Performance Driven Logic Optimization

Prof. Kurt Keutzer
Michael Orshansky
EECS
University of California
Berkeley, CA
Thanks to S. Devadas

RTL Design Flow
Modern Approach to Logic Optimization

Divide logic optimization into two subproblems:

- Technology-independent optimization
  - determine overall logic structure
  - estimate costs (mostly) independent of technology
  - simplified cost modeling

- Technology-dependent optimization (technology mapping)
  - binding onto the gates in the library
  - detailed technology-specific cost model

Orchestration of various optimization/ transformation techniques for each subproblem
Performance Parameters

Speed
- Speed: Improve speed at which digital circuit can be clocked
- Reduce clock period

Power
- Reduce dynamic power dissipation
- Reduce static power dissipation

Delay Optimization

Being able to meet performance requirements is absolutely essential

Timing Optimization
- Technology independent
- Technology dependent

Technology independent optimizations include global restructuring of network to minimize critical path lengths
Problem formulation - Arrival Time

Arrival time $A(v)$ for a node $v$ is time when signal arrives at node $v$.

\[
A(x) = \max_{u \in FI(v)} (A(u) + d_{u \rightarrow v})
\]

where $d_{u \rightarrow v}$ is delay from $u$ to $v$, $FI(v) = \{X,Y\}$, and $v = \{Z\}$.

Required Time

Required time $R(v)$ is the time before which a signal must arrive to avoid a timing violation.

\[
R(X) = \min_{u \in FO(v)} (R(u) - d_{u \rightarrow X})
\]

where $FO(v) = \{Y,Z\}$ and $v = \{X\}$.
**Timing Slack**

From arrival and required time can compute slack. For each node \( v \):

\[
S(v) = R(v) - A(v)
\]

Slack reflects criticality of a node

- **Positive slack**
  - Node is not on critical path. Timing constraints met.

- **Zero slack**
  - Node is on critical path. Timing constraints are barely met.

- **Negative slack**
  - There is a timing violation

Slack distribution is key for timing optimization!

---

**Delay Tracing**

Obtain arrival, required and slack times at each node given arrival times for inputs and required time at output

MAX for arrival, MIN for required
Critical Section/Critical Subgraph

The critical section of the Boolean network is the set of nodes with minimum slack. Paths through primary inputs and nodes with minimum slack are critical.

Only slacks are shown.

Modern Approach to Logic Optimization

Divide logic optimization into two subproblems:
- Technology-independent optimization
  - determine overall logic structure
  - estimate costs (mostly) independent of technology
  - simplified cost modeling
- Technology-dependent optimization (technology mapping)
  - binding onto the gates in the library
  - detailed technology-specific cost model

Orchestration of various optimization/transformation techniques for each subproblem.
Tech Independent Optimization: Restructuring to Improve Delay

- Identify Critical Section
- Late input signals

Partial Collapse of Critical Subsection

- Nodes in critical section that fan out outside of critical section are duplicated
Selective Collapsing

"Collapse" nodes into their fanout to increase the size of each node

\[
\begin{align*}
f &= a h + b h' + c d' \\
g &= b h \\
h &= a c + d'
\end{align*}
\]

Goals:
- remove bad initial structure
- reduce logic level depth
- expose further optimization opportunities

Collapse \( h \)

\[
\begin{align*}
f &= a (a c + d') + b (a' d' + c' d') + c d' \\
g &= b (a c + d')
\end{align*}
\]

Simplify

\[
\begin{align*}
f &= a c + a d' + a' b d' + b c' d' + c d \\
g &= a b c + b d'
\end{align*}
\]

Restructuring to Improve Delay

- Place timing-critical nodes closer to output
  - Make them pass through fewer gates
  - After collapse, a divisor is selected such that substituting \( k \) into \( f \) places critical signal \( c \) and \( d \) closer to output

Collapse critical section

Re-extract factor \( k \)
Restructuring to Improve Delay: Example

Generalized Bypass Transform

- Speed up paths by making them false, rather than short
  - Inspired by performance enhancements in adders
  - In adders, carry chain is the critical path
  - Carry-bypass adders use a bypass to directly generate an answer when appropriate

\[
\begin{align*}
\text{Full Adder} & : p_0 \\
\text{Full Adder} & : p_1 \\
\text{MUX} & : \text{MUX}
\end{align*}
\]
Generalized Bypass Transformation

- Generalization of bypass to arbitrary combinational circuit
- Construct bypasses around partial paths
- Speed up paths by making them false, rather than short
- Delay is given by true critical paths
- Path delay is not equal to path depth

Generalized Bypass Formulation

- Make critical path false ⇒ speed up circuit
- Bypass logic of critical path

![Diagram of generalized bypass formulation]
Identifying Transform Candidates

- Consider some long path \( P = f_0, ..., f_n \).
- Suppose the delay to each side input \( g_{ik} \) is substantially less than the delay to \( f_i \).
- Why do we have to wait for \( f_0 \)? \( f_0 \) might control value of \( f_n \).
- But we can compute the case when \( f_n \) depends on \( f_0 \): \( \frac{\partial f_n}{\partial f_0} \).
- When this is true just tie \( f_n \) to \( f_0 \). (build a bypass).

Logic Optimization

- 2-level Logic opt
- multilevel Logic opt
- netlist
- logic independent
- tech dependent
Modern Approach to Logic Optimization

Divide logic optimization into two subproblems:

- Technology-independent optimization
  - determine overall logic structure
  - estimate costs (mostly) independent of technology
  - simplified cost modeling
- Technology-dependent optimization (technology mapping)
  - binding onto the gates in the library
  - detailed technology-specific cost model

Orchestration of various optimization/transformation techniques for each subproblem

Delay Optimization

If delay is independent of the gate which is driven:

- Tree covering algorithm provides minimum arrival-time cover
  a) Compute and store arrival time at each node
  b) Choose minimum-arrival time match at each node

For more general delay model, principle of optimality does not apply
Constant Load Covering

Incorporating Load-Dependent delays

What is the load seen by g?

Optimum match depends on forward (unmapped) part of the tree
Variable Load Delay Optimization

Create bin for each load value
Array of solutions at each node, one per load value
Compute arrival time for each match for each load value
When evaluating a match, use the optimal solution at the input node which is appropriate for the load presented by this match.

Variable Load

better for real loads!
Library and Delay Information

<table>
<thead>
<tr>
<th>Area</th>
<th>Load-Dependent Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>INV (1)</td>
<td>INV (1)</td>
</tr>
<tr>
<td>NAND2 (2)</td>
<td>NAND2 (2)</td>
</tr>
<tr>
<td>NAND3 (3)</td>
<td>NAND2 (4)</td>
</tr>
<tr>
<td>AOI21 (3)</td>
<td>NAND3 (3)</td>
</tr>
<tr>
<td></td>
<td>NAND3 (5)</td>
</tr>
<tr>
<td></td>
<td>NAND4</td>
</tr>
<tr>
<td></td>
<td>NAND4</td>
</tr>
<tr>
<td></td>
<td>NAND4</td>
</tr>
<tr>
<td></td>
<td>NAND4</td>
</tr>
<tr>
<td></td>
<td>NAND4</td>
</tr>
<tr>
<td></td>
<td>NAND4</td>
</tr>
<tr>
<td></td>
<td>NAND4</td>
</tr>
<tr>
<td></td>
<td>NAND4</td>
</tr>
<tr>
<td></td>
<td>NAND4</td>
</tr>
<tr>
<td></td>
<td>NAND4</td>
</tr>
</tbody>
</table>

Variable Load Covering

Array of solutions

INV
AOI21
NAND2
NAND4
NAND3
Variable Load Covering Result

Array of solutions

(all solutions NAND2 sees INV)

NAND3

+ NAND2

AOI

AOI

INV

AOI or NAND3

AOI

If driving NAND3 will get AOI21 solution with arrival time 13

If driving INV choose either AOI21 or NAND3 solution

Minimum Arrival Time Cover

Makes all sub-trees maximally fast which wastes area

area = 8
delay = 8
(slack = 4)

delay = 12
Reclaiming Area Off Critical Path

Identify sub-trees with slack > 0 and check if lesser or minimum area result can be used instead.

NAND4  delay = 9, area = 4

delay = 12