

And Invert Graph (AIG)

Virendra Singh

Professor

Computer Architecture and Dependable Systems Lab

Department of Electrical Engineering

Indian Institute of Technology Bombay

<http://www.ee.iitb.ac.in/~viren/>

E-mail: viren@ee.iitb.ac.in

EE-224: Digital Systems



Lecture 13-B: 20 September 2020 **CADSL**

Logic Representation

- Canonical Forms in common usage:
 - Truth Table ✓
 - Sum of Minterms (SOM) ✓
 - Product of Maxterms (POM) ✓
 - Binary Decision Diagram (BDD) ✓
 - Reed Muller Representation → Testable circuit
→ Quantum circuits
- Non canonical Representation
 - Sum of Product ✓
 - Product of Sum ✓
 - And Invert Graph - Graphical - Scalable



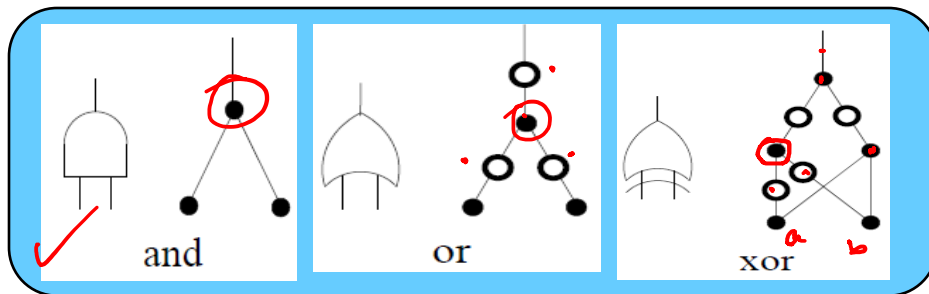
And-Invert Graph (AIG)

- AIG is a Boolean network with two types of nodes:
 - two-input ANDs, nodes
 - Inverters (NOT)
- Any Boolean function can be expressed using AIGs
 - For many practical functions AIGs are smaller than BDDs
 - Efficient graph representation (structural)
 - Very good correlation with design size
- AIGs are not canonical
 - For one function, there may be many structurally-different AIGs
 - Functional reduction and structural hashing can make them “canonical enough”



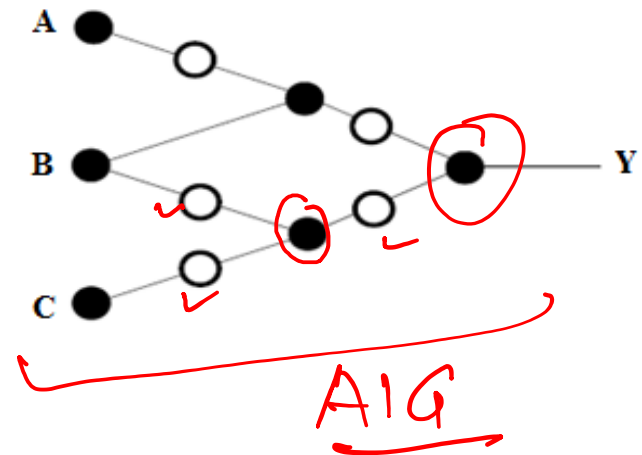
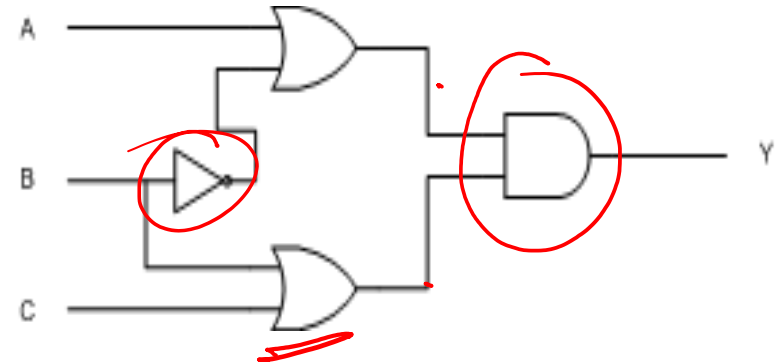
How to Create AIG ?

- AIGs are constructed from the Boolean network
- Constructed from the netlist available from technology independent logic synthesis



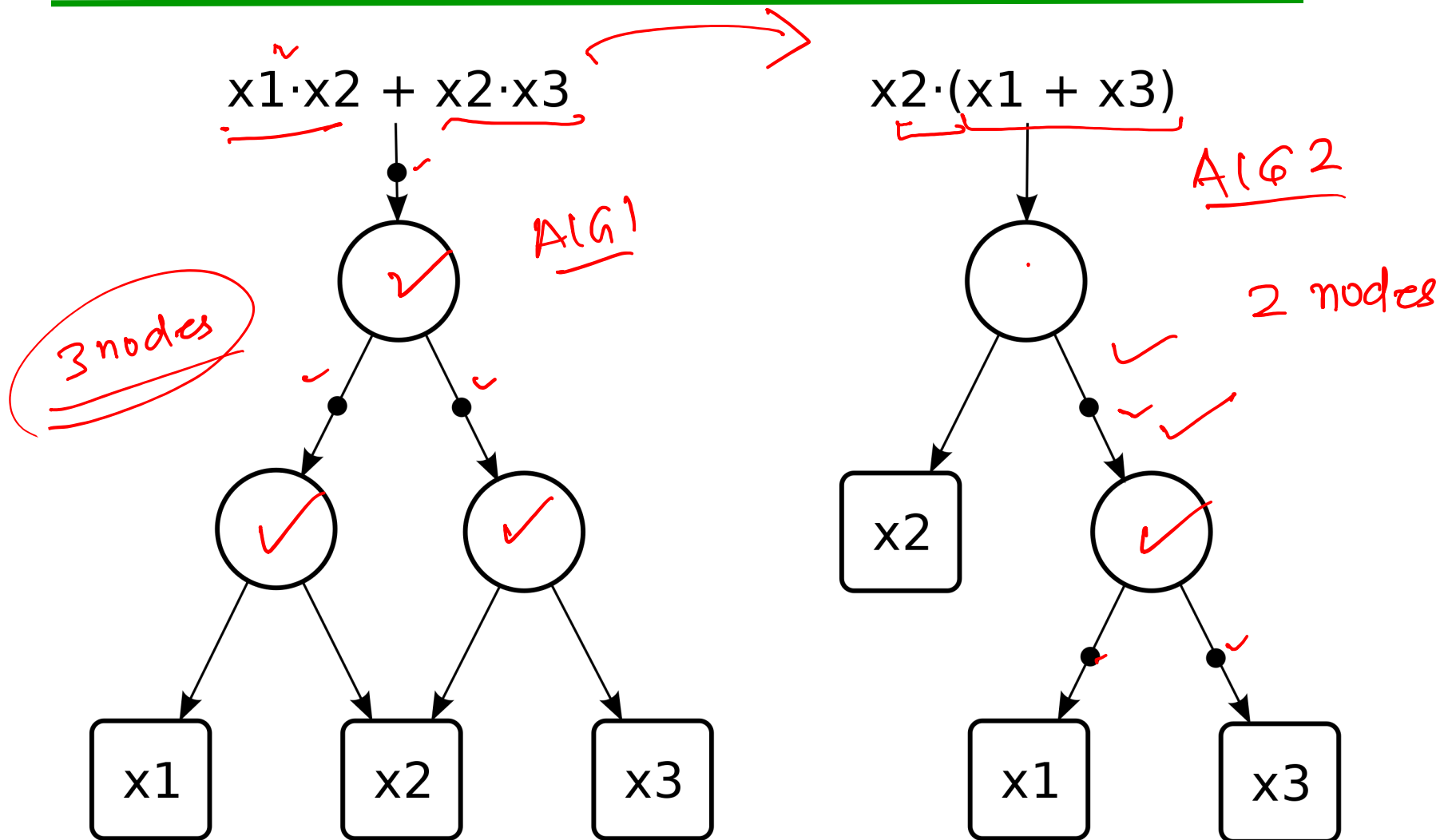
$$a + b = \overline{\bar{a} \cdot \bar{b}}$$

$$\overline{\bar{a} \cdot \bar{b}} + a \cdot b$$



AIG

AIG Non-canonicity ✓



AIG Non-canonicity

- AIGs are **not** canonical

– same function represented by two functionally equivalent AIGs with different structures

– BDDs – canonical for same variable ordering

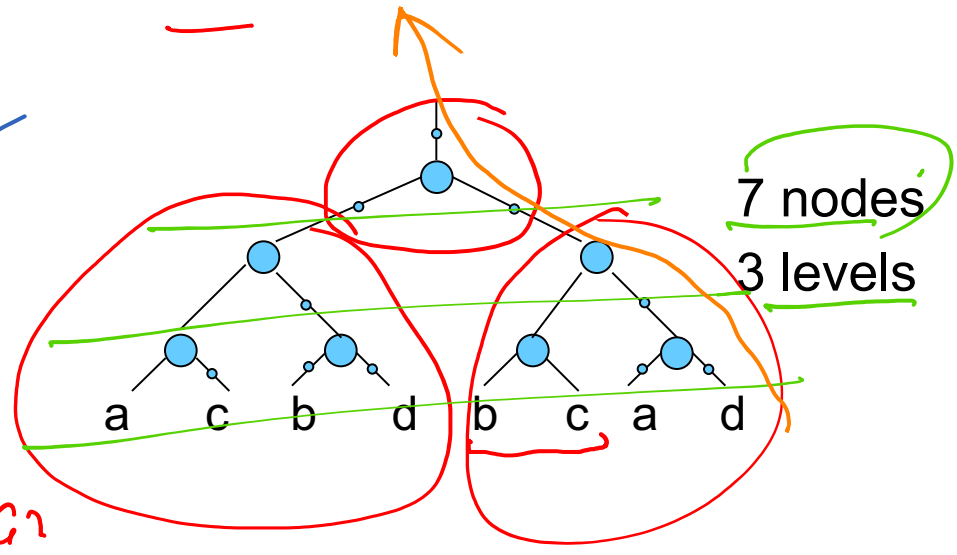
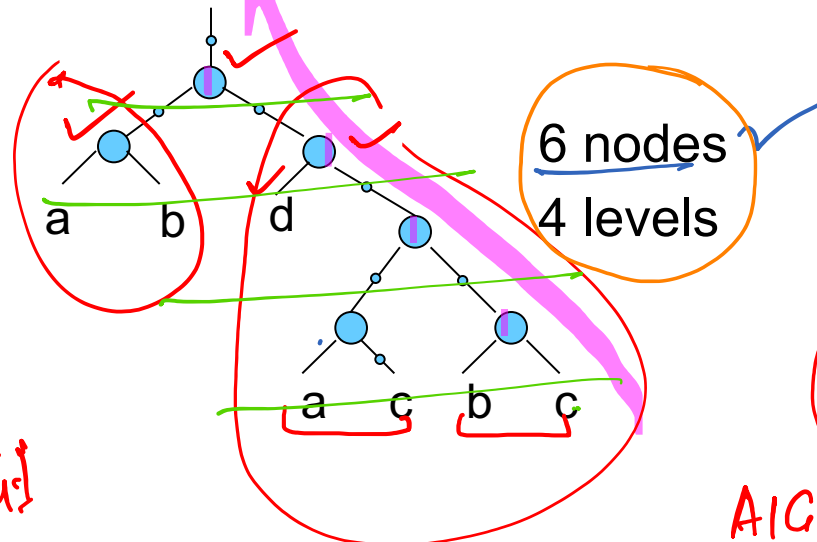
$$\underline{a\bar{c}b} + a\bar{c}d + \underline{bcd} + bcd$$

$$ab(\bar{c}+c) + d \cdot (a\bar{c} + bc)$$

$$\underline{ab + d \cdot (a\bar{c} + bc)}$$

$$\underline{a\bar{c} \cdot (b+d)} + \underline{b \cdot c (a+d)}$$

$$\underline{a \cdot b + d \cdot (a\bar{c} + bc)}$$



Basic Logic Operations

- Converting logic function into ALG graph

– Inversion

$$\neg a$$

– Conjunction

$$a \wedge b \quad (ab)$$

– Disjunction

$$a \vee b \quad (a+b) \quad \overline{a \cdot b}$$

– Implication

$$a \Rightarrow b$$

$$\begin{aligned} & \neg a \\ & a \wedge b \\ & \neg(\neg a \wedge \neg b) \\ & \neg(a \wedge \neg b) \quad \checkmark \end{aligned}$$

– Equivalence

$$a \Leftrightarrow b$$

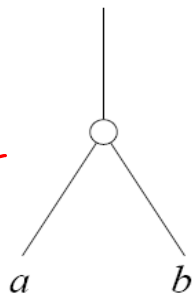
$$\begin{aligned} & \neg(a \wedge \neg b) \wedge \neg(\neg a \wedge b) \quad \checkmark \\ & \neg(\neg(a \wedge \neg b) \wedge \neg(\neg a \wedge b)) \end{aligned}$$

– $a \text{ XOR } b$

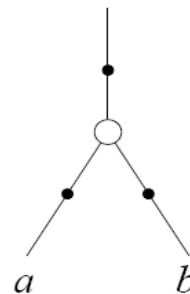
$$\begin{aligned} & a \rightarrow b \\ & b \rightarrow a \end{aligned}$$

$$\frac{a\bar{b} + \bar{a}b}{(\bar{a}\bar{b})(\bar{a}b)}$$

\rightarrow



$$a \wedge b$$



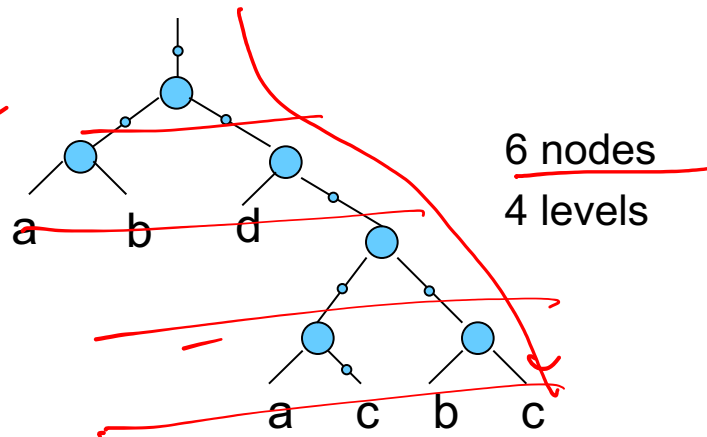
$$a \vee b$$



AIG Attributes

- AIG size *Area cost*
 - Measured by number of AND nodes
- AIG depth
 - Number of logic levels = number of AND-gates on longest path from a primary input to a primary output
 - The inverters are ignored when counting nodes and logic levels

Delay Performance

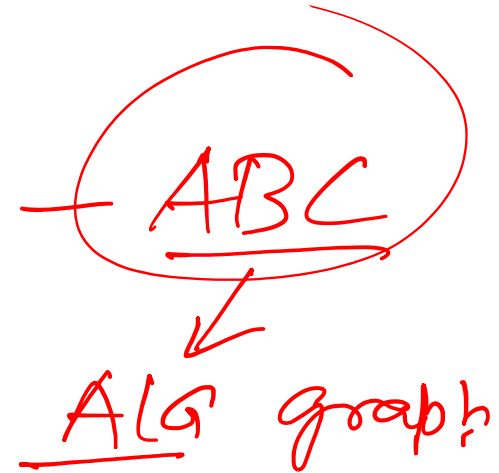


And Invert Graph (AIG)

Compact
→ Scalable ✓

, Alan Mischenko

Synthesis



Thank You

