# DVCon India 2023

| | |
|---|---|
| **TITLE OF PAPER** | **Fast Convergence Modular Advanced Smart-Hybrid Testbench (FCMAST) to automate and expedite SoC gate level simulations closure** |
| **AUTHOR 1** | **Name: Harshal Kothari**<br>**Organization: Samsung Semiconductors India Research, Bangalore**<br>**Job Title: Staff Engineer**<br>**Email ID: harshal.k1@samsung.com** |
| **AUTHOR 2** | **Name: Eldin Ben Jacob**<br>**Organization: Samsung Semiconductors India Research, Bangalore**<br>**Job Title: Staff Engineer**<br>**Email ID: eldin.jacob@samsung.com** |
| **AUTHOR 3** | **Name: Chandrachud Murali**<br>**Organization: Samsung Semiconductors India Research, Bangalore**<br>**Job Title: Staff Engineer**<br>**Email ID: chandru.m@samsung.com** |
| **AUTHOR 4** | **Name: Sriram Kazhiyur Soundarrajan**<br>**Organization: Samsung Semiconductors India Research, Bangalore**<br>**Job Title: Associate Director**<br>**Email ID: sriram.k.s@samsung.com** |
| **AUTHOR 5** | **Name: Somasunder Kattepura Sreenath**<br>**Organization: Samsung Semiconductors India Research, Bangalore**<br>**Job Title: Director**<br>**Email ID: soma.ks@samsung.com** |

## ABSTRACT

Gate level simulations (GLS) are an integral part of the ASIC verification cycle to verify the impact of synthesis, insertion of power elements and placement & routing by running functional datapaths using dynamic simulations. It is imperative to run timing gate level and power aware simulations with Standard Delay Format (SDF) back-annotation to validate that the hardware designs function as intended in the best and the worst PVT corner delays on the post PnR netlists. This however, comes with a trade-off of huge simulation times, especially when run at a full chip netlist level. With increasing size and complexity of the chip, the total number of tests to be verified and net run time rise exponentially. With time to market and adherence to schedule being critical parameters for the success of a product, the need to strategize optimal selection of tests to be run without compromise on quality while simultaneously obtaining quicker results without doing multiple iterations is of paramount importance. Fast Convergence Modular Advanced Smart-Hybrid Testbench (FCMAST) addresses these vulnerabilities with its robust and flexible architecture. The testbench environment is self-aware and capable of auditing the logs, failures and test plan status, reporting all issues, timing violations and intelligently identifying erroneous environment which will need reruns at later stages. This paper highlights how FCMAST, a holistic approach to expedite gate

level simulation, resulted in achieving 100% pass rate closure well ahead of metal tapeout. Audit and automations results in early capture of environment issues even ahead of a simulation being run. Smart modular hybrid testbench setup can be reused across all flavours of simulations - unit delay GLS, timing GLS, unit delay power aware GLS and timing power aware GLS across projects. Furthermore, with Dynamic Save and Restore (DSNR), Capture and Replay (CNR) and LSF optimizations integrated across simulation flavours, FCMAST which is deployed on AR/VR SoC and HPC 3DIC SoC delivered up to 65% savings on simulation time and 35% less resources without compromising on quality.

**Keywords:** Verfication, gate, netlist, testbench, simulation, performance, automation.

# INTRODUCTION

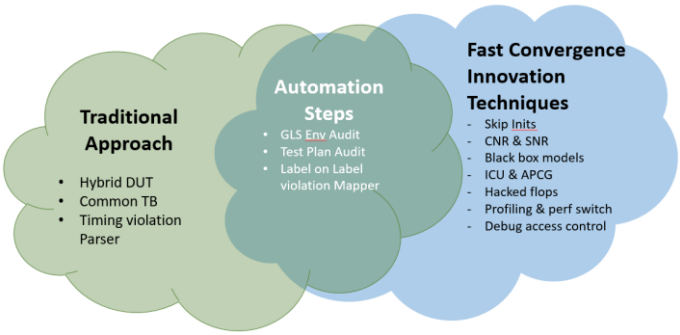


*Figure 1. FCMAST Implementation Strategy*

FCMAST employs a few techniques which are a mix of traditional and innovative and novel processes to expedite and automate the GLS cycle (Figure 1). The traditional techniques are common testbench for different flavours of DUTs resulting in environment variable based version control for RTL/PRENET/POSTNET/PGNET/SDF, on the fly hybrid elaboration and simulation and timing violation parser. GLS audit of elaboration and simulation logs, test plan status audit, label on label timing violation mapper employ a mix of traditional and novel methods. The innovative and novel methods involve improving simulation time by optimizing environment on which the simulation is run and accelerating the tool and compute along with scaling up the methodologies and deploying them across simulation flavours. All these methodologies work in tandem under the FCMAST hood.

# FCMAST Traditional Techniques

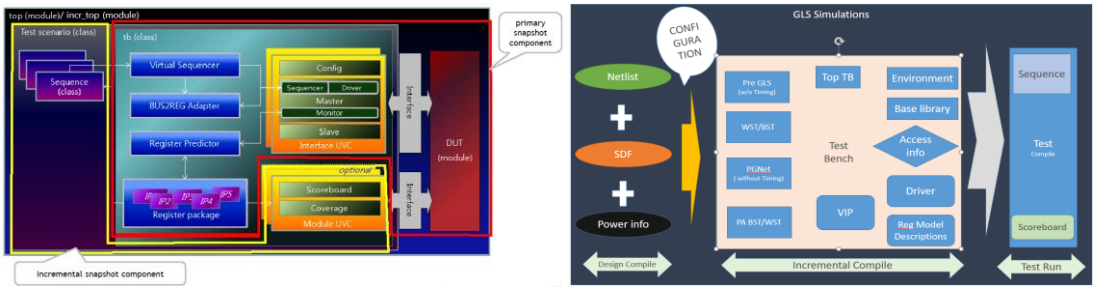


*Figure 2. Traditional FCMAST testbench architecture*

**Common TB:** To develop the common testbench for different flavoured DUTs, the testbench is architected on UVM that provides high standardization with respect to skeleton architecture, file/directory structure, naming rule, Verification IP (VIP) configuration, register model and general access sequences. The environment (Figure 2) is scalable to any IP/Sub-system/SoC that bolsters the reusability aspect or different flavours of DUTs (RTL/PRENET/POSTNET/PGNET). An internal tool utility dumps out this skeleton testbench based on excel input containing top level DUT spec.

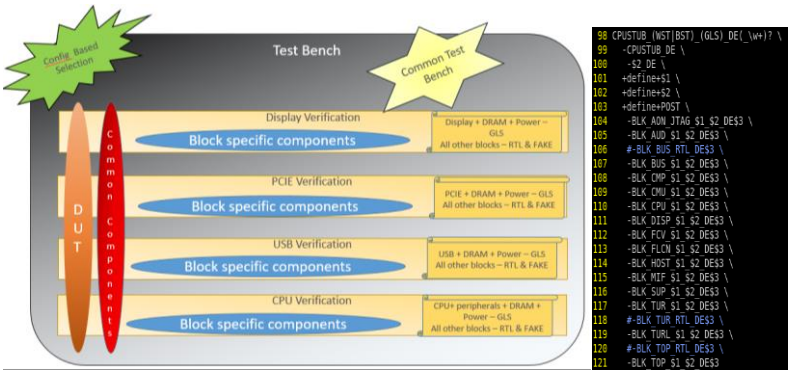


*Figure 3. IP wise on the fly Hybrid Verification Environment*

**Hybrid DUT:** Traditionally, by replacing non-targeted block netlist with black-box or RTL and employing fast boot using forces, time saving of around 6-10% can be achieved by bringing down the gate count. Apart from the simulation time, the hybrid verification method (Figure 3) helps in saving precious engineer's time due to the reduced volume of errors originating from a block/IP which is not the point of interest. PERL macro based configuration offers the flexibility to pick simulator options [1], switches, the model type (RTL/Netlist/Fake) and the DUT version of each block on the fly for elaboration/simulation. The combinations of configuration results in creation of a unique Design Environment (DE) and Verification Environment (VE). The environment also supports the combinational usage of DE/VEs resulting in faster simulation of cross functional datapaths. Using a hybrid DUT for simulation helps save tremendous amount of run time when a functional datapath does not traverse all the blocks. For example, when running a camera stream datapath to DRAM, only the blocks containing CPU, DPHY, Camera Serial Interface, system network fabric and memory controller are needed as netlists. Remaining blocks can either be stubbed out by fake blackbox driving a known output or RTL. Similarly, for 3DIC, if top die is running DMA transfers to local SRAM, bottom die is stubbed out to save half the total gate count. The multi-step incremental elaboration and simulation flow eliminates recompile of DUT due to VE changes.

| | A | B | C | D | E | F | G | H | I | J |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Block | Violation Type | Module Name | Scope | Simulation Time | Unique | Jira Number | Status | Comments | Log path |
| 2 | BLK_AON_JTAG | setup | SDFFRPQRIV_D1_N_S6TR_C54L10 | top.dut.BLK_A | 1427101722 PS | Y | | | | |
| 3 | BLK_AON_JTAG | setup | SDFFNRPQRIV_D1_N_S6TR_C54L10 | top.dut.BLK_A | 2822921287 PS | Y | | | | /user/avob0v3/SIM/EVT1_ML |
| 4 | BLK_AON_JTAG | setup | SDFFRPQRIV_D1_N_S6TR_C54L10 | top.dut.BLK_A | 1384468010 PS | N | | | | |
| 5 | BLK_AON_JTAG | setup | SDFFRPQRIV_D1_N_S6TR_C54L10 | top.dut.BLK_A | 1388434764 PS | Y | | | | |
| 6 | BLK_AON_JTAG | setup | SDFFRPQRIV_D1_N_S6TR_C54L10 | top.dut.BLK_A | 1388434764 PS | N | | | | /user/avob0v3/SIM/EVT1_ML |
| 7 | BLK_AON_JTAG | setup | SDFFRPQRIV_D1_N_S6TR_C54L10 | top.dut.BLK_A | 1384468010 PS | N | | | | |
| 8 | BLK_AON_JTAG | setup | SDFFRPQRIV_D1_N_S6TR_C54L10 | top.dut.BLK_A | 1388434764 PS | N | | | | |
| 9 | BLK_AON_JTAG | setup | SDFFRPQRIV_D1_N_S6TR_C54L10 | top.dut.BLK_A | 1388434764 PS | N | | | | /user/avob0v3/SIM/EVT1_ML |
| 10 | | | | | | | | | No timing violations found | /user/avob0v3/SIM/EVT1_ML |
| 11 | | | | | | | | | No timing violations found | /user/avob0v3/SIM/EVT1_ML |

*Figure 4. Final output of timing violation parser utility*

**Violation Parser:** Timing GLS simulation aids in identifying all hold/setup/recovery/removal violations of the DUT. The violation parser utility comprises of a set of Python scripts with artificial intelligence to map the violation type and reports the timing violation analysis using machine learning with in depth details about the block, violation type, scope, violation time in the run at the end of the simulation automatically (Figure 4). It can also be run on multiple simulations or regression in parallel.

**ENV Audit:** While running simulations with a highly hybrid yet complex environment to get faster turnaround, the probability of an oversight occurring increases. GLS environment configuration requires an in depth understanding of functional path spanning across blocks failing which may result in a fake pass, where a block of concern or importance might be run as non-netlist. SDF files, accurate timing checks off files for asynchronous flops/false path/multicycle path and incorrect file version control also impact the result of simulation and warranting reruns. With the Python based environment audit script (Figure 5), such mistakes which results in vast simulation rerun times can be caught even before starting the simulation, ensuring first time correctness of GLS environment and runs as per design requirement. This project agnostic utility checks for annotation percentage as well as ascertains the genuineness of passing test in the most optimum conditions. This audit utility brings down the number of iterations before stable environment convergence from 4 or 5 to 1 (Table 1).

Elaboration Log Audit

| BLK | Result | Expected | Alloted m | Mode Status | Comments | Expected | Tfile Stat | Commen | Expected | Used Tfil | Tfile Version | Comments |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AON | FAIL | GLS | GLS | PASS | | BLK_AON | PASS | | 3 | 1 | FAIL | Wrong version |
| AUD | FAIL | GLS | RTL | FAIL | Block AUD is not | NA | NA | | NA | NA | NA | |
| BUS | | GLS | GLS | PASS | | BLK_BUS | PASS | | 1 | 1 | PASS | |
| CMU | | GLS | GLS | PASS | | BLK_CMU | PASS | | 1 | 1 | PASS | |
| CMP | FAIL | GLS | RTL | FAIL | Block CMP is not | NA | NA | | NA | NA | NA | |
| CPU | FAIL | GLS | RTL | FAIL | Block CPU is not | NA | NA | | NA | NA | NA | |
| DISP | FAIL | GLS | RTL | FAIL | Block DISP is not | NA | NA | | NA | NA | NA | |
| FCV | FAIL | GLS | RTL | FAIL | Block FCV is not | NA | NA | Run Time | 11:30:00 | | | |
| FLCN | FAIL | GLS | RTL | FAIL | Block FLCN is not | NA | NA | SDF Statistics | | | | |
| HOST | FAIL | GLS | RTL | FAIL | Block HOST is not | NA | NA | No. of Pathd | No. of Disabled Pathdelays = 0 | | Annotated = 99.77% ( | |
| MIF | | GLS | GLS | PASS | | BLK_MIF | PASS | No. of Tcheck | No. of Disabled Tchecks = 87367 | | Annotated = 99.85% | |
| SUP | FAIL | GLS | RTL | FAIL | Block SUP is not | NA | NA | | Total(T) | Disabled | Annotate | Percentage(A/(T-D)) |
| TUR | FAIL | GLS | RTL | FAIL | Block TUR is not | NA | NA | Path Delays | 6614264 | 0 | 6599337 | 99.77 |
| TURL | FAIL | GLS | RTL | FAIL | Block TURL is not | NA | NA | $setup | 40 | 0 | 0 | 0.00 |
| TOP | | GLS | GLS | PASS | | AVOGAD | PASS | $hold | 20 | 0 | 0 | 0.00 |
| | | | | | | | | $period | 62886 | 0 | 62884 | 100.00 |
| | | | | | | | | $width | 11243399 | 55144 | 1121764 | 99.82 |
| | | | | | | | | $recrem | 1163365 | 6881 | 1160967 | 99.85 |
| | | | | | | | | $setuphold | 8346583 | 25342 | 8334034 | 99.88 |
| | | | | | | | | $nochange | 4 | 0 | 0 | 0.00 |

*Figure 5. GLS environment audit script output*

**Plan Audit:** In order to save time on GLS runs, it is prudent to have a historical data of the test progress and reference to narrow down on the failures. This utility audits the verification plan status based on the inputs from the engineers and checks for errors in the logs, existence of test database for a particular release version/flavour as required. When SoC is nearing tapeout, it might get difficult to procure a regression manager license as thousands of RTL and netlist simulations would be running in parallel. In such scenario, it can mimic that functionality without needing an additional costly license. The utility is capable of saving at least 1.5 – 2 man months with audit ensuring correctness of the databases across different flavour of DUT over different releases (Figure 7).

| | | | | | RTL ML4_DEV09 | | | POST_GLS ML4_DEV09 | | PRE_GLS ML4_DEV09 | | POST_GLS ML4_DEV10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| S.No | Block | Email | GLS Owner | TestPlan ID | Test Case | Log Path | Test Status | Log Path | Test Status | Log Path | Test Status | Log Path | Test Status |
| 1 | A | chandru@sam | Chandru | N001 | test-cpusub_wst_gls | /user/avob0v3 | PASS | /user/avob0v3/SIM/EV | PASS | /user/avob0v3/SIM/EV | PASS | /user/avob0v3/SIM/EV | WAIVED |
| 2 | B | harshal.k1@sam | Harshal | H025 | test_case1, test_case2 | /user/avob0v3 | WAIVED | log_path_gls1, log_path | FAIL | /user/avob0v3/SIM/EV | PASS | /user/avob0v3/SIM/EV | FAIL |
| 3 | C | harshal.k1@sam | Harshal | H154 | test_case1, test_case2 | /user/avob0v3 | PASS | log_path_gls1, log_path | PASS | /user/avob0v3/SIM/EV | PASS | /user/avob0v3/SIM/EV | WAIVED |

*Figure 6. Sample input test plan status xls*

| S.No | Block | IP owner | GLS Owner | TestPlan ID | Log Path | Test Status | Audit Result | Comments |
|---|---|---|---|---|---|---|---|---|
| | 1 A | chandru@san | Chandru | N001 | | PASS | FAIL | FAIL Number of log paths and test cases don't match |
| | 2 B | Harshal.k1@s | Harshal | H025 | | WAIVED | | |
| | | Harshal.k1@s | | | /user/avob | | | PASS |
| | 3 C | | Harshal | H154 | /user/dipc | PASS | FAIL | FAIL Log path does not exist |

RTL ML4_DEV09 | POST_GLS ML4_DEV09 | PRE_GLS ML4_DEV09 | POST_GLS ML4_DEV10

*Figure 7. Audit output based on DUT type and release label*

**Violation Mapper**: Simulation free of timing violations is key to successful silicon results. Debugging the violation on GLS is very time consuming and has a direct impact on valuable engineer's time. This calls for an efficient and methodical approach to map the violations over different DUT versions, grouping similar violations

across blocks to avoid duplicate efforts from engineers working on the SoC and to dynamically track the resolution of the violation in subsequent release and in the JIRA or bug tracker system. The Violation Mapper script (Figure 8) with artificial intelligence built-in, reduced not just the turn-around time for the resolution of the issues but also improved the efficiency of the team by avoiding duplicate/rework resulting in a saving of at least 10 – 15 months for bigger SoCs with efficient tracking of all the violations.

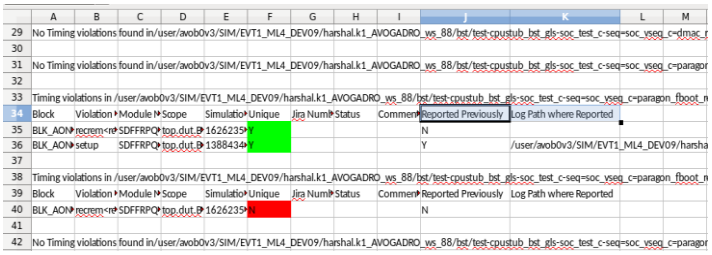


*Figure 8. Utility output*

With a mix of traditional and audit automation techniques combined with erstwhile pure traditional techniques, around 4x-5x performance improvement is observed for a GLS suite of around 120 tests (Table 1).

| Initial Metrics | | | | | | Improved Metrics via Traditional+Audit techniques | | | |
|---|---|---|---|---|---|---|---|---|---|
| Block | Total Tests | No. of netlist labels | Avg Iterations for first pass | Avg run time (hrs) | Total runtime (hrs) | Avg Iterations for first pass | Avg run time (hrs) | Improved Total runtime (hrs) | Net Improvement (X times) |
| A | 19 | 8 | 4 | 85 | 51680 | 1 | 78 | 11856 | 4.4 |
| B | 12 | 8 | 4 | 76 | 29184 | 1 | 68 | 6528 | 4.5 |
| C | 4 | 8 | 2 | 94 | 6016 | 1 | 84 | 2688 | 2.2 |
| D | 3 | 7 | 3 | 63 | 3969 | 1 | 57 | 1197 | 3.3 |
| E | 2 | 8 | 2 | 67 | 2144 | 1 | 55 | 880 | 2.4 |
| F | 14 | 8 | 4 | 78 | 34944 | 1 | 69 | 7728 | 4.5 |
| G | 8 | 8 | 3 | 163 | 31296 | 1 | 145 | 9280 | 3.4 |
| H | 22 | 8 | 4 | 102 | 71808 | 1 | 97 | 17072 | 4.2 |
| I | 13 | 6 | 3 | 73 | 17082 | 1 | 62 | 4836 | 3.5 |
| J | 17 | 8 | 4 | 68 | 36992 | 1 | 60 | 8160 | 4.5 |
| K | 6 | 8 | 5 | 152 | 36480 | 1 | 129 | 6192 | 5.9 |
| L | 10 | 8 | 3 | 79 | 18960 | 1 | 74 | 5920 | 3.2 |

*Table 1. Initial v/s improved GLS metrics*

# FCMAST Fast Convergence Innovation Techniques

- **Custom Simulation Strategies (CSS)**

The simulations for the current DUT takes approximately 60 hours to complete a basic SoC boot with full chip netlist which can go up to 200+ hours for timing PG net simulations. The internal methods via CSS to curtail these simulation times include the design and sequence based changes that regulates the load on the simulator. FCMAST employs Skip Initialization Analysis to retain the intent of simulation by eliminating irrelevant clock/firewall/controller/DRAM initialisations. Upon further analysis, a matrix was developed for skipping power up based on block/IP function resulting in further saving of simulation time. X propagation originating from non-resettable flops which was optimized at zero time with library cell hack deposits instead of tcl initialisation. Additionally, with CSS deployed across the project, savings on disk space became notably visible with reduction in build/simulation sizes. The CSS techniques resulted in saving close to 8000 man-hours (Table 2) over the course of the project.

| Scenario | Compile | Boot test | Mem test | CPU test | Power test | Disp test | PCIE test |
|---|---|---|---|---|---|---|---|
| Traditional | 23hrs | 38hrs | 71hrs | 60hrs | 62hrs | 66hrs | 72hrs |
| Skip Init runs | 23hrs | 28hrs | 45hrs | 42hrs | 38hrs | 40hrs | 55hrs |
| Skip Power up runs | 23hrs | 24hrs | 38hrs | 36hrs | 34hrs | 46hrs | 53hrs |
| GLS + RTL runs | 13hrs | 24hrs | 38hrs | 36hrs | 34hrs | 46hrs | 53hrs |
| GLS + FAKE runs | 12hrs | 20hrs | 29hrs | 30hrs | 26hrs | 32hrs | 38hrs |
| Improvement | 1.9x | 1.9x | 2.4x | 2x | 2.3x | 2x | 2.2x |

*Table 2. FCMAST improvement metrics via CSS*

- **Simulation Performance Optimization Wrapper (SPOW)**

SPOW deploys combination of simulation tool methods and Load Sharing Facility (LSF) optimisations to deliver upto 5x simulation improvements (Table 5).

**Capture and Replay (CNR):** CNR is a mechanism by which the simulation result is captured first and run/replayed on a different version of the same design without the need to run the test. In FCMAST architecture this flow is deployed to capture stimulus from RTL to replay at GLS for the first time. It additionally optimises runs for the targeted block and aids VCD generation for power estimation and IR drop analysis.

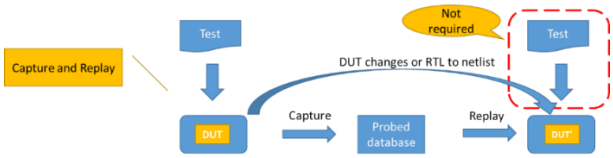


*Figure 9. Capture and replay from RTL to GLS*

**Incremental Checkpoint Utility - Save and Restore (ICU - SNR):** SNR is a method in which the common sequences of a SoC level test can be saved and any other test can be restored from the end of previous simulation snapshot. FCMAST additionally enhances the simulation time savings with extending the save timeframe to IP specific sequences with ICU. Every IP has its own set of initialization sequences that are required to be executed after the common sequences. Adding it as a part of the SoC common sequences will be counterproductive. ICU enables the creation of checkpoints at the end of the IP specific initialization sequences by leveraging the SoC snapshot (Figure 10).

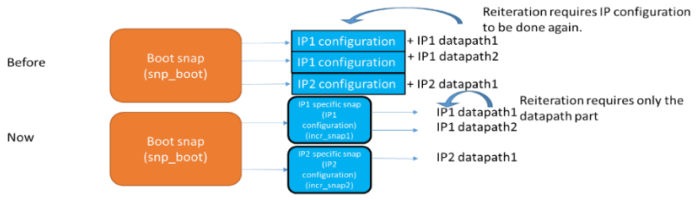


*Figure 10. Before and after incremental snapshot implementation*

| Criteria | Memory overhead (For 1 IP) | Avg Run time (For 1 IP) |
|---|---|---|
| Without ICU | 0 | 86.1hrs |
| With ICU | 31GB | 28.4hrs |
| Improvement | - | 3.02x |

*Table 3. Runs with and without ICU*

**Automatic Periodic Checkpoint Generation (APCG):** APCG plugin periodically saves the simulation snapshot of the run based on simulation time or wall clock time. The plugin has flexibility to program the number of snapshots to be saved to get rid of redundant snapshots. The periodic checkpoint generation (Figure 11) setting is based on the analysis of simulation kill, user mistakes, failure stop & rerun scenarios and regression run times. In Figure 11, the periodic checkpoint generation is represented by picking the number of snapshots that are stored at any time to be 3. Here, the parallel running tcl script spawns 3 individual breakpoints and each breakpoint maintains its own checkpoint. Another add-on to APCG plugin ensures that restarted simulations and their status are accurately represented in the HTML regression dashboards. This custom script ensures that the correct test command is replaced by the older test command so that when the abruptly ended run is restarted, it will appropriately modify the regression dashboard to reflect the correct number of tests and their run status.

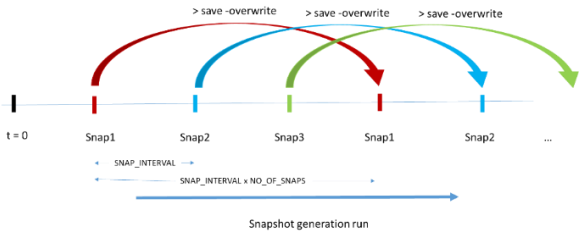


*Figure 11. Automatic Periodic Checkpoint Generator flow*

| Criteria for 1 IP | Memory overhead | No. of abruptly killed runs | Time spent on rerun of abruptly killed runs |
|---|---|---|---|
| Without APCG | 0 | 5 | 157h |
| With APCG | 90GB | 7 | 84h |
| Improvement ( %) | - | - | 46 |

*Table 4. Normal v/s runs with APCG*

**Profiling & Directed Access:** Simulation profiling and further analysis of the weight each component had resulted in modifying the testbench and RTL coding styles for optimal performance. The profiling activity resulted in tweaking the simulator with appropriate switches that are custom suited for the database resulting in up to 20% simulation speed improvements [2]. The simulation tool debug permission across the full compiled database was tweaked due to the confidence resulting from other FCMAST checks. Optimisation at signals level were carried with AI and ML scripts that restricts the debug access to signals that explicitly require permissions. Experiments, including dedicated machine runs, were carried out to understand, analyse and optimise the machine (LSF) on which the simulation is executed.

| Scenario | Boot test | Mem test | CPU test | Power test | Disp test | PCIE test |
|---|---|---|---|---|---|---|
| Access +RWC | 50hrs | 75hrs | 69hrs | 72hrs | 73hrs | 85hrs |
| Access +R | 38hrs | 71hrs | 60hrs | 62hrs | 66hrs | 72hrs |
| Afile | 32hrs | 65hrs | 56hrs | 57hrs | 60hrs | 65hrs |
| Dedicated | 31hrs | 63hrs | 52hrs | 54hrs | 54hrs | 63hrs |
| Improvement | 1.4x | 1.2x | 1.3x | 1.4x | 1.4x | 1.3x |
| With SNR | 31hrs | 29hrs | 33hrs | 25hrs | 28hrs | 40hrs |
| Improvement | 1x | 2.2x | 1.7x | 2.1x | 2.1x | 1.7x |
| w/o ICU & APCG Avg time for 100 tests | 31hrs | 156hrs | 120hrs | 98hrs | 186hrs | 264hrs |
| w/ ICU & APCG Avg time for 100 tests | 31hrs | 45hrs | 43hrs | 30hrs | 49hrs | 60hrs |
| Improvement | 1x | 3.4x | 2.8x | 3.2x | 3.8x | 4.4x |

*Table 5. FCMAST improvement metrics for external methods via SPOW*

# RESULT

FCMAST's modularity and ease to deploy across projects and different abstraction of the testbench resulted in huge savings in terms of manual effort the engineer spends, simulation run times and performance enhancement in the range of 6x-10x (Table 6). FCMAST has been implemented in 2 recent SoC DV projects for AR/VR and HPC 3DIC applications and it has provided consistent improvement in GLS execution.

| Scenario | Boot test | Memory tests | CPU tests | Power tests | Display tests | PCIE tests |
|---|---|---|---|---|---|---|
| Initial avg | 50hrs | 156hrs | 120hrs | 98hrs | 186hrs | 264hrs |
| FCMAST Optimized avg | 18hrs | 26hrs | 25hrs | 24hrs | 28hrs | 29hrs |
| Improvement | 2.7x | 6x | 4.8x | 4.1x | 7x | 9.1x |

*Table 6. Improvement metrics with FCMAST per IP test*

The methodology guarantees first run correctness and fast convergence to meet stringent timelines with its automated plug-ins. FCMAST has proven to save both precious man months and the costlier licensing, storage and infrastructure costs (Table 7). Future scope with integration of emulation into FCMAST and leveraging AI and ML tools for wavemining and automated debugs have already been evaluated and are under development stage. EDA tool vendors are also evolving their solutions based on the feedback and reports to further ameliorate the simulation performance.

| Initial Metrics | | | | | | FCMAST Improved Metrics | | | |
|---|---|---|---|---|---|---|---|---|---|
| Block | Total Tests | No. of netlist labels | Avg Iterations for first pass | Avg run time (hrs) | Total runtime (hrs) | Avg Iterations for first pass | Avg run time (hrs) | Improved Total runtime (hrs) | Net Improvement (X times) |
| A | 19 | 8 | 4 | 85 | 51680 | 1 | 48 | 7296 | 7.1 |
| B | 12 | 8 | 4 | 76 | 29184 | 1 | 33 | 3168 | 9.2 |
| C | 4 | 8 | 2 | 94 | 6016 | 1 | 42 | 1344 | 4.5 |
| D | 3 | 7 | 3 | 63 | 3969 | 1 | 35 | 735 | 5.4 |
| E | 2 | 8 | 2 | 67 | 2144 | 1 | 31 | 496 | 4.3 |
| F | 14 | 8 | 4 | 78 | 34944 | 1 | 38 | 4256 | 8.2 |
| G | 8 | 8 | 3 | 163 | 31296 | 1 | 101 | 6464 | 4.8 |
| H | 22 | 8 | 4 | 102 | 71808 | 1 | 44 | 7744 | 9.3 |
| I | 13 | 6 | 3 | 73 | 17082 | 1 | 24 | 1872 | 9.1 |
| J | 17 | 8 | 4 | 68 | 36992 | 1 | 28 | 3808 | 9.7 |
| K | 6 | 8 | 5 | 152 | 36480 | 1 | 86 | 4128 | 8.8 |
| L | 10 | 8 | 3 | 79 | 18960 | 1 | 31 | 2480 | 7.6 |

*Table 7. Consolidated initial v/s final FCMAST GLS metrics*

# REFERENCES

[1] Harshal Kothari, Vinay Swargam, Sriram Kazhiyur Soundarrajan, Somasunder Kattepura Sreenath, "*A Novel Approach to Expedite Verification Cycle using an Adaptive and Performance Optimized Simulator Independent Verification Platform Development*", DVCON Europe 2022.

[2] Harshal Kothari, Eldin Ben Jacob, Sriram Kazhiyur Sounderrajan, Somasunder Kattepura Sreenath, "*Centralized Regression Optimization Toolkit (CROT) for expediting Regression Closure with vManager & Xcelium Performance Optimization*", CadenceLive India 2021.