Problem

- Implement a "gate-level" netlist in terms of some library of primitives
- General
  - easy to change technology
  - 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>![INVERTER Diagram]</td>
</tr>
<tr>
<td>NAND2</td>
<td>![NAND2 Diagram]</td>
</tr>
<tr>
<td>NAND3</td>
<td>![NAND3 Diagram]</td>
</tr>
<tr>
<td>NAND4</td>
<td>![NAND4 Diagram]</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>![AOI21 Diagram]</td>
</tr>
<tr>
<td>AOI22</td>
<td>![AOI22 Diagram]</td>
</tr>
</tbody>
</table>

Input Circuit Netlist

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

into this library

**Why covering now?**
- Nice/simple cost model
- problem can be solved well
  - somewhat clever solution
- general/powerful technique
- show off special cases
  - harder/easier cases
- show off things that make hard
- show off bounding

**What's the problem? Trivial Covering**
subject DAG

7 NAND2 (3) = 21
5 INV (2) = 10
Area cost 31

**Cost Models**

**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**

- inv, nand3, inv, AOI, nor3, inv
- All cells uniform height
- Width of channel determined by routing
- Width of channel fairly constant?
Cost Model: Delay

- Delay in gates
  - at least assignable to gates
    - $T_{wire} \ll T_{gate}$
    - $T_{wire} \sim$constant
  - delay exclusively/predominantly in gates
    - Gates have Cout, Cin
    - lump capacitance for output drive
    - delay $\sim T_{gate} + \text{fanout} \cdot \text{Cin}$
    - $C_{wire} \ll \text{Cin}$
    - or $C_{wire}$ can lump with $C_{out}/T_{gate}$

Logic Delay

Parasitic Capacitances

Delay of Net

Cost Model: Delay

- Delay in gates
  - at least assignable to gates
    - $T_{wire} \ll T_{gate}$
    - $T_{wire} \sim$constant
  - delay exclusively/predominantly in gates
    - Gates have Cout, Cin
    - lump capacitance for output drive
    - delay $\sim T_{gate} + \text{fanout} \cdot \text{Cin}$
    - $C_{wire} \ll \text{Cin}$
    - or $C_{wire}$ can lump with $C_{out}/T_{gate}$

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
Approaches

Greedy work?

- Greedy = pick next locally "best" choice

Greedy In → Out

Greedy Out → In

Greedy In → Out

Greedy Out → In
But…

\[
\begin{array}{ccc}
4 & 2 & 4 \\
\end{array}
\]

= 10

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?
  \[ T_{\text{brute}}(\text{node}) = \sum_{i=0}^{\max \text{ pattern}} \left( T_{\text{match}}(i) \right) + \sum_{j=0}^{\max \text{ in j}} \left( T_{\text{brute}}(i, j) \right) \]
- Say P patterns, constant time to match each
  - (if patterns long could be \( \geq O(1) \))
- P-way branch at each node…
- …exponential
  - \( O(\langle P \rangle \text{depth}) \)
Structure inherent in problem to exploit?

- There are only N unique nodes to cover!

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*P*T(match)
  - w/ P fixed/bounded \( \Rightarrow \) linear in N
  - w/ cleverness work isn’t P*T(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)
Work Example (area)

\[3+3+2 = 8\]

Work Example (area)

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

\[3 + 5 = 8\]

Work Example (area)

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

13 + 2 = 15

3 + 2 + 4 = 9

9 + 4 + 3 = 16

8 + 2 + 4 = 18
Work Example (area)

\[16 + 2 = 18\]

Work Example (area)

\[13 + 4 = 22\]

Work Example (area)

\[18 + 3 = 21\]

Work Example (area)

\[9 + 4 + 4 = 17\]

Work Example (area)

\[8 + 2 + 4 + 5 = 19\]
Work Example (area)

Optimal Cover

Note

- There are nodes we cover which will not appear in final solution.

“Unused” Nodes

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
  - 1 + 1 + 1 = 3
  - 3 + 3 + 1 = 7?
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
  - (put some refs on web)
Admin

• Reminder: Reading for Monday
  – Flowmap → classic FPGA-mapping paper
  – (will mail pointers this afternoon)
• Assignment 1 out today

Big Ideas

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