VC ZO1X Functional Safety - Methodology to accelerate Diagnostic Coverage(DC) by leveraging STL libraries from CPU cores

Naveen Srivastava, Amresh Kumar Lenka, Varun Kumar C, Subramanian R, Sekhar Dangudubiyyam
Samsung Semiconductor India R&D
Gopi Krishna Nagalla
Synopsys

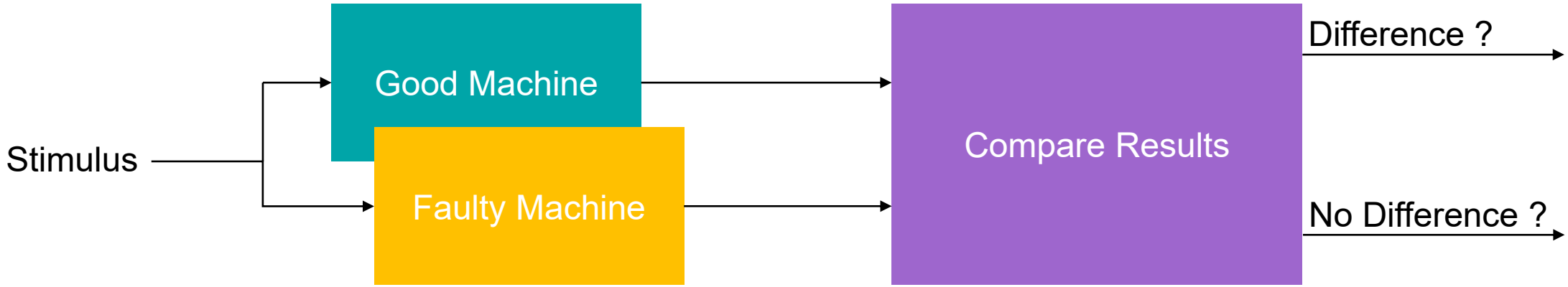
Outline

- Introduction
  - What is fault simulation ?
  - Need for fault simulation in the context of functional safety.
- VC Z01X with STL methodology for functional safety
  - Methodology
  - Scope of STL in functional safety
  - Why STL is required ?
  - Overview of VC-Z01X in functional safety
- STL Capabilities
  - STL Capabilities example
- Proof of concept
  - Execution Flow
  - Results
- FMEDA*
  - FMEDA for permanent faults
- References

Introduction

What is fault simulation ?

- Fault simulation is an execution of test stimulus on a good machine and a faulty machine (identical copy of good machine but with systematically injected faults) and check whether the test stimulus resulted in a difference at the required observation points.



- Detecting a difference is expected as it indicates the visibility of faults in the circuit.
- Failure to produce the difference indicates the need for change in the internal logic so that missed faults are now detectable.

Need for fault simulation

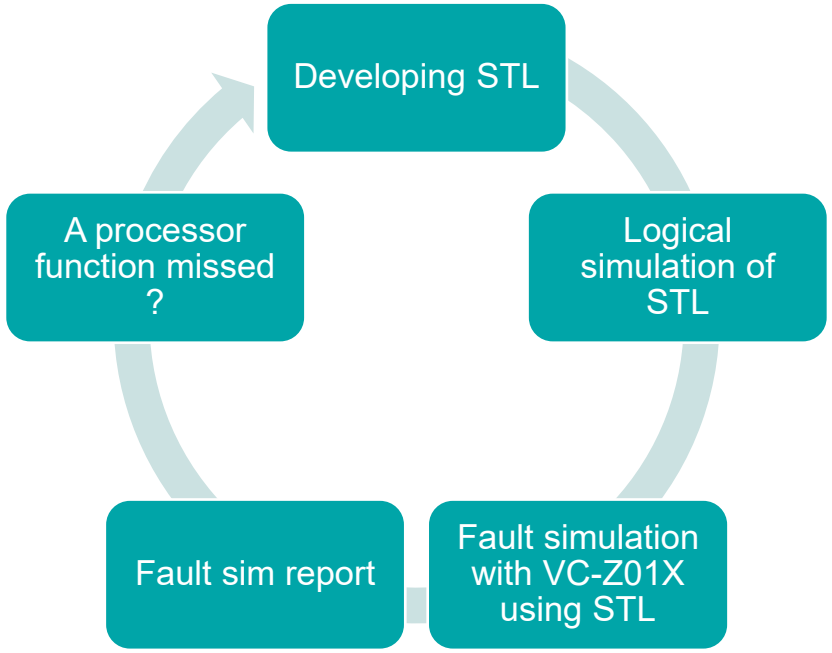
- Fault simulation can be used in two contexts:
    1. To catch bad parts or manufacturing defects.
    2. Functional safety verification.

In this presentation we will be discussing the functional safety verification context of fault simulation.

- Fault simulation is required in safety critical applications like space stations and automotive vehicles.
- Space has harsh environment with ionizing radiations. These high energy particles can cause permanent stuck at faults in the circuits, which are getting used for life support system of astronauts.
- The chips in automotive vehicles are continuously subjected to various attacks from supply, frequency, laser and temperature. This causes faults being generated resulting in unexpected behavior of vehicles.
- Simulating these real life fault scenarios is paramount.
- Functional safety verification gives the overview about a design's capability to recognize the faults.

VC Z01X with STL methodology for functional safety

Methodology

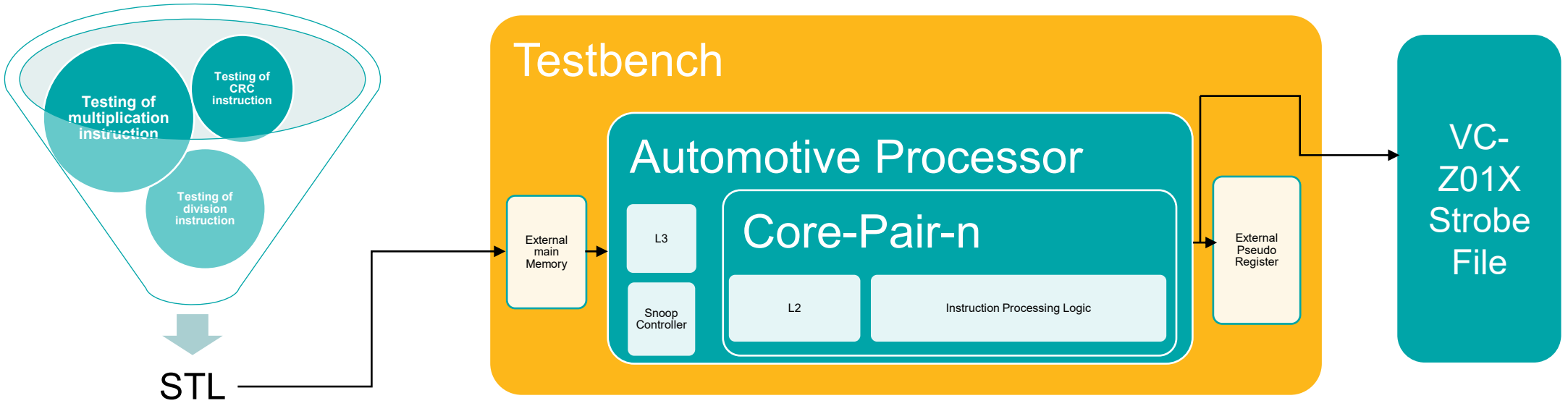


- In this methodology there is an integration of fault injection and simulation capabilities of the VC-Z01X tool with software test libraries (STL).
- STL libraries enable us to simulate the full range of logical functionalities in the core.
- Since the STL exercises critical processor pathways, the injected faults are more meaningful by providing valuable faulty machines.

Scope of STL in functional safety

- STL stands for software test library, Which contain assembly + C written test cases to target/test individual functions and registers of the processing core.
- In the scope of functional Safety, STL is considered as “Safety element out of context”, for ISO26262 standard[2][3]. And a “Complaint item”, for IEC 61508 standard.
- But why our STL is called a “Safety element out of context” or a “Complaint item” ?

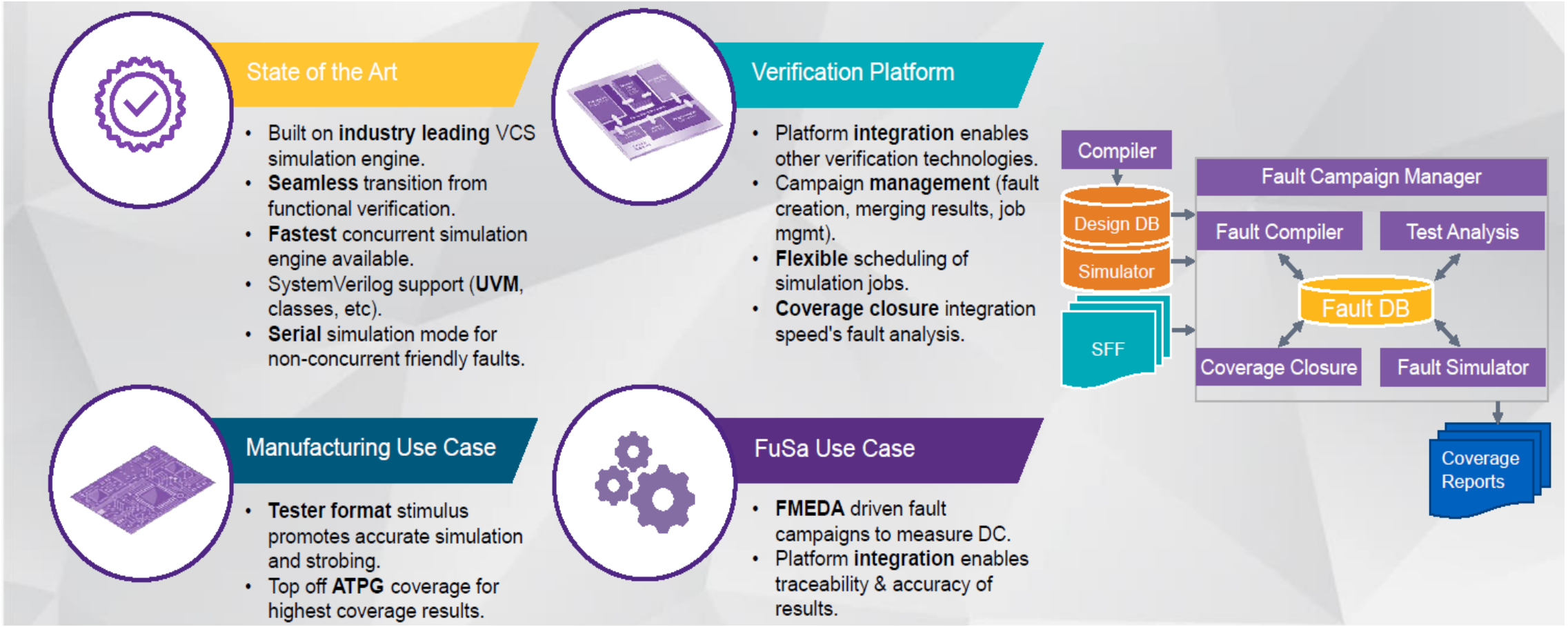
Why STL is required ?



- In a typical testing procedures, there is adherence to standard processor tests employing the Dhrystone benchmark or executing cache hit/miss scenarios.
- However, these scenario-based tests may not comprehensively cover all logical functionalities, leaving potential safety gaps.
- Transitioning to Software Test Libraries (STL), can liberate us from scenario-centric testing and address each function of the design individually.
- As STL reports the status of the test outside of the DUT, to a pseudo register, strobing becomes really easy and makes the setup reusable for both RTL and GLS simulation.

Overview of VC-Z01X in functional safety

- VC-Z01X overview.



STL Capabilities

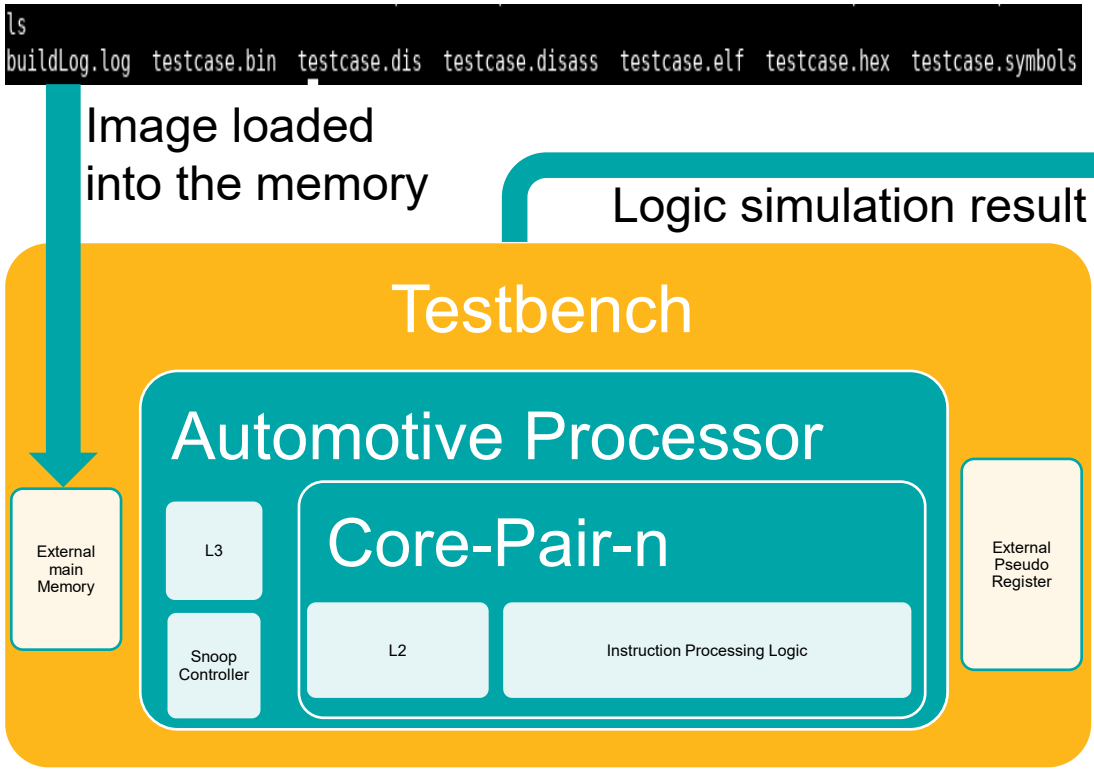
STL Capabilities example

Feature List	Description
Feature 1	Testing of logical instructions AND, EOR and ORR
Feature 2	Testing of multiplication instructions
Feature 3	Testing of branch and addition instructions
Feature 4	Testing of GPR registers
Feature 5	Testing of CRC instructions
Feature 6	Testing of load/store instruction
Feature 7	Testing of bit operation instructions
Feature 8	Testing of subtraction instructions
Feature 9	Testing of shift instructions
Feature 10	Testing of conditional select instructions
Feature 11	Testing of division instructions
Feature 12	Testing operand arithmetic with carry instructions

Proof of Concept

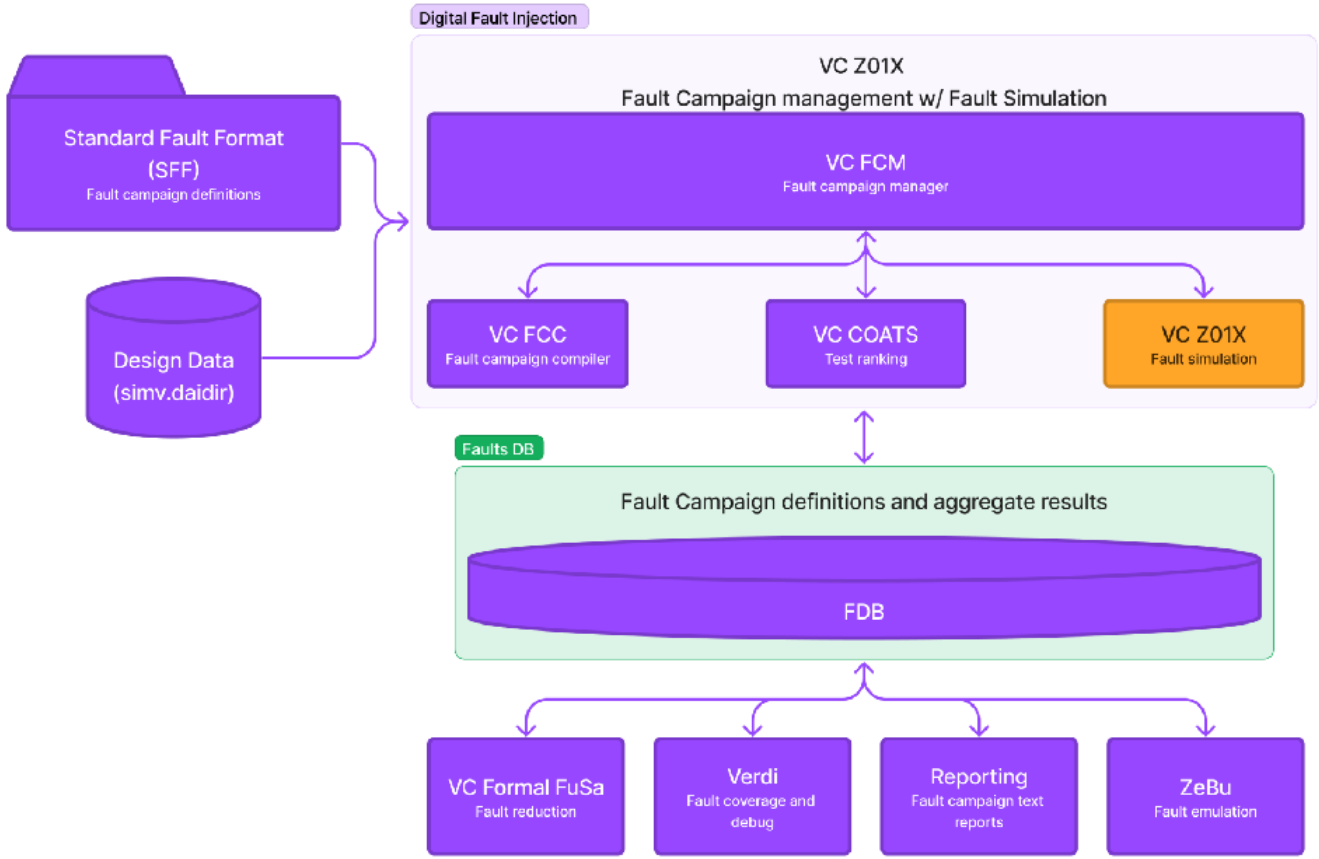
Execution Flow

- The test case can be generated for required instruction, like for testing multiplication, division and addition instructions.
- After the build, compiled hex and elf files are loaded into external main memory.
- Logical simulation is run, to check the credibility of test case.




```
Loading /user/sim/tests/testcase/testcase.elf into memory
  0x00000000:  0xaa1f00e0 0xaa1f0de1 0xaa1f0de2 0xaa1f0fe3
  0x00000010:  0xaa1f00e4 0xaa1f0de5 0xaa1f0de6 0xaa1f0fe7
  0x00000020:  0xaa1f00e8 0xaa1f0de9 0xaa1f0dea 0xaa1f0feb
  0x00000030:  0xaa1f00ec 0xaa1f0ded 0xaa1f0dee 0xaa1f0fef
Creating memory 'execution_tb memory', with a default memory value of 0x_00000000_00000000:
Loading memory at time  0.00ms
*Verdi* FSDB: For performance reasons, the Memory Size Limit has been increased to 1024M.
(    6232ns ) PE 0:
(    6251ns ) PE 0:
(    6653ns ) PE 0: ** TEST PASSED **
(    6672ns ) PE 0:
PE 0 terminated the test at 6691ns
```

- Once the logical simulation confirms “TEST PASSED” status, proceeded for fault simulation using VC-Z01X. Below image gives the brief idea on the fault simulation flow of VC-Z01X.



- Preparation of .SFF file for status definition and fault injection location.

```
FaultGenerate STL {
    NA [0,1] { PORT [INPUT,OUTPUT] "automotive_cpu_wrapper_execution_tb.u_automotive_cpu_wrapper.u_core_pair.u_core_0.u_cpu.u_instruction_execute.**"}
    NA [0,1] { PORT [INPUT,OUTPUT] "automotive_cpu_wrapper_execution_tb.u_automotive_cpu_wrapper.u_core_pair.u_core_0.u_cpu.u_instruction_fetch.**"}
    NA [0,1] { PORT [INPUT,OUTPUT] "automotive_cpu_wrapper_execution_tb.u_automotive_cpu_wrapper.u_core_pair.u_core_0.u_cpu.u_instruction_decode.**"}
    NA [0,1] { PORT [INPUT,OUTPUT] "automotive_cpu_wrapper_execution_tb.u_automotive_cpu_wrapper.u_core_pair.u_core_1.u_cpu.u_instruction_execute.**"}
    NA [0,1] { PORT [INPUT,OUTPUT] "automotive_cpu_wrapper_execution_tb.u_automotive_cpu_wrapper.u_core_pair.u_core_1.u_cpu.u_instruction_fetch.**"}
    NA [0,1] { PORT [INPUT,OUTPUT] "automotive_cpu_wrapper_execution_tb.u_automotive_cpu_wrapper.u_core_pair.u_core_1.u_cpu.u_instruction_decode.**"}
}
```

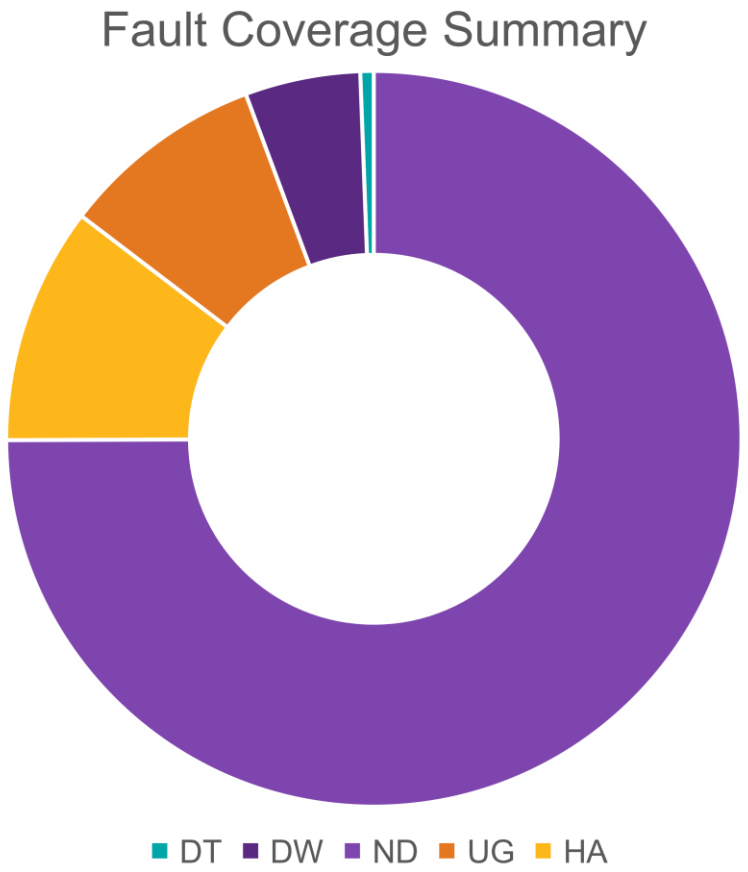
- Tcl script preparation

```
set_submit_cmd -grid_type LSF -cmd {bsub -R "select[(osversion==RHEL7.9||osversion==RHEL6.10)] rusage[mem=300000]"} -task_type default
create_campaign -args "-full64 -daidir simv.daidir -sff ./stl.sff -campaign automotive_cpu_ss -sample percent:20 -dut automotive_cpu_wrapper_execution_tb.u_automotive_cpu_wrapper -overwrite"
set_campaign -campaign automotive_cpu_ss
set_config -global_max_jobs 1000
report -campaign automotive_cpu_ss  -report before_fault_sim.rpt -showfaultid -overwrite
create_testcases -name {"a78ae_stl_core_diagnose_p001_n001_lock"} -exec "./simv" -args "+elfname=./tests/testcase/testcase.elf" -fsim_args "-fsim=fault+dictionary"
fsim
report -campaign automotive_cpu_ss -report diagnostic_coverage.rpt -showfaultid -overwrite
```

Results

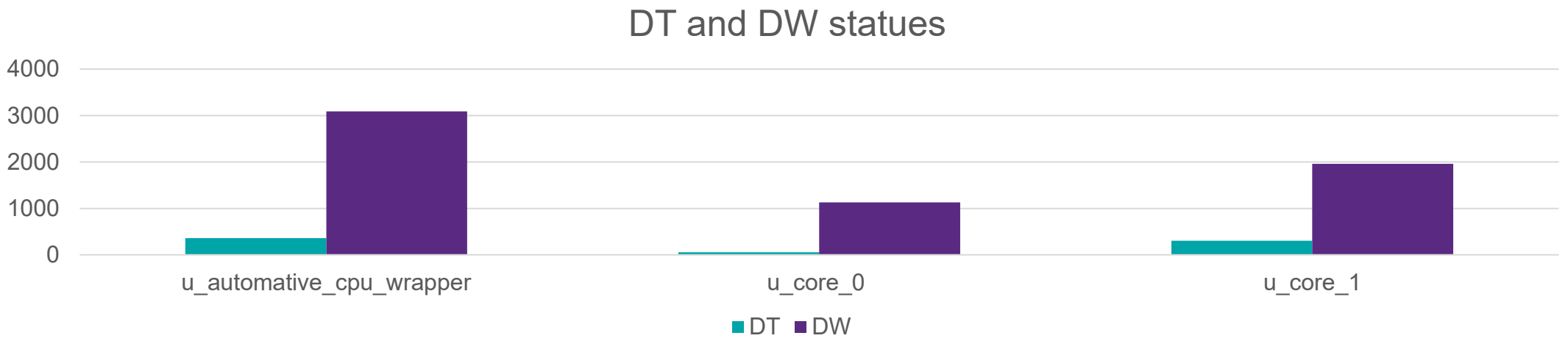
- After the successful execution of faultsim, below diagnostic coverage report is generated. This particular report is with testing just load and store instruction.

```
#---------------------------------------------------------------------
# Number of Faults:                          60916 100.00%
#
# Untestable Faults:                          5468   8.98%  100.00%
#   Untestable Unused                  UU     5264   8.64%   96.27%
#   Untestable Tied                    UT      204   0.33%    3.73%
#
# Testable Faults:                           55448  91.02%  100.00%
#   Hyperactive                        HA     6347  10.42%   11.45%
#   Not Detected by STL                ND    45650  74.94%   82.33%
#   Detected by STL                    DT      360   0.59%    0.65%
#   Detected by Watchdog               DW     3091   5.07%    5.57%
#
# Status Groups -------------------------------------------------------
#   Hyper                              HG     6347  10.42%
#   Untestable                         UG     5468   8.98%
#
# Coverage ------------------------------------------------------------
#   Diagnostic Coverage                          7.03%
#---------------------------------------------------------------------
```



- VC-Z01X also facilitates the fault simulation status per hierarchy, hence the complete picture of DC status for all hierarchies from top to bottom is obtained.

```
# Statuses: ND, HA, UU, UT, DT, DW
Total              ND              HA              UU              UT              DT              DW Scope
----- --------------- --------------- --------------- --------------- --------------- --------------- -----
60916 45650  (74.94%) 6347  (10.42%) 5264   (8.64%) 204   (0.33%) 360   (0.59%) 3091   (5.07%) automative_cpu_wrapper_execution_tb
60916 45650  (74.94%) 6347  (10.42%) 5264   (8.64%) 204   (0.33%) 360   (0.59%) 3091   (5.07%) -u_automative_cpu_wrapper
30458 22917  (75.24%) 3620  (11.89%) 2632   (8.64%) 102   (0.33%)  56   (0.18%) 1131   (3.71%) --u_core_pair
30458 22917  (75.24%) 3620  (11.89%) 2632   (8.64%) 102   (0.33%)  56   (0.18%) 1131   (3.71%) ---u_core_0
30458 22917  (75.24%) 3620  (11.89%) 2632   (8.64%) 102   (0.33%)  56   (0.18%) 1131   (3.71%) -----u_cpu
30458 22917  (75.24%) 3620  (11.89%) 2632   (8.64%) 102   (0.33%)  56   (0.18%) 1131   (3.71%) ------u_instruction_execute
30458 22733  (74.64%) 2727   (8.95%) 2632   (8.64%) 102   (0.33%) 304   (1.00%) 1960   (6.44%) ---u_core_1
30458 22733  (74.64%) 2727   (8.95%) 2632   (8.64%) 102   (0.33%) 304   (1.00%) 1960   (6.44%) -----u_cpu
30458 22733  (74.64%) 2727   (8.95%) 2632   (8.64%) 102   (0.33%) 304   (1.00%) 1960   (6.44%) ------u_instruction_execute
```



FMEDA*

FMEDA For Permanent faults

Part	Failure Mode	Technology	Safety Related	FM_TYPE	No of Gates	λp	Sp%	λpd	λp %	DCp %	λpr
u_instruction_ execute	Data corruption	3nm	YES	MISSION	350	350*1FI T=350	8.89%	350-8.89% = 318.885	0%	7.03 %	296.27

References

1. J. Seaton Zycad Corp., Menlo Park, CA, USA, Fault Simulation Basics, In https://ieeexplore.ieee.org/xpl/conhome/874/proceeding, DOI: 10.1109/ASIC.1989.123161
2. Alejandra Ruiz Division Tecnalia, ICT–European Software Institute, Derio, Spain, Alberto Melzi Division Tecnalia, ICT–European Software Institute, Derio, Spain,Tim Kelly Division Tecnalia, ICT–European Software Institute, Derio, Spain, Systematic application of ISO 26262 on a SEooC: Support by applying a systematic reuse approach in https://ieeexplore.ieee.org/xpl/conhome/7076741/proceeding, DOI: 10.7873/DATE.2015.0177
3. https://www.synopsys.com/automotive/what-is-iso-26262.html
4. https://www.synopsys.com/verification/simulation/vc-z01x.html
5. https://www.synopsys.com/verification/resources/whitepapers/functional-safety-fault-simulation-wp.html
6. https://www.synopsys.com/verification/simulation/z01x-functional-safety.html
7. https://www.synopsys.com/content/dam/synopsys/verification/datasheets/vc-z01x-ds.pdf

SAMSUNG snug

THANK YOU

Our
Technology,
Your
Innovation™