Multi-level logic Synthesis

Dr. Chandan Karfa

CSE IIT Guwahati

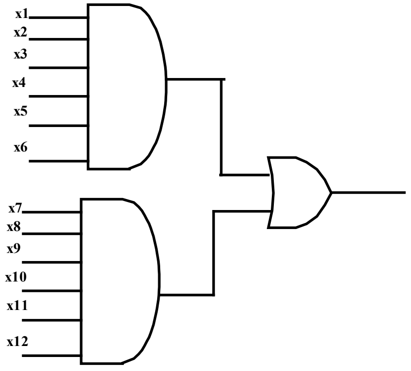
Text Book

- Chapter 6, Z. Kohavi and N. Jha, Switching and Finite Automata Theory, 3rd Ed., Cambridge University Press, 2010.

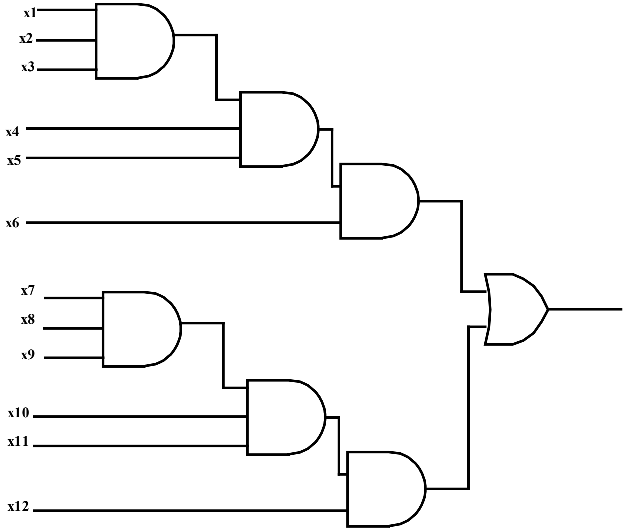
Objective

- Introduction to Multilevel logic synthesis
- Technology-independent synthesis
  - Factoring
  - Decomposition
  - Extraction
  - Substitution
  - Elimination

Two-level vs Multi-level Logic



(a) Two level Implementation



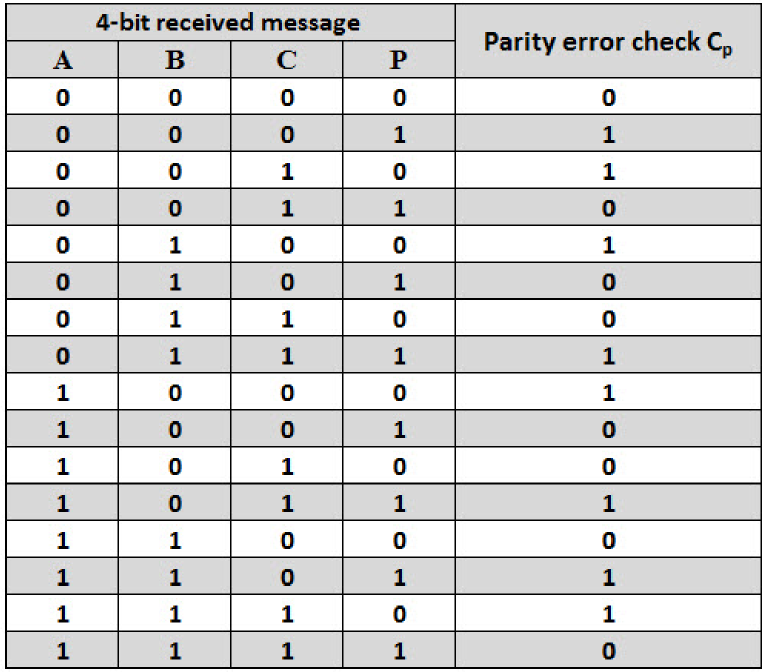
(a) Multi-level Implementation

- Multi-level logic realization contains more than two levels of logic gates.
- Circuits in which an arbitrary number of gates may lie on any path between a primary input and a primary output.

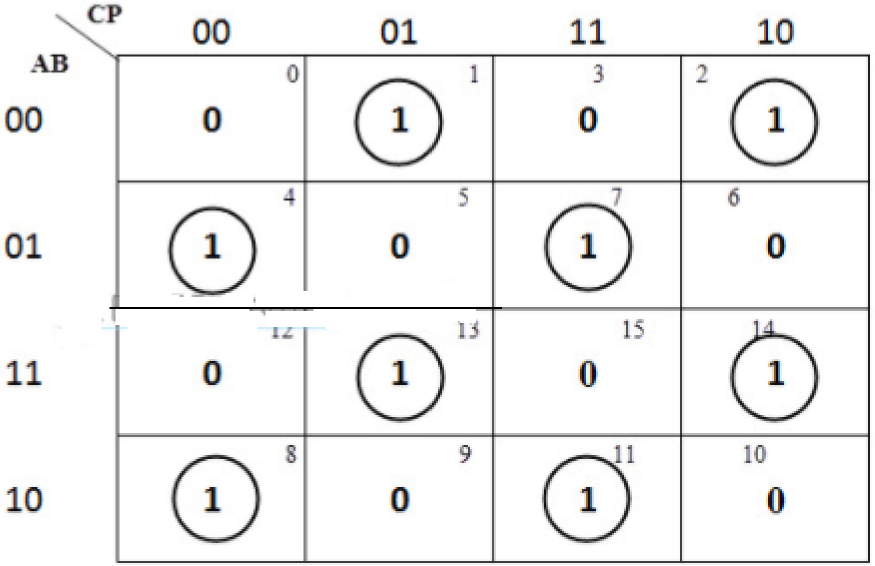
$xyz + wxz + yz$

Need for **Multi-level Logic Synthesis**

- In some cases SoP representation is impractical
  - Example: Parity bit generator, multiplication



4-bit received message				Parity error check C_p
A	B	C	P	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0



AB \ CP	00	01	11	10
00	0 0	1 1	3 0	2 1
01	4 1	5 0	7 1	6 0
11	12 0	13 1	15 0	14 1
10	8 1	9 0	11 1	10 0

Exponential number of implicants in SoP form for parity bit generator

Need for **Multi-level Logic Synthesis**

- **Advantage:**
  - Multi-level logic circuits require less area and delay compared to the corresponding two-level realizations and hence are more practical.
- **Disadvantage:**
  - It is difficult to obtain provably optimal multi-level realizations because of the much larger design space available for exploration.

Phases Of Multi-level Logic Synthesis

- Technology Independent Synthesis
- Technology Dependent Synthesis

Objectives

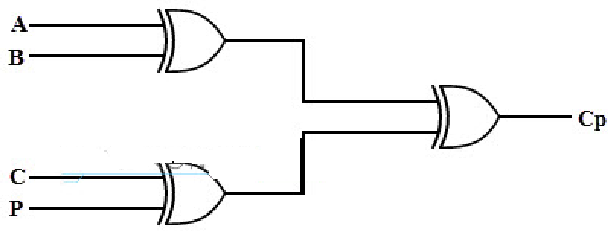
- Reduces the number of literal in expression
  - No of transistors is twice of no of literals
- Identifying common sub-expression within an Boolean Expression (for single output) or between multiple Boolean expressions (for multiple outputs)

$$PEC = \bar{A}\,\bar{B}\,(\bar{C}\,D + C\,\bar{D}) + \bar{A}\,B\,(\bar{C}\,\bar{D} + C\,D) + A\,B\,(\bar{C}\,D + C\,\bar{D}) + A\,\bar{B}\,(\bar{C}\,\bar{D} + C\,D)$$

$$= \bar{A}\,\bar{B}\,(C \oplus D) + \bar{A}\,B\,(\overline{C \oplus D}) + A\,B\,(C \oplus D) + A\,\bar{B}\,(\overline{C \oplus D})$$

$$= (\bar{A}\,\bar{B} + A\,B)\,(C \oplus D) + (\bar{A}\,B + A\,\bar{B})\,(\overline{C \oplus D})$$

$$= (A \oplus B) \oplus (C \oplus D)$$



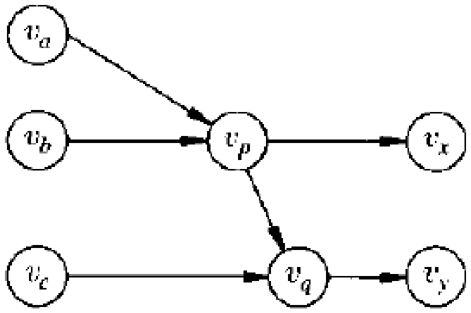
AB \ CP	00	01	11	10
00	0 0	1 1	3 0	2 1
01	4 1	5 0	7 1	6 0
11	12 0	13 1	15 0	14 1
10	8 1	9 0	11 1	10 0

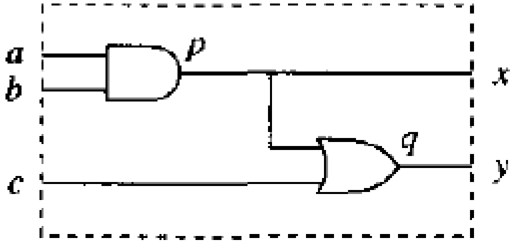
Representation

- As Logic Network

$$x = ab$$

$$y = c + ab$$

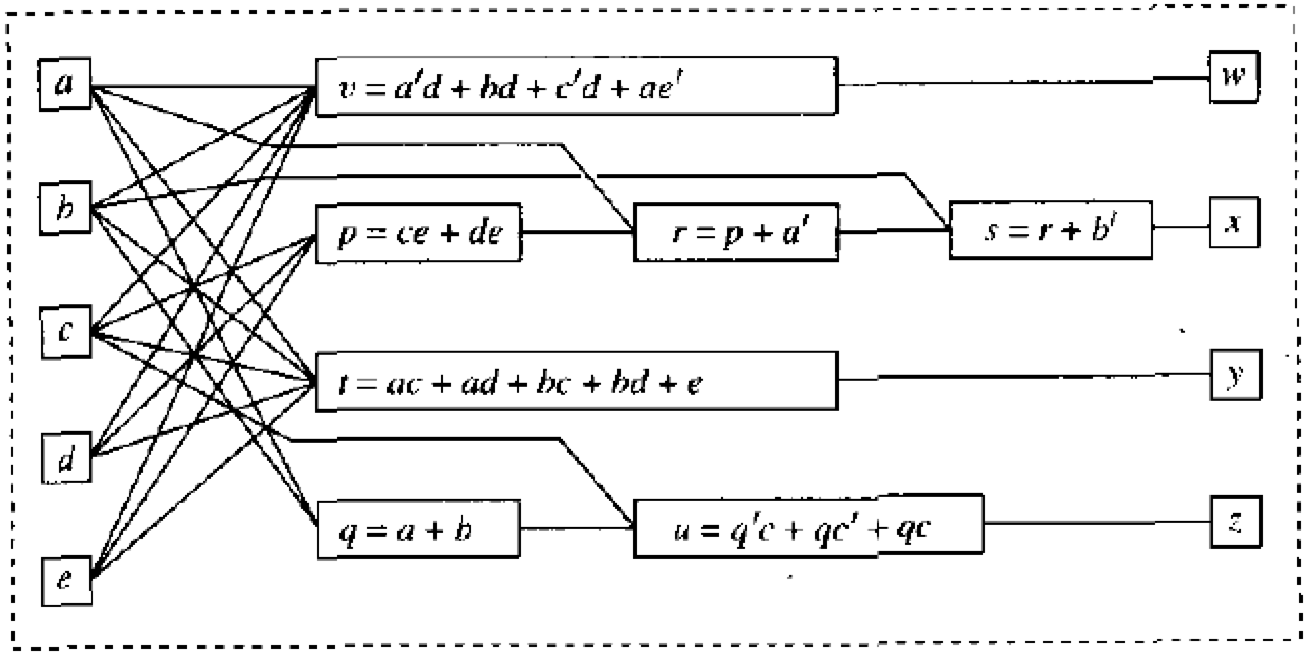




Logic Network

- input variables *(a,* b, c, *d, e)*
- primary output variables ***(w,*** *x* , ***y,*** *z).*

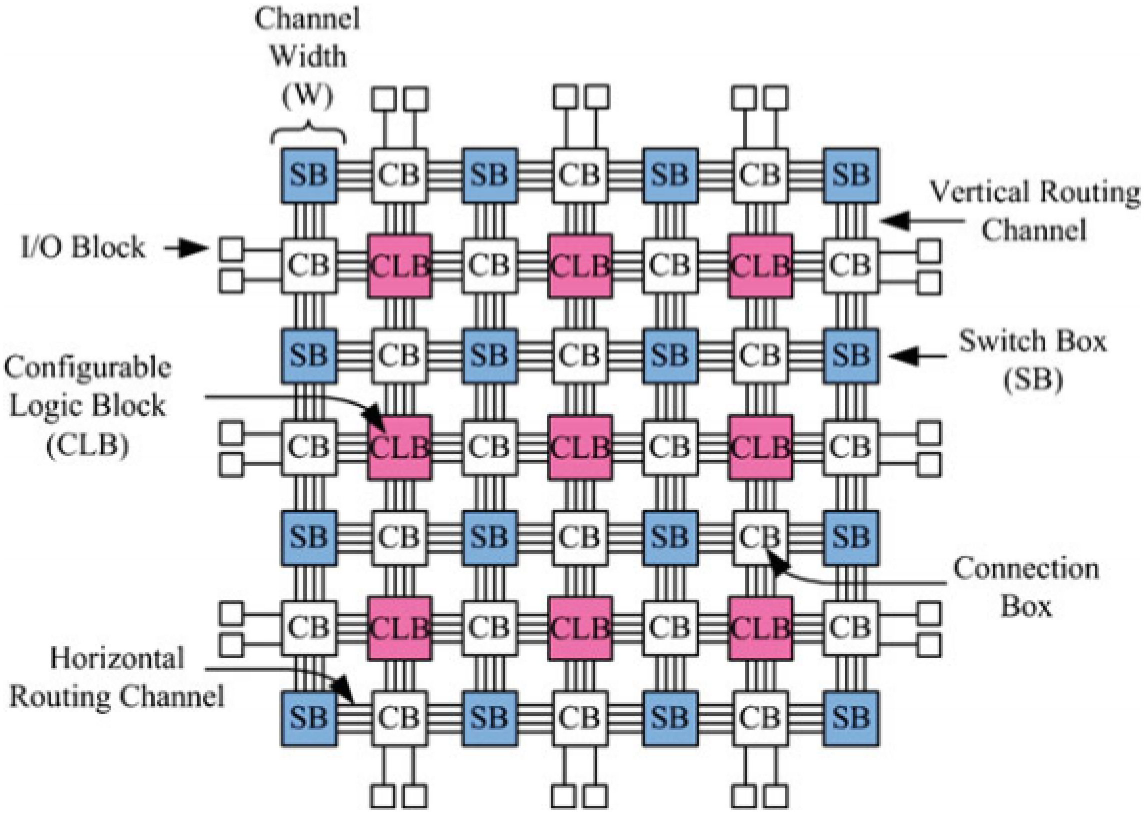
$$
\begin{aligned}
p &= ce + de \\
q &= a + b \\
r &= p + a' \\
s &= r + b' \\
t &= ac + ad + bc + bd + e \\
u &= q'c + qc' + qc \\
v &= a'd + bd + c'd + ae' \\
w &= v \\
x &= s \\
y &= t \\
z &= u
\end{aligned}
$$



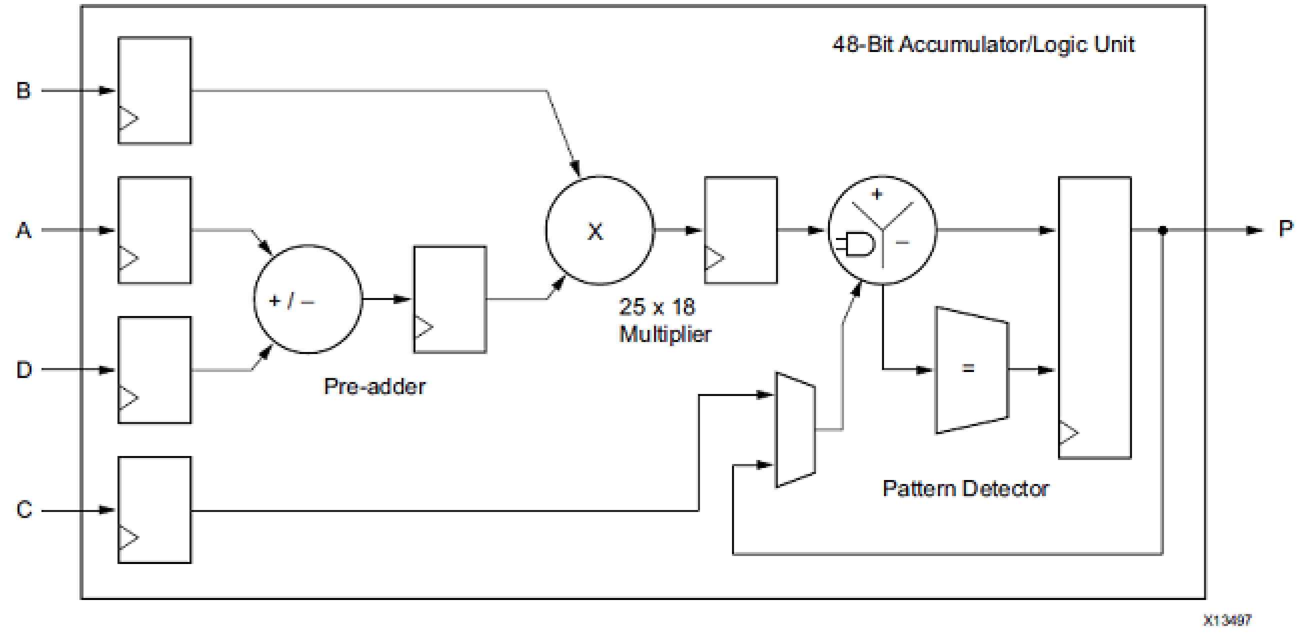
Technology Independent Synthesis

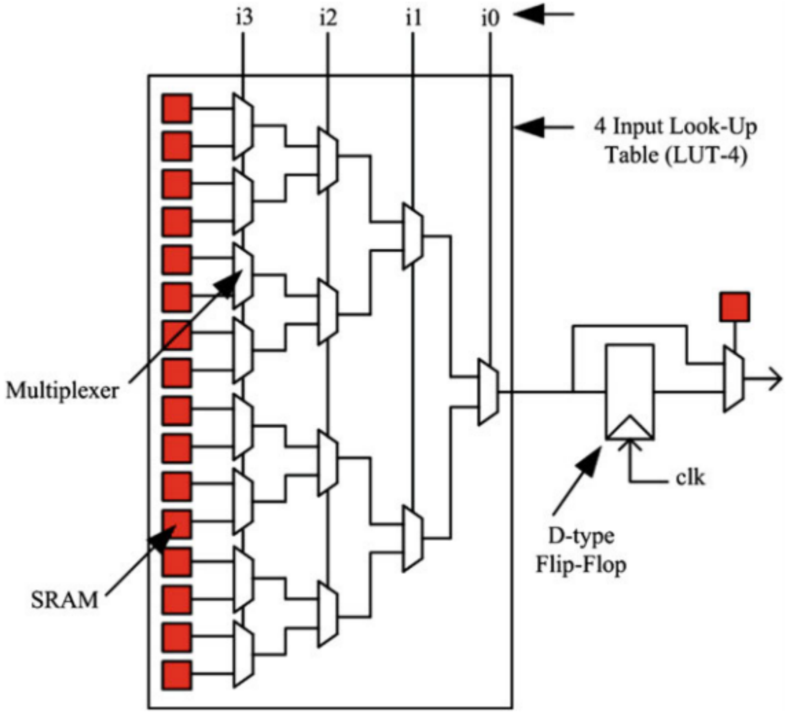
- We design the circuit without keeping in mind we have to map that circuit to certain technology.
- The problem is intractable.
  - Primarily the heuristic based approaches are used.
- Technology-independent multi-level logic synthesis is carried out with the help of various logic transformations.

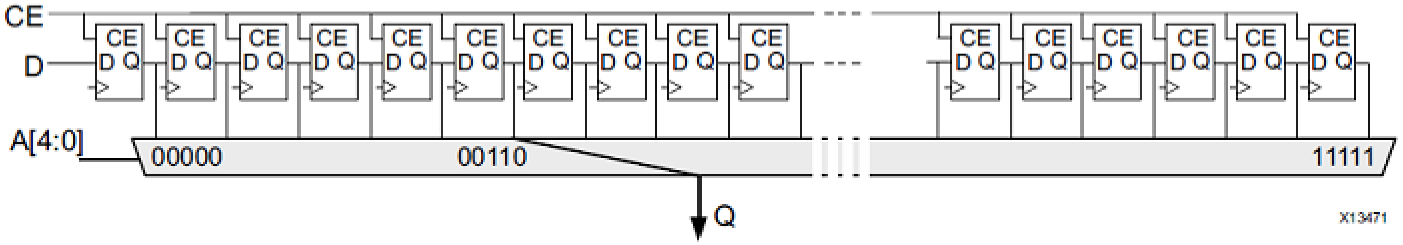
FPGA Technology Mapping



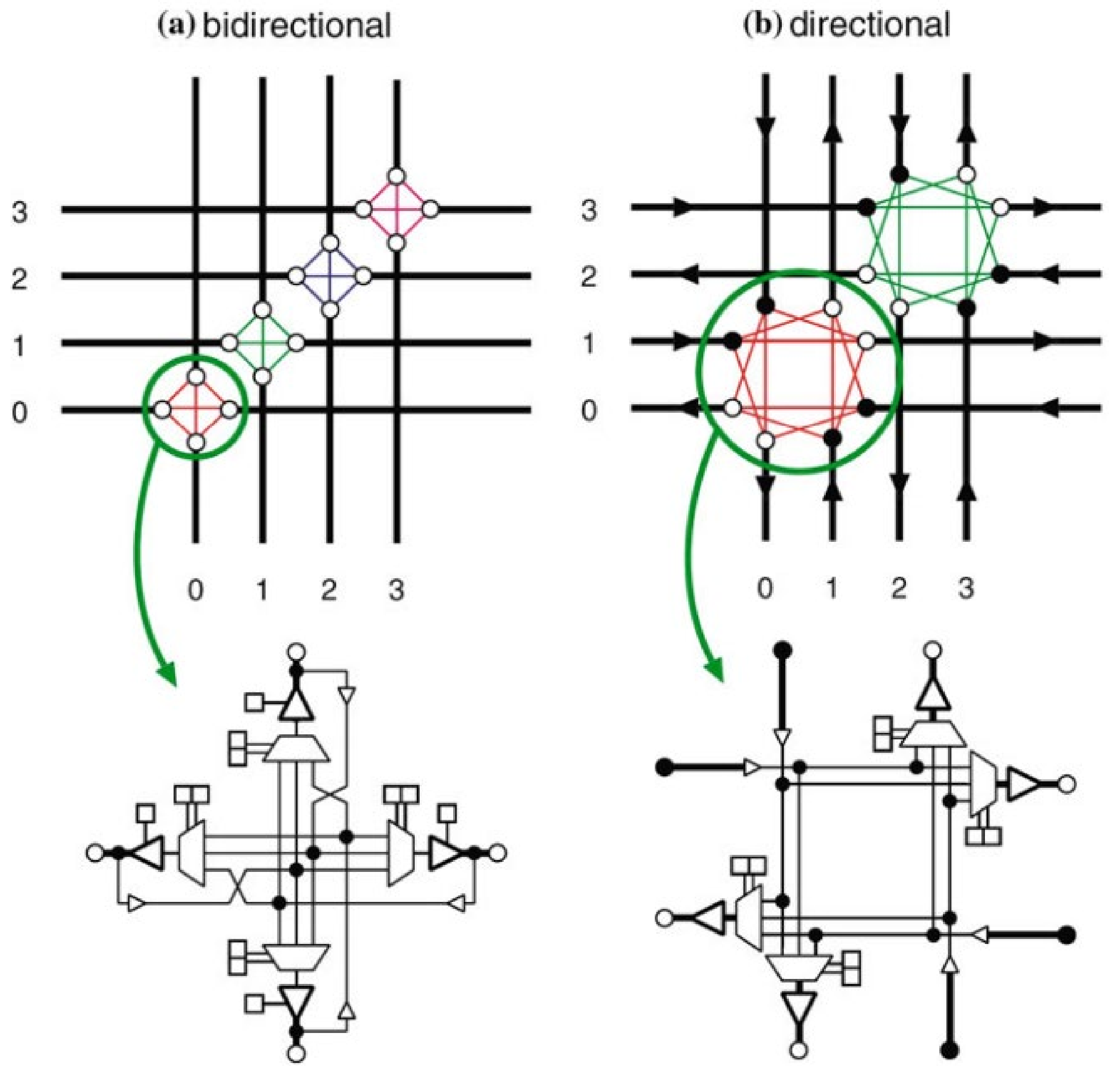
FPGA Components







Interconnections



FPGA Technology Mapping

- Technology Mapping:
- ASIC – Cell Library (e.g., NAND, NOT, FF)
- FPGA – fixed architecture
  - LUT
  - DSP
  - RAM/ROM
- In FPGA:
  - Gate level design ➔ LUTs
  - Multiplier/MAC ➔ DSP
  - Memory/Large Registers ➔ RAM/ROM

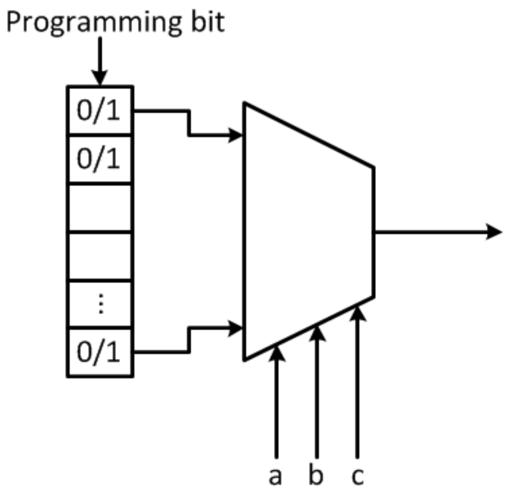
LUT Mapping

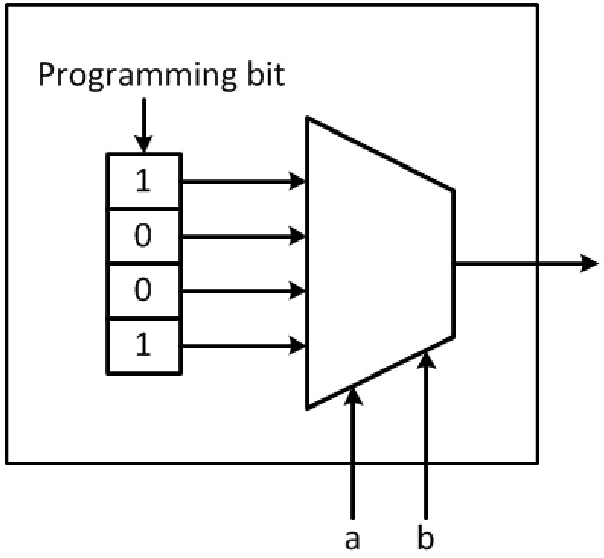
Look Up Table

2^k SRAM bits

A k-LUT can implement any Boolean function having k-variable,

B: 2^k → {0,1}



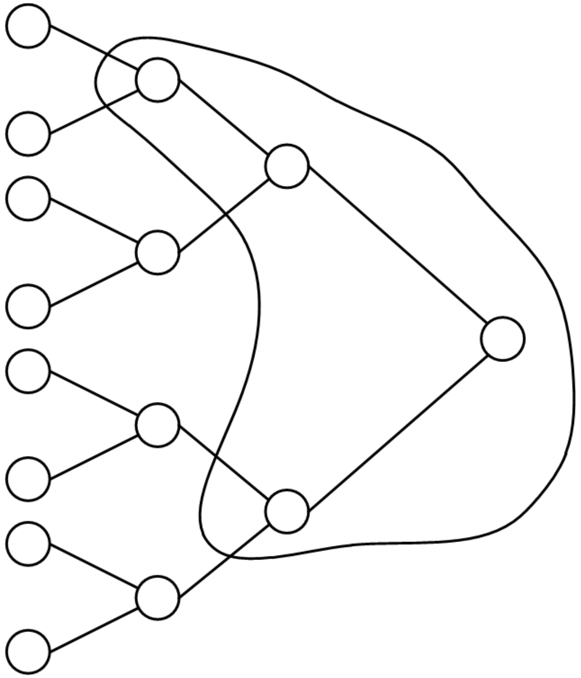


2 inputs LUTs

F = ab + a'b'

a	b	F
0	0	1
0	1	0
1	0	0
1	1	1

LUT Mapping

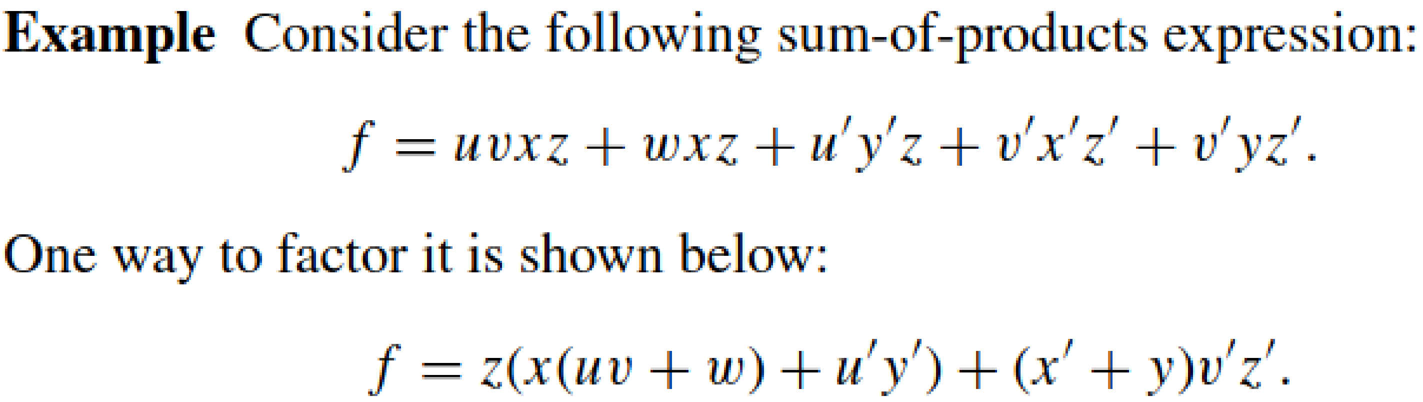


Introduction to logic transformations

- There are various methods of logic transformations. The methods are as follows:
  1. Factoring
  2. Decomposition
  3. Extraction
  4. Substitution
  5. Elimination.

Factoring

- A *factored form* is a recursive sum-of products representation in which the products themselves can consist of a sum of products
- In factoring, an expression in sum-of-products form is converted into an expression with multiple levels without introducing any sub-functions.



Example Consider the following sum-of-products expression:

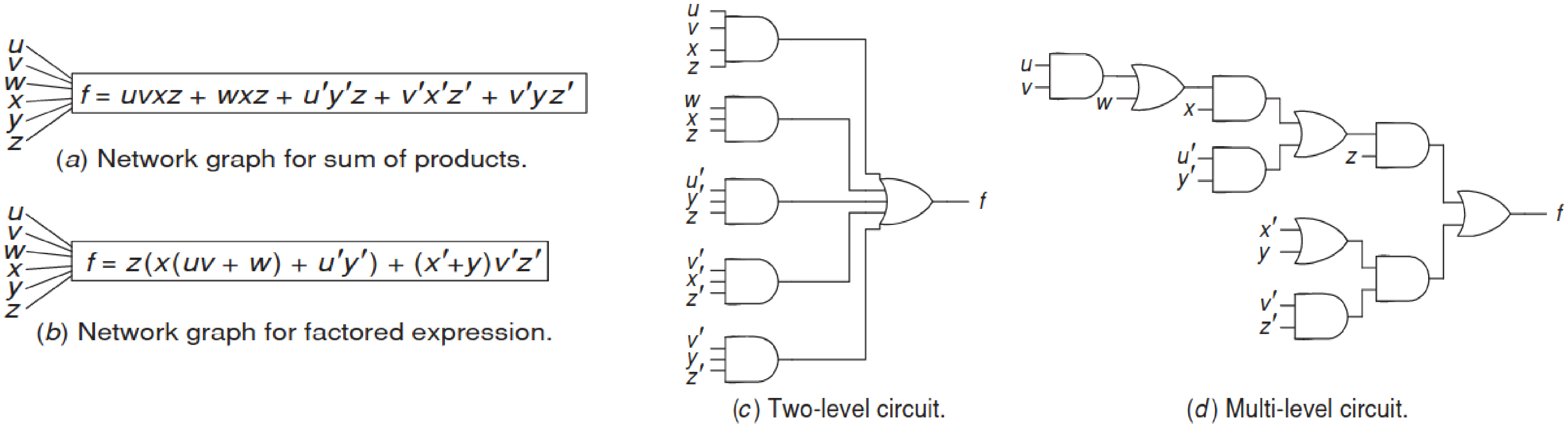
$$f = uvxz + wxz + u'y'z + v'x'z' + v'yz'.$$

One way to factor it is shown below:

$$f = z(x(uv + w) + u'y') + (x' + y)v'z'.$$

Factoring cont'd...

- The above expressions can be represented by logic network graphs
- the multi-level circuit has six levels of logic.


u v w x y z
f = uvxz + wxz + u′y′z + v′x′z′ + v′yz′
u v w x y z
f = z(x(uv + w) + u′y′) + (x′+y)v′z′
u v x z
w x z
u′ y′ z
v′ x′ z′
v′ y z′
f
u v w x z u′ y′ x′ y v′ z′
f

(*a*) Network graph for sum of products.

(*b*) Network graph for factored expression.

(*c*) Two-level circuit.

(*d*) Multi-level circuit.

Fig. 6.1 Network graphs and corresponding circuits.

Factoring cont'd...

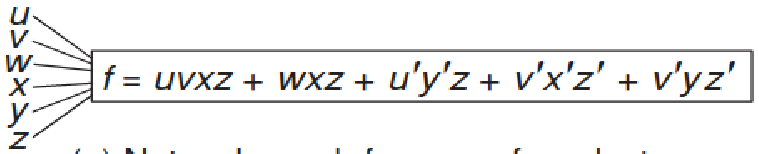
- Factoring makes expression more compact.
- The minimal sum-of-products expression has 16 literals whereas its factored form has only 11 literals.
- The number of transistors in the complex CMOS-gate implementation of an expression is twice its literal-count.
- As the number of literal in the expression increases, the implementation complexity of the expression also get increases.
- Literal count is good measure of implementation complexity.

Decomposition

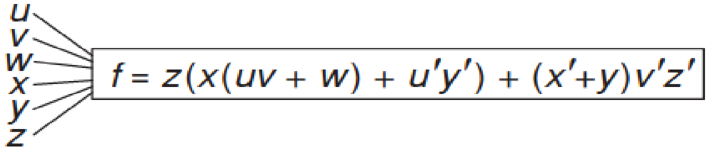
- In decomposition, a factored switching expression is replaced with a set of new expressions.

Example Consider the factored expression in Eq. (6.2). It can be decomposed as follows:

$$f_1 = uv + w, \qquad f_2 = x' + y,$$
$$f_3 = v'z', \qquad f_4 = xf_1 + u'y',$$
$$f = f_2 f_3 + z f_4.$$



(a) Network graph for sum of products.

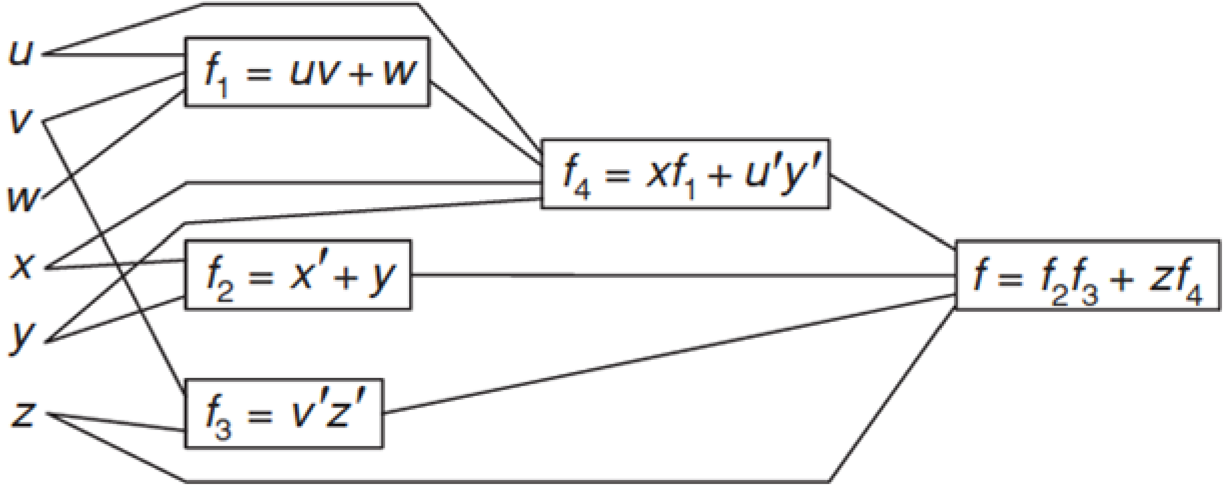


(b) Network graph for factored expression.

Decomposition cont'd…

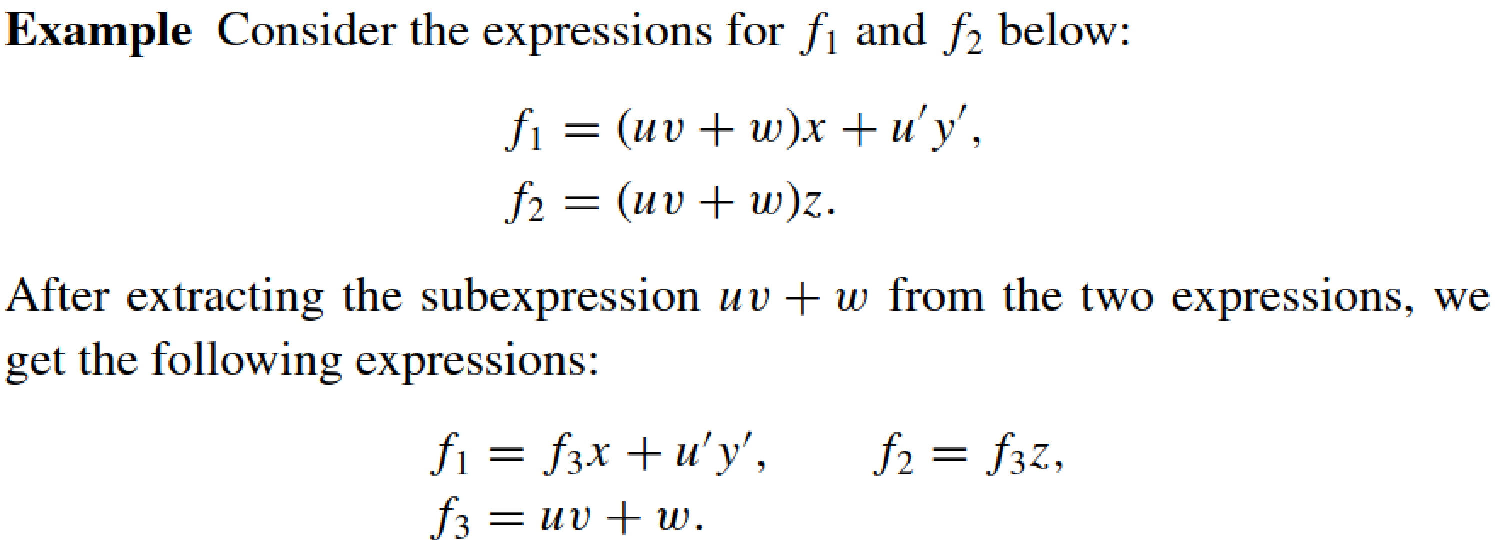
- Decomposition replaces subexpression in the expression by set of functions. These functions are said to be implemented separately.
- The literal-count after the decomposition is the sum of the literal-counts for each function. Thus, the literal count is now 15.

$$f_1 = uv + w, \qquad f_2 = x' + y,$$
$$f_3 = v'z', \qquad f_4 = xf_1 + u'y',$$
$$f = f_2 f_3 + z f_4.$$



Extraction

- Extraction is the process of extracting common sub-expressions (CSE) from two or more expressions in factored form.
- If there is CSE between two or more expressions, then we replace CSE by a new function.



Example Consider the expressions for f_1 and f_2 below:

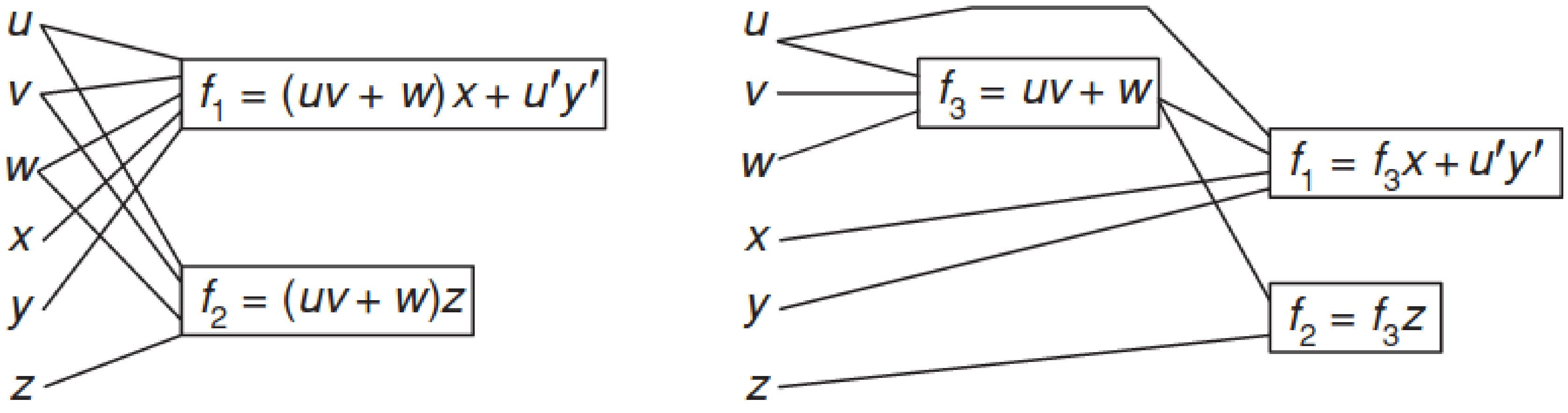
$$f_1 = (uv + w)x + u'y',$$
$$f_2 = (uv + w)z.$$

After extracting the subexpression $uv + w$ from the two expressions, we get the following expressions:

$$f_1 = f_3x + u'y', \qquad f_2 = f_3z,$$
$$f_3 = uv + w.$$

Extraction Cont'd...

- The literal count reduces from 10 to 9.



(*a*) Network graph before extraction.

(*b*) Network graph after extraction.

Fig. 6.3 Network graphs depicting extraction.

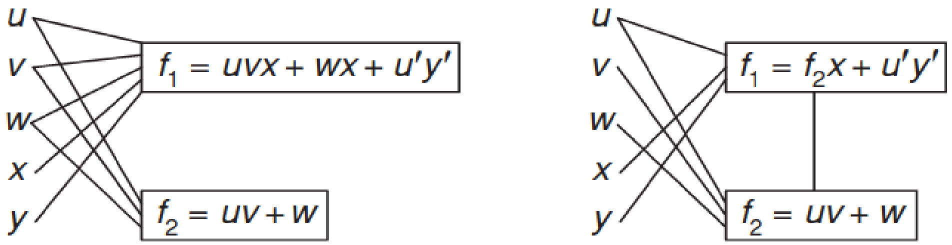
Substitution

- Substitution is the process of replacing a subexpression in an expression f with a variable g corresponding to a node in a network graph. In other words, g is substituted into for f is expressed in terms of g.

Example Consider f_1 and f_2 below:

$$f_1 = uvx + wx + u'y',$$
$$f_2 = uv + w.$$

The expression f_1 can be given in terms of f_2 as $f_2x + u'y'$. Thus, f_2 has been substituted into f_1. Figure 6.4 shows the network graphs for this example.



(*a*) Network graph before substitution.

(*b*) Network graph after substitution.

Fig. 6.4 Network graphs depicting substitution.

Elimination

- Elimination removes internal nodes from the network graph.
- It becomes possible if the corresponding expression replaces the variable corresponding to that node. Whenever the elimination step reduces the literal-count, it may be useful to employ it.

Example: Consider $f_1 = x + f_2$ and $f_2 = y + z$. If f_2 is not needed else-where in the network then it can be eliminated in the expression for f_1 by replacing it with $y + z$, thus obtaining $f_1 = x + y + z$. This reduces the literal-count from four to three.

Approach

- In multi-level logic synthesis, the above five logic transformations are applied to an initial logic network iteratively until no more improvement is possible in the targeted objective
    - They need not be applied in the given order
- Synthesis objectives are optimization of area, delay, or power.

Example

$p = ce + de$

$q = a + b$

$r = p + a'$

$s = r + b'$

$t = ac + ad + bc + bd + e$

$u = q'c + qc' + qc$

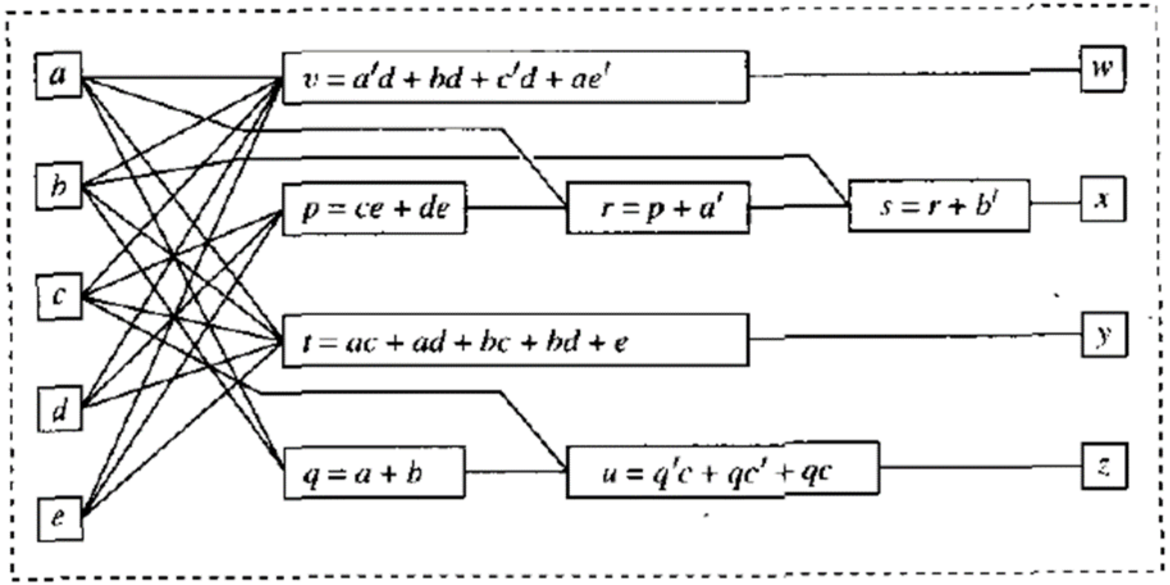
$v = a'd + bd + c'd + ae'$

$w = v$

$x = s$

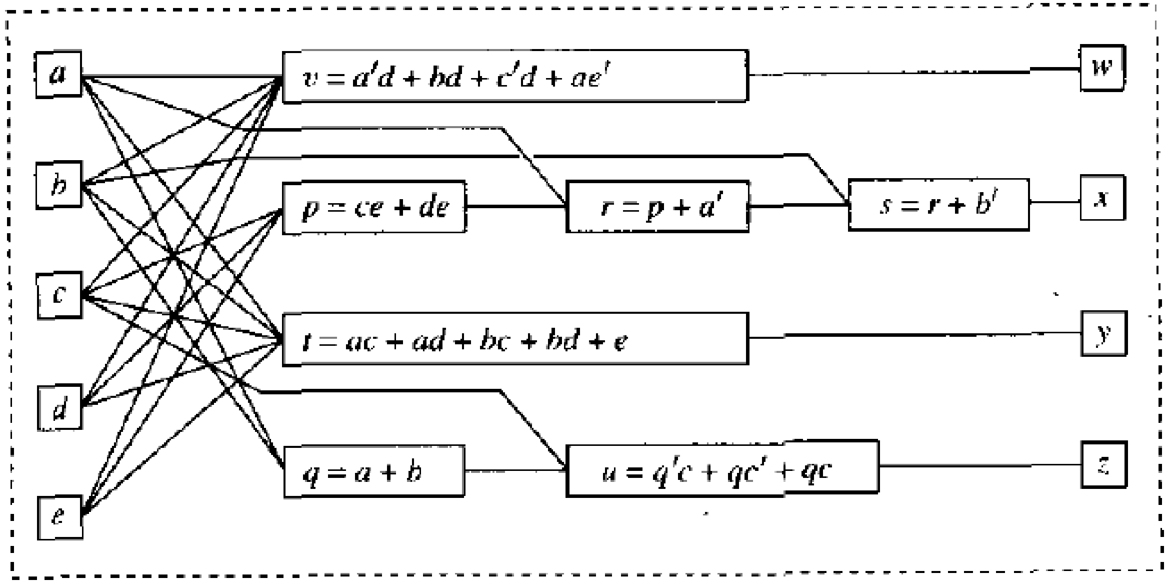
$y = t$

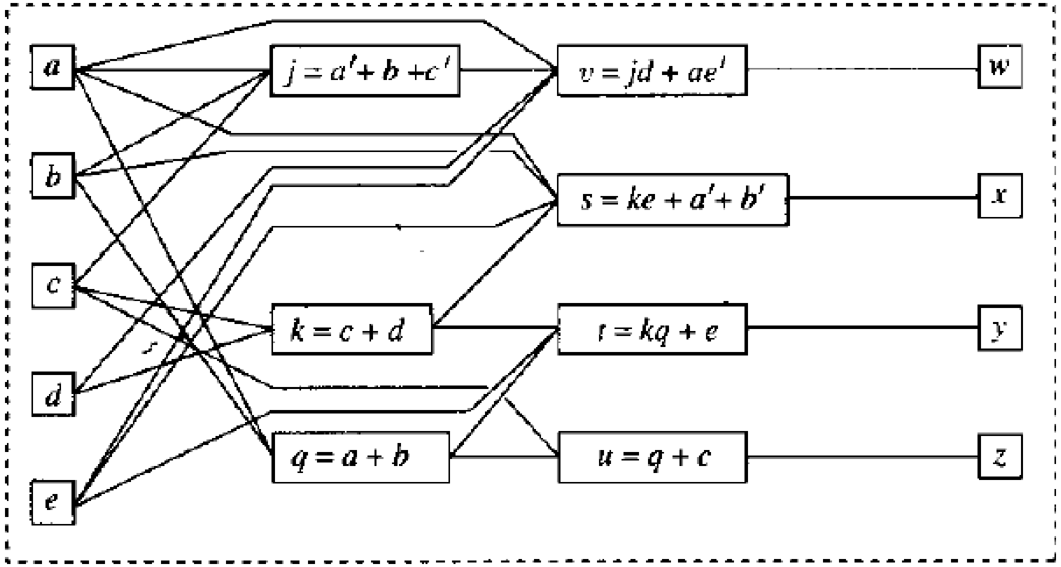
$z = u$



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

$$\begin{aligned}
j &= a' + b + c' \\
k &= c + d \\
q &= a + b \\
s &= ke + a' + b' \\
t &= kq + e \\
u &= q + c \\
v &= jd + ae' \\
w &= v \\
x &= s \\
y &= t \\
z &= u
\end{aligned}$$


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 = jd + ae'
s = ke + a' + b'
k = c + d
t = kq + e
q = a + b
u = q + c
w
x
y
z