ISEDA’23 Tutorial, 2023/05/08，Nanjing

NINGBO UNIVERSITY

Introduction of Logic Synthesis and Optimization

Speaker: Prof. Zhufei Chu

E-Mail: chuzhufei@nbu.edu.cn

Homgpage: https://zfchu.github.io/

EECS, Ningbo University

ISEDA 2023 International Symposium of EDA

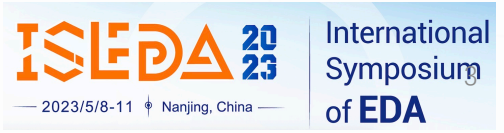
2023/5/8-11 Nanjing, China

Outline

- Introduction
- Logic Representations
- Logic Optimization
- Technology Mapping
- Overview of Logic Synthesis Tools
- Demo
- Conclusions

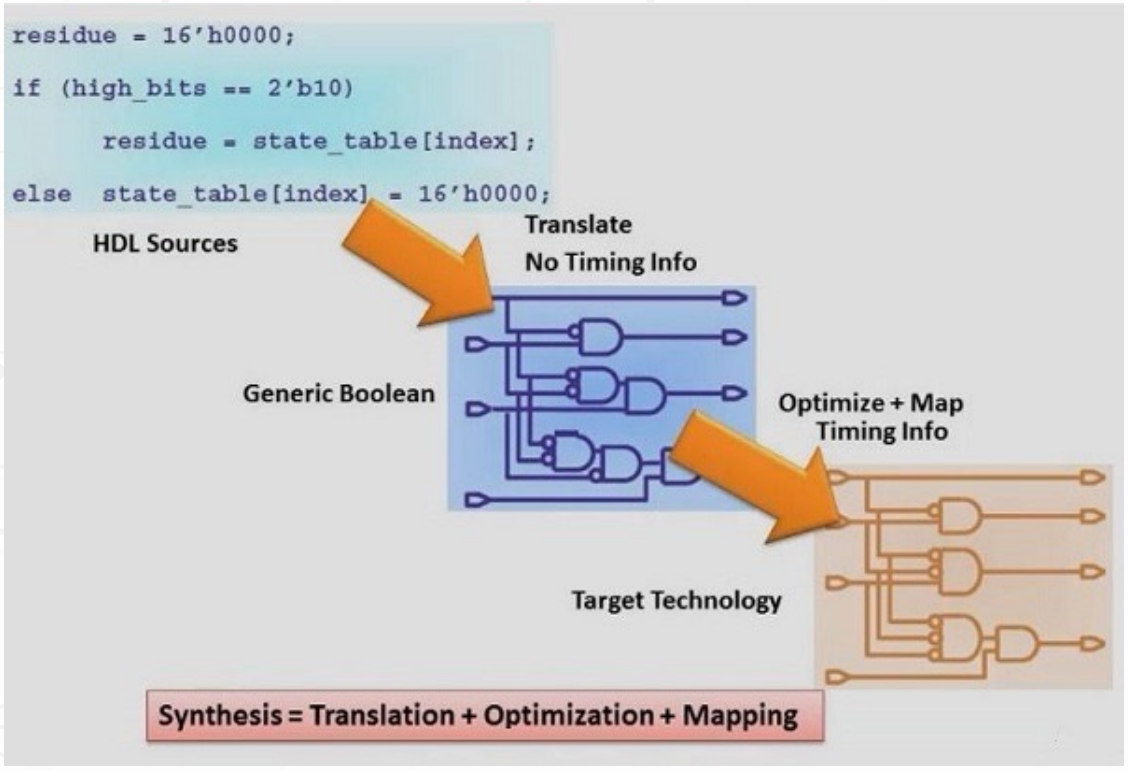
NINGBO UNIVERSITY

Introduction

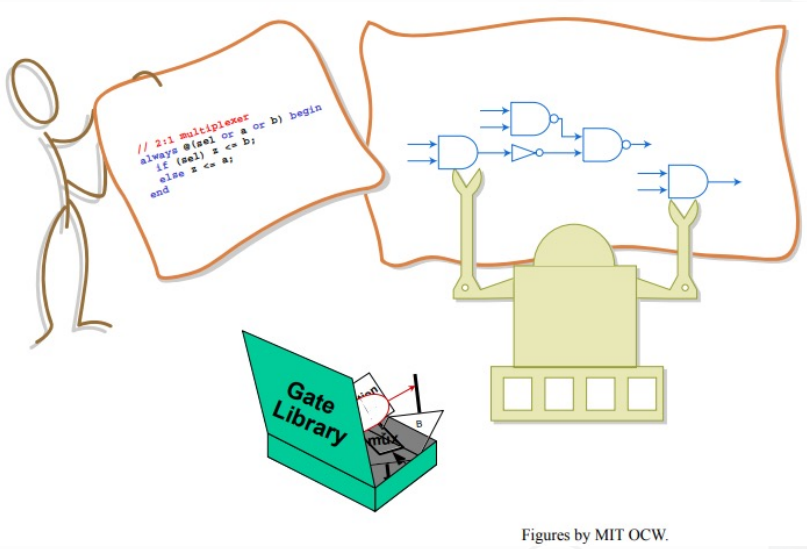


What is logic synthesis?

NINGBO UNIVERSITY

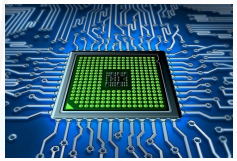


[Courtesy: https://vlsi-backend-adventure.com/logic_synthesis.html]

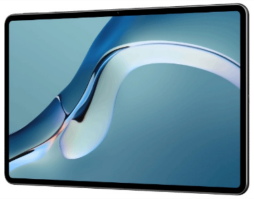


[Courtesy: MIT OCW]

Digital Computing Systems



From circuits to systems



IoT (Internet of Things)

[Courtesy: J. Rabaey]

Design Objective

NINGBO UNIVERSITY

Mobile Computing

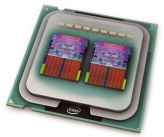
- Low power
- Portable

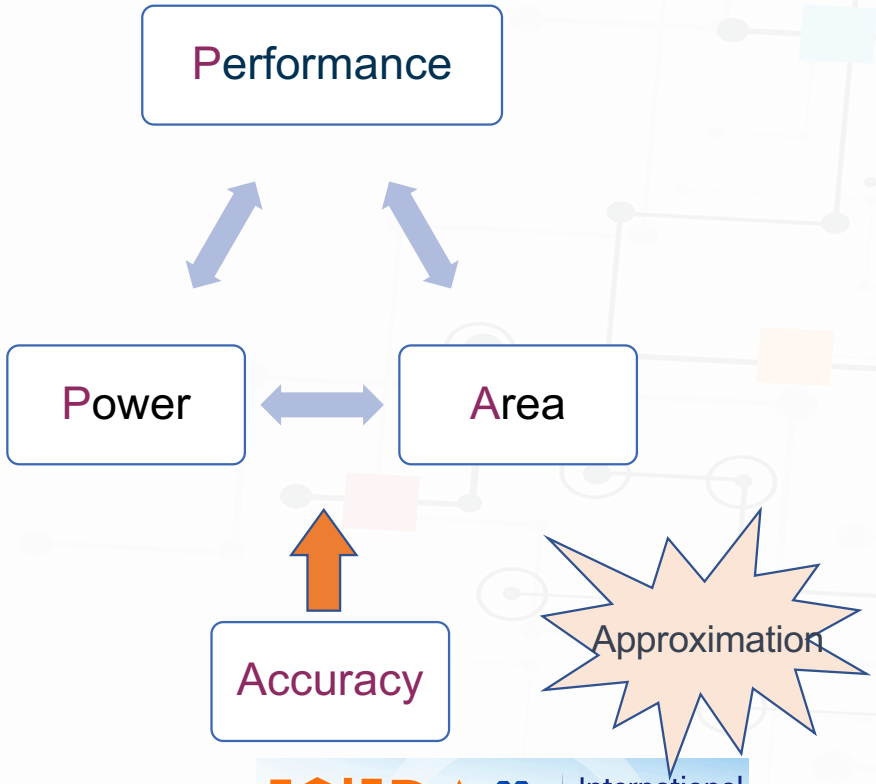
Aeronautics and Astronautics

- High reliability
- Add redundant backups

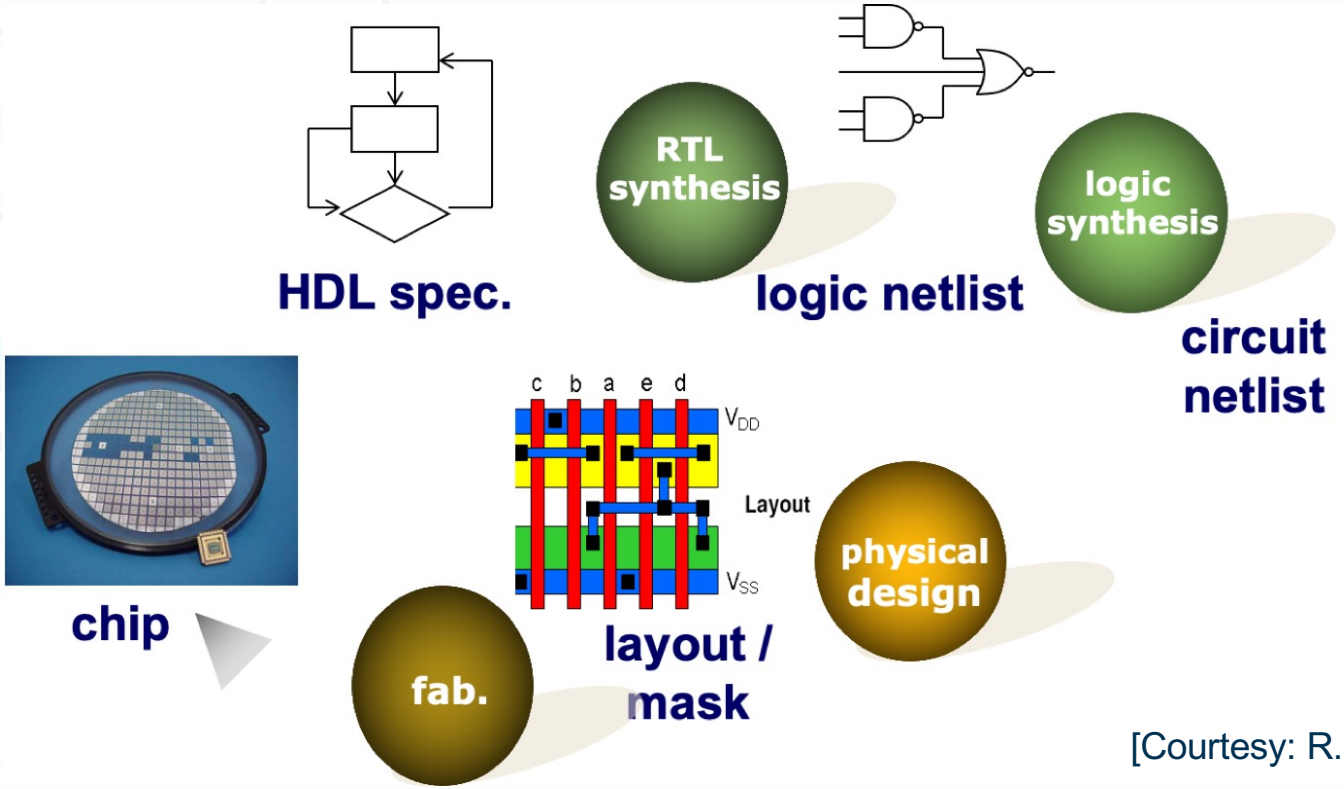
Data center\Cloud computing

- High performance
- Parallel、Multi-core



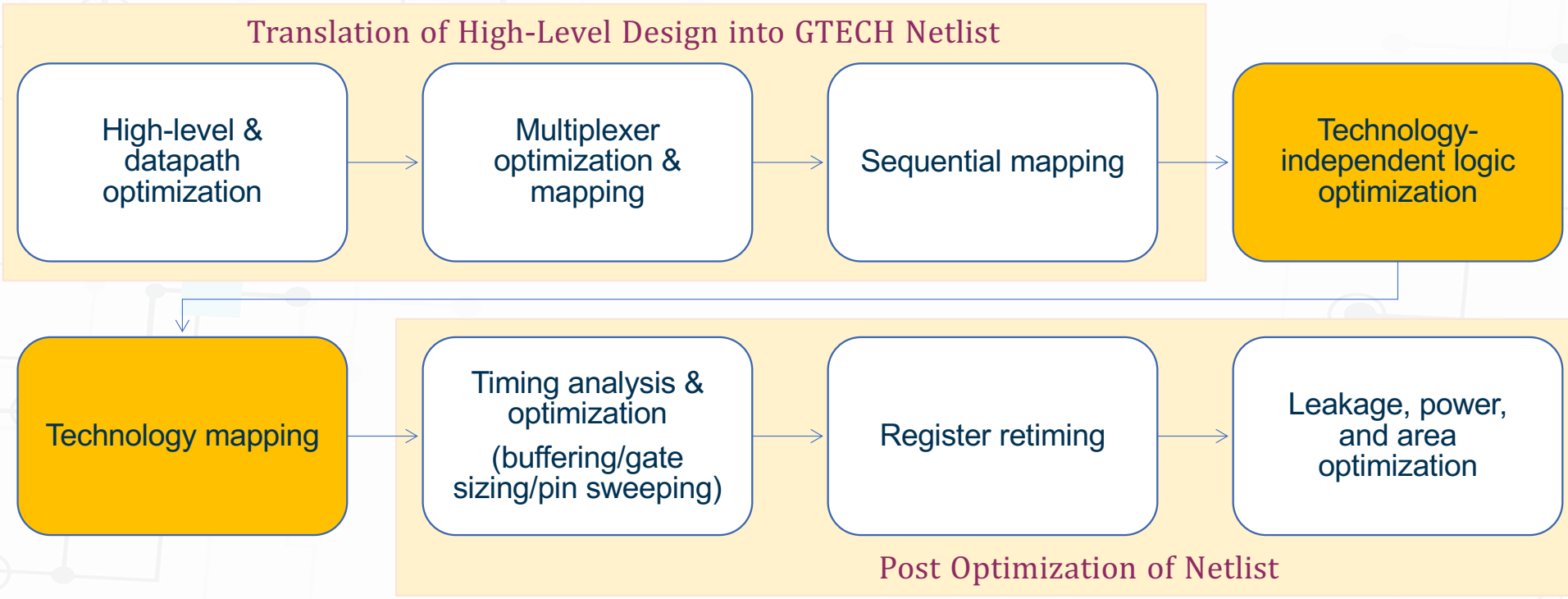


Digital Integrated Circuit Design Overall Process



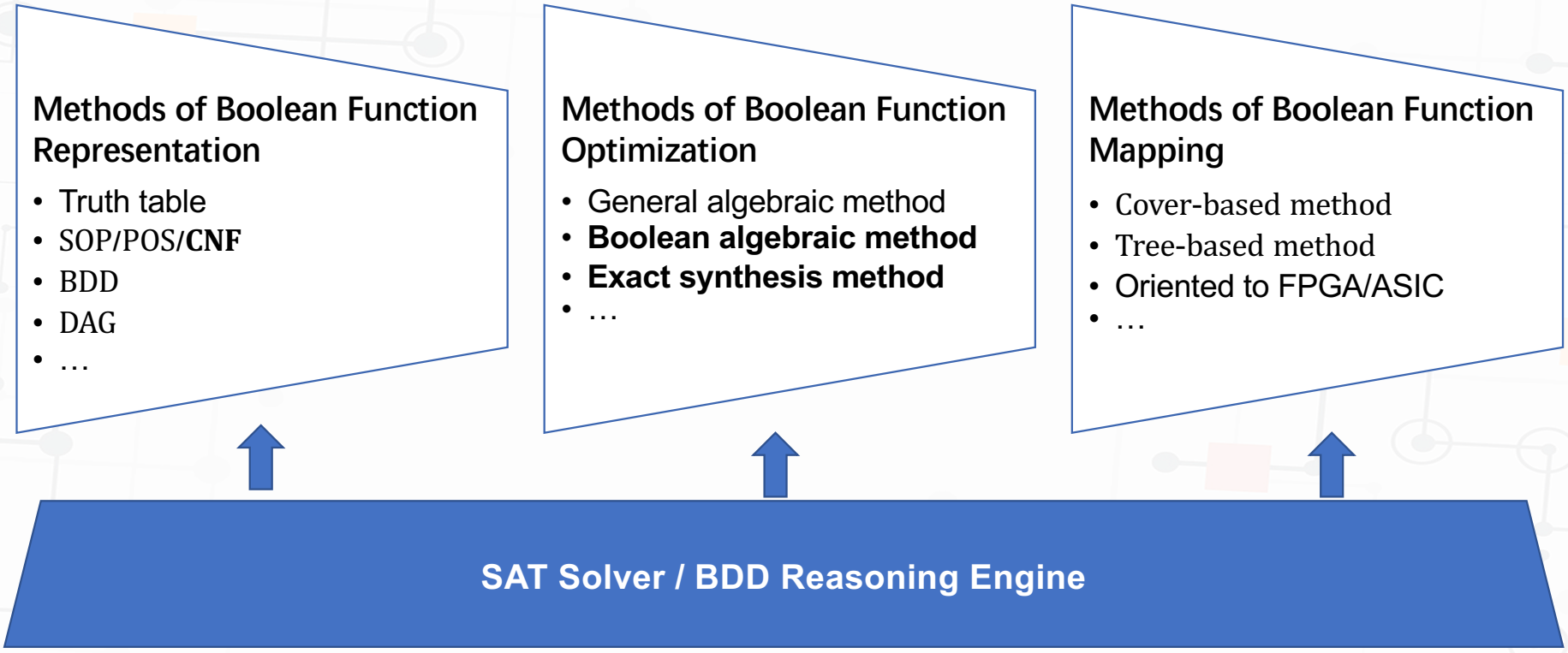
[Courtesy: R. Jiang]

Logic Synthesis Flow

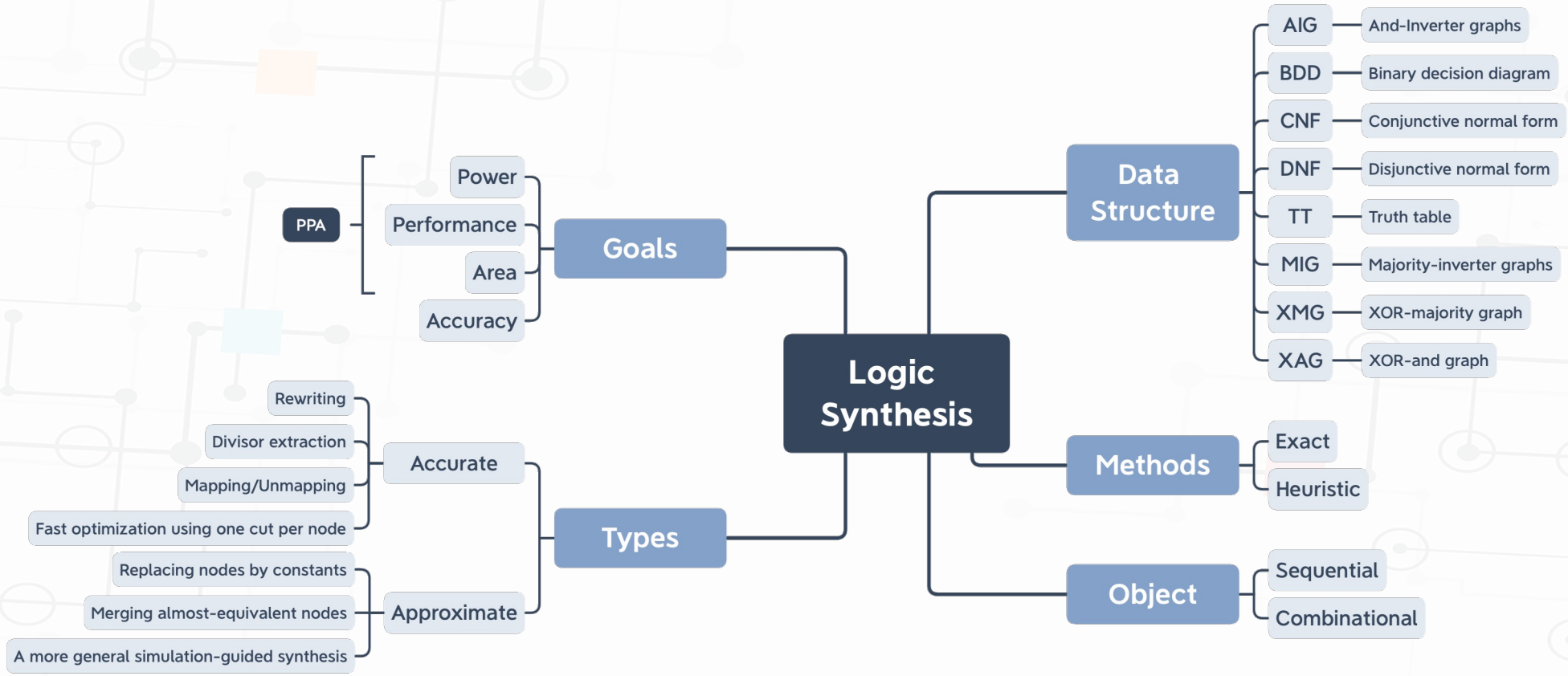


[Courtesy: Dr. Yu Huang, HiSilicon]

Critical Issues in Logic Synthesis Research



Logic Synthesis Technique Map



Logic Synthesis Tools

NINGBO UNIVERSITY

The development history of logic synthesis tools

70's

- Few logic synthesis algorithms and tools existed in the 70's
- Link to place and route for automatic design
  - Innovative methods at IBM, Bell Labs, Berkeley, Stanford

Early 80's

- First prototype synthesis tools in the early 80's
  - YLE (Yorktown Logic Editor) [Brayton]
  - MIS（Multi-level Interactive System）[Berkeley]
  - Espresso [IBM+Berkeley]

Late 80's

- First logic synthesis companies in the late 80's
  - Synopsys and others

Logic Synthesis Tools

NINGBO UNIVERSITY

- Commercial logic synthesis tools

Design Compiler (DC) [Synopsys]

Vivado [Xilinx]

Fusion Compiler [Synopsys]

Genus [Cadence]

90’s 00’s 10’s

SIS [Berkeley]

ABC [Berkeley]

Yosys [Clifford Wolf]

The EPFL Logic Synthesis Libraries [EPFL]

CIRKIT [EPFL]

ALSO [Ningbo University]

- Academic logic synthesis tools

open source

宁波大学
NINGBO UNIVERSITY

Logic Representations

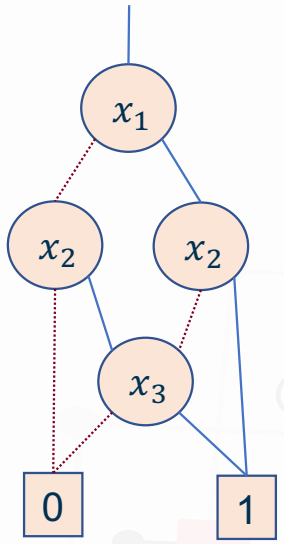
Logic representation zoo

- $f = x_1x_2 + x_2x_3 + x_1x_3$
  - Sum-of-Products, SOP
  - Disjunctive Normal Form, DNF
- $f = (x_1+x_2)(x_2 + x_3)(x_1 + x_3)$
  - Product-of-Sums, POS
  - Conjunctive Normal Form, CNF
- $f = (1110\ 1000)_2 = 0xe8$ (Truth tables, TT)
- BDD
- AIG
- 。。

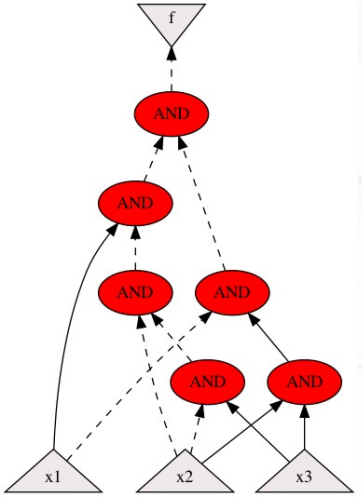
x_1	x_2	x_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

8

e



BDD
(Binary Decision Diagram)



AIG
(AND-Inverter Graph)

Truth tables

Truth table representation is

- intractable for large n
- canonical

Example:

$$f = abc\bar{d} + ab\overline{cd} + a\bar{b}c\bar{d} + \bar{a}bc\bar{d} + \bar{a}b\overline{cd} + \overline{a\bar{b}}cd + \overline{abc}d + \overline{abcd}$$

a	b	c	d	f	
0	0	0	0	0	a
0	0	0	1	1	
0	0	1	0	0	
0	0	1	1	1	
0	1	0	0	0	2
0	1	0	1	1	
0	1	1	0	0	
0	1	1	1	0	

a	b	c	d	f	
1	0	0	0	0	a
1	0	0	1	1	
1	0	1	0	0	
1	0	1	1	1	
1	1	0	0	0	e
1	1	0	1	1	
1	1	1	0	1	
1	1	1	1	1	

SOP

Example:

$$ab\bar{c} + \bar{a}bd + \bar{b}\bar{d} + \bar{b}\bar{e}f$$

Advantages:

- Easy to manipulate and minimize
- Many algorithms available
- Two-level theory applies

Disadvantages:

- Not representative of logic complexity. For example:

$$f = ad + ae + bd + be + cd + ce \qquad \bar{f} = \bar{a}\bar{b}\bar{c} + \bar{d}\bar{e}$$

- Not easy to estimate logic size and performance
- Difficult to estimate progress during logic manipulation

The two differ in their implementation by an inverter

POS

NINGBO UNIVERSITY

- Product of Sum (POS) representation of Boolean function
- Describes solution using a set of constraints
  - very handy in many applications because new constraints can just be added to the list of existing constraints
  - very common in AI community

Example:

$$f = (a + \bar{b} + c)(\bar{a} + b + c)(a + \bar{b} + c)(a + b + c)$$

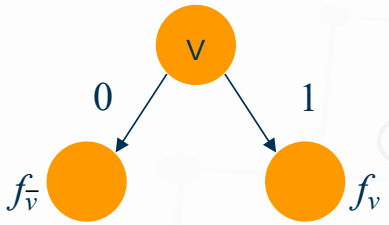
- SAT on CNF (POS) <=> Tautology on DNF (SOP)

Binary Decision Diagrams (BDD)

NINGBO UNIVERSITY

- General idea: Representation of a logic function as graph (DAG)
  - S. B. Akers proposed in 1978, R.E Bryant extended in1986 (ROBDD)
  - Based on recursive Shannon expansion
    - one root node, two terminals 0, 1
    - each node, two children, and a variable

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



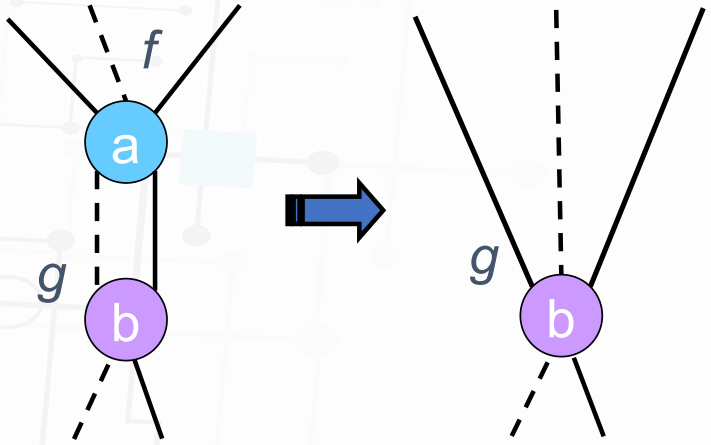
- Reduced and ordered (ROBDD)
  - Reduced:
    - any node with two identical children is removed
    - two nodes with isomorphic BDD's are merged
  - Ordered:
    - Co-factoring variables (splitting variables) always follow the same order along all paths

Reduced ordered BDD

NINGBO UNIVERSITY

BDD Reduction Rules -1

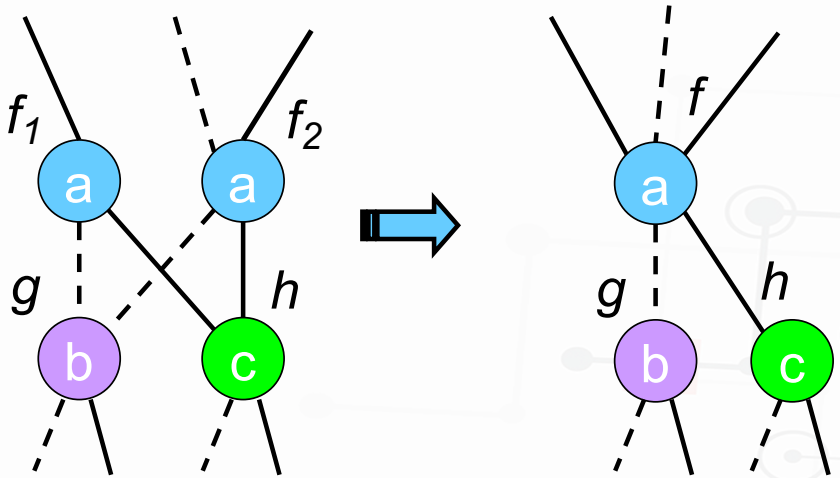
- with both edges pointing to same node



$$f = ag(b) + \bar{a}g(b) = g(b)$$

BDD Reduction Rules -2

- Merge duplicate nodes



$$f_1 = \bar{a}g(b) + ah(c) = f_2 \qquad f = f_1 = f_2$$

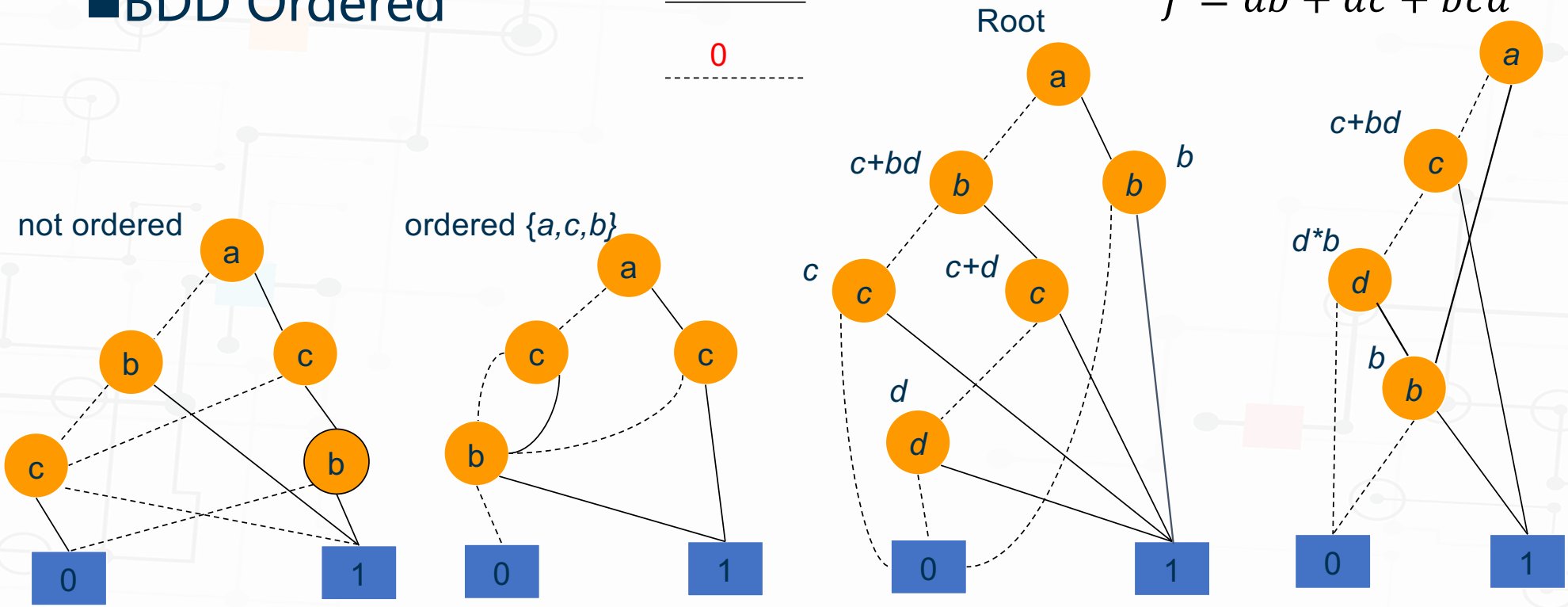
BDD Example

- BDD Ordered

1

0

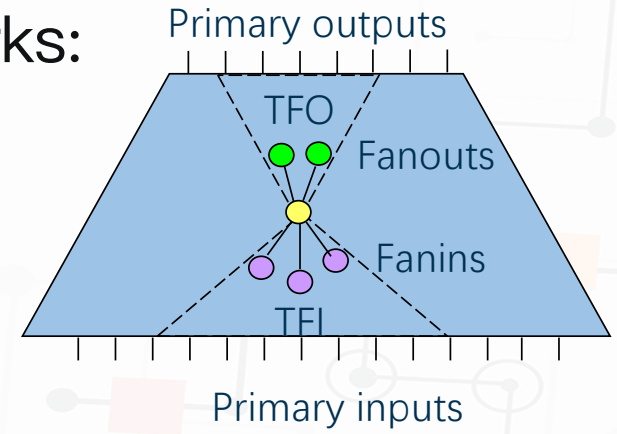
$f = ab + \bar{a}c + b\bar{c}d$

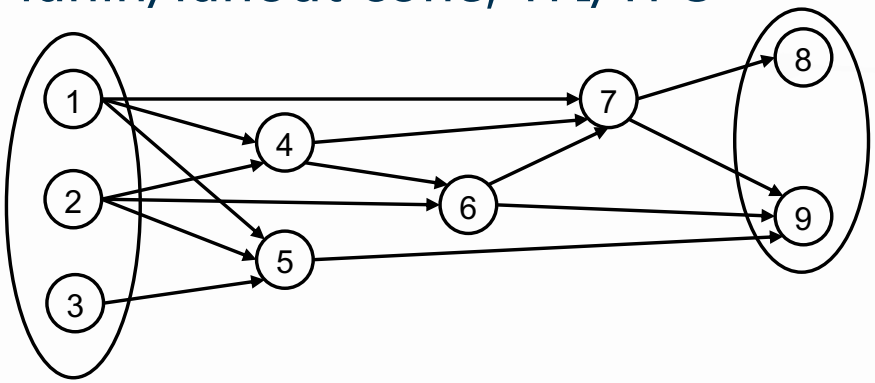


Logic network

NINGBO UNIVERSITY

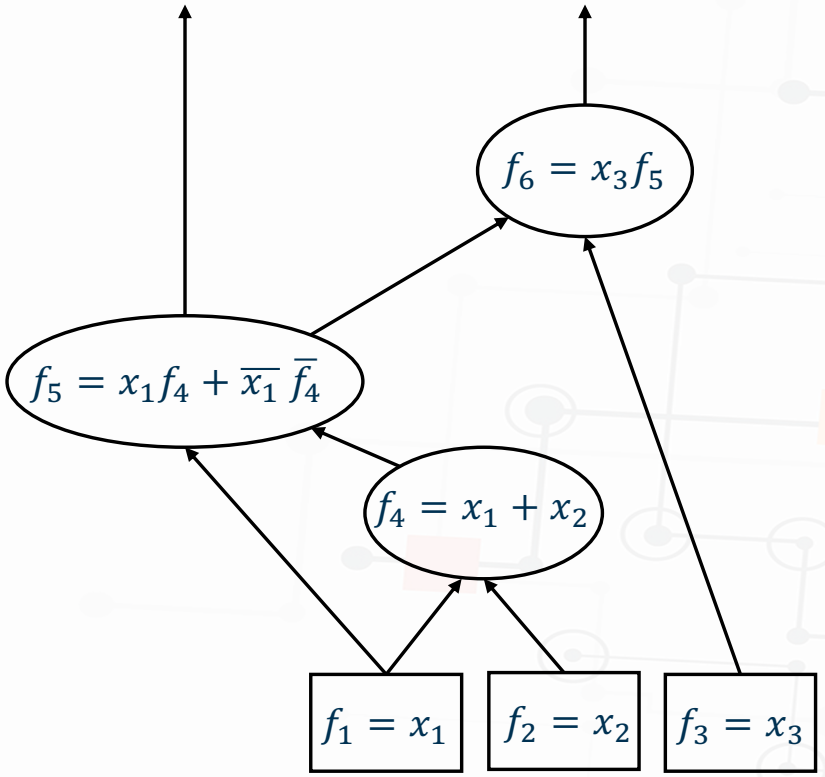
- Logic Network is a generic representation of a logic circuit.
- Several common elements of logical networks:
  - Primary Input/Output, PI/PO
  - Node
  - Fanin/Fanout, FI/FO
  - Transitive fanin/fanout cone, TFI/TFO





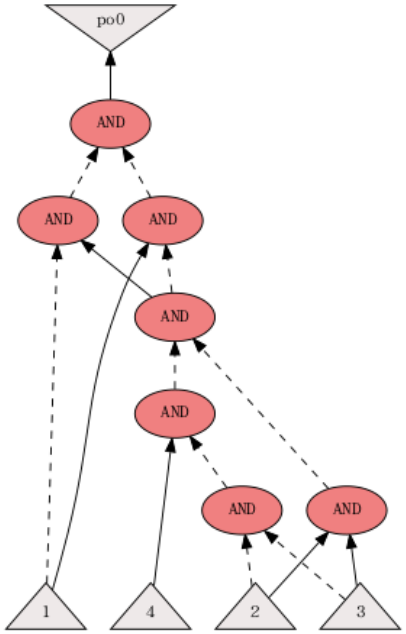
Logic network

- $f_4 = x_1 + x_2$
- $f_5 = x_1(x_1 + x_2) + \overline{x_1}\overline{(x_1 + x_2)}$
- $f_5 = x_1 f_4 + \overline{x_1}\overline{f_4}$
- $f_6 = x_3(x_1 f_4 + \overline{x_1}\overline{f_4})$

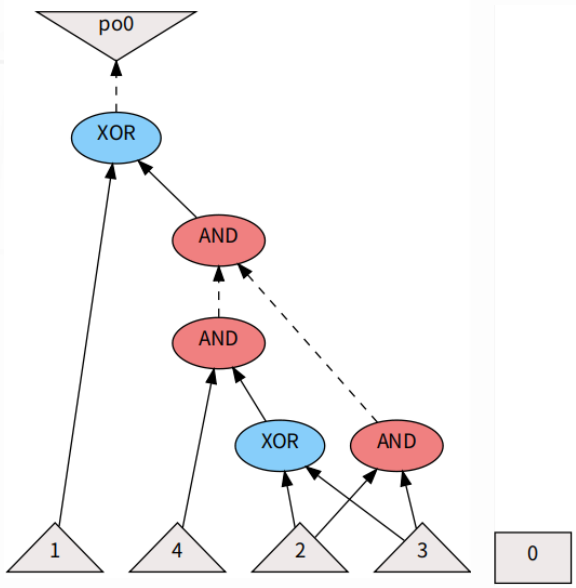


DAGs

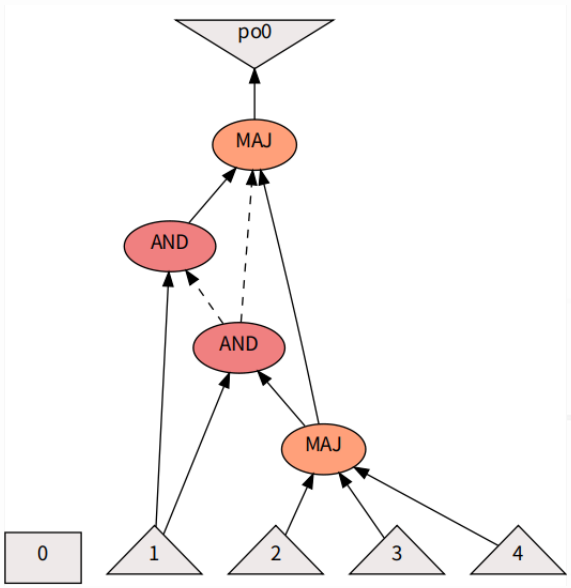
$$f = a \oplus (bc + cd + bd)$$



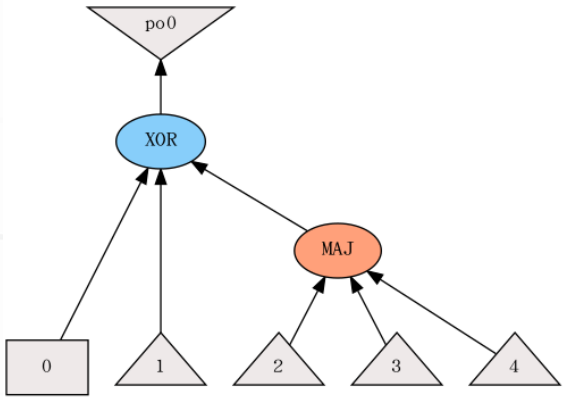
AIG



XAG

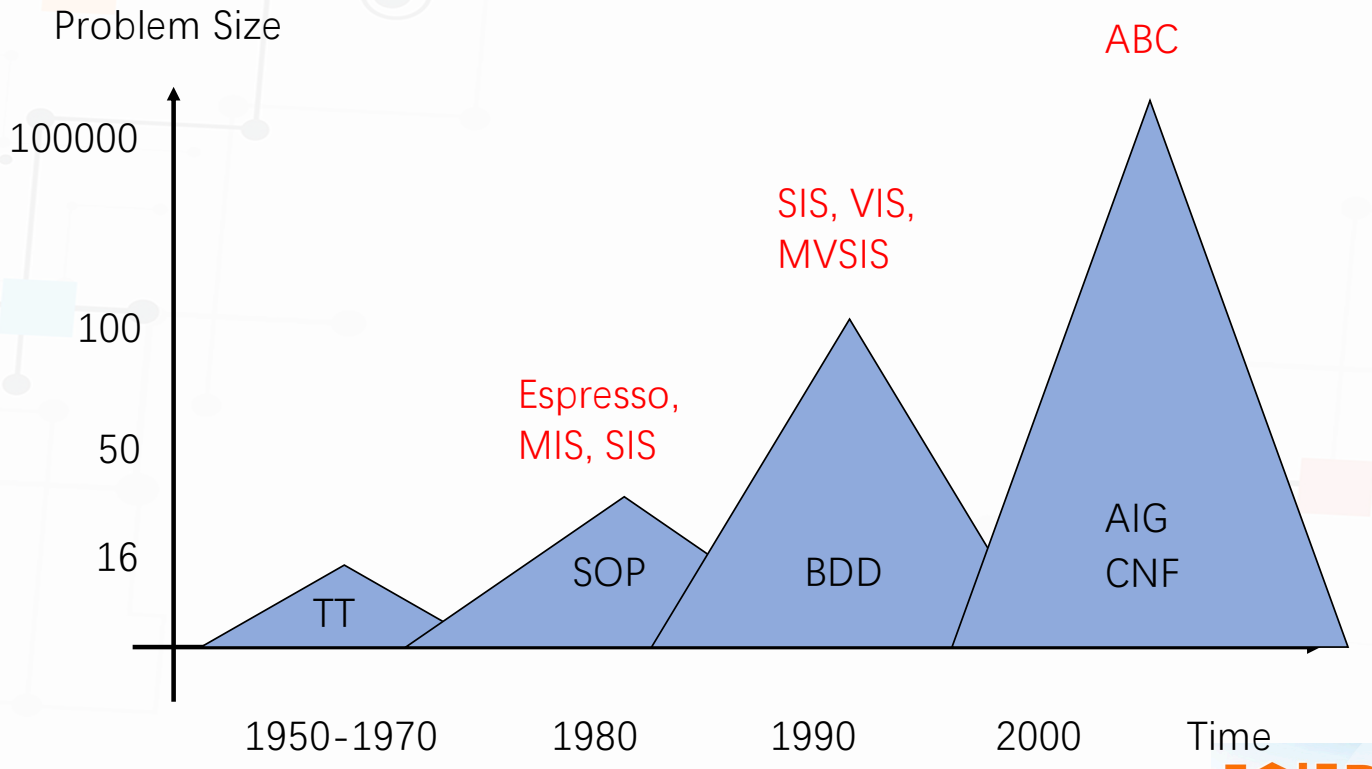


MIG

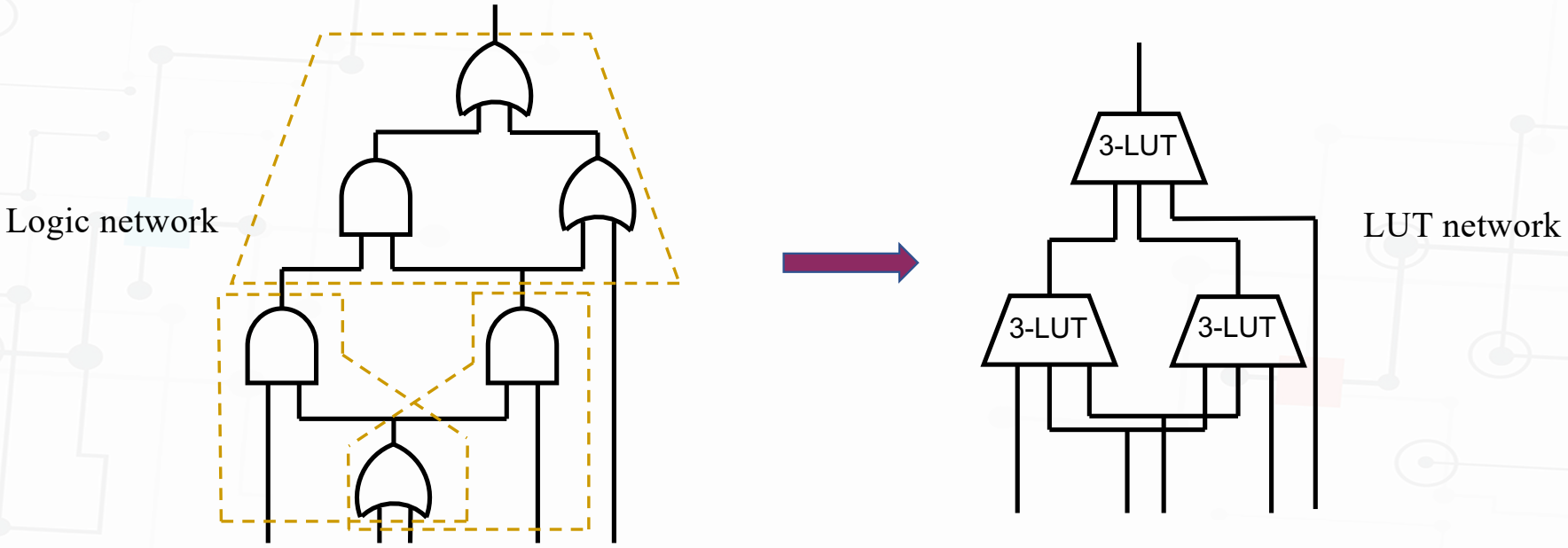


XMG

Historical perspective

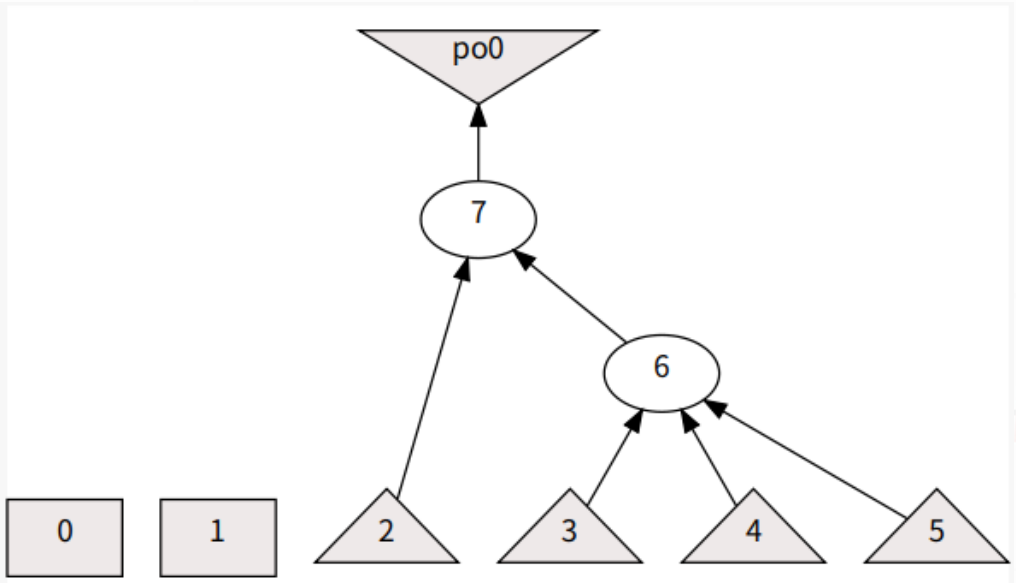


LUT network



LUT network

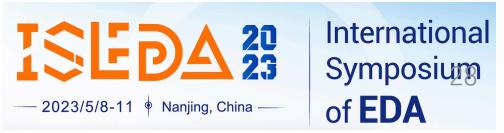
$$f = a \oplus (bc + cd + bd)$$



LUT

NINGBO UNIVERSITY

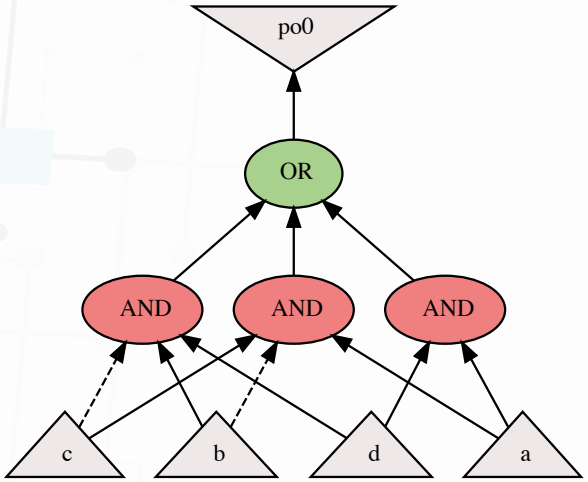
Logic Optimization



Two-level vs multi-level

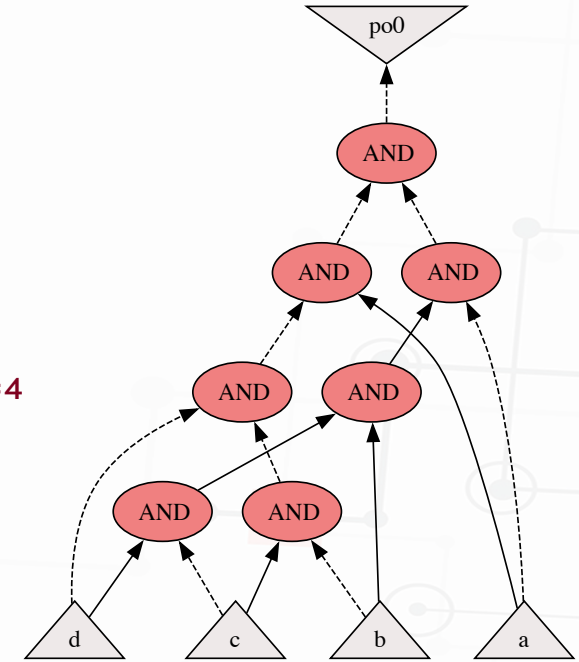
$$f = a\bar{b}c + ad + b\bar{c}d$$

depth=2



two-level

depth=4



multi-level

Two-level logic synthesis

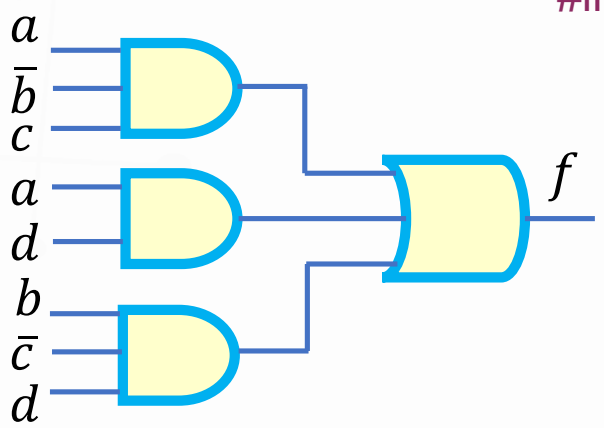
NINGBO UNIVERSITY

■ $f = a\bar{b}c + ad + b\bar{c}d$ logic expression based on SOP

3 product implicants , 8 literals

#implicants → number of rows in the PLA

#literals → number of transistors



multiple AND gates in the first level

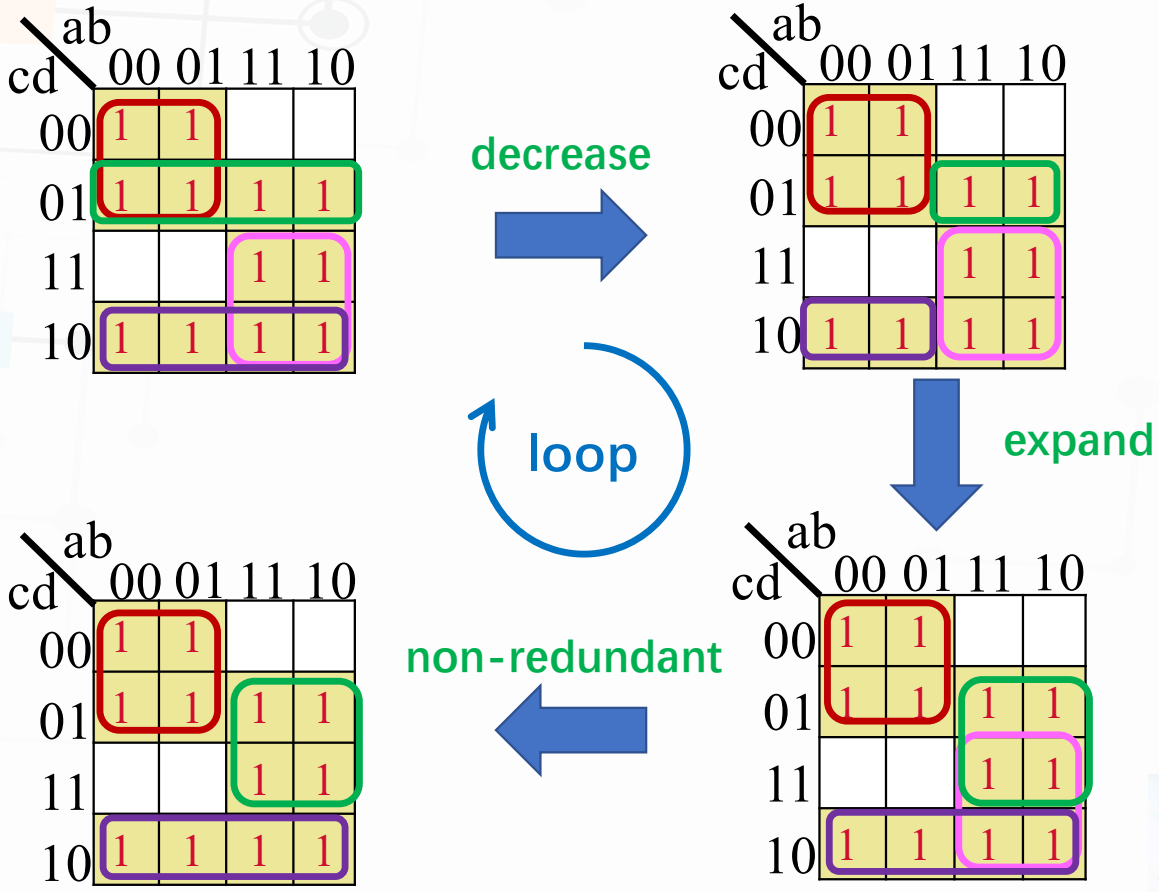
a large OR gate in the second level

Quine-McCluskey Algorithm

- Minimization of a second-order logic formula
  - Reducing the SOP expression to its minimal literals
  - Willard Quine proposed it in 1952, Edward J. McCluskey expanded it in 1956.
- Base idea
  - Combining adjacent minterms and eliminating redundant factors, e.g.: $xy + x\bar{y} = x$
  - A computable version of the Karnaugh map that is easy to implement in programming
  - Simplification applicable to any complex logic function

Two-level logic synthesis

■Espresso

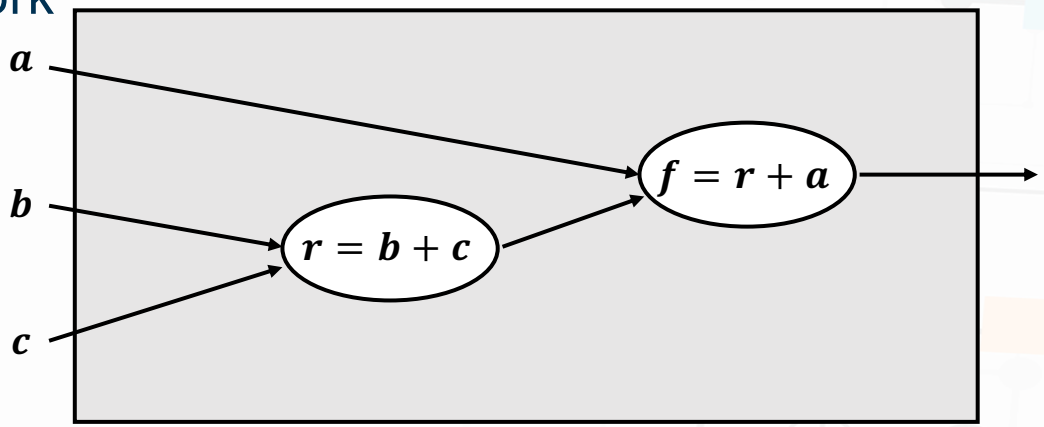


Multi-level logic synthesis

- Elimination
- Decomposition
- Extraction
- Simplification
- Substitution
- …

Elimination

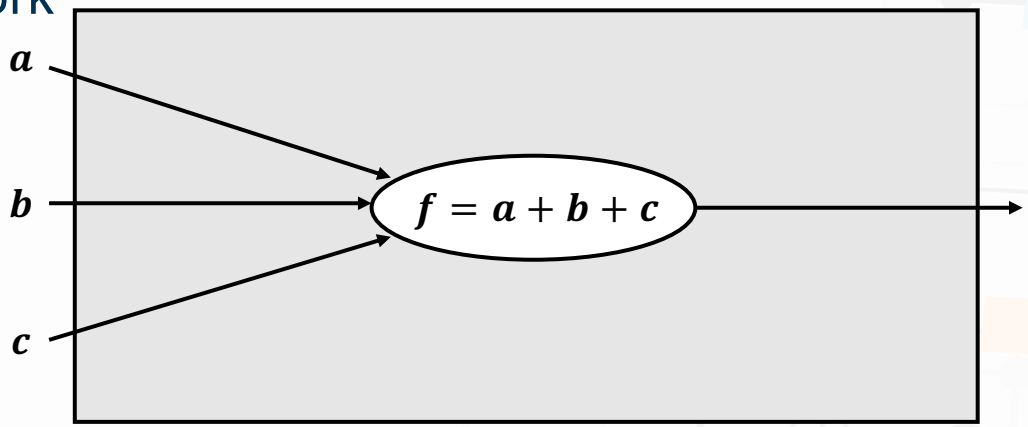
- Eliminate a function from the network
  - Similar to Gaussian elimination
- Perform variable substitution



- Example:
  - $f = r + a;\ r = b + c;$
  - $f = a + b + c;$

Elimination

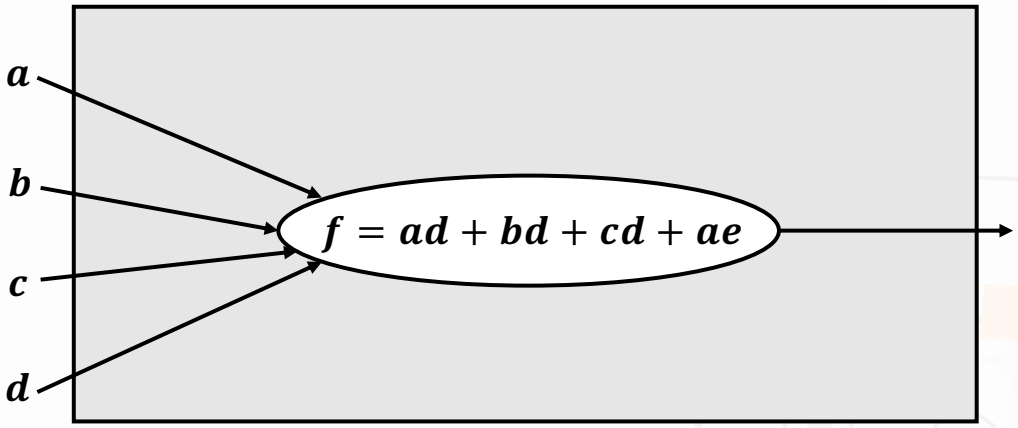
- Eliminate a function from the network
  - Similar to Gaussian elimination
- Perform variable substitution



- Example:
  - $f = r + a;\ r = b + c;$
  - $f = a + b + c;$

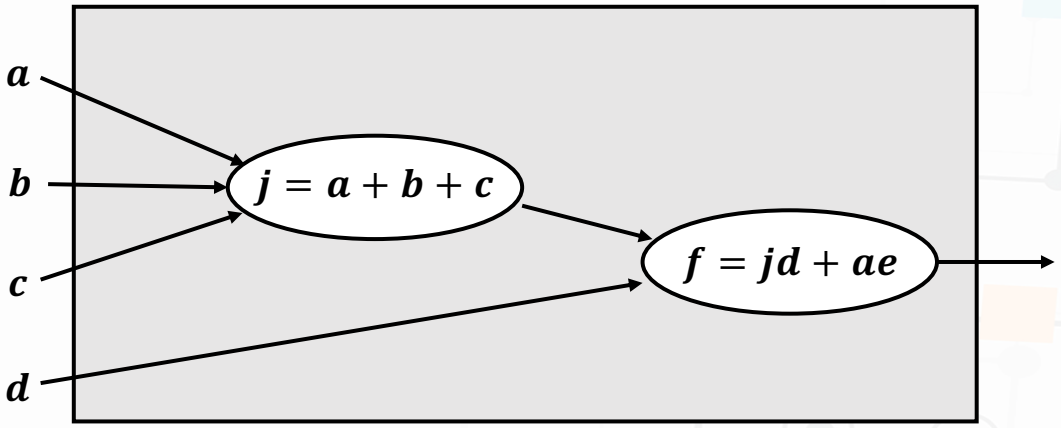
Decomposition

- Decompose a function into smaller functions
  - Opposite to elimination
- Introducing new variables into the network
- Example:
  - $f = ad + bd + ac;$
  - $j = a + b;\ f = jd + ac;$



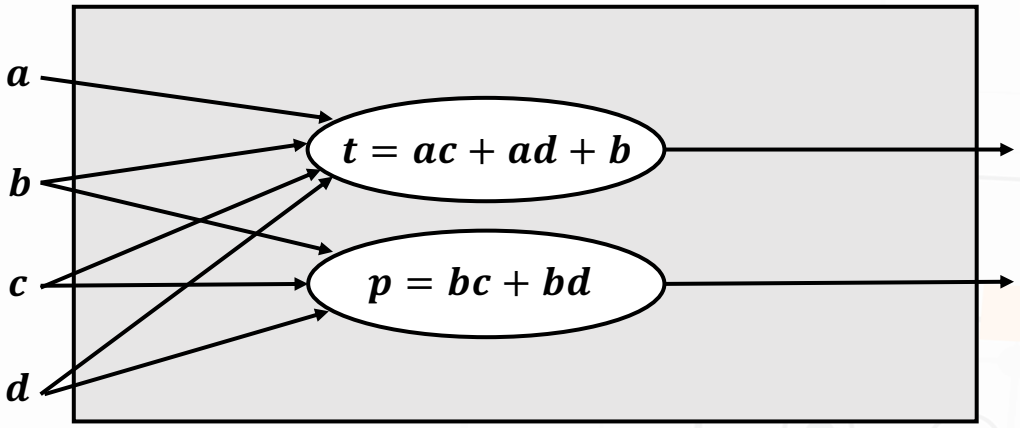
Decomposition

- Decompose a function into smaller functions
  - Opposite to elimination
- Introducing new variables into the network
- Example:
  - $f = ad + bd + ac;$
  - $j = a + b;\ f = jd + ac;$



Extraction

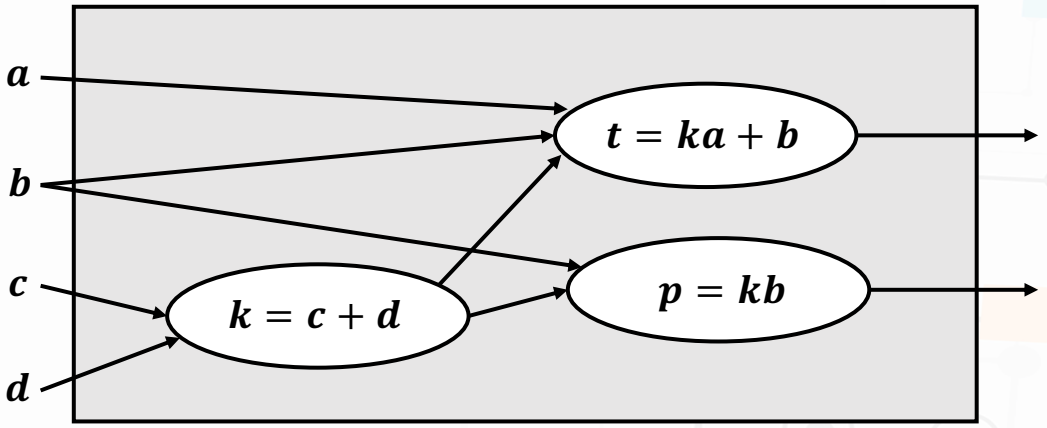
- Finding common sub-expressions between two (or more) expressions
  - Extracting new sub-expressions as a new function
  - Introducing new blocks into the circuit



- Example:
  - $p = bc + bd; t = ac + ad + b;$
  - $p = (c + d)\ b; t = (c + d)\ a + b;$
  - $k = c + \ d; p = kb;\ t = ka + b;$

Extraction

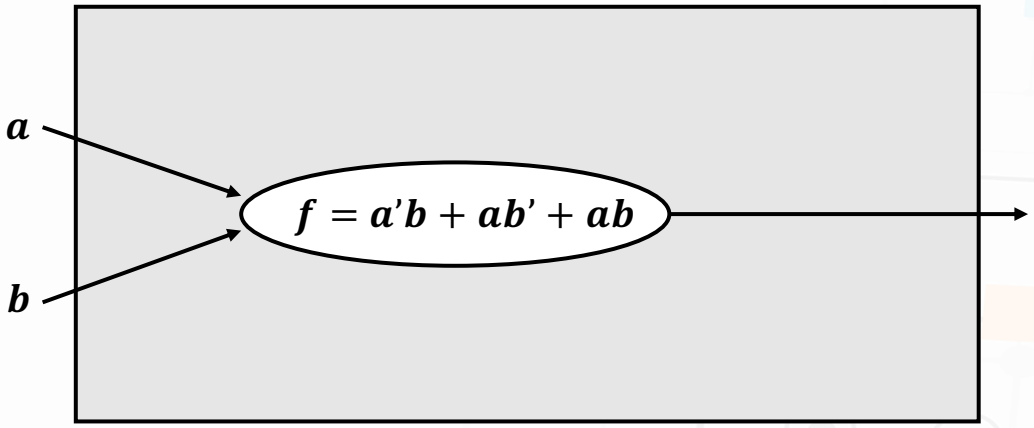
- Finding common sub-expressions between two (or more) expressions
  - Extracting new sub-expressions as a new function
  - Introducing new blocks into the circuit



- Example:
  - $p = bc + bd; t = ac + ad + b;$
  - $p = (c + d)\ b; t = (c + d)\ a + b;$
  - $k = c + \ d; p = kb;\ t = ka + b;$

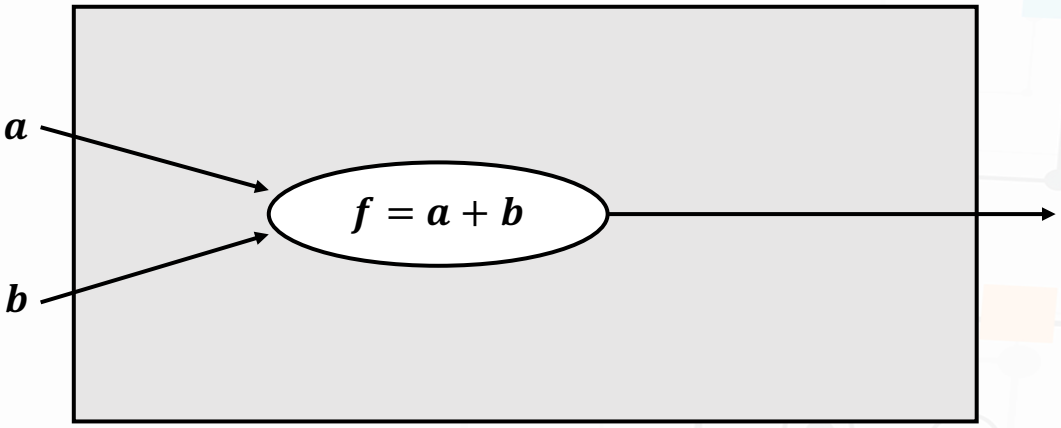
Simplification

- Simplifying local functions
  - Using heuristic minimization techniques, such as Espresso
  - Modifying the fanin of the target node
- Example:
  - $f = a'b + ab' + ab;$
  - $f = a + b;$



Simplification

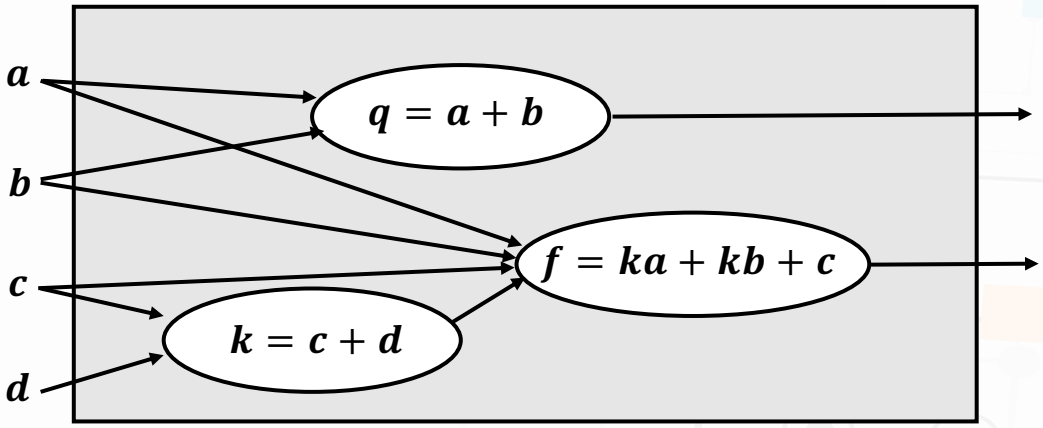
- Simplifying local functions
  - Using heuristic minimization techniques, such as Espresso
  - Modifying the fanin of the target node
- Example:
  - $f = a'b + ab' + ab;$
  - $f = a + b;$



Substitution

NINGBO UNIVERSITY

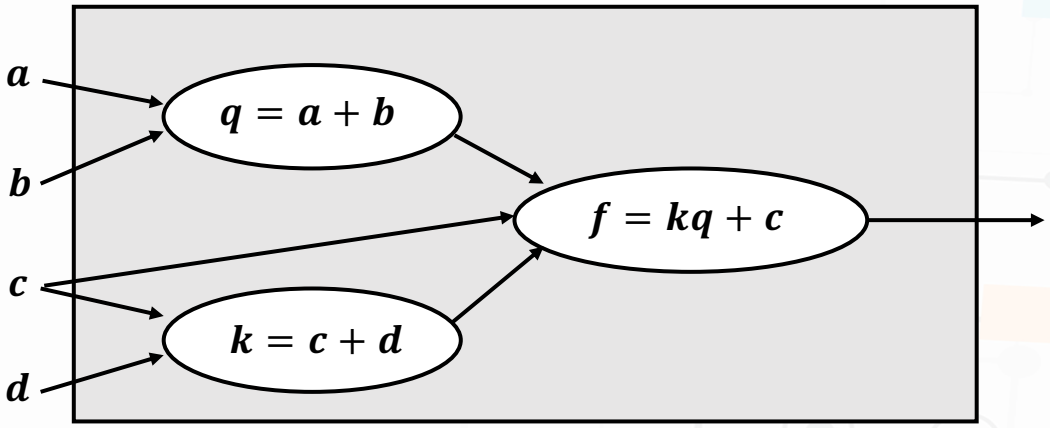
- Simplifying local functions by introducing new variables



- Example:
  - $f = ka + kb + c$;
  - $f = kq + c$;
  - Because $q = a + b$ is already part of the network

Substitution

- Simplifying local functions by introducing new variables



- Example:
  - $f = ka + kb + c$;
  - $f = kq + c$;
  - Because $q = a + b$ is already part of the network

Optimization approach

- Algorithmic approach
  - Define an algorithm for each transformation type
  - Algorithm is an *operator* on the network
  - Algorithms are sequenced by *scripts*
- Rule-based approach
  - Rule data base
    - Set of pattern pairs
  - Pattern replacement is driven by rules
- AI-based approach
  - AI explores *combinations*(*scripts*) of existing optimizations
  - AI-based novel optimization method
- Most modern tools use the algorithmic approach to synthesis, even though rules are used to address specific issues, AI has a better QoR

Exact synthesis

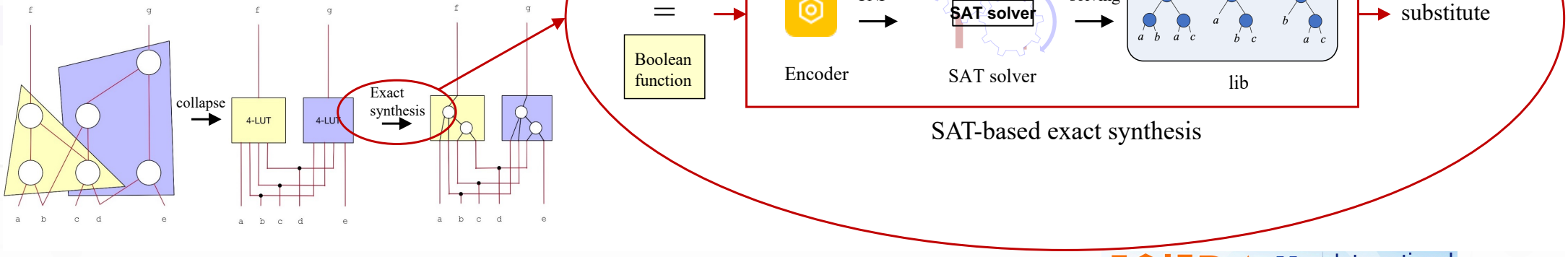
NINGBO UNIVERSITY

- Exact synthesis refers to synthesizing logic representations that exactly meets a specification.
  - It is not used in opposition to approximate synthesis.
  - Exact synthesis can be implemented in several methods
    - Boolean Satisfiability (SAT)
    - Integer Liner Programming (ILP)
    - Satisfiability modular theory (SMT)
    - …
- We may ask "Does there exist a logic network *N* such that *N* implements *f* with exactly *r* gates?"

SAT-based exact synthesis

NINGBO UNIVERSITY

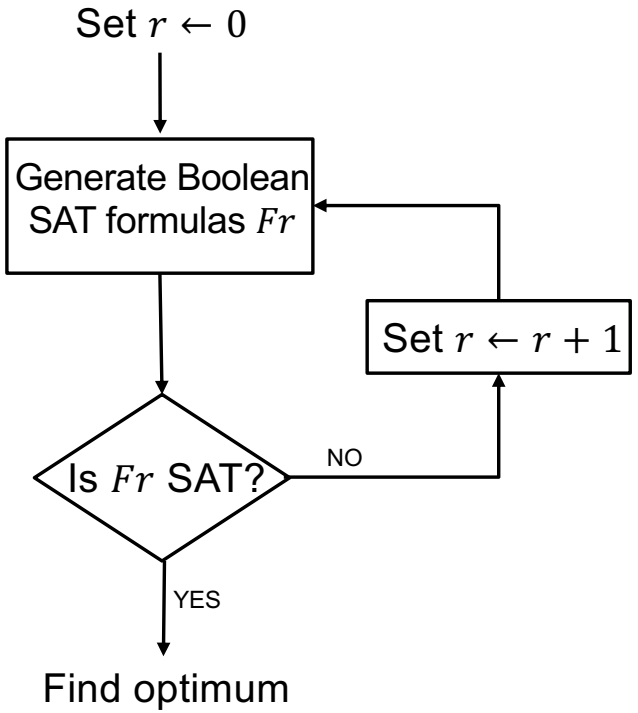
- SAT-based exact synthesis essentially solves the **two** tasks
  - Find logic network topology (**DAG**)
  - Assign **Boolean operators** to vertices



Exact synthesis algorithm

NINGBO UNIVERSITY

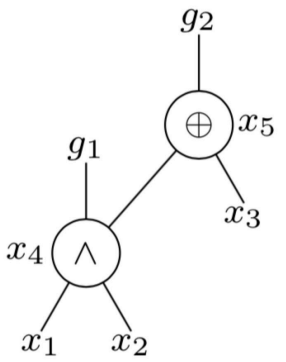
- Given a Boolean function *f*, for size-optimum synthesis, take the SAT encoding method as an example:
  - Start with $r \leftarrow 0$
  - Encode the question as SAT formulas F_r
  - Feed the formulas to a SAT solver and wait for the its result
  - It the result it SAT, we obtained an optimal result, otherwise, we set $r \leftarrow r + 1$ and continue the encoding
- Hence, the size-optimum problem can be solved by a sequence of SAT formulas.



Exact synthesis - Encode

■Variables

Simulation:	x_{it} :	t^{th} bit of x_i's truth table
Output:	g_{hi} :	$[g_h = x_i]$
Selection:	s_{ijk} :	$[x_i = x_i \circ_i x_k]$ for $1 \leq j < k < i$
Operands:	f_{ipq} :	$\circ_i(p, q)$ for $0 \leq p, q \leq 1, p + q > 0$



■Running example

t	=	7	6	5	4	3	2	1
x_{4t}	=	1	0	0	0	1	0	0
x_{5t}	=	0	1	1	1	1	0	0

k	=	2	3	4
s_{41k}	=	1	0	
s_{42k}	=		0	
s_{51k}	=	0	0	0
s_{52k}	=		0	0
s_{53k}	=			1

$g_{14} = 1,\ g_{15} = 0,\ g_{24} = 0,\ g_{25} = 1$

p, q	=	0,1	1,0	1,1
f_{4pq}	=	0	0	1
f_{5pq}	=	1	1	0

D. E. Knuth. *The Art of Computer Programming, Volume 4, Fascicle 6: Satisfiability.* Addison-Wesley, 2015.

Exact synthesis - Encode

NINGBO UNIVERSITY

Main clauses

$$((s_{ijk} \wedge (x_{it} \oplus \bar{a}) \wedge (x_{jt} \oplus \bar{b}) \wedge (x_{kt} \oplus \bar{c})) \mapsto (f_{ibc} \oplus \bar{a}))$$ Constrain the right Boolean operation

$$(\bar{s}_{ijk} \vee (x_{it} \oplus a) \vee (x_{jt} \oplus b) \vee (x_{kt} \oplus c)) \vee (f_{ibc} \oplus \bar{a}))$$

$$(\bar{g}_{hi} \vee (\bar{x}_{it} \oplus g_h(t_1, \ldots, t_n)))$$ Constrain the output value they point to

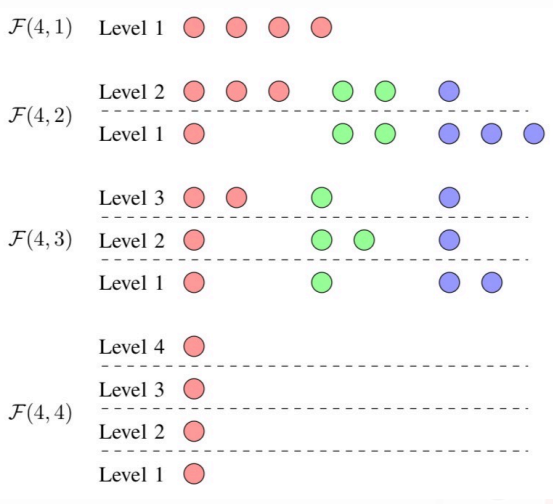
$$\bigvee_{i=n+1}^{n+r} g_{hi}$$ Each output is realized by the network

$$\bigvee_{k=2}^{i-1} \bigvee_{j=1}^{k-1} s_{ijk}$$ Each gate has exactly two inputs

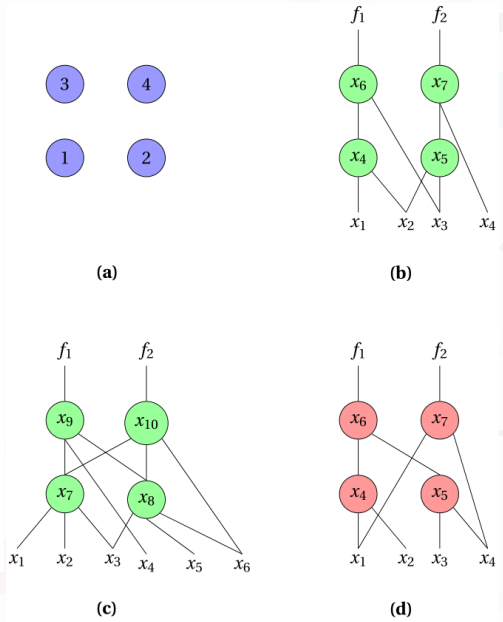
D. E. Knuth. *The Art of Computer Programming, Volume 4, Fascicle 6: Satisfiability.* Addison-Wesley, 2015.

Additional constraints

- Breaking symmetry
  - E.g.
    - Only non-trivial operands
    - Use all steps
    - …
- DAG Topology
  - E.g.
    - Fences
    - Partial DAGs
    - …



Fences



Partial DAGs

Haaswijk W, Soeken M, Mishchenko A, et al. *SAT-based exact synthesis: Encodings, topology families, and parallelism*[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2019, 39(4): 871-884.

寧波大學
NINGBO UNIVERSITY

Technology Mapping

ASIC technology mapping

NINGBO UNIVERSITY

- The recursive tree-covering algorithm can be used to implement ASIC mapping in a simple and near-optimal manner.
- Step:
  - Mapped netlist and the logic gate of the technology library
    - Describing a netlist using only NAND2 and NOT logic gates.
    - The logic gates in the technology library are the same as above, and their corresponding costs are calculated.
  - Tree transformation
    - Splitting nodes with multiple outputs into trees.
  - Matching minimum implementation cost
    - Recursively searching the library for matching logic gates to implement at minimum cost.

ASIC technology mapping

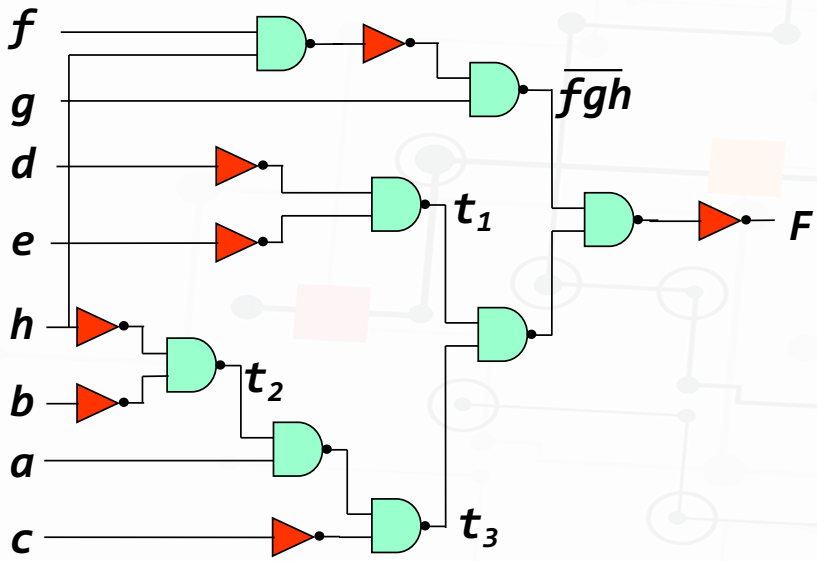
NINGBO UNIVERSITY

Simple gate mapping

- Applying De Morgan's law to a Boolean function to convert it into a representation using only NAND2 and NOT logic gates, for example:

```
t1 = d + e;
t2 = b + h;
t3 = at2 + c;
t4 = t1t3 + fgh;
F = t4';
```

$$t_1 = d + e = \overline{\bar{d} \cdot \bar{e}} = NAND(\bar{d},\ \bar{e})$$
$$t_2 = b + h = \overline{\bar{b} \cdot \bar{h}} = NAND(\bar{b},\ \bar{h})$$
$$t_3 = at_2 + c = \overline{\overline{at_2} \cdot \bar{c}} = NAND(NAND(a,\ t_2),\ \bar{c})$$
$$t_4 = t_1 t_3 + fgh = \overline{\overline{t_1 t_3} \cdot \overline{fgh}} = NAND(\overline{t_1 t_3},\ \overline{fgh})$$
$$fgh = \overline{\overline{fh} \cdot g} = \overline{\overline{fh} \cdot g} = NAND(\overline{NAND(f,\ h)},\ g)$$
$$F = \overline{t_4}$$

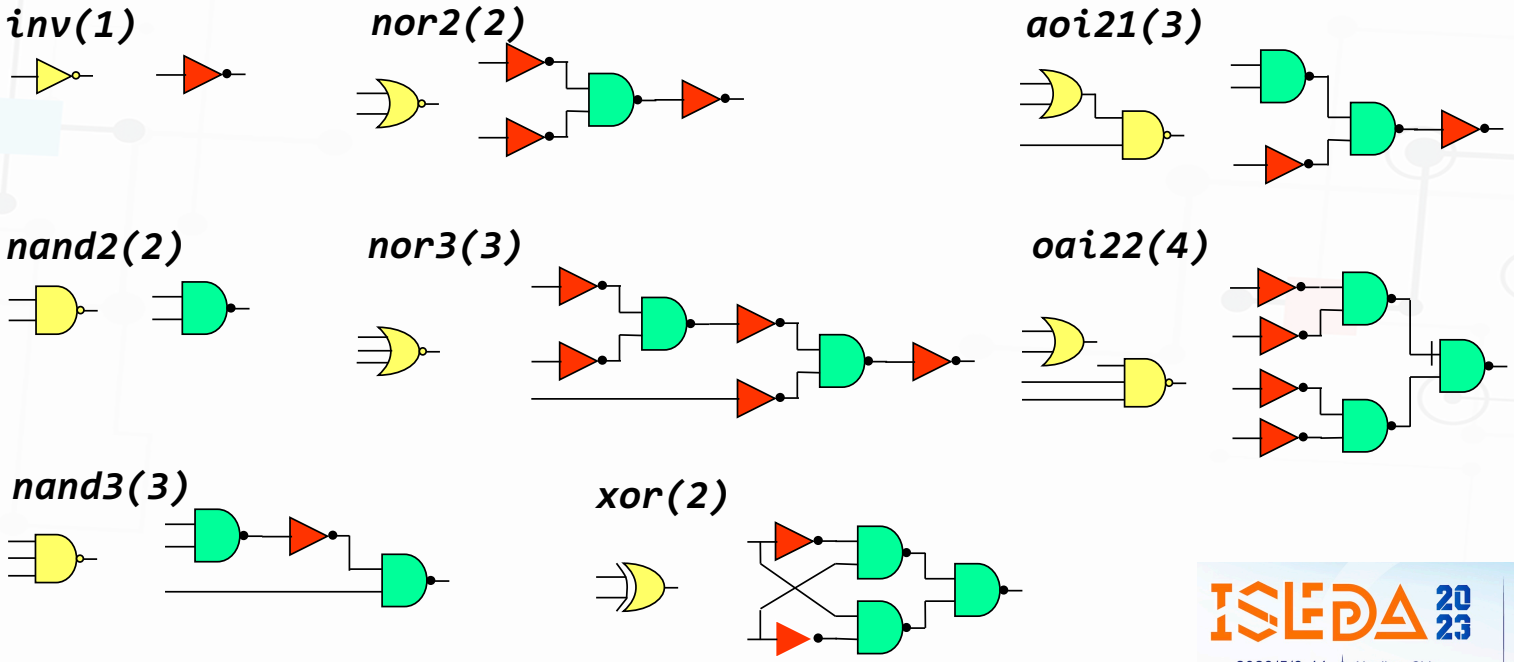


ASIC technology mapping

NINGBO UNIVERSITY

Simple gate mapping

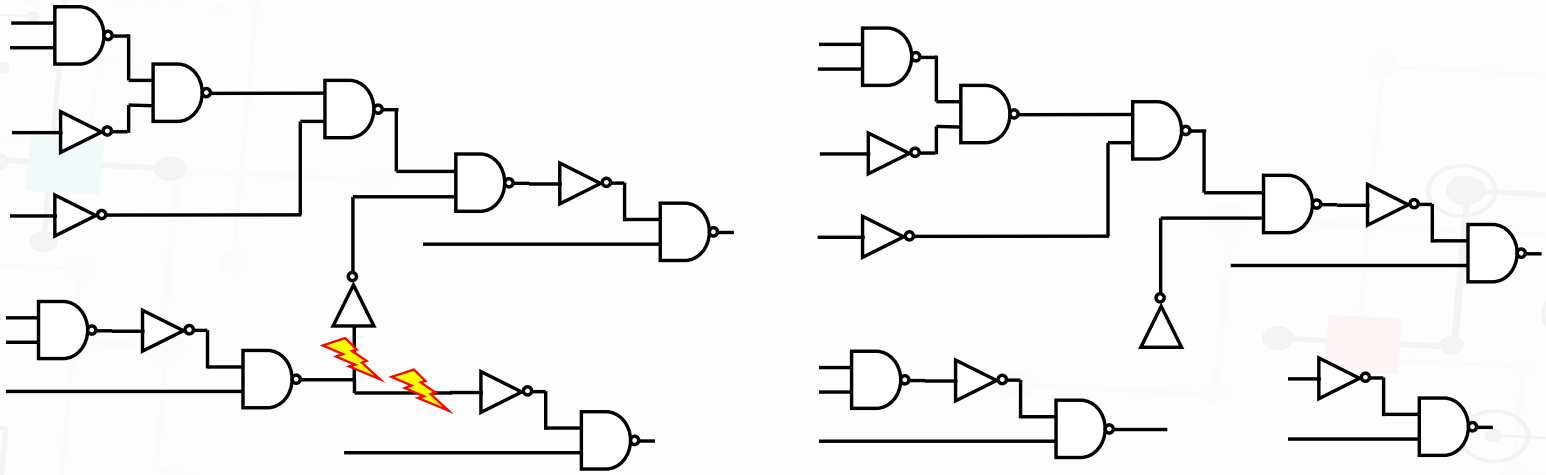
- For the logic gates in a library of technology, they can be represented using both NAND2 and NOT logic gates.



ASIC technology mapping

NINGBO UNIVERSITY

- Tree transformation
  - To apply tree covering algorithm, we must first have a tree!



Obtain three trees from left to right.

ASIC technology mapping

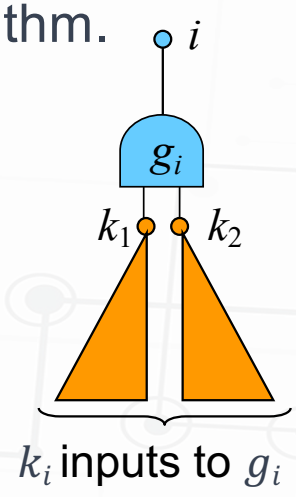
NINGBO UNIVERSITY

Minimum Spanning Tree Cover

- Implementing minimum vertex cover using a recursive algorithm.
  - Start with the output of the graph.
  - For each node, find all the ways it can be matched.
  - Node i cost function is as follows:

$$\text{cost}(i) = \min_k \left\{ \text{cost}(g_i) + \sum_k \text{cost}(k_i) \right\}$$

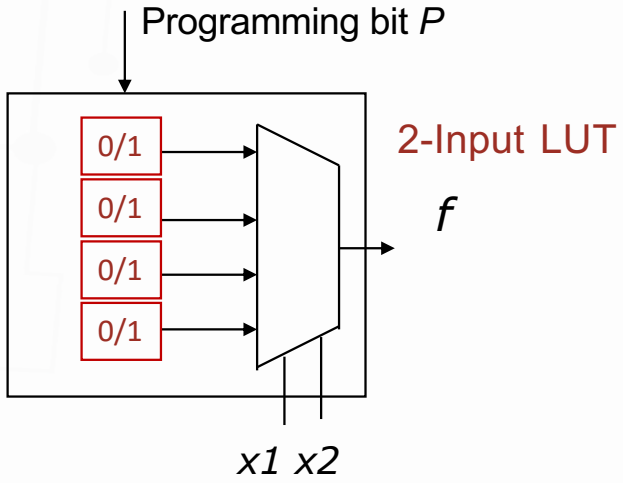
  - g_i represents a logical gate, and k_i represents the input to g_i.



FPGA technology mapping

NINGBO UNIVERSITY

- How is it different from ASIC (standard cells)
  - Structural in nature, simpler
  - Any function with k inputs can be mapped into a k-LUT
  - Typically implemented by cut mapping
- FPGA architecture: k-LUT



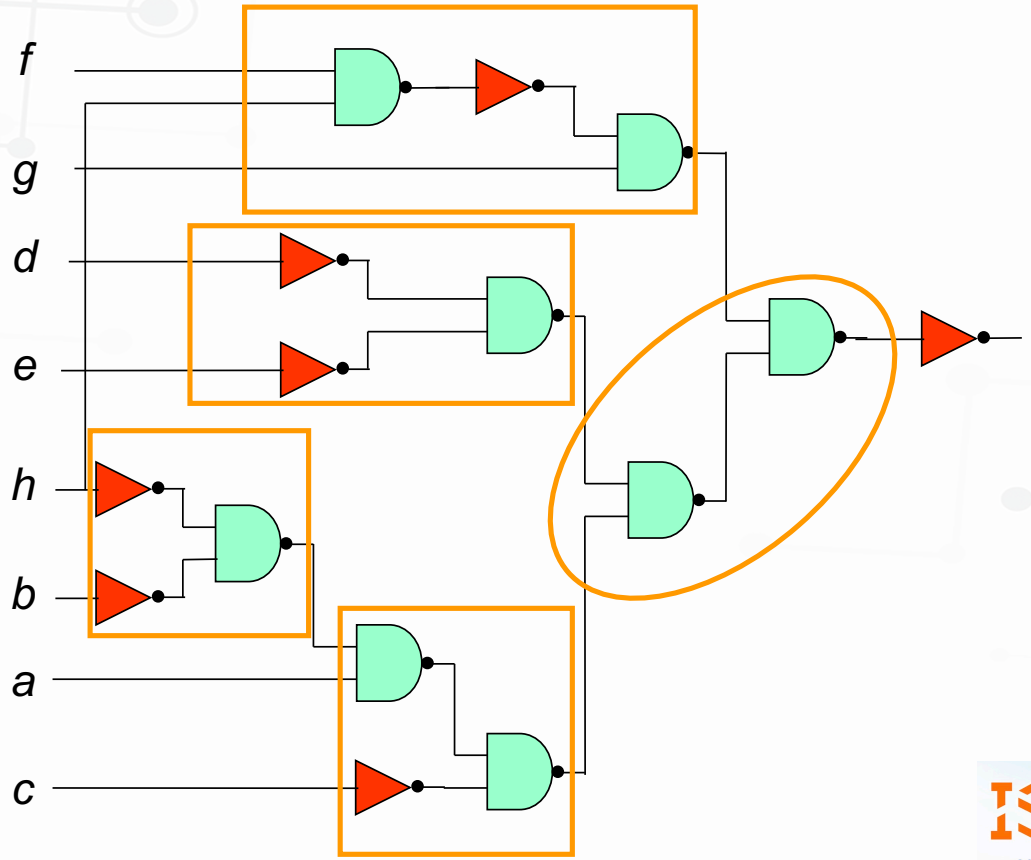
$$f = x_1'x_2' + x_1x_2$$

x1	x2	f
0	0	1
0	1	0
1	0	0
1	1	1

FPGA technology mapping

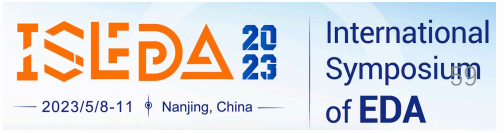
NINGBO UNIVERSITY

- A possible mapping onto 3-LUTs



宁波大学
NINGBO UNIVERSITY

Overview of Logic Synthesis Tools



Industry tools

- Synopsys Design Compiler (DC)
  - For ASIC，DC NXT，Fusion Compiler
  - For FPGA，Synplify
- Cadence Genus
  - For ASIC
- FPGA Vendor
  - Xilinx Vivado
  - Altera Quartus Prime
  - Pango Design Suite (PDS)
  - …

Academic tools

- ABC
  - https://github.com/berkeley-abc/abc
  - Alan Mishchenko，UC Berkeley
  - Predecessor：SIS、MIS、VIS
- Yosys
  - https://github.com/YosysHQ/yosys
  - Claire Xenia Wolf，use ABC for logic optimization
- EPFL open-source logic synthesis library
  - https://github.com/lsils/lstools-showcase
- ALSO
  - https://gitee.com/zfchu/also
- Espresso…

宁波大学
NINGBO UNIVERSITY

Demo

Q & A

chuzhufei@nbu.edu.cn

https://zfchu.github.io/