Binary Decision Diagrams
Static Timing Analysis
Announcements

- Start early on Lab 1 (CORDIC design)
  - Fixed-point design should not have usage of DSP48s

Course survey

<table>
<thead>
<tr>
<th>Question</th>
<th>Unfamiliar</th>
<th>Comfortable</th>
<th>Experienced</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digital design with Verilog or VHDL</td>
<td>15.79%</td>
<td>25.58%</td>
<td>30.77%</td>
</tr>
<tr>
<td>C/C++ programming</td>
<td>10.53%</td>
<td>20.93%</td>
<td>42.31%</td>
</tr>
<tr>
<td>Python programming</td>
<td>10.53%</td>
<td>32.56%</td>
<td>23.08%</td>
</tr>
<tr>
<td>Combinatorial optimization algorithms</td>
<td>63.16%</td>
<td>20.93%</td>
<td>3.85%</td>
</tr>
</tbody>
</table>
Review: Quantization for Fixed-Point Types

- Write down the value of ‘y’ in decimal after the assignment for each of the following cases

(1) \[\text{ap} \_ \text{fixed}<4, 2> \ x = -0.25; \]
\[\text{ap} \_ \text{fixed}<3, 1, \text{AP} \_ \text{TRN}, \text{AP} \_ \text{WRAP}> \ y = x; \]

(2) \[\text{ap} \_ \text{fixed}<4, 2> \ x = 0.25; \]
\[\text{ap} \_ \text{fixed}<3, 3, \text{AP} \_ \text{TRN}> \ y = x; \]
Review: Graph Connectivity

▸ Paths
  - A **path** is a any sequence of edges that connect two vertices
  - A **simple path** never goes through any vertex more than once

▸ Connectivity
  - A graph is **connected** if there is a path between any two vertices
  - Any subgraph that is connected can be referred to as a **connected component**
  - A directed graph is **strongly connected** if there is always a directed path between vertices
Trees and DAGs

- A **cycle** is a path starting and ending at the same vertex. A cycle in which no vertex is repeated other than the starting vertex is said to be a **simple cycle**

- An undirected graph with no cycles is a **tree** if it is connected, or a **forest** otherwise
  - A **directed tree** is a directed graph which would be a tree if the directions on the edges were ignored

- A directed graph with no directed cycles is said to be a **directed acyclic graph (DAG)**
Examples

Directed graphs with cycles

Directed acyclic graph (DAG)

Tree
A topological sort (or order) of a directed graph is an ordering of nodes where all edges go from an earlier vertex (left) to a later vertex (right)
- Feasible if and only if the subject graph is a DAG
Agenda

- Graph algorithms applied to two EDA problems
  - Binary decision diagrams
  - Static timing analysis

[source: Alberto Sangiovanni-Vincentelli’s keynote at ICCAD’2012]
Example: Boolean Voting Function

- A Boolean voting function
  - An $n$-ary Boolean function $f(x_1, x_2, \ldots, x_n)$ evaluates to 1 if 50% or more (≥[$n/2$]) of its inputs are set to 1
  - Examples:
    - $f(0,0) = 0$
    - $f(0,1) = 1$
    - $f(0,0,1) = 0$
    - $f(1,0,1) = 1$

- How to formally represent this function?
  - Truth table
  - Karnaugh map
  - Sum of Products (SOP)
  ...
Ideal Representation of a Boolean Function

- We hope to find a representation with the following characteristics
  - **Compact** (in terms of size)
  - **Efficient** to compute the output with the given inputs and efficient to manipulate and modify
  - **Ideally**, a **canonical** representation
    - A unique form for equivalent functions under certain conditions
<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Truth table is canonical
But $2^n$ table entries are required!

Canonical SOP (4 minterms):
$xyz' + xy'z + xyz + x'yz$
What about \( n \) inputs? (esp. where \( n \) is large)

Minimized SOP (3 terms): \( xy + xz + yz \)

Note: K-map only handles up to 6 inputs
# Boolean Voting Function: Exponential Growth Rate of SOP

- An n-input voting function has at least $C(n, n/2)$ prime implicants

- **Growth rate of** $C(n, k)$ **in terms of** $n$
  - For $k=1$, $C(n,1) = n$
  - For $k=2$, $C(n,2) = n(n-1)/2$
  - For $k=3$, $C(n,3) = n(n-1)(n-2)/6$
  - …
  - For $k=n/2$, $C(n, n/2) = \frac{n!}{[(n/2)!]^2} \in \Theta(2^n n^{-0.5})$ (involves Stirling formula)
An Alternative: Binary Decision Diagrams

“One of the only really fundamental data structures that came out in the last twenty-five years” – Donald Knuth, 2008
Truth Table, Shannon Expansion, and Decision Tree

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
<th>z</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>x=1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ f = x' \cdot f_{x=0} + x \cdot f_{x=1} \]

- Nonterminal node in orange
  - Follow dashed line for value 0
  - Follow solid line for value 1
- Terminal (leaf) node in green
  - Function value determined by leaf values
Reduction Rule #1

- Merge equivalent leaves

![Diagram showing the reduction rule with binary tree examples]
Reduction Rule #2

- Remove redundant tests
  - For a node $v$, $\text{left}(v) = \text{right}(v)$
Reduction Rule #3

- Merge isomorphic nodes
The Reduction Algorithm

- Recursively determine an identifier (id) for each node, starting from leaves; nodes with the same IDs are merged
  - If id(left(v)) = id(right(v)), then id(v) = id(left(v))
  - If id(left(v)) = id(left(u)) and id(right(v)) = id(right(u)), then id(v) = id(u)
BDD Construction

- BDDs are usually directly constructed bottom up, avoiding the reduction step

- One approach is using a hash table called unique table, which contains the IDs of the Boolean functions whose BDDs have been constructed
  - A new function is added if its associated ID is not already in the unique table
BDDs History

- Proposed by Lee 1959, Akers 1976
  - Idea of representing Boolean function as a rooted DAG with a decision at each vertex

- Popularized by Bryant 1986
  - Further restrictions + efficient algorithms to make a useful data structure (ROBDD)
  - \( \text{BDD} = \text{ROBDD} \text{ since then} \)
ROBDDs

- Reduced and Ordered (ROBDD)
  - Directed acyclic graph (DAG)
    - Two children per node
    - Two terminals 0, 1
  - Ordered:
    - Co-factoring variables (splitting variables) always follow the same order along all paths $x_1 < x_2 < x_3 < \ldots < x_n$
  - Reduced:
    - Any node with two identical children is removed (rule #2)
    - Two nodes with isomorphic BDDs are merged (rules #1 and #3)

3-input voting function in BDD form
More on Variable Ordering

- Follow a total ordering to variables
  - e.g., $x < y < z$
- Variables must appear in the same ascending order along all paths

![Correct Ordering Diagram]

![Incorrect Ordering Diagram]
Canonical Representation

- ROBDD is a canonical representation of Boolean functions
  - Given the same variable order, two functions equivalent if and only if they have the same BDD form
    - “0” unique unsatisfiable function
    - “1” unique tautology
Compactness

- 8-input voting function in BDD with 20 nonterminal vertices
- In contrast to 70 prime implicants in SOP form
There are many, but to list a few more:

- Can represent an exponential number of paths with a DAG

- Can evaluate an $n$-ary Boolean function in at most $n$ steps
  - By tracing paths to the 1 node, we can count or enumerate all solutions to equation $f = 1$

- Every BDD node (not just root) represent some Boolean function in a canonical way
  - A BDD can be multi-rooted representing multiple Boolean functions sharing subgraphs
BDD Limitations

- NP-hard problem to construct the optimal order for a given BDD
- No efficient BDD exists for some functions regardless of the order
- Existing heuristics work reasonably well on many combinational functions from real circuits
  - Lots of research in ordering algorithms

Same function, two different orderings, different graphs
Static Timing Analysis

- In circuit graphs, **static timing analysis** (STA) refers to the problem of finding the delays from the input pins of the circuit (esp. nodes) to each gate
  - In sequential circuits, flip-flop (FF) input acts as output pin, FF output acts as input pin
  - Max delay of the output pins determines clock period
  - **Critical path** is a path with max delay among all paths

- Two important terms
  - **Required time**: The time that the data signal needs to arrive at certain endpoint on a path to ensure the timing is met
  - **Arrival time**: The time that the data signal actually arrives at certain endpoint on a path
STA: Arrival Times

- Assumptions
  - All inputs arrive at time 0
  - All gate delays = 1ns (di = 1); all wire delays = 0

- Questions: **Arrival time** (AT) of each gate output? Minimum clock period?

\[ AT_i = \max_{j \in \text{pred}(i)} \{ AT_j \} + d_i \]

Gates are visited in a topological order
STA: Required Times

- Assumptions
  - All inputs arrive at time 0
  - All gate delays = 1ns (di = 1); all wire delays = 0
  - Clock period = 5ns (200MHz frequency)

- Question: **Required time** (RT) of each gate output in order to meet the clock period?

\[
RT_i = \min_{j \in \text{succ}(i)} \{RT_j - d_j\}
\]
More on Static Timing Analysis

- In addition to the arrival time and required time of each node, we are interested in knowing the **slack** \((= RT - AT)\) of each node / edge
  - Negative slacks indicate unsatisfied timing constraints
  - Positive slacks often present opportunities for additional (area/power) optimization
  - Node on the **critical path** have zero slacks
STA: Slacks

- **Assumptions:**
  - All inputs arrive at time 0
  - All gate delays = 1ns, wire delay = 0
  - Clock period = 5ns

- **Question:** What is the maximum slowdown of each gate without violating timing?

\[
\text{Slack}_i = \text{RT}_i - \text{AT}_i
\]
Summary

- Graph algorithms are applicable to a wide range of EDA problems
  - Neatly capture the circuit topology
  - DAG is an important class of directed graph and will be used frequently in this class
Before Next Class

- Start early on CORDIC design!
- Next lecture: Front-end compilation and CDFG
Acknowledgements

- These slides contain/adapt materials from / developed by
  - Prof. Randal Bryant (CMU)