

EDA (CS286.5b)

Day 1
Introduction

Apple Pie Intro (1)

- How do we design modern computational systems?
 - Millions->billions of devices
 - used in everything
 - billion dollar businesses
 - rapidly advancing technology
 - more “effects” to address
 - rapidly developing applications and uses

Apple Pie Intro (2)

- Options:
 - human handles all the details
 - human solves problem, machine checks
 - human defines something about the solution and machine fills in the details
- Remember:
 - millions of devices, changing world, TTM

Apple Pie Intro (3)

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

Apple Pie Intro (4)

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

Central Questions

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

Outline

- Apple Pie Intro (done)
- Instructor
- The Problem
- Decomposition
- Costs
- Not Solved
- This Class

Instructor

- VLSI/CAD user
- Avoid tedium
- FPGA prototype
- Analyze Architectures
 - necessary to explore
 - costs different
- Requirements of Computation

Problem

- Map from a problem specification down to an efficient implementation on a particular computational substrate.
- What's
 - a specification
 - a substrate
 - have to do during mapping

Problem: Specification

- Recall: basic tenant of CS theory
 - we can specify computations percisely
 - Universal languages/building blocks exist
 - Turing machines
 - nand gates

Specifications

- netlist
- logic gates
- FSM
- programming language
 - C, C++, Lisp, Java
 - block diagram
- RTL
- behavioral
- dataflow graph
- layout
- SPICE netlist

Substrate

- “full” custom VLSI
- metal-only gate-array
- FPGA
- Processor (scalar, VLIW, Vector, MIMD)
- billiard balls
- molecules
- DNA

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

- layout
- TR level circuits
- logic gates
- netlist
- FSM
- block diagram
- dataflow graph
- RTL
- behavioral
- programming language
 - C, C++, Lisp, Java

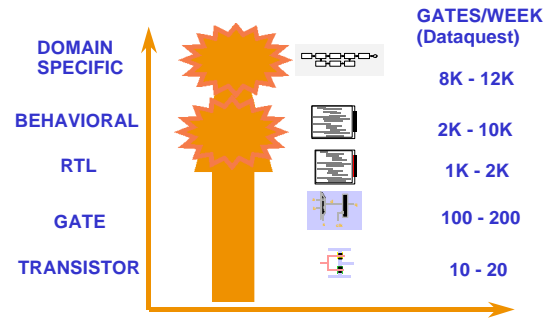
Specification not Optimal

- $Y = a*b*c + a*b*/c + /a*b*c$
- Multiple representations with the same semantics (computational meaning)
- Only have to implement the semantics, not the “unimportant” detail
- Exploit to make smaller/faster/cooler

Problem Revisited

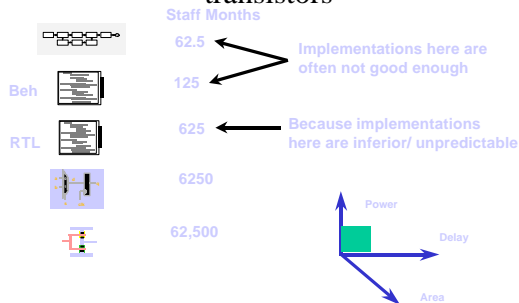
- Map from some “higher” level down to substrate
- Fill in details:
 - device sizing, placement, wiring, circuits, gate or functional-unit mapping, timing, encoding, data movement, scheduling, resource sharing

Design Productivity by Approach



Source: Keutzer (UCB EE 244)

To Design, Implement, Verify 10M transistors



Source: Keutzer (UCB EE 244)

Decomposition

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

Decomposition (easy?)

- All steps are (in general) NP-hard.
 - Routing
 - placement
 - partitioning
 - covering
 - logic optimization
 - scheduling
- (what do we do about NP-hard problems?)

Decomposition

- Easier to solve
 - only worry about one problem at a time
- Less computational work
 - smaller problem size
- Abstraction hides important objectives
 - solving 2 problems optimally in sequence often not give optimal result of simultaneous solution

Mapping and Decomposition

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

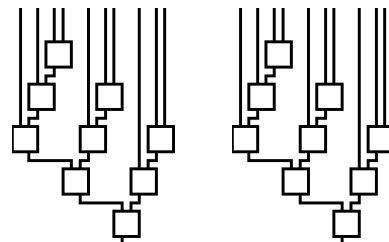
Costs

- 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
 - area

Costs

- Different cost criteria (e.g. EDA)
 - behave differently under transformations
 - lead to tradeoffs among them
 - [DO: LUT cover example]
 - even have different optimality/hardness
 - e.g. optimally solve delay covering in poly time, but not area mapping

Costs: Area vs. Delay



Big Challenge

- Rich, challenging, exciting space
- Great value
 - practical
 - theoretical
- Worth vigorous study
 - fundamental/academic
 - pragmatic/commercial

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 (area) in gates, delay (area) in wires
 - Require: formulate problem around relative costs
- Simplify problem at cost of generality

Coping with NP-hard Problems

- Simpler sub-problem based on dominate cost or special problem structure
- problems exhibit structure
 - optimal solutions found in reasonable time in practice
- approximation algorithms
- heuristic solutions
- high density of good/reasonable solutions?

Not a solved problem

- 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 (relative costs are changing!)
- new effects and mapping problems crop up with new architectures, substrates

This Class

- Toolkit of techniques at our disposal
- Common decomposition and subproblems
- Big ideas that give us leverage
- Formulating problems and analyze success
- Cost formulation

This Class: Toolkit

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

This Class: Decomposition

- Scheduling
- Logic Optimization
- Covering/gate-mapping
- Partitioning
- Placement
- Routing
- Select composition

Student Requirements

- Reading
- Class
- Projects (2)
 - month long, applying ideas from class
 - [ask about computers, access]
- Exam
- Spring Term: major, student-selected project

Misc.

- Web page
- syllabus
- assignment 1
- questionnaire
- lecture note handouts?

Today's Big Ideas:

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