EECS150 - Digital Design
Lecture 7 - Boolean Algebra II

February 12, 2002

John Wawrzynek

Outline

- Canonical Forms
  - They give us a method to go from TT to Boolean Equations
- Two-level Logic Simplification
  - K-map method
- Multi-level Logic
- NAND/NOR networks
- EXOR revisited

Canonical Forms

- Standard form for a Boolean expression - unique algebraic expression from a TT.
- Two Types:
  - *Sum of Products (SOP)*
  - *Product of Sums (POS)*
- **Sum of Products** (disjunctive normal form, minterm expansion).

Example:

minterms	a b c	f	f'
a'b'c'	0 0 0	0	1
a'b'c	0 0 1	0	1
a'bc'	0 1 0	0	1
a'bc	0 1 1	1	0
ab'c'	1 0 0	1	0
ab'c	1 0 1	1	0
abc'	1 1 0	1	0
abc	1 1 1	1	0

One product (and) term for each 1 in f:

f = a'bc + ab'c' + ab'c +abc' +abc

f' = a'b'c' + a'b'c + a'bc'

Sum of Products (cont.)

Canonical Forms are usually not minimal:

Our Example:

f = a'bc + ab'c' + ab'c + abc' +abc

= a'bc + ab' + ab

= a'bc + a (x'y + x = y + x)

= a + bc

f' = a'b'c' + a'b'c + a'bc'

= a'b' + a'bc'

= a' (b' + bc')

= a' (b' + c')

= a'b' + a'c'

Canonical Forms

- **Product of Sums** (conjunctive normal form, maxterm expansion).

Example:

maxterms	a b c	f	f'
a+b+c	0 0 0	0	1
a+b+c'	0 0 1	0	1
a+b'+c	0 1 0	0	1
a+b'+c'	0 1 1	1	0
a'+b+c	1 0 0	1	0
a'+b+c'	1 0 1	1	0
a'+b'+c	1 1 0	1	0
a'+b'+c'	1 1 1	1	0

One sum (or) term for each 0 in f:

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

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

Mapping from SOP to POS (or POS to SOP): *Derive TT then proceed.*

Two-level Logic Simplication

Key tool: The Uniting Theorem

x (y' + y) = x (1) = x

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

f = ab' + ab = a(b'+b) = a

b values change within the on-set rows

a values don't change

b is eliminated, a remains

a	b	g
0	0	1
0	1	0
1	0	1
1	1	0

g = a'b'+ab' = (a'+a)b' =b'

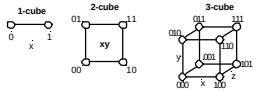
b values stay the same

a values changes

b' remains, a eliminated

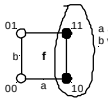
Boolean Cubes

Visual technique for identifying when the Uniting Theorem can be applied



- Sub-cubes of on nodes can be used for simplification.
  - On-set - filled in nodes, off-set - empty nodes

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



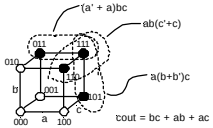
a asserted & unchanged
b varies

ab + ab' = a

3-variable cube example

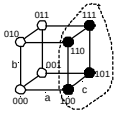
FA carry out:

a	b	c	cout
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



cout = bc + ab + ac

What about larger sub-cubes?



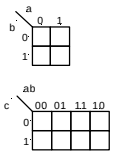
ab'c' + ab'c + abc' + abc

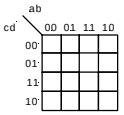
ac' + ac + ab = a + ab = a

- Both b & c change, a is asserted & remains constant.

Karnaugh Map Method

- K-map is an alternative method of representing the TT and to help visual the adjacencies.

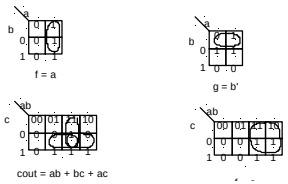




5 & 6 variable k-maps possible

Karnaugh Map Method

- Examples



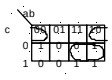
f = a

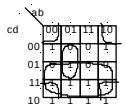
g = b'

cout = ab + bc + ac

1. Circle the largest groups possible.
2. Group dimensions must be a power of 2.

K-maps (cont.)

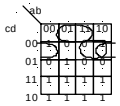




f = c + a'bd + b'd'

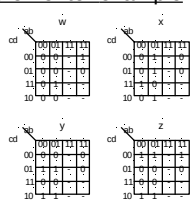
(bigger groups are better)

Circling Zeros



BCD incrementer example

a b c d	w x y z
0 0 0 0	0 0 0 1
0 0 0 1	0 0 1 0
0 0 1 0	0 0 1 1
0 0 1 1	0 1 0 0
0 1 0 0	0 1 0 1
0 1 0 1	0 1 1 0
0 1 1 0	0 1 1 1
0 1 1 1	1 0 0 0
1 0 0 0	1 0 0 1
1 0 0 1	0 0 0 0
1 0 1 0	- - - -
1 0 1 1	- - - -
1 1 0 0	- - - -
1 1 0 1	- - - -
1 1 1 0	- - - -
1 1 1 1	- - - -



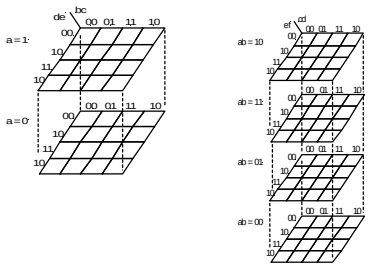
w =

x =

y =

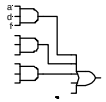
z =

Higher Dimensional K-maps



Multi-level Combinational Logic

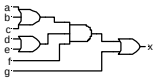
- Example: reduced sum-of-products form
  x = adf + aef + bdf + bef + cdf + cef + g
- implementation in 2-levels with gates:



cost: 7-input OR, 6 3-input AND
50 transistors + 25 wires
(19 literal plus 6 internal)

delay: 3-input AND gate delay + 7-input OR gate delay

- Factored form:
  x = (a + b +c)(d + e)f + g



cost: 1 3-input OR, 2 2-input OR, 1 3-input AND
20 transistors

delay: 3-input OR + 3-input AND + 2-input OR

Which is faster?

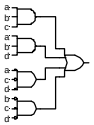
In general: Using multiple levels (more than 2) will reduce the cost. Sometimes also delay. Sometime a tradeoff between cost and delay.

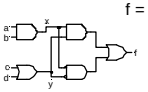
Multi-level Combinational Logic

Example: F = abc + abd +a'c'd' + b'c'd'

let x = ab y = c+d

f = xy + x'y'





No convenient hand methods for multi-level logic simplification:

1 CAD Tools, example misII (UCB)
2 exploit some special structure, example adder

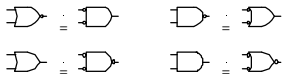
Are these optimizations still relevant for LUT implementations?

NAND-NAND & NOR-NOR Networks

DeMorgan's Law:

(a + b)' = a' b' (a b)' = a' + b'

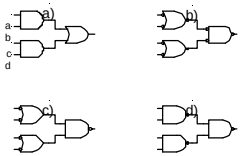
b + b = (a' b')' (a b) = (a' + b')'



push bubbles or *introduce in pairs* or *remove pairs.*

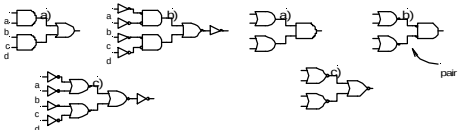
NAND-NAND & NOR-NOR Networks

- Mapping from AND/OR to NAND/NAND

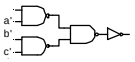


NAND-NAND & NOR-NOR Networks

- Mapping AND/OR to NOR/NOR
- Mapping OR/AND to NOR/NOR

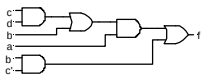


- OR/AND to NAND/NAND

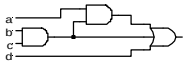


Multi-level Networks

F = a(b + cd) + bc'



Convert to NANDs (note fanout)

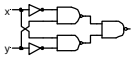


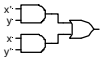
EXOR Function

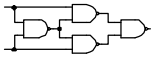
Parity, addition mod 2

x xor y = x'y + xy'

x	y	xor	xnor
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1







Another approach:

if x=0 then y else y'