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

GSM Phone

digital

analog

Logic

RAM & ROM

900MHz

70MHz

10.7MHz

SH

SAW FILTER

RF IF

40Ms/sec - 540ks/sec

Viterbi

Eqls.

demod

and

sync

BB

phone

book

keypad

intfc

protocol

control

speech quality

enhancement

voice

recognition

speech

decoder

RPE-LTP

speech

decoder

speech

quality

enhancement

voice

recognition

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

SourceModels

SmartModels

DesignWare

Hardware Modeler

Modeling and Simulation Techniques
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

- Functional simul.
- Behavior capture
- Architecture simul.
- Architecture capture
- Performance simul.
- Mapping (partitioning)
- Cycle-based, logic simul.
- Synthesis, coding
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
- MPEG 2 decoder
- Video output

**Functional simulation**

- Timeless (not really co-simulation...)
- Algorithm exploration, functional debugging, virtual prototyping
- Different formal models:
  - control-dominated: CSP, EFSMs, DE, 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

- Accuracy
- Throughput

- Hardware centric
- Software oriented
- Functional

Design flow

- Functional simul.
- Architecture simul.
- Mapping (partitioning)
- Cycle-based, logic simul.

- Behavior capture
- Architecture capture
- Performance simul.
- Synthesis, coding
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
Design flow

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

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.
- IDCT
- Image memory
- Video output

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

Stream in
Video out

bus 10Mb/s
Another mapping

Flow splitting

Motion comp.

IDCT

Image memory

Video output

Stream in

Video out

CPU (10 SPEC)

ASIC (160Mops)

Memory (2Mb)

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 Classic
  - Simulation environment
  - Different simulation models can coexist
  - Nice graphical interface
The POLIS design flow

- Formal Verification
- Partitioning
- Simulation
- Graphical EFSM
- ESTEREL
- Compilers
- CFSMs
- Sw Synthesis
- Inter Synthesis
- Hw Synthesis
- Logic Netlist
- Sw Code + RTOS
- Prototype
- 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
a := a + 1
a := 0
emit(y)
```

```
BEGIN
emit(y)
detect(c)
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();
END
```
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

- Timing generators
- Data processing
- PWM drivers
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

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

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

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

- PCI bus 132 Mb/s
- bus 10 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
- Software oriented

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
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**
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/generated from the instruction set description
- ADD R1, R2, R3
  
  \[
  \begin{align*}
  &A1.\text{OP} = 0x01 \\
  &A1.\text{RA} = R1 \\
  &A1.\text{RB} = R2 \\
  &\ldots \\
  &R3 <- \text{ADD}(R2, R1)
  \end{align*}
  \]
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
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**

- exact timing of the signals at all input and output pins
- full visibility of all internal signals and storage

**Clock Cycle-Accurate Architectural Description**

- 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...