

DAVOS toolkit: simulation-based fault injection (SBFI)

Quick Start Guide

Ver. 1.0

Contents

PREREQUISITES	2
SETTING-UP A SIMULATION-BASED FAULT INJECTION EXPERIMENT	2
EXECUTING THE FAULT INJECTION EXPERIMENT.....	5
EXPLORING AND VISUALIZING SBFI RESULTS	6

Ilya Tuzov

Universitat Politècnica de València

Valencia, 2023

Prerequisites

DAVOS SBFI Tool requires:

- [mandatory] python ver. 2.x (basic distribution)
- [mandatory] ModelSim/Questa simulator, added to the path environment variable
- [mandatory] ModelSim/Questa simulation checkpoint file for the design under test
- [optional] A web-server (e.g. Apache) configured to execute CGI/python scripts, only needed when using the interactive visualization web-interface

Setting-up a simulation-based fault injection experiment

Fault injection experiments are configured by means of parameters listed in the DAVOS configuration XML file. Before proceeding with XML configuration, ensure that simulation of HDL design under test (DUT) runs without errors in ModelSim/Questa, and create a simulation checkpoint file by issuing a command in the Modelsim console: `"checkpoint demo.sim"`.

Create a copy of configuration XML file for your SBFI experiment (see examples in the `/testconfig` folder), and adjust the parameters according to the Table 1.

TABLE 1. DESCRIPTION OF SBFI PARAMETERS OF THE DAVOS XML CONFIGURATION FILE

XML tag: <ModelConfig /> (Configuration of HDL model)	
Multiple ModelConfig tags can be included into the XML file to test several configurations of HDL model (different HDL or workload parameters) in the same SBFI experiment (with the same faultload configuration).	
work_dir	The base folder to load and store the files related to SBFI experiment. Usually it is a folder containing a Modelsim project file.
checkpoint	A path to the simulation checkpoint file (relative to the work_dir)
design_type	Determines the type of targeted design nodes (and applicable fault models): RTL – faults are injected into signals and variables NETLIST – faults are injected into Verilog or VHDL/VITAL macrocells
label	Any label of choice to be attributed to different files and results of this particular SBFI run
clk_period	The duration of DUT clock cycle, used only when specifying the faultload timing parameters in terms of clock cycles (faultload time_mode is set to 'ClockCycle'). Otherwise this parameter can be skipped (assigned any value or deleted from XML file).
simulation_time	The duration of simulated time interval in each fault injection run (in nanoseconds), starting from the checkpoint. Usually, simulation_time equals the duration of the workload executed by the DUT. At the end of this time, the values of selected DUT nodes (signals, registers, memory arrays) will be used to analyse the effect of injected fault.
Example: An RTL model located in <code>"/home2/tuil/selene_sim"</code> with a ModelSim checkpoint <code>demo.sim</code> , labelled in SBFI experiment as <code>SeleneMC</code> , with a workload duration of 20000ns. <pre><ModelConfig work_dir = "/home2/tuil/selene_sim" checkpoint = "demo.sim" design_type = "RTL" label = "SeleneMC" clk_period = "10.0" simulation_time = "20000" /></pre>	

XML tag: <InjectionScope /> (Fault Injection scope) Design unit targeted by fault injection. Add as many fault injection scope tags as needed, all of them will be included into the list of valid fault targets	
unit_path	The hierarchical name of the targeted DUT component. DAVOS will automatically extract the list of valid fault injection points within the selected unit path.
node_filter	Filter to select only those injection points (in the selected unit) that match certain name pattern. Leave empty if all nodes in unit_path should be considered for fault injection.
scope_filter	Signal location filter: “-internal” for internal signals, “-input” to inject inly in input ports, “-output” to inject in output ports.
Example: inject faults into the registers (node name starting with ‘r.’) of the integer pipeline unit of the CPU core-0. <pre> <InjectionScope unit_path = "/testbench/cpu/cpu/core0/gpp0/noelv0/cpuloop(0)/core/u0/iu0" node_filter = "r.*" scope_filter = "-internal" /> </pre>	
XML tag: <ObservationScope /> (Design scope to be monitored during fault injection experiment) DUT nodes contained in each Observation Scope are monitored and logged during simulation to analyse the effects of injected faults (failure modes). Include as many observation scopes as needed to detect latent errors (internal nodes) and failures (outputs).	
unit_path	The hierarchical name of the monitored DUT component. DAVOS will recursively extract the list of all nodes (signals/registers/wires at RTL or logic primitives at netlist level) available in the selected DUT component. The value changes of every node will be logged for the subsequent analysis of fault effects.
node_filter	Name filter to monitor only those DUT nodes that match certain name pattern
sampling_options	When set to ‘-notrigger’, this signal doesn’t trigger sampling of trace vectors into the log file. Used to prevent saturation (excessive size) of SBFI log. In this case it is supposed that there is another signal that triggers ModelSim to log the values of all monitored nodes. When left empty – every value change of any observed node will trigger the simulator to log the state of all monitored nodes in the form of array with a time-stamp (see ModelSim List tool).
scope_filter	Filter to monitor only "-internal", "-input", or "-output" signals. Can be omitted (empty) to include all types of signals
group	Each monitored node should be linked to one of two groups: "INTERNALS" – nodes that define the state of targeted design unit, their values are compared against golden run to detect latent errors "OUTPUTS" – nodes that define/contain the outputs of targeted design unit, their values are compared against golden run to detect failures (either signalled or silent)
label_prefix	A shortcut to replace the long hierarchical name of the monitored DUT nodes in the logfile. Can be omitted (empty)
domain	A virtual domain, to which monitored signals are attributed. Typically used to define several groups of monitored nodes (one observation scope per group) when analysing fault effects in multicore CPUs, and replicated modules.
Example: monitor registers of the register file of the CPU core-0, more specifically signals internal to the rf0 whose node name starts with ‘r.’. Do not log every value change of these registers, but log it only when some other monitored signal (without <i>-notrigger</i> option) changes its value. <pre> <ObservationScope unit_path = "testbench/cpu/cpu/core0/gpp0/noelv0/cpuloop(0)/core/u0/dffrf/rf0" node_filter = "r.*" sampling_options = "-notrigger" scope_filter = "-internal" </pre>	

```

group = "INTERNALS"
label_prefix = "Core0_RegFile_"
domain = "Core0"
/>

```

XML tag: <ObservationItem /> (Individual DUT node to be monitored during SBFI experiment)

This tag defines individual DUT nodes to be monitored and logged for analysis, with a support for array slicing. Add as many ObservationItems as needed to analyse fault effects.

path	The hierarchical name of monitored DUT node (should be individual signal: scalar or array).
array_range	The range of monitored array, when applicable, otherwise empty
label	A shortcut to replace the long hierarchical name of the monitored DUT node in the logfile. Can be omitted (empty)
sampling_options	Same as in the case of ObservationScope (see above)
group	Same as in the case of ObservationScope (see above)
domain	Same as in the case of ObservationScope (see above)

Example: monitor first 128 words (range 0 to 127) of the memory page 1. Log the content of these 128 words on every value change (sampling options empty). These values are attributed to the outputs (processing results) of the Core0, and therefore will be used to detect failures (data corruption).

```

<ObservationItem
  path= "/testbench/cpu/cpu/core0/mem0/mig_gen/sim_ram_gen/axi_mem/rb/p/logpages(1).data"
  array_range = "0:127"
  label = "Core0.Res"
  sampling_options = ""
  group = "OUTPUTS"
  domain = "Core0"
/>

```

XML tag: <faultload /> (Faultload parameters)

This tag configures parameters of different fault models considered during SBFI experiment. Add as many tags as needed to cover the relevant fault space.

model	Name of the fault model from the associated fault dictionary. Typical fault models from the predefined fault dictionary are: BitFlip, StuckAt, pulse, indetermination. Note: the fault dictionary used in SBFI experiment is linked to the XML configuration file by means of <i>fault_dictionary</i> attribute of the <SBFI /> XML tag.
target_logic	The type of design nodes, to which the fault model is applied. Relation between fault models and applicable logic is defined by the fault dictionary. Typically, at RTL target logic includes "SIGNAL", and at the netlist level target logic includes macrocells such as "X_FF, X_SFF, X_FDD" (flip-flops), "LUT2, LUT3, LUT4, LUT5, LUT6" (look-up tables), "X_RAMB18E1" (block RAM), etc.
forced_value	A value to be forced by such fault models as stuck-at-1 and stuck-at-0
multiplicity	The number of faults injected in each individual fault simulation run.
time_mode	Determines how fault injection time and duration are defined: "Absolute" – nanoseconds, "Relative" – percentage of workload duration (default), "ClockCycle" – number of clock cycles (relative to clk_period from the <ModelConfig /> tag
injection_time_start	The start of fault injection interval
injection_time_end	The end of fault injection interval
experiments_per_target	The number of faults injected per each node from the list of fault targets (selected from the injection scope)
sample_size	The number of faults sampled from the fault space (constrained by the injection scope, injection time interval, target_logic, etc.)
rand_seed	Determines the fault sampling randomization results (change it to generate different faultload with the same constraints, or keep it to reproduce the same faultload)
CCF	[Optional] node name pattern to inject common-cause faults

Example: a faultload comprising a sample of 5000 BitFlip and 5000 StuckAt-0, and 5000 StuckAt-1 faults. Faults are randomly (uniformly) distributed in the time and space (targeted nodes). Both fault models target signals (RTL model). The StuckAt faults are injected at the beginning of simulation checkpoint (interval 0.0 to 0.0), while BitFlips are injected in the first half of workload time (0.0 to 0.5).

```
<faultload
  model = "BitFlip"
  target_logic = "SIGNAL"
  multiplicity = "1"
  time_mode = "Relative"
  injection_time_start = "0.0"
  injection_time_end = "0.5"
  experiments_per_target = "1"
  sample_size = "5000"
  rand_seed = "10"
  CCF = ""
/>

<faultload
  model = "StuckAt"
  forced_value = "0"
  target_logic = "SIGNAL"
  multiplicity = "1"
  time_mode = "Relative"
  injection_time_start = "0.0"
  injection_time_end = "0.0"
  experiments_per_target = "1"
  sample_size = "5000"
  rand_seed = "10"
  CCF = ""
/>

<faultload
  model = "StuckAt"
  forced_value = "1"
  target_logic = "SIGNAL"
  multiplicity = "1"
  time_mode = "Relative"
  injection_time_start = "0.0"
  injection_time_end = "0.0"
  experiments_per_target = "1"
  sample_size = "5000"
  rand_seed = "10"
  CCF = ""
/>
```

Executing the fault injection experiment

DAVOS supports parallel execution of multiple SBFI experiments on desktop PCs, as well as on the clusters running a Sun Grid Engine (SGE). The former case is used for light parallelization, usually up to 16 processes (depends on the amount of CPU cores and RAM). The Grid-based parallelization commonly allows higher parallelization factors. The type of execution platform and the number of parallel processes is configured by the platform and maxproc attributes of the root DAVOS XML tag. For instance, an example of DAVOS tag below configures an SBFI experiment to run on local multicore PC with 4 parallel simulation processes, and in addition it also configures the path to the report folder (DAVOS results will be exported to the *report_dir*).

```
<DAVOS
  report_dir = "/home/user/HTWEB"
  experiment_label = "DEMO"
  platform = "Multicore"
  maxproc = "4" >
```

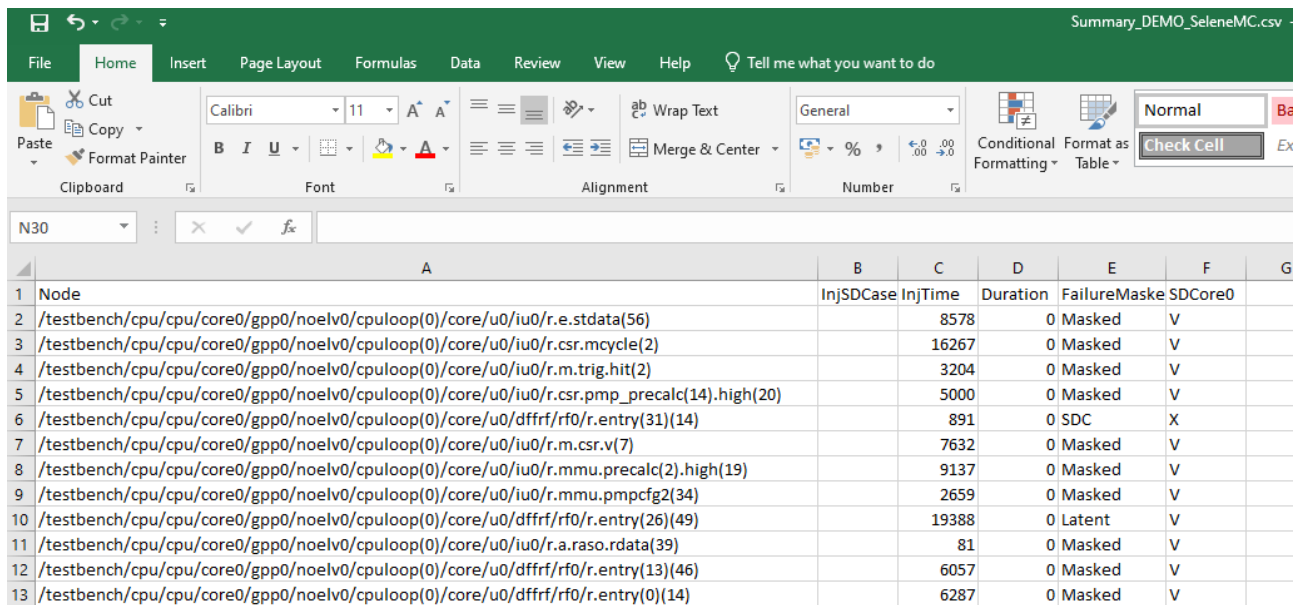
After configuring the parameters in XML file, an SBFI experiment can be started by issuing a following command in the Linux terminal (assuming that XML file is located in DAVOS/testconfig/demo.xml):

```
home/user/> cd DAVOS
home/user/DAVOS/> python SBFI_Tool.py testconfig/demo.xml
```

Exploring and visualizing SBFI results

Once DAVOS reports that SBFI experiment is completed, the user navigates to the results folder, previously configured in the **report_dir** attribute of the root **<DAVOS>** XML tag. These results include: (a) a CSV file with the description of each injection run and its results, as depicted in the Fig.1, and (b) an SQLite dataset with an attached querying and visualization web-interface, as depicted in Fig. 2.

The name of the resulting CSV file starts with a prefix 'Summary_', and includes an *experiment_label*, and a *ModelConfig* label. For each SBFI run this csv table lists a targeted model Node, an injection point (injection case) within the Node in the case of netlist macrocells, an injection time (InjTime), a fault duration (in the case of Pulses, delays, and other transient faults), and a Failure Mode: masked, latent, SDC (silent data corruption), signalled failure, etc.



	A	B	C	D	E	F	G
	Node	InjSDCase	InjTime	Duration	FailureMaske	SDCore0	
2	/testbench/cpu/cpu/core0/gpp0/noelv0/cpuloop(0)/core/u0/iu0/r.e.stdata(56)		8578	0	Masked	V	
3	/testbench/cpu/cpu/core0/gpp0/noelv0/cpuloop(0)/core/u0/iu0/r.csr.mcycle(2)		16267	0	Masked	V	
4	/testbench/cpu/cpu/core0/gpp0/noelv0/cpuloop(0)/core/u0/iu0/r.m.trig.hit(2)		3204	0	Masked	V	
5	/testbench/cpu/cpu/core0/gpp0/noelv0/cpuloop(0)/core/u0/iu0/r.csr.pmp_precalc(14).high(20)		5000	0	Masked	V	
6	/testbench/cpu/cpu/core0/gpp0/noelv0/cpuloop(0)/core/u0/dffrf/rf0/r.entry(31)(14)		891	0	SDC	X	
7	/testbench/cpu/cpu/core0/gpp0/noelv0/cpuloop(0)/core/u0/iu0/r.m.csr.v(7)		7632	0	Masked	V	
8	/testbench/cpu/cpu/core0/gpp0/noelv0/cpuloop(0)/core/u0/iu0/r.mmu.precalc(2).high(19)		9137	0	Masked	V	
9	/testbench/cpu/cpu/core0/gpp0/noelv0/cpuloop(0)/core/u0/iu0/r.mmu.pmpcfg2(34)		2659	0	Masked	V	
10	/testbench/cpu/cpu/core0/gpp0/noelv0/cpuloop(0)/core/u0/dffrf/rf0/r.entry(31)(49)		19388	0	Latent	V	
11	/testbench/cpu/cpu/core0/gpp0/noelv0/cpuloop(0)/core/u0/iu0/r.a.raso.rdata(39)		81	0	Masked	V	
12	/testbench/cpu/cpu/core0/gpp0/noelv0/cpuloop(0)/core/u0/dffrf/rf0/r.entry(13)(46)		6057	0	Masked	V	
13	/testbench/cpu/cpu/core0/gpp0/noelv0/cpuloop(0)/core/u0/dffrf/rf0/r.entry(0)(14)		6287	0	Masked	V	

Fig. 1 – Example of SCV table with SBFI results, exported by DAVOS

To use the querying and visualization interface, the resulting **report_dir** folder should be made accessible from the web-server, and the latter should be configured to run the CGI/python scripts. Once these prerequisites are satisfied, the interactive report can be visualized in the web browser by navigating to the corresponding URI: **web_server_root/report_dir/query.html**. This interface provides several widgets to query SBFI results attending to different filters (fault model, forced value, instance type, HDL model, etc), and visualize these results in interactive piecharts and barcharts. An interactive diagram below the chart explains the contribution of each component in the design tree into the resulting failure percentages. The Table below lists the details and results of each SBFI run (similar to the csv Table). A hyperlink in the first cell of each row navigates to the visualization of detailed SBFI trace for this SBFI run, which highlights all mismatches with the fault-free simulation.

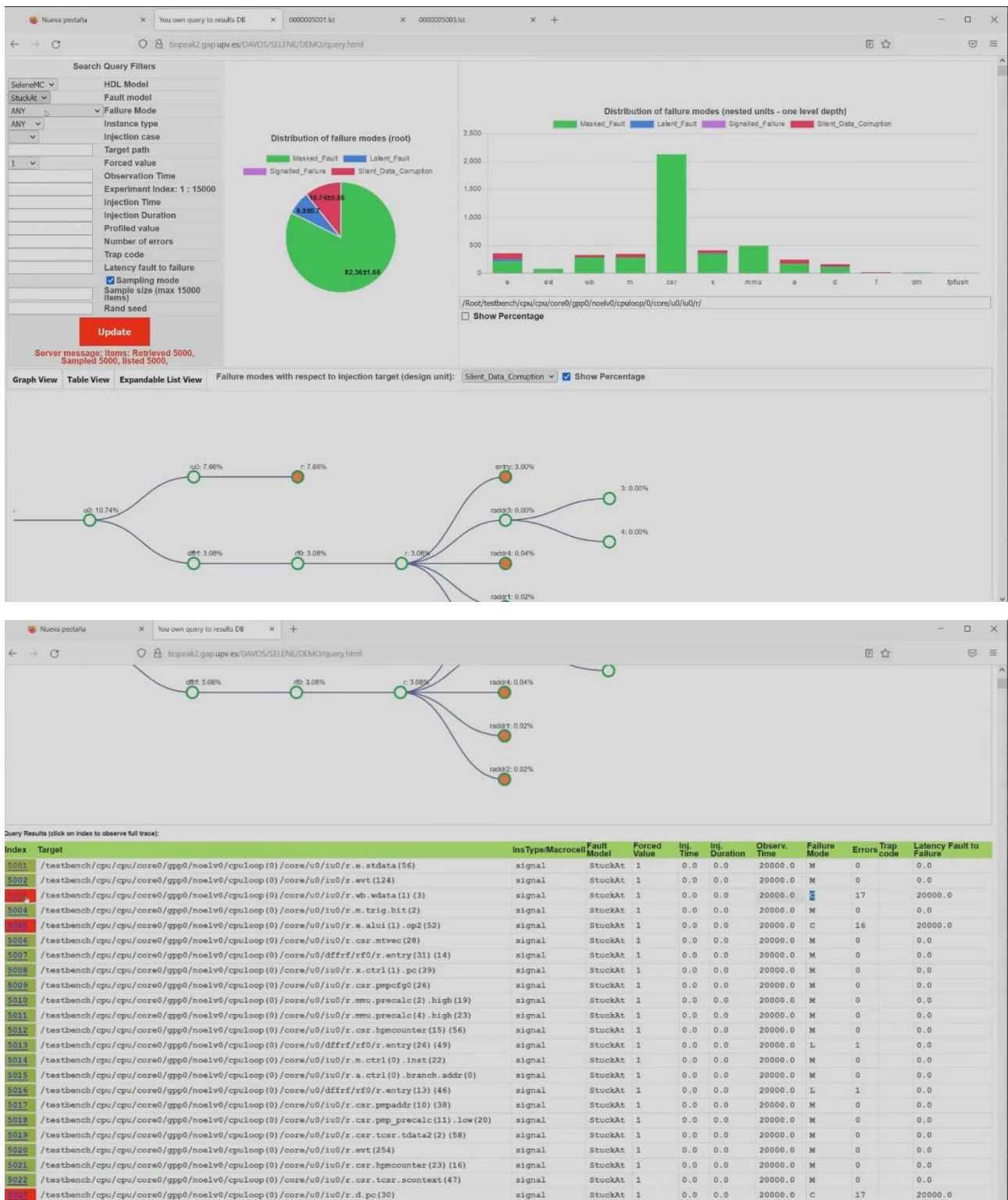


Fig. 2 – Querying and visualization interface for the resulting SBFI datasets

