Chapter 1
VLSI Design Methods

Jin-Fu Li
Advanced Reliable Systems (ARES) Laboratory
Department of Electrical Engineering
National Central University
Jhongli, Taiwan
Outline

- Introduction
- VLSI Design Flows & Design Verification
- VLSI Design Styles
- System-on-Chip Design Methodology
Complexity & Productivity Growth of ICs

- Complexity grows 58%/yr (doubles every 18 mos)
- Productivity grows 21%/yr (doubles every $3^{1/2}$ yrs) unless methodology is updated

[Source: MITRE]
VLSI Design Methodologies

- Design methodology
  - Process for creating a design

- Methodology goals
  - Design cycle
  - Complexity
  - Performance
  - Reuse
  - Reliability
IC Community

[Diagram showing the relationship between Silicon foundry, IC design, CAD tool provider, and software tools.]

[Text: Design rules, Simulation models and parameters, Mask layouts, Integrated circuits, Process information, Software tools]

[M. M. Vai, VLSI design]
System to Silicon Design

System Requirements

Algorithm
\[ X[k] = \sum x[n]e^{\frac{2\pi jk}{N}} \]
\[ x[n] = \sum X[k]e^{\frac{2\pi jk}{N}} \]

Hardware Architecture

Synthesis

System Integration

Fabricate and Test

Physical

Design For Test

[Source: MITRE]
Y-Chart
VLSI Design Flow

Concept

Designer

Behavior Specification

Behavior Synthesis

RTL Design

Behavior Synthesis

RTL Verification

Logic Synthesis

Logic Verification

Final Product

Manufacturing

Layout (Masks)

Layout Verification

Netlist (Logic Gates)

Layout Synthesis

Product Verification

Advanced Reliable Systems (ARES) Lab. Jin-Fu Li, EE, NCU
Behavioral Synthesis & RTL Synthesis

Behavioral Synthesis

\[ z = a(i) \times b(i) - c \times d(k) + f \]

- Vary clock period
- Vary # clock cycles

Multiple Architectures

- 4 cycles = 16 ns
- 3 cycles = 15 ns
- 2 cycles = 20 ns

RTL Synthesis

- Vary clock period
- 1 clock cycle

Single Architecture

1 cycle = 50 ns

Source: Synopsys
Behavioral Synthesis (Resource Allocation)

- Source code defines the functionality
  
  \[
  y_{\text{real}} := a_{\text{real}} \times b_{\text{real}} - a_{\text{imag}} \times b_{\text{imag}} \\
  y_{\text{imag}} := a_{\text{real}} \times b_{\text{imag}} + a_{\text{imag}} \times b_{\text{real}}
  \]

- Constraints allow designer to explore different architectures, trading off speed vs. area

- Two possible implementations for a complex multiplier:

  - Fast, but large
    (4 mult, 1 add, 1 sub)

  - Small, but slow
    (1 mult, 1 add/sub)

[Source: MITRE]
Behavioral Synthesis (Retiming)

- Allows designer to trade off latency for throughput by adding and moving registers in order to meet timing constraints.
- Specification needs to include registers at *functional* boundaries, without regard to register-to-register timing: software takes care of optimizing register placement.

Compiled Functional Description

Max. Speed = $\frac{1}{23.0\ \text{ns}} = 43\ \text{MHz}$

Latency = 1 clock cycle

Circuit After Behavioral Retiming

Max. Speed = $\frac{1}{10\ \text{ns}} = 100\ \text{MHz}$

Latency = 3 clock cycles

[Source: MITRE]
Verification

- The four representations of the design
  - Behavioral, RTL, gate level, and layout
- In mapping the design from one phase to another, it is likely that some errors are produced
  - Caused by the CAD tools or human mishandling of the tools
- Usually, simulation is used for verification, although more recently, formal verification has been gaining in importance
- Two types of simulations are used to verify the design
  - Functional simulation & timing simulation
DFT Flow

Behavioral Description → Behavioral DFT Synthesis → RTL Description → Logic DFT Synthesis → Gate Description → Test Pattern Generation → Fault Coverage?

Gate → Technology Mapping → Layout → Parameter Extraction → Manufacturing → Product → Test Application → Good Product

Low → High
Design Styles – *Full Custom*

![Logic Diagram]

Advanced Reliable Systems (ARES) Lab.  Jin-Fu Li, EE, NCU
Design Styles – Programmable Logic Array

AND array

OR array

Buffering

Inputs

Outputs

Advanced Reliable Systems (ARES) Lab.  Jin-Fu Li, EE, NCU
Design Styles – Gate Array
Design Styles – Field Programmable Gate Array (FPGA)

- Xilinx SRAM-based FPGA
Design Styles – Field Programmable Gate Array (FPGA)

Xilinx XC4000 Configurable logic block
Design Styles – *Field Programmable Gate Array (FPGA)*

- SRAM-based programmable switch
Design Styles – Field Programmable Gate Array (FPGA)

- Architecture of Altera FLEX 10K FPGA

Diagram showing the layout of an Altera FLEX 10K FPGA, including Embedded Array Block (EAB), Logic Array Block (LAB), I/O, and FastTrack Interconnect.
Design Styles – Field Programmable Gate Array (FPGA)

Logic array block
Standard-Cell Design Styles

- Design entry

  Enter the design into an ASIC design system, either using a *hardware description language* (HDL) or *schematic entry*

- An example of Verilog HDL

```verilog
module fadder(sum, cout, a, b, ci);
output sum, cout;
input a, b, ci;
reg sum, cout;

always @(a or b or ci) begin
    sum = a^b^ci;
    cout = (a&b)|(b&ci)|(ci&a);
end
endmodule
```
<table>
<thead>
<tr>
<th>Design Styles</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full-custom</td>
<td>- Compact designs;</td>
<td>- Very time consuming;</td>
</tr>
<tr>
<td></td>
<td>- Improved electrical characteristics;</td>
<td>- More error prone;</td>
</tr>
<tr>
<td>Semi-custom</td>
<td>- Well-tested standard cells which can be shared between users;</td>
<td>- Can be time consuming to built-up standard cells;</td>
</tr>
<tr>
<td></td>
<td>- Good for bottom-up design;</td>
<td>- Expensive in the short term but cheaper in long-term costs;</td>
</tr>
<tr>
<td>FPGA</td>
<td>- Fast implementation;</td>
<td>- Can be wasteful of space and pin connections;</td>
</tr>
<tr>
<td></td>
<td>- Easy updates;</td>
<td>- Relatively expensive in large volumes;</td>
</tr>
</tbody>
</table>
Emergence of SOC Idea

- **Motivation:**
  - Transistor density
  - Moor's law
- **Integration with analog parts**
  - AMS specification, synthesis, simulation
- **SoC Design Methodology and Tools**
  - Filling the Gap through
    - Reuse
    - Design Automation
    - System Specification Methodology

Source: Synopsys
What’s a System?

[Source: M. Gudarzi]
What’s a System?

☐ Customer’s view:

\[ \text{System} = \text{User/Customer-specified}\ \textit{functionality} + \textit{requirements} \text{ in terms of: Cost, Speed, Power, Dimensions, Weight, …} \]

☐ Designer’s view:

\[ \text{System} = \textit{HW} \text{ components} + \textit{SW} \text{ modules} \]

[Source: M. Gudarzi]
Hierarchical Design Flow for an System Chip

Specifications

Hardware Architecture

Detail Design

Integration

Test

Requirements

Architecture

Hardware Design

Software Design

Integration

System test

Test

Software Architecture

Module Design

Integration
HDL’s & SDL’s: Requirements

- HDL’s
  - HardwareC
  - Verilog
  - AHDL
  - VHDL

- SDL’s
  - C
  - Pascal
  - ADA

[Source: M. Gudarzi]
HDL’s & SDL’s: Realization

[Source: M. Gudarzi]
HDL’s & SDL’s: Features

Any *SW-realizable algorithm* is *HW-realizable* as well.

- **Hardware Realization**
  - Speed
  - Energy Efficiency
  - Cost Efficiency (in high volumes)

- **Software Realization**
  - Flexibility
  - Ease of Development
  - Ease of Test and Debug
  - Cost = SW + Processor

[Source: M. Gudarzi]
HW-SW Co-design

☐ How much SW + how much HW?

☐ Objectives:
  ■ Power
  ■ Speed
  ■ Area
  ■ Memory space
  ■ Time-to-market

☐ Implementation platform:
  ■ Collection of chips on a board (MCM)
  ■ …
HW/SW Co-Design Methodology

- Must architect hardware and software together:
  - provide sufficient resources;
  - avoid software bottlenecks.
- Can build pieces somewhat independently, but integration is major step.
- Also requires bottom-up feedback
HW/SW Co-design Main Topics

System

Synthesis

Specification

Verification

[Source: M. Gudarzi]
Co-Synthesis

System Specification

Partitioning

HW Parameter Estimation

SW Parameter Estimation

HW Synthesis

SW Synthesis

ASIC

OS

EXE Code

System Integration

Verification

Verification

Verification

Verification

Final Verification

Automated, Under Designer Control

Designers Task
SOC Design Essentials

- **Realization strategy:**
  - Automated HW-SW Co-design + Reusable Cores
  - Intellectual Property: IP Cores

- **IP Core Examples:**
  - Processors: PowerPC, 680x0, ARM, …
  - Controllers: PCI, …
  - DSP Processors: TI
  - …

- **IP Core Categories:**
  - Soft Cores: HDL, SW/HW Cores
  - Firm Cores: Synthesized HDL
  - Hard Cores: Layout for a specific fabrication process
Trends and Challenges of SOC Designs

- Technology trends
  - 3D technology + System-in-package

- Architecture trends
  - Regular architectures, e.g., multi-core architecture
  - Network-on-chip communication

- Challenges
  - Power
  - Reliability
  - Yield
  - Design-for-manufacturability
  - …