

CAD for VLSI

Two-level Logic Optimization 1

Outline

- Fundamentals of logic synthesis
 - Mathematical formulation
 - Definition of the problems
 - Two-level logic optimization
 - Motivation
 - Models
 - Exact algorithms for logic optimization
 - Boolean Relations
 - Motivation of using relations
 - Optimization of realization of Boolean relation
 - Comparisons to two-level optimization
-

Combinational Logic Design

Background

- Boolean Algebra
 - Quintuple $(B, +, \cdot, 0, 1)$
 - Binary Boolean algebra $B = \{ 0, 1 \}$
- Boolean function
 - Single output $f : B^n \rightarrow B$
 - Multiple output $f : B^n \rightarrow B^m$
 - Incompletely-specified:
 - *Don't care symbol:* *
 - $f : B^n \rightarrow \{ 0, 1, * \}^m$

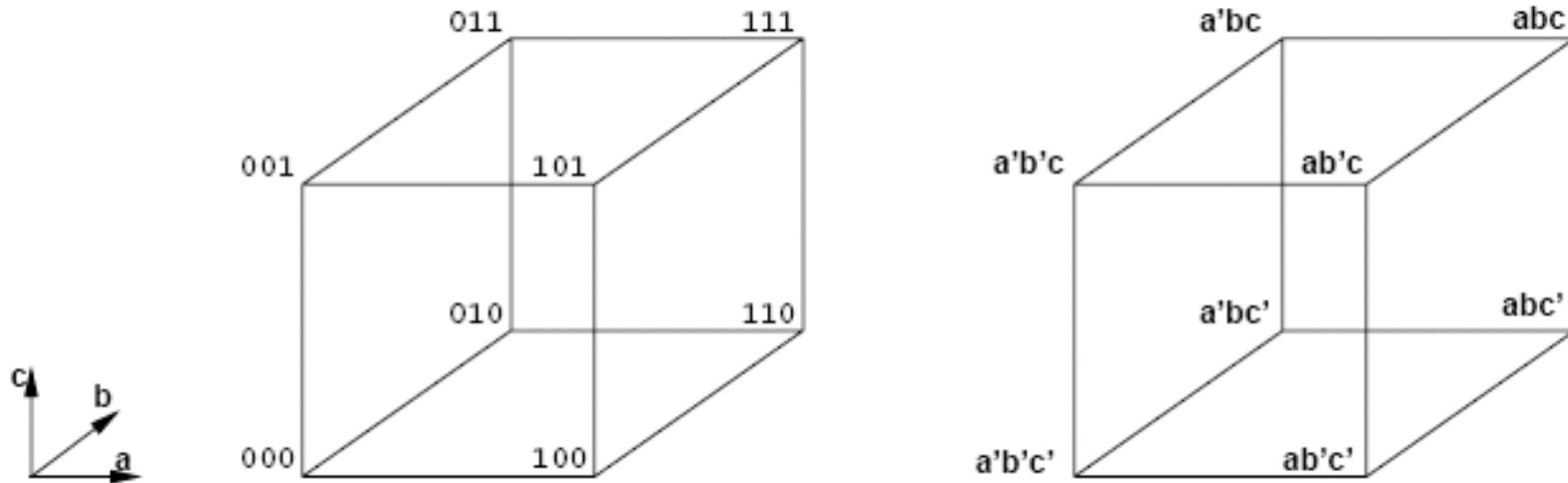
The *don't care* Conditions

- We do not care about the value of a function
- Related to the environment
 - Input patterns that never occur
 - Input patterns such that some output is never observed
- Very important for synthesis and optimization

Definitions

- Scalar function:
 - ON-set
 - Subset of the domain such that f is true
 - OFF-set
 - Subset of the domain such that f is false
 - DC-set
 - Subset of the domain such that f is a *don't care*
- Multiple-output function:
 - ON, OFF, DC-sets defined for each component

Cubical Representation



Definitions

- Boolean variables
- Boolean literals
 - Variables and their complement
- Product or cube
 - Product of literals
- Implicant
 - Product implying a value of the function (usually 1)
 - Hypercube in the Boolean space
- Minterm
 - Product of all input variables implying a value of the function (usually 1)
 - Vertex in the Boolean space

Tabular Representations

- Truth table
 - List of all minterms of a function
- Implicant table or cover
 - List of implicants sufficient to define a function
- Note
 - Implicant tables are smaller in size as compared to truth tables

Example of Truth Table

- $x = ab + a'c; \quad y = ab + bc + ac$

abc	xy
000	00
001	10
010	00
011	11
100	00
101	01
110	11
111	11

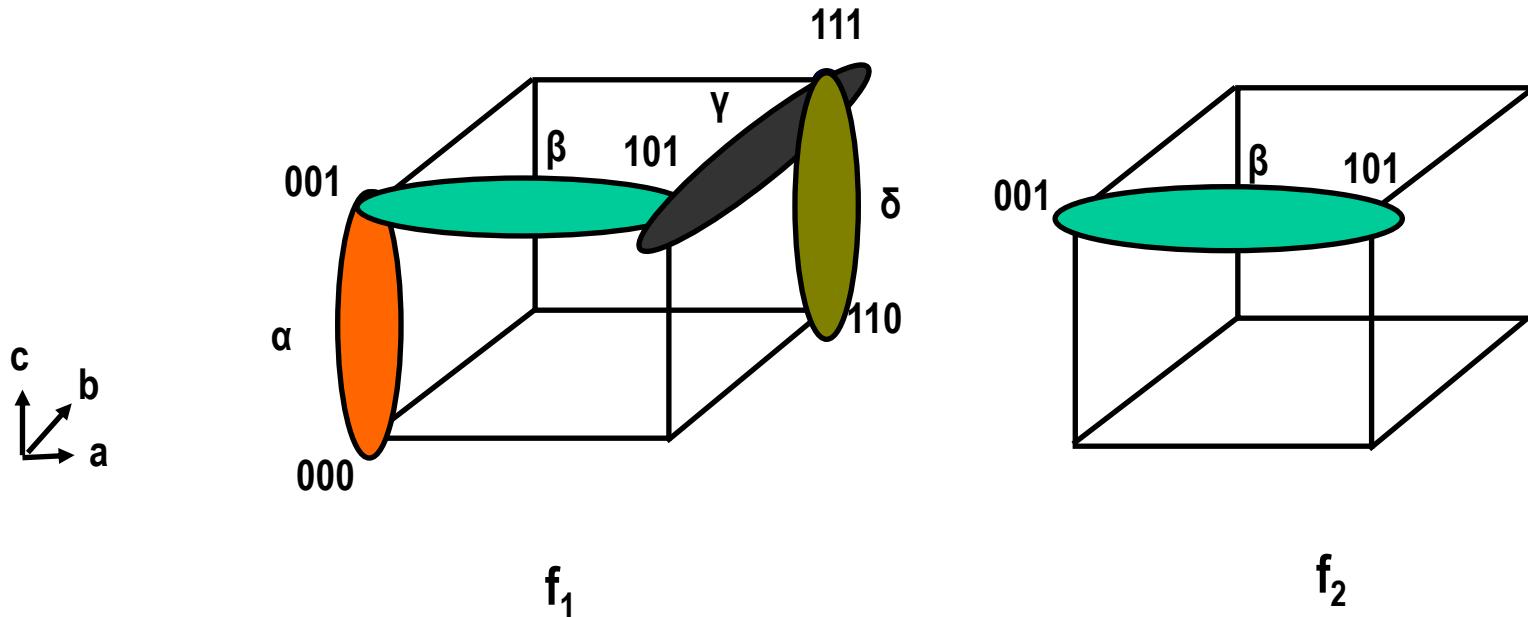
Example of Implicant Table

- $x = ab + a'c; \quad y = ab + bc + ac$

abc	xy
001	10
*11	11
101	01
11*	11

Cubical Representation of Minterms and Implicants

- $f_1 = a'b'c' + a'b'c + ab'c + abc + abc'$
- $f_2 = a'b'c + ab'c$



Representations

- Visual representations
 - Cubical notation
 - Karnaugh maps
- Computer-oriented representations
 - Matrices
 - Sparse
 - Various encoding
 - Binary-decision diagrams
 - Address sparsity and efficiency

Outline

- Fundamentals of logic synthesis
 - Mathematical formulation
 - Definition of the problems
- Two-level logic optimization
 - Motivation
 - Models
 - Exact algorithms for logic optimization
- Boolean Relations
 - Motivation of using relations
 - Optimization of realization of Boolean relation
 - Comparisons to two-level optimization

Two-level Logic Optimization

Motivation

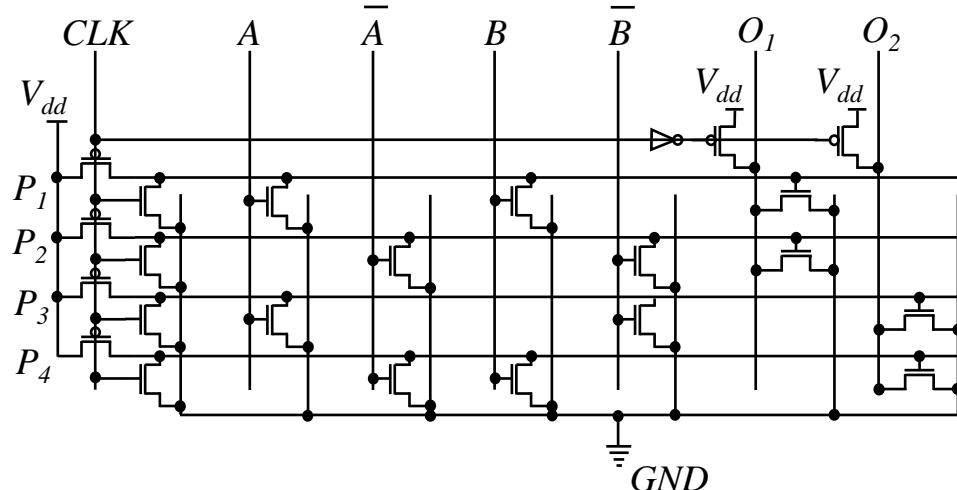
- Reduce size of the representation
- Direct implementation
 - PLAs reduce size and delay
- Other implementation styles
 - Reduce amount of information
 - Simplify local functions and connections

Programmable Logic Arrays

- Macro-cells with rectangular structure
 - Implement multi-output function
 - Layout generated by module generators
 - Fairly popular in the 1970s and 1980s
- Advantages
 - Simple, predictable timing
- Disadvantages
 - Less flexible than cell-based realization
 - Dynamic operation
- Open issue
 - Will PLA structures be useful with new nanotechnologies?
(ex: nanowires)

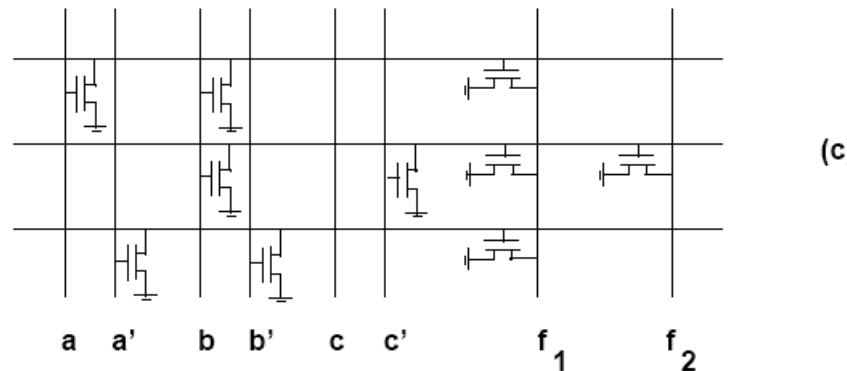
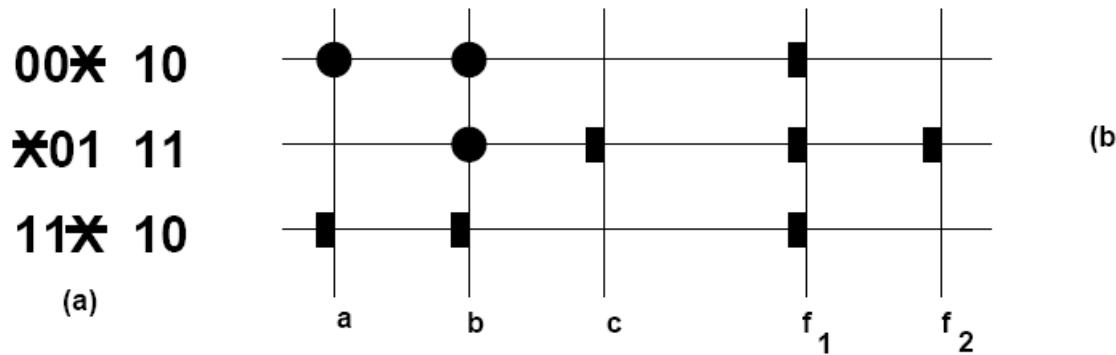
Dynamic PLA

- Sum-of-products/product-of-sums: regular layout
- Advantage – fast clock speed
 - Commonly used in control logic
- Drawback – narrow noise margin
 - Crosstalk problem



PLA Example

- $f_1' = a'b' + b'c + ab; f_2' = b'c$
- $f_1 = (a + b)(b + c')(a' + b'); f_2 = b + c'$



Two-level Minimization

□ Assumptions

- Primary goal is to reduce the number of implicants
- All implicants have the same cost
- Secondary goal is to reduce the number of literals

□ Rationale

- Implicants correspond to PLA rows
- Literals correspond to transistors

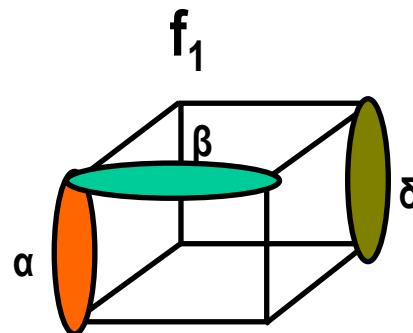
Definitions

- Minimum cover
 - Cover of a function with minimum number of implicants
 - Global optimum
- Minimal cover or irredundant cover
 - Cover of the function that is not a proper superset of another cover
 - No implicant can be dropped
 - Local optimum
- Minimal w.r.t. single-implicant containment
 - No implicant contained by another one
 - Weak local optimum

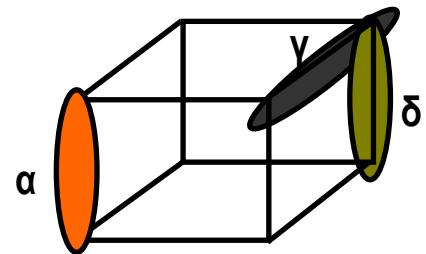
Example

□ $f_1 = a'b'c' + a'b'c + ab'c + abc + abc'; f_2 = a'b'c + ab'c$

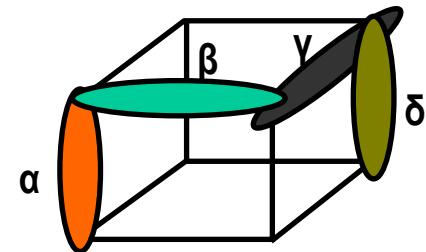
Minimum cover



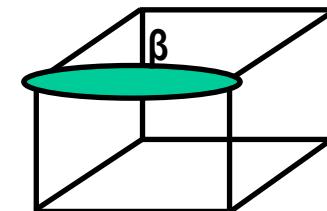
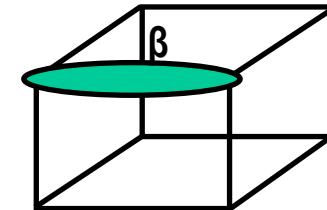
Irredundant cover



Minimal cover w.r.t. single implicant containment



f_2



Definitions

- Prime implicant
 - Implicant not contained by any other implicant
- Prime cover
 - Cover of prime implicants
- Essential prime implicant
 - There exist some minterms covered only by that prime implicant
 - MUST be included in the cover

Two-level Logic Minimization

- Exact methods
 - Compute minimum cover
 - Often difficult/impossible for large functions
 - Based on Quine-McCluskey method
- Heuristic methods
 - Compute minimal covers (possibly minimum)
 - Large variety of methods and programs
 - MINI, PRESTO, ESPRESSO

Exact Logic Minimization

- Quine's theorem:
 - There is a minimum cover that is prime
- Consequence
 - Search for minimum cover can be restricted to prime implicants
- Quine-McCluskey method
 - Compute prime implicants
 - Determine minimum cover

Prime Implicant Table

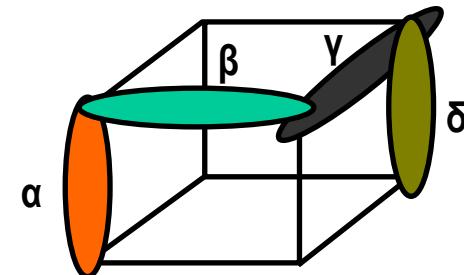
- ❑ Rows: minterms
- ❑ Columns: prime implicants
- ❑ Exponential size
 - 2^n minterms
 - Up to $3^n / n$ prime implicants (3 for 0/1/*)
- ❑ Remarks
 - Some functions have much fewer primes
 - Minterms can be grouped together
 - Implicit methods for implicant enumeration

Example

- $f = a'b'c' + a'b'c + ab'c + abc + abc'$

- Primes:

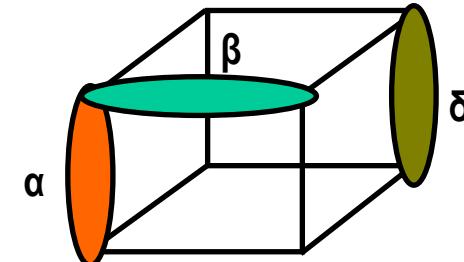
α	00*	1
β	*01	1
γ	1*1	1
δ	11*	1



Prime implicants of f

- Table:

	α	β	γ	δ
000	1	0	0	0
001	1	1	0	0
101	0	1	1	0
111	0	0	1	1
110	0	0	0	1



Minimum cover of f

Minimum Cover Early Methods

- Table reduction
 - Iteratively identify essentials
 - Save them in the cover
 - Remove covered minterms
 - Petrick's method
 - Write covering clauses in *POS* form
 - Multiply out *POS* form into *SOP* form
 - Select cube of minimum size
 - Remark
 - Multiplying out clauses has exponential cost
-

Example

- POS clauses

- $(\alpha) (\alpha + \beta) (\beta + \gamma) (\gamma + \delta) (\delta) = 1$

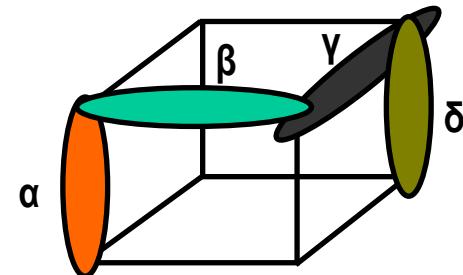
- SOP form:

- $\alpha\beta\delta + \alpha\gamma\delta = 1$

- Solutions:

- $\{ \alpha \beta \delta \}$

- $\{ \alpha \gamma \delta \}$



Matrix Representation

- View table as Boolean matrix: A
- Selection Boolean vector for primes: x
- ILP-based formulation to determine x such that
 - $A x \geq 1$
 - Select enough columns to cover all rows
- Minimize cardinality of x

	α	β	γ	δ
000	1	0	0	0
001	1	1	0	0
101	0	1	1	0
111	0	0	1	1
110	0	0	0	1

Example

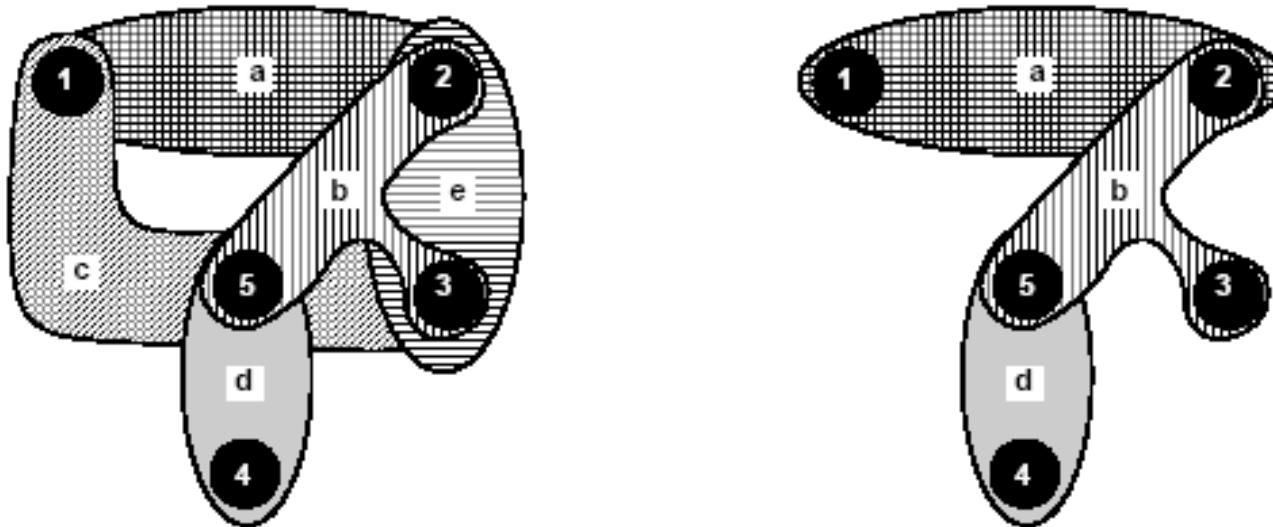
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

Covering Problem

- Set covering problem:
 - A set S (minterm set)
 - A collection C of subsets (implicant set)
 - Select fewest elements of C to cover S
- Computationally intractable problem
- Exact solution method
 - Branch and bound algorithm
- Several heuristic approximation methods

Example Edge-cover of a Hypergraph

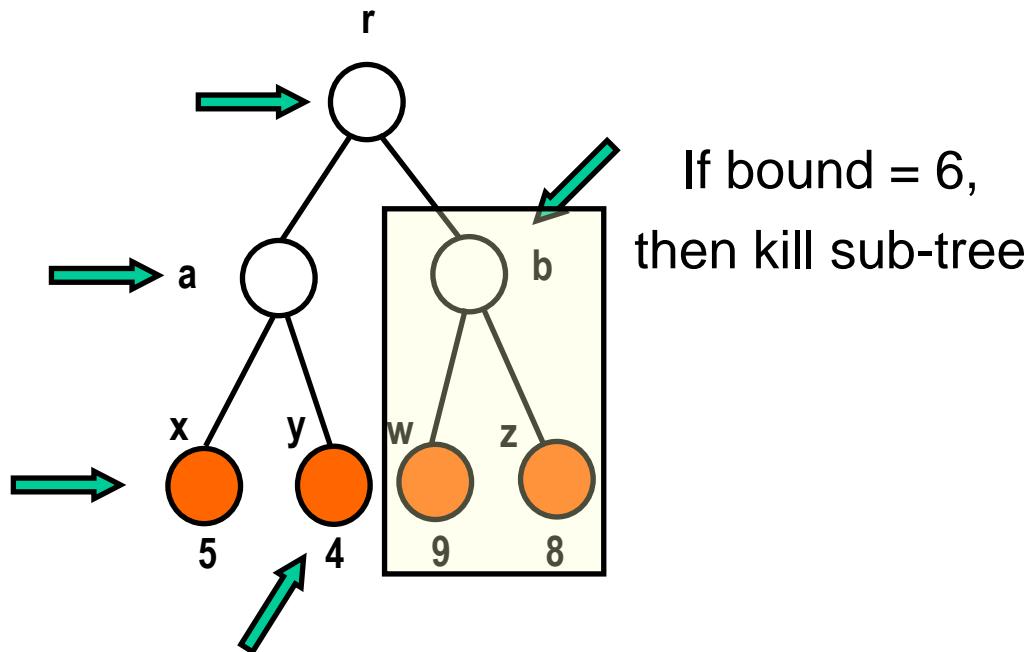
- Hypergraph is a generalization of a graph, which an edge (hyperedge) can join any number of vertices
- Each vertex represents the one minterm
- Each hyperedge represents one implicant



Branch and Bound Algorithm

- Tree search in the solution space
 - Potentially exponential
- Use bounding function:
 - If the lower bound on the solution cost that can be derived from a set of future choices exceeds the cost of the best solution seen so far, then kill the search
 - Bounding function should be fast to evaluate and accurate
- Good pruning may expedite the search

Branch and Bound Example



Branch and Bound for Logic Minimization Reduction Strategies

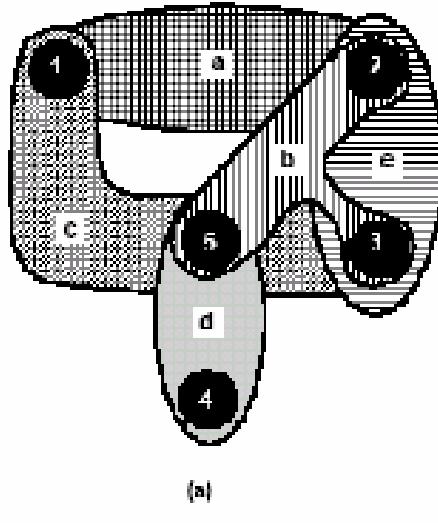
- Use matrix formulation of the problem
- Partitioning:
 - If A is block diagonal:
 - Solve covering problems for the corresponding blocks
- Essentials
 - Column incident to one (or more) rows with single 1
 - Select column
 - Remove covered row(s) from table

Branch and Bound for Logic Minimization Reduction Strategies

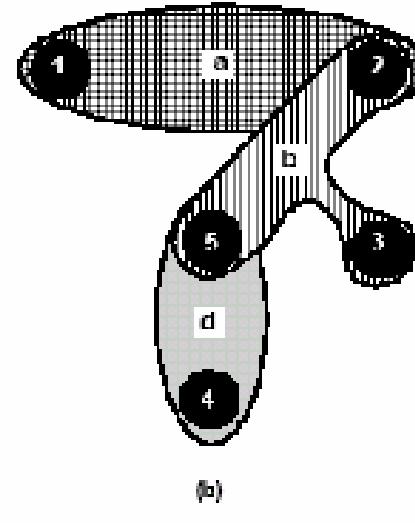
- Column (implicant) dominance:
 - If $a_{ki} \geq a_{kj}$ for all k
 - Remove column j (dominated)
 - Dominated implicant (j) has its minterms already covered by dominant implicant (i)

- Row (minterm) dominance:
 - If $a_{ik} \geq a_{jk}$ for all k
 - Remove row i (dominant)
 - When an implicant covers the dominated minterm, it also covers the dominant one

Example



(a)



(b)

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

Example

- ❑ Fifth column is dominated
- ❑ Fifth row is dominant
- ❑ Fourth column is essential
- ❑ Matrix after reductions

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

$$\mathbf{A}' = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

Branch and Bound Covering Algorithm

```
EXACT_COVER( $A, x, b$ ) {
```

 Reduce matrix A and update corresponding x ;

 if (current_estimate $\geq |b|$) return (b);

 if (A has no rows) return(x);

 select a branching column c ;

$x_c = 1$;

$\tilde{A} = A$ after deleting c and rows incident to it;

$x' = EXACT_COVER(\tilde{A}, x, b)$;

 if ($|x'| < |b|$)

$b = x'$;

$x_c = 0$;

$\tilde{A} = A$ after deleting c ;

$x' = EXACT_COVER(\tilde{A}, x, b)$;

 if ($|x'| < |b|$)

$b = x'$;

 return(b);

}

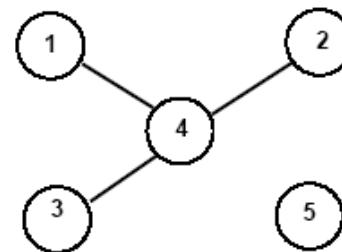
Bounding Function

- Estimate lower bound on covers that can be derived from current solution vector x
- The sum of the 1s in x , plus bound of cover for local A
 - Independent set of rows
 - No 1 in the same column
 - Require independent implicants to cover
 - Construct graph to show pairwise independence
 - Find clique number
 - Size of the largest clique
 - Approximation (lower) is acceptable

Example

- Row 4 independent from 1,2,3
- Clique number and bound is 2

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$



Example

$$A = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

- There are no independent rows
 - Clique number is 1 (one vertex)
 - Bound is $1+1=2$
 - Because of the essential already selected

Example Branching on the Cyclic Core

$$A = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

- Select first column
 - Recur with $\tilde{A} = [11]$
 - Delete one dominated column
 - Take other column (essential)
 - New cost is 3
- Exclude first column
 - Find another solution with cost equal to 3
 - Discard

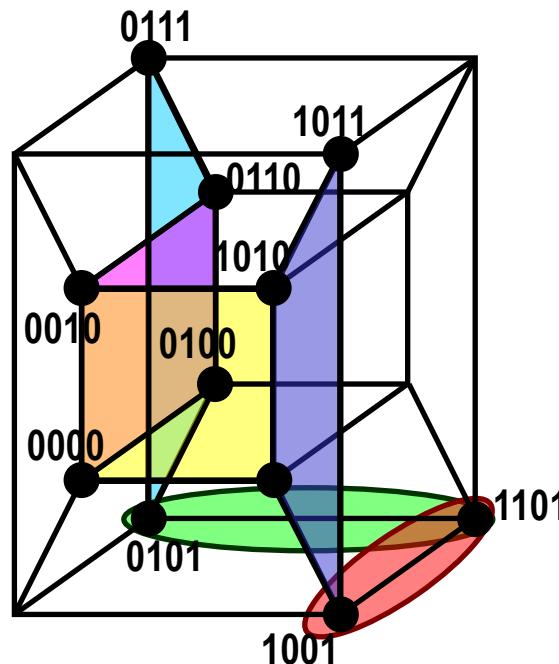
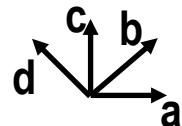
Espresso-exact

- Exact 2-level logic minimizer
- Exploits iterative reduction and branch and bound algorithm on cyclic core
- Compact implicant table
 - Rows represent groups of minterms covered by the same implicants
- Very efficient
 - Solves most benchmarks

Example

After removing the essentials

	α	β	ε	ζ
0000,0010	1	1	0	0
1101	0	0	1	1



α	0 * * 0	1
β	* 0 * 0	1
γ	0 1 * *	1
δ	1 0 * *	1
ε	1 * 0 1	1
ζ	* 1 0 1	1

Exact Two-level Minimization

- There are two major difficulties:
 - Storage of the implicant table
 - Solving the cyclic core
- Implicit representation of prime implicants
 - Methods based on binary decision diagrams
 - Avoid explicit tabulation
- Recent methods make 2-level optimization solve exactly almost all benchmarks
 - Heuristic optimization is just used to achieve solutions faster

Outline

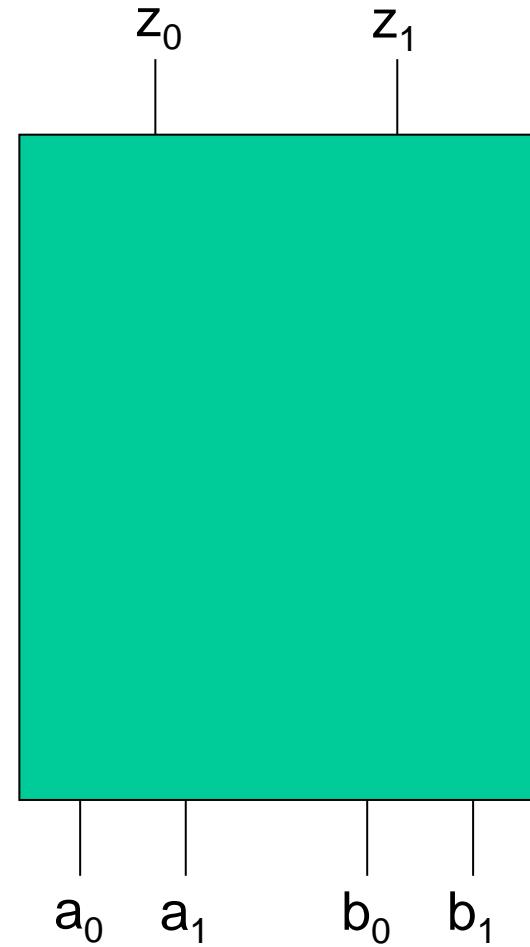
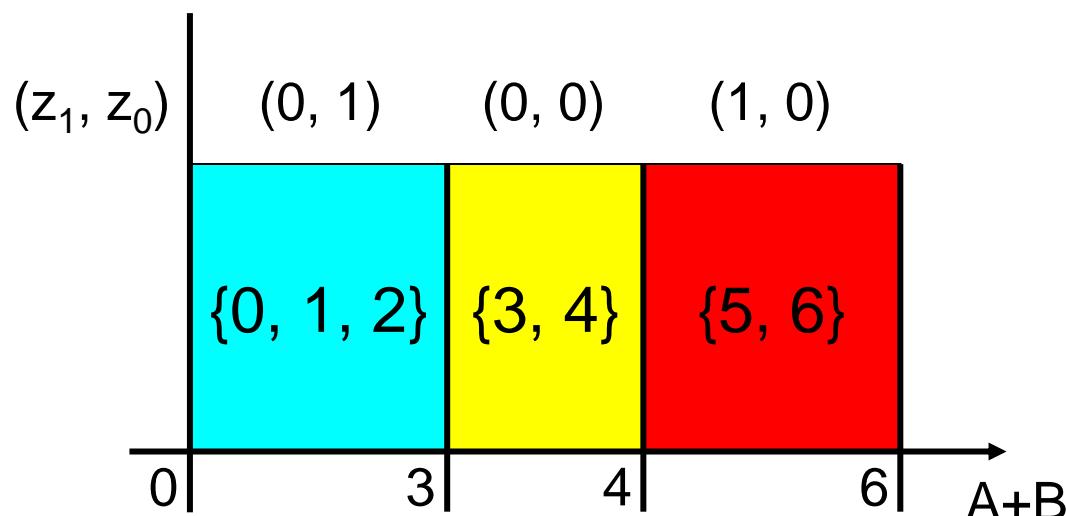
- Fundamentals of logic synthesis
 - Mathematical formulation
 - Definition of the problems
 - Two-level logic optimization
 - Motivation
 - Models
 - Exact algorithms for logic optimization
 - Boolean Relations
 - Motivation of using relations
 - Optimization of realization of Boolean relation
 - Comparisons to two-level optimization
-

Boolean Relations

- Generalization of Boolean functions
- More than one output pattern may correspond to an input pattern
 - Multiple-choice specifications
 - Model inner blocks of multi-level circuits
- Degrees of freedom in finding an implementation
 - More general than *don't care* conditions
- Problem
 - Given a Boolean relation, find a minimum cover of a compatible Boolean function that can implement the relation

Example

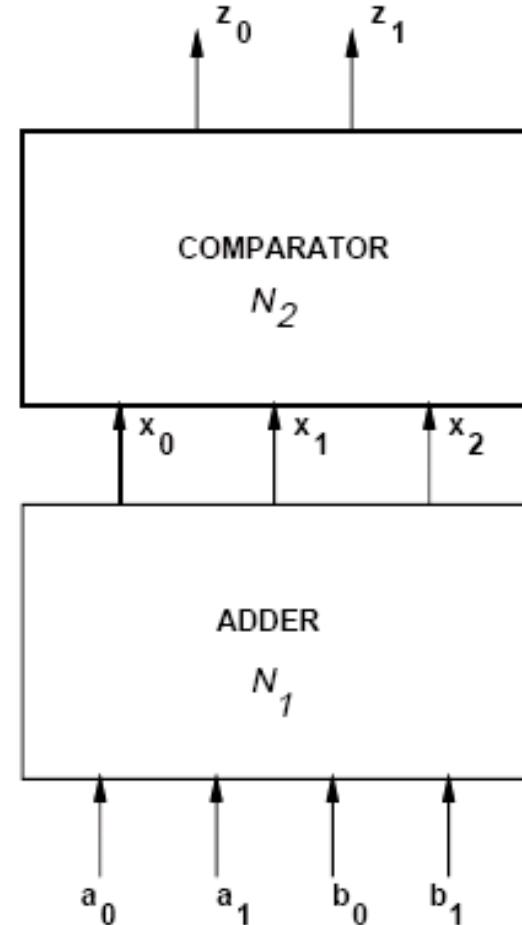
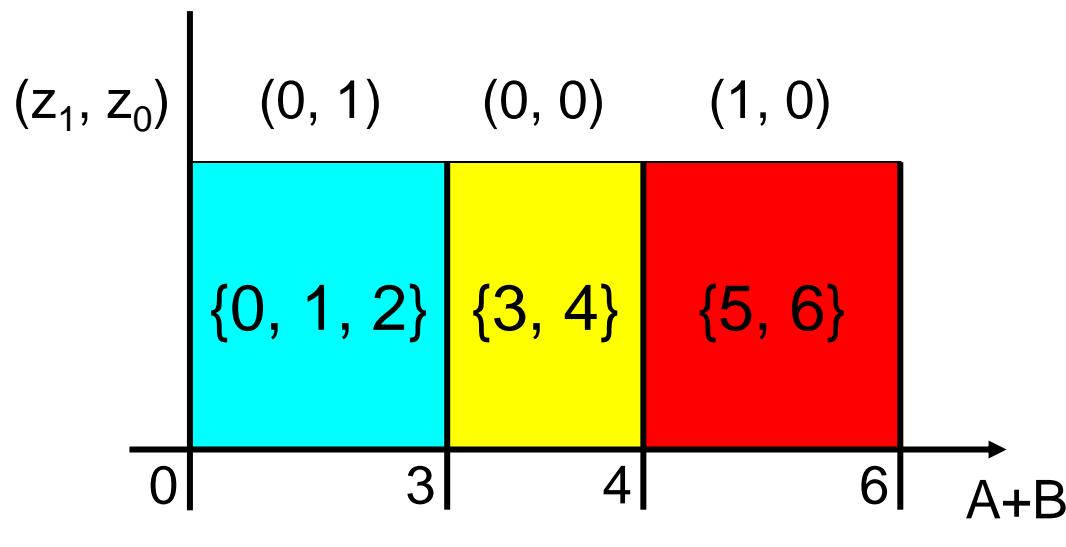
- Divide $A+B$ into 3 regions
 - <3
 - $[3, 4]$
 - >4



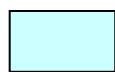
Example

□ Compare:

- $a + b > 4$?
- $a + b < 3$?



Example



adder + comparator

a_1	a_0	b_1	b_0	x	z
0	0	0	0	{ 000, 001, 010 }	
0	0	0	1	{ 000, 001, 010 }	
0	0	1	0	{ 000, 001, 010 }	
0	1	0	0	{ 000, 001, 010 }	
1	0	0	0	{ 000, 001, 010 }	
0	1	0	1	{ 000, 001, 010 }	
<hr/>					
0	0	1	1	{ 011, 100 }	
0	1	1	0	{ 011, 100 }	
1	0	0	1	{ 011, 100 }	
1	0	1	0	{ 011, 100 }	
1	1	0	0	{ 011, 100 }	
0	1	1	1	{ 011, 100 }	
1	1	0	1	{ 011, 100 }	
<hr/>					
1	0	1	1	{ 101, 110, 111 }	
1	1	1	0	{ 101, 110, 111 }	
1	1	1	1	{ 101, 110, 111 }	(1, 0)

Example

- Circuit is no longer an adder

a_1	a_0	b_1	b_0	$X = (x_2, x_1, x_0)$
0	*	1	*	010
1	*	0	*	010 x_1
1	*	1	*	100 x_2
*	*	*	1	001
*	1	*	*	001 x_0

$$x_0 = a_0 + b_0$$

$$x_1 = a'_1 b_1 + a_1 b'_1$$

$$x_2 = a_1 b_1$$

Example

adder + comparator

another implementation

$$x_0 = a_0 + b_0$$

$$x_1 = a'_1 b_1 + a_1 b'_1$$

$$x_2 = a_1 b_1$$

a_1	a_0	b_1	b_0	x	z
0	0	0	0	{ {000}, 001, 010 }	
0	0	0	1	{ 000, {001}, 010 }	
0	0	1	0	{ 000, 001, {010} }	
0	1	0	0	{ 000, {001}, 010 }	
1	0	0	0	{ 000, 001, {010} }	
0	1	0	1	{ 000, {001}, 010 }	(0, 1)
<hr/>					
0	0	1	1	{ {011}, 100 }	
0	1	1	0	{ {011}, 100 }	
1	0	0	1	{ {011}, 100 }	
1	0	1	0	{ {011}, {100} }	
1	1	0	0	{ {011}, 100 }	
0	1	1	1	{ {011}, {100} }	(0, 0)
1	1	0	1	{ {011}, {100} }	
<hr/>					
1	0	1	1	{ {101}, 110, 111 }	
1	1	1	0	{ {101}, 110, 111 }	
1	1	1	1	{ {101}, {110}, 111 }	(1, 0)

Minimization of Boolean Relations

- Since there are many possible output values (for any input), there are many logic functions implementing the relation
 - Compatible functions
- Problem
 - Find a minimum compatible function
- Do not enumerate all compatible functions
 - Compute the primes of the compatible functions
 - C-primes
 - Derive a logic cover from the c-primes

Binate Covering

- Covering problem is more complex
 - As compared to minimizing logic functions.
- In classic Boolean minimization we just need enough implicants to cover the minterm
 - Covering clause is **unate** in all variables
 - Any additional implicant does not hurt
- In Boolean relation optimization, we need to pick implicants to realize a compatible function
 - Some implicants cannot be taken together
 - Covering clause is **binate** (implicant mutual exclusion)
 - Non-compact Boolean space

Solving Binate Covering

- Binate cover can be solved with branch and bound
 - In practice much more difficult to solve, because it is harder to bound effectively
- Binate cover can be reduced to min-cost SAT
 - SAT solvers can be used
- Binate cover can be also modeled by BDDs
- Several approximation algorithms for binate cover

Boolean Relations

- Generalization of Boolean functions
 - More degrees of freedom than don't care sets
- Useful to represent multiple choice
- Useful to model internals of logic networks
- Elegant formalism, but computationally-intensive solution method

