Implementation Verification: Equivalence Checking

Profs. Kurt Keutzer & Sanjit Seshia

EECS

UC Berkeley

With thanks to Srinivas Devadas, MIT

Design Process

Design : specify and enter the design intent

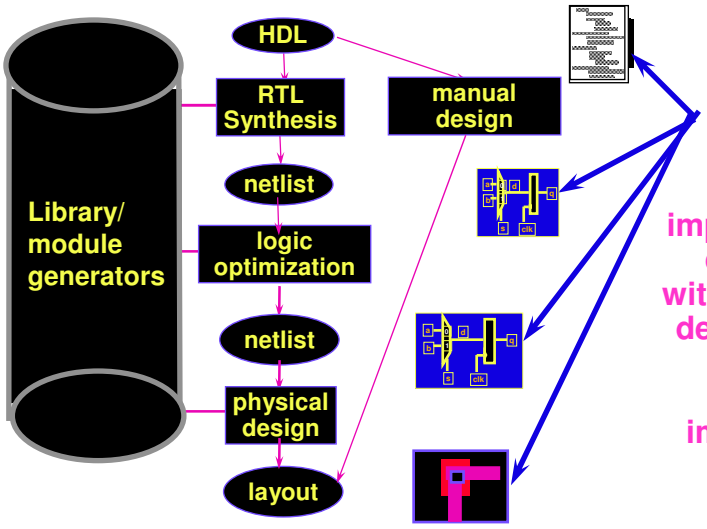
Verify:

verify the correctness of design and implementation

Implement:

refine the design through all phases

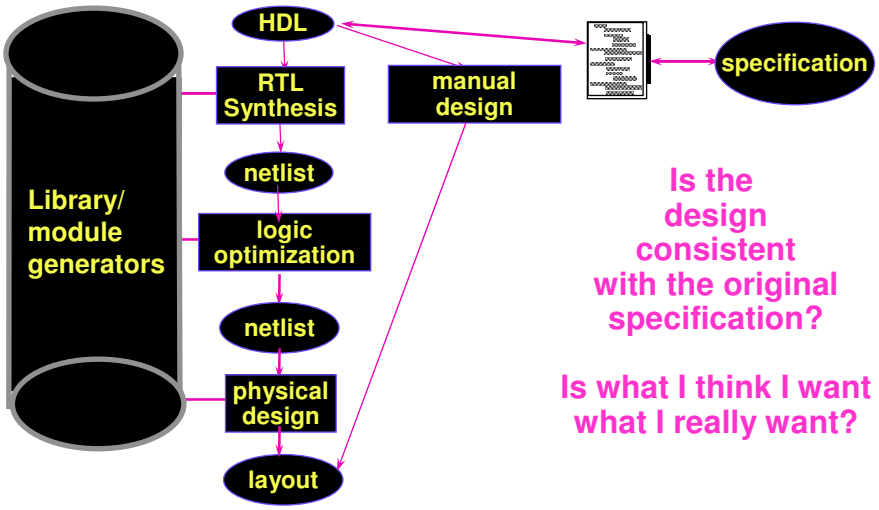
Implementation Verification



Is the implementation consistent with the original design intent?

Is what I implemented what I wanted?

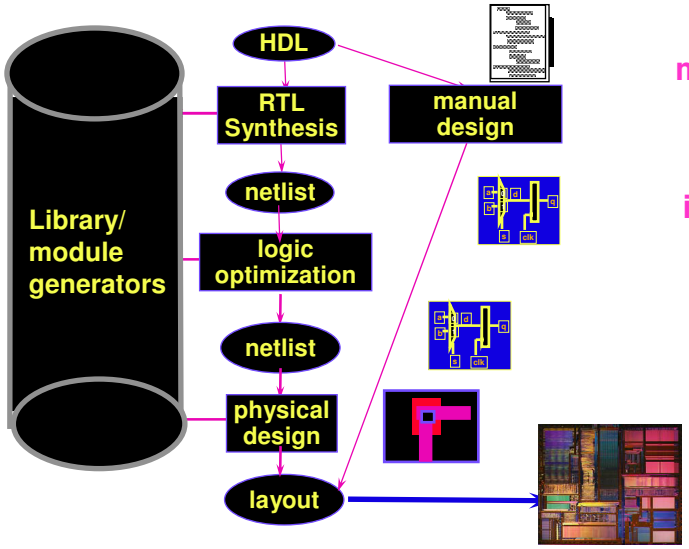
Design Verification



Is the design consistent with the original specification?

Is what I think I want what I really want?

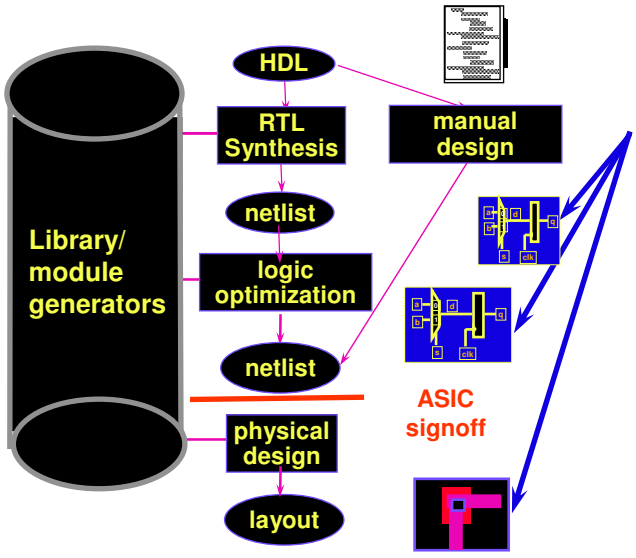
Manufacture Verification (Test)



Is the manufactured circuit consistent with the implemented design?

Did they build what I wanted?

Impl. Verification for ASIC's by Simulation

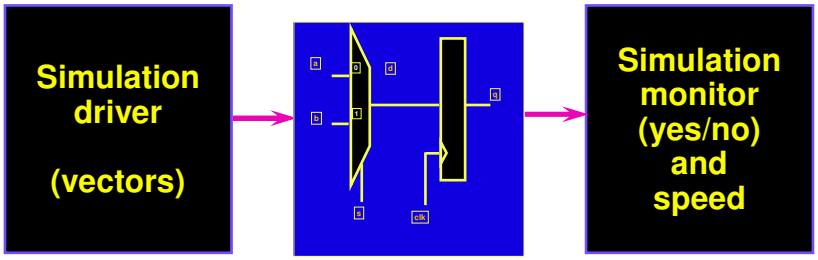


Apply gate-level simulation at each step to verify

functionality:
• 0-1 behavior on regression test set

and timing:
• maximum delay of circuit across critical paths

Software Simulation

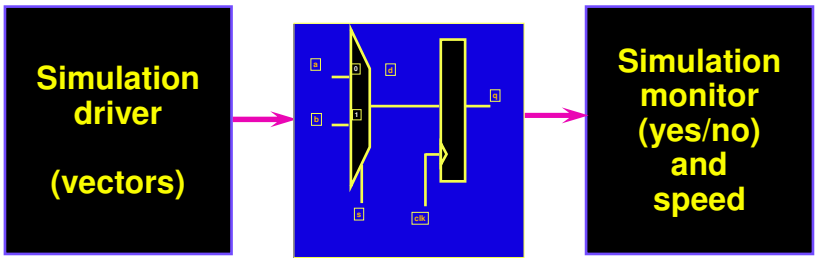


Advantages of gate-level simulation

- verifies timing and functionality simultaneously
- approach well understood by designers

Disadvantages of gate-level simulation?

Software Simulation



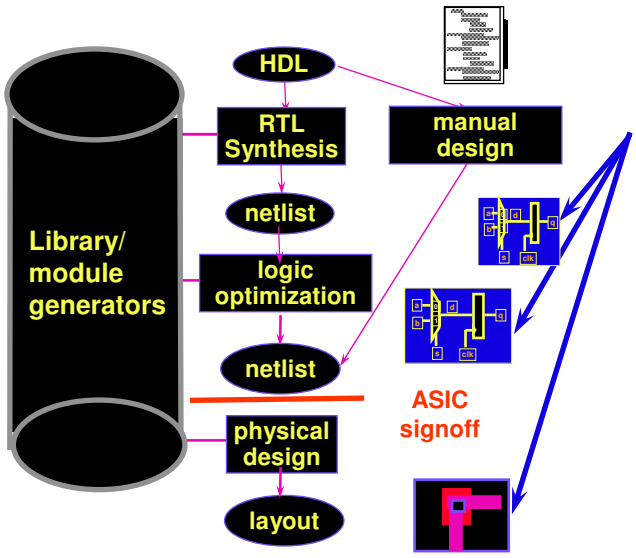
Advantages of gate-level simulation

- verifies timing and functionality simultaneously
- approach well understood by designers

Disadvantages of gate-level simulation?

- computationally intensive - only ~10 clock cycles of 100K gate design per 1 CPU second
- incomplete - results only as good as your vector set - easy to overlook incorrect timing/behavior

Alternative – "Static Sign-off"



Use static analysis techniques to verify:

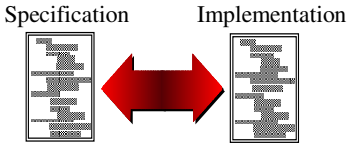
functionality:
- *formal equivalence-checking techniques*

and timing:
- use static timing analysis – discussed earlier in the semester, but don't forget about false paths!

Problem: RTL to RTL Verification

After verification RTL may still be modified

- RTL level improvements for :
  - performance
  - power
  - area
  - testability



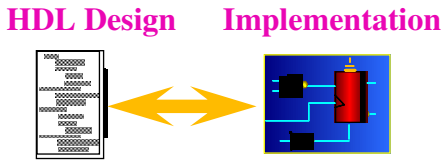
Need to verify that new RTL is correct

Problem: RTL to Gates Verification

Verify the gate level implementation is consistent with the RTL level design

Errors may have occurred due to

- synthesis (heaven forbid!!)
- manual intervention

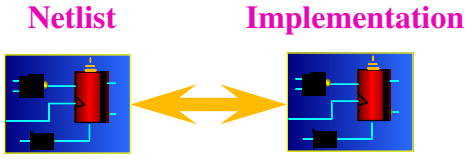


Problem: Gates to Gates Verification

Verify the modified gate level implementation is consistent with the RTL level design

Errors may have occurred due to

- Incorrect synthesis or module generation (heaven forbid!!)
- Test insertion
- Scan chain reordering
- Clock tree synthesis
- Post layout "tweaks"



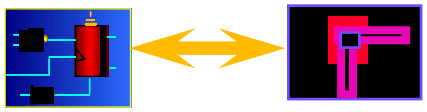
Problem: Layout to Gates Verification (LVS)

Verify the modified gate level implementation is consistent with the RTL level design

Errors may have occurred due to

- Errors in physical design tools
- Manual changes in layout

Verification is primarily graphical or ``topological''

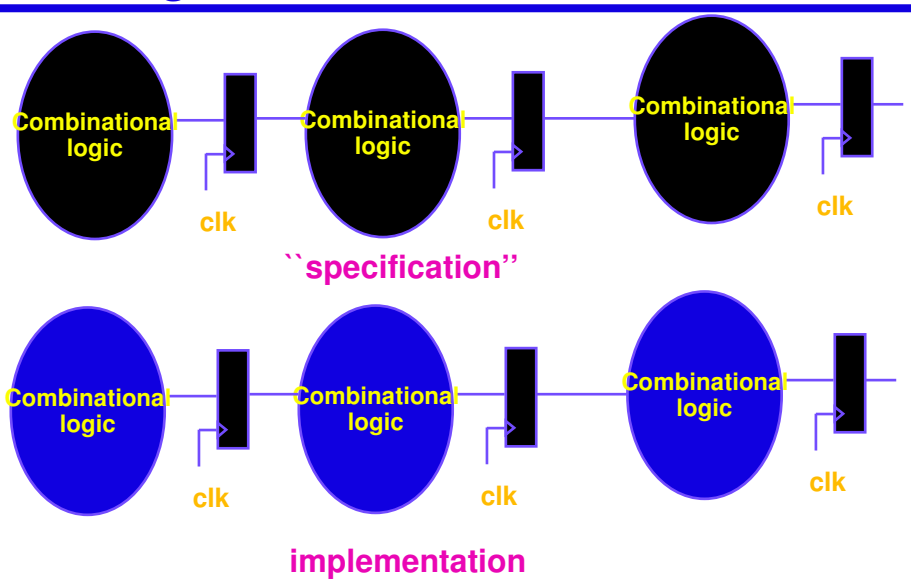


netlist physical layout

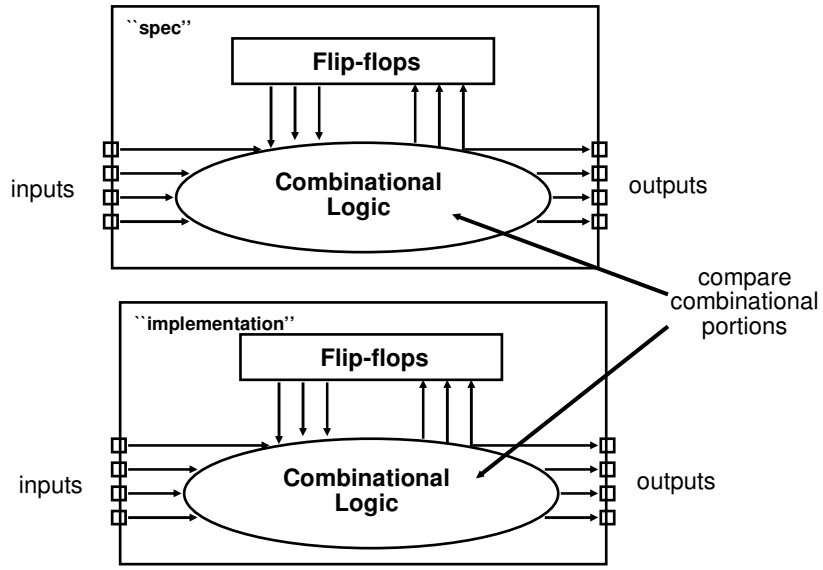
This Lecture

- Solving the "Gates-to-Gates" verification problem
  - Special case: when only combinational parts differ
- Binary Decision Diagrams (BDDs)
  - A very, very useful data structure!
  - Many applications, in EDA and elsewhere

Solving Gates to Gates Verification



Extract combinational portions



Combinational Equivalence Checking

Given:

- Combinational circuits C1 and C2
  (Boolean functions B1 and B2)

How can we prove that C1 is/isn't equivalent to C2, in a reasonable amount of time?

Combinational Equivalence Checking

Presumes equivalence-relation given (or discovered) between sequential circuits

Approaches

- Boolean satisfiability (SAT)
- Set-theoretic approaches (used in 2-level examples)
- Symbolic simulation
- Structural techniques
  - graph isomorphism
- Canonical forms - BDD's and variants
- Test-oriented methods

These techniques form the foundation of modern equivalence checking/implementation verification

2-level circuits

$$(F \Leftrightarrow G) \Leftrightarrow (F \rightarrow G) \bullet (G \rightarrow F)$$

$$\Leftrightarrow (\overline{F} \vee G) \wedge (F \vee \overline{G})$$

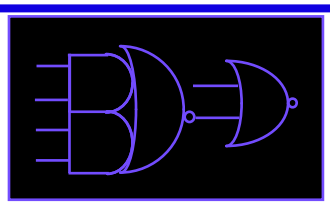
Now, treating F and G as sets of cubes we can check if

$$(\overline{F} \cup G) \cap (F \cup \overline{G}) \Leftrightarrow 1$$

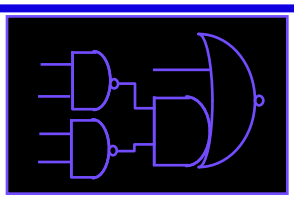
Which is feasible for most 2-level circuits/SOP expressions

Worked well in the Espresso era – doesn't generalize to multilevel

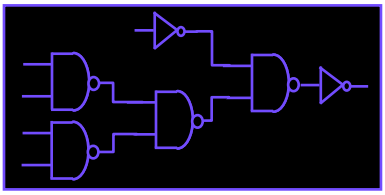
Multilevel: Structural Methods



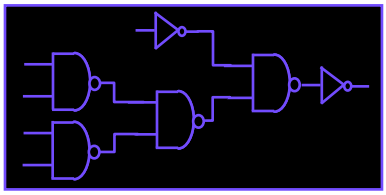
Combinational circuit 1



Combinational circuit 2



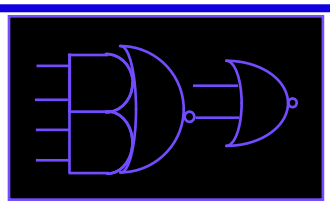
unmapped circuit 1



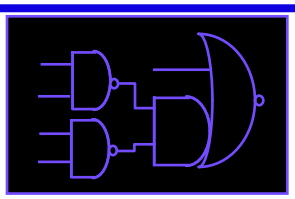
unmapped circuit 2

Compare them as graphs.
Is this tough or easy to solve algorithmically?

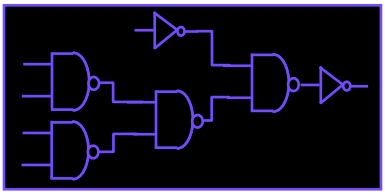
Structural Methods



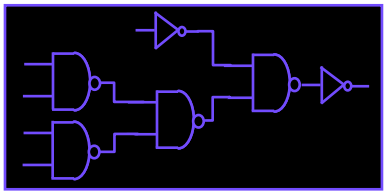
Combinational circuit 1



Combinational circuit 2



unmapped circuit 1



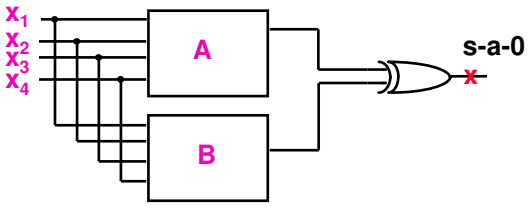
unmapped circuit 2

Looks tough – graph isomorphism
Turns out to be easy – DAGs
If this returns "Equivalent", are the ckts equivalent?
How about when it returns "Not Equivalent"?

More powerful: Testing

Given two single-output circuits A and B

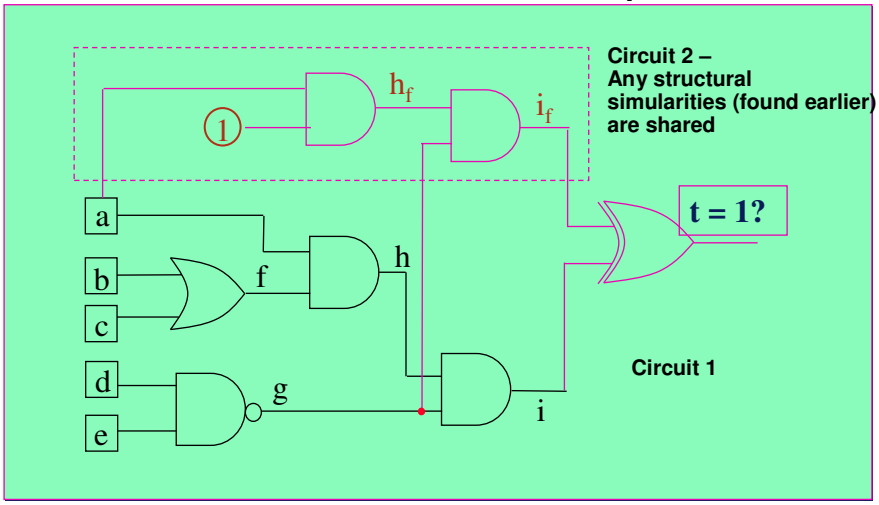
Are A and B equivalent can be posed as: Is there a test for F s-a-0?



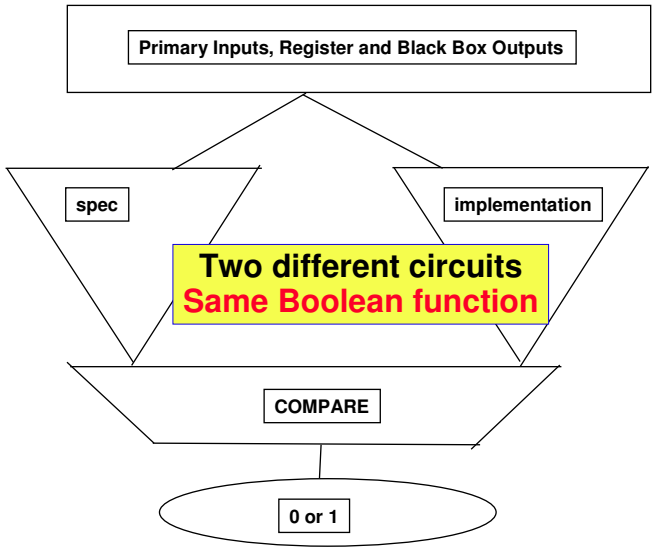
If F s-a-0 is redundant, A ≡ B else test vector produces different outputs for A and B.

Boolean Satisfiability (SAT) Again

This time ask whether there is an input on which Circuit 1 and Circuit 2 differ? This time we don't expect one!



Looking at the Problem Afresh



Strategy

- For each combinational circuit, compute the Boolean function that it represents
- Compare the two Boolean functions you get
  - Circuits are equivalent iff functions are identical

What is a Boolean function representation of a circuit?

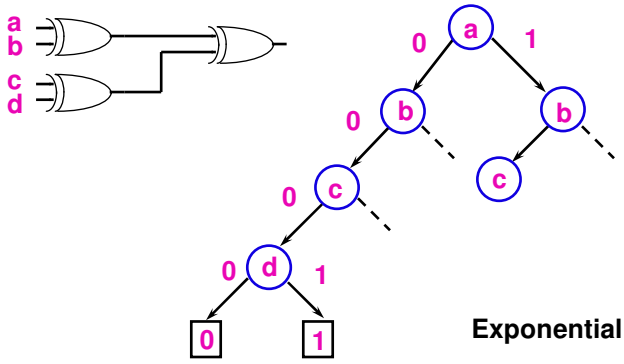
- Syntactic
- Semantic (Captures the meaning)

What we need is a canonical form (why?)

Boolean Function Representations

- Syntactic: e.g.: 2-level form (SOP)
- Semantic: e.g.: Truth table
  - Is a truth table good enough?

Canonical Form: Binary Decision Tree

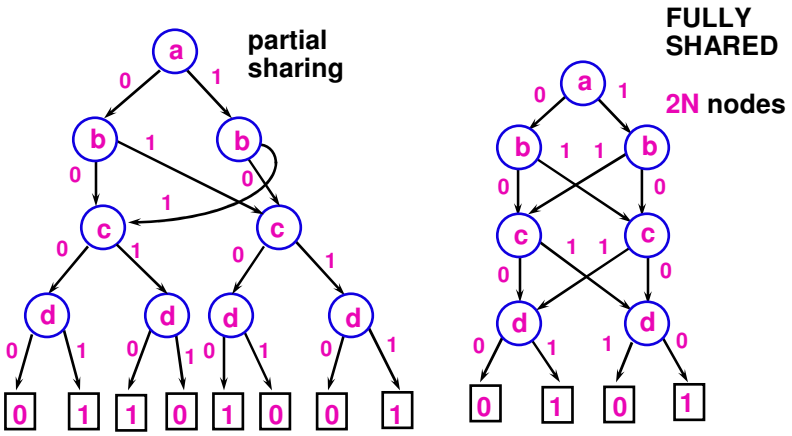


If you don't store entire set of nodes, you have to enumerate them

- **Doing what SAT does**

Decision Graph

Share nodes in tree ⇒ graph.



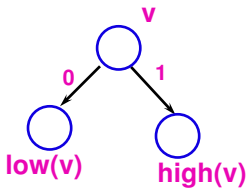
Definition of a Binary Decision Diagram

A Binary Decision Diagram having root vertex *v* denotes a Boolean function f_v

1. If *v* is a terminal vertex:

(a) if *value(v)* = 1, then $f_v = 1$

(b) if *value(v)* = 0, then $f_v = 0$



2. If *v* is a nonterminal vertex with *index(v) = n* then f_v is the function:

$$f_v(x_1, \ldots, x_n) = !x_n\, f_{low(v)}(x_1, \ldots, x_{n-1}) + x_n\, f_{high(v)}(x_1, \ldots, x_{n-1})$$

Called the "Shannon decomposition"

Definition of an Ordered BDD

A Binary Decision Diagram is *ordered* iff:

1. If *v* is a non-terminal vertex:

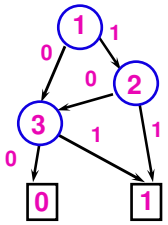
(a) if *low(v)* is a non-terminal then, *index(v) < index(low(v))* and

(b) if *high(v)* is a non-terminal then, *index(v) < index(high(v))* and

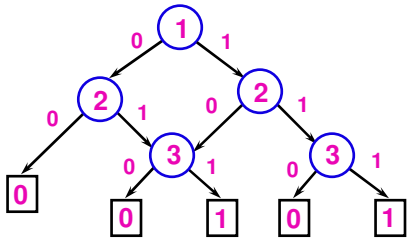
This property implies the property of *freedom* in BDDs: In traversing any path from a vertex in a OBDD to its root then we encounter each decision variable at most once.

Ordered Binary Decision Diagram

$f = x_1 x_2 + x_3$



$f = x_1\bar{x}_2x_3 + x_1x_2x_3 + \bar{x}_1x_2x_3$



Inputs satisfy ordering restriction. Each node/vertex v in the graph has index(v). Two children are low(v) and high(v). 0 and 1 are terminal vertices, others are non-terminal.

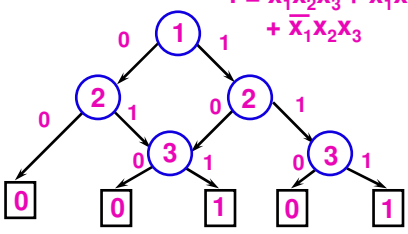
index(v) < index(low(v)) for all v

index(v) < index(high(v))

Ordered BDDs Enough?

Storage is always a problem for Ordered Binary Decision Diagram (OBDD) can we simplify them further?

$f = x_1x_2x_3 + x_1\bar{x}_2x_3 + \bar{x}_1x_2x_3$

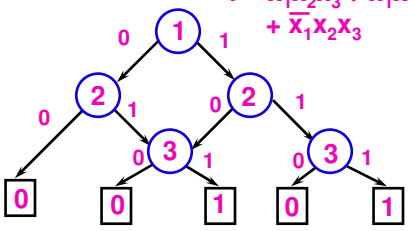


Reduced, Ordered BDDs

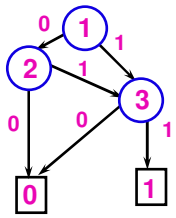
An Ordered Binary Decision Diagram (OBDD) may still have ``redundant'' vertices.

Definition: An OBDD is reduced, if it contains no vertex v with low(v) = high(v) , nor does it contain distinct vertices v and v' such that the subgraphs rooted by v and v' are isomorphic.

$f = x_1x_2x_3 + x_1\bar{x}_2x_3 + \bar{x}_1x_2x_3$



$f = x_1x_3 + \bar{x}_1x_2x_3$

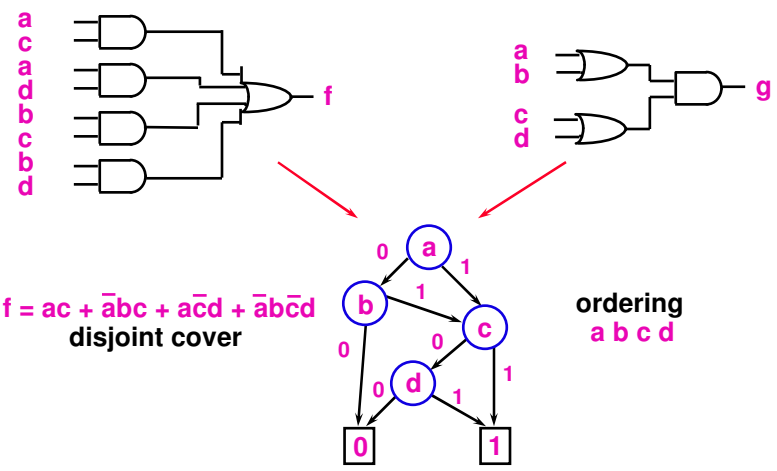


Can reduce an OBDD in O(|G| log |G|) time.

Reduced Ordered BDDs

- Invented by Randal E. Bryant in mid-80s
  - IEEE Transactions on Computers 1986 paper is one of the most highly cited papers in EECS
- Key data structure for many EDA problems including in synthesis & verification
- Commonly known simply as BDDs
- Many variants of BDDs have proved useful in other tasks
- Links to coding theory (trellises), etc.

ROBDDs are Canonical



Proof that ROBDDs are canonical - 1

Theorem (R. Bryant): If G, G' are ROBDD's of a Boolean function f with k inputs then G and G' are identical.

Exercise for next Wed. class (hint: use induction)