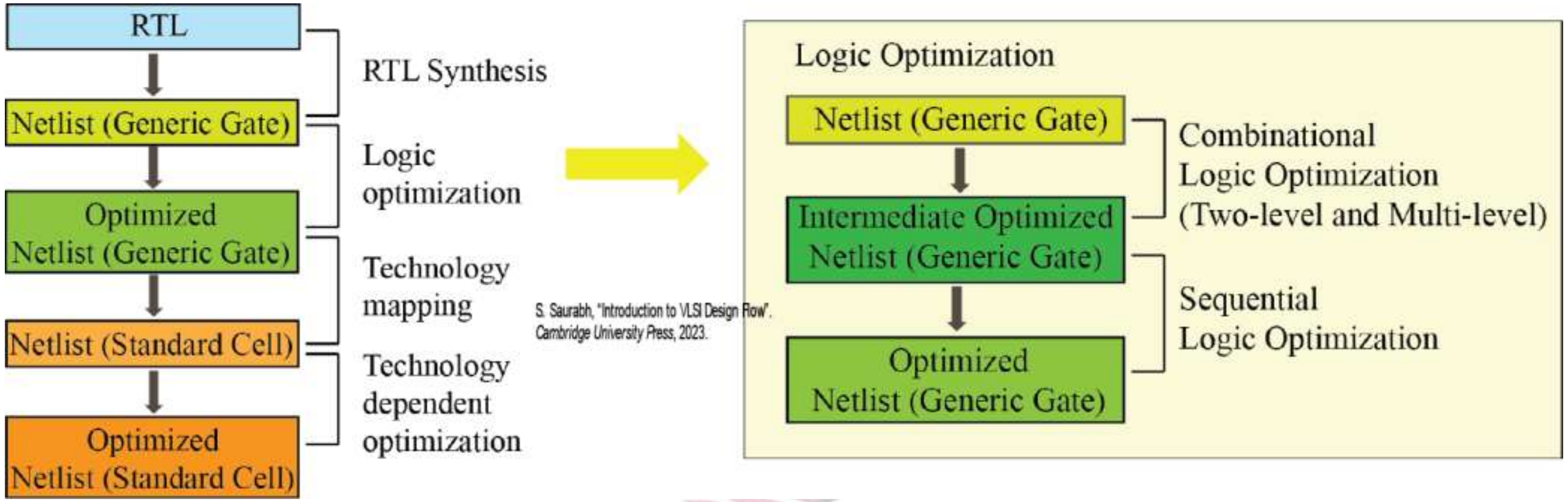
# VLSI DESIGN FLOW: RTL TO GDS

Lecture 15
Logic Optimization: Part II

Sneh Saurabh
Electronics and Communications Engineering
IIIT Delhi

## Lecture Plan


RTL
Netlist (Generic Gate)
Optimized Netlist (Generic Gate)
Netlist (Standard Cell)
Optimized Netlist (Standard Cell)
RTL Synthesis
Logic optimization
Technology mapping
Technology dependent optimization
Logic Optimization
Netlist (Generic Gate)
Intermediate Optimized Netlist (Generic Gate)
Optimized Netlist (Generic Gate)
Combinational Logic Optimization (Two-level and Multi-level)
Sequential Logic Optimization

S. Saurabh, "Introduction to VLSI Design Flow". Cambridge University Press, 2023.

- Multilevel Logic Optimization

# Multilevel Logic Optimization

# Multilevel Logic Optimization: Introduction

**Multilevel Logic:** more than two levels of logic

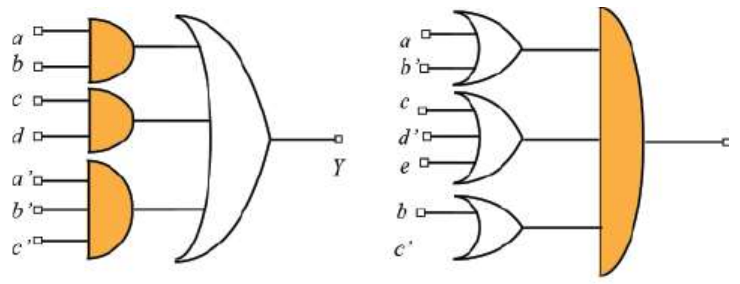
- Can appear naturally in RTL

<u>Limitations of two-level logic</u>

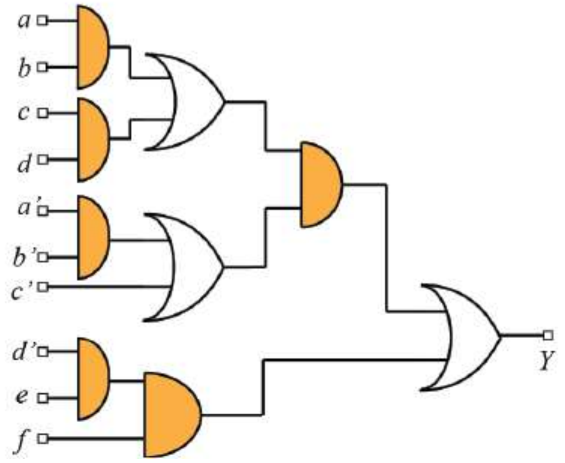
- Sum of Product (SOP) representations of some functions can become too big
  - Example: parity functions, adders, and multipliers.
- Cannot reduce area by trading off speed
  - Multilevel logic circuits offer more flexibility in exploring area–delay trade-off

<u>Multilevel Logic Representations</u>

- Factored Form
- Boolean Logic Network



S. Saurabh, "Introduction to VLSI Design Flow". *Cambridge University Press*, 2023.



# Factored Form

A factored form consists of:

a) literals

b) sum (logical OR) of factored form

c) product (logical AND) of factored form.

A factored form is an SOP, of SOP, of SOP, …, of arbitrary depth.

- Given an SOP, it can be converted to a factored form by ***factoring***
- Factoring can convert a two-level representation to a multilevel representation.

- Given an SOP, its factored form is not unique

- Consider the following Boolean function in the SOP form: $ac + ad + bc + bd + ce$
- We can obtain a factored form representation as:

$(a + b)(c + d) + ce$

$(a + b)d + (a + b + e)c$

- Number of literals in the factored form correlates with the circuit area

## Boolean Logic Network (1)

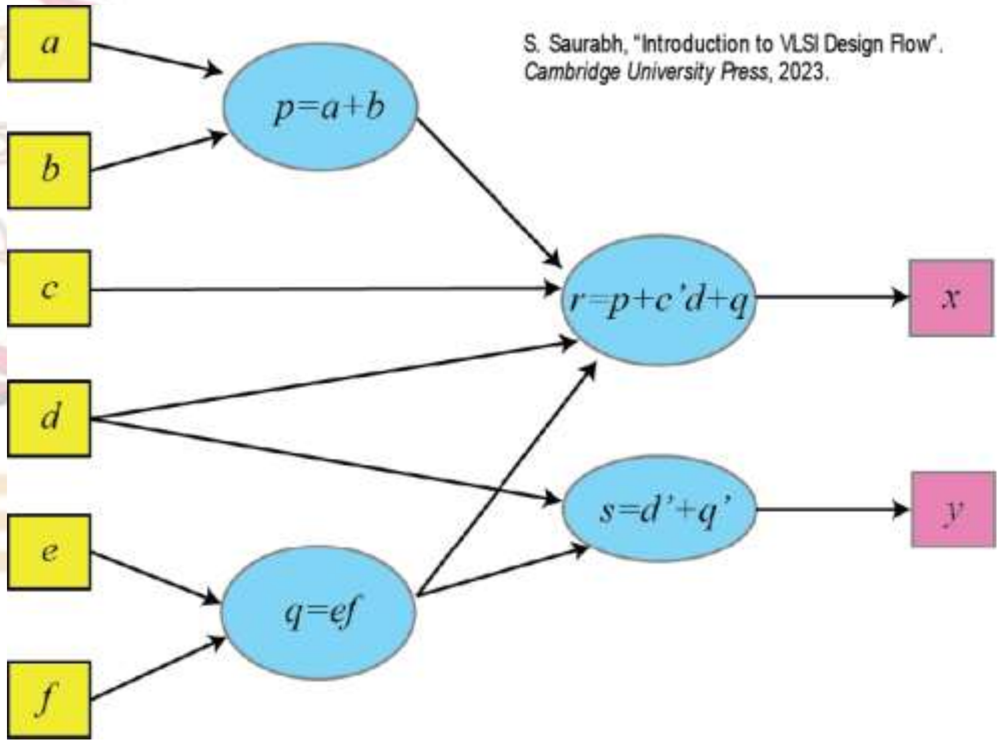
- Boolean logic network is a directed acyclic graph
  - ➢ annotate each vertex with a single-output local Boolean function
- It can conveniently represent multilevel logic circuit with more than one output.

- Consider the following set of equations:

$$p = a + b$$
$$q = ef$$
$$r = p + c'd + q$$
$$s = d' + q'$$
$$x = r$$
$$y = s$$

- Assume that inputs are $a, b, c, d, e,$ and $f$.
- Assume that $x$ and $y$ are outputs

- The incoming edges to a vertex denote the variables on which the local function at that vertex depends



S. Saurabh, "Introduction to VLSI Design Flow". Cambridge University Press, 2023.

## Boolean Logic Network (2)

**Flexibility of Boolean logic network:**

- Local functions in a Boolean logic network can be arbitrarily complicated
- Both its underlying graph and the local functions can be manipulated during optimization
- Optimization can explore both the behavioral and the structural features of the implementation

**Estimating Area and Delay:**

- Local functions restricted to be in an SOP form and made *minimal with respect to single-implicant containment*
- **Estimating area:** sum of all the literal counts of the local functions
- **Estimating delay:** number of stages of vertices in the logic network

# Multilevel Logic Optimization

## Transformations

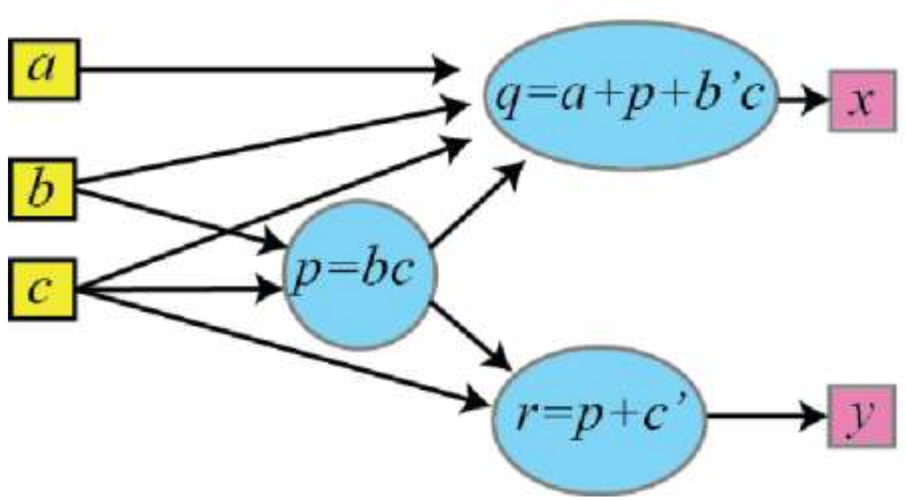
## Transformations

- Multilevel logic optimization is performed by applying transformations.
  - These transformations can be viewed as **operators** for the Boolean logic network.

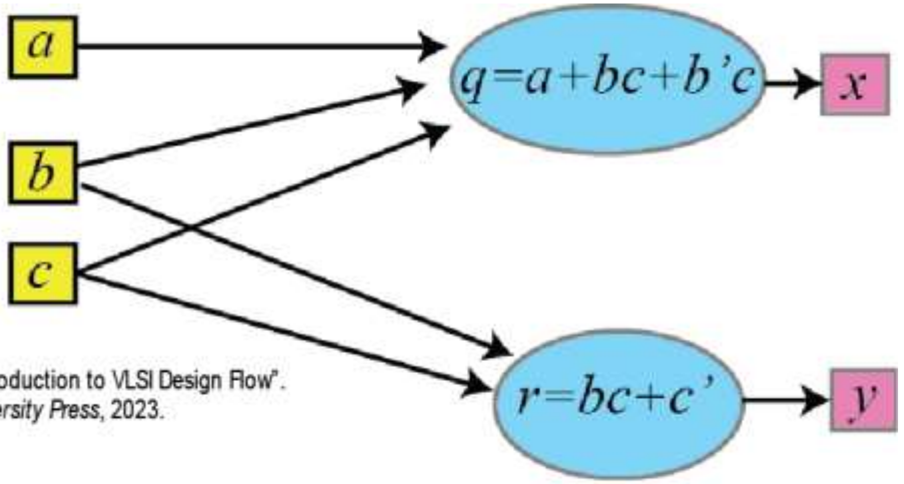
- These operators applied iteratively until no more improvement in some QoR measures is possible.
  - The final QoR depends on the order of operation and is hard to predict.

# Eliminate

- It removes a vertex from the graph and replaces all its occurrences in the network with the corresponding local function



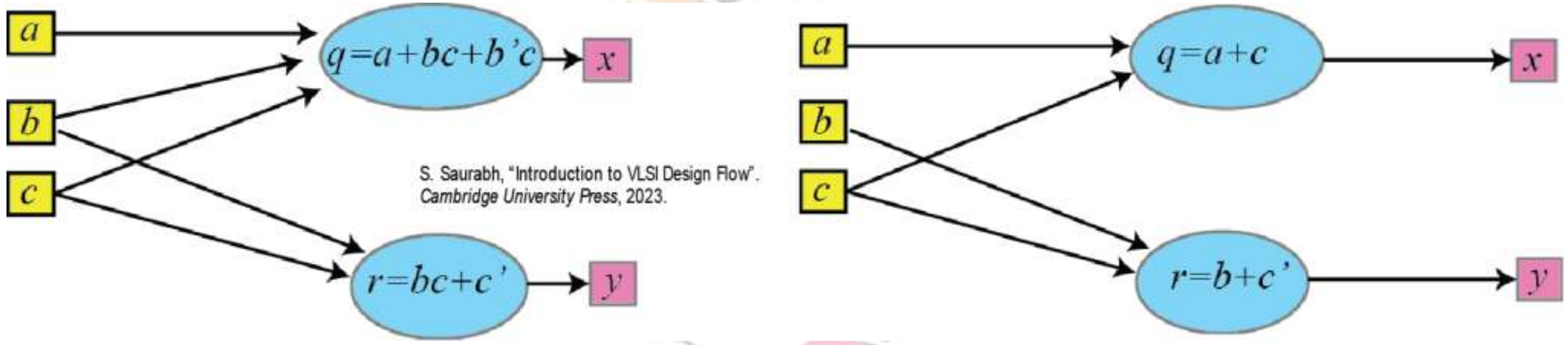
S. Saurabh, "Introduction to VLSI Design Flow". *Cambridge University Press*, 2023.



- We carry out eliminate in the hope that subsequent transformations can reduce the cost (area)

## Simplify

- It simplifies the associated local SOP expression to reduce the literal count (two-level logic optimization carried out on each local function individually)

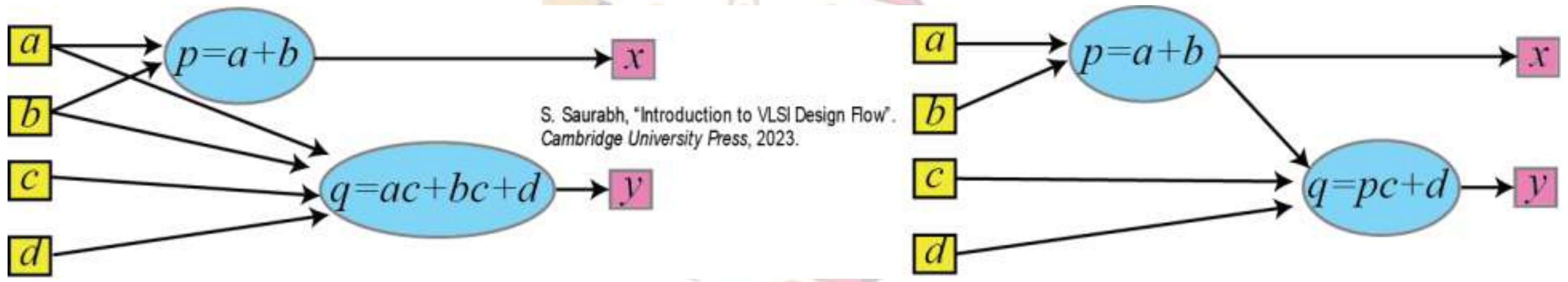


S. Saurabh, "Introduction to VLSI Design Flow". *Cambridge University Press*, 2023.

- Reduces the literal count from 8 to 4

## Substitute

- It replaces the local function with a simpler SOP by creating new dependencies and possibly removing other dependencies.
- Creates dependencies by searching for an appropriate match.
  - ➢ It adds more structural information to the network.

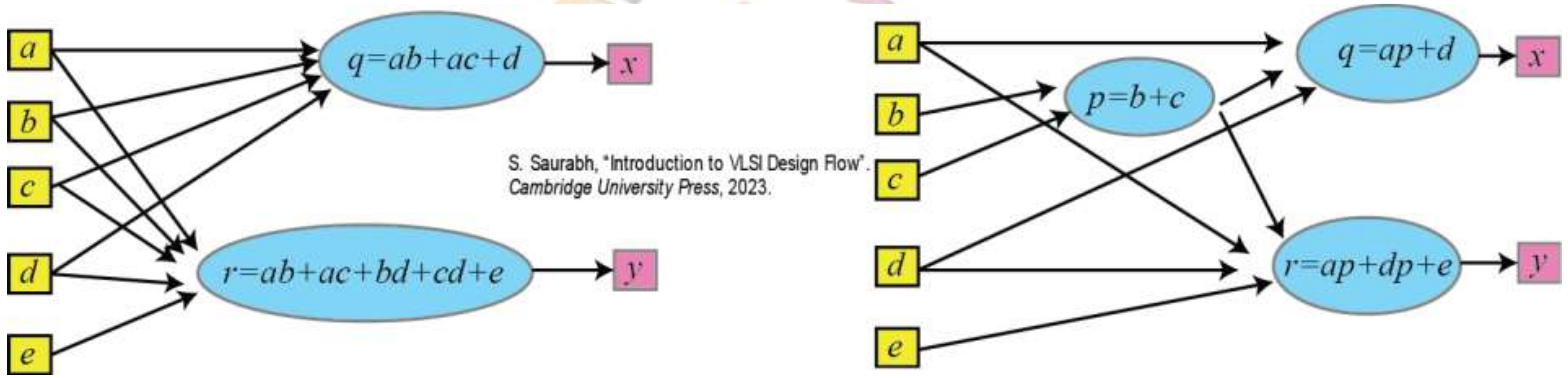

a
b
c
d
p=a+b
q=ac+bc+d
x
y
S. Saurabh, "Introduction to VLSI Design Flow". Cambridge University Press, 2023.
a
b
c
d
p=a+b
q=pc+d
x
y

- Reduces the literal count from 7 to 5

- Substitute operator needs to find whether local function $f_i$ divides another local function $f_j$
  - ➢ If it divides, we can replace $f_j = Q.f_i + R$
  - ➢ Need to perform division efficiently

## Extract

- It finds a common subexpression for functions associated with two or more vertices.
  - ➢ Subsequently, it creates a new vertex associated with the subexpression.
  - ➢ Then replaces the common subexpressions in the original functions with the variable of the new vertex.



S. Saurabh, "Introduction to VLSI Design Flow". Cambridge University Press, 2023.

- Reduces the literal count from 14 to 10

- Extract operator needs to find divisors for the local functions and then search vertices with matching divisors.
  - ➢ Implementing extract operator is computationally challenging

## Challenges of Multilevel Logic Optimization

- Huge search space for optimization

**Division Operation:**

- During multilevel logic optimization, we need to carry out division too many times.
- Practical circuits have thousands of vertices in the Boolean logic network.
  - Might need to divide each vertex with the rest [$O(n^2)$ times, where $n$ is the number of vertices in the Boolean logic network]
- Efficiency of division of expressions is critical for multilevel logic optimization

**Divisors:**

- Need to find good divisors (one that can reduce cost) for Boolean expressions
- Finding a good set of divisors for a given Boolean expression is nontrivial

## Algebraic Model

**Algebraic Model:**

- Neglecting some Boolean properties of the local functions
- Simplified model treats the local Boolean functions as polynomials and employs rules of polynomial algebra
  - Treat a variable and its complement as separate variables.

**Applications of algebraic model:**

- Efficient algorithms can be designed to carry out division in the algebraic model (rather than in the Boolean model)
- Good divisors or common subexpressions in a complex Boolean logic network can be determined efficiently in a complex Boolean logic network
  - By intelligently pruning the search space (applying properties of algebraic models)

## Algebraic Model versus Boolean Model

- Algebraic model (and associated mathematics) form the basis of fast multilevel optimization in the contemporary logic optimization tools.

**Boolean Model:**

- An algebraic model is weaker than a Boolean model for optimization
  - Cannot fully optimize a Boolean logic network.
- Post algebraic model-based optimization, transformations that utilize the power of the Boolean model is applied

## Multilevel Logic Optimization

### Boolean Model

# Don’t Care (DC) Conditions

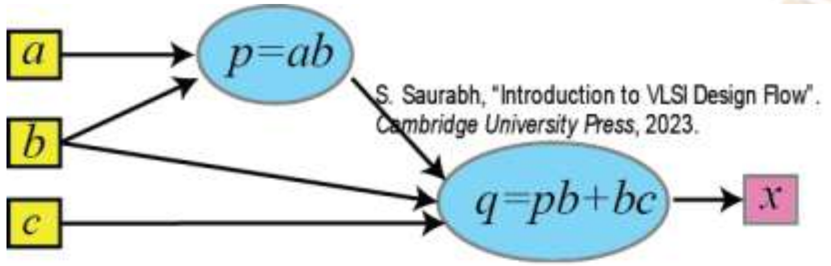
- Don’t Care (DC) conditions arise naturally in Boolean logic network
  - Due to the graph structure and dependencies among local functions.
- DCs are a rich source of optimization in multilevel logic synthesis
  - Simplifying local functions and improving the circuit’s overall QoR.
- A logic synthesis tool needs to discover them using Boolean algebra (in contrast to given DCs)

There are two types of DC that are useful in simplifying local functions in a Boolean logic network:

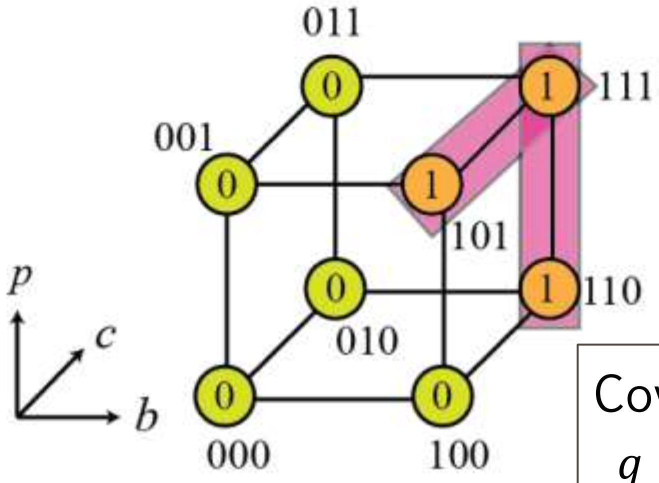
1. Controllability Don’t Cares (CDCs)
2. Observability Don’t Cares (ODCs).

# Controllability Don't Cares (CDCs)

- The combination of input variables that can never occur at a given vertex in a Boolean logic network produces CDCs.
- Local functions can be simplified by accounting for CDCs and two-level logic minimizers

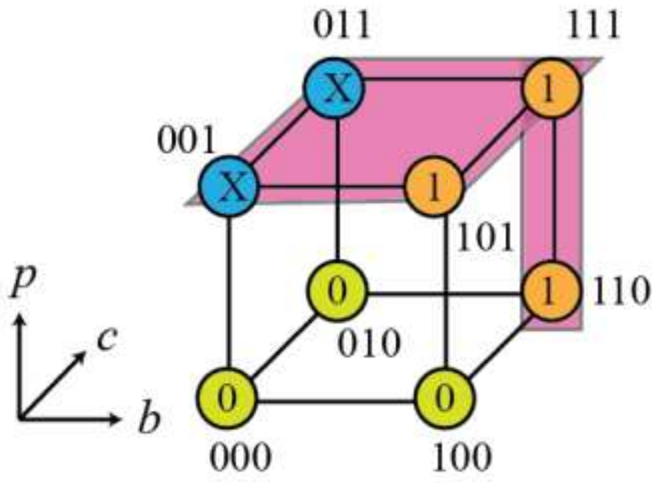

a
b
c
p=ab
q=pb+bc
x
S. Saurabh, "Introduction to VLSI Design Flow". Cambridge University Press, 2023.

- At the input of vertex $q$, $p = 1$ and $b = 0$ can never occur
- $pb'$ can be treated as CDC



Cover is:

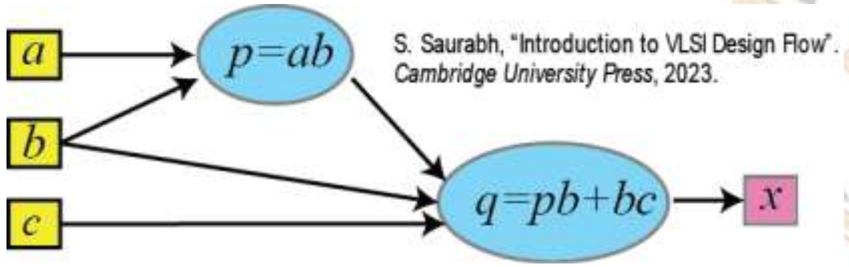
$q = pb + bc$



- Cover is: $q = p + bc$
- Reduces the literal count by 1

## Satisfiability Don’t Cares (SDCs)

- CDCs can be computed using efficient algorithms by exploiting Satisfiability Don’t Cares (SDCs).
- SDCs get enforced by the local functions associated with a vertex at its output.



S. Saurabh, “Introduction to VLSI Design Flow”. Cambridge University Press, 2023.

Consider the vertex $p = ab$

- The following function will never evaluate to 1:

$$p \oplus ab = pa' + pb' + p'ab$$

- Hence, the combination of values that make the above function 1 can never occur in the network:

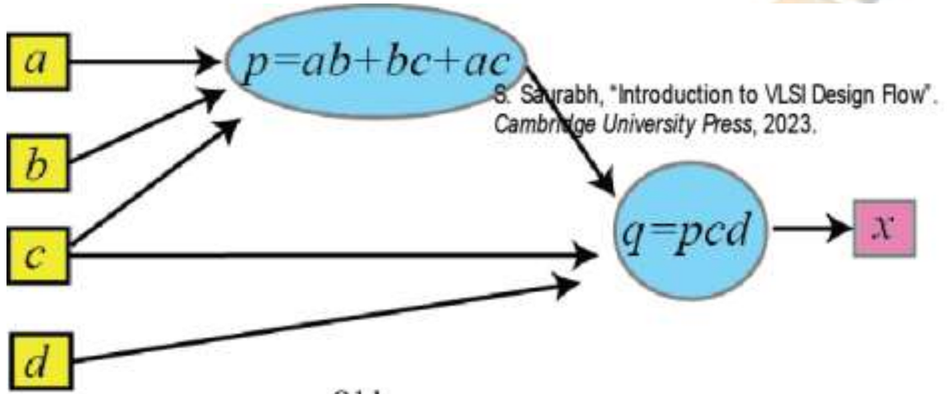
$$p = 1, a = 0,$$
$$p = 1, b = 0,$$
$$p = 0, a = 1, b = 1$$

- These values can be treated as DCs for the Boolean logic network

- Logic synthesis tools typically derive CDCs algorithmically using SDCs.
- Subsequently, the CDCs get utilized in simplifying local functions.

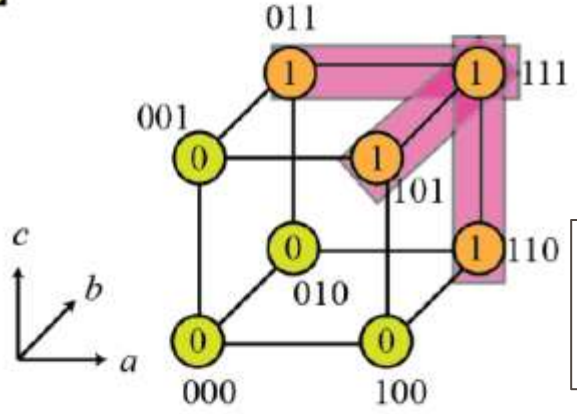
# Observability Don’t Cares (ODCs)

- The ODCs are input variable combinations that obstruct the vertex output from being observed at the network output.
- We derive ODCs induced by vertices in the fanout of a given vertex.



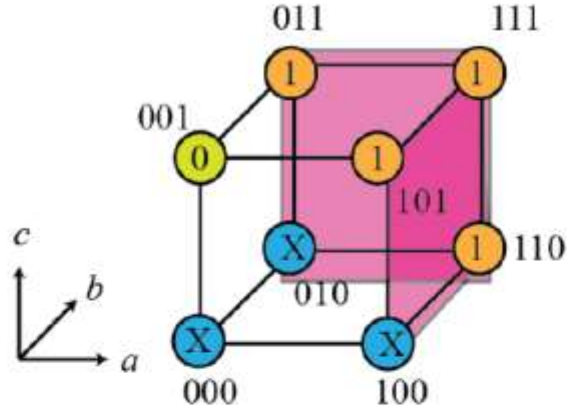
a
b
c
d
p=ab+bc+ac
q=pcd
x
S. Saurabh, "Introduction to VLSI Design Flow". Cambridge University Press, 2023.

- If $c = 0$, then $x = 0$ irrespective of $p$
- $c = 0$, can be treated as ODC for $p$



Cover is:

$p = ab + bc + ac$



- Cover is: $p = a + b$
- Literal count reduces from 9 to 5

## References

- G. D. Micheli. “Synthesis and Optimization of Digital Circuits”. *McGraw-Hill Higher Education*, 1994.
- S. Saurabh, “Introduction to VLSI Design Flow”. Cambridge: *Cambridge University Press*, 2023.