A Meta-Modeling-Based Approach for Automatic Generation of Fault-Injection Processes

B.-A. Tabacaru,
M. Chaari, W. Ecker, T. Kruse

Infineon Technologies AG
Outline

- Motivation
- Fault-Injection Theory
- Fault-Injection Requirements
- SCFIT (SystemC Fault-Injection Tool)
- SCFIT Gen
- Results
- Discussion
- Next Steps
Motivation

• Safety-critical applications
  – Dependability analysis
  – ISO-26262 based safety analysis

• System reaction to fault injection
  – Virtual prototyping for early fault analysis
  – Get results before hardware is ready
  – Faster simulation time

• System exploration of fault-tolerant designs
  – Chip area
  – Design complexity/cost
Fault-Injection Theory

Fault-Injection Types
- Hardware
- Software
- Simulation based

Fault-Injection Methods
- Invasive
- Non-Invasive

Fault Attributes
- Time
- Location
- Type

Fault Space

Fault-Injection Tool
Generic Framework

Controller
- Workload Lib
- Fault Lib

Workload Generator
Fault Injector
Target System

Monitor
Data Collector/Analyzer
Fault-Injection Requirements

- SystemC models (e.g., TLM)
  - Separate solutions for FW and RTL already exist
  - Unique solution needed

- Non-intrusive simulation based fault injection

- Access private/protected data members
  - Newly introduced by the C++ coding style

- Separate fault behavior description from original behavior in models
  - Commercial simulators solved this problem for RTL code
SCFIT Architecture

SCFIT Gen

Fault-Injection Modules

- Fault-Injection Processes
- Fault-Injection Objects

SCFIT Simulation Kernel

- Python API
- GDB

SystemC Simulator

SystemC Environment

Verification Environment

User

SCFIT SC Module
SCFIT Flow Diagram

Start → Initialize SCFIT

Start SystemC Simulation

Start SystemC Simulation → 

Stop SystemC Simulation

(De)activate Fault-Injection Processes

(De)activate Fault-Injection Processes → 

(Enable/Disable Break/WATCHPOINTS

Break/WATCHPOINTS Triggered

Break/WATCHPOINTS Triggered → 

Inject Fault(s)
SCFIT Gen GUI

- Tree view of automatically parsed System Top
- Fault-Injection Process references mapped SystemC Port
- User-friendly fault-behavior description
SCFIT Gen Meta-Model
Results

- Simplified CPU architecture
- 12 bit wide registers
- Set of 32 instructions
- RTL model
- TLM reference model

<table>
<thead>
<tr>
<th>SCFIT</th>
<th># of Faults</th>
<th>Simulation Time (s)</th>
<th>Slowdown (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
<td>-</td>
<td>3.71</td>
<td>-</td>
</tr>
<tr>
<td>yes</td>
<td>0</td>
<td>3.77</td>
<td>1.61</td>
</tr>
<tr>
<td>yes</td>
<td>1</td>
<td>4.10</td>
<td>10.51</td>
</tr>
<tr>
<td>yes</td>
<td>2</td>
<td>4.50</td>
<td>21.29</td>
</tr>
</tbody>
</table>
Discussion I

- User-friendly fault-behavior description
- Measured code reduction by a factor of at least forty
  - User-written code
- Inject faults only in system specific variables
  - e.g., cannot change sc_time variable
- Automatic documentation of fault-injection scenarios per test case
Discussion II

Advantages

• Injection of any number of faults per simulation
• Fast simulation speed
  – For one fault per simulation
• Usable with O3 optimization
• User-friendly fault-behavior description
• Usable in regression runs

Disadvantages

• Dramatic slowdown for 2+ faults per simulation
  – Max 4 hardware supported watchpoints
• Compiler debug-mode enabled in SystemC models
  – Not portable to customers
Next Steps

- Automatic fault-detection and propagation analysis
  - Fault $\rightarrow$ Error $\rightarrow$ Failure
  - Simulation report generation
- Windows portability
  - GDB only available under Unix machines
- SCFIT’s Simulation Speed Optimization
  - Migrate interpreted Python code to compiled C++ code
- Integrate SCFIT in project flow
  - Test its performance on bigger SoC designs
Questions

Thank you for your attention!