

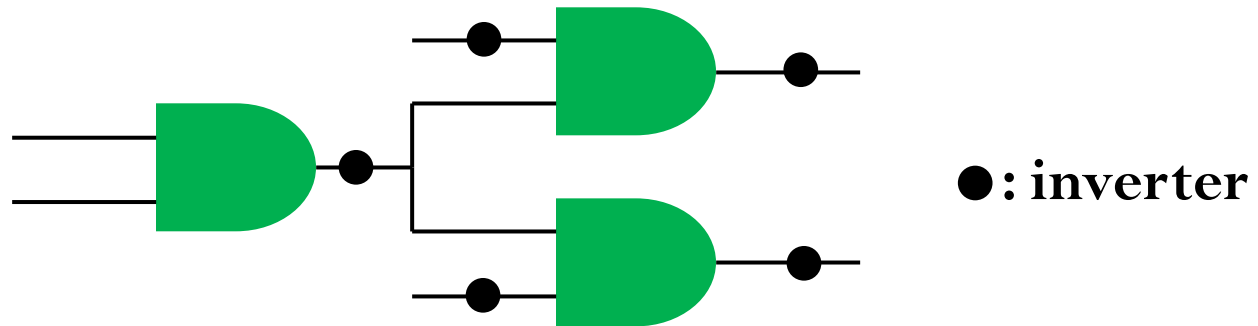
ECE6703J

Computer-Aided Design of Integrated Circuits

Multi-Level Logic Synthesis:
Modern AND-Inverter Graph (AIG)-based Method

AND-Inverter Graphs (AIGs)

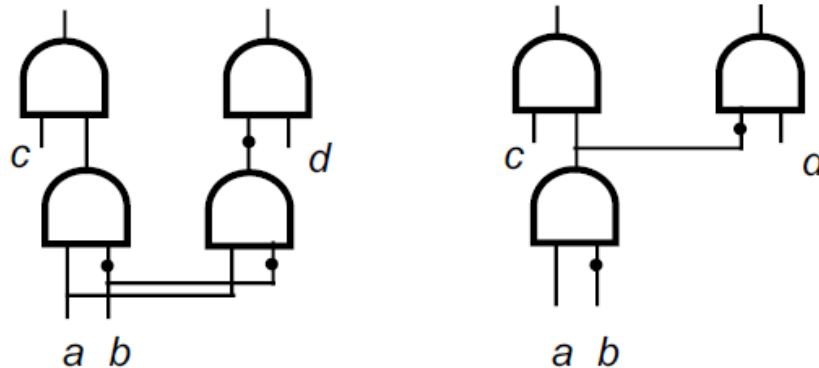
- Any Boolean function can be represented as an AIG!



- But AIG is not **canonical**
 - For the same function, many different AIG representations.
- For area reduction, minimize the **number of AND nodes**.
- Tool: ABC: A System for Sequential Synthesis and Verification
 - <http://www.eecs.berkeley.edu/~alanmi/abc>

AIG Structural Hashing (Strashing)

- When building AIGs, always add AND node
 - When an AIG is constructed without strashing, AND gates are added one at a time without checking whether AND gate with the same fanins already exists
- One-level strashing
 - When adding a new AND-node, check the hash table for a node with the same input pair (fanin)
 - If it exists, return it; otherwise, create a new node

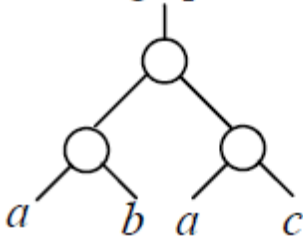
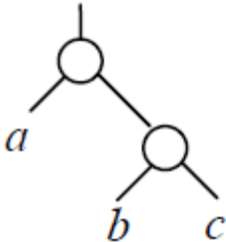
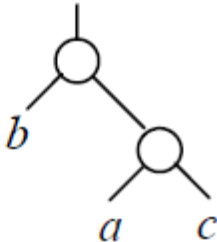


AIG Optimization Operations

- Rewriting ✓
 - A. Mishchenko, S. Chatterjee, and R. Brayton, “DAG-aware AIG rewriting: A fresh look at combinational logic synthesis”, DAC’06.
- Resubstitution ✓
 - A. Mishchenko, R. Brayton, J.-H. R. Jiang, and S. Jang, “Scalable don't care based logic optimization and resynthesis”, FPGA’09.
- Balancing
 - J. Cortadella, “Timing-driven logic bi-decomposition”, IEEE TCAD, vol. 22(6), June 2003, pp. 675-685.
- Refactoring
 - R. Brayton and C. McMullen, “The decomposition and factorization of Boolean expressions,” ISCAS’82.

Rewriting: Phase 1

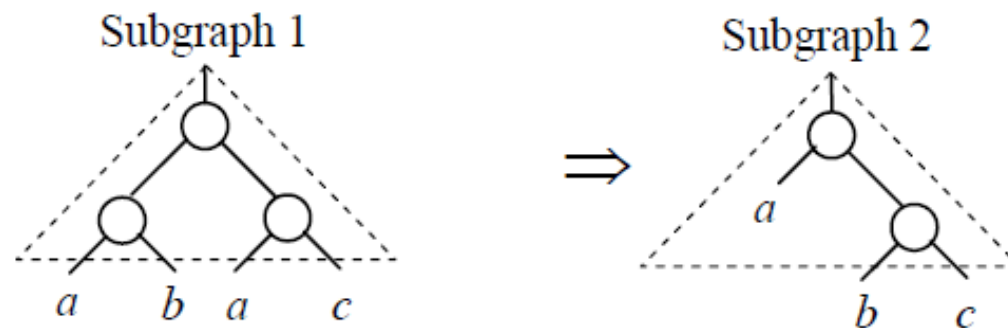
- All two-level AIG subgraphs are **pre-computed** and stored in a **hash table** by their functionality.
- The table contains all non-redundant AIG implementations of logic functions with four variables or less

Hash table	Function	All non-redundant AIG implementations		
		Subgraph 1	Subgraph 2	Subgraph 3
	abc			
	$a(b + c)$...		
		

Rewriting: Phase 2

- For each node in topological order, find its two-level AIG subgraph and compute its Boolean function.
- Use the function to access the hash table to find equivalent subgraphs.
- Try each subgraph, while **taking into account logic sharing** between the new subgraph nodes and the existing nodes.
- Choose the subgraph with largest save in # of nodes.

Example 1

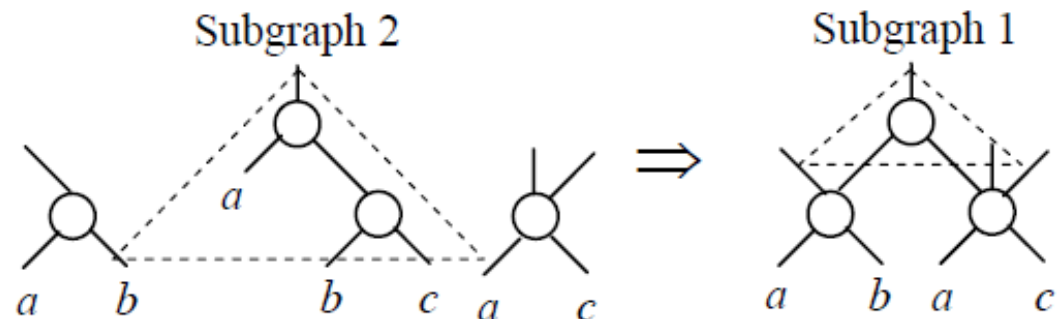


Rewriting: Phase 2

- For each node in topological order, find its two-level AIG subgraph and compute its Boolean function.
- Use the function to access the hash table to find equivalent subgraphs.
- Try each subgraph, while **taking into account logic sharing** between the new subgraph nodes and the existing nodes.
- Choose the subgraph with largest save in # of nodes.

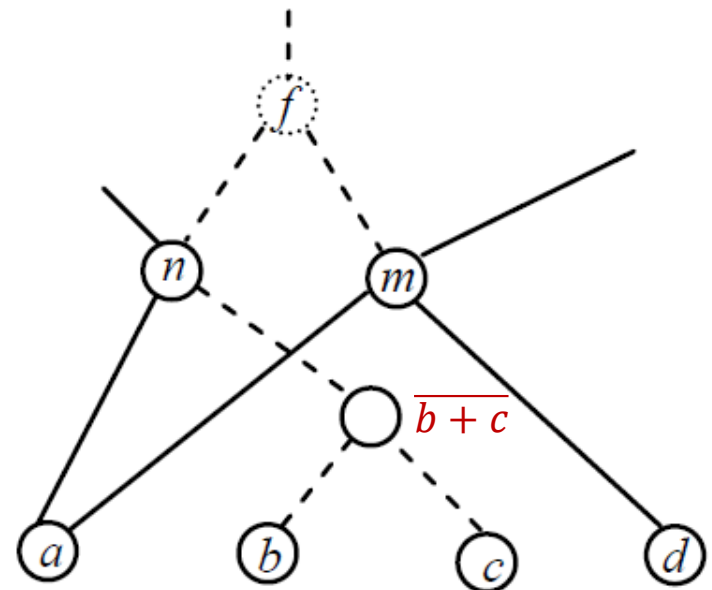
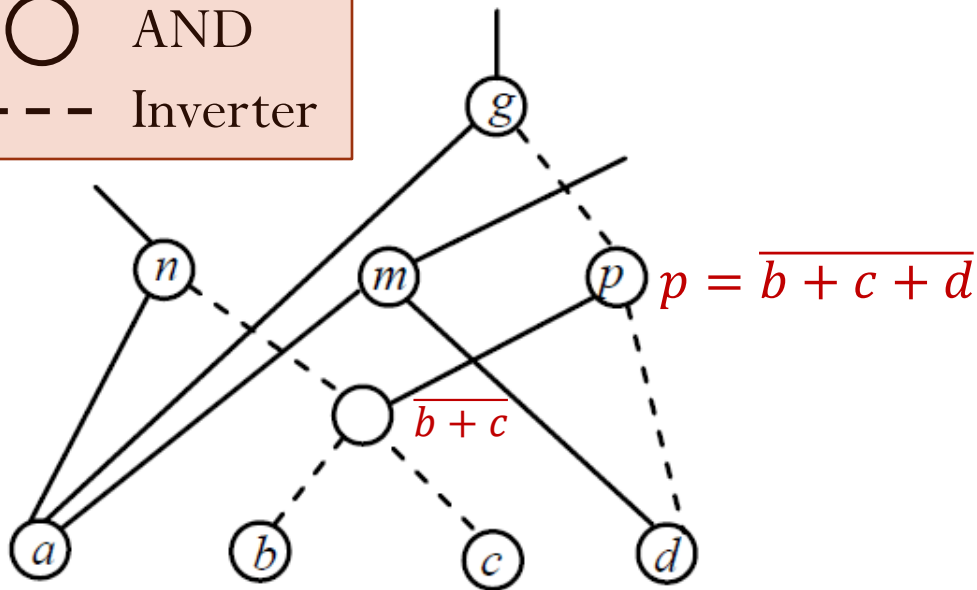
Example 2

Taking into account
logic sharing



Resubstitution

- Express the function of a node using other nodes (divisors).
- Example:
 - Replace function $g = a(b + c + d)$ as $f = n + m$, where $n = a(b + c)$ and $m = ad$



More about AIG Optimization in ABC

- Some other operations, e.g., balancing and refactoring
- Write **scripts of basic operations**.
 - Do **several passes** of different optimizations over AIG.
 - E.g., balancing -> rewriting -> refactoring -> resubstitution
 - Lots of “art” in the engineering of these scripts.
- To explore more:
 - Visit ABC webpage:
<http://www.eecs.berkeley.edu/~alanmi/abc>