A senior-level course
in hardware-software codesign

Patrick Schaumont
schaum@vt.edu

Secure Embedded Systems Group
ECE Department

The need for hw/sw codesign education

<table>
<thead>
<tr>
<th>'Hardware'</th>
<th>MPU</th>
<th>DSP</th>
<th>ASIP</th>
<th>FPGA</th>
<th>Multi-core</th>
</tr>
</thead>
<tbody>
<tr>
<td>'Software'</td>
<td>C</td>
<td>C + Assembly</td>
<td>C + HDL</td>
<td>VHDL, Verilog</td>
<td>?</td>
</tr>
</tbody>
</table>

VirginiaTech
Invent the Future
Undergraduate hw/sw codesign education

- Classic hardware-software codesign is complex
  - Large amount of prerequisite knowledge
  - Embedded software, operating systems
  - VHDL simulation and synthesis
  - Hardware-software interfacing and partitioning

- Need to have a simplified approach
  - Minimize amount of prerequisite knowledge
  - Stress the common concepts in codesign, not the diversity

- Senior course given at Virginia Tech in fall 2006

A cause of learning trouble: CONTEXT

What you want to do: SystemC design
What you have to do: VHDL Synthesis
Context: C++

FPGA Board

Manual
Reducing the context in codesign

- Increase the abstraction level of design
  - But do NOT unify hardware with software
- Take an incremental approach
  - Don't solve general hardware-software partitioning
  - But focus on the case of translating C into C-with-Coprocessor
- Simplify program semantics
  - Software $\rightarrow$ single-thread sequential behavior (in C)
  - Hardware $\rightarrow$ cycle-based synchronous RTL
- Simplify Tools
  - GEZEL codesign environment
  - Knoppix-based cosimulation environment + Xilinx EDK

Prerequisites

- Target Audience: senior (4th year) CE undergraduates

<table>
<thead>
<tr>
<th>Low-level Software</th>
<th>3rd Year</th>
<th>Comb &amp; Seq Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microprocessors</td>
<td></td>
<td>Digital Design I</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Embedded Software</td>
<td>4th Year</td>
<td></td>
</tr>
<tr>
<td>Embed Systems</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Codesign</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Digital Design II</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Software Focus

Hardware Focus
Course outline

- Basic concepts
  - Concurrency versus Parallelism
  - Control Flow versus Data Flow

- Fundamentals of Custom Architecture Design
  - Finite-state Machine with Datapath
  - Micro-programmed Architecture
  - Coprocessors in SoC context

- Methods to map Applications into Architectures
- Recent developments in Codesign

Basic Concepts: Parallelism vs Concurrency

- Explain using synchronous dataflow (SDF [Lee 87])
- Model/implement single-rate SDF in HW and SW
Basic Concepts: Control - versus Dataflow

Any implementation of a given C program
- MUST implement the full DFG
- MAY implement the CFG

GEZEL Example

32-bit StrongARM Microprocessor

32-bit running counter
mem-mapped interface
GEZEL Example

ipblock qout(in d : ns(32)) {
  iotype "armsystemsink";
  ipparm "address = 0x4000";
}

dp counter {
  reg c : ns(32);
  sig q : ns(32);
  use qout(q);
  always {
    c = c + 1;
    q = c;
  }
}

#include <stdio.h>

int main() {
  volatile int *d = 0x4000;
  int c0, c1;
  c0 = *d;
  ...
  c1 = *d;
  printf("%d\n", c1 - c0);
  return 0;
}
GEZEL Example

```c
#include <stdio.h>

int main() {
    volatile int *d = 0x4000;
    int c0, c1;

    c0 = *d;
    ...
    c1 = *d;

    printf("%d\n", c1 - c0);
    return 0;
}
```

GEZEL Example

```c
ipblock qout(in d : ns(32)) {
    iptype "armsystemsink";
    ipparm "address = 0x4000";
}

dp counter {
    reg c : ns(32);
    sig q : ns(32);
    use qout(q);
    always {
        c = c + 1;
        q = c;
    }
}
```
Knoppix CDROM

- Live-boot linux system with cross-compilers, cosimulator, VHDL generator, and examples.

Design Project

System Design  C for i386
Embedded Software  C for StrongARM
Hardware Design  GEZEL
System Integration  GEZEL + C cosimulation
Prototyping  Spartan 3E Starter Board, MicroBlaze System, Xilinx EDK

DES  Blowfish  Salsa20  MD5  Misty
**Evaluation: Measurable Learning Objective**

- **Student Questionnaire at end of course**

**Strongly Disagree**  **Disagree**  **Agree**  **Strong Agree**

![Bar chart showing student responses to Measurable Learning Objectives]

1. Analyze & explain data- and control flow of a C program
2. Transform simple C programs into cycle-based hardware descriptions and vice versa.
3. Use simulation software to co-simulate C programs with cycle-based hardware descriptions

4. Design memory-mapped/ interrupt-driven interfaces to implement hw/sw communication
5. Identify performance bottlenecks across hw/sw and optimize them
Conclusions

- With rapidly evolving architecture space, a broader view of application mapping is essential

- Introductory course into hardware/software codesign
  - Emphasis is on combining hardware/software paradigms
  - Avoid complexity and excessive context

- Transforming C into hardware is a viable approach for education
  - Provides the basis for more complex (distributed) designs

- Thanks!
  - Patrick Schaumont (schaum@vt.edu)