Combination Logic optimization.

- Power optimization can be achieved at system, architectural (behavioral), register-transfer, logic level, transistor, and physical design, circuit level.

- Logic synthesis design level.
  Automatic synthesis of gate-level netlists, minimizing some objective function subject to various constraints.
  E.g., goal: minimal area
  Subject to: delay requirement.

- Techniques used for logic synthesis

  ![Diagram]

  - Extraction
  - Re-substitution
  - Collapsing
  - Decomposition
  - Factorization

  Try to optimize several functions

  Try to optimize a single function
Look for expressions observed many times. Extract them and implemented only once. Then, they are shared.

E.g.,

\[ F = (a \cdot b) \cdot c \cdot d + e \]
\[ G = (a \cdot b) \cdot \bar{e} \quad \text{literal count} = 16 \]
\[ H = c \cdot d \cdot e \]
\[ I = (a \cdot b) \cdot c \cdot d \cdot e \]

We identify \( a \cdot b \), \( c \cdot d \) as common terms.

\[ x = a \cdot b \]
\[ y = c \cdot d \]
\[ F = x \cdot y + e \quad \text{literal count} = 14 \]
\[ G = x \cdot \bar{e} \]
\[ H = y \cdot e \]
\[ I = x \cdot y \cdot e \]

original ckt
New ckt:

* use literal count to estimate the ckt area.

Re-substitution decompositions: use the function of a node already in the network to implement the function of another node in the ckt.

\[ G = a + b \]

\[ F = (a + b) \cdot c + cd \]

\[ \Rightarrow F = G \cdot c + cd \]
Factorization: Identify logic sharing within the same function. Derive a factored form from a sum-of-product form.

\[ F = a \cdot c + a \cdot d + b \cdot c + b \cdot d + e \]

\[ \Rightarrow F = c(a \cdot b) + d(a \cdot b) + e \]

\[ = (a \cdot b)(c + d) + e \]

Literal count = 9

Literal count = 5

Decomposition

Reexpress a single function as a collection of new functions.

Two goals:
1. Smaller-sized expression are more likely to be substituted by other functions
2. Reduce the size of expression to that of library cells

E.g., if too many 'or' then still need to decompose!!!

P.278 of De Michell.

\[ x = a \cdot b \]

\[ y = c \cdot d \]

\[ F = x \cdot y + x \cdot \overline{y} \]

Collapsing

Example: The collapsing of an internal vertex is its removal from the network. (increase speed)

A collapsing of \( V_r \) \( \Rightarrow V_s = V_r + b' \)

Motivation: Try to remove a simple local function

E.g. \( V_r \) does not have fanout to other nodes or \( V_s \) is not a PI.
Technology independent power optimization

- Boolean equation

- Technology independent logic synthesis

- Concentrate on this!!

- Netlist

- Technology mapper

- Cell library

- Special technology or technology dependent netlist

* Includes algebraic logic restructuring techniques.

- Boolean optimization techniques.

* Power optimization by common sub-expression extraction.

Given a boolean network, find a set of nodes to introduce in the network such that the power cost of the network in the SOP form is minimized.
Example:

\[ F_1 = a \odot y + a \odot u + u z \]
\[ F_2 = b \odot y + b \odot u + u z \]

with the following switch probabilities:

\[ p(a) = 0.97 \quad p(w) = 0.91 \quad p(x) = 0.67 \]
\[ p(b) = 0.02 \quad p(x) = 0.93 \quad p(y) = 0.47 \]
\[ p(c) = 0.51 \quad p(z) = 0.35 \quad p(z) = 0.65 \]

How to rearrange the boolean equations s.t. the power consumption can be minimized.

Ref: Two functions are orthogonal if they do not have common literals.

Eg, \( f = a \cdot b \), \( g = c \cdot d \)

\( f \) and \( g \) are orthogonal. \(( g \perp f )\)

Ref: Function \( g \) is an algebraic divisor of \( f \) if \( \exists h, r \)

s.t. \( f = g \cdot h + r \) where \( h \neq \phi \) and \( g \neq h \).

Example:

\[ f = ab^c + abd + de \]
\[ g = a \cdot b + e \]

\[ \frac{f}{g} = d + abc \]

\( h \leftarrow r \)

\[ \frac{abc}{ab} = c, \quad \frac{abd}{ab} = d, \quad \frac{de}{ab} = \phi \]

\( \vdash \ h_1 = \{ c, d \} \)

\( \frac{abc}{e} = \phi, \quad \frac{abd}{e} = \phi, \quad \frac{de}{e} = d \)

\( \vdash \ h_2 = \{ d \} \)

\( \vdash \ h = d \).
The primary divisors of \( f \) is \( P(f) = \{ f/c | \text{c is a cube} \} \).

\( f = abc + abde \)

\( f/a = bc + bde \) is a primary divisor.

The kernels of \( f \) are \( K(f) = \{ k / k \in P(f), \text{the only cube dividing } k \text{ evenly is } 1 \} \).

Example:

\( f = a.b.c + a.b.d.e \)

\( f/a = bc + bde \) is a primary divisor, but not a kernel.

\( f/ab = (c + de) \) is a kernel and \( ab \) is called a co-kernel.

Example:

\( f = ab + cd \)

\( G = af + bf + ace + bce \)

\( G/a \Rightarrow ce + f \)

\( G/b \Rightarrow ce + f \)

\( G/c \Rightarrow ae + be \)

\( G/f \Rightarrow a + b \)

\( G/ce \Rightarrow a + b \)

\( (cdde)(ab) \to \text{Reduce the literal Count }!!! \)

\( 7 \to 5 \)
\[ H = ade + cde \]

**Kernel**

\[ H/de = a + c \]

**Co-Kernel**

\[ de \]

Kernel Extraction targeting Low Power

\[
\begin{array}{c}
\text{cubes} \\
\text{c}_{1}, \text{c}_{2}, \ldots, \text{c}_{N}
\end{array}
\]

\[
\begin{array}{c}
v_{1} \quad f_{1} \\
v_{2} \quad f_{2} \\
\vdots \\
v_{m} \quad f_{l}
\end{array}
\]

\[ D = d_{1} + d_{2} + \ldots + d_{p} \quad \leftarrow \text{kernel of } f_{1} - f_{l} \text{ used as a divisor.} \]

\[ Q = \{ q_{1}, q_{2}, \ldots, q_{r} \} \quad \leftarrow \text{co-kernels} \]

**Example:**

\[ F_{1} = axy + auw + uz \]

\[ F_{2} = bcxy + bcuw + uz \]

\[
\begin{array}{c}
D = \frac{xy + uw}{d_{1}, d_{2}} \\
q_{1} = a \\
q_{2} = bc
\end{array}
\]

\[ R = 2, \quad p = 2 \]

\[ D = xy + uw \quad F_{1} = ad + uz \]

\[ F_{2} = bcD + uz \]

\[ \# \text{of kernels} = 13 \]
The area saved by extracting D is

\[(R-1) \sum_{i=1}^{p} \text{lit}(d_i) + (P-1) \sum_{i=1}^{R} \text{lit}(\tilde{g}_i) - R\]

- Literals saved by not repeating the kernel
- Literals saved by not repeating the co-kernels
- Introduced by exactly kernel D

Example:

\[\left(2-1\right) \sum_{i=1}^{2} \text{lit}(d_i) \left(1 + \text{lit}(d_2)\right) + \left(2-1\right) \sum_{i=1}^{2} \text{lit}(\tilde{g}_i) \left(1 + \text{lit}(\tilde{g}_2)\right) - 2\]

\[= 4 + 3 - 2 = 5\]

\[\text{Saved}\]

\[\text{Saved}\]

\[D = \overline{a}y + \overline{a}u\overline{w}\]

\[\tilde{g}_1 = a, \tilde{g}_2 = bc\]

F1 = \[\overline{a}D\]

F2 = \[bcD\]

Both D's are newly added.

\[\text{Saved}\]

\[\text{Saved}\]
The power saved by extracting 0 is: if sum all capacitances are constant = unit, i.e., don't care it.

Equation (3) in page 274 of textbook.

\[
R-1 \quad \sum_{i=1}^{N} E(V_i) L S(V_i, D) \quad \text{Switching activity at node } W.
\]

\[
(2-1) \times \left( t(x) + t(y) + t(u) + t(w) \right) + \left[ \begin{array}{c}
\text{Switching activity \times Capacity} \\
\text{Effective capacitance}
\end{array} \right]
\]

x, y, u, w originally feeds 2 gates for each signal, so 1.

now only one to kernel \((xy + uw)\)

\[
(2-1) \times \left( t(a) + t(b) + t(c) \right) + 
\]

Each of signals a, b, c feeds 2 gates originally.

Now, only one to the co-kernel \((a, bc)\)

\[
\text{i.e., power saved by the reduction in the load on the inputs to the co-kernels of the given kernel.}
\]

\[
(axy) + t(auw) + t(bcy) + t(bcuw)
\]

Power saved by removing cubes from the original function.

\[
(\text{gates})
\]
\[ 2 \times t(D) - \]

\[
\begin{align*}
  & x \\
  & g \\
  & w \\
  & u \\
  & a \\
  & b \\
  & c \\
\end{align*}
\]

\[ D \quad t(D) \quad 2t(D) \quad F_1 \quad F_2 \]

\[
\begin{align*}
  & v_z \\
  & \alpha \\
  & \beta \\
  & v_z
\end{align*}
\]

Power consumption added to output of the new node \( D \) inserted.

\[
(t(xg) + t(uw)) -
\]

Power consumption at the output of the cubes of the new node inserted \( D \).

\[
(t(aD) + t(bCD))
\]

Power consumption at the output of the new cubes \( \Delta x, \beta \).
For the above example, with $P(x) = P(z)$, we can calculate all $t$ values.

Finally, the power value = 0.647

...the kernel extraction for D can same power.

Summary

$F_1 = \ldots$

$F_2 = \ldots$

$F_3 = \ldots$

$F_n = \ldots$

$K_1 = \{ \text{kernel} \}$

$K_2 = \{ \text{kernel} \}$

$k_n = \{ \text{kernel} \}$

1. Compute $K$: the set of all kernels for all functions

2. Generate $D$: the set of all kernel intersections for kernels in $K$.

3. Extract the sub-expression $D_i \in D$ which has the maximum power value.

4. Repeat 3 when there exists a sub-expression $D_i$ with a positive power value.
• Power optimization by substitution

• Substitution does not always guarantee a reduction in the power cost of the circuit.

• Example: $F_1 = abc$, $F_2 = ab$

\[ P(a) = P(b) = P(c) = 0.9, \quad P(F_2) = 0.5 \]

\[ P(F_2) \neq P(a) \times P(b) \text{ due to spatial dependence between } a \text{ and } b. \]

**Implementation 1:**

\[ F_1 = abc \]

\[ F_2 \]

\[ F_2 \]

\[ a \]

\[ b \]

\[ c \]

Loss in $F_1$:

\[ (0.5)(0.5) \times 2 + 0.9 \times 0.1 \times 2 \]

\[ = 0.5 + 0.18 \]

\[ = 0.68 \]

**Implementation 2:**

\[ F_1 = F_2 \cdot c \]

\[ F_2 \]

\[ C \]

For $F_2$: only input changes.

Input of $F_1$ has been clipped from

\[ (0.9)(0.1) \times 2 + (0.9 \times 0.1) \times 2 + \]

\[ \frac{0.9 \times 0.1 \times 2}{a} = 0.18 \times 3 = 0.54 \]
3.9.

Only 00 & 11 can be applied to i & j.

\[ P(\text{output} = 1) = P(i = 1, j = 1) \]
\[ = P(i = 1) \]
\[ = 0.9. \]

Instead of

\[ 0.9 \times 0.9 = 0.81 \]
• For literal count, we use one line (F2→F1) to replace 2 lines (a→F1, b→F1).

• For power cost, the gain in a→F1, b→F1 is less than the power loss in F1→F2.

  Conclusion

  * A node nj can be substituted into a node Ni?
  * The power cost of Ni is computed before and after substitution nj.
  * Substitution is performed if the power cost of Ni is reduced after substitution.

• Boolean optimization techniques

  • Take advantage of don't cares.

  • Show by two-level circuits
The power consumption of each product term:

\[ P(b_i) = \frac{V_{dd}^2 f}{2} \left( \text{swr} \cdot E(b_i) + \sum_{l_j \in \text{lit}(b_i)} \text{swr} \cdot E(l_j) \right) \]

Load seen by the inputs of AND,

Load seen by the output of the AND gate.

The power consumption of \( f \):

\[ P = \frac{V_{dd}^2 \cdot f}{2} \left( \text{or} \cdot E(f) + \sum_{b_i \in Q} P(b_i) \right) \]

Load seen by the output of the OR gate.
Problem: Given a boolean function \( f \) with input set \( V = (v_1, v_2, \ldots, v_n) \) and signal probability \( p(i) \) for each input, find a two-level implementation of the function \( f \) such that the power as given by the above equation is minimum.

- For minimizing area, only prime implicants need to be included.

\[
\begin{array}{ccc|cc}
ab & 00 & 01 & 11 & 10 \\
\hline
ca & 00 & 1 & 1 & 1 \\
01 & 1 & 1 & 1 & 1 \\
11 & 1 & 1 & 1 & 1 \\
10 & 1 & 1 & 1 & 1 \\
\end{array}
\]

(Any non-prime implicant \( g \) can be improved by making \( g \) prime.)

For minimum area: we use \( \overline{a}e + \overline{b}c \overline{d} + ac \overline{d} \)

For power, we may use: \( \overline{a}e + ab \overline{d} + ac \overline{d} \)

- Not the case for minimizing power consumption

Example:

\( p(a) = 0.9, \quad p(b) = p(c) = 0.5 \)

\( F_1 = ab + bc \quad \rightarrow \quad \text{for area, we use this} \)

\( F_2 = ab + \overline{a}bc \quad \rightarrow \)
F1 implementation: \( ab + bc \)

\[ P_1 = 0.25 \]
\[ P_0 = 0.75 \]
\[ 2P_1P_0 = 0.375 \]

\[ \text{This implementation} \]

\[ x = 1 \text{ if } abc = 011, 110, 111 \]

\[ P_0 = \overline{abc} \]
\[ = 0.1 \cdot 0.5 \cdot ab \]
\[ = 0.025 \]
\[ 2P_1P_0 = 2 \cdot 0.025 \cdot 0.975 \]
\[ = 0.049375 \]

\[ x = 1 \text{ if } abc = 011, 110, 111 \]

\[ \text{d in this side saves } 0.32625 \text{ of switching power.} \]

\[ a_1, a_9, 2, 2 = 0.36 \]

\[ \text{So, power increase about } 0.03375 \]

\[ \text{Should choose} \]

\[ \text{If output load of G is very large, then F2 implementation saves power.} \]

\[ (\text{e.g., G fanouts to several other gates}) \]
The reduction in power at output of node a offsets the increase in the power due to including literal a.

Def. An implicant $g_i$ is a power prime implicant (PP) if

$\forall g_j \quad g_i \subset g_j \Rightarrow \text{pow}(g_i) < \text{pow}(g_j)$.

This definition is very restrictive!!!
Example: use a very exasperating case

\[ p(a) = 0.01 \quad p(b) = p(c) = 0.5 \]

\[
[2 \cdot 0.5 \cdot 0.5] + \left[ 2 \left( \frac{1}{2} \right) \left( \frac{3}{4} \right) \right] = 0.5 + 0.375 + 0.5 = 1.375
\]

\[
2 \cdot (0.01 \cdot 0.99) + 2 \cdot (0.5 \cdot 0.5) + 2 \cdot (0.5 \cdot 0.5) + 2 \left[ \frac{0.01 \cdot 0.5 \cdot 0.5}{1 - 0.01 \cdot 0.5 \cdot 0.5} \right]
\]

\[
= 0.0198 + 1 + 2 \cdot (0.0025) \cdot (0.9975)
\]

\[
= 1.0198 + 0.005 = 1.025
\]

If \( Pr(abc) < \left[ \frac{Pr(abc)}{Pr(abc's \ predecessor \ pp2)} \right]. \)

\[
\Rightarrow \quad \text{abc is a ppi.}
\]

\[
2 \cdot (0.01 \cdot 0.99) + 2 \cdot (0.5 \cdot 0.5) + 2 \left[ 0.005 \right] \left[ 1 - 0.005 \right]
\]

\[
= 0.0198 + 0.5 + 0.01
\]

\[
= 0.529
\]

\& \quad \text{abc is not a power prime implcitant (ppi).}
Definition: predecessor cubes of an implicant \( g_i \) with \( n \) literals.

Successor cubes of an implicant \( g_i \) ...

Example: cube \((abc)\) for a function with inputs \{a, b, c, d, e\}

\[ \frac{ab}{abc} \]

\[ \frac{ac}{abc} \]

\[ \frac{bc}{abc} \]

\[ \frac{a}{abc} \]

\[ \frac{b}{abc} \]

\[ \frac{c}{abc} \]

has predecessor cubes \((ab) (ac) (bc)\) and successor cubes \((abcd) (abc\bar{d}) (abc\bar{c})\) and \((abc\bar{e})\) for example.

Basic idea:  
1. Generate all prime implicants (they are ppi's obviously)
2. Develop power prime implicants from all ppi's
3. Find ppi's to cover the function with minimum power consumption.

Lemma: If \((ab)\) is a ppi and \(a, abc, abd, abe\) are not ppi's, then none of the cubes \(abcd\), \(abc\bar{e}\) abde and \(ab\bar{c}e\) is a ppi.

(assume we have \(a, b, c, d, e\) literals)

Theorem:

Key point: Just need to check the immediate predecessor's power.

1. \( Q: \) all ppi's with \( n-1 \) literals
2. If \( g_i \) \( \notin Q \) and \( g_j \) has the smallest power designation in \( Q \)
3. \( g_i \) has at least one predecessor \( g_j \) cube in \( Q \)

\( g_i \) is a ppi \( \iff \forall g_j \in Q, g_i < g_j, \) Pui\((g_i) < \) Pui\((g_j)\)
Note: We can use \( \leftarrow \) of the above theorem to check for \( \bar{g}_i \).

Def: Given \( g_i \), an implicant of function \( f \), \( g_i' = \bar{g}_l \cdot \bar{g}_2 \cdot \bar{g}_k \) represents the implicant generated by lowering literals \( g_l, g_2, \ldots, g_k \) in \( g_i \).

Example:

\[ g_i = a \cdot b \cdot c \]
\[ g_i^d e = a \cdot b \cdot c \cdot d \cdot e \]

Lemma: Given implicant \( g_i \) with \( p(g_i) = x \) and literal \( l \):

\[ p_{ur} (g_i^l) < p_{ur} (g_i) \iff p_l < \frac{x - x^2}{1 + x^2} \]

How to get this important property?

\[ E(g_i + l) + \left( \sum_{\alpha} C_{M_{\alpha}} E(L_{\alpha}) \right) + C_{M_{0}} E(L) \]

\[ < E(g_i) + \sum_{l} C_{M_{\alpha}} E(L_{\alpha}) \]

\[ E(g_i + l) + E(L) < E(g_i) \]

\[ 2 \cdot p_l (1 - p_l) \cdot 2 \cdot x (1 - x) \]

\[ 2 \left( \frac{p_l p_2 \cdots p_m \cdot p_i}{2} \right) \left( 1 - p_l p_2 \cdots p_m p_i \right) = 2 p_l x (1 - p_l x) \]
function Generate_PPI(F)

begin

    p = generateAllPrimes(F);
    PP = initializePP(p); place prime implicants with n literals in PP;

    for ( i = 1; i < N; i++) do

        Q = findSuccessorCubes(PPi); ← all implicants in Q have i+1 literals

        for each (g ∈ Q) do

            gj = findMinPowerPredecessor(g, PPi); ← return the predecessor cube of g which has the smallest power of g.

            L = literalLoweredInQ(g, gj)

            if (p(L) < \( p(g_j) - p(g_j)^2 \) / (1 + p(g_j)^2)) then

                PP_{i+1} = PP_{i+1} ∪ g;

            endif

        endfor

    endfor

    Return PP

end
Once the set of all PPIs of the function are generated, a minimum covering problem will be used to select a set of PPIs which cover the function and have minimal power cost.

*Power optimization for PLA: no time to cover.*

**Low power Technology Mapping**

[Diagram of logic network and cell library]

**Two phases:**

1. The logic network is decomposed into basic gates (Technology decomposition).

2. The basic-gate network is covered using the library gates by a tree covering technique (cell binding).
• The role of technology mapping:
  
  finish the synthesis of ckt by performing the final
  gate selection from a particular cell library.

• It is not the role of technology mapping to

① Change the ckt structure radically

  e.g. should not find common subexpressions
  between two or more parts of the ckt

② Reduce the number of levels of logic along the
  critical path.

  e.g. TM can just choose fastest gates along
  the critical path, and use the most
  area-efficient combination of gates off
  the critical path.

• Example cell library used for T.M.

<table>
<thead>
<tr>
<th>Gate</th>
<th>Cost</th>
<th>Symbol</th>
<th>primitive</th>
<th>DAG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inverter</td>
<td>2</td>
<td><img src="image" alt="Inverter Symbol" /></td>
<td><img src="image" alt="Inverter Diagram" /></td>
<td></td>
</tr>
<tr>
<td>NAND2</td>
<td>3</td>
<td><img src="image" alt="NAND2 Symbol" /></td>
<td><img src="image" alt="NAND2 Diagram" /></td>
<td></td>
</tr>
<tr>
<td>NAND3</td>
<td>4</td>
<td><img src="image" alt="NAND3 Symbol" /></td>
<td><img src="image" alt="NAND3 Diagram" /></td>
<td></td>
</tr>
</tbody>
</table>

Represent 

everything by

2-input 

NAND gates.
NAND 4 5

\[ \text{\textbullet} \quad \text{AOI21} \quad 4 \]

\[ \text{\textbullet} \quad \text{AOI22} \quad 5 \]

\[ \text{\textbullet} \quad \text{XOR} \quad 4 \]

\[ \text{T.M} \]

Step 1: Convert general cell synthesized to \( \text{NAND} \) gates + inverters

Example:
Called Subject DAG
(directed acyclic graph)

Step II: Technology decomposition
Subject graph partitioning (Remove all fanouts)

Step III. Technology decomposition (Decompose everything to
inverted or 2-input nano-gate)
all others are not changed
Step 4. Tree matching using primitive DAG
(Note: shown by another ckt example)
* Each tree ckt is too small.
Technology Decomposition for Low-power

* Convert a Boolean network to another set consisting of only 2-input NAND and inverter gates.

* Try to minimize the sum of switching activities at the internal nodes of the network.

* Principle: try to inject high switching activity inputs to the decomposition tree as late as possible.

Example: show by AND.

Like Huffman Code generation

\[ E^A(SW) = 0.246 \]

\[ E^B(SW) = 0.512 \]
Principle: Try to hide nodes with high switching activity inside the gates where they drive smaller load capacitances.

Example:

[Diagram of logic gates with labels and connections]

Template matching

Basic gate

Cell-library gates

[Diagram of simplified cell-library gates]
• Post-mapping power optimization
  
• Performed after technology mapping has been finished.

• Try to reduce power further by re-wiring.

Example:

![Diagram showing re-wiring example]

All gates are provided by cell library.

If the switching activity of $B$ is $<$ that of $A$ & input port capacitance is large

$\Rightarrow$ the stated re-wiring can save significant of power.