Performance Driven Logic Optimization

Prof. Kurt Keutzer
EECS
University of California
Berkeley, CA

RTL Design Flow
Perform a variety of transformations and optimizations
- Structural graph transformations
- Boolean transformations
- Mapping into a physical library

smaller, faster
less power
Performance Parameters

Speed – we will talk about this today
- Speed: Improve speed at which digital circuit can be clocked
- Reduce clock period

Power – we will talk about this later
- Reduce dynamic power dissipation
- Reduce static power dissipation

Cycle Time - Critical Path Delay

Cycle time (T) cannot be smaller than longest path delay ($T_{\text{max}}$)

Longest (critical) path delay is a function of:

Total gate, wire delays
- logic levels

\[ T_{\text{max}} + T_{\text{setup}} + T_{\text{skew}} + T_{\text{clk-to-Q}} \leq T \]
Speed up - Clock-skew

If clock network has unbalanced delay – clock skew

Cycle time is also a function of clock skew ($T_{skew}$)

Two approaches:
- Minimize skew
- “Useful skew”
- We’ll talk about these later

$$T_{max} + T_{setup} + T_{skew} + T_{clk-to-Q} \leq T$$

Speed-up - Clock to Q

$T_{clk-to-Q}$: time from arrival of clock signal till change at FF output

Reduce clk-to-Q with more efficient registers – more later?

$$T_{max} + T_{setup} + T_{skew} + T_{clk-to-Q} \leq T$$
How do we speed up a circuit?

Cycle time (T) cannot be smaller than longest path delay (T_{max})

Longest (critical) path delay is a function of:

- Total gate, wire delays
- Logic levels

\[ T_{\text{clock1}} + T_{\text{setup}} + T_{\text{skew}} + T_{\text{clk-to-Q}} \leq T \]

So how do we reduce gate delays?

- Eliminate gates altogether
- Optimization, simplification
- Move gates off the critical path
- Increase Vdd – problems? We’ll talk about this later in the semester
- Reduce Vth – problems? We’ll talk about this later in the semester
- Better circuit design of gates (example?)
- Increase drive of driving gates through cell/transistor sizing
- SOI, Thin-oxide
- Dynamic logic
- Pin swapping
- Critical dimension reduction (e.g. poly linewidth) – phase shift mask
Delay Optimization

Eliminate gates altogether
- Optimization, simplification
- Move gates off the critical path

Timing Optimization
- Technology independent
- Technology dependent

Technology independent optimizations include global restructuring of network to minimize critical path lengths

Opt: Reduce to Combinational Optimization

- Flip-flops
- Combinational Logic
- Input arrival times
- Input drive
- Output required times
- Output load
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

Logic Optimization

2-level Logic opt
multilevel Logic opt
Generic Library
Real Library

Library
netlist
logic optimization
netlist
tech independent
tech dependent
Tech Indep: Restructuring to Improve Delay

Identify Critical Section

Late input signals

Restructuring to Improve Delay - 2

Collapse critical section

Re-extract factor $k$

Close to output
Interesting Example: Carry Bypass Adder

V. Oklobdzija - Jrnl. of VLSI Signal Processing, 1991

Generalized Bypass Formulation

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

Kurt Keutzer
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

Logic Optimization

- Library
  - logic optimization
  - netlist

- tech independent
  - 2-level Logic opt
  - multilevel Logic opt

- tech dependent
  - Generic Library
  - Real Library

Kurt Keutzer
Can we use dynamic programming?

Can we use a dynamic programming formulation to find a minimum speed cover of the candidate tree?

Dynamic Programming for Min Speed

- INV delay 1 + max(6) = 7
- NAND2 delay 2 + max(4, 1) = 6
- NAND2 Delay 2 + max(2, 0) = 4
- AOI21 delay 3 max(2, 0, 0) = 5
What else do we need to consider?

We need to time the cover based on proper arrival times
- Arrival times will only be known when the arrival times of prior (topologically) trees in the DAG are known
- Map from inputs to outputs

Mapping of the tree may produce too much slack on off-critical paths — we’ll discuss this later in the lecture

Selection of a cell in the network depends on the load it is facing!

Tree Covering for Delay – driving inv

![Diagram showing tree covering for delay with calculations and timing details]
Variable Load

$$2 + 2 = 4$$

$$\text{MAX} (4,0,0) + 4 = 8$$

$$\text{MAX} (8,0) + 5 = 13$$

better for real loads!

Incorporating load-dependent delays

What is the load seen by g?

Optimum match depends on forward (unmapped) part of the tree

How can we handle this in the dynamic programming framework?
Variable Load Delay Optimization

Create bin for each load value that we may face

*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

---

Library and Delay Information

<table>
<thead>
<tr>
<th>Area</th>
<th>INV (1)</th>
<th>NAND2 (3)</th>
<th>NAND2 (3)</th>
<th>INV (1)</th>
<th>ND2 (2)</th>
<th>NAND2 (3)</th>
<th>AOI21</th>
<th>NAND3 (3)</th>
<th>AOI21</th>
<th>NAND3 (3)</th>
<th>AOI21</th>
<th>NAND4</th>
</tr>
</thead>
<tbody>
<tr>
<td>CELL</td>
<td>NAND2</td>
<td>NAND2</td>
<td>NAND3</td>
<td>AOI21</td>
<td>NAND2</td>
<td>AOI21</td>
<td>NAND3</td>
<td>NAND4</td>
<td></td>
<td>NAND3</td>
<td>NAND4</td>
<td></td>
</tr>
<tr>
<td>DELAY WHEN DRIVING</td>
<td>NAND3</td>
<td>NAND2</td>
<td>NAND3</td>
<td>NAND3</td>
<td>NAND2</td>
<td>NAND3</td>
<td>NAND4</td>
<td>NAND4</td>
<td>NAND4</td>
<td>NAND4</td>
<td>NAND4</td>
<td>NAND4</td>
</tr>
</tbody>
</table>
Variable Load Covering

Array of solutions

INV
AOI21
NAND2
NAND4
NAND3

If driving NAND3 will get AOI21 solution with arrival time 13
If driving AOI21 or ND2 will get AOI21 solution
If driving INV choose either AOI21 or NAND3 solution

Variable Load Covering Result

(all solutions NAND2 sees INV)

Array of solutions

INV
AOI21
NAND2
NAND4
NAND3

cell chosen/delay when driving

If driving NAND3 will get AOI21 solution with arrival time 13
If driving AOI21 or ND2 will get AOI21 solution
If driving INV choose either AOI21 or NAND3 solution
Summary of load-dependent mapping

Load-dependent delay shows how dynamic programming paradigm can be extended
What’s the computation time of this approach?
What’s wrong or incomplete with this picture?
When do we know wiring capacitance?
What can we do to address it?

Overall bin-loading not so successful, but it’s an interesting way of extending dynamic programming

Minimum Arrival Time Cover

Makes all sub-trees maximally fast which wastes area

2 + 2 = 4
4 + 4 = 8
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

Improving Libraries

Library

logic optimization

netlist

tech independent

tech dependent

netlist

2-level Logic opt

multilevel Logic opt

Library

Timing Constraints
Another approach: Fluid Library

Basic synthesis methodology remains the same except:
- fixed library replaced by cell generation on-the-fly
- Synthesis and optimization could be further enhanced to exploit more complex functionality - avoid technology decomposition

Opportunities for Improvement -1a

Sizing:
- resize transistors on critical paths to speed up circuit
- Ideally each transistor individually sized
- Independent sizes for P/Pull-up, N/Pull-down transistors
Opportunities for Improvement -1b

Reducing levels of logic through complex functions
- reduces area and sometimes reduces speed
  - fewer cells, less overhead due to guard bands and signal wires
  - more complex cells may be slower

Fluid Library: Approach 2

Basic synthesis methodology remains the same except:
- fixed library replaced by cell generation on-the-fly
- Synthesis and optimization could be further enhanced to exploit more complex functionality - avoid technology decomposition
- Physical design incorporates a post-pass peephole optimization step that further exploits the library
Opportunities for Improvement - 2a

Post pass optimization to maximize diffusion sharing
- reduce area
- reduced area decreases wire lengths and increases speed a little as a result

Opportunities for Improvement - 2b

Post pass optimization to improve porosity (Sechen)
- negligible increase in cell area
- significant reduction in routing congestion, faster and smaller circuits
Opportunities for Improvement - 2c

Post pass optimization window to remove guard banding of signals to nearby (replace by “unsafe” cell):

 Fluid Library: Approach 3

Basic synthesis methodology remains the same except:
- fixed library replaced by cell generation on-the-fly
- Synthesis and optimization could be further enhanced to exploit more complex functionality - avoid technology decomposition
- Cell-based place and route is replaced by a custom 2-D layout generator
Lecture Summary

Increasing the speed of the circuit has many dimensions
Clock/memory element related
• Reduce skew/positive skew
• Faster-FF’s, Latch-based design
Combinational network
• Lower-Vth, Higher Vdd
• Speed up combinational portion
  – Tech independent:
    • Refactor/restructure
    • Generalized by-pass transform
  – Tech dependent
    • Load-dependent mapping shows interesting extension to dynamic programming
    • Local optimizations
• Library improvements/liquid library

Status on Tech Mapping/Libraries

Technology independent/dependent formulation still working
Tech Independent
  – All approaches discussed are used, generalized by-pass said to be an important technique in FPGA optimization
Covering based approaches to mapping still at core of many systems
  – Load dependent “binning” technique not used – sizing done in a post-pass
Dream of ``libraryless design”, ``liquid libraries”, ``fluid libraries” has existed for 16 years
  – Used at IBM – Burns’ C5
  – Used on DEC/Alpha – Cleo
  – Zenasis is the closest to making commercially available
#define arrival time of a signal $s$ denoted $A_s$ is the time at which the signal settles to its steady state value.

The required time of a signal $s$ denoted $R_s$ is the time at which the signal is required to be stable.

The slack of a signal $s$ $S_s = R_s - A_s$
### Required Times and Slack Times

**Required Times:** Given required times on primary outputs

- Traverse in reverse topological order (i.e., from primary outputs to primary inputs)

- If \((k_i, k)\) is an edge between \(k_i\) and \(k\),
  \[ R_{k_i,k} = R_k - D_k \]

- Hence, the required time of output of node \(k\) is
  \[ R_k = \min\{R_{k,k_j} | k_j \in \text{fanout}(k)\} \]

---

### Delay Analysis

**Simple model 2:**

\[ A_k = \max\{A_1 + D_{k_1}, A_2 + D_{k_2}, A_3 + D_{k_3}\} \]

Can also have different times for rise time delay and fall time delay.
**Delay Tracing**

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

![Diagram of Delay Tracing]

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.

![Diagram of Critical Section/Critical Subgraph]

Only slacks are shown
Generalized Bypass Transformation

New Approach:
- Based on depth ≠ delay
- Find True Critical Paths
- Find Cutset S of partial paths covering critical paths
- Construct bypasses around partial paths
- Very little work on this; mostly carry-bypass adders
- New idea for speeding up circuits: instead of making paths short: make them false

Reducing Optimizing Threshold Voltage

\[ t_d = K \cdot \frac{C_L \cdot V_{DD}}{\beta \cdot (0.9 \cdot V_{DD} - V_T)^q} \]

Parameters:
- \( \beta = I_0 \cdot \left( \frac{e}{\alpha \cdot N_s} \right)^\alpha \)
- \( N_s = \frac{n \cdot k \cdot T}{q} \)
- \( I_0 : I_{DS} \) at \( V_{GS} = V_T \)
- \( n : \) subthreshold slope factor
- \( K : \) delay coefficient

[models: K. Nose, T. Sakurai]
Dual Voltages: A harder problem

Layout synthesis with dual voltages: major geometric constraints

- H -- High Voltage Block
- L -- Low Voltage Block

Cell Library with Dual Power Rails

Identifying Transform Candidates

- Consider some long path $P = f_0, ..., f_n$.
- Suppose the delay to each side input $y_{ik}$ 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).