ECE 697B (667)

Spring 2003

Synthesis and Verification of Digital Systems

Multi-level Minimization

- Factored forms

Slides adopted (with permission) from A. Kuehlmann, UC Berkeley 2003

Outline

- Factored forms
  - Definitions
  - Examples
- Manipulation of Boolean networks
  - Algebraic (structural) vs Boolean methods
    - Decomposition
    - Extraction
    - Factorization
    - Substitution (elimination)
    - Collapsing

Factored Forms

Example:

$$(ad+b'c)(c+d'(e+ac'))+(d+e)fg$$

Advantages

- good representative of logic complexity

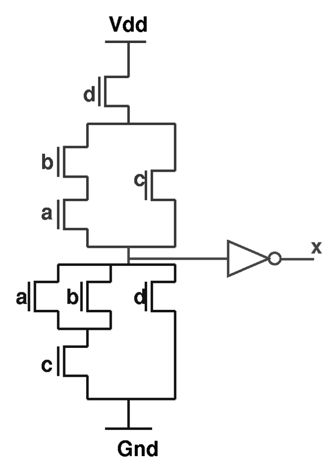
  $f=ad+ae+bd+be+cd+ce \quad f'=a'b'c'+d'e' \Rightarrow f=(a+b+c)(d+e)$
- in many designs (e.g. complex gate CMOS) the implementation of a function corresponds directly to its factored form
- good estimator of logic implementation complexity
- doesn't blow up easily

Disadvantages

- not as many algorithms available for manipulation
- hence often just convert into SOP before manipulation

Factored Forms

X=(a+b)c + d



Note:

literal count $\approx$ transistor count $\approx$ area

- however, area also depends on
  - wiring
  - gate size etc.
- therefore very crude measure

Factored Forms

Definition 1: *f* is an *algebraic expression* if *f* is a set of cubes (SOP), such that no single cube contains another (minimal with respect to single cube containment)

Example:

a+ab is not an algebraic expression (factoring gives *a(1+b)*)

Definition 2: The *product* of two expressions *f* and *g* is a set defined by $fg = \{cd \mid c \in f \text{ and } d \in g \text{ and } cd \neq 0\}$

Example: *(a+b)(c+d+a')=ac+ad+bc+bd+a'b*

Definition 3: *fg* is an *algebraic product* if *f* and *g* are algebraic expressions and have disjoint support (that is, they have no input variables in common)

Example: *(a+b)(c+d)=ac+ad+bc+bd* is an algebraic product

Factored Forms

Definition 4: a *factored form* can be defined recursively by the following rules. A factored form is either a product or sum where:

- a product is either a single literal or a product of factored forms
- a sum is either a single literal or a sum of factored forms

A factored form is a parenthesized algebraic expression.

In effect a factored form is a product of sums of products … or a sum of products of sums …

Any logic function can be represented by a factored form, and any factored form is a representation of some logic function.

Factored Form Examples

Examples of factored forms:

x

y'

abc'

a+b'c

((a'+b)cd+e)(a+b')+e'

(a+b)'c is not a factored form since complementation is not allowed, except on literals.

Three equivalent factored forms (factored forms are not unique):

ab+c(a+b) *bc+a(b+c)* *ac+b(a+c)*

Factored Forms

Definition 5: The factorization value of an algebraic factorization $F=G_1G_2+R$ is defined to be

$$fact\_val(F,G_2) = lits(F)-(lits(G_1)+lits(G_2)+lits(R))$$
$$= (|G_1|-1)\ lits(G_2) + (|G_2|-1)\ lits(G_1)$$

assuming G_1, G_2 and R are algebraic expressions. Where $|H|$ is the number of cubes in the SOP form of *H*.

Example: The algebraic expression

$$F = ae+af+ag+bce+bcf+bcg+bde+bdf+bdg$$

can be expressed in the form *F = (a+b(c+d))(e+f+g)*, which requires 7 literals, rather than 24.

If $G_1=(a+bc+bd)$ and $G_2=(e+f+g)$, then $R=\emptyset$.

$$fact\_val(F,G_2) = 2\times 3+2\times 5=16.$$

The factored form above saves 17 literals, not 16. The extra literal comes from recursively applying the formula to the factored form of G_1.

Factored Forms

Factored forms are more compact representations of logic functions than the traditional sum of products form.

Example: factored form

(a+b)(c+d(e+f(g+h+i+j)

while its SOP represented is:

ac+ade+adfg+adfh+adfi+adfj+bc+bde+bdfg+ bdfh+bdfi+bdfj

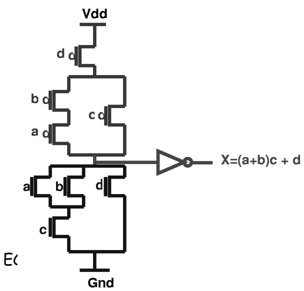
Of course, every SOP is a factored form but it may not be a good factorization.

Factored Forms

When measured in terms of number of inputs, there are functions whose size is exponential in sum of products representation, but polynomial in factored form.

Example: Achilles' heel function $\boxed{\prod_{i=1}^{i=n/2} (x_{2i-1} + x_{2i})}$

There are *n* literals in the factored form and $(n/2)\times 2^{n/2}$ literals in the SOP form.



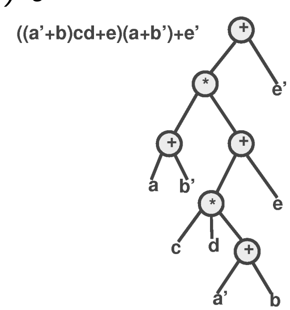
Factored forms are useful in estimating area and delay in a multi-level synthesis and optimization system.

In many design styles (e.g. complex gate CMOS design) the implementation of a function corresponds directly to its factored form.

Factored Forms

Factored forms cam be graphically represented as labeled *trees*, called *factoring trees*, in which each internal node including the root is labeled with either + or ×, and each leaf has a label of either a variable or its complement.

Example: factoring tree of *((a'+b)cd+e)(a+b')+e'*



Factored Forms

Definition: The size of a factored form *F* (denoted ρ(*F*)) is the number of literals in the factored form.

Example: ρ((*a+b)ca') = 4* ρ(*(a+b+cd)(a'+b')) = 6*

A factored form is *optimal* if no other factored form (for that function) has fewer literals.

A factored form is *positive unate* in *x*, if *x* appears in *F*, but *x'* does not.
A factored form is *negative unate* in *x*, if *x'* appears in *F*, but *x* does not.

F is unate in *x* if it is either positive or negative unate in *x*, otherwise *F* is binate in *x*.

Example:

(a+b')c+a' is positive unate in *c*, negative unate in *b*, and binate in *a*.

Manipulation of Boolean Networks

Basic Techniques:

- structural operations (change topology)
  - algebraic
  - Boolean
- node simplification (change node functions)
  - don't cares
  - node minimization

Structural Operations

Restructuring Problem: Given initial network, find best network.

Example: $f_1 = abcd+abce+ab'cd'+ab'c'd'+a'c+cdf+abc'd'e'+ab'c'df'$

$f_2 = bdg+b'dfg+b'd'g+bd'eg$

minimizing,

$f_1 = bcd+bce+b'd'+a'c+cdf+abc'd'e'+ab'c'df'$

$f_2 = bdg+dfg+b'd'g+d'eg$

factoring,

$f_1 = c(b(d+e)+b'(d'+f)+a')+ac'(bd'e'+b'df')$

$f_2 = g(d(b+f)+d'(b'+e))$

decomposing,

$f_1 = c(x+a')+ac'x'$

$f_2 = gx$

$x = d(b+f)+d'(b'+e)$

Two problems:

- find good common subfunctions
- effect the division

Structural Operations

Basic Operations:

1. *Decomposition* (single function)

$f = abc+abd+a'c'd'+b'c'd'$

$\Downarrow$

$f = xy+x'y', \quad x = ab, \quad y = c+d$

2. *Extraction* (multiple functions)

$f = (az+bz')cd+e \quad g = (az+bz')e' \quad h = cde$

$\Downarrow$

$f = xy+e, \quad g = xe', \quad h = ye, \quad x = az+bz', \quad y = cd$

3. *Factoring* (series-parallel decomposition)

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

$\Downarrow$

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

Structural Operations

4. *Substitution*

$g = a+b \quad f = ac+bc + d$

$\Downarrow$

$f = gc+d$

5. *Collapsing* (also called *elimination*)

$f = ga+g'b \quad g = c+d$

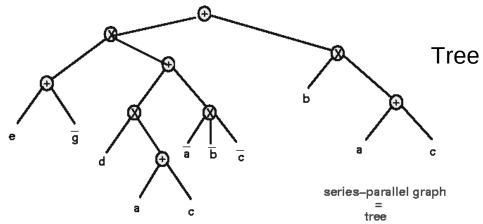
$\Downarrow$

$f = ac+ad+bc'd'$

Note: “division” plays a key role in all of these

Factoring vs. Decomposition

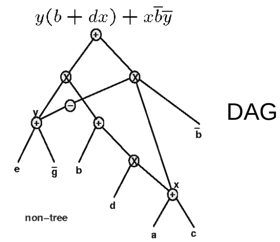
Factoring: *f=(e+g')(d(a+c)+a'b'c')+b(a+c)*



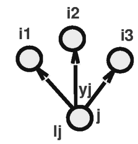
Decomposition: *y(b+dx)+xb'y',*

where: *x = a+c, y = e+g'*

Note:: this is similar to BDD collapsing of common nodes and using negative pointers. But not canonical, so don't have perfect identification of common nodes.



Value of a Node and Elimination



$$value(j) = \left(\sum_{i \in FO(j)} n_i \right) (l_j - 1) - l_j$$

where

n_i = number of times literals y_j and y_j' occur in factored form f_i

l_j = number of literals in factored f_j

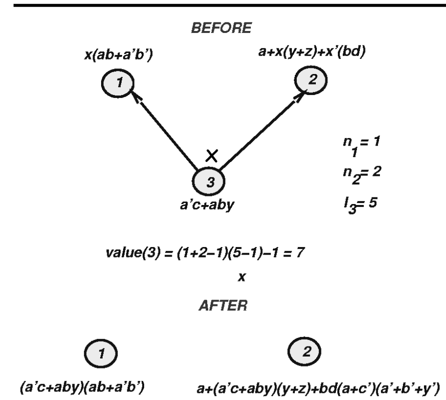
with factoring $l_j + \sum_{i \in FO(j)} n_i + c$

without factoring $l_j \sum_{i \in FO(j)} n_i + c$

- *value (gain)* = cost(without factoring) - cost(with factoring)

Can treat y_j and y_j' the same since $\rho(F_j) = \rho(F_j')$.

Value of a Node and Elimination



$$value(j) = \left(\sum_{i \in FO(j)} n_i \right) (l_j - 1) - l_j$$
$$= (n_1 + n_2)(l_3 - 1) - l_3$$
$$= (1+2)(5-1) - 5 = 7$$

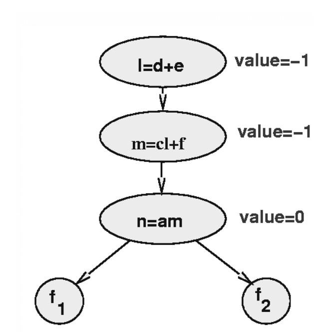
Literals before = 5+7+5 =	17
Literals after = 9+15 =	24

	7

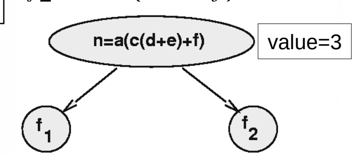
Difference *after - before* = *value* = 7

But we may not have the same value if we were to eliminate, simplify and then re-factor.

Value of a Node and Elimination



$$\begin{aligned} n &= a(c(d+e)+f) \\ f_1 &= b(n+ag)+h \\ f_2 &= i(n+aj)+k \end{aligned}$$



Note: value of a node can change during elimination

Optimum Factored Forms

- **Definition:**
  - Let f be a completely specified Boolean function, and $\rho(f)$ be the minimum number of literals in any factored form of f.
- **Definition:**
  - Let $sup(f)$ be the true variable support of f, i.e. the set of variables f depends on. Two functions f and g are orthogonal ($f \perp g$), if

$$sup(f) \cap sup(g)=\emptyset.$$

Optimum Factored Forms

Lemma: Let $f=g+h$ such that $g \perp h$, then $\rho(f)=\rho(g)+\rho(h)$.

Proof:

Let F, G and H be the optimum factored forms of f, g and h. Since $G+H$ is a factored form, $\rho(f)=\rho(F) \le \rho(G+H)=\rho(g)+\rho(h)$.

Let c be a minterm, on $sup(g)$, of g'. Since g and h have disjoint support, we have $f_c=(g+h)_c=g_c+h_c=0+h_c=h_c=h$.

Similarly, if d is a minterm of h', $f_d=g$.

Because $\rho(h)=\rho(f_c)\le\rho(F_c)$ and $\rho(g)=\rho(f_d)\le\rho(F_d)$, $\rho(h)+\rho(g)\le\rho(F_c)+\rho(F_d)$.

Let m (n) be the number of literals in F that are from $SUPPORT(g)$ ($SUPPORT(h)$). When computing F_c (F_d), we replace all the literals from $SUPPORT(g)$ ($SUPPORT(h)$) by the appropriate values and simplify the factored form by eliminating all the constants and possibly some literals from $sup(g)$ ($sup(h)$) by using the Boolean identities. Hence $\rho(F_c)\le n$ and $\rho(F_d) \le m$. Since $\rho(F)=m+n$,

$$\rho(F_c)+\rho(F_d)\le m+n=\rho(F).$$

We have $\rho(f)\le\rho(g)+\rho(h) \le \rho(F_c)+\rho(F_d)\le\rho(F) \Rightarrow \rho(f)=\rho(F)$.

Optimum Factored Forms

Note, the previous result does not imply that *all* minimum literal factored forms of f are sums of the minimum literal factored forms of g and h.

Corollary: Let $f=gh$ such that $g \perp h$, then $\rho(f)=\rho(g)+\rho(h)$.

Proof: Let F' denote the factored form obtained using DeMorgan's law. Then $\rho(F)=\rho(F')$, and therefore $\rho(f)=\rho(f')$. From the above lemma, we have

$$\rho(f)=\rho(f')=\rho(g'+h')=\rho(g')+\rho(h')=\rho(g)+\rho(h).$$

Theorem: Let $f=\sum_{i=1}^{n}\prod_{j=1}^{m} f_{ij}$ such that $f_{ij} \perp f_{kl}$, $\forall$ $i\neq j$ or $k\neq l$,

then $\rho(f)=\sum_{i=1}^{n}\prod_{j=1}^{m}\rho(f_{ij})$

Proof:

Use induction on m and then n, and lemma 1 and corollary 1.