Outline

• Multi-level circuit representations
• Minimization methods
  – goals: area, delay, power
  – algorithms: algebraic, boolean
  – rule-based methods
• Examples of transformations
• Boolean and algebraic models

Disclaimer: lecture notes based on originals by Giovanni De Micheli
Motivation

• Multiple level networks
  – Semi-custom libraries
  – Advantages of gates versus macros (PLA)
    ‣ more flexible
    ‣ better performance
      • might want to minimize certain paths

• Applicable to a variety of designs
Circuit modeling

• Logic network
  – interconnection of logic functions
  – hybrid structural/behavioral model

• Bound (mapped) gates
  – interconnection of logic gates
  – structural model
Example of a bound network
Logic equations of example

\[
\begin{align*}
p &= ce + de \\
q &= a + b \\
r &= p + a' \\
s &= r + b' \\
t &= ac + ad + bc + bd + e \\
u &= q'c + qc' + qc \\
v &= a'd + bd + c'd + ae' \\
w &= v \\
x &= s \\
y &= t \\
z &= u
\end{align*}
\]
Example of network

(a)

(b)
Example
circuit output function

\[ f = \begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix} = \begin{bmatrix} a'd + bd + c'd + ae' \\ a' + b' + ce + de \\ ac + ad + bc + bd + e \\ a + b + c \end{bmatrix} \]
Network optimization

• Minimize area (power) estimate
  – subject to delay constraints
• Minimize maximum delay
  – subject to area (power) constraints
• Maximize testability
• Minimize power
Estimation

• Area:
  – number of literals
    ‣ in MOS, # literals = # poly strips
  – number of functions/gates

• Delay:
  – number of stages
    ‣ most often used metric
  – refined gate delay models
  – sensitizable paths
    ‣ paths for which there are conditions under which a signal propagates through the path
Problem analysis

• Multiple-level optimization is hard

• Exact methods:
  – exponential complexity
  – impractical
    ‣ die on reasonably sized networks

• Approximate (heuristic) methods:
  – algorithms which explore part of exact soln. space
  – rule-based methods
    ‣ replacement of subcircuits by other subcircuits
Strategies for minimization

- Improve circuits step by step
  - 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
  – $s = r + b'$; $r = p + a'$
  – $\Rightarrow s = p + a' + b'$
Example: elimination

\[ v = a'd + bd + c'd + ae' \]

\[ p = ce + de \]
\[ r = p + a' \]
\[ s = r + b' \]

\[ t = ac + ad + bc + bd + e \]

\[ q = a + b \]
\[ u = q'c + qc' + qc \]

\[ v = a'd + bd + a'c + ae' \]

\[ p = ce + de \]

\[ s = p + a' + b' \]

\[ t = ac + ad + bc + bd + e \]

\[ q = a + b \]
\[ u = q'c + qc' + qc \]
Example decomposition

- Break one function into smaller ones
- Introduce new vertices in the network
- Example:
  - $v = a'd + bd + c'd + ae'$
  - $\Rightarrow j = a' + b + c'; \ v = jd + ae'$
Example: decomposition

\[ v = a'd + bd + c'd + ae' \]
\[ p = ce + de \]
\[ r = p + a' \]
\[ s = r + b' \]
\[ t = ac + ad + bc + bd + e \]
\[ q = a + b \]
\[ u = q'c + qc' + qc \]
\[ j = a' + b + c' \]
\[ v = j d + ae' \]
Example extraction

• Find a common sub-expression of two (or more) expressions
• Extract sub-expression as new function
• Introduce new vertex in the network
• Example:
  – $p = ce + de; \quad t = ac + ad + bc + bd + e$
  – $p = (c + d)e; \quad t = (c + d)(a + b) + e$
  – $\Rightarrow k = c + d; \quad p = ke; \quad t = ka + kb + e$
Example: extraction

\[ v = a'd + bd + c'd + ae' \]

\[ p = ce + de \]
\[ r = p + a' \]
\[ s = r + b' \]

\[ t = ac + ad + bc + bd + e \]

\[ q = a + b \]
\[ u = q'c + qc' + qc \]
Example
simplification

• Simplify a local function
• Example:
  – $u = q'c + qc' + qc$
  – $\Rightarrow u = q + c$
Example: simplification

\[
v = a'd + bd + c'd + ae'
\]

\[
p = ce + de
\]

\[
r = p + a'
\]

\[
s = r + b'
\]

\[
t = ac + ad + bc + bd + e
\]

\[
q = a + b
\]

\[
u = q'c + qc' + qc
\]
Example substitution

• Simplify a local function by using an additional input that was not previously in the function’s support set

• Example:
  – \( t = ka + kb + e \)
  – \( \Rightarrow t = kq + e \)
  – (because \( q = a + b \))
Example: substitution

\[ v = a'd + bd + c'd + ae' \]

\[ p = ke \]

\[ r = p + a' \]

\[ s = r + b' \]

\[ k = c + d \]

\[ t = ka + kb + e \]

\[ q = a + b \]

\[ u = q'c + qc' + qc \]
Example outcome
sequence of transformations

\[ j = a' + b + c' \]
\[ k = c + d \]
\[ q = a + b \]
\[ s = ke + a' + b' \]
\[ t = kq + e \]
\[ u = q + c \]
\[ v = jd + ae' \]
Minimization 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

• Set a threshold $k$ (usually 0)
  – $k$ corresponds to the maximum area (or power or delay) increase allowed

• Examine all expressions

• Eliminate expressions if the increase in literals does not exceed the threshold
  – goal: reduce the number of logic expressions
Example elimination algorithm

\[
\text{ELIMINATE} \ (G(V,E), \ k) \ \{
\text{repeat} \ \\
\quad v_x = \text{selected vertex with value} < k \ ; \ \\
\quad \text{if} \ (v_x == \text{empty vertex}) \ \text{return} ; \ \\
\quad \text{replace} \ x \ \text{by} \ f_x \ \text{in the network} ; \ \\
\}
\]
Example
MIS/SIS rugged script

- sweep; eliminate -1
- simplify -m nocomp
- eliminate -1
- sweep; eliminate 5
- simplify -m nocomp
- resub -a
- fx
- resub -a; sweep
- eliminate -1; sweep
- full-simplify -m nocomp
Algebraic Division

• $f_{\text{quotient}} = f_{\text{dividend}} / f_{\text{divisor}}$ when
  \[- f_{\text{dividend}} = f_{\text{divisor}} \ast f_{\text{quotient}} + f_{\text{remainder}}
  \]- $f_{\text{divisor}} \ast f_{\text{quotient}} \neq 0$
  \[- \text{the literals of } f_{\text{divisor}} \text{ are disjoint from the literals of } f_{\text{quotient}}\]

• Note: if $f_{\text{remainder}} = 0$, then $f_{\text{divisor}}$ is called a \textit{factor}, and $f_{\text{divisor}}$ is said to \textit{factor} $f_{\text{quotient}}$
Definitions

- **Cube-free expression**
  - cannot be factored by a cube (implicant)

- **Kernel of an expression**
  - cube-free quotient of the expression divided by a cube (the cube used for division is called a co-kernel)

- **Kernel set** $K(f)$ of an expression
  - set of kernels
Example

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

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

\[ y = ad + bd + cde + ge \]

- The literals are \( a, b, c, d, e, g \)
- Try all combinations of literals
- There exists a recursive function for calculating kernels very fast (we won't cover this recursive function)
Example: use kernel computation to implement extraction

Steps: (1) calculate $K(x)$, (2) calculate $K(y)$, (3) calculate $K(z)$, (4) if there are some kernels in common, perform extraction with largest kernel (largest # of cubes, break tie by choosing largest # of literals)
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
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 substitution
  – $t = ka + kb + e$
  – $\Rightarrow t = kq + e$
  – (because $q = a + b$)
Summary

• Multi-level logic synthesis is performed by step-wise transformations
• Algorithms are based on both the boolean and algebraic models
• Rule-based models