CAD for VLSI Design - I

Lecture 21

V. Kamakoti and Shankar Balachandran
Overview of this Lecture

• Understanding the process of Logic synthesis
• Logic Synthesis of HDL constructs
Logic Synthesis

• What is this?
  – Process of converting a high-level description of the design into an optimized gate-level representation given a standard-cell library and certain design constraints.

• Standard Library or Technology Library
  – It can have basic logic gates (cells) like *and*, *or* etc.
  – It can have macro cells like *adders*, *muxes* and *flip flops*
Objectives

• Computer Aided logic synthesis tools have greatly reduced the design cycle time and improved productivity.

• Designers write technology-independent, highly abstract high-level descriptions and produce technology-dependent, optimized gate-level netlists.

• Both combinational and sequential RTL descriptions can be synthesized.
Evolution

• The designer’s mind was the synthesis tool.
• Design constraints remain the same – such as *timing*, *area*, *testability* and *power*.
• Manual iterations were done before the gate-level netlist met the design constraints
• Focus was to generate optimized netlists manually
Architectural Description

Partitioning into Higher level blocks

Designer’s mind

Gate-level Representation

Meets Design Constraints

yes

Optimized Gate-level Representation

no

Design Constraints

Standard Cell Library (Technology dependent)
CAD Synthesis Tools

• Shift focus to
  – Architectural trade-offs
  – High-level description of the design
  – Accurate design constraints
  – Optimization of the cells in the standard cell library

• Iterations and generation of the optimized gate-level netlist is done by the Synthesis tool.
Architectural Description

High-Level Description

Computer-Aided Logic Synthesis

Optimal Gate-level Netlist

Meets Design Constraints

yes

no

Design Constraints

Standard Cell Library (Technology dependent)

Place and Route
The Impact

• Before Automation
  – Prone to human errors, especially larger designs
  – No foolproof method of verifying design constraints at an early stage
  – If design constraints not met, then huge time for this back annotation
  – If timing has to be improved, say from 20ns to 15ns, requires total redesign (What-if scenarios)
  – Design reuse was not possible – technology specific.
The Impact

• After Automation
  – Designs described like higher level of abstraction and hence less prone to errors.
  – High-level design can be done caring less for design constraints
  – Design reuse, solving What-if scenarios, and in general turnaround time for redesign/back annotation are significant benefits to the designer.
Verilog HDL Synthesis

• Synthesizable Constructs
  – In general, any construct that is used to define a *cycle-by-cycle RTL description* is acceptable to the logic synthesis tool.

• Most synthesis tools can convert designs specified using a combination of data flow and specific behavioral constructs.

• Behavioral synthesis tools that convert a behavioral description into an RTL description are slowly evolving.
## Synthesisable Verilog Constructs

<table>
<thead>
<tr>
<th>Construct Type</th>
<th>Keyword or Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ports</td>
<td>Input, inout, output</td>
<td></td>
</tr>
<tr>
<td>parameters</td>
<td>parameter</td>
<td></td>
</tr>
<tr>
<td>Module definition</td>
<td>module</td>
<td></td>
</tr>
<tr>
<td>Signals and variables</td>
<td>wire, reg, tri</td>
<td>Vectors are allowed</td>
</tr>
</tbody>
</table>
## Synthesizable Verilog Constructs

<table>
<thead>
<tr>
<th>Construct Type</th>
<th>Keyword or Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>instantiation</td>
<td>Module instances</td>
<td>Eg. my_mux m1(out,i0,i1,s);</td>
</tr>
<tr>
<td></td>
<td>Primitive gate instances</td>
<td></td>
</tr>
<tr>
<td>Functions and tasks</td>
<td>function, task</td>
<td>Timing constructs ignored</td>
</tr>
<tr>
<td>procedural</td>
<td>always, if, then, else, case, casex, casez</td>
<td>initial is not supported</td>
</tr>
</tbody>
</table>
# Synthesisable Verilog Constructs

<table>
<thead>
<tr>
<th>Construct Type</th>
<th>Keyword or Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Procedural blocks</td>
<td>begin, end, named blocks, disable</td>
<td>Disabling of named blocks not allowed</td>
</tr>
<tr>
<td>Data flow</td>
<td>assign</td>
<td>Delay information is ignored</td>
</tr>
<tr>
<td>loops</td>
<td>for, while, forever</td>
<td>while and forever loops must contain @(posedge clk) or @(negedge clk)</td>
</tr>
</tbody>
</table>
Synthesisable Verilog Operators

• All arithmetic, logical, relational, equality (except ===, !==), bit-wise, reduction, shift, concatenation and conditional are synthesisable.
References

• Chapter 14 – Samir Palnitkar – Verilog HDL.
Questions and Answers

Thank You