



Efficient Regression Management with Smart Data Mining Technique

Tejbal Prasad
Cadence Design Systems

Abstract- With the increasing complexity of design, every design has multiple modes and features. Efficient regression management is the key to project execution & success. The verification team's biggest effort goes into making sure that the existing status remains intact while making progress on the new features. On top of that at any point in time, there will be known failures such as TB issues, known design & VIP bugs, etc. Filtering these known failures to focus on the real issues is a cumbersome job, often done manually. Knowing the holistic status of the project is a very challenging task across different modes & features on a day-to-day basis. One needs an automated way to replicate top category unique failures after filtering known issues from the regression run on the cloud to local server. This paper talks about how smart data mining is done over the vManager regression data with Python/Jenkins script is helping us overcome these problems. The first part talks about how changes done in VSIF are solving the daily regression triage problem. The second part talks about how a python script does data mining and filters out all the known failures in an efficient manner. The third part talks about how we are making use of the data mining script and its extensions to auto-rerun top category failures. It shows the author's experience and its deployment in a multimillion gate IP to successfully converge on the verification signoff.

Keywords— IP, SoC, Regression, vManager, Jenkins

I. INTRODUCTION

Efficient regression management is the key to project execution & success in today's era. As the design's complexity is growing every year, verification complexity is growing exponentially. Verification team need to stress the design with positive, negative & error scenarios while testing all the features and modes of the design.

In regression management following are main problems.

- **Holistic Status of Project:** Design undergoes multiple fixes every day and we don't know how those fixes are going to impact other feature and modes of the design. And knowing the holistic status of project with respect to different features across each mode is a challenging task.
- **Known Issue Tracking:** To focus on the new failures, one needs to filter out the existing known issues. Management of known issues/failures and filtering it out from regression status is another cumbersome task.
- **Auto Failure Re-Creation:** After filtering out the known issues, replication of failures with additional debug info is a manual job, which should be automated to speed up the verification process.

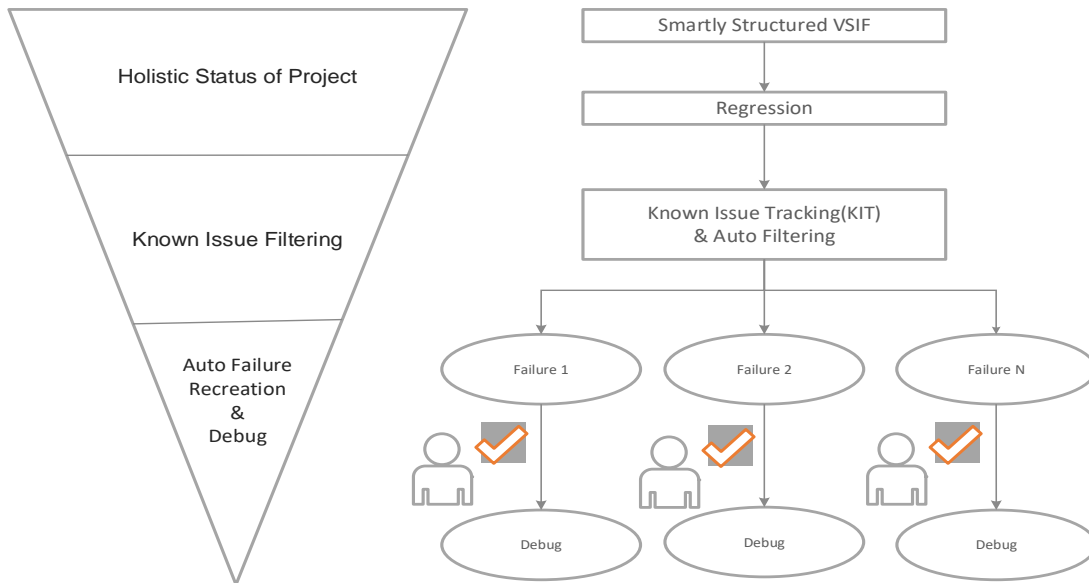


Figure 1: Proposed Solution

This paper reflects our project experience where we solved these problems and automated the flow using vManager, Python & Jenkins scripts. Figure 1 shows the proposed solution.

II. HOLISTIC STATUS OF PROJECT

Various features of the design and presence of various modes of the design where all features of the design should be exercised makes the status tracking process cumbersome. Often there's a confusion on whether some feature “**Fi**” is verified in one of the modes “**Mj**” or not and what is its status in current regression. Normally to get this information we depend on functional coverage database, or we rely on the expertise of the individual engineers working on it. While the project is in development phase where only some features/modes are enabled, getting the latest and greatest information from coverage database is not practical as coverage enabled database takes a longer run time compared to non-coverage enabled runs and also it requires lot of area for dumping coverage information.

This paper talks about how smart structuring was done in regression setup to overcome this problem. This enables us to extract the overall health of the design for each feature in every mode just by looking at the regression result.

We used the features provided by vManager to efficiently resolve this. In vManager regression flow we use VSIF which has session and can have one or multiple groups. Each group contain one or multiple tests.

We came up with a strategy of using above feature in a manner that we get a cross of feature with mode. We made these changes in the environment.

- i. **Random Test:** Each test targets a feature (“Fi”) of the design and is agnostic of the Mode of the design. In vManager we have Test Args and Test Name which can be different for different test. Using this we track each feature of the design.
- ii. **Random Test List:** All the random Test are added in the Random Test List. This ensures that when we use this random test list all the features are getting verified.
- iii. **Regression:** In regression inside session, we have multiple groups. Same Random Test List is called inside each group. Each group corresponds to a Mode (“Mj”) of the design. We pass high level control knob which is different in each group. In vManager we have group_args which is used for passing this control knob.

After doing all above changes we get the regression result where we get the per mode regression view as well as per feature view inside every mode. When a feature is failing in all the modes would indicate that this feature needs verification attention. Whereas when pass percentage of certain mode is down that implies that this specific mode of the design is broken. Now just the regression status across different modes and features gives the holistic status of the design so we don’t really need a coverage enabled regression on day-to-day basis.

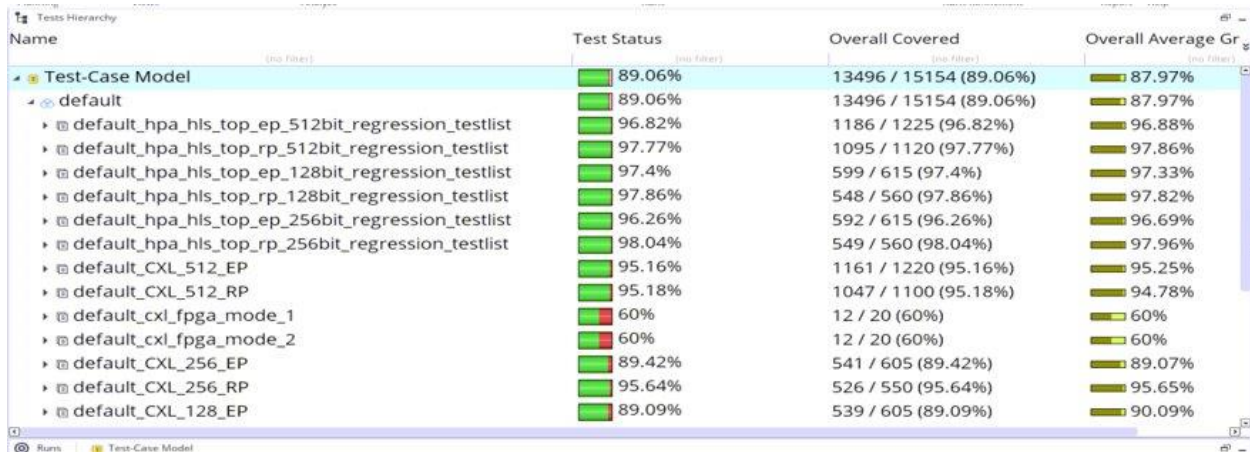


Figure 2: Snippet of regression result across modes

As shown is Figure 2 each group represent one of the major modes (Mj) of the design. So here just by looking at the regression result one can figure out which all modes require verification attention. Like in the current case “cxl_fpga_mode_1” & “cxl_fpga_mode_2” need more attention, as pass percentage in these modes are less compared to other modes. Similarly, we get a view of all the modes.

Name	Test Status	Overall Covered	Overall Average Gr
Test-Case Model	89.06%	13496 / 15154 (89.06%)	87.97%
default	89.06%	13496 / 15154 (89.06%)	87.97%
default_hpa_hls_top_ep_512bit_regression_testlist	96.82%	1186 / 1225 (96.82%)	96.88%
cor_ecc_test	95%	19 / 20 (95%)	95%
uncor_ecc_test	100%	20 / 20 (100%)	100%
base_test	100%	20 / 20 (100%)	100%
sanity_traffic_test	100%	20 / 20 (100%)	100%
all_traffic_base_test_medium	96%	96 / 100 (96%)	96%
all_traffic_base_test_small	95%	19 / 20 (95%)	95%
alltraffic_link_flr_test_wo_traffic	95%	19 / 20 (95%)	95%
alltraffic_lm_flr_test_wo_traffic	100%	20 / 20 (100%)	100%
alltraffic_linkdown_test_wo_traffic	80%	16 / 20 (80%)	80%
alltraffic_linkdown_test_with_traffic	100%	20 / 20 (100%)	100%
hard_reset_test	100%	20 / 20 (100%)	100%
alltraffic_max_pfvf_mode1_test	90%	18 / 20 (90%)	90%

Figure 3: Snippet of regression result across function in each mode

Figure 3 shows per feature (“Fi”) view where each test represents a design feature. Same Test List is called inside each group so, which means every feature is getting tested in each mode. When same test is failing in multiple modes which implies that this feature requires verification attention.

With this anyone can get to know the holistic status of the design just by looking at regression status. We have achieved this using the vManager tool and a similar setup can be done in any other regression management tool.

III. SMART DATA MINING: KNOWN ISSUE TRACKER(KIT)

At any point of the verification cycle there will be known failures in regression. The known failures could be because of incomplete design feature/TB checker, known Design/VIP bugs etc. Efficient management of these failures and filtering out these failures from the current regression to focus on the new/unknown failures is very important.

Usually, these failures are managed **manually**, which is **cumbersome, error prone** and **time-consuming process**.

We solved this problem using a smart data mining technique by developing Python and Jenkin script (Known Issue Tracker/KIT script)

- It is tightly integrated with vManger
- It filters out the known failures from the regression result
- It is called after regression run in Jenkins setup
- Automatically it populates all the information after regression is over

It eliminates all the manual intervention that is usually required and gives us the new failures signature sorted with the order failure impact.

Structured Known issue list: As we are taking about know issue tracking so we will have to maintain a known issue list. To support automation, we defined a structure so that scripts can parse it. We also added features so that we will not have to enter the same signature again and again, while having a provision to capture the signatures which are applicable for a specific feature and/or mode.



	TEST_NAME	GRP_NAME	TEST_STATUSES	RUN_DIR	FIRST_FAILURE_DESC	ANALYSIS
1	alltraffic_link_flr_test_wo_traffic	ALL	FAIL	run_3686	inconsistent_internal_data_structure_in_pcie_cfg.cpp_24626_pc	VIPSR_5544331
2	alltraffic_linkdown_test_with_traffic	ALL	FAIL	run_3692	Tried_to_decrement_the_passive_credit_count_on_channel	VIPSR_1200001
3	ALL	ALL	FAIL	run_2	TX_FATAL_MalformedTlp_TL_TLP_MF_INVALID_TAGSCAL	JIRA_CXL1234
4	ALL	cxl	FAIL	run_125	TX_NONFATAL_UnsupportedRequest_TL_TLP_USERTAG_2__	VIPSR_46532491
5	loopback	ALL	FAIL	run_286	cfg_0_0__TL_CFG_UNKQID_4__	TB_ISSUE_LOOPB
6	ALL	ALL	FAIL	run_1945	PL_SRIS_SKP_TX_NONE__	VIPSR_11

Table 1: Structured Known issue format

Table 1 shows Structured Known issue format. We have these columns in this CSV file:

- 1st column is just an index.
- 2nd column is Test Name (TEST_NAME): Name of the test that is expected to fail due to a known reason.
- 3rd column is the Group Name (GRP_NAME): Name of the Group where test is expected to fail due to a known reason
- 4th column is run_dir, this is ignored by this script. These are used in Failure Auto replication, so we have kept them here.
- 5th column is the test status, this is also ignored by this script. These are again used in Failure Auto replication.
- 6th column is the First failure description (FIRST_FAILURE_DESC)
 - This is the 1st failure signature which will be looked while processing any new regression result.
- 7th column captures the analysis (ANALYSIS) that user has done against each failure signature.
- Every row corresponds to one known failure entry.
 - We capture each known failure signature with Test Name (TEST_NAME), Group Name (GRP_NAME) and First Failure Signature Description (FIRST_FAILURE_DESC).



	TEST_NAME	GRP_NAME
1	alltraffic_link_flr_test_wo_traffic	ALL
2	alltraffic_linkdown_test_with_traffic	ALL
3	ALL	ALL
4	ALL	exl
5	loopback	ALL
6	ALL	ALL

Table 2: Handling Generic Failure

Handling Generic failures: While capturing the known failure signatures we have added provisions for handling generic failures. This helps in reducing the number of failure signatures that we must maintain.

- If a failure is independent of the test name, in that case in place of the Test Name we use “ALL”
- Similarly, if a signature is independent of the group name, in place of the Group Name we again use “ALL”
- Partial Substring Matching is enabled in Test Name, Group Name and First Failure description.
 - In place of test name “pl_loopback_test_wo_traffic” and “pl_loopback_with_traffic” we can use “loopback”
 - In place of “cfg_0_0___PL_SRIS_SKP_TX_NONE___” and “cfg_1_2___PL_SRIS_SKP_TX_NONE___” we can use “PL_SRIS_SKP_TX_NONE” in FIRST_FAILURE_DESC

Known Issue Tracker (KIT) Script: It’s a python script that auto populate the known failure after vManager regression is over. These are the inputs needed to run KIT script.

- Existing Failure Analysis Table in CSV format like it was shown in Table 1.
- Session Directory Path
- Output Directory Path (Optional)

Steps done by the KIT script.

- It parses the entire regression and captures the status of the regression in CSV file, where it captures Test Name, Group Name, Test Status & First Failure signature for each run in the regression.
- Then for each failing test it checks the First Failure Signature with the existing failure table. It will match the Test Name, Group Name and First Failure Signature. All the Generic Failure matching that we described are checked here while matching the error signature.
- If a Match is found it updates Analysis content from the existing Failure Table.
- If no match is found in the existing table, then it will be updated as “NEW_FAILURE”.
- As a result, it dumps out the failure signature table in different flavor after filtering and populating the known failures.

- All the results are sorted in the order of impact, the failure signature that is causing the maximum number of failures is at the top.

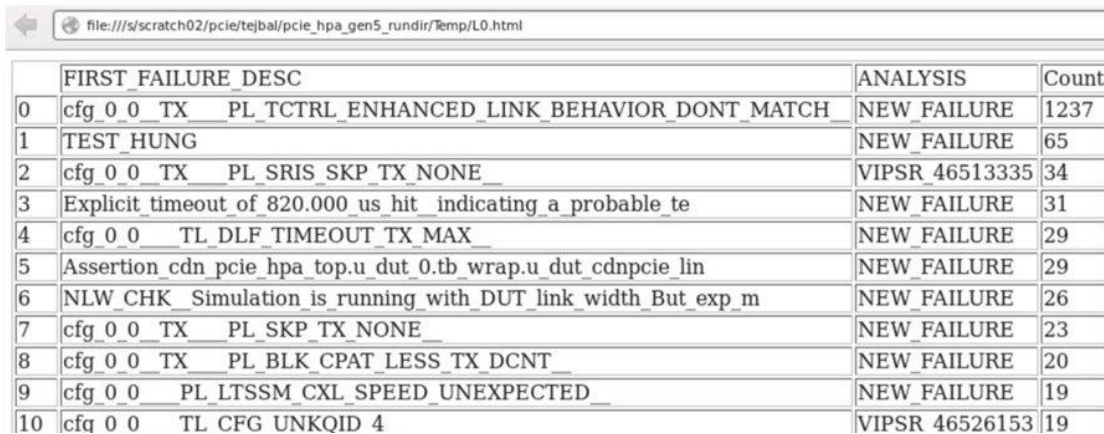
Output of the Script: As shown in figure 4, KIT generates report in multiple formats.

- *L0* is for Level 0 report sorted with failure count wrt just failure signature
- *L1* is for Level 1 report sorted with test name and failure signature
- *L2* is for Level 2 report sorted with test name, Group Name and failure signature
- Similarly, N0, N1 & N2 reports are for the same granularity after filtering the known failures.

```
ls /s/scratch02/pcie/tejbal/pcie_hpa_gen5_rundir/Temp
failure_run_detail.html  L1.html  N1.html          status_full.L0.csv  status_full.N0.csv
grp_test.csv            L2.html  N2.html          status_full.L1.csv  status_full.N1.csv
L0.html                N0.html  status_full.csv  status_full.L2.csv  status_full.N2.csv
```

Figure 4: KIT output reports

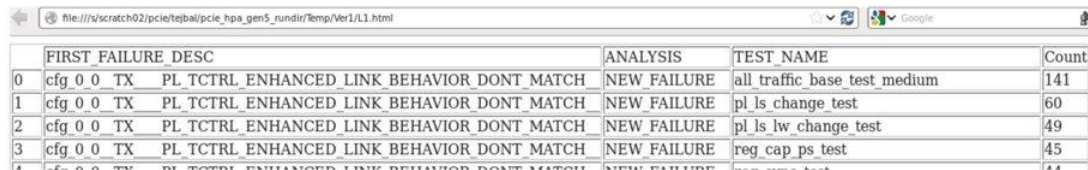
As shown in Figure 5, Level 0 (L0.html) report will contain the updated status with prefilled existing analysis & it will be grouped only wrt First Failure description. This gives an overall picture about which failure signature is causing major damage in the regression.



	FIRST_FAILURE_DESC	ANALYSIS	Count
0	cfg 0 0 TX PL TCTRL_ENHANCED_LINK_BEHAVIOR_DONT_MATCH	NEW FAILURE	1237
1	TEST HUNG	NEW FAILURE	65
2	cfg 0 0 TX PL SRIS_SKP_TX_NONE	VIPSR 46513335	34
3	Explicit timeout of 820.000 us hit indicating a probable te	NEW FAILURE	31
4	cfg 0 0 TL DLF_TIMEOUT_TX_MAX	NEW FAILURE	29
5	Assertion cdn pcie_hpa_top.u dut 0.tb wrap.u dut cdnpcie lin	NEW FAILURE	29
6	NLW_CHK Simulation is running with DUT link width But exp m	NEW FAILURE	26
7	cfg 0 0 TX PL SKP_TX_NONE	NEW FAILURE	23
8	cfg 0 0 TX PL BLK_CPAT_LESS_TX_DCNT	NEW FAILURE	20
9	cfg 0 0 PL LTSSM_CXL_SPEED_UNEXPECTED	NEW FAILURE	19
10	cfg 0 0 TL CFG_UNKQID 4	VIPSR 46526153	19

Figure 5: Level 0 Combined view

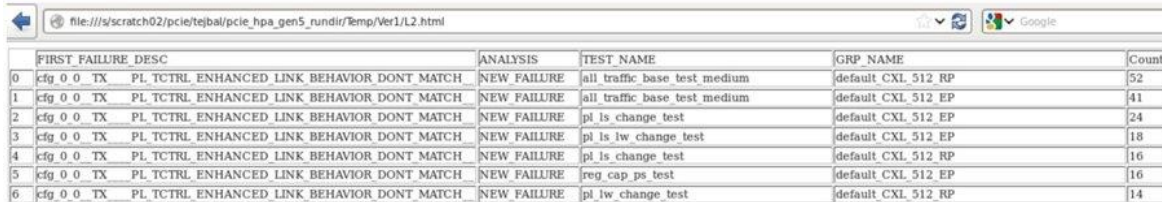
As shown in Figure 6, Level 1 (L1.html) report – Adds Test Name while generating & sorting the report. At times knowing the test name that is mostly failing gives a better picture of the problem. This report enables that.



	FIRST_FAILURE_DESC	ANALYSIS	TEST_NAME	Count
0	cfg 0 0 TX PL TCTRL_ENHANCED_LINK_BEHAVIOR_DONT_MATCH	NEW FAILURE	all_traffic_base_test_medium	141
1	cfg 0 0 TX PL TCTRL_ENHANCED_LINK_BEHAVIOR_DONT_MATCH	NEW FAILURE	pl_ls_change_test	60
2	cfg 0 0 TX PL TCTRL_ENHANCED_LINK_BEHAVIOR_DONT_MATCH	NEW FAILURE	pl_ls_lw_change_test	49
3	cfg 0 0 TX PL TCTRL_ENHANCED_LINK_BEHAVIOR_DONT_MATCH	NEW FAILURE	reg_cap_ps_test	45
4	cfg 0 0 TX PL TCTRL_ENHANCED_LINK_BEHAVIOR_DONT_MATCH	NEW FAILURE	...	44

Figure 6: Level 1 Combined view

As shown in Figure 7, Level 2 (L2.html) report – Adds Test Name & Group Name while generating & sorting the report. At times while fixing bug for one mode causes failures in other mode. So having the group information helps to analyze the report from this angle.



	FIRST_FAILURE_DESC	ANALYSIS	TEST_NAME	GRP_NAME	Count
0	cfg_0_0_TX_PL_TCTRL_ENHANCED_LINK_BEHAVIOR_DONT_MATCH	NEW_FAILURE	all traffic base test medium	default CXL 512 RP	52
1	cfg_0_0_TX_PL_TCTRL_ENHANCED_LINK_BEHAVIOR_DONT_MATCH	NEW_FAILURE	all traffic base test medium	default CXL 512 EP	41
2	cfg_0_0_TX_PL_TCTRL_ENHANCED_LINK_BEHAVIOR_DONT_MATCH	NEW_FAILURE	pl_ls_change_test	default CXL 512 EP	24
3	cfg_0_0_TX_PL_TCTRL_ENHANCED_LINK_BEHAVIOR_DONT_MATCH	NEW_FAILURE	pl_ls_lw_change_test	default CXL 512 EP	18
4	cfg_0_0_TX_PL_TCTRL_ENHANCED_LINK_BEHAVIOR_DONT_MATCH	NEW_FAILURE	pl_ls_change_test	default CXL 512 RP	16
5	cfg_0_0_TX_PL_TCTRL_ENHANCED_LINK_BEHAVIOR_DONT_MATCH	NEW_FAILURE	reg_cap_ps_test	default CXL 512 EP	16
6	cfg_0_0_TX_PL_TCTRL_ENHANCED_LINK_BEHAVIOR_DONT_MATCH	NEW_FAILURE	pl_lw_change_test	default CXL 512 RP	14

Figure 7: Level 2 Combined view

As shown in Figure 8, Level 0 (N0.html) report – This report is the filtered version of the L0 report. So, all the known issues are filtered out, and only new failures are kept in this report sorted with respect failure impact.

	FIRST_FAILURE_DESC	ANALYSIS	Count
0	Assertion_cdn_pcie_hpa_top.u_dut_0.tb_wrap.u_dut_cdnpcie_lin	NEW_FAILURE	1069
1	TEST_HUNG	NEW_FAILURE	258
2	cfg_0_0_RX_PL_SYNC_LN_GAP_COMPLIANCE__	NEW_FAILURE	2
3	RX_L0S_DUT_moved_to_Recovery_from_RX_L0s_not_within_NFTS_ti	NEW_FAILURE	2
4	np_p_Actual_from_exp_q_Bypass_happen_on_invalid_conditio	NEW_FAILURE	1
5	exp_np_q_is_not_empty_size_2_	NEW_FAILURE	1
6	cfg_0_4_TX_NONFATAL_UnsupportedRequest_TL_TLP_USERTAG_2__	NEW_FAILURE	1
7	cfg_0_0_TX_NONFATAL_UnsupportedRequest_TL_TLP_vlTxPmDPkg_	NEW_FAILURE	1
8	cfg_0_0_RX_NONFATAL_UnsupportedRequest_TL_TLP_vlRxPmDPkg_	NEW_FAILURE	1
9	cfg_0_0_RX_COR_Receiver_PL_FRAME_TLP_END__	NEW_FAILURE	1
10	_CDN_HPA_PCIE_TLP_CPL_INVALID_REQ_TAG_UNEXP_ERR_Detected_E	NEW_FAILURE	1

Figure 8: Level 0 Filtered view

As shown in Figure 9, Level 1 (N1.html) report – This report is the filtered version of the L1 report. So, all the known issues are filtered, and only new failures are kept in this report sorted with respect to most failing Failure Signature & Test Name.



	FIRST_FAILURE_DESC	ANALYSIS	TEST_NAME	Count
0	Assertion_cdn_pcie_hpa_top.u_dut_0.tb_wrap.u_dut_cdnpcie_lin	NEW FAILURE	all_traffic_base_test_medium	76
1	Assertion_cdn_pcie_hpa_top.u_dut_0.tb_wrap.u_dut_cdnpcie_lin	NEW FAILURE	reg_be_test	40
2	Assertion_cdn_pcie_hpa_top.u_dut_0.tb_wrap.u_dut_cdnpcie_lin	NEW FAILURE	reg_hw_reset_test	37
3	Assertion_cdn_pcie_hpa_top.u_dut_0.tb_wrap.u_dut_cdnpcie_lin	NEW FAILURE	reg_cap_ps_test_cfg	35
4	Assertion_cdn_pcie_hpa_top.u_dut_0.tb_wrap.u_dut_cdnpcie_lin	NEW FAILURE	reg_read_all_test	35
5	Assertion_cdn_pcie_hpa_top.u_dut_0.tb_wrap.u_dut_cdnpcie_lin	NEW FAILURE	reg_sync_test	35
6	Assertion_cdn_pcie_hpa_top.u_dut_0.tb_wrap.u_dut_cdnpcie_lin	NEW FAILURE	pl_ls_change_test	32
7	Assertion_cdn_pcie_hpa_top.u_dut_0.tb_wrap.u_dut_cdnpcie_lin	NEW FAILURE	pl_lw_change_test	32
8	Assertion_cdn_pcie_hpa_top.u_dut_0.tb_wrap.u_dut_cdnpcie_lin	NEW FAILURE	pl_ls_lw_change_test	30
9	Assertion_cdn_pcie_hpa_top.u_dut_0.tb_wrap.u_dut_cdnpcie_lin	NEW FAILURE	reg_wr_follow_rd	29
10	TEST_HUNG	NEW FAILURE	all_traffic_base_test_medium	27
11	Assertion_cdn_pcie_hpa_top.u_dut_0.tb_wrap.u_dut_cdnpcie_lin	NEW FAILURE	pl_ep_recovery_lock_eq_err_test	25
12	Assertion_cdn_pcie_hpa_top.u_dut_0.tb_wrap.u_dut_cdnpcie_lin	NEW FAILURE	alltraffic_max_pfvf_mode2_test	20
13	Assertion_cdn_pcie_hpa_top.u_dut_0.tb_wrap.u_dut_cdnpcie_lin	NEW FAILURE	alltraffic_max_pfvf_mode1_test	20
14	Assertion_cdn_pcie_hpa_top.u_dut_0.tb_wrap.u_dut_cdnpcie_lin	NEW FAILURE	alltraffic_max_pfvf_mode1_hard_reset_test	20
15	Assertion_cdn_pcie_hpa_top.u_dut_0.tb_wrap.u_dut_cdnpcie_lin	NEW FAILURE	pl_alltraffic_ioerror_framing_error_test	18
16	Assertion_cdn_pcie_hpa_top.u_dut_0.tb_wrap.u_dut_cdnpcie_lin	NEW FAILURE	pl_delay_sync_signal_test	18
17	Assertion_cdn_pcie_hpa_top.u_dut_0.tb_wrap.u_dut_cdnpcie_lin	NEW FAILURE	pl_nominal_empty_mode_test	18
18	Assertion_cdn_pcie_hpa_top.u_dut_0.tb_wrap.u_dut_cdnpcie_lin	NEW FAILURE	pl_rxst_err_signal_test	18
19	Assertion_cdn_pcie_hpa_top.u_dut_0.tb_wrap.u_dut_cdnpcie_lin	NEW FAILURE	alltraffic_vec_test	18
20	Assertion_cdn_pcie_hpa_top.u_dut_0.tb_wrap.u_dut_cdnpcie_lin	NEW FAILURE	reg_write_all	18
21	Assertion_cdn_pcie_hpa_top.u_dut_0.tb_wrap.u_dut_cdnpcie_lin	NEW FAILURE	alltraffic_lm_flr_test_wo_traffic	18

Figure 9: Level 1 Filtered view

As shown in Figure 10, Level 2 (N3.html) report – This report is the filtered version of the L2 report. So, all the known issues are filtered, and only new failures are kept in this report sorted with respect to most failing Failure Signature, Test Name & Group Name.

	FIRST_FAILURE_DESC	ANALYSIS	TEST_NAME	GRP_NAME	Count
0	Assertion_cdn_pcie_hpa_top.u_dut_0.tb_wrap.u_dut_cdnpcie_lin	NEW FAILURE	all_traffic_base_test_medium	default_hpa_hls_top_ep_512bit_regression_testlist	41
1	TEST_HUNG	NEW FAILURE	all_traffic_base_test_medium	default_CXL_256_RP	25
2	Assertion_cdn_pcie_hpa_top.u_dut_0.tb_wrap.u_dut_cdnpcie_lin	NEW FAILURE	all_traffic_base_test_medium	default_hpa_hls_top_ep_128bit_regression_testlist	23
3	Assertion_cdn_pcie_hpa_top.u_dut_0.tb_wrap.u_dut_cdnpcie_lin	NEW FAILURE	reg_be_test	default_hpa_hls_top_ep_512bit_regression_testlist	20
4	Assertion_cdn_pcie_hpa_top.u_dut_0.tb_wrap.u_dut_cdnpcie_lin	NEW FAILURE	reg_cap_ps_test_cfg	default_hpa_hls_top_ep_512bit_regression_testlist	20
5	Assertion_cdn_pcie_hpa_top.u_dut_0.tb_wrap.u_dut_cdnpcie_lin	NEW FAILURE	reg_hw_reset_test	default_hpa_hls_top_ep_512bit_regression_testlist	20
6	Assertion_cdn_pcie_hpa_top.u_dut_0.tb_wrap.u_dut_cdnpcie_lin	NEW FAILURE	reg_sync_test	default_hpa_hls_top_ep_512bit_regression_testlist	20
7	Assertion_cdn_pcie_hpa_top.u_dut_0.tb_wrap.u_dut_cdnpcie_lin	NEW FAILURE	reg_read_all_test	default_hpa_hls_top_ep_512bit_regression_testlist	20
8	Assertion_cdn_pcie_hpa_top.u_dut_0.tb_wrap.u_dut_cdnpcie_lin	NEW FAILURE	pl_lw_change_test	default_hpa_hls_top_ep_512bit_regression_testlist	19
9	Assertion_cdn_pcie_hpa_top.u_dut_0.tb_wrap.u_dut_cdnpcie_lin	NEW FAILURE	pl_ls_change_test	default_hpa_hls_top_ep_512bit_regression_testlist	18
10	Assertion_cdn_pcie_hpa_top.u_dut_0.tb_wrap.u_dut_cdnpcie_lin	NEW FAILURE	pl_ls_lw_change_test	default_hpa_hls_top_ep_512bit_regression_testlist	16
11	Assertion_cdn_pcie_hpa_top.u_dut_0.tb_wrap.u_dut_cdnpcie_lin	NEW FAILURE	all_traffic_base_test_medium	default_CXL_128_EP	11
12	Assertion_cdn_pcie_hpa_top.u_dut_0.tb_wrap.u_dut_cdnpcie_lin	NEW FAILURE	reg_be_test	default_hpa_hls_top_ep_128bit_regression_testlist	10
13	Assertion_cdn_pcie_hpa_top.u_dut_0.tb_wrap.u_dut_cdnpcie_lin	NEW FAILURE	reg_read_all_test_cfg	default_hpa_hls_top_ep_512bit_regression_testlist	10
14	Assertion_cdn_pcie_hpa_top.u_dut_0.tb_wrap.u_dut_cdnpcie_lin	NEW FAILURE	reg_read_all_test	default_hpa_hls_top_ep_128bit_regression_testlist	10

Figure 10: Level 2 Filtered view

IV. AUTO FAILURE RE-CREATION

A significant portion of the verification time goes in debug. Out of this debug time a good portion of the time is spent in picking the right failure which will have biggest impact on regression pass percentage & then replicating them with additional information's needed for debug. KIT script helps us in picking the right failures that will have the biggest impact. The next step is to replicate this in the standalone setup with waveform and additional debug information enabled. Often individual have their own local changes or they are not on the same version on which the regression is run. So, one need to ensure that he checks out the same version of databases and takes command line from regression database and would add the additional arguments to enable waveform dumping. On top that now a days the regressions are run on cloud where one might not have GUI access.



As a solution to this problem, we developed additional script to collect and copy below information from cloud to local server.

- Run command of each failing test
- NO report from KIT
- Design version information

At the local server side, we use the design information to get to the same design version and can pick top N (e.g. 10) failures from the NO report and recreate them with dump.

```
$: head status_full.NO.csv
,FIRST_FAILURE_DESC,ANALYSIS,Count
0,Assertion_cdn_pcie_hpa_top.u_dut_0.tb_wrap.u_dut_cdnpcie_lin,NEW_FAILURE,141
1,cfg_0_0_RX_NONFATAL_UnsupportedRequest__TL_TLP_vlRxPmDPkg_,NEW_FAILURE,5
2,Register__local_ep_model_h.PCIe_Link_0_local_ep_ip_cfg_ctrl_,NEW_FAILURE,4
3,cfg_0_0_TX_NONFATAL_UnsupportedRequest__TL_TLP_vlTxPmDPkg_,NEW_FAILURE,3
```

After running the script this will run one failure of each type by matching the failure signature.

```
$: ls AUTO_RERUN/* -d
AUTO_RERUN/run_1816.exp_p_q_is_not_empty__size_1_
AUTO_RERUN/run_3742.Register__local_ep_model_h.PCIe_Link_0_local_ep_ip_reg_bank_
AUTO_RERUN/run_5497.Assertion_cdn_pcie_hpa_top.u_dut_0.tb_wrap.u_dut_cdnpcie_lin
AUTO_RERUN/run_5613.cfg_0_0_RX_NONFATAL_UnsupportedRequest__TL_TLP_vlRxPmDPkg_
```

Run Command inside each auto rerun carries the information about the test name, group name & Actual regression log path

```
$: cat run_1816.exp_p_q_is_not_empty__size_1_/run_cmd
## Regression PATH
:/s/scratch02/pcie/pciedip/hpa2_link_top_nightly_rundir/results/vmanager/hpa2_link_top_nightly.pciedip.22_0
3_01_04_31_00_8748/chain_0/run_1816/local_log.log
## TEST NAME : pl_ptm_test
## GRP NAME : default_hpa_hls_top_ep_128bit_regression_testlist

make -f $HPAMF TLBYPASS=1 UVM_VERBOSITY=UVM_DEBUG TEST=cdn_pcie_cxl_base_test
VM_SEED_MODE=1 SEED=989410900 COV=0 CONFIG=hpa2_ga_config CUSTOMER_CONFIG=default
TB_MODE=DUT_VIP run DUMP=1 TRACE=1 RAND_CFG=1 WITH_PHY=0 DONT_GEN_TB=0
CFG_POLICY=cdn_pcie_strap_128bit_random_policy REGR_RUN_ARGS=" +ALL_TRAFFIC=1
+STRAP_IS_RP=0 "
```

With the addition of this Auto failure regression setup. Now in place of assigning failure signature, we directly point the waveform and log path of the failing scenario to the respective feature owners. This eliminates the failure replication hassle and thereby speeds up the verification process.



V. APPLICATION

This framework is useful for any complex IP/Sub-system/SoC verification. It organizes the verification process and automates the repetitive manual process very efficiently.

VI. RESULTS

We deployed this in PCIe/CXL verification environment. This being one of the most complicated IP with multiple features and multiple mode was an ideal candidate for deploying this framework. In fact, we developed the framework because we were not able to handle the regression status efficiently. There was a phase in the project where everyone was struggling to figure out what is causing major drop in regression. After deploying all these, things came in right order. Team was able to focus on the right feature set and modes of the design. Auto filtering of known failures and auto-replication of top failure signature helped the team close the verification on time.

VII. CONCLUSIONS

Smartly Structured VSIF:

- Organizes the regression framework for daily Triage.
- Avoids missing a feature across various modes of the design
- One can see the health of each feature across various modes

KIT Script:

- This filters out the noise in the regression result automatically.
- As the filtering is done by the script so it no more manual, time consuming or error prone.

Auto Failure Rerun:

- This eases out the chores for each verification engineer
- Now instead of assigning a failure signature, one will get dump & log of failing scenario
- Increases the focus & productivity of the team

VIII. REFERENCES

- [1] Shahid Ikram & Jim Ellis “*Dynamic Regression Suite Generation Using Coverage-Based Clustering*” DVCON 2017 [<https://dvcon-proceedings.org/wp-content/uploads/dynamic-regression-suite-generation-using-coverage-based-clustering-presentation.pdf>]
- [2] David Lacey & Ed Powell “*Creating the Optimal Regression Farm Infrastructure That Meets All Your Team’s Simulation Requirements*” DVCON 2018 [<https://dvcon-proceedings.org/wp-content/uploads/creating-the-optimal-regression-farm-infrastructure-that-meets-all-your-teams-simulation-requirements.pdf>]