Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.111 Introductory Digital Systems Laboratory

Lecture # 2

Now some basics (This IS about digital logic...)

The values here (x and y) represent something like voltage (is it +5 volts (1) or zero (0)? Or is a light ON or OFF?

(That is, anything that can take on one of two values)

Slic

AND:

x	y	$x \bullet y$
0	0	0
0	1	0
1	0	0
1	1	1

OR:

x	y	$x \bullet y$
0	0	0
0	1	1
1	0	1
1	1	1

NOT:

x	$\bar{x}$
0	1
1	0

Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.111 Introductory Digital Systems Laboratory

Identities:

Boolean Algebra

Elementary:

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

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:

$A * (\overline{A} + B) = A * B$	$A + (\overline{A} * B) = A + B$

Concensus:

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

Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.111 Introductory Digital Systems Laboratory

DeMorgan's Theorem:

$$\overline{A \bullet B \bullet \ldots} = \overline{A} + \overline{B} + \ldots$$

$$\overline{A + B + \ldots} = \overline{A} \bullet \overline{B} \bullet \ldots$$

Duality:

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

$$F_d\ (A, B, 1, 0, +, *) = \overline{F_d}\ (\overline{A}, \overline{B}, 0, 1, *, +)$$

Proof of DeMorgan's Theorem:

x	y	$x+y$	$\overline{(x+y)}$	$\overline{x}$	$\overline{y}$	$\overline{x} \bullet \overline{y}$
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	$x \bullet y$	$\overline{(x \bullet y)}$	$\overline{x}$	$\overline{y}$	$\overline{x} + \overline{y}$
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

Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science

6.111 Introductory Digital Systems Laboratory

Massachusetts Stoplight Example

F=1 implies stoplight is working correctly

F=0 implies stoplight is busted

Slide 3

Truth Table:

r	y	g	F	F =
0	0	0	0	
0	0	1	1	/r*/y*g +
0	1	0	1	/r*y*/g +
0	1	1	0	
1	0	0	1	r*/y*/g +
1	0	1	0	
1	1	0	1	r*y*/g
1	1	1	0	

Obsolete Stoplight Example: Reduction using Boolean Algebra

F = R * /Y * /G + /R * Y * /G + /R * /Y * G + R * Y * /G

Step 1: Since Y + /Y = 1,
Slide 3
R*/Y*/G + R*Y*/G = R*(Y + /Y) * /G = R * /G

F = R * /G + /R * Y * /G + /R * /Y * G

Step 2: Use Absorption: R + /R * Y = R + Y
R*/G + /R * Y * /G = (R + /R * Y) * /G = (R + Y) * /G

F = (R + Y) * /G + /R * /Y * G = R * /G + Y * /G + /R * /Y * G

Using Demorgan:

/F = ((/R * /Y) + G) * (/G + (R * Y)) = /R*/Y*/G + G * (R + Y)

Truth Table:

r	y	g	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Or look at the zeros:

/F = /r*/y*/g + /r*y*g + r*/y*g + r*y*g

Slide 3

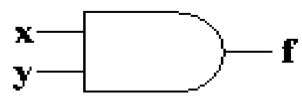
Which, by Demorgan (Duality) is:

F = (/r + /y + /g)*(/r + y + g)*(r + /y + g)*(r + y + g)

Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.111 Introductory Digital Systems Laboratory

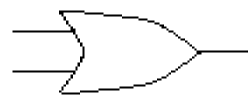
Common Logic Functions and Gate Symbols

AND



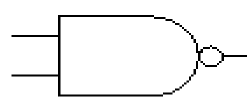
x	y	f
0	0	0
0	1	0
1	0	0
1	1	1

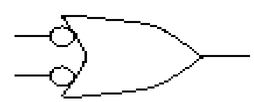
OR



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

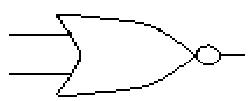
NAND
(Not AND)

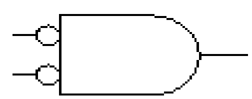




x	y	f
0	0	1
0	1	1
1	0	1
1	1	0

NOR
(Not OR)



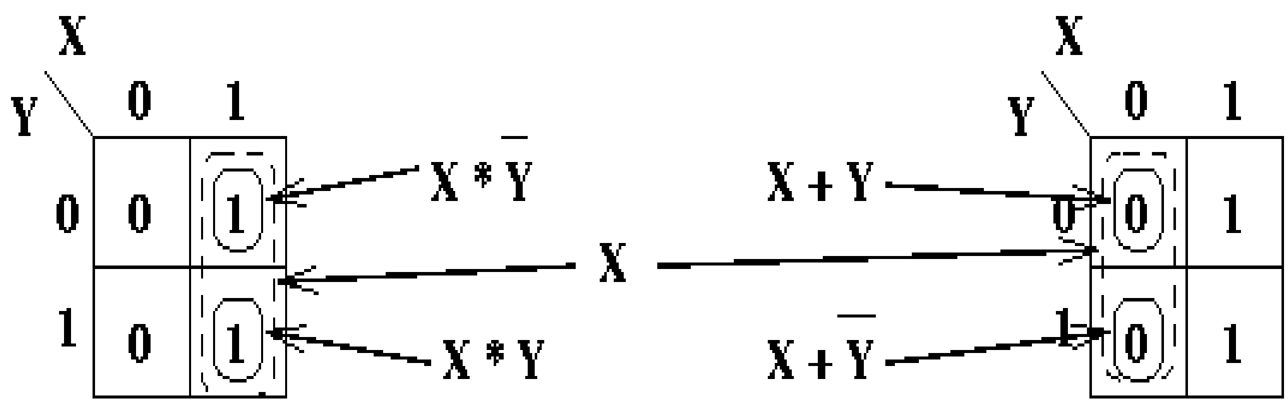


x	y	f
0	0	1
0	1	0
1	0	0
1	1	0

Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.111 Introductory Digital Systems Laboratory

Karnaugh Maps are:

1. A simple re-mapping of truth tables
2. A graphical means of reducing logic functions

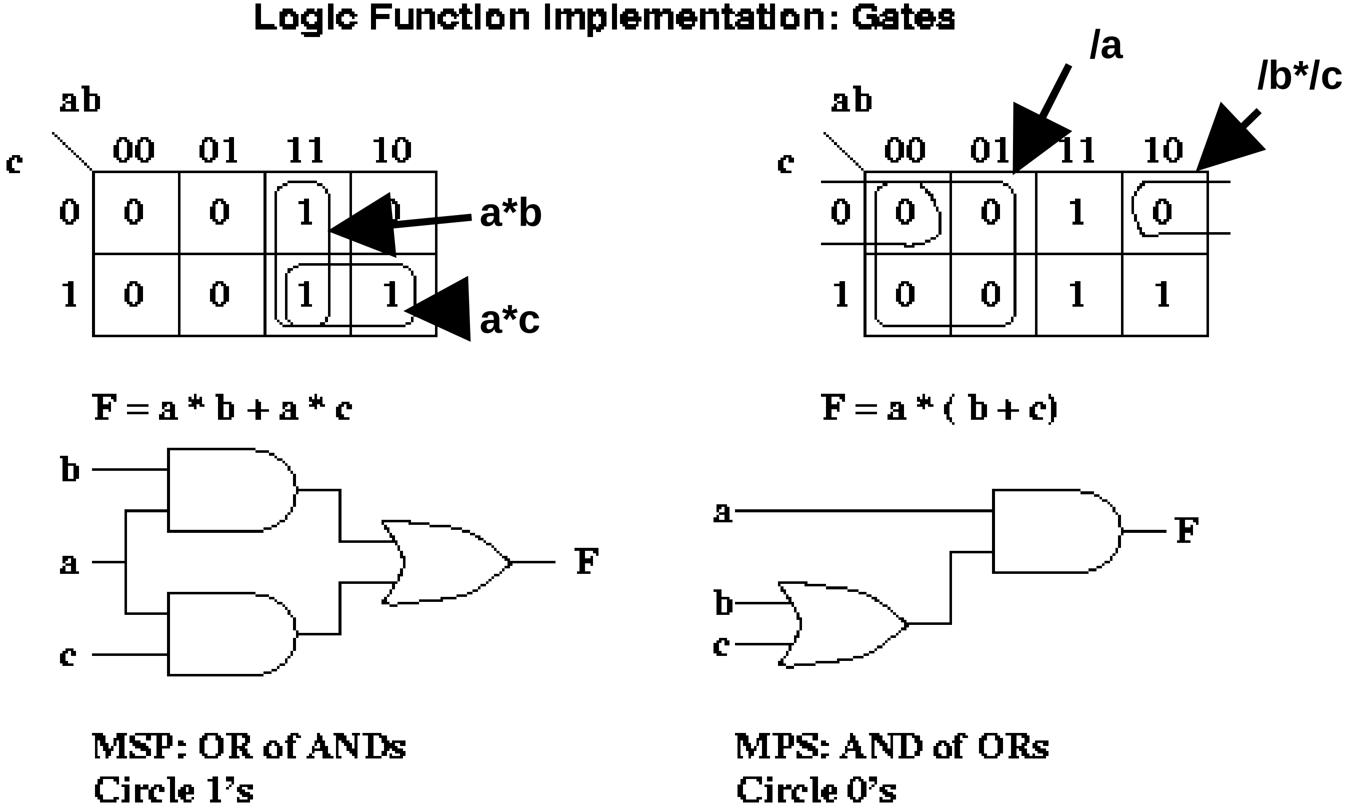


$$X = X * Y + X * \overline{Y}$$

$$\overline{X} = (\overline{X} + Y) * (\overline{X} + \overline{Y})$$

Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.111 Introductory Digital Systems Laboratory

Logic Function Implementation: Gates



F = a*b + a*c = a*(b+c)

/F = /(/a + /(b+c)) = /(/a + (/b * /c))

F = /a + /b * /c

K- maps are useful for 3-6 variables (HARD for > 4!)
Adjacent cells have one bit change, like a Gray Code

Karnaugh Maps

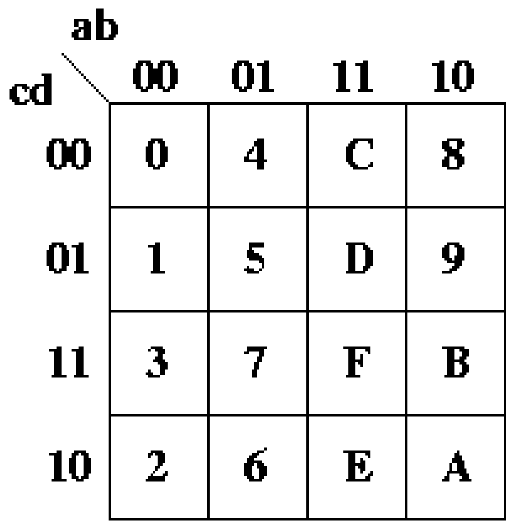
Truth Table

A B C	Cell
0 0 0	0
0 0 1	1
0 1 0	2
0 1 1	3
1 0 0	4
1 0 1	5
1 1 0	6
1 1 1	7

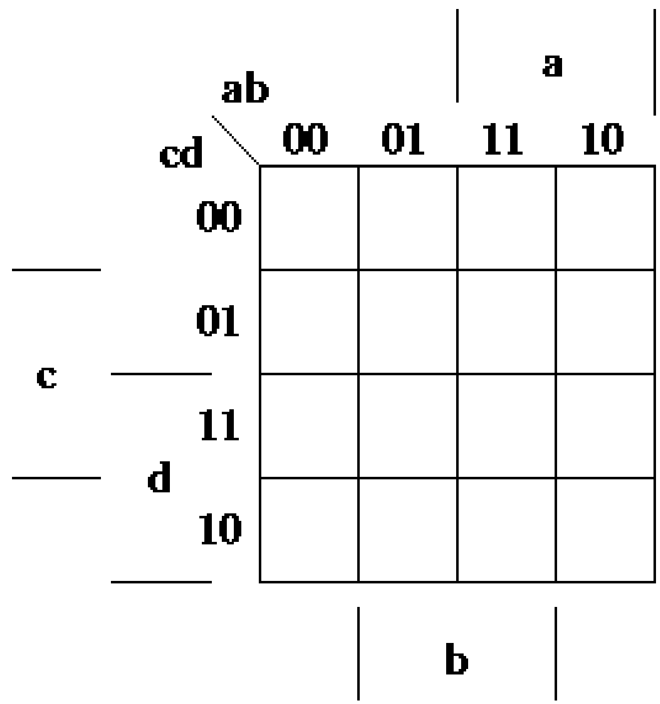
BC \ A	0	1
00	0	4
01	1	5
11	3	7
10	2	6

C \ AB	00	01	11	10
0	0	2	6	4
1	1	3	7	5

Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.111 Introductory Digital Systems Laboratory



4- Input K map



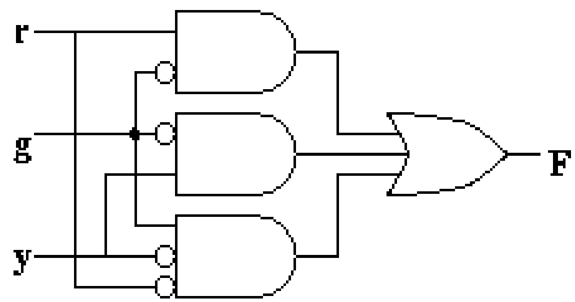
Inputs group like this

Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.111 Introductory Digital Systems Laboratory

Massachusetts Stoplight Check Function

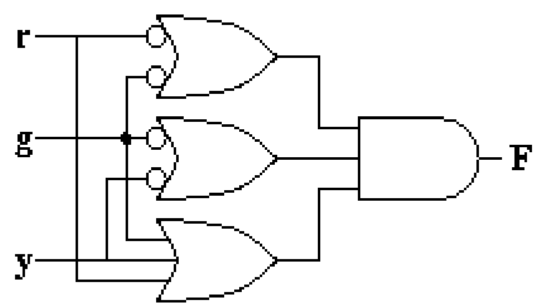
g \ ry	00	01	11	10
0	0	1	1	1
1	1	0	0	0

MSP = r * /g + y * /g + /r * /y * g

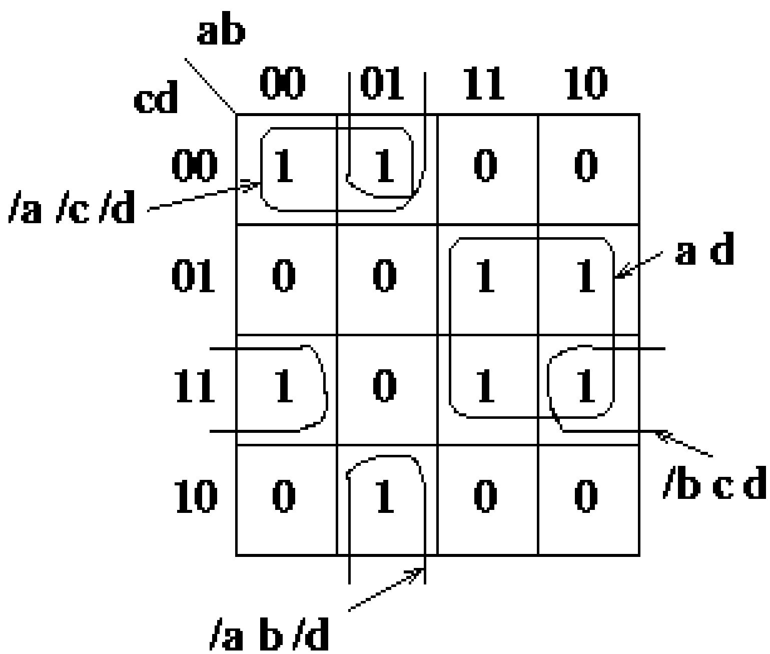


g \ ry	00	01	11	10
0	0	1	1	1
1	1	0	0	0

MPS = (/r + /g) * (/y + /g) * (r + y + g)

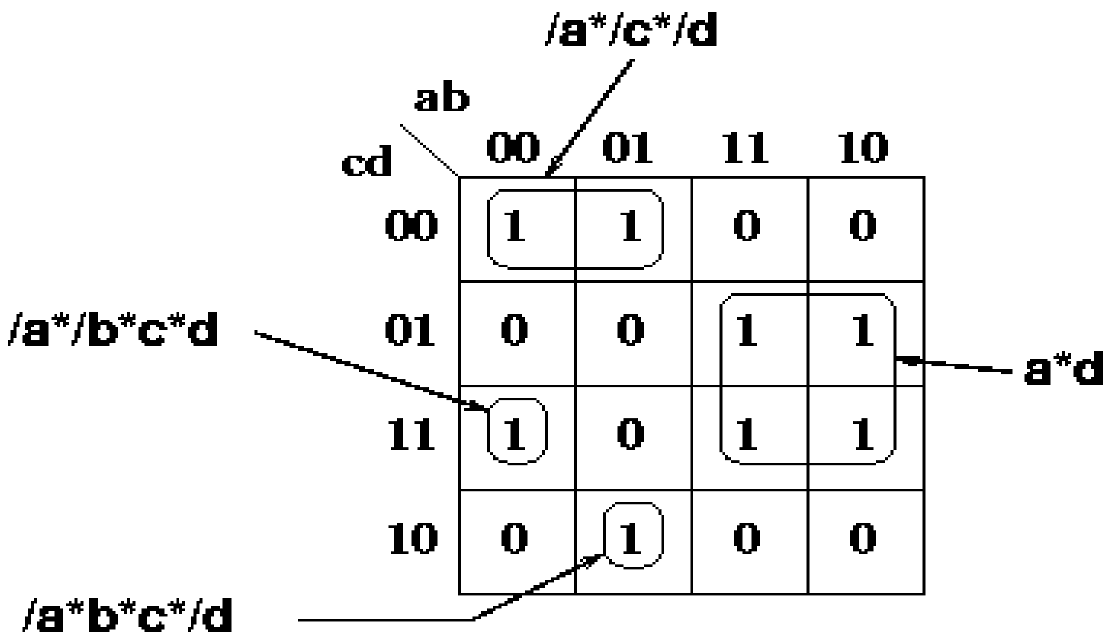


The simplest groups are the largest: this is how we can use K-maps to simplify logical expressions



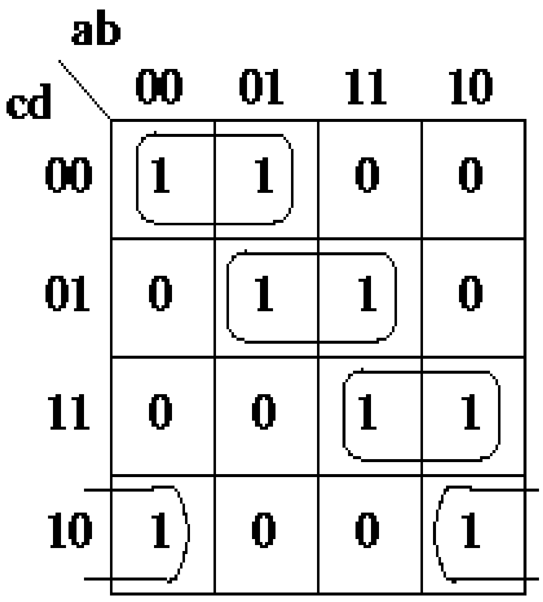
Simplest Groupings are the largest

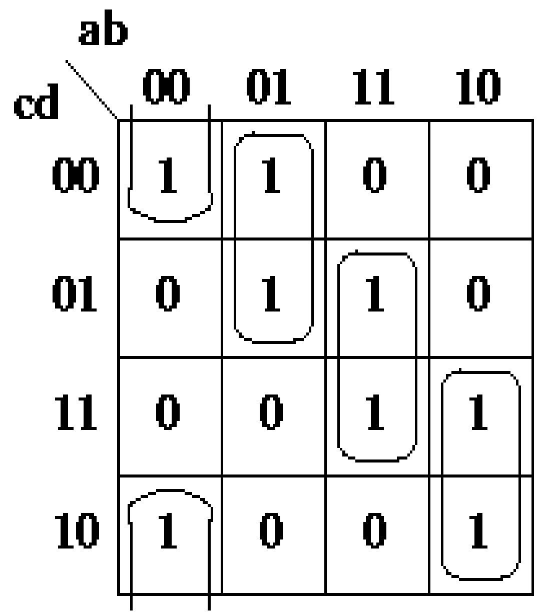
This one is more complex than need be!



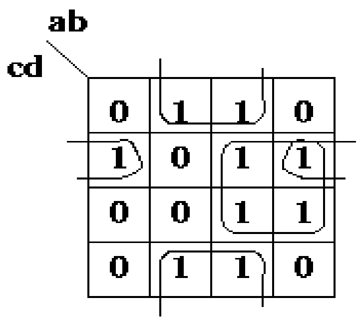
Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.111 Introductory Digital Systems Laboratory

Groupings may not be unique!

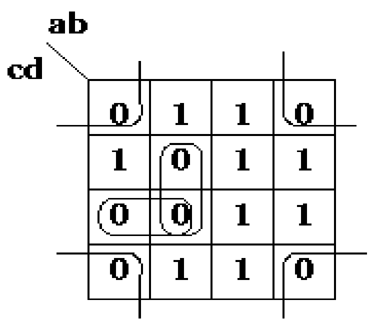




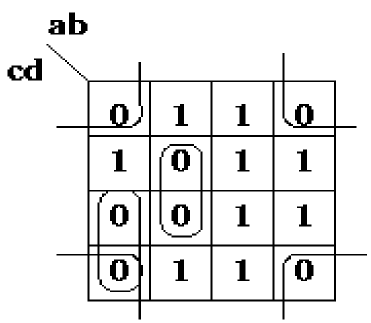
Or MSP may be unique and MPS not, or vice versa



F = b /d + a d + /b /c d



F = (b + d) * (a + /b + /d) * (a + /c + /d)

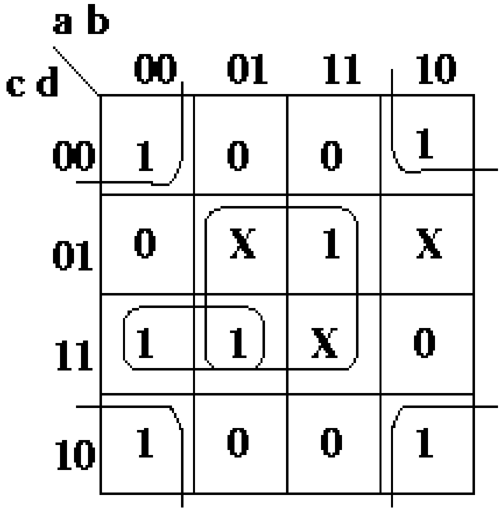


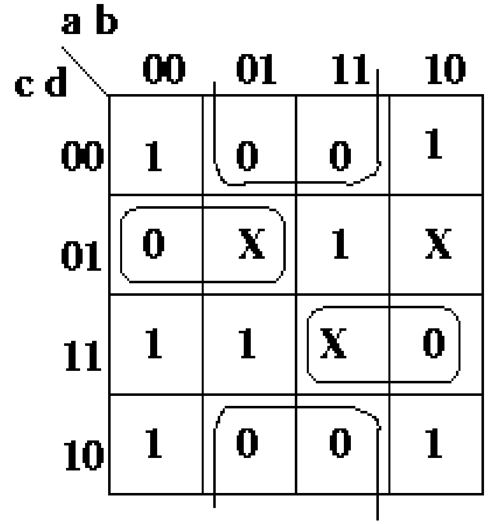
F = (b + d) * (a + /b + /d) * (a + b + /c)

Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science

6.111 Introductory Digital Systems Laboratory

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





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

Here abcd = 0101, 1111 and 1001 are "don't care"s

Note that MSP may not equal MPS (and doesn't here)

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

Like this one: a "parity" function

$$F = \bar{a}\,b\,\bar{c} + a\,\bar{b}\,\bar{c} + \bar{a}\,\bar{b}\,c + a\,b\,c$$

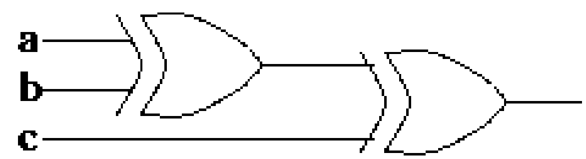
$$= (\bar{a}\,b + a\,\bar{b}) * \bar{c} + (\bar{a}\,\bar{b} + a\,b) * c$$

$$= (a \oplus b) * \bar{c} + \overline{(a \oplus b)} * c$$

$$= (a \oplus b) \oplus c$$

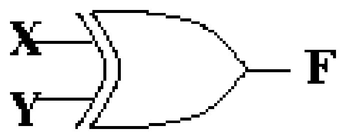
c \ ab	00	01	11	10
0	0	1	0	1
1	1	0	1	0

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

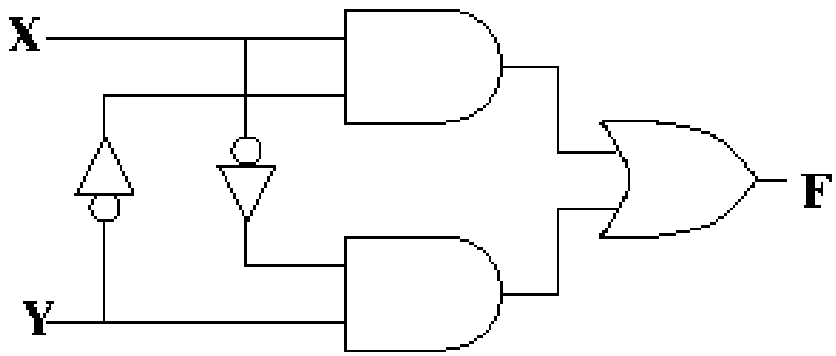


Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.111 Introductory Digital Systems Laboratory

Exclusive OR $F = X \oplus Y$



X	Y	F
0	0	0
0	1	1
1	0	1
1	1	0



if file foo.txt contains:

```
a = (x + z) * (/x + y) * (z + y);
b = a ^ c;
d = x * a;
```


then if you do:

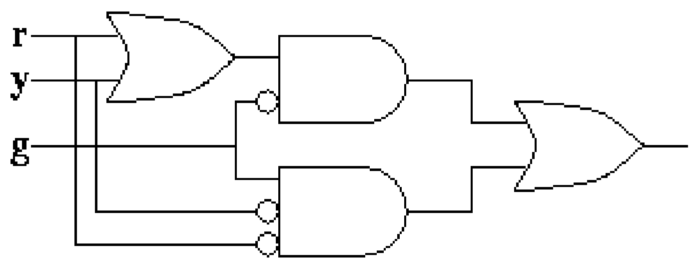
```
reduce -b < foo.txt > foo_out.txt
```

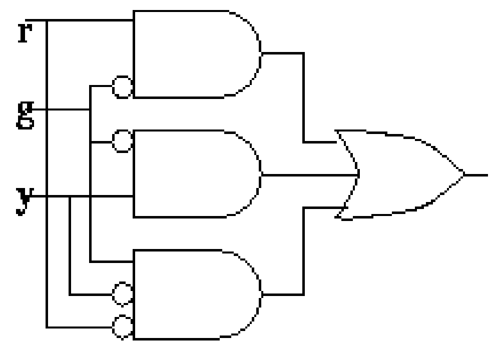
you get in foo.out:

```
a = x * y + /x * z;
/a = x * /y + /x * /z;
b = a * /c + /a * c;
/b = /a * /c + a * c;
d = x * a;
/d = /a + /x;
```

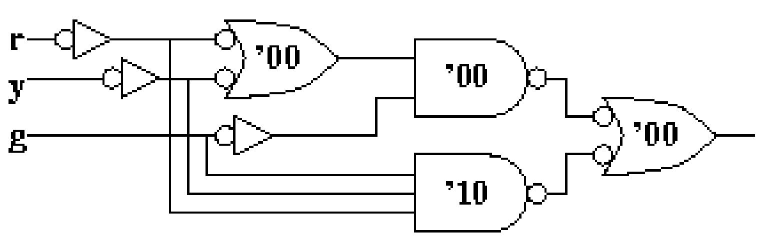
Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.111 Introductory Digital Systems Laboratory

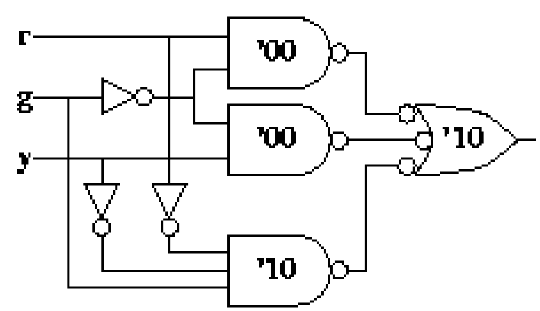
Massachusetts Stoplight Check:





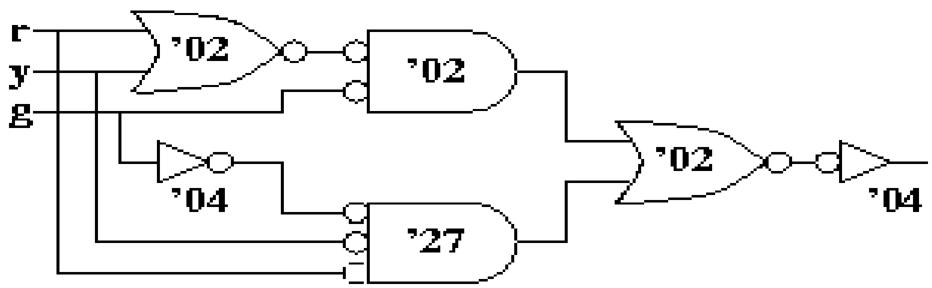
Done with real gates: NAND’s



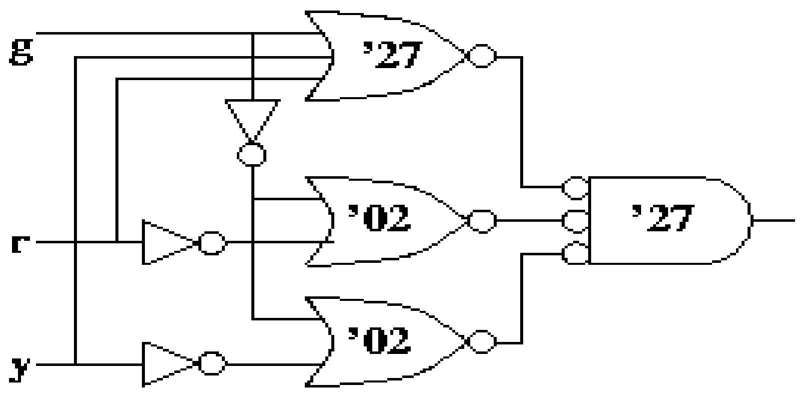


Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.111 Introductory Digital Systems Laboratory

Here it is with NOR gates



MPS = (r + y + g) * (/g + /r) * (/g + /y)

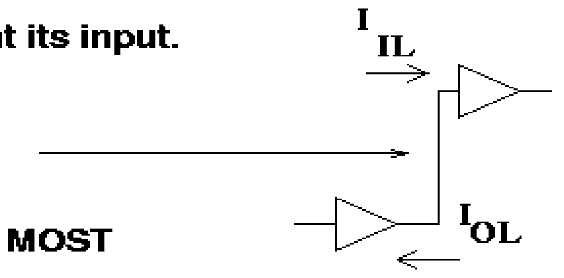


Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.111 Introductory Digital Systems Laboratory

TTL logic requires current at its input.
This leads to loading limits.

Here is the sign convention

Logical LOW dominates for MOST gates.



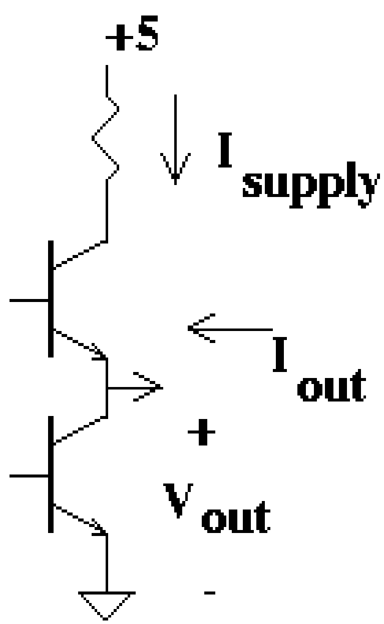
Current (mA)

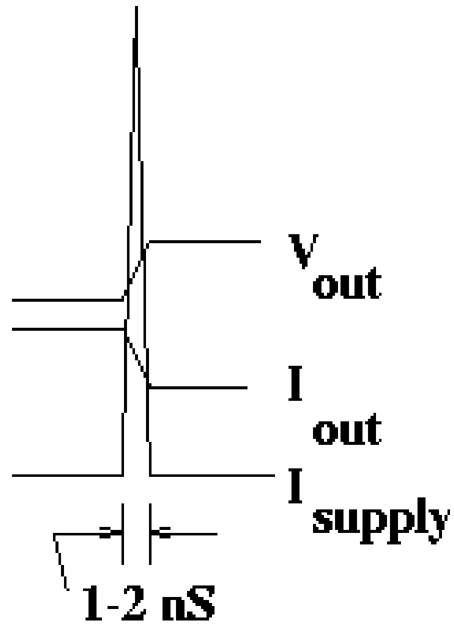
	74LS	74S	74
Output Capability I_{OL}	8	20	16
Input Required I_{IL}	-0.4	-2	-1.6
Equivalent LS Inputs	1	5	4
Output Can Drive LS Inputs	20	50	40
74 LS Can Drive	20	4	5

These are typical numbers, but there are many exceptions.
i.e. Read the data sheets to be sure.

Voltage Levels For TTL:	Input	Output
High	> 2.0	> 2.7
Low	< 0.8	< 0.4

Totem Pole Output (Common for TTL)

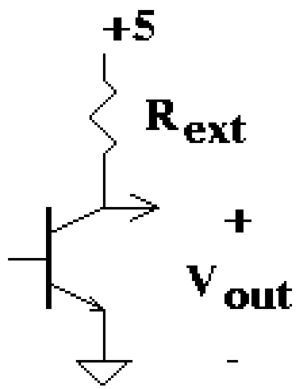




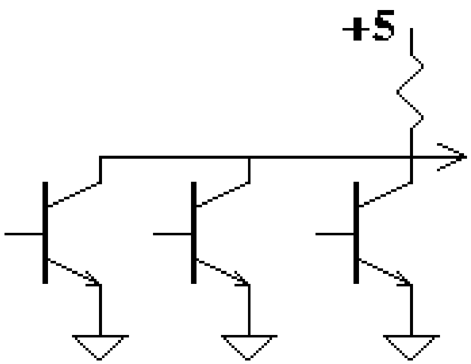
TTL Totem Pole Outputs can draw LARGE current spikes on switching

Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.111 Introductory Digital Systems Laboratory

Some outputs are open collector: need a pull-up resistor. Speed is affected by R_{ext} and by external and junction capacitance



Open collector gates can be wired together like this to make 'wired AND's.



This is a 'bus' that can be driven by more than one input source

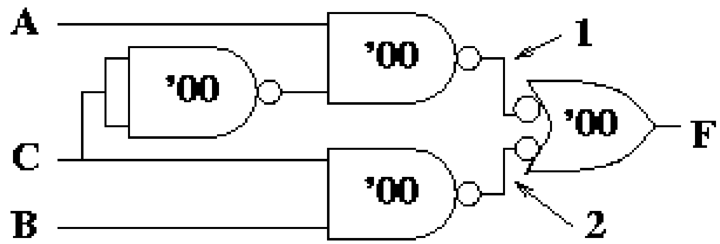
You can't do this with Totem Pole outputs!

Static Hazards: Consider this function:

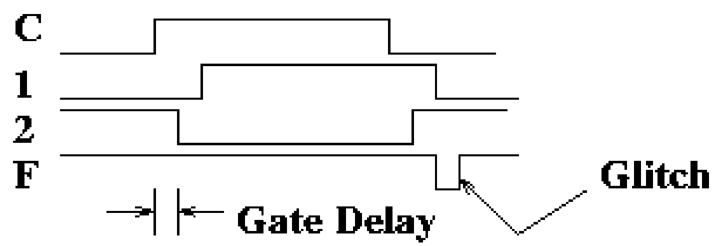
$$F = A * \overline{C} + B * C$$

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

Implemented with MSI gates:



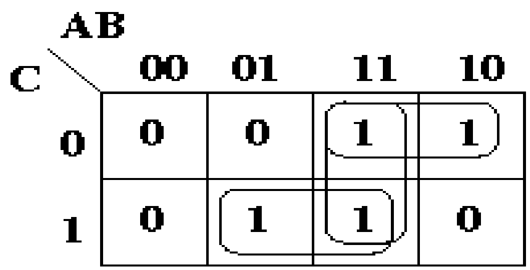
Consider this transient: $A = B = 1$



The 'glitch is the result of timing differences in parallel data paths.

Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.111 Introductory Digital Systems Laboratory

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



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