Teaching Hardware/Software Codesign to the Next Generation of Computer Engineers

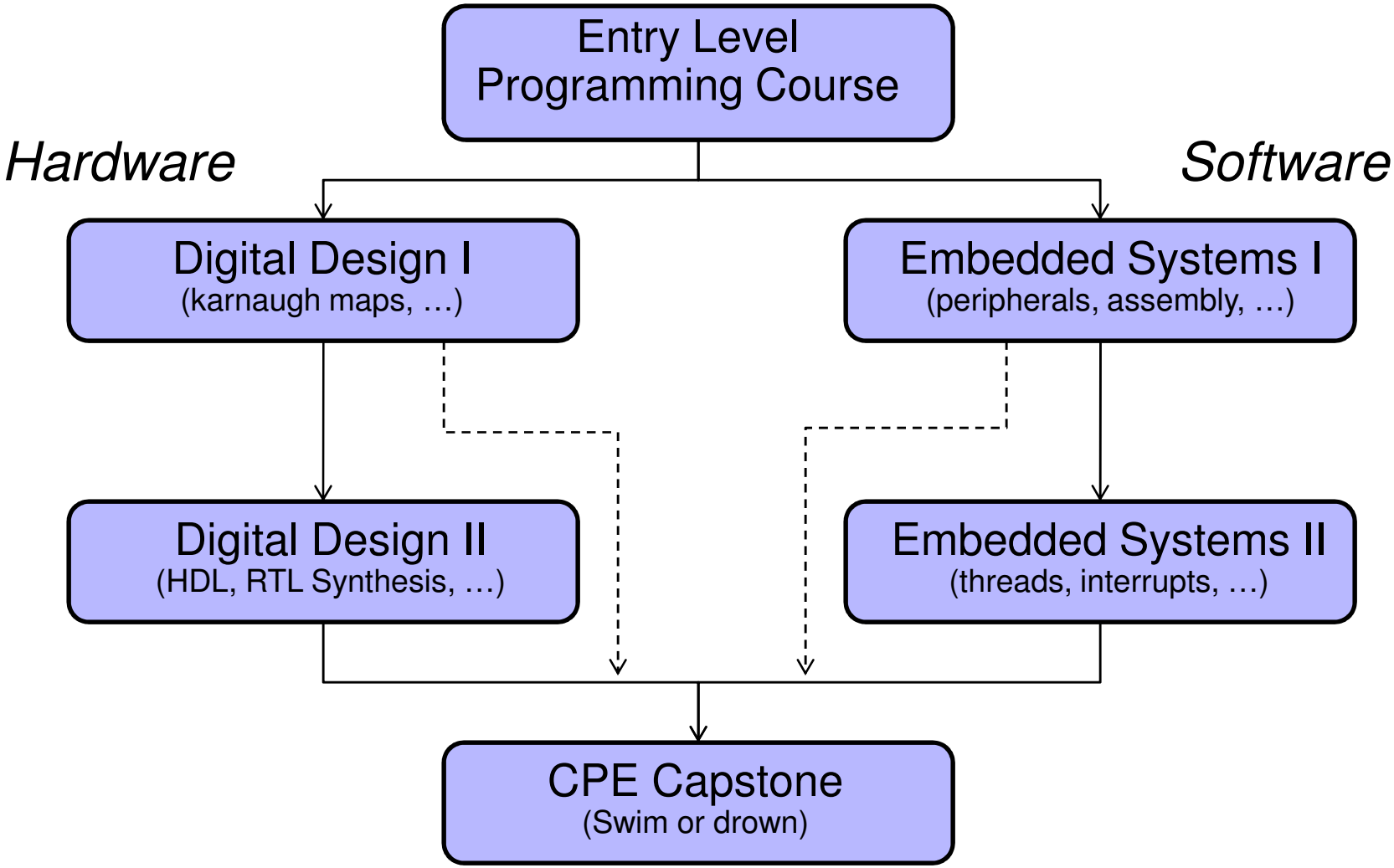
Patrick Schaumont
Virginia Tech
Dept. of Electrical and Computer Engineering

University of California at Riverside
6 June 2008

Outline

- Current Curricula and New Realities
- Codesign starts with Modeling
- A Senior-Level Course in Codesign
- Some Results
- Open Challenges & Conclusions

Current Comp Eng Curricula

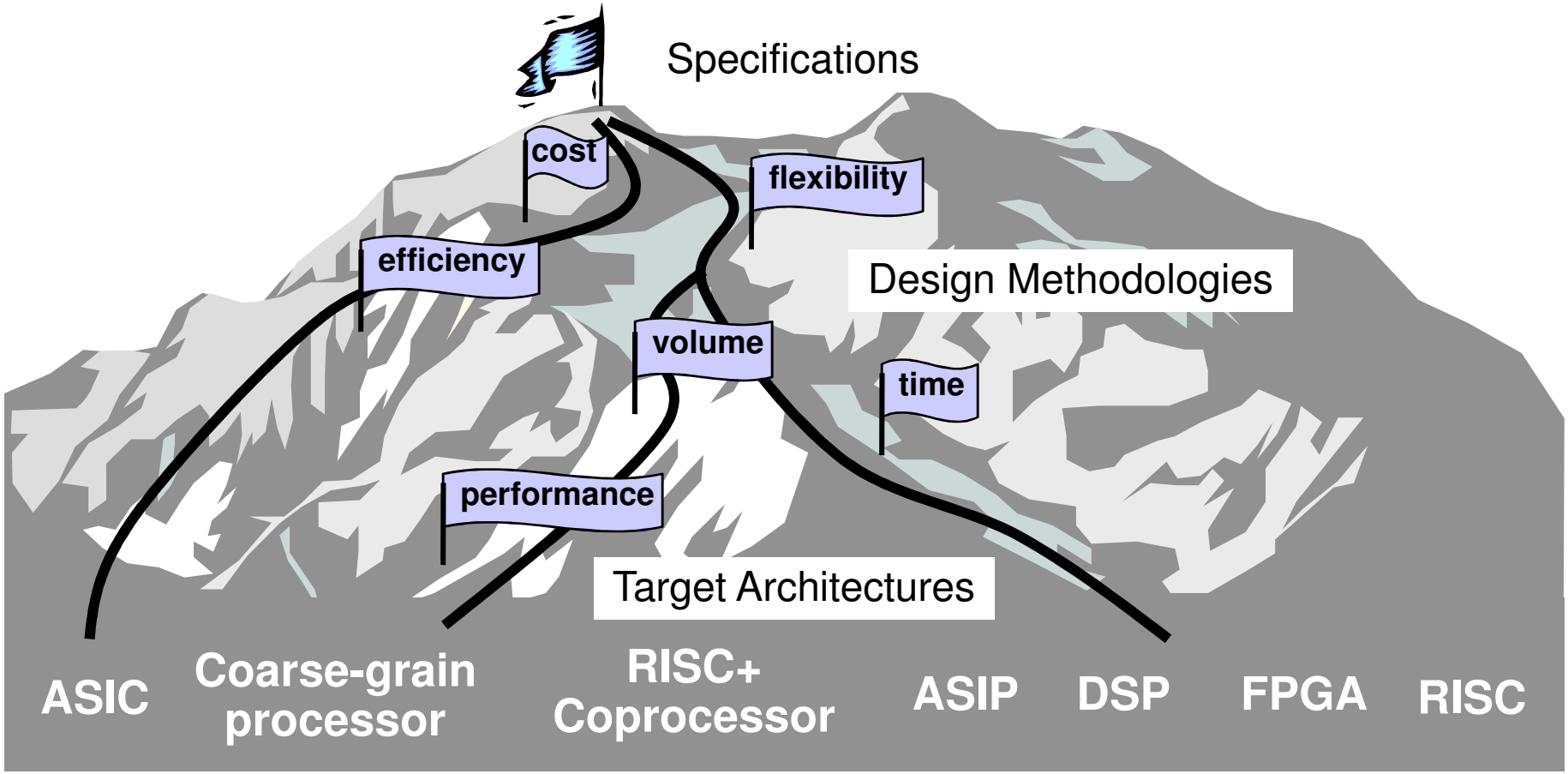


Current Comp Eng Curricula - Issues

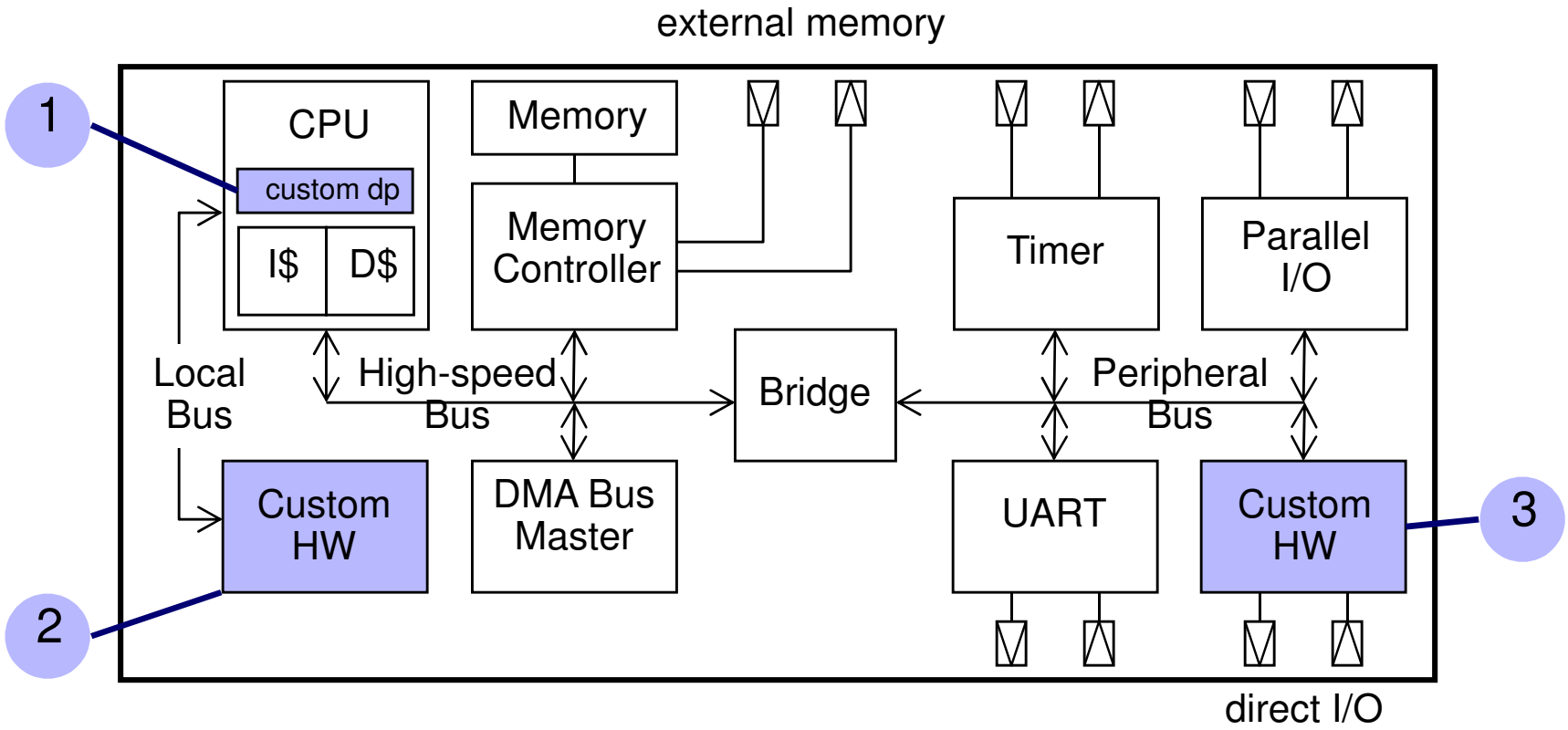
- *Hardware* and *Software* tracks are separated
- Focus is on component design, not on systems building
- Integration & systems are considered only at capstone (way too late)

New Realities – One top, many slopes

- Embedded Systems Mountain



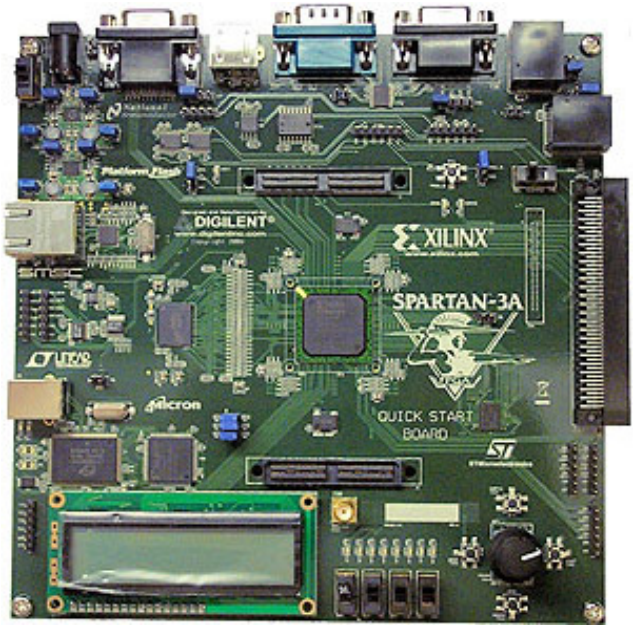
New Realities – Codesign Comeback



1. ASIP Design (processor extension/customization)
2. Coprocessor Design (processor specific interface)
3. Custom Peripheral Design (bus interface)

New Realities - SW is more than C

- Programmable Components rule modern design

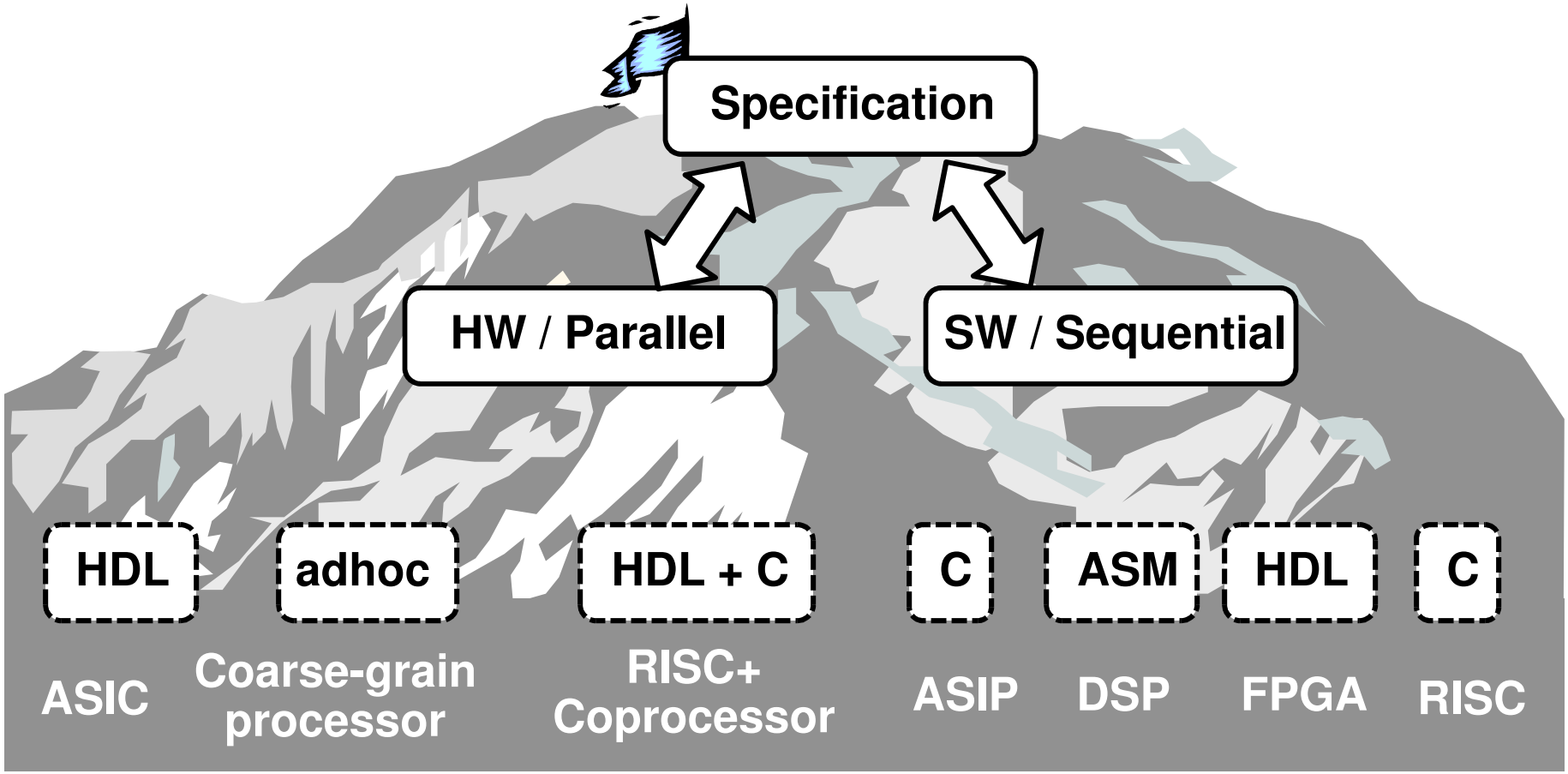


$109 FPGA board
*.. used in **5** courses at VT*

***250** boards in use at*
any given semester

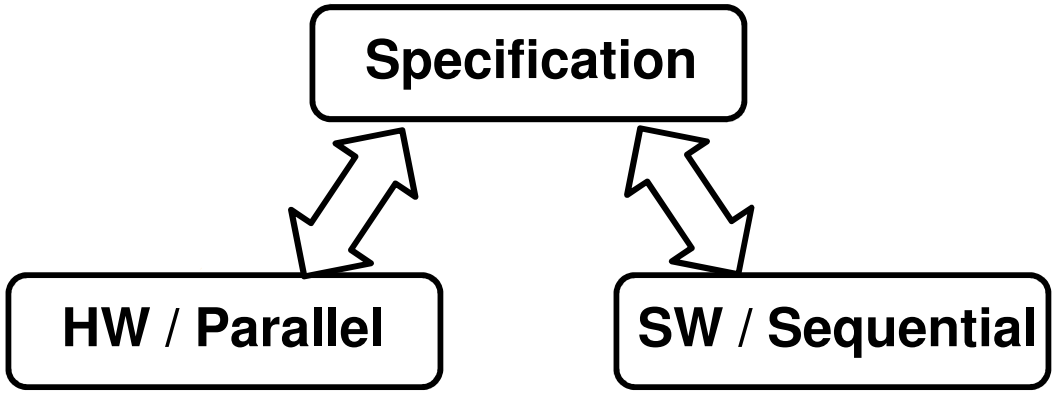
Key issue in codesign is modeling

- Objective: Learn the equivalence between a HW (parallel) and a SW (sequential) implementation

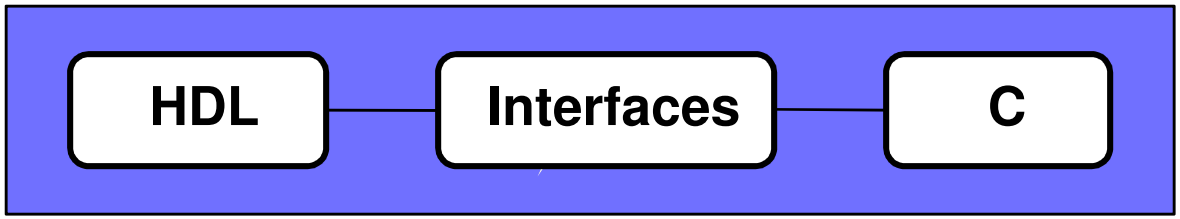


Simple concept, easy implementation?

- Codesign Model based on HDL, Interfaces and C: (too) complex in a senior course



Codesign Model

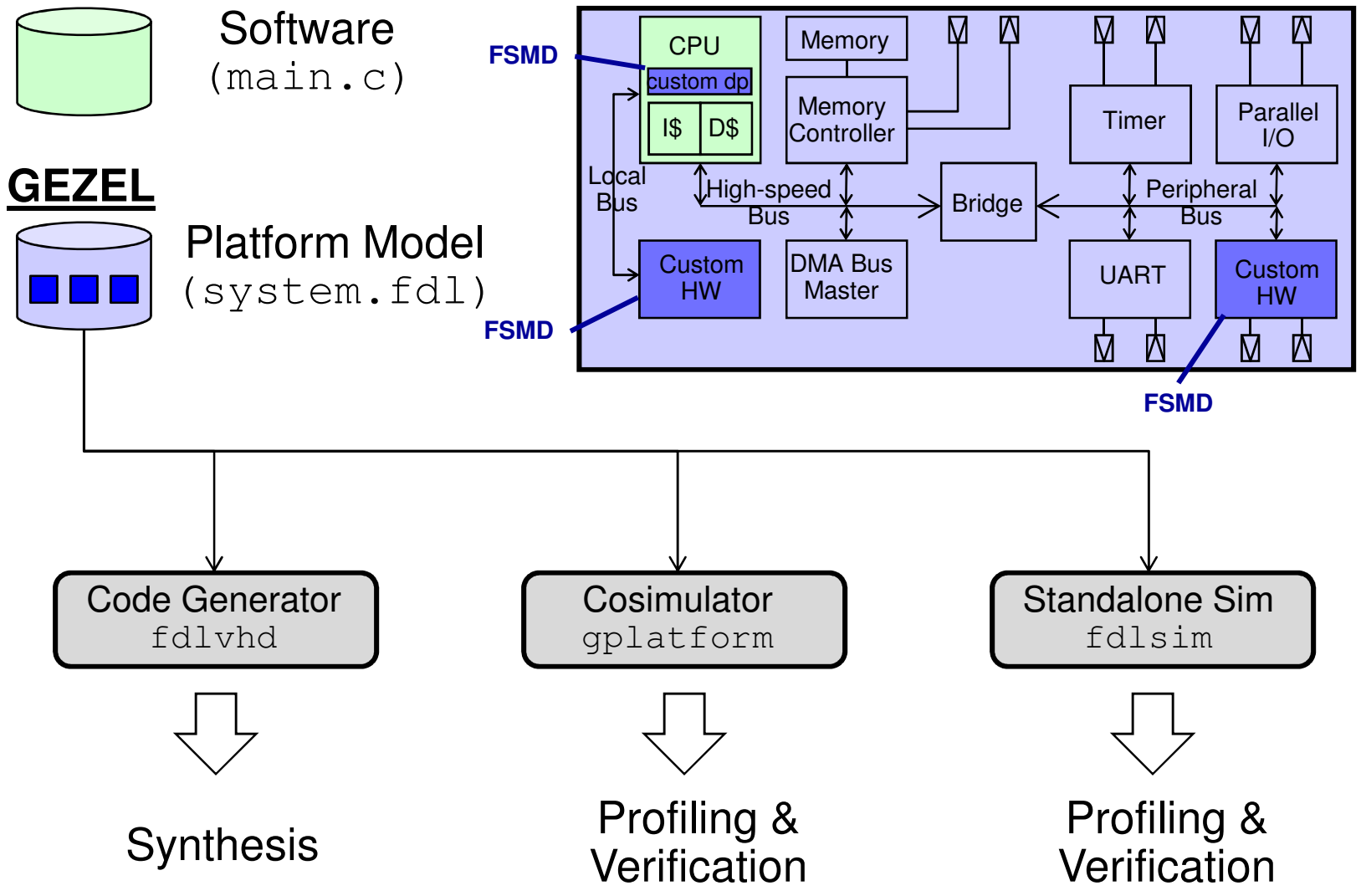


GEZEL Codesign Environment

GEZEL is:

- A cycle-based hardware design language combining
  - Finite-State-Machine-with-Datapath models (HW)
  - Custom interfaces (interfaces to SW)
- A simulation kernel
  - Determinate two-phase simulation semantics
  - Extensible through standard interface (Java, SystemC, Simit-ARM, Dalton-8051, ..)
- A code generator
  - VHDL Code
  - GEZEL code is 100% synthesizable

GEZEL Implementation

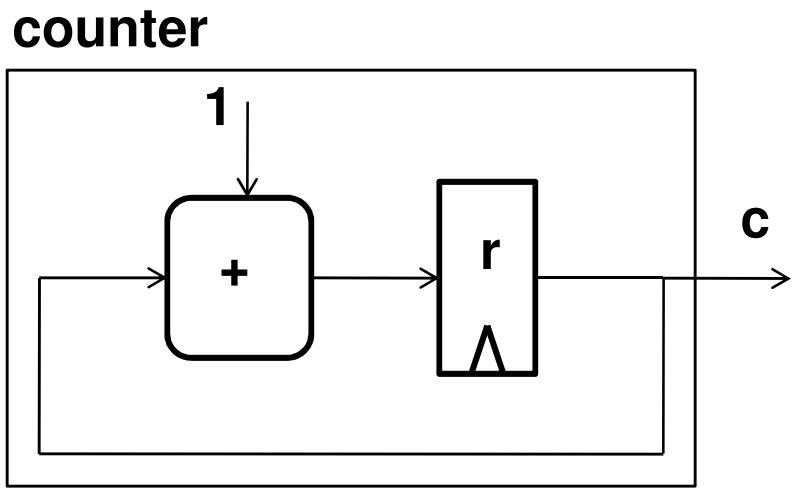


Advantage of GEZEL for Codesign

1. Cycle-based, implementation oriented modeling

GEZEL

```
dpcounter(out c : ns(8)) {
reg r : ns(8);
  always {
    r = r + 1;
    c = r;
  }
}
```



Advantage of GEZEL for Codesign

2. Explicit Modeling of Hardware Control

GEZEL

```
dpupdncounter(out c : ns(8)) {
regr : ns(8);
sfgup {r = r + 1; }
sfgdn {r = r - 1; }
  always {c = r; }
}

fsmctl(updncounter) {
initial s0;
state s1;
  @s0 if (r == 255) then (dn) -> s1;
                    else (up) -> s0;
  @s1 if (r ==   0) then (up) -> s0;
                    else (dn) -> s1;
}
```

Advantage of GEZEL for Codesign

3. Abstract HW/SW Interfaces

C

driver.c

```
int main() {
  volatile int *d = (int *) 0x80000000;
  ...
  *d = 15;
  ...
  return 0;
}
```

GEZEL

```
ipblock myarm {
iptype "armsystem";
ipparm "exec=driver";
}

ipblock b_in(out data : ns(32)) {
iptype "armsystemsource";
ipparm "core=myarm";
ipparm "address=0x80000000";
}
```

ECE 4530: Hardware/Software Codesign

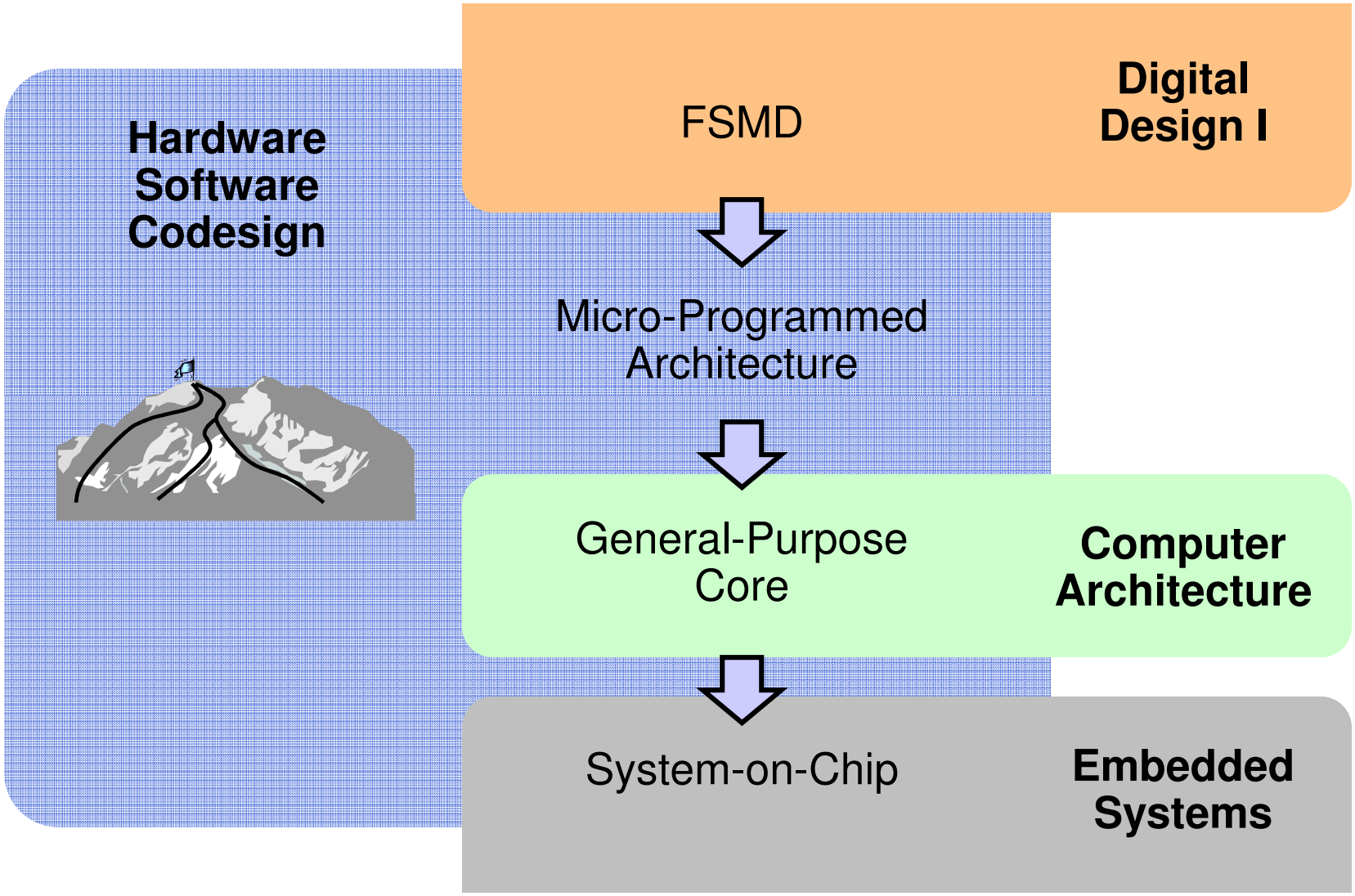
- Design Technical Elective for CPE seniors and incoming graduate students (30 – 40 students)
- Lecture Organization
  - Part 1 – Fundamentals
  - Part 2 – Custom Architecture Design Space
  - Part 3 – Interfaces
- Assignments
  - Weekly assignments with hands-on design experiments
  - Final project 'Codesign Challenge': Competition to create fastest implementation for a given spec (in C)

Part 1 - Fundamentals

- Synchronous Dataflow
  - Nice formal properties *and* practical applications
  - Analysis of stability [Lee 87]
  - Refinement in software and hardware
  - Optimizations – multi-rate expansion, pipelining, ...

- Control Dependence and Data Dependence
  - A data dependence must be implemented regardless of the underlying architecture
  - A control dependence may be removed if the underlying architecture can handle the resulting concurrency

Part 2 - Custom Architectures



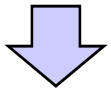
Part 2 - Custom Architectures

FSMD

+ Hardware Equivalent for a C function
- Non-programmable, non-scalable, complex

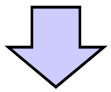
Micro-Programmed Architecture

+ Programmable version of FSMD
- Does not cope well with pipelining



General-Purpose Core

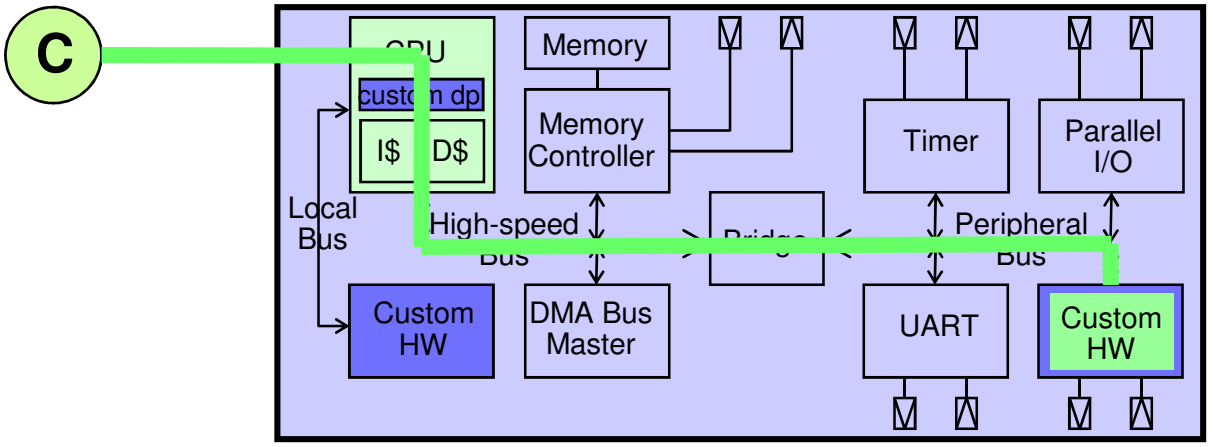
+ Automatic hazard resolution
- No custom hardware



System-on-Chip

+ Combines FSMD and GP Core
- May have bus bottlenecks

Part 3 - Interfaces



- The path from C into hardware
  - On-chip bus with memory-mapped hardware
  - Processor-specific bus with coprocessor hardware
  - Processor-instructions for custom datapath
- Key Elements
  - On-chip busses (OPB, PLB), Interfaces (FSL)
  - Control Shell for Custom Hardware

Overhead is in the interconnections

- Hardware acceleration without considering integration is usually pointless

AES in **SW**
32-bit ARM

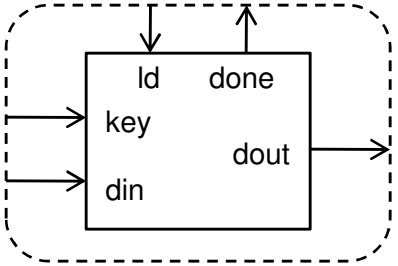
3627 cycles/ encryption

```
void aes_enc(char *text,
char *key,
char *out)
```

Ideal Speedup 330x

AES in **HW**
standalone

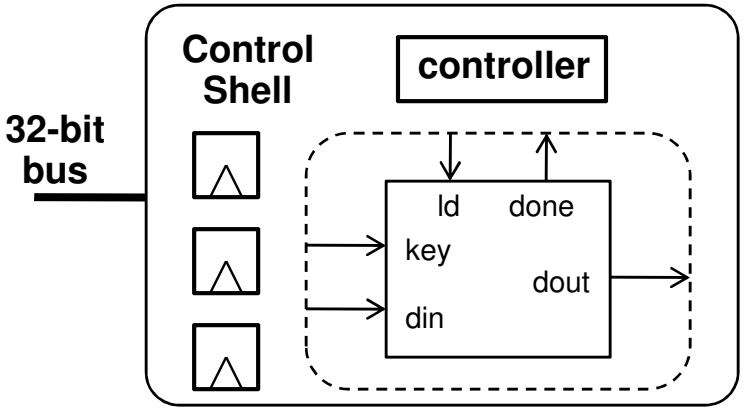
11 cycles/ encryption



Actual Speedup 1.1x

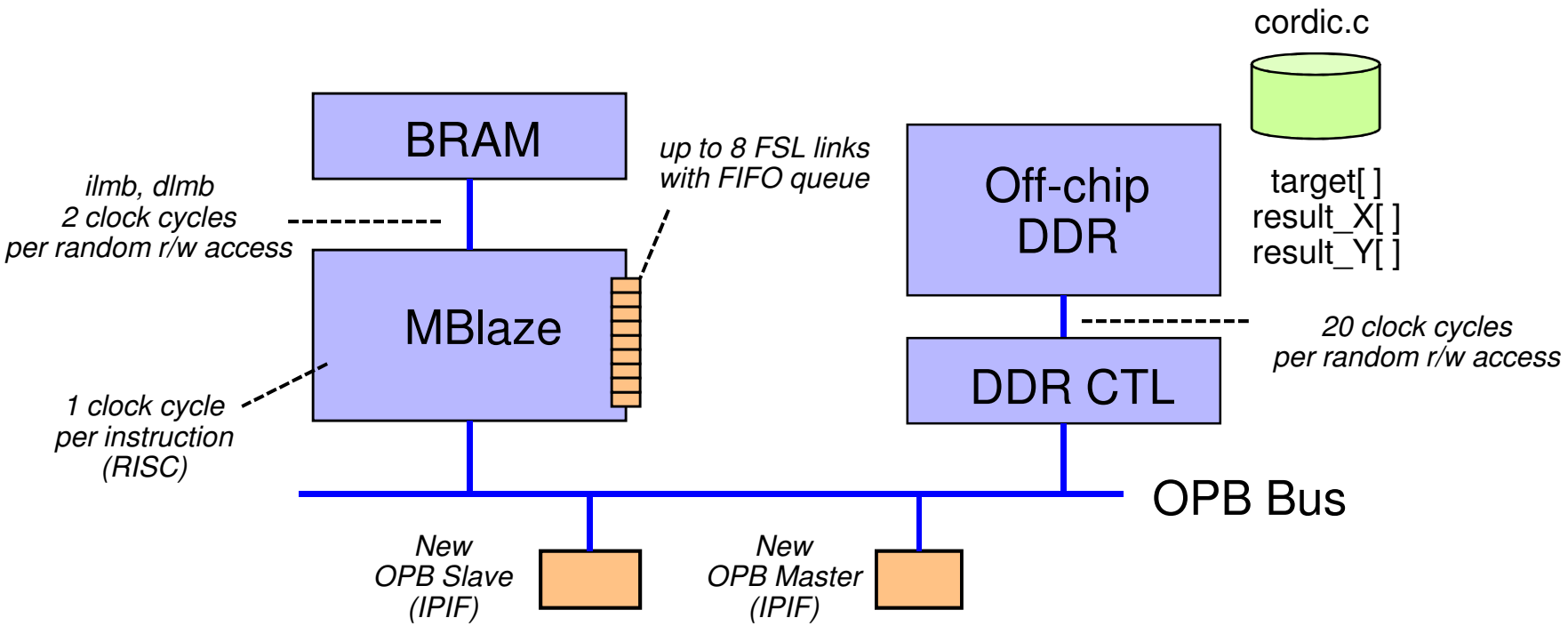
AES in **HW**
with memory mapped control shell

3338 cycles/ encryption

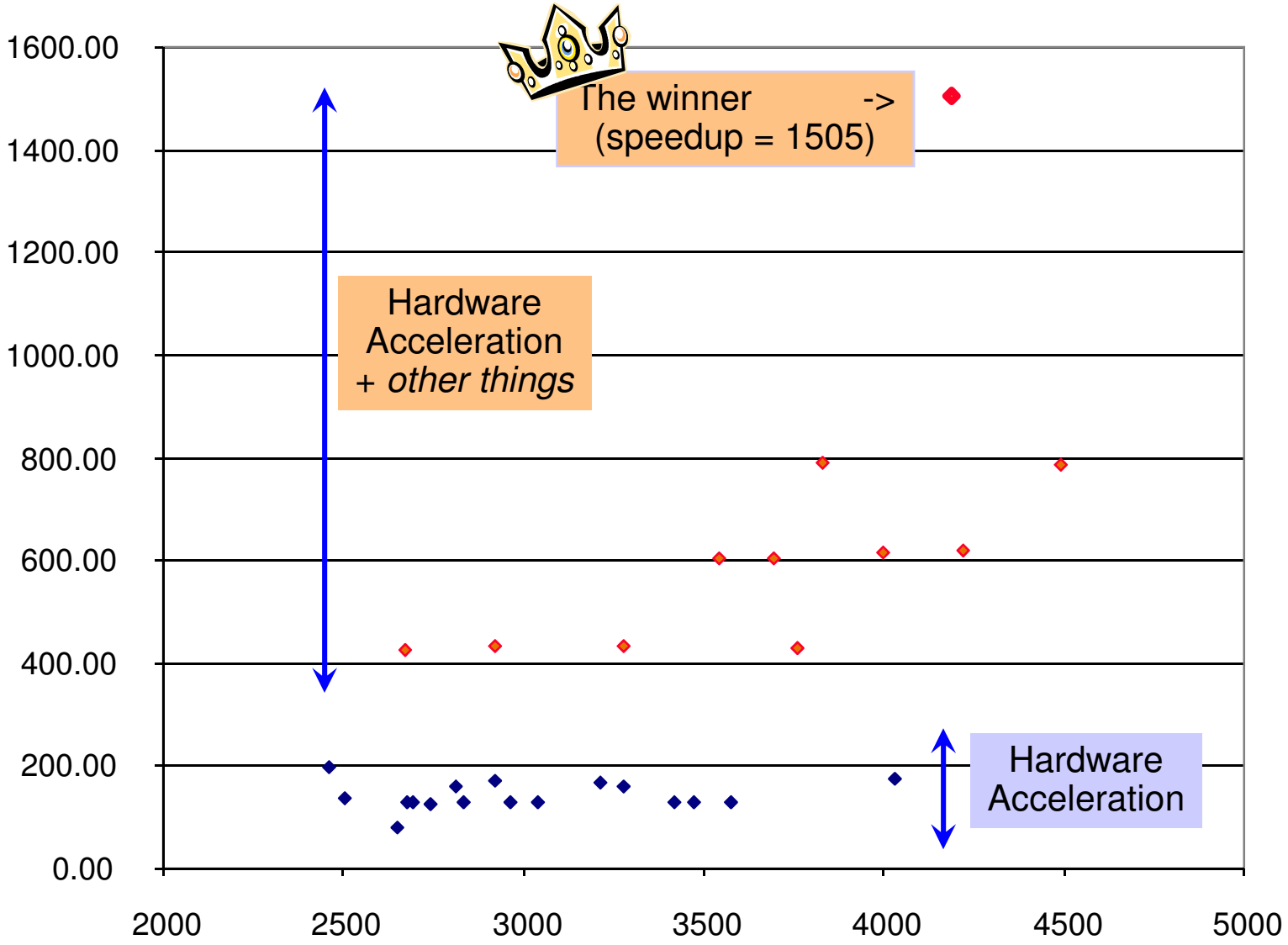


Some Results

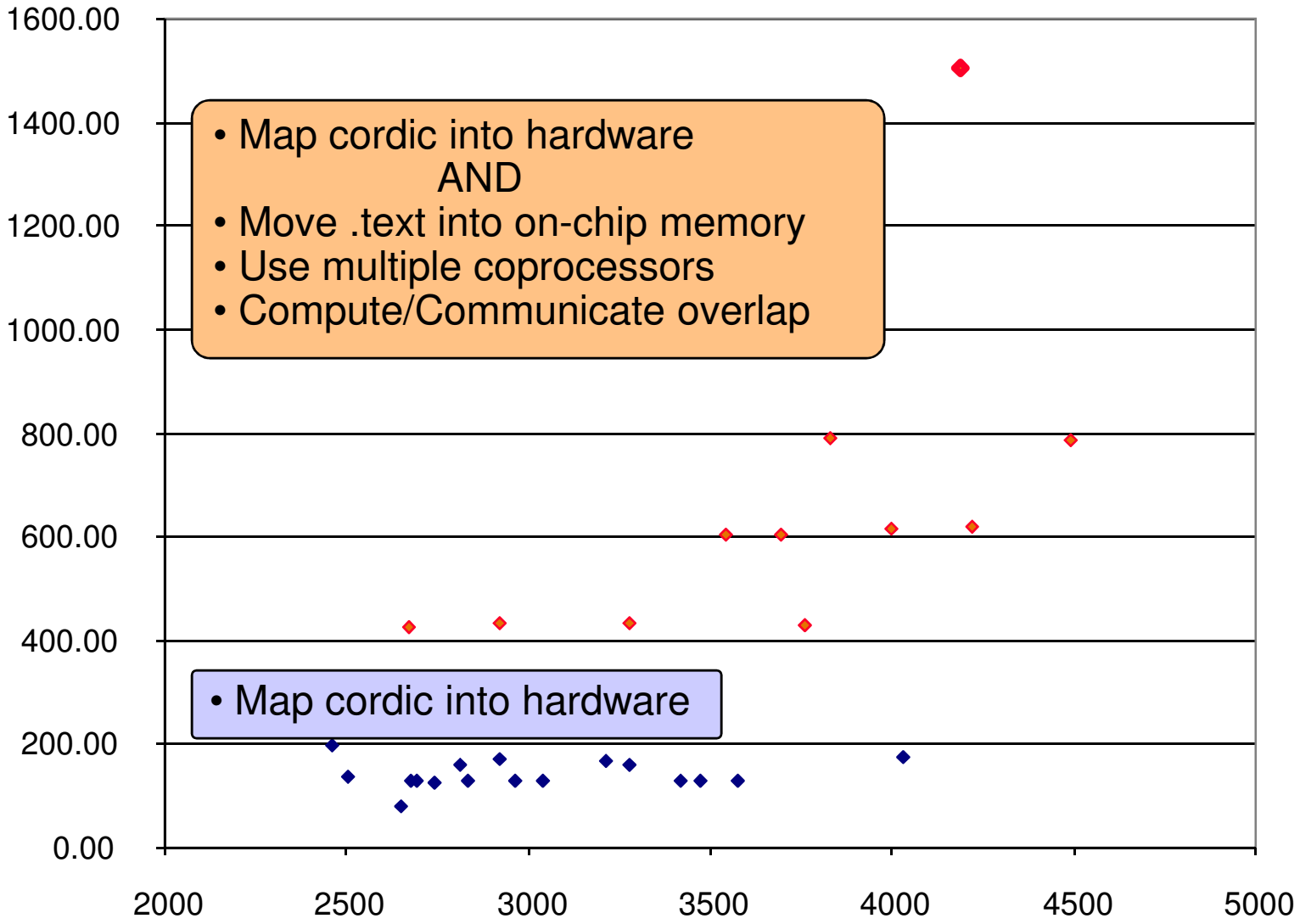
- Final course project is *the codesign challenge*
- Fall 07: Given cordic.c (64K cordic rotations), provide maximal possible speedup in two weeks design time



Codesign Challenge: speedup vs slices



Codesign Challenge: speedup vs slices



Conclusions

- Codesign is a first step to integrate application and architecture design
- Seniors enjoy co-design
  - F06 (23), F07 (31), F08 (40?)
- Modeling is key
  - HDL is not up to the job
  - Abstraction by itself is not sufficient (RTL *still* very useful)
  - Don't forget the path to implementation
- Open challenges
  - Need a textbook
  - Build *systems* (not components) *early on* in CPE
  - Architecture space exploding; structured approach required