Portable Stimulus vs Formal vs UVM
A Comparative Analysis of Verification Methodologies Throughout the Life of an IP Block

Gaurav Bhatnagar
Staff Engineer, Analog Devices, Inc

David Brownell
Manager, Analog Devices, Inc

Abstract- ***Verification techniques and methodologies continually evolve to tackle the increasing verification challenge. The state of the art in the industry today are UVM based and Formal based verification flows. Both techniques have proven to significantly improve verification quality but have weaknesses in that the testcases or stimulus are not 'reusable'.***

Portability of test cases has long been a goal for verification teams across industry. No one wants to reinvent the wheel by having to rewrite the same tests for different testing environments. The new Portable Stimulus standard solves this problem by writing the test intention once, and then re-using the test intent to create tests for different target testing applications.

The language devised for this standard is an extension of C++. The Portable Stimulus models can be used with UVM based Verification environments, C/C++ based SoC level environments, Pre-Si and Post-Si Validation environments. This paper seeks to find the answer by comparing the verification process of an AHB to APB Gasket IP, using UVM, Portable Stimulus and Formal Verification techniques, across the life of an IP block from block level DV, System DV, and board validation, and pros and cons of each methodology at each stage.

I. Introduction

The Functional Verification Process has evolved from manual waveform inspection to today's industry standard verification methodologies such as UVM and Formal Verification flows that are widely used and industry progenies. Portable Stimulus (PS) is a new industry standard created to allow the specification of test intent and behavior such that the intent can be re-used across any target platform, and this standard is promising to be the next big thing in Design Verification.

UVM, Formal, and PS all claim to be comprehensive verification techniques but require different tools and execution techniques so it is important to take this into consideration to determine the best suited verification technique for a given design to be verified. This paper explores this question with a comparative analysis of all these techniques for an IP block throughout its life cycle from initial development, system verification, and post silicon validation.

II. Details on Verification Methodologies

The aim of every Verification Methodology is to define an approach which ensures high quality verification. All the verification techniques start with a basic Verification Plan and this plan is executed in different ways. At some point, the execution is evaluated with the help of coverage and pass/fail metrics based on which the plan is revisited and executed repeatedly until it reaches an agreed level of completeness. The same is true with the UVM, Formal and Portable Stimulus based techniques. Despite this initial similarity, all the three approaches are very different when looking at other factors like planning infrastructure, environment bring up time, ease of integration, extent of reuse, test generation effectiveness, ease of analyzing reports, ease in closing the uncovered coverage goals and finally, portability.

Portability of the verification environments and tests is a big factor in use of verification methodologies. The Fig.1 (below) shows the various stages in the life of an IP block starting from block design to the Post Silicon process along with the widely used methodologies and the amount of reuse as we transition to different stages. At the block level verification, all the methodologies UVM, PS and Formal are being used extensively, depending on the kind of design. As we transition to SoC based verification, all the three methodologies are still used but a complete reuse is 'possible' only in the case of PS. The UVM based environment components are partially reusable while most of the dynamic simulation is done using C based tests. Formal verification also allows the block level assertions to be

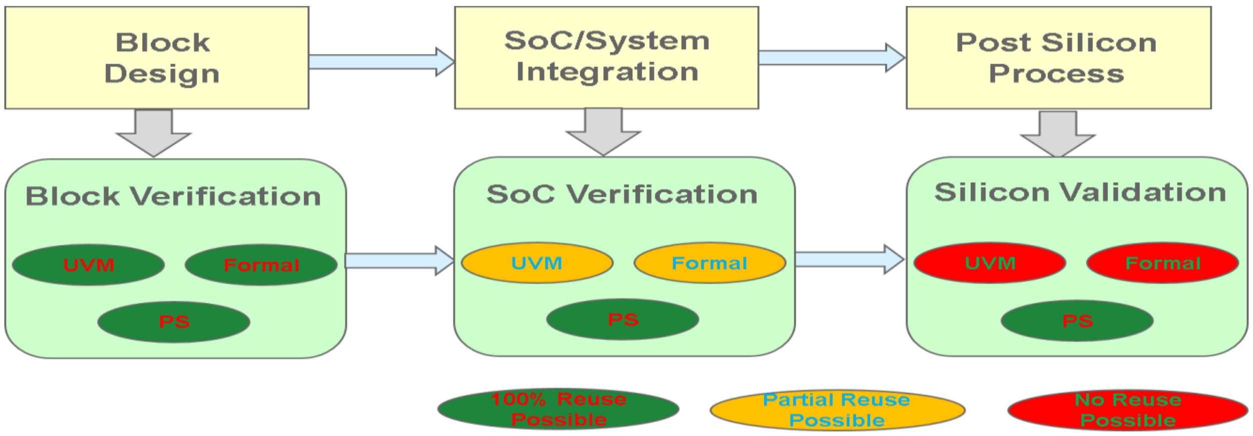


Figure1: Verification Methodologies throughout the life of an IP block

reused but the tool performance governs the reuse at the SoC level. PS based verification on the other hand allows test reuse by generating C-based tests. When we move to Post Si process, the UVM and Formal based verification are not used while it is still possible to use the PS based verification. We will analyze all these factors with the help of an AHB to APB Gasket usecase using the UVM, Formal and Portable Stimulus based verification. The AHB2APB gasket is a legacy IP block pre-verified using older verification techniques with system specific glue logic. It is re-verified so that it can be used with latest products using the latest under-given verification techniques.

A: UVM based verification of AHB2APB Gasket

The Verification with UVM is a pretty standard process. It starts with the creation of Verification Plan as per design specification and the setting up of the verification environment with the standard UVM components; agents, scoreboards, configuration and environments, all of them assembled such that they are reusable if we decide to change the scope of verification.

Directed and random tests are written on top of the environment with the help of Virtual Sequences. The Functional Coverage Points are created based on the verification plan. After this, the simulations are run and coverage database is created to collect Code and Functional Coverage. The Coverage holes are analyzed and depending on their severity, they are either waived or the Verification Plan is modified. The process is repeated until we meet the desired Coverage goals ensuring quality verification. The Fig .2 (below) represents this process with a flow diagram.

Along with the directed tests as a part of Verification Plan, this technique relies upon the random tests to achieve the coverage goals. It starts with random stimulus and gradually tightens the constraints until coverage goals are met, relying on the power of randomization and compute server farms to cover the state space.

While the code coverage is the quantitative measure, the functional coverage is the qualitative measure of the DUT code execution. Typically, this quality is limited by the diligence and thoroughness of the humans who draw up the verification plan and analyze the coverage reports.

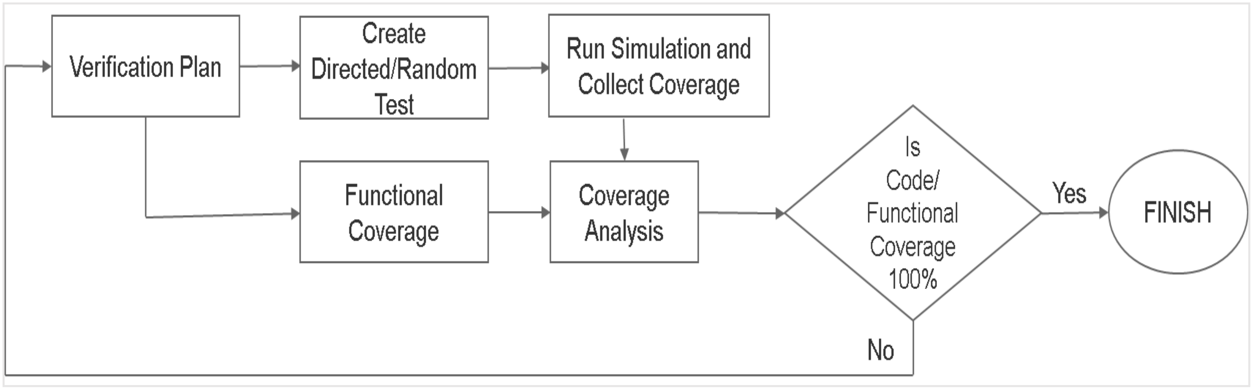

Verification Plan
Create Directed/Random Test
Run Simulation and Collect Coverage
Functional Coverage
Coverage Analysis
Is Code/ Functional Coverage 100%
Yes
FINISH
No

Figure2: Verification Flow with UVM

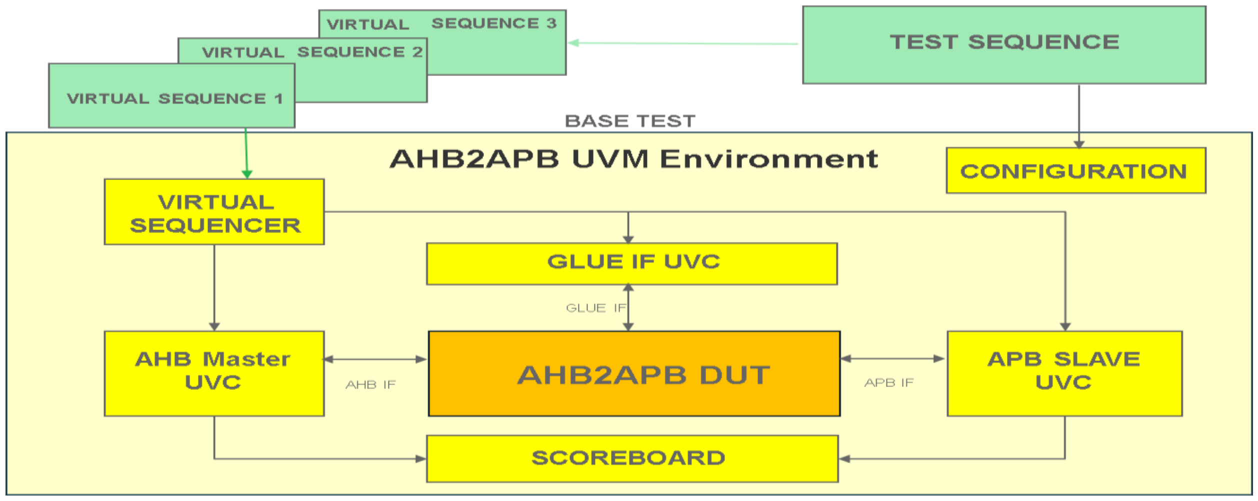


Figure3: AHB2APB Gasket UVM based environment

The other factor which decides the quality verification is the effective automated checking. A combination of packet comparison using the scoreboards and assertions based check points decide the number of Post-Silicon bugs discovered later in the flow.

This can be better understood by looking at the verification of AHB to APB Gasket using the UVM based environment. The Fig .3 (above), shows the standard UVM based verification environment setup for the AHB2APB gasket. It consists of AHB Master UVC, APB Slave UVC and Glue Interface UVC to drive sideband signals required for supporting logic. The UVM testcase consists of the test intent and controls the sequences of the VIP with Virtual Sequences. The tests are run in accordance with the UVM test plan with the directed and random test cases. The functional coverage and the code coverage are used as a sign-off criterion for verification. The regressions are run and reports are generated and analyzed.

The Table1 (below) shows the time taken (in weeks) to develop the UVM based environment. The initial setup to bring up the UVM environment integrating all the components took a week. The directed testcase development took 2 weeks while developing and debugging random test took an additional week. After this, the coverage analysis started and it took additional 2 weeks to close the coverage to an acceptable level. The table also shows the results collected from the regression run on AHB2APB gasket. Overall 5 directed and 100 random testcase run totaling to 105 runs with all of them passing. The column on the right indicates the overall coverage obtained without exclusions. The uncovered logic is either non-reachable code with disabled functionality or waived with designer consent making this coverage practically 100%. As mentioned earlier, the design was pre-verified, so the UVM based verification with the given testplan was not able to find any new bugs in the existing design apart from reconfirming the known issues or limitations. But, it does provides an effective way to check the fixes and allows reference for any new future design changes.

TABLE I
AHB2APB UVM SETUP AND REGRESSION

Initial Setup	**Directed Tests**	**Random Test**	**Coverage Closure**	**Overall Development Time**
1w	2w	1w	2w	6w
Tests Run	**Passed**	**Failed**	**Not Run**	**Overall Code Coverage**
105	105	0	0	2634/2864

B: Formal verification of AHB2APB Gasket

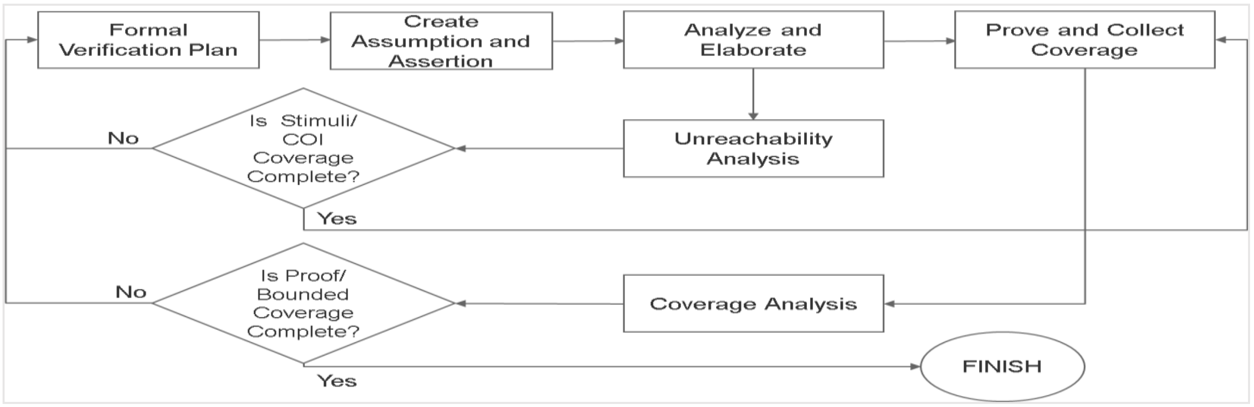

Formal Verification Plan
Create Assumption and Assertion
Analyze and Elaborate
Prove and Collect Coverage
Is Stimuli/ COI Coverage Complete?
No
Yes
Unreachability Analysis
Is Proof/ Bounded Coverage Complete?
No
Yes
Coverage Analysis
FINISH

Figure4: Verification Flow with Formal

Formal Verification is a method to mathematically prove that a design behaves as desired for all possible operating states. The Fig .4 (above) shows the Verification Flow with Formal Verification technique. The Formal Verification process starts with a Verification Plan which describes a set of Assumptions and Assertions required to operate the design. This plan is derived from the design specifications in more literal sense as compared to the other Verification Methodologies. Once we have list of assumptions and assertions listed, they are coded in SVA and then they are compiled along with the design by the Formal Verification tools.

At this stage the Unreachability analysis can be done, which is finding unreachable in the RTL due to coding limitations or the way stimuli is applied to the design under the given set of constraints. The Cone of Influence (COI) created from the coded Assertions indicates the completeness of property set applied to the DUT. The Fig .5 (below) represents the creation of Cone of Influence (in Blue) as per Constraints applied and assertions written. The cumulative coverage of Cone of Influence indicates the areas of code which are reachable by the assertions, while the area which is outside the COI indicates the unreachables. The revisiting of the constraints in the Verification Plan and the coded Assumptions can be done at this stage ensuring that the most of the design is accessible.

After this, the actual proving of the coded assertions is done and a more precise coverage is calculated. When an Assertion is run it can either Pass, Fail or be in an undetermined stage depending on how it is proven. In Fig .5 (below), the parts of code which are being exercised appear in Green while the Red dots represent the uncovered code. The area under the COI but not green indicates the unproven code. If the bug exist in green part, it is likely to get caught while it is not guaranteed to be caught even if it exists in the COI. Thus, the Coverage is again collected at this stage in terms of Proof or Bounded Coverage and Verification Plan is revisited to ensure over-constraining is not applied. This can also lead to change in DUT or change in coded Assertions. The process is repeated unless we meet the desired Coverage goals ensuring quality verification.

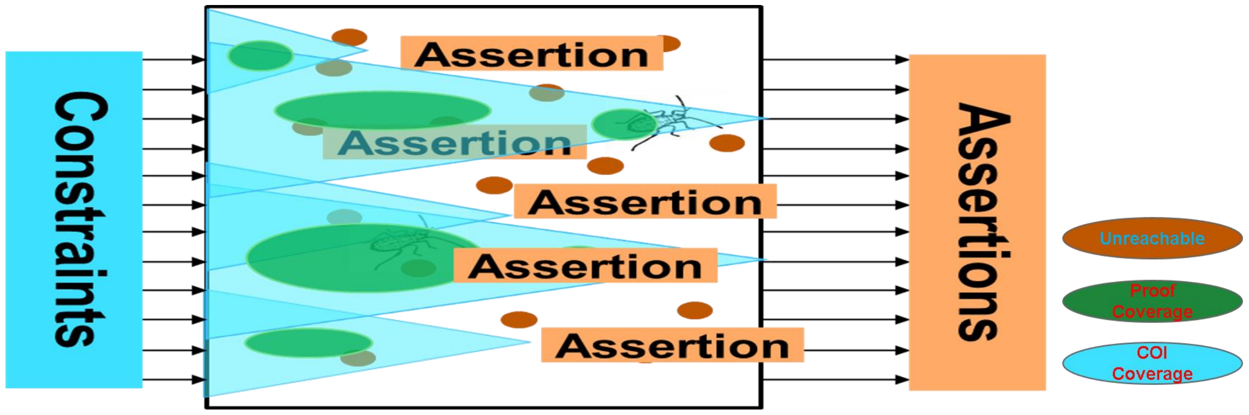


Figure5: Coverage Analysis with Formal

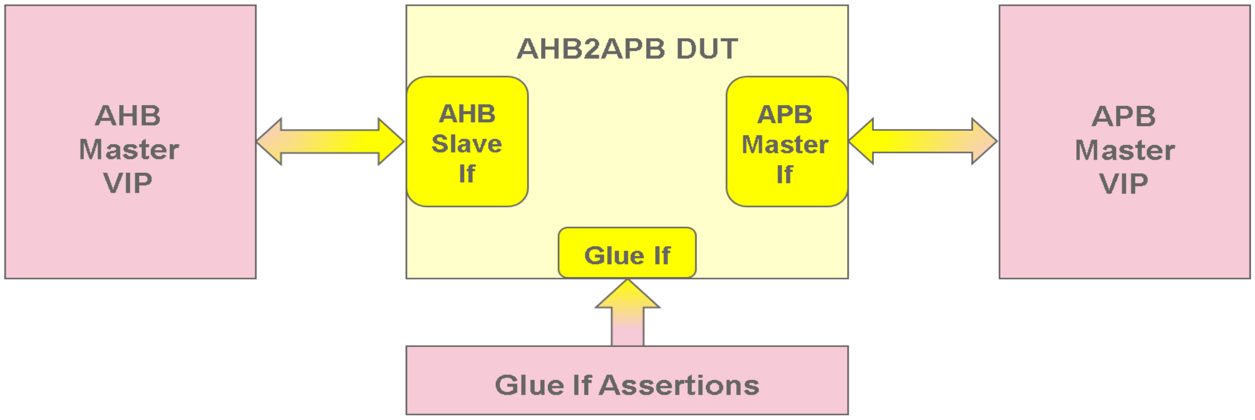


Figure6: AHB2APB Gasket Formal Verification environment

The quality of the Formal Verification is effected by how well the Assumptions and Assertions are coded. This in turn is limited by the quality of the verification plan and analysis of the coverage reports. The other factor which determines the Verification Quality here is how well the Formal Tool performs on a given set of design. The scale and type of design is known to effect the verification quality although this has been improving over time.

Now, coming to the verification of AHB to APB Gasket using the Formal Verification. The Fig. 6 (above) shows the details of the Formal Verification environment for the AHB2APB Gasket. The verification environment consists of the AHB Master ABVIP and the APB Slave ABVIP. The protocol based assertions and assumptions are part of the ABVIP and they are used to check the protocol compliance. It also contains a few custom assertion meant for the small glue logic block. The stimuli, COI and proof coverage are used to evaluate the completeness of Formal Verification. The regressions are run and the reports are generated and analyzed.

The Table2 (below) shows the setup time required for the Formal Verification of the AHB2APB Gasket. The set-up with ABVIP and DUT integration took a week. After that the unreachability analysis on the code was done and assumptions were revisited with respect to the reduced AHB protocol under test. It took a week to run the set of assertions provided by the ABVIPs and collect results. There were some specific assertions and cover statements meant for some specific scenarios.

This was coded and run and the failing assertions were debugged and RTL was fixed for the same. We were able to find a bug related to the glue logic block using this technique. The important thing to note here is that the bug was caught very early in the second week of Formal Verification and that lead to discovery of hole in the Verification Plan for the Glue Logic using UVM based verification. After the bug was found, we added a functional coverage point and directed test to include the given and related scenario.

There was final analysis done on the reports and the Verification was closed. This process took another 3 weeks. The overall process took 5 weeks to close the verification. The table also shows the results collected from the regression run on AHB2APB gasket. Overall 483 assertions and cover statements have been proven with no failing assertions. The unreachable or waived items are mostly due to disabled and untargeted functionality in the VIP.

TABLE 2
AHB2APB FORMAL SETUP AND REGRESSION

Initial Setup	**Unreachability Analysis, ABVIP Assertion and Debug**	**Manual Assertion Coding and Debug**	**Analysis and Coverage Closure**	**Overall Development Time**
1w	1w	1w	2w	5w
Total number of COI Items	**Unreachable Waived Items**	**Undetermined**	**Bounded and Waived**	**Proof Coverage**
415	45	0	5	83
Proven	**Unprocessed**	**Processing**	**Failed**	**Passed**
483	0	0	0	483

C: Portable Stimulus based verification of AHB2APB Gasket

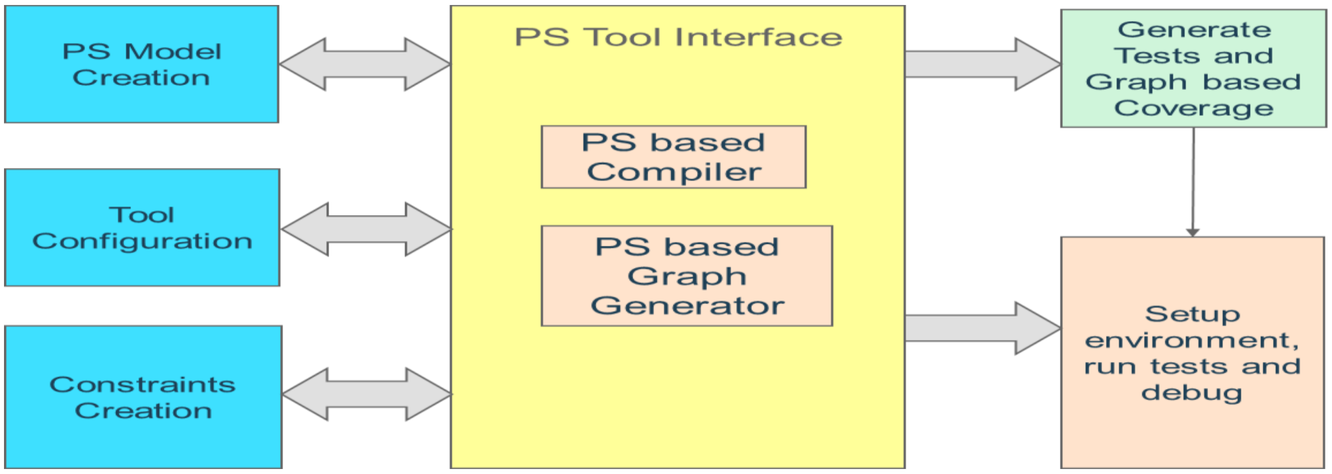


Figure7: Portable Stimulus Work Flow

Portable Stimulus is a new standard which defines a new test writing language that will allow automatic creation of tests targeted to different platforms from a single test source. The tests developed at the IP level would be easily integrated and reused at the SOC level. In addition to horizontal re-use (simulation, emulation, board level, tester etc.), the new language will allow vertical re-use of tests as well.

The Portable Stimulus works at a higher layer of abstraction that is completely independent from the type of target platform. The target platform here can be a UVM based Verification Environment, a C/C++ based SoC based environment, a C and python based Post Silicon evaluation platform and so forth.

The Fig .7 (above) shows how the verification based flow changes when we use Portable Stimulus process. The test intent is represented in the form of Portable Stimulus (PS) models. These models are written in a generic way so that they can be used across multiple platforms. In order to target them to specific platform a Tool Configuration needs to be written. The Portable Stimulus models are created along with the Configuration and given to the tool compiler. The compiler parses these inputs and generates a visual representation of the tests in form of graphs or flow diagram. The constraints for the tests can be applied on this visual representation and then the tests are generated for the target platform along with the Graph based coverage. These generated tests then needs to be integrated with the given target platform. For example, if the target platform is UVM based environment, the test needs to be integrated with the UVM-SV side with some interfacing logic and system calls used by the tool compiler. Once, this system is in place, the verification process runs as usual.

The Fig .8 (below) shows the Verification Flow with Portable Stimulus. It starts with the creation of Verification Plan as per design specification and the setting up of the verification environment. The test intent is captured in terms of Portable Stimulus Models, Constraints and Configuration files. The tools supporting this standard can then generate the tests for a given type of Verification Environment and graph based coverage is collected. The analysis of this type of coverage can indicate the holes in the test constraints and configuration and the process can be revisited.

The Functional Coverage Points are also created with respect to the verification plan to ensure we meet the specification. After this, the simulations are run and coverage database is created collecting Code and Functional Coverage. The database is analyzed and coverage holes are either waived or end up causing changes in the Verification Plan. The process is repeated unless we meet the desired Coverage goals ensuring quality verification.

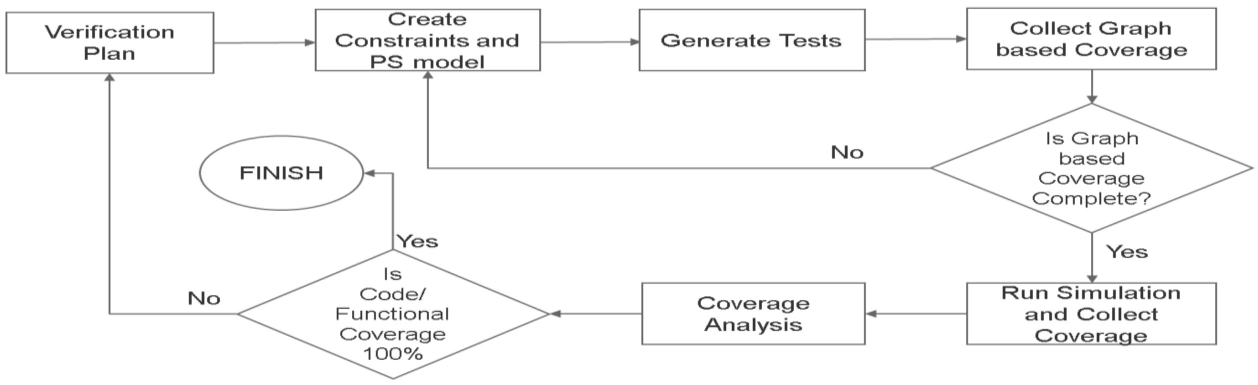

Verification Plan
Create Constraints and PS model
Generate Tests
Collect Graph based Coverage
Is Graph based Coverage Complete?
No
Yes
Run Simulation and Collect Coverage
Coverage Analysis
Is Code/ Functional Coverage 100%
Yes
No
FINISH

Figure8: Verification Flow with Portable Stimulus

The randomization mechanism in PS based verification starts from an abstract description of the legal transitions between the high-level states of the DUT and automatically enumerates the minimum set of tests needed to cover the paths through this state space. This is a very promising feature which can ensure a better quality tests as compared to manually written tests. The tests can be visually seen allowing users to understand the control and data flows in a better way. The graph based coverage allows user to view the transversed paths and allows tests to be generated such that maximum length of the graph is covered. It is thus able to achieve higher coverage in a far fewer cycles than the usual constrained random verification. Also, some tools allows active checks to be put during the run-time allowing an effective automated checking. This would combine with the scoreboard checking and assertions based check points improving the quality of verification.

The Portable Stimulus Methodology works at a higher layer of abstraction and then integrated with underlying verification process. Due to this, although there is a definite improvement in the test or stimulus generation process, this verification methodology would still inherit the underlying process in its original form. In case of integration with UVM based environments, on one hand it will be benefitted by the reuse of verification components, on the other hand, it will get limited by its complexity. On the same lines, the quality of verification is limited by the quality of the verification plan and analysis of the coverage reports.

The Fig .9 (below), shows the Portable Stimulus based Verification Environment setup for the AHB2APB gasket. The PS based environment is inherited from the AHB2APB UVM based environment. It consists of AHB Master UVC and APB Slave UVC along with top level environment, configuration, Virtual Sequencer and Scoreboard. The environment is controlled by a top level UVM test which on one hand calls the Virtual Sequences to control the UVC operation, on the other hand it interacts with the Portable Stimulus generated format with PLI or DPI based system calls.

The Portable Stimulus Model for AHB2APB gasket is a single representation of stimulus and test scenarios. It consist of two parts:

- Exec Blocks: The *Exec* blocks are statements from the external code used in target platforms under the PS based wrapper. For the UVM SV part it has logic derived from tool provided macros (TX Gen) which translates and interact with the SV world with PLI based system calls. It also has a part (C Gen) which can translate and interact with C side and allows seamless reuse across different platforms.
- Test Intent: This is the actual use case model based on entire Verification Plan. It consists of collection of functions which will be equivalent of higher level sequences to perform a set of actions. These functions will eventually call the functions from the Exec Blocks to execute commands on the SV side.

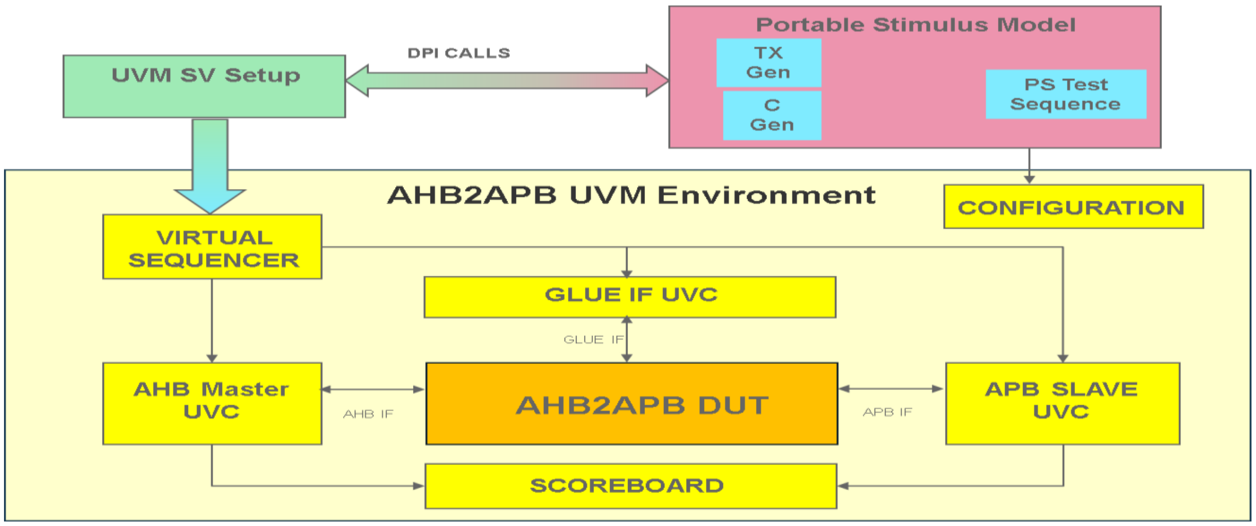


Figure9: AHB2APB Gasket PS based environment

The model is compiled and converted into its visual representation in the forms of flow diagram or graphs. The Figure10 (below) shows the graph generated from the model. The process block in green represents the sequences coded and the conditions generated for this design. Each sequence, if selected, can generate a different set of tests or a combination with other sequences can create more complicated tests sequences. The paths of the graph are analyzed and the constraints are applied on the model. We can select the paths from the graph and create a use case or let the tool randomize from the given paths and create a test based on constraints provided.

The use-cases and random tests are generated along with the graph based coverage which describes the effectiveness of the test generation. The graph coverage in this case can tell us which sequences have been covered while which of them need to be covered. A visual representation here allows to see the uncovered conditions and pick the conditions which can be covered. We can also make the tool walk through all the possible combinations of the graph for a given set of constraints and generate tests. This kind of approach naturally creates test which will cover more conditions as compared to generating them with a System Verilog based randomization. Also, this would reduce the number of cycles to reach to a particular point in coverage earlier than the traditional simulation.

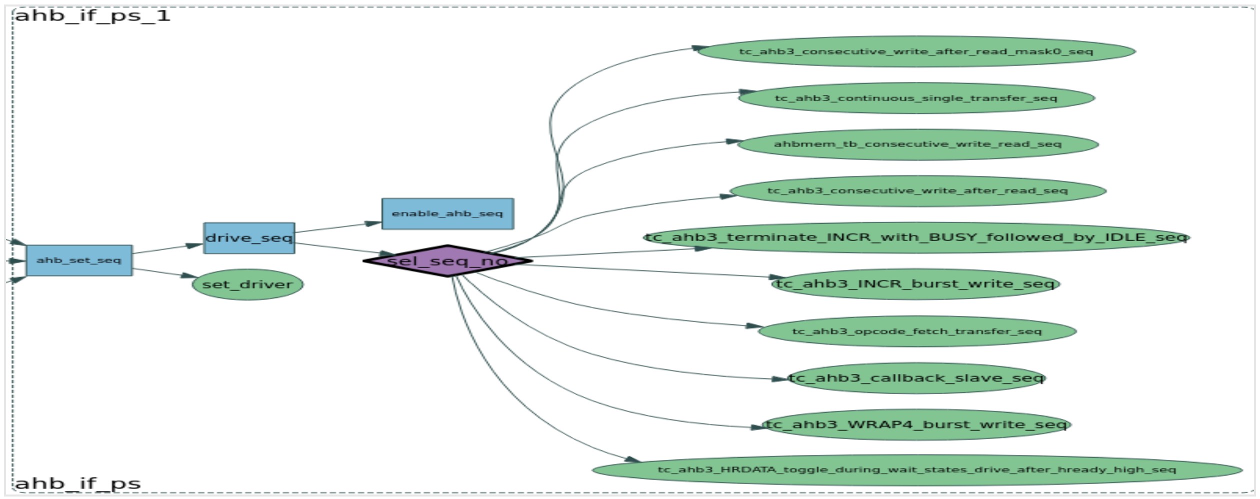
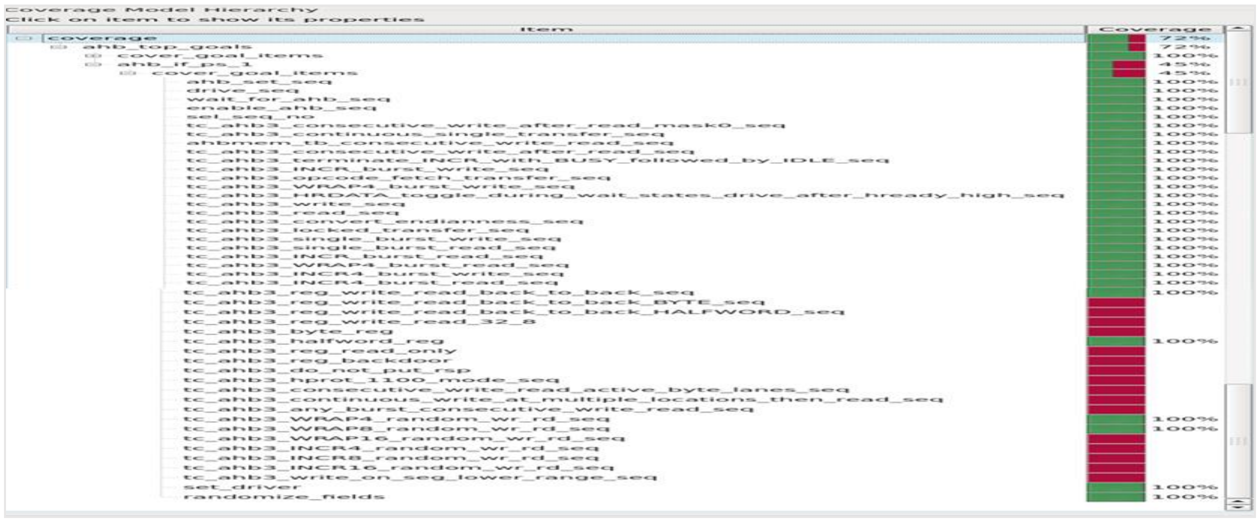

ahb_if_ps_1
ahb_set_seq
drive_seq
set_driver
enable_ahb_seq
sel_seq_no
tc_ahb3_consecutive_write_after_read_mask0_seq
tc_ahb3_continuous_single_transfer_seq
ahbmem_tb_consecutive_write_read_seq
tc_ahb3_consecutive_write_after_read_seq
tc_ahb3_terminate_INCR_with_BUSY_followed_by_IDLE_seq
tc_ahb3_INCR_burst_write_seq
tc_ahb3_opcode_fetch_transfer_seq
tc_ahb3_callback_slave_seq
tc_ahb3_WRAP4_burst_write_seq
tc_ahb3_HRDATA_toggle_during_wait_states_drive_after_hready_high_seq
ahb_if_ps

Figure10: Graph generation from Model



Coverage Model Hierarchy
Click on item to show its properties

Item	Coverage
coverage	72%
ahb_top_goals	72%
cover_goal_items	100%
ahb_if_ps_1	45%
cover_goal_items	45%
ahb_set_seq	100%
drive_seq	100%
wait_for_ahb_seq	100%
enable_ahb_seq	100%
sel_seq_no	100%
tc_ahb3_consecutive_write_after_read_mask0_seq	100%
tc_ahb3_continuous_single_transfer_seq	100%
ahbmem_tb_consecutive_write_read_seq	100%
tc_ahb3_consecutive_write_after_read_seq	100%
tc_ahb3_terminate_INCR_with_BUSY_followed_by_IDLE_seq	100%
tc_ahb3_INCR_burst_write_seq	100%
tc_ahb3_opcode_fetch_transfer_seq	100%
tc_ahb3_WRAP4_burst_write_seq	100%
tc_ahb3_HRDATA_toggle_during_wait_states_drive_after_hready_high_seq	100%
tc_ahb3_write_seq	100%
tc_ahb3_read_seq	100%
tc_ahb3_convert_endianness_seq	100%
tc_ahb3_locked_transfer_seq	100%
tc_ahb3_single_burst_write_seq	100%
tc_ahb3_single_burst_read_seq	100%
tc_ahb3_INCR_burst_read_seq	100%
tc_ahb3_WRAP4_burst_read_seq	100%
tc_ahb3_INCR4_burst_write_seq	100%
tc_ahb3_INCR4_burst_read_seq	100%
tc_ahb3_reg_write_read_back_to_back_seq	100%
tc_ahb3_reg_write_read_back_to_back_BYTE_seq	
tc_ahb3_reg_write_read_back_to_back_HALFWORD_seq	
tc_ahb3_reg_write_read_32_8	
tc_ahb3_byte_reg	
tc_ahb3_halfword_reg	100%
tc_ahb3_reg_read_only	
tc_ahb3_reg_backdoor	
tc_ahb3_do_not_put_rsp	
tc_ahb3_hprot_1100_mode_seq	
tc_ahb3_consecutive_write_read_active_byte_lanes_seq	
tc_ahb3_continuous_write_at_multiple_locations_then_read_seq	
tc_ahb3_any_burst_consecutive_write_read_seq	
tc_ahb3_WRAP4_random_wr_rd_seq	100%
tc_ahb3_WRAP8_random_wr_rd_seq	100%
tc_ahb3_WRAP16_random_wr_rd_seq	
tc_ahb3_INCR4_random_wr_rd_seq	
tc_ahb3_INCR8_random_wr_rd_seq	
tc_ahb3_INCR16_random_wr_rd_seq	
tc_ahb3_write_on_seg_lower_range_seq	
set_driver	100%
randomize_fields	100%

Figure11: Graph based coverage generation

The Figure 11 (above) shows the example of the graph based coverage as generated out of the PS models. The red parts here shows the uncovered paths while the Green parts show the covered paths of the graph. The user can look at this graph and can try to achieve a higher coverage by modifying the graph constraints or by increasing the number of targeted tests. An analysis of this graph based coverage can lead to constraints modification or the modifications in model as well. This process can get repetitive unless desired results are obtained. Once the test are generated satisfactorily, the simulation is run as usual. The regression runs and collection of the coverage are the next natural steps. After the regressions are run, the coverage reports are generated, merged and analyzed. The functional coverage, the code coverage and Graph based coverage are used as a sign-off criterion for verification.

The Table3 (below) shows the time taken (in weeks) to develop the PS based environment. We reused the setup from the UVM environment and integrated PS related setup in a couple of days. Thus concluding that it should not be very different from the UVM environment bring up time of 1 week even if we develop it from scratch. As this is relatively newer language it took us 2 weeks for the early model development. After this test generation and analysis of the tests from the graph coverage and run roughly took 1 weeks although this was a cumulative process. After this, the coverage analysis started and it took additional 2 weeks to close the coverage to an acceptable level. Overall effort in this case was roughly 6 weeks which is very similar to the UVM based Verification effort. The table also shows the results collected from the regression run on AHB2APB gasket with PS based Methodology. Overall, 50 testcases were generated and run with all of them Passing. The column on the right indicates the overall coverage obtained without exclusions which is exactly same as the one seen in UVM based environment. The uncovered logic is either non-reachable code with disabled functionality making this coverage practically 100%. The important point to note here is that the number of tests required to reach this number has reduced almost by half because the test covers more conditions. Thus, we can conclude that better quality tests are generated using PS based approach.

TABLE 3
AHB2APB PS SETUP AND REGRESSION

Initial Setup	**Initial Model Development**	**Test Generation, Graph Coverage and Run**	**Coverage Closure**	**Overall Development Time**
1w	2w	1w	2w	6w
Tests Run	**Passed**	**Failed**	**Not Run**	**Overall Code Coverage**
50	50	0	0	2634/2864

D: SoC or SYSTEM LEVEL ANALYSIS

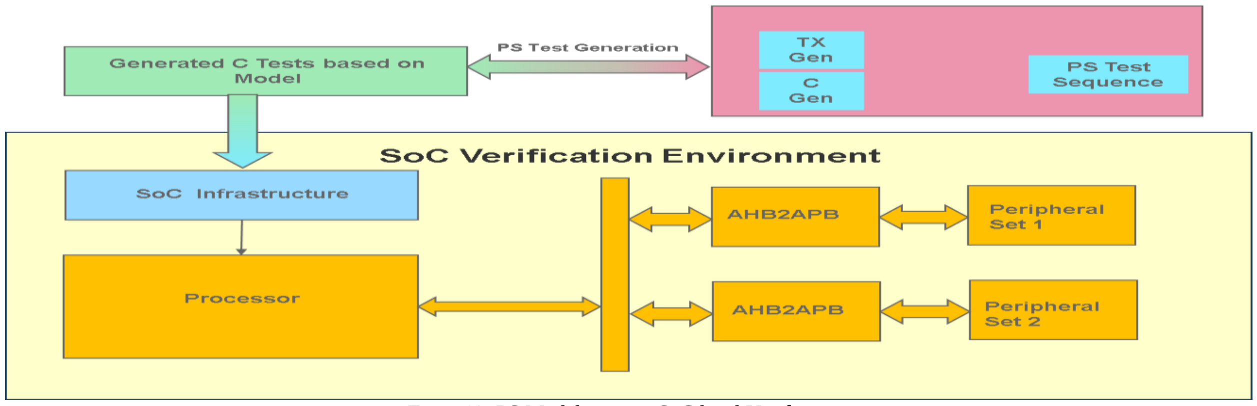


Figure12: PS Model reuse at SoC level Verification

At the SoC level verification, the UVM and Formal techniques are used with different approaches. The UVM based verification is generally used along with the directed C tests for integration checks and use cases. The reuse from the IP level can be in the form of UVM Monitors to monitor the protocol or Scoreboards to check specific points of interests. The test and the sequences which contains major part of the specification needs to be re-done in C with different focus.

The Formal Verification is generally used for connectivity checks at the SoC level using automated techniques but this is completely different from what we have been doing at the block level. The assumptions and assertions written at IP level can be used to some sub-system level checking but that requires a lot of work on writing new sub-system level assumptions. The Formal tool performance on different kind of designs can be a deciding factor in its reuse even at the sub-system level verification.

The PS based verification techniques are on the other hand designed for IP to SoC test reuse. The Fig. 12(above) shows the reuse of AHB2APB PS models at the SoC level verification. The models coded at the IP level are configured for different address maps as per SoC specification and targeted for C test generation. Once the C test are generated, they are integrated with the SoC setup with some system specific standard infrastructure. The C tests are then compiled and run on the processor to generate transactions. The same set of sequences written at the IP level with the graph based constrained randomization are possible to reuse. Almost all the sequences in the model with exception of the parts meant for "Exec" code are reusable when we write model for the Processor based applications. Using this reuse technique, we were able to find an integration bug at the system level where a sync signal was not connected properly.

The ability to reproduce graph based constrained random tests at the IP level without the need to actual recoding of the scenarios is a major advantage in PS based verification. It also allows the test generation to different instances of the same IP with different address maps. Along with this, when different kind of PS models for different IP combined together at the SoC level, the creation of complex scenario is possible which are otherwise difficult to code manually.

E: SILICON VALIDATION

The Silicon Validation process is a completely different process as compared to simulation with a lot of dependence on the Evaluation board setups and the hardware configuration for testing. The standard process of UVM and Formal based verification can't be used here. The PS based verification on the other offers a reuse opportunity as the PS models coded at the IP level can be targeted for C test generation for Post Si Validation.

Although, we haven't reused the C tests for AHB2APB model at the Post-Si based application as yet, but we are confident that it can be used on any evaluation platform employing C-based tests. Infact, this kind of models where SoC based PS models are reused for Post Si evaluation boards have been proven for other processor based applications. This reuse is the kind of application which is unique to Portable Stimulus based approach only.

III. COMPARATIVE ANALYSIS: PS VS UVM

The Portable Stimulus solution uses the UVM based environment and adds value to it. It allows the advantages of UVM, in terms of component reusability and a pre-defined standard approach to be used. But, it has an additional overhead though of learning a new Portable Stimulus Standard. Let us compare both the methodologies with respect to undergiven factors as applied to AHB2APB environment:

A: Verification Planning and Development Time

The Verification Plan for AHB2APB verification is based on the design specifications and it is very similar in both the cases. The planning for Portable Stimulus model should have more details on the model organization than its UVM version.

The PS standard is relatively new and would require some learning before the project execution. But, given the fact that it is an extension of C++, it is relatively easier to learn. Once, the learning part is over, the effort required to code the models is roughly same as required for writing the tests.

The AHB2APB environment with PS reused the same planning, infrastructure and environment and required some extra work to integrate models to the UVM environment. The model development time was roughly the same as writing the tests cases. In summary, there was minor additional work required in PS based AHB2APB environment as compared to the UVM based environments but it also added value to it.

B: Infrastructure development and Ease of integration

As mentioned above, the PS models requires some additional work on top of the UVM infrastructure for their integration. It requires some extra logic, parts of it coded in PS Standard and some in SV, for effective integration. More specific details would depend on the PS compiler used, but in general, there would be placeholders required for data exchange between the UVM components and PS models.

The AHB2APB environment have placeholders in the UVM base test to allow data transfer between the model and SV layer. The PS model have an “exec-code” or “user-defined code” coded in the PS standard for the same purpose. In addition to this, the SV side have virtual sequence to control the AHB UVC basic sequences. All of this, is more of a one-time effort for a given environment and majority of it is tied to the kind of UVC used in the environment.

This also implies that the integration code would largely remain the same, if we work on some other verification environment employing AHB and APB UVC. There is an opportunity here to automate the integration flow with PS models further reducing the infrastructure development time to match the UVM Verification Environment.

C: Effective test generation

The Portable Stimulus solution allows models to be represented as a graph or a data flow diagram as shown in Figure 10 (in PS section). This definitely looks a better way to visualize the programming flow and test conditions than one would do in directed testcase development. This also allows the clubbing of multiple test conditions in one model giving a different take on the possible scenarios, at times, reconsidering the tests originally planned for the environment.

The PS standard also gives an opportunity for the tool to randomly walk through the paths and create constrained random tests. Some of the tool providers also add macros and checking routines which can be used as a part of the models, thus allowing better quality tests and shorter regression cycles.

The Table 4 (below) shows the comparison in coverage numbers with UVM and PS based methodologies with same number of random loops and with same number (10) regression runs with different seeds. The higher coverage with less number of runs indicates that the test generated with the PS based approach covers more number of scenarios as compared to the UVM based random tests. As the test generate are same in number of cycles the individual simulation run time is similar in each case but the ability to hit more specific cases allows the overall regression time to be much lesser than the UVM tests.

The other advantage of the test generated from PS models is the graph coverage. The 100% graph coverage makes sure that the given set of tests completes all the possible scenarios portrayed by the models. For AHB2APB environment, the number of tests required to achieve same level of coverage was reduced to almost half from 105 to 50. This itself is an indication of better quality of the tests, covering the higher number of scenarios in less number of simulation cycles, as compared to traditional simulation.

TABLE 4
AHB2APB PS SIMULATION COMPARISON

Verification Environment Type	**Coverage numbers with 10 seed regression**
UVM based	65.38%
PS based	85.31%

D: Ease of analyzing report and closing the uncovered coverage goals

Both UVM and PS based solution uses the Code Coverage and Functional Coverage to judge the quality of the verification. The PS based solution has an extra parameter of graph coverage as shown in Figure 11(in PS section) that is a measure of how well all the scenarios in a model have been covered. A 100% graph coverage would mean that all the branches of the graph generated from the PS model have been covered. This is very helpful in visualizing the scenarios which are likely to get missed or take more iterations when we are looking to create directed tests for the uncovered points. The Table 4 (above) also indicates better coverage numbers in case of PS based verification indicating that the coverage goals are easier to meet. This would also mean that the overall regression run time would also be much lesser than the UVM based regressions. The results indicated in Table 3 (in PS section) also indicates that the number of tests required to reach the same number of end goal code coverage (without waivers) is lesser in case of PS based verification. Thus we can conclude that the coverage closure would be faster when we use Portable Stimulus based solution.

E: Extent of reuse and portability to other verification environments

The reusability of the UVM components in both UVM and PS based solutions is same when we are looking to do a vertical reuse. The tests written in System Verilog cannot be used in the processor based system which generally have C/C++ based tests. The PS based models can generate tests seamlessly which can be used not only in the processor based systems in the simulations but at the other systems like Evaluation boards, FPGA, emulation and testing platforms etc. The tests AHB2APB example were reused in the System Verilog processor model and has potential to be used in other applications utilizing C based tests. Also, as mentioned in the PS section we were able to find an integration bug related with the Sync signal using the reused tests from the block level. This kind of test reuse is not possible in UVM based tests.

IV. COMPARATIVE ANALYSIS: PS VS FORMAL

A comparison of PS vs Formal will essentially have an essence of dynamic vs static verification. The two techniques are very different from each other and it is difficult to compare them directly. The coverage matrix for each techniques are quite different but still results at the end of the activity are the measure of quality. The complexity and scale of design is also something which changes this analysis. The analysis for the AHB2APB verification environment the observations are given below:

A: Verification Planning and Development Time

The verification plan for the Formal Verification contains assertions and assumptions derived from the specifications and they are more closely linked to the specification as compared to UVM or PS based verification plan. The plan for the AHB2APB VIP is more detailed and takes more time as compared to the PS Verification plan.

The Table 2 (in Formal Section) indicates the time taken to develop and run the assertion was one week lesser as compared to PS Model development time. The availability of the Assertion based VIP is a major advantage in this case as it covers almost complete protocol details for this. It almost covers the major part of the specifications which could be hard to code and bring-up in SVA. The parts which are non-protocol related are lesser in this case leading to lesser assertion work. Due to this, the assertion development time is very less as compared to PS Models in this case.

B: Infrastructure development and Ease of integration

The Formal Verification requires the design to be read by the compiler and in order for the tool algorithm to work efficiently parts of the design need to be blackboxed or glassboxed. In case of AHB2APB gasket there was no such requirement because it is a smaller design. The infrastructure and integration in this case is straight forward as compared to PS based Verification.

C: Effective test generation

The stimulus generated by the Formal Verification tool depends on the constraints or assumptions applied to the design. In this case, the AHB and APB ABVIP pretty much take care of this and so this was again very straight forward apart for the need to disable a few assertions which were meant for a bigger scale of AHB protocol. With this, pretty much all the desired protocol checks happened and bugs were seen which were not seen with the UVM based verification. The PS based verification on the other hand does effective test generation as well but it is a different approach to verification and the ability to hit bugs get limited by the functionality coded into the models.

D: Ease of analyzing report and closing the uncovered coverage goals

As mentioned earlier, it is difficult to measure the Formal and Dynamic Coverage directly due to difference in their approaches but still looking at the results in a different light does convey information about the quality of verification. The coverage reports in the case of Formal Verification is of four types namely COI, Stimuli, Proof and Bounded. These four give a fair idea of the design covered by the assertions and are easy enough to analyze at two different stages. The Table 2 (in Formal Section) illustrates the different type of coverage data in brief. The uncovered points can be easily covered by coding an assertion for it or waiving depending on the end goal. The PS on the other hand, uses the traditional code and functional coverage along with the additional graph based coverage which makes the analysis simple as well.

E: Extent of reuse and portability to other verification environments

The assertions are portable and assumptions can be converted to assertions when we are working in simulation. But when we change the simulation platform the reuse is not possible. This is where the Portable Stimulus scores because the test intent can be reused for different target platforms like Evaluation boards, FPGA, emulation and testing platforms. In the AHB2APB gasket example the AHB model was reused at SoC level as a processor model where it was used to generate the C tests. This can be easily carried over to the Post Si platform as well and reuse of the logic once written is possible. Also, as mentioned in the PS section we were able to find an integration bug related with the Sync signal using the reused tests from the block level which may take more effort to catch in case of assertions.

V. SUMMARY

As compared to UVM based solution, the PS based verification environment fares equally in terms of many factors like component reuse, environment bring up time and ease of integration. But, it offers value addition in terms of visual test representation, constraints setting and dataflow based randomization. It clearly has an edge when it comes to reusability to other target application like SoC based environments, automatic test generation to cover all the graphical nodes and an additional graph based coverage. It has an additional overhead though of learning a new Portable Stimulus Standard but to its defense the language is an extension of C++ and so is relatively easier to learn.

The Formal Verification is effectively better in finding corner case scenarios at the IP level but PS with its visual test approach can definitely perform better than the traditional dynamic simulation approach. The major advantage that PS have over Formal is that the test intent can be reused for test generation for different target applications like SoC level verification.

The PS based solution compliments the dynamic verification techniques like UVM to improve the quality of the verification. Formal Verification on the other hand, is clearly a winner on specific kind of designs if we are looking to perform white box testing. When it comes stimulus reuse, PS based solutions are definitely the way going forward.

REFERENCES:

[1] http://www.accellera.org/activities/working-groups/portable-stimulus
[2] https://www.arm.com/products/system-ip/amba-specifications
[3] http://accellera.org/downloads/standards/uvm
[4] https://www.cadence.com/content/cadence-www/global/zh_TW/home/tools/system-design-and-verification/formal-and-static-verification/jasper-gold-verification-platform.html
[5] http://www.brekersystems.com/products/trekuvm