Case Study of Verification Planning to Coverage Closure @Block, Subsystem and System on Chip Level

Paul Kaunds, Revati Bothe, Jesvin Johnson

sondrel

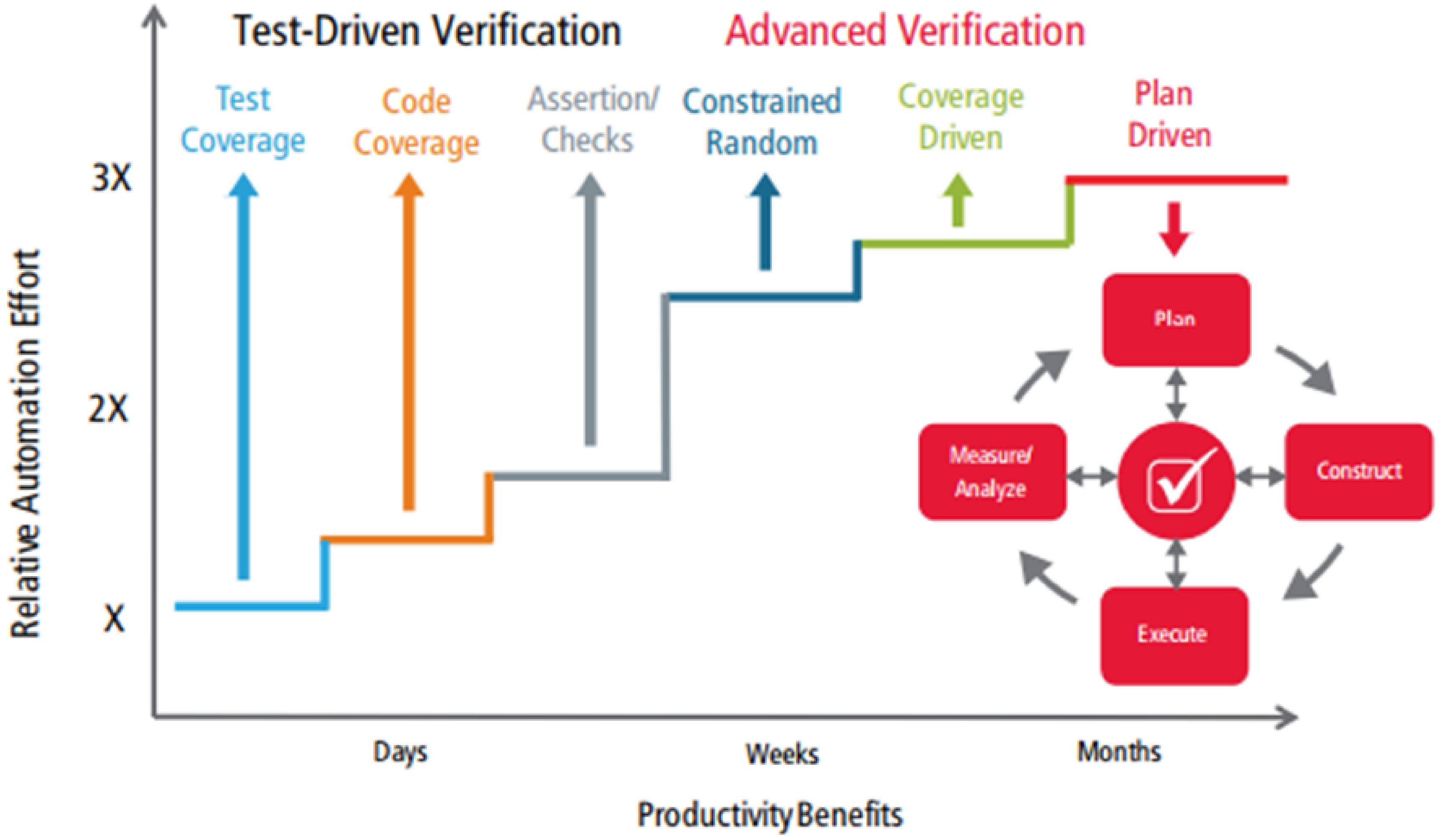
Success through partnership

Credits

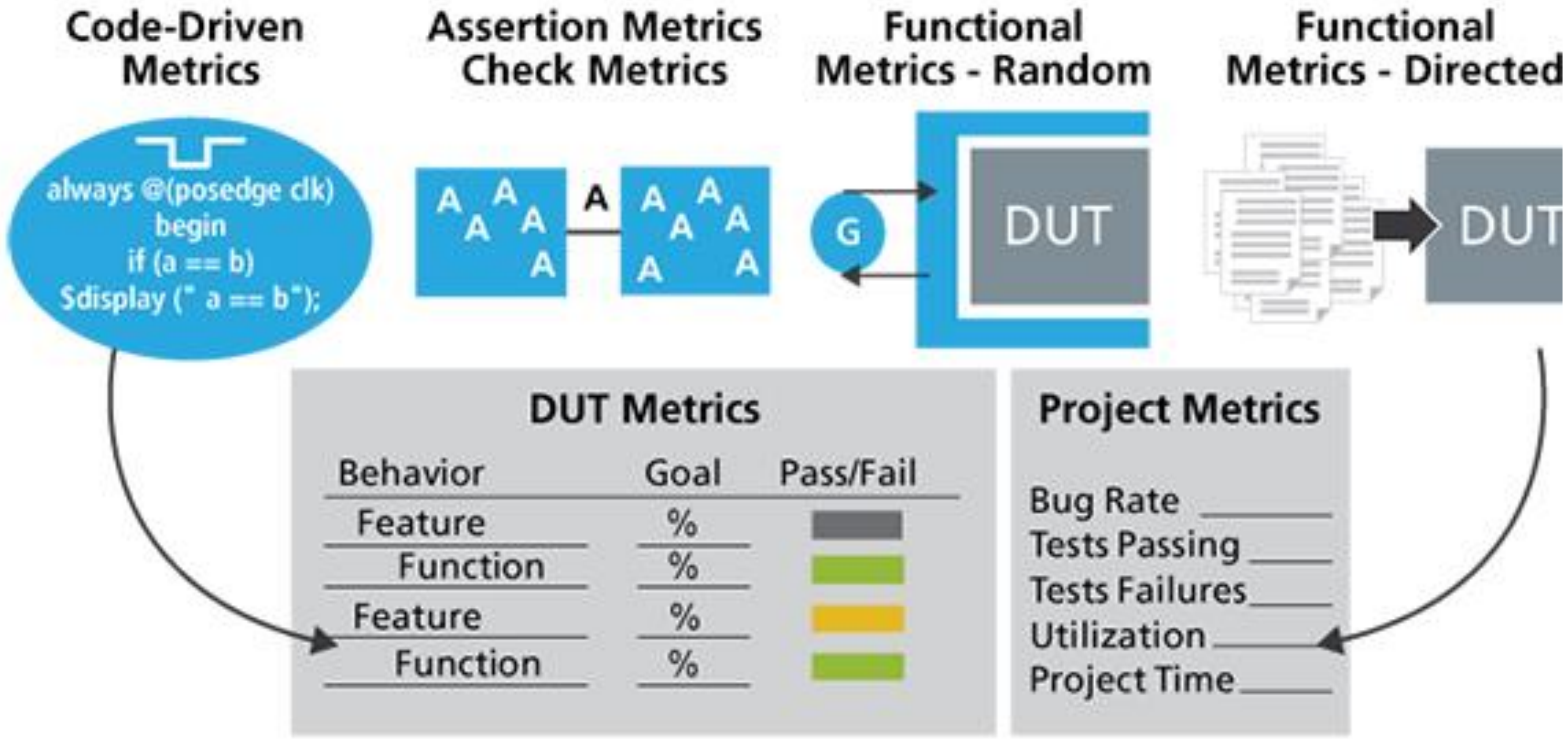
- All EDA flow/methodology diagram credits goes to respective EDA Vendors.

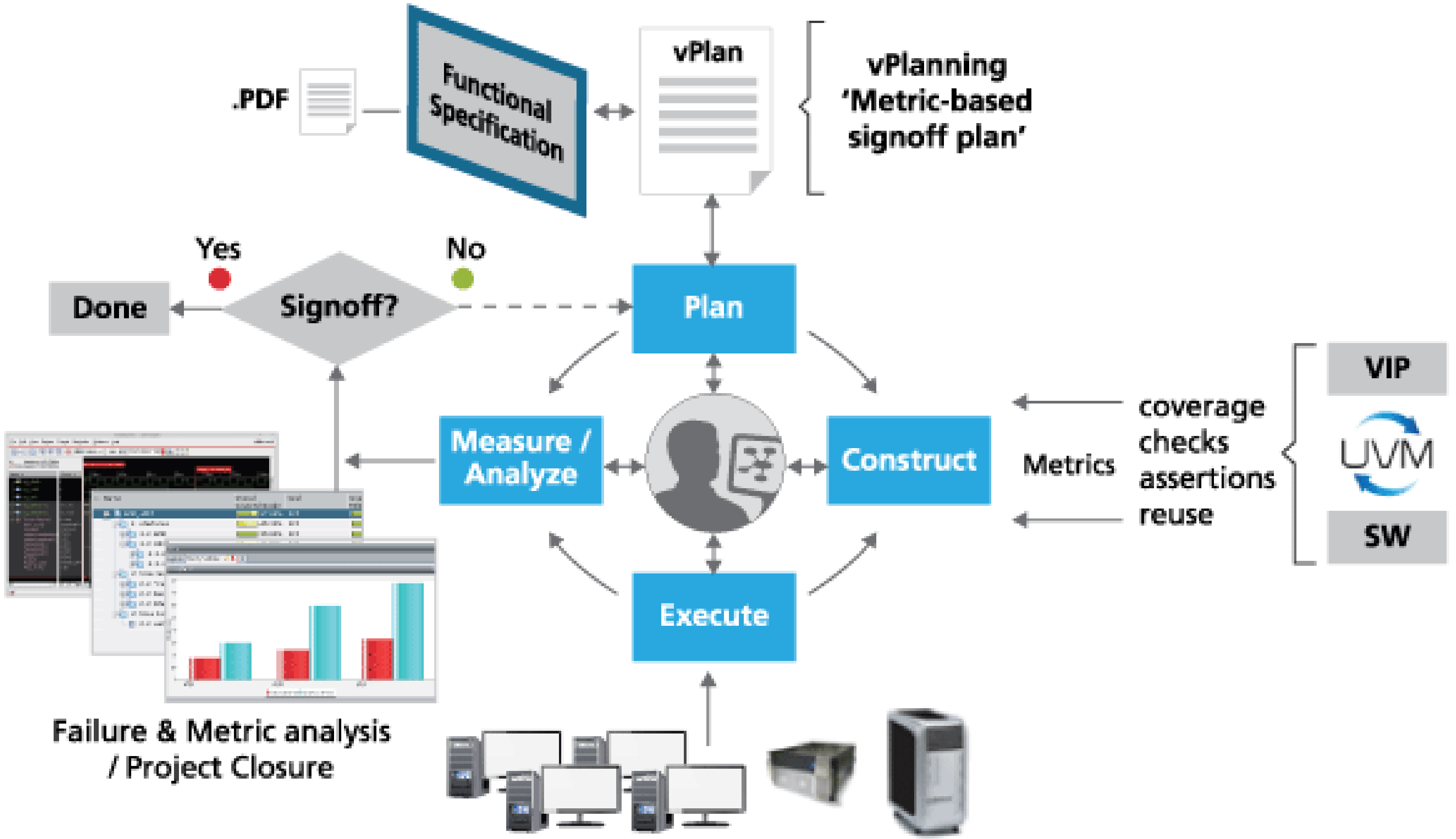
What will you learn

- Metric Driven Verification flow and Best Practices at Planning, Development and Execution stages
- Verification Methodologies, Challenges faced and lessons learnt during verification at block level, subsystem level and system on chip level



MDV investment alternatives for RTL verification


Code-Driven Metrics
always @(posedge clk)
begin
if (a == b)
Sdisplay (" a == b");
Assertion Metrics Check Metrics
A
Functional Metrics - Random
G
DUT
Functional Metrics - Directed
DUT
DUT Metrics
Behavior
Goal
Pass/Fail
Feature
%
Function
%
Feature
%
Function
%
Project Metrics
Bug Rate
Tests Passing
Tests Failures
Utilization
Project Time


.PDF
Functional Specification
vPlan
vPlanning
'Metric-based signoff plan'
Yes
No
Done
Signoff?
Plan
Measure / Analyze
Construct
Execute
Metrics
coverage
checks
assertions
reuse
VIP
UVM
SW
Failure & Metric analysis / Project Closure
Simulation, Formal, Acceleration LowPower, AMS, Emulation

Metric Driven Verification Phases

- MDV consists of the following phases:
  - Verification Planning Phase
  - Verification Environment Development Phase
  - Test case development and Coverage closure Phase

- Each phase is signed off based on a standard sign-off document for that phase.

Planning Phase

- Defines the strategy for verifying the design under test.
- Two documents are developed during this phase
  - Verification Strategy Document
  - Verification Plan Document

- These documents are reviewed by the relevant stake holders and are signed off based on the planning phase sign-off document.

Verification Strategy Development

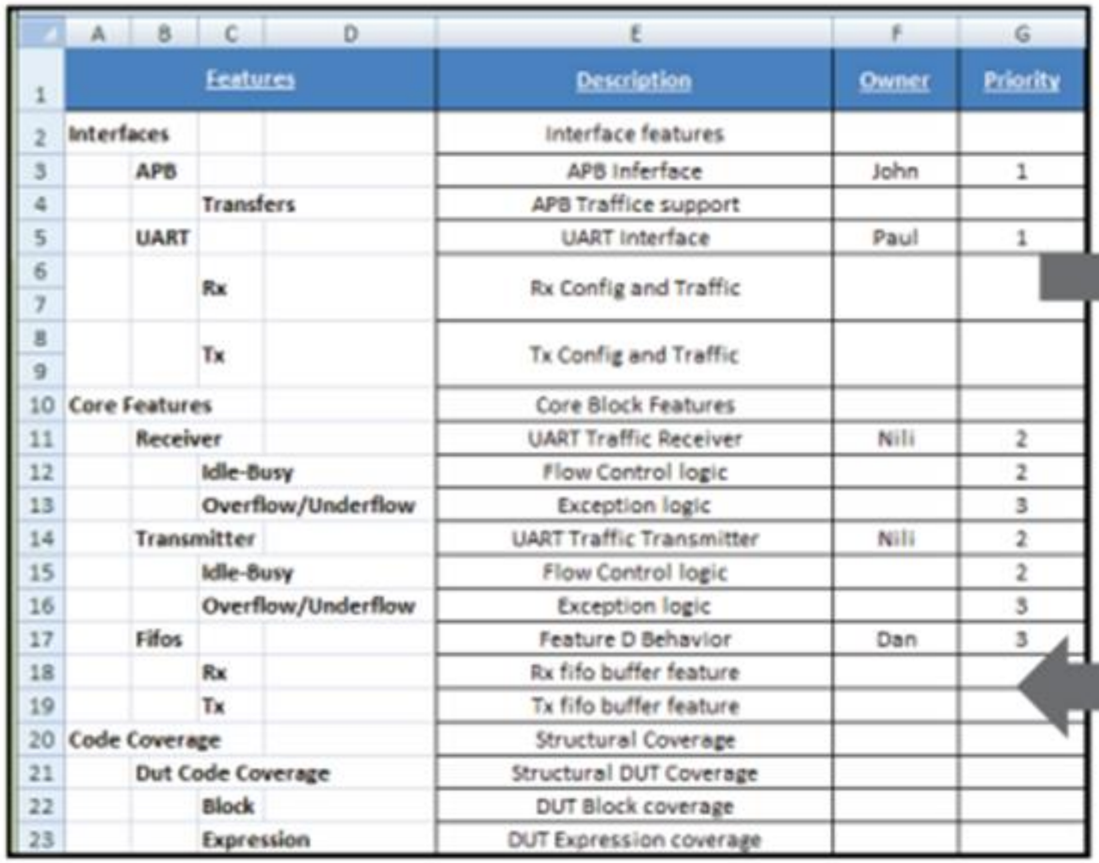
- Verification approach like Constrained random, Usecases, Directed testcases
- Verification Environment block diagram
- Reuse of External and Internal VIPs/UVCs/Sequence library/Tests

Verification Plan Development

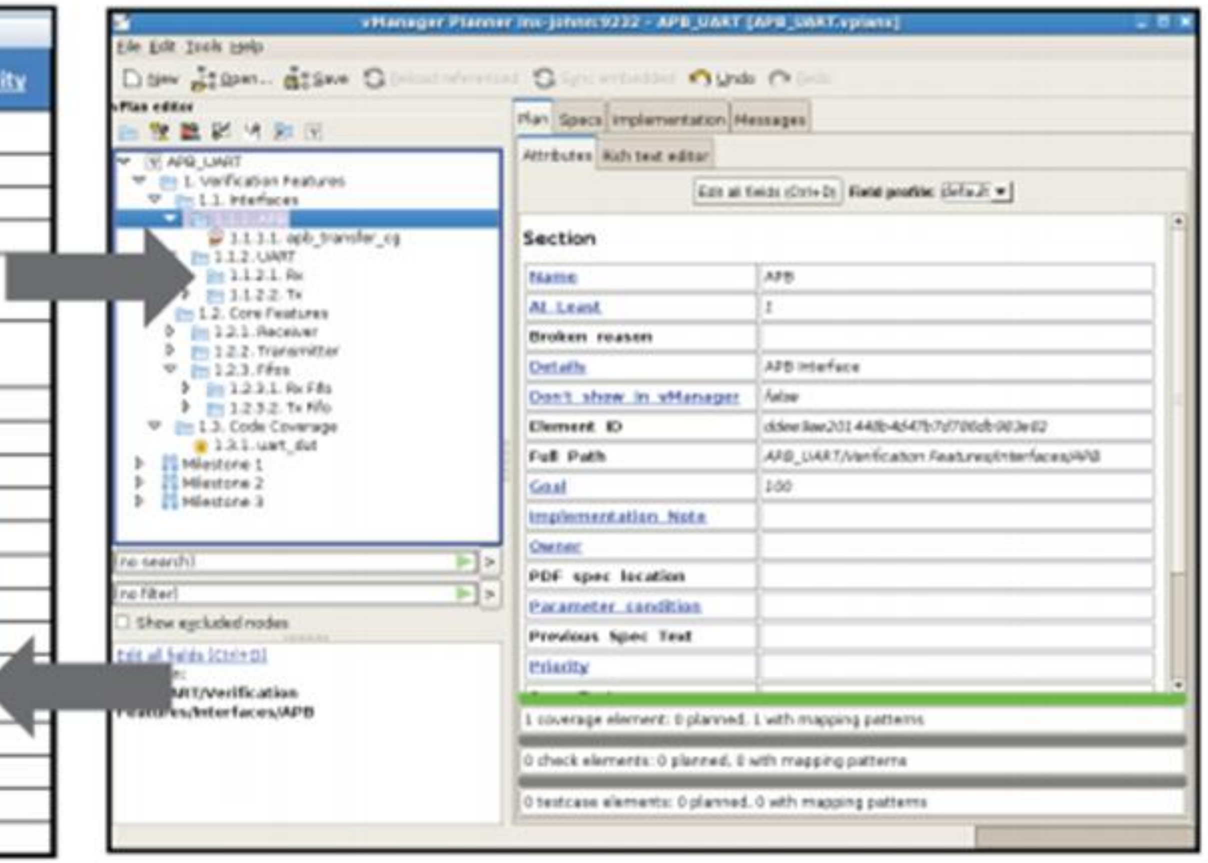
- Plan links detailed feature lists to cover points, checkers and testcases, which allows tracking of progress and measuring against the plan.
- Spec Annotation
- It is imporant to categorise the plan according to the milestones like for example "pre-alpha" , "alpha", "beta" , "beta with verification" and "final" for better project tracking. It also helps to deliver with definitive understanding of risks. These milestone would provide clarity on the status of the design and the level of verification. Usually done setting perspectives/filters within the verification plan.

Verification Plan Milestone

Milestone	Status
Pre-Alpha	Early RTL Design stage and basic testbench architecture development , clock and reset checks.
Alpha	Testing register interfaces. Plan reviewed. Verification at 60 %
Beta	RTL design almost complete and ready for code freeze. Tests memory map of the design and basic features of the design . Internal review of testbench architecture and test sequences and testcases.
Beta + Verification	RTL frozen. Verification at upper 90%
Final	Verification 100% .



	A	B	C	D	E	F	G
1	Features				Description	Owner	Priority
2	Interfaces				Interface features		
3		APB			APB Inferface	John	1
4			Transfers		APB Traffice support		
5		UART			UART Interface	Paul	1
6			Rx		Rx Config and Traffic		
7							
8			Tx		Tx Config and Traffic		
9							
10	Core Features				Core Block Features		
11		Receiver			UART Traffic Receiver	Nili	2
12			Idle-Busy		Flow Control logic		2
13			Overflow/Underflow		Exception logic		3
14		Transmitter			UART Traffic Transmitter	Nili	2
15			Idle-Busy		Flow Control logic		2
16			Overflow/Underflow		Exception logic		3
17		Fifos			Feature D Behavior	Dan	3
18			Rx		Rx fifo buffer feature		
19			Tx		Tx fifo buffer feature		
20	Code Coverage				Structural Coverage		
21		Dut Code Coverage			Structural DUT Coverage		
22			Block		DUT Block coverage		
23			Expression		DUT Expression coverage		



vManager Planner ins-johnn:9222 - APB_UART [APB_UART.vplanx]
File Edit Tools Help
New Open... Save Reload referenced Sync embedded Undo Redo
Plan editor
Plan Specs Implementation Messages
Attributes Rich text editor
Edit all fields (Ctrl+D) Field profile: Default

```
APB_UART
  1. Verification Features
    1.1. Interfaces
      1.1.1. APB
        1.1.1.1. apb_transfer_cg
      1.1.2. UART
        1.1.2.1. Rx
        1.1.2.2. Tx
    1.2. Core Features
      1.2.1. Receiver
      1.2.2. Transmitter
      1.2.3. Fifos
        1.2.3.1. Rx Fifo
        1.2.3.2. Tx fifo
    1.3. Code Coverage
      1.3.1. uart_dut
  Milestone 1
  Milestone 2
  Milestone 3
```

(no search)
(no filter)
☐ Show excluded nodes
Edit all fields (Ctrl+D)
...UART/Verification Features/Interfaces/APB

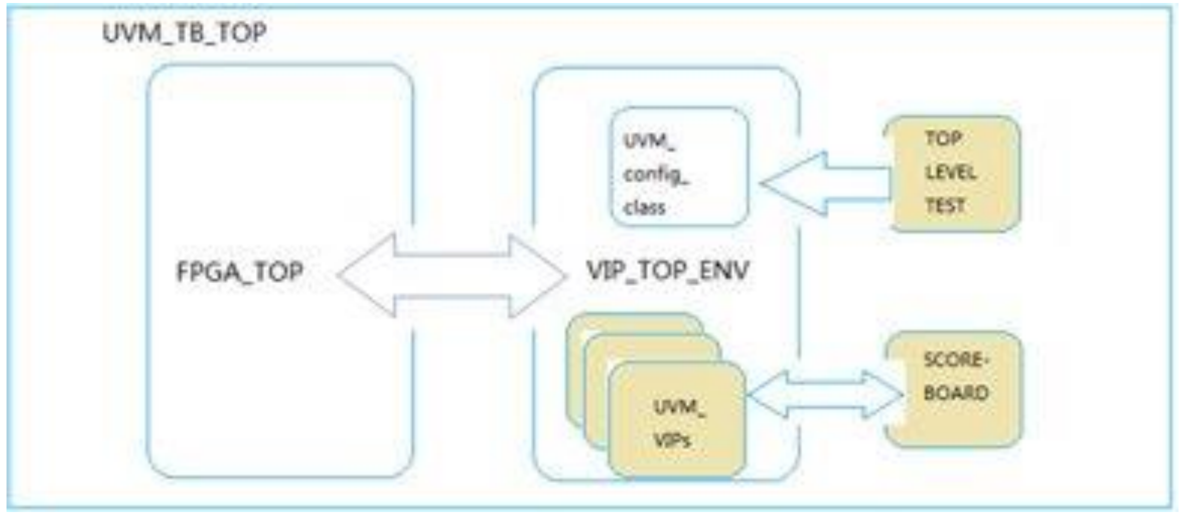
Section

Name	APB
At Least	1
Broken reason	
Details	APB Interface
Don't show in vManager	false
Element ID	[illegible]
Full Path	APB_UART/Verification Features/Interfaces/APB
Goal	100
Implementation Note	
Owner	
PDF spec location	
Parameter condition	
Previous Spec Text	
Priority	

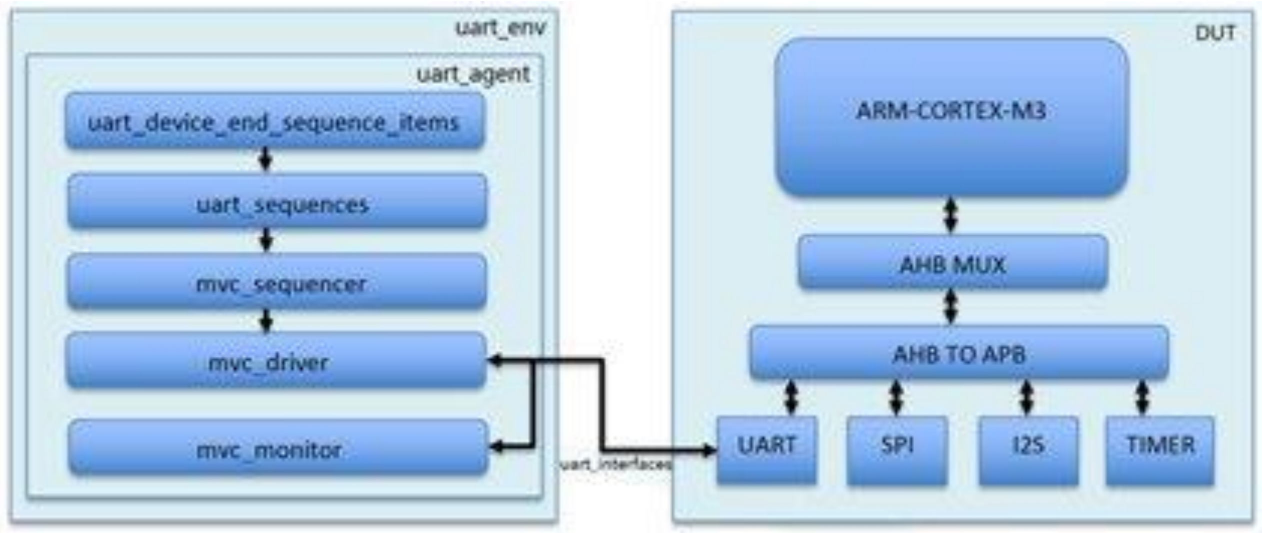
1 coverage element: 0 planned, 1 with mapping patterns
0 check elements: 0 planned, 0 with mapping patterns
0 testcase elements: 0 planned, 0 with mapping patterns

vPlanner feature for specific-purpose verification planning

Verification Architecture Development(1)



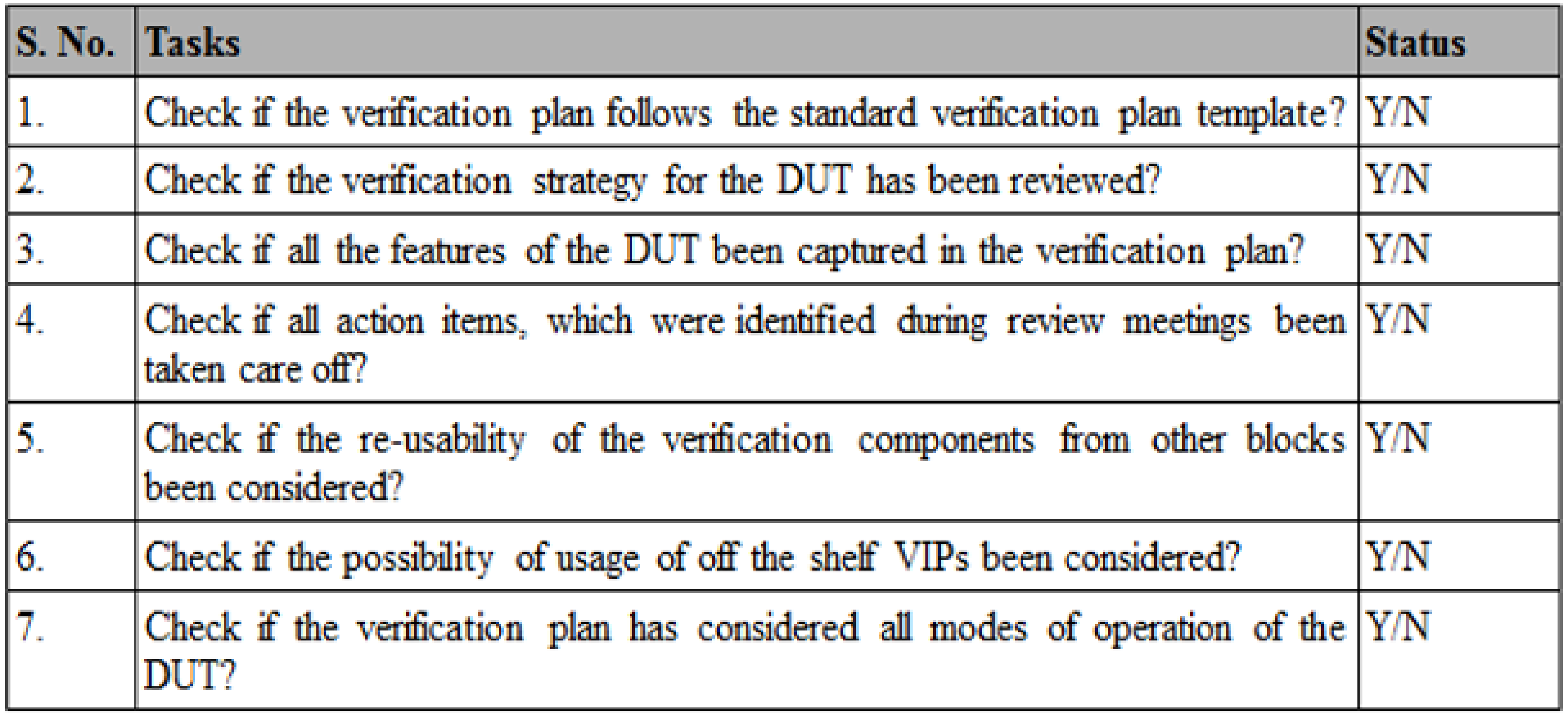
Verification Architecture Development(2)



Review Process of Planning Phase

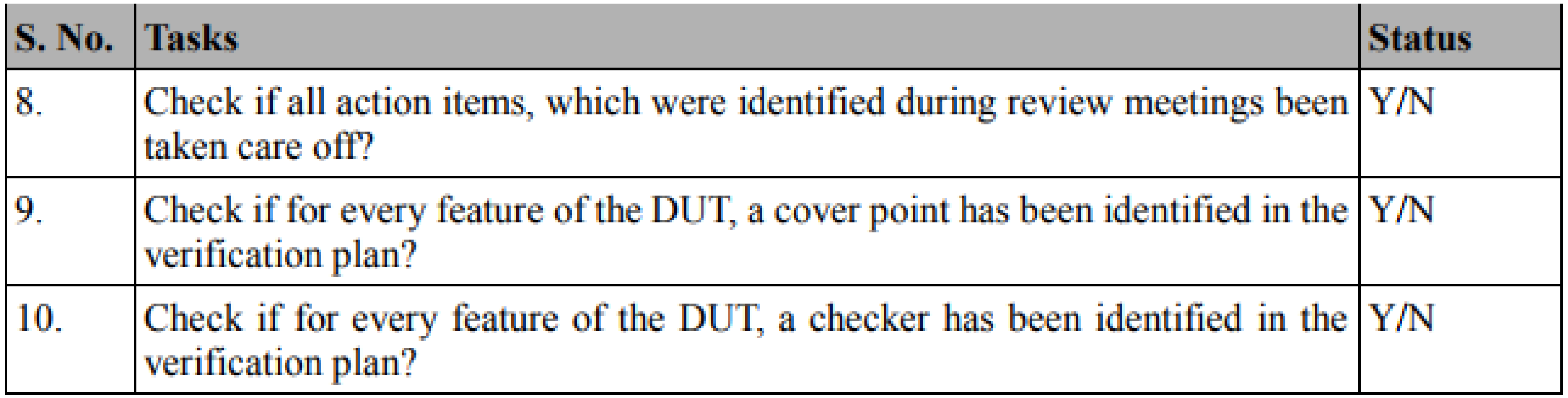
- Based on verification sign-off document.
- Sign off document lists a number of tasks that need to be completed to properly sign-off this phase and it is filled by stake holders.
- Architect, Project technical Lead, Verification engineers and Design engineer reviews sign-off document.
- First review is called by verification engineer verifying that block.
- Can be repeated a number of times to make sure all requirements of the sign-off document have been accomplished.

Planning Phase: Review Sign Off Tasks (1)



S. No.	Tasks	Status
1.	Check if the verification plan follows the standard verification plan template?	Y/N
2.	Check if the verification strategy for the DUT has been reviewed?	Y/N
3.	Check if all the features of the DUT been captured in the verification plan?	Y/N
4.	Check if all action items, which were identified during review meetings been taken care off?	Y/N
5.	Check if the re-usability of the verification components from other blocks been considered?	Y/N
6.	Check if the possibility of usage of off the shelf VIPs been considered?	Y/N
7.	Check if the verification plan has considered all modes of operation of the DUT?	Y/N

Planning Phase: Review Sign Off Tasks (2)



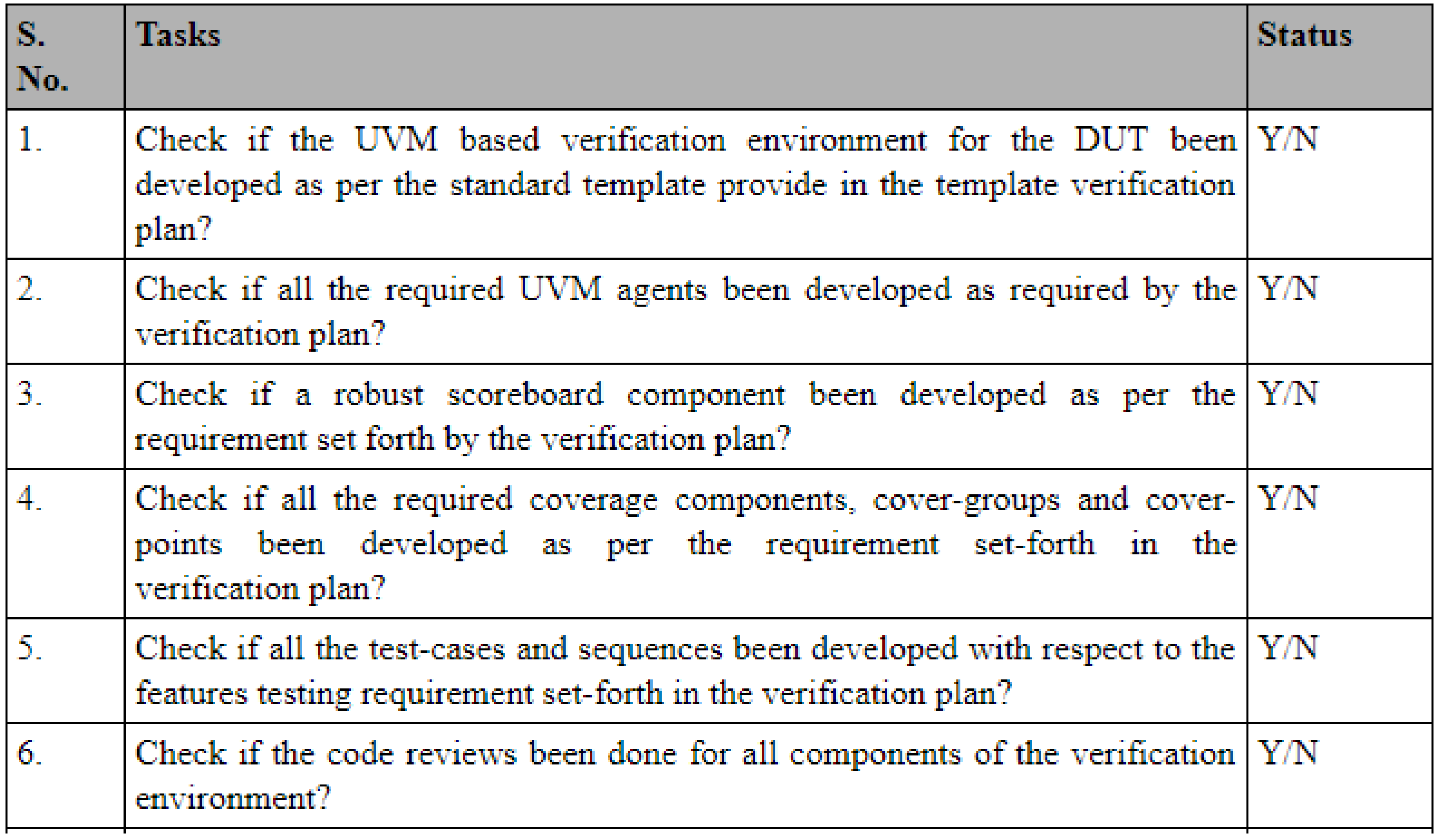
S. No.	Tasks	Status
8.	Check if all action items, which were identified during review meetings been taken care off?	Y/N
9.	Check if for every feature of the DUT, a cover point has been identified in the verification plan?	Y/N
10.	Check if for every feature of the DUT, a checker has been identified in the verification plan?	Y/N

Development Phase

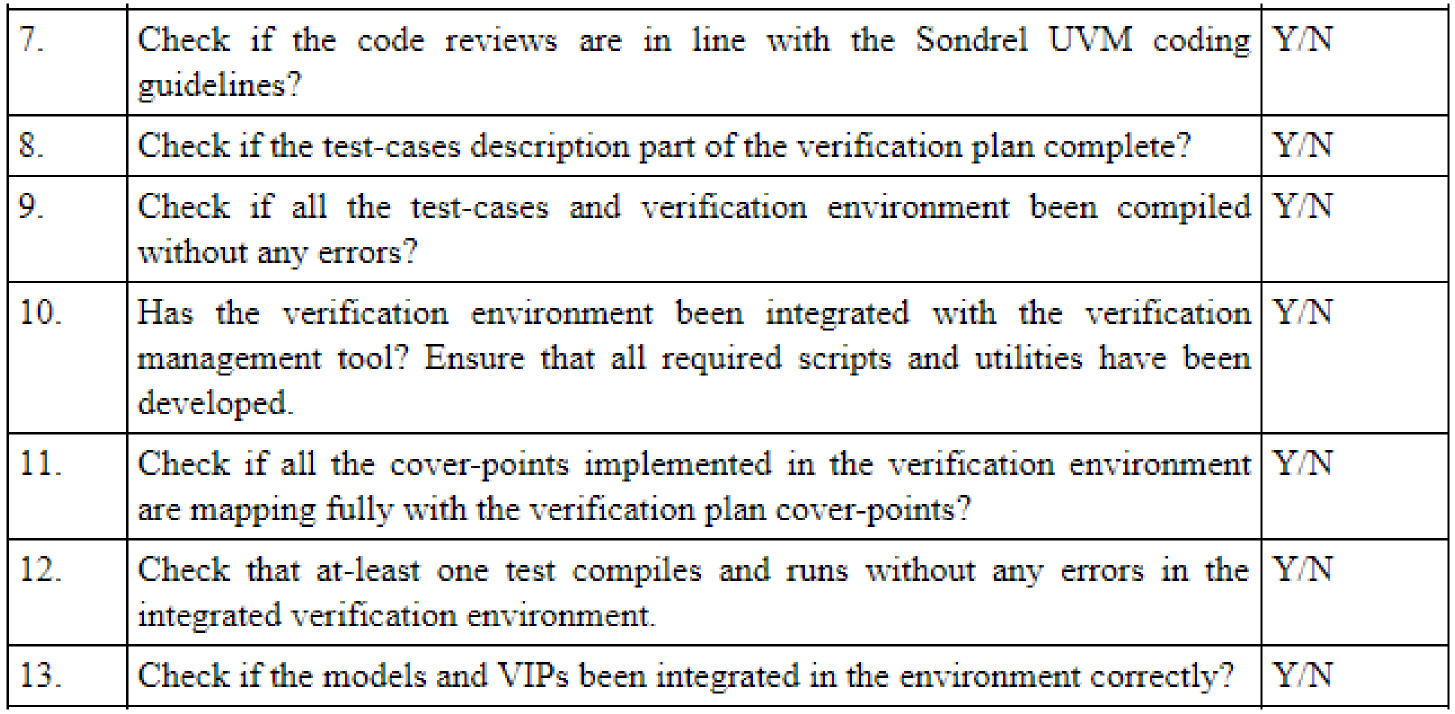
- Development of a number of all components as required by the UVM methodology.
- Improve testbench reuse, code portability, and create universal high quality verification IP.

Review Process of Development Phase

- Based on verification development sign-off document filled in by the relevant stakeholders.
- Architect, Project Technical Lead, Verification engineer and Design engineer are responsible for reviewing the development phase.
- Peer review from Verification Engineer is important.
- An early review is conducted to make sure that the environment is consistent with the flow in place.



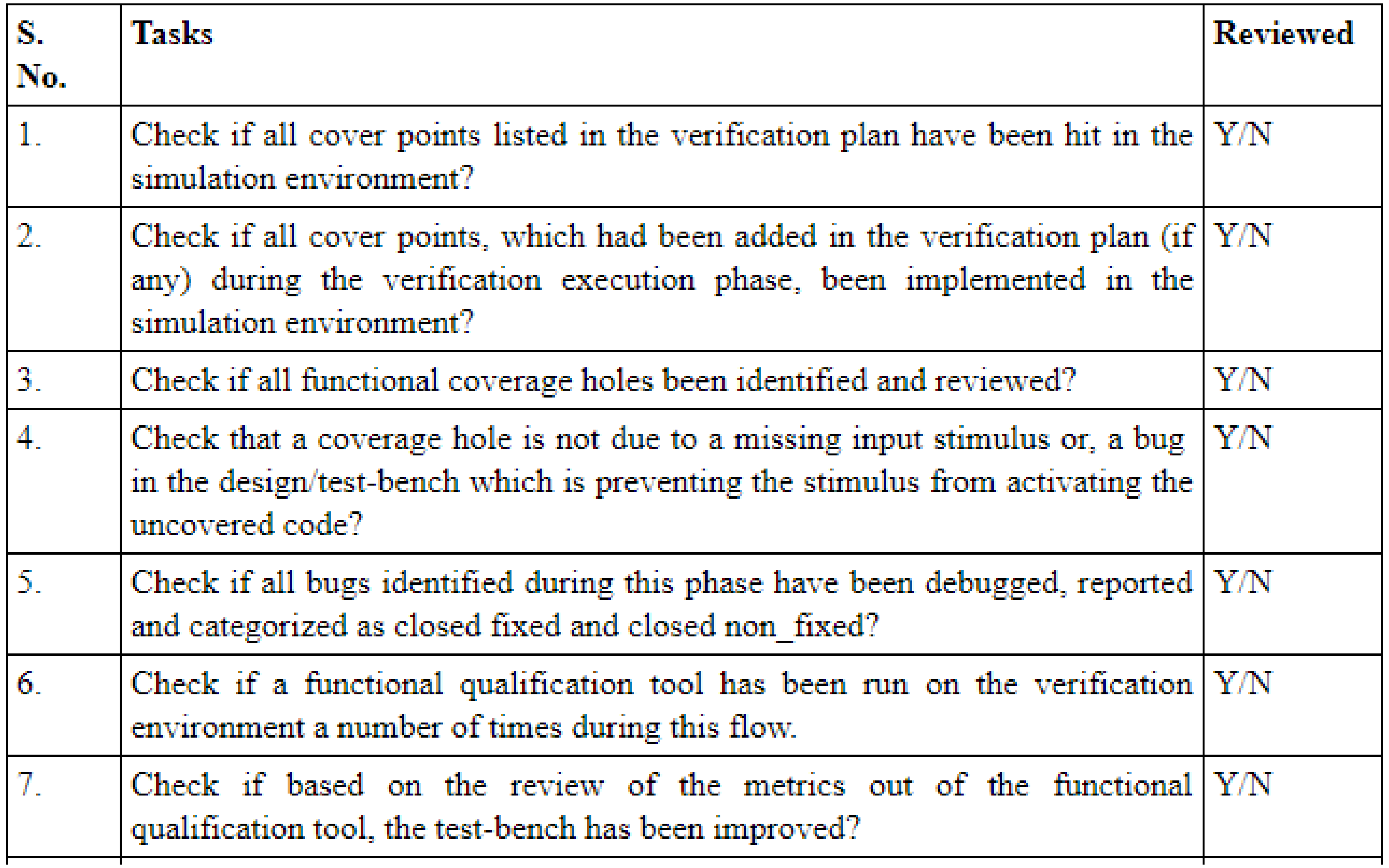
S. No.	Tasks	Status
1.	Check if the UVM based verification environment for the DUT been developed as per the standard template provide in the template verification plan?	Y/N
2.	Check if all the required UVM agents been developed as required by the verification plan?	Y/N
3.	Check if a robust scoreboard component been developed as per the requirement set forth by the verification plan?	Y/N
4.	Check if all the required coverage components, cover-groups and cover-points been developed as per the requirement set-forth in the verification plan?	Y/N
5.	Check if all the test-cases and sequences been developed with respect to the features testing requirement set-forth in the verification plan?	Y/N
6.	Check if the code reviews been done for all components of the verification environment?	Y/N



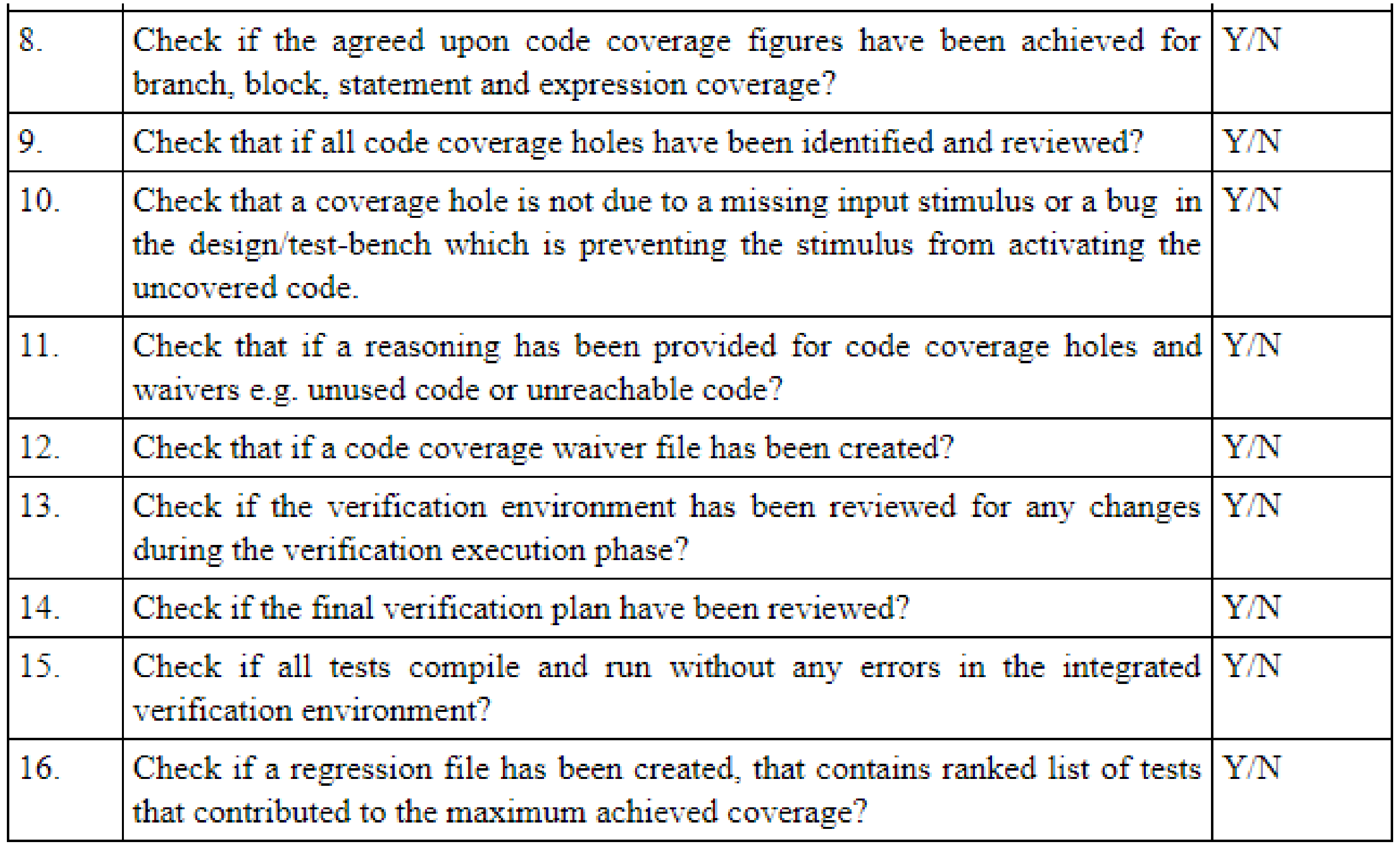
7.	Check if the code reviews are in line with the Sondrel UVM coding guidelines?	Y/N
8.	Check if the test-cases description part of the verification plan complete?	Y/N
9.	Check if all the test-cases and verification environment been compiled without any errors?	Y/N
10.	Has the verification environment been integrated with the verification management tool? Ensure that all required scripts and utilities have been developed.	Y/N
11.	Check if all the cover-points implemented in the verification environment are mapping fully with the verification plan cover-points?	Y/N
12.	Check that at-least one test compiles and runs without any errors in the integrated verification environment.	Y/N
13.	Check if the models and VIPs been integrated in the environment correctly?	Y/N

Review Process of Execution Phase

- It is based on verification execution sign-off document which is filled in by the relevant stake holders.
- Review if all coverage goals are met
- Review if all SV implemented cover points are in sync with the verification plan.
- Multiple reviews can be conducted to make sure all required tasks are accomplished.



S. No.	Tasks	Reviewed
1.	Check if all cover points listed in the verification plan have been hit in the simulation environment?	Y/N
2.	Check if all cover points, which had been added in the verification plan (if any) during the verification execution phase, been implemented in the simulation environment?	Y/N
3.	Check if all functional coverage holes been identified and reviewed?	Y/N
4.	Check that a coverage hole is not due to a missing input stimulus or, a bug in the design/test-bench which is preventing the stimulus from activating the uncovered code?	Y/N
5.	Check if all bugs identified during this phase have been debugged, reported and categorized as closed fixed and closed non_fixed?	Y/N
6.	Check if a functional qualification tool has been run on the verification environment a number of times during this flow.	Y/N
7.	Check if based on the review of the metrics out of the functional qualification tool, the test-bench has been improved?	Y/N



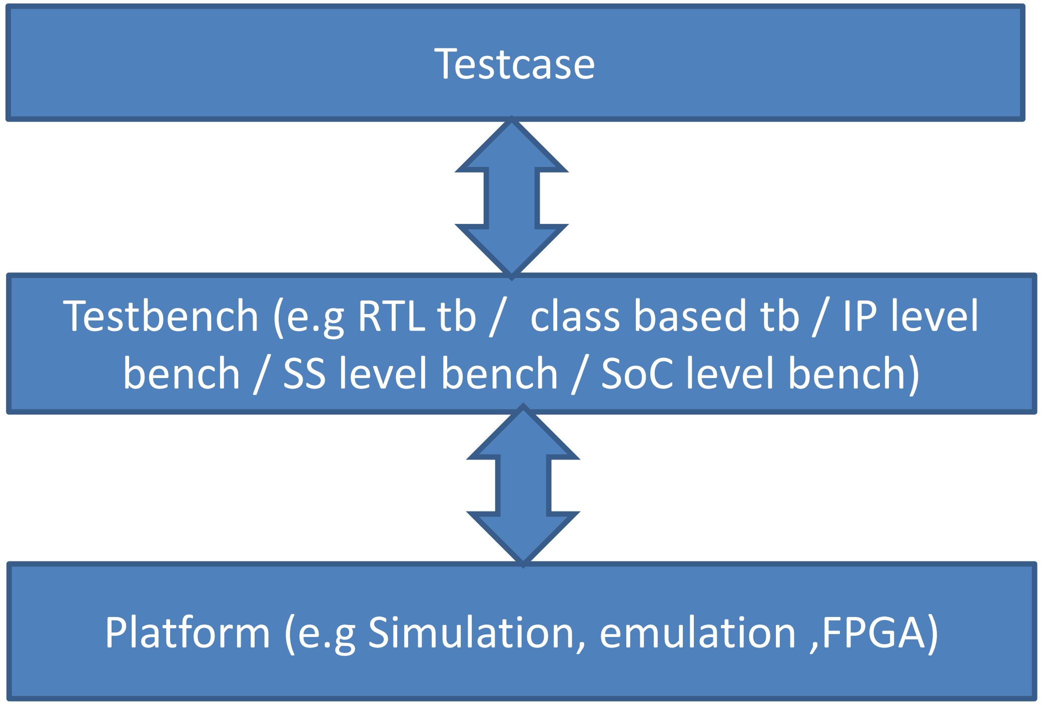
8.	Check if the agreed upon code coverage figures have been achieved for branch, block, statement and expression coverage?	Y/N
9.	Check that if all code coverage holes have been identified and reviewed?	Y/N
10.	Check that a coverage hole is not due to a missing input stimulus or a bug in the design/test-bench which is preventing the stimulus from activating the uncovered code.	Y/N
11.	Check that if a reasoning has been provided for code coverage holes and waivers e.g. unused code or unreachable code?	Y/N
12.	Check that if a code coverage waiver file has been created?	Y/N
13.	Check if the verification environment has been reviewed for any changes during the verification execution phase?	Y/N
14.	Check if the final verification plan have been reviewed?	Y/N
15.	Check if all tests compile and run without any errors in the integrated verification environment?	Y/N
16.	Check if a regression file has been created, that contains ranked list of tests that contributed to the maximum achieved coverage?	Y/N

Block Level Verification: Planning (1)

- Aims to target specified feature of the IP
- Ensure protocol compliancy as IP gets integrated to a larger system (e.g AMBA)
- Performance and maximum throughput testing
- Testcases categorized into:
  - Exhaustive IP level test
  - Reusable integration test

Block Level Verification: Planning (2)

- Reusable test structure


Testcase
Testbench (e.g RTL tb / class based tb / IP level bench / SS level bench / SoC level bench)
Platform (e.g Simulation, emulation ,FPGA)

Block Level Verification: Planning (3)

- IP level verification engineer identifies the list of testcases that are intended to be promoted to test the integration of the IP within a larger system. Does not necessarily cover key features.
- The functional testcases exploring the features of the IP and corner cases may/may not be reusable at system level as these testcases are beyond the scope of a system level verification and is most like be run only at IP level.

Block Level Verification: Planning (4)

- Verification engineers prepares a strategy document from the original IP's specification and uses it to develop the verification plan which will then be used to track progress of verification and sign off.
- If time and licenses are of constraints , the extreme and nominal operational modes will be targeted instead of a permutation of all possible usage scenarios.

Block Level Verification: Development (1)

Testcase Reusability

- The main thread that runs the simulation implements a list of "immutable tasks", e.g. clock setup, IP configuration , functional testing, reporting etc. These tasks wont change regardless of the platform or the testbench that the IP is used.
- The "immutable tasks" consists of a list of "interface tasks" whose behavior could change based on the platform/testbench running the test. (conceptually similar to Polymorphism in OOPs) E.g. a clock setup task at IP level is simply toggling a signal in the testbench, where as in a SoC this is more elaborate process like configuring PLL's etc. This could also vary between simulation , FPGA OR emulation

Block Level Verification: Development (2)

- Deploying SCEMI transactors within test-environment provides benefit of targeting various platform e.g. Simulation, FPGA and emulation at an earlier stage of project lifecycle with maximum code reuse.
- System Verilog assertions can be used to predict maximum latency thereby a potential system lock up can be caught.
- Add hooks within test sequences for error injection and recovery.

Block Level Verification: Development (3)

- All identified/planned item should have a place holder item within the verification plan and should by default report as 0% coverage (unless addressed).
- Use Covergroup's, SVA or other functional coverage metrics to report progress instead of testcase pass/fail figures

Block Level Verification: Execution (1)

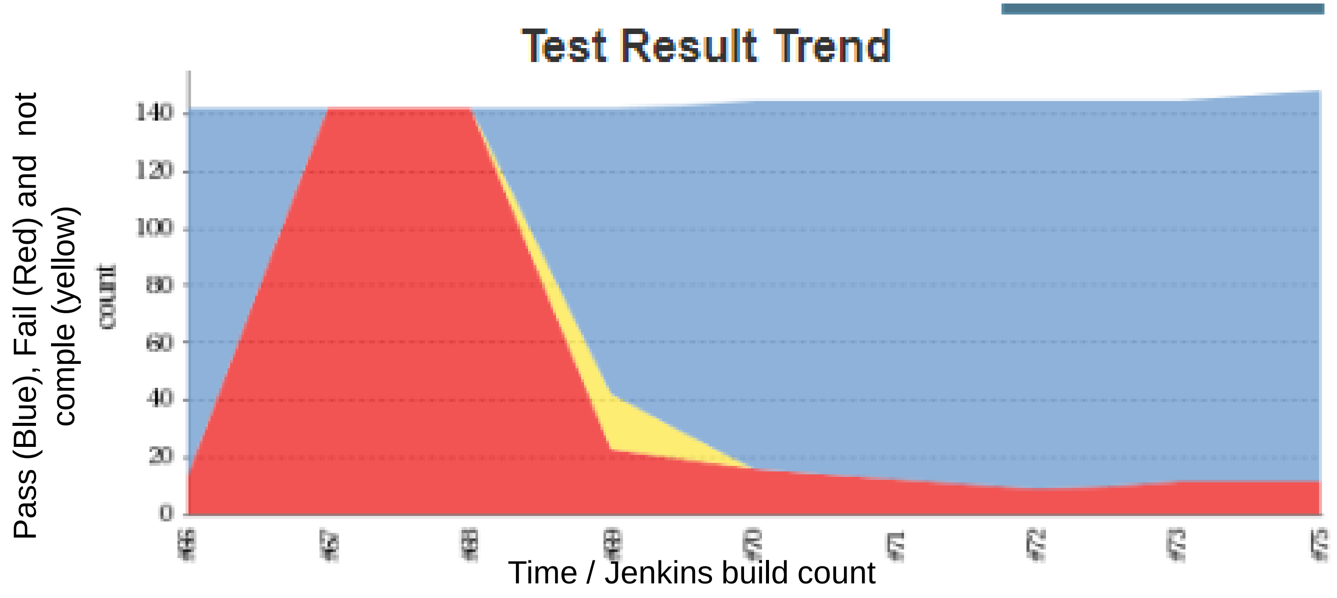
- Regular scrum meeting with Designer to discuss about progress, bugs, debug support etc.
- All bugs / deviation from spec gets raised via bug tracker.
- Using Continuous Integration (C.I) systems like Jenkins and using the results from C.I for reporting will offload these activities from verification engineers and will provide an unbiased progress of the project.
- The test plan, verification strategy document and the test environment has to be reviewed at an earlier stage in the project by more experienced engineer OR by Verification lead of the project.

Block Level Verification: Execution (2)

- A handover document should be maintained from an earlier stage of the project.
- The verification engineer should be notified of any updates to design.
- Pre-verified legacy modules used within IP i.e. FiFo , RAM's can be excluded from code coverage report (Block,Expression,FSM), although port toggle coverage of these components is required to prove integration of these IP's.

Block Level Verification: Execution (3)

- Test pass rate



Block Level Verification: Challenges and Lessons learnt(1)

- The example picked is an IP which decodes CMOS camera sensor data and streams to an ISP
- The scripts used for compiling, simulating and running regression were non-generic, hence adding an additional challenge for new engineers joining the project to learn and maintain a new flow. Generic scripts should be used in the future.

Block Level Verification: Challenges and Lessons learnt(2)

- The IP was purpose build for a given sensor model, later adding capability to support a variety of sensors. However the behavioral model of the CMOS sensor modelled only for the specific model of the sensor there by making it difficult to add support for other sensors types. Using a UVM agent as sensor BFM would have made this transition easier
- Intermediate points was created in C model to improve Debugability.

Subsystem Level Verification Requirements

- Functional correctness of overall application
- Interoperability of all the IP in the Subsystem
- Access to shared resources such as memory and bus structures
- Use case requirements
- Overall performance of the system
- Parallel external interface behaviour
- Connectivity of all blocks and sub-systems

Subsystem Level Verification Planning

- Planning phase includes preparing verification strategy in terms of Test plan, Coverage plan and Assertion plan.
- Verification of complex subsystems requires all micro level data to be collected at a common place for better tracking.
- Automating the whole verification tracking process is highly recommended

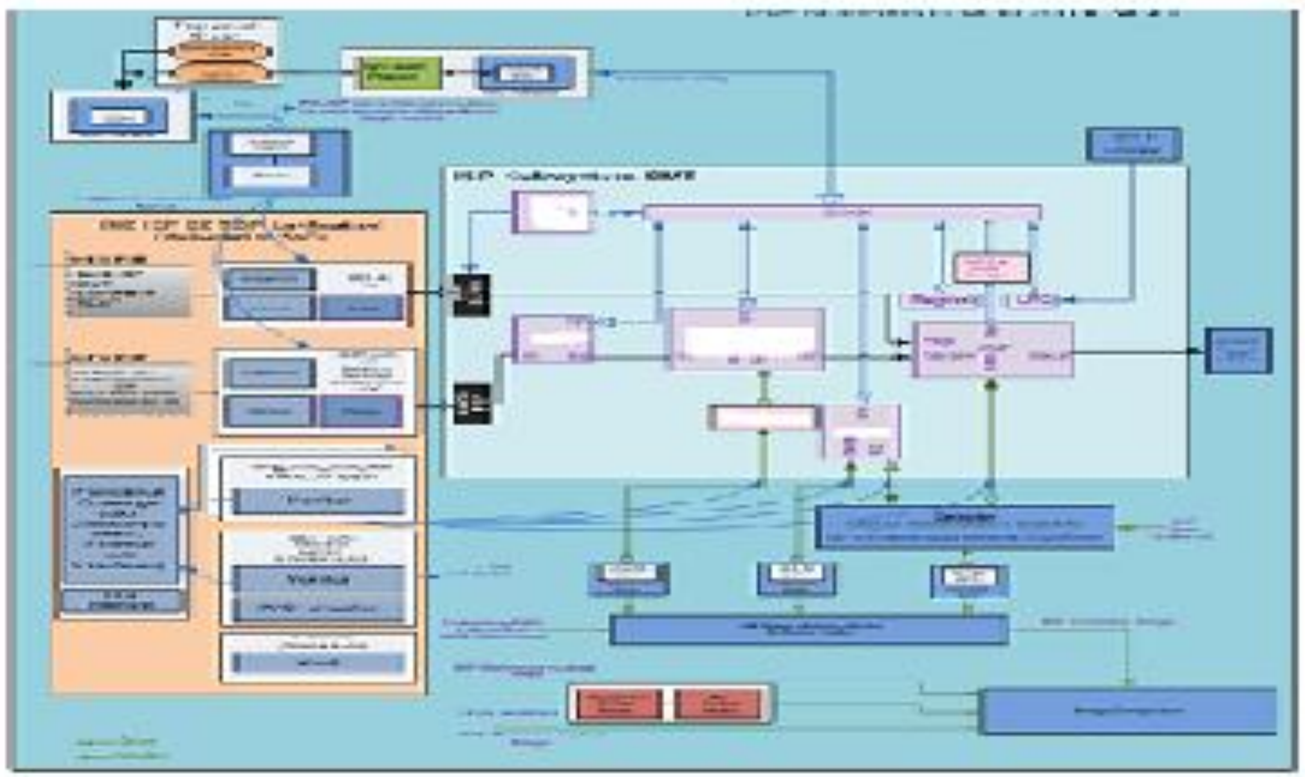
Verification Plan

- Example being used here is a Image Processing subsystem verification solution
- Verification was focussed on interconnectivity and interoperability solution
- USE CASE based testing and performance testing
- We have used Synopsys's Verification Planner
- While creating a plan we made sure to add a feature and description for every item and ensured that each lowest leaf feature has a measure with a correctly associated metric
- Based on the priority each measure is assigned a milestone like Alpha, Beta Final etc . These milestone corresponds to the quality of the RTL to be released

Testcase Re-usablity

- Testcase Promotion :
  - Some testcases are promoted from the IP Level to Subsystem Level
  - Some of the testcases are promoted from Subsystem Level to SoC Level
- Integration Tests and Functional Tests :
  - The Subsystem provides Integration tests which can be promoted at the Soc Level .These are C testcases which can be reused in SoC Level Environment
  - Functional testcases are present to check the basic functionality of the IPs

Representative Subsystem Block Diagram



Verification Environment Development (1)

- The environment comprised of SV test components, SV Assertions ,UVM Components and C test based infra structure to accomplish the Verification
- The generic components are BFMs for the bus interfaces like AXI4 interfaces which can be configured as masters or slaves.

Verification Environment Development(2)

- All the memory interfaces are connected to the AXI slave transactor which is configured in the Software , all the transactor components are synthesizable. This is done keeping in mind the portability to emulator.
- AXI UVC can be also be used in the environment. The verification environment contains a common SV memory for the Image Processing and the other intermediate block

Verification Environment Development (3)

- The Functional Coverage model had Write and Read latencies, use case coverage, Subsystem Address Coverage, Memory Range Coverage, and Clock Frequency Coverage.
- Toggle Coverage was collected at the top level.
- Regression setup and flow was implemented using internal flow. Weekly regressions were run and monitored.

Verification Environment - Reference Generation(1)

- The subsystem works on the pre-processed data.
- The pre-processed data is generated by using set of scripts that operate on the C Model which is called the Toolchain
- The toolchain splits the input image .dat files into nibbles and in the simulation this given to the Tx Driver to drive data onto the input interface.
- RTL output captured

Verification Environment- Reference Generation (2)

- Toolchain is expected to generate intermediate IP's output data and Meta data in *.ppm format, which will be used to compare
- The final output of the ISP Subsystem will be compared to that is generated by the toolchain .
- The pre-processing of the image data reduces the rather long simulation and debug cycles

Execution of Planned Verification

- The verification was mainly about the Sensor Rx's interoperability with the Image Signal Processing block.
- USECASE with different frame sizes of 2K, 4K & 8K were chosen to validate the overall working of the subsystem
- Toggle Coverage of the module interface and full coverage of glue logic was measured and monitored

Performance Analysis

- Performance Analysis was to make sure that the design will meet its targeted bandwidth and latency levels . We have verified
  - Minimum and Maximum read/write latencies using AXI UVCs.
  - Buffer fill rates
  - Data Frame rates

Scoreboard and Checkers

- The Subsystem does not have a scoreboard. The reference image and the output image from the RTL are compared using the scripts
- Relevant checkers are present at all the interfaces

Verification Sign Off Criteria

- All Integration, Usecases, Performance tests pass
- Toggle Coverage at the interfaces
- Full Code Coverage of the glue logic
- Bug Trend Analysis

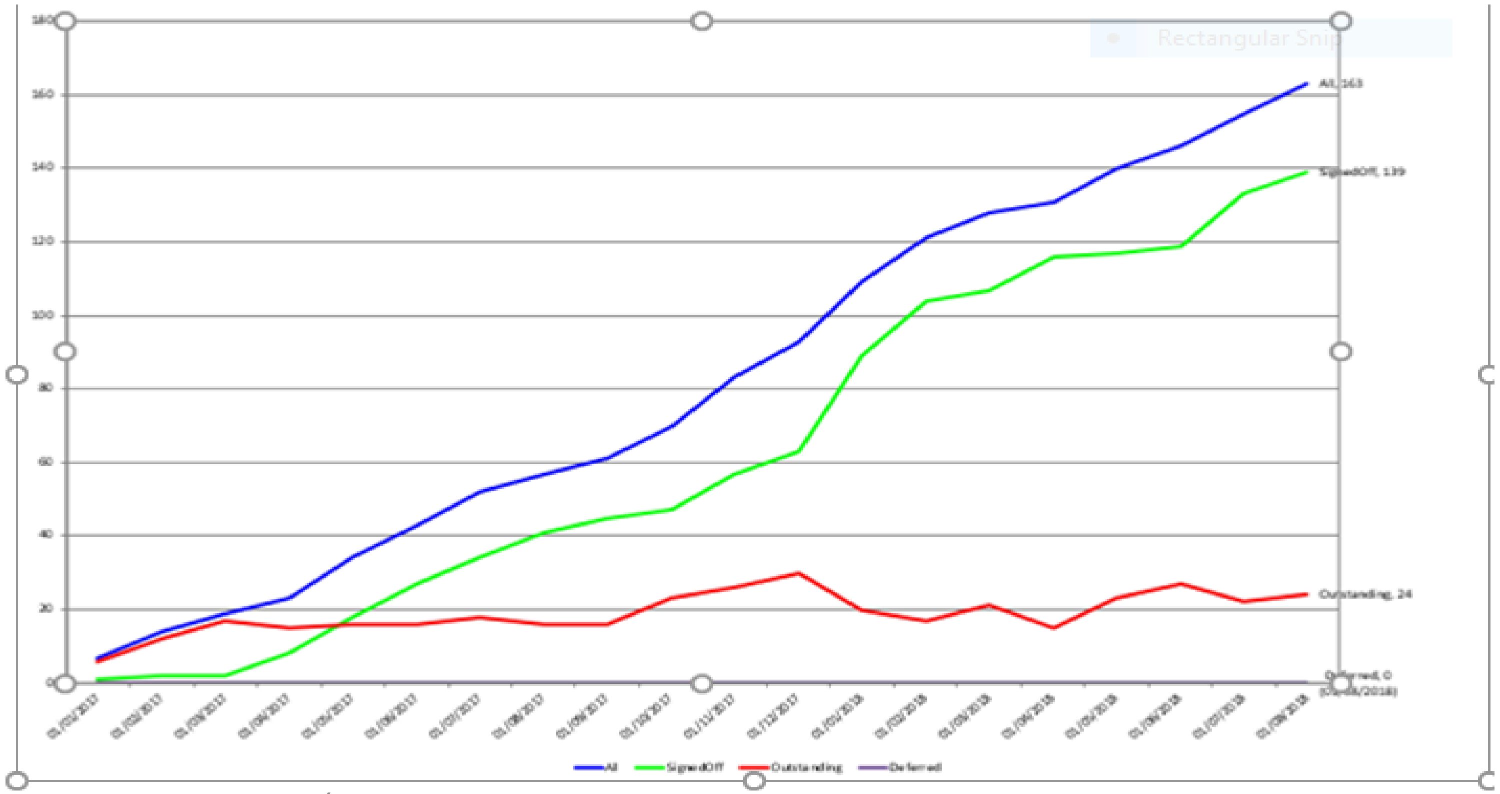
Verification Progress

- Listing the examples of metrics used to determine the verification status/progress

Categories	Total	Pass	Weightage	Pass %	Notes
Usecases	12	9	5	75.00	
Integration	33	33	1	100.00	
Performance	14	7	1	50.00	
Internal Block	12	3.5	1	29.16	
Error/Corner	12	0	1	0.0	
Code Coverage	100	98.45	1	98.45	

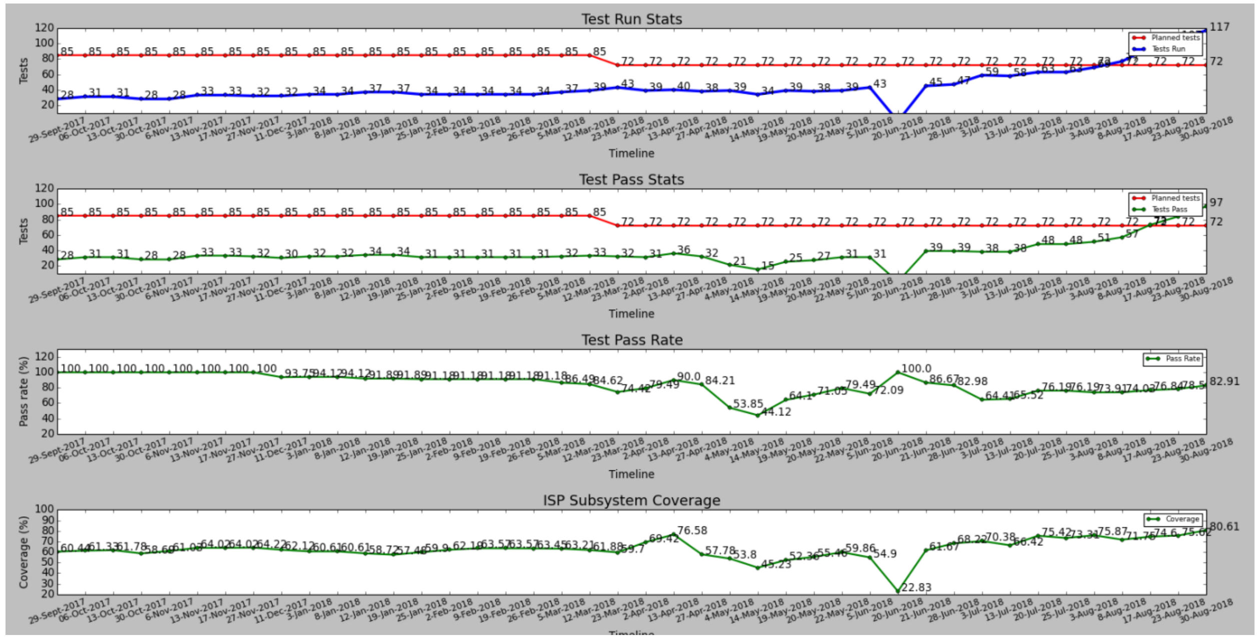
Verification Status

- Bug Trend Analysis over the period of verification cycle is shown below, on the X axis is the days (dates) and on the Y axis is number of issues filed. Blue indicates total number of issues, green is closed issues and red is outstanding issues



Verification Status

- Test Pass Rate : This graph explains the test run stats, test pass stats, test pass rate and cumulative coverage(functional + code).



Subsystem Level Verification: Challenges and Lessons learnt (1)

- Huge Simulation Time:
  - Planned and developed debug points along the datapath
  - Reference image was generated as the part of pre-processing step
  - Comparison of the image and POLL checks were done as a part of post-processing

Subsystem Level Verification: Challenges and Lessons learnt (2)

- Performance Testing: Started early on in the Verification Cycle to detect the system level issues.
- Improve Debug :
  - Short frames with reduced height were used
  - Add debug register bits and debug signals to improve verification and root cause detection.

Subsystem Level Verification: Challenges and Lessons learnt (3)

- Planning Phase Review:
  - The review of Vplan led us to correct some of the measures, which in-turn helped the coverage numbers
  - After the detailed review we had to add new features to some of the existing re-usable verification components.

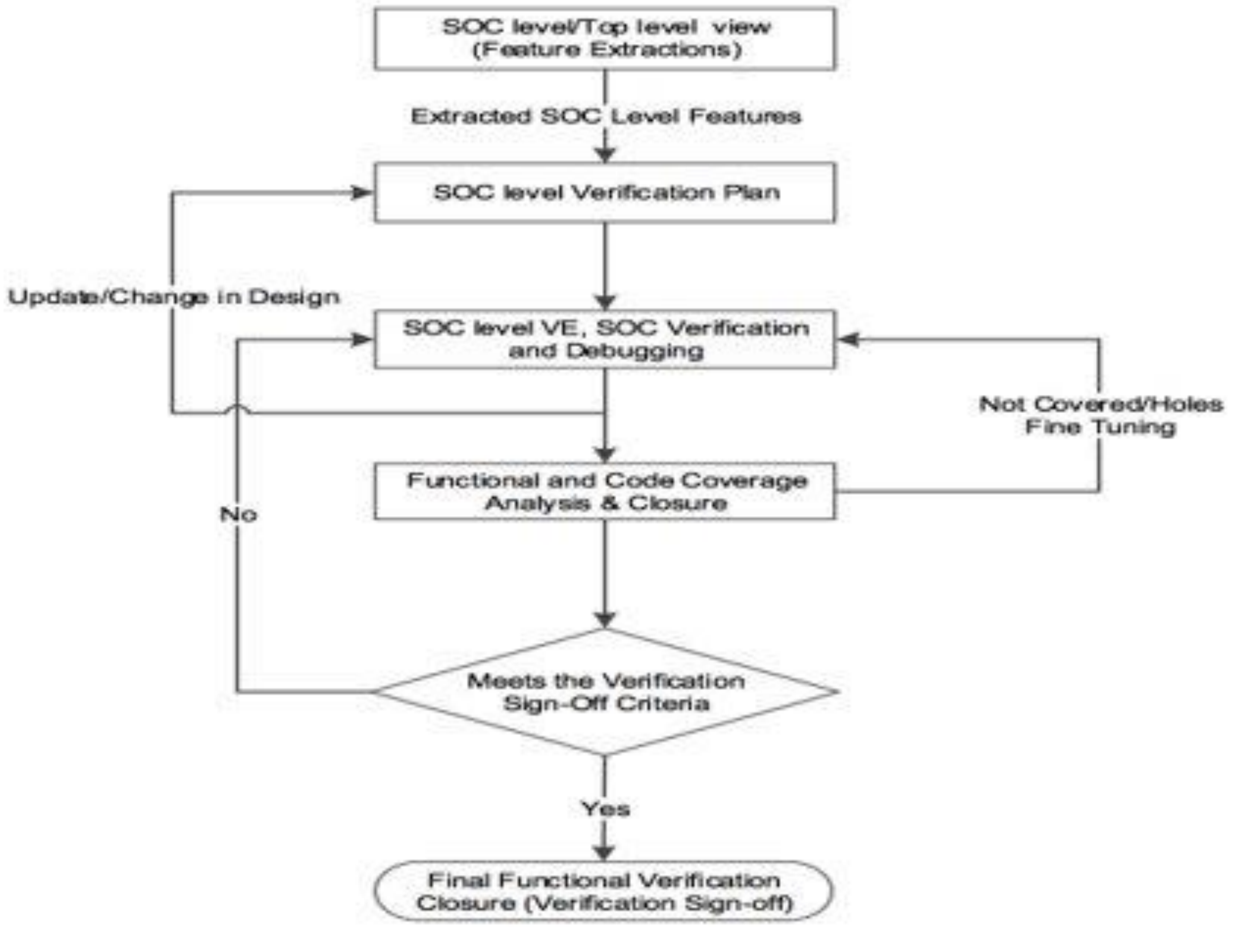
SOC Level Verification

- An SOC has one or more processors (CPUs, GPUs) at the heart of it . All other logic constituting the system are around it. (peripherals, memory/controllers, network etc)
- A processor executes a software program (and multiple layers in real world) in terms of instructions. It will always fetch instructions from the memory and execute them which will trigger the various logic around it in the SOC

SOC Level Verification

- Most of SOC level tests are in a high level language like C. It need not be strictly C, but could be in other languages which all finally translates to the correct assembly code and an object file that can be loaded in to the memory.
- Python/Perl also popular to generate SOC level test cases.
- Replace the processor and some subsystems of SOC with behavioural models then for that verification you can use SV/UVM.

SOC Level Verification Flow


SOC level/Top level view (Feature Extractions)
Extracted SOC Level Features
SOC level Verification Plan
Update/Change in Design
SOC level VE, SOC Verification and Debugging
Not Covered/Holes Fine Tuning
Functional and Code Coverage Analysis & Closure
No
Meets the Verification Sign-Off Criteria
Yes
Final Functional Verification Closure (Verification Sign-off)

Top Level Verification Requirements

- Functional correctness of overall application
- Interaction of all module and sub-systems
- Access to shared resources such as memory and bus structures
- Operation with realistic clock and power domain behaviour
- Overall performance of the system
- Parallel external interface behaviour
- Connectivity of all blocks and sub-systems

Vertical, Horizontal & Diagonal Reuse

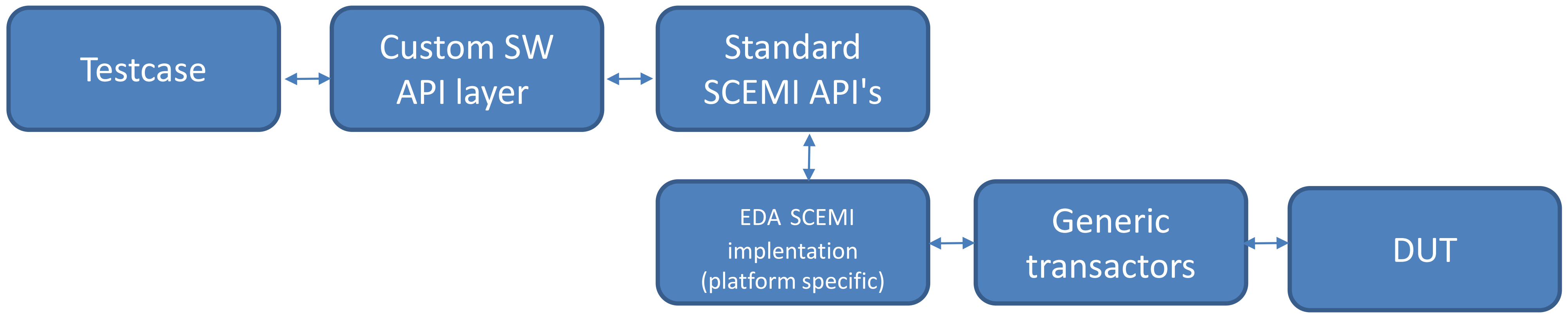
- Horizontal reuse – one SOC to derivatives
- Horizontal typically means using a verification component in a different system or project but at roughly the same level of abstraction and with the same functional role.
- Vertical reuse – IP to SOC
- Vertical reuse means using a verification component in a different hierarchy level, usually with an implied change of role.
- Diagonal reuse – various level of abstractions(Simulation, Emulation, FPGA, Post Silicon)

Vertical, Horizontal & Diagonal Reuse Advantages

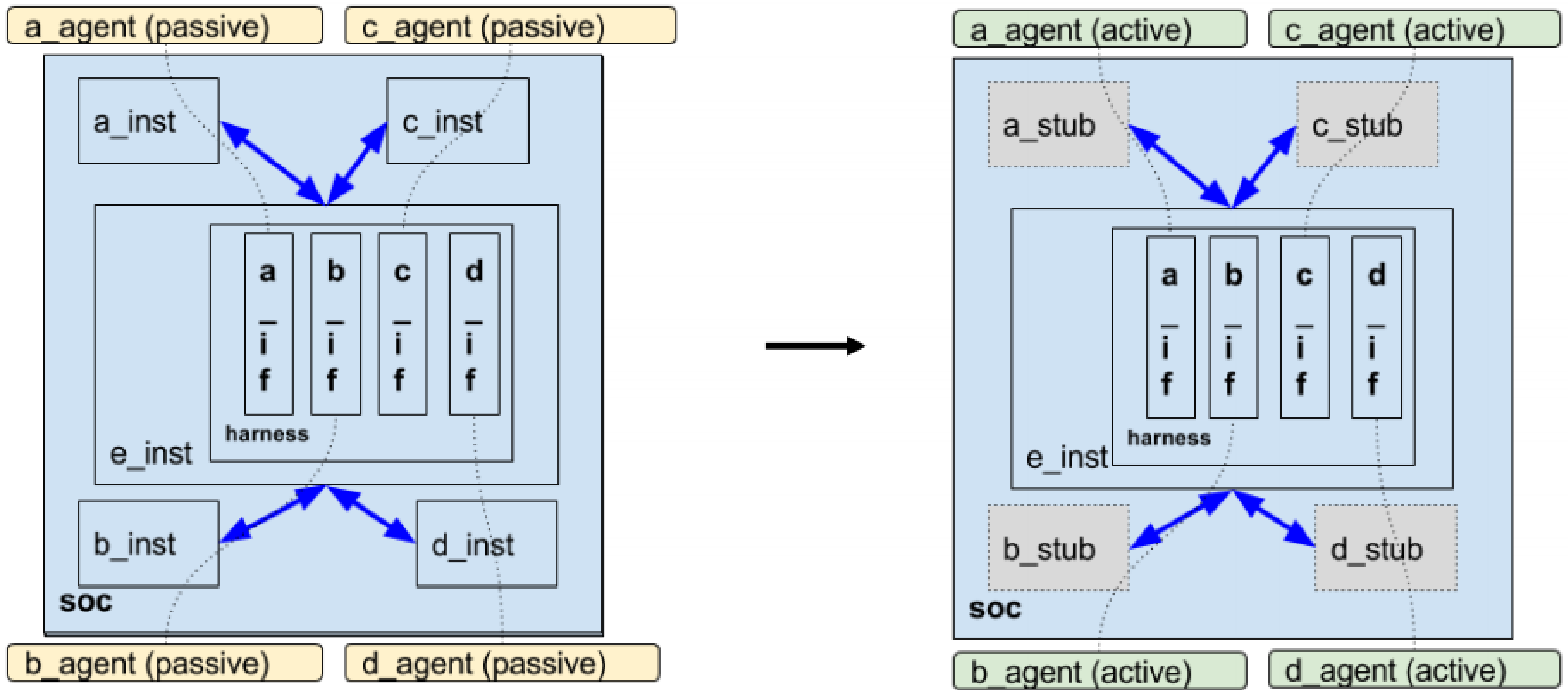
- Additional checks, stimulus, coverage and messages
- Detecting more bugs
- Helps speed up debug for other system-level defects by providing improved internal visibility and enhanced bug isolation
- Huge Time and Cost savings.
- Reuse of stable code and improve confidence in interoperability of protocols like AMBA, USB, PCIe

Generic Transactors

- Promotes resuse across Platforms (simulation,emulation,FPGA e.t.c) and between Block, Subsystem and SOC level testbenches



Block level verification in System level TB



SVP: UVM <==> C communication (1)

- We can read/write any register or memory location which can be accessible by Proc (C code), using SV.
- We can pass valuable information back and forth from Proc to UVM Env and vice versa, to achieve synchronization between the two. This will be very useful in generating complex SoC scenarios.

SVP: UVM <==> C communication (2)

Alternative Methods

- Assuming that your C/ASM is running on a CPU inside the SoC, a common technique is to have some GPIO ports on the SoC connected to "registers" in the testbench. Your BFM would be triggered by the data written into the registers when the C/ASM test wants to control the BFM
- Monitor to SRAM access generate different events.

SOC Level Verification: Planning (1)

- Usecase scenarios need to be discussed with all stake holders and mapped
- Performance/Error/Corner case scenarios need to be discussed with all stake holders and mapped.
- Performance features like maximum and minimum latency at Interconnect/NOC/DDR interface needs to be considered
- Buffer fill rate at various stages help to track the performance and identify any system issues.
- Maximum clock rates at various levels needs to be discussed and frozen.

SOC Level Verification: Planning (2)

- Simulation time has to be considered while planning Usecases
- Debug ability has to be considered while planning initial tests and order of execution.
- Area of concerns like new module developed has to be stressed first.
- Module/Subystem/Clocks/Resets integration needs to be considered
- Reusability of testbench components/C Testcases/VIPs/Sequences/Checkers from Subsystem/module level has to be considered
- Bootup modes and short Boot up modes needs to be planned.

SOC Level Verification: Development

- The environment comprised of VIPs, SV test components, SV Assertions, UVM Components and C test based infrastructure to accomplish the Verification goal.
- The generic components are BFMs for the bus interfaces like AXI4 interfaces which can be configurable as masters or slaves.
- All the development is carried out using the best practice rules.

SOC Level Verification: Execution

- The execution phase included running the created Use cases, debugging and collecting the coverage.
- Performance Analysis was carried out by running huge stimulus with minimum and maximum latencies at Interconnect/NOC/DDR interfaces.

SOC Level Verification: Execution

- The execution phase included running the created Use cases, debugging and collecting the coverage.
- Performance Analysis was carried out by running huge stimulus with minimum and maximum latencies at Interconnect/NOC/DDR interfaces.

SOC Level Verification: Challenges and Lessons Learnt (1)

- Debug:
  - Short frames with reduced height has been used
  - Planned and developed debug points along the Datapath
  - Added debug register bits and debug signals to improve verification and quick root cause detection.
- Huge Simulation Time:
  - Reference image was generated as the part of pre-processing step
  - Comparison of the image and POLL checks were done as a part of post-processing

SOC Level Verification: Challenges and Lessons Learnt (2)

- Performance testing was started early to detect system level issues early in life cycle.
- VIP integration/bringup from multiple vendors
- C System testcases integration with VIP.
- Toggle coverage at IP interface was difficult with Abstraction of testcases being high.
- Glue logic for integration needs full coverage and was difficult with higher level abstraction tests.

Verification Economics

- Verification plan is always done with constraints of available time, budget and changing requirements.
- Focus on most important features to be verified.
- If Derivative chips, Prioritize verification of code which has changed.
- Use VIPs when possible to reuse the stability(of previous working silicon), interoperability(like USB, PCIE standards) factors of VIP
- Use the stable UVCs when possible to leverage the previous projects code/experience.
- Verification estimates are always dependent on the quality of Specification and RTL

Thank You

Any Questions?