Two-Level
Logic Minimization

Prof. Srinivas Devadas

MIT

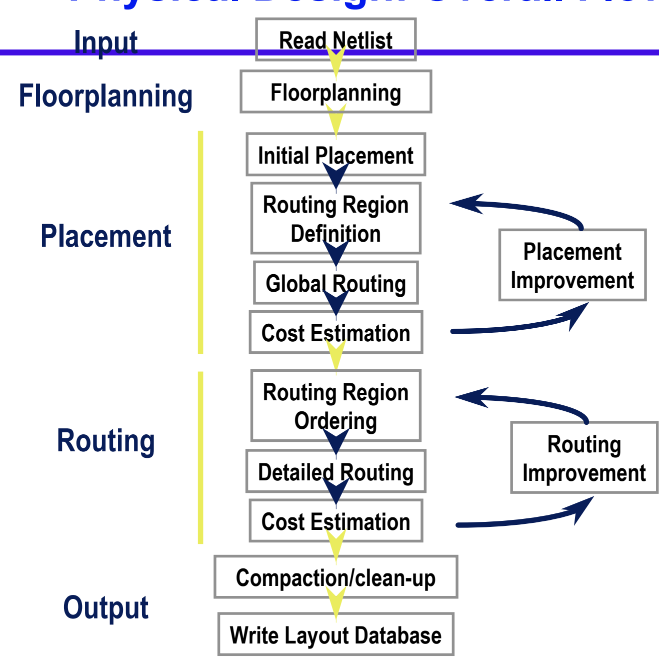
Prof. Kurt Keutzer

Prof. Richard Newton

University of California

Berkeley, CA

Physical Design: Overall Flow

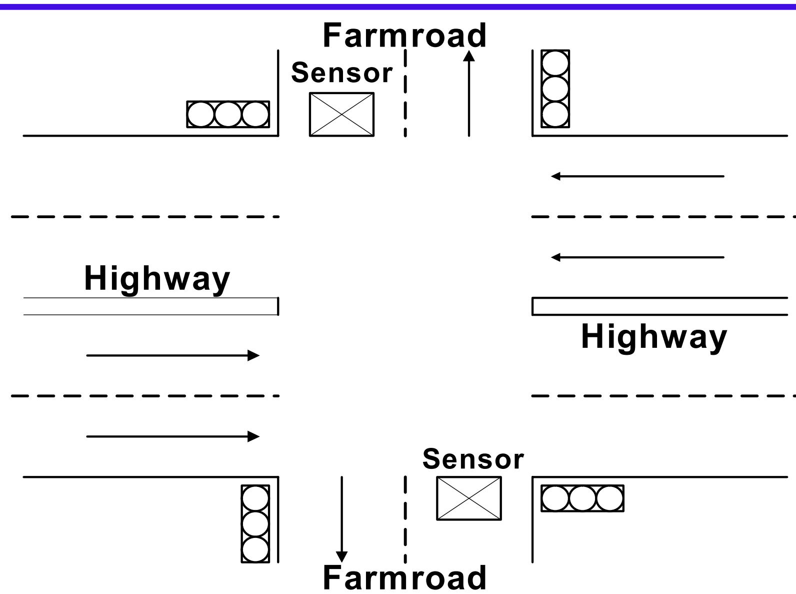


Schematic Entry Era

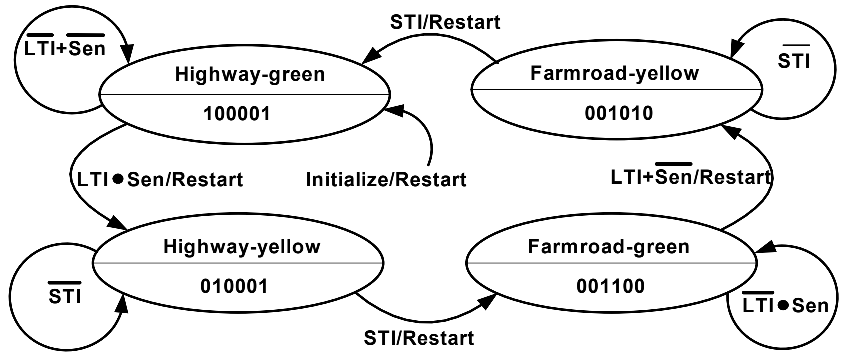
Given:

- **Gate-level schematic entry editor**
- **Gate-level simulator (we haven't talked about this)**
- **Gate level static-timing analyzer**
- **Netlist → Layout flow**
- **We can (and did) build large-scale integrated (35,000 gate) circuits**
- **EDA vendors provided front-end tools and ASIC vendor (e.g. LSI Logic) provided back-end flow**
- **But … It may be much more natural, and productive, to describe complex control logic by Boolean equations than by a schematic netlist of gates**

For example: traffic light controller



As a State transition diagram



Boolean Logic Equations

$$J_A = \overline{A} \bullet \left(Sen \bullet LTI + \overline{Sen} + LTI\right)$$

$$K_A = J_B = K_B = A \bullet STI$$

$$Restart = \overline{A} \bullet Sen \bullet LTI + \overline{A} \bullet B \bullet Sen + A \bullet STI$$

$$CHWG = \overline{A} \bullet \overline{B}$$

$$CHWY = A \bullet \overline{B}$$

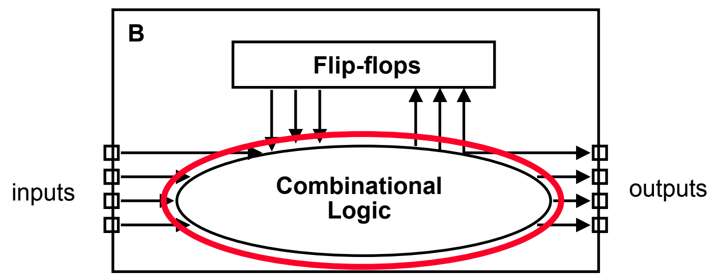
$$CHWR = B$$

$$CFRG = \overline{A} \bullet B$$

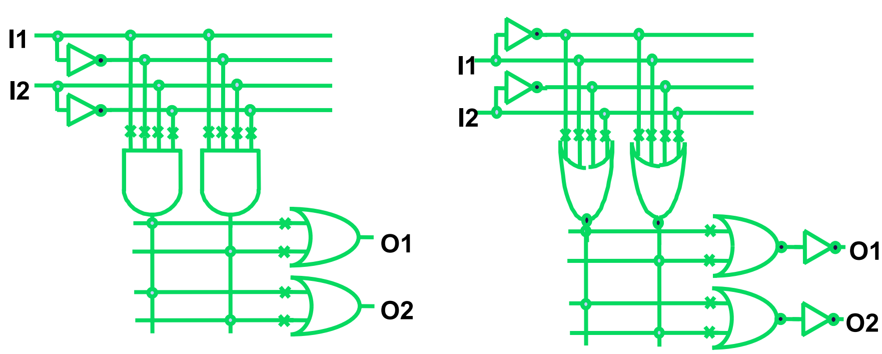
$$CFRY = A \bullet B$$

$$CFRR = \overline{B}$$

Synthesize Logic to Implement equations

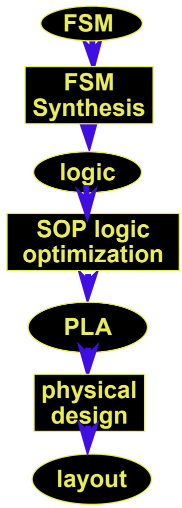


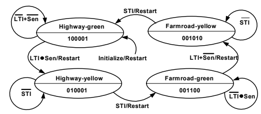
Physically Implement: AND-OR and NOR-NOR PLAs



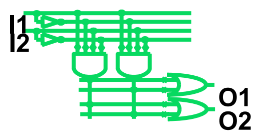
Logic increases with the number of product terms

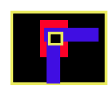
Early “Synthesis” Flow





$F1 = B + D + A\,C + A\,C$





Key Technology: SOP Logic Minimization

Can realize an arbitrary logic function in sum-of-products or two-level form

$$F1 = \overline{A}\,\overline{B} + \overline{A}\,B\,D + \overline{A}\,B\,\overline{C}\,\overline{D} + A\,B\,C\,\overline{D} + A\,\overline{B} + A\,B\,D$$

$$F1 = \overline{B} + D + \overline{A}\,\overline{C} + A\,C$$

Of great interest to find a minimum sum-of-products representation

Definitions - 1

Basic definitions:

don't care – aka "X"

Let $B = \{0, 1\}$ and $Y = \{0, 1, 2\}$

Input variables: $X_1, X_2 \dots X_n$

Output variables: $Y_1, Y_2 \dots Y_m$

A logic function ff (or Boolean function, switching function) in n inputs and m outputs is the map

$$ff: B^n \longrightarrow Y^m$$

11

Definitions - 2

If $b \in B^n$ is mapped to a 2 then function is incompletely specified, else completely specified

For each output we define:

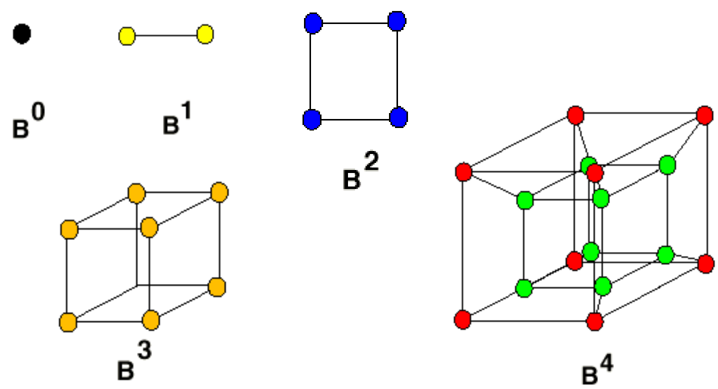
$\text{ON-SET}_i \subseteq B^n$, the set of all input values for which $ff_i(x) = 1$

$\text{OFF-SET}_i \subseteq B^n$, the set of all input values for which $ff_i(x) = 0$

$\text{DC-SET}_i \subseteq B^n$, the set of all input values for which $ff_i(x) = 2$

12

The Boolean n-Cube, B^n

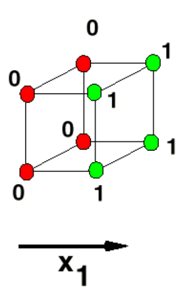


- $\mathcal{B} = \{0, 1\}$
- $\mathcal{B}^2 = \{0, 1\} \times \{0, 1\} = \{00, 01, 10, 11\}$

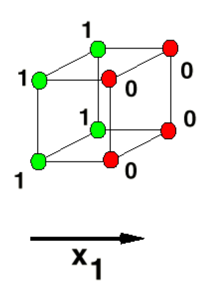
Literals

A literal is a variable or its negation $y, \overline{y}$

It represents a **logic function**



$f = x_1$



$f = \overline{x}_1$

Green – ON-set
Red – OFF-set

Boolean Formulas

Boolean functions can be **represented** by **formulas** defined as catenations of

- parentheses - (,)
- literals - $x, y, z, \overline{x}, \overline{y}, \overline{z}$
- Boolean operators - $+$ (OR), $\times$ (AND)
- complementation - e.g. $\overline{x+y}$

Examples:

$$\begin{aligned} f &= x_1 \times \overline{x}_2 + \overline{x}_1 \times x_2 \\ &= (x_1 + x_2) \times (\overline{x}_1 + \overline{x}_2) \\ h &= a + b \times c \\ &= \overline{\overline{a} \times (\overline{b} + \overline{c})} \end{aligned}$$

We will usually replace $\times$ by catenation, e.g. $a \times b \rightarrow ab$.

Example Boolean Function

EXAMPLE: **Truth table form of an incompletely specified function**

ff: $B^3 \longrightarrow Y^2$

X_1	X_2	X_3	Y_1	Y_2
0	0	0	1	1
0	0	1	1	0
0	1	0	0	1
0	1	1	0	1
1	0	0	1	0
1	0	1	1	2
1	1	0	1	1
1	1	1	2	1

Y_1: ON-SET$_1$ = {000, 001, 100, 101, 110}
OFF-SET$_1$ = {010, 011}
DC-SET$_1$ = {111}

Cube Representation

$$F1 = \overline{A}\,\overline{B} + \overline{A}\,B\,D + \overline{A}\,B\,\overline{C}\,\overline{D} + A\,B\,C\,\overline{D} + A\,\overline{B} + A\,B\,D$$

Inputs	Outputs
0 0 - -	1
0 1 - 1	1
0 1 0 0	1
1 1 1 0	1
1 0 - -	1
1 1 - 1	1

$$F1 = \overline{B} + D + \overline{A}\,\overline{C} + A\,C$$

minimum representation

- 0 - -	1
- - - 1	1
0 - 0 -	1
1 - 1 -	1

Operations on Logic Functions

(1) Complement: $f \longrightarrow \overline{f}$
interchange ON and OFF-SETS

(2) Product (or intersection or logical AND)
$h = f \bullet g$ or $h = f \cap g$

(3) Sum (or union or logical OR):
$h = f + g$ or $h = f \cup g$

(4) Difference $h = f - g = f \cap \overline{g}$

Prime Implicants

A cube p is an implicant of f if it does not intersect the OFF-SET of f

$p \subseteq f_{ON} \cup f_{DC}$ (or $p \cap f_{OFF} = 0$)

A prime implicant of f is an implicant p such that

(1) No other implicant q is such that $q \supset p$ in the sense that q covers all vertices of p
(2) $f_{DC} \not\supset p$

A minterm is a fully specified implicant e.g., 011, 111 (not 01-)

Examples of Implicants/Primes

X_1	X_2	X_3	Y_1
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	2

000, 00- are implicants, but not primes (-0-)

1-1

0-0

Prime and Irredundant Covers

A cover is a set of cubes C such that

$C \supseteq f_{ON}$ and

$C \subseteq f_{ON} \cup f_{DC}$

All of the ON-set is covered by C

C is contained in the ON-set and Don't Care Set

A prime cover is a cover whose cubes are all prime implicants

An irredundant cover is a cover C such that removing any cube from C results in a set of cubes that no longer covers the function

Minimum covers

A minimum cover is a cover of minimum cardinality

Theorem: A minimum cover can always be found by restricting the search to prime and irredundant covers.

Given any minimum cover C

(a) if redundant, not minimum

(b) if any cube q is not prime, replace q with prime $p \supset q$ and it is a minimum prime cover

Example Covers

X_1	X_2	X_3	Y_1
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	2

0 0 -
1 0 - is a cover. Is it prime?
1 1 - Is it irredundant?

What is a minimum prime and irredundant cover for the function?

Example Covers

X_1	X_2	X_3	Y_1
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	2

0 0 -
1 0 - is a cover. Is it prime?
1 1 - Is it irredundant?

\- 0 -
1 1 - is a cover. Is it prime?
Is it irredundant?
Is it minimum?

What is a minimum prime and irredundant cover for the function?

The Quine - McCluskey Method

Step 1: List all minterms in ON-SET and DC-SET

Step 2: Use a prescribed sequence of steps to find all the prime implicants of the function

Step 3: Construct the prime implicant table

Step 4: Find a minimum set of prime implicants that cover all the minterms

Example

0	0000	0,8	-000 Ⓔ	8,9,10,11	10-- Ⓑ
5	0101	5,7	01-1 Ⓓ	10,11,14,15	1-1- Ⓐ
7	0111	7,15	-111 Ⓒ		
8	1000	8,9	100-		
9	1001	8,10	10-0		
10	1010	9,11	10-1		
11	1011	10,11	101-		
14	1110	10,14	1-10		
15	1111	11,15	1-11		
		14,15	111-		

Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ are prime implicants

Prime Implicant Table

Minterms (ON-SET only)

	A	B	C	D	E
0					X
5				X	
7			X	X	
8		X			X
9		X			
10	X	X			
11	X	X			
14	X				
15	X		X		

X's indicate minterms covered by PIs

Essential Prime Implicants

	A	B	C	D	E
0					X
5				X	
7			X	X	
8		X			X
9		X			
10	X	X			
11	X	X			
14	X				
15	X		X		

Row with a single X identifies an essential prime implicant (EPI)

Essential PI's E, D, B, A $\Rightarrow$ Form minimum cover

Dominating Rows

In general EPIs do not form a cover

At Step 4, we need to select PIs to add to the EPIs so as to form a minimum cover

	A	B	C	D	F	G
1		X	X			
8	X		X			
9	X	X	X			
24	X					X
25	X	X			X	X
27				X	X	

Row 9 dominates 8
Row 25 dominates 24

Can remove 8 since covering 9 implies covering of 8

Dominating Columns

	A	B	C	D	F	G
1		X	X			
8	X		X			
9	X	X	X			
24	X					X
25	X	X			X	X
27				X	X	

F dominates D

Can remove D since F covers all minterms D covers

Can this happen in the original table?

May happen after removal of EPIs

Step 4 Issues

Removal of dominating columns or dominated rows may introduce columns with single X's.

– Need to iterate

A cover may still not be formed after all essential elements and dominance relations have been removed

– Need to branch over possible solutions

Recursive Branching (Step 4)

(a) Select EPIs, remove dominated columns and dominating rows iteratively till table does not change

(b) If the size of the selected set (+ lower bound) exceeds or equals best solution so far, return from this level of recursion. If no elements left to be covered, declare selected set as the best solution recorded.

(c) Select (heuristically) a branching column.

Recursive Branching (Step 4) - 2

(d) Given the selected column, recur

- On the sub-table resulting from deleting the column and all rows covered by this column. Add this column to the selected set.
- On the sub-table resulting from deleting the column without adding it to the selected set.

Example - a1

	A	B	C	D	E	F	G	H
0	X							X
1	X	X						
5		X	X					
7			X	X				
8							X	X
10						X	X	
14					X	X		
15				X	X			

No essential primes, dominated rows or columns.

Select prime A

Example - a2

	B	C	D	E	F	G	H
5	X	X					
7		X	X				
8						X	X
10					X	X	
14				X	X		
15			X	X			

Selected set
= { A **}**

B **is dominated by** C

H **is dominated by** G

Remove B, H

Example - a3

	C	D	E	F	G
5	X				
7	X	X			
8					X
10				X	X
14			X	X	
15		X	X		

C, G **essential <u>to</u>**
<u>this table</u>

Selected set
= {A, C, G**}**

	D	E	F
14		X	X
15	X	X	

Selected set
= {A, C, G, E**}**

Example - b1

	B	C	D	E	F	G	H
0							X
1	X						
5	X	X					
7		X	X				
8						X	X
10					X	X	
14				X	X		
15			X	X			

Selected set = { }

Essential primes
<u>in this table</u> are B, H

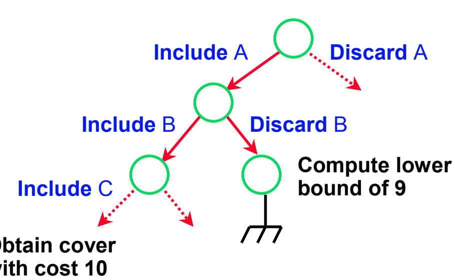
Selected set = {B, H}

	C	D	E	F	G
7	X	X			
10				X	X
14			X	X	
15		X	X		

Selected set
= {B, H, D, F}

Espresso-Exact (1987)

Efficient lower bounding at Step 4(b) to terminate unprofitable searches high in the recursion



Size of selected set + Lower bound equals or exceeds best solution already known, quit level of recursion

Lower Bounding

	A	B	C	D	E	F
0	X				X	
1	X		X			X
4					X	
6		X	X			
8		X				
10				X		X
12				X		X

Lower bound: Maximal independent set of rows all of which are pairwise disjoint

Maximal independent set = {1, 4, 8} or {0, 6, 10}

Need to select at least one PI/column to cover each row.

NOTE: **Finding maximum independent set is itself worst-case exponential**

Complexity of Q-M based Methods

(1) There exist functions for which the number of prime implicants is $O(3^n)$ (n is number of inputs)

(2) Given a PI table, recursive branching could require $O(2^m)$ time (m is the number of PIs)

Current logic minimizers able to find exact solutions for functions with 20-25 input variables

⇒ Need heuristic methods for larger functions

Heuristic Logic Minimization

Presently, there appears to be a limit of ~20-25 input variables in problems that can be handled by exact minimizers

Easy for complex control logic to exceed 20- 25 input variables

HISTORY

50's	Karnaugh Map	≤ 5 variables
60's	Q-M method	< 10 variables
70's	Starner, Dietmeyer	< 15 variables
1974	MINI	heuristic
1980-84	ESPRESSO	approaches
1986	McBoole	< 25 variables
1987	ESPRESSO-EXACT	< 25 variables

Also, Multiple Output Functions

Truth table is AND-OR representation

AND			OR	
a	b	c	f	g
0	1	–	1	0
0	1	1	0	1
1	0	1	0	1

What does vector 0 1 1 produce?

ON-SET of f = {0 1 –, 0 1 1} = {0 1 –}

ON-SET of g = {0 1 1, 1 0 1}

Multiple-Output Function Primes

Same definition as in single-output case

- Cube with most minterms that will intersect OFF-SET if you add any more minterms to them

f g		CUBE	TYPE
0 0 0 0	1 0	0 0 0 0 1 0	
0 0 0 1	1 0	0 0 0 – 1 0	
1 0 0 1	1 0	1 0 0 1 1 0	
0 0 0 0	0 1	1 0 0 1 1 1	
0 0 1 0	0 1	0 0 0 – 1 1	
1 0 0 1	0 1		

MINI

S.J. Hong, R.G. Cain, D.L. Ostapko - 1974

Final solution is obtained from initial solution by iterative improvement rather than by generating and covering prime implicants

Three basic modifications are performed

- Reduction of implicants while maintaining coverage
- Reshaping implicants in pairs
- Expansion of implicants (and removal of covered implicants)

MINI Algorithm

MINI (F, DC) {

F is ON-SET
DC is Don't Care Set

1. $\overline{F} = U - F$ — U is universe cube

2. (Cover) f = *Expand* $\overline{f}$ against F
   p = Compute solution size

3. f = *Reduce* each cube of f
   against other cubes of F $\vee$ DC

4. *Reshape* f

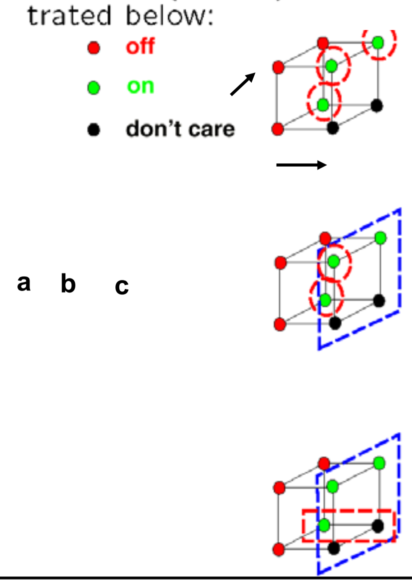
5. f = *Expand* f against $\overline{F}$
   n = compute solution size

6. If n < p go to 3, else, exit

}

Example: Expansion

Consider $\mathcal{F}(a,b,c) = (f,d,r)$, where $f = \{\overline{a}b\overline{c}, a\overline{b}c, abc\}$ and $d = \{a\overline{b}\overline{c}, ab\overline{c}\}$, and the sequence of covers illustrated below:



$F^1 = abc + a\overline{b}c + \overline{a}b\overline{c}$

EXPAND abc → a

$F^2 = a + a\overline{b}c + \overline{a}b\overline{c}$

$a\overline{b}c$ is redundant
a is prime

$F^3 = a + \overline{a}b\overline{c}$

EXPAND $\overline{a}b\overline{c}$ → $b\overline{c}$

$F^4 = a + b\overline{c}$

Expansion Example

Step 2 in MINI:

Expand f against $\overline{F}$

f		$f_{expanded}$		$\overline{F}$	
1 0 0 1	0 1	1 0 0 1	0 1	1 1 1 0	1 0
0 1 1 0	1 0	0 – 1 0	1 0	0 1 0 1	1 0
1 1 0 1	1 0	1 1 0 1	1 0	0 0 1 0	0 1
1 0 0 0	1 0	1 0 – 0	1 0	0 0 0 1	0 1
1 0 1 0	0 1	1 0 1 0	0 1	0 0 0 0	1 0
– 1 0 0	1 0	– 1 0 0	1 0	– 1 0 1	0 1
– 1 1 1	0 1	– – 1 1	1 1	– 0 0 0	0 1
1 0 1 1	0 1				
– 1 1 1	1 0				
– 0 1 0	1 0				
– 1 – 0	0 1	– 1 – 0	0 1		
– 0 – 1	1 0	– 0 – 1	1 0		

Order small cubes first

Reduction

Reduce the size (in the sense of the number of minterms/vertices that it covers) of cubes in f without affecting coverage

The smaller the size of the cube, the more likely it will be covered by an expanded cube

Reduction Examples

Reducing covers:

f	1 – –	1	$f_{reduced}$	1 0 0	1
	– 1 –	1		– 1 –	1
	– – 1	1		– – 1	1

(– 0 1 1   1 0)

1 0 0 1	0 1	– – 1 1	1 1	– – 1 1	1 1
0 – 1 0	1 0	– 0 – 1	1 0	– 0 0 1	1 0
1 1 0 1	1 0	– 1 – 0	0 1	– 1 – 0	0 1
1 0 – 0	1 0	1 0 – 0	1 0	1 0 – 0	1 0
1 0 1 0	0 1	– 1 0 0	1 0	– 1 0 0	1 0
– 1 0 0	1 0	0 – 1 0	1 0	0 – 1 0	1 0
– – 1 1	1 1	1 1 0 1	1 0	1 1 0 1	1 0
– 1 – 0	0 1	1 0 1 0	0 1	1 0 1 0	0 1
– 0 – 1	1 0	1 0 0 1	0 1	1 0 0 1	0 1

Larger cubes first

Reshaping

Attempt to change the shape of the cubes without changing coverage or number

Reshaping transforms a pair of cubes into another pair such that coverage is unaffected (perturbs solution so next expand does things differently)

Reshaping Example

f

–	–	1	1	1	1
–	0	0	1	1	0
–	1	–	0	0	1
1	0	–	0	1	0
–	1	0	0	1	0
0	–	1	0	1	0
1	1	0	1	1	0
1	0	1	0	0	1
1	0	0	1	0	1

f_{rdered}

–	–	1	1	1	1
–	1	–	0	0	1
1	0	–	0	1	0
–	0	0	1	1	0
–	1	0	0	1	0
0	–	1	0	1	0
1	1	0	1	1	0
1	0	1	0	0	1
1	0	0	1	0	1

1	–	–	1	1	1	1
2	–	1	–	0	0	1
3	1	0	–	0	1	0
4	–	0	0	1	1	0
5	–	1	0	0	1	0
6	0	–	1	0	1	0
7	1	1	0	1	1	0
8	1	0	1	0	0	1
9	1	0	0	1	0	1

$f_{reshaped}$

1	–	–	1	1	1	1
(2,5)	–	1	1	0	0	1
	–	1	0	0	1	1
(3,8)	1	0	0	0	1	0
	1	0	1	0	1	1
(4,9)	0	0	0	1	1	0
	1	0	0	1	1	1
6	0	–	1	0	1	0
7	1	1	0	1	1	0

A Complete Example

initial f

cd \ ab	f: 00	f: 01	f: 11	f: 10	g: 00	g: 01	g: 11	g: 10	
00		9	9	10		5	5		00
01	4		3	4				2	01
11	1	1	1	1	1	1	1	1	11
10	8	7		8		5	5	6	10

	a	b	c	d	f	g								
1	–	–	1	1	1	1		–	–	1	1	1	1	
2	1	0	0	1	0	1		1	0	0	1	0	1	
3	1	1	0	1	1	0	expand	1	1	0	1	1	0	
4	–	0	0	1	1	0	→	–	0	–	1	1	0	
5	–	1	–	0	0	1		–	1	–	0	0	1	
6	1	0	1	0	0	1		1	0	1	0	0	1	
7	0	1	1	0	1	0	→	0	–	1	0	1	0	←
8	–	0	1	0	1	0								
9	–	1	0	0	1	0		–	1	0	0	1	0	
10	1	0	0	0	1	0	→	1	0	–	0	1	0	←

Example - 2

expanded f

cd \ ab	00	01	11	10	00	01	11	10
00		9	9	10		5	5	
01	4		3	4				2
11	1,4	1	1	1,4	1	1	1	1
10	7	7		10		5	5	6

reduce →

	a b c d	f g	a b c d	f g
1	– – 1 1	1 1	– – 1 1	1 1
2	1 0 0 1	0 1	1 0 0 1	0 1
3	1 1 0 1	1 0	1 1 0 1	1 0
4	– 0 – 1	1 0	– 0 0 1	1 0
5	– 1 – 0	0 1	– 1 – 0	0 1
6	1 0 1 0	0 1	1 0 1 0	0 1
7	0 – 1 0	1 0	0 – 1 0	0 1
9	– 1 0 0	1 0	– 1 0 0	1 0
10	1 0 – 0	1 0	1 0 – 0	1 0

Example - 3

reduced f

cd \ ab	00	01	11	10	00	01	11	10
00		9	9	10		5	5	
01	4		3	4				2
11	1	1	1	1	1	1	1	1
10	7	7		10		5	5	6

reshape

	a b c d	f g	a b c d	f g	
1	– – 1 1	1 1	– – 1 1	1 1	
2	1 0 0 1	0 1	1 0 0 1	1 1	2,4
3	1 1 0 1	1 0	1 1 0 1	1 0	
4	– 0 0 1	1 0	0 0 0 1	1 0	2,4
5	– 1 – 0	0 1	– 1 1 0	0 1	5,9
6	1 0 1 0	0 1	1 0 1 0	1 1	6,10
7	0 – 1 0	1 0	0 – 1 0	1 0	
9	– 1 0 0	1 0	– 1 0 0	1 1	5,9
10	1 0 – 0	1 0	1 0 0 0	1 0	6,10

Example - 4

cd \ ab	00	01	11	10	00	01	11	10	
00		9	9	10		9	9		00
01	4		3	2				2	01
11	1	1	1	1	1	1	1	1	11
10	7	7		6		5	5	6	10

reshaped f

	a b c d	f g			
1	– – 1 1	1 1	expand	– – 1 1	1 1
2	1 0 0 1	0 1	→	1 0 – 1	1 1
3	1 1 0 1	1 0	→	1 – 0 –	1 0 ←
4	0 0 0 1	1 0	→	– 0 – 1	1 0
5	– 1 1 0	0 1	→	– 1 – 0	0 1
6	1 0 1 0	1 1	→	1 0 1 –	1 1
7	0 – 1 0	1 0		0 – 1 0	1 0
9	– 1 0 0	1 1		– 1 0 0	1 1
10	1 0 0 0	1 0			

Example - 5

cd \ ab	00	01	11	10	00	01	11	10	
00		9	3,9	3		5,9	5,9		00
01	4		3	2,3,4				2	01
11	1,4,7	1,7	1	1,2,4,6	1	1	1	1,2,6	11
10	7	7		6		5	5	6	10

final expanded f

final F

	a b c d	f g
1	– – 1 1	1 1
2	1 0 – 1	1 1
3	1 – 0 –	1 0
4	– 0 – 1	1 0
5	– 1 – 0	0 1
6	1 0 1 –	1 1
7	0 – 1 0	1 0
9	– 1 0 0	1 1

Summary of 2-level

2-level optimization is very effective and mature. Expresso (developed at Berkeley) is the "last word" on the subject

2-level optimization is directly useful for PLA's/PLD's – these were widely used to implement complex control logic in the early 80's – they are rarely used these days

2-level optimization forms the theoretical foundation for multilevel logic optimization

2-level optimization is useful as a subroutine in multilevel optimization