Accelerating SOC Verification Using Process Automation and Integration

Seonghee Yim[1] Hanna Jang[1] Sunchang Choi[1] Seonil Brian Choi[1]

[1]Samsung Electronics, Hwaseong, Korea.
({sh.yim, hn01.jang, s86.choi, seonilb.choi}@samsung.com)

Abstract:
As a competition in the smart device industries is getting fierce, the importance of TAT (Turn Around Time) reduction in SoC (System on a Chip) development increases dramatically. The proposed system could combine cross-disciplinary workflows together, automate execution, explore test results and check if design/verification issues are satisfied with the required criteria. The system enables users to manipulate and configure process steps with parametric input/output data and automates multiple simulations/emulation scenarios, and as a result, it is expected to greatly improve efficiency of SoC development process by reducing manual errors and accelerate product design.

1. Introduction:

While TAT reduction gets more important and difficult, the size and complexity of SoC increase as well, meaning it is getting harder to meet TAT on time. In order to meet TAT on time, SoC is broken into sub-blocks. In addition, the overall SoC development process is divided into 3 sub-processes: the first stage is an IP-level design/verification, the second stage is a block-level which includes various design/verification stages and the last stage is a top-level, full SoC-level design/verification stage. As a SoC project and the development process are divided into sub-parts, more manpower and efforts are required in the SoC project. In general, test benches are developed for functional verification in every design/verification stages. Regression tests are followed by functional verification. Suitable input and output data are produced while regression tests are performed. Design/verification engineers who are in charge of their own blocks develop and run their own scripts and application in order to automate design/verification processes. The script files and applications are developed and tested in users' development environments as described in Figure 1. As the complexity of SoC design increases, it is becoming more difficult and tricky job to manage the increasing number of user-specific development environments.

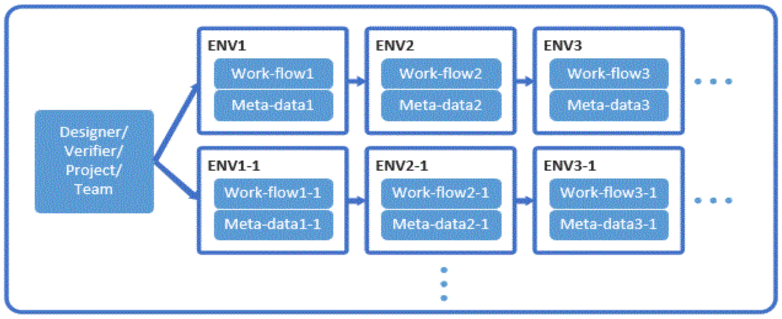


Figure 1: Various Scenarios by Designers, Verification Engineers, Projects and Teams

In this paper, a system for design/verification process automation and integration is proposed to improve efficiency in a collaborative work process in SoC development.

2. Related Works:

a. Continuous Integration

In a collaborative project, there are source codes files generated by developers involved in the project. When source codes developed by a developer are merged into a project, there may be some critical problems: the project may not be executed successfully because of inconsistent development environments throughout the developers.

Another problem is an inconsistent version control. Continuous integration is a concept to prevent the inconsistent development environment and version control problems. The most commonly used tool is Jenkins.

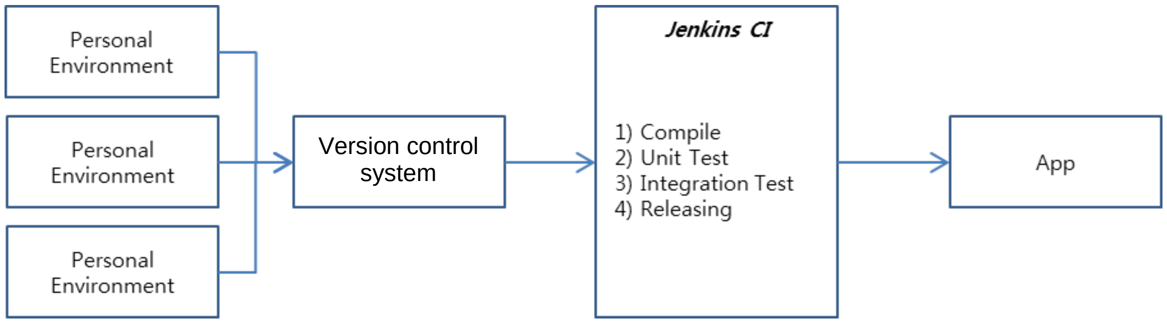


Figure 2: Continuous Integration Flow of Jenkins

The continuous integration flow with Jenkins is shown in Figure 2. First, all developers share source codes through version control system or other configuration management tools. Jenkins takes the codes from version control system, checks failures and errors and downloads relevant libraries from the repositories. And then Jenkins run test scenarios to check if source codes are correctly programmed. Only if the test scenarios are done successfully, the source codes are released to the operating servers.

b. Triage

Triage is a military medical term, which means a process of classifying and scheduling patients by their injury severity. Nowadays, this term, triage, has been widely used in IT (Information Technology) and business industries. For example, an IT operation department triages issues to decide which problems are urgent most. The top priority issues are handled as soon as they arise, and the issues in a medium-priority arise when there is no issue with a higher priority than themselves. If there is no medium-priority issue, low-priority issues are handled. However, in the worst cases, the low-priority issues might not be dealt unless their priorities are reassessed as a higher priority level. It's crucial to deal with high-priority requirements as soon as possible to meet the project plan on time. In agile software development, requirements are typically triaged at the start of each iteration. Because an iteration is a short development cycle, it's crucial to deal with high-priority requirements quickly to ensure that they are satisfied in time. In this paper, the term, triage, is used for priority of failures and errors. Failures and errors are triaged by some criteria, and debugging failures is performed based on the priority which is a result of triage.

c. SoC Design and Verification Flow

The design/verification of the SoC HW (Hardware) starts at the SoC specification stage as shown in Figure 3. It proceeds in a sequence of IP development, block development, and SoC integration. Not all progresses are performed in a strict sequence manner, meaning the order is not a tail-to-tail sequence. Instead, some progresses can be overlaid and proceeded in parallel.

The development of IP is carried out in accordance with SoC's development schedule. As the IP development begins, the verification of the IP starts, which IP will be used in the progress of determining the specification of SoC, and which version will be used in the SoC. In addition, block configuration is determined according to SoC specification, and block development is started. Block design/verification includes an Inter-IP connection in the block and functionalities of the block. In parallel with this, inter-block connection starts at the top SoC perspective. During the connection of inter-block, design of SoC top module functionalities are followed. Figure 3 shows this process over time. This paper defines verification steps for each development stage and defines them as Verification Level (VL). In VL1, IP-level design/verification is performed, and block-level design/verification is started following. In VL2, IP-level design/verification and block-level design/verification are mainly performed, and design and verification for SoC top is started. IP-level design/verification is done before VL3. In the VL3 and VL4, the design/verification of blocks and SoC is mainly performed. The difference between VL3 and VL4 is regression test. In VL4, regression test for blocks and SoC is performed mainly.

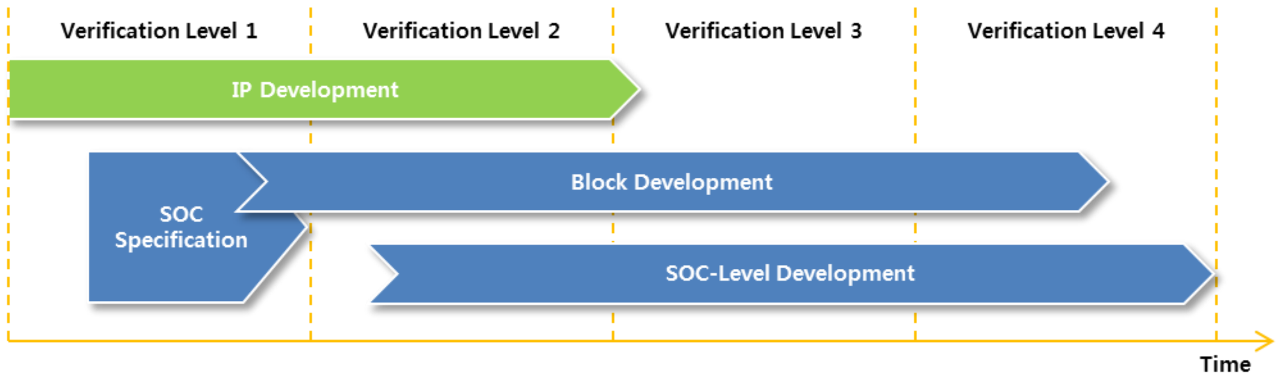


Figure 3: SOC Development/Verification Flow (Level).

3. Proposed System:

This paper proposes the integration of the verification environment and its processes. There are three methods to integrate design/verification processes: a top-down method, a bottom-up method and a hybrid method of top-down and bottom-up method. The top-down method is forcing current environments and processes to be integrated into a unified one, and the bottom-up method creates an interface to unify various kinds of environments and processes while preserving the current development ones. The hybrid method is a mixed one of the top-down and bottom-up method. The ultimate method of integration is the last third method. On the first trial of integration, it is hard to step into the first and third methods directly because of verification stability and risk. For that reason, this paper adopts the second method that is based on bottom-up way. Another reason is that the current verification environment has long been optimized for each verification engineer and each verification unit. The environments are already optimized in many parts of each unit but there still remain inefficient parts. It is expected that many of the remaining inefficient parts can be efficient as they are integrated into the platform proposed in this paper.

Another main idea of this paper is automation. While the basic processes of current design/verification are preserved, the proposed system maximizes the efficiency of building environment for design verification by automating manual tasks. The quality of verification environments can be maintained high and can be built quickly by removing errors which are caused by human errors. In addition, if the flow needs to be changed, the change can be handled more flexibly. The proposed system could be settled down as a new standard workflow, out of the traditional verification methodology guidance. In this paper, the system is focused on each verification level of SoC verification.

Many design/verification environments are diverse in each design stage such as IP, block and SoC level in actual SoC projects. Each design unit and design team has their own well-structured workspaces such as directory structures and verification types. In the same design stage, some standardized workspace structures can be created, however, it is very difficult to standardize workspace structures in different design stages. In order to integrate the design/verification environment, it is preferable to support all environments rather than formalizing all stages of environment. In this paper, 5 modules were developed as shown in Figure 4 to integrate and automate various verification environments. The first module is a flow controller.

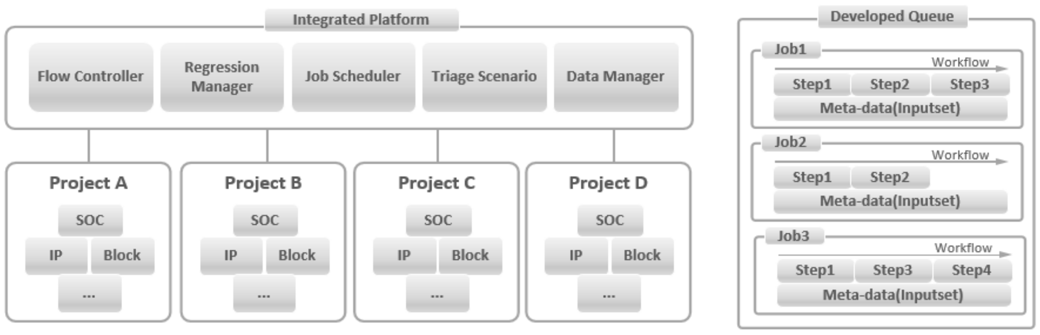
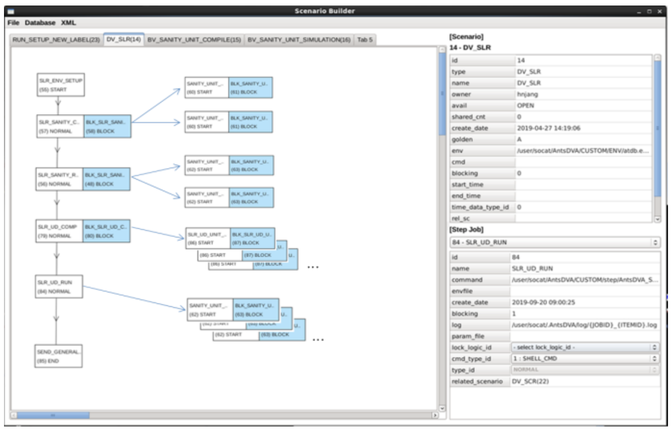


Figure 4: Integrated Platform and Internal Queue for Automation and Verification

1) Flow controller

Flow controller system is that can unify and automate work flow that exists in each team and step of verification stage. And it is similar to the existing Continuous Integration (CI [https://jenkins.io/]) services while the controllability of workflow has been improved. This is because during the SOC verification period various workflows are integrated into the proposed automation system. The proposed system treats each workflow as a unit of work. A workflow consists of a sequence of steps where a step is pre-defined execution commands or operations. A sequence of steps is defined as scenario. A scenario starts from start-step and ends at end-step. There may be several steps between start-step and end-step, which are defined as normal step. Each step has a link information (head IP, tail ID) that connects to each other. Furthermore, each step is divided into three steps of pre, body and post-step. On the outside, only the body of step is visible, and pre/post-step is responsible for the operation to supplement body-step. The scenarios and the steps are reusable, and the operation of each step can be determined at the time the work flow applied to the action is executed. So, we can insert various conditional input set Even if using the same step and scenario. Figure 5 is an example of the actual step and scenario used.


Scenario Builder
File Database XML
RUN_SETUP_NEW_LABEL(23)
DV_SLR(14)
BV_SANITY_UNIT_COMPILE(15)
BV_SANITY_UNIT_SIMULATION(16)
Tab 5
SLR_ENV_SETUP (55) START
SLR_SANITY_C... (57) NORMAL
BLK_SLR_SANI... (58) BLOCK
SLR_SANITY_R... (56) NORMAL
BLK_SLR_SANI... (48) BLOCK
SLR_UD_COMP (79) NORMAL
BLK_SLR_UD_C... (80) BLOCK
SLR_UD_RUN (84) NORMAL
SEND_GENERAL... (85) END
SANITY_UNIT_... (60) START
BLK_SANITY_U... (61) BLOCK
SANITY_UNIT_... (62) START
BLK_SANITY_U... (63) BLOCK
SLR_UD_UNIT_... (86) START
BLK_SLR_UD_U... (87) BLOCK
...

[Scenario]
14 - DV_SLR

id	14
type	DV_SLR
name	DV_SLR
owner	hnjang
avail	OPEN
shared_cnt	0
create_date	2019-04-27 14:19:06
golden	A
env	/user/socat/AntsDVA/CUSTOM/ENV/atdb.e...
cmd	
blocking	0
start_time	
end_time	
time_data_type_id	0
rel_sc	

[Step Job]
84 - SLR_UD_RUN

id	84
name	SLR_UD_RUN
command	/user/socat/AntsDVA/CUSTOM/step/AntsDVA_S...
envfile	
create_date	2019-09-20 09:00:25
blocking	1
log	/user/socat/.AntsDVA/log/{JOBID}_{ITEMID}.log
param_file	
lock_logic_id	- select lock_logic_id -
cmd_type_id	1 : SHELL_CMD
type_id	NORMAL
related_scenario	DV_SCR(22)

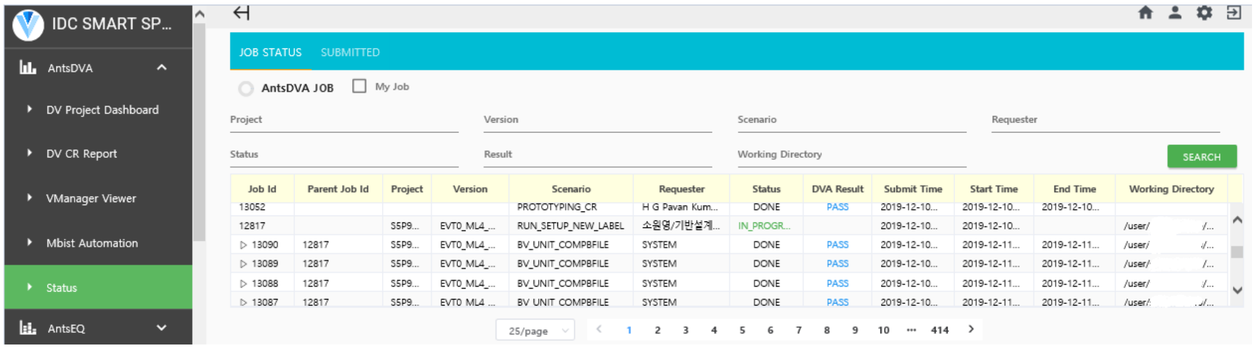
Figure 5: Verification automation scenario and step

2) Regression Manager

Regression Manager is system of regression test cases for managing and control and leverages internally developed queues and commercial regression tools. Currently, a thirty party tool used in Samsung has been integrated. In the IP stage, the regression environment is configured as a script, and information on this part is reported through meta data. In the IP-level design stage, the regression environment is configured by a script, and information on this is reported through meta data. A large amount of jobs provided by regression managers and information of their status and results can be checked at a glance because those information can be connected by data structures managed for each scenario.

3) Job Scheduler

Job Scheduler is to schedule jobs based on a priority or reserve the time to execute (reservation time). It is internally developed. In the job scheduler developed in this paper, the basic unit of job is composed of scenario. One job is submission to the job queue with one scenario. A job is performed when certain conditions are satisfied. The unit of execution of each job is a group of each entity called Actor and is parallel for load balancing of service group or daemon. Several daemons will perform a job coming into their actor group. All jobs are performed while the start-step and end-step of the scenario are continuously managed during the execution. Each job has the status information of scenario and the status information of each step and controls life time of job with the status information. Figure 6 shows the real time status of various types of scenario which is managed through the system proposed in this paper.



Job Id	Parent Job Id	Project	Version	Scenario	Requester	Status	DVA Result	Submit Time	Start Time	End Time	Working Directory
13052				PROTOTYPING_CR	H G Pavan Kum...	DONE	PASS	2019-12-10...	2019-12-10...	2019-12-10...	
12817		S5P9...	EVT0_ML4_...	RUN_SETUP_NEW_LABEL	소원영/기반설계...	IN_PROGR...		2019-12-10...	2019-12-10...		/user/ /...
▷ 13090	12817	S5P9...	EVT0_ML4_...	BV_UNIT_COMPBFILE	SYSTEM	DONE	PASS	2019-12-10...	2019-12-11...	2019-12-11...	/user/ /...
▷ 13089	12817	S5P9...	EVT0_ML4_...	BV_UNIT_COMPBFILE	SYSTEM	DONE	PASS	2019-12-10...	2019-12-11...	2019-12-11...	/user/ /...
▷ 13088	12817	S5P9...	EVT0_ML4_...	BV_UNIT_COMPBFILE	SYSTEM	DONE	PASS	2019-12-10...	2019-12-11...	2019-12-11...	/user/ /...
▷ 13087	12817	S5P9...	EVT0_ML4_...	BV_UNIT_COMPBFILE	SYSTEM	DONE	PASS	2019-12-10...	2019-12-11...	2019-12-11...	/user/ /...

Figure 6: Job status managed through automation system

4) Triage Scenario

Triage Scenario (TS) module is to classify issues or failures based on data generated from Regression. For example, if a bug or a failure occurs while any jobs are executed, TS have several algorithms to narrow down a region or target range of possible issues dynamically. It also re-runs if necessary to provide sufficient information for designers or verification engineers for rapid debugging. In the current version, binary search algorithm and machine learning based clustering are implemented. Our proposed system collects many regression test results and creates clustering data for each blocks by density-based clustering technique which can re-ordering the bunch of test cases for efficient verification. It has a test schedule to re-order the test list for error cases per cluster. The reasons to why we are applying those solutions are two things. Firstly, when design data changes; it makes too many tests to perform every particular period of time and resources inefficiently. Secondly, since the test cases for verification of each change block are connected to different blocks and affect each other, the error case that occurs is not only a failure of the test case but also a potential error element. If you connect it to a strategy to verify it in advance, you can expect to improve efficiency and verification speed.

5) Data Manager

Data manager collects all data while every job execution and uses them for further analysis. For example, all design changes (especially RTL changes) are collected over the SOC development cycle and are used to correlate with any simulation failures to assist verification engineers to debug those failures. The data manager implemented in this paper can integrate the verification information for each IP, block. SoC verification stage and interwork with the verification information for each stage. A verification can be efficient through interworking of verification information. The information that is integrated in the data manager is the verification task for each verification step and the result information for each verification operation. These data are formatted and stored in a database, and user input information and log information for each step are stored in a disk in the form of a file. The meta information for the abnormal data is formulated and managed. The formalized meta information is summarized using mutual relationship. The verification information accumulated in this way can be used in different verification stages, and the linked information leads to the high efficiency of verification. The verification status of each stage is transferred to the verification manager through the dashboard system or mail alarming service. Figure 7 shows the dashboard system and e-mail reporting service.

Figure 7: Dashboard and Mail service

In addition, the ongoing research is informatizing the amount of change in the design information for each stage of development. It is expected to shorten the verification period and increase the verification quality by linking this to verification information.

Experimental Results: The integrated platform is applied to our design verification to automate the regression. The experimental results is shown as below

Until now, verification work for each project has been classified as a role of each level's verification engineer. In other words, the work was independent of each other. Verification for each IP and block level was repeated or added every time the development version was changed, and it was being progressed separately because it was not synchronized with the top level verification. Automation and information integration of each verification phase can optimize the verification flow and test for each event. In order to calculate the effect, a number of verification efficiency generated by each verification step was written and viewed. It, Of course, will be different for each IP, Block, and SOC product, but the verification time according to the complexity has been quantified by referring to the results of the actual project.

Table 1 shows experimental numbers of comparative results of verification time between existed method and the proposed platform. A, B, and C mean the verification time for each verification target. A1 is the verification time in the existed IP-level verification. The verification time of the proposed platform can be defined as the maximum value of A1 or B1 or verification time used for failed cases in IP verification or verification time used for failed cases in block verification. At the VL1, A1 is defined as 4T and B1 is defined as 16T, the existing verification time becomes 20T, and the proposed platform application result could be 16T when fail cases are not happening in IP or Block level. At the VL2, all of levels verification test should be conducted, so the verification time could be reduced to max run time among the A2, B2 and C2 test running time compared to the previous sum of each level. At the VL3 and VL4, verification time is reduced by max run time of each level. Analyzing the time for failure occurrence as statistical probability is not clear from differences in distribution by verification level. Thus, It will be interpreted as an approximation through actual application.

Table 1: Verification time by each verification step

Verification Level / Change events	VL 1		VL 2		VL 3		VL 4	
	Before	After	Before	After	Before	After	Before	After
IP level change	A_1	$\text{Max}(A_1,B_1)$ OR $A_{1\_IP\_fail}$ OR $B_{1\_Block\_fail}$	$A_2+B_2+C_2$	$\text{Max}(A_2,B_2,C_2)$ OR $A_{2\_IP\_fail}$ OR $B_{2\_Block\_fail}$				
Block level change	B_1	B_1	B_2+C_2	$\text{Max}(B_2,C_2)$ OR $B_{2\_Block\_fail}$	B_3+C_3	$\text{Max}(B_3,C_3)$ OR $B_{3\_Block\_fail}$	B_4+C_4	$\text{Max}(B_4,C_4)$ OR $B_{4\_Block\_fail}$
SOC level change			C_2	C_2	C_3	C_3	C_4	C_4

* A_N = 4T, 6T, 0, 0 (N : number of verification level, T: unit time), B_N = 16T, 20T, 24T, 20T, C_N=0, 60T, 90T, 120T

Figure 8 shows the results applied to the actual project. This result shows efficiency for the verification time before and after each verification step. For reference, it has not been applied to the overall verification of the actual project, has been applied to some IPs and corresponding blocks, and in the top level of SOC, has all been applied.
At the VL1, as mentioned above, verification time is reduced by around 35% and the verification time is reduced by 43% at the VL2. At the VL3 and VL4, verification time is reduced by 28% and 27% separately. The verification time reduction in VL3 and VL4 is less than that in VL1 and VL2 because it does not includes the time reduction effect of IP-level verification. The verification time for each step was decreased by up to 43%, when the triage flow in the integrated platform applied to test failure cases at each verification stage as Figure 8 shows.

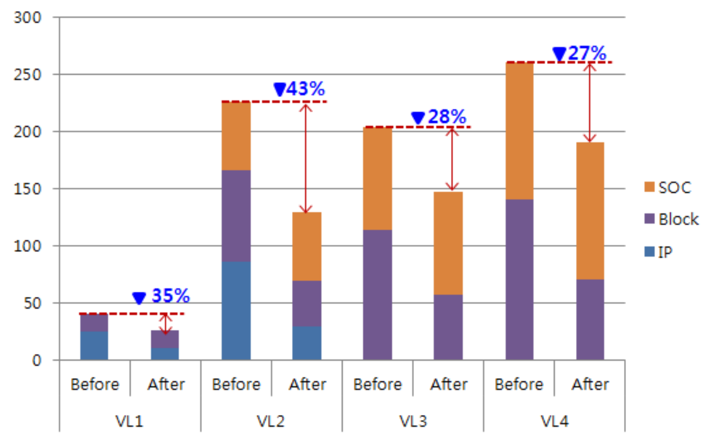


Figure 8: Reduction of verification time by each verification step

Conclusion: In order to provide verification environment, which is optimized for complex SOC, each verification workflow process is being automated for each target design level. Furthermore, verification time can decrease and verification quality can increase as these processes are integrated into automation environment

The main idea is to integrate existing workflows and verification environments so that they can be reused in the form of platforms. This integrated platform can enable verification environment setup rapidly and can unify scattered information of each verification level. Furthermore, the unified information can enable to optimize verification environment.

In this paper, a platform for automating and integrating workflow scattered by stage for each verification target was proposed to shorten the verification time. In the actual flag ship mobile AP SoC design project, many local workflows from IP verification to block and SoC verification were integrated into the proposed platform to verify the maximum 43% of the verification time per step. To this end, some of the existing un-automated workflows have been automated and verification data has been linked to the integrated system, but the integration of each department has not been proceeded. In addition to the part where each verification phase information is linked and efficient As the number of tests was reduced by automating the execution of the processes in Workflow and applying the proposed triage ratio to the cases where the verification status and information were integrated, the verification time could be greatly reduced. This reduced time has helped improve the quality of verification because it can be used for additional verification. Moreover, the proposed system has greatly reduced efforts to create an integrated platform because it can re-use the existing workflow, the verification environment, and the existing system. In conclusion, the verification information distributed and produced at each verification stage could be integrated into one view through the proposed data manager, and the integrated information could be analyzed and utilized to optimize the verification environment. These results resulted in reduced verification time and improved verification quality.

Through additional research, we plan to re-order the test list of error cases by clustering each block by density-based clustering using a lot of regression test result information for each verification phase. In addition, we would like to consider using the classification model to determine the group of sanity test for error cases. In addition, we expect that by additionally applying the dynamic distributed structure technique of job scheduling algorithm, the verification system will be improved to shorten the verification time.

References:

[1] A. MOLINA, O. CADENAS, "FUNCTIONAL VERIFICATION: APPROACHES AND CHALLENGES", Computer Architecture Department, Universitat Politècnica de Catalunya, Barcelona, Spain,2007

[2] Design and Verification Gaps information is at https://www.researchgate.net/figure/Design-and-Verification-Gaps-Design-productivity-growth-continues-to-remain-lower-than_fig3_237116903

[3] continuous integration constraint information is at https://www.ionos.com/digitalguide/websites/web-development/continuous-integration/

[4] Fine, S. and A. Ziv, “Coverage directed test generation for functional verification using bayesian net-works,” Proc. 40th Design Automation Conf. (DAC), Anaheim, CA, USA, 286-291 (2003).

[5] G. Rothermel, R.H. Untch, Chengyun Chu, M.J. Harrold, "Prioritizing test cases for regression testing", IEEE Transactions on Software Engineering (Volume: 27 , Issue: 10 , Oct 2001)

[6] Validation of deep-learning-based triage and acuity score using a large national dataset is at https://doi.org/10.1371/journal.pone.0205836 (2018)

[7] Zissis Poulos , Student Member, IEEE, and Andreas Veneris, Senior Member, IEEE, "Failure Triage in RTL Regression Verification", IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, VOL. 37, NO. 9, SEPTEMBER 2018 1893