Chapter-14

Samir Palnitkar
Logic Synthesis

The 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.
Basic computer-aided logic synthesis process

1. Architectureal Description
2. High-level Description
3. Computer-Aided Logic Synthesis
4. Optimized Gate-level Netlist
5. Meets Constraints
   - no
   - yes
5.1. Place and Route
5.2. Standard Cell Library (technology Dependent)
Limitations without Logic synthesis

► For large designs, manual conversion was prone to human errors. A small gate missed somewhere could mean redesign of entire blocks.

► The designer could never be sure that the design constraints were going to be met until the gate-level implementation was completed and tested.

► If the gate-level design did not meet requirements, the turnaround time for redesign of blocks was very high.

► Each designer would implement the blocks differently. There was little consistency in design styles.
Limitations (continue…)

► Timing, area and power dissipation in library cells are fabrication-technology specific. Thus if IC fabrication vendor changed after the gate-level design was complete, this would mean redesign of entire circuit and a possible change in design methodology.

► Design reuse was not possible. Designs were technology specific, hard to port and very difficult to reuse.
Logic synthesis flow from RTL to gates

- RTL Description
- Translation
- Unoptimized Intermediate Representation
- Logic Optimization
- Technology Mapping and Optimization
- Optimized gate-level representation
- Library of available gates (technology library)

Design Constraints
### Verilog HDL constructs for logic synthesis

<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>
<tr>
<td>instantiation</td>
<td>Module instances, primitive</td>
<td>E.g., mymux m1(out,i0,i1,s);</td>
</tr>
<tr>
<td></td>
<td>gate instances</td>
<td>e.g., nand (out,a,b);</td>
</tr>
<tr>
<td>functions and tasks</td>
<td>function, task</td>
<td>Timing constructs ignored</td>
</tr>
</tbody>
</table>
Verilog HDL constructs for logic synthesis (continue....)

<table>
<thead>
<tr>
<th>Construct type</th>
<th>Keyword or description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>procedural</td>
<td>always, if, then, else, case, casex, casez</td>
<td>initial is not supported</td>
</tr>
<tr>
<td>procedural blocks</td>
<td>begin, end, named blocks, disable</td>
<td>Disabling of named blocks 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>
The designer describes the design at a high level by using RTL constructs.
The designer spends time in functional verification to ensure that the RTL description functions correctly.
After the functionality is verified, the RTL description is input to the logic synthesis tool.
The RTL description is converted by the logic synthesis tool to an unoptimized, intermediate, internal representation.

Design constraints such as area, timing and power are not considered in the translation process.
The translation process yields an unoptimized intermediate representation of the design. The design is represented internally by the logic synthesis tool in terms of internal data structures.
The logic is now optimized to remove redundant logic. Various technology independent Boolean logic optimization techniques are used.

It is a very important step in logic synthesis, and it yields an optimized internal representation of the design.
Technology mapping and optimization

► Until this step, the design description is independent of a specific target technology.

► In this step the synthesis tool takes the internal representation and implements the representation in gates, using the cells provided in the technology library.

► The design is *mapped* to the desired *target technology*. 
Technology library

- The technology library contains library cells.
- Physical layout of library cells is done first. Then the area of each cell is computed from the cell layout. Then the area of each cell is computed from the cell layout. This process is called cell characterization.
- The cell description contains information about the following:
  - Functionality of the cell
  - Area of the cell layout
  - Timing information about the cell
  - Power information about the cell
Design constraints

- **Timing** → The circuit must meet certain timing requirements. An internal static timing analyzer checks timing.
- **Area** → The area of the final layout must not exceed a limit.
- **Power** → The power dissipation in the circuit must not exceed a threshold.
Optimized gate-level description

► After the technology mapping is complete, an optimized gate-level netlist described in terms of target technology components is produced.

► If this netlist meets the required constraints, it is handed out for final layout. Otherwise, the designer modifies the RTL or reconstrains the design to achieve the desired results.
Modeling tips for logic synthesis

► Use meaningful names for signals and variables
► Avoid mixing positive and negative edge-triggered flip-flops
► Use basic building blocks instead of continuous assign statements
  ▪ The final structure is not always symmetrical.
  ▪ Basic building blocks creates symmetric designs.
Modeling tips (Continue..)

► Instantiate multiplexers instead if-else or case statements

► Use parentheses to optimize logic structure
  ▪ Out = a + b + c + d;  //3 adders in series
  ▪ Out = (a + b) + (c + d);
    //2 adders in series and one final adder to sum results

► Use arithmetic operators *, /, and % instead design building blocks
Modeling tips (continue..)

- Be careful with multiple assignments to the same variable

- Define if-else or case statements explicitly
  - if(control)
    - out <= a; //latch is inferred, incomplete specification
  - if(control)
    - out <= a;
  - else
    - out <= b; //complete specification for all values of control
Horizontal Partitioning

- Use bit slices to give the logic synthesis tool a smaller block to optimize. This is called horizontal partitioning.
- It reduces complexity of the problem and produces more optimal results for each block.
- e.g. Instead of directly designing a 16-bit ALU, design a 4-bit ALU and build the 16-bit ALU with four 4-bit ALUs.

Thus the logic synthesis tool has to optimize only the 4-bit ALU, which is a smaller problem than optimizing the 16-bit ALU.
Design Partitioning (continue..)

- **Vertical Partitioning**
  - Vertical partitioning implies that the functionality of a block is divided into smaller submodules.
  - Different from horizontal partitioning
    - In horizontal partitioning all blocks do the same function
    - In vertical partitioning all blocks do different function
  - e.g. Divide the ALU module in *add*, *subtract*, *shift right*, *shift left* blocks.
Parallelizing design structure

► In this technique we use more resources to produce faster designs.

► We convert *sequential* operations into *parallel* operations by using more logic.
  - *E.g.* *Carry look ahead adder*...