2021 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE
OCTOBER 26-27, 2021

Centralized Regression Optimisation Toolkit (CROT) for expediting Regression Closure With Simulator Performance Optimisation

Harshal Kothari, Associate Staff Engineer, SSIR, Bengaluru, India (harshal.k1@samsung.com)

Pavan M, Senior Engineer, SSIR, Bengaluru, India (pavan2019.m@samsung.com)

Ajay Vamshi Krishna, Senior Engineer, SSIR, Bengaluru, India (a.krishnaka@samsung.com)

Eldin Ben Jacob, Associate Staff Engineer, SSIR, Bengaluru, India (eldin.jacob@samsung.com)

Sriram Kazhiyur Sounderrajan, Associate Director, SSIR, Bengaluru, India (sriram.k.s@samsung.com)

Somasunder Kattepura Sreenath, Director, SSIR, Bengaluru, India (soma.ks@sasmung.com)

***Abstract*—Few of the main requirements for Design Verification closure are clean regression and coverage closure with lesser turnaround time with every design label. Centralized Regression Optimization Toolkit (CROT) based on Smart Centralized Regression(SCR) [1] is a utility where all the DV tests from each IP/subsystem/SoC are automated to run centrally to ensure 100% pass rate qualifying a bug-free DUT with each design release label and gather requisite coverage metrics.**

Keywords—regression; verification; automation; coverage; simulation time

I. INTRODUCTION

The simulation time depends on the size and complexity of the design, testbench and simulator performance and Load Sharing Facility (LSF)/Compute availability. In addition to RTL simulations, with X-propagation, power aware simulations and gate level simulations closure being mandatory for DV closure, the turnaround time for regression closure has increased exponentially. With the increasing number of subsystems and IPs in an SoC, the number of tests increase exponentially thereby causing a surge in net run times and license requirement if the overall performance is not ameliorated to its most optimized state. The CROT addresses each of these concerns by managing the regression process efficiently in an optimized manner without compromising on the quality of verification and prudently saving up to 60% time and resources and eventually the cost saved up per project.

The CROT employs 3 key aspects to expedite the CR process:

- Live tracking via HTML dashboard ➔ Improved status reporting, failure analysis, rerun, and automated incremental coverage merging.
- Simulation speed optimization ➔ Profile, Analyze, Perf Knobs, Save and Restore (SnR).
- LSF and compute optimization ➔ Tracking and projecting the LSF scheduling and license acquisition mechanism.

II. LIVE TRACKING VIA HTML DASHBOARD

Logistical challenges while running a centralized regression are automated initiation, live report presentation, easy tracking & debug of failures and coverage analysis. The HTML user interface is intuitive, user-friendly and simple. The main aim is to reduce the regression time, & iterations and ease of tracking & first-cut failure analysis. All the features are designed keeping this in mind. When the regression iterations are reduced, the number of licenses required for closing the regression activity comes down which in turn brings down the cost of license involved.

Once the design release is available, CROT analyses the test-plan spreadsheet and handles the comprehensive list of compiles, elaborations, boot snapshot and smoke test which need to be run to ensure sanity. If smoke tests of particular IPs or BLKs fail, full regression is not launched till smoke suite is clean. This directly results in preventing the usage of LSF and License by filtering tests which we know is anyways going to fail and these tests are directly marked fail even without firing the simulation. Regressions are run on Cadence vManager tool with Xcelium simulator and report is generated via batch mode. The CSV report are custom generated for each iteration. To enable history and track the progress of both test and project development, a repository of each CR is created for the different flavours of the runs like RTL, low-power enabled, XPROP enabled etc. The real time regression report and merged coverage report is generated in HTML format with a customisable refresh rate. The HTML has the capability of triggering the coverage merging based on the list of tests/suites selected by the user as well. Email alerts are also automatically sent with a list of first failures and unique run ID of the test for quick analysis and to alleviate the need to sift through bulky log files and dump. The user can check the status of the tests on the live HTML webpage dashboard (Figure 1) and provide fixes resulting in saving the precious GUI licenses by at least 77%. We were also able to save the need for manual intervention and engineer bandwidth by 25% with these enhancements across projects.

Coverage summary

Last updated 16:25

TOTAL	PASS	FAIL	RUN	WAIT	OTHER
2541	**2541**	**0**	**0**	**0**	**0**

	%PASS	%FAIL	%RUNNING	%WAITING	%OTHERS
	100.00	**0.00**	**0.00**	**0.00**	

BLOCK	PASS	FAIL	RUN	WAIT	OTH	BLOCK TOTAL	Pass%	COMPILATION ISSUES	TEST INCOMPLETE
A	153	0	0	0	0	153	100.00	0	0
B	205	0	0	0	0	205	100.00	0	0
C	381	0	0	0	0	381	100.00	0	0
D	182	0	0	0	0	182	100.00	0	0
E	245	0	0	0	0	245	100.00	0	0
F	369	0	0	0	0	369	100.00	0	0
G	427	0	0	0	0	427	100.00	0	0
H	230	0	0	0	0	230	100.00	0	0
I	322	0	0	0	0	322	100.00	0	0
J	27	0	0	0	0	27	100.00	0	0

Owner summary

Jobs summary

Figure 1. Live HTML Regression Dashboard

A. *Pre-processing*

Before the start of regression all the test-vectors which are obtained from the testplan excel sheet are analyzed for correctness. Wrong test-vectors naming will take unnecessary licenses and failures will be reported. Such incorrect commands will require individual edits which results in slowing down the turnaround time. The preprocessing does the quality check of the content filled by the user. By pre-processing we can avoid manual errors and hence few iterations of regression in order to get 100% pass rate. Few important test cases for each IPs are identified as smoke tests. We can choose to run the full regression or only the smoke suite. Smoke tests are run

when a major RTL change has been incorporated or when the testbench has got an overhaul we check the environment changes with the smoke test suite to see if the important datapath tests are passing cleanly. This will save a lot of resources being used up front. Let's say for a suite of 300 tests, we have 1 smoke test, if the smoke fails, we save on running 299 tests and they are marked as fail even without using LSF slot/License. This has resulted in close to 58% of tests marked as fail on the recent project.

B. HTML Dashboard web-page

Once the tests are launched, the complete regression can be tracked by this automated HTML. This HTML gives a hierarchical summary of the whole SoC regression. It gives a complete block wise report and inside each block IP wise report. For each IP, it shows all the test cases and its status. All the information needed about the test case- its run path, job-id, failure reason, how much time the test ran, start time, end time and how much time it waited to acquire license will be summarized. It captures all the warnings, errors and dumps to a separate file which can be accessed from the HTML itself.

One step higher in the hierarchy, owner can find the details about the block. It shows all the IPs in that block and its individual pass, fail, running, waiting status and owner name who's verifying the IP. The average run time of each IP along with long running test cases (which can be set based on project), maximum and minimum run times are a part of the dashboard. Abnormalities can be identified and looked into. It summarizes how many failures due to compilation issues and tool issues separately. So too many failures in these two categories need immediate attention.

BLOCK A

TOTAL	PASS	FAIL	RUN	WAIT	OTHER
153	153	0	0	0	0

	%PASS	%FAIL	%RUNNING	%WAITING	%OTHERS
	100.00	0.00	0.00	0.00	0.00

IP_NAME	PASS	FAIL	RUN	WAIT	OTH	BLOCK TOTAL	Pass%	COMPILATION ISSUES	TEST INCOMPLETE	AVG TIME(hrs)	MIN TIME(hrs)	MAX TIME(hrs)	OWNER
IP_1	79	0	0	0	0	79	100.00	0	0	2.31	1.63	3.03	owner_1
IP_2	17	0	0	0	0	7	100.00	0	0	2.45	1.69	3.20	owner_2
IP_3	20	0	0	0	0	2	100.00	0	0	2.33	1.73	3.11	owner_3
IP_4	13	0	0	0	0	13	100.00	0	0	2.46	1.61	3.31	owner_4
IP_5	22	0	0	0	0	22	100.00	0	0	2.60	1.71	3.22	owner_5
IP_6	2	0	0	0	0	2	100.00	0	0	2.70	1.66	3.06	owner_6

Figure 2. HTML BLOCK Summary

One more step higher in the hierarchy will give a detailed summary (Figure 1) about all the blocks' status and average runtime. One can get to know by how much percentage a block's regression is completed, compilation issues and tool issues. These details will give help in identifying on which block to prioritize.

C. Smart Coverage Merging

As the tests with coverage enabled complete, its coverage is merged and published in the HTML. This saves a lot of time for the engineers to run individual test at their end and merge coverage. They can directly analyze un-merged coverage on Cadence IMC tool with the regression coverage database. IPs of interest can be set for which the coverage will be enabled resulting in simulation speed improvement compared to a full SoC coverage enabling. Incremental coverage merging happens once a test completes i.e. the new test is merged to the previous database

hence saving resources. This is flexible to consider coverage waivers also and will be showed as waived coverage in the HTML.

Generally, 20-30% of the tests run contribute to 75-85% coverage. After the first regression completion, the tests were ranked to extract the list of tests resulting in 80-85% coverage. Subsequent regressions harnessed this coverage to skip running significant amount of tests with coverage enabled. Furthermore, the coverage is dynamically merged on the fly with every fixed % of incremental completion of passed tests in CR via automation, thus avoiding repetition of coverage database merging. With these advancements leading to reduction in the overall number of tests runs with coverage enabled, the net regression TAT over multiple iterations reduced by about 25%. This feature also reduces the need to use another tool to view the merged coverage which requires a separate license to be acquired. This saves tremendous licensing costs and issues which can arise when multiple people work on the coverage analysis at the same time creating license crunch and reduced efficiency in analysis.

Sheet 1: *Summary*

Title: top Dut Toggle Coverage Summary				
Author: owner2				
IP BLOCK	Total Signals	Sim Coverage	Unchecked Items	Waived Coverage
1.BLK_A.IP_2	226	83.63%	37	83.63%
2.BLK_B.IP_4	226	83.63%	37	83.63%
3.BLK_C.IP_3	447	85.46%	65	85.46%
4.BLK_D.IP_1	203	84.73%	31	84.73%
5.BLK_F.IP_2	299	78.93%	63	78.93%
6.BLK_F.IP_4	211	84.36%	33	84.36%
7.BLK_G.IP_2	242	85.12%	36	85.12%
8.BLK_G.IP_3	242	85.12%	36	85.12%
9.BLK_J.IP_4	209	79.90%	42	79.90%

Figure 3. HTML BLOCK Summary

D. Owner Summary

A separate owner wise summary is available. It allows project managers to track regression results with owner wise progress and the IPs they are owning. Any discrepancy can be addressed.

OWNER SUMMARY						
ONWER NAME	IP NAME	PASS	FAIL	RUN	WAIT	OTHER
owner1	IP_1	49	0	0	0	0
	IP_2	42	0	0	0	0
	IP_3	25	0	0	0	0
	IP_4	40	0	0	0	0
owner2	IP_5	20	0	0	0	0
	IP_6	15	0	0	0	0
	IP_7	15	0	0	0	0
owner3	IP_8	10	0	0	0	0
	IP_9	93	0	0	0	0
	IP_10	2	0	0	0	0
	IP_11	9	0	0	0	0
	IP_12	13	0	0	0	0

Figure 4. HTML OWNER Summary

E. Job summary

A detailed report on number of jobs submitted each day is recorded and reflected on the job summary HTML. It encapsulates the number of passes, fails and running jobs on a given day. Most importantly, it shows average wait time and resource acquisition time. This feature is helpful to understand if we are facing any license crunch or loading of snapshot is taking more time than expected and can be addressed immediately which in turn will help smooth and faster closure of regression. It also tracks disk space usage and sends email alerts so that we do not run out of space which leads to simulation failures and rerunning of all those tests taking more licenses and time.

Date	No of jobs Submitted	No of jobs Completed	Avg Wait Time(s)	Avg Resource collected Time(s)	Pass Ratio	Fail Ratio	Running jobs	Disk usage
11/20/20	2343	2122	15.91	10721.95	89.68	10.32	190	40T 37T 3.9T 91% /user/disk0
11/21/20	236	315	319.45	2349.00	94.60	5.40	108	40T 37T 3.9T 91% /user/disk0
11/22/20	20	53	6.21	3923.78	94.34	5.66	70	40T 37T 3.8T 91% /user/disk0
11/23/20	9	19	6.52	1054.00	15.79	84.21	50	40T 37T 3.1T 93% /user/disk0
11/24/20	25	15	13.82	2180.35	100.00	0.00	56	40T 37T 3.2T 93% /user/disk0
11/25/20	36	5	6.25	12992.75	80.00	20.00	77	40T 37T 3.2T 93% /user/disk0
11/26/20	33	34	656.49	6607.12	29.41	70.59	50	40T 37T 3.2T 93% /user/disk0
11/27/20	54	54	4643.05	7862.59	68.52	31.48	47	40T 37T 3.2T 93% /user/disk0
11/28/20	27	13	5101.59	1819.48	92.31	7.69	35	40T 37T 3.1T 93% /user/disk0
11/29/20	1	15	5.21	1533.03	100.00	0.00	19	40T 37T 3.1T 93% /user/disk0
11/30/20	44	14	1.16	2922.18	85.71	14.29	36	40T 37T 3.1T 93% /user/disk0
12/1/20	68	44	2.26	12642.36	97.73	2.27	33	40T 37T 3.1T 93% /user/disk0
12/2/20	3	27	34.57	7421.00	96.30	3.70	2	40T 38T 3.0T 93% /user/disk0

Figure 5. HTML JOB Summary

F. POST-PROCESSING

We can update the filters for the failure categorization at any stage of regression. Comprehensive checkers and filter files are developed in order to have zero fake passes which can lead to a potential bug in silicon. There are known fake failures as well for which filters are added. This results in saving engineer's precious time by not looking into known fake failures.

III. SIMULATION SPEED OPTIMIZATION

A. PROFILING

Cadence Xcelium simulator memory profiling [3] (xmprof) was used to analyse the major contributors utilizing simulation resources, identify avenues for optimization like always on clocks and assertions, poorly written HDL code and design logic consuming high memory to execute. For instance, simulator dependent $xm_mirror was being used in multiple places which is an expensive method especially if it crosses over primary snapshot and incremental snapshot boundaries with high frequency clocks. There were multiple clocks being used across various checkers which caused additional overhead. We have changed the usage from $xm_mirror to assign statements to reduce this and we were able to improve the run times by 1-2%. With overall profiling, average run times were alleviated by 5-7%.

	Without Profiling	With Profiling	% improve
Average Elab Time (h)	0.5	0.47	6.0
Average Sim Time (h)	7.2	6.75	6.3

Table 1. Comparative Analysis of Profiling Results

B. *Auto-analysis and perf knobs*

Sim speed auto-analysis utility was used to perform an in-depth audit of the options used during elaboration and simulation stages and identify the switches causing bottlenecks and eating simulator efficiency. The debug access authority was provided only to requisite instances and modules. The performance switches along with latest tool versions resulted in improving the elaboration and simulation run times by 15-17%.

	Without Auto Analysis	With Auto Analysis	% improve
Average Elab Time (h)	0.47	0.39	17.0
Average Sim Time (h)	6.75	5.7	15.6

Table 2. Comparative Analysis of 100 CR jobs with auto-perf-analysis

Figure 6. Snippet of Auto-perf-analysis Utility Report Output

C. *"SNR" (SAVE AND RESTORE)*

By making each Verification Entity (VE) compatible to Cadence Save and Restart flow [2], we could bring down the average sim time by an additional 49%. In SoC simulations the basic boot, clock, power and memory controller initialization are required to bring up the design for further testing for each IP/Blocks. Making a snapshot of this boot sequence as a base for other tests resulted in saving 4-6ms of sim time for each test. Furthermore, common initialization of PHY, link, controller etc. which are required across multiple tests for IPs like PCIe, CSI, DSI can also be saved in this snapshot. With ~2500 tests in regression the regression TAT reduced by 47-54% (Table 3)

BLK	No. of Tests	Average Test run time (hours)		% improve
		Non-SNR	SNR	
A	153	4.9	2.5	49.0
B	205	6.3	3.1	50.8
C	381	7.2	3.5	51.4
D	182	4.8	2.3	52.1
E	245	5.8	2.7	53.4
F	369	7.9	3.8	51.9
G	427	4.1	1.9	53.7
H	230	3.6	1.9	47.2
I	322	7.9	3.9	50.6
J	27	4	3.2	20.0
TOTAL	2541	5.7	2.9	49.0

Table 3. Comparative Analysis for each Block with SNR

D. MISCELLANEOUS OPTIMIZATION

Includes smart coverage merging, running with limited debug access capabilities, auto generated access file for debug access permissions, no waveform dumping, disk usage tracker and cleaner to avoid crashes and rerun et al. The Smoke test column in attribute sheet, fixes common VE issues before even starting the full regression. A script is being used to search and collate all active forces and warnings during each and every test run in an excel sheet. IP owner audits it and removes unnecessary forces and resolves warnings through which we decreased the possibility of getting bugs on Silicon. The CHECK_ATTRIBUTE script, identifies the issues with test vectors and makes sure that improper test vectors won't be a part of CR. The summary of issues is collated in a log which is shared with test owners so that fix happens at the test owner's end. Through the use of filters on various kinds of errors, we make sure that there no fake passes/failures.

As a result of above upgrades, we were able to complete first iteration of CR in 2-3 days which saved over 60% time and resources for 25+ custom lib compiles, DUT elaboration, snapshot and test runs. An AI based mechanism to detect unchanged design between subsequent releases to reduce the tests required to qualify the DUT is under development.

IV. LOAD SHARING FACILITY AND COMPUTE OPTIMIZATION

Total run time of the regression (Estimated run time) was calculated from the regression log files based on LSF utilization and the actual run time from regression start to end (Actual run time) collected for IPs and SoCs. Ideally, estimated run time should be very close to actual run time. However, various factors involved in LSF job scheduling and license(s) acquisition mechanism result in variation of actual run time. An LSF profiler & analyser script was developed to aid in tracking and projecting the LSF scheduling and license acquisition mechanism. The profiler monitors the status of the running jobs periodically and dumps out log report for analysis. With regression profiler, the turnaround time improved between 15– 25%, LSF utilization improved significantly from 84% to 97% and license usage improved between 25–40% resulting in huge cost savings and effective utilization of resources.

LSF/License Utilization	Normal Flow	CROT flow	% improve
No. of hours License Utilized/iteration	14000	10500	25.0
LSF utilization	84%	97%	13%

Table 4. LSF and license acquisition optimization

Table 5 below shows the improvement of about 57% after incorporating all above enhancements in our CROT environment which is project agnostic and can be extended to any IP/SoC DV. The pilot has been successfully

implemented for our current SoC DV project and it shows tremendous improvement over traditional regression approach with real-time HTML tracking and reporting, sim speed enhancement with profiling, auto-perf-analysis, SNR and targeted incremental coverage merging & LSF and license usage optimization.

BLK	No. of Tests (n)	Mean original run time per sim (T)	Net Test run time (hours) with no. of iterations = 15				Mean CROT run time per sim (T')	Net % improve
			Base (n*T*15)	Profiling	Prof+autoanalysis	Prof+autoanalysis+SNR		
A	153	5.3	12164	11256	9520	5738	2.5	52.8
B	205	6.9	21218	20104	17659	9533	3.1	55.1
C	381	8.2	46863	43783	35220	20003	3.5	57.3
D	182	5.8	15834	14704	11954	6279	2.3	60.3
E	245	6.7	24623	22985	19901	9923	2.7	59.7
F	369	8.9	49262	46178	37520	21033	3.8	57.3
G	427	5.1	32666	31092	26237	12170	1.9	62.7
H	230	4.2	14490	13601	11574	6555	1.9	54.8
I	322	8	38640	36087	28338	18837	3.9	51.3
J	27	4.7	1904	1787	1450	1296	3.2	31.9
TOTAL	2541	7.2	257661	241577	199373	111365	2.9	56.8

Table 5. CROT consolidated results

V. RUNTIME TO COST COVERSION

The improvements done raised the curiosity to document and quantify the savings done per project. Typically, we come across SoCs with average 6000 test cases each running with an average of 24hrs. Over the whole cycle of verification, typically around 14 iterations of centralized regression are run. We need around 10-12 different licenses for different IPs in a SoC to complete the regression with an average cost $25 each. Table 6 projects the cost estimate for the licenses with and without optimization. From table 5 we have got net% improvement on run time more than 50%. This brings down the average run time of SoCs to 12 hrs. Now we can see average estimation of cost (in USD) coming down by almost 50%. This is the case for 1 project. When we consider it across entire Samsung where typically around 100s of DV projects are being executed parallel, the amount saved will be humongous if the flow is incorporated. The tool as a package will have a huge impact (~50%) on saving ASIC project licensing/LSF costs across the organization.

Avg no of tests = 6000

Avg Runtime	Avg tests * Runtime	No of Iteration	Run time(hrs)	Run time(weeks)	Lic Cost(USD)	Total Cost(USD)
24	144000	14	2016000	12000	251	3012000
12	72000	14	1008000	6000	250	1500000

Table 6. Runtime-Cost comparison

VI. CENTRAL REGRESSION IN GATE LEVEL SIMULATIONS

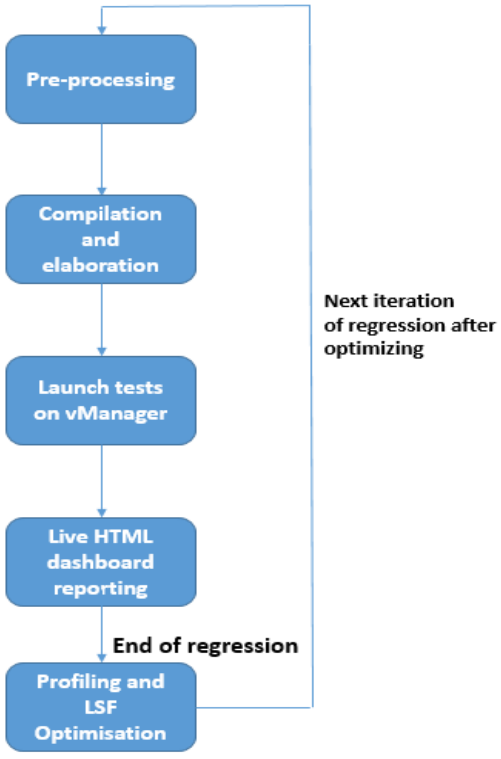
PRE Gate Level Simulations (unit delay simulations with DC netlists) runs take high times to complete and Timing GLS would take even more time because of SDF compilation and back-annotation. Using the above Optimization techniques in GLS showed efficiency in bringing run times down by around 25-35% which saved around 10 days of time. Below table shows the reduction of total run time when using the Profiling, auto analysis and SNR techniques. Although the number of tests are less compared to RTL simulations, CROT deployment for GLS is very crucial as the turnaround time for same test is considerably higher. For example, if we have a test that runs for around 1 hr in RTL simulations and the DUT compile and TB elaboration time is ½ hr, the post PNR netlist

based DUT takes 6 hours to compile and elaborate and around 8 hours to complete the simulation, which is depending on the resource available. Thus the improvements introduced via CROT regarding total time taken for the simulations makes it very important to achieve 100% passing gate level regression before tape-out.

BLK	No. of Tests (n)	Mean original run time per sim(T)	Net test run time(hours) with no. of iterations=10				Mean CROT run time per sim(T')	Net % improve
			Base(n*T*10)	Profiling	Prof+autoanalysis	Prof+autoanalysis+SNR		
A	13	27	3510	3408	3015	2587	19.9	26.3
B	18	32	5760	5575	4970	4199	23.3	27.1
C	21	23	4830	4680	4182	3410	16.2	29.4
D	14	34	4760	4588	4014	2922	20.9	38.6
E	6	30	1800	1749	1544	1130	18.8	37.2
F	12	82	9840	9495	8323	6632	55.2	32.6
G	19	49	9310	8956	7843	5605	29.5	39.8
H	8	32	2560	2485	2208	1751	21.9	31.6
I	12	64	7680	7434	6819	5399	45	29.7
J	2	43	860	833	798	701	35.1	18.5
TOTAL	117	43.5	5090	4920	4231	3433	29.3	32.6

Table 7. GLS CROT consolidated results

VII. CROT FLOW



Test vectors are taken directly from the exhaustive test plan which has the DV attribute sheets per subsystem/IP. These test vectors are run through the preprocessing scripts to check initial errors. A complete list containing what builds needs to be run is generated. In this step we need to enable coverage and hierarchies we are interested in. We also choose which blocks to run and prioritize whether to run the full regression or smoke test. In the next step that build list is launched by the CROT tool. Once the compilation and elaboration step is completed the tests are

launched in vManager. As the tests progress in parallel to it the live HTML shows the progress of the regression along with merging coverage, and other summaries as mentioned above. Once the regression is complete, Profiling and LSF optimization is done to analyze if the regression time can be reduced further. After taking the changes into consideration and fixing the test plan errors next iteration is started in the required release.

VIII. CONCLUSION

Typically, during SoC DV lifecycle, CR is run for 50~75 iterations traversing RTL development, PARTL, RTL freeze stages to unit delay and timing GLS. The current SoC taken up for the regression had a total of 6000+ tests and the individual sims had a mean run time of about 4hours with RTL. With 100 licenses in parallel, it would take around 6-7 days to get initial results. The bug fixes and their reruns would make the regression turnaround time exceed 10 days in RTL to about 30 days in GLS. All above activities are time and bandwidth consuming manual processes. With time to market and first pass silicon becoming a key differentiator, it is imperative to strategically reduce this cycle-time which often consumes a lot of engineering bandwidth resulting in productivity loss. With CROT deployment in our DV project, we were able to save around $1,500,000 in licensing cost and close to 200 net engineering hours per week per DV team with our enhanced interactive GUI.

IX. FUTURE SCOPE

Using Simulator Independent Verification Platform Development (SIVPD), we can remove many dependencies on tool limitations as CROT is intended to be tool and project agnostic. We are planning for few enhancements such as killing few SNR jobs that take huge time when there's a problem with loading incremental snapshot automatically, auto rerunning the tests which failed due to some known errors which can be ignored, checker audit for elaboration/simulation options passed to simulator, audit on DUT hacks using forces and mirror etc.

REFERENCES

[1] Published Paper: Smart Centralized Regression (SCR): An Efficient Way of Managing Regression and Analyzing Failures – DVCON 2017: Authors: Sriram Kazhiyur Sounderrajan, Somasunder Kattepura Sreenath. https://dvcon-india.org/content/event-details?id=241—11

[2] Published Paper: Efficient Methodology for Design Verification Closure for Complex SoCs using Save and Restore – CDN Live 2019: Authors: Arushi Mittal, Vasundhara Gupta, Raghavendra Bidanagere R, Somasunder Kattepura Sreenath

[3] Xcelium Profiling User Documentation: https://support.cadence.com/apex/techpubDocViewerPage?xmlName=xcelium_kpns.xml&title=Xcelium%20Known%20Problems%20and%20Solutions%20--%20Xcelium%20Profiler%20Known%20Problems%20and%20Solutions%20-%20%20Product%20Level%202:%20PROFILER%20&hash=XceliumProfilerKnownProblemsandSolutions-ProductLevel2:PROFILER&c_version=21.03&path=XCELIUM_KPNS/XCELIUM_KPNS21.03/Xcelium_Profiler_Known_Problems_and_Solutions.html#XceliumProfilerKnownProblemsandSolutions-ProductLevel2:PROFILER