CS137: Electronic Design Automation

Day 1: January 16, 2008

Introduction

Apple Pie Intro (1)

- 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

Apple Pie Intro (2)

- Options:
  - A. human handles all the details
  - B. human solves problem, machine checks
  - C. human defines something about the solution and machine fills in the details
- Remember:
  - billions 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 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 + Novel Tech. consumer
  - Architect, Computer Designer
- Avoid tedium
- Analyze Architectures
  - necessary to explore
  - costs different (esp. in new technologies)
- 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

Specifications

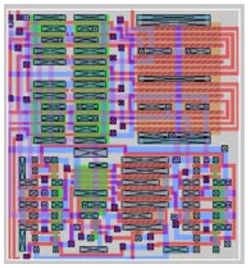
- netlist
- logic gates
- FSM
- programming language
  - C, C++, Lisp, Java, block diagram
- DSL
  - MATLAB, Snort
- RTL
  - Register Transfer Level
  - (*e.g.* subsets of Verilog, VHDL)
- behavioral
- dataflow graph
- layout
- SPICE netlist

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

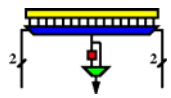
Full Custom

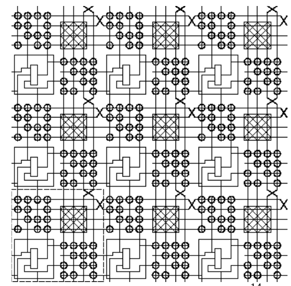
- Get to define all layers
- Use any geometry you like
- Only rules are process design rules



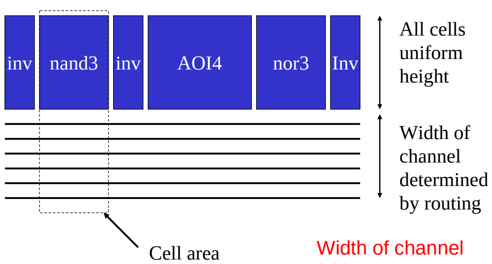
FPGA

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



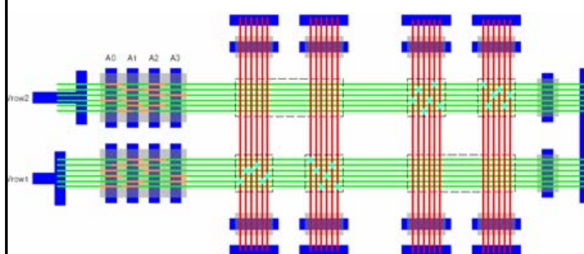


Standard Cell Area



Width of channel fairly constant?

Nanowire PLA



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

- layout
- TR level circuits
- logic gates / netlist
- FSM
- RTL
- behavioral
- programming language
  - C, C++, Lisp, Java
- DSL: MATLAB

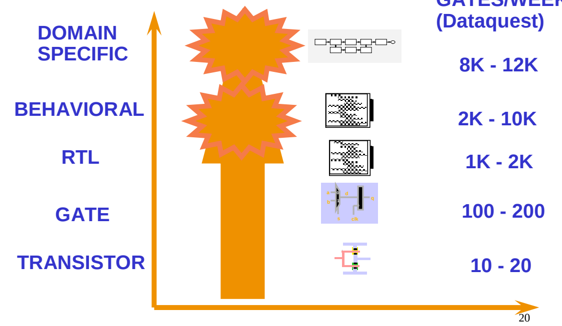
Specification not Optimal

- 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

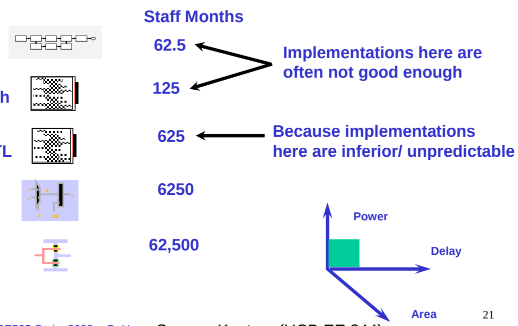
- Map from some “higher” level down to substrate
- Fill in details:
  - device sizing, placement, wiring, circuits, gate or functional-unit mapping, timing, encoding, data movement, scheduling, resource sharing

Design Productivity by Approach



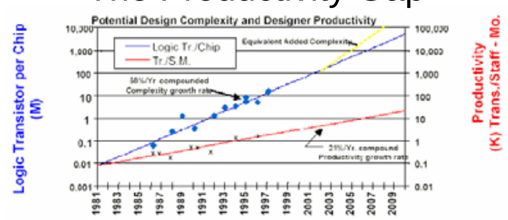
Source: Keutzer (UCB EE 244)

To Design, Implement, Verify 10M transistors



Source: Keutzer (UCB EE 244)

The Productivity Gap



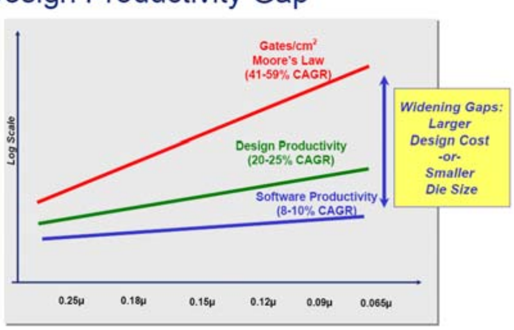
Year	Technology	Chip Complexity	Frequency	3 Yr. Design Staff	Staff Cost*
1997	250 nm	13 M Tr.	400	210	90 M
1998	250 nm	20 M Tr.	500	270	120 M
1999	180 nm	32 M Tr.	600	360	160 M
2002	130 nm	130 M Tr.	800	800	360 M

* @ $150K / Staff Yr. (In 1997 Dollars)

Source: Newton (UCB/GSRC)

PHILIPS

Design Productivity Gap



Source: Payne (CTO Philips Semi) 2004

Decomposition

- Conventionally, decompose into phases:
  - provisioning, 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?
  - Return to this problem in a few slides…

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

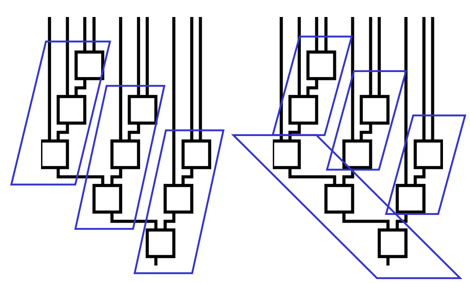
Costs

- 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 (worst-case, expected….)
  - Area
- Future
  - Yield
  - Reliability
  - Operational Lifetime

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
      - (dig into on Day 2)

Costs: Area vs. Delay



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 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

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
  - …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

- 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
- SAT

This Class: Decomposition

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

Student Requirements

- Reading
- Class
- Homework
- Projects
  - Implement algorithm
  - One ~ 2 weeks
  - One ~ 4 weeks
- Essentially something due every 2 weeks

Graduate Class

- Assume you are here to learn
  - Motivated
  - Mature
  - Not just doing minimal to get by and get a grade

Materials

- Reading
  - Mostly online (some handouts)
  - If online, linked to reading page on web; I assume you will download/print/read.
- Lecture slides (after today)
  - I'll try to link to web page by 10am (maybe 9am?); you can print

Misc.

- Web page
  - http://www.seas.upenn.edu/~ese535/
- [make sure get names/emails]
  - Discuss programming experience
  - Discuss optimization problems
    - …goals from experience/research

Questions?

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)