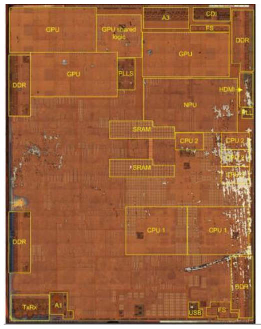
# ESE532:
# System-on-a-Chip Architecture

Day 1: August 29, 2018
Introduction and Overview

## Today

- Case for Programmable SoC
- Goals
- Outcomes
- New/evovling Course, Risks, Tools
- Sample Optimization
- This course (incl. policies, logistics)
- Zed Boards

## Apple A11 Bionic



- 90mm$^2$ 10nm FinFET
- 4.3B transistors
- iPhone 8, 8s, X
- 6 ARM cores (64b)
  - 2 fast (2.4GHz)
  - 4 low energy
- 3 custom GPUs
- Neural Engine
  - 600 Bops?
- Motion, image accel.
- 8MB L2 cache

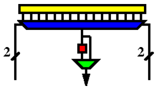
https://wccftech.com/apple-iphone-8-plus-a11-bionic-complete-teardown/

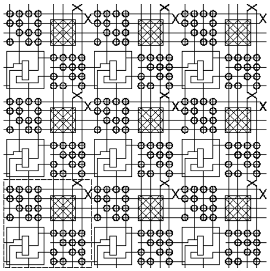
## Questions

- Why do today's SoC look like they do?
- How approach programming modern SoCs?
- How design a custom SoC?
- When building a System-on-a-Chip (SoC)
  - How much area should go into:
    - Processor cores, GPUs, FPGA logic, memory, interconnect, custom functions (which) …. ?

## FPGA
## Field-Programmable Gate Array

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





ESE171, CIS371

## Case for Programmable SoC

## The Way things Were

20 years ago

- Wanted programmability
  - used a processor
- Wanted high-throughput
  - used a custom IC
- Wanted product differentiation
  - Got it at the board level
  - Select which ICs and how wired
- Build a custom IC
  - It was about gates and logic

## Today

- Microprocessor may not be fast enough
  - (but often it is)
  - Or low enough energy
- Time and Cost of a custom IC is too high
  - $100M's of dollars for development, Years
- FPGAs promising
  - But build everything from prog. gates?
- Premium for small part count
  - And avoid chip crossing
  - ICs with Billions of Transistors

## Non-Recurring Engineering (NRE) Costs

- Costs spent up front on development
  - Engineering Design Time
  - Prototypes
  - Mask costs
- Recurring Engineering
  - Costs to produce each chip

$$Cost(N_{chips}) = Cost_{NRE} + N_{chips} \times Cost_{perchip}$$

## NRE Costs

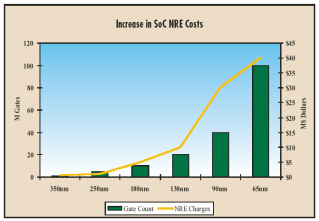


*Figure 1 - NRE costs by process geometry* Source: Semico Research Corp.

- **28-nm SoC development costs doubled over previous node – EE Times 2013**

**28nm+78%, 20nm+48%, 14nm+31%, 10nm+35%**

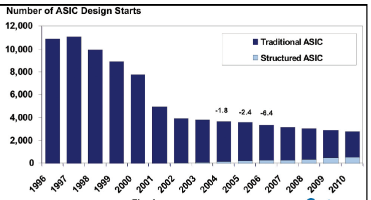
## Amortize NRE with Volume

$$Cost(N_{chips}) = Cost_{NRE} + N_{chips} \times Cost_{perchip}$$

$$Cost = \frac{Cost_{NRE}}{N_{chips}} + Cost_{perchip}$$

## Economics

Forcing fewer, more customizable chips



© 2006 Gartner, Inc. All Rights Reserved. Fig. 1

- Economics force fewer, more customizable chips
  - Mask costs in the millions of dollars
  - Custom IC design NRE 10s—100s of millions of dollars
    - Need market of billions of dollars to recoup investment
    - With fixed or slowly growing total IC industry revenues
    - ➔ Number of unique chips must decrease

## Large ICs

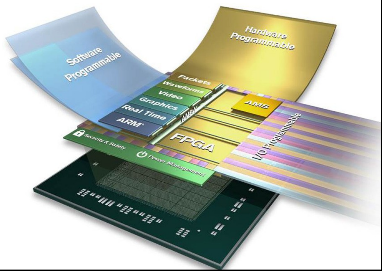
- Now contain significant software
  - Almost all have embedded processors
- Must co-design SW and HW
- Must solve complete computing task
  - Tasks has components with variety of needs
  - Some don’t need custom circuit
  - 90/10 Rule

## Given Demand for Programmable

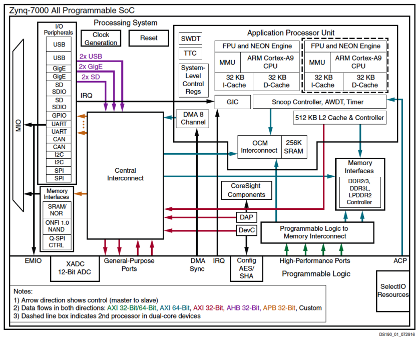
- How do we get higher performance than a processor, while retaining programmability?

## Programmable SoC

- Implementation Platform for innovation
  - This is what you target (avoid NRE)
  - Implementation vehicle



## Programmable SoC



## Then and Now

20 years ago

- Programmability?
  - use a processor
- High-throughput
  - used a custom IC
- Wanted product differentiation
  - board level
  - Select & wired IC
- Build a custom IC
  - It was about gates and logic

Today

- Programmability?
  - uP, FPGA, GPU, PSoC
- High-throughput
  - FPGA, GPU, PSoC, custom
- Wanted product differentiation
  - Program FPGAs, PSoC
- Build a custom IC
  - System and software

## Goals, Outcomes

## Goals

- Create Computer Engineers
  - SW/HW divide is wrong, outdated
  - Parallelism, data movement, resource management, abstractions
  - Cannot build a chip without software
- SoC user – know how to exploit
- SoC designer – architecture space, hw/sw codesign
- Project experience – design and optimization

## Roles

- PhD Qualifier
  - One broad Computer Engineering
- CMPE Concurrency
- Hands-on Project course

## Outcomes

- Design, optimize, and program a modern System-on-a-Chip.
- Analyze, identify bottlenecks, design-space
  - Modeling → write equations to estimate
- Decompose into parallel components
- Characterize and develop real-time solutions
- Implement both hardware and software solutions
- Formulate hardware/software tradeoffs, and perform hardware/software codesign

## Outcomes

- Understand the system on a chip from gates to application software, including:
  - on-chip memories and communication networks, I/O interfacing, design of accelerators, processors, firmware and OS/ infrastructure software.
- Understand and *estimate* key design metrics and requirements including:
  - area, latency, throughput, energy, power, predictability, and reliability.

## New and Evolving Course

- Spring 2017 – first offering
  - Raw, all assignments new … some buggy
  - Assignments too tedious, long
- Fall 2017 – second offering
  - Refine assignments, project
  - Increased explicit modeling emphasis
  - Hard, not insane
- Fall 2018 – now
  - Not much different from 2017
  - Hopefully fewer rough edges

## Tools

- Are complex
- Will be challenging, but good for you to build confidence can understand and master
- Tool runtimes can be long
- Learning and sharing experience will be part of assignments

# Distinction

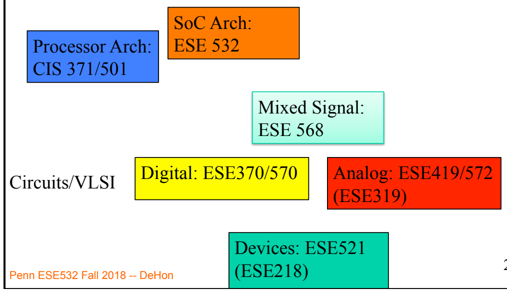
**CIS240, 371, 501**

- Best Effort Computing
  - Run as fast as you can
- Binary compatible
- ISA separation
- Shared memory parallelism

**ESE532**

- Hardware-Software codesign
  - Willing to recompile, maybe rewrite code
  - Define/refine hardware
- Real-Time
  - Guarantee meet deadline
- Non shared-memory models

# Abstraction Stack



Processor Arch: CIS 371/501

SoC Arch: ESE 532

Mixed Signal: ESE 568

Circuits/VLSI

Digital: ESE370/570

Analog: ESE419/572 (ESE319)

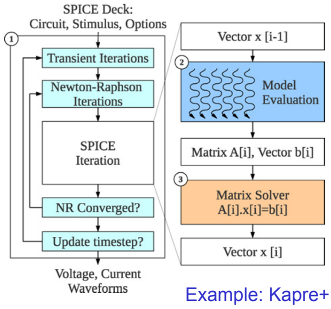
Devices: ESE521 (ESE218)

# Approach -- Example

# Abstract Approach

- Identify requirements, bottlenecks
- Decompose Parallel Opportunities
  - At extreme, how parallel could make it?
  - What forms of parallelism exist?
    - Thread-level, data parallel, instruction-level
- Design space of mapping
  - Choices of where to map, area-time tradeoffs
- Map, analyze, refine
  - Write equations to understand, predict

# SPICE Circuit Simulator



SPICE Deck: Circuit, Stimulus, Options

1 Transient Iterations

Newton-Raphson Iterations

SPICE Iteration

NR Converged?

Update timestep?

Voltage, Current Waveforms

Vector x [i-1]

2 Model Evaluation

Matrix A[i], Vector b[i]

3 Matrix Solver A[i].x[i]=b[i]

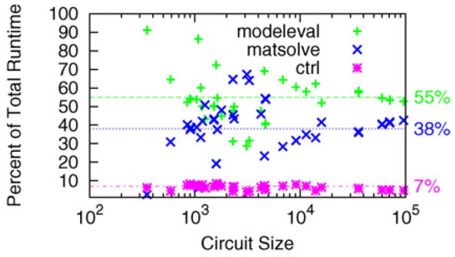
Vector x [i]

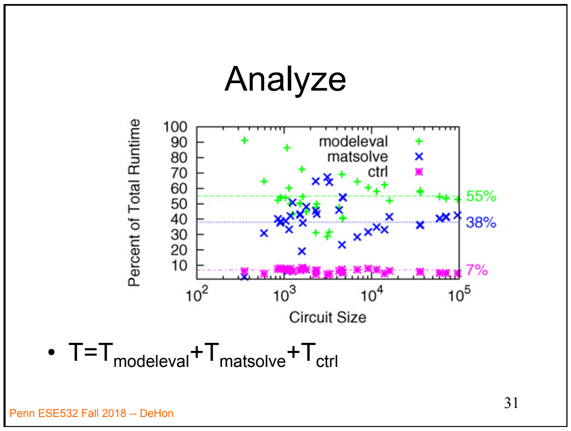
Matrix Solve
Ax=B
A matrix
B vector
x unknown vector
Solve for x

Linear Algebra
solving n eqns
in n unknowns.

Example: Kapre+DeHon, TRCAD 2012

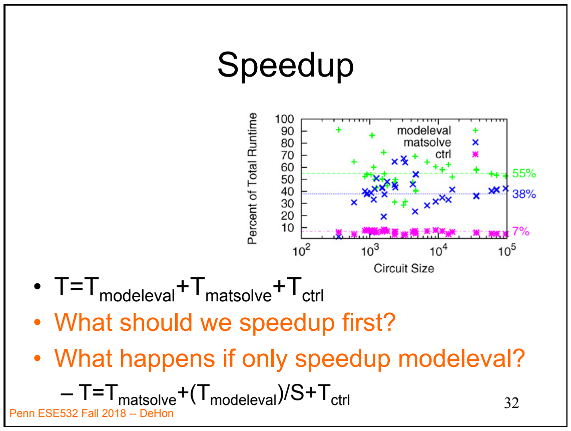
# Analyze





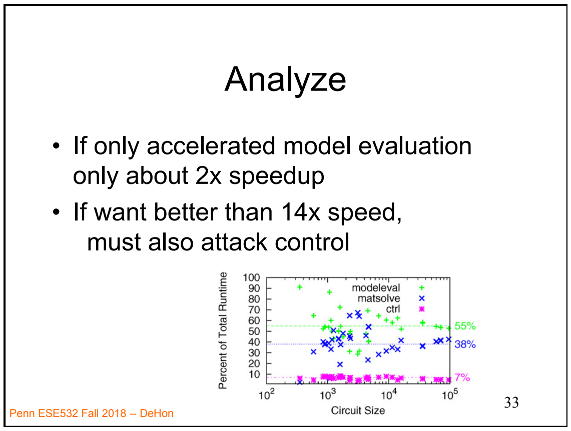
## Analyze

- $T = T_{modeleval} + T_{matsolve} + T_{ctrl}$



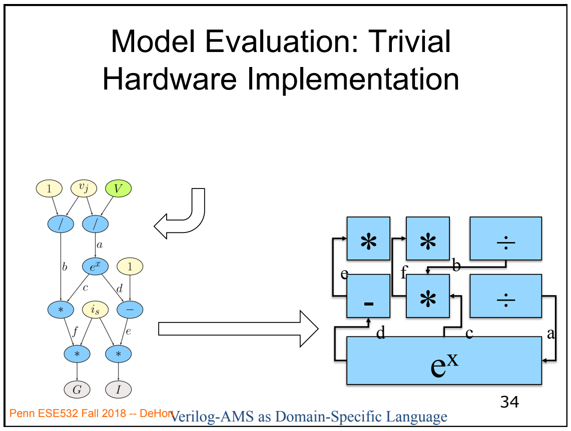
## Speedup

- $T = T_{modeleval} + T_{matsolve} + T_{ctrl}$
- What should we speedup first?
- What happens if only speedup modeleval?
  - $T = T_{matsolve} + (T_{modeleval})/S + T_{ctrl}$



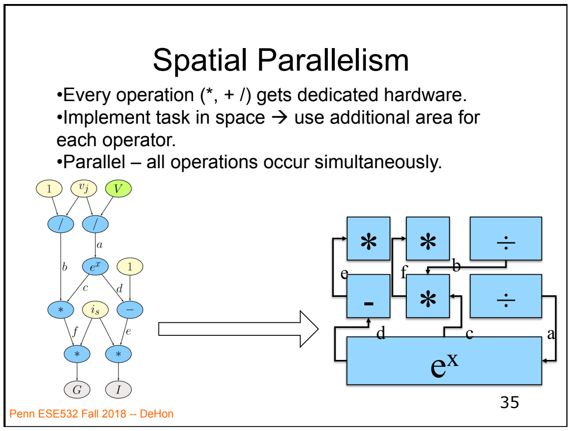
## Analyze

- If only accelerated model evaluation only about 2x speedup
- If want better than 14x speed, must also attack control



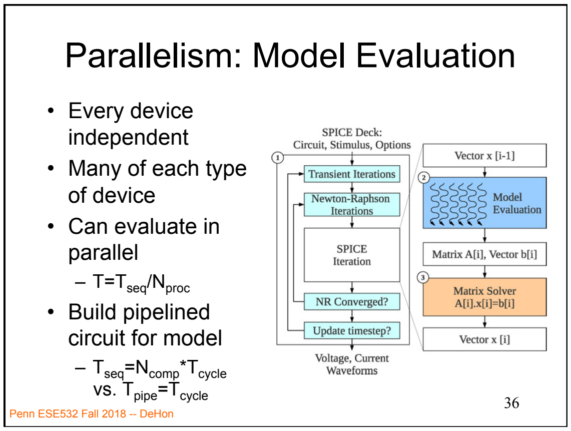
## Model Evaluation: Trivial Hardware Implementation

Verilog-AMS as Domain-Specific Language



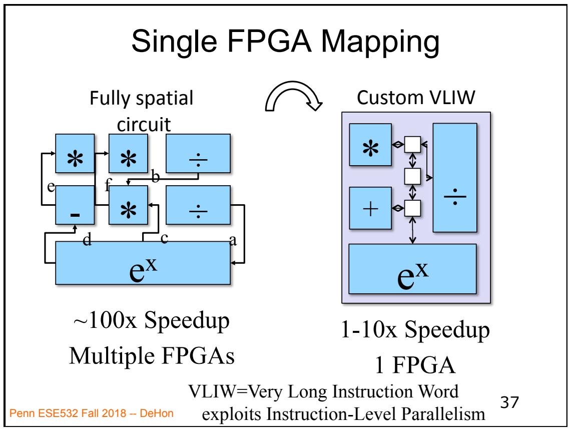
## Spatial Parallelism

•Every operation (*, + /) gets dedicated hardware.
•Implement task in space → use additional area for each operator.
•Parallel – all operations occur simultaneously.



## Parallelism: Model Evaluation

- Every device independent
- Many of each type of device
- Can evaluate in parallel
  - $T = T_{seq}/N_{proc}$
- Build pipelined circuit for model
  - $T_{seq} = N_{comp} * T_{cycle}$ vs. $T_{pipe} = T_{cycle}$

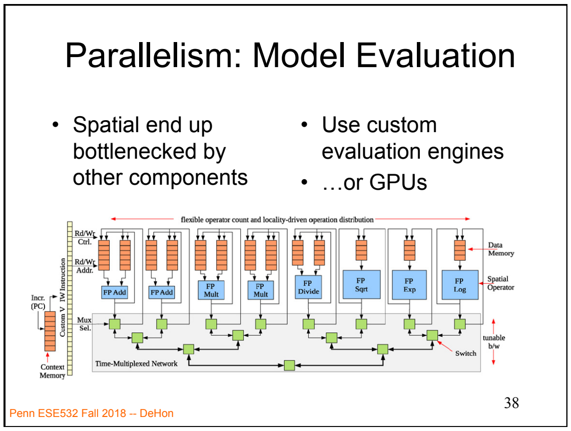


## Single FPGA Mapping

~100x Speedup
Multiple FPGAs

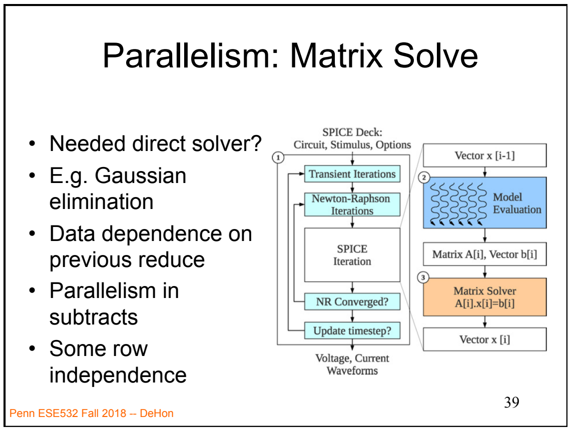
1-10x Speedup
1 FPGA

VLIW=Very Long Instruction Word
exploits Instruction-Level Parallelism



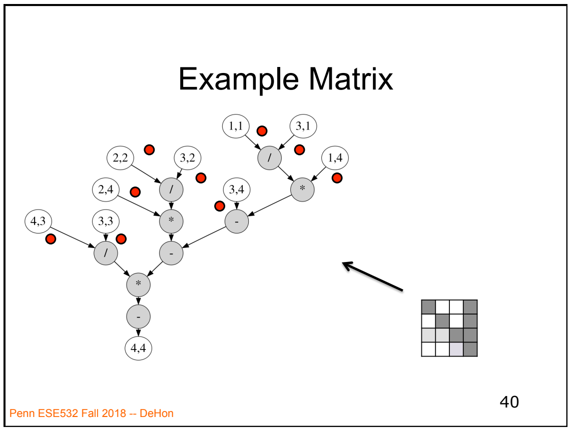
## Parallelism: Model Evaluation

- Spatial end up bottlenecked by other components
- Use custom evaluation engines
- …or GPUs

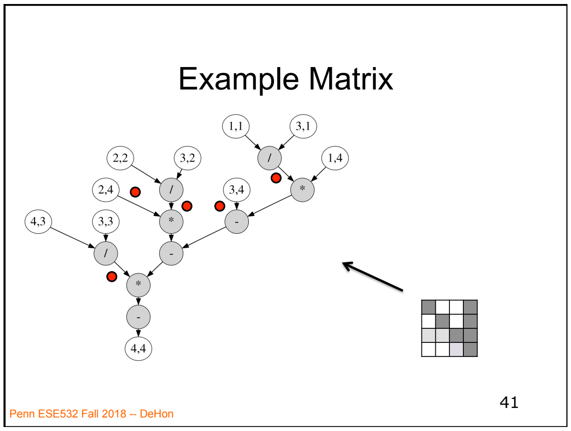


## Parallelism: Matrix Solve

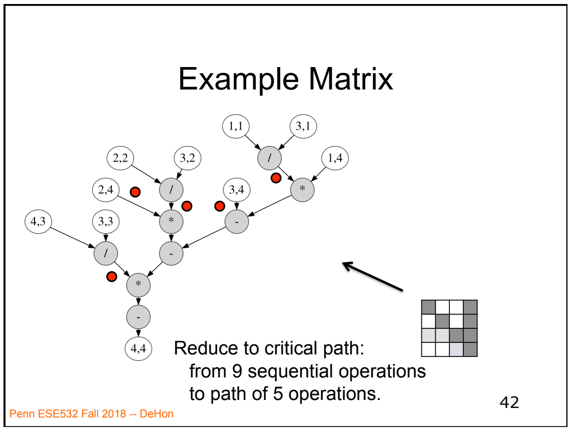
- Needed direct solver?
- E.g. Gaussian elimination
- Data dependence on previous reduce
- Parallelism in subtracts
- Some row independence



## Example Matrix



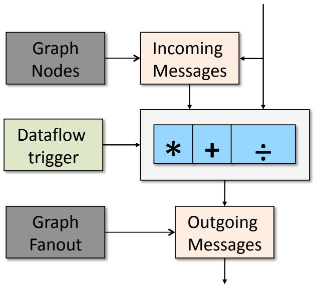
## Example Matrix



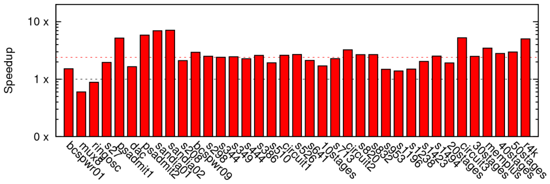
## Example Matrix

Reduce to critical path:
from 9 sequential operations
to path of 5 operations.

## Dataflow PE



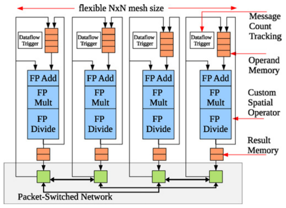
## Matrix Solve Only



~2.4x mean

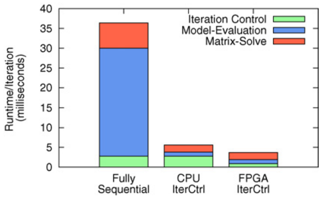
## Parallelism: Matrix Solve

- Settled on constructing dataflow graph
- Graph can be iteration independent
  - Statically scheduled
  - (cheaper)
- This is bottleneck to further acceleration



## Parallelism Controller?

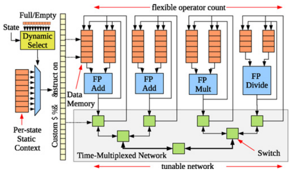
- Could leave sequential
- For some designs, becomes the bottleneck once others accelerated
- Has internal parallelism in condition evaluation



$T=T_{modeleval}/S_1+(T_{matsolve})/S_2+T_{ctrl}$

## Parallelism Controller

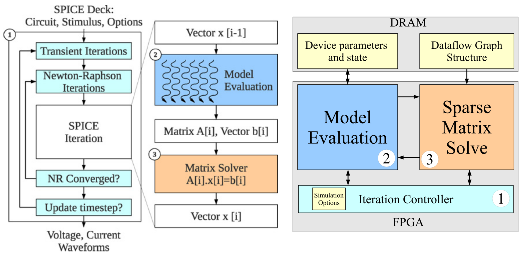
- Customized datapath controller

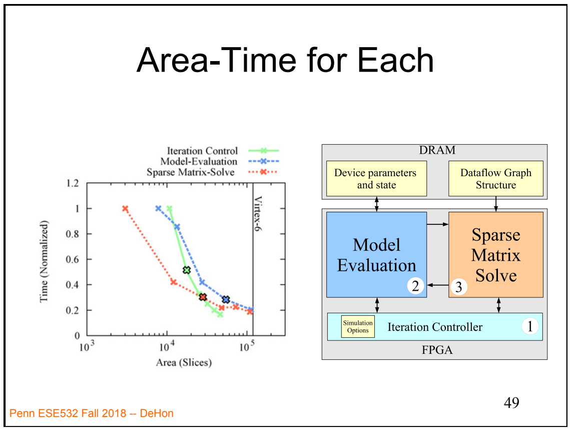


$T_{seqctrl}=N_{add}+N_{mul}+10*N_{divide}$

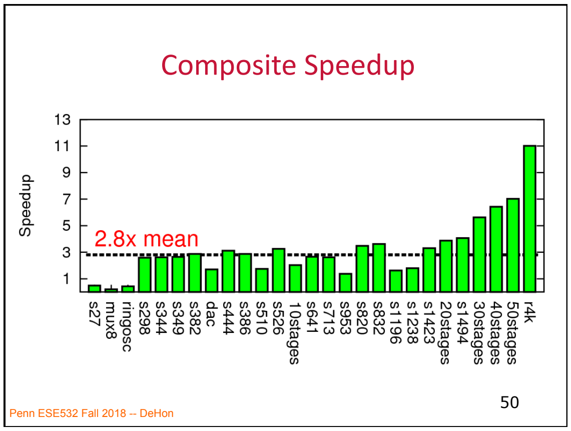
$T_{vliwctrl}=Max(N_{add}/2,N_{mul},10*N_{divide})$

## Single-Chip Solution

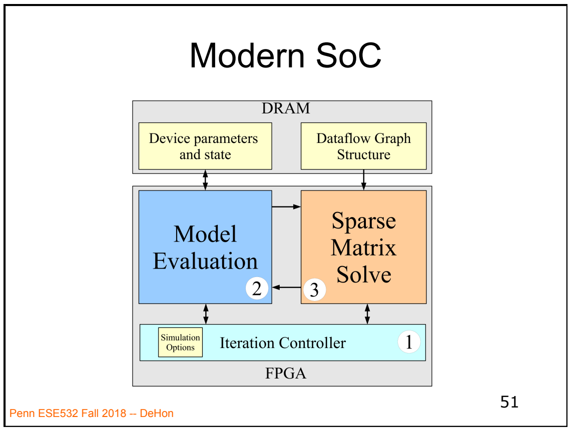




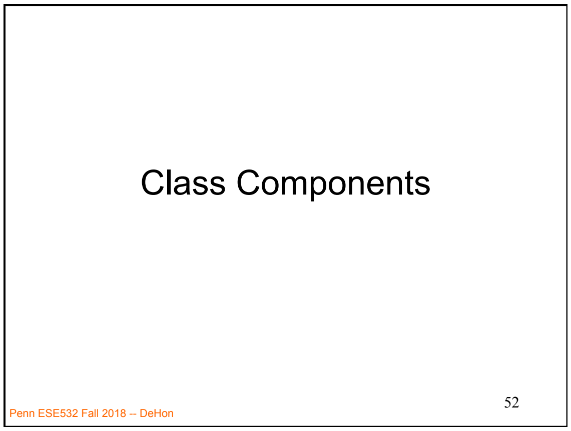
## Area-Time for Each



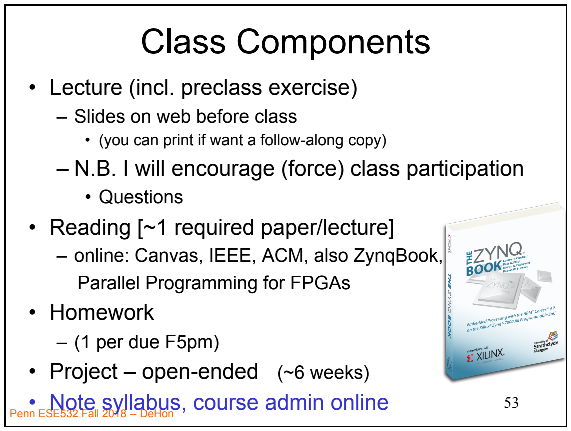
## Composite Speedup



## Modern SoC

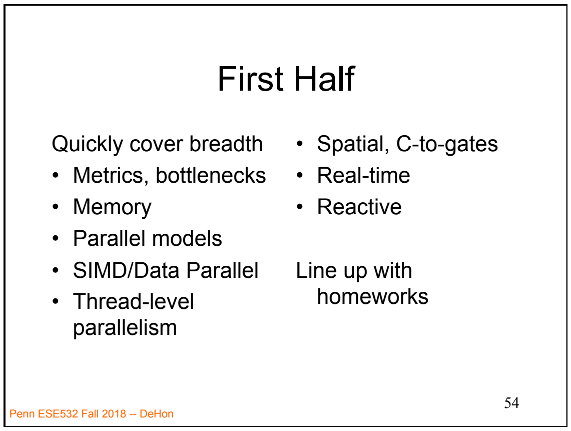


## Class Components



## Class Components

- Lecture (incl. preclass exercise)
  - Slides on web before class
    - (you can print if want a follow-along copy)
  - N.B. I will encourage (force) class participation
    - Questions
- Reading [~1 required paper/lecture]
  - online: Canvas, IEEE, ACM, also ZynqBook, Parallel Programming for FPGAs
- Homework
  - (1 per due F5pm)
- Project – open-ended (~6 weeks)
- Note syllabus, course admin online



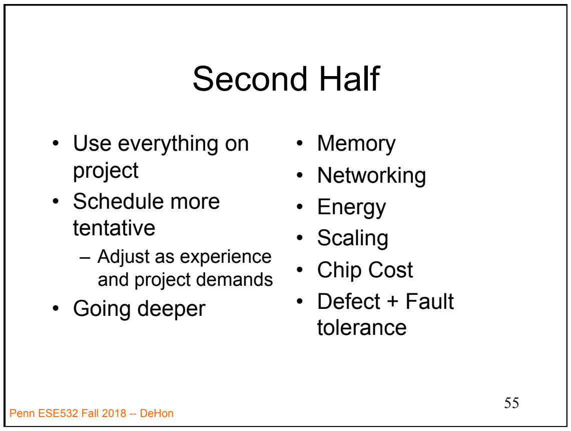
## First Half

Quickly cover breadth

- Metrics, bottlenecks
- Memory
- Parallel models
- SIMD/Data Parallel
- Thread-level parallelism

- Spatial, C-to-gates
- Real-time
- Reactive

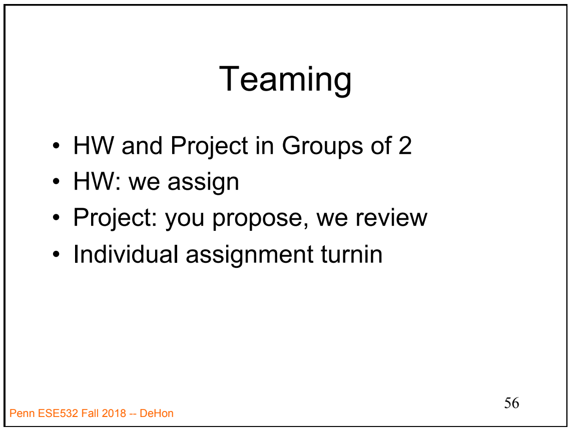
Line up with homeworks



## Second Half

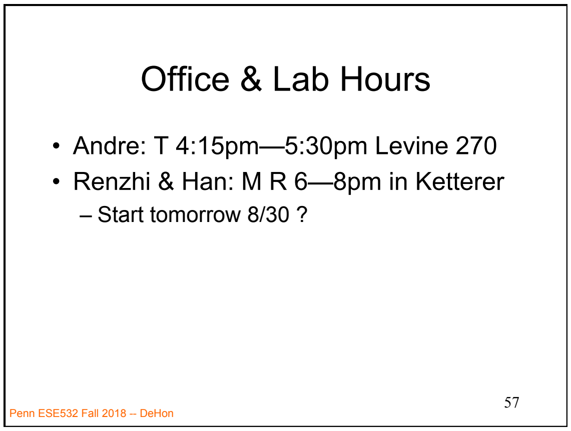
- Use everything on project
- Schedule more tentative
  - Adjust as experience and project demands
- Going deeper

- Memory
- Networking
- Energy
- Scaling
- Chip Cost
- Defect + Fault tolerance



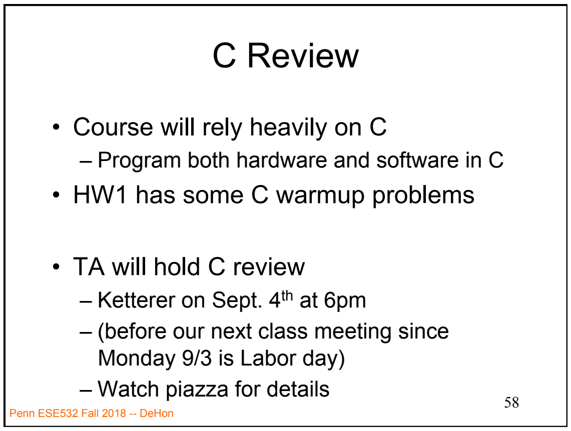
## Teaming

- HW and Project in Groups of 2
- HW: we assign
- Project: you propose, we review
- Individual assignment turnin



## Office & Lab Hours

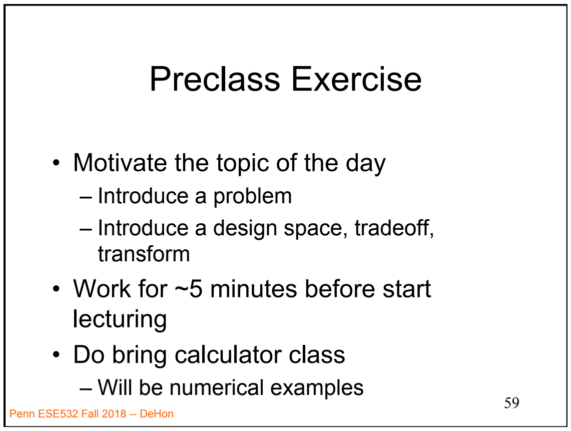
- Andre: T 4:15pm—5:30pm Levine 270
- Renzhi & Han: M R 6—8pm in Ketterer
  - Start tomorrow 8/30 ?



## C Review

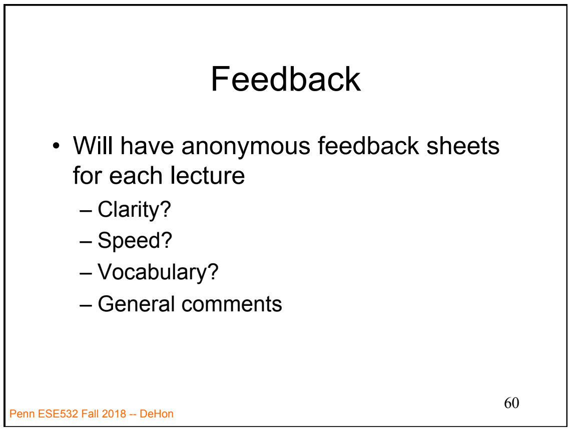
- Course will rely heavily on C
  - Program both hardware and software in C
- HW1 has some C warmup problems

- TA will hold C review
  - Ketterer on Sept. 4th at 6pm
  - (before our next class meeting since Monday 9/3 is Labor day)
  - Watch piazza for details



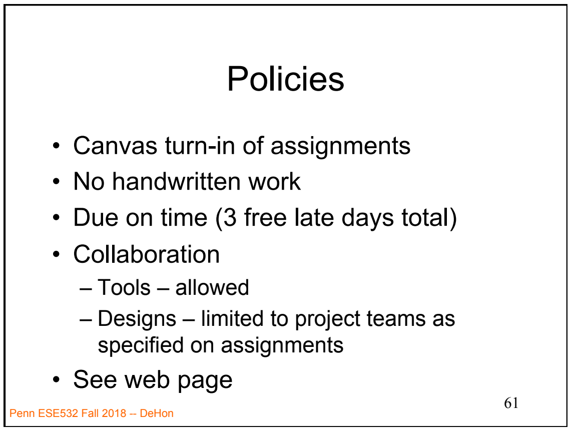
## Preclass Exercise

- Motivate the topic of the day
  - Introduce a problem
  - Introduce a design space, tradeoff, transform
- Work for ~5 minutes before start lecturing
- Do bring calculator class
  - Will be numerical examples



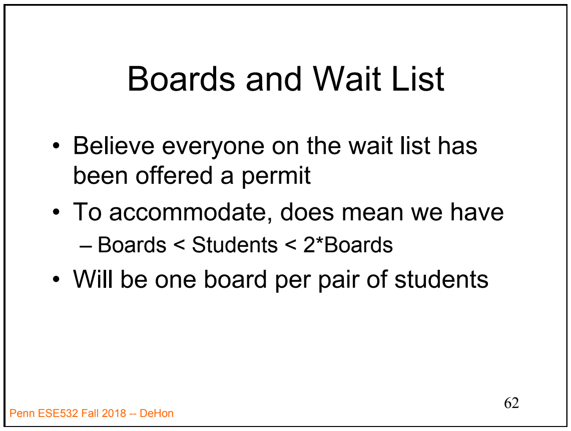
## Feedback

- Will have anonymous feedback sheets for each lecture
  - Clarity?
  - Speed?
  - Vocabulary?
  - General comments



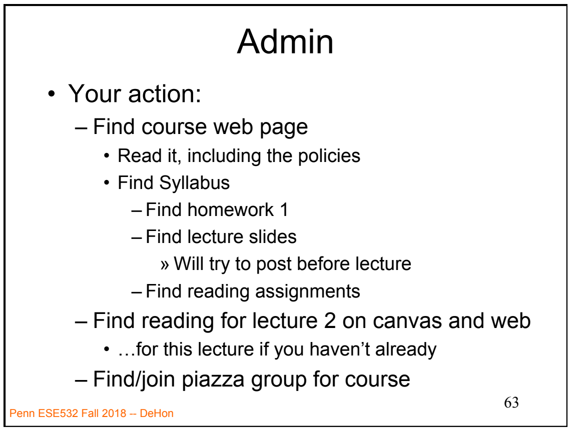
## Policies

- Canvas turn-in of assignments
- No handwritten work
- Due on time (3 free late days total)
- Collaboration
  - Tools – allowed
  - Designs – limited to project teams as specified on assignments
- See web page



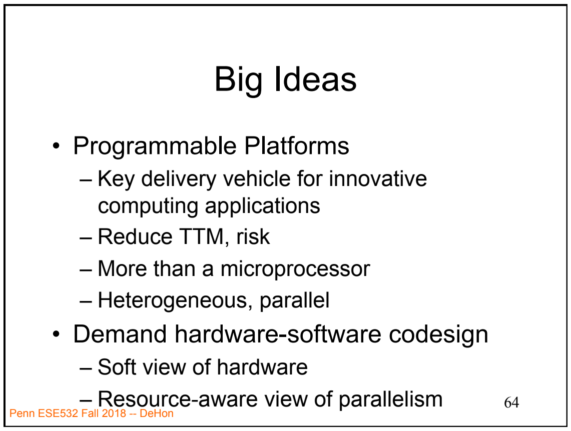
## Boards and Wait List

- Believe everyone on the wait list has been offered a permit
- To accommodate, does mean we have
  - Boards < Students < 2*Boards
- Will be one board per pair of students



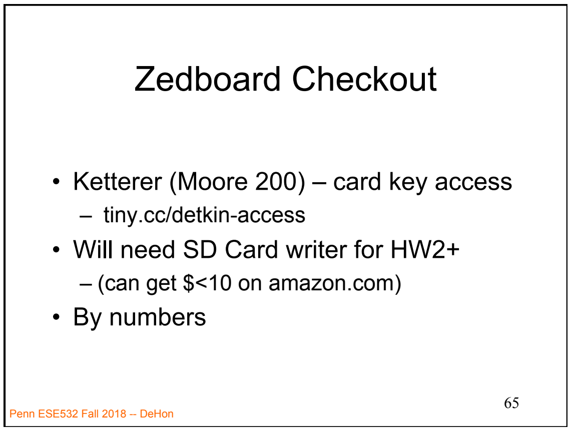
## Admin

- Your action:
  - Find course web page
    - Read it, including the policies
    - Find Syllabus
      - Find homework 1
      - Find lecture slides
        - » Will try to post before lecture
      - Find reading assignments
  - Find reading for lecture 2 on canvas and web
    - …for this lecture if you haven't already
  - Find/join piazza group for course



## Big Ideas

- Programmable Platforms
  - Key delivery vehicle for innovative computing applications
  - Reduce TTM, risk
  - More than a microprocessor
  - Heterogeneous, parallel
- Demand hardware-software codesign
  - Soft view of hardware
  - Resource-aware view of parallelism



## Zedboard Checkout

- Ketterer (Moore 200) – card key access
  - tiny.cc/detkin-access
- Will need SD Card writer for HW2+
  - (can get $<10 on amazon.com)
- By numbers