BEE 271 Digital circuits and systems

Spring 2017

Lecture 3: Karnaugh maps and Verilog

Nicole Hamilton

https://faculty.washington.edu/kd1uj

Topics

1. *Review:* Binary numbers and Boolean algebra
2. minterms and Maxterms
3. Sum of products
4. Product of sums
5. Karnaugh maps
6. Verilog

Numbers are positional

In decimal

1492

1's
10's
100's
1000's

In binary

1011

1's
2's
4's
8's

Numbers are positional

In decimal

1492

10^0

10^1

10^2

10^3

In binary

1011

$2^0 = 1$

$2^1 = 2$

$2^2 = 4$

$2^3 = 8$

Adding zeroes to the left doesn't change the value.

In decimal

001492 = 1492

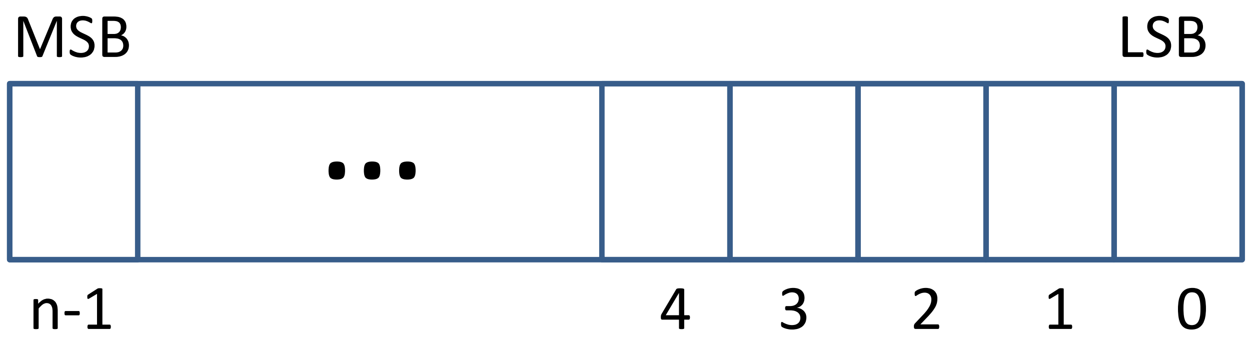
In binary

001011 = 1011

When we add numbers we get carries.

In decimal	In binary
110	*011*
1492	1011
+ 525	+ 011
2017	1110

Binary numbers

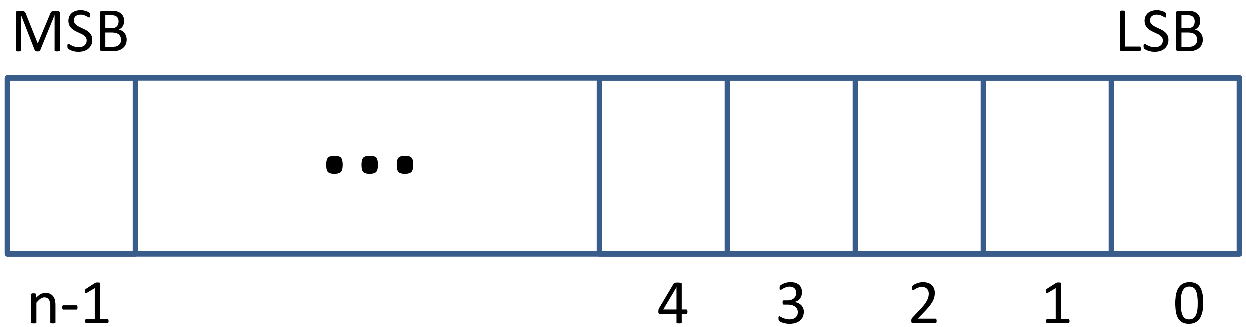


Numbering of the individual bits is from least significant bit (LSB) to most significant bit (MSB).

If b0 = 0, the number is even.
If b0 = 1, the number is odd.

Each bit represents a power of 2.

Value of a binary number



$$Value = \sum_{i=0}^{n-1} b_i 2^i$$

Hex

1. Hard to read long strings of nothing but 1’s and 0’s.
2. So we break it up into groups of 4 bits called *nibbles*, starting *at the LSB*.
3. Take each 4-bit group as a value from 0 to 15.
4. Values 10 to 15 written as A to F.

0111010010011111

0111 0100 1001 1111

7 4 9 F

Binary	Decimal	Hex
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

In hex

A12D

$16^0 = 1$

$16^1 = 16$

$16^2 = 256$

$16^3 = 4096$

A12D

$$13 * 16^0 = 13$$
$$2 * 16^1 = 32$$
$$1 * 16^2 = 256$$
$$10 * 16^3 = 40960$$

$$41261$$

Chapter 2

Introduction to Logic Circuits

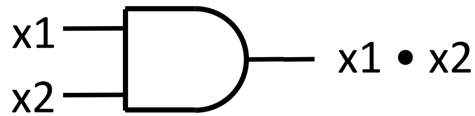
Boolean Algebra

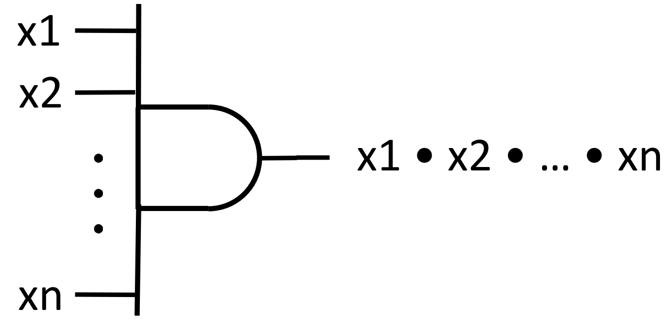
Values	0, 1
Variables	a, b, c, X, Y, s0, s1, DoorOpen, ..
Operations	NOT, AND, OR, XOR

Operation	*Written as*
NOT a	$\overline{a}$ or a’
a AND b	a • b or a b
a OR b	a + b
a XOR b	a ^ b = a b’ + a’ b

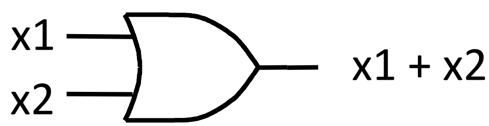
The basic gates.

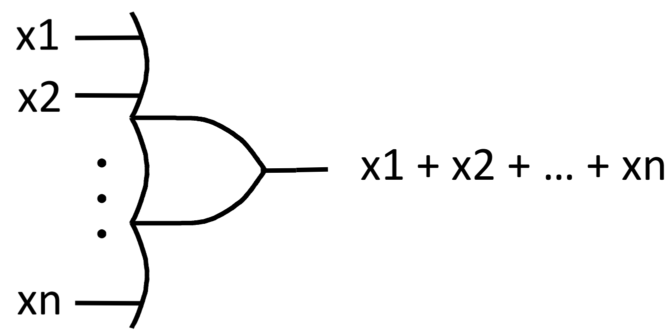
AND If all inputs are true, the output is true.



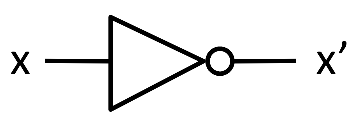


OR If any input is true, the output is true.

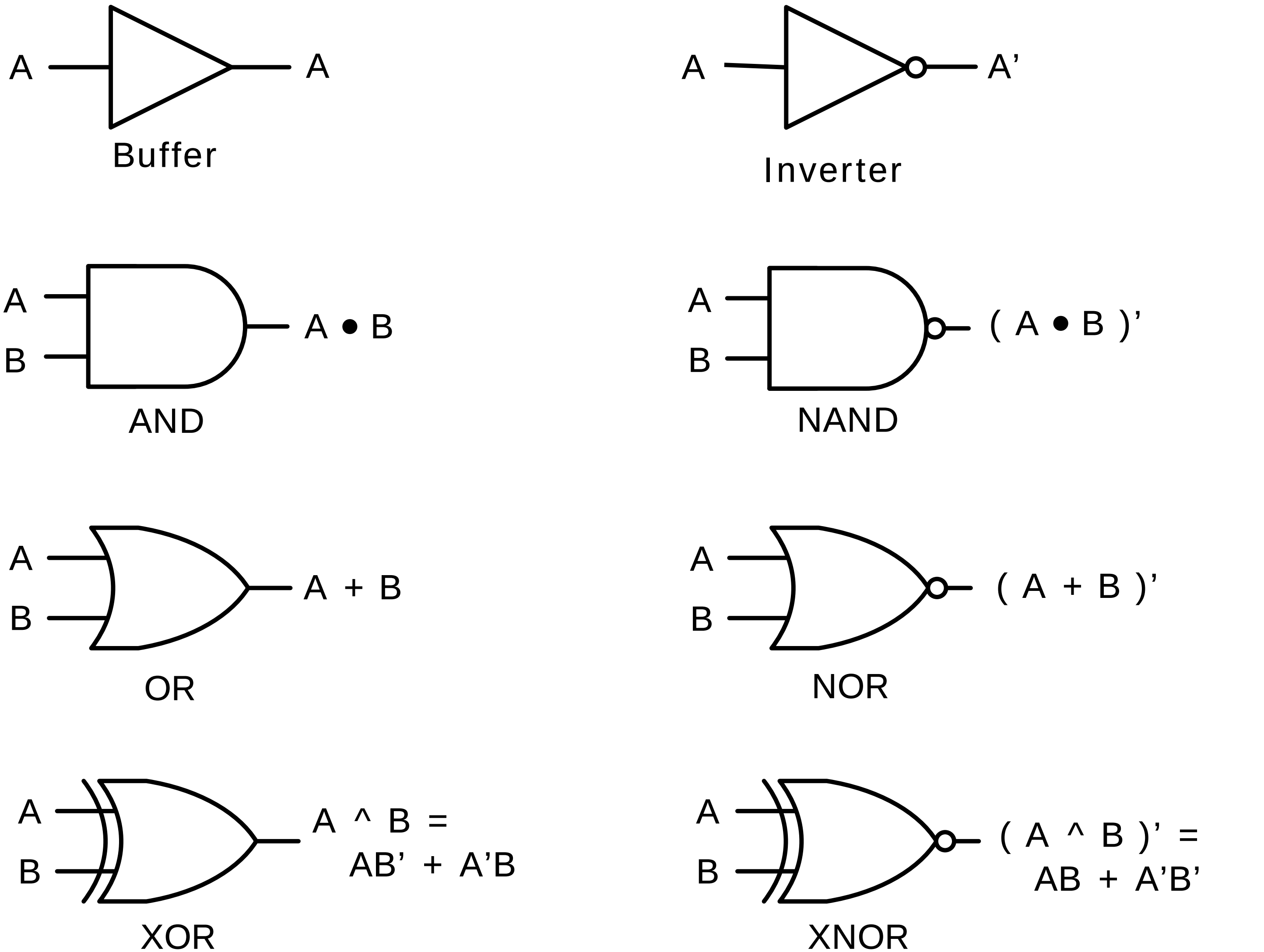




NOT The output is the inverse of the input.



A more complete set of gates



Truth tables

We describe Boolean functions with truth tables.

a	b	a AND b
0	0	0
0	1	0
1	0	0
1	1	1

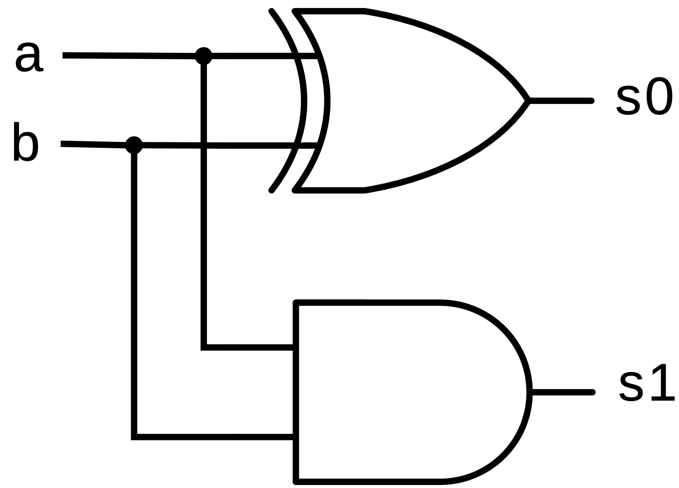
a	b	a OR b
0	0	0
0	1	1
1	0	1
1	1	1

a	b	a XOR b
0	0	0
0	1	1
1	0	1
1	1	0

a	NOT a
0	1
1	0

a	0	0	1	1
+b	+0	+1	+0	+1
s1 s0	0 0	0 1	0 1	1 0

a	b	s1	s0
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



Addition of one-bit binary numbers.

Truth tables

Deriving Boolean equations from truth tables:

a	b	s1	s0
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

OR together *product* terms for each truth table row where the function is 1.

If input variable is 0, it appears in complemented form; if 1, it appears uncomplemented.

Truth tables

Deriving Boolean equations from truth tables:

a	b	s1	s0
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

s0 = a ^ b

s1 = a b

Example: a full adder

A	B	Cin	Cout	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Sum =

Cout =

Example: a full adder

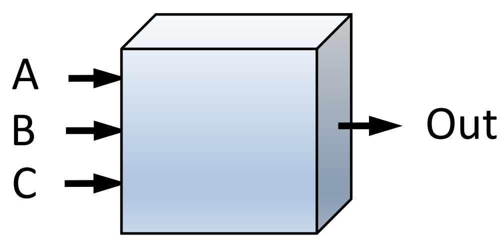
A	B	Cin	Cout	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Sum = A’ B’ Cin + A’ B Cin’ + A B’ Cin’ + A B Cin

Cout = A’ B Cin + A B’ Cin + A B Cin’ + A B Cin

Example: A majority function

1. Output should = 1 if the majority of the inputs = 1.



Truth Table

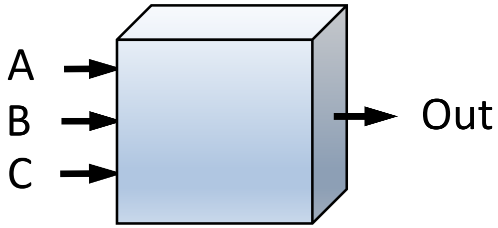
A	B	C	Out

2. Consider the unknown circuit a “black box”.

3. Create a truth table.

Example: A majority function

4. Enumerate all the possible input combinations.

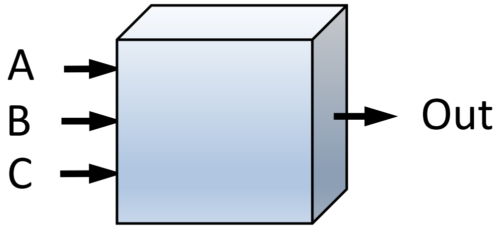


Truth Table

A	B	C	Out
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Example: A majority function

5. Fill in the outputs.

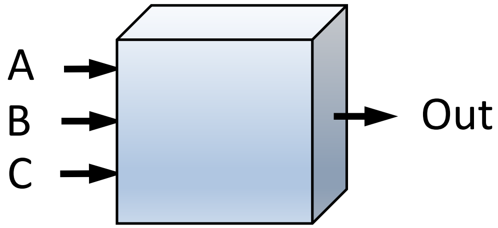


Truth Table

A	B	C	Out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Example: A majority function

5. Write the equation summing up all the 1's.



Truth Table

A	B	C	Out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Out = A' B C + A B' C + A B C' + A B C

A deeper dive into Boolean algebra

Axioms of Boolean Algebra

1a. $0 \bullet 0 = 0$

1b. $1 + 1 = 1$

2a. $1 \bullet 1 = 1$

2b. $0 + 0 = 0$

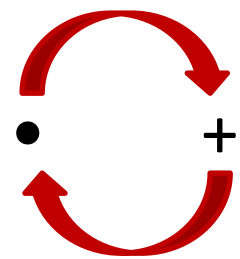
3a. $0 \bullet 1 = 1 \bullet 0 = 0$

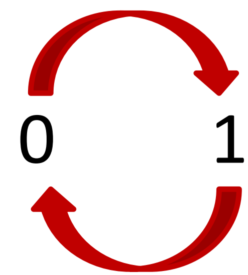
3b. $1 + 0 = 0 + 1 = 1$

4a. If $x = 0$, then $x' = 1$

4b. If $x = 1$, then $x' = 0$

Notice the *duality*:





Single-variable theorems

5a.	$x \bullet 0 = 0$	
5b.	$x + 1 = 1$	
6a.	$x \bullet 1 = x$	
6b.	$x + 0 = x$	
7a.	$x \bullet x = x$	Replication
7b.	$x + x = x$	
8a.	$x \bullet x' = 0$	
8b.	$x + x' = 1$	
9.	$(x')' = x$	

Easily proved by *perfect induction,* trying all the possibilities.

2 and 3-variable properties

10a.	$x \bullet y = y \bullet x$	Commutative
10b.	$x + y = y + x$	
11a.	$x \bullet (y \bullet z) = (x \bullet y) \bullet z$	Associative
11b.	$x + (y + z) = (x + y) + z$	
12a.	$x \bullet (y + z) = x \bullet y + x \bullet z$	Distributive
12b.	$x + y \bullet z = (x + y) \bullet (x + z)$	
13a.	$x + x \bullet y = x$	Absorption
13b.	$x \bullet (x + y) = x$	
14a.	$x \bullet y + x \bullet y' = x$	Combining
14b.	$(x + y) \bullet (x + y') = x$	

Easily proved by *perfect induction,* trying all the possibilities.

2 and 3-variable properties

15a. $(x \bullet y)' = x' + y'$ DeMorgan's theorem

15b. $(x + y)' = x' \bullet y'$

16a. $x + x' \bullet y = x + y$

16b. $x \bullet (x' + y) = x \bullet y$

17a. $x \bullet y + y \bullet z + x' \bullet z = x \bullet y + x' \bullet z$ Consensus

17b. $(x + y) \bullet (y + z) \bullet (x' + y) = (x + y) \bullet (x' + z)$

Easily proved by *perfect induction,* trying all the possibilities.

Can prove Boolean theorems by

1. Perfect induction
2. Algebraically
3. Venn diagrams

DeMorgan’s theorem by perfect induction

$$(x \bullet y)' = x' + y'$$

		LHS		RHS		
x	y	x • y	(x • y)'	x'	y'	x' + 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

Identical

Proof of DeMorgan’s theorem by perfect induction, enumerating all the possibilities in a truth table.

Algebraic proof of the Combining theorem

Combining theorem:

14a. $x \bullet y + x \bullet y' = x$

14b. $(x + y) \bullet (x + y') = x$

$$
\begin{aligned}
x \bullet y + x \bullet y' &= x\,(y + y') \\
&= x
\end{aligned}
$$

$$
\begin{aligned}
(x + y)\,(x + y') &= x\,x + x\,y' + x\,y + y\,y' \\
&= x + x\,(y' + y) + 0 \\
&= x + x = x
\end{aligned}
$$

Algebraic proof of the Consensus theorem

Prove we can ignore this term.

Prove: $x\,y + x'\,z + \boxed{y\,z} = x\,y + x'\,z$

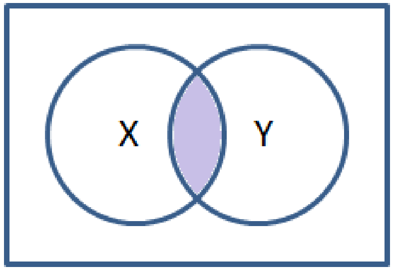
$(x + x') = 1$

$y\,z = (x + x')\,y\,z = x\,y\,z + x'\,y\,z$

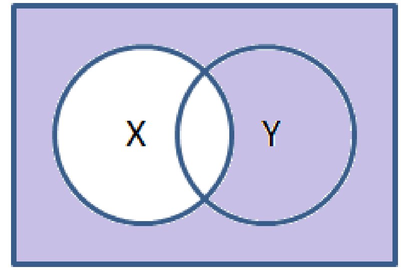
Substituting back into the original LHS:

$$
\begin{aligned}
x\,y + x'\,z + y\,z &= x\,y + x'\,z + (x\,y\,z + x'\,y\,z) \\
&= x\,y + x\,y\,z + x'\,z + x'\,y\,z \\
&= x\,y\,(1 + z) + x'\,z\,(1 + y) \\
&= x\,y + x'\,z
\end{aligned}
$$

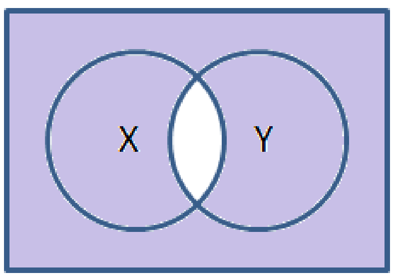
Proof of DeMorgan’s Theorem



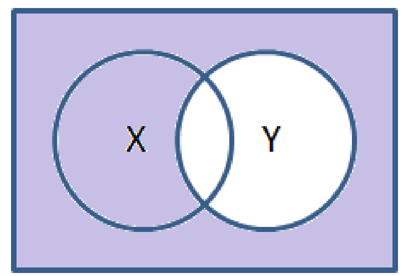
X • Y



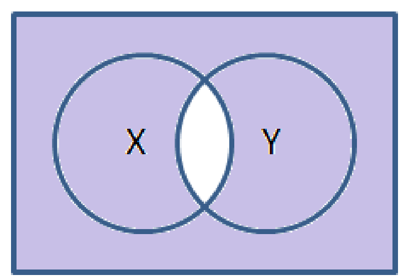
X’



(X • Y)’

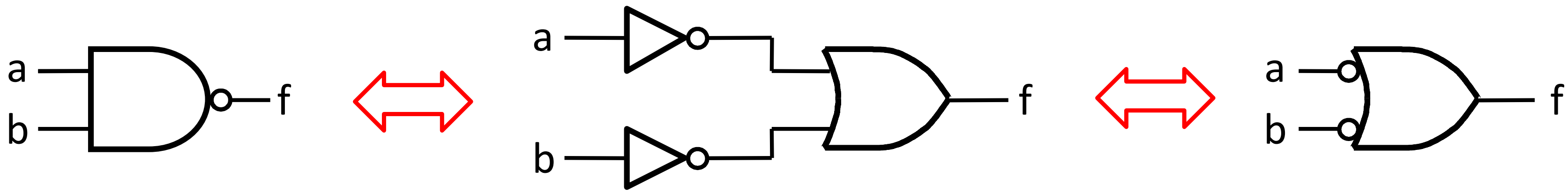


Y’

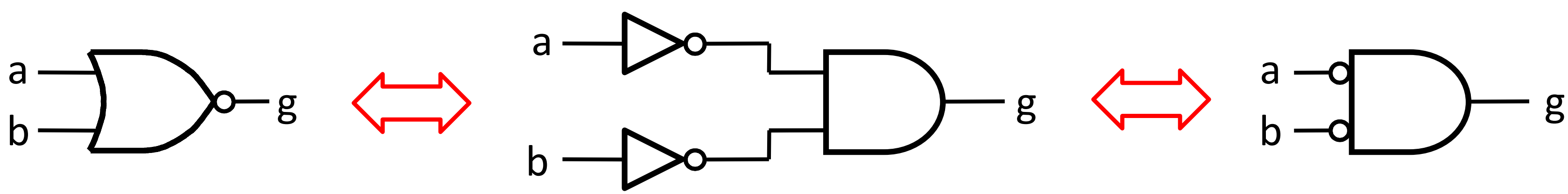


X’ + Y’

Bubble pushing



$$f = (a\, b)' = a' + b'$$



$$g = (a + b)' = a'\, b'$$

DeMorgan's theorem in terms of logic gates.

Operator precedence

Highest	NOT	x'
	AND	$\bullet$
Lowest	OR	$+$

Example: $x + y \bullet z' = x + (y \bullet (z'))$

Parentheses can be used to specify a different order of evaluation, for example:

$$((x + y) \bullet z)'$$

We tend to omit the • when the meaning is clear.

Minimization

Often relies on these Boolean theorems:

1. $a + a\,b = a\,(1 + b) = a$
2. $a\,b + a\,b' = a\,(b + b') = a$
3. $(a + b)\,(a + b') = a$
4. $a + a = a$

Synthesis is the process of beginning with a description of the *desired* functional behavior and then generating a circuit that *realizes* that behavior.

Exercise: Synthesize this function

x1	x2	f(x1, x2)
0	0	1
0	1	1
1	0	0
1	1	1

Exercise: Synthesize this function

x1	x2	f(x1, x2)
0	0	1
0	1	1
1	0	0
1	1	1

f(x1, x2) = x1' x2' + x1' x2 + x1 x2

We like to simplify both

x1' x2' + x1' x2 = x1' (x2' + x2) = x1'

x1' x2 + x1 x2 = (x1' + x1) x2 = x2

To do that, we add a copy of the middle term. We can do that because x + x = x.

Exercise: Synthesize this function

x1	x2	f(x1, x2)
0	0	1
0	1	1
1	0	0
1	1	1

$$f(x1, x2) = \boxed{x1'\, x2' + x1'\, x2} + x1\, x2$$

Since x + x = x, we can replicate the middle term:

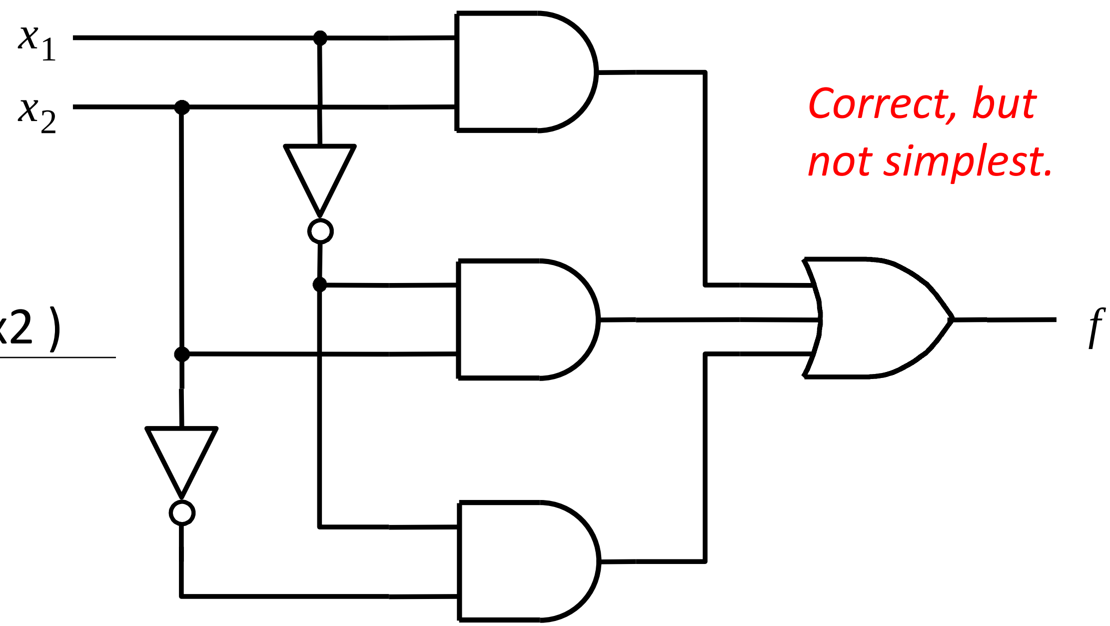
$$f(x1, x2) = x1'\, x2' + x1'\, x2 + x1'\, x2 + x1\, x2$$

Using the distributive property:

$$f(x1, x2) = x1'\ (x2' + x2) + (x1' + x1)\, x2$$

$$= x1' + x2$$

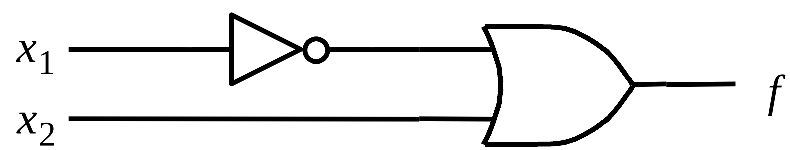
x1	x2	f(x1, x2)
0	0	1
0	1	1
1	0	0
1	1	1



Correct, but not simplest.

(a) Canonical sum-of-products

f(x1, x2) = x1’ + x2



(b) Minimal-cost realization

Figure 2.20. Two implementations of the function in Figure 2.19.

Two ways to synthesize a function

Sum of products: *Include* all rows where f = 1 using minterms.

Product of sums: *Exclude* all rows where f = 0 using Maxterms.

minterms and Maxterms

A *minterm* is 1 for only one row.

A *Maxterm* is 0 for only one row.

Minterms and maxterms for all possible combinations of 3 variables

Row	x1	x2	x3	Minterm	Maxterm
0	0	0	0	m0 = x1' x2' x3'	M0 = x1 + x2 + x3
1	0	0	1	m1 = x1' x2' x3	M1 = x1 + x2 + x3'
2	0	1	0	m2 = x1' x2 x3'	M2 = x1 + x2' + x3
3	0	1	1	m3 = x1' x2 x3	M3 = x1 + x2' + x3'
4	1	0	0	m4 = x1 x2' x3'	M4 = x1' + x2 + x3
5	1	0	1	m5 = x1 x2' x3	M5 = x1' + x2 + x3'
6	1	1	0	m6 = x1 x2 x3'	M6 = x1' + x2' + x3
7	1	1	1	m7 = x1 x2 x3	M7 = x1' + x2' + x3'

Minterms are small m

Maxterms are big M

Minterms (small m)

Row	x1	x2	x3	Minterm
0	0	0	0	m0 = x1' x2' x3'
1	0	0	1	m1 = x1' x2' x3
2	0	1	0	m2 = x1' x2 x3'
3	0	1	1	m3 = x1' x2 x3
4	1	0	0	m4 = x1 x2' x3'
5	1	0	1	m5 = x1 x2' x3
6	1	1	0	m6 = x1 x2 x3'
7	1	1	1	m7 = x1 x2 x3

A *minterm* is 1 for only one row.

It's an AND expression in which each of the input variables appears once.

Each variable can be in complemented, or uncomplemented, e.g., x' or x.

To match a row in a truth table, use the *uncomplemented* form to match a 1 and the *complemented* form to match a 0.

For example, x1 x2' x3 matches the row where
(x1, x2, x3) = (1, 0, 1)

Maxterms (big M)

Row	x1	x2	x3	Maxterm
0	0	0	0	M0 = x1 + x2 + x3
1	0	0	1	M1 = x1 + x2 + x3'
2	0	1	0	M2 = x1 + x2' + x3
3	0	1	1	M3 = x1 + x2' + x3'
4	1	0	0	M4 = x1' + x2 + x3
5	1	0	1	M5 = x1' + x2 + x3'
6	1	1	0	M6 = x1' + x2' + x3
7	1	1	1	M7 = x1' + x2' + x3'

A *maxterm* is a 0 for only one matching row.

It's an OR expression in which each of the input variables appears once.

Each variable can be in complemented, or uncomplemented, e.g., x' or x.

To match a row in a truth table, use the *complemented* form to match a 1 and the *uncomplemented* form to match a 0.

For example, x1' + x2 + x3' matches the row where (x1, x2, x3) = (1, 0, 1).

Sum of products

Include all rows where f = 1 using minterms.

$$f = \sum \left(m_i \bullet f_i \right)$$

Where f_i is the desired result for row i.
If f_i is 0, we can eliminate that term.

Exercise: A 3-variable function we'd like to synthesize

Row	x1	x2	x3	f(x1, x2, x3)
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0

$$f = \sum \left(m_i \bullet f_i \right)$$

Using minterms for the rows where we want ones:

Exercise: A 3-variable function we'd like to synthesize

Row	x1	x2	x3	f(x1, x2, x3)
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0

$$f = \sum \left(m_i \bullet f_i \right)$$

Using minterms for the rows where we want ones:

f = Σm(1, 4, 5, 6) = m1 + m4 + m5 + m6

= (m1 + m5) + (m4 + m6)

= (x1’ x2’ x3 + x1 x2’ x3) + (x1 x2’ x3’ + x1 x2 x3’)

= (x1’ + x1) x2’ x3 + x1 (x2’ + x2) x3’

= x2’ x3 + x1 x3’

Product of sums

Exclude all rows where f = 0 using Maxterms.

$$f = \prod \left(M_i + f_i \right)$$

Where f_i is the desired result for row i.
If f_i is 1, we can eliminate that term.

Exercise: A 3-variable function we'd like to synthesize

Row	x1	x2	x3	f(x1, x2, x3)
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0

$$f = \prod \left(M_i + f_i \right)$$

Using Maxterms for the rows where we want zeros:

f =

Exercise: A 3-variable function we'd like to synthesize

Row	x1	x2	x3	f(x1, x2, x3)
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0

$$f = \prod \left(M_i + f_i \right)$$

Using Maxterms for the rows where we want zeros:

f = ΠM(0, 2, 3, 7) = M0 • M2 • M3 • M7

Exercise: A 3-variable function we'd like to synthesize

Row	x1	x2	x3	f(x1, x2, x3)
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0

$$f = \prod \left(M_i + f_i \right)$$

Using Maxterms for the rows where we want zeros:

f = ΠM(0, 2, 3, 7) = M0 • M2 • M3 • M7

= (x1 + x2 + x3) (x1 + x2’ + x3) (x1 + x2’ + x3’) (x1’ + x2’ + x3’)

Exercise: A 3-variable function we'd like to synthesize

Row	x1	x2	x3	f(x1, x2, x3)
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0

$$f = \prod \left(M_i + f_i \right)$$

Using Maxterms for the rows where we want zeros:

f = ΠM(0, 2, 3, 7) = M0 • M2 • M3 • M7

= (x1 + x2 + x3) (x1 + x2’ + x3) (x1 + x2’ + x3’) (x1’ + x2’ + x3’)

= ((x1 + x3) + x2) ((x1 + x3) + x2’) (x1 + (x2’ + x3’)) (x1’ + (x2’ + x3’))

Using Maxterms for the rows where we want zeros:

$f = M0 \bullet M2 \bullet M3 \bullet M7$

$= (x1 + x2 + x3)(x1 + x2' + x3)(x1 + x2' + x3')(x1' + x2' + x3')$

$= ((x1 + x3) + x2) ((x1 + x3) + x2') (x1 + (x2' + x3'))(x1' + (x2' + x3'))$

Combining theorem:

14a. $x \bullet y + x \bullet y' = x$

14b. $(x + y) \bullet (x + y') = x$

$f = ((x1 + x3) + x2) ((x1 + x3) + x2') (x1 + (x2' + x3')) (x1' + (x2' + x3'))$

$= (x1 + x3) (x2' + x3')$

Exercise: A 3-variable function we'd like to synthesize

Row	x1	x2	x3	f(x1, x2, x3)
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0

$$f = \prod \left(M_i + f_i \right)$$

Using Maxterms for the rows where we want zeros:

f = ΠM(0, 2, 3, 7) = M0 • M2 • M3 • M7

= (x1 + x2 + x3) (x1 + x2’ + x3) (x1 + x2’ + x3’) (x1’ + x2’ + x3’)

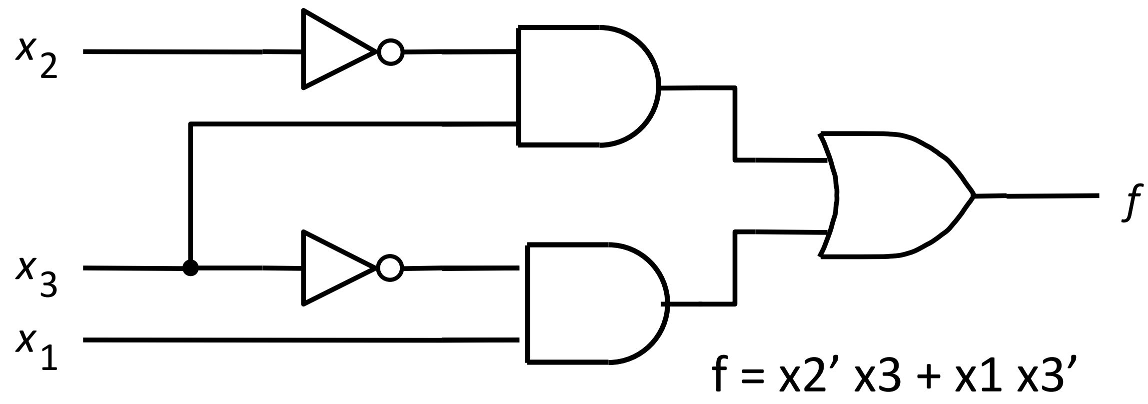
= ((x1 + x3) + x2) ((x1 + x3) + x2’) (x1 + (x2’ + x3’)) (x1’ + (x2’ + x3’))

= (x1 + x3)(x2 + x2’)(x2’ + x3’)(x1 + x1’)

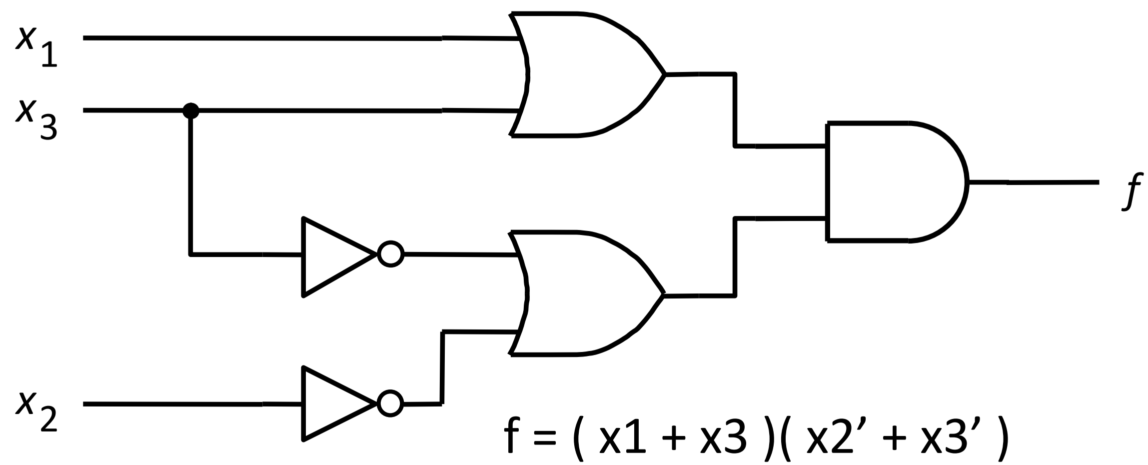
= (x1 + x3)(x2’ + x3’)

Exercise: A 3-variable function we'd like to synthesize

Row	x1	x2	x3	f(x1, x2, x3)
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0



(a) A minimal sum-of-products realization



(b) A minimal product-of-sums realization

Figure 2.24. Two realizations of the function.

Exercise: A 3-variable function we'd like to synthesize

Row	x1	x2	x3	f(x1, x2, x3)
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0

Are POS and SOP solutions always equivalent cost?

1. Does it matter how many rows are 1s and how many are 0s? Why or why not?
2. Does it matter which rows are 1s or 0s in relation to each other?

Karnaugh maps

Want simplest forms but the algebra is difficult.

Karnaugh maps

Invented by Maurice Karnaugh in 1954 as a graphical method for simplifying Boolean equations.

http://www.ithistory.org/sites/default/files/honor-roll/Maurice%20Karnaugh.jpg

Karnaugh maps

Truth table

row	a b	f
0	0 0	
1	0 1	
2	1 0	
3	1 1	

		b	
		0	1
a	0	*0*	*1*
	1	*2*	*3*

		a	
		0	1
b	0	*0*	*2*
	1	*1*	*3*

Map rows in a truth table to cells in a matrix.

May choose either assignment of columns and rows.

Example

Truth table

row	a b	f
0	0 0	*1*
1	0 1	*1*
2	1 0	*0*
3	1 1	*1*

⇨

		b	
		0	1
a	0	*1*	*1*
	1	*0*	*1*

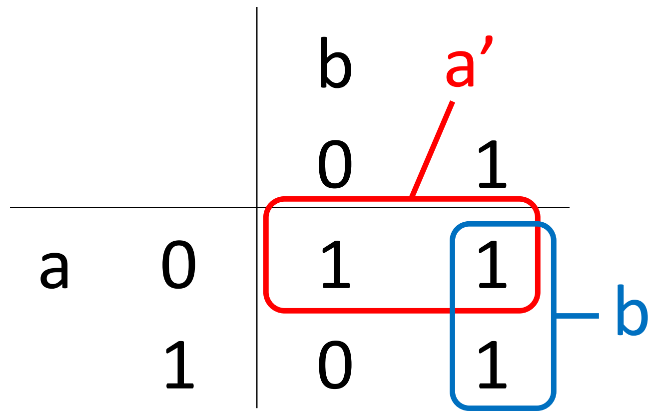
		a	
		0	1
b	0	*1*	*0*
	1	*1*	*1*

Fill in the desired output values.

Truth table

row	a b	f
0	0 0	1
1	0 1	1
2	1 0	0
3	1 1	1

Karnaugh map



Use the Combining property to group neighboring cells where the output should be the same.

14a. $x \bullet y + x \bullet y' = x$

Truth table

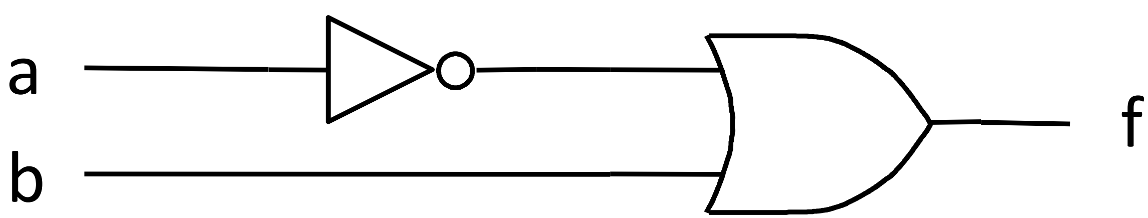
row	a b	f
0	0 0	1
1	0 1	1
2	1 0	0
3	1 1	1

Karnaugh map

		b	a’	
		0	1	
a	0	1	1	b
	1	0	1	

f = a’ + b

Form the minimal SOP solution.



A function of 3 variables

Truth table

row	a b c	f
0	0 0 0	
1	0 0 1	
2	0 1 0	
3	0 1 1	
4	1 0 0	
5	1 0 1	
6	1 1 0	
7	1 1 1	

Karnaugh map

		bc			
		00	01	11	10
a	0	*0*	*1*	*3*	*2*
	1	*4*	*5*	*7*	*6*

Map the rows to a 2 x 4 matrix.
Columns are arranged so each differs by only 1 bit from the next.

Example

Truth table

row	a b c	f
0	0 0 0	*0*
1	0 0 1	*0*
2	0 1 0	*0*
3	0 1 1	*1*
4	1 0 0	*0*
5	1 0 1	*1*
6	1 1 0	*1*
7	1 1 1	*1*

Karnaugh map

		bc			
		00	01	11	10
a	0	*0*	*0*	*1*	*0*
	1	*0*	*1*	*1*	*1*

Example

Truth table

row	a b c	f
0	0 0 0	0
1	0 0 1	0
2	0 1 0	0
3	0 1 1	1
4	1 0 0	0
5	1 0 1	1
6	1 1 0	1
7	1 1 1	1

Karnaugh map

		bc			
		00	01	11	10
a	0	0	0	1	0
	1	0	1	1	1

$$f = a\,b + a\,c + b\,c$$

A function of four variables

Truth table

row	a b c d	f
0	0 0 0 0	
1	0 0 0 1	
2	0 0 1 0	
3	0 0 1 1	
4	0 1 0 0	
5	0 1 0 1	
6	0 1 1 0	
7	0 1 1 1	
8	1 0 0 0	
9	1 0 0 1	
10	1 0 1 0	
11	1 0 1 1	
12	1 1 0 0	
13	1 1 0 1	
14	1 1 1 0	
15	1 1 1 1	

Karnaugh map

		cd			
		00	01	11	10
ab	00	*0*	*1*	*3*	*2*
	01	*4*	*5*	*7*	*6*
	11	*12*	*13*	*15*	*14*
	10	*8*	*9*	*11*	*10*

		ab			
		00	01	11	10
cd	00	*0*	*4*	*12*	*8*
	01	*1*	*5*	*13*	*9*
	11	*3*	*7*	*15*	*11*
	10	*2*	*6*	*14*	*10*

Map the rows to a 4 x 4 matrix either way. (I use the top one.)

Example

Truth table

row	a b c d	f
0	0 0 0 0	*0*
1	0 0 0 1	*0*
2	0 0 1 0	*0*
3	0 0 1 1	*1*
4	0 1 0 0	*1*
5	0 1 0 1	*0*
6	0 1 1 0	*1*
7	0 1 1 1	*0*
8	1 0 0 0	*0*
9	1 0 0 1	*0*
10	1 0 1 0	*0*
11	1 0 1 1	*1*
12	1 1 0 0	*1*
13	1 1 0 1	*0*
14	1 1 1 0	*1*
15	1 1 1 1	*0*

Karnaugh map

		cd			
		00	01	11	10
ab	00	*0*	*0*	*1*	*0*
	01	*1*	*0*	*0*	*1*
	11	*1*	*0*	*0*	*1*
	10	*0*	*0*	*1*	*0*

Copy the outputs to the Karnaugh map.

Example

Truth table

row	a b c d	f
0	0 0 0 0	0
1	0 0 0 1	0
2	0 0 1 0	0
3	0 0 1 1	1
4	0 1 0 0	1
5	0 1 0 1	0
6	0 1 1 0	1
7	0 1 1 1	0
8	1 0 0 0	0
9	1 0 0 1	0
10	1 0 1 0	0
11	1 0 1 1	1
12	1 1 0 0	1
13	1 1 0 1	0
14	1 1 1 0	1
15	1 1 1 1	0

Karnaugh map

		cd			
		00	01	11	10
ab	00	0	0	1	0
	01	1	0	0	1
	11	1	0	0	1
	10	0	0	1	0

f = b d' + b' c d

Notice that the Karnaugh map "wraps" vertically and horizontally.

Example

Truth table

row	a b c d	f
0	0 0 0 0	0
1	0 0 0 1	0
2	0 0 1 0	0
3	0 0 1 1	1
4	0 1 0 0	1
5	0 1 0 1	0
6	0 1 1 0	1
7	0 1 1 1	0
8	1 0 0 0	0
9	1 0 0 1	0
10	1 0 1 0	0
11	1 0 1 1	1
12	1 1 0 0	1
13	1 1 0 1	0
14	1 1 1 0	1
15	1 1 1 1	0

Karnaugh map

		cd 00	01	11	10
ab	00			1	
	01	1			1
	11	1			1
	10			1	

f = b d’ + b’ c d

If we’re collecting 1’s for an SOP solution, we can leave out the 0’s.

SOP terminology

Literal	A variable or its complement, e.g., *x* or *x'*.
Product term	A product, e.g., *x y' z*, of some number of literals.
Implicant	A product term for which the output is 1. That product term *implies* the output is true.
Prime implicant	An implicant that cannot be combined with another with fewer literals.

Truth table

row	a b	f
0	0 0	1
1	0 1	1
2	1 0	0
3	1 1	1

Karnaugh map

		b	a’	
		0	1	
a	0	1	1	b
	1	0	1	

f = a’ + b

Each row or cell where f = 1 is an *implicant*.

The ***prime** implicants* are a’ and b.

Cover A collection of implicants that account for all cases for which the output = 1.

Essential prime implicant

A *prime implicant* that *must* be included in any *cover*.

Truth table

row	a b	f
0	0 0	1
1	0 1	1
2	1 0	0
3	1 1	1

Karnaugh map

		b	
		0	1
a	0	1	1
	1	0	1

$f = a' + b$

a' and b form a *cover* for f.

Both are *essential prime implicants*.

Truth table

row	a b	f
0	0 0	1
1	0 1	1
2	1 0	0
3	1 1	1

Karnaugh map

		b	a’	
		0	1	
a	0	1	1	
	1	0	1	b

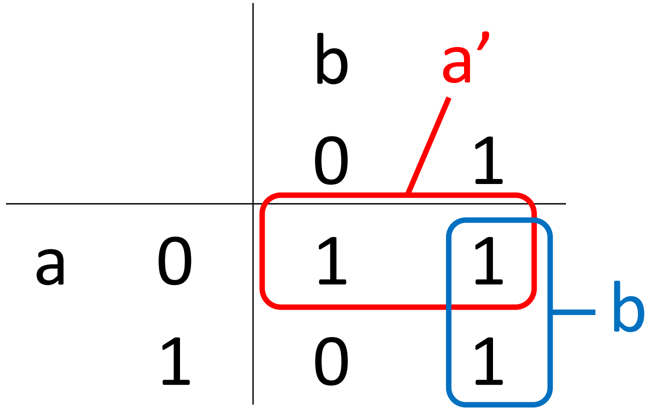
f = a’ + b

For a function of *n* variables, there will be 2^n rows in the truth table and 2^n cells in the Karnaugh map.

Truth table

row	a b	f
0	0 0	1
1	0 1	1
2	1 0	0
3	1 1	1

Karnaugh map



f = a’ + b

The number of cells in an implicant must be a power of 2.

Truth table

row	a b	f
0	0 0	1
1	0 1	1
2	1 0	0
3	1 1	1

Karnaugh map

		b	a’	
		0	1	
a	0	1	1	b
	1	0	1	

f = a’ + b

For a function of *n* variables, if an implicant has *k* literals, it must cover 2^{n-k} cells.

Examples

		b	
		0	1
a	0	0	1
	1	0	1

f =

		b	
		0	1
a	0	1	1
	1	0	1

g =

		bc			
		00	01	11	10
a	0	0	1	1	1
	1	0	0	1	0

h =

		bc			
		00	01	11	10
a	0	0	0	1	1
	1	0	0	1	1

j =

Examples

		b	
		0	1
a	0	0	1
	1	0	1

f = b

		b	
		0	1
a	0	1	1
	1	0	1

g =

		bc			
		00	01	11	10
a	0	0	1	1	1
	1	0	0	1	0

h =

		bc			
		00	01	11	10
a	0	0	0	1	1
	1	0	0	1	1

j =

Examples

		b	
		0	1
a	0	0	1
	1	0	1

f = b

		b	
		0	1
a	0	1	1
	1	0	1

g = a’ + b

		bc			
		00	01	11	10
a	0	0	1	1	1
	1	0	0	1	0

h =

		bc			
		00	01	11	10
a	0	0	0	1	1
	1	0	0	1	1

j =

Examples

		b	
		0	1
a	0	0	1
	1	0	1

f = b

		b	
		0	1
a	0	1	1
	1	0	1

g = a’ + b

		bc			
		00	01	11	10
a	0	0	1	1	1
	1	0	0	1	0

h = a’ c + b c + a’ b

		bc			
		00	01	11	10
a	0	0	0	1	1
	1	0	0	1	1

j =

Examples

		b	
		0	1
a	0	0	1
	1	0	1

$f = b$

		b	
		0	1
a	0	1	1
	1	0	1

$g = a' + b$

		bc			
		00	01	11	10
a	0	0	1	1	1
	1	0	0	1	0

$h = a'\,c + b\,c + a'\,b$

		bc			
		00	01	11	10
a	0	0	0	1	1
	1	0	0	1	1

$j = b$

We can also use Karnaugh maps to help us with Boolean algebra.

Simplify

$k = a' \, b \, c' + a \, b \, c' + a \, b \, c$

Simplify

$k = a' \, b \, c' + a \, b \, c' + a \, b \, c$
$\quad = \Sigma m(\, 2, 6, 7 \,)$

		bc			
		00	01	11	10
a	0				1
	1			1	1

1. Create the Karnaugh map.

Simplify

k = a’ b c’ + a b c’ + a b c

   = Σm(2, 6, 7)

		bc			
		00	01	11	10
a	0				1
	1			1	1

2. Find the prime implicants.

Simplify

$$k = a'\,b\,c' + a\,b\,c' + a\,b\,c$$
$$= a'\,b\,c' + a\,b\,c' + a\,b\,c' + a\,b\,c$$
$$= b\,c'\,(\,a' + a\,) + a\,b\,(\,c' + c\,)$$
$$= b\,c' + a\,b$$

We'll need to duplicate the middle term.

		bc			
		00	01	11	10
a	0				1
	1			1	1

3. Use this as a guide to solving the algebra.

Simplify

$k = a' \, b' \, c' + a' \, b \, c' + a \, b \, c' + a \, b \, c$

Simplify

$k = a' \, b' \, c' + a' \, b \, c' + a \, b \, c' + a \, b \, c$
$= \Sigma m(0, 2, 6, 7)$

		bc			
		00	01	11	10
a	0	1			1
	1			1	1

1. Create the Karnaugh map.

Simplify

$k = a'\,b'\,c' + a'\,b\,c' + a\,b\,c' + a\,b\,c$

$\quad = \Sigma m(\,0,\,2,\,6,\,7\,)$

		bc			
		00	01	11	10
a	0	1			1
	1			1	1

2. Find the prime implicants.

Simplify

$k = a'\,b'\,c' + a'\,b\,c' + a\,b\,c' + a\,b\,c$

$\quad = a'\,(\,b' + b\,)\,c' + a\,b\,(\,c + c'\,)$

$\quad = a'\,c' + a\,b$

		bc 00	01	11	10
a	0	1			1
	1			1	1

$b\,c'$ is non-essential.
So we should combine the other terms instead.

3. Use this as a guide to solving the algebra.

Simplify

$k = a'\,b'\,c' + a'\,b\,c' + a\,b\,c' + a\,b\,c$
$\quad = a'\,(\,b' + b\,)\,c' + a\,b\,(\,c + c'\,)$
$\quad = a'\,c' + a\,b$

		bc			
		00	01	11	10
a	0	1			1
	1			1	1

$b\,c'$ is non-essential.
So we should combine the other terms instead.

3. Use this as a guide to solving the algebra.

Simplify

$m = a' \; b' \; c + a' \; b \; c + a \; b' \; c' + a \; b' \; c$

Simplify

$m = a'\,b'\,c + a'\,b\,c + a\,b'\,c' + a\,b'\,c$
$\quad = \Sigma m(\,1, 3, 4, 5\,)$

		bc 00	01	11	10
a	0		1	1	
	1	1	1		

1. Create the Karnaugh map.

Simplify

$m = a'\, b'\, c + a'\, b\, c + a\, b'\, c' + a\, b'\, c$

$\quad = \Sigma m(\, 1, 3, 4, 5\,)$

		bc			
		00	01	11	10
a	0		1	1	
	1	1	1		

2. Find the prime implicants.

Simplify

$$m = a'\,b'\,c + a'\,b\,c + a\,b'\,c' + a\,b'\,c$$
$$= a'\,(\,b' + b\,)\,c + a\,b'\,(\,c + c'\,)$$
$$= a'\,c + a\,b'$$

		bc			
		00	01	11	10
a	0		1	1	
	1	1	1		

$b'\,c$ is non-essential.
So we should combine the other terms instead.

3. Use this as a guide to solving the algebra.

Simplify

$f(a, b, c) = \Sigma m(0, 2, 4, 5, 6)$

Write directly from the Karnaugh map

$f(a, b, c) = \Sigma m(0, 2, 4, 5, 6)$

$= a\,b' + c'$

a \ bc	00	01	11	10
0	1			1
1	1	1		1

Write directly from the Karnaugh map

$f(a, b, c) = \Sigma m(1, 4, 5, 6)$

Write directly from the Karnaugh map

$f(a, b, c) = \Sigma m(1, 4, 5, 6)$

$= a\,c' + b'\,c$

$a\,b'$ is not essential.

		bc			
		00	01	11	10
a	0		1		
	1	1	1		1

Examples

		cd 00	01	11	10
ab	00				
	01			1	1
	11	1			1
	10	1			1

f =

		cd 00	01	11	10
ab	00				
	01			1	1
	11	1	1	1	1
	10	1	1	1	1

g =

		cd 00	01	11	10
ab	00	1			1
	01				
	11	1	1	1	
	10	1	1		1

h =

		cd 00	01	11	10
ab	00	1	1	1	
	01	1	1	1	
	11			1	1
	10			1	1

j =

Examples

		cd 00	01	11	10
ab	00				
	01			1	1
	11	1			1
	10	1			1

f = a d’ + a’ b c

		cd 00	01	11	10
ab	00				
	01			1	1
	11	1	1	1	1
	10	1	1	1	1

g =

		cd 00	01	11	10
ab	00	1			1
	01				
	11	1	1	1	
	10	1	1		1

h =

		cd 00	01	11	10
ab	00	1	1	1	
	01	1	1	1	
	11			1	1
	10			1	1

j =

Examples

ab \ cd	00	01	11	10
00				
01			1	1
11	1			1
10	1			1

$f = a\,d' + a'\,b\,c$

ab \ cd	00	01	11	10
00				
01			1	1
11	1	1	1	1
10	1	1	1	1

$g = a + b\,c$

ab \ cd	00	01	11	10
00	1			1
01				
11	1	1	1	
10	1	1		1

$h =$

ab \ cd	00	01	11	10
00	1	1	1	
01	1	1	1	
11			1	1
10			1	1

$j =$

Examples

		cd			
		00	01	11	10
ab	00				
	01			1	1
	11	1			1
	10	1			1

$f = a\,d' + a'\,b\,c$

		cd			
		00	01	11	10
ab	00				
	01			1	1
	11	1	1	1	1
	10	1	1	1	1

$g = a + b\,c$

		cd			
		00	01	11	10
ab	00	1			1
	01				
	11	1	1	1	
	10	1	1		1

$h = a\,c' + b'\,d' + a\,b\,d$

		cd			
		00	01	11	10
ab	00	1	1	1	
	01	1	1	1	
	11			1	1
	10			1	1

$j =$

Examples

		cd			
		00	01	11	10
ab	00				
	01			1	1
	11	1			1
	10	1			1

f = a d’ + a’ b c

		cd			
		00	01	11	10
ab	00				
	01			1	1
	11	1	1	1	1
	10	1	1	1	1

g = a + b c

		cd			
		00	01	11	10
ab	00	1			1
	01				
	11	1	1	1	
	10	1	1		1

h = a c’ + b’ d’ + a b d

		cd			
		00	01	11	10
ab	00	1	1	1	
	01	1	1	1	
	11			1	1
	10			1	1

Either: j = a’ c’ + a c + c d

j = a’ c’ + a c + a’ d

A 5-variable Karnaugh map

Must be done in *layers*.

		cd			
		00	01	11	10
ab	00				
	01			1	1
	11	1	1		
	10	1	1		

e = 0

		cd			
		00	01	11	10
ab	00				1
	01			1	1
	11	1	1		
	10	1	1		

e = 1

$f = a\,c' + a'\,b\,c + a'\,c\,d'\,e$

Realistically, Karnaugh maps are impractical beyond 5 variables. (We turn the problem over to software.)

Often, there are many different networks than can realize a given function.

Some may be simpler than others.

$$f(a, b, c, d) = \Sigma m(5, 7, 8, 9, 10, 11, 12, 13, 14)$$

Not all the implicants are needed.

Which are the essential prime implicants?

What cells are still not covered?

Of the rest, which do you choose?

ab \ cd	00	01	11	10
00				
01		1	1	
11	1	1		1
10	1	1	1	1

f =

$$f(a, b, c, d) = \Sigma m(5, 7, 8, 9, 10, 11, 12, 13, 14)$$

Not all the implicants are needed.

Which are the essential prime implicants?

$a' b d + a b' + a d'$

What cells are still not covered?

Of the rest, which do you choose?

ab \ cd	00	01	11	10
00				
01		1	1	
11	1	1		1
10	1	1	1	1

f =

$f(a, b, c, d) = \Sigma m(5, 7, 8, 9, 10, 11, 12, 13, 14)$

Not all the implicants are needed.

Which are the essential prime implicants?

a’ b d + a b’ + a d’

What cells are still not covered?

Only 1101, covered by either a c’ or b c’ d

Of the rest, which do you choose?

ab \ cd	00	01	11	10
00				
01		1	1	
11	1	1		1
10	1	1	1	1

f =

$f(a, b, c, d) = \Sigma m(5, 7, 8, 9, 10, 11, 12, 13, 14)$

Not all the implicants are needed.

Which are the essential prime implicants?

$a' \, b \, d + a \, b' + a \, d'$

What cells are still not covered?

Only 1101, covered by either $a \, c'$ or $b \, c' \, d$

Of the rest, which do you choose?

Obviously, $a \, c'$.

ab \ cd	00	01	11	10
00				
01		1	1	
11	1	1		1
10	1	1	1	1

$f = a' \, b \, d + a \, b' + a \, d' + a \, c'$

$$f(a, b, c, d) = \Sigma m(0, 1, 2, 3, 7, 10, 14, 15)$$

Not all the implicants are needed.

Which are the essential prime implicants?

What cells are still not covered?

Of the rest, which do you choose?

ab \ cd	00	01	11	10
00	1	1	1	1
01			1	
11			1	1
10				1

f =

$$f(a, b, c, d) = \Sigma m(0, 1, 2, 3, 7, 10, 14, 15)$$

Not all the implicants are needed.

Which are the essential prime implicants?

Only a’ b’

What cells are still not covered?

Of the rest, which do you choose?

ab \ cd	00	01	11	10
00	1	1	1	1
01			1	
11			1	1
10				1

f =

$f(a, b, c, d) = \Sigma m(0, 1, 2, 3, 7, 10, 14, 15)$

Not all the implicants are needed.

Which are the essential prime implicants?

Only a’ b’

What cells are still not covered?

Everything else.

Of the rest, which do you choose?

ab \ cd	00	01	11	10
00	1	1	1	1
01			1	
11			1	1
10				1

f =

f(a, b, c, d) = Σm(0, 1, 2, 3, 7, 10, 14, 15)

Not all the implicants are needed.

Which are the essential prime implicants?

Only a’ b’

What cells are still not covered?

Everything else.

Of the rest, which do you choose?

Two choices.

ab \ cd	00	01	11	10
00	1	1	1	1
01			1	
11			1	1
10				1

ab \ cd	00	01	11	10
00	1	1	1	1
01			1	
11			1	1
10				1

$f(a, b, c, d) = \Sigma m(0, 1, 2, 3, 7, 10, 14, 15)$

Not all the implicants are needed.

Which are the essential prime implicants?

Only a’ b’

What cells are still not covered?

Everything else.

Of the rest, which do you choose?

Best to choose b c d + a c d’.

ab \ cd	00	01	11	10
00	1	1	1	1
01			1	
11			1	1
10				1

$f = a' b' + b c d + a c d'$

$$f(a, b, c, d) = \Sigma m(0, 1, 5, 7, 8, 10, 14, 15)$$

Not all the implicants are needed.

Which are the essential prime implicants?

What cells are still not covered?

Of the rest, which do you choose?

ab \ cd	00	01	11	10
00	1	1		
01		1	1	
11			1	1
10	1			1

f =

$$f(a, b, c, d) = \Sigma m(0, 1, 5, 7, 8, 10, 14, 15)$$

Not all the implicants are needed.

Which are the essential prime implicants?

There are none.

What cells are still not covered?

Everything.

Of the rest, which do you choose?

		cd			
		00	01	11	10
ab	00	1	1		
	01		1	1	
	11			1	1
	10	1			1

f =

$$f(a, b, c, d) = \Sigma m(0, 1, 5, 7, 8, 10, 14, 15)$$

Not all the implicants are needed.

Which are the essential prime implicants?

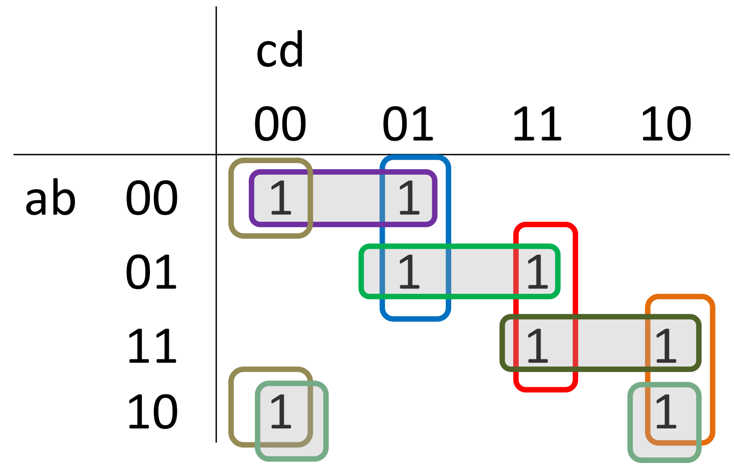
There are none.

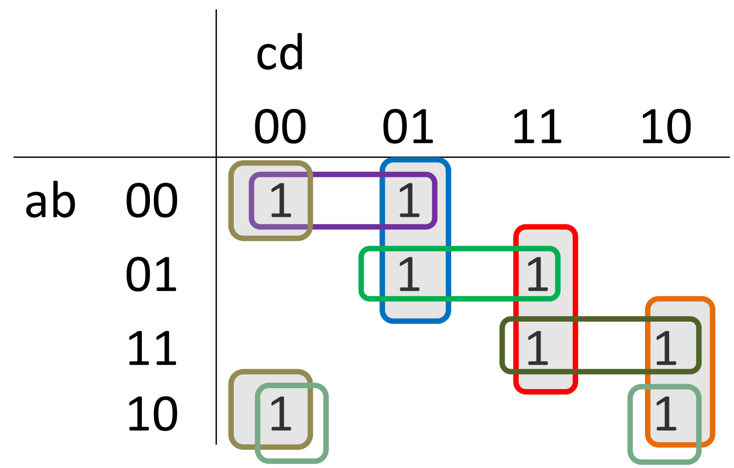
What cells are still not covered?

Everything.

Of the rest, which do you choose?

Two choices.





$f(a, b, c, d) = \Sigma m(0, 1, 5, 7, 8, 10, 14, 15)$

Not all the implicants are needed.

Which are the essential prime implicants?

There are none.

What cells are still not covered?

Everything.

Of the rest, which do you choose?

Two equivalent choices.

f = a’ b’ c’ + a’ b d + a b c + a b’ d’

f = b’ c’ d’ + a’ c’ d + b c d + a c d’

ab \ cd	00	01	11	10
00	1	1		
01		1	1	
11			1	1
10	1			1

ab \ cd	00	01	11	10
00	1	1		
01		1	1	
11			1	1
10	1			1

Karnaugh maps can also be used for POS minimization, collecting zeros instead of ones.

$f = \Pi M(2, 3, 6)$

$= (a + b')(b' + c)$

		bc 00	01	11	10
a	0			0	0
	1				0

$f = \Pi M(0, 1, 2, 3, 4, 6, 15)$

$= (a + d)(a + b)(a' + b' + c' + d')$

		cd 00	01	11	10
ab	00	0	0	0	0
	01	0			0
	11			0	
	10				

Examples

	cd 00	01	11	10
ab 00				
01			0	0
11	0			0
10	0			0

f =

	cd 00	01	11	10
ab 00				
01			0	0
11	0	0	0	0
10	0	0	0	0

g =

	cd 00	01	11	10
ab 00	0			0
01				
11	0	0	0	
10	0	0		0

h =

	cd 00	01	11	10
ab 00	0	0	0	
01	0	0	0	
11			0	0
10			0	0

j =

Examples

	cd			
ab	00	01	11	10
00				
01			$\boxed{0}$	$\boxed{0}$
11	$\boxed{0}$			$\boxed{0}$
10	$\boxed{0}$			$\boxed{0}$

$f = (a' + d)(a + b' + c')$

	cd			
ab	00	01	11	10
00				
01			0	0
11	0	0	0	0
10	0	0	0	0

$g =$

	cd			
ab	00	01	11	10
00	0			0
01				
11	0	0	0	
10	0	0		0

$h =$

	cd			
ab	00	01	11	10
00	0	0	0	
01	0	0	0	
11			0	0
10			0	0

$j =$

Examples

		cd			
		00	01	11	10
ab	00				
	01			0	0
	11	0			0
	10	0			0

$f = (a' + d)(a + b' + c')$

		cd			
		00	01	11	10
ab	00				
	01			0	0
	11	0	0	0	0
	10	0	0	0	0

$g = a'(b' + c')$

		cd			
		00	01	11	10
ab	00	0			0
	01				
	11	0	0	0	
	10	0	0		0

$h =$

		cd			
		00	01	11	10
ab	00	0	0	0	
	01	0	0	0	
	11			0	0
	10			0	0

$j =$

Examples

ab \ cd	00	01	11	10
00				
01			0	0
11	0			0
10	0			0

$f = (a' + d)(a + b' + c')$

ab \ cd	00	01	11	10
00				
01			0	0
11	0	0	0	0
10	0	0	0	0

$g = a'(b' + c')$

ab \ cd	00	01	11	10
00	0			0
01				
11	0	0	0	
10	0	0		0

$h = (a' + c)(b + d)(a' + b' + d')$

ab \ cd	00	01	11	10
00	0	0	0	
01	0	0	0	
11			0	0
10			0	0

$j =$

Examples

ab \ cd	00	01	11	10
00				
01			0	0
11	0			0
10	0			0

$f = (a' + d)(a + b' + c')$

ab \ cd	00	01	11	10
00				
01			0	0
11	0	0	0	0
10	0	0	0	0

$g = a'(b' + c')$

ab \ cd	00	01	11	10
00	0			0
01				
11	0	0	0	
10	0	0		0

$h = (a' + c)(b + d)(a' + b' + d')$

ab \ cd	00	01	11	10
00	0	0	0	
01	0	0	0	
11			0	0
10			0	0

$j = (a + c)(a' + c')(c' + d')$

$j = (a + c)(a' + c')(a + d')$

don’t cares

$$f = \Sigma m(1, 5, 8, 9, 10) + D(3, 7, 11, 15)$$

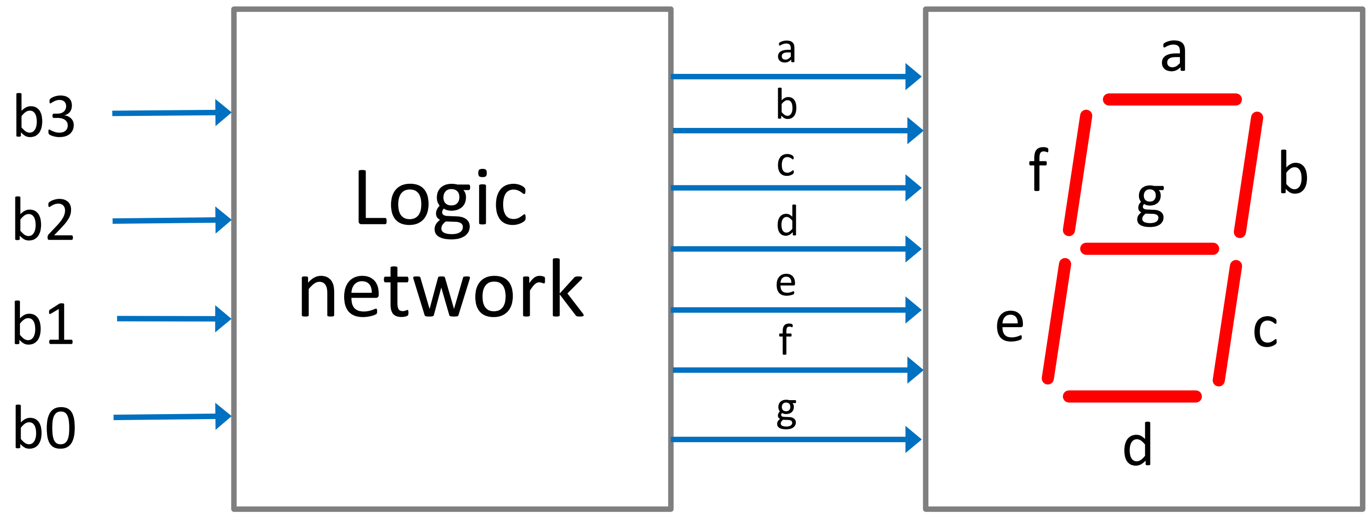
A ***don’t care*** is a cell where we really don’t care whether the output is a 1 or a 0.

We can decide whether to make it a 1 or a 0 depending on which makes for a simpler circuit.

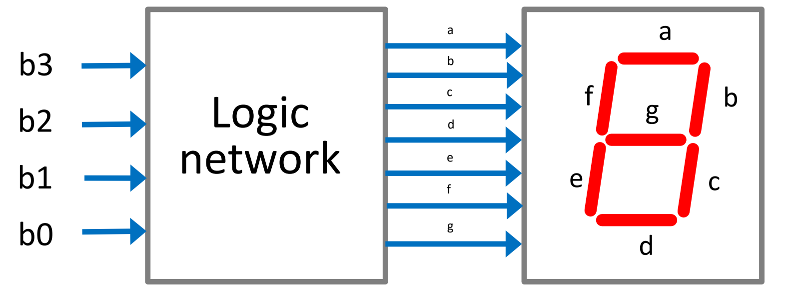
ab \ cd	00	01	11	10
00		1	d	
01		1	d	
11			d	
10	1	1	d	1

$$f = a' d + a b'$$

Example: A seven segment decoder to be used for displaying BCD digits 0 through 9 only.



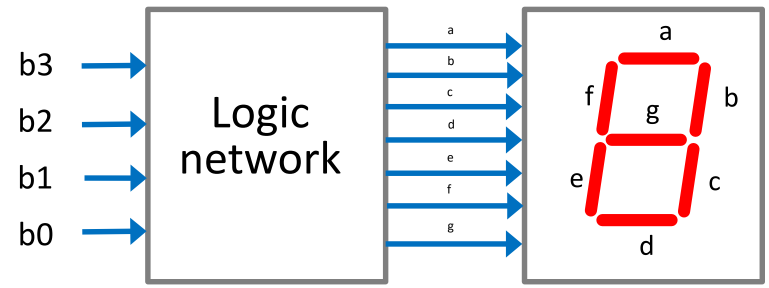
Start with a truth table and a blank Karnaugh map with don’t cares.



decimal	BCD	a b c d e f g
0	0 0 0 0	1 1 1 1 1 1 0
1	0 0 0 1	0 1 1 0 0 0 0
2	0 0 1 0	1 1 0 1 1 0 1
3	0 0 1 1	1 1 1 1 0 0 1
4	0 1 0 0	0 1 1 0 0 1 1
5	0 1 0 1	1 0 1 1 0 1 1
6	0 1 1 0	1 0 1 1 1 1 1
7	0 1 1 1	1 1 1 0 0 0 0
8	1 0 0 0	1 1 1 1 1 1 1
9	1 0 0 1	1 1 1 1 0 1 1

	b1 b0 00	01	11	10
b3 b2 00				
01				
11	d	d	d	d
10			d	d

Using don’t-cares for segment a.



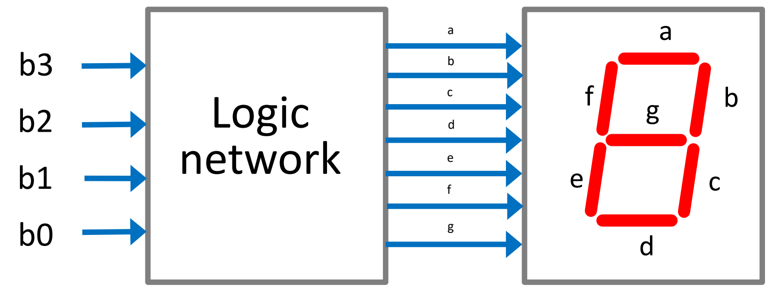
decimal	BCD	a b c d e f g
0	0 0 0 0	1 1 1 1 1 1 0
1	0 0 0 1	0 1 1 0 0 0 0
2	0 0 1 0	1 1 0 1 1 0 1
3	0 0 1 1	1 1 1 1 0 0 1
4	0 1 0 0	0 1 1 0 0 1 1
5	0 1 0 1	1 0 1 1 0 1 1
6	0 1 1 0	1 0 1 1 1 1 1
7	0 1 1 1	1 1 1 0 0 0 0
8	1 0 0 0	1 1 1 1 1 1 1
9	1 0 0 1	1 1 1 1 0 1 1

	b1 b0 00	01	11	10
b3 b2 00		0		
01	0			
11	d	d	d	d
10			d	d

$$a = \Pi M(1, 4) + D(10, 11, 12, 13, 14, 15)$$

$$= (b2' + b1 + b0)(b3 + b2 + b1 + b0')$$

Using don’t-cares for segment b.



decimal	BCD	a b c d e f g
0	0 0 0 0	1 1 1 1 1 1 0
1	0 0 0 1	0 1 1 0 0 0 0
2	0 0 1 0	1 1 0 1 1 0 1
3	0 0 1 1	1 1 1 1 0 0 1
4	0 1 0 0	0 1 1 0 0 1 1
5	0 1 0 1	1 0 1 1 0 1 1
6	0 1 1 0	1 0 1 1 1 1 1
7	0 1 1 1	1 1 1 0 0 0 0
8	1 0 0 0	1 1 1 1 1 1 1
9	1 0 0 1	1 1 1 1 0 1 1

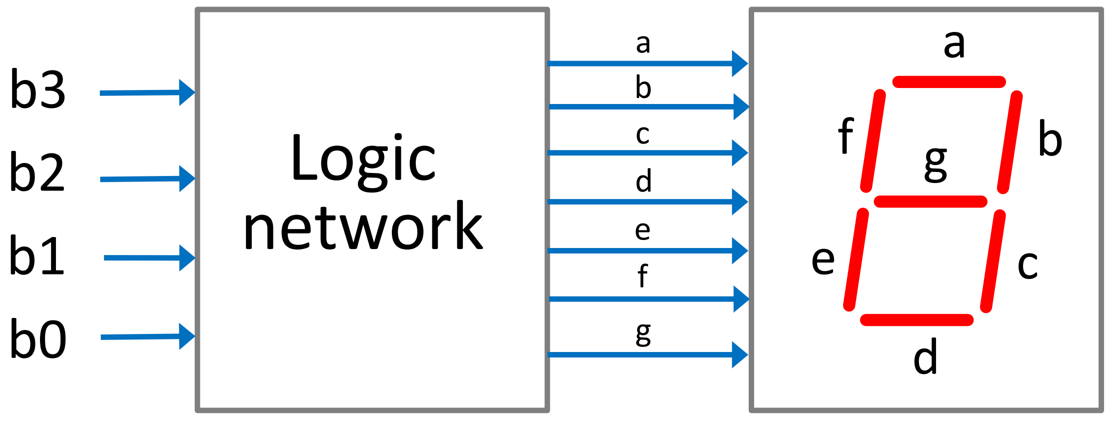
b3 b2 \ b1 b0	00	01	11	10
00				
01		0		0
11	d	d	d	d
10			d	d

$$b = \Pi M(5, 6) + D(10, 11, 12, 13, 14, 15)$$

$$= (b2' + b1 + b0')\,(b2' + b1' + b0)$$

Continue for c, d, e, f and g.

Multiple-input / multiple-output problems



Often offer opportunities to share terms.

(But they can be difficult to find by hand.)

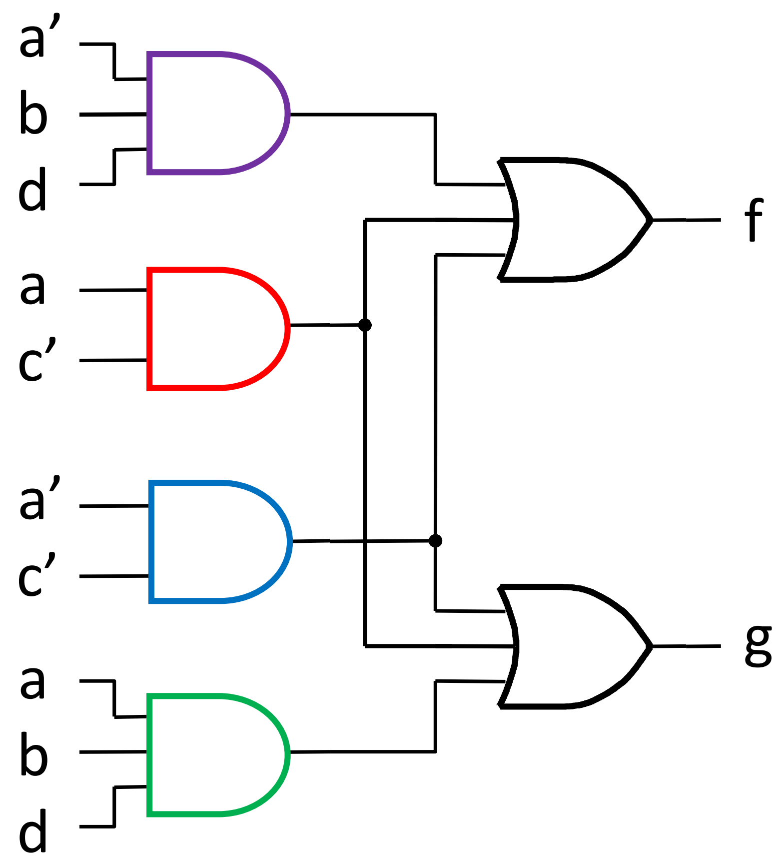
Example: Simple sharing of terms

	cd			
ab	00	01	11	10
00			1	1
01		1	1	1
11	1	1		
10	1	1		

$f = a\,c' + a'\,c + a'\,b\,d$

	cd			
ab	00	01	11	10
00			1	1
01			1	1
11	1	1	1	
10	1	1		

$g = a\,c' + a'\,c + a\,b\,d$



Example: Sharing requiring joint optimization

ab \ cd	00	01	11	10
00				
01	1	1	1	
11	1	1	1	
10		1		

Individually optimized h

ab \ cd	00	01	11	10
00				
01	1		1	1
11	1		1	1
10		1		

Individually optimized j

ab \ cd	00	01	11	10
00				
01	1	1	1	
11	1	1	1	
10		1		

Jointly optimized h

ab \ cd	00	01	11	10
00				
01	1		1	1
11	1		1	1
10		1		

Jointly optimized j

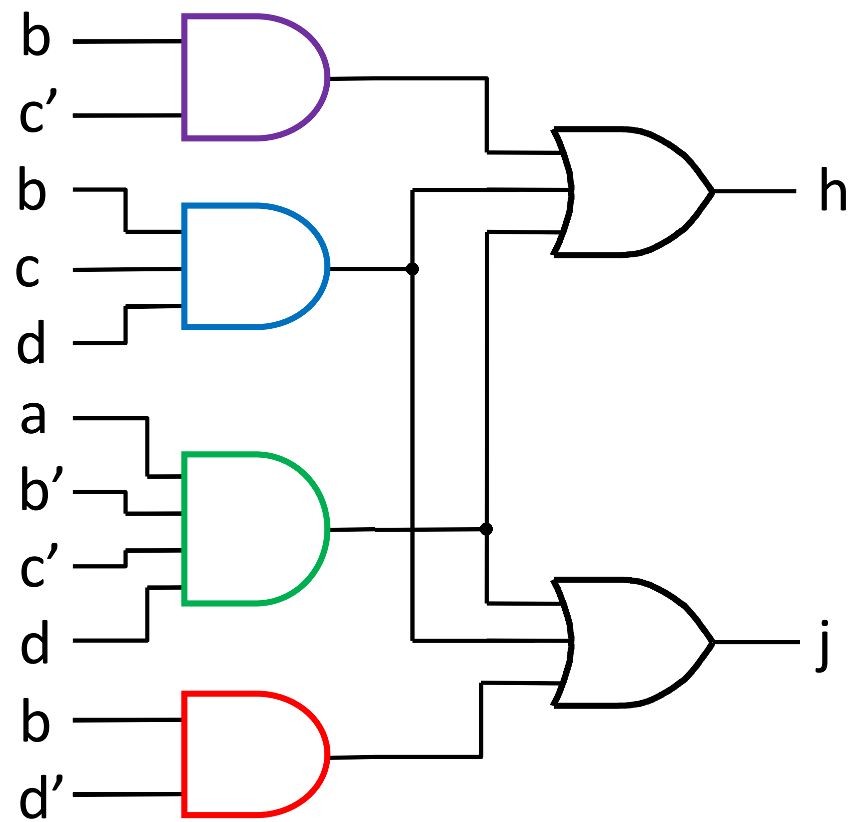
Example: Sharing requiring joint optimization

ab \ cd	00	01	11	10
00				
01	1	1	1	
11	1	1	1	
10		1		

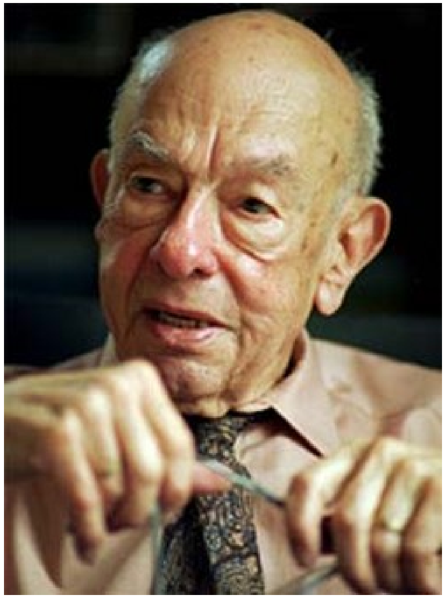
$$h = b\,c' + b\,c\,d + a\,b'\,c'\,d$$

ab \ cd	00	01	11	10
00				
01	1		1	1
11	1		1	1
10		1		

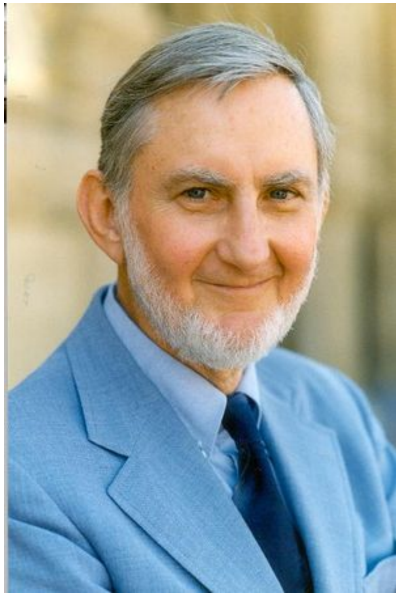
$$j = b\,d' + b\,c\,d + a\,b'\,c'\,d$$



Multiple-input, multiple-output minimization



Willard Van Orman Quine



Edward J. McCluskey

One of the first algorithmic methods was *Quine-McCluskey minimization* invented in 1952, but the runtime grows exponentially with the number of variables.

Compilers today optimize using heuristics.

Image sources: http://ethw.org/Edward_McCluskey
http://www.philosophybasics.com/philosophers_quine.html

Verilog

Designs

The basic decisions for any design.

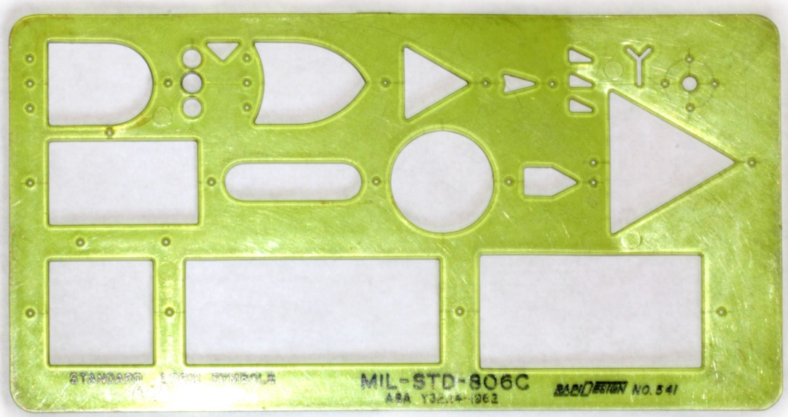
1. What should it do.
2. How should it do it.

You’ll work top-down through the problem by breaking it up into pieces, filling in detail.

As you create your design, you write it out in a programming language. Here it’s Verilog.

A lot of the work is guided by intuition and experience.

40 years ago



Today

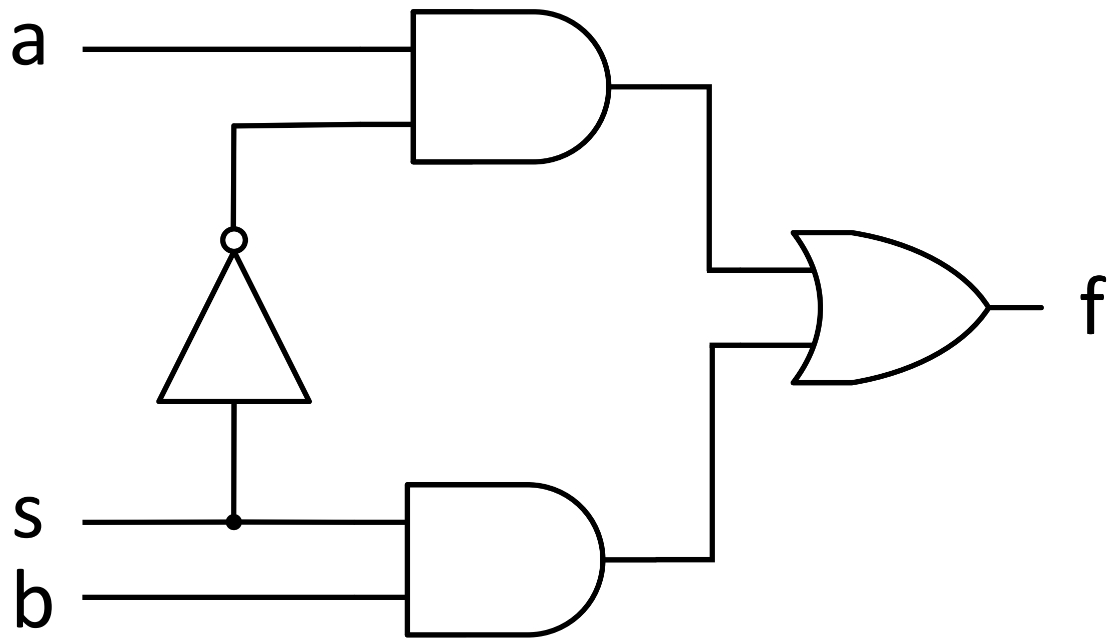
We no longer draw gates for complex designs.

Hardware description languages (HDLs) have replaced schematics.

HDL

A ***hardware description language*** (HDL) allows us to describe logic circuits as if we were writing software in C or Java.

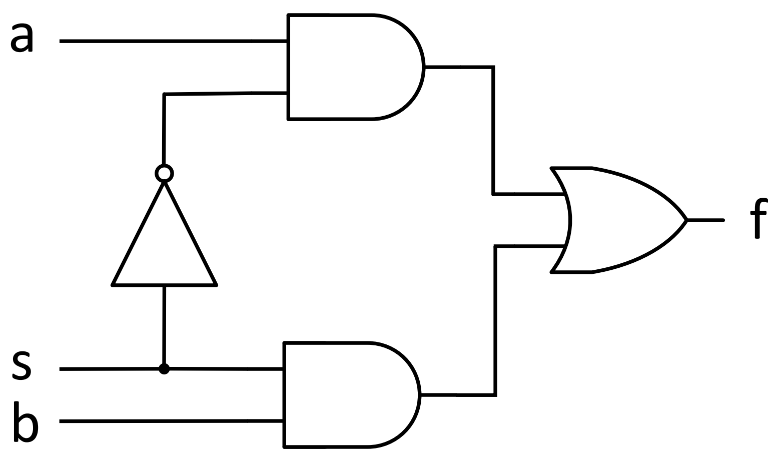
The Multiplexer



s	f
0	a
1	b

Selects a or b based on s, *multiplexing* these signals onto the output f.

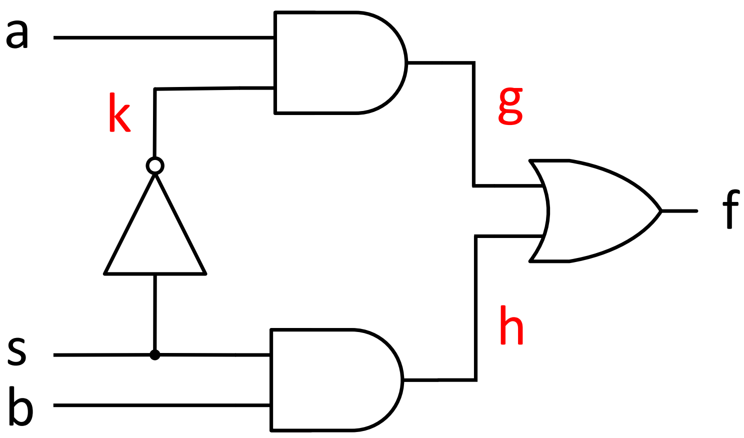
Three ways to represent the multiplexer in Verilog



1. Structural representation as gates.
2. Boolean expressions.
3. Behavioral description.

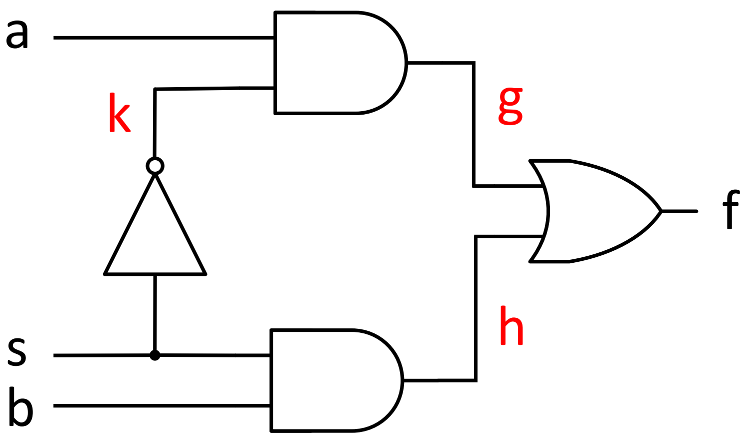
1. Structural representation

Structural representation as gates




```
module Mux2To1A(
      input  s, a, b,
      output f );

   wire g, h, k;
   not ( k, s );
   and ( g, k, a );
   and ( h, s, b );
   or  ( f, g, h );

endmodule
```

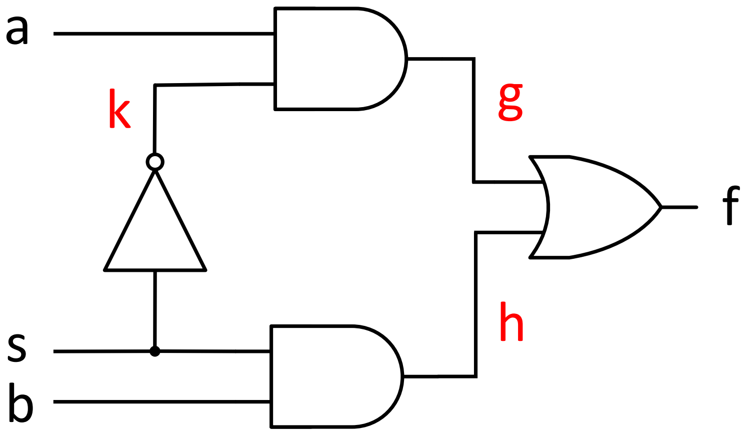
Verilog code for a multiplexer.




```
module Mux2To1A(
        input  s, a, b,
        output f );

    wire g, h, k;
    not ( k, s );
    and ( g, k, a );
    and ( h, s, b );
    or  ( f, g, h );

endmodule
```

The “ports”

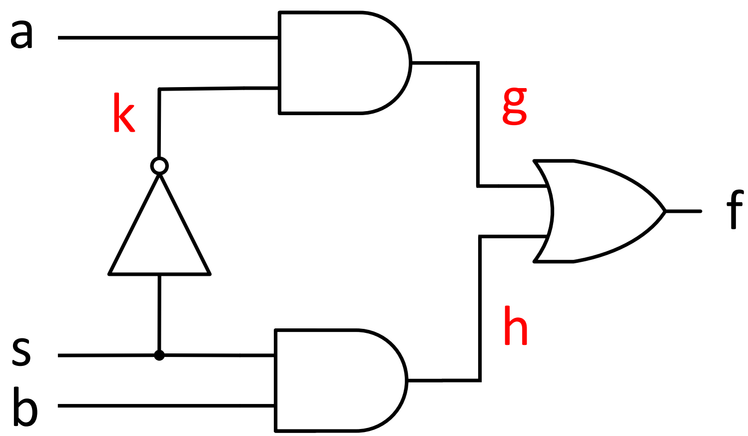
Verilog code for a multiplexer.




```
module Mux2To1C( s, a, b, f );

  input s, a, b;
  output f;

  wire g, h, j, k;
  not ( k, s );
  and ( g, k, a );
  and ( h, s, b );
  or  ( f, g, h );

endmodule
```

Alternate form of specifying the ports.

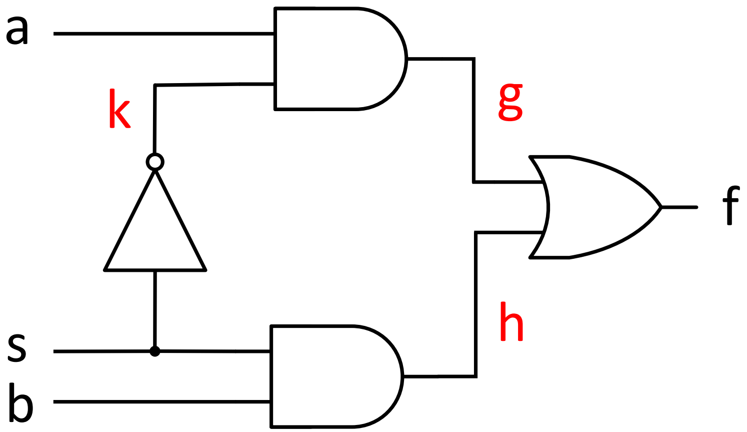
Verilog code for a multiplexer.




```
module Mux2To1A(
        input  s, a, b,
        output f );

   wire g, h, k;
   not ( k, s );
   and ( g, k, a );
   and ( h, s, b );
   or  ( f, g, h );

endmodule
```

Wires constantly reflect the value of whatever they're connected to.

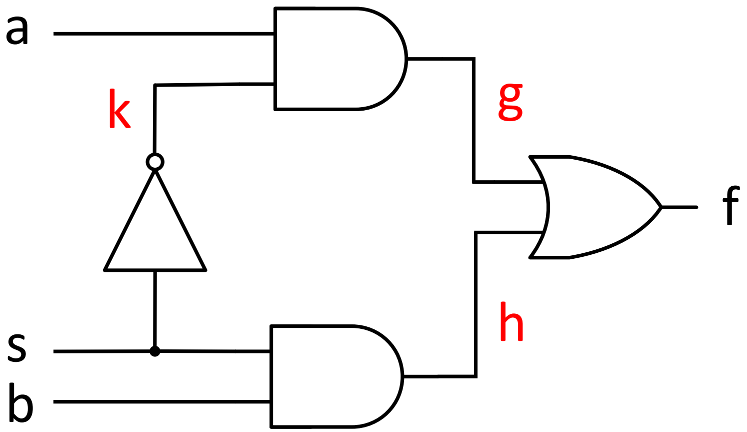
Verilog code for a multiplexer.




```
module Mux2To1B(
        input  s, a, b,
        output f );

    not ( k, s );
    and ( g, k, a );
    and ( h, s, b );
    or  ( f, g, h );

endmodule
```

Undeclared variables default to 1-bit wires.

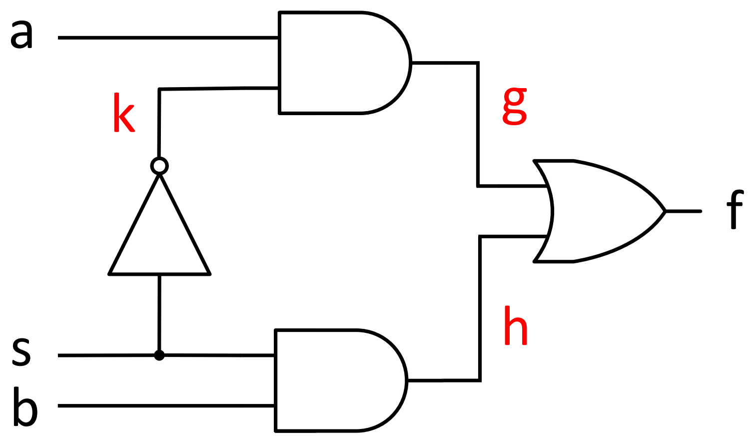
Verilog code for a multiplexer.




```
module Mux2To1A(
        input  s, a, b,
        output f );

    wire g, h, k;
    not ( k, s );
    and ( g, k, a );
    and ( h, s, b );
    or  ( f, g, h );

endmodule
```

The first port to a gate is the output.

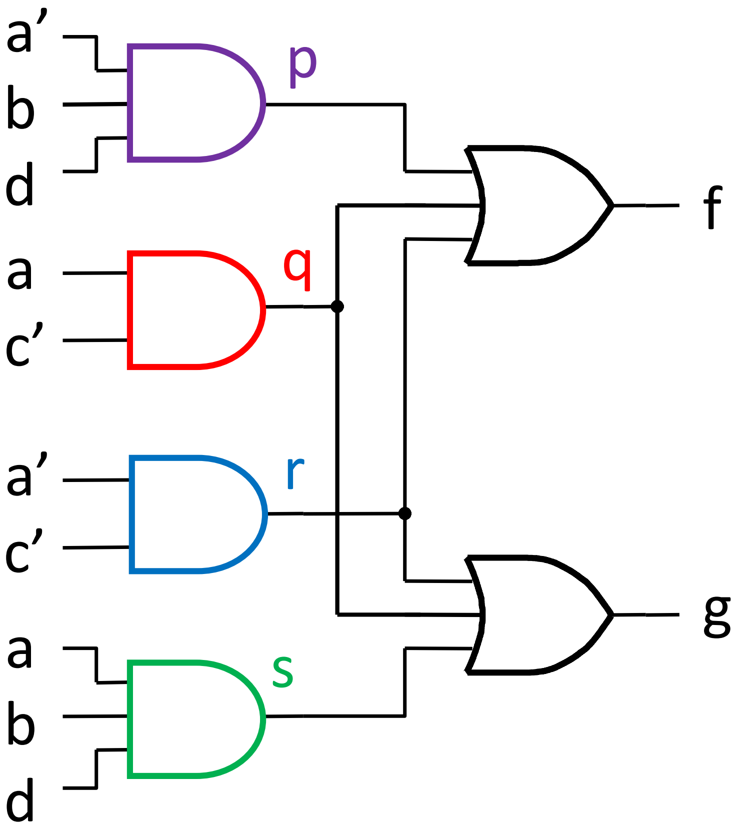
Verilog code for a multiplexer.




```
module Mux2To1A(
        input  s, a, b,
        output f );

     // This 2-in mux module.

     wire g, h, k;
     not ( k, s );
     and ( g, k, a );
     and ( h, s, b );
     or  ( f, g, h );

endmodule
```

Comments start with //.

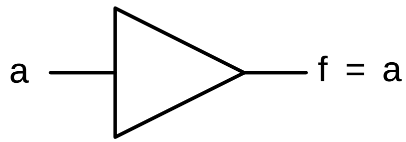
Verilog code for a multiplexer.



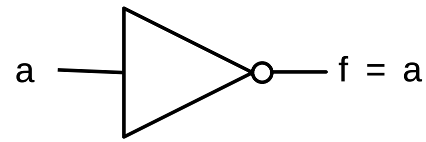

```
module MultiOutput(
        input a, b, c, d,
        output f, g );

    wire p, q, r, s;
    and ( p, ~a, b, d );
    and ( q, a, ~c );
    and ( r, ~a, ~c );
    and ( s, a, b, d );
    or ( f, p, q, r );
    or ( g, q, r, s );

endmodule
```

A multiple output example.

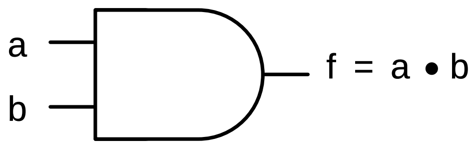
Basic gates in Verilog



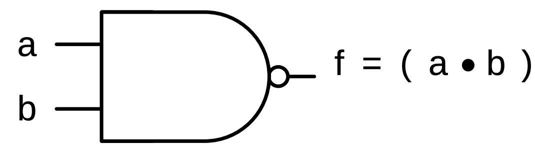
buf(f, a);



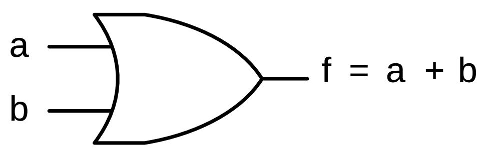
not(f, a);



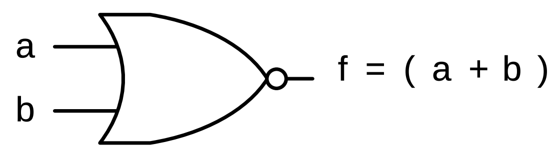
and(f, a, b, …);



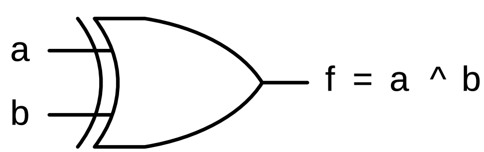
nand(f, a, b, …);



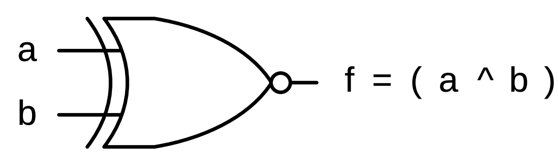
or(f, a, b, …);



nor(f, a, b, …);

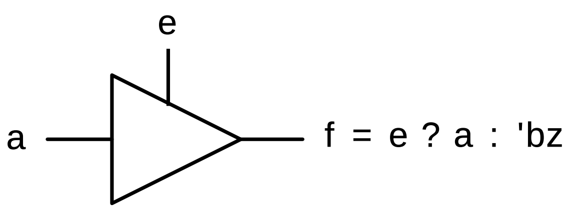


xor(f, a, b, …);

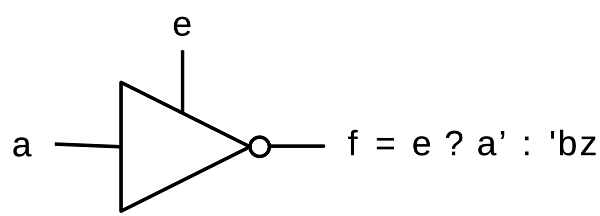


xnor(f, a, b, …);

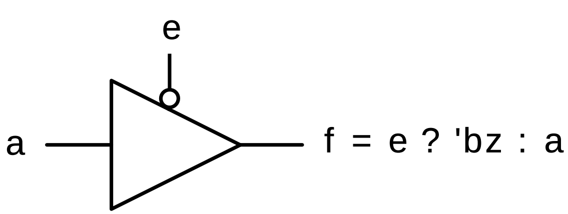
Tri-state drivers in Verilog



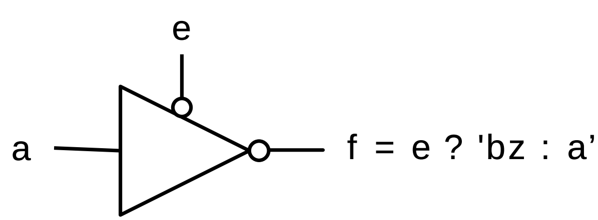
bufif1(f, a, e);



notif1(f, a, e);

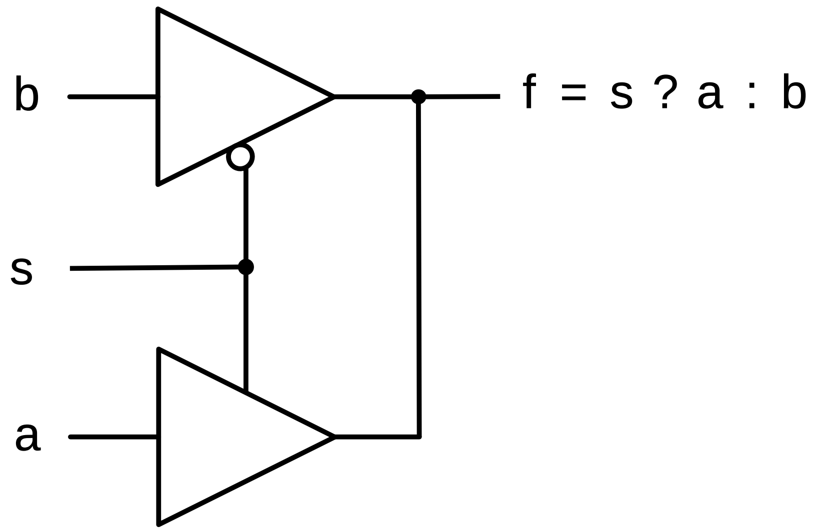


bufif0(f, a, e);

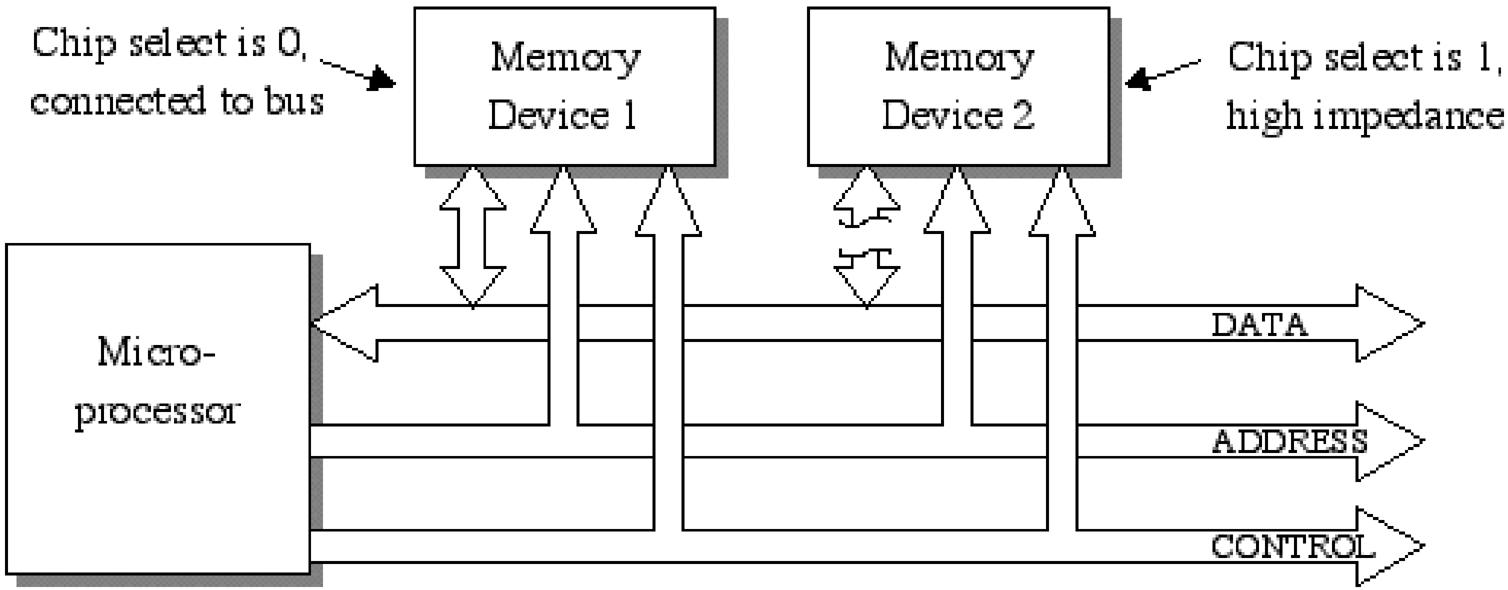


notif0(f, a, e);

A tri-state driver presents a high impedance (high Z) load unless enabled. It’s as if it’s disconnected.



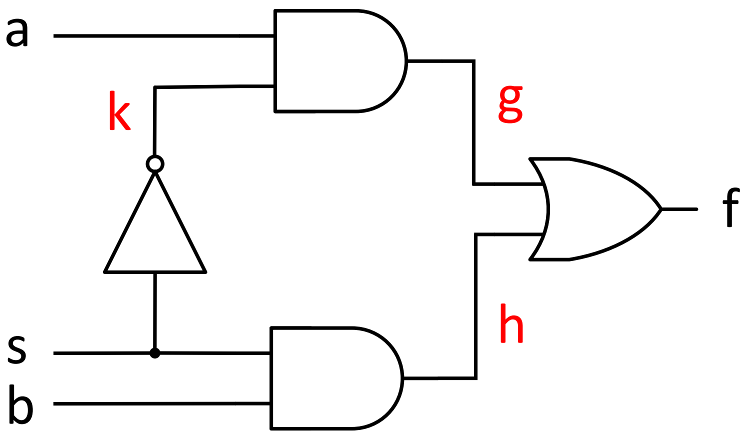
A multiplexer built from 2 tri-state drivers.



A more realistic application as a device or chip select.

Image source: http://faculty.etsu.edu/tarnoff/ntes2150/memory/memory.htm

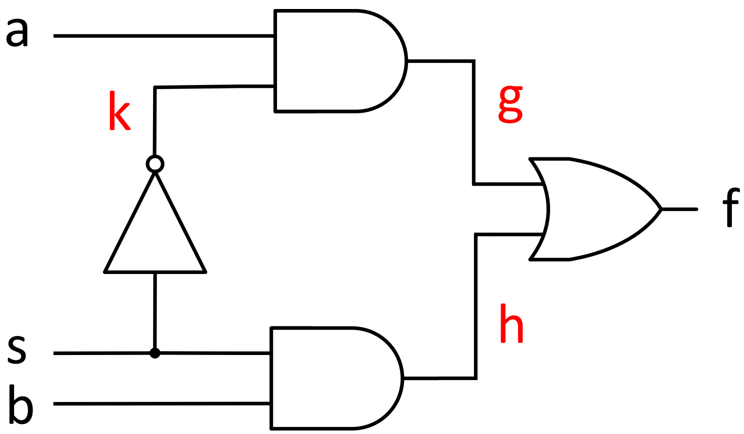
2. Boolean expressions




```
module Mux2To1D(
        input  s, a, b,
        output f );

    assign f = ~s & a | s & b;

endmodule
```

f will continuously reflect the value of the RHS.

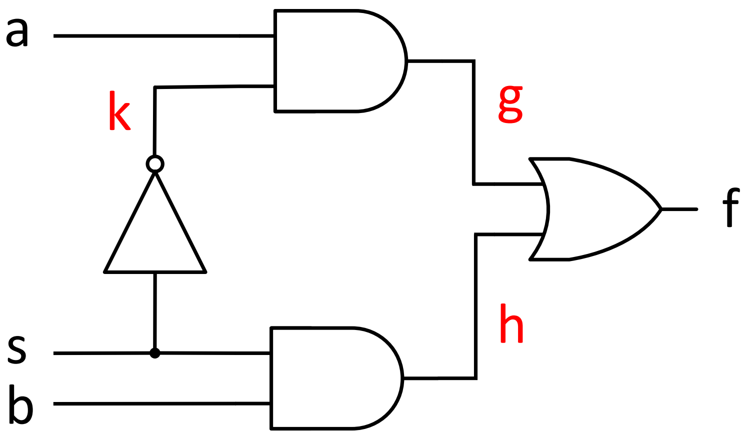
Continuous assignment




```
module Mux2To1D(
        input  s, a, b,
        output f );

    assign f = ~s & a | s & b;

endmodule
```

~ is done before &, which is done before |.

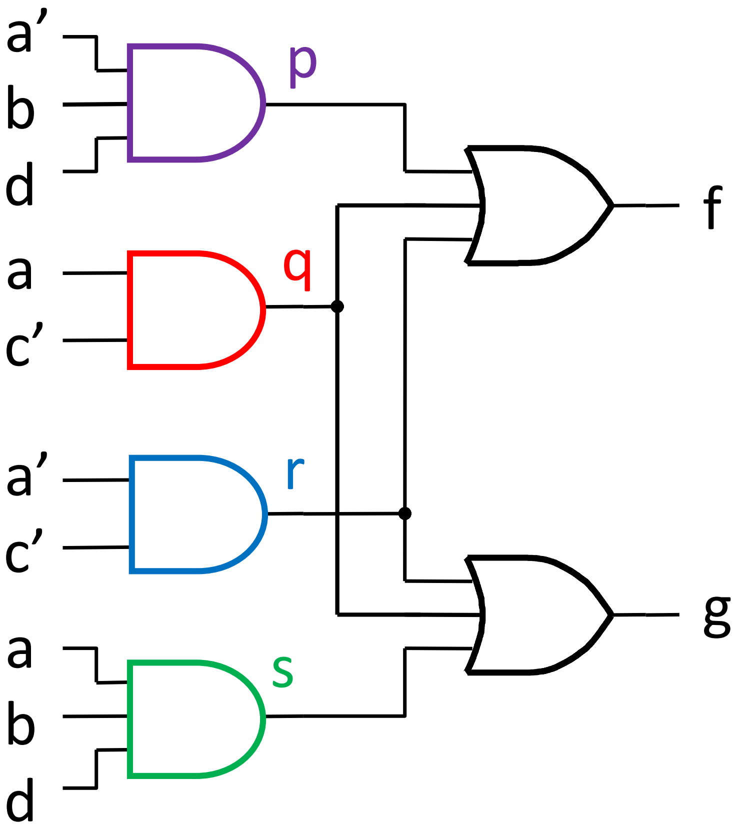
Continuous assignment




```
module Mux2To1E(
        input  s, a, b,
        output f );

    assign f = s ? b : a;

endmodule
```

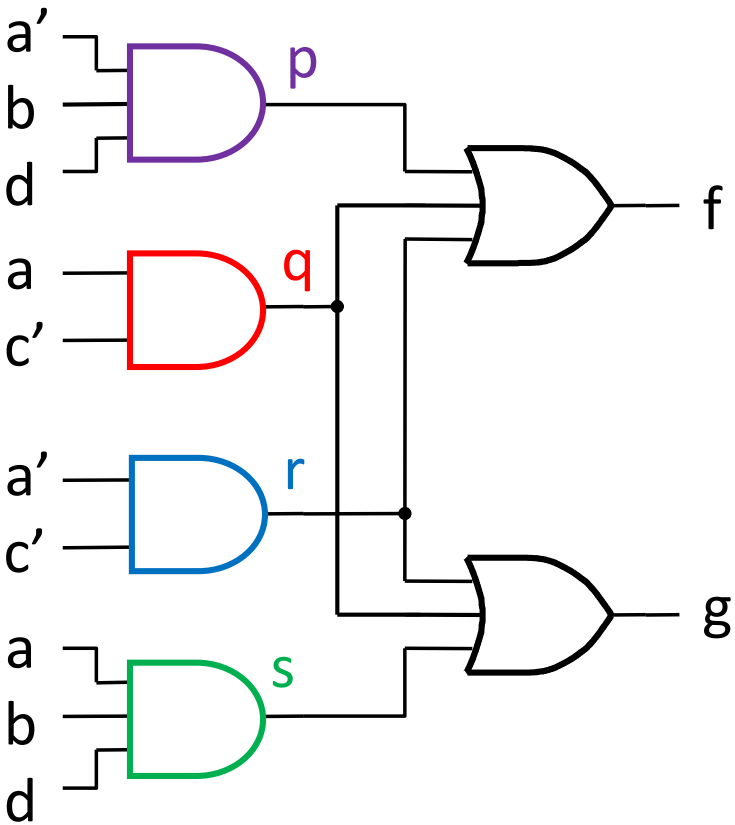
If s is true, f = b,
otherwise, f = a.

The trinary operator.




```
module MultiOutput(
        input a, b, c, d,
        output f, g );

    wire p, q, r, s;
    assign p = ~a & b & d;
    assign q = a & ~c;
    assign r = ~a & ~c;
    assign s = a & b & d;
    assign f = p | q | r;
    assign g = q | r | s;

endmodule
```

A multiple output example.




```
module MultiOutput(
        input a, b, c, d,
        output f, g );

    wire p, q, r, s;
    assign p = ~a & b & d,
           q = a & ~c,
           r = ~a & ~c,
           s = a & b & d,
           f = p | q | r,
           g = q | r | s;

endmodule
```

assign statements can be chained with commas.

Verilog operator precedence

<table>
<tr><td>( )   [ ]</td><td>Grouping.</td></tr>
<tr><td>~   -   !   +   & | ^</td><td>Unary bitwise, arithmetic and logical complements and plus and the AND, OR and XOR reduction operators. Right to left associativity.</td></tr>
<tr><td>*   /   %</td><td>Multiplication, division and remainder.</td></tr>
<tr><td>+   -</td><td>Addition and subtraction.</td></tr>
<tr><td><<   >></td><td>Bit-shifting.</td></tr>
<tr><td><   <=   >=   ></td><td>Relation testing.</td></tr>
<tr><td>==   !=</td><td>Equality testing.</td></tr>
<tr><td>&</td><td>Bitwise AND.</td></tr>
<tr><td>^</td><td>Bitwise XOR.</td></tr>
<tr><td>|</td><td>Bitwise OR.</td></tr>
<tr><td>&&</td><td>Logical AND.</td></tr>
<tr><td>||</td><td>Logical OR.</td></tr>
<tr><td>? :</td><td>Trinary conditional operator.</td></tr>
<tr><td>=   <=</td><td>Blocking and non-blocking assignment.</td></tr>
<tr><td>{ }   { { } }</td><td>Concatenation and replication.</td></tr>
</table>

Source: Table 11-2—Operator precedence and associativity, *IEEE Standard for SystemVarilog*, IEEE Std 1800-2012, IEEE, 2013, p. 221.

Basic Verilog operators

Assume a = 4’b0101 = 5, b = 3’b011 = 3.

Operator	Meaning	Result
Bitwise		
~ a	Bitwise inversion	1010
a & b	Bitwise *AND* = a b	0001
a | b	Bitwise *OR* = a + b	0111
a ^ b	Bitwise *XOR* = a’ b + a b	0110
Logical		
! a	Logical *NOT*: 1 if all bits of a = 0	0
a && b	Logical *AND*: 1 if both a and b are non-zero	1
a || b	Logical *OR*: 1 if either a or b is non-zero	1
Reduction		
& a	*AND* of all bits in a	0
| a	*OR* of all bits in a	1
^ a	*XOR* of all bits in a	0

Basic Verilog operators

Assume a = 4’b0101 = 5, b = 3’b011 = 3.

Operator	Meaning	Result
Relational		
a == b	a *equals* b	0
a != b	a *not equal* b	1
a > b	a *greater than* b	1
a < b	a *less than* b	0
a >= b	a *greater than or equal* b	1

Basic Verilog operators

Assume a = 4’b0101 = 5, b = 3’b011 = 3.

Operator	Meaning	Result
Arithmetic		
- a	Arithmetic complement	-5
+ a	Unary plus.	5
a * b	Multiplication.	15
a / b	Integer division.	1
a % b	Modulo (remainder) division.	2
Shifting		
a >> b	Shift a right b bits	0000
a << b	Shift a left b bits	1000

Basic Verilog operators

Assume a = 2 = 3’b010, b = 3’b011, c = 3’b101.

Operator	Meaning	Result
Concatenation and replication		
{ a, b }	Concate a and b.	010011
{ b { a } }	Replicate and concatenate b copies of a. b must be a constant.	010010010
Conditional (Trinary)		
a ? b : c	If a is non-zero, result = b. Otherwise, result = c.	011