LEON2/3 SystemC Instruction Set Simulator

Luca Fossati
Luca.Fossati@esa.int

European Space Agency
Outline

1. LEON2/3 IP Model Contract Aim
2. Instruction Set Simulator
3. Results
4. 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 time-timed Instruction Accurate
  - Loosely/Approximate time-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** are provided:

- Debugger
- Operating-System Emulator
- Profiler
Models are 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).
Results

- Functionally and Timing 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
Results

- **Functionally and Timing 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
- **Instruction-Accurate model:**
  - Average Execution speed of 7.7 MIPS
  - Timing accuracy: 99.8%
Results

- Functionally and Timing 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
- Instruction-Accurate model:
  - Average Execution speed of 7.7 MIPS
  - Timing accuracy: 99.8%
- Cycle-Accurate model:
  - Average Execution speed of 80 KIPS
  - Timing accuracy: 100%
Results

- Functionally and Timing 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
- Instruction-Accurate model:
  - Average Execution speed of 7.7 MIPS
  - Timing accuracy: 99.8%
- Cycle-Accurate model:
  - Average Execution speed of 80 KIPS
  - Timing accuracy: 100%
- Testing of the interfaces by integration with external IP models into a Virtual Platform
1 LEON2/3 IP Model Contract Aim

2 Instruction Set Simulator
   - Overview
   - Generated Simulator
   - Code Structure
   - Tools

3 Results

4 Conclusion
TRAP: Code Generator

ISA Behavior
ISA Encoding
Architecture Description
TRAP API
C++ Writer
Python

Instruction Accurate
Cycle Accurate
Retargetable debugger
Retargetable Compiler

Standalone
TLM based
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, ...
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:**
  - **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
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)
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
- Object file loader
- Operating-System emulator
- Profiler
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.
1. LEON2/3 IP Model Contract Aim

2. Instruction Set Simulator

3. Results
   - Execution Speed
   - Comparison with TSIM

4. Conclusion
Execution Speed: comparison among different models

![Bar chart showing execution speed comparison among different models.

- **standalone**: 7.74 MIPS
- **funLT**: 4.94 MIPS
- **funA**: 2.66 MIPS
- **accLT**: 0.58 MIPS
- **accA**: 0.082 MIPS
- **time warp funLT**: 0.073 MIPS

The chart displays the execution speed in MIPS for each model, with error bars indicating variability.
## 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></td>
</tr>
<tr>
<td><strong>target</strong></td>
<td>Software Development</td>
<td>Software Development Hardware Optimization</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Architecture Exploration</td>
</tr>
</tbody>
</table>
Outline

1. LEON2/3 IP Model Contract Aim
2. Instruction Set Simulator
3. Results
4. 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, maintained by Politecnico di Milano and the OpenSource community:
  
  http://trap-gen.googlecode.com

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