White Paper

Delivering Functional Verification Engagements

A Proven, Systematic and Assured Approach

July 2015

Author

Subhranil Deb
Sr. Design Consultant, Synopsys India Pvt. Ltd.

Introduction

With the advent of smarter and higher performing devices, there has been a tremendous increase in design complexity. Driven by new high-end hardware feature and intelligent software requirements, these devices are comprised of multi-core processors and a multitude of interface IP, memory and other analog circuitry, communicating via many different interface protocols. This poses a huge challenge in terms of scoping, resource planning and delivering a bug-free design. To address these technical challenges (as depicted in Figure 1) within budget and schedule, the emphasis on first-pass silicon success is paramount – which calls for a thoroughly verified design. To transform an optimal specification to a fully verified system, you need a rich package of industry-proven tools and reliable methodologies combined with an experienced team of verification specialists.

This white paper provides insight into the challenges and need for a robust functional verification infrastructure, plus a means to achieve closure by leveraging Synopsys' suite of high-end tools and applications. Additionally, best practices followed by Synopsys Professional Services consultants to deliver advanced SoC/ASIC verification solutions to leading customers are explained.

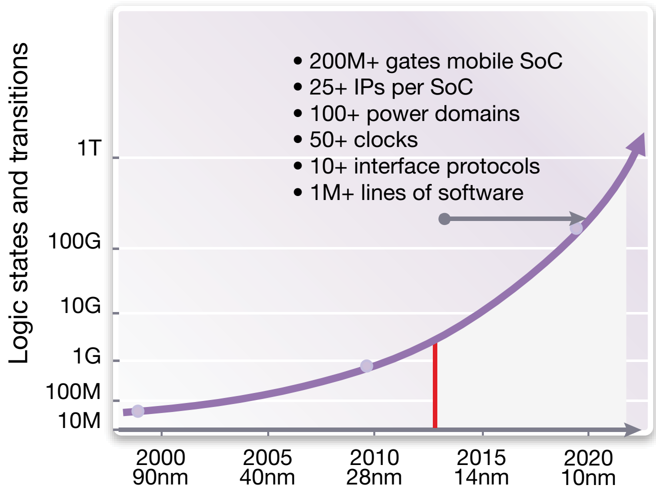

Figure 1: Verification challenge as process technology advances and design complexity increases. Courtesy: Synopsys

Increasing Need for Functional Verification

Consumer demand for new features and functionality, higher performance and lower power are key factors in driving the evolution of devices. As demand increases, so does competition, putting immense pressure on companies to be first to market. First-silicon success has become imperative, but achieving it is extremely difficult. Failure can be disastrous, resulting in the need for design re-spin. The impact of design re-spin is increased cost and risk of missing the market window versus competition.

According to IBS, at 28-nm the price of each mask set is between two and three million dollars and five and eight million dollars for 22-nm/20-nm. Studies also suggest that sixty percent of ASIC/IC respins are attributed to “logical/functional failures” and “specification changes”. Finding and fixing these issues earlier in the design process is less costly as shown in Figure 2.

So how do you ensure the quality and compliance of the ASIC to the specification? Though architecting and developing a design is fundamental, validating correctness by testing the design against the specification is critical. The earlier you identify a problem, the easier it is to fix. The use of pre-verified design blocks and verification IP coupled with industry proven verification tools and techniques can be a game-changer. In the following sections, we outline some of the key functional verification challenges design teams face, explain the phases of verification and share Synopsys’ verification consultant recommendations and best practices, followed by two case studies.

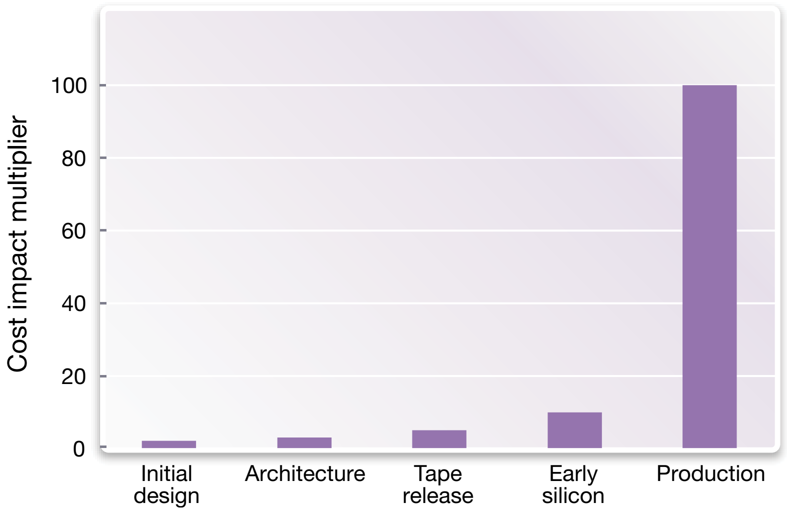

Figure 2: Cost impact of bugs at different phases of execution

Functional Verification – Challenges

The key to delivering a successful ASIC is to ensure that the intent of the product (specification or requirement) and the final deliverable (design) converge. This effort to verify design convergence is often significant, therefore it is imperative to develop a strategy that is both scalable and reusable. Important elements of this strategy that must be addressed are:

Domain Expertise

Verification is a perfect example where expertise is inversely proportional to the time required to validate a design, hence there is no substitute for experience. The work is manual and how a testbench is configured varies with schedule, technical and system requirements. Understanding design priorities are essential. Most designs use multiple interfaces and interleaved bus interconnects. Experts are needed to understand and look at the design from the system point of view to tackle various protocol and performance nuances during the project. As we learn later, development and debug are two areas that need specialist attention to identify the issues, determine whether they are design or testbench related and then resolve them. This has been a challenge for engineering teams to find the right balance of specialists and tools versus cost. Apart from protocol and verification techniques, there are other areas including low power and hardware/software simulation that need domain expertise, but these are not addressed herein.

Planning

Determining the extent of verification required and the time it takes to verify a design per schedule stipulated is difficult. Unless it is well-planned and executed, the likelihood of failure is high. A thorough upfront understanding of the specification is necessary to develop the testbench and testcase plan. Essential aspects of planning involves estimation of effort based on testbench methodology and functionality verified, identification and assignment of resources, risk assessment, contingency formulation and securing stakeholder support. With better planning, issues that could potentially derail the project are mitigated and operational efficiencies can be realized.

Simulation and Debug

Simulation and debug times can be a significant contributory factor in project uncertainty – so much that you can get bogged down debugging for days without making much headway. It is important to have the right blend of debug and testbench expertise as well as design knowledge. Debug can become a critical bottleneck because of poor design or quality of the testbench. The later bugs are found in the design and verification process, the longer it takes to fix – so the mantra is to find bugs as early as possible in the project. Simulation run-time needs to be managed well. Turnaround time due to debug coupled with longer regression time can dramatically affect the delivery schedule. Finally, as constraint random verification becomes more prevalent, it's important to balance simulation time versus test case intent. Focus should be on getting maximum yield with shorter simulation cycles.

Verification Convergence or Closure

When is verification complete? Is the design compliant with the specification and requirements? How well can we measure it? Defining the right sign-off criteria for a design is extremely important. Functional and structural coverage provide reliable statistics, but any verification engineer will tell you that substantial unaccounted for effort is spent in verifying the last few requirements. To achieve convergence, sometimes new tests are added, out-of-the-way techniques are used or we add them as exclusions. The balance one needs to strike between verification intent versus time to rigorously verify the design must be taken under consideration.

Verification Phases – Concepts and Recommendations

The verification process is largely task-based so it is important to follow a structured approach. This verification process is divided into smaller phases as shown in Figure 3. As you become more familiar with these phases, you might conclude some steps are redundant or unnecessary for a particular engagement, but never the less should be followed to maintain verification efficiency and effectiveness. Some phases can be executed in parallel but care needs to be taken regarding dependencies. The input to the verification effort is the design specification that must be verified to achieve sign-off, a set of mutually agreed criteria defining the verification coverage and feature goals.

Study

The study phase is where design knowledge is gathered via reading specifications, standards, other documents and discussion with all stakeholders. The purpose is to come up with an execution strategy and initial estimation of effort – this is where domain experts are needed who can foresee all the issues. This helps in budgeting optimum time and resource. It is a challenge to estimate the complexity of a design in the beginning when details are not fully scoped. Getting the team involved early enables better planning and provides time for the team to ramp up on any new protocol or domain. The other outcome of this phase is Feature Extraction. As engineers read through the documents they extract every feature and classify into categories such as “Requirement ID”, “Feature Description”, “Closure Metric”, etc. The extraction of negative features, errors and exceptions is also important. The objective is to create a feature list and test plan that helps to gauge testbench complexity and guide the verification effort through the phases.

Planning

The absence of a well thought out and documented plan can lead to disastrous consequences. The key ingredient for success is experience. You need experienced engineers planning your verification project. Planning is divided into two key components, resource estimation and verification planning.

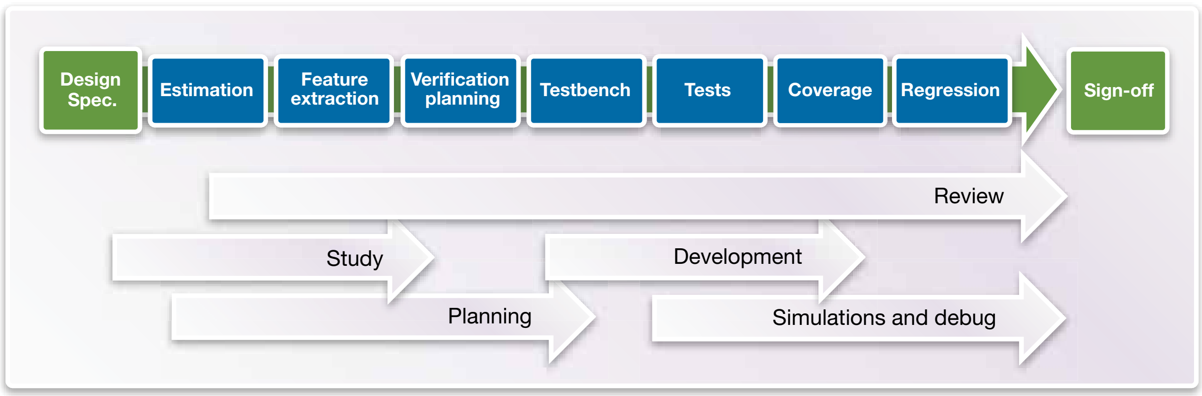

Figure 3: Typical functional verification phases

Estimation and alignment of resources to develop the project schedule and then execute on plan depends on prior experience and tape-out requirements. Meeting schedule milestones is not the only goal, quality is equally if not more important. Hence planning requires one to understand the design intricacies, loopholes and dependencies to clearly map out a plan. Key things to be included in planning are assumptions made, risk mitigation and tasks. It should highlight dependencies as design deliverables that gate the verification effort and be broken down into short term goals and deliverables that stakeholders can use to more easily assess progress against plan. It also helps to have a test plan with coverage annotation that enables you to track where you are in the verification process and provide insight into what features have been verified. Finally, the verification plan should be a living document that is updated on an ongoing basis.

The other component is planning the verification architecture as a framework is devised to verify all the features of the design (under verification or DUV) in a thorough and efficient manner. The architecture should map out a detailed data flow how the DUV is handled and manipulated in the testbench. Coverage closure strategies are also planned during this phase. The extra time invested upfront to develop this plan will result in a higher quality design, delivered on time.

Synopsys Verdi® provides innovative planning and coverage technology integrated across all debug views, enabling users to quickly analyze and cross-probe holes identified through coverage analysis. It provides the ability to quickly generate verification plans and link them to specification and requirements documents to ensure that projects stay on track as high-level requirements change. Integration with a regression management solution enables users to execute verification tasks, view results and triage data to meet coverage goals and achieve verification closure more quickly. Verdi also supports integrated visualization of results across logic simulation, static checking, formal verification, verification IP (VIP) and FPGA-based prototyping solutions to deliver a unified view of verification closure.

Synopsys VCS® Execution Manager provides an integrated management solution that helps balance tool execution and techniques that drive projects toward closure. Execution Manager provides a web-based user interface that includes test filtering, on-the-fly visibility, real-time regression monitoring, debug triage and regression trend charting that improves time to coverage closure and enhances the ability of geographically diverse teams to coordinate their efforts and accurately estimate schedules. It also provides an extensible infrastructure allowing home-grown scripting to be integrated.

Development

Planning is followed by development – implementation and execution of the plan. One of the longest phases in duration, it includes testbench and test development. These tasks typically done in parallel include development or reuse of testbench components hooked into the top environment. Significant manual work is involved, but following a standard methodology framework can accelerate testbench bring-up.

Based on requirements, the verification architect needs to analyze and select the appropriate verification technique, either formal, emulation, C-based or constraint random. The focus of this paper is on the constraint random approach. A winning verification strategy should include a structured methodology that exploits the capabilities of industry-leading tools and standards. UVM with its vast set base classes, factory classes and macros should be used extensively to develop the testbench and reduce development time drastically. The UVM configuration database is used to pass variables and parameters hierarchically, instead of through methods to improve reusability. Agents should be made active or passive depending on whether they are used to drive stimulus or only monitor them. Try to restrict randomization in a single configuration object during the build phase to help scenario re-creation based on seeds. When there are multiple agents in a system, use virtual sequencer to streamline data and make user testcases easier to code. Handle resets at a later time but keep provisions for resets in the components to avoid re-tuning of otherwise stable components. To avoid confusion, a verification engineer should follow the specification and not the design. Interpretation of specification grey areas need to be discussed with the team and plan of action mutually agreed to.

Often testbench and design development coincide. In such situations, verifying the testbench flow with an "Empty DUV" setup can expose testbench errors early. "Empty DUV" setup is like having the full-testbench wired or connected directly without the DUV. A driver connects to its peer (or receive) monitor and likewise for all interfaces. When tests are simulated, it can pipe-clean the components and also help develop the automated scoreboard. Figure 4 highlights an example Empty DUV setup for early testbench flow verification, thus enabling actual verification of the design to start immediately once it becomes available.

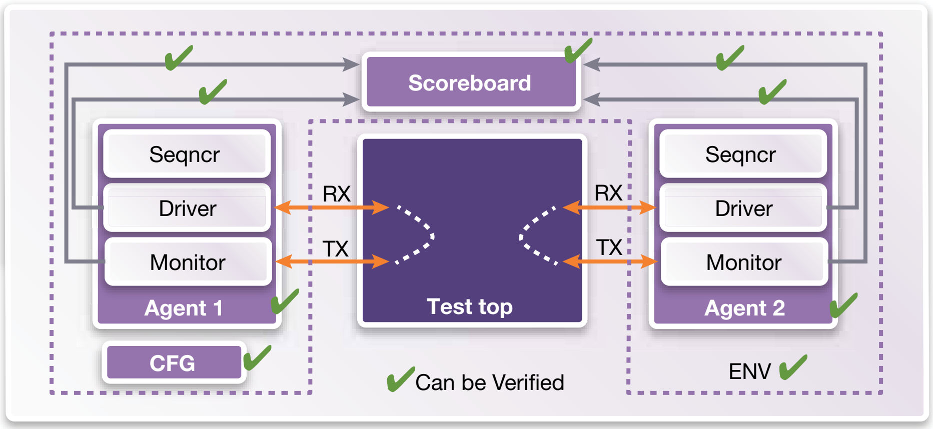

Figure 4: Example Empty DUV setup

One of the common scoreboard challenges is when it tries to perform cycle accurate predictions to match the output. The data is sometimes early, sometimes late, but in both cases seen as a testbench error. In such cases, one approach is to have multiple references of expected data. The data in current cycle and previous cycle can be compared with the received data and decision can be made. The idea is to increase tolerance to compensate for cycle dependency. While implementing such checker models, it is important to ensure bad data is not passed through.

It is good practice to identify basic tests to pipe-clean the DUV and then gradually try out more complex test cases. Try to reuse the testcases as much as possible. Combine or modularize common flows like initialization, basic configurations and checking. You might also consider development of a self-checking automated scoreboard. From the stimulus side, use the configuration class to randomize and efficiently hit the target zone. Judiciously select what should be part of the configuration class and what is part of the transaction class. By now, the test plan should have been reviewed and synced with the latest specification and therefore should be the guide during this step. A good way to choose and prioritize test cases is:

1. **Basic Tests** comprised of minimum checks intended for flow pipe-cleaning. These tests should ensure data flow and configuration access is working. Such tests are used by designers in early bring-up of the RTL.
2. **Complex and Critical Tests** intended to expose critical errors. These are the non-trivial tests which potentially will fail the design. Checking for such scenarios should be automated and started as early as possible in the verification flow.
3. **Random Tests** to improve confidence in completeness of verification at the expense of more time to setup. Start random testing once checkers are in place. Tune the constraints and gradually expand the coverage of the tests. Having coverage enabled while running random tests can help with gauging progress.
4. **Error and Exception Tests** comprised of the resets, interrupts and negative scenarios that can potentially fail the design. These tests should generally be handled with callbacks, but sometimes might need exceptions in monitoring and checking. Also these tests should always check for recovery from exception conditions.

As tests are created and more random seeds are run, it becomes important to start tracking coverage. Coverage groups should be implemented before creation of random tests. Coverage for random stimulus should be captured for passing seeds only and cover bins should be efficiently set to capture relevant values. Care needs to be taken when cover bins are crossed as it can incorrectly increase the coverage target, therefore identify scenarios and create cover bins accordingly. These cover groups may be reused later if some manual tweaks or exclusions are required.

Synopsys' Certitude® comes in handy if you want to check the quality of the verification environment. It is different from functional coverage, helping to expose the quality and effectiveness of the testbench. It is difficult to measure how good your verification environment is. Certitude works by inserting faults in the design and if that causes a passing test to fail, it tells you the testbench is robust enough to track design issues, otherwise you need to re-check your testbench.

Simulation and Debug

Simulation starts as soon as the testbench is setup. Setting up scripts for simulation and subsequent regression is the fundamental task apart from monitoring and generating coverage numbers. As Figure 5 shows, more than a third of the time spent in verification is in debug, and if we add coverage it is more than half.

The challenge of simulation is not running tests but rather managing those runs and debugging simulation errors. Simulation scripts should support regressions, capture coverage numbers, report fails and facilitate debug logs and waveforms. While running regression, it is good practice to define a small suite of sanity check tests which should be executed before making any change to the testbench. Similarly, split regressions into smaller sets and trigger it with time-gap. The random test when iterated should hit all intended corners otherwise you may need to take another look at the constraints defined.

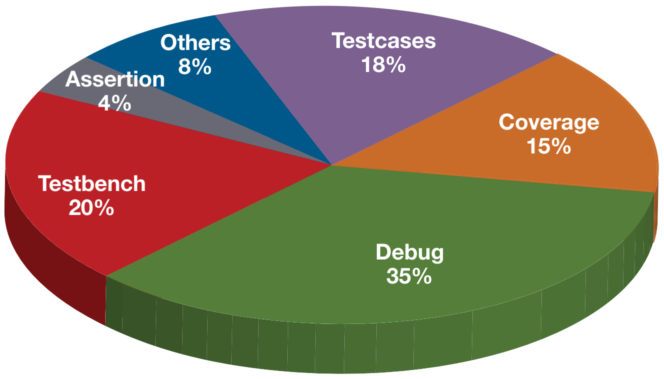

Figure 5: Time spent by type of verification. Courtesy: Synopsys, Inc.

Debugging is about finding the cause of the failure – it can be a testbench or design issue. Except for design issues, the onus lies on the verification team to debug. Design issues sometimes trace back to a poorly defined testbench when bad configuration or data is sent inadvertently. The biggest bottleneck in identifying an issue is availability of information or the lack thereof at that particular level of abstraction. This is further amplified if debug expertise is limited or missing. Significant effort is spent in debugging and resolving testbench issues to behaviorally mimic the design architecture. With well-architected testbench components, you should be able to identify most causes of errors and fails from generated testbench log files. On the top-level, encrypted codes, gate-level simulations and C models present bigger visibility and traceability challenges. In these instances, a good understanding of the design and testbench is needed to correlate error messages to the actual defect.

Synopsys' Verdi3™ is an automated debug solution for debugging digital designs with powerful technology that helps you comprehend complex and unfamiliar design behavior, automate difficult and tedious debug processes and unify diverse and complicated design environments. It has been shown to typically cut debug time more than fifty percent. It provides transaction-based debug with flexible transaction and message support for debugging and analyzing designs at higher levels of abstraction as well as assertion-based debug. Your SystemVerilog testbench can be debugged with full source code and libraries from UVM, while also enabling UVM-aware verification results exploration.

Once regressions are mostly passing, it is time to check the coverage numbers. Planned coverage goals and cover groups are implemented during the development phase. Tracking coverage numbers early won't yield much if substantial number tests or features are not completed. Analyzing the coverage numbers and finding the holes in coverage is necessary to determine if some corner scenarios have been missed to ascertain if more tests are needed. Sometimes it is easy to create a directed test to hit a particular corner. Complete functional coverage is a necessity, while structural coverage exclusions are an optional consideration post design review. One-hundred percent functional coverage and passing regression tests should be the absolute minimum sign-off criterion.

Review

Periodic reviews are essential to track progress. It is important to involve the right set of people in reviews to be most constructive. From the initial estimations, to verification planning, to test extraction, to coverage closure, every deliverable and dependency should be reviewed. Reviews can potentially expose holes in verification strategy, missing features and helping to prioritize work. Decision makers and other key stakeholders gain useful insight into project status and issues that require escalation to fix as early as possible during verification to keep the project on schedule.

Case Study 1

This design was based on an image sensor application comprised of two back-to-back modules that supported three incoming sensor protocols mapped into a periodic data stream which were controlled by configuration registers. The configuration space was relatively large with interdependent variables. The prime objective was to completely architect and verify the ASIC with one-hundred percent functional coverage.

The verification team ramped up on the different protocols and created the initial estimates and schedules for discussion and finalization after the team understood protocol complexity and initial design architecture. Following this, the team worked on the design specification and sensor standards with all stakeholders to create the feature list. The feature list was a living document on which the testbench was architected. The idea was to stimulate the DUV with different sensor information packed into transactions from the active agents (AGENT1 and AGENT2) and monitor the data at different interfaces based on the configuration done through AGENT3. The testbench architecture was scalable, enabling top-level reuse. The same testbench was reused for verification of both design (DUV1 and DUV2) blocks. Figure 6 shows the testbench architecture with data flow across the testbench. The passive agent (AGENT4) snoops the traffic between DUV1 and DUV2 and sends the packet to corresponding scoreboards, DUV1 output for DATA SB1 and DUV2 input for DATA SB2. A single configuration class was used to setup the testbench, with all random variables controlling the design registers and testbench components.

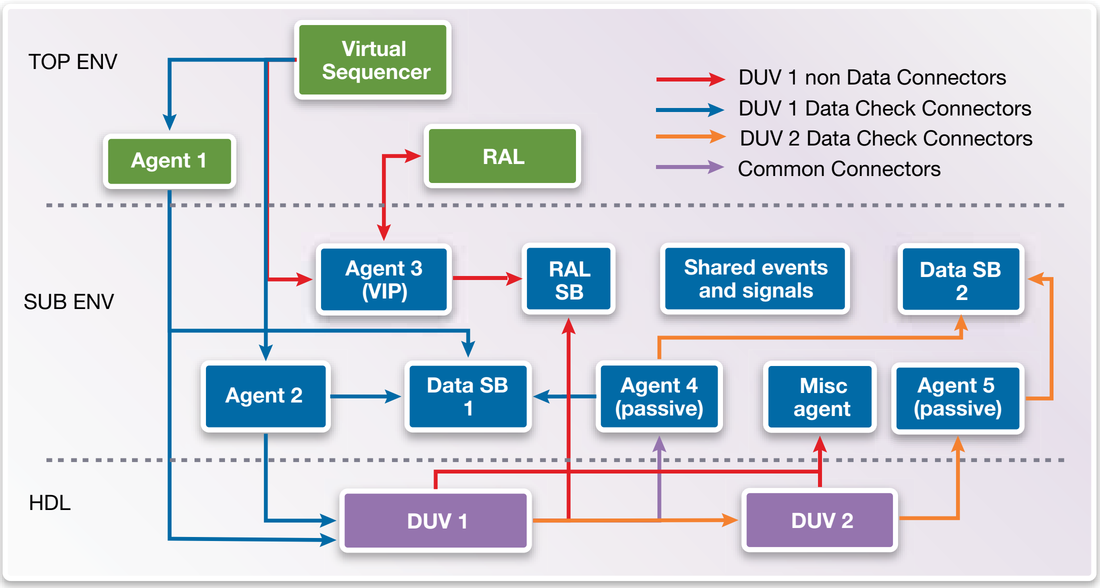

Figure 6: Block-level testbench architecture

The major verification challenges were:

Evolving Specification

The design was based on the standard protocols which were suggested by the verification team as well, so there was overlap in the development phase. Because of this, testbench components were aligned to the actual standards and not to the design specification when it was released. This caused substantial rework and patches as the project progressed. Some important lessons learned were:

- Special care needs to be taken about feature ambiguity. All features should be reviewed with the system architect to ensure no inconsistent, unintended, overlapping or unclear features. This may not be a simple task. For example, you may have feature A plus feature B that somehow violates or invalidates feature Z, as described in a different chapter or specification.
- When a standard protocol is designed, the stimulus should follow the standard even if the design is a subset of the standard. However, the monitor should align with the design requirements.
- Sometimes it helps to overhaul logic early in the project rather than to make incremental changes or quick-fixes.
- Testbench architecture should include dependencies on multiple clock frequencies or a range of frequencies initially, as this can have implications on testbench synchronization.

Verification Closure

Once the design specification was released, the verification team started extracting features using this specification and all other specifications including industry standard protocols. With each subsequent revision to the design specification, extracted features would need to be modified. The challenge was to plan schedule closure with the ever growing number of features. Because of time to market pressure, some of the new features were excluded after approval from customer and design team to focus on the functionally critical features. Priorities were assigned based on complexity of the scenario. During the last phase, error scenarios and exceptions were verified using random configuration, leading to identification of more bugs in both the design and testbench. With pumped-up regression, corner cases were hit, debugged and resolved. Important things that worked leading to higher confidence of verification closure were:

- Assumptions and verification goals set at the beginning of project were done accurately after understanding the design intricacies, setting the right tone for project execution
- Planning was done thoroughly and accurately, resulting in desired milestones being achieved in a time-bound manner
- Regressions were run early enough to stress the design and detect design issue requiring fixes
- Tests were made generic to cover more scenarios through constraint random configuration
- Tests were not overcomplicated. For example if two errors need not coexist, they were randomly enabled making it easy for the testbench to predict.

Figure 7 highlights project estimates and execution phases. The project spanned close to twelve months including initial ramp-up, vacations and ramp-down. The team size grew when new protocols were added as reflected by more man-days in month three. The second module was drafted in month five and testbench started for it. After month six following stabilization of the testbench, primary focus shifted to feature closure. Around months nine and ten, the design was updated with more features thus increasing testbench work. Figure 8 shows the rate at which features were closed, once they were identified by end of month five. Features were closed in month eleven followed by three to four weeks spent on design regression with huge random traffic to meet coverage goals.

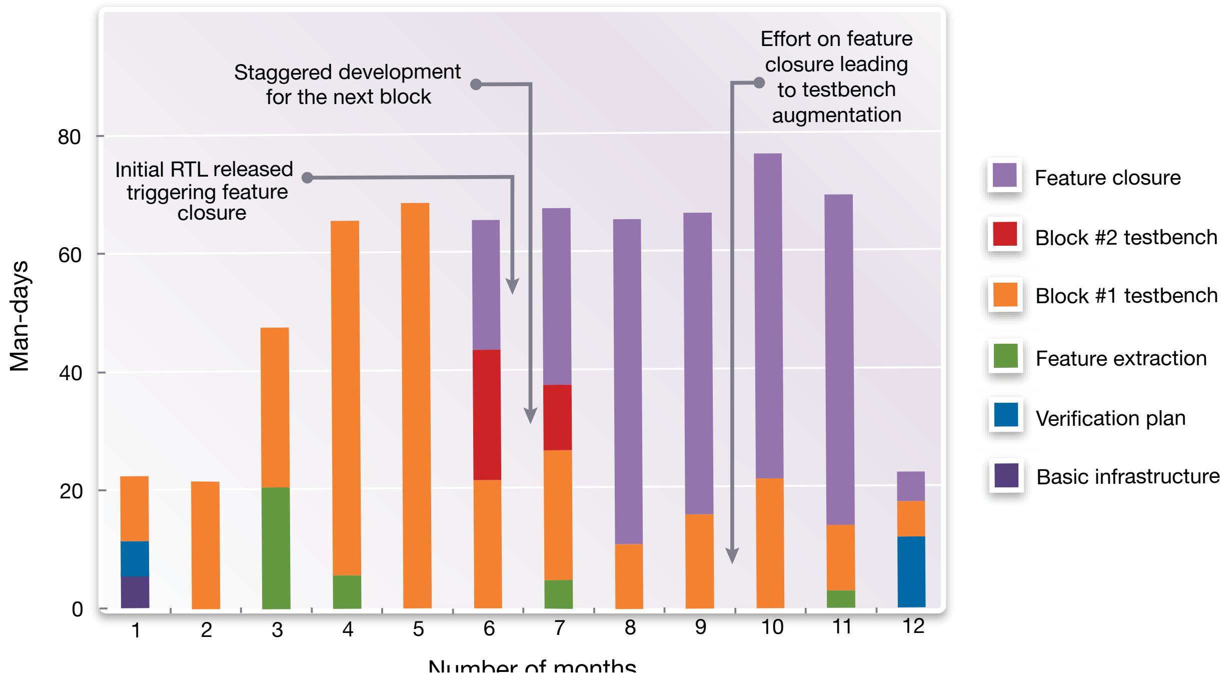

Figure 7: Distribution of effort during the verification phase

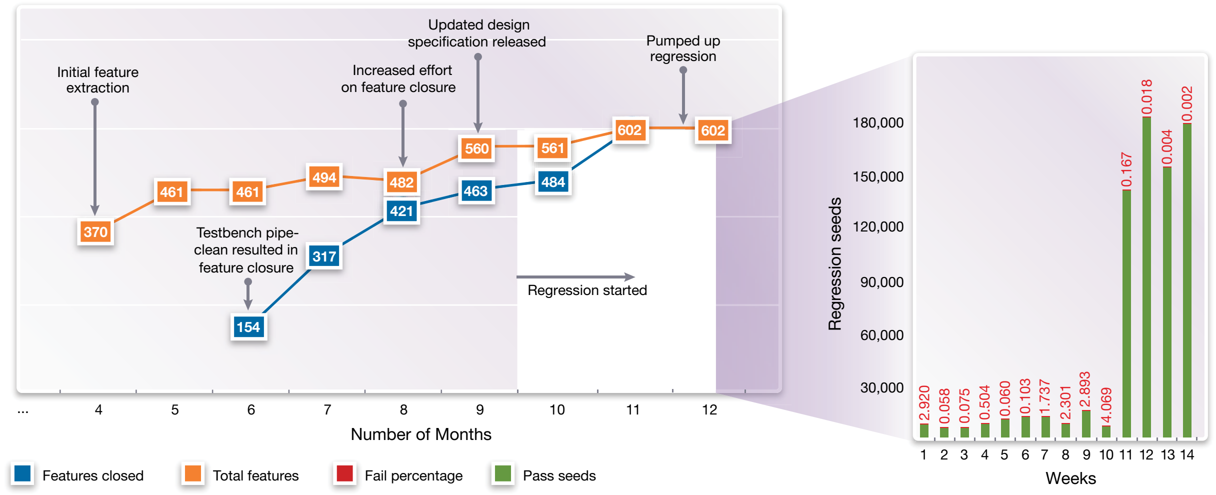

Figure 8: Rate of feature closure compared to total feature extraction and fail percentages versus pass seeds

The lower right insert in Figure 8 depicts the fails percentages of regressions run. During the initial eight to nine weeks of regression, the fail percentage varied between 0.06% and 4.06% with approximately nine thousand seeds. During the final four weeks, when the number of seeds were increased significantly, fail percentages decreased dramatically going down from 0.167% to 0.002% for one-hundred-eighty-thousand seeds.

Deliveries included:

- A fully UVM compliant testbench with approximately sixty random test cases
- One-hundred percent functional and assertion coverage and ninety-one percent line coverage with no exclusions
- Six-hundred and two features closed with 182,225 seeds passing
- Exhaustive verification plan including the testbench, test cases and explanation how to execute

Case Study 2

In this case the design was a custom interconnect with industry standard protocols P1, P2, P3 (names withdrawn for confidentiality) and Ethernet. It was a router with 6x9 bidirectional interface and configurable protocol support. The input protocol was mapped and routed to output protocol based on configuration, which led to a very large configuration space with multiple options including number of lanes, speeds, and data types, while maintaining bandwidth and throughput. The prime objective was to completely verify the ASIC with one-hundred percent coverage of all features, datapath and configuration combinations. A set of system scenario-like use cases was also created. The major verification phases are highlighted below.

Verification Planning

Evolving specifications with multiple interdependencies made planning, estimation and scheduling challenging. Addition of features and scenarios toward the end of the project led to testbench modification, new test creation and additional regression. The planning phase involved creating test and verification plans and a strategy to verify the design at the top-level instead of module-level. These were later updated with test case and transactor descriptions. Test plans were reviewed periodically.

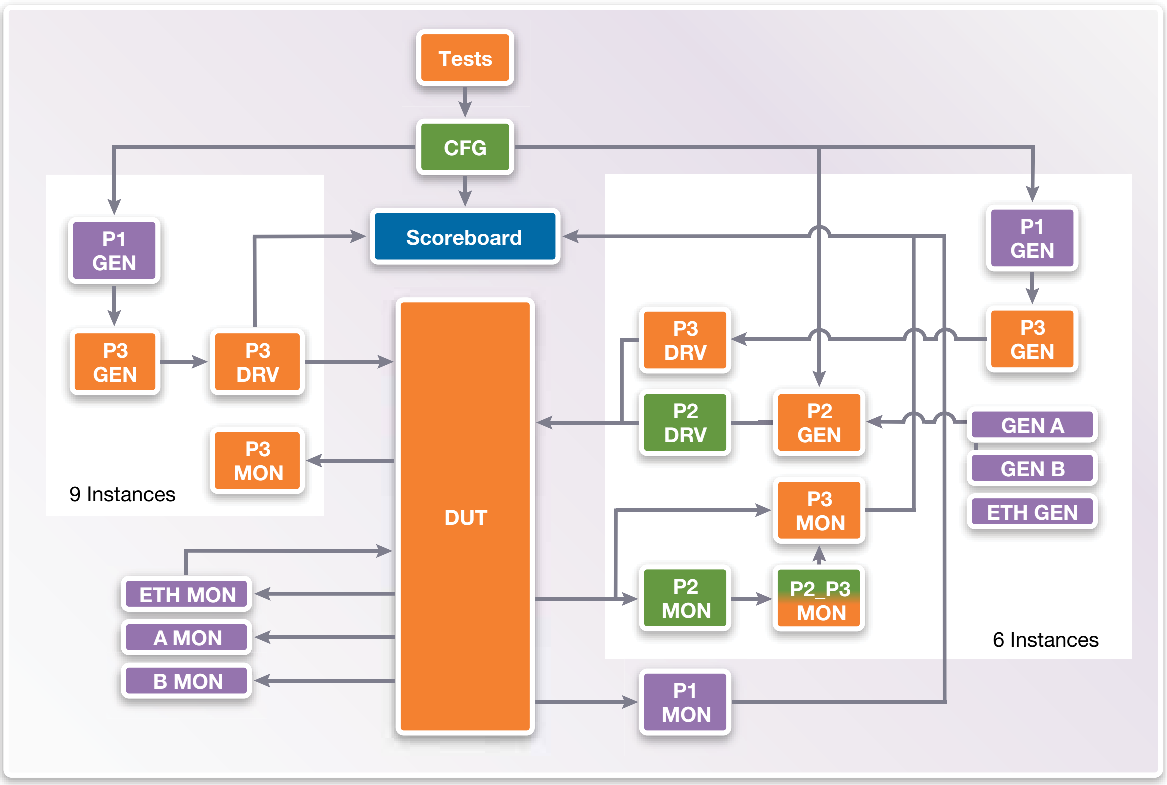

Figure 9: Testbench architecture showing driver and monitors

Testbench Setup

Except for the Ethernet Verification IP, almost all the components were created from scratch. This was critical as the protocols were adapted for the design. The testbench as depicted in Figure 9 replicated design behaviors for buffer logic, synchronization logic, data handling and ports. The testbench also handled multiple protocols from different ports which lead to multiple data streams into the scoreboard which supported automated comparison based on the packet headers. In this setup there are two sets of components, one with nine instances with P1 data mapped to P3 and sent through the P3 agent and the other with six instances with P3 data mapped to P2. The P2 and P3 agents were interlinked and converter logic added. Similar to the previous study, a single configuration class controlling the whole testbench was exercised by the test cases.

Constraint Random Stimulus and Tests

Configuration space was huge with too many interdependent variables. The objective was to maintain the total throughput of the design because of the design's buffer management scheme. Any input lane out of the six or nine lanes could be enabled or disabled, support any of the 3 protocols, run any of five different speeds, and carry any of seven different data types. The output lanes could also be similarly configured, but had to be constrained such that all the incoming data was exactly mapped.

As the specifications were derived from different standard protocols, the verification phase overlapped with the architecture and design phase. This helped the verification team study the actual protocols and understand the derivatives used in the project. Prior verification experience with similar designs enabled the verification team to contribute in pre-design review which helped in developing basic transactors well before the DUV was ready. This enabled us to pipe-clean the testbench with Empty DUV approach as depicted in Figure 4.

Development

In the development phase verification components including drivers, monitors, checkers and scoreboard were developed for different interfaces. Some components and codes were reused which were analyzed before their actual usage. All components were then integrated and connected to the actual design. Once the flow stabilized, complex directed test with multiple parameters were developed to verify the design. Issues found were resolved and re-verified. As the design matured, random testcases were developed along with functional coverage bins and regression suites. The random testbench was modified to support random testcases. Regressions were run with random as well as directed tests. This phase exposed some critical design issues which otherwise were tough to detect with directed testing. With every regression, the coverage numbers were captured and analyzed for design holes which led to more directed tests or more seeds in random tests.

Review of test cases and test scenario documentation improved the test scenarios. These tests were then added to the verification plan document which also included transaction description and testbench flow. Verification closure was achieved once the coverage goals converged and review of test and verification plans were completed.

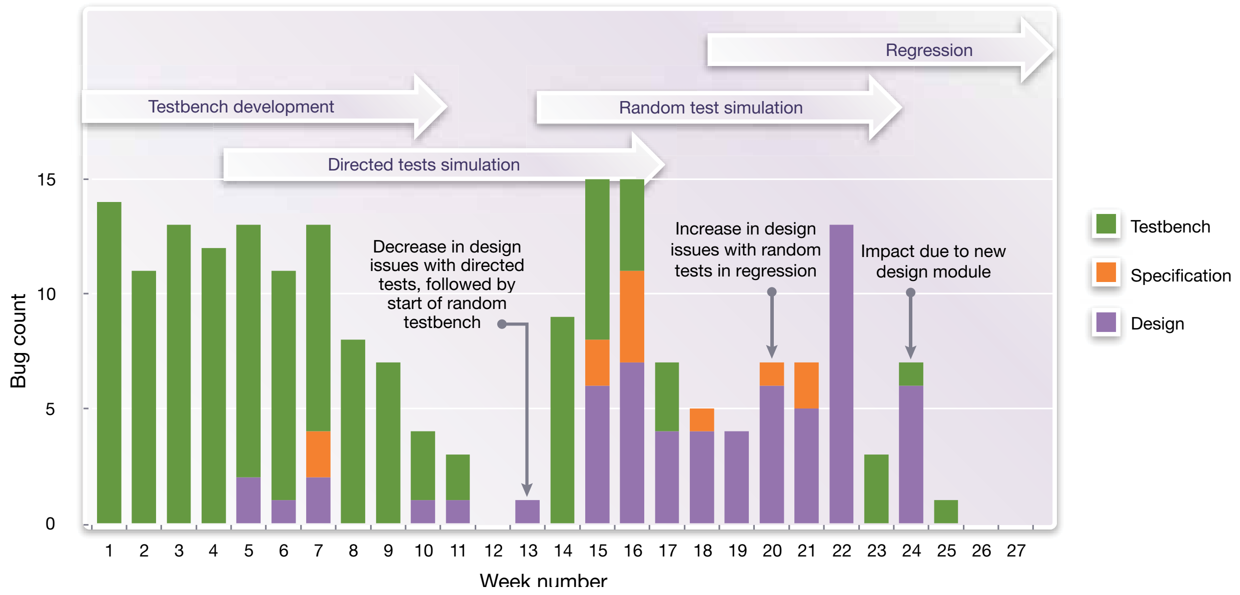

Figure 10: Bug counts throughout the project

From Figure 10, some important facets of verification became very clear:

- Initial testbench development had bugs that created design or specification issues
- Almost fifty percent of time was used in stabilizing the testbench before regressing with random stimulus
- Change in design leads to increased bug count
- Week fourteen marked the start of random test that identified lots of testbench issues which settled down gradually to expose more design issues
- The valleys (weeks twelve, eighteen and twenty-five) depict stable design and testbench

Deliverables included:

- Complete verification solution with more than five-hundred tests cases in four different testbenches
- Regression suite with more than fifteen-hundred different seeds
- Exhaustive verification plan including test case description
- Back-annotated test plan (verification planner)

Summary

As design features and complexity continue to rise, the need for robust functional verification has become increasingly important. Design knowledge coupled with verification methodology expertise is crucial to defining the optimal verification solution.

The key ingredients to creating an optimum verification solution are:

- Defining a verification strategy with adequate planning
- Reliable verification estimations and schedules
- Strong understanding of architecture and verification requirements
- Reuse of components
- Predicting and planning scalability
- Smart tracking of verification milestones from onset
- Setting up elaborate functional coverage hooks
- Smart usage of available tools and methodology

Synopsys has enabled the industry to bridge the gap between the desired and the actual verification implementation. VCS™ and its complimentary verification solutions provide a seamless foundation to simulate designs. UVM-SV with different applications and tools enables engineers to setup testbenches and perform verification for the most complex designs. The use of DesignWare™ IP including verification IP, significantly enhances design and verification productivity.

In addition to Synopsys tools and IP solutions, Synopsys has strong technical support and professional services teams. With presence in different geographies, Synopsys Professional Services has extensive experience in delivering methodology, design and verification solutions for today's cutting-edge designs. For more information on these services, please contact us or call your local Synopsys sales representative.