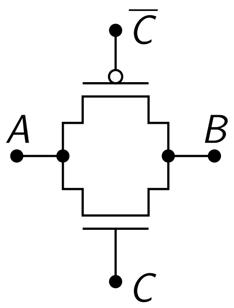
Introduction to Computer Engineering – EECS 203

http://ziyang.eecs.northwestern.edu/~dickrp/eecs203/

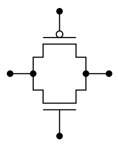
Instructor:	Robert Dick	TA:	Neal Oza
Office:	L477 Tech	Office:	Tech. Inst. L375
Email:	dickrp@northwestern.edu	Phone:	847-467-0033
Phone:	847–467–2298	Email:	nealoza@u.northwestern.edu
		TT:	David Bild
		Office:	Tech. Inst. L470
		Phone:	847-491-2083
		Email:	d-bild@northwestern.edu

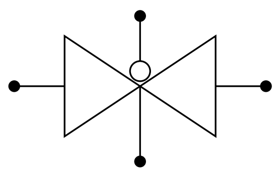
NORTHWESTERN
UNIVERSITY

CMOS transmission gate (TG)

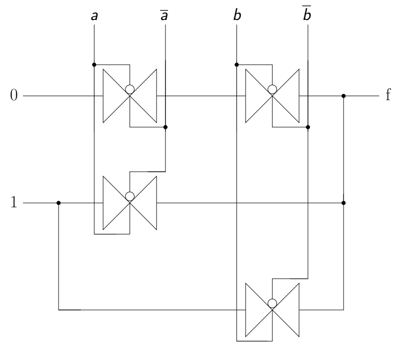


Other TG diagram

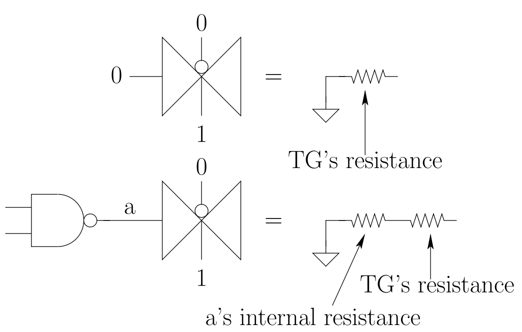




TG example



Impact of control on input



Practice w/ Boolean Minimization

Simplify this expression so that it has the minimal possible literal count:

$$(a + \overline{b})\,\overline{(\overline{a}\,b + cd)}$$

Two-Level Logic and Canonical Forms

- The previous example illustrated one standard representation (product of sums).
- These standard forms are known collectively as two-level logic:
  - Product of Sums (POS) e.g.

$$f = (a + \overline{b})\,(\overline{a} + b)$$

  - Sum of Products (SOP) e.g.

$$f = \overline{a}\,\overline{b} + ab$$

  - Can you see why these two are equivalent?
  - Why is this known as two-level logic?

Canonical forms - SOP

For Sum of Products (SOP) the canonical form is constructed out of *minterms*.

- Product term in which all variables appear in complemented or uncomplemented forms once
- For an n-input function corresponds to one of of 2^n possible input combinations
- Use binary representation to enumerate minterms

Canonical forms – SOP

x	y	z	term	symbol	m_0	m_1	m_2	m_3	m_4	m_5	m_6	m_7
0	0	0	$\overline{x}\,\overline{y}\,\overline{z}$	m_0	1	0	0	0	0	0	0	0
0	0	1	$\overline{x}\,\overline{y}\,z$	m_1	0	1	0	0	0	0	0	0
0	1	0	$\overline{x}\,y\,\overline{z}$	m_2	0	0	1	0	0	0	0	0
0	1	1	$\overline{x}\,yz$	m_3	0	0	0	1	0	0	0	0
1	0	0	$x\overline{y}\,\overline{z}$	m_4	0	0	0	0	1	0	0	0
1	0	1	$x\overline{y}\,z$	m_5	0	0	0	0	0	1	0	0
1	1	0	$xy\overline{z}$	m_6	0	0	0	0	0	0	1	0
1	1	1	xyz	m_7	0	0	0	0	0	0	0	1

Using Canonical Sum of Products Representation

Convenient representation uses $\sum$ operator and minterms:

$$f(x_1,\ldots,x_n) = \sum m_i$$

For given function f, list all minterms for which the function is true:

$$f(a,b,c) = ab + \overline{a}\,\overline{c}$$
$$= (m_6 + m_7) + (m_0 + m_2)$$
$$= \sum m\,(0,2,6,7)$$

Canonical forms – POS

For Products of Sums (POS) the canonical form is constructed out of *maxterms*.

- Sum term in which all variables appear in complemented or uncomplemented forms once
- Use binary representation to enumerate maxterms (note: function is not true for maxterms)

Canonical forms – POS

x	y	z	term	symbol	M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7
0	0	0	$x+y+z$	M_0	0	1	1	1	1	1	1	1
0	0	1	$x+y+\overline{z}$	M_1	1	0	1	1	1	1	1	1
0	1	0	$x+\overline{y}+z$	M_2	1	1	0	1	1	1	1	1
0	1	1	$x+\overline{y}+\overline{z}$	M_3	1	1	1	0	1	1	1	1
1	0	0	$\overline{x}+y+z$	M_4	1	1	1	1	0	1	1	1
1	0	1	$\overline{x}+y+\overline{z}$	M_5	1	1	1	1	1	0	1	1
1	1	0	$\overline{x}+\overline{y}+z$	M_6	1	1	1	1	1	1	0	1
1	1	1	$\overline{x}+\overline{y}+\overline{z}$	M_7	1	1	1	1	1	1	1	0

Using Canonical Products of Sums Representation

Convenient representation uses $\prod$ operator and minterms:

$$f(x_1,\ldots,x_n) = \prod M_i$$

For given function f, list all maxterms for which the function is false:

$$f(a,b,c) = (a+\overline{c})\,(\overline{a}+b)$$
$$= M_1 \cdot M_3 \cdot M_4 \cdot M_5$$
$$= \prod M\,(1,3,4,5)$$

More examples of two-level logic

$$f(a,b,c,d) = \sum m(1,4,8,10,13,15)$$

$$f(w,x,y,z) = \prod M(5,13,14)$$

$$f(u,v) = u + \overline{u}\,\overline{v}$$

Logic minimization motivation

- Want to reduce area, power consumption, delay of circuits
- Hard to exactly predict circuit area from equations
- Can approximate area with SOP cubes
- Minimize number of cubes and literals in each cube
- Algebraic simplification difficult
  - Hard to guarantee optimality

Logic minimization motivation

- K-maps work well for small problems
  - Too error-prone for large problems
  - Don't ensure optimal prime implicant selection
- Quine-McCluskey optimal and can be run by a computer
  - Too slow on large problems
- Some advanced heuristics usually get good results fast on large problems
- Want to learn how these work and how to use them?
- Take Advanced Digital Logic Design

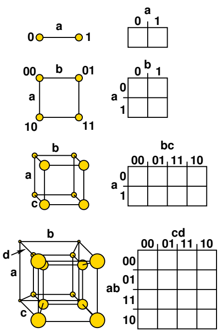
Boolean function minimization

- Algebraic simplification
  - Not systematic
  - How do you know when optimal solution has been reached?
- Optimal algorithm, e.g., Quine-McCluskey
  - Only fast enough for small problems
  - Understanding these is foundation for understanding more advanced methods
- Not necessarily optimal heuristics
  - Fast enough to handle large problems

Karnaugh maps (K-maps)

- Fundamental attribute is adjacency
- Useful for logic synthesis
- Helps logic function visualization
- General Idea: Circle groups of output values (typically 1's)
- Result: Circled terms correspond to minimized product terms

Karnaugh maps



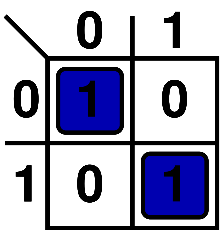
Sum of products (SOP) - Truth table

a	b	f
0	0	1
0	1	0
1	0	0
1	1	1

$f = (\bar{a}\,\bar{b}) + (ab)$

Sum of products (SOP) - KMap

Equivalent way of expressing the same function:



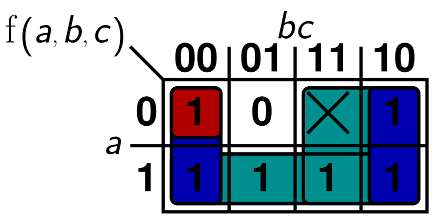
$(\bar{a}\,\bar{b}) + (ab)$

Some definitions

- *implicant* - a product term (or sum term) which covers/includes one or more minterms (or maxterms)
- *prime implicant* - implicant that cannot be covered by a more general implicant (i.e. one with fewer literals)
- *essential prime implicants* - cover an output of the function that no other prime implicant (or sum thereof) is able to cover

Implicants

For now, treat × as a "wildcard"



Prime implicants are not covered by other implicants Essential prime implicants uniquely cover minterms

K-map example

- Minimize $f(a, b, c, d) = \sum(1, 3, 8, 9, 10, 11, 13)$
- f(a, b, c, d) $= a\bar{b} + \bar{b}\,d + a\bar{c}\,d$

K-map simplification technique

For all minterms

- Find maximal groupings of 1's and X's adjacent to that minterm.
- Remember to consider top/bottom row, left/right column, and corner adjacencies.
- These are the prime implicants.

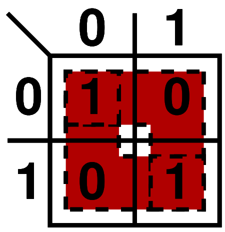
K-map simplification technique

- Revisit the 1's elements in the K-map.
- If covered by single prime implicant, the prime is essential, and participates in final cover.
- The 1's it covers do not need to be revisited.

K-map simplification technique

- If there remain 1's not covered by essential prime implicants,
- Then select the smallest number of prime implicants that cover the remaining 1's.
- This can be difficult for complicated functions.
- Will present an algorithm for this in a future lecture.

Product of sums (POS)



$(\bar{a} + b) \cdot (a + \bar{b})$

POS K-map techniques

- Direct reading by covering zeros and inverting variables

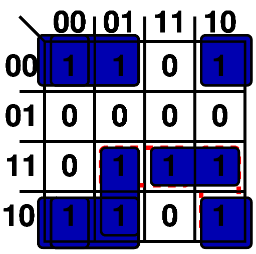
Or

- Invert function
- Do SOP
- Invert again
- Apply De Morgan's laws

POS K-map example

- Minimize $f(a, b, c) = \prod(2, 4, 5, 6)$
- $f(a, b, c) = (\bar{b} + c)(\bar{a} + b)$

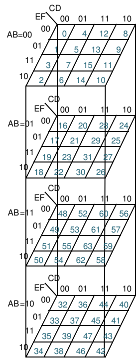
SOP from Karnaugh map

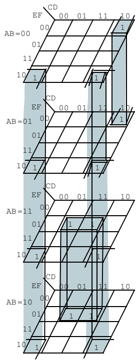


Six-variable K-map example

$$z(a, b, c, d, e, f) = \sum(2, 8, 10, 18, 24, 26, 34, 37, 42, 45, 50, 53, 58, 61)$$

Six-variable K-map example





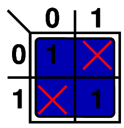
Six-variable K-map example

$$z(a,b,c,d,e,f) = \overline{d}\,e\overline{f} + ad\overline{e}\,f + \overline{a}\,C\overline{d}\,\overline{f}$$

DON'T CARE logic

- All specified Boolean values are 0 or 1
- However, during design some values may be unspecified
  - Don't care values ($\times$)
- $\times$s allow circuit optimization, i.e.,
  - Incompletely specified functions allow optimization

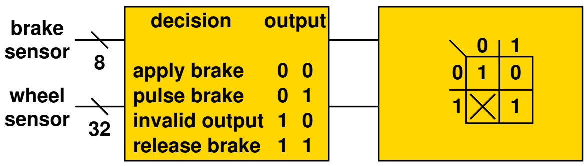
DON'T CARE values



Instead, leave these values undefined ($\times$)

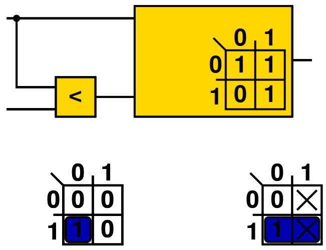
- Also called DON'T CARE values
- Allows any function implementing the specified values to be used
- E.g., could use $(\overline{a}\,\overline{b}) + (ab)$
- However, best to use simpler 1

Satisfiability DON'T CARES



- Input can never occur
- This can happen within a circuit
- Some modules will not be capable of producing certain outputs

Observability DON'T CARES

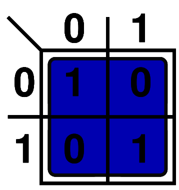


Output will be ignored for certain inputs

Don't care K-map example

- Minimize $f(w,x,y,z) = \sum(1,3,8,9,10,11,13) + d(5,7,15)$
- $f(w,x,y,z) = w\overline{x} + z$

Two-level logic is necessary



Some Boolean functions can not be represented with one logic level

$(\overline{a}\,\overline{b}) + (ab)$

Two-level logic is sufficient

$f(a,b)$	$b=0$	$b=1$
$a=0$	1	0
$a=1$	0	1

- All Boolean functions can be represented with two logic levels
- Given k variables, 2^K minterm functions exist
- Select arbitrary union of minterms

Two-level well-understood

- As we will see later, optimal minimization techniques known for two-level
- However, optimal two-level solution may not be optimal solution
  - Sometimes a suboptimal solution to the right problem is better than the optimal solution to the wrong problem

Two-level sometimes impractical

$f(a,b,c,d)$ ab \ cd	00	01	11	10
00	0	1	0	1
01	1	0	1	0
11	0	1	0	1
10	1	0	1	0

Consider a 4-term XOR (parity) gate: $a \oplus b \oplus c \oplus d$
$(\bar{a}\,\bar{b}\,\bar{c}\,d) + (\bar{a}\,\bar{b}\,c\,\bar{d}) + (\bar{a}\,b\,\bar{c}\,\bar{d}) + (\bar{a}\,bcd) + (ab\bar{c}\,d) + (abc\bar{d}) + (a\bar{b}\,\bar{c}\,\bar{d}) + (a\bar{b}\,cd)$

Two-level weakness

- Two-level representation is exponential
- However, it's a simple concept
  - Is $\sum_i^n x_i$ odd?
- Problem with representation, not function

Two-level weakness

Two-level representations also have other weaknesses

- Conversion from SOP to POS is difficult
  - Inverting functions is difficult
  - $\cdot$-ing two SOPs or $+$ing two POSs is difficult
- Neither general POS or SOP are canonical
  - Equivalence checking difficult
- POS satisfiability $\in \mathcal{NP}$-complete

Reading assignment

- M. Morris Mano and Charles R. Kime. *Logic and Computer Design Fundamentals*. Prentice-Hall, NJ, fourth edition, 2008
- Section 2.6
- Also read TTL reference, Don Lancaster. *TTL Cookbook*. Howard W. Sams & Co., Inc., 1974, as needed