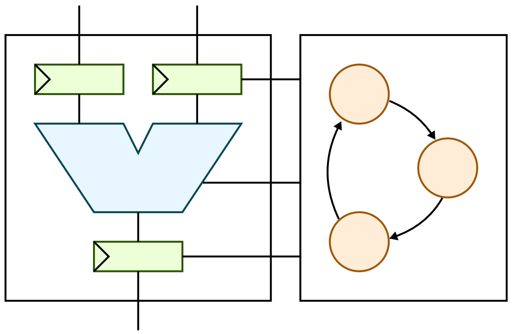
ECE 5745 Complex Digital ASIC Design
Topic 12: Synthesis Algorithms

Christopher Batten

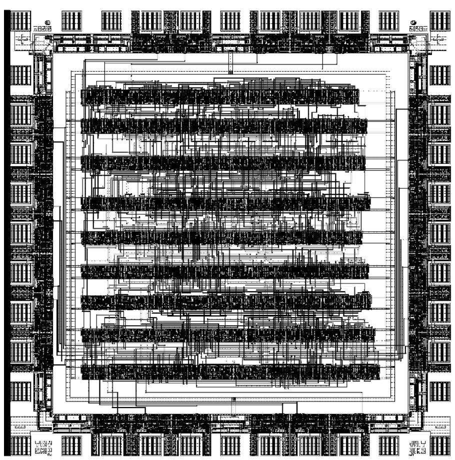
School of Electrical and Computer Engineering
Cornell University

http://www.csl.cornell.edu/courses/ece5745

Course Structure

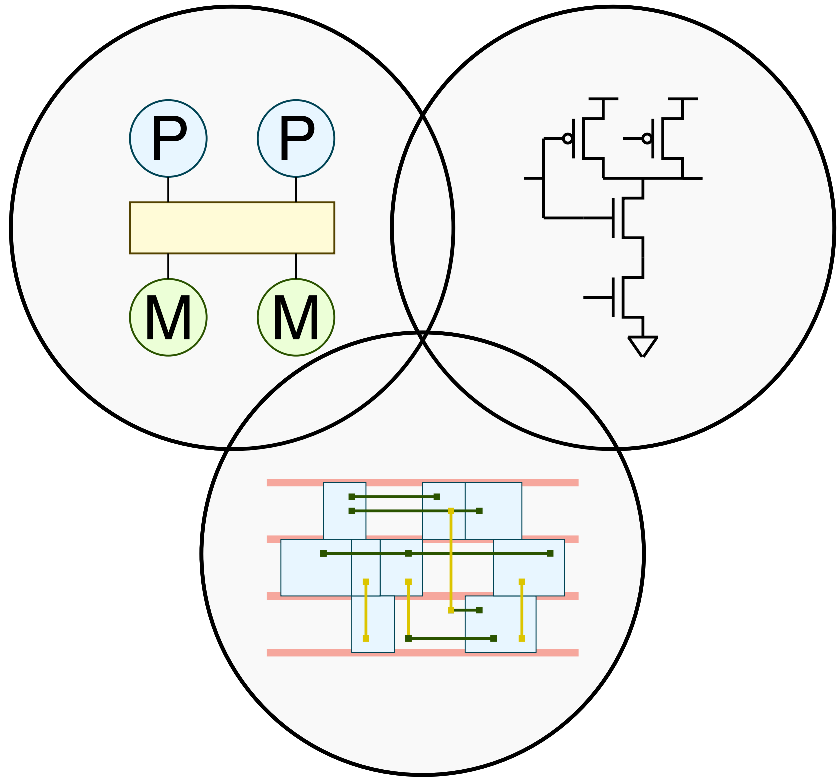


Part 1
ASIC Design Overview



Prereq
Computer Architecture

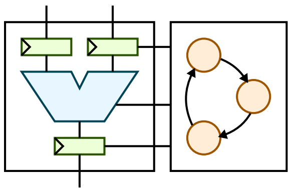
Part 2
Digital CMOS Circuits



Part 3
CAD Algorithms

Part 3: CAD Algorithms

Topic 12
Synthesis Algorithms



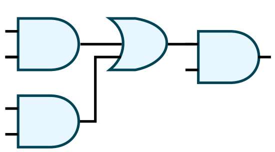
RTL to Logic Synthesis

x = a'bc + a'bc'
y = b'c' + ab' + ac

Technology Independent Synthesis

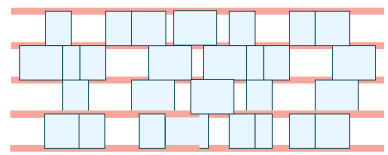
x = a'b
y = b'c' + ac

Technology Dependent Synthesis

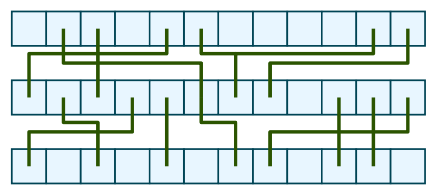


Topic 13
Physical Design Automation

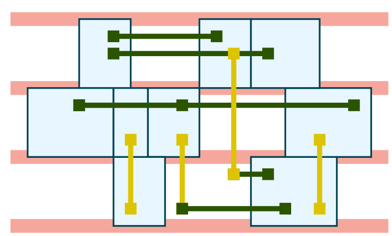
Placement



Global Routing



Detailed Routing



Step 1: Single Assignment Form

```
// 1b inputs: y, z, a, q
// 2b inputs: f
// 3b inputs: g, c, b, e

wire x = y && z;

wire [2:0] b
  = (a) ? {2'b0,x} : c;

wire [2:0] d, h;
always @(*) begin

  d = b + e;
  if ( q ) d = 3'b101;

  if ( f )
    h = 3'b0;
  else
    h = g << 1;

end
```

For each output, create exactly one assignment that is a function only of the inputs

```
wire x = y && z;

wire [2:0] b
  = (a) ? {2'b0,x} : c;

wire [2:0] d
 = (q) ? 3'b101
       : ((a) ? {2'b0,x} : c)
         + e;

wire [2:0] h
 = (f) ? 3'b0 : ( g << 1 );
```

Step 2: Bit Blast Outputs

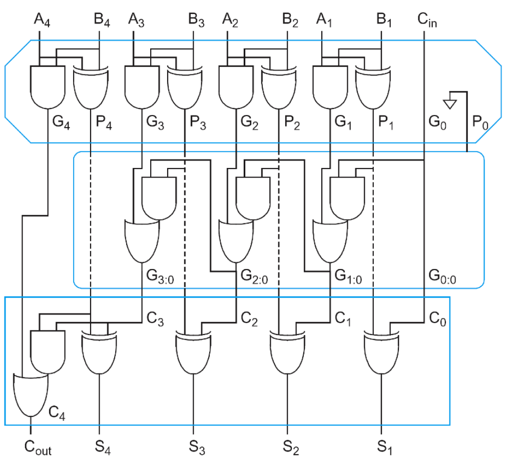
```
// 1b inputs: y, z, a, q
// 2b inputs: f
// 3b inputs: g, c, b, e

wire x = y && z;

wire [2:0] b
  = (a) ? {2'b0,x} : c;

wire [2:0] h
 = (f) ? 3'b0
         : ( g << 1 );
```

Generate separate assignment for each bit, removes arithmetic operators leaving only boolean operators (assume ternary operator is short hand for equivalent boolean operator)

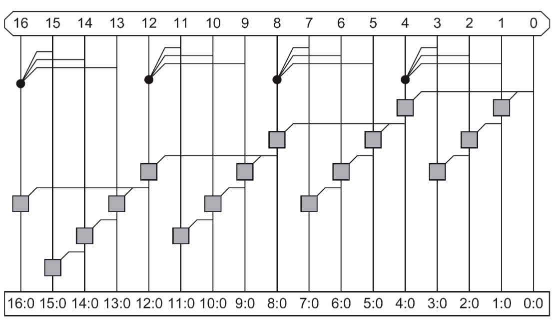
```
wire x = y && z;

wire b[0] = (a) ? x    : c[0];
wire b[1] = (a) ? 1'b0 : c[1];
wire b[2] = (a) ? 1'b0 : c[2];

wire h[0] = (f[0]|f[1]) ? 1'b0 : 1'b0;
wire h[1] = (f[0]|f[1]) ? 1'b0 : g[0];
wire h[2] = (f[0]|f[1]) ? 1'b0 : g[1];
```

RTL Datapath Synthesis

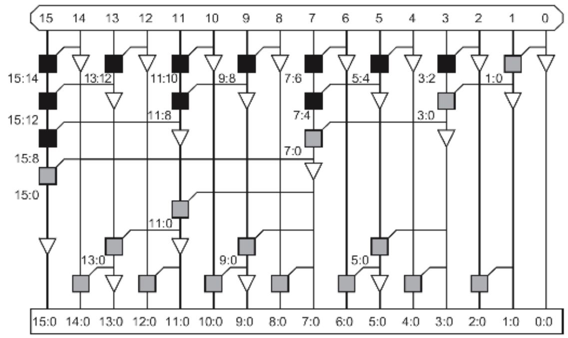
```
wire [15:0] a= b + c;
```



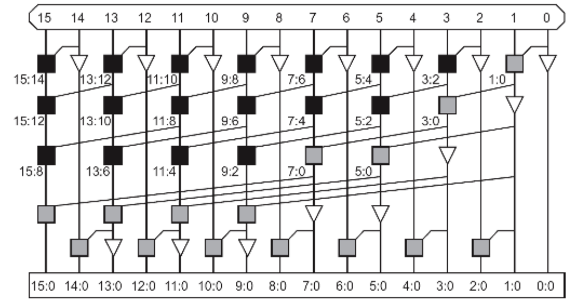
Ripple-Carry



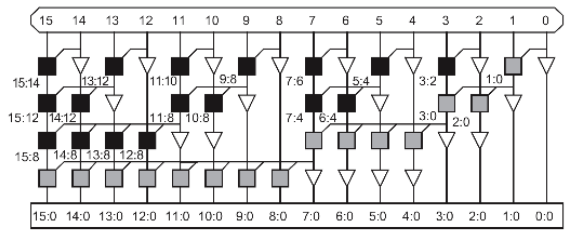
Carry Lookahead



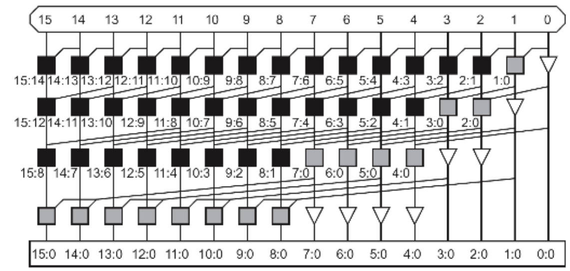
(a) Brent-Kung



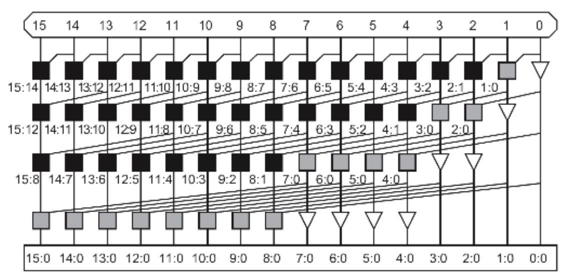
(d) Han-Carlson



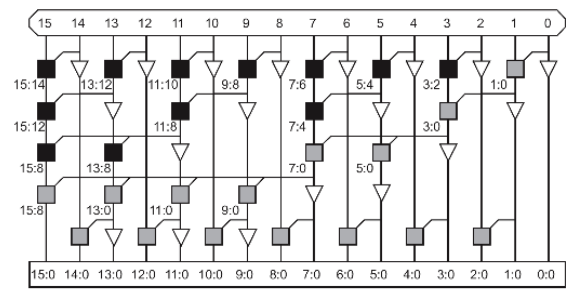
(b) Sklansky



(e) Knowles [2,1,1,1]



(c) Kogge-Stone



(f) Ladner-Fischer

Parallel-Prefix Tree-Based

Adapted from [Weste'11]

DWBB Quick Reference

DW01_add
Adder

DW01_add
Adder

- Parameterized word length
- Carry-in and carry-out signals

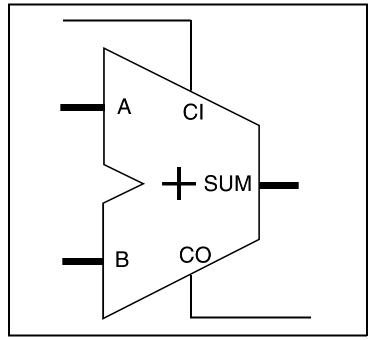


Table 1-1 Pin Description

Pin Name	Width	Direction	Function
A	*width* bit(s)	Input	Input data
B	*width* bit(s)	Input	Input data
CI	1 bit	Input	Carry-in
SUM	*width* bit(s)	Output	Sum of (A + B + CI)
CO	1 bit	Output	Carry-out

Table 1-2 Parameter Description

Parameter	Values	Description
width	≥1	Word length of A, B, and SUM

Table 1-3 Synthesis Implementations[a]

Implementation	Function	License Feature Required
rpl	Ripple-carry synthesis model	none
cla	Carry-look-ahead synthesis model	none
pparch	Delay-optimized flexible parallel-prefix	DesignWare

a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.

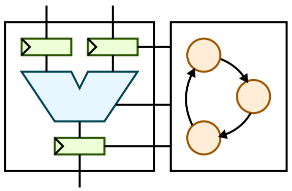
Example from Synopsys DesignWare

RTL Datapath Synthesis directly transforms arithmetic operators into technology independent optimized gate-level netlists

Adapted from [Synopsys'11]

Part 3: CAD Algorithms

Topic 12
Synthesis Algorithms



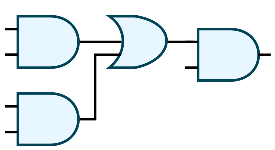
RTL to Logic Synthesis

x = a'bc + a'bc'
y = b'c' + ab' + ac

Technology Independent Synthesis

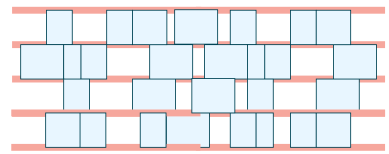
x = a'b
y = b'c' + ac

Technology Dependent Synthesis

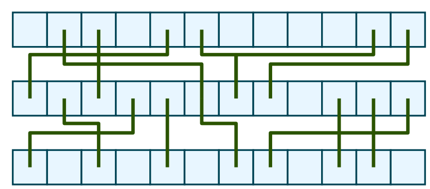


Topic 13
Physical Design Automation

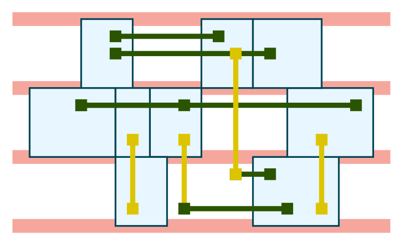
Placement



Global Routing



Detailed Routing



Technology-Independent Synthesis

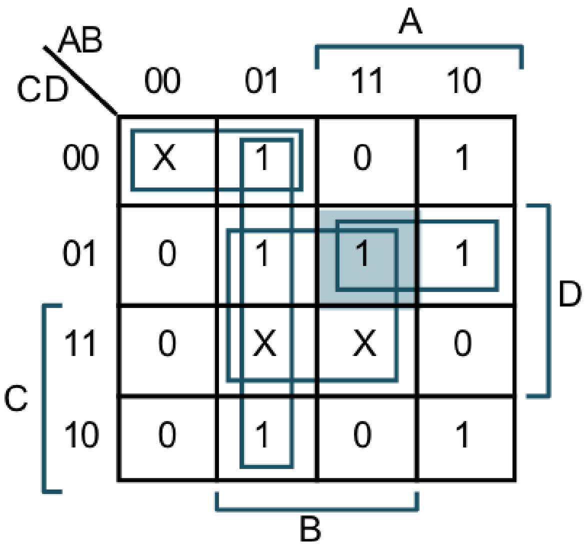
- **Two-level boolean minimization –** based on assumption that reducing the number of product terms in an equation and reducing the size of each product term will result in a smaller/faster implementation
- **Optimizing finite-state machines –** look for equivalent FSMs (i.e., FSMs that produce the same outputs give the same sequence of inputs) that have fewer states
- **FSM state encodings –** minimize implementation area (= size of state storage + size of logic to implement next state and output functions).

Note that none of these optimizations are completely isolated from the target technology, but experience has shown that it's advantageous to reduce the size of the problem as much as possible before starting the technology-dependent optimizations.

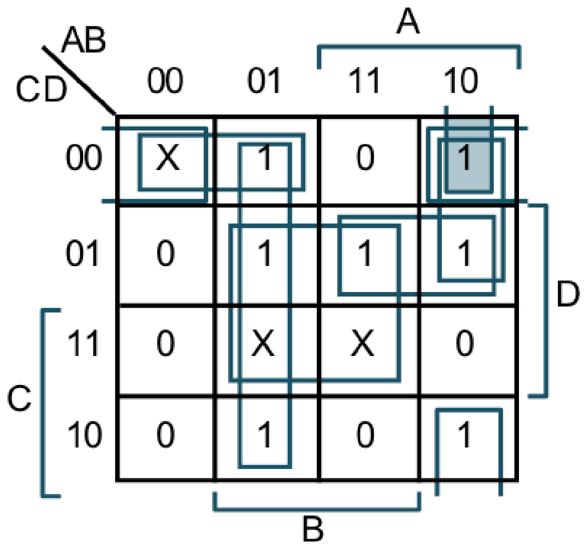
Karnaugh Map Method Review

- 1. Choose an element of ON-set not already covered by an implicant
- 2. Find "maximal" groups of 1's and X's adjacent to that element. Remember to consider top/bottom row, left/right column, and corner adjacencies. This forms prime implicants.
- Repeat steps 1 and 2 to find all prime implicants
- 3. Revise the 1's elements in the K-map. If covered by single prime implicant, it is *essential*, and participates in the final cover. The 1's it covers do not need to be revisited.
- 4. If there remain 1's not covered by essential prime implicants, then select the smallest number of prime implicants that cover the remaining 1's

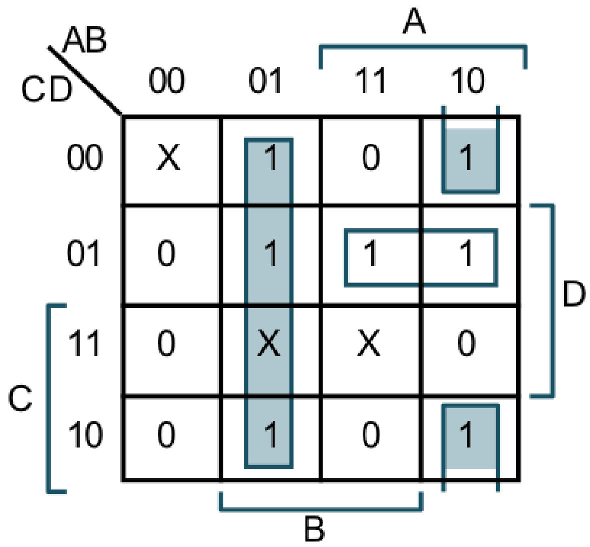
Karnaugh Map Method Example



Primes around A B C' D



Primes around A B' C' D'



Essential Primes with Min Cover

Adapted from [Zhou'02]

Quine-McCluskey (QM) Method

- Quine-McCluskey method is an exact alogorithm which finds a minimum-cost sum-of-products implementation of a boolean function
- Four main steps
  - 1. Generate prime implicants
  - 2. Construct prime implicant table
  - 3. Reduce prime implicant table
    - a. Remove essential prime implicants
    - b. Row dominance
    - c. Column dominance
    - d. Iterate at this step until no further reductions
  - 4. Solve prime implicant table

QM Example #1 – Step 1

Start by expressing your Boolean function using 0-terms (product terms with no don't care care entries). For compactness the table for example 4-input, 1-output function F(w,x,y,z) shown to the right includes only entries where the output of the function is 1 and we've labeled each entry with it's decimal equivalent.

W	*X*	*Y*	*Z*	*label*
0	0	0	0	0
0	1	0	1	5
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	1	0	14
1	1	1	1	15

Look for pairs of 0-terms that differ in only one bit position and merge them in a 1-term (i.e., a term that has exactly one '–' entry). Next 1-terms are examined in pairs to see if the can be merged into 2-terms, etc. Mark k-terms that get merged into (k+1) terms so we can discard them later.

1-terms:

```
 0, 8   -000 [A]
 5, 7   01-1 [B]
 7,15   -111 [C]
 8, 9   100-
 8,10   10-0
 9,11   10-1
10,11   101-
10,14   1-10
11,15   1-11
14,15   111-
```

2-terms:

```
 8, 9,10,11   10-- [D]
10,11,14,15   1-1- [E]
```

3-terms: none!

Label unmerged terms: these terms are prime!

Example due to Srini Devadas

Adapted from [Terman'02]

QM Example #1 – Step 2, Step 3a

An "X" in the prime term table in row R and column C signifies that the 0-term corresponding to row R is contained by the prime corresponding to column C.

Goal: select the minimum set of primes (columns) such that there is at least one "X" in every row. This is the classical minimum covering problem.

	A	B	C	D	E	
0000	X	.	.	.	.	→ A is essential
0101	.	X	.	.	.	→ B is essential
0111	.	X	X	.	.	
1000	X	.	.	X	.	
1001	.	.	.	X	.	→ D is essential
1010	.	.	.	X	X	
1011	.	.	.	X	X	
1110	.	.	.	.	X	→ E is essential
1111	.	.	X	.	X	

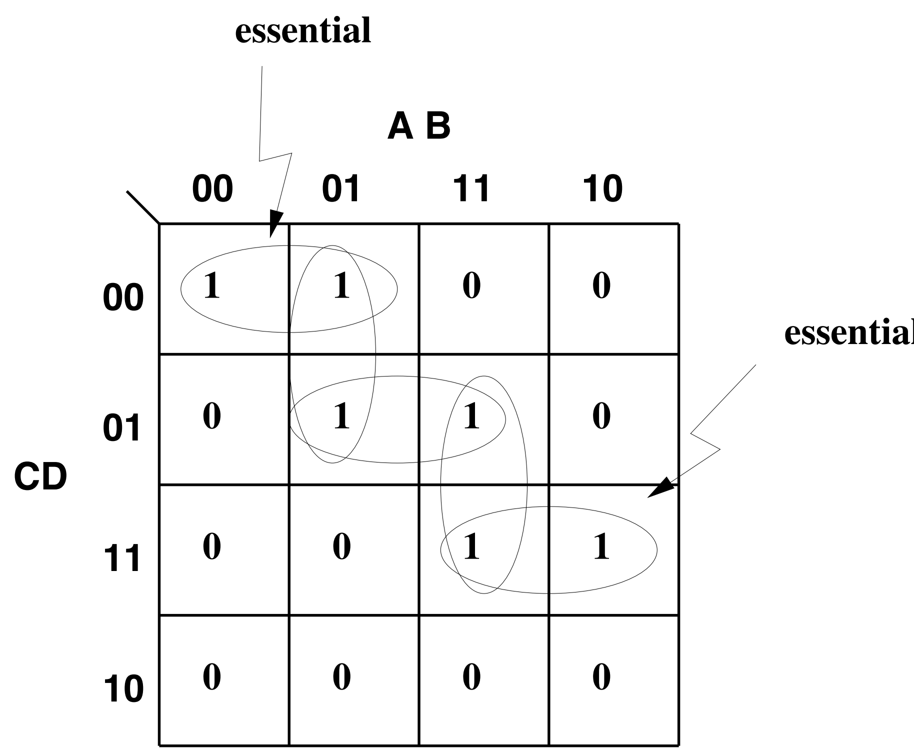
Each row with a single X signifies an essential prime term since any prime implementation will have to include that prime term because the corresponding 0-term is not contained in any other prime.

In this example the essential primes "cover" all the 0-terms.

Adapted from [Terman'02]

Column Dominance

- 5 prime implicants, each covers 2 ON-set minterms
- A'C'D' and ACD are essential prime implicants, must be in final cover
- Pick min subset of remaining 3 prime implicants which covers ON-set



Karnaugh map with set of prime implicants: illustrating "column dominance"

Adapted from [Nowick'12]

Column Dominance

- Cross out columns A'C'D' and ACD since they are essential
- Each row intersected by one of the essential prime columns is also crossed out because that minterm is already covered

	$A'C'D'$ (0,4)	$A'BC'$ (4,5)	$BC'D$ (5,13)	ABD (13,15)	ACD (11,15)
0	X				
4	X	X			
5		X	X		
11					X
13			X	X	
15				X	X

- BC'D covers minterm 5 and 13, but A'BC' only covers minterm 5 and ABD only covers minterm 13
- BC'D *column dominates* A'BC' and ABD, dominated prime implicants can be crossed out
- Only column BC'D remains

Adapted from [Nowick'12]

Row Dominance

- 4 prime implicants, no essential prime implicants
- Pick min subset of the 4 prime implicants to cover the 5 ON-set minterms

CD \ AB	00	01	11	10
00	–	0	0	0
01	1	1	–	–
11	1	1	0	0
10	1	–	0	0

Adapted from [Nowick'12]

Row Dominance

- Row 3 is contained in 3 columns: A’B’, A’D, and A’C
- Row 2 is covered by two of these three columns, so any prime implicant which contains row 2 also contains row 3

	$A'B'$ (1,2,3)	$C'D$ (1,5)	$A'D$ (1,3,5,7)	$A'C$ (2,3,7)
1	X	X	X	
2	X			X
3	X		X	X
5		X	X	
7			X	X

- Row 7 is covered by two of these three columns, so any prime implicant which contains row 7 also contains row 3
- Thus we can ignore row 3, it will always be covered as long as cover row 2 or row 7
- Cross out row 3, similarly row 1 dominates row 5 so cross out row 1

Adapted from [Nowick’12]

QM Example #2 – Step 1

Column I

0	0000	√
2	0010	√
8	1000	√
5	0101	√
6	0110	√
10	1010	√
12	1100	√
7	0111	√
13	1101	√
14	1110	√
15	1111	√

Column II

(0,2)	00-0	√
(0,8)	-000	√
(2,6)	0-10	√
(2,10)	-010	√
(8,10)	10-0	√
(8,12)	1-00	√
(5,7)	01-1	√
(5,13)	-101	√
(6,7)	011-	√
(6,14)	-110	√
(10,14)	1-10	√
(12,13)	110-	√
(12,14)	11-0	√
(7,15)	-111	√
(13,15)	11-1	√
(14,15)	111-	√

Column III

(0,2,8,10)	-0-0
(0,8,2,10)	-0-0
(2,6,10,14)	–10
(2,10,6,14)	–10
(8,10,12,14)	1–0
(8,12,10,14)	1–0
(5,7,13,15)	-1-1
(5,13,7,15)	-1-1
(6,7,14,15)	-11-
(6,14,7,15)	-11-
(12,13,14,15)	11–
(12,14,13,15)	11–

Eliminate redundant entries in table

Adapted from [Nowick'12]

QM Example #2 – Step 2, Step 3a

	$B'D'(*)$ (0,2,8,10)	CD' (2,6,10,14)	$BD(*)$ (5,7,13,15)	BC (6,7,14,15)	AD' (8,10,12,14)	AB (12,13,14,15)
(○)0	X					
2	X	X				
(○)5			X			
6		X		X		
7			X	X		
8	X				X	
10	X	X			X	
12					X	X
13			X			X
14		X		X	X	X
15			X	X		X

* indicates an essential prime implicant

○ indicates a distinguished row, i.e. a row covered by only 1 prime implicant

Adapted from [Nowick'12]

QM Example #2 – Step 3b, 3c, 3a

	CD' (2,6,10,14)	BC (6,7,14,15)	AD' (8,10,12,14)	AB (12,13,14,15)
6	X	X		
12			X	X
14	X	X	X	X

	CD' (2,6,10,14)	BC (6,7,14,15)	AD' (8,10,12,14)	AB (12,13,14,15)
6	X	X		
12			X	X

	$CD'(**)$ (2,6,10,14)	$AD'(**)$ (8,10,12,14)
(○)6	X	
(○)12		X

3.b Row Dominance – row 14 dominates both row 6 and 12; remove row 14 since if some product covers row 6, row 14 is guaranteed to be covered

3.c Column Dominance – column CD’ dominates column BC (actually they co-dominate each other); remove column BC since it is redundant with column CD’

3.a Remove Essential Prime Implicants – second iteration of step 3, both remaining prime implicants are essential

F = B’D’ + BD + CD’ + AD’

Adapted from [Nowick’12]

QM Example #3 – Step 2

$$F(A, B, C, D) = \Sigma m(0, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13)$$

	$A'D'$	$B'D'$	$C'D'$	$A'C$	$B'C$	$A'B$	BC'	AB'	AC'
0	X	X	X						
2	X	X		X	X				
3				X	X				
4	X		X			X	X		
5						X	X		
6	X			X		X			
7				X		X			
8		X	X					X	X
9								X	X
10		X			X			X	
11					X			X	
12			X				X		X
13							X		X

Adapted from [Nowick'12]

QM Example #3 – Step 3a, 3b, 3c, 3a

- 3a. No essential prime implicants
- 3b. Row dominance: 2>3, 4>5, 6>7, 8>9, 10>11, 12>13

	$A'D'$	$B'D'$	$C'D'$	$A'C$	$B'C$	$A'B$	BC'	AB'	AC'
0	X	X	X						
3				X	X				
5						X	X		
7				X		X			
9								X	X
11					X			X	
13							X		X

	$A'D'(**)$	$A'C$	$B'C$	$A'B$	BC'	AB'	AC'
$(\circ)$0	X						
3		X	X				
5				X	X		
7		X		X			
9						X	X
11			X			X	
13					X		X

** indicates a secondary essential prime implicant

$\circ$ indicates a distinguished row

3.c Column Dominance – column A'D', B'D', and C'D' dominate each other; remove any two of them

3.a Remove Essential Prime Implicants – second iteration of step 3, remove A'D'

Adapted from [Nowick'12]

QM Example #3 – Step 4

	$A'C$	$B'C$	$A'B$	BC'	AB'	AC'
3	X	X				
5			X	X		
7	X		X			
9					X	X
11		X			X	
13				X		X

No essential prime implicants; no row dominance; no column dominance
Cyclic covering problem – can be solved using a branch-and-bound search technique (see notes for details)

Adapted from [Nowick'12]

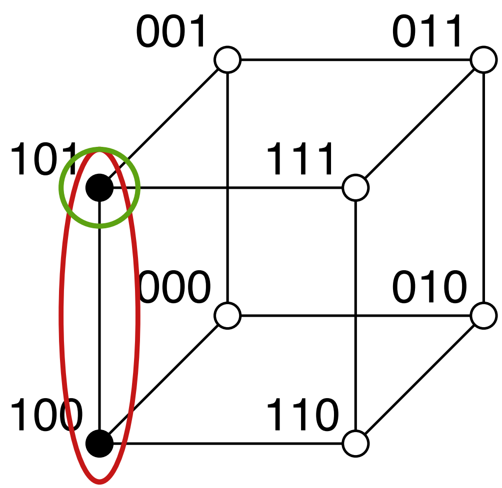
Heuristic Espresso Method

- Quine-McCluskey
  - Number of prime implicants grows rapidly with number of inputs
  - Finding a minimum cover is NP-complete (i.e., computationally expensive)
- Espresso
  - Don't generate all prime implicants (i.e., QM step 1)
  - Carefully select a subset of primes that still covers ON-set
  - Heuristically explore space of covers
  - Similar in spirit (but more structured) to finding primes in K-map

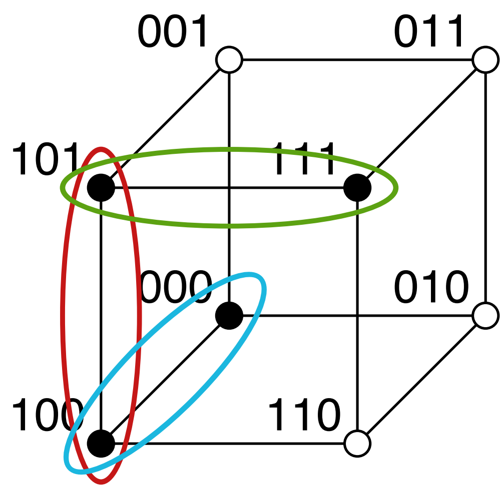
Adapted from [Zhou'02]

Redundancy in Boolean Space

- Every point in boolean space is an assignment of values to variables
- Redundancy involves inclusion or covering in boolean space
- Irredundant cover has no redundancy



g = AB'C, h = AB'
g is redundant



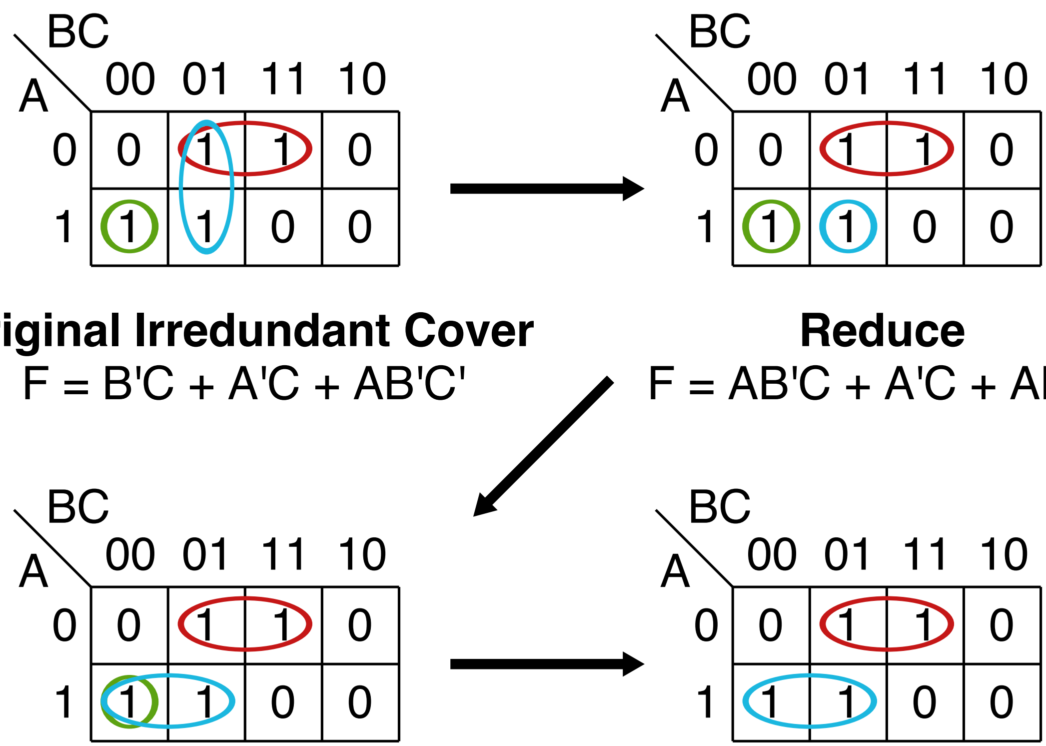
f = AC, g = B'C', h = AB'
h is redundant

Can exhaustively check if each prime is redundant with any other prime

Adapted from [Zhou'02]

Irredundant Covers vs. Minimal Covers

- Irredundant cover is not necessarily a minimal cover
- Can use reduce, expand, remove redundancy operations to explore space of irredundant covers



Original Irredundant Cover
F = B'C + A'C + AB'C'

Reduce
F = AB'C + A'C + AB'C'

Expand
F = AB' + A'C + AB'C'

New Irredundant Cover
F = AB' + A'C

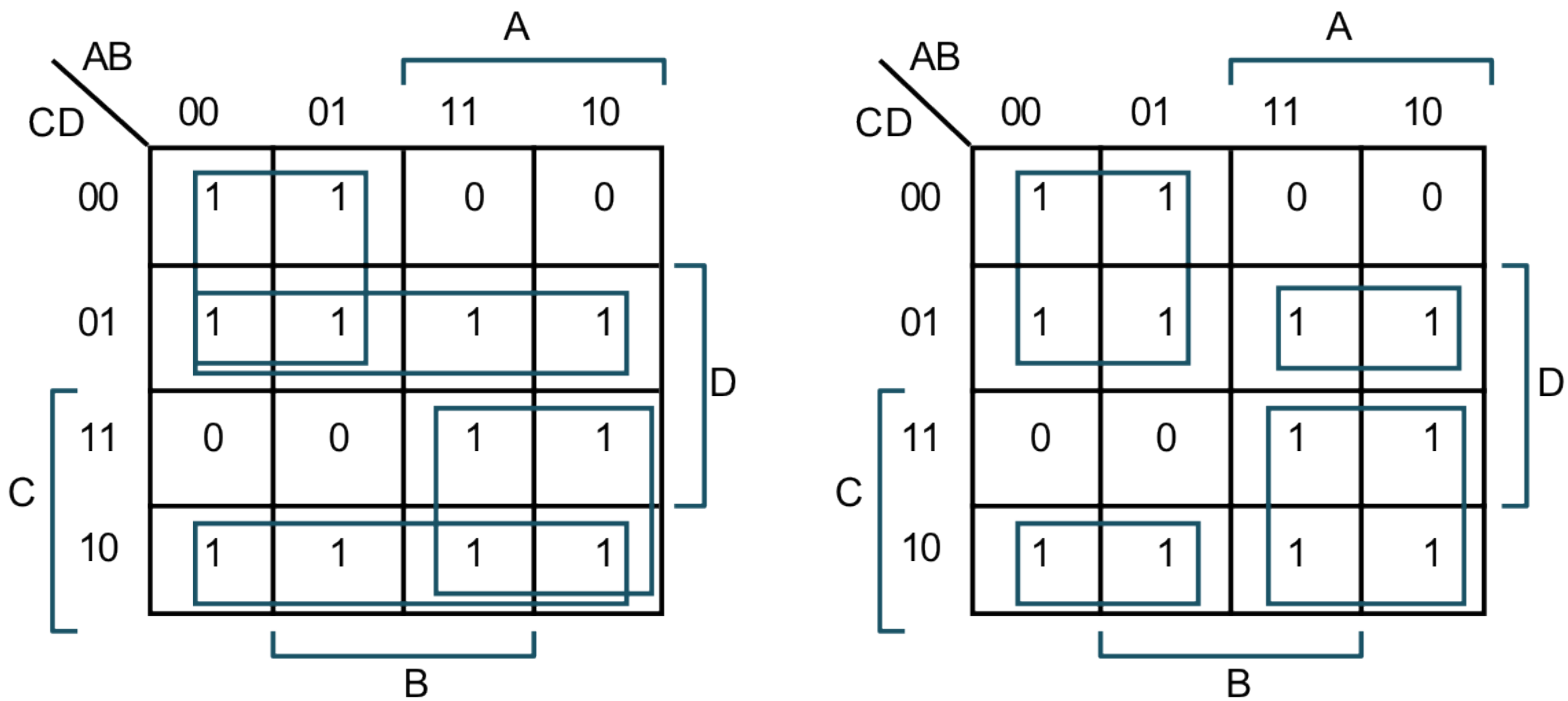
Adapted from [Zhou'02]

Espresso Algorithm

- 1. EXPAND implicants (choice of which implicants requires heuristic)
- 2. REMOVE-REDUNDANCY
- 3. REDUCE implicants (choice of which implicants requires heuristic)
- 4. EXPAND implicants (choice of which implicants requires heuristic)
- 5. REMOVE-REDUNDANCY
- 6. Goto step 3

Adapted from [Zhou'02]

Espresso Example



Initial Set of Primes found by Steps1 and 2 of the Espresso Method

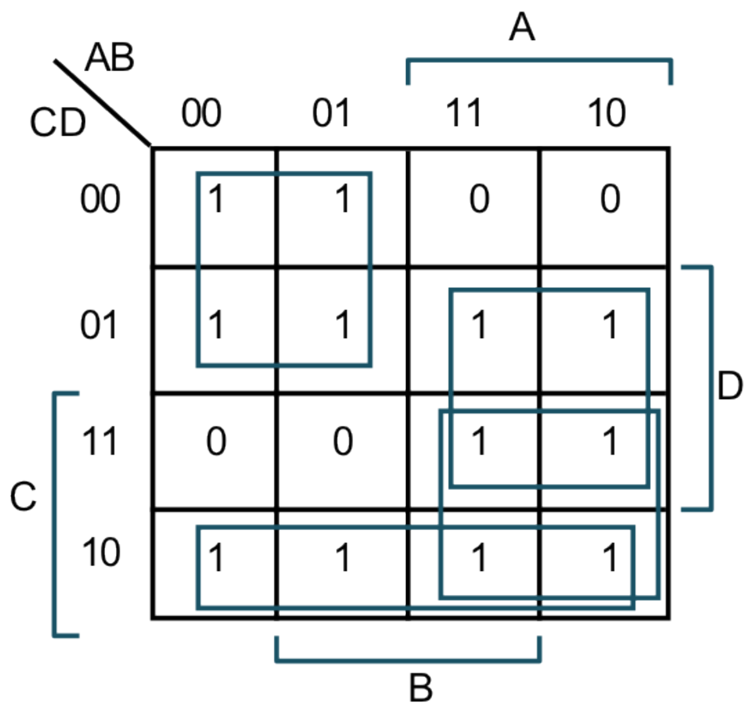
4 primes, irredundant cover, but not a minimal cover!

Result of REDUCE: Shrink primes while still covering the ON-set

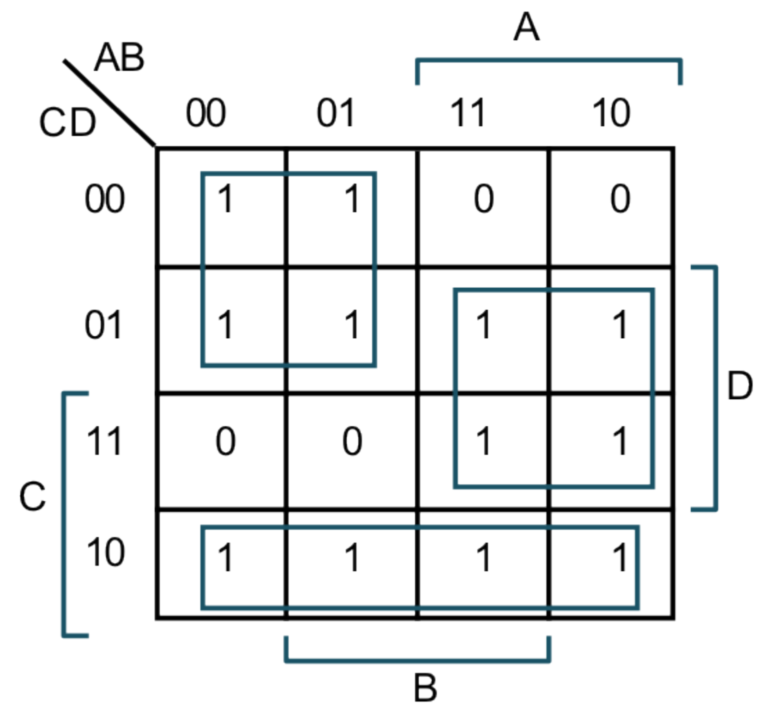
Choice of order in which to perform shrink is important

Adapted from [Zhou'02]

Espresso Example



Second EXPAND generates a different set of prime implicants



REMOVE-REDUNDANCY generates only three prime implicants!

Adapted from [Zhou'02]

Two-Level Logic vs. Multi-Level Logic

2-Level:

$$f_1 = AB + AC + AD$$
$$f_2 = \overline{A}B + \overline{A}C + \overline{A}E$$

6 product terms which cannot be shared.

24 transistors in static CMOS

Multi-level:

Note that $B + C$ is a common term in f_1 and f_2

$$K = B + C$$
$$f_1 = AK + AD$$
$$f_2 = \overline{A}K + \overline{A}E$$

3 Levels

20 transistors in static CMOS not counting inverters

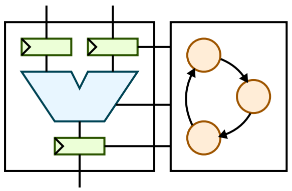
Adapted from [Devadas'06]

Two-Level Logic vs. Multi-Level Logic

- Two-Level Logic
  - At most two gates between primary input and primary output
  - Real life circuits: programmable logic arrays
  - Exact optimization methods: well-developed, feasible
  - Heuristic methods also possible
- Multi-Level Logic
  - Any number of gates between primary input and primary output
  - Most circuits in real life are multi-level
  - Smaller, less power, and (in many cases) faster
  - Exact optimization methods: few, high complexity, impractical
  - Heuristic methods pretty much required

Part 3: CAD Algorithms

Topic 12
Synthesis Algorithms



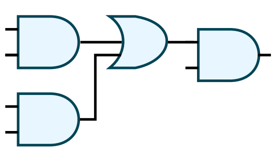
RTL to Logic Synthesis

x = a'bc + a'bc'
y = b'c' + ab' + ac

Technology Independent Synthesis

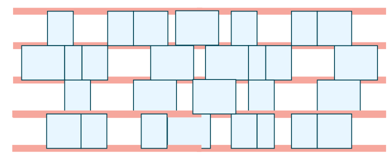
x = a'b
y = b'c' + ac

Technology Dependent Synthesis

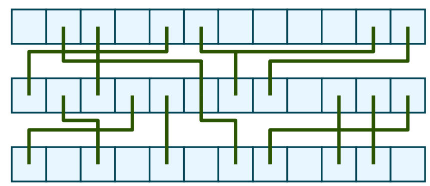


Topic 13
Physical Design Automation

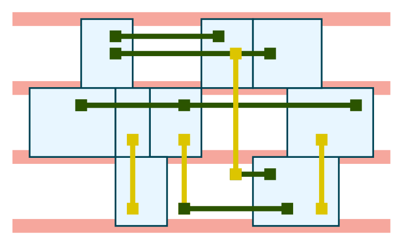
Placement



Global Routing

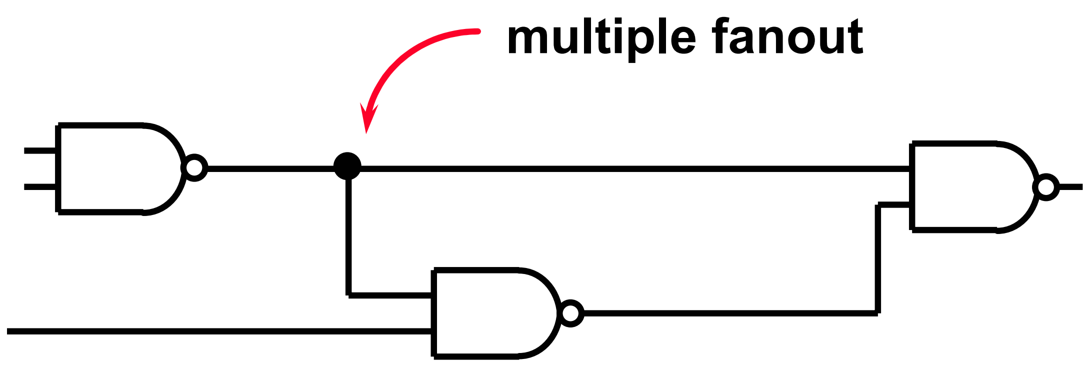


Detailed Routing



Technology Mapping

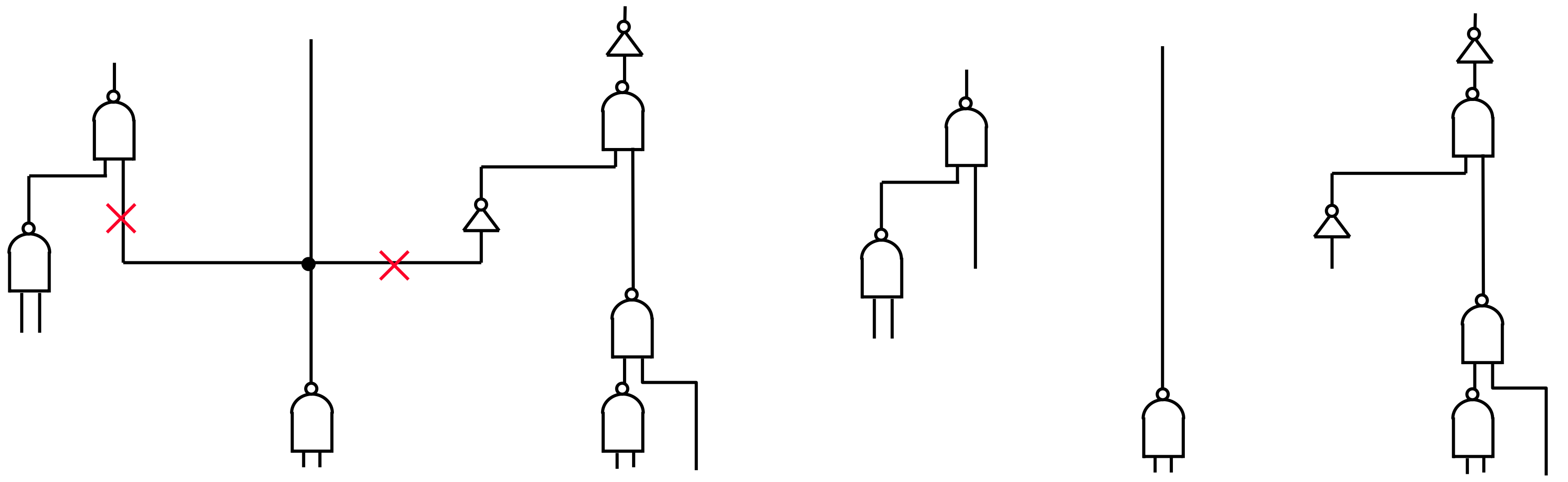
- Once minimized logic equations, next step is to map each equation to the gates in the target standard cell library
- Classic approach uses DAG covering (K. Keutzer)
  - "Normal Form" : use basic gates (e.g., 2-input NAND gates, inverters)
  - Represent logic equations as input netlist in normal form (subjective DAG)
  - Represent each library gate in normal form (primitive DAG)
  - GOAL: Find a min cost covering of subjective DAG by primitive DAGs
- Sound algorithmic approach, but is NP-hard optimization problem



Adapted from [Devadas'06]

Tree Heuristic Transformation

If subject and primitive DAGs are trees, efficient algorithm can find optimum cover in linear time via dynamic programming, so use tree covering heuristic approach by partitioning graph into subtrees

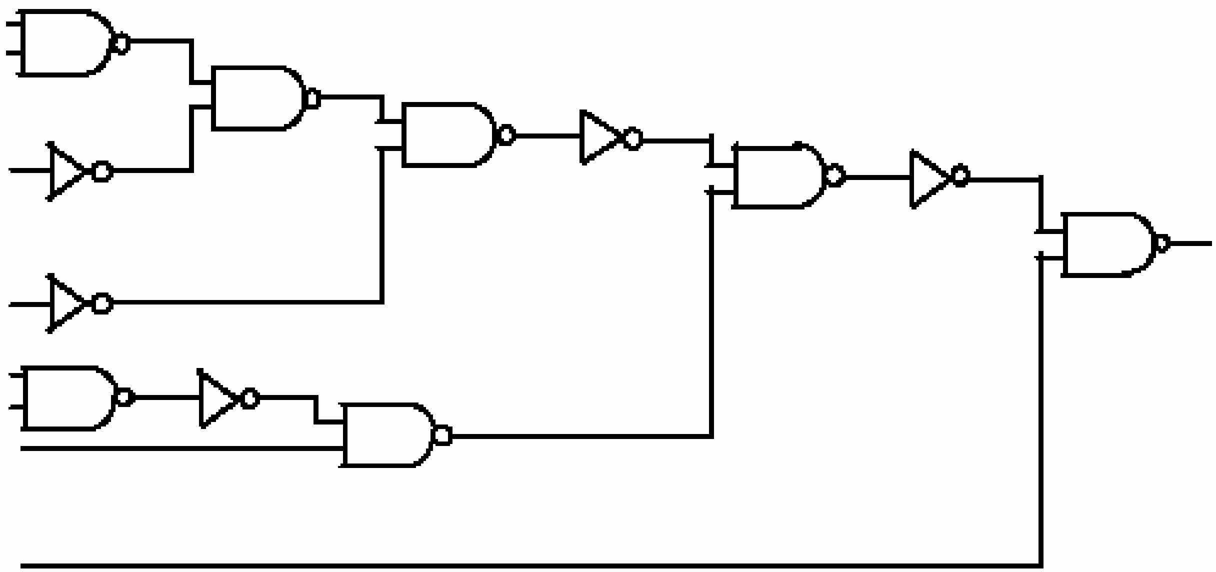


Original Graph **Partitioned Graph**

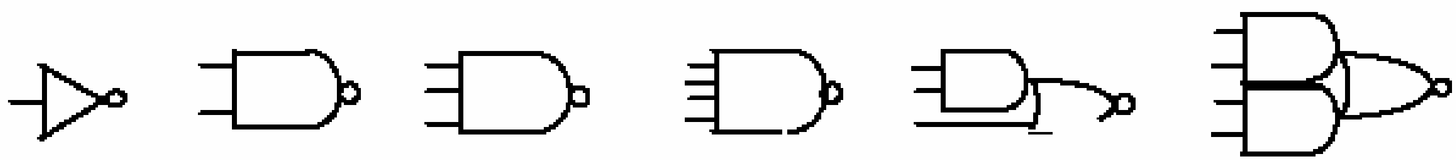
Adapted from [Devadas'06]

Technology Mapping Example

Problem statement: find an "optimal" mapping of this circuit:

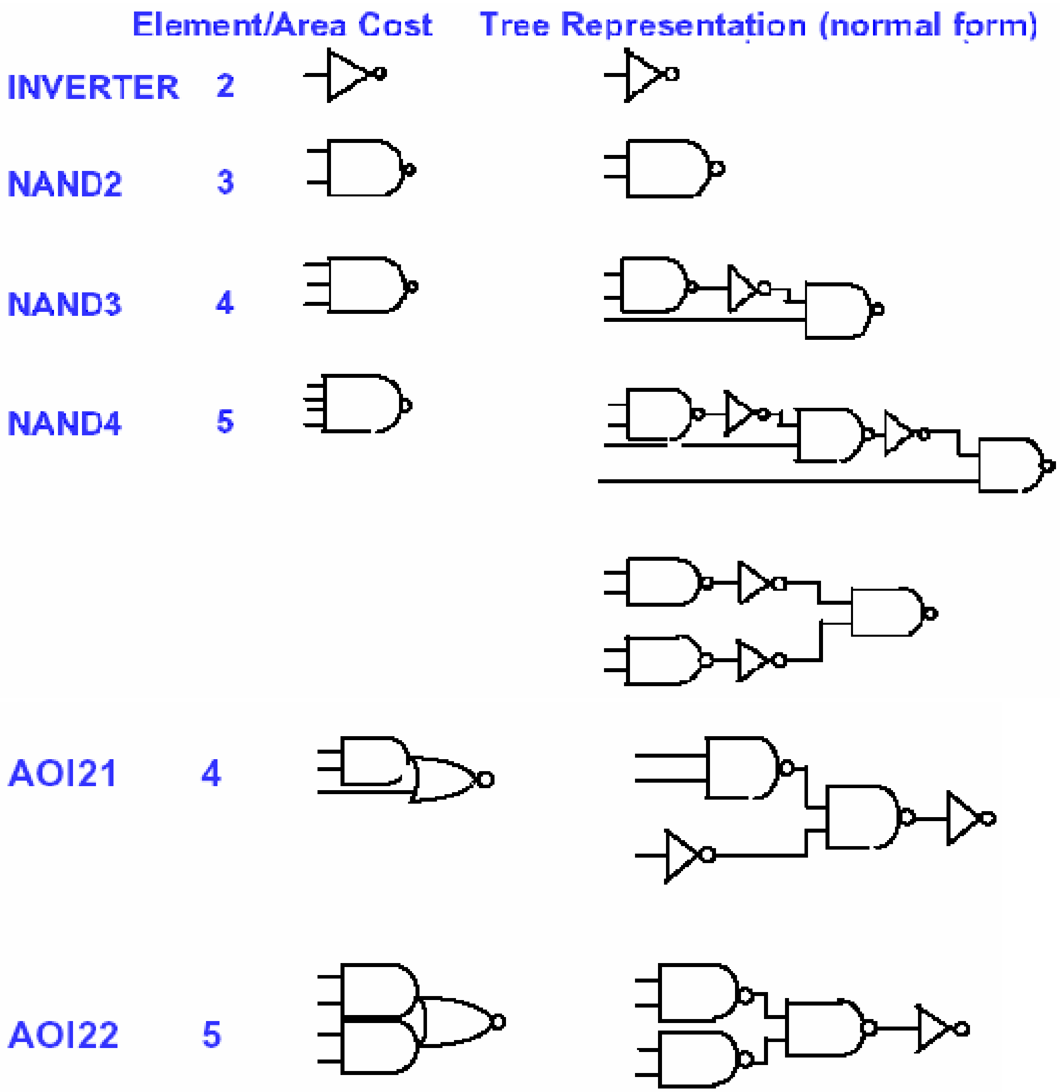


Into this library:



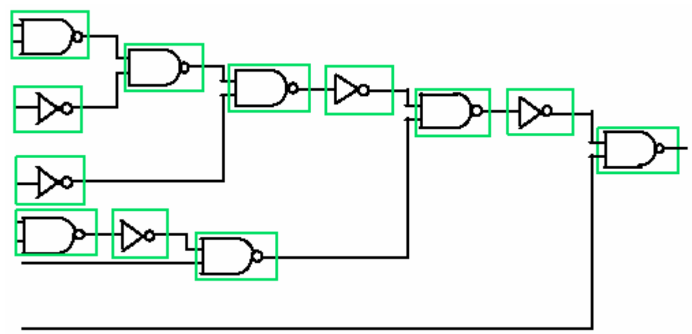
Adapted from [Terman'02,Devadas'06]

Primitive DAGs for Standard-Cell Library Gates



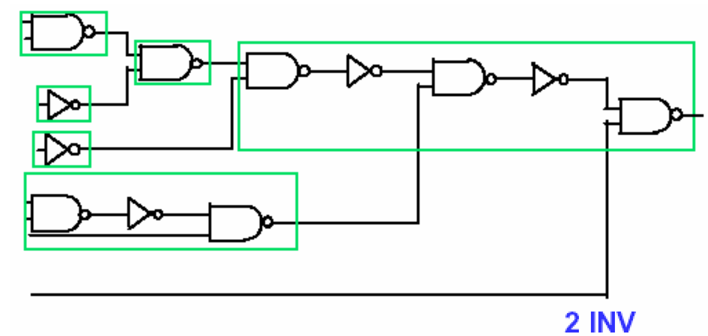
Adapted from [Terman'02,Devadas'06]

Possible Covers



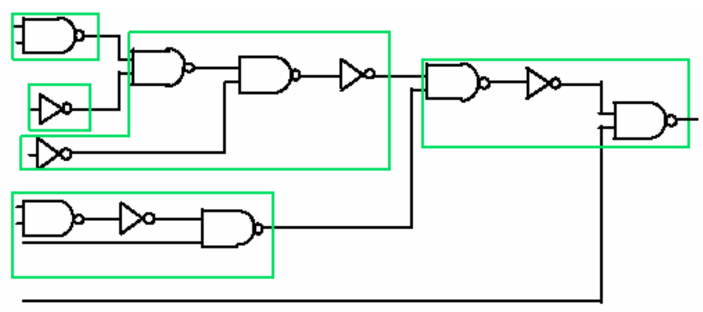
7 NAND2 (3) = 21
5 INV (2) = 10

Area cost 31



2 INV = 4
2 NAND2 = 6
1 NAND3 = 4
1 NAND4 = 5

Area cost 19



1 INV = 2
1 NAND2 = 3
2 NAND3 = 8
1 AOI21 = 4

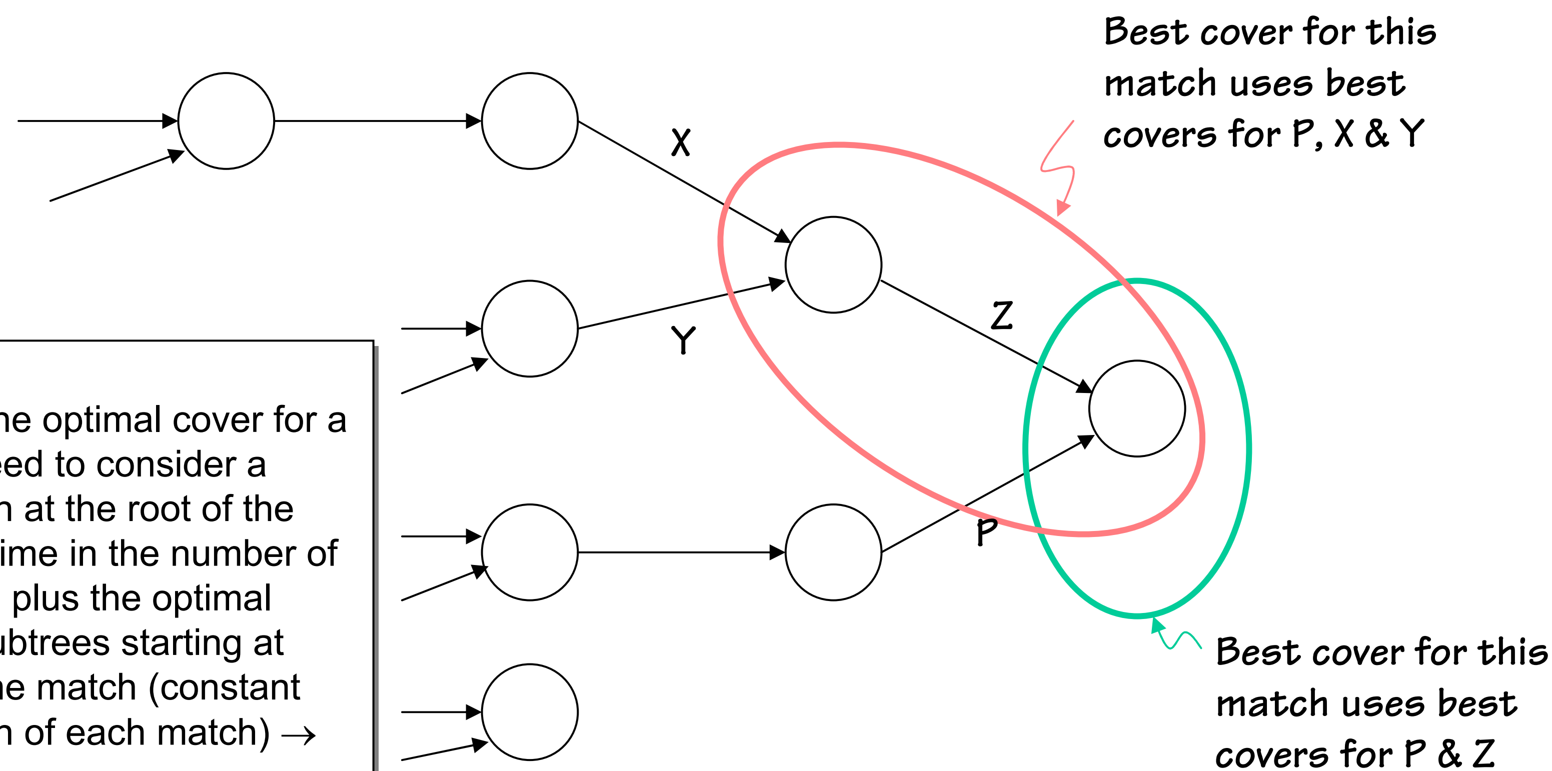
Area Cost 17

Hmmm. Seems promising but is there a systematic and efficient way to arrive at the optimal answer?

Adapted from [Terman'02,Devadas'06]

Use Dynamic Programming!

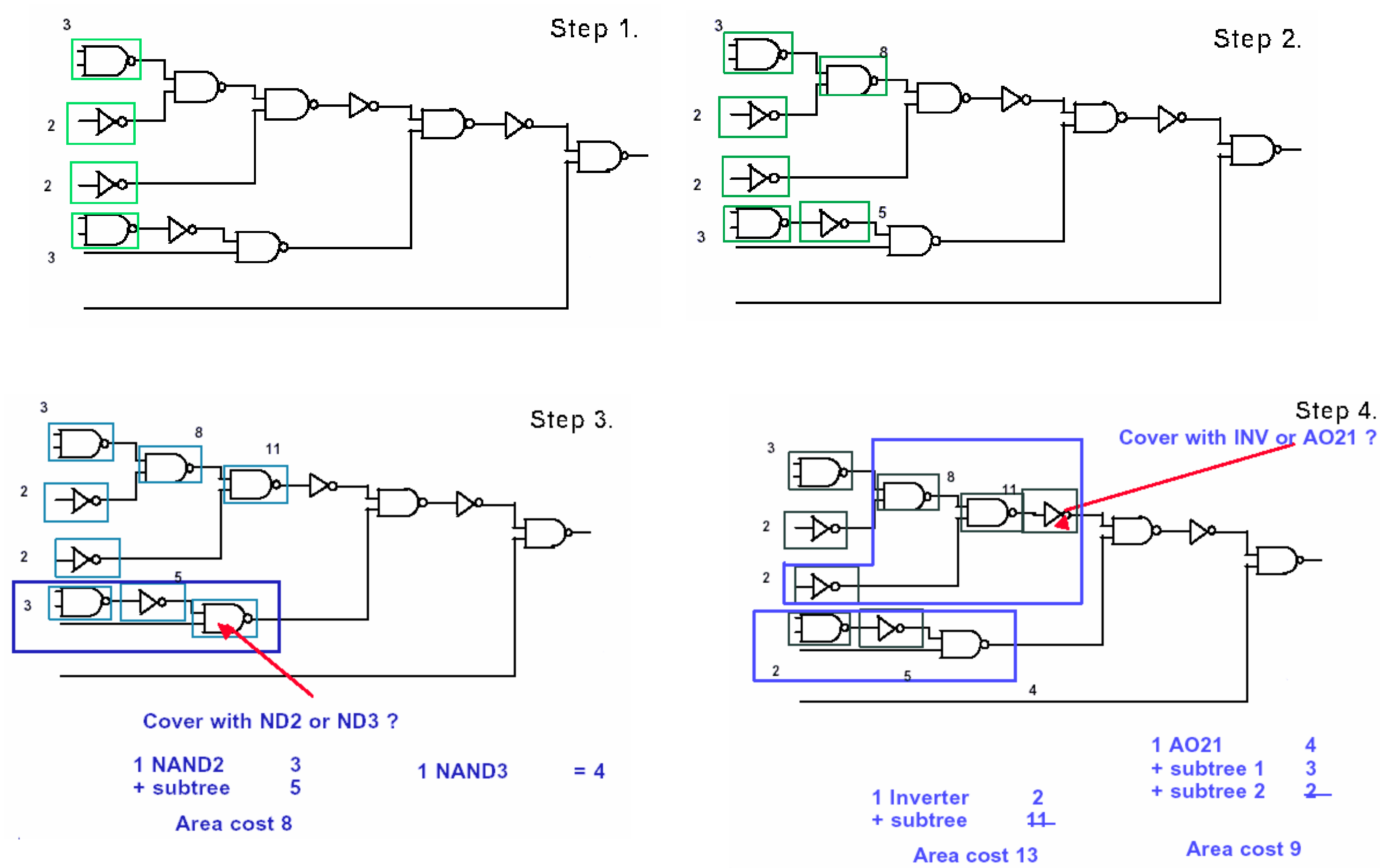
Principle of optimality: Optimal cover for a tree consists of a best match at the root of the tree plus the optimal cover for the sub-trees starting at each input of the match.



Complexity:
To determine the optimal cover for a tree we only need to consider a best-cost match at the root of the tree (constant time in the number of matched cells), plus the optimal cover for the subtrees starting at each input to the match (constant time in the fanin of each match) → O(N)

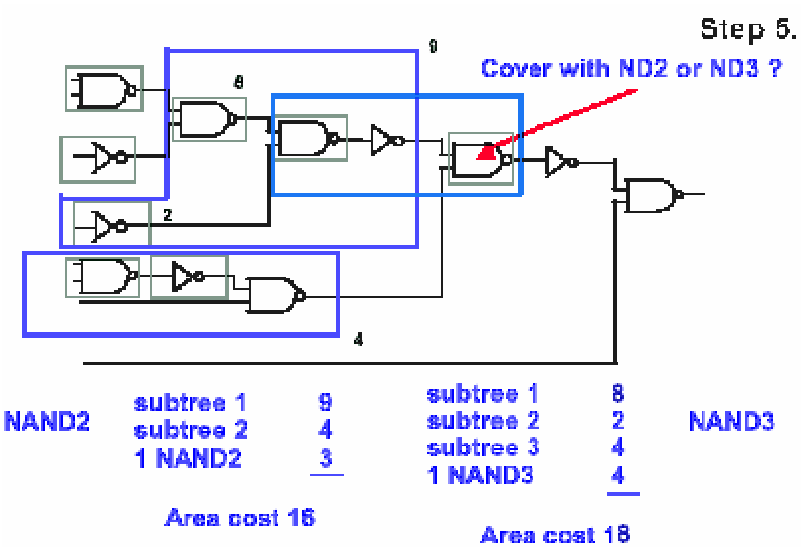
Adapted from [Terman'02,Devadas'06]

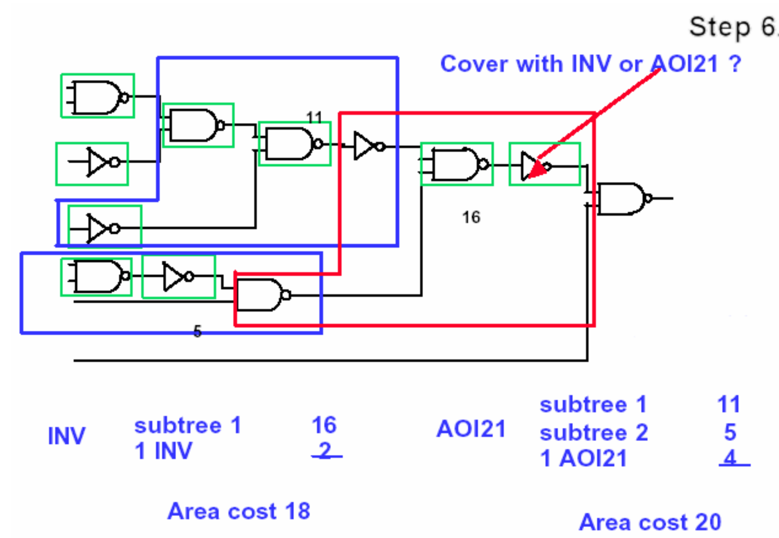
Optimal Tree Covering Example

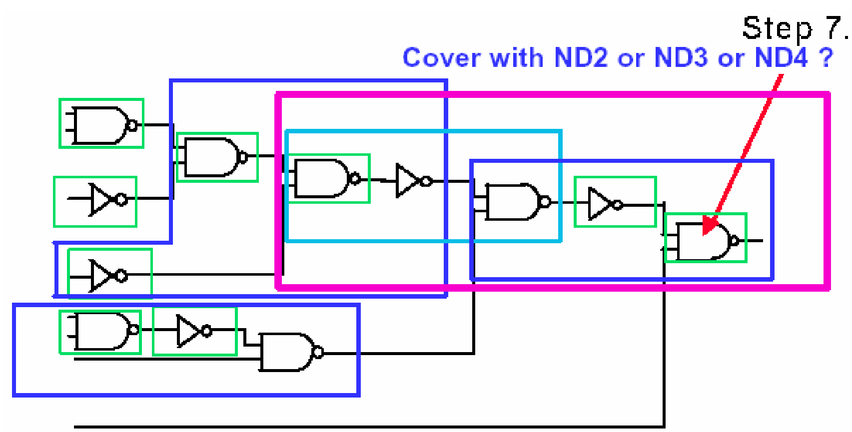


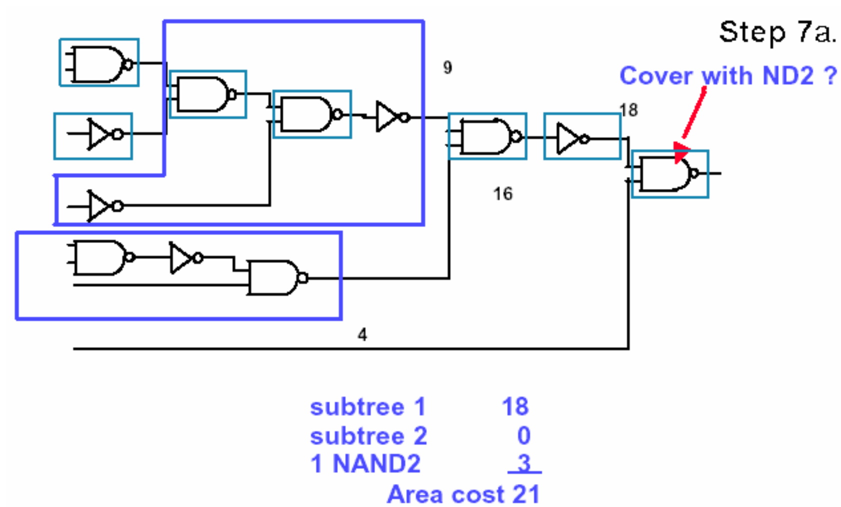
Adapted from [Terman'02,Devadas'06]

Optimal Tree Covering Example



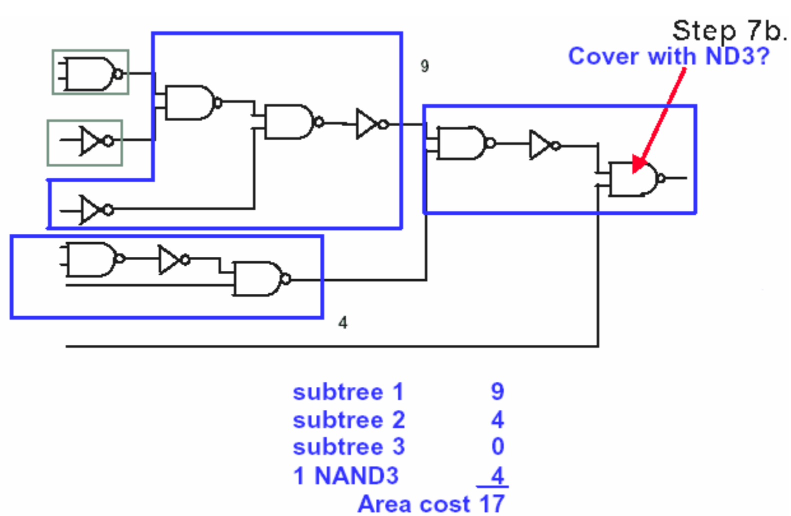


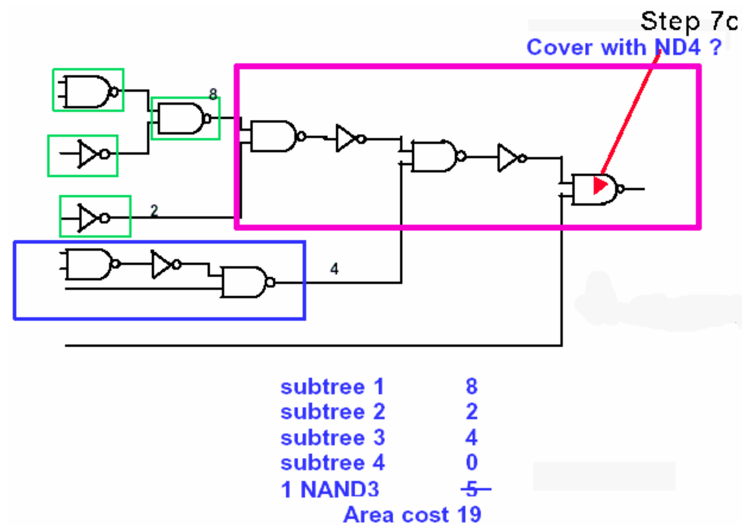


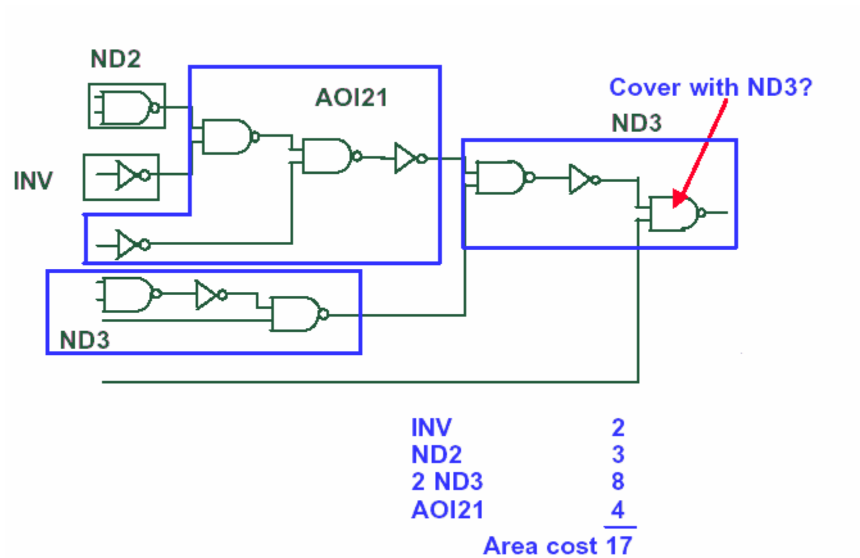


Adapted from [Terman'02,Devadas'06]

Optimal Tree Covering Example







This matches our earlier intuitive cover, but accomplished systematically.

Refinements: timing optimization incorporating load-dependent delays, optimization for low power.

Adapted from [Terman'02,Devadas'06]

Acknowledgments

- [Weste'11] N. Weste and D. Harris, "CMOS VLSI Design: A Circuits and Systems Perspective," 4th ed, Addison Wesley, 2011.
- [Synopsys'11] "DesignWare Datapath and Building Block IP: Quick Reference," Synopsys, 2011.
- [Zhou'02] H. Zhou, Northwestern ECE 303 Advanced Digital Design, Lecture Slides, 2002.
- [Terman'02] C. Terman and K. Asanović, MIT 6.371 Introduction to VLSI Systems, Lecture Slides, 2002.
- [Nowick'12] S. Nowick, "The Quine-McCluskey Method," Columbia CSEE E6861y Computer-Aided Design of Digital Systems, Handout, 2012.
- [Devadas'06] S. Devadas, "VLSI CAD Flow: Logic Synthesis, Placement, and Routing," MIT 6.375 Complex Digital Systems Guest Lecture Slides, 2006.