**System-level simulation (HW/SW co-simulation)**

**Outline**

- Problem statement
- Simulation and embedded system design
  - functional simulation
  - performance simulation
    - POLIS implementation
    - partitioning example
  - implementation simulation
    - software-oriented
    - hardware-centric
- Summary
Embedded Heterogeneous System

- Digital downconversion (900Mhz, 70Mhz, 10.7Mhz)
- SAW FILTER
- RF IF (40Ms/sec - 540ks/sec)
- Viterbi equalizers
- Demodulation and synchronization
- Baseband processing
- Phonebook keypad interface
- Protocol control
- Speech quality enhancement
- Speech recognition

Logic

Analog digital

Logic

RAM & ROM

DSP core

uC core

Modeling and Simulation Techniques

- Execute C code on a workstation
- Execute object code using
  - Instruction level model
  - Instruction cycle accurate
  - Clock cycle accurate
- HDL model of the processor
  - RTL
  - Gate level netlist
  - Phase accurate model
  - Pin accurate model
  - Fully functional model
  - Spice model of the processor
  - In-circuit emulators
  - Real hardware
  - Bus functional model

Source Models
Smart Models
DesignWare
Hardware Modeler
Problem Statement

- To model the behavior of a combined hardware and software system based on models of the behavior of the hardware and software components.
- Usually requires trading off:
  - accuracy
  - throughput
  - convenience
- Using the right abstractions for the task.

Function and Interface Abstraction

How much visibility do I need to have into:
- what is going on inside a subsystem
- how do two subsystems communicate
- function abstraction
- interface abstraction
Outline

- Problem statement
- Simulation and embedded system design
  - functional simulation
  - performance simulation
    - POLIS implementation
    - partitioning example
  - implementation simulation
    - software-oriented
    - hardware-centric
- Summary

Design flow


for (i=0;i<8;i++)
idctrow(i);
for (i=0;i<8;i++)
idctcol(i);

**Functional model**

- Variable-length dec.
- Flow splitting
- Motion comp.
- IDCT
- Image memory
- Video output
- MPEG 2 decoder
- Video out
- Stream in

**Functional simulation**

- Timeless (not really co-simulation...)
- Algorithm exploration, functional debugging, virtual prototyping
- Different formal models:
  - control-dominated: CSP, EFSMs, D E, etc.
  - event-based: Bones, StateCharts, etc.
  - data dominated: DF networks
  - token-based: Cossap, SPW, etc.
- Single process network (Ptolemy-style)
HW and SW Modeling Techniques

- Hardware centric
- Software oriented

Design flow

- Functional simulation
- Architecture simulation
- Mapping (partitioning)
- Synthesis, coding
- Cycle-based, logic simulation
- Performance simulation
- Behavior capture
- Architecture capture
**Architecture model**

- Abstract model for mapping
  - no detailed wiring (busses, serial links, etc.)
  - black-box components (ASICs, micro-controllers, DSPs, memories, etc.)
- Later refined to a detailed design
  - implement communication
  - refine interfaces

---

**Architecture model**

- CPU (10 SPEC) → ASIC(s) (160 Mops) → Memory (2Mb)
- bus 10Mb/s
Design flow

- Functional simul.
- Behavior capture
- Architecture simul.
- Architecture capture
- Performance simul.
- Mapping (partitioning)
- Synthesis, coding
- Cycle-based, logic simul.

Mapping

- Associates functional units with architectural units
- Performs HW/SW partitioning
- Associates functional communication with resources (buffers, busses, serial links, etc.)
- Provides estimates of performance of a given function on a given architectural unit
Mapping

Variable-length dec. → Flow splitting → Motion comp. → Image memory → Video output

CPU (10 SPEC) → ASIC (160Mops) → Memory (2Mb)

Stream in → Video out

bus 10Mb/s
Mapping

Variable-length dec. → Flow splitting → Motion comp.

IDCT → Image memory → Video output

Video out

CPU (10 SPEC) → ASIC (160Mops) → Memory (2Mb)

Stream in

bus 10Mb/s

Video output

Flow splitting

Motion comp.

Image memory

Video output

ASIC (160Mops)

Memory (2Mb)

CPU (10 SPEC)

Variable-length dec.
Another mapping

- Variable-length dec.
- Flow splitting
- Motion comp.
- IDCT
- Image memory
- Video output
- Video out
- ASIC (160Mops)
- Memory (2Mb)
- CPU (10 SPEC)
- Stream in
- bus 10Mb/s

Performance simulation

- Analyzes performance of behavior on given architecture
- Architectural model provides:
  - performance estimates
  - resource constraints
    - CPU scheduling
    - bus arbitration policy
    - (abstract) cache modeling
Outline

- Problem statement
- Simulation and embedded system design
  - functional simulation
  - performance simulation
    - POLIS implementation
    - partitioning example
  - implementation simulation
    - software-oriented
    - hardware-centric
- Summary

Performance simulation prototype

- Polis
  - High level specification
  - Unified model for both hardware and software (CFSMs)
  - Code generation and estimation
  - Hardware synthesis
- Ptolemy
  - Simulation environment
  - Different simulation models can coexist
  - Nice graphical interface
The POLIS design flow

Performance simulation in POLIS

- Based on synthesized software timing estimates
- Generate C code for both hardware and software components
  - each statement is labeled with the estimated running time
  - accumulate delays during execution of software components
  - use cycle based simulation for hardware components
- Very fast
  - The system is compiled on the host machine
  - Don’t need a model of the processor
Performance estimation

- Current practice: manual guess or detailed simulation
- Software: need restrictions on coding style
  - still requires lots of user inputs
  - can be automated if software is synthesized
- Hardware: need RTL specification or fast behavioral synthesis
- Communication: need communication mapping (and refinement)

```plaintext
BEGIN
  if (detect(c)) goto L1;
  else goto end;
L1:  if (a < b) goto L3;
    else goto L2;
L2:  a := a + 1;
    goto L4;
L3:  a := 0;
L4:  emit(y);
end:  clean_up();
```
CFSM scheduling policy

- Concurrency and shared resources
  - hardware components are concurrent
  - only one software component can be executed at any time
  - if a software component receives an event, but the simulated processor is busy, its execution is postponed
  - need a scheduler to choose among all enabled software processes

- Discrete event simulation with time-stamped events is used to synchronize HW and SW

Car dashboard example
Trade-off evaluation

- Interactively changed simulation parameters
  - Define different aspects:
    - Implementation of each CFSM
    - CPU and clock frequency
    - Scheduler
  - May be inherited in hierarchy
  - Automatically transmitted to following synthesis steps

Trade-off evaluation

- Hw/Sw implementation and partitioning
  - meeting timing constraints
  - trade-offs: speed, code size, chip area

- Scheduling policies
  - Round Robin
  - Pre-emptive and non pre-emptive

- CPU selection
  - MC 68HC11, MC 68332, MIPS R3000

- Don’t need to recompile the system
Trade-off evaluation

- **Input patterns**
  - Impulse, clock, random
  - Sliders, buttons
  - Waveforms from file

- **Monitoring the system**
  - Processor utilization and task scheduling charts
  - Missed deadlines
  - Cost of implementation
  - Internal values

Performance evaluation

<table>
<thead>
<tr>
<th>Target proc.</th>
<th>MHz</th>
<th>Part.</th>
<th>Wheel &amp; Engine</th>
<th>Missed</th>
</tr>
</thead>
<tbody>
<tr>
<td>68HC11</td>
<td>4</td>
<td>1</td>
<td>260-8000</td>
<td>0</td>
</tr>
<tr>
<td>68HC11</td>
<td>1</td>
<td>1</td>
<td>180-6000</td>
<td>900</td>
</tr>
<tr>
<td>68HC11</td>
<td>4</td>
<td>2</td>
<td>50-3000</td>
<td>5000</td>
</tr>
<tr>
<td>68332</td>
<td>20</td>
<td>3</td>
<td>50-3000</td>
<td>30</td>
</tr>
<tr>
<td>68332</td>
<td>40</td>
<td>3</td>
<td>260-8000</td>
<td>20</td>
</tr>
</tbody>
</table>

Partition 1: HW={Timing unit, PWM drivers} SW={Data Processing}
Partition 2: HW={Timing unit} SW={Data processing, PWM drivers}
Partition 3: HW={} SW={Timing unit, Data processing, PWM drivers}
**Simulation speed**

<table>
<thead>
<tr>
<th>Target proc.</th>
<th>MHz</th>
<th>Part.</th>
<th>Graph.</th>
<th>cycles/sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>68HC11</td>
<td>4</td>
<td>3</td>
<td>No</td>
<td>&lt;20000</td>
</tr>
<tr>
<td>68HC11</td>
<td>4</td>
<td>2</td>
<td>No</td>
<td>650000</td>
</tr>
<tr>
<td>68HC11</td>
<td>4</td>
<td>1</td>
<td>No</td>
<td>550000</td>
</tr>
<tr>
<td>68HC11</td>
<td>4</td>
<td>2</td>
<td>Yes</td>
<td>100000</td>
</tr>
</tbody>
</table>

Partition 1: HW={Timing unit, PWM drivers} SW={Data Processing}
Partition 2: HW={Timing unit} SW={Data processing, PWM drivers}
Partition 3: HW={} SW={Timing unit, Data processing, PWM drivers}

**Design flow**

1. **Behavior capture**
2. **Architecture capture**
3. **Mapping (partitioning)**
4. **Performance simul.**
5. **Cycle-based, logic simul.**
6. **Synthesis, coding**

- Functional simul.
**Architecture refinement**

- CPU (10 SPEC)
- ASIC (160 Mops)
- Memory (2Mb)

Bus 10Mb/s

- 486 (66 MHz)
- ASIC (30 Kgates 66 MHz)
- Memory (2Mb)

PCI bus 132 Mb/s

**Architecture simulation**

- Performed on refined model of the architecture, ignoring behavior
- Simulation verifies interface correctness
  - bus-functional model for processors, with random address and data generation (Logic Modeling, etc.)
  - cycle-based or logic simulation for the hardware interfaces
- Interface refinement simulation (Rowson et al., Hines et al.)
**HW and SW Modeling Techniques**

- **Hardware Centric**
  - HW
  - SW & Proc HW

- **Software Oriented**
  - SW
  - hw

**Bus Functional Model**

- **command interpreter**
  - executing memory bus access cycles
  - drives all relevant processor pins
  - does not implement the complete instruction set of the processor

**C/C++**

- HDL language and/or simulator binding

**HDL**

- data
- rd
- wr
Refined mapping

- Variable-length dec.
- Flow splitting
- Motion comp.
- IDCT
- Image memory
- Video output

ASIC (30Kgates, 66 MHz)
Memory (2Mb)

486 (66 MHz)
PCI bus 132 Mb/s

Design flow

- Functional simul.
- Behavior capture
- Architecture simul.
- Architecture capture
- Performance simul.
- Mapping (partitioning)
- Cycle-based, logic simul.
- Synthesis, coding
Synthesis and coding

- Implement functional specification on architecture
- Ideally should be automated
  (e.g. if function for HW is specified as RTL...)
- In practice generally a mix of hand design and synthesis

Outline

- Problem statement
- Simulation and embedded system design
  - functional simulation
  - performance simulation
    - POLIS implementation
    - partitioning example
  - implementation simulation
    - software-oriented
    - hardware-centric
- Summary
Implementation simulation

- Necessary because:
  - Need detailed performance analysis
  - Hand design is error-prone

- Most errors should have been caught before

- How to use test vectors and compare results between different abstraction levels?

HW and SW Modeling Techniques

- Hardware centric
- Software oriented
- Functional
Software Oriented

- combine program and data memory with the processor into a single model
- implement instruction fetch, decode, and execution cycle in opcode interpreter in software

Clock Cycle Accurate Instruction Set Architecture Model

- model derived/genereted from the instruction set description

```c
ADD R1, R2, R3
{ A1.OP = 0x01
 A1.RA = R1
 A1.RB = R2
 ... }
{ R3<- ADD(R2,R1) }
```
Clock Cycle Accurate Instruction Set Architecture Model

- hardware devices hanging off the memory bus
- memory read and write cycles are executed when specific memory addresses are being referenced

Instruction Cycle Accurate Instruction Set Architecture Model

- same basic idea as Clock Cycle Accurate ISA Model
  - processor status can only be observed at instruction cycle boundaries
  - usually no notion of processor pins, besides serial and parallel ports
  - can be enhanced to drive all pins, including the memory interface
    - Clock Cycle Accurate ISA Model
**Virtual Processor**

- C/C++ code is compiled and executed on a workstation
  - accuracy of numeric results may be an issue

- Hardware devices hanging off the memory bus
  - additional function calls to execute memory read and write cycles when specific memory addresses are being referenced can be inserted in the original software
  - need estimation and synchronization with hardware simulation to model software timing

**HW and SW Modeling Techniques**

- Hardware-centric
- Software oriented
- Functional
Possible descriptions of the hardware:
- actual hardware
- gate level netlist
- clock cycle accurate architectural description

Actual Hardware:
- processor is plugged into a device connected to a workstation
- discrete event simulator just sees another component
- software image is loaded into memory and clock is applied
**Gate Level Netlist**

- Gate level netlist executed in a discrete event verification environment
- **exact timing of the signals at all input and output pins**
- **full visibility of all internal signals and storage**

**Clock Cycle-Accurate Architectural Description**

- Architectural description executing in a cycle based verification environment
- **model delivers at each clock edge a set of stable output signals given a set of stable input signals**
- **visibility of internal signals and storage depends on degree of model refinement**
Summary

- Functional model: VERIFY FUNCTIONALITY
- Architecture model: VERIFY INTERFACES
- Performance model: VERIFY PERFORMANCE
- Implementation model: VERIFY ABSTRACTIONS

Conclusions

- Abstraction key to speedup
- Separate behavior (functionality), communication and timing
- Architecture refinement essential for true co-design and fast co-simulation
- Software and hardware synthesis helps performance estimation
- Rapid prototyping may be the ultimate co-simulation tool...