Hardware/Software Co-Design
&
LEON2/3 SystemC Instruction Set Simulator

Luca Fossati
fossati@elet.polimi.it  Luca.Fossati@esa.int

Politecnico di Milano  European Space Agency

1st, April 2010
Outline

1. Hardware/Software Co-Design
2. LEON2/3 IP Model Contract Aim
3. Instruction Set Simulator
4. Results
5. Conclusion
The *Software* Role in Today's *Hardware*

- **Software** is becoming more and more important in payload processing and on-board control:
  - Need for flexibility, complex functionalities, more processing power is available, . . .
**Software** is becoming more and more important in payload processing and on-board control:

- Need for flexibility, complex functionalities, more processing power is available, . . .

Its development must take place concurrently with the **Hardware**:

- *Software adapts to hardware as well as hardware to software*
- Partitioning of functionalities between Hardware and Software not clearly defined at early design stages
The *Software* Role in Today's *Hardware*

- **Software** is becoming more and more important in payload processing and on-board control:
  - Need for flexibility, complex functionalities, more processing power is available, ... 

- Its development must take place concurrently with the **Hardware**:
  - *Software adapts to hardware as well as hardware to software*
  - Partitioning of functionalities between Hardware and Software not clearly defined at early design stages

- **Software** complexity often **dominates the system development** cost and schedule:
  - Concurrency issues in Multi-Processor Systems (e.g. NGMP)
  - Timing measurements necessary to assess real-time properties
  - ...
Virtual Platform:
A Software Model of the Hardware System

Definition
A Virtual Platform is a software based system that can fully mirror the functionality of a target SoC or board.
Virtual Platform:

A Software Model of the Hardware System

**Definition**

A Virtual Platform is a software based system that can fully mirror the functionality of a target SoC or board.

- Aids solving the software design issues . . .
  - Early availability of reference hardware
  - Software development can start before the first hardware prototype is ready
  - Full control, observability, etc. of the modeled hardware system

  . . . but not only:

  - Enables tuning the hardware and determining the right tradeoffs early in the design cycle (i.e. without the need to already have an initial prototype)
  - Enables seamless interconnection of IP models written using appropriate standards (e.g. OSCI SystemC and TLM)
Virtual Platform: a new simulation paradigm

With respect to standard simulators a VP is:

- Configurable:
  - At system level (different architecture of the SoC)
  - At IP level (different configuration of each IP)
With respect to standard simulators a VP is:

- Configurable:
  - At system level (different architecture of the SoC)
  - At IP level (different configuration of each IP)

- Enables Design Space Exploration to determine an optimal architecture
Virtual Platform: a new simulation paradigm

With respect to standard simulators a VP is:

- Configurable:
  - At system level (different architecture of the SoC)
  - At IP level (different configuration of each IP)

- Enables Design Space Exploration to determine an optimal architecture

- Enables Hardware/Software co-design

...
On Going Activities

A VP is mainly composed of three elements:

a. Models of the hardware components
b. Embedded Software
c. Glue, connecting everything together and providing facilities for inspection, debugging, performance tuning, etc.
Virtual Platform

- Processor (e.g. LEON3)
- Interrupt Controller
- Cache
- Memory Controller
- Embedded Memory
- SpaceWire Interface
- Interconnection Network or Bus (e.g. AMBA AXI)

Virtual Platform
- Virtual Platform Infrastructure (VPI)
- Controller, Monitor, Performance Analyzer

Hardware Model

Software Development
- Application
- Compiler
- Binary Image
- OS

Runs on

Luca Fossati (PdM - ESA)
On Going Activities

A VP is mainly composed of three elements:

- Models of the hardware components
- Embedded Software
- Glue, connecting everything together and providing facilities for inspection, debugging, performance tuning, etc.

Three parallel ongoing activities pose the foundations for the VP which will drive the development of future payload processing and on-board control systems:

1. Development of **models of the main IPs** used in most of the LEON based architectures and of the **VP infrastructure** to manage them
2. Development of the **models of the communication IPs** (SpaceWire and, possibly, CAN)
3. Development of the **models of the LEON2 and LEON3 processors**
SystemC and TLM libraries

All activities are based on the OSCI SystemC and TLM standards
SystemC and TLM libraries

All activities are based on the OSCI SystemC and TLM standards

**SystemC: System-Level Specification Standard**

- Implemented as a set of C++ classes
- Uses a C++ compiler to generate an executable simulator
- The idea is to use the same language during high-level and low-level design
SystemC and TLM libraries

All activities are based on the **OSCI SystemC and TLM standards**

**SystemC: System-Level Specification Standard**
- Implemented as a set of C++ classes
- Uses a C++ compiler to generate an executable simulator
- The idea is to use the same language during high-level and low-level design

**Transaction Level Modeling (TLM)**
- Well-established methodology for modeling complex systems (like MPSOCs)
- It separates *communication* from *computation*
- Modules communicate with the rest of the world by performing transactions
  - A transaction is the operation with which two modules exchange data
  - Data is transferred as a data structure
Outline

1. Hardware/Software Co-Design

2. LEON2/3 IP Model Contract Aim
   - Goals
   - Current Status

3. Instruction Set Simulator

4. Results

5. Conclusion
Aim of the contract is:

- Development and Implementation of a SystemC executable model of the LEON2 and LEON3 processors
Overview

Aim of the contract is:

- Development and Implementation of a **SystemC executable model** of the LEON2 and LEON3 processors

- Various accuracy levels are required:
  - Standalone Instruction-Accurate simulator
  - Standalone Cycle-Accurate simulator
  - Loosely/Approximate -timed Instruction Accurate
  - Loosely/Approximate -timed Cycle Accurate
Overview

Aim of the contract is:

- Development and Implementation of a **SystemC executable model** of the LEON2 and LEON3 processors

- Various accuracy levels are required:
  - Standalone Instruction-Accurate simulator
  - Standalone Cycle-Accurate simulator
  - Loosely/Approximate -timed Instruction Accurate
  - Loosely/Approximate -timed Cycle Accurate

- Following **Tools** shall be provided:
  - Debugger
  - Operating-System Emulator
  - Profiler
Models shall be carefully verified for what concerns:

- **Correctness** of the Instruction-Set behavior:
  - Tests on individual instructions
  - Tests on the overall model using synthetic tests and real-world benchmarks

- **Timing accuracy**:
  - Reference model: simulation with TSIM/HW (LEON2) and TSIM (LEON3).
Current Status

- Functionally correct Instruction-/Cycle-Accurate models
- Behavioral testing performed with:
  - 1424 test over the 145 identified ISA instructions
  - 160 synthetic benchmarks for checking the correctness of single instruction patterns (memory access, shift, etc.)
  - 104 real-life applications (MiBench, PowerStone, JPEG, etc.) for checking the overall processor model

Average Execution speed of 7.7 MIPS
Cycle-Accurate model:
Carefully analyzed the VHDL code, since scarce documentation on the pipeline exists
Average Execution speed of 80 KIPS
Timing validation in progress
Testing of the interfaces by integration with external IP models into a Virtual Platform
Current Status

- **Functionally correct Instruction-/Cycle-Accurate models**
- Behavioral testing performed with:
  - 1424 test over the 145 identified ISA instructions
  - 160 synthetic benchmarks for checking the correctness of single instruction patterns (memory access, shift, etc.)
  - 104 real-life applications (MiBench, PowerStone, JPEG, etc.) for checking the overall processor model
- Average Execution speed of **7.7 MIPS**
Current Status

- Functionally correct Instruction-/Cycle- Accurate models
- Behavioral testing performed with:
  - 1424 test over the 145 identified ISA instructions
  - 160 synthetic benchmarks for checking the correctness of single instruction patterns (memory access, shift, etc.)
  - 104 real-life applications (MiBench, PowerStone, JPEG, etc.) for checking the overall processor model
- Average Execution speed of 7.7 MIPS
- Cycle-Accurate model:
  - Carefully analyzed the VHDL code, since scarce documentation on the pipeline exists
  - Average Execution speed of 80 KIPS
- Timing validation in progress
Current Status

- **Functionally correct Instruction-/Cycle- Accurate models**
- Behavioral testing performed with:
  - 1424 test over the 145 identified ISA instructions
  - 160 synthetic benchmarks for checking the correctness of single instruction patterns (memory access, shift, etc.)
  - 104 real-life applications (MiBench, PowerStone, JPEG, etc.) for checking the overall processor model
- Average Execution speed of 7.7 MIPS
- Cycle-Accurate model:
  - Carefully analyzed the VHDL code, since scarce documentation on the pipeline exists
  - Average Execution speed of 80 KIPS
- Timing validation in progress
- Testing of the interfaces by integration with external IP models into a Virtual Platform
Outline

1. Hardware/Software Co-Design

2. LEON2/3 IP Model Contract Aim

3. Instruction Set Simulator
   - Overview
   - Generated Simulator
   - Code Structure
   - Tools

4. Results

5. Conclusion
Processor modeling performed using *automatic code generation* starting from a high level model

- **5 files used for LEON model (5K lines of Python code), containing:**
  - **Architecture Structure:**
    - List of storage elements (registers, memories, etc.)
    - List of pipeline stages
    - Detailed hardware structure is ignored
  - **Instructions Encoding:**
    - Specify how the bits of the machine code relate to the instruction parts
    - which bits are the opcode, which one identify the operands, . . .
  - **Instructions Behavior (split into 2 files):**
    - C++ code implementing the behavior of each instruction
    - Behavior separated among the different pipeline stages
  - **Instructions Tests:**
    - Enables separate tests for each instruction
    - We specify the processor status before the execution of the instruction and the *expected* status after the execution
Instruction Set Simulator

From the model description, TRAP (our code generator) creates:

- C++ code implementing the simulator itself

- Lines of code:
  - Functional Model 20K (21 files)
  - Cycle Accurate Model 90K (23 files)
  - Instruction Tests 110K

- Implementing an average of 300 distinct C++ classes
Instruction Set Simulator

From the model description, TRAP (our code generator) creates:

- C++ code implementing the simulator itself
- Compilation scripts

Currently working under Unix Operating Systems (Linux, Mac OSX, Cygwin)
Instruction Set Simulator

From the model description, TRAP (our code generator) creates:

- C++ code implementing the simulator itself
- Compilation scripts
- Tests of the single instructions

- Each single instruction tested with an average of 9 tests
- Tested the correct decoding of randomly-selected instruction patterns
Instruction Set Simulator

From the model description, TRAP (our code generator) creates:

- C++ code implementing the simulator itself
- Compilation scripts
- Tests of the single instructions

- Each single instruction tested with an average of 9 tests
- Tested the correct decoding of randomly-selected instruction patterns

TRAP libraries (4.5K lines of code)

- GDB debugger server
- Operating-System emulator
- Object file loader
- profiler
Code Structure

Created code is written in C++ and it makes extensive use of object oriented features of the language

Most Important Data Structures

- **Register**
- **Alias** ease access to registers, working like a hardware mux
- **Instruction** with its subclasses, implements the actual behavior of the Instruction Set
- **Processor**: the entity which glues everything together, containing the registers and calling the instruction behaviors.
- **Pipeline Stages**: each one is a separate SystemC thread concurrent with the others
- **Decoder**, translating the instruction word into the appropriate class and the actual behavior.
- **External Pins**, e.g. the interrupt port for receiving incoming interrupts
- **Memory Ports**, for communication with caches, memories, busses, etc
- **Tools**, such as debugger, profiler, Operating System emulator, etc.
Tools

Analysis and Debugging Tools

- Without analysis tools, simulators are of limited usefulness
- Commonly used tools are debuggers, profilers, etc.
- Simple means for integrating new tools by decoupling the simulator from the tool through a well defined interface are provided

Default tools (part of every generated model):
- Debugger: connects via network to standard GNU/GDB debugger
- Profiler: keeping statistics on the software running in the processor model
- Operating System emulator: enables execution of bare applicative software by forwarding every supervisor call to the host OS.
Analysis and Debugging Tools

- Without analysis tools, simulators are of limited usefulness

- Commonly used tools are debuggers, profilers, etc.

- Simple means for integrating new tools by decoupling the simulator from the tool through a well defined interface are provided

- Default tools (part of every generated model):
  - Debugger: connects via network to standard GNU/GDB debugger
  - Profiler: keeping statistics on the software running in the processor model
  - Operating System emulator: enables execution of *bare applicative software* by forwarding every supervisor call to the host OS.
Outline

1. Hardware/Software Co-Design
2. LEON2/3 IP Model Contract Aim
3. Instruction Set Simulator
4. Results
   - Execution Speed
   - Comparison with TSIM
5. Conclusion
Execution Speed: comparison among different models

![Execution Speed Chart]

Luca Fossati (PdM - ESA)  
Hw/Sw co-design & LEON ISS  
1st, April 2010  
20 / 25
## Comparison with TSIM

<table>
<thead>
<tr>
<th></th>
<th>TSIM</th>
<th>LEON2/3 ISS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>scope</strong></td>
<td>full system</td>
<td>integer unit</td>
</tr>
<tr>
<td><strong>interfaces</strong></td>
<td>self-contained (custom for GRSIM)</td>
<td>IEEE standard (OSCI SystemC and TLM)</td>
</tr>
<tr>
<td><strong>speed</strong></td>
<td>up to 45 MIPS (5 MIPS for GRSIM)</td>
<td>up to 12 MIPS</td>
</tr>
<tr>
<td><strong>tools</strong></td>
<td>full set (debugger, profiler, instruction trace, etc.)</td>
<td>Software Development</td>
</tr>
<tr>
<td><strong>target</strong></td>
<td><strong>Software Development</strong></td>
<td>Hardware Optimization</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Architecture Exploration</td>
</tr>
</tbody>
</table>
Outline

1. Hardware/Software Co-Design
2. LEON2/3 IP Model Contract Aim
3. Instruction Set Simulator
4. Results
5. Conclusion
   - Development Status
   - Areas to be Improved
Development Status

- Functional and Cycle-accurate Simulator behaviorally correct
  - Including support for Hardware/Software analysis tools (OS emulation, GDB server, and profiler)
- Different versions:
  - standalone, including an internal memory
  - using memory ports with different accuracy levels
  - with or without instruction tracing capabilities
- Compiles under unix environments
- Cygwin is necessary for the use under Windows
Areas to be Improved/Future Work

- **Simulation speed:**
  - concentrating on instruction decoding
  - cycle-accurate: propagation of registers in the pipeline, stages synchronization mechanisms
  - profiler
- **Integration in a Virtual Platform** to carefully test TLM interfaces.
- **Improvement of the tools**
  - Support of additional GDB commands
  - Emulation of pthread routines in addition to standard OS ones
- **Native support for compilation/execution under Microsoft Windows**
Further Information

TRAP development website together with processor models (LEON, ARM, MicroBlaze, etc.), maintained by Politecnico di Milano: http://trap-gen.googlecode.com

More information on the IP models, the Virtual Platform, etc., soon available on the ESA Microelectronics Website http://www.esa.int/TEC/Microelectronics/