths. It is associated with the function jumping between groupings or product terms on the K-map. To fix it, cover it up with another grouping or product term!



C	AB			
	00	01	11	10
0	0	0	1	1
1	0	1	1	0

$$F = A * \bar{C} + B * C + A * B$$