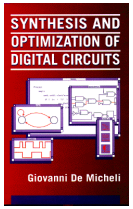
Introduction to Logic Synthesis

Giovanni De Micheli

Integrated Systems Laboratory



Module 1

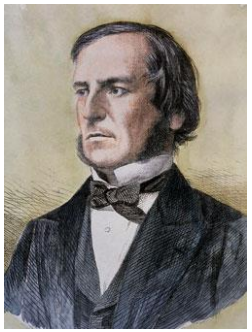
- **Objectives**
  - **Fundamentals of logic synthesis**
  - **Mathematical formulation**
  - **Definition of the problems**

What is logic synthesis?

- A *logic-level representation* is:
  - A Boolean function
  - A set of Boolean functions and their dependences
  - A schematic with logic gates
  - A logic-level HDL description
- Logic synthesis is the process of optimizing logic representations with the final goal of:
  - Speeding up a circuit
  - Reducing its area and manufacturing cost
  - Reducing the energy consumption

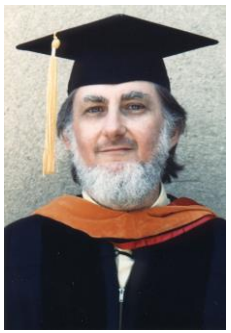
Logic synthesis has a long history

- George Boole:
  - Boolean Algebra



- Claude Shannon
  - Switching Theory

- Modern logic synthesis
  - Ed McCluskey, Robert Brayton, Randy Bryant
    - … and many other noteworthy scientists



Combinational logic design
Background

- **Boolean Algebra**
  - **Quintuple (B, +, . , 0, 1)**
  - **Binary Boolean algebra B = { 0, 1 }**
- **Boolean function**
  - **Single output $f : B^n \rightarrow B$**
  - **Multiple output $f : B^n \rightarrow B^m$**
  - **Incompletely-specified:**
    - ***Don't care* symbol: ***
    - **$f : B^n \rightarrow \{ 0, 1, * \}^m$**

Examples of Boolean Algebras

- **Algebra of classes:**
  - Universal set **S** – classes are subsets of **S**
  - 2^S set of subsets of S
  - Quintuple $(2^S, U, \Pi, \emptyset, S)$
- **Arithmetic Boolean algebra**
  - **n** = product of distinct primes
  - D_n = divisors of n
  - Quintuple $(D_n, lcm, gcd, 1, n)$
  - Example n = 30, then ({1,2,3,5,6, 10, 15,30}, lcm, gcd, 1, 30)

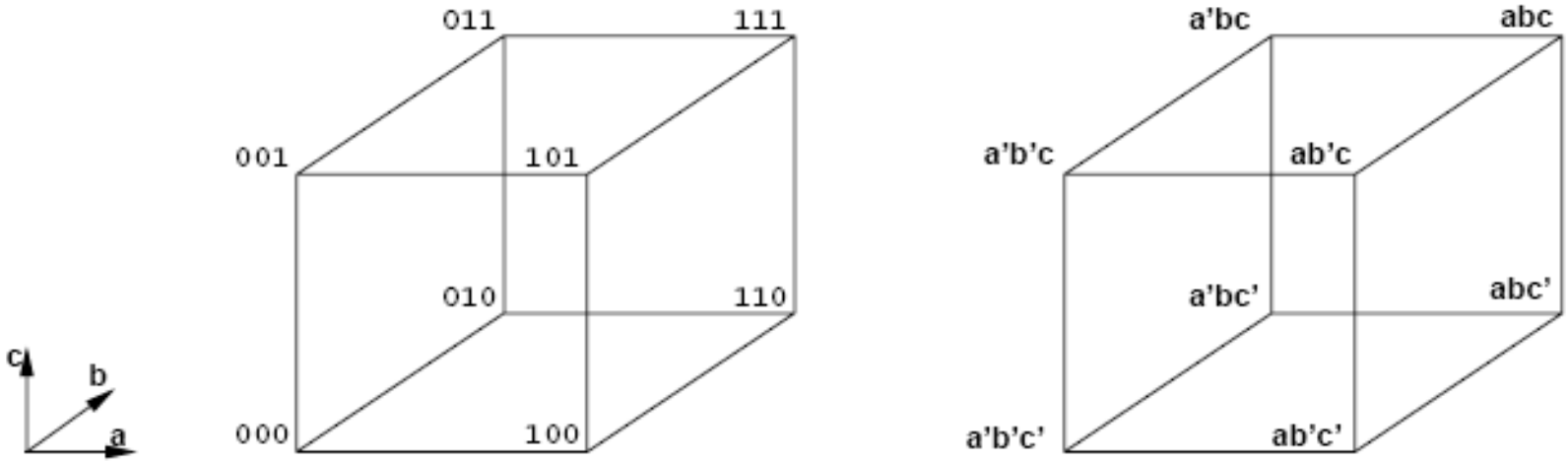
The *don't care* conditions

- We do not care about the value of a function
- Related to the environment
  - Input patterns that never occur
  - Input patterns such that some output is never observed
- Very important for synthesis and optimization

Definitions

- **Scalar function:**
  - **ON-set**
    - **Subset of the domain such that f is true**
  - **OFF-set**
    - **Subset of the domain such that f is false**
  - **DC-set**
    - **Subset of the domain such that f is a *don't care***
- **Multiple-output function:**
  - **ON, OFF, DC-sets defined for each component**

Cubical representation



Definitions

- Boolean variables
- Boolean literals:
  - Variables and their complement
- Product or cube:
  - Product of literals
- Implicant:
  - Product implying a value of the function (usually 1)
  - Hypercube in the Boolean space
- Minterm:
  - Product of all input variables implying a value of the function (usually 1)
  - Vertex in the Boolean space

Tabular representations

- **Truth table**
  - List of all minterms of a function
- **Implicant table** or **cover**
  - List of implicants sufficient to define a function
- **Note:**
  - Implicant tables are smaller in size as compared to truth tables

Example of truth table

- $x = ab + a'c; \quad y = ab + bc + ac$

abc	xy
000	00
001	10
010	00
011	11
100	00
101	01
110	11
111	11

Example of an encoded truth table

abc	xy
000	00
001	10
010	00
011	11
100	00
101	01
110	11
111	11

- Using the reverse order:
  - x = 11001010
  - y = 11101000
- Encoded in hexadecimal:
  - x = ca
  - y = e8

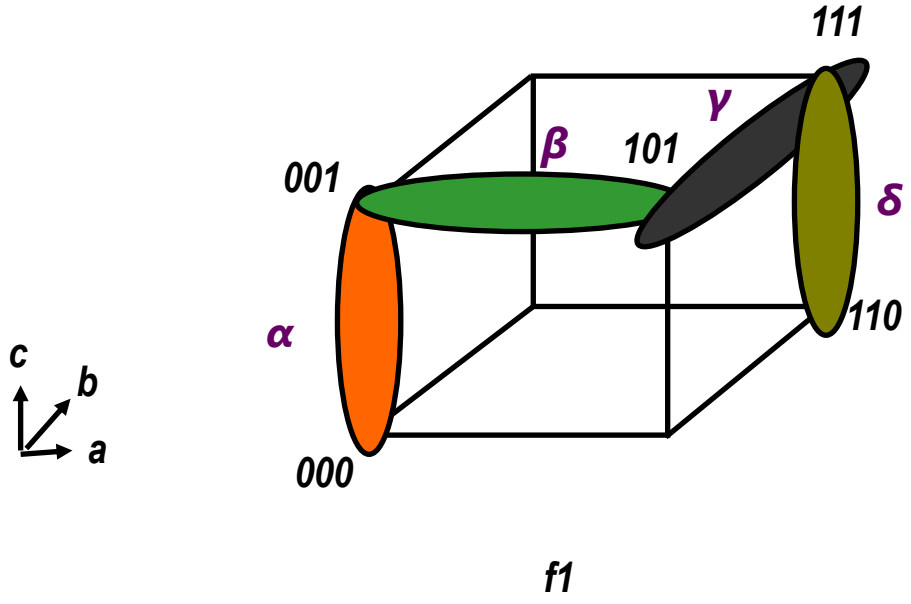
Example of implicant table

- x = ab+a’ c; y = ab+bc+ac

abc	xy
001	10
*11	11
101	01
11*	11

Cubical representation of minterms and implicants

- $f_1 = a' b' c' + a' b' c + a b' c + a b c + a b c'$
- $f_2 = a' b' c + a b' c$



Representations

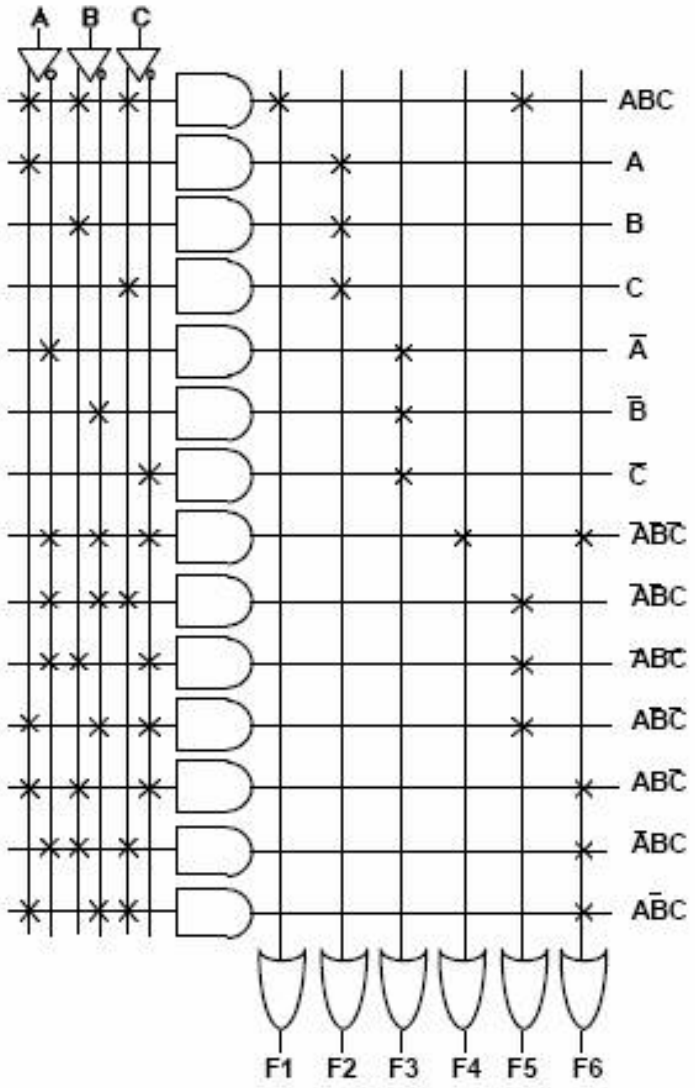
- Visual representations
  - Cubical notation
  - Karnaugh maps
- Computer-oriented representations
  - Matrices
    - Sparse
    - Various encoding
  - Binary-decision diagrams
    - Address sparsity and efficiency

Two-level logic optimization motivation

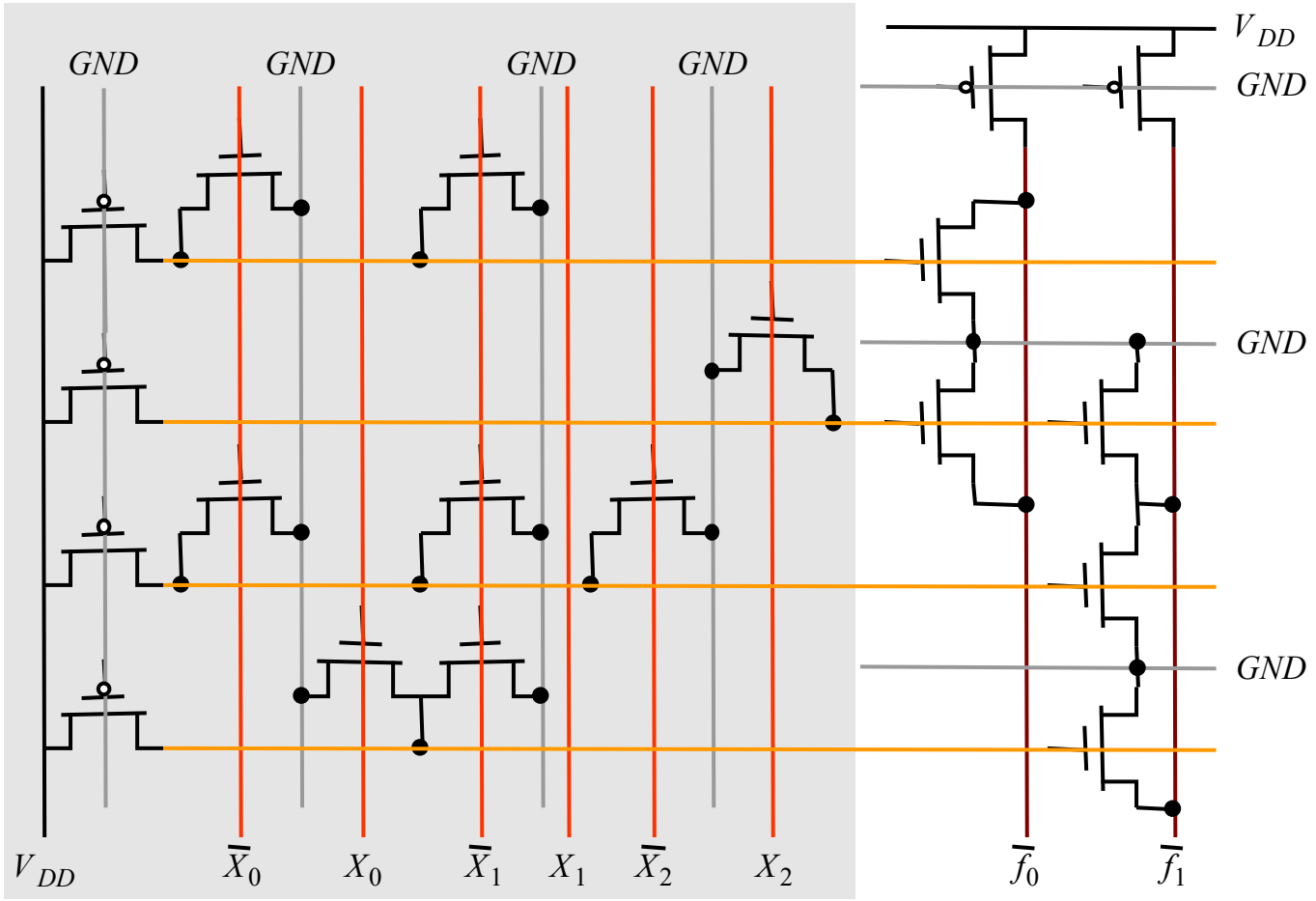
- Reduce size of the representation
- Direct implementation
  - PLAs reduce size and delay
- Other implementation styles
  - Reduce amount of information
  - Simplify local functions and connections

Programmable logic arrays

- **Macro-cells with rectangular structure**
  - Implement any multi-output function
- **Programmable**
  - Old technology using fuses
  - Grandfather of FPGAs
- **Programmed**
  - Layout generated by module generators
  - Fairly popular in the seventies/eighties
- **Advantages**
  - Simple, predictable timing
- **Disadvantages**
  - Less flexible than cell-based realization
  - Dynamic operation in CMOS
- **Open issue**
  - Will PLA structures be useful with new nanotechnologies?



Pseudo NMOS PLA



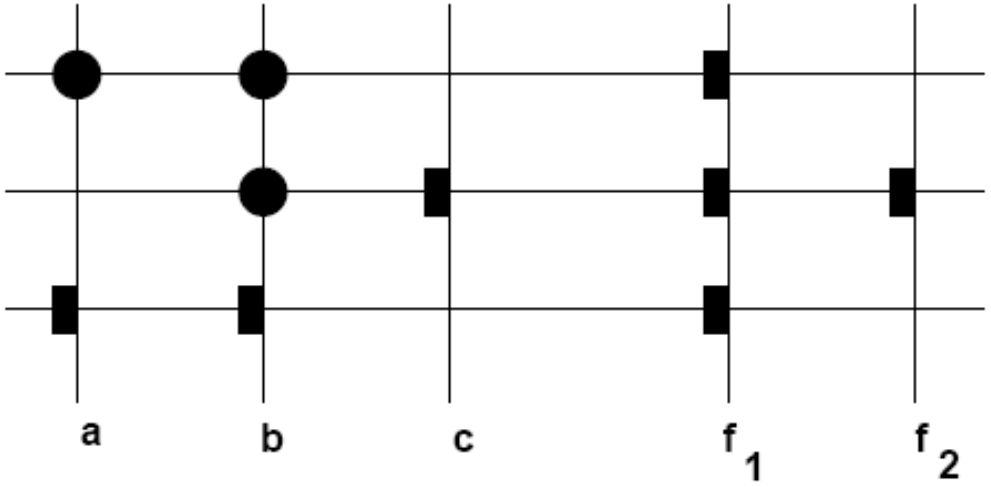
AND-plane **OR-plane**

Programmable logic array

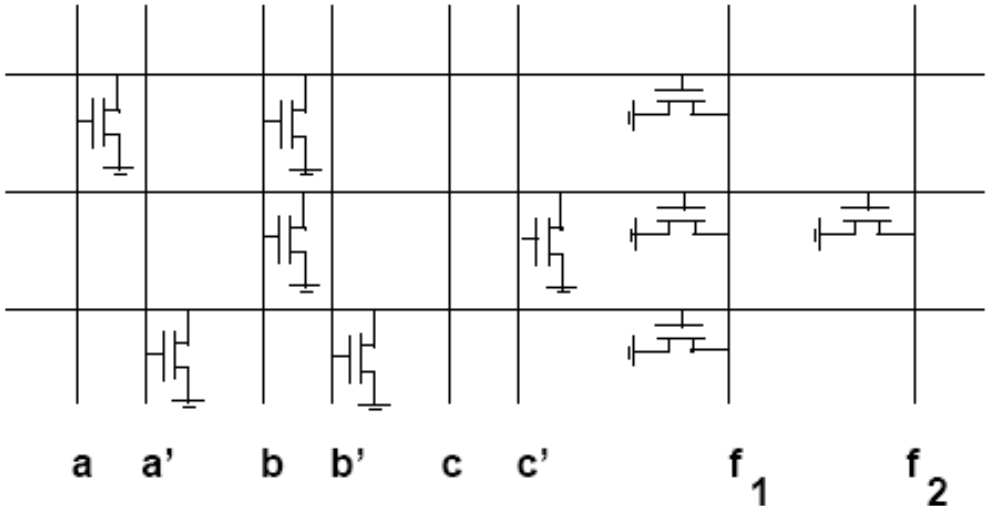
- $f_1 = a'b' + b'c + ab$; $f_2 = b'c$

00* 10
*01 11
11* 10

(a)



(b)



(c)

Two-level minimization

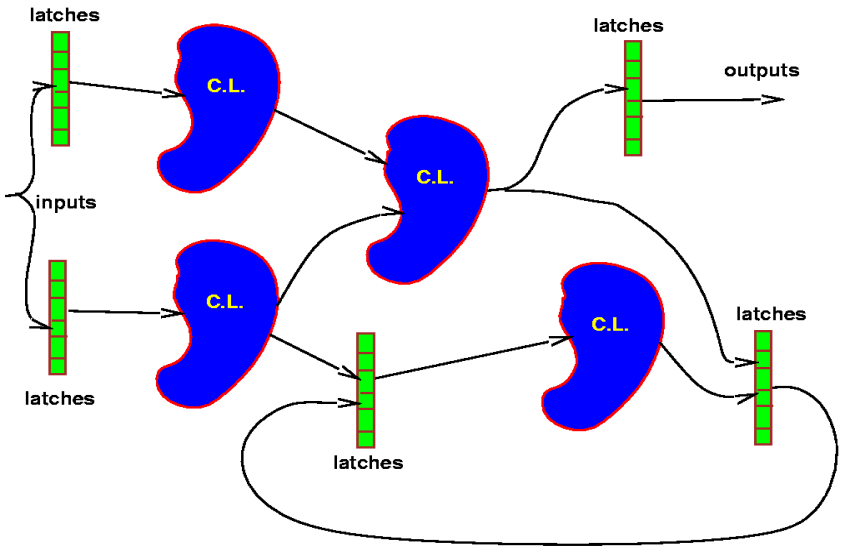
- **Assumptions**
  - **Function represented as a Boolean cover**
    - **Set of implicants (covering all minterms)**
  - **Primary goal is to reduce the number of implicants**
  - **All implicants have the same cost**
  - **Secondary goal is to reduce the number of literals**
- **Rationale**
  - **Implicants correspond to PLA rows**
  - **Literals correspond to transistors**

Module 2

- **Objectives**
  - **What is multi-level logic synthesis**
  - **What are the specific goals**
  - **Stepwise transformations**

Motivation

- Multiple-level logic networks
  - Semi-custom libraries
  - Logic gates versus macro-cells
    - More flexibility
    - Privilege specific paths on others
    - Better performance

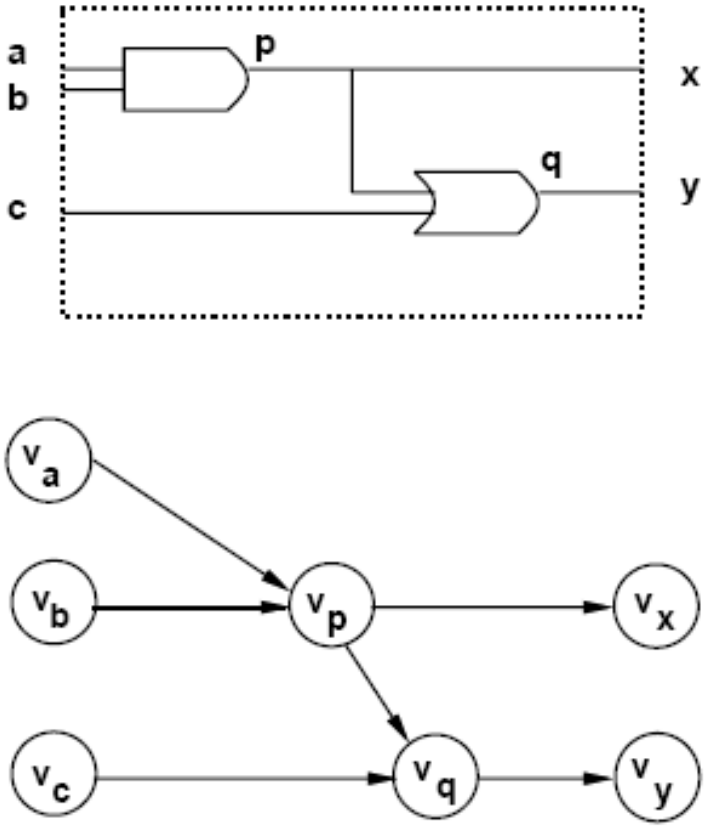


- Applicable to a large variety of designs
- The importance of logic synthesis grew in parallel with the growth of foundries for the semi custom market

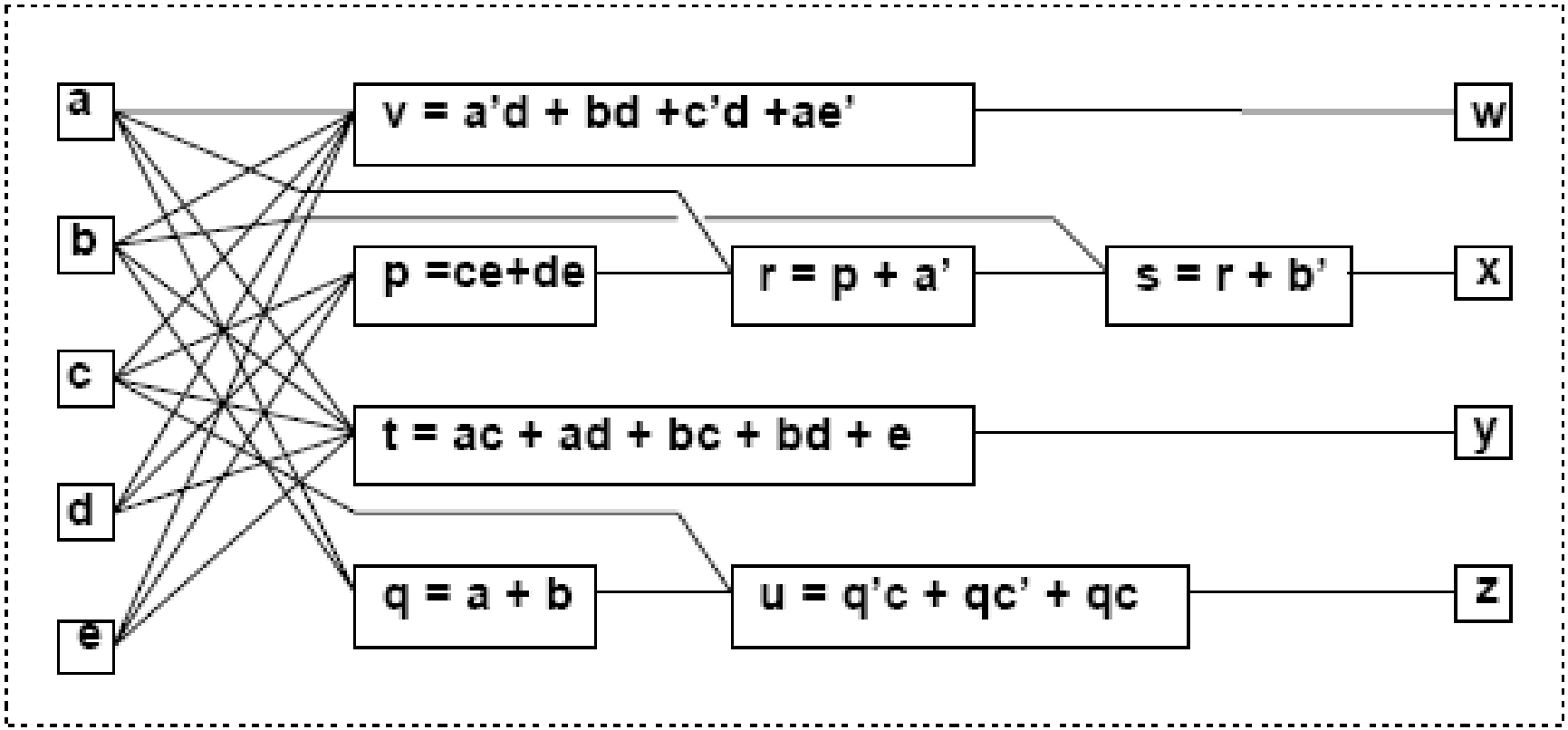
Circuit model

- **Logic network**
  - **An interconnection of blocks**
    - **Each block modeled by a Boolean function**
  - **Usual restrictions:**
    - **Acyclic and memoryless**
    - **Single-output functions**
- **The model has a structural/behavioral semantics**
  - **The structure is induced by the interconnection**
- **Mapped network**
  - **Special case when the blocks correspond to library elements**

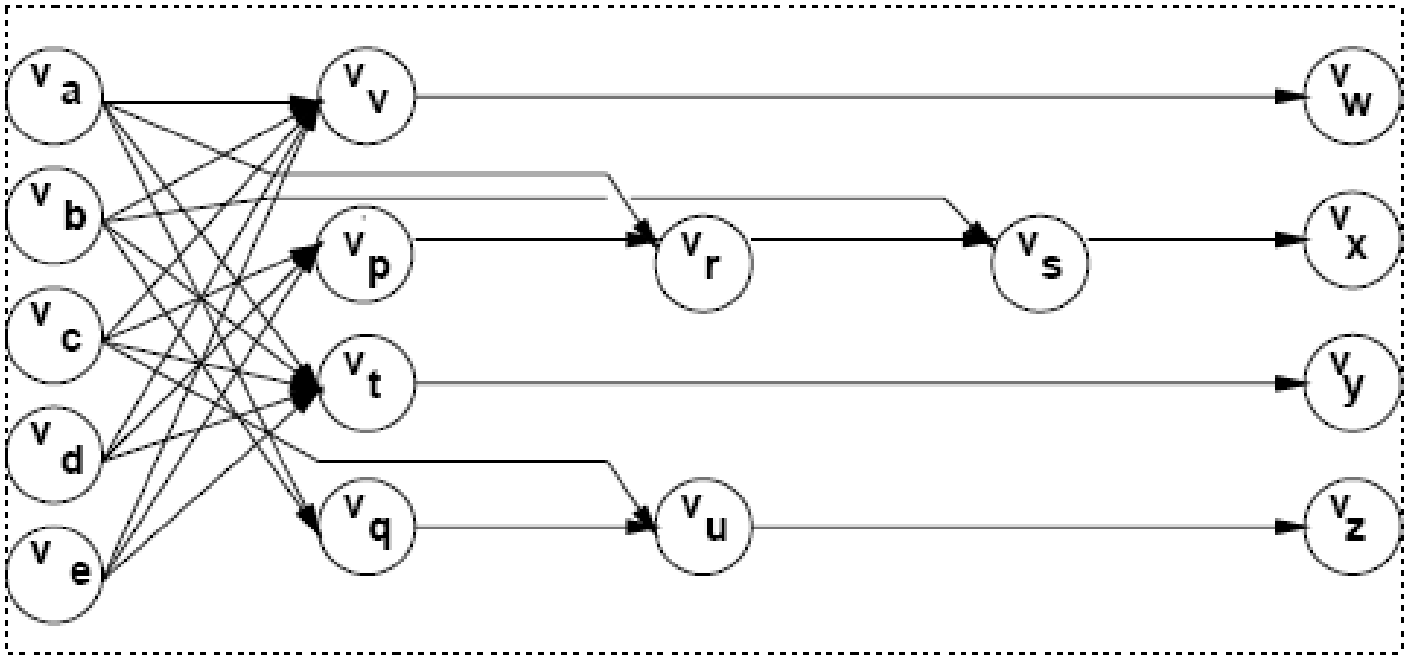
Example of mapped network



Example of general network



Example of general network graph



Network represented by assignments

$$
\begin{aligned}
p &= ce + de \\
q &= a + b \\
r &= p + a' \\
s &= r + b' \\
t &= ac + ad + bc + bd + e \\
u &= q'c + qc' + qc \\
v &= a'd + bd + c'd + ae' \\
w &= v \\
x &= s \\
y &= t \\
z &= u
\end{aligned}
$$

Example of terminal behavior

- I/O functional behavior
  - Vector with as many entries as primary outputs
  - Each entry is a logic function

$$f = \begin{bmatrix} a'd + bd + c'd + ae' \\ a' + b' + ce + de \\ ac + ad + bc + bd + e \\ a + b + c \end{bmatrix}$$

Network optimization

- Minimize maximum delay
  - (Subject to area or power constraints)
- Minimize area
  - Subject to delay constraints
- Minimize power consumption
  - Subject to timing constraints

Estimation

- **Area:**
  - **Number of literals**
    - **Easy, widely accepted, good estimator**
- **Delay:**
  - **Number of stages (under fanout constraint?)**
  - **Gate delay models with wire-loads**
  - **Sensitizable paths**
- **Power**
  - **Switching activity at each node**
  - **Capacitive loads**

Problem analysis

- Even the simplest problems are computationally hard
  - E.g., multi-input single-output network
- Few exact methods proposed
  - High complexity
  - Practical for small circuits only, but… useful!
- Approximate optimization methods
  - Heuristic algorithms
  - Rule-based methods

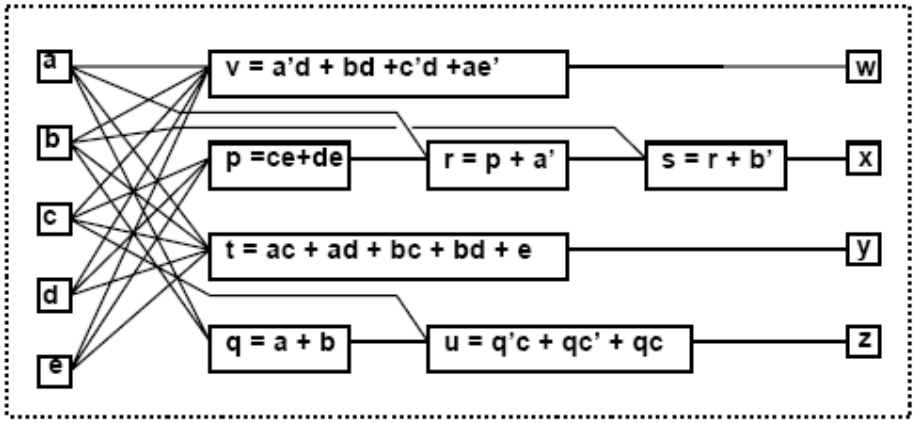
Strategies for optimization

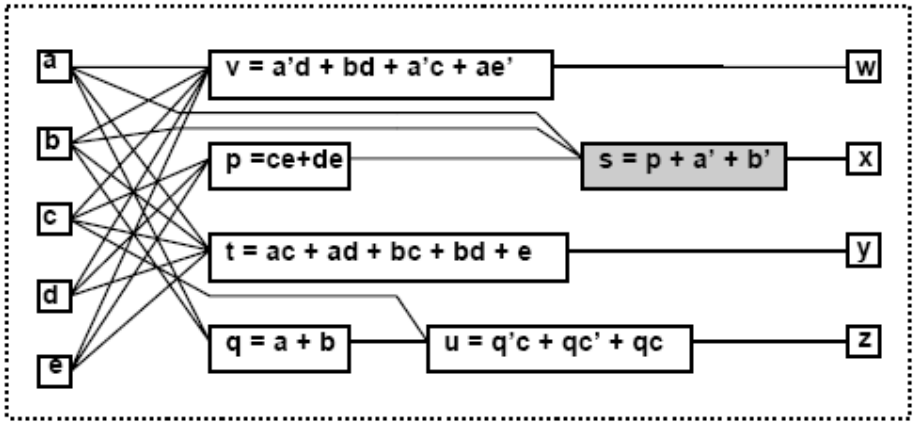
- Improve network step by step
  - Circuit transformations
- Preserve network I/O behavior
  - Exploit environment *don't cares* if desired
- Methods differ in:
  - Types of transformations applied
  - Selection and order of the transformations

Elimination

- Eliminate one function from the network
  - Similar to Gaussian elimination
- Perform variable substitution
- Example:
  - s = r + b’; r = p + a’;
  - s = p + a’ + b’;

Example

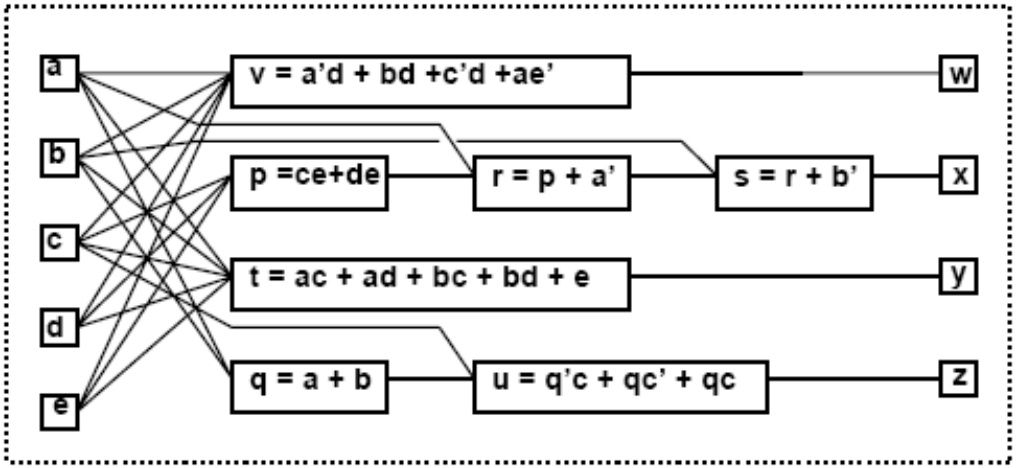

a
b
c
d
e
v = a'd + bd +c'd +ae'
p =ce+de
r = p + a'
s = r + b'
t = ac + ad + bc + bd + e
q = a + b
u = q'c + qc' + qc
w
x
y
z

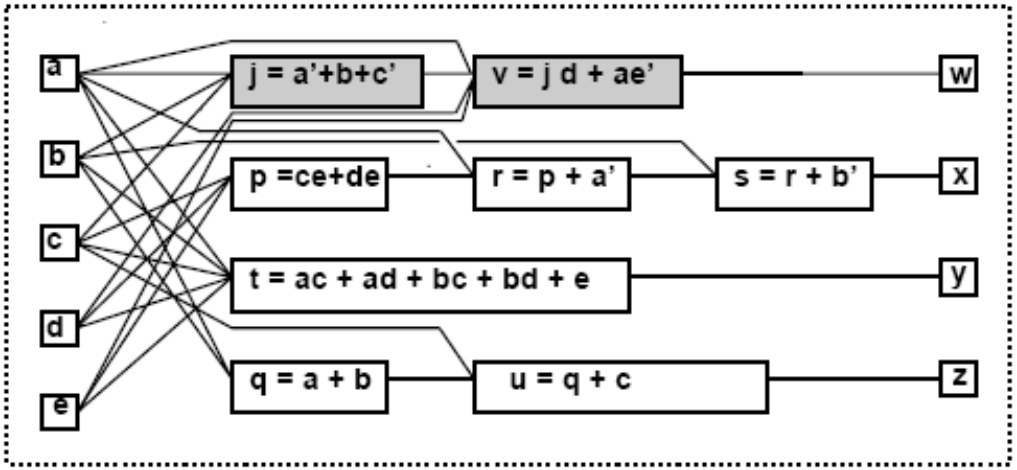

a
b
c
d
e
v = a'd + bd + a'c + ae'
p =ce+de
s = p + a' + b'
t = ac + ad + bc + bd + e
q = a + b
u = q'c + qc' + qc
w
x
y
z

Decomposition

- **Break a function into smaller ones**
  - **Opposite to elimination**
- **Introduce new variables/blocks into the network**
- **Example:**
  - **v = a’d + bd +c’d +ae’**
  - **j = a’ + b + c’; v = jd + ae’;**

Example

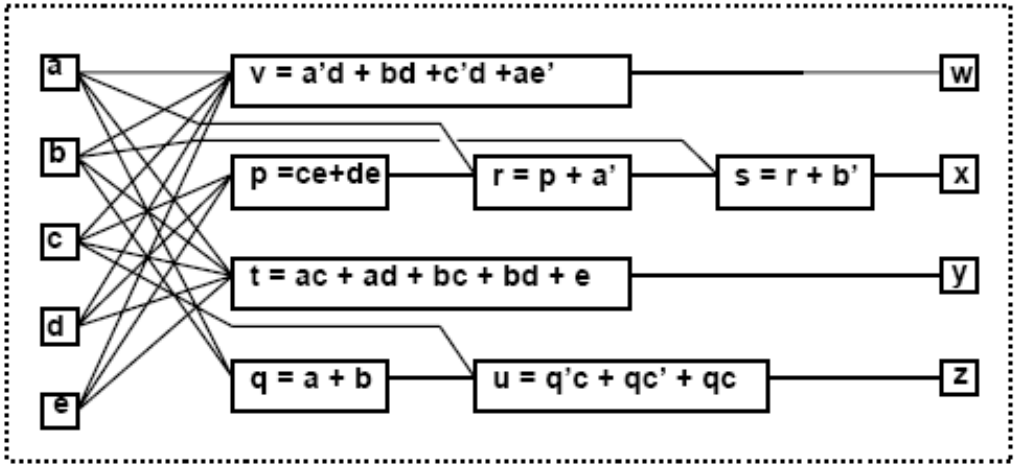

a
b
c
d
e
v = a'd + bd +c'd +ae'
p =ce+de
r = p + a'
s = r + b'
t = ac + ad + bc + bd + e
q = a + b
u = q'c + qc' + qc
w
x
y
z

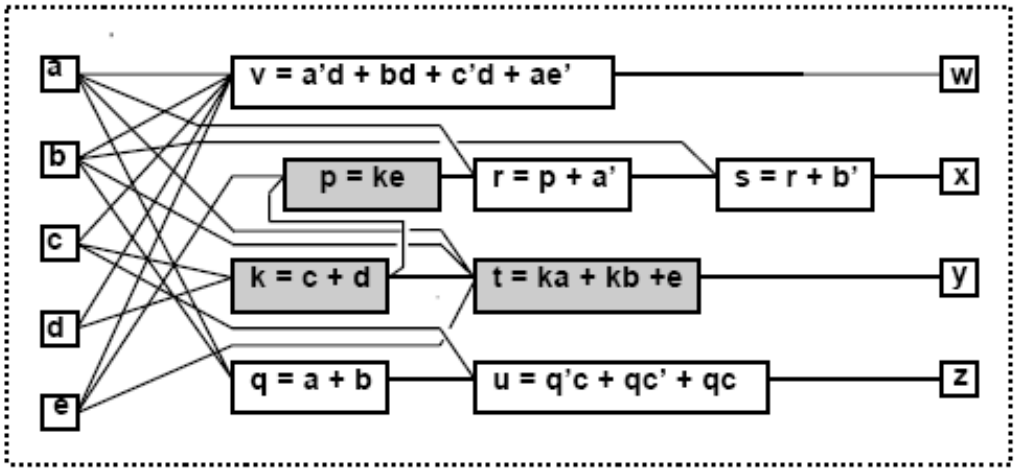

a
b
c
d
e
j = a'+b+c'
v = j d + ae'
p =ce+de
r = p + a'
s = r + b'
t = ac + ad + bc + bd + e
q = a + b
u = q + c
w
x
y
z

Extraction

- Find a common sub-expression of two (or more) expressions
  - Extract new sub-expression as new function
  - Introduce new block into the circuit
- Example
  - p = ce + de; t = ac + ad + bc + bd + e;
  - p = (c + d) e; t = (c + d) (a + b) + e;
  - k = c + d; p = ke; t = ka + kb + e;

Example

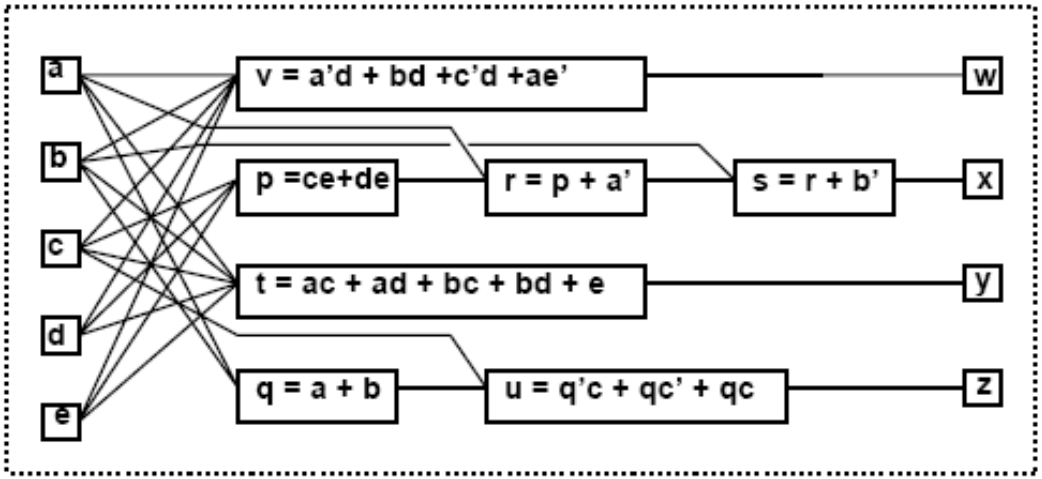

a
b
c
d
e
v = a'd + bd +c'd +ae'
p =ce+de
r = p + a'
s = r + b'
t = ac + ad + bc + bd + e
q = a + b
u = q'c + qc' + qc
w
x
y
z

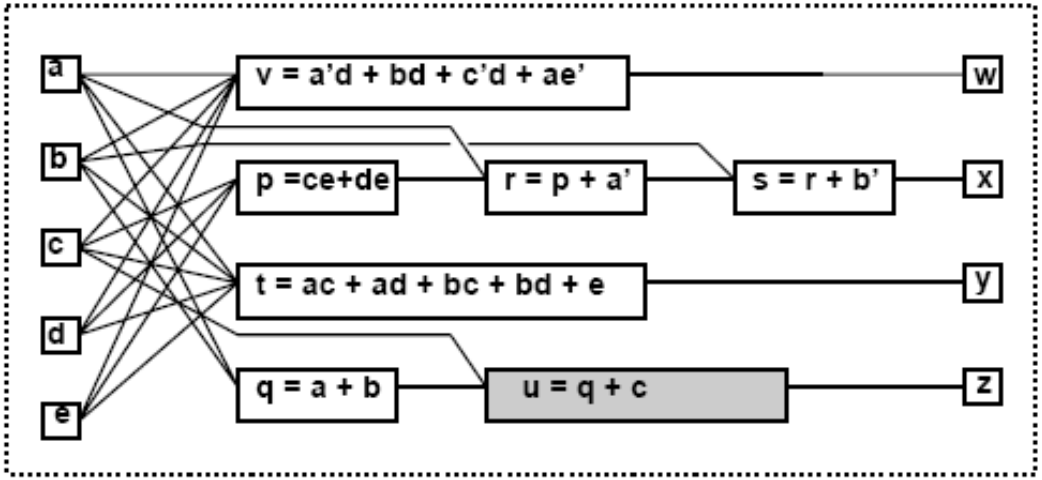

a
b
c
d
e
v = a'd + bd + c'd + ae'
p = ke
r = p + a'
s = r + b'
k = c + d
t = ka + kb +e
q = a + b
u = q'c + qc' + qc
w
x
y
z

Simplification

- **Simplify local function**
  - **Use heuristic minimizer like Espresso**
  - **Modify fanin of target node**
- **Example:**
  - **u = q’c + qc’ + qc;**
  - **u = q + c;**

Example

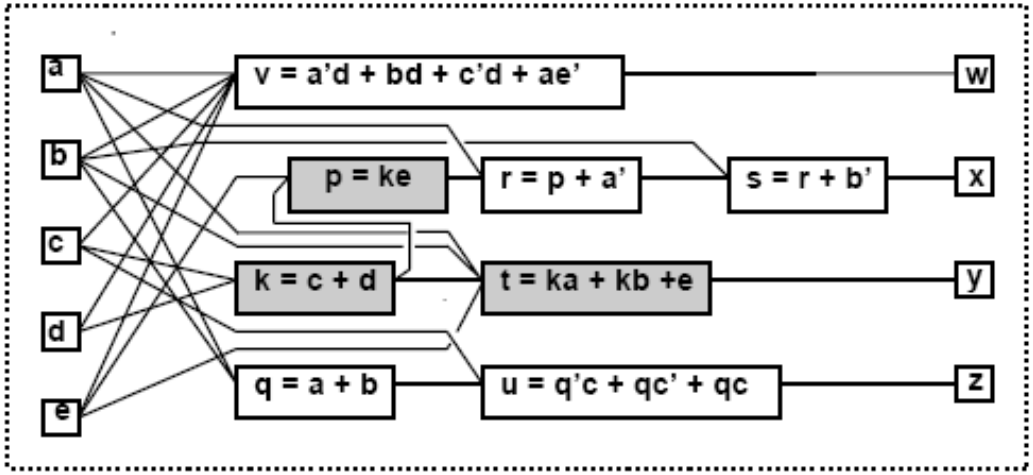

a
b
c
d
e
v = a'd + bd +c'd +ae'
p =ce+de
r = p + a'
s = r + b'
t = ac + ad + bc + bd + e
q = a + b
u = q'c + qc' + qc
w
x
y
z

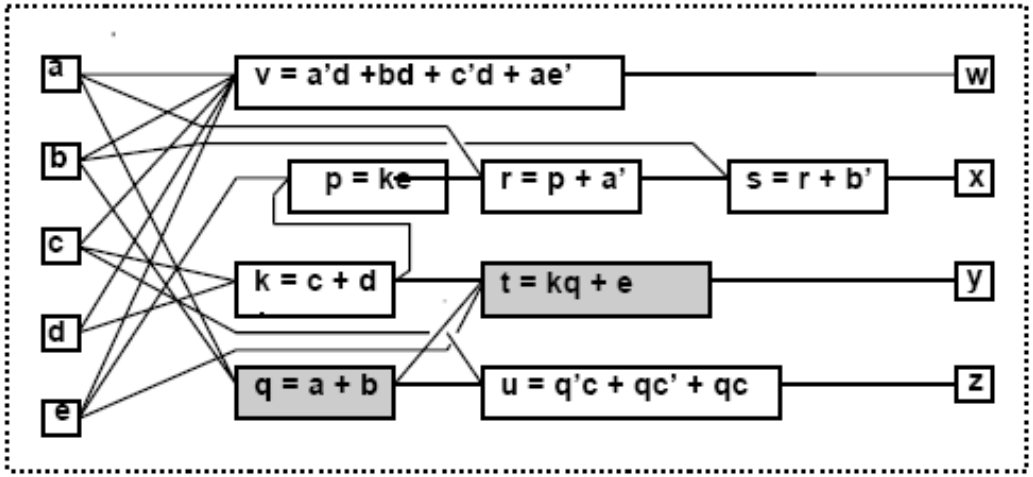

a
b
c
d
e
v = a'd + bd + c'd + ae'
p =ce+de
r = p + a'
s = r + b'
t = ac + ad + bc + bd + e
q = a + b
u = q + c
w
x
y
z

Substitution

- Simplify a local function by using an additional input that was not previously in its support set
- Example:
  - t = ka + kb + e;
  - t = kq + e;
  - Because q = a + b is already part of the network

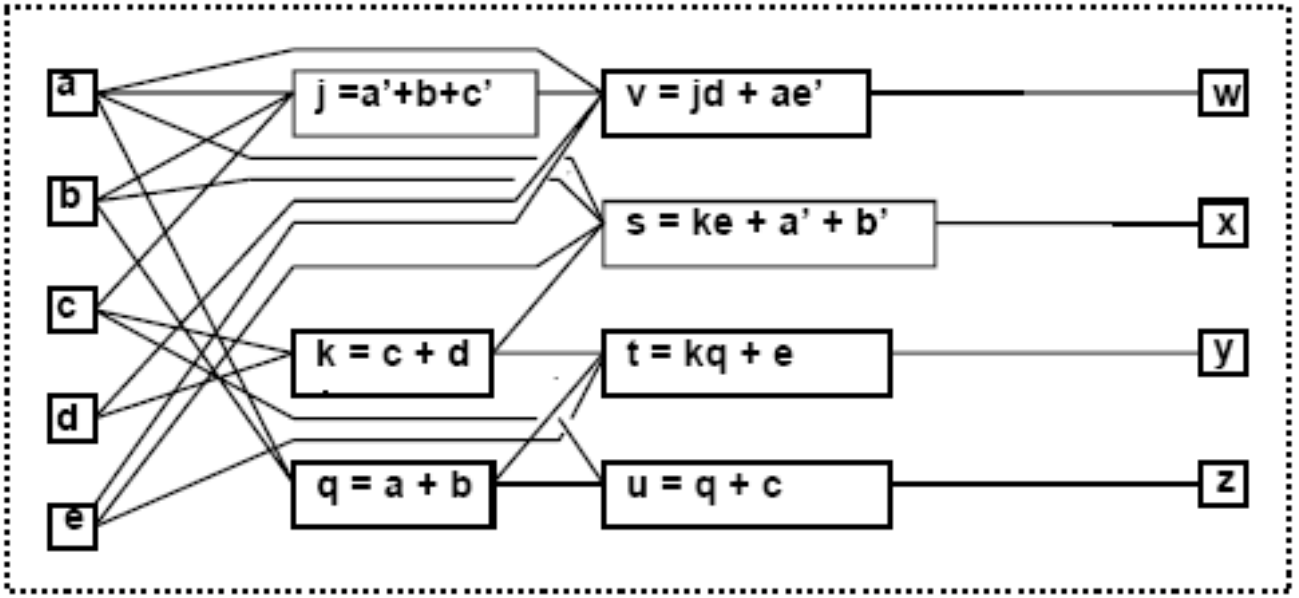
Example


a
b
c
d
e
v = a'd + bd + c'd + ae'
p = ke
r = p + a'
s = r + b'
k = c + d
t = ka + kb +e
q = a + b
u = q'c + qc' + qc
w
x
y
z


a
b
c
d
e
v = a'd +bd + c'd + ae'
p = ke
r = p + a'
s = r + b'
k = c + d
t = kq + e
q = a + b
u = q'c + qc' + qc
w
x
y
z

Example – Sequence of transformations

- j = a' + b + c
- k = c + d
- q = a + b
- s = ke + a' + b'
- t = kq + e
- u = q + c
- v = jd + ae'



Boolean and algebraic methods

- **Boolean methods for multilevel synthesis**
  - **Exploit properties of Boolean functions**
  - **Use *don't care* conditions**
  - **Computationally intensive**
- **Algebraic methods**
  - **Use polynomial abstraction of logic function**
  - **Simpler, faster, weaker**
  - **Widely used**

Example

- **Boolean substitution:**
  - Given: $h = a + bcd + e; \quad q = a + cd;$
  - Obtain: $h = a + bq + e;$
  - Because: $a + bq + e = a + b(a+cd) + e = a + bcd + e;$
- **Algebraic substitution:**
  - Given: $t = ka + kb + e; \; q = a + b;$
  - Obtain: $t = kq + e;$
  - Because: $kq = ka + kb;$

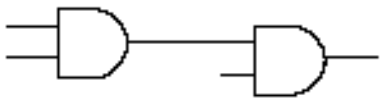
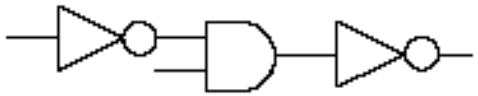
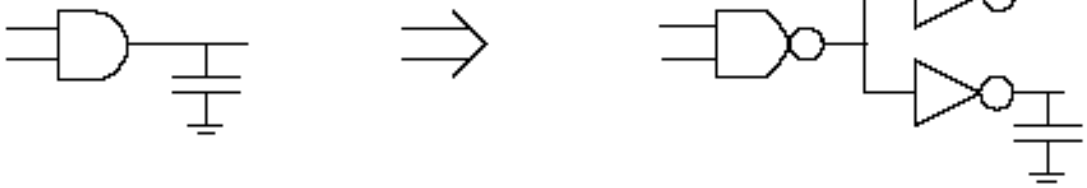
Algebraic vs Boolean methods

- Algebraic methods abstract functions as polynomials
  - Methods are fast and widely applicable
- Algebraic methods miss opportunities for optimization
  - As compared to Boolean methods
- Algebraic transformations are easily reversible
  - Ease transformations back and forward to trade off area and speed

Library binding

- Given an unbound logic network and a set of library cells
  - Transform into an interconnection of instances of library cells
  - Optimize delay
    - (under area or power constraints)
  - Optimize area
    - Under delay and/or power constraints
  - Optimize power
    - Under delay and/or area constraints
- Library binding is called also technology mapping
  - Redesigning circuits in different technologies

Major approaches

- **Rule-based systems**
  - **Generic, handle all types of cells and situations**
  - **Hard to obtain circuit with specific properties**
  - **Data base:**
    - **Set of pattern pairs**
    - **Local search: detect pattern, implement its best realization**
- **Hei**
  - **T**  **s**
  - **L** 
- **Mo** **hes:**
  - **F**  **.**

Summary
Logic Synthesis

- Refine and optimize a logic representation:
  - Technology independent transformation (algebraic and Boolean)
  - Combinational and sequential transformations
  - Library binding
- Logic mapped models are input to physical design tools
  - Placement and routing
  - Strong coupling between Logic Synthesis and Physical Design