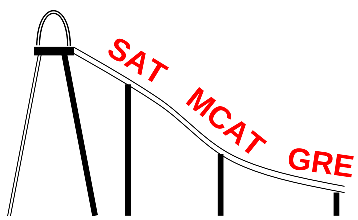
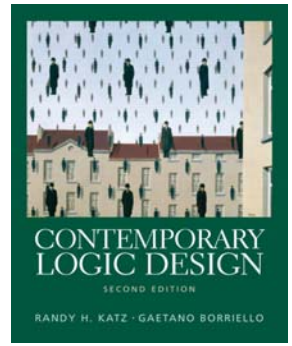
Test Slide



CSE 370 Spring 2006
Introduction to Digital Design

Lecture 5: Canonical Forms



Last Lecture
- Logic Gates
- Different Implementations
- Bubbles

Today
- Canonical Forms
- Sum of Products
- Product of Sums
- Boolean Cubes

Administrivia

- Homework 2 was modified Monday evening. See online for the modification. The two problems that were dropped will appear on the next homework.

Puzzle

Suppose a light has to be lit by switching on simultaneously n switches. We may push any one of the n buttons at any time but we don't know if they are on or off. What is the smallest number of steps necessary to guarantee that we turn the light on starting from any initial configuration of the switches?

Canonical Forms

- Unique forms for Boolean functions
- Unique algebraic signatures:
- Generically not the simplest
  - Can be simplified

- Two canonical forms
  - Sum of products
  - Product of sums

Sum of Products Canonical Form

- Also called disjunctive normal form (DNF)
  - Commonly called a **minterm expansion**

minterm

001 011 101 110 111

F = A'B'C + A'BC + AB'C + ABC' + ABC

A	B	C	F	F'
0	0	0	0	1
0	0	1	1	0
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	0

F' = A'B'C' + A'BC' + AB'C'

Minterms

- Variables appears exactly once in each minterm
  - In true or inverted form (but not both)

A	B	C	minterms	
0	0	0	A'B'C'	m0
0	0	1	A'B'C	m1
0	1	0	A'BC'	m2
0	1	1	A'BC	m3
1	0	0	AB'C'	m4
1	0	1	AB'C	m5
1	1	0	ABC'	m6
1	1	1	ABC	m7

short-hand notation

F in canonical form:

F(A,B,C) = Σm(1,3,5,6,7)
= m1 + m3 + m5 + m6 + m7
= A'B'C+A'BC+AB'C+ABC'+ABC

canonical form → minimal form

F(A,B,C) = A'B'C+A'BC+AB'C+ABC+ABC'
= (A'B'+A'B+AB'+AB)C+ABC'
= ((A' + A)(B' + B))C + ABC'
= ABC' + C
= AB + C

Product of Sums Canonical Form

- Also called conjunctive normal form (CNF)
  - Commonly called a **maxterm expansion**

maxterm

000 010 100

F = (A + B + C) (A + B' + C) (A' + B + C)

A	B	C	F	F'
0	0	0	0	1
0	0	1	1	0
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	0

F' = (A+B+C')(A+B'+C')(A'+B+C')(A'+B'+C)(A'+B'+C')

Maxterms

- Variables appears exactly once in each maxterm
  - In true or inverted form (but not both)

A	B	C	maxterms	
0	0	0	A+B+C	M0
0	0	1	A+B+C'	M1
0	1	0	A+B'+C	M2
0	1	1	A+B'+C'	M3
1	0	0	A'+B+C	M4
1	0	1	A'+B+C'	M5
1	1	0	A'+B'+C	M6
1	1	1	A'+B'+C'	M7

short-hand notation

F in canonical form:

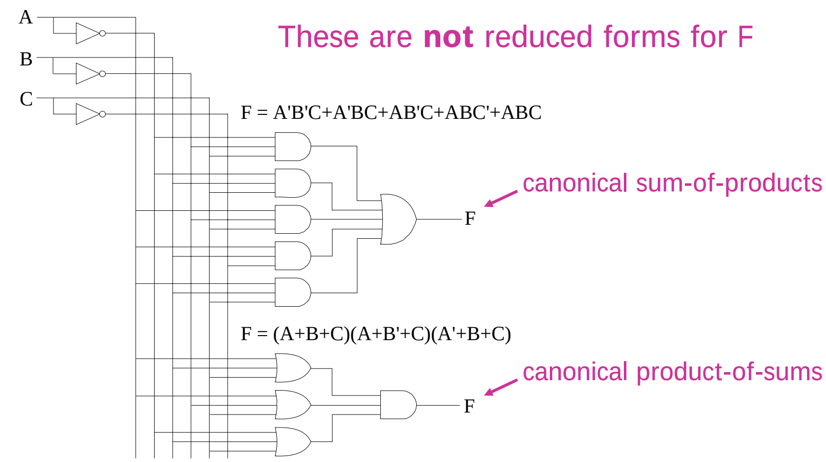
$$\begin{aligned} F(A,B,C) &= \Pi M(0,2,4) \\ &= M0 \bullet M2 \bullet M4 \\ &= (A+B+C)(A+B'+C)(A'+B+C) \end{aligned}$$

canonical form $\rightarrow$ minimal form

$$\begin{aligned} F(A,B,C) &= (A+B+C)(A+B'+C)(A'+B+C) \\ &= (A+B+C)(A+B'+C) \bullet \\ &\quad\; (A+B+C)(A'+B+C) \\ &= (A + C)(B + C) \end{aligned}$$

Canonical Decompositions of F=AB+C

These are **not** reduced forms for F



Exercise

Express the following binary function in canonical sum of products and product of sum form:

A	B	C	F
0	0	0	1
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	0

POS, SOP, and DeMorgan

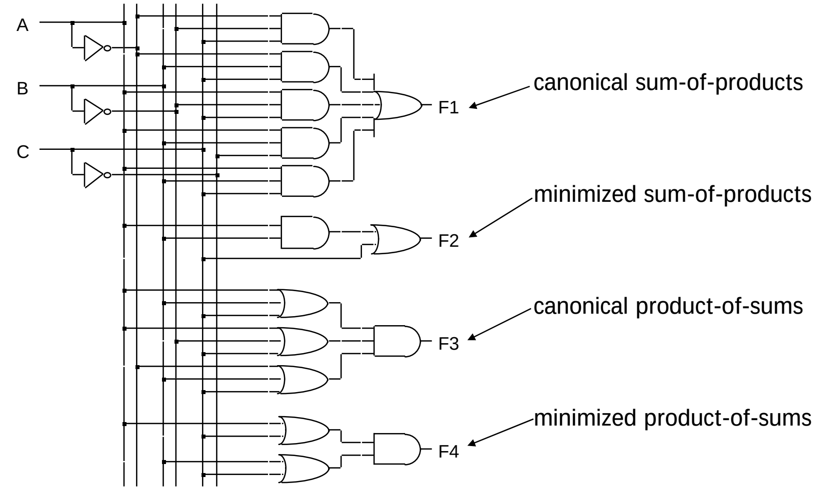
- Sum-of-products
  - F' = A'B'C' + A'BC' + AB'C'
- Apply de Morgan's to get POS
  - (F')' = (A'B'C' + A'BC' + AB'C')'
  - F = (A+B+C)(A+B'+C)(A'+B+C)

- Product-of-sums
  - F' = (A+B+C')(A+B'+C')(A'+B+C')(A'+B'+C)(A'+B'+C')
- Apply de Morgan's to get SOP
  - (F')' = ((A+B+C')(A+B'+C')(A'+B+C')(A'+B'+C)(A'+B'+C'))'
  - F = A'B'C + A'BC + AB'C + ABC' + ABC

Conversions Between Canonical Forms

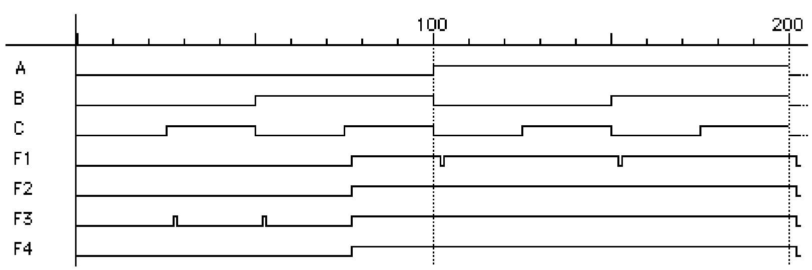
- Minterm to maxterm
  - Use maxterms that aren't in minterm expansion
  - $F(A,B,C) = \sum m(1,3,5,6,7) = \prod M(0,2,4)$
- Maxterm to minterm
  - Use minterms that aren't in maxterm expansion
  - $F(A,B,C) = \prod M(0,2,4) = \sum m(1,3,5,6,7)$
- Minterm of F to minterm of F'
  - Use minterms that don't appear
  - $F(A,B,C) = \sum m(1,3,5,6,7)$ $F'(A,B,C) = \sum m(0,2,4)$
- Maxterm of F to maxterm of F'
  - Use maxterms that don't appear
  - $F(A,B,C) = \prod M(0,2,4)$ $F'(A,B,C) = \prod M(1,3,5,6,7)$

Alternative Implementations of F=AB+C



Waveforms of Four Alternatives

- Waveforms are essentially identical
  - except for timing hazards (glitches)
  - delays almost identical (modeled as a delay per level, not type of gate or number of inputs to gate)



Incompletely Specified Functions

- Example: binary coded decimal increment by 1
  - BCD digits encode the decimal digits 0 – 9 in the bit patterns 0000 – 1001

	A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	0	1
1	0	0	0	1	0	0	1	0
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	1	0	0
4	0	1	0	0	0	1	0	1
5	0	1	0	1	0	1	1	0
6	0	1	1	0	0	1	1	1
7	0	1	1	1	1	0	0	0
8	1	0	0	0	1	0	0	1
9	1	0	0	1	0	0	0	0
	1	0	1	0	X	X	X	X
	1	0	1	1	X	X	X	X
	1	1	0	0	X	X	X	X
	1	1	0	1	X	X	X	X
	1	1	1	0	X	X	X	X
	1	1	1	1	X	X	X	X

off-set of W

on-set of W

don't care (DC) set of W

these inputs patterns should never be encountered in practice – **"don't care"** about associated output values, can be exploited in minimization

Incompletely Specified Functions: Notation

- Don't cares and canonical forms
  - so far, only represented on-set
  - also represent don't-care-set
  - need two of the three sets (on-set, off-set, dc-set)
- Canonical representations of the BCD increment by 1 function:
  - Z = m0 + m2 + m4 + m6 + m8 + d10 + d11 + d12 + d13 + d14 + d15
  - Z = Σ [m(0,2,4,6,8) + d(10,11,12,13,14,15)]
  - Z = M1 • M3 • M5 • M7 • M9 • D10 • D11 • D12 • D13 • D14 • D15
  - Z = Π [M(1,3,5,7,9) • D(10,11,12,13,14,15)]

Simplification of Two Level Logic

- Find a minimal sum of products or product of sums realization
  - exploit don't care information in the process
- Algebraic simplification
  - not an algorithmic/systematic procedure
  - how do you know when the minimum realization has been found?
- Computer-aided design tools
  - precise solutions require very long computation times, especially for functions with many inputs (> 10)
  - heuristic methods employed – "educated guesses" to reduce amount of computation and yield good if not best solutions
- Hand methods still relevant
  - to understand automatic tools and their strengths and weaknesses
  - ability to check results (on small examples)

Uniting Theorem

- Key tool to simplification: A (B' + B) = A
- Essence of simplification of two-level logic
  - find two element subsets of the ON-set where only one variable changes its value – this single varying variable can be eliminated and a single product term used to represent both elements

F = A'B'+AB' = (A'+A)B' = B'

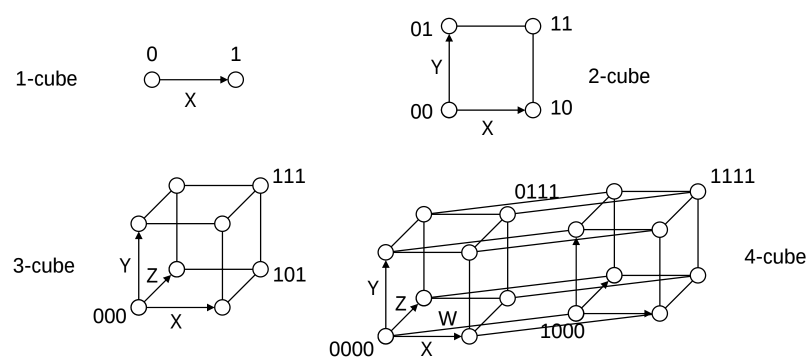
A	B	F
0	0	1
0	1	0
1	0	1
1	1	0

B has the same value in both on-set rows – B remains

A has a different value in the two rows – A is eliminated

Boolean Cubes

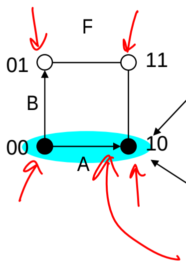
- Visual technique for identifying when the uniting theorem can be applied
- n input variables = n-dimensional "cube"



Truth Tables To Boolean Cubes

- Uniting theorem combines two "faces" of a cube into a larger "face"
- Example:

A	B	F
0	0	1
0	1	0
1	0	1
1	1	0



two faces of size 0 (nodes) combine into a face of size 1(line)

A varies within face, B does not this face represents the literal B'

ON-set = solid nodes
OFF-set = empty nodes
DC-set = ×'d nodes

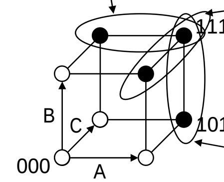
Three Variable Example

- Binary full-adder carry-out logic

$A'BC_{in} + ABC_{in} = BC_{in}$

A	B	Cin	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

(A'+A)BCin $= BC_{in}$



AB(Cin'+Cin)

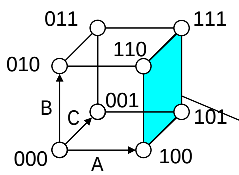
A(B+B')Cin

the on-set is completely covered by the combination (OR) of the subcubes of lower dimensionality - note that "111" is covered three times

Cout = BCin+AB+ACin

Three Variable Example

- Sub-cubes of higher dimension than 2



F(A,B,C) = Σm(4,5,6,7)

on-set forms a square
i.e., a cube of dimension 2

represents an expression in one variable
i.e., 3 dimensions – 2 dimensions

A is asserted (true) and unchanged
B and C vary

This subcube represents the literal A

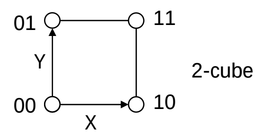
m-dimensional cubes in a n-dimensional Boolean space

- In a 3-cube (three variables):
  - a 0-cube, i.e., a single node, yields a term in 3 literals
  - a 1-cube, i.e., a line of two nodes, yields a term in 2 literals
  - a 2-cube, i.e., a plane of four nodes, yields a term in 1 literal
  - a 3-cube, i.e., a cube of eight nodes, yields a constant term "1"
- In general,
  - an m-subcube within an n-cube (m < n) yields a term with n – m literals

Gray Codes

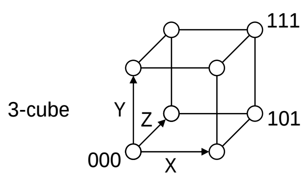
- A listing of the vertices of the Boolean cube in which we only move along the edges.

Example:



0	00
1	01
2	11
3	10

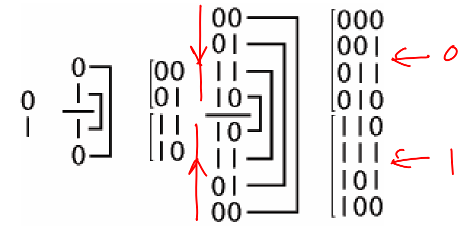
Example:



0	000
1	001
2	011
3	010
4	110
5	111
6	101
7	100

Gray Codes

- Binary reflected Gray codes

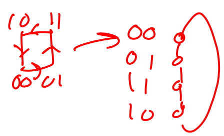


Gray code on n bits => Gray code on n+1 bits

Append 0 to Gray code of n bits, followed by appending 1 to the reverse-ordered Gray code of n bits

Gray Code Uses

- Use Gray codes to help visualize higher dimensional Boolean cubes.



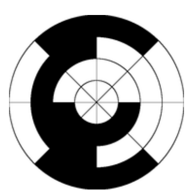
- Avoid spurious intermediate states

  We want $001 \Rightarrow 110$

  We get $001 \Rightarrow 000 \Rightarrow 010 \Rightarrow 110$

  Gray codes help avoid synchronizing errors

- Example: measuring angles



Puzzle

Suppose a light has to be lit by switching on simultaneously n switches. We may push any one of the n buttons at any time but we don't know if they are on or off. What is the smallest number of steps necessary to guarantee that we turn the light on starting from any initial configuration of the switches?