Simplified stimuli generation for scenario and assertion based verification

Luca Piccolboni, Graziano Pravadelli
University of Verona, Italy
Orienteering...

• Requires navigational skills using a **compass** to navigate from point to point in diverse and usually unfamiliar **terrain**, moving at speed.
• Participants are given a **map**, which they use to find control points.
Stimuli generation...

- Requires navigational skills using a **metrics** to navigate from point to point in diverse and usually unfamiliar **DUV**, moving at speed.

- Participants are given a **test plan**, which they use to find control points.
Bad strategy for orienteering…

- Exploring all the terrain
  - i.e., using a structure-oriented coverage-driven (SOCD) stimuli generation
    - independently from a specific objective
    - only to achieve a high DUV coverage

- Scenario-based and assertion-based verification require a smarter orienteering strategy
Outline

• Introduction
  – Scenario- and assertion-based verification
  – Constraint-based stimuli generation
  – Goal

• The proposed framework
  – Stimuli specification
  – Stimuli generation
  – Comparison with Universal Verification Methodology

• Experimental results
• Concluding remarks
SBV and ABV

• A *scenario* is a story that describes a hypothetical situation. In testing, you check how the program copes with this hypothetical situation. 

(C. Kaner, “What is a good test case”)

• ABV is a methodology in which designers use *assertions* to capture design intents, and verify that the design correctly implements them.

(Mentor Graphics)
Test plans

- SBV and ABV follow test plans that require to bring the DUV into particular states where it is possible to check specific conditions.
Orienteering… with an SOCD stimuli generator?

- Bring the mixer machine in a situation where:
  - $T_{\text{cold}}=5^\circ\text{C}$, $T_{\text{pipe}}=20^\circ\text{C}$, $T_{\text{hot}}=40^\circ\text{C}$, and $T_{\text{set}}=25^\circ\text{C}$
  - then linearly increase $T_{\text{pipe}}$ till 30°C
  - then activate the water discharge
  - and check if the final water flow is composed only by water arriving from the water supply network and from the hot water container
Constraint-based stir

- A better choice for SBV and ABV…

- … but often too complex
Goal

- Framework to overcome UVM, OVM, SCV, etc. complexity
  - based on a *simple specification language* to define constraints for SBV and ABV
  - independent from the DUV language
  - working for both embedded SW and HW
THE PROPOSED FRAMEWORK
Stimuli specification

\[
\begin{align*}
\langle \text{constant} \rangle & ::= \text{‘Constant’} \langle \text{Value} \rangle \\
\langle \text{uniform} \rangle & ::= \text{‘Uniform’} \langle \text{Range\_Min} \rangle \langle \text{Range\_Max} \rangle \\
\langle \text{sum} \rangle & ::= \text{‘Sum’} \langle \text{generator} \rangle \langle \text{generator} \rangle \\
\langle \text{product} \rangle & ::= \text{‘Product’} \langle \text{generator} \rangle \langle \text{generator} \rangle \\
\langle \text{uminus} \rangle & ::= \text{‘UMinus’} \langle \text{generator} \rangle \\
\langle \text{function} \rangle & ::= \text{‘Function’} \langle \text{Ftype} \rangle \langle \text{Initial\_Value} \rangle \\
& \quad \langle \text{Offset} \rangle \quad \text{Ftype(Initial\_Value + t * Offset)} \\
\langle \text{constraint} \rangle & ::= \text{‘Constraint’} \langle \text{constraint\_list} \rangle \\
& \quad \text{‘end’} \langle \text{Expression} \rangle \text{‘;’}
\end{align*}
\]
Stimuli specification (cont.)

\[
\begin{align*}
\langle reference \rangle & ::= \text{‘Reference’ } \langle Name \rangle \\
\langle sequence \rangle & ::= \text{‘Sequence’ } \langle Do\_Loop \rangle \\
& \quad \langle sequence\_list \rangle \text{‘end’} \\
\langle delay \rangle & ::= \text{‘Delay’ } \langle Delay\_Amount \rangle \\
& \quad \langle Initial\_Value \rangle \text{‘end’ } \langle generator \rangle \\
\langle range\_restrict \rangle & ::= \text{‘Range\_Restrict’ } \langle Range\_Min \rangle \\
& \quad \langle Range\_Max \rangle \langle generator \rangle \\
\langle time\_expand \rangle & ::= \text{‘Time\_Expand’ } \langle Factor \rangle \\
& \quad \langle generator \rangle \quad g(t) = f(t/k) \\
\langle input \rangle & ::= \text{‘Input’ } \langle Name \rangle \langle Do\_Loop \rangle
\end{align*}
\]
Stimuli generation

• A C++ engine creates a sequence of values by assuming a discrete time model (timestamp)
  – Function generator
    • standard C functions already built-in
    • user defined functions can be added
  – Uniform generator
    • Random Boost library
  – Constraint generator
    • based on SMT-Lib 2.0 library
    • currently MathSAT 5 is used as constraint solver
Example

OUTPUT gen_1
  Sequence 0
  Uniform −1 1 20
  Function sin 0 0.1
  Constant 0.24 1
end

OUTPUT gen_2
  Range_Restrict 0 2
  Function log 1 0.1

OUTPUT gen_3
  Sum
  Constant −0.01
  Delay 1 0.01 end Reference gen_3

\[ f(t) = f(t-1) - 0.01 \]
Comparison with UVM

• To verify that:
  – the machine moves from the state \textit{idle} to the state \textit{Second\_temp} when
  • $T_{\text{cold}} - \Delta A < T_{\text{set}}$
  • $T_{\text{set}} < T_{\text{cold}} + \Delta A$
  • $|\text{deriv}(T_{\text{cold}})| < 0.3$

TSL

19 simple lines

```
OUTPUT tcold Uniform -20 5
OUTPUT tcold\_der
  Constraint
    Constant 0.3 x
  end (assert (< this 0.3));
OUTPUT delta\_A Constant 2
TEMP FLOAT 0 lower_bound
  Sum
    Reference tcold
    UMinus Reference delta\_A
TEMP FLOAT 0 upper_bound
  Sum
    Reference delta\_A
    Reference tcold
OUTPUT tset
  Constraint
    Reference lower_bound low
    Reference upper_bound up
  end (assert (and (< this up) (> this low)));
```

UVM

• About 100 lines of SystemVerilog code
EXPERIMENTAL RESULTS
ABV for embedded SW

<table>
<thead>
<tr>
<th>DUV</th>
<th>NAME</th>
<th>LINES</th>
<th>ASS.</th>
<th>ACT.</th>
<th>TIME</th>
<th>ACT.</th>
<th>DIR.</th>
<th>TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breadmaker</td>
<td>2006</td>
<td>20</td>
<td>30%</td>
<td>0.008s</td>
<td>100%</td>
<td>7</td>
<td>0.064s</td>
<td></td>
</tr>
<tr>
<td>RPC-3D-door</td>
<td>22043</td>
<td>63</td>
<td>98%</td>
<td>0.012s</td>
<td>100%</td>
<td>168</td>
<td>0.132s</td>
<td></td>
</tr>
<tr>
<td>DSC-2L</td>
<td>37545</td>
<td>14</td>
<td>35%</td>
<td>0.004s</td>
<td>100%</td>
<td>114</td>
<td>1.676s</td>
<td></td>
</tr>
</tbody>
</table>

- Initialization sequences required to fire assertions
  - Structure-oriented (coverage driven) ATPG unable to bring the systems in the required state
Fault simulation for HW modules

<table>
<thead>
<tr>
<th>Name</th>
<th>Lines</th>
<th>Faults</th>
<th>St.-Oriented</th>
<th>Our Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADPCM</td>
<td>341</td>
<td>143</td>
<td>75.5%</td>
<td>96.5%</td>
</tr>
<tr>
<td>AM2910</td>
<td>622</td>
<td>156</td>
<td>48.7%</td>
<td>85.3%</td>
</tr>
</tbody>
</table>

• RTL faults to be killed
  – Structure-oriented (coverage driven) ATPG unable to mimic a real program for AM2910
Concluding remarks

- Specification language and generation engine for SBV and ABV
  - Simpler than UVM
  - Independent from
    - abstraction level
    - DUV language
  - Working for
    - embedded SW/HW

Orient your stimuli generation!
Thank You!