ESE535:
Electronic Design Automation

Day 16: March 23, 2009
Covering

Previously

- How to optimize two-level logic
  - Clear cost model of Product-Terms
- Generic optimizations for multi-level logic
  - Common-sub expressions
  - Kernel extraction
  - Node decomposition
  - Speedup
- Have been abstract about cost of gates

Covering Problem

- Implement a "gate-level" netlist in terms of some library of primitives
  - Potentially netlist came from multi-level optimization we did last week
- General Formulation
  - Make it easy to change technology
  - Make it easy to experiment with library requirements
    - Evaluate benefits of new cells...
    - Evaluate architecture with different primitives

Input

- netlist
- library
- represent both in normal form:
  - nand gate
  - inverters

Elements of a library - 1

<table>
<thead>
<tr>
<th>Element/Area Cost</th>
<th>Tree Representation (normal form)</th>
</tr>
</thead>
<tbody>
<tr>
<td>INVERTER</td>
<td><img src="image" alt="Inverter" /></td>
</tr>
<tr>
<td>NAND2</td>
<td><img src="image" alt="NAND2" /></td>
</tr>
<tr>
<td>NAND3</td>
<td><img src="image" alt="NAND3" /></td>
</tr>
<tr>
<td>NAND4</td>
<td><img src="image" alt="NAND4" /></td>
</tr>
</tbody>
</table>

Example: Keutzer

Elements of a library - 2

<table>
<thead>
<tr>
<th>Element/Area Cost</th>
<th>Tree Representation (normal form)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AOI21</td>
<td><img src="image" alt="AOI21" /></td>
</tr>
<tr>
<td>AOI22</td>
<td><img src="image" alt="AOI22" /></td>
</tr>
</tbody>
</table>
Input Circuit Netlist

``subject DAG``

Problem statement
Find an "optimal" (in area, delay, power) mapping of this circuit (DAG)

Some things to watch today
- Cost model
- Problem can be solved well
  - somewhat clever solution
- Technique: general and powerful
- Special cases
  - harder/easier cases
- What makes hard
- Bounding

What's the problem? Trivial Covering

Cost Model: Area
- **Assume**: Area in gates
- or, at least, can pick an area/gate
  - so proportional to gates
- e.g.
  - Standard Cell design
  - Standard Cell/route over cell
  - Gate array
Standard Cell Area

Cost Model: Delay

- Delay in gates
  - at least assignable to gates
    - Twire << Tgate
    - Twire ~constant
  - delay exclusively/predominantly in gates
    - Gates have Cout, Cin
    - lump capacitance for output drive
    - delay ~ Tgate + fanout\cdot Cin
    - Cwire << Cin
    - or Cwire can lump with Cout/Tgate
Cost Models

• Why do I show you models?
  – not clear there’s one “right” model
  – changes over time
  – you’re going to encounter many different kinds of problems
  – want you to see formulations so can critique and develop own
  – simple cost models make problems tractable
    • are surprisingly adequate
  – simple, at least, help bound solutions
  – may be wrong today…need to rethink

Greedy work?

• Greedy = pick next locally “best” choice

Greedy In→Out

Greedy Out→In
Greedy Out $\rightarrow$ In

But...

Greedy Problem
- What happens in the future (elsewhere in circuit) will determine what should be done at this point in the circuit.
- Can't just pick best thing for now and be done.

Brute force?
- Pick a node (output)
- Consider
  - all possible gates which may cover that node
  - branch on all inputs after cover
  - pick least cost node

Pick a Node

Brute force?
- Pick a node (output)
- Consider
  - all possible gates which may cover that node
  - recurse on all inputs after cover
  - pick least cost node
- Explore all possible covers
  - can find optimum
Analyze brute force?

- Time?
  \[ T_{\text{brute}}(\text{node}) = \max_{p \in P} \left( T_{\text{match}}(p) + \max_{n \in N} \left( T_{\text{brute}}(n) \right) \right) \]
- Say \( P \) patterns, constant time to match each
  - (if patterns long could be \( \geq O(1) \))
- \( P \)-way branch at each node...
  - ...exponential
    - \( O(P^{\text{depth}}) \)

Structure inherent in problem to exploit?

- There are only \( N \) unique nodes to cover!

Structure inherent in problem to exploit?

- Structure
  - If subtree solutions do not depend on what happens outside of its subtree
    - separate tree
    - farther up tree
  - Should only have to look at \( N \) nodes.
  - Time(\( N \)) = \( N \cdot P \cdot T(\text{match}) \)
    - w/ \( P \) fixed/bounded \( \Rightarrow \) linear in \( N \)
    - w/ cleverness work isn’t \( P \cdot T(\text{match}) \) at every node

Idea Re-iterated

- Work from inputs
- Optimal solution to subproblem is contained in optimal, global solution
- Find optimal cover for each node
- Optimal cover:
  - examine all gates at this node
  - look at cost of gate and its inputs
  - pick least

Work front-to-back
Work Example (area)

\[ 3 + 3 + 2 = 8 \]
3 + 5 = 8
8 + 2 + 3 = 13
Work Example (area)

\[ 3 + 2 + 4 = 9 \]

\[ 9 + 4 + 3 = 16 \]

\[ 8 + 2 + 4 + 4 = 18 \]
Work Example (area)

13 + 5 + 4 = 22

18 + 3 = 21

9 + 4 + 4 = 17
Work Example (area)

Optimal Cover

Note

• There are nodes we cover which will not appear in final solution.
Dynamic Programming Solution
• Solution described is general instance of dynamic programming
• Require:
  – optimal solution to subproblems is optimal solution to whole problem
  – (all optimal solutions equally good)
  – divide-and-conquer gets same (finite/small) number of subproblems
• Same technique used for instruction selection

Delay
• Similar
  – Cost(node) = Delay(gate)+Max(Delay(input))

DAG
• DAG = Directed Acyclic Graph
  – Distinguish from tree (tree ⊂ DAG)
  – Distinguish from cyclic Graph
  – DAG ⊂ Directed Graph (digraph)

Trees vs. DAGs
• Optimal for trees
  – why?
  • Area
  • Delay

Not optimal for DAGs
• Why?

1+1+1=3

Not optimal for DAGs
• Why?

1+1+1=3
Not optimal for DAGs

- Why?

1 + 1 + 1 = 3

Not Optimal for DAGs (area)

- Cost(N) = Cost(gate) + Σ Cost(input nodes)

- think of sets
- cost is magnitude of set union
- Problem: minimum cost (magnitude) solution isn’t necessarily the best pick
  - get interaction between subproblems
  - subproblem optimum not global...

Not Optimal for DAGs

- Delay:
  - in fanout model, depends on problem you haven’t already solved (delay of node depends on number of uses)

What do people do?

- Cut DAGs at fanout nodes
- optimally solve resulting trees

- Area
  - guarantees covered once
  - get accurate costs in covering trees, made “premature” assignment of nodes to trees

- Delay
  - know where fanout is

Bounding

- Tree solution give bounds (esp. for delay)
  - single path, optimal covering for delay
  - (also make tree by replicating nodes at fanout points)

- no fanout cost give bounds
  - know you can’t do better

- delay bounds useful, too
  - know what you’re giving up for area
  - when delay matters

(Multiple Objectives?)

- Like to say, get delay, then area
  - won’t get minimum area for that delay
  - algorithm only keep best delay
  - …but best delay on off critical path piece not matter
  - …could have accepted more delay there
  - don’t know if on critical path while building subtree
  - (iterate, keep multiple solutions)
Many more details...

- Implement well
- Combine criteria
  - (touch on some later)
- ...see literature

Admin

- No class Wednesday
  - Work on problems 1 and 2 of assignment
- Reading for next Monday online
  - Flowmap → classic FPGA-mapping paper

Big Ideas

- simple cost models
- problem formulation
- identifying structure in the problem
- special structure
- characteristics that make problems hard
- bounding solutions