MULTIPLE-LEVEL LOGIC OPTIMIZATION

©Giovanni De Micheli

Stanford University

Motivation

- Multiple-level networks:
  - Semi-custom libraries.
  - Gates versus macros (PLAs):
    » More flexibility.
    » Better performance.
- Applicable to a variety of designs.

Outline

- Representations.
- Taxonomy of optimization methods:
  - Goals: area/delay.
  - Algorithms: algebraic/Boolean.
  - Rule-based methods.
- Algebraic methods.

Circuit modeling

- Logic network:
  - Interconnection of logic functions.
  - Hybrid structural/behavioral model.
- Bound (mapped) networks:
  - Interconnection of logic gates.
  - Structural model.
Example of network

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

- Minimize area estimate:
  - subject to delay constraints.
- Minimize maximum delay:
  - subject to area constraints.
- Maximize testability.
- Minimize power.

Problem analysis

- Multiple-level optimization is hard.
- Exact methods:
  - Exponential complexity.
  - Impractical.
- Approximate methods:
  - Heuristic algorithms.
  - Rule-based methods.

Estimation

- Area:
  - Number of literals.
  - Number of functions/gates.
- Delay:
  - Number of stages.
  - Refined gate delay models.
  - Sensitizable paths.

Strategies for optimization.

- Improve circuit by step-wise transformations:
  - Modify parts of the network one at a time.
  - Circuit transformations.
- Preserve network behavior.
- Methods differ in:
  - Types of transformations.
  - Selection and order of transformations.
Example elimination

- Eliminate one function from the network.
- Perform variable substitution.
- Example:
  - \( s = r + b' \); \( r = p + a' \)
  - \( s = p + a' + b' \).

Example decomposition

- Break one function into smaller ones.
- Introduce new vertices in the network.
- Example:
  - \( v = a'd + bd + c'd + a'c' \).
  - \( j = a' + b + c'; v = jd + a'c' \).
- Find a common sub-expression of two (or more) expressions.

- Extract sub-expression as new function.

- Introduce new vertex in the network.

- Example:
  - $p = ac + de$; $t = ac + ad + bc + bd + c$;
  - $p = (c + d)e$; $t = (c + d)(a + b) + e$;
  - $k = c + d$; $p = ke$; $t = ka + kb + c$;

- Example:
  - $u = q'c + qc' + qc$; $u = q + c$;
• Simplify a local function by using an additional input that was not previously in its support set.

• Example:
  - $t = ka + kb + c$.
  - $\Rightarrow \quad t = kq + c$
  - Because $q = a + b$.

### Example sequence of transformations

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

---

### Optimization approaches

- **Algorithmic approach:**
  - Define an algorithm for each transformation type.
  - Algorithm is an operator on the network.

- **Rule-based approach:**
  - Rule-data base:
    - Set of pattern pairs.
  - Pattern replacement driven by rules.
Algorithmic approach

- Each operator has well-defined properties:
  - Heuristic methods still used.
  - Weak optimality properties.

- Sequence of operators:
  - Defined by scripts.
  - Based on experience.

Example elimination algorithm

```plaintext
ELIMINATE( G_n(V, E), k) {
    repeat {
        v_x = selected vertex with value below k;
        if (v_x = ∅) return;
        replace x by f_x in the network;
    }
}
```

Example MIS/SIS rugged script

- sweep; eliminate -1
- simplify -m nocomp
- eliminate -1
- sweep; eliminate 5
- simplify -m nocomp
- resub -a
- f_x
- resub -a; sweep
- eliminate -1; sweep
- full-simplify -m nocomp
**Boolean and algebraic methods**

- **Boolean methods:**
  - Exploit properties of logic functions.
  - Use *don’t care* conditions.
  - Complex at times.
- **Algebraic methods:**
  - View functions as *polynomials*.
  - Exploit properties of polynomial algebra.
  - Simpler, faster but weaker.

**Boolean substitution:**
- $h = a + bcd + e; q = a + cd$
- $\Rightarrow h = a + bq + e$
- Because $a + bq + e = a + b(a + cd) + e = a + bcd + e$.

**Algebraic substitution:**
- $t = ka + kb + e$.
- $\Rightarrow t = kq + e$
- Because $q = a + b$.

**Algebraic model**

- **Boolean algebra:**
  - Complement.
  - Symmetric distribution laws.
  - *Don’t care* sets.
- **Algebraic methods:**
  - Boolean functions $\rightarrow$ polynomials.
  - Expressions (*sum of product* forms).

**Example**

- **Boolean substitution:**
  - $h = a + bcd + e; q = a + cd$
  - $\Rightarrow h = a + bq + e$
  - Because $a + bq + e = a + b(a + cd) + e = a + bcd + e$.

**Algebraic division**

- Given two algebraic expressions:
  - $f_{\text{quotient}} = f_{\text{dividend}}/f_{\text{divisor}}$ when:
    - $f_{\text{dividend}} = f_{\text{divisor}} \cdot f_{\text{quotient}} + f_{\text{remainder}}$
    - $f_{\text{divisor}} \cdot f_{\text{quotient}} \neq 0$
    - and the support of $f_{\text{divisor}}$ and $f_{\text{quotient}}$ is disjoint.
**Example**

- **Algebraic division:**
  - Let \( f_{\text{dividend}} = ac + ad + bc + bd + e \) and \( f_{\text{divisor}} = a + b \).
  - Then \( f_{\text{quotient}} = c + d \) and \( f_{\text{remainder}} = e \).
  - Because \( (a + b) \cdot (c + d) + e = f_{\text{dividend}} \) and \( \{a, b\} \cap \{c, d\} = \emptyset \).

- **Non-algebraic division:**
  - Let \( f_i = a + bc \) and \( f_j = a + b \).
  - Then \( (a + b) \cdot (a + c) = f_i \) but \( \text{sup}(f_j) \cap \{a, c\} \neq \emptyset \).

**An algorithm for division**

**ALGEBRAIC\text{DIVISION}(A, B)\**

\[
\begin{align*}
\text{for} \ (i = 1 \ \text{to} \ n) \ \{ \\
\quad D = \{C_i^A \text{ such that } C_j^A \supseteq C_i^B \}; \\
\quad \text{if} \ (D == \emptyset) \ \text{return}(\emptyset, A); \\
\quad D_i = D \ \text{with var. in } \text{sup}(C_i^B) \ \text{dropped}; \\
\quad \text{if} \ i = 1 \\
\quad \quad Q = D_i; \\
\quad \text{else} \\
\quad \quad Q = Q \cap D_i; \\
\quad \}\ \\
R = A - Q \times B; \\
\text{return}(Q, R);
\end{align*}
\]

**Example**

\( f_{\text{dividend}} = ac + ad + bc + bd + e \); \( f_{\text{divisor}} = a + b \):

- \( A = \{ac, ad, bc, bd, e\} \) and \( B = \{a, b\} \).
- \( i = 1 \):
  - \( C_i^B = a, D = \{ac, ad\} \) and \( D_1 = \{c, d\} \).
  - Then \( Q = \{c, d\} \).
- \( i = 2 = n \):
  - \( C_i^B = b, D = \{bc, bd\} \) and \( D_2 = \{c, d\} \).
  - Then \( Q = \{c, d\} \cap \{c, d\} = \{c, d\} \).

- Result:
  - \( Q = \{c, d\} \) and \( R = \{e\} \).
  - \( f_{\text{quotient}} = c + d \) and \( f_{\text{remainder}} = e \).
Theorem

Given $f_i$ and $f_j$, then $f_i/f_j$ is empty when:
- $f_j$ contains a variable not in $f_i$.
- $f_j$ contains a cube whose support is not contained in that of any cube of $f_i$.
- $f_j$ contains more terms than $f_i$.
- The count of any variable in $f_j$ than in $f_i$.

Substitution algorithm

SUBSTITUTE( $G_n(V,E)$ ){
  for ( $i = 1, 2, \ldots |V|$ ) {
    for ( $j = 1, 2, \ldots |V|; j \neq i$ ) {
      $A$ = set of cubes of $f_i$;
      $B$ = set of cubes of $f_j$;
      if ($A, B$ pass the filter test ) {
        $(Q, R) = ALGEBRAIC\_DIVISION(A, B)$
        if ($Q \neq 0$) {
          $f_{quotient}$ = sum of cubes of $Q$;
          $f_{remainder}$ = sum of cubes of $R$;
          if ( substitution is favorable)
            $R = j \cdot f_{quotient} + f_{remainder}$;
        }
      }
    }
  }
}

Substitution

Consider expression pairs.

Apply division (in any order).

If quotient is not void:
- Evaluate area/delay gain
  - Substitute $f_{dividend}$ by $j \cdot f_{quotient} + f_{remainder}$
    where $j = f_{divisor}$.
- Use filters to reduce divisions.

Extraction

Search for common sub-expressions:
- Single-cube extraction: monomial.
- Multiple-cube (kernel) extraction.

Search for appropriate divisors.
**Definitions**

- **Cube-free expression:**
  - Cannot be factored by a cube.

- **Kernel** of an expression:
  - Cube-free **quotient** of the expression divided by a cube, called **co-kernel**.

- **Kernel set** \( K(f) \) of an expression:
  - Set of kernels.

**Theorem**

(Brayton and McMullen)

- Two expressions \( f_a \) and \( f_b \) have a common multiple-cube divisor \( f_d \) if and only if:
  - there exist kernels \( k_a \in K(f_a) \) and \( k_b \in K(f_b) \) such that \( f_d \) is the sum of 2 (or more) cubes in \( k_a \cap k_b \).

- **Consequence:**
  - If kernel intersection is void, then the search for common sub-expression can be dropped.

**Example**

\[ f_x = ace + bce + de + g \]

- Divide \( f_x \) by \( a \). Get \( ce \). Not cube free.
- Divide \( f_x \) by \( b \). Get \( ce \). Not cube free.
- Divide \( f_x \) by \( c \). Get \( ae + be \). Not cube free.
- Divide \( f_x \) by \( ce \). Get \( a + b \). Cube free. **Kernel!**
- Divide \( f_x \) by \( d \). Get \( e \). Not cube free.
- Divide \( f_x \) by \( e \). Get \( ac + bc + d \). Cube free. **Kernel!**
- Divide \( f_x \) by \( g \). Get 1. Not cube free.
- Expression \( f_x \) is a kernel of itself because cube free.
- \( K(f_x) = \{ (a + b); (ac + bc + d); (ace + bce + de + g) \} \).

**Example**

\[ f_x = ace + bce + de + g \]
\[ f_y = ad + bd + cde + ge \]
\[ f_z = abc \]

- \( K(f_x) = \{ (a + b); (ac + bc + d); (ace + bce + de + g) \} \).
- \( K(f_y) = \{ (a + b + ac); (cd + g); (ad + bd + cde + ge) \} \).
- The kernel set of \( f_x \) is empty.
- Select intersection \( (a + b) \)

\[ f_w = a + b \]
\[ f_z = abc \]
Kernel set computation

- Naive method:
  - Divide function by elements in power set of its support set.
  - Weed out non cube-free quotients.

- Smart way:
  - Use recursion:
    * Kernels of kernels are kernels.
  - Exploit commutativity of multiplication.

Recursive kernel computation

Simple algorithm

```plaintext
R_KERNELS(f) {
  K = 0;
  foreach variable x \in sup(f) {
    if |CUBES(f, x)| \geq 2 {
      f^C = largest cube containing x,
      such that CUBES(f, C) = CUBES(f, x);
      K = K \cup R_KERNELS(f/f^C);
    }
  }
  K = K \cup f;
  return(K);
}

CUBES(f, C) {
  return the cubes of f whose support includes C;
}
```

Analysis

- Some computation may be redundant:
  - Example:
    * Divide by $a$ and then by $b$.
    * Divide by $b$ and then by $a$.
  - Obtain duplicate kernels.

- Improvement:
  - Keep a pointer to literals used so far.
Example

\[ f = ace + bce + de + g \]

- Literals \(a\) or \(b\). No action required.
- Literal \(c\). Select cube \(\alpha\):
  - Recursive call with arguments: \((ace + bce) / \alpha = a + b\);
    pointer \(j = 3 + 1\).
  - Call considers variables \(\{d, e, g\}\). No kernel.
  - Add \(a + b\) to the kernel set at the last step.
- Literal \(d\). No action required.
- Select cube \(e\):
  - Recursive call with arguments: \((ac + bc + d)\) and pointer \(j = 5 + 1\).
  - Call considers variable \(\{g\}\). No kernel.
  - Add \(ac + bc + d\) to the kernel set at the last step.
- Literal \(e\). No action required.
- Add \(ac + bce + de + g\) to the kernel set.
- \(K = \{(ac + bce + de + g), (ac + bc + d), (a + b)\}\).

Matrix representation of kernels

- Boolean matrix:
- Rectangle \((R, C)\):
  - Subset of rows and columns with all entries equal to 1.
- Prime rectangle:
  - Rectangle not inside any other rectangle.
- Co-rectangle \((R, C')\) of a rectangle \((R, C)\):
  - \(C'\) are the columns not in \(C\).
- A co-kernel corresponds to a prime rectangle with at least two rows.

Example

\[ f_x = ace + bce + de + g \]

<table>
<thead>
<tr>
<th>cube</th>
<th>var (R \backslash C)</th>
<th>(a)</th>
<th>(b)</th>
<th>(c)</th>
<th>(d)</th>
<th>(e)</th>
<th>(g)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ace</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>bce</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>de</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>(g)</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

- Rectangle (prime): \(\{1, 2\}, \{3, 5\}\)
  - Co-kernel \(ce\).
- Co-rectangle: \(\{1, 2\}, \{1, 2, 4, 6\}\).
  - Kernel \(a + b\).
Single-cube extraction

- Form auxiliary function:
  - Sum of all local functions.

- Form matrix representation:
  - A rectangle with two rows represents a common cube.
  - Best choice is a prime rectangle.

- Use function ID for cubes:
  - Cube intersection from different functions.

Cube extraction algorithm

\[
CUBE\_EXTRACT( G_n(V,E) )\{
\text{\textbf{while} (some favorable \underline{\text{common \ cube}} exist) \{} \\
\phantom{{\text{\textbf{while} (some favorable \underline{\text{common \ cube}} exist) \{} } C = \text{select common cube to extract}; \\
\phantom{{\text{\textbf{while} (some favorable \underline{\text{common \ cube}} exist) \{} } \text{Generate new label } l; \\
\phantom{{\text{\textbf{while} (some favorable \underline{\text{common \ cube}} exist) \{} } \text{Add } v_l \text{ to the network with expression } f_l = f^C; \\
\phantom{{\text{\textbf{while} (some favorable \underline{\text{common \ cube}} exist) \{} } \text{Replace all functions } f, \text{ where } f_l \text{ is a divisor,} \\
\phantom{{\text{\textbf{while} (some favorable \underline{\text{common \ cube}} exist) \{} } \text{by } l : \text{quotient} + \text{remainder}; \\
\text{\textbf{endwhile}}; \\
\text{\textbf{end}}}
\]

Example

- Expressions:
  - \( f_x = ace + bce + de + g \)
  - \( f_y = cde + b \)

- Auxiliary function:
  - \( f_{aux} = ace + bce + de + g + cde + b \)

- Matrix:

<table>
<thead>
<tr>
<th>( \text{cube} )</th>
<th>( \text{var} )</th>
<th>( a )</th>
<th>( b )</th>
<th>( c )</th>
<th>( d )</th>
<th>( e )</th>
<th>( g )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( ace )</td>
<td>( x )</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>( bce )</td>
<td>( x )</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>( de )</td>
<td>( x )</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>( g )</td>
<td>( x )</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( cde )</td>
<td>( s )</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>( b )</td>
<td>( s )</td>
<td>6</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- Prime rectangle: \( \{(1, 2, 5), (3, 5)\} \)

- Extract cube \( ce \).

Multiple-cube extraction
Multiple-cube extraction

- We need a kernel/cube matrix.
- Relabeling:
  - Cubes by new variables.
  - Kernels by cubes.
- Form auxiliary function:
  - Sum of all kernels.
- Extend cube intersection algorithm.

Example (2)

- \( f_{\text{aux}} = x_\alpha x_b + x_\omega x_b + x_\alpha x_\beta x_d \).
- Co-kernel: \( x_\alpha x_b \).
  - \( x_\alpha x_b \) corresponds to kernel intersection \( a + b \).
  - Extract \( a + b \) from \( f_p \) and \( f_q \).

Example

- \( f_p = ace + bce + de \).
  - \( K(f_p) = \{(a + b)\} \).
- \( f_q = ace + b + d \).
  - \( K(f_q) = \{(a + b); (ac + be + d)\} \).
- Relabeling:
  - \( x_a = a; x_b = b; x_\alpha = ae; x_\beta = be; x_c = d \);
    * \( K(f_p) = \{x_\alpha, x_b\} \)
    * \( K(f_q) = \{x_\alpha, x_b; x_\omega, x_\beta, x_d\} \).

Kernel extraction algorithm

\( \text{KERNEL}_{\text{EXTRACT}}(G(V, E), n, k) \)\{
  while (some favorable common kernel intersection exist) \{
    Compute kernel set of level \( \leq k \);
    for \( i = 1 \) to \( n \) \{
      Compute kernel intersections; \( f \) = select kernel intersection to extract; Generate new label \( l \);
      Add \( v_j \) to the network with expression \( f_l = f \);
      Replace all functions \( f \) where \( f_j \) is a divisor
      by \( l \cdot f_{\text{quotient}} + f_{\text{remainder}} \);
    \}
  \}
\}
Decomposition

\[ x = ace + bce + de + g \]

\[ x = te + g \]

\[ s = a + b \]

\[ t = sc + dt = ac + bc + d \]

Different ways:

- Method of Ashenhurst and Curtis.
- NAND/NOR decomposition.

Kernel-based decomposition:

- Divide expression recursively.

Example

\[ f_x = ace + bce + de + g \]

- Select kernel \( ac + bd + d \).
- Decompose: \( f_x = te + g \); \( f_t = ac + bc + d \).
- Recur on the quotient \( f_t \):
  - Select kernel \( a + b \):
  - Decompose: \( f_t = a + b \); \( f_s = a + b \);

Decomposition algorithm

\[ DECOMPOSE( G_n(V, E) , k)\{
  \text{repeat } \{
    v_x = \text{selected vertex with expression whose size is above } k;
    \text{if } (v_x = \emptyset) \text{ return;}
    \text{decompose expression } f_x;
  \}
}\]
Summary

Algebraic transformations

- View Boolean functions as algebraic expression.
- Fast manipulation algorithms.
- Some optimality lost, because Boolean properties are neglected.
- Useful to reduce large networks.