Assertion and Model Checking of SystemC

Sofiène Tahar

Hardware Verification Group
Department of Electrical and Computer Engineering
Concordia University
Montreal, Quebec, CANADA

First Annual North American SystemC Users Group (NASCUG) Meeting
June 07, 2004
Outline

SystemC Verification: State-of-the-art
Proposed Verification Approach
Model Checking
Assertion Based Verification
Conclusion
SystemC Verification Problematic

SystemC Verification:
- Functional verification of SoC design (in particular SW) is difficult.
- Checking each feature/subsystem separately is not enough to ensure correct operation.
- Verifying SystemC under construction only makes things worse: adding HW/SW immaturity issues.

State-of-the-art:
- No relevant new techniques.
- Adapting existing methodologies using:
  - Assertion based verification (ABV).
  - Model checking.
  - Guiding test vector generation (functional coverage).
Proposed Solution for SystemC Verification

- SystemC Code Guide
- Guide the Abstraction
- USER
  - Properties
  - Test Benches
  - Assertions

Reduced Hypergraph
- Confined Hypergraph
  - Junction Type
  - Disjunctive Type
  - Conjunctive Type
- Entry Point
  - Entry Branch
  - Distribution Point
  - Distribution Branch
  - Junction Branch
  - Junction Point
- Output Type
  - Output Branch
  - Output Point

Conjunctive Type
- Junction Type
- Output Type

Disjunctive Type
- Distribution Branch
- Distribution Point

Confined Hypergraph
- PSL

Junction Type
- Junction Branch
- Junction Point

Output Type
- Output Branch
- Output Point

Conjunctive Type
- Conjunctive Branch

Disjunctive Type
- Disjunctive Branch
Case Study: Bus Structure

- Clock
- Master1
- Master2
- Master3
- Bus
- Slave 1
- Slave 2
- Arbiter
Snapshot of the Hypergraph

EventsStack
- master_direct.main_action()
- fast_memory.main_action()
- slow_memory.main_action()
- master_blocking.main_action()
- master_nonblock.main_action()

EventManager
- clk
- arbiter.arbitrate()
- sbus.main_action()

Stack
- Id
- l1
- l2
- l3
- ...

Env
- Idp

Simulation Manager
- master1
- master2
- arbiter
- slave1
- slave2

Event’s Environment

Program Environment
Static Code Analysis

Static code analysis:

- Verify some properties statically.
- Simplify the system structure.
- Offer an interactive and graphical platform.

Hypergraph statically executed:

- Extract a reduced representation.
- Verify some of the system’s properties (infinite loops, etc.)

Reduced hypergraph:

- Translated to a format supported by the model checker.
- Used in abstract debugging.
- Executed to get “simplified” graphical snapshots of the system during execution.
Model Checking: A Simple Approach

- SystemC Code
- Guide the Abstraction

Diagram:
- Static Code Analyzer
  - Hypergraph to ASM
    - PSL (ASM) to C#
    - ASM to FSM
      - PSL
      - Model Checker
        - Test Bench Generator
          - Assertions
          - Verifier

PSL

USER

Properties

Test Benches

Assertions
Model Checking: A Simple Approach

SystemC Design

Static Code Analyzer

Reduced Hypergraph

Hypergraph to Verilog Converter

Model Checker (FormalCheck)

Design in Verilog

Verification terminates (property verified or failed) or never terminates (state explosion)
Model Checking: An ASM Based Approach

- SystemC Code
- Guide the Abstraction
- USER
- Static Code Analyzer
- Hypergraph to ASM
- PSL (ASM) to C#
- ASM to FSM
- PSL
- Model Checker
- Test Bench Generator
- Properties
- Test Benches
- Assertions
Model Checking: An ASM Based Approach

SystemC Design

C++ Compiler

Hypergraph Generator

Hypergraph to ASM Translator

Design Modeled in ASM

SystemC Simulator Semantics (ASM)

AsmL Tool (AsmL Compiler)

System’s FSM

FSM to Model Checker Input Language Translator

Model Checker

PSL Property

Assertion Parser

ASM/PSL Property Generator

PSL Semantics (ASM)

PSL Property Modeled in ASM
ABV integration on top of SystemC:

- External monitors: easy to integrate and re-use.
- Start with SystemVerilog Assertion (SVA) then extend to Sugar’s assertions.

ABV verification:

- Needs good coverage.
- Requires guiding the test vector’s generation.
- Yields very large system state space.
ABV: A Simple Approach

**SystemC Code**

- **Static Code Analyzer**
  - Hypergraph to ASM
  - PSL (ASM) to C#
  - ASM to FSM
  - PSL

- **Model Checker**
  - Test Bench Generator

- **Assertions Verifier**

**Guide the Abstraction**

**USER**

- Properties
- Test Benches
- Assertions
ABV: A Simple Approach

SystemC Design

GCC Compiler

Table of Symbols

Design Updater

Updated Design

Assertion Integrator

SystemC updated design containing the assertion’s monitor
ABV: An Enhanced Approach

1. Dependency Relations
2. Inputs Ranges

Hypergraph Generation → Reduced Hypergraph → Dependency Check → Initial DNA Generation → DNA Evaluation → DNA Update → Final Generator’s DNA
ABV: An ASM Based Approach

SystemC Code

Static Code Analyzer

Hypergraph to ASM

PSL (ASM) to C#

ASM to FSM

PSL

Model Checker

Test Bench Generator

Assertions Verifier

Guide the Abstraction

USER

Properties

Test Benches

Assertions
ABV: An ASM Based Approach

- SystemC design
- GCC compiler
- Design updater
- Updated design
- Assertion integrator
- SystemC updated design containing the assertion’s monitor

- PSL Assertion
- Assertion Parser
- ASM/PSL Property Generator
- AsmL Tool (AsmL Compiler)

- List of symbols
- Table of symbols

- PSL Semantics (ASM)
- PSL Assertion in C#
Related Work

Finite-state verification of software:

- **JAVA**: BANDERA [Kansas University]: provides support for the extraction of safe, compact, finite-state models suitable for verification.
- **C**: SLAM [Microsoft], BLAST [University of Berkeley], etc.

Static code analysis of object-oriented languages:

- [Ferdinand'96]: Abstract debugging.
- **OO** languages abstraction:
  - [Chambers‘90] : stack allocation and synchronization (Java programs)
  - [Vederine‘00]: a platform for total analysis (both C++ and ML).
Case Study: Bus Structure

Clock → Master1 → Bus → Master2 → Bus → Master3 → Arbiter

Bus → Slave 1

Bus → Slave 2
Bus Verification

Property 1:

\[
\text{NEVER}( (\text{simple\_bus.request} == \text{true}) \land (\text{simple\_bus.status} != \text{Bus\_OK}) )
\]

Property 2:

\[
\text{AFTER}( (\text{simple\_bus.request} == \text{true}) \land (\text{simple\_bus.request.block} == \text{Bus\_OK}) )
\]

\[
\text{EVENTUALLY} (\text{simple\_bus.status} == \text{BUS\_BLOCK})
\]

Property 3:

\[
\text{EVENTUALLY} (\text{simple\_bus.status} == \text{BUS\_OK})
\]

Direct verification using FormalCheck: \text{failed to complete} after few minutes with out of memory problem!
Snapshot of the Hypergraph

**EventsStack**

- master_direct.main_action()
- fast_memory.main_action()
- slow_memory.main_action()
- master_blocking.main_action()
- master_nonblock.main_action()

**EventManager**

- clk
- arbiter.arbitrate()
- sbus.main_action()

**Simulation Manager**

- master1
- master2
- arbiter
- slave1
- slave2

**Program Environment**

- Stack
- Id₁
- Id₂
- Id₃

**Event’s Environment**

- Program Environment
- Event’s Environment
The main reductions concern:

- Transforming the SystemC simulator into a while loop.
- Reducing the packet to its packet header.
- Reducing some of the variables sizes.

<table>
<thead>
<tr>
<th>Property</th>
<th>CPU Time</th>
<th>Memory (in MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>6:59:12</td>
<td>93.59</td>
</tr>
<tr>
<td>P2</td>
<td>15:23:02</td>
<td>183.91</td>
</tr>
<tr>
<td>P3</td>
<td>17:46:54</td>
<td>293.63</td>
</tr>
</tbody>
</table>
Assertions Definition

**Assertion 1:**

```markdown
assert property1 @ (posedge clk) (st == ack) && (flag == 1) \rightarrow !req[*8])
```

**Assertion 2:**

```markdown
assert property2 @(posedge clk) (simple bus.request == true) &&
  (master1.active||master2.active||master3.active)
```

**Assertion 3:**

```markdown
assert property3 @(posedge clk) (simple bus.request == true) &&
  (simple bus.request.nonblock == true)
  \rightarrow simple bus.status == BUS OK[*1])
```
Coverage as Function of the Generation

Coverage as Function of the Generation

Coverage (in %)

Generation
## Genetic versus Random Coverage

<table>
<thead>
<tr>
<th></th>
<th>Assertion 1</th>
<th>Assertion 2</th>
<th>Assertion 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random Test (%)</td>
<td>10</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>Initialization of GA (%)</td>
<td>34</td>
<td>42</td>
<td>32</td>
</tr>
<tr>
<td>GA after 35 Iterations (%)</td>
<td>92</td>
<td>93</td>
<td>85</td>
</tr>
</tbody>
</table>

- Static code analysis refines the space of possible values.
- GA allows to offer better coverage of the assertions (more than 90% for Assertion 1).
- Mutation mechanism: overcomes the local maxima problem.
ABV Monitors Output

SystemC updated design containing the assertion's monitor

- Verification Report
- Exceptions or Warnings
- Notification Signals
SystemC functional verification:
- Difficult for complex SoCs.
- Verification must consider IPs and their interaction.
- SystemC software functionality testing is inherently difficult.

Need to develop SystemC verification methodologies.

Our solution: combination of several techniques:
- Static code analysis.
- Model checking.
- Assertion based verification.
- Interface SystemC to existing tools through ASM semantics.
For any further details, visit the project webpage at:
System-on-Chip Verification <http://hvg.ece.concordia.ca/Research/SoC/>

Thanks!

Hardware Verification Group
2004