

ECE 260C

Logic Synthesis

Cho Moon

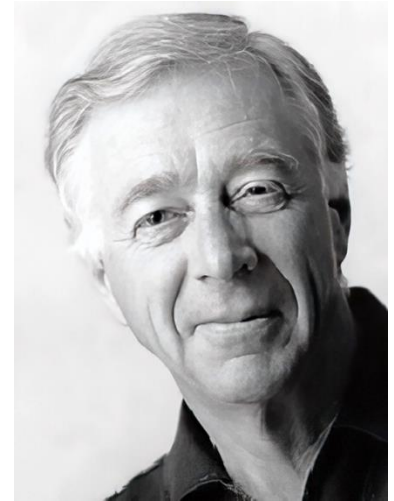
- Cho Moon has over 25 years of experience in EDA. He is currently a distinguished engineer at Precision Innovations. He has led advanced node development for tools like PrimeTime, Fusion Compiler and First Encounter at Synopsys, Blaze DFM, Cadence and other EDA and semiconductor companies.
- He has expertise in static timing analysis, pre-route and post-route optimization, Engineering Change Order (ECO) optimization, logic synthesis and formal verification.
- He holds a PhD from UC Berkeley. Bob Brayton was his advisor.



Bob Brayton's Favorite Quote

- *Your manuscript is both good and original. Unfortunately, the part that is good is not original, and the part that is original is not good.*

• Samuel Johnson



Prof. Bob Brayton

Outline

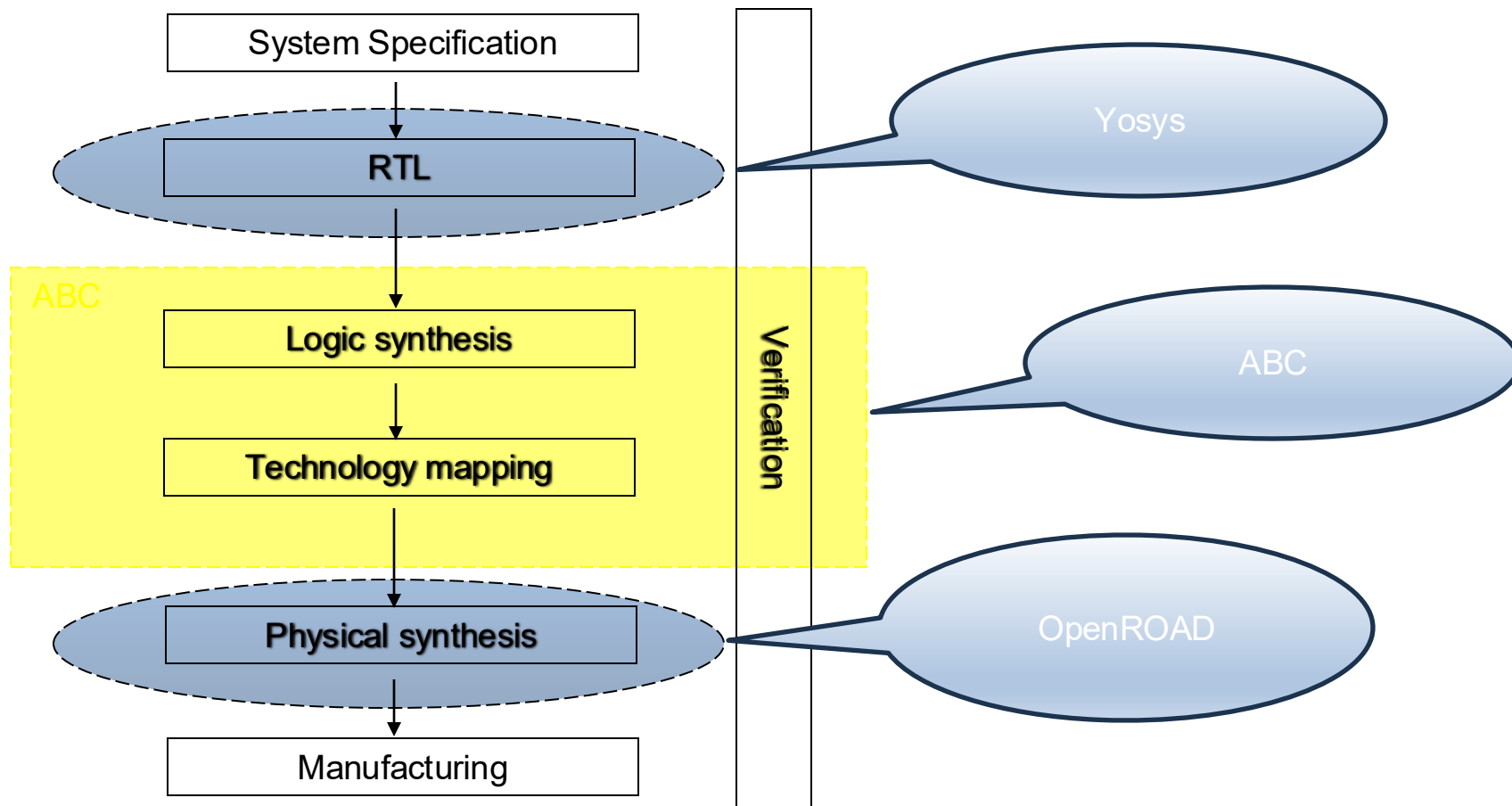
- Introduction
- Two-level Logic Synthesis
- Multi-level Logic Synthesis
 - Classical
 - Contemporary
- Technology Mapping
 - Classical
 - Contemporary

Why Logic Synthesis?

- Indispensable
 - Problem is too hard without it
 - Similar to C++/C compilers
- Big impact on final PPA (performance, power and area)
- Foundation for many applications like
 - Verification
 - Timing analysis
 - Testing
 - Sequential optimization

Design Flow

- Where does logic synthesis fit in?



Synthesis Example (Yosys + ABC)

```
module adder1 (  
    input  clk,  
    input  a,  
    input  b,  
    input  cin,  
    output reg sum,  
    output reg cout  
);  
    reg [1:0]  result;  
    always @(posedge clk) begin  
        result <= a + b + cin;  
        sum <= result[0];  
        cout <= result[1];  
    end  
endmodule
```



```
module adder1(clk, a, b, cin, sum, cout);  
    wire _0_;  
    wire _1_;  
    input a;  
    wire a;  
    input b;  
    wire b;  
    input cin;  
    wire cin;  
    input clk;  
    wire clk;  
    output cout;  
    wire cout;  
    wire \result[0] ;  
    wire \result[1] ;  
    output sum;  
    wire sum;  
    sky130_fd_sc_hd_xor3_1_2_ ( .A(cin), .B(a), .C(b), .X(_1_) );  
    sky130_fd_sc_hd_maj3_1_3_ ( .A(cin), .B(a), .C(b), .X(_0_) );  
    sky130_fd_sc_hd_dfxt_1_1_ \cout$DFF_P_ ( .CLK(clk), .D(\result[1] ), .Q(cout) );  
    sky130_fd_sc_hd_dfxt_1_1_ \result[0]$DFF_P_ ( .CLK(clk), .D(_1_), .Q(\result[0] ) );  
    sky130_fd_sc_hd_dfxt_1_1_ \result[1]$DFF_P_ ( .CLK(clk), .D(_0_), .Q(\result[1] ) );  
    sky130_fd_sc_hd_dfxt_1_1_ \sum$DFF_P_ ( .CLK(clk), .D(\result[0] ), .Q(sum) );  
endmodule
```

RTL (Register Transfer Level) Specification
= “Behavioral” Verilog

Gate-level Netlist Implementation
= “Structural” Verilog

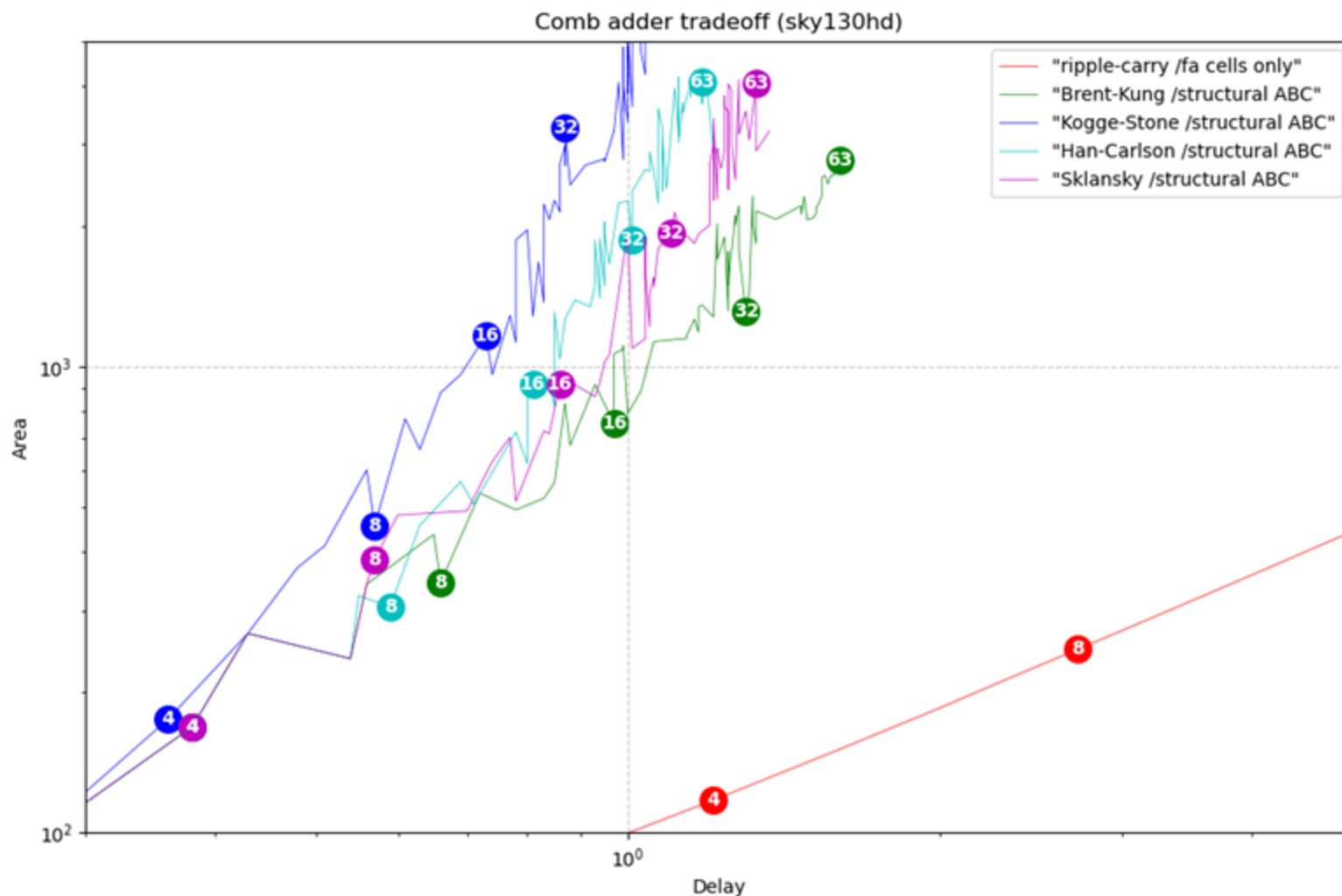
Problem

- Given
 - Initial design
 - Design constraints
 - Timing
 - Power
 - Area
 - Target technology libraries
- Produce
 - Optimal gate-level netlist that meets design constraints

Very hard optimization problem!

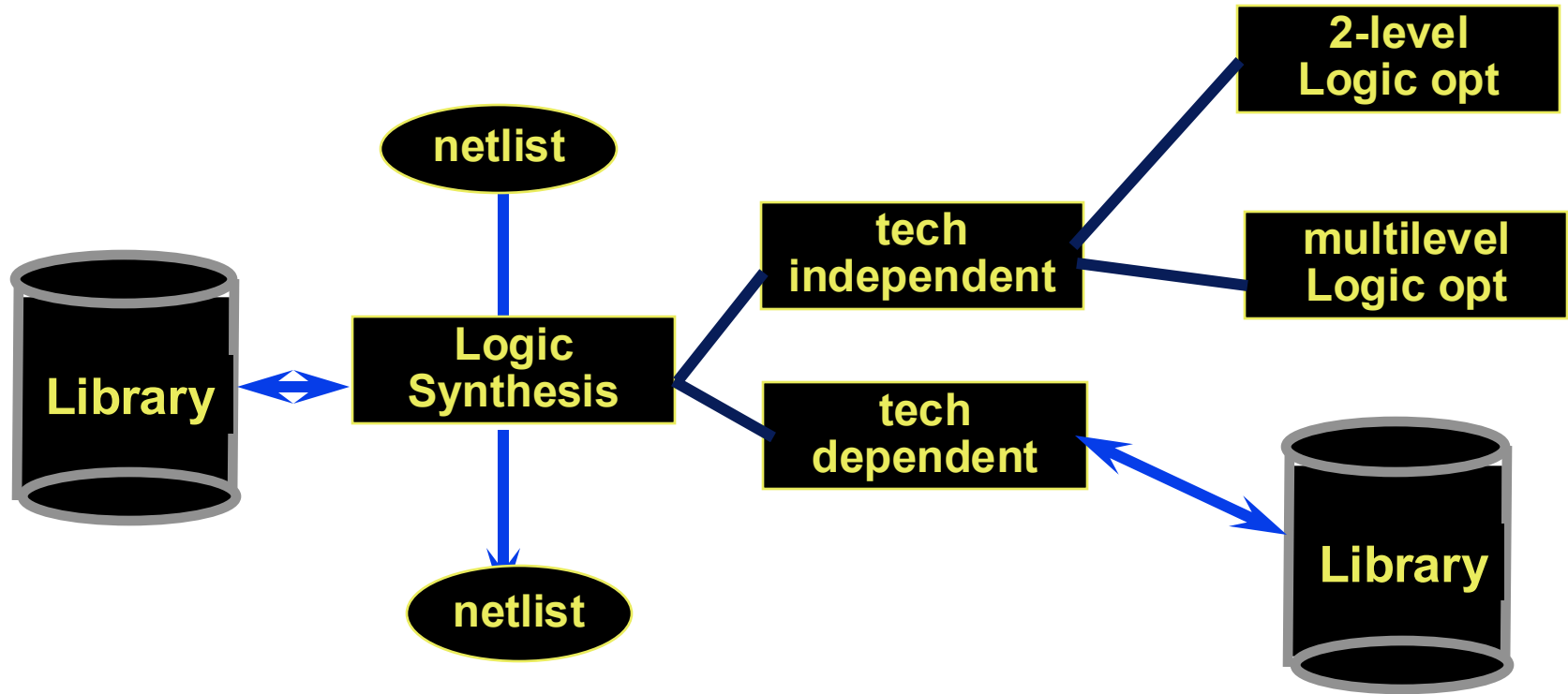
Adder Choices

Area vs. delay tradeoff for different adders with varying bit widths



Slide Courtesy of Martin Poviser and Emil Tywoniak at YosysHQ

Combinational Logic Synthesis



Slide courtesy of Devadas, et. al

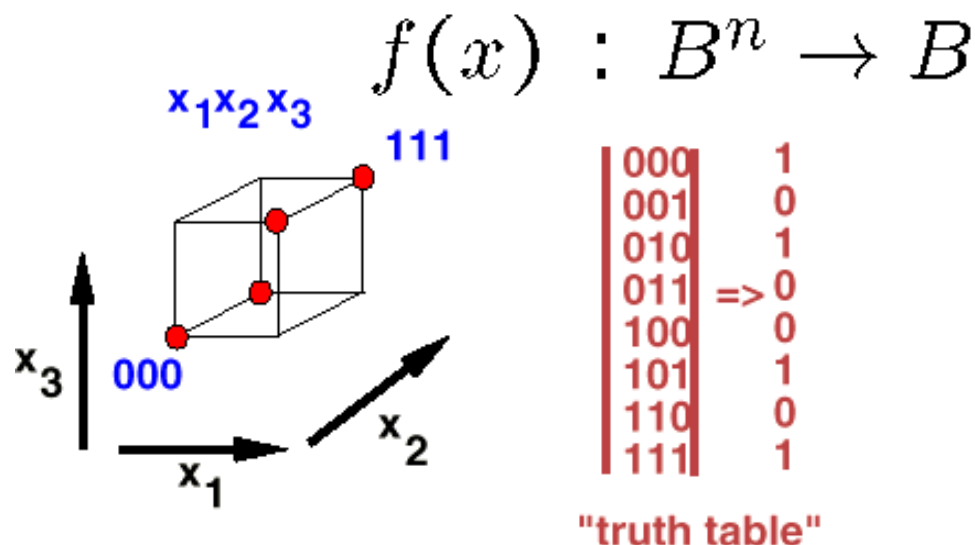
Boolean Functions

$$f(x) : B^n \longrightarrow B$$

$$B = \{0, 1\}, x = (x_1, x_2, \dots, x_n)$$

- x_1, x_2, \dots are **variables**
- $x_1, \overline{x_1}, x_2, \overline{x_2}, \dots$ are **literals**
- each vertex of B^n is mapped to 0 or 1
- the **onset** of f is a set of input values for which $f(x) = 1$
- the **offset** of f is a set of input values for which $f(x) = 0$

Logic Formulas



There are 2^n vertices in input space B^n
There are 2^{2^n} distinct logic **functions**. Each subset of vertices is a distinct logic function: $f^1 \subseteq B^n$
There are ∞ number of logic **formulas**

$$\begin{aligned} f &= x + y \\ &= x\bar{y} + xy + \bar{x}y \\ &= x\bar{x} + x\bar{y} + y \\ &= (x + y)(x + \bar{y}) + \bar{x}y \end{aligned}$$

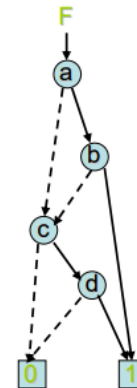
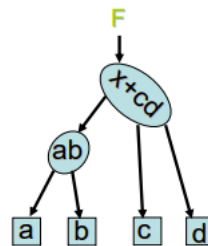
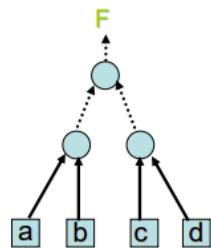
SYNTHESIS = Find the "best" formula (or "representation")

Logic Representation

- Sum of products (SOP): $F = ab + cd$
- Product of sums (POS): $F = (a+c)(a+d)(b+c)(b+d)$
- Truth Table (TT)

F	0	0	0	1	0	0	0	1	0	0	0	1	1	1	1	1
ab	00	00	00	00	01	01	01	01	10	10	10	10	11	11	11	11
cd	00	01	10	11	00	01	10	11	00	01	10	11	00	01	10	11

- Binary decision diagram (BDD)
- Logic Network (LN)
- And-inverter Graph (AIG)



- Slide Courtesy of Alan Mischenko

Which Representation and Which Optimization?

Design Size	# Inputs	Representation	Optimization	Tools
Small	<= 16	Truth Table (TT) Sum of Products (SOP)	Minimize number of products or literals <ul style="list-style-type: none">ab is better than abcab is better than abc + abc'	Espresso
Medium	16-100	Binary Decision Diagrams (BDD) Logic Network (LG) And-inverter Graph (AIG)	Minimize number of nodes or widths Minimize "area" or levels Minimize number of nodes or levels	SIS VIS, MVSIS ABC
Large	> 100	And-inverter Graph (AIG)	Minimize number of nodes or levels	ABC

ab
cd 00 01 11 10

00	0	0	1	0
01	0	0	1	1
11	0	1	1	0
10	0	0	1	0

$F(a,b,c,d) = ab + d(ac' + bc)$

6 nodes
4 levels


ab
cd 00 01 11 10

00	0	0	1	0
01	0	0	1	1
11	0	1	1	0
10	0	0	1	0

$F(a,b,c,d) = ac'(b'd')' + c(a'd')' = ac'(b+d) + bc(a+d)$

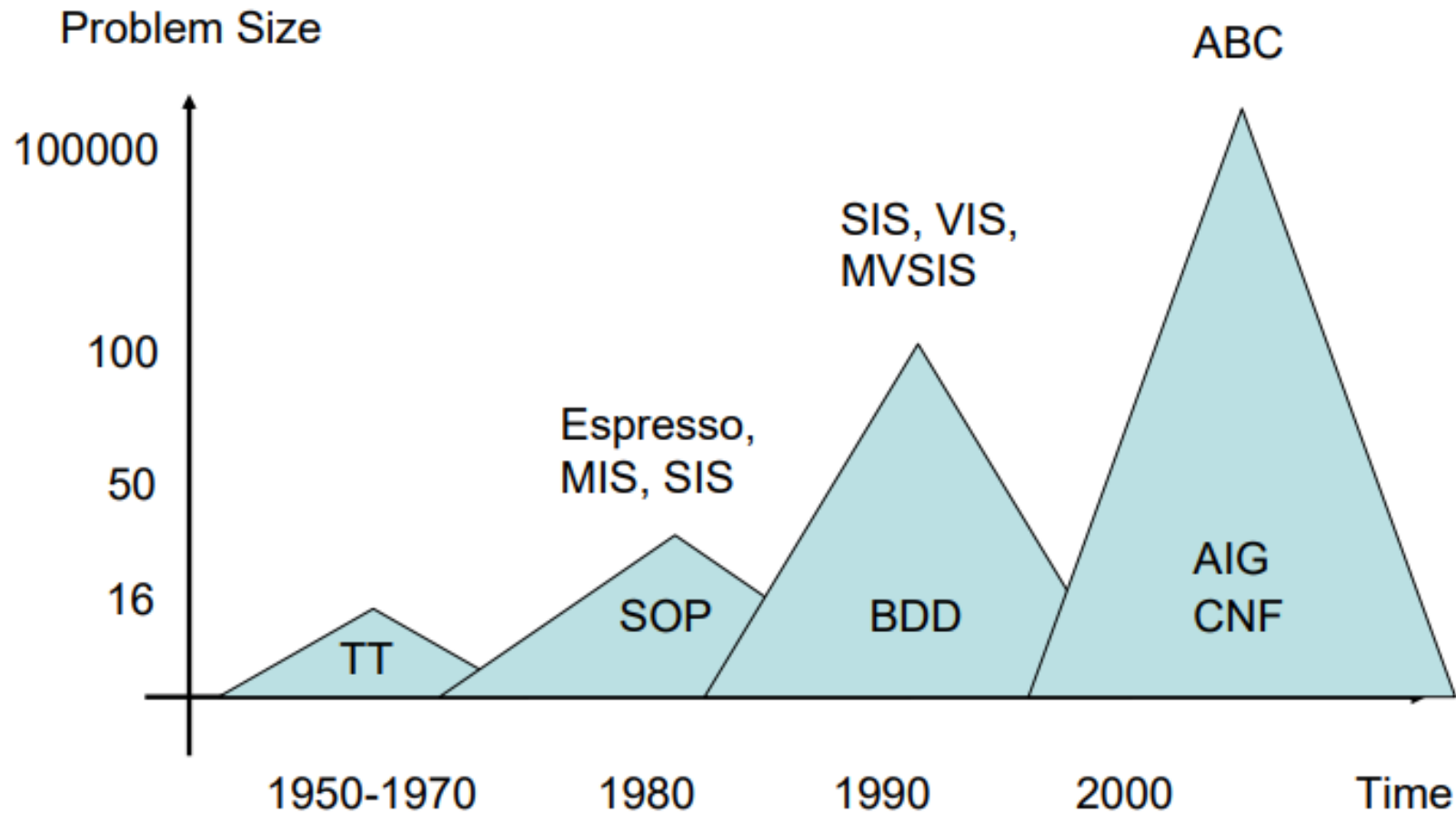
7 nodes
3 levels
23

• Slide Courtesy of Alan Mischenko

Kahng ECE 260C SP25

14

Historical Perspective



- Slide Courtesy of Alan Mischenko

Outline

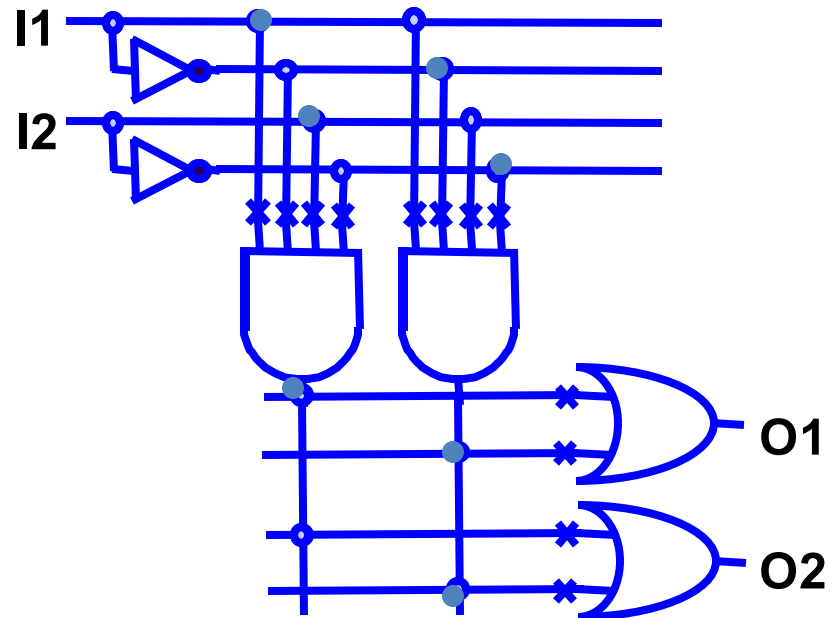
- Introduction
- **Two-level Logic Synthesis**
- Multi-level Logic Synthesis
 - Classical
 - Contemporary
- Technology Mapping
 - Classical
 - Contemporary

Two-level Logic Synthesis Problem

- Given an arbitrary logic function in two-level form, produce a smaller representation.
- For sum-of-products (SOP) implementation on PLAs (programmable logic arrays), fewer product terms and fewer inputs to each product term mean smaller area.

$$O1 = I1 I2 + I1' I2'$$

$$O2 = I1' I2'$$

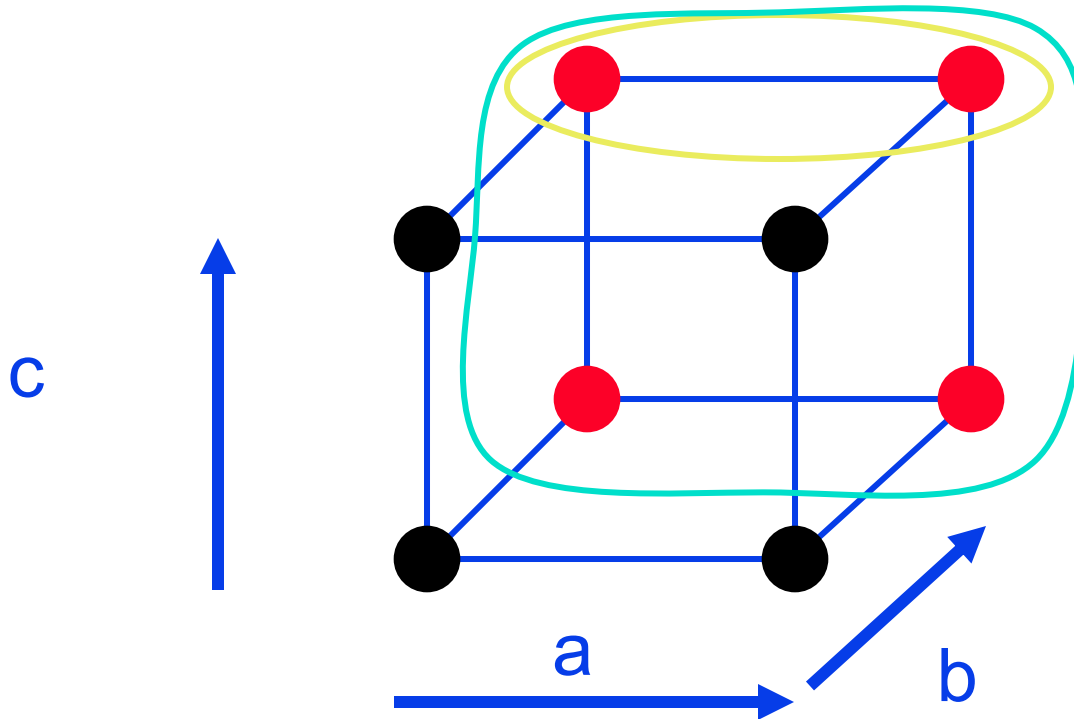


Sum-of-products (SOP)

- A function can be represented by a sum of cubes (products):
 - $f = ab + ac + bc$
- Since each cube is a product of literals, this is a “sum of products” representation
- A SOP can be thought of as a set of cubes F
 - $F = \{ab, ac, bc\} = C$
- A set of cubes that represents f is called a cover of f .
 - $F = \{ab, ac, bc\}$ is a cover of $f = ab + ac + bc$.

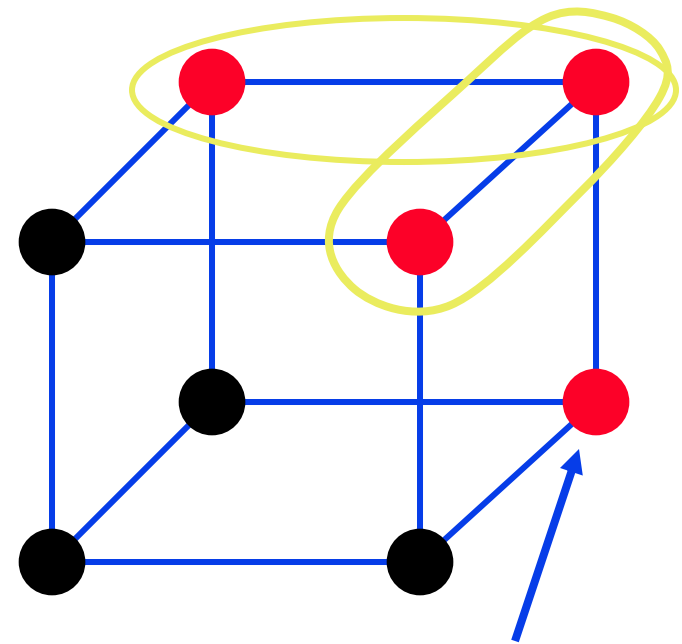
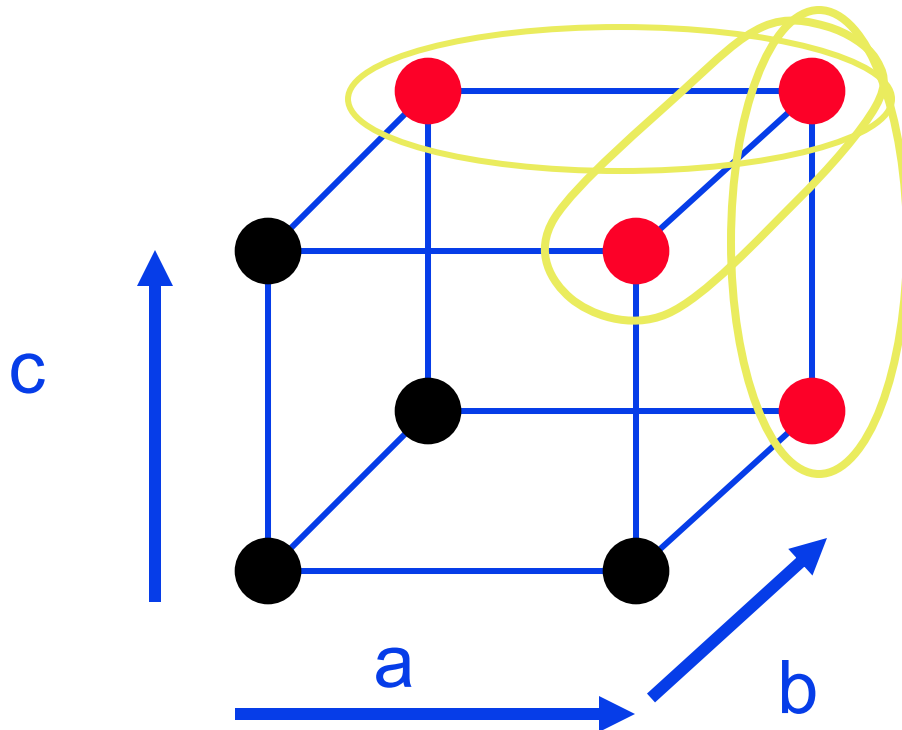
Prime Cover

- A cube is prime if there is no other cube that contains it
 - (for example, bc is not a prime but b is)
- A cover is prime iff all of its cubes are prime



Irredundant Cube

- A cube c of a cover C is irredundant if C fails to be a cover if c is dropped from C
- A cover is irredundant iff all its cubes are irredundant (for example, $F = a b + a c + b c$)



Not covered

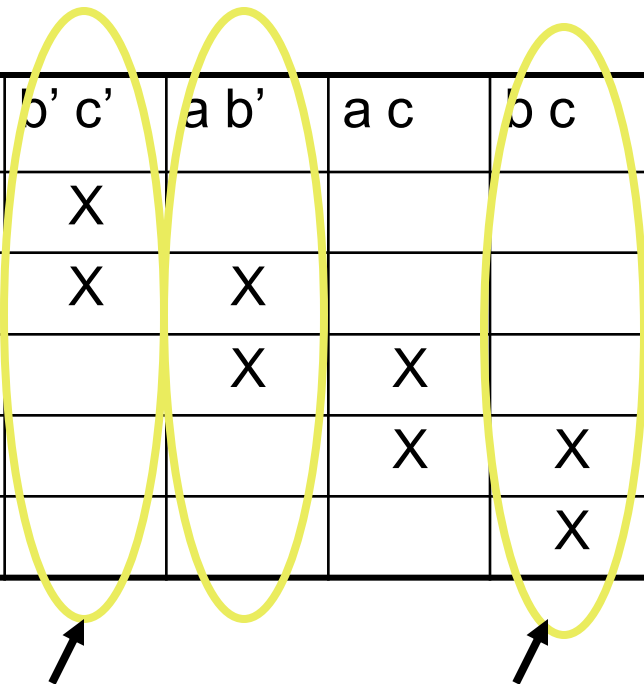
Quine-McCluskey Method

- We want to find a minimum prime and irredundant cover for a given function.
 - Prime cover leads to min number of inputs to each product term.
 - Min irredundant cover leads to min number of product terms.
- Quine-McCluskey (QM) method (1960's) finds a minimum prime and irredundant cover.
 - Step 1: List all minterms of on-set: $O(2^n)$ $n = \text{\#inputs}$
 - Step 2: Find all primes: $O(3^n)$ $n = \text{\#inputs}$
 - Step 3: Construct minterms vs primes table
 - Step 4: Find a min set of primes that covers all the minterms: $O(2^m)$ $m = \text{\#primes}$

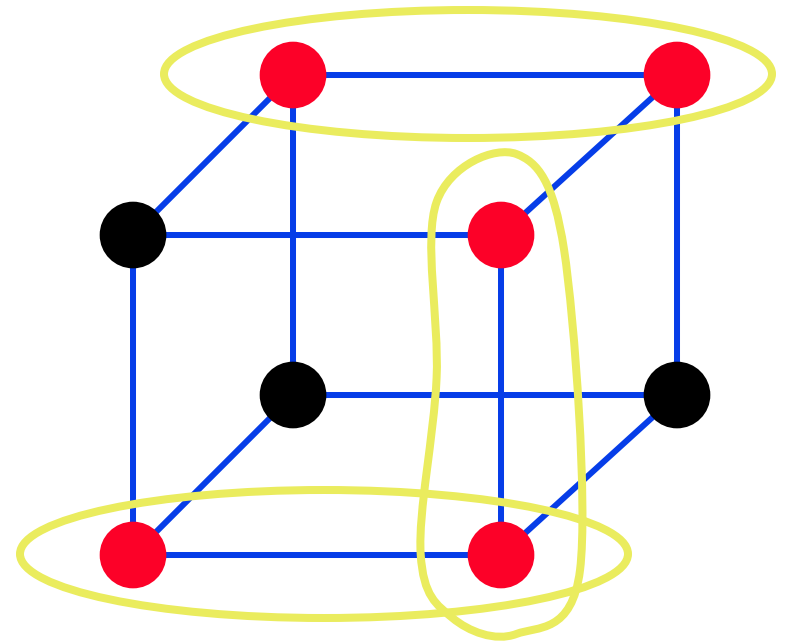
QM Example

- $F = a' b' c' + a b' c' + a b' c + a b c + a' b c$
- Find a minimum set of primes that covers all the minterms
“Minimum column covering problem”

	$b' c'$	$a b'$	$a c$	$b c$
$a' b' c'$	X			
$a b' c'$	X	X		
$a b' c$		X	X	
$a b c$			X	X
$a' b c$				X



Essential primes

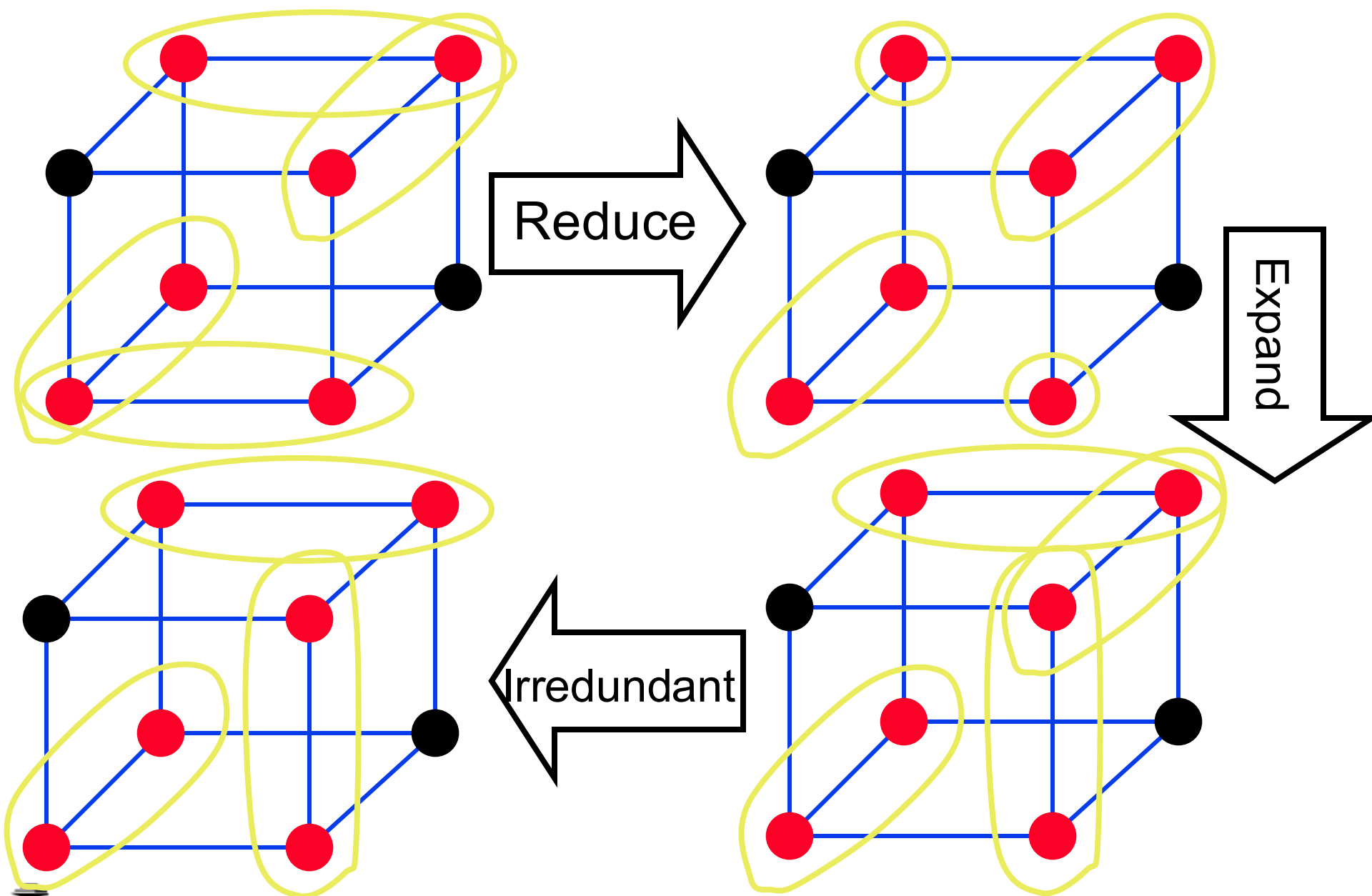


ESPRESSO – Heuristic Minimizer

- Quine-McCluskey gives a minimum solution but is only good for functions with small number of inputs (< 10)
- ESPRESSO is a heuristic two-level minimizer that finds a “minimal” solution

```
ESPRESSO(F) {  
  do {  
    reduce(F);  
    expand(F);  
    irredundant(F);  
  } while (fewer terms in F);  
  verify(F);  
}
```

ESPRESSO ILLUSTRATED



Outline

- Introduction
- Two-level Logic Synthesis
- **Multi-level Logic Synthesis**
 - **Classical**
 - Contemporary
- Technology Mapping
 - Classical
 - Contemporary

Multi-level Logic Synthesis

- Two-level logic synthesis is effective and mature
- Two-level logic synthesis is directly applicable to PLAs and PLDs

But...

- There are many functions that are too expensive to implement in two-level forms (too many product terms!)
- Two-level implementation constrains layout (AND-plane, OR-plane)
- Rule of thumb:
 - Two-level logic is good for control logic
 - Multi-level logic is good for datapath or random logic

Multi-level Logic Synthesis Problem

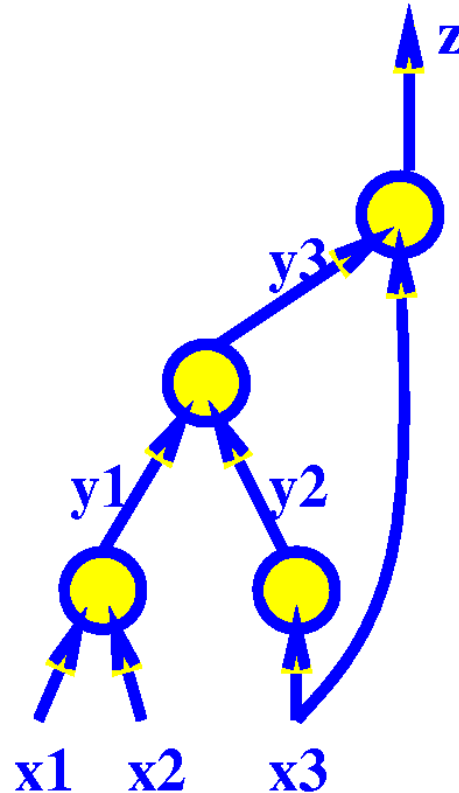
- Given
 - Initial logic network
 - Design constraints
 - Arrival times, required times, power consumption, noise immunity, etc...
 - Target technology libraries
- Produce
 - a minimum “cost” netlist consisting of the gates from the target libraries such that design constraints are satisfied

Modern Approach to Logic Optimization

- Divide logic optimization into two subproblems:
 - Technology-independent optimization
 - determine overall logic structure
 - estimate costs (mostly) independent of technology
 - simplified cost modeling
 - Technology-dependent optimization (technology mapping)
 - binding onto the gates in the library
 - detailed technology-specific cost model
- Orchestration of various optimization/transformation techniques for each subproblem

Network Representation

- Logic Network:



$$\begin{aligned}y1 &= x1 \ x2 \\y2 &= x3' \\y3 &= y1 \ y2 \\z &= y3 + x3\end{aligned}$$

Node Representation: Sum of Products (SOP)

- Example:

$abc' + a'bd + b'd' + b'e'f$ (sum of cubes)

- Advantages:

- easy to manipulate and minimize
- many algorithms available (e.g. AND, OR, TAUTOLOGY)
- two-level theory applies

- Disadvantages:

- Not representative of logic complexity. For example,
 - $f = ad + ae + bd + be + cd + ce$
 - $f' = a'b'c' + d'e'$
 - These differ in their implementation by an inverter.
- hence not easy to estimate logic; difficult to estimate progress during logic manipulation

Technology-Independent Optimization

- Technology-independent optimization is a bag of tricks:

- Two-level minimization (also called simplify)

- Constant propagation (also called sweep)

$$f = a b + c; b = 1 \Rightarrow f = a + c$$

- Decomposition (single function)

$$f = abc + abd + a'c'd' + b'c'd' \Rightarrow f = xy + x'y'; \quad x = ab; \quad y = c + d$$

- Extraction (multiple functions)

$$f = (az + bz')cd + e \quad g = (az + bz')e' \quad h = cde$$

↓

$$f = xy + e \quad g = xe' \quad h = ye \quad x = az + bz' \quad y = cd$$

More Technology-Independent Optimization

- More technology-independent optimization tricks:

- Substitution

$$g = a+b \quad f = a+bc$$

\Downarrow

$$f = g(a+c)$$

- Collapsing (also called elimination)

$$f = ga+g'b \quad g = c+d$$

\Downarrow

$$f = ac+ad+bc'd' \quad g = c+d$$

- Factoring (series-parallel decomposition)

$$f = ac+ad+bc+bd+e \Rightarrow f = (a+b)(c+d)+e$$

Outline

- Introduction
- Two-level Logic Synthesis
- Multi-level Logic Synthesis
 - Classical
 - **Contemporary**
- Technology Mapping
 - Classical
 - Contemporary

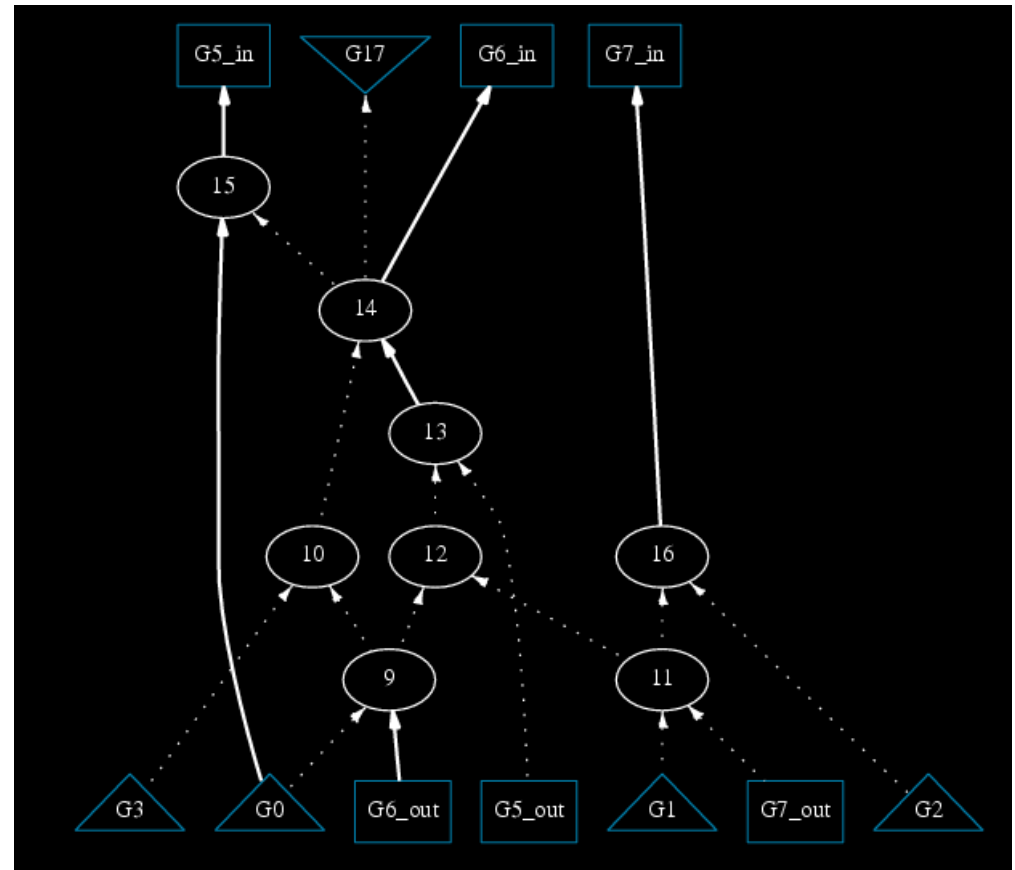
Motivation for Contemporary Synthesis

- Classical combinational tech-independent synthesis
 - suboptimal
 - complicated
 - hard to implement
 - slow
- What if we replace it with something that is
 - suboptimal, but
 - simple
 - easy to implement
 - fast

Slide Courtesy of Alan Mischenko & Bob Brayton

And-Inverter Graph (AIG)

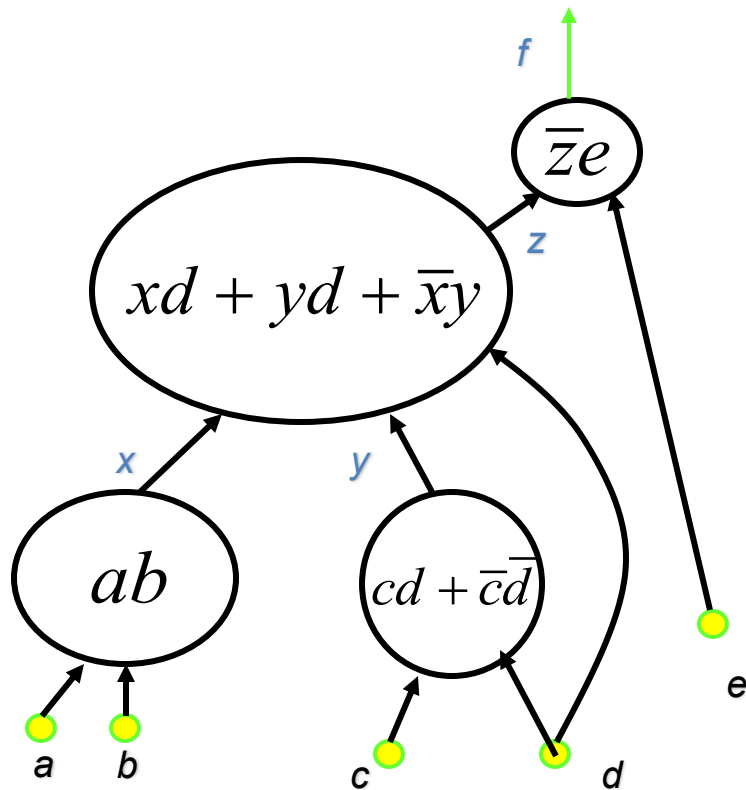
- Every node is two-input AND
- Dotted line denotes inversion
- Primary inputs and register outputs at bottom
- Primary outputs and register inputs at top
- Can represent any set of logic functions



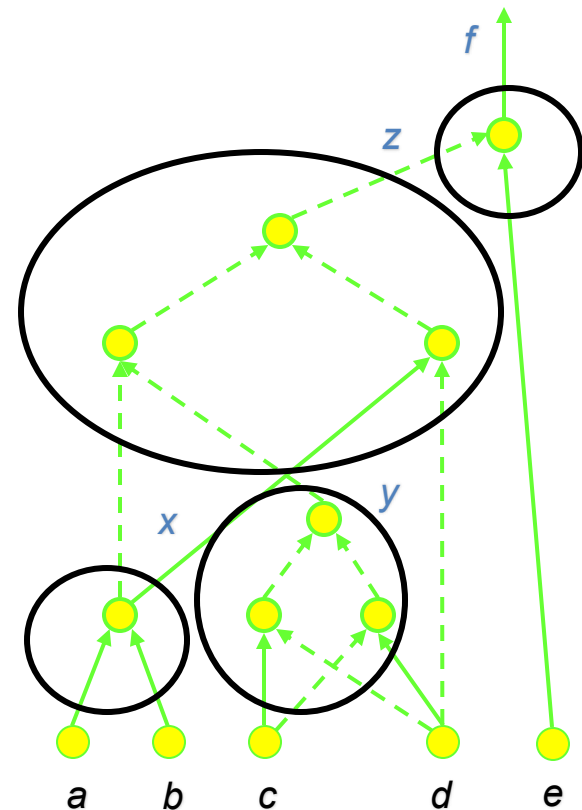
Slide Courtesy of Alan Mischenko & Bob Brayton

Logic Network vs. AIG

Logic network in SIS



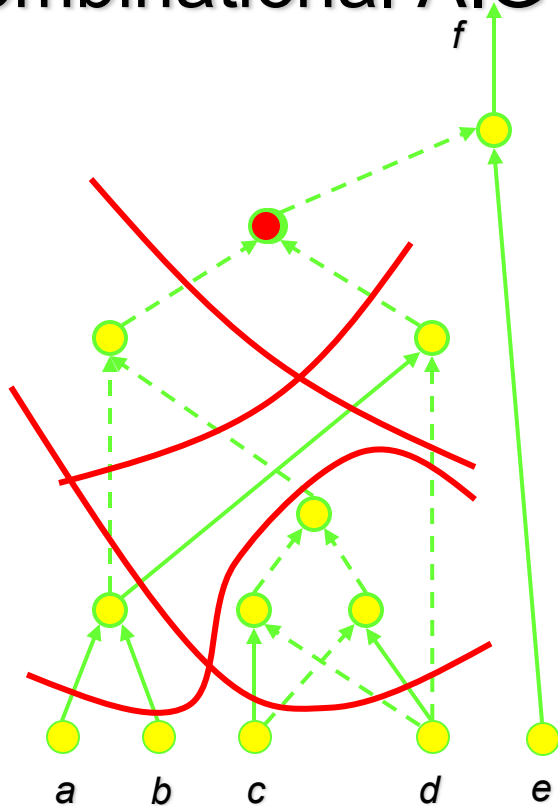
Equivalent AIG in ABC



AIG is a logic network of 2-input AND nodes and inverters (dotted lines)

One AIG Node – Many Cuts

Combinational AIG



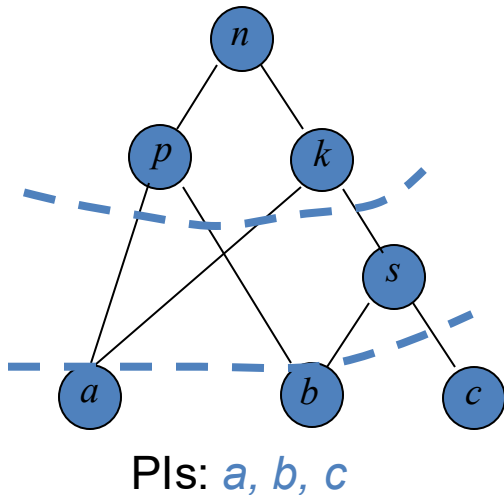
AIG can be used to compute many cuts for each node

- Each cut in AIG represents a **different** SIS node
- No a priori fixed boundaries
- Implies that AIG manipulation with cuts is equivalent to working on many logic networks at the same time

Different cuts for the same node

k -Feasible Cuts

- **Definition.** A set of nodes C is a k -feasible cut for a node n if
 - (1) all the paths from the primary inputs (PIs) to node n pass through at least one node in C ,
 - (2) the number of nodes in C does not exceed k



- 3-feasible cuts of n :

$$C1 = \{ p, k \}$$

$$C2 = \{ a, b, s \}$$

- Not 3-feasible cuts of n :

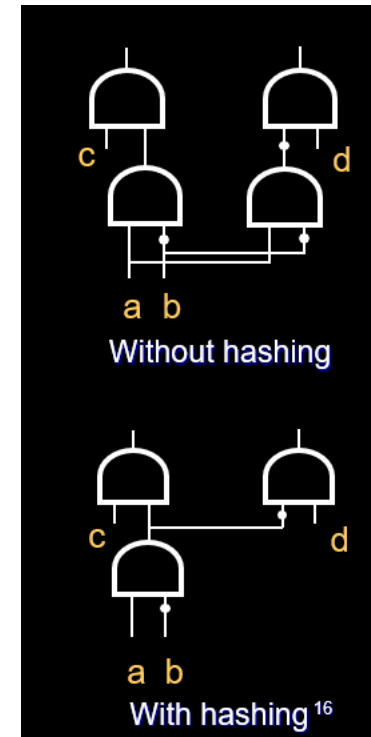
$$C3 = \{ p, b, c \}$$

$$C4 = \{ a, b, s, c \}$$

k	Average number of cuts per node
4	6
5	20
6	80
7	150

AIG Structural Hashing

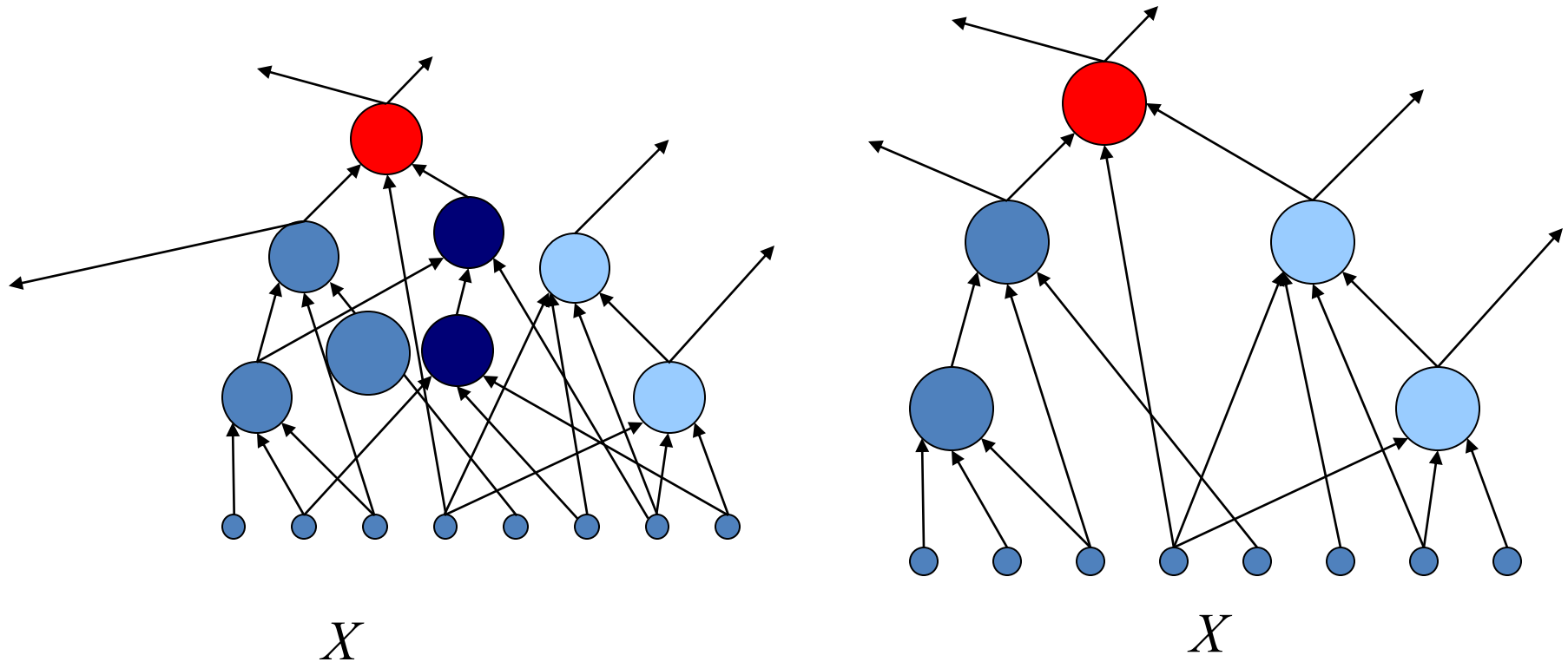
- Leads to a compact representation
- Is applied during AIG construction
 - Propagates constants
 - Makes each node structurally unique



Slide Courtesy of Alan Mischenko & Bob Brayton

AIG Resubstitution

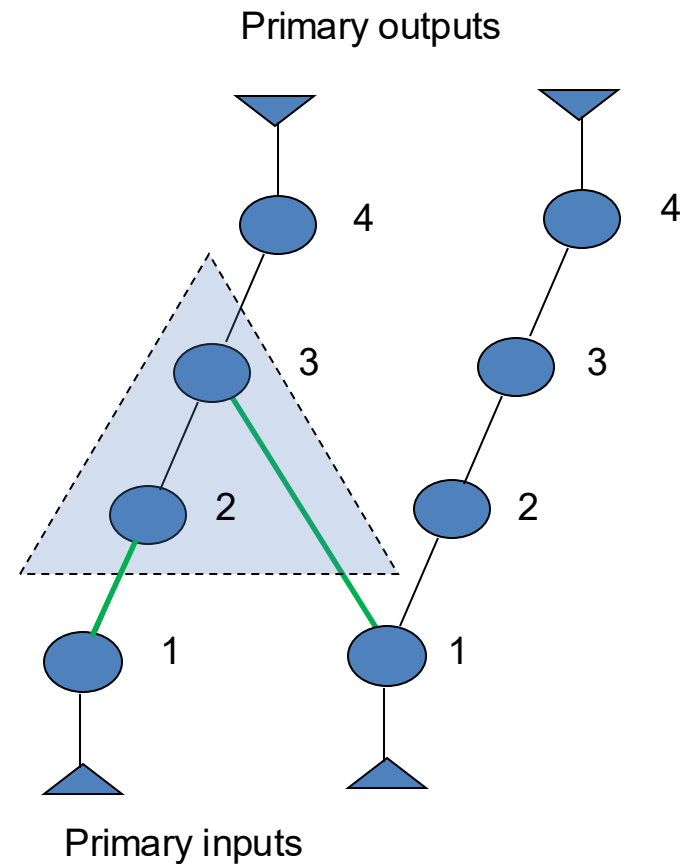
Resubstitution considers a **node** in a **logic network** and expresses it using a different set of fanins



AIG “speedup”

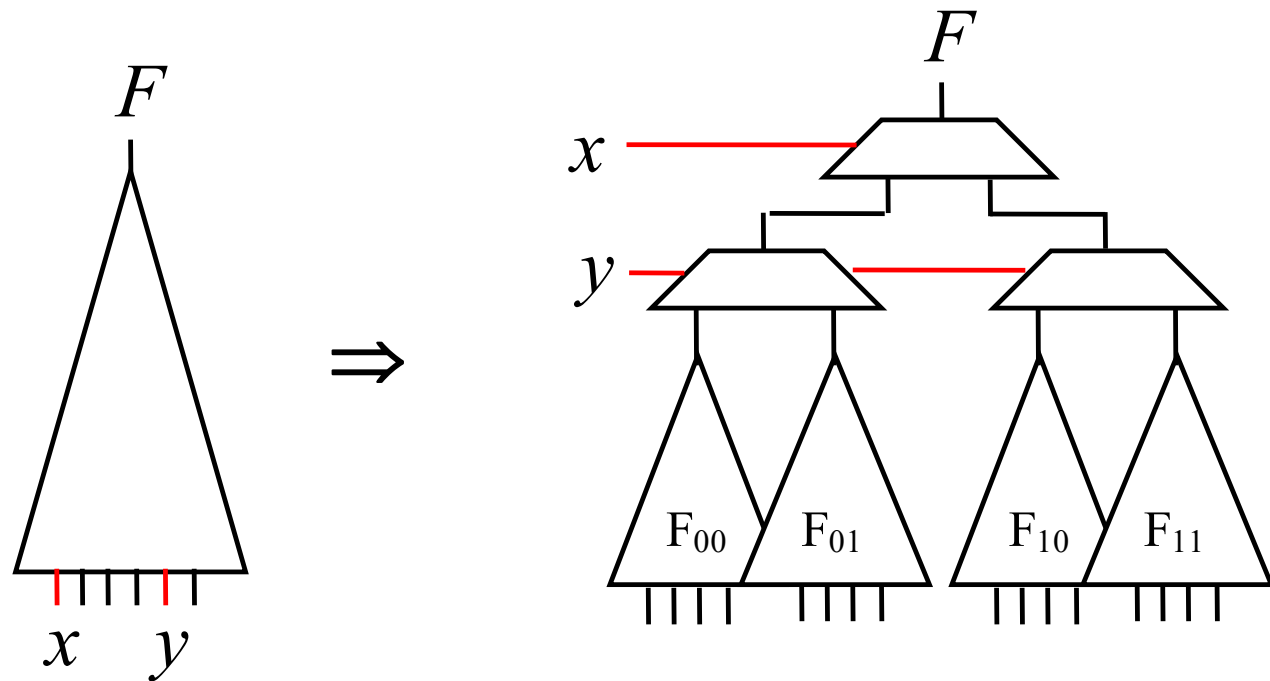
Timing Criticality

- **Critical nodes**
 - Used by many traditional algorithms
- **Critical edges**
 - Used by ABC
- ABC pre-computes critical edges of critical nodes
 - Reduces computation
- An edge between critical nodes may not be critical
 - See illustration: edge $1 \rightarrow 3$



Delay-Oriented Restructuring

- Using traditional MUX-restructuring
 - AKA *generalized select transform*

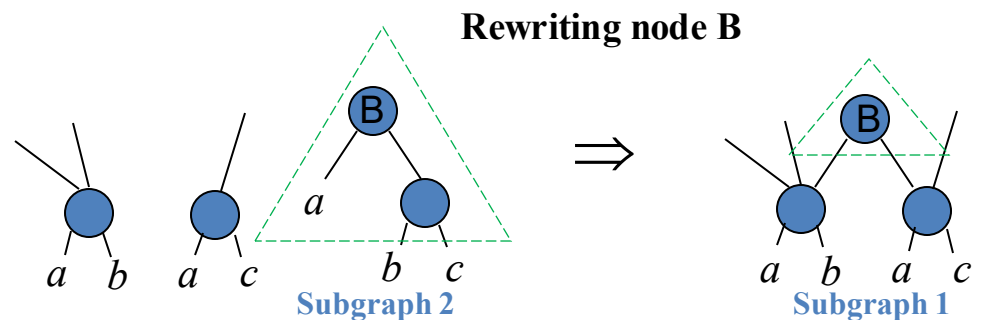
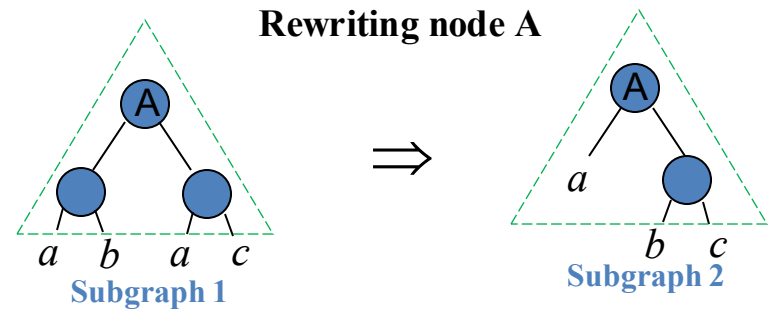
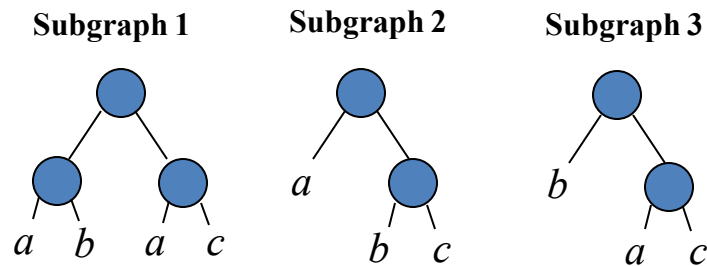


x and y are the critical edge inputs

AIG Rewriting

- Pre-computing subgraphs
 - Consider function $f = abc$

■ Rewriting subgraphs



In both cases 1 node is saved

Slide Courtesy of Alan Mischenko & Bob Brayton

Combinational Synthesis By AIG Rewriting

```
iterate 10 times {  
  for each AIG node {  
    for each k-cut  
      derive node output as function of cut variables  
      if ( smaller AIG is in the pre-computed library )  
        rewrite using improved AIG structure  
    }  
  }  
}
```

Note: each AIG node has, on average, 5 4-cuts compared to a SIS node with only 1 cut

Rewriting at a node can be very fast – using hash-table lookups, truth table manipulation, disjoint decomposition

Experimental Results (MCNC)

Logic synthesis flow used for optimization	Standard cells		FPGAs (k=5)		Runtime ratio
	Area ratio	Delay ratio	Area ratio	Delay ratio	
No optimization	1.00	1.00	1.00	1.00	0.00
ABC (AIG rewriting)	0.87	0.96	0.93	0.98	1.00
MVSIS (mvsis.rugged)	0.91	1.10	0.93	1.03	7.12
SIS (script.delay)	0.94	0.99	0.98	0.97	~100.00
SIS (script.rugged + speed_up)	0.94	0.90	0.98	0.94	~1000.00

A. Mischenko, S. Chatterjee, R. Brayton, “DAC-Aware AIG Rewriting: A Fresh Look at Combinational Logic Synthesis”, DAC 2006

Outline

- Introduction
- Two-level Logic Synthesis
- Multi-level Logic Synthesis
 - Classical
 - Contemporary
- **Technology Mapping**
 - **Classical**
 - Contemporary

Technology-Dependent Optimization

Technology-dependent optimization consists of

- Technology mapping: maps logic network to a set of gates from technology libraries
- Local transformations
 - Discrete resizing
 - Cloning
 - Fanout optimization (buffering)
 - Logic restructuring

Slide courtesy of Keutzer

Classical Technology Mapping

Input

- Technology independent, optimized logic **network**
- Description of the gates in the **library** with their cost

Output

- **Netlist of gates** (from library) which minimizes total cost

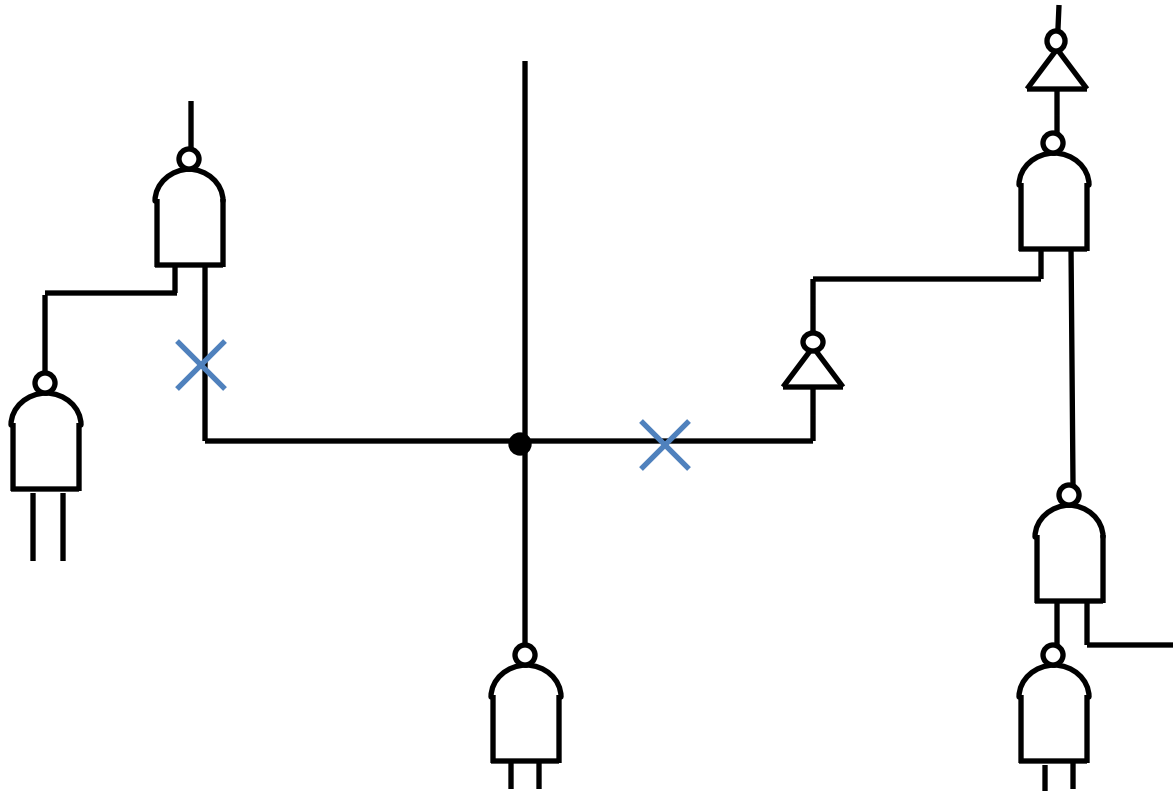
General Approach

- Construct a subject DAG for the network
- Represent each gate in the target library by pattern DAG's
- Find an optimal-cost covering of subject DAG using the collection of pattern DAG's
- Canonical form: 2-input NAND gates and inverters

DAGON: Technology Binding and Local Optimization by DAG Matching
(K. Keutzner, 1987)

DAG Covering

- DAG covering is an NP-hard problem
- Solve the sub-problem optimally
 - Partition DAG into a forest of trees
 - Solve each tree optimally using tree covering
 - Stitch trees back together



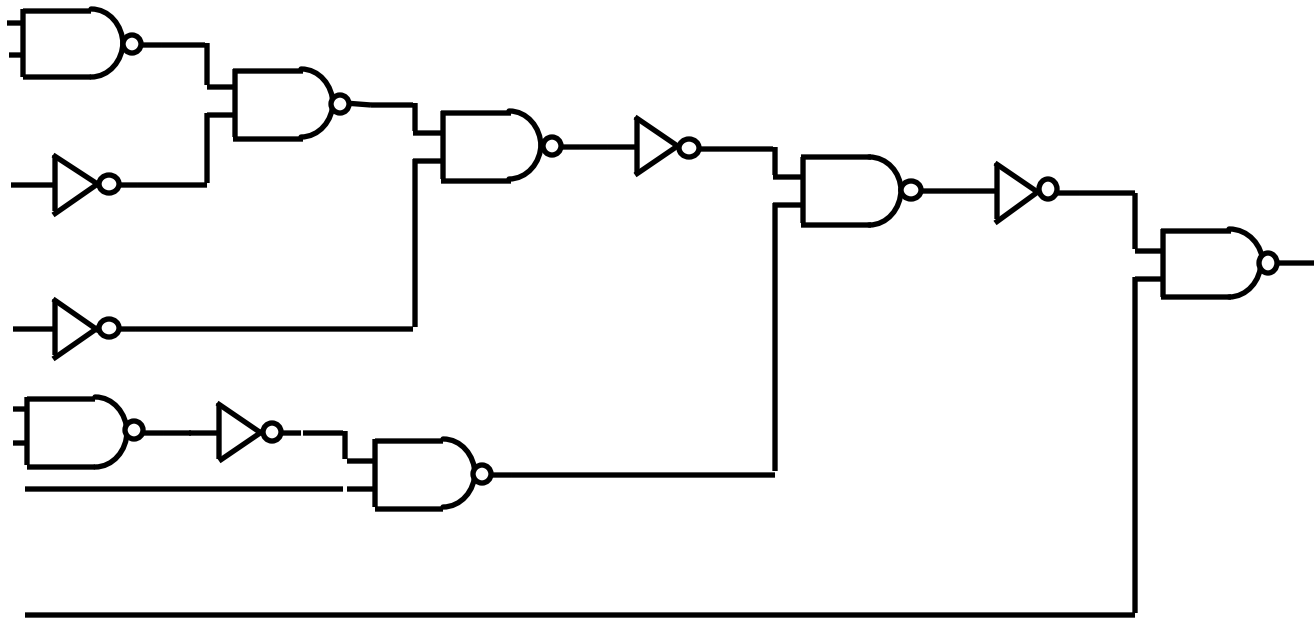
Slide courtesy of Keutzer

Tree Covering Algorithm

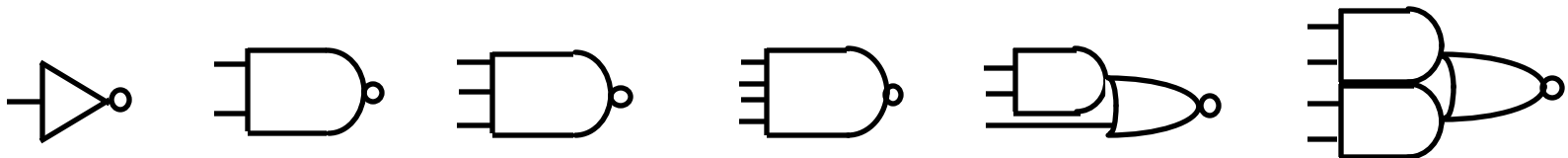
- Transform netlist and libraries into canonical forms
 - 2-input NANDs and inverters
- Visit each node in BFS from inputs to outputs
 - Find all candidate matches at each node N
 - Match is found by comparing topology only (no need to compare functions)
 - Find the optimal match at N by computing the new cost
 - New cost = cost of match at node N + sum of costs for matches at children of N
 - Store the optimal match at node N with cost
- Optimal solution is guaranteed if cost is area
- Complexity = $O(n)$ where n is the number of nodes in netlist

Tree Covering Example

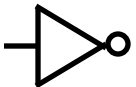
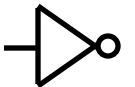
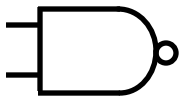


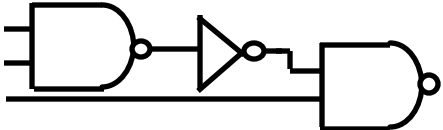

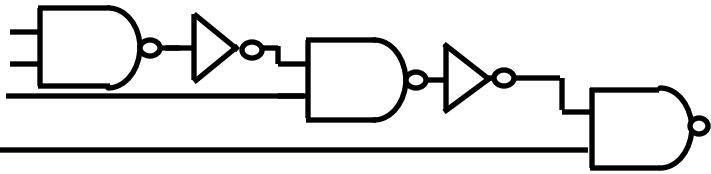
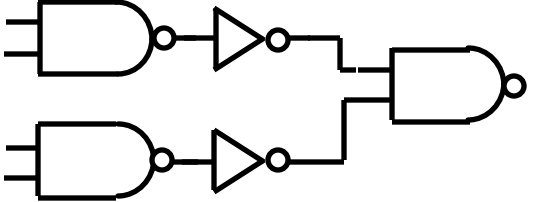
Find an “optimal” (in area, delay, power) mapping of this circuit



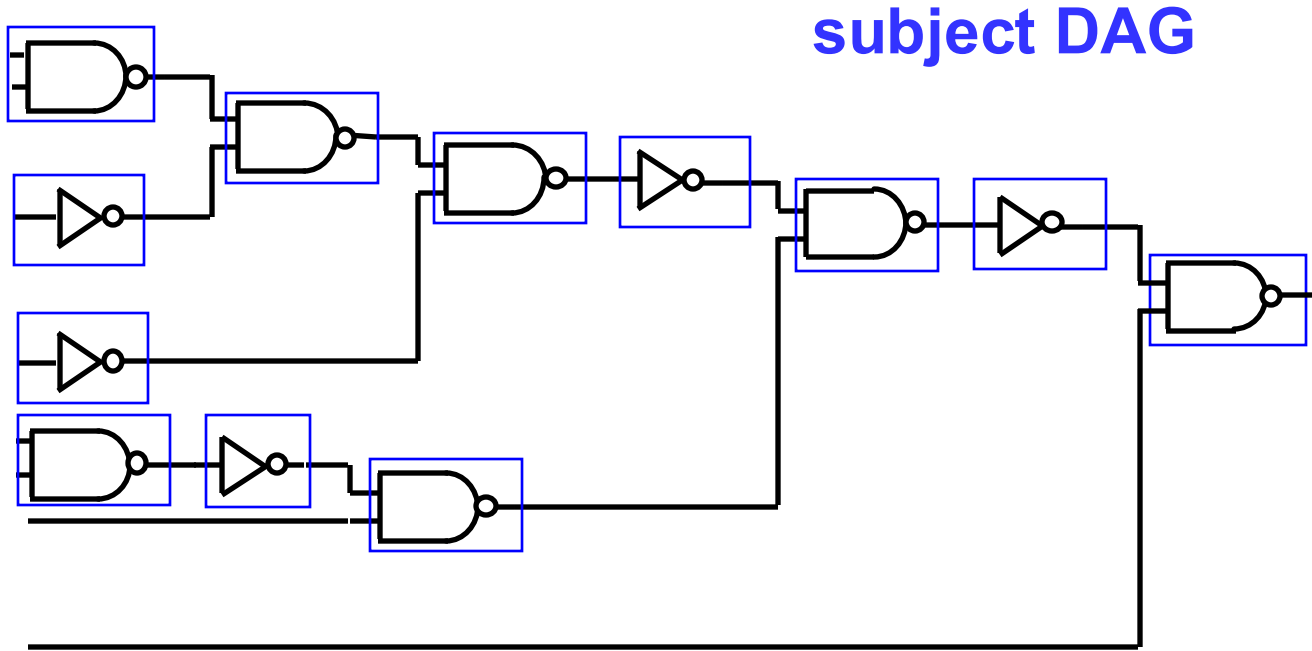
into the technology library (simple example below)



Library Gates

	Element/Area Cost	Tree Representation (normal form)
INVERTER	2 	
NAND2	3 	
NAND3	4 	
NAND4	5 	 

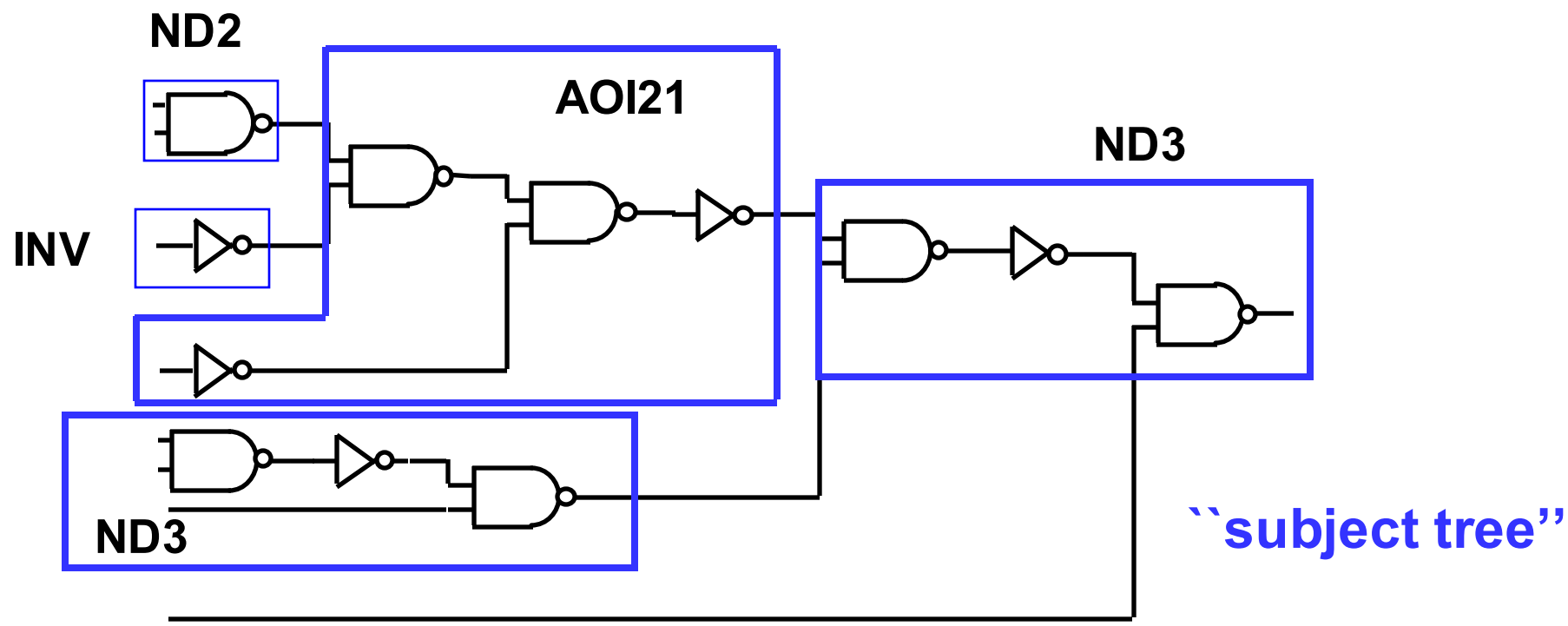
Trivial Covering



7 NAND2 (3) = 21
5 INV (2) = 10
Area cost 31

Can we do better with tree covering?

Optimal Covering



INV	2
ND2	3
2 ND3	8
AOI21	4
	<hr/>

Area cost 17

Summary of Classical Technology Mapping

- DAG covering formulation
 - Separated library issues from mapping algorithm (can't do this with rule-based systems)
- Tree covering approximation
 - Very efficient (linear time)
 - Applicable to wide range of libraries (std cells, gate arrays) and technologies (FPGAs, CPLDs)
- Weaknesses
 - Problems with DAG patterns (Multiplexors, full adders, ...)
 - Large input gates lead to many patterns

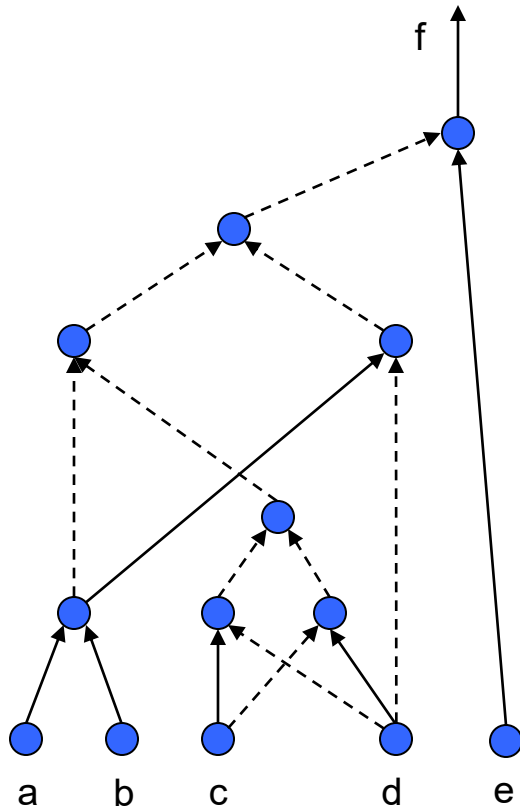
Outline

- Introduction
- Two-level Logic Synthesis
- Multi-level Logic Synthesis
 - Classical
 - Contemporary
- Technology Mapping
 - Classical
 - **Contemporary**

AIG Technology Mapping

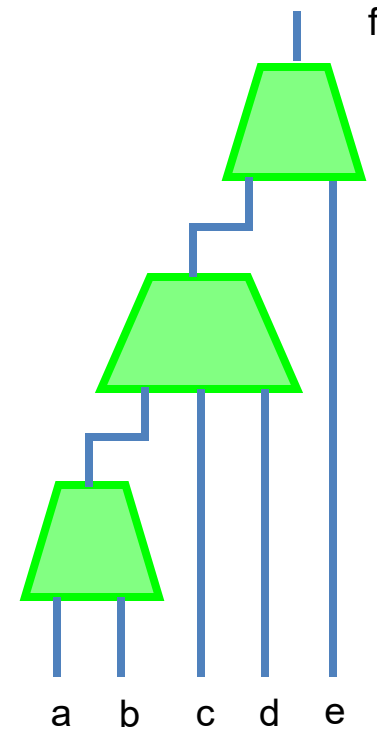
Input: A logic network (And-Inverter Graph)

Output: A netlist of K-LUTs implementing AIG and optimizing some cost function



The subject graph

Technology Mapping



The mapped netlist

Sample ABC Scripts

Area Optimization Script	
Command	Function
strash	transform combo logic into AIG by "structural hashing"
dch	similar to &dch decompose to AND-INV and tech map subject to timing constraints
map -B 0.9	perform std cell mapping of the current network using 2-level logic minimization approach (Boolean matching + resynthesis) -B 0.9 delay multiplier to bias gate selection (0.0 = area mode, 1.0 = delay mode)
topo	rearrange nodes to be in topological order for STA
stime -c	perform STA using liberty library
buffer -c	perform buffering and sizing on mapped network
upsie -c	selectively increase gate sizes on the critical path
dnsize -c	selectively decrease gate sizes while maintaining delay

Delay Optimization Script		
Command	Repeat	Function
&get -n	X1	convert current network into GIA (generalized-iterative-and)
&st		perform structural tech map by considering physical layout
&dch		decompose to AND-INV and tech map subject to timing constraints
&nf		perform tech mapping of the network using network flow algorithm
&st	X5	perform structural tech map by considering physical layout
&syn2		perform secondary delay/area opt subject to timing constraints
&if -g -K 6		perform FPGA tech map based on priority cuts
&synch2		perform 2nd stage sequential opt including retiming and clock gating
&nf	X1	perform tech mapping of the network using network flow algorithm
&put		put current network into memory buffer
buffer -c		perform buffering and sizing on mapped network
topo		rearrange nodes to be in topological order for STA
stime -c		perform STA using liberty library
upsie -c		selectively increase gate sizes on the critical path
dnsize -c		selectively decrease gate sizes while maintaining delay

Comparison of Classical vs. AIG Synthesis

“Classical” synthesis

- Boolean network
- Network manipulation (algebraic)
 - Elimination
 - Factoring/Decomposition
 - Speedup
- Node minimization
 - Espresso
 - Don't cares computed using BDDs
 - Resubstitution
- Technology mapping
 - Tree based

ABC “contemporary” synthesis

- AIG network
- DAG-aware AIG rewriting (Boolean)
 - Several related algorithms
 - Rewriting
 - Refactoring
 - Balancing
 - Speedup
- Node minimization
 - Boolean decomposition
 - Don't cares computed using simulation and SAT
 - Resubstitution with don't cares
- Technology mapping
 - Cut based with choice nodes

40

Slide Courtesy of Alan Mischenko & Bob Brayton

Thank You
