AUTOMATED FAULT INJECTION FOR SYSTEMC & SYSTEMC AMS

Stephan Gerth, Fraunhofer IIS/EAS
Coside User Group Meeting 2016

© Photo: Sergey Nivens / Fotolia
Current state of fault injection
What limitations do we want to overcome?

- Test a given DUT for e.g.
  - Diagnostic coverage
  - Functional safety
  - Fault tolerance and fault latency
- Solutions available mostly for basic single occurrence faults
  - Stuck-at-*
  - Bit-Flips
Challenges
What do we aim for?

- Large designs
- High-level faults
- Intermittent faults
- Multiple faults scattered throughout the design
- No test artefacts within designs
- Different MoCs
Approach

How do we want it?

- Create an allround solution for fault injection in SystemC & SystemC AMS
- Avoid DUT changes, obey separation of concerns
  - Keep the DUT strictly to its function
  - Don’t introduce artefacts for testing in the DUT code
  - Fault injection should only appear in tests
- Controlled distribution of faults over the DUT
  - Where does occur what kind of fault
  - At which rate does it occur (permanent, intermittent, …)
Solution
How to implement?

- Provide a library with
  - fault injection mechanisms
  - Base models
  - Support for different MoCs
  - Statistical configurability for occurrence rates and region

- Enable fault specializations for
  - high-level faults
  - User specific needs

- Insert faults only dynamically
  - Driven by the testbench
  - Keeping the DUT untouched
Fault injection library
Structured approach open for extensions

- Injection mechanisms for different MoCs
- Crosstalk, Delays, Glitching, Open, Stuck-At
- User specific detailing
- Combination & configuration for specific applications
Dynamic fault insertion
Don’t touch the DUT code base!

- Keep description of faults in tests
- Instrument the DUT during runtime
- Avoid bugs introduced by testing artefacts
- Faults can be active concurrently regardless of their MoC domain
Fault Injection for Electrical Linear Network designs
Stuck-at-value for basic filter

V_{\text{stuck at}} = \text{stuck_at}("*i_{\text{sca_r3.p}}", 1.5);
Fault Injection for Timed Data Flow designs
Cross-talk for PLL model
Fault Injection for Timed Data Flow designs
Cross-talk for PLL model

```cpp
new sin_src_tdf<double>("i_sin_src_tdf1");
fault_scenario_template<double>({"*i_gain_tdf1.tdf_i","*i_sin_src_tdf1.tdf_o"}, FAULT_CROSSTALK, ADD) f;
...
wait(50.0, SC_US); f.activate();
wait(50.0, SC_US); f.deactivate();
wait(150.0, SC_US); f.activate();
wait(100.0, SC_US); f.deactivate();
```
Fault Injection for digital designs

Multiple faults at simple SPI transmissions

- const source
- SPI_MASTER
- SPI_SLAVE
- i_spi_master1
- i_spi_slave1
- CLK_SRC_SC
- i_clk_src_sc1
- OUT
- i_const_src_sc1
- CONST_SRC_SC<1>
- sc_o
- rstin
- rstin_in
- file_name = "/spi.simp"
- timeout = sc_time(1.0, SC_MS)

Graphical representation of signal behavior over time.
Fault Injection for digital designs

Multiple faults at simple SPI transmissions

```cpp
fault_scenario_template<bool> ({{"*i_clk_src_sc1.clk_o","*i_spi_slave1.csq_in"},FAULT_CROSSTALK,REPLACE);
fault_scenario_template<bool> ({{"*i_spi_slave1.sclk_in"},FAULT_STUCK_AT,true});
...
wait(stats::exponential(10.0e-6, 0.0));
fault_scenarios [stats::exponential(fault_scenarios.size())]
 .activate(sc_time(3.0,SC_US));
```
Fault Injection for digital designs

Multiple faults at simple SPI transmissions (2)
Fault Injection for digital designs
Multiple faults at simple SPI transmissions (2)

```cpp
for (sc_object* obj: get_matching_objects("*mosi")) {
  ...
  fault_scenario_template<bool> ({*obj}, FAULT_STUCK_AT, true) f;
  f.configure (exponential, location_fct, sc_time(30.0, SC_US));
}
```
Project application (IKEBA): Battery management system
Handling of aging and faulty battery cells

- Battery model consisting of several battery cells
- Daisy chained monitoring logic
- Controlled by programmable µC
- Testing of
  - Reliable handling of aging and faulty battery cells
  - Battery monitoring ICs
Project application (IKEBA): Battery management system
Handling of aging/faulty battery cells
Project application (IKEBA): Battery management system

Battery monitoring IC testing

1. Ramp generator for checking ADC value range
2. Adding non-ideal behaviour
Further application areas

Other contexts

- ISO 26262
  - Low- & High-level hardware faults
  - Software safety on unreliable hardware
  - HiL testing

- Parallelisation
  - Improve coverage related testing
  - Almost for free
Outlook

- Extension of concurrent handling of different fault types
  - Currently some limitations apply
- Fault injection control via external input
  - Scripting
  - Coside Plug-In
- Adding more generic fault models
THANK YOU FOR YOUR ATTENTION

YOUR CONTACTS

Roland Jancke
Head of Department
Design Methodology

Roland.Jancke@eas.iis.fraunhofer.de
+49 351 4640-747

Stephan Gerth
Group Manager
Functional Modeling and Verification

Stephan.Gerth@eas.iis.fraunhofer.de
+49 351 4640-847

Fraunhofer-Institute for Integrated Circuits IIS
Division Engineering of Adaptive Systems EAS
Zeunerstraße 38
01069 Dresden

www.eas.iis.fraunhofer.de