ECE/CS 3700: Fundamentals of Digital System Design

Chris J. Myers

Lecture 8: Optimized Implementation of Logic Circuits

Introduction

- Chapter 2 shows how to find a lowest-cost implementation using algebraic manipulation or Karnaugh maps.
- These methods do not scale well for larger circuits.
- Even when CAD tools are used, important to know what they are doing, so they can be configured properly.
- This chapter briefly introduces some of the logic synthesis algorithms used by these tools.
    - Multilevel synthesis
    - Alternative representations of logic functions
    - Optimization methods based on these representations

Multilevel Synthesis

- SOP or POS implementations are two-level circuits.
- Depending on technology, not always most efficient or even realizable (*fan-in problem*).
- Instead need to derive a *multilevel* implementation.
  - Factoring
  - Functional decomposition

Factoring

$$f(x_1, \ldots, x_7) = x_1 x_3 \overline{x}_6 + x_1 x_4 x_5 \overline{x}_6 + x_2 x_3 x_7 + x_2 x_4 x_5 x_7$$

Factoring

$$f(x_1, \ldots, x_7) = x_1 x_3 \overline{x}_6 + x_1 x_4 x_5 \overline{x}_6 + x_2 x_3 x_7 + x_2 x_4 x_5 x_7$$

Five 4-input LUTs!

Factoring

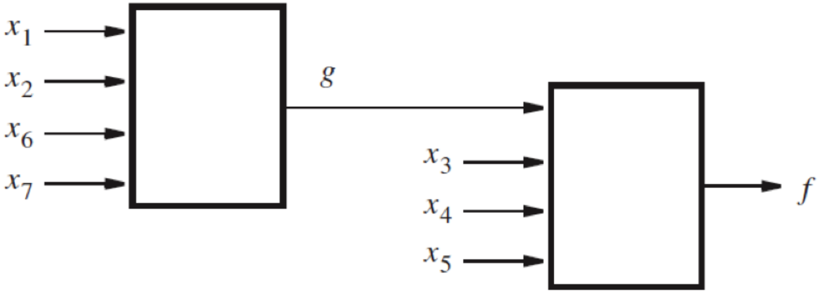
$$
\begin{aligned}
f(x_1, \ldots, x_7) &= x_1 x_3 \overline{x}_6 + x_1 x_4 x_5 \overline{x}_6 + x_2 x_3 x_7 + x_2 x_4 x_5 x_7 \\
&= x_1 \overline{x}_6 (x_3 + x_4 x_5) + x_2 x_7 (x_3 + x_4 x_5)
\end{aligned}
$$

Factoring

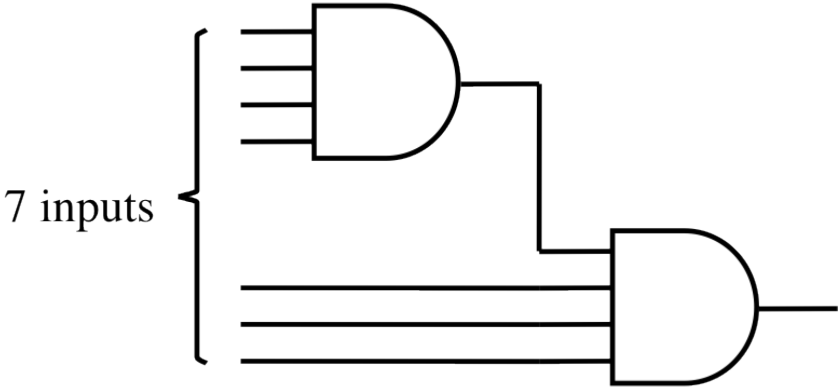
$$
\begin{aligned}
f(x_1, \ldots, x_7) &= x_1 x_3 \overline{x}_6 + x_1 x_4 x_5 \overline{x}_6 + x_2 x_3 x_7 + x_2 x_4 x_5 x_7 \\
&= x_1 \overline{x}_6 (x_3 + x_4 x_5) + x_2 x_7 (x_3 + x_4 x_5) \\
&= (x_1 \overline{x}_6 + x_2 x_7)(x_3 + x_4 x_5)
\end{aligned}
$$

Factoring

$$
\begin{aligned}
f(x_1, \ldots, x_7) &= x_1 x_3 \overline{x}_6 + x_1 x_4 x_5 \overline{x}_6 + x_2 x_3 x_7 + x_2 x_4 x_5 x_7 \\
&= x_1 \overline{x}_6 (x_3 + x_4 x_5) + x_2 x_7 (x_3 + x_4 x_5) \\
&= (x_1 \overline{x}_6 + x_2 x_7)(x_3 + x_4 x_5)
\end{aligned}
$$



Using 4-input AND Gates to Realize a 7-input Product Term



A Factored Circuit

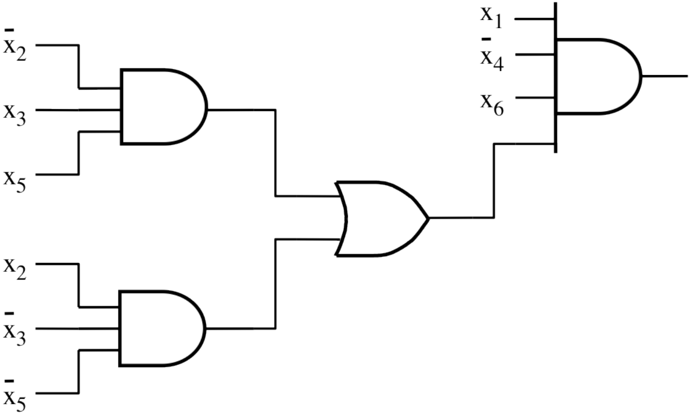
$$f = x_1\overline{x}_2x_3\overline{x}_4x_5x_6 + x_1x_2\overline{x}_3\overline{x}_4\overline{x}_5x_6$$

A Factored Circuit

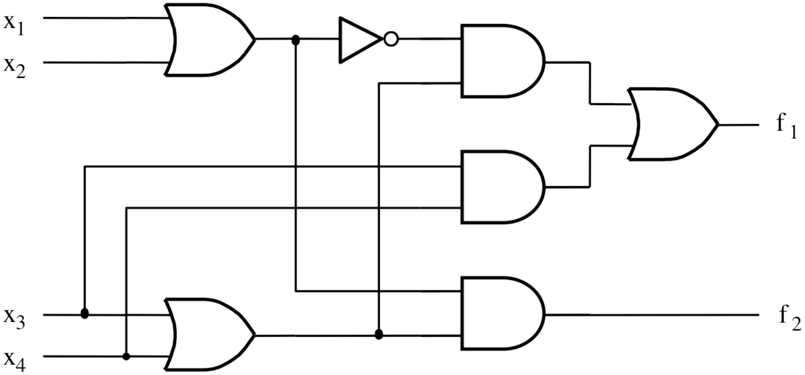
$$
\begin{aligned}
f &= x_1\overline{x}_2x_3\overline{x}_4x_5x_6 + x_1x_2\overline{x}_3\overline{x}_4\overline{x}_5x_6 \\
f &= x_1\overline{x}_4x_6(\overline{x}_2x_3x_5 + x_2\overline{x}_3\overline{x}_5)
\end{aligned}
$$

A Factored Circuit

$$f = x_1\overline{x}_2x_3\overline{x}_4x_5x_6 + x_1x_2\overline{x}_3\overline{x}_4\overline{x}_5x_6$$
$$f = x_1\overline{x}_4x_6(\overline{x}_2x_3x_5 + x_2\overline{x}_3\overline{x}_5)$$



A Multilevel Circuit with Gate Sharing



Impact on Wiring Complexity

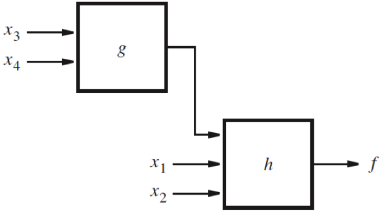
- Space on chip is used by gates and wires.
- Wires can be a significant portion.
- Each literal corresponds to a wire.
- Factoring reduces literal count, so it can also reduce wiring complexity.

Functional Decomposition

- Multilevel circuits often require less area.
- Complexity is reduced by decomposing 2-level function into subcircuits.
- Subcircuit implements function that may be used in multiple places.
- Reduces cost but increase propagation delay.

$x_3x_4 \backslash x_1x_2$	00	01 ($\bar{g}$)	11 (1)	10 (g)
00	0	1	1	0
01	0	0	1	1
11	0	1	1	0
10	0	0	1	1

(a) Subfunctions

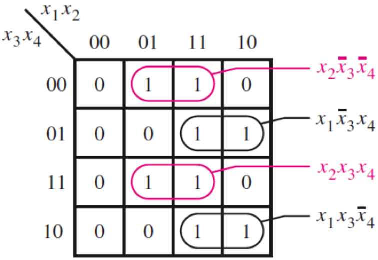


(b) The structure of decomposition

$g \backslash x_1x_2$	00	01	11	10	
0	0	1	1	0	$x_2\bar{g}$
1	0	0	1	1	x_1g

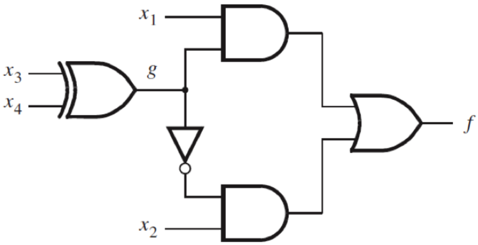
(c) Karnaugh map for $h(x_1, x_2, g)$

Functional Decomposition Example



x_3x_4 \ x_1x_2	00	01	11	10
00	0	1	1	0
01	0	0	1	1
11	0	1	1	0
10	0	0	1	1

(a) Product terms



(b) Multilevel circuit

Another Functional Decomposition Example

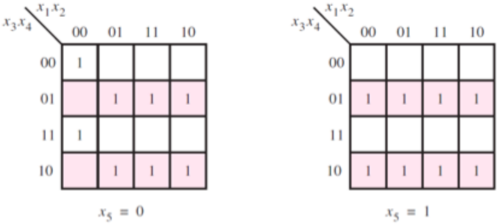
$x_5 = 0$

$x_3x_4 \backslash x_1x_2$	00	01	11	10
00	1			
01		1	1	1
11	1			
10		1	1	1

$x_5 = 1$

$x_3x_4 \backslash x_1x_2$	00	01	11	10
00				
01	1	1	1	1
11				
10	1	1	1	1

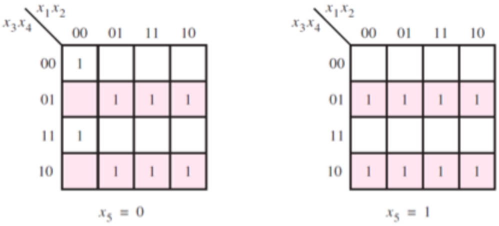
(a) Karnaugh map for the function f



(a) Karnaugh map for the function f

$$g(x_1, x_2, x_5) = x_1 + x_2 + x_5$$

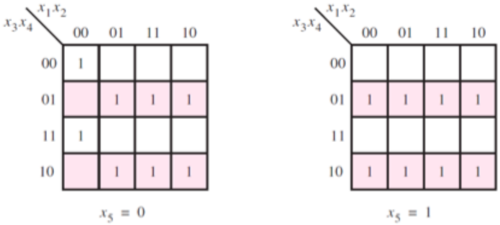
Another Functional Decomposition Example



(a) Karnaugh map for the function f

$$
\begin{aligned}
g(x_1, x_2, x_5) &= x_1 + x_2 + x_5 \\
k(x_3, x_4) &= \overline{x}_3 x_4 + x_3 \overline{x}_4 = x_3 \oplus x_4
\end{aligned}
$$

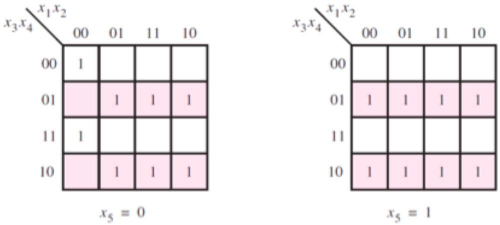
Another Functional Decomposition Example



(a) Karnaugh map for the function f

$$
\begin{aligned}
g(x_1, x_2, x_5) &= x_1 + x_2 + x_5 \\
k(x_3, x_4) &= \overline{x}_3 x_4 + x_3 \overline{x}_4 = x_3 \oplus x_4 \\
f(x_1, x_2, x_3, x_4, x_5) &= h[g(x_1, x_2, x_5), k(x_3, x_4)]
\end{aligned}
$$

Another Functional Decomposition Example



(a) Karnaugh map for the function f

$$
\begin{aligned}
g(x_1, x_2, x_5) &= x_1 + x_2 + x_5 \\
k(x_3, x_4) &= \overline{x}_3 x_4 + x_3 \overline{x}_4 = x_3 \oplus x_4 \\
f(x_1, x_2, x_3, x_4, x_5) &= h[g(x_1, x_2, x_5), k(x_3, x_4)] \\
&= kg + \overline{k}\,\overline{g} = \overline{k \oplus g}
\end{aligned}
$$

Another Functional Decomposition Example

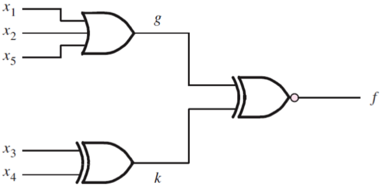
x_3x_4 \ x_1x_2	00	01	11	10
00	1			
01		1	1	1
11	1			
10		1	1	1

$x_5 = 0$

x_3x_4 \ x_1x_2	00	01	11	10
00				
01	1	1	1	1
11				
10	1	1	1	1

$x_5 = 1$

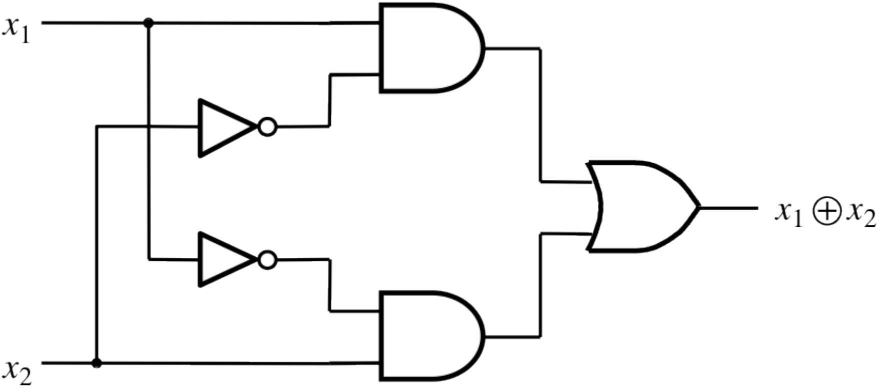
(a) Karnaugh map for the function f



(b) Circuit obtained using decomposition

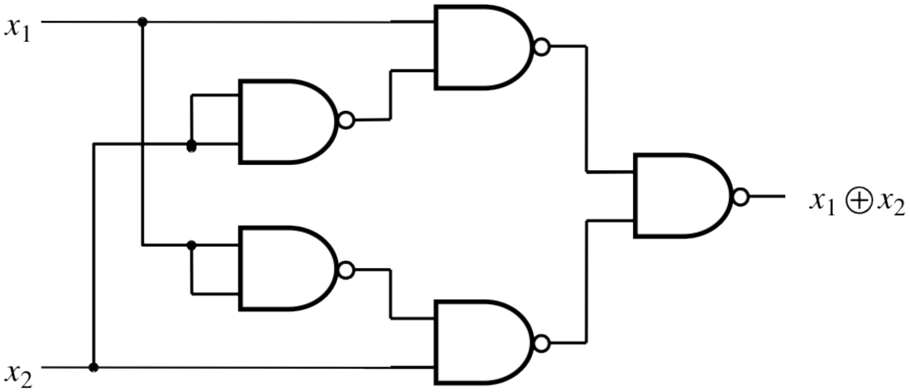
Cost = 10 while minimum-cost SOP = 55

Implementation of an XOR



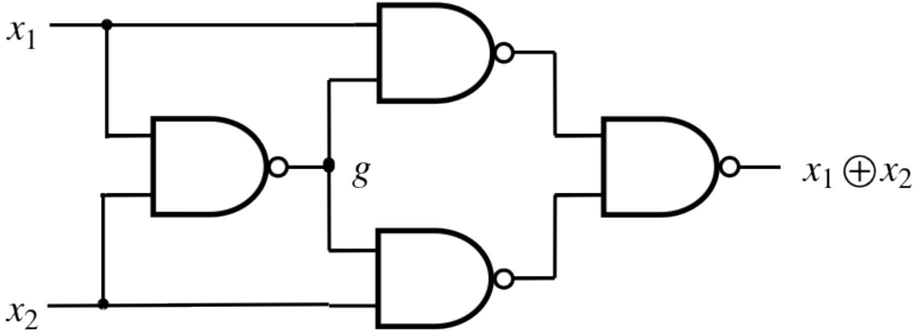
(a) Sum-of-products implementation

Implementation of an XOR



(b) NAND gate implementation

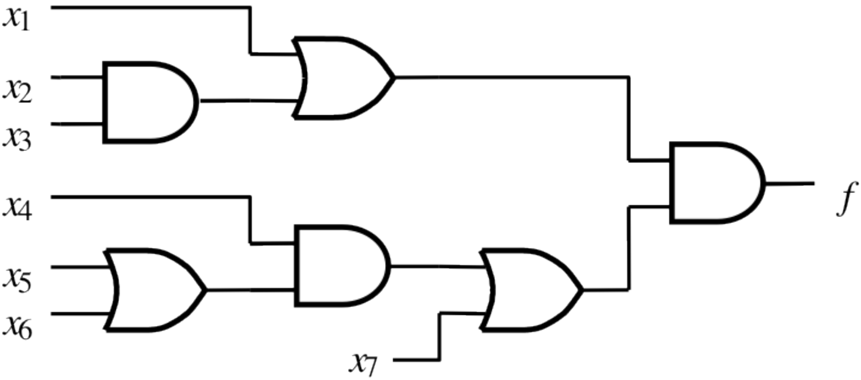
Implementation of an XOR



(c) Optimal NAND gate implementation

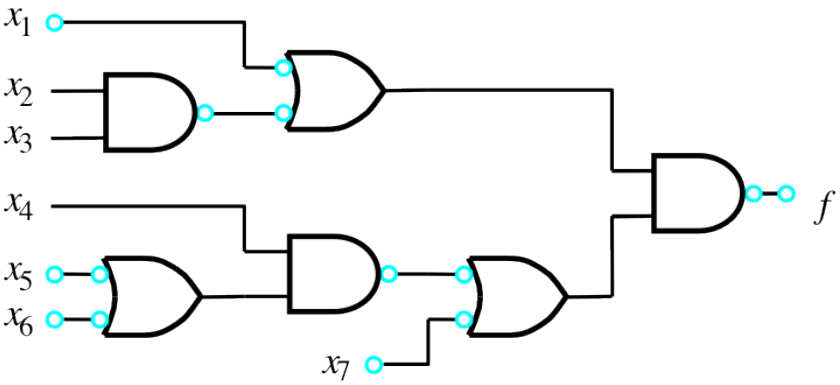
Practical Issues

- Functional decomposition is a powerful technique for reducing circuit complexity.
- However, enormous numbers of possible subfunctions leads to necessity for heuristic algorithms.



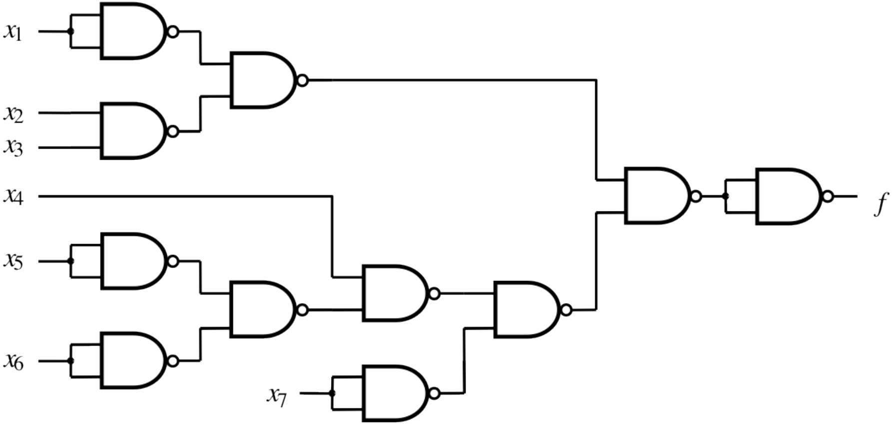
(a) Circuit with AND and OR gates

Conversion to a NAND-gate Circuit

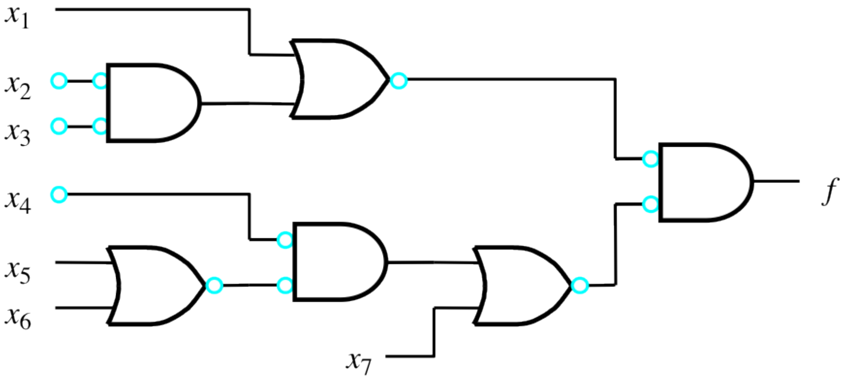


(b) Inversions needed to convert to NANDs

Conversion to a NAND-gate Circuit

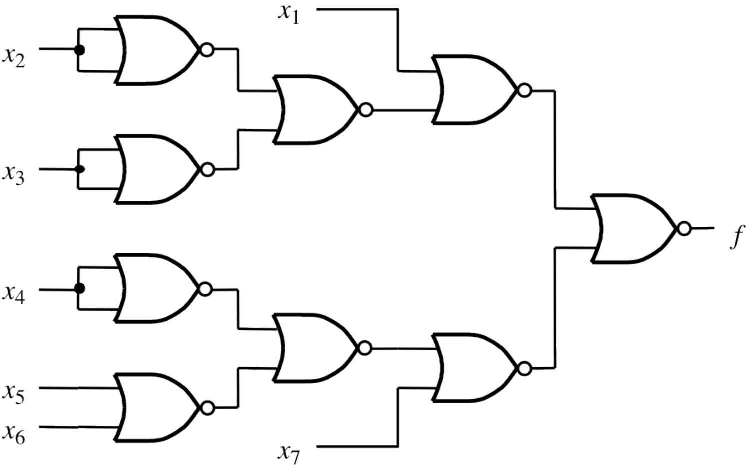


(c) NAND-gate circuit



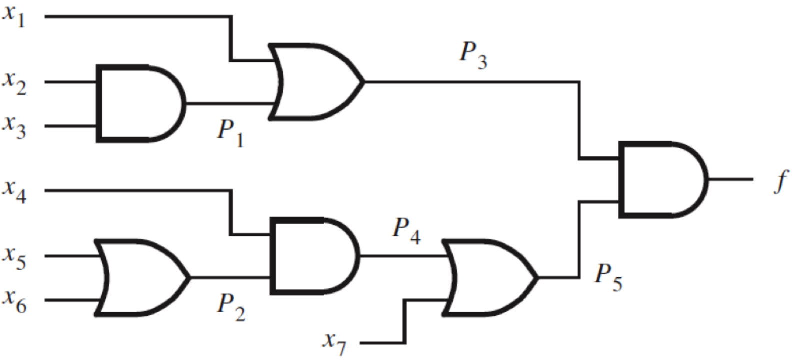
(a) Inversions needed to convert to NORs

Conversion to a NOR-gate Circuit



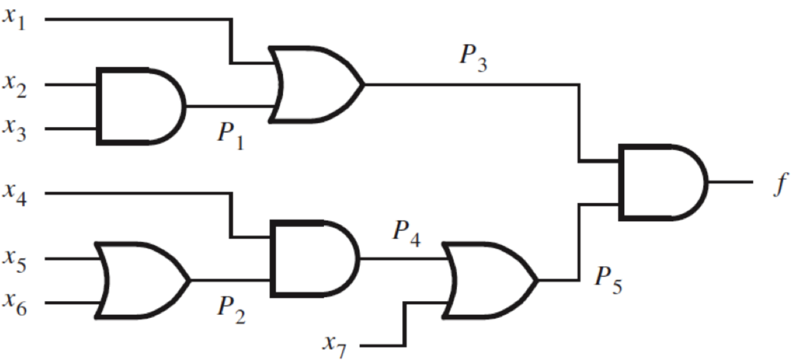
(b) NOR-gate circuit

Circuit Example for Analysis



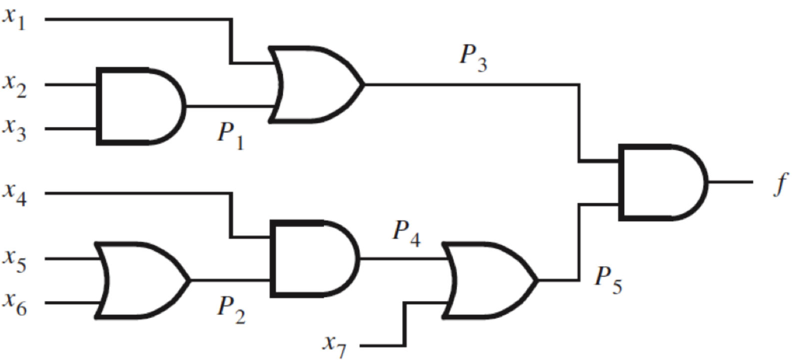
$$P_1 = x_2 x_3$$

Circuit Example for Analysis



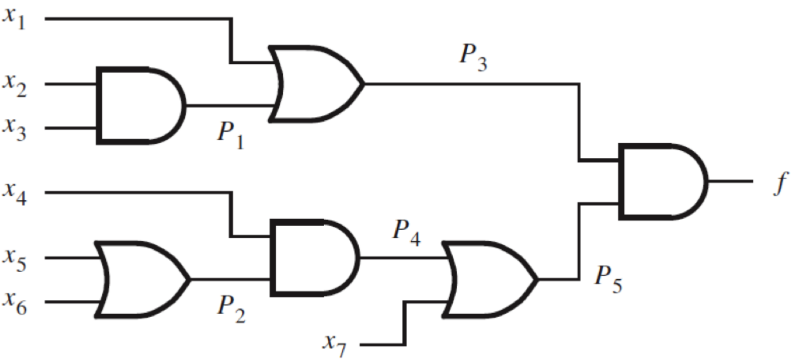
$$
\begin{aligned}
P_1 &= x_2 x_3 \\
P_3 &= x_1 + P_1 = x_1 + x_2 x_3
\end{aligned}
$$

Circuit Example for Analysis



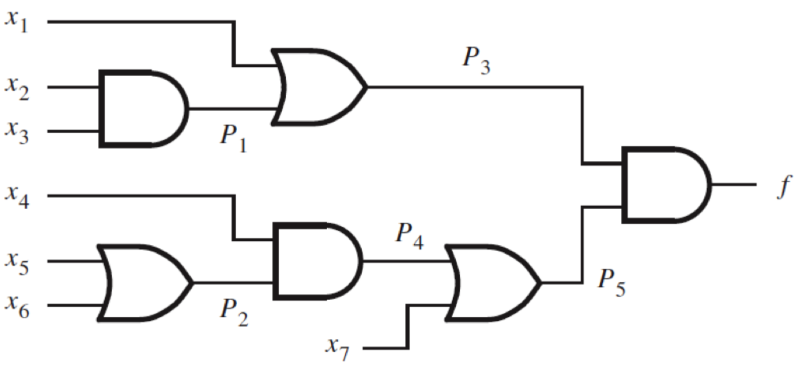
$$
\begin{aligned}
P_3 &= x_1 + P_1 = x_1 + x_2 x_3 \\
P_2 &= x_5 + x_6
\end{aligned}
$$

Circuit Example for Analysis



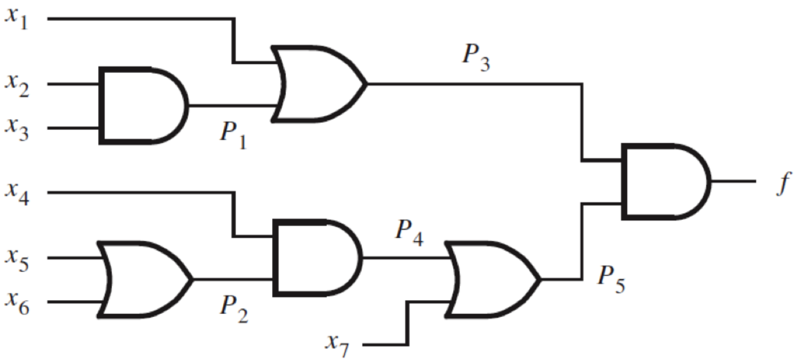
$$
\begin{aligned}
P_3 &= x_1 + P_1 = x_1 + x_2 x_3 \\
P_2 &= x_5 + x_6 \\
P_4 &= x_4 P_2 = x_4 (x_5 + x_6)
\end{aligned}
$$

Circuit Example for Analysis



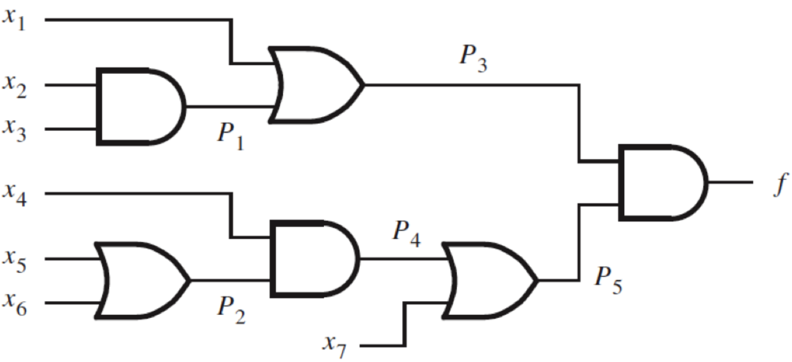
$$
\begin{aligned}
P_3 &= x_1 + P_1 = x_1 + x_2 x_3 \\
P_4 &= x_4 P_2 = x_4(x_5 + x_6) \\
P_5 &= P_4 + x_7 = x_4(x_5 + x_6) + x_7
\end{aligned}
$$

Circuit Example for Analysis



$$
\begin{aligned}
P_3 &= x_1 + P_1 = x_1 + x_2 x_3 \\
P_5 &= P_4 + x_7 = x_4(x_5 + x_6) + x_7 \\
f &= P_3 P_5 = (x_1 + x_2 x_3)(x_4(x_5 + x_6) + x_7)
\end{aligned}
$$

Circuit Example for Analysis



$$
\begin{aligned}
f &= P_3 P_5 = (x_1 + x_2 x_3)(x_4(x_5 + x_6) + x_7) \\
&= x_1 x_4 x_5 + x_1 x_4 x_6 + x_1 x_7 + x_2 x_3 x_4 x_5 + x_2 x_3 x_4 x_6 + x_2 x_3 x_7
\end{aligned}
$$

CAD Tools

- *espresso* - finds exact and heuristic solutions to the 2-level synthesis problem.
- *sis* - performs multilevel logic synthesis.
- Numerous commercial CAD packages are available from Cadence, Mentor, Synopsys, and others.

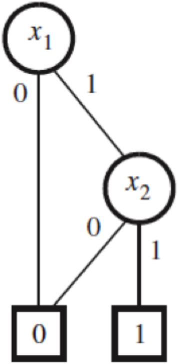
Logic Function Representation

- Truth tables
- Algebraic expressions
- Venn diagrams
- Karnaugh maps
- Binary decision diagrams (BDDs)
- *n*-dimensional cubes

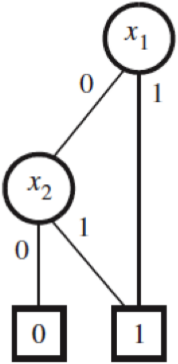
Logic Function Representation

- Truth tables
- Algebraic expressions
- Venn diagrams
- Karnaugh maps
- Binary decision diagrams (BDDs)
- *n*-dimensional cubes

Binary Decision Diagrams (BDDs)



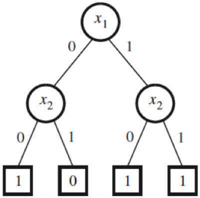
(a) AND



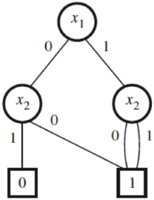
(b) OR

x_1	x_2	f
0	0	1
0	1	0
1	0	1
1	1	1

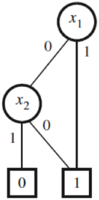
(a) Truth table



(b) Decision tree

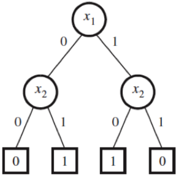


(c) Reducing nodes

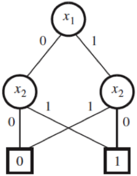


(d) BDD

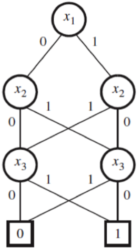
Derivation of BDDs for XOR Functions



(a) Decision tree



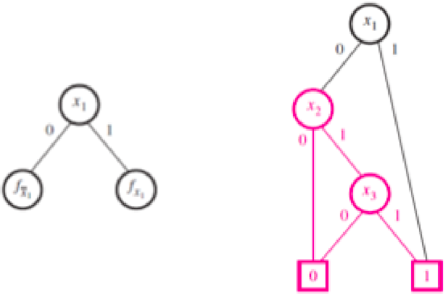
(b) BDD



(c) 3-input BDD

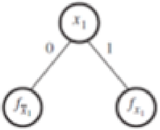
Derivation of BDDs Using Shannon's Expansion

$$f = x_1 + x_2 x_3$$

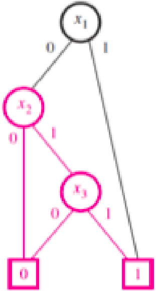


(a) Expansion using x_1 (b) BDD ordered x_1, x_2, x_3

$$f = x_1 + x_2 x_3$$
$$f_{\overline{x}_1} = x_2 x_3$$



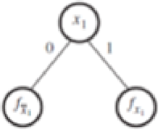
(a) Expansion using x_1



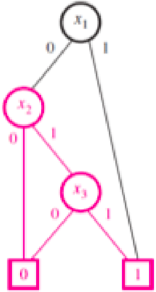
(b) BDD ordered x_1, x_2, x_3

Derivation of BDDs Using Shannon's Expansion

$$
\begin{aligned}
f &= x_1 + x_2 x_3 \\
f_{\overline{x}_1} &= x_2 x_3 \\
f_{x_1} &= 1
\end{aligned}
$$



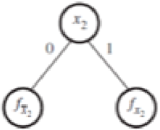
(a) Expansion using x_1



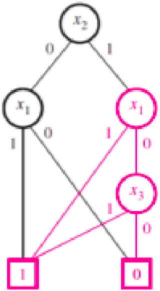
(b) BDD ordered x_1, x_2, x_3

Derivation of BDDs Using Shannon's Expansion

$$f = x_1 + x_2 x_3$$



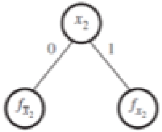
(c) Expansion using x_2



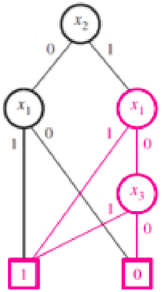
(d) BDD ordered x_2, x_1, x_3

Derivation of BDDs Using Shannon's Expansion

$$
\begin{aligned}
f &= x_1 + x_2 x_3 \\
f_{\overline{x}_2} &= x_1
\end{aligned}
$$



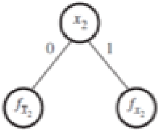
(c) Expansion using x_2



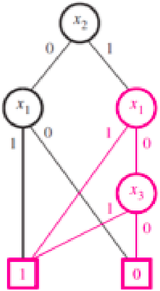
(d) BDD ordered x_2, x_1, x_3

Derivation of BDDs Using Shannon's Expansion

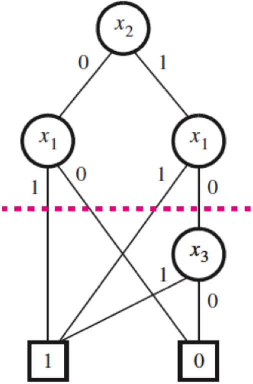
$$
\begin{aligned}
f &= x_1 + x_2 x_3 \\
f_{\overline{x}_2} &= x_1 \\
f_{x_2} &= x_1 + x_3
\end{aligned}
$$



(c) Expansion using x_2



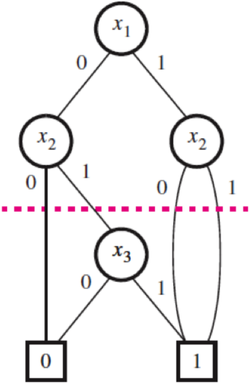
(d) BDD ordered x_2, x_1, x_3



(a) BDD ordered x_2, x_1, x_3

x_1	x_2	Node
0	0	0
0	1	x_3
1	0	1
1	1	1

(b) Truth table



(c) Order x_1, x_2, x_3

Derivation of a BDD Using Shannon Expansion

$$f = x_1x_3 + x_1x_4 + x_2x_4 + x_2x_3 + \overline{x}_1\overline{x}_2x_3x_4$$

Derivation of a BDD Using Shannon Expansion

$$
\begin{aligned}
f &= x_1 x_3 + x_1 x_4 + x_2 x_4 + x_2 x_3 + \overline{x}_1 \overline{x}_2 x_3 x_4 \\
f_{\overline{x}_1} &= x_2 x_4 + x_2 x_3 + \overline{x}_2 x_3 x_4
\end{aligned}
$$

Derivation of a BDD Using Shannon Expansion

$$
\begin{aligned}
f &= x_1x_3 + x_1x_4 + x_2x_4 + x_2x_3 + \overline{x}_1\overline{x}_2x_3x_4 \\
f_{\overline{x}_1} &= x_2x_4 + x_2x_3 + \overline{x}_2x_3x_4 \\
f_{\overline{x}_1\overline{x}_2} &= x_3x_4
\end{aligned}
$$

Derivation of a BDD Using Shannon Expansion

$$
\begin{aligned}
f &= x_1x_3 + x_1x_4 + x_2x_4 + x_2x_3 + \overline{x}_1\overline{x}_2x_3x_4 \\
f_{\overline{x}_1} &= x_2x_4 + x_2x_3 + \overline{x}_2x_3x_4 \\
f_{\overline{x}_1\overline{x}_2} &= x_3x_4 \\
f_{\overline{x}_1x_2} &= x_4 + x_3
\end{aligned}
$$

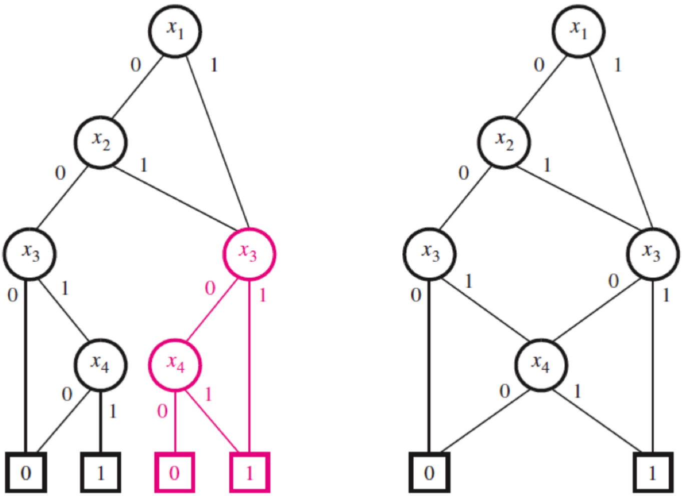
Derivation of a BDD Using Shannon Expansion

$$
\begin{aligned}
f &= x_1 x_3 + x_1 x_4 + x_2 x_4 + x_2 x_3 + \overline{x}_1 \overline{x}_2 x_3 x_4 \\
f_{\overline{x}_1} &= x_2 x_4 + x_2 x_3 + \overline{x}_2 x_3 x_4 \\
f_{\overline{x}_1 \overline{x}_2} &= x_3 x_4 \\
f_{\overline{x}_1 x_2} &= x_4 + x_3 \\
f_{x_1} &= x_3 + x_4 + x_2 x_4 + x_2 x_3
\end{aligned}
$$

Derivation of a BDD Using Shannon Expansion

$$
\begin{aligned}
f &= x_1 x_3 + x_1 x_4 + x_2 x_4 + x_2 x_3 + \overline{x}_1 \overline{x}_2 x_3 x_4 \\
f_{\overline{x}_1} &= x_2 x_4 + x_2 x_3 + \overline{x}_2 x_3 x_4 \\
f_{\overline{x}_1 \overline{x}_2} &= x_3 x_4 \\
f_{\overline{x}_1 x_2} &= x_4 + x_3 \\
f_{x_1} &= x_3 + x_4 + x_2 x_4 + x_2 x_3 \\
&= x_3 + x_4
\end{aligned}
$$

Derivation of a BDD Using Shannon Expansion



(a) Diagram (b) BDD

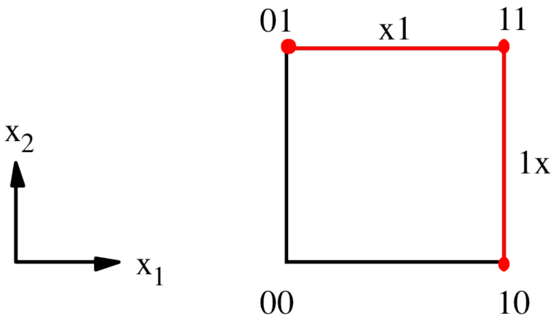
Practical Use of BDDs

- BDDs provide an efficient *canonical* representation of a Boolean function.
- Easily manipulated using *BDD packages* such as BuDDy or CUDD.
- A common data structure used in many logic synthesis tools.

Logic Function Representation

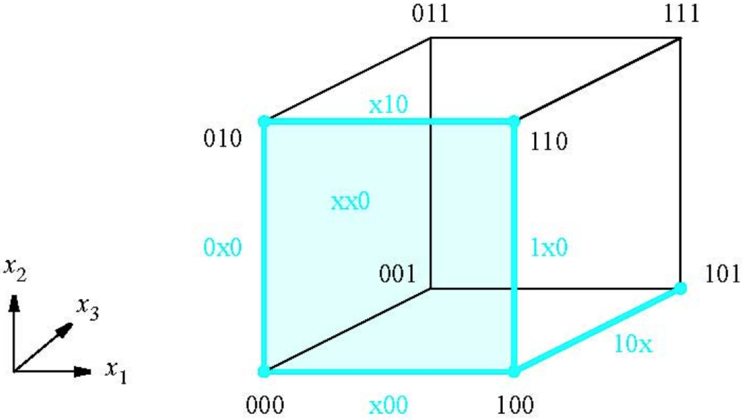
- Truth tables
- Algebraic expressions
- Venn diagrams
- Karnaugh maps
- Binary decision diagrams (BDDs)
- *n*-dimensional cubes

Representation of $f(x_1, x_2) = \sum m(1, 2, 3)$

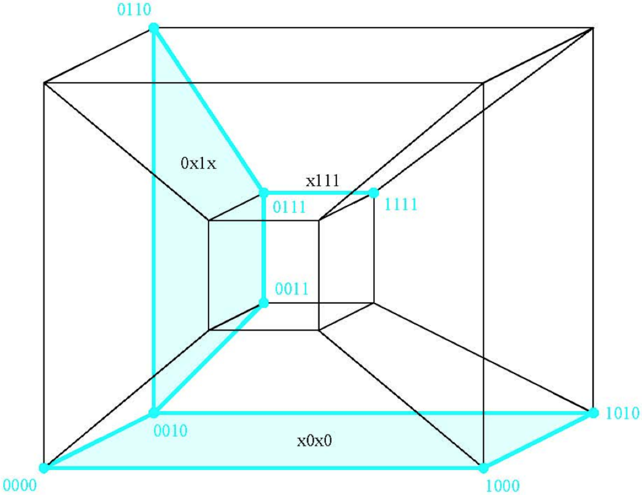


x_1	x_2	f
0	0	0
0	1	1
1	0	1
1	1	1

Representation of $f(x_1, x_2, x_3) = \sum m(0, 2, 4, 5, 6)$



Representation of $f(x_1, x_2, x_3, x_4) = \sum m(0, 2, 3, 6, 7, 8, 10, 15)$



n-Dimensional Hypercube

- Function of n variables maps to *n*-cube.
- *Size* of a cube is number of vertices.
- A cube with k x's consists of 2^k vertices.
- *n*-cube has 2^n vertices.
- 2 vertices are adjacent if they differ in one coordinate.
- Each vertex in *n*-cube adjacent to *n* others.

Optimization Based on Cubical Representation

- Optimization techniques often use cubical representation.
- Can be programmed and used efficiently in CAD tools.
- Example: Quine-McCluskey tabular method for minimization.

Generation of Prime Implicants

List 1

0	0 0 0 0	✓
4	0 1 0 0	✓
8	1 0 0 0	✓
10	1 0 1 0	✓
12	1 1 0 0	✓
11	1 0 1 1	✓
13	1 1 0 1	✓
15	1 1 1 1	✓

List 2

0,4	0 x 0 0	✓
0,8	x 0 0 0	✓
8,10	1 0 x 0	
4,12	x 1 0 0	✓
8,12	1 x 0 0	✓
10,11	1 0 1 x	
12,13	1 1 0 x	
11,15	1 x 1 1	
13,15	1 1 x 1	

List 3

0,4,8,12	x x 0 0

Selection of a Cover

Prime implicant	0	4	8	10	11	12	13	15
p_1 = 1 0 x 0			✓	✓				
p_2 = 1 0 1 x				✓	✓			
p_3 = 1 1 0 x						✓	✓	
p_4 = 1 x 1 1					✓			✓
p_5 = 1 1 x 1							✓	✓
p_6 = x x 0 0	✓	✓	✓			✓		

(a) Initial prime implicant cover table

Selection of a Cover

Prime implicant						0	4	8	10	11	12	13	15
p_1	=	1	0	x	0			✓	✓				
p_2	=	1	0	1	x				✓	✓			
p_3	=	1	1	0	x						✓	✓	
p_4	=	1	x	1	1					✓			✓
p_5	=	1	1	x	1							✓	✓
p_6	=	x	x	0	0	✓	✓	✓			✓		

(a) Initial prime implicant cover table

p_6 is an *essential prime implicant*.

Selection of a Cover

Prime implicant	Minterm 10	11	13	15
p_1	✓			
p_2	✓	✓		
p_3			✓	
p_4		✓		✓
p_5			✓	✓

(b) After the removal of essential prime implicants

Selection of a Cover

Prime implicant	Minterm 10	11	13	15
p_1	✓			
p_2	✓	✓		
p_3			✓	
p_4		✓		✓
p_5			✓	✓

(b) After the removal of essential prime implicants

Prime p_2 dominates p_1 and p_5 dominates p_3.

Selection of a Cover

Prime implicant	Minterm 10	11	13	15
p_2	✓	✓		
p_4		✓		✓
p_5			✓	✓

(c) After the removal of dominated rows

Prime implicant	Minterm 10	11	13	15
p_2	✓	✓		
p_4		✓		✓
p_5			✓	✓

(c) After the removal of dominated rows

p_2 and p_5 are now essential.

Selection of a Cover

Prime implicant					0	4	8	10	11	12	13	15
p_1 =	1	0	x	0			✓	✓				
p_2 =	1	0	1	x				✓	✓			
p_3 =	1	1	0	x						✓	✓	
p_4 =	1	x	1	1					✓			✓
p_5 =	1	1	x	1							✓	✓
p_6 =	x	x	0	0	✓	✓	✓			✓		

(a) Initial prime implicant cover table

Final solution is p_2, p_5, and p_6.

Generation of Prime Implicants

List 1

0	0 0 0 0	✓
1	0 0 0 1	✓
2	0 0 1 0	✓
8	1 0 0 0	✓
5	0 1 0 1	✓
6	0 1 1 0	✓
9	1 0 0 1	✓
12	1 1 0 0	✓
7	0 1 1 1	✓
13	1 1 0 1	✓
15	1 1 1 1	✓

List 2

0,1	0 0 0 x	✓
0,2	0 0 x 0	
0,8	x 0 0 0	✓
1,5	0 x 0 1	✓
2,6	0 x 1 0	
1,9	x 0 0 1	✓
8,9	1 0 0 x	✓
8,12	1 x 0 0	✓
5,7	0 1 x 1	✓
6,7	0 1 1 x	
5,13	x 1 0 1	✓
9,13	1 x 0 1	✓
12,13	1 1 0 x	✓
7,15	x 1 1 1	✓
13,15	1 1 x 1	✓

List 3

0,1,8,9	x 0 0 x
1,5,9,13	x x 0 1
8,9,12,13	1 x 0 x
5,7,13,15	x 1 x 1

Selection of a Cover

Prime implicant	Minterm							
	0	2	5	6	7	8	9	13
p_1 = 0 0 x 0	✓	✓						
p_2 = 0 x 1 0		✓		✓				
p_3 = 0 1 1 x				✓	✓			
p_4 = x 0 0 x	✓					✓	✓	
p_5 = x x 0 1			✓				✓	✓
p_6 = 1 x 0 x						✓	✓	✓
p_7 = x 1 x 1			✓		✓			✓

(a) Initial prime implicant cover table

Prime implicant	0	2	5	6	7	8	9	13
p_1 = 0 0 x 0	✓	✓						
p_2 = 0 x 1 0		✓		✓				
p_3 = 0 1 1 x				✓	✓			
p_4 = x 0 0 x	✓					✓	✓	
p_5 = x x 0 1			✓				✓	✓
p_6 = 1 x 0 x						✓	✓	✓
p_7 = x 1 x 1			✓		✓			✓

(a) Initial prime implicant cover table

Minterm 9 dominates minterm 8.

Minterm 13 dominates minterm 5.

Selection of a Cover

Prime implicant	Minterm 0	2	5	6	7	8
p_1 = 0 0 x 0	✓	✓				
p_2 = 0 x 1 0		✓		✓		
p_3 = 0 1 1 x				✓	✓	
p_4 = x 0 0 x	✓					✓
p_5 = x x 0 1			✓			
p_6 = 1 x 0 x						✓
p_7 = x 1 x 1			✓		✓	

(b) After the removal of columns 9 and 13

Selection of a Cover

Prime implicant		0	2	5	6	7	8
p_1	= 0 0 x 0	✓	✓				
p_2	= 0 x 1 0		✓		✓		
p_3	= 0 1 1 x				✓	✓	
p_4	= x 0 0 x	✓					✓
p_5	= x x 0 1			✓			
p_6	= 1 x 0 x						✓
p_7	= x 1 x 1			✓		✓	

(b) After the removal of columns 9 and 13

Prime p_7 dominates p_5.

Prime p_4 dominates p_6.

Selection of a Cover

Prime implicant	Minterm 0	2	5	6	7	8
p_1	✓	✓				
p_2		✓		✓		
p_3				✓	✓	
p_4	✓					✓
p_7			✓		✓	

(c) After the removal of rows p_5 and p_6

Selection of a Cover

Prime implicant	Minterm 0	2	5	6	7	8
p_1	✓	✓				
p_2		✓		✓		
p_3				✓	✓	
p_4	✓					✓
p_7			✓		✓	

(c) After the removal of rows p_5 and p_6

Primes p_4 and p_7 are now essential.

Selection of a Cover

Prime implicant	Minterm 2	Minterm 6
p_1	✓	
p_2	✓	✓
p_3		✓

(d) After including p_4 and p_7 in the cover

Selection of a Cover

Prime implicant	Minterm 2	Minterm 6
p_1	✓	
p_2	✓	✓
p_3		✓

(d) After including p_4 and p_7 in the cover

Prime p_2 dominates remaining primes and becomes essential.

Selection of a Cover

Prime implicant	Minterm 0	2	5	6	7	8	9	13
p_1 = 0 0 x 0	✓	✓						
p_2 = 0 x 1 0		✓		✓				
p_3 = 0 1 1 x				✓	✓			
p_4 = x 0 0 x	✓					✓	✓	
p_5 = x x 0 1			✓				✓	✓
p_6 = 1 x 0 x						✓	✓	✓
p_7 = x 1 x 1			✓		✓			✓

(a) Initial prime implicant cover table

Final solution is p_2, p_4, and p_7.

Cyclic Cover Table Example

Prime implicant	Minterm 0	3	10	15
p_1 = 0 0 x x	✓	✓		
p_2 = x 0 x 0	✓		✓	
p_3 = x 0 1 x		✓	✓	
p_4 = x x 1 1		✓		✓
p_5 = 1 x 1 x			✓	✓

(a) Initial prime implicant cover table

Cyclic Cover Table Example

Prime implicant	Minterm 0	3	10	15
p_1 = 0 0 x x	✓	✓		
p_2 = x 0 x 0	✓		✓	
p_3 = x 0 1 x		✓	✓	
p_4 = x x 1 1		✓		✓
p_5 = 1 x 1 x			✓	✓

(a) Initial prime implicant cover table

No essentials or dominance, must use *branching*.

Let us select prime p_3.

Cyclic Cover Table Example

Prime implicant	Minterm 0	15
p_1	✓	
p_2	✓	
p_4		✓
p_5		✓

(b) After including p_3 in the cover

Cyclic Cover Table Example

Prime implicant	Minterm 0	Minterm 15
p_1	✓	
p_2	✓	
p_4		✓
p_5		✓

(b) After including p_3 in the cover

Option to include prime p_1 or p_2, AND p_4 or p_5 for 3 prime cover.
Select primes with minimum cost.

Cyclic Cover Table Example

Prime implicant	Minterm 0	3	10	15
p_1	✓	✓		
p_2	✓		✓	
p_4		✓		✓
p_5			✓	✓

(c) After excluding p_3 from the cover

Cyclic Cover Table Example

Prime implicant	Minterm 0	3	10	15
p_1	✓	✓		
p_2	✓		✓	
p_4		✓		✓
p_5			✓	✓

(c) After excluding p_3 from the cover

Minimum cost cover possible by selecting only two primes.
Either p_1 and p_5 OR p_2 and p_4

Summary of the Tabular Method

1. List all minterms where f is 1 or don't-care, generate prime implicants by successive pairwise comparisons.
2. Derive a cover table which indicates primes that cover each minterm where f is 1.
3. Select essential primes and reduce cover table by removing essential primes and covered minterms.
4. Use row and column dominance to reduce the table further.
5. Repeat steps 3 and 4 until table is empty or no reduction possible.
6. If cover table is not empty, use branching.

Heuristic Minimization Methods

- Functions seldom defined in the form of minterms, usually algebraic expressions or as sets of cubes.
- List of minterms can be very large.
- Results in numerous comparisons and computation of primes is slow.
- Solving covering tables can also be computationally intensive.
- Many heuristics have been developed to improve computation time.
- See Section 8.4.2 for one example heuristic minimization method.

Concluding Remarks

- Introduced multilevel logic synthesis.
- Presented BDD and cubical representations for logic functions which are commonly used in CAD tools for logic synthesis.
- Described a more scalable 2-level logic synthesis method.
- More details on logic synthesis in ECE/CS 5740.