

Introduction to Electronic Design Automation

Jie-Hong Roland Jiang

江介宏

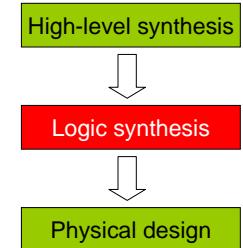
Department of Electrical Engineering
National Taiwan University



Spring 2011

1

Logic Synthesis



Part of the slides are by courtesy of Prof. Andreas Kuehlmann

2

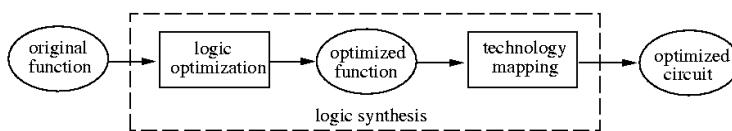
Logic Synthesis

Course contents

- Overview
- Boolean function representation
- Logic optimization
- Technology mapping

Reading

- Chapter 6



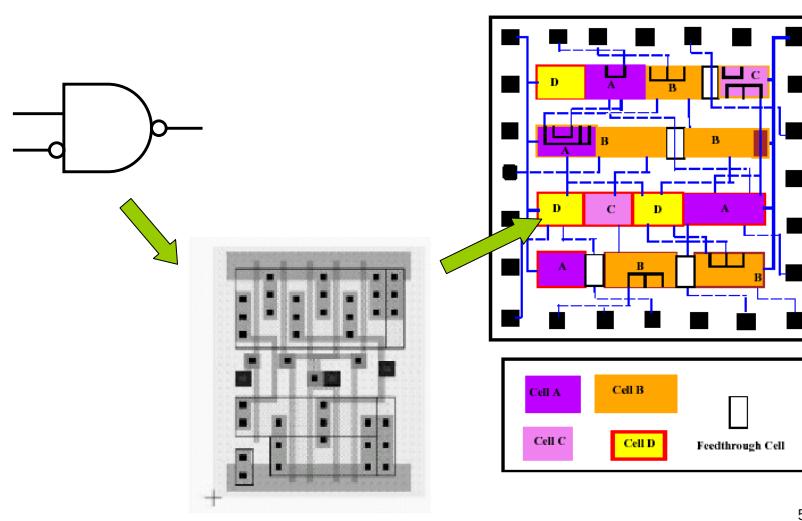
3

High-Level to Logic Synthesis

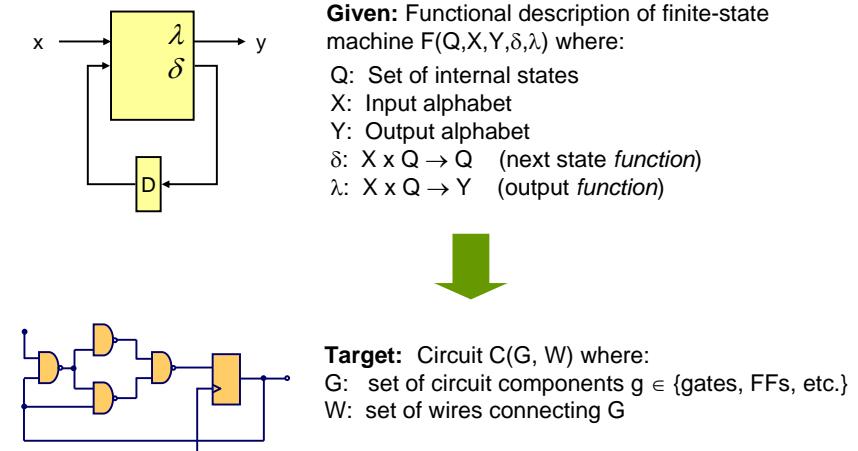
- Hardware is normally partitioned into two parts:
 - **Data path:** a network of functional units, registers, multiplexers and buses.
 - **Control:** the circuit that takes care of having the data present at the right place at a specific time (i.e. FSM), or of presenting the right instructions to a programmable unit (i.e. microcode).
- High-level synthesis often focuses on data-path optimization
 - The control part is then realized as an FSM
- Logic synthesis often focuses on control-logic optimization
 - Logic synthesis is widely used in application-specific IC (ASIC) design, where standard cell design style is most common

4

Standard-Cell Based Design



Transformation of Logic Synthesis



Boolean Function Representation

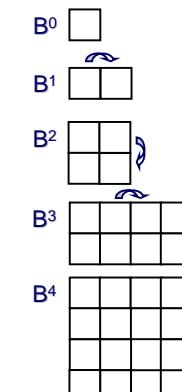
- Logic synthesis translates Boolean functions into circuits
- We need representations of Boolean functions for two reasons:
 - to represent and manipulate the actual circuit that we are implementing
 - to facilitate *Boolean reasoning*

7

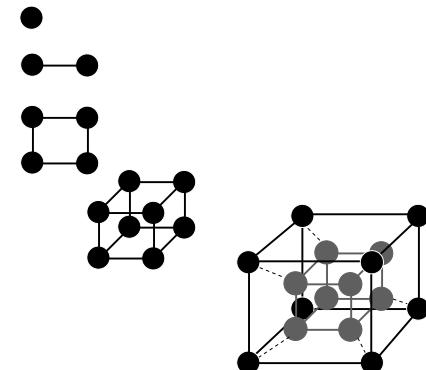
Boolean Space

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

Karnaugh Maps:



Boolean Lattices:



8

Boolean Function

- A Boolean function f over input variables: x_1, x_2, \dots, x_m , is a mapping $f: \mathbf{B}^m \rightarrow Y^n$, where $\mathbf{B} = \{0,1\}$ and $Y = \{0,1,d\}$
 - E.g.
 - The output value of $f(x_1, x_2, x_3)$, say, partitions \mathbf{B}^m into three sets:
 - on-set** ($f=1$)
 - E.g. $\{010, 011, 110, 111\}$ (characteristic function $f^1 = x_2$)
 - off-set** ($f=0$)
 - E.g. $\{100, 101\}$ (characteristic function $f^0 = x_1 \neg x_2$)
 - don't-care set** ($f=d$)
 - E.g. $\{000, 001\}$ (characteristic function $f^d = \neg x_1 \neg x_2$)
- f is an **incompletely specified function** if the don't-care set is nonempty. Otherwise, f is a **completely specified function**
 - Unless otherwise said, a Boolean function is meant to be completely specified

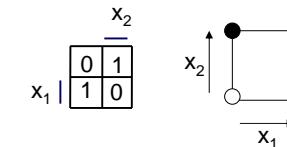
9

Boolean Function

- A Boolean function $f: \mathbf{B}^n \rightarrow \mathbf{B}$ over variables x_1, \dots, x_n maps each Boolean valuation (truth assignment) in \mathbf{B}^n to 0 or 1

Example

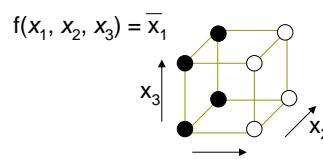
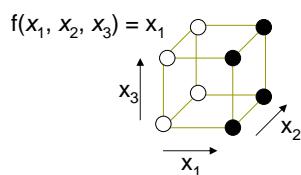
$f(x_1, x_2)$ with $f(0,0) = 0$, $f(0,1) = 1$, $f(1,0) = 1$, $f(1,1) = 0$



10

Boolean Function

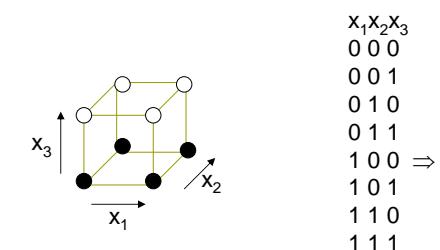
- Onset** of f , denoted as f^1 , is $f^1 = \{v \in \mathbf{B}^n \mid f(v)=1\}$
 - If $f^1 = \mathbf{B}^n$, f is a **tautology**
- Offset** of f , denoted as f^0 , is $f^0 = \{v \in \mathbf{B}^n \mid f(v)=0\}$
 - If $f^0 = \mathbf{B}^n$, f is **unsatisfiable**. Otherwise, f is **satisfiable**.
- f^1 and f^0 are sets, not functions!**
- Boolean functions f and g are **equivalent** if $\forall v \in \mathbf{B}^n$, $f(v) = g(v)$ where v is a truth assignment or Boolean valuation
- A **literal** is a Boolean variable x or its negation x' (or $x, \neg x$) in a Boolean formula



11

Boolean Function

- There are 2^n vertices in \mathbf{B}^n
- There are 2^{2^n} distinct Boolean functions
- Each subset $f^1 \subseteq \mathbf{B}^n$ of vertices in \mathbf{B}^n forms a distinct Boolean function f with onset f^1



$x_1 x_2 x_3$	f
0 0 0	1
0 0 1	0
0 1 0	1
0 1 1	0
1 0 0	$\Rightarrow 1$
1 0 1	0
1 1 0	1
1 1 1	0

12

Boolean Operations

Given two Boolean functions:

$$f : \mathbf{B}^n \rightarrow \mathbf{B}$$

$$g : \mathbf{B}^n \rightarrow \mathbf{B}$$

- $h = f \wedge g$ from **AND** operation is defined as

$$h^1 = f^1 \cap g^1; h^0 = \mathbf{B}^n \setminus h^1$$

- $h = f \vee g$ from **OR** operation is defined as

$$h^1 = f^1 \cup g^1; h^0 = \mathbf{B}^n \setminus h^1$$

- $h = \neg f$ from **COMPLEMENT** operation is defined as

$$h^1 = f^0; h^0 = f^1$$

13

Cofactor and Quantification

Given a Boolean function:

$$f : \mathbf{B}^n \rightarrow \mathbf{B}, \text{ with the input variable } (x_1, x_2, \dots, x_i, \dots, x_n)$$

- **Positive cofactor on variable x_i**

$$h = f_{x_i} \text{ is defined as } h = f(x_1, x_2, \dots, 1, \dots, x_n)$$

- **Negative cofactor on variable x_i**

$$h = f_{-x_i} \text{ is defined as } h = f(x_1, x_2, \dots, 0, \dots, x_n)$$

- **Existential quantification over variable x_i**

$$h = \exists x_i. f \text{ is defined as } h = f(x_1, x_2, \dots, 0, \dots, x_n) \vee f(x_1, x_2, \dots, 1, \dots, x_n)$$

- **Universal quantification over variable x_i**

$$h = \forall x_i. f \text{ is defined as } h = f(x_1, x_2, \dots, 0, \dots, x_n) \wedge f(x_1, x_2, \dots, 1, \dots, x_n)$$

- **Boolean difference over variable x_i**

$$h = \partial f / \partial x_i \text{ is defined as } h = f(x_1, x_2, \dots, 0, \dots, x_n) \oplus f(x_1, x_2, \dots, 1, \dots, x_n)$$

14

Boolean Function Representation

- Some common representations:

- Truth table
- Boolean formula
 - SOP (sum-of-products, or called disjunctive normal form, DNF)
 - POS (product-of-sums, or called conjunctive normal form, CNF)
- BDD (binary decision diagram)
- Boolean network (consists of nodes and wires)
 - Generic Boolean network
 - Network of nodes with generic functional representations or even subcircuits
 - Specialized Boolean network
 - Network of nodes with SOPs (PLAs)
 - And-Inv Graph (AIG)

- Why different representations?

- Different representations have their own strengths and weaknesses (no single data structure is best for all applications)

15

Boolean Function Representation Truth Table

- Truth table (function table for multi-valued functions):

The **truth table** of a function $f : \mathbf{B}^n \rightarrow \mathbf{B}$ is a tabulation of its value at each of the 2^n vertices of \mathbf{B}^n .

In other words the truth table lists all **mintems**

$$\begin{aligned} \text{Example: } f = & a'b'c'd + a'b'cd + a'bc'd + \\ & ab'c'd + ab'cd + abc'd + \\ & abcd' + abcd \end{aligned}$$

The truth table representation is

- impractical for large n
- canonical

If two functions are the equal, then their **canonical** representations are isomorphic.

abcd	f	abcd	f
0	0000	0	1000
1	0001	1	1001
2	0010	0	1010
3	0011	1	1011
4	0100	0	1100
5	0101	1	1101
6	0110	0	1110
7	0111	0	1111

16

Boolean Function Representation

Boolean Formula

- A **Boolean formula** is defined inductively as an expression with the following formation rules (syntax):

```
formula ::=      (' formula ')
           | Boolean constant      (true or false)
           | <Boolean variable>
           | formula "+" formula  (OR operator)
           | formula "." formula  (AND operator)
           | ¬ formula            (complement)
```

Example

$$f = (x_1 \cdot x_2) + (x_3) + \neg(\neg(x_4 \cdot \neg x_1))$$

typically “.” is omitted and ‘(‘, ‘)’ are omitted when the operator priority is clear, e.g., $f = x_1 x_2 + x_3 + x_4 \neg x_1$

17

Boolean Function Representation

Boolean Formula in SOP

- Any function can be represented as a **sum-of-products (SOP)**, also called **sum-of-cubes** (a **cube** is a product term), or **disjunctive normal form (DNF)**

Example

$$\varphi = ab + a'c + bc$$

18

Boolean Function Representation

Boolean Formula in POS

- Any function can be represented as a **product-of-sums (POS)**, also called **conjunctive normal form (CNF)**
- Dual of the SOP representation

Example

$$\varphi = (a+b+c) (a'+b+c) (a+b'+c') (a+b+c)$$

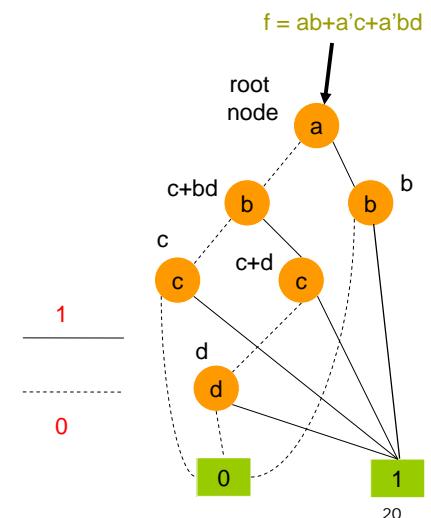
- Exercise: Any Boolean function in POS can be converted to SOP using De Morgan's law and the distributive law, and vice versa

19

Boolean Function Representation

Binary Decision Diagram

- BDD – a graph representation of Boolean functions
 - A **leaf node** represents constant 0 or 1
 - A **non-leaf node** represents a decision node (multiplexer) controlled by some variable
 - Can make a BDD representation **canonical** by imposing the **variable ordering** and **reduction** criteria (ROBDD)



20

Boolean Function Representation Binary Decision Diagram

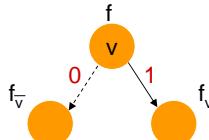
- Any Boolean function f can be written in term of **Shannon expansion**

$$f = v f_v + \neg v f_{\neg v}$$

- Positive cofactor: $f_{x_i} = f(x_1, \dots, x_i=1, \dots, x_n)$
- Negative cofactor: $f_{\neg x_i} = f(x_1, \dots, x_i=0, \dots, x_n)$

- BDD is a compressed Shannon cofactor tree:

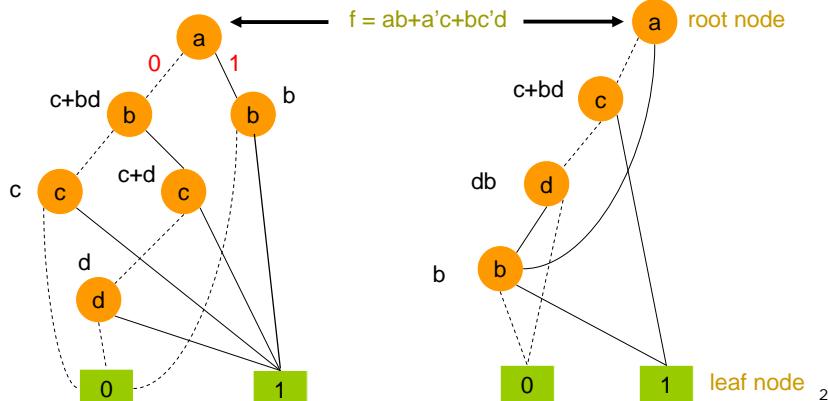
- The two children of a node with function f controlled by variable v represent two sub-functions f_v and $f_{\neg v}$



21

Boolean Function Representation Binary Decision Diagram

- For a Boolean function,
 - ROBDD is unique with respect to a given variable ordering
 - Different orderings may result in different ROBDD structures



23

Boolean Function Representation Binary Decision Diagram

- Reduced and ordered BDD (ROBDD) is a **canonical** Boolean function representation

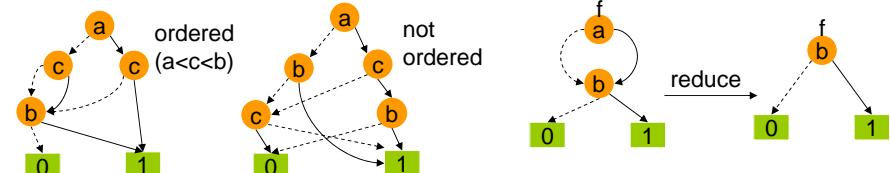
Ordered:

- cofactor variables are in the **same order** along all paths
- $x_{i_1} < x_{i_2} < x_{i_3} < \dots < x_{i_n}$

Reduced:

- any node with two identical children is removed
- two nodes with isomorphic BDD's are merged

These two rules make any node in an ROBDD represent a distinct logic function



22

Boolean Function Representation Boolean Network

- A **Boolean network** is a directed graph $C(G, N)$ where G are the gates and $N \subseteq (G \times G)$ are the directed edges (nets) connecting the gates.

Some of the vertices are designated:

Inputs: $I \subseteq G$

Outputs: $O \subseteq G$

$$I \cap O = \emptyset$$

Each gate g is assigned a Boolean function f_g which computes the output of the gate in terms of its inputs.

24

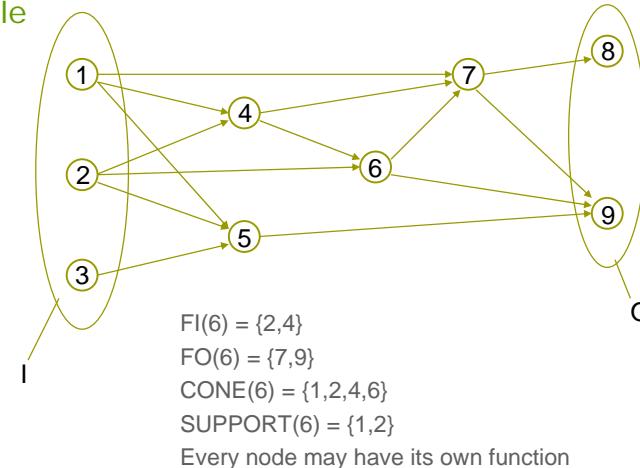
Boolean Function Representation Boolean Network

- The **fanin** $FI(g)$ of a gate g are the predecessor gates of g :
 $FI(g) = \{g' \mid (g',g) \in N\}$ (N : the set of nets)
- The **fanout** $FO(g)$ of a gate g are the successor gates of g :
 $FO(g) = \{g' \mid (g,g') \in N\}$
- The **cone** $CONE(g)$ of a gate g is the transitive fanin (TFI) of g and g itself
- The **support** $SUPPORT(g)$ of a gate g are all inputs in its cone:
 $SUPPORT(g) = CONE(g) \cap I$

25

Boolean Function Representation Boolean Network

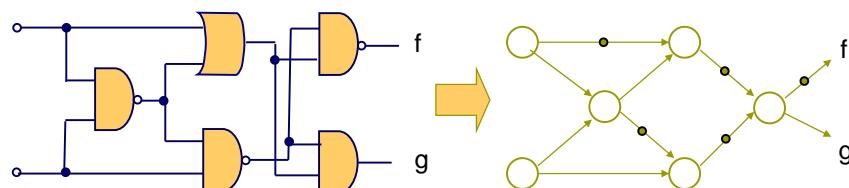
Example



26

Boolean Function Representation And-Inverter Graph

- AND-INVERTER graphs (AIGs)
 vertices: 2-input AND gates
 edges: interconnects with (optional) dots representing INVs
- Hash table to identify and reuse structurally isomorphic circuits



27

Boolean Function Representation

- A **canonical form** of a Boolean function is a **unique** representation of the function
 - It can be used for verification purposes

Example

- Truth table is canonical
 - It grows exponentially with the number of input variables
- ROBDD is canonical
 - It is of practical interests because it may represent many Boolean functions compactly
- SOP, POS, Boolean networks are NOT canonical

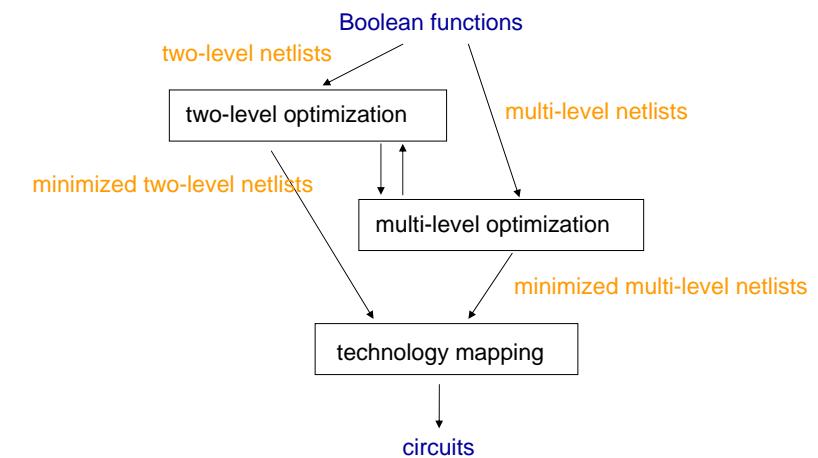
28

Boolean Function Representation

- ❑ Truth table
 - Canonical
 - Useful in representing small functions
- ❑ SOP
 - Useful in two-level logic optimization, and in representing local node functions in a Boolean network
- ❑ POS
 - Useful in SAT solving and Boolean reasoning
 - Rarely used in circuit synthesis (due to the asymmetric characteristics of NMOS and PMOS)
- ❑ ROBDD
 - Canonical
 - Useful in Boolean reasoning
- ❑ Boolean network
 - Useful in multi-level logic optimization
- ❑ AIG
 - Useful in multi-level logic optimization and Boolean reasoning

29

Logic Optimization



30

Two-Level Logic Minimization

- ❑ Any Boolean function can be realized using PLA in two levels: AND-OR (sum of products), NAND-NAND, etc.
 - Direct implementation of two-level logic using PLAs (programmable logic arrays) is not as popular as in the nMOS days
- ❑ Classic problem solved by the *Quine-McCluskey* algorithm
 - Popular cost function: #cubes and #literals in an SOP expression
 - #cubes – #rows in a PLA
 - #literals – #transistors in a PLA
 - The goal is to find a **minimal irredundant prime cover**

31

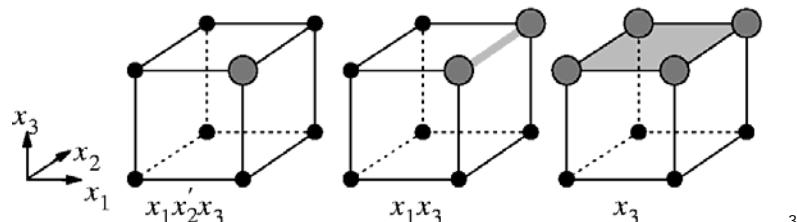
Two-Level Logic Minimization

- ❑ Exact algorithm
 - Quine-McCluskey's procedure
- ❑ Heuristic algorithm
 - Espresso

32

Two-Level Logic Minimization Minterms and Cubes

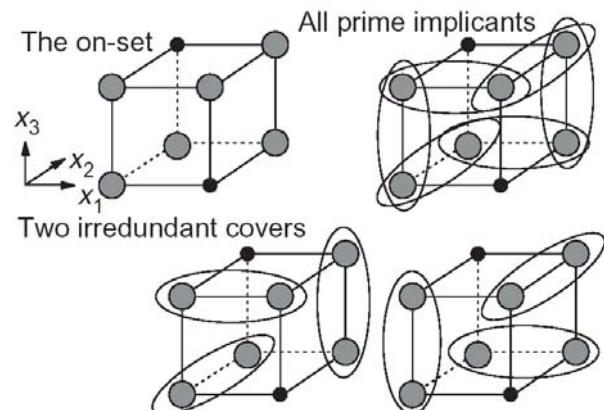
- A **minterm** is a product of **every** input variable or its negation
 - A minterm corresponds to a single point in B^n
- A **cube** is a product of literals
 - The fewer the number of literals is in the product, the bigger the space is covered by the cube



33

Two-Level Logic Minimization Cover

- Example
 - $f = \neg x_1 \neg x_3 + \neg x_2 x_3 + x_1 x_2$
 - $f = \neg x_1 \neg x_2 + x_2 \neg x_3 + x_1 x_3$



35

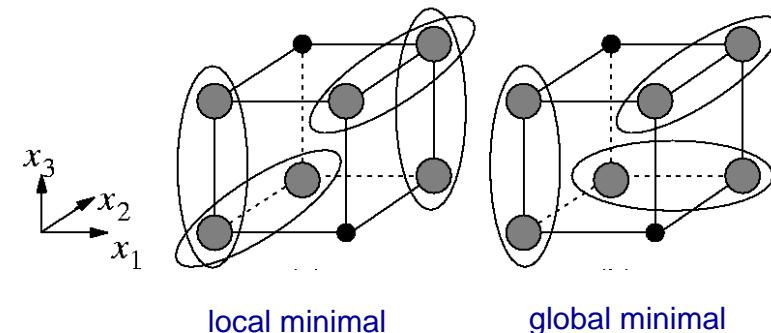
Two-Level Logic Minimization Implicant and Cover

- An **implicant** is a cube whose points are either in the on-set or the dc-set.
- A **prime implicant** is an implicant that is not included in any other implicant.
- A set of prime implicants that together cover all points in the on-set (and some or all points of the dc-set) is called a **prime cover**.
 - A prime cover is **irredundant** when none of its prime implicants can be removed from the cover.
 - An irredundant prime cover is **minimal** when the cover has the minimal number of prime implicants.
 - (c.f. minimum vs. minimal)

34

Two-Level Logic Minimization Cover

- Example



36

Two-Level Logic Minimization Quine-McCluskey Procedure

- Given G and D (covers for $\mathfrak{I} = (f, d, r)$ and d , respectively), find a minimum cover G^* of primes where:

$$f \subseteq G^* \subseteq f + d \quad (G^* \text{ is a prime cover of } \mathfrak{I})$$
 - f is the onset, d don't-care set, and r offset
- Q-M Procedure:**
 - Generate all primes of \mathfrak{I} , $\{P_j\}$ (i.e. primes of $(f+d) = G+D$)
 - Generate all minterms $\{m_i\}$ of $f = G \wedge \neg D$
 - Build Boolean matrix B where

$$B_{ij} = 1 \text{ if } m_i \in P_j$$

$$= 0 \text{ otherwise}$$
 - Solve the minimum column covering problem for B (unate covering problem)

37

Two-Level Logic Minimization Quine-McCluskey Procedure

Generating Primes

Tabular method
(based on *consensus* operation):

- Start with all **minterm canonical form** of F
- Group **pairs** of adjacent minterms into cubes
- Repeat merging cubes until no more merging possible; mark (\checkmark) + remove all covered cubes.
- Result: set of **primes** of f .

Example

$$F = x'y' + wx'y + x'y'z' + w'y'z$$

$$F = x'y' + wx'y + x'y'z' + w'y'z$$

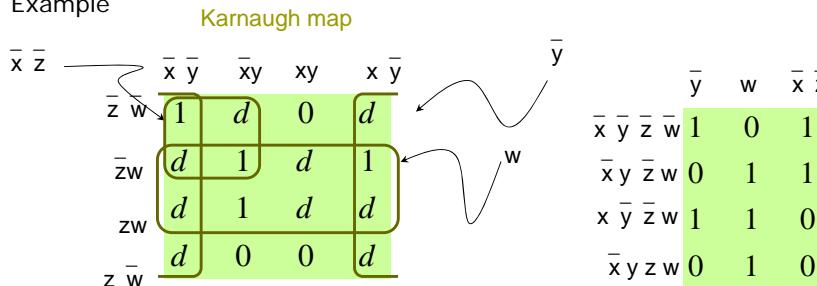
$w'x'y'z'$	\checkmark	$w'x'y'$	\checkmark	$x'y'$
		$w'x'z'$	\checkmark	$x'z'$
		$x'y'z'$	\checkmark	
$w'x'y'z'$	\checkmark	$x'y'z$	\checkmark	
		$x'y'z'$	\checkmark	
$wx'y'z'$	\checkmark	$wx'y'$	\checkmark	
		$wx'z'$	\checkmark	
$wx'y'z$	\checkmark	$wy'z$		
		$wy'z'$		
$wxy'z'$	\checkmark	wxy		
		wxz		
$wxyz$	\checkmark			

Courtesy: Maciej Ciesielski, UMASS

38

Two-Level Logic Minimization Quine-McCluskey Procedure

Example



$$F = \overline{xyzw} + \overline{xy} \overline{zw} + \overline{xy} \overline{zw} + \overline{xyzw} \quad (\text{cover of } \mathfrak{I})$$

$$D = \overline{yz} + xyw + \overline{x} \overline{yzw} + \overline{x} \overline{yw} + \overline{x} \overline{yzw} \quad (\text{cover of } d)$$

$$\text{Primes: } \overline{y} + w + \overline{x} \overline{z}$$

Covering Table

Solution: $\{1, 2\} \Rightarrow \overline{y} + w$ is a minimum prime cover (also $w + \overline{x} \overline{z}$)

39

Two-Level Logic Minimization Quine-McCluskey Procedure

Column covering of Boolean matrix

	\overline{y}	w	$\overline{x} \overline{z}$	Primes of $f+d$
$\overline{x} \overline{y} \overline{z} \overline{w}$	1	0	1	
$\overline{x} y \overline{z} \overline{w}$	0	1	1	
$x \overline{y} \overline{z} w$	1	1	0	
$\overline{x} y z w$	0	1	0	

Minterms of f

Essential prime

Row singleton (essential minterm)

- Definition.** An **essential prime** is a prime that covers an onset minterm of f not covered by any other primes.

40

Two-Level Logic Minimization Quine-McCluskey Procedure

■ Row equality in Boolean matrix:

- In practice, many rows in a covering table are identical. That is, there exist minterms that are contained in the same set of primes.

■ Example

m_1	0101101
m_2	0101101

41

Two-Level Logic Minimization Quine-McCluskey Procedure

■ Row dominance in Boolean matrix:

- A *row* i_1 whose set of primes is contained in the set of primes of *row* i_2 is said to **dominate** i_2 .

■ Example

i_1	011010
i_2	011110

■ i_1 dominates i_2

■ Can remove row i_2 because have to choose a prime to cover i_1 , and any such prime also covers i_2 . So i_2 is automatically covered.

42

Two-Level Logic Minimization Quine-McCluskey Procedure

■ Column dominance in Boolean matrix:

- A *column* j_1 whose rows are a superset of another *column* j_2 is said to **dominate** j_2 .

■ Example

j_1	j_2
1	0
0	0
1	1
0	0
1	1

■ j_1 dominates j_2

■ We can remove column j_2 since j_1 covers all those rows and more. We would never choose j_2 in a minimum cover since it can always be replaced by j_1 .

43

Two-Level Logic Minimization Quine-McCluskey Procedure

Reducing Boolean matrix

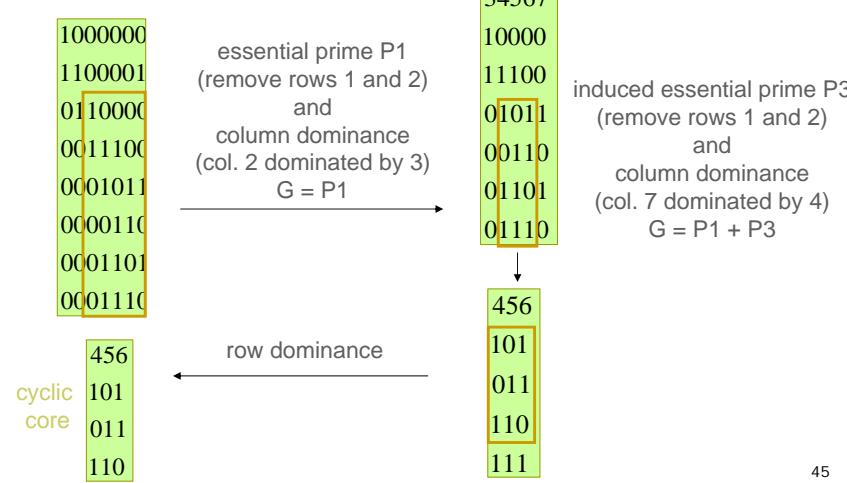
1. Remove all rows covered by essential primes (columns in row singltons). Put these primes in the cover G .
2. Group identical rows together and remove dominated rows.
3. Remove dominated columns. For equal columns, keep one prime to represent them.
4. Newly formed row singltons define **induced essential primes**.
5. Go to 1 if covering table decreased.

- The resulting reduced covering table is called the **cyclic core**. This has to be solved (**unate covering problem**). A minimum solution is added to G . The resulting G is a minimum cover.

44

Two-Level Logic Minimization Quine-McCluskey Procedure

Example (reducing Boolean matrix)



45

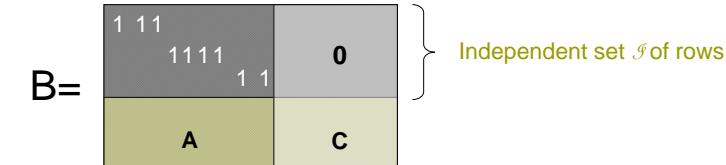
Two-Level Logic Minimization Quine-McCluskey Procedure

Solving cyclic core

- ◻ Best known method (for unate covering) is **branch and bound** with some clever bounding heuristics
- ◻ **Independent Set Heuristic:**
 - Find a maximum set I of “independent” rows. Two rows B_{i_1}, B_{i_2} are independent if **not** $\exists j$ such that $B_{i_1j} = B_{i_2j} = 1$. (They have no column in common.)

Example

A covering matrix B rearranged with independent sets first



46

Two-Level Logic Minimization Quine-McCluskey Procedure

Solving cyclic core

- ◻ Heuristic algorithm:
 - Let $I = \{I_1, I_2, \dots, I_k\}$ be the independent set of rows
- 1. choose $j \in I_i$ such that column j covers the most rows of A . Put P_j in G
- 2. eliminate all rows covered by column j
- 3. $I \leftarrow I \setminus \{I_j\}$
- 4. go to 1 if $|I| > 0$
- 5. If B is empty, then done (in this case achieve minimum solution)
- 6. If B is not empty, choose an independent set of B and go to 1



47

Two-Level Logic Minimization Quine-McCluskey Procedure

◻ Summary

- Calculate all prime implicants (of the union of the onset and don't care set)
- Find the minimal cover of all minterms in the onset by prime implicants
 - Construct the covering matrix
 - Simplify the covering matrix by detecting **essential** columns, **row and column dominance**
 - What is left is the **cyclic core** of the covering matrix.
 - The covering problem can then be solved by a branch-and-bound algorithm.

48

Two-Level Logic Minimization Exact vs. Heuristic Algorithms

Quine-McCluskey Method:

1. Generate cover of all primes $G = p_1 + p_2 + \dots + p_{3^n/n}$
2. Make G irredundant (in optimum way)
 - Q-M is **exact**, i.e., it gives an exact minimum

Heuristic Methods:

1. Generate (somehow) a cover of \mathfrak{I} using some of the primes $G = p_{i_1} + p_{i_2} + \dots + p_{i_k}$
2. Make G irredundant (maybe not optimally)
3. Keep best result - try again (i.e. go to 1)

49

Two-Level Logic Minimization ESPRESSO

Heuristic two-level logic minimization ESPRESSO(\mathfrak{I})

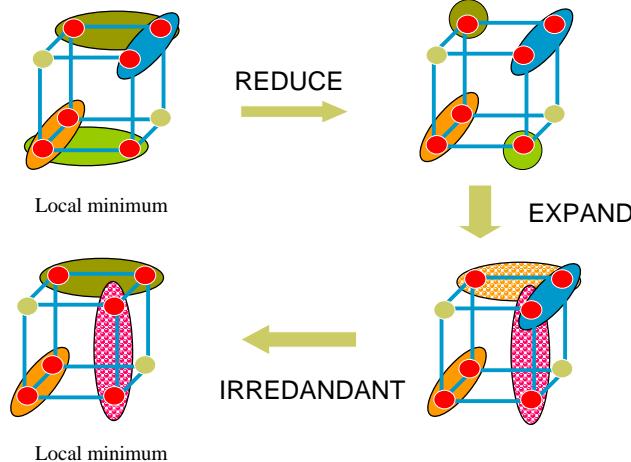
```

{
  (F,D,R) ← DECODE( $\mathfrak{I}$ )
  F ← EXPAND(F,R)
  F ← IRREDUNDANT(F,D)
  E ← ESSENTIAL_PRIMES(F,D)
  F ← F-E; D ← D + E
  do{
    do{
      F ← REDUCE(F,D)
      F ← EXPAND(F,R)
      F ← IRREDUNDANT(F,D)
    }while fewer terms in F
    F ← F + E; D ← D-E
    LOWER_OUTPUT(F,D)
    RAISE_INPUTS(F,R)
  }while fewer terms in F
  error ← (Fold ⊂ F) or (F ⊂ Fold + D)
  return (F,error)
}

```

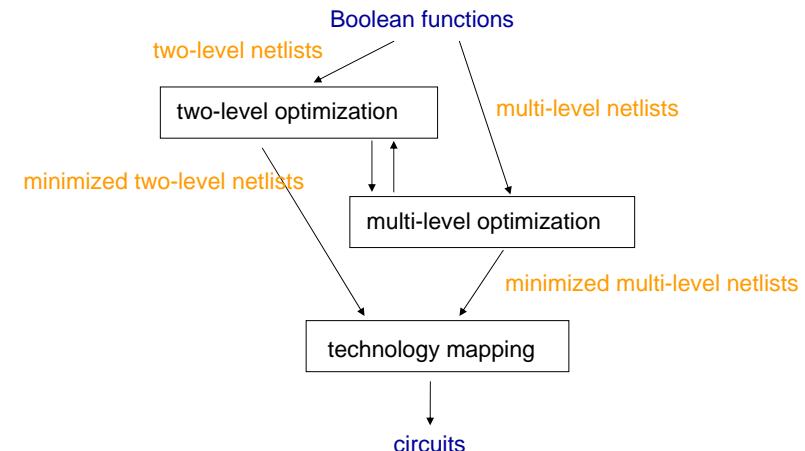
50

Two-Level Logic Minimization ESPRESSO



51

Logic Minimization



52