Additional Slides to De Micheli Book

Sungho Kang

Yonsei University
Design Style - Decomposition

• Behavioral Synthesis
  - Resource allocation; Pipelining; Control flow parallelization; Communicating Sequential Processes; Partitioning

• Sequential Synthesis
  - Register Movement and Retiming; State Minimization; State Assignment; Synthesis for Testable FSM’s; State Machine Verification

• Logic Synthesis
  - Extraction of combinational logic to HDL; Two-level minimization; Algebraic Decomposition; Multilevel Logic Minimization; Synthesis for Multifault Testability; Test Generation via Minimization; Technology mapping; Timing optimization

• Technology Mapping
  - Mapping to Library of Logic Gates; Timing Optimization

• Physical Design Synthesis
  - Cell Placement; Routing; Fabrication; Engineering Changes
Area vs Delay: The Bit-Serial Adder

- A typical tradeoff is area versus delay
- With just one full adder, this circuit can do 32-bit addition
- But, it is 32x slower than a parallel adder (32 full adders, 1 bit output per clock cycle)
Design Tradeoff Curve

- Holding other factors constant, the Area vs Delay tradeoff curve is typically parabolic.
- The first design requirement is to meet **Constraints** on Chip Area and Critical Path Delay (0 to 1).
• The next priority is to optimize a feasible design
• Design 2 is optimal, in the sense that area and delay cannot both be decreased from this point
• Tradeoff is now necessary, according to Policy
A typical design **Policy** is to optimize area subject to a delay constraint (2 to 3).

Often a preferred policy is to optimize delay subject to an area constraint (2 to 4).
Prioritizing Testability

- Sometimes other factors, such as testability or power, take priority
- Typically this moves the area-delay tradeoff curve up and to the right
- Thus designs 1 and 2 are optimal
Area Optimization

- Typically performed in a *technology independent view* of the circuit
- In this view gates are regarded as logic functions
- These functions are converted to physical gates by *Technology Mapping*
**Technology Independent View**

In this view the gates of the full adder circuit are just logic equations:

- \( a = x_i \ y_i \)
- \( b = x'_i \ y'_i \)
- \( e = a' \ b' \)
- \( z_i = ec_{i-1} + e'c_{i-1} \)
- \( c = x_i \ y_i \)
- \( d = x_i + y + i \)
- \( f = dc_{i-1} \)
- \( c_i = c + f \)

• In this view the gates of the full adder circuit are just logic equations.
- Common subfunctions shared
- Functions “Technology Mapped” to negative gates
Testing

• Faults Models
  - Stuck-at faults
  - Delay faults

• Test vectors
  - Fault simulation
  - Automatic test pattern generation

• Diagnosis

• Testable Design
Delay Optimization

First step is to identify the Critical Path

Simplest delay model: “number of logic levels”
Critical Path Analyzers

- **Static Delay Models:**
  - Levels of Logic
  - Delay function of size, load
  - Worst, best case models

- **Dynamic Delay Models**
  - Simplified device models
  - Full Spice analysis
VLSI: Scalable Algorithms

- In the 70s IBM found that 75% of all its cpu cycles went to critical path algorithms
- Now the bottleneck is Formal Verification
- Scalable algorithms required, for which cpu time and space increase:
  - linearly in problem size $n$ ($O(n)$)
  - log-linearly ($O(n\log(n))$)
  - even quadratic ($O(n^2)$) too expensive
Graph Models and FSM

- Graph
- Edge
- Vertex
- Undirected graph
- Digraph
  - All edges are directed
- Mixed graph
- DAG (directed acyclic graph)
Graph Terminology

Graph: ordered set of two sets

\[ G = (V, E) \]

- \( V \): a set of vertices or nodes
- \( E \): a set of edges or arcs

\( \rightarrow 4 \): the successors (fanouts) of node 4

\[ 4 \rightarrow = \{1,3\} \]

\( \leftarrow 4 \): the predecessors (fanins) of node 4

\[ 4 \leftarrow = \{1,2\} \]
Graph Models

- Transitive closure
- fanout
- fanin
- Source
  - v has no predecessors
- Sink
  - v has no successors
### Products of Sets of Sets

- **Intersecting 2 sets of sets**

**Procedure SET_CARTESIAN_PRODUCT(G,H)**

<table>
<thead>
<tr>
<th>Ops</th>
<th>Times/call</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best</td>
<td>Worst</td>
</tr>
<tr>
<td>c1</td>
<td>1</td>
</tr>
<tr>
<td>c2</td>
<td>1</td>
</tr>
<tr>
<td>c3m</td>
<td>1</td>
</tr>
<tr>
<td>c4n</td>
<td>m</td>
</tr>
<tr>
<td>c5</td>
<td>mn</td>
</tr>
<tr>
<td>c6</td>
<td>1</td>
</tr>
</tbody>
</table>

\[
m = |G|;\ n = |H|;\]

\[
P = \text{NULL}
\]

\[
\text{for (i = 1, 2, ..., m) }
\]

\[
\text{\quad for (j = 1, 2, ..., n) }
\]

\[
\text{\quad \quad P = P } \cup (Gi \cap Hj)
\]

\[
\text{\quad }
\]

\[
\text{\quad }
\]

\[
\text{return (P)}
\]

\[
\]

\[
\]
Computing Critical Path Length

• This problem is modeled as that of finding the longest path in a DAG (Directed Acyclic Graph)
• Thus we digress for a while, and introduce the notions of sets and graphs
• Then we discuss a scalable (linear) complexity algorithm for finding the longest path in a DAG
Longest Paths

```
Procedure LONGEST_PATH(V, E, L, I, spec) {
    n = |V|; m = |E|; g = |I|
    for(v ∈ V) {
        λ(v) = 0
        D_v = |→v|
    }
    Q = I
    while(Q ≠ ∅) {
        v = DEQUEUE(Q)
        foreach(a ∈ v→) {
            λ_a = max(λ_a, (λ_v + L_v,a))
            D_a = D_a - 1
            if(D_a = 0) QUEUE(Q, a)
        }
        λ* = max_{v∈V} {λ_v}
        v* = SELECT1(V, λ*)
        π = BACK_TRACE(V, E, L, λ, v*, (spec − λ*))
        return(π, λ)
    }
}
```
Backtracking

- The slack of an edge \((a,v)\) is the slack of \(v\) plus the difference between the length of the longest path to \(v\) and the longest path to \(v\) through \((a,v)\).
- In formula, \(\text{slack}_{a,v} = \text{slack}_v + (\lambda_v - (\lambda_a + L_{a,v}))\).
- Here \(\lambda_v\) is the length of the longest path to \(v\), and \((\lambda_a + L_{a,v})\) is the length of the longest path to \(v\) that passes through the edge \((a,v)\).
- The slack of a node \(u\) is defined as the minimum of its fanout edge slacks, so \(\text{slack}_a = \min \text{slack}_v\).
Asymptotic Complexity

- A function $F(n)$ is in the set $O(g(n))$ if and only if there exist positive constants $c_0$ and $n_0$ such that $F(n) \leq c_0 g(n)$ for all $n \geq n_0$.

- This means that $F(n)$ is asymptotically bounded from above by a linear function of $g(n)$.

- A function $F(n)$ is in the set $\Omega(g(n))$ if and only if there exist positive constants $c_\Omega$ and $n_\Omega$ such that $F(n) \geq c_\Omega g(n)$ for all $n \geq n_\Omega$.

- This means that $F(n)$ is asymptotically bounded from below by a linear function of $g(n)$.