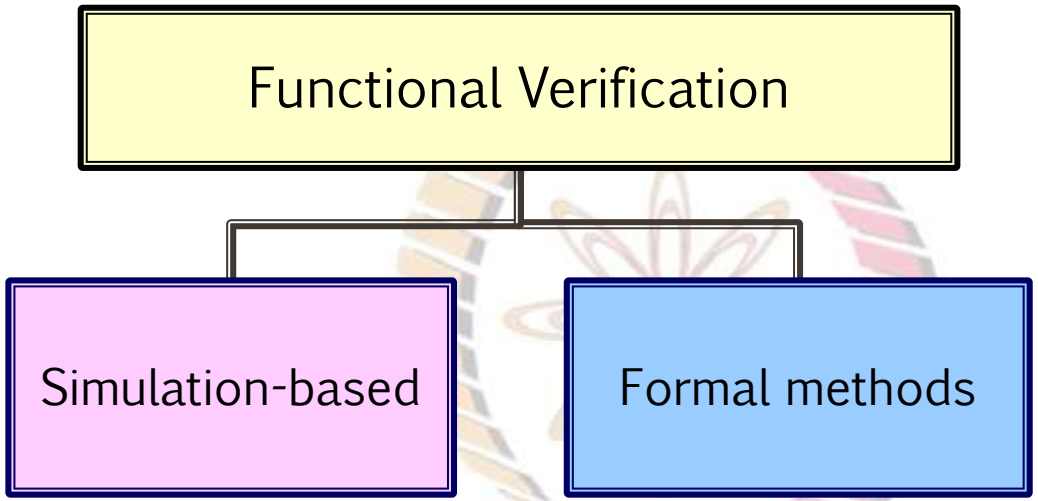
VLSI DESIGN FLOW: RTL TO GDS

Lecture 17
Formal Verification- I

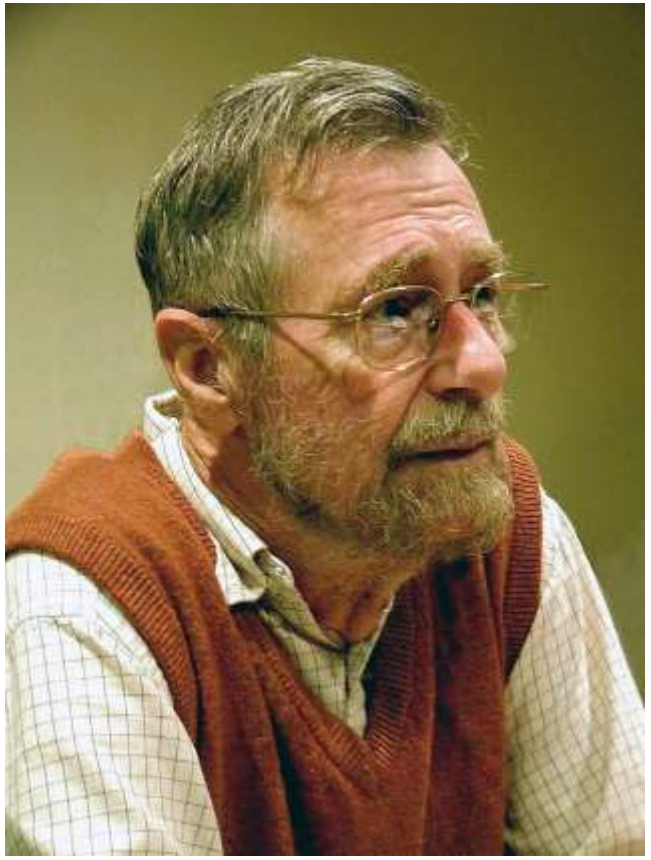
Sneh Saurabh
Electronics and Communications Engineering
IIIT Delhi

Lecture Plan



- Functional Verification using Formal Methods

Proof of correctness …



'…program testing can be a very effective way to show the presence of bugs, but is hopelessly inadequate for showing their absence. The only effective way to raise the confidence level of a program significantly is to give a convincing proof of its correctness…"

—Edsger W. Dijkstra, "The humble programmer," *ACM Turing Lecture*, 1972

Source: https://commons.wikimedia.org/wiki/File:Edsger_Wybe_Dijkstra.jpg Hamilton Richards, CC BY-SA 3.0 <http://creativecommons.org/licenses/by-sa/3.0/>, via Wikimedia Commons

Formal Verification

Limitations of Simulation-based Verification

Illustration:

- Consider multiplication of two 32-bit integers
- There are $2^{32} \times 2^{32} = 2^{64}$ combinations.
- If simulation of one test vector take $1\mu s$ to simulate,
  - Will take $2^{64} \times 1 \times 10^{-6}$ seconds (≈ 0.5 million years approximately!)
- Simulation-based exhaustive verification is not feasible for real-world designs

Problems of Simulation-based Verification:

- Input patterns biased towards anticipated sources of errors
  - Errors often occur where not anticipated
- Can never prove the correctness of a given design.
- Formal verification methodologies provide a feasible alternative.

Formal Verification as an Alternative

Formal Verification as an alternative

- Applies mathematical reasoning to establish proof of correctness
- Correctness ⇒ the system behaves correctly irrespective of input vectors
  - ➢ All cases implicitly covered in Formal Verification

Specification: $y = (x-4)^2$

Design: $y = x^2 - 8x + 16$

x	Specification: $y = (x-4)^2$	Design: $y = x^2 - 8x + 16$	Pass/Fail?
0	16	16	Pass
1	9	9	Pass
2	4	4	Pass
-1	25	25	Pass
104	10000	10000	Pass

Formal verification

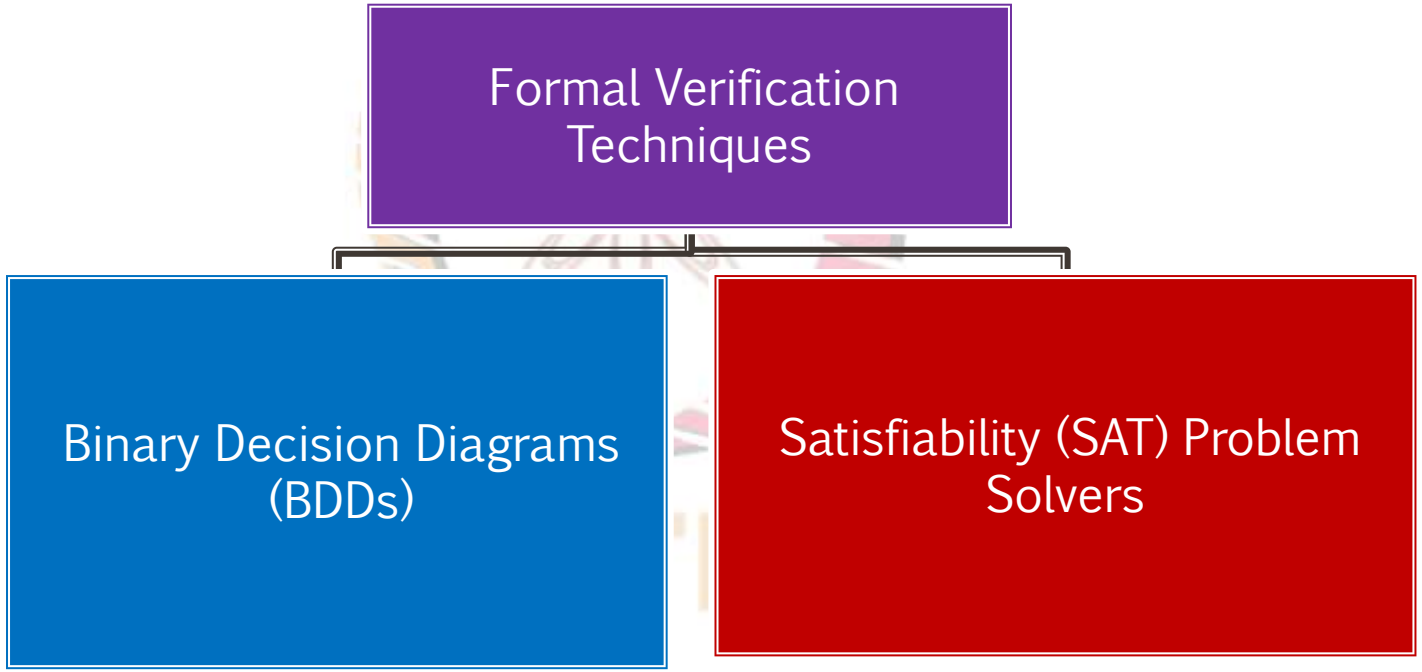
$$
\begin{aligned}
&(x-4)^2 \\
&= (x-4).(x-4) \\
&= x.(x-4) - 4.(x-4) \\
&= x.x - x.4 - 4.x + 16 \\
&= x^2 - 8x + 16
\end{aligned}
$$

Formal Verification : Differences with Simulation

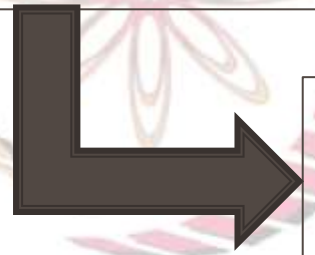
	Simulation-based verification	Formal verification
Test Vectors	Required	Not Required
Completeness	No	Yes
Mechanism	Test Vectors simulate the design and output response compared with the expected response	A mathematical proof of correctness is established or a counter-example is produced
Memory Requirement	Comparatively Low	Comparatively High

Formal Verification: Techniques

- Formal verification is computationally challenging
- Early 1990s, efficient formal verification techniques for VLSI applications developed



Formal Verification



Binary Decision Diagram

Boolean Function Representations: Compactness

- Boolean functions can be represented in many ways

Compactness of Representation:

- Quantifies the growth in the size of a representation with the increase in the number of Boolean variables

- **Examples:**
  - A truth table has 2^N rows for a Boolean function of N variables.
    - Truth table size increases exponentially with the number of Boolean variables
  - A logic formula representation of a function: $y = ab + acd + b'd + bc'$ can be very compact

- **Desirable:** a Boolean function representation should be as small as possible.
  - Data structure consumes less memory and easy manipulation

Boolean Function Representations: Canonicity

Canonicity of Representation:

- If a representation is canonical, then the two equivalent functions are represented identically.
- Conversely, if a representation is canonical, and if two functions have the same representation, they are functionally equivalent.

- **Examples:**
  - A truth table is a canonical representation of a Boolean function
  - A logic formula is not canonical:
    - We can represent $y = ab + ac + bc$ as $y = a(b + c) + bc$, $y = ab + c(a + b)$, $y = ab + ac + b(c + aa')$, $y = ab + bc.(a + a') + ac$, etc.

- **Desirable:** In general, we want a representation to be canonical.
  - Manipulating Boolean functions becomes easier in a canonical representation (Example: checking the equivalence of two functions is trivial)

Compact canonical representation of Boolean functions is of great interest for logic synthesis and verification

Binary Decision Diagrams (BDD)

BDD: a data structure to represent Boolean functions canonically
- Is compact for many practically relevant functions
- Boolean operations relevant to formal verification is very efficient

- BDDs are built using *Shannon expansion*
  - Boolean Function is split into two sub-functions by assigning 0/1 to a variable
  - The subfunction obtained by assigning 0 is called negative cofactor
  - The subfunction obtained by assigning 1 is called positive cofactor

$$\boldsymbol{y = ab + acd + b'd + bc'}$$

$a = 0 : y0 = b'd + bc'$
$a = 1 : y1 = b + cd + b'd + bc'$

$$\boldsymbol{y = a'(b'd + bc') + a(b + cd + b'd + bc')}$$

Binary Decision Tree

Boolean functions can be represented as **Binary Decision Tree** by applying **Shannon Expansion Theorem** recursively

$$y = x_1x_2 + x_2'x_3 + x_1'x_3'$$

Level 1: (Expand w.r.t x_1)

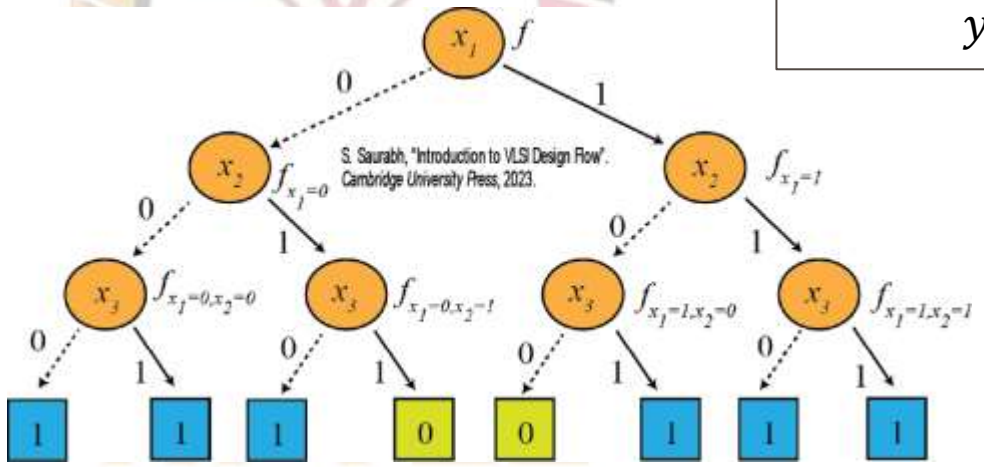
$$y_{x_1=0} = x_2'x_3 + x_3'$$
$$y_{x_1=1} = x_2 + x_2'x_3$$

Level 2: (Expand w.r.t x_2)

$$y_{x_1=0,x_2=0} = x_3 + x_3'$$
$$y_{x_1=0,x_2=1} = x_3'$$
$$y_{x_1=1,x_2=0} = x_3$$
$$y_{x_1=1,x_2=1} = 1$$

Level 3: (Expand w.r.t x_3)

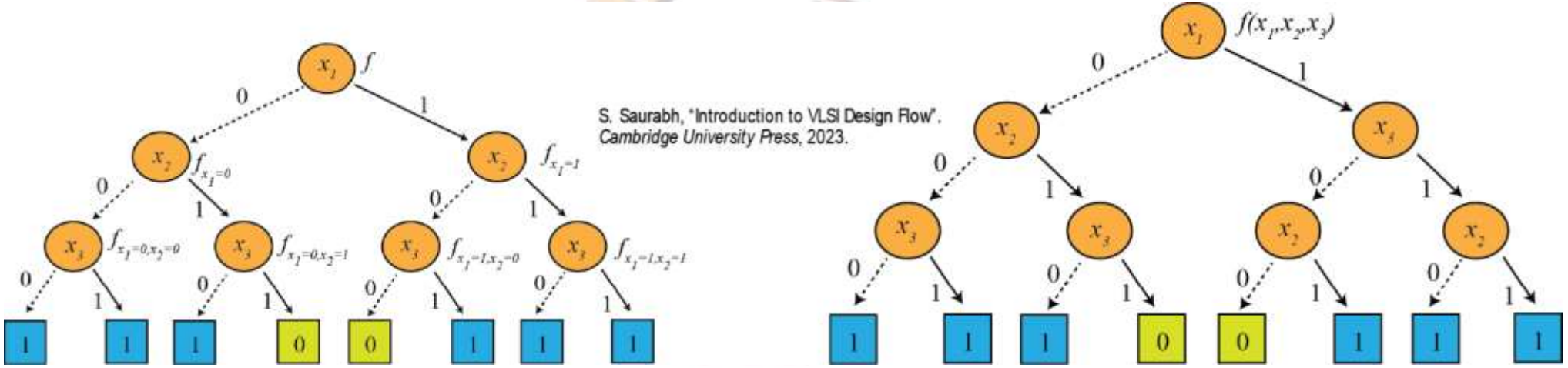
$$y_{x_1=0,x_2=0,x_3=0} = 1$$
$$y_{x_1=0,x_2=0,x_3=1} = 1$$
$$y_{x_1=0,x_2=1,x_3=0} = 1$$
$$y_{x_1=0,x_2=1,x_3=1} = 0$$
$$y_{x_1=1,x_2=0,x_3=0} = 0$$
$$y_{x_1=1,x_2=0,x_3=1} = 1$$
$$y_{x_1=1,x_2=1,x_3=0} = 1$$
$$y_{x_1=1,x_2=1,x_3=1} = 1$$



To make it canonical and compact, we need to add constraint on variable ordering and remove redundancies

Binary Decision Tree: Variable Order

- At an intermediate node, we can choose any variable for expansion in a binary decision tree
- In general, different variable orders can give different binary decision tree



To enforce canonicity, we need to add constraint on the Binary Decision Tree

Ordered Binary Decision Diagram (OBDD): A BDD becomes an OBDD if the decision variables follow the same order in all the paths from the root to the leaf nodes

Ordered Binary Decision Diagram (OBDD)

- Consider a Boolean function $f(x_1,x_2,x_3,\dots,x_n)$ of n Boolean variables $\{x_1,x_2,x_3,\dots,x_n\}$
- OBDD is a rooted, directed, acyclic graph consisting of two types of vertices:
  - Terminal vertices
  - Non-terminal vertices

Terminal vertices (outdegree is 0):
- 0 Node : function assumes a value "0"
- 1 Node : function assumes a value "1"

Each non-terminal vertex v has:
- Two children $low(v)$ and $high(v)$
- Attribute $index(v) \in \{1,2,\dots n\}$ and it refers to the corresponding variable in the set $\{x_1,x_2,x_3,\dots,x_n\}$

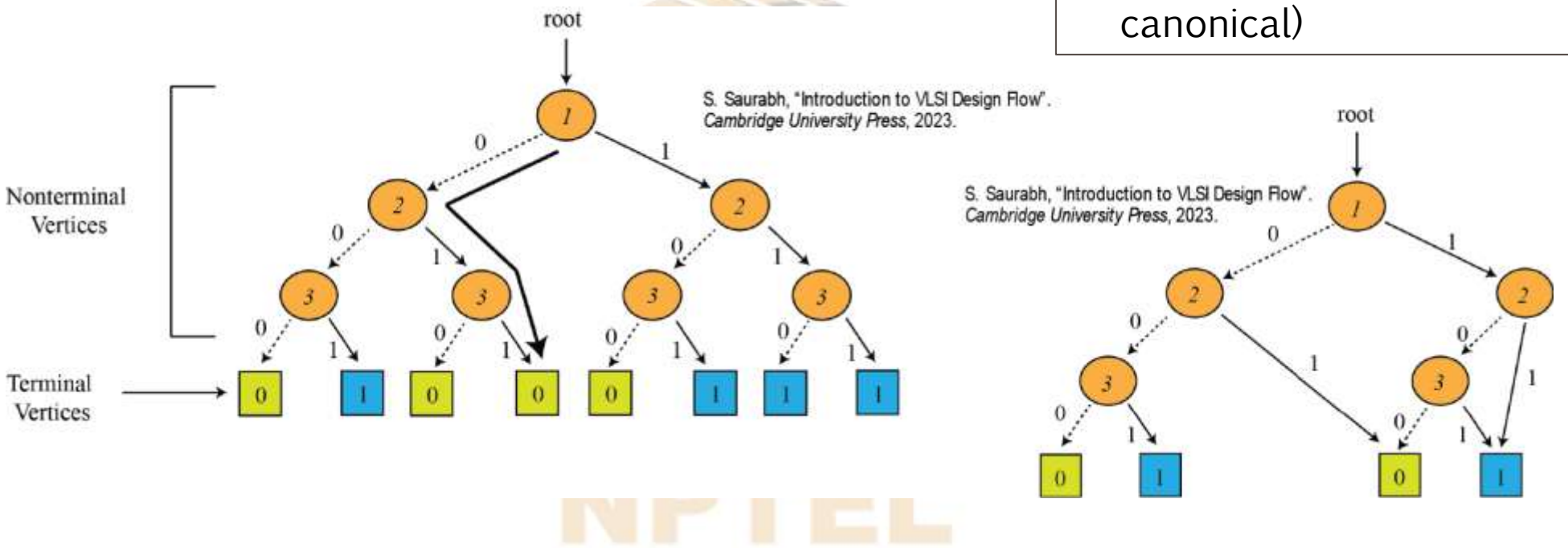
- Edge between a vertex v and $low(v)$ represents the case when the corresponding variable assumes $x_{index(v)} = 0$
- Similarly $\text{high}(v)$ corresponds to $x_{index(v)} = 1$

- To enforce ordering of non-terminal vertices: $index(v) < low(index(v))$ and $index(v) < high(index(v))$

OBDD: Example

$$y = x_1x_2 + x_1x_3 + x_2'x_3$$

- A Boolean function is not uniquely defined by an OBDD (i.e. OBDD is not canonical)

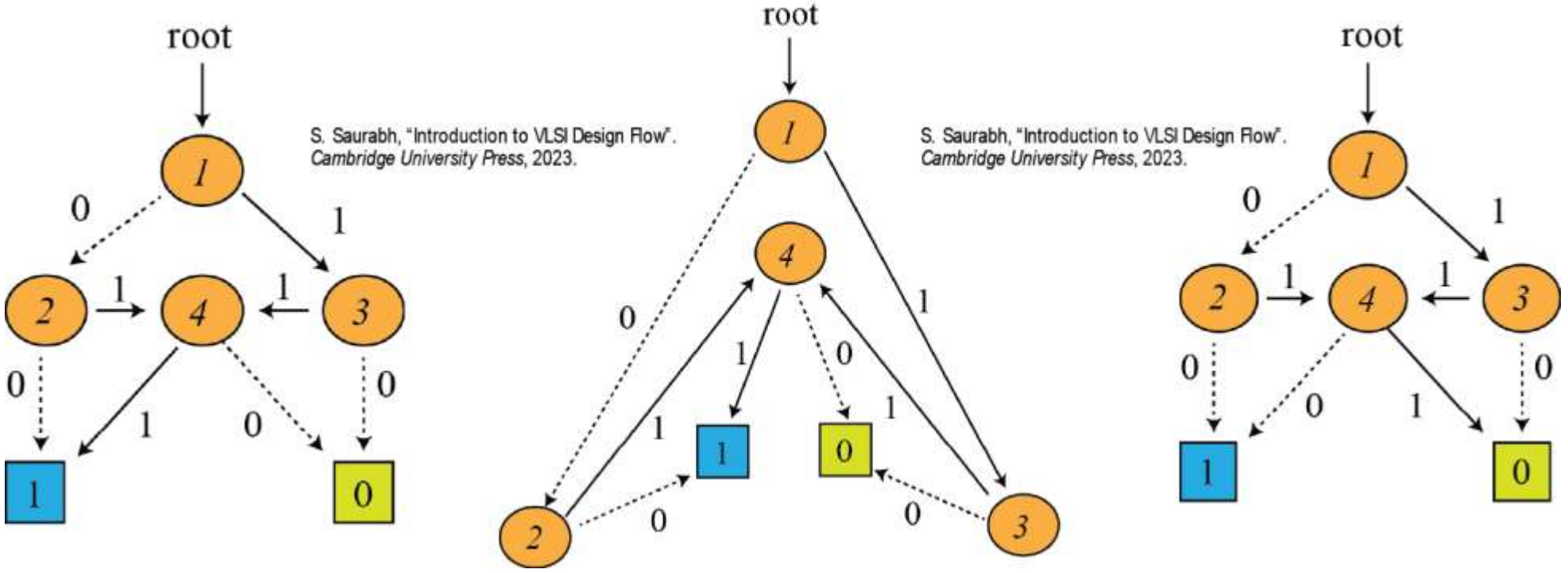


- OBDD can be made canonical by removing redundancies
  - Obtain a Reduced OBDD (ROBDD)

Isomorphic OBDDs

Isomorphism:

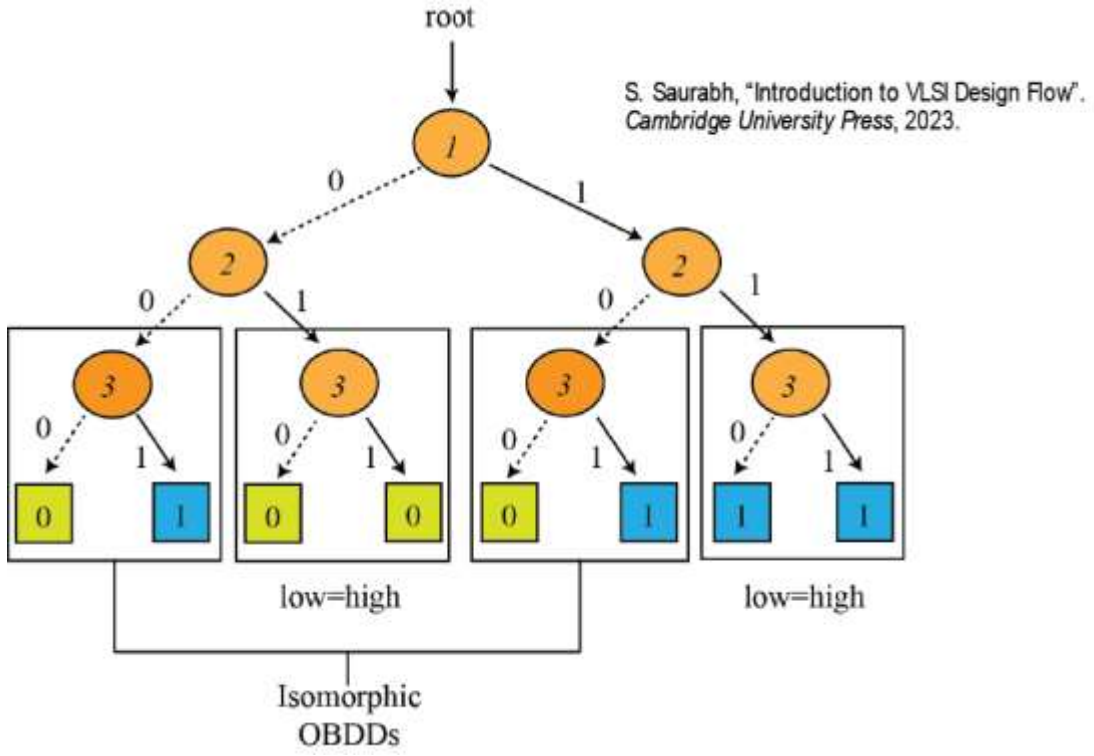
- Two OBDDs $F1$ and $F2$ are isomorphic if there exists a one-to-one mapping between their set of vertices such that the adjacency is preserved.
- The correspondence of the value at the terminal vertices and index at the nonterminal vertices must exist.



Reduced Ordered Binary Decision Diagram (ROBDD)

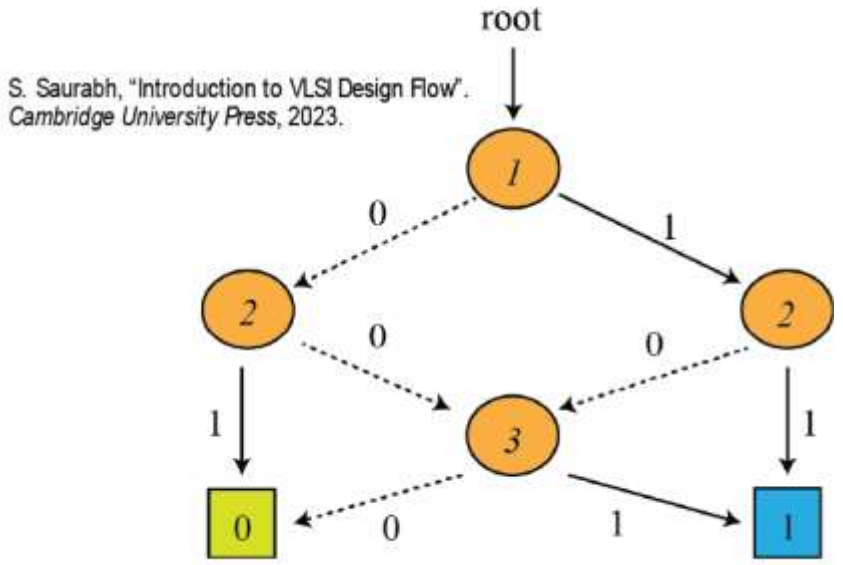
ROBDD is an OBDD with the following constraints:

- No vertex v has $low(v) = high(v)$
- No pair of vertices $\{u, v\}$ exists in the OBDD such that subgraph rooted at u and v are isomorphic.



Can be obtained by systematically removing vertices from the OBDD:

- Any vertex with identical children is removed and replaced with any of its children.
- Two vertices with identical OBDDs are merged into one.



Reduced Ordered Binary Decision Diagram (ROBDD)

- Bryant proved that an ROBDD is a canonical representation of a Boolean function
- When we talk of BDD, we typically refer to ROBDD

Applications:
- Testing equivalence of two functions is easy
- Testing satisfiability and tautology becomes easy

Compactness:
- Size of ROBDDs for many Boolean functions grows as polynomial with the number of variables
- Size of ROBDDs for a given function depends on the variable order.
  - Depending on the variable order, the size of ROBDD can be linear or exponential for an adder circuit
  - Finding a good variable order is difficult
  - For some functions, such as a multiplier, the size of ROBDD is always exponential

References

- R. E. Bryant. “Symbolic Boolean manipulation with ordered binary-decision diagrams.” *ACM Computing Surveys (CSUR) 24*, no. 3 (1992), pp. 293–318.
- S. Saurabh, “Introduction to VLSI Design Flow”. Cambridge: *Cambridge University Press*, 2023.