Logic CAD of VLSI (046880)

Dr. Eitan Zahavi – Mellanox Technologies
Spring semester 2016-2017
Lecture 1: Introduction
Logic Design Automation for VLSI 046880

Today’s topics:
- Why you should take this course?
- Course outline
- Administration
- History of the field
- Basic concepts in system design
- Principles in handling complexity
Course Objectives

- Provide students with knowledge and insights to become a better Digital VLSI designer or CAD tools developer
- Describe the role of Logic CAD tools in VLSI design
- Explain some major problems and their solutions
  - Design complexity / productivity
  - Correctness validation / verification
  - Optimizations
- Learn design-representations and data-structures
- Learn algorithmic and heuristic approaches used in CAD
- Note the follow-up course “Physical CAD of VLSI”
Course Outline

- **Introduction to Logic VLSI CAD**
  why CAD? Past/future trends; principles; examples – 1 class

- **Digital hardware modeling**
  structure and behavior; logic networks; connectivity data-structures; 1 class

- **Dynamic verification (logic simulators)**
  logic values and timing; how simulators are built; simulation algorithms; simulation environments - 2 classes

- **Logic Modeling and Representations**
  structure and operation of the BDD data-structure; logic and state-machine equivalence checking, SAT – 3 classes

- **Logic synthesis**
  2-level and multilevel minimization, exact vs. heuristic methods, cell-library mapping – 2 classes

- **Static timing analysis**
  delay modeling; critical path analysis; false path elimination; time borrowing; interconnect effects; - 2 classes
Course Requirements

- **Prerequisites**
  
  ( (044268 Intro. to algorithms and data structures) || (234246/7 Graph Algorithms 1) )

- **Preferably**
  
  ( (046237 Intro. to VLSI) || (236354 VLSI Circuit Design) ) &&
  
  ( (044101 Intro. to Software Systems) || (234122 Intro. to Systems Programming))

  Or instructor's consent.

- **Practically**
  
  Coding in C++ over STL in Linux environment

- **Homework: 50%**
  
  - **Wet: 40% 3 out of 6 (some are mandatory)**
    
    - HCM Data Structure – Optional
    - Event Driven Logic OR Compiled Code Logic Simulator – One of these is compulsory
    - BDD OR SAT based Formal Equivalence Checker – One of these is compulsory
    - Markup details can be obtained on request
  
  - **Dry: 10% 3 out of 5**

- **Final exam: 50%**
Why Computer-Aided Design?

- Moore’s law: exponential growth in complexity

The infamous “Design Productivity Gap”
VLSI Development Process

- Do it right the first time!
  - Iterations increase cost and effort

- Overall trend:
  - From ART to ENGINEERING DISCIPLINE

- Issues:
  - Complexity, correctness, optimization, productivity

We focus on DESIGN
Challenges of VLSI CAD

- Capacity of designs
- Span of styles
- A complex optimization target:
  - Optimize given tradeoffs
Evolution of VLSI CAD

- 1960’s: (Prehistory): Mask design on mylar
- 1970’s: Calma mask digitizer; Design Rule Checkers (DRC) Circuit and logic simulators Online layout editors
- 1980’s: Workstations; RTL simulation Standard cell design; “Silicon compilation” - module generators
- 1990’s: Logic synthesis, static timing, formal verification, test generation
- 2000’s: Tools for h/w-s/w co-design, dynamic environments formal model checking, physical synthesis, OCV timing
- 2010’s: 2.5D and 3D layout, 3D Partitioning, Floor-planning and Synthesis?
Observations about Past Trends

- Whenever *evolution* happens, the result is… A ZOO!

- **Bottom-up** evolution along the design path

- **Verification first**, synthesis later

- Each step actually caused a revolution in design methods, step-function in productivity

- Computing capabilities REALLY changed over these years!
  - Lesser the last decade unless you go parallel
VLSI Technology Trends

Data collected by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, C. Batten
Current Nanometer Design Issues:

- **System level requirements:**
  - $>10^9$ transistors on chip
  - System-On-Chip: Re-use, co-design of h/w and s/w
  - Logic Complexity vs. Quality
  - Higher level abstractions
  - Accelerated development cycle

- **Physical effects at the circuit level:**
  - High-frequency & low-voltage/low-power circuit design
  - Interconnect design becomes critical
  - Complicated design rules and reliability requirements
  - In-die variation, statistical design
  - Need redundancy/error-correction/self repair?
Principles of Dealing with Complexity

- **Abstraction**: show relevant features without associated details
- **Hierarchy**: “divide and conquer”
  - How hierarchy helps abstraction? Encapsulation, Modularity
- **Regularity**
  - How regularity fits with hierarchy? Instantiation
  - How regularity reduces complexity? Re-use
- **Design Methodology**
  - What is methodology? What is design?
    - Answers will come next….
      - But of course it includes Automation!

What is gained and what is lost when adopting these principles?
What is *Methodology*?

- Without methodology, you have free-form design
  - In how many different ways one can connect 100M transistors? (S/D/G)

- Methodology is

  *A set of rules about what you can or can’t do, and guidelines about how to do things.*

  - With methodology, engineering becomes a *discipline*
  - Predefined decisions result in saved time and effort
  - Self-imposed restrictions helps prevent problems, saves checking
  - It’s the basis of automation
  - Engineers are often resistant about methodology (NIH)
  - Methodology must be enforced (how?)
Examples of *Methodology*

- Gate arrays (channel-based, sea-of-gates,....)
- Standard-cell based design
- Synchronous design
- Logic synthesis from RTL model to library cells
- Naming conventions. E.g. power nets are VDD*
What is Design?

- VLSI Design is:

  The correct refinement of the highest level (architectural) specification of a device into implementation details (masks)

1) Architecture  2) Micro Arch

3) RTL  4) Gates  5) Circuit  6) Layout

module HA (A, B, S, Co);
input A, B;
output S, Co;
and2 a (.A(A), .B(B), .Y(S));
xor2 c (.A(A), .B(B), .Y(Co));
endmodule // HA
A Persistent State Checker

Example of the 4 Principles

- Consider a modern VLSI design > $10^9$ transistors
- In low power designs some supplies may be turned off occasionally
- A state element is defined as the following structure:

Need an automatic tool to check that all state elements are connected to always-ON VDD (never turned off)
The Naïve Implementation

- Given:
  - Node names of the Always ON supplies {PVDD}
  - The design

- Algorithm:
  - Flatten the design to transistors
  - Use well known “Sub-Graph Isomorphism” algorithm to find all state circuits
  - Check that all P1 and P2 connect to one of the given {PVDD} set

- Advantages
  - Straight forward approach - easy to show why correct

- Disadvantages
  - Complexity and runtime
    - NP-Complete see later
Using the 4 Principles

- **Methodology:**
  - Cell based design: each state circuit is in a “Flat” cell
  - Standard Cell design: Use only a well known set of cells

- **Abstraction:**
  - Sub Graph Isomorphism only on cells with transistors (no flattening)
  - Then place a property on specific cell ports – NeedAlwaysOnVDD

- **Hierarchy:**
  - Design is hierarchical so enable using Cells and Reuse
  - Propagate NeedAlwaysOnVDD property up

- **Reuse:**
  - Since cells are re-used as much as possible there is a finite “small” number of cells to run Sub Graph Isomorphism on
  - Propagation up of the NeedAlwaysOnVDD is O(Unique Hierarchy Tree Edges)
Productivity calls for Many Levels of Abstraction

<table>
<thead>
<tr>
<th>Representation of a uP</th>
<th># of such Entities</th>
<th>Manual Entry Productivity</th>
<th>Effort Weeks</th>
</tr>
</thead>
<tbody>
<tr>
<td>High level model</td>
<td>~50k lines</td>
<td>~5k lines/week</td>
<td>10</td>
</tr>
<tr>
<td>Register Transfer Level (RTL)</td>
<td>~500k lines</td>
<td>~5k lines/week</td>
<td>100</td>
</tr>
<tr>
<td>Gates</td>
<td>~5M</td>
<td>~1000 gates/week</td>
<td>5000</td>
</tr>
<tr>
<td>Transistors</td>
<td>~50M</td>
<td>~100 transistors/week</td>
<td>100000</td>
</tr>
<tr>
<td>Polygons</td>
<td>~500M</td>
<td>~500 polygons/week</td>
<td>1000000</td>
</tr>
</tbody>
</table>

When you design at a low level, you lose twice: more mistakes, more work
What is Refined?

Gajski’s Y-diagram

Structure
- units
- blocks
- cells
- gates
- transistors

Behaviors
- abstract
- processes
- instruction set
- register transfer
- State transitions
- Boolean values
- waveforms

Behavior specification

Physical
- Sized polygons
- lines
- mask layers
- topology/symbolic power
- cell location
- chip plan
- package

circuit

layout

logic
Two types of processes:
- Implementation = Refinement e.g. Synthesis RTL to Gates
- Verification e.g. Formal Equivalence Check (FEV)
The “principle” VLSI Design Flow

- The refinement of specifications through a sequence of design representations

It does not work perfectly in reality. There are more loops!
Type of Problems: P, NP, NP-Hard and NP-Complete

- P are decision problems ("yes/no") solved in *polynomial* time
- NP (co-NP) are decision problems that if a proof exists for "Yes" ("No") it can be verified in *polynomial* time
  - E.g. "SAT": Given boolean formula F is there assignment that F is true?
    - If the answer is "Yes" – proof is variables assignment can be check it by substitution – so it is NP
- NP-Hard IFF a polynomial time to solve it means a polynomial time algorithm exists for every NP problem
  - Cook and Levin proved that SAT is NP-hard (~1970)
- NP-Complete: A problem that is *both* NP-hard and is NP (co-NP)
  - Eventhough SAT is NP (check the assignment)
  - Cook showed SAT is NP-Hard
  - So SAT is NP-Complete
Types of Problems in CAD

- Mathematically, most CAD tools address combinatorial decision and optimization problems
  - *Optimization problems* aim to find a minimum cost solution
    - e.g. Find a minimum delay logic implementation of a function
    - Can be mapped to a decision problem: “is there a better solution?”

- Most are *intractable*: NP-hard or NP-complete
  - Exact algorithms are of exponential complexity or higher
  - Must use *heuristics* (approximation algorithms) to get inexact but practical solutions, using reasonable computer time and memory
Methodology to the Help!

- Methodologies are key to reducing CAD complexity
  - Limiting the design space
  - Tools can take “short cut” and assumptions

- As we have seen
Summary

- Goals of CAD:
  - Handle complexity, optimize tradeoffs

- Evolution and trends

- Principles for handling complexity:
  - Hierarchy, regularity, abstraction, methodology

- Design process:
  - Specify, implement, check
  - Design representations
  - Successive refinements

- Examples: Check state is preserved during power-down