ESE535: Electronic Design Automation

Day 1: January 9, 2013
Introduction

Complete questionnaire

Warmup Poll

• How many of you have:
  – Drawn geometry for transistors and wires
  – Sized transistors
  – Placed logic and/or memory cells
  – Selected the individual gates
  – Specified the bit encoding for an FSM
  – Designed a bit-slice for an Adder or ALU
  – Written RTL Verilog or VHDL
  – Written Behavioral Verilog, VHDL, etc. and compiled to hardware?
  – Written SystemC or Bluespec System Verilog?
  – Compiled C to gates?

Modern Design Challenge

• How do we design modern computational systems?
  – billions of devices
  – used in everything
  – billion dollar businesses
  – rapidly advancing technology
  – more “effects” to address
  – rapidly developing applications and uses
  – short product cycles
  – extreme time-to-market pressures

Productivity Gap

• Human brain power is the bottleneck
  – to producing new designs
  – to creating new things
    • (applications of technology)
  – to making money

The Productivity Gap

Source: ITRS2009 Design Chapter

Bottleneck

Source: Newton (UCB/GSRC)
Avoiding the Bottleneck

- How do we unburden the human?
  - Take details away from him/her
    - raise the level of abstraction at which human specifies computation
  - Pick up the slack
    - machine take over the details

To Design, Implement, Verify 10M transistors

<table>
<thead>
<tr>
<th>Approach</th>
<th>GATES/WEEK (Dataquest)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOMAIN SPECIFIC</td>
<td>8K - 12K</td>
</tr>
<tr>
<td>BEHAVIORAL</td>
<td>2K - 10K</td>
</tr>
<tr>
<td>RTL</td>
<td>1K - 2K</td>
</tr>
<tr>
<td>GATE</td>
<td>100 - 200</td>
</tr>
<tr>
<td>TRANSISTOR</td>
<td>10 - 20</td>
</tr>
</tbody>
</table>

Source: Keutzer (UCB EE 244)

Central Questions

- How do we make the machine fill in the details (elaborate the design)?
- How well can it solve this problem?
- How fast can it solve this problem?

Outline

- Intro/Setup
- Instructor
- The Problem
- Decomposition
- Costs
- Not Solved
- This Class

Instructor

- VLSI/CAD user + Novel Tech. consumer
  - Architect, Computer Designer
  - Spatial designs: FPGAs, Reconfigurable
  - Hybrid: Multicontext FPGAs, P+FPGA
  - Nanoscale: CNT, NW-based, NEMS
  - Avoid tedium (impatient)
- Analyze Architectures
  - necessary to explore
  - costs different (esp. in new technologies)
- Mapping as part of runtime?
  - Variation, wear, reliability
- Requirements of Computation
Problem

• Map from a problem specification down to an efficient implementation on a particular computational substrate.
• What is
  – a specification
  – a substrate
  – have to do during mapping

Problem: Specification

• Recall: basic tenant of CS theory
  – we can specify computations precisely
  – Universal languages/building blocks exist
    • Turing machines
    • nand gates
• EEs:
  – Can build any function out of nand gates
  – Any FSM out of gates + registers

Specifications

• netlist
• logic gates
• FSM
• programming language
  – C, C++, Lisp, Java, block diagram
• DSL (domain specific)
  – MATLAB, Snort
• RTL
  – Register Transfer Level
  – (e.g. subsets of Verilog, VHDL)
• behavioral
• dataflow graph
• layout
• SPICE netlist

Substrate

• “full” custom VLSI
• Standard cell
• metal-only gate-array
• FPGA
• Processor (scalar, VLIW, Vector)
• Array of Processors (SoC, {multi,many}core)
• billiard balls
• Nanowire PLA
• molecules
• DNA

Full Custom

• Get to define all layers
• Use any geometry you like
• Only rules are process design rules
• ESE570

FPGA

K-LUT (typical k=4)
Compute block w/ optional output Flip-Flop

ESE171, CIS371
Standard Cell

Standard Cell Area

All cells uniform height

Width of channel determined by routing

Cell area

Nanowire PLA

What are we throwing away? (what does mapping have to recover?)

- layout
- TR level circuits
- logic gates / netlist
- FSM
- Allocation of functional units and assignment
- Cycle-by-cycle timing
- Operation sequencing
- How task implemented
dSL: MATLAB

Specification not Optimal

- \( Y = a \cdot b \cdot c + a \cdot b / c + /a \cdot b \cdot c \)

- Multiple representations with the same semantics (computational meaning)
- Only have to implement the semantics, not the “unimportant” detail
- Exploit freedom to make smaller / faster / cooler

Problem Revisited

- Map from some “higher” level down to substrate
- Fill in details:
  - device sizing, placement, wiring, circuits, gate or functional-unit mapping, timing, encoding, data movement, scheduling, resource sharing
Decomposition

- Conventionally, decompose into phases:
  - Arch. select, scheduling, assignment -> RTL
  - sequential opt. -> logic equations
  - logic opt., covering -> gates
  - retiming -> gates and registers
  - placement-> placed gates
  - routing->mapped design
- Good abstraction, manage complexity

Easy once decomposed?

- All steps are (in general) NP-hard.
  - routing
  - placement
  - partitioning
  - covering
  - logic optimization
  - scheduling
- What do we do about NP-hard problems?
  - Return to this problem in a few slides...

Costs

- Once get (preserve) semantics, trying to minimize the cost of the implementation.
  - Otherwise this would be trivial
  - (none of the problems would be NP-hard)
- What costs?
- Typically: EDA [:]
  - Energy
  - Delay (worst-case, expected....)
  - Area
- Future
  - Yield
  - Reliability
  - Operational Lifetime

Mapping and Decomposition

- Two important things to get back to
  - disentangling problems
  - coping with NP-hardness

Mapping and Decomposition

- Easy once decomposed?
  - All steps are (in general) NP-hard.
    - Can verify solution in polytime
      - N, N^2, N^{100}
    - Do not know how to find in polytime
      - only known e^N
    - If there were a polytime solution then P=NP
  - What do we do about NP-hard problems?
    - Return to this problem in a few slides...

Costs

- Different cost critera (e.g. E,D,A)
  - behave differently under transformations
  - lead to tradeoffs among them
- [LUT cover example next slide]
- Even have different optimality/hardness
  - e.g. optimally solve delay covering in poly time, but not area mapping
    - e.g. covering
Costs: Area vs. Delay

Example of exploiting freedom of mapping choice.

Costs

- Cannot, generally, solve a problem independent of costs
  - costs define what is “optimal”
  - e.g.
    - (A+B)+C vs. A+(B+C)
    - [cost=pob. Gate output is high]
    - A, B, C independent
      - \(P(A)=P(B)=0.5, P(C)=0.01\)
      - \(P(A)=0.1, P(B)=P(C)=0.5\)

Costs may also simplify problem

- Often one cost dominates
  - Allow/supports decomposition
  - Solve dominant problem/effect first (optimally)
  - Cost of other affects negligible
    - total solution can’t be far from optimal
  - e.g.
    - Delay in gates,
    - Delay in wires
    - Require: formulate problem around relative costs
- Simplify problem at cost of generality

Coping with NP-hard Problems

How do we cope with?

- simpler sub-problem based on dominant cost or special problem structure
- problems exhibit structure
  - optimal solutions found in reasonable time in practice
- approximation algorithms
  - Can get within some bound of optimum
- heuristic solutions
  - high density of good/reasonable solutions?
    - Try many ... filter for good ones
- …makes it a highly experimental discipline

Not a solved problem

Why need to study – not just buy tool from C, M, or S?

- NP-hard problems
  - almost always solved in suboptimal manner
  - or for particular special cases
- decomposed in suboptimal ways
- quality of solution changes as dominant costs change
  - …and relative costs are changing!
- new effects and mapping problems crop up with new architectures, substrates

Big Challenge

- Rich, challenging, exciting space
- Great value
  - practical
  - theoretical
- Worth vigorous study
  - fundamental/academic
  - pragmatic/commercial
This Class: Student Outcomes

- You will learn:
  - Freedom exists in design mappings and how to exploit
  - Formulate & abstract optimization problems
  - How to decompose large problems
  - Techniques for attacking these problems
  - Traditional design objectives (e.g. E,D,A, map time.)
  - Canonical representations for problems
  - Evaluate the quality of a design mapping
  - Implement design automation algorithms

This Class: Technique Toolkit

- Dynamic Programming
- Linear Programming (LP, ILP)
- Graph Algorithms
- Greedy Algorithms
- Randomization
- Search
- Heuristics
- Approximation Algorithms
- SAT

This Class: Decomposition

- Provisioning
- Scheduling
- Logic Optimization
- Covering/gate-mapping
- Partitioning
- Placement
- Routing

Student Requirements

- Reading
- Class
- Projects
  - Will involve programming algorithms
  - Roughly weekly
  - Cumulative build toward an overall mapping goal
  - Choose what you do for final piece
  - Last month

Graduate Class

- Assume you are here to learn
  - Motivated
  - Mature
  - Not just doing minimal to get by and get a grade
- Not plug-in-numbers and get solution
- Things may be underspecified
  - Reason
  - Ask questions
  - State assumptions

Materials

- Reading
  - Online
    - several on blackboard
    - Rest on Xplore, ACM DL, web
      - Linked from syllabus page
    - If online, linked to reading page on web; I assume you will download/print/read.
    - Possible reference texts (on web)
- Lecture slides
  - I’ll try to link to web page by 10am
  - you can print
Today’s Big Ideas

• Human time limiter
• Leverage: raise abstraction+fill in details
• Problems complex (human, machine)
• Decomposition necessary evil (?)
• Implement semantics
  – Exploit freedom to xform to reduce costs
• Dominating effects
• Problem structure
• Optimal solution depend on cost (objective)

Questions?

Administrivia

• Return Info sheets
• Feedback – every lecture – return@end
• Web page
  – http://www.seas.upenn.edu/~ese535/
  – Policies on web page
    • READ THIS (you are responsible for knowing)
  – Syllabus linked off page (reading, assign)
  – Note Piazza group